컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2021년 10월 25일 (월)

제출일자: 2021년 11월 06일 (토)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적

A. 제목

FIFO

B. 목적

해당 수업을 통해 FIFO에 대해 이해할 수 있다. queue에 대해 이해하여 이를 기반으로 이를 FIFO의 구조를 이해할 수 있다. FSM을 통해 해당 모듈을 설계할 수 있다.

2. 원리(배경지식)

i. FIFO

FIFO란 First In First out의 약자로 선입선출이라고도 불린다. 해당 방식은 메모리 공간에 먼저 입력된 데이터를 먼저 출력하는 방식을 의미한다. 해당 방식을 사용하는 방식은 대표적으로 Queue가 있다.

해당 방식으로 회로를 설계하기 위해서는 우리가 제작하는 방식에서는 register File을 이용한다. 읽을 위치와 쓰는 위치를 각각 head, tail로 주소를 정의한다. 그리고 저장된 데이터의 수를 data_count를 통해 기억한다. 현재 우리가 제작한 register file은 8개의 32-bit 데이터를 저장할 수 있다.

INIT 상태는 reset_n을 통해 초기화한 상태를 의미한다. NO_OP 상태는 쓰거나 읽는 동작을 수행하지 않는다. WRITE에서는 쓰는 동작을 READ에서는 읽는 동작을 수행한다. WR_ERROR와 RD_ERROR에서는 쓰거나 읽는 동작이 각각 기존 데이터가 다 차있거나 비어있어서 수행할 수 없는 상태를 의미한다.

만약 마지막으로 쓸 위치가 111(2)일 경우 다음 tail의 위치는 000(2)으로 설정한다. 만약 마지막으로 읽을 위치가 000(2)일 경우 다음 head의 위치는 111(2)으로 설정한다.

만약 data_count를 통해 저장된 데이터의 개수가 0임을 확인한 경우 empty를 1로 출력한다. 또는 데이터의 개수가 8인 경우 full을 1로 출력함으로써 데이터가 전부 차 있음을 확인한다.

쓰는 동작이 정상적으로 실행된 경우에는 wr_ack를 1로 출력하여 확인시켜주고, 읽는 동작이 정상적으로 실행된 경우에는 rd_ack를 1로 출력하여 이를 확인시켜

준다.

쓰는 동작에서 오류가 발생한 경우(full 상태) wr_err를 1로 출력하여 쓰는 동작에서 문제가 있음을 알려주고, 읽는 동작이 오류가 있는 경우(empty 상태) rd_err를 1로 출력하여 피드백 해준다.

3. 설계 세부사항

i. fifo_ns

해당 모듈은 다음 state를 계산하는 logic이다. 해당 모듈에서는 case문을 활용하여 state의 각 상태에 따라 if문을 통해 wr_en, rd_en, data_count를 토대로 next_state를 결정한다.

ii. fifo_cal_addr

해당 모듈은 현재 상태에 따라 기존의 data_count, head, tail의 주소를 결정하고, 읽기와 쓰기모드를 결정한다. cla4를 이용하여 head++, tail, data_count++, data_count—의 값을 계산해둔다. 그 후 case 문을 활용하여 state의 각 상태에 따라 next_head, next_tail, next_data_count, we, re의 값을 결정한다. 기본적으로 head와 tail, data_count는 이전 상태의 값을 유지한다. 하지만 쓰기나 읽기 상태의 경우 각 값을 위에서 만든 모듈의 값으로 바꾸어 값을 증감시켜준다. 그리고 읽기 상태에서는 re를 1로, 쓰기 상태에서는 we를 1로 출력하도록 한다.

iii. Register_file

해당 모듈은 저번 주차에서 만든 것이며, 읽는 위치와 쓰는 위치에서 읽기, 쓰기 동작을 수행한다. 단 쓰는 동작은 we가 1일 경우에만 수행한다.

iv. fifo_out

해당 모듈은 현재 상태를 출력해주는 logic이다. 해당 모듈에서는 case문을 활용하여 state의 값에 따라 주어진 값을 설정해준다. 기본적으로 모든 동작은 0으로 수행한다. 단 아래의 경우에는 각 값을 1로 바꾸어준다. 만약 NO_OP일 경우 if 문을 통해 0000이나 1000일 경우에는 empty를 1, full을 1로 각각 설정해준다. 만약 쓰기가 1일 경우 wr_ack를 1로 설정해주고 이 때 data_count가 1000일 경우 full을 1로 출력해준다. 만약 읽기가 1일 경우 rd_ack를 1로 설정해주고 이 때 data_count가 0000일 경우 empty을 1로 출력해준다. 각각의 에러 상태에서는

해당하는 에러를 1로 출력해준다.

v. register32_r_en

해당 모듈은 Register_file을 만드는데 쓰이는 모듈이기도 하며, fifo 안에서 한 번 더 추가로 사용되는 모듈이다.

vi. register3_r & register4_r

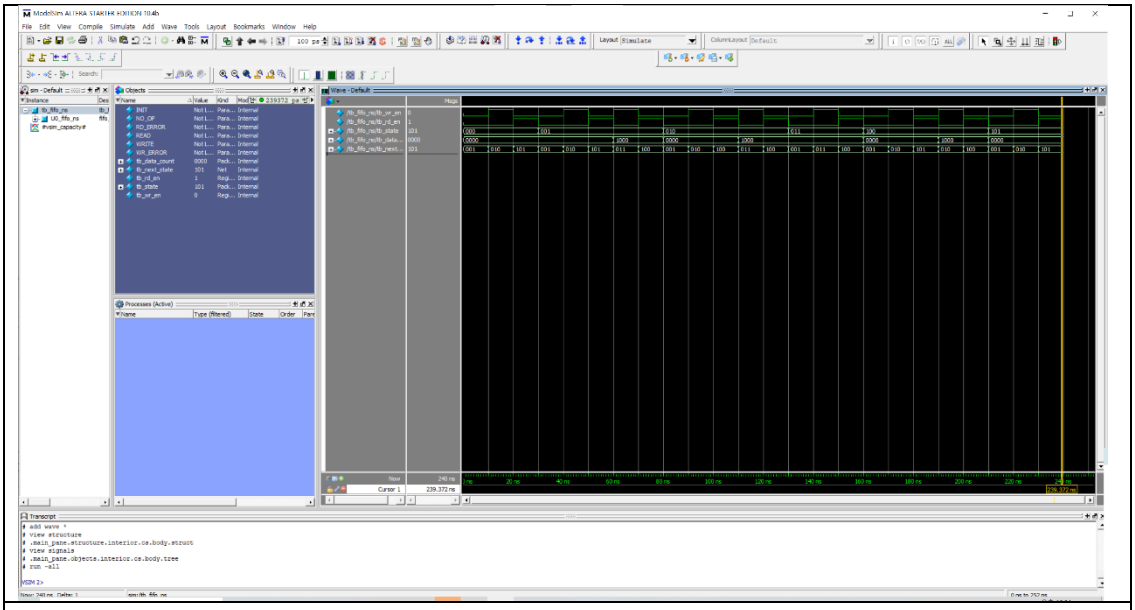
해당 모듈은 기존의 register32_r_en을 수정하여 제작되었으며 state, tail, head, data_count를 저장하기 위해 제작되었다.

vii. fifo

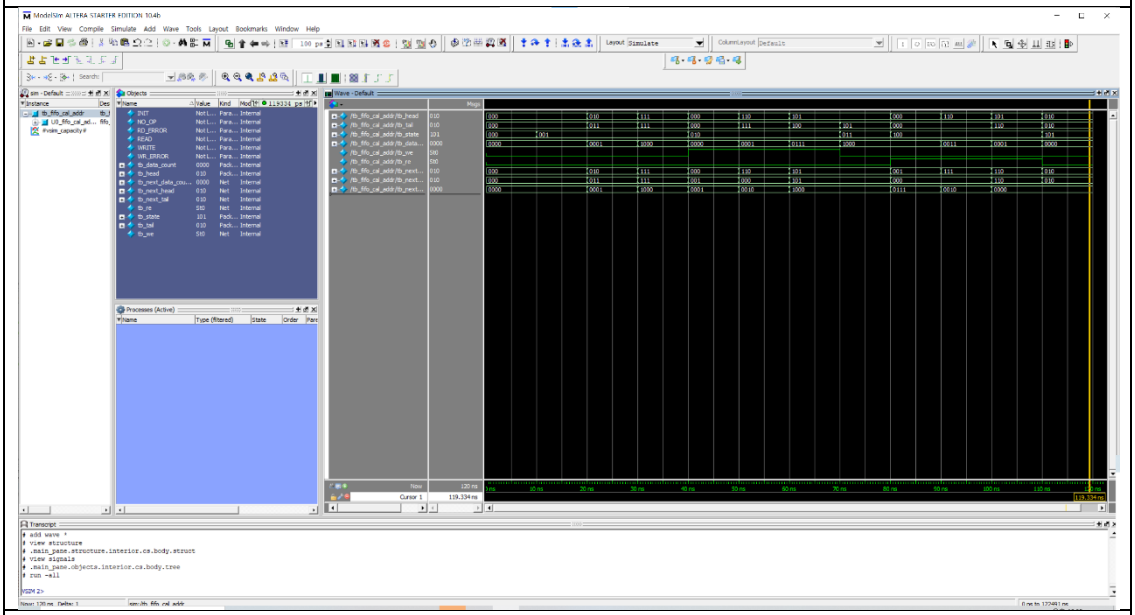
해당 모듈은 FIFO를 최종적으로 구현한다. fifo_ns를 통해 현재 state와 data_count를 토대로 다음 state를 결정한다. 그리고 fifo_cal_addr에서는 fifo를 통해 변경된 state와 현재 head, tail, data_count의 값을 토대로 다음 주소들과 count를 결정시키고, we와 re의 값을 결정시켜준다. U2에서는 각각 state, head, tail, data_count를 register에 잠시 저장시켜준다. fifo_out에서는 U2의 register의 값들을 토대로 출력으로 보여지는 handshake signal을 출력한다. Register_file에서는 tail과 head의 주소에서 각각 읽거나 쓰는 동작을 수행한다. mx2_32bits는 이전에 구현했던 모듈이며 해당 모듈에서는 re가 1일 경우에만 register_file의 데이터를 출력하고 이외에는 0x0000_0000을 출력한다. register32_r_en에서는 mx2_32bits에서 나온 데이터를 레지스터를 통해 잠시 저장시켜준다.

4. 설계 검증 및 실험 결과

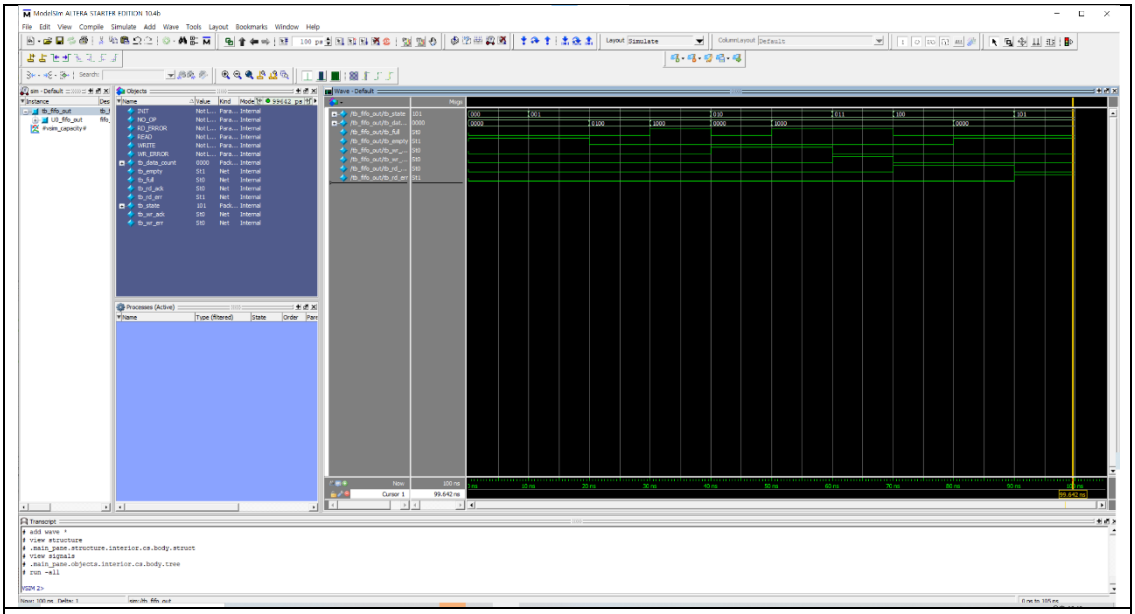
A. 시뮬레이션 결과



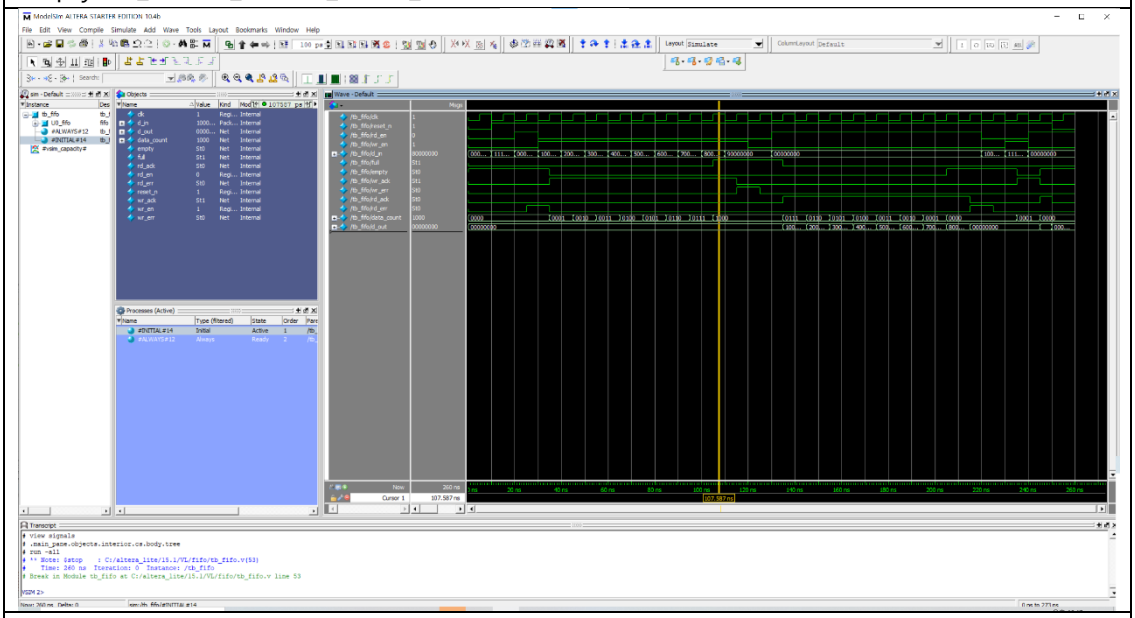
해당 testbench는 fifo_ns를 검증한다. 위에서 결과를 보면 현재 상태와 wr_en, rd_en, data_count의 값에 따라 다음 상태가 적절히 바뀌는 것을 확인할 수 있다.



해당 testbench는 fifo_cal_addr를 검증한다. 위에 결과를 보현 현재 상태와 data_count에 따라 we, re의 값과 next_head, next_tail, next_count가 적절하게 바뀐 것을 확인할 수 있다.



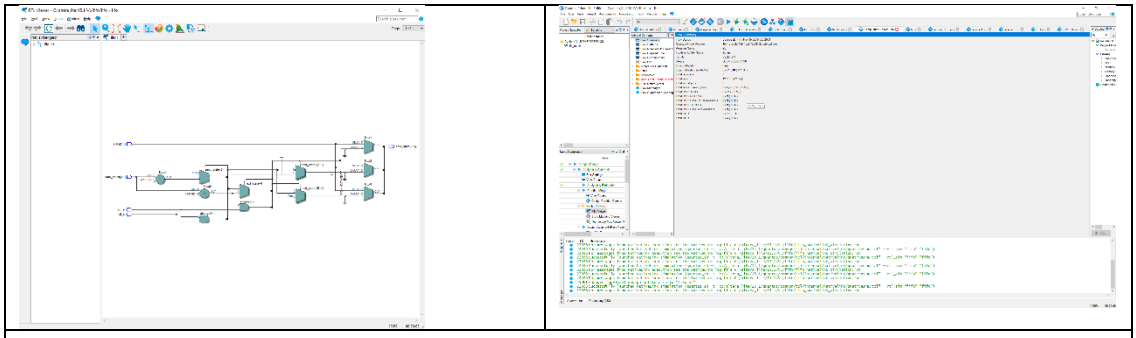
해당 testbench는 fifo_out을 검증한다. 해당 결과를 보면 현재 상태와 count에 따라 full, empty, wr_ack, rd_ack, wr_err, rd_err가 적절하게 바뀌는 것을 확인할 수 있다.



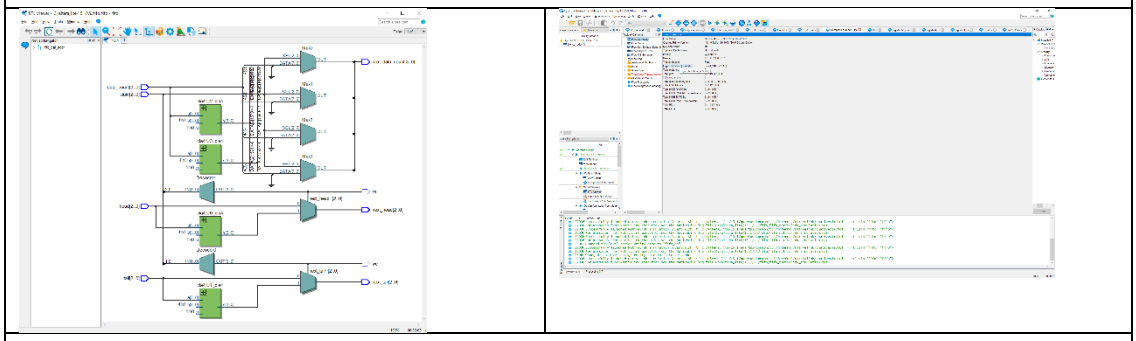
해당 testbench에서는 fifo를 검증한다. 처음초기화 후 데이터가 없는 상태에서는 읽기가 불가능하다. 쓰기를 8번 수행한 후 한 번 더 수행하면 쓰기가 불가능하다. 그 후 읽기를 8번 수행하고 한 번 더 수행하면 읽기 불가능이 표시된다. 해당 수행 이후에도 쓰기와 읽기가 가능하고 reset이 0이 되면 다시 해당 회로가 초기화된다.

B. 합성(synthesis) 결과

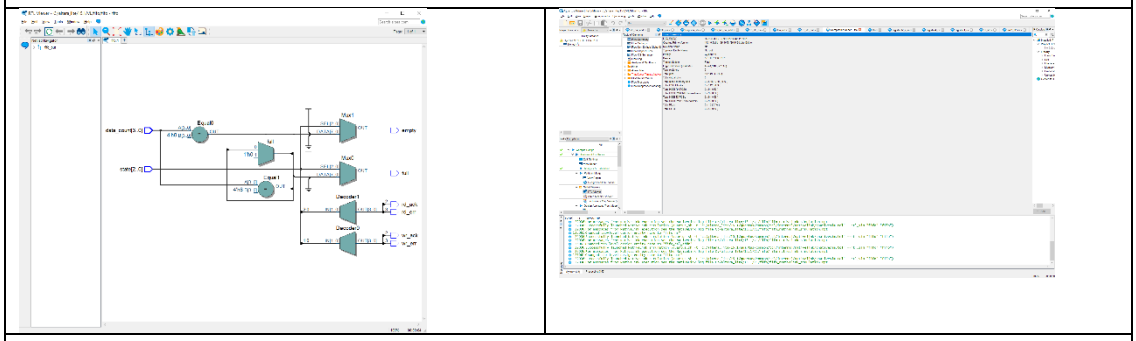
fifo_ns



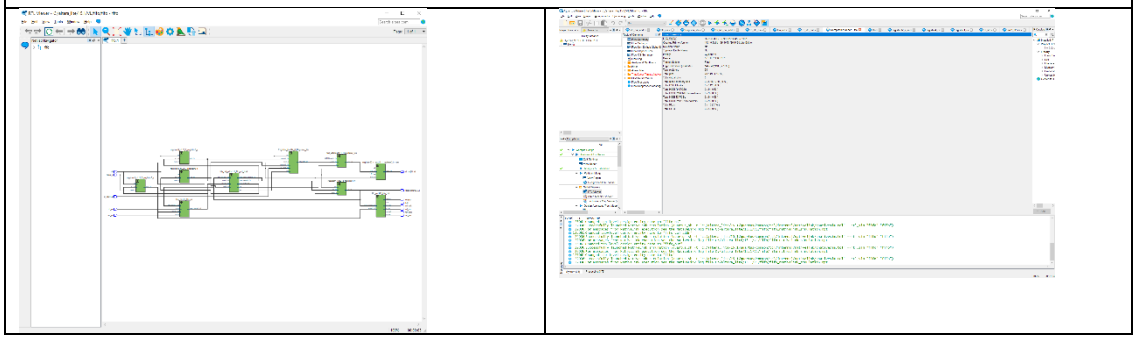
fifo_cal_addr



fifo_out



fifo



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 큰 어려움은 없었다. next_state를 fifo_cal_addr에 연결해야 하는 것을 cur_state을 연결하면서 오류가 있었으나 금방 찾아서 고칠 수 있었다. 이번 과제에서는 특히나 case문을 많이 활용한 것이 인상 깊었다.

B. 결론

해당 과제를 통해 FIFO를 이해할 수 있다. 제작한 fifo는 circle 구조를 통해 계속 메모리 낭비가 없다. 여러 복잡한 상태를 module로 세분화하여 제작할 경우 좀 더 쉽게 회로를 구성할 수 있다.

6. 참고문헌

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