D

컴퓨터 공학 기초 실험2 보고서

실험제목: Multiplier

실험일자: 2023년 11월 03일 (금)

제출일자: 2023년 11월 22일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 0, 1, 2

학 번: 2020202037

성 명: 염정호

1. 제목 및 목적

A. 제목

Multiplexer

B. 목적

64bit 피승수와 64bit승수를 외부로부터 입력 받아 곱셈을 수행하는 하드웨어인 booth multiplier를 구현한다.

2. 원리(배경지식)

Booth Multiplier : Booth Multiplier란 2의 보수 표기법으로 두 개의 부호 있는 이진수를 곱셈하는 곱셈 알고리즘이다. 이 알고리즘은 기본적인 곱셈 알고리즘 보다 더 효율적으로 동작하며 곱셈을 위한 연산에서 추가 연사 횟수를 줄인다.

Booth Multiplier의 핵심 알고리즘은 이웃하는 비트 쌍을 검사하여 부분 곱을 생성하는 것이다. 곱하는 숫자인 승수의 특정 비트 패턴을 찾아 이러한 패턴을 기반으로 곱셈기 값에 대한 덧셈과 뺄셈을 수행함으로써 곱셈과정을 최적화 한다.

1. 곱셈 피연산자들을 이진수로 표현한다.
2. 승수의 비트 패턴을 분석한다. 승수의 이웃하는 비트 쌍을 검사하여 특정 패턴을 ㅂ 찾는다.
3. 부분 곱을 생성한다. 승수의 비트 패턴에 따라 부분 곱을 생성한다. 곱셈기의 값에 대해 덧셈 또는 뺄셈을 통해 계산된다.
4. 부분 곱을 합친다.

Rule of booth multiplication

Radix 2

X(i)	X(i-1)	Operation	Description	Yi
0	0	Shift only	String of zeors	0
0	1	Add and shift	End of a string of ones	1
1	0	Subtract and shift	Beginning of a string of ones	1′
1	1	Shift only	String of ones	0

X(i)현재 비트, X(i-1)이전 비트, Yi

현재 값과 이전 값이 같은 경우 (00 or 11인 경우)오직 shift만 진행

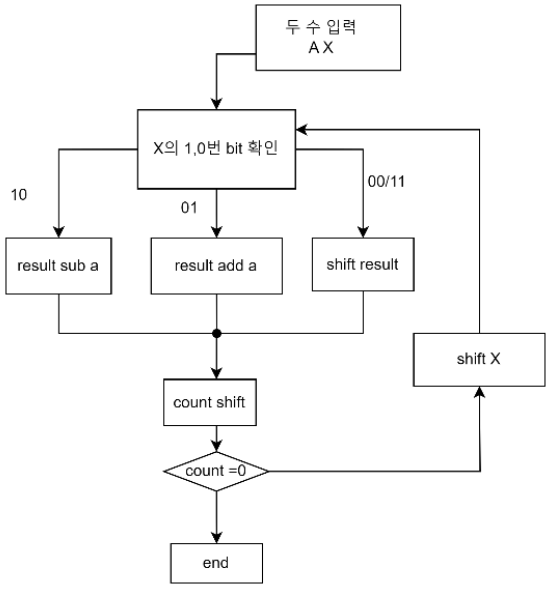
현재 값과 이전 값이 다른 경우 경우에 따라 다르게 연산

01 : 1의 연속이 끝나고 0이 시작되는 경우 2^i값을 가산한다.(1′)

10 : 0의 연속이 끝나고 1이 시작되는 경우 2^i값을 감산한다.(1′)

이러한 순서로 Yi의 코드가 작성된다.(signed digit)

3. 설계 세부사항



위 그림은 booth multiplier의 전체적인 흐름을 나타낸 순서도 이다

-구현해야 할 모듈 기능

현재 값을 확인하고 shift, add sub연산 중 하나를 선택 해주는 모듈 – select_func

해당 모듈에서는 피승수, 승수, 현재 누적된 결과값, count를 입력 받아 다음 결과값과 승수의 변화를 출력한다.

다음 result값은 현재 승수의 [1:0]번 비트에 의해 정해지며 count가 0인 경우 연산의 종료를 나타내기 때문에 입력 받은 값 그대로를 출력한다.

승수의 [1:0]번 비트에 따라 다음과 같이 세가지 경우로 나뉜다.

00/11 : 1또는 0의 연속인 경우 결과값을 shift해준다.

10 : 0의 연속이 끝나고 1이 시작된 경우 피승수 값 만큼 result의 최상위 64비트에서 감산한다.

01 : 1의 연속이 끝나고 0의 시작점인 경우 피승수 만큼 result의 최상위 64비트에서 가산한다.

감산과 가산의 경우 CLA모듈을 이용했고 감산의 경우 2의 보수를 취하기 위해 1의 보수를 취해준 뒤 감산이 필요한 경우에 cla의 첫 cin에 1을 넣음으로써 2의 보수의 형태로 들어갈 수 있도록 설계했다.

최상위 64비트에서 계산이 이루어 지는 이유는 연산 과정에서 shift가 진행함에 따라 계속해서 값이 오른쪽으로 밀려 자신의 위치로 찾아가기 때문이다.

마지막 쉬프트에서 next_result[127]= next_result[126];를 추가해준 이유는 >>>를 이용한

경우 msb를 그대로 유지한체 shift되어야 하는데 그렇지 않아서 값의 변화를 막기 위해 이전 비트를 그대로 유지할 수 있도록 추가해 주었다.
추가적으로 op_START가 1일 때는 다음 값을 생성하지만 OP_START가 0인 경우 이전 값을 그대로 내보내도록 설계하였다.

- select_func에서 도출된 값을 현재 값에 반영시켜주는 모듈 - Register_r_op

Clk,reset, op_clear, op_start값에 따라 현재 값을 변화시킨다. Reset과 clear는 같은 역할을 하도록 수행하였다. Op_clear의 용도는 다른 모듈과 같이 쓰일 때 사용 된다고 들었는데 이번 과제에서는 곱셈기 단일 모듈임으로 위와 같이 설계하였다.
리셋, 클리어 작동시 해당 모듈 내의 레지스터를 전부 0값으로 돌릴 수 있도록 설계하였다. 따라서 리셋 또는 클리어 활성화를 한다면 처음부터 다시 계산이 실행된다.
그 이외의 경우에는 select_func로 부터 생성된 값을 반영 할 수 있도록 설계하였다.

Cla64

- 이전에 용한 32bit cla를 64bit로 확장.

Main multiplier 모듈

1. 현재 Xi,Xi-1값을 받아서 결과값에 shift, add, sub연산이 가능한 기능
2. op_clear가 들어왔을 때 모듈 전체를 초기화 해줄 기능,

Top module – multiplier

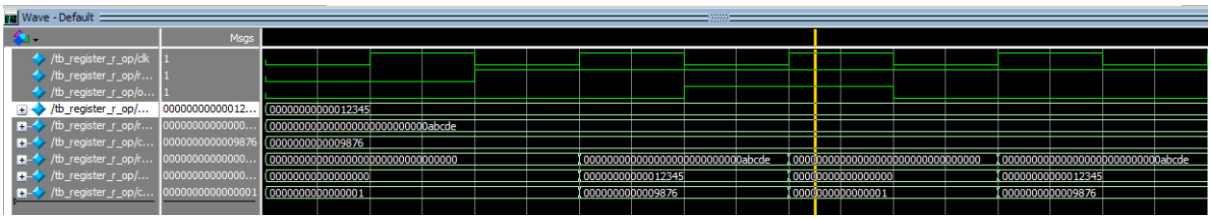
승수, 피승수, 클럭, 리셋, op를 입력 받는다.

Count가 1일 때 X값에 0을 붙여서 연산을 진행해야 하기 때문에 카운트 1에 대한 예외와 count가 0 즉 연산이 끝났을 때 op_done출력을 여기서 처리해 주었다. 이 외외으 경우는 mulr을 갱신해주는 역할만 수행한다. Mulr의 리셋 또한 처리해야 하기 때문에 resiger에 mulr2를 연결하고 해당 값을 mulr로 옮기도록 하였다. logic변수를 이용해 처리하고 싶었지만. 사용이 불가능 했다.

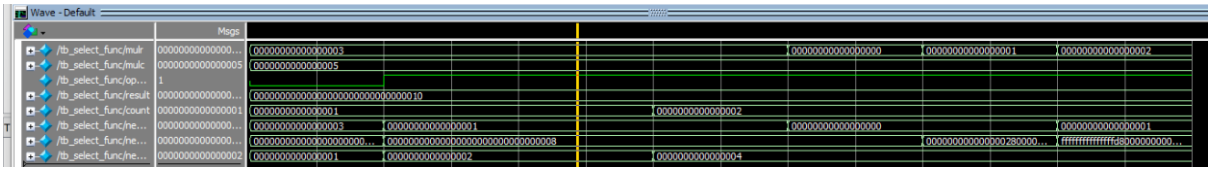
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

tb_register_r_op



Reset이 0이거나 op_clear가 활성화 되었을 때 다음 값이 0인 것을 확인 가능하다.(count의 경우 초기값을 1로 세팅)



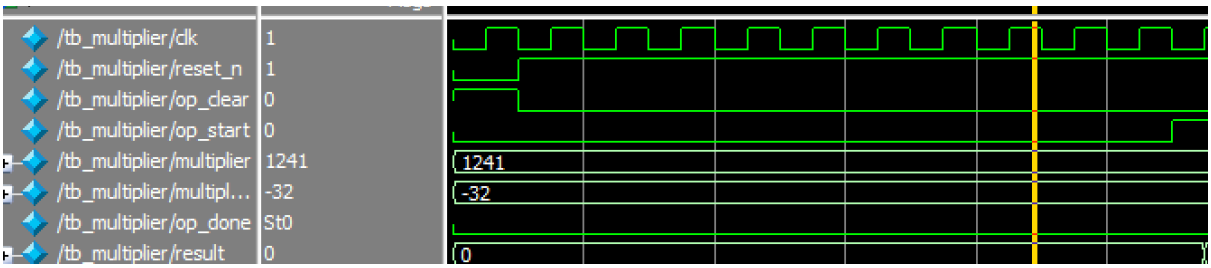
Op_start가 0일 때 이전 값을 유지

11일 때 shift

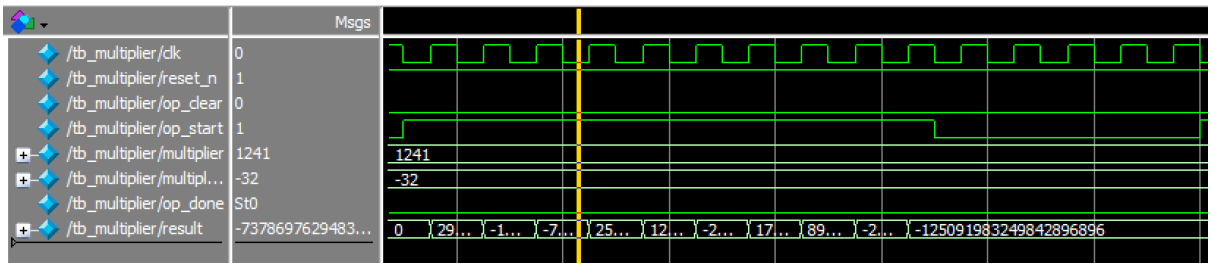
덧셈 동작과 뺄셈 동작 확인.

Top module:

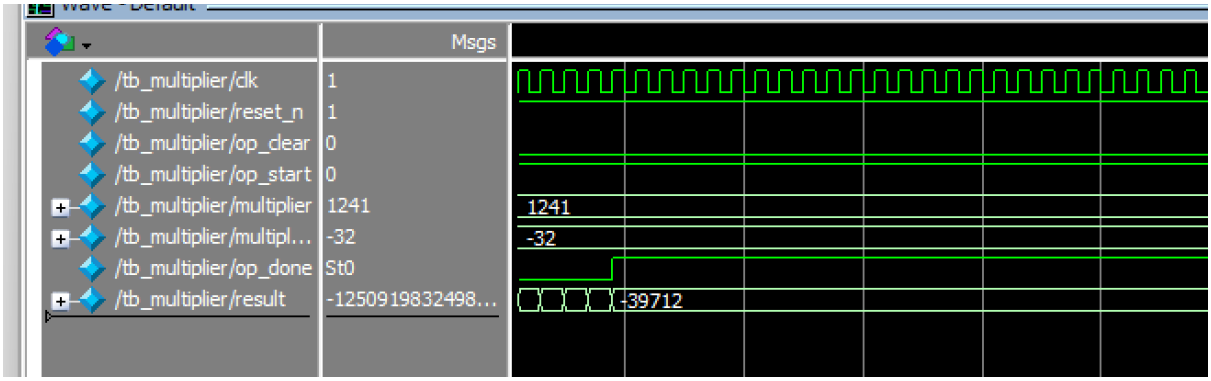
양수 * 음수



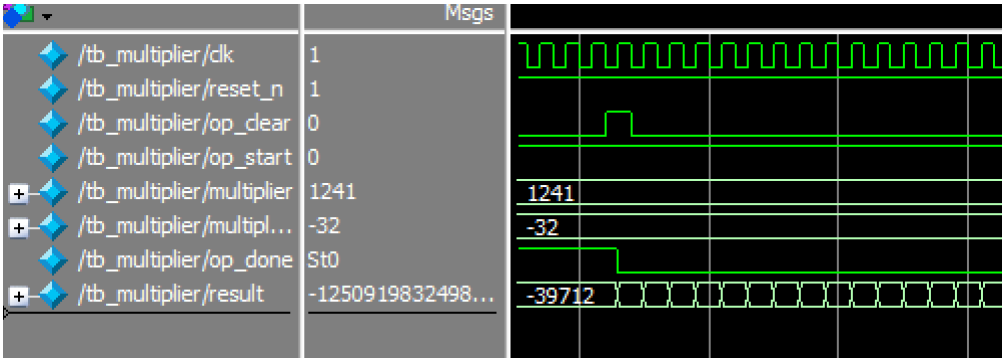
Op_clear, start 작동 여부 확인. 값 생성 x



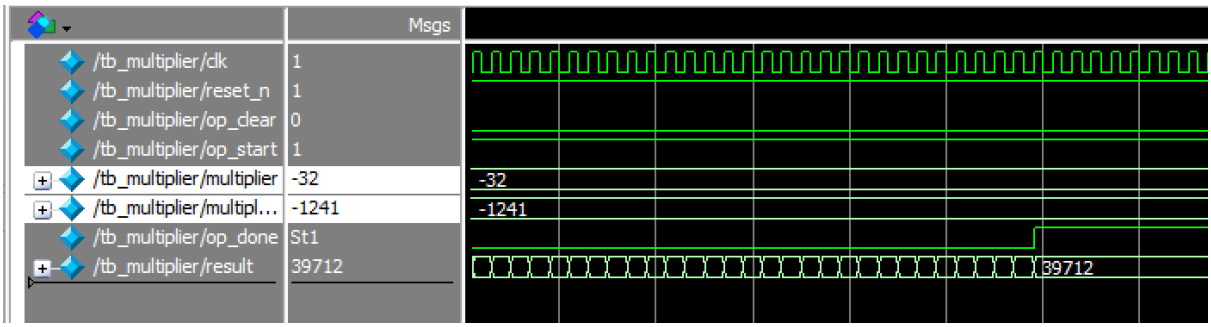
연산 진행, 중간에 op_start 비활성화시 정지



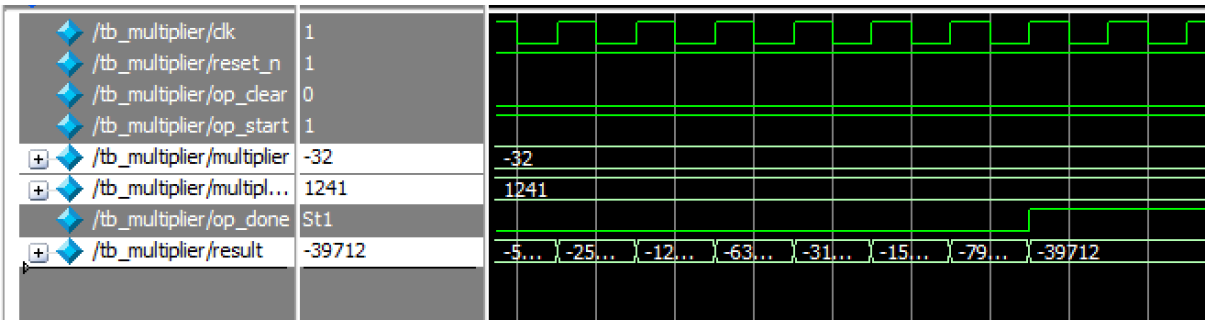
마지막 값 확인후 연산 중지, 연산 결과 성립, 이후 연산 x , done활성화



Op clear진행 후 다시 값 재 생성 확인.

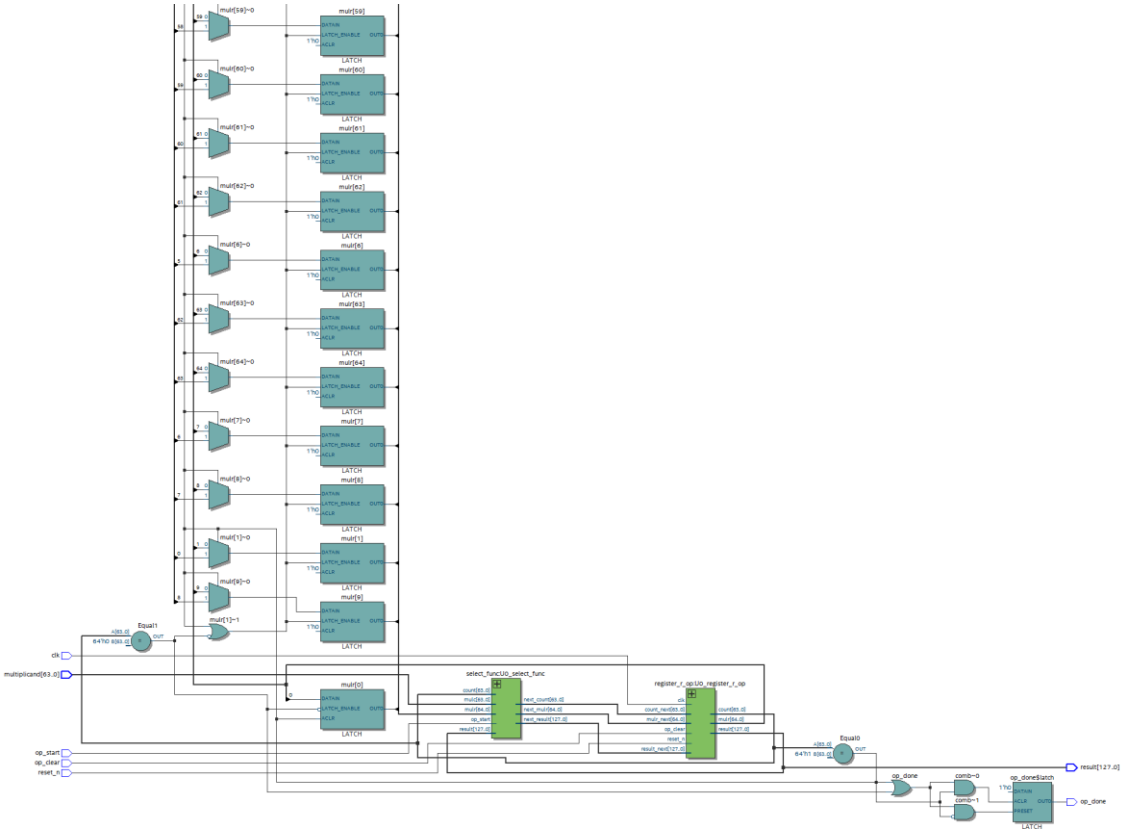


두 수 모두 음수일 때



음수 * 양수

B. 합성(synthesis) 결과



Flow Summary	
<<Filter>>	
Flow Status	Successful - Wed Nov 22 23:54:09 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	256
Total pins	261

총 261개 핀 사용 256개의 레지스터

5. 고찰 및 결론

A. 고찰

가장 문제가 많았던 과제였다.128bit shift연산중에 62번재 bit가 x가 나오는 경우가 있었다. 아무리 찾아도 모르겠어서 next_result에 이전 결과값을 넣고 다시 새로운 값을 넣어주니 작동지 잘 됐다. 아마 비어있는 레지스터에 값을 일부분만 삽입하고 쉬프트 연산을 진행하니 63번비트가 62번으로 이동하면서 x가 출력된 것 같다.

또한 128비트 쉬프트 연산에서 마지막 sign bit가 이어지지 않는 결과가 발생했다 혹시

몰라 <<. <<<둘 다 사용해 보았으나 답을 찾지 못해 원래 128번 자리에 존재했던 127번을 쉬프트 후에 옮겨주는 것으로 문제를 해결 할 수 있었다.
베릴로그를 구성하면서 현재 가장 어려운 것은 문법이나 변수들의 쓰임이 익숙하지 않다는 것이다. C++의 경우 이제 문법이 익숙해 로직을 잘 짜야하는 반면 베릴로그의 문법은 아직 익숙하지 않아 문법 공부를 하면서 코드를 짜야하니 복잡함이 배가 되는 것 같다.

B. 결론

부스 멀티플라이어의 동작 방식을 이해할 수 있었다. 처음 해당 모듈을 구성할 때만 하더라도 마지막 비트가 1일 때 예외 처리나 처음 연산 시 앞에 0이 있다고 가정하는 등에서 예외처리나 신경써야 할 부분이 많다고 느꼈지만, 생각보다 어렵지 않았다. 승수를 분석하는 코드만 잘 구상한다면 결과값은 쉽게 도출 가능했다. Radix2로 진행하다 보니 연산 결과를 확인하는데 64번의 사이클이 필요했다. 프로젝트 구현시는 radix4로 구현해 연산 횟수를 더 줄여볼 생각이다.

6. 참고문헌

Booth mul / https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm
이준환/디지털논리회로2/광운대학교/2023
이형근/컴퓨터공학기초실험2/광운대학교/2023