

1bit 전가산기와 4bit 덧셈 연산기 74LS283에서의 정 논리와 부 논리에 대한 분석

정동호*, 정태상*, 유준복
중앙대 전자전기 공학부*, 중앙대 전자전기 공학부, 중앙대 제어계측공학과

Analysis of Positive Logic and Negative Logic in 1bit adder and 4 bit adder 74LS283

Tong-Ho Chung*, Tea-Sang Chung, Jun-Bok You

School of Electrical and Electronics Engineering, Chung-Ang Univ *,

School of Electrical and Electronics Engineering, Dept. of Control & Instrumentation Engineering, Chung-Ang Univ *

Abstract - 1bit full adder have 3 input (including carry_in) and 2 outputs(Sum and Carry_out). Because of 1 bit full adder's propagation delay, We usually use 4-bit binary full adder with fast carry, 74LS283.

The 74LS283 is positive logic circuit chip.

But the logic function of binary adder is symmetrical, so it can be possible to use it not only positive logic but also the negative logic.

This thesis use symmetrical property, such as $C_{i+1}(\bar{a}_i, \bar{b}_i, \bar{c}_i) = \bar{C}_{i+1}(a_i, b_i, c_i)$ and

$$S_i(\bar{a}_i, \bar{b}_i, \bar{c}_i) = \bar{S}_i(a_i, b_i, c_i).$$

And prove this property with logic operation.

Using these property, the 74LS283 adder is possible as the negation logic circuit. It's very useful to use the chip in negative logic, because many system chip is negative logic circuit, for example when we have negative logic chip with 74LS283, we don't need any not gate for 74LS283 input, and just use output of adder(74LS283) as the negation of original output.

1. 서 론

시스템 설계 및 구현에 있어서 덧셈 연산은 어떤 연산보다도 그 사용이 빈번하다. 그렇기에 덧셈 연산기의 그 정확도와 연산 속도는 시스템의 성능에 중요한 영향을 미치므로 덧셈기에 대한 분석은 중요하다.

본 본문에서는 1 bit 전가산기와 빠른 캐리 연산을 수행하는 4 bit 덧셈기인 74LS283칩에 대한 분석을 하고자 한다. 특별히 이 칩들에 대한 디자인을 논리 게이트 레벨로 까지 구현하여, 내부적인 로직 형태와 일반화되어 구현된 회로에 대해 분석을 하고자 한다.

또한 구현된 덧셈기의 2진 합수 연산기의 대칭적인 특성을 활용하여, 1 bit 전가산기와 4bit 전가산기에서 사용하고 있는 정 논리(Positive Logic) 회로 뿐 아니라 시스템 구현에 있어서 많이 사용되어지는, 부 논리(Negative Logic) 회로로의 활용 가능성을 보이고자 한다.

2. 본 론

2.1 1bit 덧셈기의 덧셈 연산

1bit 전가산기에서 있어서 입력은 캐리(c_i)와 1bit 입력 a_i 와 b_i 를 포함한 3bit를 입력을 받아, 합과 캐리 출력을 내 보낸다. 그 진리표를 보면 표 1 과 같음을 볼 수 있다. 그 블록 다이어그램을 보면 그림1과 같다

이것을 표1을 바탕으로 합(Sum) 출력을 민타임들의 합(Sum of Minterm) 형식으로 전개하여 표현하면

$$s_i = \bar{a}_i \bar{b}_i c_i + a_i \bar{b}_i c_i + a_i b_i \bar{c}_i + a_i b_i c_i$$

$$= (\bar{a}_i \bar{b}_i + a_i b_i) c_i + (\bar{a}_i b_i + a_i \bar{b}_i) \bar{c}_i \quad \text{이다.}$$

$$= (\bar{a}_i \oplus b_i) c_i + (a_i \oplus b_i) \bar{c}_i = (a_i \oplus b_i) \oplus c_i$$

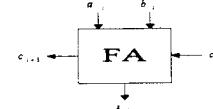


그림1. 1bit 전가산기

또한 캐리 출력도 아래와 같은 Sum of Minterm 형식으로 표현되어질 수 있다.

$$c_{i+1} = \bar{a}_i b_i c_i + a_i \bar{b}_i c_i + a_i b_i \bar{c}_i + a_i b_i c_i$$

$$\begin{aligned} &= \{ a_i b_i + b_i c_i + c_i a_i \\ &= (a_i + b_i) c_i + a_i b_i \\ &= (\bar{a}_i b_i + a_i \bar{b}_i) c_i + a_i b_i = (a_i \oplus b_i) c_i + a_i b_i \end{aligned}$$

표1. 1bit 전가산기 진리표

Input				
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2.1.1 1bit 덧셈기의 논리 회로로의 구현

$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = (a_i \oplus b_i) c_i + a_i b_i$$

이 연산 결과를 이용하여, NAND 게이트 레벨로 회로를 구현하여 보면 그림2와 같이 구현할 수 있다.

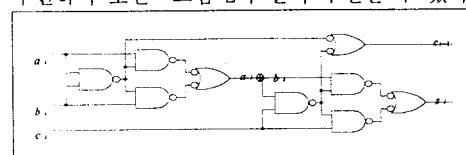


그림2. 2 Input NAND을 이용한 1bit 전가산기

이 회로에서 배타적 OR (XOR)을 구현하기 위해 NAND 회로 4개를 사용하여 구현을 하였다.

XOR을 식을 전개하여 보면

$$\begin{aligned} b_i \bar{a}_i \bar{b}_i + a_i \bar{a}_i b_i &= b_i (\bar{a}_i + \bar{b}_i) + a_i (\bar{a}_i + b_i) \\ &= \bar{a}_i b_i + a_i \bar{b}_i = a_i \oplus b_i \end{aligned}$$

임을 알 수 있다.

2.1.2 1bit 덧셈기의 부 논리 (Negative Logic) 구현

1bit 덧셈기에서의 정 논리 (Positive Logic) 관점에서 캐리 출력에 대한 진리표를 만들어 보면

$c_{i+1} (a_i, b_i, c_i)$ 는 다음과 같다.

		b_i, c_i	00	01	11	10
		a_i	0	0	1	0
a_i	0	0	0	1	1	1
	1	0	1	0	0	0

이것을 부 논리 (Negative Logic) 분석으로 접근하여 출력에 보수를 취한 $\overline{c_{i+1}} (a_i, b_i, c_i)$ 은

		b_i, c_i	00	01	11	10
		a_i	0	1	0	1
a_i	0	1	1	0	1	
	1	1	0	0	0	

입력에 보수를 취한 $c_{i+1} (\overline{a}_i, \overline{b}_i, \overline{c}_i)$

		$\overline{b}_i \overline{c}_i$	11	10	00	01
		\overline{a}_i	1	1	0	1
\overline{a}_i	1	1	1	0	1	1
	0	1	0	0	0	0

임을 볼 수 있다.

즉 보수를 취한 입력값의 캐리 출력은 원래 캐리 출력값에 negation 한 값과 동일하다.

이것을 논리 연산으로 증명하면

$$\begin{aligned} c_i (\overline{a}_i, \overline{b}_i, \overline{c}_i) &= \overline{a}_i \overline{b}_i + \overline{b}_i \overline{c}_i + \overline{c}_i \overline{a}_i \\ &= \overline{(a_i + b_i)(b_i + c_i) + (c_i + a_i)} \\ &= \overline{c_{i+1}(a_i, b_i, c_i)} \end{aligned}$$

합(Sum)의 경우에서도 마찬가지로 $\overline{S_{i+1}} (a_i, b_i, c_i)$ 은

		b_i, c_i	00	01	11	10
		a_i	0	1	0	1
a_i	0	1	0	1	0	0
	1	0	1	0	1	0

이고,

입력에 보수를 취한 $S_{i+1} (\overline{a}_i, \overline{b}_i, \overline{c}_i)$

		$\overline{b}_i \overline{c}_i$	11	10	00	01
		\overline{a}_i	1	1	0	1
\overline{a}_i	1	1	0	1	0	0
	0	0	1	0	1	0

임을 볼 수 있다.

이것을 논리 연산 식으로 전개하면

$$\begin{aligned} s_i(a_i, b_i, c_i) = a_i \oplus b_i \oplus c_i \text{ 일 때} \\ s_i(\overline{a}_i, \overline{b}_i, \overline{c}_i) = \overline{a}_i \oplus \overline{b}_i \oplus \overline{c}_i \\ = \overline{(a_i \oplus b_i \oplus c_i)} \\ = \overline{s_i(a_i, b_i, c_i)} \end{aligned}$$

즉 합 연산에서도 1bit 덧셈 연산에서 Positive Logic의 출력을 보수를 취한 것(Negation)은 원래 함수의 입력에 보수를 취해 연산되어 나온 출력 값과 같다.

2.2 4bit 덧셈기 74283의 연산 원리

74LS283은 4-bit Carry-Look-Ahead 덧셈기로서 1bit 덧셈기를 4개 직렬로 연결해서 사용함으로 발생하는 전달 지연 시간 (Propagation Delay Time)이 1bit 덧셈 수행할 때 보다 4배가 되어지는 것을 극복하고 지연시간을 줄이도록 설계되어 있다.

즉 하위 bit의 캐리 연산을 기다리지 않고, 각 bit 별로 캐리를 만들고, 동시에 덧셈 연산을 수행한다.

2.2.1 4bit 덧셈기 74LS283의 캐리 루 어헤드 (Carry Look Ahead) 와 bit별 덧셈 연산

74LS283에서의 캐리 발생기(Carry Generator g_i) 와 캐리 전달기 (Carry Propagator p_i)를 각 bit 별로 연산한다.

$$\begin{aligned} g_i &= a_i b_i \\ p_i &= a_i + b_i \quad i = 0, 1, \dots, n-1 \end{aligned}$$

캐리 표현을 g_i 와 p_i 을 이용해서 표현하면

$$c_{i+1} = \begin{cases} a_i b_i + (a_i + b_i)c_i & = g_i + p_i c_i \\ (a_i + b_i)(a_i b_i + c_i) & = p_i(g_i + c_i) \end{cases}$$

곱의 합(Sum of Products) 형식으로 표현된 것을 선택하면 $c_{i+1} = g_i + p_i c_i$ 이고 각 bit 별로 대입하면 $c_1 = g_0 + p_0 c_0$

$$c_2 = g_1 + p_1 c_1 = g_1 + p_1(g_0 + p_0 c_0)$$

$$= g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_0)$$

$$= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 g_2$$

$$= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0)$$

$$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

$$= G_0 + P_0 c_0$$

$$\begin{cases} G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \\ P_0 = p_3 p_2 p_1 p_0 \end{cases}$$

합의 곱 (Product of Sums) 표현식은

$c_{i+1} = p_i(g_i + c_i)$ 이고 이 또한 각 bit 별로 표현하면

$$c_1 = p_0(g_0 + c_0)$$

$$c_2 = p_1(g_1 + c_1) = p_1[g_1 + p_0(g_0 + c_0)]$$

$$= p_1(g_1 + p_0)(g_1 + g_0 + c_0)$$

$$c_3 = p_2(g_2 + c_2)$$

$$= p_2[g_2 + p_1(g_1 + p_0)(g_1 + g_0 + c_0)]$$

$$= p_2(g_2 + p_1)(g_2 + g_1 + p_0)(g_2 + g_1 + g_0 + c_0)$$

$$\begin{aligned}
 c_4 &= p_3(g_3 + c_3) \\
 &= p_3[g_3 + p_2(g_2 + p_1)(g_2 + g_1 + p_0)(g_2 + g_1 + g_0 + c_0)] \\
 &= P_0(G_0 + c_0)
 \end{aligned}$$

$$G_0 = g_3 + g_2 + g_1 + g_0$$

$$P_0 = p_3(g_3 + p_2)(g_3 + g_2 + p_1)(g_3 + g_2 + g_1 + p_0)$$

이렇게 계산한 각 캐리 bit과 각 비트당 생성한 g_i 와 p_i 을 사용하여 합(Sum) 출력을 계산한다.

$$\begin{aligned}
 a_i \oplus b_i &= \overline{a_i}b_i + a_i\overline{b_i} = (a_i + b_i)(\overline{a_i} + \overline{b_i}) \\
 &= (a_i + b_i)\overline{a_i b_i} = p_i \overline{g_i} \text{ 이므로}
 \end{aligned}$$

$$s_i = (a_i \oplus b_i) \oplus c_i = (\overline{g_i}p_i) \oplus c_i \text{이고 각 bit 별로}$$

$$s_0 = (\overline{g_0}p_0) \oplus c_0$$

$$s_1 = (\overline{g_1}p_1) \oplus c_1$$

$$s_2 = (\overline{g_2}p_2) \oplus c_2$$

$$s_3 = (\overline{g_3}p_3) \oplus c_3 \text{이다.}$$

이렇게 구현된 74LS283 덧셈기의 논리 회로는 그림 3과 같다.

2.2.2 4bit 덧셈기 74283의 부 논리 회로 (Negation Logic)

캐리 출력을 캐리 전달기와 캐리 발생기의 합수로 표현하였고, 그것을 다시 활용하면 다음과 같다.

$$c_{i+1} = \begin{cases} a_i b_i + (a_i + b_i)c_i & = g_i + p_i c_i \\ (a_i + b_i)(a_i b_i + c_i) & = p_i(g_i + c_i) \end{cases}$$

여기에서 부 논리 회로로의 활용 가능성을 보이고자 한다. 캐리의 입력에 원래 값에 보수를 취한(negation 한 값을 넣어 연산을 수행하면

$$\begin{aligned}
 c_{i+1}(\overline{a_i}, \overline{b_i}, \overline{c_i}) &= (\overline{a_i} + \overline{b_i})(\overline{a_i} \overline{b_i} + \overline{c_i}) \\
 &= \overline{a_i} \overline{b_i} (\overline{a_i} + \overline{b_i} + \overline{c_i}) \\
 &= \overline{g_i} (\overline{p_i} + \overline{C_i}) \\
 &= \overline{g_i} + \overline{p_i} \overline{c_i} \\
 &= \overline{c_{i+1}}
 \end{aligned}$$

캐리 연산 결과를 보면 negation한 입력 값은 넣었을 때 그 결과가 원래 값의 보수가 취해져서 나옴을 볼 수 있다.

합의 연산에서도 마찬가지로 그 연산을 보일 수 있다.

$$s_i = (a_i \oplus b_i) \oplus c_i = (\overline{g_i}p_i) \oplus c_i \text{에서}$$

$$\begin{aligned}
 s_i(\overline{a_i}, \overline{b_i}, \overline{c_i}) &= (\overline{a_i} \oplus \overline{b_i}) \oplus \overline{c_i} \\
 &= (\overline{g_i}p_i) \oplus \overline{c_i} \\
 &= \overline{(\overline{g_i}p_i) \oplus c_i} \\
 &= \overline{s_i(a_i, b_i, c_i)}
 \end{aligned}$$

임을 볼 수 있다.

입력을 negation 해서 연산을 수행해도 그 결과는 negation 된 값이 나오므로, 4bit 덧셈 연산에 있어서

negation 연산은 제대로 수행됨을 볼 수 있다.

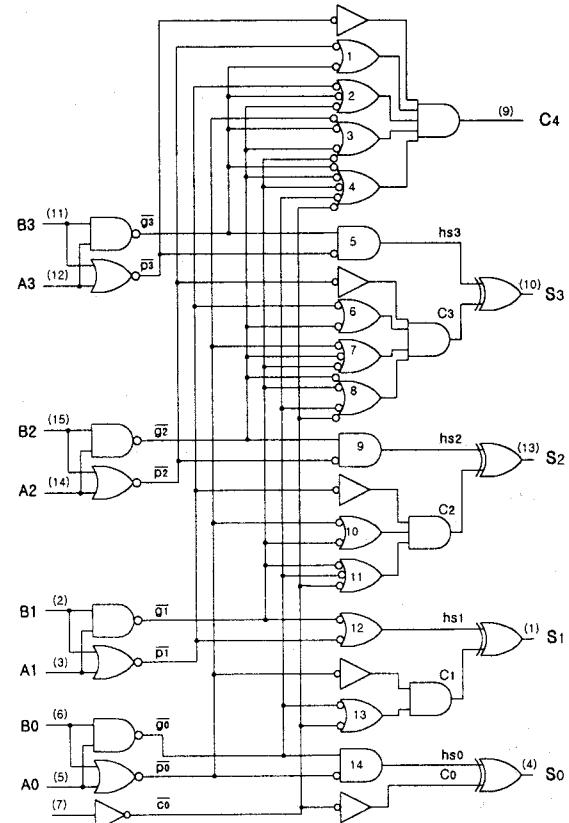


그림 3. 74LS283 Four-bit Adder

3. 결 론

이 논문은 로직 연산에 있어서 상당 부분을 차지하는 덧셈 연산에 대한 분석과 확장된 4 bit의 덧셈기인 74LS283에서의 캐리 루크 어헤드(Carry Look Ahead)연산을 수행하므로, 그 지연 시간을 줄여 덧셈을 수행함을 분석하였다.

또한 74LS283 회로가 정 논리(Positive Logic) 회로로 동작할뿐만 아니라, 부 논리(Negative Logic) 회로로도 잘 작동됨을 논리 연산을 통해 보이고 있다. 이것은 덧셈 연산기의 연산 합수가 2진수에 대해 대칭되어지는 원리가 있기 때문이고, 이러한 Negative Logic 기능을 사용하여 시스템 설계를 하면, 부호의 negation를 줄여 시스템을 보다 최적화 시킬 수 있다.

(참 고 문 헌)

[1] John F. Wakerly, "Digital Design: Principle & Practices", Prentice-Hall International Editons, 430-446, 2000

[2] Signetics Cop, "TTL DATA MANUAL", Signetics Cop, 5,475 -5,478 1971

[3] "Datasheets of 74LS283" Texas Instruments