

Borrow Look-ahead Subtractor 설계에 대한 분석

유장표·정태상
중앙대 전자전기공학부·중앙대 전자전기공학부

Analysis of the Borrow Look-ahead Subtractor Design

Jang-Pyo Yu* Tae-Sang Chung
School of Electrical and Electronics Engineering, Chung-Ang Univ.

Abstract - This paper implements and analyzes logically the Borrow Look-ahead Subtractor using Borrow Generator and Borrow Propagator. In subtracting calculation, we improve the calculating efficiency with using 4-bit subtracter which has Borrow Look-ahead Subtracters connection, and show that this is compatible with adder using the concept of Carry Generator and Carry Propagator. This subtracter may be useful in frequent subtracting calculation. We think this approach makes it possible to implement simple ALU(Arithmetic Logic Unit) with combining the concept of Borrow Look-ahead Subtractor and Carry Look-ahead Adder.

1. 서 론

거의 모든 시스템에서 이뤄지는 연산들은 대부분 가감산과 비트의 이동이 그 기초를 이룬다. 감산의 계산은 대부분 가산기를 통해서 이루어지는데, 연산에서 감산 계산을 빈번히 할 때나 감산 계산의 내용이 많을 때는 감산 전용 연산기가 필요함을 느끼게 된다.

이러한 필요성을 충족시키고, 감산 계산을 편리하게 하기 위한 감산기, Borrow Look-ahead Subtractor를 설계하였다. 이 감산기는 Borrow Generator와 Borrow Propagator의 개념을 사용하여 디자인함으로써, 가산기를 이용하여 감산 계산시 부호를 바꾸어 더하는 계산 방법보다 디자인 면에서 더 효율적으로 나타낼 수 있다.

Borrow는 각 비트에서 감산 계산을 할 때 바로 윗 자리에서 빌려오는 수를 말한다. Borrow Generator와 Borrow Propagator는 Carry Generator와 Carry Propagator의 개념을 사용하여 디자인한 가산기에서 차안한 것이다.

2. 본 론

2.1 Borrow Look-ahead Subtractor

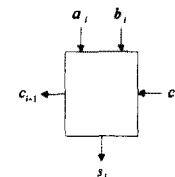
Borrow Look-ahead Subtractor는 감산 계산에서 모자라는 수를 바로 윗자리에서 빌려와서 계산하는 감산기이다. a_i , b_i , c_i 의 세 개의 입력을 받아, c_{i+1} , s_i 의 두 개의 출력을 낸다. 감산 계산을 할 때, 각각의 비트에서 계산을 수행하므로, 이를 연결하면 여러 비트의 감산 계산을 행할 수 있다. 따라서, Borrow Look-ahead Subtractor의 동작 원리를 설명하고, 4-bit subtracter를 설계해 보고자 한다.

2.1.1 Borrow look-ahead Subtractor의 설계

Borrow Look-ahead Subtractor는 연산자와 피연산자, 그리고 바로 아랫자리의 빌림수(Borrow), 이렇게

세 개의 입력을 받아 연산결과와 바로 윗자리의 빌림수 (Borrow)의 두 개의 출력을 내보내는 감산기이다. 이의 불록도를 나타낸 것이 [그림 1]이다.

그림에서 a_i 와 b_i 는 연산자와 피연산자를, c_i 는 ($i-1$)자리의 빌림수를 나타내고, c_{i+1} 과 s_i 는 각각 i 자리의 빌림수와 감산 결과를 나타낸다.



[그림 1]

$$a_i + b_i + c_i = -2, -1, 0, 1 \Rightarrow (c_{i+1} s_i)$$

위의 수식은 입력과 출력의 관계를 나타낸 식이고, [표 1]은 Borrow Look-ahead Subtractor의 진리표를 나타낸 것이다.

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

[표 1]

우선 위의 진리표를 이용하여, 각각의 출력값을 주어진 입력값으로 나타내기 위해 Karnaugh map을 구하였다.

a_i	$b_i c_i$	00	01	11	10	c_{i+1}
0		1	1	1	1	0
1				1		1

		$b_i c_i$	00	01	11	10
		a_i	0	1		
			0	1	1	1
				s_i		
				(표 2)		

위의 Karnaugh map을 이용하여 c_{i+1} 과 s_i 에 대해 minterm 표현법으로 각각 수식으로 나타내면 다음과 같다.

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

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

이를 정리하여 나타내면 다음과 같다.

$$\begin{aligned} c_{i+1} &= \overline{a_i} b_i + (\overline{a_i} + b_i) c_i \\ s_i &= a_i \oplus b_i \oplus c_i \end{aligned}$$

2.1.1 Borrow Generator and Borrow Propagator

Borrow Generator와 Borrow Propagator는 g_i 와 p_i 로 각각 표현한다. Borrow Generator는 첫자리에서 빌려올 수를 발생시키는 연산자이고, Borrow Propagator는 아랫자리에 꾸어줄 수가 필요할 때 동작하는 연산자이다. 즉 Borrow Generator는 하위 비트와 상관없이 빌림수를 생성시키는 연산자이다. 따라서, a_i 가 0, b_i 가 1일 때 출력값으로 1을 낸다. Borrow Propagator는 하위 비트에서 빌림수가 발생했을 때 생성되는 연산자이다. 따라서, $a_i b_i$ 가 각각 00, 01, 11 일 때 1을 출력으로 내는 연산자이다. 이를 수식으로 나타내면 다음과 같다. $a_i b_i$ 가 01일 때는 둘 다 1을 출력으로 내지만, 이 때는 Borrow Generator가 우선하여 동작한다.

$$\overline{a_i} b_i = g_i : \text{Borrow Generator}$$

$$(\overline{a_i} + b_i) = p_i : \text{Borrow Propagator}$$

먼저 c_{i+1} 을 Borrow Generator와 Borrow Propagator의 식으로 나타내면 다음과 같다. 이는 Sum of Products의 형태로 나타낸 것이다.

$$c_{i+1} = g_i + p_i c_i$$

우리가 설계할 것은 4비트 감산기이므로, i 가 각각

0, 1, 2, 3일 때 c_{i+1} 을 나타내면 아래와 같다.

$$\begin{aligned} c_1 &= g_0 + p_0 c_0 \\ c_2 &= g_1 + p_1 c_1 = g_1 + p_1 (g_0 + p_0 c_0) \\ &= g_1 + p_1 g_0 + p_1 p_0 c_0 \\ c_3 &= g_2 + p_2 c_2 \\ &= g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0) \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\ c_4 &= g_3 + p_3 c_3 \\ &= g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \\ &\quad + p_3 p_2 p_1 p_0 c_0 \end{aligned}$$

다음으로 s_i 를 Borrow Generator와 Borrow Propagator로 나타내기 위해서, 우선 $a_i \oplus b_i$ 를 g_i 와 p_i 의 식으로 나타낸다.

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

따라서, s_i 는 다음과 같이 나타낼 수 있다.

$$s_i = (a_i \oplus b_i) \oplus c_i = (g_i + \overline{p}_i) \oplus c_i$$

마찬가지로 i 가 각각 0, 1, 2, 3일 때 s_i 를 나타내면 아래와 같다.

$$\begin{aligned} s_0 &= (g_0 + \overline{p}_0) \oplus c_0 \\ s_1 &= (g_1 + \overline{p}_1) \oplus c_1 \\ &= (g_1 + \overline{p}_1) \oplus (g_0 + p_0 c_0) \\ s_2 &= (g_2 + \overline{p}_2) \oplus c_2 \\ &= (g_2 + \overline{p}_2) \oplus (g_1 + p_1 g_0 + p_1 p_0 c_0) \\ s_3 &= (g_3 + \overline{p}_3) \oplus c_3 \\ &= (g_3 + \overline{p}_3) \oplus \\ &\quad (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \end{aligned}$$

다음은 위의 표현 방법과는 달리, Product of Sums의 형태로 나타내면, c_{i+1} 을 다음과 같은 식으로 나타낼 수 있다.

$$\begin{aligned} c_{i+1} &= (b + c)(\overline{a} + b)(\overline{a} + c) \\ &= (\overline{a} + b)(\overline{a} b + c) \\ &= p_i (g_i + c_i) \end{aligned}$$

i 에 각각 0, 1, 2, 3을 대입하면, 다음과 같이 정리 할 수 있다.

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

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

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

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

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

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

$$c_4 = p_3(g_3 + c_3)$$

$$= p_3(g_3 + p_2(g_2 + p_1))$$

$$(g_2 + g_1 + p_0)(g_2 + g_1 + g_0 + c_0))$$

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

$$(g_3 + g_2 + g_1 + g_0 + c_0)$$

$s_i = (a_i \oplus b_i) \oplus c_i = (g_i + \bar{p}_i) \oplus c_i$ 이므로, 위의 식들을 이용하여 s_i 에 i 가 0, 1, 2, 3일 때, c_0 , c_1 , c_2 , c_3 을 각각 대입하여 나타내면 다음과 같이 된다.

$$s_0 = (g_0 + \bar{p}_0) \oplus c_0$$

$$s_1 = (g_1 + \bar{p}_1) \oplus c_1$$

$$= (g_1 + \bar{p}_1) \oplus (p_0(g_0 + c_0))$$

$$s_2 = (g_2 + \bar{p}_2) \oplus c_2$$

$$= (g_2 + \bar{p}_2) \oplus (p_1(g_1 + p_0)(g_1 + g_0 + c_0))$$

$$s_3 = (g_3 + \bar{p}_3) \oplus c_3$$

$$= (g_3 + \bar{p}_3) \oplus (p_2(g_2 + p_1)(g_2 + g_1 + p_0))$$

$$(g_2 + g_1 + g_0 + c_0))$$

위에서 Product of Sums 표현법으로 나타낸 c_{i+1} 과 s_i 를 이용하여 4-bit Borrow Look-ahead Subtractor를 설계하였다. 그 회로도는 [그림 2]와 같다.

2.2 Carry Look-ahead Adder와의 비교 분석

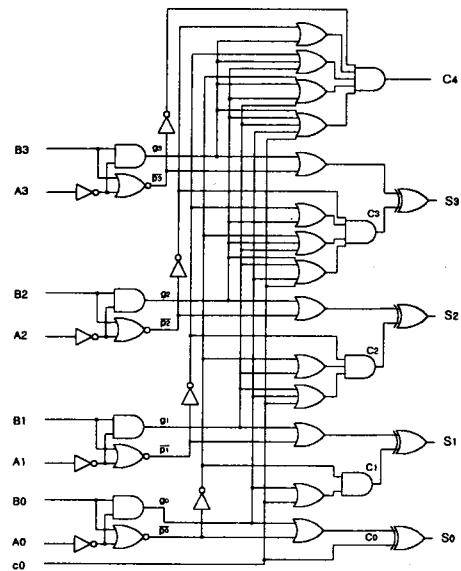
전가산기(Full Adder)를 Carry Generator와 Carry Propagator의 개념을 도입하여 연결한 4-bit Carry Look-ahead Adder에서는, 마찬가지로 a_i , b_i , c_i 의 세 개의 입력을 받아, c_{i+1} , s_i 의 두 개의 출력을 낸다. Carry Generator와 Carry Propagator를 각각 입력 a_i 와 b_i 로 나타내면 다음과 같다.

$$g_i = a_i b_i$$

$$p_i = a_i + b_i$$

또, 이를 이용하여 c_{i+1} 과 s_i 를 구하면 다음과 같다.

Logic Diagram with Product of Sums Form



(그림 2)

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

이와 위에서 구한 감산기에서의 c_{i+1} , s_i 를 비교해 보면, 각각의 논리적 결합이 유사해 두 연산기의 결합이 효과적으로 이루어질 수 있음을 알 수 있다.

3. 결 론

이 논문은 4-bit Borrow Look-ahead Subtractor를 논리적으로 구현하고, 설계해 본 것이다. 기존에 가산기를 이용하여 감산 계산을 할 경우에는 부호를 바꾸어 더해야하는 방법을 사용하였다. 그러나, 이 감산기는 Borrow Generator와 Borrow Propagator의 개념을 사용하여 디자인함으로써 논리적으로 매우 간결하게 디자인되었다.

이 감산기는 Carry Generator와 Carry Propagator의 개념을 사용하여 디자인한 4-bit Carry Look-ahead Adder에서 그 동작원리를 확인한 것으로, 서로 간단하게 결합할 수 있다. 따라서, 이를 결합하여 사용할 경우 가감산을 동시에 할 수 있는 간단한 연산기가 만들어 질 수 있다.

(참 고 문 헌)

- [1] John F. Wakerly, "Digital design : principles and Practices", Prentice-Hall, 3th edition, 430-436, 2000
- [2] Texas Instruments Inc., "Designing with TTL Integrated Circuits", McGraw-Hill, 1971
- [3] "Data sheets of and 74LS283", Texas Instruments.