Mini processor

2019202009 서여지

Abstract

간단한 동작을하는 mini-processor block MP를 설계한다. MP는 ALU와 MUL을 이용하여 연산을 수행할 수 있다. 또한 BUS를 통해 외부의 testbench에서 정보를 전달받거나, 내부 RF의 정보를 전달할 수 있다. 지금까지의 실습에서 설계한 모듈을 활용하여 MP 모듈을 구성할 수 있었다.

I. Introduction

이번 프로젝트는 10개의 정보와 10개의 명령어를 입력 받아 실행하는 mini processor를 설계하는 것이다. mini processor mp는 모두 10가지의 명령을 수행할 수 있다. mp가 받는 명령어는 명령의 종류를 표현하는 opcode와 함께 결과를 저장할 레지스터인 Rd, 연산에 이용되는 레지스터인 Ra, Rb의 주소로 이루어져있다. mp는 아무런 연산도 하지 않는 NOP을 포함한 10종류의 연산을 수행할 수 있다. 그 중 곱셈 연산은 mul에서 실행하고, 덧셈과 뺄셈, 논리연산은 alu에서 계산한다. top module은 mp와 bus로 이루어져있다. 이때 bus의 master는 testbench가 되고, slave는 mp module이다.

II. Project Specification

RF

레지스터 파일은 여러 개의 레지스터와 write/read logic으로 이루어져있다. 인자로 clk, reset_n, Addr, wData, we를 받고, rData를 출력한다. 레지스터는 clk에 동기되어있으며, reset_n이 low일 때 값을 0으로 초기화한다. we이 1일 때 Addr에 해당하는 레지스터에 wData값을 저장하고, we이 0일 때는 Addr에 해당하는 레지스터에 저장되어 있던 값을 rData로 출력한다. 값을 저장하는 write 동작은 Addr로 입력된 주소를 decoder를 통해 변환한 뒤, 해당 register에만 쓰기 작업을 실행하도록 한다. 반면 read동작은 Addr의 주소값을 이용한 mux를 통해 출력할 레지스터의 값을 결정한다.

ALU – CLA

ALU는 덧셈과 뺄셈, 논리연산을 진행하고, 그 결과 중 opcode에 해당하는 것을 선택하여 출력하는 동작을 한다. alu 내부의 cla를 이용하여 덧셈과 뺄셈을 각각 구하고, 논리연산은 assign문을 이용하여 behavior하게 구현하였다. 각각의 계산 결과는 always문의 내부에서 opcode에 해당하는 것이 선택되어 출력된다.

ALU에서 덧셈과 뺄셈 연산을 하는 cla module은 덧셈 과정에서 발생하는 carry를 우선 계산하여 총 계산에 걸리는 시간을 단축할 수 있는 구조이다. carry의 값을 뒷자리부터 구하여 앞으로 전달하는 rca보다, 계산할 수의 길이가 길 때 속도차이가 크게 나타난다. 뺄셈을 구하는 경우는 계산할 수에 carry in과 보수의 값을 더하는 계산과 같기 때문에 동일한 모듈을 이용할 수 있다.

MUL

multiplier는 booth 알고리즘을 이용한다. booth 알고리즘은 multiplier와 multiplicand를 곱할 때, multiplicand의 연속성에 따라 multiplier를 더하거나 빼는 것을 반복하여 곱셈 결과를 구한다. booth알고리즘은 다음의 순서로 진행된다. multiplicand의 하위비트에서부터 연속된 두 비트를 확인한다. 두 비트의 배열에 따라 결과 result의 상위 bit에 multiplier의 값을 더하거나 뺀다. result를 1bit만큼 ASL 한다. 이 과정을 반복하여 multiplicand를 모두 조회한다. 계산 과정에서 연속된 두 bit의 내용이 같은 경우 더하거나 빼기 연산을 하지 않고, ASL연산만을 진행한다. 위의 알고리즘을 실제로 구현할 때는 multiplicand의 값을 LSL하여 항상 고정된 위치의 두

bit를 읽어 연산을 수행하도록 설계하였다. booth multiplier의 연산에 걸리는 시간은 사용한 radix의 값에 따라 다르다. 이번 프로젝트에서 사용한 radix-2의 경우, 한 번의 clock cycle에 대해 한 자리의 연산을 수행하므로 곱셈을 계산하려는 수의 자리 수만큼의 clock cycle이 필요하다. 반면 한 번에 3bit의 muliplicand를 비교하여 연산하는 radix-4의 경우 한 번의 clock cycle에 2자리의 수를 계산하여 radix-2를 이용했을 때에 비교하여 절반의 clock cycle만을 필요로 한다.

BUS

bus는 master와 slave사이에서 정보를 주고받을 수 있도록 하는 모듈이다. bus에는 여러 개의 master와 slave가 존재할 수 있다. 각각의 master가 grant신호를 통해 bus의 사용을 요청하면, arbiter의 지정된 state에 따라 현재 bus를 사용할 수 있는 master를 결정한다. master는 지정된 주소의 레지스터에 읽기/쓰기 동작을 slave에게 요청할 수 있다. slave는 master에게 주소값을 전달받아 해당 주소의 레지스터에 함께 전달받은 내용을 작성하거나, 해당 레지스터의 내용을 다시 master에 전달한다. 이번 프로젝트에서 사용한 bus는 하나의 master와 하나의 slave를 갖는다. 이 경우 다른 master가 존재하지 않기 때문에 bus를 사용하는 master는 항상 testbench이다. 따라서 별도의 arbiter를 사용하지 않고 구현할 수 있다.

MP

mp는 rf, alu, mul을 포함하고, FSM을 구성하는 next_state, select 등의 combinational circuit을 instance하여 설계하였다. bus를 통해 testbench에서 전달된 명령어와 정보를 rf에 저장한 뒤, 10개의 명령어를 순서대로 실행한 뒤 종료한다. mp의 실행단계는 크게 3가지 과정으로 나눌 수 있다. 첫 번째 단계는 testbench에게서 총 21개의 값을 입력받는 것이다. 10개의 data와 10개의 명령어, 하나의 op_start신호를 입력받는다. op_start신호를 입력받으면 다음 단계로 넘어간다. 다음 단계는 앞에서 입력받은 명령어를 수행하는 단계이다. 이 단계는 다시 여러 단계로 나누어진다. 계산에 필요한 각각의 값을 rf에서 받아오는 과정을 반복한다. 모든 값을 불러오면 연산을 시작한다. 계산을 위해 alu에서 이용하는 시간보다 mul에서 요구하는 시간이 더 길다. 따라서 mul의 계산종료 신호인 op_done을 기준으로 연산 종료를 판단한다. 마지막 단계는 계산의 결과를 다시 저장하는 과정이다. 명령어의 Rd 주소에 해당하는 레지스터에 연산의 결과를 저장한다. 이후 10개의 연산이 모두 끝났다면 interrupt 신호를 출력하고, 아닌 경우 다음 연산을 실행한다.

TOP

top 모듈은 mp와 bus를 연결한 모듈이다.

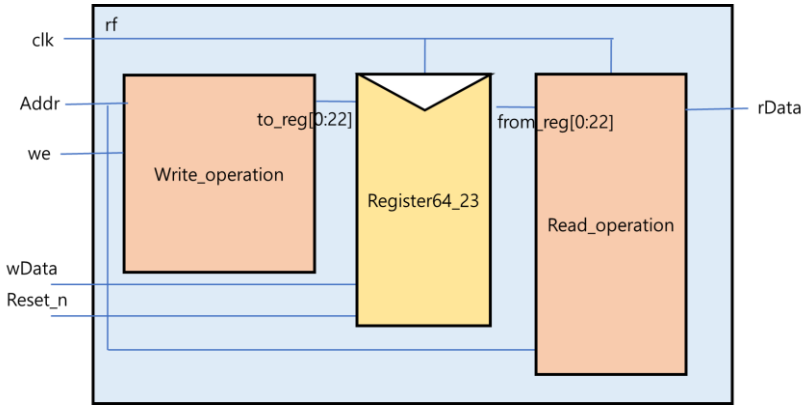
III. Design Details

RF

(1) pin description

direction	port name	bit	description
input	clk	1	clock
	reset_n	1	reset (active low)
	we	1	write enable: 1일 때 쓰기, 0이면 읽기
	Addr	16	address
	wData	64	Addr위치에 작성할 내용
output	rData	64	Addr위치에서 읽은 내용

(2) block diagram



ALU – CLA

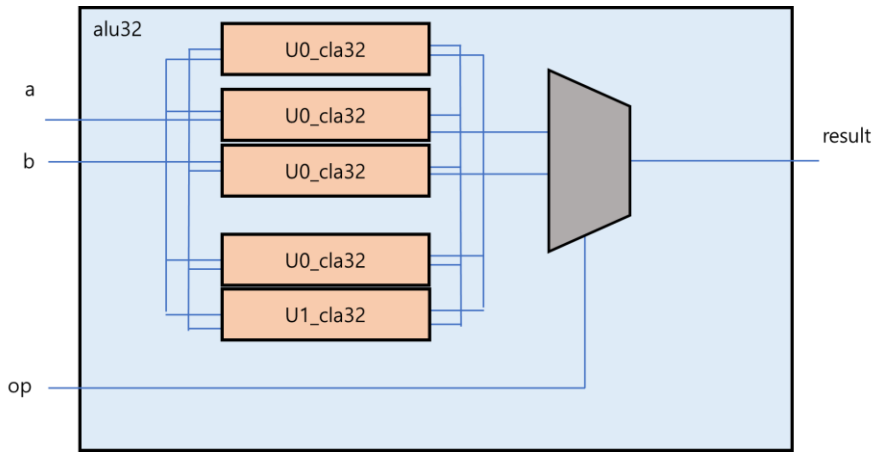
(1) pin description

direction	port name	bit	description
input	a	32	Ra, 계산할 수1
	b	32	Rb, 계산할 수2
	op	4	opcode, 명령어
output	result	64	명령어에 해당하는 계산결과

alu가 수행할 수 있는 명령어의 종류는 다음과 같다.

opcode	encoding	description	
NOT_A	0001	~a	a의 모든 bit를 반전한다.
AND	0010	a&b	a와 b가 모두 1인 자리를 반환한다.
OR	0011	a|b	a또는 b가 1인 자리를 반환한다.
XOR	0100	a^b	a와 b가 일치하지 않는 자리를 반환한다.
XNOR	0101	~(a^b)	a와 b가 일치하는 자리를 반환한다.
ADD	0110	a+b	a와 b의 합을 반환한다.
SUB	0111	a-b	a와 b의 차를 반환한다.

(2) block diagram



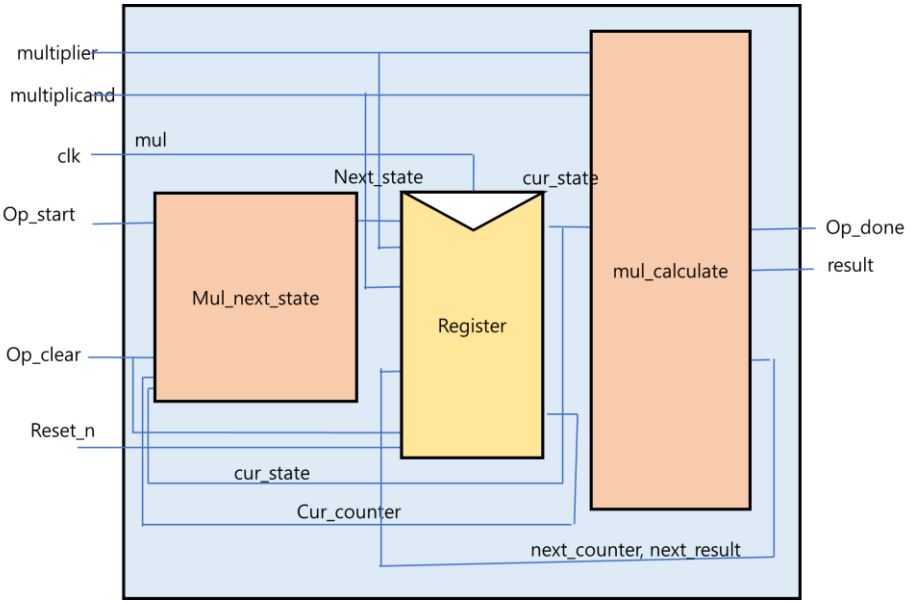
MUL

(1) pin description

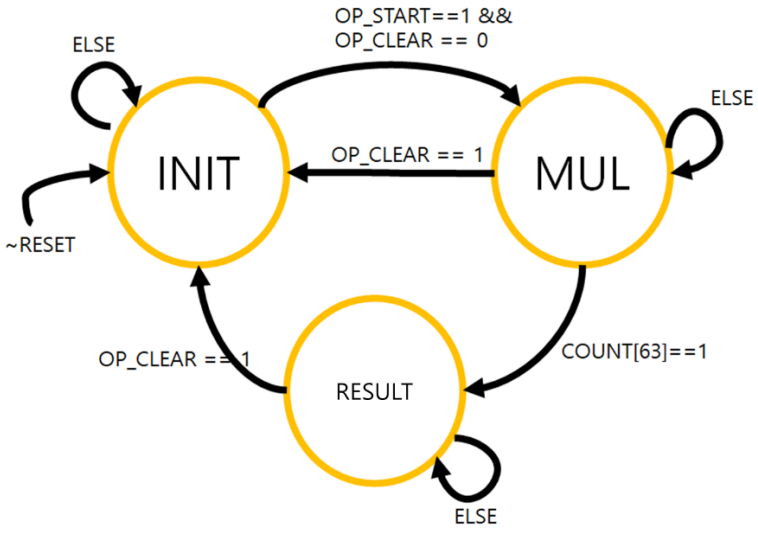
direction	port name	bit	description
input	clk	1	clock
	reset_n	1	active low reset

	op_start	1	연산 시작신호
	op_clear	1	계산 중단, 초기화 신호
	multiplier	32	Ra, 계산할 값1
	multiplicand	32	Rb, 계산할 값2
output	op_done	1	계산 종료 신호
	result	64	계산결과

(2) block diagram



(3) state transition diagram



제안서에서의 state transition diagram에서 state의 이름이 변경되었다. end는 keyword이므로 state의 이름으로 이용할 수 없었다.

BUS

(1) pin description

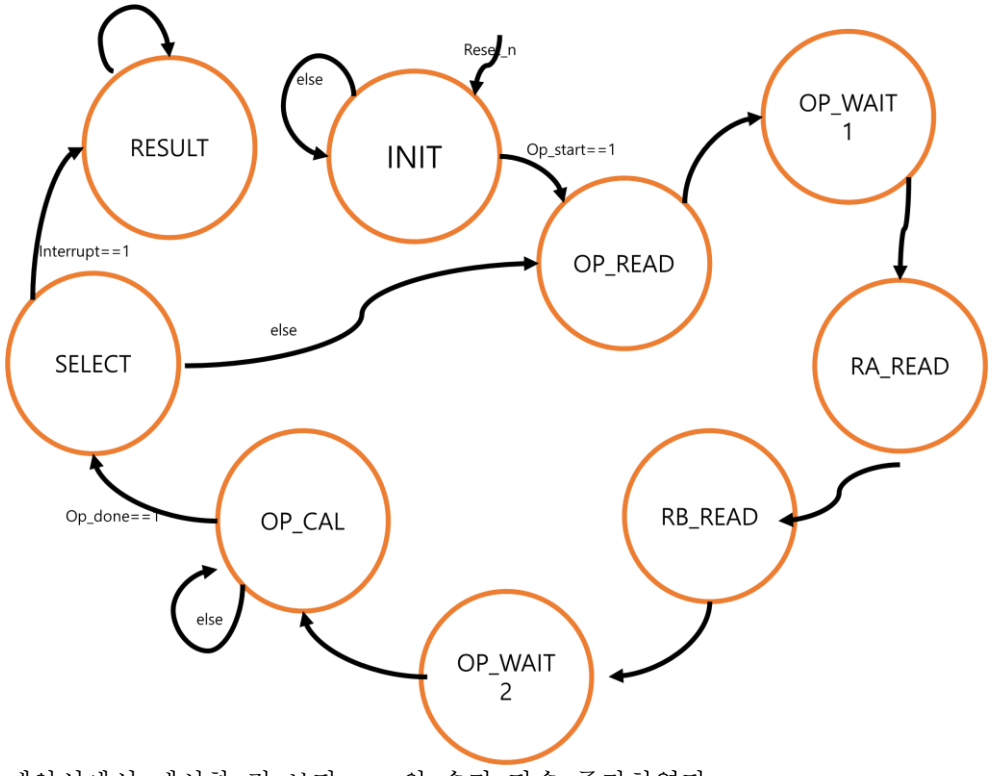
direction	port name	bit	description
input	clk	1	clock
	reset_n	1	reset (active low), 모든 레지스터 0으로 초기화
	m_req	1	master request
	m_wr	1	master write/read: 1인 경우 write/0일 때 read
	m_addr	16	master address
	m_dout	32	master data out
	s_dout	64	slave data out
output	m_grant	1	master grant: 1일 때 사용가능
	m_din	64	master data in
	s_addr	16	slave address
	s_wr	1	slave write/read
	s_din	64	slave data input
	s_sel	1	slave select: 1일 때 사용가능

MP

(1) pin description

direction	port name	bit	description
input	clk	1	clock
	reset_n	1	active low reset
	s_wr	1	slave write/ read
	s_addr	16	slave address
	s_din	32	data input
	s0_sel	1	slave0 select
output	s_dout	64	data output
	interrupt_out	1	interrupt: INT_MASK가 1일 때, 모든 연산 종료 신호

(2) state transition diagram



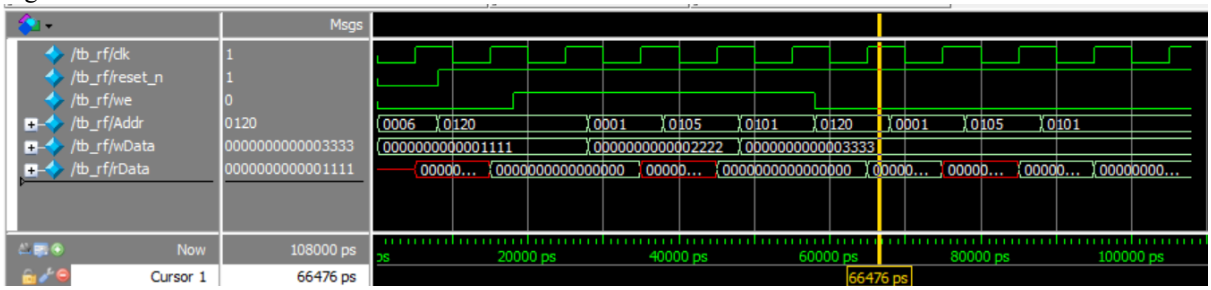
제안서에서 제시한 것 보다 state의 수가 다수 증가하였다.

IV. Design Verifiaction Strategy and Results

각 component 별 검증 전략 및 waveform 작성

(1) RF

register의 각 주소별로 쓰기/읽기가 정상적으로 동작하는지 확인한다.

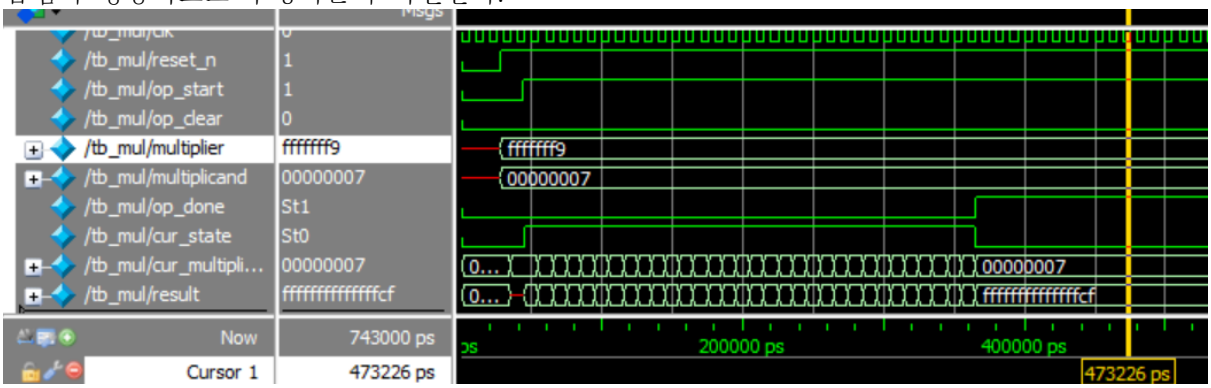


(2) ALU

각 명령어가 정상적으로 계산되는지 확인한다.

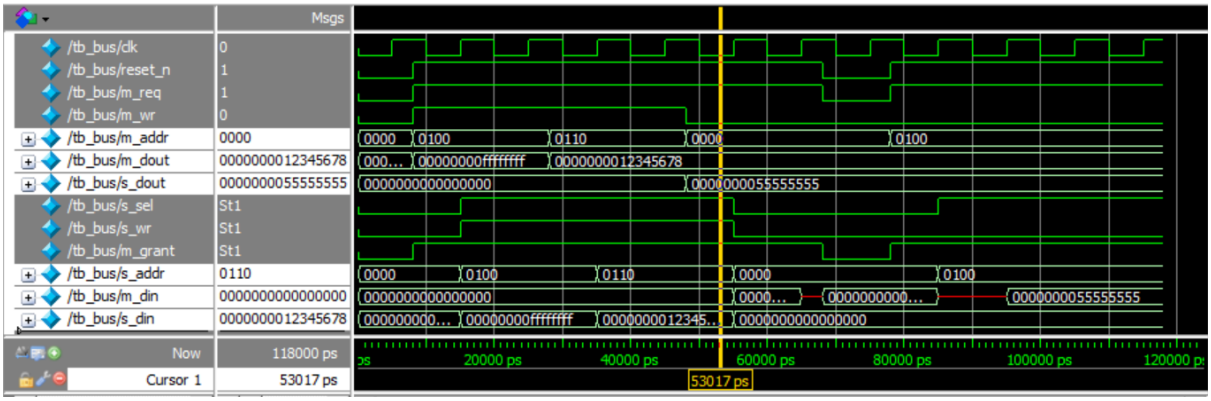
(3) MUL

곱셈이 정상적으로 수행되는지 확인한다.



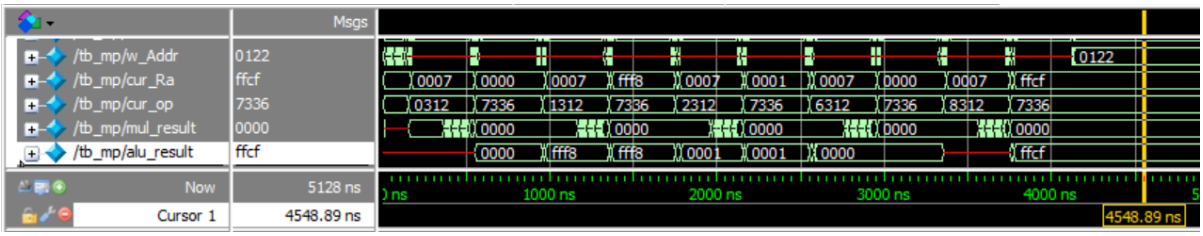
(4) BUS

데이터의 전달이 정상적으로 진행되는지 확인한다.



(5) MP

데이터의 입력과, 입력된 데이터를 바탕으로 하는 연산이 정상적으로 진행되는지 확인한다. 아래 testbench 결과를 통해 10번의 연산이 이루어진 후 종료되고, 각 연산의 결과가 정상적으로 저장되었음을 확인할 수 있다. 아래 실험 결과는 연산 과정을 확인하기 위해 별도의 출력을 추가한 것이다.



V. Conclusion

한 학기동안 실습시간을 통해 직접 구현한 모듈을 활용하여 mini processor를 설계하는 프로젝트였다. 기존에 작성한 모듈을 프로젝트에 맞도록 수정하는 과정에서 오류를 만들어내거나, 모듈의 입/출력 혹은 모듈의 역할 자체가 불분명해지는 경험을 했다. 분명히 정의해 둔 모듈을 그 형식에 따라서 이용하는 것이 바람직하다는 생각을 하게 되었다. 또한 mp module을 설계할 때 state를 정의하는 과정에서 어려움을 겪었다. RF를 이용하여 레지스터의 자료를 읽으면 1clock cycle이 지난 뒤에 data가 들어오기 때문에 이것을 d flip-flop을 이용한 register module에 삽입하기 위하여 wait state를 추가하게 되었다. 중간에 삽입된 state는 encoing을 수정하거나 state의 bit수를 증가시켜서 코드를 자주 수정하게 되었다. 더불어 모듈이 갖게 된 state가 과도하게 늘어나서 비효율적이라는 느낌을 받기도 했다. mp가 계산에 이용할 정보를 입력받고, 10개의 명령어를 수행하여 결과를 저장하는 것 까지는 구현에 성공했지만, 이후 mask 값을 이용하여 모듈 밖으로 interrupt를 출력하는 것과 상위 모듈인 top모듈의 작성에 실패한 점이 아쉽다. bus의 구조는 이해하였지만 bus가 master와 slave사이에서 동작하는 과정에 대한 이해가 부족했던 것 같다.

VI. Reference

[1] 이준환교수님, 디지털논리회로2 강의자료, 광운대학교 컴퓨터정보공학과,2020
[2] 이준환교수님, 컴퓨터공학기초실험2 강의자료, 광운대학교 컴퓨터정보공학과,2020
[3] David Money Harris 외1인, Digital Design and Computer Architecture, Elesvier

always @* https://stackoverrun.com/ko/q/2625050