Gebze Technical University
Computer Engineering

CSE 331 – 2019 Fall

HOMEWORK 2 REPORT

BERKE SÜSLÜ
161044076

1 INTRODUCTION

1.1 Problem Definition

Design 4:1 mux, 1-bit ALU and 32-bit ALU with using only AND,OR and NOT gate.

1.2 System Requirements

Quartus II 64-Bit Version 13.1.0 Build 162 10/23/2013 SJ Web Edition

2 METHOD

2.1 Problem Solution Approach

My 4:1 mux gate has 2 NOT gate,4 AND gate(3-input) and 1 OR gate(4-input). (Total Gates = 7)

I designed an XOR gate using 2 NOT gate, 2 AND gate and 1 OR gate.(Total Gates = 5)

My 1-bit ALU has 4 AND gate, 3 OR gate , 2 NOT gate, 1 XOR gate (which is i designed before) and 1 4:1 mux gate. (Total Gates = 21)

My 32-bit ALU has 32 quantity of 1-bit ALU. (Total Gates = 21x32 = 672).

My modules are xor gate,4:1 mux gate, 1-bit ALU, 32-bit ALU. Each module has own test module.

3 RESULT

3.1 Test Cases

My XOR gate test cases have 4 different input:
A=0 B=0
A=0 B=1
A=1 B=0
A=1 B=1

My 4:1 Mux gate test cases have 4 different select bit:

A0 = 1
A1 = 0
A2 = 0
A3 = 1

S1=0 S2=0
S1=0 S2=1
S1=1 S2=0
S1=1 S2=1

My 1-bit ALU test cases:
5 different operation with 4 different cases:

My 32-bit ALU test cases:
5 different operation with 1 different case:

3.2 Running Results

XOR gate:

```
# time =  0, a =0, b=0, sum=0
# time = 20, a =0, b=1, sum=1
# time = 40, a =1, b=0, sum=1
# time = 60, a =1, b=1, sum=0
```

4:1 mux gate:

```
# time =  0, a1 =1,a2=0,a3=0,a4=1,s1=0,s2=0,mux_out=1
# time = 20, a1 =1,a2=0,a3=0,a4=1,s1=0,s2=1,mux_out=0
# time = 40, a1 =1,a2=0,a3=0,a4=1,s1=1,s2=0,mux_out=0
# time = 60, a1 =1,a2=0,a3=0,a4=1,s1=1,s2=1,mux_out=1
```

1-bit ALU:

```
# time =  0, a=0,b=0,c=0,aluop0=0,aluop1=0,aluop2=0,less=0,c_out=0,r=0
# time = 20, a=1,b=0,c=0,aluop0=0,aluop1=0,aluop2=0,less=0,c_out=0,r=0
# time = 40, a=0,b=1,c=0,aluop0=0,aluop1=0,aluop2=0,less=0,c_out=0,r=0
# time = 60, a=1,b=1,c=0,aluop0=0,aluop1=0,aluop2=0,less=0,c_out=1,r=1
# time = 80, a=0,b=0,c=0,aluop0=1,aluop1=0,aluop2=0,less=0,c_out=0,r=0
# time = 100, a=1,b=0,c=0,aluop0=1,aluop1=0,aluop2=0,less=0,c_out=0,r=1
# time = 120, a=0,b=1,c=0,aluop0=1,aluop1=0,aluop2=0,less=0,c_out=0,r=1
# time = 140, a=1,b=1,c=0,aluop0=1,aluop1=0,aluop2=0,less=0,c_out=1,r=1
# time = 160, a=0,b=0,c=0,aluop0=0,aluop1=1,aluop2=0,less=0,c_out=0,r=0
# time = 180, a=1,b=0,c=0,aluop0=0,aluop1=1,aluop2=0,less=0,c_out=0,r=1
# time = 200, a=0,b=1,c=0,aluop0=0,aluop1=1,aluop2=0,less=0,c_out=0,r=1
# time = 220, a=1,b=1,c=0,aluop0=0,aluop1=1,aluop2=0,less=0,c_out=1,r=0
# time = 240, a=0,b=0,c=0,aluop0=0,aluop1=1,aluop2=1,less=0,c_out=0,r=1
# time = 260, a=1,b=0,c=0,aluop0=0,aluop1=1,aluop2=1,less=0,c_out=1,r=0
# time = 280, a=0,b=1,c=0,aluop0=0,aluop1=1,aluop2=1,less=0,c_out=0,r=0
# time = 300, a=1,b=1,c=0,aluop0=0,aluop1=1,aluop2=1,less=0,c_out=0,r=1
# time = 320, a=0,b=0,c=0,aluop0=1,aluop1=1,aluop2=1,less=0,c_out=0,r=0
# time = 340, a=1,b=0,c=0,aluop0=1,aluop1=1,aluop2=1,less=0,c_out=1,r=0
# time = 360, a=0,b=1,c=0,aluop0=1,aluop1=1,aluop2=1,less=0,c_out=0,r=0
# time = 380, a=1,b=1,c=0,aluop0=1,aluop1=1,aluop2=1,less=0,c_out=0,r=0
```

32-bit ALU:

```
# time =  0, a=11101111101110111100011110110110,b=01011110001011100101100111110010,c=0,aluop0=0,aluop1=0,aluop2=0,less=0,c_out=0,r=01001110001010100100000110110010
# time = 20, a=11101111101110111100011110110110,b=01011110001011100101100111110010,c=0,aluop0=1,aluop1=0,aluop2=0,less=0,c_out=0,r=11111111101111111101111111110110
# time = 40, a=11101111101110111100011110110110,b=01011110001011100101100111110010,c=0,aluop0=0,aluop1=1,aluop2=0,less=0,c_out=0,r=10001110010010100111111010111101
# time = 60, a=11101111101110111100011110110110,b=01011110001011100101100111110010,c=0,aluop0=0,aluop1=1,aluop2=1,less=0,c_out=1,r=00111101010010110100100100111000
# time = 80, a=11101111101110111100011110110110,b=01011110001011100101100111110010,c=0,aluop0=1,aluop1=1,aluop2=1,less=0,c_out=1,r=00000000000000000000000000000000
```