컴퓨터 공학 기초 실험2 보고서

실험제목: Shifter & Counter

실험일자: 2021년 10월 18일 (월)

제출일자: 2021년 10월 30일 (토)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적

A. 제목

Shifter & Counter

B. 목적

해당 수업을 통해 Shifter에서의 동작(LSL, LSR, ASR)에 대해 이해할 수 있다. Counter의 동작 방식을 이해하고 loadable counter와 ring counter의 차이를 이해한다. FSM을 통해 Combinational Circuit과 Sequential Circuit으로 나누어 설계할 수 있다.

2. 원리(배경지식)

i. Moore FSM

Moore FSM은 쉽게 말해 output이 현재 state에만 의존하는 것을 의미한다. Moore FSM은 clk edge에 의해서만 output이 바뀌기 때문에 도중에 값이 바뀔 수 있는 Mealy FSM보다 안전하다. 또한 설계 단계에서 동작을 단순화하여 설명함으로서 더욱 쉽게 설계를 할 수 있다는 장점이 있다.

ii. Mealy FSM

Mealy FSM은 output이 현재 state와 output 둘 다 의존한다. 해당 방식을 사용할 경우 결과가 sequential circuit을 거치지 않기 때문에 더욱 빠른 동작이 가능하다. 또한 상태가 Moore보다 더 적게 설계가 가능하다.

iii. ring counter

ring counter란 Flip-Flop이 이어진 형태로 설계된 shift register를 의미한다. 레지스터의 값이 각각에 상황에 맞게 옆으로 전달되면서 값이 one-hot 방식으로 값을 표현한다. 해당 방식을 사용할 경우 각 값, 즉 state의 개수만큼 register가 필요하기 때문에 카운트하는 수가 많아질수록 register의 양으로 인한 회로의 단가가 올라가게 된다. 또한 초기값이 항상 0으로 고정되어 있다는 단점이 있다.

iv. loadable counter

해당 회로는 중간에 값을 변경할 수 있는 counter를 의미한다. 해당 회로는 굳이 one-hot 방식으로 설계하지 않아도 설계가 가능하다. 또한 초기 값을 직접 설정할 수 있다는 장점이 있다.

v. shifter

해당 회로는 주어진 비트를 좌 또는 우로 일정한 비트만큼 옮기는 역할을 한다. 좌로 옮길 때는 LSL로 부르며 좌측에 0를 추가하는 방식으로 가동한다. 우로 옮기는 방법은 LSR 또는 ASR이 있다. LSR은 우측으로 0을 삽입하는 방식으로 가동하고, ASR은 우측에 MSB의 비트를 삽입하는 방식으로 가동한다.

3. 설계 세부사항

i. mx4

해당 모듈은 4-to-1 MUX를 구현하였다. 해당 모듈에서 이전에 설계한 MX2를 활용하여 2자리 입력을 받아 4가지 경우를 구분할 수 있도록 했다.

ii. LSL8

해당 모듈은 LSL 8bits를 구현하였다. 해당 모듈에서는 mx4를 통해 각 자리에 shamt만큼 좌측으로 옮겨진 값을 연결하여 구현하도록 했다. 빈 자리는 0을 넣는다.

iii. LSR8

해당 모듈은 LSR 8bits를 구현하였다. 해당 모듈에서는 mx4를 통해 각 자리에 shamt만큼 우측으로 옮겨진 값을 연결하여 구현하도록 했다. 빈 자리는 0을 넣는다.

iv. ASR8

해당 모듈은 ASR 8bits를 구현하였다. 해당 모듈에서는 mx4를 통해 각 자리에서 shamt만큼 우측으로 옮겨진 값을 연결하여 구현하도록 했다. 빈 자리에는 MSB를 넣는다.

v. _register8_r

해당 모듈은 8bit register를 구현하였다. 해당 모듈에서는 _dff_r를 8개 연결하여 각 자리의 데이터를 저장할 수 있도록 하였다.

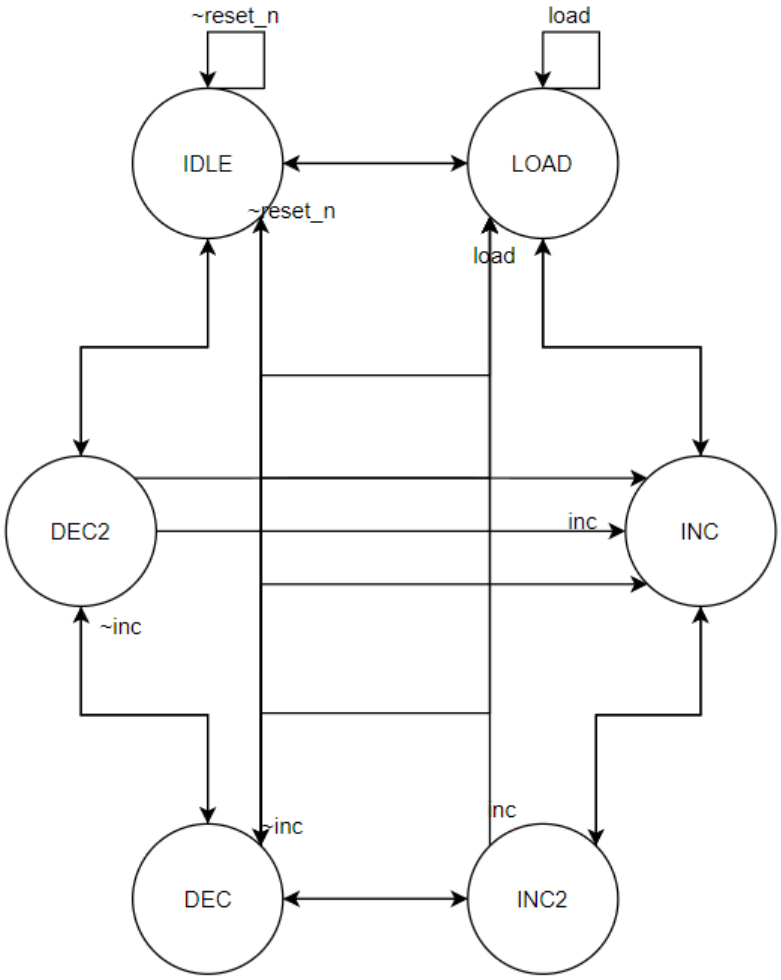
vi. cc_logic

해당 모듈에서는 shifter8의 combinational circuit 부분을 한 번에 구현한 회로다. 해당 회로에서는 각 상태를 parameter로 이름을 지어준 뒤 always문과 case문을 활용해 다음 상태를 넘겨주는 역할을 만들었고, LSL8, LSR8, ASR8 모듈을 이용하여 해당 값을 출력하는 역할을 수행하였다.

vii. shifter8

해당 모듈은 shifter를 실질적으로 구현한 회로다. 해당 모듈에서는 cc_logic을 통해 Combinational 부분을, _register8_r을 통해 sequential을 구현하고 이를 연결함으로써 FSM을 완성했다.

viii. Drawing the finite state diagram



ix. op

IDLE = 000

LOAD = 001

INC = 010

INC2 = 011

DEC = 100

DEC2 = 101

IDLE : d_out = 0000_0000

LOAD : d_out = d_in

INC : d_out = d_out+1

INC2 : d_out = d_out+1

DEC : d_out = d_out-1

DEC2 : d_out = d_out-1

x. cla8

해당 모듈은 8bit CLA Adder가 기존에 만든 cla4 2개를 이어 붙어서 구성하였다.

xi. ns_logic

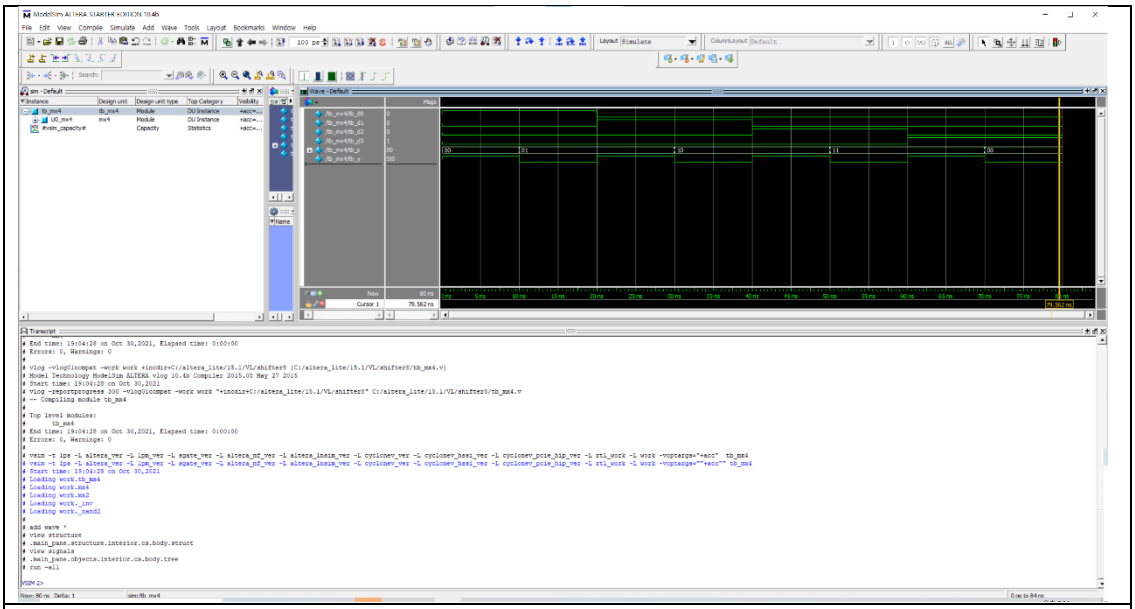
해당 모듈은 case문을 이용하여 지정된 현재 parameter와 load, inc에 따라 다음 상태을 바꿔줌으로써 회로를 구성한다.

xii. os_logic

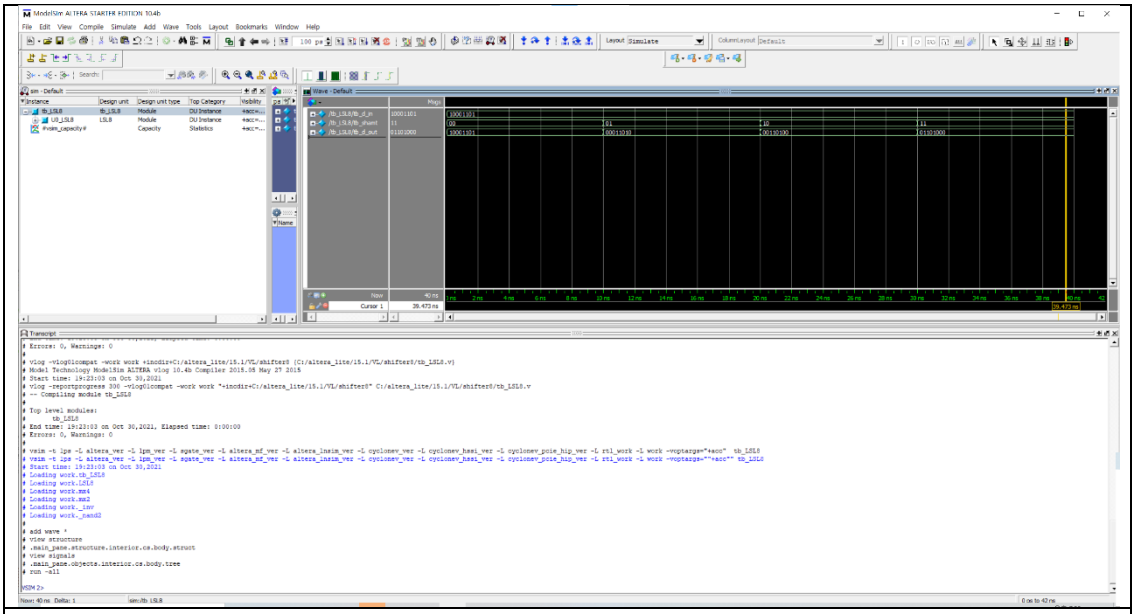
해당 모듈은 case문을 이용하여 현재 parameter에 따라 1을 더하는 cla, 1을 빼는 cla, 현재 값 3개 중 하나로 출력을 결정함으로써 회로를 구성하였다.

4. 설계 검증 및 실험 결과

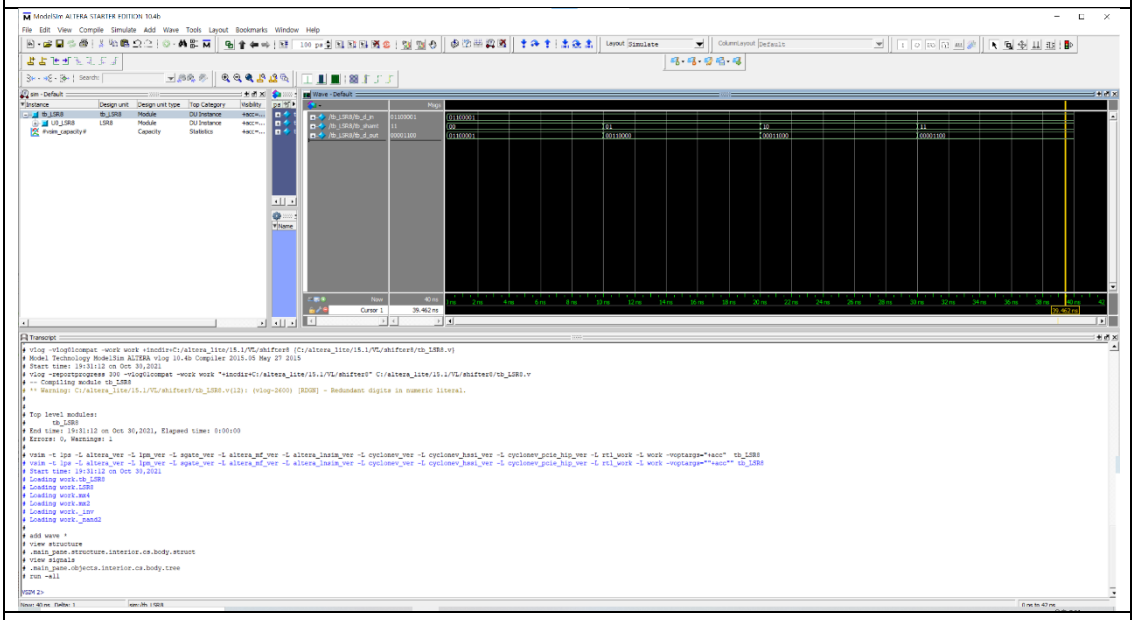
A. 시뮬레이션 결과



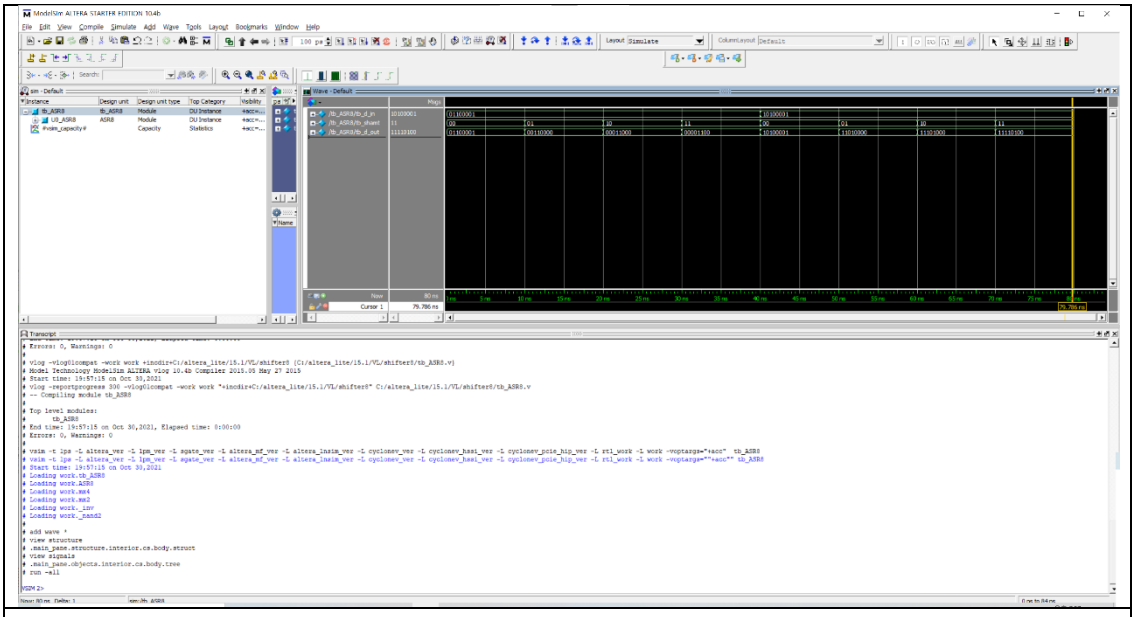
해당 testbench는 mx4를 검증한다.



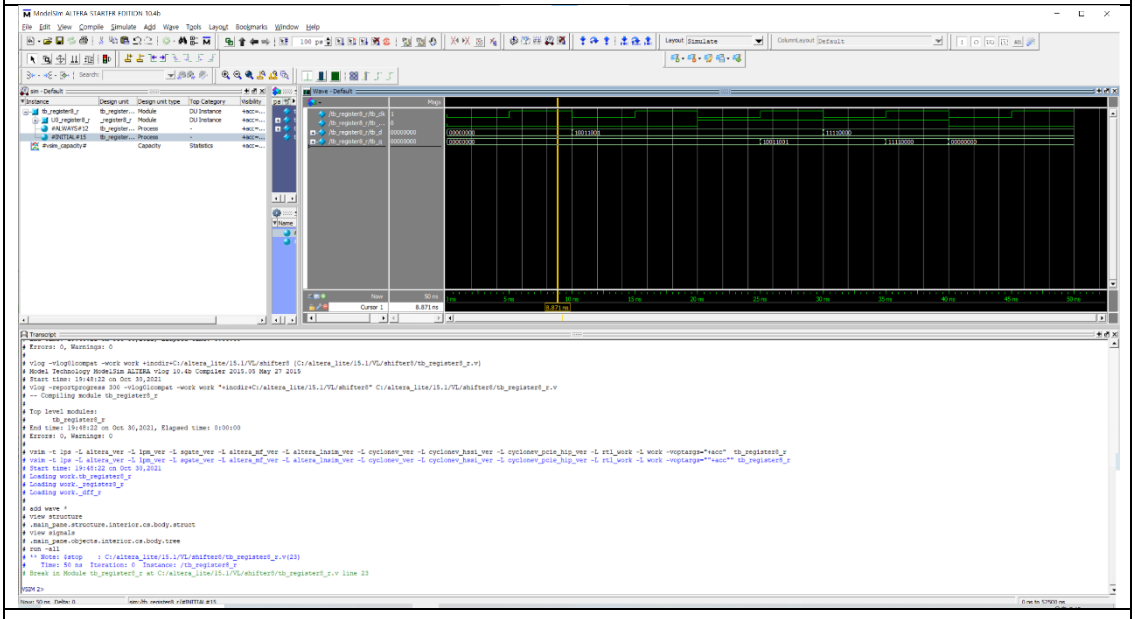
해당 testbench는 LSL8을 검증한다. shamt에 따라 값이 왼쪽으로 이동하는 것을 볼 수 있다.



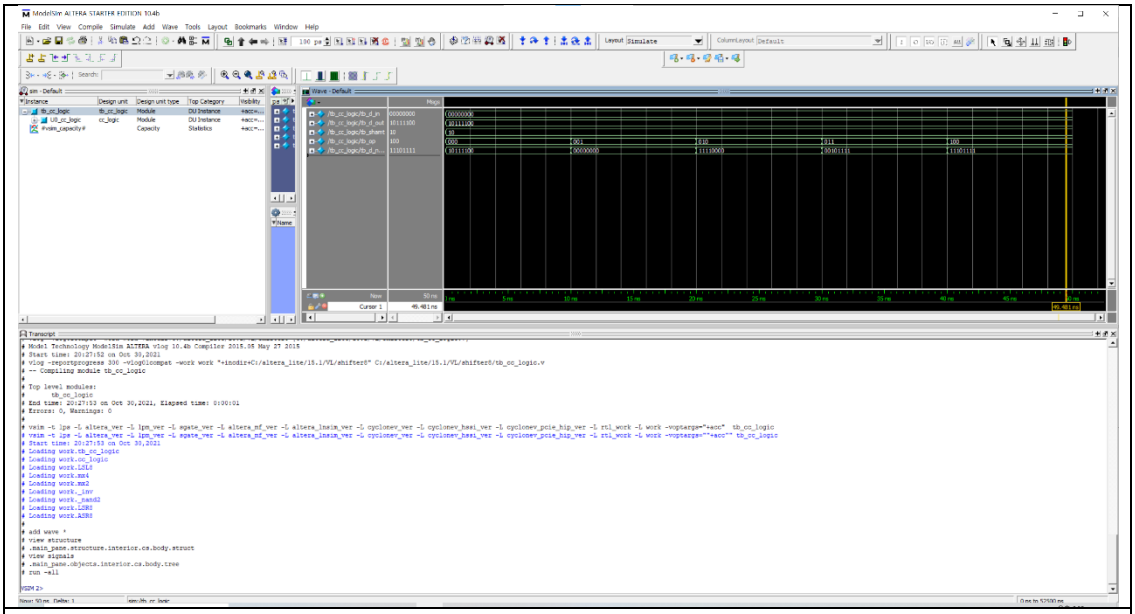
해당 testbench는 LSR8을 검증한다. shamt에 따라 값이 오른쪽으로 이동하고 0이 채워진다.



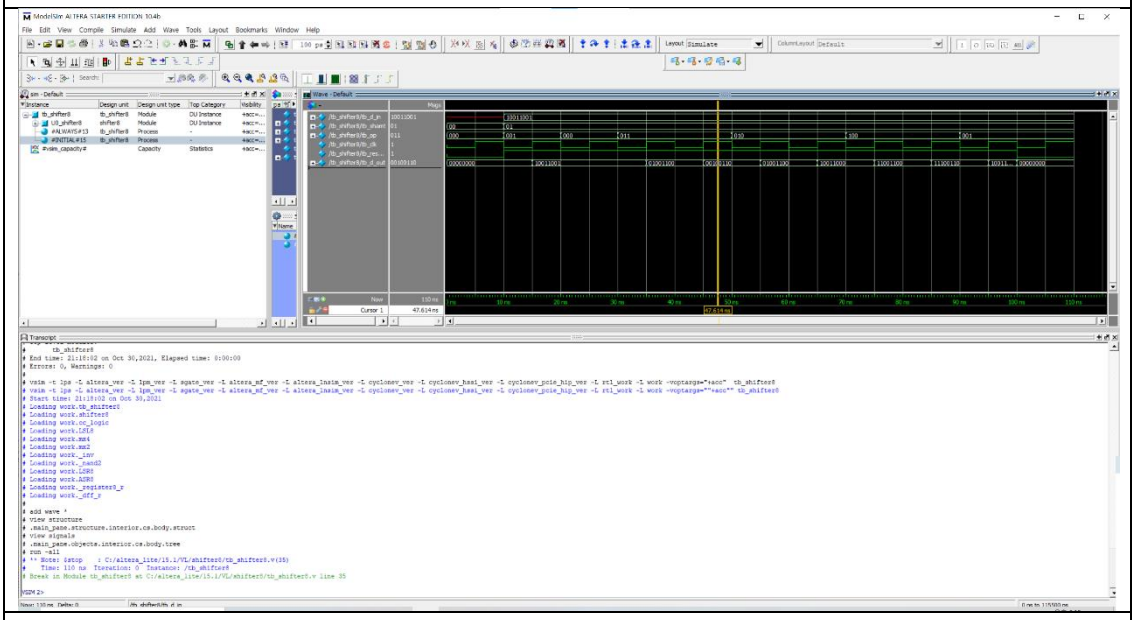
해당 testbench에서는 ASR8을 검증한다. shamt에 따라 값이 우측으로 이동하는 모습이다. 이 때 MSB의 값이 채워진다.



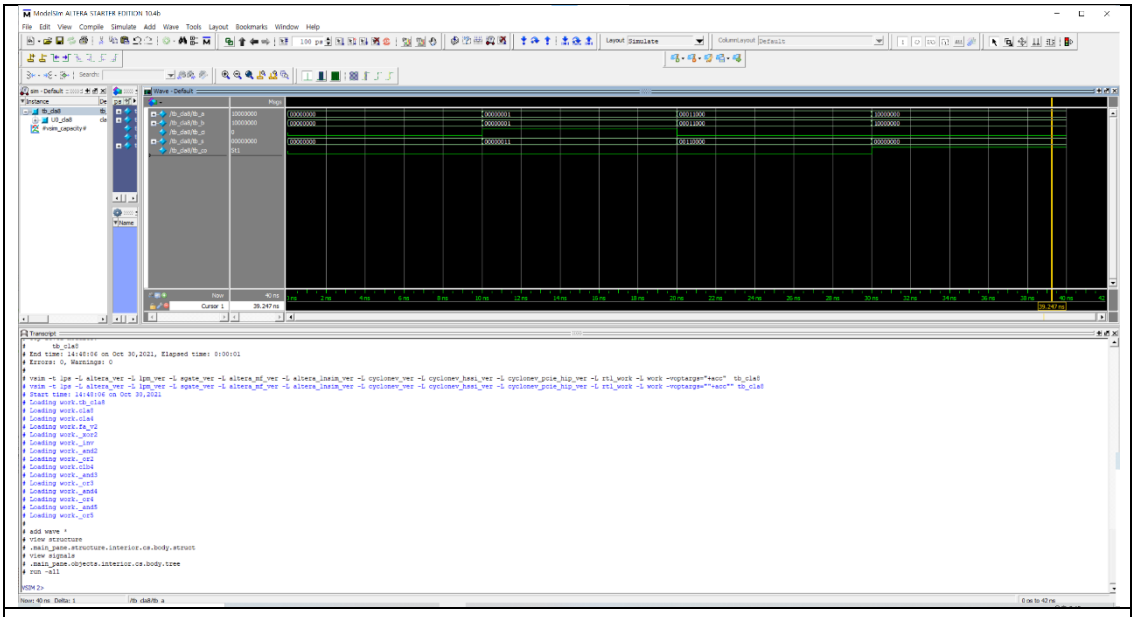
해당 testbench에서는 _register8_r를 검증한다. 정상적으로 모든 비트에 값이 들어가는 것을 볼 수 있다.



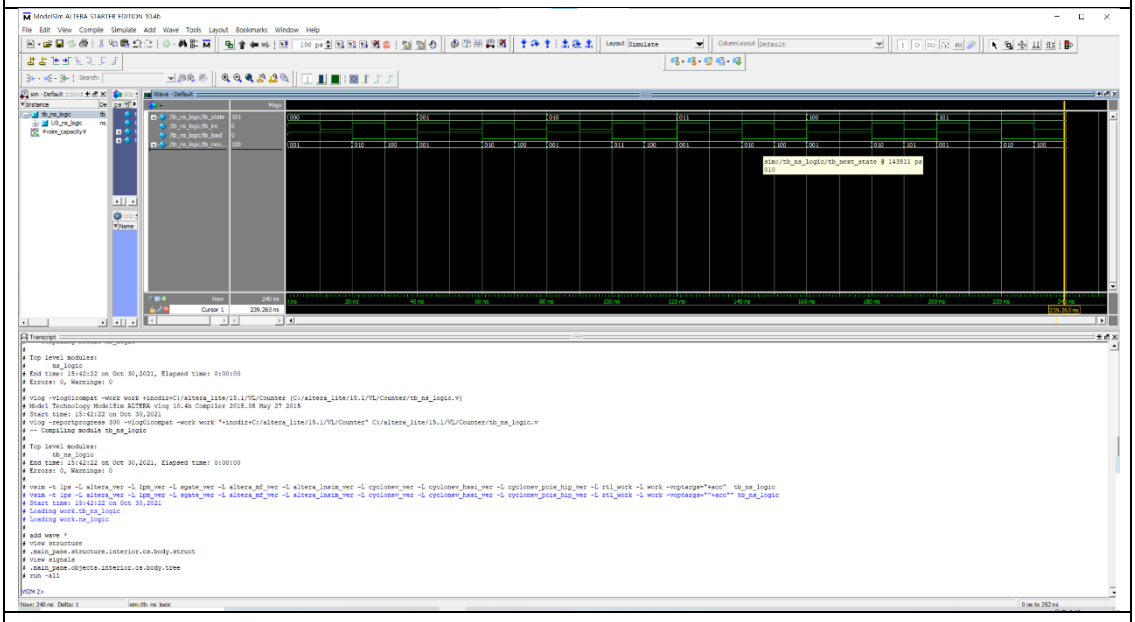
해당 testbench에서는 cc_logic을 검증하였다. 각 op마다 해당하는 shifter가 정상적으로 동작하는 것을 알 수 있다.



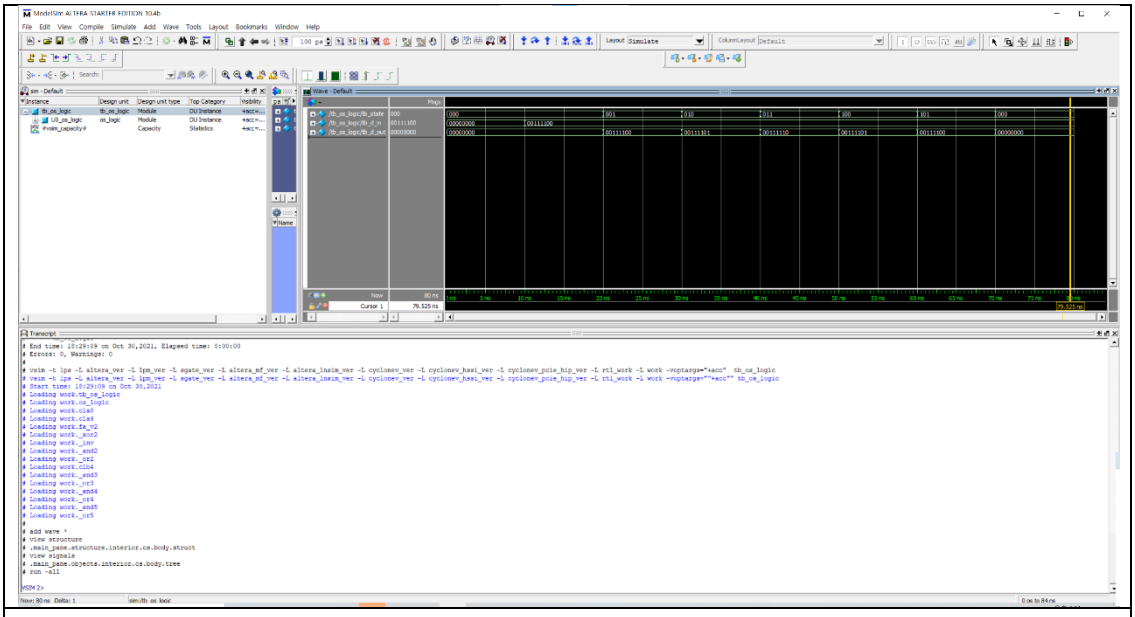
해당 testbench에서는 shifter8을 검증하였다. 현재 값을 활용하여 지속적으로 shift를 할 수 있음을 볼 수 있다.



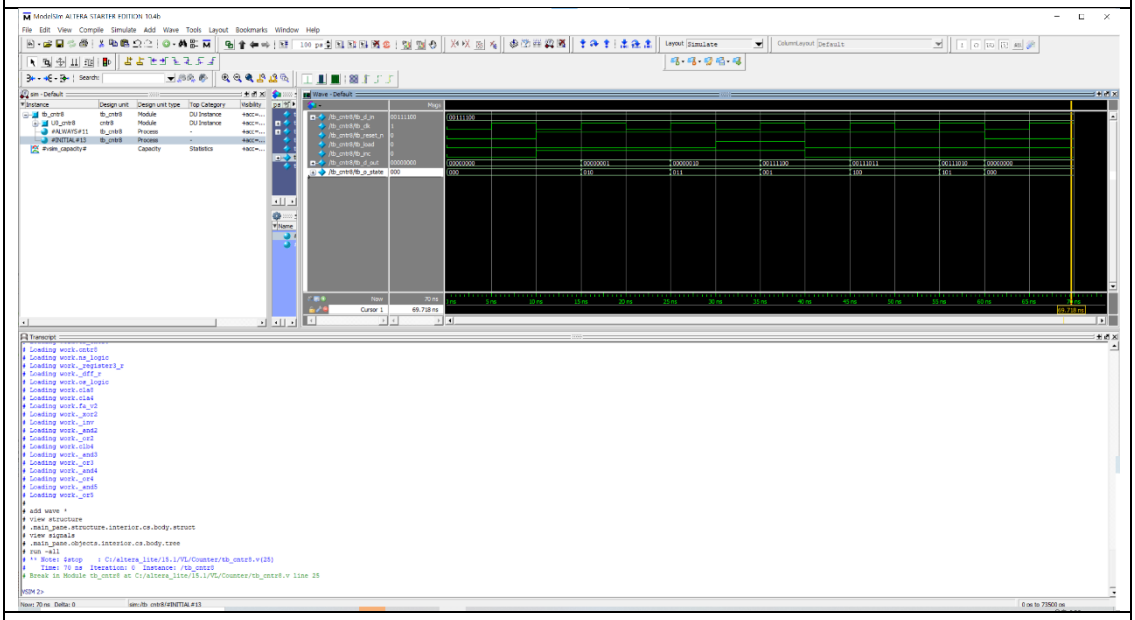
해당 모듈에서는 cla8을 검증한다. cla4 2개가 각각 잘 연결되어 있는 것을 확인할 수 있다.



해당 모듈에서는 ns_logic을 검증한다. 각 파라미터에서 load와 inc에 따라 값이 바뀌는 것을 알 수 있다.



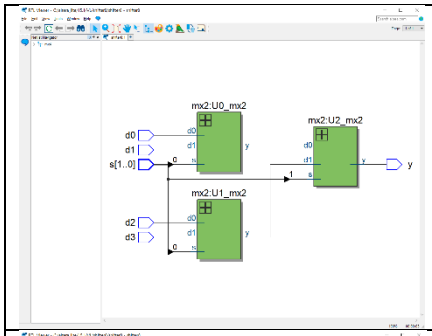
해당 모듈에서는 os_logic을 검증한다. os_logic에서는 현재 파라미터에 따라 cla나 현재 상태를 바꿔가면서 출력하는 과정이 정상적으로 나온다.

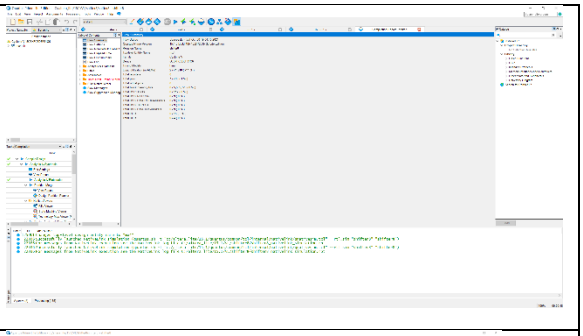


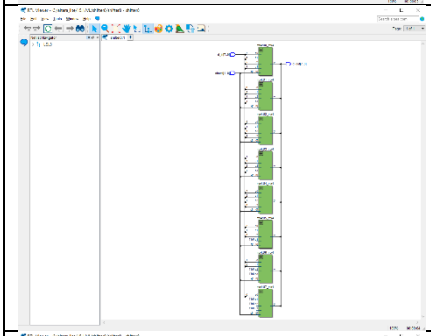
해당 모듈에서는 cntr8을 점증한다. 해당 회로에서 각 op의 값마다 정상적으로 값이 도출되는 것을 확인할 수 있다.

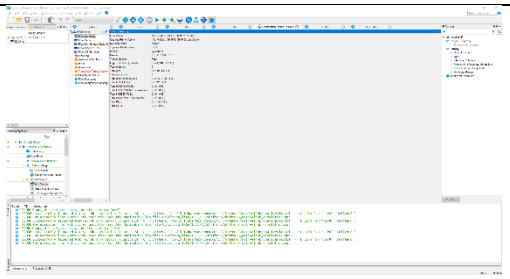
B. 합성(synthesis) 결과

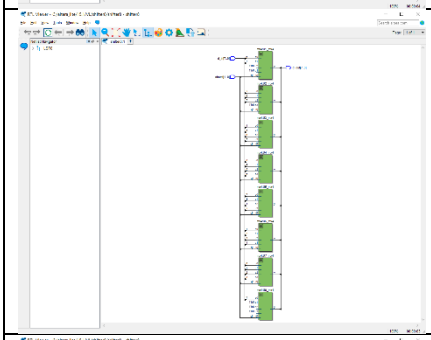
shifter8

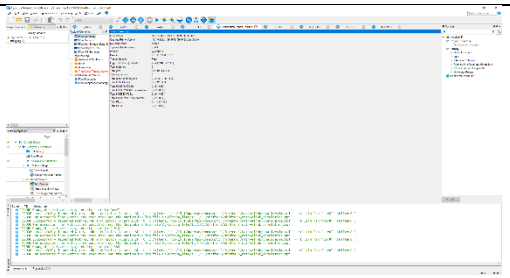

mx2:U0_mx2
mx2:U2_mx2
mx2:U1_mx2
d0
d1
s[1..0]
d2
d3
y

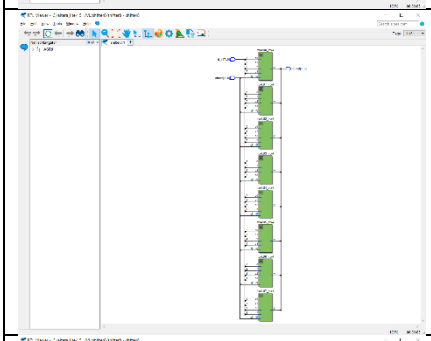


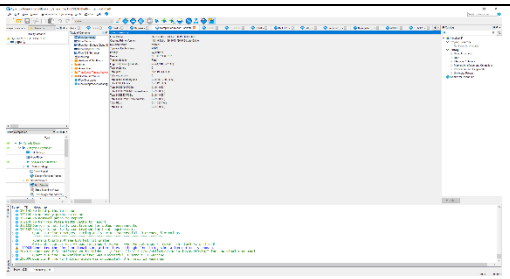


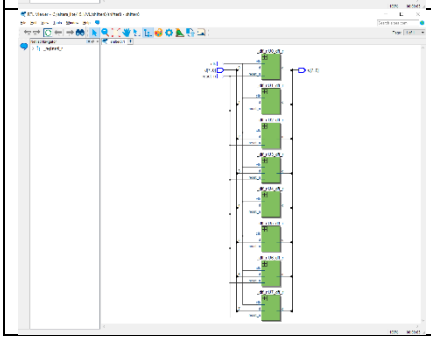


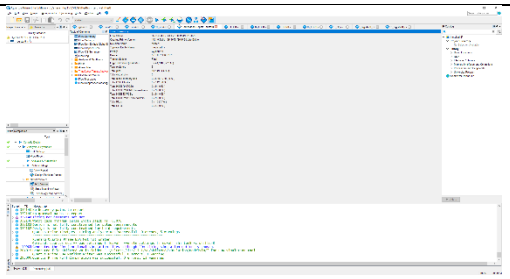


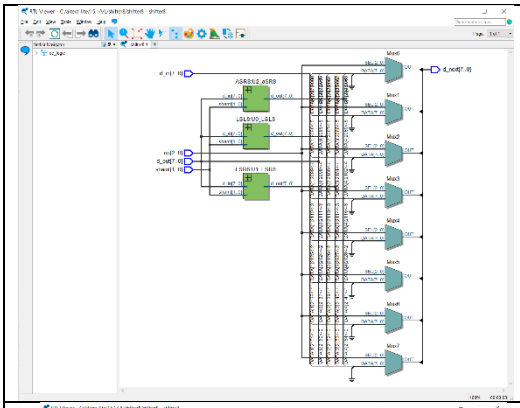


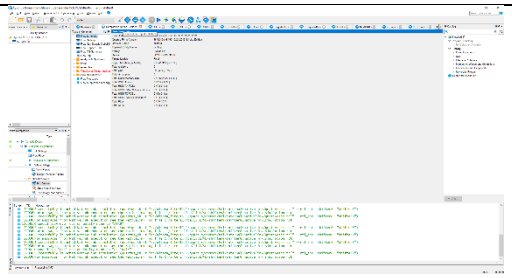


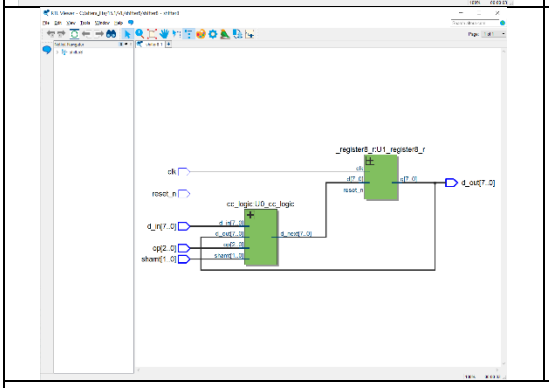


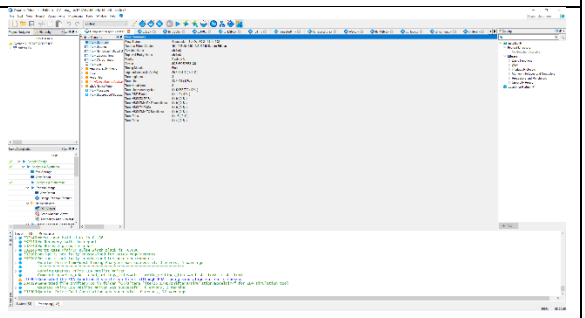




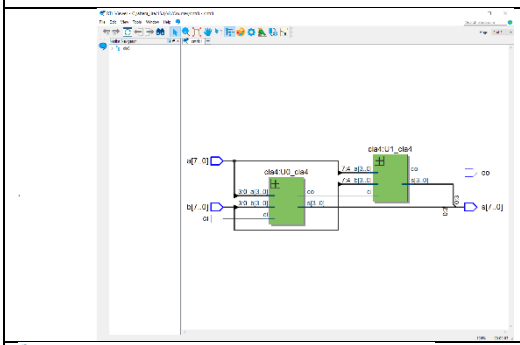


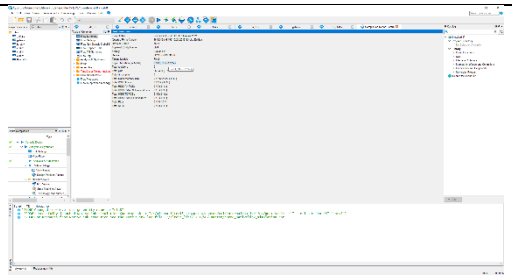


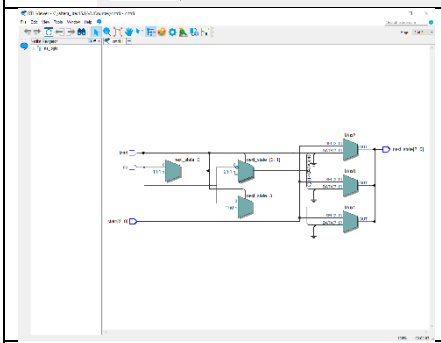


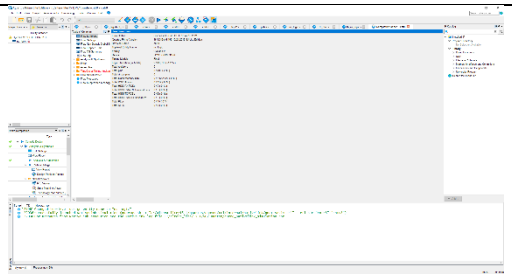


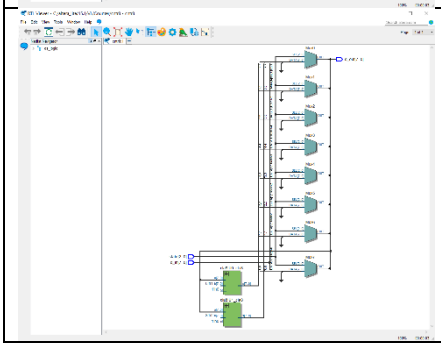
counter8

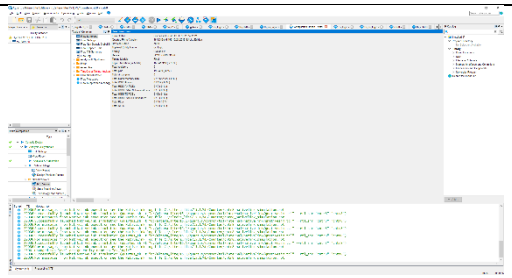


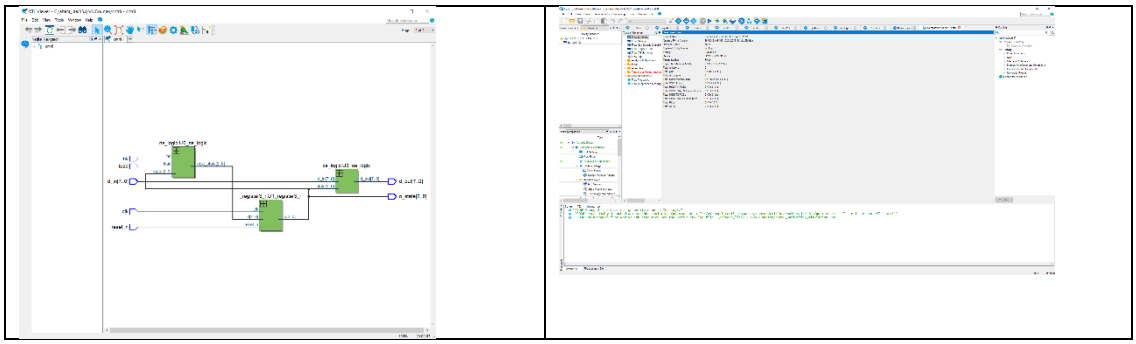












해당 모듈의 설계를 확인하면 각각의 모듈이 정상적으로 설계되어 연결되었음을 알 수 있다.

C. FPGA board targeting 결과

5. 고찰 및 결론

A. 고찰

이번 주차의 과제를 진행하면서 counter를 검증하는 과정에서 pin 개수가 모자라는 문제가 발생하였다. 해당 문제는 조교님에게 메일로 장치를 바꾸라는 조언을 통해 이를 해결할 수 있었다. 또한 각각의 모듈이 양이 많아 시간적으로 촉박함이 있었다.

B. 결론

loadable counter의 경우 초기값을 설정할 수 있다는 장점으로 인해 지정된 시간부터 시간을 측정하는 등에 이용할 수 있다. ring counter의 경우 초기값은 0부터 시작하지만 register만을 사용하여 간편하게 구성할 수 있으므로 고정된 시간을 측정하는 등에 활용하기 좋을 것이다. barrel shifter는 한번의 연산을 통해 다수의 비트를 이동시킬 수 있는 하드웨어 장치를 의미한다. n비트의 register에서 n개의 비트를 이동해야할 경우 n-to-1 MUX가 n개 필요한 것을 알 수 있다.

6. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021

barrel shifter/여러가지 case문과 barrel shifter : 네이버 블로그 (naver.com)