디지털 논리회로2 프로젝트 제안서

Factorial computation system

학　　과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 수요일 0, 1, 2

학　　번: 2022202093

성　　명: 문원주

1. Title & Object

A. Title

Factorial computation system

B. Object

그동안 배운 지식들을 집합하여 factorial 연산이 가능한 시스템을 구성한다. 시스템은 Bus, Memory, Factorial Core로 이루어진다. 이때 Slave로는 Memory와 Factorial Core가 있다. 마스터로부터 명령이 들어오면 Bus에 의해 어떤 Slave를 결정되고 해당 Slave가 다시 Bus를 통하여 마스터에게 결과값을 전달한다. 이러한 일련의 과정을 모두 이해하고 직접 구현해본다.

2. Component concept

A. Factorial Core

Bus에서 S1에 해당하는 역할을 하며 opstart, opclear, opdone, intrEn을 인자로 갖는 Factorial Controller와 Booth multiplier의 연산의 시작과 끝을 알려주는 역할을 한다. Booth multiplier는 실제 값들을 마스터로부터 전달받아 연산을 진행하고 해당 연산을 다시 Factorial Controller에 반환한다. 값을 반환받은 Factorial Controller는 다시 Bus를 통해서 마스터에 값을 전달하게 된다. Booth Multiplier에서 연산을 진행할 때 여러가지 선택지가 있지만 radix-4를 이용하여 구현할 생각이다. radix-2의 경우 AREA는 크지만 속도가 느리다는 단점이 있고, radix-16은 속도는 빠르지만 AREA가 너무 크다는 단점이 있다. 그렇기에 속도가 AREA를 모두 고려하여 radix-4로 구현하는 것이 가장 이상적이지 않을까 생각이 들어서 radix-4로 구현할 계획이다.

B. Bus

Bus는 마스터와 slave 사이를 연결해주는 역할을 하며, 아비터를 가지고 있다. 각 마스터는 Bus를 사용하기 위해 m_req 요청을 보내면 Bus는 가지고 있는 아비터와 mux를 통해 Bus를 마스터에게 할당한다. 그 후 함께 전달받은 m_address를 decoding하여 원하는 slave에게 역할을 할당하며 slave의 마스터를 다시 찾아주기 위해서 flip flop에 이전 사이클의 m_address를 저장해둔다. 그리고 다시 clk이 rising 되면 연산을 끝낸 후 전달받은 결과값을 mux에 넣고 m_address를 다시 인자로 넘겨줘 원하는 마스터에게 다시 값을 리턴하는 구조를 가지고 있다. 그러나 우리가 구현할 Bus에서는 마스터가 하나밖에 존재하지 않는다. 따라서 Bus에서 Bus를 마스터에게 할당하는 아비터는 해당 프로젝트에서 사용할 필요가 없다.

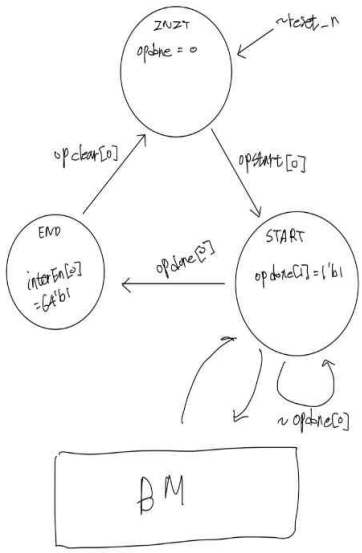
C. Memory

메모리는 값을 저장하는 레지스터 덩어리이기 때문에 flipflop 덩어리이다. cen==1 wen==1이 되면 s_in에 데이터를 저장하고 이때 wen이 0으로 바뀌면 s_out으로 s_in을 출력한다.

3. Schedule

4. State transition diagram

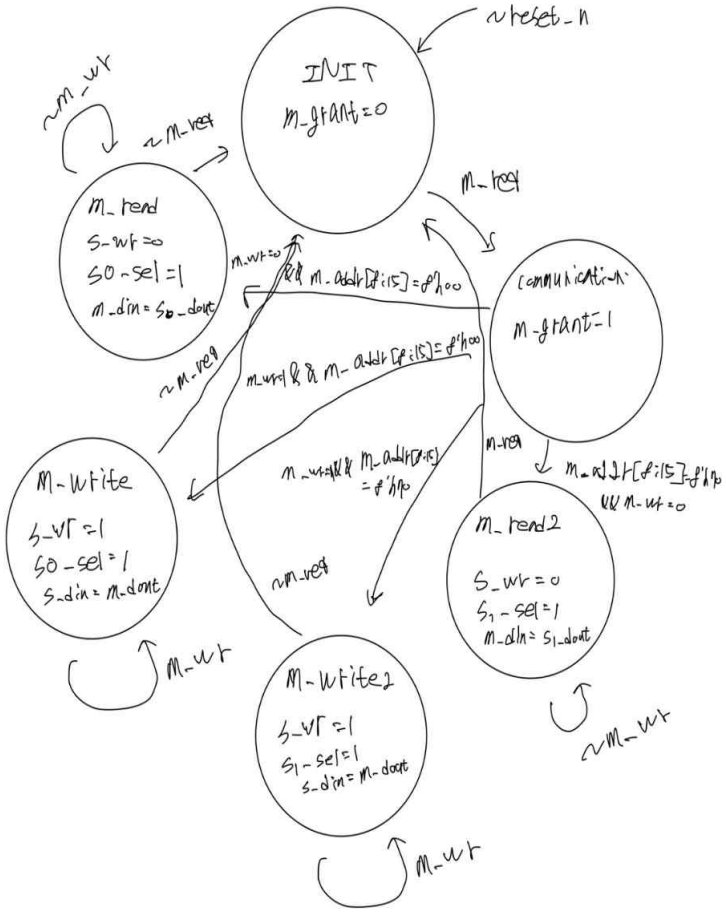
A. Factorial Core



전체적인 FSM 동작을 보면 core 중 controller 위주로 구성했다. controller는 시스템이 작동중일 때는 op_start[0]이 1로 바뀌면 op_done[1]이 1로 바뀐다. 이때 BM은 Booth Multiplier를 뜻하는데, opdone[1]이 1의 값을 가질 때는 FSM연산이 진행되고 있다는 뜻이다. 진행이 끝나면 opdone[0]을 1로 바꿔 연산이 끝났다는 사실을 알려주고 END state로 진입한다. 이때 op_clear가 입력되면 초기상태로 되돌아오는 것이다. 이때 BM에서 굴러가는 cycle에 버스로 왔다갔다하는 cycle 2번을 더한만큼 cycle

이 필요할 것이다. 이때 radix-4를 사용하여 AREA와 속도를 조절할 수 있지 않을까 생각한다.

B. Bus



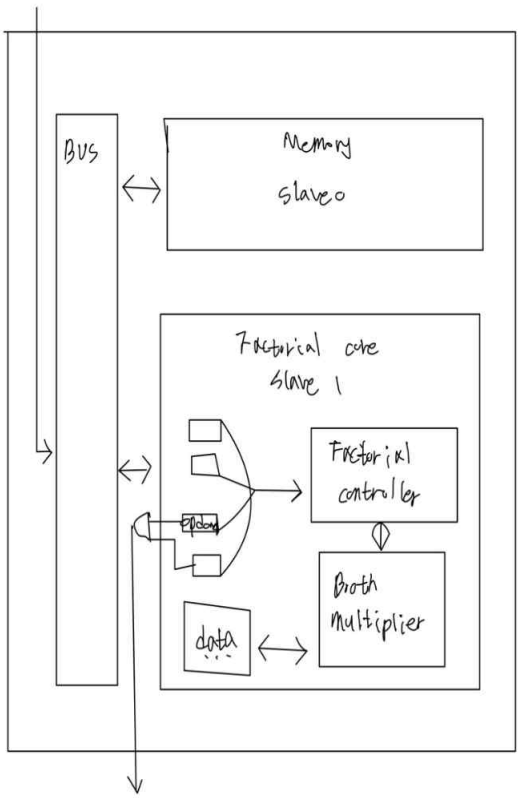
버스는 초기 m_grant가 0인 상태에서 m_req가 요청되면 아비터가 요청한 마스터에게 m_grant를 할당한다. m_grant가 1인 경우 해당 마스터에 Bus가 성공적으로 할당된 것으로 알 수 있다. slave가 두 종류기 때문에 m_read, m_write 각각 2개의 state가 필요하므로 총 6개의 state를 선언해주었다. Bus를 통해서 마스터에서 온 데이터를 slave에 전달하는 데 1cycle이 필요하고, slave로부터 전달받은 데이터를 다시 마스터에게 전달하는데 1cycle이 더 필요하다. 이때 마스터에게 할당한 slave가 다시 마스터를 찾기 위해서 이전 사이클에서 전달받은 address를 다시금 전달하여 mux의 인자로 들어가 slave에 데이터를 전달해준 마스터를 찾아갈 수 있다. 우리가 구현할 Bus에서는 마스터가 하나이기에 reset과 m_req가 같은 역할을 한다. 마스터의 요청

이 들어오면 Bus가 무조건 할당되기 때문에 아비터 또한 필요없지 않을까 생각이 든다. 이러한 불필요한 조건을 모두 제거하면 AREA를 대폭 줄일 수 있을 것이라고 생각한다.

C. Memory

메모리는 flipflop 덩어리로 데이터를 저장하고 다시 꺼낼 수 있는 기능을 가지고 있다. 데이터를 쓸 때는 decoding을 통해서 해당 register에 값을 저장하고 출력할 때는 mux를 통해서 어떤 값을 결과값으로 할당할지 결정하게 된다.

5. Module instance design



전체적인 top-module의 도식이다. 그러나 프로그램 특성상 opdone과 interEn의 AND 게이트는 필요 없을 것 같다고 생각한다. 우선, factorial controller가 일방향적이고 중간 값

을 취할 이유가 없기 때문에 해당 게이트를 삭제시켜 AREA를 줄일 수 있다고 생각한다. 그러나 문제의 조건에서 사용하라고 하였기에 구현은 하려고 한다.

6. Design verification strategy

각 모듈에 대해 직접 테스트벤치를 돌려볼 계획이다. 우선, interEn을 구현하였으니 연산 중간에 값을 취할 수 있는가 고려해봐야 할 것이고, 사실 마스터가 하나여서 Bus에서도 많은 AREA를 줄일 수 있을 것이라고 생각하지만 조건에 의해서 검증이 필요한 것들이 있다. 아비터를 뺀 상태로 Bus를 구현하려고 한다. 따라서 아비터가 없으므로 불필요한 mux들도 모두 빼려고 한다. 이렇게 구현한 Bus는 마스터가 여러개일 때 본래의 역할을 하는 정상적인 Bus는 아니므로 Bus를 직접 구현해보면서 여러 경우의 수를 생각해봐야 할 것 같다. 또한 메모리의 경우 기존의 레지스터 파일을 만들어놨기 때문에 해당 레지스터 파일을 살짝 변형하여 사용하면 되지 않을까 생각이 든다. 이러한 경우 검증이 모두 끝났기 때문에 따로 검증을 하지 않을 계획이다. 우선, Bus가 잘 작동하는지 그리고 factorial core가 잘 작동하는지를 중점적으로 확인할 계획이다.