Factorial computation system

2020202031 김재현

I. Introduction

팩토리얼 연산을 진행하는 Factorial computation system를 구현합니다. 하지만 모두가 알고있는 팩토리얼 연산이 아닌, 팩토리얼 연산을 진행하는 중, 곱셈결과의 하위 64비트만을 operand와 곱연산하며, 하위 64비트가 0일 경우 상위 64비트를 operand와 곱연산하는 팩토리얼 연산을 진행하는 Factorial computation system를 구현합니다.

master, bus, slave로 구성된 Top module을 구현합니다. master는 테스트벤치가 담당하고, slave는 memory, factoCore 두 개로 구성됩니다.

II. Project Specification

1) BUS

m_req에 따라 m_grant가 정해지고, m_grant와 m_address에 따라 s_sel가 정해집니다. 정해진 s_sel에 따라 m_dout이 어느 slave로 입력될지가 정해지고, 동시에 m_din으로 들어가는 s_dout이 정해집니다. 다만 주의해야할 점은, master의 grant와 s_dout은 clock에 synchronous하게 변화합니다.

Address decorder에 m_grant가 입력되는 이유는, address가 올바른 Slave memory map에 접근하고 있더라도, m_grant가 0 이면 slave에 write/read 어떠한 접근도 하면 안되기 때문에, m_grant가 0일 때의 예외처리로 s0_sel, s1_sel 전부를 0으로 주기 위함입니다.

2) ram

ram은 레지스터에 값을 write/read 하는 동작만을 수행합니다. cen이 0이면 아무런 동작도 수행되면 안되기 때문에, s_dout에 0을 출력합니다. cen이 1이고 s_wr이 1이면 write 동작을 수행해야 하기 때문에, s_dout에는 0을 출력시키고, s_addr에 해당하는 주소의 레지스터에 s_din을 write합니다. cen이 1이고 s_wr이 0이면 read 동작을 수행해야 하므로 s_dout에 addr에 해당하는 주소의 레지스터를 출력시킵니다.

이러한 동작을 이중 if문을 통해 cen의 값에 따라 1차적으로 상황을 구분하고, s_wr 값에 따라 2차적으로 상황을 구분함으로써, cen, s_wr : 0 0, 0 1, 1 0, 1 1 4가지 경우를 모두 커버할 수 있습니다.

3) FactoCore

a) state에 관하여

reset_n이 0일 때, INIT이 되고, reset_n이 1이 되면 다음 clock이 rising edge에서 INPUT state로 넘어갑니다.

INPUT state에서 BUS를 통해 writable registers에 값을 입력하고, opstart[0]에 1을 씀과 동시에 FCTRL state로 넘어갑니다. next_opstart[0]이 1일 때, next_state를 FCTRL state로 설정해두어, clock의 rising edge가 들어온 후, opstart[0]에 1을 write함과 동시에 state가 FCTRL state로 넘어갈 수 있게끔 설계했습니다.

FCTRL state에서는 실질적으로 booth multiplier의 input으로 사용되는 in_operand가 1 초과인지, 1 이하인지 판별하는 역할을 합니다. in_operand가 1 초과면 MUL state로 넘어가고, 1 이하면 곱셈연산을 진행하지 않고 바로 END state로 넘어갑니다.

MUL state에서는 booth multiplier에서 곱셈을 완수하여 muldone 신호에 1을 넘겨줄 때까지 대기합니다. muldone 신호가 1이 되면, next_state를 FCTRL로 설정해줍니다.

END state에서는 아무런 연산이 진행되지 않고 오직 writable registers에 write 하는 동작만을 수행합니다. next_opclear[0]이 1일 때 next_state를 CLEAR로 설정하여 opclear[0]에 1이 들어옴과 동시에 CLEAR state로 넘어갈 수 있도록 설계했습니다.

CLEAR state에서는 아무런 연산이 진행되지 않고 오직 writable registers에 write 하는 동작만을 수행합니다. next_opclear[0]이 0일 때 next_state를 INPUT으로 설정하여 opclear[0]에 0이 들어옴과 동시에 INPUT state로 넘어갈 수 있도록 설계했습니다.

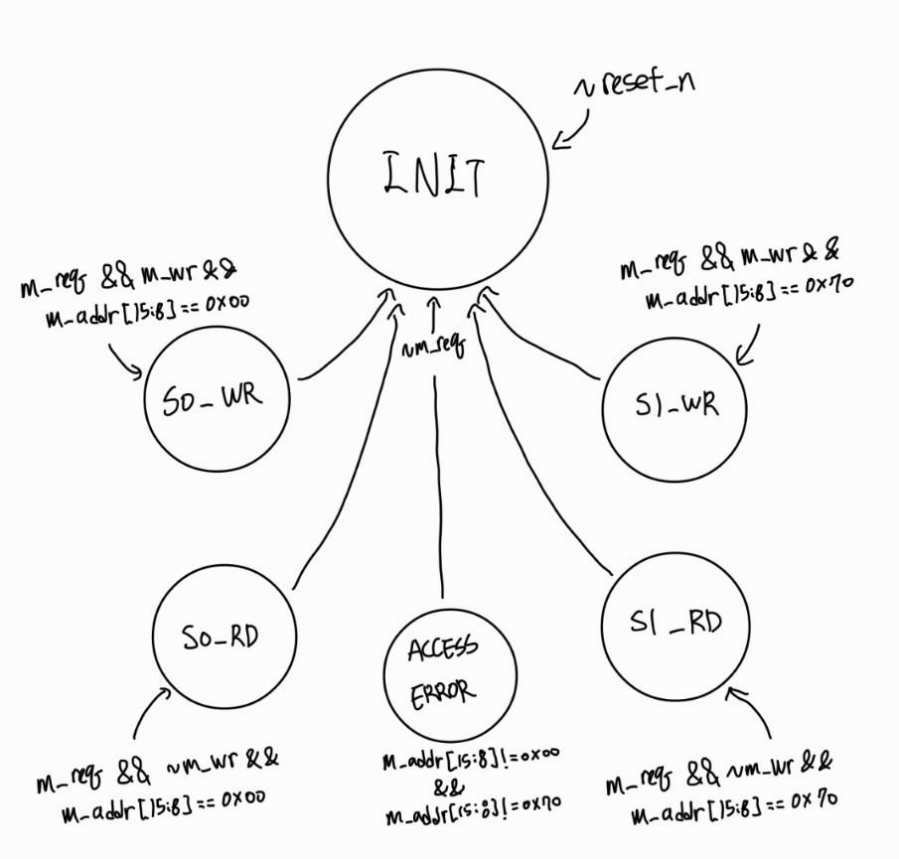
b) registers에 관하여
registers의 경우 W/R Type의 registers들이 존재하므로 이 두 type의 registers를 따로 관리해줬습니다. W Type의 경우, clk의 rising edge에서 값이 write 되므로, (s_sel, s_wr, s_addr, s_din, state)의 변화에 따라 W Type next_registers들의 값을 계속해서 변경해주고, clk의 posedge에서 (s_sel, s_wr, s_addr, state)의 값에 따라 W Type registers에 W Type next_registers의 값을 대입해줍니다.
입력된 operand 값은 FactoCore 내부에서 값을 변경시키면 안되기 때문에, operand 값을 in_operand 레지스터에 복사해서, booth multiplier의 입력 값으로 전달했습니다. in_operand로의 값의 복사는 state INPUT에서 FCTRL로 넘어가면서 진행이 되며, 감산은 state가 MUL로 넘어가면서 진행됩니다. multiplicand 레지스터에 result_h or result_l 값 중 어느 값을 대입할지에 대한 조건문 또한 state가 MUL로 넘어가면서 실행이 됩니다. MUL로 넘어갈 때 mulstart, mulclear 값 또한 1, 0으로 변경해줌으로써, in_operand, multiplicand의 값을 정해주고, 곱셈 연산을 시작하게 됩니다.

III. Design Details

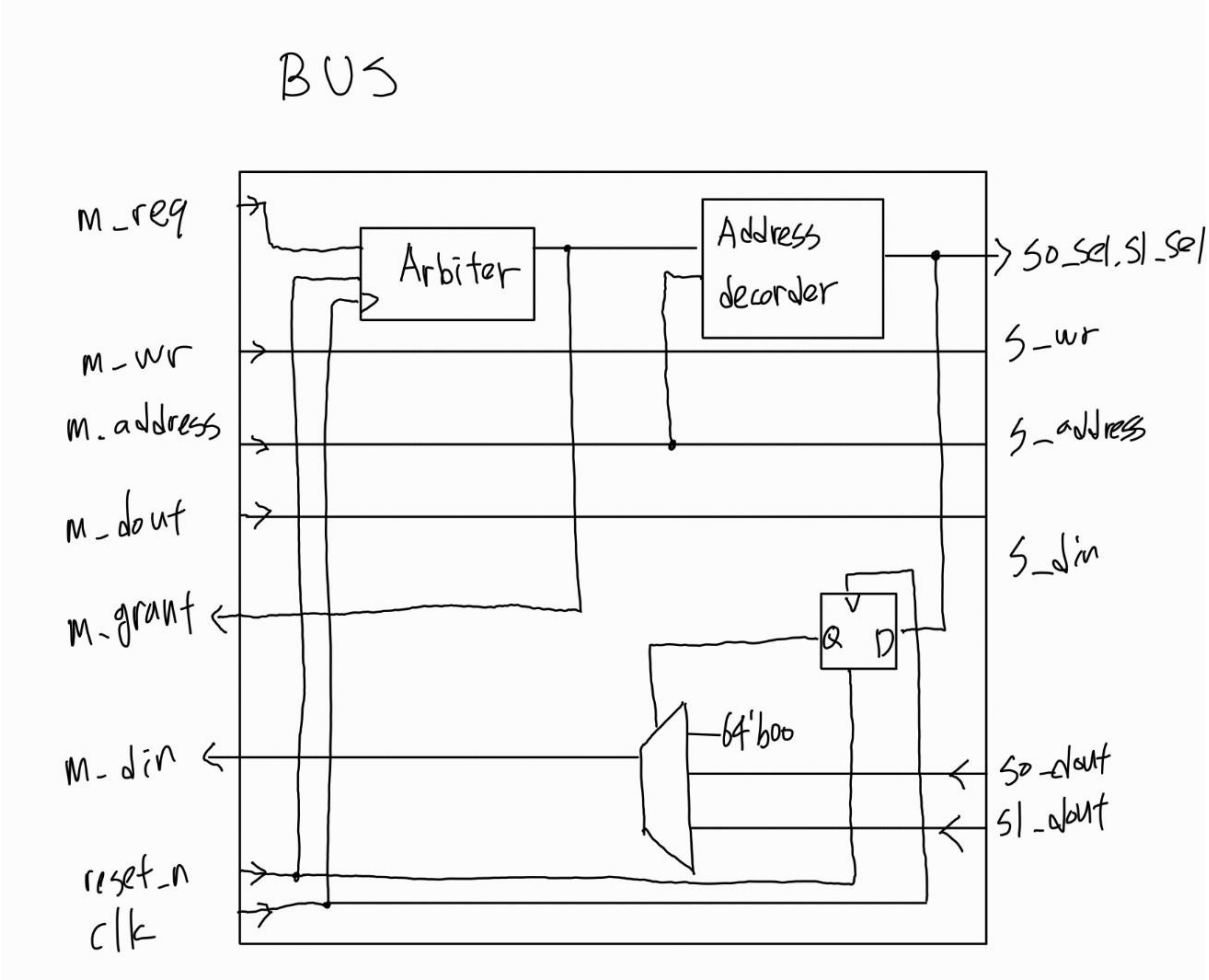
1) BUS

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 address
	s_wr	1	Slave write/read
	s_din	64	Slave data input

BUS의 Pin description 입니다.



BUS의 state diagram입니다.

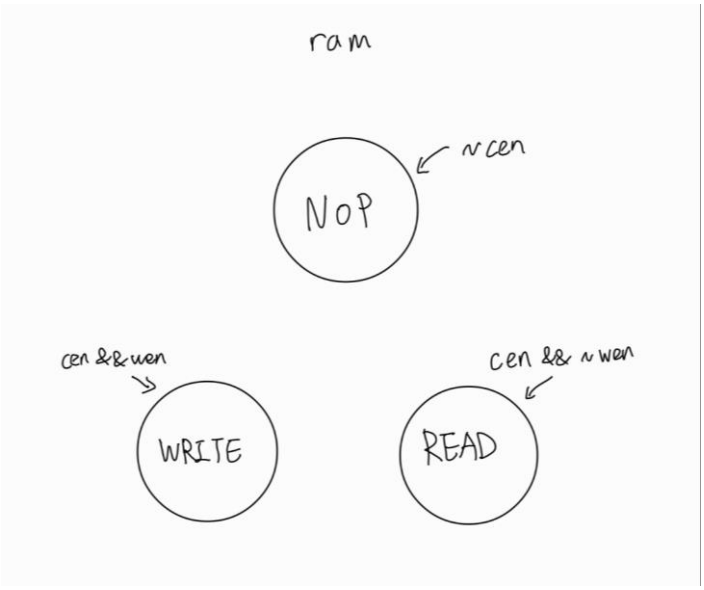


BUS의 Design입니다.

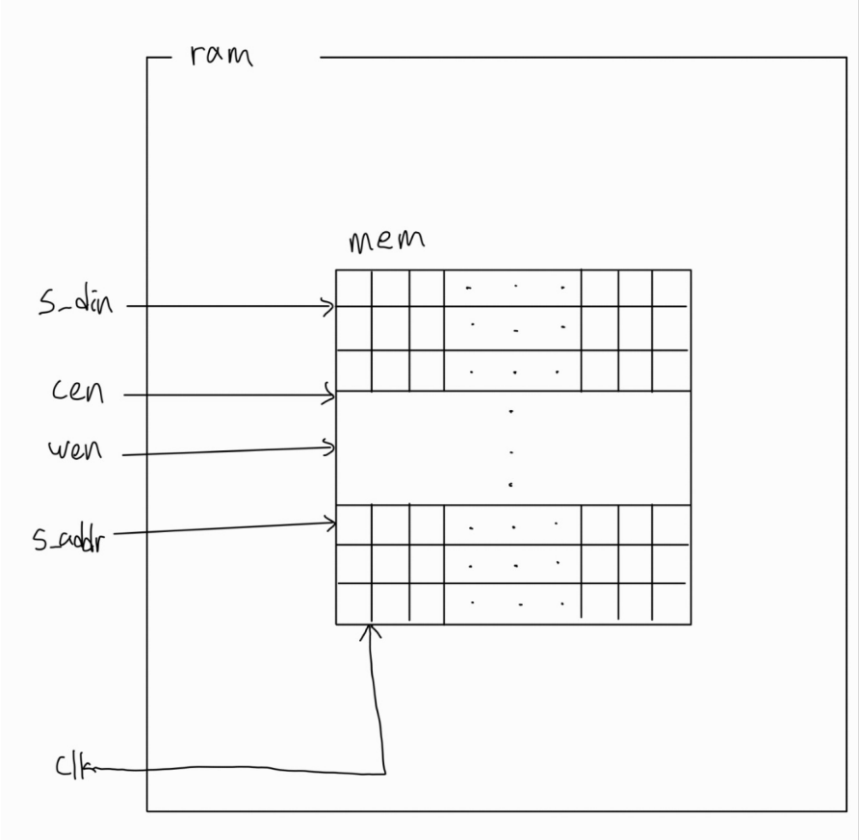
2) ram

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 in
Output	s_dout	64	Data out

ram의 Pin description입니다.



ram의 state diagram입니다. 각 state에서 이동 가능한 state가 제한돼있지 않고, 오직 cen, wen 값에 따라 state가 정해지기 때문에 state 간의 연결선은 필요하지 않습니다.



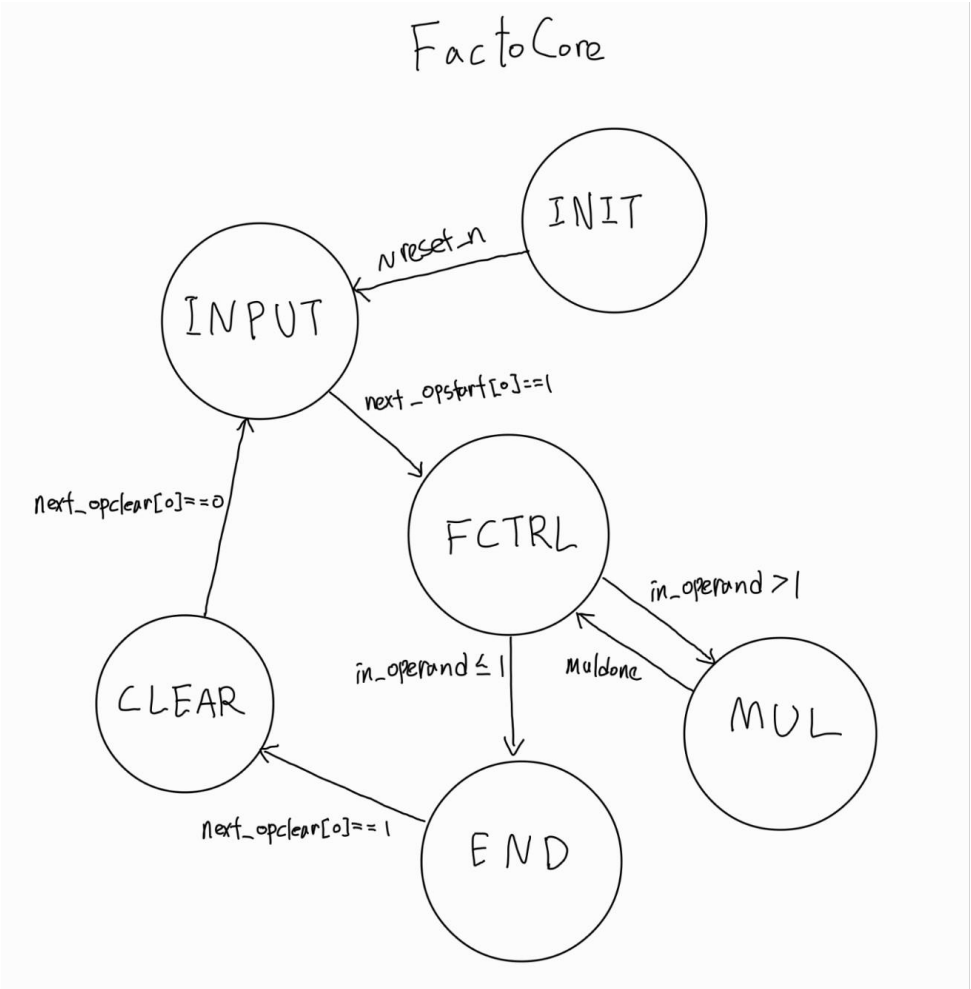
ram의 Design입니다.

action / state	INPUT	OUTPUT
NOP		s_dout = 0
WRITE	mem[s_addr] = s_din	s_dout = 0
READ		s_dout = mem[s_addr]

3) FactoCore

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

FactoCore의 Pin description입니다.



FactoCore 의 state diagram입니다.

4) Top

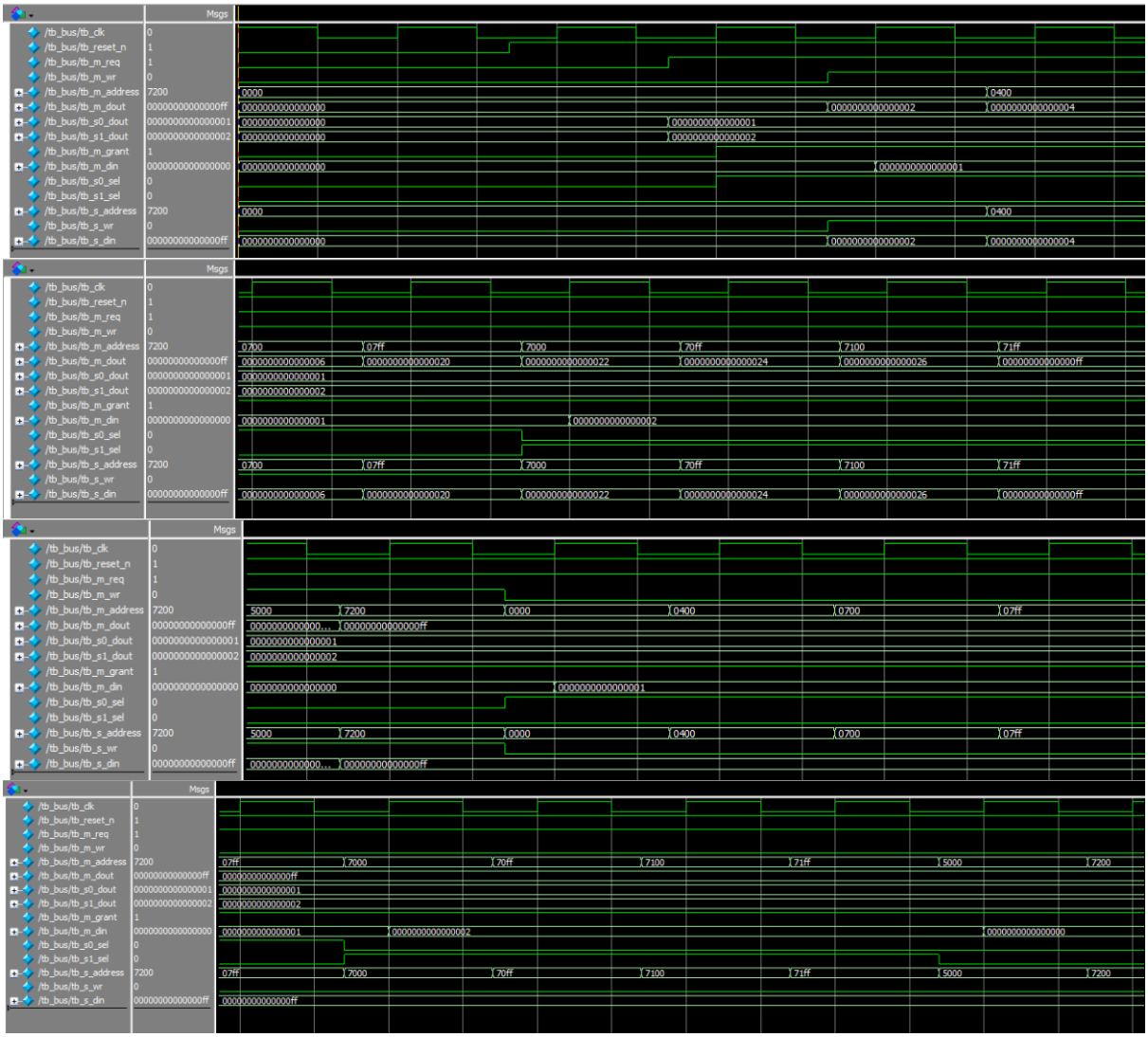
Direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset
	m_req	1	Master request
	m_wr	1	Master write/read
	m_addr	16	Master address
	m_dout	64	Master data output
Output	m_grant	1	Master grant
	m_din	64	Master data in

	interrupt	1	Factorial core interrupt

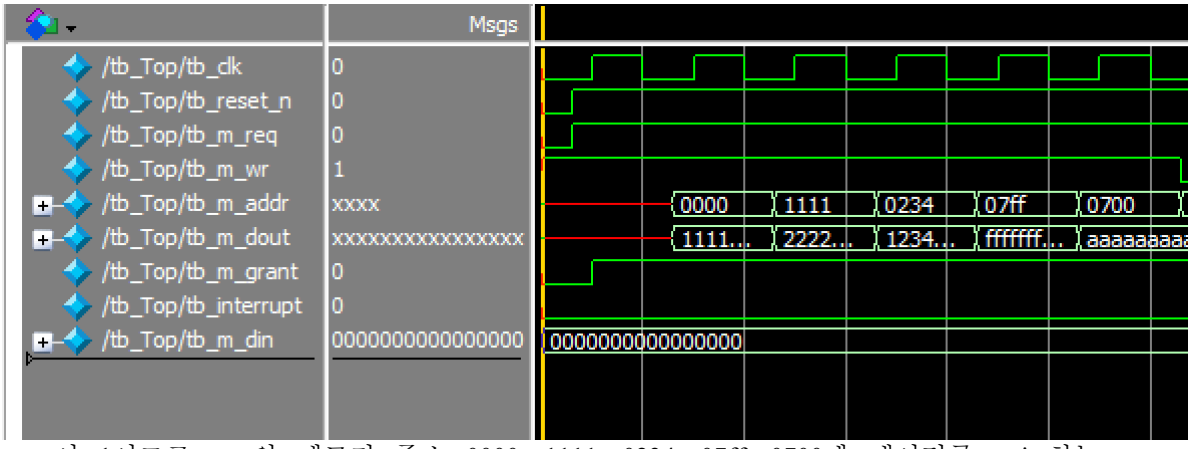
Top의 Pin description입니다.

IV. Design Verifiaction Strategy and Results

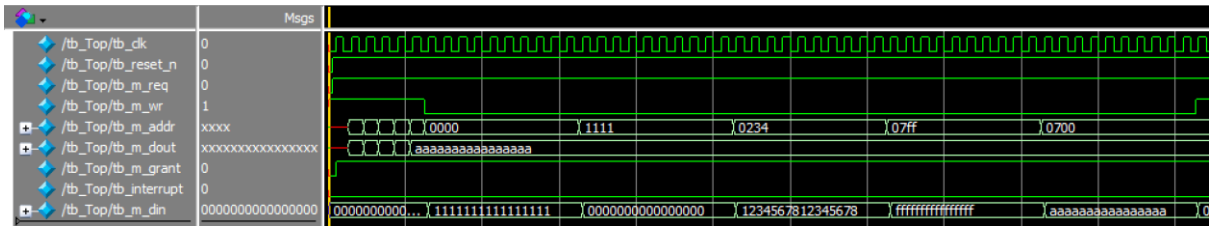
1) BUS



2) ram

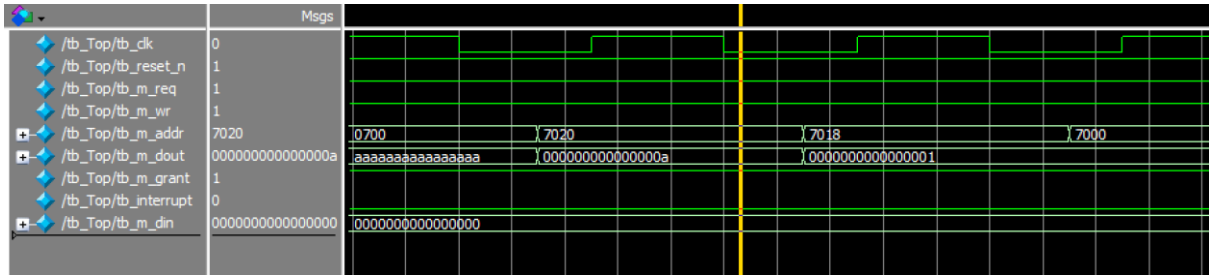


s_wr이 1이므로 ram의 메모리 주소 0000, 1111, 0234, 07ff, 0700에 데이터를 write하는 중입니다.

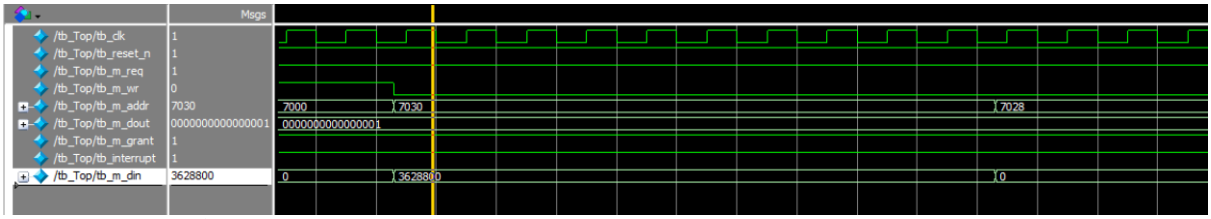


s_wr이 0이므로 ram의 메모리 주소 0000, 1111, 0234, 07ff, 0700에 write한 데이터들은 read 함으로써 write 동작이 잘 수행됐음을 확인할 수 있습니다.

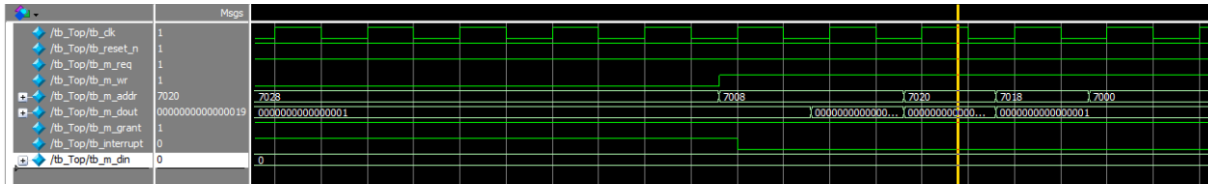
3) FactoCore



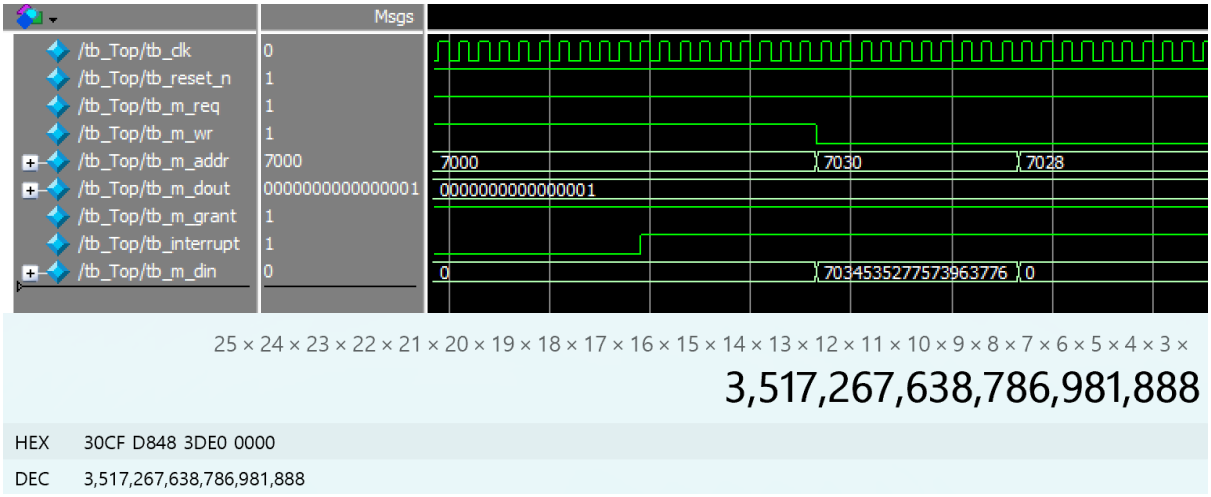
operand에 10진수 10, intrEn 1, opstart 1을 부여함으로써 연산을 시작한다.



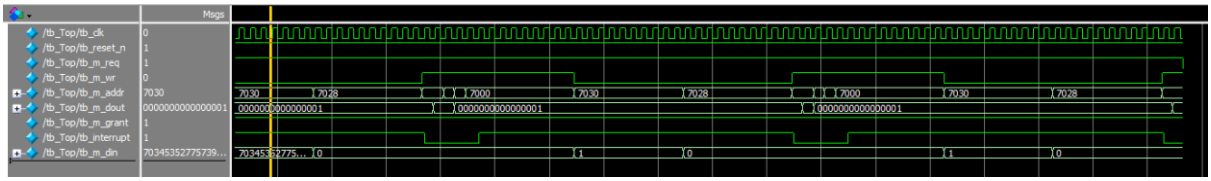
interrupt가 1이 출력되는 것으로 보아, 연산이 완료됐고, result_l을 read해보니, 10! 값인 3628800고, result_h는 0임을 확인할 수 있습니다.



opclear를 1로 주고 0으로 되돌리므로써 clear를 실행해줬고, 이번에는 operand에 10진수 25, intrEn 1, opstart 1를 부여함으로써 연산을 시작합니다.



그 결과, 값이 제대로 출력되는 것을 확인할 수 있습니다.



operand로 0 or 1을 부여하면 결과 값이 1이 출력되는 것을 확인할 수 있습니다.

V. Conclusion

저는 clk rising edge에서 registers에 값이 입력됨과 동시에 state도 함께 이동하도록 구현하고자 했습니다. 그러나 registers는 clock에 synchronous하므로 registers에 값이 입력되고 한 사이클 후에 state를 변경시킬 수 밖에 없었습니다. 그렇기에 버스를 통해 입력되는 값을 next_registers가 계속해서 추적하고 clk rising edge에서 registers에 next_registers를 입력해주고, next_state는 next_registers를 추적하고, 역시 마찬가지로 clk의 rising edge에서 state가 next_state로 이동하도록 구현함으로써 문제를 해결했습니다.

VI. Reference

[1] 이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2023
[2] 이준환 교수님/컴퓨터공학기초실험2/광운대학교(컴퓨터정보공학부)/2023