컴퓨터 공학 기초 실험2 보고서

BUS & ALU with Multiplier & Memory

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화 1,2,3

학 번: 2019202050

성 명: 이강현

1. 제목 및 목적

A. 제목

BUS & ALU with Multiplier & Memory

B. 목적

이번 프로젝트는 BUS를 통한 ALU with Multplier의 동작과 memory접근이 가능한 시스템을 설계해보면서 논리연산, 덧셈, 뺄셈, 곱셈기의 개념과 memory의 개념, 그리고 그들간의 인스턴스를 이해하고 구현해보고 학습하는 것이 목적이다.

<일정>

	11주차	12주차	13주차	14주차
제안서				
코드 작성				
코드 검증				
결과 보고서				

일정에 맞게 프로젝트를 진행했고 12주차에 코드를 일단 구성한 후 지속적으로 업데이트되는 git discussion을 참고하여 14주차까지 코드 작성과 검증을 겸하여 진행했다.

<Introduction>

프로젝트는 testbench를 통해 제어되는 bus,memory,alu, multiplier로 구성된 디지털 시스템을 구성하는 것이다. testbench라는 master와 alu,multiplier라는 slave들의 데이터를 주고받는 관계를 bus를 통해 정의하고 alu,multiplier는 정의된 관계에 맞게 데이터를 연산하며 저장하고 testbench는 이렇게 연산된 데이터를 얻을 수 있으며 연산을 하게끔 slave들을 bus를 통해 제어할 수 있다. 프로젝트를 진행하면서 제시된 수행순서에 따라 시스템이 작동하도록 구현해본다.

2. 원리(배경지식)

<Project specification>

<ALU>

ALU는 arithmetic logic unit의 약자로 CPU에서 연산을 담당한다. Opcode를 해독하여 연산을 진행할 operator를 결정하며 이를 통해 operand와 연산한다. 프로젝트에서는 NOT A, NOT B, AND, OR, XOR, XNOR, Set less than, Set greater than, Shift left logical, Shift right logical, Shift right arithmetic, Addition, Substraction 총 13가지의 연산을 ALU에서 진행한다. 기존의 ALU는 연산의 결과로 추가적인 flag를 반환하는데 carry, negative, zero, overflow 4가지의 flag를 반환하면서 연산의 결과를 뒷받침한다. 이번 프로젝트에서는 flag를 사용하지 않으며 결과적으로 cla32에서 추가적인 carry out을 사용하지 않는다 따라서 cla의 이부분을 수정하였다.

<Multiplier>

Multiplier는 곱셈기로 binary방식과 booth방식이 존재하는데 binary방식은 일반적으로 우리가 곱셈하는 방식을 사용하여 multiplicand에 multiplier를 한자리씩 이동하며 곱하는 방식이라 연산이 너무 길어진다는 단점 때문에 booth방식을 구현에 사용했다. Booth multiplication은 2-radix multiplication방식이며 이는 multiplier의 마지막 자릿수와 1bit를 비교하여 00과 11인경우 shift만 01인경우 add를 10인경우 substract를 진행하여 연산을 진행한다. 00과 11인경우 단순히 shift연산만 진행하므로 연산의 수를 줄일 수 있는 것이 장점이다. 이번 프로젝트에서는 위와 같은 2-radix multiplication방식을 사용했으며 결과적으로 32clock이 연산에 드는 시간이다. 따라서 count변수를 통해 32clock후에 연산결과가 나오게끔 구현하였고 opdone을 통해 연산결과를 뽑아내는 타이밍을 결정하였다. 또한 multiplication은 multiplier에서 진행되므로 프로젝트의 ALU with Multiplier 모듈의 14번째 연산 multiplication을 해당 모듈에서 담당한다.

<Memory,ram>

Memory는 기억장치로 데이터를 읽고 저장하는 작업을 처리하는 시스템이다.
컴퓨터에서 대체적으로 전원이 꺼지면 데이터가 지워지는 휘발성이라는 특징을 가진다.
레지스터,캐시,주 기억장치, 버퍼, 보조기억장치등 목적에 맞게 여러가지로 사용된다.
Ram은 random access memory의 약자로 순차적으로 메모리에 접근하지 않고 특정 주소값에 따라 임의적으로 접근이 가능한 기억장치를 말한다. 따라서 읽기 쓰기 속도가 빠르다는 장점이 있다. 프로젝트내에서 slave0에 해당하며 testbench에 의해 bus로부터 데이터를 받고 저장하는 담당이다.

<bus>

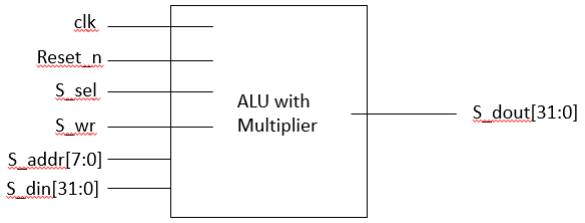
BUS는 여러 구성요소간 data의 전송이 가능케 하는 연결 구조이다. Master와 slave가 존재하며 Master가 요청을 보내면 grant신호로 회답하여 slave에 데이터를 보낼 수 있게 되고 접근할 수 있는 권한을 얻는다. Master와 slave의 1대1 연결을 위해 master는 arbiter, slave는 mux를 통해 서로 데이터를 주고받게 된다. Bus의 장점으로는 새로운 component를 추가하기 쉽다는 것이다. 프로젝트에서 testbench라는 master와 memory라는 slave0,ALU with Multiplier라는 slave1을 가지고 그 사이에서 데이터 이동을 담당한다. Master가 1개이기 때문에 m_req가 결과적으로 m_grant에 그대로 인가되고 slave의 결정은 testbench에서 입력되는 address에 따라 0x00보다 크거나 같고 0x20보다 작으면 slave0 0x30보다 크거나 같고 0x40보다 작으면 slave1의 데이터를 받는다. 그외의 주소에서는 데이터를 받지 않는다.

3. 설계 세부사항

<Design details>

<ALU with Multiplier>

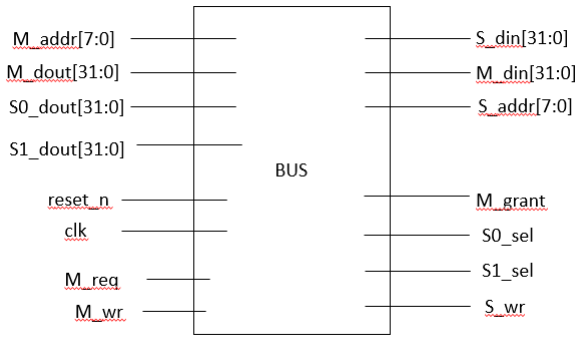
direction	Port name	Bit width	Description
Input	clk	1	Clock
	reset_n	1	Active low reset
	S_sel	1	Select
	S_wr	1	Write/read
	S_addr	8	Address
	S_din	32	Data input
output	S_dout	32	Data output



ALU with Multiplier의 pin이다. Clock에 맞게 동작하고 ALU with Multiplier가 slave중 하나이기 때문에 선택되었다는 것을 나타내기 위해 S_sel이 있고 읽고 쓰기를 선택하는 S_wr, 어떤 연산을 진행할지를 선택하기 위한 S_addr, 어떤 data를 넣을지를 담당하는 S_din을 입력값으로 받고 결과값을 출력하는 S_dout이 있다. Register8개와 alu32, multiplier가 인스턴스 되었다.

<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	8	Master address
	M_dout	32	Master data output
	S0_dout	32	Slave 0 data out
	S1_dout	32	Slave 1 data out
output	M_grant	1	Master grant
	M_din	32	Master data input
	S0_sel	1	Slave0 select
	S1_sel	1	Slave1 select
	S_wr	1	Slave write/read
	S_addr	8	Slave address
	S_din	32	Slave data input



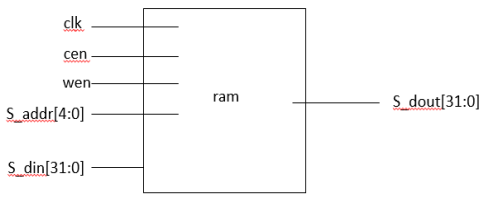
BUS의 pin을 나타낸 것이다.

Master가 data(M_addr,M_dout,M_wr)를 넘겨 주고 이를 grant상태에 따라 slave에게 넘길지 말지를 결정한다. slave선택은 master에게 넘겨받은 M_addr를 해독하여 결정하고 S0_sel과 S1_sel로 선택받은 slave를 나타낸다. slave에게 데이터를 넘기고 그에대한 응답으로 S_dout을 master에게 넘겨준다.

Master가 한 개이므로 arbiter의 역할을 1비트 register로 clock에 따라 작동하게끔 하였고 bus_addr모듈을 통해 S_addr을 해독하여 S0_sel과 S1_sel둘 중 하나를 고른다. 3개의 데이터중 하나를 고르게끔 mux를 인스턴스하여 아무 slave도 선택되지 않았을 경우, slave0의 데이터,slave1의 데이터중 하나를 master에 넘겨준다.

<memory>

direction	Port name	Bit width	Description
Input	clk	1	Clock
	cen	1	Chip enable
	wen	1	Write enable
	S_addr	5	Address
	S_din	32	Data in
output	S_dout	32	Data output



Reg형식으로 32비트 데이터를 저장할 수 있는 32개의 크기를 가진 배열을 선언하고 이를 for문으로 초기화 한다. 그 후 메모리를 cen이 1일 때 접근하고 wen이 0일 때는 S_addr에 해당하는 데이터를 읽고 wen이 1일때는 S_addr에 해당하는 곳에 데이터를 쓴다.

3개의 모듈을 인스턴스하여 Top모듈을 생성하였고 BUS를 통해 데이터를 주고받고 memory에 접근 및 저장이 가능한 디지털 시스템을 behavior하게 구현하였다.

4. 설계 검증 및 실험 결과

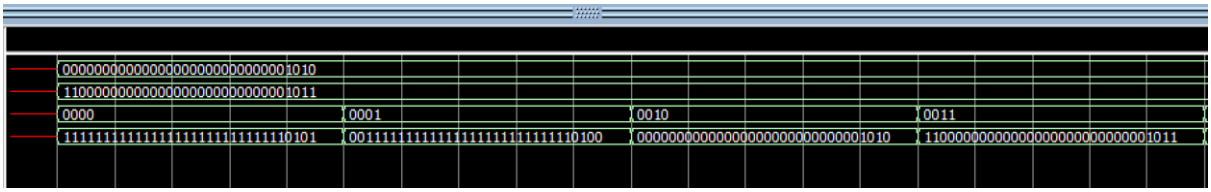
<Design verification strategy and results>

모든 모듈들은 exhaustive verification이 아닌 directed verification이 사용되었다. 32비트이므로 모든 경우의 수를 파악하기 힘들기 때문에 필요하다고 생각되는부분과 그렇게 testbench를 구성한 이유를 같이 결과별로 설명한다.

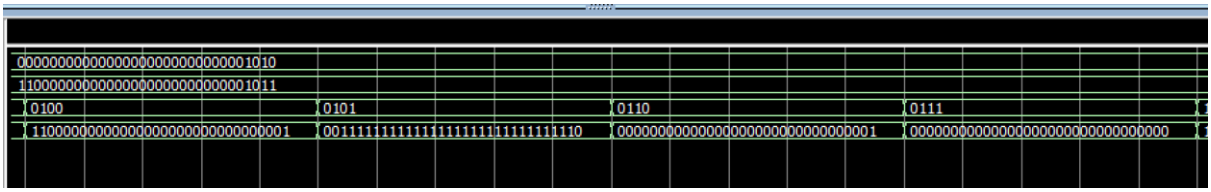
A. 시뮬레이션 결과

ALU는 0000000A와 C000000B를 operandA,operandB로 설정하여 연산했다. 그 이유는 binary로 표현될 때 논리연산에서 0과 0, 0과 1,1과1이 논리연산되는 것을 모두 확인할 수 있고 끝과 끝에 값이 있으므로 shift연산이 되는 것을 직접 확인할 수 있고 음수와 양수의 연산 또한 표현 가능하기 때문에 B는 음수이지만 A보다 절댓값이 더 크므로 unsigned기준으로 set less than이나 set greater than 연산이 잘 되는지도 알 수 있다. 하나의 입력값 케이스로 다양한 결과를 확인할 수 있기 때문에 위와같이 값을 설정했다.

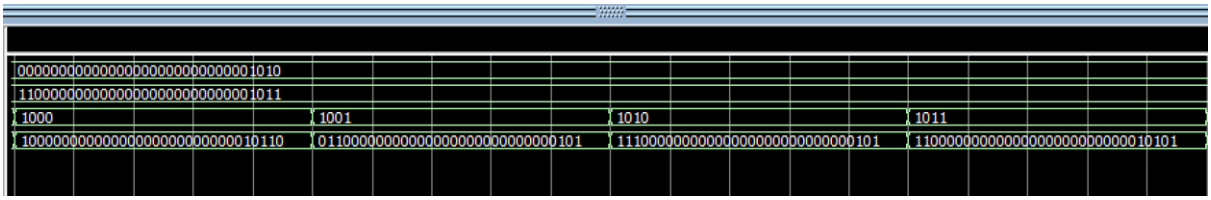
<ALU>



ALU에서는 모든 명령어들의 결과를 검증했다. NOT A(op-0000)명령어는 a[3],a[1]을 제외한 모든 비트가 1로 셋되어 NOT명령어가 잘 실행되었다. NOT B(op-0001)또한 b[31],b[30],b[3],b[1],b[0]을 제외한 모든 비트가 1로 셋되어 잘 실행됨을 알 수 있다. AND(op-0010)은 1과 1이 만나는 자리만 1로 셋되었고 OR(op-0011)은 1이 하나라도 존재하는 자리는 1이 된 것을 확인할 수 있다.



XOR(op-0100)은 자리에 1이 하나만 있어야 출력되고 XNOR(op-0101)은 자리에 1이 1이 짝수개수만큼 있어야 1이 출력되어야 하기 때문에 둘다 올바른 값을 얻었다. Set less than(op-0110)은 unsigned 기준으로 B가 음수이지만 절댓값이 더 크므로 A가 더 작다고 인식하여 1을 출력한 것을 확인할 수 있고 Set greater than(op-0111)은 그 반대로 0을 출력했다.

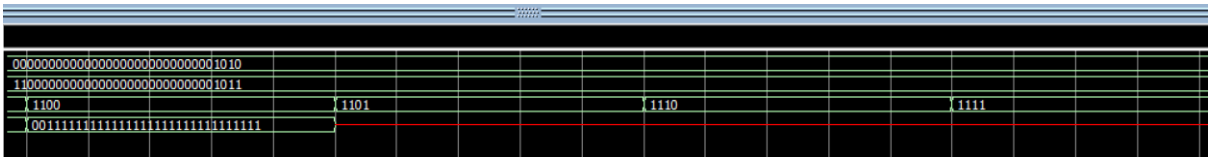


Shift left logical(op-1000)은 끝과 끝을 보고 B가 한칸 왼쪽으로 이동한 것을 알 수 있다. Shift right logical(op-1001)은 한칸 오른쪽으로 이동한 것을 알 수 있다.

Shift right arithmetic(op-1010)은 부호비트가 복사되어 오른쪽으로 이동하여 shift right logical과의 차이를 알 수 있다.

Addition(op-1011)은 비트끼리 잘 더해진 것을 확인할 수 있고 캐리도 확

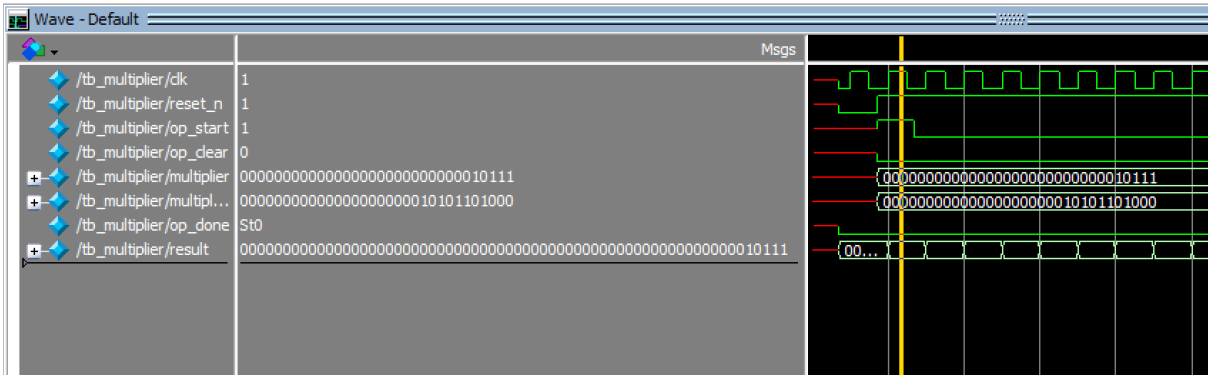
인할 수 있다.



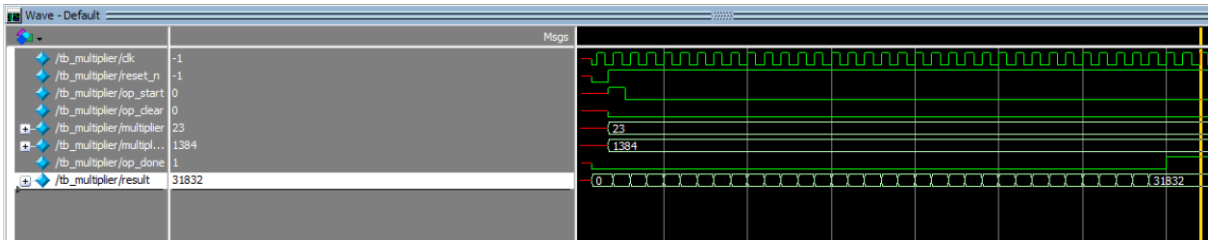
Subtraction(op-1100)은 b의 보수를 더하는 연산으로 b의 마지막 부분이 0101이고 a가 1010이므로 결과값은 맨 앞 두비트를 제외한 모든부분이 1인 것을 확인할 수 있다.

<Multiplier>

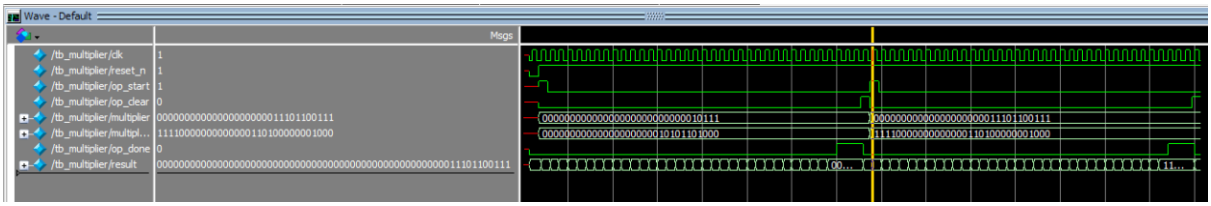
Multiplier는 무작위 두 숫자를 연산하였다.



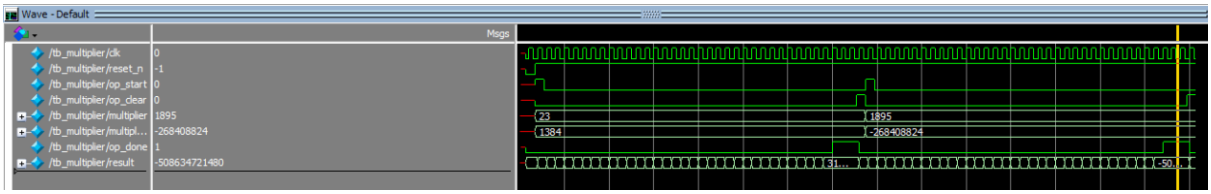
Start가 1이 될 때 booth 방식대로 result에 multiplier를 넣어두고 대기한다.



op_start가 0이되고부터 최종결과값까지 진행되는 연산의 수는 32clock이고 값이 변하고 나서 오는 첫 rising clock에서 op_done이 1로 변하는 것을 알 수 있다. 위는 한번의 연산시 2-radix booth multiplier의 연산 과정을 알 수 있는 결과 사진이다.



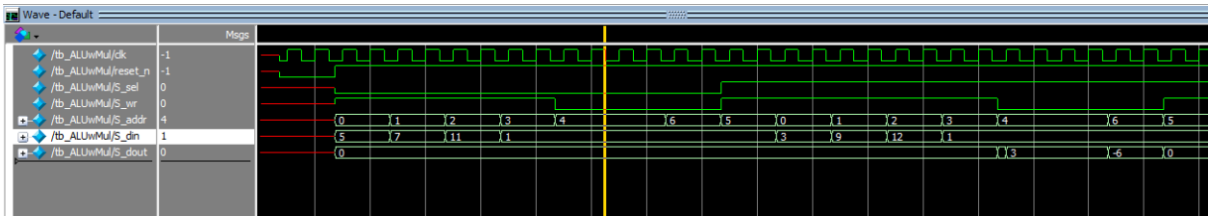
두번째 연산이다. 위의 연산과 마찬가지로 op_start가 1이 되면 multiplier 값을 넣어두고 대기하고 0이 되자마자 32clock후에 값을 확인할 수 있다.



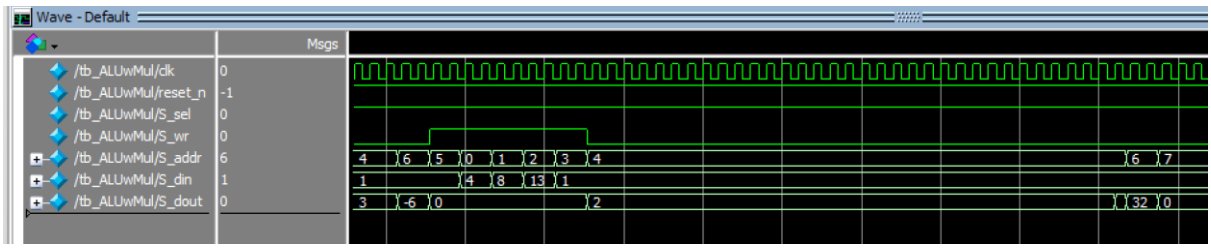
Decimal로 확인한 결과 올바른 값을 얻었음을 확인했다.

<ALU with multiplier>

이번 모듈에서는 slave가 선택되지 않았을 때, 선택되고 ALU를 사용하는 경우, Multiplier를 사용하는 경우 그리고 clear를 통해 초기화 하고 연산을 시작하는 경우와 초기화를 진행하지 않고 연산을 시작하는 경우로 나누었다.

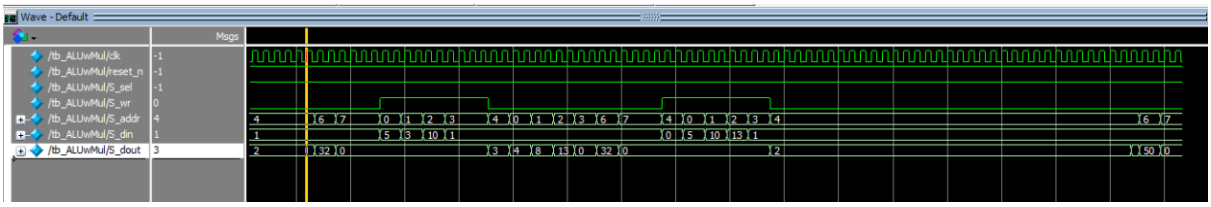


첫번째로 모듈이 선택되지 않은 경우에는 어떤 값도 register에 들어가지 않아 offset 6인 register에서 0이 출력되는 것을 볼 수 있다. 하지만 선택된 후에 들어간 값 3과 9로 명령어 12에 해당하는 뺄셈을 진행한 결과 -6이 한 clock내로 op_done이 11로 바뀌어서 나온 것을 확인할 수 있다.



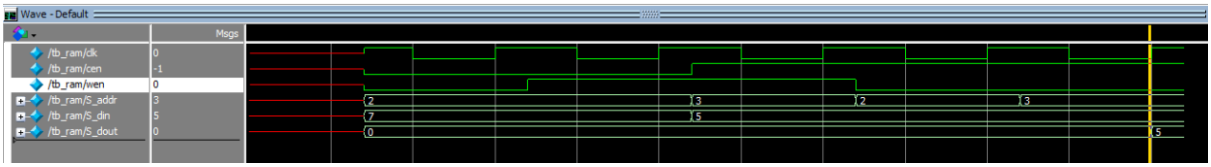
위는 clear에 1을 쓰고 연산을 새로이 진행하는 모습이다. 4와 8로 13에 해당하는 곱셈을 진행한 결과 10으로 진행상황을 나타내주고 11로 변한 후 32라는 결과를 올바르게 얻은 것을 확인할 수 있다. 연산의 횟수는 multiplier에 위의 사진기준을 3이라는 주소에 1을 쓰면 start에 1을 쓰는 행위를 하는 것이고 그 이후 처음 만난 클록에서 실질적으로 opstart가 1

이 된다. 1이 되자마자 0을 쓰게 되기 때문에 0이 써지게 되면 연산을 시작한다. 사진기준 주소 4의 바로 전 rising edge에서 연산이 시작되는 것이다. 그 후 연산은 사진보다 먼저 끝나지만 multiplier에서 결과가 나온 후 done을 쓰는 작업, done을 확인하고 현재 모듈에서 register에 값을 쓰는 작업이 필요하므로 결과가 나오고 2clock이후에 opdone이 11이 되어 값을 확인할 수 있게 된다.



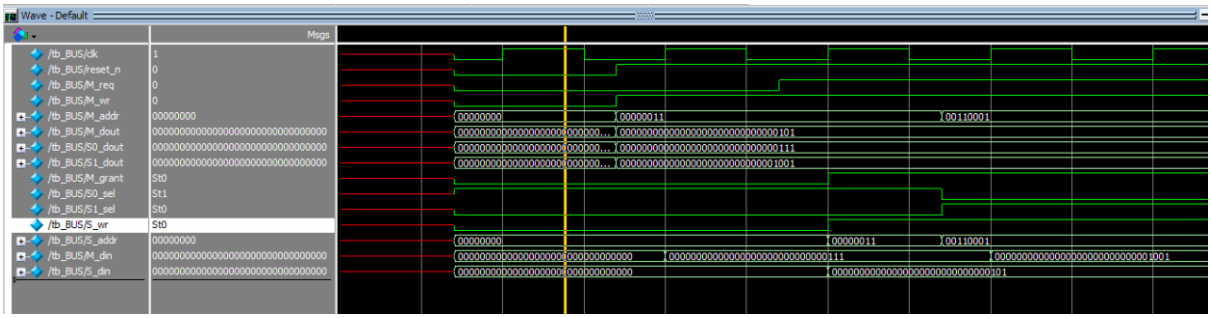
위는 32라는 결과를 얻고 초기화를 하지 않았을 때 값을 새로이 쓰고 진행상황을 확인한 결과 연산완료를 나타내는 3(11)이 나오고 레지스터에 전 연산의 값들이 아직 저장되어 있는 것을 확인할 수 있다. 또한 이번에는 opclear register가 아닌 opdone register에 0을 쓰는 방법으로 새로이 연산을 시작한 결과 새로운 값들을 기준으로 50이라는 값을 얻은 것을 확인할 수 있다.

<memory>



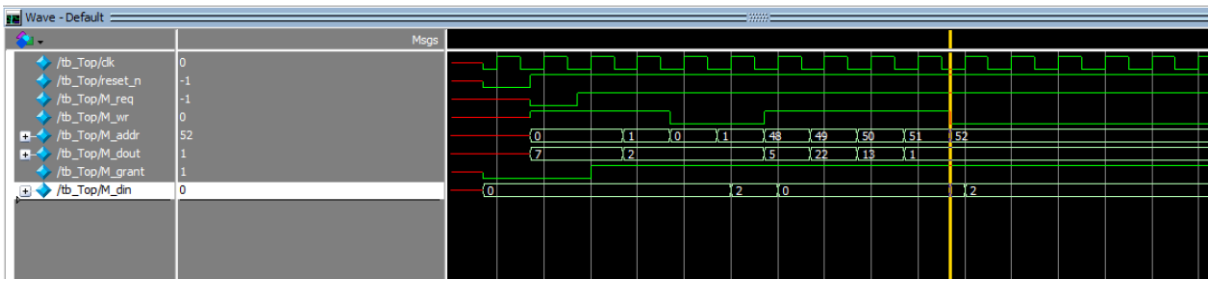
Enable 신호가 중요한 역할을 하는 모듈이므로 cen과 wen을 중점적으로 검증을 진행했다. 2라는 주소에 7을 넣으라는 명령은 cen이 0이므로 뒤에 cen을 1로 만들고 wen이 0으로 읽기를 한 결과 아무 결과도 나오지 않은 않은 것 보아 저장되지 않았음을 알 수 있고 cen이 1일 때 wen도 1로 만들어 3이라는 주소에 5를 넣으라는 명령은 5로 잘 저장되었음을 확인할 수 있다.

<BUS>

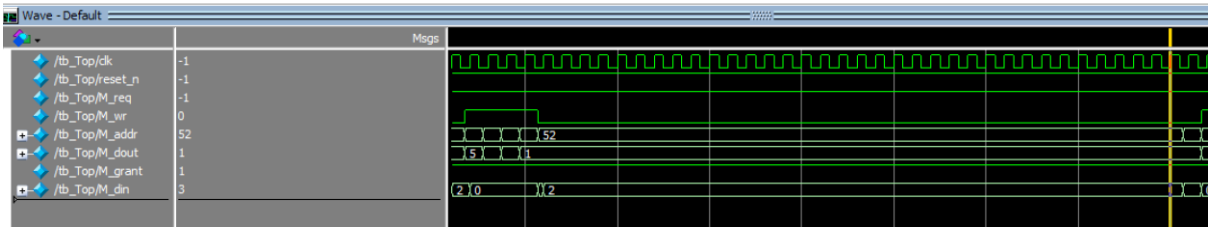


노란줄을 기점으로 보면 모든 값이 reset_n이 0일때는 모두 0임을 알 수 있다. 그 후 reset_n을 1로 바꾸고 M_addr:3,M_wr:1,M_dout:5의 입력값들을 넣어준 결과 M_req가 0이므로 data입력의 권한을 얻지 못했으므로 S_wr,S_addr,S_dout이 모두 0으로 slave에서 값을 받지 못했음을 확인할 수 있다. 그 후 M_req가 1로바뀌면 값들이 그대로 인가된다. 어떤 slave를 선택할지는 S_addr 즉 입력값에서 인가된 주소값을 통해 선택하는데 S_addr가 slave1의 범위에 해당할 때 slave1의 값이 들어오는 것을 확인했다.

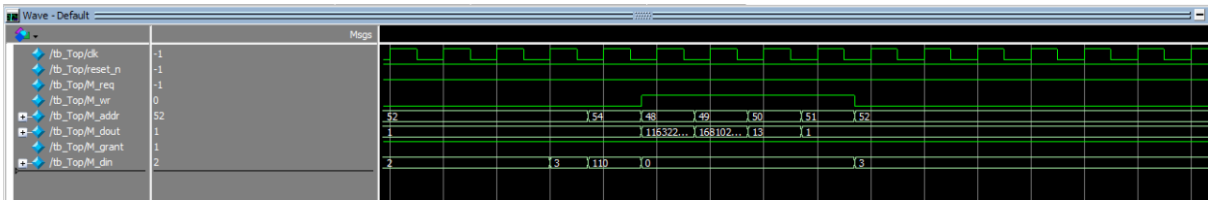
<TOP>



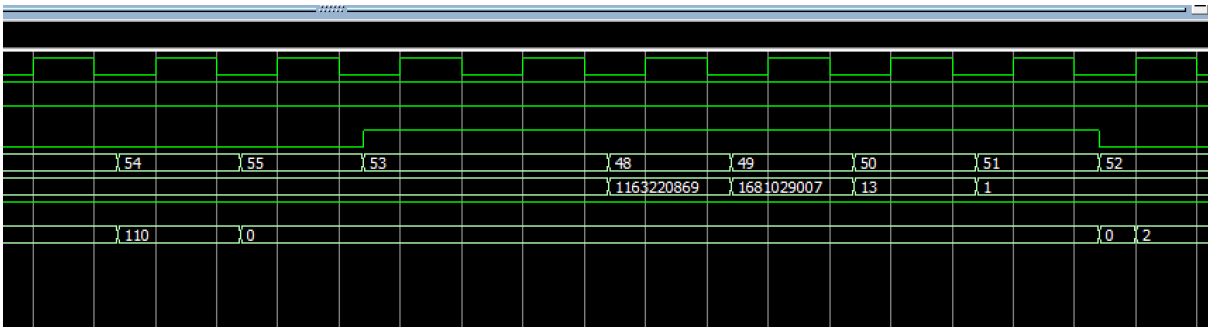
우선 slave0인 메모리에 값을 적어보았다 M_req가 0일때는 7이라는 값을 넣어도 값이 저장되지 않았고 1로 M_req를 바꾼 후 2라는 값을 넣었을 때는 값이 잘 저장되었다. 그 이후 48(0x30)부터 ALUwMul 모듈에서 검증한 방법과 같이 5와 22를 넣어 곱셈을 연산한 결과



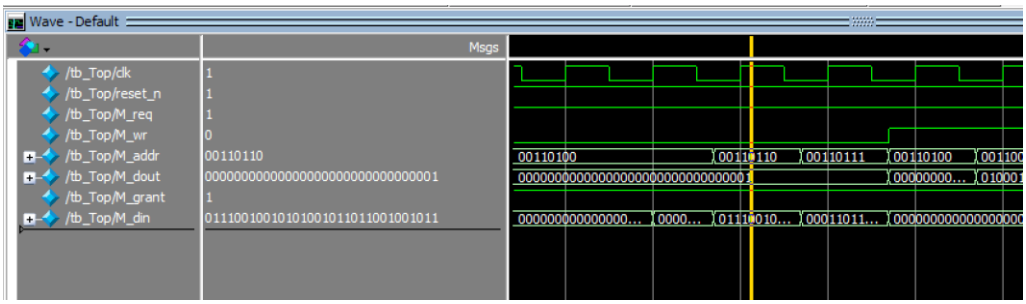
위와 같이 연산이 3(opdone 11)이 되어 연산이 완료되었음을 알 수 있고



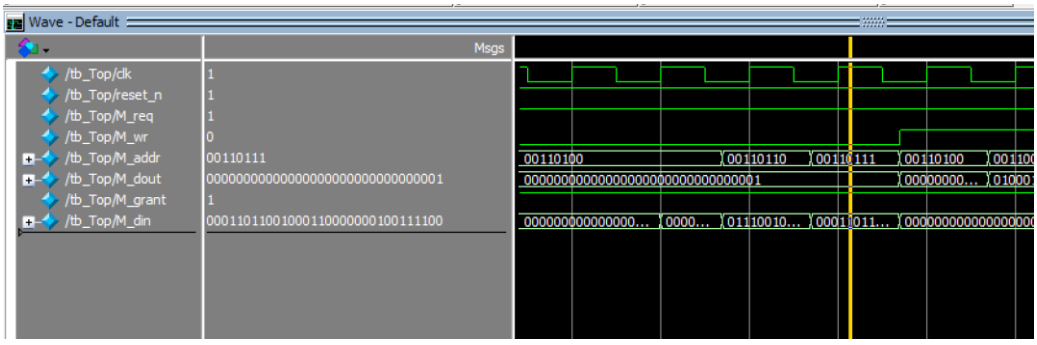
110이라는 값을 올바르게 얻었다. 그 이후 새로운 연산을 초기화 하지 않으니 3으로 opdone이 유지되어 연산이 진행되지 않았다.



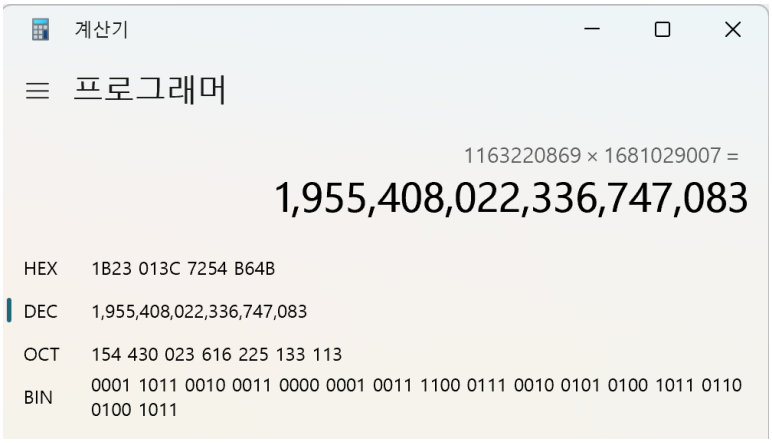
따라서 왼쪽에 이전에 연산한 값이 그대로 나오는 것을 확인했다. 이제 모든 자릿수를 다 채워서 곱셈을 진행해 본다. 곱셈을 진행하기 전 53(opclear)에 1을 써 초기화하고 진행한다.



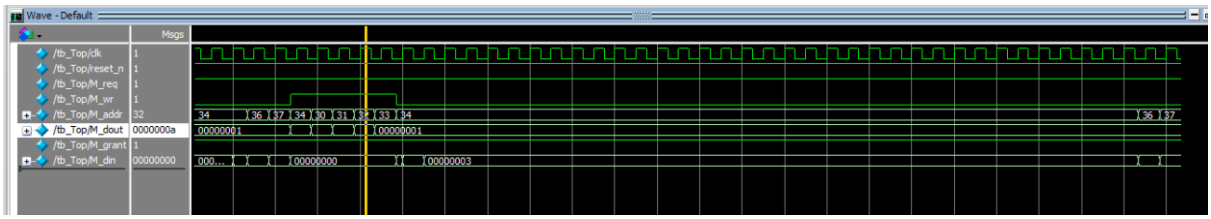
하위비트는 위와 같고



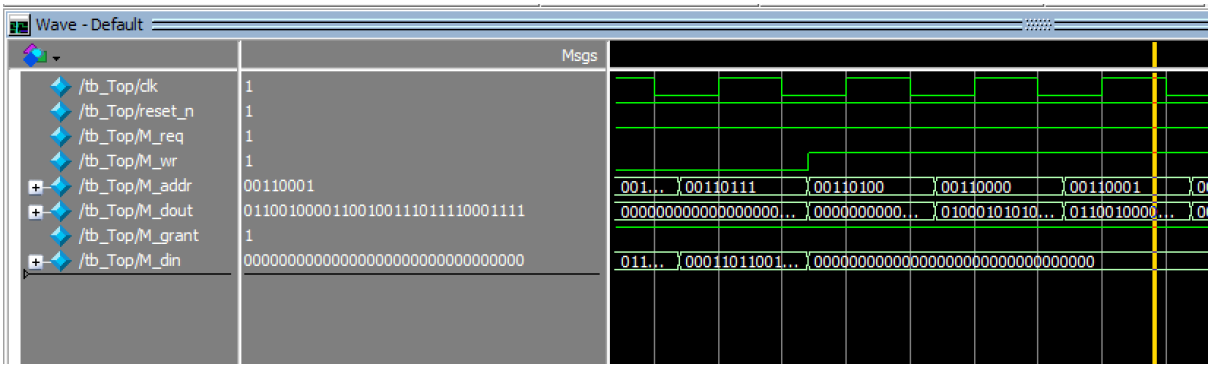
상위비트는 위와같다.



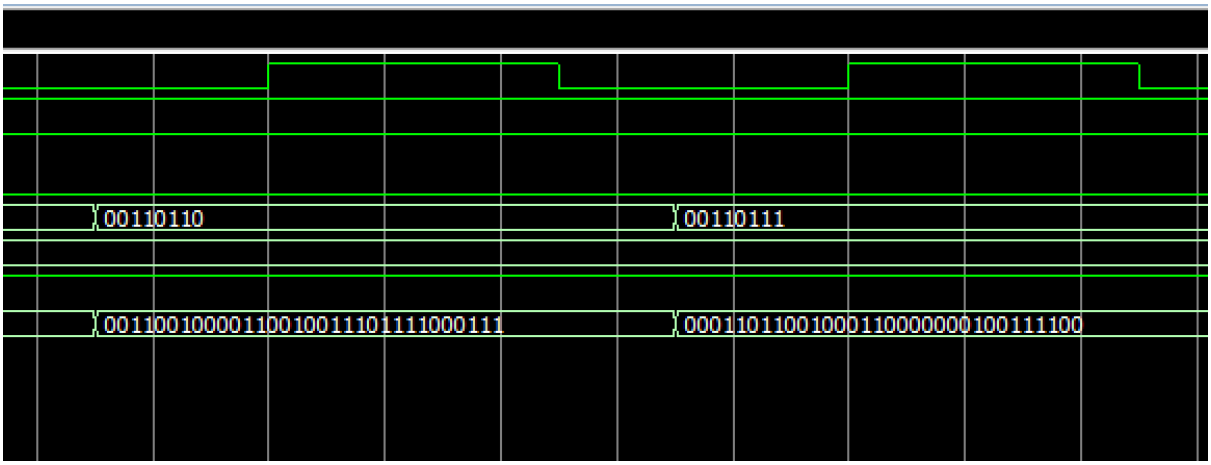
값이 32비트씩 나누어 저장되었음을 확인한다.



이제 ALU를 사용해본다. 이번에는 opdone을 0으로 초기화하여 연산을 진행한다. A라는 명령어를 통해 값을 연산한결과 바로 1clock안에 연산되어 값이 나오는 것을 확인했고



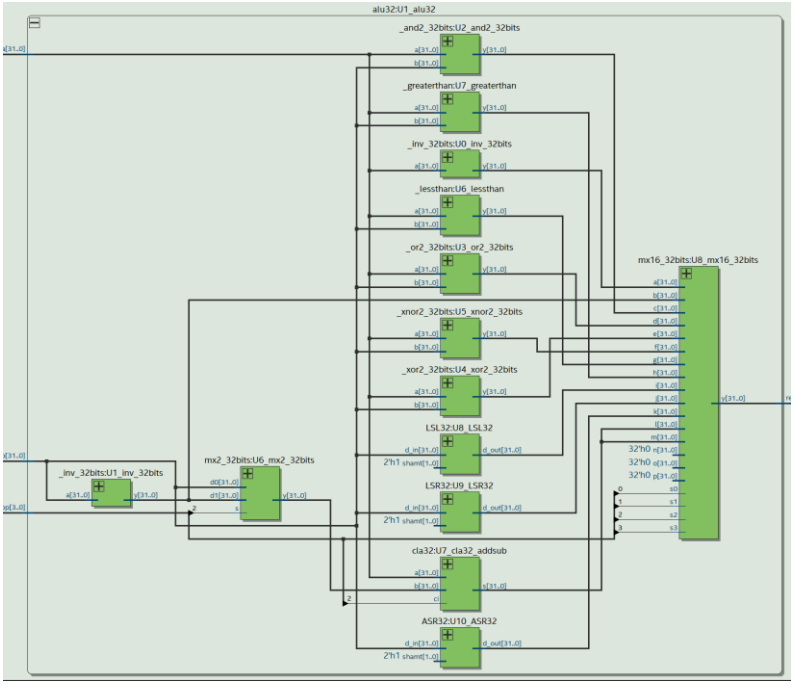
OperandB의 값이 arithmetic shift right되어야하기 때문에 위의 값의 결과가 result1에 저장되어야 한다.



Result1에 저장되었고 result2에는 opdone을 0으로 초기화하고 연산을 진행했기때문에 register에 이전의 곱셈 연산값이 그대로 저장되어있음을 확인했다. 따라서 opclear를 통한 초기화와 opdone을 이용한 초기화의 차이점을 확인했다.

B. 합성(synthesis) 결과

<ALU>



ALU의 합성 결과이다. 각각의 명령어들을 연산하는 모듈들이 연결되어있고 어떤값을 뽑아낼지를 결정하는 mux와 모두 연결되어있다.

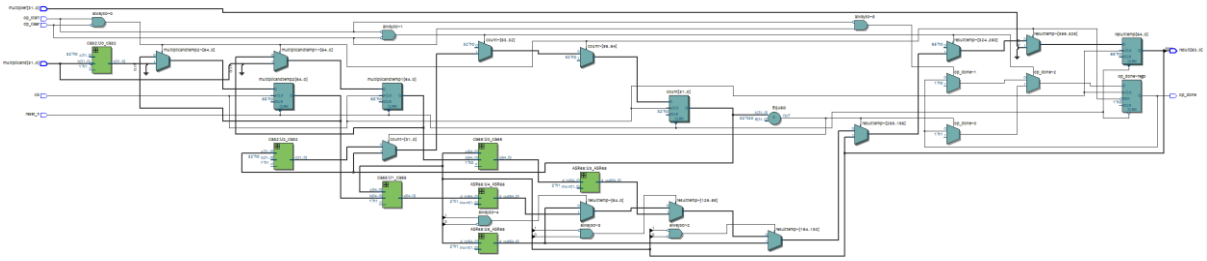
Flow Summary

<<Filter>>

Flow Status	Successful - Thu Dec 01 15:24:08 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	alu32
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	100
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

alu32만 흐름을 검사한 결과 성공적인 결과를 얻었다.

<Multiplier>

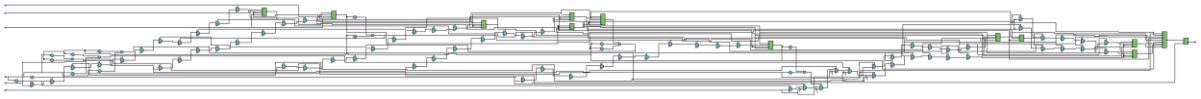


위는 multiplier의 합성 결과이다.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Dec 01 16:04:48 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	multiplier
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	160
Total pins	133
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

올바른 흐름을 나타낸다.

<ALU with multiplier>

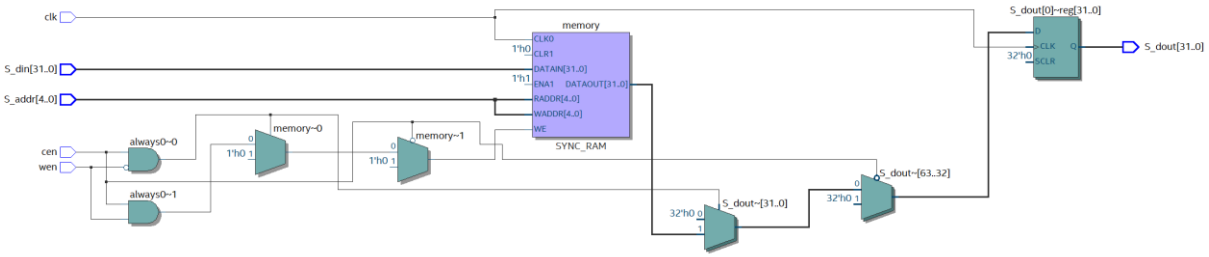


ALU와 multiplier가 잘 연결되어 있고 덧셈과 쉬프트에 필요한 각 모듈들도 잘 연결되어있다.

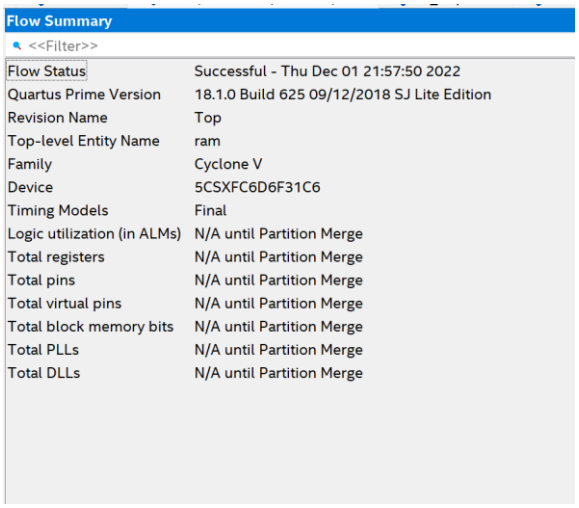
Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Dec 01 16:17:41 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	ALUwMul
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	415
Total pins	76
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

올바른 흐름을 알 수 있다.

<memory>



Memory 또한 잘 연결되어있다. Reg로 선언한 메모리 모듈이 생성되었다.



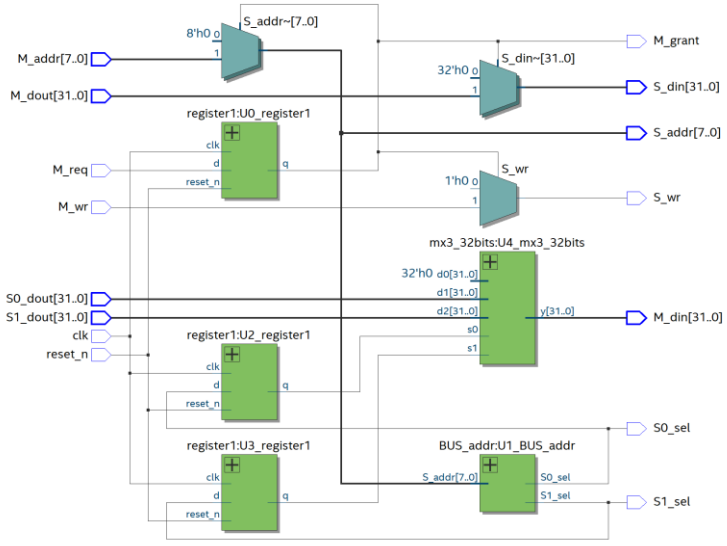
Flow Summary

<<Filter>>

Flow Status	Successful - Thu Dec 01 21:57:50 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A until Partition Merge
Total registers	N/A until Partition Merge
Total pins	N/A until Partition Merge
Total virtual pins	N/A until Partition Merge
Total block memory bits	N/A until Partition Merge
Total PLLs	N/A until Partition Merge
Total DLLs	N/A until Partition Merge

Ram 모듈도 올바른 흐름을 보여준다.

<BUS>



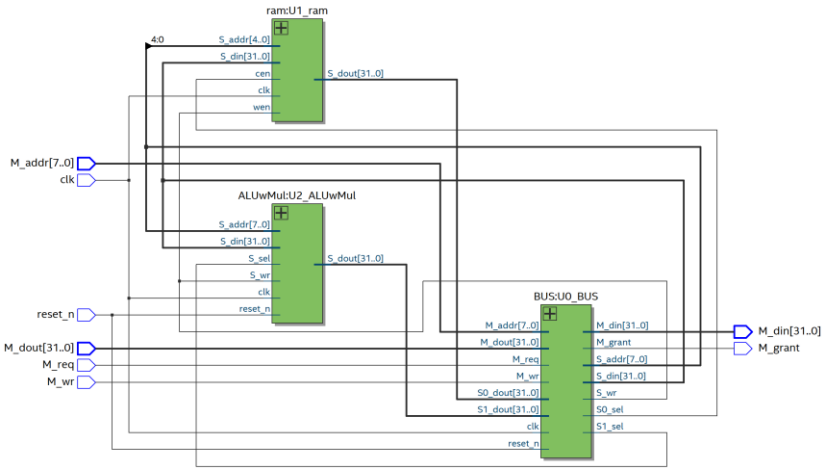
BUS는 M_req의 전달이 클록에 따라 잘 전달되고 mux를 통한 데이터 이

동도 잘 나타났다.

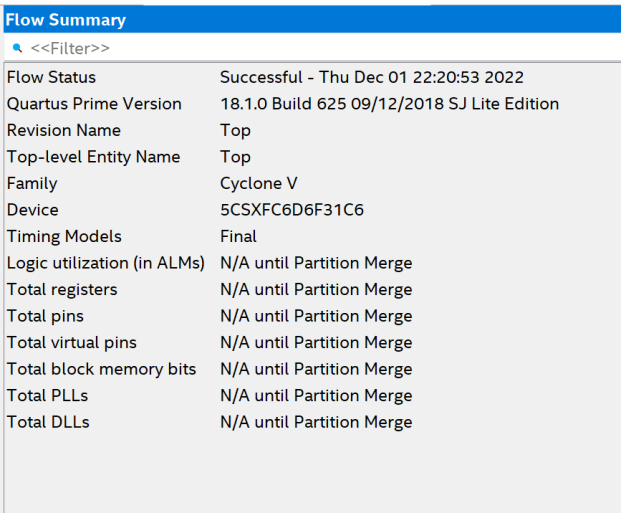
Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Dec 01 22:00:55 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Top
Top-level Entity Name	BUS
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A until Partition Merge
Total registers	N/A until Partition Merge
Total pins	N/A until Partition Merge
Total virtual pins	N/A until Partition Merge
Total block memory bits	N/A until Partition Merge
Total PLLs	N/A until Partition Merge
Total DLLs	N/A until Partition Merge

올바른 흐름을 보여준다.

<TOP>



마지막으로 TOP 모듈이다. TOP모듈은 BUS에 입력값들이 잘 연결되어있으며 BUS의 출력값들이 slave들의 입력값으로 잘 들어갔고 slave들의 출력값들이 BUS의 입력값으로 들어가 TOP의 출력값으로 나오는 것을 확인했다.



Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Dec 01 22:20:53 2022
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)	N/A until Partition Merge
Total registers	N/A until Partition Merge
Total pins	N/A until Partition Merge
Total virtual pins	N/A until Partition Merge
Total block memory bits	N/A until Partition Merge
Total PLLs	N/A until Partition Merge
Total DLLs	N/A until Partition Merge

올바른 흐름을 보여준다.

5. 고찰 및 결론(conclusion)

A. 고찰

이번 프로젝트는 그동안 배운 모듈들을 적당히 프로젝트의 조건에 맞게 올바르게 수정하고 인스턴스할 줄 알면 충분히 쉽게 구현할 수 있는 난이도의 프로젝트였다. 하지만 쉬프트 모듈이나 cla같은 모듈들을 통해 산술연산을 진행해야했고 이전에 배운 모듈들이라 많이 헷갈리고 기억이 나지 않았다. 다시금 프로젝트를 통해 내용들을 상기할 수 있었고 특히 ALUwMul 모듈을 작성할 때 시간을 정말 많이 사용했다. 첫번째로 register file을 사용하려고 하였으나 register에 opstart가 쓰여짐과 동시에 opstart에 0을 쓰고 opdone을 1로 만든다는 부분이 동시에 일어나야한다는 점 때문에 하나의 주소에 하나의 데이터를 읽고 쓸 수 있다는 부분을 어떻게 해야할지 막막했고 클럭단위로 동작하기에 생각을 많이 해야했다. Register에 있는 클럭을 제거해보기도 하고 클럭단위로 움직이지 않도록 always구문을 사용해보기도 하였으나 클럭에 맞게 동작해야 값을 유지할 수 있을 것 같았고 결과적으로는 register 8개를 직접 모듈내에 인스턴스하여 사용했다. 읽기 쓰기모드가 다른 register들은 어떻게 해야하는지도 어려웠고 이는 register를 읽는 것은 언제든지 할 수 있게 끔 입력값과 read operation모듈과 함께 엮어 클럭단위로 읽을 수 있고 쓰기 논리를 always문을 통해 구현함으로서 읽기와 쓰기를 다르게 동작하도록 입력논리, 출력논리를 잘 구현해야만 했다. 또한 multiplier를 구현하는 과정에서 multiplier의 clear를 3-input nand로 엮어 opdone[3],opdone[2],opdone[0]을 입력 인자로 주어 명령어가 multiplication이라면 clear를 해제하고 연산을 준비하게끔 구현했으나 multiplication을 두번 연속 진행할 때 opdone을 통한 초기화를 하게 된다면 mulitplication명령이 바뀌지 않고 그대로 실행되므로 multiplier모듈을 초기화 하지못하고 연산을 진행하여 연산에 문제가 생겼다. 따라서 이를 연산이 끝난 후 opdone[1],opdone[0]를 and에 사용하여 연산

이 완료되면 multiplier를 초기화하는 것으로 바꾸었다. 따라서 done이 1로 바뀌고 register에 값을 인가한뒤 opdone이 11로 바뀌면 multiplier를 초기화하므로 한 클록의 차이로 값을 저장하고 모듈을 초기화 하게끔 구현하였더니 문제를 해결할 수 있었다.

B. 결론

이번 프로젝트는 컴퓨터와 같은 시스템을 구현해본 것 같다. 우리가 키보드를 통해 입력하면 안에서 특정 작업을 진행하고 그에 대한 결과를 화면으로 볼 수 있기 때문이다. 이러한 과정이 BUS와 ALU같은 작업모듈, 데이터를 저장하는 memory를 통해 간단하게 구현될 수 있어 신기하기도 했다. 또한 프로젝트에서 +같은 산술연산이나 concatenate한 방식을 사용할 수 없었다. 처음에 미리 이렇게 구현해두고 프로젝트를 마쳤는데 그럴 수 없다는 것을 듣고 처음엔 짜증이 났지만 이렇게 간단하게 코드로 구현할 수 있는 것들을 직접 모듈로서 구현하니 기본적인 것들을 다시 학습할 수 있는 계기가 되었고 또한 코드로는 몇줄밖에 안되는 거지만 RTL viewer로 확인하면 엄청 복잡하게 나오는 것을 보면 나중에 실제 모듈을 구현해야할 때 가장 기계적으로 가깝게 코드를 구현하는 것이 효율적이라는 생각을 했다. 인간에게는 간편하지만 그러한 방법이 기계에게는 어려운 연산이자 복잡하게 다가가기 때문에 기초적인 모듈로 계층적으로 구현하는 것의 중요성을 배운 것 같다.

6. 참고문헌

공영호/컴퓨터공학기초실험2/광운대학교/2022