Ryan Chang prelab7

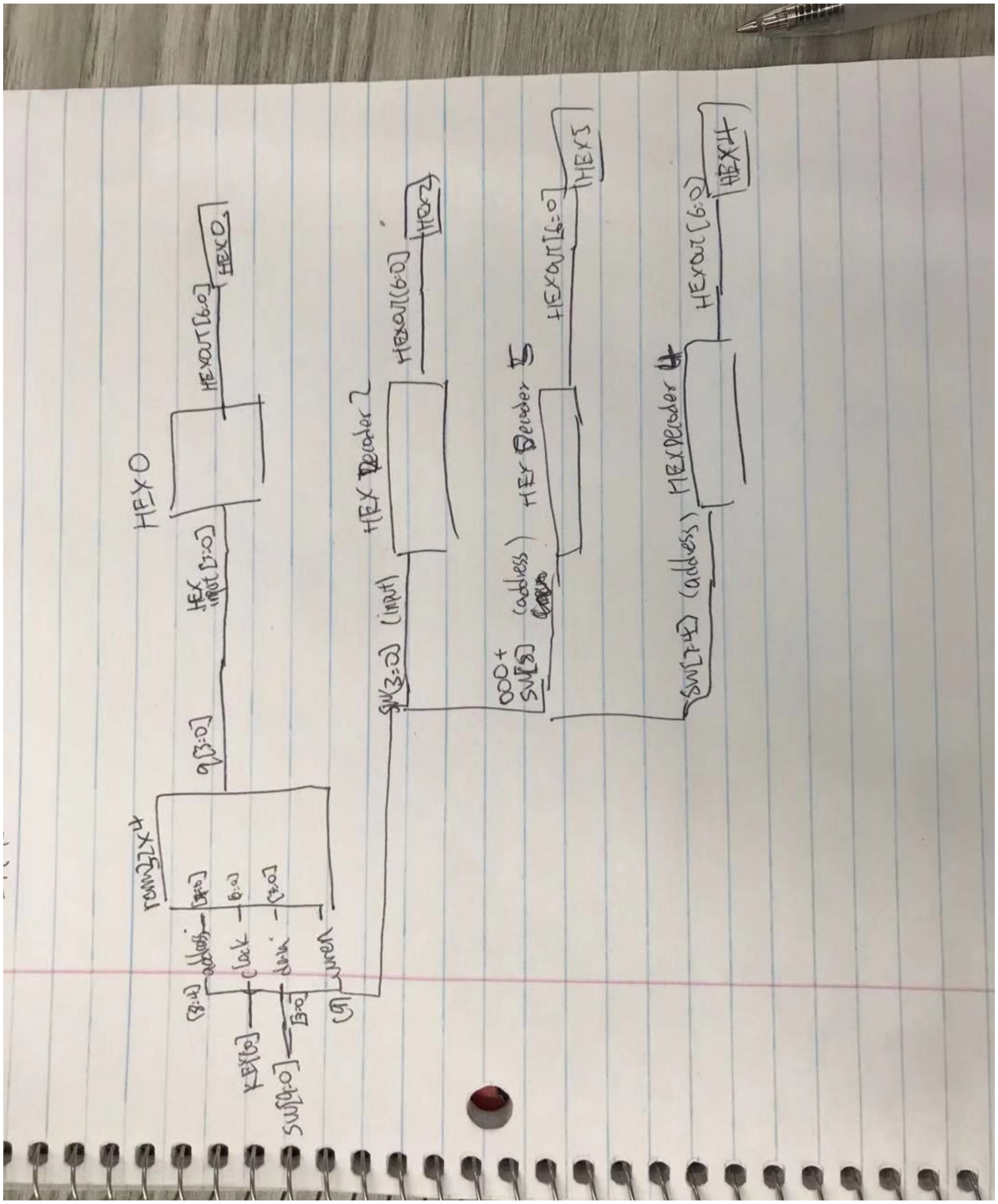
PART 1 VERILOG

```
module ram(SW, KEY, HEX5, HEX4, HEX2, HEX0);
   input [9:0] SW;
   input [0:0]KEY;
   output [6:0] HEX5, HEX4, HEX2, HEX0;
   wire [3:0] out;

   ram32x4 r(
         .address(SW[8:4]),
         .clock(KEY[0]),
         .data(SW[3:0]),
         .wren(SW[9]),
         .q(out[3:0])
   );

   wire [3:0] w;
   assign w[3:0] = {3'b000, SW[8]};

   hex_decoder h5(
        .hex_digit(w[3:0]),
        .hexOut(HEX5)
        );

   hex_decoder h4(
        .hex_digit(SW[7:4]),
        .hexOut(HEX4)
        );

   hex_decoder h2(
        .hex_digit(SW[3:0]),
        .hexOut(HEX2)
        );

   hex_decoder h0(
        .hex_digit(out[3:0]),
        .hexOut(HEX0)
        );
endmodule

module ram32x4 (
   address,
	clock,
	data,
	wren,
	q);

	input	[4:0]  address;
	input	  clock;
	input	[3:0]  data;
	input	  wren;
	output	[3:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [3:0] sub_wire0;
	wire [3:0] q = sub_wire0[3:0];

	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.data_a (data),
				.wren_a (wren),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_b (1'b1),
				.eccstatus (),
				.q_b (),
				.rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_b (1'b0));
    defparam
            altsyncram_component.clock_enable_input_a = "BYPASS",
            altsyncram_component.clock_enable_output_a = "BYPASS",
            altsyncram_component.intended_device_family = "Cyclone V",
            altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
            altsyncram_component.lpm_type = "altsyncram",
            altsyncram_component.numwords_a = 32,
            altsyncram_component.operation_mode = "SINGLE_PORT",
            altsyncram_component.outdata_aclr_a = "NONE",
            altsyncram_component.outdata_reg_a = "UNREGISTERED",
            altsyncram_component.power_up_uninitialized = "FALSE",
            altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
            altsyncram_component.widthad_a = 5,
            altsyncram_component.width_a = 4,
            altsyncram_component.width_byteena_a = 1;

endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
```

```
module ram32x4 (
    address,
    clock,
    data,
    wren,
    q);

    input    [4:0]  address;
    input      clock;
    input    [3:0]  data;
    input      wren;
    output    [3:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1      clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [3:0] sub_wire0;
    wire [3:0] q = sub_wire0[3:0];

    altsyncram    altsyncram_component (
                                .address_a (address),
                                .clock0 (clock),
                                .data_a (data),
                                .wren_a (wren),
                                .q_a (sub_wire0),
                                .aclr0 (1'b0),
                                .aclr1 (1'b0),
                                .address_b (1'b1),
                                .addressstall_a (1'b0),
                                .addressstall_b (1'b0),
                                .byteena_a (1'b1),
                                .byteena_b (1'b1),
                                .clock1 (1'b1),
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_b (1'b0));
    defparam
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
        altsyncram_component.intended_device_family = "Cyclone V",
        altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 32,
        altsyncram_component.operation_mode = "SINGLE_PORT",
        altsyncram_component.outdata_aclr_a = "NONE",
        altsyncram_component.outdata_reg_a = "UNREGISTERED",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.read_during_write_mode_port_a =
"NEW_DATA_NO_NBE_READ",
        altsyncram_component.widthad_a = 5,
        altsyncram_component.width_a = 4,
        altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "32"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "5"
// Retrieval info: PRIVATE: WidthData NUMERIC "4"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "4"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 5 0 INPUT NODEFVAL "address[4..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 4 0 INPUT NODEFVAL "data[3..0]"
// Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 5 0 address 0 0 5 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 4 0 data 0 0 4 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 4 0 @q_a 0 0 4 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf


module hex_decoder(hex_digit, hexOut);
	input [3:0] hex_digit;
	output reg [6:0] hexOut;

	always @(*)
	case (hex_digit)
	4'h0: hexOut = 7'b100_0000;
	4'h1: hexOut = 7'b111_1001;
	4'h2: hexOut = 7'b010_0100;
	4'h3: hexOut = 7'b011_0000;
	4'h4: hexOut = 7'b001_1001;
	4'h5: hexOut = 7'b001_0010;
	4'h6: hexOut = 7'b000_0010;
	4'h7: hexOut = 7'b111_1000;
	4'h8: hexOut = 7'b000_0000;
	4'h9: hexOut = 7'b001_1000;
	4'hA: hexOut = 7'b000_1000;
	4'hB: hexOut = 7'b000_0011;
	4'hC: hexOut = 7'b100_0110;
	4'hD: hexOut = 7'b010_0001;
	4'hE: hexOut = 7'b000_0110;
	4'hF: hexOut = 7'b000_1110;
	default: hexOut = 7'h7f;
	endcase
endmodule
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "32"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "5"
// Retrieval info: PRIVATE: WidthData NUMERIC "4"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "4"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 5 0 INPUT NODEFVAL "address[4..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 4 0 INPUT NODEFVAL "data[3..0]"
// Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 5 0 address 0 0 5 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 4 0 data 0 0 4 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 4 0 @q_a 0 0 4 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf


module hex_decoder(hex_digit, hexOut);
        input [3:0] hex_digit;
```

```
	output reg [6:0] hexOut;

	always @(*)
	case (hex_digit)
	4'h0: hexOut = 7'b100_0000;
	4'h1: hexOut = 7'b111_1001;
	4'h2: hexOut = 7'b010_0100;
	4'h3: hexOut = 7'b011_0000;
	4'h4: hexOut = 7'b001_1001;
	4'h5: hexOut = 7'b001_0010;
	4'h6: hexOut = 7'b000_0010;
	4'h7: hexOut = 7'b111_1000;
	4'h8: hexOut = 7'b000_0000;
	4'h9: hexOut = 7'b001_1000;
	4'hA: hexOut = 7'b000_1000;
	4'hB: hexOut = 7'b000_0011;
	4'hC: hexOut = 7'b100_0110;
	4'hD: hexOut = 7'b010_0001;
	4'hE: hexOut = 7'b000_0110;
	4'hF: hexOut = 7'b000_1110;
	default: hexOut = 7'h7f;
	endcase
Endmodule
```

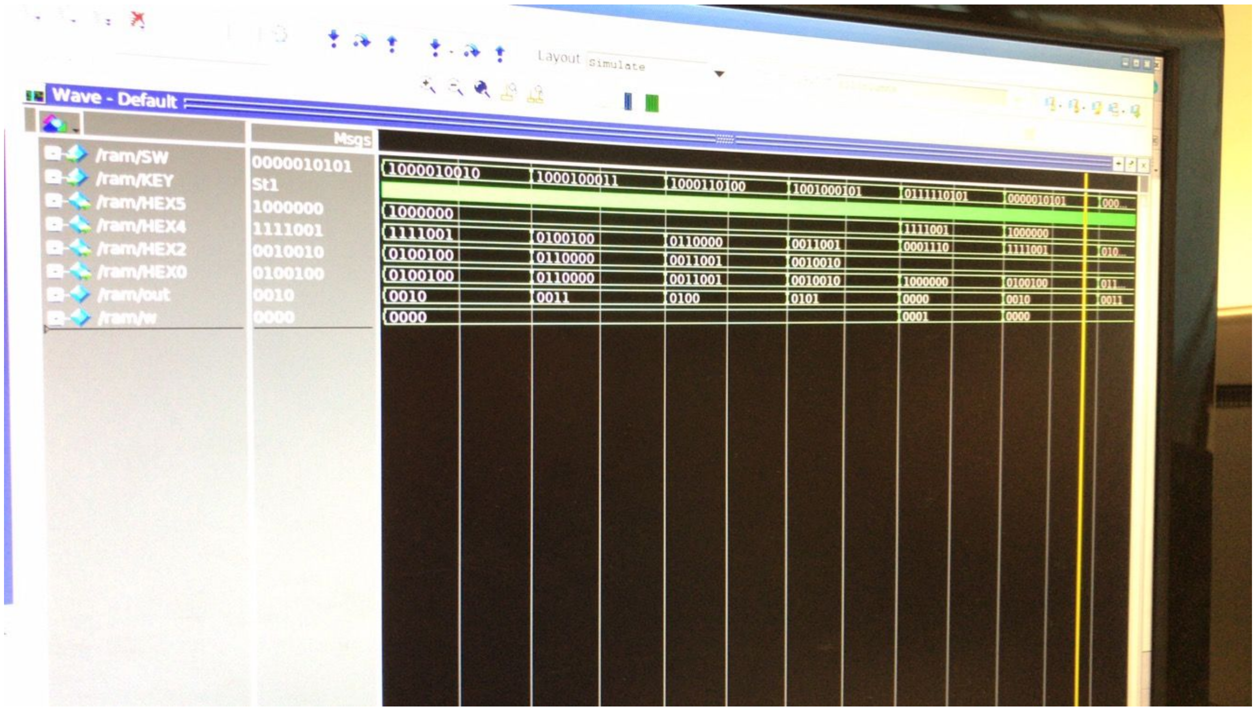
PART 1 SCHEMATIC



PART 1 VSIM CODE

```
vlib work
vlog -timescale 1ns/1ns ram.v
vsim -L altera_mf_ver ram
```

```
log {/*}
add wave {/*}
force {KEY[0]} 0 0, 1 10 -r 20
#write
force {SW[9]} 1
force {SW[8:4]} 00001
force {SW[3:0]} 0010
run 20ns
force {SW[8:4]} 00010
force {SW[3:0]} 0011
run 20ns
force {SW[8:4]} 00011
force {SW[3:0]} 0100
run 20ns
force {SW[8:4]} 00100
force {SW[3:0]} 0101
run 20ns
# SW9 == 0 IS READ
force {SW[9]} 0

force {SW[8:4]} 11111
run 20ns
force {SW[8:4]} 00001
run 20ns
force {SW[8:4]} 00010
run 20ns
force {SW[8:4]} 00011
run 20ns
force {SW[8:4]} 00100
run 20ns
```



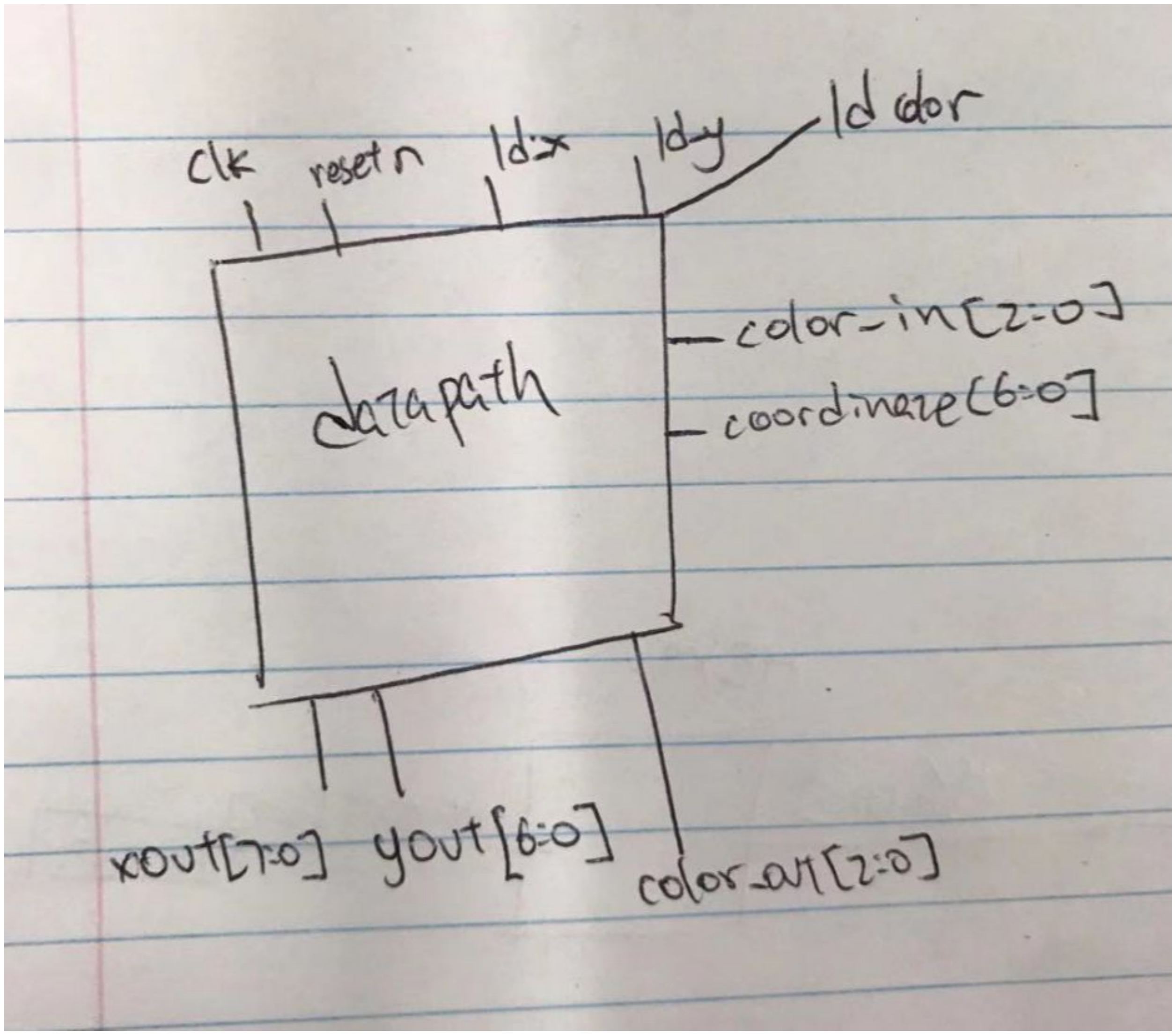
```
/ram/SW     0000010101  1000010010  1000100011  1000110100  1001000101  0111110101  0000010101  000...
/ram/KEY    St1
/ram/HEX5   1000000     1000000
/ram/HEX4   1111001     1111001     0100100     0110000     0011001     1111001     1000000
/ram/HEX2   0010010     0100100     0110000     0011001     0010010     0001110     1111001     010...
/ram/HEX0   0100100     0100100     0110000     0011001     0010010     1000000     0100100     011...
/ram/out    0010        0010        0011        0100        0101        0000        0010        0011
/ram/w      0000        0000                                            0001        0000
```

PART 2 DATAPATH VERILOG

```
module datapath(
	input clk, resetn, ld_x, ld_y, ld_color,
	input [2:0] color_in,
	input [0:0] enable,
	input [6:0] coordinate,
	output [7:0] x_out,
	output [6:0] y_out,
	output [2:0] color_out
	);

	reg [7:0] x;
	reg [6:0] y;
	reg [2:0] color;
	reg [3:0] counter;

	//reset or load
	always @(posedge clk) begin
		if (!resetn) begin
			x <= 8'b0;
			y <= 7'b0;
			color <= 3'b0;
		end
		else begin
			if (ld_x)
```

```
                    x <= {1'b0, coordinate};
                if (ld_y)
                    y <= coordinate;
                if (ld_color)
                    color <= color_in;
            end
    end
    //counter
    always @(posedge clk) begin
            if (!resetn)
                counter <= 4'b0000;
            else if (enable)
                if (counter == 1111)
                    counter <= 4'b0000;
                else
                    counter <= counter + 1'b1;
    end

    assign x_out = x + counter[1:0];
    assign y_out = y + counter[3:2];
    assign color_out = color;
endmodule
```

PART 2 DATAPATH SCHEMATIC

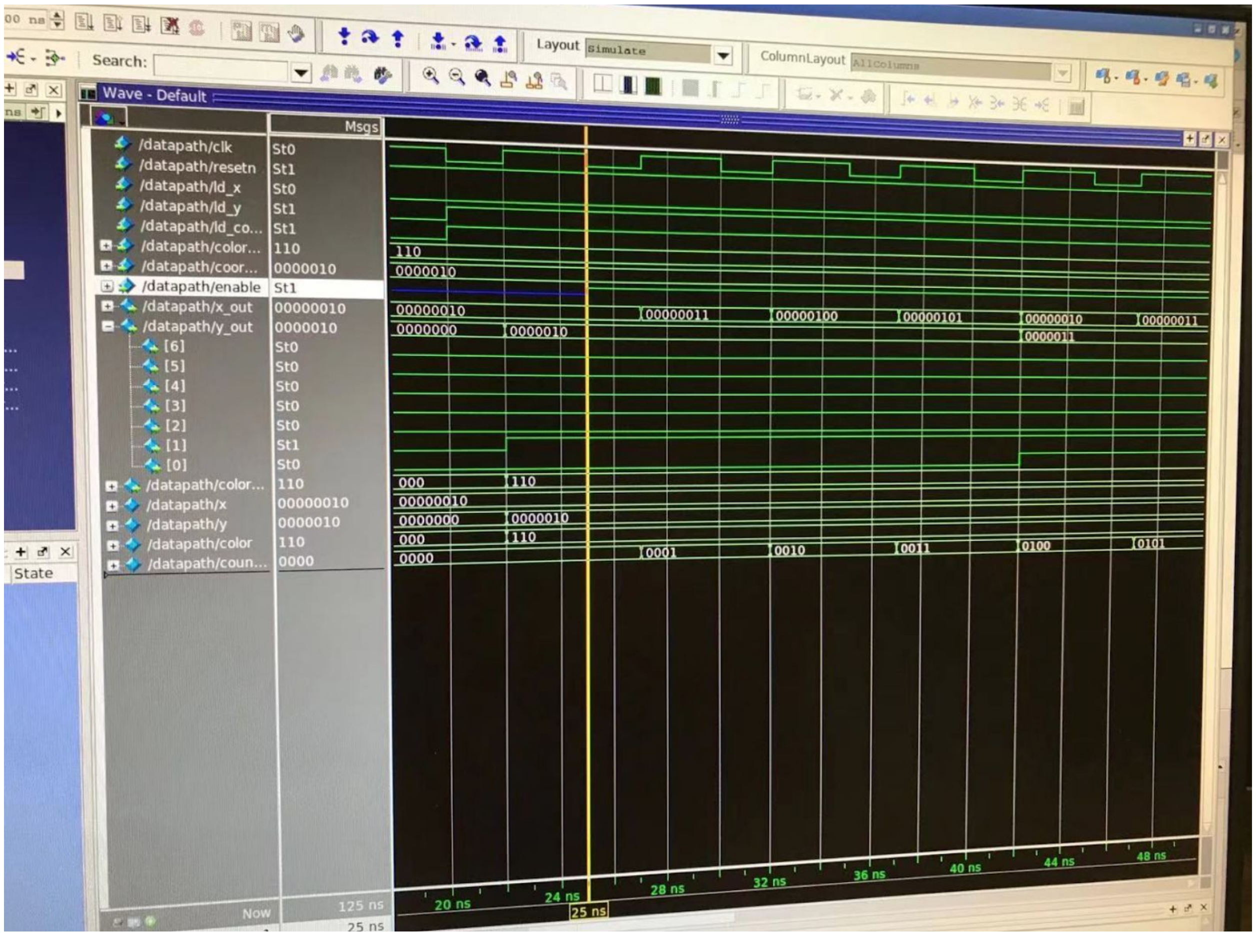


PART 2 DATAPATH VSIM CODE

```
vlib work
vlog -timescale 1ns/1ns part2dataparh.v
vsim datapath
log {/*}
add wave {/*}
force {clk} 0 0, 1 2.5 -r 5
force {resetn} 0
run 5ns
force {resetn} 1
```

```
run 5ns
force {ld_x} 1
force {ld_y} 0
force {ld_color} 0
force {color_in[2:0]} 110
force {coordinate[6:0]} 0000010
run 5ns
force {ld_x} 0
run 5ns
force {ld_y} 1
force {ld_color} 1
run 5ns
force {enable[0:0]} 1
force {color_in[2:0]} 110
force {coordinate[6:0]} 0000010
run 100ns
```

PART 2 DATAPATH VSIM



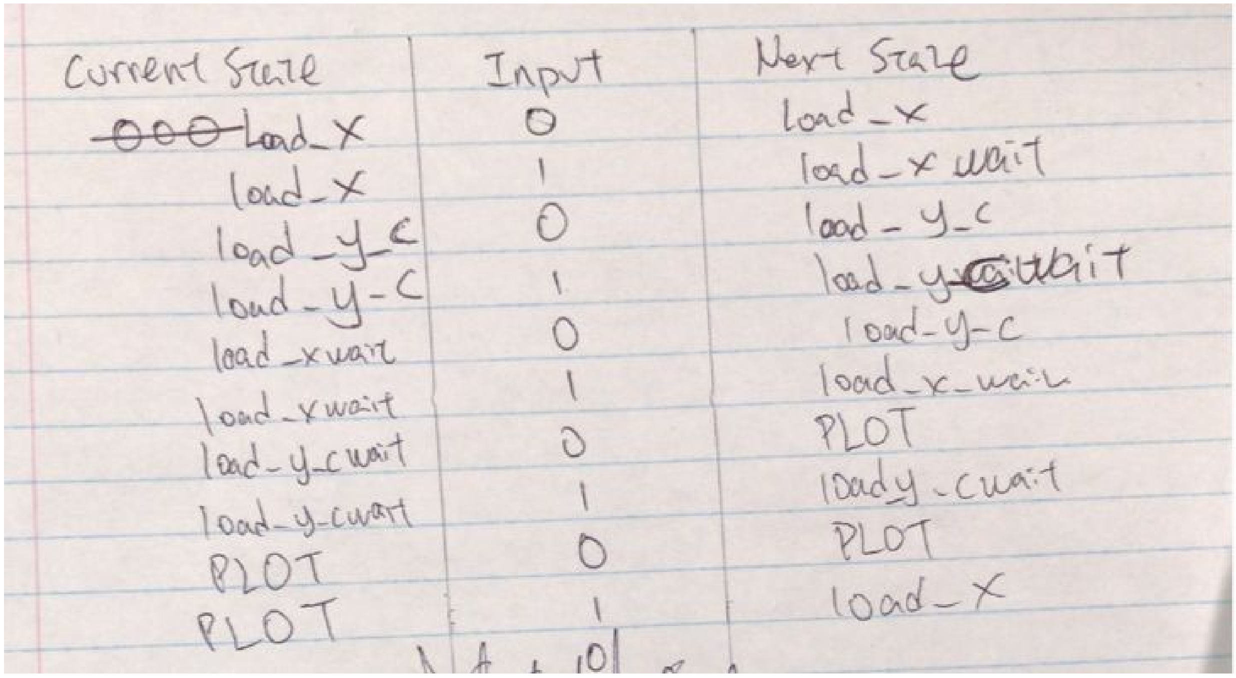
```
/datapath/clk          St0
/datapath/resetn       St1
/datapath/ld_x         St0
/datapath/ld_y         St1
/datapath/ld_co...     St1
/datapath/color...     110
/datapath/coor...      0000010
/datapath/enable       St1
/datapath/x_out        00000010
/datapath/y_out        0000010
  [6]                  St0
  [5]                  St0
  [4]                  St0
  [3]                  St0
  [2]                  St0
  [1]                  St1
  [0]                  St0
/datapath/color...     110
/datapath/x            00000010
/datapath/y            0000010
/datapath/color        110
/datapath/coun...      0000
Now                    125 ns
                       25 ns
```

PART 2 CONTROL VERILOG

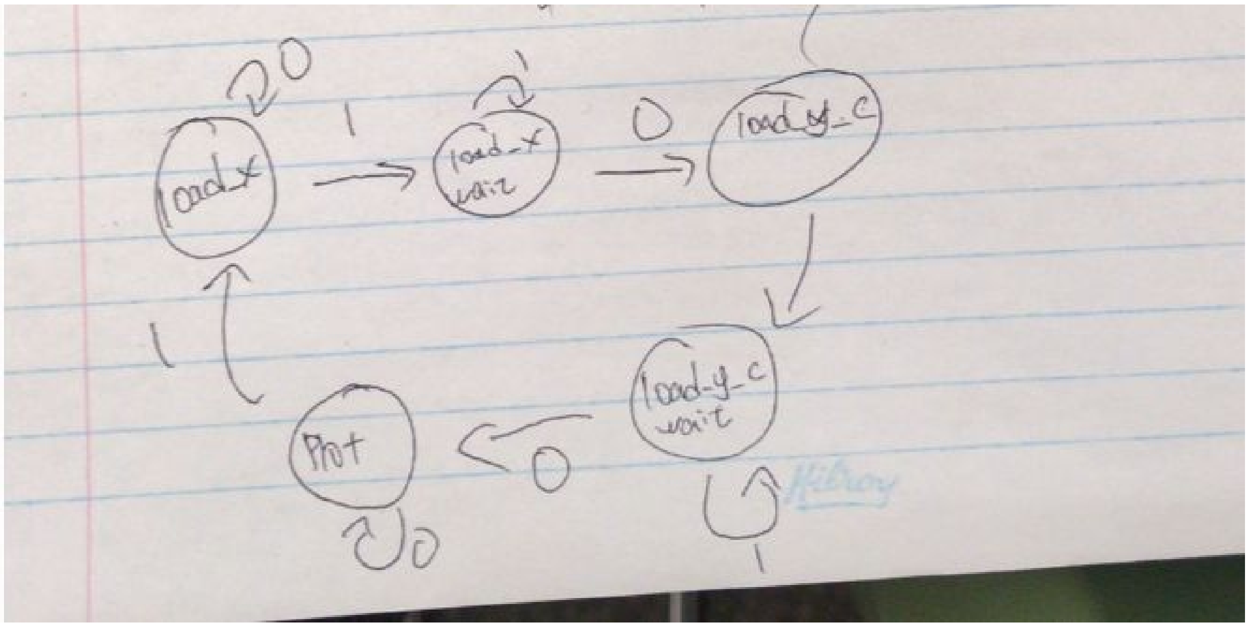
```
module control(
	input clk, resetn, load, go,
	output reg ld_x, ld_y, ld_color, writeEn);

	reg [2:0] current_state, next_state;

	localparam  LOAD_X = 3'b000,
				LOAD_X_WAIT = 3'b001,
				LOAD_Y_C = 3'b010,
				LOAD_Y_C_WAIT = 3'b011,
				PLOT = 3'b100;
	//reset
	always @(posedge clk) begin
```

```
                if (!resetn)
                        current_state <= LOAD_X;
                else
                        current_state <= next_state;
        end
        //state table
        always @(*)
        begin: state_table
                case (current_state)
                        LOAD_X: next_state = load ? LOAD_X_WAIT : LOAD_X;
                        LOAD_X_WAIT: next_state = load ? LOAD_X_WAIT : LOAD_Y_C;
                        LOAD_Y_C: next_state = go ? LOAD_Y_C_WAIT : LOAD_Y_C;
                        LOAD_Y_C_WAIT: next_state = go ? LOAD_Y_C_WAIT : PLOT;
                        PLOT: next_state = load ? LOAD_X : PLOT;
                        default: next_state = LOAD_X;
                endcase
        end

        always @(*)
        begin
                ld_x = 1'b0;
                ld_y = 1'b0;
                ld_color = 1'b0;
                writeEn = 0;
                case (current_state)
                        LOAD_X: ld_x = 1;
                        LOAD_X_WAIT: ld_x = 1;
                        LOAD_Y_C:
                                begin
                                        ld_x = 0;
                                        ld_y = 1;
                                        ld_color = 1;
                                end
                        LOAD_Y_C_WAIT:
                                begin
                                        ld_x = 0;
                                        ld_y = 1;
                                        ld_color = 1;
                                end
                        PLOT: writeEn = 1;
                endcase
        end
endmodule
```

PART 2 CONTROL STATE TABLE AND STATE DIAGRAM



Current State	Input	Next State
~~000~~ load_x	0	load_x
load_x	1	load_x wait
load_y_c	0	load_y_c
load_y-c	1	load_y_~~c~~wait
load_xwait	0	load-y-c
load_xwait	1	load_x_wait
load-y-cwait	0	PLOT
load-y-cwait	1	loady_cwait
PLOT	0	PLOT
PLOT	1	load_x

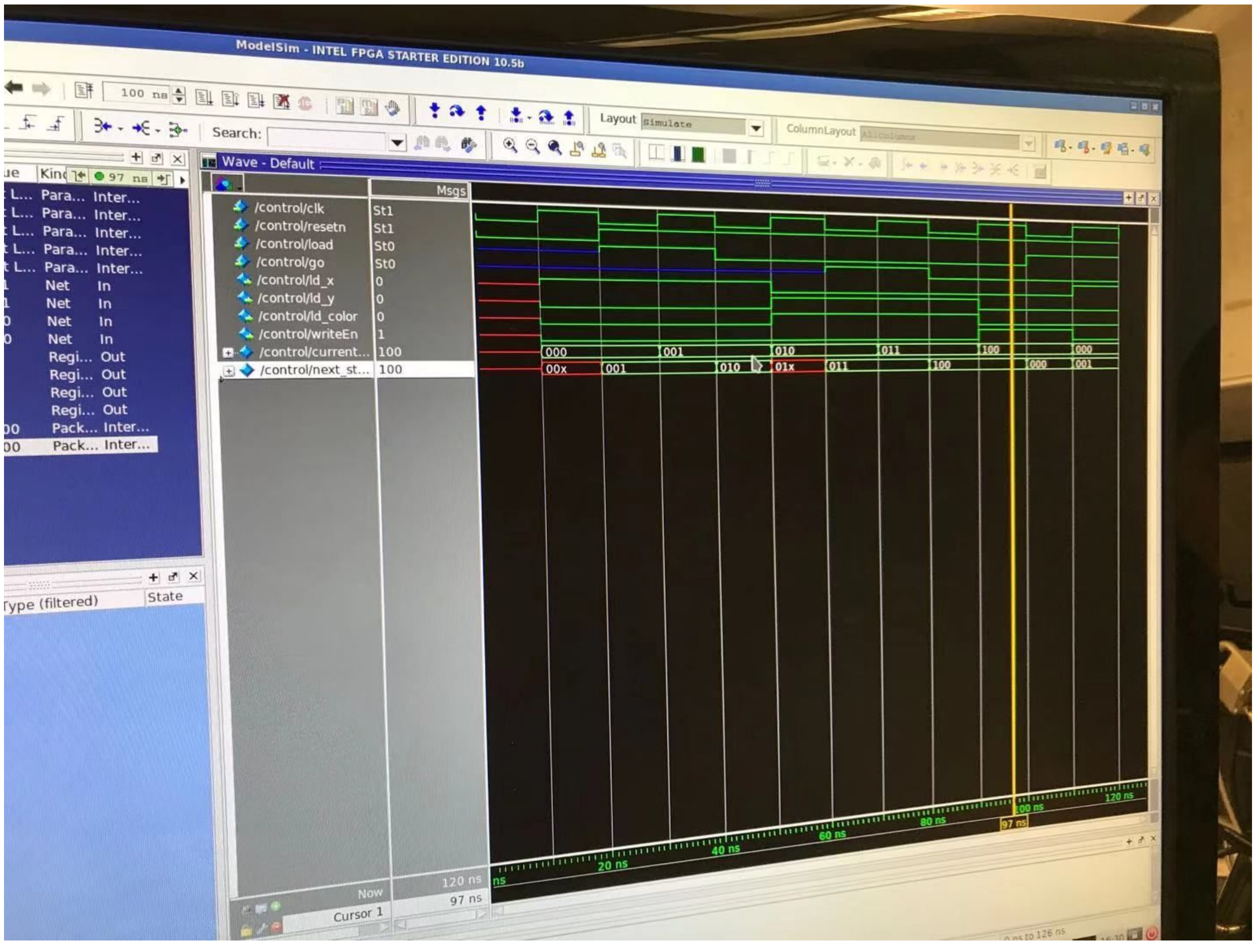


PART 2 CONTROL VSIM CODE

```
vlib work
vlog -timescale 1ns/1ns part2control.v
vsim control
log {/*}
```

```
add wave {/*}
force {clk} 0 0, 1 10 -r 20
force {resetn} 0
run 20ns
force {resetn} 1
force {load} 1
run 20ns
force {load} 0
run 20ns
force {go} 1
run 20ns
force {go} 0
run 20ns
force {load} 1
run 20ns
```

PART 2 CONTROL VSIM SCREENSHOT



```
                        Msgs
/control/clk            St1
/control/resetn         St1
/control/load           St0
/control/go             St0
/control/ld_x           0
/control/ld_y           0
/control/ld_color       0
/control/writeEn        1
/control/current...     100
/control/next_st...     100
```

PART 2 COMBINATION VERILOG

```
// Part 2 skeleton

module part2
	(
		CLOCK_50,						//	On Board 50 MHz
		// Your inputs and outputs here
   KEY,
   SW,
		// The ports below are for the VGA output.  Do not change.
		VGA_CLK,						//	VGA Clock
		VGA_HS,							//	VGA H_SYNC
		VGA_VS,							//	VGA V_SYNC
		VGA_BLANK_N,						//	VGA BLANK
		VGA_SYNC_N,						//	VGA SYNC
		VGA_R,						//	VGA Red[9:0]
		VGA_G,							//	VGA
Green[9:0]
		VGA_B						//	VGA Blue[9:0]
	);

	input			CLOCK_50;				//	50 MHz
	input   [9:0]   SW;
	input   [3:0]   KEY;

	// Declare your inputs and outputs here
	// Do not change the following outputs
	output			VGA_CLK;   				//	VGA Clock
	output			VGA_HS;					//	VGA H_SYNC
	output			VGA_VS;					//	VGA V_SYNC
	output			VGA_BLANK_N;				//	VGA BLANK
	output			VGA_SYNC_N;				//	VGA SYNC
	output	[9:0]	VGA_R;   				//	VGA Red[9:0]
	output	[9:0]	VGA_G;	 				//	VGA Green[9:0]
	output	[9:0]	VGA_B;   				//	VGA Blue[9:0]

	wire resetn;
	assign resetn = KEY[0];

	// Create the colour, x, y and writeEn wires that are inputs to the controller.
	wire [2:0] colour;
	wire [7:0] x;
	wire [6:0] y;
	wire writeEn;

	// Create an Instance of a VGA controller - there can be only one!
	// Define the number of colours as well as the initial background
	// image file (.MIF) for the controller.
	vga_adapter VGA(
			.resetn(resetn),
			.clock(CLOCK_50),
			.colour(colour),
			.x(x),
			.y(y),
			.plot(writeEn),
			/* Signals for the DAC to drive the monitor. */
			.VGA_R(VGA_R),
			.VGA_G(VGA_G),
			.VGA_B(VGA_B),
			.VGA_HS(VGA_HS),
			.VGA_VS(VGA_VS),
			.VGA_BLANK(VGA_BLANK_N),
			.VGA_SYNC(VGA_SYNC_N),
			.VGA_CLK(VGA_CLK));
		defparam VGA.RESOLUTION = "160x120";
		defparam VGA.MONOCHROME = "FALSE";
		defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
		defparam VGA.BACKGROUND_IMAGE = "black.mif";

	// Put your code here. Your code should produce signals x,y,colour and writeEn/plot
	// for the VGA controller, in addition to any other functionality your design may require.

    // Instansiate datapath
	// datapath d0(...);

    // Instansiate FSM control
    // control c0(...);

endmodule

module combination(
	input clk, resetn, load, go,
	input [2:0] color_in,
	input [6:0] coordinate,
		output [7:0] x_out,
		output [6:0] y_out,
		output [2:0] color_out
		);

		wire ld_x, ld_y, ld_color, enable;

		control c0(
			.clk(clk),
			.resetn(resetn),
			.load(load),
			.go(go),
			.ld_x(ld_x),
			.ld_y(ld_y),
			.ld_color(ld_color),
			.enable(enable)
		);

		datapath d0(
			.clk(clk),
			.color_in(color_in[2:0]),
			.resetn(resetn),
			.enable(enable),
			.ld_x(ld_x),
			.ld_y(ld_y),
			.ld_color(ld_color),
			.coordinate(coordinate[6:0]),
			.x_out(x_out),
			.y_out(y_out),
			.color_out(color_out)
		);

endmodule

module datapath(
		input clk, resetn, ld_x, ld_y, ld_color, enable,
		input [2:0] color_in,
		input [6:0] coordinate,
		output [7:0] x_out,
		output [6:0] y_out,
		output [2:0] color_out
		);
```

```
	reg [7:0] x;
	reg [6:0] y;
	reg [2:0] color;
	reg [3:0] counter;

	//reset or load
	always @(posedge clk) begin
		if (!resetn) begin
			x <= 8'b0;
			y <= 7'b0;
			color <= 3'b0;
		end
		else begin
			if (ld_x)
				x <= {1'b0, coordinate};
			if (ld_y)
				y <= coordinate;
			if (ld_color)
				color <= color_in;
		end
	end
	//counter
	always @(posedge clk) begin
		if (!resetn)
			counter <= 4'b0000;
		else if(enable)
			if (counter == 1111)
				counter <= 4'b0000;
			else
				counter <= counter + 1'b1;
	end

	assign x_out = x + counter[1:0];
	assign y_out = y + counter[3:2];
	assign color_out = color;
endmodule

module control(
	input clk, resetn, load, go,
	output reg ld_x, ld_y, ld_color, enable);

	reg [2:0] current_state, next_state;
```

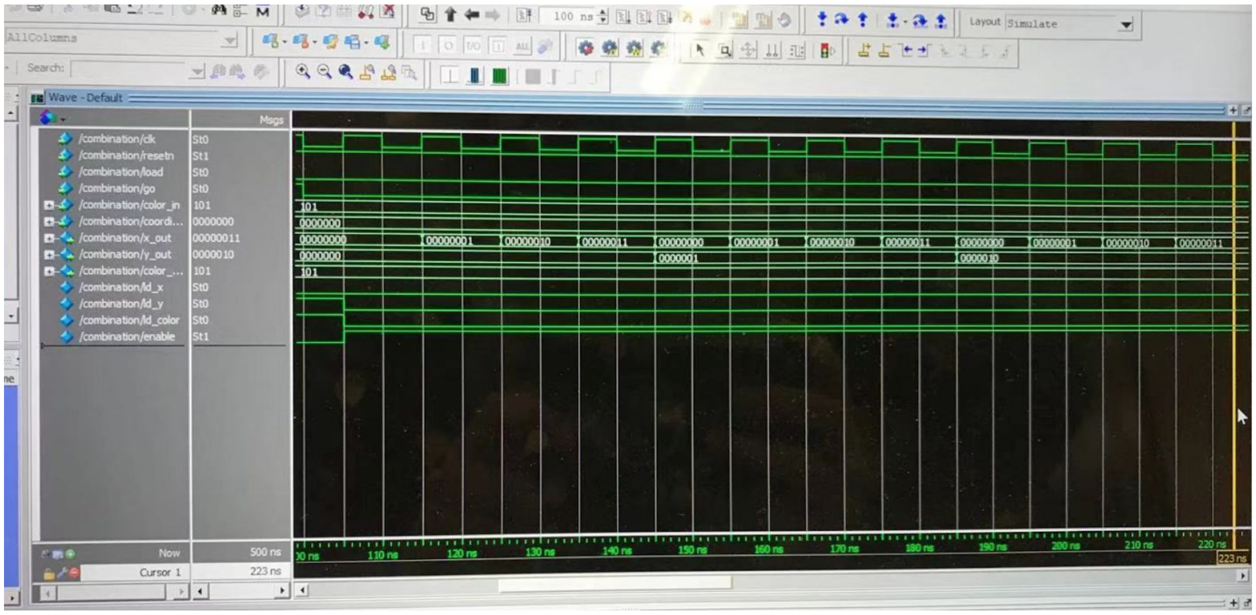
```
localparam  LOAD_X = 3'b000,
                        LOAD_X_WAIT = 3'b001,
                        LOAD_Y_C = 3'b010,
                        LOAD_Y_C_WAIT = 3'b011,
                        PLOT = 3'b100;
//reset
always @(posedge clk) begin
        if (!resetn)
                current_state <= LOAD_X;
        else
                current_state <= next_state;
end
//state table
always @(*)
begin: state_table
        case (current_state)
                LOAD_X: next_state = load ? LOAD_X_WAIT : LOAD_X;
                LOAD_X_WAIT: next_state = load ? LOAD_X_WAIT : LOAD_Y_C;
                LOAD_Y_C: next_state = go ? LOAD_Y_C_WAIT : LOAD_Y_C;
                LOAD_Y_C_WAIT: next_state = go ? LOAD_Y_C_WAIT : PLOT;
                PLOT: next_state = load ? LOAD_X : PLOT;
                default: next_state = LOAD_X;
        endcase
end

always @(*)
begin
        ld_x = 1'b0;
        ld_y = 1'b0;
        ld_color = 1'b0;
        enable = 0;
        case (current_state)
                LOAD_X: ld_x = 1;
                LOAD_X_WAIT: ld_x = 1;
                LOAD_Y_C:
                        begin
                                ld_x = 0;
                                ld_y = 1;
                                ld_color = 1;
                        end
                LOAD_Y_C_WAIT:
                        begin
                                ld_x = 0;
```

```
                                        ld_y = 1;
                                        ld_color = 1;
                                end
                        PLOT: enable = 1;
                endcase
        end
endmodule
```

PART 2 COMBINATION VSIM CODE

```
vlib work
vlog -timescale 1ns/1ns part2.v
vsim combination
log {/*}
add wave {/*}
force {clk} 0 0, 1 5 -r 10
force {resetn} 0
force {load} 0
force {go} 0
run 20ns
force {resetn} 1
force {color_in[2:0]} 101
run 20ns
force {coordinate[6:0]} 0000000
force {load} 1
run 20ns
force {load} 0
run 20ns
force {coordinate[6:0]} 0000000
force {go} 1
run 20ns
force {go} 0
#enable should be turn on so the x will ++
run 300ns
```

PART 2 COMBINATION VSIM SCREENSHOT



```
/combination/clk         St0
/combination/resetn      St1
/combination/load        St0
/combination/go          St0
/combination/color_in    101
/combination/coordi...   0000000
/combination/x_out       00000011
/combination/y_out       0000010
/combination/color_...   101
/combination/ld_x        St0
/combination/ld_y        St0
/combination/ld_color    St0
/combination/enable      St1
Now                      500 ns
Cursor 1                 223 ns
```

INLAB

```
module part2(
                CLOCK_50,           //On Board 50 MHz
                KEY,
                SW,
                VGA_CLK,            //VGA Clock
                VGA_HS,                     //VGA H_SYNC
                VGA_VS,                     //VGA V_SYNC
                VGA_BLANK_N,        //VGA BLANK
                VGA_SYNC_N,                 //VGA SYNC
                VGA_R,                      //VGA Red[9:0]
                VGA_G,                      //VGA Green[9:0]
                VGA_B                       //VGA Blue[9:0]
                );
        input                   CLOCK_50;           //      50 MHz
        input   [9:0]   SW;
        input   [3:0]   KEY;
        output                  VGA_CLK;            //      VGA Clock
        output                  VGA_HS;                     //      VGA H_SYNC
        output                  VGA_VS;                     //      VGA V_SYNC
        output                  VGA_BLANK_N;        //      VGA BLANK
        output                  VGA_SYNC_N;                 //      VGA SYNC
        output  [9:0]   VGA_R;                      //      VGA Red[9:0]
        output  [9:0]    VGA_G;                          //          VGA Green[9:0]
        output  [9:0]    VGA_B;                          //          VGA Blue[9:0]

        wire [2:0] colour;
        wire [7:0] x;
        wire [6:0] y;
        wire writeEn, ld_x, ld_y, ld_color;
        wire resetn;
        assign resetn = KEY[0];

        vga_adapter VGA(
                        .resetn(resetn),
                        .clock(CLOCK_50),
                        .colour(colour),
                        .x(x),
                        .y(y),
                        .plot(writeEn),
                        .VGA_R(VGA_R),
                        .VGA_G(VGA_G),
                        .VGA_B(VGA_B),
                        .VGA_HS(VGA_HS),
                        .VGA_VS(VGA_VS),
                        .VGA_BLANK(VGA_BLANK_N),
                        .VGA_SYNC(VGA_SYNC_N),
                        .VGA_CLK(VGA_CLK));
                defparam VGA.RESOLUTION = "160x120";
                defparam VGA.MONOCHROME = "FALSE";
                defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
                defparam VGA.BACKGROUND_IMAGE = "black.mif";

        datapath d0(
                .clk(CLOCK_50),
                .color_in(SW[9:7]),
                .resetn(resetn),
                .ld_x(ld_x),
                .ld_y(ld_y),
                .ld_color(ld_color),
                .coordinate(SW[6:0]),
                .x_out(x),
                .y_out(y),
                .color_out(colour)
        );
```

```
    control c0(
                    .clk(CLOCK_50),
                    .resetn(resetn),
                    .load(!(KEY[3])),
                    .go(!(KEY[1])),
                    .ld_x(ld_x),
                    .ld_y(ld_y),
                    .ld_color(ld_color),
                    .writeEn(writeEn)
            );
endmodule
```