# 컴퓨터 공학 기초 실험2 보고서

## 실험제목: Multiplier

실험일자: 2021년 11월 01일 (월)

제출일자: 2021년 11월 14일 (일)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

# 1. 제목 및 목적

## A. 제목

Multiplier

## B. 목적

해당 수업을 통해 Multiplier에 대해 이해할 수 있다. booth algorithm에 대해 이해하여 이를 기반으로 이를 Multiplier의 구조를 이해할 수 있다. FSM을 통해 해당 모듈을 설계할 수 있다.

# 2. 원리(배경지식)

i. Booth algorithm

booth's algorithm은 2's compliment로 이루어진 두 수를 곱하는 곱셈 알고리즘이다.

해당 알고리즘은 승수와 피승수, 그리고 곱하는 인자의 2배의 크기의 결과로 이루어져 있다. 피승수의 패턴을 확인하여 덧셈 또는 뺄셈을 진행하고 피승수와 결과에 shift를 진행하는 방식으로 진행된다. 해당 보고서에서는 radix-4 방식으로 해당 알고리즘을 설명하겠다.

우선 피승수의 마지막 2가지 패턴과 이전 패턴의 가장 앞을 확인한다. 해당 패턴의 구성에 따라 아래와 같은 덧셈을 진행한다.

| x(i+1) | x(i) | x(i-1) | 연산 |
|---|---|---|---|
| 0 | 0 | 0 | 없음 |
| 0 | 0 | 1 | + 피승수 |
| 0 | 1 | 0 | + 피승수 |
| 0 | 1 | 1 | + 피승수 * 2 |
| 1 | 0 | 0 | - 피승수 * 2 |
| 1 | 0 | 1 | - 피승수 |
| 1 | 1 | 0 | - 피승수 |
| 1 | 1 | 1 | 없음 |

연산에 있는 값을 결과의 1/2 지점의 비트와 더한다. 그 후 해당 프로그램은 2개씩 패턴을 읽고 있음으로 ASR shift 2 연산을 result와 피승수에 진행한다.

해당 연산은 피승수의 비트 / 2의 cirle로 진행되며 만약 연산이 끝나고 shift만

남은 경우를 예외처리를 통해 더욱 짧은 cycle로 회로를 동작하게 할 수 있다.

## 3. 설계 세부사항

i. cla64

해당 모듈은 64bit의 덧셈 연산을 진행하는 module이다. 해당 모듈은 기존에 제작하였던 cla32 2개를 이용하여 올림수를 넘겨줌으로써 제작할 수 있었다.

ii. add_logic

해당 모듈은 현재 result의 값에 multiplier를 multiplicand와 이전 패턴을 토대로 덧셈과 뺄셈을 진행한다. 해당 모듈에서는 cla64를 통해 각각의 패턴에 따른 연산을 result[127:64]와 진행하여 준비한다. 그 후 always 문을 통해 각 패턴에 따라 next_result의 값을 적절하게 대입함으로써 회로를 구성한다. next_result[63:0] 비트는 result의 값을 그대로 대입한다.

iii. register10_r_en

해당 모듈은 10개의 레지스터를 포함하는 회로다. 해당 모듈은 이전에 제작한 register8_r_en을 토대로 제작하였다.

iv. register64_r_en

해당 모듈은 64개의 레지스터를 포함하는 회로다. 해당 모듈은 이전에 제작한 register8_r_en을 토대로 제작하였다.

v. register128_r_en

해당 모듈은 128개의 레지스터를 포함하는 회로다. 해당 모듈은 이전에 제작한 register8_r_en을 토대로 제작하였다.

vi. shift_logic

ASR 2 연산 multiplicand와 result에 진행하는 모듈이다. 해당 모듈에서는 signed를 통해 정의된 input과 output을 가져야만 연산자 >>>를 진행하였을 때 arithmetic 연산이 정상적으로 수행된다. 만약 signed를 기입하지 않은 경우 단순한 LSR 2연산이 진행될 수 있다. 그 후 always 문을 사용하여 se가 1일 때에는 기존 값을 저장하고, se가 0일 때 shift 연산이 진행되도록 코드를 작성하였다.

vii. counter_mul

해당 모듈은 32부터 0까지 수를 count를 진행한다. 숫자 33개를 세기 때문에 6bit의 숫자가 필요하다. assign문을 통해 dec_count에서 1을 뺀 수를 미리 계산한다. 그 후 always문을 이용하여 state에 따라 next_count의 값을 적절하게 대입한다.

viii. next_state_mul

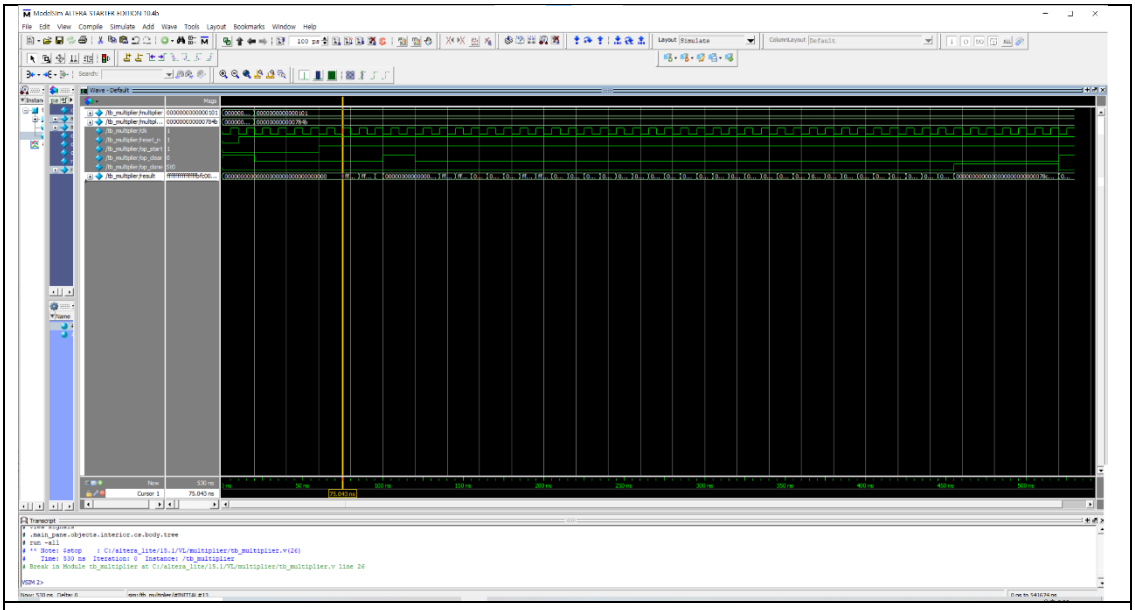
해당 모듈은 다음 상태와 input을 결정하기 위해 제작되었다. 해당 모듈은 always문을 활용하여 구성된다. 만약 기존 명령이 CLEAR일 경우 승수와 피승수를 초기화 하고 clear 신호가 0, start 신호가 1일 때만 start로 전환한다. 만약 START일 경우 승수와 피승수를 입력값으로 바꾸고 clear가 다시 1이 되지 않는 이상 DOING으로 이동한다. DOING일 때에는 기존 값을 그대로 유지하고 clear가 들어올 시 초기화, count가 0일 시 finish로 이동한다. FINISH일 때에는 현재 값을 그대로 유지한 체로 done을 1로 바꾼다. 그리고 clear가 1일 시 초기화, 아닐 시 현 상태를 그대로 유지한다.

ix. multiplier

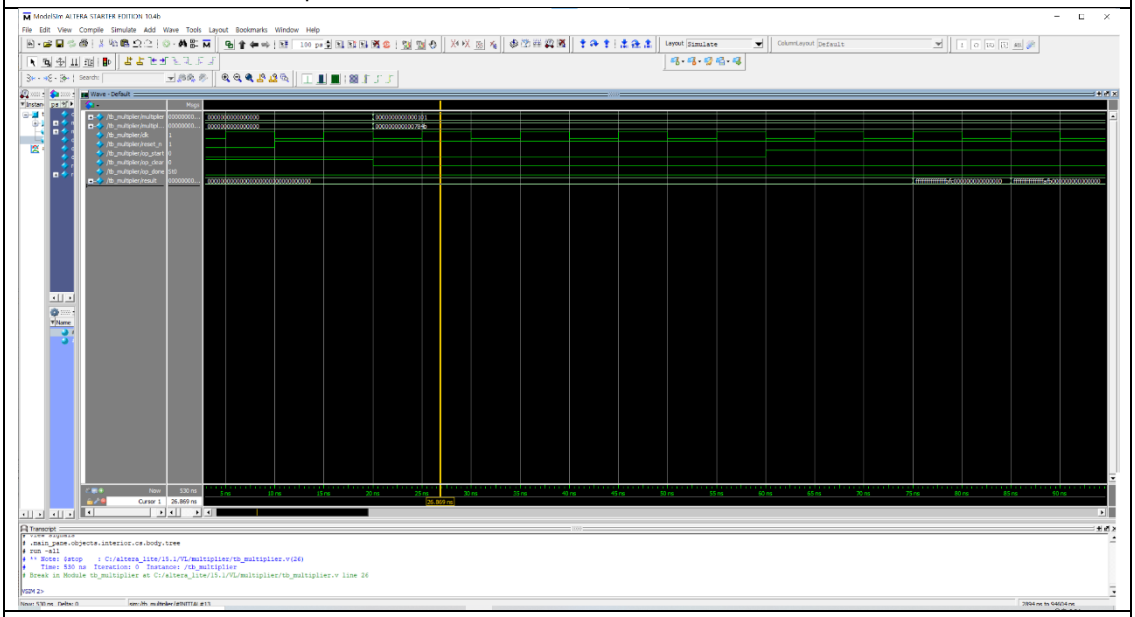
해당 모듈은 multiplier를 최종적으로 구현하는 모듈이다. 해당 모듈에서는 next_state_mul을 통해 다음 state를 계산하고, counter를 통해 숫자를 카운트한다. U2~U5는 데이터를 저장하는데 필요한 레지스터들이 모여있다. add_logic과 shift_logic을 통해 연산을 진행하고 이를 레지스터에 보내는 것을 반복한다.

## 4. 설계 검증 및 실험 결과

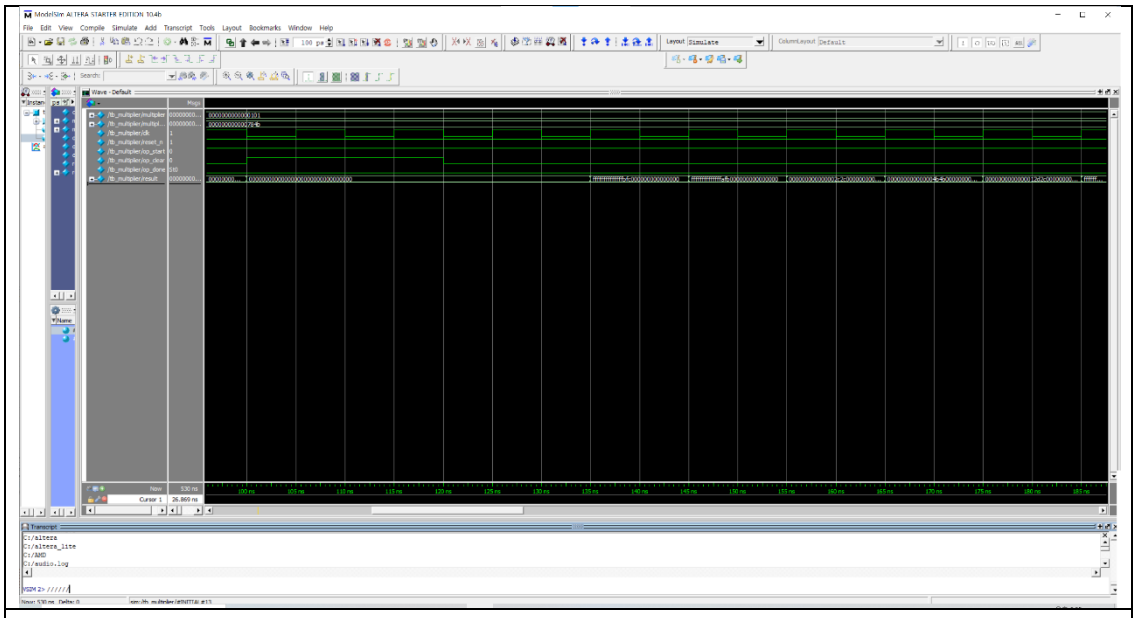
### A. 시뮬레이션 결과



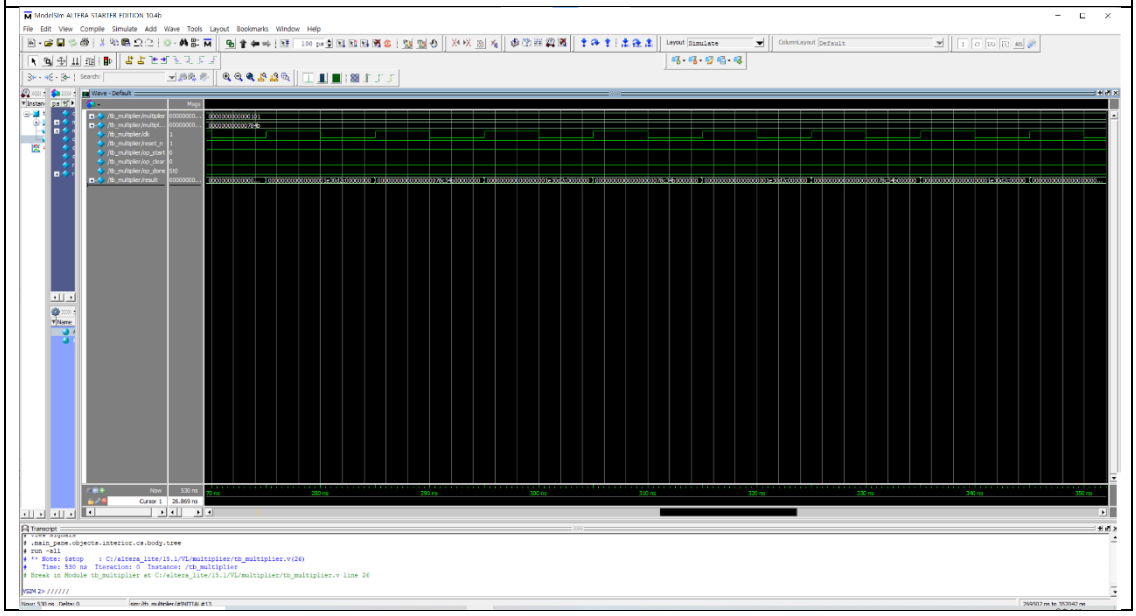
해당 testbench는 multiplier를 검증한다. 해당 사진은 전체 사진이다.

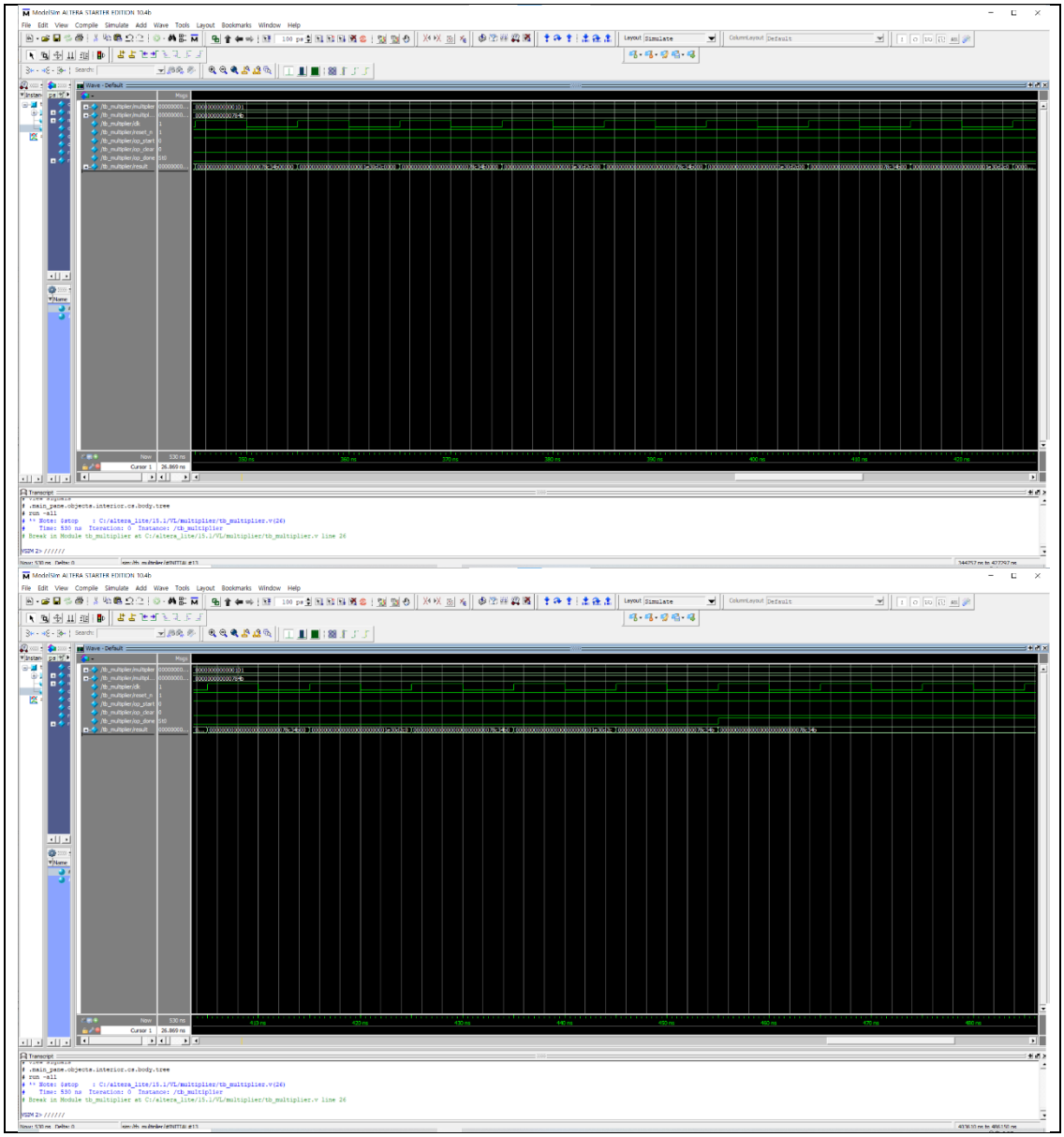


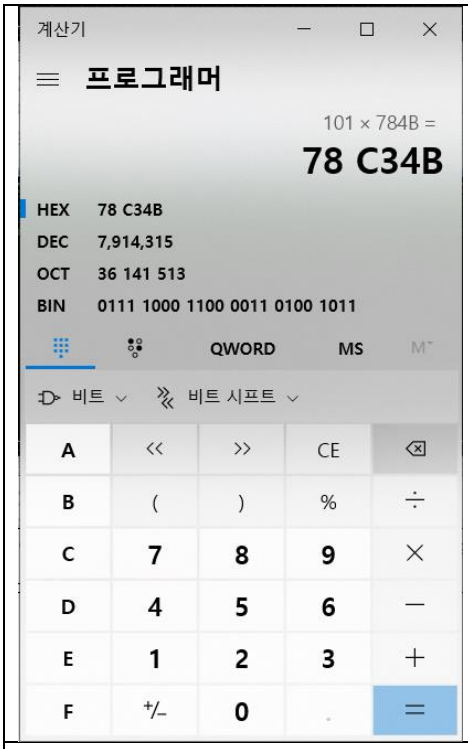
처음에는 clear 상태일 때 start가 들어오지 않으면 초기에 아무런 값도 받지 않는 모습을 보인다. start가 들어오면 연산을 시작한다.



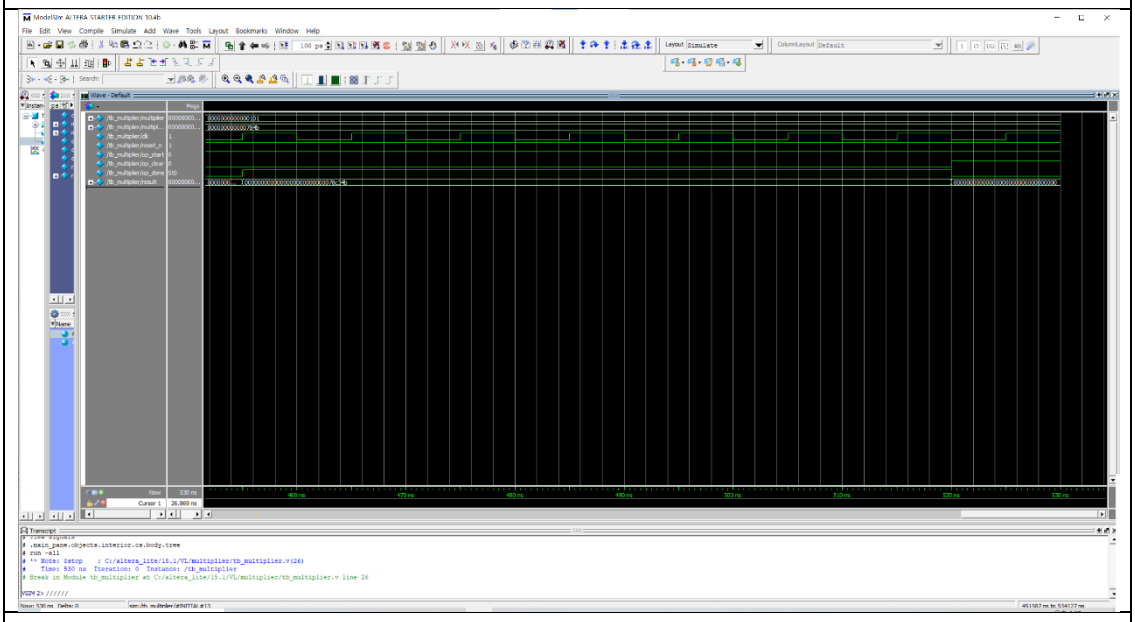
만약 clear가 다시 들어오면 기존 값을 다 지운다. 그리고 start를 통해 다시 값을 받는다.




ModelSim ALTERA STARTER EDITION 10.4b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
/tb_multiplier/multiplier
/tb_multiplier/multiplicand
/tb_multiplier/clk
/tb_multiplier/reset_n
/tb_multiplier/op_start
/tb_multiplier/op_clear
/tb_multiplier/op_done
/tb_multiplier/result
Cursor 1
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: $stop : C:/altera_lite/15.1/ML/multiplier/tb_multiplier.v(26)
# Time: 530 ns Iteration: 0 Instance: /tb_multiplier
# Break in Module tb_multiplier at C:/altera_lite/15.1/ML/multiplier/tb_multiplier.v line 26
VSIM 2>
ModelSim ALTERA STARTER EDITION 10.4b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
/tb_multiplier/multiplier
/tb_multiplier/multiplicand
/tb_multiplier/clk
/tb_multiplier/reset_n
/tb_multiplier/op_start
/tb_multiplier/op_clear
/tb_multiplier/op_done
/tb_multiplier/result
Cursor 1
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: $stop : C:/altera_lite/15.1/ML/multiplier/tb_multiplier.v(26)
# Time: 530 ns Iteration: 0 Instance: /tb_multiplier
# Break in Module tb_multiplier at C:/altera_lite/15.1/ML/multiplier/tb_multiplier.v line 26
VSIM 2>



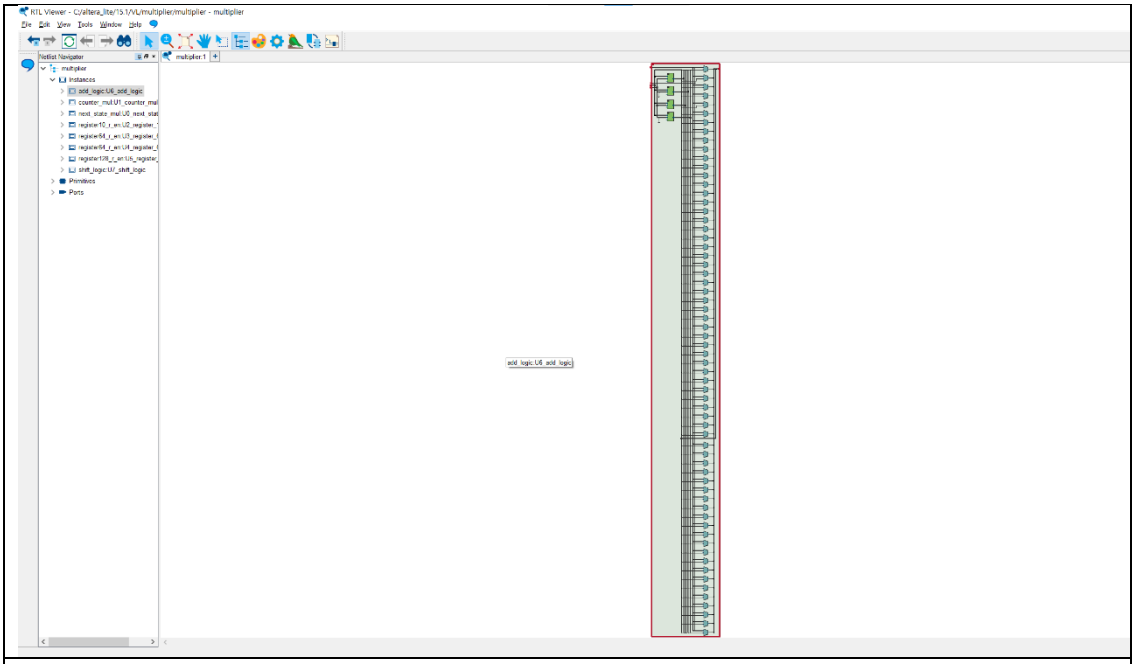

해당 과정을 통해 ASR이 정상적으로 진행되고 덧셈 또한 정상적으로 진행되는 것을 알 수 있다. 연산이 끝나고 난 후에는 op_done 신호가 1로 나오는 모습 또한 보인다. 또한 결과도 일치한다.



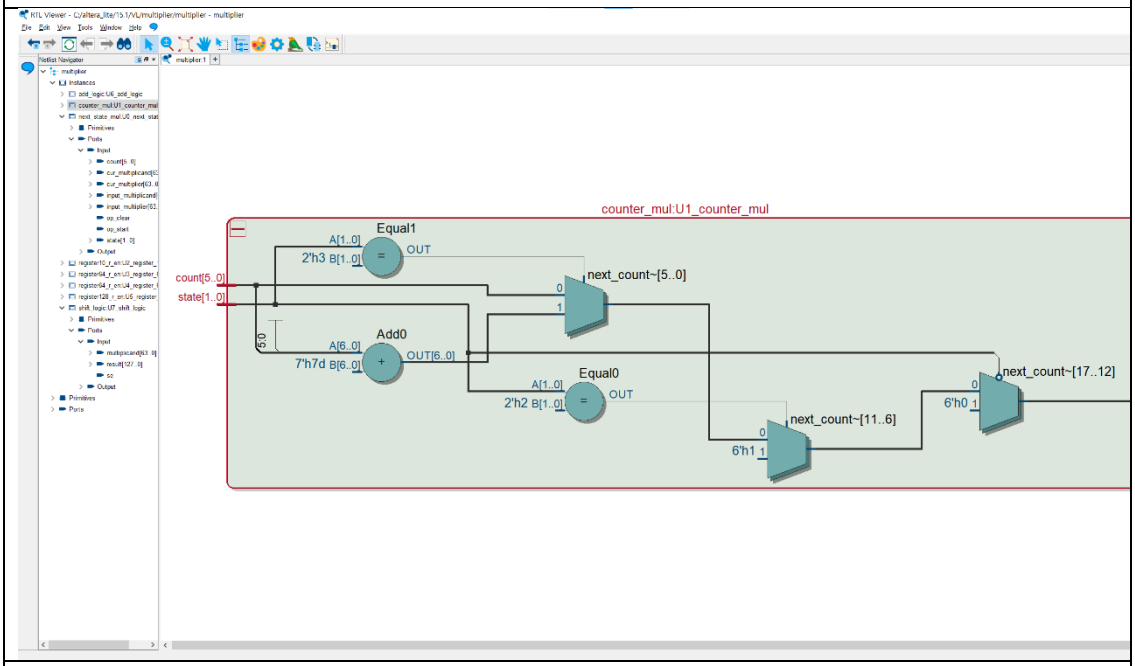
끝난 상태에서 초기화를 진행할 수 있는 모습을 보인다.

## B. 합성(synthesis) 결과

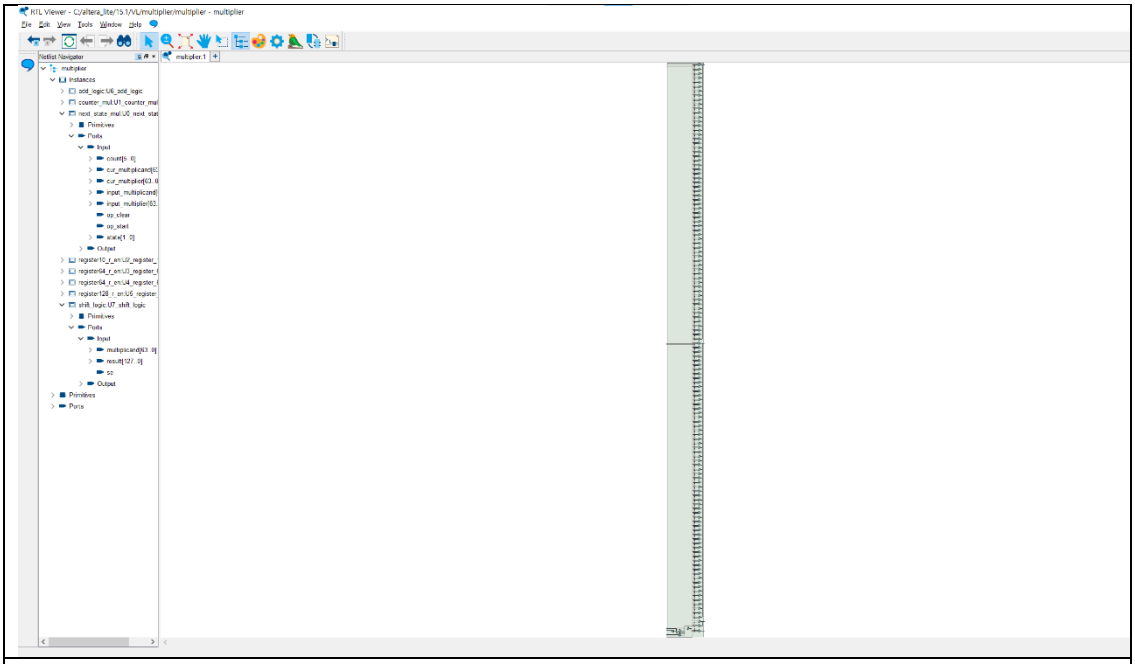
add_logic



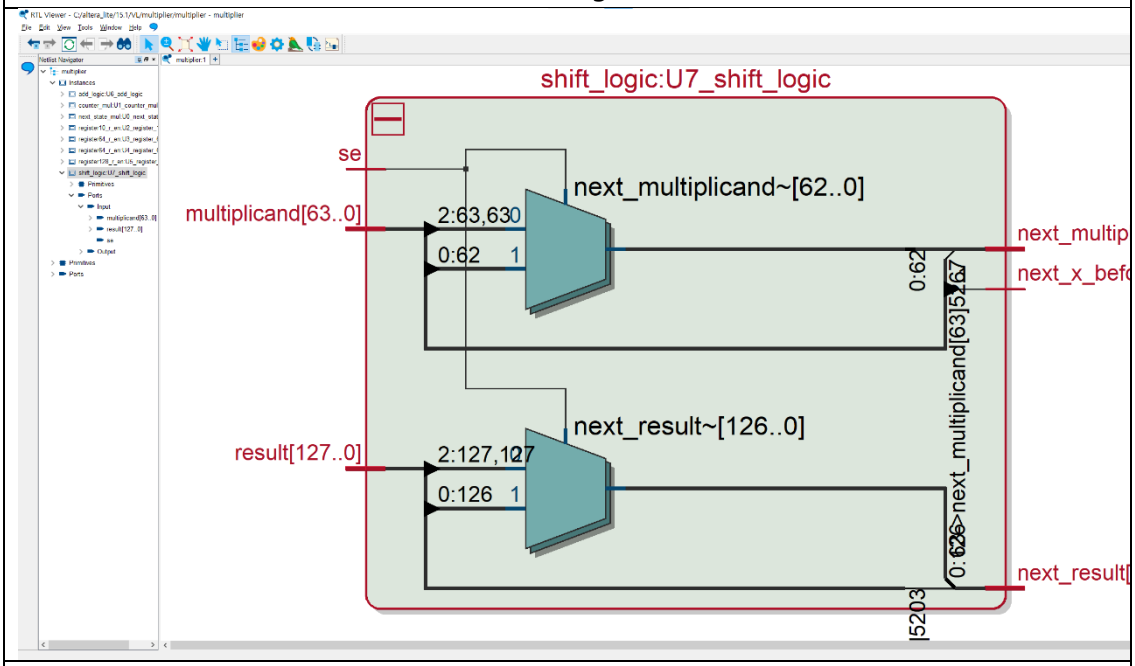
counter_mul



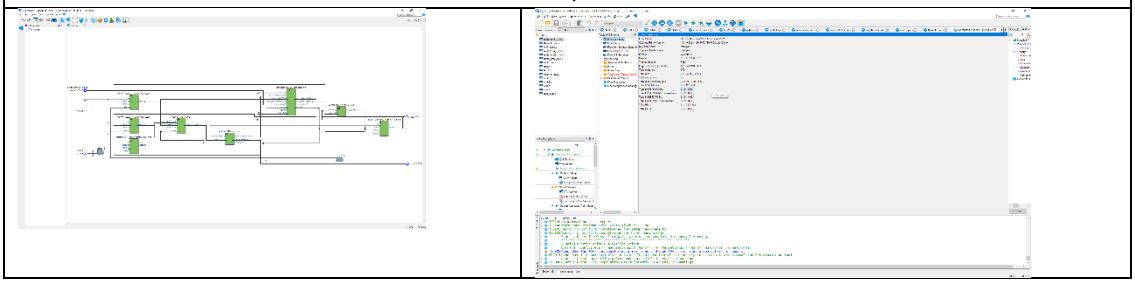

next_state_mul



shift_logic




multiplier



해당 모듈의 설계를 확인하면 각각의 모듈이 정상적으로 설계되어 연결되었음을 알 수 있다.

### C. FPGA board targeting 결과

## 5. 고찰 및 결론

### A. 고찰

이번 주차의 과제를 진행하면서는 기존에 모듈의 양식을 주지 않고 우리가 직접 구현하는 방식이기 때문에 아무래도 설계 과정에서 여러가지 오류들을 고쳐 나아가야 했다. 또한 multiplier의 연산과정 또한 내용이 어려운 부분이 있어 이를 다시 이해하는데 시간이 소요되었다. 결과적으로 radix-4 방식을 사용해서 구성해볼 수 있어서 의미있는 시간이었다.

### B. 결론

제작한 Radix-4 booth multiplier는 101010과 같은 수를 곱셈을 진행할 때 덧셈이나 뺄셈이이 여러 번 발생되지 않는 장점이 있다. 또한 2개의 수를 읽어가기 때문에 Cycle이 기존에 비에 1/2 줄어드는 효과가 있다. 단 현재 재작한 회로는 예외처리를 하지 않았기 때문에 무조건 정해진 사이클인 32회 연산을 수행해야 한다.

## 6. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021
이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021