컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2022년 11월 1일 (화)

제출일자: 2022년 11월 2일 (수)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

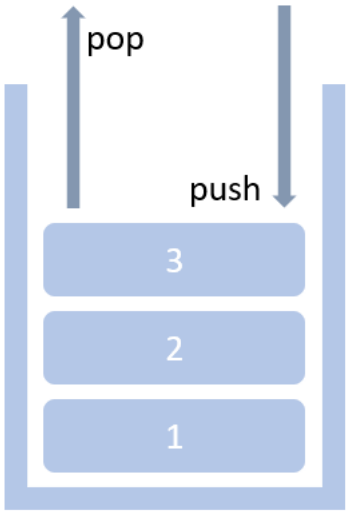
FIFO

B. 목적

이전 실습에서 설계한 Register file 을 사용하여 FSM 설계 순서에 따라 FIFO 를 설계한다. FIFO 에서 자료가 write, read 되는 것을 이해한다. 또한 next state logic, calculate address logic, ouput logic 을 활용하여 FIFO 를 설계한다.

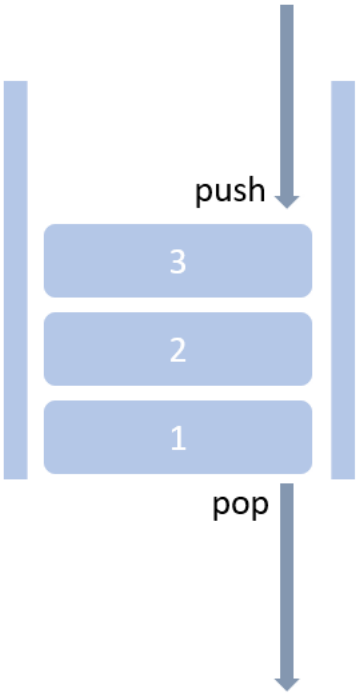
2. 원리(배경지식)

- Stack



스택(stack)은 한 쪽 끝에서만 자료를 넣거나 뺄 수 있다. 선형 구조, 즉 LIFO로 되어 있다. LIFO는 Last In First Out으로 먼저 들어간 데이터가 가장 아래에 깔리며, 데이터를 꺼낼 때는 제일 위에서부터 꺼내는 구조이다. 자료를 넣는 것을 push, 자료를 꺼내는 것을 pop이라고 하며 가장 최근에 push, 즉 가장 최근에 넣은 것이 가장 먼저 나오게 된다. d 여기서 put은 insert를, push는 delete를 의미하며 stack에서 insert와 delete는 같은 곳에서 발생한다. 이 과정이 위의 그림과 같다.

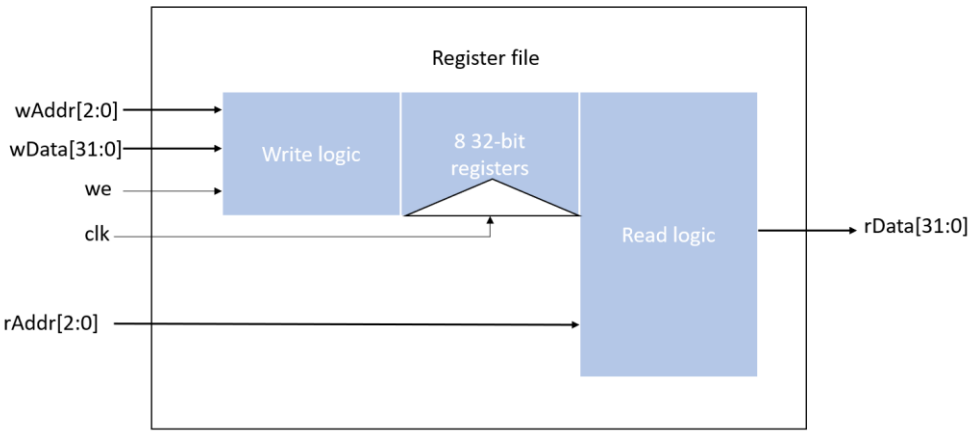
- Queue



큐(Queue)는 FIFO의 구조이다. FIFO는 First In First Out으로 먼저 집어 넣은 데이터가 먼저 나오는 구조를 말한다. 나중에 집어 넣은 데이터가 먼저 나오는 스택(Stack)과 반대되는 개념이다.

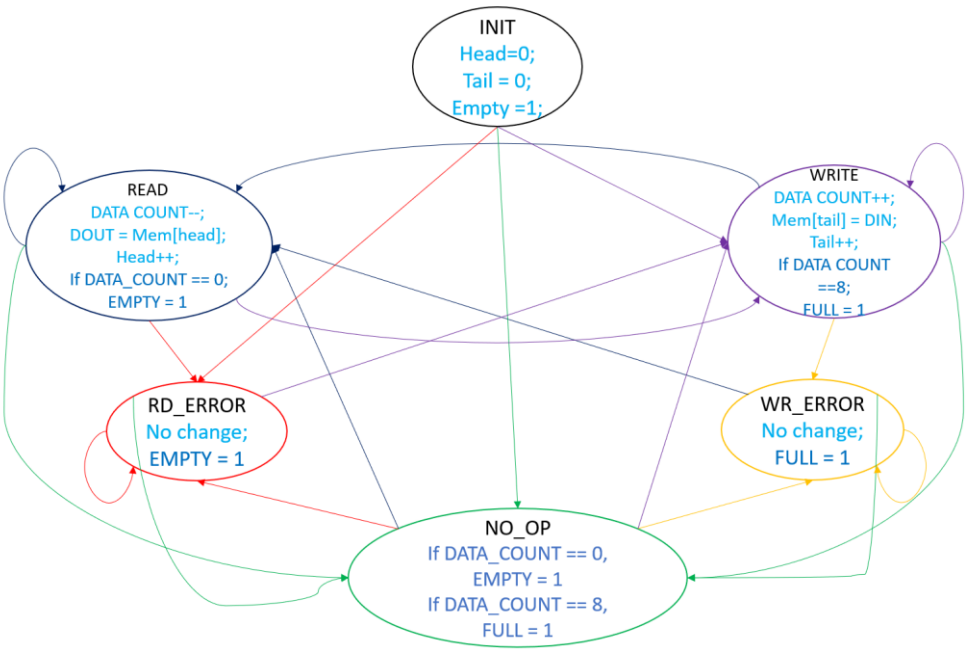
큐에서 insert하는 것을 enqueue, delete하는 것을 dequeue라고 하는데 enqueue는 뒤에서, dequeue는 앞에서부터 진행한다. 이때 데이터를 insert 할 수 있는 위치를 front(head), 데이터를 delete 할 수 있는 위치를 rear(tail)이라고 한다. 또한 큐가 꽉 차서 더 이상 자료를 넣을 수 없는 경우를 overflow, 큐가 비어 있어서 자료를 꺼낼 수 없는 경우를 underflow라고 한다.

- Register file



Register file은 사용자로부터 받은 입력을 저장하거나 알맞은 register의 값을 출력한다. 이때 위의 그림과 같이 register에서 read와 write를 선택적으로 할 수 있다.

3. 설계 세부사항



FIFO FSM의 설계도로, 보라색 화살표는 WR_EN & ~FULL, 노란색 화살표는 WR_EN & FULL / WR_ERR, 초록색 화살표는 nop, 빨간색 화살표는 RD_EN & EMPTY / RD_ERR 마지막으로 파랑색 화살표는 RD_EN & ~EMPTY을 의미한다.

State operation은 다음과 같다.

State	Operations
INIT	Head = tail = data_count = 0
WRITE	Mem[tail] = din, tail ++, data_count++; Data_count == 8, FULL = 1
READ	Dout = mem[head], head ++, data_count --, data_count == 0, EMPTY = 0
WR_ERROR	No change, FULL = 1
RD_ERROR	No change, Empty = 1
NO_OP	No change

Data count를 기준으로 full과 empty를 판단한다.

Data_count	FULL	EMPTY
0	0	1

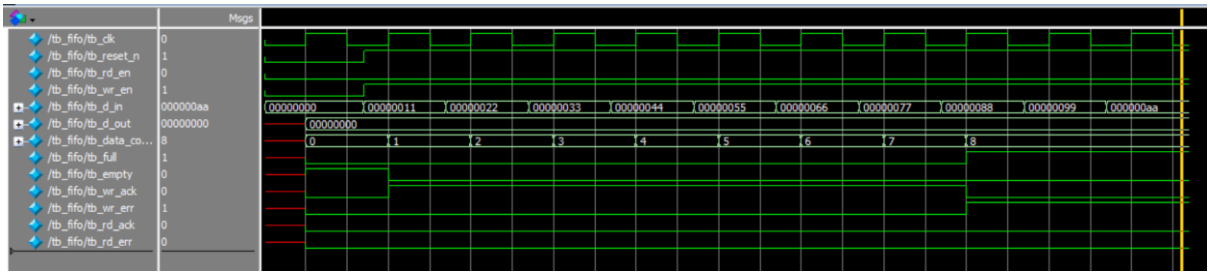
8	1	0
1~7	0	0

Register file을 구현할 때 사용했던, register32_r_en, write_operation, read_operation, Register_file을 불러오고, 위의 표와 fifo fsm을 참고하여 next state logic, calculate address logic, output logic, register file를 설계하였다. FIFO를 설계하면서 연결을 위한 tail, head, data_count, state, dout은 2 input mux, 3 input d filp-flip, 4 input d flip-flop, 32 input d flip-flop을 만들어 추가해주었다.

4. 설계 검증 및 실험 결과

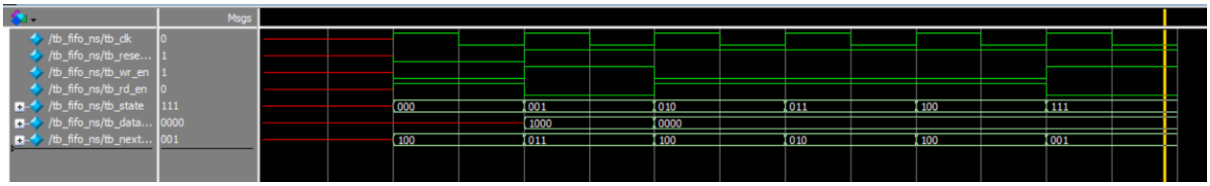
A. 시뮬레이션 결과

- fifo



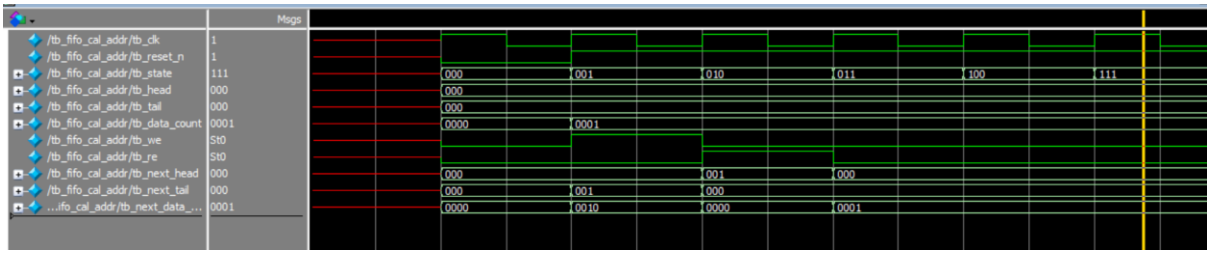
데이터가 0개일 땐 empty가 1이 되는 것을, 데이터가 1개에서 7개일 때는 empty와 full 모두 0인 것을, 데이터가 8개 일 때 full이 1이 되는 것을 확인할 수 있다.

- fifo_ns



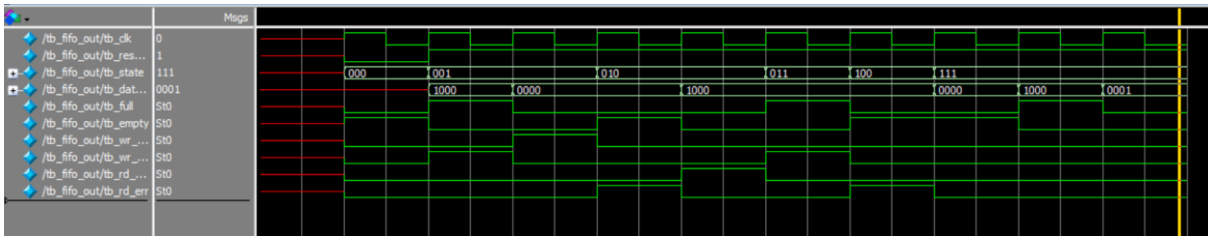
Sub module인 fifo_ns의 testbench이다. 값이 잘 저장되는 것을 확인할 수 있다.

- fifo_cal_addr



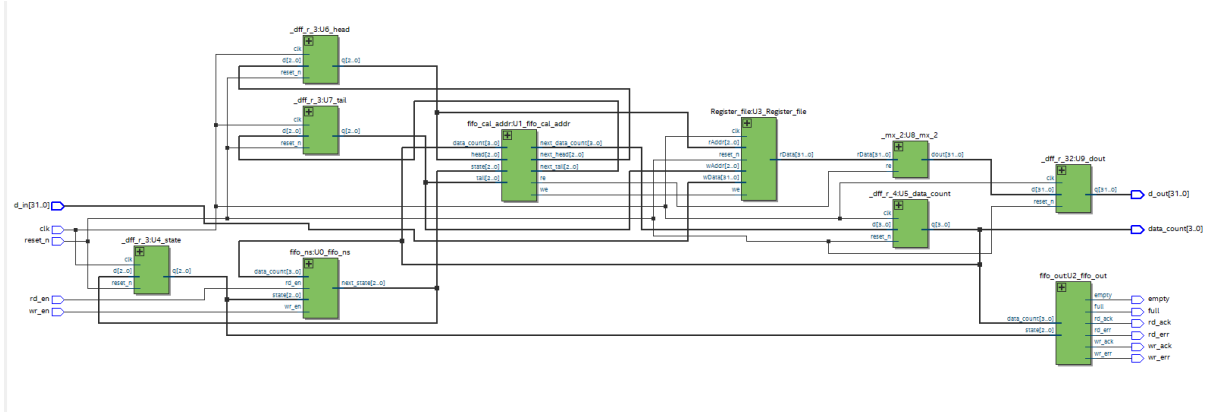
Sub module인 fifo_cal_addr의 testbench이다. 값이 잘 저장되는 것을 확인할 수 있다.

- fifo_out



Sub module인 fifo_out의 testbench이다. 값이 잘 저장되는 것을 확인할 수 있다.

B. 합성(synthesis) 결과



FIFO의 rtl viewer로, register file에서 구현했던 파일 이외에도 3 input d filp-flop, 4 input d flip-flop, 32 input d flip-flip, 2 input mux를 추가하여 Next State Logic, Calculate Address Logic, Output Logic 그리고 Register File이 서로 잘 연결되어 설계된 것을 확인할 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Nov 02 05:42:18 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	fifo
Top-level Entity Name	fifo
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	301
Total pins	78
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

FIFO의 flow summary이다. 총 301개의 register와 78개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

이번 실습은 주어진 block diagram과 fifo fsm, state operations만 참고해 구현해야 하다 보니 시간이 오래 걸렸다. 또한 block diagram을 state, data_count, head, tail, dout과 같이 따로 추가해주어야 하는 부분도 있어 파일이 많아 복잡했다. FIFO의 testbench에서 항상 U0_fifo(tb_clk, reset_n) 이런 식으로 작성하였는데 오류가 떠 data_count 부분이 출력되지 않았다.

'fifoU0_fifo(.clk(tb_clk),.reset_n(tb_reset_n),.rd_en(tb_rd_en),.wr_en(tb_wr_en),.d_in(tb_d_in),.d_out(tb_d_out),.data_count(tb_data_count),.full(tb_full),.empty(tb_empty),.wr_ack(tb_wr_ack),.wr_err(tb_wr_err),.rd_ack(tb_rd_ack),.rd_err(tb_rd_err));' 다음과 같이 작성하니 잘 출력되었는데 이유를 모르겠어 이것저것 수정해서 실행해보았지만 이유를 찾지 못해 아쉬웠다.

B. 결론

파일도 많고 회로 자체가 복잡해 다른 실습 보다 오류를 찾는데 가장 많은 시간이 걸렸던 것 같다. 또한 wire와 reg를 선언하는 것이 항상 헷갈렸는데 이번 실습을 통해 언제 무엇을 사용해야 하는지 알게 된 것 같다. 미리 ppt에 제공된 testbench를 통해 공부하고 실습을 진행하니 이해도 잘 됐고 무엇보다 코드를 짜는 면에 있어서는 수월했던 것 같다. 다만 아직 회로도만 보고 코드를 짜는 것은 어색해 더 많은 연습이 필요할 것 같다고 느꼈다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료
공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료