컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2022년 11월 1일 (화)

제출일자: 2022년 11월 2일 (수)

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화요일 0,1,2

학 번: 2019202050

성 명: 이강현

1. 제목 및 목적

A. 제목

FIFO

B. 목적

실험을 통해 first in first out 구조인 FIFO을 이해하고 그러한 구조를 띄는 Queue 자료구조에 대해 이해한다. State machine과 Register file등을 이용하여 fifo을 직접 구현해 본다.

2. 원리(배경지식)

<Stack>

Stack은 LIFO(Last Input First Output)로 후입선출 자료구조를 말한다. 스택에서 데이터의 추가시에는 스택의 맨 아래서부터 순차적으로 쌓이고 이러한 과정을 push라고 칭한다. 또한 스택에서 데이터가 삭제될 시에는 스택의 맨 위에서부터 순차적으로 나가며 이를 pop이라 칭한다. Top은 맨 위 즉 pop을 할 때 나가는 데이터를 가리키는 역할을 한다.

<Queue>

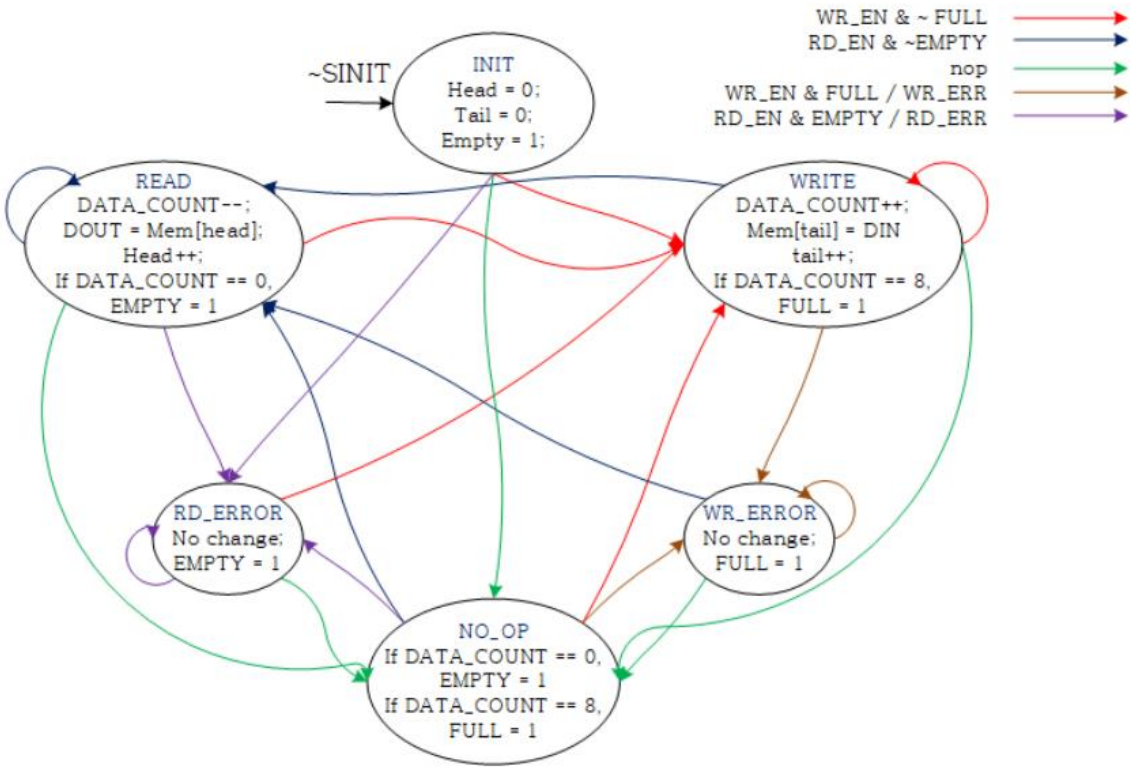
Queue는 FIFO(First Input First Output)로 선입선출 자료구조를 말한다. 따라서 스택과는 달리 가장 먼저 들어온 데이터가 가장 먼저 나가게되는 구조이다. 스택과 다른 점은 pop을 할 때 맨 아래데이터부터 시행되므로 가장 먼저 들어온 데이터가 가장 먼저 나갈 수 있다.

<Register File>

Register File은 register로 이루어져 있으며 주소를 통해 데이터의 접근과 수정이 가능하다. Write operation, read operation, 8_32bits register들로 이루어져 있다. Write operation logic에서 디코더를 통해 8개의 register중 하나의 enable 신호를 1로 만들어주면 이를 read operation logic에서 mux를 통하여 8개의 register 중 한 개를 선택하여 선택된 레지스터의 값을 출력하는 방식으로 작동한다.

3. 설계 세부사항

<state diagram>



<Encoding>

State	Encoding
Init	3'b000
Read	3'b001
Write	3'b010
No_op	3'b011
Rd_err	3'b100
Wr_err	3'b101

<output table>

Input	Output	
Data count	Full	Empty
0	0	1
8	1	0
1~7	0	0

Input	Output			
State	Wr_ack	Rd_ack	Wr_err	Rd_err
000	0	0	0	0
001	0	1	0	0
010	1	0	0	0
011	0	0	0	0
100	0	0	0	1
101	0	0	1	0

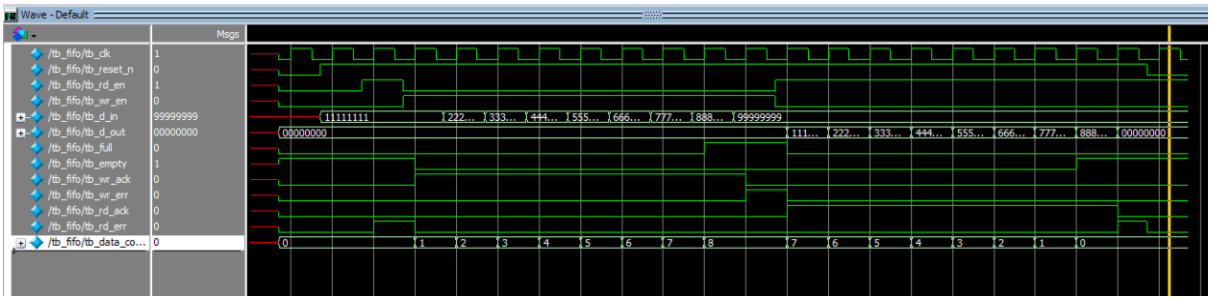
지난 시간에 구현한 register file을 수정하여 fifo를 구현하였고 fifo 모듈 아래에 fifo_ns, fifo_out, fifo_cal_addr, register file 이렇게 4가지 하위 모듈들이 있다.
Input으로 reset과 clock 그리고 읽기 쓰기를 선택할 수 있는 enable signal과 쓰기에 사용되는 데이터가 있고 output으로는 데이터의 양을 알려주는 data count와 데이터가 가득찼는지 알수 있는 full, 데이터가 없다는 것을 알려주는 empty그리고 state에 따라 실행이 잘 이루어졌는지 확인할 수 있는 handshake signal인 wr_ack,rd_ack,wr_err,rd_err와 마지막으로 읽기시에 데이터를 확인할 수 있는 d_out이 있다.

Next logic은 always문과 case문을 통해 input값이 변경될 시에 state를 조건에 맞춰 바꾸도록 하였고 fifo_cal_addr모듈부분은 state에 따라 비슷하게 always문과 case문을 사용하여 head,tail,data_count를 수정하도록 구성하였고 마지막으로 fifo_out부분은 현재 state에 맞게 위와 동일하게 always문과 case문을 통해 flag를 수정하였다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

<fifo>(top module)

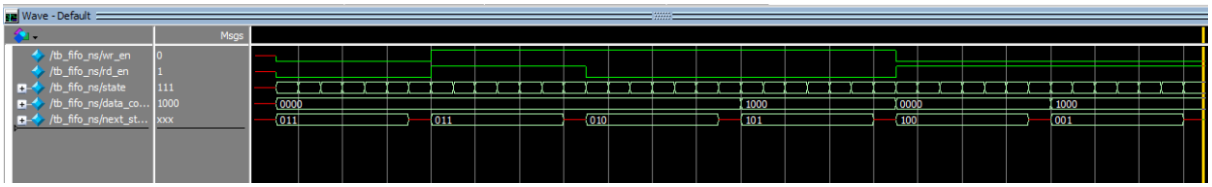


fifo모듈의 시뮬레이션이다. 테스트벤치는 모든 state를 확인할 수 있게끔 구성하였으며 그에 따라 handshake signal또한 확인 가능하다.

우선 reset이 0일때는 init state이므로 empty값을 제외한 모든 값들이 0으로 초기화된다. 그 후 reset을 1로 설정하고 만나는 상승엣지에서 enable값들이 모두 0이므로 d_in에 11111111값이 들어왔지만 아무런 행동을 취하지 않는다. 그 후 rd_en이 1로 설정되자 empty값이 1인상태이고 아무런 데이터가 없는 상태이므로 rd_err가 1로 세팅되며 read에 오류가 났음을 나타낸다.

그후 wr_en을 1로 설정하고 rd_en을 0으로 설정하여 각각 값을 11111111~부터 99999999까지 순차적으로 집어넣으니 11111111을 넣었을 때는 첫데이터를 넣은 것이므로 empty가 0으로 바뀌고 그후 data_count는 점차 증가하며 wr_ack값또한 1로 세팅되어 유지된다. 또한 88888888까지 넣었을 때 full signal이 1이되며 99999999을 넣으려고 할 때 wr_err가 나타나는 것을 확인할 수 있다. 그 이후로는 read를 하여 데이터를 하나씩 읽으니 data_count가 하나씩 감소하고 rd_ack는 1로 유지된다. 마지막으로 data_count가 0일 때 즉 empty일 때 데이터를 읽으려고 하니 rd_err가 1로 세팅되어 error를 잘 나타낸것을 확인할 수 있고 reset이 0으로 세팅되며 프로그램이 종료된다.

<fifo_ns>(sub module)



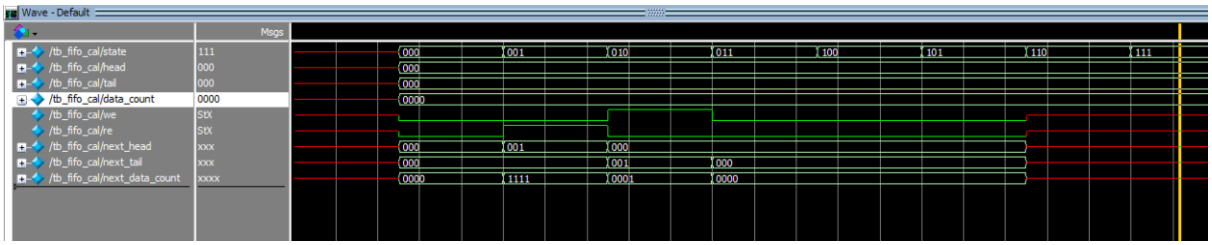
위 시뮬레이션을 보면 rd_en, wr_en, state는 모든경우를 고려했고 data_count는 empty일 때 full일 때만을 고려하여 testbench를 진행했다. Data_count는 값이 0000일때는 read오류 write가능, 1000일때는 write 오류, read가능이므로 다음상태를 모두 고려할 수 있기 때문이다.

Rd_en과 wr_en이 모두 0이거나 1일때는 no_op로 구현하였기 때문에 no_op의 상태인 011이 잘 나오는 것을 확인할 수 있고 x값이 나오는 것은 state가 6개이므로 state가 110과 111이 될때는 default값을 3'bx로 주었기 때문에 이것이 잘 나오는지 확인한 것이다.

Rr_en과 wr_en이 서로 다를때는 full과 empty값을 고려하여 read, write,

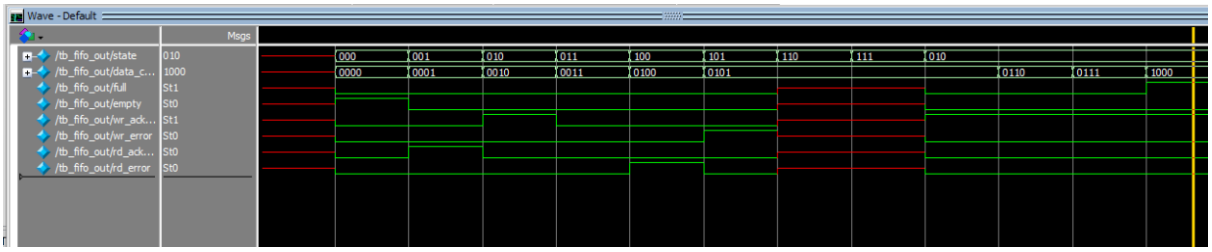
rd_err, wr_err등의 상태들이 잘 나오는 것을 확인할 수 있다.

<fifo_cal_addr>(sub module)



Calculate address하는 모듈의 시뮬레이션이다. 각각의 state에 따라 head,tail,datacount를 모두 0으로 설정하고 state에 맞는 행동을 하는지만을 확인하면 되기 때문에 위와같이 testbench를 작성하였다. 우선 000(init)일때는 모두 0이며, 001(read)일때는 head가 1증가, read_enable이 1 그리고 data_count는 1감소이나 초기값이 0이었기 때문에 1111이 되었다. top 모듈에서는 next state logic에 의해 data_count가 0일 때 read state상태일 수 없고 rd_err state로 넘어가기 때문에 read state일 때 데이터 수를 하나 감소하였다라는 것만 나타나면 검증에는 무리가 없다고 판단하였다. 010(write)일때는 tail이 1증가, write_enable이 1을 나타내고 data_count가 1증가하였다.
011(no_op)일때는 초기값 그대로 아무런행동도 취하지 않고 100(rd_err),101(wr_err)또한 아무런 행동도 취하지 않아 정상적임을 확인한다. 110과 111은 state가 없으므로 default로 3'bx를 나타내도록 구현했으며 위와 같이 나왔다.

<fifo_out>(sub module)



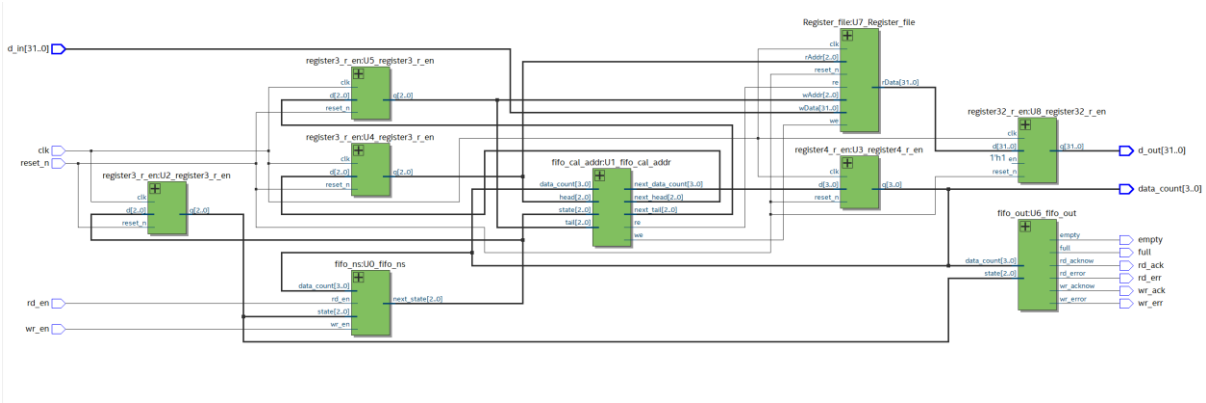
Fifo_out의 시뮬레이션이다. 모든 state에 따라 flag가 어떻게 세팅되는지 확인할 수 있다. 000일때는 empty만 1이고 001일때는 rd_ack가 1, 010일때는 wr_ack가 1, 011일때는 모두 0, 100일때는 rd_err가 1,101일때는 wr_err가 1이고 없는 state에 대해서는 x, data_count는 0일때는 empty, 1~7일때는 empty

와 full이 모두 0, 8일때는 full이 1로 올바른 결과를 나타내었다.

B. 합성(synthesis) 결과

<fifo>(top module)

<RTL viewer>



모든 단자가 올바르게 잘 연결되었다.

<flow summary>

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Nov 02 01:55:29 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

사용된 register는 301개이며 pin은 78개가 사용되었다. 성공적인 flow status를 나타낸다.

5. 고찰 및 결론

A. 고찰

Fifo를 구현하면서 data를 queue의 자료구조 형태로 어떻게 관리하는지를 알 수 있었고 head와 tail과 같이 데이터의 처음과 끝을 나타낼 수 있는 것을 사용하면서 데이터에 쉽게 접근이 가능하였다. 또한 tail과 head은 3비트로 0에서 7까지밖의 수를 나타낼수 밖에 없는데 7에서 1을 증가한다면 0이되므로 원형큐의 형식을 띄기 때문에 단순히 1을 증가하는 연산만으로 큐를 구현할 수 있었다.

B. 결론

Fifo 모듈을 직접 구현해보면서 c++에서 구현한 것과는 베릴로그가 많이 다른 것을 알 수 있었다. 실제 회로로 구현이 가능해야하기 때문에 데이터를 저장하는 register file부터 구현하기까지가 어려웠고 c++에서는 단순히 first in first out을 구현하는 것이었어서 클래스하나로 선언이 가능했다면 베릴로그로 구현한 fifo는 많은 하위모듈들을 사용해야만 했다. 이를 통해 구조적으로 queue라는 자료구조를 좀 더 잘 이해한 것 같다.

6. 참고문헌

공영호 교수님/컴퓨터공학기초실험2/2022