LABORATORY # 1

Decoders and Muxes

PART 1[1] (decoder) **Design of Sprinkler Valve Controller**

PART 2 (multiplexer) **Design of Computer Data Bus**

[1] Design Example guided through by TA

Objectives

Lab 1 contains 2 parts: **Part 1** – guided design and **Part 2** – individual design. Its purposes are to get familiar with:

1. EDA Playground environment.
2. Simulation and Design of controller systems based on combinational logic.
3. Generation of testbenches for logic design testing and verification.
4. Generation of waveforms.

Equipment

- PC or compatible

Parts

- N/A

PART 1. Design of a Sprinkler Valve Controller[2]

In this guided software experiment, we will design and test a 3 x 8 decoder (with "enable" switch) for a sprinkler valve controller system:

Specification

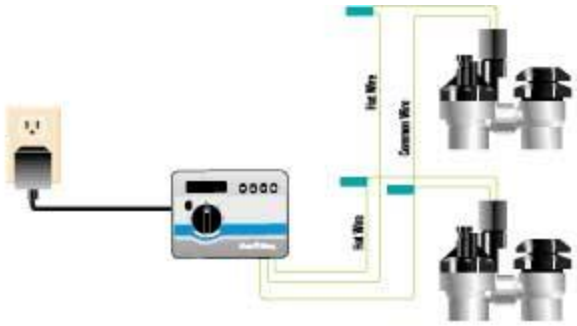

Figure 1. Sprinkler Digital Controller System

Part A: Automatic lawn sprinkler systems control the opening and closing of water valves. A sprinkler system must support several different zones, such as the backyard, left side yard, right side yard, etc. Only one zone's valve can be opened at a time to maintain enough pressure in the sprinklers in that zone. In this design assignment a sprinkler system must support up to 8 zones. Note that typically a sprinkler system is controlled by a small microcontroller unit (MCU) which executes a program that opens each valve only at specific times of the day and for specific durations. However, we will limit ourselves to a sub-project that is dealing only with opening and closing of the valves. The system must also provide a facility to disable the opening of any valve.

Analysis and Design

Assuming a microcontroller has only four output pins a system based on a 3 x 8 decoder (with "enable" switch) will do the job.

[2] Both parts of the lab are based on examples from Frank Vahid's "Digital Design"

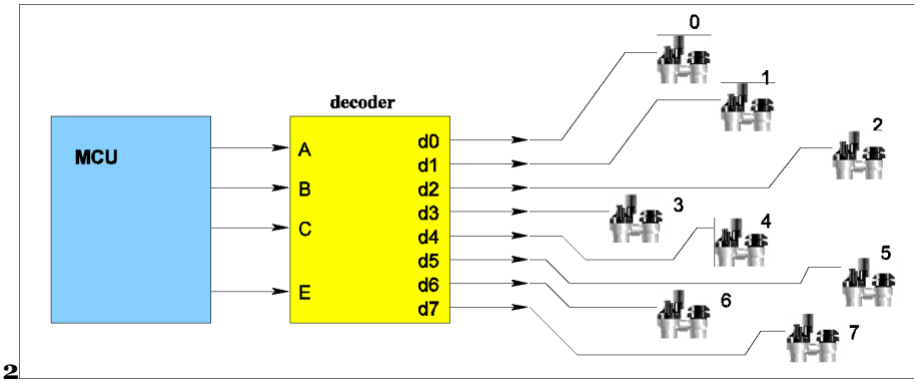

Figure 2

MCU has one pin to indicate whether the system is active (enabled) and the other three pins indicate the binary number of a valve to be opened. The system is a combinational logic circuit that has 4 inputs: E (enabler) and A, B, C (the binary value of the active zone), and 8 outputs d7, …, d0 (the valve controls). The truth table of the system is shown below.

E	A	B	C	d0	d1	d2	d3	d4	d5	d6	d7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Table 1. Truth Table of the Sprinkler System ('x' stands for "don't care")

Following the procedure outlined in Lecture 4, Slide 13 constructing sum-of-products (SOP) minterm based logic expression for each of the data outputs d.

Step 1 – Capture the function

$$d0 = E \cdot A' \cdot B' \cdot C'$$
$$d1 = E \cdot A' \cdot B' \cdot C$$
$$d2 = E \cdot A' \cdot B \cdot C'$$
$$d3 = E \cdot A' \cdot B \cdot C$$

d4 = E . A . B' . C'
d5 = E . A . B' . C
d6 = E . A . B . C'
d7 = E . A . B . C

Step 2 – Convert to equations and/or minimize.
Nothing to do here (already minimized).

Step 3 – Implement as a gate based logic circuit
It is shown below using a non-standard[3] 4-input AND gates and inverters.

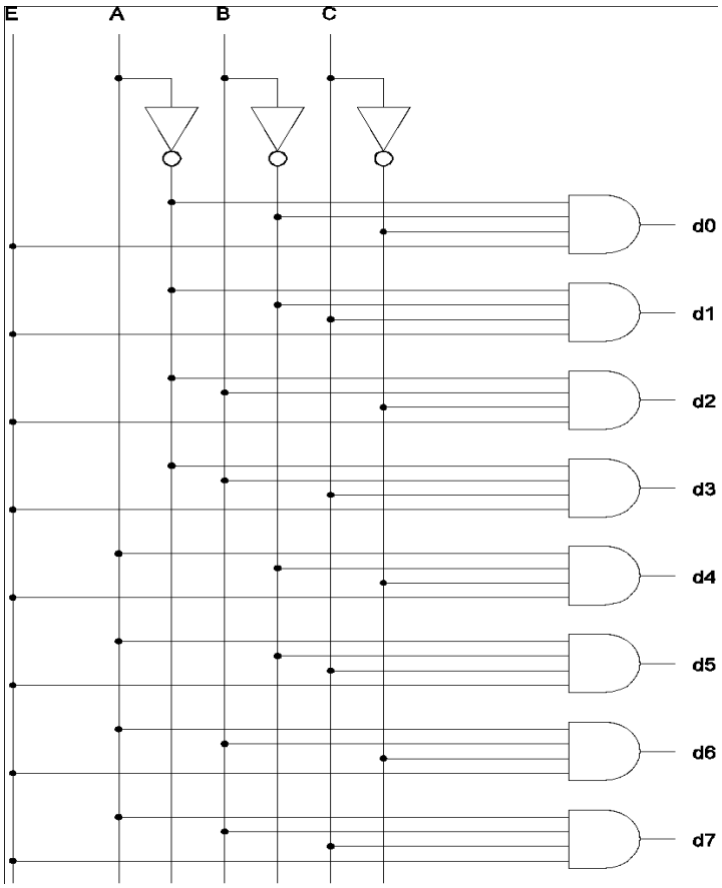

Figure 3. Logic Circuit Schematic of the Sprinkler System based on **Table 1**

Circuit Logic Behavioral Simulation and Verification

[3] Usage of non-standard gates largely clarifies a circuit schematic but in many cases requires a creation of a gate library manually within Logic Design Software Environments such as Xilinx ISE Design Suite

Part B: In this part of the lab you are required to implement the sprinkler valve controller system in verilog. In order to practice **structural and behavioral** modeling, you should have **two** verilog modules: one with the structural implementation and the other one with the behavioral implementation. Your verilog implementation should have the following set of input and output signals.

```
module and4(
input wire enable ,
input wire a,
input wire b,
input wire c,
output wire r
 );

//

assign r = enable & a & b & c ;

endmodule


// structural model

module decoder_st(
// I/O ports
 input  wire enable ,
 input  wire a ,
 input  wire b ,
 input  wire c ,
 output wire d0,
 output wire d1,
 output wire d2,
 output wire d3,
 output wire d4,
 output wire d5,
 output wire d6,
 output wire d7

 );
```

```
// Using the and4 module to set all outputs

and4 c1(enable, ~a, ~b, ~c, d0) ;
// Your code goes here (7 cases left to implement)
                                        and4 c2(enable, ~a, ~b, c, d1) ;
endmodule                               and4 c3(enable, ~a, b, ~c, d2) ;
                                        and4 c4(enable, ~a, b, c, d3) ;
// behavioral model                     and4 c5(enable, a, ~b, ~c, d4) ;
                                        and4 c6(enable, a, ~b, c, d5) ;
module decoder_bh(                      and4 c7(enable, a, b, ~c, d6) ;
                                        and4 c8(enable, a, b, c, d7) ;
// I/O ports

 input wire enable ,
 input wire a ,
 input wire b ,
 input wire c ,
 output reg d0,
 output reg d1,
 output reg d2,
 output reg d3,
 output reg d4,
 output reg d5,
 output reg d6,
 output reg d7

 );

// Internal wire
wire [3:0] bundle ;
assign bundle = {enable , a, b, c } ;

// Behavioral description

always @(*) begin

  d0 = 1'b0 ;
  d1 = 1'b0 ;
  d2 = 1'b0 ;
```

```
 d3 = 1'b0 ;
 d4 = 1'b0 ;
 d5 = 1'b0 ;
 d6 = 1'b0 ;
 d7 = 1'b0 ;

// Setting the correct output

 case (bundle)

        4'b1000: d0 = 1'b1 ;
        4'b1001: d1 = 1'b1 ;
        4'b1010: d2 = 1'b1 ;
        4'b1011: d3 = 1'b1 ;
        4'b1100: d4 = 1'b1 ;
        4'b1101: d5 = 1'b1 ;
        4'b1110: d6 = 1'b1 ;
        4'b1111: d7 = 1'b1 ;

        default : begin
                d0 = 1'b0 ;
        end

 endcase

end


endmodule
```

Next we will perform “exhaustive” circuit testing. To facilitate the implementation of the test bench, the following template is given. Copy the codes to the testbench panel in EDA Playground.

```
module decoder_tb;
        // Inputs
```

```
        reg enable;
        reg a;
        reg b;
        reg c;
        // Outputs
        wire d0;
        wire d1;
        wire d2;
        wire d3;
        wire d4;
        wire d5;
        wire d6;
        wire d7;

        // Instantiate the Unit Under Test (UUT)
        decoder_st uut ( // change to “decoder_bh” for testing your
behavioral model
                .enable(enable),
                .a(a),
                .b(b),
                .c(c),
                .d0(d0),
                .d1(d1),
                .d2(d2),
                .d3(d3),
                .d4(d4),
                .d5(d5),
                .d6(d6),
                .d7(d7)
        );

        initial begin

       $dumpfile("dump.vcd"); $dumpvars;

                enable = 1;
                a = 0;
                b = 0;
                c = 0;
                #100; // Wait for 100 ns
                $display("TC11 ");
```

```
if ( d0 != 1'b1 ) $display  ("Result is wrong");

a = 0;
b = 0;
c = 1;
#100;
$display("TC12 ");
if ( d1 != 1'b1 ) $display  ("Result is wrong");


a = 0;
b = 1;
c = 0;
#100;
$display("TC13 ");
if ( d2 != 1'b1 ) $display  ("Result is wrong");

a = 0;
b = 1;
c = 1;
#100;
$display("TC14 ");
if ( d3 != 1'b1 ) $display  ("Result is wrong");

a = 1;
b = 0;
c = 0;
#100;
$display("TC15 ");
if ( d4 != 1'b1 ) $display  ("Result is wrong");


a = 1;
b = 0;
c = 1;
#100;
$display("TC16 ");
if ( d5 != 1'b1 ) $display  ("Result is wrong");

a = 1;
b = 1;
```

```
            c = 0;
            #100;
            $display("TC17 ");
            if ( d6 != 1'b1 ) $display  ("Result is wrong");

            a = 1;
            b = 1;
            c = 1;
            #100;
            $display("TC18 ");
            if ( d7 != 1'b1 ) $display  ("Result is wrong");

          // Your test cases ********************

    end

endmodule
```

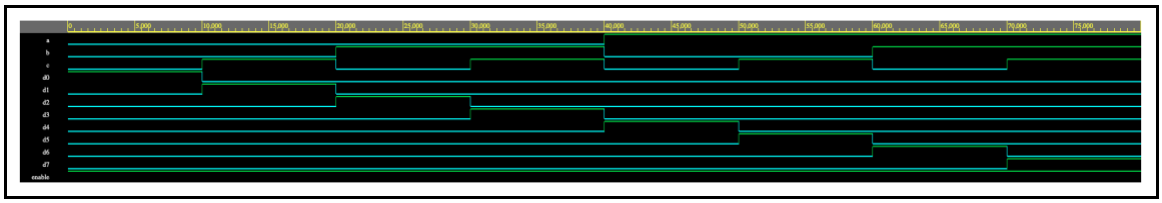

Figure 4. Behavioral Simulation of Sprinkler Controller Circuit

The sequence of input and output signals can be re-arranged by selecting the signal and pressing the up or down button.

Demonstration

See Figure 4. Observe that when E=0 ('E' is "Enable"), all outputs are 0's no matter what is in A,B,C inputs. Other rows from the truth table are correct.

Questions

1. What is a waveform?
2. What is a testbench?
3. Can we replace the 4-input AND gates in the circuit with the 2-input AND gates? If yes, how?

Yes, and please describe

Conclusion for Part 1

We have gone through the whole cycle of system design, analysis and circuit logic behavioral verification.

PART 2. Design of Computer Data Bus

In this assignment, we will design a 4 x 1 multiplexer that will control the flow of data in a single wire data bus and study its basic properties. This technology allows to accomplish, for example, partial serial communication with multiple peripheral devices using just one output pin of a microcontroller.

Specification

Design a 1-wire data bus controller that performs the following function:

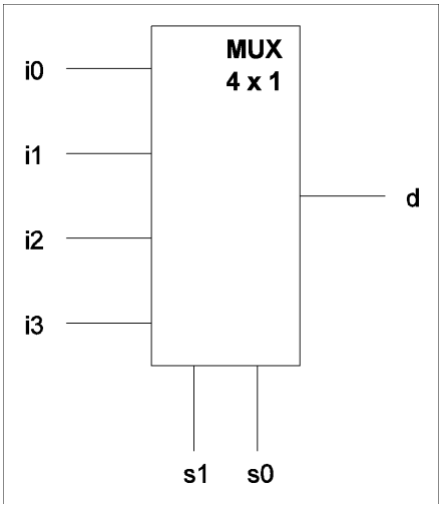

Figure 5. Single-wire data bus multiplexer

Inputs s0,s1 control which of the input data i0, i1, i2, i3 is present in the output d where the binary s1,s0 represents the input pin number.

Example: s1,s0 = 1,0 indicates that data in the input line i2 appear in the output d.

Design a logic circuit that will perform this function and verify the logic functionality using EDA playground environment.

Part B: In this part of the lab, you are required to implement the **behavioral and the structural** verilog model of the multiplexor described above. The following is the template given, and you will have to complete both the structural and behavioral models.

```
module and3(
input wire a,
input wire b,
input wire c,
output wire r
 );

assign r = a & b & c ;

endmodule


// structural model

module mux_st(

// Ports I/O
input wire s1,
input wire s0,
input wire i0,
input wire i1,
input wire i2,
input wire i3,
output wire d
);

wire   r1, r2, r3, r4 ;

and3 c1 ( ~s1,~s0,  i0,  r1 ) ;
```

```
// Your code goes here (3 cases left)

assign d = r1 | r2 | r3 | r4 ;

endmodule


// behavioral model

module mux_bh(

// Ports I/O

input wire s1,
input wire s0,
input wire i0,
input wire i1,
input wire i2,
input wire i3,
output reg d
) ;

always @(*) begin

        d = 1'b0 ;

        case ( {s1,s0} )

                2'b00 : d = i0 ;
                // your code goes here (3 cases left)

        endcase

end

endmodule
```

To facilitate the implementation of the test bench, the following template is given as well.

```
`timescale 1ns / 1ps

module mux_tb;

	// Inputs
	reg s1;
	reg s0;
	reg i0;
	reg i1;
	reg i2;
	reg i3;

	// Outputs
	wire d;

	// Instantiate the Unit Under Test (UUT)
	mux_bh uut ( // change to “mux_st” for testing your structural model
			.s1(s1),
			.s0(s0),
			.i0(i0),
			.i1(i1),
			.i2(i2),
			.i3(i3),
			.d(d)
	);

	initial begin

	  $dumpfile("dump.vcd"); $dumpvars;

			i0 = 1;
			i1 = 0;
			i2 = 1;
			i3 = 0;

			s1 = 0;
			s0 = 0;
			#100;
			$display("TC11 ");
			if ( d != i0 ) $display  ("Result is wrong");
```

```
            s1 = 0;
            s0 = 1;
            #100;
            $display("TC12 ");
            if ( d != i1 ) $display  ("Result is wrong");

            s1 = 1;
            s0 = 0;
            #100;
            $display("TC13 ");
            if ( d != i2 ) $display  ("Result is wrong");

            s1 = 1;
            s0 = 1;
            #100;
            $display("TC14 ");
            if ( d != i3 ) $display  ("Result is wrong");

          // Your test cases

    end

endmodule
```

Demonstration

Demo the waveform obtained during the behavioral simulation and explain why it is correct. Provide the truth table, algebraic expression of the logic function, and logic circuit schematic.

Procedures

1. EDA playground environment;
2. Manipulation of input signal timings in testbenches;

Presentation and Report

Must be presented according to the general EE/CS120A lab guidelines.

Prelab

1. Familiarize yourself with EDA playground and verilog tutorials posted on Canvas;
2. Review relevant lecture material (e.g. decoders, muxes);
3. Try to answer all the questions and do all necessary computations in this manual