컴퓨터 공학 기초 실험2 보고서

실험제목: Register File

실험일자: 2023년 10월 16일 (월)

제출일자: 2023년 10월 24일 (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

Register File

B. 목적

register file은 write operation, register sets, read operation으로 구성됩니다. 이번 실습에서는 지난 실습 때 설계한 32bits register를 여러 개 instance 하고 각 register에 address를 할당하여 read & write 을 수행하는 register file 을 설계합니다.

2. 원리(배경지식)

1) D flip-flop

D flip-flop은 clock의 rising edge에서 output Q 값이 input D 값으로 변경되게 됩니다. clock의 rising edge를 제외한 나머지 상황에서는 output Q 값이 input D 값으로 변경되지 않고, 이전 Q값을 계속해서 유지하게 됩니다.

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

2) resettable D flip flop

D flip flop에 reset 기능이 추가된 D flip flop입니다. resettable D flip flop에서 reset은 active low 에서 동작합니다. reset이 0이면 input D 값과, clock의 rising edge와 관계없이 output Q 값을 0으로 초기화시킵니다. reset이 1이면 평범한 D flip flop처럼 작동합니다. clock의 rising edge에서 output Q 값 input D 값으로 변경되고, 그 외의 상황에서는 output Q가 이전 Q 값을 유지하게 됩니다.

Input (rising edge)			Output
R	D	CLK	Q
0	X	X	0
1	0	rising	0
	1		1
	x	The others	이전 Q

3) Stack vs Queue

Stack은 먼저 입력된 값이 나중에, 나중에 입력된 값이 먼저 출력되는 Last In First Out(LIFO) 자료구조입니다. 스택 맨위에 있는 데이터를 반환하는 top, 스택에 데이터를 삽입하는 push, 스택에서 데이터를 삭제하는 pop, 스택이 비어있는지를 판별하는 isEmpty, 스택이 가득 찼는지를 판별하는 isFull 연산 등이 있습니다.

Queue는 먼저 입력된 값이 먼저, 나중에 입력된 값이 나중에 출력되는 First In First Out(FIFO) 자료구조입니다. 큐에 데이터를 삽입하는 enQueue, 큐에서 데이터를 빼내는 deQueue, 큐가 비어있는지를 판별하는 isEmpty, 큐가 가득 찼는지를 판별하는 isFull 연산 등이 있습니다. 하지만 큐는 데이터를 빼내는 deQueue 연산을 하고 나면, 데이터가 존재하는 공간이 비게 되는데, 이 공간을 재사용하기가 불가능하기 때문에 deQueue 연산을 진행할수록 큐의 가용범위가 줄어든다는 문제점이 있습니다. 그렇기에 원형 큐를 사용합니다.

따라서 Stack과 Queue는 값을 출력하는 순서가 반대인 자료구조라고 할 수 있습니다.

4) 설계 세부사항

1) write_operation

and 연산에 사용되는 we는 모두 동일한 값이므로 결국 we가 1이면 decoder의 값이 그대로 register32_8에 입력되게 되고, we가 0이면 모든 and 연산이 0이 되어 register32_8에 입력되게 됩니다. 따라서 and 연산 대신 mux를 사용하여 we가 1이면 decoder의 값을, we가 0이면 0을 register32_8에 입력할 수 있도록 삼항연산자를 활용하여 output wEn에 w_a와 0 중 한 가지를 assign 할 수 있게 설계했습니다.

2) register32_r_en

resettable, enabled register는 input 값들의 연산 우선순위가 resettable > enabled > clk 순이므로 clk과 reset_n의 edge를 감지하면 제일 먼저 reset_n이 0인지를 확인하게 됩니다. reset_n이 0이면 q는 0을 출력하게 됩니다. reset_n이 1이라면, 다음으로 en의 값을 확인하게 됩니다. en이 1이라면, clk의 rising edge에서 output q가 input d 값을 따라가게 되고, en이 0이라면, q가 이전 q 값을 계속해서 유지하게 됩니다.

3) read_operation

4) _3_to_8_decoder

always 문을 통해 input 값들의 변화를 감지하고, case 문을 통해 sel 값에 대응하는 input 값을 output으로 내보냅니다.

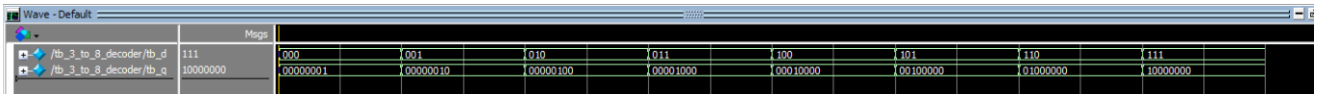
5) _8_to_1_MUX

always 문을 통해 input d 값의 변화를 감지하고, case 문을 통해 d 값에 따라 output q에 적절한 값을 대입하도록 설계했습니다.

5) 설계 검증 및 실험 결과

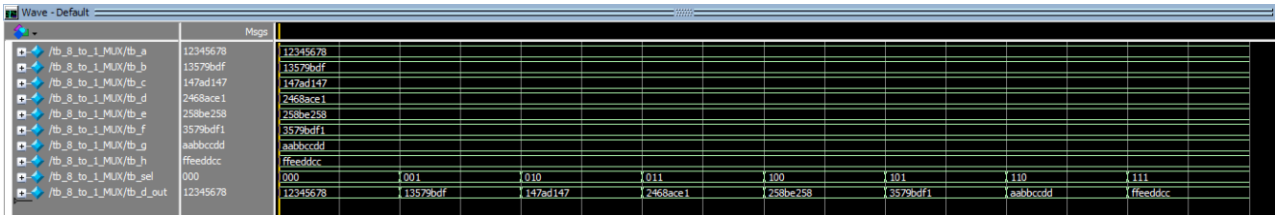
A. 시뮬레이션 결과

1) _3_to_8_decoder



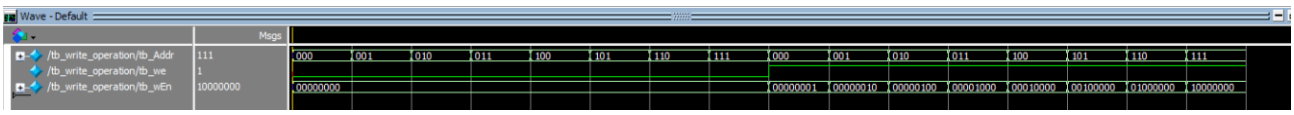
tb_3_to_8_decoder의 waveform입니다. 3bits input의 결과 8bits output이 출력되는 one hot decoder인 것을 확인할 수 있습니다.

2) _8_to_1_MUX



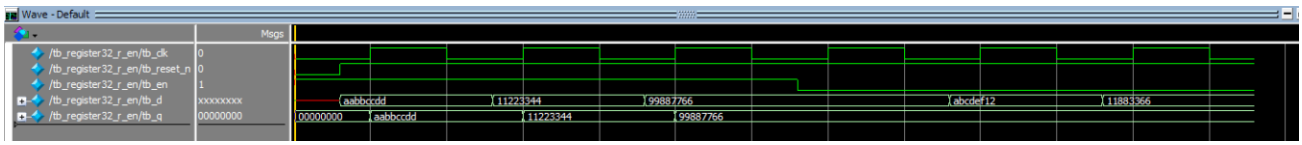
tb_8_to_1_MUX의 waveform입니다. 3bits sel input에 따라 tb_a ~ tb_h 중 한 가지 값을 tb_d_out으로 출력합니다. sel이 000~111 까지 증가함에 따라 tb_a ~ tb_h 가 일대일 대응합니다.

3) write_operation



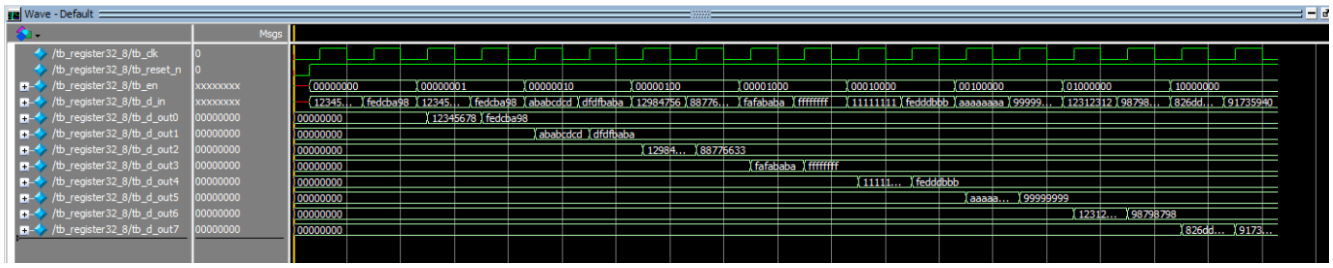
tb_write_operation의 waveform입니다. tb_we가 0이면 write를 진행하지 말라는 뜻이기 때문에 tb_wEn의 모든 bit를 0으로 출력합니다. tb_we가 1이면 tb_Addr의 값에 대응되는 tb_wEn의 비트만을 1로, 나머지는 0으로 출력합니다.

4) register32_r_en



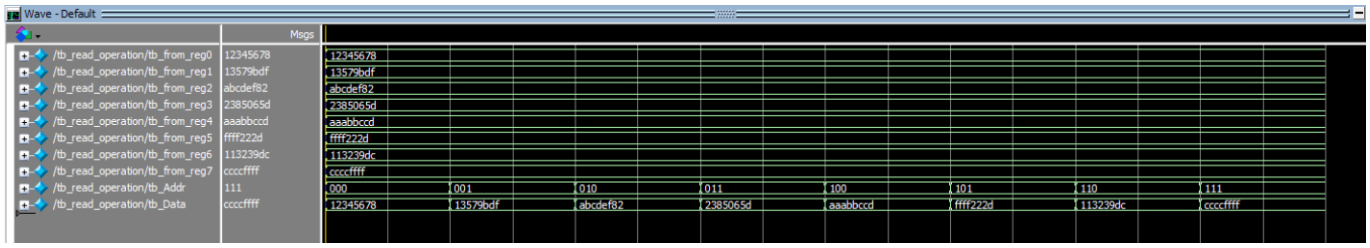
tb_register32_r_en의 waveform입니다. en이 1일 때는 clk의 rising edge에서 q가 d값을 따라가는 것을 확인할 수 있으며, en이 0일 때는 q가 계속해서 이전 q 값을 유지하는 것을 확인할 수 있습니다.

5) register32_8



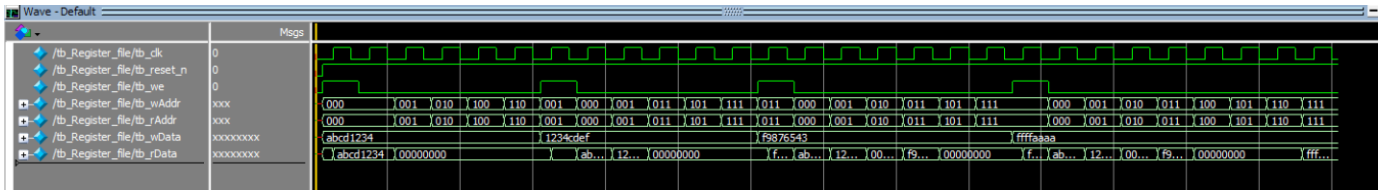
tb_register32_8의 waveform입니다. en의 값과 대응되는 d_out이 clk의 rising edge에서 d_in의 값을 따라간다. en의 값과 대응되지 않는 d_out들은 모두 이전의 값을 유지하는 것을 확인할 수 있다.

6) read_operation



tb_read_operation의 waveform입니다. Addr의 값에 대응하는 from_reg로부터 값을 읽어와 출력하는 것을 확인할 수 있습니다.

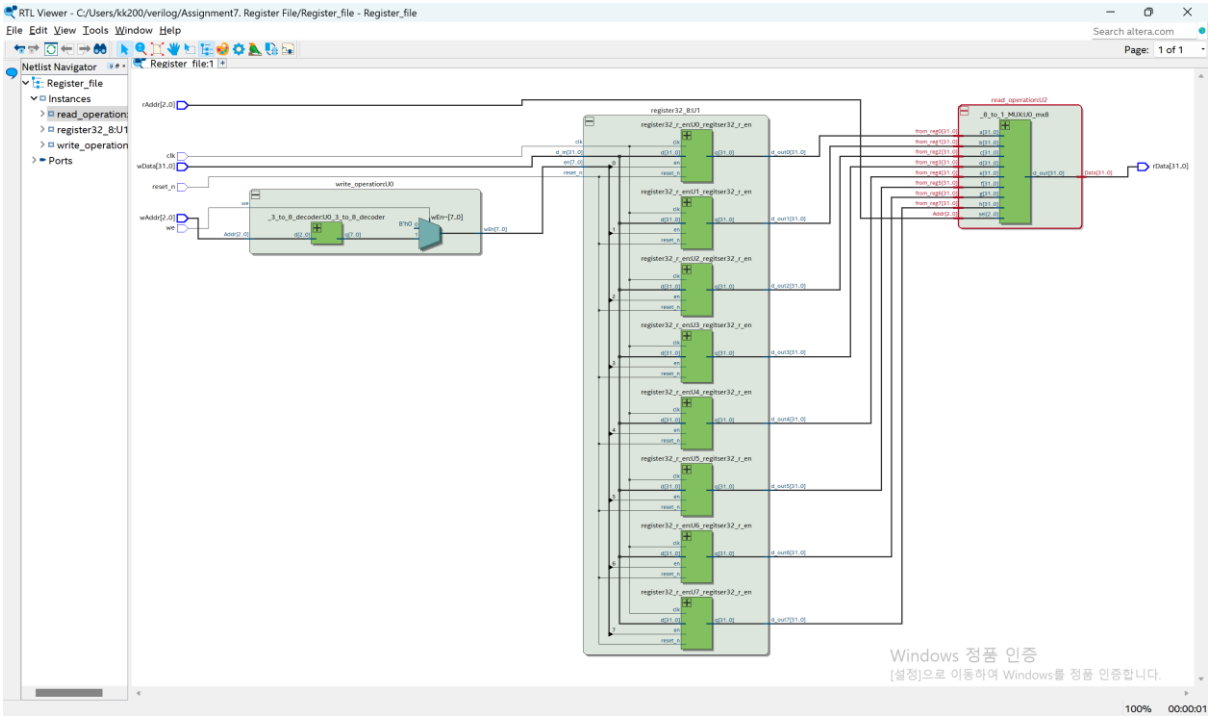
7) Register_file



tb_Register_file의 waveform입니다. we가 1일 때, clk의 rising edge에서 register의 값이 update 되어 rData의 값이 변하는 것을 확인할 수 있습니다. we가 0일 때는

wAddr가 어떤 값이든 register의 값을 update 시키지 않습니다.

B. 합성(synthesis) 결과



Register_file의 RTL Viewer입니다. write_operation 내부에서 3 to 8 decoder를 통해 출력된 값들을 we와 and 연산을 통해 도출된 wEn값을 register32_8에 입력합니다. register32_8에서는 입력 받은 wEn의 값과 대응되는 register를 wData 값으로 update 시켜줍니다. 그리고 모든 register의 값을 read_operation으로 넘겨주고 read_operation에서는 rAddr의 값에 대응하는 register의 값만을 출력합니다.

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 24 02:24:58 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_file의 Flow Summary입니다. total pins는 clk 1bit + reset_n 1bit + we 1bit + wAddr 3bits + rAddr 3bits + wData 32bits + rData 32bits = 73bits입니다.

Logic utilization은 123임을 확인할 수 있습니다.

6) 고찰 및 결론

A. 고찰

write_operation에서 we가 1일 때는 wEn에 w_a 값을 assign 하고, we가 0일 때 wEn에 0을 assign하기 위해서 if else 문을 활용했습니다. 하지만 계속해서 if else 문에서 오류가 발생하였고, 그 이유를 알아본 결과, assign문은 if 조건문과 함께 쓰일 수 없다는 것을 알게 되었습니다. 따라서 assign문을 어떠한 조건과 함께 사용하고 싶으면 삼항연산자를 쓰면 된다는 사실을 깨닫게 됐습니다.

B. 결론

write_operation에서 3bit data를 8bit one-hot coding으로 바꿔줌으로써 단 한 개의 register에만 값을 업데이트 할 수 있도록 설계했습니다. read_operation module에서는 register의 값들 중에서 원하는 값을 출력할 수 있도록 mux를 통해 입력 받은 3bit input 값에 맞는 적절한 한 가지 값을 출력해준다. 따라서

register에 저장하고, 저장돼 있는 것들을 출력하는 방식을 실제 파일관리에 적용한다면, 더 편리해질 수도 있겠다고 생각했습니다.

7) 참고문헌

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2023

이준환 교수님/컴퓨터공학기초실험2/광운대학교(컴퓨터정보공학부)/2023