Remi de Ferrieres
X664611
CS 120A Section 021
Lab 2 – Decoders ans Muxes
Yann Abou Jaoude
X663581

OVERVIEW :

In the first part we worked on the design of a sprinkler valve controller. We worked on decoders for this part. We built the shematic, we attached to it a Verilog and did the wave form related to this. We had 3 inputs and 8 outputs (decoder 3x8). For each combinations of the inputs we had one output activated. So every sprinkler worked on one of the inputs combination.
In part B we did the same but based on a Verilog module instead of making a schematic.

In the second part we worked on the design of a computer data bus. We worked on Muxes. We didn't build any schematics but coded a module instead (mux_st). We had 4 inputs, 2 selectors and 1 output (mux 4x1). The results on the timing diagram matched the truth table of the mux, as expected.

NEW CONCEPTS:

In this lab we use modules. They represent the building blocks of the Verilog design.

We used the following logic gates: AND (AND3 and AND4) NOT(inv). These are basic logic gates.

ANALYSIS:

The typical procedure we followed during this lab is the following: first we build our circuit using a schematic or coding a module. Then based on that we coded a Verilog that represented what value should the variables take in a condition (like a truth table). Then we visualized the timing diagram of that.

Truth table of the 3x8 decoder of part 1:

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							
1	0	0	1		1						
1	0	1	0			1					
1	0	1	1				1				
1	1	0	0					1			
1	1	0	1						1		
1	1	1	0							1	
1	1	1	1								1

Equations:

D0 = EA’B’C’ D4 = EAB’C’
D1 = EA’B’C D5 = EAB’C
D2 = EA’BC’ D6 = EABC’
D3 = EA’BC D7 = EABC

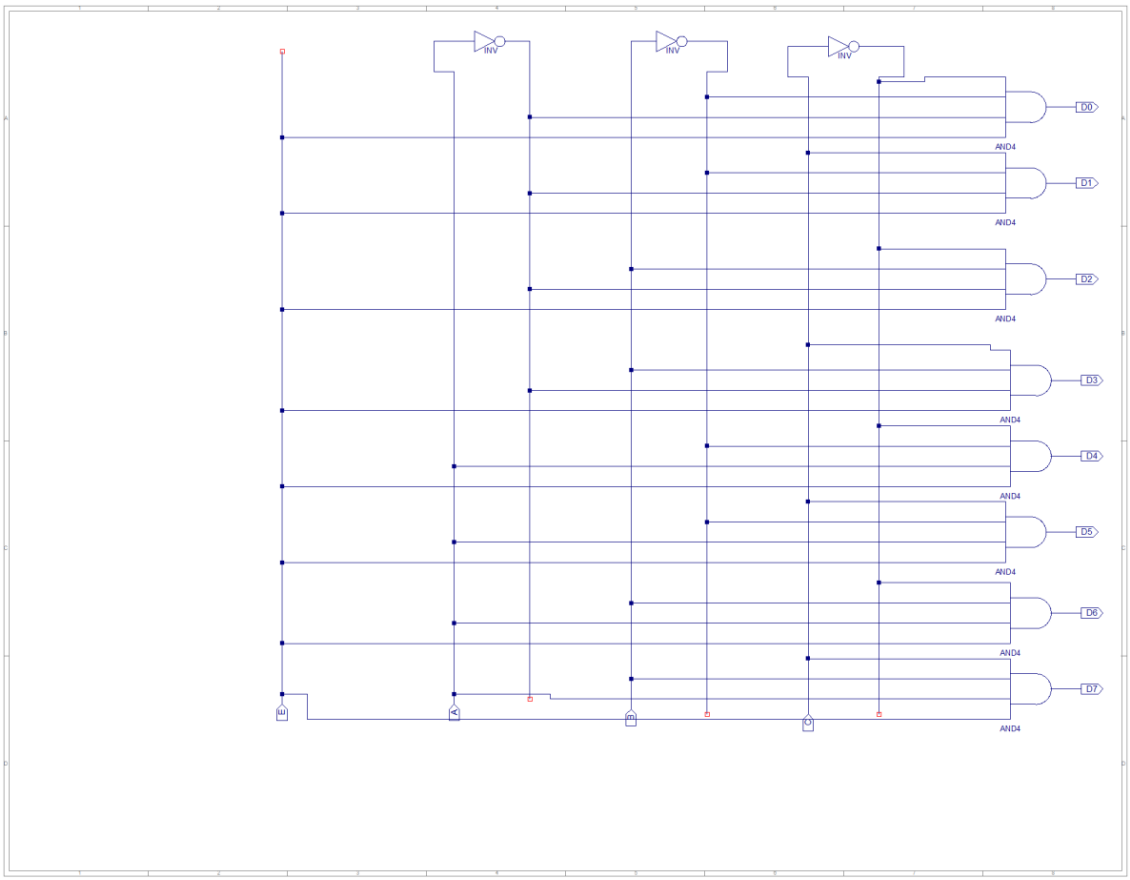
Truth table of the 4x1 mux:

E	S0	S1	D
0	x	x	0
1	0	0	I0
1	0	1	I1
1	1	0	I2
1	1	1	I3

Equation:

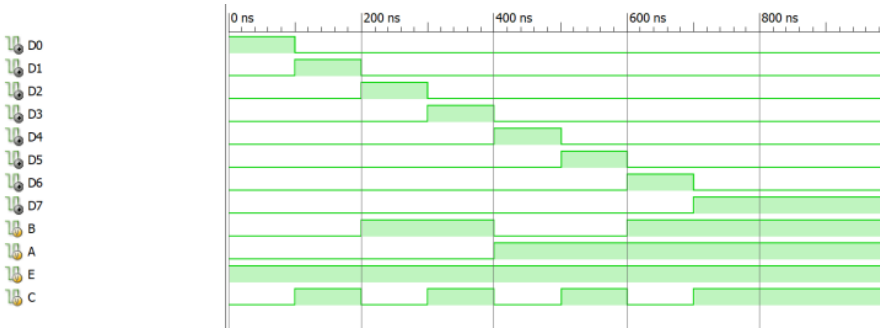
D = i0S1’S0’E + i1S1S0’E + i2S1’S0E + i3S1S0E

RECORDS:



Schematic of the 3x8 decoder

Timing diagram for the 3x8 decoder:

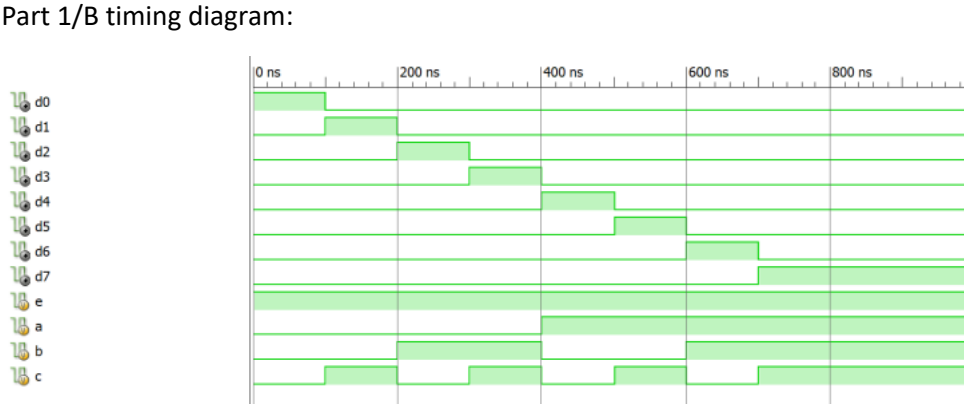


Part 1/B source code:

```
module LAB_2_2_verilog(
    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) ;
and4 c2(e, ~a, ~b, c, d1) ;
and4 c3(e, ~a, b, ~c, d2) ;
and4 c4(e, ~a, b, c, d3) ;
and4 c5(e, a, ~b, ~c, d4) ;
and4 c6(e, a, ~b, c, d5) ;
and4 c7(e, a, b, ~c, d6) ;
and4 c8(e, a, b, c, d7) ;
endmodule
// Your code goes here
module and4(
    input wire e ,
    input wire a ,
    input wire b ,
    input wire c ,
    output wire d
    );
    assign d = e&a&b&c;
endmodule
```

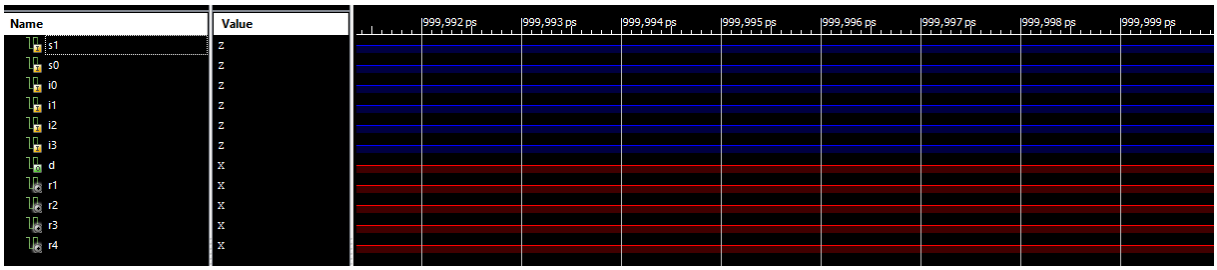
Part 1/B timing diagram:



Part 2 mux source code:

```
21  module and3(
22     input wire a,
23     input wire b,
24     inout wire c,
25     output wire d
26     );
27
28     assign d = a&b&c;
29  endmodule
30
31  module mux_st(
32  // Ports I/O
33  input wire s1,
34  input wire s0,
35  input wire i0,
36  input wire i1,
37  input wire i2,
38  input wire i3,
39  output wire d
40  );
41
42  wire r1, r2, r3, r4 ;
43  and3 c1 ( ~s1,~s0, i0, r1 ) ;
44  and3 c2 ( ~s1,s0, i1, r2 ) ;
45  and3 c3 ( s1,~s0, i2, r3 ) ;
46  and3 c4 ( s1,s0, i3, r4 ) ;
47  // Your code goes here
48  assign d = r1 | r2 | r3 | r4 ;
49  endmodule
50
```

Part 2 mux timing diagram:



DISCUSSION:

We compared the waveform to our truth table. The results are the same, our system is fine according to our bench-test. Our major difficulty was to navigate through the software. The way we must create/navigate through the tabs and the files are very precise. Sometimes, we misunderstood an English word so we were lost. This difficulty may disappear when we will be used to xilinks. Even though our system is simple, there are plenty of ways to improve it: - We can improve the code part: We can create integers and use loops to increment them, so we don't copy-paste a hundred of times the same lines. This also allows changing the numbers of input way more easily. -We can build our circuit in a lower level. Especially we can verify the 4-input AND gates. How are they done? Are they a smart 2 Levels 2-input AND gates combination or a less smart 3 levels? Our system is quite little, it allows us to go deeper: Is there a way to simplify some Pmos and Nmos?

CONCLUSION:

In addition to creating our system, xilinks allows us to test them. The code is very similar to the C language. For now, we have learned to compare visually with the waveform. One can imagine creating a program that tests all the input possibilities of a system and compares them to a truth-table itself. Even if the xilinks software is sometimes a little mysterious, Its great potential is no doubt The Multiplexer allows you to select an input to output it. The decoder makes it possible to turn an output to 1 according to an input combination. We can easily see that we have done on a small scale what happens in a memory register.

QUESTIONS:

Waveform: A graphic representation of the shape of a wave that indicates its characteristics. For our system, our waveform is equivalent to a truth table. We use the x ax, which represent time, to test every possible entry. On the y ax there is only two value, 0-1 according to the line's value. Test-bench: A protocol testing if a machine or device is working properly. It consists in testing all the entries possible value and comparing practical results to theoretical result. This is what we do. Our waveform shows us our practical values, to complete our testbench, we must compare our waveform to our theoretical truth-table. Can we replace the 4-input AND gates in the circuit with the 2-input AND gates? Yes, we can, with a combination of 3 2-input AND. There is multiple ways to it, but the smarter is to do it with a 2 lever circuit. The two outputs of the first two AND gates should go in input in the third AND gate.