Factorial computation system

2022202064 최봉규

Abstract

디지털 논리 term project를 최종 보고서 작성을 위한 기본 template format입니다. Abstract는 해당 보고서의 내용을 전체적으로 요약하는 부분입니다. 이 뒤로 introduction, project specification, design details, design verification strategy and results에 대하여 각각의 해당하는 session에 작성하여 주시기 바랍니다. 마지막에 references는 자기가 참조한 서적, 논문 등에 대하여 정리하는 것입니다. References의 내용을 보고서에 적을 때는 예를 들어 'multiplier는 ~~~하는 장치이다. [1]' 이런 식으로 적어주시면 됩니다.

I. Introduction

이 프로젝트는 Factorial core, Bus, Memory로 구성되고 testbench를 이용하여 시스템의 동작을 제어할 수 있도록 한다. Factorial core는 Bus를 통해 접근할 수 있는 register의 집합으로 opstart, opclear, opdone, intrEn, operand, result_h, result_l을 가지고 있다. Factorial core는 이 register 집합을 통해 외부 모듈과 데이터를 주고받는다. System이 시작하면 testbench는 Bus를 통해 Factorial core에 접근하여 register read/write를 수행한다. 예를 들어 연산을 시작하기 위해서는 testbench가 operand에 피연산자를 opstart[0]에 1을 순서대로 write해야 한다. 만약 testbench가 interrupt 사용을 원하면 opstart[0]에 1을 write하기 전 intrEn[0]을 1로 write해야 한다. 연산 종료 후에는 opclear[0]에 1을 write하여 Factorial core를 초기화한다. Memory 역시 bus를 통해 접근할 수 있으며, 각각의 slave들은 address region을 갖는다. 해당 범위 내의 address를 받을 경우에 slave가 가능하다는 신호를 받으며 동작할 수 있다.

프로젝트를 진행하기 위한 계획은 다음과 같다. 제안서를 완성하여 설계를 진행한 뒤 Verilog로 그에 대한 설계를 진행한다. 설계를 어느 정도 진행한 후에는 검증과 병행하여 프로젝트를 진행한다. 마지막 주차에서는 전체 top에 대한 검증과 함께 보고서를 작성한다. 아래는 schedule table이다.

항목 / 차수	11	12	13	14
제안서				
코드 작성				
코드 검증				
결과 보고서				

II. Project Specification

각 component 별 작동원리 기재 (ex) DMAC 작동원리, Multiplier 알고리즘 등…

1. Factorial Core

Factorial core는 Bus와 연결된 slave component로 bus와 연결된 master는 Factorial core의 slave interface인 s_sel, s_wr, s_addr, s_din을 통하여 Factorial core의 내부 동작을 제어한다.

Master는 slave interface를 통하여 Factorial core에 포함된 register에 미리 정의된 값을 사용하여 제어한다. Factorial core의 slave interface는 s_addr를 이용해 offset을 계산한다. Offset과 s_wr를 이용해 register에 s_din 데이터를 write하거나 register의 값을 s_dout으로 내보낸다. S_wr이 0일 경우 register read를, s_wr이 1일 경우 register write를 한다.

Factorial 연산을 시작하게 되면 factorial controller를 이용하여 multiplier를 제어한다. Multiplier에 값을 clear신호를 계속 주면서 초기화를 진행하다가 operand의 값이 쓰이게 되면 그제 state를 움직이면서 multiplier에 start 신호를 준다. Multiplier가 모든 연산을 종료하고 결과값을 주면 next_result_l과 next_result_h에 분배한다. 이때 하위 64bits에 대해 64'b0의 값을 갖게 되면 next_result_l은 상위 64bits의 값을 받도록 한다. Controller는 factorial 연산을 위해 multiplier에 clear 신호를 주고, next_operand의 1을 감소한 상태로 전달해준다. Next_operand의 값이 0이 되면 factorial의 연산을 종료하고, opdone에 종료 상태임을 알려주도록 한다.

- Multiplier 알고리즘

Multiplier는 multiplicand(피승수)와 multiplier(승수)를 곱하여 결과값을 도출하는 hardware이다. Multiplicand와 multiplier의 각각의 bit length는 64bits이며, 곱의 결과값은 그의 두 배인 128bits로 나온다.

구현한 multiplier는 radix-4로 3개의 bit에 대해서 검사를 하여 어떤 식으로 연산할 것인지 결정한다. 승수 A에 대해 단지 2bit shift할지, A를 더하거나 빼는 연산을 할지, 2A에 대하여 더하거나 빼는 연산을 진행한다. 다음은 radix-4의 경우에 booth's multiplier의 규칙이다.

x_i	x_{i-1}	x_{i-2}	Operation	y_i	y_{i-1}	y
0	0	0	0 + 0 = 0	0	0	0
0	0	1	0 + A = A	0	1	+1
0	1	0	2A - A = A	0	1	+1
0	1	1	2A + 0 = 2A	1	0	+2
1	0	0	-2A + 0 = -2A	-1	0	-2
1	0	0	-2A + A = -A	0	-1	-1
1	1	0	0 – A = -A	0	-1	-1
1	1	1	0 + 0 = 0	0	0	0

2. Memory

Ram은 임의의 address에 대해 data를 write / read할 수 있다. 8bits의 address를 받아 해당 address에 대해 참조한다. 이 때, ram slave에 대해서 사용할 수 있다는 cen 신호를 받아 수행할 수 있다. Memory는 256개의 data를 address에 기반하여 해당 address 위치에서 write / read를 한다. 이 때 data의 bit는 64bits이다. Clk에 synchronous하게 움직이고, cen input port는 해당 module 즉, ram을 사용할지 허가를 받는 것이다. Cen을 받지 못하면 addr, din값에 상관없이 dout 은 64'b0의 값을 출력한다. Cen이 1인 경우 메모리를 사용하는데, wen 값에 따라 write / read를 결정한다. 값이 1인 경우 write, 0인 경우 read를 수행한다.

3. Bus

Bus는 여러 component 들 간에 data를 전송할 수 있도록 연결해주는 component이다. Bus는 새로운 component들을 추가하기가 쉬우며, 가격이 저렴한 특징을 가지고 있다. Bus는 1개의 master interface와 2개의 slave interface를 가지고 있다. Address는 16bits이고, data는 64bits이다. Slave 0 (Memory)는 0x0000 ~ 0x07FF 사이의 address를 memory map region으로 갖는다. Slave 1(Factorial core)은 0x7000 ~ 0x71FF 사이의 address를 memory map region으로 갖는다. Slave 0과 slave 1 address area이외의 주소에 접근할 경우 s0_sel과 s1_sel을 0으로 바꾼다. 해당 역할은 bus의 address decoder에서 진행하게 된다. 하여 벗어난 주소에 대해 어떠한 slave device가 선택되지 않도록 한다.

4. Top

Top 모듈은 BUS, factorial core와 memory를 instance 하여 연결한 모듈이다. Top 모듈의 input port를 이용해 Top 모듈 내에 있는 BUS의 master port에 접근을 할 수 있다. Top 은 memory mapped I/O 방식을 사용하여 외부(testbench)에서 주소를 통해 Top 모듈의 slave(Factorial core, Memory) device에 접근할 수 있다. 또한 Top은 내부의 Factorial core에 직결된 interrupt를 통해 core의 동작 상태를 testbench에 전달할 수 있다. 다음은 두 slave의 map region이다.

Address region	Component
0x0000 ~ 0x07FF (0b 0000_0000_0000_0000 ~ 0b 0000_0111_1111_1111)	Memory
0x7000 ~ 0x71FF (0b 0111_0000_0000_0000 ~ 0b 0111_0001_1111_1111)	Factorial core

III. Design Details

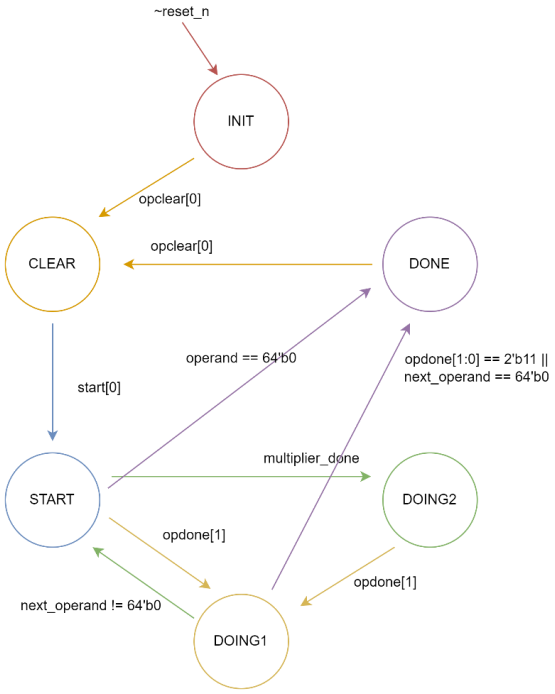
각 component 별 pin description, block diagram, FSM 기재

1. Factorial core

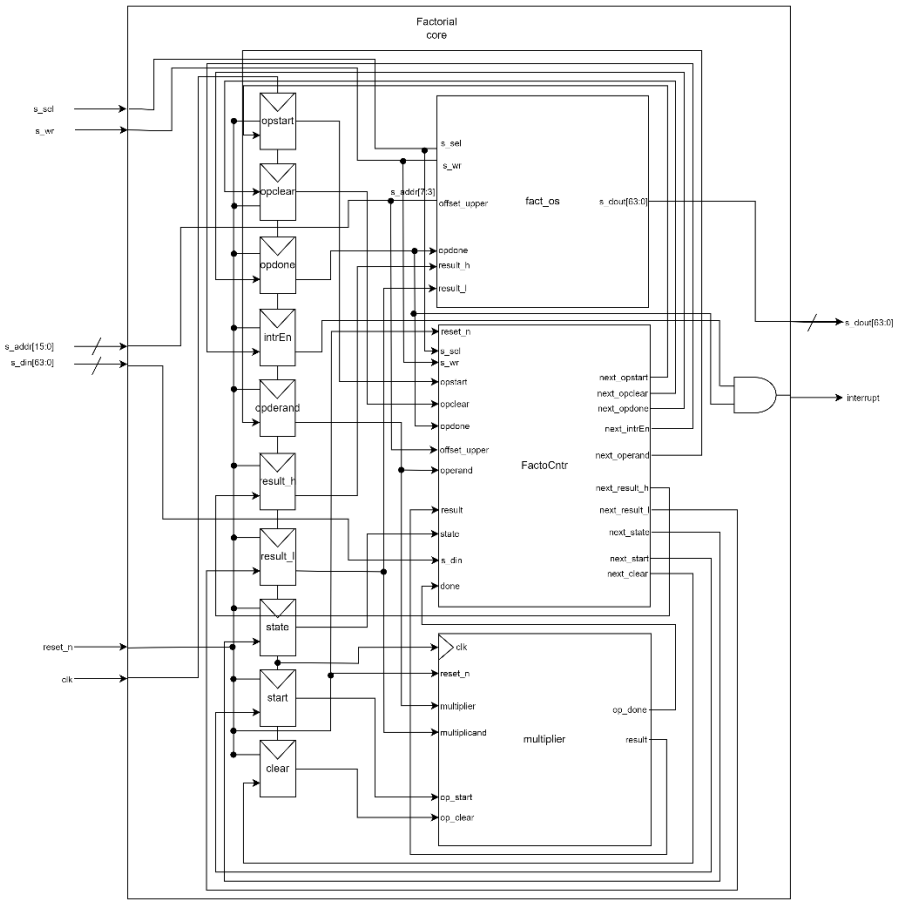
Pin description

Direction	Port name	Bit width	Description
Input	Clk	1	Clock
	Reset_n	1	Active low reset
	S_sel	1	(slave interface) select
	S_wr	1	(slave interface) write/read
	S_addr	16	(slave interface) address
	S_din	64	(slave interface) data input
Output	S_dout	64	(slave interface) data output
	Interrupt	1	Interrupt out

FSM



Block diagram

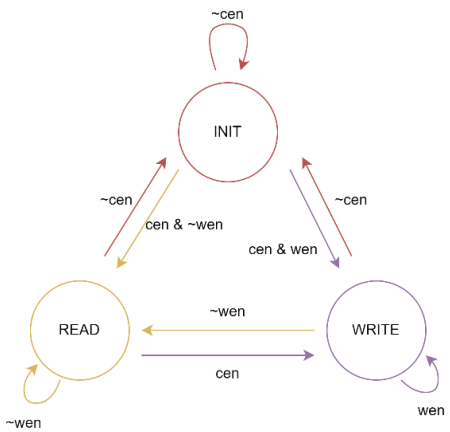


2. Memory

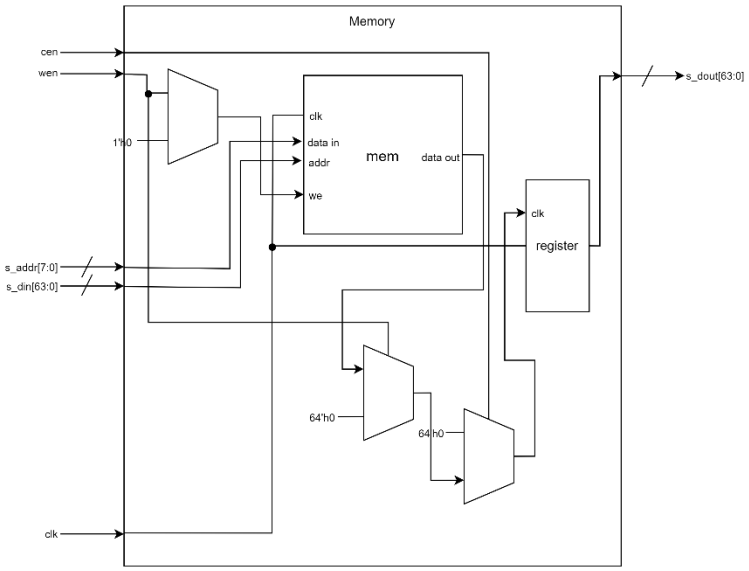
Pin description

Direction	Port name	Bit width	Description
Input	clk	1	Clock
	cen	1	Chip enable
	wen	1	Write enable
	s_addr	8	Address
	s_din	64	Data int
Output	S_dout	64	Data out

FSM



Block diagram

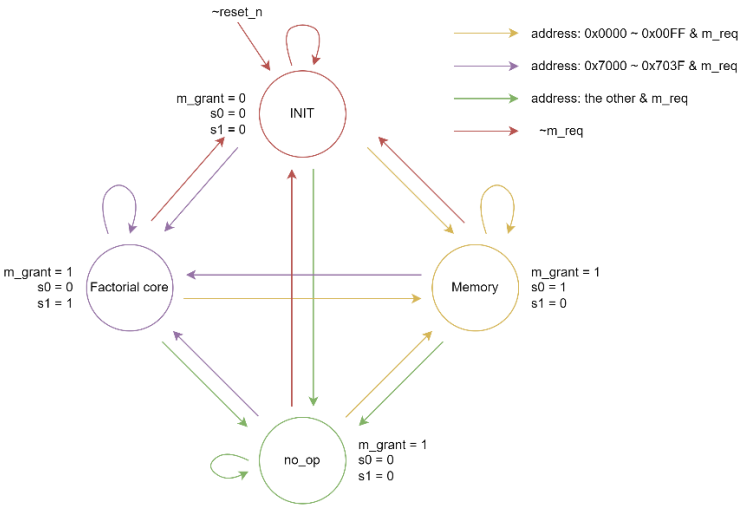


3. Bus

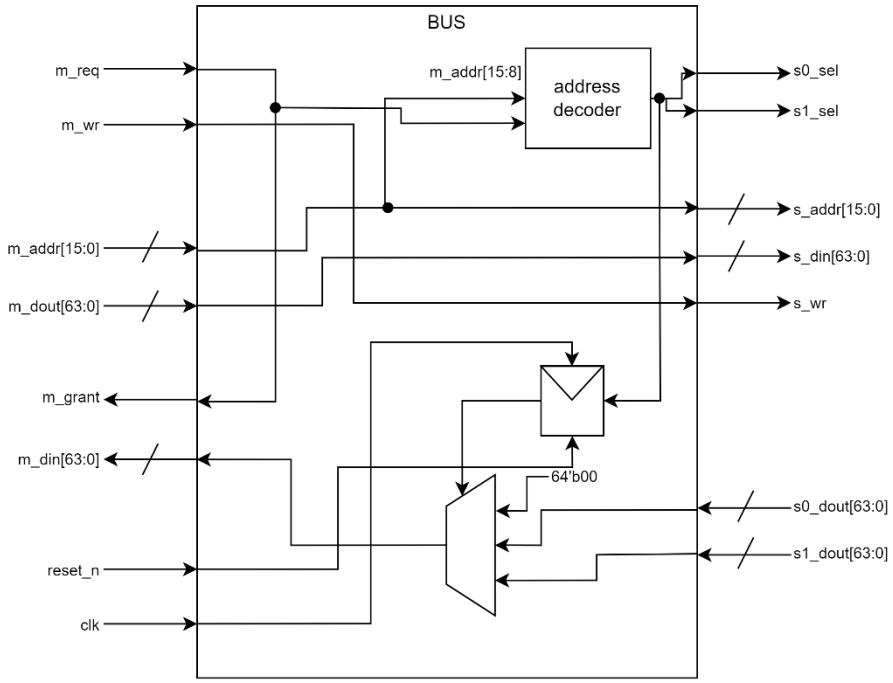
Pin description

Direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset
	m_req	1	Master request
	m_wr	1	Master write / read
	m_addr	16	Master address
	m_dout	64	Master data output
	s0_dout	64	Slave 0 data out
	s1_dout	64	Slave 1 data out
Output	m_grant	1	Master grant
	m_din	64	Master data input
	s0_sel	1	Slave 0 select
	s1_sel	1	Slave 1 select
	s_addr	16	Slave 1 select
	s_wr	1	Slave write / read
	s_din	64	Slave data input

FSM



Block diagram

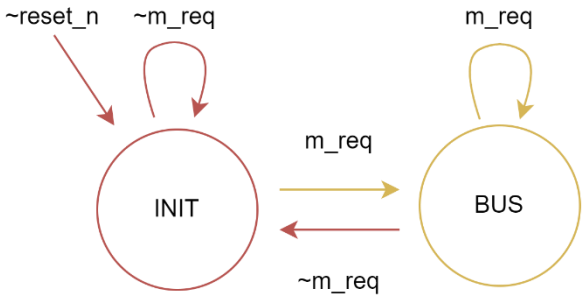


4. Top

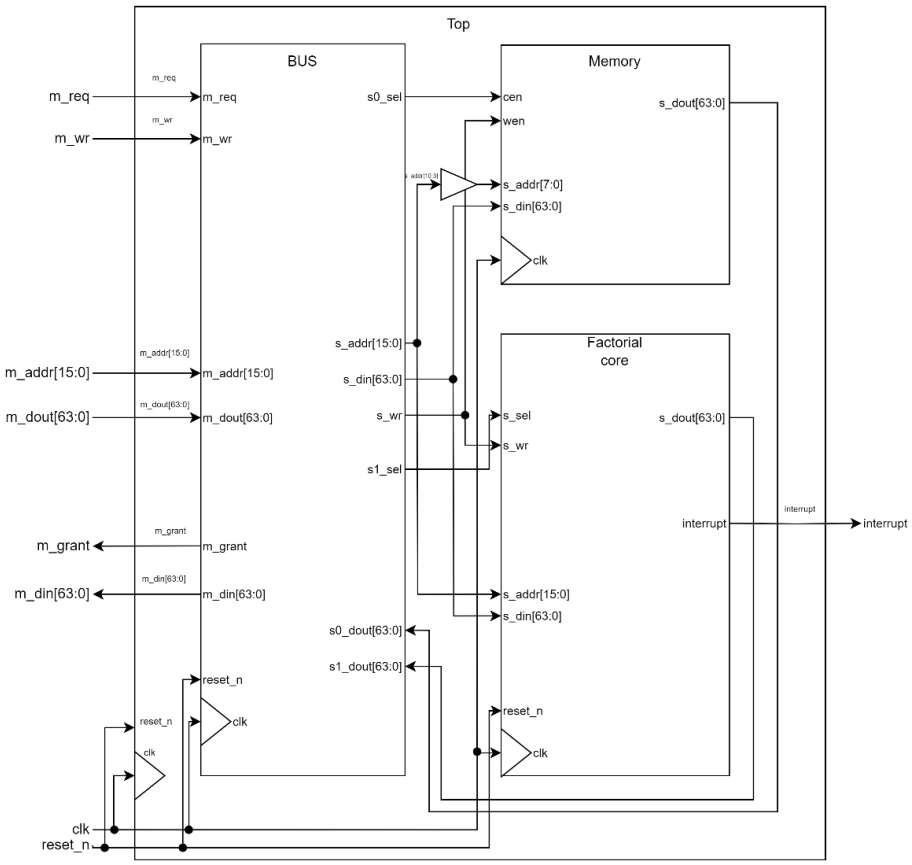
Pin description

Direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset
	m_req	1	Master request
	m_wr	1	Master write / read
	m_addr	16	Master address
	m_dout	64	Master data output
Output	m_grant	1	Master grant
	m_din	64	Master data in
	Interrupt	1	Factorial core interrupt

FSM



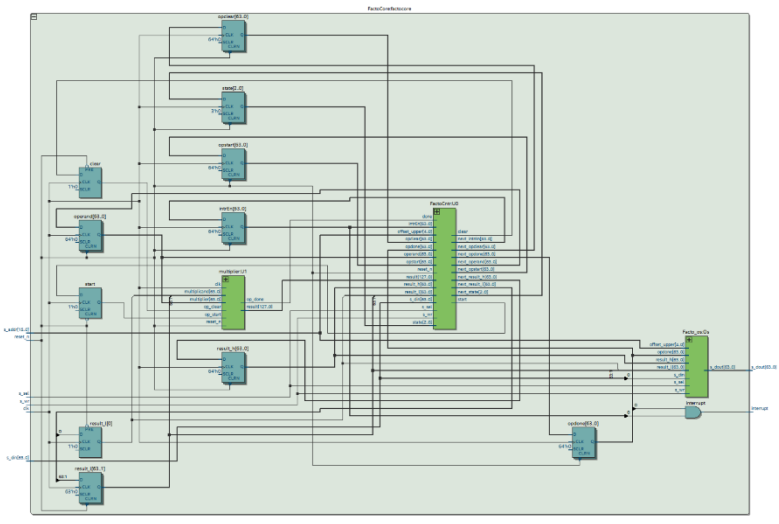
Block diagram



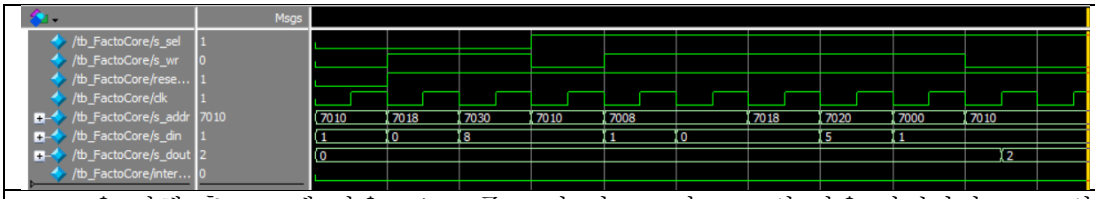
IV. Design Verification Strategy and Results

각 component 별 검증 전략 및 waveform 작성

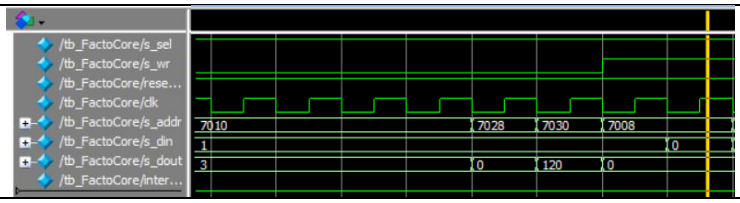
1. Factorial core



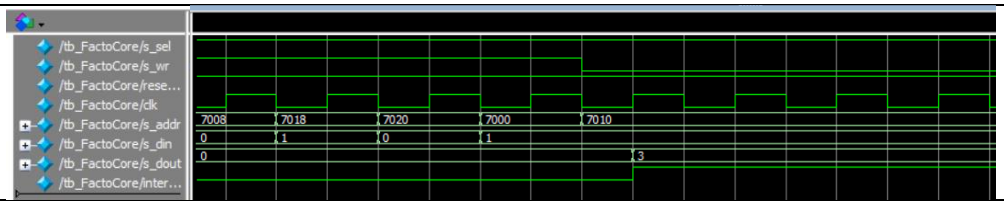
해당 이미지는 factorial core의 RTL map viewer로 여러 개의 resettable register와 multiplier 1개와 output logic, factorial controller로 이루어진 것을 확인할 수 있다 .



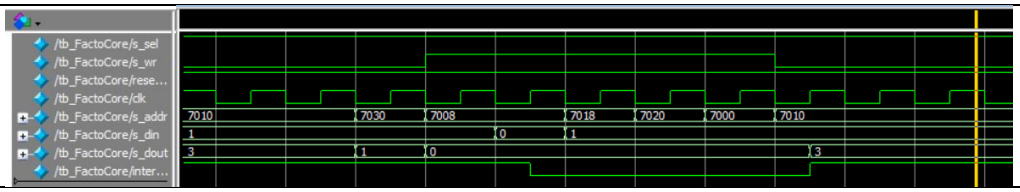
Reset을 진행 후 s_sel에 값을 0으로 주고 난 뒤, addr과 s_din의 값을 바꿔가며 s_dout의 값이 변화가 있는지 관찰 후, clear를 시켜주고 있다. 이후 7020 즉, operand 주소에 5를 입력 후 start 신호를 주었다. Start 신호를 준 이후 opdone의 값을 관찰하면서 64'b3의 값이 되는 것을 기다린다.



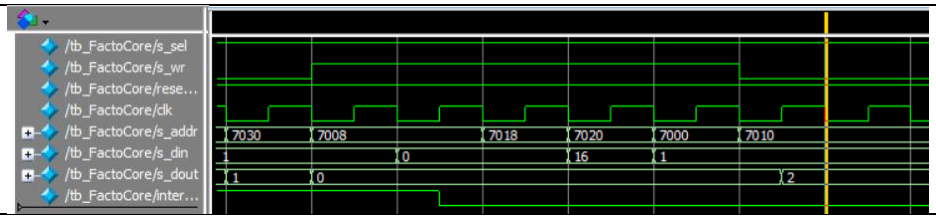
3이 된 이후 result_h와 result_l값을 확인한다. Result_h는 64'h00, result_l은 64'd120으로 5!에 대한 값이 제대로 저장되었음을 확인할 수 있다. 이후 clear 신호를 주는 모습을 확인할 수 있다.



0!에 대해 확인하는 모습으로 intrEn에 1을 주고 나서 operand에는 0의 값을 주었다. 이후 start 신호를 주고, opdone의 값을 확인한 결과 바로 2'b11로 가면서 0!에 대한 예외처리가 잘 이루어졌음을 확인할 수 있다.

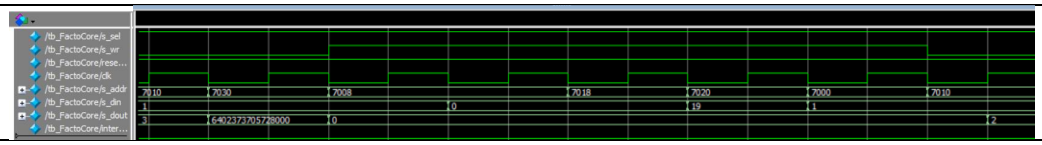


1!에 대해서도 확인하는데, 해당 케이스도 intrEn에 1의 값을 주고 난 뒤 opdone의 값을 관찰하였다. 0!과 같이 바로 연산이 종료됨을 알려주는 것을 확인할 수 있다. 해당 케이스를 이용하여 factorial은 2까지만 곱하고 1을 곱하는 과정은 예외처리로 빠졌음을 설명할 수 있다.

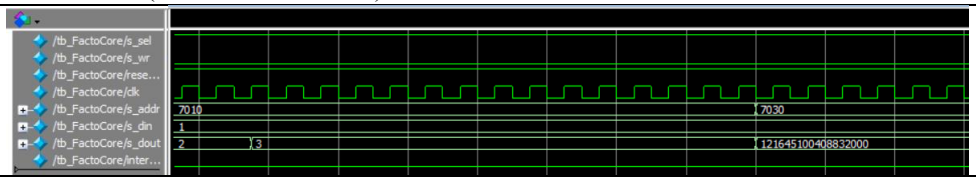


해당 케이스는 16!에 대한 값을 확인하기 위해 살펴본 것으로 1!에 대한 값을 clear를 한 후 operand에 16을 적어 주었다. 이후 opstart 신호에 1을 주면서 opdone을 보니 64'b10의 값을 갖게 되는 것을 확인할 수 있다.

연산이 종료된 이후 result_l에 값을 확인하고 나니 16!에 대한 값이 들어간 것을 확인할 수 있고 이후 clear신호를 준 뒤 operand에 18을 주면서 18!(6402373705728000)에 대한 값을 확인하려 한다.

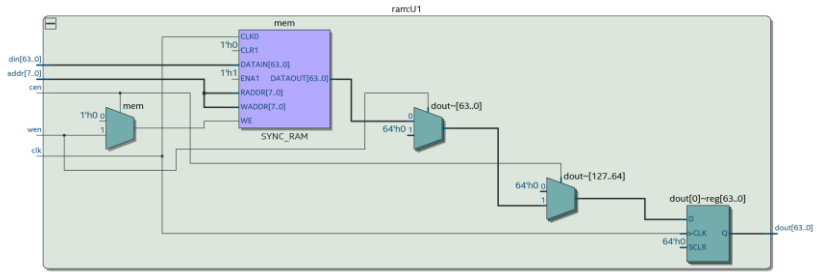


Opdone에 64'b11이 들어오면서 result_1를 확인하니 18!(6402373705728000)에 대한 계산이 제대로 이루어진 것을 확인할 수 있었다. 다시 clear신호를 준 뒤 operand에 19를 주면서 19!(6402373705728000)에 대한 결과 값을 확인하려 한다.

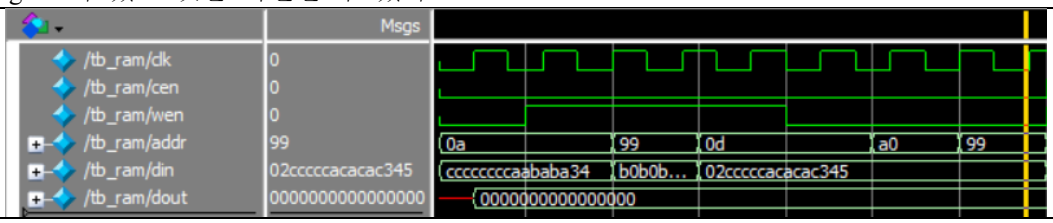


Opdone에 64'b11이 들어오면서 result_1를 확인하니 19!(6402373705728000)에 대한 계산이 제대로 이루어진 것을 확인할 수 있었다.

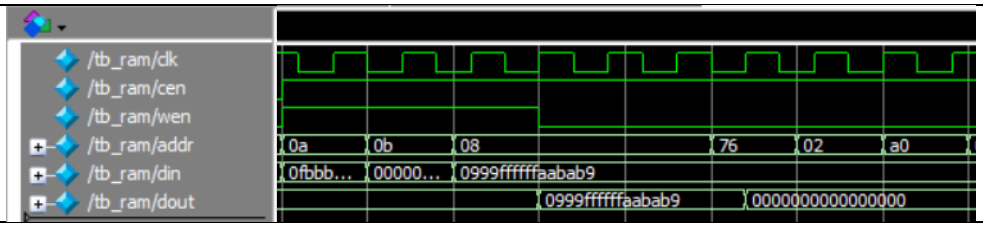
2. Memory



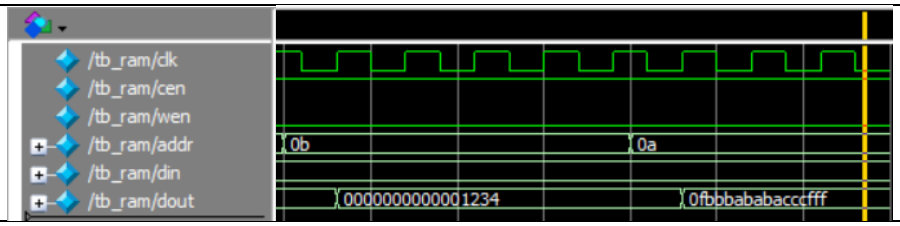
해당 이미지는 ram의 RTL map viewer이다. Mux가 3개 synchronous ram 1개, 64bits register가 있는 것을 확인할 수 있다.



Cen = 0일 때 wen과 addr에 상관없이 dout 값이 모두 64'h00이 나오는 것을 확인할 수 있다.

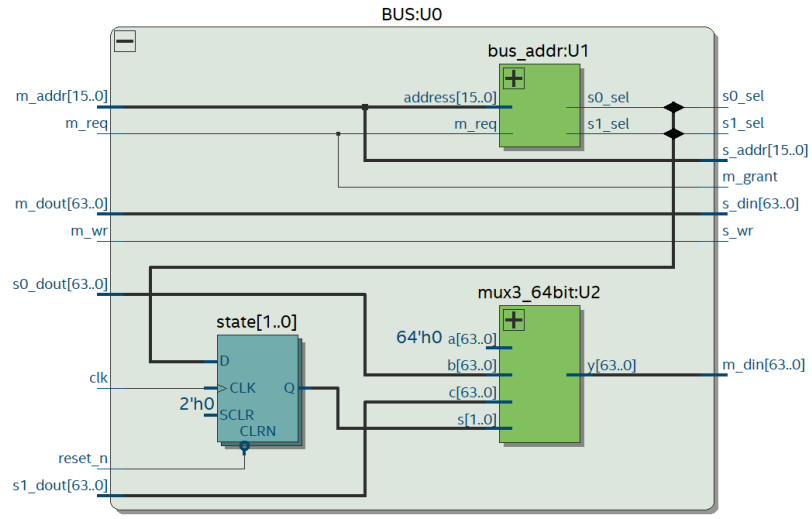


Cen = 1인 경우 ram을 사용하겠다는 신호인데 해당 값이 1인 경우 메모리를 사용하는데, 앞에 있는 0a, 0b, 08에 값을 쓰고 있고 뒤에 나오는 08, 76, 02, a0에 있는 값들을 불러오는 것으로 08에는 이미 값을 썼기 때문에, 앞에 값이 나오는 것을 확인할 수 있고 나머지 주소에 대해서는 write를 한 적이 없기 때문에 64'h00값이 나오는 것을 확인할 수 있다.

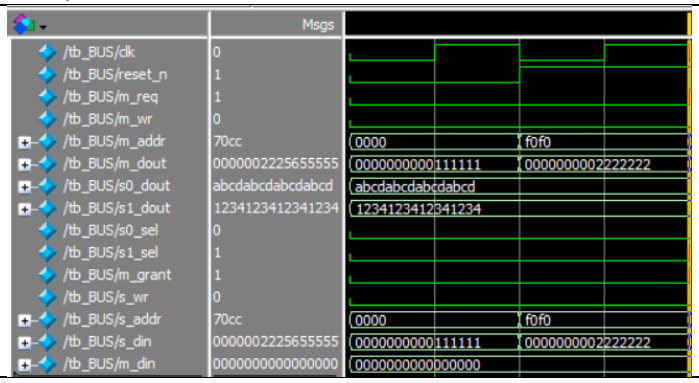


이후 주소가 0b, 0a인 경우 write 했던 것을 read했을 때 저장했던 것들이 잘 나오는지 확인하는 것으로 이전에 넣은 값들이 잘 나오는 것을 확인할 수 있다.

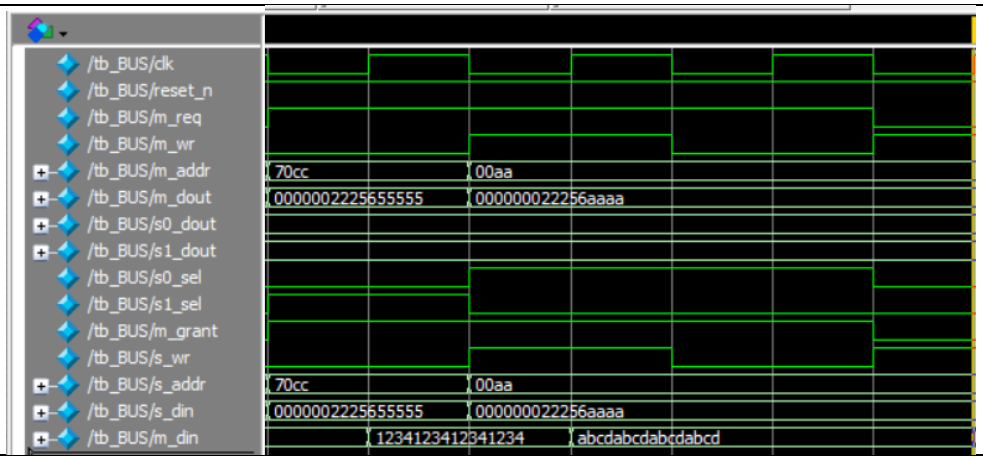
3. Bus



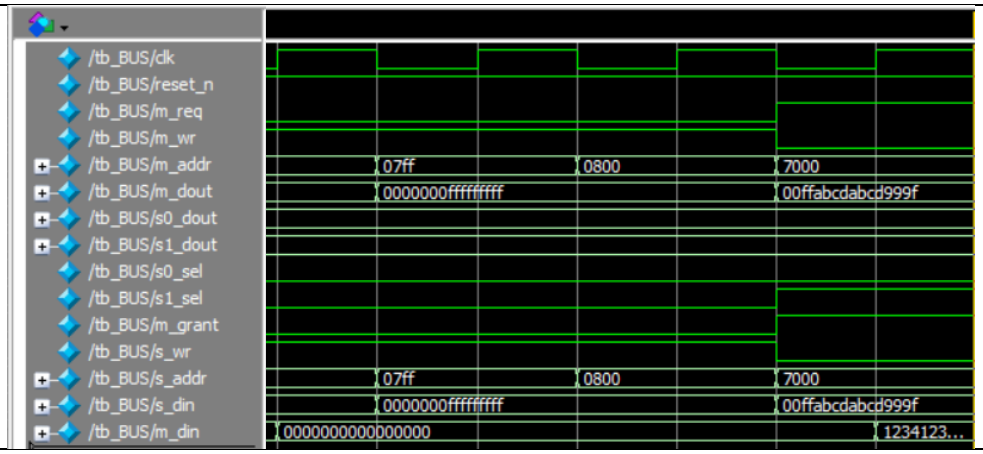
해당 이미지는 BUS module의 RTL map viewer이다. 그림에서 확인할 수 있듯이 state flip flop, 64bits mux3, bus address decoder로 이루어진 것을 확인할 수 있다.



주소가 0000일 때 m_req = 0일 때, s0_sel, s1_sel 둘 모두 선택되지 않도록 된 것을 확인할 수 있다. 또한 아무도 선택되지 않았으므로, m_din이 64'h00인 것을 확인할 수 있다. 이후 f0f0은 s0와 s1이 지원하는 주소가 아니므로 m_din은 또한 64'h00이 나오는 것을 확인할 수 있다. 이와는 별개로 slave에 전달되는 address나 s_din, s_wr은 master에 있는 것이 그대로 전달된 것을 확인할 수 있다.

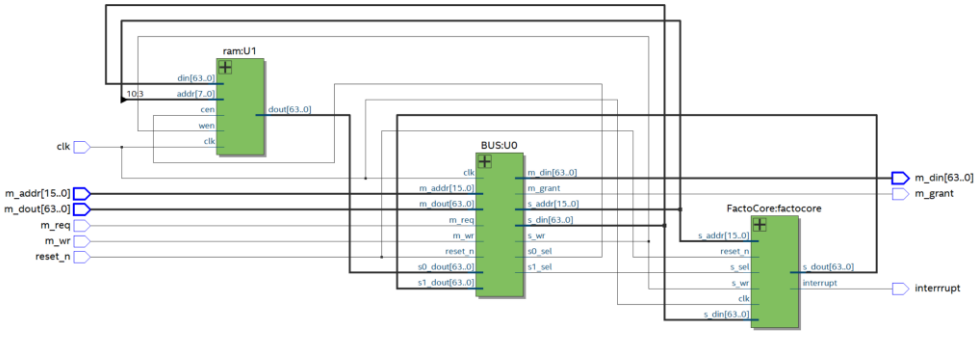


m_req에 대해 1의 값을 받고 진행된 모습이다. Slave로 master가 준 값에 대해 잘 전달된 것을 확인할 수 있다. Address에 따라 m_din값이 바뀐 것을 확인할 수 있는데, 70cc인 경우 s1_dout으로 전달한 값이, 00aa인 경우 s0_dout으로 전달한 값으로 갖는 것을 확인할 수 있다.

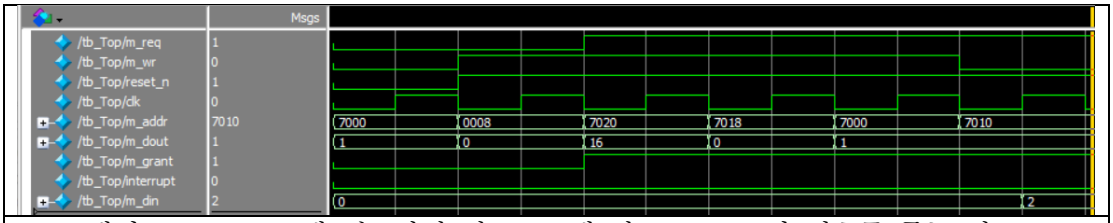


이후 m_req에 대한 값을 0으로 주면서 m_din은 다시 64'h00의 값을 갖고 있으며, m_req = 1이 됐을 때, m_addr = 7000으로 s1_dout 값을 m_din으로 가는 것을 확인할 수 있다.

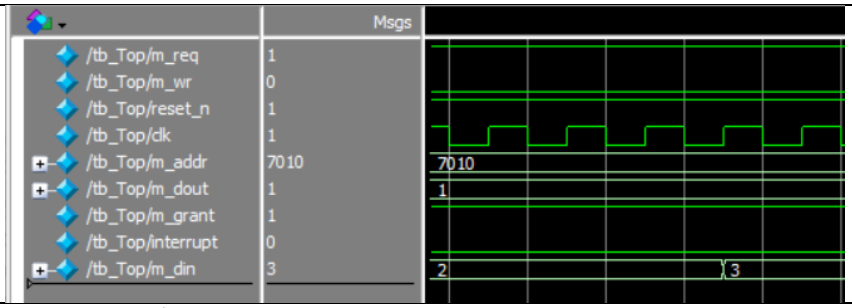
4. Top



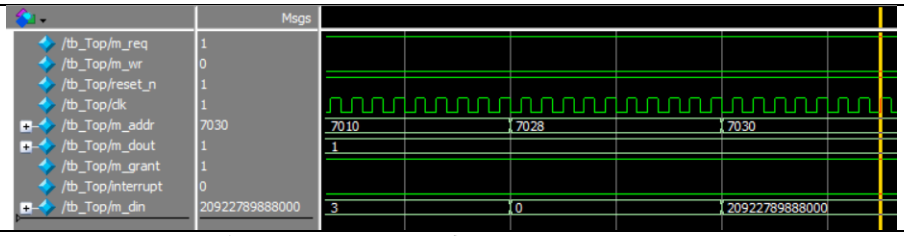
해당 이미지는 Top module의 RTL map viewer이다. 그림에서 확인할 수 있듯이 ram, bus, factocore로 이루어진 것을 확인할 수 있다.



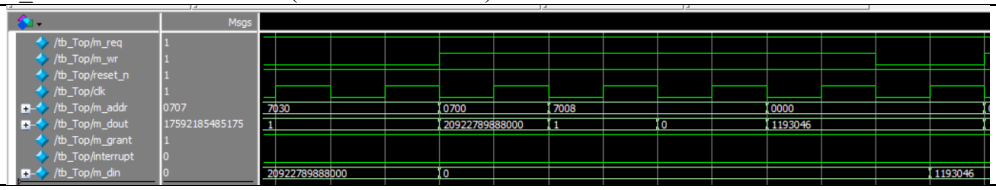
Top에서 Factorial core에 접근하기 전 m_req에 따른 m_grant가 신호를 주는 지 확인하는 과정이다. 이를 확인한 이후 operand에 16의 값을 주고, intrEn에 0의 값을 주고, opstart에 신호 1을 준 상태이다. 이후 opdone을 관찰하는데 진행중이므로 2'b10인 10진수로 2가 출력되는 과정을 확인할 수 있다.



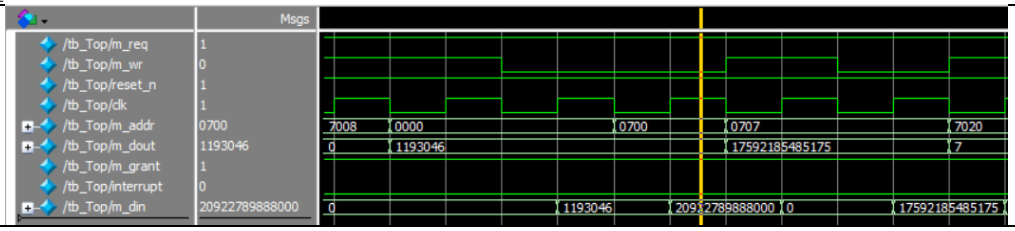
이후 연산이 진행된 후 opdone의 address에서 3의 값이 써지는 것을 확인할 수 있다. 해당 의미는 2'b11로 factorial core에서의 연산이 종료됨을 알려주는 신호이다.



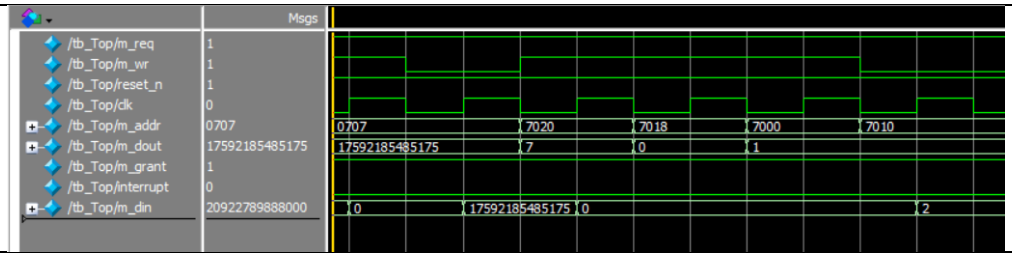
Opdone에 신호가 올라온 것을 확인한 후 result_h와 result_l에 대한 값을 확인한 것으로 16! 의 값은 64bits 내로 결과 저장이 가능하여 result_h에는 0이 들어가 있고, result_l에는 16! 에 대한 값(20922789888000)이 들어 가 있음을 확인할 수 있다.



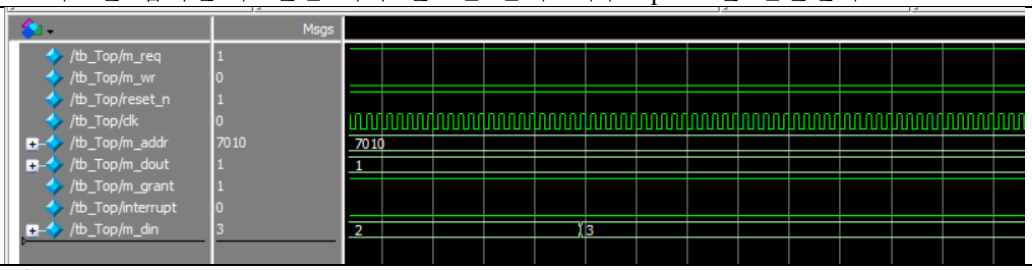
이후 메모리의 기능을 확인하기 위해 해당 결과를 받아 0700 주소에 저장을 하고 난 후 factorial core를 클리어를 해주었다. 0000번지에도 값을 쓰고 m_wr에 0신호를 주면서 해당 값이 저장됐는지 m_din을 통해 확인할 수 있다. 그리고 m_wr = 1인 과정 중에는 m_din의 값이 0으로 나오는 것을 확인할 수 있다.



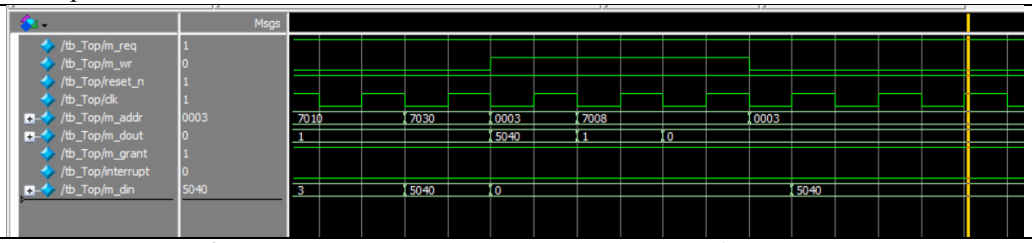
Factorial 연산을 저장한 주소에서 read 기능을 이용하니 저장된 것을 다시 읽을 수 있는 것을 확인할 수 있다. 이후 메모리가 되는지 다시 한번 0707로 메모리 주소를 바꾸어 입력 값을 넣으니, 넣을 때에는 0의 값이 출력, 넣고 나서 read를 하니 넣은 값 그대로가 나온 것을 확인할 수 있다. 처음에는 Factorial 연산을 보여주는 것이 좀 더 뒤였으나, 하위 3bit를 무시하여 offset을 계산하여 0707과 0700과 메모리 주소가 겹치므로 덮어쓰기가 되었다. 이를 확인 후 지금 내용으로 수정했다.



팩토리얼이 7!이 제대로 작동하는 가를 다시 한번 보면서 operand에 7의 값을 입력, intrEn에 0을 입력한 후 연산 시작 신호를 준다. 이후 opdone을 관찰한다.



이후 opdone에 3이 써지면서 연산이 종료됨을 알려준다.



연산이 완료된 이후 result_1에 대해 보면 결과인 5040이 나온 것을 확인할 수 있다. 이를 메모리에 넣기 위해 결과를 m_din으로 0003 주소에 저장을 한 후, factorial core를 clear를 해 주었다. 이후 메모리가 기능을 잘 수행하고 있는지 0003에서 read를 한 결과인 5040이 저장된 것을 확인할 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Sat Dec 02 19:33:03 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	Top
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	8,699 / 41,910 (21 %)
Total registers	16788
Total pins	150 / 499 (30 %)
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 %)

해당 flow summary는 Top에 대한 것으로, Logic utilization은 8699이 나왔고, register의 수는 16788이다. Pins의 수는 150이다.

5. Conclusion

결론 및 고찰

이번 프로젝트에서 중요한 것은 multiplier를 이용하여, factorial core를 만드는 것이었다. Memory와 bus에 대해서는 수월하게 진행할 수 있었지만, factorial core에 대해 원래 설계했던 대로 state 수를 3개로 만들기에는 좀 더 복잡했다. 단순히 초기 상태의 INIT, 연산시작 혹은 중을 나타내는 START, 연산 종료인 DONE으로 하려 했으나, clear의 상태와 연산 중에서도 operand를 감산하는 작업을 따로 state로 정의해야 구현하기 쉬울 것으로 보여 state 수가 5개인 것을 가지고 사용했다. 이후 operand에 감산이 두 번 이루어지는 것을 확인하며 factorial에 대한 연산이 제대로 이루어지지 않은 것을 확인했다. 아무것도 시행하지 않는 state를 하나 추가하여 감산을 두 번 시행하는 것을 피할 수 있었다. 총 6개의 state를 이용하였다. 이외의 모든 모듈은 초기 설계할 때 생각했던 대로 잘 된 것 같다.

초기에는 clear가 들어오면 명시적으로 0의 값을 주어 clear를 해제하는 것이 아닌 factorial core에서 자동적으로 clear를 해제하는 것으로 생각했다. 명세서의 버전이 바뀌면서 다시 읽었을 때, 명시적으로 0의 값을 입력한다는 것을 인지한 후, clear이후에도 default로 다른 state로 가는 것을 막았다. Bus에 대한 것도 request가 들어오지 않은 경우에도 sel에 대해 신호가 1이 될 수 있었는데, 이후 명세서 v3를 참조하여 해당 현상에 대해서도 고쳤다.

이 프로젝트를 통해 factorial 연산과 비슷한 시행을 진행했다. resut_l와 operand의 곱으로만 이루어지니 어느 순간 factorial중 중간 결과값이 64bits를 넘는 경우에는 factorial처럼 연산이 되지 못한다는 단점이 생겨버린 것이다. 결과적으로 완전한 factorial 연산의 결과가 나오는 것을 실험해 보니 20!정도 까지는 가능하다는 것을 확인했다. 21!이 되면 result_l에 64bits가 꽉 차게 되어 그 이후에 값들은 결과를 제대로 관측하지 못한다. 정상적인 값이 나오는 것을 기대하기 위해서는 operand의 값을 20이하의 값을 주면서 result_l의 값이 factorial 연산이 잘 이루어지는가에 대하여 확인했다.

프로젝트를 진행하게 되면서 cpu의 동작원리에 대해 간단한 형태로 간접 체험을 한 것 같다. Bus를 이용하여 memory와 factorial core로 factorial core를 명령어 들에 대해 실행하는 어떤 slave라고 생각한다면 bus에서 이 두 slave를 조절하면서 사용하는 식으로 명령을 주고 해당 명령을 실행하는 것까지 설계를 하고 직접 만들면서 이제 여러 개의 salve를 컨트롤할 수 있으면 factorial core 뿐만이 아닌 여러 기능을 추가하여 bus로 컨트롤을 해 더 성공적인 cpu의 구현이 가능해질 것이라고 생각한다.

6. Reference

[1] References should be given in this section.

[2] D. M. Harris and S. L. Harris, Digital design and computer architecture, Morgan Kaufmann, 2007 (This is an example.)