Mini-Processor block with Bus

2019202052 김호성

Abstract

디지털 논리 term project를 최종 보고서 작성을 위한 기본 template format입니다. Abstract는 해당 보고서의 내용을 전체적으로 요약하는 부분입니다. 이 뒤로 introduction, project specification, design details, design verification strategy and results에 대하여 각각의 해당하는 session에 작성하여 주시기 바랍니다. 마지막에 references는 자기가 참조한 서적, 논문 등에 대하여 정리하는 것입니다. References의 내용을 보고서에 적을 때는 예를 들어 'multiplier는 ~~~하는 장치이다. [1]' 이런 식으로 적어주시면 됩니다.

I. Introduction

BUS와 간단한 processor 역할을 하는 mini-processor block MP를 이용하여, 곱셈, 덧셈, 뺄셈 및 논리연산을 한다. MP내에는 register들과 ALU와 MUL이 존재한다.

II. Project Specification

ALU

Arithmetic Logic Unit의 약자로 산술 논리 장치라고 한다. 두 숫자의 산술연산과 논리연산등을 계산하는 디지털 회로이며, 프로젝트에서 존재하는 기능은 다음과 같다.

0000 NOP No operation
0001 NOT A Rd = ~Ra
0010 AND Rd = Ra and Rb
0011 OR Rd = Ra or Rb
0100 XOR Rd = Ra xor Rb
0101 XNOR Rd = Rd xnor Rb
0110 ADD Rd = Ra + Rb
0111 SUB Rd = Ra – Rb

기존의 ALU에서는 4개의 flag 값이 존재하였으나(C, N, Z, V), 프로젝트에서는 flag를 구하지 않고 진행한다.

MUL

Multiplier는 multiplicand(피승수)와 multiplier(승수)를 곱하여 결과 값을 도출하는 Logic이다.

지난 과제에서는 bit length가 64bit로 결과 값이 128bit였으나, 프로젝트에서는 bit length가 32bit로 변경되었으며, 곱의 결과값은 64bit다.

Booth Multiplication Radix-2

Booth Multiplication 이란 이진법으로 표현된 어느 두 변수의 곱셈에서 승수의 마지막 비트와 마지막에서 두 번째 비트를 비교하여 이에 따른 연산을 진행하고 그 연산의 결과에 따라 곱셈의 결과를 얻을 수 있는 곱셈방법을 말한다.

우선 이 방법을 사용하기 위해서는 4 개의 임의의 변수가 필요하다.

이 4 개의 변수를 u, v, x, x-1 이라고 했을 때, u, v, x-1 은 0으로 초기화하고 x는 승수가 된다 x의 마지막 비트와 x-1을 비교하여 연산을 진행하는데

{x, x-1} = {0,0} 일 경우 u의 값을 ASR연산을 하고 v는 u의 마지막비트를 최상위 비트로 가지며, LSR 연산을 진행한다.

{x, x-1} = {0,1} 일 경우 u의 값을 피승수와 전의 u의 값을 더한 후 ASR 연산을 한다. v는 u의 마지막 비트를 최상위 비트로 가지며, LSR 연산을 진행한다.

{x, x-1} = {1,0} 일 경우 u의 값을 피승수와 전의 u의 값과 뺀 후 ASR 연산을 한다. v는 u의 마지막비트를 최상위 비트로 가지며 LSR 연산을 진행한다.

{x, x-1} = {1,1} 일 경우 u의 값을 ASR연산을 하고 v는 u의 마지막비트를 최상위 비트로 가지며 LSR 연산을 진행한다. 이러한 방법으로 연산이 진행되는데 이 때, 승수의 비트(x)가 32번째이면 마지막으로 연산을 진행한 후 연산을 멈추고 그 때의 계산된 u 와 v의 값을 이어 붙여 64비트짜리 결과 값을 얻는다.

RF

register file(RF)는 다음과 같은 구성으로 Logic이 동작한다.

Set of Registers

Read operations

Write operations

프로젝트의 경우 DATA_REG 10개, INST_REG10개, CONT_REG3개로 총 23개의 register가 존재하며, register를 23개를 묶어서 만든 Logic은 주석처리 하였고, 각각의 register (10, 10, 3)으로 묶은 Logic을 구현했다.

후에 Register를 관리하는 FSM을 어떤 방식으로 만들지에 따라서 두 가지의 버전 중 하나를 선택해 프로젝트를 완성할 계획이다.

BUS

BUS는 여러 component들 간에 data를 전송할 수 있도록 연결해주는 Logic이다.

지난 과제의 경우 2개의 master와 2개의 slave를 가지고 있었으나, 프로젝트에서는 1개의 master와 1개의 slave를 가지고 있으며, address의 bandwidth는 16bit이다.

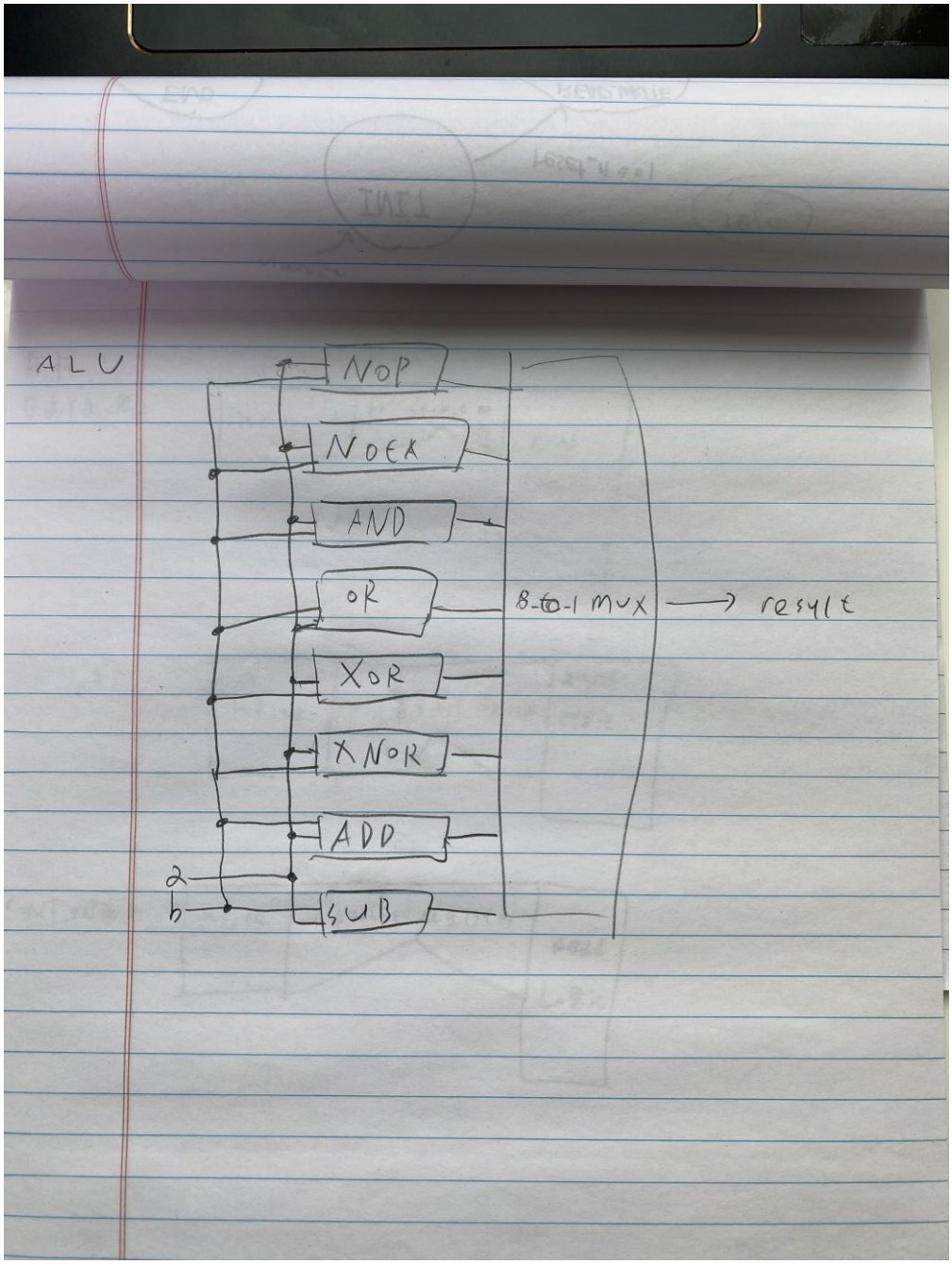
또, BUS의 decoder의 경우, offset이 0x01??인지 아닌지 확인한다. 만약 offset이 0x01??라면 s_sel이 1이고, 아니라면, s_sel = 0이다. s_sel의 값의 따라 m_din 값이 정해진다.

또, m_req의 값이 0이 아니라면 프로그램이 일시정지 되게끔 설계해야 한다.

III. Design Details

각 component 별 pin description, block diagram, FSM 기재
ALU
pin description
a: Ra에 저장되어 있는 값을 가져온다.
b: Rb에 저장되어 있는 값을 가져온다.
op: OPCODE에 따라 어떤 기능을 수행할지 정한다.
ALU_result: ALU의 최종 결과 값을 나타낸다.

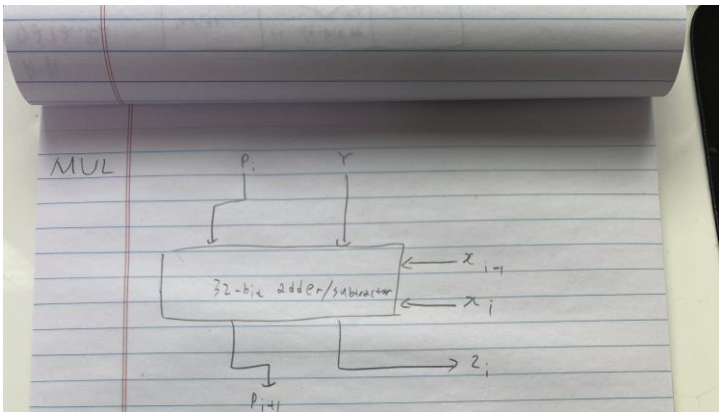
block diagram



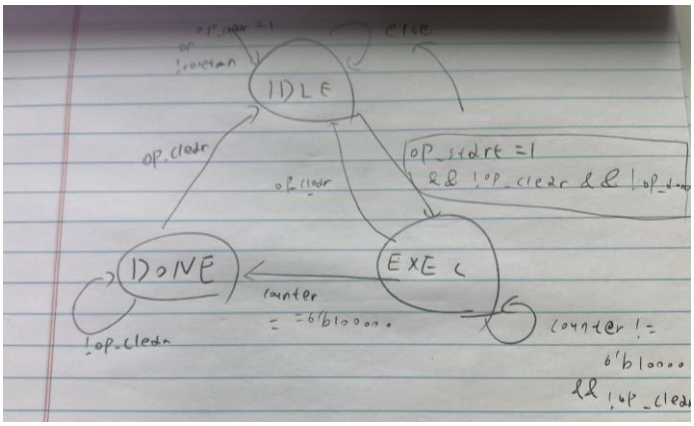
FSM
존재하지 않는다.

MUL
pin description
clk: clock
reset_n; Active low reset
multiplier: 승수
multiplicand: 피승수
op_start: start operation
op_clear: Clear operation
op_done Done operation
MUL_result: Multiplier result

block diagram

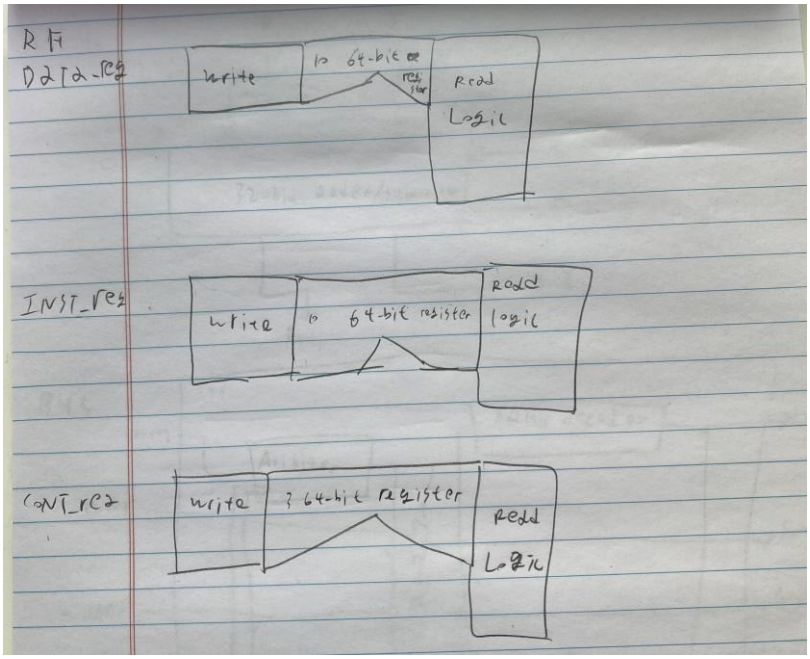


FSM

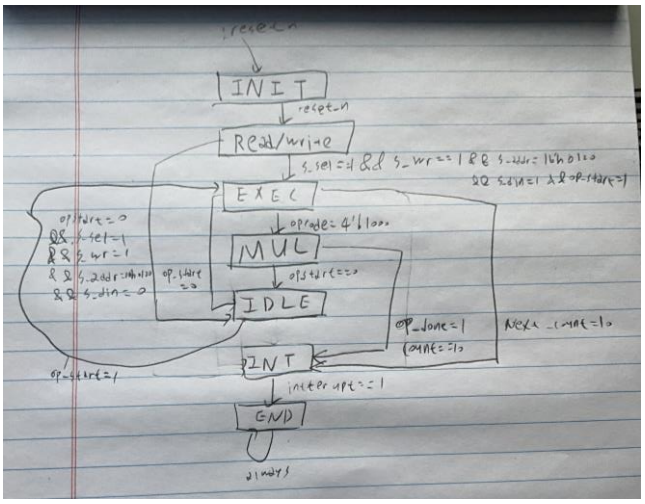


RF
pin description
clk: clock
reset_n: reset_n
S_addr: Slave address
wData: Write Data
we: Write enable
rData: Read Data

block diagram

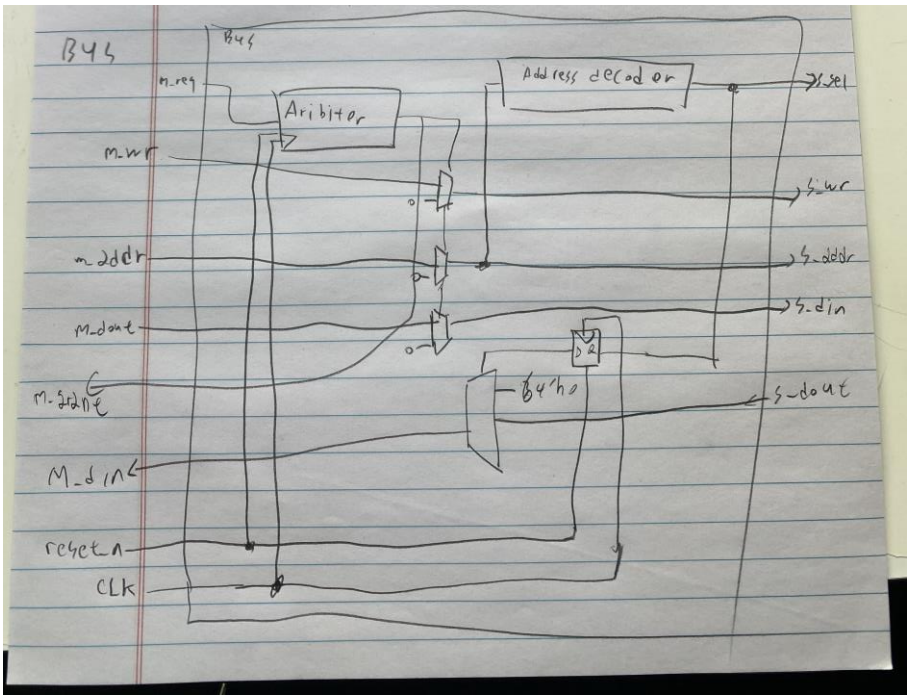


FSM



BUS
pin description
clk: Clk
reset_n: Reset_n
m_req: Master request
m_wr: Master write/read
m_addr: Master address
m_dout: Master data output
S_dout: Slave data out
m_grant: Master grant
m_din: Master data input
s_sel: Slave select
s_wr: Slave write/read
s_addr: Slave address
s_din: Slave data input

block diagram

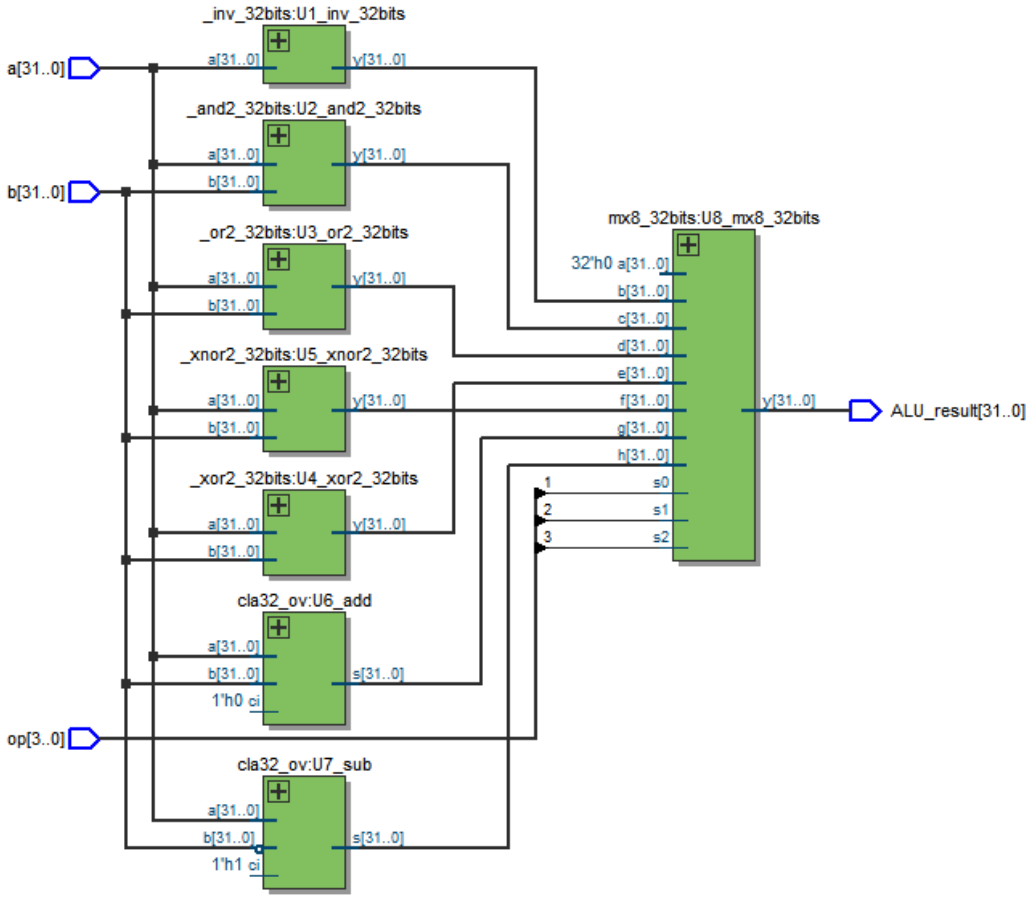


FSM
존재하지 않음.

IV. Design Verifiaction Strategy and Results

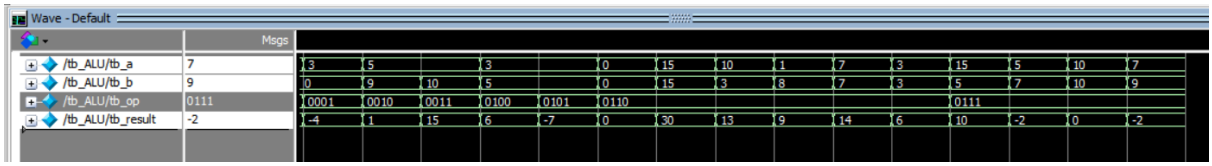
각각의 module별 RTL Viewer는 test라는 프로젝트 창을 만들어 테스트하였다.
또, 각각의 testbench는 기존 과제에서 사용하였던 testbench를 응용한 것이다.
[예: ALU의 모듈 이름을 test로 바꾼 후 진행]

ALU
1. RTL Viewer



OPCODE가 0000일때는 NOP이므로 모듈을 생성하지 않고 32'h0이라는 임의 값을 넣어 testbench를 확인하였다. 또, 기존의 ALU의 경우 CAL flag가 존재했지만, 프로젝트를 진행하면서 삭제하였다.

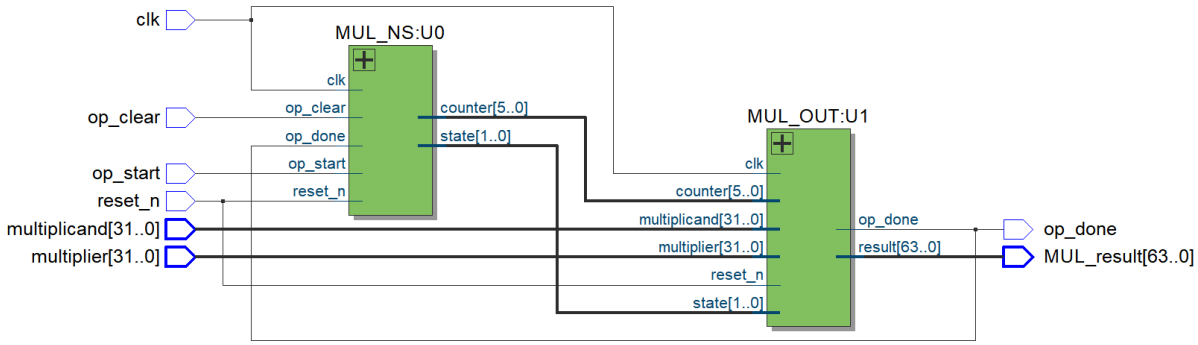
2. testbench



모든 값이 정상적으로 나오는 것을 확인할 수 있다.

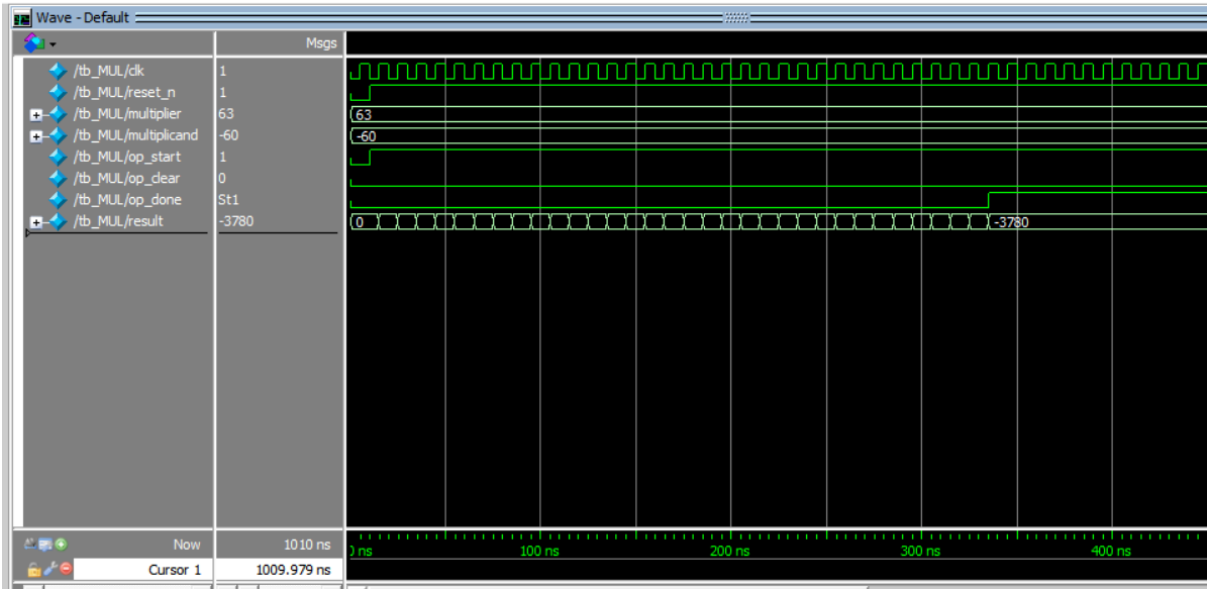
MUL

1. RTL Viewer



Next state를 결정해주는 NS Logic과 Output을 결정해주는 OUT Logic으로 이루어져 있다.

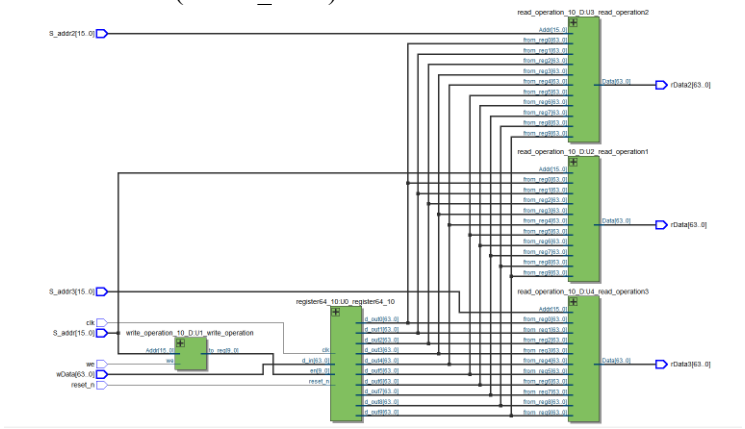
2. testbench



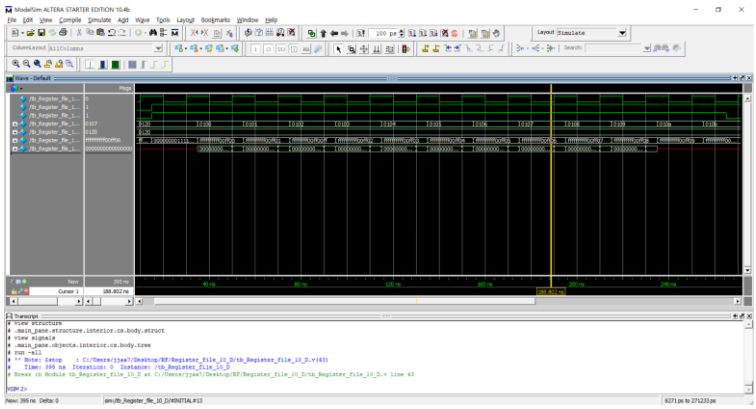
정상적으로 값이 나오는 것을 확인할 수 있다.

RF

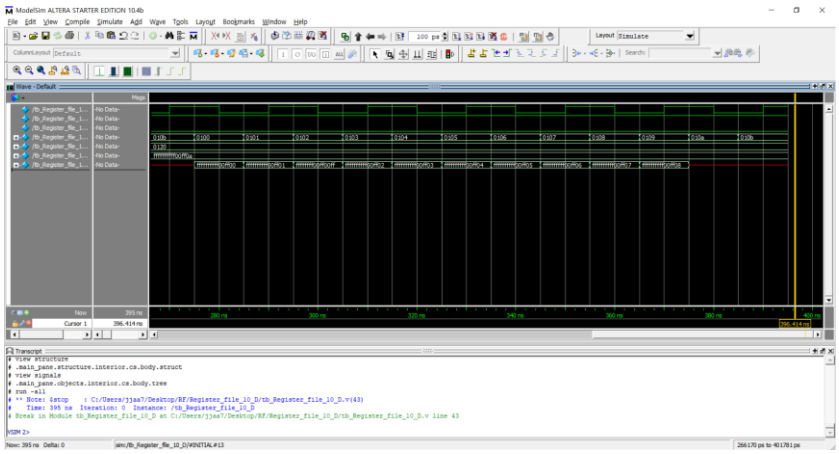
1. RTL Viewer (DATA_REG)



2. testbench (DATA_REG)

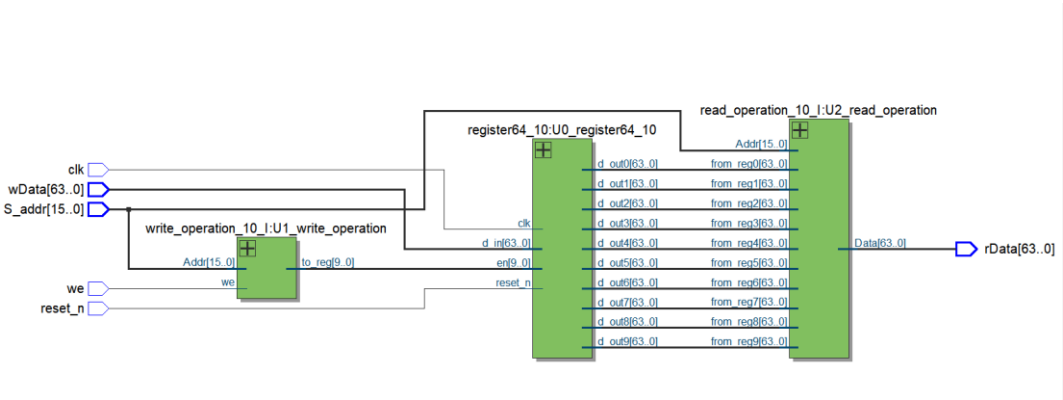


Write의 경우이다. 입력이 진행된 다음 clk이후부터 rData가 나오게 되기 때문에 rData값이 나오게 된다. 또 0x0100 ~ 0x0109 사이의 offset주소가 아니라면 값이 입력되지 않는다. Register_file_10_D의 경우 S_addr과 rData를 3개씩 주어 Ra, Rb, Rd값을 받으려고 하였으나, Register_file_10_D가 Top module일 경우, 핀이 307개로 너무 많이 사용한다는 이유로 Logic이 올바르더라도 컴파일이 되지 않는 문제가 있어, testbench를 진행할 때는 S_addr과 rData를 하나씩 해서 컴파일했다.

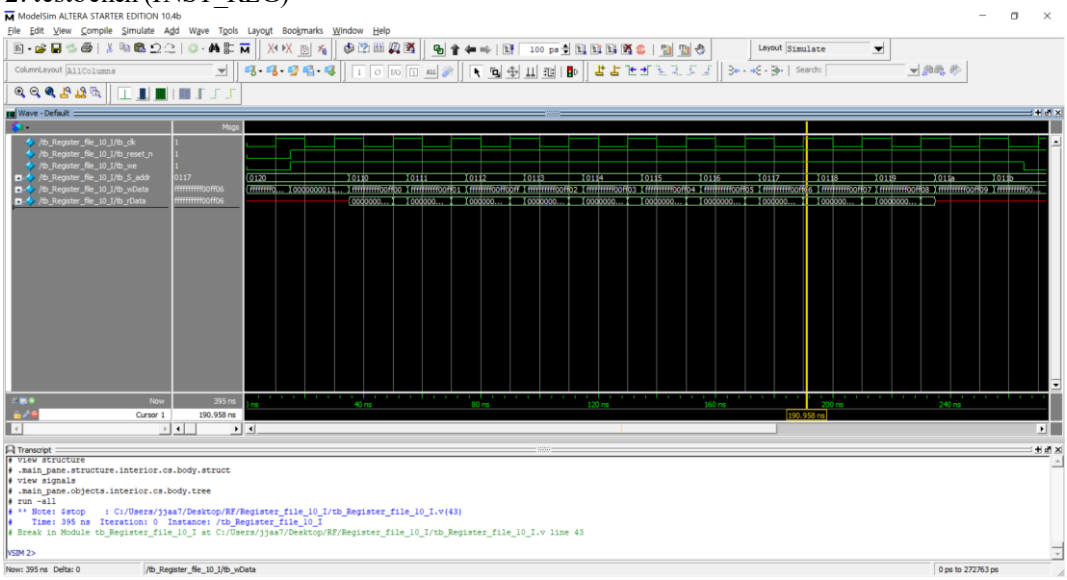


Read의 경우이다. Write에서 저장된 값이 read에서 정상적으로 출력되는 것을 확인할 수 있다. Read와 Write의 차이는 we값이 0이면 Read, 1이면 Write가 진행된다.

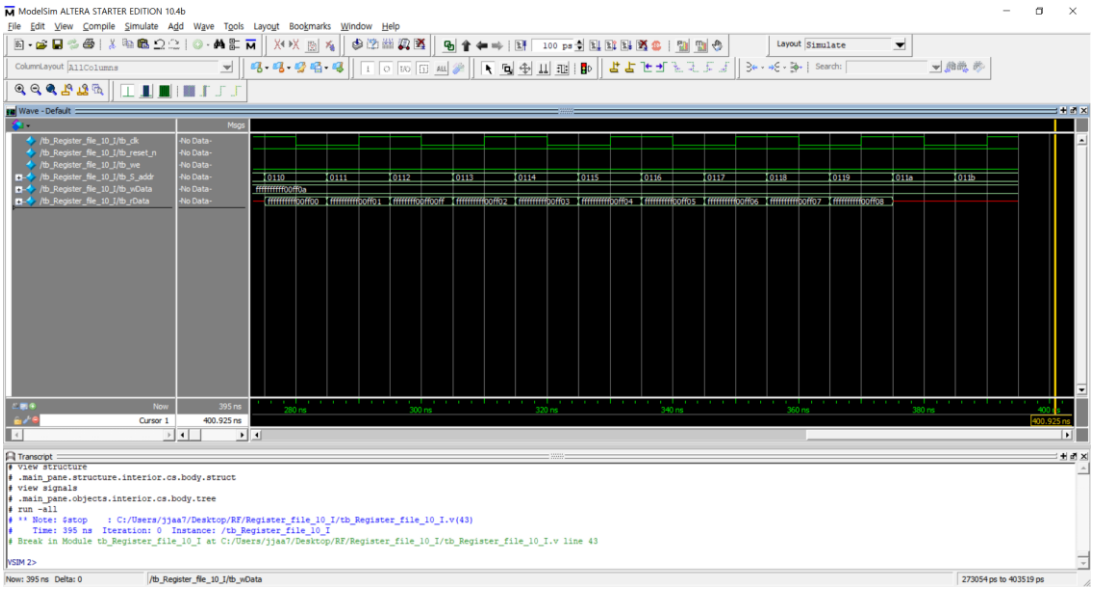
1. RTL Viewer (INST_REG)



2. testbench (INST_REG)

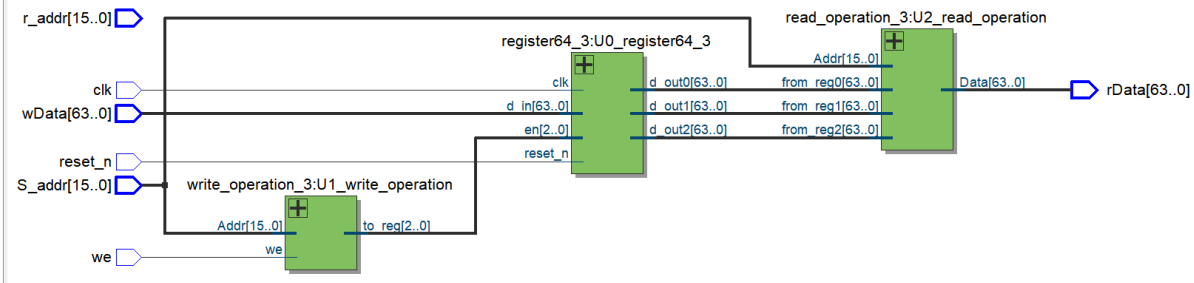


Write의 경우이다. 입력이 진행된 다음 clk이후부터 rData가 나오게 되기 때문에 rData값이 나오게 된다. 또 0x0110 ~ 0x0119 사이의 offset주소가 아니라면 값이 입력되지 않는다.



Read의 경우이다. Write에서 저장된 값이 read에서 정상적으로 출력되는 것을 확인할 수 있다. Read와 Write의 차이는 we값이 0이면 Read, 1이면 Write가 진행된다.

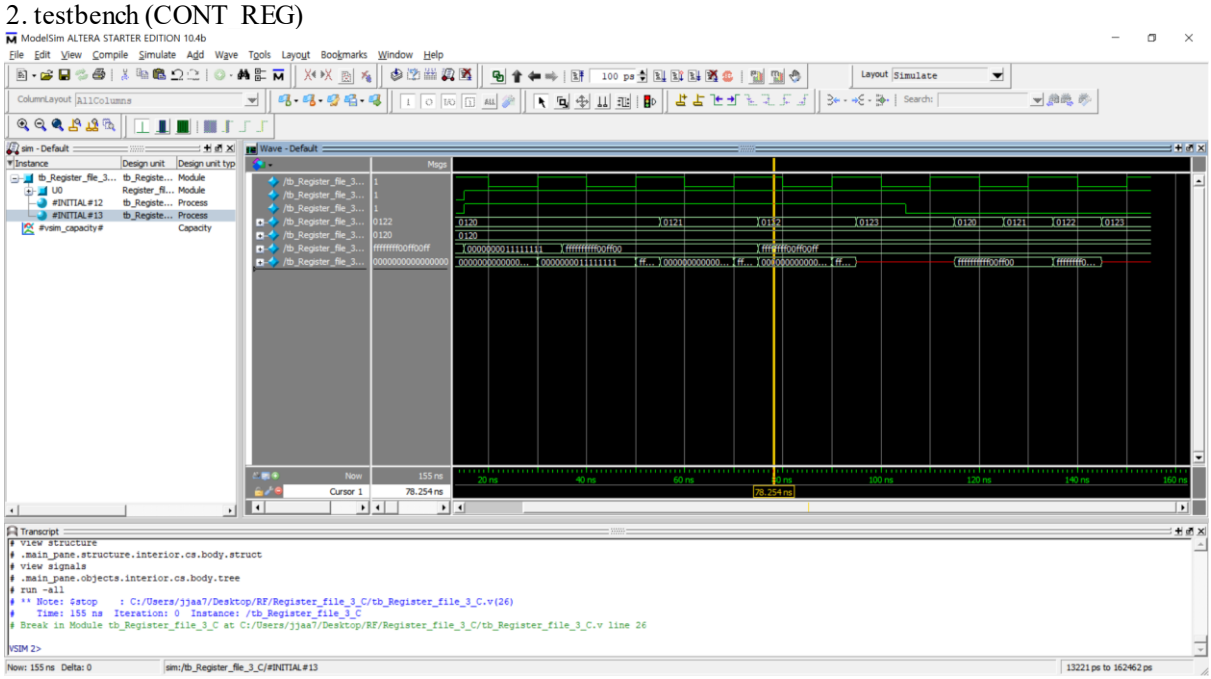
1. RTL Viewer (CONT_REG)



r_addr은 의미 없는 input이다.(실제로 프로젝트에서는 제거 하였다.)

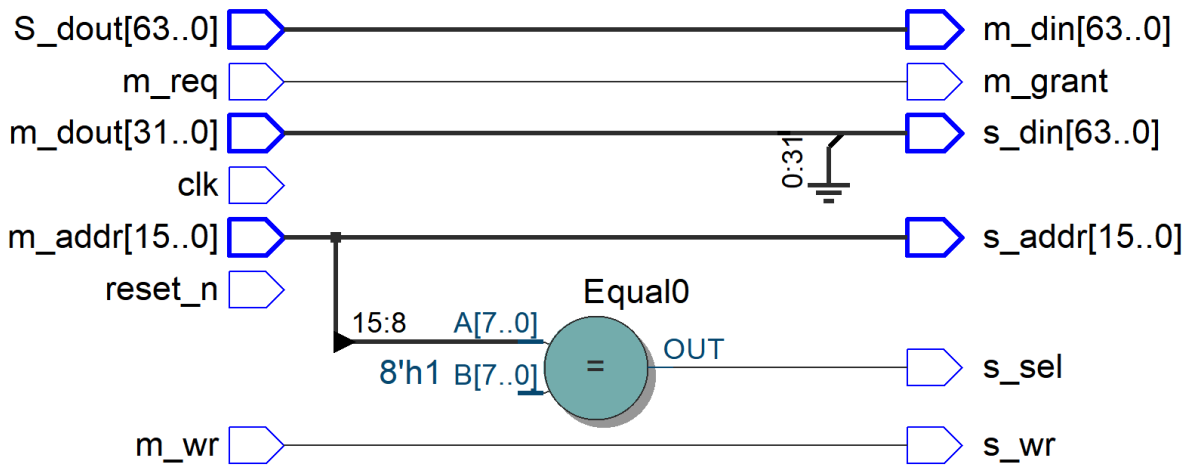
we값으로 read상태인지 write상태인지 제어할 수 있다. we에 s_wr이 들어오면, read/write가 동작할 수 없게끔 설계한다.

2. testbench (CONT_REG)



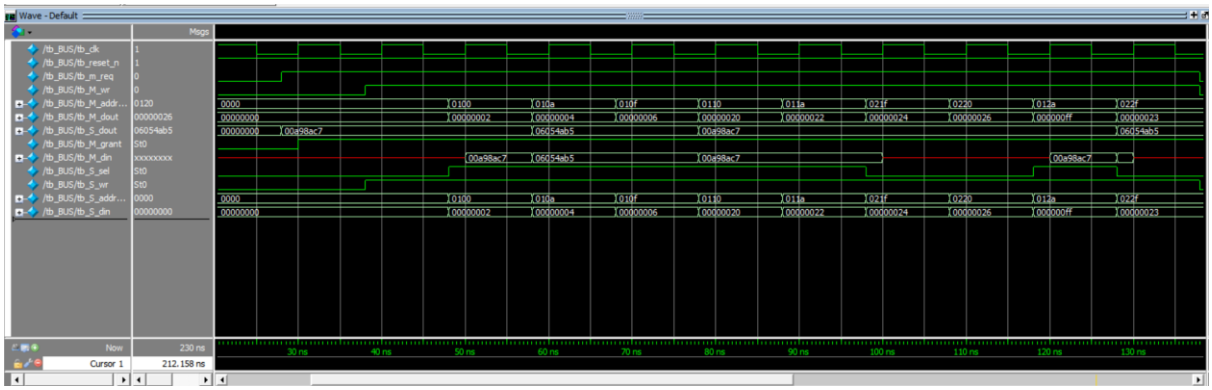
read상태는 125ns이후의 상태이다. write에서 입력된 값이 정상적으로 출력되는 것을 확인할 수 있으며 만약 0x0123처럼 올바르지 않은 주소일 경우 예외처리로 x값이 들어간다.

BUS
1. RTL Viewer

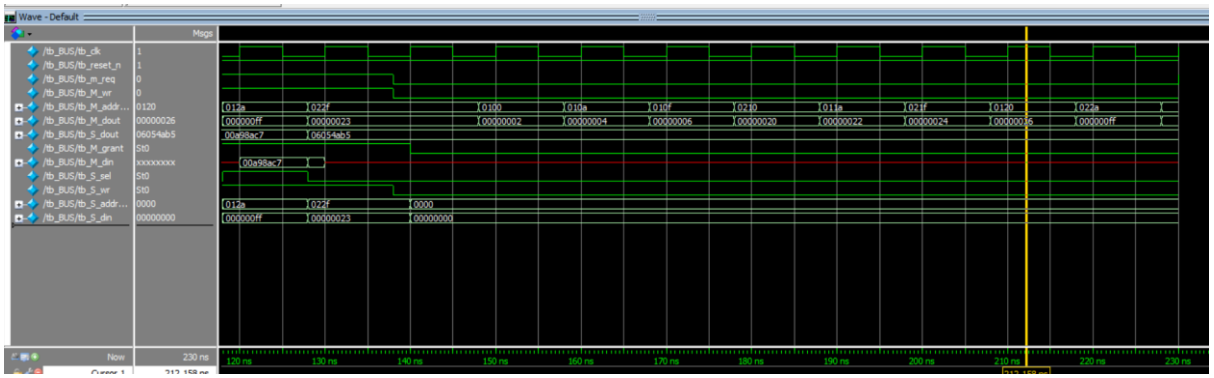


지난 과제와 구조가 거의 비슷한 것을 확인할 수 있다.

2. testbench



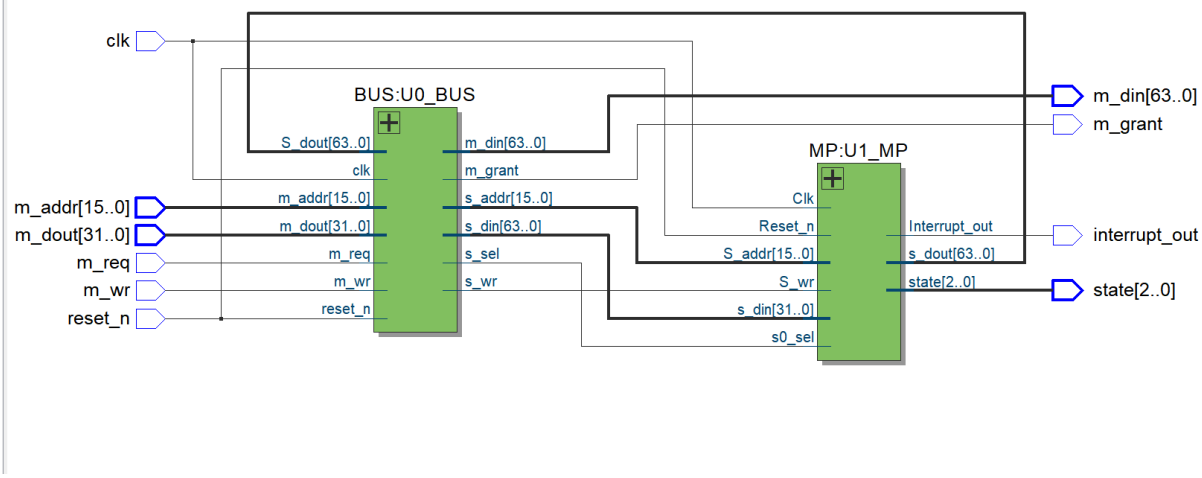
m_req = 1이고, M_din이 x값으로 나왔을 경우는 S_sel이 0일 때 값이다. 프로젝트르 진행할 때는 32'bx가 아닌, 32'b0으로 값을 교체했다. 정상적인 주소일 경우에만, master data input이 output으로 나올 수 있는 걸 확인할 수 있다.



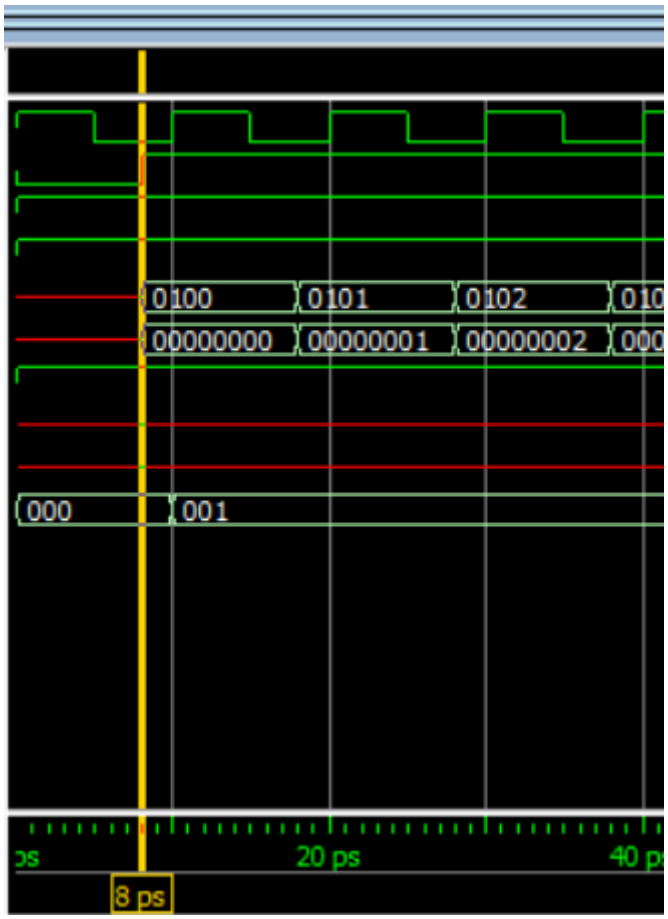
m_req=0일 때는 모든 값이 전달되지 않도록 설정했다. 즉 프로그램이 일시적으로 멈춘다.

TOP

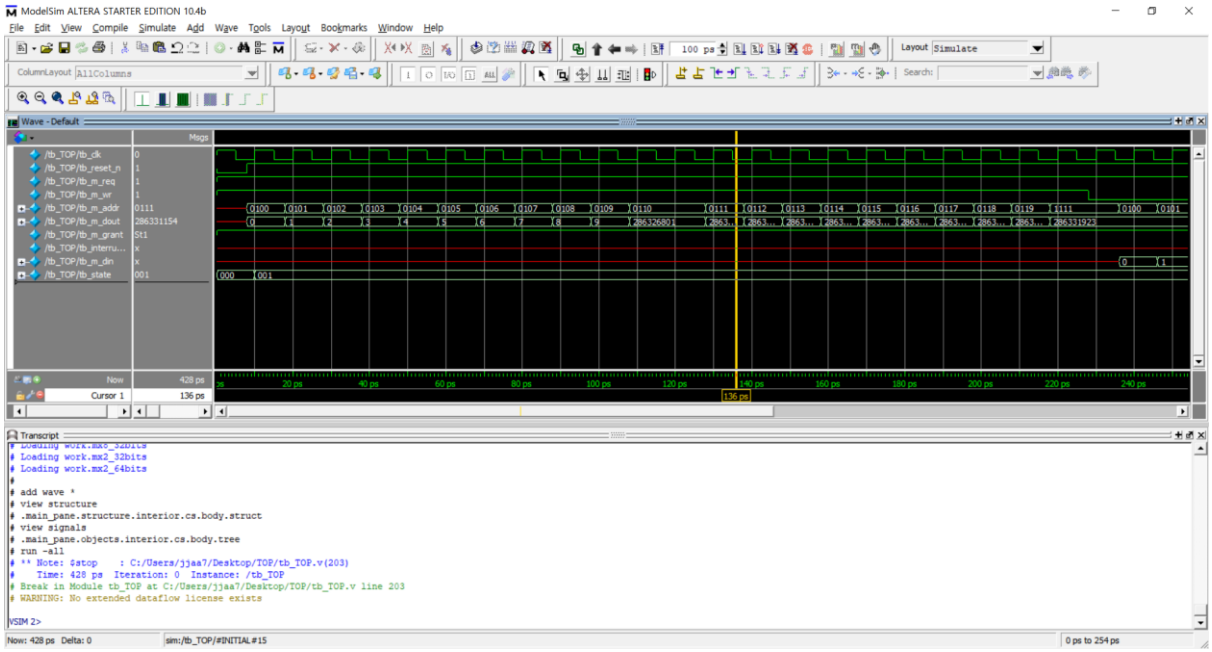
1. RTL Viewer



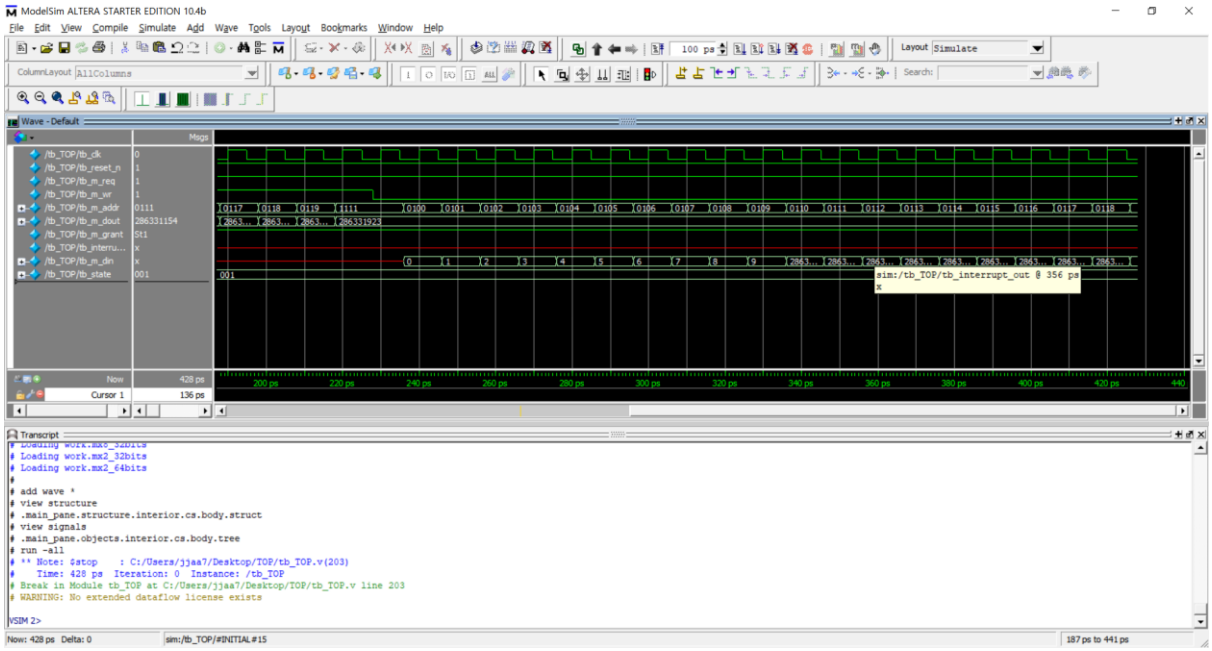
2. testbench



state가 0에서 1로 정상적으로 넘어 감.



write의 경우 데이터가 정상적으로 입력하였음.

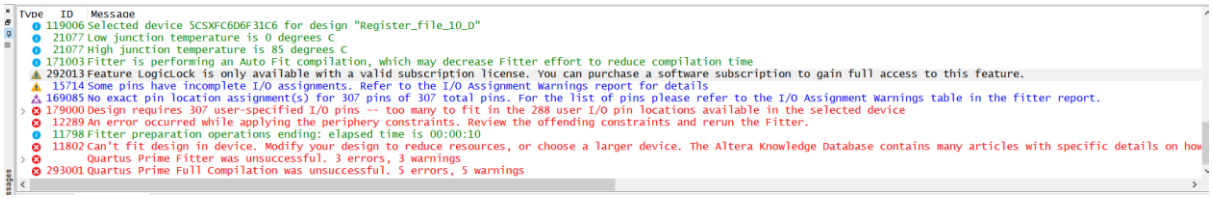


Read의 경우 Write에서 입력한 값을 정상적으로 가져옴.

V. Conclusion

결론 및 고찰

1. DATA_REG가 개인적으로 잘 돌아가는 것을 확인하기 위해 개인 모듈을 돌려보았을 때 Pin 값이 307이 나와 Logic상으론 정상적이지만, pin이 너무 많아 컴파일에 실패했었다. 이 경우 RTL Viewer는 출력되지만, Testbench는 작동하지 않는다.



2. RF를 컨트롤하기 위해, FSM을 제작하였다. FSM의 경우 MUL에서 값이 나오는 경우 32cycle을 대기해줘야 했는데, 그 FSM을 짜는데 대단히 어려웠다. 또, INIT와 IDLE이 서로 다르다는 것을 인지했다.

3. BUS는 W/R신호와 offset이 0x01??인지, req신호가 들어왔는지에 영향을 주며, MP는 BUS에서 받은 signal에 따라 연산 및 RF에 읽기 쓰기를 담당한다.

4. TOP이외에 다른 폴더들을 확인한다면, 각각의 testbench파일이 존재한다. ALU, MUL, RF,BUS 각각 Top 모듈을 주어 따로따로 확인한 경우에는 모두 정상적으로 돌아가는 것을 확인했다. 그러나 next_state와 output_state를 마지막까지 작성하지 못해 값이 정상적으로 들어오고 나가는 것까지 만들었다.

5. 제작 순서는 다음과 같다.
ALU변경 -> MUL변경 -> BUS변경 -> RF변경 -> MP제작 -> TOP testbench 제작

완료하지 못한 단계로는 MP의 state를 정해주는 next_state와 output_state를 마저 만들어야 한다. FSM의 경우 INIT, Read/Write, EXEC, MUL, IDLE, INT, END가 존재하지만, 현재 Read/Write까지만 제작하였으며, 만약 EXEC가 제작되었다고 가정한다면 MUL의 cycle이 32회 걸리므로 연산과정에서 ALU로 연산을 해야 하더라도 32cycle을 지연시킨 후 값이 나온 이후에 2 to 1 MUX로 ALU의 result와 MUL의 result 중 하나를 선택해 값을 Rd에 저장하게 하면 된다.
이후 IDLE에선 op_start가 0일 때 이므로, 계산을 일시정지 하도록 제작하며 INT는 interrupt 값을 정해주는 INT_MASK[0] & INTERRUPT[0] 값에 따라 END로 넘어가주면 되며, END의 경우 END state를 무한하게 유지해주면 된다. END state가 유지되는 동안, 10번이 계산이 완료된 값을 출력하면 된다.(DATA_REG 10번 호출)

VI. Reference

[1] References should be given in this section.
[2] D. M. Harris and S. L. Harris, Digital design and computer architecture, Morgan Kaufmann, 2007 (This is an example.)
[3]컴퓨터 공학 기초 실험실2 강의자료실