

# 컴퓨터 공학 기초 실험2 보고서

실험제목: Term Project

BUS & Memory & ALU with Multiplier

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

## 1. 제목 및 목적

### A. 제목

Term Project (BUS & Memory & ALU with Multiplier)

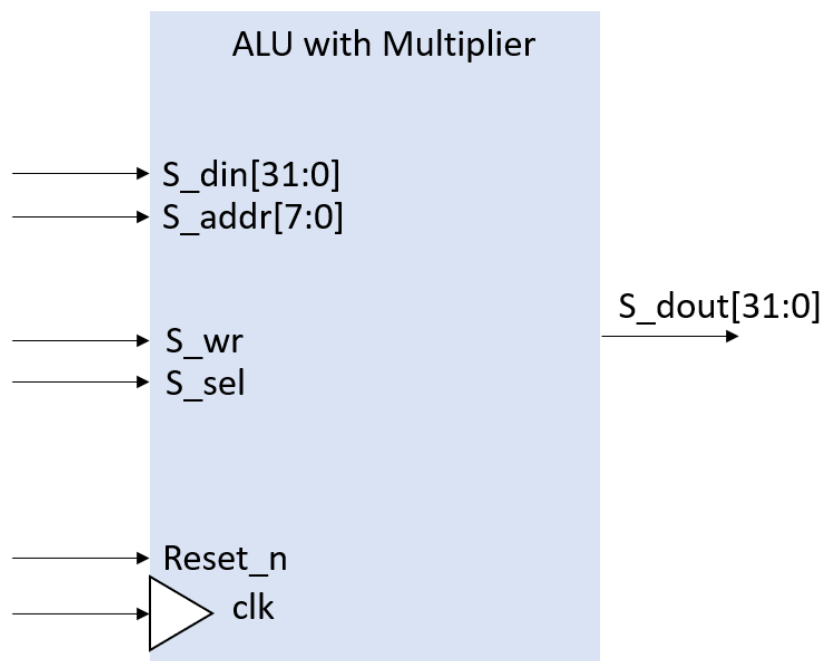
### B. 목적

Testbench가 BUS를 통해 ALU with Multiplier에 명령을 내리고, 그 결과를 memory에 저장하는 디지털시스템을 설계하고 검증해본다. Testbench는 ALU with Multiplier, BUS, Memory로 구성되어 있고, testbench를 이용하여 시스템의 동작을 제어한다.

## 2. 원리(배경지식)

### A. ALU with Multiplier

ALU with Multiplier는 operandA 혹은 operandA 와 B를 이용한 산술, 논리 연산 결과 값을 도출하는 하드웨어이다. 먼저 ALU는 CPU에서 연산을 담당하는 unit으로 Arithmetic Logic Unit의 약자이다. ALU에서는 덧셈 연산, 뺄셈 연산과 같은 산술연산과 AND, OR, NOT A, NOT B, XOR, XNOR과 같은 논리 연산을 한다. ALU는 Operand와 Operator로 구성되는데, Operator는 Opcode로 식별한다. Opcode를 통해 ALU는 지정된 연산이 무엇인지 파악하며 지정된 연산을 수행한 뒤, 연산의 결과를 출력 레지스터 중 하나에 저장한다.



다음으로 multiplier에는 크게 Binary와 Booth 두 가지가 있다. Binary multiplication은 아래의 사진과 같다.

A 0010(2) Multiplicand	
X 0110(6) Multiplier	
0000	) Partial products
0010	
0010	
0000	
0001100 (12)	

Binary Multiplication은 승수의 값이 0인 경우 0을, 1인 경우 피승수에 해당하는 값을 shift해 bit수에 맞춰 적어준다. 그 후 모든 값을 더해주어 곱셈의 결과를 얻는다. 위의 사진에서는 0010(2)과 0110(6)을 곱해 0001100(12)이라는 값을 얻은 것을 확인할 수 있다. 하지만 bit 수가 많은 계산의 경우 계산과정이 길어지며 비효율적이다.

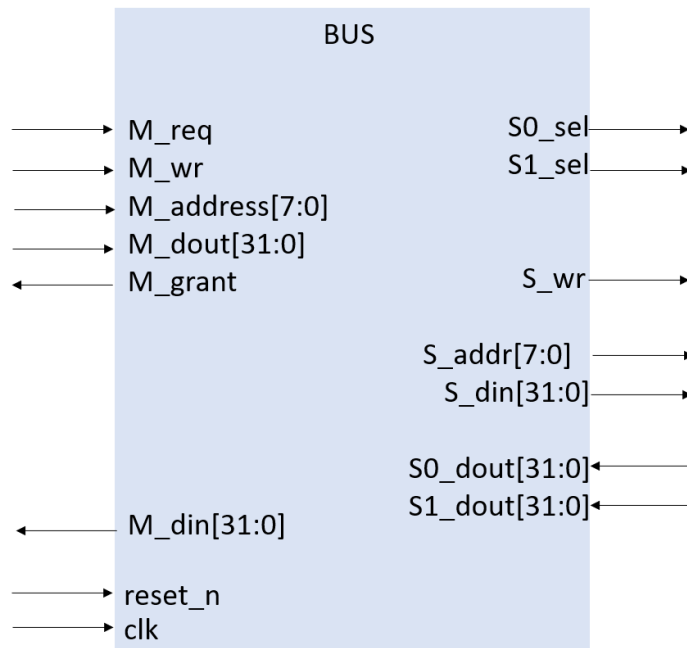
Booth Multiplication은 signed binary number의 곱을 산출하는 곱셈기이다. 아래의 사진은 0010(2)와 0110(6)의 연산과정을 나타낸 것이다. 먼저 Y값은 승수 즉 X에 LSB의 오른쪽에 0이 있다고 가정한다. 00은 0, 10은 -1, 11은 0, 01은 1이므로 Y는 10(-1)0이다. 먼저 Y가 0일 때 shift이므로, 0000에서 shift해 00000이 된다. 다음으로 Y가 1일 때 subtract shift를 진행하면 subtract는 0010->1110이고, 00000과 더해지면 11100이 된다. 그 후 shift 해주면 111100이된다. Y가 0일 때 shift이므로 1111100이며, 마지막으로 Y가 1일 때 add shift를 진행하면 1111100에 A를 더해지면 0001100이고, shift해주면, 00001100이 된다. 00001100은 12이므로 알맞은 결과가 나온 것을 알 수 있다.

A 0010(2) Multiplicand	
X 0110(6) Multiplier	
Y 10 $\bar{1}$ 0	
shift 00000 ( $y_0 = 0$ )	
Add -A +1110 ( $y_1 = \bar{1}$ )	
11100	
Shift 111100	
Shift 1111100 ( $y_2 = 0$ )	
Add A +0010	
0001100	
shift 00001100 (12) ( $y_3 = 1$ )	

$X_i$	$X_{i-1}$	Operation	Description
0	0	Shift only	String of zeros
0	1	Add and shift	End of a string of ones
1	0	Subtract and shift	Beginning of a string of ones
1	1	Shift only	String of ones

위 표는 radix-2의 Booth Multiplication의 규칙이다.

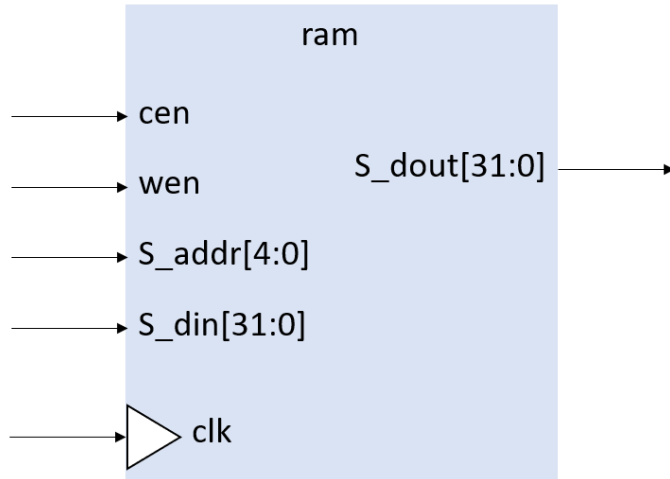
## B. Bus



BUS는 여러 컴퓨터 내부 회로에서 CPU인 중앙 처리 장치와 주기억장치, I/O component 들 간에 data를 전송할 수 있도록 연결해주는 component이다. BUS는 크게 데이터 버스, 주소 버스, 제어 버스로 나뉜다. 데이터 버스는 시스템 모듈 사이의 데이터의 이동 경로를 제공한다. 주소 버스는 데이터의 근원지나 목적지의 일정한 메모리 주소를 전달하는 버스를 말한다. 마지막으로 제어 버스는 데이터 버스와 주소 버스를 제어하기 위해 사용한다. Operation에는 Read와 Write가 있다. BUS에는 master와 slave가 있는데 master는 bus를 통해서 데이터를 전송하라고 요청하거나 시작하는 역할을 하며, slave는 요청된 데이터의 전송을 처리하는 역할을 한다. Bus의 내부에는 Arbiter, 중재자가 있으며 bus master를 조절하는 역할을 한다. 즉 여러 Bus master들이 서로 bus를 장악하려고 할 때 특정 bus master에게 bus를 제공해준다. Bus는 새로운 component들을 추가하기가 쉬우며, 가격이 저렴하다는 특징이 있다.

## C. Memory

Memory는 크게 휘발성(Volatile)과 비휘발성(Non-Volatile)로 나뉘는데, 각각의 특성에 따라 ROM과 RAM으로 나뉜다. ROM은 Read Only Memory를 뜻하며, RAM은 Random Access Memory를 뜻한다. 그 중 이번 실습에서는 Ram을 설계했다.



RAM은 Address에 기반하여 데이터를 저장하는 hardware로 프로그램이 실행되는 동안 기억된 정보를 읽거나 다른 정보를 기억할 수 있는 메모리이다. RAM은 휘발성 Memory이기 때문에 전원이 꺼지면 가지고 있던 데이터가 사라지게 된다. 또한 미리 정해진 순서로만 저장 미디어에 있는 데이터를 액세스 할 수 있다는 특징이 있다. ROM은 RAM과는 반대로 비휘발성 Memory이므로 전원이 꺼져도 데이터가 사라지지 않는다. Ram의 종류에는 SRAM, DRAM, SDRAM 등이 있다. SRAM은 Static Ram으로 전원이 공급되는 한 저장된 데이터가 지워지지 않는다. 또한 DRAM보다 일반적으로 속도가 빠르지만 용량이 작고 가격이 비싸다. DRAM은 Dynamic RAM으로 휘발성 메모리이다. SRAM보다 가격도 싸고 용량이 크지만 대비 속도가 느리다. 또한 DRAM은 capacitor를 통해 데이터를 저장하는 방식을 사용한다.

### 3. 설계 세부사항

#### - Introduction

Testbench가 BUS를 통해 ALU with Multiplier에 명령을 내리고, 그 결과를 memory에 저장하는 디지털시스템을 설계하고 검증해본다. ALU with Multiplier, BUS, Memory로 구성되어 있고, testbench를 이용하여 시스템의 동작을 제어한다. ALU with Multiplier는 BUS를 통해 접근할 수 있는 register 집합을 가지고 있다. 이 register를 통해 외부 모듈과 데이터를 주고받을 수 있다. testbench로부터 20개의 data를 memory1에 각각 저장한다. Testbench에 지정된 20개의 execution data set을 이용하여 ALU with Multiplier를 이용해 결과를 연산한다.

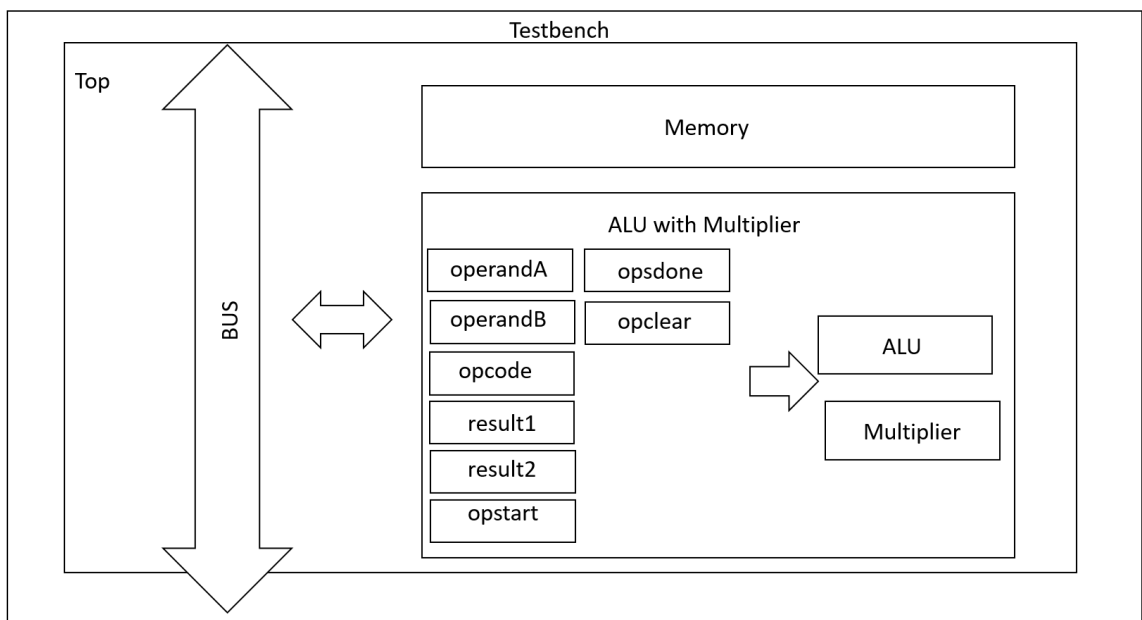
- Schedule

	11주차	12주차	13주차	14주차
제안서				
코드 작성				
코드 검증				
결과 보고서				

전체적인 흐름은 계획에 맞게 잘 진행하였다. 코드 작성과 코드 검증은 동시에 진행하였다.

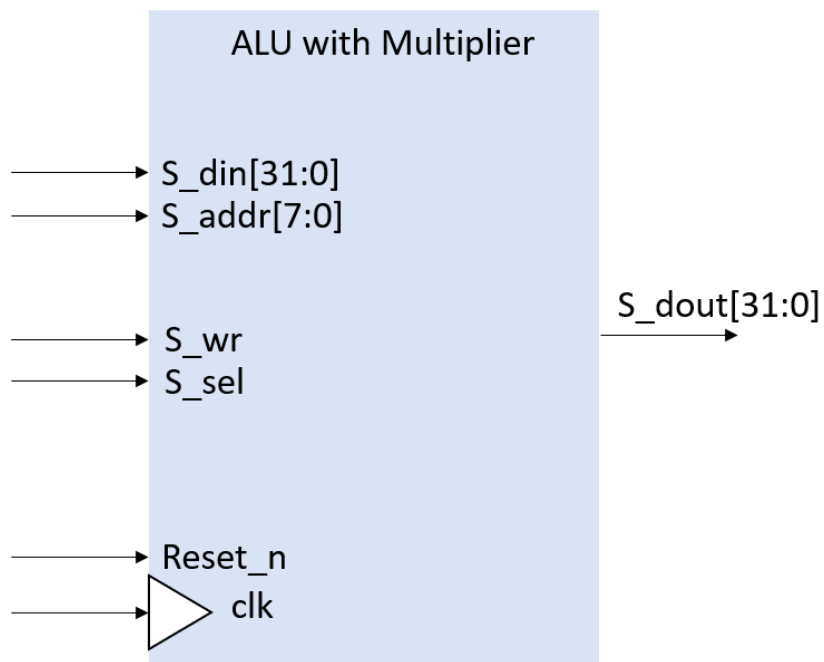
- Design details

A. Top



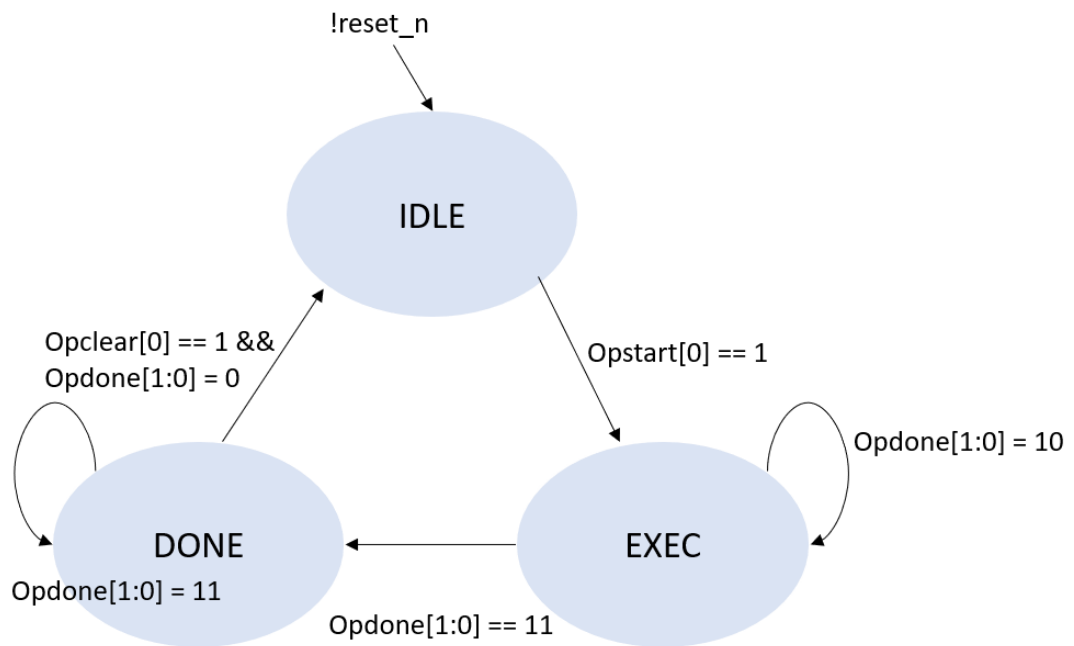
각 module이 Top-level module에서 어떻게 instance 되는지 도식화한 그림이다. Testbench가 BUS를 통해 ALU with Multiplier에 명령을 내리고, 그 결과를 memory에 저장하도록 한다. BUS, ram, ALU with Multiplier로 구성되어 있다.

B. ALU with Multiplication



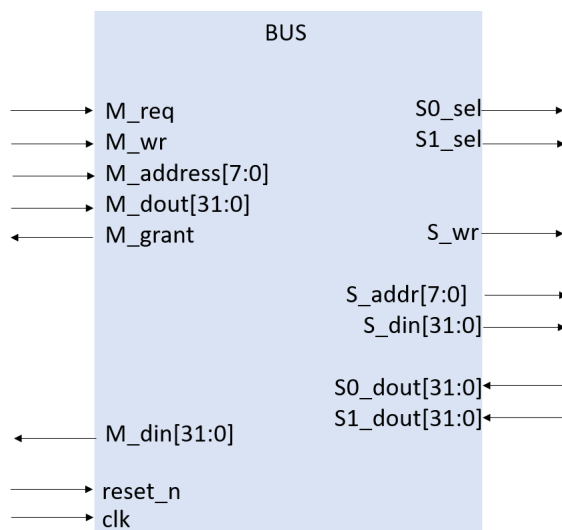
Direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset
	S_sel	1	Select
	S_wr	1	Write/read
	S_addr	8	Address
	S_din	32	Data input
Output	S_dout	32	Data output

ALU with Multiplier의 pin description과 pin을 구조화시킨 것이다. ALU with Multiplier는 multiplier와 alu32, register 8개(operandA, operand, opcode, opstart, opdone, opclear, result1, result2)로 구성되어 있다.



ALU with Multiplier의 State transition diagram이다. `opdone[1:0] == 2'b00`일 때 연산 대기, `opdone[1:0] == 2'b10`일 때 연산 시작, `opdone[1:0] == 2'b11`일 때 연산 완료를 해준다.

### C. BUS

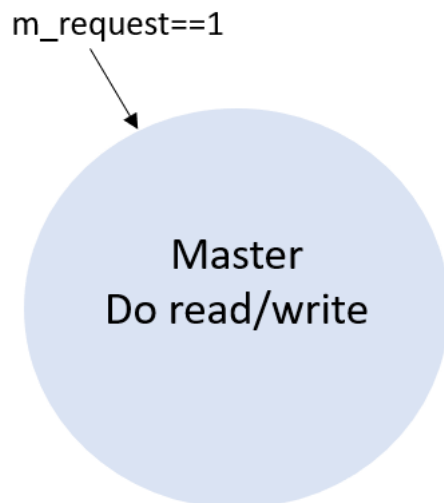


Direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset



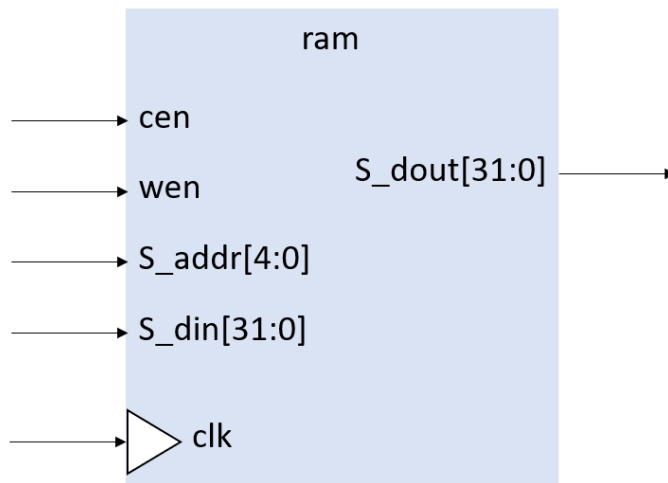
	M_req	1	Master request
	M_wr	1	Master write/read
	M_addr	8	Master address
	M_dout	32	Master data output
	S0_dout	32	Slave 0 data out
	S1_dout	32	Slave 1 data out
Output	M_grant	1	Master grant
	M_din	32	Master data in
	S0_sel	1	Slave0 select
	S1_sel	1	Slave1 select
	S_addr	8	Slave address
	S_wr	1	Slave write/read
	S_din	32	Slave data input

BUS의 pin description과 pin을 구조화시킨 것이다. Master는 grant에 따라 slave에 값을 넘겨준다. 프로젝트에서는, 1개의 master와 2개의 slave를 가지고 있으며, slave0은 0x00~0x1F까지의 주소 범위를 가지고, slave1은 0x20~0x3F의 주소 범위를 가지며 이 외의 경우는 선택하지 않는다. BUS는 bus\_arbit, bus\_addr, 3 input 32bit mux로 구성되어 있다.



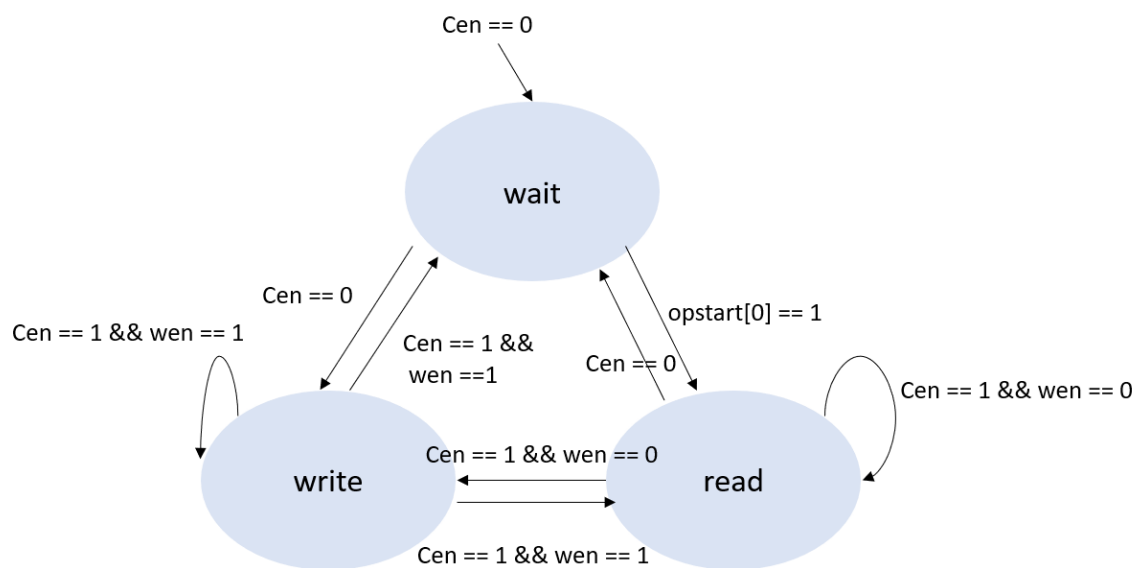
BUS의 State transition diagram이다. Master가 req를 1로 만들면 grant를 통해 가능여부를 알려주고 master의 데이터에 따라 slave에 접근하여 데이터를 주고받게 된다.

#### D. Memory



Direction	Port name	Bit width	Description
Input	clk	1	Clock
	cen	1	Chip enable
	wen	1	Write enable
	S_addr	5	Address
	S_din	8	Data in
Output	S_dout	32	Data output

ram의 pin description과 pin을 구조화시킨 것이다. Cen, wen에 따라 값을 write 혹은 read한다.

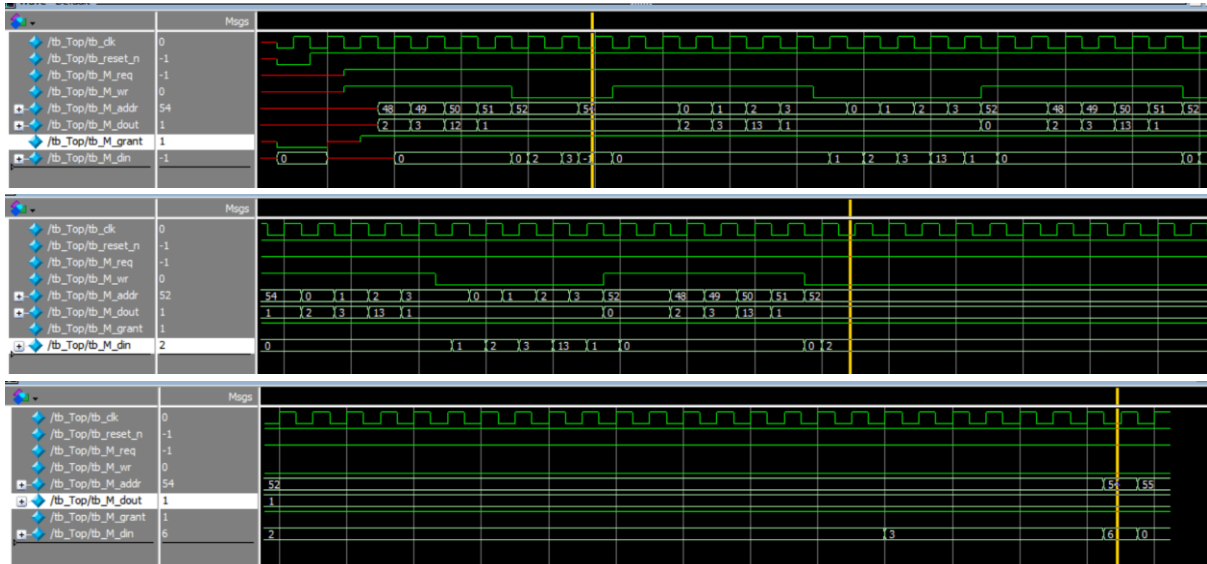


Memory, ram의 State transition diagram이다. Cen, wen에 따라 결정하도록 한다.

#### 4. 설계 검증 및 실험 결과

##### A. 시뮬레이션 결과

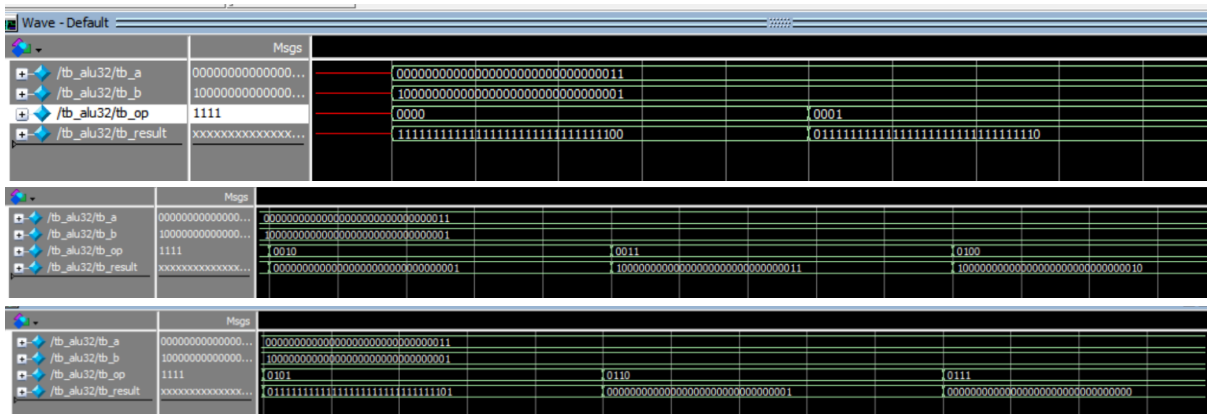
- Top

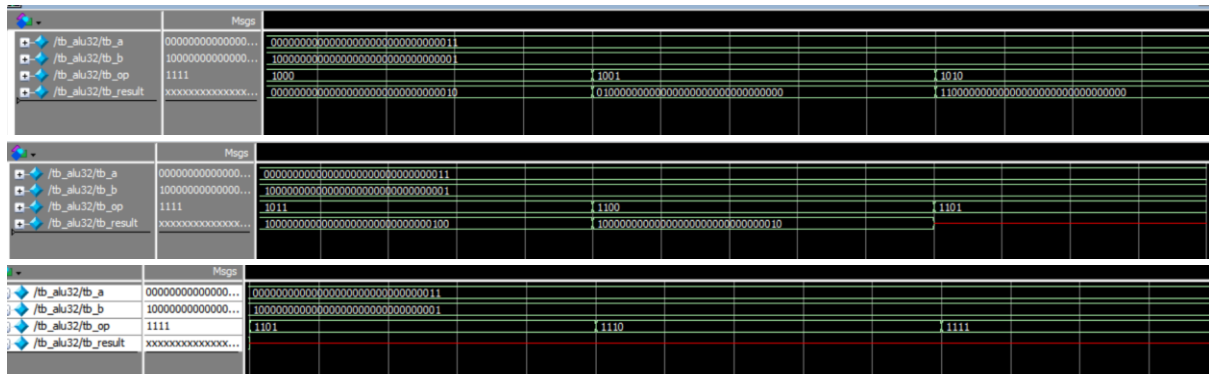


첫번째 사진은 Slave 0인 메모리에 값을 넣은 testbench로 M\_req가 1일 때 2와 3의 sub연산의 결과인 -1이 result1인 54에 잘 저장된 것을 알 수 있다. 이는 아래의 ALUwMUL module의 testbench 값과 동일하다. 두번째 사진은 Slave1인 메모리에 값을 넣은 testbench로 read와 write를 잘 수행하는 것을 알 수 있다. 세번째 사진은 Slave 0인 메모리에 값을 넣은 testbench로 2와 3의 multiplier연산의 결과인 6이 result1인 54에 잘 저장된 것을 알 수 있다. 모든 경우의 수를 testbench에 넣어 확인해보지는 못 했지만 Top의 결과가 잘 나오는 것을 알 수 있다.

\* 첫 번째 사진에서 M\_din의 출력 사이에 빨간 줄이 뜨는 것을 볼 수 있는데, 이는 코드의 문제로 출력이 안되는 것이 아니라, 값을 넣어주지 않아 생기는 것임을 알 수 있다.

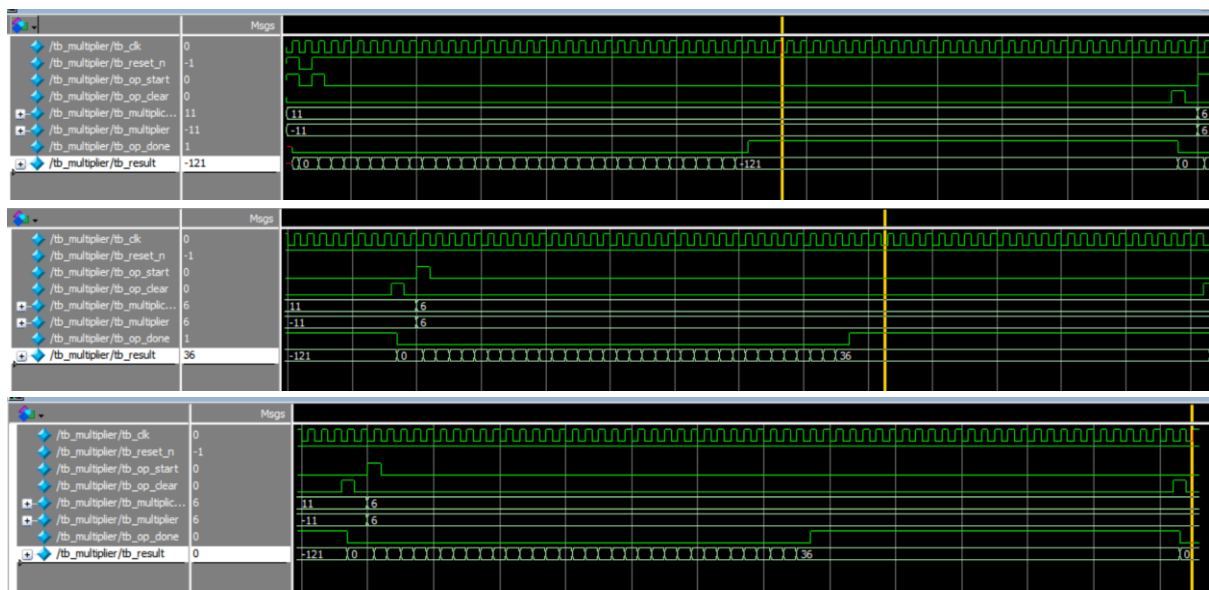
- ALU





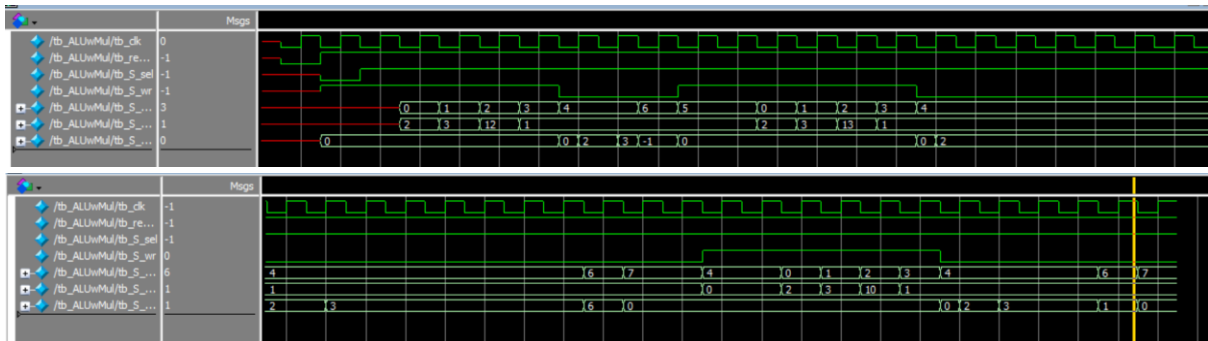
ALU는 앞에서부터 NOT A, NOT B, AND, OR, XOR, XNOR, Set less than, Set greater than, Shift left logical, Shift right logical, Shift right arithmetic, Addition, Subtraction, Multiply의 연산 결과를 보여주는 testbench이다. 각각의 opcode에 따라 연산을 진행하며 결과가 모두 잘 나오는 것을 확인할 수 있다. 주어진 명령어가 아닌 경우 결과를 출력하지 않는 것도 확인할 수 있다.

#### - Multiplier



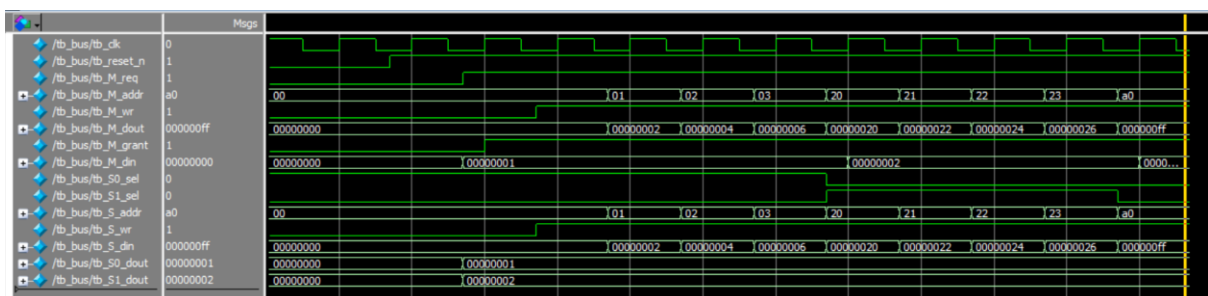
2 radix의 연산과정으로, Multiplier에서는  $11 \times (-11)$ 와  $6 \times 6$ 의 예시를 들어 testbench를 작성하였다. 먼저 첫 번째 사진에서, tb\_result의 값이 32번 바뀌며 32번의 계산 과정을 거친 후 즉 tb\_op\_done이 1일 때,  $11 \times (-11)$ 의 결과 값인 -121이 되는 것을 확인할 수 있다. 두 번째 사진에서, tb\_op\_clear가 1일 때 tb\_op\_done과 tb\_result가 0이 되는 것을 확인할 수 있다. 또한 두 번째 사진과 세 번째 사진에서 볼 수 있듯이,  $6 \times 6$ 의 연산에서는 tb\_op\_done이 1일 때,  $6 \times 6$ 의 결과 값인 36이 되는 것을 확인할 수 있다. 모든 경우의 수를 testbench에 넣어 확인해보지는 못 했지만 Multiplier의 연산 결과가 잘 나오는 것을 알 수 있다.

#### - ALU with Multiplier



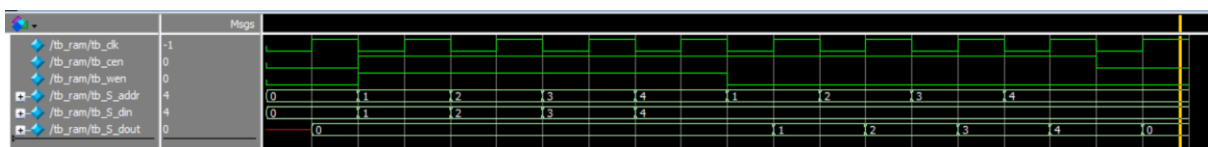
첫번째 사진에서 0, 즉 operandA에 2를, 1, 즉 operandB에 3을 넣어 2, 즉 명령어 12에 해당하는 sub연산을 진행할 때 6, result1에 -1이 저장된 것을 확인할 수 있다. 이어서 operandA에 2, operandB에 3을 넣어 명령어 13에 해당하는 곱셈 연산을 진행할 때, result1에 6이 저장된 것을 확인할 수 있다. 첫 번째 사진은 clear를 통해 초기화를 하고 연산을 진행한 경우이며, 두 번째 사진은 opdone을 통해 초기화를 하고 연산을 진행한 경우이다. 두 번째 사진에서 operandA에 2, operandB에 3을 넣어 명령어 10에 해당하는 shift right arithmetic 연산을 진행할 때, result1에 1이 저장된 것을 확인할 수 있다. 모든 경우의 수를 testbench에 넣어 확인해보지는 못 했지만 ALU with Multiplier의 연산 결과가 잘 나오는 것을 알 수 있다.

#### - BUS



Bus를 가진 master가 주는 address에 따라 sel 신호가 변경된다. 즉 tb\_M\_address가 0x1F를 넘어가는 경우 S0\_sel이 0, S1\_sel이 1이 되는 것을 확인할 수 있으며 tb\_M\_address가 slave0과 slave1의 범위를 넘어가는 경우 0이 출력되는 것을 확인할 수 있다. 따라서 결과가 잘 나오는 것을 확인할 수 있다.

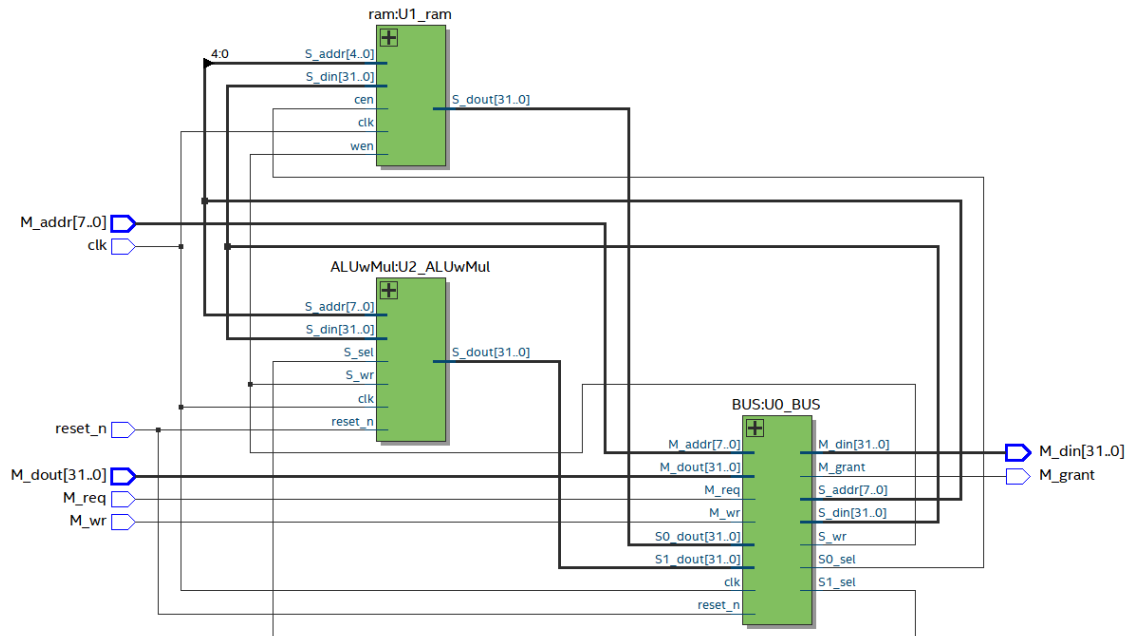
#### - ram



Cen이 1, wen이 1일 때 address가 가리키는 메모리 1, 2, 3, 4에 차례로 din, 1, 2, 3, 4가 write 되었다. Cen이 1이고 wen이 0일 때 address가 가리키는 메모리의 값인 1, 2, 3, 4가 dout에 write 되었다. Cen이 0일 때 dout이 0이 되었다. 따라서 결과가 잘 나온 것을 확인할 수 있다.

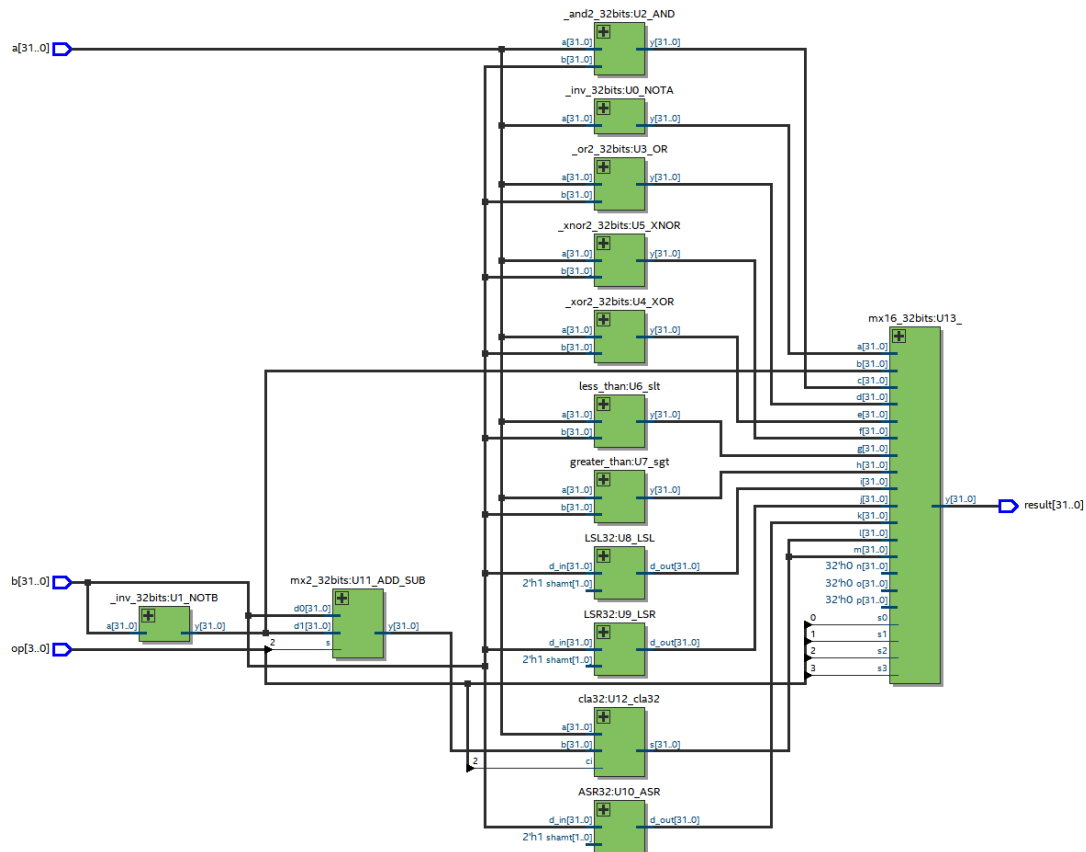
## B. 합성(synthesis) 결과

### - Top



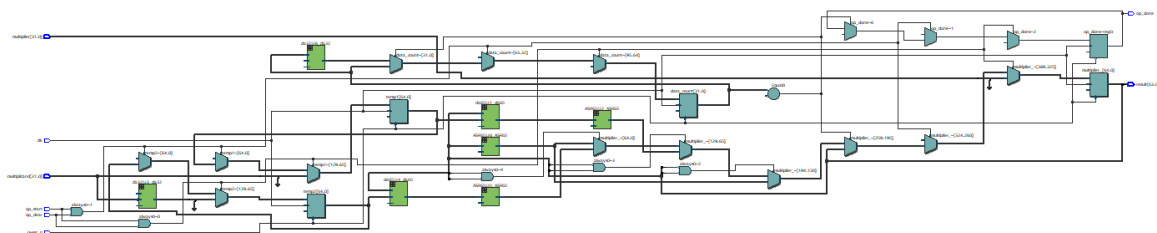
Top의 rtl viewer로, BUS와 ALU with Multiplier, ram이 잘 연결되어 회로가 잘 설계된 것을 확인할 수 있다.

### - ALU



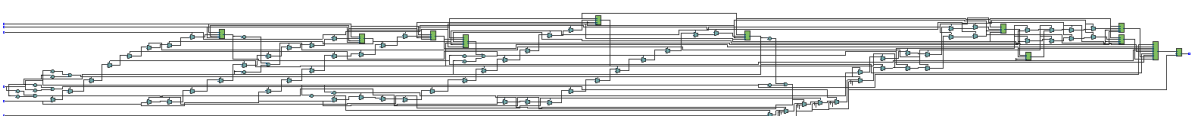
ALU의 rtl viewer로 register들과 register를 연산하는 module이 잘 연결되어 회로가 잘 설계된 것을 확인할 수 있다.

#### - multiplier



multiplier의 rtl viewer로, clk, reset\_n, multiplier, multiplicand, op\_start, op\_clear가 입력되어 op\_done, result가 출력되는 것을 확인할 수 있으며, multiplier하는 과정을 rtl viewer을 통해 잘 설계된 것을 확인할 수 있다.

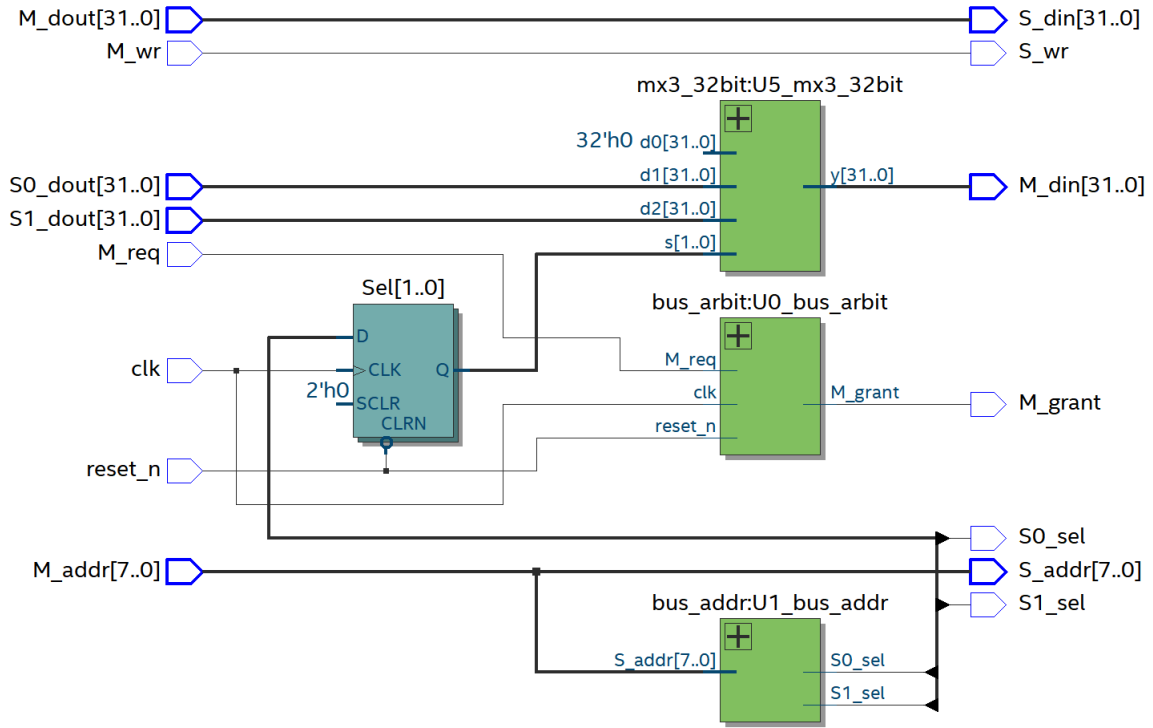
#### - ALU with Multiplier



ALU with Multiplier의 rtl viewer로 register와 alu32, multiplier module들이 잘 연결되어 회

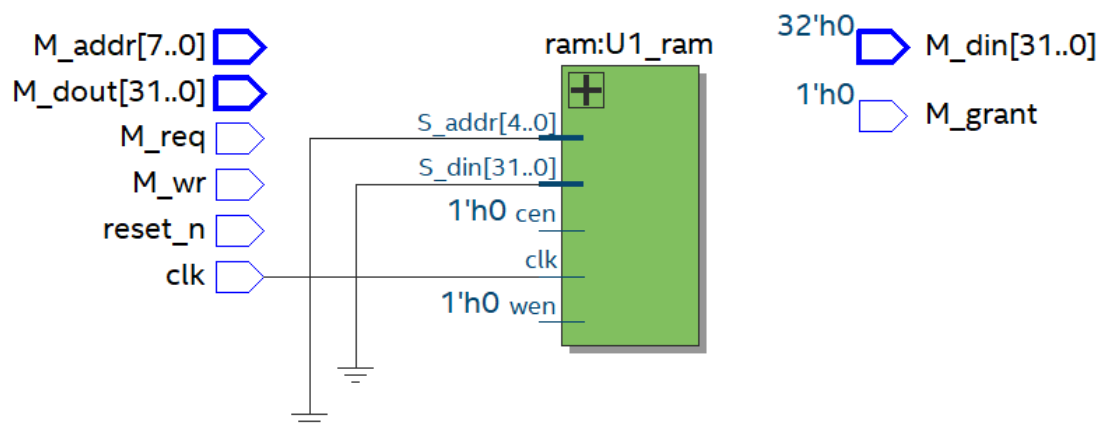
로가 잘 설계된 것을 확인할 수 있다.

- BUS



Bus의 rtl viewer로, arbit와 addr와 mux와 sel이 연결되어 있으며 회로가 잘 설계된 것 확인할 수 있다.


- Ram



Memory, ram의 rtl viewer로, 회로가 잘 설계된 것을 확인할 수 있다.


- Top



Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sat Dec 03 15:45:15 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	Top
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1474
Total pins	77
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Top의 flow summary이다. 총 1474개의 register와 77개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- ALU

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sat Dec 03 01:45:55 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	alu32
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	100
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

ALU의 flow summary이다. 총 100개의 register를 사용한 것을 알 수 있으며 성공적으로 컴파

일이 완료된 것을 알 수 있다.

- Multiplier

Flow Summary	
<<Filter>>	
Flow Status	Successful - Fri Dec 02 23:21:39 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	multiplier
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	160
Total pins	133
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Multiplier의 flow summary이다. 총 160개의 register와 133개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- ALU with Multiplier

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sat Dec 03 14:35:33 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	ALUwMul
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	415
Total pins	76
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

ALU with Multiplier의 flow summary이다. 총 415개의 register와 76개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- BUS

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sat Dec 03 03:19:11 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	BUS
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	3
Total pins	184
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

BUS의 flow summary이다. 총 3개의 register와 184개의 pin을 사용한 것을 알 수 있으며 성

공적으로 컴파일이 완료된 것을 알 수 있다.

- Ram

Flow Summary	
<<Filter>>	
Flow Status	Successful - Fri Dec 02 23:40:14 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1056
Total pins	72
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Memory, ram의 flow summary이다. 총 1056개의 register와 72개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

## 5. 고찰 및 결론

### A. 고찰

BUS의 코드를 구현한 후 실행하는 과정에서 M\_addr 부분에서 오류가 났다. 기존의 과제에서는 2개의 master와 2개의 slave를 사용하였기 때문에 동일한 코드가 아닌 프로젝트의 조건에 맞게 코드를 수정해야 했다. 기존 코드에서는 mux를 사용하였지만 프로젝트의 BUS의 경우 Master가 1개이므로 mux가 불필요해진다. 이부분에서 Master가 M\_grant 신호를 받은 후에는 M\_addr 신호를 통해 slave의 memory map에 따라 slave와 communication을 수행해야 하는데, 이때 mux를 사용하면 M\_addr과 S\_addr이 동일해져 오류가 났다. 이 오류를 해결하기 위해 M\_addr과 S\_addr은 동일하다고 assign해준 후 활용하니 오류를 해결할 수 있었다. 또한 Multiplier를 구현하면서, +와 \*, {}를 사용할 수 없다 보니, 연산의 중간중간 과정을 모두 wire나 reg로 선언해주어야 했다. 그 과정에서 cl

와 ASR 등의 module을 instance하여 사용해주었는데, 기존의 코드의 경우 co를 사용하였는데, cla 65에서 bit수를 맞춰주기 위해 fa\_v2를 instance하고 output으로 co를 넣어주니 오류가 발생하였다. C16을 wire로 선언한 후 co 대신 c16을 사용하여 오류를 해결할 수 있었으나, 이유를 아직 깨닫지 못해 아쉬웠다. 또한 ALU with Multiplication의 검증을 하는 과정에서 testbench의 값이 일부가 제대로 나오지 않았었다. delay주는 과정에서 실제 값에 0이 써지기 때문에 결과가 0이 나왔던 것이었는데, reset 후 #20만큼의 시간 필요하다는 것을 알게 되었고, testbench의 결과 역시 바르게 나오는 것까지 확인할 수 있었다. 프로젝트 제안서를 작성할 때 생길 것이라고 예상했던 문제들이 생겼었다. Instance 할 것도 많고 기존의 과제들 보다 복잡하고 난이도가 있다 보니 시간도 많이 걸리고 어려웠지만 testbench에서 안 나오는 값들을 기준으로 오류를 해결해 나갔다. 다시 한 번 testbench의 중요성을 알 수 있었고, 지금까지 과제를 수행하며 warning을 무시했던 적이 많은 것 같다. 하지만 완벽하게 작동을 하기 위해선 warning도 무시해서는 안 된다는 것을 마지막에 깨달은 점이 아쉬움으로 남는다.

## B. 결론

초반에 실습으로 진행하였던 alu, register 등등의 부분은 시간이 많이 지나 기억이 잘 나지 않았었다. 이번 프로젝트를 진행하며 초반에 진행하였던 실습은 물론 최근에 했던 multiplier, bus 등등을 복습할 수 있었고, 그때 미처 이해하지 못했던 부분을 이해할 수 있었던 것 같다. 지금까지 했던 과제들을 instance 하여 코드를 구현하였다는 점이 신기하고 뿌듯하기도 하다. 이번 프로젝트를 통해 과제를 수행하면서 몰랐던 verilog의 기능도 알 수 있었다. 한 학기가 끝나고 알게 되어 아쉽지만, 앞으로 verilog를 활용하게 된다면 유용하게 verilog의 기능들을 사용할 수 있을 것 같다. 또한 1학기 때 이론으로만 진행하였던 디지털논리회로의 내용을 verilog를 통한 실습으로 진행하니 이해가 더 잘 됐던 것 같다. 직접 코드를 구현하고, design을 보고만 구현도 해보고, 등등의 과정을 통해 한층 더 성장할 수 있었다.

## 6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료

공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료