컴퓨터 공학 기초 실험2 보고서

실험제목: FIFO

실험일자: 2019년 10월 01일 (화)

제출일자: 2019년 10월 14일 (월)

학 과: 컴퓨터정보공학과

담당교수: 이준환 교수님

실습분반: 화요일 0, 1, 2

학 번: 2015722031

성 명: 박 태 성

1. 제목 및 목적

A. 제목

FIFO

B. 목적

이전 실습보다 적어진 description을 토대로 FIFO를 구현하면서 FSM 설계능력을 높인다. 설계와 검증 과정을 통해 register file의 주소에 data를 읽고 쓰는 과정에 대한 이해를 높이고 waveform에서 오류를 찾아 수정하는 능력 향상에 목적을 둔다.

2. 원리(배경지식)

A. Stack

Stack은 후입 선출 구조를 갖는 자료 구조이다. 이를 Last In First Out(LIFO)로 부르기도 한다. 자료를 넣는 것을 push, 자료는 빼는 것을 pop이라 한다. 가장 먼저 push한 데이터는 가장 아래 쌓이고 마지막에 넣은 데이터가 가장 위에 쌓인다. 따라서, 가장 늦게 push된 자료가 가장 빨리 pop 된다. Stack 구조는 메모리 할당과 접근에 사용된다.

B. Queue

Queue는 선입 선출 구조를 갖는 자료구조이다. 이를 First In First Out(FIFO)로 부르기도 한다. 자료를 넣는 것을 push, 자료는 빼는 것을 pop이라 한다. 가장 먼저 push 된 데이터를 시작으로 순서대로 쌓인다. 데이터를 빼낼 때는 가장 먼저 push된 데이터가 가장 먼저 pop된다. 파이프 모양의 통로에 자료를 넣는 것으로 많이 설명된다. 예로 마트 계산대에 가장 먼저 온 사람이 계산을 하고 나가는 것이 있다. 컴퓨터에는 프로세스 관리, 입력된 데이터 처리에 queue 구조가 사용된다.

3. 설계 세부사항

A. State Encoding

State	Encoding		
INIT	0	0	0
NO_OP	0	0	1
WRITE	0	1	0
WR_ERROR	0	1	1
READ	1	0	0
RD_ERROR	1	0	1

B. Input/Output configuration

i. FIFO

Module	Port	Name	Bandwidth	Description
fifo	Input	clk	1-bit	Clock
		reset_n	1-bit	Active low에 동작
		rd_en	1-bit	Read Enable
		wr_en	1-bit	Write Enable
		d_in	32-bits	Input Data
	Output	d_out	32-bits	Output Data
		full	1-bit	Data full signal
		empty	1-bit	Data empty signal
		wr_ack	1-bit	Write acknowledge
		wr_err	1-bit	Write error
		rd_ack	1-bit	Read acknowledge
		rd_err	1-bit	Read error
		data_count	4-bits	Data count vector

ii. FIFO_ns

Module	Port	Name	Bandwidth	Description
fifo_ns	Input	wr_en	1-bit	Write Enable
		rd_en	1-bit	Read Enable
		state	3-bits	Current state
		data_count	4-bits	Data count vector
	Output	next_state	3-bits	Next state

iii. FIFO_cal_addr

Module	Port	Name	Bandwidth	Description
fifo_cal_addr	Input	state	3-bits	Current state
		head	3-bits	Current head pointer
		tail	3-bits	Current tail pointer
		data_count	4-bits	Current data count vector
	Output	we	1-bit	Register file write enable
		re	1-bit	Register file read enable
		next_head	3-bits	Next head pointer
		next_tail	3-bits	Next tail pointer
		next_data_count	4-bits	Next data count vector

iv. FIFO_out

Module	Port	Name	Bandwidth	Description

fifo_out	Input	state	3-bits	Current state
		data_count	4-bits	Current data count vector
	Output	full	1-bit	Data full signal
		empty	1-bit	Data empty signal
		wr_ack	1-bit	Write acknowledge
		wr_err	1-bit	Write error
		rd_ack	1-bit	Read acknowledge
		rd_err	1-bit	Read error

C. Diagram

i. Next state

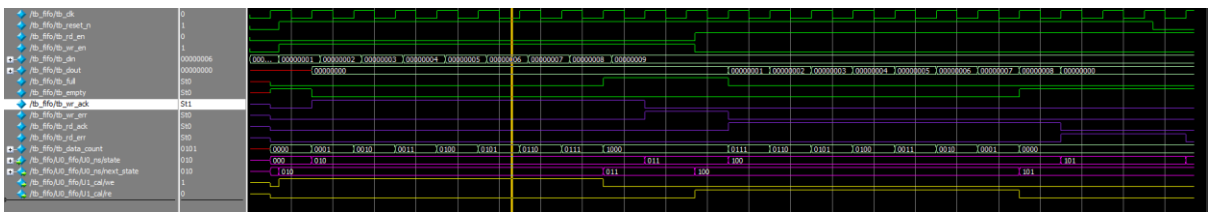
SINIT	RD_EN	WR_EN	DATA_COUNT	Next state
0	X	X	X	INIT
1	0	0	X	NO_OP
1	0	1	< MAX_SIZE	WRITE
1	1	0	> 0	READ
			0	RD_ERROR
1	1	1	No change	NO_OP

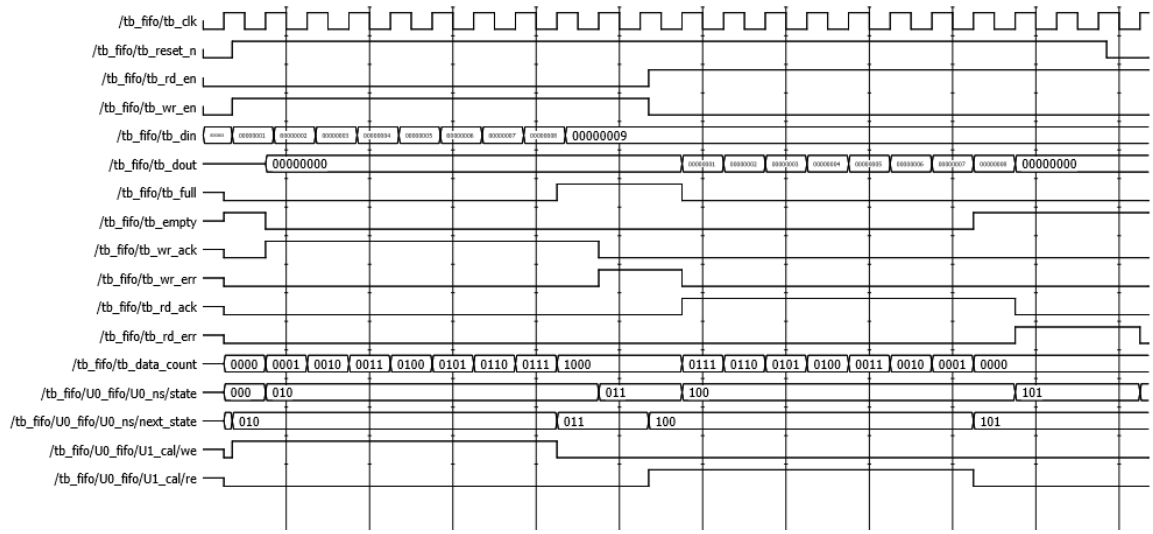
ii. Output

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

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과





편의를 위하여 같은 waveform을 각기 다른 형식으로 2개 첨부하였다.

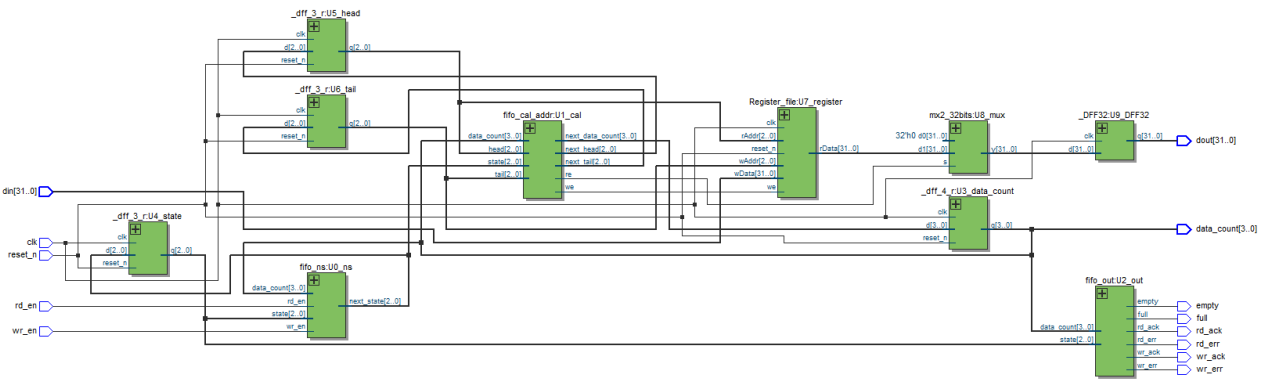
Reset_n = 0일 때, 출력이 정상적으로 나오는지 확인하기 위하여 reset_n = 0을 초기값으로 시작하였다.

8개의 data를 연속적으로 write하고 read하여 정상 작동하는지 확인하였다. 따라서, write enable signal과 read enable signal을 상황에 맞게 바꾸어 주고 그에 따라 출력되는 data count, handshake signal을 관찰하였다.

Data count = 8이 된 후에는 state이 WR_ERR로 바뀌어 더 이상 write가 되지 않고 data count도 증가하지 않음을 확인하였다. 반면, data count = 0이 된 후에는 state이 RD_ERR-로 바뀌어 더 이상 read가 되지 않고 data count가 감소하지 않음을 확인하였다. 추가적으로 2-to-1 mux에 selection 신호로 0값이 전달되어 8'h00000000이 출력되는 것을 확인하였다.

B. 합성(synthesis) 결과

i. FIFO RTL Viewer



ii. FIFO Flow summary

Flow Summary	
Flow Status	Successful - Thu Oct 10 01:54:06 2019
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	fifo
Top-level Entity Name	fifo
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	193 / 41,910 (< 1 %)
Total registers	256
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 %)

회로를 구현하는데 총 논리 소자 193개와 256개의 register를 사용하였다. 칩에서 78개의 pin을 사용하였다.

5. 고찰 및 결론

A. 고찰

Waveform에서 State이 바뀌는 동시에 그에 따른 결과가 출력되어야 하는데 1 clock cycle 뒤에 출력되었다. 이는 calculate address logic에 input으로 next state이 들어가야하는데 state을 넣어주어 발생한 문제였다. 따라서 top module fifo에서 input을 수정하여 해결하였다.

Module 복잡도가 이전보다 올라가서 하위 module의 동작 눈에 보이지 않아 오류의 원인을 찾는데 어려움이 있었다. Waveform에서 하위 module을 추가하고 wave color 수정 후 이를 wave.do 파일로 저장하여 지속적으로 확인하면서 원인을 찾았다.

Testbench를 구현 할 때, instantiation하는 module 이름을 잘못 입력하여 waveform이 나오지 않았다. Compile 때는 오류가 발생하지 않아 몰랐다. 이는 model-sim transcript에서 오류 메시지를 확인할 수 있다.

B. 결론

하위 module을 작성하고 이를 top module로 연결하는 방법을 선호하였다. 그러나, 복잡도가 높아지면 top module을 먼저 작성하고, 구성하는 하위 module을 기능별로 구현하는 것이 더 나은 방법임을 알았다. 동기들도 이와 같은 방식을 구현하는 것을 선호하였다.

State가 바뀌면 1 clock cycle 뒤에 이에 따른 결과가 출력되는 것이 정상인 줄 잘못 알고 설계하였다. 그러나, state가 바뀌는 동시에 결과가 출력되는 것이 정상이다.

Head와 tail이 111_2 다음에 어떤 출력을 내보낼지 궁금하였다. 000_2를 출력한다. 실제로 원형 버퍼와 유사한 구조를 통해 head와 tail이 지속적으로 증가해도 오류가 나지 않게 설계하는 것을 알게 되었다.

동기를 통해 FIFO의 동작 원리가 자료 구조 queue나 운영체제 메모리 관리에 활용된다는 것을 새롭게 알게 되었다.

6. 참고문헌

이준환/ FIFO/ 광운대학교/ 2019