디지털 논리회로2 프로젝트 제안서

ALU with Multiplier

학　　과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학　　번: 2022202065

성　　명: 박나림

1. Title & Object

A. Title

ALU with Multiplier

B. Object

Booth multiplier와 register의 집합으로 이루어진 Factorial core를 설계하고 이를 검증한다. BUS를 이해하고 설계하여 register과의 접근을 통해 검증해보도록 한다. Memory를 설계하고 BUS와 연결을 하여 Factorial core와의 전체적인 시스템을 구성한다.

2. Component concept

A. ALU with Multiplier

사용자로부터 주어진 oprand의 값에 대해 factorial 연산을 수행하는 모듈이다. operand 감산을 제외한 덧셈 및 뺄셈은 adder를 구현하여 이용하며, 이때 곱셈은 Booth Multiplier가 사용된다. 이러한 연산을 전체적으로 관리하는 것이 ALU(Arithmetic and Logical Unit)로, 산술 논리 장치라고도 한다. 덧셈, 뺄셈 등의 연산을 지원하며 CU(Control Unit)로부터 명령을 받아 CPU로 들어온 모든 데이터들을 산술, 논리 연산을 한다. 따라서 연산을 담당하는 가산기, 보수기와 같은 요소들이 존재하며, 본 프로젝트에서는 곱셈 연산을 해야 하기 때문에 이는 Booth Mutiplier가 추가되게 된다. Booth Multiplier는 2의 보수법으로 부호가 존재하는 2개의 이진수를 곱셈 연산하는 곱셈기이다. 곱셈 알고리즘은 다음과 같다.

multiplicand(피승수)와 multiplier(승수)를 곱할 때, multiplier의 하위 bit를 x1이라 하고 추가로 0을 붙여 x0으로 정한다. 두개의 bit를 비교하여 각각 0, 0이거나 1, 1이면 ASR로 shift 연산을 진행하며, 1, 0일 시 multiplicand의 보수를 더하여 뺄셈을 진행하고 shift한다. 0, 1일 때에는 덧셈을 하고 shift 연산을 진행한다. 각 연산이 끝나면 multiplier의 bit를 shift하여 다시 2개의 bit를 비교하여 위와 같은 연산을 반복한다. 곱셈하는 수의 bit 수만큼 연산이 진행되며, 64bit x 64bit 같은 경우에는 총 64번의 cycle이 걸려서 마지막 cycle 때 결과 값이 출력되는 형식이다. 이런 식으로 2bit식 비교하여 곱하는 형태는 Radix-2 Booth multiplication이고, 이 외에 Radix-4 등 더 빠르게 연산하는 곱셈기도 있다.

B. Bus

여러 component들 간에 데이터들을 전송할 수 있도록 연결해주는 component를 BUS

라 한다. 컴퓨터의 구성요소들을 서로 연결하고 데이터 전달을 위한 경로가 되며, 주소, 데이터, 제어 버스로 구성된다. Address BUS는 메모리의 주소나 I/O의 포트들을 전달하고 주소 전달은 CPU에서 메모리로만 가능하다. Data Bus는 각 구성요소들에서 양방향으로 데이터 전달이 가능한 버스를 사용한다. Control Bus는 Read, Write의 제어 신호가 전달되며 마찬가지로 양방향으로 데이터 전달이 가능한 Bus를 사용한다. CPU와 I/O unit간에는 input, output, inputerrupt가 있다. input은 Read신호를 전달하고 output은 Write신호를 전달한다. inputerrupt는 요청을 보낼 시 CPU에 입출력 작업을 요청할 수 있다. 요청을 하면 I/O unit이 CPU에 입출력 작업을 시작할 것을 요청하고, 확인을 하면 CPU가 입출력 동작을 수행할 것을 I/O unit에게 알리는 형식이다. 이러한 BUS는 새로운 component들을 추가하기가 쉽고 가격이 저렴하다는 장점을 가지고 있다.

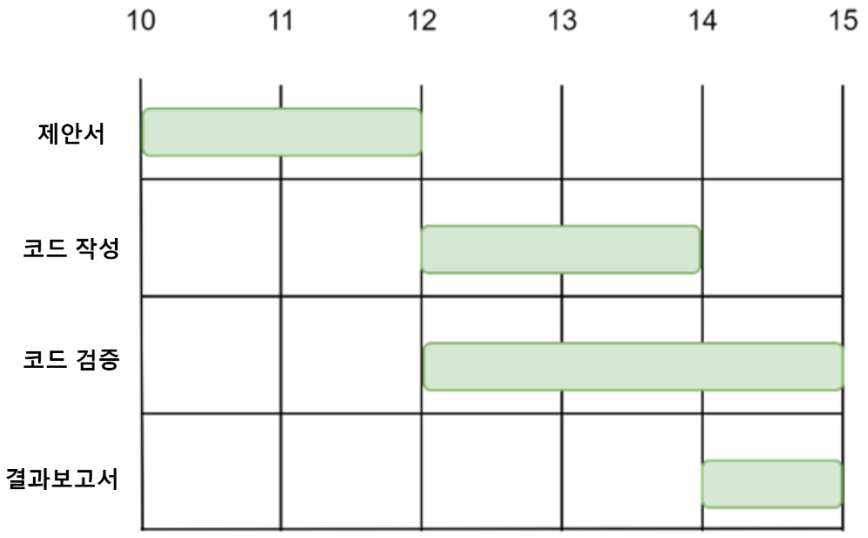
본 프로젝트에서는 1개의 master interface와 2개의 slave interface를 가지며 Address는 16bit, Data는 64bit을 가진다. master가 1개이므로 여러 개의 master가 Bus를 요청하는 동작은 생략된다. 처음에 master가 request signal를 1로 하여 Bus 사용 가능 여부를 확인할 때, Bus는 master에게 grant signal을 통해 Bus 사용을 허락한다. 이는 master에게 Bus에 대한 소유권을 할당해주는 것으로 볼 수 있다. 이후 소유권을 가진 master는 Address signal을 통해 slave의 Memory map에 따라 slave와 communication을 수행하게 된다. 이 동작이 완료될 시, request signal을 0으로 하여 Bus 사용을 끝내며, Bus도 grant signal을 0으로 만든다.

C. Memory

본 프로젝트에서 사용되는 memory는 모두 RAM(Random Access Memory)을 의미한다. 이러한 RAM은 임의의 address에 대해 데이터를 읽고 쓰는 memory를 말하며, 프로그램이 실행되는 동안 필요한 정보들을 저장하는 컴퓨터 memory이다. 저장된 데이터들을 순차적이 아닌 임의의 순서로 접근할 수 있으며, 다른 유형의 메모리 장치들 같은 경우에는 미리 정해진 순서로만 데이터에 접근할 수 있다는 특징을 가지고 있다. RAM은 각 작업의 진행 상황을 기억하는 동시에 빠르게 전환할 수 있도록 해주기 때문에, 메모리 용량이 커질수록 더욱 빠르고 간편하게 정보에 접근할 수 있다.

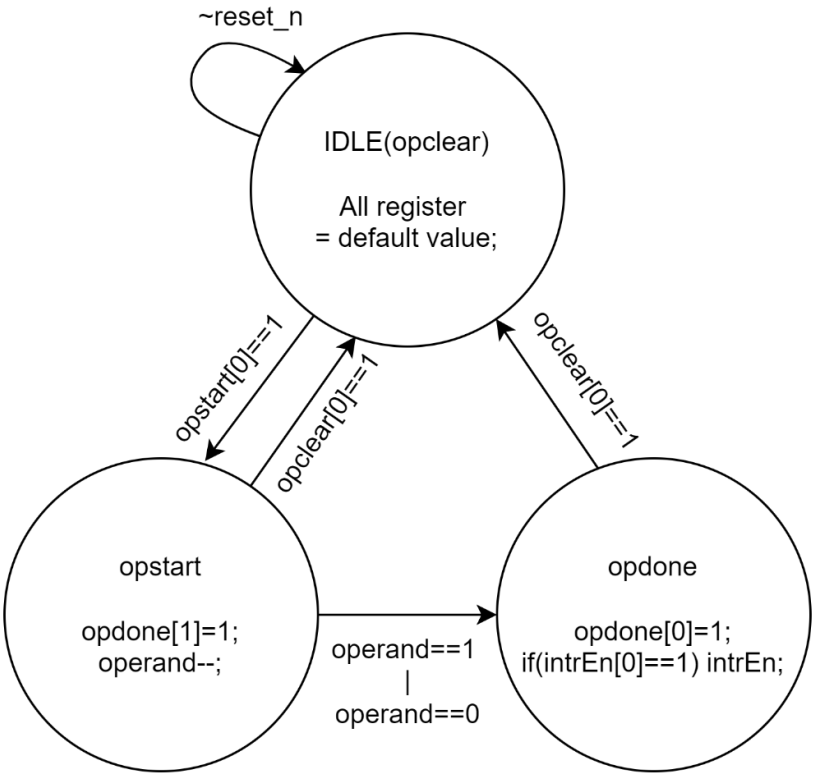
이번 프로젝트의 RAM의 Address는 8bit으로 구성되며, 이러한 address는 Bus의 LSB부터 address를 받게 된다. Data는 64bit이며 내부에 256개의 data들을 address에 기반하여 저장하게 된다. chip enable과 write enable의 신호가 모두 1일 때 address가 가리키는 memory에 Data in 값을 write하며, Data out값으로 0을 출력한다. chip enable만 1인 상황에서는 address가 가리키는 memory의 값을 Data out에 write한다.

3. Schedule

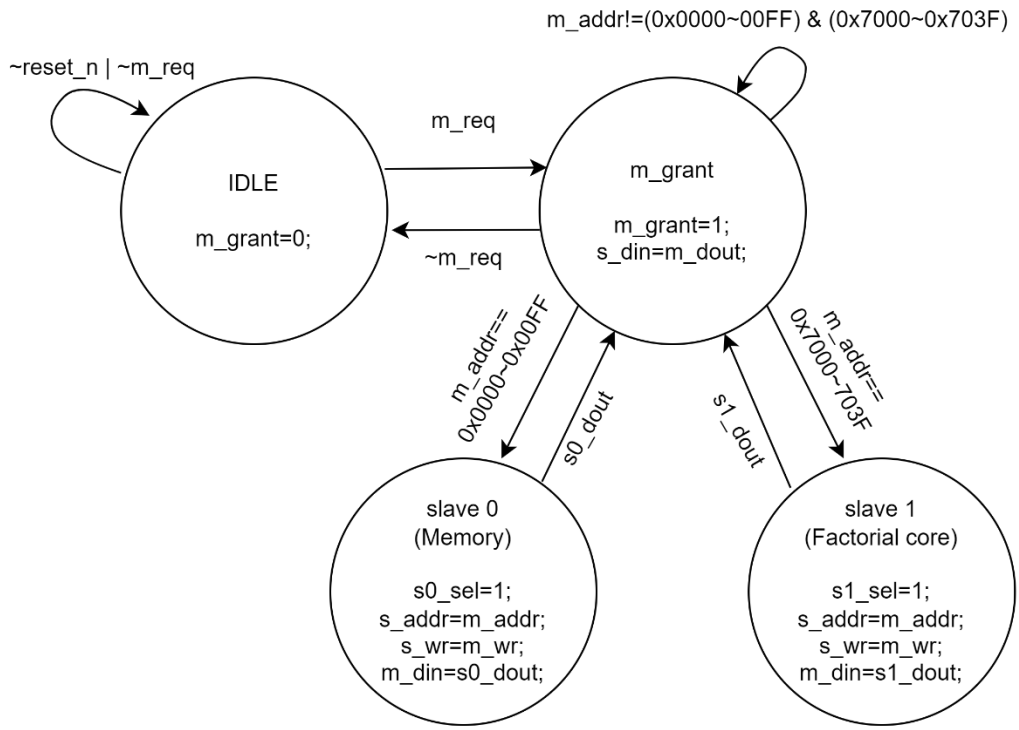


4. State transition diagram

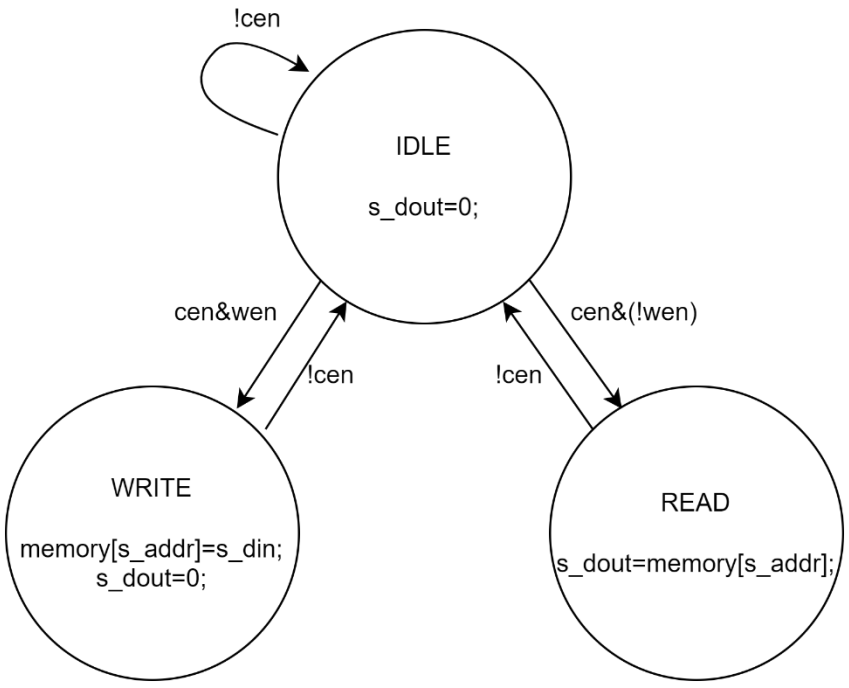
A. ALU with Multiplier



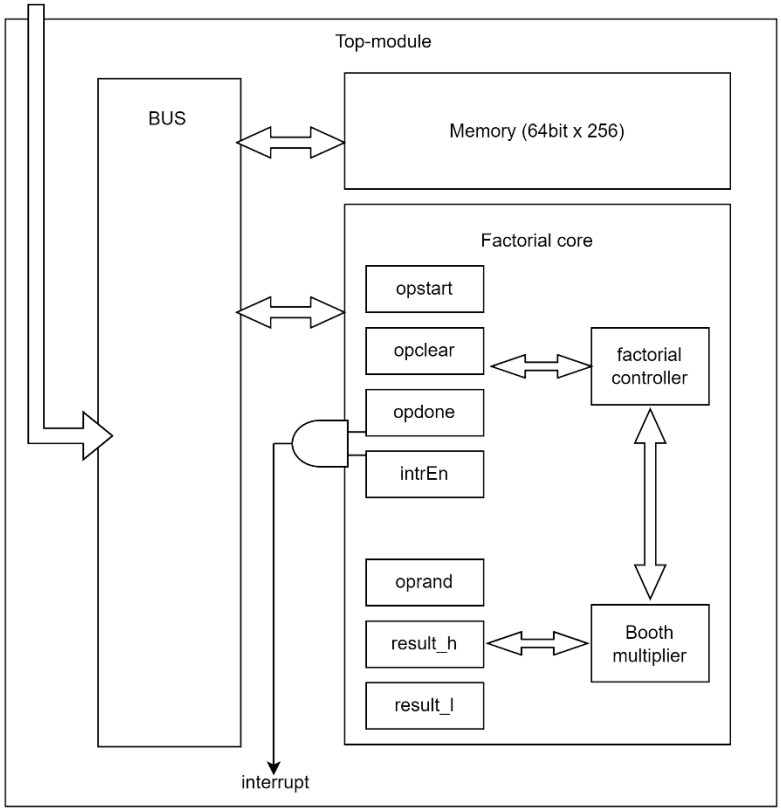
B. Bus



C. Memory



5. Module instance design



6. Design verification strategy

Bus, Memory, Factorial core를 설계할 때 각각 module 별로 하나씩 testbench로 검증한 뒤 다음 단계로 넘어간다. 우선 첫번째로 Booth multiplier module을 설계한 뒤 64bit 씩의 곱셈을 테스트해본다. 이때 opstart, opdone과 같은 signal에 따라 변화하는 state에 대해서도 next logic과 output logic을 고려하여 factorial controller도 같이 테스트를 진행한다.이렇게 Factorial core가 전체적으로 제대로 동작하는지 검증하는 작업이 끝나면, Memory를 설계한 뒤 임의의 값으로 테스트를 진행하여 적절한 주소에 값이 저장되는지 검증하여 본다. 이 작업까지 끝나면 Bus를 설계한 뒤 여러 input, output인 signal들이 정상적으로 이동하는지 테스트를 하여 검증한다.

각 module별로 개별적인 검증이 끝나면, 마지막으로 Top-module에서 Bus, Memory, Factorial core를 instance하여 연결시킨 뒤 전체적인 testbench로 검증을 진행한다. 이런 식으로 각 module별로 먼저 검증을 진행하고 다음 단계를 설계하는 식으로 프로젝트를 설계한다면 오류가 발생하더라도 문제의 원인을 빠르게 파악할 수 있을 것이다.