Factorial, Ram, DMAC and Bus

2012722028 장한별

Abstract

이번 프로젝트는 Factorial, Ram, DMAC 그리고 Bus 를 구현하고 이를 Instance 하여 Top module 을 설계하는 프로젝트이다. 전체적인 system은 Factorial operation 을 수행하는 hardware 인 Factorial과 값을 불러오거나, 저장시키는 RAM을 control 하는 Direct Memory Access Controller(DMAC)를 설계하고, 이를 Bus를 통해 data를 전송함으로써 연결하여 검증하는 프로젝트다.

I. Introduction

이번 프로젝트는 Factorial operation 을 수행하는 hardware를 설계하는 것이 목표이다. 먼저, 각 submodule (Factorial_Top, DMAC_Top, ram, BUS) 을 구현하고, 이를 Instance 하여 최종 프로그램을 설계할 수 있도록 한다

주차	일 정	추진 일정			
		1	2	3	4
1	Bus, Memory 구현				
2	DMAC 구현				
3	Factorial 구현 및 예비 검증				
4	예비 검증 후 보완 및 최종 검증				

► 12 월 1 일 : 최종 검증

► 12 월 5 일 : 최종 보고서 제출

II. Project Specification

1. Factorial

Factorial 은 factorial operation 을 수행하는 module 인데, 연산 과정의 곱셈 연산은 Multiplier를 이용해 구현하도록 한다. Multiplier는 multiplier와 multiplier slave 두가지로 나눠서 구현을 해야한다.

Multiplier는 인자로 32bit의multiplicand(피승수) 와 multiplier(승수)를 받게 되고 이를 곱셈 연산을 통해 output으로 내보내게 된다. 이때 사용한 곱셈 방식은 radix-2를 사용하고, radix-2방법은 두 개의 signed 2진 숫자를 2의보수의 원리를 이용하여 곱셈을 진행한다. Multiplier의 x_i 와 x_{i-1}을 1개씩 늘려가면서 비트 수만큼 비교를 하는 것이다.

자세히 말하자면, 일단 처음 Multiplier 의 LSB(least significant bit)의 오른쪽에는 0 비트가 있다고 가정한다. x_i 와 x_{i-1} 를 비교할 때(이때, x_i와 x_{i-1}을 비교하는 숫자는 multiplier 이다), x_i 가 0 이고 x_{i-1}이 0 이면 오른쪽으로 1bit shift 만한다. 이때, 모든 shift 는 RSB 이다. 그리고, x_i 가 이고 x_{i-1}이 1 이면 add 후 오른쪽으로 1bit shift 한다. 이때, add 하는 숫자는 multiplicand 이다. x_i가 1 이고, x_{i-1}이 0 이면, Subtract 후에 shift 한다. 이때 subtract 하는 숫자는 multiplicand 이다. subtract 하는 방법은 2 의보수를 취한 뒤에 add 를 하는 방식으로 하면 된다. 이렇게 진행하다 보면, 비트 수만큼 연산을 하게 되고 결과값을 도출할 수 있게 된다. 이러한 방식으로 Multiplier 를 구현해야한다.

Fatorial 은 S_sel, S_wr, S_address, S_din, clk, reset_n 을 input 으로 받고, S_dout 과 interrupt 를 output 으로 내보낸다. 먼저 S_sel 이 1 일 때, 동작을 하도록 설정을 해야한다. S_sel 의 의미는 slave 로 사용할지 안할지를 결정하는 것이다. 또한 S_wr 은 해당 module 안의 register 를 읽기용으로 사용할 건지 쓰기용으로 사용할 건지 결정하는 신호이며, S_address 는 offset 으로서, 어떤 레지스터를 사용할 것인지 결정해주는 input 이다. S_din 은 사용자가 넣어주는 값이다. register 에 S_din 으로 들어온 값을 저장하는 구조인데 총 10 개의 register 가 존재하는데, OPERATION CLEAR, INTERRUPT ENABLE, OPERATION START, N_FIFO, R_FIFO, N_FIFO COUNT, R_FIFO COUNT, R_FIFO FLAGS, OPERATION DONE 이다. 각 register 들의 특징들은 아래에서 알아보도록 한다.

2. DMAC

DMAC는 Master와 Slave를 동시에 포함하는 Module이다. 간략히 말하면 먼저 testbench가 master이고, DMAC가 sel신호를 받아서 slave 일때 slave의 역할을 하여 값들을 받게 된다. 이때 해당하는 offset address의 값, wr신호에 따라 해당하는 register에 값을 저장하거나 읽을 수 있고, 저장한 값들은 신호에 따라서 fifo에 저장하게 된다. 이후 DMAC가 request를 보내어, master의 권한을 갖게 되면 master의 권한으로 fifo에 저장되어있는 값들을 읽어 들어와서 output으로 내보내게 된다. 이때 각 주소값을 받게 되는데 이 주소값은 ram의 저장 address이다. 자세한 offset register discription은 아래에 상세히 적어 놓았다.

알고리즘을 좀더 상세히 이야기 하자면, 먼저 S_sel을 1로 주고 M_grant를 0으로 주어 slave의 권한으로 값들을 받게 된다. Source address를 입력 받고, 이때 해당하는 offset값과, wr은 1로 설정해줘야 한다. source address를 저장한 후, 그 다음에 destination address, data size를 입력 받고 해당하는 register에 저장하게 된다. 이후 wr_en register에도 1을 입력해 source address, destination address, data size 값들을 fifo에 push하게된다. 이후 opmode도 설정을 해줘서 address값이 linear하게 증가시켜 주어야한다. 이후 operation start register에도 1을 써주어 data를 이동, 복사를 시작하게 된다.

M_req를 1이 되어 master의 권한을 요청한 후, master가 된 DMAC는 fifo에 저장되어있는 값들을 내보내게 된다. 이때 값들은 linear하게 증가되어 나간다. 이동이 모두 완료가 된다면 opdone signal이 켜지게 되고, 이때 Interrupt register에 1을 써주게 되면 module종료 신호인 Interrupt값이 1로 출력되게 된다.

3. Memory

Memory는 address에 기반하여 data를 저장하는 hardware로써, 이번 프로젝트에서는 RAM을 구현하도록 한다. Input으로 cen과 wen이 들어오게 되는데, cen과 wen이 둘다 1이면 인자로 받은address가 가리키는 memory주소값에 인자로 받은 din을 저장하고 dout은 0을 출력한다. cen이 1이고 wen이 0이면 address로 받은 memory에 있는 값을 output으로 내보내게 되고, cen, wen 둘 다 0이면 아무런 동작을 하지 않고 dout은 0을 출력하게 된다.

4. Bus

Bus는 여러 Component들 간의 Data transfer를 원활하게 해주고 전송해주는 Component이다. 먼저 Bus에는 Master, Slave부분이 존재한다. Master는 Bus를 통해서 Slave component중에서 하나를 Master의 address를 통해서 선택 해 줄 수 있다. Bus는 2개의 Master와 4개의 Slave를 연결해주는데, 해당 프로젝트에서 Master가 되는 Component는 Testbench와 DMAC가 된다.

BUS는 2개의master와 4개의 slave로 구성되어 있다. Slave에 해당하는 Component는 Ram1, Ram2, DMAC, Factorial 이다. Bus request란 BUS가 Master에게 BUS에 대한 소유권을 요청하는 것이고 이에 대한 signal이 Grant이다. Bus에게 Grant를 할당 받기 위해서는 각 Master는 Bus를 이용해야 할 시점에 Bus Request 신호를 보내서 Bus에게 Grant를 받고 사용을 해야 한다. 이를 결정해 주기 위해서 필요한 부분이 바로 Bus Arbitrator 이다.

III. Design Details

1. Factorial

► **Pin description**

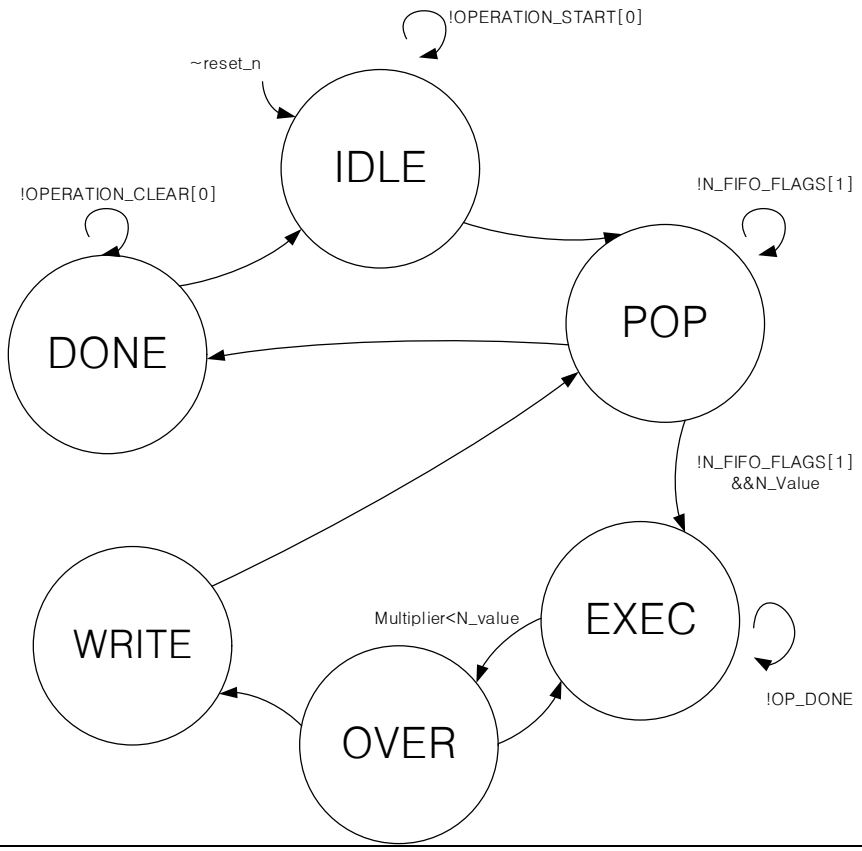
Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	S_sel	(Slave interface) Select
	S_wr	(Slave interface) Write / read
	S_address[7:0]	(Slave interface) Address (factorial 내부에선 해당 address 8bits 중 **하위 4-bit 만을 사용**하여 register를 구분하는 offset 으로 사용한다.)
Outp0ut	S_dout[31:0]	(Slave interface) Data output
	interrupt	Interrupt(Factorial에서 연산이 완료되었을 때 내부 register 값에 따라 interrupt를 발생한다.)

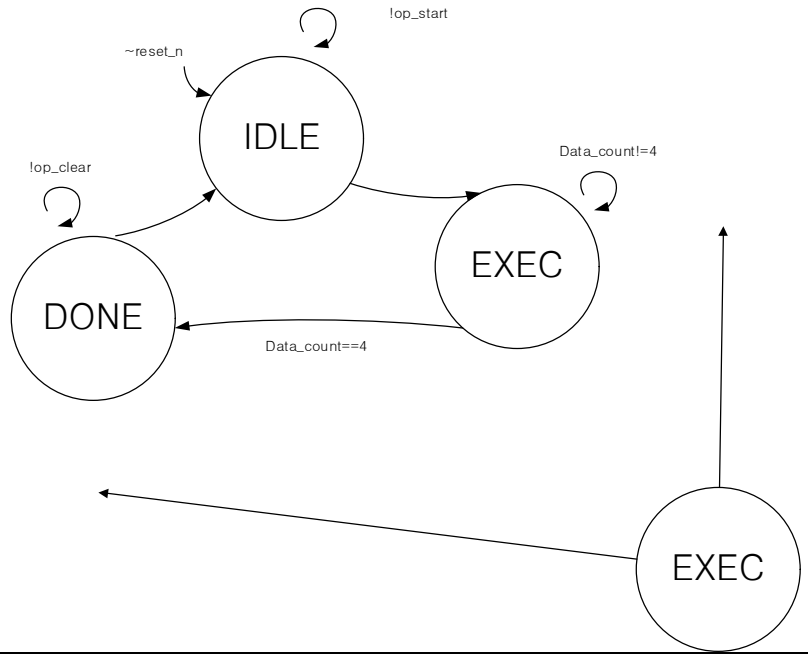
► **Register description**

Offset	Type	Name	Description	Default value
0x0	W	OPERATION CLEAR	해당 register 의 [0]bit 에 1이 써지면 모든 register 의 값이 default value 가 된다. 해당 register의 [31:1] bits 는 reserved 이다.	0x00

0x1	W	INTERRUPT ENABLE	해당 register 의 [0]bit 가 1이면 Factorial 의 OPERATION DONE register 값이 0x01 일 때 (종료되었을 때), interrupt port 에서 1이 출력된다. 해당 register 의 [0]bit 가 0이면 OPERATION DONE register 값이 0x01 일 때, interrupt port 에서 0이 출력된다. 해당 register의 [31:1] bits 는 reserved 이다.	0x00
0x2	W	OPERATION START	해당 register 의 [0] bit 에 1이 써지면 FIFO 에서 N_value 를 POP 하여 factorial 연산이 시작된다. 곱셈 연산 중 해당 register 의 [0]에 1이 써지면 해당 값은 무시된다. 해당 register 의 [31:1] bits 는 reserved 이다.	0x00
0x3	R/W	N_FIFO	DMAC 로부터 전달받은 Factorial 의N_value 에 해당하는 값을 최대 8개 저장하고 있는 register 이다.	0x00
0x4	R/W	R_FIFO	Factorial 연산이 완료된 R에 해당하는 값을 저장하고 있는 register 이다.	0x00
0x5	R/W	N_FIFO COUNT	현재 N_FIFO 에 몇 개의 N_value가 있는지에 대한 정보를 담고있는 register 이다.	0x00
0x6	R/W	N_FIFO FLAGS	현재 N_FIFO 의 상태(empty, full, read_error, write_error) 들의 정보를 갖고 있는 register 이다.	0x00
0x7	R/W	R_FIFO COUNT	현재 R_FIFO 에 몇 개의 result 들이 저장되어 있는지에 대한 정보를 담고있는 register 이다.	0x00
0x8	R/W	R_FIFO FLAGS	해당 R_FIFO 의 상태(empty, full, read_error, write_error) 들의 정보를 갖고있는 register 이다.	0x00
0x9	R	OPERATION DONE	해당 register 의 값을 read 할 때 [0]bit 가 1이면 Factorial 연산을 완료하였음을 나타낸다. 해당 register 의 [7:1] bits는 reserved 이다.	0x00

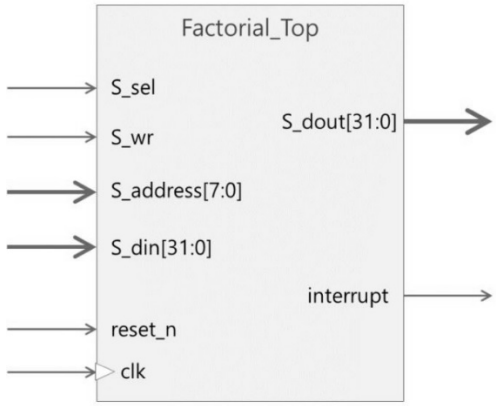
► **State diagram**





위 그림은 Factorial 의 State diagram 이다. 위의 그림은 Factorial 의 전체적인 FSM 이고, 두번째 그림은 factorial operation이다. reset_n 이 0 일때 IDLE 로 시작해서 OPERATION_START 가 주어지지않으면 IDLE 에 계속 머문다. OPERATION_START 가 1이 되면 POP state 로 넘어가고, 이때 N_FIFO_FLAGS 가 0이고 N_Value가 있으면 EXEC state 로 넘어가고, 아니면 DONE 으로 간다.

► **Block diagram**



위 그림은 Factorial_Top 의 Block diagram 이다. S_sel, S_wr, S_address, S_din, reset_n, clk 을 input 으로 받고, S_dout 과 interrupt 를 output 함을 확인할 수 있다.

2. DMAC

► **Pin description**

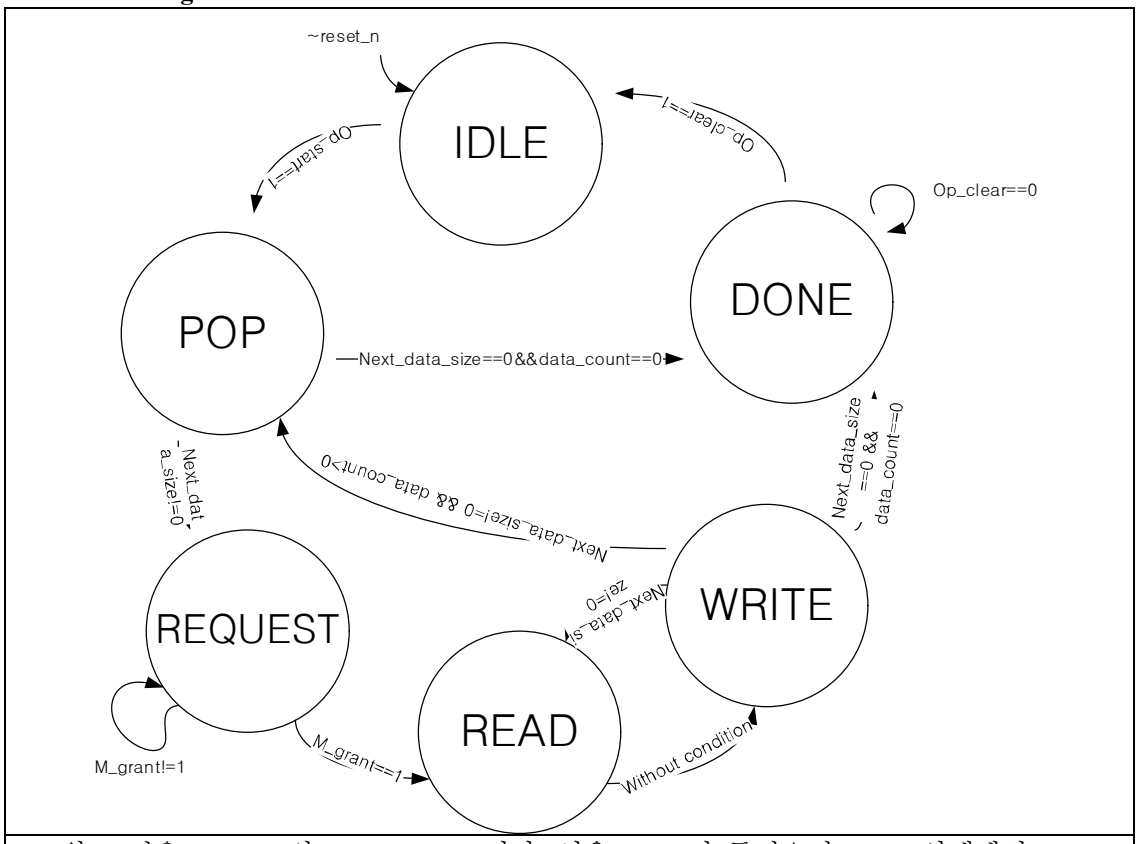
Direction	Port name	Description
Input	Clk	Clock
	reset_n	Active low reset
	M_grant	(Master interface) Grant (Bus 에서 DMAC 가 data를 주고 받는 것을 허락해주는 signal로, 해당 signal이 1인 동안 bus를 통해 다른 slave component를 제어할 수 있다.)
	M_din[31:0]	(Master interface) Data input
	S_sel	(Slave interface) Select
	S_wr	(Slave interface) Write / read
	S_address[7:0]	(Slave interface) Address (DMAC 내부에선 해당 address 8 bits 중 **하위 4-bits만을 사용**하여 register를 구분하는 offset으로 사용한다.)
	S_din[31:0]	(Slave interface) Data input
Output	M_req	(Master interface) Request (DMAC가 bus를 통해 data를 주고 받을 수 있도록 허락해달라고 요청하는 signal이다.)
	M_wr	(Master interface) Write / read
	M_address[7:0]	(Master interface) Address
	M_dout[31:0]	(Master interface) Data output
	S_dout[31:0]	(Slave interface) Data output
	Interrupt	Interrupt (DMAC에서 data의 이동 / 복사가 끝났을 때 내부 register 값에 따라 interrupt를 발생한다.)

► **Register description**

Offset	Type	Name	Description	Default value
0x0	W	OPERATION CLEAR	해당 register 의 [0]bit 에 1이 써지면 모든 register 의 값이 default value 가 된다. 해당 register 의 [31:1] bits 는 reserved 이다.	0x00
0x1	W	OPERATION START	해당 register 의 [0]bit 에 1이 써지면, DMAC 가 data 의 이동/복사를 시작한다. DMAC가 동작 중일 경우에는 해당 register의 [0]bit에 1이 써지는 것은 무시된다. 해당 register 의 [31:1] bits 는 reserved 이다.	0x00
0x2	R/W	INTERRUPT ENABLE	해당 register 의 [0]bit 가 1이면 DMAC 의 OPERATION DONE register 가 1이 되었을 때, interrupt port 에서 1이 출력된다. 0이면 DMAC의 OPERATION DONE register 가 1이 되었을 때, interrupt port 에서 0이 출력된다. 해당 register 의 [31:1] bits 는 reserved 이다.	0x00
0x3	R/W	SOURCE ADDRESS	해당 register 는 DMAC가 data의 이동 / 복사를 수행할 때, read를 시작할 address 를 저장한다. 해당 register 의 [31:8] bits는 reserved 이다.	0x00
0x4	R/W	DESTINATION ADDRESS	해당 register는 DMAC가 data의 이동 / 복사를 수행할 때, write 를 시작할 address를 저장한다. 해당 register 의 [31:8] bits는 reserved이다.	0x00
0x5	W	PUSH DESCRIPTOR	해당 register의 [0]bit에 1이 써지면, SOURCE ADDRESS register, DESTINATION ADDRESS register, DATA SIZE register 의 값 (descriptor 에 대한 information) 을 내부 FIFO 에 저장한다. 이때 최대 8개의 descriptor 에 대한 information 을 저장할 수 있다. 해당 register 의 [31:1] bits 는 reserved이다.	0x00
0x6	R	DESCRIPTOR SIZE	내부 memory(또는 FIFO)에 저장되어 있는 descriptor 의 수를 나타낸다. 최대 8개의 descriptor 가 저장될 수 있기 때문에 [3:0] bits 만을 사용하며, [31:4] bits는 reserved 이다.	0x00
0x7	R/W	DATA SIZE	해당 register는 DMAC 가 data의 이동/ 복사를 수행할 때, 전송할 data의 크기를 저장한다. 단위는 4bytes 이다. 해당 register의 [31:8] bits는 reserved이다.	0x00
			Zero initialize 기능, source address와 destination	

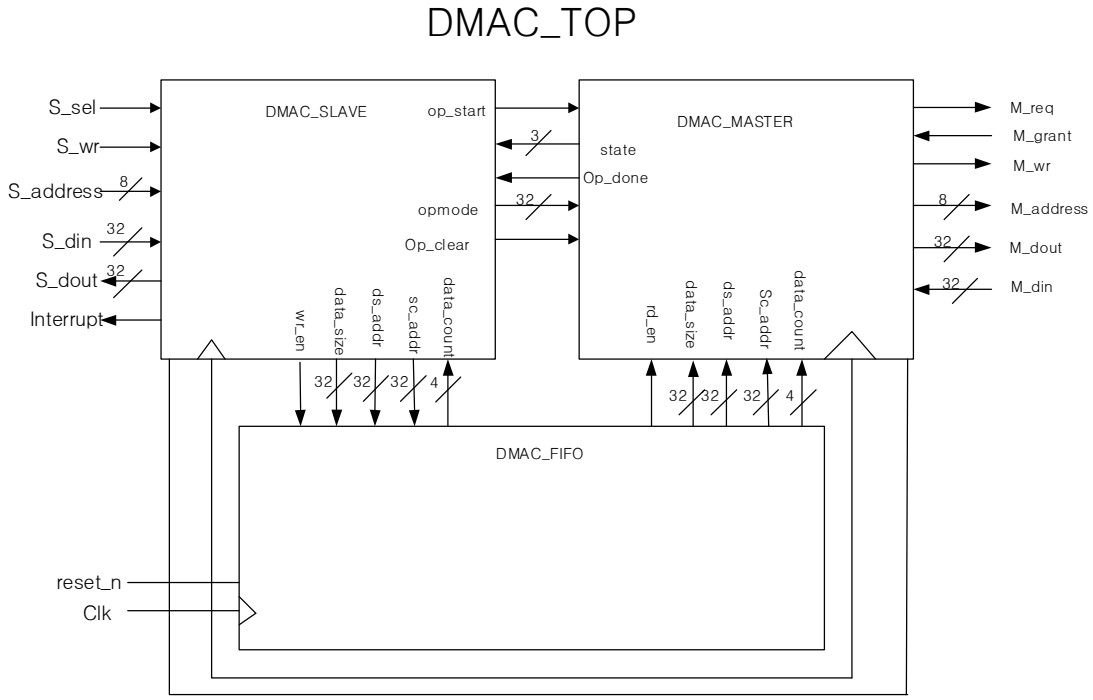
0x8	R/W	OPERATION MODE	address 의 increment mode 가 있다. opmode[0]: source address increment mode (0-none, 1-linear) opmode[1]: destination address increment mode (0-none, 1-linear) opmode[2]: zero initialize 해당 register의 [31:3]bits는 reserved 이다.	0x00
0x9	R	OPERATION DONE	해당 register의 값을 read할 때 [0]bit 가 1이면 DMAC가 data의 이동 / 복사를 완료하였음을 나타낸다. 해당 register의 [7:1] bits는 reserved 이다.	0x00

► **State diagram**



위 그림은 DMAC 의 state diagram 이다. 처음 reset 이 들어오면 IDLE 상태에서 시작하게 되고 op_start signal 을 받으면 POP 으로 넘어가게 된다. POP이 되면 FIFO 에 저장되어있는 값들을 pop 하게 되고, next_data_size 가 0이 아니라면 REQUEST 로 간다. REQUEST 에서는 M_grant 신호가 1이면 READ state로 가게 되고, M_grant 신호를 받지 못한다면 계속 REQUEST 에 머물러 grant 를 요청해야한다. READ 로 가면 별다른 신호 없이 WRITE 를 하게 되고, WRITE 에서 data_count 가 0보다 크다면 POP 으로 넘어가게되고 위와 같은 과정을 반복하게 된다. 이때, WRITE 에서 data_count 가 0이면 더 이상 읽을 명령어가 없다는 뜻이므로, DONE 상태로 가게 된다. 이때 DONE 상태에서 op_clear 에 1이 들어오면 다시 IDLE 상태로 돌아가게 되어 모든 값을 초기화 시켜준다.

► **Block diagram**



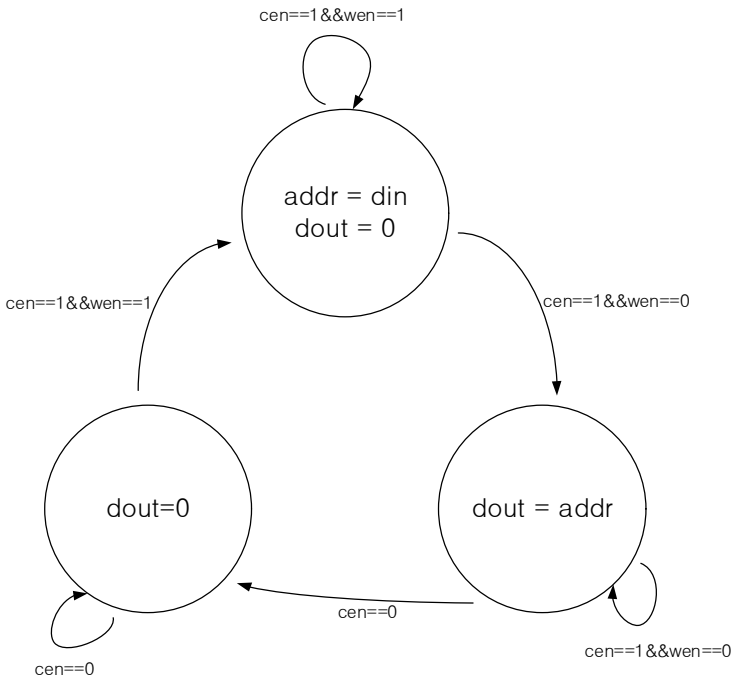
위 그림은 DMAC_Top 의 Block diagram 이다. 처음 DMAC 가 slave 로서, S_din 으로 값들을 받아 해당하는 S_address 의 offset 에 따라서 해당 register 에 값을 저장한다. 이후, wr_en 에 1을 write 하여 SOURCE ADDRESS register, DESTINATION ADDRESS register, DATA SIZE register 의 값을 FIFO 에 보내고 저장한다. 그 후, Bus 에 request 한 뒤, master 권한을 얻으면 register 에 있는 값들을 bus를 통해 내보내게 된다.

3. Memory

► **Pin description**

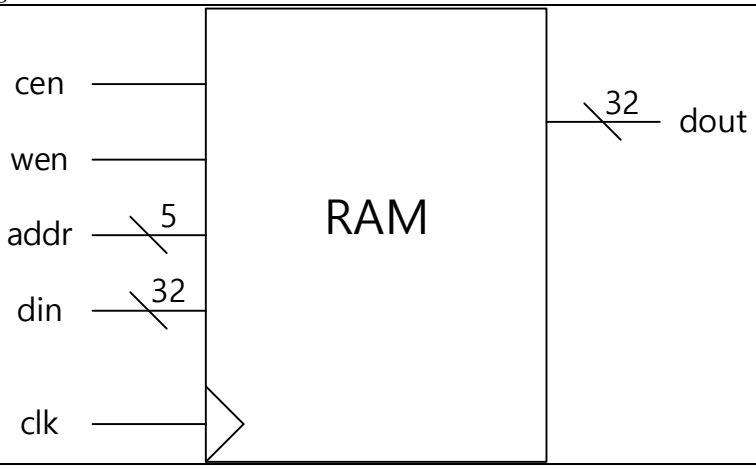
Direction	Port name	Desctiption
Input	clk	Clock
	cen	Chip enable
	wen	Write enable
	addr[4:0]	Address
	din[31:0]	Data in
Output	dout[31:0]	Data out

► **State diagram**



위 그림은 Memory 의 State diagram 이다. cen 은 memory 자체의 작동을 제어하는 기능이고, wen 은 memory 에서 write 의 기능을 제어한다. 이때, cen 과 wen 이 모두 1로 set 된다면, din 으로 들어온 값을 지정 addr 에 저장하고, 출력으로는 0을 출력하게 된다. cen 이 1로 set 이 되고, wen이 0으로 clear 된다면, write enable 신호가 없으니, 따로 주소값에 저장은 하지 않고, 지정 addr 에 있는 값을 dout 으로 출력하게 된다. cen 이 0으로 clear 된다면 memory 자체가 작동하지 않는 다는 의미이므로 어떠한 저장도 하지 않고, dout 으로 0을 출력하게 된다.

► **Block diagram**



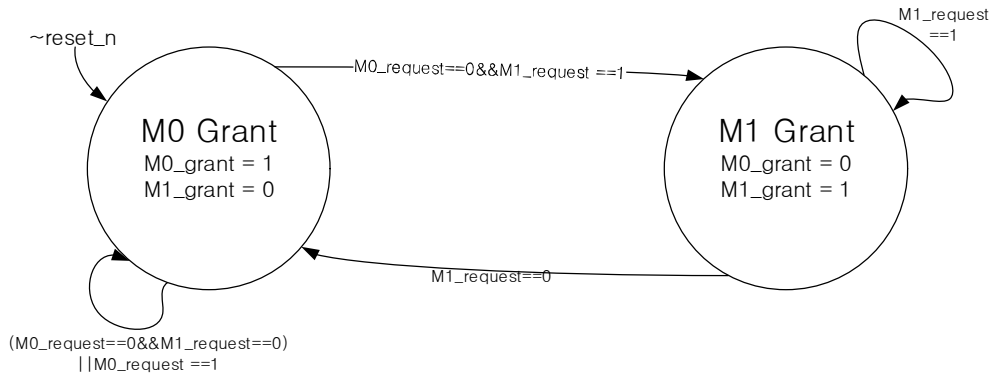
위 그림은 memory의 block diagram이다. input으로써 cen, wen, addr, din, clk이 들어가고 이때, addr 은 5비트이고, din은 32bit이다. 또한 output으로는 32bit의 dout이 나오게 된다. Cen의 뜻은 chip enable이란 뜻으로 해당 ram이 동작을 할 것인지, 안할 것인지 판별하는 역할이다. Wen은 write enable로써 해당 signal이 1로 들어오면 address가 가리키는 memory에 입력값 din을 write하는 구조로 설계하였다. 이때 cen, wen모두 1이면 addr이 가리키는 memory에 입력값 din을 write하고 0을 출력하도록 설계하였고, cen이 1이고 wen이 0이면 din을 를 신경 쓰지않고, addr이 가리키는 memory 값을 dout에 출력하고, cen이 0 이면 wen은 신경 쓰지않고, dout은 그냥 0을 출력하도록 설계했다.

4. Bus

► Pin description

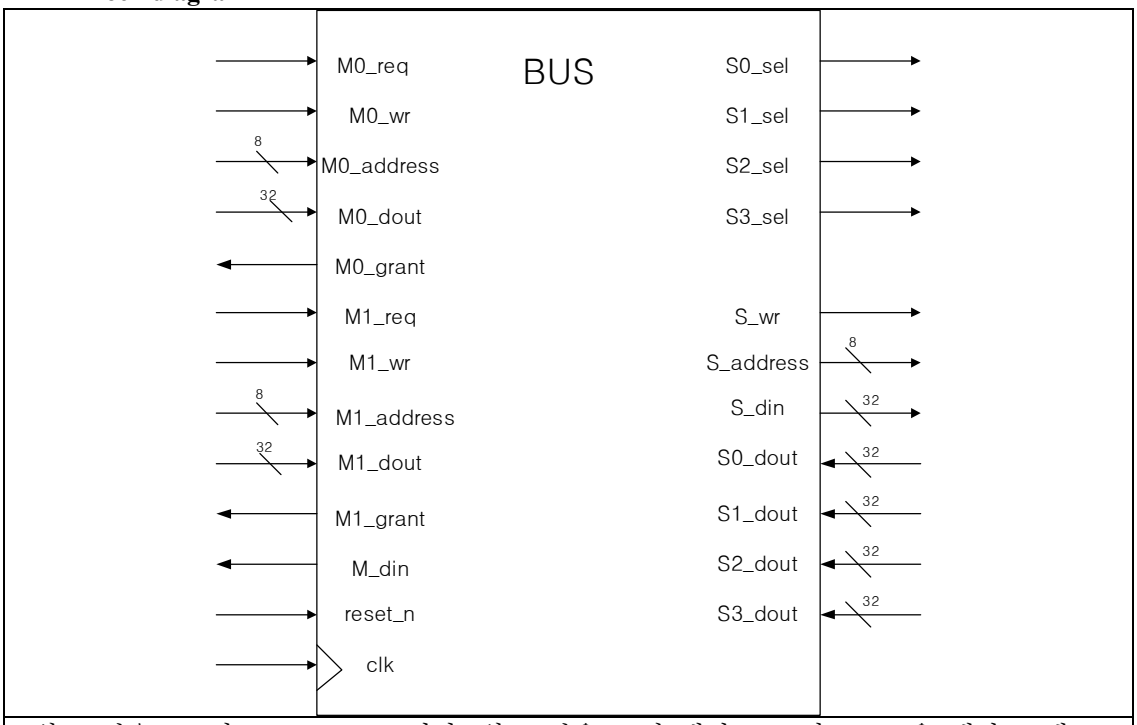
Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	M0_req	Master 0 request
	M0_wr	Master 0 write/read
	M0_address[7:0]	Mater 0 address
	M0_dout[31:0]	Master 0 data output
	M1_req	Master 1 request
	M1_wr	Master 1 write / read
	M1_address[7:0]	Master 1 address
	M1_dout[31:0]	Master 1 data out
	S0_dout[31:0]	Slave 0 data out
	S1_dout[31:0]	Slave 1 data out
	S2_dout[31:0]	Slave 2 data out
	S3_dout[31:0]	Slave 3 data out
Output	M0_grant	Master 0 grant
	M1_grant	Master 1 grant
	M_din[31:0]	Master data input
	S0_sel	Slave 0 select
	S1_sel	Slave 1 select
	S2_sel	Slave 2 select
	S3_sel	Slave 3 select
	S_address[7:0]	Slave address
	S_wr	Slave write / read
	S_din[31:0]	Slave data input

► **State diagram**



위 그림은 Bus arbitrator 의 State diagram이다. Reset이 clear 되면, M0 Grant로 초기화 되고, 해당 state일 때 request값이 M0_grant는 M0_grant가 1, M1_grant가 0고, M1_grant는 M0_grant가 0, M1_grant가 1 인 것으로 설정하였다. 또한, 각 state에서 해당 request조건일 때 다음 state로 넘어가거나, 현재 state를 유지하도록 설계하였다.

► **Block diagram**



위 그림은 Bus의 Block diagram이다. 위 그림을 보면 해당 input과 output을 해당 bit에 맞춰서 표기하였고, master0과 master1의 request신호를 받아서 grant를 결정하게 되고, 해당 grant에 따라서 입력된 값을 설정하게 된다. M1_address의 상위 4bit을 보고 Slave0부터 Slave3에서 하나를 결정하게 되고, 그 sel을 선택하게 된다. 이후 master grant는 output으로 나가게 된다.

5. TOP

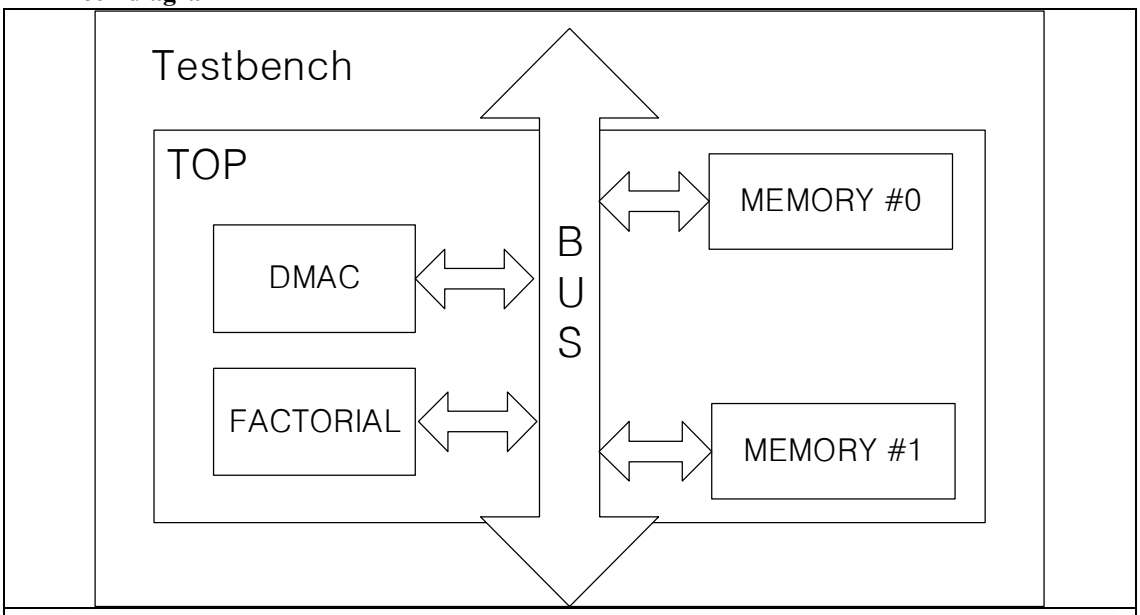
► **Memory map**

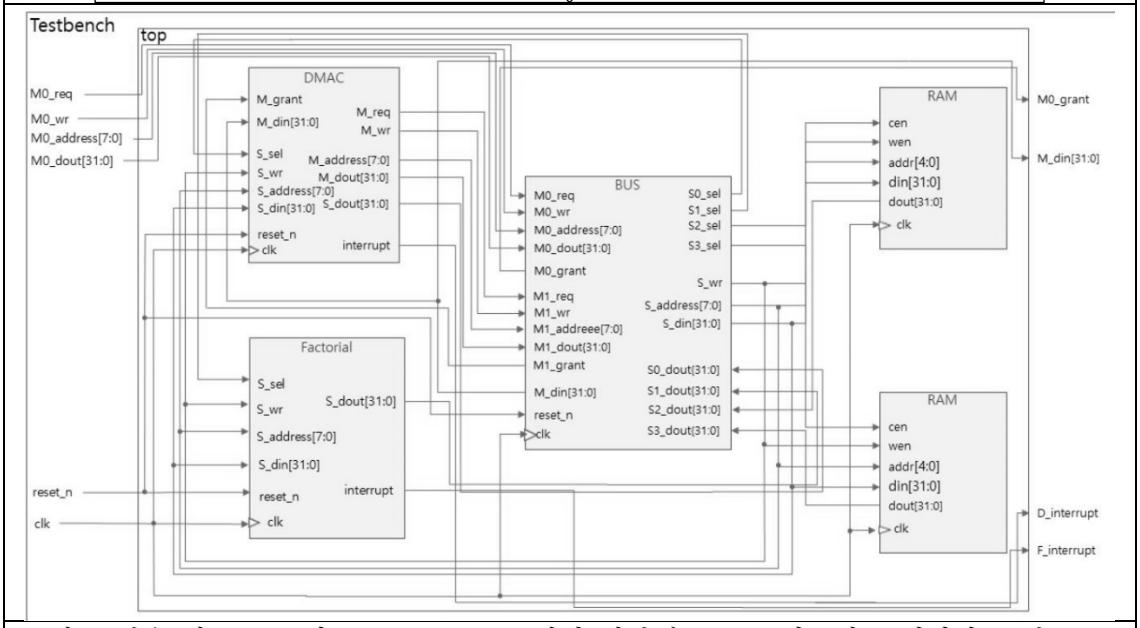
Component	Address region
0x00~0x0F	DMAC
0x10~0x1F	FACTORIAL
0x20~0x3F	RAM(N_VALUE)
0x40~0x5F	RAM(result)

► **Pin description**

Direction	Pop name	Description
Input	clk	Clock
	reset_n	Active low reset
	M0_req	Master request
	M0_wr	Master write / read
	M0_address[7:0]	Master address
	M0_dout[31:0]	Master data output
Output	M0_grant	Master 0 grant
	F_interrupt	FIFO interrupt
	D_interrupt	Dmac interrupt
	M_din[31:0]	Master data input

► **Block diagram**





위 그림은 각 module이 Top-level module에서 어떻게 instance되는지 도식화한 그림(block diagram)이다. DMAC, FACTORIAL, BUS, MEMORY가 instance되어 있고, 각각의 위치에서 화살표로 추상적으로 그려놓았다. 가장 상위에는 testbench가 있다. 이 testbench 즉 processor의 역할을 하는 module이 DMAC에게 소유권을 주게 된다. 그럼 master역할을 하는 DMAC가 bus를 통해서 memory에서 값을 불러들여 FACTORIAL을 통해 factorial operation을 하고 다시 bus를 통해 memory에 저장하게 된다.

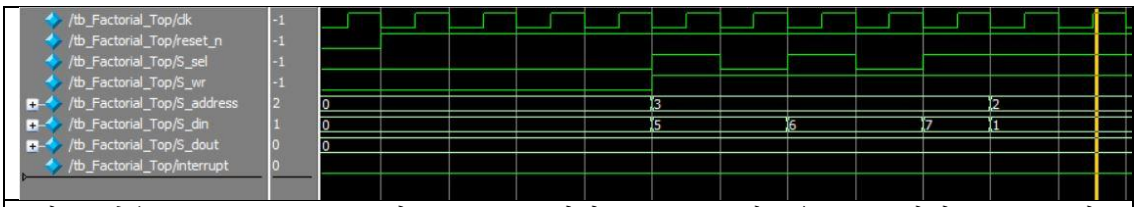
IV. Design Verifiaction Strategy and Results

1. Factorial

► 검증 전략

Factorial 을 검증하기 위해선 먼저 계산기로 factorial operation 이 제대로 수행이 되는지 확인한 뒤, slave를 구현하여 값들이 제대로 이동이 되고, 정확한 타이밍에 값들이 이동이 되고, register file에 값이 정확하게 입력이 되었는지 fifo에 값들이 제대로 들어갔는지 등을 확인했다. 또한 register file 에 있는 값들이 output 으로 제대로 출력이 순차적으로 정확한 타이밍에 나오는지 확인했다.

► **Waveform**



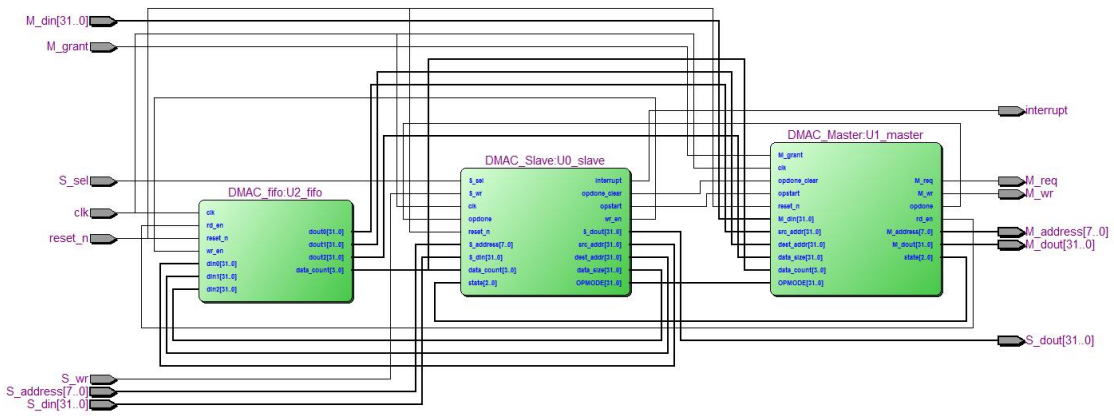
위 그림은 Factorial testbench 의 Waveform 이다. S_address 에 3을 input 시켜 N_FIFO 에 S_din 에 5, 6, 7을 넣어준 후, R_FIFO 에서 input 시킨 값들이 올바르게 factorial operation 연산을 수행했음을 확인할 수 있다.

2. DMAC

► 검증 전략

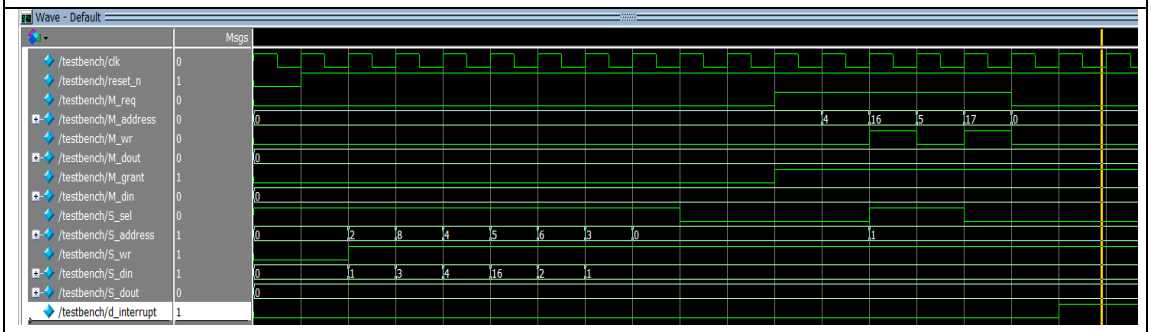
먼저 각 DMAC의 각 register에서 해당하는 값들을 제대로 저장하는지 확인 하였고, 이를 fifo에 동시에 제대로 push하는지, fifo에서 master로 제대로 값을 보내주는지, 또한 해당 opmode에 linear하게 address값이 증가하는지, 모든 동작이 끝나고 제대로 Interrupt신호가 뜨는지 등을 검증하였고, 해당 Clk주기를 맞추기 위하여 많은 노력을 하였다.

► **RTL viewer**



위 그림은 DMAC_TOP의 RTL Viwer이다. 해당 port들이 정확히 연결되어 있음을 확인할 수 있다.

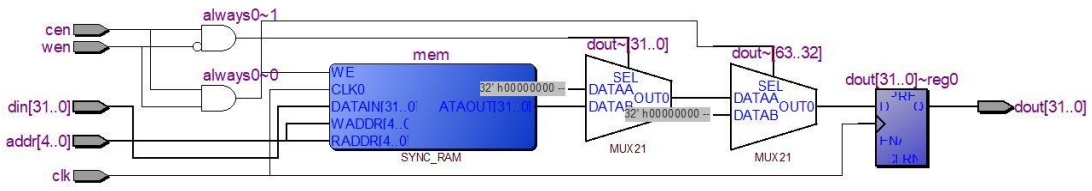
► **Waveform**



위 그림은 DMAC의 testbench이다. 처음 slave권한으로 값을 입력 받는다. 이후 fifo에 push 하였고, address와 destination address가 linear하게 data size만큼 증가하게 설정하였다. 이후 req신호를 보내어 master 권한으로 M_address를 보게되면 linear하게 증가 한 것을 확인 할 수 있고 모든 동작을 끝내고, Interrupt가 set된 것을 확인할 수 있다.

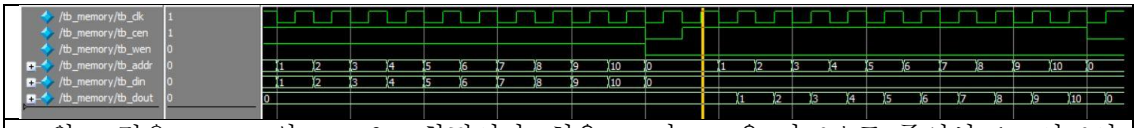
3. Memory

► **RTL viewer**



위 그림은 memory 의 RTL Viewer 이다. 정확하게 연결되어있음을 확인할 수 있다.

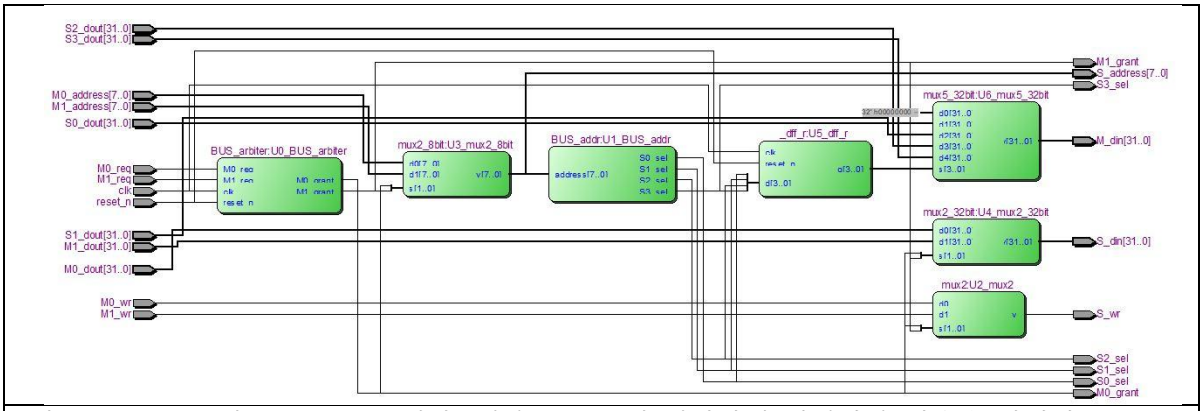
► **Waveform**



위 그림은 memory의 waveform화면이다. 처음 cen과 wen을 다 0으로 주어서 dout이 0이 출력되고 있고, 그 다음 cen과 wen을 모두 1로 넣어주어서 해당하는 address에 din값을 넣어주고 있고 dout에는 넣어주는 din값이 그대로 출력되고 있음을 확인할 수 있다. 또한 중간쯤 넘어가서 0부터 10까지 메모리에 넣어준 뒤 wen과 cen값을 모두 0으로 바꾸어 초기화 한 뒤에 cen만 1로 넣어서 해당하는 address에 저장되어있는 값들을 출력하고있다. Address에 1부터 10까지의 메모리주소를 순차적으로 호출하고있고, 이 주소 값을 받고 clk이 rising edge일 때 해당하는 메모리주소에 저장되어있는 값이 출력됨을 확인할 수 있다.

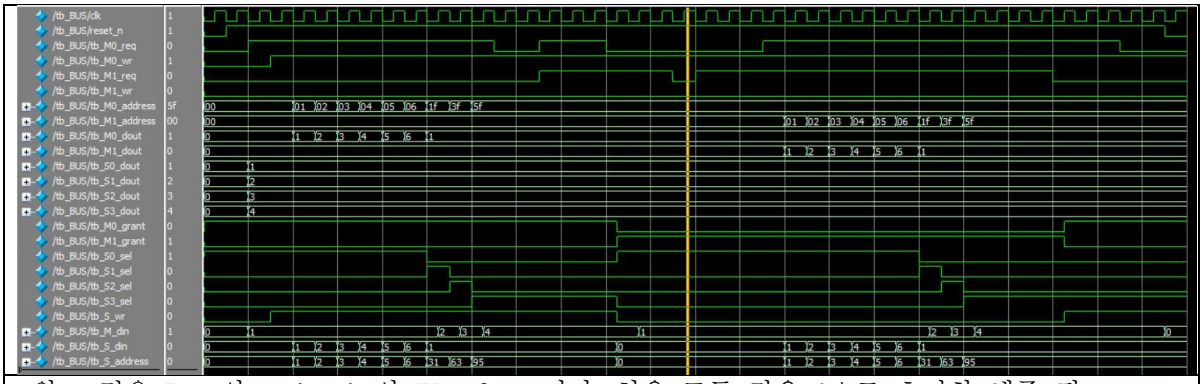
4. Bus

► **RTL viewer**



위 그림은 Bus 의 RTL viewer 이다. 해당 port 들이 정확하게 연결되어 있음을 확인할 수 있다.

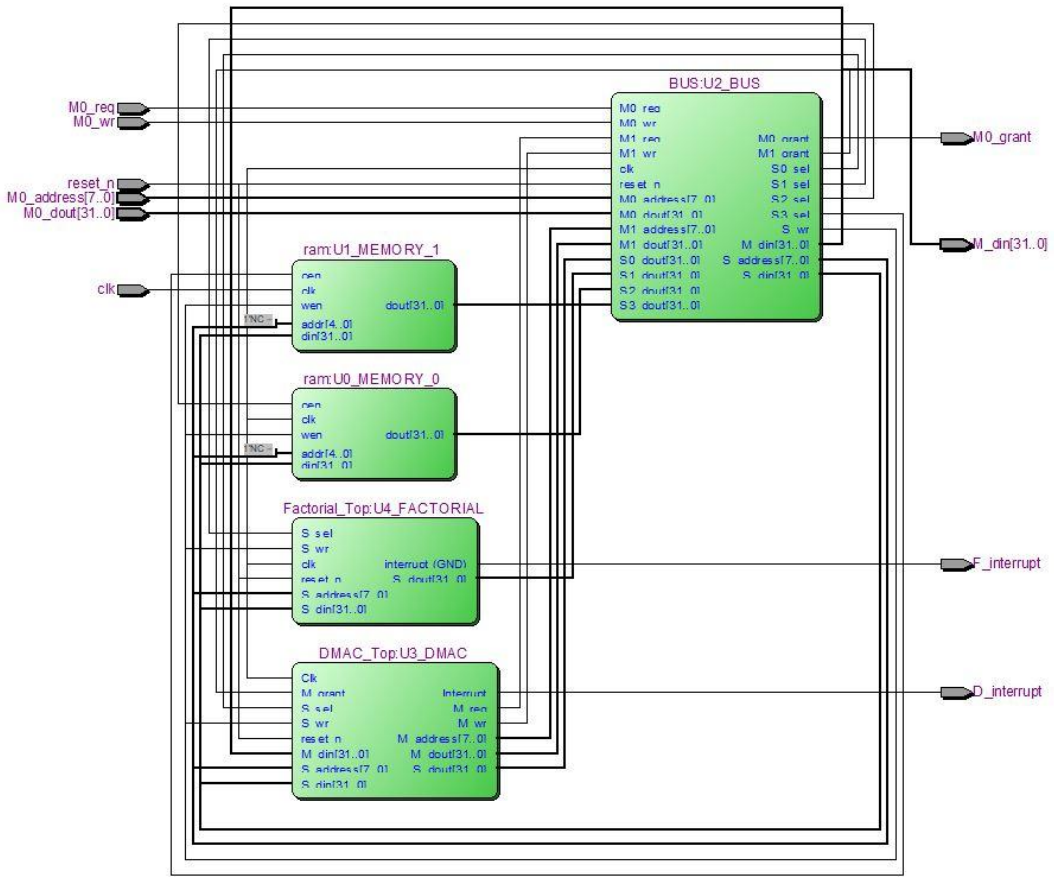
► **Waveform**



위 그림은 Bus 의 testbench 의 Waveform 이다. 처음 모든 값을 0으로 초기화 해준 뒤, reset_n signal에 0을 넣어 M0_grant 에서 시작했다. 각 component 의 주소 범위에 해당하는 값을 M0_address 와 M1_address 에 input 시킨 후, output 값이 출력이 잘 되고 있음을 확인할 수 있다.

5. TOP

► **RTL viewer**



위 그림은 TOP 의 RTL Viewer이다. 해당 port 들이 정확히 연결되어 있음을 확인할 수 있다.

► **Flow Summary**

Flow Summary	
Flow Status	Successful - Tue Dec 05 22:10:40 2017
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	TOP
Top-level Entity Name	TOP
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	2,624
Total combinational functions	1,344
Dedicated logic registers	1,517
Total registers	1517
Total pins	79
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

위 그림은 TOP 의 Flow Summary 이다. Total logic elements 는 2624, Total registers는 1517, Total pins 는 79 임을 확인할 수 있다.

V. Conclusion

이번 프로젝트는 실습 시간에 구현했던 module 을 활용하여 factorial operation을 수행하는 Hardware를 설계하는 프로젝트였다. 구현한지 오래된 module 들도 다시 보고, Verilog 를 설계하는 방법에서 시작하여, 기본적인 문법들이나 FSM 을 그리는 방법 등의 한 학기 동안 배웠던 모든 내용을 다시 한 번 복습하고, 익히는 시간이 되었다. Verilog를 작성하는 것이 쉽지 않았지만, 이번 프로젝트를 통해서 배웠던 내용들을 다시 복습하고, Hardware를 설계하는 실력이 향상됨을 느낄 수 있었다.

처음 프로젝트를 시작할 땐 어떤 부분부터 시작해야하는지 너무도 막막했다. 과제로 수행했던 Memory 와 BUS 를 다시 복습해보고, 이번 프로젝트에 맞게 수정하는 과정에서 약간의 감을 잡고 DMAC를 구현했다. DMAC 를 구현하는 부분에선 offset에 해당하는 register 의 해당 조건이 되었을 때 신호를 output 하는 것이 잘 되지않았고, 원하는 결과를 얻을 수 없었다. 대부분 clk 에 module 이 동기화가 되어있기 때문에 하나의 실수로 모든 동작이 뒤로 밀려버리거나 앞으로 당겨질 수 있으므로 전체적인 설계를 할 때, 타이밍을 맞춰 원하는 때에 값을 받아오는 것이 가장 어려웠던 것 같다. 그렇게 여러 번의 시행착오 끝에 DMAC를 구현할 수 있었지만, Factorial은 구현하지 못했다.

각각의 module 들을 따로 검증할 때는 동작이 잘 돼서 TOP module 에 instance 하여 검증을 진행했다. 하지만 동작이 제대로 이루어지지 않아 크게 좌절했다. Top 에서 Bus 를 통해 data를 전송하는 과정에서 사용되는 조건들이 겹치는 부분도 있고, bit 수가 안 맞는 문제도 많았고, 그 중에서도 타이밍 문제가 가장 컸다. 각각의 sub module 들을 구현할 때 원하는 타이밍에 값이 나오지 않아서 register 도 많이 쓰고, 여러가지 enable 신호들을 사용한 것이 화근이었다. 결국 코드를 많이 수정하게 되었다.

예비 검증을 받을 때는 과제에서도 충실하게 수행했고 자신있었던 bus 에서 오류가 나는 것을 확인했다. bus에 해당 grant에 대한 request가 없다면 다른 grant로 권한이 넘어가도록 구현해야 했는데, grant에대한 request가 없고, 다른 grant에 대한 request도 없다면 현재 grant를 유지하도록 구현을 했고, 이를 확인 후 코드를 수정할 수 있었다. 또한 Interrupt신호에 대해 무한 루프가 도는 상황도 발생하였는데, 이 부분도 수정했다.

이번 프로젝트에서 Factorial 을 구현하지 못한 점이 너무 아쉬웠다. 시간이 부족했다는 것은 핑계인 것 같고, 더욱 집중하고, 열심히 공부 했어야 했는데 라는 반성도 많이 했다. 앞으로 Verilog를 꾸준히 공부하고 연구 해야겠다는 생각이 들었고 hardware 에 큰 발자국을 내딛은 것 같다.

VI. Reference

공진흥 / 컴퓨터공학기초실험2 / 새빛관303호(광운대학교) / 2017년.

이준환 / 디지털논리회로2 / 참빛관B101호(광운대학교) / 2017년.