Arithmetic & logical computing system

2015722031 박태성

Abstract

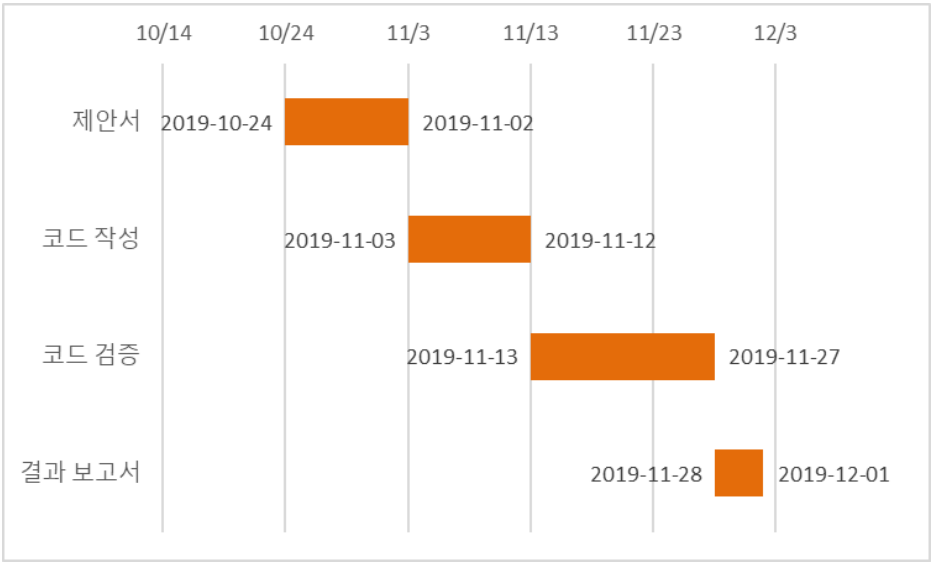
본 시스템은 Arithmetic Logical Unit(ALU)을 이용하여 곱셈, 덧셈, 뺄셈, 및 논리연산을 한다. 시스템은 ALU, direct memory access controller(DMAC), random access memory(RAM), bus로 구성되고, test bench를 이용하여 제어한다.

I. Introduction

본 시스템은 ALU에서 2n 개 혹은 이보다 적은 수의 operand에 대해 n 개의 instruction 을 ALU에서 수행한 후, n 개의 결과 result를 RAM#3에 저장한다. 다음은 시스템의 동작 순서이다.

1. 시스템이 시작되면, Testbench가 bus master 가 되어 OP, INST를 RAM에 저장한다.
2. Testbench 는 DMAC에 REQ_i 를 저장한다.
3. Testbench 는 DMAC에게 opstart 명령을 내려서 source address로부터 destination address로 데이터를 copy 하도록 한다.
4. DMAC이 bust master 가 되어 RAM#1의 operand와 RAM#2의 instruction을 ALU에 copy 한다.
5. DMAC이 전송을 완료하면, testbench 에게 d_interrupt 신호를 발생한다.
6. Testbench 는 bus master 가 되어, DMAC의 d_interrupt 신호를 0으로 변경하고, 모든 전송이 완료 될 때까지, DMAC에게 추가 전송 명령을 한다. (2 ~ 5번)
7. 모든 전송이 완료되면, testbench 는 ALU에게 opstart 명령을 내려서 연산을 시작하게 한다.
8. ALU는 operand와 instruction을 이용하여 연산을 수행하고, 그 결과를 result FIFO에 push 한다.
9. 모든 연산이 끝나면, ALU는 testbench 에게 d_interrupt 신호를 발생한다.
10. Testbench 는 d_interrupt 신호를 0으로 만들고, DMAC에게 result fifo 값을 RAM#3로 이동하도록 명령을 내린다.
11. 모든 전송이 완료되면, testbench 에게 d_interrupt 신호를 발생한다.
12. Testbench 는 DMAC 의 d_interrupt 를 clear 한다.

다음은 본 프로젝트의 수행 계획이다.



II. Project Specification

1. ALU

ALU는 두 개의 입력을 operation code(opcode)에 따른 연산을 이용해 결과값을 도출하는 하드웨어이다. ALU는 register description에 의해 동작한다. ALU에는 operand(OP)를 저장하는 Register File(RF), instruction(INST)를 저장하는 FIFO, 결과를 저장하는 FIFO가 있다. RF는 32bit register, decoder, mux로 구성된다. Data를 read하거나 write하는 device이다. FIFO는 가장 먼저 들어간 data가 가장 늦게 출력되는 device이다. Fault state가 되면 그냥 아무것도 안하는 상태다. 예를 들어 DONE 상태에서 opdone_clear가 와야 IDLE로 넘어가듯이 FAULT 상태에서는 reset 이 와야 IDLE로 넘어간다.

2. DMAC

DMA는 Direct Memory Access의 준말이다. 일반적으로 I/O 디바이스들은 메인 메모리에 직접 접근할 수 없다. 따라서, CPU가 I/O 디바이스들 인터럽트에 대한 처리를 한다. 이는 기존 CPU 작업들을 방해하여 비효율적이다. 이를 해결하기 위하여, Direct Memory Access(DMAC)가 memory와 I/O device 간의 data 전송을 제어하게 한다. Data를 이동/복사하는 동안 processor는 다른 작업을 수행할 수 있으므로 system의 performance가 향상된다. CPU는 DMAC의 인터럽트만 처리하면 된다. DMAC이 작업 완료를 인지하는 2가지 방식이 있다. 첫 번째는 polling method이다. 이는 CPU가 주기적으로 DMAC의 OPREATION_DONE register의 상태를 확인한다. 두 번째는 interrupt-driven method이다. DMAC과 CPU에 연결을 만들어 interrupt signal을 주고 받게 하게 한다. 정리하면, DMAC은 bus로부터 grant를 받아 data의 이동/복사를 제어하는 hardware이다. Testbench가 Master이고, Testbench의 m0_wr = 0 일 때, Testbench가 DMAC의 Slave Register를 읽고, 이 때 DMAC의 Slave Register값이 반환되어 나오는 값이 s_dout 이다. 그리고 s_dout이 BUS의 slave0 port에서 나와 BUS의 m0_din으로 전달되어 값을 받게 된다. 모든 전송이 완료된다면 DMAC의 d_interrupt값이 1이 된다. 이걸로 판단하는게 interrupt-driven method 이다. 혹은 testbench로 DMAC의 register를 읽어서 INTERRUPT register의 값이 1인지 확인하여 판단하는 것을 polling method 라 한다. INTERRUPT_ENABLE Register 값을 통해 판별한다. 즉, INTERRUPT_ENABLE = 0 -> Polling, INTERRUPT_ENABLE = 1 -> interrupt-driven 다.

3. RAM

RAM은 address에 기반하여 data를 저장하는 하드웨어이다. Bus로부터 address와 signal을 입력으로 받아 동작을 수행한다. 이때, bus는 slave이다. Chip enable과 write enable을 통해 address에 data를 read 또는 write 한다. Address는 6bit의 bandwidth를 가지며, data는 32bit의 bandwidth를 갖는다.

4. BUS

BUS는 여러 component 들 간에 data 를 전송 할 수 있도록 연결해주는 component이다. BUS는 arbiter와 address decoder로 구성된다. Arbiter는 master를 결정하는 grant signal을 출력한다. Address decoder는 master로부터 address를 받아서 slave를 선택한다. Address region에 따라서 slave를 구분한다. 본 프로젝트의 BUS는 2개의 master와 5개의 slave를 가지고 있다. Address는 16bit의 bandwidth를 가지며, data는 32bit의 bandwidth를 갖는다.

다음은 BUS의 base address Map 이다.

Memory Map	
Address Range	Component
0x0000 ~ 0x001F	Slave 0
0x0100 ~ 0x011F	Slave 1
0x0200 ~ 0x023F	Slave 2

0x0300 ~ 0x033F	Slave 3
0x0400 ~ 0x043F	Slave 4

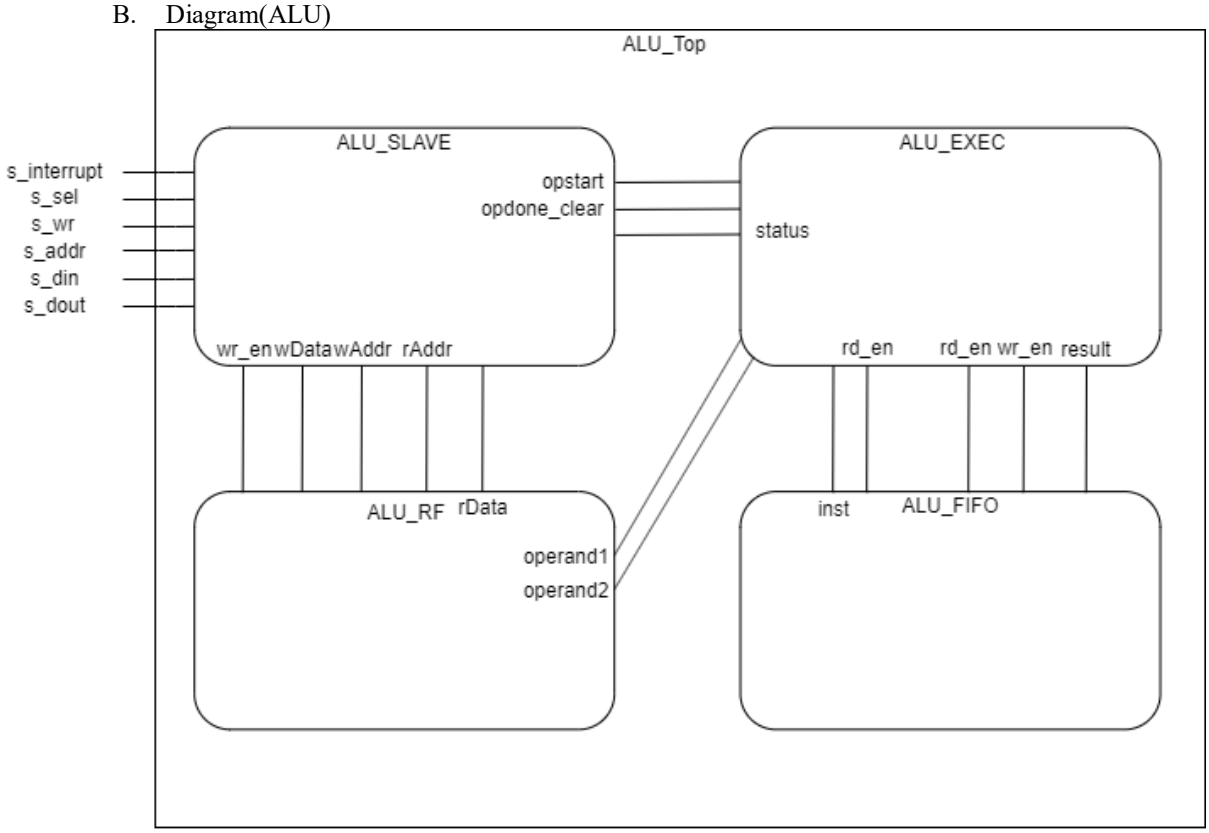
III. Design Details

1. ALU

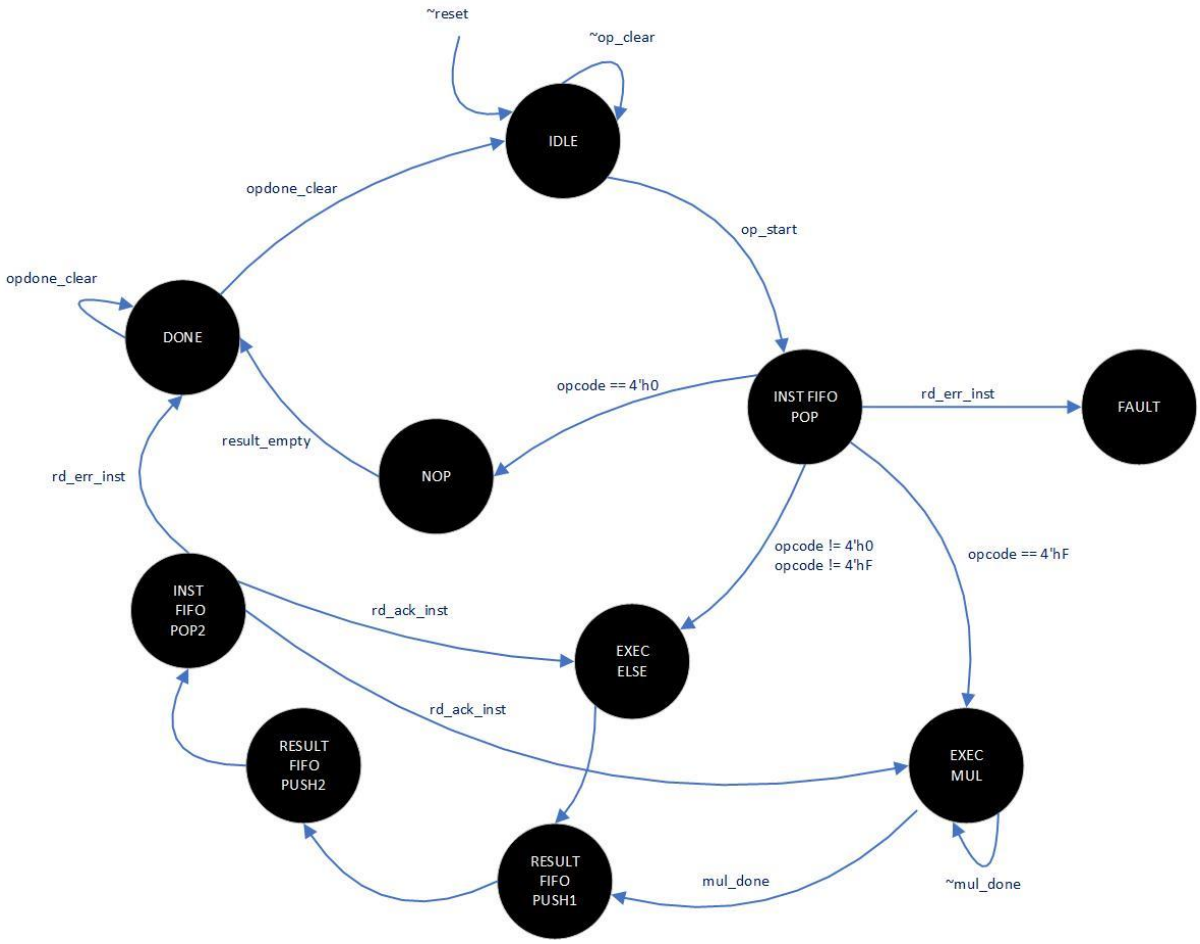
A. Pin description(ALU)

Direction	Port name	Description
Input	clk	Clock
	reset_n	Synchronous active low reset
	s_sel	(Slave) select
	s_wr	(Slave) R/W
	s_addr[15:0]	(Slave) Address
	s_din[31:0]	(Slave) Data input
Output	s_dout[31:0]	(Slave) Data output
	s_interrupt	Interrupt signal to testbench

B. Diagram(ALU)



C. FSM(ALU)

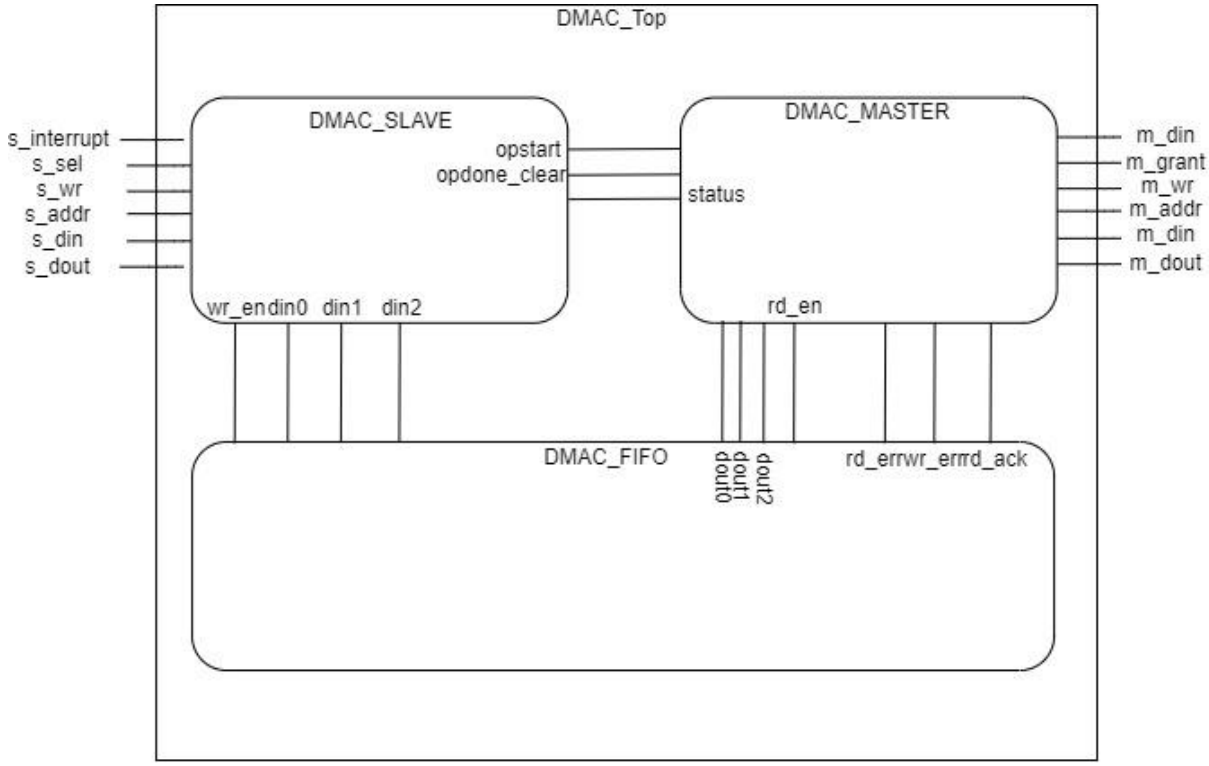


2. DMAC

A. Pin description(DMAC)

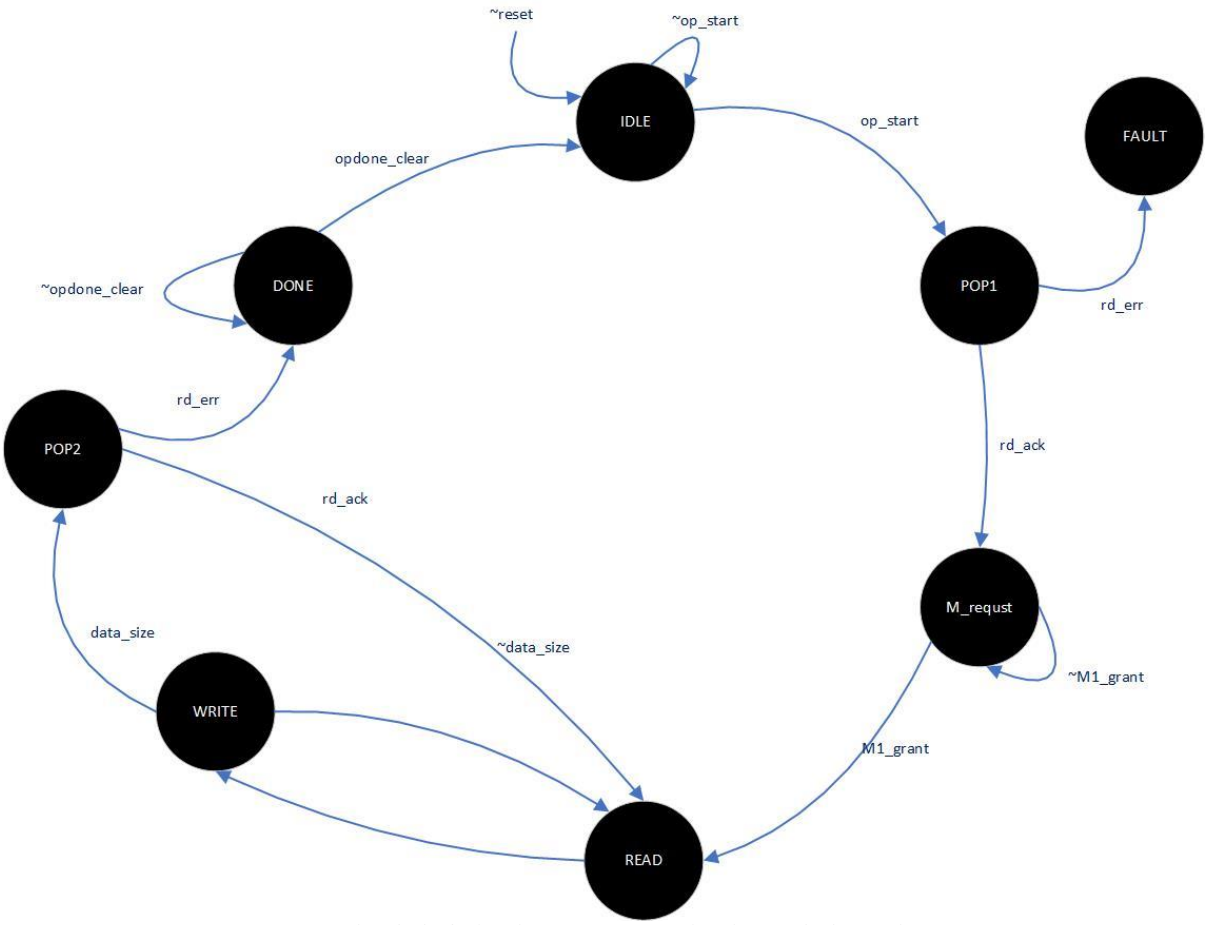
Direction	Port name	Description
Input	clk	Clock
	reset_n	Synchronous active low reset
	m_grant	Master Grant
	m_din[31:0]	Master Din
	s_sel	(Slave) select
	s_wr	(Slave) R/W
	s_addr[15:0]	(Slave) Address
	s_din[31:0]	(Slave) Data input
Output	m_req	Master Request
	m_wr	Master Write Enable
	m_addr	Master Address
	m_dout	Master Data Out
	s_dout[31:0]	(Slave) Data output
	s_interrupt	Interrupt signal to testbench

B. Diagram(DMAC)



DMAC SLAVE: Register I/O 및 interrupt를 제어하는 component.
DMAC MASTER: master interface를 통해 data를 read/ write 하기 위한 component.
DMAC_FIFO: Data의 이동, 복사를 위한 정보(source address, destination address, data size)

C. FSM(DMAC)



Descriptor size: FIFO에 저장되어 있는 descriptor의 개수, 최대 16개.
Data size: DMAC이 data의 이동, 복사 할 때, 전송할 data의 크기, 단위는 4bytes.

3. RAM

A. Pin description(RAM)

Direction	Port name	Description
Input	clk	Clock
	cen	Chip enable
	wen	Write enable
	addr[15:0]	Address
	din[31:0]	Data input
Output	dout[31:0]	Data output

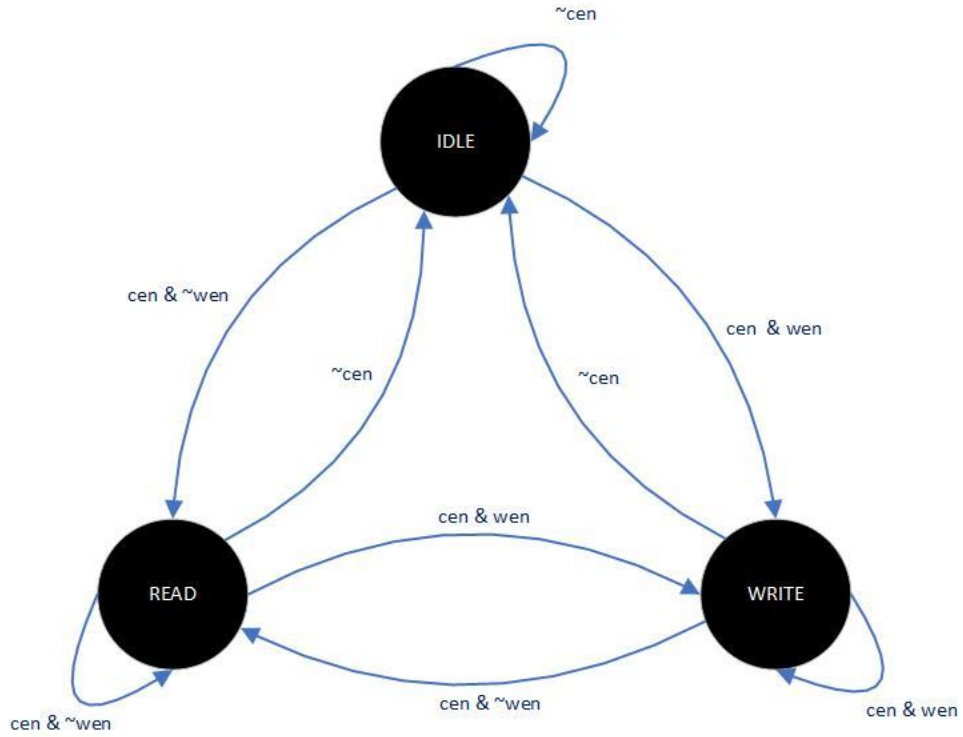
32bit data 64개를 저장.

B. FSM(RAM)

i. State description

State	Description
IDLE	쉬고 있는, 대기 상태, 초기 상태
READ	해당 address로부터 data 읽기
WRITE	해당 address에 data 쓰기

ii. State transition diagram



4. BUS

A. Pin description(BUS)

Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low에 동작
	m0_req	Master 0 request
	m0_wr	Master 0 write/read
	m0_address	Master 0 address
	m0_dout	Master 0 data output
	m1_req	Master 1 request
	m1_wr	Master 1 write/read
	m1_address	Master 1 address
	m1_dout	Master 1 data output
	s0_dout	Slave 0 data output
	s1_dout	Slave 1 data output
	s2_dout	Slave 2 data output
	s3_dout	Slave 3 data output
	s4_dout	Slave 4 data output
Output	m0_grant	Master 0 grant
	m1_grant	Master 1 grant
	m_din	Master data input
	s0_sel	Slave 0 select
	s1_sel	Slave 1 select
	s2_sel	Slave 2 select
	s3_sel	Slave 3 select
	s4_sel	Slave 4 select
	s_address	Slave address
	s_wr	Slave write/read
	s_din	Slave data input

Master의 data out이 BUS에게는 input pin이고 BUS의 output pin이 slave에게는 data in이다.

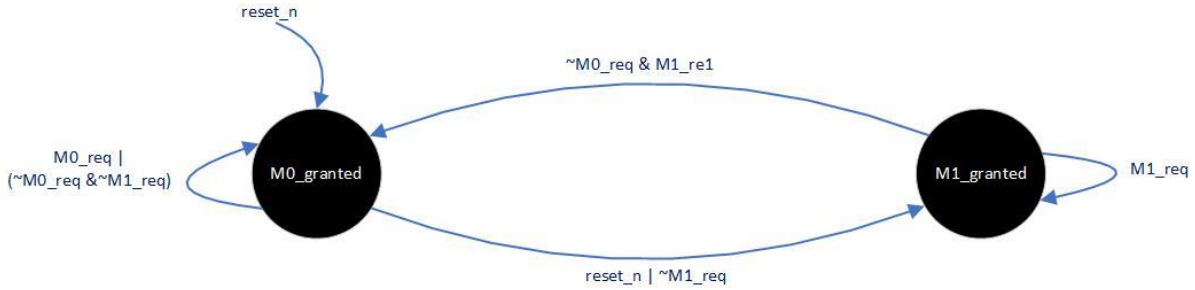
B. FSM(BUS)

i. State description

State	Description
M0_granted	M0에게 권한 부여, 허가
M1_granted	M1에게 권한 부여, 허가

BUS의 arbiter: master 선택
BUS의 decoder: slave 선택

ii. State transition diagram

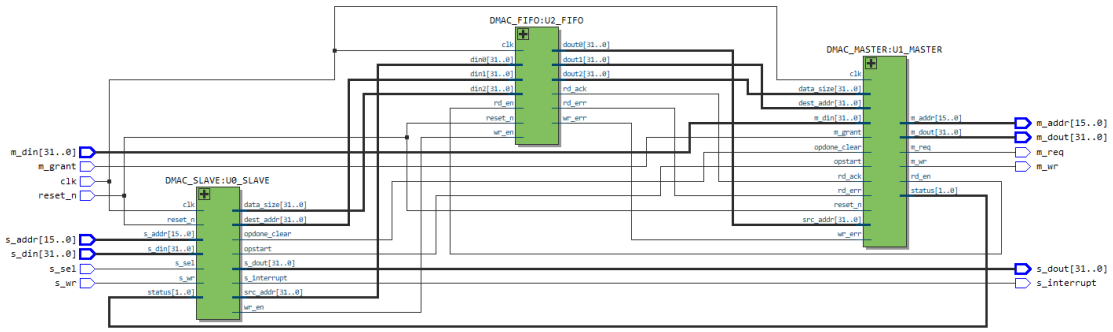


Master가 BUS에게 m_req = 1을 보내면 m_grant = 1 출력.

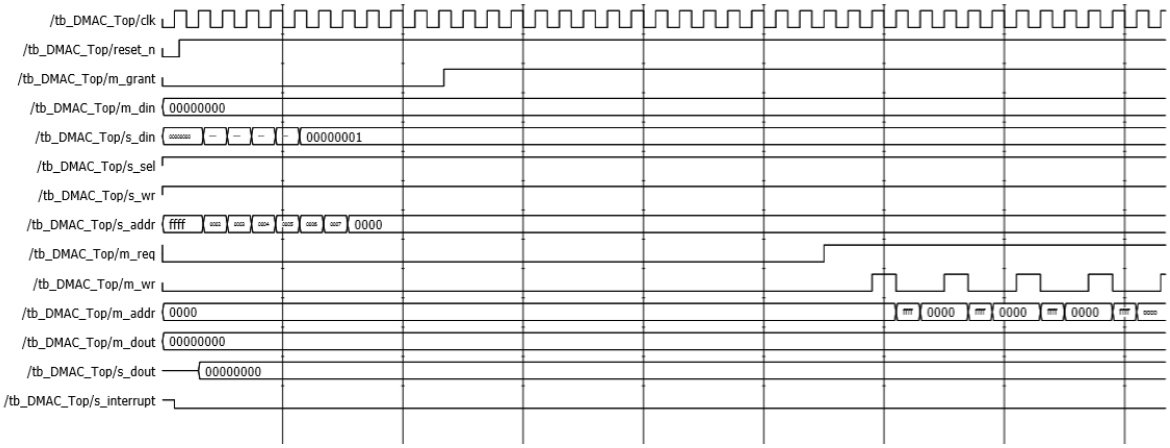
IV. Design Verification Strategy and Results

1. ALU
   A. Design Verification Strategy
      i. ALU는 state가 많고, Register file, FIFO가 많아 data의 저장, 복사가 제대로 되는지 확인한다.
      ii. ALU 연산 결과가 result FIFO에 제대로 저장되는지 확인한다.
   B. Result
      구현하지 못하였다.

2. DMAC
   A. Design Verification Strategy
      i. DMAC SLAVE의 s_dout
         9개의 register 중 select 된 register의 현재 값.
      ii. Source address에서 data를 가져와서 destination address에 복사. (source address: RAM#1, RAM#2, ALU result FIFO, destination address: ALU RF, FIFO, RAM#3)
   B. RTL Viewer

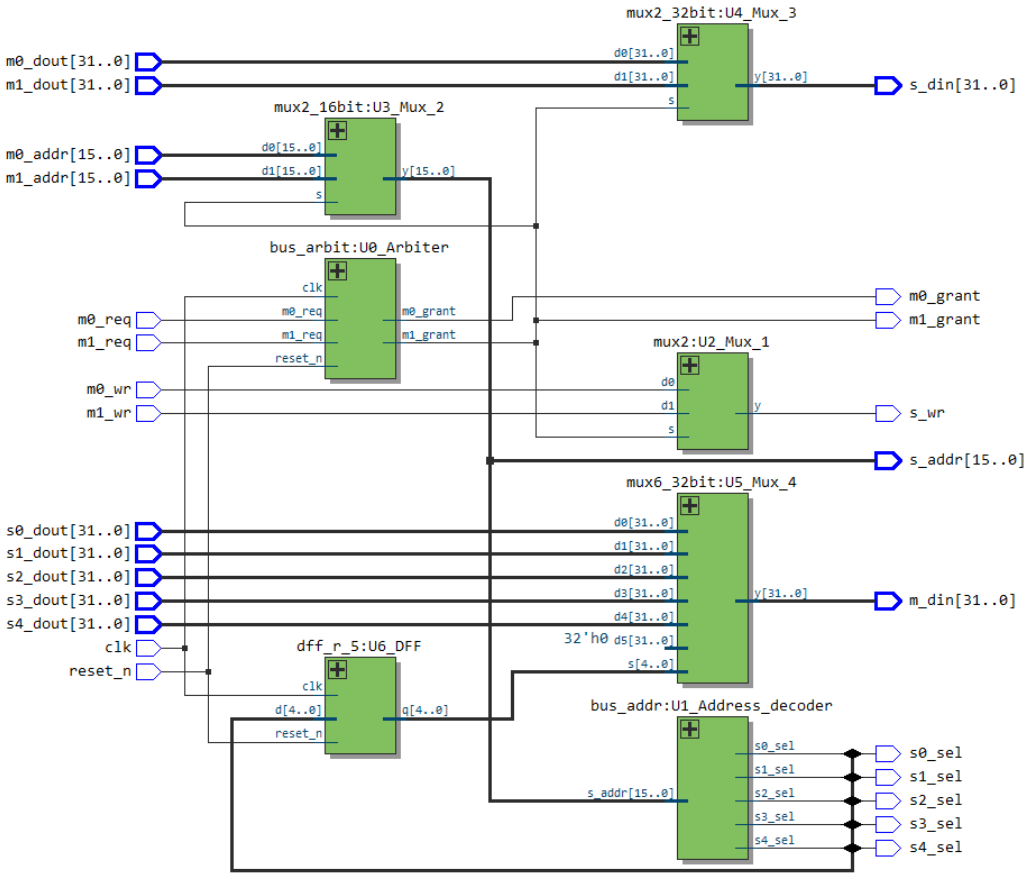


   C. Result

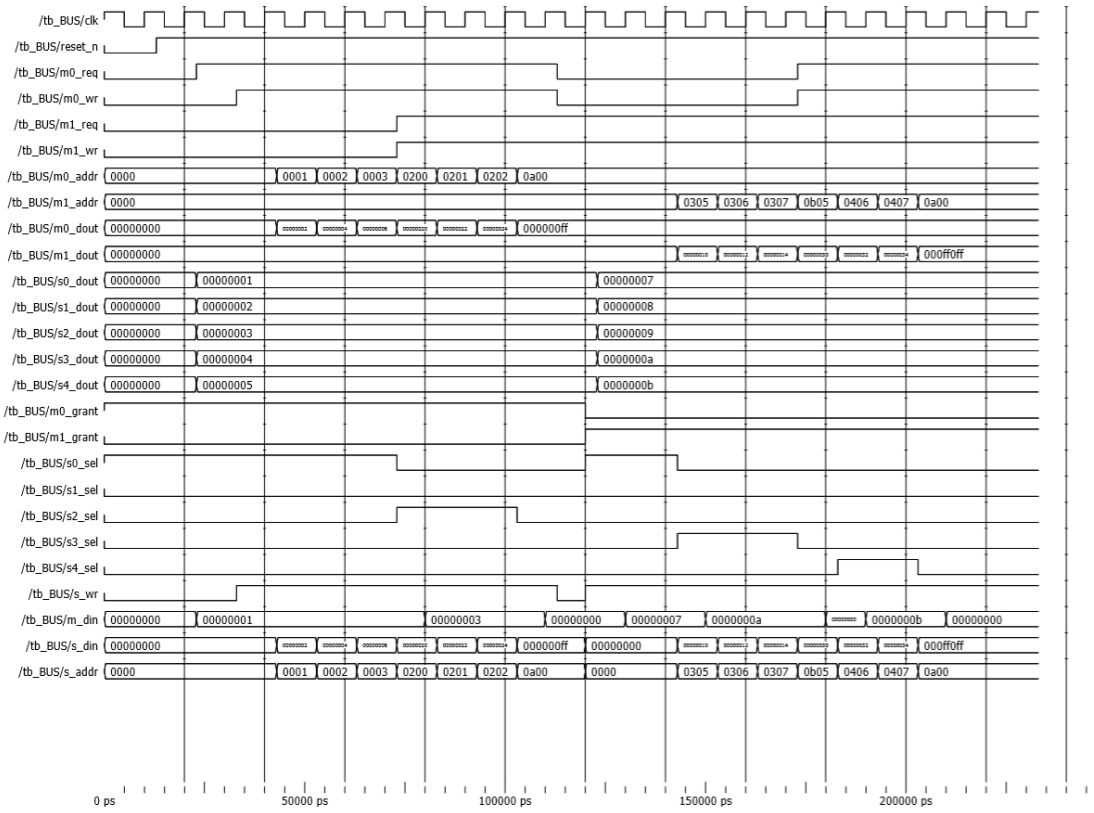


DMAC 검증을 위해서는 descriptor에 REQ를 직접 넣어주어야 한다. 그리고, opstart를 입력으로 주면 FIFO에서 pop하고 bus로부터 master 권한을 받아 data 복사를 시작한다. M_grant signal 출력을 성공하고, FIFO pop을 성공하였으나 cycle이 과도하게 오래 걸려 문제가 있다.

3. RAM
   A. Design Verification Strategy
      i. Chip enable, write enable signal에 따라 정상 작동하는지 확인한다.
   B. Result
      BUS와 연결하여 함께 검증하였다.
4. BUS
   A. Design Verification Strategy
      i. Bus의 arbiter와 address decoder가 정상 작동하는지 확인한다.
      ii. RAM과 BUS 연결 검증
      Testbench가 RAM#1과 RAM#2에 값을 저장하는 동작. 검증 시에는 DMAC, ALU가 없기 때문에 RAM#3도 testbench가 값을 저장하는 것으로 검증한다. M0_wr을 0으로, s_addr를 조절하여 해당 address의 data가 dout으로 출력되는지 검증한다.
   B. RTL Viewer

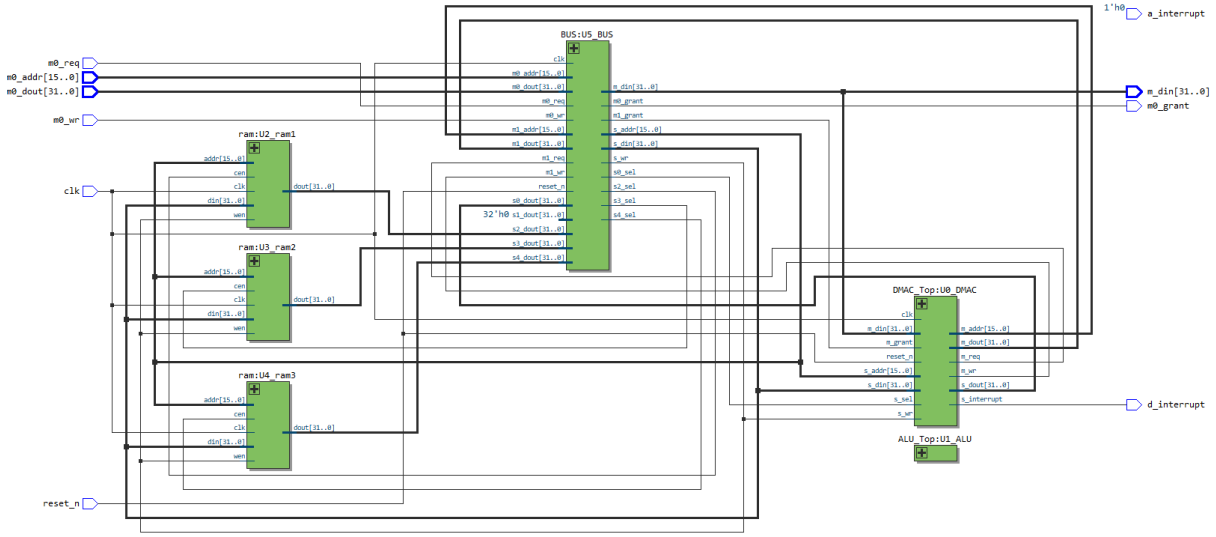

mux2_32bit:U4_Mux_3
m0_dout[31..0]
m1_dout[31..0]
d0[31..0]
d1[31..0]
y[31..0]
s
s_din[31..0]
mux2_16bit:U3_Mux_2
m0_addr[15..0]
m1_addr[15..0]
d0[15..0]
d1[15..0]
y[15..0]
s
bus_arbit:U0_Arbiter
clk
m0_req
m1_req
reset_n
m0_grant
m1_grant
m0_grant
m1_grant
mux2:U2_Mux_1
m0_wr
m1_wr
d0
d1
y
s
s_wr
s_addr[15..0]
mux6_32bit:U5_Mux_4
s0_dout[31..0]
s1_dout[31..0]
s2_dout[31..0]
s3_dout[31..0]
s4_dout[31..0]
d0[31..0]
d1[31..0]
d2[31..0]
d3[31..0]
d4[31..0]
32'h0
d5[31..0]
s[4..0]
y[31..0]
m_din[31..0]
clk
reset_n
dff_r_5:U6_DFF
clk
d[4..0]
reset_n
q[4..0]
bus_addr:U1_Address_decoder
s_addr[15..0]
s0_sel
s1_sel
s2_sel
s3_sel
s4_sel
s0_sel
s1_sel
s2_sel
s3_sel
s4_sel

C. Result

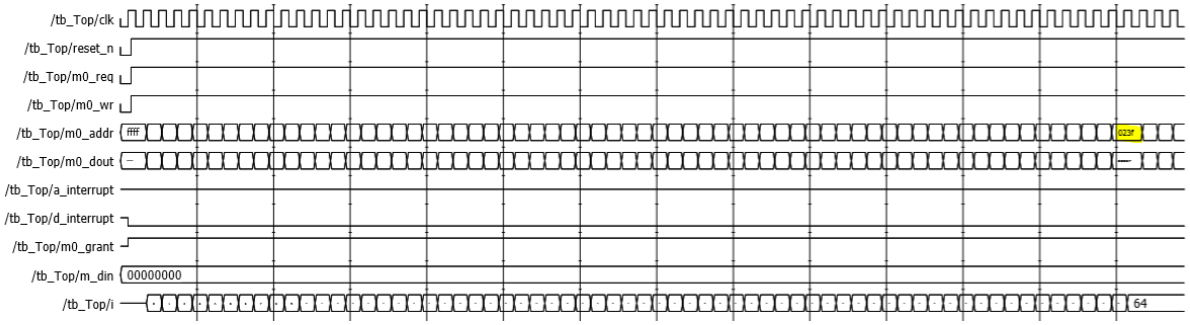

/tb_BUS/clk
/tb_BUS/reset_n
/tb_BUS/m0_req
/tb_BUS/m0_wr
/tb_BUS/m1_req
/tb_BUS/m1_wr
/tb_BUS/m0_addr
/tb_BUS/m1_addr
/tb_BUS/m0_dout
/tb_BUS/m1_dout
/tb_BUS/s0_dout
/tb_BUS/s1_dout
/tb_BUS/s2_dout
/tb_BUS/s3_dout
/tb_BUS/s4_dout
/tb_BUS/m0_grant
/tb_BUS/m1_grant
/tb_BUS/s0_sel
/tb_BUS/s1_sel
/tb_BUS/s2_sel
/tb_BUS/s3_sel
/tb_BUS/s4_sel
/tb_BUS/s_wr
/tb_BUS/m_din
/tb_BUS/s_din
/tb_BUS/s_addr
0 ps
50000 ps
100000 ps
150000 ps
200000 ps

Slave 가 5개로 기존 과제에 비하여 3개 추가된 모습이다. Master가 2개, Slave가 5개인 BUS이다. FSM에 따라, m_grant signal이 정상 출력됨을 확인하였다.

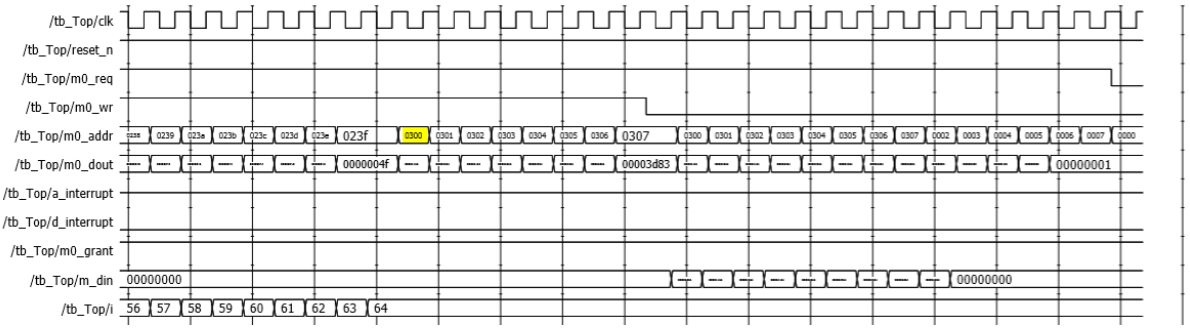
5. Top
   A. Design Verification Strategy
      i. Polling method는 interrupt register read, interrupt driven method는 interrupt port 값 확인.
   B. RTL Viewer



C. Result
   i. RAM + BUS

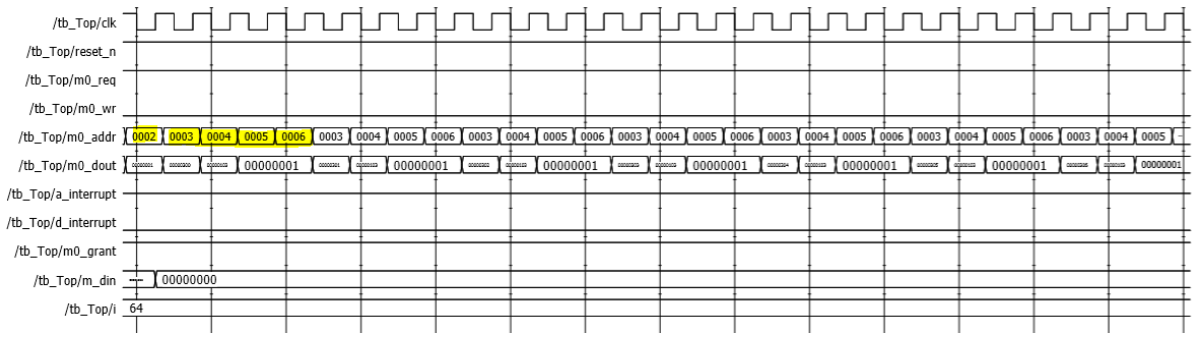


Ram과 BUS의 연결을 확인하였다. Testbench가 bus master가 되어 RAM#1에 data write 하는 wave form이다.



Testbench가 bus master가 되어 RAM#2에 data write, read 하는 wave form이다. Ram의 data, write, read는 m0_wr을 통해 제어한다.

   ii. RAM + BUS+DMAC



Descriptor에 REQ_i 를 초기화하는 wave form이다. Interrupt driven method로 설정한다. 그리고, 차례로 source address, destination address, data size를 초기화한다.

V. Conclusion

Direct Memory Access(DMA) 동작에 특화된 하드웨어인 DMAC 와 여러 component 들 간에 data 를 전송 할 수 있도록 연결해주는 component인 bus, 두 개의 입력을 이용해 operation code(opcode)에 따른 연산을 이용해 결과값을 도출하는 하드웨어인 ALU, 그리고 address에 기반하여 data를 저장하는 하드웨어인 Random Access Memory(RAM) 을 구현하면서, memory-mapped I/O의 원리를 이해하고, 더 나아가 컴퓨터 시스템 구조에 대해 학습하는 데 좋은 기회가 되었다. 검증을 통해 system의 문제의 원인을 찾고 문제를 해결하는 능력을 배양할 수 있었다.

VI. Reference

[1] 이준환, 디지털 논리회로 2, 광운대학교 컴퓨터정보공학부, 컴퓨터공학기초실험2 – Memory & Bus 강의 및 실습자료, 2019-2학기