컴퓨터 공학 기초 실험2 보고서

실험제목: Register File

실험일자: 2019년 09월 24일 (화)

제출일자: 2019년 09월 30일 (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

Register File

B. 목적

32-bits register 8개를 이용하여 register file을 구현해보고 각 register에 데이터를 read, write하여 데이터가 register에 어떻게 저장되고 불러와지는지 이해하는 데 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. Register File

i. I/O configuration

Module	Port	Name	Bandwidth	Description
Register_file	Input	clk	1-bit	Clock
		reset_n	1-bit	Active low에 동작
		we	1-bit	Write enable
		wAddr	3-bits	Write address
		rAddr	3-bits	Read address
		wData	32-bits	Write Data
	Output	rData	32-bits	Read Data

Module	Port	Name	Bandwidth	Description

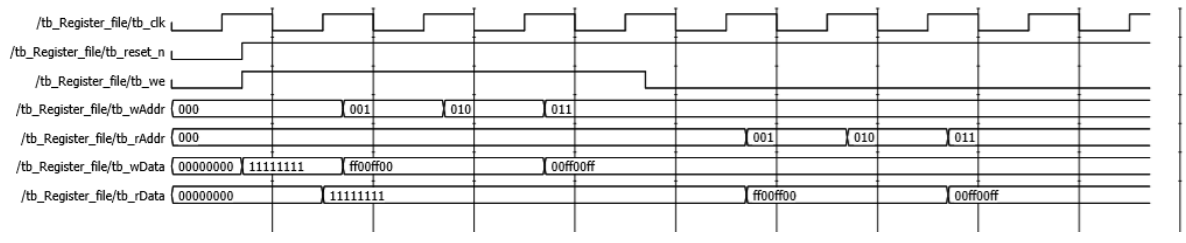
Register32_8	Input	clk	1-bit	Clock
		reset_n	1-bit	Active low에 동작
		en	1-bit	Register enable
		d_in	32-bits	Data in
	Output	d_out0~ d_out7	32-bits	Register data out
write_operation	Input	we	1-bit	Write enable
		Addr	3-bits	Write address
	Output	to_reg	8-bits	Selected register enable signal
read_operation	Input	from_reg0 ~from_reg7	32-bits	8 register' data
		Addr	3-bits	Read address
	Output	Data	32-bits	Data out

ii. Truth Table

Inputs					**Output**							
reset	we	I[2]	I[1]	I[0]	O[7]	O[6]	O[5]	O[4]	O[3]	O[2]	O[1]	O[0]
0	X	X	X	X	0	0	0	0	0	0	0	0
1	0	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	1	0	0	0	0	0	0	1	0
1	1	0	1	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	1	0	0	0	0
1	1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



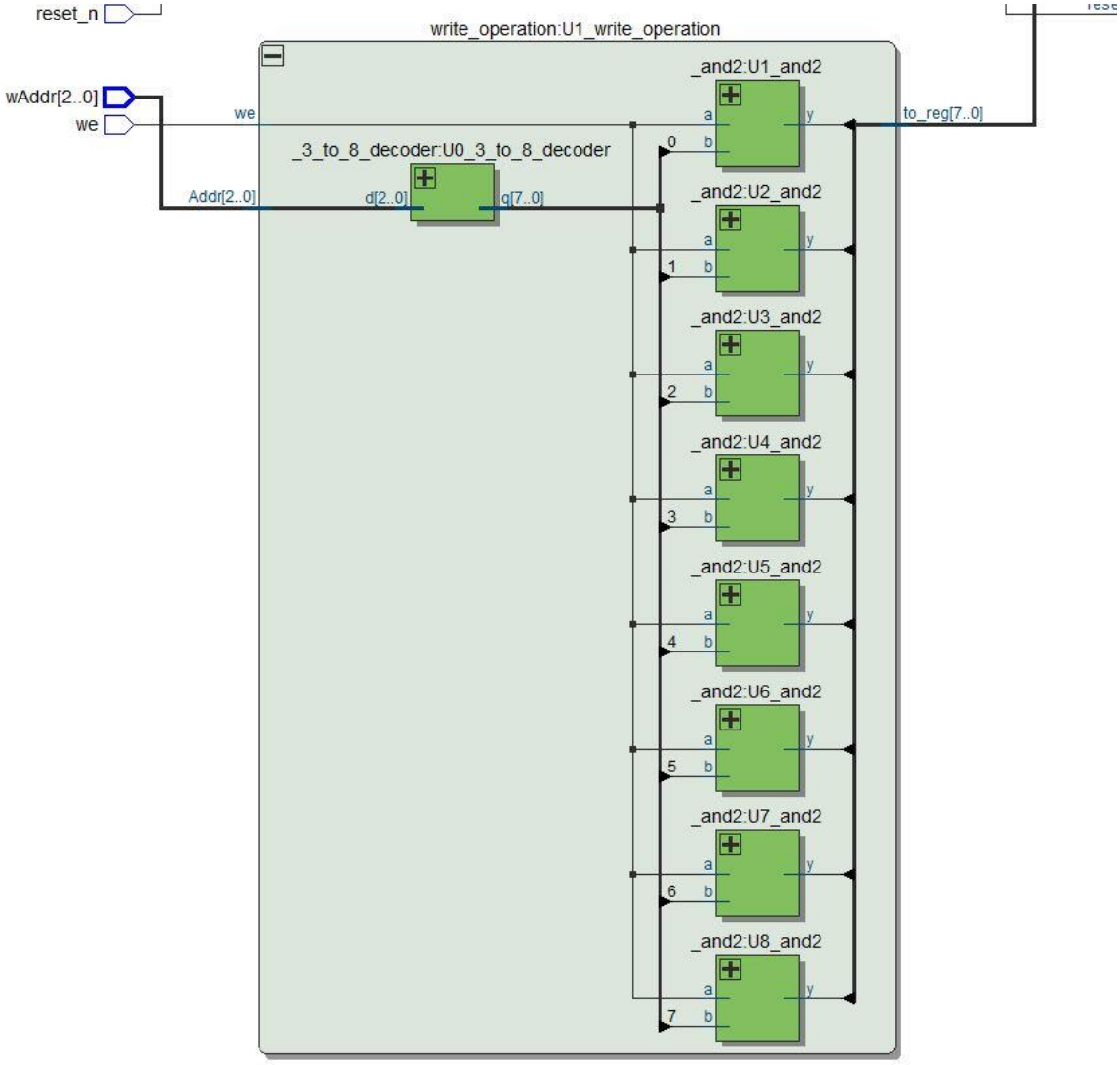
Write Enable이 1이면, wAddr에 해당하는 주소에 데이터를 write하는 것이 가능하다.

Waddr이 000이면, wData가 000에 해당하는 주소에 write 되었다가 clock의 rising edge 일 때, read 되어 rData로 출력된다. Waddr이 001이면 해당 주소에 wData가 저장되었다가 rAddr가 001이면 rData로 출력된다. Clock의 rising edge에 register에 값이 저장된다. Raddr값에 따라 즉시 해당하는 주소에서 rData가 출력된다.

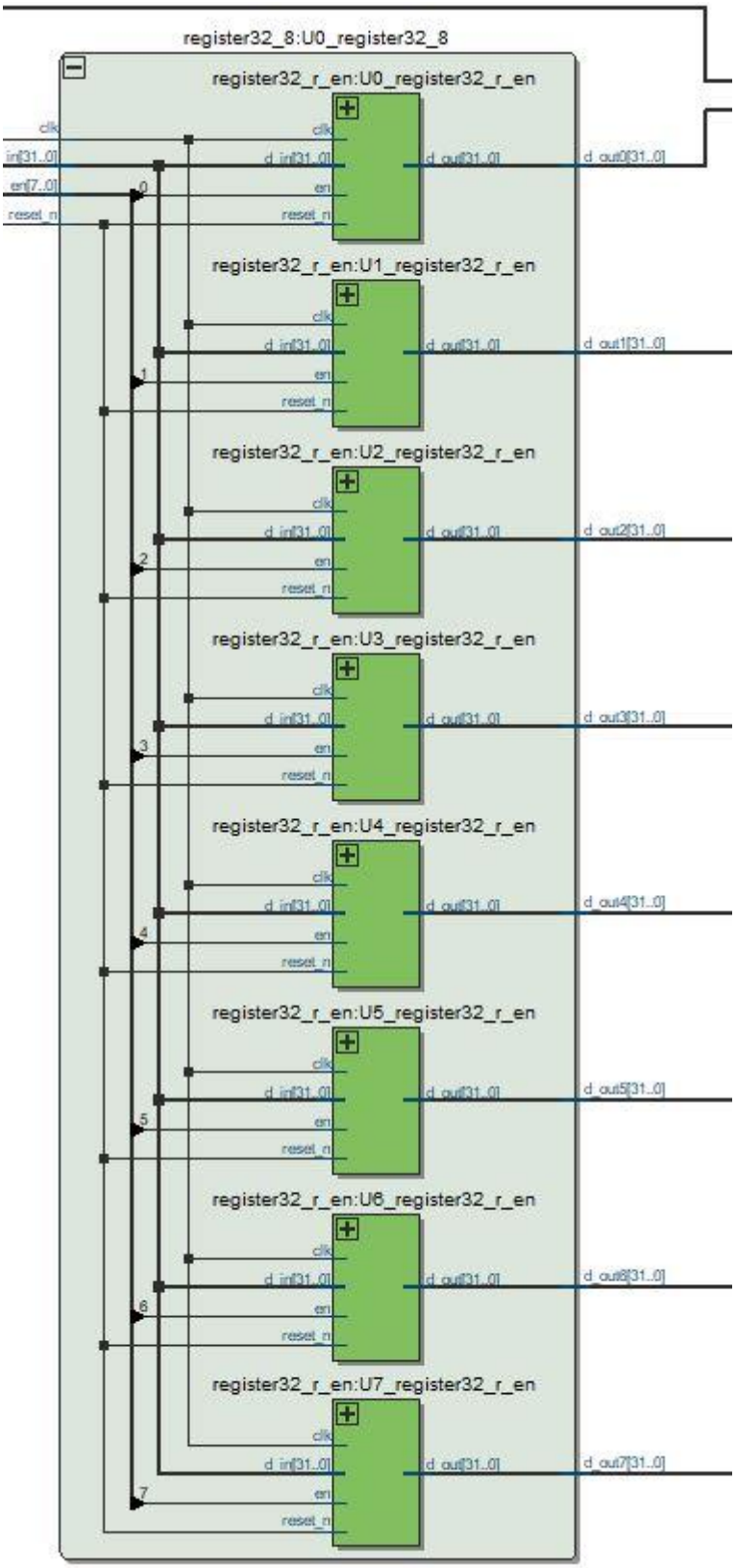
B. 합성(synthesis) 결과

i. RTL Viewer

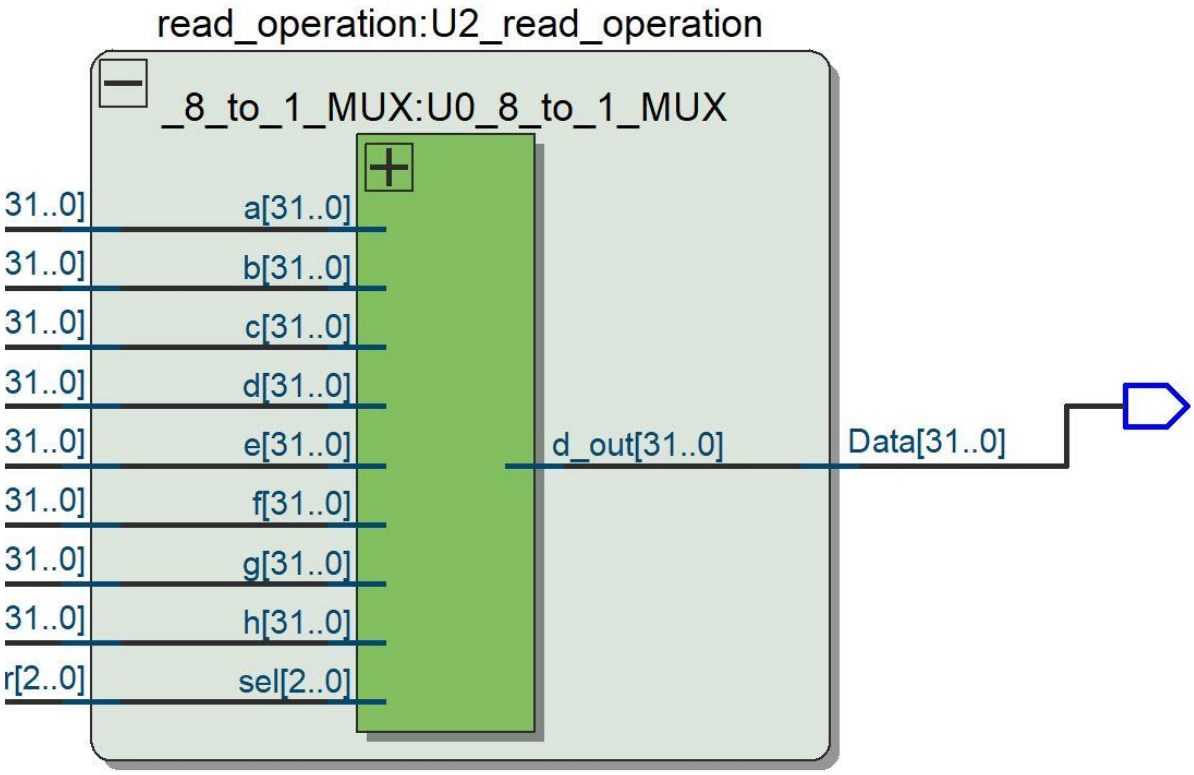
1. write_operation



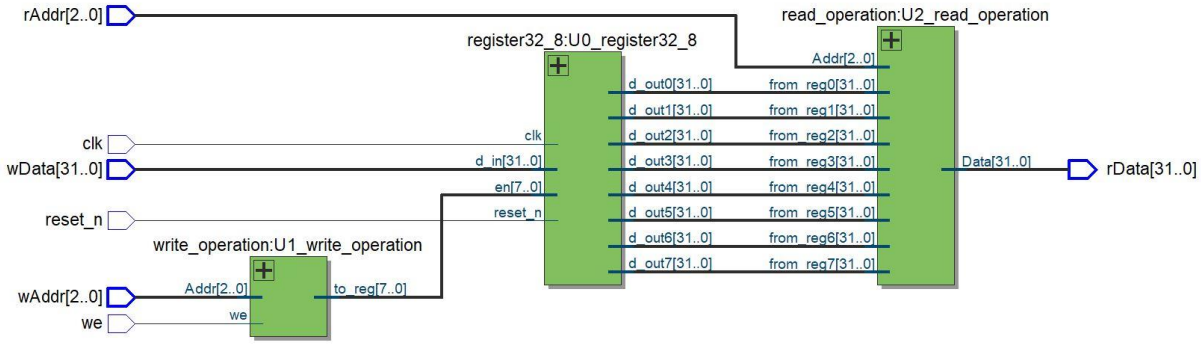
2. register32



3. read_operation



4. Register File



ii. Flow Summary

Flow Summary	
Flow Status	Successful - Sat Sep 28 15:35:42 2019
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	Register_file
Top-level Entity Name	Register_file
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	256
Total pins	73
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

5. 고찰 및 결론

A. 고찰

예상했던 waveform이 나오지 않아 model-sim에서 waveform에 하위 모듈을 추가하여 오류 발생 지점을 추적했다. Dot operator를 사용하지 않고 기억에 의존하여 port들을 연결하다가 실수를 했다. 에러를 찾는데 waveform은 유용한 tool이다. 편의성 문제로 Dot operator를 회사에서는 많이 쓰지 않는다고 들었다. 그러나, 이번 실습을 통해서 나는 dot operator를 적절히 사용하면 오류 발생률을 줄일 수 있을 것이라 생각한다.

B. 결론

Clock의 rising edge에는 register에 데이터가 저장된다. Output rData는 rAddr 값에 변화에 즉시 결과가 출력된다. CPU 내부의 register의 데이터 write과 read를 이해하였다. 데이터를 read하여 산술, 논리 연산 후에 다시 write하는 과정이 register file에서 이용된다.

6. 참고문헌

Register File / https://en.wikipedia.org/wiki/Register_file