컴퓨터 공학 기초 실험2 보고서

실험제목: Synchronous FIFO

실험일자: 2023년 11월 01일 (수)

제출일자: 2023년 11월 01일 (수)

학 과: 컴퓨터정보공학부

담당교수: 이혁준 교수님

실습분반: 수요일 0, 1, 2

학 번: 2022202075

성 명: 우나륜

1. 제목 및 목적

A. 제목

Synchronous FIFO

B. 목적

선입 선출 구조인 Synchronous FIFO의 구조와 작동 방식을 이해하여 Finite State Machine을 설계하고 이를 Verilog로 구현함으로써 병렬 데이터 처리와 데이터 구조에 대해 이해하고 학습하는 데 목적을 둔다.

2. 원리(배경지식)

1. FIFO (First-In-First-Out, 선입선출)

   FIFO란 선입선출 구조로 data를 정리하고 이용하는 방식을 의미한다. 이 선입선출 구조는 시간을 기준으로 가장 먼저 들어온 자료가 가장 먼저 제거되는 특징을 가진다. 따라서 순서대로 데이터를 처리하는 경우에 유용하게 사용된다.

   Synchronous FIFO는 clock, reset_n, write_enable, read_enable, write_data, read_data, full, empty를 요소 가지며 clock의 rising edge에서 결과값이 나타나게 된다. Reset_n은 active-low 상태로 0일 때 동작하지 않고, 1일 때 동작하게 된다.

   FIFO에서 write는 값을 입력하고 read는 값을 가져온다. 각각 자료구조에서 push-pop 또는 enqueue-dequeue로 표현하는 것과 같다. 따라서 write_enable은 데이터를 입력 또는 저장하는 동작을 활성화하고 read_enable은 데이터를 출력 또는 제거하는 동작을 활성화시킨다. 다음으로, write_data는 write_enable에서 저장할 데이터의 값을 가지고 있으며, read_data는 읽은 데이터의 값을 가지고 있다.

   FIFO 내부 용량의 상태가 비었는지 또는 꽉 찼는지 나타내기 위해 empty와 full의 status를 가진다. Empty는 FIFO 내부에 데이터가 하나도 존재하지 않는 상태로, 저장되어 있는 데이터가 존재하지 않아 출력할 수 없는 경우를 알려주기 위한 상태이다. Full은 FIFO 내부에 데이터가 모두 차지된 상태로, 공간이 모두 차지되어 데이터를 더 이상 저장할 수 없는 경우를 알려주기 위한 상태이다.

   이러한 FIFO의 구조적 특성은 여러 분야에서 유용하게 사용될 수 있다. 통신 시스템에서 데이터 패킷 처리, 데이터 버퍼링 및 스케줄링에 사용될 수 있다. 데이터를 송수신할 때 패킷의 도착 순서를 유지하고, 데이터 전송의 시간적 불일치를 조절하는 데 사용된다.

2. Verilog에서 문자열 나타내기

   Verilog에서 문자 또는 문자열을 저장하기 위해서 비트수가 큰 bus를 선언해서 사용할 수 있다. 문자열은 ASCII 코드로 나타내어지기 때문에 7비트 당 하나의 문자를

표현할 수 있다. 본 과제에서 문자열을 나타내기 위해 범위를 [8*xx:0] (xx는 정수)로 나타내어 사용하였다. 그러면 xx만큼의 문자를 나타낼 수 있다. -

3. 설계 세부사항

1. State encoding table

State	Encoding	Description
INIT	000	Initial state
WRITE	001	Enqueue data successfully
WR_ERR	010	Enqueue request is rejected because the queue is full
NO_OP	011	No operation
READ	100	Dequeue data successfully
RD_ERR	101	Dequeue request is rejected because the queue is empty

위의 table은 FIFO의 상태를 나타낼 6가지 상태를 binary encoding을 사용하여 나타낸 것이다. INIT은 reset 이후의 초기상태로 000이다. WRITE는 FIFO 내에 데이터를 작성할 수 있는 상태로 001이다. WR_ERR는 FIFO 내의 용량이 꽉 차 더 이상 데이터를 작성할 수 없는 상태로 010이다. NO_OP는 아무런 연산을 수행하지 않는 상태로 011이다. READ는 FIFO 내에 저장된 첫 번째 데이터를 읽을 수 있는 상태로 100이다. RD_ERR는 FIFO가 비어있어 데이터를 읽어올 수 없는 상태로 101이다.

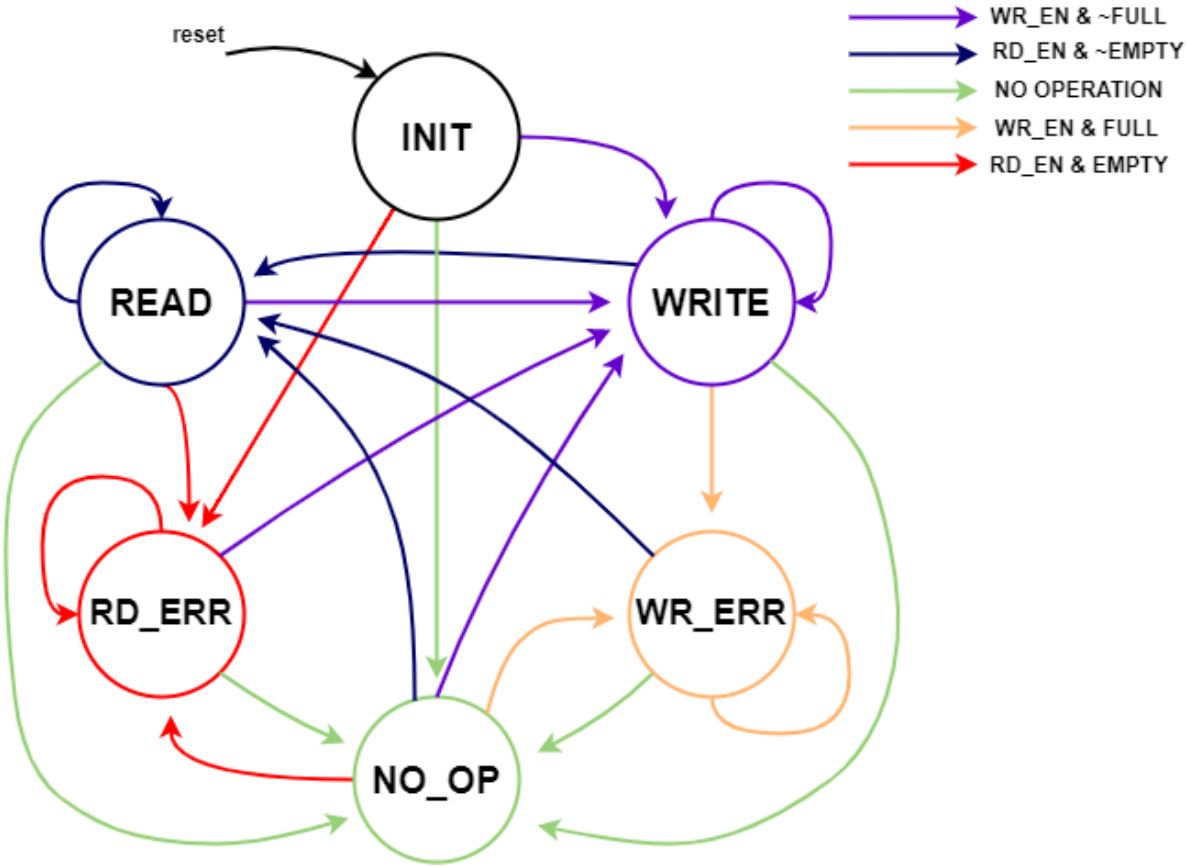
2. Functional description

Inputs			Internal register	Outputs		Operation or state
WR_EN	RD_EN	CLK	DATA_COUNT	FULL	EMPTY	
1	0	↑	< MAX_SIZE	0	0	Enqueue
0	1	↑	0 <	0	0	Dequeue
1/0	1/0	↑	-	No change		Invalid (not allowed)
X	X	X	= MAX_SIZE	1	0	Full
X	X	X	= 0	0	1	Empty

위의 표는 WR_EN, RD_EN의 값과 clk의 rising edge에 따른 기능을 표로 정리한 것이다. WR_EN = 1, RD_EN = 0, clock의 rising edge에서 DATE_COUNT가 MAX_SIZE보다 작으면 FULL, EMPTY 상태는 모두 0이며 이때 enqueue 기능을 수행한다. WR_EN = 0, RD_EN = 1, clock의 rising edge에서 DATE_COUNT가 0보다 크면 FULL, EMPTY 상태는 모두 0이며 이때 dequeue 기능을 수행한다. Clock의 rising edge에서 WR_EN = 0, RE_EN = 0 또는

WR_EN = 1, RD_EN = 1일 때, 아무런 기능을 수행하지 않는다. 마지막으로 WR_EN과 RD_EN, clock에 상관없이 DATA_COUNT가 MAX_SIZE면 FULL 상태를 가지고, 0이면 EMPTY 상태를 가지게 된다.

3. Finite state Diagram



위의 그림은 reset과 RD_EN, WR_EN, FULL, EMPTY 상태에 따른 상태 천이도를 그린 것이다.

4. Module configuration

구분	이름	설명
Top module	fifo	FIFO의 top module
Sub module	fifo_ns	Next state module
Sub module	fifo_cal	Calculate address module
Sub module	fifo_out	Output logic module
Sub module	Register_file	Register file module
Sub module	_dff3_r	3-bit D flip-flop module for state, head, tail
Sub module	_dff4_r	4-bit D flip-flop module for data_count
Sub module	_dff32_r	32-bit D flip-flop module for d_out

Sub module	mx2_32bits	32-bit 2-to-1 multiplexer module to select 0 or d_out depending on read_enable

위의 표는 top module인 fifo과 fifo에서 instance로 선언하여 사용한 sub module의 이름과 기능을 나열한 것이다. Sub module로 fifo_ns, fifo_cal, fifo_out, register_file, _dff3_r, _dff_4_r, _dff32_r, 그리고 mx2_32bits가 사용된 것을 확인할 수 있다.

5. I/O configuration

Module 이름	구분	이름	비트 수	설명
fifo	Input	clk	1-bit	Clock
		reset_n		Active-low에서 동작하는 reset 신호로 값이 인가되면 register의 값을 0으로 초기화
		rd_en		Read enable
		wr_en		Write enable
		d_in	32-bit	Data in
	output	d_out		Data out
		full	1-bit	Data full signal
		empty		Data empty signal
		wr_ack		Write acknowledge
		wr_err		Write error
		rd_ack		Read acknowledge
		rd_err		Read error
		data_count	4-bit	Data count vector
fifo_ns	input	wr_en	1-bit	Write enable
		rd_en		Read enable
		state	3-bit	Current state
		data_count	4-bit	Data count vector
	output	next_state	3-bit	Next state
fifo_cal	input	state	3-bit	Current state
		head		Current head pointer
		tail		Current tail pointer
		data_count	4-bit	Current data count vector
	output	we	1-bit	Write enable
		re		Read enable
		next_head	3-bit	Next head pointer
		next_tail		Next tail pointer

		next_dadta_count	4-bit	Next data count vector
fifo_out	input	state	3-bit	Current state
		data_count	4-bit	Data count vector
	output	full	1-bit	Data full signal
		empty		Data empty signal
		wr_ack		Write acknowledge
		wr_err		Write error
		rd_ack		Read acknowledge
		rd_err		Read error
_dff3_r / _dff4_r / _dff32_r	input	clk	1-bit	Clock
		reset_n		Active-low reset
		d	3-bit / 4-bit / 32-bit	Input data
	output	q		Output data
mx2_32bits	Input	d0	32-bit	First data
		d1		Second data
		s	1-bit	Selection signal
	output	y	32-bit	Output

위의 표는 top module fifo의 I/O와 내부에서 선언한 instance들의 I/O를 정리하고 그에 대한 설명을 간략히 작성한 것이다.

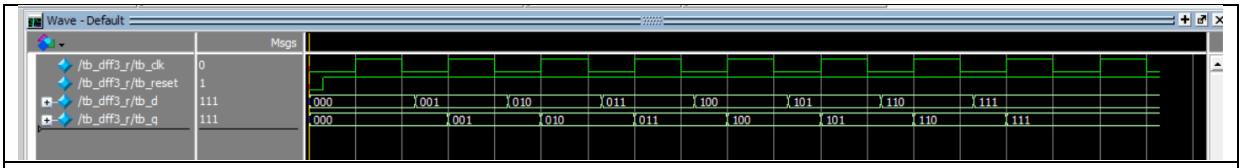
6. Outputs of each state

State	WR_ACK	WR_ERR	RD_ACK	RD_ERR	DATA_COUNT	DOUT
INIT	0	0	0	0	0	0
WRITE	1	0	0	0	++	Previous value
WR_ERR	0	1	0	0	No change	Previous value
NO_OP	0	0	0	0	No change	Previous value
READ	0	0	1	0	--	Mem[head]
RD_ERR	0	0	0	1	No change	Previous value

위의 표는 각 상태에 따른 output값인 wr_ack, wr_err, rd_ack, rd_err, data_count, 그리고 d_out의 상태를 나타내는 표이다. INIT 상태일 때, 모든 output들은 0으로 초기화된다. WRITE 상태일 때, WR_ACK는 1의 값을 같고 DATA_COUNT의 값이 +1된다. WR_ERR 상태일 때, WR_ERR는 1의 값을 가진다. NO_OP 상태일 때, 아무런 변화도 일어나지 않는다. READ 상태일 때, RD_ACK는 1의 값을 가지고 DATA_COUNT의 값이 -1되고 DOUT은 head pointer가 가리키는 데이터 값을 가진다. RD_ERR 상태일 때, RD_ERR의 값은 1이 된다.

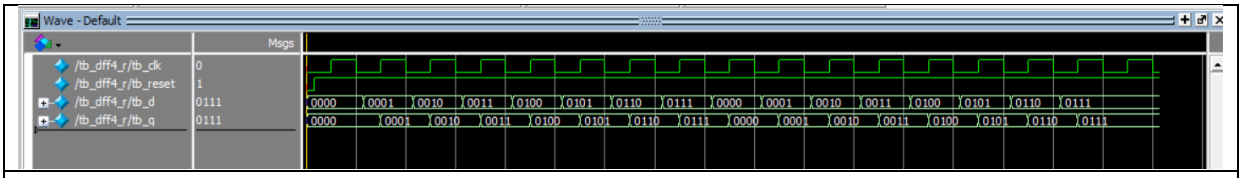
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



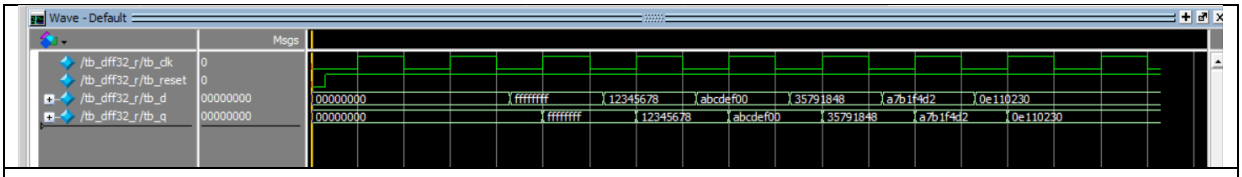
_dff3_r testbench waveform

3-bit에서 입력될 수 있는 모든 경우의 수를 입력하고 그에 따른 결과를 보여주는 waveform이다. 위의 그림에서 확인할 수 있듯이, reset = 1일 때 동작하며 clock의 rising edge에서 q의 값이 업데이트 되는 것을 확인할 수 있다.



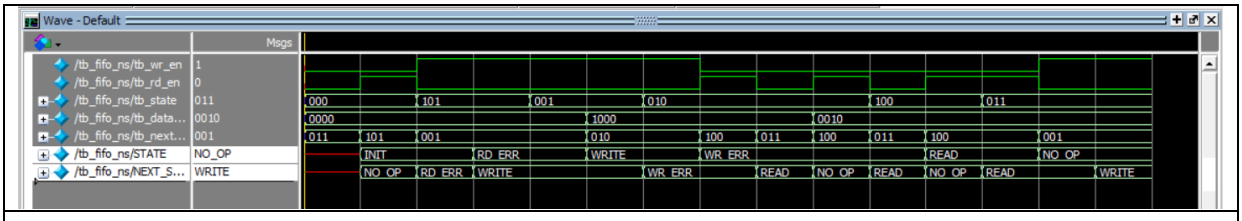
_dff4_r testbench waveform

4-bit에서 입력될 수 있는 모든 경우의 수를 입력하고 그에 따른 결과를 보여주는 waveform이다. 위의 그림에서 확인할 수 있듯이, reset = 1일 때 동작하며 clock의 rising edge에서 q의 값이 업데이트 되는 것을 확인할 수 있다.



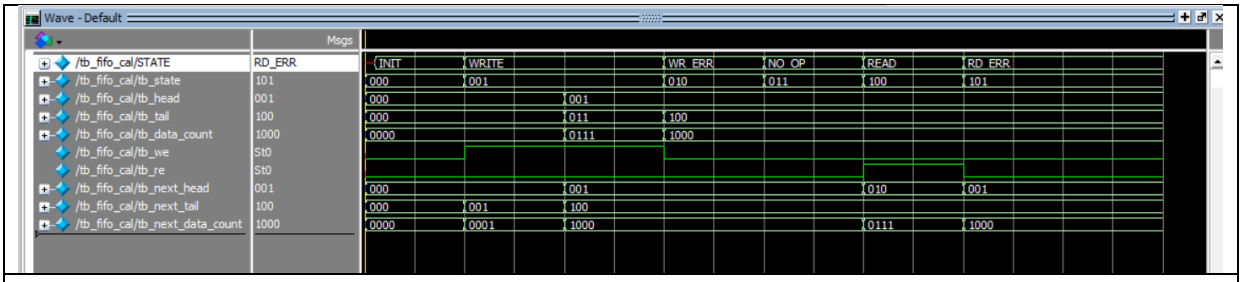
_dff32_r testbench waveform

32-bit에서 입력될 수 있는 몇 가지의 경우의 수를 입력하고 그에 따른 결과를 보여주는 waveform이다. 위의 그림에서 확인할 수 있듯이, reset = 1일 때 동작하며 clock의 rising edge에서 q의 값이 업데이트 되는 것을 확인할 수 있다.



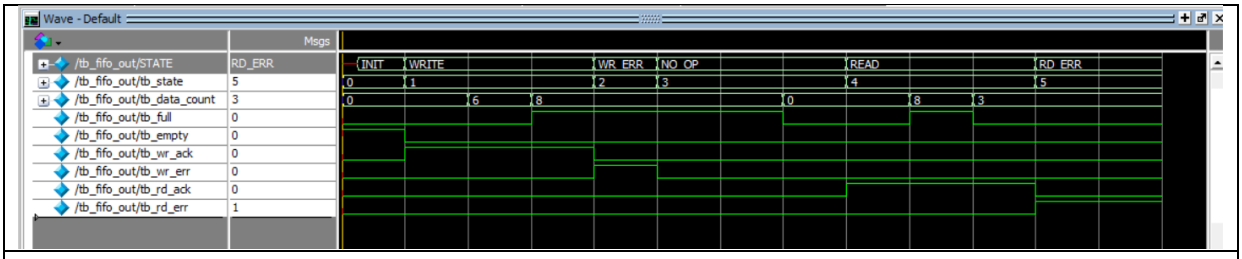
fifo_ns.v testbench waveform

FIFO의 next_state를 계산하는 module의 waveform이다. 위의 그림에서 확인할 수 있듯이, 입력값인 현재 상태와 wr_en, rd_en, data_cout의 상태에 따라 next_state가 달라지는 것을 확인할 수 있다. 맨 아래 2줄은 3-bit state와 next_state를 알아보기 쉽게 string으로 표현한 것이다.



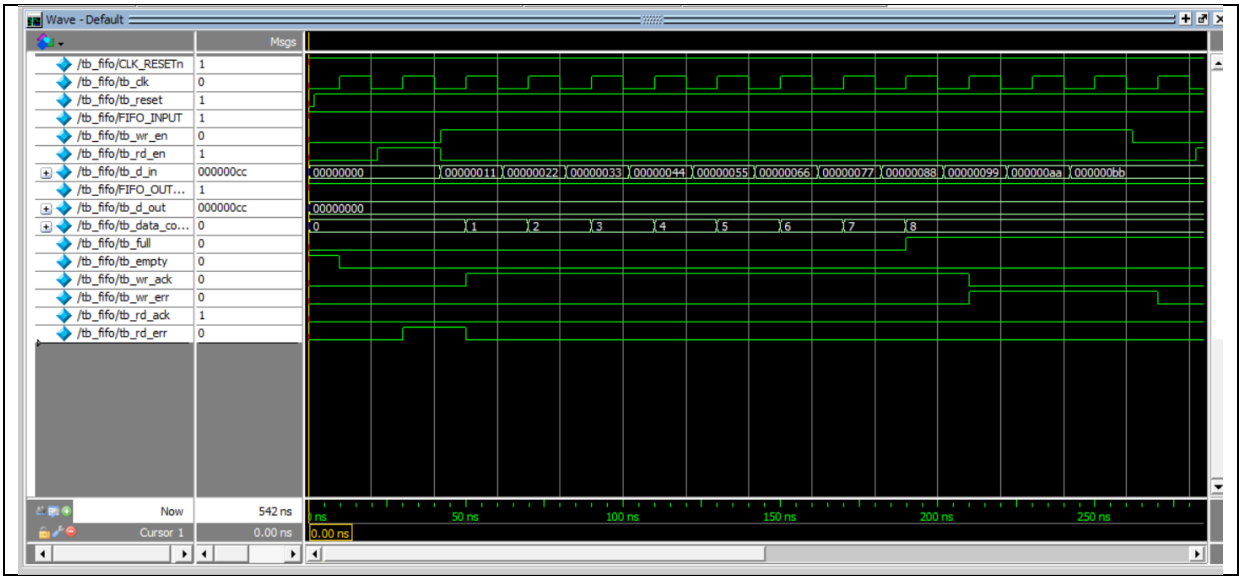
fifo_cal.v testbench waveform

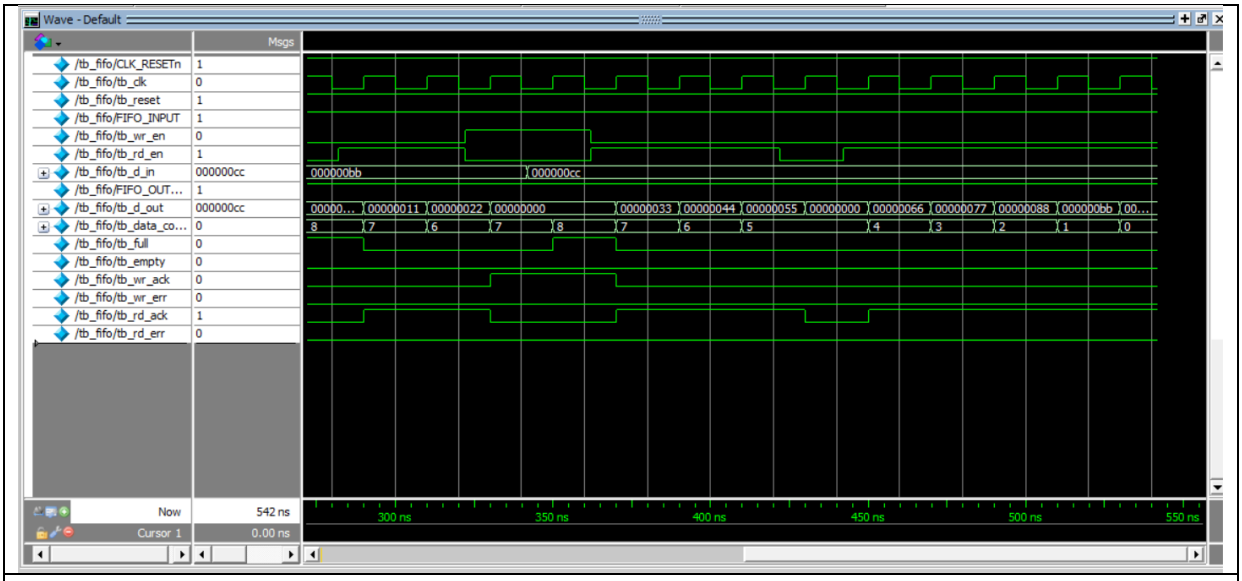
FIFO의 flag값인 we(write enable), re(read enable), next_head, next_tail, next_data_count를 계산하는 module의 waveform이다. 위의 그림에서 확인할 수 있듯이, 입력값인 현재 상태와 head, tail, data_count에 따라 입력값이 달라지는 것을 확인할 수 있다. 맨 윗줄은 3-bit state를 알아보기 쉽게 string으로 표현한 것이다.



fifo_out.v testbench waveform

FIFO의 outputs를 계산하는 module의 waveform이다. 위의 그림에서 확인할 수 있듯이, 입력값인 현재 상태와 data_cout의 상태에 따라 status flag인 full, empty와 handshake signal인 wr_ack, wr_err, rd_ack, rd_err의 값이 바뀌는 것을 확인할 수 있다.





fifo.v testbench waveform

FIFO module의 waveform이다. 위의 그림에서 확인할 수 있듯이, reset_n = 0일때는 output들은 0의 값을 가지고 reset_n = 1일 때는 clock의 rising edge에 따라 값이 변화하는 것을 확인할 수 있다. 다음으로, 입력값인 wr_en, rd_en에 따라 data_count의 값이 변화하고 d_in의 값이 제대로 저장되어 d_out으로 값이 똑같이 출력되는 것을 확인할 수 있다. 또한, 입력되고 출력된 데이터에 따라 data_count의 값이 바뀌는 것을 확인할 수 있다. FIFO가 full인 상태에서 wr_en의 값이 1이면 wr_err의 값이 1이 되며, FIFO가 empty인 상태에서 rd_en의 값이 1이면 rd_err의 값이 1이 된다. 그 외에는 오류가 없는 상태인 wr_ack 또는 rd_ack가 1이되는 것이 된다.

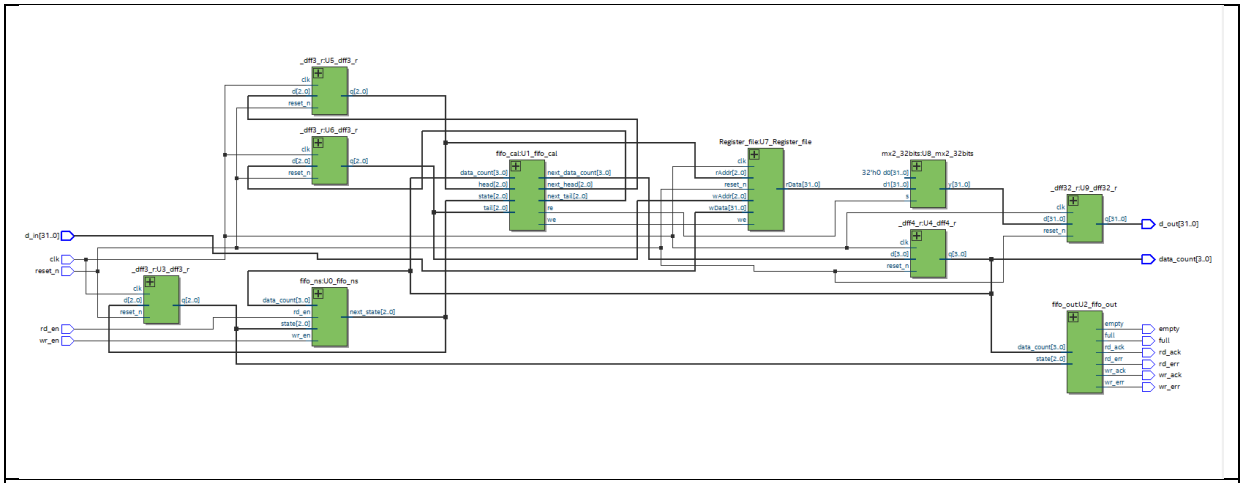
B. 합성(synthesis) 결과

Flow Summary

<<Filter>>

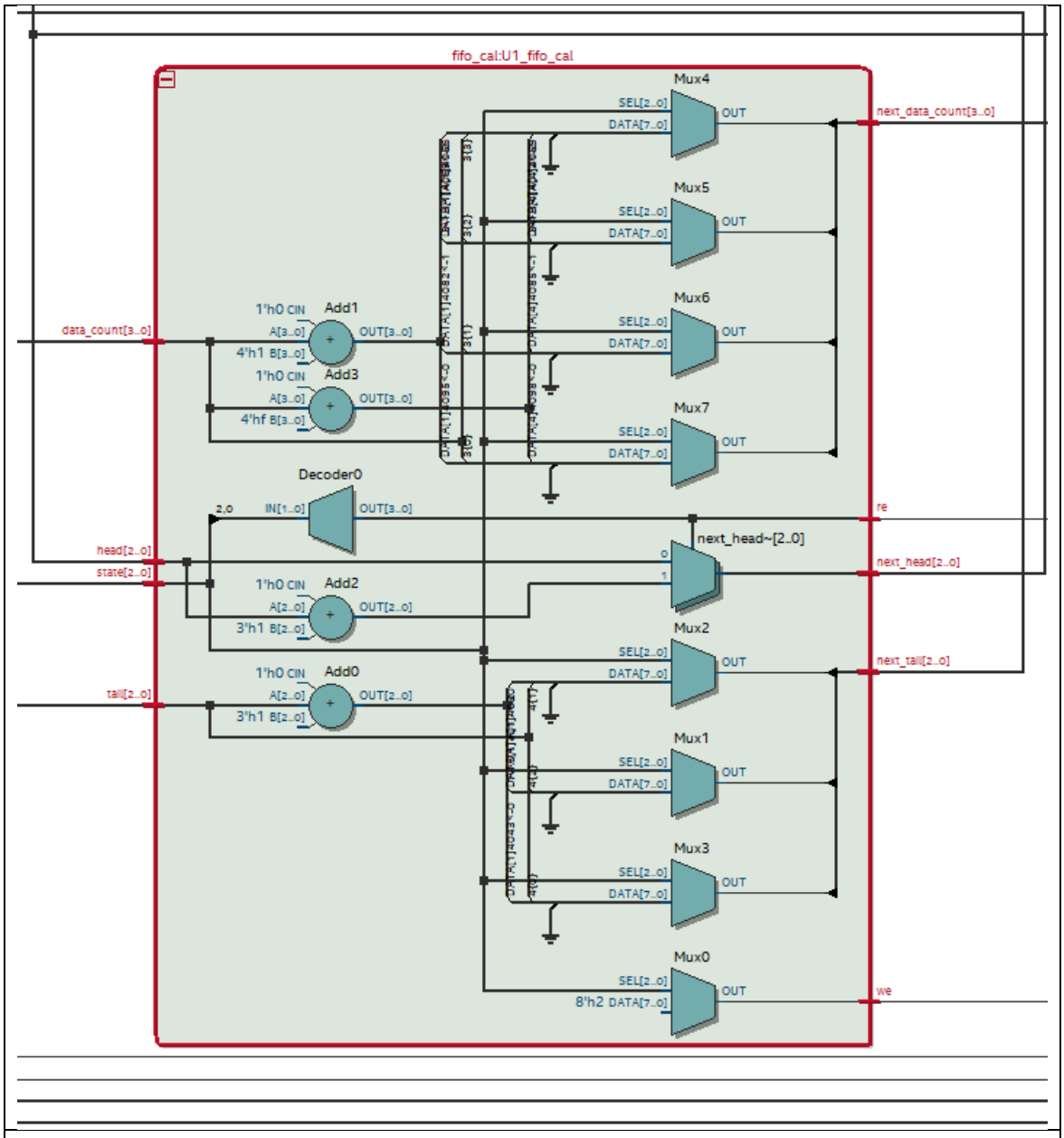
Flow Status	Successful - Wed Nov 01 09:17:24 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	fifo
Top-level Entity Name	fifo
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	155 / 41,910 (< 1 %)
Total registers	301
Total pins	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 %)

위의 그림은 fifo.v의 compilation의 flow summary이다. Logic utilization은 155로 1%미만이고, register는 총 301개 사용되었으며, pins는 78개로 16%정도 사용되었다.



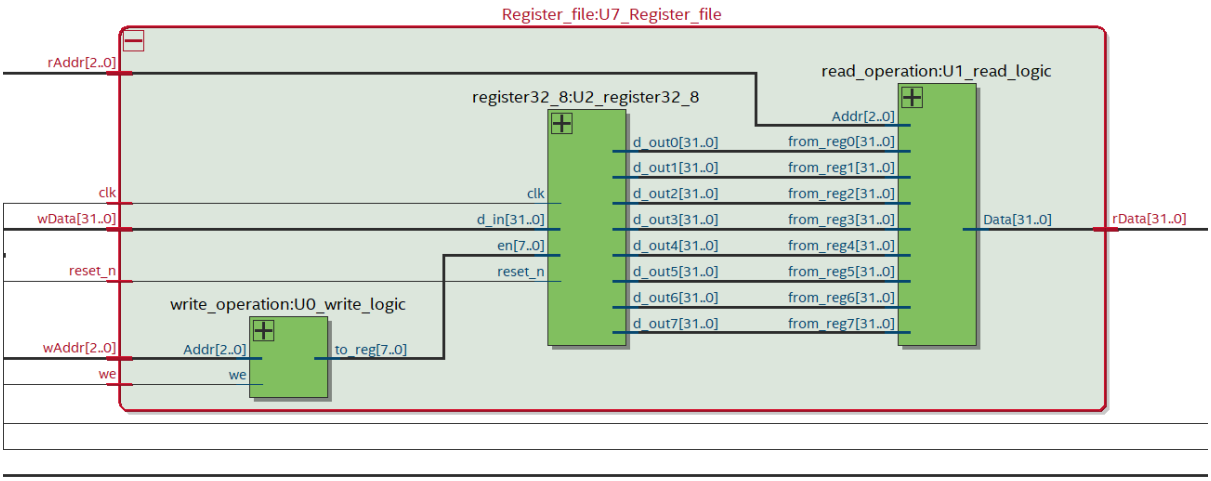
fifo.v RTL viewer

위의 그림은 fifo.v의 전체 RTL viewer 모습이다. 1개의 fifo_ns, fifo_out, fifo_cal, mx2_32bits, register_file, _dff4_r, _dff32_r과 3개의 _dff3_r이 사용된 것을 확인할 수 있다.



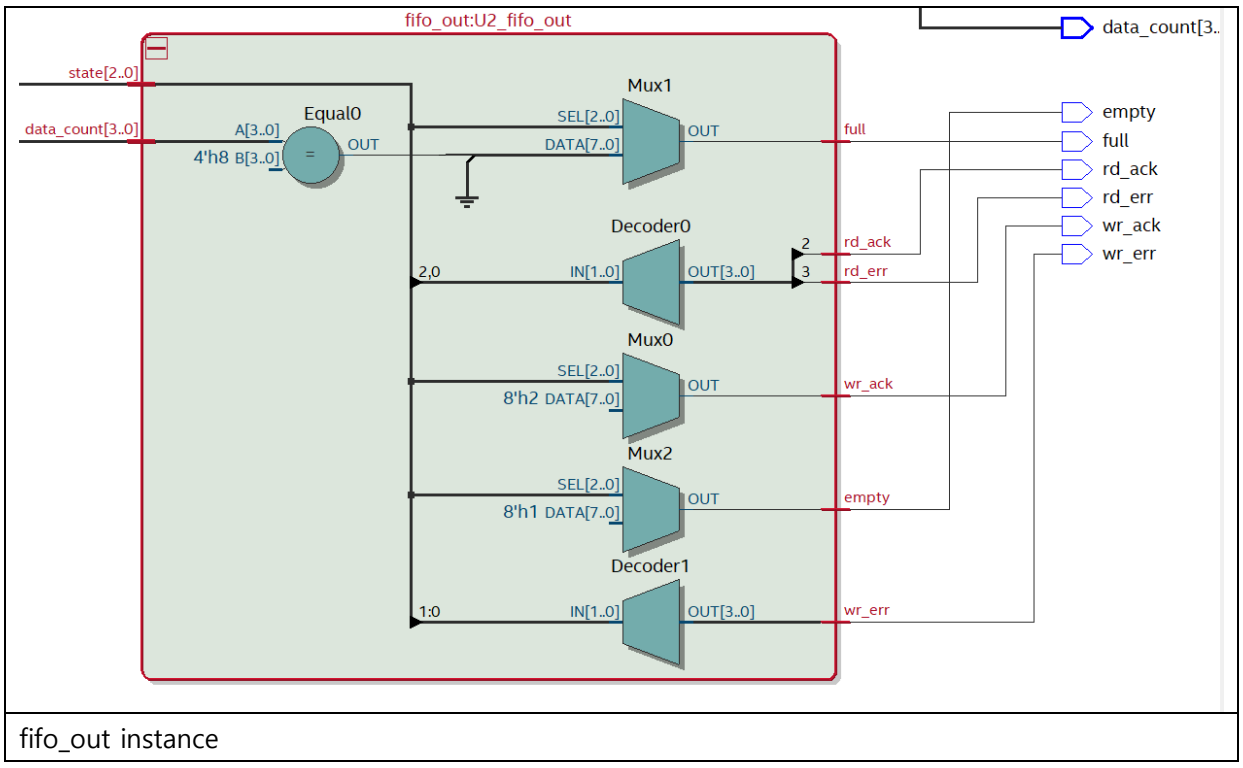
fifo_cal instance

위의 그림은 fifo_cal instance의 내부 회로도이다. Input으로 state, head, tail, data_count를 입력받고 we, re, next_head, next_tail, next_data_count를 출력한다. 각 상태에 따른 output를 계산하는 것을 case문으로 구현하였는데, case문이 mux로 표현된 것을 확인할 수 있다. Add1과 Add3은 4-bit 덧셈 연산을 수행하며 이는 next_data_count를 계산하는 데 사용된다. Add2와 Add4는 3-bit 덧셈 연산을 수행하며, 이는 next_head와 next_tail을 계산하는데 사용된다.



Register_file instance

위의 그림은 register_file instance의 내부 회로이다. 이전의 과제에서 구현하였던 write_operation, register32_8, read_operation instance가 사용된 것을 확인할 수 있다.



fifo_out instance

위의 그림은 fifo_out instance의 내부 회로이다. Input으로 현재 state와 data_count를 입력받고, output으로 full, empty status flag와 handshake signal인 wr_ack, wr_err, rd_ack, rd_err이 출력되는 것을 확인할 수 있다. 각 상태에 따른 output를 계산하는 것을 case문으로 구현하였는데, case문이 mux로 표현된 것을 확인할 수 있다.

5. 고찰 및 결론

A. 고찰

1. 출력값 d_out이 reset이 0일 때 값을 가지는 경우가 발생하였다. Reset은 active-low 상태에서 동작하기 때문에 reset의 값이 0일 때는 d_out이 0의 값을 가져야 한다. 이는 reset에 따른 d_out의 값을 변화시키는 flip-flop이 누락되었기 때문이다. 따라서 top module인 fifo module의 맨 마지막에 32-bit resettable D flip-flop instance를 추가하니 해결되었다.
2. RTL simulation에서 design load error가 발생하였다. 이는 top module인 fifo.v에서 register file의 instance를 선언할 때, instance 이름을 빠트렸기 때문이다.
3. 결과값인 d_out에서 fans out 오류가 발생하였다. 이는 d_out이 여러 instance에서 output으로 입력받았기 때문이다. 따라서 임시변수 wire를 선언하여 사용하니 해결할 수 있었다.

B. 결론

1. Waveform에서 문자열을 표현하기 위해서는 radix(진수)를 ASCII로 바꾸면 waveform 화면에 문자열로 나타낼 수 있었다.
2. 본 과제에서는 32-bit data를 처리하고 capacity가 8인 FIFO를 구현하였다. 이후에 이를 확장하면 더 많은 비트 수를 처리하고 더 많은 데이터를 저장할 수 있는 FIFO를 구현할 수 있을 것이다. FIFO를 구현하면서 선입선출에 관한 구조를 확실히 이해할 수 있었으며, 내부 logic을 구현하는 과정이 매우 보람차고 재미있었다.

6. 참고문헌

[1] 이준환 / FIFO / 광운대학교 컴퓨터정보공학부 / 2023년

[2] 이혁준 / 컴퓨터공학기초실험2 Lab#8 FIFO / 광운대학교 컴퓨터정보공학부 / 2023년

[3] waveform에 문자열 나타내기 / https://zrr.kr/pFa9

[4] FIFO/ https://velog.io/@gju06051/Verilog%EC%8B%AC%ED%99%943-FIFOSingle-Clock