```
module AD670_controller(clk, status, dataavail, r_w_bar, state, reset, count, enable);

input clk;
input reset;
input status;

output dataavail, r_w_bar, enable;
output [3:0] state;
output [14:0] count;

//internal state
reg [3:0] state;
reg [3:0] nextstate;
reg status_d1, status_d2;
reg dataavail, r_w_bar, r_w_bar_int;

//Timing signal count registers
reg [14:0] count;
reg [3:0] r_w_bar_waitcount;
reg [3:0] read_delaycount;

//Timing signal enable registers
reg enable;
reg r_w_bar_waitenable;
reg read_delayenable;

//State Declarations
parameter IDLE = 0;
parameter CONV0 = 1;
parameter CONV1 = 2;
parameter CONV2 = 3;
parameter WAITSTATUS0 = 4;
parameter WAITSTATUS1 = 5;
parameter READDELAY0 = 6;
parameter READDELAY1 = 7;
parameter READCYCLE = 8;

// Max Count for sampling time
parameter MAX_COUNT = 12'd3150;

always @(posedge clk) // this generates a delay so we only sample 10,000 times/sec
	begin
		if(reset)
			begin
				count <= 0;
				enable <= 0;
			end
```

```
        else
            begin
                count <= 1 + count;
                    if (count == MAX_COUNT)
                        begin
                            enable <= 1;
                            count <= 0;
                        end
                    else
                        enable <= 0;
            end
    end

always @(posedge clk) //this is a counter for a wait delay for the rwbar signal
    begin
        if(reset)

            r_w_bar_waitcount <= 0;

        else
            if(r_w_bar_waitenable)

                r_w_bar_waitcount <= 1 + r_w_bar_waitcount;

            else

                r_w_bar_waitcount <= 0;
    end

always @(posedge clk) //this is a counter for the read wait delay
    begin
        if(reset)

            read_delaycount <= 0;

        else
            if(read_delayenable)

                read_delaycount <= 1 + read_delaycount;

            else

                read_delaycount <= 0;
    end

always @(posedge clk) // this is for next state assignment
    begin
        if(reset)
```

```
                state <= IDLE;
        else
                state <= nextstate;

        status_d1 <= status;
        status_d2 <= status_d1;
        r_w_bar <= r_w_bar_int;
    end

always @(state or status_d2 or enable)
    begin
        r_w_bar_int = 1;
        dataavail = 0;
        r_w_bar_waitenable = 0;
        read_delayenable = 0;

        case(state)

            IDLE: begin
                        if(enable)
                            nextstate = CONV0;
                        else
                            nextstate = IDLE;
                    end

            CONV0: begin
                        r_w_bar_int = 0;
                        r_w_bar_waitenable = 1;
                        nextstate = CONV1;
                    end

            CONV1: begin
                        r_w_bar_int = 0;
                        r_w_bar_waitenable = 1;
                        nextstate = CONV2;
                    end

            CONV2: begin
                        r_w_bar_int = 0;
                        r_w_bar_waitenable = 1;
                            if(r_w_bar_waitcount == 10) // to ensure 10 clk cycle wait before turning off rwbar
                                nextstate = WAITSTATUS0;
                            else
                                nextstate = CONV2;
                    end

            WAITSTATUS0:
                    begin
```

```
                    if(status_d2)
                        nextstate = WAITSTATUS0;
                    else
                        nextstate = WAITSTATUS1;
                end


            WAITSTATUS1:
                begin
                    if(!status_d2)
                        nextstate = READDELAY0;
                    else
                        nextstate = WAITSTATUS1;
                end

            READDELAY0:
                begin
                    nextstate = READDELAY1;
                    read_delayenable = 1;
                end

            READDELAY1:
                begin
                    read_delayenable = 1;
                    if(read_delaycount == 10)       //to ensure 10 clock cycle wait before data available
                        nextstate = READCYCLE;
                    else
                        nextstate = READDELAY1;
                end

            READCYCLE:
                begin
                    dataavail = 1;
                    nextstate = IDLE;
                end


            default: nextstate = IDLE;

        endcase
    end
endmodule
```

```
module AD670_controller_TB_v;

	// Inputs
	reg clk;
	reg status;
	reg reset;

	// Outputs
	wire dataavail;
	wire r_w_bar;
	wire [3:0] state;
	wire [14:0] count;
	wire enable;

	// Instantiate the Unit Under Test (UUT)
	AD670_controller uut (
		.clk(clk),
		.status(status),
		.dataavail(dataavail),
		.r_w_bar(r_w_bar),
		.state(state),
		.reset(reset),
		.count(count),
		.enable(enable)
	);

	always #37 clk=~clk;

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

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

		// Add stimulus here
```

```
    end

endmodule
```

```
module AD670_datareg(clk, reset, dataavail, AD670_data, data_in);

//The purpose of this module is to prevent data glitches from occuring
//the A/D chip will convert 1000 times per second. When the A/D gets
//a signal to start conversion the data output of the A/D will not be valid
//for many clock cycles. This module registers that output each time the A/D
//signals a completion of a conversion, thereby illiminating the possibility
//of data glitches.

input clk, dataavail, reset;
input [7:0] data_in;

output [7:0] AD670_data;

reg [7:0] AD670_data;

always @(posedge clk)
	begin
		if(reset)
			AD670_data <= 0;
		else
			if(dataavail)
				AD670_data <= data_in;
			else
				AD670_data <= AD670_data;
	end
endmodule
```

```
module AD670_datareg_TB_v;

	// Inputs
	reg clk;
	reg reset;
	reg dataavail;
	reg [7:0] data_in;

	// Outputs
	wire [7:0] AD670_data;

	// Instantiate the Unit Under Test (UUT)
	AD670_datareg uut (
		.clk(clk),
		.reset(reset),
		.dataavail(dataavail),
		.AD670_data(AD670_data),
		.data_in(data_in)
	);

	always #10 clk=~clk;
	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		dataavail = 0;
		data_in = 0;

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100;
		reset = 0;
		dataavail = 1;
		data_in = 10;
		#20;
		dataavail = 0;
		#100;
		dataavail = 1;
		data_in = 189;
		#20;
```

```
        dataavail = 0;


        // Add stimulus here

    end

endmodule
```

```
module FSM_FIR(clk, reset, AD670_data, AD670_filtered, state, FIR_enable,
                                    AD670_tempreg);

input clk, reset;

//Coefficients from the ROM
wire [7:0] h0, h1, h2, h3, h4, h5, h6, h7;

//following values for coefficients are from matlab
//FDAtool for lowpass filter
assign h0 = 8'h03;
assign h1 = 8'hEC;
assign h2 = 8'h29;
assign h3 = 8'h4E;
assign h4 = 8'h29;
assign h5 = 8'hEC;
assign h6 = 8'h03;
assign h7 = 8'h03;

//Data from the AD670
input [7:0] AD670_data;

//Filtered output and register to hold intermediate values
output [7:0] AD670_filtered;
output [3:0] state;
output FIR_enable;
output [16:0] AD670_tempreg;
reg [16:0] AD670_tempreg;
reg [7:0] AD670_filtered;

//Registers for the count and FIR compute-begin sequence
reg [14:0] count;
reg FIR_enable;

//Register for states
reg [3:0] state;
//parameter MAX_COUNT = 15'd31500;
  parameter MAX_COUNT = 50;
//Parameters for filter states
parameter IDLE = 0;
parameter SHIFT0 = 1;
parameter SHIFT1 = 2;
parameter SHIFT2 = 3;
parameter SHIFT3 = 4;
parameter SHIFT4 = 5;
parameter SHIFT5 = 6;
parameter SHIFT6 = 7;
```

```
parameter SHIFT7 = 8;
parameter FINAL0 = 9;

// this generates a timing signal so we only retrieve samples
// or begin an FIR filter computation 1000 times/sec
always @(posedge clk)
	begin
		if(reset)
			begin
				count <= 0;
				FIR_enable <= 0;
			end
		else
			begin
				count <= 1 + count;
					if (count == MAX_COUNT)
						begin
							FIR_enable <= 1;
							count <= 0;
						end
					else
						FIR_enable <= 0;
			end
	end

always @(posedge clk)
	begin
		if(reset)
			begin
				AD670_tempreg <= 0;
				AD670_filtered <= 0;
				state <= IDLE;
			end
		else case(state)

			IDLE:	begin
					AD670_tempreg <= 0;
					if(FIR_enable)
						state <= SHIFT0;
					else
						state <= IDLE;
				end

			SHIFT0: begin
					AD670_tempreg <= (AD670_data*h0);
					state <= SHIFT1;
				end
```

```
            SHIFT1: begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h1);
                        state <= SHIFT2;
                    end

            SHIFT2: begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h2);
                        state <= SHIFT3;
                    end

            SHIFT3:     begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h3);
                        state <= SHIFT4;
                    end

            SHIFT4:     begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h4);
                        state <= SHIFT5;
                    end

            SHIFT5: begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h5);
                        state <= SHIFT6;
                    end

            SHIFT6:     begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h6);
                        state <= SHIFT7;
                    end

            SHIFT7:     begin
                        AD670_tempreg <= AD670_tempreg + (AD670_data*h7);
                        state <= FINAL0;
                    end

            FINAL0: begin
                        AD670_filtered <= (AD670_tempreg[16:9] >> 2) + (AD670_tempreg[16:9] >> 1);
                        state <= IDLE;
                    end


            default:    state <= IDLE;

        endcase
    end

endmodule
```

```
module FSM_FIR_TB_v;

	// Inputs
	reg clk;
	reg reset;
	reg [7:0] AD670_data;

	// Outputs
	wire [7:0] AD670_filtered;
	wire [3:0] state;
	wire FIR_enable;
	wire [16:0] AD670_tempreg;

	// Instantiate the Unit Under Test (UUT)
	FSM_FIR uut (
		.clk(clk),
		.reset(reset),
		.AD670_data(AD670_data),
		.AD670_filtered(AD670_filtered),
		.state(state),
		.FIR_enable(FIR_enable),
		.AD670_tempreg(AD670_tempreg)
	);

	always #10 clk=~clk;
	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		AD670_data = 0;

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100
		reset = 0;
		AD670_data = 2;


		// Add stimulus here
```

```
    end

endmodule
```

```
module FSM_Integrator_Accel(clk, reset, AD670_data, direct_a, accel, BIAS);

input clk, reset;
input [7:0] AD670_data; //this is AD670 data latched to the data_reg 1000 times / sec
input [7:0] BIAS; //this is a voltage BIAS parameter to control gravity bias, 0 movement.

output direct_a;
output [7:0] accel;

reg direct_a; // this is a flag for pos/neg direction on the axis. 1 for pos, 0 for neg.
reg [7:0] accel;


always @(posedge clk)
	begin
		if(reset)
			begin
				direct_a <= 1;
				accel <= 0;
			end
		else
		if(AD670_data > BIAS)
			begin
				direct_a <= 1;
				accel <= (AD670_data - BIAS);
			end
		else
			begin
				direct_a <= 0;
				accel <= (BIAS - AD670_data);
			end
	end

endmodule
```

```
module FSM_Integrator_Accel_TB_v;

	// Inputs
	reg clk;
	reg reset;
	reg [7:0] AD670_data;
	reg [7:0] BIAS;

	// Outputs
	wire direct;
	wire [7:0] accel;

	// Instantiate the Unit Under Test (UUT)
	FSM_Integrator_Accel uut (
		.clk(clk),
		.reset(reset),
		.AD670_data(AD670_data),
		.direct(direct),
		.accel(accel),
		.BIAS(BIAS)
	);

	always #10 clk=~clk;

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		AD670_data = 0;
		BIAS = 0;

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100;
		reset = 0;
		BIAS = 168;
		AD670_data = 150;
		#100;
		AD670_data = 180;
		#100;
```

```
        BIAS = 190;
        #100;
        AD670_data = 210;


        // Add stimulus here

    end

endmodule
```

```
module Integrator_Accel(clk, reset, direct_a, direct_v, accel, vel);

input clk, reset, direct_a;
input [7:0] accel;

output direct_v; // up if 1, down if 0
output [14:0] vel;

reg direct_v;
reg [14:0] vel;

//Stuff for the counter
reg [18:0] count;
reg enable;
parameter MAX_COUNT = 19'd315000;

always @(posedge clk) // this generates a timing signal so we only integrate 100 times/sec
	begin
		if(reset)
			begin
				count <= 0;
				enable <= 0;
			end
		else
			begin
				count <= 1 + count;
					if (count == MAX_COUNT)
						begin
							enable <= 1;
							count <= 0;
						end
					else
						enable <= 0;
			end
	end

always @(posedge clk)
	begin
		if(reset)
			begin
				direct_v <= 1;
				vel <= 0;
```

```
        end
    else
    if(enable)
        begin
            if(direct_a)
                begin
                    if(((vel + accel) < 15'd16383) && direct_v && (vel > accel))
                        begin
                            direct_v <= 1;
                            vel <= vel + accel;
                        end
                    else
                    if(((vel + accel) < 15'd16383) && (vel > accel))
                        begin
                            vel <= vel - accel;
                            direct_v <= 0;
                        end
                    else
                    if((vel + accel) < 15'd16383)
                        begin
                            vel <= vel + accel;
                            direct_v <= 1;
                        end
                    else
                        begin
                            vel <= vel;
                            direct_v <= 1;
                        end
                end
            else
                begin
                    if((vel > accel) && !direct_v && ((vel + accel) < 15'd16383))
                        begin
                            direct_v <= 0;
                            vel <= vel + accel;
                        end
                    else
                    if(((vel + accel) < 15'd16383) && (vel > accel))
                        begin
                            direct_v <= 1;
                            vel <= vel - accel;
                        end
                    else
```

```
                        if(((vel + accel) < 15'd16383))
                            begin
                                direct_v <= 0;
                                vel <= vel + accel;
                            end
                        else
                            begin
                                vel <= vel;
                                direct_v <= 0;
                            end
                    end
                end
            end

endmodule
```

```
module Integrator_Accel_FSM_v;

	// Inputs
	reg clk;
	reg reset;
	reg direct;
	reg [7:0] accel;

	// Outputs
	wire [13:0] vel;

	// Instantiate the Unit Under Test (UUT)
	Integrator_Accel uut (
		.clk(clk),
		.reset(reset),
		.direct(direct),
		.accel(accel),
		.vel(vel)
	);

	always #10 clk=~clk;

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

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100;
		reset = 0;
		direct = 1;
		accel = 150;

		// Add stimulus here

	end
```

```
endmodule
```

```
module Integrator_Accel_TB_v;

	// Inputs
	reg clk;
	reg reset;
	reg direct_a;
	reg [7:0] accel;

	// Outputs
	wire direct_v;
	wire [14:0] vel;

	// Instantiate the Unit Under Test (UUT)
	Integrator_Accel uut (
		.clk(clk),
		.reset(reset),
		.direct_a(direct_a),
		.direct_v(direct_v),
		.accel(accel),
		.vel(vel)
	);

	always #10 clk=~clk;

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

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100;
		reset = 0;
		accel = 80;
		direct_a = 1;
		#40000;
		direct_a = 0;
```

```
        // Add stimulus here

    end

endmodule
```

```
module Integrator_Position(clk, reset, reset_position, direct_a, vel, axis_constant, position, vel_shift);

input clk, reset, reset_position;
input direct_a;
//input [14:0] vel;
input [7:0] vel; //for testing

output [9:0] position;
//output [6:0] vel_shift;
output [7:0] vel_shift;

//This is a bias value which we can use
//at instantiation to specify which axis'
//position value the module will compute.
//X is variable between 0 to 639 while
//Y is variable between 0 479.
input [9:0] axis_constant;

//position register
reg [9:0] position;

//This is the shifted version of the velocity
//used for position computation
//reg [6:0] vel_shift;
reg [7:0] vel_shift; //for testing

//Stuff for the counter
reg [18:0] count;
reg enable;
parameter MAX_COUNT = 19'd315000;

always @(posedge clk) // this generates a timing signal so we only integrate 100 times/sec
	begin
		if(reset)
			begin
				count <= 0;
				enable <= 0;
			end
		else
			begin
				count <= 1 + count;
					if (count == MAX_COUNT)
```

```
                        begin
                            enable <= 1;
                            count <= 0;
                        end
                    else
                        enable <= 0;
                end
        end



always @(posedge clk)
        begin
//          vel_shift <= {3'b00, vel[14:11]};       // vel_shift is first 7 MSB's of velocity
            vel_shift <= {4'b00, vel[3:0]};
            if(reset || reset_position)
                begin
                    vel_shift <= 0;
                    position <= 0;
                end
            else
            if(enable)
                begin
                    if(direct_a)
                        begin
                            if(position < axis_constant)
                                position <= vel_shift + position;
                            else
                                position <= axis_constant;
                        end
                    else
                        begin
                            if(position > vel_shift)
                                position <= position - vel_shift;
                            else
                                position <= 0;
                        end
                end
        end

endmodule
```

```
module Integrator_Position_TB_v;

	// Inputs
	reg clk;
	reg reset;
	reg reset_position;
	reg direct;
	reg [13:0] vel;
	reg [9:0] axis_constant;

	// Outputs
	wire [9:0] position;

	// Instantiate the Unit Under Test (UUT)
	Integrator_Position uut (
		.clk(clk),
		.reset(reset),
		.reset_position(reset_position),
		.direct(direct),
		.vel(vel),
		.axis_constant(axis_constant),
		.position(position)
	);

	always #10 clk=~clk;
	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		reset_position = 0;
		direct = 0;
		vel = 0;
		axis_constant = 0;

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		#100;
		reset = 0;
		direct = 1;
		axis_constant = 639;
		vel = 9684;
```

```
        reset = 0;


        // Add stimulus here

    end

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   19:51:01 05/07/2006
// Design Name:   charblock
// Module Name:   charblock_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: charblock
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module charblock_tb_v;

  // Inputs
  reg clk;
  reg en;
  reg [34:0] newblock;
  reg [9:0]  px;
  reg [9:0]  py;

  // Outputs
  wire       on;
  wire [9:0] idx;
  wire [9:0] opxo;

  // Instantiate the Unit Under Test (UUT)
  charblock2 uut (
                .clk(clk),
                .en(en),
```

```
        .newblock(newblock),
        .px(px),
        .py(py),
        .opx(10'd10),
        .on(on),
        .idx(idx),
        .opxo(opxo)
        );

initial begin
  // Initialize Inputs
  clk = 1;
  en = 0;
  newblock = 0;
  px = 0;
  py = 0;

  // Wait 100 ns for global reset to finish
  #100;
  newblock = 35'b01110100011001110101110011000101110;
  en = 1;
  #10;
  en = 0;

  // shouldn't show up, since py too low
  #10;
  px = 9;
  #10;
  px = 10;
  #10;
  px = 11;

  #10;
  py = 472;
  #10; px = 9;
  #10; px = 10;
  #10; px = 11;
  #10; px = 12;
  #10; px = 13;
  #10; px = 14;
  #10; px = 15;
  #10; px = 16;
```

```
    py = 473;
    #10; px = 9;
    #10; px = 10;
    #10; px = 11;
    #10; px = 12;
    #10; px = 13;
    #10; px = 14;
    #10; px = 15;
    #10; px = 16;
  end

  always #5 clk = ~clk;
endmodule
```

```
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

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

`timescale 1ns/1ps

module dmem(
	addra,
	addrb,
```

```
        clka,
        clkb,
        dina,
        douta,
        doutb,
        wea);


input [13 : 0] addra;
input [13 : 0] addrb;
input clka;
input clkb;
input [31 : 0] dina;
output [31 : 0] douta;
output [31 : 0] doutb;
input wea;

// synopsys translate_off

    BLKMEMDP_V6_1 #(
            14,     // c_addra_width
            14,     // c_addrb_width
            "0",    // c_default_data
            9400,   // c_depth_a
            9400,   // c_depth_b
            0,      // c_enable_rlocs
            1,      // c_has_default_data
            1,      // c_has_dina
            0,      // c_has_dinb
            1,      // c_has_douta
            1,      // c_has_doutb
            0,      // c_has_ena
            0,      // c_has_enb
            0,      // c_has_limit_data_pitch
            0,      // c_has_nda
            0,      // c_has_ndb
            0,      // c_has_rdya
            0,      // c_has_rdyb
            0,      // c_has_rfda
            0,      // c_has_rfdb
            0,      // c_has_sinita
            0,      // c_has_sinitb
            1,      // c_has_wea
```

```
        0,      // c_has_web
        18,     // c_limit_data_pitch
        "mif_file_16_1",        // c_mem_init_file
        0,      // c_pipe_stages_a
        0,      // c_pipe_stages_b
        0,      // c_reg_inputsa
        0,      // c_reg_inputsb
        "0",    // c_sinita_value
        "0",    // c_sinitb_value
        32,     // c_width_a
        32,     // c_width_b
        0,      // c_write_modea
        0,      // c_write_modeb
        "0",    // c_ybottom_addr
        1,      // c_yclka_is_rising
        1,      // c_yclkb_is_rising
        1,      // c_yena_is_high
        1,      // c_yenb_is_high
        "hierarchy1",   // c_yhierarchy
        0,      // c_ymake_bmm
        "16kx1",        // c_yprimitive_type
        1,      // c_ysinita_is_high
        1,      // c_ysinitb_is_high
        "1024", // c_ytop_addr
        0,      // c_yuse_single_primitive
        1,      // c_ywea_is_high
        1,      // c_yweb_is_high
        1)      // c_yydisable_warnings
    inst (
        .ADDRA(addra),
        .ADDRB(addrb),
        .CLKA(clka),
        .CLKB(clkb),
        .DINA(dina),
        .DOUTA(douta),
        .DOUTB(doutb),
        .WEA(wea),
        .DINB(),
        .ENA(),
        .ENB(),
        .NDA(),
        .NDB(),
        .RFDA(),
```

```
        .RFDB(),
        .RDYA(),
        .RDYB(),
        .SINITA(),
        .SINITB(),
        .WEB());


// synopsys translate_on

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

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

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   00:05:21 05/11/2006
// Design Name:   drawshape
// Module Name:   drawshape_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: drawshape
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module drawshape_tb_v;

  // Inputs
  reg clk;
  reg reset;
  reg [3:0] ssel;
  reg       lock;
  reg [9:0] sx;
  reg [9:0] sy;
  reg [9:0] ex;
  reg [9:0] ey;
  reg en;

  wire [9:0] px;
  wire [9:0] py;
  wire rden;
  sync S(clk, reset, hsync, vsync, sync_b, blank_b, px, py, rden);
```

```
// Outputs
wire     pon;
wire [13:0] idx;
wire [31:0] data;
wire wren;

// Instantiate the Unit Under Test (UUT)
drawshape uut (
            .clk(clk),
            .reset(reset),
            .rden(rden),
            .en(en),
            .ssel(ssel),
            .lock(lock),
            .sx(sx),
            .sy(sy),
            .ex(ex),
            .ey(ey),
            .px(px),
            .py(py),
            .pon(pon),
            .idx(idx),
            .data(data),
            .wren(wren)
            );

initial begin
  // Initialize Inputs
  clk = 1;
  reset = 0;
  ssel = 3;
  lock = 0;
  sx = 0;
  sy = 0;
  ex = 0;
  ey = 0;
  en = 1;

  // Wait 100 ns for global reset to finish
  #30;

  // Add stimulus here
  reset = 1;
```

```
    ex = 100;
    ey = 100;
  end

  always #1 clk=~clk;
endmodule
```

```
module drawutil(clk, reset, rden, px, py, cx, cy, curcmd, cstate, urgb);
  input clk;
  input reset;
  input rden;
  input [9:0] px;
  input [9:0] py;
  input [9:0] cx;
  input [9:0] cy;
  input [5:0] curcmd;
  input [5:0] cstate;
  output [23:0] urgb;

  reg [2:0] rtmode;
  reg resetting;
  reg updating;

  wire [25:0] pto;
  reg [25:0] pte;

  reg [5:0] rtidx;
  wire [4:0] cbidx;
  wire [5:0] cbaddr;
  charsel csel(clk, rtidx, cx, cy, mode, cbidx, cbaddr, curcmd, cstate);

  reg [5:0] addr;
  wire [5:0] maddr;
  assign maddr = resetting?addr:cbaddr;
  wire [34:0] cblock;
  ascii5x7 char_rom(maddr, clk, cblock);

  // X/Y position
  charblock pt0(clk, pte[0], cblock, px, py, 10'd6, pto[0]);     //P
  charblock pt1(clk, pte[1], cblock, px, py, 10'd12, pto[1]);    //O
  charblock pt2(clk, pte[2], cblock, px, py, 10'd18, pto[2]);    //S
  charblock pt3(clk, pte[3], cblock, px, py, 10'd24, pto[3]);    //=
  charblock pt4(clk, pte[4], cblock, px, py, 10'd30, pto[4]);    //<
  charblock pt5(clk, pte[5], cblock, px, py, 10'd36, pto[5]);    //x0
  charblock pt6(clk, pte[6], cblock, px, py, 10'd42, pto[6]);    //x1
  charblock pt7(clk, pte[7], cblock, px, py, 10'd48, pto[7]);    //x2
  charblock pt8(clk, pte[8], cblock, px, py, 10'd56, pto[8]);    //,
  charblock pt9(clk, pte[9], cblock, px, py, 10'd62, pto[9]);    //y0
  charblock pt10(clk, pte[10], cblock, px, py, 10'd68, pto[10]); //y1
```

```
charblock pt11(clk, pte[11], cblock, px, py, 10'd74, pto[11]); //y2
charblock pt12(clk, pte[12], cblock, px, py, 10'd80, pto[12]); //>

// Mode
charblock pt13(clk, pte[13], cblock, px, py, 10'd100, pto[13]); //C
charblock pt14(clk, pte[14], cblock, px, py, 10'd106, pto[14]); //C
charblock pt15(clk, pte[15], cblock, px, py, 10'd112, pto[15]); //=
charblock pt16(clk, pte[16], cblock, px, py, 10'd118, pto[16]); //cc0
charblock pt17(clk, pte[17], cblock, px, py, 10'd124, pto[17]); //cc1
charblock pt18(clk, pte[18], cblock, px, py, 10'd130, pto[18]);
charblock pt19(clk, pte[19], cblock, px, py, 10'd150, pto[19]); //C
charblock pt20(clk, pte[20], cblock, px, py, 10'd156, pto[20]); //S
charblock pt21(clk, pte[21], cblock, px, py, 10'd162, pto[21]); //=
charblock pt22(clk, pte[22], cblock, px, py, 10'd168, pto[22]); //cs0
charblock pt23(clk, pte[23], cblock, px, py, 10'd174, pto[23]); //cs1
charblock pt24(clk, pte[24], cblock, px, py, 10'd180, pto[24]);

parameter bgcolor = 24'hFFFFFF;
parameter poscolor = 24'hFF0000;
parameter cccolor = 24'h0000FF;
parameter cscolor = 24'h00FF00;

assign urgb = (|pto[12:0])?poscolor:
              ((|pto[17:13])?cccolor:
               ((|pto[23:19])?cscolor:
                bgcolor));

always @(posedge clk) begin
  if(!reset) begin
    resetting <= 1;
    rtidx <= 0;
    rtmode <= 0;
  end else if(resetting) begin
    case(rtmode)
      0: begin
        pte <= 13'b0;
        rtmode <= (rtidx<25)?1:4;
      end
      1: begin
        case(rtidx)
          0: addr <= 25;
          1: addr <= 24;
          2: addr <= 28;
```

```
          3: addr <= 41;
          4: addr <= 39;
          8: addr <= 42;
          12: addr <= 40;
          13: addr <= 12;
          14: addr <= 12;
          15: addr <= 41;
          19: addr <= 12;
          20: addr <= 28;
          21: addr <= 41;
          default: addr <= 63;
        endcase
        rtmode <= 2;
      end
      2: rtmode <= 3;
      3: begin
        pte[rtidx] <= 1;
        rtidx <= rtidx + 1;
        rtmode <= 0;
      end
      4: begin
        resetting <= 0;
      end
    endcase
  end else if(rden) begin
    updating <= 1;
    rtidx <= 0;
  end else if(updating) begin
    rtidx <= rtidx + 1;
    if(rtidx==48)
      updating <= 0;

    if(rtidx & 6'h01) // Enable write on odd rtidx
      pte[cbidx] <= 1;
    else // Clear writes and set next values on even rtidx
      pte <= 25'b0;
  end
 end
endmodule

module charsel(clk, rtidx, cx, cy, mode, cbidx, caddr, curcmd, cstate);
 input clk;
 input [5:0] rtidx;
```

```
input [9:0] cx;
input [9:0] cy;
input [4:0] mode;
output [4:0] cbidx;
output [5:0] caddr;
input [5:0] curcmd;
input [5:0] cstate;

wire [4:0] hrtidx;
assign hrtidx = rtidx[5:1];

reg[4:0] cbidx;
reg [5:0] caddr;
always @(posedge clk) begin
  if(!(rtidx & 6'h01)) begin // If it's odd...
    if(hrtidx < 10)
      case(hrtidx)
        0: begin
          cbidx <= 5;
          caddr <= cx[9:8];
        end
        1: begin
          cbidx <= 6;
          caddr <= cx[7:4];
        end
        2: begin
          cbidx <= 7;
          caddr <= cx[3:0];
        end
        3: begin
          cbidx <= 9;
          caddr <= cy[9:8];
        end
        4: begin
          cbidx <= 10;
          caddr <= cy[7:4];
        end
        5: begin
          cbidx <= 11;
          caddr <= cy[3:0];
        end
        6: begin
          cbidx <= 16;
```

```
          caddr <= curcmd[5:4];
        end
        7: begin
          cbidx <= 17;
          caddr <= curcmd[3:0];
        end
        8: begin
          cbidx <= 22;
          caddr <= cstate[5:4];
        end
        9: begin
          cbidx <= 23;
          caddr <= cstate[3:0];
        end
      endcase // case(hrtidx)
    end
  end
endmodule

module charblock(clk, en, newblock, px, py, opx, on);
  input clk;
  input en;
  input [34:0] newblock;
  input [9:0] px;
  input [9:0] py;
  input [9:0] opx;
  output on;

  reg on;
  reg [34:0] cblock;
  always @(posedge clk) begin
    if(en)
      cblock <= newblock;

    if((px >= opx && px < opx+5) &&
       (py >= 472 && py < 479))
      on <= cblock[34-5*(py-472)-(px-opx)];
    else
      on <= 0;
  end
endmodule // charblock
```

```
module mdetect(clk, reset, dirx, diry, vx, vy, en, mnum, aen);
  input clk;
  input reset;
  input dirx;
  input diry;
  input [6:0] vx;
  input [6:0] vy;
  input en;
  output [4:0] mnum;
  output [5:0] aen;

  reg [37:0] mrom [31:0];

  reg [5:0] mnum;
  reg [31:0] mindiff;
  reg [31:0] curdiff;
  reg aen;
  reg [4:0] idx;

  parameter maxidx = 7;
  parameter maxmin = 20;
  reg [2:0] state;

  wire [37:0] cur;
  assign cur = mrom[idx];

  always @(posedge clk) begin
    if(!reset) begin
      /////////   DX    DY    VX    VY
      mrom[0] <= {1'h0, 1'h0, 7'h00, 7'h0F,
                  4'h0, 4'hF, 4'hA, 4'h5, 6'h00};
      mrom[1] <= {1'h1, 1'h0, 7'h0F, 7'h0F,
                  4'hF, 4'hF, 4'h8, 4'h8, 6'h01};
      mrom[2] <= {1'h1, 1'h0, 7'h0F, 7'h0F,
                  4'hF, 4'h0, 4'h5, 4'hA, 6'h01};
      mrom[3] <= {1'h1, 1'h1, 7'h0F, 7'h0F,
                  4'hF, 4'hF, 4'h8, 4'h8, 6'h01};
      mrom[4] <= {1'h0, 1'h1, 7'h00, 7'h0F,
                  4'h0, 4'hF, 4'hA, 4'h5, 6'h04};
      mrom[5] <= {1'h0, 1'h1, 7'h0F, 7'h0F,
                  4'hF, 4'hF, 4'h8, 4'h8, 6'h01};
      mrom[6] <= {1'h0, 1'h0, 7'h00, 7'h0F,
                  4'hF, 4'h0, 4'h5, 4'hA, 6'h06};
      mrom[7] <= {1'h0, 1'h0, 7'h0F, 7'h0F,
                  4'hF, 4'hF, 4'h8, 4'h8, 6'h01};
      state <= 0;
    end else if(en && state==0) begin
      state <= 1;
      idx <= 0;
      mindiff <= ~32'h0;
```

```
   end else begin
     case(state)
       0: begin
         aen <= 0;
       end
       1: begin
         curdiff <= (cur[37]-dirx)*cur[21] + (cur[36]-diry)*cur[20] + (cur[35:29]-vx)*cur[19:13] + (cur[28:22]-vy)*cur[12:6];
         state <= 1;
       end
       2: begin
         if(curdiff < mindiff) begin
           mindiff <= curdiff;
           mnum <= cur[5:0];
         end
         idx <= idx + 1;
         state <= (idx==maxidx)?3:1;
       end
       3: begin
         if(mindiff < maxmin)
           aen <= 1;
         state <= 0;
       end
     endcase
   end
 end
endmodule

module adetect(clk, reset, en, mnum, cmd, cmden);
 input clk;
 input reset;
 input en;
 input [4:0] mnum;
 output [5:0] cmd;
 output cmden;

 reg [5:0] cmd;
 reg cmden;

 reg [2:0] state;
 parameter maxidx = 31;
 reg [19:0] crom [31:0];

 reg [4:0] idx;
 wire [19:0] cur;
 assign cur = crom[idx];

 always @(posedge clk) begin
   if(!reset) begin
     state <= 0;
   end else if(en && state==0) begin
     state <= 1;
```

```
    idx <= 0;
   end
  end
endmodule
```

```
module drawover(clk, reset, rden, ovmode, ec, full, ovgo, ovbusy, px, py, pon,
                sx, sy, ex, ey,
                ridx, wbdata, wbwren, dout,
                shidx, shdata, shwren,
                ovstate, state1);
  input clk;
  input reset;
  input rden;
  input [2:0] ovmode;
  input ec;
  input full;
  input ovgo;
  output ovbusy;
  input [9:0] px;
  input [9:0] py;
  output pon;
  input [9:0] sx;
  input [9:0] sy;
  input [9:0] ex;
  input [9:0] ey;
  output [13:0] ridx;
  input [31:0] wbdata;
  output wbwren;
  output [31:0] dout;
  input [13:0] shidx;
  input [31:0] shdata;
  input shwren;

  output [2:0] ovstate;
  output [2:0] state1;

  reg ovbusy;
  wire wbwren;
  wire [31:0] ovdout;
  wire [31:0] dout;
  wire pon;

  wire [13:0] ridx;
  wire [31:0] rdin;
  wire rwe;

  memdisp ovbuf(clk, reset, rden, ridx, rdin, rwe, ovdout,
```

```
            px, py, pon);

reg go1, go2;
wire busy1, busy2;

reg shcp;
wire [13:0] idx1;
wire [13:0] idx2;
wire [31:0] din2;
wire ovwren;

assign ridx = shcp?shidx:(busy1?idx1:idx2);
assign rdin = shcp?shdata:din2;
assign rwe = shcp?shwren:ovwren;

wire [2:0] state1;
copy_to_wb Ov2Wbc(clk, reset, go1, busy1, idx1, ovdout, wbdata, dout, wbwren,
                  sx, sy, ex, ey, ec, full, state1);

copy_from_wb Wb2Ovc(clk, reset, go2, busy2, idx2, wbdata, din2, ovwren,
                    sx, sy, ex, ey, full);

reg [2:0] state;

parameter Idle = 0;
parameter Sh2Ov = 1;
parameter Ov2Wb = 2;
parameter Wb2Ov = 3;

always @(posedge clk) begin
  if(!reset) begin
    state <= 0;
    ovbusy <= 0;
    shcp <= 0;
    go1 <= 0;
    go2 <= 0;
  end else if(ovgo && !ovbusy) begin
    ovbusy <= 1;
    state <= 0;
  end else if(ovbusy) begin
    case(ovmode)
      Sh2Ov: begin
        case(state)
```

```
    0: begin // Set shcp
      shcp <= 1;
      state <= 1;
    end
    1: begin // Wait for !go
      if(!ovgo) begin
        shcp <= 0;
        ovbusy <= 0;
      end
    end
  endcase
end // Sh2Ov
Ov2Wb: begin
  case(state)
    0: begin // Start copy
      go1 <= 1;
      state <= 1;
    end
    1: begin // Wait for busy
      if(busy1) begin
        go1 <= 0;
        state <= 2;
      end
    end
    2: begin // Wait for !busy
      if(!busy1)
        ovbusy <= 0;
    end
  endcase
end // Ov2Wb
Wb2Ov: begin
  case(state)
    0: begin // Start copy
      go2 <= 1;
      state <= 1;
    end
    1: begin // Wait for busy
      if(busy2) begin
        go2 <= 0;
        state <= 2;
      end
    end
    2: begin // Wait for !busy
```

```
            if(!busy2)
              ovbusy <= 0;
           end
         endcase
       end // Wb2Ov
     endcase // case(ovmode)
   end
  end
endmodule // drawover

module copy_to_wb(clk, reset, go, busy, idx, ovdata, wbin, wbout, wbwr,
                  sx, sy, ex, ey, ec, full, state, cx, cy);
  input clk;
  input reset;
  input go;
  output busy;
  output [13:0] idx;
  input [31:0] ovdata;
  input [31:0] wbin;
  output [31:0] wbout;
  output wbwr;
  input [9:0] sx;
  input [9:0] sy;
  input [9:0] ex;
  input [9:0] ey;
  input ec; // 1 = erase, 0 = copy;
  input full;
  output [2:0] state;
  output [9:0] cx;
  output [9:0] cy;

  reg busy;
  reg [13:0] idx;
  reg [31:0] wbout;
  reg wbwr;

  parameter S_idle = 0;
  parameter S_read1 = 1;
  parameter S_read2 = 2;
  parameter S_write1 = 3;
  parameter S_write2 = 4;
  parameter S_write3 = 5;
  parameter S_done = 6;
```

```
parameter maxidx = 9399;

reg [9:0] cx;
reg [9:0] cy;
parameter LJB = 32'h80000000;

reg [2:0] state;
always @(posedge clk) begin
  if(!reset) begin
    idx <= 0;
    wbout <= 0;
    state <= S_idle;
  end else begin
    case(state)
      S_idle: begin
        if(go) begin
          busy <= 1;
          state <= S_read1;
          idx <= 0;
          cx <= 0;
          cy <= 0;
        end
      end
      S_read1: begin
        state <= S_read2;
      end
      S_read2: begin
        if(full)
          wbout <= ec?32'h0:(ovdata | wbin);
        else if(cy >= sy && cy <= ey) begin
          if(sx >= cx && sx <= cx+32) begin
            if(ex >= cx && ex <= cx+32) begin // both in
              wbout <= (((LJB >>> sx[4:0]) |
                        ~(LJB >>> ex[4:0])) & wbin) |
                       ((~(LJB >>> sx[4:0]) &
                         (LJB >>> ex[4:0])) & (ec?32'h0:ovdata));
            end else begin // start in
              wbout <= (~(LJB >>> sx[4:0]) & wbin) |
                       ((LJB >>> sx[4:0]) & (ec?32'h0:ovdata));
            end
          end else if(ex >= cx && ex <= cx+32) begin // end in
            wbout <= (~(LJB >>> ex[4:0]) & wbin) |
```

```
                ((LJB >>> ex[4:0]) & (ec?32'h0:ovdata));
          end
        end else
          wbout <= wbin;
        state <= S_write1;
      end
      S_write1: begin
        wbwr <= 1;
        state <= S_write2;
      end
      S_write2: begin
        wbwr <= 0;
        state <= S_write3;
      end
      S_write3: begin
        idx <= idx + 1;
        if(cx < 600) begin
          cx <= cx + 32;
        end else begin
          cx <= 0;
          cy <= cy + 1;
        end
        state <= (idx==maxidx)?S_done:S_read1;
      end
      S_done: begin
        state <= S_idle;
        busy <= 0;
      end
    endcase
  end
end
endmodule

module copy_from_wb(clk, reset, go, busy, idx, wbout, ovin, ovwr,
                    sx, sy, ex, ey, full);
  input clk;
  input reset;
  input go;
  output busy;
  output [13:0] idx;
  input [31:0] wbout;
  output [31:0] ovin;
  output ovwr;
```

```
input [9:0] sx;
input [9:0] sy;
input [9:0] ex;
input [9:0] ey;
input full; // 1 = copy all, 0 = copy region

reg busy;
reg [13:0] idx;
reg [31:0] ovin;
reg ovwr;

parameter S_idle = 0;
parameter S_read1 = 1;
parameter S_read2 = 2;
parameter S_write1 = 3;
parameter S_write2 = 4;
parameter S_write3 = 5;
parameter S_done = 6;

parameter maxidx = 9399;

reg [9:0] cx;
reg [9:0] cy;

reg [31:0] mask;
reg [2:0] state;
parameter LJB = 32'h80000000;

always @(posedge clk) begin
  if(!reset) begin
    idx <= 0;
    ovin <= 0;
    ovwr <= 0;
    state <= S_idle;
  end else begin
    case(state)
      S_idle: begin
        ovwr <= 0;
        if(go) begin
          busy <= 1;
          state <= S_read1;
          idx <= 0;
          cx <= 0;
```

```
    cy <= 0;
  end
end
S_read1: begin
  state <= S_read2;
end
S_read2: begin
  if(full)
    ovin <= wbout;
  else if(cy >= sy && cy <= ey) begin
    if(sx >= cx && sx <= cx+32) begin
      if(ex >= cx && ex <= cx+32) begin // both in
        ovin <= ~(LJB >>> sx[4:0]) &
                (LJB >>> ex[4:0]) &
                wbout;
      end else begin // start in
        ovin <= ~(LJB >>> sx[4:0]) &
                wbout;
      end
    end else if(ex >= cx && ex <= cx+32) begin // end in
      ovin <= (LJB >>> ex[4:0]) &
              wbout;
    end
  end else
    ovin <= 0;
  state <= S_write1;
end
S_write1: begin
  ovwr <= 1;
  state <= S_write2;
end
S_write2: begin
  ovwr <= 0;
  state <= S_write3;
end
S_write3: begin
  idx <= idx + 1;
  if(cx < 600) begin
    cx <= cx + 32;
  end else begin
    cx <= 0;
    cy <= cy + 1;
  end
```

```
        state <= (idx==maxidx)?S_done:S_read1;
      end
      S_done: begin
        state <= S_idle;
        busy <= 0;
      end
    endcase
  end
end
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:56:51 05/15/2006
// Design Name:   drawover
// Module Name:   overlay_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: drawover
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module overlay_tb_v;

  // Inputs
  reg clk;
  reg reset;
  reg [2:0] mode;
  reg       ec;
  reg       ovgo;
  reg [9:0] sx;
  reg [9:0] sy;
  reg [9:0] ex;
  reg [9:0] ey;
  reg [31:0] wbdata;
  reg [13:0] shidx;
  reg [31:0] shdata;
  reg        shwren;

  wire [9:0] px;
```

```
wire [9:0] py;
wire rden;
sync S(clk, reset, hsync, vsync, sync_b, blank_b, px, py, rden);

// Outputs
wire      ovbusy;
wire      pon;
wire [13:0] ridx;
wire       wbwren;
wire [31:0] dout;

// Instantiate the Unit Under Test (UUT)
drawover uut (
            .clk(clk),
            .reset(reset),
            .rden(rden),
            .mode(mode),
            .ec(ec),
            .ovgo(ovgo),
            .ovbusy(ovbusy),
            .px(px),
            .py(py),
            .pon(pon),
            .sx(sx),
            .sy(sy),
            .ex(ex),
            .ey(ey),
            .ridx(ridx),
            .wbdata(wbdata),
            .wbwren(wbwren),
            .dout(dout),
            .shidx(shidx),
            .shdata(shdata),
            .shwren(shwren)
            );

wire [13:0] idx;
wire [31:0] data;
wire wren;
wire on;
wire inbounds;

drawshape uut2 (
```

```
		.clk(clk),
		.reset(reset),
		.rden(rden),
		.en(en),
		.ssel(ssel),
		.lock(lock),
		.sx(sx),
		.sy(sy),
		.ex(ex),
		.ey(ey),
		.px(px),
		.py(py),
		.pon(),
		.idx(idx),
		.data(data),
		.wren(wren),
		.on(on),
		.inbounds(inbounds)
		);

initial begin
  // Initialize Inputs
  clk = 1;
  reset = 0;
  mode = 0;
  ec = 0;
  ovgo = 0;
  sx = 0;
  sy = 0;
  ex = 0;
  ey = 0;
  wbdata = 0;
  shidx = 0;
  shdata = 0;
  shwren = 0;

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

  // Add stimulus here

end
```

```
  always #1 clk=~clk;

endmodule
```

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

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

`timescale 1ns/1ps

module ovmem(
    addra,
    addrb,
```

```
        clka,
        clkb,
        dina,
        douta,
        doutb,
        wea);


input [13 : 0] addra;
input [13 : 0] addrb;
input clka;
input clkb;
input [31 : 0] dina;
output [31 : 0] douta;
output [31 : 0] doutb;
input wea;

// synopsys translate_off

    BLKMEMDP_V6_1 #(
            14,     // c_addra_width
            14,     // c_addrb_width
            "0",    // c_default_data
            9400,   // c_depth_a
            9400,   // c_depth_b
            0,      // c_enable_rlocs
            1,      // c_has_default_data
            1,      // c_has_dina
            0,      // c_has_dinb
            1,      // c_has_douta
            1,      // c_has_doutb
            0,      // c_has_ena
            0,      // c_has_enb
            0,      // c_has_limit_data_pitch
            0,      // c_has_nda
            0,      // c_has_ndb
            0,      // c_has_rdya
            0,      // c_has_rdyb
            0,      // c_has_rfda
            0,      // c_has_rfdb
            0,      // c_has_sinita
            0,      // c_has_sinitb
            1,      // c_has_wea
```

```
        0,      // c_has_web
        18,     // c_limit_data_pitch
        "mif_file_16_1",        // c_mem_init_file
        0,      // c_pipe_stages_a
        0,      // c_pipe_stages_b
        0,      // c_reg_inputsa
        0,      // c_reg_inputsb
        "0",    // c_sinita_value
        "0",    // c_sinitb_value
        32,     // c_width_a
        32,     // c_width_b
        0,      // c_write_modea
        0,      // c_write_modeb
        "0",    // c_ybottom_addr
        1,      // c_yclka_is_rising
        1,      // c_yclkb_is_rising
        1,      // c_yena_is_high
        1,      // c_yenb_is_high
        "hierarchy1",   // c_yhierarchy
        0,      // c_ymake_bmm
        "16kx1",        // c_yprimitive_type
        1,      // c_ysinita_is_high
        1,      // c_ysinitb_is_high
        "1024", // c_ytop_addr
        0,      // c_yuse_single_primitive
        1,      // c_ywea_is_high
        1,      // c_yweb_is_high
        1)      // c_yydisable_warnings
    inst (
        .ADDRA(addra),
        .ADDRB(addrb),
        .CLKA(clka),
        .CLKB(clkb),
        .DINA(dina),
        .DOUTA(douta),
        .DOUTB(doutb),
        .WEA(wea),
        .DINB(),
        .ENA(),
        .ENB(),
        .NDA(),
        .NDB(),
        .RFDA(),
```

```
        .RFDB(),
        .RDYA(),
        .RDYB(),
        .SINITA(),
        .SINITB(),
        .WEB());


// synopsys translate_on

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

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

endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   13:24:06 05/15/2006
// Design Name:   pulseinvert
// Module Name:   pulseinvert_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: pulseinvert
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module pulseinvert_tb_v;

	// Inputs
	reg clk;
	reg reset;
	reg in;

	// Outputs
	wire out;

	// Instantiate the Unit Under Test (UUT)
	pulseinvert uut (
		.clk(clk),
		.reset(reset),
		.in(in),
		.out(out)
	);
```

```
    initial begin
        // Initialize Inputs
        clk = 1;
        reset = 0;
        in = 0;

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

        // Add stimulus here
    end

    always #5 clk = ~clk;

    always #50 in = ~in;

endmodule
```

```
module drawshape(clk, reset, rden, en, ssel, slock, sx, sy, ex, ey, px, py, pon,
                 idx, data, wren);

  input clk;
  input reset;
  input rden;
  input en;
  input [3:0] ssel;
  input slock;
  input [9:0] sx;
  input [9:0] sy;
  input [9:0] ex;
  input [9:0] ey;
  input [9:0] px;
  input [9:0] py;
  output pon;
  output [13:0] idx;
  output [31:0] data;
  output wren;

  reg [9:0] csx;
  reg [9:0] csy;
  reg [9:0] cex;
  reg [9:0] cey;
  reg [3:0] cssel;

  parameter blank = 0;
  parameter line = 1;
  parameter square = 2;
  parameter fsquare = 3;
  parameter circle = 4;
  parameter fcircle = 5;

  reg on;
  always @(posedge clk) begin
    if(!reset) begin
      cssel <= 0;
      csx <= 0;
      csy <= 0;
      cex <= 0;
      cey <= 0;
    end else if(!slock) begin
```

```
    cssel <= ssel;
    if(sx <= ex) begin
      csx <= sx;
      cex <= ex;
    end else begin
      csx <= ex;
      cex <= sx;
    end

    if(sy <= ey) begin
      csy <= sy;
      cey <= ey;
    end else begin
      csy <= ey;
      cey <= sy;
    end
  end
end

wire [9:0] dx;
wire [9:0] dy;
assign dx = (cex >= csx)?(cex - csx):(csx-cex);
assign dy = (cey >= csy)?(cey - csy):(csy-cey);

wire [9:0] dpx;
wire [9:0] dpy;
assign dpx = (px >= sx)?(px - sx):(sx - px);
assign dpy = (py >= sy)?(py - sy):(sy - py);

wire [9:0] rad;
wire [20:0] rad2;
assign rad = ((dx <= dy)?dx:dy)>>1;
assign rad2 = rad*rad;

wire [9:0] centx;
wire [9:0] centy;
assign centx = csx + rad;
assign centy = csy + rad;

wire [20:0] pixdist2;
assign pixdist2 = ((px-centx)*(px-centx) + (py-centy)*(py-centy));

wire [19:0] tmp1;
```

```
  wire [19:0] tmp2;
  assign tmp1 = dpx*dy;
  assign tmp2 = dpy*dx;
  wire [19:0] diff;
  assign diff = (tmp1 >= tmp2)?(tmp1-tmp2):(tmp2-tmp1);

  wire [19:0] thresh;
  assign thresh = (dx+dy);

  always @(cssel or csx or csy or cex or cey) begin
    case(cssel)
      blank: on = 0;
      line: on = (diff < thresh);
      square: on = (px==csx || px==cex || py==csy || py==cey);
      fsquare: on = 1;
      circle: on = (pixdist2 < rad2 && pixdist2 > rad2-(rad<<1));
      fcircle: on = (pixdist2 < rad2);
      default: on = 0;
    endcase
  end

  wire inbounds;
  assign inbounds = (px >= csx && py >= csy && px <= cex && py <= cey);

  wire pon;
  assign pon = inbounds & on & en;

  wire [13:0] idx;
  wire [31:0] data;
  wire wren;
  pixtoram ptr(clk, reset, rden, px, py, pon, idx, data, wren);
endmodule // drawshape

module pixtoram(clk, reset, rden, px, py, pon, idx, data, wren);
  input clk;
  input reset;
  input rden;
  input [9:0] px;
  input [9:0] py;
  input pon;
  output [13:0] idx;
  output [31:0] data;
  output wren;
```

```
reg [31:0] curbuf [1:0];
reg bufsel;

reg [13:0] idx;
reg [31:0] data;
reg wren;

parameter S_waitrden = 0;
parameter S_waity = 1;
parameter S_waitx = 2;
parameter S_norm0 = 3;
parameter S_norm1 = 4;
parameter S_norm2 = 5;

reg [2:0] state;

always @(posedge clk) begin
 if(!reset) begin
  state <= S_waitrden;
 end else begin
  case(state)
   S_waitrden: begin
    idx <= 0;
    bufsel <= 0;
    state <= rden?S_waity:S_waitrden;
   end
   S_waity: begin // Wait for vblank to end
    if(px==0 && py==0) begin
     state <= S_norm0;
     curbuf[bufsel] <= (pon?(32'h80000000):32'h0);
    end
   end
   S_waitx: begin
    if(px==0) begin
     state <= S_norm0;
     curbuf[bufsel] <= (pon?(32'h80000000):32'h0);
    end
   end
   S_norm0: begin // Start write, switch bufsel
    if(px[4:0]==5'h00) begin
     state <= S_norm1;
     wren <= 1;
```

```
        data <= curbuf[bufsel];
        bufsel <= ~bufsel;
        curbuf[~bufsel] <= (pon?(32'h80000000 >> px[4:0]):32'h0);
      end else
        curbuf[bufsel] <= curbuf[bufsel] | (pon?(32'h80000000 >> px[4:0]):32'h0);
    end
    S_norm1: begin
      wren <= 0;
      curbuf[bufsel] <= curbuf[bufsel] | (pon?(32'h80000000 >> px[4:0]):32'h0);
      state <= S_norm2;
    end
    S_norm2: begin
      curbuf[bufsel] <= curbuf[bufsel] | (pon?(32'h80000000 >> px[4:0]):32'h0);
      idx <= idx + 1;
      state <= (px<630)?
            S_norm0:
            ((py<469)?S_waitx:S_waitrden);
    end
   endcase
  end
 end
endmodule // pixtoram
```

```
`timescale 1ns / 1ps
module sync(clk, reset, hsync, vsync, sync_b, blank_b, x, y, rden);

  input clk; // 31.5 MHz pixel clock
  input reset; // system reset
  output hsync; // horizontal sync
  output vsync; // vertical sync
  output sync_b; // hardwired to Vdd
  output blank_b; // composite blank
  output [9:0] x; // number of the current pixel
  output [9:0] y; // number of the current line
  output rden; // Signal to modules set high near start of vblank period
  // 640x480 75Hz parameters
  parameter    PIXELS = 800;
  parameter    LINES = 525;
  parameter    HACTIVE_VIDEO = 640;
  parameter    HFRONT_PORCH = 16;
  parameter    HSYNC_PERIOD = 96;
  parameter    HBACK_PORCH = 48;
  parameter    VACTIVE_VIDEO = 480;
  parameter    VFRONT_PORCH = 11;
  parameter    VSYNC_PERIOD = 2;
  parameter    VBACK_PORCH = 32;
  // current pixel count
  reg [9:0]    x = 10'b0;
  reg [9:0]    y = 10'b0;

  // registered outputs
  reg rden;
  reg hsync = 1'b1;
  reg vsync = 1'b1;
  reg blank_b = 1'b1;
  wire sync_b; // Connected to Vdd

  wire clk;
  wire [9:0] next_x;
  wire [9:0] next_y;
  always @ (posedge clk)
    begin
      if (!reset)
        begin
          x <= 10'b0;
          y <= 10'b0;
```

```
        hsync <= 1'b1;
        vsync <= 1'b1;
        blank_b <= 1'b1;
        rden <= 0;
      end
    else
      begin
        x <= next_x;
        y <= next_y;
        rden <= ((next_x==0) && (next_y==VACTIVE_VIDEO+VFRONT_PORCH+VSYNC_PERIOD));
        hsync <=
              (next_x < HACTIVE_VIDEO + HFRONT_PORCH) |
              (next_x >= HACTIVE_VIDEO+HFRONT_PORCH+
               HSYNC_PERIOD);
        vsync <=
              (next_y < VACTIVE_VIDEO+VFRONT_PORCH) |
              (next_y >= VACTIVE_VIDEO+VFRONT_PORCH+
               VSYNC_PERIOD);
        blank_b <=
                (next_x < HACTIVE_VIDEO) &
                (next_y < VACTIVE_VIDEO);
      end
   end
  // next state is computed with combinational logic
  assign next_x = (x == PIXELS-1) ?
                       10'h000 : x + 1'b1;
  assign next_y = (x == PIXELS-1) ?
                      (y == LINES-1) ? 10'h000 : y + 1'b1 :
              y;
  assign sync_b = 1'b1;
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   21:50:57 05/08/2006
// Design Name:   sync
// Module Name:   sync_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: sync
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module sync_tb_v;

	// Inputs
	reg clk;
	reg reset;

	// Outputs
	wire hsync;
	wire vsync;
	wire sync_b;
	wire blank_b;
	wire [9:0] x;
	wire [9:0] y;
	wire rden;

	// Instantiate the Unit Under Test (UUT)
	sync uut (
		.clk(clk),
```

```
        .reset(reset),
        .hsync(hsync),
        .vsync(vsync),
        .sync_b(sync_b),
        .blank_b(blank_b),
        .x(x),
        .y(y),
        .rden(rden)
    );

    initial begin
        // Initialize Inputs
        clk = 1;
        reset = 0;
        // Wait 100 ns for global reset to finish
        #100;
  reset = 1;
        // Add stimulus here

    end
 always #1 clk = ~clk;
endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:07:34 05/15/2006
// Design Name:   whiteboard
// Module Name:   wb_test.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: whiteboard
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module wb_test_v;

	// Inputs
	reg clk;
	reg reset;
	reg [9:0] curx;
	reg [9:0] cury;
	reg [5:0] cmd;
	reg cmden;

  wire [9:0] px;
  wire [9:0] py;
  wire rden;
  sync S(clk, reset, hsync, vsync, sync_b, blank_b, px, py, rden);

	// Outputs
	wire [23:0] rgb;
	wire [7:0] led;
```

```
	wire [13:0] shidx;
	wire [31:0] shdout;
	wire shwren;

	// Instantiate the Unit Under Test (UUT)
	whiteboard uut (
		.clk(clk),
		.reset(reset),
		.rden(rden),
		.px(px),
		.py(py),
		.curx(curx),
		.cury(cury),
		.cmd(cmd),
		.cmden(cmden),
		.rgb(rgb),
		.led(led),
		.shidx(shidx),
		.shdout(shdout),
		.shwren(shwren)
	);

  initial begin
    // Initialize Inputs
    clk = 1;
    reset = 0;
    curx = 0;
    cury = 0;
    cmd = 0;
    cmden = 0;

    // Wait 100 ns for global reset to finish

  end

  always #1 clk=~clk;

endmodule
```

```
module whiteboard(clk, reset, rden, px, py, curx, cury, cmd, cmden, rgb, led,
                  shidx, shdout, shwren);
  input clk;
  input reset;
  input rden;
  input [9:0] px;
  input [9:0] py;
  input [9:0] curx;
  input [9:0] cury;
  input [5:0] cmd;
  input cmden;
  output [23:0] rgb;
  output [7:0] led;
  output [13:0] shidx;
  output [31:0] shdout;
  output shwren;

  ////////// Drawing //////////
  reg [23:0] rgb;

  wire ccpon; // Current cursor
  wire wbpon; // Whiteboard
  wire shpon; // Shape
  wire mqpon; // Marquee
  wire ovpon; // Overlay

  reg oven;

  parameter cccolor = 24'hFF0000;
  parameter wbcolor = 24'hFFFFFF;
  parameter shcolor = 24'h00FF00;
  parameter mqcolor = 24'h00CCCC;
  parameter ovcolor = 24'hAAAAFF;

  wire [23:0] urgb; // Utility bar

  assign ccpon = (px==curx && py==cury);

  always @(posedge clk) begin
    if(py < 470) begin
      if(ccpon)
        rgb <= cccolor;
```

```
    else if(mqpon)
      rgb <= mqcolor;
    else if(shpon)
      rgb <= shcolor;
    else if(wbpon)
      rgb <=  wbcolor;
    else if(oven & ovpon)
      rgb <= ovcolor;
    else
      rgb <= 0;
  end else
    rgb <= urgb;
end
////////// End drawing //////////

// Some cases are substantially the same, so we use same code
// to handle them, but differentiate with these flags

reg [2:0] flags; // 2 = Done, 1 = Cancel, 0 = Select
reg drawmode; // 0 = Erase, 1 = Draw
reg [3:0] ssel; // Shape select (See shapes.v)

reg dsen;
wire [13:0] shidx;
wire [31:0] shdout;
wire shwren;

wire [13:0] wbidx;
wire [31:0] wbdin;
wire wbwren;
wire [31:0] wbdout;

reg [9:0] mx; // Mark coordinates
reg [9:0] my;
reg enmarquee; // Enable marquee

reg [2:0] ovmode;
reg ec;
reg full;
reg ovgo;
wire ovbusy;

reg [5:0] curcmd; // Current command being handled; only makes sense C_* > 6
```

```
reg [5:0] cstate; // Command state

wire mgo;
wire mbusy;
wire [2:0] ovstate;
wire [2:0] state1;
drawover dov(clk, reset, rden, ovmode, ec, full, ovgo, ovbusy, px, py, ovpon,
        mx, my, curx, cury, wbidx, wbdout, wbwren, wbdin,
        shidx, shdout, shwren, ovstate, state1);
wire [7:0] led;
assign led = {2'h3, ~ovstate, ~state1};
drawshape ds(clk, reset, rden, dsen, ssel, 1'b0, mx, my, curx, cury, px, py, shpon,
        shidx, shdout, shwren);
memdisp wb(clk, reset, rden, wbidx, wbdin, wbwren, wbdout, px, py, wbpon);
drawmarquee dm(clk, px, py, mx, my, curx, cury, enmarquee, mqpon);
drawutil du(clk, reset, rden, px, py, curx, cury, curcmd, cstate, urgb);

parameter C_Null = 0;
parameter C_Done = 1;
parameter C_Cancel = 2;
parameter C_Select = 3;
parameter C_Clear = 4;
parameter C_Cut = 5;
parameter C_Copy = 6;
parameter C_Paste = 7;
parameter C_Erase = 8;
parameter C_DrawFree = 9;
parameter C_DrawLine = 10;
parameter C_DrawRect = 11;
parameter C_DrawFRect = 12;
parameter C_DrawCirc = 13;
parameter C_DrawFCirc = 14;

parameter S_Done = ~6'b0;

always @(posedge clk) begin
  if(!reset) begin
    curcmd <= C_Null;
    flags <= 0;
    ssel <= 0;
  end else if(cstate == S_Done) begin
    curcmd <= C_Null;
    flags <= 0;
```

```
  end else if(cmd != C_Null && cmden) begin // <0> Getting input?
    if(cmd < C_Select) // <1,2> Done/Cancel are special
      flags[3-cmd] <= 1;
    else if(curcmd == C_Null) begin // Are we bored?
      if(cmd == C_Select) begin // <3> Select is special, too!
        flags[0] <= 1;
        curcmd <= C_Select;
      end else if(cmd >= C_DrawLine) begin // <10+>
        curcmd <= C_DrawLine;
        ssel <= cmd - C_DrawFree;
      end else if(cmd <= C_Paste) // <4,5,6,7>
        curcmd <= cmd;
      else begin // <8,9> OK, must be Erase or DrawFree command
        drawmode <= cmd - C_Erase;
        curcmd <= C_Erase;
        ssel <= 1;
      end
    end
  end
end

reg [15:0] freehandcount;
reg resetting;
always @(posedge clk) begin
  if(!reset) begin
    cstate <= 0;
    mx <= 0;
    my <= 0;
    dsen <= 0;
    oven <= 0;
    resetting <= 1;
  end else if(resetting) begin
    case(cstate)
      0: begin
        full <= 1;
        ec <= 1;
        ovmode <= 2;
        ovgo <= 1;
        cstate <= 1;
      end
      1: begin
        if(ovbusy) begin
          ovgo <= 0;
```

```
        cstate <= 2;
      end
    end
    2: begin
      if(!ovbusy) begin
        cstate <= S_Done;
        resetting <= 0;
      end
    end
  endcase
end else if(curcmd == C_Null)
  cstate <= 0;
else if(curcmd >= 3) begin
  case(curcmd)
    C_Select: begin
      case(cstate)
        0: begin // Set mark, set marquee on
          mx <= curx;
          my <= cury;
          enmarquee <= 1;
          cstate <= S_Done;
        end
        // Select is a special case, as it will be completed by
        // Clear, Cut or Copy below
      endcase
    end
    C_Clear: begin
      case(cstate)
        0: begin // Start overlay to wb copy
          if(flags[0])
            cstate <= S_Done;
          else begin
            ovmode <= 2;
            ec <= 1; // erase
            full <= 0; // only erase region
            ovgo <= 1;
            cstate <= 1;
          end
        end
        1: begin // Wait for overlay busy
          if(ovbusy) begin
            cstate <= 2;
            ovgo <= 0;
```

```
      end
    end
    2: begin
      if(!ovbusy) begin
        enmarquee <= 0;
        cstate <= S_Done;
      end
    end
  endcase
end
C_Cut: begin
  case(cstate)
    0: begin // Start wb to overlay copy
      if(flags[0])
        cstate <= S_Done;
      else begin
        ovmode <= 3;
        ec <= 0;
        full <= 0; // operate on region
        ovgo <= 1;
        cstate <= 1;
      end
    end
    1: begin // Wait for overlay busy
      if(ovbusy) begin
        cstate <= 2;
        ovgo <= 0;
      end
    end
    2: begin
      if(!ovbusy) begin // start ov to wb region copy
        ovmode <= 2;
        ec <= 1; // write blanks for cut
        cstate <= 3;
        ovgo <= 1;
      end
    end
    3: begin // wait for ov busy
      if(ovbusy) begin
        cstate <= 4;
        ovgo <= 0;
      end
    end
```

```
    4: begin
      if(!ovbusy) begin
        enmarquee <= 0;
        cstate <= S_Done;
      end
    end
  endcase
end
C_Copy: begin
  case(cstate)
    0: begin // Start overlay to wb copy
      if(flags[0])
        cstate <= S_Done;
      else begin
        ovmode <= 3;
        ec <= 0; // erase
        full <= 0; // only erase region
        ovgo <= 1;
        cstate <= 1;
      end
    end
    1: begin // Wait for overlay busy
      if(ovbusy) begin
        cstate <= 2;
        ovgo <= 0;
      end
    end
    2: begin
      if(!ovbusy) begin
        enmarquee <= 0;
        cstate <= S_Done;
      end
    end
  endcase
end
C_Paste: begin
  case(cstate)
    0: begin
      oven <= 1;
      ec <= 0;
      full <= 1;
      cstate <= 1;
      ovgo <= 1;
```

```
      ovmode <= 2;
    end
    1: begin
      if(ovbusy) begin
        cstate <= 2;
        ovgo <= 0;
      end
    end
    2: begin
      if(!ovbusy) begin
        cstate <= S_Done;
        oven <= 0;
      end
    end
  endcase
end
C_Erase: begin // Also handles DrawFree
  case(cstate)
    0: begin // Init vars
      freehandcount <= 0;
      cstate <= 1;
      mx <= curx;
      my <= cury;
    end
    1: begin // Check for end
      if(flags[1]||flags[2])
        cstate <= S_Done;
      else begin
        cstate <= 2;
        dsen <= 1;
        ovgo <= 1;
        ovmode <= 1;
      end
    end
    2: begin
      if(freehandcount==16'h1FFF) begin
        freehandcount <= 0;
        cstate <= 3;
      end else
        freehandcount <= freehandcount + 1;
    end
    3: begin // Draw
      ovgo <= 0;
```

```
      ec <= 0;
      full <= 1;
      cstate <= 4;
    end
    4: begin
      dsen <= 0;
      ovmode <= 2;
      ovgo <= 1;
      cstate <= 5;
    end
    5: begin
      mx <= curx;
      my <= cury;
      if(ovbusy) begin
        cstate <= 6;
        ovgo <= 0;
      end
    end
    6: begin
      if(!ovbusy)
        cstate <= 1;
    end
  endcase
end
C_DrawLine: begin // Also handles Draw[Rect, FRect, Circ, FCirc]
  case(cstate)
    0: begin // Set mark, enable shapes
      mx <= curx;
      my <= cury;
      dsen <= 1;
      cstate <= 1;
      ovgo <= 1;
      ovmode <= 1;
    end
    1: begin // Wait for Done or cancel
      if(flags[2] || flags[1])
        cstate <= 2;
    end
    2: begin // Stop shape to overlay copy
      ovgo <= 0;
      ec <= 0; // write w/ overlay data
      full <= 1; // write entire overlay
      cstate <= 3;
```

```
          end
          3: begin // Start overlay to wb copy
            dsen <= 0;
            ovmode <= 2;
            ovgo <= 1;
            cstate <= 4;
          end
          4: begin // Wait for overlay busy
            if(ovbusy) begin
              cstate <= 5;
              ovgo <= 0;
            end
          end
          5: begin
            if(!ovbusy) begin
              cstate <= S_Done;
            end
          end
        endcase
      end
      // TODO - add default case for error detection?
    endcase // case(curcmd)
  end
end

endmodule

module drawmarquee(clk, px, py, mx, my, curx, cury, mqen, mqon);
  input clk;
  input [9:0] px;
  input [9:0] py;
  input [9:0] mx;
  input [9:0] my;
  input [9:0] curx;
  input [9:0] cury;
  input mqen;
  output mqon;

  reg [9:0] csx;
  reg [9:0] csy;
  reg [9:0] cex;
  reg [9:0] cey;
```

```
  always @(posedge clk) begin
    // This way, the rest of code can depend on invariant
    // that sx <= ex, sy <= ey
    if(mx <= curx) begin
      csx <= mx;
      cex <= curx;
    end else begin
      csx <= curx;
      cex <= mx;
    end

    if(my <= cury) begin
      csy <= my;
      cey <= cury;
    end else begin
      csy <= cury;
      cey <= my;
    end
  end

  wire mqon;
  assign mqon = (px==csx || px==cex || py==csy || py==cey) && // Box
                (px >= csx && py >= csy && px <= cex && py <= cey) && // Bounded
                (px[0]^py[0]) && // "Dashed"
                mqen; // Enabled
endmodule
```

```
module memdisp(clk, reset, rden, idx0, din0, we0, dout0, px, py, on,
            bufsel, idx1, state);
  input clk;
  input reset;
  input rden;
  input [13:0] idx0;
  input [31:0] din0;
  input we0;
  output [31:0] dout0;
  input [9:0] px;
  input [9:0] py;
  output on;
  output bufsel;
  output [13:0] idx1;
  output [2:0] state;

  reg [2:0] state;
  parameter S_waitrden = 0;
  parameter S_init0 = 1;
  parameter S_waity = 2;
  parameter S_waitx = 3;
  parameter S_norm0 = 4;
  parameter S_norm1 = 5;

  reg [13:0] idx1;
  wire [31:0] dout1;
  reg [31:0] curbuf [1:0];
  reg bufsel;

  wire on;
  assign on = |(curbuf[bufsel] & (32'h80000000 >> px[4:0]));

  dmem buf0(idx0, idx1, clk, clk, din0, dout0, dout1, we0);

  always @(posedge clk) begin
    if(!reset)
      state <= S_waitrden;
    else begin
      case(state)
        S_waitrden: begin
          idx1 <= 0;
          bufsel <= 0;
```

```
      curbuf[0] <= 32'b0;
      curbuf[1] <= 32'b0;
      state <= rden?S_init0:S_waitrden;
    end
    S_init0: begin // prefetch first block
      curbuf[bufsel] <= dout1;
      state <= S_waity;
    end
    S_waity: begin // Wait for vblank to end
      if(px==0 && py==0)
        state <= S_norm0;
    end
    S_waitx: begin // Wait for hblank to end
      if(px==0)
        state <= S_norm0;
    end
    S_norm0: begin // Wait for px=0x[blah]1E to prefetch next
      if(px[4:0]==5'h10)
        idx1 <= idx1 + 1;
      else if(px[4:0]==5'h1E) begin
        curbuf[~bufsel] <= dout1;
        state <= S_norm1;
      end
    end
    S_norm1: begin
      bufsel <= ~bufsel;
      state <= (px<630)?
            S_norm0:
            ((py<469)?S_waitx:S_waitrden);
    end
   endcase // case(state)
  end
 end
endmodule
```

```
`timescale 1ns / 1ns

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:15:22 05/10/2006
// Design Name:   wbdisp
// Module Name:   wbdisp_tb.v
// Project Name:  fps
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: wbdisp
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module wbdisp_tb_v;

	// Inputs
	reg clk;
	reg reset;

	// Outputs
	wire on;
	wire bufsel;
	wire [13:0] idx1;
	wire [2:0] state;

  wire [9:0] px;
  wire [9:0] py;
  wire rden;
  sync S(clk, reset, hsync, vsync, sync_b, blank_b, px, py, rden);

	// Instantiate the Unit Under Test (UUT)
```

```
	wbdisp uut (
		.clk(clk),
		.reset(reset),
		.rden(rden),
		.idx0(),
		.din0(),
		.we0(),
		.dout0(),
		.px(px),
		.py(py),
		.on(on),
		.bufsel(bufsel),
		.idx1(idx1),
		.state(state)
	);

  wire [31:0] cbit;
  assign cbit = (32'h80000000 >> px[4:0]);

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

		// Wait 100 ns for global reset to finish
		#100;
		reset = 1;
		// Add stimulus here



	end

  always #1 clk = ~clk;

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
//
//
// Created: March 13, 2006
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

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

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

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

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

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

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

               clock_feedback_out, clock_feedback_in,

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

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

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

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

               switch,

               led,
```

```
        user1, user2, user3, user4,

        daughtercard,

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

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

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

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

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

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

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

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

input  clock_feedback_in;
output clock_feedback_out;

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

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
```

```
input  clock_27mhz, clock1, clock2;

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

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

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

inout [43:0] daughtercard;

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

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

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

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

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

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
```

```
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;

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

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;

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

// Buttons, Switches, and Individual LEDs
// assign led = 8'hFF;

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

```
// Daughtercard Connectors
assign daughtercard = 44'hZ;

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


////////////////////////////////////////////////////////////////////////
//
// Lab 4 Components
//
////////////////////////////////////////////////////////////////////////

//
// Generate a 31.5MHz pixel clock from clock_27mhz
//

wire pclk, clk;
DCM clk_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
// synthesis attribute CLKFX_DIVIDE of clk_dcm is 6
// synthesis attribute CLKFX_MULTIPLY of clk_dcm is 7
// synthesis attribute CLK_FEEDBACK of clk_dcm  is "NONE"
BUFG clk_buf (.I(pclk), .O(clk));

wire reset;
debounce dbsync(button0, clk, button0, reset);

////////////////////
// VGA stuff
wire hsync, vsync;
wire [9:0] px;
wire [9:0] py;
wire rden;
sync s(clk, reset, hsync, vsync, vga_out_sync_b, vga_out_blank_b, px, py, rden);

wire hsyncbar, vsyncbar;
assign hsyncbar = ~hsync;
assign vsyncbar = ~vsync;
delayer hsd(clk, hsyncbar, vga_out_hsync);
delayer vsd(clk, vsyncbar, vga_out_vsync);

wire [24:0] rgb;

assign vga_out_pixel_clock = ~clk;
assign vga_out_red = rgb[23:16];
assign vga_out_green = rgb[15:8];
assign vga_out_blue = rgb[7:0];
// End VGA stuff
////////////////////
```

```
  wire [5:0] cmd;
  wire cmden_db;
  wire cmden;

  debounce cdb(reset, clk, button1, cmden_db);
  pulseinvert cpi(clk, reset, cmden_db, cmden);

  assign cmd = switch[5:0];
  wire [9:0] curx;
  wire [9:0] cury;

/*
  mouse fakemouse(clk, reset,
              button_up, button_down, button_left, button_right,
              curx, cury);
  */
  wire [13:0] shidx;
  wire [31:0] shdout;
  wire shwren;

  //Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = {user4[29:22], user4[7:0]};
   assign analyzer2_clock = clock_27mhz;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
 //  assign analyzer4_data = {/*user3[7:0]*/2'h0, x1_vel/*user3[29:22]*/};
   assign analyzer4_clock = clock_27mhz;


  wire [7:0] fakeled;
  whiteboard wbc(clk, reset, rden, px, py, curx, cury, cmd, cmden, rgb, fakeled,
              shidx, shdout, shwren);

      //instantiate AD670_controller
      wire r_w_bar, dataavail, x1_dataavail;

      AD670_controller x1_controller(.clk(clk), .status(user4[8]), .dataavail(x1_dataavail),
                                                .r_w_bar(r_w_bar_x1), .state(), .reset(~reset), .count(), .enable());

      AD670_controller y1(.clk(clk), .status(user4[30]), .dataavail(y1_dataavail),
                                                .r_w_bar(r_w_bar_y1), .state(), .reset(~reset), .count(), .enable());

      AD670_controller x2(.clk(clk), .status(user2[8]), .dataavail(),
                                                .r_w_bar(r_w_bar_x2), .state(), .reset(~reset), .count(), .enable());

      AD670_controller y2(.clk(clk), .status(user1[8]), .dataavail(),
                                                .r_w_bar(r_w_bar_y2), .state(), .reset(~reset), .count(), .enable());

      AD670_controller x3(.clk(clk), .status(user3[30]), .dataavail(),
                                                .r_w_bar(r_w_bar_x3), .state(), .reset(~reset), .count(), .enable());

      AD670_controller y3(.clk(clk), .status(user3[8]), .dataavail(),
```

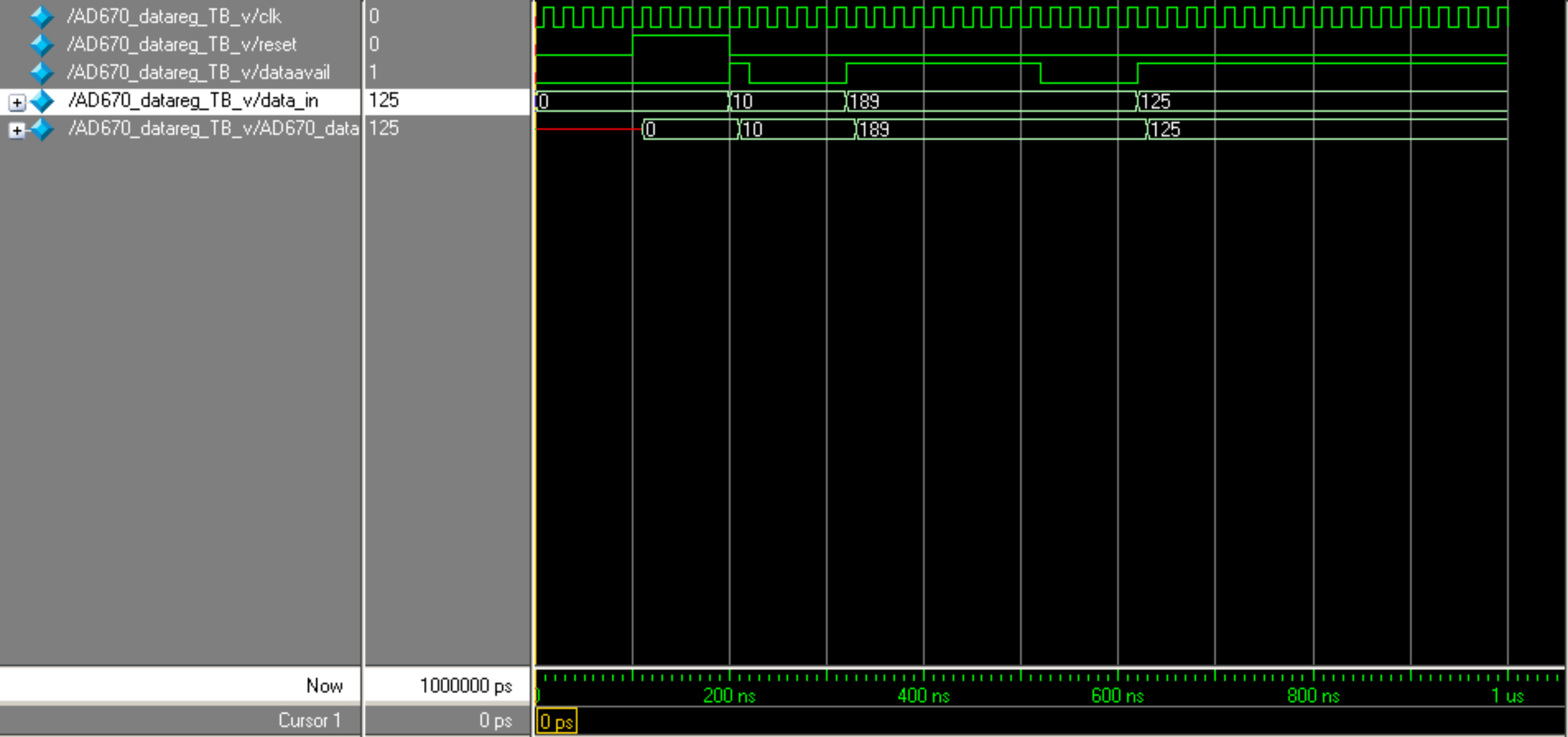
```
                                        .r_w_bar(r_w_bar_y3), .state(), .reset(~reset), .count(), .enable());



//nathan dataflow starts here
    //instantiate AD670_datareg
    wire [7:0] AD670_data, data_in, x1_reg;

    AD670_datareg x1_datareg(.clk(clk), .reset(~reset), .dataavail(x1_dataavail), .AD670_data(x1_reg),
                             .data_in(user4[7:0]));

    wire [7:0] y1_reg;
    AD670_datareg y1_datareg(.clk(clk), .reset(~reset), .dataavail(y1_dataavail), .AD670_data(y1_reg),
                             .data_in(user4[29:22]));




    //instantiate FSM_Integrator_Accel
    wire [7:0] BIAS, accel, x1_accel;
    wire direct_a, x1_direct_a;

    FSM_Integrator_Accel x1_accelfsm(.clk(clk), .reset(~reset), .AD670_data(x1_reg), .direct_a(x1_direct_a),
                                     .accel(x1_accel), .BIAS(168));

    wire [7:0] y1_accel;
    wire y1_direct_a;

    FSM_Integrator_Accel y1_accelfsm(.clk(clk), .reset(~reset), .AD670_data(y1_reg), .direct_a(y1_direct_a),
                                     .accel(y1_accel), .BIAS(157));

    //instantiate Integrator_Accel
    wire [14:0] vel, x1_vel;

    Integrator_Accel x1_accelintegrator(.clk(clk), .reset(~reset), .direct_a(x1_direct_a), .direct_v(x1_direct_v),
                                        .accel(x1_accel), .vel(x1_vel));

    wire [14:0] y1_vel;

    Integrator_Accel y1_accelintegrator(.clk(clk), .reset(~reset), .direct_a(y1_direct_a), .direct_v(y1_direct_v),
                                        .accel(y1_accel), .vel(y1_vel));



    assign analyzer4_data = {/*user3[7:0]*/2'h0, x1_vel/*user3[29:22]*/};


    //instantiate Integrator_Position
```

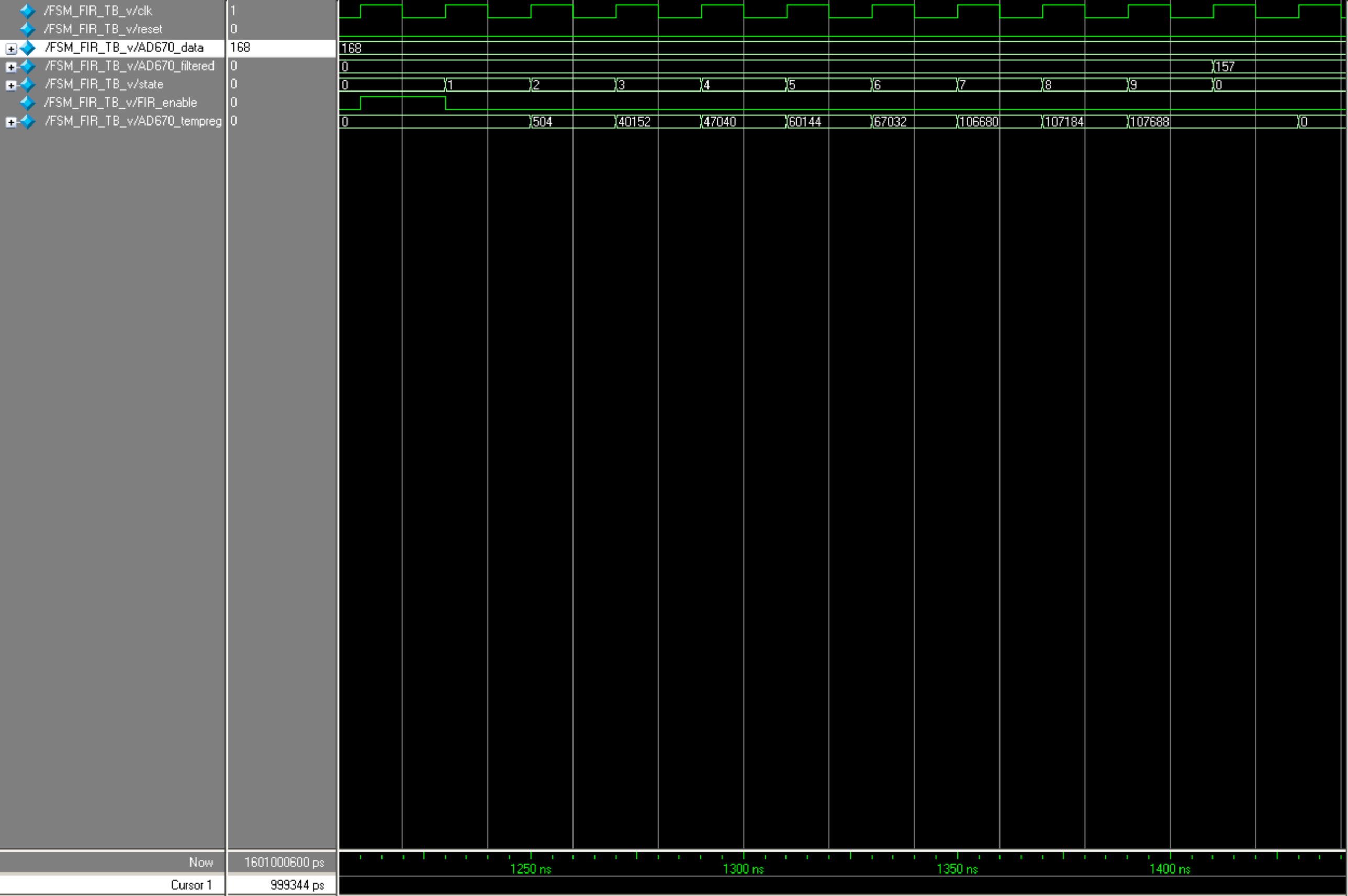
```
    wire reset_position;
    wire [9:0] axis_constant, position, x1_position;
    wire [7:0] vel_shift, x1_vel_shift;

    Integrator_Position x1_positionintegrator(.clk(clk), .reset(~reset), .reset_position(~reset), .direct_a(x1_direct_a), .vel(x1_accel),
                                              .axis_constant(639), .position(x1_position), .vel_shift(x1_vel_shift));

    wire [9:0] y1_position;
    wire [7:0] y1_vel_shift;

    Integrator_Position y1_positionintegrator(.clk(clk), .reset(~reset), .reset_position(~reset), .direct_a(y1_direct_a), .vel(y1_accel),
                                              .axis_constant(469), .position(y1_position), .vel_shift(y1_vel_shift));

assign curx = x1_position;
    assign cury = y1_position;

    reg [11:0] mdcount;
    wire [4:0] mnum;
    reg [4:0] lastnum;
    reg mden;
    wire aen;
    always@(posedge clk) begin
        if(!reset)
            mdcount <= 0;
        else if(mdcount == ~12'hFFF) begin
            mdcount <= 0;
            mden <= 1;
        end else begin
            mdcount <= mdcount + 1;
            mden <= 0;
        end

        if(aen)
            lastnum <= mnum;
    end

    assign led = {3'h7, ~lastnum};
    mdetect md(clk, reset, x1_direct_v, 1, x1_vel_shift, mden, mnum, aen);


    // User IO for x1 and y1
    assign user4[9] = 1'b1 ? r_w_bar_x1 : 1'bz;
    assign user4[31] = 1'b1 ? r_w_bar_y1 : 1'bz;
    assign user4[8:0] = 9'bz;
    assign user4[30:22] = 9'bz;
    assign user4[21:10] = 12'hZ;

    // User IO for x2 and y2
assign user2[9] = 1'b1 ? r_w_bar_x2 : 1'bz;
    assign user2[8:0] = 9'bz;
    assign user2[31:10] = 22'hZ;
    assign user1[9] = 1'b1 ? r_w_bar_y2 : 1'bz;
    assign user1[8:0] = 9'bz;
    assign user1[31:10] = 22'hZ;
```

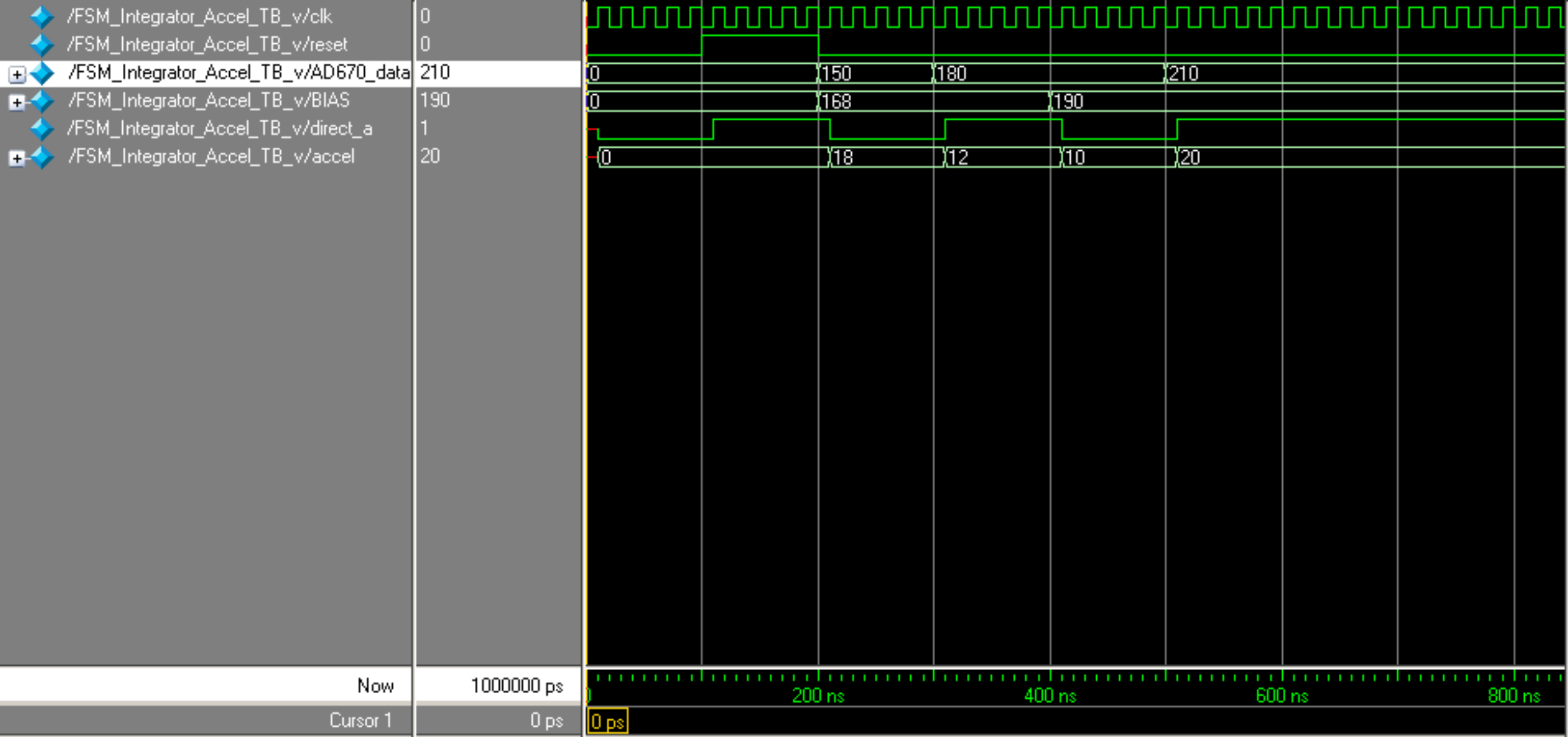
```
      // User IO for x3 and y3
  assign user3[9] = 1'b1 ? r_w_bar_y3 : 1'bz;
      assign user3[31] = 1'b1 ? r_w_bar_x3 : 1'bz;
      assign user3[8:0] = 9'bz;
      assign user3[30:22] = 9'bz;
      assign user3[21:10] = 12'hZ;

endmodule

// To simulate mouse inputs
module mouse(clk, reset, button_up, button_down, button_left, button_right,
          x, y);
  input clk;
  input reset;
  input button_up;
  input button_down;
  input button_left;
  input button_right;
  output [9:0] x;
  output [9:0] y;

  reg [9:0] x;
  reg [9:0] y;
  wire up, down, left, right;
  debounce bud(reset, clk, button_up, up);
  debounce bdd(reset, clk, button_down, down);
  debounce bld(reset, clk, button_left, left);
  debounce brd(reset, clk, button_right, right);

  parameter cycle = 150000;
  reg [19:0] count;
  reg [9:0] nx;
  reg [9:0] ny;

  always @(posedge clk) begin
    if(!reset) begin
      x <= 0;
      y <= 0;
      count <= 0;
    end else if(count==cycle) begin
      count <= 0;
      x <= nx;
      y <= ny;
    end else begin
      count <= count + 1;
    end
  end

  always @(count) begin
    nx = x;
    ny = y;
    if(count==cycle) begin
      if(!up)
```

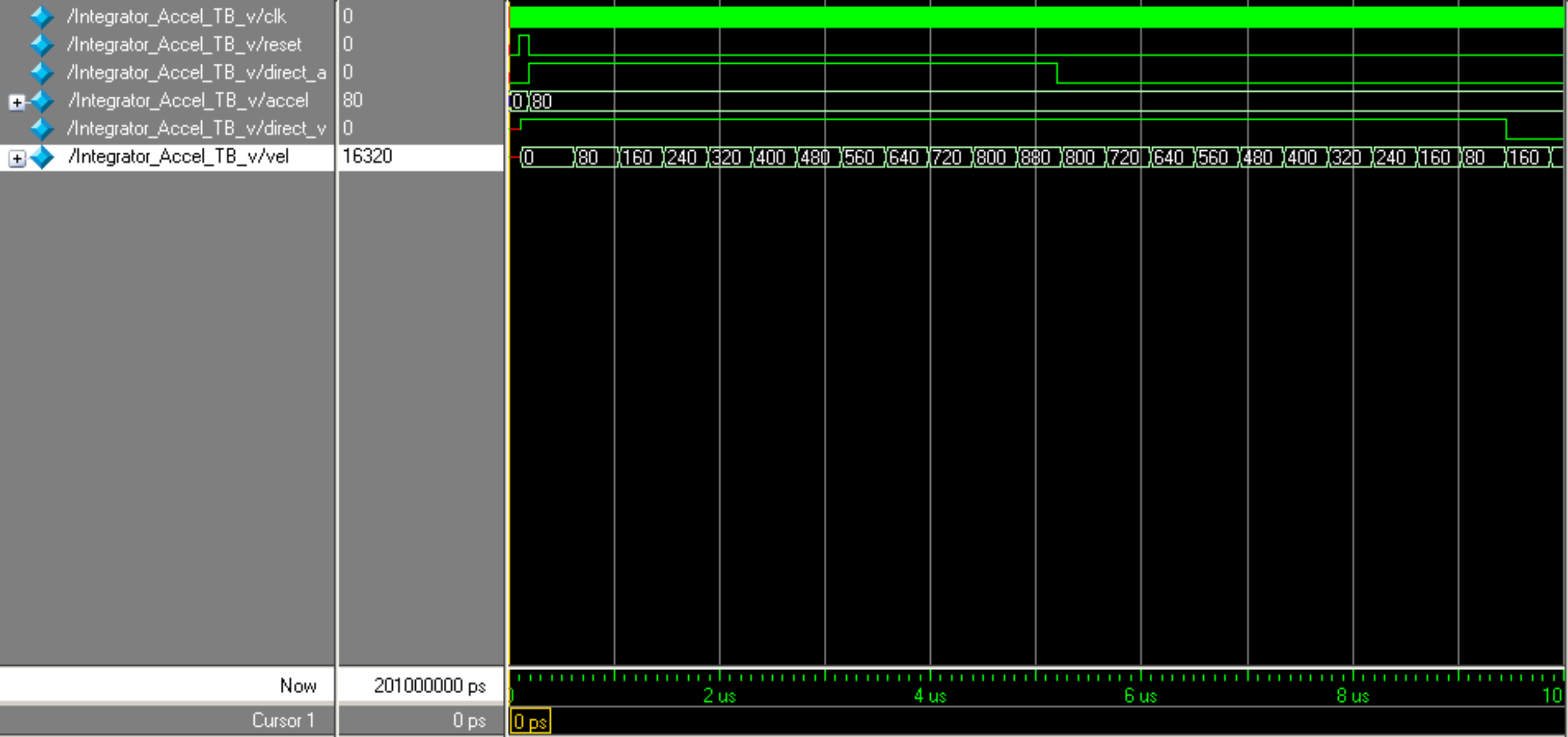
```
      ny = y - 1;
    else if(!down)
      ny = y + 1;
    if(!left)
      nx = x - 1;
    else if(!right)
      nx = x + 1;

    if(ny > 900)
      ny = 0;
    else if(ny > 469)
      ny = 469;

    if(nx > 900)
      nx = 0;
    else if(nx > 639)
      nx = 639;
   end
  end
endmodule

module debounce (reset, clock, noisy, clean);
  parameter DELAY = 315000;   // .01 sec with a 31.5Mhz 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 delayer(clk, in, out);
  input clk;
  input in;
  output out;

  reg out;
  reg [5:0] queue;

  always @(posedge clk) begin
    queue <= 2*queue + in;
    out = queue[3];
```

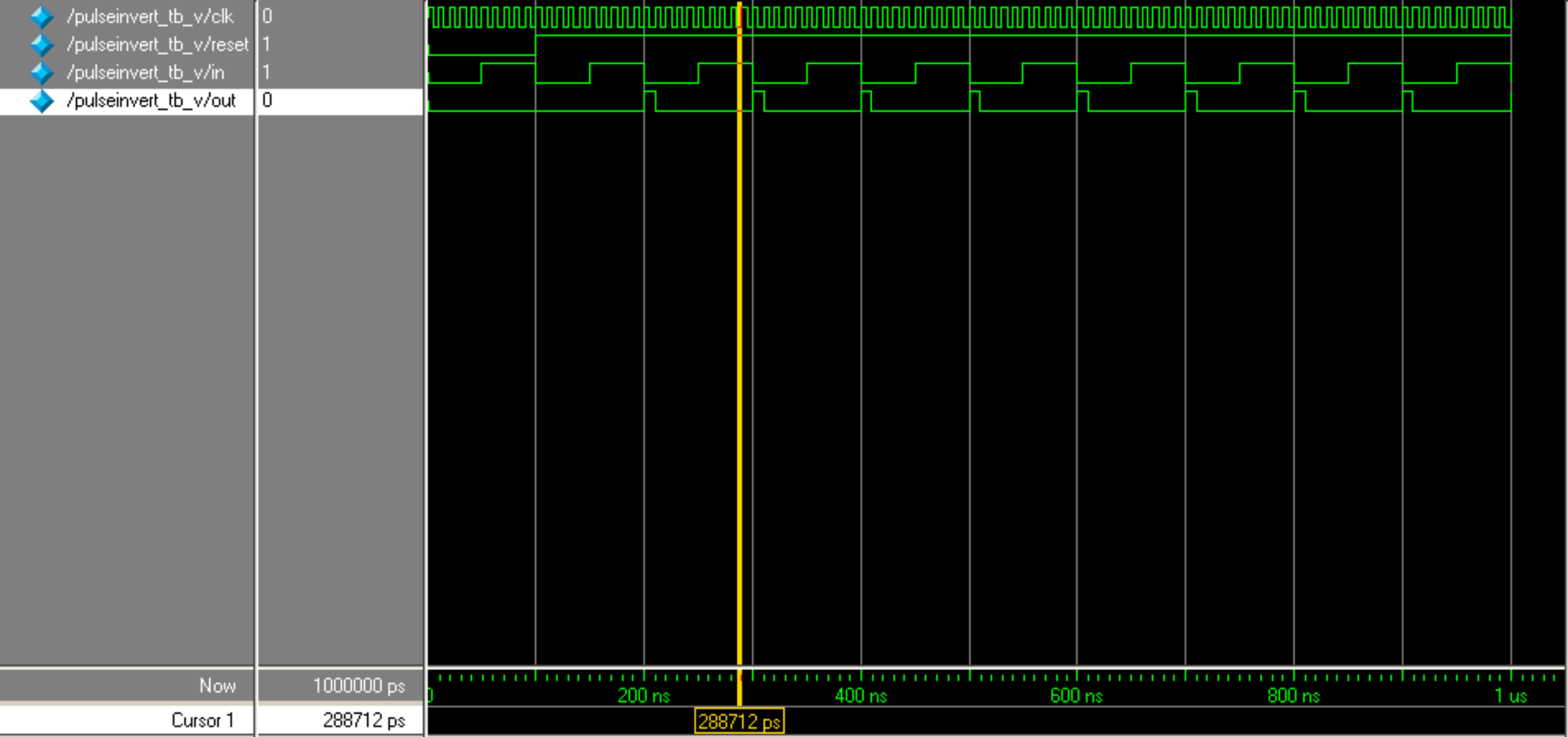
```
  end
endmodule // delayer

module pulseinvert(clk, reset, in, out);
  input clk;
  input reset;
  input in;
  output out;

  reg canfire;
  reg out;

  always @(posedge clk) begin
    if(!reset) begin
      out <= 0;
    end else if(out)
      out <= 0;
    else if(!in) begin
      if(canfire) begin
        out <= 1;
        canfire <= 0;
      end
    end else
      canfire <= 1;
  end
endmodule
```

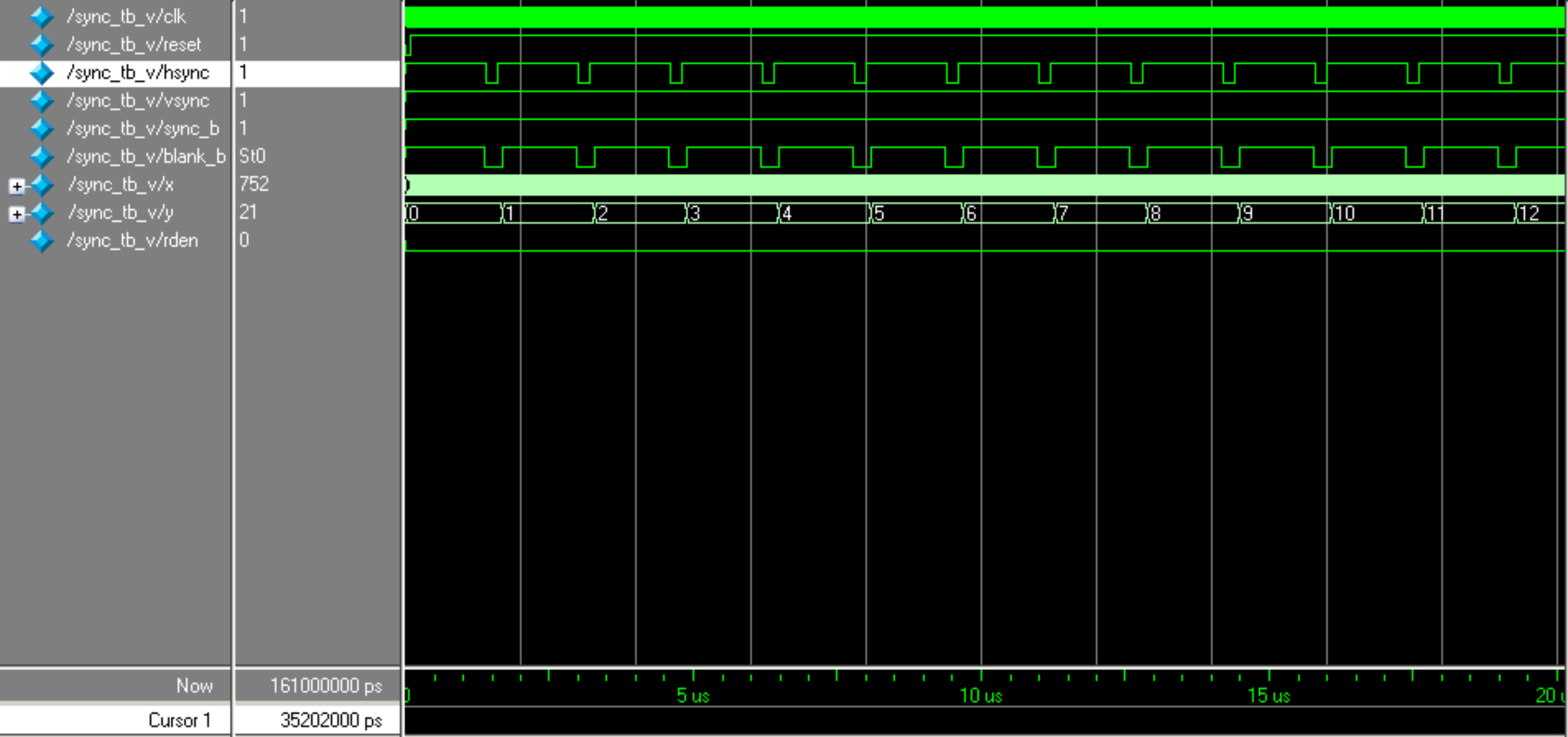

/AD670_datareg_TB_v/clk
0
/AD670_datareg_TB_v/reset
0
/AD670_datareg_TB_v/dataavail
1
/AD670_datareg_TB_v/data_in
125
0
10
189
125
/AD670_datareg_TB_v/AD670_data
125
0
10
189
125
Now
1000000 ps
200 ns
400 ns
600 ns
800 ns
1 us
Cursor 1
0 ps
0 ps

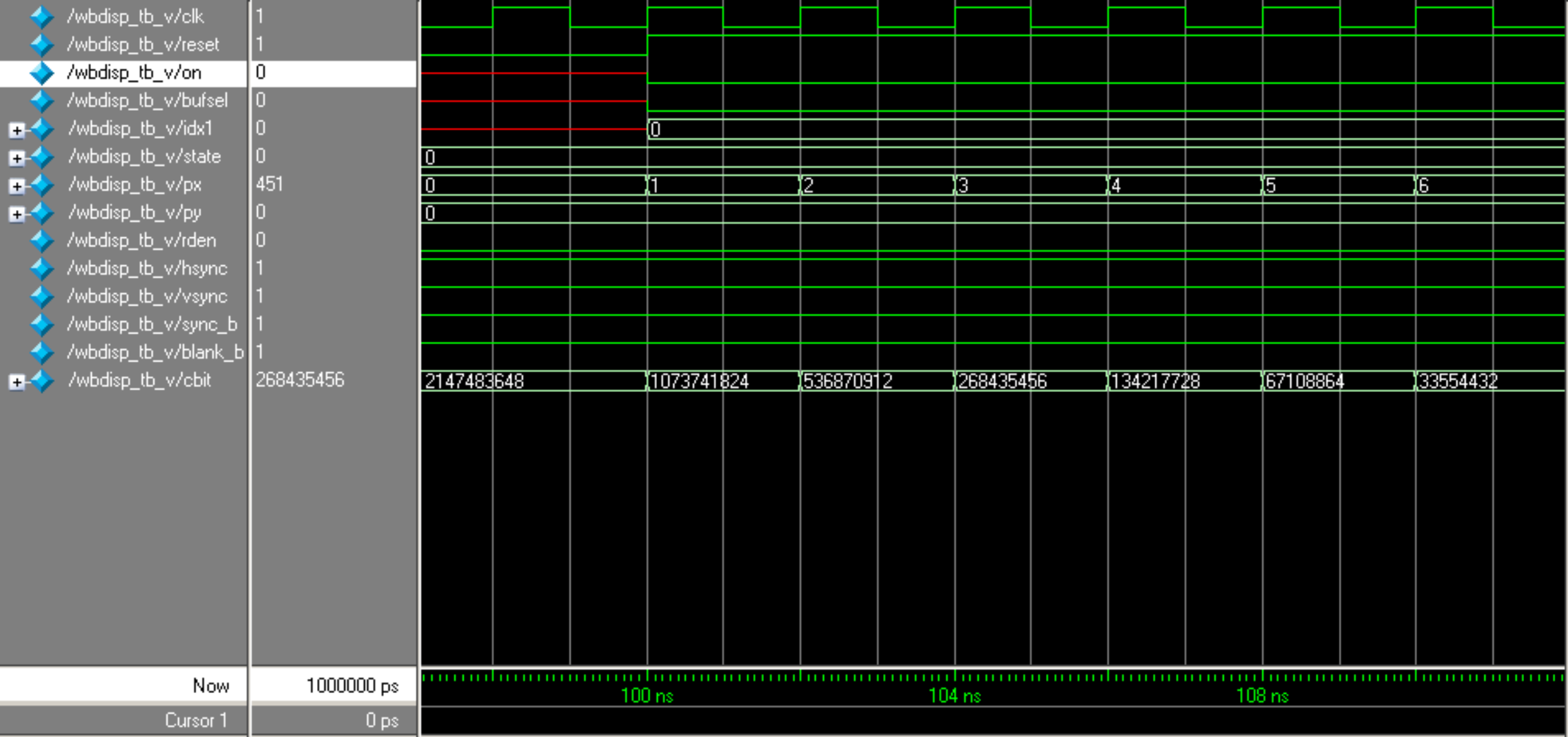

/FSM_FIR_TB_v/clk
/FSM_FIR_TB_v/reset
/FSM_FIR_TB_v/AD670_data
/FSM_FIR_TB_v/AD670_filtered
/FSM_FIR_TB_v/state
/FSM_FIR_TB_v/FIR_enable
/FSM_FIR_TB_v/AD670_tempreg
1
0
168
0
0
168
0
1
2
3
4
5
6
7
8
9
157
0
0
504
40152
47040
60144
67032
106680
107184
107688
0
Now
1601000600 ps
Cursor 1
999344 ps
1250 ns
1300 ns
1350 ns
1400 ns


/FSM_Integrator_Accel_TB_v/clk
0
/FSM_Integrator_Accel_TB_v/reset
0
/FSM_Integrator_Accel_TB_v/AD670_data
210
0
150
180
210
/FSM_Integrator_Accel_TB_v/BIAS
190
0
168
190
/FSM_Integrator_Accel_TB_v/direct_a
1
/FSM_Integrator_Accel_TB_v/accel
20
0
18
12
10
20
Now
1000000 ps
200 ns
400 ns
600 ns
800 ns
Cursor 1
0 ps
0 ps


/Integrator_Accel_TB_v/clk
/Integrator_Accel_TB_v/reset
/Integrator_Accel_TB_v/direct_a
/Integrator_Accel_TB_v/accel
/Integrator_Accel_TB_v/direct_v
/Integrator_Accel_TB_v/vel
0
0
0
80
0
16320
0 80
0 80 160 240 320 400 480 560 640 720 800 880 800 720 640 560 480 400 320 240 160 80 160
Now
201000000 ps
Cursor 1
0 ps
2 us
4 us
6 us
8 us
0 ps


/pulseinvert_tb_v/clk
/pulseinvert_tb_v/reset
/pulseinvert_tb_v/in
/pulseinvert_tb_v/out
0
1
1
0
Now
1000000 ps
Cursor 1
288712 ps
200 ns
400 ns
600 ns
800 ns
1 us
288712 ps


/sync_tb_v/clk
1
/sync_tb_v/reset
1
/sync_tb_v/hsync
1
/sync_tb_v/vsync
1
/sync_tb_v/sync_b
1
/sync_tb_v/blank_b
St0
/sync_tb_v/x
752
/sync_tb_v/y
21
/sync_tb_v/rden
0
0
1
2
3
4
5
6
7
8
9
10
11
12
Now
161000000 ps
5 us
10 us
15 us
Cursor 1
35202000 ps


/wbdisp_tb_v/clk
1
/wbdisp_tb_v/reset
1
/wbdisp_tb_v/on
0
/wbdisp_tb_v/bufsel
0
/wbdisp_tb_v/idx1
0
/wbdisp_tb_v/state
0
/wbdisp_tb_v/px
451
/wbdisp_tb_v/py
0
/wbdisp_tb_v/rden
0
/wbdisp_tb_v/hsync
1
/wbdisp_tb_v/vsync
1
/wbdisp_tb_v/sync_b
1
/wbdisp_tb_v/blank_b
1
/wbdisp_tb_v/cbit
268435456
0
0
0
1
2
3
4
5
6
0
2147483648
1073741824
536870912
268435456
134217728
67108864
33554432
Now
1000000 ps
Cursor 1
0 ps
100 ns
104 ns
108 ns