ECE 271

Digital Logic Design
Final Project

Nick Olson
Michael ASD
Sienna ASD

November 30, 2019
Instructor Shuman
Oregon State University

Contents

List of Figures

1 Project Description

Intro to Project paragraph. The inputs and outputs of the overall design immediately follow. An overall description diagram is then shown in **Figure 1** and a hardware diagram is shown in **Figure 2**.

- **Inputs:** inputs
- **Outputs:** outputs

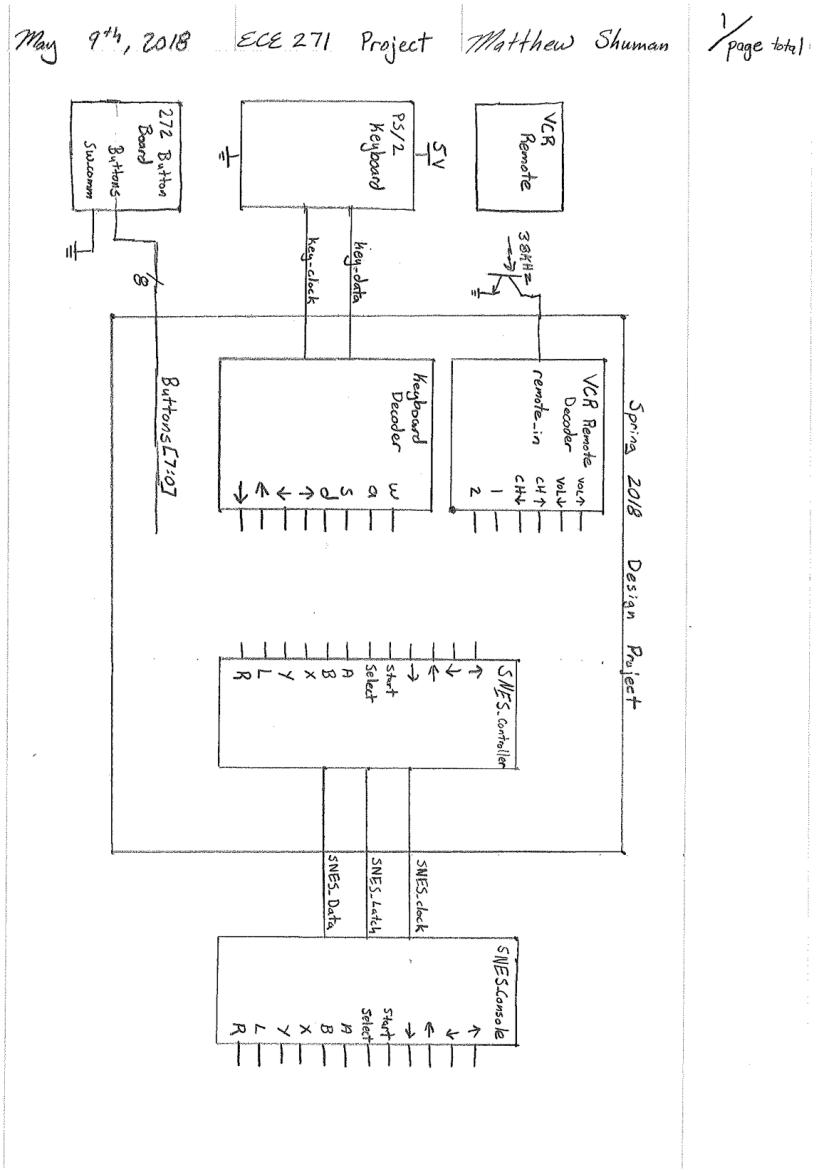


Figure 1: This image is legible, and conveys the point of the design. Your image can be hand drawn, but it must have straight lines, use your OSU ID. I don't recommend drafting this on the computer, because there aren't any decent tools to draw these block diagrams quickly.

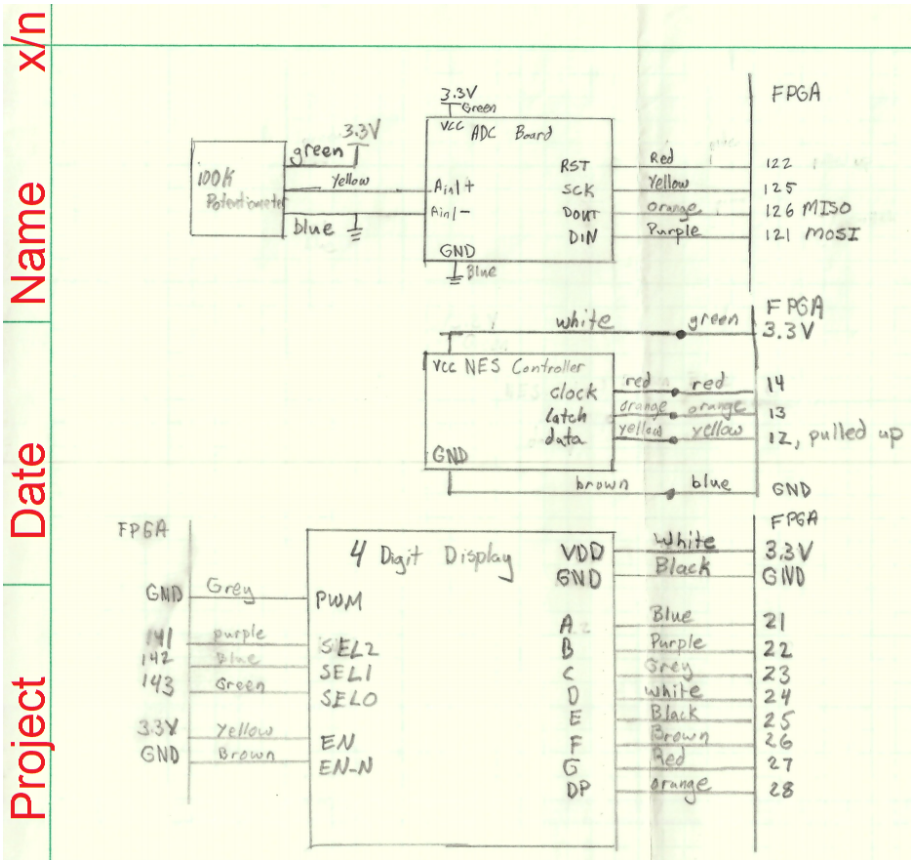


Figure 2: The hardware diagram shows which pins are used on the FPGA, module boards, and relevant supply voltages for the different pieces of hardware used in the system.

2 High Level Description

Top level introduction. The input and output specifications follow, a toplevel diagram follows in **Figure 3**, and the simulation results follow in **Figure 4**.

- **Inputs:** inputs
- **Outputs:** outputs

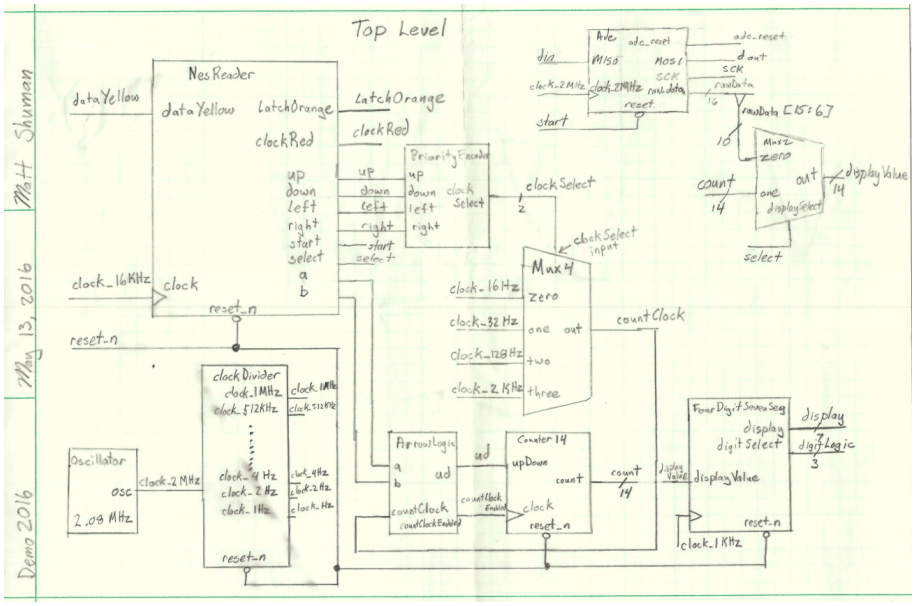


Figure 3: The top level design for the project. This would be improved by combining the priority encoder and Mux4 into a single clock select block. Combining the ArrowLogic and Counter14 would also make this diagram better. Use chapter 1 concepts wisely on this diagram, specifically hierarchy, modularity, regularity, and discipline.

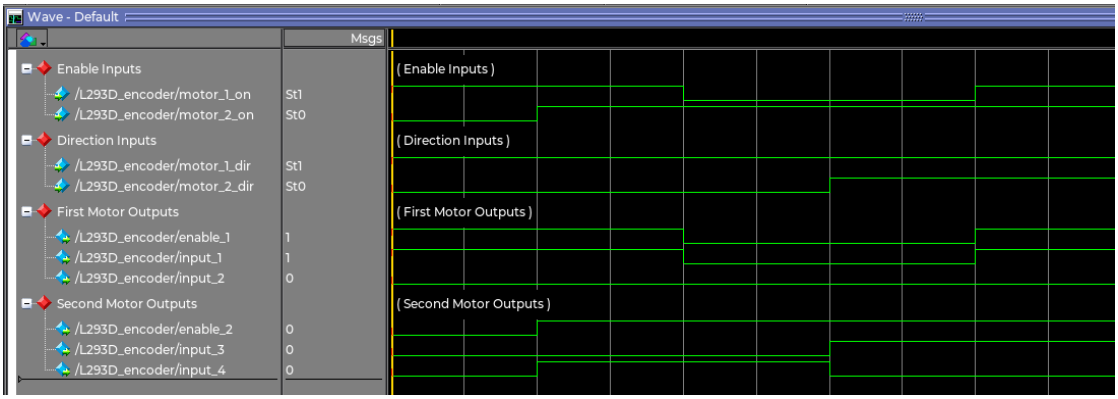


Figure 4: The simulation results of the top level design for the project.

The following subsections will discuss the inputs, outputs, designs, and simulation results of all elements of the design at two levels of scrutiny: functional units and individual blocks of digital logic.

2.1 Functional Unit 1 Name

Introduction to functional unit. The input and output specifications follow, a block diagram of the unit follows in **Figure A**, the simulation results for the unit follows in **Figure B**, and the details for each individual block comprising the unit follow after.

- **Inputs:** inputs
- **Outputs:** outputs

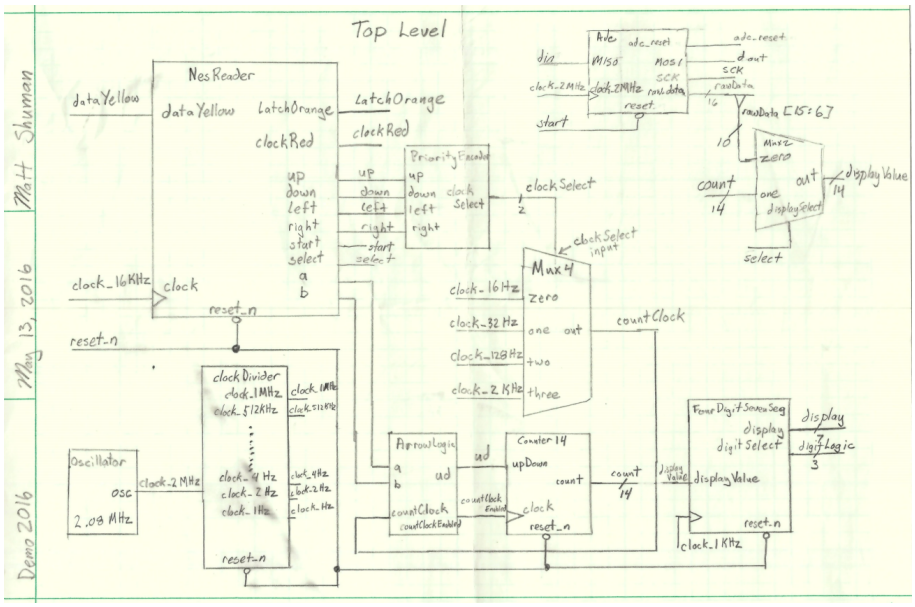


Figure 5: The logic design of the (NAME) functional unit used in the final design.

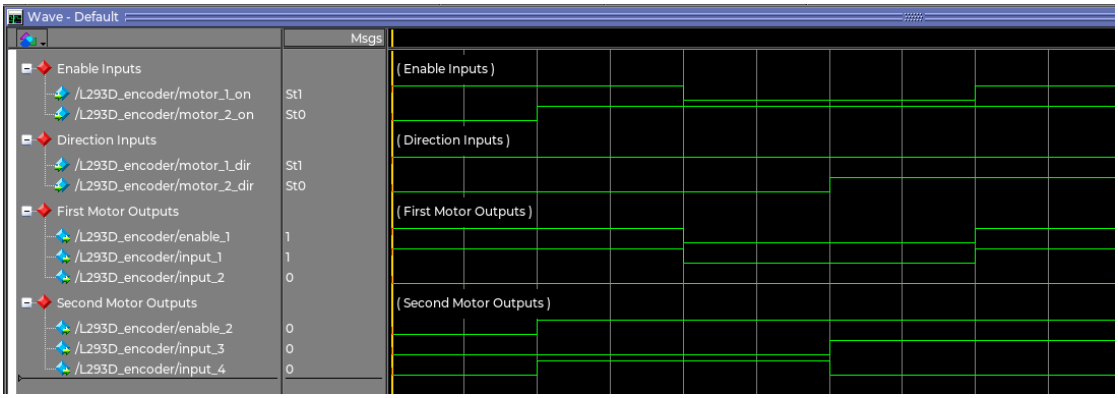


Figure 6: The simulation results of the top level design for the project.

2.2 Functional Unit 1 Name

Introduction to functional unit. The input and output specifications follow, a block diagram of the unit follows in **Figure A**, the simulation results for the unit follows in **Figure B**, and the details for each individual block comprising the unit follow after.

- **Inputs:** inputs
- **Outputs:** outputs

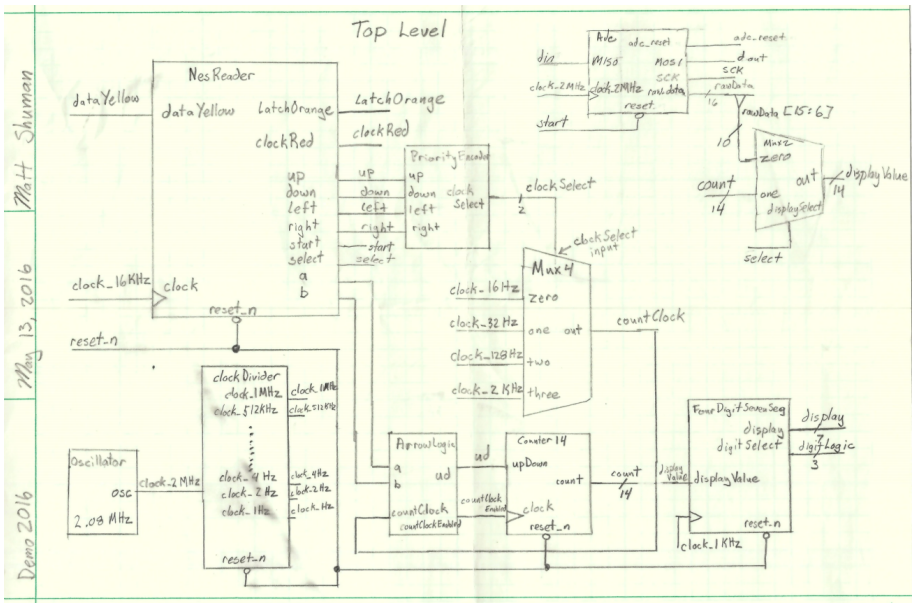


Figure 7: The logic design of the (NAME) functional unit used in the final design.

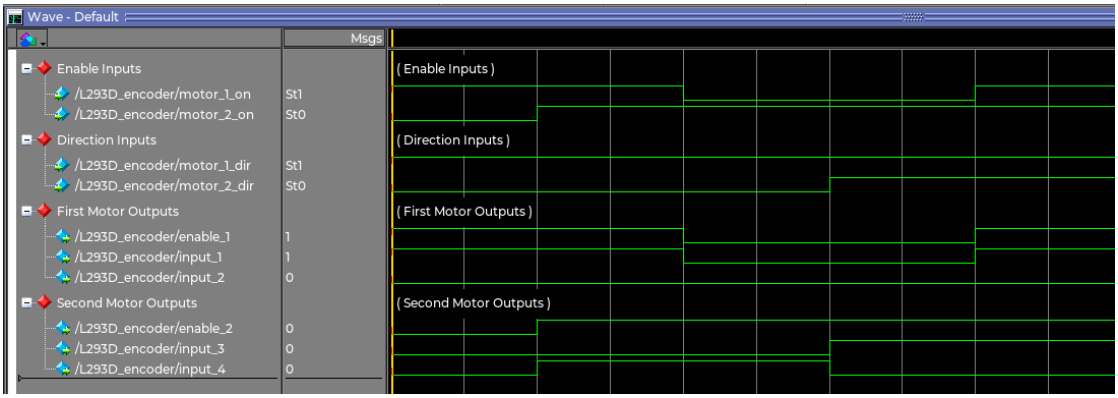
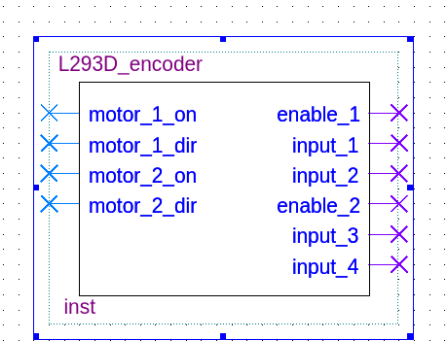


Figure 8: The simulation results of the top level design for the project.

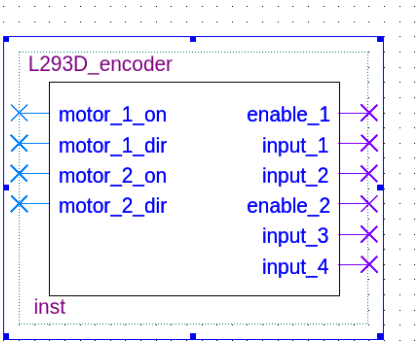
2.2.1 Individual Block 1 Name (with testbench)

Introduction to individual block. The input and output specifications follow, as well as the block diagram (**Figure C**), and simulation results **Figure D** for the individual block.

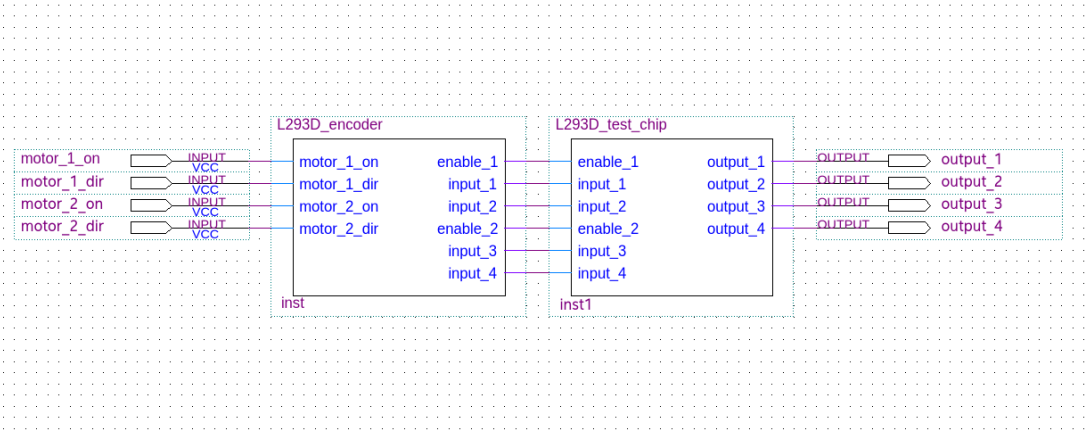
- **Inputs:** inputs
- **Outputs:** outputs



(a) The block symbol of the (NAME) individual block used in the (NAME) functional unit.

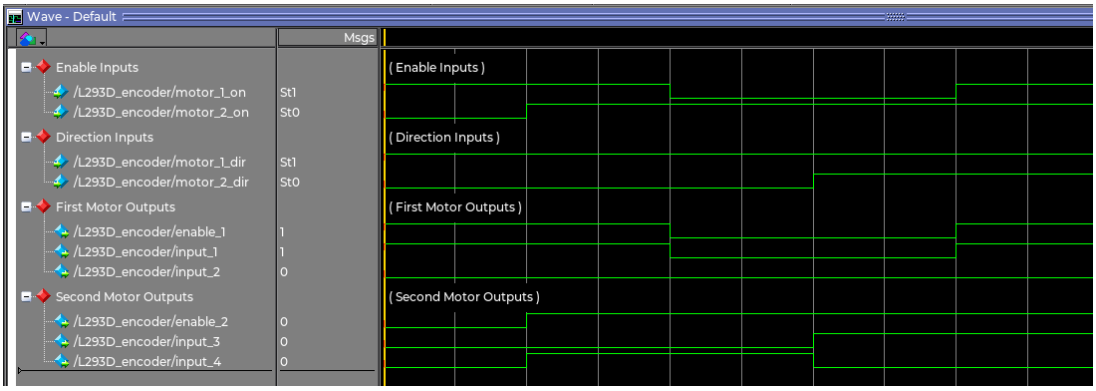


(b) The block symbol of the test block used in the testbench made to further simulate the (NAME) individual block.

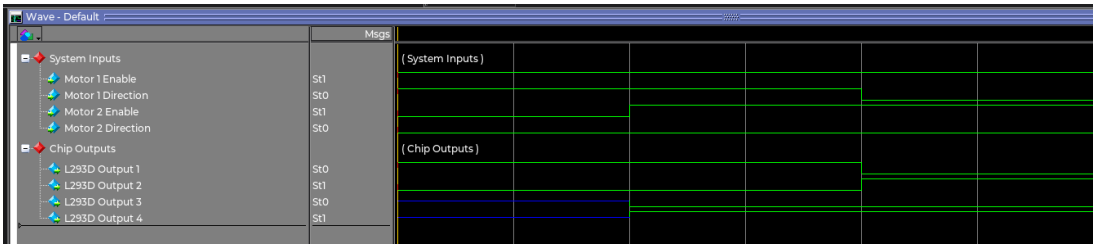


(c) The testbench used to further simulate the (NAME) individual block.

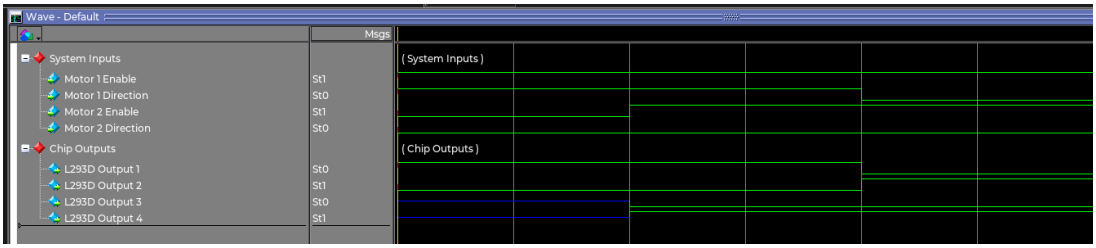
Figure 9: The block symbol of the (NAME) individual block used in the (NAME) functional unit, the block diagram of the logic of the testbench used to further simulate the block, and the block symbol of the test chip used in the testbench of the block.



(a) The simulation results for the (NAME) individual block alone.



(b) The simulation results of the test block used in the testbench.



(c) The simulation results of the testbench used.

Figure 10: The simulation results of the block alone, the testbench used to further simulate the block, and the simulation results for the testbebch of the (NAME) individual block used in the (NAME) functional unit.

2.2.2 Individual Block 2 Name (without testbench)

Introduction to individual block. The input and output specifications follow, as well as the block diagram (**Figure C**), and simulation results **Figure D** for the individual block.

- **Inputs:** inputs
- **Outputs:** outputs

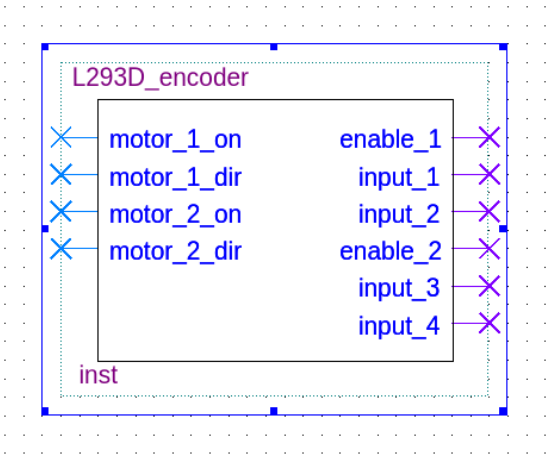


Figure 11: The block symbol of the (NAME) individual block used in the (NAME) functional unit.

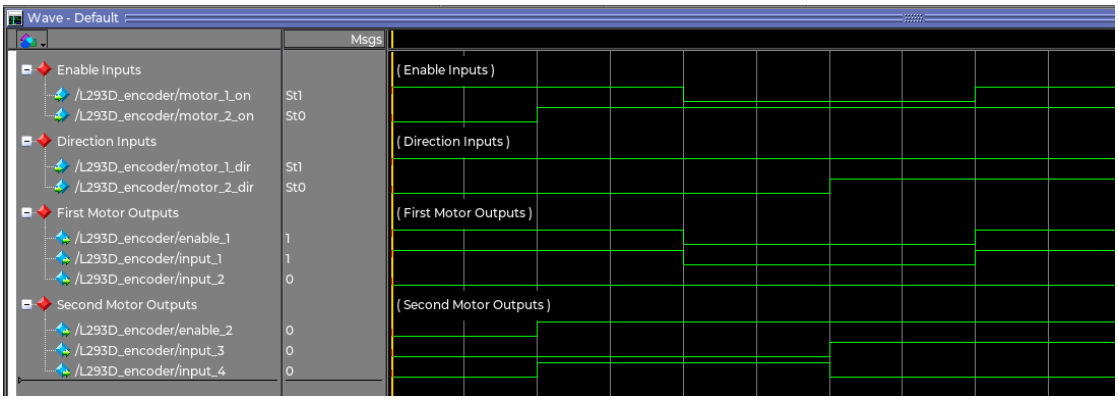


Figure 12: The simulation results of the (NAME) individual block used in the (NAME) functional unit.

2.3 DisplayDecoder Functional Unit

The DisplayDecoder Module converts a 4-bit input value into a display value on the seven segment display of the FPGA. The DisplayDecoder is able to output a hexadecimal display value between 0 and F. A block diagram of the unit follows in **Figure A** and the simulation results for the unit follows in **Figure B**.

- **Inputs:** The DisplayDecoder module takes a 4-bit binary value input, data, as its only input.
- **Outputs:** The DisplayDecoder module outputs a 7-bit binary value that is used to activate specific segments in the FPGA seven segment display.

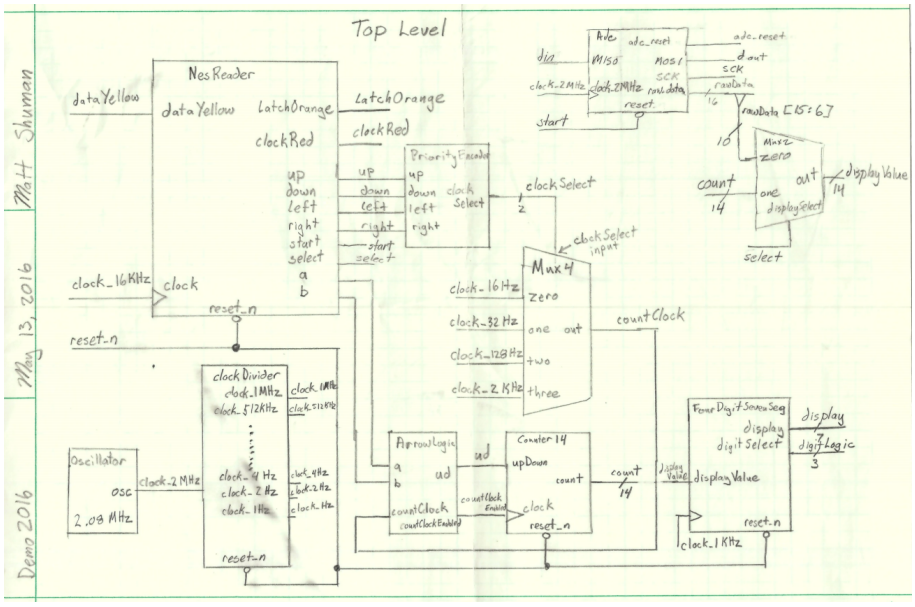


Figure 13: The logic design of the DisplayDecoder functional unit used in the final design.

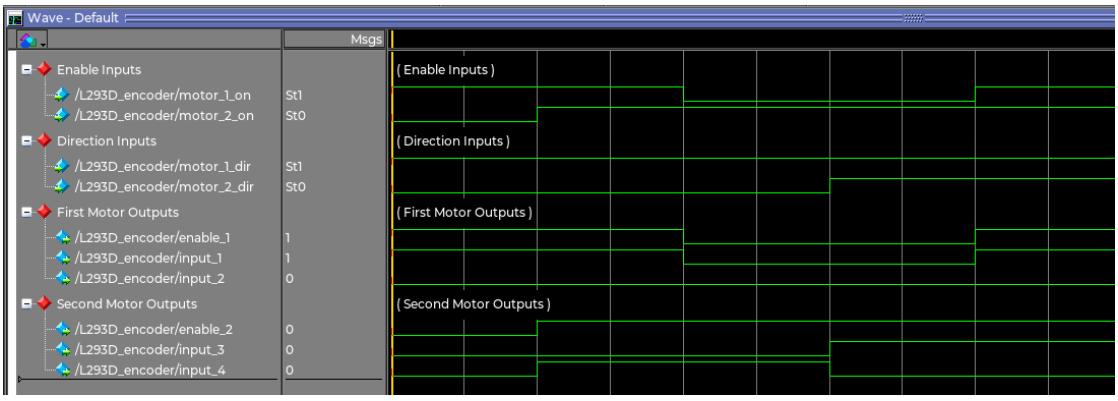


Figure 14: The simulation results for the DisplayDecoder Module.

2.4 vcr_decoder Functional Unit

The vcr_decoder module converts an IR signal sent from a VCR remote into a decimal value between 0 and 9. A block diagram of the unit follows in **Figure A**, the simulation results for the unit follows in **Figure B**, and the details for each individual block comprising the unit follow after.

- **Inputs:** The vcr_decoder module two inputs, clk and IR. clk is a 10 KHz clock signal that is used to drive the module. IR is the Infrared signal coming from the VCR remote that will be translated by the module.
- **Outputs:** The vcr_decoder module has a single output, displayValue, which is the 0-9 representing the IR signal that was received by the module as input.

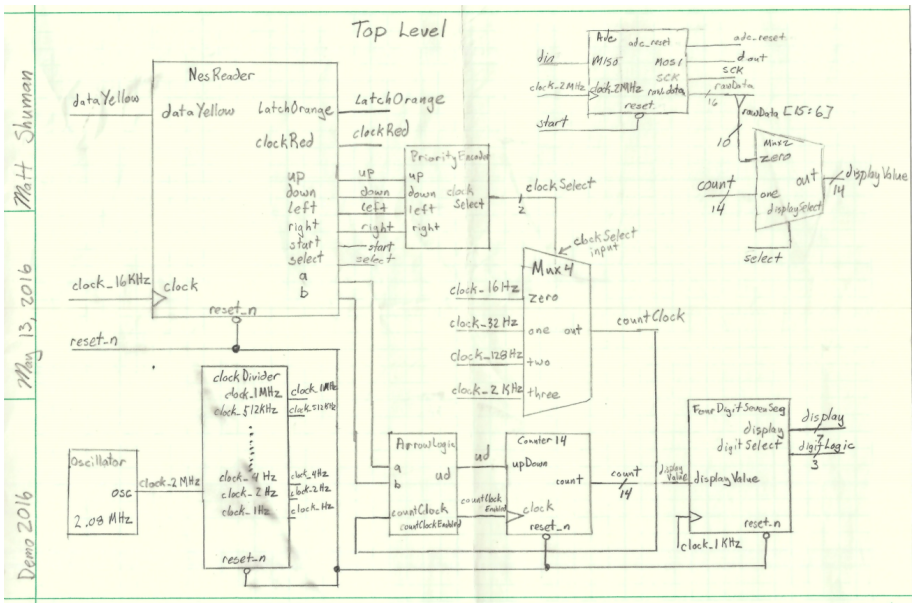


Figure 15: The logic design of the vcr_decoder functional unit used in the final design.

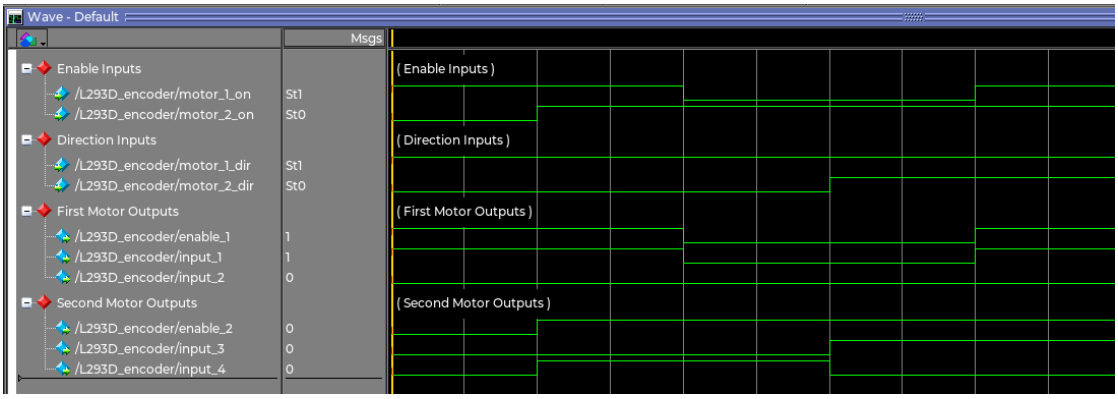


Figure 16: The simulation results of the vcr_decoder Functional Unit.

2.4.1 ReadState Module

Introduction to individual block. The input and output specifications follow, as well as the block diagram (**Figure C**), and simulation results **Figure D** for the individual block.

- **Inputs:** inputs
- **Outputs:** outputs

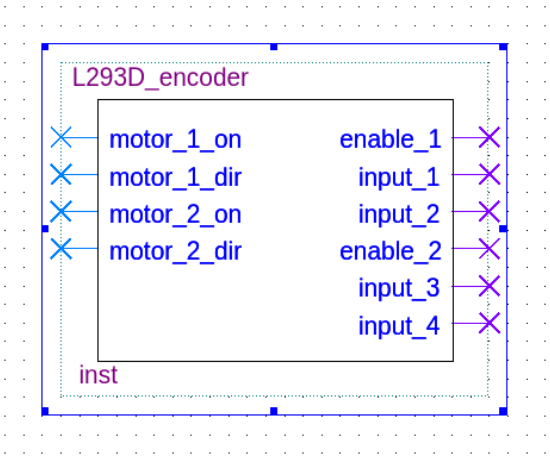


Figure 17: The block symbol of the (NAME) individual block used in the (NAME) functional unit.

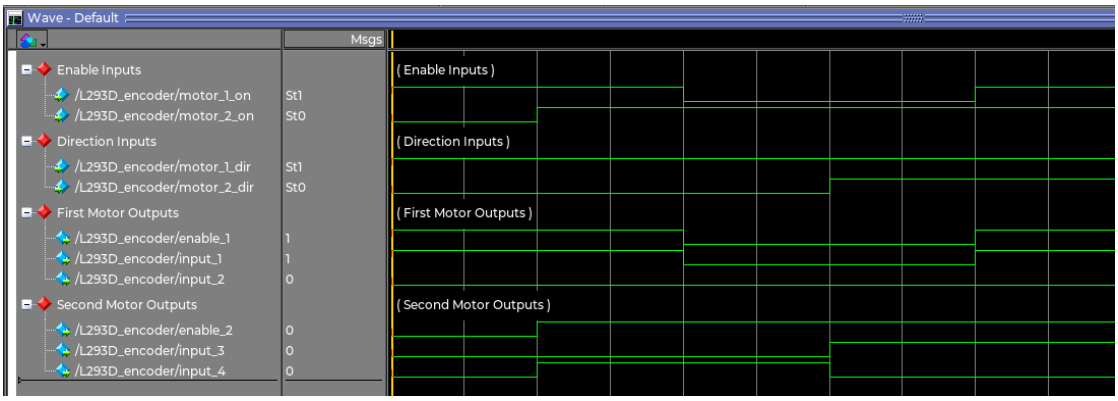


Figure 18: The simulation results of the (NAME) individual block used in the (NAME) functional unit.

A SystemVerilog Files

This appendix will list the SystemVerilog code used for each block used in the design project.

A.1 seven$_seg_6$

A.1.1 Component Blocks of seven$_seg_6Unit$

```
module sevenseg(input   logic [3:0] data,
                output  logic [6:0] segments);

   always_comb
      case(data)
         //                       gfe_dcba
         0:       segments =  7'b100_0000;
         1:       segments =  7'b111_1001;
         2:       segments =  7'b010_0100;
         3:       segments =  7'b011_0000;
         4:       segments =  7'b001_1001;
         5:       segments =  7'b001_0010;
         6:       segments =  7'b000_0010;
         7:       segments =  7'b111_1000;
         8:       segments =  7'b000_0000;
         9:       segments =  7'b001_1000;
         default: segments =  7'b111_1111;
      endcase

endmodule
```

(a) SystemVerilog code for theseven$_seg_6block used in the design$.

A.2 L293D$_{e}ncoder$

```
1
2  /* Takes input as 4-bit register describing whether
3     each motor is on and its direction */
4
5  module L293D_encoder (
6     input logic     motor_1_on,     // whether the first motor should spin
7                     motor_1_dir,    // the direction of spin (0 is back, 1 is forward)
8
9                     motor_2_on,     // whether the second motor should spin
10                    motor_2_dir,    // the direction of spin (0 is back, 1 is forward)
11
12    output logic    enable_1,    // pin 1    -  enable for inputs 1 and 2
13                    input_1,     // pin 2    -  input 1 (forward for motor 1)
14                    input_2,     // pin 7    -  input 2 (backward for motor 1)
15
16                    enable_2,    // pin 9    -  enable for inputs 3 and 4
17                    input_3,     // pin 10   -  input 3 (forward for motor 2)
18                    input_4      // pin 15   -  input 4 (backward for motor 2)
19    );
20
21    always_comb begin
22       enable_1     <=  motor_1_on;
23       input_1      <= (motor_1_on &&  (motor_1_dir));
24       input_2      <= (motor_1_on && ~(motor_1_dir));
25
26       enable_2     <=  motor_2_on;
27       input_3      <= (motor_2_on &&  (motor_2_dir));
28       input_4      <= (motor_2_on && ~(motor_2_dir));
29    end
30
31 endmodule
```

(a) SystemVerilog code for the$L293D_{e}ncoderblockusedinthedesign$.

A.3 $\mathbf{nes}_d ecoder$

```
1
2  module nes_decoder (input  logic     nes_data,    // the current button data coming from cont
3                                       in_clock,    // will synchronize output to the positive
4                                       read_data,   // will read the data on the positive edge
5                                       reset,       // does nothing so far
6
7                      output logic     nes_latch,   // sent to controller, begins process of re
8                                       nes_clock,   // sent to controller, tells controller to
9
10                     output logic     nes_A,       // A button output, active low
11                                      nes_B,       // B button output, active low
12                                      nes_START,   // START button output, active low
13                                      nes_SELECT,  // SELECT button output, active low
14                                      nes_UP,      // UP button output, active low
15                                      nes_DOWN,    // DOWN button output, active low
16                                      nes_LEFT,    // LEFT button output, active low
17                                      nes_RIGHT,   // RIGHT button output, active low
18
19                     output logic     ready_to_read); // output telling system whether the mo
20
21
22       // Parameter
23       localparam                 NUM_BUTTONS = 8;
24
25       // Internal
26       logic  [3:0]               count = 3'b0000;
27       logic                      pause = 0;
28       logic                      next = 0;
29       logic                      apply = 0;
30       logic  [NUM_BUTTONS-1:0]   tmp_buttons = {NUM_BUTTONS{1'b1}};
31       logic  [NUM_BUTTONS-1:0]   prev_input  = {NUM_BUTTONS{1'b1}};
32       logic                      tmp_ready   = 0;
33
34       // Start reading process if latch, read until all buttons read, apply after
35       always_ff @(posedge in_clock, posedge read_data, posedge reset) begin
36
37             if (reset) begin
38                /* On reset, set to default values */
39
40                   tmp_ready      = 1;
41                   nes_latch      = 0;
42                   nes_clock      = 0;
43                   tmp_buttons    = {NUM_BUTTONS{1'b1}};
44                   prev_input     = {NUM_BUTTONS{1'b1}};
45                   next           = 0;
46                   apply          = 0;
47                   count          = 0;
48                   tmp_ready      = 1;
49
50             end
51             else if (read_data) begin
52                /* On start (read_data) if the system is ready to read (ready_to_read),
53                 * send nes_latch signal to controller to start reading process, signal syst
54                 * is no longer ready to read, and set that it should read the next value. *
55
56                   if (tmp_ready) begin
57                      nes_latch      =  1;
58                      tmp_ready      =  0;
59                      count          =  0;
60                      tmp_buttons    =  {NUM_BUTTONS{1'b1}};
61                      next           =  1;
62                   end
63
64             end
65             else if (pause) begin
66                /* In between each read (after each [next] case where a button is read),
67                 * either send the system to the apply state or to the next button read
68                 * depending on if we have read all buttons; gives nes_clock time to be 0. *
69
```

A.4 parsed$_{c}lock$

A.4.1 Component Blocks of parsed$_{c}lockUnit$

```
module parser #(parameter IN_BITS=30, TICK_PER_SEC=9537)
               (input logic [IN_BITS-1:0] count,
                output logic [IN_BITS-1:0] total_seconds,
                output logic [3:0] seconds_ones,
                output logic [3:0] seconds_tens,
                output logic [3:0] minutes_ones,
                output logic [3:0] minutes_tens,
                output logic [3:0] hours_ones,
                output logic [3:0] hours_tens);

   reg   [5:0]          seconds;
   reg   [5:0]          minutes;
   reg   [4:0]          hours;

   always_comb
      begin
         total_seconds = count / TICK_PER_SEC;

         seconds = total_seconds % 60;
         seconds_ones = seconds % 10;
         seconds_tens = seconds / 10;

         minutes = (total_seconds / 60) % 60;
         minutes_ones = minutes % 10;
         minutes_tens = minutes / 10;

         hours = (total_seconds / 3600) % 24;
         hours_ones = hours % 10;
         hours_tens = hours / 10;
      end

endmodule
```

(a) SystemVerilog code for theparsed$_{c}lockblockusedinthedesign$.

```
module comparator #(parameter N=30)
                    (input  logic [N-1:0] a, b,
                     output logic eq, neq, lt, lte, gt, gte, agt24h);

   assign eq    =  (a == b);
   assign neq   =  (a != b);
   assign lt    =  (a < b);
   assign lte   =  (a <= b);
   assign gt    =  (a > b);
   assign gte   =  (a >= b);

   assign agt24h =   (a >= 24*3600);

endmodule
```

(a) SystemVerilog code for theparsed_c*lockblockusedinthedesign*.

```
// mux4 module

module mux4 #(parameter IN_WIDTH = 1)                    // Width of each input option

              (input  logic    [3:0] [IN_WIDTH-1:0] in,
// 4, 4-bit inputs (select 00, 01, 10, 11 respectively)
               input  logic  [1:0] sel,
// input sel that selects between inputs

               output logic    [IN_WIDTH-1:0] out);       // 4-bit output based on input sel

   // Whenever an input or select changes, reassign output
   always @ (in or sel) begin

         out <= in[sel];

   end

endmodule
```

(a) SystemVerilog code for theparsed_c*lockblockusedinthedesign*.

```
module counter #(parameter N=30)
                (input  logic clk, reset, enable,
                 output logic [N-1:0]  q);

   always_ff@(posedge clk, posedge reset)
      begin
         if(reset)          q <= 0;
         else if(enable)    q <= q+1;
      end

endmodule
```

(a) SystemVerilog code for theparsed_c*lockblockusedinthedesign*.

A.5 $\mathbf{nes}_{t}o_{m}otor$

```
1
2  module nes_to_motor (input logic    forward,
3                                      backward,
4                                      left,
5                                      right,
6
7                       output logic   motor_1_on,
8                                      motor_1_dir,
9                                      motor_2_on,
10                                     motor_2_dir);
11
12     // Internal (NO DOUBLE INPUTS)
13     logic int_forward    = 0;
14     logic int_backward   = 0;
15     logic int_left       = 0;
16     logic int_right      = 0;
17
18     always_comb begin
19
20        int_forward    = forward   && (forward ^ backward ^ left ^ right);
21        int_backward   = backward  && (forward ^ backward ^ left ^ right);
22        int_left       = left      && (forward ^ backward ^ left ^ right);
23        int_right      = right     && (forward ^ backward ^ left ^ right);
24
25        motor_1_on  =  int_left || int_right || int_forward || int_backward;
26        motor_1_dir =  int_backward || int_left;
27
28        motor_2_on  =  int_left || int_right || int_forward || int_backward;
29        motor_2_dir =  int_forward || int_left;
30
31     end
32
33 endmodule
```

(a) SystemVerilog code for the$\mathbf{nes}_{t}o_{m}otor$*blockusedinthedesign.*

B Simulation Files (Do Scripts)

This appendix will list the Do Scripts used to simulate each block used in the design project.

B.1 seven$_seg_6$

```
vsim design-project.seven_seg_6
add wave -position insertpoint sim:/seven_seg_6/*

force in0 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200
force in1 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200
force in2 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200
force in3 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200
force in4 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200
force in5 10#0 25, 10#1 50, 10#2 75, 10#3 100, 10#4 125, 10#5 150, 10#6 175, 10#7 200

run 275
```

(a) Do Script code for the seven$_seg_6$ *block used in the design.*

B.1.1 Component Blocks of seven$_seg_6Unit$

```
vsim design-project.sevenseg
add wave -position insertpoint sim:/sevenseg/*

force -freeze sim:/sevenseg/data 0000 0
run 50

force -freeze sim:/sevenseg/data 0001 0
run 50

force -freeze sim:/sevenseg/data 0010 0
run 50

force -freeze sim:/sevenseg/data 0011 0
run 50

force -freeze sim:/sevenseg/data 0100 0
run 50

force -freeze sim:/sevenseg/data 0101 0
run 50

force -freeze sim:/sevenseg/data 0110 0
run 50

force -freeze sim:/sevenseg/data 0111 0
run 50

force -freeze sim:/sevenseg/data 1000 0
run 50

force -freeze sim:/sevenseg/data 1001 0
run 50
```

(a) Do Script code for theseven$_seg_6blockusedinthedesign$.

B.2 L293D$_encoder$

```
vsim design-project.L293D_encoder

add wave -position insertpoint sim:/L293D_encoder/motor_1_on
add wave -position insertpoint sim:/L293D_encoder/motor_2_on

add wave -position insertpoint sim:/L293D_encoder/motor_1_dir
add wave -position insertpoint sim:/L293D_encoder/motor_2_dir

add wave -position insertpoint sim:/L293D_encoder/enable_1
add wave -position insertpoint sim:/L293D_encoder/input_1
add wave -position insertpoint sim:/L293D_encoder/input_2

add wave -position insertpoint sim:/L293D_encoder/enable_2
add wave -position insertpoint sim:/L293D_encoder/input_3
add wave -position insertpoint sim:/L293D_encoder/input_4

force -freeze motor_1_on     0 0, 1 100, 0 {300}
force -freeze motor_2_on     0 0, 1 100, 0 {300}
force -freeze motor_1_dir    0 0, 1 {200}
force -freeze motor_2_dir    0 0, 1 {200}

run 400
```

(a) Do Script code for the$L293D_{e}ncoderblockusedinthedesign.$

B.3 nes$_d$*ecoder*

```
vsim design-project.nes_decoder

add wave -position insertpoint sim:/nes_decoder/in_clock
add wave -position insertpoint sim:/nes_decoder/read_data
add wave -position insertpoint sim:/nes_decoder/reset
add wave -position insertpoint sim:/nes_decoder/nes_*
add wave -position insertpoint sim:/nes_decoder/ready_to_read
add wave -position insertpoint sim:/nes_decoder/count
add wave -position insertpoint sim:/nes_decoder/tmp_buttons
add wave -position insertpoint sim:/nes_decoder/prev_input
add wave -position insertpoint sim:/nes_decoder/pause
add wave -position insertpoint sim:/nes_decoder/next
add wave -position insertpoint sim:/nes_decoder/apply
add wave -position insertpoint sim:/nes_decoder/tmp_ready

force -freeze   in_clock    1 0,    0 {25}    -r 50
force -freeze   nes_data    0 0,    1 250     -r 500
force -freeze   read_data   0 0,    1 10,     0 {20}
force -freeze   reset       1 0,    0 {10}

run 900
```

(a) Do Script code for thenes$_d$*ecoderblockusedinthedesign.*

B.4 parsed$_{c}$*lock*

```
vsim design-project.parsed_clock

add wave -position insertpoint sim:/parsed_clock/clock_50MHz
add wave -position insertpoint sim:/parsed_clock/enable_in
add wave -position insertpoint sim:/parsed_clock/reset_in
add wave -position insertpoint sim:/parsed_clock/double_speed_in
add wave -position insertpoint sim:/parsed_clock/super_speed_in

add wave -position insertpoint sim:/parsed_clock/total_sec

add wave -position insertpoint sim:/parsed_clock/seconds_ones
add wave -position insertpoint sim:/parsed_clock/seconds_tens
add wave -position insertpoint sim:/parsed_clock/minutes_ones
add wave -position insertpoint sim:/parsed_clock/minutes_tens
add wave -position insertpoint sim:/parsed_clock/hours_ones
add wave -position insertpoint sim:/parsed_clock/hours_tens

force -freeze clock_50MHz    0 0,          1  1us       -r {2us}
force -freeze enable_in      0 0,          1  1us,       0 {2us}
force -freeze reset_in       1 0,          0 {1us}

force -freeze double_speed_in  1 0, 0 50000000us, 1 75000000us, 0 {100000000us}
force -freeze super_speed_in   1 0, 0 75000000us

run 150000000us
```

(a) Do Script code for theparsed$_{c}$*lockblockusedinthedesign.*

B.4.1 Component Blocks of parsed$_c lockUnit$

```
vsim design-project.counter
add wave -position insertpoint sim:/counter/*
force -freeze sim:/counter/clk 0 0, 1 {25 ps} -r 50
force -freeze sim:/counter/reset 1 0 -cancel 50
force -freeze sim:/counter/enable 1 0 -cancel 100
run 100
force -freeze sim:/counter/enable 0 0 -cancel 50
run 50
force -freeze sim:/counter/enable 1 0
run 150
```

(a) Do Script code for theparsed$_c lockblockusedinthedesign$.

```
vsim design-project.parser
add wave -position insertpoint sim:/parser/*

force -freeze sim:/parser/count 0 0
run 50

force -freeze sim:/parser/count 1010001011111100011 0
run 50

force -freeze sim:/parser/count 10101110101000001011000 0
run 50

force -freeze sim:/parser/count 10000010111110001000010000 0
run 50
```

(a) Do Script code for theparsed$_c lockblockusedinthedesign$.

```
vsim design-project.enable_flip_flop

add wave -position insertpoint sim:/enable_flip_flop/enable_in
add wave -position insertpoint sim:/enable_flip_flop/reset
add wave -position insertpoint sim:/enable_flip_flop/enable_out

force -freeze enable_in      0 0,     1  50     -r {100}
force -freeze reset          1 0,     0 {10}

run 400
```

(a) Do Script code for theparsed$_c lockblockusedinthedesign$.

B.5 nes$_t o_m otor$

```
vsim design-project.delay
add wave -position insertpoint sim:/delay/clock_in
add wave -position insertpoint sim:/delay/enable
add wave -position insertpoint sim:/delay/reset
add wave -position insertpoint sim:/delay/double_speed
add wave -position insertpoint sim:/delay/super_speed
add wave -position insertpoint sim:/delay/select
add wave -position insertpoint sim:/delay/clock_option
add wave -position insertpoint sim:/delay/clock_out

force -freeze sim:/delay/clock_in 0 0, 1 {50 ps} -r 100
force -freeze sim:/delay/enable 1 0
force -freeze sim:/delay/reset 1 0 -cancel 200
force -freeze sim:/delay/double_speed 1 0, 0 25000, 1 50000, 0 {100000}
force -freeze sim:/delay/super_speed  1 0, 0 50000

run 200000
```

(a) Do Script code for theparsed_c*lockblockusedinthedesign*.

```
vsim design-project.mux4
add wave -position insertpoint sim:/mux4/*


force -freeze {in[0]} 0000 {0}
force -freeze {in[1]} 0011 {0}
force -freeze {in[2]} 1100 {0}
force -freeze {in[3]} 1111 {0}

force -freeze sel 00 0, 01 50, 10 100, 11 {150}

run 200
```

(a) Do Script code for theparsed_c*lockblockusedinthedesign*.

```
vsim design-project.comparator
add wave -position insertpoint sim:/comparator/*
force -freeze sim:/comparator/a 0 0
run 100
force -freeze sim:/comparator/a 0010101000110000000 0
run 100
force -freeze sim:/comparator/a 0010101000101111111 0
run 100
```

(a) Do Script code for theparsed_c*lockblockusedinthedesign*.

```
vsim design-project.nes_to_motor
add wave -position insertpoint sim:/nes_to_motor/forward
add wave -position insertpoint sim:/nes_to_motor/backward
add wave -position insertpoint sim:/nes_to_motor/left
add wave -position insertpoint sim:/nes_to_motor/right
add wave -position insertpoint sim:/nes_to_motor/motor_1_on
add wave -position insertpoint sim:/nes_to_motor/motor_1_dir
add wave -position insertpoint sim:/nes_to_motor/motor_2_on
add wave -position insertpoint sim:/nes_to_motor/motor_2_dir

force -freeze   forward     1 0, 0 30,          1 {200}
force -freeze   backward    0 0, 1 50,  0 80,   1 {200}
force -freeze   left        0 0, 1 100, 0 130, 1 {200}
force -freeze   right       0 0, 1 150, 0 180, 1 {200}

run 250
```

(a) Do Script code for the$nes_{t}o_{m}otorblockusedinthedesign.$