

6주차 예비보고서

전공: 신문방송학과

학년: 4학년

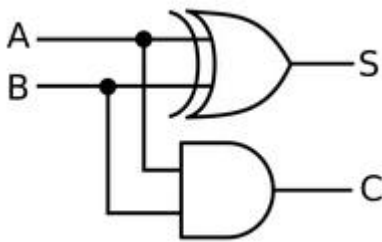
학번: 20191150

이름: 전현길

1. 전가산기 및 반가산기에 대해 조사하시오(예시 포함).

가산기(adder)는 덧셈 연산을 수행하기 위한 논리회로이다. 이 중 전가산기와 반가산기는 1bit 덧셈 연산을 수행하기 위한 논리회로이다. 둘 간의 차이점은 반가산기는 A, B 두 입력에 대한 1bit 덧셈 연산만 수행하지만, 전가산기는 이전 bit의 자리올림수를 포함하여 세 개의 입력에 대해 1bit 덧셈 연산을 수행한다는 점이다.

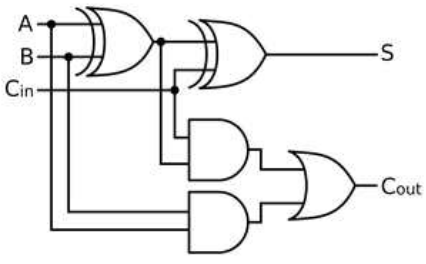
가산기는 디지털 계산, 연산에 광범위하게 사용되는 기초 소자로, 대표적으로 사용되는 예시가 CPU에서 볼 수 있는 32bit, 64bit ALU 가산기이다. ALU 내에 들어 있는 가산기의 경우 복수 bit 덧셈을 수행하기 위해서 전가산기로 구성된다.

반가산기(half adder)					
digital circuit		truth table			
		input		output	
		A	B	Sum	Carry
		0	0	0	0
		0	1	1	0
		1	0	1	0
1	1	0	1		

반가산기는 A와 B 두 입력을 받아서 합(sum)과 자리올림수(carry)를 출력한다. 동작하는 방식은 이진수 덧셈과 같다. 1과 1을 더할 때에는 Sum 1bit만으론 표현할 수 없다는 점이 문제가 되는데, 이 경우 sum은 0을 출력하고 carry는 1을 출력한다. 구현식은 다음과 같다.

$$\text{Sum} = A \oplus B \mid \text{Carry} = A \cdot B$$

구현 원리는 생각보다 복잡하지 않다. 진리표를 확인했을 때 Sum의 동작은 $A \text{ XOR } B$ 와 동일한 동작을 수행하므로 $A \text{ XOR } B$ 로 구현한다. 마찬가지로 Carry의 동작은 $A \text{ AND } B$ 와 동일한 동작을 수행하므로 AND 회로로 구현한다.

전가산기(full adder)					
digital circuit			truth table		
			input		output
			A	B	C _{in}
			Sum	C _{out}	
			0	0	0
			0	0	1
			0	1	0
			0	1	1
			1	0	0
			1	0	1
			1	1	0
			1	1	1

전가산기는 A와 B, C_{in} 세 입력을 받아서 **합(sum)**과 **자리올림수(carry)**를 출력한다. 따라서 1bit 입력 A + B + C_{in}을 모두 더한 뒤 적절한 합과 자리올림수를 출력해야 한다. 모든 입력이 0이라면 Sum/Carry 0/0을 출력하고, 한 개의 입력이 1이라면 1/0, 두 개의 입력이 1이라면 0/1, 세 개의 입력이 1이라면 1/1을 출력한다.

Sum과 C_{out}의 간소화된 이진함수식을 구하기 위해 카르노 맵과 불 대수 정리를 사용할 수 있으며 해당 과정은 보고서가 너무 길어지기 때문에 생략했다. 결과식은 다음과 같다.

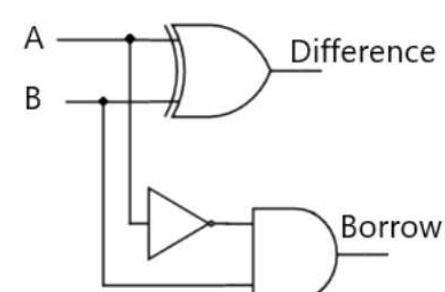
$$\text{Sum} = A \oplus B \oplus C_{in} \mid C_{out} = C_{in} \cdot (A \oplus B) + A \cdot B$$

위 디지털 회로의 형태를 자세히 살펴보면 반가산기 2개와 OR 회로 1개로 구성된 것을 알 수 있다. 즉 전가산기는 반가산기로 구성된다.

2. 전감산기 및 반감산기에 대해 조사하시오(예시 포함).

감산기(subtractor)는 뺄셈 연산을 수행하기 위한 논리회로이다. 이 중 전감산기와 반감산기는 1bit 덧셈 연산을 수행하기 위한 논리회로이다. 둘 간의 차이점은 반가산기는 A, B 두 입력에 대한 1bit 덧셈 연산만 수행하지만, 전가산기는 이전 bit의 빌림수(borrow)를 포함하여 세 개의 입력에 대해 1bit 덧셈 연산을 수행한다는 점이다.

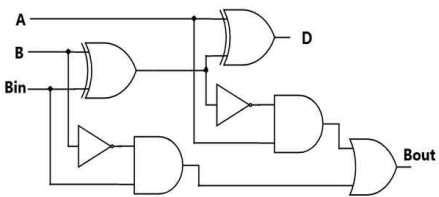
감산기가 사용되는 대표적인 예시 역시 컴퓨터의 ALU로, 가산기와 유사하게 복수 bit 뺄셈을 계산해야 하기 때문에 전감산기를 사용한다. 이외에 디지털 계산기, 디지털 신호 처리, 이미지 처리 및 음성 신호 처리 등의 다양한 분야에서 전감산기를 사용할 수 있다.

반감산기(half subtractor)					
digital circuit		truth table			
		input		output	
		A	B	Diff	B _{out}
		0	0	0	0
		0	1	1	1
		1	0	1	0
		1	1	0	0

반가산기는 A와 B 두 입력을 받아서 차(difference; D)와 빌림(borrow; B_{out})을 출력한다. 동작하는 방식은 2의 보수 뺄셈 A-B와 같다. A, B가 각각 0/0, 1/1이면 빌리지 않고, 차도 0이므로 Diff/B_{out} 각각 0/0을 출력한다. A가 1, B가 0일 때는 차는 1이고 빌리지도 않으므로 1/0을 출력하며, A가 0, B가 1일 때는 값을 빌려 뺄셈을 수행하므로 1/1을 출력한다.

$$\text{Difference}(D) = A \oplus B \quad | \quad \text{Borrow}(B_{\text{out}}) = A \cdot B$$

구현 원리는 진리표에 따라 Diff의 동작은 A XOR B와 동일한 동작을 수행하므로 A XOR B로 구현한다. 마찬가지로 B_{out}의 동작은 앞서 1bit 비교기에서 보았던 (NOT A) AND B와 같으므로 동일하게 구현한다. 반가산기와 반감산기는 inverter 1개가 붙은 것 외에는 형태가 동일하므로 반가산기 하나와 인버터 하나를 통해 반감산기를 구현할 수 있다.

전감산기(full subtractor)				
digital circuit	truth table			
	input			output
	A	B	B _{in}	Diff B _{out}
	0	0	0	0
	0	0	1	1
	0	1	0	1
	0	1	1	0
	1	0	0	1
	1	0	1	0
	1	1	0	0
	1	1	1	1

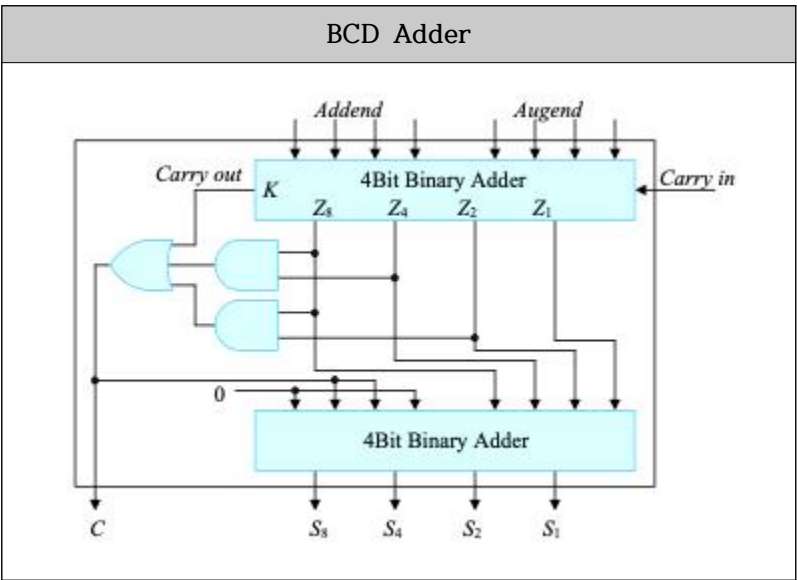
전감산기는 A와 B, B_{in} 세 입력을 받아서 차(difference; D)와 빌림수(borrow; B_{out})를 출력한다.. 따라서 1bit 입력 A - B - B_{in}의 식과 동일하게 동작하고, 이에 따라 적절한 차와 빌림수를 계산한다. 반가산기처럼 식의 결과에 따라 값을 빌려와야 한다면 B_{out}이 1이 되고, 그렇지 않으면 0이 된다.

Diff과 B_{out}의 간소화된 이진함수식을 구하기 위해 카르노 맵과 불 대수 정리를 사용할 수 있으며 해당 과정은 보고서가 너무 길어지기 때문에 생략했다. 결과식은 다음과 같다. Diff의 논리식이 전가산기에서 Sum의 논리식과 동일한 것을 확인할 수 있다.

$$\text{Diff} = A \oplus B \oplus B_{in} \mid \text{Carry} = C \cdot (\overline{A \oplus B}) + \overline{A} \cdot B$$

위 디지털 회로의 형태를 자세히 살펴보면 반감산기 2개와 OR 회로 1개로 구성된 것을 알 수 있다. 즉 전감산기는 반감산기로 구성된다. 뿐만 아니라, 반가산기에 inverter를 추가하면 반감산기를 구현할 수 있으므로 반가산기와 인버터, OR 게이트를 잘 연결하여 전감산기를 구현할 수 있다.

3. BCD 가산기에 대해 조사하시오.



이진화 십진법(BCD; Binary-Coded Decimal)은 이진수 4자리를 묶어서 십진수 1자리를 표현하는 기법이다. BCD 코드에서 0~19는 다음처럼 표현한다.

10진수	2진수	BCD (8421코드)		10진수	2진수	BCD (8421코드)	
0	0000	0000	0000	10	1010	0001	0000
1	0001	0000	0001	11	1011	0001	0001
2	0010	0000	0010	12	1100	0001	0010
3	0011	0000	0011	13	1101	0001	0011
4	0100	0000	0100	14	1110	0001	0100
5	0101	0000	0101	15	1111	0001	0101
6	0110	0000	0110	16	10000	0001	0110
7	0111	0000	0111	17	10001	0001	0111
8	1000	0000	1000	18	10010	0001	1000
9	1001	0000	1001	19	10011	0001	1001

확인할 수 있듯이, BCD 코드는 2진수 계산의 원리를 따르지 않기 때문에 BCD 덧셈은 다음과 같은 특별한 계산법을 따른다.

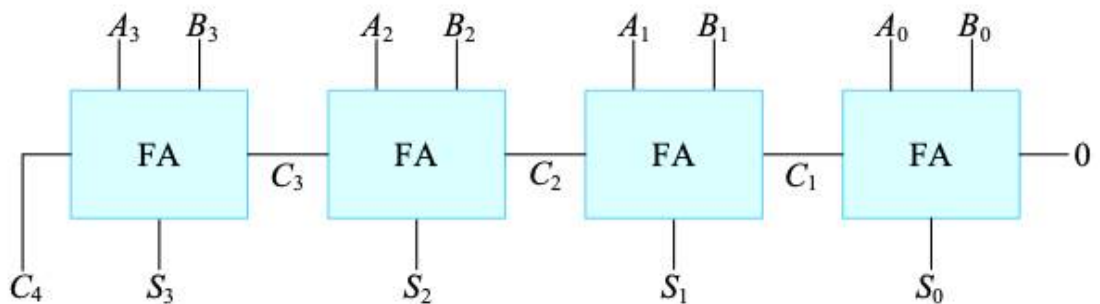
- 1) BCD 숫자를 4bit 단위로 더한다.
- 2) 4bit의 값이 9보다 크거나 캐리가 발생하면 계산 값에 0110(6)을 더한다.
- 3) 0110을 더했을 때 캐리가 발생하면 자리올림한다.

6을 더하는 이유는 사용하지 않는 6개의 코드의 영역(10~15)을 벗어나서 자리올림하기 위해서이다. BCD 가산기는 위의 계산법을 아래의 진리표를 따라

구현하며, 회로는 위에서 확인할 수 있다.

2진 합					BCD 합					10진값
K	Z ₈	Z ₄	Z ₂	Z ₁	C	S ₈	S ₄	S ₂	S ₁	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

4. 병렬 가감산기에 대해 조사하시오.



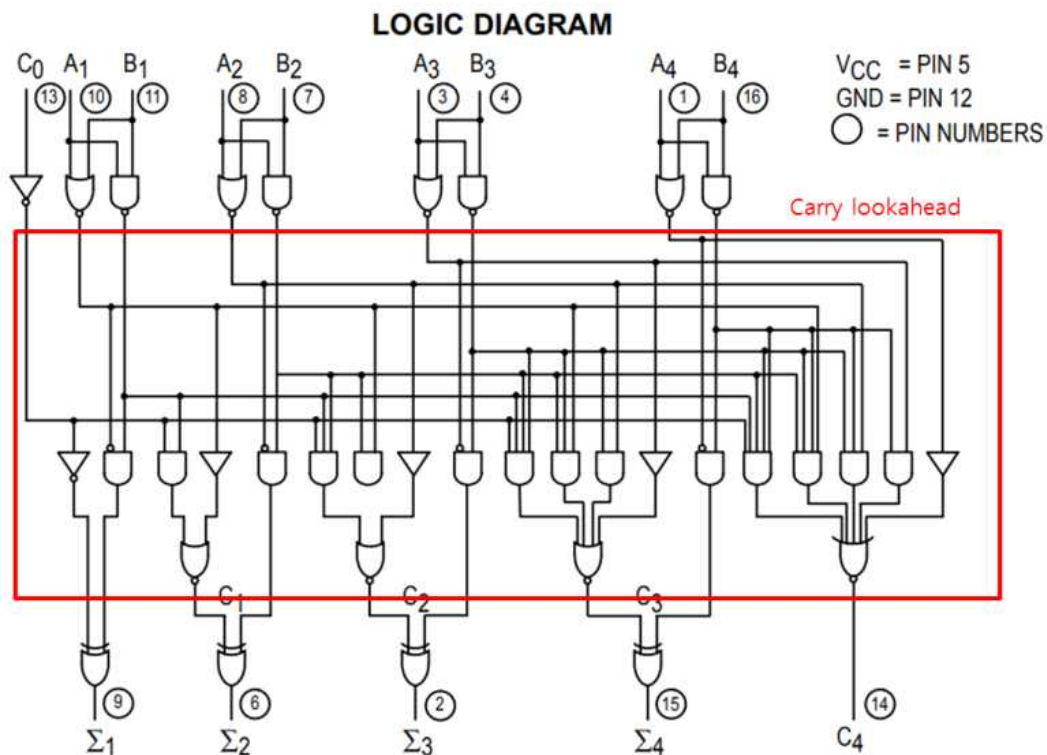
병렬 가감산기는 전가산기, 전감산기 여러 개를 병렬로 배치한 회로이다. 위에서 살펴보았던 자리올림수, 빌림수를 입력으로 받은 뒤 3개의 입력에 대해 앞서 설명했던 동작을 수행한다. 개별 가산기/감산기를 연결하는 것만으로도 4bit, 8bit, 16bit, 32bit 연산 등을 얼마든지 수행할 수 있다.

단 첫 번째 연산(가장 작은 자리수)의 경우 자리올림수/빌림수가 없으므로 C_{in} , B_{in} 에 0을 입력한다.

병렬 가산기의 자리올림수/빌림수 값이 마치 파도가 물결치는 것처럼 퍼진다(전송된다)고 해서, 병렬 가산기는 따로 Ripple Carry Adder라는 이름으로 부르기도 한다.

5. Carry Look-Ahead Adder을 Ripple Carry Adder와 비교하여 설명하시오.

Ripple Carry Adder에서 덧셈은 1bit 단위로 연쇄적으로 이뤄진다. 이전 자리의 자리올림수, 빌림수가 출력되어야 다음 자리의 C_{in} , B_{in} 입력이 결정되기 때문이다. 즉, 필연적으로 기다리는 시간이 발생한다. 때문에 4bit, 8bit 등 덧셈의 단위가 커지면 커질수록 그에 따라 선형적인 연산 지연이 발생한다.



CLA adder에서는 그렇지 않다. CLA Adder에서는 carry에 대한 식을 k-bit(주로 4bit) 단위로 끊어서 계산함으로써, k-bit 계산에 쓰이는 carry를 병렬적으로 계산한다. 이를 통해서 n-bit 덧셈에 따른 선형적인 연산 지연($O(n)$)을 이론상 $O(n/k)$ 까지 줄일 수 있다. 이러한 계산은 자리올림수 생성(generatio

n)과 전파(propagation)라는 두 가지 원리를 이용해서 이뤄진다.

$$G(\text{generation}) \rightarrow G(A, B) = A \cdot B$$

$$P(\text{propagation}) \rightarrow P(A, B) = A \oplus B$$

G는 자리올림수 생성(generation)으로, 기존의 전가산기 연산과 관계없이 반드시 carry가 생기는 경우(A, B가 모두 1)를 검사한다. P는 자리올림수 전파(propagation)로, 추가로 carry가 생길 경우를 검사한다(A + B의 결과가 1이라, 이전 bit의 carry와 조합되어 새 carry가 생기는 경우). 간단히 말해 G는 A AND B, P는 A XOR B일 경우에 1이 된다.

G와 P를 사용해 가산기의 Sum과 Carry를 다음처럼 치환할 수 있다.

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

위의 치환 방식을 바탕으로 C₁만을 이용해 C₂, C₃, C₄의 값을 다음처럼 치환할 수 있고, 이 식을 바탕으로 회로를 구성하여 CLA Adder를 구현할 수 있다.

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 \cdot (G_1 + P_1 C_1)$$

$$C_4 = G_3 + P_3 C_3$$

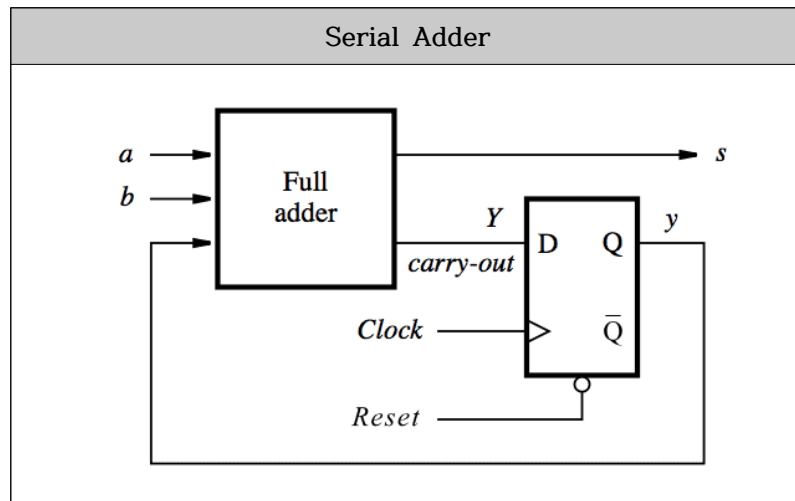
$$= G_3 + P_3(G_2 + P_2 C_2)$$

$$= G_3 + P_3(G_2 + P_2(G_1 + P_1 C_1))$$

위의 치환 방식을 바탕으로 C₁만을 이용해 C₂, C₃, C₄의 값을 다음처럼 치환할 수 있고, 이와 같은 방식을 바탕으로 회로를 구성하여 CLA Adder를 구현할 수 있다. 왼쪽의 회로의 경우 C₀부터 병렬로 연결했으므로 앞서 했던 작업을 C₀부터 반복하면 된다.

6. 기타 이론

내용을 조사하면서 병렬 가산기/감산기는 전가산기/전감산기를 직렬로 연결한 회로임에도 불구하고 왜 parallel adder라고 부르는지 궁금했는데, 전가산기/전감산기 하나로 n-bit의 계산을 수행하는 직렬 가산기/감산기(serial adder/subtractor)를 조사하면서 알 수 있었다.



직렬 가산기, 감산기는 전가산기 하나와 클록 사이클을 바탕으로 구현되며, 클록 사이클당 1bit를 처리한다. 때문에 병렬 가산기보다 훨씬 느리게 동작하지만, 구조가 훨씬 단순하다.