Factorial computation system

2022202075 우나륜

Abstract

본 문서는 2023학년도 2학기 디지털논리회로2 과목 프로젝트인 Factorial computation system을 Quartus Prime Lite Edition 18.1 프로그램과 Verilog 언어를 사용하여 구현하였다. 본 프로젝트는 이전 과제에서 구현한 Carry Look-ahead adder, memory, bus, booth multiplier 등을 사용하여 주요 module인 Top, BUS, ram, FactoCore을 구현한다. 문서의 마지막에서는 각 module에 대한 testbench와 그에 대한 waveform을 통해 기능이 제대로 수행되었는 지 검증한다.

I. Introduction

본 프로젝트는 2023학년도 2학기 광운대학교 디지털논리회로2 과목의 프로젝트이며, Factorial computation system를 Verilog를 사용하여 설계한다. Factorial computation system은 사용자가 원하는 factorial 연산을 수행하며, Memory에 접근하여 특정 주소에 값을 작성하거나 값을 읽을 수 있다. 구현한 프로젝트는 1개의 master interface와 2개의 slave interface를 가지며 각 interface들은 bus를 통해 상호작용할 수 있다. 문서의 II. Project Specification에서는 Top, BUS, ram, FactoCore 네 개의 모듈에 대한 작동원리와 알고리즘 등을 설명한다. III. Design Detail에서는 각 component 별 pin description, block diagram, finite state machine을 설명한다. IV. Design Verification Strategy and Result는 각 component 별 검증전략과 그에 대한 testbench waveform 결과를 설명한다. 또한, top module인 Top의 flow summary와 RTL map viewer를 통해 top 모듈과 내부에 인스턴스 된 모듈들의 구조를 확인한다.

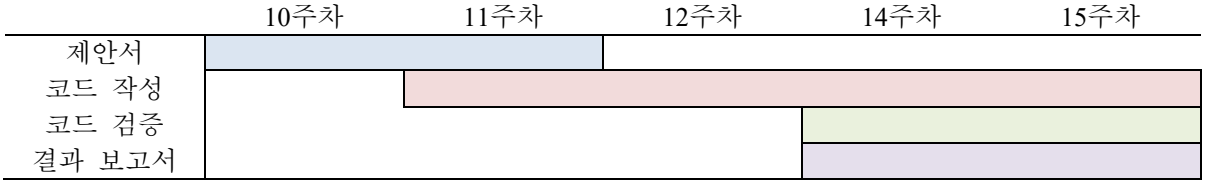


Table I. Project Schedule

위의 Table I은 본 프로젝트의 진행 일정을 나타낸다. 프로젝트 시간은 2023년 11월 07일 (10주차)부터 2023년 12월 06일 (14주차) 동안 진행되었다. 프로젝트가 공고된 10주차부터 11주차 동안은 프로젝트의 설계 및 제안서 작성을 진행하였다. 그리고 11주차부터 14주차 동안 factorial computation system의 코드를 작성하였으며, 대략적인 코드가 작성된 14주차에는 testbench를 작성하여 검증을 진행하면서 코드를 수정하였다. 또한, 14주차부터 보고서 작성을 시작하였고, 부족한 부분을 계속 보완하였다. 15주차에는 제공된 testbench를 사용하여 검증을 진행하였고, 해당 testbench를 통해 발생하는 오류 또는 구현 결함을 수정하고, 이에 따른 보고서를 수정하였다.

II. Project Specification

1. Top component

Top module 내에는 각각 bus, memory, factorial core module들이 instance로 선언되어 있다. 외부로부터 받은 input들을 bus, memory, factorial core에 연결하고, memory와 factorial core로부터 얻은 s0_dout, s1_dout의 값들 bus로 입력한 뒤, slave select signal에 따라 한 가지의 값만 m_din에 작성하여 출력한다. 만약, 어떠한 slave interface도 선택되지 않으면, m_din은 0의 값을

가지게 된다. Slave interface는 외부로부터 입력받은 m_addr의 값에 따라 선택되며, m_dout의 값을 통해 해당 주소에 값을 작성하거나 값을 읽어올 수 있다.

2. Bus component

Bus module은 master interface와 slave interface 사이의 상호작용을 수행한다. Slave interface는 slave 0인 Memory와 slave 1인 Factorial core가 있다. Master write/read (m_wr), master data output (m_dout_, master address (m_addr), master request (m_req)는 assign 구문을 사용하여 각각 s_wr, s_din, s_addr, m_grant에 할당한다.

Master data input (m_din)은 slave select signal에 따라 값이 결정된다. 만약 s0_sel = 1이면, slave 0의 output인 s0_dout을 m_din에 할당한다. 만약, s1_sel = 1이면, slave 1의 output인 s1_dout을 m_din에 할당한다. 만약 s0_sel과 s1_sel의 값이 모두 0이면 m_din = 0을 할당한다. Master data input을 결정하는 과정을 3항 연산자(ternary operator)를 사용하여 구현하였다.

Always 구문의 sequential logic에서는 reset_n과 m_req, m_addr의 값에 따라 s0_sel과 s1_sel의 값을 변화하는 기능을 구현하였다. 만약 reset_n = 0, m_req = 0 또는 m_addr의 값이 slave 0과 slave 1의 address map region을 벗어나면, s0_sel과 s1_sel의 값을 0으로 초기화하였다. 만약, m_addr의 값이 0x0000 ~ 0x07FF 사이면, s0_sel = 1, s1_sel = 0으로 설정한다. 만약, m_addr의 값이 0x700 ~ 0x71FF 사이면, s0_sel = 0, s1_sel = 1로 설정한다. 아래의 Table II은 memory와 factorial core의 address map region을 나타낸다.

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

Table II. Address map region

3. Ram component

Ram (Memory) module은 bus로부터 s0_sel, s_wr, s_addr[7:0], s_din, clk을 입력받는다. Ram module은 처음에 64-bit word size를 가지고, storage size가 256인 memory를 선언한 뒤, 모든 값을 0으로 초기화한다.

만약 cen (= s0_sel)의 값이 0이면, ram module이 선택되지 않은 것이므로 아무런 동작을 수행하지 않는다. Cen의 값이 1이고 wen (s_wr) = 1이면, s_addr의 주소에 s_din의 값을 write하고, s_dout은 0을 출력한다. Cen = 1, wen = 0이면, s_addr의 주소에 저장된 값을 s_dout에 write한다.

4. Factorial Core component

Factorial Core를 구현한 FactoCore module은 bus로부터 clk, reset_n, s_wr, s_addr, s_din을 입력받고, s_dout과interrupt 값을 출력한다. Factorial Core module 내에는 FactoCntr, FactoCore_os, multiplier가 instance되어 있으며, 각각 FC, FO, Booth라는 이름으로 선언되어 있다. 또한, operand, opstart, opclear, opdone, intrEn, result_h, result_l 등의 여러 register들이 선언되어 있으며, 이들은 offset 값을 사용하여 register에 값을 write하거나 read할 수 있다. Module 내에는 parameter를 사용하여 Factorial Core에 필요한 state들을 binary encoding을 사용하여 선언하였다. Always 구문을 사용하여 reset_n = 0일 때, offset에 필요한 모든 register들을 0으로 초기화하고, state는 CLEAR가 된다. 이외의 경우에는 Factorial Controller에서 얻은 next 값들을 사용하여 값을 할당한다. 아래는 Factorial Core내에 instance로 선언된 module들의 설명이다. Table III를 통해 factorial core내에서 사용하는 register들의 offset, type, default value등을 확인할 수 있다.

Offset	Type	Bit width	Name	Description	Reserved bits	Default value
0x00	W	64-bit	opstart	[0] bit에 1이 써지면, operand의 값을 사용하여 연산을 수행한다.	[63:1]	64'd0
0x08	W		opclear	[0] bit에 1이 써지면, 모든 register의 값을 default value로 초기화한다,	[63:1]	
0x10	R		opdone	연산이 시작되면 [1] bit에 1을 쓰고, 연산이 완료되면, [0] bit에 1을 쓴다.	[63:2]	

0x18	W		intrEN	intrEN[0] = 1이면, 연산 종료 후 interrupt 신호가 발생한다.	[63:1]	
0x20	W		operand	Facotorial의 피연산자가 저장되어 있으며, 32-bit signed number를 저장한다.	[63:32]	
0x28	R		result_h	Factorial 곱셈연산 결과의 상위 64-bit를 저장한다.	-	
0x30	R		result_l	Factorial 곱셈연산 결과의 하위 64-bit를 저장한다.	-	64'd1

Table III. Register description of Factorial core

4.1 Factorial Controller module (FactoCntr.v)

Factorial Controller는 bus로부터 입력받은 주소값 s_addr[7:3]으로 상위 5비트만 사용하고 이를 offset이라 부른다. 만약, s_sel의 값이 0이거나 s_wr의 값이 0이면, next state는 INIT이 된다. 그렇지 않으면, offset의 값에 따라 해당 register에 s_din의 값을 write한다. 만약 opclear[0] = 1이면, next_state는 CLEAR가 된다. Factorial core 내에서 선언된 state와 그에 대한 자세한 설명은 III. Design Details에서 다룬다.

Combinational logic에서는 offset에 해당하는 register에 값을 write하는 것뿐만 아니라, 현재 state와 register들의 값에 따라 next_state를 결정하는 기능을 case문을 사용하여 구현하였다. 또한, intrEN[0]과 opdone[0]의 값이 1이면, interrupt의 값을 1로 초기화하고, 그렇지 않으면 0으로 할당하였다.

- 만약, 현재 state가 mul_clear라면, multiplier에 입력되는 다음 opclear값을 0으로 설정하고, multiplier의 result가 0이 아니면, 다음 result_h와 result_l에 각각 result의 상위 64비트, 하위 64비트를 저장한다. 그리고 next_state를 EXEC로 설정한다.
- INIT 상태는 factorial core의 initial state이며, opstart[0]에 1이 작성되기 전까지 대기한다. 만약, opstart[0] == 1이면, 다음 opdone[1] = 1, next_state를 EXEC로 설정한다.
- CLEAR 상태는 intrEN, operand, opclear를 제외한 모든 offset의 register값들을 default value로 초기화하는 과정을 수행한다. 또한, multiplier의 다음 opclear와 opstart값을 모두 1로 초기화한다. 모든 값을 초기화한 뒤에는 INIT 상태로 옮겨간다.
- EXEC 상태는 factorial 연산을 수행중인 상태로, operand의 값이 0이되기 전까지 연산을 계속 수행한다. 만약, booth multiplier의 mul_opdone 신호가 1이면, 곱셈 연산을 완료한 것이므로, 다음 mulitiplier의 opclear, = 1, operand – 1, next_state는 mul_clear로 설정한다. mul_opdone = 0이고 operand의 값이 0이면, 다음 opdone[0] = 1, next_state는 DONE으로 설정한다. 그렇지 않으면 EXEC 상태를 반복한다.
- DONE 상태는 factorial 연산을 마친 상태로, opclear[0] == 0일 때까지 DONE 상태를 유지한다.

4.2 Factorial Output logic module (FactoCore_os.v)

Factorial Output logic은 입력받은 s_wr과 reset_n, s_addr의 값에 따라 s_dout에 값을 write한다. Always 문의 sequential logic에서 reset_n의 값이 0이거나 s_wr의 값이 1이면 s_dout은 0이 된다. s_wr = 0이면, offset 값에 따라 opdone, result_h, 또는 result_l의 값을 s_dout에 write한다. Combinational logic에서는 booth multiplier에 입력할 다음 피연산자를 선택하는 기능을 수행한다. 만약 reset_n = 0이면, next_mul = 0이 된다. Result_l의 값이 0이면, next_mul = result_h가 된다. 그렇지 않으면, next_mul = result_l이 된다.

4.3 Radix-2 Booth multiplier (multiplier.v)

Booth multiplication algorithm은 2의 보수를 사용하여 signed number를 곱하는 곱셈 알고리즘이다. multiplier module은 radix-2 booth multiplier를 수행하며, multiplier_ns와 multiplier_os를 instance하여 구현하였다. 입력받은 64-bit multiplier와 multiplicand를 입력받아 128-bit result를 출력한다. Multiplier의 연산이 종료되면, op_done의 값이 1이 된다.

4.3.1. multiplier_os

Multiplier_os은 multiplier의 output logic에 대한 module이다. Multiplier module로부터 op_clear, state, cnt, multiplier와 multiplicand를 입력받고, op_done과 result를 output으로 가진다. Cla64 module을 2개 instance화하여 이전 result의 상위 64비트와 multiplicand를 각각 더하고 뺀 값을 w_add, w_sub에 저장하였다. Always 문의 sequential logic에서 reset_n = 0 또는 op_clear = 1이면 result_add와 result_sub의 값을 0으로 초기화한다. 그렇지 않으면 result_add와 result_sub에 위에서 구한 w_add, w_sub의 값을 할당한다. 그 밑의 logic에서는 output인 result 값을 매 cycle마다 할당해준다. 만약 reset_n = 0, op_clear = 1이면, result = 0이된다. 그렇지 않으면 w_result의 값을 할당한다.

Combinational logic에서는 입력받은 state와 opclear에 따라 연산을 수행한다. 만약, op_clear의 값이 1이면, 모든 register들의 값을 0으로 초기화한다. 그렇지 않으면 case문을 사용하여 각 state에 해당하는 연산을 수행한다. 만약, INIT 상태이면w_result의 하위 64비트에 multiplier를 할당하고 나머지 register들은 0으로 초기화한다.

EXEC은 곱셈을 수행중인 상태이다. Radix-4는 32 cycle이 필요하므로, cycle인 cnt의 값이 0b1000000이면 계산을 시작할 준비를 한다. 그렇지 않으면, x_i, x_{i-1}, x_{i-2}를 사용하여 연산을 수행한다. 해당 비트들에 대한 연산 기준은 아래 Table IV와 같다.

x_i	x_{i-1}	x_{i-2}	Operation	Description	y_i	y_{i-1}
0	0	0	Shift only	String of zeros	0	0
0	0	1	Add and shift	End of ones	0	1
0	1	0	Add and shift	A single one	0	1
0	1	1	Add twice and shift	End of ones	1	0
1	0	0	Subtract twice and shift	Beginning of ones	-1	0
1	0	1	Subtract and shift	A single zero	0	-1
1	1	0	Subtract and shift	Beginning of ones	0	-1
1	1	1	Shift only	String of ones	0	0

Table IV. Radix-4 Booth multiplication table

현재 result[1]를 x_i, result[0]를 x_{i-1}, 이전 result[1]를 x_{i-2}라 하자. 각각 위의 값에 따라 w_temp의 상위 64비트에 +A, +2A, -A, -2A를 저장한다. 그리고 하위 64비트에 result의 하위 64비트를 저장한다. 그리고 결합연산자를 사용하여 Logical Right shift 연산을 2번 수행한다. 그리고 현재 result[1]을 bb_bit에 저장한다. 만약 000, 111인 경우에는 w_temp에 result를 저장하고 shift 연산만 수행한다.

DONE 상태는 곱셈 연산이 종료된 상태로 op_done을 1로 설정하고, w_result에 result를 저장하고 나머지 값들은 0으로 초기화한다. 만약, 위 상태 중 해당되는 상태가 없다면, 오류이므로 모든 값을 x (unknown)로 초기화한다.

4.3.2. multiplier_ns

multiplier_ns module은 booth multiplier에서 사용할 cycle인 cnt를 계산하는 기능을 수행한다. 또한, reset_n, op_clear, op_start에 따라 state를 설정한다. Cla8 module을 하나 instance화하여 next_cnt에 -1을 더하여 result_cnt에 값을 받는다. 그런데 여기서 next_cnt는 7비트로 선언되어 있으므로 결합연산자를 사용하여 맨 앞에 0을 붙여 입력한다.

Sequential logic에서 reset_n = 0이면 cnt의 값을 0b1000000로 초기화하고 INIT 상태로 설정한다. 만약 op_clear = 1 또는 op_start = 0이면 cnt의 값을 0으로 초기화하고 INIT 상태로 설정한다. 그렇지 않으면, cnt = result_cnt[6:0]로 저장하고 state는 next_state로 할당한다.

Combinational logic는 state transition을 수행한다. Case 문을 사용하여 각 상태에 대한 기능을 수행한다. INIT 상태에서 op_start = 1 && op_done = 0 && op_clear = 0이면, next_state는 EXEC로 설정하고 next_cnt는 0b1000000로 할당한다. EXEC 상타애서 cnt의 값이 0이되면, next_state는 DONE으로 설정한다. 그렇지 않으면 EXEC 상태를 반복하며, next_cnt에 cnt 값을 할당한다. DONE 상태에서는 현재 상태를 계속 유지하며, next_cnt = 0으로 초기화한다. 그 외의 상태인 경우에는 오류이므로 next_state와 next_cnt의 값을 x (unknown)으로 설정한다.

III. Design Details

1. Top

아래의 Table V는 Top module의 input/output port와 그에 대한 bit width, description을 나타낸다. Interrupt 신호는 slave 1 (Factorial Core)에서 발생하지만, bus를 통하지 않고 바로 top module에 값이 전달된다. Figure I은 Top module의 schematic symbol이다.

Module name	Direction	Port name	Bit width	Description
Top	Input	clk	1-bit	Clock
		reset_n		Active-low reset
		m_req		Master request
		m_wr		Master write/read
		m_addr	15-bit	Master address
		m_dout	64-bit	Master data output
	Output	m_grant	1-bit	Master grant
		m_din	64-bit	Master data in
		Interrupt	1-bit	Factorial core interrupt

Table V. Pin description of Top

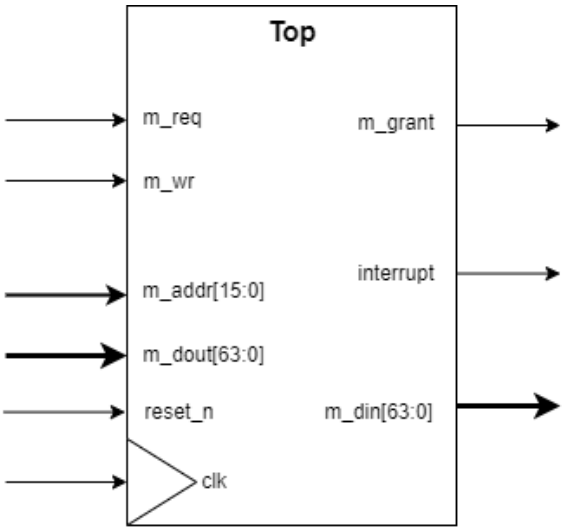


Figure I. schematic symbol of Top

Top module은 내부에 bus, memory, factorial core를 instance하여 구성하였다. 아래의 Figure II를 통해 top module의 block diagram을 확인할 수 있다. 파란색 선은 bus module에서 slave 0 (memory)와 slave 1 (factorial core)로 입력되는 data들을 표현한 것이고, 빨간색 선은 slave interface에서 bus module로 입력되는 data들을 표현한 것이다.

Top module은 testbench에서 입력값으로 m_req, m_wr, m_addr, m_dout, reset_n, clk을 입력받고, m_grant, m_din, interrupt를 testbench에 출력한다.

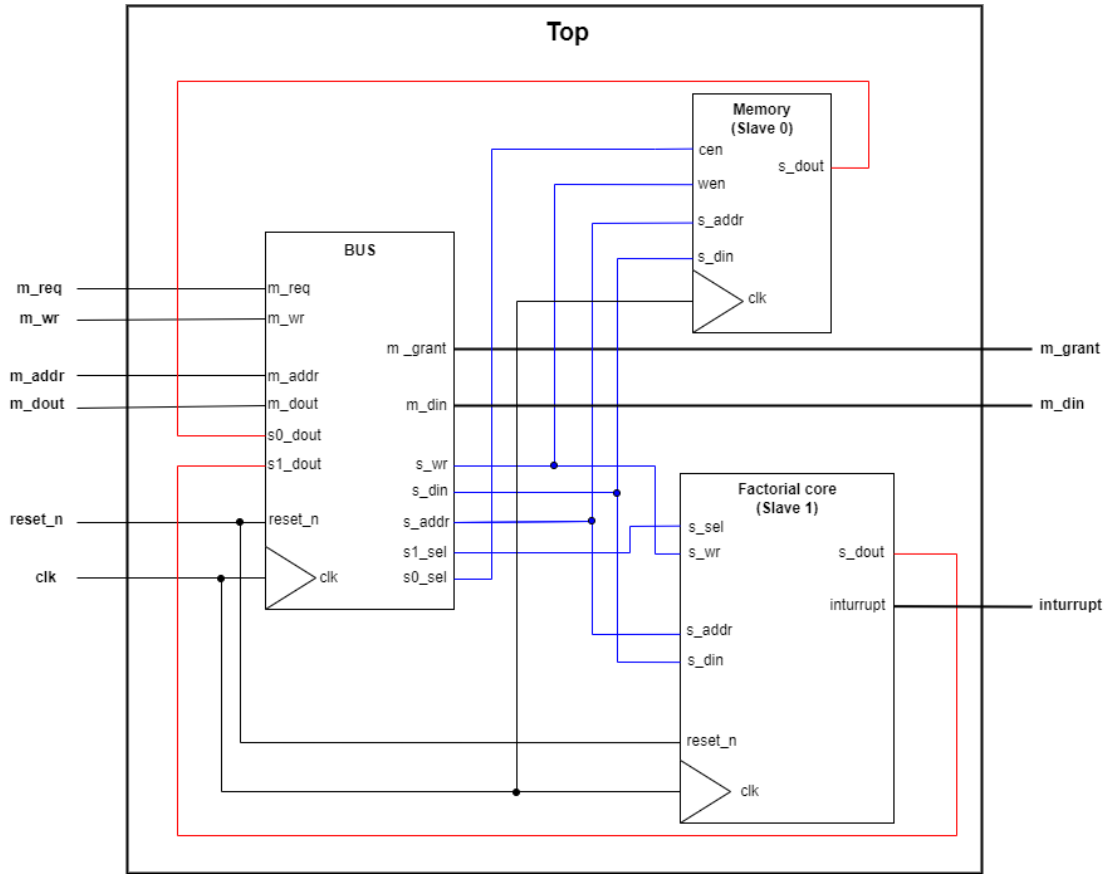
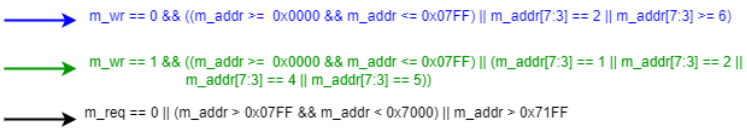


Figure II. Block diagram of Top module

Inputs	m_wr, m_addr, m_req
States	NON_OP, WRITE, READ
Outputs	m_din, m_grant



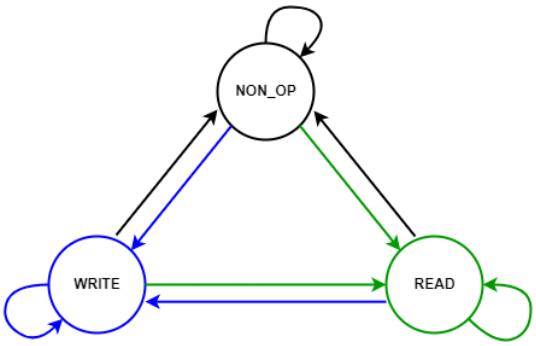


Figure III. FSM of Top

Top module은 크게 3가지의 state를 가지고 있다.

- NON_OP는 m_req = 0 이거나, 어떠한 slave interface도 선택되지 않아 아무런 동작을 수행하지 않는 상태이다.

- WRITE는 m_addr의 값이 쓰기가 가능한 address map region이고, m_wr = 1일 때 m_dout의 값을 해당 register에 작성할 수 있는 상태이다.
- READ는 m_addr의 값이 읽기가 가능한 address map region이고, m_wr = 0일 때, m_din에 bus로부터 입력받은 값을 작성하여 값을 얻는 상태이다.

2. Bus

아래의 Table VI는 BUS module의 input/output port와 그에 대한 bit width, description을 나타낸다. BUS는 top으로부터 master interface의 값들을 입력받는다. 또한 master interface의 값에 따라 slave interface를 선택하고, m_addr에 저장된 값을 읽거나 값을 작성할 수 있으며 m_din의 값을 선택하여 출력할 수 있다. s_wr, s_din, s_addr, s1_sel, s0_sel의 값은 slave 0 또는 slave 1에 입력되는 값이고, s0_dout과 s1_dout은 slave interface에서 bus module로 입력되는 값들이다. 또한, m_grant와 m_din은 master interface로 전달될 출력값이며, 그 외의 값들은 top module에서 입력받은 값들이다. Figure IV는 BUS module의 schematic을 나타낸다.

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

Table VI. Pin description of Bus

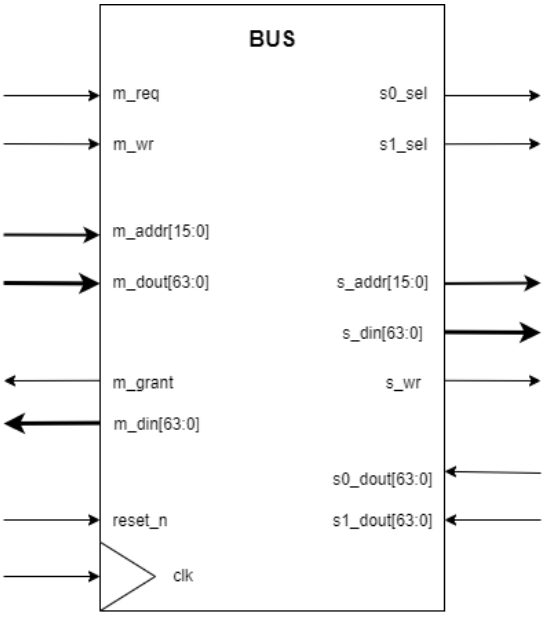


Figure IV. schematic symbol of BUS

Bus module은 내부에 mux, d flip flop, and gate 등으로 구성된다. 아래의 Figure V를 통해 bus module의 block diagram을 확인할 수 있다. Bus module은 간단한 assign 연산, 비교 연산을 수행하기 때문에 내부 logic이 매우 간단하다. Top에서 입력받은 m_wr, m_dout, m_addr, m_req는

각각 s_wr, s_din, s_addre, m_grant에 그대로 할당된다. 그리고 s0_sel, s1_sel의 값에 따라 s0_dout, s1_dout, 64'h0의 값을 m_din에 할당한다. 마지막으로 reset_n, m_req의 값과 m_addr의 범위에 따라 s0_sel과 s1_sel의 값을 정해준다.

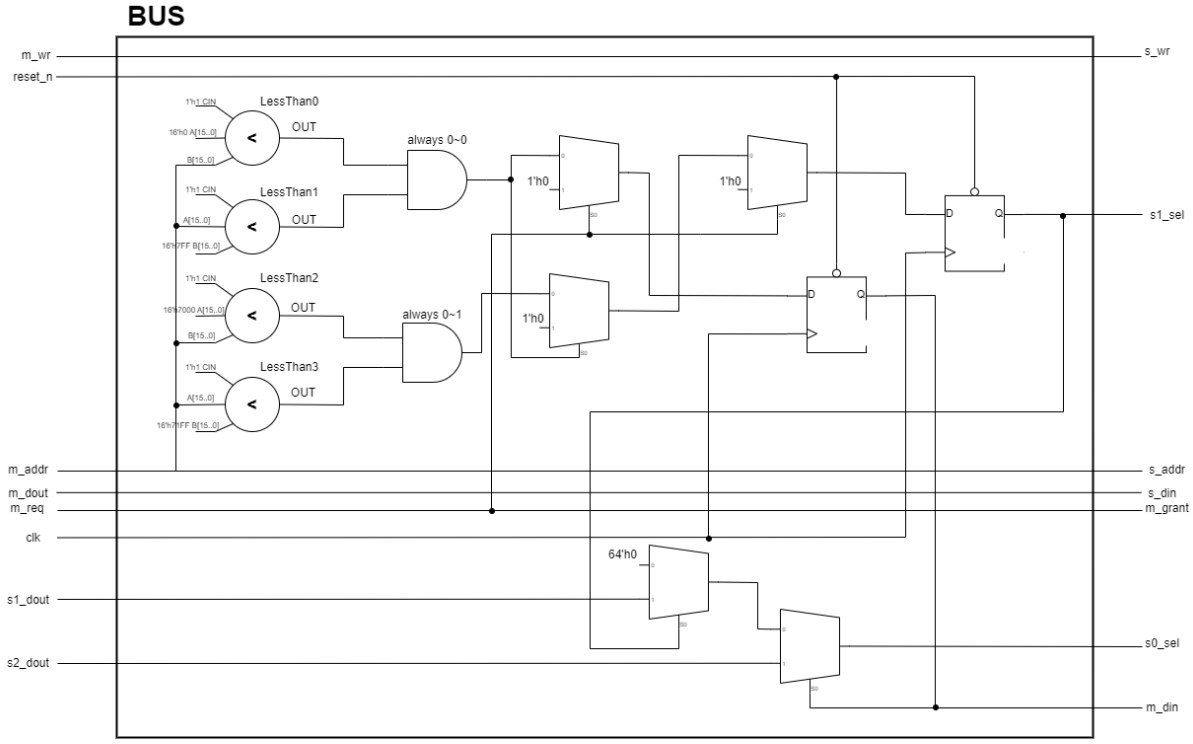


Figure V. Block diagram of BUS

Inputs	reset_n, m_req, m_addr, m_wr
States	INIT, MEM_RD, MEM_WR, FAC, RGT, NON
Outputs	m_grant, s0_sel, s1_sel, s_wr

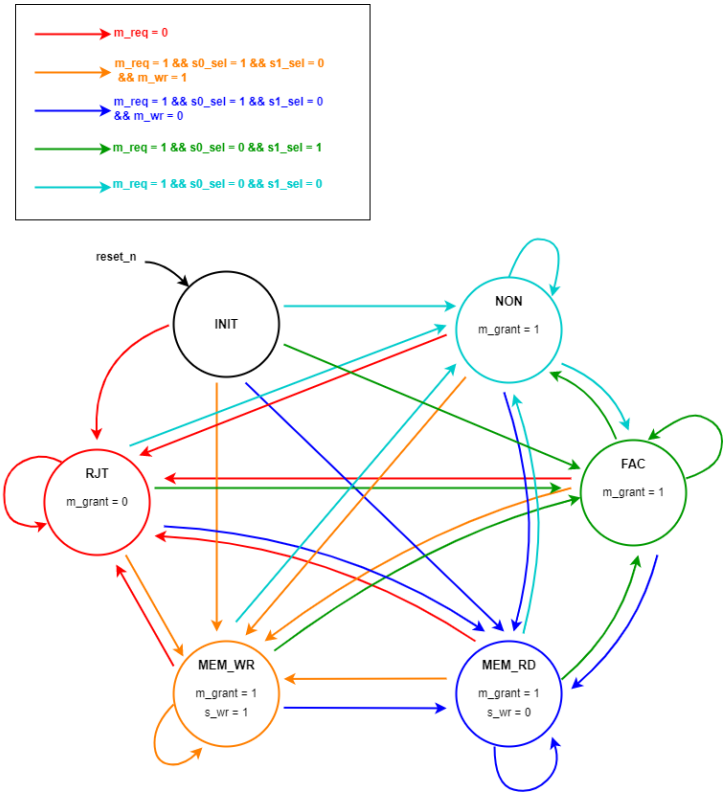


Figure VI. FSM of BUS

Bus module은 INIT, MEM_RD, MEM_WR, FAC, RGT, NON의 6가지 state를 가진다.

- INIT은 Bus module의 초기상태로 s0_sel = 0, s1_sel = 0로 초기화하는 상태이다.
- NON은 m_req = 1이고, m_addr이 Slave0과 Slave1의 memory map region을 넘어 가면 어떠한 slave device도 선택되지 않는 상태이다.
- FAC은 m_req = 0이고, m_addr의 memory map region에 포함되면, Slave1 device (Factorial core)를 선택하는 상태이다.
- MEM_RD는 m_req = 0, m_wr = 0이고, m_addr의 memory map regiondl 0x7000 ~ 0x703F 사이에 포함되면, s_wr = 0으로 설정하고 Slave0 device (Memory)를 선택하는 상태이다.
- MEM_WR은 m_req = 0, m_wr = 1이고, m_addr의 memory map regiondl 0x0000 ~ 0x00FF 사이에 포함되면, s_wr = 1로 설정하고 Slave0 device (Memory)를 선택하는 상태이다.
- RJT은 m_req = 0이면, master에 대한 BUS의 소유권을 거부 (reject)하는 상태이다.

3. Memory

아래의 Table VII는 ram module의 input/output port와 그에 대한 bit width, description을 나타낸다. Memory는 bus로부터 s0_sel (cen), s_wr (wen), clk, s_addr, s_din을 입력받고 그 값에 따라 memory에 데이터를 읽거나 작성하거나 아무런 동작도 수행하지 않는다. Figure VII는 ram module의 schematic을 나타낸다.

Module name	Direction	Port name	Bit width	Description
ram	Input	clk	1-bit	Clock
		cen		Chip enable
		wen		Write enable
		s_addr	8-bit	Address
		s_din	64-bit	Data in
	Output	s_dout		Data out

Table VII. Pin description of ram

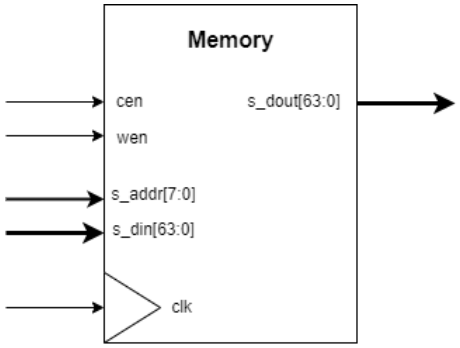


Figure VII. schematic symbol of ram

Ram module은 내부에 mux, ram, d flip-flop으로 구성된다. 아래의 Figure VIII를 통해 ram module의 block diagram을 확인할 수 있다.

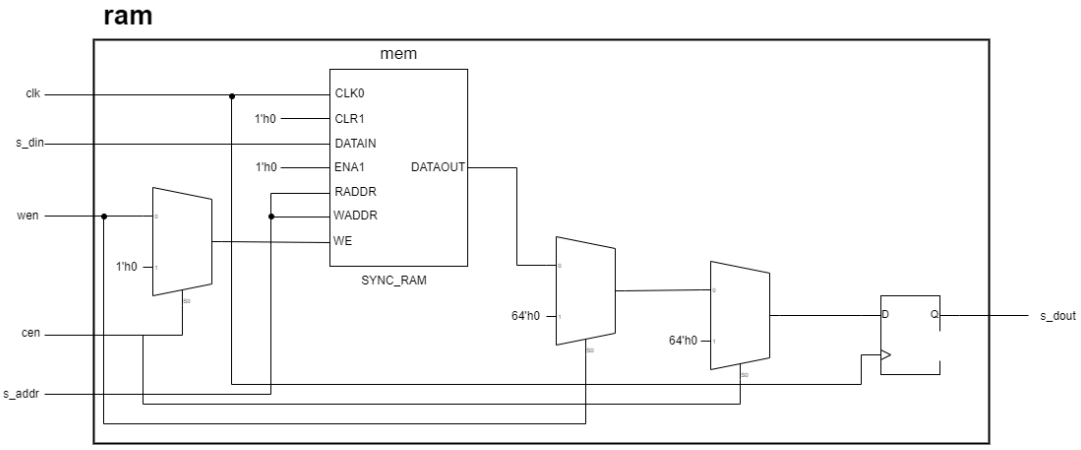


Figure VIII. Block diagram of ram

Inputs	cen, wen, s_addr, s_din
States	INIT, READ, WRITE
Outputs	s_dout

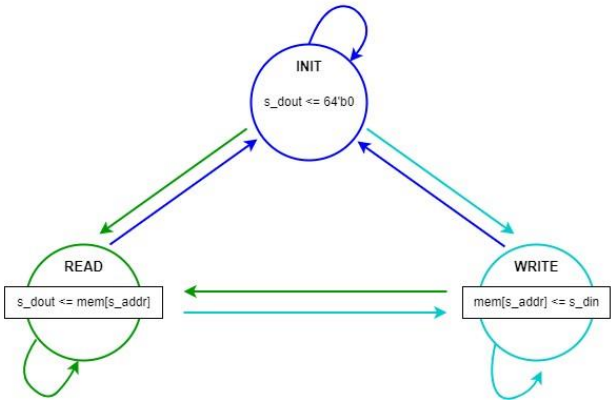


Figure VIIII. FSM of ram

Ram module은 INIT, READ, WRITE의 3가지 state를 가진다.

- INIT은 ram module의 초기상태로 s_dout = 64'b0으로 초기화하고, cen = 0일 때 상태이다.
- REAN는 cen = 1 && wen = 0일 때, s_addr의 주소에 저장된 데이터 값을 읽어 s_dout에 write하는 상태이다.
- WRITE는 cen = 1 && wen = 1일 때, s_addr의 주소가 가리키는 memory에 s_din을 write하는 상태이다.

4. Factorial Core

아래의 Table VIII는 FactoCore, FactoCntr, FactoCore_os, multiplier module의 input/output port와 그에 대한 bit width, description을 나타낸다. FactoCore는 입력받은 값들을 통해 module동작을 제어하고, s_wr, s_addr, s_din을 통해 내부 register의 값을 변화시킨다. 그리고 s_dout과 interrupt를 결과값으로 가진다. Figure X는 FactoCore module의 schematic symbol이다.

Module name	Direction	Port name	Bit width	Description
FactoCore	Input	clk	1-bit	Clock
		reset_n		Active-low reset
		s_sel		Slave interface select signal
		s_wr		Slave interface write/read
		s_addr	16-bit	Slave interface address
		s_din	64-bit	Slave interface data input
	Output	s_dout		Slave interface data output
		interrupt	1-bit	Interrupt out
FactoCntr	Input	operand	64-bit	Operand register
		opstart		Opstart register
		opclear		Opclear register
		opdone		Opdone register
		intrEN		intrEN register
		s_din		Slave interface data input
		s_addr	5-bit	Slave interface address
		state	3-bit	Current state
		s_wr	1-bit	Slave interface write/read
		s_sel		Slave interface select signal
		mul_done		Multiplier done signal
		mul_opstart		Multiplier start signal
		mul_opclear		Multiplier clear signal
		result	128-bit	Result of multiplier
	Output	next_operand	64-bit	Next operand register
		next_opstart		Next opstart register
		next_opclear		Next opclear register
		next_opdone		Next opdone register
		next_intrEN		Next intrEN register
		next_result_h		Next result high register
		next_result_l		Next result low register
		next_state	3-bit	Next state
		interrupt	1-bit	Interrupt signal
		next_mul_opstart		Next multiplier start signal
		next_mul_opclear		Next multiplier clear signal
FactoCore_os	Input	clk		Clock
		reset_n		Active-low reset
		s_wr		Slave interface write/read
		s_sel		Slave interface select signal
		opdone	64-bit	Opdone register
		result_h		Result high register
		result_l		Result low register
		s_addr	5-bit	Slave interface address
	Output	s_dout	64-bit	Slave interface output
		next_mul		Next multiplier input

multiplier	Input	clk	1-bit	Clock
		reset_n		Active-low reset
		op_start		Start signal
		op_clear		Clear signal
		multiplier	64-bit	승수
		multiplicand		피승수
	Output	op_done	1-bit	Done signal
		result	128-bit	Result of multiplication

Table VIII. Pin description of Factorial Core

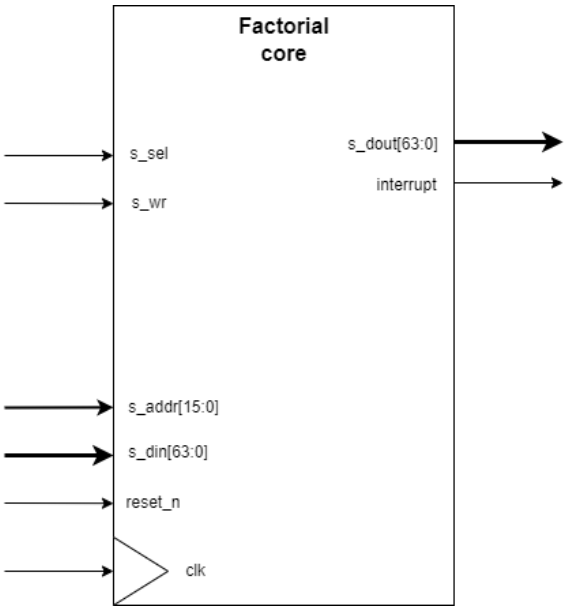


Figure X. schematic symbol of Factorial core

Factorial core는 module 내부에 FactoCntr, FactoCore_os, multiplier를 instance하여 구성하였다. 아래의 Figure VIII를 통해 FactoCore module의 block diagram을 확인할 수 있다.

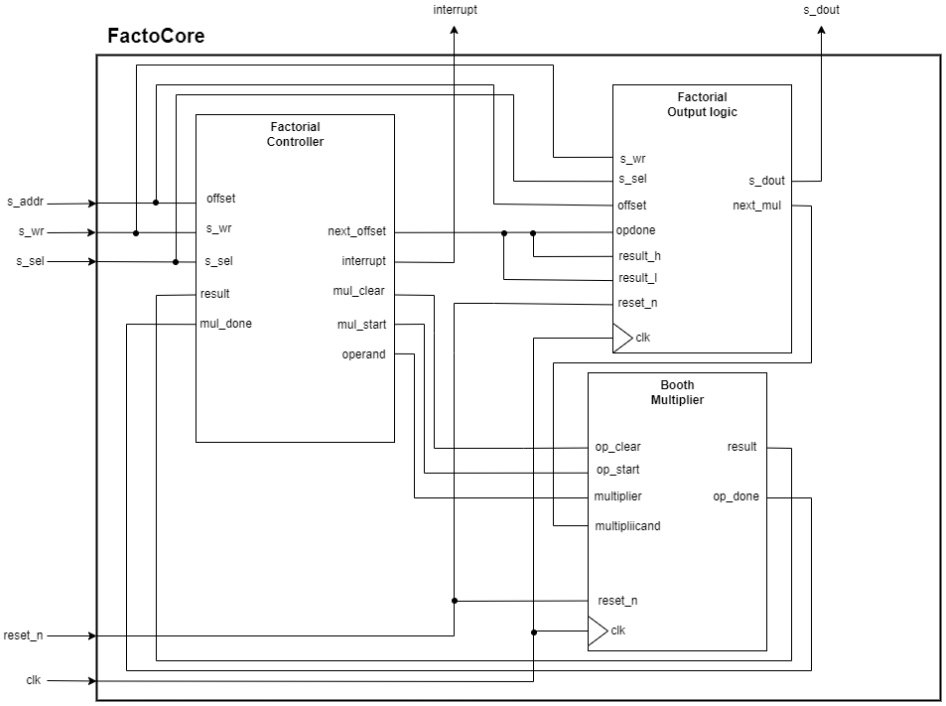


Figure XI. Block diagram of factorial Core

Inputs	s_sel, reset_n, opdone, opclear, mul_done, mul_clear
States	INIT, EXEC, CLEAR, DONE, mul_clear
Outputs	next_operand, next_opstart, next_opdone, next_opclear, next_intrEN, next_result_h, next_result_l, next_state, next_mul_opstart, next_mul_opclear

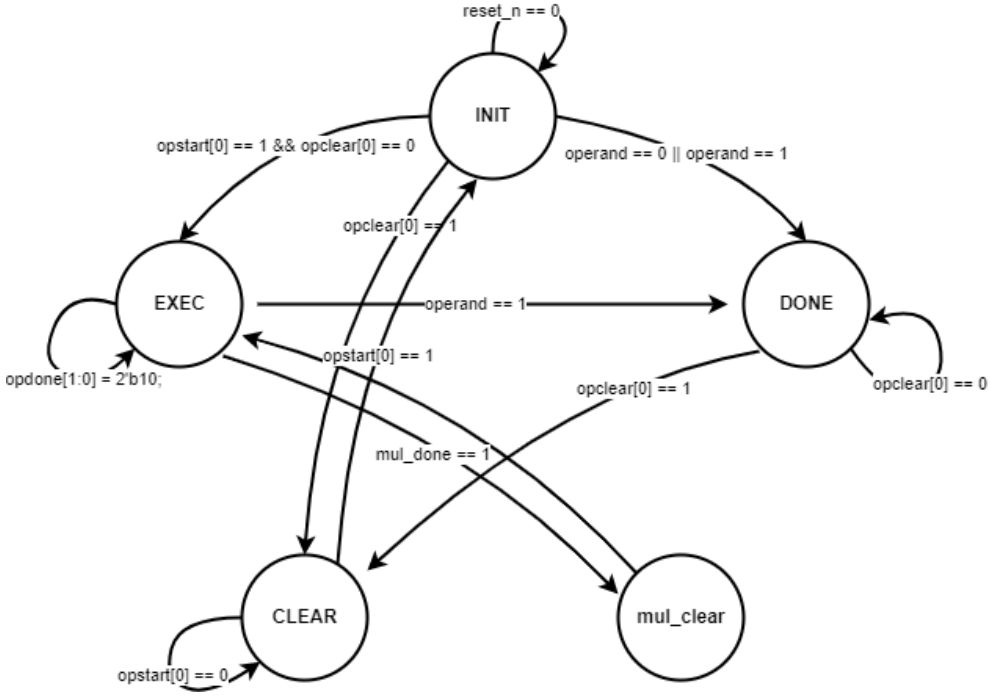


Figure XII. FSM of FactoCore

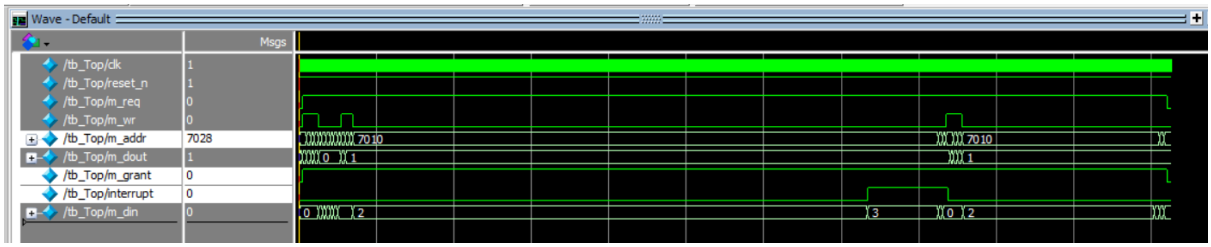
FactoCore module은 INIT, EXEC, CLEAR, DONE, mul_clear의 5가지 state를 가진다.

- INIT은 FactoCore module의 초기상태로 opstart의 값이 x (unknown)값이면 CLEAR state로 넘어가 모든 register의 값들을 default value로 초기화시킨다. opstart[0] = 1이 되면, operand의 값을 비교한다. 만약, operand의 값이 0 또는 1이면 바로 DONE state로 넘어간다. 그렇지 않으면, EXEC state로 넘어간다.
- EXEC는 factorial 연산을 진행중인 상태이다. 만약, multiplier의 연산 완료 신호인 mul_done의 값이 1이 되면, mul_clear 상태로 이동한다. 또한, mul_done = 1이고, operand가 1이되면, DONE 상태로 이동한다.
- CLEAR는 opclear[0] == 1일 때 동작하며, intrEN, operand, opclear를 제외한 모든 register의 값들을 default value로 초기화한다. 만약 opstart[0] = 0이 되면 초기화가 진행된 것이므로 INIT 상태로 돌아간다. 그렇지 않으면 CLEAR state로 돌아간다.
- mul_clear 상태는 multiplier 내부의 모든 값들을 초기화하는 상태이다. 초기화를 수행한 후, 다시 EXEC 상태로 돌아간다.
- DONE은 연산이 완료된 상태로 opclear[0] == 0일때까지 현재 상태를 유지한다. 만약 opclear[0] == 1이 되면 CLEAR state로 이동한다.

IV. Design Verifiaction Strategy and Results

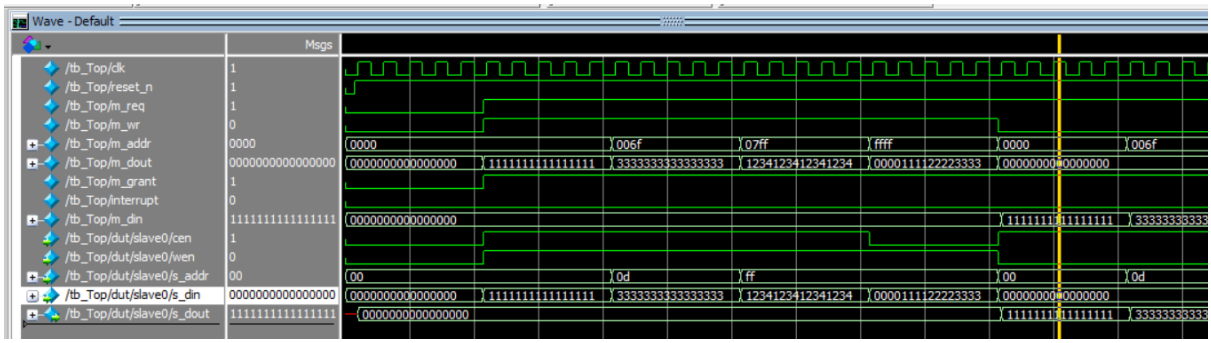
1. Top module

Top module에서는 bus를 통해 slave interface에 값이 제대로 전달이 되는지, 전달된 값으로 원하는 기능을 수행하는가에 대해 검증한다. 아래의 waveform은 Top module의 testbench를 실행한 전체 결과화면이다.

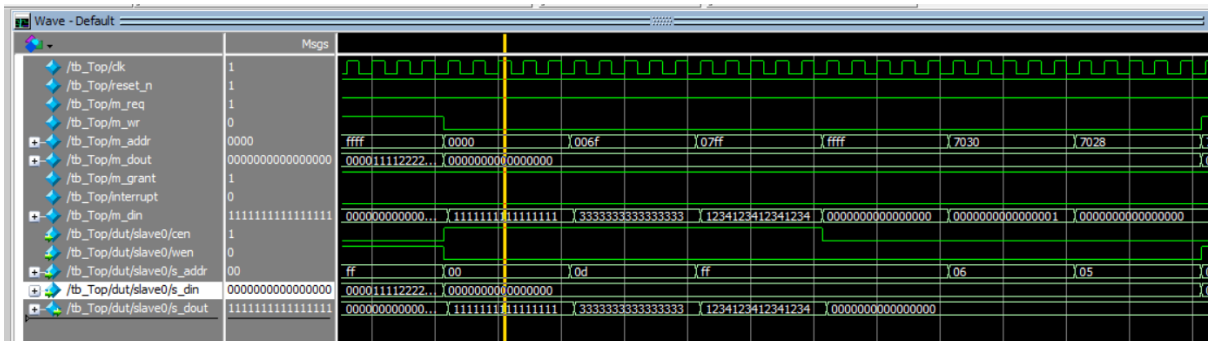


<Slave 0 (Memory)>

아래의 두 결과 화면은 slave0에 대한 동작을 나타낸다.



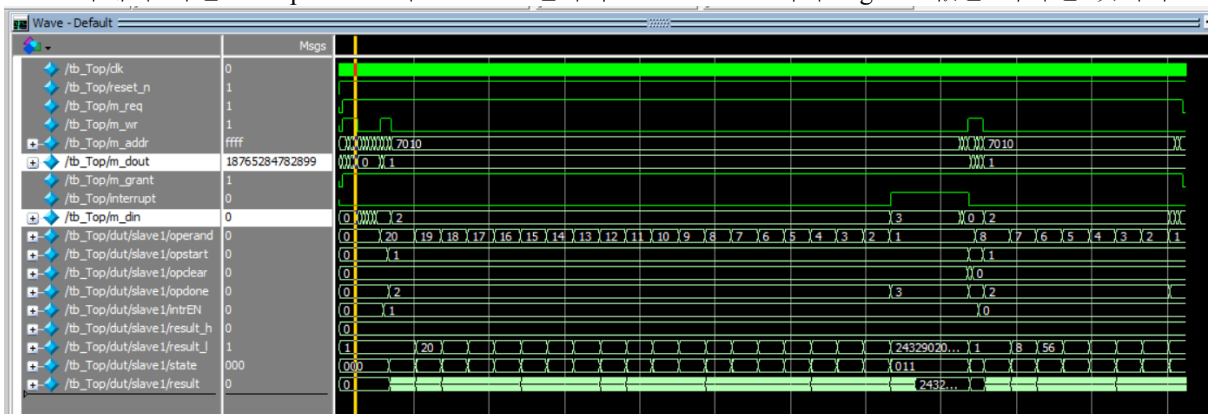
입력받은 m_req의 값이 1이되면 m_grant의 값이 1로 제대로 값이 출력된다. 또한, m_wr = 1일 때, m_addr과 m_dout를 사용하여 memory 내에 값을 저장하는 작업을 수행하였다.

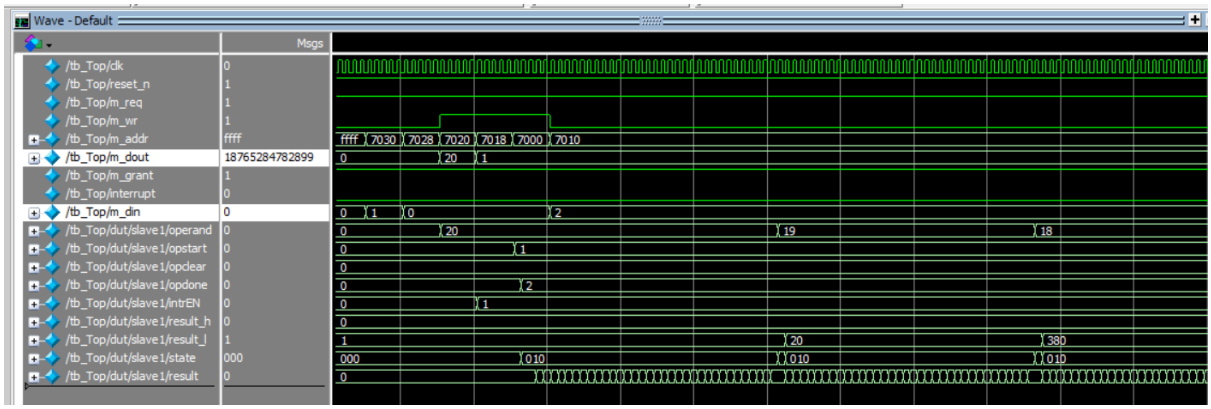


값을 저장한 후, m_wr = 0으로 변경하여 이전에 사용한 주소값 0x0000 ,0x006f, 0x01ff, 0xffff에 저장된 값을 불러오는 작업을 수행하였다. 0x0000, 0x006f, 0x01ff는 모두 slave 0의 주소 범위이므로 m_din에 저장한 값이 제대로 출력되는 것을 확인할 수 있다. 하지만 0xffff의 경우, 범위를 벗어났으므로 0이 출력된다.

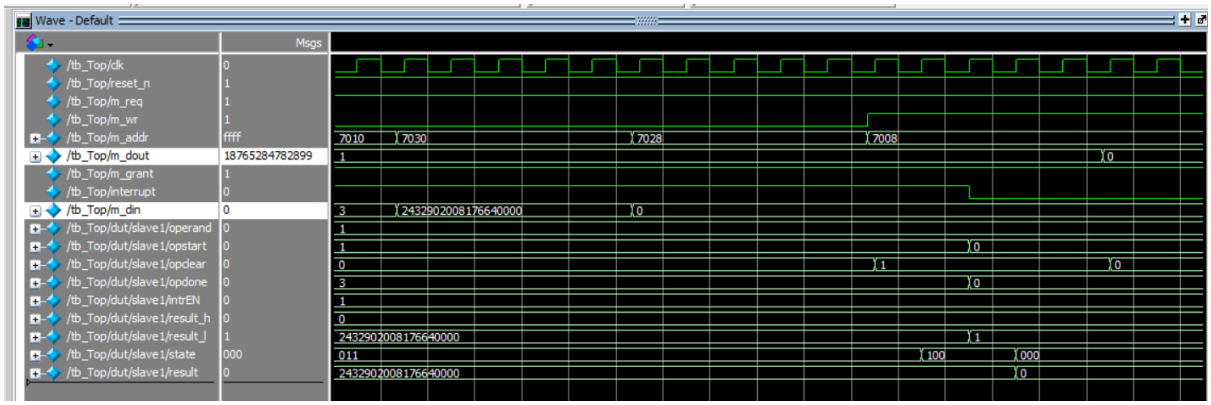
<Slave 1 (Factorial Core)>

아래의 사진은 Top module의 testbench결과와 Factorial core내의 register 값들 나타낸 것이다.

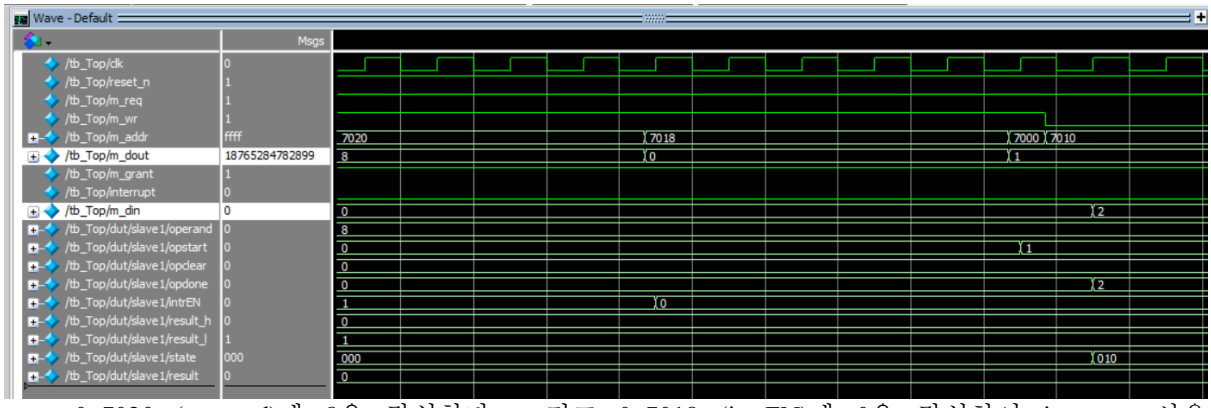




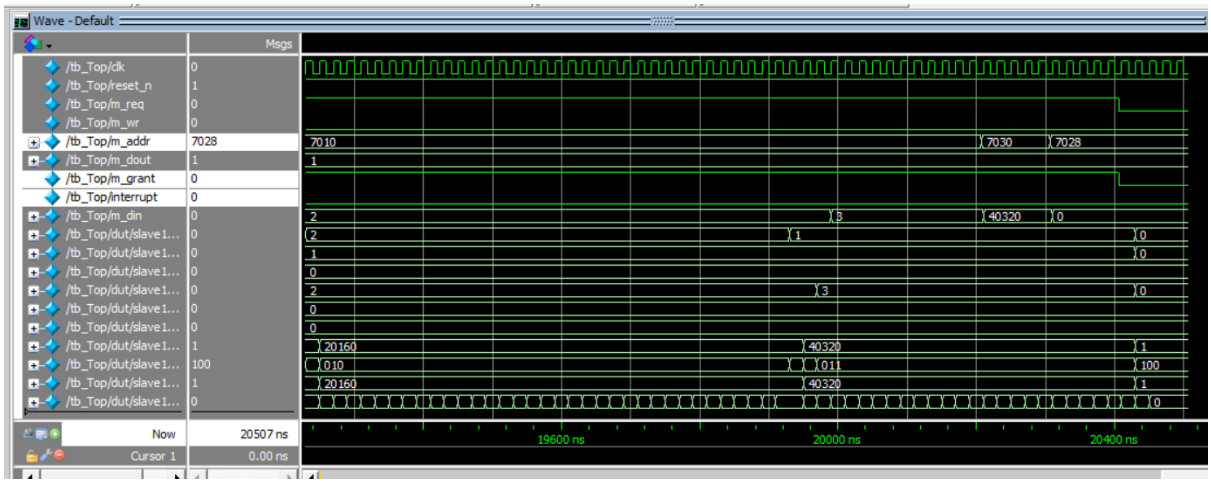
Memory 동작 이후에 factorial core에 접근하기 위해, 해당 주소 범위에 접근한다. 계산 전에 0x7008 (opclear)를 사용하여 register의 값을 초기화한다. 그리고 0x7020 (operand)에 20를 작성한다. 그리고 0x7018 (intrEN)에 1을 작성하여 interrupt 사용을 작성한다. 0x7000 (opstart)에 1을 작성하여 연산을 시작한다. 그리고 0x7010을 적어 opdone의 값을 읽는다.



Interrupt의 값이 1이 될 때, opdone의 값도 3으로 연산이 종료되었음을 알 수 있다. 그리고 0x7030 (result_l)의 값을 읽는다. 20!의 결과값은 2432902008176640000이므로 값이 제대로 m_din에 저장되어있고, 이로 인해 계산이 제대로 수행되었음을 알 수 있다. 그리고 0x7028 (result_h)의 값을 읽고, 이때 m_din의 값이 0으로 출력되었다. 이후에 0x7008 (opclear)에 1을 작성하여 모든 값을 초기화하고 다음 연산을 준비한다.



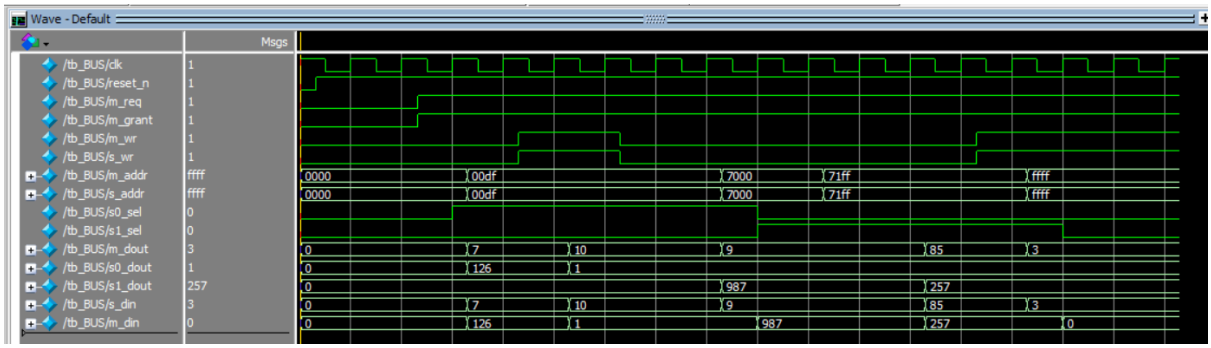
0x7020 (operand)에 8을 작성한다. 그리고 0x7018 (intrEN)에 0을 작성하여 interrupt 사용 여부를 작성한다. 그리고 0x7000으로 opstart에 1을 작성하여 연산을 시작하고, 0x7010을 통해 opdone의 값을 읽어온다. 현재 값은 2로 opdone[1] = 1인 것을 알 수 있고 연산중임을 나타낸다.



Opdone의 값이 3으로, opdone[0] = 1인 것을 알 수 있다. 연산이 종료되었으므로 result_l와 result_h를 읽는다. 8!의 결과값은 40320으로 연산이 제대로 수행되었음을 알 수 있다. 마지막으로 m_req의 값을 0으로 바꾼 뒤 종료하였다.

2. Bus

Bus module은 입력받은 reset_n과 주소값의 범위에 따라 s0_sel과 s1_sel을 변화시키며, 이러한 신호를 통해 m_din의 값이 결정된다. s_wr, s_din, s_addr, m_grant는 master의 값을 그대로 저장되므로 위 모든 사항을 확인하면 된다.

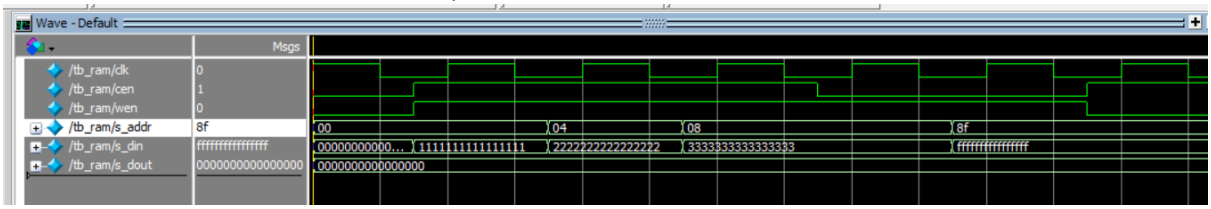


먼저, reset_n의 값이 0일 때, 모든 register의 값들이 0으로 초기화되었다. M_req의 값이 0일 때, bus내에서는 어떠한 동작도 수행하지 않는다. M_req = 1, reset_n = 1일 때, 입력한 주소의 범위에 따라 slave select signal이 변화하는 것을 확인할 수 있다. Slave 1의 주소 범위일 때는 s0_sel의 값이 1이 되고 s1_sel은 0이 되는 것을 확인할 수 있다. 그리고 slave 0의 주소 범위일 때는 s0_sel = 0, s1_sel = 1이 된다. 이 둘의 범위를 벗어난 경우, s0_sel과 s1_sel의 값이 모두 0이 되는 것을 확인할 수 있다.

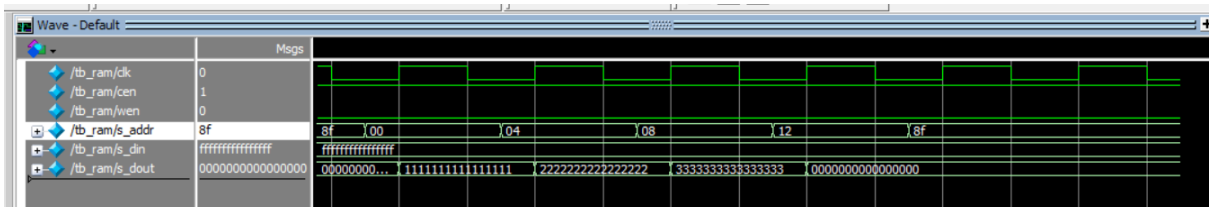
이외에 다른 값들은 assign을 통해 값을 할당해 주었으므로 그래도 slave interface에 입력되는 register에 master로부터 받은 값이 그대로 저장된 것을 확인할 수 있다. 또한, m_din의 값은 slave select signal에 따라 s0_dout, s1_dout, 64'h0의 값이 선택되어 제대로 출력되는 것을 알 수 있다.

3. Memory

Memory 역할을 수행하는 ram을 검증하기 위해 cen = 1일 때와 cen = 0일 때, 각각 값이 작성되는지 되지 않는지 확인하였으며, 저장한 값을 다시 읽는 작업을 수행하였다.



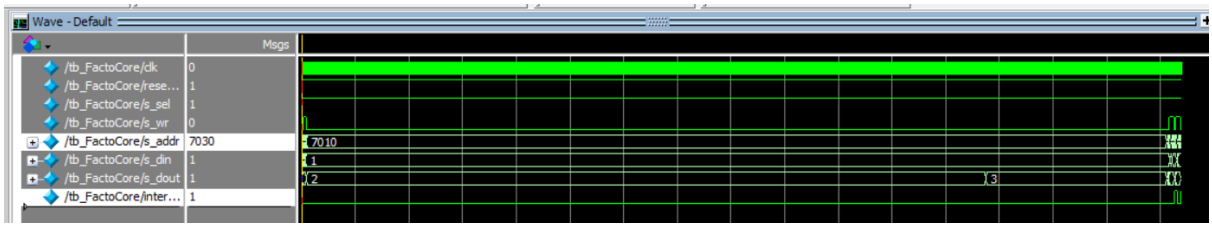
먼저, cen = 1, wen = 1일 때, 0x00, 0x04, 0x08의 주소에 각각 다른 값을 저장하였다. 그리고 0x8F의 주소에 값을 저장할 때는 cen = 0으로 바꾸었다.



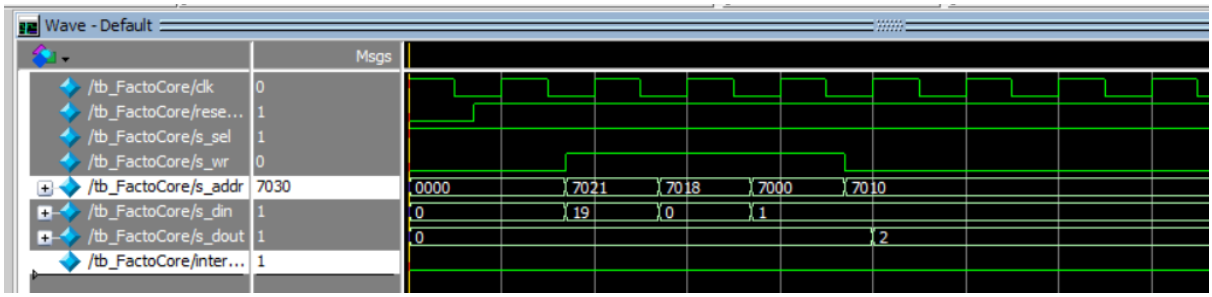
이후에 저장된 값을 읽기 위해 wen = 0으로 바꾼 뒤, 0x00, 0x04, 0x08, 0x12, 0x8f 주소에 저장된 값을 읽어왔다. 0x00, 0x04, 0x08에는 위에서 저장한 값이 제대로 저장되어 있는 것을 s_dout을 통해 확인할 수 있다. 0x12 주소에는 값을 저장한 적이 없으므로 default value인 0이 출력되었다. 마지막으로, 0x8F 주소에 값을 저장할 때는, cen의 값이 0이었으므로 0xFFFFFFFFFFFFFFFF 값이 저장되지 않은 것을 확인할 수 있다.

4. Factorial Core

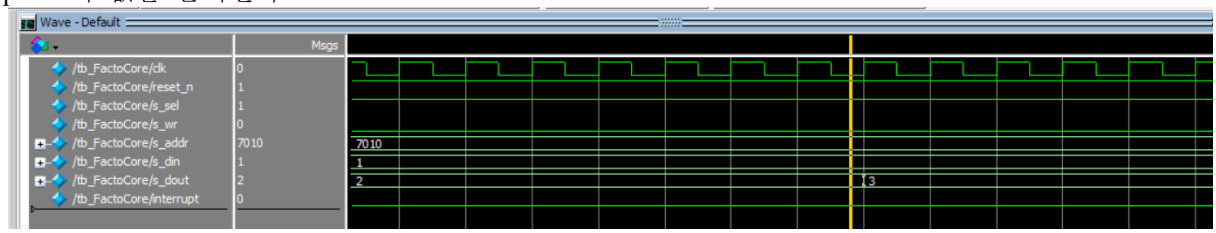
Factorial core는 factorial 연산을 수행하는 module이다. 따라서 각 상태에 따라 register에 값이 제대로 작성되는지, 연산의 결과값이 올바른지, interrupt 신호가 제대로 발생하는지, clear가 제대로 수행되는지 확인해야 한다.



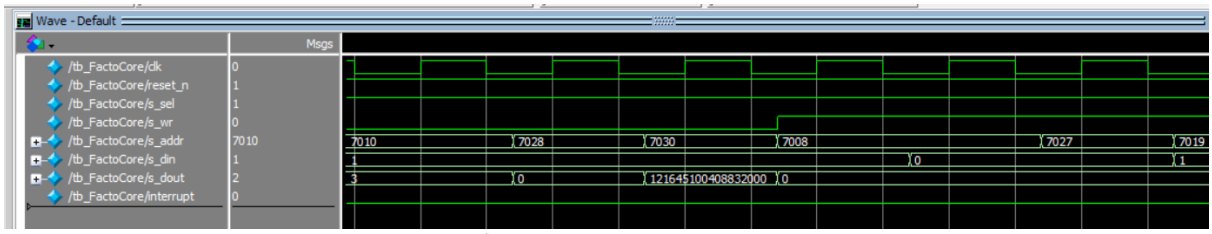
위 waveform은 FactorCore의 testbench 결과 전체 화면이다.



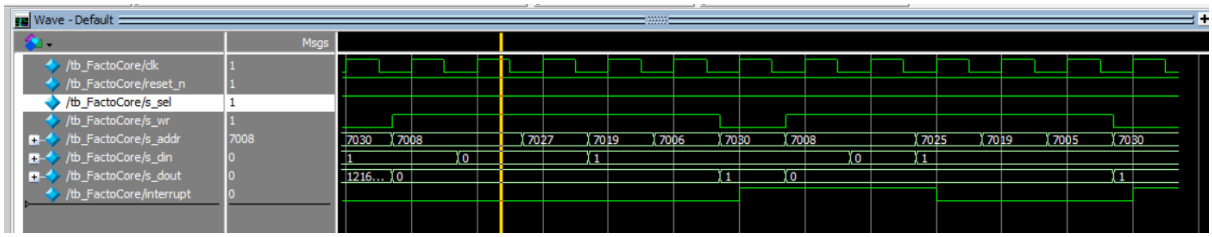
맨 처음, operand에 19를 작성하고 (0x7021), 0x7018에 0을 사용하여 interrupt 사용 여부를 작성한다. 그리고 opstart[0] = 1을 작성하여(0x7000) 연산을 수행하고 0x7010의 값을 읽어 현재 opdone의 값을 출력한다.



위 사진을 통해 약 6300ns에 연산이 종료되고 opdone의 값이 3이 된 것을 확인할 수 있다.



위 waveform은 19! 연산이 종료된 후, 각각 result_h (0x7028)와 result_l (0x7030)를 읽은 값이다. Result_h에는 0이 작성되어 있고, result_l에는 19!의 값인 1216451004088320000이 저장되어 있다. 이후에 opclear를 통해 register의 값을 초기화한 후(0x7008) s_din = 0을 작성하여 초기화를 종료한다. 그 다음 연산을 수행하기 전에 operand에 0을 작성하고 (0x7027), intrEN[0] = 1을 작성한다 (0x7019).



그리고 opstart[0] = 1을 작성하여(0x7006) 연산을 시작하였다. 0!은 연산을 거치지 않고 결과값이 바로 나와야 한다. 따라서 opstart 신호가 들어온 다음 clock의 rising edge에서 interrupt의 값이 1로 연산이 완료되었음을 나타낸다. 결과값은 1로 result_l의 값을 읽었을 때(0x7030), s_dout에 1이 작성된 것을 확인할 수 있다. 다시 opclear를 통해 register의 값을 초기화한 후(0x7008) s_din = 0을 작성하여 초기화를 종료한다. Operand에 다음 연산 값인 1을 작성하고(0x7025), interrupt 사용을 작성하고(0x7019), 연산을 시작한다(0x7005). 1!의 값도 0과 마찬가지로 바로 연산이 되어 inerrupt의 값이 다음 사이클에 1이 된 것을 확인할 수 있다. 또한, Result_l의 값을 읽었을 때(0x7030), 1이 저장된 것으로 보아 연산이 제대로 수행되었다는 것을 알 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Dec 04 20:44:02 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)	8,837 / 41,910 (21 %)
Total registers	16971
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 %)

Figure XIII. Top module flow summary

위 Figure XIII는 Top module의 flow summary이다. 이를 통해 Logic utilization은 8,837으로 21%가 사용되었고, total register는 16,971 개가 사용된 것을 알 수 있다. Total pins는 150으로 30% 사용되었고 나머지 값들은 0이다.

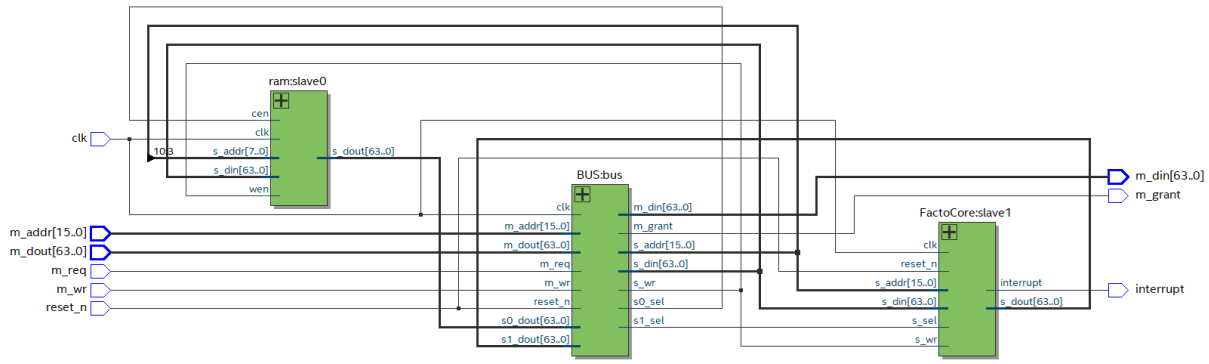
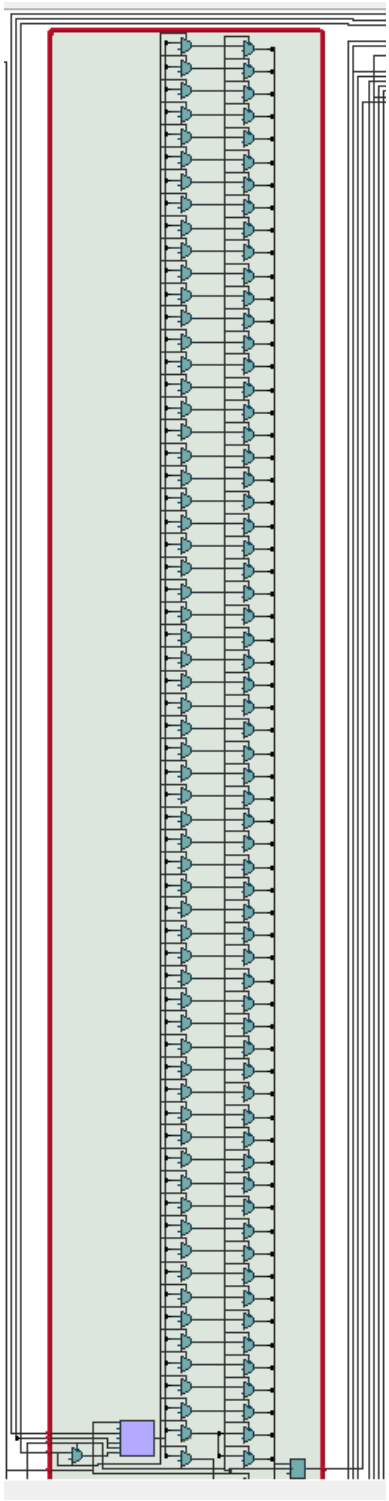
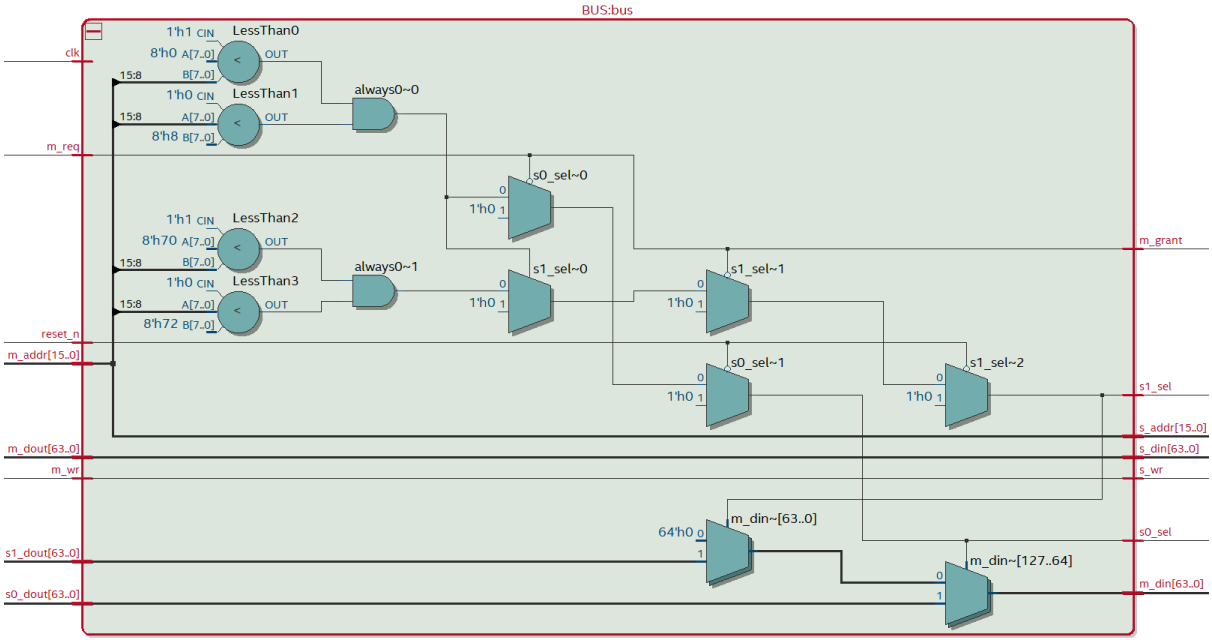


Figure XIV. Top module RTL viewer

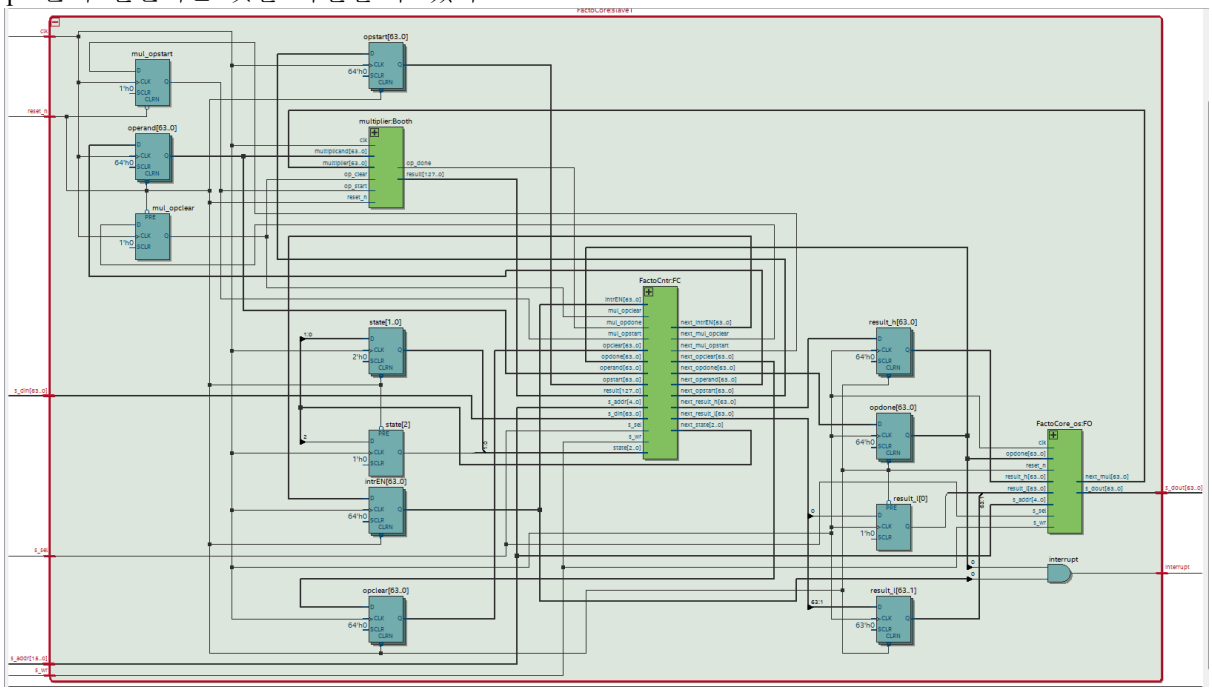
위 Figure XIV는 Top module의 RTL viewer이다. Top module 내에 ram, BUS, FactoCore가 instance되어 있는 것을 확인할 수 있다.



위 그림은 ram의 instance를 확대한 모습이다. 내부에 memory가 선언되어 있는 것을 확인할 수 있다.



위 그림은 BUS의 instance를 확대한 모습이다. 내부에 간단한 연산들을 통해 slave interface의 input들이 연산되는 것을 확인할 수 있다.



위 그림은 FactoCore의 instance를 확대한 모습이다. 내부에 여러 flip-flop들과 FactoCntr, FactoCore_os, multiplier가 instance되어 있는 것을 확인할 수 있다.

V. Throughput

Throughput (처리율)이란 통신 네트워크 상에서 어떠한 노드에서 다른 노드로 전달되는 단위 시간당 데이터 전송으로 처리하는 양을 말한다. 높은 Throughput은 빠른 데이터 전송을 의미하고, 낮은 throughput은 느린 데이터 전송을 의미한다.

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	21.44 MHz	21.44 MHz	FactoCore:slave1|state[0]	
2	22.06 MHz	22.06 MHz	m_addr[10]	
3	52.76 MHz	52.76 MHz	clk	

Figure XV. Maximum frequency report

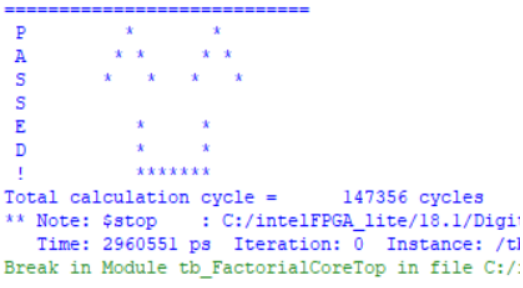


Figure XVI. Result of given testbench

본 프로젝트에서 구현한 factorial computation system의 clock의 Maximum frequency는 52.76MHz이며, 제공된 testbench를 통해 얻은 total calculation cycle은 147,356 cycles이다. Testbench의 개수는 총 20개로 20을 total cycle로 나눈 값은 약 1.357 * 10^{-4}이다. 따라서 **Throughput의 값은 52.76MHz * 1.357 * 10^{-4} = 71.59532 * 10^{-4} 이다.**

VI. Conclusion

1. BUS module에서 s_addr의 값이 0일 때, s0_sel의 값이 1이 되지 않는 문제가 발생하였다. BUS에서 combinational logic을 사용하여 next_s0_sel, next_s1_sel의 값을 할당하고, sequential logic을 사용하여 clock의 rising edge에서 s0_sel, s1_sel에 next_s0_sel, next_s1_sel의 값이 할당되게 구현하였다. 하지만, s_addr의 값이 0일 때, next_s0_sel에 1이 할당되었지만, 바로 다음 clock의 rising edge에서 s_addr의 값이 바뀌어 결국에 s_addr의 값이 0일 때는 slave 0가 선택되지 않아 값이 저장되지 않고, s0_sel의 값도 한 clock 늦게 rising된 것이었다. 따라서 next 변수를 사용하지 않고 바로 sequential logic에서 s0_sel과 s1_sel의 값에 1 또는 0을 할당하는 과정으로 수정하였더니 문제가 해결되었다.
2. Factorial Core에서 register에 값이 작성되지 않는 문제가 발생하였다. 이는 factorial core의 address region을 0x7000 ~ 0x71FF가 아닌, 0xF000 ~ 0xF1FF로 설정하여 발생한 오류였다. 이후에 address region을 수정하니 해결할 수 있었다.
3. Factorial Core을 구현하는데 일주일이라는 기간 동안 많은 어려움을 겪었다. 이는 과제에서 register를 작성하고 읽는 기능을 이해하지 못해 발생한 것이었다. 본 설계는 외부로부터 register들의 값을 받아 factorial core내의 register에 저장하는 것이 목적이었으나, 과제를 제대로 이해하지 못한 탓에 입력된 offset의 값에 따라 s_din의 값을 작성하지 않고 register에 바로 1을 작성하였다. 이 때문에 FactoCore module 내에서 wire와 reg 변수가 꼬이는 문제가 발생하였고, state transition이 제대로 수행되지 않았다. 이후에 offset과 그 값을 write하는 것에 대해 충분히 숙지한 뒤 설계하였더니 빠르게 구현할 수 있었다.
4. Tb_FactoCoreTop.v의 testbench 결과로 “expected cycle exceeded, continue”오류가 발생하였다. 이는 FactoCore module에서 slave 1을 선택하는 신호인 s1_sel의 값이 0이면 계산이 수행되지 않도록 설계하였기 때문이다. 따라서 factorial controller 내에서 s1_sel의 값이 0이 아니여도 계산이 수행되도록 수정하였다. 또한, interrupt의 신호가 opdone의 신호가 11로 설정되고 2 cycle이후에 1로 켜지는 딜레이를 수정하였으며, operand에 0 또는 1이 써질 때, 연산을 수행하지 않고 바로 DONE state로 넘어가도록 수정하여 cycle을 줄였다. 따라서 이후에 제공된 testbench를 실행하니 20개 모두 실행된 것을 확인할 수 있었다. 마지막으로 “Fatal BUS error, test halted” 오류는 원인을 모르겠으나, 위의 오류를 해결하니 해결되었다.
5. 본 프로젝트를 통해 팩토리얼 연산을 수행하는 시스템을 구현하였다. 또한, bus를 통해 memory 내에 값을 작성하거나 값을 읽을 수 있었으며, factorial core 내에 값을 작성하여 연산을 제어하고, 진행 상태 또는 결과값의 상위 64비트, 하위 64비트를 읽을 수 있었다. 해당 프로젝트를 통해 우선, Verilog 프로그래밍을 학습할 수 있다. 이전까지 디지털논리회로2 시간에 배웠던 지식(예, for문, 베릴로그 회로 설계 순서 등)과 추가적으로 공부한 기능(예, 결합연산자 등)을 사용하여 Verilog 언어에 익숙해질 수

있으며 능숙하게 다룰 수 있게 된다. 또한, module들의 인스턴스화를 통해 상위 module을 구성하는 과정을 통해 각 모듈의 구성요소를 이해하고 상호작용하는 능력을 기를 수 있다. 이후에 ALU 내에 팩토리얼 연산 기능을 추가하여 현재의 factorial core 대신에 ALU를 추가하여 XOR, AND 등 논리 연산과 덧셈 및 뺄셈 연산을 추가로 수행할 수 있도록 구현할 수 있을 것이다.

VII.Reference

[1] Timing Latency / https://returnclass.tistory.com/237
[2] Booth multiplication / https://en.wikipedia.org/wiki/Booth's_multiplication_algorithm
[3] Ternary Operator / https://m.blog.naver.com/iixxii1234/221957743928
[4] Verilog Operator / https://hizino.tistory.com/entry/verilog-%EC%97%B0%EC%82%B0%EC%9E%90
[5] 이준환 교수 / Booth Multiplication / 광운대학교 컴퓨터정보공학부 / 2023년
[6] 스루풋 / https://ko.wikipedia.org/wiki/%EC%8A%A4%EB%A3%A8%ED%92%8B