GTU

**DEPARTMENT OF COMPUTER ENGINEERING**

**CSE 331 – Autumn 2022**

**HOMEWORK 2  
REPORT**

SÜLEYMAN GÖLBOL 1801042656

# Main Components

# 

# Multiplication Component

# metin, beyaz tahta içeren bir resim Açıklama otomatik olarak oluşturuldu

# HL STATE MACHINE

# 

# FINITE STATE MACHINE OF VARIABLES

# metin, beyaz tahta içeren bir resim Açıklama otomatik olarak oluşturuldu

# TABLE FOR EVERY STATE

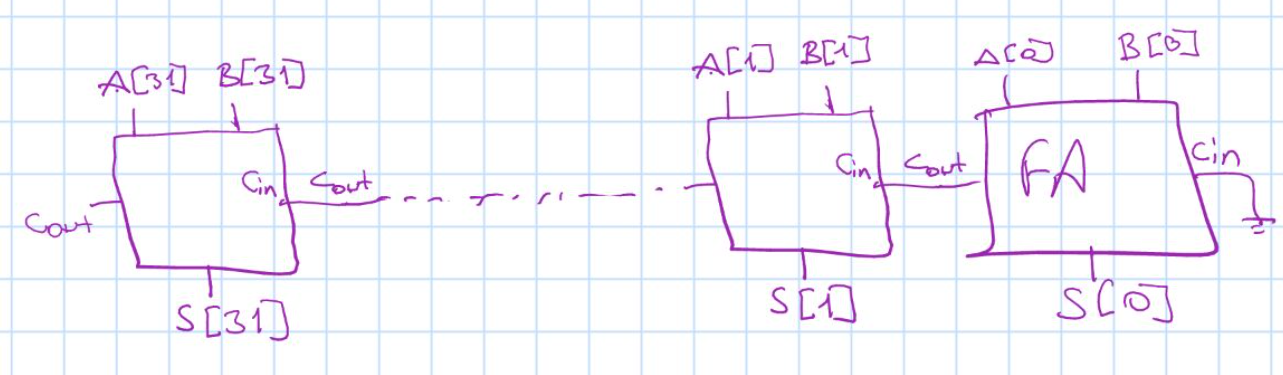
# tablo içeren bir resim Açıklama otomatik olarak oluşturuldu

# DERIVING LOGIC

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# ADDER COMPONENT

For adder component, I created a 1-bit full adder. To create 32 bits adder, I used every full adder’s “carry out” value to connect. Then I put the output into 32 bits sum variable.



***Testing***

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

I tried to add 48050 and 11111. The sum should be 59161.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

# CONTROL COMPONENT

# 

# For the control part, I needed to use state table variables values to achieve mult\_or\_shift, reset\_or\_increment, load\_loop\_checker, load\_higher, load\_lower values using inputs reset, clk and reg\_least\_bit.

***At the end values are like that.***

# 

# DATAPATH COMPONENT

# 

# For the datapath component I used 2 multiplexer to select current state or selecting 64 bit number to produce. Multiplexer above takes first 32 bits and below takes last 32 bits. Mult\_or\_Shift is connected to first multiplexer to control selection. Below, add\_or\_shift is connected to multiplexer to control selection. Shiftright\_64bit has control signal which is 1, so it will shift right the 64 bit number.

# Since it will continue until 32 there is a loop checker and ctr32 connected to incrementer. Incrementer uses mux2x1\_8bits, adder\_8bits and setonlessthan\_32bits to increment the number.

# Also, to prevent repetition for array-like variables, I used generate keyword with “genvar” variable.

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# It’s just a helper not to write same thing tens of times.

# MULT32.V

# Multiplier module connects them all to multiply number.

# 

As it seen first uses control then uses datapath module with all needed parameters.

***Testing***

# Multiplying 67 \* 883

# 

# After adding waves in ModelSim app to testbench, this is the result.

# 

# This is the result as expected because 67 \* 883 = 59161

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# ALU32.V

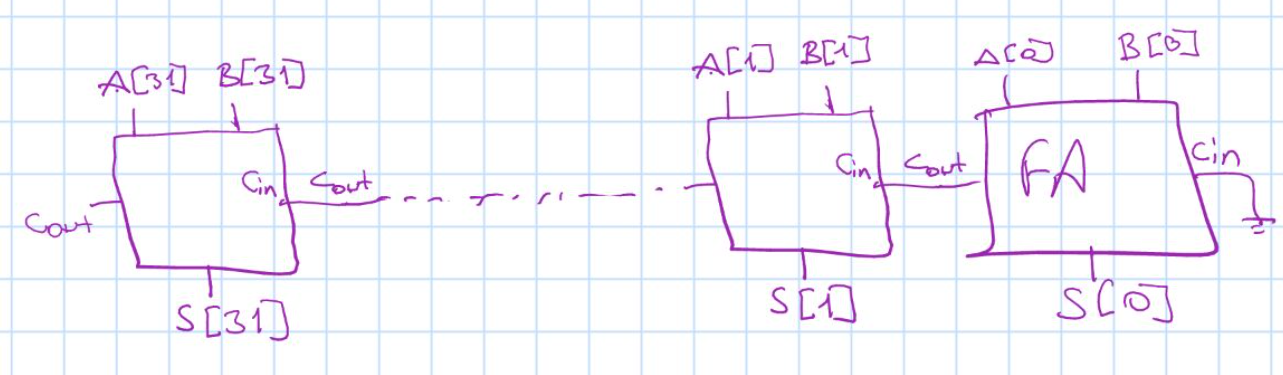
# Alu32 module have a multiplexer to select one of the 8 options.

|  |  |
| --- | --- |
| 000->ADD | 100->AND |
| 001->SUB | 101->OR |
| 010->MULT | 110->SLT |
| 011->XOR | 111->NOR |

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# Adder

Like I wrote above, it’s made by connecting full adders.



# A and B Inputs are: 12345 and 21543. A+B = 33888

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# Sub

# 

# It only takes 2’s complement for sub

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# A and B Inputs are: 12345 and 21543. A-B = -9198

# tablo içeren bir resim Açıklama otomatik olarak oluşturuldu

# Mult

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# A and B Inputs are: 12345 and 21543. A\*B= 265948335

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# Ignore 0’s in the beginning in the mult result.

# Important Note:

# ModelSim is printing all intermediate stages for multiplication. Please ignore

# Other AluOp:010 results and look only to the last ‘ALUop:010’ result.

# Xor

# 

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# A and B Inputs are: 12345 and 21543. Result is same.

# And

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# 

# Or

# 

# metin, tablo içeren bir resim Açıklama otomatik olarak oluşturuldu

# They are same.

# Slt

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# First 32 bits are full of 0’s. And the last bit is found out by using subtractions most significant bit.

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# Since A(12345) is smaller than B(21543), result will be 1.

# Nor

# Just nors every bit.

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# Results are same.