컴퓨터공학기초실험2

프로젝트 결과 보고서

Term Project

학 과: 컴퓨터정보공학부

담당교수: 이준환 교수님

강의 시간: 수 7 8, 금 2

학 번: 2018202046

성 명: 이준휘

1. Introduction

A. Title

Term Project

B. Object

해당 과제를 통해 지금까지 Verilog 코드를 작성하는 코드를 토대로 프로젝트를 제작할 수 있다. 기존 모듈을 활용하고 변형하여 새로운 모듈을 만들 수 있다. 제작한 모듈을 검증함으로써 큰 Module의 검증을 하는 방법을 알 수 있다.

C. Schedule

	11/26~11/27	11/28~11/29	11/30~12/1	12/2~12/3
제안서 작성				
프로젝트 수행				
보고서 작성				

2. Project Specification

해당 프로젝트에서는 ALU with Multiplier, Ram 모듈을 만든다.

ALU with Multiplier 모듈은 연산자와 피연산자, 그리고 여러 결과와 상태를 저장할 레지스터를 가지고 연산자에 따라 논리 연산, 비교 연산, shift 연산, 덧셈 연산, 뺄셈 연산, 곱셈 연산을 수행한다. 곱셈을 제외한 연산의 경우 0x03번지에만 저장되고, 곱셈의 연산의 경우에는 0x04번지 또한 상위 비트가 저장된다. Op_done 신호는 만약 연산이 진행되지 않았을 경우 00을, 연산이 진행되고 있는 경우, 10을, 연산이 완료된 경우에는 11을 출력한다. 연산의 시작은 start 레지스터에 1을 넣음으로써 가능하다. 초기화의 경우에는 clear 레지스터에 1을 넣음으로써 모든 레지스터를 초기화할 수 있다.

Ram에서는 32비트 크기의 레지스터 32개에 값을 저장하고 출력할 수 있다. 이 때 access 되는 신호와 쓰기/읽기 구분 신호를 받아 읽거나 쓰기, 동작 없음이 결정된다.

위의 두 ALU와 Ram은 Bus통해 Slave로 연결되며, Master는 testbench 하나로 되었다. 0x00~0x0F 주소에서는 ALU에 접근하며, 0x20~0x3F의 주소에는 메모리가 접근하도록 한다. 해당 Slave에 주소와 값, 그리고 select 여부를 알려주는 신호를 같이 보낸다. Slave로부터 받는 신호는 즉시 신호를 받는 것이 아니라 이후 clk에서 값을 읽을 수 있다.

3. Design Detail

A. multiplier

해당 함수는 32비트끼리의 곱셈을 통하여 64비트의 값을 출력하는 회로다. 해당 회로는 기존에 구현한 64 bit Multiplier를 수정하여 제작하였다.

B. alu32

해당 함수는 ALU_w_mul 모듈에서 계산이 이루어지는 실질적인 부분을 의미한다. 해당 모듈의 틀은 기존에 제작한 alu32를 이용하여 제작되었으며, 해당 부분에 추가로 shifter, 비교 연산, muliplier를 추가하여 해당 코드를 구현하였다. 코드는 always문을 통하여 opcode의 값을 확인하고, start가 1이고 opdone이 발생되고 있지 않은 상태에서 연산이 진행된다. 각 값은 case문과 if문을 통해 비교하고, 해당하는 값에 맞추어 reseult1을 바꿔준다. 이 때 만약 곱셈의 경우 op_done을 01로 바꾸어주며 연산중임을 알려준다. 연산이 끝났을 경우 초기화가 되기 전까지 값을 유지시켜준다.

C. ALU_w_mul

해당 함수는 실질적으로 연산을 하는 모듈이다. 해당 모듈에서는 we 신호와 re 신호를 S_sel과 S_wr 신호를 통해 만들어낸다. State는 출력하는 S_dout이 무엇인지를 구분하는 신호이며 해당 값에 따라 result1, result2, op_done 신호를 출력하는 역할을 맡는다. write_operation 모듈은 기존에 만든 rf 모듈에서 만든 sub_module을 활용하여 제작하였으며 쓰기 신호일 때만 주소를 디코딩해주는 역할을 맡는다. Register8_32 모듈 또한 rf 모듈에서 가져왔으며 해당 모듈에서는 저장해야 할 값들이 들어있다. Alu32에서는 연산을 통해서 값을 출력하며 출력한 값들은 register8_32에 들어가 저장된다.

D. bus

해당 모듈은 ram과 ALU_w_mul을 slave로 하고 testbench를 master로 하여 값을 전달하는 역할을 맡는다. 해당 모듈은 기존 10주차 과제였던 bus를 토대로 제작되었으며 해당 모듈에서 인자의 값을 바꾸고 Master의 수를 1 줄이며 address_decoder의 주소를 조건에 맞게 조정하여 해당 회로를 구현하였다. 해당 회로에서는 arbiter가 필요 없기 때문에 (Master의 수가 1개) 이를 없애고 M_grant의 신호는 M_req로 레지스터에 입력 받는 것으로 바꾸었다.

E. ram

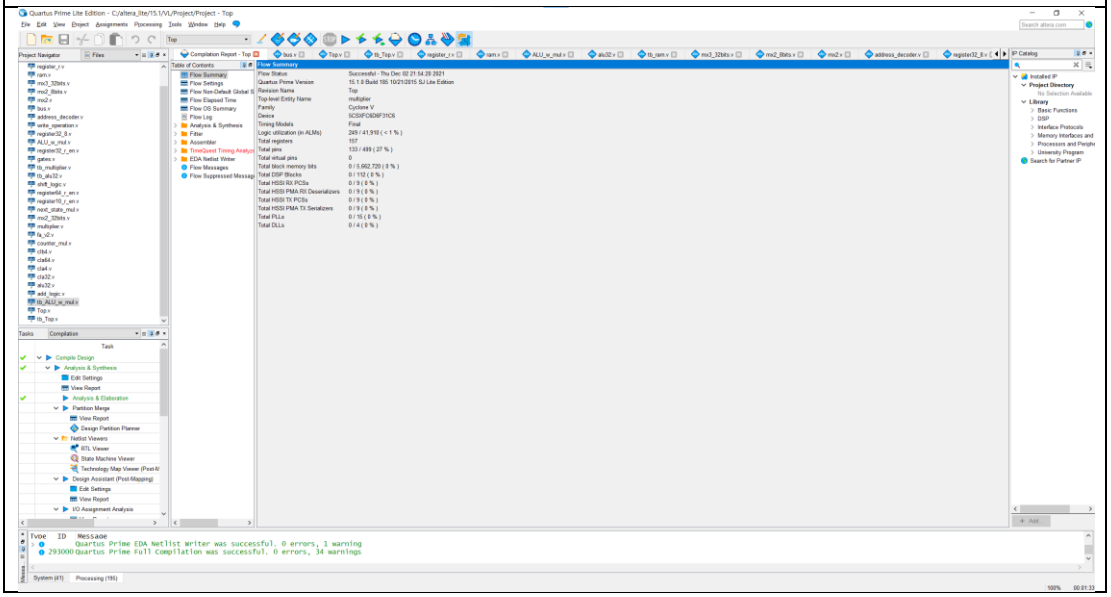
ram의 경우에는 32-bits 데이터 32개를 저장하는 역할을 수행한다. 해당 모듈은 기존에 만든 모듈을 그대로 사용하여 인자의 이름만 조정하였다.

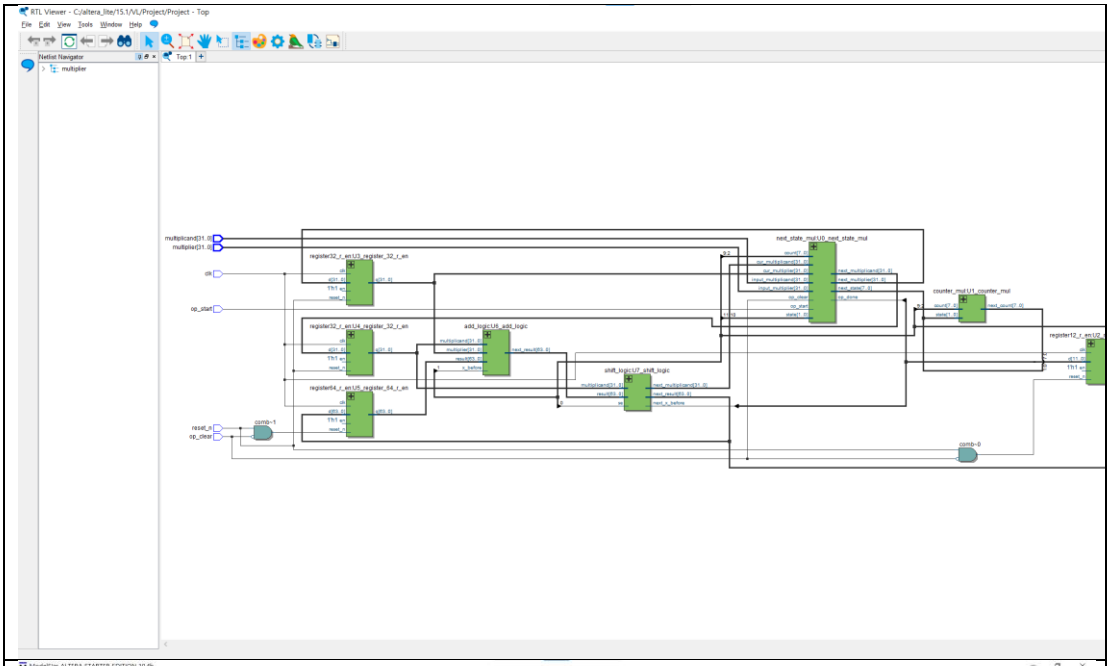
F. Top

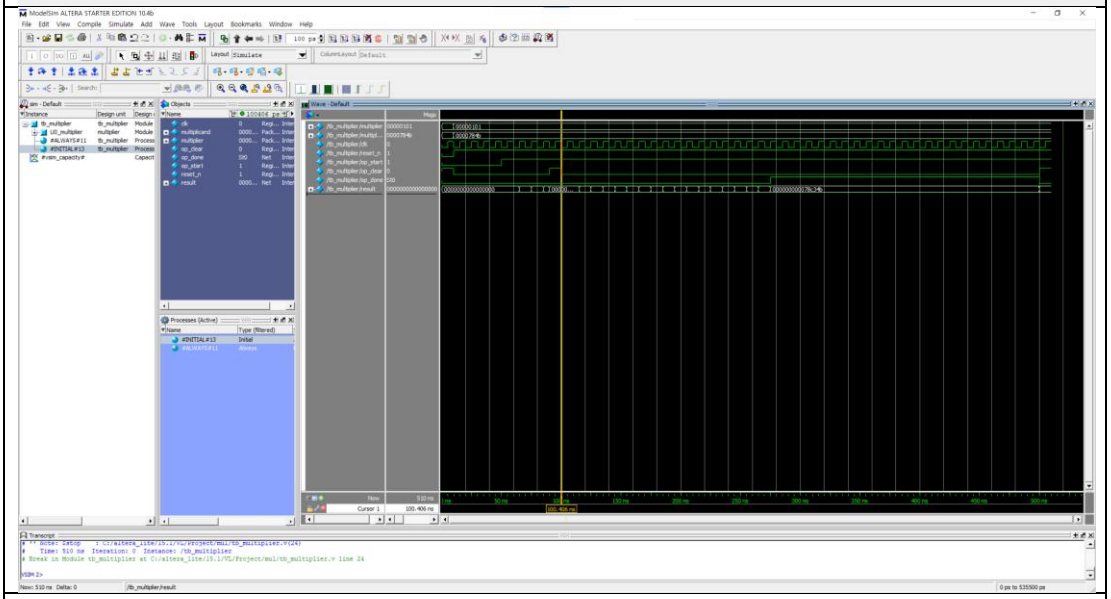
해당 모듈은 ALU_w_mul, ram을 bus 모듈을 통해 연결하여 testbench로 조작할 수 있도록 도와주는 가장 top module이다. 해당 모듈에서는 각 포트의 이름에 맞게 연결해주었다.

4. Design Verification strategy and results

multiplier

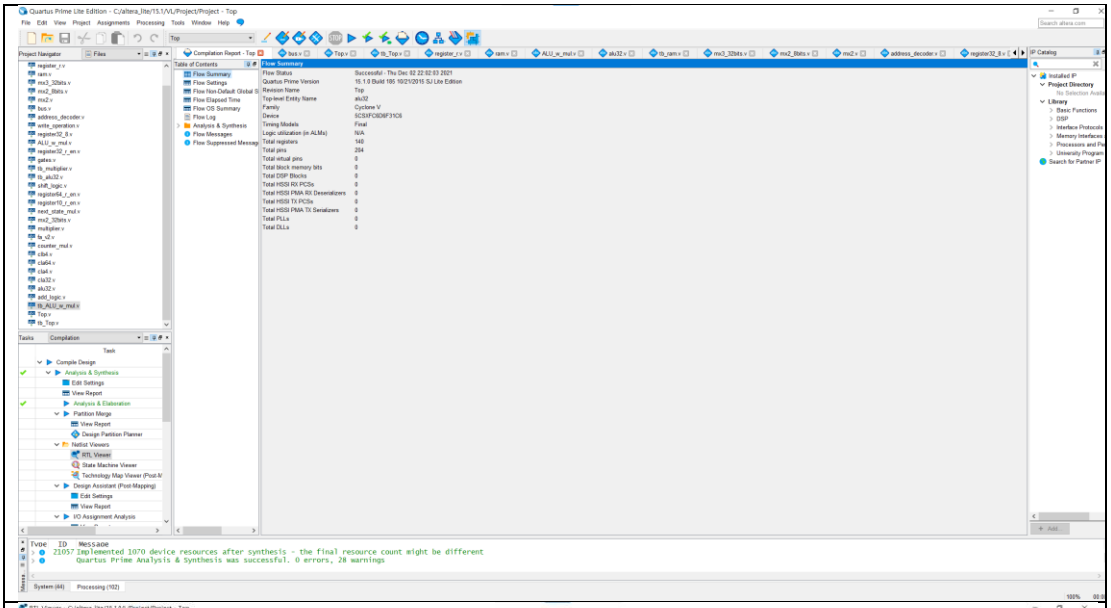


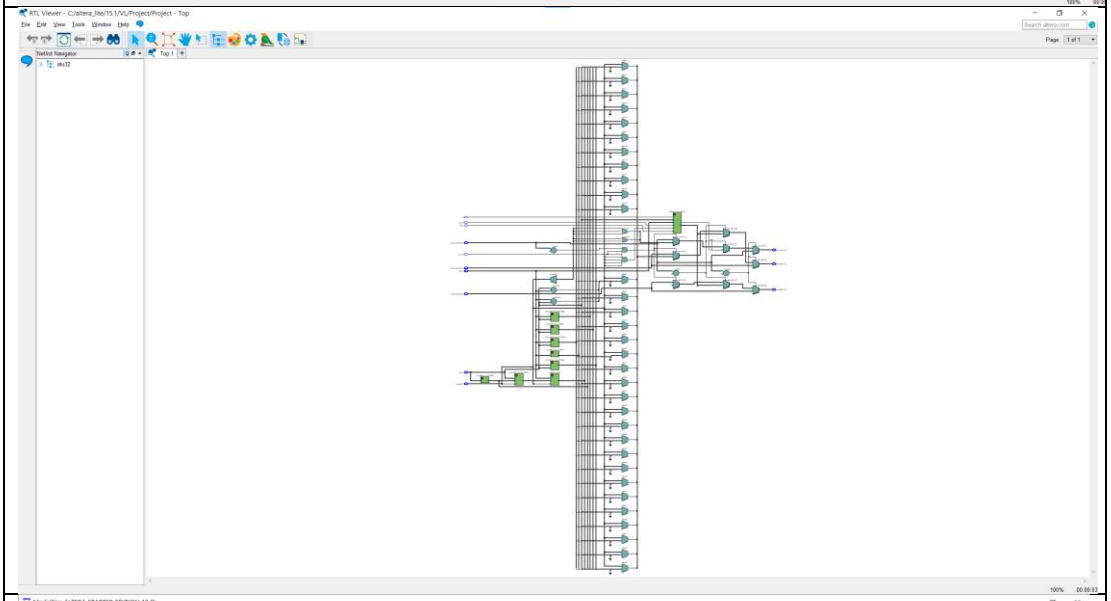


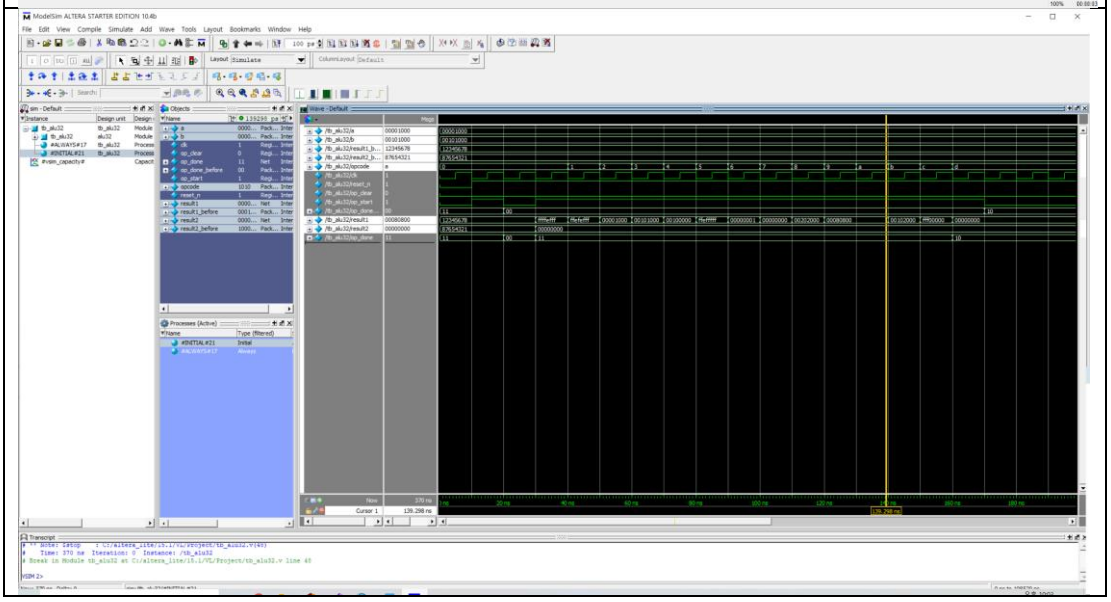


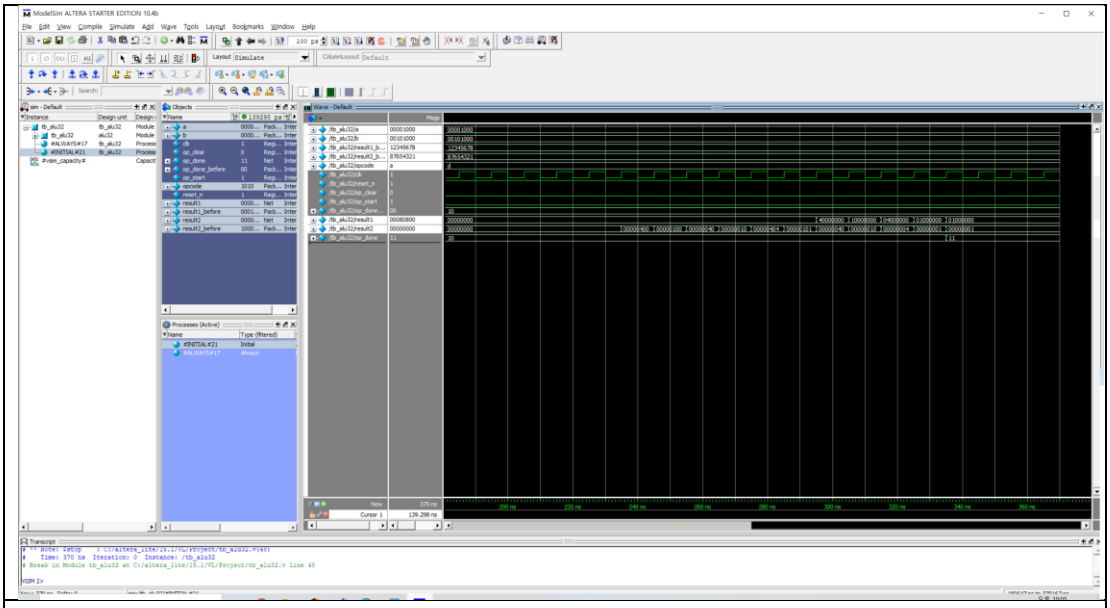
해당 testbench는 multiplier를 검증한 것이다. 해당 결과는 저번 과제에서 진행했던 testbench에서 값의 크기를 줄여서 검증을 해본 결과이다. 해당 결과는 이전과 같이 정확하게 값이 나오는 모습을 볼 수 있고, 초기화가 1일 경우 계산 중 초기화되는 모습까지 볼 수 있다.

alu32



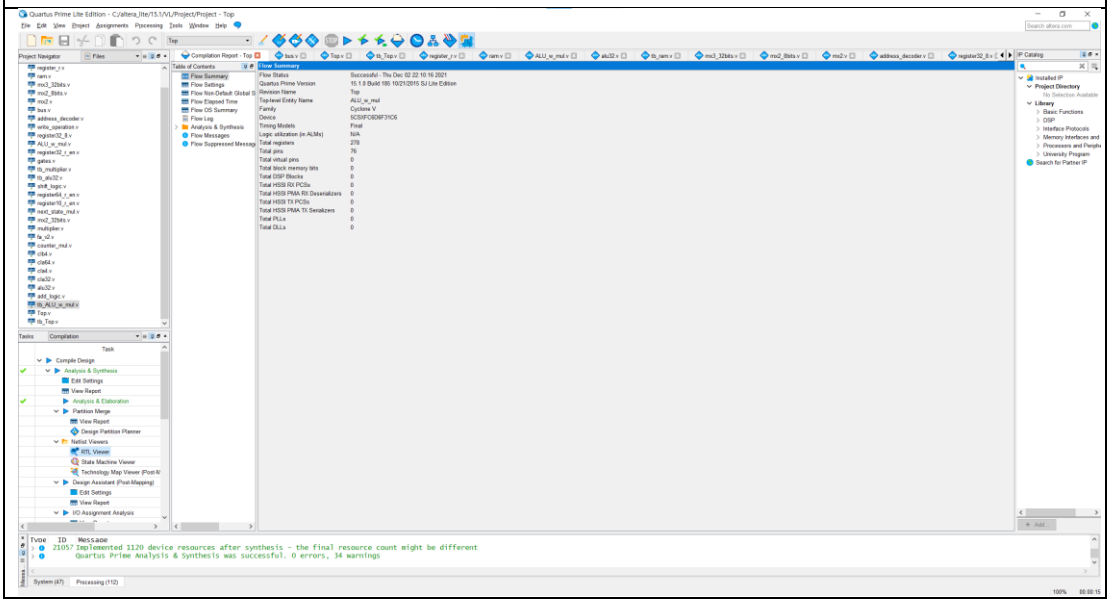


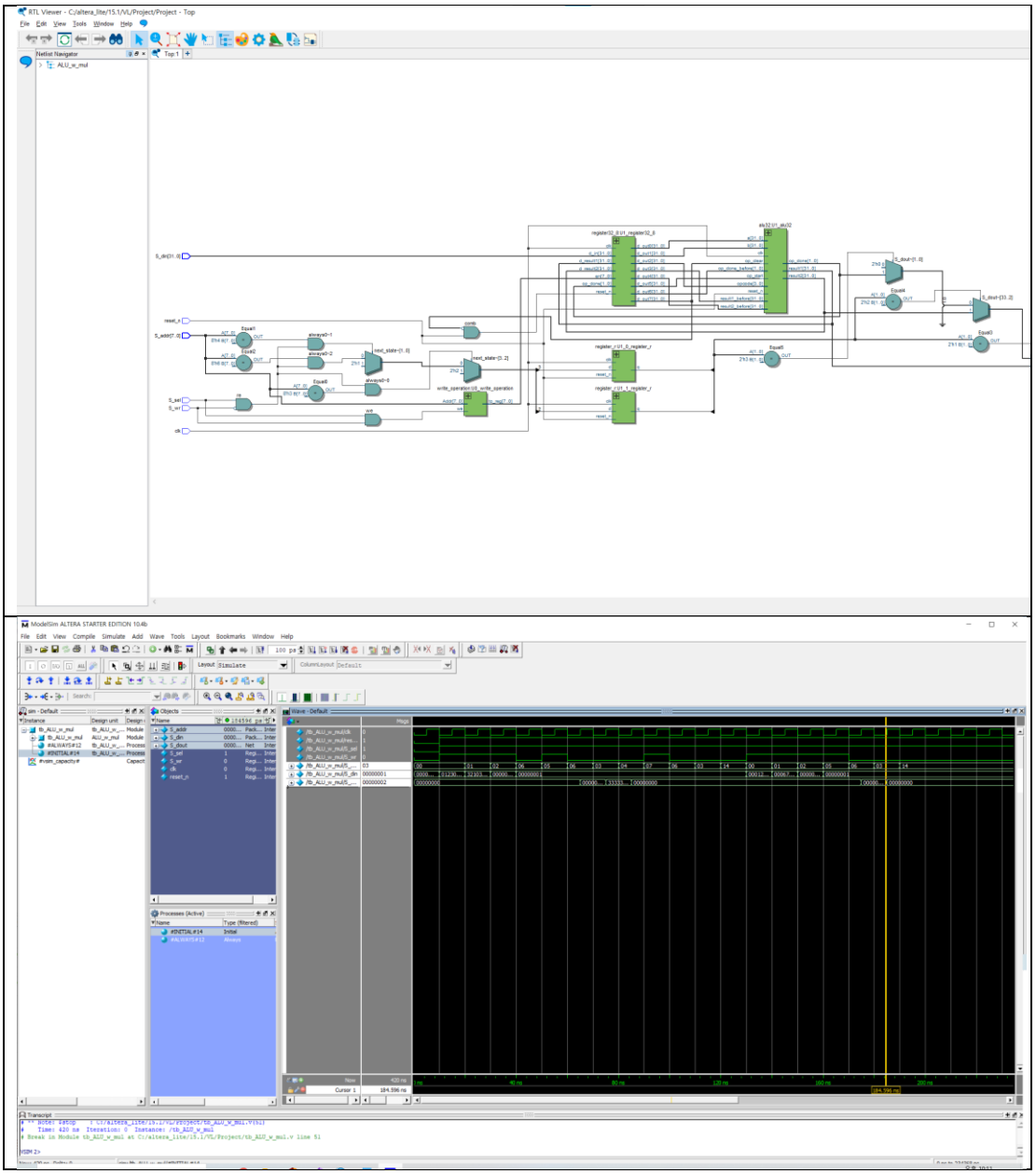


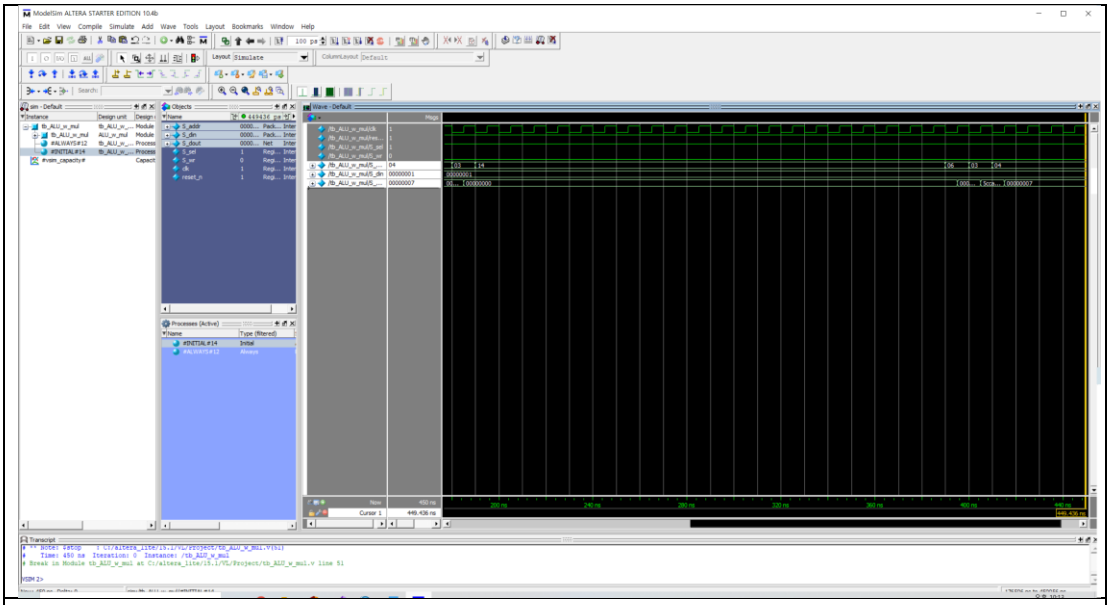


해당 testbench는 alu32를 검증한다. 위의 결과를 살펴보면 각 주소의 값에 따라 operand와 opcode ,start, clear가 들어가고 들어갔을 때 해당하는 동작이 제대로 동작하는 것을 확인할 수 있다. 특히 이전 op_done이 11일 때에는 start가 1이어도 동작하지 않고, op_done이 00일 때만 동작이 가능하다. 그리고 각 인자에 맞는 연산 또한 이루어지는 모습을 볼 수 있다. 곱셈 또한 동작 중에는 op_done = 10, 동작 완료 시 11을 출력하는 모습을 보인다.

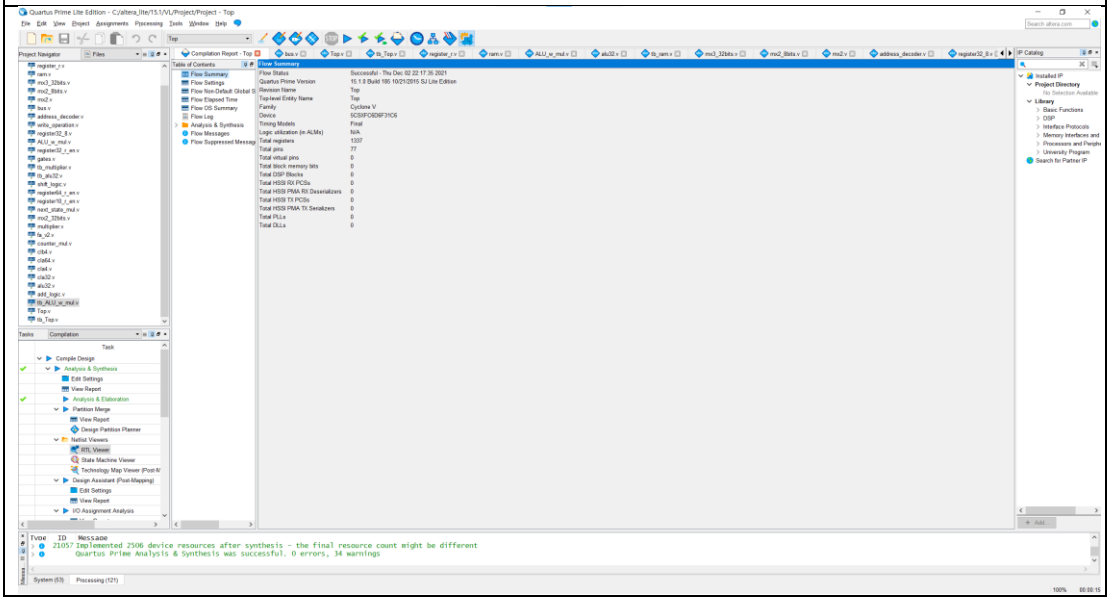
ALU_w_mul

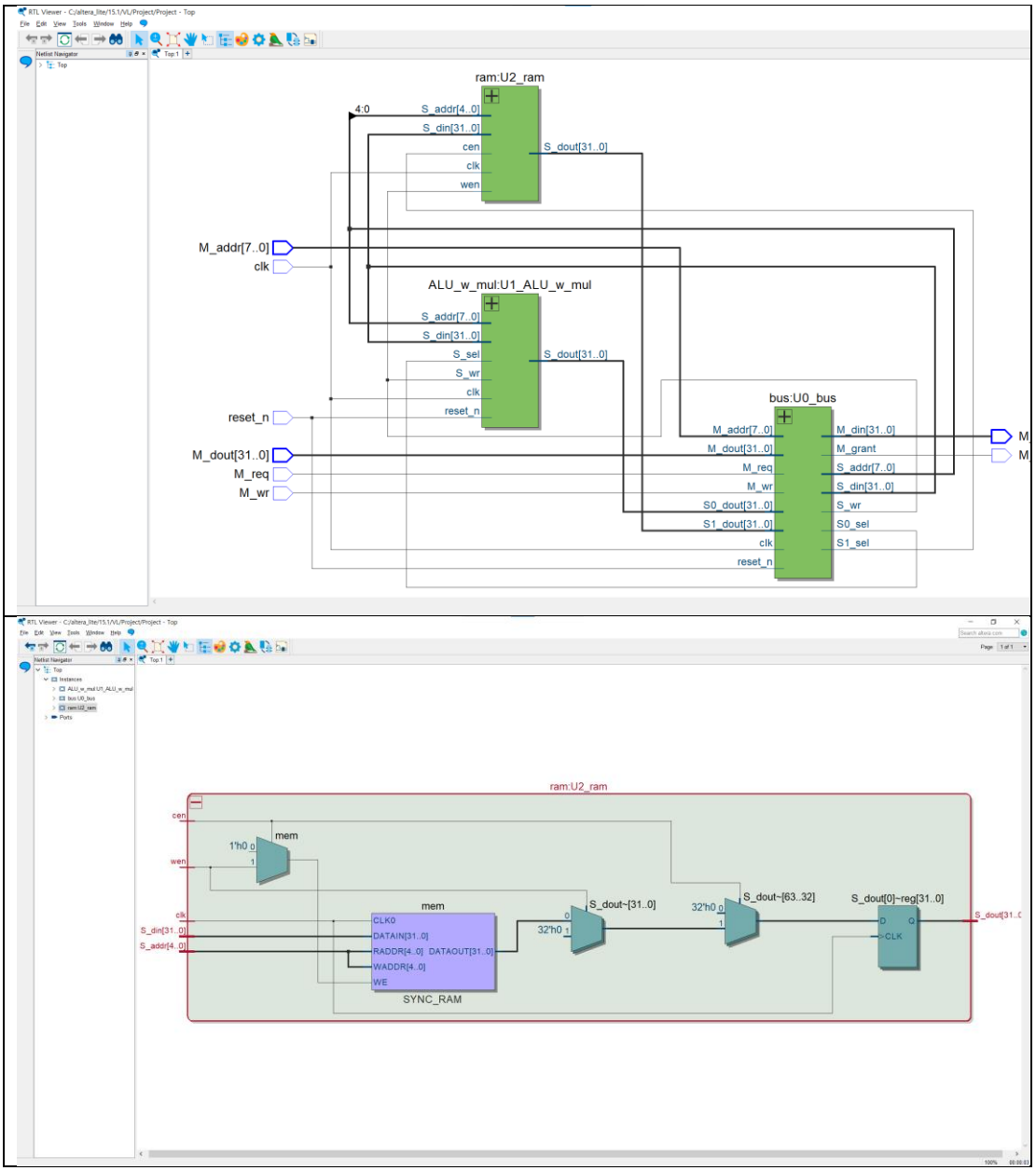




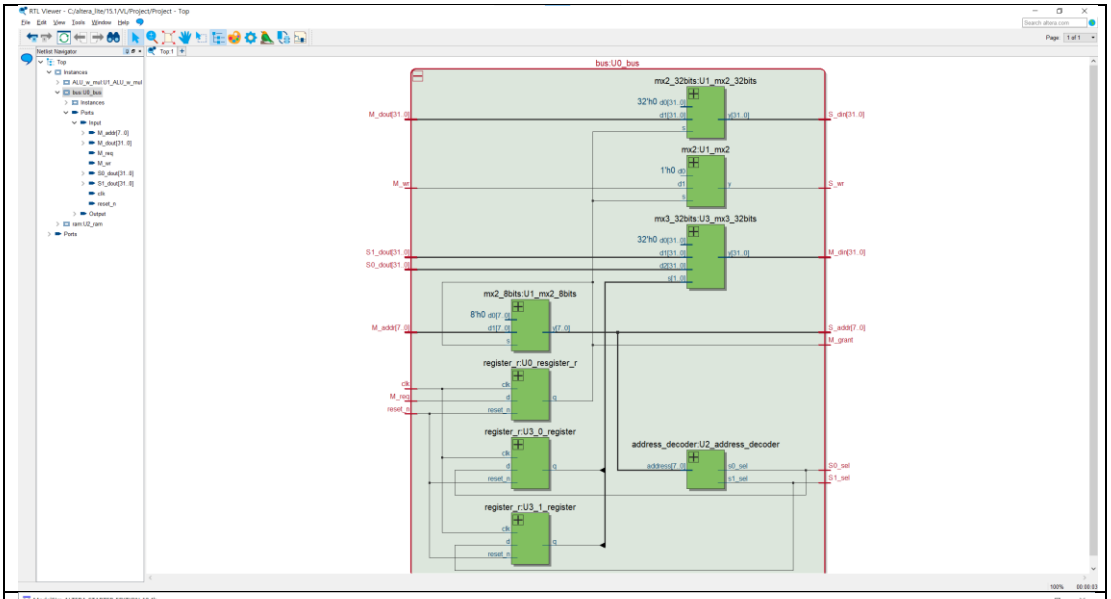


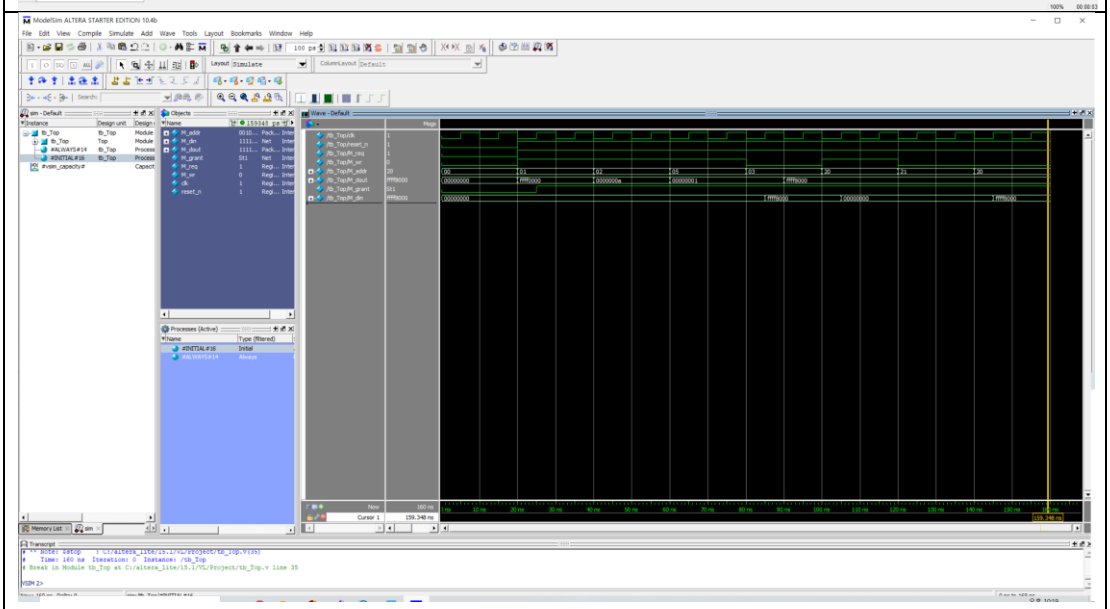
해당 Testbench에서는 ALU_w_mul을 확인한다. 해당 결과를 보면 레지스터에 각 값들이 제대로 들어가고 alu32에 제대로 연결되어있어 각 값이 전달되는 모습을 확인할 수 있다. 결과는 연산에 의한 1cycle과 저장에 의한 1cycle 즉 2cycle 후에 출력되는 모습 또한 확인할 수 있다.




ram:U2_ram
4:0
S_addr[4..0]
S_din[31..0]
cen
clk
wen
S_dout[31..0]
M_addr[7..0]
clk
ALU_w_mul:U1_ALU_w_mul
S_addr[7..0]
S_din[31..0]
S_sel
S_wr
clk
reset_n
S_dout[31..0]
reset_n
M_dout[31..0]
M_req
M_wr
bus:U0_bus
M_addr[7..0]
M_dout[31..0]
M_req
M_wr
S0_dout[31..0]
S1_dout[31..0]
clk
reset_n
M_din[31..0]
M_grant
S_addr[7..0]
S_din[31..0]
S_wr
S0_sel
S1_sel

ram:U2_ram
cen
wen
clk
S_din[31..0]
S_addr[4..0]
1'h0
mem
mem
CLK0
DATAIN[31..0]
RADDR[4..0]
DATAOUT[31..0]
WADDR[4..0]
WE
SYNC_RAM
S_dout~[31..0]
32'h0
S_dout~[63..32]
32'h0
S_dout[0]~reg[31..0]
D
Q
CLK





해당 TestBench에서는 Top을 검증한다. ram과 bus는 기존의 만든 모듈을 약간 변형하였기 때문에 따로 testbench를 만들지 않았다. 결과를 살펴보면 ALU 쪽으로 값을 넣고 result를 읽는 과정을 통해 Bus가 Slave0를 제대로 읽는 것을 확인할 수 있다. 또한 읽는 사이클은 한 Cycle 밀려서 읽히는 것 또한 정확히 동작한다고 볼 수 있다. 그리고 ram에 접근하여 데이터를 읽고 쓰는 동작 또한 제대로 실행되고 있다.

5. Conclusion

위와 같은 회로 설계를 통해 기존에 공부한 여러가지 방법을 통해 프로젝트를 완수할 수 있었다. 하지만 제작 과정에서 cycle에 맞지 않게 ALU의 값이 출력되지 않는 상황이 나타났다. 해당 사유는 FSM을 통해 state에 따라 값을 출력하지 않고 바로 출력했기 때문에 나타났던 오류였다. 이를 고치기 위해 register와 always문을 활용하여 출력의 타이밍

을 조정해줌으로써 이를 해결할 수 있었다. 해당 과제를 통해 미리 FSM을 통해 설계를 하고 코드를 작성하였다면 이러한 오류를 줄일 수 있지 않을까라는 생각을 할 수 있었다. 해당 과제를 진행하면서 기존에 만들어졌던 모듈들을 활용할 수 있지 않을까라는 생각이 들었다. 그리고 실제로 해당 모듈에서 기존의 ALU, multiplier, ram, bus, rf 모듈을 적극적으로 활용하여 해당 코드를 작성하였다. 이를 보고 module화 시키는 것이 다른 코드로 확장시키기에 유용하다는 점 또한 느꼈다.