Factorial computation system

2022202065 박나림

Abstract

Factorial 연산을 수행하는 프로그램을 만드는 것이 본 프로젝트의 목표이다. 주요 구현 모듈로는 BUS, Memory, Factorial core가 있으며 각각의 모듈은 전체적으로 TOP에서 instance되어 testbench로 조정되어 검증된다. BUS의 데이터는 slave 선택을 통해 slave 0은 memory, slave 1은 Factorial core로 이동된다. 이때 접근될 수 있는 것은 Factorial core 안에 있는 7개의 register이다. 이러한 register를 offset을 이용해 계산하고 그에 맞는 연산을 수행한다. 따라서 Factorial core는 내부에 Register file, Factorial controller를 가지고 있고, 이러한 연산은 Radix-2 Booth multiplier를 이용하여 계산된다. 이번 프로젝트에서는 BUS, Memory, Register File, Factorial controller까지만 검증되었다.

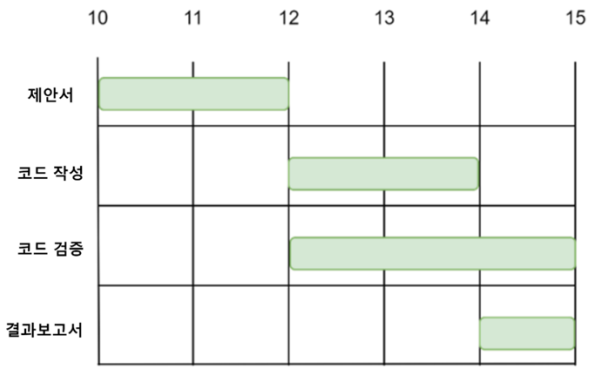
I. Introduction

Factorial 연산을 사용자가 원하는 대로 수행하게 하는 프로그램을 구현하는 것이 목적이다. 해당 시스템은 Factorial core, BUS, Memory로 구성되며 testbench로 동작을 제어 받는다. Factorial core와 Memory는 BUS의 slave 선택을 통해 접근이 가능하며, Factorial core는 이러한 접근을 받는 register 집합을 가지고 있다. Factorial 시스템의 수행 내용은 다음과 같다.

testbench가 BUS를 통해 Factorial core에 접근하여 각각의 register에 read/write를 수행하면 시스템이 시작되는 형식이다. 사용자가 원함에 따라 interrupt를 발생시킬 수도 있다. intrEn register와 opdone register값을 AND연산하여 interrupt를 만드는 것으로, interrupt를 사용하는 경우에는 시작하기 전에 intrEn register에 1을 write해야 하며, testbench는 interrupt의 값에 따라 연산 진행 상황을 파악할 수 있다. interrupt를 사용하지 않는 경우에는 연산이 종료될 때까지 매 사이클 opdone register 값을 읽어서 연산의 진행 상황을 파악한다.

Factorial core의 연산에서 operand의 값은 음수 혹은 $2^{31} - 1$을 초과하는 값은 들어올 수 없으며, 해당 범위 내에서만 연산을 진행할 수 있다. 0!의 대한 예외 처리도 하여 계산할 수 있으며, 이러한 곱셈 연산은 Booth multiplier를 이용하여 실행된다.
연산이 끝난 뒤 testbench는 factorial의 계산 결과를 Memory에 쓰며, 다음 연산을 진행할 수 있도록 opclear register를 통해 초기화 시킬 수 있다.

-일정 및 계획



II. Project Specification

1. BUS

BUS는 여러 component들 간에 data를 전송시킬 수 있도록 연결해주는 component이다. 이러한 BUS는 새로운 component들을 추가하기 쉽고 가격이 저렴하다는 장점을 가지고 있다. [1] 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. 이때 master의 request signal이 1인 동안에는 grant signal이 0이 되지 않으며 계속 communication을 수행한다.

2. 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. [1]

이번 프로젝트에서 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. Factorial core

Factorial core는 BUS와 연결된 slave component로, 주어진 operand의 값에 대해 factorial 연산을 수행한다. Factorial core는 register의 집합과 Factorial controller, Booth multiplier로 구성된다. Factorial core의 slave interface는 slave address를 이용하여 offset을 계산한다. 이를 이용하여 register에 data를 write하거나 read할 수 있다. register 집합은 register file 모듈로 구현되며, 동작 설명은 다음과 같다.

-Register File

Register File은 Factorial core안에 있는 7개의 64bit Register로 이루어진 모듈이다. Address와 w/r signal을 통해 read 또는 write를 할 수 있다. Register에서 사용되지 않는 bit는 reserved이며, 각 register의 default value는 reset이 되었을 때 초기 값을 의미한다. 7개의 register는 다음과 같이 구성된다.

(1) opstart: 0x00의 offset을 가지며 [0]bit에 1이 써지면 연산을 시작한다.
(2) opclear: 0x08의 offset을 가지며 [0]bit에 1이 써지면 opclear, intrEn, operand를 제외한 나머지 register의 값들을 reset 시킨다. 이는 clock과 synchronous 하다.
(3) opdone: 0x10의 offset을 가지며 [1]bit에 1이 써지면 동작 수행 상태, [0]bit에 1이 써지면 연산 종료 상태를 나타낸다.
(4) intrEn: 0x18의 offset을 가지며 [0]bit에 1이 써진 경우에만 interrupt를 발생시킨다.
(5) operand: 0x20의 offset을 가지며 factorial 연산의 피연산자를 나타낸다.
(6) result_h: 0x28의 offset을 가지며 factorial 연산 결과의 상위 64bit를 저장한다.
(7) result_l: 0x30의 offset을 가지며 factorial 연산 결과의 하위 64bit를 저장한다.

이러한 register의 data들이 factorial controller에 전달되면, Radix-2 Booth multiplier를 통해 곱셈 연산을 시작한다. 이에 대한 알고리즘은 다음과 같다.

-Radix-2 Booth multiplier

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 bit수만큼 연산이 진행되며, 64bit x 64bit 같은 경우에는 총 64번의 cycle이 걸려서 마지막 cycle때 결과 값이 출력되는 형식이다. [2]

4. TOP

BUS, Factorial core, Memory를 instance하여 연결한 모듈이다. TOP 모듈의 input port로 모듈 내에 있는 BUS의 master port에 접근시킬 수 있다. 이러한 TOP 모듈은 memory mapped I/O 방식을 통해 testbench 같은 외부에서 주소로 slave device에 접근이 가능하다. Memory map은 다음과 같다.

0x0000 ~ 0x07FF (0000 0000 0000 0000 ~ 0000 0111 1111 111)	Memory (slave 0)
0x7000 ~ 0x71FF (0111 0000 0000 0000 ~ 0111 0001 1111 1111)	Factorial core (slave 1)

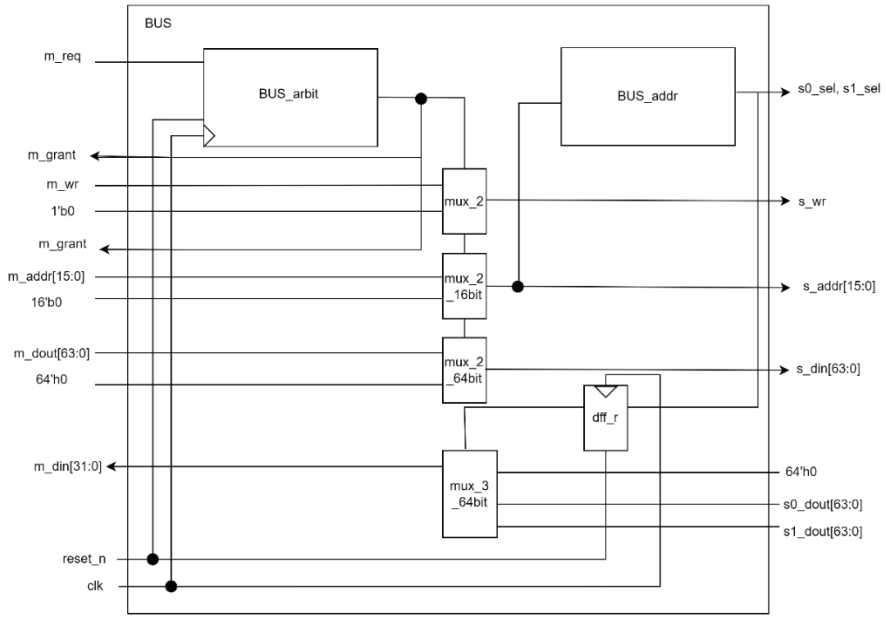
III. Design Details

1. BUS

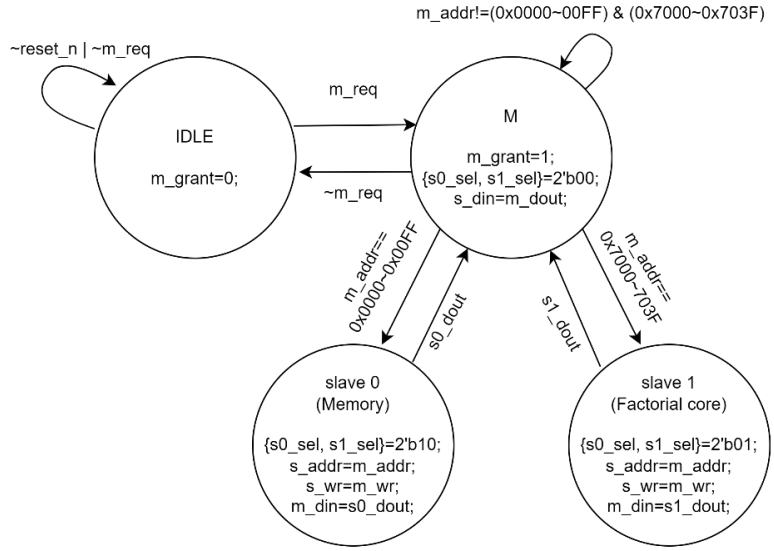
1) pin description

Input	clk	1 bit	clock
	reset_n	1 bit	Active low reset
	m_req	1 bit	Master request
	m_wr	1 bit	Master write/read
	m_addr	16 bits	Master address
	m_dout	64 bits	Master data output
	s0_dout	64 bits	Slave 0 data out
	s1_dout	64 bits	Slave 1 data out
Output	m_grant	1 bit	Master grant
	m_din	64 bits	Master data input
	s0_sel	1 bit	Slave 0 select
	s1_sel	1 bit	Slave 1 select
	s_addr	16 bits	Slave address
	s_wr	1 bit	Slave write/read
	s_din	64 bits	Slave data input

2) block diagram



3) FSM

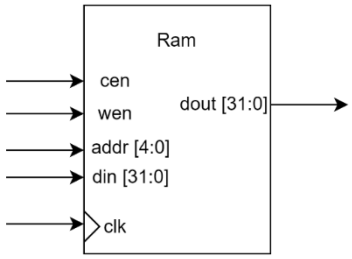


2. Memory(ram)

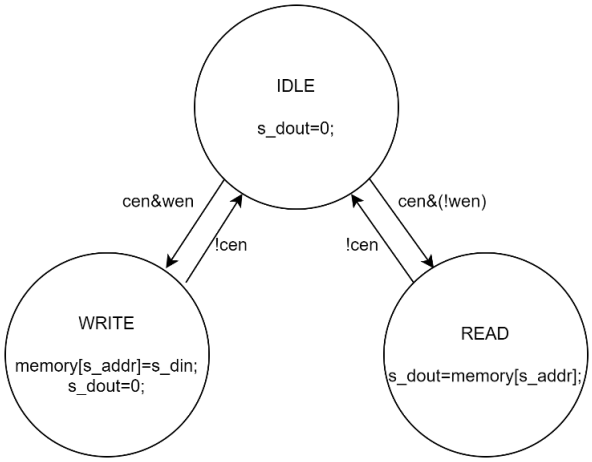
1) pin description

Input	clk	1 bit	clock
	cen	1 bit	Chip enable
	wen	1 bit	Write enable
	s_addr	8 bits	Address
	s_din	64 bits	Data in
Output	s_dout	64 bits	Data out

2) block diagram



3) FSM

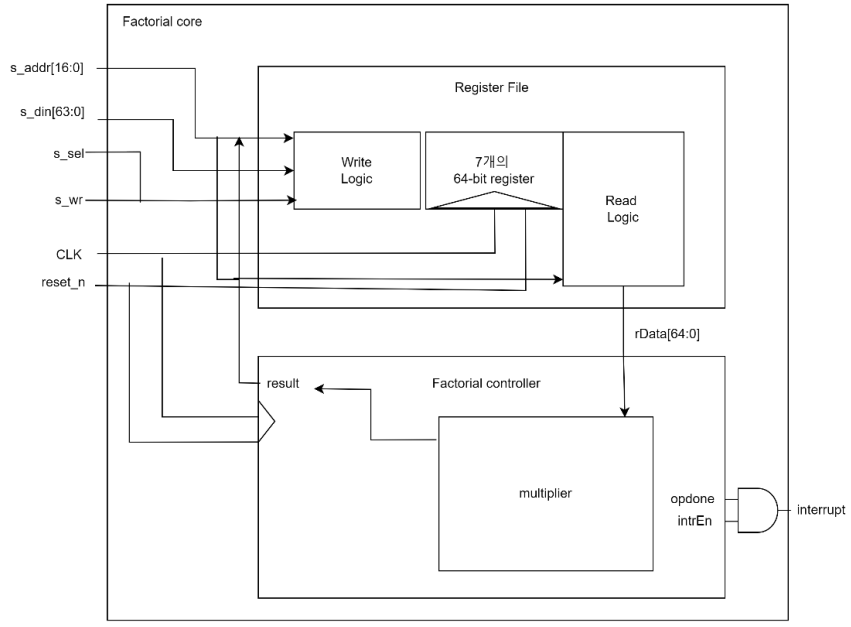


3. Factorial core

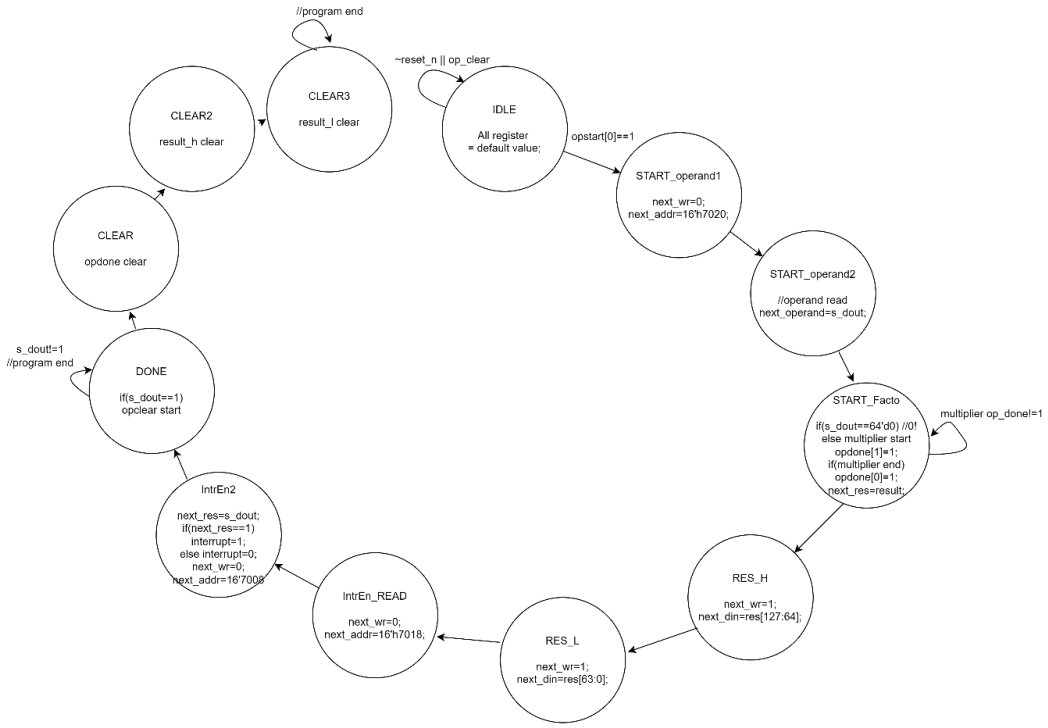
1) pin description

Input	clk	1 bit	clock
	reset_n	1 bit	Active low reset
	s_sel	1 bit	Slave Select
	s_wr	1 bit	Slave Write/read
	s_addr	16 bits	Slave address
	s_din	64 bits	Slave Data input
Output	s_dout	64 bits	Slave Data output
	interrupt	1 bit	Interrupt out

2) block diagram



3) FSM

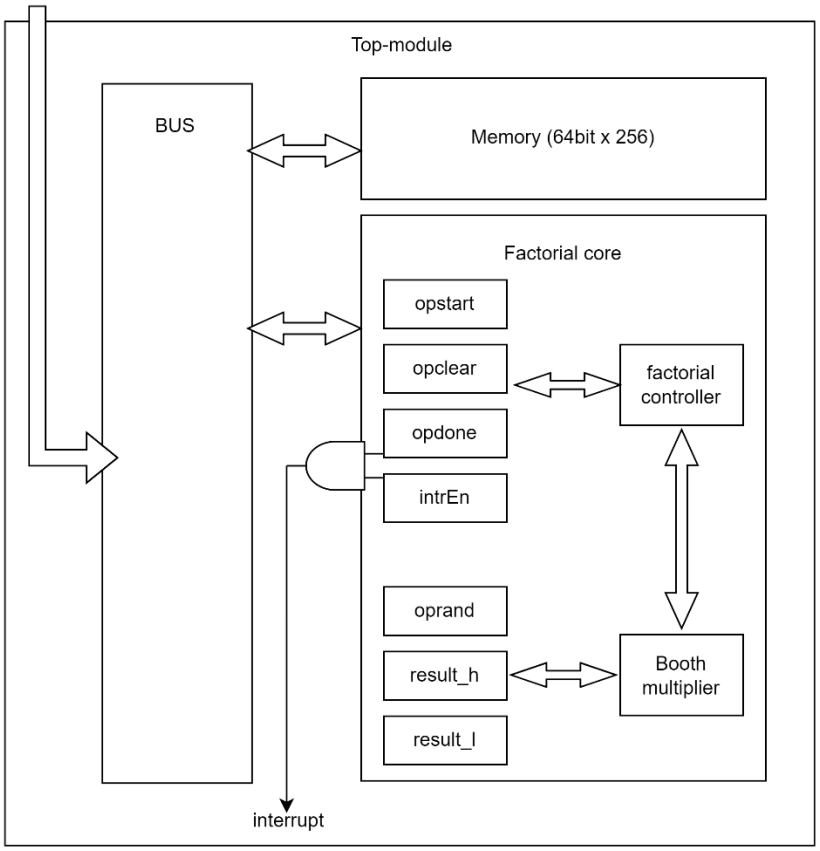


4. TOP

1) pin description

Input	clk	1 bit	clock
	reset_n	1 bit	Active low reset
	m_req	1 bit	Master request
	m_wr	1 bit	Master write/read
	m_addr	16 bits	Master address
	m_dout	64 bits	Master data output
Ouput	m_grant	1 bit	Master grant
	m_din	64 bits	Master data input
	interrupt	1 bit	Interrupt out

2) block diagram



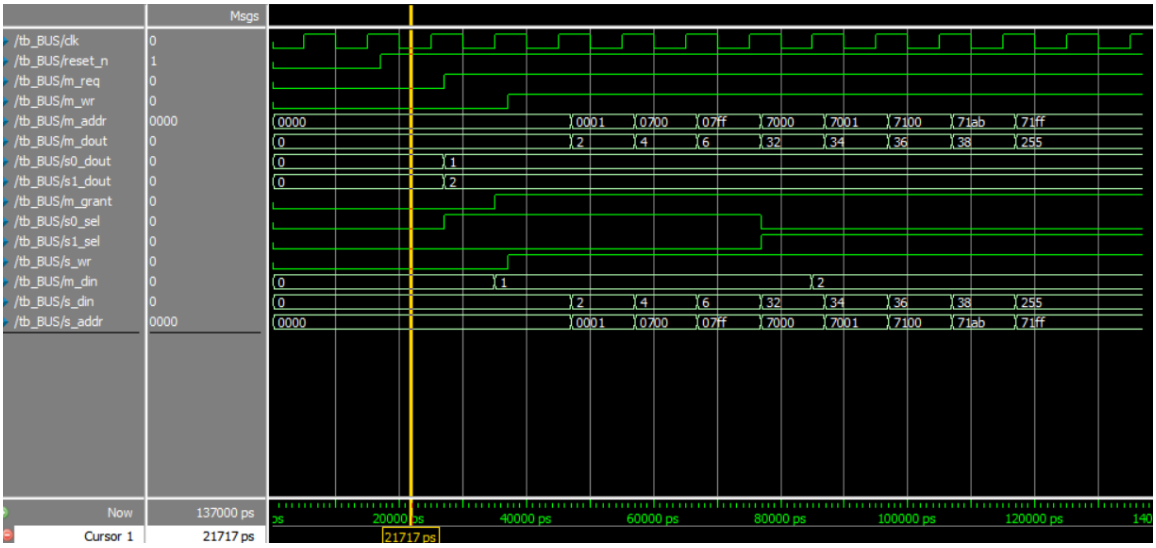
IV. Design Verifiaction Strategy and Results

각 component 별 검증 전략 및 waveform 작성

1. BUS

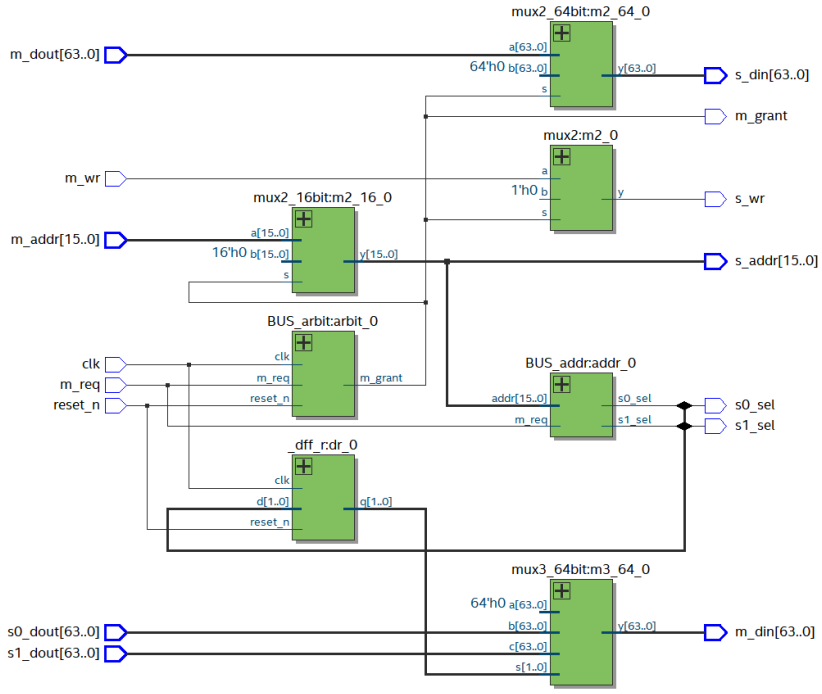
BUS는 m_req가 들어오기 전, 후를 비교하여 주소에 따라 slave가 정상적으로 선택되는지를 중점으로 두어 검증한다. 시뮬레이션 결과는 다음과 같다.

처음에 m_req가 0이었을 때는 어떤 slave도 선택되지 않는다. 이후 m_req가 1로 바뀌면서 m_grant가 1로 나오고, m_addr의 주소에 따라 slave가 선택되는 걸 볼 수 있다. m_addr가 16'h0001 ~ 16'h07FF 까지는 s0_sel이 선택되다가 16'h7000 ~ 16'h71FF 까지는 s1_sel이 선택된다. 또한 s_dout에 값이 나오는 건 s_wr이 1로 된 시점부터이다. m_dout의 값에 따라 s_din이 출력된다. 여러 개의 mater가 있는 것이 아니기 때문에 선택되는 과정은 생략됨을 알 수 있다.



BUS의 합성 결과는 다음과 같다. I/O pin의 크기가 커서 큰 종류의 디바이스를 선택하였다.

-RTL viewer



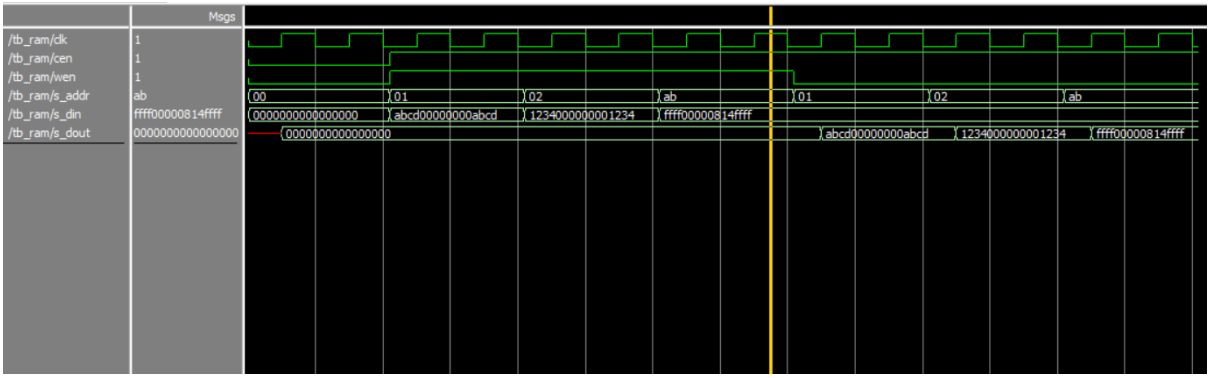
-Flow Summary

Flow Summary	
Flow Status	Successful - Fri Dec 01 08:27:31 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	BUS
Top-level Entity Name	BUS
Family	Cyclone V
Device	5CGXFC9E7F35C8
Timing Models	Final
Logic utilization (in ALMs)	76 / 113,560 (< 1 %)
Total registers	3
Total pins	360 / 616 (58 %)
Total virtual pins	0
Total block memory bits	0 / 12,492,800 (0 %)
Total DSP Blocks	0 / 342 (0 %)
Total HSSI RX PCSs	0 / 12 (0 %)
Total HSSI PMA RX Deserializers	0 / 12 (0 %)
Total HSSI TX PCSs	0 / 12 (0 %)
Total HSSI PMA TX Serializers	0 / 12 (0 %)
Total PLLs	0 / 20 (0 %)
Total DLLs	0 / 4 (0 %)

2. Memory

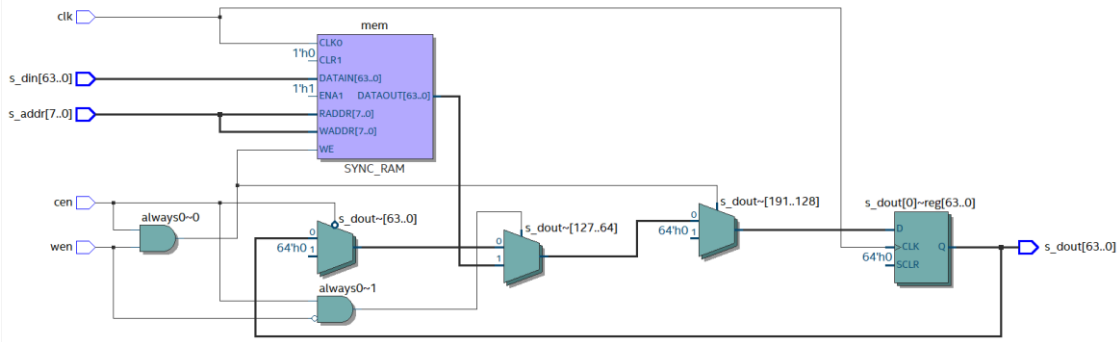
Memory는 cen과 wen 신호에 따라 정상적으로 메모리에 쓰고 읽기가 가능한지를 중점으로 검증한다. 시뮬레이션 결과는 다음과 같다.

clock 신호가 들어간 뒤부터 s_dout이 출력된다. 처음엔 cen이 0이었으니 동작을 하지 않는다. 이후 cen이 1이 되고 wen도 1이 되면 write모드로 전환되어 각 s_addr가 가리키는 메모리에 s_din값을 쓴다. 이때 s_dout은 0값으로 출력된다. wen이 0이 되면 read모드로 전환되면서 해당 주소의 값을 가진 메모리의 데이터를 s_dout으로 출력한다.



Memory의 합성 결과는 다음과 같다. 64bit register를 256개 저장함으로써 16000대의 register가 필요한 것을 볼 수 있다.

-RTL viewer



-Flow summary

Flow Status	Successful - Fri Nov 24 15:19:42 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ram
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	7,838 / 41,910 (19 %)
Total registers	16448
Total pins	139 / 499 (28 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

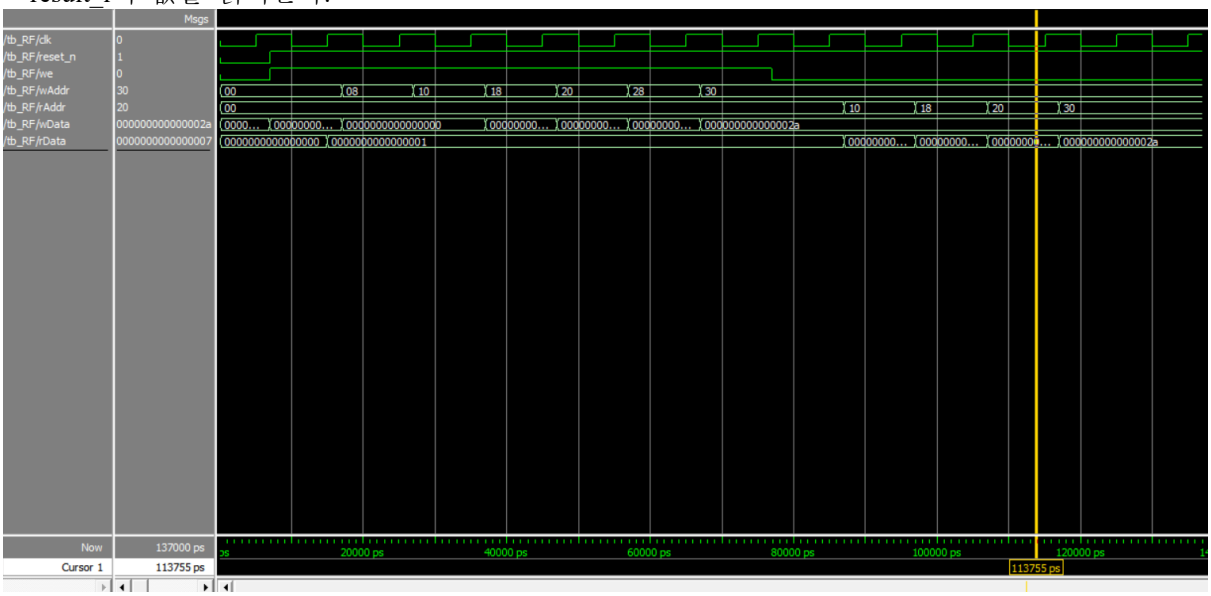
3. Factorial core

Factorial core에서는 Register file과 Factorial Controller 모듈을 새로 구현하여 사용했으므로, 먼저 각각의 모듈들을 검증한 뒤 진행하였다.

1) Register file

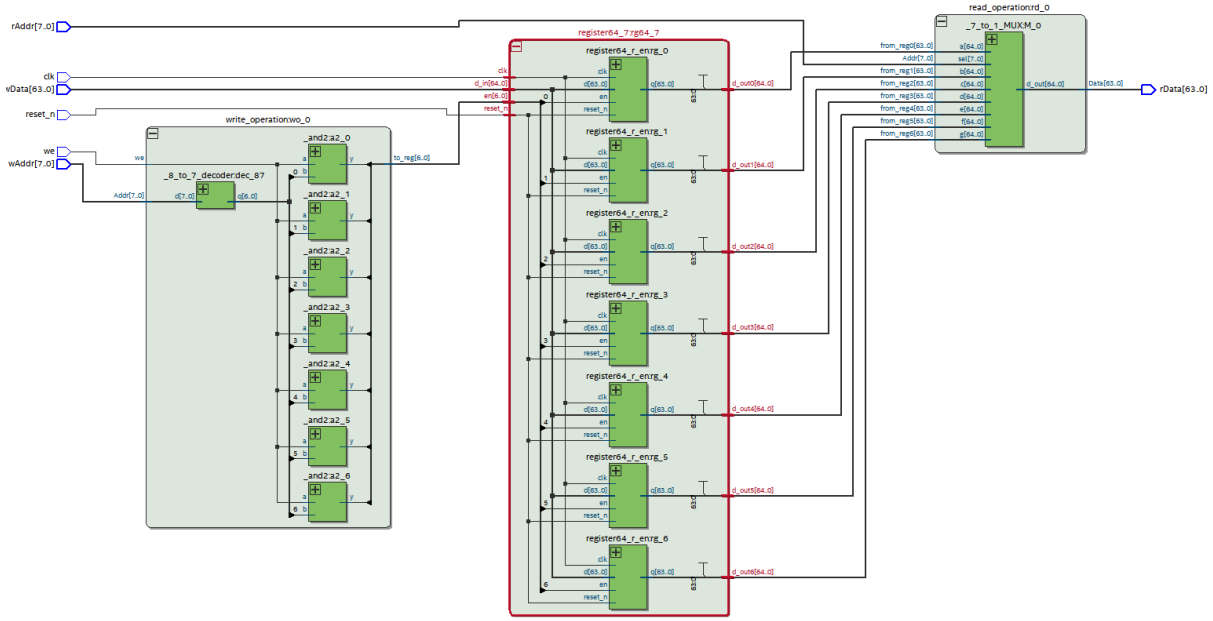
Register file에서는 7개의 64bit register 집합을 구현하여 이를 읽고 쓰는 것이 정상적으로 되는지 검증하였다. 이 register 집합은 프로젝트의 register를 나타낸다. 시뮬레이션 결과는 다음과 같다.

reset_n과 we가 1이 되면 write모드로 전환되어 각 주소에 값을 쓴다. 이때 주소는 Factorial core에서 주소를 받으면 offset을 계산하여 저장하도록 만들었다. 시뮬레이션의 주소가 8'h08, 10, 18, 20, 28, 30으로 되어있는데 이는 각각 opclear, opdone, intrEn, operand, result_h, result_l을 나타낸다. 값들을 저장한 뒤 we가 0이 되어 read모드로 전환되면, 각각 opdone, intrEn, operand, result_l의 값을 읽어온다.



Register file의 합성 결과는 다음과 같다. write, read, register 부분으로 나누어진 것을 볼 수 있다.

-RTL viewer



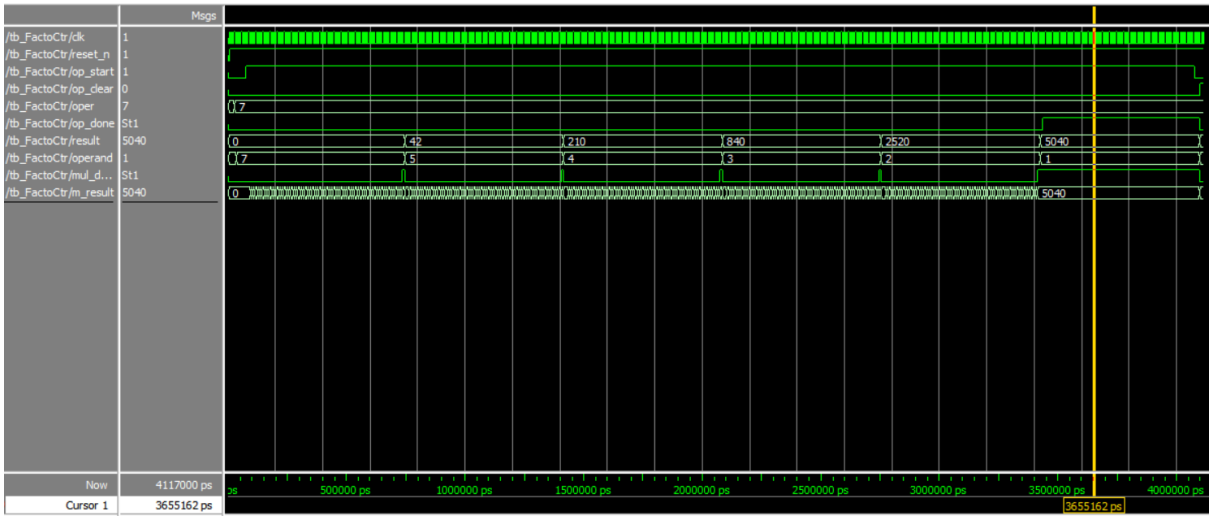
-Flow summary

Flow Status	Successful - Fri Nov 24 16:04:14 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	RF
Top-level Entity Name	RF
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	212 / 41,910 (< 1 %)
Total registers	448
Total pins	147 / 499 (29 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

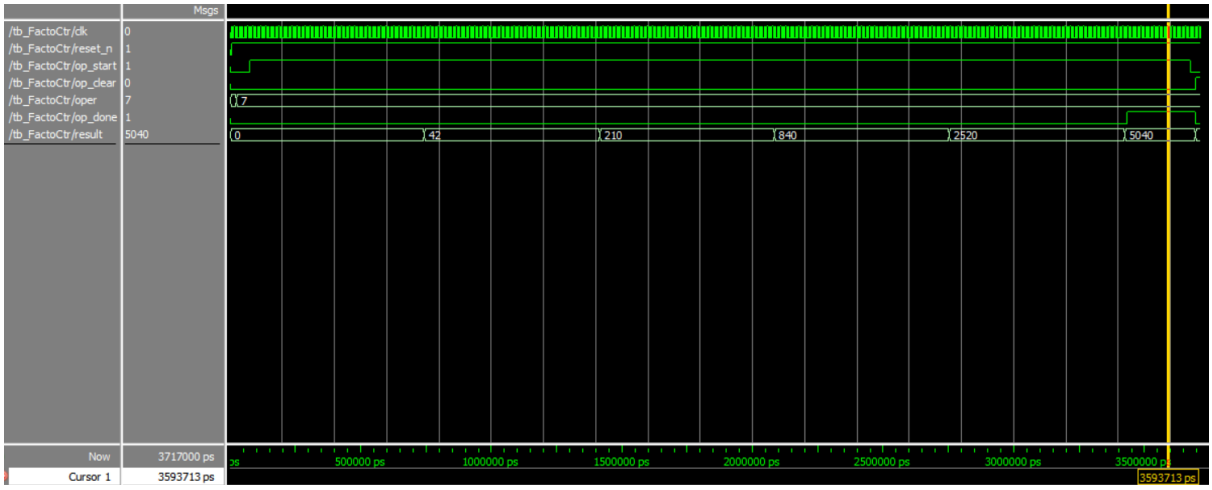
2) Factorial Controller

Factorial controller는 operand의 값에 따라 정상적으로 연산 결과가 나오는지를 중점으로 검증하였다. 시뮬레이션 결과는 다음과 같다.

7!을 계산하기 위해 operand로 7이 들어오면 연산을 시작한다. Radix-2 Booth multiplier를 사용하므로 한번 연산할 때 64cycle이 걸린다. 해당 cycle이 지날 때마다 multiplier의 opdone 신호가 1이 되어 Factorial controller에 전달된다. Factorial controller는 해당 신호를 받으면 multiplier를 clear시키고 operand를 감소시켜 다시 연산을 시작한다. 이러한 과정은 operand가 2일 때까지 계속 반복한다. 최종적으로 연산 결과가 나오고 operand가 1이 되면 Factorial controller는 Factorial core에서 opclear를 보내기 전까지 해당 상태를 유지한다.

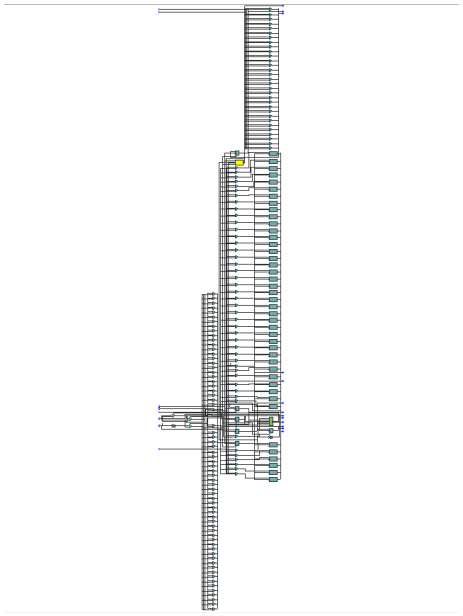


검증을 위해 multiplier의 상태까지 출력한 것은 위와 같으며, 프로젝트에 맞게 입출력을 최소로 조절한 상태의 시뮬레이션은 아래와 같다. 결과는 동일하게 나온다.



Factorial controller의 합성 결과는 다음과 같다. 이 모듈도 pin의 크기 때문에 큰 디바이스를 선택하였다.

-RTL viewer



-Flow summary

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Dec 01 12:45:00 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	FactoCtr
Top-level Entity Name	FactoCtr
Family	Cyclone V
Device	5CGXBC9E6F35C7
Timing Models	Final
Logic utilization (in ALMs)	613 / 113,560 (< 1 %)
Total registers	510
Total pins	197 / 616 (32 %)
Total virtual pins	0
Total block memory bits	0 / 12,492,800 (0 %)
Total DSP Blocks	0 / 342 (0 %)
Total HSSI RX PCSs	0 / 12 (0 %)
Total HSSI PMA RX Deserializers	0 / 12 (0 %)
Total HSSI TX PCSs	0 / 12 (0 %)
Total HSSI PMA TX Serializers	0 / 12 (0 %)
Total PLLs	0 / 20 (0 %)
Total DLLs	0 / 4 (0 %)

V. Conclusion

프로젝트를 진행하면서 전체적으로 여러 시행착오를 겪었다. 우선 BUS나 Factorial Controller 모듈을 실행할 때 I/O pin의 크기가 크다는 에러가 발생하였다. 이에 더 많은 pin을 사용할 수 있는 디바이스로 변경하니 컴파일 시간은 더 걸렸지만 정상적으로 실행이 됐었다.

Factorial controller에서는 multiplier를 instance하여 반복 연산이 제대로 작동하도록 설계하는 것이 문제였는데, 처음에는 operand를 일반적으로 하나씩 감소하고 이를 result와 곱하는 식으로 코드를 작성하였었다. 하지만 결과를 보니 이렇게 하면 제대로 연산이 안되었다. 예를 들어 4!을 계산하기 위해 위와 같은 방식으로 코드를 작성하면 4x3을 진행한 뒤 다시 3x12가 되는 결과가 나왔었다. 처음 operand를 감소하여 이를 result로 취급하고 곱한 뒤에 operand는 2가 감소되어야 연산이 제대로 될 수 있었다. 따라서 sub_operand로 새로운 register를 추가 생성하여 여기에 처음 operand에서 2를 감소한 값을 저장하고 이를 이용하는 방식으로 수정하였다.
이 문제를 해결하니 이번에는 끝나는 지점에서 문제가 나타났는데, opdone이 출력되어도 끝나지 않고 무한 반복 연산이 진행되는 것이었다. 이는 DONE state에서 clear 전까지는 상태가 유지되도록 next register의 값들을 지정하니 해결될 수 있었다.

Factorial core 모듈부터는 제대로 완성하지 못하였다. Register file에서 값들을 차례대로 읽고 써서 그 값들을 다시 Factorial controller에 보내야 되는데 이에 대한 에러들이 많았기 때문이다. 특히 register 변수를 이용하는 데에 어려움이 있었는데, register의 값들을 다시 register file의 변수로써 보낼 때 사용이 불가한 것 같았다. 다른 변수로도 바꾸어 봤지만 잘 해결이 안 되었다. offset을 계산하여 그에 대한 register file의 값들을 이용하는 것에 여러 시행착오를 겪어서 아직 관련 내용에 대한 지식이 부족한 것 같다고 느꼈다. state를 여러 개 만들어서 순서대로 하나씩 전달하는 방법으로 구현해봤으나 그렇게 하니 모델심을 실행할 때 code 7 에러가 뜨면서 화면이 나타나지 않았다. 이는 다른 문제로 해결 방안을 찾아봐야 할 것 같다. 따라서 Factorial core를 구현하지 못하여 TOP 모듈까지 미완성인 상태이므로, 이에 대한 공부를 더 해봐야 할 것 같다.

VI. Reference

[1] D. M. Harris and S. L. Harris, Digital design and computer architecture, Elsevier, 2007
[2] Booth. Andrew Donald, A signed Binary Multiplication Technique, Oxford University Press, 2018