(Factorial computation system)

(2020202037) (염정호)

I. Introduction

본 프로젝트는 booth multilplier 이용한 유사 팩토리얼 연산을 수행하는 시스템 구현을 목표로 한다.

구현 일정 및 계획

11/27	11/28	11/29	11/30	12/01	12/02	12/03	12/04	12/05	12/06
프로젝트 요구사항 분석									
	프로젝트 구현								
							수정 및 검증 , 보고서 작성		

II. Project Specification

본 시스템은 Factorial core, BUS, Memory로 구성되며 Testbench(master)를 이용해 동작을 제어한다. 팩토리얼 코어는 버스를 통해 접근 할 수 있는 레지스터의 집합(opstart, opclear, opdone, intrEn, operand, result_h, result_l)을 가지고 있다. 팩토리얼 코어는 해당 레지스터를 통해 외부 모듈과 데이터를 주고 받을 수 있다. 아래는 각 모듈별로 진행하는 동작을 간단하게 설명한 것으로 해당 모듈의 레지스터에 따른 동작 변화는 Design Detail에서 설명한다.
해당 시스템 동작 시 테스트벤치는 버스를 통해 팩토리얼 코어에 접근하여 해당 레지스터 값을 읽고 쓴다. 예를 들어 팩토리얼 연산을 수행하기 위해선 operand에 피연산자를 적고 opstart[0]에 1를 적어 팩토리얼 연산을 실행한다.

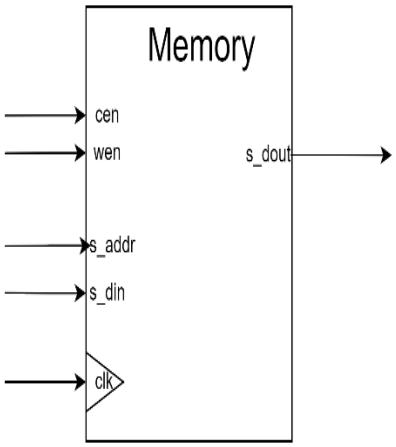
Ram :
임의의 주소를 통해 데이터를 저장하거나 저장된 데이터를 읽어오는 것이 가능한 메모리이다.
해당 메모리는 버스를 통해 FactoCore와 데이터를 주고 받게 된다. 메모리 저장용량은 64 bit * 256이며 메모리는 버스에서 출력된 s_addr[10:3] 8bit값에 따라 선택하게 된다.

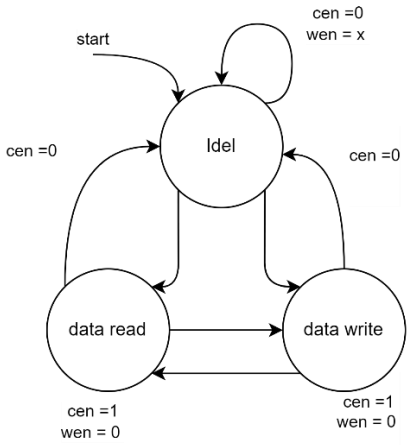
Bus :
요소들 간에 데이터를 전송할 수 있도록 하는 모듈이다. 램과 펙토리얼 코어 사이에서 데이터를 읽고 쓰는데 사용된다. 1개의 마스터인터페이스(Testbech)와 2개의 slave인터페이스(ram, FactoCore)를 가지고 있다. 버스내 Address는 16bit 버스는 masterinterface로부터 받은 address를 이용해 0x0000 ~ 0x07FF인 경우 Ram을 0x7000 ~ 0x71FF 사이인 경우 Facto core와 데이터를 주고받게 된다. Master가 해당 주소값 이외의 주소에 접근하거나 m_req가 0인 경우 두 slave모두 선택되지 않는다.

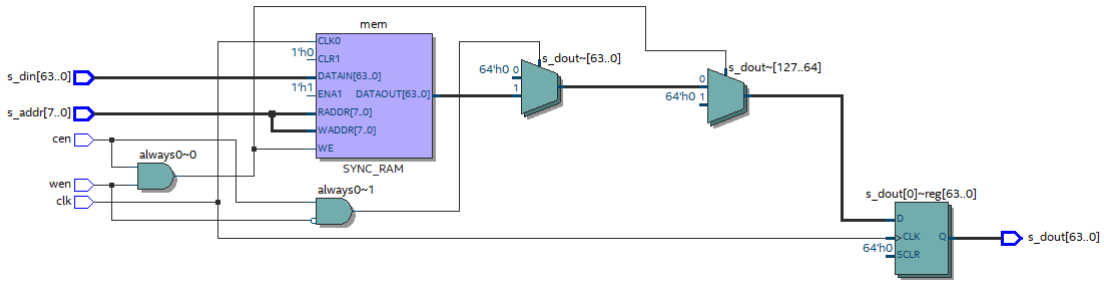
Facto core :
팩토리얼 코어는 팩토리얼 연산을 수행하고 연산 결과값을 버스로 전달하는 모듈이다. 팩토리얼의 연산 결과는 result -h와 result_l에 저장되며 각각 상위 64bit, 하위 64bit가 저장되게 된다.

III. Design Details

Module Memory RAM







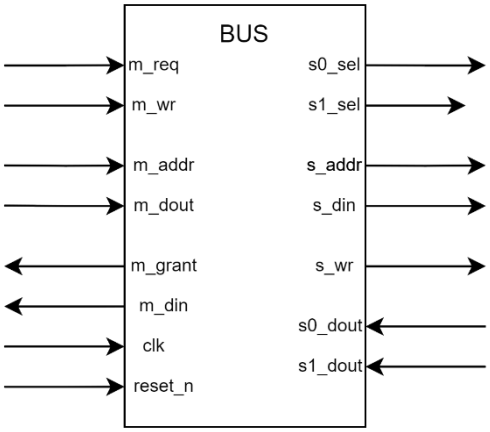
Name	Type	size	description
clk	Input	1bit	동작을 위한 clk
Cen	Input	1bit	Ram 사용 여부 결정 1입력시 데이터 읽기/쓰기 가능.
Wen	Input	1bit	Wen == 1 data write Wen == 0 data read
S_addr	Input	8bit	Address 256개의 메모리중에 어떤 메모리에 값을 읽고 쓰는데 사용할지 결정
S_din	input	64bit	Data input bus로부터 입력 받은 데이터이다. Wen이 1인 경우 s_addr에 해당하는 메모리에 S_din값을 삽입한다.
S_dout	Output reg	64bit	Wen == 0일 때 현재 s_addr에 해당되는 메모리 값을 출력한다. Wen == 1이 일 때 s_dout은 1을 출력한다.
Mem	reg	64bit *256	데이터가 저장될 메모리

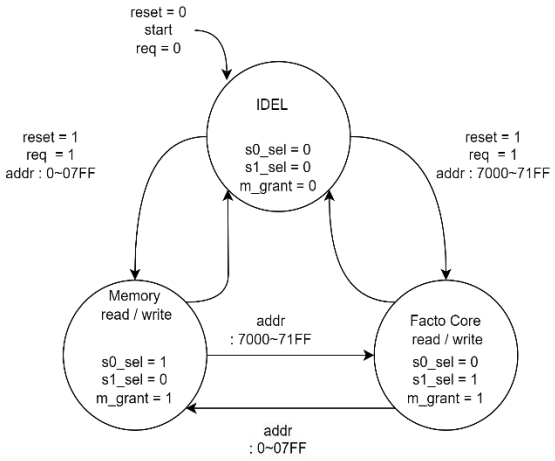
램은 데이터 저장공간으로 wen, cen값에 따라 동작이 결정된다. Cen이 0인경우 어떠한 동작도 하지 않고 s_dout으로 0을 출력한다.

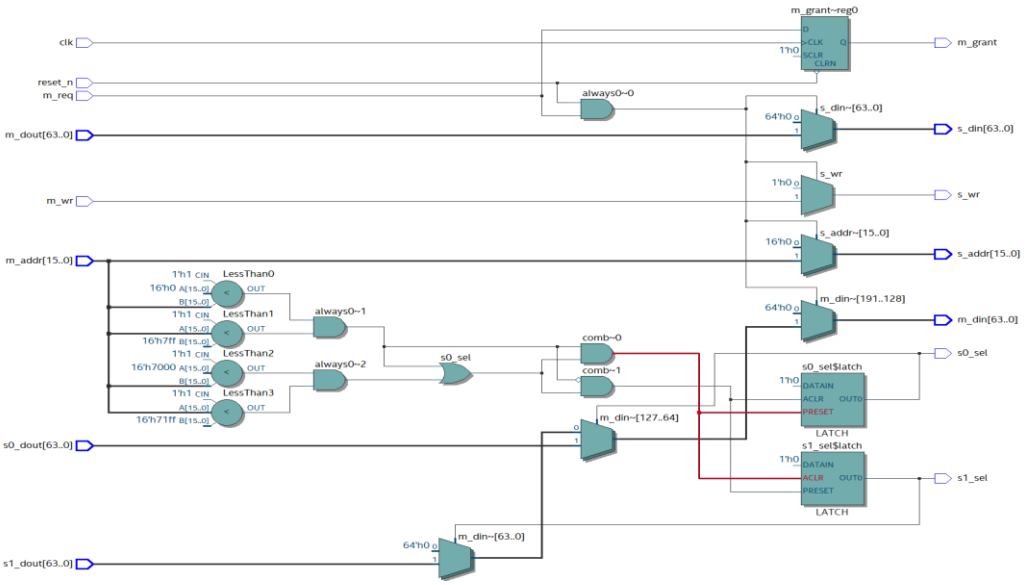
Cen = 1이고 wen = 0인 경우 데이터를 읽는 기능을 수행하게 된다. 주소 값에 해당하는 메모리로부터 데이터를 읽어와 이를 d_dout으로 출력하게 된다.

Cen = 1이고 wen = 1인 경우 데이터 쓰기 기능을 수행하게 된다. 주소 값에 해당하는 메모리에 버스로부터 받아온 데이터를 저장한다. 이 때 dout은 0이 된다.

Module – BUS



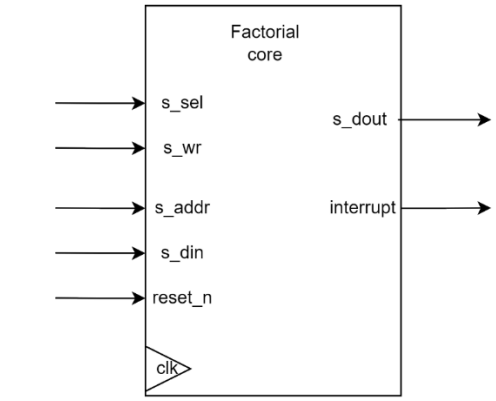


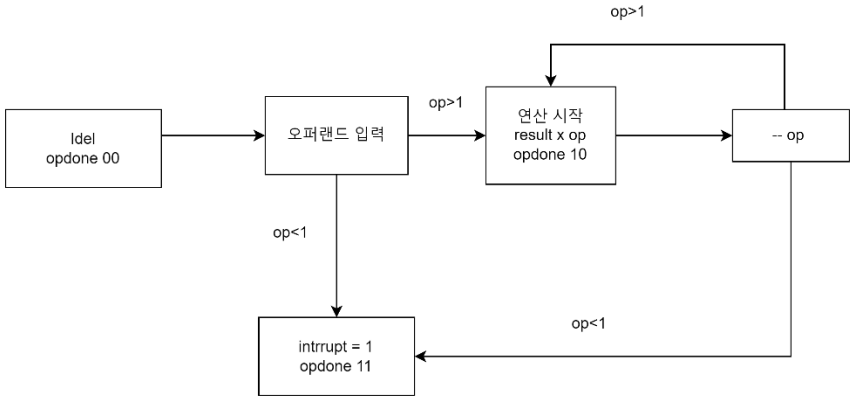


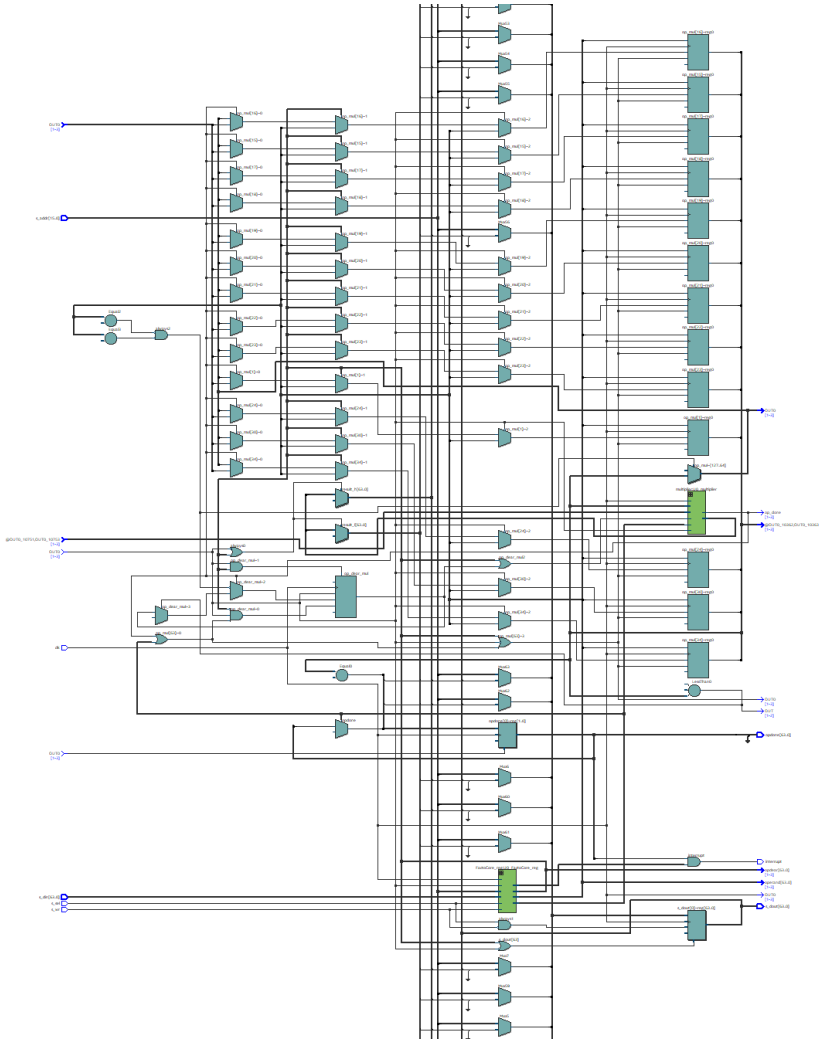
Name	Type	size	description
clk	Input	1bit	동작을 위한 clk
Reset_n	Input	1bit	0일 때 데이터 초기화
m_req	Input	1bit	버스 사용여부 결정 사용시 1
m_wr	Input	8bit	데이터 입력시 1 데이터 출력시 0
m_addr	input	16bit	master로부터 입력된 주소값
m_dout	input	64bit	마스터로부터 입력된 데이터
s1_dout	Input	64bit	s0에서 입력된 데이터
s0_dout	Input	64bit	s1에서 입력된 데이터
m_grant	output reg	1bit	req가 1일 때 master로부터 버스 사용 여부를 알린다.
s0_sel	output reg	1bit	slave0 선택 여부
s1_sel	output reg	1bit	slave1 선택여부
s_wr	output reg	1bit	마스터 wr입력에 따라 slave wr결정
s_addr	output reg	15bit	slave addr입력
m_din	output reg	64bit	slave에서 마스터로 입력된 데이터 sel값에 따라 둘 중 하나의 값 선택
s_din	output reg	64bit	마스터에서 slave로 전달될 데이터

버스는 testbench와 각 slave모듈간의 연결하는 역할을 수행한다. 해당 모듈은 master로부터 데이터를 입력 받으며 req가 1일 때 m_grant에 1을 넣음으로써 데이터 사용 가능 여부를 판별한다. Req가 0인경우 reset_n과 마찬가지로 버스 내 데이터를 초기화 하며 grant는 0이 된다. 버스는 사용자로부터 입력 받은 m_addr에 따라 slave0 또는 slave1를 선택해 데이터를 교환하게 되는데 m_addr가 0~ 07FF인 경우 slave0(ram)을 7000~71FF인 경우 slave1(facto core)를 선택한다. 만약 해당 범위이외의 주소값이 선택 된다면 두 slave모두 선택하지 않는다. 선택된 slave는 주소 값과 데이터, wr을 입력 받게 되고 마스터는 해당 slave로부터 데이터를 입력 전달받는다. Wr이 1인 경우 slave는 값을 받아오며 wr 1인 경우 해당 slave로부터 데이터를 가져온다.

Module – Factocore







Name	Type	size	description
clk	input	1bit	동작을 위한 clk
reset_n	input	1 bit	0일 때 데이터 초기화
s_sel	input	1 bit	해당 모듈 사용 여부 확인.
s_wr	input	1 bit	읽기(0) / 쓰기(1) 사용
s_addr	input	16 bit	레지스터 접근을 위한 주소 값
s_din	input	64 bit	데이터 인풋
interrupt	output reg	1 bit	연산 종료를 알려주는 레지스터 intrEn이 1일 때 사용 가능하며 탑 모듈과 직접적으로 연결된다. 연산 종료시 1을 출력한다.
s_dout	output reg	64 bit	입력 받은 address주소에 따라 opdone result_h result_ l 중 하나의 값을 출력한다
opstart	wire	64 bit	[0]번 주소에 1이 쓰여 있다면 연산을 시작하고 0이라면 연산하지 않는다.
opclear	wire	64 bit	[0]번 주소에 1이 있다면 초기화를 실행
interEn	wire	64 bit	interrupt사용 여부 확인 [0]비트에 1 입력 시 인터럽트 사용
operand	wire	64 bit	외부로부터 입력 받은 오퍼랜드 저장.
opdone	reg	64 bit	하위 2 비트에 따라 프로그램 상태 파악 00 :연산 진행 x

			10: 연산 진행중 11 : 연산 완료
result_h	reg	64 bit	곱셈기 연산 결과에서 상위 64비트가 저장될 레지스터
result_i	reg	64 bit	곱셈기 연산 결과에서 하위 64비트가 저장될 레지스터
result_mul	wire	127 bit	곱셈기로부터 나온 전체 결과값이 저장 해당 결과값을 result h,l에 나누어 저장.
op_done_mul	wire	1 bit	멀티플라이어의 연산 종료 확인을 위해 사용되는 레지스터 연산 종료 후 현재 op_mul값에 따라 동작 선택
op_clear_mul	reg	1 bit	멀티플라이어의 초기화를 위해 이용되는 레지스터
op_clear_mul2	reg	1 bit	팩토리얼 코어를 초기화 할 때 멀티플라이어도 같이 초기화 하기 위해 사용되며 op_clear_mul 과 opclear[0]의 or 연산을 통해 값이 정해지며 멀티플라이와 연결된다.
mulr	reg	64 bit	result_l와 result_h중 값을 선택해 오퍼렌드와 연산
op_mul	reg	64 bit	처음 입력된 오퍼랜드가 저장되며 1씩 감소하면 2까지 연산 실행.

연산진행 과정

팩토리얼 코어는 버스를 통해서 받은 오퍼랜드를 통해 팩토리얼 연산을 수행한다.
팩토리얼 연산을 수행하기 위해서는 오퍼랜드에 데이터를 넣은 뒤 opstart[1]번 포트에 1을 삽입해야 연산 수행이 가능하다. 이 때 opdone[1]에 1을 쓰게 된다.

```
if(opstart[0]== 1)
   opdone[1] = 1;
```

이번 프로젝트에서는 radix 2 booth multilplier가 이용되었다.
오퍼랜드 값과 이전 result_h or result_l값이 멀티플라이어의 연산자로 들어가 연산을 수행하게 되며 result_l값이 0인경우 result_h값이 들어가고 0이 아닌 경우 result l값과 연산을 수행하게 된다.

```
if(result_l == 0)
   mulr<= result_h; // if result_l == 0 mulr = result h
else
   mulr <= result_l;
end
```

처음 곱 연산하게 되는 경우 result_l의 초기 값이 1이기 때문에 문제 없이 처음 오퍼랜드 값과 연산 가능하게 구현하였다.

예시로 20!연산을 과정을 설명하겠다.
처음에 20 * 1연산을 수행하게 된다. 해당 연산 수행 후 20 * 1값이 result_l로 들어가게 되며 오퍼랜드 값은 1감소해 19가 된다 이 때 멀티플라이어 안에서는 연산이 끝난 것으로 인식하기 때문에 멀티플라이어를 한번 초기화 해준다. 멀티플라이어 내에서 연산의 끝을 확인 하기 위해 op_done_mul을 이용하였다.

```
else if(op_done_mul == 1)begin
   if(op_mul>2)begin // remain data
      op_clear_mul = 1; // set booth mul
      op_mul = op_mul - 1; // op --
      if(result_l == 0)
         mulr<= result_h; // if result_l == 0 mulr = result h
      else
         mulr <= result_l;
   end
```

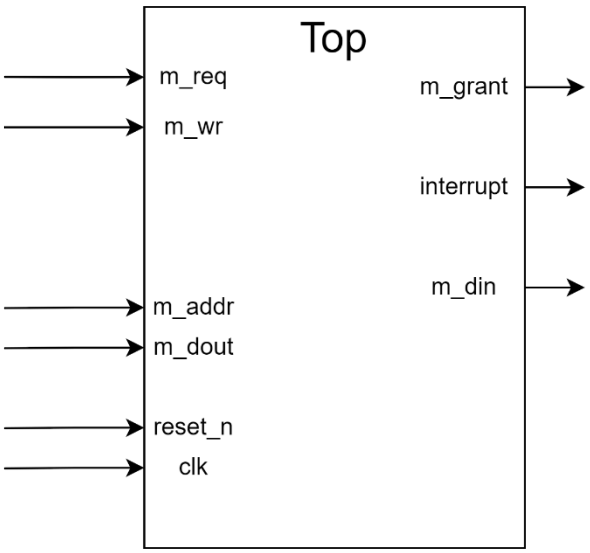
해당 레지스터 값이 1인 경우 연산이 끝난 것이기 때문에 오퍼렌드 값에 따라 연산을 더 수행할지 아님 여기서 종료할지 정해진다.
다음 연산으로 20 * 19가 실행되고 오퍼랜드가 2일 때 까지 연산을 수행하게 된다. * 2 연산 수행 후 operand가 1이되면 더 이상 멀티플라이어를 초기화 하지 않음으로써 곱셈을 종료할 수 있고 op_mul이 1이기 때문에 opdone[0]에 1을 삽입하게 된다. Interrupt의 경우 intrEn[0]번과 opdone[0]의 and연산을 통해 인터럽트 사용시 결과 값을 확인 할 수 있도록 하였다.

```
else if (opstart[0] == 1)begin
   if(operand!=0 && operand !=1)
      op_clear_mul <= 0;
   else begin
      op_mul = 1;
      op_clear_mul <= 1;
      end
```

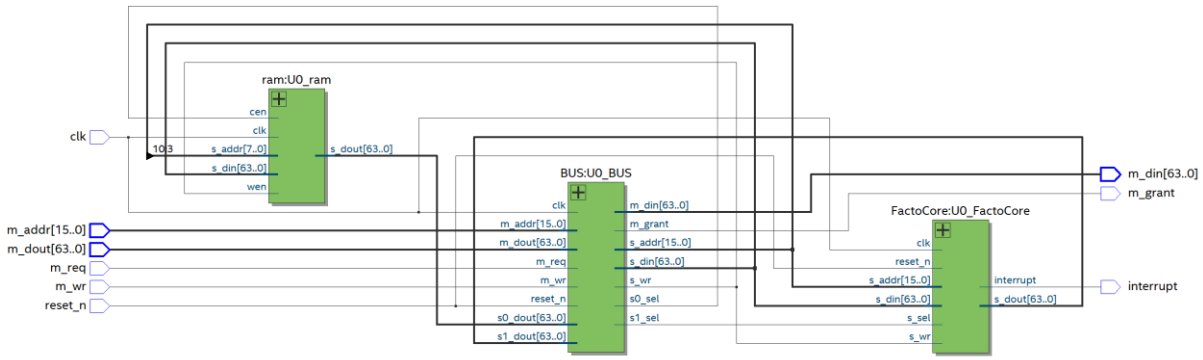
만약 operand로 0이 들어온 경우 멀티플라이어를 한번도 초기화 하지 않고 op_mul에 1을 삽입함으로써 opdone이 작동할 수 있게 하며 연산의 종료를 알린다.

팩토리얼 코어의 s_dout은 주소값에 따라 result_h, result_l, opdone 값을 내보낸다.

Module -Top



Address region	Component
0x0000 ~ 0x07FF	Memory
0x7000 - 0x71FF	Factorial Core



Name	Type	size	description
clk	input	1bit	동작을 위한 clk
reset_n	input	1bit	0일 때 데이터 초기화
m_req	input	1bit	버스의 사용을 위한 레지스터

m_wr	input	1bit	읽기 / 쓰기 설정
m_addr	input	16bit	주소값 입력 해당 주소값에 따라 s0 또는 s1 선택
m_dout	input	64bit	마스터로 입력 받은데이터
s0_sel	wire	1bit	bus에서 주소값을 통해 slave0 사용 여부 설정
s1_sel	wire	1bit	bus에서 주소값을 통해 slave1 사용 여부 설정
s_wr	wire	1bit	slave를 읽기로 사용할것인지 쓰기로 사용할 것인지 설정
s_addr	wire	16bit	slave에 입력될 주소값 버스와 연결
s_din	wire	64bit	버스에서 출력되어 slave입력될 데이터
s0_dout	wire	64bit	slave0으로부터 bus로 들어오는 데이터
s1_dout	wire	64bit	slave1으로부터 bus로 들어오는 데이터
m_grant	output	1bit	현재 bus의 사용 가능 여부를 보여주는 레지스터
m_din	output	64bit	주 개의 슬레이브 데이터중 어떤 것이 선택 됬는지 확인 가능
interrupt	output	1bit	팩토리얼 코어로부터 직접적으로 데이터를 확인 할 수 있는 출력

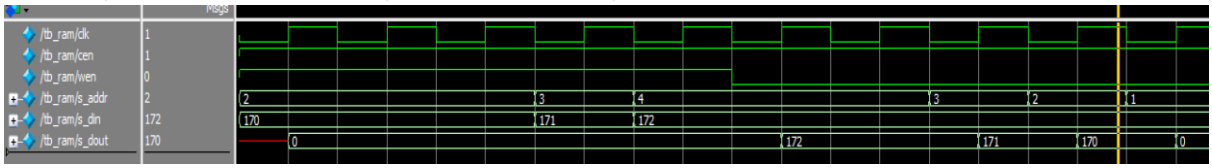
탑 모듈은 위의 3개 모듈을 연결해주는 역할을 수행한다. 각 모듈들은 탑 모듈의 버스를 통해 데이터를 입출력이 가능하다. Bus로 들어간 데이터를 통해 slave 1 or slave0중 하나를 선택에 해당 모듈에 데이터를 입력하고 해당 데이터의 출력을 버스를 통해 확인 할 수 있다. 단 interrupt의 경우 팩토리얼 코어와 직접적으로 연결했기 때문에 해당 값은 탑 모듈에서 바로 확인이 가능하다.

ram모듈의 경우 메모리 주소를 8bit만 사용하기 때문에 버스에서 출력된 주소 값 중 [10:3]부분만을 잘라 사용하였다.

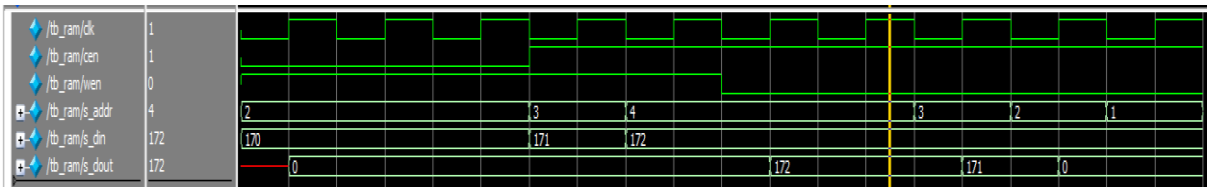
IV. Design Verifiaction Strategy and Results

Ram

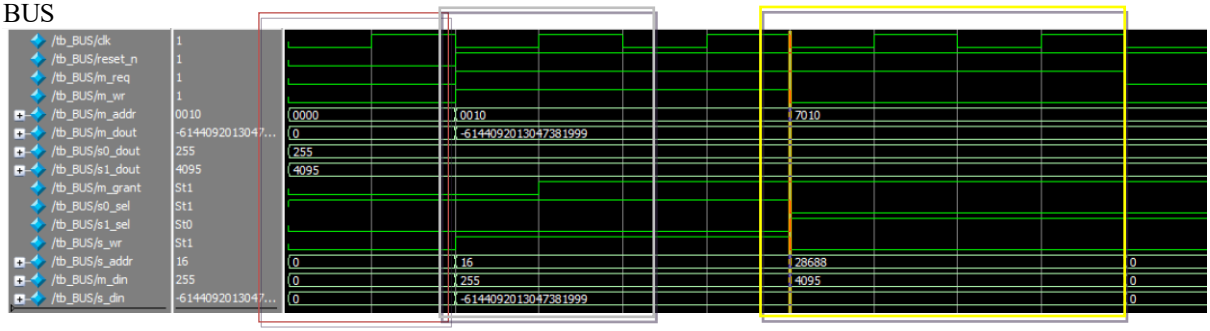
검증 : 주소별로 데이터 삽입 후 데이터 출력 여부 확인.



Cen이 0일 때 데이터 입력 x



BUS

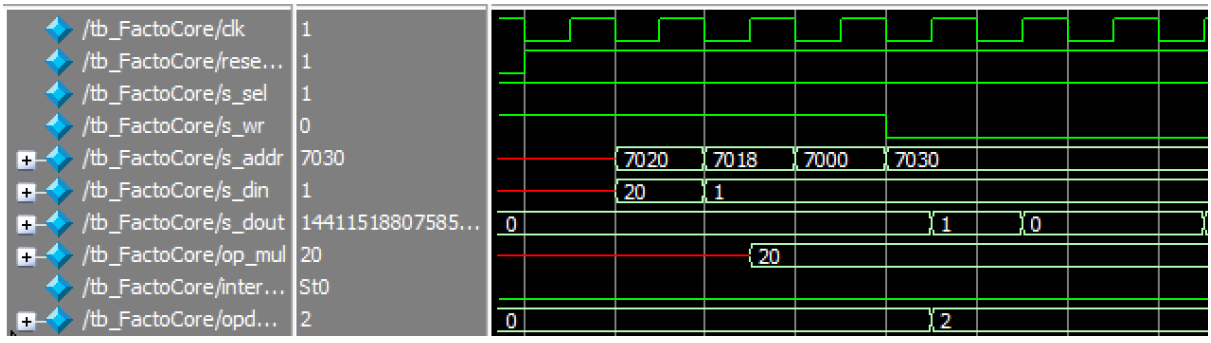


Req가 0이거나또는 reset이 0일땐 데이터가 초기 상태를 유지한다.

Req와 Reset이 모두 1일 때 m_addr에 따라 sel값과 m_din값을 선택한다. 이번 프로젝트에서는 마스터가 하나이기 때문에 s_wr과 s_din가 받는 데이터 값은 동일하다. 처음 테스트벤치에선 0~ 07FF사이의 주소가 선택되었기 때문에 slave0와 데이터를 교환하게 되고 이후 다음 어드레스의 경우 7000~71FF가 선택되어 slave1과 데이터를 교환하는 못습이다. 이후 req가 0이 되어서 데이터가 초기화 되는 모습이다.

주의 할 점은 m_grant의 경우 clk가 posedge일 때 변경되고 다른 데이터의 경우 값 변경이 바로 적용된다.

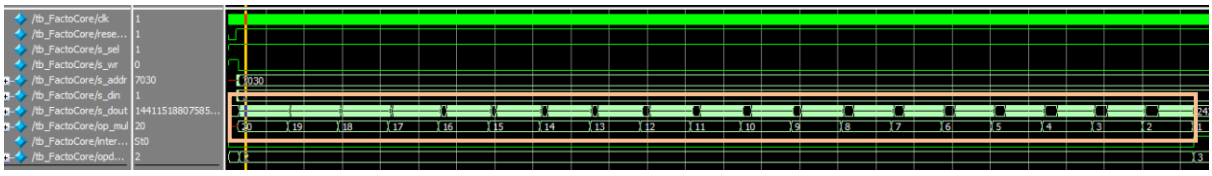
Module – Factocore



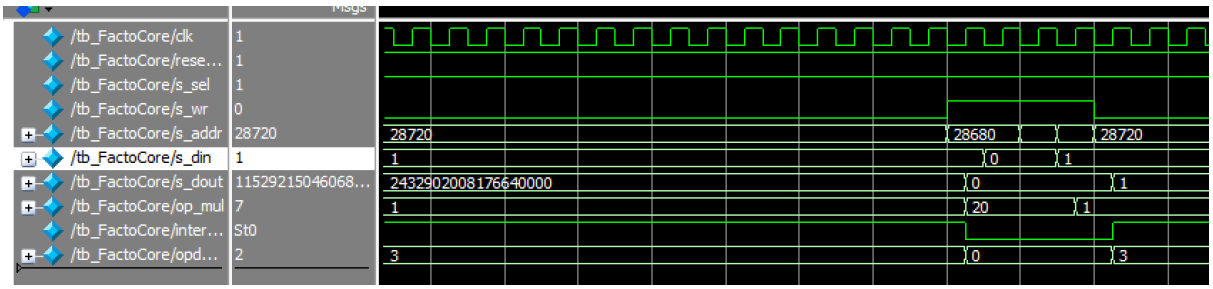
1. Operand에 20삽입
2. intrrupt값을 확인하기 위해 intrEn에 1삽입.
3. Opstart에 1삽입 ->연산 시작
4. 연산 결과 확인을 위해 s_wr에 입력 후 result_I데이터를 읽어온다.

연산에 쓰일 오퍼렌드에 값 20이 들어간 것을 확인,

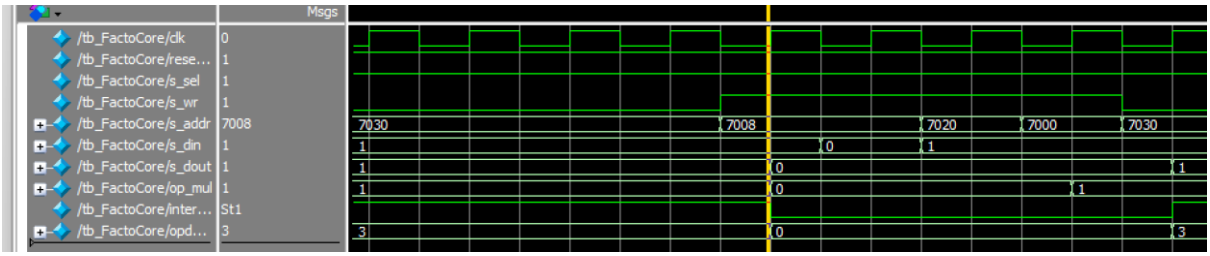
Opdone에 연산 시작시 2'b10입력



연산 진행 과정 출력 화면 opreand가 1씩깜소하면서 연산을 진행.
Operand가 2일 때 까지 곱해지며 1로 바뀌면서 연산이 종료된 것을 확인 할 수 있다.
연산 종료와 동시에 opdone이 3으로 변하며 그 위의 인터럽트 또한 상승한 것을 확인 할 수 있었다. 이후에 추가적인 입력 없이 연산은 진행하지 않는다.

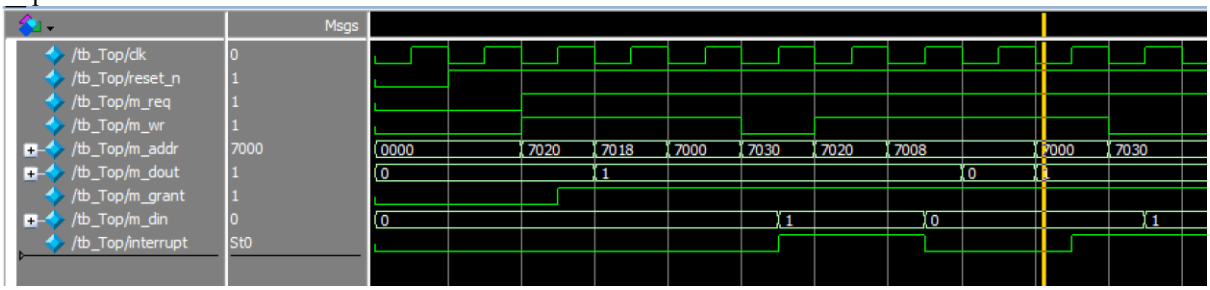


20!값이 정상적으로 출력된 것을 확인 20!값 까지는 하위 32bit안에서 확인이 가능했다.
이후 다시 연산을 위해 opclear에 1입력 데이터가 opdone, dout 초기화 이후 오퍼렌드에 새로운 값 0을 삽입된다. 해당 0은 op_mul에 삽입이 이루어지지 않고 이후 연산 시작 시 바로 인트에 1이 삽입되었고 opdone에 3 입력되는 것을 확인했다.

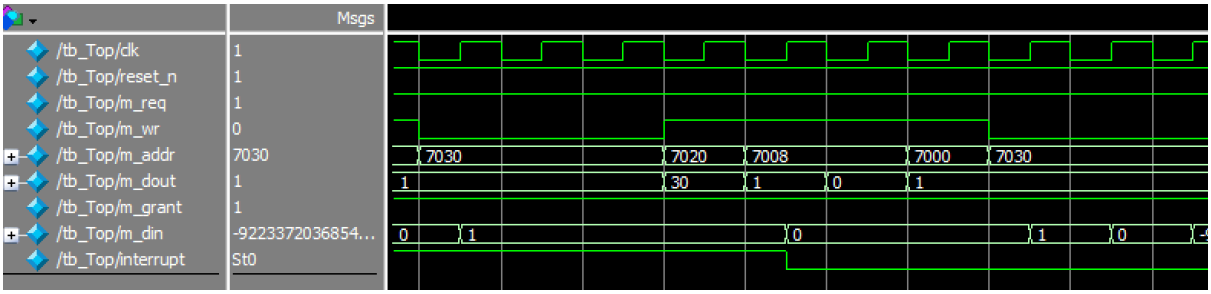


이후 operand에 값이 1이 들어올 경우 마찬가지로 데이터연산 없이 바로 3을 출력 하는 것을 확인했다.

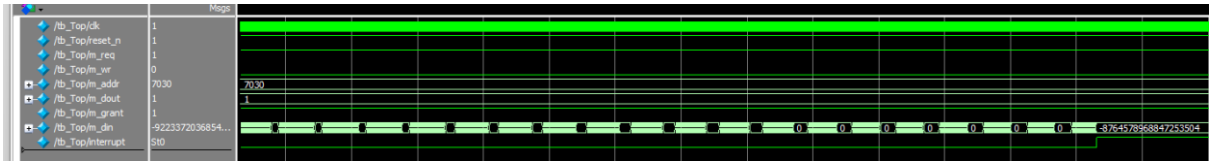
Top bus -factorial core



1. 버스 사용 가능 여부 확인
2. 오퍼랜드 값이 1이 들어왔을 경우 연산을 진행하지 않고 인터럽트에 1 출력을 확인
3. 그 뒤 오퍼랜드가 0이 들어가도 연산을 실행하지 않고 인터럽트가 발생하는 것을 확인했다.

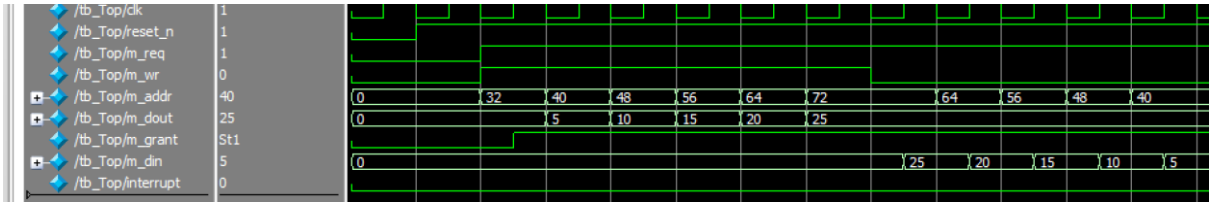


1. 오퍼랜드에 30을 넣고 프로그램 실행



연산 종료 후 알맞게 인터럽트가 발생하는 것을 확인 할 수 있었다.

Top bus -ram



1. Grant 정상 발생 여부 확인
2. 데이터 입력 값 확인
3. 데이터 출력 여부 확인.

V. Conclusion

이번 프로젝트에서는 팩토리얼과 유사하게 작동하는 모듈을 구현했다. 이번 프로젝트를 하면서 느낀 것은 조건에 맞는 모듈을 구현하는 것은 어렵다 이다. 팩토리얼 연산 구현 자체는 그리 어렵지 않았으나 조건에 맞게 연산을 구현 하는 부분에서 매우 시간이 많이 소비되었다. 최종 테스트벤치가 돌아가지 않아 제안서를 반복해서 읽어야 했고 그 때 마다 모듈 수정이 필요했다. 내가 구현한 부분에서 고쳐야 할 부분도 많았고 해당 부분을 통째로 수정해야 하는 일도 생겼다. 결국 최종 테스트 벤치 실행은 실패했다. 프로젝트 시작 전 제안서를 충분히 읽어보고 요구조건을 잘 파악했으면 이라는 아쉬움이 남았던 프로젝트였다.

이번 과제를 통해 요구사항을 파악하는 것과 이를 제대로 수행 할 수 있는 모델을 구현하는 것의 어려움을 느낀 것 같다. 기타사항을 제외하고 연산 자체 만을 본다면 문제 없으나 기타 사항이 포함되었기에 어려웠던 프로젝트였다.

VI. Reference

이준환/디지털논리회로2/광운대학교/2023
이형근/컴퓨터공학기초실험2/광운대학교/2023