컴퓨터 공학 기초 실험2 보고서

실험제목: Register file

실험일자: 2023년 10월 18일 (수)

제출일자: 2023년 10월 18일 (수)

학 과: 컴퓨터정보공학부

담당교수: 이혁준 교수님

실습분반: 수요일 0, 1, 2

학 번: 2022202075

성 명: 우나륜

1. 제목 및 목적

A. 제목

Register file

B. 목적

32bits register를 8개 가지는 register file을 설계하고 각 register에 address (주소)를 할당하여 선택된 주소의 register에 read, write 기능을 수행하는 동작을 구상할 수 있다. Verilog로 이를 구현하여 register 내에 read, write하는 기능에 대해 이해하고 이를 응용하는데 목적을 둔다.

2. 원리(배경지식)

1. Register file

Register file이란 CPU에 있는 프로세서 레지스터의 배열이다. 따라서 데이터를 읽는 register를 read register라 라고 데이터를 작성하는 register를 write register라고 한다. 데이터를 읽는 레지스터인 read register는 n-bit의 D flip-flop으로 구성된다. 그리고 MUX (multiplexer)는 N개의 레지스터 중 하나와 연결한다. 그러면 입력되는 레지스터 input의 32-bit 비트 수에 따라 출력 비트 수도 32-bit로 같아진다. Register data에서는 32-bit의 데이터를 register에 작성하는데, 이때 작성할 데이터의 주소값을 5-bit decoder를 사용하여 주소값을 지정하고 register에 값을 저장하며 write signal의 값이 1이되게 된다.

Register file은 여러 개의 register로 구성되어 각 register는 정해진 고정 크기의 데이터를 저장할 수 있다. 일반적으로 우리가 사용하는 32-bit architecture에서는 32개의 register를 가진다. Register는 r0, r1, r2 등으로 r뒤에 숫자를 붙여 명명한다. Register file은 processor 내부에서 작동하므로 메모리에 빠르게 접근할 수 있다는 장점이 있다. 따라서 원하는 데이터를 register에 읽고 작성하는 데 걸리는 시간이 최소화된다. 또한, 레지스터 파일은 주로 데이터의 값을 저장하고 중간 결과를 보관하는 데 사용하므로 processor가 명령어를 실행하는 동안 register에 데이터를 읽고 작성하면서 연산 및 제어 흐름을 알맞게 조절할 수 있다.

2. Stack

Stack이란 자료구조의 한 형태로, 가장 나중에 추가된 요소가 가장 먼저 제거되는 후입선출 (LIFO, Last-In-First-Out)의 원칙을 따른다. 스택은 주로 함수 호출 스택이나 역 추적 알고리즘, 임시 데이터 저장, 재귀 알고리즘, DFS 알고리즘 등에 사용된다. 스택은 간단하고 효율적인 구조로 데이터를 빠르게 추가하고 제거할 수 있다는 장점이 있다.

3. Queue

Queue란 자료구조의 한 형태로, 가장 먼저 추가된 요소가 가장 먼저 제거되는 선입선출 (FIFO, First-In-First-Out) 원칙을 따른다. 큐는 주로 작업 대기열, 프로세스 관리, 캐시 구현, BFS알고리즘 등에 사용된다. 큐는 데이터를 일정한 순서로 처리할 때 유용하다는 장점이 있지만, 큐의 크기가 제한되어 있을 경우 오버플로우 문제가 발생할 수 있다는 단점을 가진다.

이러한 Queue 구조는 여러가지 형태로 나타낼 수 있다. 먼저 가장 기본적인 구조인 선형 큐 (linear queue)는 삽입을 위해서 요소들을 계속 이동시켜야 한다. 맨 앞 노드를 가리키는 front와 맨 뒤를 가리키는 rear는 계속 증가만 하기 때문에 front 앞쪽에 공간이 있더라도 삽입할 수 없는 경우가 발생할 수 있다. 원형 큐 (Circular queue)는 front 앞의 공간이 남는 선형 큐의 문제점을 보완한 방식이다. 이때 공백 상태와 포화상태를 구분하기 위해 하나의 공간을 비워두게 된다.

3. 설계 세부사항

구분	이름	설명
Top module	Register_file	Register file의 top module
Sub module	register32_8	8개의 32bits register module
Sub module	write_operation	Write address decode module
Sub module	read_operation	Select register using read address

표. Module Configuration

위의 표는 Register_file module에서 사용할 주요 sun module을 나타내는 표이다. 레지스터의 값을 저장할 register32_8 module과 register에 값을 작성하고 작성한 데이터를 불러오는 기능을 수행하는 write_operation과 read_operation이 존재한다.

Module 이름	구분	이름	비트 수	설명
Register_file	Input	clk	1-bit	Clock
		reset_n		Active-low에 동작하는 reset 신호
		we		Write enable
		wAddr	3-bit	Write address
		rAddr		Read address
		wData	32-bit	Write data
	Output	rData		Read data
register32_8	input	clk	1-bit	Clock
	Input	reset_n		Active-low에 동작하는 reset 신호
		en		Register enable signal

		d_in	32-bits	Data in
	Output	d_out0 ~ d_out7		Register data out
write_operation	Input	we	1-bit	Write enable
		Addr	2-bit	Write address
	Output	wEn	8-bit	Selected register enable signal
read_operation	Input	from_reg0 ~from_reg7	32-bit	8 registers' data
	Output	Addr	3-bit	Read address
		Data	32-bit	Data out

표. I/O configuration

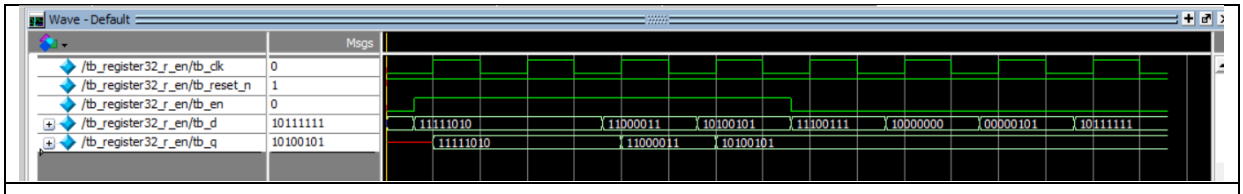
Register_file module은 register32_8, write_operation, read_operation module들을 instance화하여 사용한다. 8개의 32-bit의 값을 저장하는 모듈인 register32_8 module은 resettable enabled D Flip-Flop을 instance화하여 register module을 생성하고, 이 module를 instance화 하여 총 8개를 사용한다.

Write operation 기능을 수행하는 write_operation module은 3-to-8 decoder module을 생성한 뒤, decoder module과 AND gate 8개를 사용하여 완성하였다.

Read operation 기능을 수행하는 read_operation module은 8-to-1 MUX module을 생성한 뒤 Addr를 사용하여 register의 8개 output들 중에서 하나를 선택하여 저장된 데이터를 읽는다.

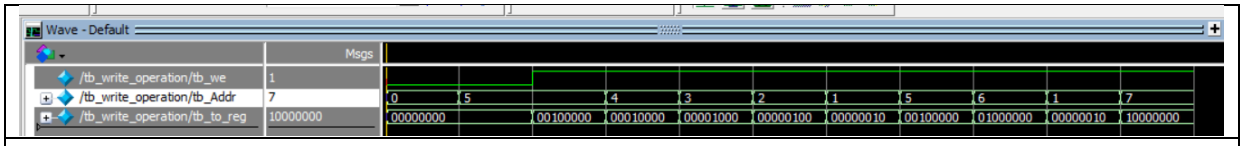
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



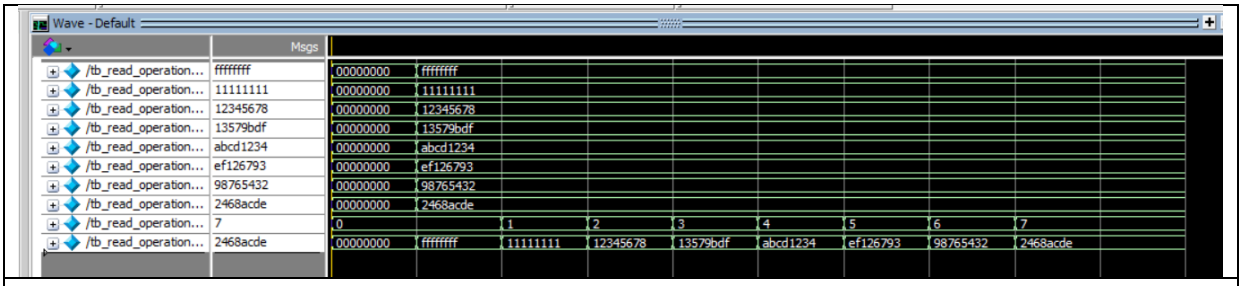
Register32_r_en testbench waveform

Clock의 rising edge에서 tb_d의 값이 tb_q에 저장되는 것을 확인할 수 있다. 만약 en의 값이 0이되면 tb_q의 값이 clock의 rising edge에서 변화하지 않는다.



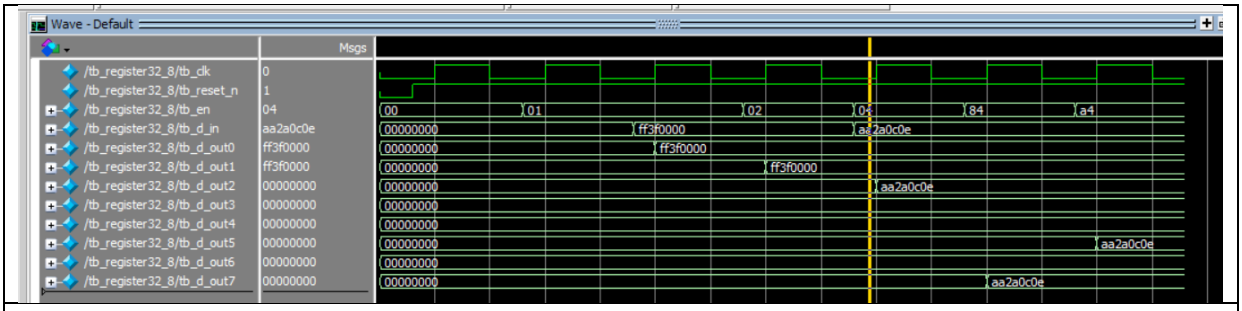
Write_operation testbench waveform

we의 값이 1일 때 Addr의 값에 따라 해당 주소에 저장된 값이 출력되는 것을 확인할 수 있다.



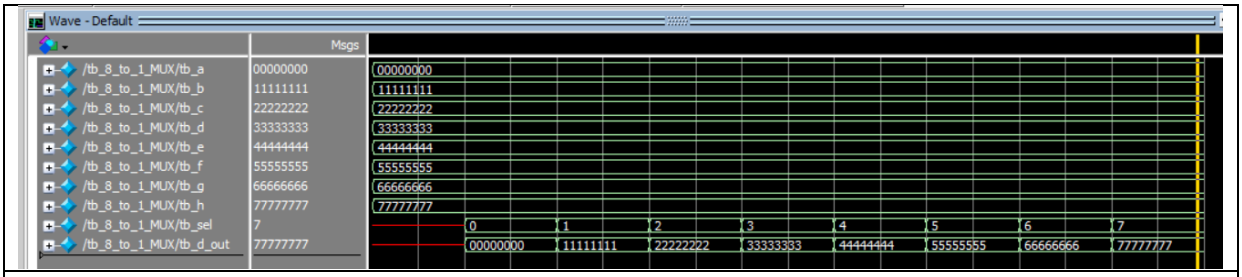
Read_operation testbench waveform

Addr의 값에 따라 그에 해당하는 주소에 저장된 값이 결과값에 저장된다. 위의 그림에서 확인할 수 있듯이, 각 주소값마다 다른 데이터가 저장되어 있으며, 모든 주소값 내의 값을 출력하여 데이터 read가 제대로 수행되는 지 확인하였다.



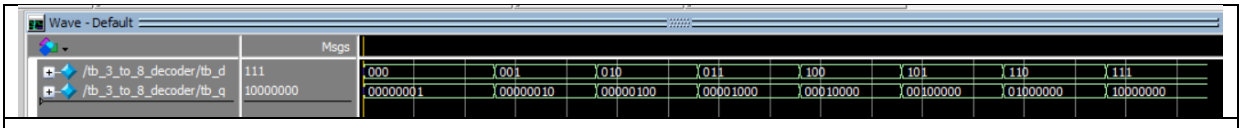
Register32_8 testbench waveform

Register32_8에서 en의 값이 0이 아닐 때, d_out의 값에 입력한 d_in의 값이 제대로 입력된 것을 확인할 수 있다.



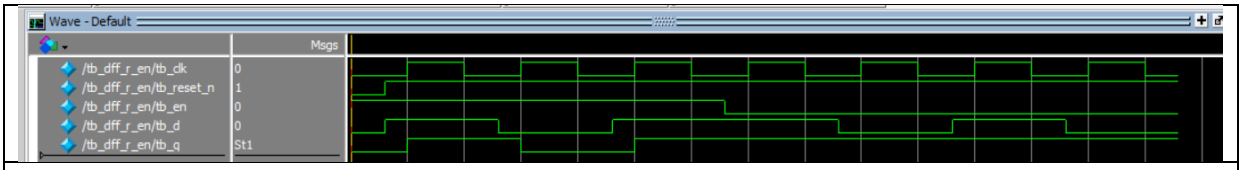
_8_to_1_MUX testbench waveform

저장된 값 tb_a ~ tb_h에 각기 다른 값들이 저장되어 있으며 tb_sel을 통해 모든 값을 하나씩 출력하였다.



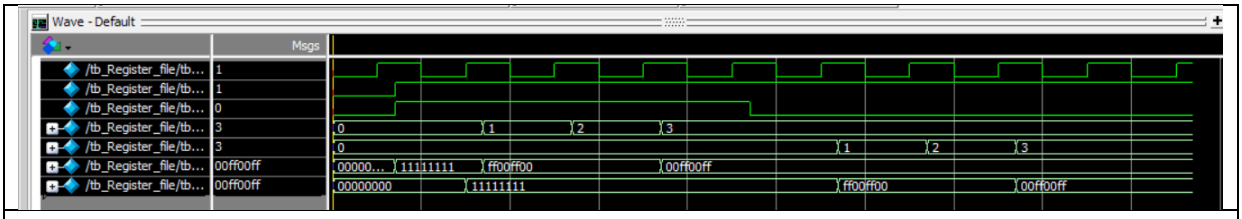
_3_to_8_decoder testbench waveform

3개의 입력비트 tb_d를 입력받아 각 입력 비트에 따라 8개의 출력비트들 중 하나를 활성화하여 출력한다.



_dff_r_en testbench waveform

Clock의 rising egde에서 tb_q에 값이 하당되며 en의 값이 0이 되면 이전의 q의 값을 계속 유지한다.



tb_Register_file testbench waveform

Register file은 clock의 rising egde에서 값이 변화되며 en의 값이 1일 때 wData가 저장되고 rData는 en의 값에 상관없이 rAddr에 따라 rData의 값이 변한다.

B. 합성(synthesis) 결과

Flow Summary

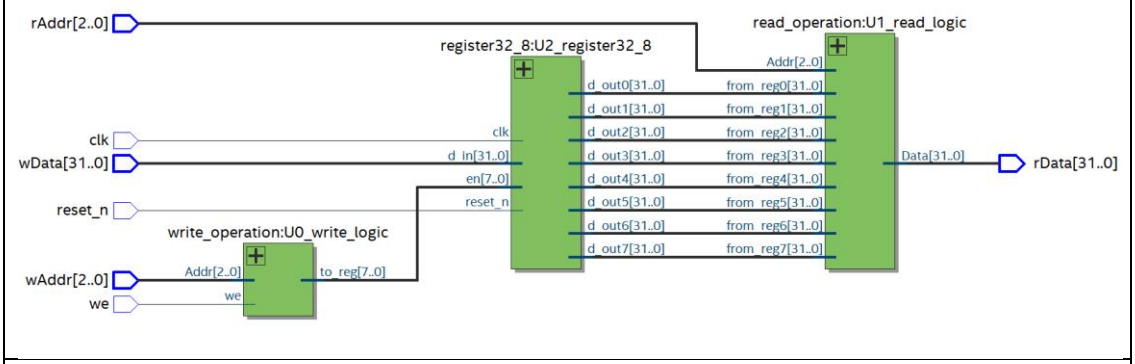
<<Filter>>

Flow Status	Successful - Wed Oct 18 10:18:37 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 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)	123 / 41,910 (< 1 %)
Total registers	256
Total pins	73 / 499 (15 %)
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 %)

Register_file flow summary

Register_file에서 logic utilization은 123로 1%미만이며 총 256개의 register가 사용되었

고 pins는 73으로 15%정도 사용되었다.

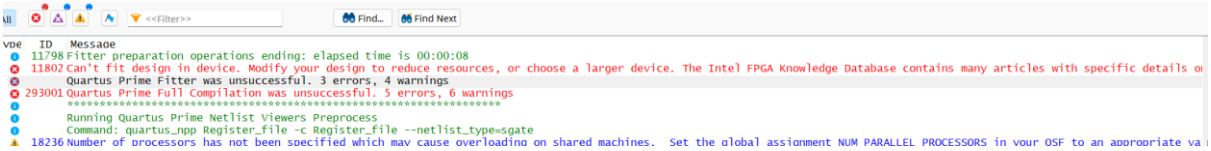


Register_file RTL viewer

Register_file의 RTL viewer를 통해 write_operation, register32_8, read_operation instance가 사용된 것을 알 수 있다,

5. 고찰 및 결론

A. 고찰



Register32_8과 read_operation의 compilation에서 compile design error가 발생하였다. 이는 pin 수가 너무 많아 발생한 오류였는데, 해결하지 못하였다.

B. 결론

Register file을 Verilog로 구현하면서 register file의 동작 방식을 이해할 수 있었다. 각 module에 대한 testbench를 작성하면서 각 module에 대한 동작 방식에 대해 확실히 이해할 수 있었고 이를 다른 module에 적용하여 구현하는 데 익숙해졌다.

6. 참고문헌

[1] 이혁준 / 컴퓨터공학기초실험2 Lab#7-2 Register file / 광운대학교 / 2023년

[2] Register file / https://en.wikipedia.org/wiki/Register_file

[3] Register file vs SRAM / http://babyworm.net/archives/312

[4] 스택과 큐 / https://devuna.tistory.com/22

[5] 자료구조 / https://wooono.tistory.com/395

[6] https://velog.io/@nnnyeong/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0-%EC%8A%A4%ED%83%9D-Stack-%ED%81%90-Queue-%EB%8D%B1-Deque