디지털 논리회로2 프로젝트 제안서

ALU with DMAC

학　　과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 화요일 0, 1, 2

학　　번: 2015722031

성　　명: 박 태 성

1. Title & Object

A. Title

ALU with DMAC

B. Object

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 목적을 둔다. 검증을 통해 system의 문제의 원인을 찾고 문제를 해결하는 능력을 배양하는 데 목적을 둔다.

2. Component concept

A. 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.

B. 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.

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

C. 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.

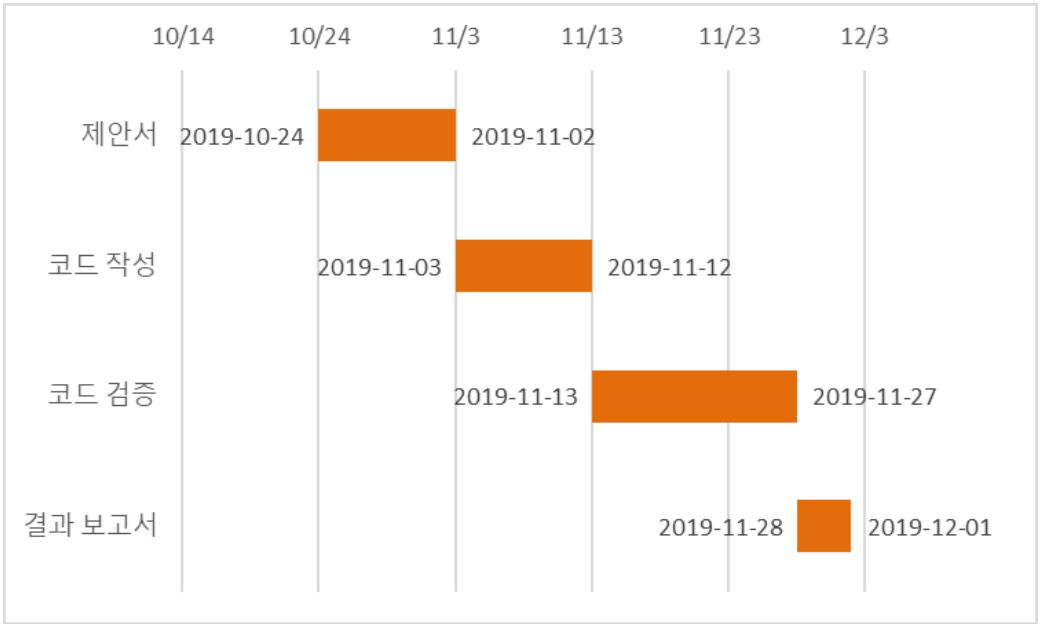
D. ALU

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.

E. TOP

ALU, RAM, DMAC, 그리고 BUS를 각각 instantiation하여 연결한 뒤 제어한다.

3. Schedule



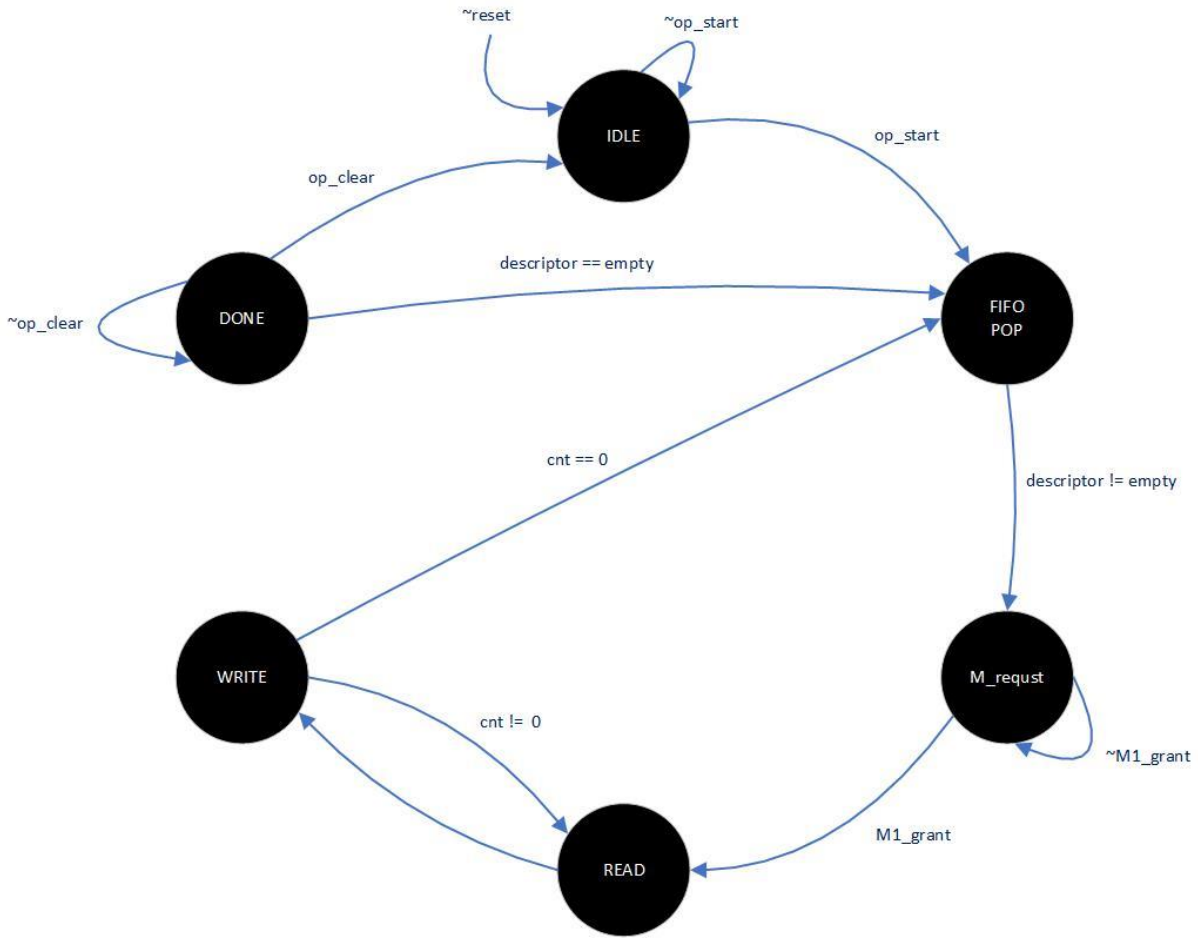
4. State transition diagram

A. DMAC

i. State description

State	Description
IDLE	쉬고 있는, 대기 상태, 초기 상태
FIFO POP	Descriptor에서 source address, destination address, 그리고 data size를 pop
M_request	BUS에게 M1_grant를 받기 위한 대기 상태
READ	현재 source address(RAM)의 data를 읽는 상태, cnt -= 1
WRITE	Read한 data를 destination address에 write
DONE	동작 완료

ii. State transition diagram

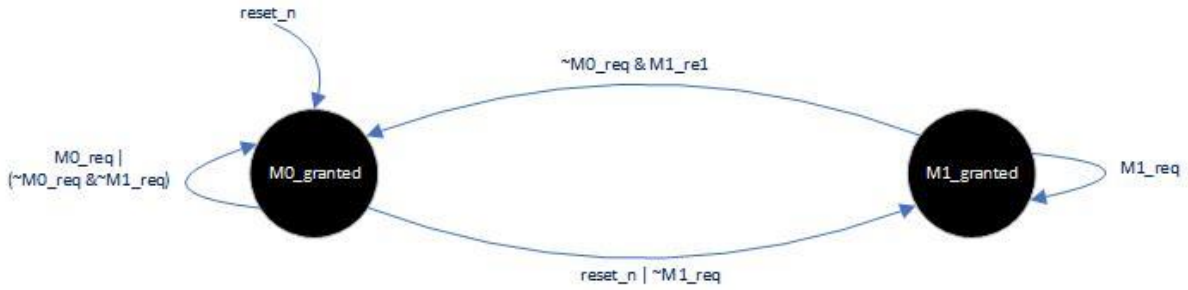


B. BUS

i. State description

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

ii. State transition diagram

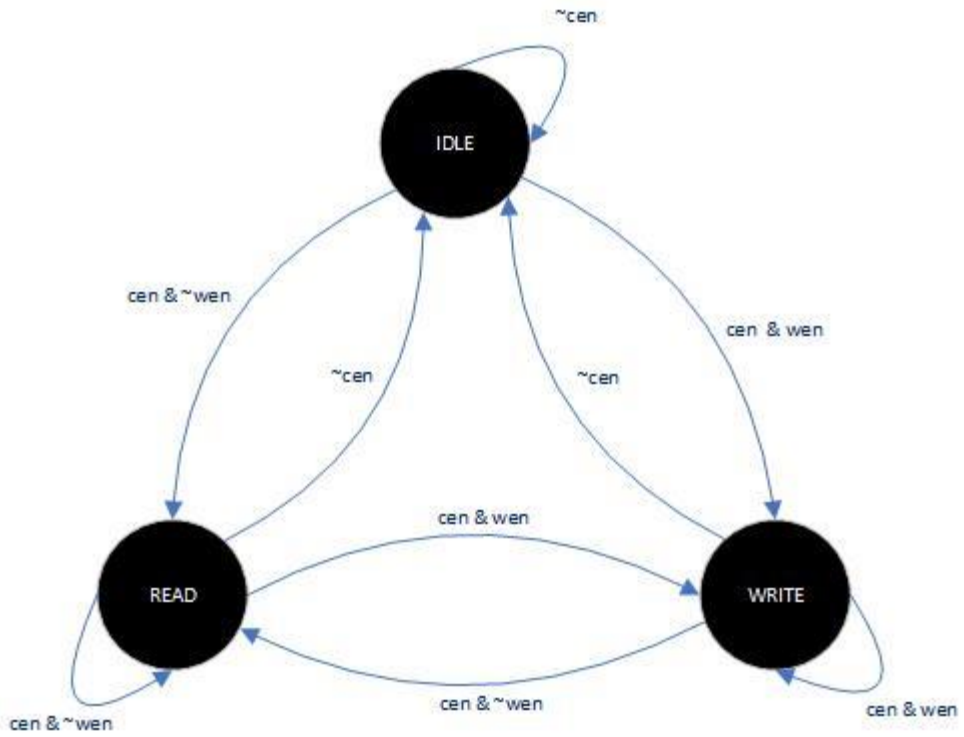


C. RAM

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



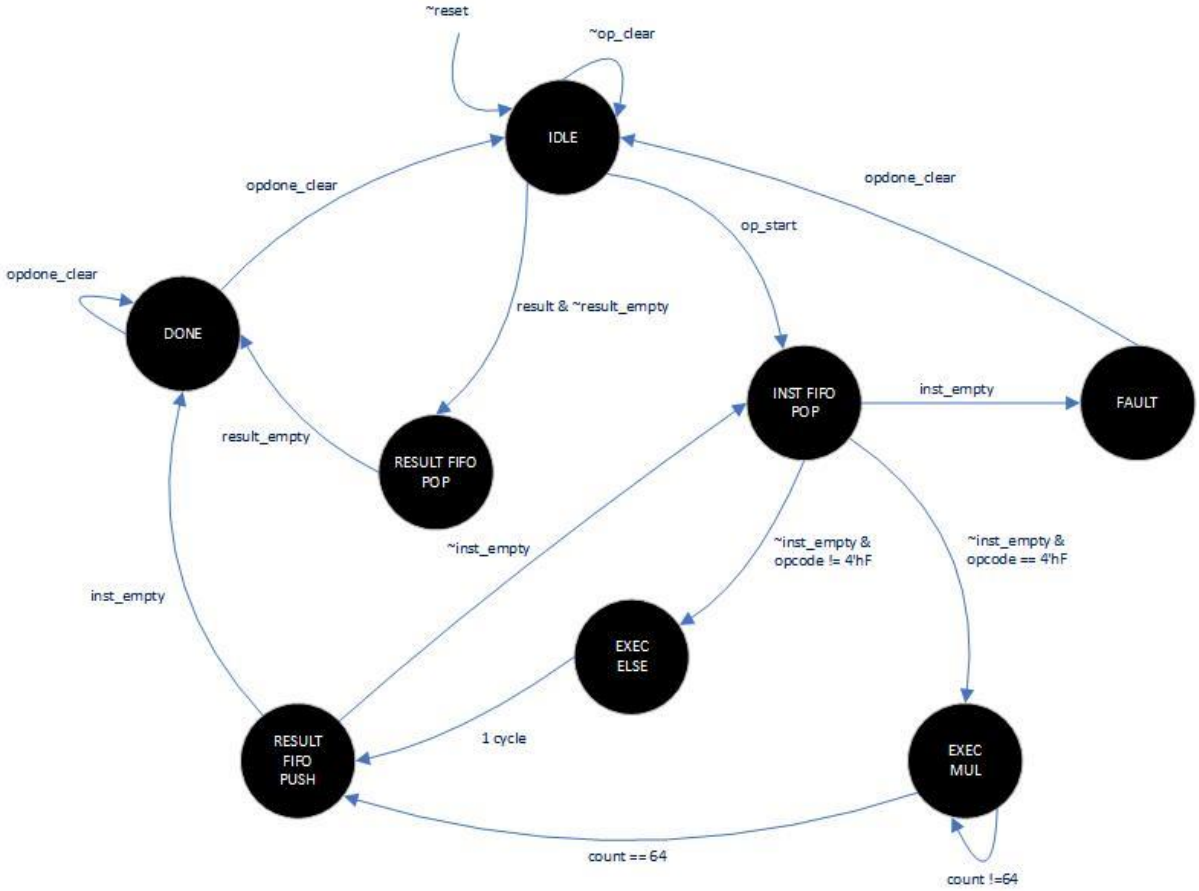
D. ALU

i. State description

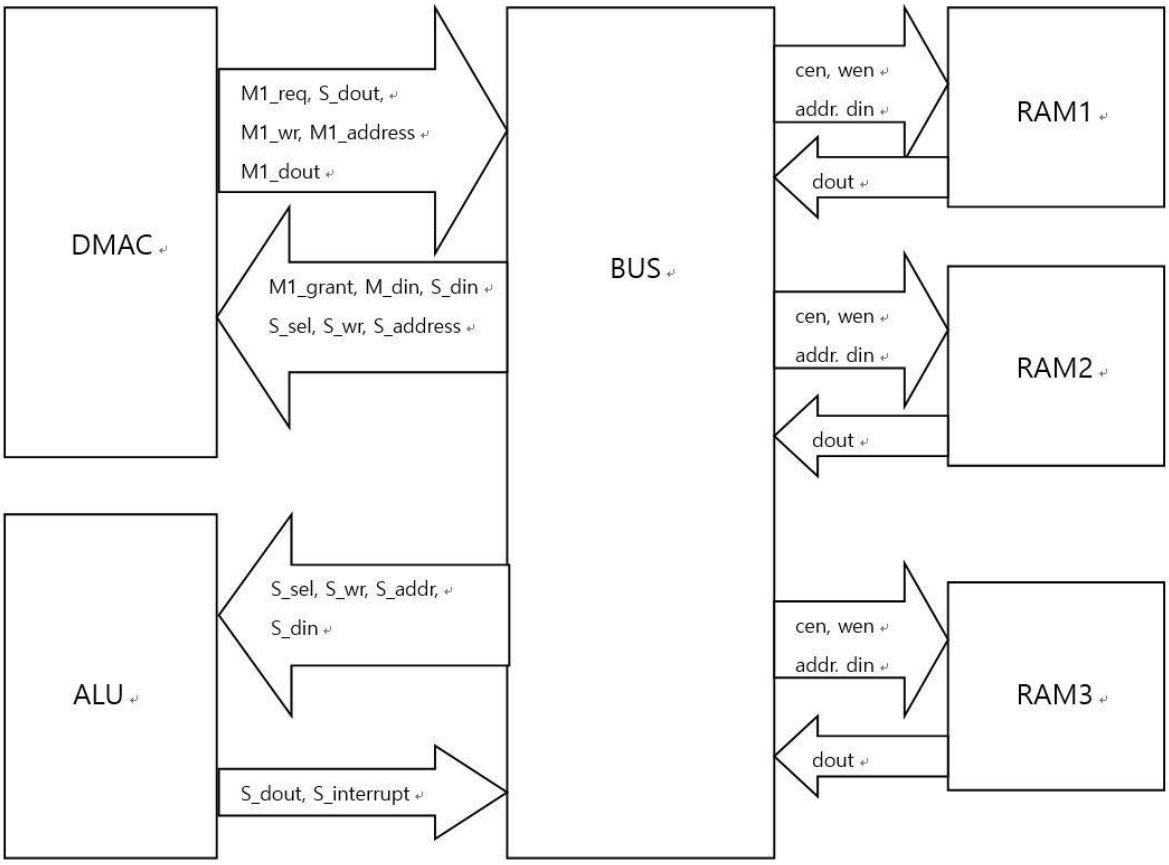
State	Description
DONE	동작 완료
IDLE	쉬고 있는, 대기 상태, 초기 상태
INST FIFO POP	Instruction FIFO에서 하나씩 pop

FAULT	Instruction FIFO가 empty
EXEC MUL	곱셈 연산
EXEC ELSE	NOP, NOT A, NOT B, AND, OR, XOR, XNOR, 등등
RESULT FIFO PUSH	연산 결과를 result FIFO에 저장
RESULT FIFO POP	RAM#3의 0 번지부터 pop 된 순서대로 이동

ii. State transition diagram



5. Module instance design



Top 모듈을 구성하는 하위 모듈들을 도식화 했다.

6. Design verification strategy

A. Expected Problem

- Device간의 data 이동, 복사가 빈번하게 발생한다. 잘못된 위치에 잘못된 data가 저장되면, system 전체가 잘못된 계산을 수행할 가능성이 높다.
- 모듈 instantiation이 많다. Synchronous하게 구현해도 수행 시간이 각 module마다 다르기 때문에, 예상한 시점에 결과가 출력되지 않을 수 있다. 결과가 미리 나오거나 늦게 나올 수 있다.
- Module의 수가 많고 복잡하여 문제가 발생했을 때 원인 규명이 어렵다.
- 많은 조건 문과 인자들이 사용된다. Strict하게 조건 문을 작성하지 않으면 예상과 다른 state으로 가서 잘못된 operation을 수행할 수 있다.
- 수행 순서에 대한 이해도가 부족한 상태에서 설계하면, requirement와 다른 기능을 수행하는 system을 설계할 수 있다.

B. 검증

i. 각 모듈 별 검증

Simple Test bench 작성. 예상한 output과 결과를 simulator program을 활용하여 비교한

다. 그리고 정상 동작하는지 확인한다. 발생 가능한 예외 상황을 만들어 state transition이 잘 이루어지는지 확인한다.

ii. 총합 검증

Self-checking test bench with test vector를 작성.

iii. 모듈 별 검증

- ALU는 state가 많고, Register file, FIFO가 많아 data의 저장, 복사가 제대로 되는지 확인한다.
- ALU 연산 결과가 result FIFO에 제대로 저장되는지 확인한다.
- 연산 결과가 메모리에 정상적으로 저장되는지 확인한다.
- Bus의 arbiter와 address decoder가 정상 작동하는지 확인한다.
- 메모리가 chip enable, write enable signal에 따라 정상 작동하는지 확인한다.