컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2023년 10월 30일 (월)

제출일자: 2023년 11월 07일 (화)

학 과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습 분반: 월요일 0, 1, 2

학 번: 2020202031

성 명: 김재현

1. 제목 및 목적

A. 제목

FIFO

B. 목적

FIFO에는 읽기 및 쓰기 포인터를 관리하는 제어 논리가 있습니다. 상태 플래그를 생성하고 사용자 로직과의 인터페이스를 위한 핸드세이크 신호를 제공하는 FIFO를 베릴로그 코딩을 통해 구현 해봄으로써 FIFO의 작동원리를 이해합니다.

2. 원리(배경지식)

A. D flip-flop

D flip-flop은 clock의 rising edge나 falling edge에서만 D값으로 출력이 바뀌게 된다. clk의 rising edge가 발생했을 때, output Q값으로 input D값이 전달되어 출력되고, clk의 값이 유지되고 있을 경우에는 output값으로 input D값이 output Q값으로 전달되지 않고, 이전의 Q값이 출력으로 나오게 된다.

CLK	Q
rising edge or falling edge	D
other case	이전 Q

B. resettable D flip flop

D flip flop에 reset 기능이 추가된 D flip flop이다. resettable D flip flop에서 reset은 active low 또는 active high에 동작한다. active low에서 동작할 때, reset이 0이면 input으로 D, clk값을 받지 않고, 1이면 clk의 edge 여부에 따라 output의 Q값이 달라진다. 반대로 active high에서 동작할 때는 reset이 1이면 input으로 D, clk값을 받지 않고, 0이면 clk의 edge 여부에 따라 output의 Q값이 달라진다.

Input (rising edge)			Output
R	D	CLK	Q
0	X	X	0
1	0	rising	0
1	1	rising	1
1	x	falling or 0 or 1	이전 Q

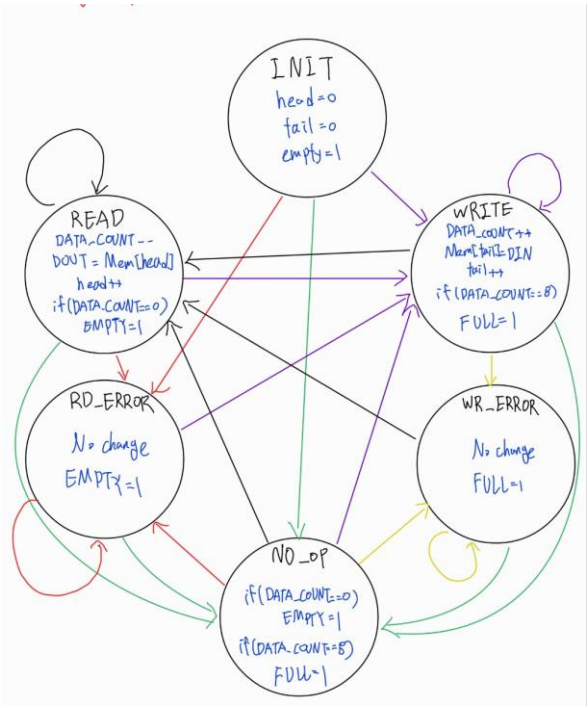
C. register

입력된 값을 저장하고 그 다음의 신호가 들어올 때 output으로 출력해주는 장치이다. verilog에서는 reg로 선언을 하고 변수 명을 적으면 register의 역할을 하게 된다. wire와 달리 값을 저장하고 있는 기능을 수행한다.

D. Queue

Queue는 먼저 입력된 값이 가장 먼저, 나중에 입력된 값이 가장 나중에 출력되는 알고리즘이다. 예를 들자면, 음식점 줄을 서는 것을 생각해볼 수 있다. 이번 실험에서 구현하는 FIFO도 동일한 방식을 구현한다.

3. 설계 세부사항



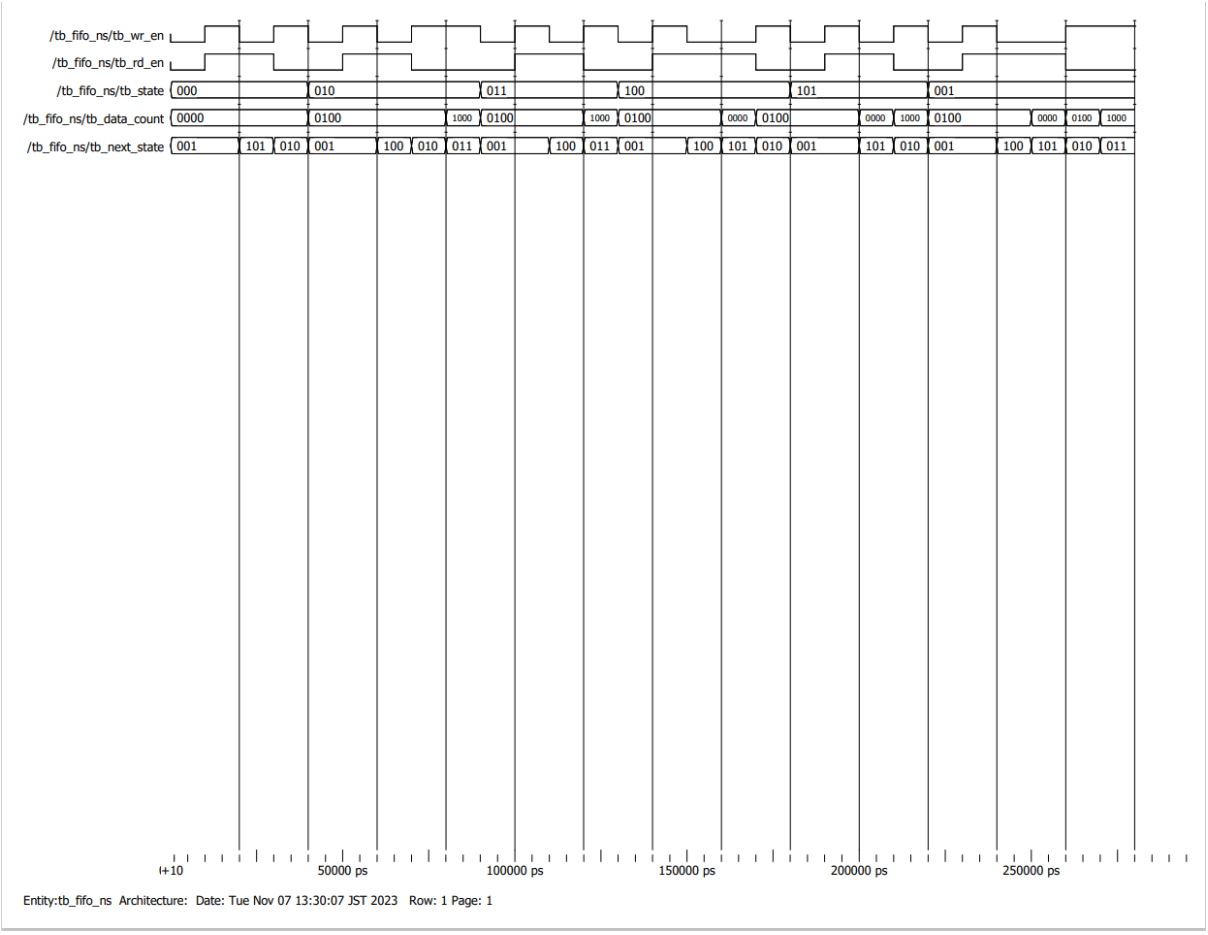
status flag로 full과 empty를 제공하고, invalid(인식 불가능한) 요청에는 FIFO 의 상태를 변화시키지 않습니다. 또한 output으로 handshake signal을 input we, re에 대하여 제공한다. next state logic은 input으로 wr_en, rd_en을 받고, 내부에서 현재 state와 data_count를 받아 next_state를 출력합니다. calculate address logic은 state, data_count, head, tail을 input으로 받아 next_state에 대한 next_head, next_tail값과 next_data_count를 계산해서 output으로 내보냅니다. output logic은 현재 stae와 data_count를 받아 full, empty, handshake signal을 출력합니다. register file은 원하는 위치의 register에 저장하기 위해서 입력된 3bit값을 8bit one hot encoding으로 변환하여 각 bit의 값이 하나만 변해도, 다른 말로 하면 값이 변할 때 register의 사용개수를 줄일 수 있도록 write operation module에

서 처리하여 output으로 내보냅니다. 이 내보내진 8bit을 register module에서 원하는 register의 위치에 입력된 32bit짜리 input값을 저장하도록 합니다. 그리고 다음 module인 read operation module에 각 register에 저장돼 있는 값들을 output으로 전달해줍니다. 그러면 read operation에서 3bit input값을 입력 받아 어느 register의 값을 출력할지를 판단하여 해당 register에 저장돼 있는 32bit짜리 값을 출력합니다..

fifo에서 state가 READ가 아닐 땐 output을 0으로 출력해주기 위해 register file read operation에서 8-to-1 MUX에서 re값을 input으로 받아, re가 1이면 rAddr에 알맞은 데이터를 출력하고, re가 0이면 0을 출력하도록 설계했습니다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



tb_fifo_ns의 testbench waveform입니다.

먼저, wr_en과 re_en이 같으면 next_state는 무조건 NO_OP가 됩니다.

state가 INIT일 때, data_count는 0이 됩니다. 따라서 사용자가 wr_en 0, rd_en 1을 입력하

면 next_state는 RD_ERROR가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WRITE가 됩니다.

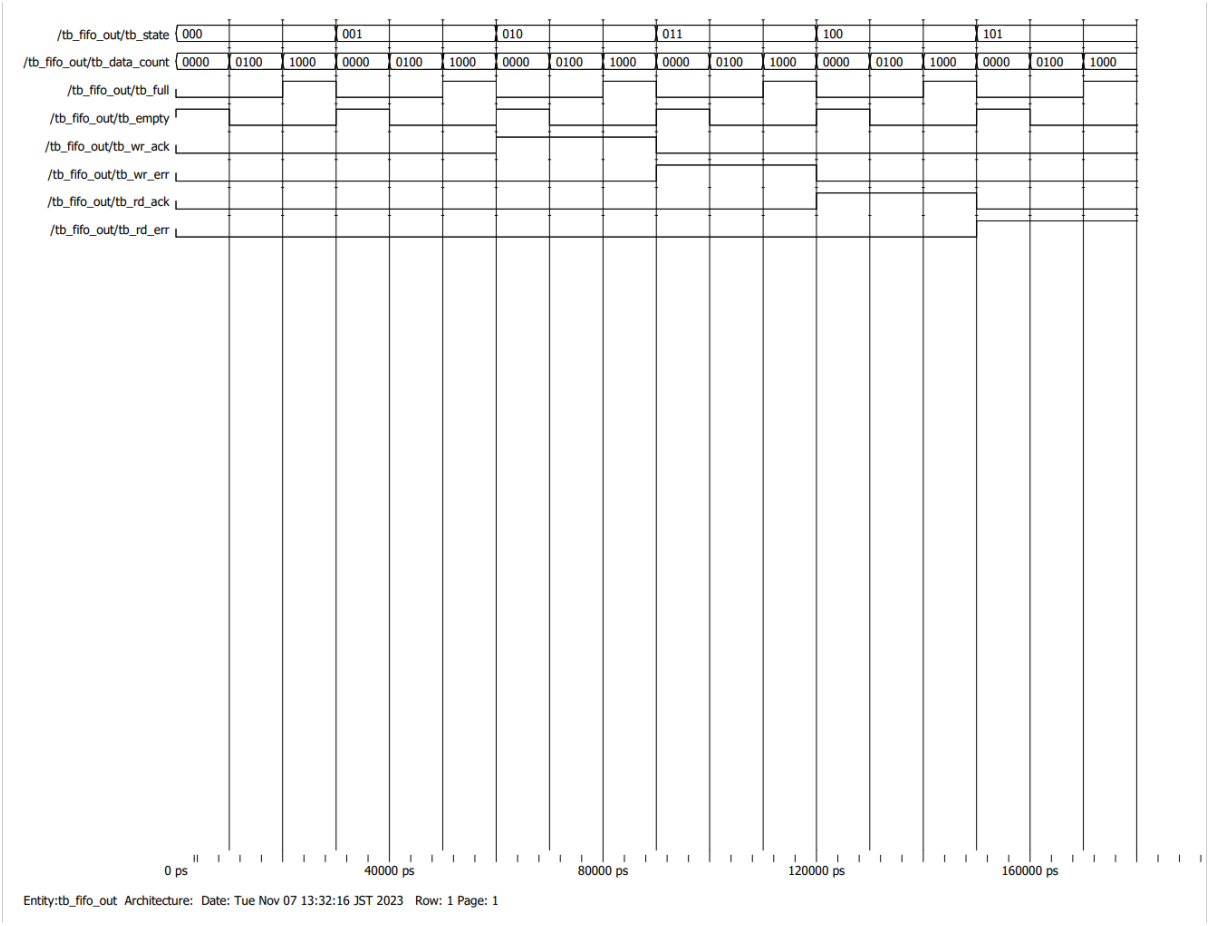
state가 WRITE이면 data_count는 1이상이 됩니다. 따라서 data_count가 8일 때, 사용자가 wr_en 0, rd_en 1을 입력하면 next_state는 READ가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WR_ERROR가 됩니다. data_count가 8 미만일 때, wr_en 0 rd_en 1을 입력하면 next_state가 READ가 되고, wr_en 1 rd_en 0을 입력하면 next_state가 WRITE가 됩니다.

state가 WR_ERROR이면 data_count는 8입니다. 따라서 사용자가 wr_en 0, rd_en 1을 입력하면 next_state는 READ가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WR_ERROR가 됩니다.

state가 READ이면 data_count는 8 미만이 됩니다. 따라서 data_count가 0일 때, 사용자가 wr_en 0, rd_en 1을 입력하면 next_state는 RD_ERROR가 되고, wr_en 1, rd_en 0을 입력하면 next_state가 WRITE가 됩니다. data_count가 0 초과일 때, wr_en 0 rd_en 1을 입력하면 next_state가 READ가 되고, wr_en 1 rd_en 0을 입력하면 next_state가 WRITE가 됩니다.

state가 RD_ERROR이면 data_count는 0입니다. 따라서 사용자가 wr_en 0, rd_en 1을 입력하면 next_state는 RD_ERROR가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WRITE가 됩니다.

state가 NO_OP이면, data_count 값에 따라 next_state가 결정됩니다. data_count가 8일 때, 사용자가 wr_en 0, rd_en 1을 입력하면 next_state는 READ가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WR_ERROR가 됩니다. data_count가 0일 때, wr_en 0 rd_en 1을 입력하면 next_state가 RD_ERROR가 되고, wr_en 1 rd_en 0을 입력하면 next_state가 WRITE가 됩니다. data_count가 0과 8 사이면, wr_en 0, rd_en 1을 입력하면 next_state는 READ가 되고, wr_en 1, re_en 0을 입력하면 next_state가 WRITE가 됩니다.

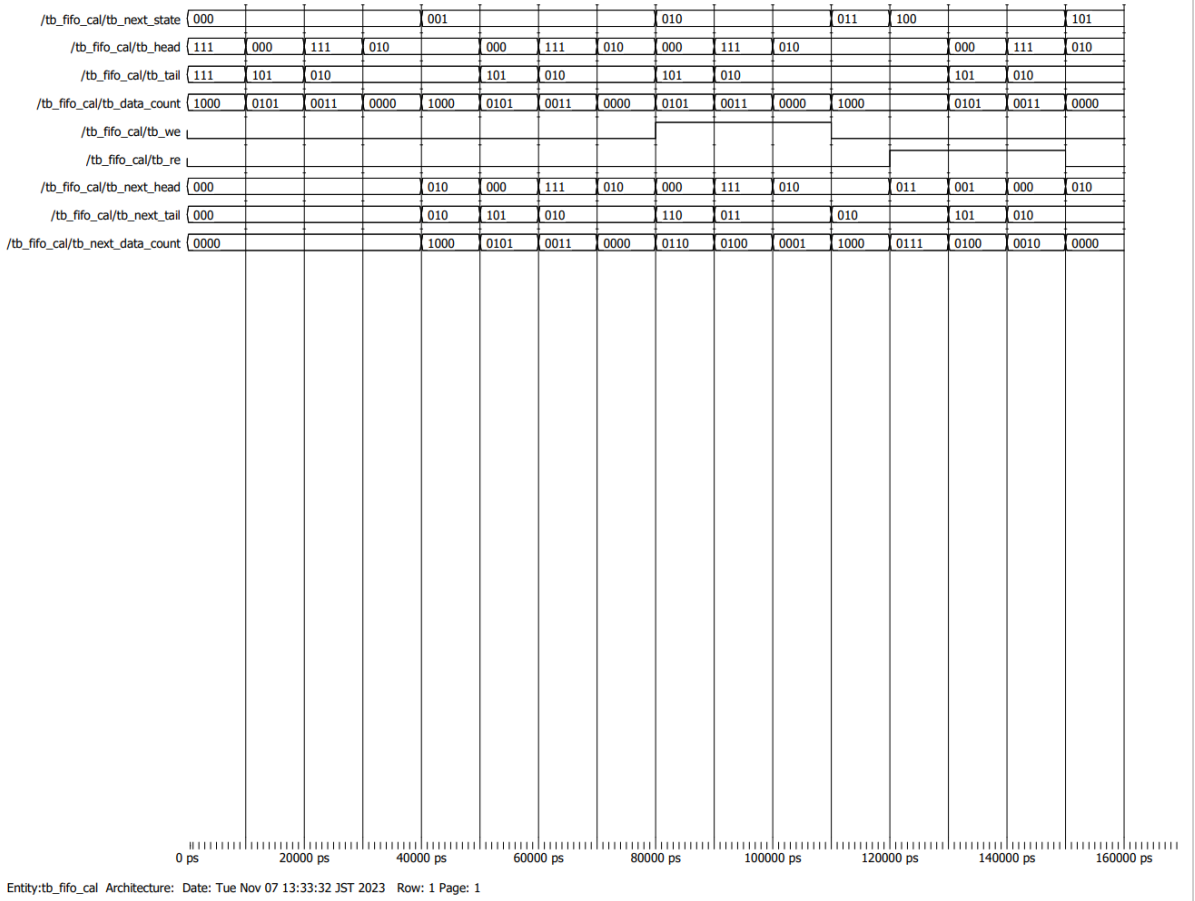


tb_fifo_out의 testbench waveform입니다.

state 값을 통해 wr_ack, wr_err, rd_ack, rd_err을 출력하고 data_count 값을 통해 full, empty를 출력합니다.

state가 INIT일 때 wr_ack, wr_err, rd_ack, rd_err 순으로 0000을, state가 NO_OP일 때 0000을, state가 WRITE일 때 1000을, state가 WR_ERROR일 때 0100을, state가 READ일 때 0010을, state가 RD_ERROR일 때 0001을 출력합니다.

data_count가 0일 때 full 0 empty 1이, data_count가 8일 때 full 1 empty 0이, data_count가 0과 8 사이일 때 full 0 empty 0이 출력됩니다.



tb_fifo_cal의 testbench waveform입니다.

next_state 값에 따라 next_head와 next_tail의 값, we와 re의 값을 변경해주고, next_data_count 값을 계산해줍니다.

next_state이 INIT일 때, we 0, re 0, next_head 0, next_tail 0, next_data_count 0을 출력합니다.

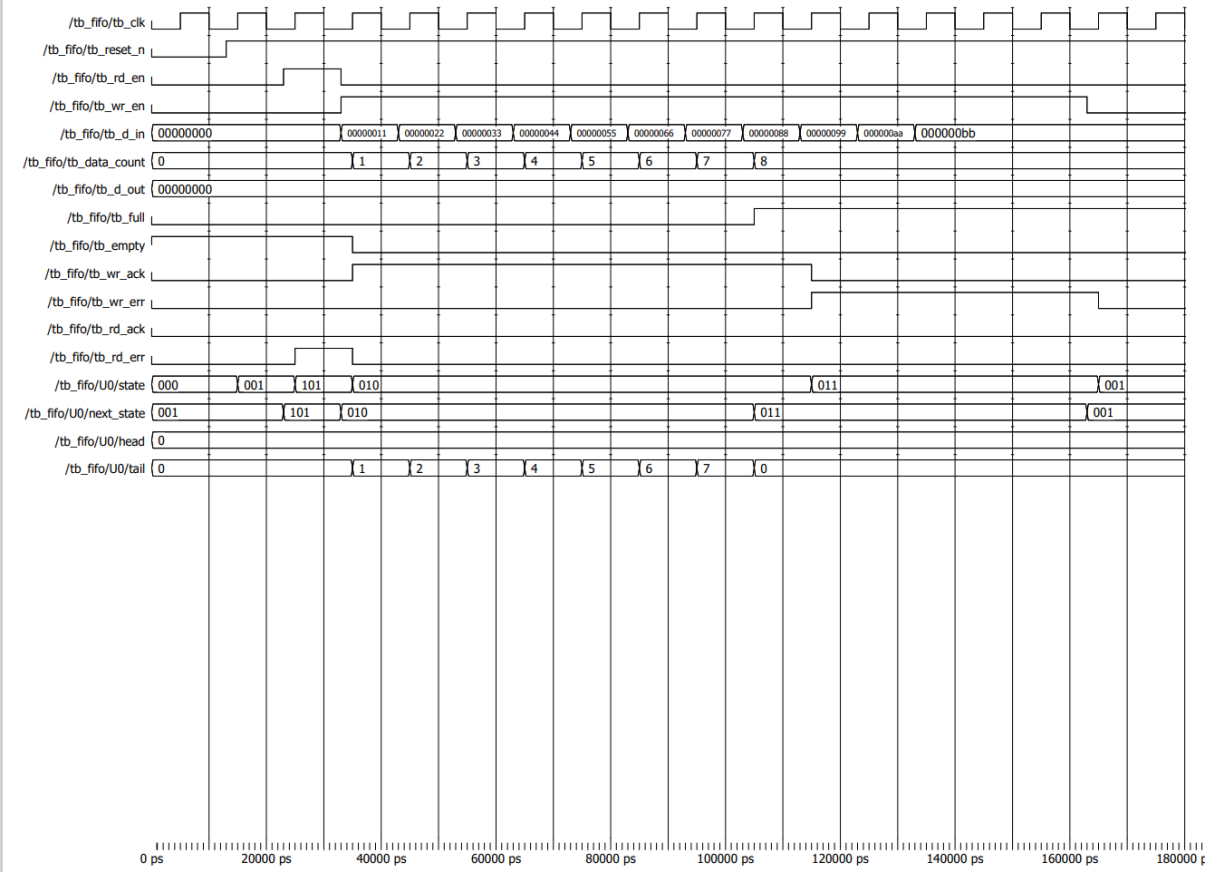
next_state이 NO_OP일 때, we 0 , re 0, next_head는 head와 동일하고 next_tail도 tail과 동일하며, next_data_count 또한 data_count와 동일하게 유지됩니다.

next_state이 WRITE일 때, we 1, re 0, next_head은 head와 동일하고 next_tail은 tail + 1이 됩니다. next_data_count 또한 data_count + 1을 출력합니다.

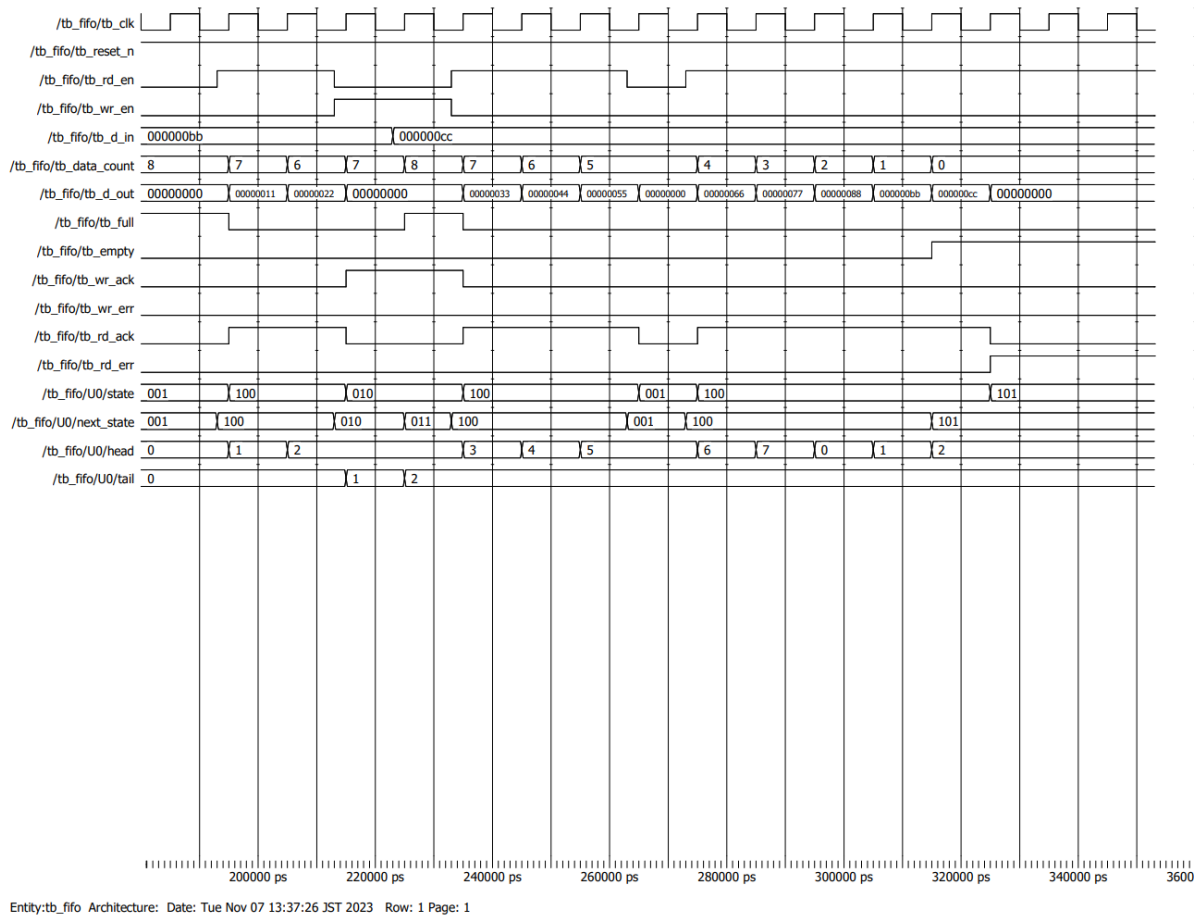
next_state이 WR_ERROR일 때, we 0, re 0, next_head은 head와 동일하고 next_tail도 tail과 동일하며, next_data_count 또한 data_count 와 동일하게 유지됩니다.

next_state이 READ일 때, we 0, re 1, next_head은 head + 1이 되고, next_tail은 tail과 동일하게 유지됩니다. next_data_count 또한 data_count - 1을 출력합니다.

next_state이 RD_ERROR일 때, we 0, re 0, next_head은 head와 동일하고 next_tail도 tail과 동일하며, next_data_count 또한 data_count 와 동일하게 유지됩니다.



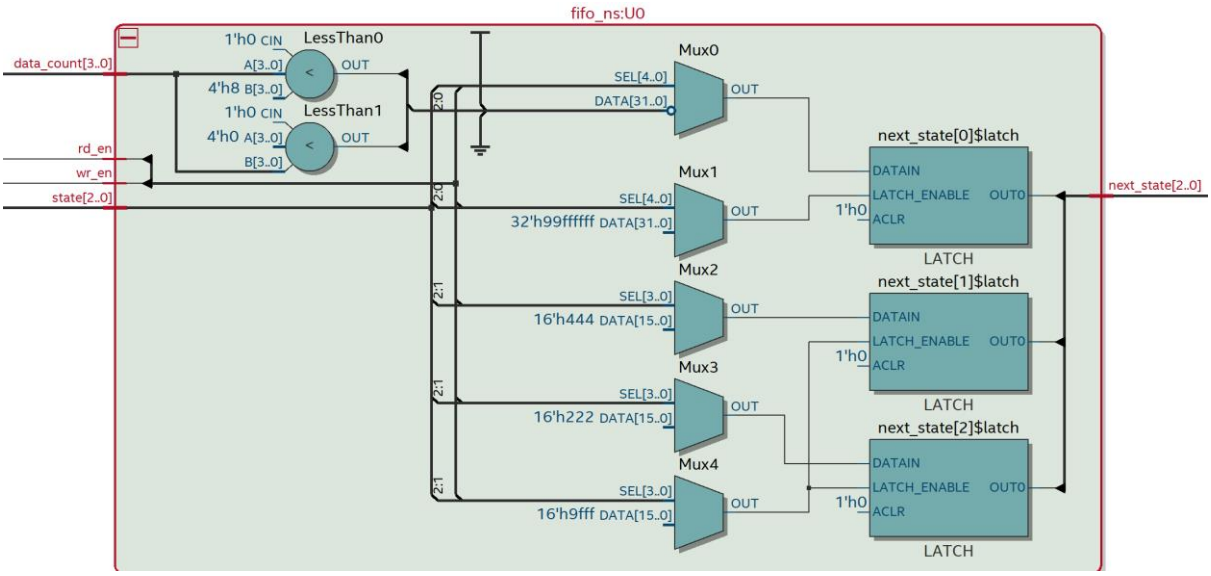
Entity:tb_fifo Architecture: Date: Tue Nov 07 13:36:41 JST 2023 Row: 1 Page: 1



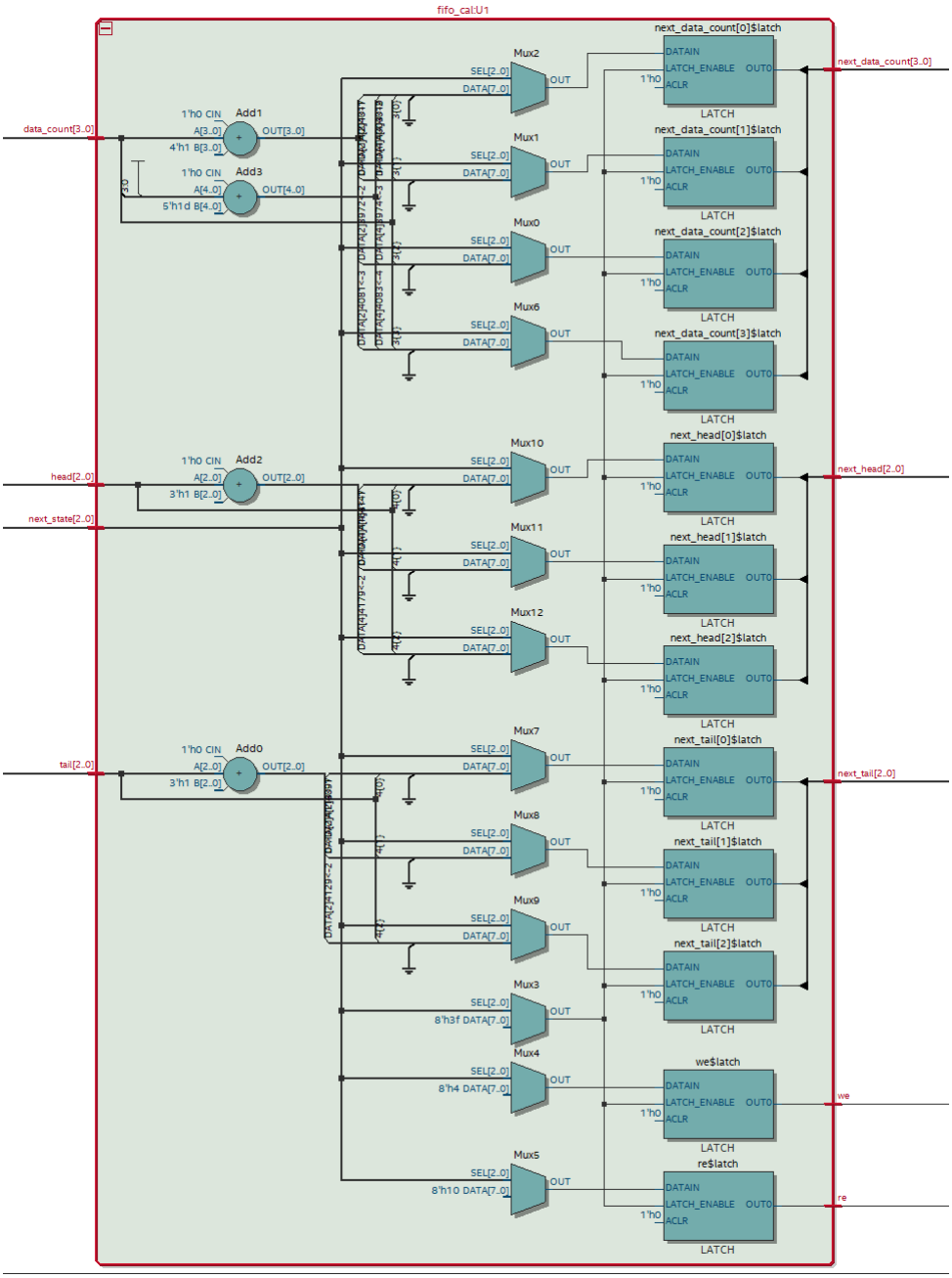
tb_fifo의 testbench waveform입니다.

11, 22, 33, 44, 55, 66, 77, 88이 register file에 push되고, 11, 22가 pop 됩니다. bb, cc가 push되고, 33, 44, 55가 pop됩니다. 그 후 66, 77, 88, bb, cc가 pop되면서 register file에 저장된 data의 개수는 0개가 됩니다.

B. 합성(synthesis) 결과

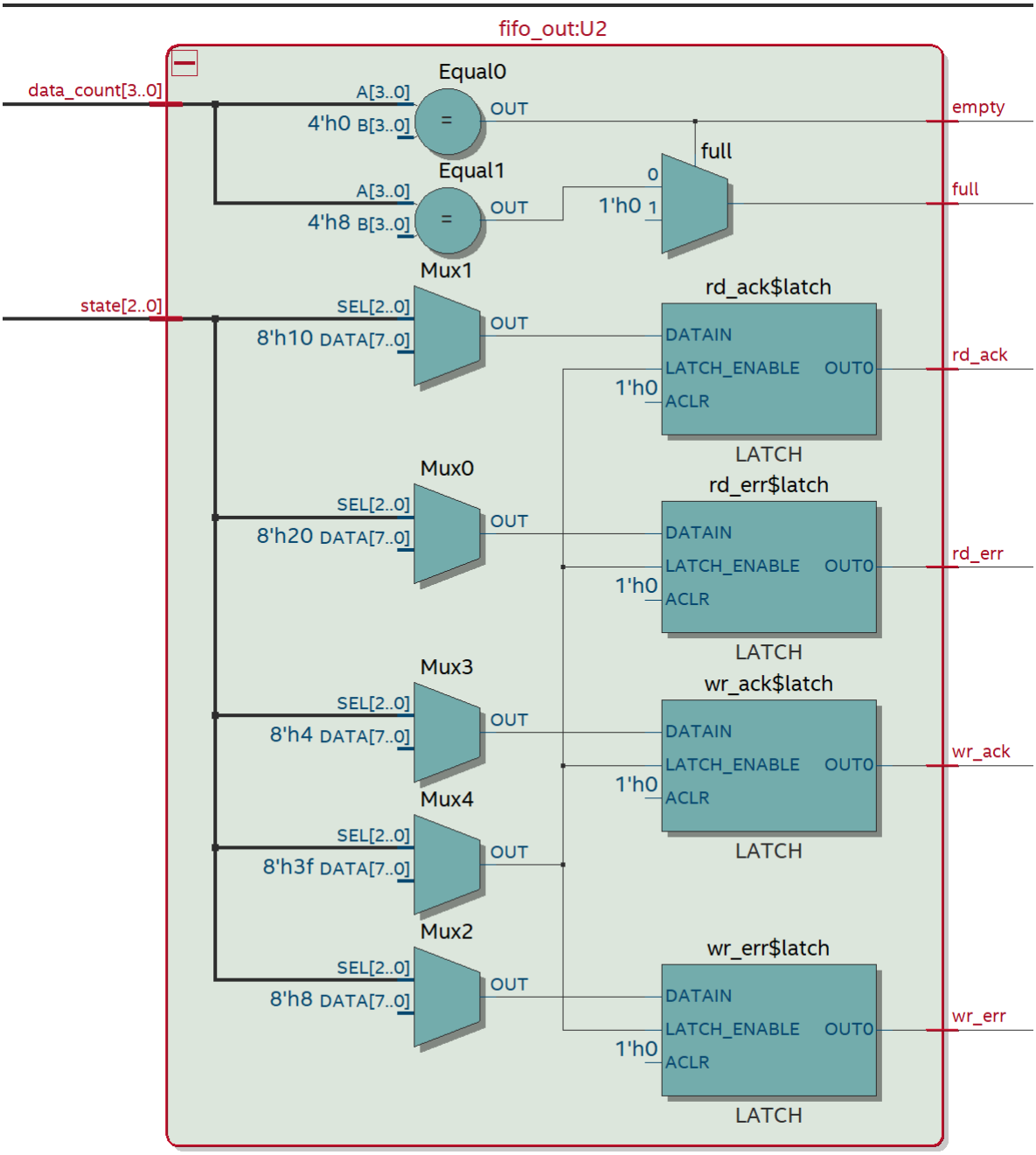


fifo_ns의 RTL Viewer입니다. input값으로 wr_en, rd_en을 입력 받고, state의 상태도 입력 받아서 next_state를 결정하는 module입니다. next_state로 INIT이 나올 수 없기 때문에 MUX가 INIT을 제외한 state 5개만이 존재하는 것을 확인할 수 있습니다.



fifo_cal의 RTL Viewer입니다.

data_count와 data_count + 1, head와 head + 1, tail과 tail + 1이 next_state에 의해 둘 중 하나만이 next_data_count, next_head, next_tail 로 출력되는 것을 MUX를 거쳐가는 것으로 확인할 수 있습니다.



fifo_out의 RTL Viewer입니다.

data_count가 0인지 8인지를 판별하여 0이면 empty 1 full 0, 8이면 empty 0 full 1, 그 외엔 empty 0 full 0 을 출력하는 것을 위 회로를 통해 확인할 수 있습니다.

state를 통해 실행하고자 하는 연산을 잘 수행했는지 알려주는 handshake 값 4개를 출력하는 것을 확인할 수 있습니다.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Nov 06 18:14:43 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)	149 / 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 %)

FIFO의 flow summary 화면입니다. 정해준 family와 device를 사용하였고, total pins의 값이 clk, reset_n, rd_en, wr_en 1bit + d_in 32bits + d_out 32bits + full, empty, wr_ack, wr_err, rd_ack, rd_err 1bit + data_count 4bits = 78이 나온 것을 확인할 수 있고, register의 값이 301, logic utilization(area)의 값도 149로 파악됩니다.

5. 고찰 및 결론

A. 고찰

Calculate Address Logic에서 현재 data_count의 값만을 참조하여, data_count가 0이면 re 0 we 1을 출력하고, data_count가 8이면 re 1 we 0을 출력하고, data_count가 1이상 7이하면 re 1 we 1을 출력했습니다. 하지만 이렇게 구현하니, state가 READ가 아닐 때도, d_out에 이전 값이 계속 유지되는 문제점을 발견하게 됐습니다. 이에, next_state를 참조하여 re, we값을 결정했습니다. we와 re는 dff을 거치지 않고 바로 register file로 연결되기 때문에, next_state에 맞는 we re 값을 register_file로 연결하면 다음 clock에서 state에 맞는 we re가 register_file에 입력되는 것입니다.

B. 결론

개념적으로만 알고있던 FIFO, 즉 Queue 자료구조를 회로를 통해 구성한다면 어떻게 설계해야 되는지에 대해 알게 됐습니다. fifo module을 구현하고 그에 딸린 sub module들

을 구현하면서 Queue에 대한 이해도를 높였습니다.

6. 참고문헌

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2023

이준환 교수님/컴퓨터공학기초실험2/광운대학교(컴퓨터정보공학부)/2023