컴퓨터 공학 기초 실험2 보고서

실험제목: Multiplier

실험일자: 2020년 10월 26일 (월)

제출일자: 2020년 11월 09일 (월)

학 과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2019202052

성 명: 김 호 성

1. 제목 및 목적

A. 제목

Multiplier

B. 목적

Booth multiplication 기능을 radix-2로 구현해본다. 64비트의 값끼리 곱해서 128비트의 결과가 나오는 하드웨어를 설계해본다.

2. 원리(배경지식)

-Booth multiplication

Booth multiplication이란 비트 수가 같은 두 숫자를 곱할 때 연산자의 비트를 쪼개서 피연산자에 곱하는 방법이다. 이번 실험에서 구현한 방법은 radix-2 booth multiplication인데 이는 연산자의 LSB 뒤에 0이 있다고 가정한 후 두 bit씩 각각의 경우에 따라 연산을 실행하는 방식이다. 각 경우 별로 결과값을 32비트 변수 u, overflow 된 값을 32비트 변수 v, LSB를 변수 x_l이라 하고 그 왼쪽 비트를 x라 한다면

- x, x_l = 0, 0일 경우

  u 값을 오른쪽으로 shift 및 0으로 채워주고 거기서 빠져나온 LSB가 v 값의 MSB가 되어서 v를 또 오른쪽으로 shift 해준다.

- x, x_l = 0, 1일 경우

  이전 연산 결과 u값과 피연산자를 더한 후 오른쪽으로 shift 해준다. V는 마찬가지로 u에서 빠져나온 LSB를 MSB로 가지고 오른쪽으로 shift 해준다.

- x, x_l = 1, 0일 경우

  피연산자와 이전 결과 u값의 2의 보수 방식으로 감산해준 후 그 값을 오른쪽으로 shift 해준다. V는 이번에도 u에서 빠져나온 LSB를 MSB로 가지고 오른쪽으로 shift 해준다.

- x, x_l = 1, 1일 경우

  u 값을 오른쪽으로 shift 및 1로 채워주고 v 값은 이전과 같다.

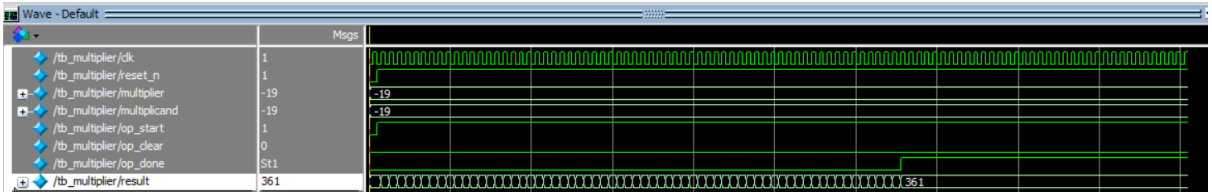
이는 기존의 곱셈 연산 중 한 bit씩 곱한 결과를 더하는 방법에서 더하는 과정을 생략하고 오로지 비트의 shift 연산으로 계산을 진행하여 마지막에 나온 u와 v값을 붙여주는 방식이기 때문에 더 효율적이라고 할 수 있다.

3. 설계 세부사항

FSM은 다음과 같다. IDLE 상태는 계산을 시작하기 전이다. op_start 신호가 들어오면 계산을 시작하는데, EXEC 상태를 유지하면서 진행한다. 연산자와 피연산자가 64bit이므로 64번의 연산을 수행하며 횟수를 나타내는 변수 count가 64가 되면 연산을 마치고 DONE 상태로 가게 된다. 모듈을 초기화하는 신호인 op_clear가 1이 된다면 값들을 초기화하고 IDLE 상태로 간다.

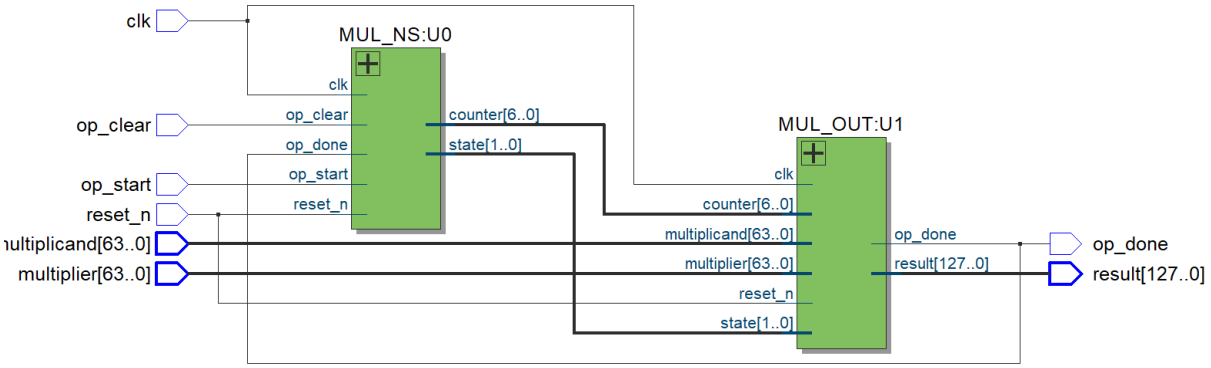
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

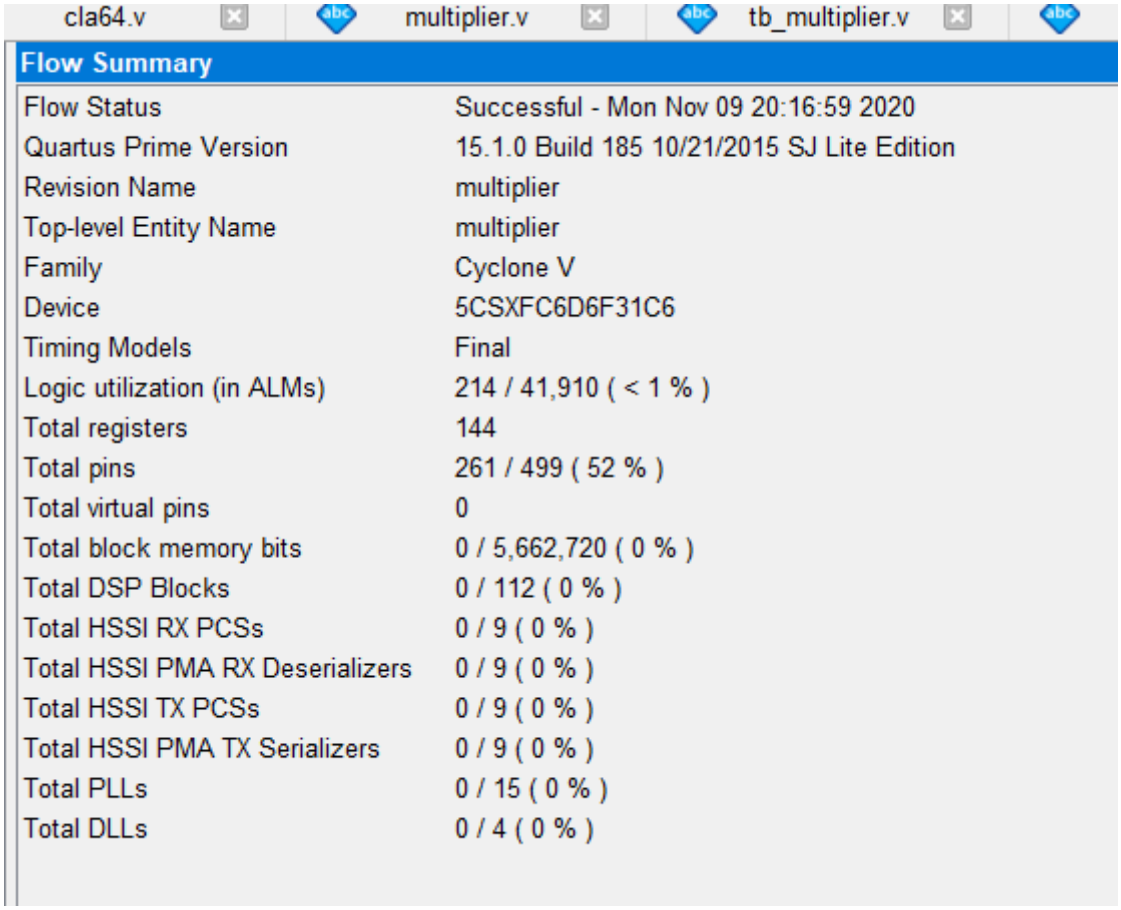


Op_start 신호를 미리 1로 설정하고 testbench를 실행하면 바로 연산을 하도록 설정했다. 총 64번의 cycle을 가지며, 64번의 cycle이 정상적으로 작동된 이후에 361이라는 값이 정상적으로 나오는 것을 확인할 수 있다.

B. 합성(synthesis) 결과



상태를 결정해주는 m_state, 계산하는 m_cal, 값을 저장하는 레지스터가 순서대로 붙어있다. M_cal 모듈의 내부를 보면 아주 복잡하게 생겼는데, 결과값 u와 피연산자를 더하거나 빼는 기능의 인스턴스 모듈들과 32비트 입력값을 기반으로 논리 게이트 들이 넓게 분포하고 있다.



Multiplier 내부에 구성된 회로나 레지스터 등의 개수는 위와 같다.

5. 고찰 및 결론

A. 고찰

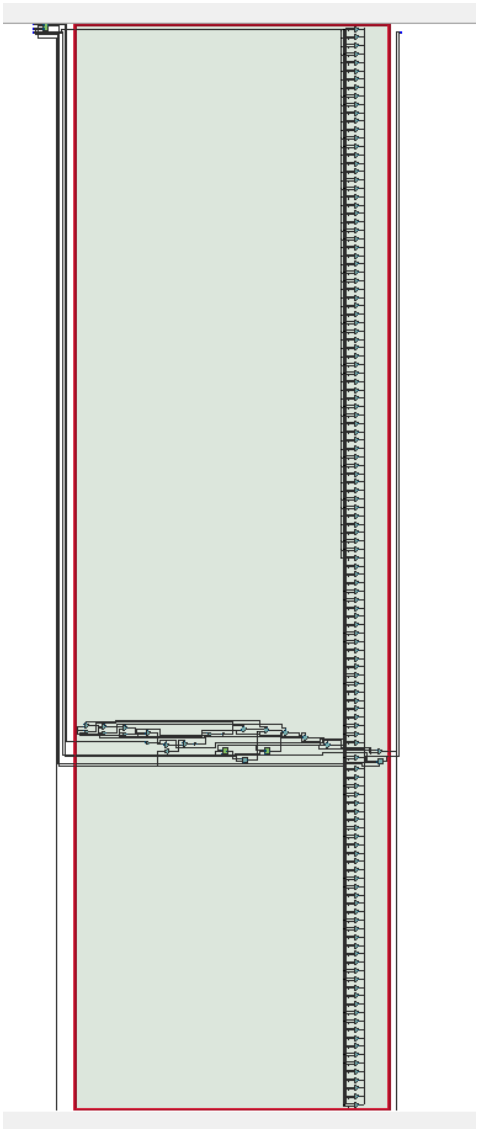
Radix-2방식을 사용하여 만들었으며, 기존의 곱셈 연산 중 한 bit씩 곱한 결과를 더하는 방법에서 더하는 과정을 생략하고 오로지 비트의 shift 연산으로 계산을 진행하여 마지막에 나온 u와 v값을 붙여주는 방식이기 때문에 더 효율적이라고 할 수 있다. Radix 2의 경우 Radix 4에 비해 CLK을 2배 소요한다는 점에서 큰 단점을 가지고 있다. 그러나, Radix 4에 비해 경우의 수가 적기 때문에 구현하는데 있어서 더욱 쉽다는 장점이 있다.

또, Radix 2를 구현하는데 있어서 지켜야할 경우의 수들은 원리(배경지식)파트에 서술했기 때문에 넘어가도록 한다.

B. 결론

이번 모듈의 구조는 FSM의 전반적인 구조와 크게 다르지 않다. 상태를 결정하는 모듈, 실행 모듈, 저장하는 레지스터 이렇게 구성되어 있다. 변수들의 크기나 종류가 많아서 이

하드웨어를 직접 만들려면 사람이 하기에는 꽤 많은 정성이 필요할 것 같다. 또, module 화가 개발자에 있어서 정말 필요하다는 생각이 들었다. 다음 사진은 MUL_OUT을 확대한 사진이다.



읽기조차 힘들 정도로 부품이 많이 들어간 모습을 확인할 수 있으며, 세로로 길게 나열된 부품들은 2 to 1 MUX이다. (128 bit)

6. 참고문헌

-