컴퓨터 공학 기초 실험2 보고서

실험제목: Register File

실험일자: 2021년 10월 18일 (월)

제출일자: 2021년 10월 31일 (일)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적

A. 제목

Register File

B. 목적

해당 수업을 통해 Register File이 어떻게 구성되어 있는지 알 수 있다. Register File에서 Write과 Read의 메커니즘을 이해하고 이를 구현할 수 있다. Stack과 Queue의 차이를 이해할 수 있다.

2. 원리(배경지식)

i. Stack

Stack이란 선입후출, 즉 LIFO(Last in First out)을 만족하는 데이터 구조를 의미한다. 삽입을 할 시 데이터를 stack의 최상위(TOP)에 저장한다. 그리고 데이터를 꺼낼 때는 TOP 위치에 있는 데이터를 호출하여 꺼낸다. 이를 통해 위와 같은 데이터 출력을 만족시킬 수 있다.

해당 자료 구조는 우리가 쉽게 찾아볼 수 있다. 코딩을 할 때 함수는 Stack의 방식으로 호출된다. 그리고 반환할 때에는 가장 나중에 호출한 함수 먼저 반환을 하는 구조를 만족시킨다.

우리가 크롬을 들어갔을 때 이전에 방문한 웹 페이지 또한 이러한 구조를 만족한다. 가장 최근에 접속한 웹 페이지를 우선적으로 반복하는 것이 stack 구조이기 때문이다.

ii. Queue

Queue란 선입선출, 즉 FIFO(First in First out)을 만족하는 데이터 구조를 의미한다. 삽입을 할 시 Queue의 가장 끝(rear)에 데이터가 저장된다. 그리고 데이터를 읽을 때에는 가장 처음(Front)에 있는 데이터를 POP함으로써 선입선출 구조를 만족시킨다.

해당 구조를 만들 수 있는 방법으로는 일반적으로 배열(Array)을 사용하거나 원형 Linked List를 구현하는 방법이 있다. 배열의 경우에는 가장 앞쪽에 데이터를 읽고 가장 뒤쪽에 있는 데이터를 넣음으로써 이를 만족시킬 수 있다. 하지만 해당 방식은 데이터를 POP 할수록 앞쪽의 빈공간이 많아지기 때문에 메모리 사용 측면에서 비효율적이다.

해당 방식을 개선하기 위해 원형으로 데이터를 연결하여 계속 순환하도록 함으로써 이를 해결한다.

iii. Register File

RF는 데이터가 저장되는 Register들과 읽기를 수행하는 Read Operation, 쓰기를 수행하는 Write Operation 부분으로 구성되어있다.

Write Operation에서는 wAddr를 통해 쓸 위치의 주소와 저장할 데이터, 그리고 쓰기 활성화 신호인 EN을 입력받는다. Decoder를 통해 해당 위치에 접근하여 데이터를 EN이 활성화 되어있을 경우 쓰는 방식으로 이를 구현한다.

Read Operation에서는 rAddr를 통해 읽을 위치의 주소를 가져온다. 그리고 MUX를 통해 읽을 데이터 위치에서 데이터를 꺼내어서 이를 수행한다.

3. 설계 세부사항

i. register32_r_en

해당 파일에서는 32bits의 reset_n과 enable신호를 받는 레지스터를 구현한다. Always 문을 활용하여 CLK이 falling edge를 갖거나 reset_n이 rising edge를 가질 때 동작하도록 구현한다. 그리고 if문을 활용하여 en의 신호가 없을 시 기존의 데이터를 유지하도록 하고, en 신호가 있을 시 쓰기 작업을 수행할 수 있도록 한다. 레지스터는 기존에 만든 _dff_r_en 모듈을 이용한다.

ii. register32_8

해당 모듈에서는 32bit의 register들을 8개를 만든다. 해당 모듈은 위에 resgister32_r_en 모듈을 8개 사용하여 구현하였다.

iii. 3_to_8_decoder

해당 모듈은 3bit의 신호를 8bit의 신호로 디코딩하는 모듈이다. 해당 모듈에서는 case문을 활용하여 각 값에따라 알맞는 디코딩 값을 출력으로 대입시켜준다.

iv. write_operation

해당 모듈에서는 Write operation을 구현한다. 해당 모듈에서는 3_to_8_decoder를 사용하여 저장할 위치(wAddr)를 Decoder를 통해 선별하여 데이터를 입력시

켜준다. 그리고 we(Write Enable) 신호와 and gate로 연결하여 쓰기 신호가 활성화될 경우에만 쓰기를 실행하도록 만들어준다.

v. _8_to_1_MUX

해당 모듈은 8개의 신호 중 하나의 신호를 rAddr를 통해 선별(MUX) 사용하여 읽어주는 방식을 사용한다. 해당 모듈은 case문을 활용하여 제작되었다.

vi. read_operation

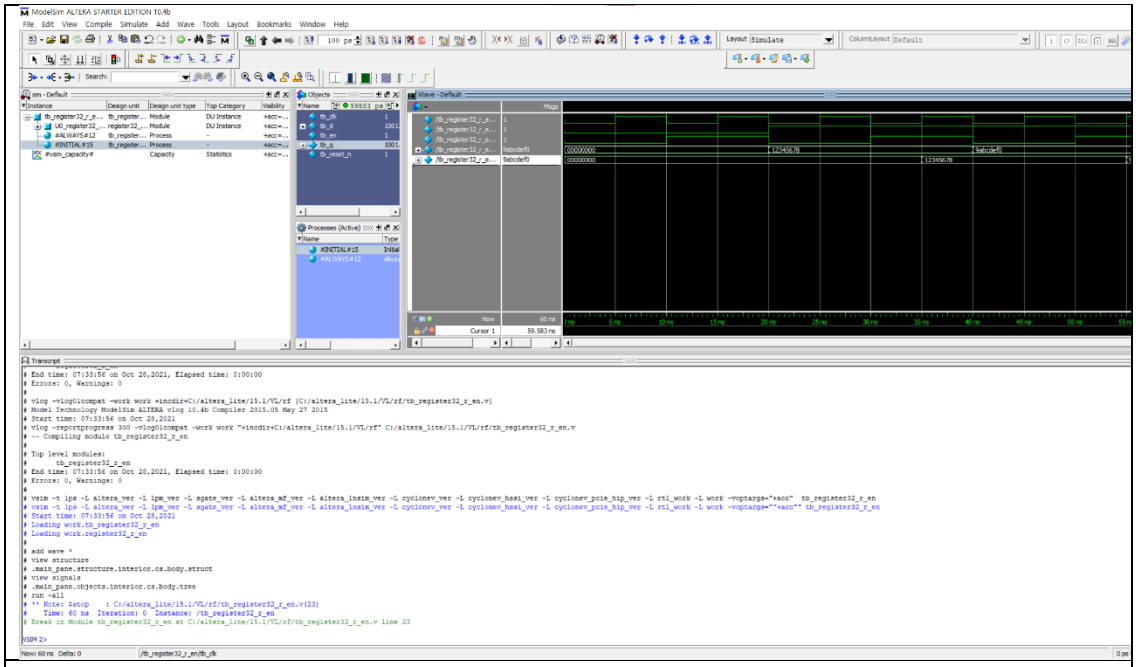
해당 모듈은 읽기 작업을 수행한다. 해당 모듈에서 8개의 32bit의 값을 선별하여 하나의 데이터 묶음만 읽도록 함으로써 이를 구현하였다. 여기에는 _8_to_1_MUX를 활용하였다.

vii. Register_file

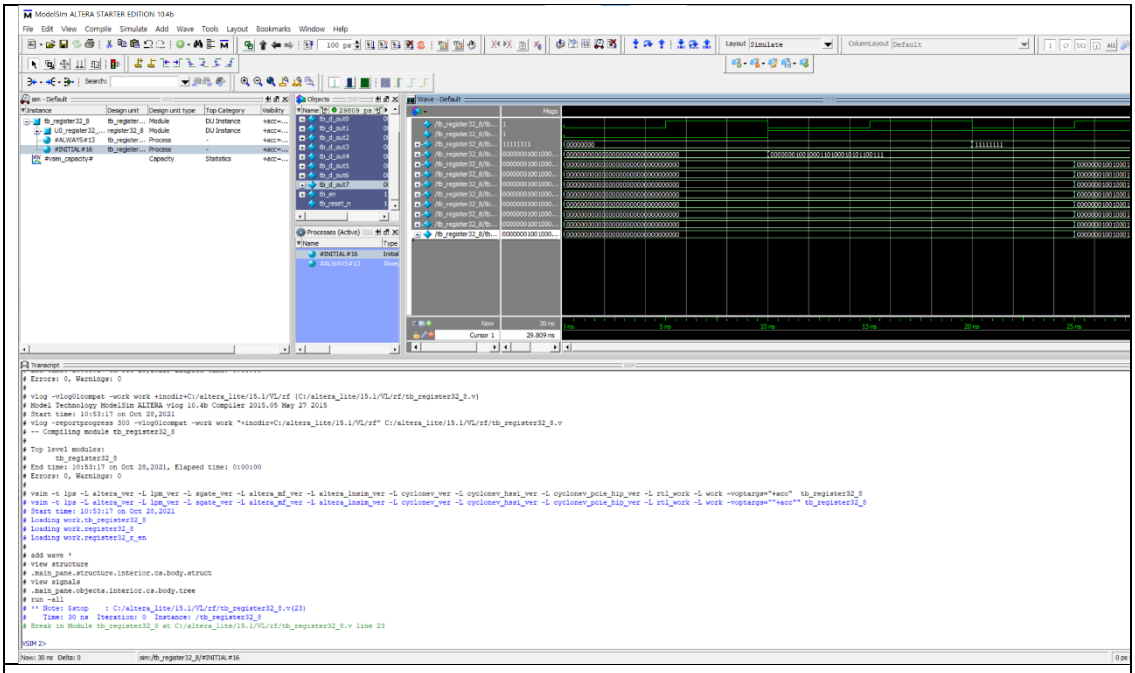
해당 모듈은 RF를 실질적으로 구현한 회로다. 해당 회로에서는 위에서 만든 write_operation, read_operation, register32_8을 연결하여 이를 구현하였다.

4. 설계 검증 및 실험 결과

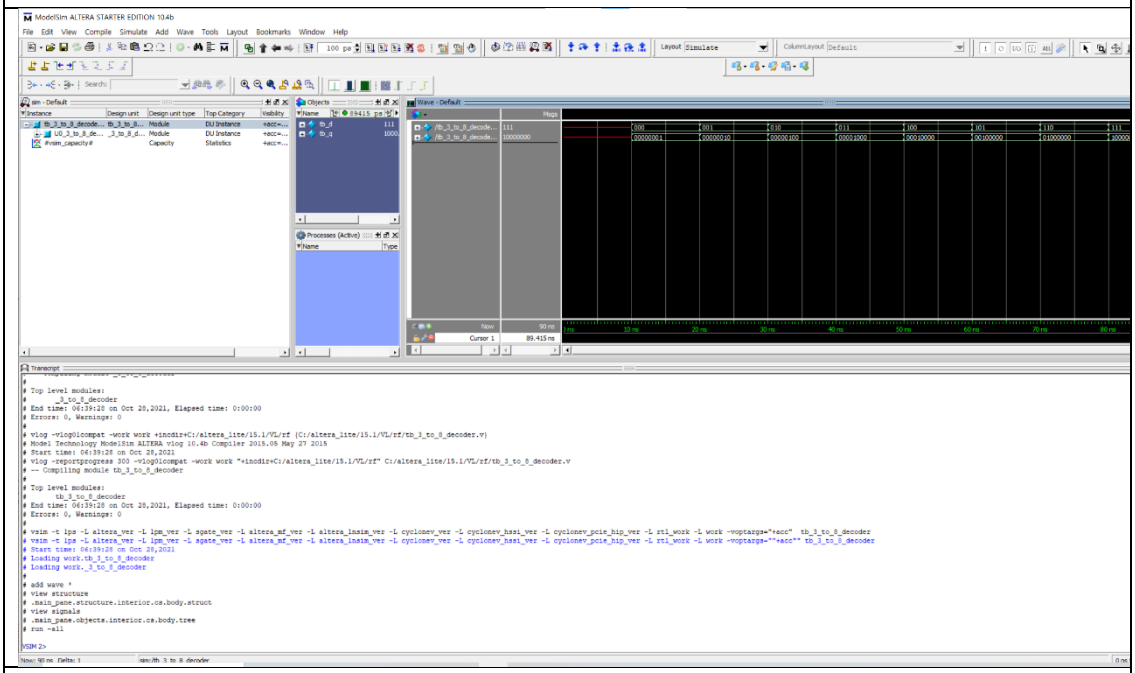
A. 시뮬레이션 결과



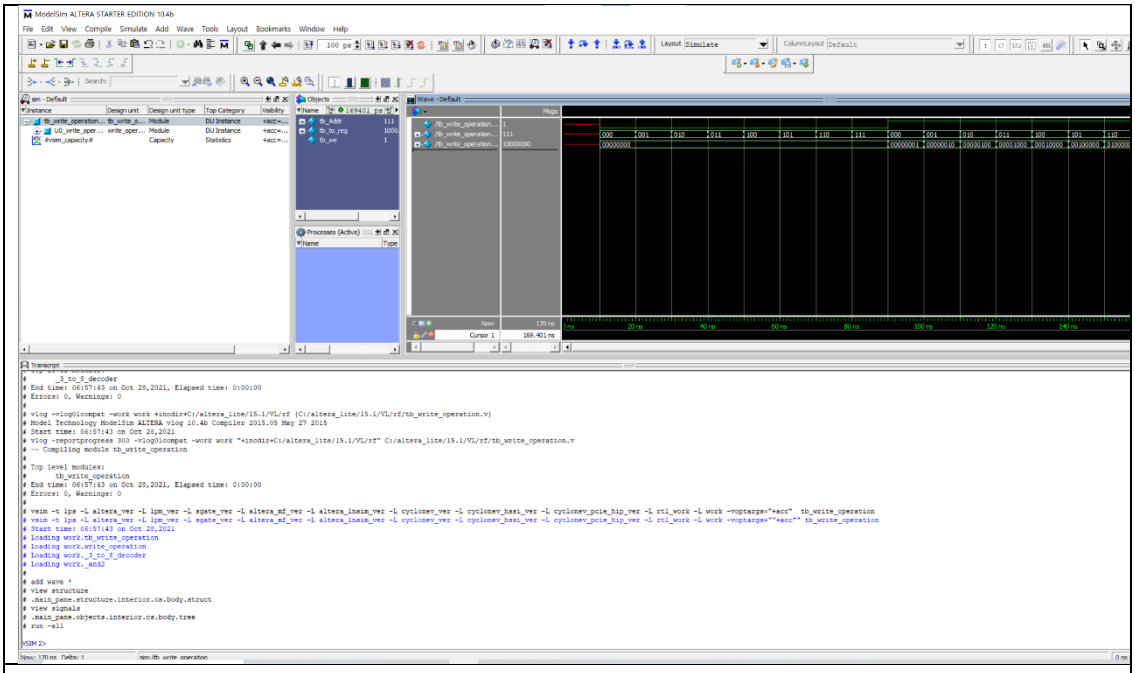
해당 testbench는 _register32_r_en을 검증한다. 해당 결과를 보면 정상적으로 32비트의 값이 저장되는 것을 알 수 있다.



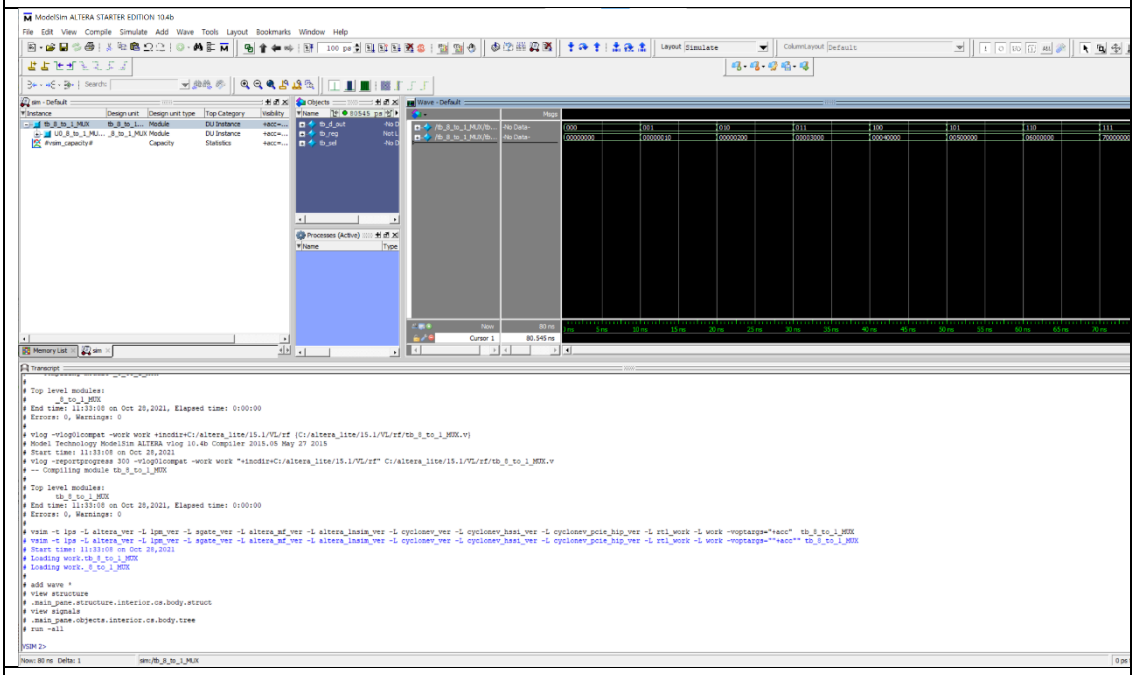
해당 testbench는 register File에 대해 검증한다. 해당 결과를 보면 8개의 값이 reset과 en의 값에 따라 데이터가 저장되는 모습을 확인할 수 있다.



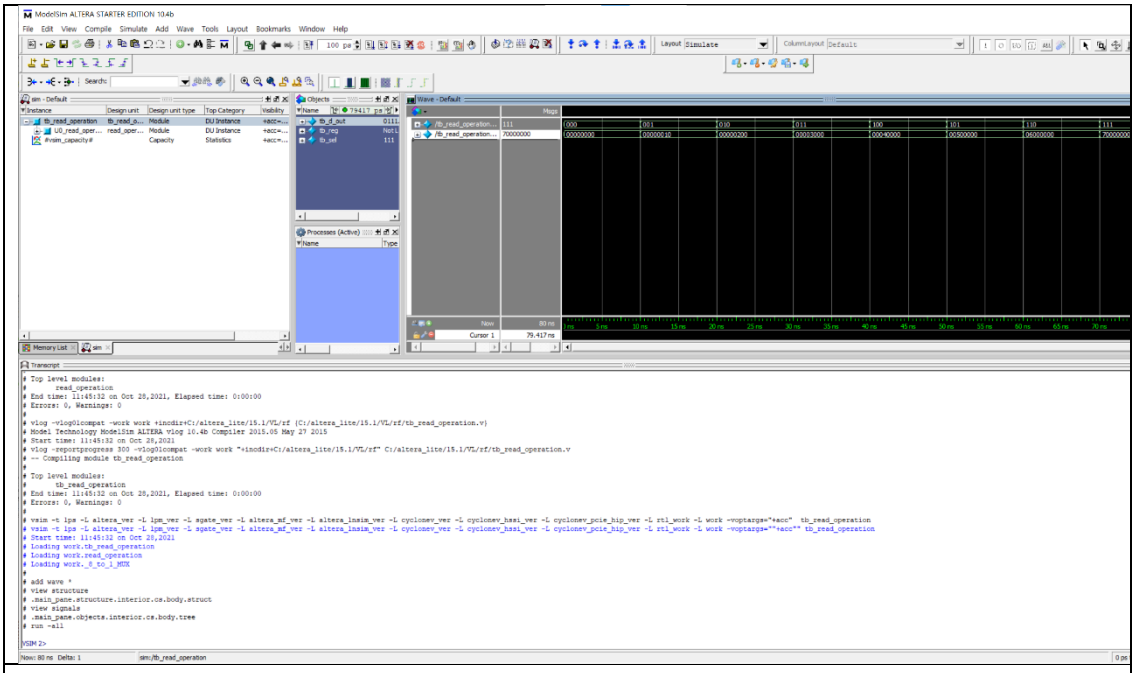
해당 testbench는 _3_.to_8_decoder에 대해 검증한다. 해당 결과를 보면 각 위치에 따라 위치에 해당하는 비트만 1로 활성화되는 것을 알 수 있다.



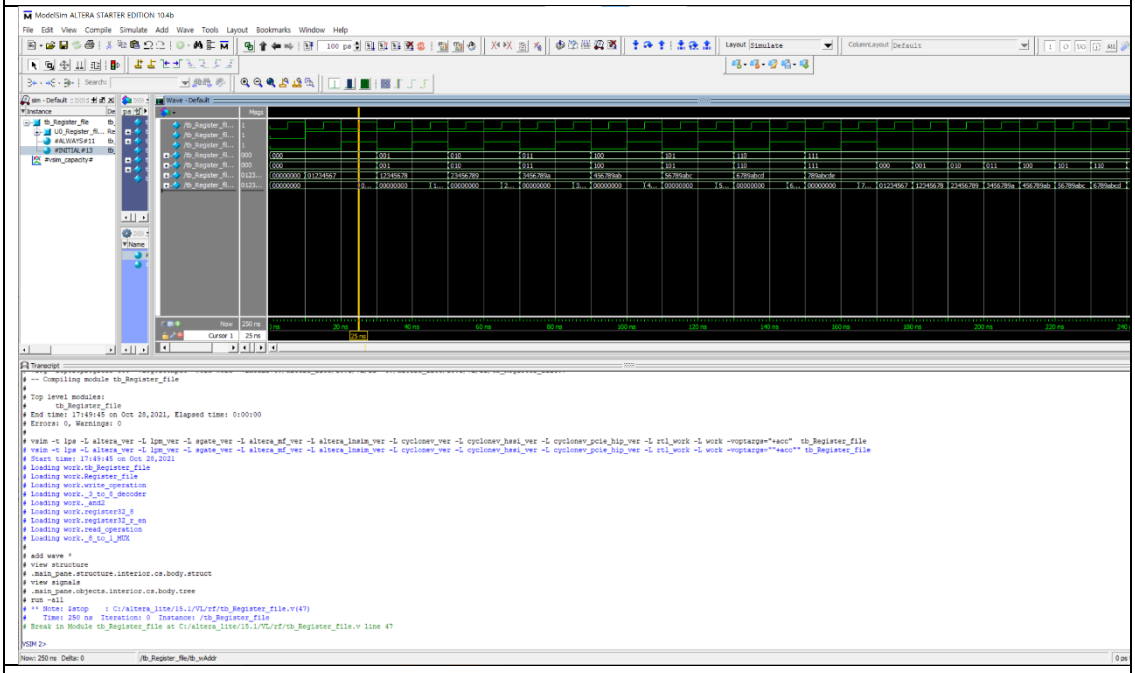
해당 testbench에서는 write_operation을 검증한다. 위의 결과를 we가 활성화 되어있을 때만 정해진 위치에 데이터가 전달되는 것을 확인할 수 있다.



해당 testbench에서는 _8_to_1_MUX를 검증한다. 해당 결과를 보면 읽는 위치에 따라 각 위치의 32비트 값이 정상적으로 읽히는 것을 확인할 수 있다.



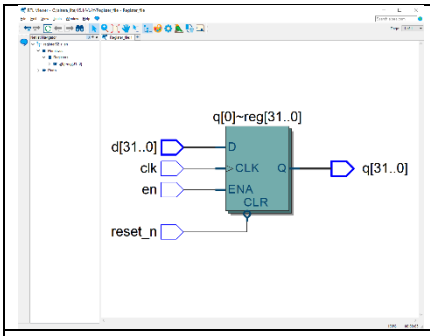
해당 testbench에서는 read_operation을 검증한다. 해당 결과를 보면 정상적으로 해당 위치의 값을 불러오는 것을 확인할 수 있다.

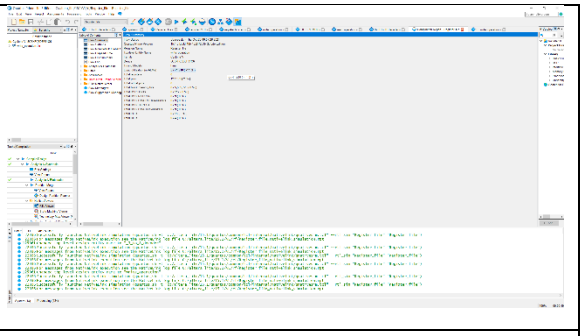


해당 testbench에서는 Register_file을 검증한다. 해당 결과에서 볼 수 있다시피 정상적으로 각각의 위치에 데이터를 저장하고, 각각의 위치에서 데이터를 읽어오는 것을 확인할 수 있다.

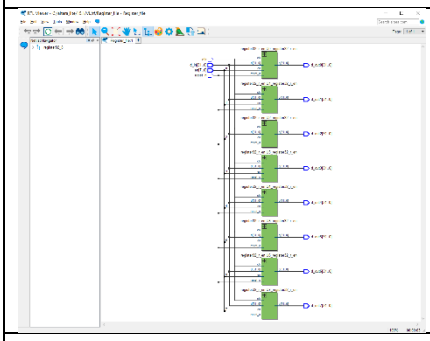
B. 합성(synthesis) 결과

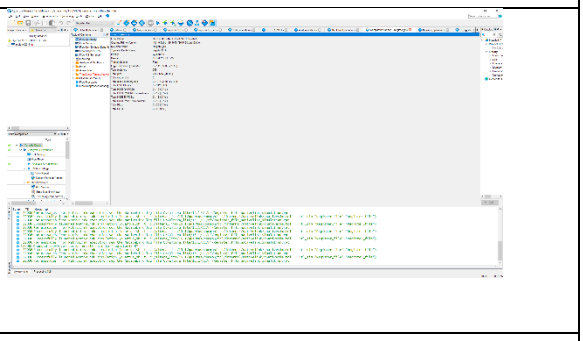
register32_r_en



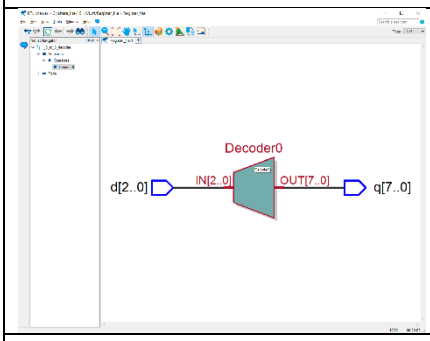


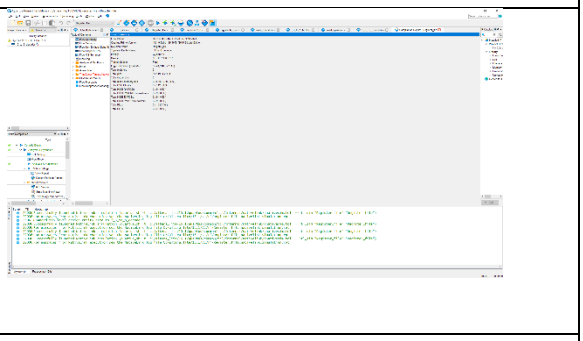
register32_8



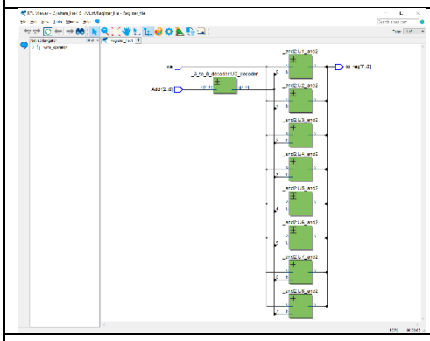


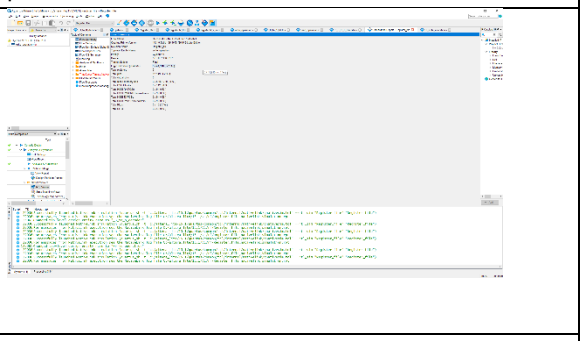
_3_to_8_decoder



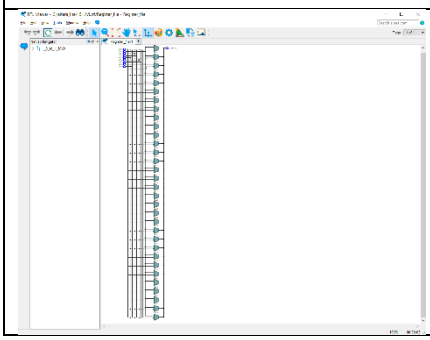


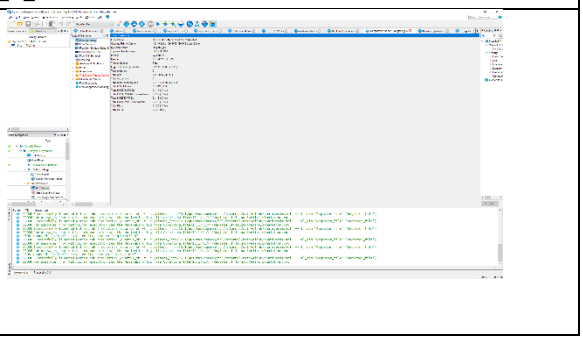
write operation



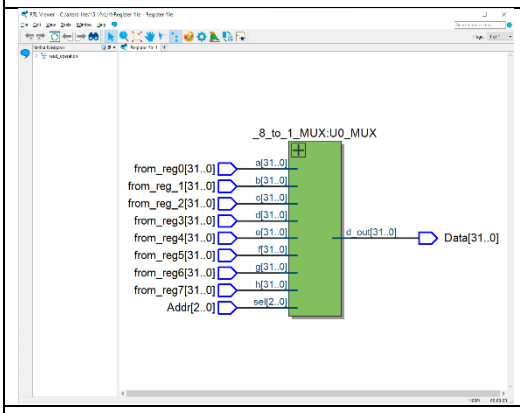


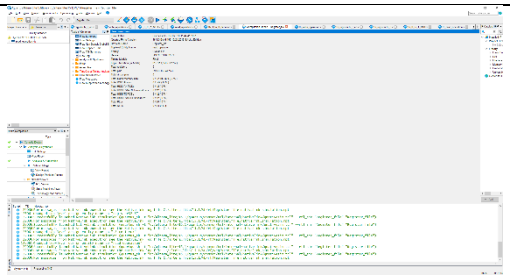
_8_to_1_MUX



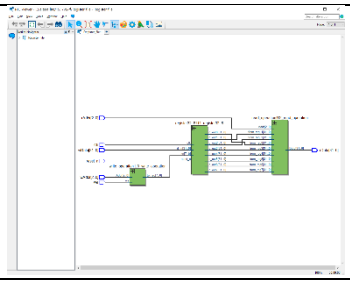


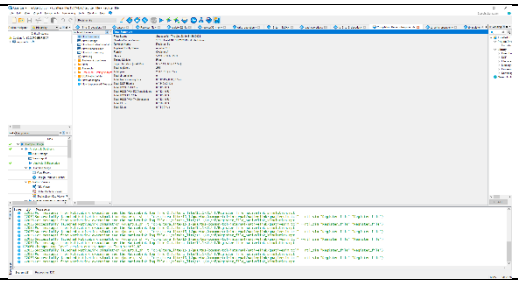
read_operation





Register_file





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

C. FPGA board targeting 결과

5. 고찰 및 결론

A. 고찰

해당 과제를 진행하면서 여러가지 에로사항이 있었다. 특히 register32_8을 검증할 때 핀의 개수가 부족하여 컴파일이 에러가 나온 상황이 발생하였다. 이를 해결하기 위해 조교님에게 메일을 보내어 작업하는 환경을 바꾸어 이를 해결할 수 있었다. 이외에는 어렵지 않게 과제를 수행할 수 있었다.

B. 결론

해당 과제를 통해 컴퓨터에서 read의 과정은 MUX를 통해 이루어지는 것을 알 수 있다. write 과정은 Decoder를 통해 구동되는 것을 알 수 있다. 기존의 데이터를 유지하기 위해 MUX를 활용하여 데이터를 지속적으로 넣어줌으로서 이를 구현할 수 있다.

6. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021