디지털 논리회로2 프로젝트 제안서

Arithmetic & logical computing system

학　　과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 수요일 0, 1, 2

학　　번: 2015722068

성　　명: 남윤창

1. Title & Object

A. Title

Arithmetic Logic Unit

B. Object

ALU(Arithmetic Logic Unit), DMAC(Direct Memory Access Controller), RAM(Random Access Memory), bus을 각각 이해하고 이를 포함한 시스템을 구현한다. ALU를 이용하여 곱셈, 덧셈, 뺄셈 및 논리연산을 수행한다. 이를 통하여 CPU와 DMAC의 관계 및 시스템의 전반적인 구동원리의 이해와 응용력을 기를 수 있다.

2. Component concept

A. DMAC

DMAC는 Direct Memory Access Controller의 준말이다. CPU가 DMAC에게 각 Peripheral Device의 메모리를 직접 관리하라는 명령을 내리면 DMAC가 각 Device간의 데이터 전송을 관리하게 된다. 즉, DMAC가 각 Device들 간의 데이터 전송역할을 전담하게 되어 CPU는 그 시간동안 다른 일을 하게 된다. 이때 DMAC가 일을 완수함을 CPU가 알 수 있는 방법은 2가지가 있다.

첫째, Polling Method. 이는 주기적으로 CPU가 DMAC이 일을 완수했는지 DMAC의 OPERATION_DONE Register를 읽어 완료되었는지 확인하는 방식이다. 장점으로는 wire가 추가적으로 필요하지 않아 공간상의 이득을 본다.

둘째, Interrupt-driven Method. 이는 DMAC과 CPU를 직접 연결하는 Bus이외의 wire를 연결해 완료될 시 interrupt signal을 발생시켜 CPU에서 DMAC가 정보를 전달하는 방식이다. 이 방식의 장점은 CPU가 DMAC의 Register를 주기적으로 읽고 있지 않아도 된다.

두 방법은 모두 각각의 장단점이 존재하기 때문에 설계자는 보통 두가지 방식을 다 지원한다. 따라서 이번 프로젝트에서도 두가지 방식을 모두 지원하게 구현한다.

B. ALU

ALU는 Arithmetic Logic Unit의 준말로 산술연산과 논리연산을 계산하는 CPU의 기본 설계 모듈이다. 전달받은 Instruction을 통하여 어떠한 연산을 할 지 정하고, Operand 1개 혹은 2개로 알맞은 연산을 진행하여 결과를 도출하는 방식이다.

본 프로젝트에서는 Input Operand는 곱셈을 제외하고 모두 64bit이다. 곱셈만 32bit input값을 갖는다. 결과는 모두 64bit bandwidth를 갖는다. Instruction 종류는 총 16가지이다. 논리연산과 산술연산 곱셈까지 포함한다. Operand들은 Register File에 저장되어 있고 Instruction은 FIFO에 저장 되어있다. Instruction에는 Operand A, B값이 저

장 되어있는 주소가 있고 이를 통해 Register File에서 불러와 연산을 수행하게 된다.

C. BUS

Bus는 각 Device간의 통로를 말한다. Bus의 기본적인 특징은 모든 Component가 연결 되어있지만 정작 사용할 경우에는 1개의 Master와 Slave가 선택되어 1대 1 Mapping이 되는 형식으로 사용된다. Bus 에는 크게 Master와 Slave를 결정해주는 Logic이 각각 있다. Master Component는 Bus 내부의 Arbitrary에서 결정하게 된다. Arbitrary의 자세한 내용은 State Transition Diagram으로 뒤에 나온다. 권한을 얻은 Master는 권한을 얻었다는 정보를 Grant신호로 돌려받게 되는데 이를 통하여 각 Master가 Slave에게 정보를 요청하여 받아올 수 있는 모듈이다. Slave는 입력된 Address중 Base Address부분으로 Slave Component를 결정한다. 이를 Address Decoding Logic으로 구현할 수 있다.

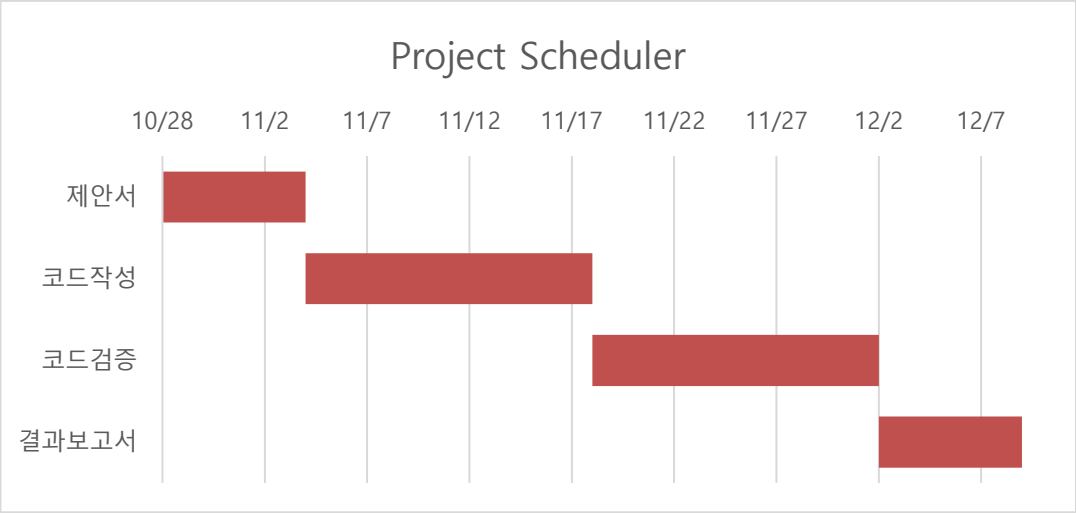
프로젝트에서는 DMAC, ALU, RAM #1,2,3 총 5개의 Device가 있고 CPU를 대체하는 Testbench가 있다. 따라서 6개의 Component가 있는데 Testbench를 Master0, DMAC을 Master1로 Mapping한다. 기본적으로 항상 Testbench가 조건이 없을 경우 Master 권한을 갖게 되고 Testbench가 Request하지 않고, DMAC이 Master Request를 하는 경우에만 DMAC이 Master권한을 얻는다. 더욱 자세한 내용은 State Transition Diagram에 명시되어 있다. 나머지 5개의 Slave Component는 DMAC, ALU, RAM#2, RAM#1, RAM#2순으로 Mapping되어있다. 이는 각각 16bit의 Address중 앞 8bit Base address로, 00, 01, 02, 03, 04 순으로 할당한다.

D. RAM

Random Access Memory의 준말로 임의의 주소가 주어질 때 똑 같은 시간으로 접근이 가능하다. 즉, Random이 무작위의 의미가 아니라 어느 주소를 접근해도 같은 시간이 걸린다는 뜻이다. RAM은 전원이 차단되면 정보가 날아가는 휘발성의 성질을 갖는다.

이번 프로젝트에서 Operand와 Instruction 그리고 Result값을 저장하는 저장공간으로 총 3개의 RAM이 사용된다. 이는 64개의 32bit Data를 Address에 따라 각각 저장되어 있는 특징을 갖는다. RAM도 Bus와 연결 되어있는 하나의 Peripheral Device이고 Slave이므로 Slave Interface를 통하여 Bus와 연결된다.

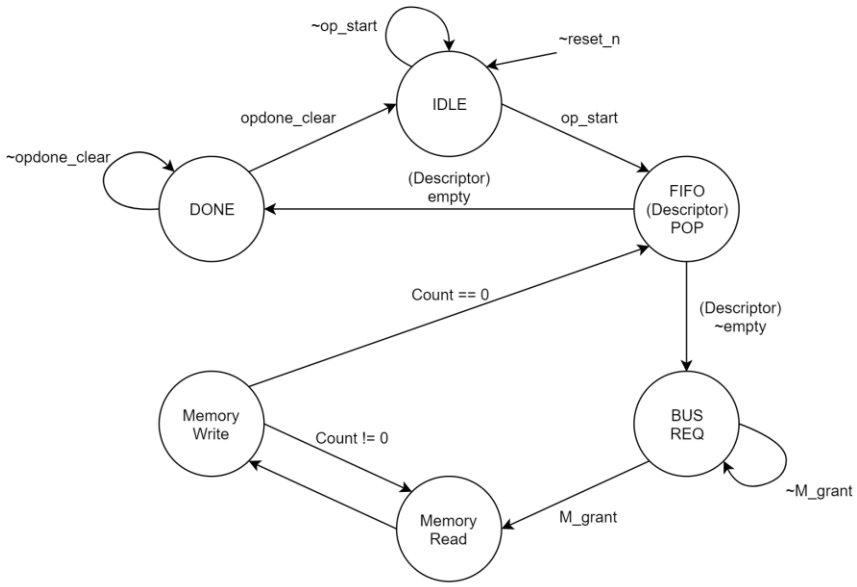
3. Schedule



<Gantt Chart – Project Schedule>

4. State transition diagram

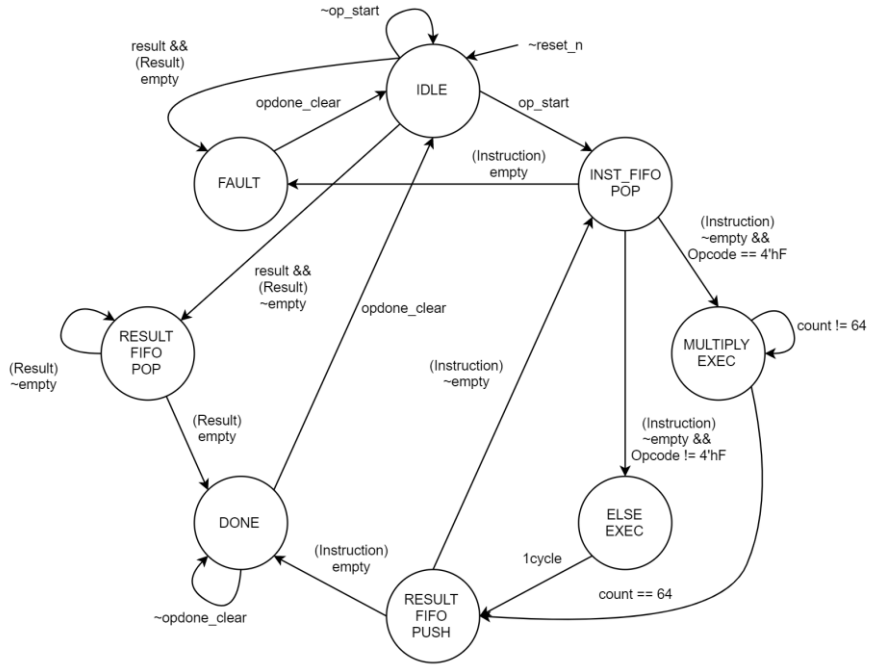
A. DMAC



DMAC의 State Transition Diagram이다. 초기에 op_start신호를 기다리고 있는 상태를 IDLE state로 정의한다. Op_start는 Slave에서 받아온다. Op_start신호가 1이 되면 Descriptor 정보가 담겨있는 FIFO에서 POP하여 Source, Destination Address, Data size를 알아낸다. 이후 Bus에 Request하여 Grant가 1이 되면 Memory간의 Data전송을 시작한다. 옮길 횟수를 Count라고 한다면 Count가 0이 될 때까지 Source Address로부터 Destination Address까지 Memory Transfer를 진행한다. Count가 0이 된다면 위의 행동을 Descriptor FIFO가 empty가 될 때까지 반복한다. Empty가 된다면 Transfer가 완료가 되었다는 뜻이므로 DONE state로 이동시켜 메모리 이동이 완료되

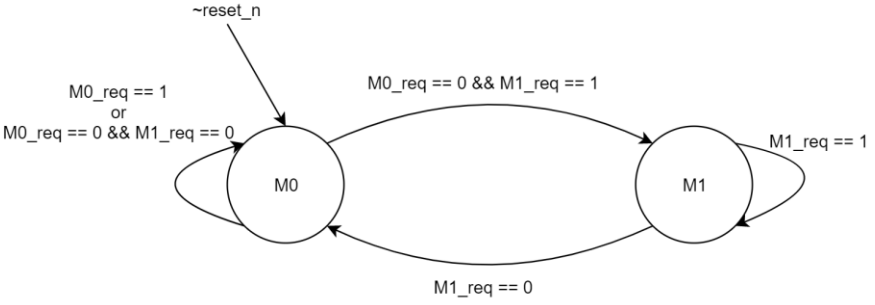
었음을 알린다. 이 때 interrupt를 발생시킨다. Interrupt를 초기화 하기 위한 opdone_clear가 1이 된다면 맨 처음 초기상태로 되돌아간다.

B. ALU



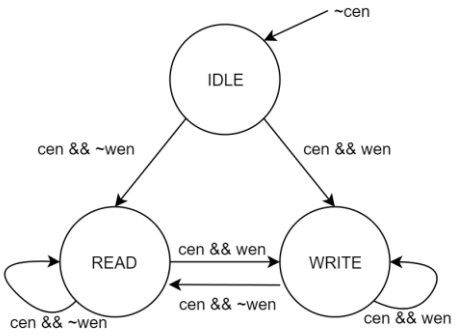
ALU Slave로부터 op_start 값을 IDLE state에서 기다리다가 1값이 들어오게 되면 Instruction fifo에서 pop하여 Opcode와 Operand A, B 주소를 분리한다. 이때 Opcode가 Multiply라면 Multiply EXEC state로 넘어가고 이외에는 EXEC로 이동한다. Multiply는 cycle이 다른 연산들에 비해 많이 필요하다고 생각하여 이와 같이 설계하였다. 연산이 모두 끝나게 되면 Result FIFO에 결과값을 2차례에 나누어 PUSH한다. Instruction FIFO가 Empty라면 DONE, 아니라면 다시 새로운 Instruction을 POP하여 위의 과정을 반복한다. 만약 FIFO가 처음부터 empty였다면 읽을 값이 없으므로 FAULT state로 보낸다. Slave에서 Result FIFO값을 RAM#3로 옮기라는 명령이 내려오면 IDLE에서 RESULT FIFO POP state로 이동한다. Result 값이 없다면 FAULT state로 이동하게 되고, RAM#3으로 Transfer를 마친 후에 DONE state로 이동하게 되어 opdone_clear값이 들어오면 다시 IDLE state로 돌아가게 된다.

C. BUS



Bus Grant에 관한 State Transition Diagram이다. Default Master는 M0이고 M1만 Request한 경우 M1에게 Grant를 부여한다. 작동은 Greedy하게 이루어진다. 한쪽이 계속 Request가 1이라면 한쪽으로 유지되는 성질을 말한다.

D. RAM

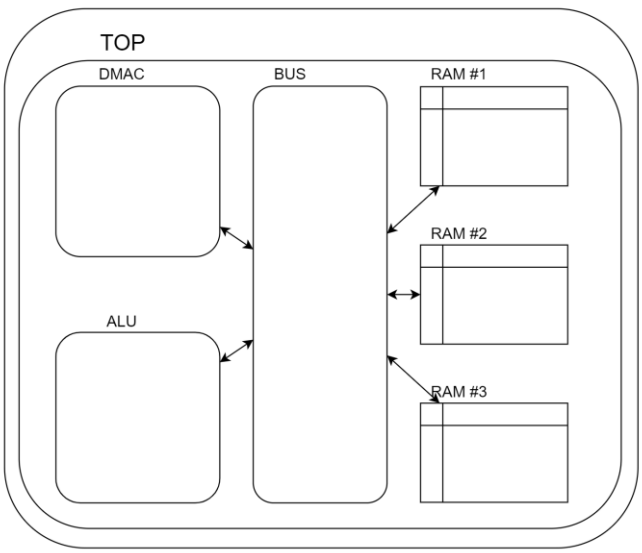


RAM은 Bus에서 Slave Selection Signal에 의해 select가 되면 cen input port와 연결되어 Chip의 Enable유무를 확인한다. Bus의 Read/Write Signal을 통하여 Data를 Read할지 Write할지 정한다. Read의 경우 Output은 0이고 Write의 경우 특정 Address에 저장 되어있는 Data를 꺼낸다.

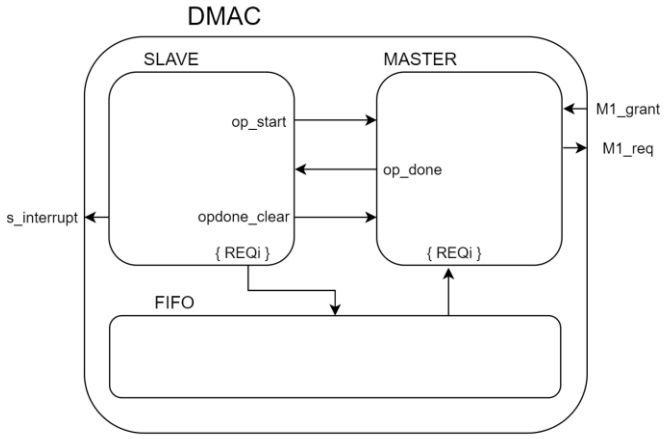
Cen값이 0이라면 select받지 못한 상황이므로 IDLE state를 유지한다. Cen값이 1이 될 경우 wen에 따라 Read할지 Write할지 결정한다. Wen이 1이면 write이다. 이 모든 행동은 1cycle만에 이루어진다. cen값은 clock과 synchronous하게 구현할 예정이다.

5. Module instance design

A. TOP

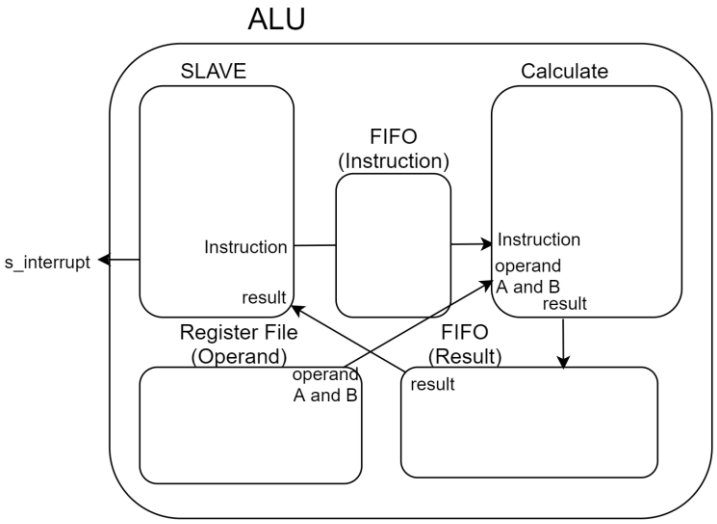


B. DMAC



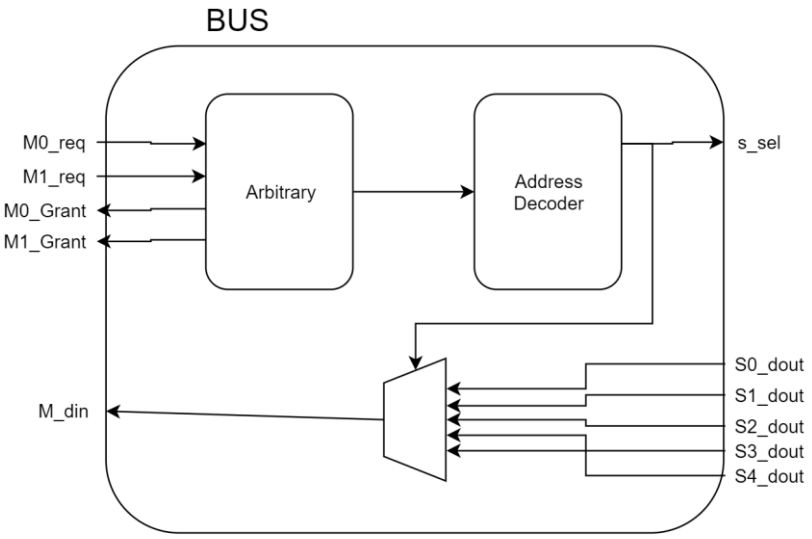
Slave가 Bus로부터 Address, Data를 받아 FIFO에 저장한 후 op_start 신호를 Master로 보내 Bus에 Master 권한을 grant로 부여받는 포괄적인 그림이다. 행동이 완료되면 Master에서 op_done신호를 보내 Slave에서 interrupt를 출력한다.

C. ALU



ALU에서 연산하기 위한 값들을 저장하는 Instruction FIFO, Operand RF, Result FIFO가 필요하다. Bus와 연결하기위한 Slave Component가 필요하고 직접 instruction을 읽어 연산을 하는 Calculate 모듈이 필요하다. 개괄적인 Data의 흐름을 그렸다.

D. BUS

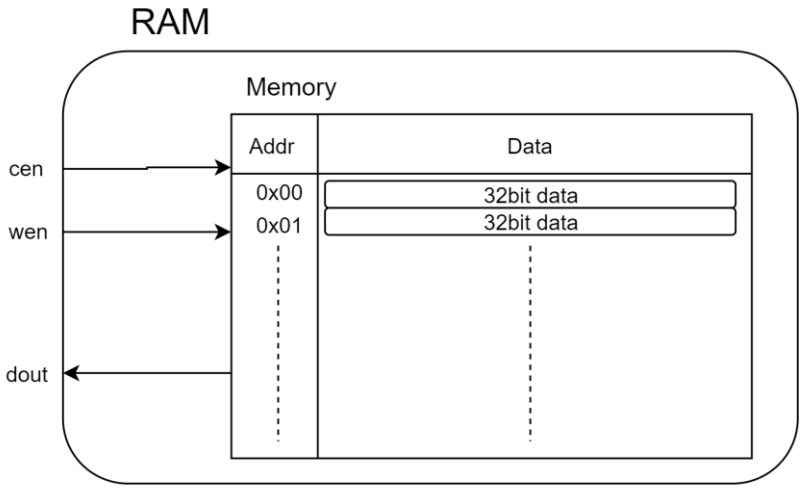


Master권한을 결정하는 Arbitrary와 Slave를 결정하는 Address Decoder가 포함된다.

Bus의 Address Decoder에서 Control해주는 Register Address Map은 다음과 같다.

Component	Address Region
0x0000 ~ 0x001F	DMAC
0x0100 ~ 0x011F	ALU
0x0200 ~ 0x023F	RAM (Instruction)
0x0300 ~ 0x033F	RAM (Operand)
0x0400 ~ 0x043F	RAM (Result)

E. RAM



Cen을 통해 Slave로 선택되었는지 판별한다. Wen을 통해 Read/Write를 판별한다.

RAM은 모두 32bit로 이루어져 있다. 개수는 최대 64개 까지의 data를 저장할 수 있다.

6. Design Verification

- 예상되는 문제점 & 검증전략

각 Sub Module의 Requirement에 대한 이해가 부족한 상태에서 설계한다면 원하는 기능을 하지 못하는 Module을 만들 수 있다. 따라서 Requirement를 꼼꼼히 따져보며 FSM설계를 완벽히 한다.

Module의 수가 많고 복잡하다. 따라서 오류가 발생했을 시 원인 규명이 어렵다. 이를 해결하기 위해 각 Sub Module단계에서 Simple Testbench를 설계하여 FSM대로 동작하여 각 부품이 원하는 방식대로 작동하는지 확인한다. 이후 각 부품들을 합친 TOP에서는 Self checking testbench with testvector를 작성하여 확인한다.

모듈마다 Flipflop이 존재하여 clock의 rising edge마다 일 처리가 일어난다. 따라서 Cycle이 정확히 맞지 않는 문제가 발생할 수 있다. 이를 설계단계에서 미리 예상을 하고 해결해야한다. 대표적으로는 ALU 모듈과 DMAC의 모듈의 소통 및 interrupt signal들을 예로 들 수 있다. 동작의 정확한 이해를 하고 있어야 한다.

ALU의 경우 본 프로젝트의 핵심 모듈이다. 가장 하는 역할이 많은 만큼 State가 많이 필요하게 되고, 부품안에 Register file, FIFO가 필요하다. 따라서 data의 저장 및 복사가 제대로 이루어지는지 확인해야 한다. 이는 64bit결과를 얻어내지만 result FIFO는 32bit이다. 이에 유의하여 데이터의 입출 순서를 확실하게 검증해야 한다.

DMAC은 Master권한을 얻고나서 일을 수행을 하게 된다. 이때 Bus로부터 grant를 제대로 받아오는지 확인하고 Interrupt신호와 그 cycle에 유의하여 구현 및 검증한다. 1cycle이라도 밀리게 되어 결과가 이상하게 나오지 않도록 한다.

마지막으로 Bus가 Master, Slave를 제대로 연결하여 결과를 전달하는지 확인한다. Bus가 data transfer의 중추이므로 더욱 철저히 검증한다.