# 디지털 논리회로2 프로젝트 제안서

## 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. Title & Object

A. Title

Term Project (BUS & Memory & ALU with Multiplier)

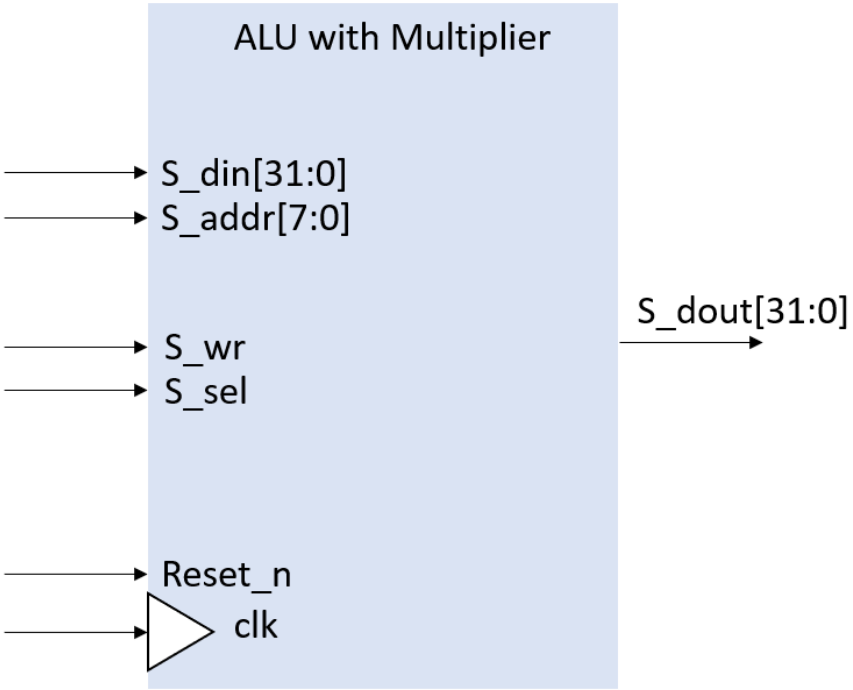
B. Object

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.

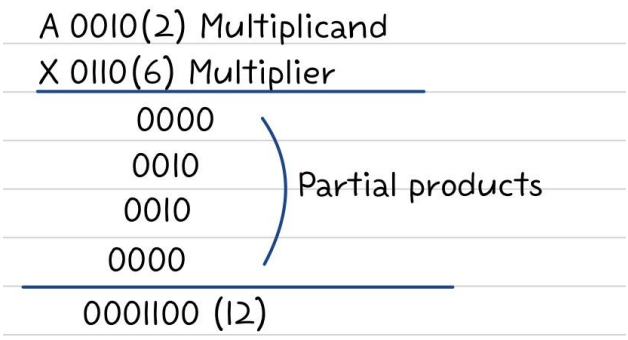
2. Component concept

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.

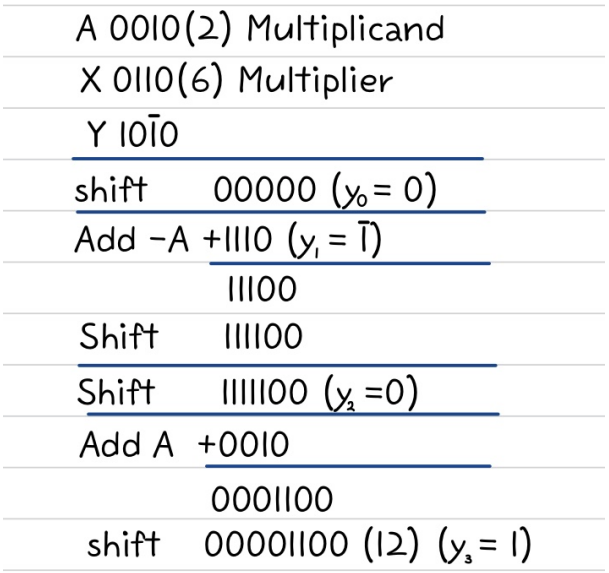



다음으로 multiplier에는 크게 Binary와 Booth 두 가지가 있다. Binary multiplication은 아래의 사진과 같다.




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.

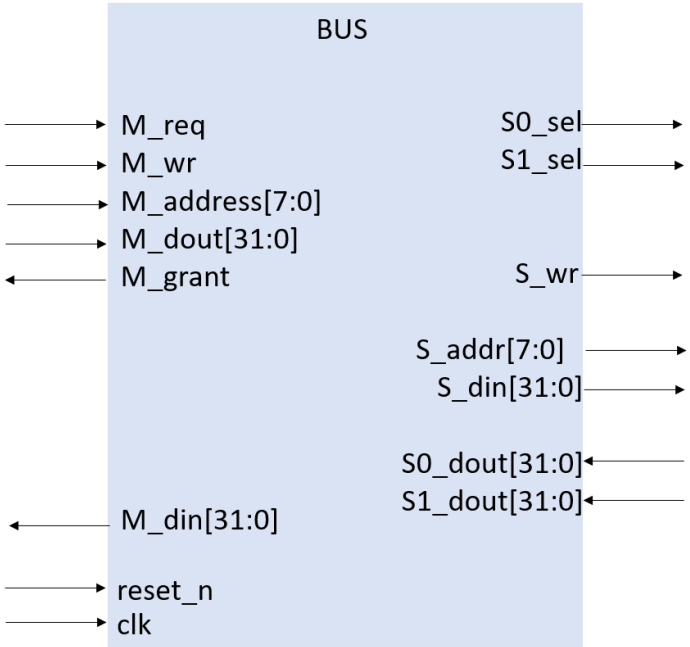
Booth Multiplication은 signed binary number의 곱을 산출하는 곱셈기이다. 아래의 사진은 0010(2)와 0110(6)의 연산과정을 나타낸 것이다. 먼저 Y값은 승수 즉 X에 LSB의 오른쪽에 0이 있다고 가정한다. 00은 0, 10은 -1, 11은 0, 01은 1이므로 Y는 10(-1)0이다. 먼저 Y가 0일 때 shift이므로, 0000에서 shift해 00000이 된다. 다음으로 Y가 1일 때 subtract shift를 진행하면 subtract는 0010->1110이고, 00000과 더해주면 11100이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



| $X_i$ | $X_{i-1}$ | 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.

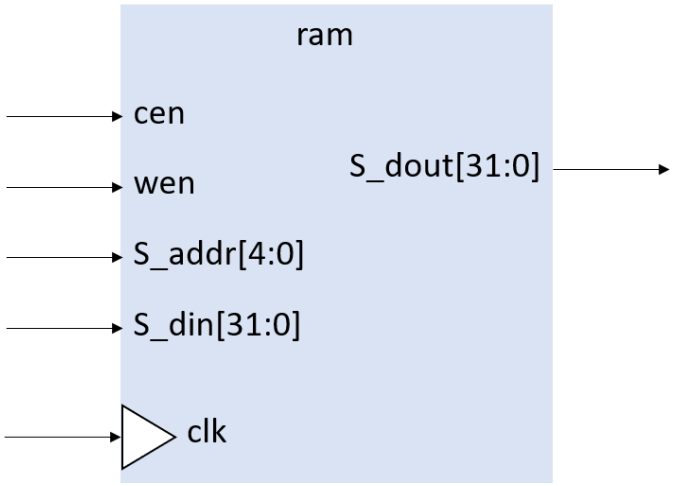
## B. Bus




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.

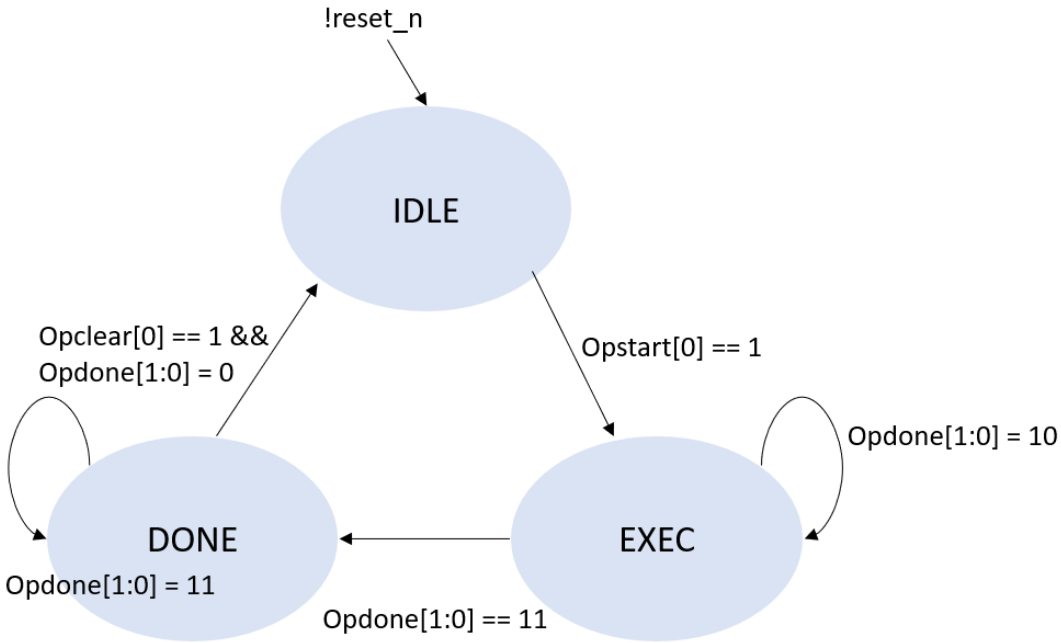
## 3. Schedule

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

계획은 추후 일정에 따라 변경될 수도 있다.

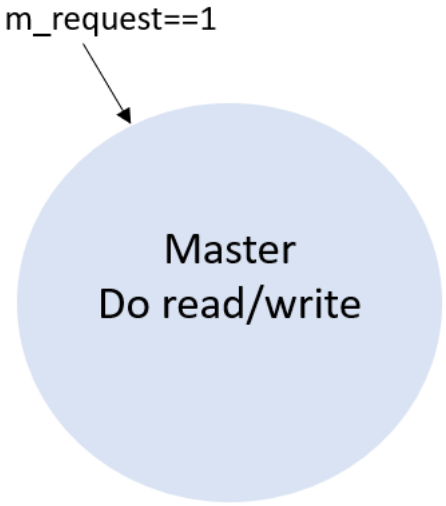
4. State transition diagram

A. ALU with Multiplier



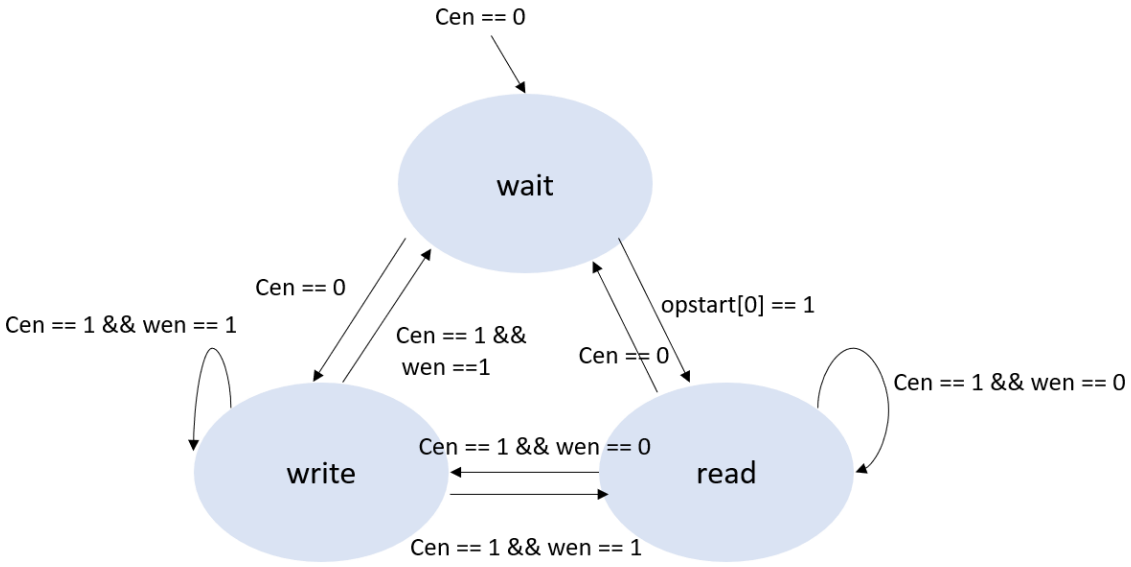

ALU with Multiplier의 State transition diagram이다. opdone[1:0] == 2'b00일 때 연산 대기, opdone[1:0] == 2'b10일 때 연산 시작, opdone[1:0] == 2'b11일 때 연산 완료를 해준다.

B. Bus



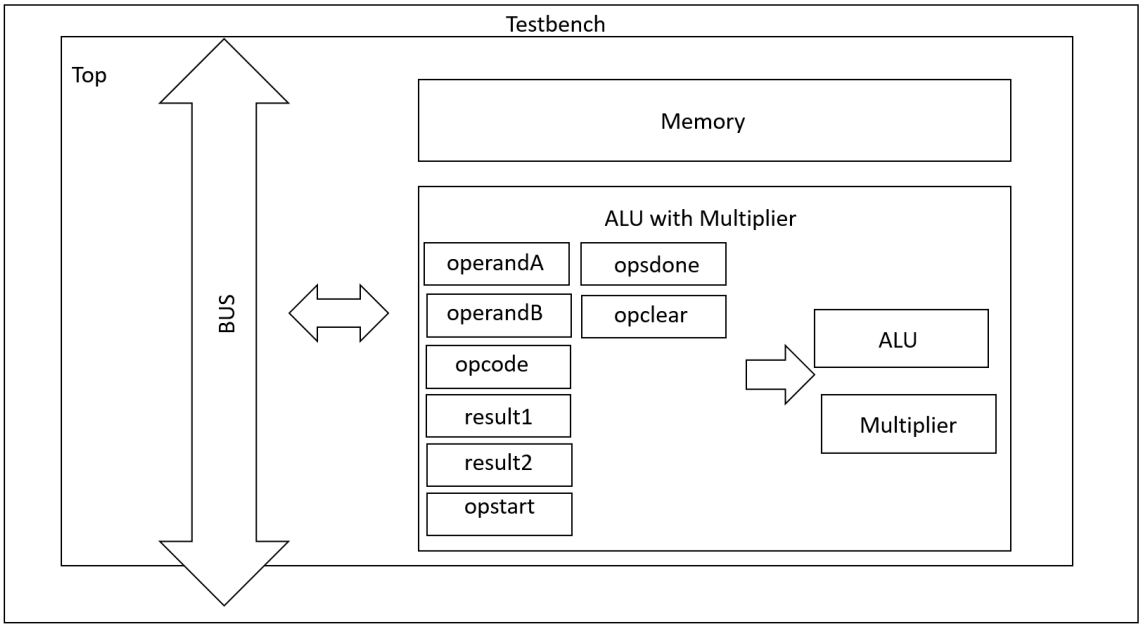
BUS의 State transition diagram이다. Req에 따라 결정하도록 한다.

### C. Memory




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

## 5. Module instance design




각 module이 Top-level module에서 어떻게 instance 되는지 도식화한 그림이다. Testbench가 BUS를 통해 ALU with Multiplier에 명령을 내리고, 그 결과를 memory에 저

장하도록 한다.

## 6. Design verification strategy

우선 project는 ram -> bus -> multiplier -> ALU-> ALU with multiplier -> Top의 순서대로 할 것이다. Ram의 코드를 구현한 후 ram을 top모듈로 설정해 코드가 잘 돌아가는지 검증할 것이다. Enable 신호, 즉 cen과 wen에 따라 write, read를 잘 수행하는지 확인할 것이다. ram에서 문제가 없다면 다음으로 bus의 코드를 구현한 후 bus를 top 모듈로 설정하여 코드가 잘 돌아가는지 검증할 것이다. Req 신호에 따라 버스의 master에서 data를 가져오는 것과 주소에 맞는 slave에 맞게 수행하는지 확인할 것이다. BUS와 ram은 Assignment11을 참고하여 검증을 진행할 것이다. Ram과 bus가 잘 돌아간다면 multiplier의 코드를 구현하여 multiplier을 top모듈로 설정해 코드가 잘 돌아가는지 검증할 것이다. Assginment9를 참고하여 32 cycle 뒤에 연산이 잘 나오는 지, op_start와 opclear에 맞게 연산이 잘 진행되는지, 연산 결과가 바르게 나오는지를 확인한다. Multiplier에 문제가 없다면 ALU의 코드를 구현하여 ALU를 top모듈로 설정하여 코드가 잘 돌아가는지 검증한다. Assignment4를 참고하여 opcode에 맞게 연산결과가 잘 나오는지 확인할 것이다. Multiplier와 ALU를 모두 구현하고 검증까지 완료했다면 ALUwMul에 instance후 코드를 구현해 top 모듈로 설정하여 코드가 잘 돌아가는지 검증할 것이다. ALUwMul에서는 Register에 따라 연산결과가 잘 저장되는지 확인할 것이다. 모든 submodule을 구현하고 검증했다면 마지막으로 Top 모듈에 instance하여 출력값을 확인할 것이다.

## 7. 예상되는 문제점

Assignment로 진행하였던 과제들을 불러와 instance 하고 필요한 것은 추가하면 되는 것이라 생각되는데, 산술연산은 operator, 즉 덧셈과 곱셈으로 구현하면 안되고, {}를 사용할 수 없으므로 기존의 코드들을 수정해야 한다. 또한 기존 과제들의 port명과 프로젝트에서 요구하는 port명이 다르기 때문에 이를 맞춰주어야 한다. 이러한 점들만 주의하여 코드를 작성한다면 평소 과제를 진행하던 것처럼 프로젝트도 진행할 수 있을 것 같다. 다만 Assignment11을 예시로 든다면, Assignment11에서는 master와 slave가 모두 2개씩 존재하였지만, 이번 프로젝트에서는 1개의 master와 2개의 slave가 주어진다. 이처럼 조건이 바뀔 경우 기존의 코드를 수정하고 변형하여 사용해야 하는 경우가 있어 이 점과 위의 작성한 점들을 고려하여 코드를 구현한다면 결과를 제대로 도출해낼 수 있을 것 같다.