컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2023년 10월 30일 (월)

제출일자: 2023년 11월 04일 (토)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적

A. 제목

FIFO

B. 목적

First-In-First-Out (FIFO)에 대해 이해하고 queue의 구조를 공부한다. 이를 이용하여 data의 개수에 따라 status flag, handshake signal을 출력하도록 설계하도록 한다. 데이터의 저장은 register file을 이용하여 설계할 수 있도록 한다.

2. 원리(배경지식)

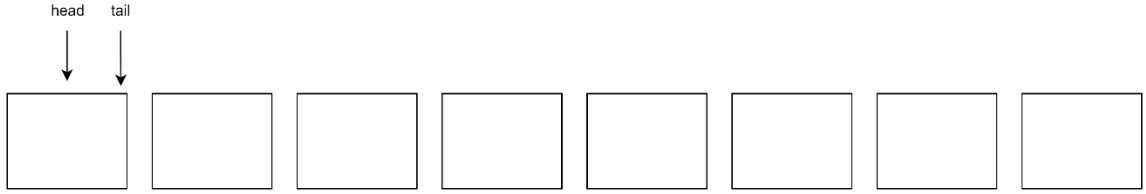
A. FIFO

fifo(first-in-first-out), 선입선출은 먼저 들어온 데이터가 먼저 처리되고 처리가 끝날 때까지 다음 데이터는 대기 상태에 있는 형식을 말한다. 데이터 구조의 queue도 이러한 fifo 형식을 사용하고 있다. 반대로 마지막에 들어온 데이터가 처음으로 나가는 형식인 lifo(last-in-first-out)는 stack 자료구조에서 쓰이고 있다.

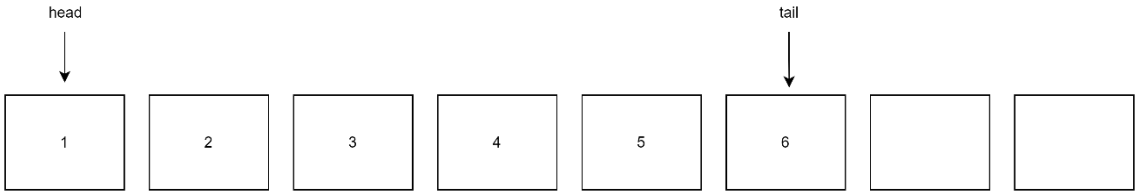
queue에서는 fifo를 쓰기 때문에 가장 처음 데이터를 가리키는 front와 마지막 데이터를 가리키는 rear이 존재한다. 이번 실습에서는 front를 head로, rear를 tail로 표기하여 사용하게 된다. head는 가장 처음 들어온 데이터를 삽입 과정을 반복하는 동안 계속 똑같이 가리키게 되며, 추가로 들어오는 데이터들은 tail을 다음 메모리 주소로 업데이트 시키면서 저장한다. head와 tail은 메모리의 주소에 접근하는 용도인 것이다. 여기서는 register file을 사용하므로, 해당 register 주소에 접근하며 저장하고 삭제할 수 있게 한다.

queue의 삽입, 삭제 과정은 다음과 같다.

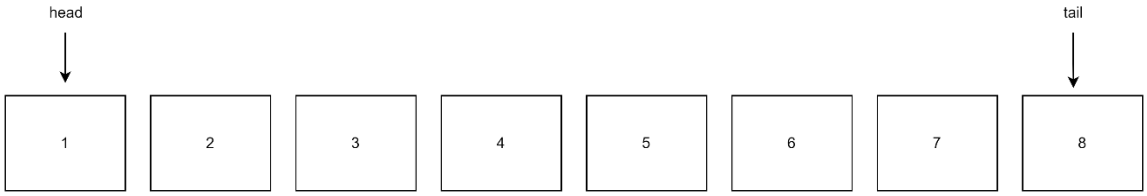
1. 데이터가 들어오지 않은 처음 상태에서는 head와 tail이 둘 다 첫번째 칸을 가리키며, empty상태가 된다.



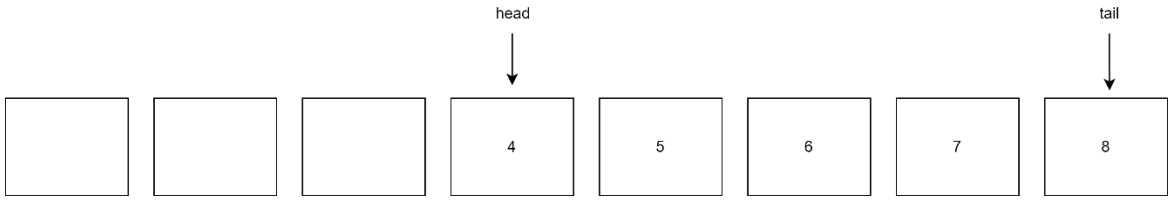
2. 데이터가 들어오면 tail을 업데이트 시키면서 다음 메모리 주소에 저장하게 된다.



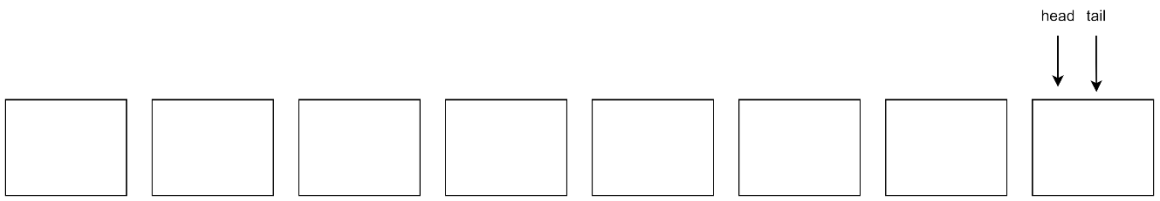
3. 모든 메모리에 데이터가 들어간다면, tail은 마지막 주소를 가리키고 있으며 현재 상태는 full이 된다. 이때는 더 이상 데이터를 저장할 수 없게 되므로 더 들어오면 error 상태가 된다.



4. 삭제할 때는 fifo구조라서 가장 처음 들어온 것부터 삭제되므로, head를 업데이트 시키면서 데이터를 삭제시킨다. 이때 실제 데이터 자체를 삭제시킨다기 보다, head가 더 이상 해당 메모리 주소를 가리키지 않게 되면서 삭제된다는 개념에 가깝다.



5. 마지막으로, 모든 데이터가 삭제되면 empty상태가 된다. 이때, 다시 원활하게 반복하여 삽입하고 삭제하는 과정을 수행할 수 있도록 주로 원형 큐로 만들어서 head와 tail이 다시 첫번째 칸을 가리키도록 만들기도 한다.



3. 설계 세부사항

1) fifo

-I/O

Input	clk	1-bit
	reset_n	1-bit
	rd_en	1-bit
	wr_en	1-bit
	d_in	32-bit
output	d_out	32-bit
	full	1-bit
	empty	1-bit
	wr_ack	1-bit
	rd_ack	1-bit
	rd_err	1-bit
	data_count	4-bit

2) fifo_ns

input	wr_en	1-bit
	rd_en	1-bit
	state	3-bit
	data_count	4-bit
output	next_state	3-bit

3) fifo_cal

input	state	3-bit
	head	3-bit
	tail	3-bit
	data_count	4-bit
output	we	1-bit
	re	1-bit
	next_head	3-bit

	next_tail	3-bit
	next_data_count	4-bit

4) fifo_out

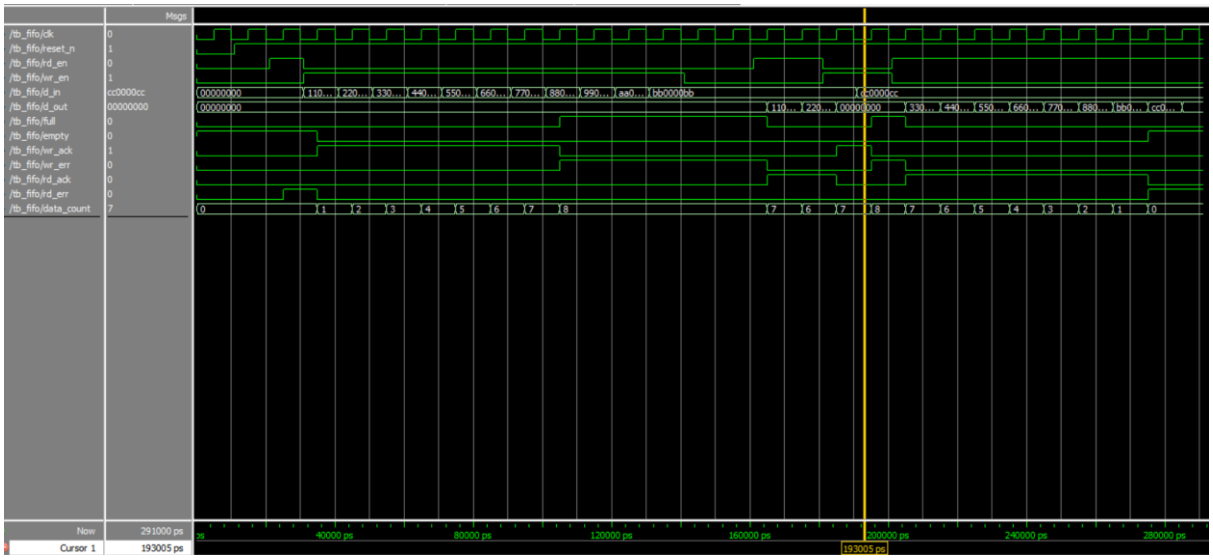
input	state	3-bit
	data_count	4-bit
output	full	1-bit
	empty	1-bit
	wr_ack	1-bit
	wr_err	1-bit
	rd_ack	1-bit
	rd_err	1-bit

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

1) fifo

처음에 데이터 개수가 0인 상태에서 READ를 하면 RD_ERR로 되고, 이후 we신호가 켜져서 데이터가 차례로 들어오면 data_count로 개수를 세면서 tail을 업데이트 시킨다. Register file에 8개가 다 차면 FULL이 되면서 WR_ERR로 더 이상 데이터를 저장할 수 없게 되며, re신호가 켜져서 읽기 시작하면 data_count는 줄어들면서 head가 업데이트 된다. 이때 중간에 we로 신호를 바꾸고 데이터를 저장하면 tail이 가리키는 곳에 저장되며, data_count가 다시 늘어난다. Re로 신호를 바꾸면 head가 업데이트되면서 이후 저장했던 곳까지 다 삭제되게 된다. 더 이상 데이터가 없으면 RD_ERR가 되면서 empty신호가 켜진다.

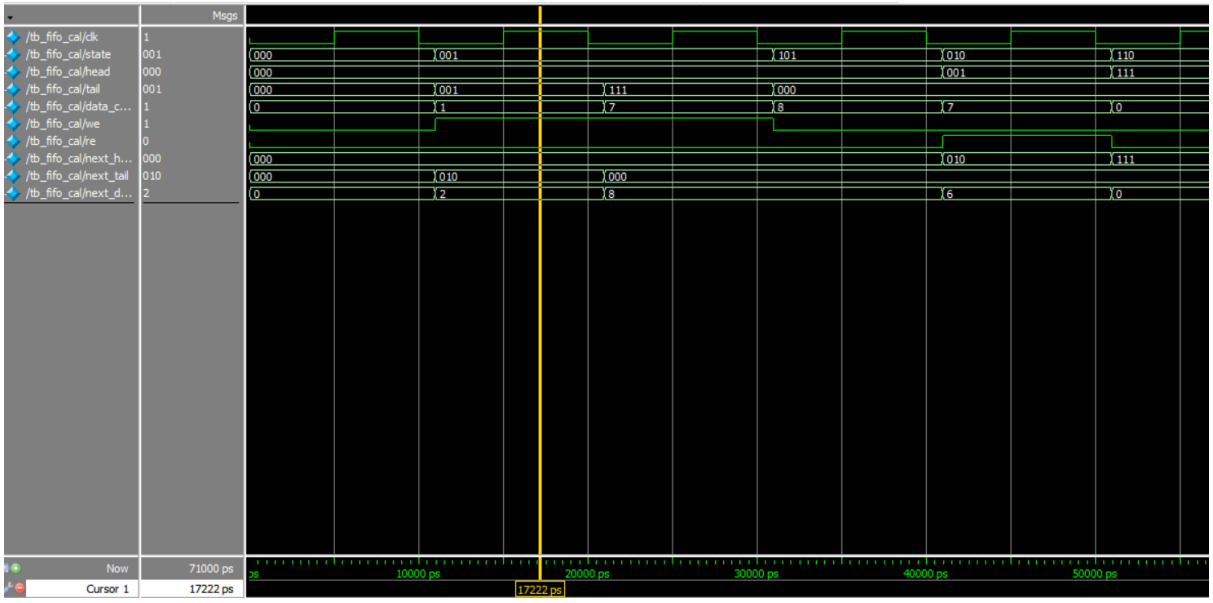


2) fifo_ns

state에 따라 next_state를 출력하는 sub module로, 현재 state에서 data_count가 올라가고 줄어들음에 따라 next_state가 출력된다.

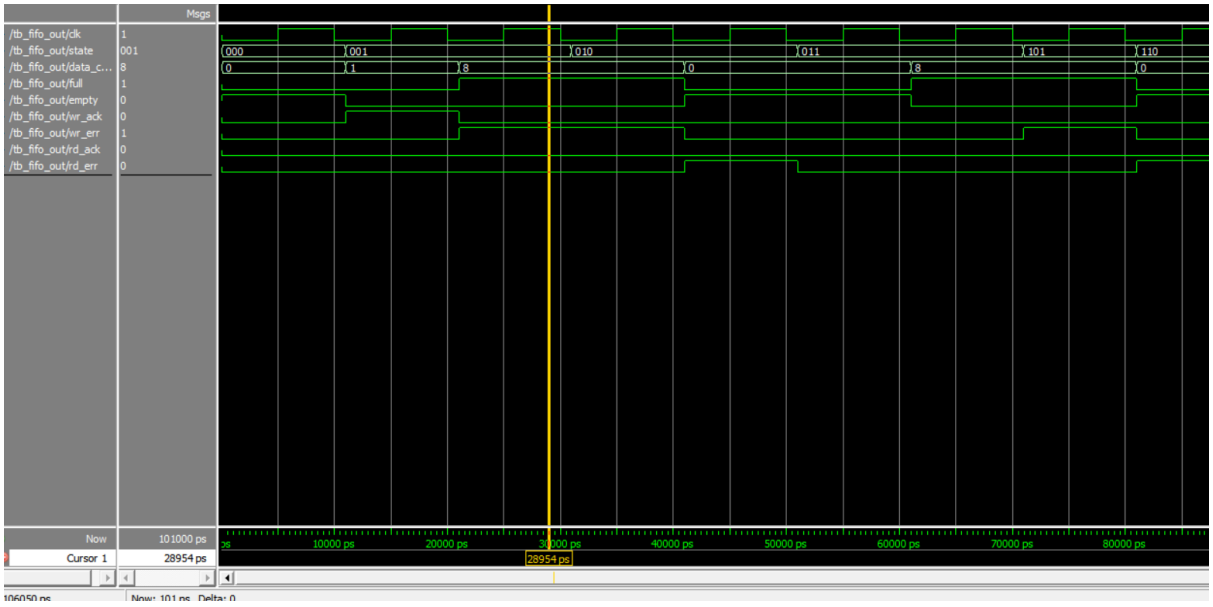
3) fifo_cal

state, head, tail, data_count에 따라 we, re, next_head, next_tail, next_data_count를 출력하는 sub module로, 표시한 곳을 보면 현재 WRITE일 때 we가 켜지고 re는 0이며, next_head는 그대로지만 next_tail이 다음 메모리 저장 주소를 가리키며 next_data_count가 다음 개수를 출력하고 있음을 보여준다.



4) fifo_out

state, data_count에 따라 full, empty, wr_ack, wr_err, rd_ack, rd_err를 출력하는 sub module로, 표시한 곳을 보면 WRITE면서 데이터 개수가 8개일 때 full이 되고 wr_err가 켜진 것을 볼 수 있다.

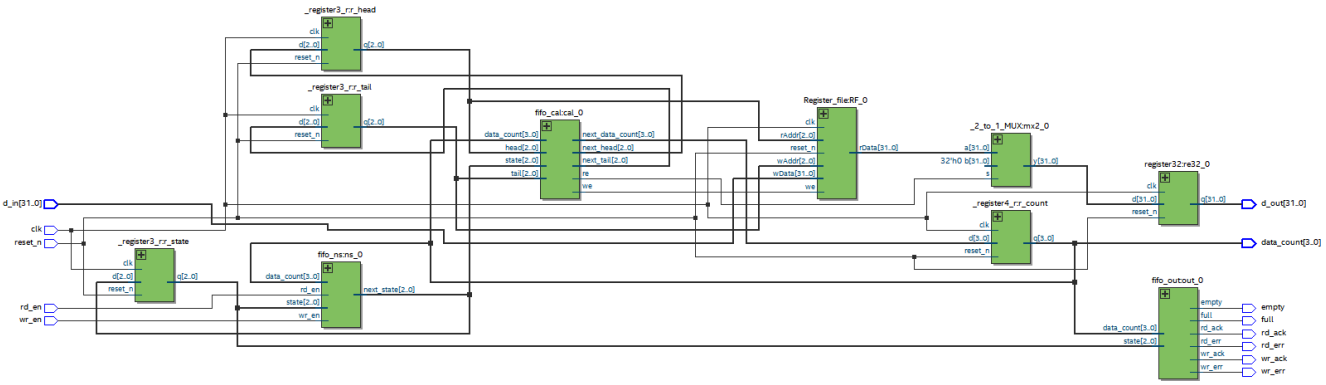


B. 합성(synthesis) 결과

fifo의 RTL viewer와 flow summary이다.

-RTL viewer

next state logic과 calculate address logic, output logic, register file module과 그 사이에 state, data_count, head, tail을 계산하는 register이 있고 마지막에 register file에서 나온 rData는 2-to-1 mux, register을 거치는 과정임을 확인할 수 있다.



-Flow summary

전체 회로의 크기가 155로, 그전에 진행했던 회로들보다 크기가 더 큰 것을 확인할 수 있다. 또한 register도 더 사용한 것을 알 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Sat Nov 04 15:00:18 2023
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)	155 / 41,910 (< 1 %)
Total registers	301
Total pins	78 / 499 (16 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

처음에 설계한 뒤 실행시켰을 때 data_count가 x값으로 안나오면서 d_out값까지 제대로 나오지 않는 상황이 발생하였다. 이에 RTL viewer를 보면서 회로를 다시 확인해보니, 마지막에 register file에서 출력된 값을 mux와 register를 안 거치고 바로 출력했음을 확인하였다. 이에 모듈들을 추가하여 d_out이 정상적으로 나오도록 수정하였다. 또한 data_count는 계산하는 case문에서 처음에 INIT 상태일 때 next_data_count<=data_count; 로 코드를 작성해서 기존 data_count값이 어떤 것인지 모르니 값이 안 나왔던 것이었다. 그래서 next_data_count<=4'b0; 으로 수정하고 다른 state일 때도 값이 안 바뀌더라도 기존 값을 할당해주는 식으로 작성하였더니 정상적으로 출력될 수 있었다.

B. 결론

fifo, queue의 구조에 대해 베릴로그로 구현해보면서 FSM으로 설계할 수 있다는 점을 깨달았다. state, data_count, head, tail을 register에서 받아서 계산할 때에는 먼저 next값들이 input으로 들어가고, output은 current 값들로 나온다는 것을 확실히 알 수 있었다. 또한 register file에 저장하고 삭제할 때에는 head와 tail을 메모리의 주소처럼 사용하여 접근하면 된다는 점도 공부할 수 있었다. 이런 식으로 next logic과 output logic으로 나누어서 설계를 한다면, fifo를 응용하여 lifo(Last-in-First_out)인 stack도 설계할 수 있을 것 같다는 생각이 들었다.

6. 참고문헌

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007