컴퓨터 공학 기초 실험2 보고서

실험제목: Multiplier

실험일자: 2023년 11월 06일 (월)

제출일자: 2023년 11월 17일 (금)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적

A. 제목

Multiplier

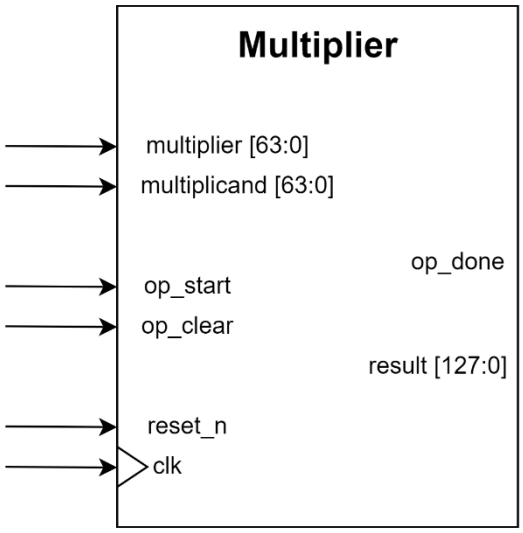
B. 목적

Booth Multiplier의 원리를 공부하여 이해하도록 한다. 이를 하드웨어적으로 하여 128-bit Multiplier를 설계할 수 있도록 한다. 산술연산은 CLA를 활용하여 응용 설계를 한다.

2. 원리(배경지식)

- Multiplier

승수와 피승수를 곱하는 곱셈기로, Booth Multiplier원리를 따르는 하드웨어이다. Booth Multiplier는 2의 보수 법으로 부호가 존재하는 2개의 이진수를 곱셈 연산하는 곱셈기이며, 이번에 설계하는 multiplier의 schematic symbol은 다음과 같다.



Booth Multiplier는 알고리즘의 방법에 따라 Radix 종류가 달라지게 되는 형식이다. 기본적인 Radix-2의 곱셈 알고리즘은 다음과 같다.

Multiplier(피승수)와 Multiplicand(승수)를 곱할 때, 먼저 multiplier의 하위 bit인 LSB를 x1이라 하고 추가로 0을 더 붙여서 x0으로 정해 놓는다. 그 다음 이 두 개의 bit를 비교하여 그 종류에 따라 연산 방법을 달리 하여 계산한다. 각각 bit가 0, 0이거나 1, 1이면 ASR(Arithmetic Shift Right) 연산을 진행한다. (Verilog: >>>)

bit가 0, 1인 경우에는 덧셈을 한 뒤 마찬가지로 shift 연산을 진행하는데, 이 때 덧셈은 result의 상위 절반 bit와 multiplicand를 더한다. 이렇게 나온 새로운 result와 기존 result 하위 절반 bit를 다시 하나의 result로 저장한다. 이후 ASR로 shift하는 것이다.

bit가 1, 0인 경우에는 뺄셈을 한 뒤 ASR 연산을 한다. 이때도 같은 방식으로, result의 상위 절반 bit에서 multiplicand를 뺀다. 이렇게 나온 새로운 result와 기존 result 하위 절반 bit를 다시 하나의 result로 저장하고 나서 ASR로 shift연산을 한다. 뺄셈 연산을 할 때에는 multiplicand를 NOT연산 하여 보수를 취한 다음 1을 더하여 같은 Adder로 뺄셈 연산까지 진행할 수 있다.

이러한 연산이 진행되고 나면 multiplier의 bit를 EOR, LSR, ASR 상관없이 오른쪽으로 shift하여 다시 새로운 LSB와 기존 하위 bit로 두 개의 bit를 temp로 묶어서 비교하면서 위 과정을 반복한다. 이때 연산 횟수를 count하여 multiplier의 bit수와 같아지면 최종 result값이 나온다. 64bit일 경우 64번 count하며, cycle횟수도 똑같이 64cycle이 걸린다. 이런 식으로 2bit씩 비교하여 곱하는 형태는 Radix-2가 되고, 이 외에도 Radix-4, Radix-16등 더 많은 bit를 한번에 비교하여 더욱 빠른 시간안에 계산할 수 있는 곱셈기도 있다.

3. 설계 세부사항

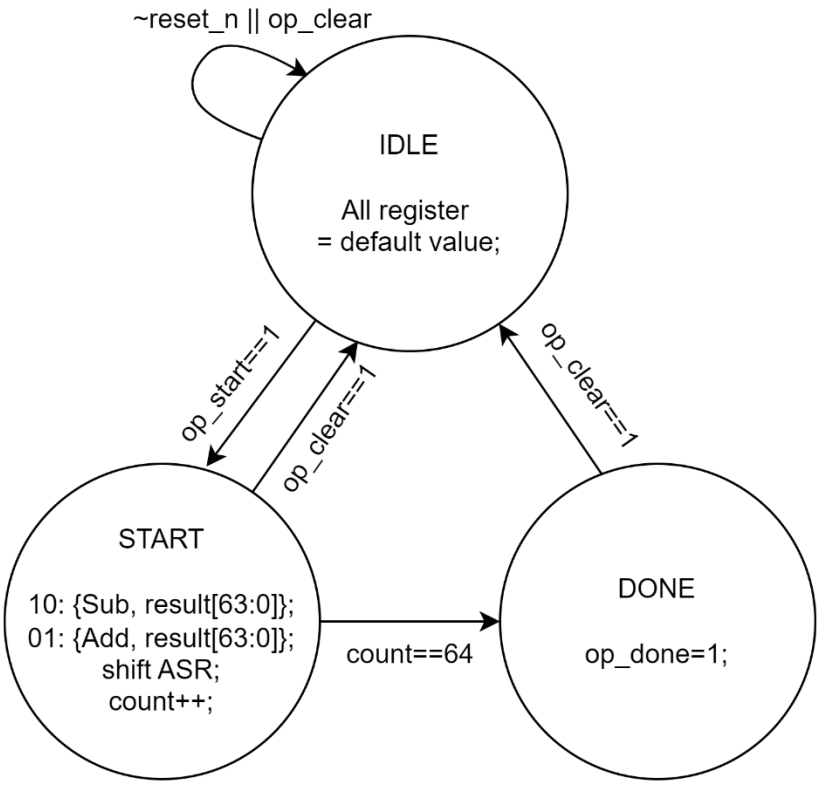
A. Multiplier

Radix-2의 Booth Multiplier 원리에 따라, multiplier의 LSB와 추가로 0을 붙여서 temp에 저장한 뒤 case문으로 비교하여 연산을 진행한다. 이때 연산은 64-bit CLA를 instance하여 계산한다.

-I/O

input	clk	clock
	reset_n	Active low reset
	multiplier [63:0]	승수
	multiplicand [63:0]	피승수
	op_start	start operation
	op_clear	clear operation
output	op_done	done operation
	result [127:0]	multiplier result

-state transition diagram



B. 64-bit CLA

-I/O

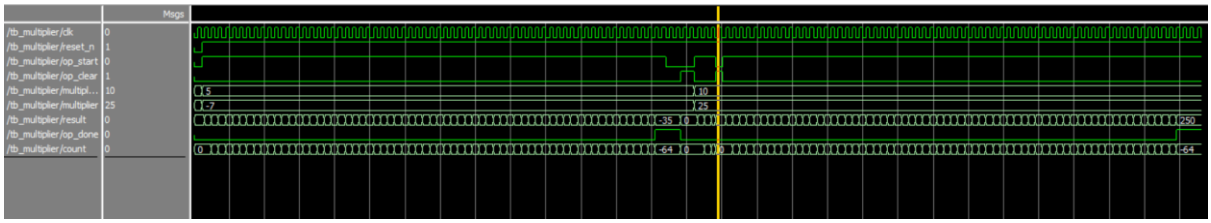
input	a [63:0]	data 1
	b [63:0]	data 2
	ci	carry in (뺄셈 용)
output	s [63:0]	sum
	co	carry out (wire 연결용)

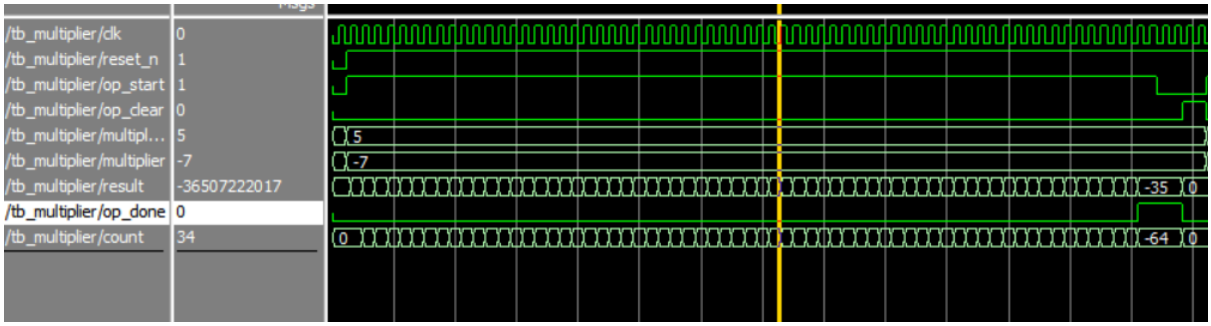
4. 설계 검증 및 실험 결과

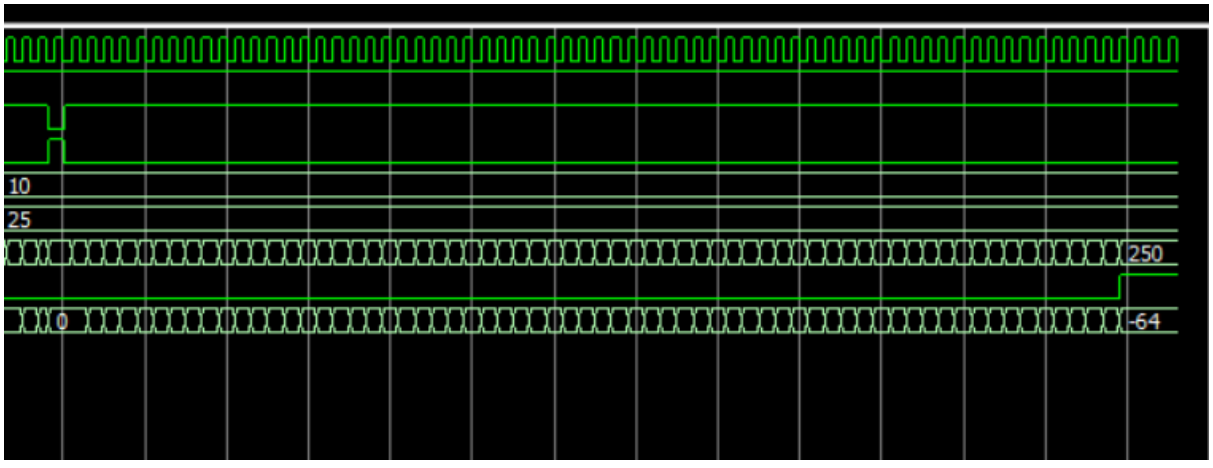
A. 시뮬레이션 결과

1. Multiplier

5 x (-7), 10 x 25를 차례대로 진행한 모습이다. 이때 정해진 output은 아니지만 cycle 횟수를 확인하기 위해 count 변수를 일시적으로 추가하여 확인하였다. 이번에 구현한 곱셈기는 Radix-2이므로 64 cycle만에 끝남을 count 변수로 확인해서, count가 64일 때 result 값이 나온 것을 볼 수 있다. 값을 확인한 뒤 op_done상태를 유지하다가 op_clear이 1로 되면 result와 count를 초기화하고 대기한다. 다시 op_start 신호가 들어오고 새로운 multiplicand와 multiplier가 들어오면 해당 값으로 연산을 시작한다. 이때 표시한 지점을 보면 연산 중간에 op_clear가 들어온 것을 볼 수 있는데, 이때는 모든 연산을 멈추고 다시 초기화 상태로 돌아간다. 이후 op_start가 다시 들어오면 그때 재연산을 진행하며, 64cylce때 결과 값이 나오게 된다.

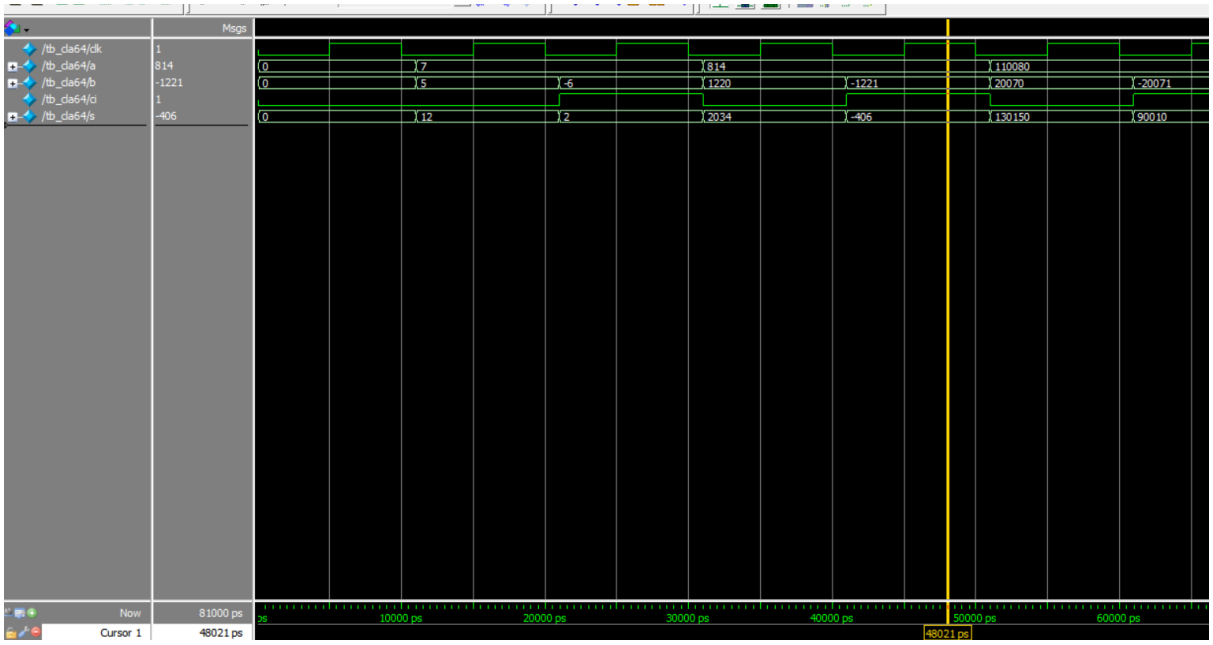






2. 64-bit CLA

이번에 Multiplier에서 산술연산을 위해 새로 만든 64-bit CLA module을 검증한 모습으로, 차례대로 덧셈 뺄셈을 번갈아 한 결과이다. 뺄셈의 경우는 b의 값을 NOT 연산을 하여 보수를 취한 뒤 1을 더하는 방식으로 하여 뺄셈 연산을 진행하였다. 따라서 CLA의 b의 값에는 ~b가, ci에는 1'b1이 들어가는 식이다. 이렇게 하여 값들이 큰 수가 되는 상황이나 뺄셈에서 b의 값이 더 큰 경우에도 정상적인 값으로 출력된 것을 볼 수 있다.

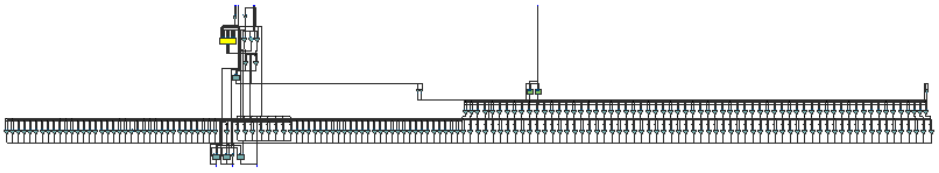


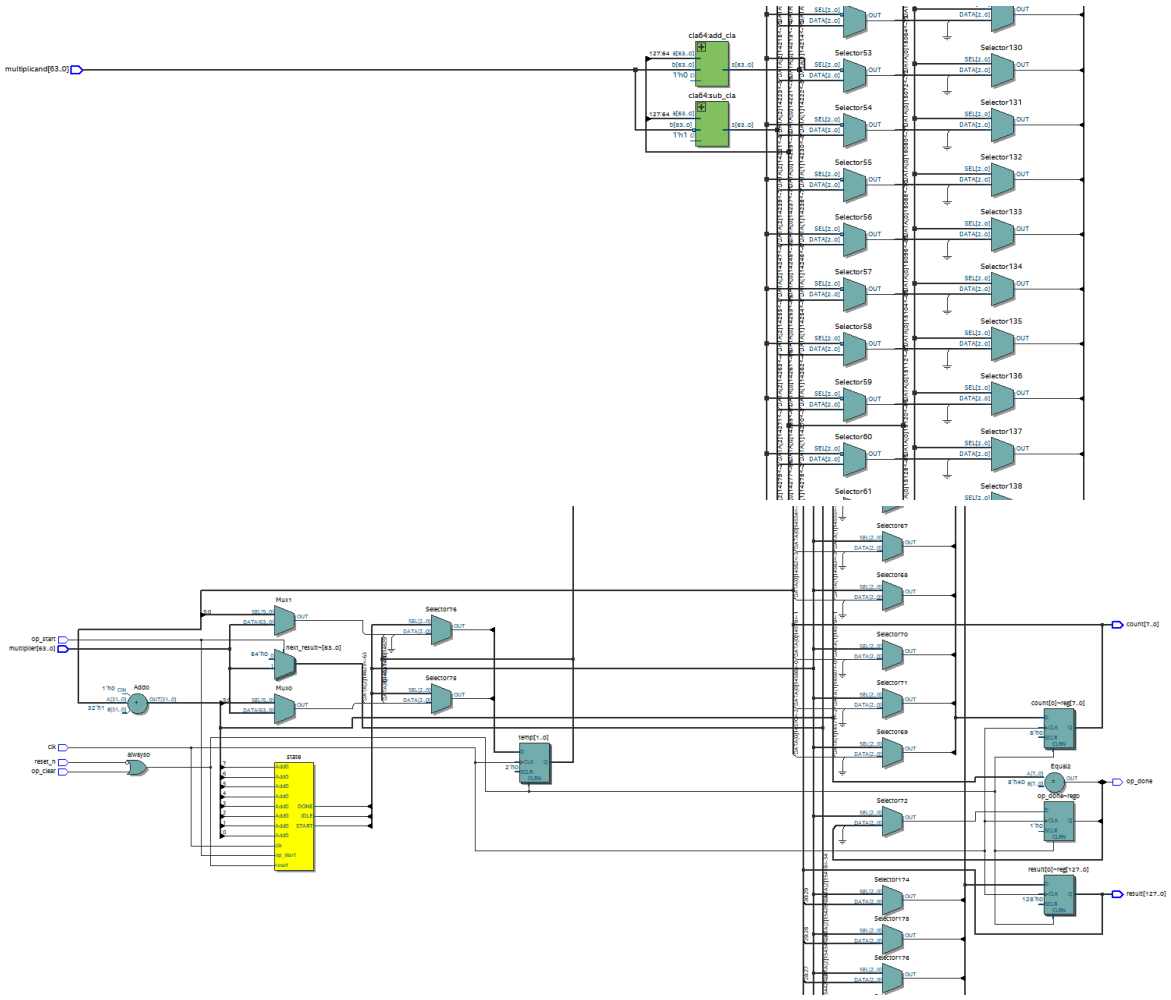
B. 합성(synthesis) 결과

1. Multiplier

-RTL viewer

전체 회로는 다음과 같다. 이를 주요 역할이 있는 부분으로 확대한 것이 2, 3번째이다.





-Flow summary

회로의 크기와 register의 수가 기존에 진행했던 다른 module들 보다 큰 편임을 확인할 수 있다.

Flow Summary

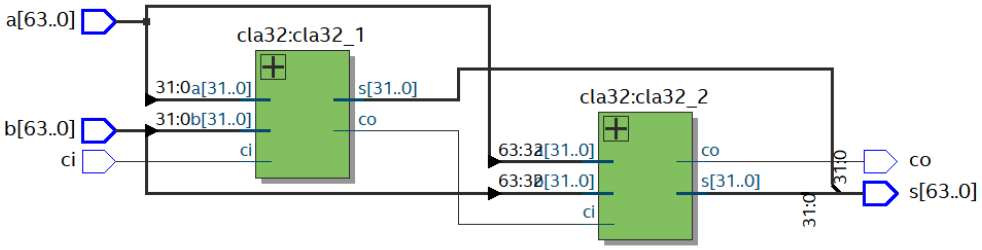
<<Filter>>

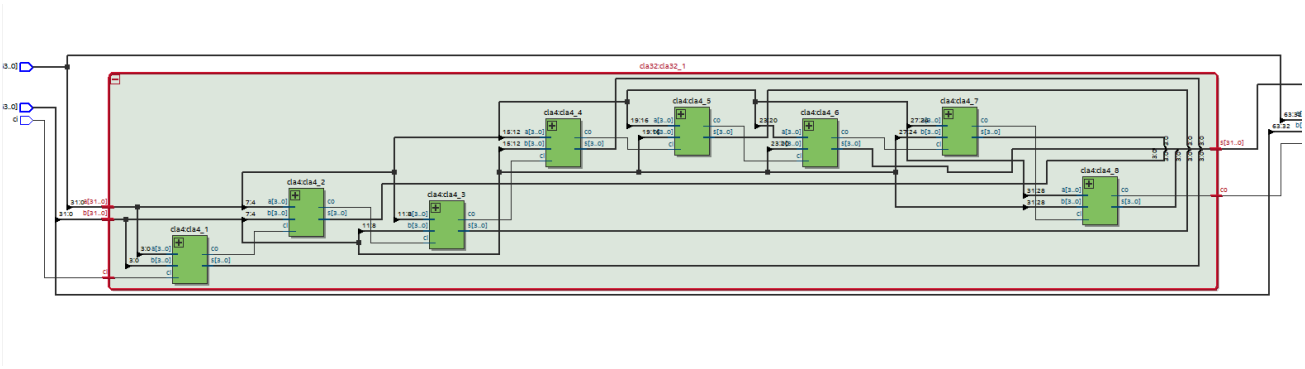
Flow Status	Successful - Thu Nov 16 22:44:32 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	295 / 41,910 (< 1 %)
Total registers	146
Total pins	269 / 499 (54 %)
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 %)

2. 64-bit CLA

-RTL viewer

기존에 만들었었던 32-bit CLA를 instance하여 64-bit CLA module을 설계한 것을 확인할 수 있다.





-Flow summary

회로의 크기를 확인할 수 있다. CLA는 register가 필요 없으므로 수가 0임을 알 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 16 23:09:24 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cla64
Top-level Entity Name	cla64
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	103 / 41,910 (< 1 %)
Total registers	0
Total pins	194 / 499 (39 %)
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 %)

5. 고찰 및 결론

A. 고찰

64-bit CLA를 검증하는 과정에서, Error가 발생하였는데 이는 64bit으로 계산할 수 있는 최대치를 넘은 값을 테스트 값으로 줘서 발생한 문제였다. 이에 범위를 확인하고 다시 적절한 값을 넣으니 정상적으로 결과가 출력될 수 있었다.

처음 Multiplier를 설계할 때 state를 IDLE과 START만 설계하고 Done상태에서는 단순히 START state에서 count를 확인하여 op_done 신호를 보내는 식으로만 하였다. 그 결과 검증하는 과정에서, 연산이 끝나고 op_clear 신호가 들어오기 전까지 그 상태를 유지하지 못하고 계속 값이 변하였다. 그래서 DONE state를 추가로 encoded하고 count가 64일 때 next_state를 DONE으로 바꾸었으며, DONE state 상태에서는 next값들을 현재 값으로 계속 저장하는 식으로 하였더니 값이 바뀌지 않고 정상적으로 출력될 수 있었다.

op_clear 신호는 처음에 각 state에서 op_clear신호를 확인하여 1일 시 IDLE state로 전환되도록 설계하였는데, 이렇게 하니 코드가 불필요하게 길어지는 등 문제가 생겨서 다시 확인해보니 op_clear 신호는 clk과 동기화 되게 설계하는 것이었다. 이에 sequential circuit part의 always문에서 posedge로 op_clear의 변화를 확인하여 reset_n과 or로 함께 초기화 하도록 진행하였더니 문제없이 작동될 수 있었다. 이때도 posedege말고 negedge로 하였다가 에러가 발생하여서, op_clear 신호는 1일 때 작동되는 것인 posedge로 수정하였었다.

B. 결론

이번 Booth Multiplier를 설계할 때 처음에는 원리를 이해하는 데에 시간을 많이 썼던 것 같다. 이를 구현하기 위해 직접 여러 곱셈 연산을 해보면서 결국 연산할 때 필요한 것은 result의 상위 절반 부분과 multiplicand만 가지고 덧셈, 뺄셈을 한다는 것을 깨달았다. multiplier는 하위 2bit으로 확인하는 용으로만 사용되니 따로 reg 변수에 저장하면 된다는 점도 알게 되었다.

이번에는 Radix-2로 설계를 진행하였다. Radix-2는 2bit씩 비교한다는 특징이 있다. 따라서 설계하기엔 다른 곱셈기보다 간단한 편이라는 장점이 있지만, 그만큼 시간이 오래 걸린다는 단점이 있다. 승수와 피승수의 bit 크기만큼 동일한 cycle 횟수가 발생하게 되기 때문이다. 그래서 이 외에 다른 Radix-n 방법들도 원리들을 더 공부해보면 여러 bit씩 비교하는 multiplier를 만들 수 있겠다고 생각하였다. 또한 이러한 곱셈기를 이용하여 산술, 논술 연산기인 ALU에 추가하는 등으로 응용할 수 있겠다고도 느꼈다.

6. 참고문헌

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007