6주차 결과보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20191629 이름: 이주헌

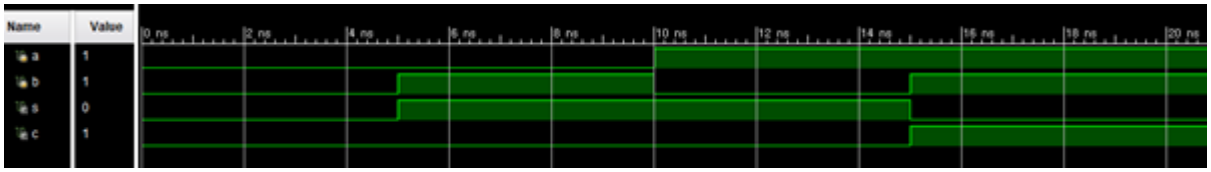
1. 실험 목적

가산기와 감산기, 부호 변환기의 개념을 이해하고, Verilog를 사용하여 해당 회로를 구현할 수 있다.

2. Full-adder 및 Half-adder의 simulation 결과 및 과정에 대해서 설명하시오.

먼저 입력 신호가 더 적은 반가산기를 먼저 구현하였다. 반가산기의 경우, 비트 두 개를 입력받아 그 결과값과 받아올림 비트를 설정하는 회로이다. 반가산기의 진리표는 다음과 같다.

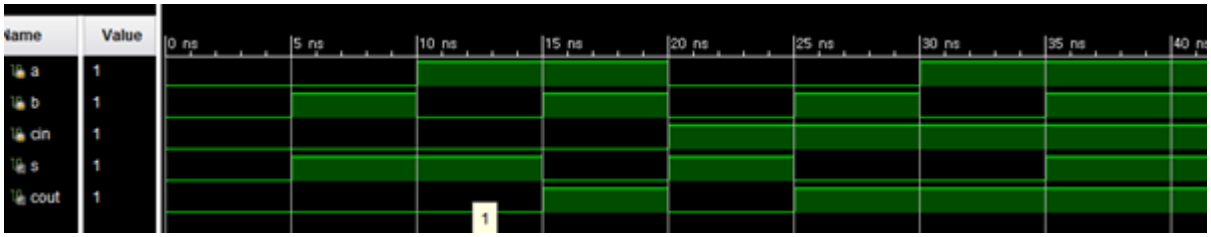
A	B	S	C
T	T	F	T
T	F	T	F
F	T	T	F
F	F	F	F



결과 비트의 경우는 두 입력이 다를 경우에만 논리적 참, 받아올림 비트는 두 입력이 모두 참일 경우에만 참이다. 즉, S = A XOR B, C = A AND B이다. 시뮬레이션 결과 또한 올바르게 나왔다.

다음은 전가산기인데, 전가산기는 A와 B 두 입력 이외에도 받아올림 비트를 입력받을 수 있다. 전가산기의 진리표는 다음과 같다.

C_in	A	B	S	C_out
T	T	T	T	T
T	T	F	F	T
T	F	T	F	T
T	F	F	T	F
F	T	T	F	T
F	T	F	T	F
F	F	T	T	F
F	F	F	F	F

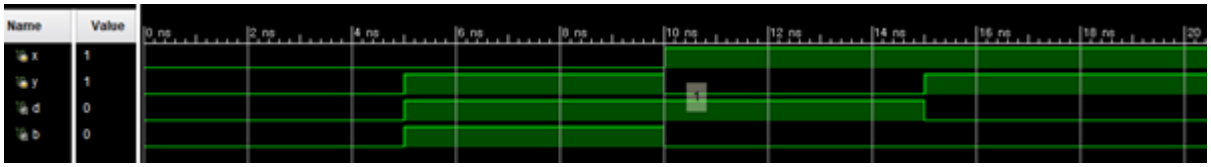


결과 비트의 경우 세 입력 중 논리적 참의 개수가 홀수일 때만 참이고, 받아올림 비트의 경우 세 입력 중 논리적 참의 개수가 2개 이상이면 참이다. 이를 논리식으로 나타내면 S = A XOR B XOR C_in, C_out = A AND B OR (A XOR B) AND C_in으로 표현할 수 있다.

3. Full-subtractor 및 Half-subtractor의 simulation 결과 및 과정에 대해서 설명하시오.

먼저 입력 신호가 더 적은 반감산기를 먼저 구현하였다. 반감산기의 경우, 비트 두 개를 입력받아 그 결과값과 받아내림 비트를 설정하는 회로이다. 반감산기의 진리표는 다음과 같다.

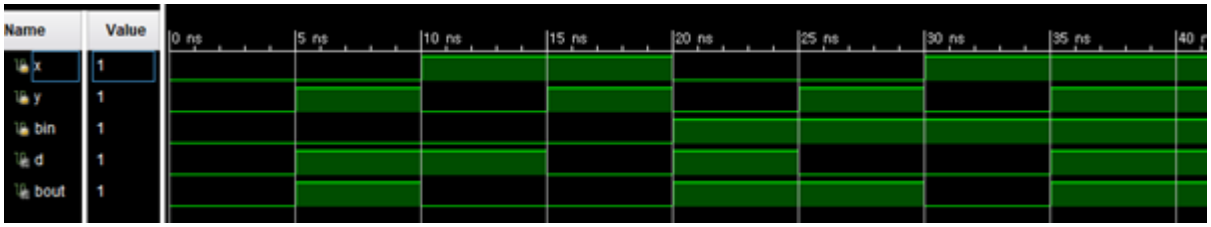
X	Y	D	B
T	T	F	T
T	F	T	F
F	T	T	F
F	F	F	F



결과 비트의 경우는 두 입력이 다를 경우에만 논리적 참, 받아내림 비트는 첫 입력이 거짓이고, 다른 입력이 참일 경우에만 참이다. 즉, D = X XOR Y, B = NOT X AND Y이다. 시뮬레이션 결과 또한 올바르게 나왔다.

다음은 전감산기인데, 전감산기는 A와 B 두 입력 이외에도 받아내림 비트를 입력받을 수 있다. 전감산기의 진리표는 다음과 같다.

B_in	X	Y	D	B_out
T	T	T	T	T
T	T	F	F	F
T	F	T	F	T
T	F	F	T	T
F	T	T	F	F
F	T	F	T	F
F	F	T	T	T
F	F	F	F	F



결과 비트의 경우 세 입력 중 논리적 참의 개수가 홀수일 때만 참이고, 받아내림 비트의 경우 첫 입력이 다른 두 입력을 합한 것보다 작으면 참이다. 이를 논리식으로 나타내면 D = X XOR Y XOR B_in, B_out = NOT X AND Y OR NOT (X XOR Y) AND B_in으로 표현할 수 있다.

4. 8421(BCD)-2421 code converter simulation 결과 및 과정에 대해서 설명하시오.

8421 코드를 W, X, Y Z로 입력받아 2421 코드 A, B, C, D로 변환하는 과정의 진리표는 다음과 같다.

W	X	Y	Z	A	B	C	D
F	F	F	F	F	F	F	F
F	F	F	T	F	F	F	T
F	F	T	F	F	F	T	F
F	F	T	T	F	F	T	T
F	T	F	F	F	T	F	F
F	T	F	T	T	F	T	T
F	T	T	F	T	T	F	F
F	T	T	T	T	T	F	T
T	F	F	F	T	T	T	F
T	F	F	T	T	T	T	T
T	F	T	F				
T	F	T	T				
T	T	F	F				
T	T	F	T				
T	T	T	F				
T	T	T	T				

8421 표기법의 정의에 따라, 0부터 9까지의 수만 표기할 수 있기 때문에 10~15 사이의 결과값은 동작에 영향을 주지 않는다. 따라서 위 진리표에서는 셀을 어둡게 칠하여 don’t care condition임을 표현했다.

이를 바탕으로 다음과 같이 각 출력 비트마다 카르노 맵을 구성할 수 있다.

		WX			
		00	01	11	10
	00	0	0		1

YZ	01	0	1		1
	11	0	1		
	10	0	1		

Output A

		WX			
		00	01	11	10
YZ	00	0	1		1
	01	0	0		1
	11	0	1		
	10	0	1		

Output B

		WX			
		00	01	11	10
YZ	00	0	0		1
	01	0	1		1
	11	1	0		
	10	1	0		

Output C

	WX			
	00	01	11	10

YZ	00	0	0		0
	01	1	1		1
	11	1	1		
	10	0	0		

Output D

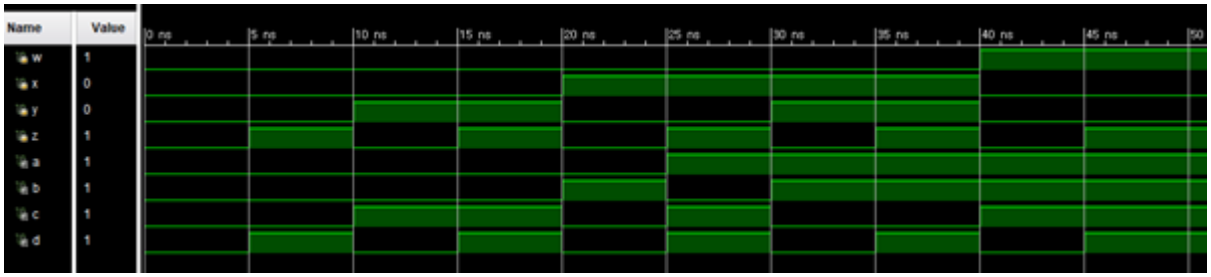
위 카르노 맵을 사용하면 A, B, C, D 출력 신호를 모두 SOP 또는 POS form으로 최소화할 수 있다.

- $A = W + XY + XZ = (W + X)(W + Y + Z)$
- $B = W + XY + X\neg Y\neg Z = (W + X)(X + Y + \neg Z)$
- $C = W + \neg XY + X\neg YZ = (W + X)(W + Y + Z)(\neg X + \neg Y)$
- $D = Z$

위 식은 NAND와 NOR 논리회로로 구성할 수 있는데, 이를 Verilog로 구현하면 다음과 같다.

```
// NAND logic
assign a = ~(~w & ~(x & y) & ~(x & z)
assign b = ~(~w & ~(x & ~y & ~z) & ~(x & y))
assign c = ~(~w & ~(~x & y) & ~(x & ~y & z))
assign d = z

// NOR logic
assign a = ~(~(w | x) | ~(w | y | z))
assign b = ~(~(w | x) | ~(x | y | z))
assign c = ~(~(w | x | y) | ~(w | y | z) | ~(~x | ~y))
assign d = z
```



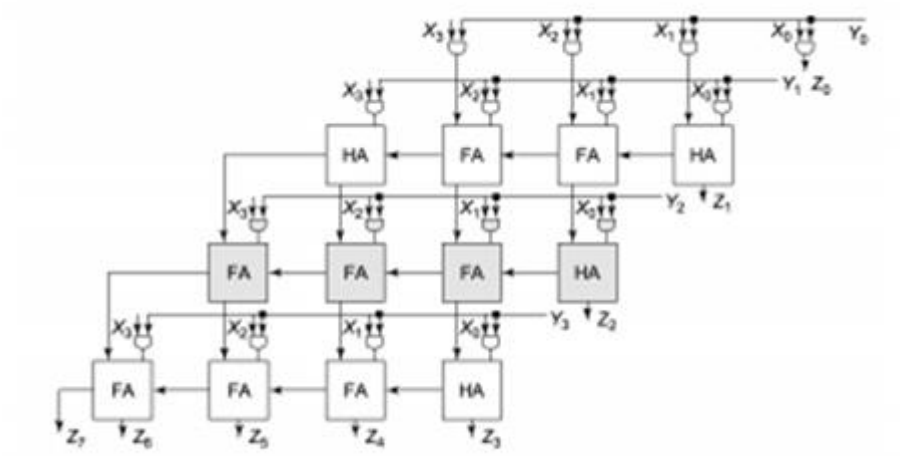
위 코드를 실행한 시뮬레이션 결과가 진리표의 것과 일치하기 때문에, 위에서 작성한 NAND 로직이 정확하다는 것을 알 수 있다.

5. 결과 검토 및 논의사항

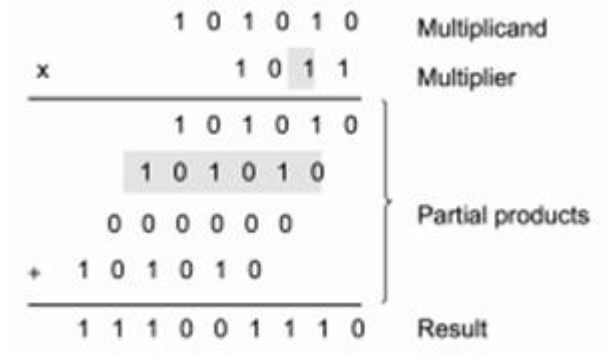
이번 실험에서는 반가산기, 전가산기, 반감산기, 전감산기를 모두 Verilog으로 구현하면서 프로세서가 어떻게 숫자를 계산하는지 알 수 있었다. 또한, 더욱 복잡한 입력과 출력값을 갖는 8421-2421 변환기를 만들면서 카르노 맵을 그려 원하는 진리표로부터 논리회로를 설계하는 실습을 할 수 있었다. 또한, 회로를 NAND와 NOR 게이트만을 사용하여 구현함으로서 NAND와 NOR의 범용성을 재확인할 수 있었다.

6. 추가 이론 조사 및 작성

가산기를 여러 개 묶어서 사용하면 두 값을 곱하는 승산기도 만들 수 있다.



해당 방식은 초등학생 때 공부한 곱셈을 세로셈으로 계산하는 과정과 비슷하게 모델링한 것이라고 볼 수 있다. 회로를 알아보기 쉽게 예를 들어 보면 위 과정은 다음과 같다.



하지만 보통 범용 CPU에서 곱셈을 계산할 때는 x86 어셈블리의 mul 명령과 같이 카운터 레지스터를 감소시키며 루프를 돌리는 경우가 많다.