

컴퓨터 공학 기초 실험2 보고서

실험제목: Multiplier

실험일자: 2022년 11월 8일 (화)

제출일자: 2022년 11월 21일 (월)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

Multiplier

B. 목적

Multiplier에 대해 이해한다. Binary Multiplication에 대해 이해하고, Booth Multiplication과 비교해본다. 더 나아가 Booth Multiplication에 대해 이해하고 이를 이용하여, Multiplicand(피승수)와 Multiplier(승수)의 곱을 계산해 결과값을 출력하는 프로그램을 설계한다.

2. 원리(배경지식)

- Binary Multiplication

$$\begin{array}{r} \text{A } 0010(2) \text{ Multiplicand} \\ \times 0110(6) \text{ Multiplier} \\ \hline 0000 \\ 0010 \\ 0010 \\ 0000 \\ \hline 0001100(12) \end{array}$$

Partial products

Binary Multiplication은 승수의 값이 0인 경우 0을, 1인 경우 피승수에 해당하는 값을 shift해 bit수에 맞춰 적어준다. 그 후 모든 값을 더해주어 곱셈의 결과를 얻는다. 위의 사진에서는 0010(2)과 0110(6)을 곱해 0001100(12)이라는 값을 얻은 것을 확인할 수 있다. 하지만 bit 수가 많은 계산의 경우 계산과정이 길어지며 비효율적이다.

- Booth Multiplication

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

위 표는 radix-2의 Booth Multiplication의 규칙이다.

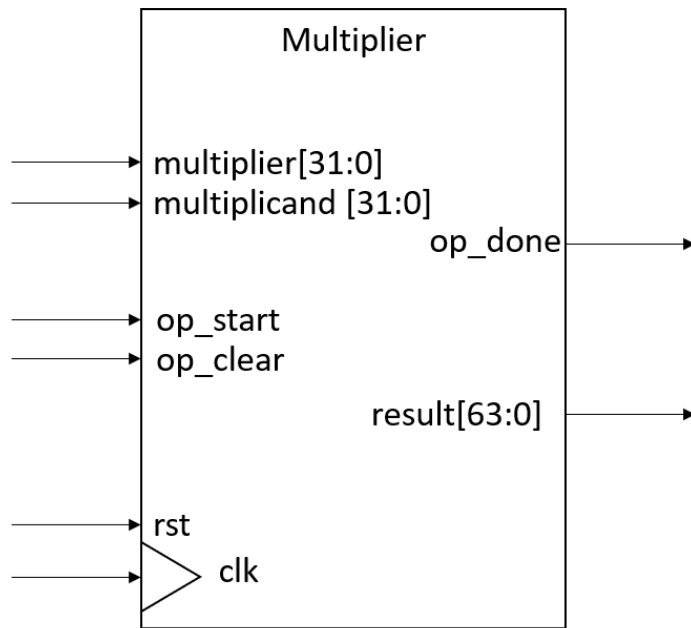
A	0010(2) Multiplicand
X	0110(6) Multiplier
Y	10 $\bar{1}$ 0
shift	00000 ($y_0 = 0$)
Add -A	+1110 ($y_1 = \bar{1}$)
	11100
Shift	111100
Shift	111100 ($y_2 = 0$)
Add A	+0010
	0001100
shift	00001100 (12) ($y_3 = 1$)

Booth Multiplication은 signed binary number의 곱을 산출하는 곱셈기이다. 위의 사진은 0010(2)와 0110(6)의 연산과정을 나타낸 것이다. 먼저 Y값은 승수 즉 X에 LSB의 오른쪽에 0이 있다고 가정한다. 00은 0, 10은 -1, 11은 0, 01은 1이므로 Y는 10(-1)0이다. 먼저 Y가 0일 때 shift이므로, 0000에서 shift해 00000이 된다. 다음으로 Y가 1일 때 subtract shift를 진행하면 subtract는 0010->1110이고, 00000과 더해주면 11100이 된다. 그 후 shift 해주면 111100이된다. Y가 0일 때 shift이므로 1111100이며, 마지막으로 Y가 1일 때 add shift를 진행하면 1111100에 A를 더해주면 0001100이고, shift해주면, 00001100이 된다. 00001100은 12이므로 알맞은 결과가 나온 것을 알 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

위 표는 radix-4의 Booth Multiplication의 규칙이다.

3. 설계 세부사항



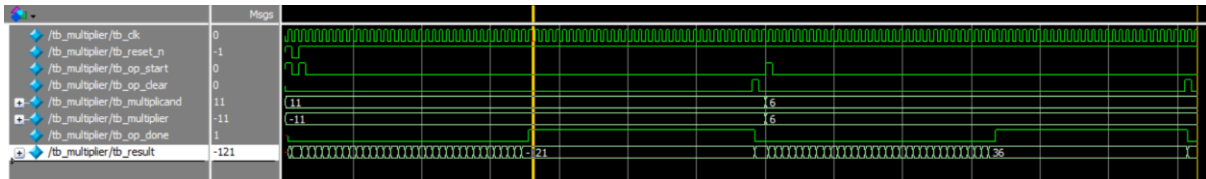
구분	이름	설명
Input	Clk	clock
	Reset_n	Active low reset 신호
	Multiplier[63:0]	승수
	Multiplicand[63:0]	피승수
	Op_start	Start operation
	Op_clear	Clear operation
Output	Op_done	Done operation
	Result[127:0]	Multiplier result

Radix 4를 이용해 multiplier을 구현하였다. Input으로 clk, reset_n, multiplicand, multiplier, op_start, op_clear을, output으로 op_done, result를 가진다. Multiplier_, m_multiplier, m_multiplicand, 그리고 temp, data_count를 reg로 선언해주었고 add, add2, sub, sub2를 wire로 추가해서 선언해주었다. X , X_{i-1} , X_{i-2} 를 비교해서 Y 값을 추출해주고, radix-4의 Booth Multiplication의 규칙을 8가지 상태로 정의 해주어 사용했다. Add, add2, sub, sub2는 assign문을 사용하여 나타냈다. Add와 add2는 1일 때 더 해야 할 값을 계산하며, sub와 sub2는 $\bar{1}$ 일 때 빼야 할 값을 계산한다.

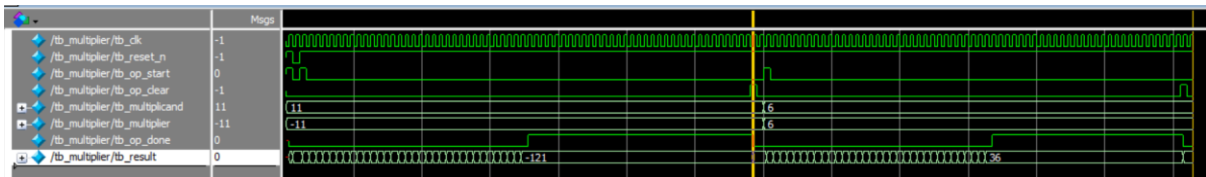
clk의 rising edge나 reset_n의 falling edge에서 reset_n이 0이면 output을 0으로 만든다. op_start의 입력 값이 1'b1이 될 때(즉 op_start는 1, op_clear는 0일 때) multiplier와 multiplication의 입력을 이용해 곱셈을 시작한다. Op_start가 0이고, op_clear가 0일 때 연산을 진행한다. 곱셈기의 연산이 완료될 경우, 즉 data_count가 32가 될 경우 result pin으로 값이 출력되며 op_done 신호가 1'b1로 출력된다. 또한 op_clear의 입력이 들어오기 전까지 이 신호를 유지한다. 연산을 진행되는 도중, op_clear 신호가 들어올 경우 곱셈을 멈추고 모든 output 및 내부에 존재하는 register 들의 값을 logical zero로 초기화 해 주었다.

4. 설계 검증 및 실험 결과

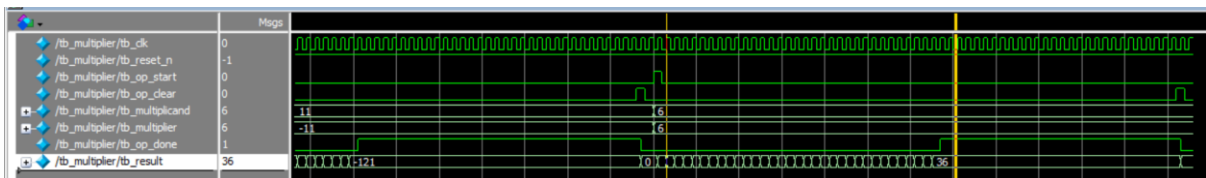
A. 시뮬레이션 결과



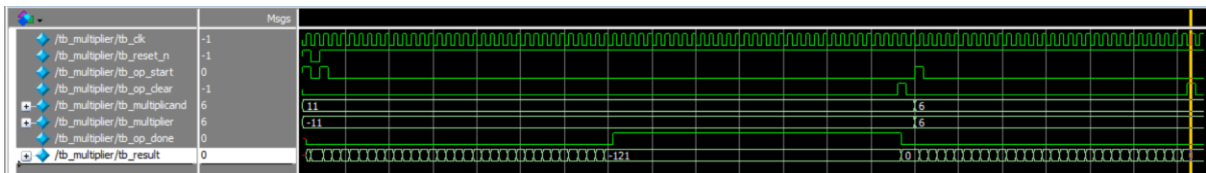
11*(-11)와 6*6의 예시를 들어 testbench를 작성하였다. 먼저 tb_result의 값이 32번 바뀌며 32번의 계산 과정을 거친 후 즉 tb_op_done이 1일 때, 11*(-11)의 결과 값인 -121이 되는 것을 확인할 수 있다.



tb_op_clear가 1일 때 tb_op_done과 tb_result가 0이 되는 것을 확인할 수 있다.

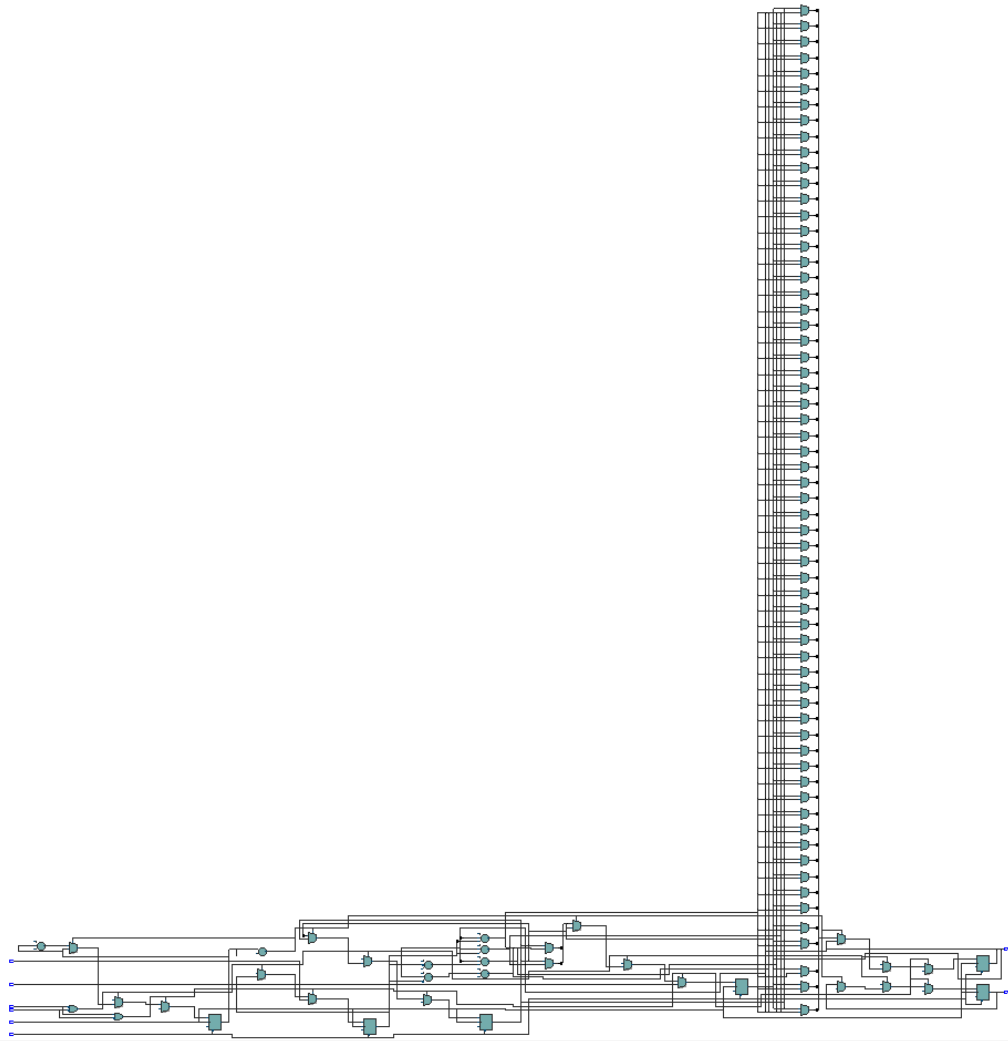


6*6의 연산에서는 tb_op_done이 1일 때, 6*6의 결과 값인 36이 되는 것을 확인할 수 있다.




계산 과정이 끝나기 전에 tb_op_clear가 1이면 tb_op_done과 tb_op_result가 0이 되는 것을 확인 할 수 있다.

B. 합성(synthesis) 결과



multiplier의 rtl viewer로, clk, reset_n, multiplier, multiplicand, op_start, op_clear가 입력되어 op_done, result가 출력되는 것을 확인할 수 있으며, multiplier하는 과정을 rtl viewer을 통해 잘 설계된 것을 확인할 수 있다.

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Mon Nov 21 15:15:44 2022
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	199
Total pins	261
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

multiplier의 flow summary이다. 총 199개의 register와 261개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

처음 개념이 이해가 잘 되지 않아 강의를 반복해서 보고 직접 계산도 해보는 과정을 거쳤다. Radix2로 곱셈을 진행한 후 Radix4로 바꾸어 코드를 짰다. Radix4의 경우 강의자료만 보고 이해하는 것이 쉽지 않아 구글링도 해보고 동영상도 찾아보았다. 교수님께서 주신 강의자료에 있는 계산기를 통해 Radix4 곱셈을 하는 법을 이해할 수 있었다. 코드를 짜고 연산이 잘 진행되는지 확인하기 위해 testbench를 통해 확인하는 과정을 반복하며 오류를 고쳤다. 그 중 bit수를 맞춰주어야 하는 점이 가장 어렵고 오래 걸렸던 것 같다.

B. 결론

4radix를 통해 곱셈기를 구현하였다. 4radix는 2radix일 때 보다 state가 4에서 8로 증가하였지만, cycle수를 절반으로 줄일 수 있다는 장점이 있다. 또한 예를 들어 001010101을 2radix로 recoding할 경우 0 1 (-1) 1 (-1) 1 (-1) 1 (-1)이 되어 연산이 복잡해지지만, radix4

를 사용하면 위의 단점을 보완해줄 수 있으며, 연산의 개수가 늘어나는 경우가 있더라도, 그 개수가 적다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료
공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료