

컴퓨터 공학 기초 실험2 보고서

실험제목: Multiplier

실험일자: 2018년 10월 24일 (수)

제출일자: 2018년 11월 06일 (화)

학 과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 수요일 0, 1, 2

학 번: 2015722025

성 명: 정 용 훈

1. 제목 및 목적

A. 제목

Multiplier

B. 목적

Multiplier의 기능을 이해하고 Multiplier의 종류와 연산 방법을 이해한다. Booth Multiplication의 계산 방법을 배우며 radix가 2인 경우와 4인경우의 장단점을 이해하고 Binary Multiplication의 연산방법과 차이점을 생각하며 구현한다.

2. 원리(배경지식)

Multiplier, 즉 곱셈기는 여러 종류가 있습니다. 가장 기본이 되는 Binary Multiplication, radix수에 따라 사이클이 변하는 Booth multiplication이 있습니다.

A. Binary multiplication

곱셈을 연산하는데 가장 이해하기 쉬운 곱셈기이다. 승수와 피승수를 1:1 대응시켜 대응시키는 승수의 값이 0이면 0의 값을 1이면 피승수에 해당하는 수를 나열하여 마지막에 더해지게 되면 곱셈의 값이 나온다.

피승수	0101	(5) Multiplicand
승수	X 0011	(3) Multiplier
<hr/>		
	0101	
	0101	
	0000	
	0000	
	<hr/>	
	0001111	

위 이미지는 곱셈기의 연산 과정을 나타낸 것이다. 비트만큼의 계산을 해야 하며 계산된 값들을 더해줘야 하므로 adder가 많이 필요하게 된다. Adder가 많이 필요하다는 뜻은 비용이 많이 들고 계산이 효율적이지 않다는 뜻이다. Binary multiplication은 계산 방법과 원리를 이해하는데 쉽지만 실질적인 하드웨어에 쓰이면 효율적이지 못한 곱셈기이다.

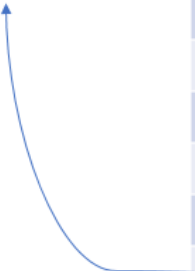
B. Booth multiplication

먼저 설명한 곱셈기의 단점을 보완하고자 나온 곱셈기이다. 연산 방법으로는 승수의 radix에 따라 경우를 나눠 덧셈, 뺄셈, Arithmetic Shift Right를 해주며 승수의 bits에 따라 cycle이 변하게 된다. 다음 표는 radix 2에 대한 경우를 나타낸 것이다.

X_i	X_{i-1}	Operation	Description
0	0	Shift only	String of zeros
0	1	Add and shift	End of a string of ones
1	0	Subtract and shift	Beginning of a string of ones
1	1	Shift only	String of ones

Radix의 수가 00인 경우 shift만 실행하며 01인 경우는 add를 실행한 후 shift를 하며 10인 경우는 Subtract를 실행한 후 shift를 하며 11인 경우에는 00과 마찬가지로 shift만 실행하면 된다. 위 표를 기반으로 Booth multiplication의 연산을 진행하면 아래와 같은 과정을 통하여 곱셈의 연산이 결정될 수 있다.

A	0111	(7)
X	10010	(-7)
<hr/>		
	11001111	(-49)



U	V	X	X-1
0000	0000	1001	0
1001			
1100	1000	1100	1
0111			
0001	1100	0110	0
0000	1110	0011	0
1001			
1100	1111	1001	1

Radix 2의 경우 Cycle의 수가 승수의 bits수만큼 나오게 된다. 이에 비해 radix 4는 승수의 bits의 절반만큼의 Cycle이 나오게 되므로 훨씬 효율적인 연산을 할 수 있다. 이와 마찬가지로 radix 8, radix 16으로 갈수록 radix에 대한 경우의 수는 증가하지만 Cycle을 줄일 수 있으므로 실제 하드웨어에서는 효율적이라고 할 수 있다. 이번 실습을 통하여 구현하게 될 곱셈기는 radix 4이며 radix 4의 경우는 아래 표와 같이 나타낼 수 있다.

X_i	X_{i-1}	X_{i-2}	Operation	Y_i	Y_{i-1}	Y
0	0	0	$0+0 = 0$	0	0	0
0	0	1	$0+A = A$	0	1	+1
0	1	0	$2A-A = A$	0	1	+1
0	1	1	$2A+0 = 2A$	1	0	+2
1	0	0	$-2A+0 = -2A$	-1	0	-2
1	0	1	$-2A+A = -A$	0	-1	-1
1	1	0	$0-A = -A$	0	-1	-1
1	1	1	$0+0 = 0$	0	0	0

Radix 4의 경우 radix 2와 마찬가지로 경우를 나눠 해당 경우가 해당되면 표에 따라 연산을 하며 shift를 하게 된다. 주의 할 점으로는 shift의 수가 2번이며 Cycle은 radix 2에 비해 반으로 줄어들게 된다.

3. 설계 세부사항

설계한 logic으로는 크게 두가지로 나뉘었습니다. 다음 state를 정해주는 next state와 state에 따른 계산을 위한 calculator를 설계하며 두가지 로직의 동작으로 곱셈을 계산할 수 있도록 하였습니다.

Multiplier_ns

구분	이름	설명
Input	Op_clear	계산 초기화 신호
Input	Op_start	연산 시작 신호
Input	Reset_n	Active low reset 신호
Input	State	현재 상태 표시
Input	Multiplier	승수
Input	Multiplicand	피 승수
Input	Count	현재 카운트 표시
output	Next_state	다음 상태 표시
output	Next_count	다음 카운트 표시
output	Next_multiplier	다음 승수 표시
output	Next_multiplicand	다음 피 승수 표시

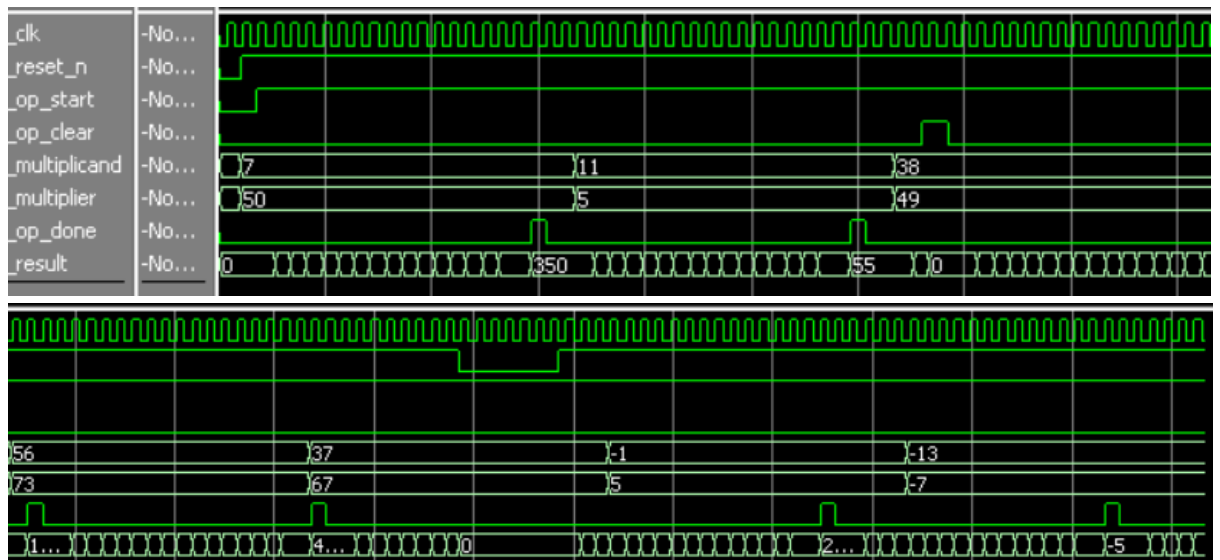
Multiplier_cal

구분	이름	설명
Input	Clk	Clk 신호
Input	Op_clear	연산 초기화 신호(계산되던 값 초기화)
Input	Reset_n	Active low reset 신호
Input	Multiplier	승수
Input	Multiplicand	피 승수
Input	State	현재 상태 표시
Input	Count	현재 카운트 표시
output	result	결과 출력

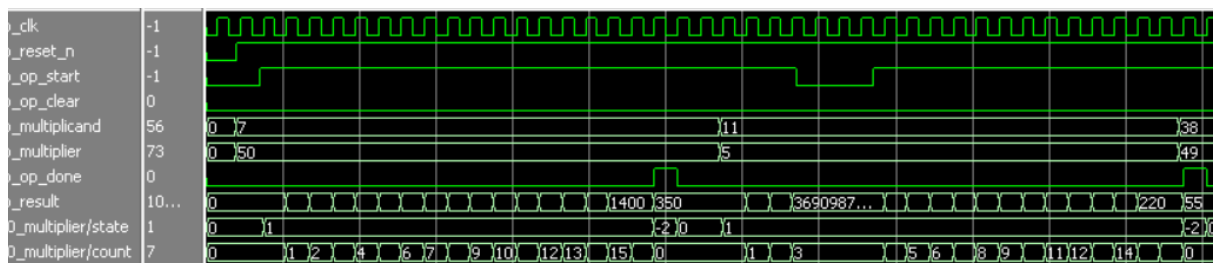
특히 calculator logic에서는 미리 계산되고 있던 v, m, u의 값이 result값에 영향을 미치기 때문에 똑같이 op_clear와, reset_n신호를 받게 설계하였으며 해당 신호가 들어오게 되면 값을 모두 초기화할 수 있도록 설계한 것이 특징입니다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



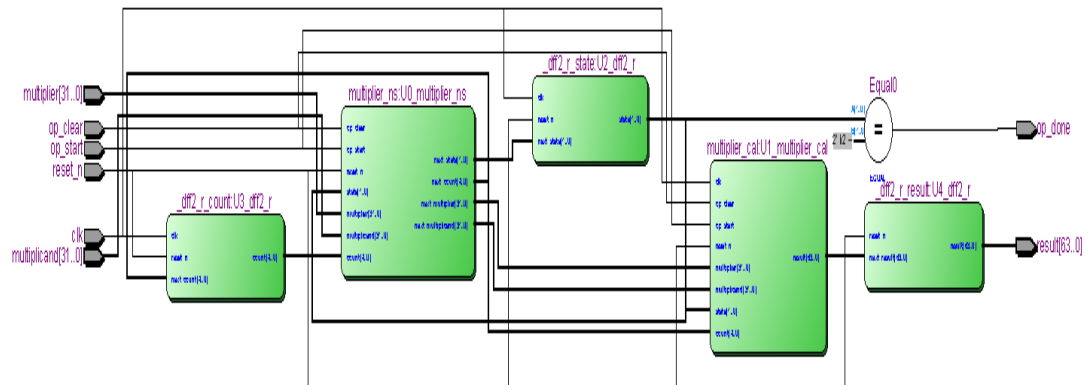
해당 wave form은 pdf의 waveform을 기반으로 음수 연산을 추가한 형태입니다. 테스트 벤치에서는 op_clear 신호와 reset 신호를 추가하였으며 다음은 그에 대한 설명입니다. 기본적으로 op_start신호가 있어야만 계산이 동작합니다. Op_clear나 reset신호가 없는 경우 승수와 피 승수의 값을 받아 순차적으로 계산을 진행하며 계산이 완료되면 다음 승수와 피 승수의 값이 들어오기 전까지 완료된 계산 값을 유지하는 것을 확인할 수 있습니다. 계산 도중 op_clear신호가 들어오게 되면 다음 클럭에 모든 값들이 초기화되며 op_clear신호가 중단되면 현재 승수와 피 승수의 값으로 처음부터 계산이 실행됩니다. Reset의 경우 동작은 똑같지만 해당 명령이 실행되는 타이밍이 다릅니다. Op_clear같은 경우 클럭에 맞춰 실행되지만 reset은 비동기식으로 실행되기 때문에 신호가 들어오는 순간 바로 모든 값들이 초기화되며 reset신호가 풀리면 다시 처음부터 계산을 진행하게 됩니다.



추가적으로 op_start는 연산 진행을 제어해주는 역할로 0으로 변하는 순간 연산을 멈추고 다시 1로 변하면 원래 하고 있던 연산을 이어서 하며 마무리하는 것을 확인할 수 있습니다. 또한 하고 있던 연산 중 다른 승수와 피 승수가 들어와도 원래 진행되던 계산은 계속 진행할 수 있도록 하였습니다.

B. 합성(synthesis) 결과

RTL Viewer



RTL Viewer입니다. 주요 logic으로는 ns logic과 cal logic이 있으며 op_done 앞에 Equal 소자는 state를 판단하여 DONE state와 일치하면 on이 되는 기능을 하고 있습니다.

Flow summary

Flow Summary	
Flow Status	Successful - Tue Nov 06 12:08:12 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	1,582 / 68,416 (2 %)
Total combinational functions	1,582 / 68,416 (2 %)
Dedicated logic registers	7 / 68,416 (< 1 %)
Total registers	7
Total pins	133 / 622 (21 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

Multiplier를 구현하는 실습을 진행하였습니다. 계산이 동작하는 구현 자체는 크게 어렵지 않았지만 op_clear와 reset이 걸린 후 다시 연산이 시작되게 하는 조건을 생각하는 것에서 많은 시간이 걸리게 되었습니다. 해당 문제는 model sim을 통해 wave form을 결과 값 뿐 아니라 다른 보이지 않는 데이터들을 담고 있는 변수를 불러와 함께 확인하며 현재의 상태와 count를 비교하며 조건으로 걸어준 동작을 실행하는지 안 하는지 확인하며 제대로 동작이 되지 않는 부분의 조건을 계속해서 바꿔주며 최대한 깔끔하게 동작 할 수 있도록 바꾸어 주었습니다. 또한 들어온 승수와 비 승수의 값을 연산 종료 후 다른 값이 들어오기 전에 유지시키기 위하여 승수와 비 승수의 값을 저장해 놓고 연산이 끝난 후 비교하였을 때 해당 값이 같으면 IDLE state를 계속 유지하며 계산을 멈출 수 있도록 설계하게 되었습니다. 평소 실습을 진행할 때는 큰 틀이 짜여서 나오기 때문에 크게 어려움없이 실습을 진행하였지만 이번 실습 같은 경우 거의 모든 logic과 연결을 스스로 생각 해야 했기 때문에 어려움이 컸습니다.

B. 결론

실습 자료에서 설명되어 있는 계산 법은 radix 2를 사용한 multiplier의 계산 방법입니다. 처음 multiplier에 대하여 설명을 들었을 때 단순히 곱셈이 되는 것이 아니라 여러 개의 Adder가 쓰여 합산한 결과가 곱셈이 된 결과라는 것을 알게 되었으며 곱셈기는 다른 로직에 비해 비용이 많이 든다는 것을 알게 되었습니다. 이런 개념을 바탕으로 비용을 줄일 수 있는 booth multiplier를 배우게 되었으며 해당 multiplier는 radix의 경우에 따라 연산이 다르다는 것을 배우게 되었습니다. 실습에서 배우게 된 Radix 2의 경우 일반적으로 계산하는 multiplier와 cycle이 같기 때문에 그렇게 효율적이라고 할 수 없습니다. 이번 실습에서 **본인은 radix 4를 사용하여 multiplier를 구현하였으며 radix 2에 비해 cycle이 반으로 줄게 되어 계산속도와 그만큼 adder를 덜 쓰기 때문에 비용이 더 적게 드는 장점이 있습니다.** 또한 구현한 multiplier의 특징으로는 해당 승수와 비 승수의 연산이 시작되면 중간에 다른 값이 들어와도 해당 연산을 무조건 끝내야 다음 연산이 가능 하도록 설계되어 있으며 연산 도중 op_clear와 reset이 들어오게 되면 0으로 동기화되는 타이밍만 다를 뿐 다시 op_clear와 reset의 명령이 없어지면 바로 연산을 시작할 수 있도록 설계되어 있습니다. 위 사항을 종합적으로 응용하여 생각했을 때 향후 진행하게 될 프로젝트에서는 radix를 조금 더 늘려 (radix 16 예정) multiplier의 연산 속도와 비용을 최대한 줄여서 설계할 예정입니다.

6. 참고문헌

김영민/Booth Multiplication/광운대학교/2018

김영민/Finite State Machine/2018

Radix 4의 계산 방법/https://www.youtube.com/watch?v=db9-g_nnZSE&t=414s