論文2003-40CI-5-1

고속 십진 가산을 위한 3초과 코드 Carry Lookahead설계

(An Excess-3 Code Carry Lookahead Design for High-Speed Decimal Addition)

崔 鐘 化 *, 劉 泳 甲 *

(Jong-Hwa Choi and Young-Gap You)

요 약

십진수를 위한 가산기 구현에서 지연시간을 줄일 수 있는 carry lookahead(CLA)을 이용한 십진수 가산 회로 설계를 제안한다. 이자 계산과 같은 십진 소수에 의한 반복계산에서 이진수 체계를 사용하면 절단오차는 누적된다. 이를 방지하기 위하여 BCD 회로 사용은 불가피하다. BCD 계산에서의 속도개선은 CLA 회로를 이용하여 개선될 수 있다. BCD 회로에서 CLA 회로 사용을 위해 제안된 캐리 생성 및 캐리 전파회로를 도출하여 가산기 설계에 사용하였다. 이 CLA 방식을 사용한 BCD 가산에서 기존의 BCD 가산회로와 지연시간을 비교하였을 때 상당한 속도개선이 이루어졌다. 또한 3초과 코드를 이용한 가산회로의 경우 CLA 방식 사용과 지연시간에 영향을 미치는 회로부분을 개선함으로써 CLA만 이용했을 때 보다 지연시간을 10 게이트 지연시간만큼 더욱 줄일 수 있었다.

Abstract

Carry lookahead(CLA) circuitry of decimal adders is proposed aiming at delay reduction. The truncation error in calculation of monetary interests may accumulate yielding a substantial amount of errors. Binary Coded Decimal(BCD) additions, for example, eliminate the truncation error in a fractional representation of decimal numbers. The proposed BCD carry lookahead scheme is aiming at the speed improvements without any truncation errors in the addition of decimal fractions. The delay estimation of the BCD CLA is demonstrated with improved performance in addition. Further reduction in delay can be achieved introducing non-weighted number system such as the excess-3 code.

Keywords : Decimal, BCD, excess-3 code, carry lookahead, adder

I. 서 론

사람은 습관적으로 십진수를 기초로 연산을 하지만 컴퓨터 내부에서는 이진 표현으로 바꾸어 계산한다. 그러나 예금에 대한 이자계산과 같은 소수점 이하의 수를 표현 하는 응용 시스템의 경우, 이진 숫자표현을 하게 되면, 절단 오차는 피할 수 없게 된다. 이러한 절단 오차를 피하기 위해 십진수 계산은 필수적인 것이다[1]. 문제는 십진수 계산회로의 속도에 있다. 특히 모든 연산의 기본이 되는 BCD 가산기의 속도개선은 필수적이다. 이러한 속도개선에 대한 연구는 꾸준하게 추구되고 있다[2].

가산회로의 속도 개선을 하기 위해서는 빠른 캐리

* 正會員, 忠北大學校 情報通信工學科
(Dept. of Information Communication Engineering, Chungbuk National University.)

接受日字:2003年4月25日, 수정완료일:2003年8月25日

전달이 필요하다. 기본적으로 리플 캐리의 경우는 한 비트 계산이 완료된 후 다음 비트 계산이 수행되기 때문에 연산 지연시간은 비트수 증가에 따라 비례하여 증가한다. 이를 개선하기 위해 빠른 캐리 전달을 위한 고속 회로가 필요하다. 캐리 예견(carry lookahead : CLA)방식은 이진수 가산기 구현에서 효과적인 속도개선을 이루었다. 그러나 고속 이진가산회로에서 사용되는 캐리 예견방식을 직접 BCD에 사용할 수 없다. 이 CLA회로를 BCD에 사용하려면 적절한 캐리 생성(carry generation), 캐리 전파(carry pro pagation)에 대한 정의와 회로 개발이 필요하다. 또한 두 회로를 이용해 CLA 회로를 이용한 고속 BCD 가산기를 설계한다. 또한 BCD 뿐만 아니라 3초과 코드에 대한 CLA 회로 구현과 회로 보정을 통하여 속도 개선 효과를 보이고자 하였다.

본 논문구성은 다음과 같다. 2절에서는 십진 소수의 오차발생을 보이고 있으며, 3절은 BCD 가산기 구조와, BCD 회로에서 CLA를 사용하기 위해 필요한 캐리 생성, 캐리 전파회로를 정의하고, 4절은 3초과 가산기 구조와 지연시간을 줄이기 위해서 기존의 3초과 가산기를 수정한 회로를 제시하고, 5절은 지연시간에 의한 속도개선 결과를 제시하여, 성능을 평가한다. 마지막으로 6절에서는 결론을 맺는다.

Ⅱ. 오차 분석

십진법으로 표기된 소수를 이진수로 변환하면 절단오차의 발생이 불가피하다. 십진소수를 N비트 이진수로 변환할 경우 $1/2^N$의 배수에 속하는 값에 대해서만 오차 없는 표현이 가능하다. 대부분 비트의 길이가 무한할 경우에 이진소수가 십진소수에 근접하게 다가 갈 것이다. 그러나 소수 부분에 표현에 사용되는 비트수가 제한되어 있다. 예를 들면 십진소수 0.6_{10}을 이진소수로 표현하면 $0.100110011001\cdots\cdots_2$로 1001_2이 반복적으로 나타나는 무한 순환소수가 된다. 무한히 반복적인 수를 유한한 자릿수로 표현하려면 절단 오차는 불가피 하다. 만약 수치표현을 다섯 번째 자리로 놓는다면 값은 0.10011_2가 된다. 이것을 십진소수로 변환하면 0.59375_{10}로서 절단오차가 발생한다. 또한 절단오차가 발생된 값들의 계산 시 발생되는 누적오차도 피할 수 없을 것이다.

<표 1>은 소수점 표현 자릿수가 4비트일 경우, 각 수별 절대오차와 상대오차를 나타내었다. <그림 1>은 디지트 증가에 따른 최대 절대오차를 보인다. 소수점 한자리에 대한 최대 절단오차는 0.025로서 25%의 상대오차를 발생시키고 있다. 이 오차를 줄이기 위하여 비트수를 늘려서 계산한 결과가 <그림 1>에 보여 졌다. 여기서는 4비트씩 증가할 때 1/16씩 감소하는 것을 볼 수 있다.

이러한 절단오차는 십진수 계산을 다루는 응용분야에서 상당한 문제를 야기 시킬 수 있다. 예금에 대한 이자율의 경우, 십진소수를 이용한다. 이것을 이진소수로 변환할 경우 절단오차가 불가피 하다. 복리 계산이나 많은 고객을 가진 예금의 이자계산에서 금액의 차이는 크게 증가하기 마련이다. 이러한 누적오차 방지를 위하여 십진 계산을 할 수 밖에 없다.

표 1. 이진화 십진 변환시 4비트 표현시 에러
Table 1. Binary to decimal conversion errors of 4bit expressions.

decimal fraction (A)	4bit binary expression (B)	decimal equivalent of (B)	truncation error | A - B |	relative truncation error | A - B | / A
0.1	0.0010	0.125	0.025	25%
0.2	0.0011	0.1875	0.0125	6.25%
0.3	0.0101	0.3125	0.0125	4.17%
0.4	0.0110	0.375	0.025	6.25%
0.5	0.1000	0.5	0	0%
0.6	0.1010	0.625	0.025	4.17%
0.7	0.1011	0.6875	0.0125	1.78%
0.8	0.1101	0.8125	0.0125	1.56%
0.9	0.1110	0.875	0.025	2.78%

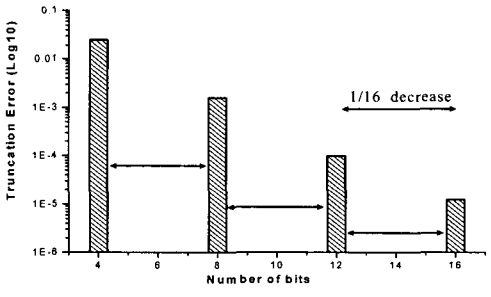


그림 1. 디지트 증가에 따른 최대 절단오차
Fig. 1. The maximum truncation error for the increase of digits.

Ⅲ. BCD 가산기

BCD에서는 한자리의 십진수를 4비트 단위로 표현한다. BCD 가산의 경우 두 입력이 a, b일 때, sum은 입력 a, b와 캐리 입력 C_{in}의 합에 의하여 모듈러-10(modulo-10) 한 나머지로 표현되고, 캐리 출력 C_{out}은 세 입력의 합 a, b, C_{in}이 10이상인 경우는 1이 되고 그렇지 않은 경우는 0이 된다. 이는 다음과 같이 수식으로 표현될 수 있다.

$$\text{Sum} = (a + b + C_{in}) \bmod 10 \tag{1}$$

$$C_{out} = \begin{cases} 1 \text{ if } (a+b+C_{in}) \geq 10 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

회로 구성은 십진수 한자리에 대응되는 4비트를 병렬로 더할 수 있는 전가산기 4개가 위치하여 각각의 비트에서 더해지게 된다. 여기에 4비트를 더한 값이 1001보다 큰 경우 이를 보정하기 위한 회로를 갖추고 있다. 보정 회로는 두 입력의 합이 1010부터 10011(10_{10}~19_{10}) 사이에 출력이 발생한 것을 체크하면 된다. 또한 입력에 대한 6개의 조합 1010부터 1111의 보정도 필요하다. 이것은 상위 비트가 1이고, 2, 3번째 비트 중 어느 하나라도 1일 때 보정을 수행하면 된다. 이 때 보정 값은 입력된 이진 값의 합에 0110(6_{10})을 더해준다. 0110을 더해주는 것은 이진수에서 발생되는 캐리와 십진수에서 발생되는 캐리 차이가 6이기 때문이다. 보정 회로는 2개의 전가산기와 한 개의 XOR로 구성된다.

N개의 십진 숫자를 더하는 N디지트 BCD 가산기는 n개의 BCD 가산기가 필요하다. 한 가산기로부터의 캐리 출력은 다음 단계 가산기의 캐리 입력에 접속 되어야 한다. 예를 들어 4디지트 BCD 가산기에서 첫 번째 BCD 가산기 동작 후 발생한 캐리 출력은 두 번째 BCD 가산기의 캐리 입력으로 전달된다. 따라서 첫 번째 캐리 발생 여부에 따라 두 번째 BCD 가산기가 동작하게 된다. 세 번째, 네 번째 가산기도 캐리 발생 여부에 따라 동작한다.

1. Carry Lookahead

캐리 예견(carry lookahead : CLA)회로를 사용하는 것은 캐리 전파에 소요되는 시간을 줄이는 것이다. i번째 단계 계산이 필요한 캐리를 구함에 있어서, 하위의 매 단계마다 차례로 전파시키기 보다는 전체 하위 단계(i-1, i-2, …, 1)의 모든 입력 신호로부터 직접 i번째 단계의 캐리를 예견하여 발생시키는 것이다. N비트 CLA 가산기에서 각 단계에 필요한 캐리 C_i는 다음의 논리식과 같이 캐리 생성신호 G_i와 캐리 전파신호 P_i를 이용하여 예측할 수 있다.

각 단계에 전송되는 캐리 C_i는 반복적인 논리식에 의해 일반화된다.

$$C_i = G_i + P_i C_{i-1} \tag{3}$$

여기서, i = 1, 2, …, n 이다.

식 (3)의 논리식을 확장하면 각 단계의 캐리 관계와 최초 캐리, C_0는 다음과 같이 주어진다.

$$C_i = G_i + G_{i-1}P_i + G_{i-2}P_iP_{i-1} + \cdots + G_1P_i\cdots P_2 + P_i\cdots P_1C_0 \tag{4}$$

여기서, i = 1, 2, …, n 이고, $C_0 = C_{in}$ 이다.

CLA 방식을 사용함에 있어서 가산기의 수가 증가하면, fan-in limitation, 캐리 발생 논리식의 복잡도가 증가하기 때문에 여기서는 캐리 예견회로의 단계 숫자를 제한할 필요가 있다.

이진 CLA의 경우 1비트 전가산기에서 캐리 생성신호와 캐리 전파신호를 사용한다. 하지만, BCD CLA 가산기의 경우 입력이 4비트 형태이므로 4비트에 대한 캐리 생성회로와 캐리 전파회로가 필요하다. <그림 2>는 4디지트 BCD CLA 가산기 회로이다. 1디지트 BCD 가산기를 기준으로 캐리 생성회로와 캐리 전파회로를 적용하여 CLA 가산기 회로로 구성한다.

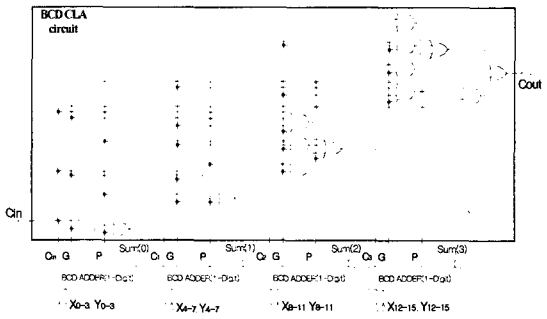


그림 2. 4디지트 BCD CLA 가산기
Fig. 2. 4-digit BCD CLA adder.

2. Carry Generation 및 Propagation 회로

캐리 생성(carry generation)회로는 현 디지트 단계에서 입력 조건에 따라 캐리를 발생을 시키는 것이다. BCD에서는 두 입력 디지트 X, Y의 합이 10이상 일 때 조건이다. 그러나 발생 조건 중 몇 가지는 제외된다. BCD의 입력에 사용되는 수 범위가 0부터 9까지 이므로 입력 4비트 표현 중 10에서 15까지는 dont' care로 처리할 수 있다. 또한 한 디지트 입력이 0인 경우는 캐리가 발생하지 않기 때문에 제외한다. 따라서 나머지 1에서 9까지의 입력에 대해 다른 하나의 입력을 더했을 때 10이상인 조건을 논리곱하면 된다. 캐리 생성 논리식은 (5)와 같이 나타나며, 간략화한 결과는 식 (6)과 같다.

Carry-Generation :

$$= X_3X_0(Y_3 + Y_2 + Y_1 + Y_0) + X_3(Y_3 + Y_2 + Y_1) + X_2X_1X_0(Y_3 + Y_2 + Y_1Y_0) + X_2X_1(Y_3 + Y_2 + Y_1Y_0) + X_2X_0(Y_3 + Y_2Y_1 + Y_2Y_0) + X_2(Y_3 + Y_2Y_1) + X_1X_0(Y_3 + Y_2Y_1Y_0) + X_1Y_3 + X_0(Y_3Y_0) \quad (5)$$

$$= X_3X_0Y_0 + X_3Y_3 + X_3Y_2 + X_3Y_1 + X_2X_1X_0Y_1Y_0 + X_2X_1Y_2 + X_2X_0Y_2Y_0 + X_2Y_3 + X_2Y_2Y_1 + X_1X_0Y_2Y_1Y_0 + X_1X_3 + X_0Y_3Y_0 \quad (6)$$

캐리 전파(carry propagation)회로는 하위에서 발생된 캐리를 상위 디지트로 전파 시키는 현 디지트의 입력 조합으로 구성된다. 캐리 전파 조건은 두 입력 X, Y와 캐리 입력의 합이 10이면 충족한다. 여기서도 입력조건 중 10부터 15까지는 don't care로 처리한다. 한 디지트 입력이 0인 경우는 캐리 생성조건과는 다르게 하위 캐리 입력이 들어왔을 경우 캐리 전파가 가능하므로 조건으로 사용한다. 캐리 전파조건은 0에서 9까지의 입력에 대해 다른 쪽 입력과의 합이 9일 조건을 논리곱 한다. 두 입력의 합이 10이상인 경우는 전파 조건으로써 의미가 없다. 캐리 전파조건 논리식은 식 (7)과 같고, 합이 10이상인 경우는 don't care처리를 함으로써 식 (8)과 같은 식을 얻을 수 있다.

Carry-Propagation :

$$= X_3X_0 + X_3(Y_3 + Y_2 + Y_1 + Y_0) + X_2X_1X_0(Y_3 + Y_2 + Y_1) + X_2X_1(Y_3 + Y_2 + Y_1Y_0) + X_2X_0(Y_3 + Y_2) + X_2(Y_3 + Y_2Y_1 + Y_2Y_0) + X_1X_0(Y_3 + Y_2Y_1) + X_1(Y_3 + Y_2Y_1Y_0) + X_0(Y_3) + (Y_3Y_0) \quad (7)$$

$$= X_3X_0 + X_3Y_0 + X_2X_1X_0Y_1 + X_2X_1Y_1Y_0 + X_2X_0Y_2 + X_2Y_2Y_0 + X_1X_0Y_2Y_1 + X_1Y_2Y_1Y_0 + X_0Y_3 + Y_3Y_0 \quad (8)$$

위 식은 모든 입력조건을 정리하여 카르노맵을 써서 간략화한 것이다. 이 논리식들은 일반 CMOS로 구현할 경우 3~4 레벨 논리 회로로서 구현하는데 무리가 없다.

Ⅳ. 3초과 가산기

이 절에서는 비가중치 코드인 3초과 코드를 이용한다. 3초과코드는 8421코드에 3을 더한 결과를 4비트 이진수로 나타낸다. 3초과 코드도 마찬가지로 4비트로 표시할 수 있는 코드 조합에서 10개만을 사용한다. (0011~1100). 나머지는 dont' care 처리를 한다. 8421코드에서 3초과 코드와의 상호변환식은 카르노맵을 이용해 <표 2>에서와 같이 각 비트위치별로 최소화할 수 있다.

표 2. 코드변환 식
Table 2. Conversion expression.

Conversion (BCD → EX-3)		
ABCD (BCD)	WXYZ (EX-3)	
0000	0011	$W = A + B(C+D)$
0001	0100	$X = \bar{B}(C+D) + B\overline{CD}$
0010	0101	$Y = \overline{C \oplus D}$
0011	0110	$Z = \bar{D}$
0100	0111	
0101	1000	
0110	1001	
0111	1010	
1000	1011	
1001	1100	

Conversion (EX-3 → BCD)		
WXYZ (EX-3)	ABCD (BCD)	
0011	0000	$A = W(X + YZ)$
0100	0001	$B = \overline{XY} + Y(XZ + W\bar{Z})$
0101	0010	$C = Y \oplus Z$
0110	0011	$D = \bar{Z}$
0111	0100	
1000	0101	
1001	0110	
1010	0111	
1011	1000	
1100	1001	

3초과 코드 가산기의 동작은 다음과 같이 나타낼 수 있다.

$$\text{if}\,(a_{ex3}+b_{ex3}+C_{i_n}) \geq 16$$
$$\text{Sum} = a_{ex3}+b_{ex3}-3+C_{in}-10\,;$$
$$C_{out}=1\,;$$
$$\text{otherwise}$$
$$\text{Sum} = a_{ex3}+b_{ex3}-3+C_{in};$$
$$C_{out}=0;$$

동작에 따른 회로 구성을 살펴보면 3초과 코드 4비트 두 입력을 받아 더해진 결과는 각 입력마다 3씩 더해진 값이 들어오기 때문에 최종 합은 6이 더해진 결과가 나온다. 따라서 캐리 출력은 입력의 합이 16이상일 때 발생하여 다음 자릿수 계산에 적용되어야 한다. 이는 4비트 이진 가산기의 캐리 발생조건과 동일하다. 또한 3초과 코드체계에 적합한 최종 sum 출력을 얻기 위해서는 보정회로가 필요하다. 보정회로는 해당 자릿수에 캐리가 발생했을 경우는 0011을 더해주는 보정회로가, 발생하지 않은 경우는 1100을 더해주는 보정회로가 필요하다.

3초과 가산기의 캐리 발생 형태는 앞에서 언급한 바와 같이 4비트 이진 가산기의 캐리 발생 조건과 동일하다. 따라서 이진 가산회로에서 사용되는 CLA 회로는 3초과 가산기 회로에서도 동일하게 적용될 수 있다. 즉, BCD 가산기와는 다르게 3초과 코드 가산기는 CLA를 적용하기 위하여 별도의 캐리 생성 및 전파회로를 구성할 필요가 없다. 캐리 생성과 전파 회로는 3초과 가산기에 사용되는 전가산기에서 사용할 수 있다.

Carry generation : G = $x_{ex3} \cdot y_{ex3}$
Carry propagation : P = $x_{ex3} \oplus y_{ex3}$

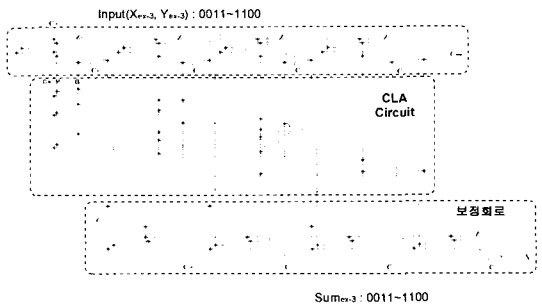


그림 3. 1디지트 3초과 CLA 가산기
Fig. 3. 1-digit excess-3 CLA adder Carry lookahead.

그러나, <그림 3>에서 보는바와 같이 CLA 회로를 사용해 캐리 전달을 빠르게 하더라도 보정회로에서 sum 출력을 위해 보정회로의 캐리 값이 리플형태로 전파 되므로, 그만큼 sum출력이 늦어진다. 또한 3초과 값에서 이진화 십진 값으로 변환해주는 변환회로가 추가되기 때문에 그만큼 출력은 지연 된다. 지연시간을 줄이기 위해서 보정회로에 CLA 구조를 사용 할 수도 있겠지만, 회로 복잡도와 팬인(fan-in), 팬아웃(fan-out)문제를 고려했을 때 바람직하지 못할 것이다.

따라서, 이런 많은 지연시간을 차지하는 회로부분의 개선방안을 제안한다. 보정회로는 두개의 3초과 코드입력에 대한 계산결과를 보정하여 3초과 코드로 출력하는 회로이다. 그런데, 보정전의 계산결과를 보면, 더한 값이 15보다 큰 경우에는 보정전의 결과가 BCD 코드임을 알 수 있다. 기존방식에서는 항상 보정회로를 거친 후 BCD 코드로 변환한다. 그러나 앞에서 언급한 경우에는 보정전의 계산결과가 BCD 코드이므로 보정 및 코드변환을 하지 않아도 된다.

하지만 계산결과가 15이하일 경우가 문제이다. 넘어온 계산 값에 대하여 이진화 십진 값으로 변환해 줄 회로가 필요하다. 이 회로를 만들기 위해서 카르노맵을 이용해 각 자릿수에 대해 최소화를 했을 때 <표 3>과 같이 나타난다.

표 3. 변환 식
Table 3. Conversion expression.

Sum conversion		
$Z_3Z_2Z_1Z_0$(Z)	$S_3S_2S_1S_0$ (Sum)	
0110	0000	
0111	0001	$Sum(3) = Z_3 \oplus Z_2 \oplus Z_1$
1000	0010	$Sum(2) = Z_2 \oplus Z_1$
1001	0011	
1010	0100	$Sum(1) = \overline{Z_1}$
1011	0101	$Sum(0) = Z_0$
1100	0110	
1101	0111	
1110	1000	
1111	1001	

결과적으로 최종출력은 보정전의 계산결과가 15를 초과하는 경우와 아닌 경우에 따라 MUX회로를 통하여 선택된다. 캐리가 발생한 경우는 그대로 sum을 출력하고, 발생하지 않을 경우는 결과보정 출력회로를 거쳐 출력을 내보내는 것이다. <그림 4>는 보정출력 회로를 나타낸다.

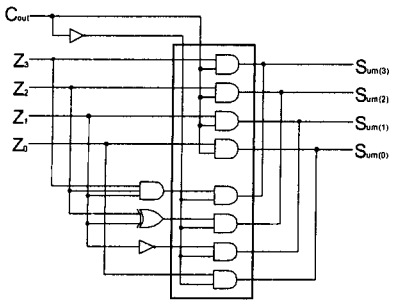


그림 4. 보정출력회로
Fig. 4. Output correction circuit

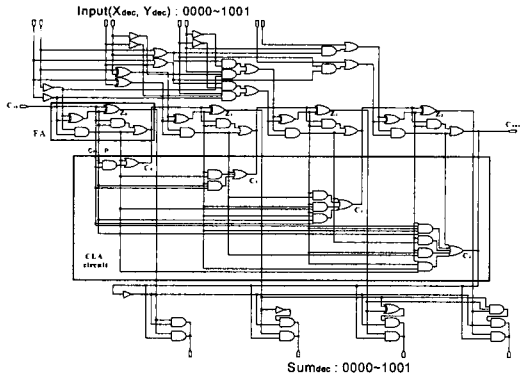


그림 5. 제안된 1-digit 3초과 CLA 가산기
Fig. 5. Proposed 1-digit excess-3 CLA adder.

<그림 5>는 보정회로와 코드 변환회로를 빼고 보정 출력회로로 바꾼 1디지트 3초과 CLA 가산기를 나타낸다. 이렇게 사용할 경우 기존의 보정회로 및 변환회로를 포함한 3초과 가산회로보다 지연시간 및 게이트수를 줄일 수 있다.

V. 지연시간 분석

이제 CLA에 의한 속도 개선효과를 보기로 하자. 먼저 <그림 6>과 같이 1디지트 BCD 가산기 회로의 경우, 각각의 출력 sum 비트와 C_{out}의 지연시간 성분은 다음과 같이 나타낼 수 있다.

$$C_{out} = C_3 + Z_3Z_2 + Z_3Z_1$$
$$Sum(3) = Z_3 \oplus C_{21}$$
$$Sum(2) = Z_2 \oplus C_{out} \oplus C_{20}$$
$$Sum(1) = Z_1 \oplus C_{out}$$
$$Sum(0) = Z_0$$

여기서 한 개의 게이트 지연시간을 ΔT라고 하면, 1디지트 BCD 가산회로의 전가산기 4개를 통과했을 때 걸리는 출력 sum 지연시간은 $2n\Delta T$, C_{out}은 $(2n+1)\Delta T$로 나타낼 수 있다. 하지만 N디지트로 구성될 경우 첫 번째 디지트를 제외한 나머지 디지트 출력 sum은 $(2(n-1)+1)\Delta T$를 갖고, C_{out}의 경우는 $2n\Delta T$를 갖는다.

C_{out}의 경우 4개의 전가산기와 마지막으로 3입력 OR 게이트를 통과한 후의 게이트 지연시간은 $2n+1=2\cdot4+1=9\Delta T$를 갖는다. sum의 경우 지연시간은 출력 비트 중 지연시간이 긴 것을 선택해야 한다. 각각 sum출력 비트들을 살펴보면, Sum(3)은 4번째 전가산기의 sum출력과 보정회로의 2번째 전가산기의 캐리의 출력이 XOR

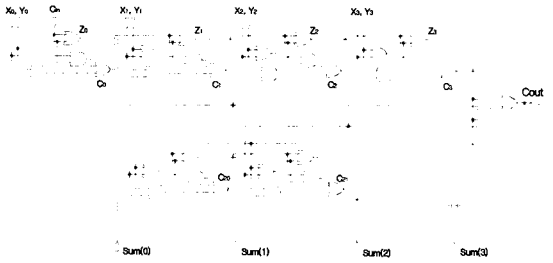


그림 6. 1디지트 BCD 가산기
Fig. 6. 1-digit BCD adder.

형태로 나타낸다. Sum(2)는 3번째 전가산기의 sum출력과 C_{out}, 그리고 보정회로의 첫 번째 캐리 출력의 XOR 형태로 나타낸다. Sum(1)은 2번째 전가산기의 sum출력과 C_{out}의 XOR 형태로 나타낸다. Sum(0)은 1번째 전가산기의 sum출력이 그대로 사용되므로, $X_0 \oplus Y_0 \oplus C_{in}$으로 나타낼 수 있다.

이때의 게이트 지연시간은 Sum(3) = $15\Delta T$, Sum(2) = $13\Delta T$, Sum(1) = $11\Delta T$, Sum(0) = $1\Delta T$와 같이 나타난다. sum의 총 지연시간은 출력 비트 중 가장 긴 Sum(3)인 $15\Delta T$를 선택한다.

N디지트 BCD 가산기일 경우 BCD 가산기를 리플 캐리형식으로 사용한다면, 지연시간은 $(C_{pd}\cdot(N-1)+S_{pd})\Delta T$ 형태로 나타날 것이다. 여기서 N은 디지트 수이고, C_{pd}는 캐리 지연시간이며, S_{pd}는 sum 지연시간을 나타낸다.

여기에 CLA 방식을 사용할 경우 모든 캐리를 병렬로 계산이 가능하기 때문에 빠른 캐리 전달로 인한 계산속도가 훨씬 빨라질 것이다. BCD에서의 캐리 계산 방식은 현재 사용하고 있는 CLA 방식과 동일하다. 다

만 BCD에서는 1디지트를 4비트로 표현하므로 비트 단위의 CLA와는 캐리 생성회로와 캐리 전파회로가 다르며 이에 따른 지연시간 차이가 있다. 캐리 생성회로와 캐리 전파회로의 지연시간은 AND 게이트와 OR 게이트로 구성하는데 팬인을 고려하여 3~4ΔT정도로 구현할 수 있는데, 여기서는 3ΔT로 한다.

제안한 BCD 가산기는 4디지트 그룹별 CLA 방식을 채택 하였다. 이때 한 그룹에서 사용되는 지연시간은 세 가지의 회로 성분에 의하여 결정된다. τ_{pg}는 각 디지트 위치에서 캐리생성 및 전파신호를 생성하는데 걸리는 시간, τ_{digit}는 4디지트 그룹 내에서 내부 캐리를 결정하는 시간, 그리고 τ_{sum}은 1디지트 BCD sum을 결정하는 시간이다. 이때 각 성분에 걸리는 지연시간은 $\tau_{pg}=3\Delta T$, $\tau_{digit}=2\Delta T$, $\tau_{sum}=15\Delta T$와 같이 나타난다.

각각의 디지트에 나오는 총 sum 지연시간은 $\tau_{pg}+\tau_{digit}+\tau s_{um}=20\Delta T$ 이다.

N디지트 BCD 가산기에서 CLA 방식을 사용하는 경우 지연시간은 스테이지 구성에 따라 지연시간 계산이 달라질 수 있다. 그룹 CLA의 경우 회로의 총 지연시간은 $(\tau_{total}+4(\log_b N-1))\Delta T$로 나타낼 수 있다. 여기서 τ_{total}은 각 디지트에서 나오는 총 sum 지연시간, N은 디지트, b는 blocking factor 이다.

<그림 7>은 디지트 증가에 따른 가산기 구성방식별 지연시간을 나타낸 것이다. 디지트의 수가 증가할수록 단일 CLA 방식보다 그룹 CLA 방식의 지연시간이 작음을 알 수 있다. 그림에서 표시된 선들은 리플 캐리 가산기, 리플 캐리 + 4 디지트 CLA(1-stage) 가산기, 리플 캐리 + 16 디지트 그룹 CLA(2-stage) 가산기를 나타낸다.

3초과 가산기의 지연시간 성분을 살펴보면 다음과 같다. 우선 입력된 값을 더해주는 가산회로와 출력된 값을 3초과 코드에 맞게 보정해주는 보정회로가 있다. 그리고 BCD 코드를 3초과 코드로, 3초과 코드를 BCD 코드로 변환해주는 회로가 필요하다. 여기서도 게이트를 통과했을 때 생기는 지연시간을 ΔT라고 할 때 sum 지연시간 성분을 보면 가산회로에서 $(2(n-1)+1)\Delta T$를 갖고 보정회로에서 같은 지연시간을 가지므로 $2(2(n-1)+1)\Delta T$만큼의 지연시간이 생기고, Cout의 경우는 $2n\Delta T$를 갖는다. 또한 sum에 추가되는 지연시간이 있는데 바로 코드 변환 시 사용되는 게이트 통과 지연시간이다.

3초과 코드 가산기의 지연시간은 BCD 가산기보다

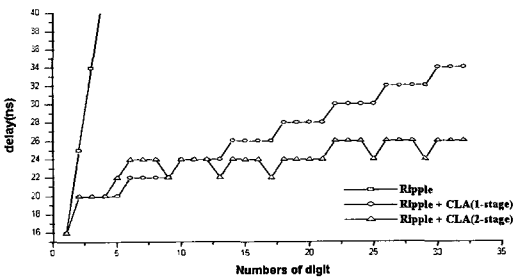


그림 7. BCD 가산기 지연시간 비교
Fig. 7. Comparison of delays of BCD CLA adders.

지연시간이 더 커지며 이는 보정회로의 차이 때문이다. BCD 가산기의 경우 보정회로에서 두 개의 전가산기만이 사용되지만 3초과 코드 가산기의 경우 각 디지트에서 비트별 가산기가 필요하다. 이렇게 3초과 코드 가산기의 경우 BCD 가산기보다 지연시간이 크지만, CLA 회로를 사용하기 위해서 특별한 캐리 생성회로 및 전파회로를 만들 필요가 없다는 장점이 있다. CLA 회로를 사용했을 경우 감소시킬 수 있는 부분은 가산회로의 지연시간 부분이다.

이 가산기의 4비트 그룹별 지연시간 성분은 다음과 같다. τ_{becon}은 BCD 코드 입력을 3초과 코드 입력 값으로 변환하는데 걸리는 시간, τ_{pg}는 각 비트 위치에서 캐리 생성과 전파신호를 전달하는데 걸리는 시간, τ_{digit}은 4비트 그룹 내에서 내부 캐리를 결정하는데 걸리는 시간, τ_{sum}은 1디지트 3초과 sum을 결정하는데 드는 시간, τ_{ebcon}은 3초과 코드 출력 값을 BCD 코드 출력 값으로 변환하는데 걸리는 시간이다. 지연시간 성분은 $\tau_{becon}=3\Delta T$, $\tau_{pg}=1\Delta T$, $\tau_{digit}=2\Delta T$, $\tau_{sum}=9\Delta T$, $\tau_{ebcon}=3\Delta T$가 되며, 각 디지트에서 나오는 총 sum 계산 지연시간은 $\tau_{becon}+\tau_{pg}+\tau_{digit}+\tau_{sum}+\tau_{ebcon}=18\Delta T$ 이다. 이것은 단일 CLA를 사용한 것으로서 디지트가 증가할수록 $2\Delta T$만큼의 지연시간이 증가하게 된다. N디지트의 경우$(16+2N)\Delta T$ 만큼의 지연시간을 갖는다. CLA 회로를 여러 스테이지로 구성 했을 경우, 회로의 총 지연시간은 $(\tau_{total}+4(\log_b N-1))\Delta T$로 나타 낼 수 있고 CLA 그룹별 지연시간을 <그림 8>에서 볼 수 있다. 디지트가 커질수록 CLA를 여러 스테이지를 쓸수록 지연시간이 줄어드는 것을 볼 수 있다.

여기에 보정회로와 BCD 변환회로를 빼고 보정출력 회로로 바꾼 1디지트 3초과 BCD CLA 가산기를 사용했을 경우 각 단마다 $10\Delta T$만큼의 지연시간을 줄일 수 있다. <그림 9>는 리플 캐리 BCD 가산기와 BCD

CLA(2-stage) 가산기, 3초과 CLA(3-stage) 가산기, 보정회로와 변환 회로 부분을 변형시킨 제안된 3초과 CLA(3-stage) 가산기의 지연시간을 나타낸다.

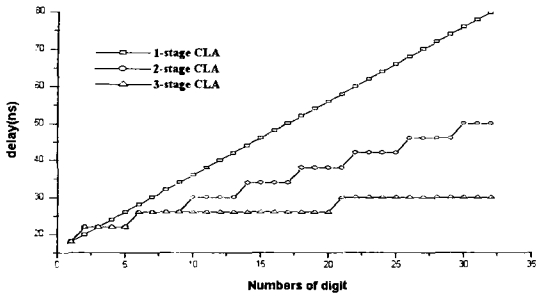


그림 8. 3초과 CLA 가산기 지연시간 비교
Fig. 8. Comparison of delays of excess-3 CLA adders.

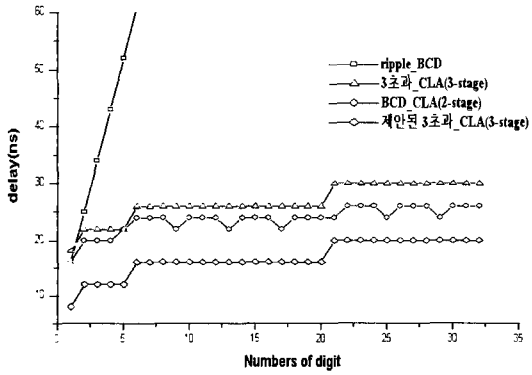


그림 9. 디지트 증가에 따른 방식별 지연시간 비교
Fig. 9. Comparison of delay due to digit increase.

<그림 9>를 보면 CLA를 사용한 회로들이 리플 캐리 BCD 가산기 보다 지연시간이 월등히 작음을 알 수 있다. 제안된 3초과 CLA(3-stage) 가산기의 경우 3초과 CLA(3-stage) 가산기 보다 각 디지트 지연시간이 10ΔT 만큼 감소하였고 BCD CLA(2-stage) 가산기 보다도 지연시간이 감소하였다. 비교 방식에서 BCD CLA를 2스테이지로 한 것은 BCD 가산기에서 CLA 회로 사용이 4디지트이기 때문이다.

Ⅵ. 결 론

절단오차가 생기지 않는 십진 가산방식에서 빠른 계산을 위해 CLA 방식을 사용한다. 이 때 BCD 가산기의 경우 CLA 회로를 사용하기 위해서 캐리 생성회로와 캐리 전파회로를 제안하였다. 인접한 하위의 캐리를 계산 한 후 상위의 덧셈을 수행하는 것이 아니라 각 단의 캐리를 미리 예측, 계산함으로써 빠른 연산이 가능하다. 또한 3초과 가산기의 경우 CLA 사용과 더불어 변환회로나 보정회로에서 걸리는 지연시간을 줄일 수 있는 회로를 제안하였다. 제안된 회로를 사용할 경우 3초과 CLA 가산기보다 10ΔT 만큼의 지연시간을 줄일 수 있었다. 제안된 캐리 생성, 전파회로가 들어간 BCD CLA 회로와 제안된 3초과 CLA 가산기를 사용할 경우 캐리 전달 지연시간의 감소로 인해 일반적인 리플 캐리 형태의 BCD 가산기보다 고속연산을 수행할 수 있다.

참 고 문 헌

[1] M. F. Cowlish, et al., "A decimal floating-point specification," Proc. 15th IEEE Sym-posium on Computer Arithmetic, pp. 147-154, June 2001.

[2] M. S. Schmookler and A. Weinberger, "High speed decimal addition," IEEE Trans-actions on computers, vol. C-20, no. 8, pp. 862-866, August 1971.

[3] B. Parhami, Computer Arithmetic Algorithms and Hardware Designs, Oxford University Press, 2000.

[4] R. J. Tocci and N. S. Widmer, Digital Systems Principles and Applications, Prentice-Hall International, 1998.

[5] ANSI/IEEE Std 754-985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, 1985.

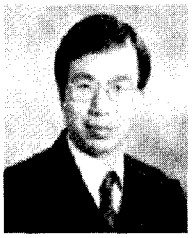
[6] M. J. Flynn, S. F. Oberman and M. J. Flynn, Advanced Computer Arithmetic Design, John Wiley & Sons, 2001.

[7] M. M. Mano, Digital Design, 3rd Ed, Prentice-Hall, 2002.

저 자 소 개

崔 鐘 化(正會員)

2001년 2월 : 충북대학교 반도체공학과(학사). 2001년~현재 : 충북대학교 정보통신 석사과정. <주관심분야 : VLSI설계, 연산회로 설계>



劉 泳 甲(正會員)

1975년 8월 : 서강대학교 전자공학과 공학사. 1975년~1979년 : 국방과학연구소 연구원. 1981년 8월 : Univ.of Michigan, Ann Arbor 전기전산학과 공학석사. 1986년 4월 : Univ.of Michigan, Ann Arbor 전기전산학과 공학박사. 1986년~1988년 : 금성반도체(주) 책임 연구원. 1993년~1994년 : 아리조나 대학교 객원교수. 1998년~2000년 : 오레곤 주립대학교 교환교수. 1988년~현재 : 충북대학교 정보통신공학과 교수. <주관심분야 : VLSI 설계 및 Test, 고속 인쇄 회로 설계, Cryptography>