논문 2006-43CI-6-2

고속 다이나믹 십진 가산기 설계

(High-Speed Dynamic Decimal Adder Design)

유 영 갑**, 김 용 대**, 최 종 화*

(Younggap You, Yong-Dae Kim, and Jong hwa Choi)

요 약

본 논문은 십진수 가산에서 속도 개선을 위한 가산 회로를 제안하였다. 속도 개선을 위한 방법으로 빠른 캐리 전달 방식으로 알려진 캐리 예견(carry loohahead) 회로를 사용하였다. 또한 빠른 십진 연산을 위해 입력식의 간략화 및 다이나믹 구조를 적용함으로서 가산 출력 지연시간을 줄였다. 제안된 회로의 가산기 구현에서 0.18 ㎛ CMOS 공정을 이용한 타이밍 시뮬레이션 측정 결과, 16 디지트 가산에 걸리는 최대 지연시간은 0.83 ns로 나타났다. 제안된 방법은 다른 십진 가산 방식과 비교 했을 때 가산에 따른 지연시간이 작다.

Abstract

This paper proposed a carry lookahead (CLA) circuitry design. It was based on dynamic circuit aiming at delay reduction in an addition of BCD coded decimal numbers. The performance of these decimal adders is analyzed demonstrating their speed improvement. Timing simulation on the proposed decimal addition circuit employing 0.18 ㎛ CMOS technology yielded the worst-case delay of 0.83 ns at 16-digit. The proposed scheme showed a speed improvement compared to several schemes for decimal addition.

Keywords: Decimal adder, Dynamic, CLA

Ⅰ. 서 론

인간 사회에 있어서 대부분의 수 체계는 십진수 체계를 사용한다. 하지만 계산 시스템의 대부분은 이진수 체계를 사용한다. 십진수 체계보다 데이터의 표현 및 저장률, 처리속도 면에서 효율성이 좋기 때문이다. 이진 연산에 비해 십진 연산 형태의 경우 저장률이 20% 낮고, 하드웨어 부분도 약 15%를 더 요구한다[1]. 이런 문제들로 인하여 대부분의 컴퓨터 및 계산기 그리고 응용 회로들은 이진수를 기반으로 하는 회로들로 이루어져 있다.

이진 응용 시스템의 경우, 십진 입력에 대한 수 체계 변환 시 오차는 피할 수 없게 된다. 그 결과로서 금융상에서 예금에 대한 오차율 계산이나, 토지 면적의 단위 환산, 소수점 단위까지의 정확한 회계 등 문제가 될 수 있는 요인으로 작용할 수 있다[2]. 이러한 변환 오차가 없는 십진 가산기의 문제점은 가산에 걸리는 처리시간이다. 처리시간을 줄이기 위해 많은 연구가 계속되어 왔다[3,4,11].

가산 회로의 속도 개선을 위해서 빠른 캐리 전달과 빠른 덧셈 출력을 위한 회로가 필요하다. 일반적으로 캐리 예견 (carry lookahead Addition : CLA) 방식은 효과적인 속도 개선을 이루었다[5]. 그러나 십진 방식에 사용하기 위해서는 십진 캐리 생성, 십진 캐리 전파에 대한 정의를 내려 사용한다. 또한 빠른 신호 전달을 위해 다이나믹 (dynamic) 방식을 이용한다[6,7].

본 논문에서는 빠른 연산을 위해 십진 CLA 회로 사용과 입력식의 간략화, 다이나믹 회로를 이용하여 십진 가산기를 제안하고, 제안된 회로의 속도 개선 효과를 보이고 분석 한다. 논문 구성은 다음과 같다. Ⅱ장에서

* 학생회원, ** 정회원, 충북대학교 정보통신공학과
(Department of Information & Communication Engineering, Chungbuk Nat'l University)
접수일자: 2006년7월26일, 수정완료일: 2006년10월30일

는 BCD 가산 방법, Ⅲ장은 빠른 캐리 전달을 위해 다이나믹 구조와 CLA를 사용한 고속 십진 가산기 회로를 제시하고, Ⅳ장은 지연시간, 회로 크기 분석 및 성능을 평가한다. 마지막으로 Ⅴ장에서는 결론을 맺는다.

Ⅱ. BCD Addition

BCD 표현은 십진수 한자리를 4 비트 단위로 나타낸다. BCD 가산의 경우 *Sum*은 입력 A, B와 캐리 입력 C_{in} 합에 의하여 모듈러-10 (modulo-10) 한 나머지로 표현되고 (1), 캐리 출력 C_{out}은 세 입력 A, B, C_{in}의 합이 10 이상인 경우는 1 이 되고 그렇지 않은 경우는 0 이 된다(2).

$$Sum = (A + B + C_{in}) \bmod 10 \tag{1}$$

$$C_{out} = if\,(A + B + C_{in}) \geq 10 \rightarrow 1,\quad otherwise \rightarrow 0 \tag{2}$$

그림 1은 BCD 덧셈에서의 두 가지 예이다.

BCD 덧셈에서 보정은 두 입력의 합이 1010 부터 10011 (10_{10}~19_{10}) 사이에 출력이 발생한 것을 체크하면 된다. 이 때 보정 값은 출력된 BCD 코드값의 합에 0110 (6_{10}) 을 더해준다. 0110 (6_{10}) 을 더해주는 것은 이진수에서 발생되는 캐리와 십진수에서 발생되는 캐리 차이가 6 이기 때문이다.

N-digit의 십진 숫자를 더할 경우 *n* 개의 BCD 가산기가 필요하다. 여기서 리플 캐리의 경우 전단의 캐리 발생 여부에 의해 다음 디지트 계산이 수행되기 때문에 연산 지연시간은 디지트 수 증가에 따라 비례한다. 이러한 리플 캐리 전파에 소요되는 시간을 줄이기 위해서

Addition without carry

	Decimal	BCD code	
Augend	23	0010	0011
Addend	+12	+0001	0010
Sum	35	0011	0101

Addition with carry

	Decimal	BCD code		
Augend	59	0101	1001	
Addend	+25	+0010	0101	
Tentative sum	7(14)	0111	1110	Unallowed code
Correction	↓		+0110	
Carry	1	0001		
Corrected sum	84	1000	0100	

그림 1. BCD 덧셈의 두 가지 예
Fig. 1. Two examples for a BCD addition.

여러 연산 방법들이 제안 되었다.

제안된 방법들 중 빠른 캐리 전달을 위해서 (Carry Save Addition : CSA)와 (Carry propagate addition : CPA)를 사용하여 합과 캐리 계산을 단축하는 방법이 있다[8]. 하지만 빨셈 보정을 해야 하는 단점이 있다.

CLA 방식은 빠른 캐리 출력을 위해서 효과적이다[9,10]. 이 방식은 가산회로에서 각각 캐리 전달, 캐리 전파 신호를 생성하여 디지트 캐리를 병렬로 출력한다. 결과적으로 여러 디지트 가산일 경우 빠른 캐리 전달로 인해 가산지연시간을 줄일 수 있다. 다른 방법으로는 3 초과 코드와 CLA 방식을 이용하여 빠른 연산을 하는 방법도 제안 되었다[10]. 그리고 두 입력 이상 연속해서 가산하는 경우 빠른 계산을 위한 멀티오퍼랜드 가산 방법이 제시 되었다[3].

이 논문에서는 빠른 캐리 출력을 위한 CLA 방식에서 십진 생성, 전파 회로를 제안하고, 입력되는 데이터에 대해 중복되거나 불필요한 부분을 제거하여 회로를 줄이고 빠른 연산 출력을 위해 다이나믹 구조를 적용시켰다.

Ⅲ. 제안된 십진 가산기 구조

1. Carry Generation and Propagation

캐리 예견 회로(carry lookahead Addition : CLA)를 사용하는 것은 캐리 전파에 소요되는 시간을 줄이는 것이다. 십진 가산기에서도 빠른 캐리 전달을 위해서 CLA 방식을 적용한다. 기본적인 캐리 생성 (carry generation), 캐리 전파 (carry propagation) 신호는 (3)과 같이 표현된다. 이 때 A, B는 가산기의 두 입력이다.

$$G = A \cdot B, \qquad P = A + B \tag{3}$$

빠른 신호 생성을 위해 사용된 다이나믹 구조를 적용한 *G*, *P* 구조는 그림 2와 같다.

여기서 *G*, *P* 신호는 BCD에서 사용되는 십진 캐리 생성 (decimal carry propagation), 십진 캐리 전파 (decimal carry generation) 신호를 구성하는데 사용할 뿐만 아니라 합 신호를 만드는 데에도 사용된다.

십진 CLA 회로를 적용하기 위해서 십진 캐리 생성 (decimal carry generation) 회로와 십진 캐리 전파 (decimal carry propagation) 회로가 제안 되었다[10].

제안된 십진 캐리 생성 회로의 캐리 생성조건은 BCD 두 입력 *A*, *B*의 합이 10 이상일 때이다. BCD 입력에 사용되는 수 범위가 0 부터 9 이므로 나머지 입력

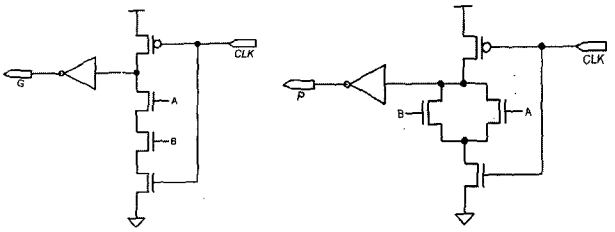


그림 2. (a) Dynamic AND : G (b) Dynamic OR : P
Fig. 2. (a) Dynamic AND : G (b) Dynamic OR : P.

10 에서 15 까지는 dont' care로 처리해야 한다. 또한 BCD 한쪽 입력이 0 인 경우는 캐리가 발생하지 않기 때문에 제외한다. 따라서 0을 제외한 1 에서 9 까지 BCD 입력에 대해, 더해지는 BCD 입력의 합이 10 이상인 조건만 논리곱하면 된다. 캐리 생성 논리식은 (4)와 같이 나타나며, (5)와 같이 간략화 시킬 수 있다.

Decimal carry-generation :

$$\begin{aligned} &= A_3A_0(B_3+B_2+B_1+B_0)+A_3(B_3+B_2+B_1)\\ &\quad +A_2A_1A_0(B_3+B_2+B_1B_0)+A_2A_1(B_3+B_2)\\ &\quad +A_2A_0(B_3+B_2B_1+B_2B_0)+A_2(B_3+B_2B_1)\\ &\quad +A_1A_0(B_3+B_2B_1B_0)+A_1B_3+A_0(B_3B_0) \end{aligned} \tag{4}$$

$$\begin{aligned} &= A_3A_0B_0+A_3B_3+A_3B_2+A_3B_1\\ &\quad +A_2A_1A_0B_1B_0+A_2A_1B_2+A_2A_0B_2B_0\\ &\quad +A_2B_3+A_2B_2B_1+A_1A_0B_2B_1B_0+A_1B_3\\ &\quad +A_0B_3B_0 \end{aligned} \tag{5}$$

간략화 식 (5)에 대하여 (3) 식을 이용하여 (6)과 같이 나타낼 수 있다.

$$\begin{aligned} Cg = {} & G_3+P_3(P_2+P_1)+\\ & G_2(P_1+G_0)+G_0(P_3+G_1P_2) \end{aligned} \tag{6}$$

(6) 식에 의한 다이나믹 십진 캐리 생성 (dynamic decimal carry generation) 회로 구조는 그림 3과 같다.

제안된 십진 캐리 전파 조건은 BCD 입력 A, B와 캐리 입력의 합이 10 이면 충족한다. 여기서도 입력조건 중 10 부터 15 까지는 don't care로 처리한다. 십진 캐리 생성 조건과는 달리 BCD 한쪽 입력이 0 인 경우에는 캐리 전파가 가능하므로 조건으로 사용한다. 캐리 전파 조건은 0 에서 9 까지 BCD 입력에 대해 더해지는 BCD 입력 합이 9 인 조건만 논리곱 한다. BCD 두 입력의 합이 10 이상인 경우는 생성 조건에서 만족하므로 전파 조건은 의미가 없다. 캐리 전파 논리식은 (7)과 같고, 간략화 식은 (8)과 같다.

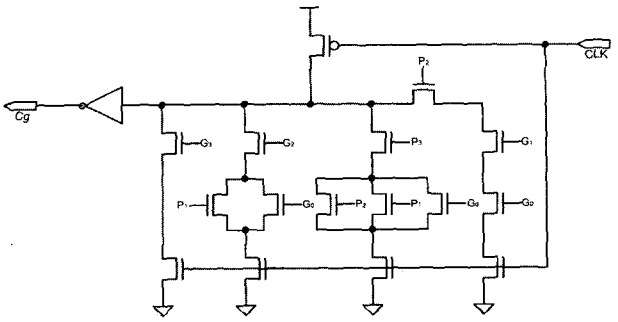


그림 3. 십진 캐리 생성을 위한 다이나믹 십진 캐리 생성 회로, Cg(1-digit)
Fig. 3. Dynamic circuit for a decimal carry generation, Cg(1-digit).

Decimal carry-propagation :

$$\begin{aligned} &= A_3A_0+A_3(B_3+B_2+B_1+B_0)\\ &\quad +A_2A_1A_0(B_3+B_2+B_1)\\ &\quad +A_2A_1(B_3+B_2+B_1B_0)+A_2A_0(B_3+B_2)\\ &\quad +A_2(B_3+B_2B_1+B_2B_0)+A_1A_0(B_3+B_2B_1)\\ &\quad +A_1(B_3+B_2B_1B_0)+A_0(B_3)+(B_3B_0) \end{aligned} \tag{7}$$

$$\begin{aligned} &= A_3A_0+A_3B_0+A_2A_1A_0B_1+A_2A_1B_1B_0\\ &\quad +A_2A_0B_2+A_2B_2B_0+A_1A_0B_2B_1+A_1B_2B_1B_0\\ &\quad +A_0B_3+B_3B_0 \end{aligned} \tag{8}$$

식 (8)은 마찬가지로 (3) 식을 이용하여 (9)와 같이 나타낼 수 있다.

$$Cp = G_1P_2P_0+P_0(G_2+P_3) \tag{9}$$

(9) 식에 의한 다이나믹 십진 캐리 전파 (dynamic decimal carry propagation) 회로 구조는 그림 4와 같다.

N 디지트 (size N) BCD 가산기의 경우, 그룹 (size K) 으로 나누어 그룹 CLA를 사용한다. 그룹 캐리 처

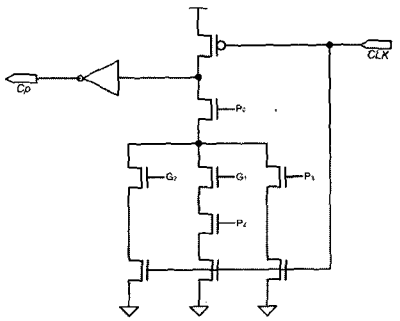
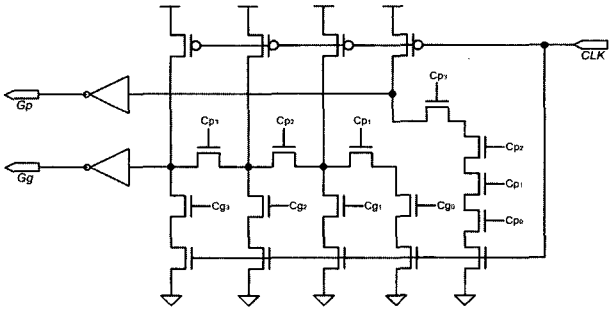
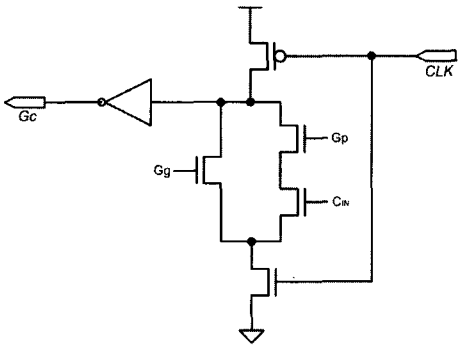


그림 4 십진 캐리 전파를 위한 다이나믹 십진 캐리 전파 회로, Cp(1-digit)
Fig. 4. Dynamic circuit for a decimal carry propagation, Cp(1-digit).



(a) 그룹 캐리 생성, 전파를 위한 다이나믹 회로, *Gg*, *Gp*
(a) Dynamic circuit for a group carry generation and propagation, *Gg*, *Gp*.



(b) 그룹 캐리 전달을 위한 다이나믹 회로, *Gc*
(b) Dynamic circuit for a group carry transmission, *Gc*.

그림 5. 그룹 캐리 처리 회로
Fig. 5. Group carry handling circuit.

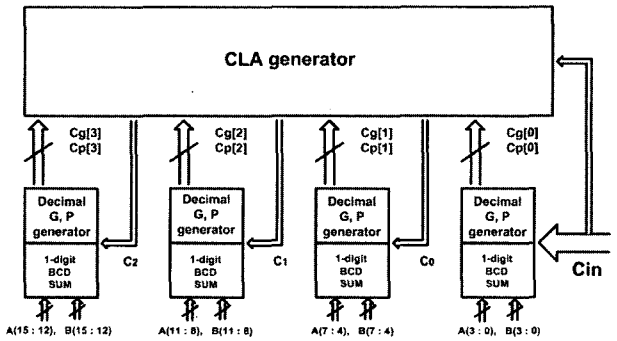


그림 6. BCD CLA 가산기 (4-digit)
Fig. 6. BCD CLA adder (4-digit).

리를 위한 다이나믹 CLA 회로는 그림 5와 같다.

그림 6은 십진 캐리 생성 회로와 십진 캐리 전파 회로가 포함된 1 디지트 BCD 가산기에 그룹 CLA 회로로 이루어진 4 디지트 BCD CLA 가산기 회로이다.

2. Decimal Sum Circuit Design

일반적인 BCD 가산기 합의 출력 지연시간은 전가산기 6 개만큼 소요 된다[5]. 이 절에서는 가산기 합의 출력에 걸리는 지연시간을 줄이기 위해 간략화 된 입력

표 1. 각 합 출력비트에서 1이 나올 수 있는 입력 조건수

Table 1. The number of input combinations of each sum output bit to be 1.

조건 \ 출력비트		S_3	S_2	S_1	S_0
C_{in} = '0'	Sum < 10	19	26	-	-
(일반적)	Sum ≥ 10	1	14	-	-
C_{in} = '1'	Sum < 10	17	22	-	-
(일반적)	Sum ≥ 10	3	18	-	-
C_{in} = '0'	Sum < 10	7	8	-	-
(간략화)	Sum ≥ 10	1	7	-	-
C_{in} = '1'	Sum < 10	5	9	-	-
(간략화)	Sum ≥ 10	2	8	-	-

조건 회로를 제안한다. 가산기 합의 출력 조건을 살펴보면, 각 BCD 입력이 0 ~ 9 이므로 200 가지 출력 조건이 발생한다. 그러나 모든 출력 경우에 대해서 만족할 필요는 없다. 예로 BCD 합에서 최상위 출력 비트인 S_3의 경우 두 입력 조건 합이 8 (1000), 9 (1001), 18 (1 1000), 19 (1 1001)에 대해 성립하면 된다.

표 1은 각 BCD 합 출력 비트의 입력 조건수이다. 여기서 출력 비트인 S_0과 S_1의 경우는 제외한다. S_0의 경우 두 BCD 하위 입력 비트가 같을 경우는 0, 다를 경우는 1 의 출력을 보이므로 입력조건에 대한 논리곱 형태로 나타내지 않고 XOR 값으로 나타낸다. 두 번째 출력 비트 S_1의 경우도 XOR 값으로 나타낼 수 있다.

S_3의 출력 조건을 살펴보자. 먼저 캐리 입력이 0 인 경우와 1 인 경우로 나누어져 있는데, 이것은 캐리가 0 인 경우의 1 인 경우 가산기 합의 출력 값이 다르기 때문에 입력조건을 나누어 표시한 것이다. 만약 캐리 입력이 0 일 때, S_3에 1 로 출력될 수 있는 값은 8 (1000) 과 9 (1001) 그리고 18 (1 1000) 일 것이다. 여기서 이 출력 값을 나타내는 BCD 두 입력 조건을 살펴보면 (0, 8), (0, 9), (1, 7), (1, 8), ……, (8, 0), (8, 1), (9, 0), (9, 9) 이렇게 20 가지의 일반적 조건이 된다.

조건 중 간략화는 일반적 입력 조건에서 중첩되는 부분을 제거하여 나타낸 것이다. 위에서 나온 20 가지 입력 조건 중 (0, 8)과 (8, 0)은 같은 입력 조건이다. 이렇게 중첩되는 입력 조건을 모두 제거할 경우 8 가지 조건으로 줄일 수 있다. 마찬가지로 S_2도 간략화 시킨다. 제안하는 BCD 합 출력 회로는 간략화 된 입력 조건을 사용한다.

간략화 된 조건식들은 (3) 식을 이용하여 아래와 같

이 나타낼 수 있다. 여기서, S_{LT}는 BCD 가산 합이 10 미만인 경우이고 S_{GT}는 10 이상인 경우이다. 합이 10 미만인 경우와 10 이상인 경우로 나눈 것은 출력 회로의 최소화로 빠른 연산 값의 전달을 위함이다.

-- S_3 --

$$S_{LT(3)} = P_3 + G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot (G_0 + P_0 \cdot C_{in})$$

$$S_{GT(3)} = G_3 \cdot G_0 + G_3 \cdot P_0 \cdot C_{in}$$

-- S_2 --

$$S_{LT(2)} = P_2 + G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_{in}$$

$$S_{GT(2)} = G_3 + P_3 \cdot P_2 \cdot (P_1 + G_0 + P_0 \cdot C_{in}) + G_2 \cdot G_1 \cdot (G_0 + P_0 \cdot C_{in})$$

-- S_1 --

$$S_1 = G_1 \oplus P_1 \oplus C_0 \oplus C_{out}$$

-- S_0 --

$$S_0 = G_0 \oplus P_0 \oplus C_{in}$$

그림 7은 제안된 BCD CLA 가산기의 전체 구조를 나타낸다. 합 출력을 위한 부분에도 다이나믹 구조가 사용된다. 회로를 살펴보면, 빠른 캐리 전달을 위한 십진 CLA 회로가 있다. 가산 회로의 경우 S_3, S_2의 경우 출력 값이 10 미만인 경우와 10 이상인 경우로 구분하여 각 회로를 구성하고, S_3은 마지막 MUX에서 C_{out} 출력 조건에 따라 값을 내보낸다. S_2의 경우 각 $\overline{S_3}$과 AND 되어진 값을 MUX를 통해 C_{out} 출력 조건에 따라 값을 내보낸다. S_1, S_0의 경우는 XOR로 구성한다.

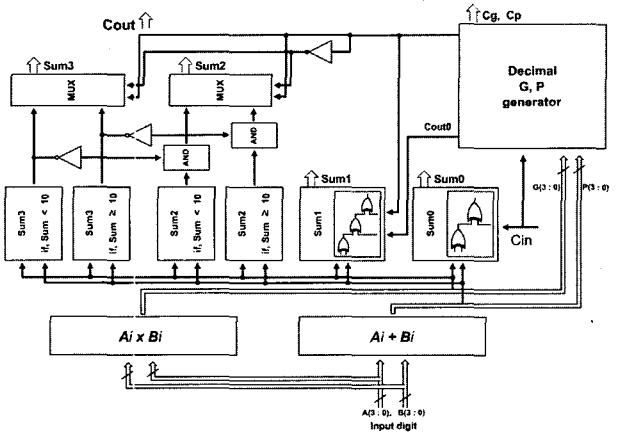


그림 7. 제안된 BCD 가산기 전체 구조 (1-digit)
Fig. 7. Proposed BCD CLA adder structure (1-digit).

IV. 결 과

검증에 사용된 환경으로는 0.18 ㎛ CMOS 공정을 사용하였다. BCD 가산기 회로의 경우, 1 디지트 가산 시 합과 캐리의 출력 지연시간 결과는 그림 8과 같다.

BCD 입력이 16 디지트 (64 bit) 일 때 그룹 CLA 방식을 이용하여 가산할 경우 최대 전달 지연시간을 살펴본다. 지연성분 경로는 그림 9와 같다.

지연 성분은 G[0], P[0] → Decimal G, P generator (Cg[3:0], Cp[3:0]) → CLA generator (Gg[15:0], Gp[15:0]) → Group CLA generator (Gc[47]) → High position CLA generator (Co[59]) → Co[63], Sum(63:60).

최대 지연 전달 경로에 따른 합과 캐리의 출력 지연시간 결과는 그림 10에 나타내었다.

기존의 가산 방식들과 제안한 가산 방식을 비교하였다. 비교를 위해 사용된 가산 방식들은 다음과 같다[3] : Binary Tree (binary), Binary Array (binary), Single Correction Speculation (decimal), Double Correction Speculation (decimal), Non-Speculative addition (decimal). 비교를 위해 사용된 가산방식의 경우 시뮬레이션 환경은 0.18 ㎛ CMOS standard cell 라이브러리를 이용하였고, Synopsys를 이용해 합성한 회로의 가산 지연시간을 나타내었다. 비교 결과로서 4 번 더했을 경우와 8 번 더했을 경우, 각 가산 방식의 지연시간과 회로 면적을 나타내었다.

그림 11는 가산 방식별 지연시간 비교 결과이다. 이진 가산 방식에서 가산 시 걸리는 지연시간에 근접함을 볼 수 있다. 또한 다른 십진 가산 설계 방식들 보다 다이나믹 CLA의 가산 지연시간이 작음을 볼 수 있다.

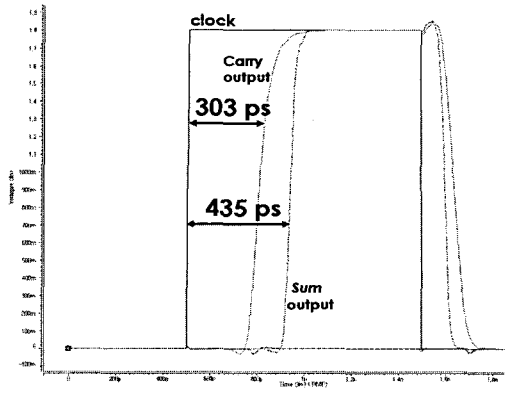


그림 8. 가산 지연시간 (1-digit)
Fig. 8. Adding delay time (1-digit).

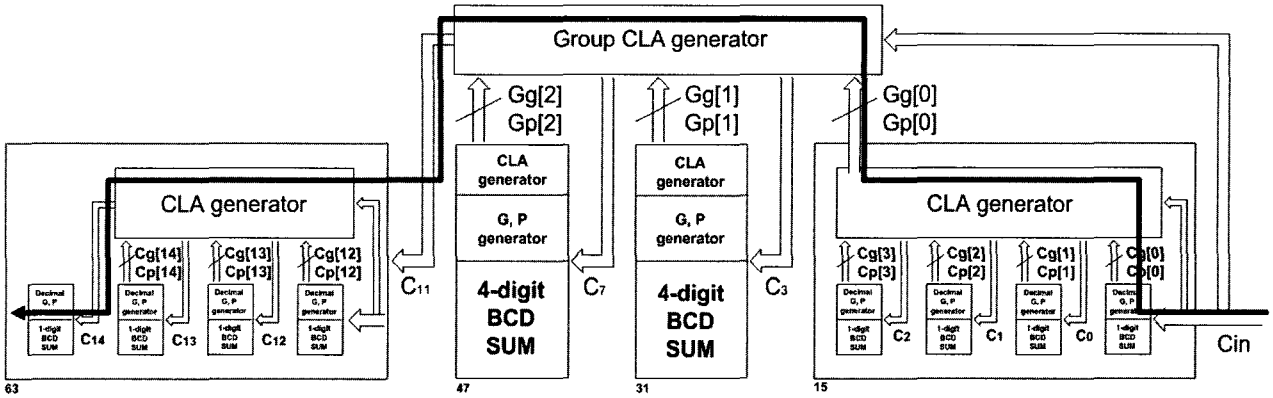


그림 9. 최대 전달 지연 경로 (16-digit)
Fig. 9. Worst-case delay path (16-digit).

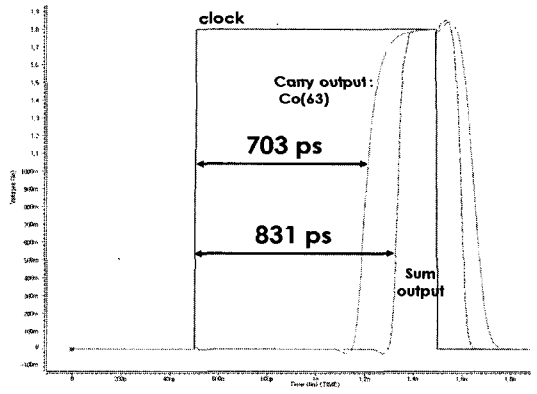


그림 10. 가산 지연 시간 (16-digit)
Fig. 10. Adding delay time (16-digit).

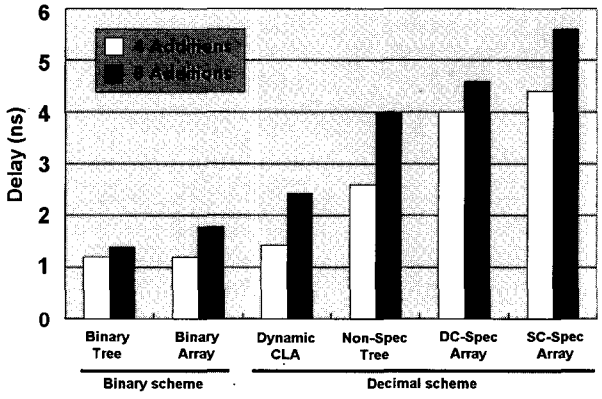


그림 11. 가산방식 별 회로 지연시간 비교
Fig. 11. Comparison of delays for addition schemes.

그림 12에서 면적 비교를 보면 이진 가산 방식에 비해서 큰 것을 볼 수 있는데, 이는 BCD 형태의 경우 보정회로와 비트 사용 제한에 따른 레지스터의 크기 때문이다. 다른 십진 가산 방식과 비교해 보면 Non-Spec Tree 방식에서 4 번 더했을 경우를 제외하고 면적의 차이가 비슷함을 확인 할 수 있다.

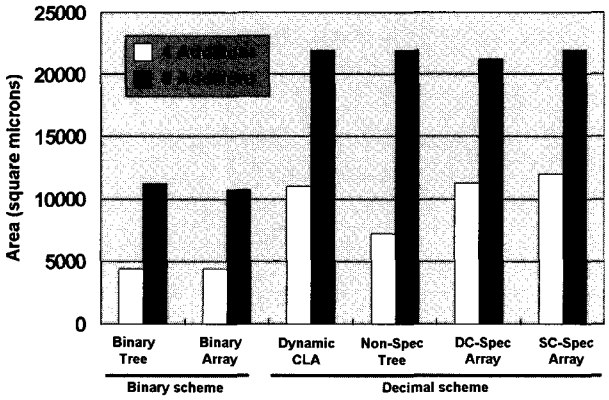


그림 12. 십진 가산 방식별 회로 면적 비교
Fig. 12. Comparison of area of addition schemes.

Ⅳ. 결 론

본 논문에서는 고속 십진 가산을 위한 회로 설계 방법을 제시하고, 0.18 ㎛ CMOS 공정을 이용하여 지연시간을 분석하였다. 절단오차가 생기지 않는 십진 가산 방식에서 빠른 캐리 전달을 위해 CLA 방식을 사용한다. 이 때 BCD 가산기에 적합한 CLA 회로 사용을 위해서 십진 캐리 생성, 전파 회로를 제안하였다. 또한 입력식의 간략화 및 다이나믹 구조를 사용함으로써 빠른 신호 전달이 가능 하였다. 16 디지트 (64 bit) 연산에서 최대 전달 가산 지연 시간은 831 ps 이었다. 다른 십진 가산 설계 방식과 비교 했을 때도 연산에 따른 지연시간이 작음을 확인 할 수 있었다. 이러한 십진 가산 회로는 절단 오차가 없는 십진 컴퓨터, 금융, 인터넷 기반 응용 분야에서 중요한 구성 요소로서 작용할 수 있다.

참 고 문 헌

[1] Bibliography of material on Decimal Arithmetic, http://www2.hursley.ibm.com/decimal/decifaq1.html, Sept. 2002.

[2] M. F. Cowlishaw, et al., "A decimal floating-point specification," *Proc. 15th IEEE Symposium on Computer Arithmetic*, pp. 147-154, June 2001.

[3] R. D. Kenney and M. J. Schulte, "High-speed Multioperand decimal adders," *IEEE Transac tions on Computers*, vol. 54, no. 8, pp. 953-963, August 2005.

[4] F. Busaba, C. Krygowski, E. Schwarz, W. Li and S. Carlough, "IBM z900 Decimal Arithmetic Unit," *Proc. the 35th Asilomar Conf. on Signals, Systems, and Computers*, pp. 1335-1339, Nov. 2001.

[5] S. Hermann, *Decimal Computation*, Wiley-Inter Science Publication, 1974.

[6] Ramchan Woo, Se-Joong Lee and Hoi-Jun Yoo, "A 670 ps, Dynamic Low-Power Adder Design," *ISCAS 2000- IEEE International Symposium on Circuit and System*, vol. 1, pp. 28-31, May 2000.

[7] R. J. Baker, H. W. Li, and D. E. Boyce, *CMOS Circuit Design, Layout, and Simulation*, IEEE PRESS, 1998, ch. 14.

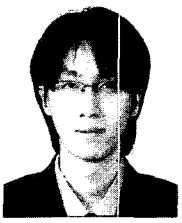
[8] R. K. Richards, *Arithmetic Operations in Digital Computers*. D. Vanostrand Company, Inc., 1955.

[9] M. S. Schmookler and A. Weinberger, "High speed decimal addition," *IEEE Transactions on Computers*, vol. C-20, no. 8, pp. 862-866, August 1971.

[10] J. Choi and Y. You, "An excess-3 code carry lookahead design for high-speed decimal addition," *IEEK J.*, vol. 40 CI, no. 5, pp 241-249, Sept. 2003.

[11] Y. You, Y. Kim and J. Choi, "Dynamic decimal adder circuit design by using the carry lookahead," *Proc. Design and Diagnostics of Electro nic Circuit and Systems*, pp.244-246, Apr. 2006.

저 자 소 개



최 종 화(학생회원)

1996년 충북대학교 반도체공학과 학사 졸업.

2002년 충북대학교 정보통신 공학과 석사 졸업.

2005년 현재 충북대학교 정보통신 공학과 박사과정.

<주관심분야 : VLSI 설계, 연산회로 설계, 암호회로 설계>

유 영 갑(정회원)

1975년 서강대학교 전자공학과 학사 졸업.

1975년~1979년 국방과학연구소 연구원.

1981년 Univ.of Michigan, Ann Arbor 전기전산학과 공학석사 졸업.

1986년 Univ.of Michigan, Ann Arbor 전기전산학과 공학박사 졸업.

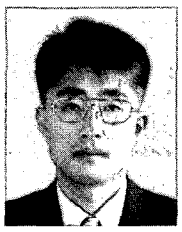
1986년~1988년 금성반도체 (주) 책임 연구원.

1993년~1994년 아리조나 대학교 객원교수.

1998년~2000년 오레곤 주립대학교 교환교수.

1988년~현재 충북대학교 정보통신공학과 교수.

<주관심분야 : VLSI 설계 및 테스트, 고속인쇄회로 설계, 암호학>



김 용 대(정회원)

1990년 충북대학교 정보통신 공학과 학사 졸업.

1993년 충북대학교 컴퓨터공학과 석사 졸업.

1989년~1998년 신흥기술연구소 팀장.

2000년~현재 충북대학교 정보통신공학과 박사과정

<주관심분야 : Computer arithmetic, ASIC 설계, 암호 시스템>