

## LABORATORY # 2

### **Decoders and Muxes**

PART 1<sup>1</sup> (decoder)      **Design of Sprinkler Valve Controller**

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

---

<sup>1</sup> Design Example guided through by TA

## Objectives

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

1. Xilinx ISE Design software system usage.
2. Simulation and Design of controller systems based on combinational logic.
3. Generation of testbenches for logic design testing and verification.
4. Generation of waveform.

## Equipment

- PC or compatible
- Xilinx ISE Design Software
- ModelSim XE III modeling software

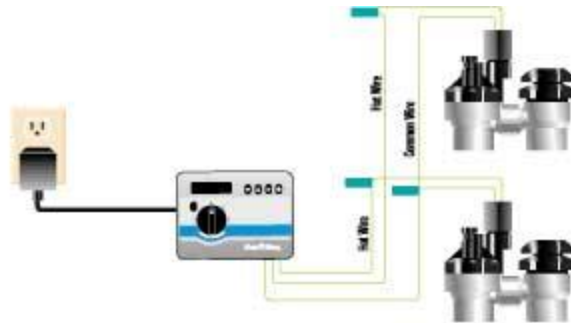
## Parts

- N/A

## **PART 1. Design of a Sprinkler Valve Controller<sup>2</sup>**

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**



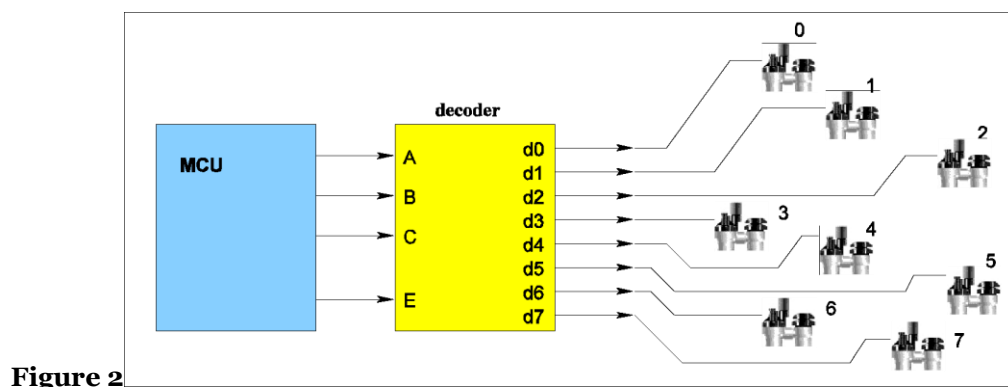
**Figure 1.** Sprinkler Digital Controller System

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.

<sup>2</sup> Both parts of the lab are based on examples from Frank Vahid’s “Digital Design”



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

$$\begin{aligned}
 d0 &= E \cdot A' \cdot B' \cdot C' \\
 d1 &= E \cdot A' \cdot B' \cdot C \\
 d2 &= E \cdot A' \cdot B \cdot C' \\
 d3 &= E \cdot A' \cdot B \cdot C
 \end{aligned}$$

$$d4 = E \cdot A \cdot B' \cdot C'$$

$$d5 = E \cdot A \cdot B' \cdot C$$

$$d6 = E \cdot A \cdot B \cdot C'$$

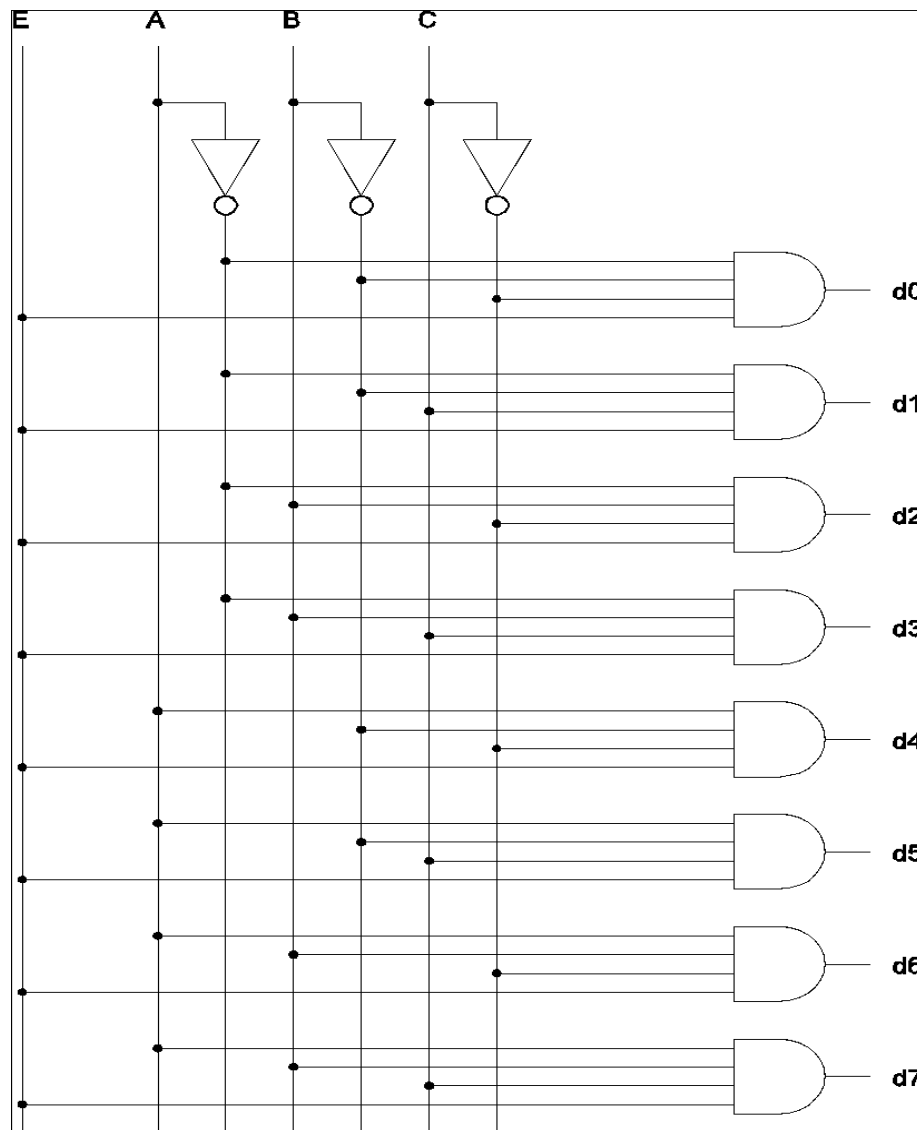
$$d7 = E \cdot A \cdot B \cdot C$$

**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<sup>3</sup> 4-input AND gates and inverters.*

---

<sup>3</sup> 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



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

### **Circuit Schematic Capture**

Using Xilinx ISE WebPack.

File -> New Project -> "ee120a\_L2P1\_sprinkler\_valve\_controller" (or appropriate name) -> (accept defaults<sup>4</sup> in the following pop-up hardware target wizard by

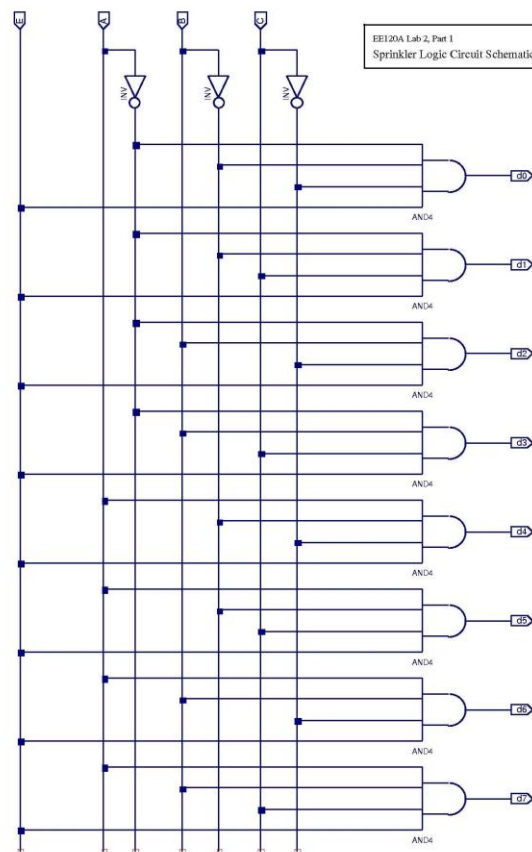
<sup>4</sup> We assume that the defaults have not changed since the settings were implemented in Lab 1. Reference the Lab 1 settings if the default looks incorrect or unfamiliar

pressing Next) -> New Source (schematic, "sprinkler\_circuit") -> Next -> Finish

In the middle panel window one can see a design summary tab. Switch to the "sprinkler\_circuit.sch" tab. This is our schematic window.

In the "Symbols" panel choose logic symbols for the 4-input AND gate "and4" and the inverter "inv". Placing them according to the schematic<sup>5</sup> in [Figure 3](#) and wire them properly with "Add Wire".

Add Inputs/Outputs with "Add I/O Marker".



**Figure 4.** Sprinkler Controller logic circuit schematic created within ISE WebPack

## **Circuit Logic Behavioral Simulation and Verification**

<sup>5</sup> In general there is no need to create two separate logic circuits with different software. One will suffice.

Next we will perform “exhaustive” circuit testing. In “Design” panel go to the “Design” tab and select `sprinkler_circuit.sch`. Now go to the “Hierarchy” panel -> Create New Source -> Verilog Test Bench -> `sprinkler_circuit_tb` (Xilinx is phasing this option out – hit “OK”). To facilitate the implementation of the test bench, the following template is given.

```

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 (
        .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

        enable = 1;
    
```



```

a = 0;
b = 0;
c = 0;
#100; // Wait for 100 ns
$display("TC11 ");
if ( do != 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

```

**NOTE: If you copy paste the above code, Xilinx will ask if you wish to convert the text, answer yes.**

Save your test bench.

Switch to Simulation mode and select the test bench you created.

Expand the ISim Simulator tree, and double click “Simulate Behavioral Model.”  
Once again, hold the Ctrl and scroll out and then adjust your waveform view.

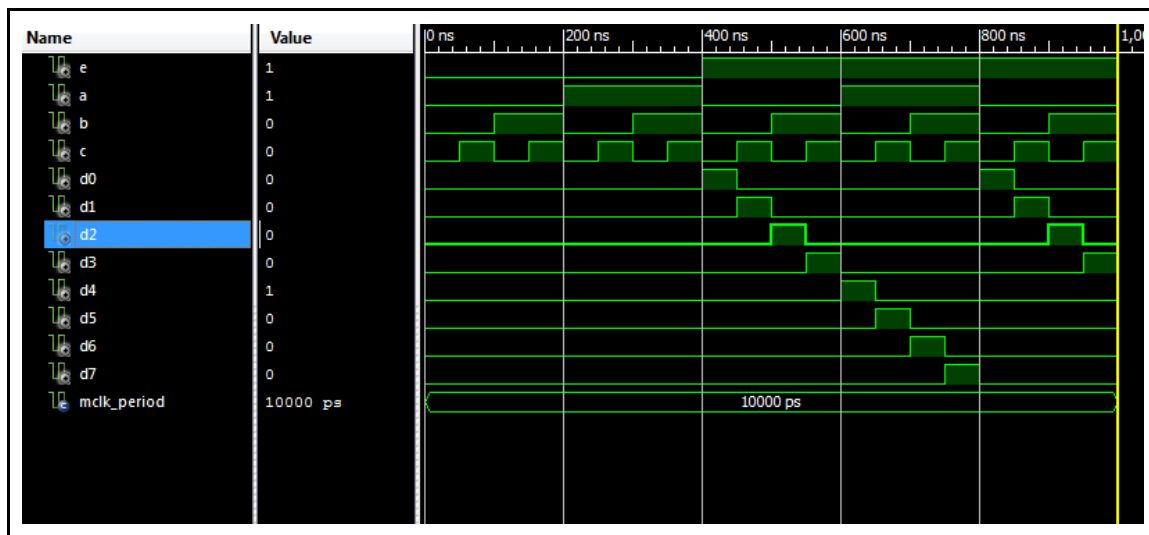


Figure 6. Behavioral Simulation of Sprinkler Controller Circuit

The sequence of input and output signals can be re-arranged by pressing and holding the signal line and moving it up or down.

Note that we can customize the test bench and the simulation windows:

- Add dividers in between sections (e.g. inputs, outputs) as well as customize the height and text/background color
- Make groups of signals to establish an expandable heirarchy; instead of deleting the extra signals that are not inputs or outputs (e.g. period\_VAR and duty\_cycle\_VAR), we can group these into a signal group and contract the group so that it is not visible (as shown in Figure 6 below)



Figure 7. Behavioral Simulation of Sprinkler Controller Circuit (note that the outputs in this figure are labeled “yo” to “y7” instead of “do” to “d7”)

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 decoder_st(
// I/o ports
input wire e ,
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(e, ~a, ~b, ~c, d0) ;
// Your code goes here

endmodule
```

### **Demonstration**

See Figure 7. 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?

### **Conclusion for Part 1**

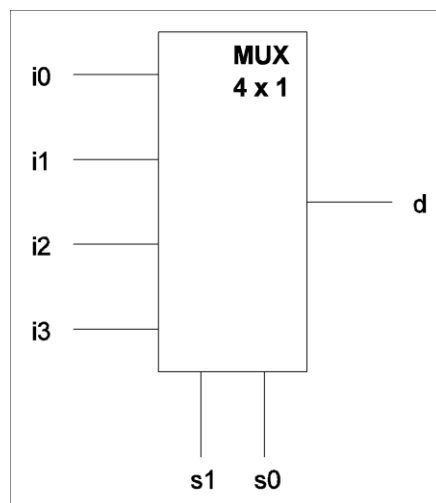
We have gone through the whole cycle of system design, analysis and circuit logic behavioral verification.

### **PART 2. Design of Computer Data Bus**<sup>6</sup>

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 7.** 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.

---

<sup>6</sup> This part must be completed individually, **NOT in groups**

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 ISE/ModelSim software environment.

**NOTE: You can keep multiple schematics and tests in every project, however, you must be very careful that you have the correct schematic “Set as Top Module” and that you create the test bench with the correct “Associate Source” in the New Source wizard.**

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 for the 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

assign d = r1 | r2 | r3 | r4 ;

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 io;
    reg i1;
    reg i2;
    reg i3;

    // Outputs
    wire d;

    // Instantiate the Unit Under Test (UUT)
    mux_bh uut (
        .s1(s1),
        .s0(s0),
        .io(io),
        .i1(i1),
        .i2(i2),
        .i3(i3),
        .d(d)
    );

    initial begin

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

        s1 = 0;
        s0 = 0;
        #100;
        $display("TC11 ");
        if ( d != io ) $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 also the truth table, algebraic expression of the logic function, and logic circuit schematic.

### **Procedures**

1. Xilinx ISE Design environment;
2. Logic circuit schematic capture;
3. Manipulation of input signal timings in ISE testbenches;
4. Behavioral Simulation and Waveform analysis of logic circuits.

### **Presentation and Report**



Must be presented according to the general EE120A lab guidelines posted in iLearn.  
You need to include all Verilog code you wrote for both PartA and PartB

## **Prelab**

1. Familiarize yourself with ISE/Xilinx tutorials posted in iLearn;
2. Review relevant lecture material (e.g. decoders, muxes);
3. Try to answer all the questions and do all necessary computations in this manual