Journal of The Institute of Electronics and Information Engineers Vol.53, NO.11, November 2016 http://dx.doi.org/10.5573/ieie.2016.53.11.056
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2016-53-11-8

다중 피연산자 십진 CSA와 개선된 십진 CLA를 이용한 부분곱 누산기 설계

(Design of Partial Product Accumulator using Multi-Operand Decimal CSA and Improved Decimal CLA)

이 양*, 박 태 신*, 김 강 희*, 최 상 방**

(Yang Lee, TaeShin Park, Kanghee Kim, and SangBang Choi©)

요 약

본 논문에선 병렬 십진 곱셈기의 축약 단계의 면적과 지연시간을 감소시켜 성능을 향상시키기 위해 다중 피연산자 십진 CSA과 개선된 십진 CLA를 이용한 트리 구조를 제안한다. 제안한 부분곱 축약 트리는 십진수 부분곱에 대해 다중 피연산자 십진 CSA를 사용하여 빠르게 부분곱을 축약한다. 각 CSA에서는 리코딩에 입력의 범위를 제한함으로써 가장 간단한 리코더 로직을 얻는다. 그리고 각 CSA는 특정한 아키텍처 트리의 특정한 위치에서 범위가 제한된 십진수를 더하기 때문에 부분곱 축약 단계의 연산을 효율적으로 수행할 수 있다. 또한, 사용되는 십진 CLA의 로직을 개선하여 BCD 결과를 빠르게 얻을 수 있다. 제안한 십진 부분곱 축약 단계의 성능의 평가를 위해 Design Compiler를 통해 SMIC사의 180nm CMOS 공정 라이브러리를 이용하여 합성하였다. 일반 방법을 이용하는 축약 단계에 비해 제안한 부분곱 축약 단계의 지연시간은 약 15.6% 감소하였고 면적은 약 16.2% 감소하였다. 또한 십진 CLA의 지연시간과 면적이 증가가 있음에도 불구하고 전체 지연시간과 전체 면적이 감소함을 확인하였다.

Abstract

In this paper, in order to reduce the delay and area of the partial product accumulation (PPA) of the parallel decimal multiplier, a tree architecture that composed by multi-operand decimal CSAs and improved CLA is proposed. The proposed tree using multi-operand CSAs reduces the partial product quickly. Since the input range of the recoder of CSA is limited, CSA can get the simplest logic. In addition, using the multi-operand decimal CSAs to add decimal numbers that have limited range in specific locations of the specific architecture can reduce the partial products efficiently. Also, final BCD result can be received faster by improving the logic of the decimal CLA. In order to evaluate the performance of the proposed partial product accumulation, synthesis is implemented by using Design Complier with 180 nm COMS technology library. Synthesis results show the delay of the proposed partial product accumulation is reduced by 15.6% and area is reduced by 16.2% comparing with which uses general method. Also, the total delay and area are still reduced despite the delay and area of the CLA are increased.

Keywords : Parallel Decimal Multiplication, IEEE 754-2008, Multi-Operand

Ⅰ. 서 론

최근의 대부분 컴퓨터는 이진 부동소수점(Binary Floating-Point, BFP) 연산을 지원하지만 높은 정밀도를 요구하는 어플리케이션 때문에 십진 부동소수점(Decimal Floating-Point, DFP) 연산도 사용한다. 특히 항공 시스템, 금융, 경제 등 어플리케이션에서 충분히 기능을 지원한다. 이는 일부의 이진 부동소수점 수와 십진 부동소수점 수 사이에 부정확한 매핑이 존재하기 때문이다. 예를 들어, 십진수 0.2를 이진수로 표현하면 무한하게 반복되는 이진수가 필요하기 때문에 그 십진수에 대한 라운딩을 제공하면 그에 따라 치명적인 오차

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
©Corresponding Author (E-mail : sangbang@inha.ac.kr)
Received ; June 1, 2016 Revised ; October 29, 2016
Accepted ; November 1, 2016

가 발생한다. 따라서 정확하게 십진수를 구현해야 하는 세금계산, 전자상거래, 환전 등의 상업업무, 그리고 항공우주 등의 과학적인 어플리케이션에서 십진 부동소수점 연산기와 그와 관련있는 연산에 대한 연구가 진행되고 있다[1~3].

이런 어플리케이션들에서 십진 연산을 위해 여러 가지 소프트웨어 패키지는 등장했지만 소프트웨어가 처리하는 과정이 느릴 뿐만 아니라 에러가 발생할 수도 있기 때문에 정확하고 높은 성능이 필요한 분야에선 부적합하다. 따라서 앞서 말한 요구 조건을 충족하며, 소프트웨어 보다 고성능을 발휘 할 수 있는 하드웨어 측면에서 십진 연산기를 설계하여 마이크로프로세서에 내장하는 것이 더 가치가 있다. 이런 십진 부동소수점 연산의 가치가 부각됨에 따라서 미국 전기 전자 학회(Institute of Electrical and Electronics Engineers, IEEE)에서는 이미 존재하는 이진 부동소수점 연산만 다룬 IEEE-754 standard[4]를 수정하여 십진 부동소수점 연산을 도입한 IEEE 754-2008 standard[5]를 제정했다.

십진 연산을 정확하게 진행하기 위해서는 십진수를 이진수로 표시할 수 있는 BCD(Binary Coded Decimal) 코드가 필요하다. BCD 코드는 0 ~ 9, 즉 0000 ~ 1001 범위만 표현할 수 있기 때문에 redundancy를 갖게 되지만 사용자 지향의 십진수를 4비트 이진수로 표현할 수 있는 측면에서 이용가치가 높다. 4비트 부호화 자릿수는 효율적으로 십진수를 연산할 수 있지만 배수 생성에서 로직이 복잡하고 최종 합 변환 단계가 필요하기 때문에 전체적인 효율이 좋지 않다. 최근의 연구는 주로 각 자리의 가중치에 따라 8421 BCD, 4221 BCD, 5211 BCD, 5421 BCD로 구별되는 BCD 코드를 이용하는 직렬[6] 또는 병렬 십진 곱셈기에 대해 더 많은 관심을 가지고 있다. 앞서 말한 코드들은 간단한 시프트(shift)와 디코드(decode)를 통해 피연산자의 배수를 얻을 수 있는 바람직한 성질을 가지고 있기 때문이다[7]. 그리고 redundant BCD 코드인 excess-3 코드를 이용하는 빠른 십진 곱셈기에 대한 연구도 있다[8].

병렬 16×16 십진 곱셈기의 부분곱 축약 단계(Partial Product Accumulation)는 주로 8421 BCD 코드에 근거하기 때문에 일반 연산기가 가지고 있는 부분곱 축약 결과를 BCD 코드로 표현하는 변환 단계가 없어진다. 이때 연산을 더 빠르게 하기 위해서 생성된 부분곱을 더 효율적으로 축약하는 것이 핵심이다. 본 논문에서는 다중 피연산자 십진 CSA(Carry Save Adder)와 십진 CLA(Carry Lookahead Adder)를 이용하여 성능을 향상시킬 수 있는 트리(tree) 구조를 제안한다. 트리 구조에서는 다중 피연산자 십진 CSA를 특정한 자리에서 설계하여 축약 단계의 연산을 효율적으로 수행할 수 있다. 또한 특정한 십진 CLA를 이용함으로써 면적과 지연시간을 더 줄일 수 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 병렬 십진 곱셈기에 대해 자세히 알아보고 본 논문의 기초가 되는 기존의 십진 CLA 트리 구조를 이용한 병렬 십진 곱셈기 축약 단계와 일반 십진 CSA와 카운터를 이용한 축약 단계에 대해 설명하고, Ⅲ장에서 제안하는 병렬 십진 부분곱 축약 방법에 대해 설명한다. 그 다음에 Ⅳ장에서는 제안하는 병렬 십진 부분곱 축약 단계를 구현해 이론을 검증하고 ASIC 환경에서 면적 및 지연시간에 대해 기존의 논문과 비교하여 분석한다. 마지막으로 본 논문의 연구 내용을 정리한 후 Ⅴ장에서는 최종 결론을 제시한다.

Ⅱ. 관련 연구

이 장에선 기본 병렬 십진 곱셈기의 과정 및 BCD 코드를 이용한 부분곱 축약 단계에 대해 알아본다.

1. 병렬 십진 곱셈기 개요

일반 십진 곱셈기는 부분곱 생성(partial product generation), 부분곱 축약(partial product accumulation), 최종 합 3단계로 구성된다.

부분곱 생성 단계는 피승수에 승수의 각 자릿수를 곱하여 부분곱을 생성하는 단계이다. 십진 곱셈의 경우에 승수의 각 자릿수는 0 ~ 9중에 하나의 값을 가지므로, 피승수 X에 대해 $0X \sim 9X$의 배수(multiple)를 병렬로 전부 생성해 낸 후에 승수의 각 자릿수에 맞게 적당한 배수를 선택하는 방법을 주로 이용한다. $0X$와 $1X$는 와이어(wire) 연결만으로 피승수의 배수를 얻을 수 있다. $2X$와 $5X$ 배수는 간단한 로직 회로를 사용해 얻을 수 있으며, $4X = 2X \times 2X, 8X = 2X \times 2X \times 2X$도 $2X$ 배수를 생성하는 방법으로 얻을 수 있다. 비교적 얻기 어려운 $\{3X, 6X, 7X, 9X\}$배수는 캐리 전달 지연이 있는 덧셈을 통해서 얻을 수 있다. 하지만 직접 인코딩 하거나[9] BCD 코드의 성질을 이용하면 캐리 지연이 있는 덧셈보다 더 효율적으로 배수를 얻을 수 있다. 이에 따라 선택된 부분곱은 곱해진 승수의 자릿수를 고려하고 다음 단계에 정렬해야 한다.

부분곱 축약단계에서 이미 얻은 승수의 각 자릿수에

해당하는 부분곱은 각 자릿수의 값을 고려하여 더해져야 한다. 승수의 자릿수만큼의 다중 피연산자 덧셈(multi-operand addition) 연산을 실행해야 함에 따라 연산 시간과 면적이 가장 큰 단계이다. 보통의 경우에는 캐리 전달 지연을 피하기 위해서 이진 CSA(Carry Save Adder) 트리를 십진수에 맞게 변형하여 연산을 실행하거나 십진 가산기를 이용한 CS(Carry Save) 구조를 사용한다. 부분곱 축약 단계를 통하여 합 벡터(sum vector)와 캐리 벡터(carry vector)를 생성할 수 있고, 최종합 단계에서 가산기를 이용하여 이 두 벡터를 더하면 최종 결과값을 얻을 수 있다.

최종합 단계에서는 십진 덧셈을 위한 BCD 가산기를 이용하거나, 이진 가산기에 사전교정(pre-correction)과 후교정(post-correction)을 이용해 최종곱(final product)을 얻는다.

2. 일반 방법을 이용한 부분곱 축약 단계

이 절에서는 본 논문의 바탕이 되는 M. Zhu의 십진 CLA을 이용한 부분곱 축약단계[10]에 대해 설명하고, 제안하는 방법의 일반화 가능성을 검증하기 위해 T. Lang의 십진 CSA와 카운터로 구성된 트리를 이용한 부분곱 축약단계[11]에 대해서 살펴본다.

가. 십진 덧셈

들어가기에 앞서 십진 덧셈에 대해 먼저 알아보도록 하겠다. 논문에서 사용하는 4비트 십진 피연산자 A와 B는 $[0,9]$의 범위를 가진다. C_{out}은 한 자릿수의 캐리이다. C_{in}은 이전 자릿수의 캐리이다[12].

$$\begin{aligned} K &= G_8 + P_8P_4 + P_8P_2 + G_4P_2, \\ L &= P_8 + G_4 + P_4G_2, \\ C_{out} &= K + LC_1. \end{aligned} \tag{1}$$

K, L는 가중치가 2, 4, 8인 열(column)의 캐리이고 $C_1 = G_1 + P_1C_{in}$은 가중치가 1인 열에서 오는 캐리이다. 가중치가 1인 열을 0으로 가정하고 가중치가 2, 4, 8인 열의 합이 10 이상인 경우에는 K가 1이며 8 이상인 경우에는 L이 1이다. 그리고 이진 캐리 전달신호 P_i는 $P_i = A_i + B_i$, 이진 캐리 생성 신호 G_i는 $G_i = A_iB_i$, 신호 H_i는 $H_i = A_i \oplus B_i$로 정의된다. 이에 따라 다음 수식을 얻을 수 있다. 합 S_i는 다음에서의 식(2)과 같이 나타낸다.

$$\begin{aligned} S_1 &= (A_1 \oplus B_1) \oplus C_{in}, \\ S_2 &= H_2 \oplus C_1 \oplus C_{out}, \\ S_4 &= \overline{P_4}G_2 + \overline{P_8}H_4\overline{P_2} + (\overline{P_8}\,\overline{P_4}P_2 + \\ &\quad G_4G_2 + P_8P_4)C_1 + (G_8 + H_4H_2)\overline{C_1}, \\ S_8 &= (G_8 + H_4H_2)\overline{H_8}C_1 + L\overline{K}\,\overline{C_1}. \end{aligned} \tag{2}$$

식 (1)과 식 (2)를 이용하여 먼저 한 자릿수의 십진 덧셈기를 만들고, 그룹(group) 캐리 전달 신호 $P_{i:j}$와 생성 신호 $G_{i:j}$를 생성하여 4개 자릿수의 CLA를 1개 자릿수의 CLA를 통해 생성한다. 그러면 마지막으로 필요한 자릿수의 CLA를 얻을 수 있다. 이 십진 덧셈은 이진 합을 생성할 필요 없이 최종 십진 합을 직접 얻을 수 있고, 십진 부분곱 축약 단계에서 부분곱을 더 효율적으로 줄일 수 있다.

나. M. Zhu의 축약 단계

부분곱 생성 단계에서 전가산기를 통해 생성되어진 부분곱의 각 자릿수는 $[0,9]$의 범위를 갖는다. 부분곱 생성 단계에서 만들어진 부분곱 배열은 $(n+1)$자릿수의 부분곱 n개가 있다. 16-digit 곱셈기에 대해 17-digit, 18-digit, 20-digit, 24-digit 가산기 4 가지를 사용하여 부분곱을 축약한다. 각각의 가산기는 십진 덧셈을 이용하여 만들어진 십진 CLA이다.

부분곱 축약 단계는 모두 4개의 단계가 있다. 첫 번째 단계에서는 8 개의 17-digit CLA을 이용하여 17개의 자릿수를 가지는 부분곱 16개를 17개의 자릿수를 가지는 부분곱 8개로 줄인다. 두 번째 단계에서는 4개의 18-digit CLA를 통해서 첫 번째 단계의 결과를 18개의 자릿수를 가지는 부분곱 4개로 줄인다. 세 번째 단계에서는 2개의 20-digit CLA를 이용하여 20개의 자릿수를 가지는 부분곱 2개를 얻는다. 마지막 단계에서 24-digit CLA는 이전 단계에서 오는 곱을 더하여 최종 결과를 생성한다.

다. T. Lang의 축약 단계

부분곱 생성 단계에서 십진 CSA를 이용하여 생성되어진 BCD 디짓(digit) 부분의 각 자릿수는 $[0,9]$의 범위를 가지고, 생성되어진 캐리 부분의 각 자릿수는 $[0,1]$의 범위를 갖는다. 부분곱은 CS(carry save) 포맷으로 표현되어 모두 $(n+1)$ 개의 자릿수의 부분곱 $2n$개를 가지고 있다. 부분곱 축약 단계에서는 첫 번째

단계인 8개의 $(n+2)$-digit 십진 CSA가 필요하다. n-digit 십진 CSA의 연산 과정은 n개의 CS 피연산자와 n개의 BCD 피연산자를 더하여 n개의 CS 결과를 생성하는 것이다. 더하지 않은 캐리 벡터를 축약하기 위해 8개의 자릿수의 캐리 카운터(carry counter, CC) m개를 이용하여 각 자릿수의 최댓값이 8인 결과를 얻을 수 있다.

16-digit 곱셈기에 대해 32개의 부분곱을 더하기 위하여 십진 CSA와 CC를 배치하고 부분곱 축약을 수행한다. 첫 번째 단계에서는 32개의 부분곱을 처리하는 8개의 18-digit 십진 CSA를 포함하고 있으며 결과가 $s1_8$인 32-digit CC 하나를 추가한다. 첫 번째 단계를 수행하여 얻어낸 8개의 CS 결과는 두 번째 단계에서 4개의 24-digit 십진 CSA를 이용하여 처리된다. 이때 처리하지 않은 캐리는 $s1_{odd}$로 표시한다. 세 번째 단계와 네 번째 단계의 수행 과정은 두 번째와 동일하게 수행한다. 수행 과정에서 더하지 않은 각각의 캐리는 $s2_{odd}$, $s3_{odd}$로 표시한다. 다섯 번째 단계에서 32-digit 십진 CSA는 4단계의 결과인 $s4_s$, $s4_c$와 1단계의 $s1_8$을 입력받아 축약하고, 32-digit 십진 CC는 $s1_{odd}$, $s2_{odd}$, $s3_{odd}$를 한꺼번에 축약한다. 마지막 단계에서는 최종 BCD-digit로 변환해야 하는 덧셈 단계의 입력인 p_s, p_c를 얻는다.

부분곱 축약 단계에서 각 자리의 디짓과 캐리의 합은 $[0,10]$ 범위이므로 합 s_i가 10이면 캐리가 발생하고, 9이면 캐리전달이 발생한다. 이와 동시에 10에 대해 나머지를 찾는 동작을 실행하고, 결과인 $s0_i$, $s1_i$를 얻는다. 캐리 프리픽스 방법을 이용해 i번째 자릿수의 캐리 c_{i+1}을 결정한다. 이에 따라 최종 BCD 결과 값은 $c_i=0$이면 캐리가 발생하지 않아 $s0_i$로 결정되고, $c_i=1$이면 이전 자리에서 캐리가 발생하여 $s1_i$로 결정된다. 이 논문에서 특별한 설명이 없으면 BCD는 8421 BCD를 가리킨다.

Ⅲ. 다중 피연산자 십진 CSA과 특정한 CLA를 이용한 병렬 부분곱 축약 방법

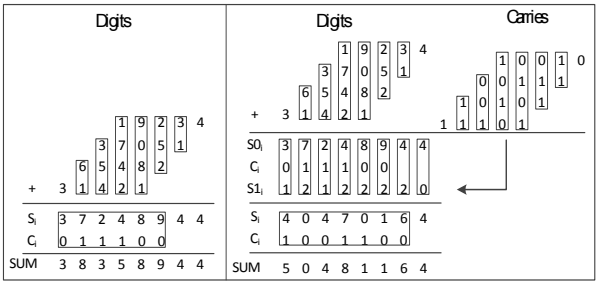
이번 장에서는 제안하는 16 자릿수의 병렬 부분곱 축약 방법에 대해 설명한다. 2.2절에서 설명한 기존의 일반 방법을 이용한 병렬 십진 축약 단계를 바탕으로 성능을 향상시키기 위한 방법을 제안한다.

1. 개요

제안하는 구조는 내부적으로 4비트 십진수를 이용해 성능을 향상시키는 병렬 십진 부분곱 축약 단계이다.

M. Zhu의 십진 부분곱에 대한 부분곱 축약 단계에서는 $(n+1)$자릿수를 가지는 n개의 부분곱 축약을 수행한다. 1단계 축약에서 다중 피연산자 십진 CSA를 이용하여 한 번에 최대 4개의 십진수를 더한 후 리코더를 통해 합의 최댓값을 고려하여 캐리 $C_{i+1}\in[0,3]$과 남은 합 $S_i\in[0,9]$로 리코딩을 수행한다. 2단계와 3단계 축약에서 1단계의 방법과 마찬가지로 이전 단계 축약에서 리코딩된 합 벡터와 캐리 벡터를 4개의 피연산자씩 더한 후 세 번째 리코더를 통해 하나의 합 벡터와 캐리 벡터를 구한다. 마지막으로 하나의 개선된 30-digit 십진 CLA를 통해 최종 결과 값을 얻는다. 연산에 대한 예는 그림 1(a) 과 같다.

T. Lang의 부분곱에 대해 $(n+1)$자릿수의 십진수 n개와 $(n+1)$자릿수의 캐리 벡터 n개를 축약한다. 1단계 축약에서 십진수 부분은 이전 축약 단계의 구조와 같이 더해지고, 캐리 부분에 대해 $n\in[2,8]$개의 자릿수를 가지는 캐리 카운터를 사용하여 십진 합 $H\in[0,8]$을 구한다. 다중 피연산자 십진 CSA보다 CC의 결과를 더 빠르게 생성할 수 있기 때문에 2단계, 3단계, 4단계 축약에서 1단계의 CC에서 생성된 십진수를 특정한 자리에 붙이고, 4개씩, 3개씩 또는 2개씩으로 더하여 네 단계를 통해 하나의 합 벡터와 캐리 벡터를 얻는다. 마지막으로 개선된 32-digit 십진 CLA를 이용한다. 연산에 대한 예는 그림 1(b)과 같다.



(a). For the partial products of M. Zhu (b). For the partial products of T. Lang

그림 1. 4×4 부분곱 축약의 예
Fig. 1. Example of 4×4 partial product accumulation.

2. 기초가 되는 부분곱의 축약 단계

최소의 단계와 최소의 CSA를 사용하기 위해 축약의 3단계를 그림 2와 같이 배치한다. 부분곱 생성 단계에서 만들어진 $(n+1)$자릿수의 십진 부분곱 n개를 가산

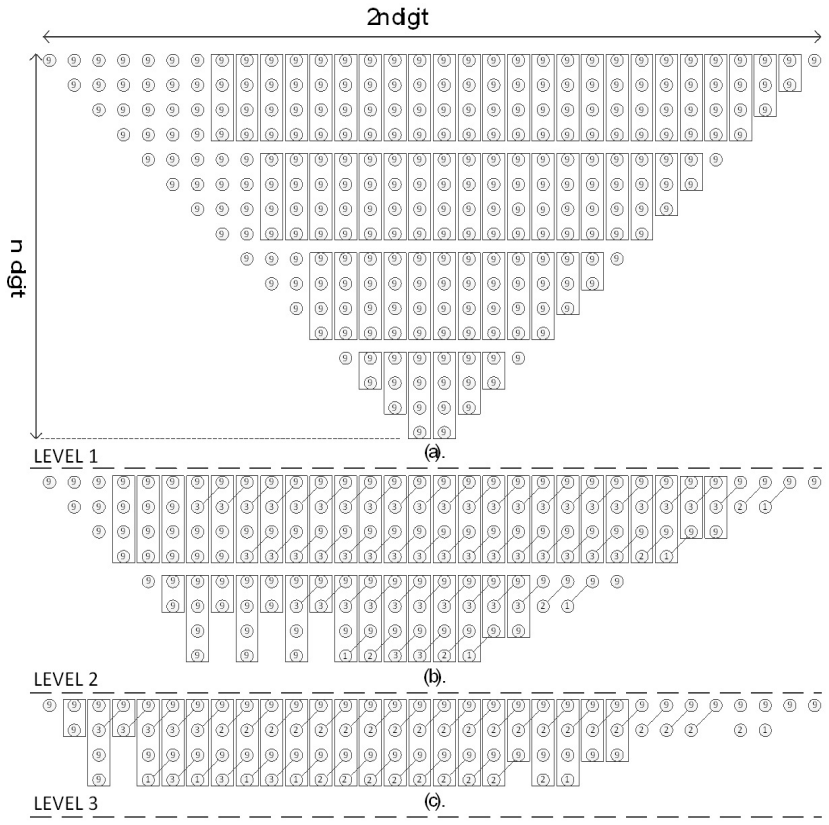


그림 2. 제안하는 3단계 부분곱 축약 트리
Fig. 2. Proposed 3 levels of partial product accumulation tree.

하기 위해 그림 2(a)와 같이 부분곱의 자릿수를 정리하여 17번째 열 이후의 부분은 위로 배열한다. 1단계인 그림 2(a)는 동시 연산 가능한 4개의 CSA 레이어(layer)로 구성된다. 그림 2(a)에서 확인 할 수 있듯이 각 자릿수의 최댓값은 하얀 원 내의 9이다. 이에 따라 4개의 자릿수의 CSA의 결과의 범위는 $[0,36]$, 3개의 자릿수에는 $[0,27]$, 2개의 자릿수에는 $[0,18]$과 같다. 사각형마다 하나의 다중 피연산자 십진 CSA를 의미한다. 4개의 자릿수를 가지는 사각형은 4to2 CSA, 3개의 자릿수에는 3to2 CSA, 2개의 자릿수에는 2to2 CSA로 정의된다.

1단계 연산 후 생성된 4개의 십진수와 캐리, 그리고 이용하지 않은 부분곱을 축약하는 단계인 부분곱 축약 단계의 2단계는 1단계의 연산과 동일한데 그림 2(b)와 같이 2개의 CSA 레이어를 받는다. 그리고 4개의 자릿수 CSA 결과 범위는 $[0,30],[0,24],[0,23],[0,22]$를 추가하고 3개의 자릿수에는 $[0,21]$, 2개의 자릿수에는 $[0,12]$를 추가한다. 3단계의 로직은 1단계, 2단계와 마찬가지로 수행되지만 4개의 자릿수를 가진 CSA의 결과 범위에도 $[0,21]$, 3개의 자릿수에도 $[0,20]$이 추가된다.

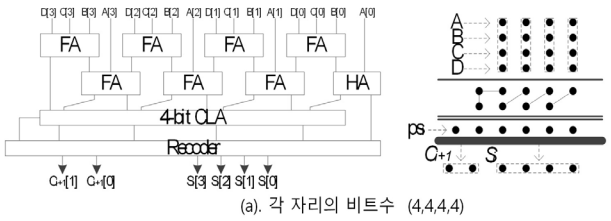
제안하는 부분곱 축약 단계의 4to2 CSA 셀(cell)에 해당하는 하드웨어 구조와 점 표시법은 그림 3과 같다. 그림 3(a), 3(b), 3(c), 3(d)의 구조는 비슷하지만 4개의 자릿수의 최댓값을 표현하기 위해 필요한 비트의 수는 $(4,4,4,4)$, $(4,2,4,4)$, $(4,2,4,2)$, $(4,2,4,1)$로 나타난다. 이는 로직의 최소화를 시킬 수 있다. 점 표시법에서 점선 사각형들은 반가산기, 전가산기, 4:2 컴프레서를 의미하며, 두 줄 선은 이진 CLA 로직, 굵은 선은 리코더를 의미한다. 제안하는 부분곱 축약 단계의 3to2 CSA 셀, 2to2 CSA 셀에 해당하는 하드웨어 구조와 점 표시법은 그림 4, 그림 5와 같다.

그림 4(a), 4(b)에서 3개의 자릿수의 최댓값을 표현하기에 필요한 비트의 수는 $(4,4,4)$, $(4,2,4)$이다. 그림 5(a), 5(b)에서 2개의 자릿수의 최댓값을 표현하기에 필요한 비트의 수는 $(4,4)$, $(4,2)$이다.

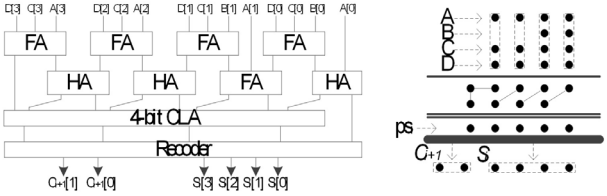
그림 3, 4, 5에서 확인 할 수 있듯이 각 CSA의 연산이 끝날 때마다 리코더를 통해 각각의 연산 가능한 범위를 고려하여 캐리 C_{i+1}과 합 S_i을 얻는다.

중간 합 ps_i의 최대 표현 범위가 다르기 때문에 최댓값이

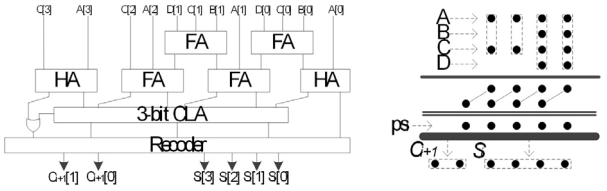
무엇이냐에 따라 아래와 같은 논리식을 사용하여 리코딩을 수행한다.



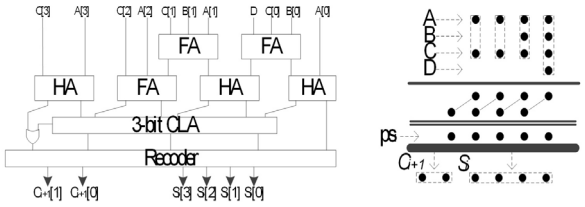
(a). 각 자리의 비트수 (4,4,4,4)



(b). 각 자리의 비트수 (4,2,4,4)

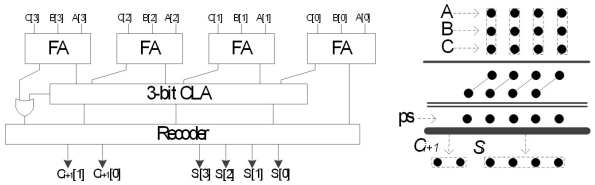


(c). 각 자리의 비트수 (4,2,4,2)

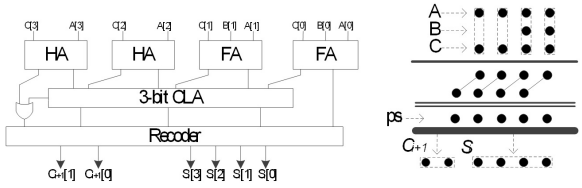


(d). 각 자리의 비트수 (4,2,4,1)

그림 3. 4to2 다중 피연산자 십진 CSA 셀
Fig. 3. 4to2 multi-operand decimal CSA cell.

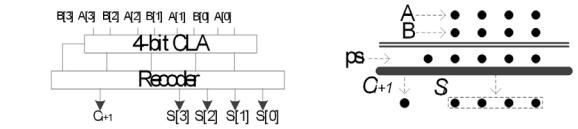


(a). 각 자리의 비트수 (4,4,4)

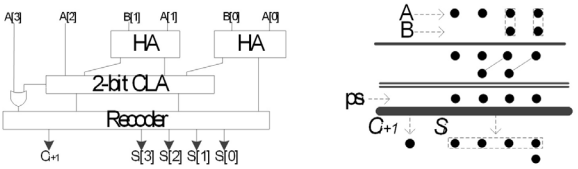


(b). 각 자리의 비트수 (4,2,4)

그림 4. 3to2 다중 피연산자 십진 CSA 셀
Fig. 4. 3to2 multi-operand decimal CSA cell.



(a). 각 자리의 비트수 (4,4)



(b). 각 자리의 비트수 (4,2)

그림 5. 2to2 다중 피연산자 십진 CSA 셀
Fig. 5. 2to2 multi-operand decimal CSA cell.

(1) 최댓값 36에 대한 리코딩

$$
\begin{aligned}
&C_{i+1}[1] = X_5 + X_4X_3 + X_4X_2,\\
&C_{i+1}[0] = X_4\overline{X_3}\,\overline{X_2} + X_5 + \overline{X_4}X_3X_2 + \\
&\qquad X_3X_2X_1 + \overline{X_4}X_3\overline{X_2}X_1,\\
&S_i[0] = X_0,\\
&S_i[1] = X_5\overline{X_1} + \overline{X_3}X_2X_1 + \overline{X_4}X_3X_2\overline{X_1} + \\
&\qquad X_4\overline{X_3}\,\overline{X_2}\,\overline{X_1} + X_4X_3\overline{X_2}X_1 + \overline{X_5}\,\overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = X_5X_1 + \overline{X_4}\,\overline{X_3}X_2 + X_4X_3\overline{X_2} + \\
&\qquad \overline{X_4}X_2X_1 + X_4\overline{X_3}\,\overline{X_2}\,\overline{X_1},\\
&S_i[3] = \overline{X_4}X_3\overline{X_2}\,\overline{X_1} + X_4X_3X_2\overline{X_1} + X_4\overline{X_3}\,\overline{X_2}X_1.
\end{aligned}
\tag{3}
$$

(2) 최댓값 30에 대한 리코딩

$$
\begin{aligned}
&C_{i+1}[0] = X_4\overline{X_3}\,\overline{X_2} + \overline{X_4}X_3X_1 + X_3X_2X_1 + \overline{X_4}X_3X_2,\\
&C_{i+1}[1] = X_4X_3 + X_4X_2,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + \overline{X_4}X_3X_2\overline{X_1} + \\
&\qquad X_4\overline{X_3}\,\overline{X_2}\,\overline{X_1} + X_4X_3\overline{X_2}X_1 + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_4}\,\overline{X_3}X_2 + X_4X_3\overline{X_2} + \overline{X_4}X_2X_1 + X_4\overline{X_2}\,\overline{X_1},\\
&S_i[3] = \overline{X_4}X_3\overline{X_2}\,\overline{X_1} + X_4X_3X_2\overline{X_1} + X_4\overline{X_3}\,\overline{X_2}X_1.
\end{aligned}
\tag{4}
$$

(3) 최댓값 27에 대한 리코딩

$$
\begin{aligned}
&C_{i+1}[0] = X_4\overline{X_3}\,\overline{X_2} + \overline{X_4}X_3X_1 + X_3X_2,\\
&C_{i+1}[1] = X_4X_2 + X_4X_3,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + X_3X_2\overline{X_1} + X_4\overline{X_3}\,\overline{X_2}\,\overline{X_1} + \\
&\qquad X_4X_3X_1 + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_4}\,\overline{X_3}X_2 + X_4X_3 + X_3X_2X_1 + X_4\overline{X_2}\,\overline{X_1},\\
&S_i[3] = \overline{X_4}X_3\overline{X_2}\,\overline{X_1} + X_4\overline{X_3}\,\overline{X_2}X_1.
\end{aligned}
\tag{5}
$$

(4) 최댓값 24에 대한 리코딩

$$\begin{aligned}
&C_{i+1}[0] = X_4\overline{X_3}\,\overline{X_2} + X_3X_1 + X_3X_2,\\
&C_{i+1}[1] = X_4X_2 + X_4X_3,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + X_3X_2\overline{X_1} + X_4\overline{X_3}\,\overline{X_2}\,\overline{X_1} + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_4}\,\overline{X_3}X_2 + X_3X_2X_1 + X_4\overline{X_2}\,\overline{X_1},\\
&S_i[3] = \overline{X_4}X_3\overline{X_2}\,\overline{X_1} + X_4\overline{X_2}X_1.
\end{aligned} \tag{6}$$

(5) 최댓값 22,23에 대한 리코딩

$$\begin{aligned}
&C_{i+1}[0] = X_4\overline{X_2} + X_3X_1 + X_3X_2,\\
&C_{i+1}[1] = X_4X_2,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + X_3X_2\overline{X_1} + X_4\overline{X_2}\,\overline{X_1} + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_4}\,\overline{X_3}X_2 + X_3X_2X_1 + X_4\overline{X_2}\,\overline{X_1},\\
&S_i[3] = X_3\overline{X_2}\,\overline{X_1} + X_4\overline{X_2}X_1.
\end{aligned} \tag{7}$$

(6) 최댓값 20,21에 대한 리코딩

$$\begin{aligned}
&C_{i+1}[0] = X_4\overline{X_2} + X_3X_1 + X_3X_2,\\
&C_{i+1}[1] = X_4X_2,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + X_3X_2\overline{X_1} + X_4\overline{X_2}\,\overline{X_1} + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_4}\,\overline{X_3}X_2 + X_2X_1 + X_4\overline{X_2}\,\overline{X_1},\\
&S_i[3] = X_3\overline{X_2}\,\overline{X_1} + X_4X_1.
\end{aligned} \tag{8}$$

(7) 최댓값 18에 대한 리코딩

$$\begin{aligned}
&C_{i+1} = X_4 + X_3X_1 + X_3X_2,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_2X_1 + X_3X_2\overline{X_1} + X_4\overline{X_1} + \overline{X_4}\,\overline{X_3}X_1,\\
&S_i[2] = \overline{X_3}X_2 + X_2X_1 + X_4\overline{X_1},\\
&S_i[3] = X_3\overline{X_2}\,\overline{X_1} + X_4X_1.
\end{aligned} \tag{9}$$

(8) 최댓값 12에 대한 리코딩

$$\begin{aligned}
&C_{i+1} = X_3X_1 + X_3X_2,\\
&S_i[0] = X_0,\\
&S_i[1] = \overline{X_3}X_1 + X_3X_2,\\
&S_i[2] = \overline{X_3}X_2,\\
&S_i[3] = X_3\overline{X_2}\,\overline{X_1}.
\end{aligned} \tag{10}$$

앞서 말한 수식을 통해 모든 리코더들은 조합 로직이 되고 상수 시간 안에 리코딩을 완성할 수 있기 때문에 부분곱에 대해 지연시간이 향상된다.

마지막 단계의 축약이 끝날 때 십진 CLA를 통해 더해져야 하는 합 $S_i \in [0,9]$와 캐리 $C_i \in [0,3]$에 관한 덧셈을 수행한다. 기존의 십진 CSA와 다르게 하나의 피연산자 비트의 수가 캐리 C_i의 범위 $[0,3]$에 따라 2비트로 변경된다. 또한 합 S_i의 범위를 고려하여 $S_i[3]$과 $S_i[2]$, $S_i[3]$과 $S_i[1]$에 대해서는 동시에 1이 되는 경우가 발생하지 않는다. 따라서 논리식 $\overline{S_i[3]}S_i[2]$, $= S_i[2]\overline{S_i[3]}S_i[1] = S_i[1]$, $S_i[3]S_i[2] = 0$이 이루어진다. 이는 앞서 말한 기존의 CLA의 수식을 개선할 수 있다. 개선된 수식은 다음과 같다.

$$\begin{aligned}
&K = X_3Y_1,\\
&L = X_3 + X_2G_1,\\
&C_{out} = K + Lc_1.
\end{aligned} \tag{11}$$

십진수 피연산자는 벡터 $X_3X_2X_1X_0$, 캐리 피연산자는 벡터 Y_1Y_0로 표시한다. 그리고 이진 캐리 전달 신호 P_i는 $P_i = X_i + Y_i$, 이진 캐리 생성 신호 G_i는 $G_i = X_iY_i$, XOR(배타적 논리합, exclusive OR) 신호 H_i는 $H_i = X_i \oplus Y_i$, 자리 1로 전달하는 캐리 c_1는 $c_1 = G_0 + P_0c_{in}$로 정의한다. 이에 따라 합 S_i는 다음 식과 같다.

$$\begin{aligned}
&S_0 = (X_0 \oplus Y_0) \oplus c_{in},\\
&S_1 = H_1\overline{c_1}\,\overline{K} + H_1c_1L + \overline{H_1}c_1\overline{L} + \overline{H_1}\,\overline{c_1}K,\\
&S_2 = \overline{X_2}G_1 + X_2\overline{P_1} + (\overline{X_2}X_1 + \overline{X_3}\,\overline{X_2}Y_1)c_1 + X_2H_1\overline{c_1},\\
&S_3 = X_2H_1c_1 + L\overline{K}\,\overline{c_1}.
\end{aligned} \tag{12}$$

식 (11)과 식 (12)를 이용하여 개선된 1-digit CLA를 얻는다. 경우에 따라 그룹 캐리 전달 신호 $P_{i:j}$와 생성 신호 $G_{i:j}$를 생성하고, 십진 다중 단계 CSA를 만든다. 분 논문에서 제안하는 30-digit CLA는 기존의 방법을 이용하는 CLA보다 성능을 향상시킨다. 전체적으로 최종 결과를 효율적으로 얻을 수 있다.

3. 검증을 위한 부분곱의 축약 단계

본 절에서는 3.2절에서 제안한 축약 방법의 일반화

가능성을 검증하기 위해 T. Lang의 논문에서 생성되는 십진 부분곱에 대해 제안한 방법을 이용하여 새로운 축약 단계를 만들었다. 기존의 최종 변환 단계를 포함하는 7단계 축약 단계에 비해 제안하는 축약 단계는 마지막 십진 CLA를 포함하는 5단계로 줄인다. 캐리를 더하는 개수를 세는 캐리 카운터도 사용하지만 자리에 의해 [2,8]개 자릿수의 CC를 사용한다.

앞서 언급한 범위를 제외하면 4개 자릿수의 최댓값을 표현하기에 필요한 비트의 수는 (4,2,4,3), 3개 자릿수에는 (4,3,4), (4,2,3), (4,1,3), (4,1,2), 2개 자릿수에는 (4,1)이 추가된다. 앞서 언급한 범위를 제외하면 4개 자릿수 CSA의 결과의 범위는 [0,28], [0,27], [0,26], [0,25], [0,20], 3개 자릿수에는 [0,26], [0,24], [0,19], [0,17], [0,16], [0,15], [0,13], 2개 자릿수에는 [0,11], [0,10]을 추가한다.

앞서 말한 개선된 1-digit 다중 피연산자 십진 CSA를 이용하여 마지막 단계인 32-digit CLA를 만들고 최종 십진 결과를 얻는다.

Ⅳ. 실험 결과 및 성능 분석

1. 시뮬레이션 및 검증

본 논문에서 제안한 부분곱 축약 단계는 Verilog HDL을 이용해 설계하였고 정확한 동작을 검증하기 위해 Mentor Graphics사의 ModelSim을 사용하여 RTL 시뮬레이션을 수행하였다. 우선 하위 모듈을 설계하고 시뮬레이션을 통해 검증한 후에 상위 모듈을 구성하는 방식으로 설계를 진행하였다.

2. 실험 결과

제안한 부분곱 축약 단계의 성능 평가를 위해서 기존의 일반 방법들을 이용한 부분곱 축약 단계와 제안한 부분곱 축약 단계를 동일한 ASIC 환경에서 합성하여 이를 비교한다. 설계된 모듈은 Synopsys사의 Design Complier를 사용하였고, 논리 합성을 위해 SMIC사의 180nm CMOS 공정 라이브러리를 사용했다.

표 1과 2는 기존의 일반적인 방법들을 이용한 부분곱 축약 단계와 제안한 부분곱 축약 단계를 동일한 환경에서 동일한 제약조건을 이용하여 합성한 결과이다.

M. Zhu의 부분곱에 대한 합성 결과에서 기존의 일반 십진 CLA 방법을 이용한 부분곱 축약 단계의 임계경로 지연시간은 2.37 ns, 면적은 154190.9 μm^2를 얻을 수 있었고 제안한 부분곱 축약 단계를 적용하면 임계경로 지연시간은 2.05 ns, 면적은 132747.9 μm^2을 얻을 수 있었다.T. Lang의 부분곱에 대한 합성 결과에서 일반 십진 CSA 방법을 이용한 축약 단계의 지연시간은 4.16 ns, 면적은 309861.1 μm^2을 얻을 수 있었고 제안한 부분곱 축약 단계의 임계경로 지연시간은 4.09 ns, 면적은 248974.7 μm^2을 얻을 수 있었다.

표 1. M. Zhu의 부분곱의 합성 결과
Table1. Synthesis result of M. Zhu's partial products.

Architecture	Delay		Area	
	(ns)	Ratio	(μm^2)	Ratio
M. Zhu et al.	2.37	1.000	154190.9	1.000
Proposed	2.05	0.865	132747.9	0.861

표 2. T. Lang의 부분곱의 합성 결과
Table2. Synthesis result of T. Lang's partial products.

Architecture	Delay		Area	
	(ns)	Ratio	(μm^2)	Ratio
T. Lang et al.	4.16	1.000	309861.1	1.000
Proposed	4.09	0.983	248974.7	0.804

그림 6는 M. Zhu의 부분곱에 대한 일반 축약 단계와 제안한 축약 단계의 각 단계별 지연사간을 분석한 결과이고, 그림 7는 각 단계별 면적을 분석한 결과이다. 제안한 부분곱 축약 단계에서는 1단계, 2단계, 3단계에서 사용된 가장 복잡한 다중 피연산자 십진 CSA가 더 큰 가중치를 가지는 위치에 존재하기 때문에 각각 경로의 지연시간을 균등하게 만들었다. 그리고 앞의 세 단계에서 일반 십진 CLA로 부분곱을 축약하는 방식과 비교했을 때 제안하는 다중 피연산자 십진 CSA는 부분곱을 빠르게 축약할 수 있는 것을 확인할 수 있다. 1단계에서의 지연시간은 0.62 ns에서 0.37 ns로 약 67.6% 감소하였고, 2단계에서의 지연시간은 0.63 ns에서 0.27 ns로 약 233.3% 감소하였고, 3단계에서의 지연시간은 0.44 ns에서 0.25 ns로 약 76.0% 감소하였다. 각 제안하는 CSA는 최소 비트 수로 자릿수의 최댓값을 표현함으로써 면적은 단계마다 약 22.3%, 15.7%, 21.1% 줄어듦을 확인할 수 있다. 최종 합을 얻을 때 개선된 십진 CLA를 사용하면 일반 CLA보다 면적이 감소하고 병렬 연산의 시간이 단축되지만 임계경로의 캐리 전달 시간과 면적이 6자리를 증가시키기 때문에 이전 세 단계의

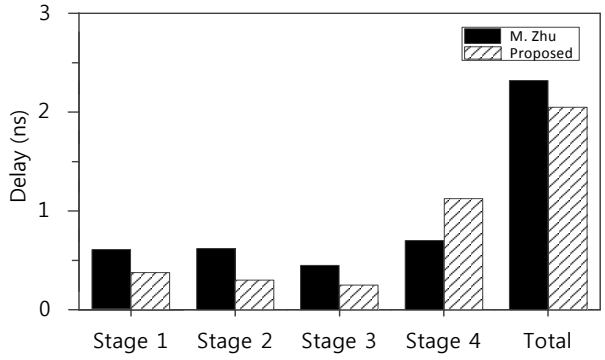


그림 6. 각 하위 모듈의 지연시간 비교
Fig. 6. Delay comparison of each sub-module.

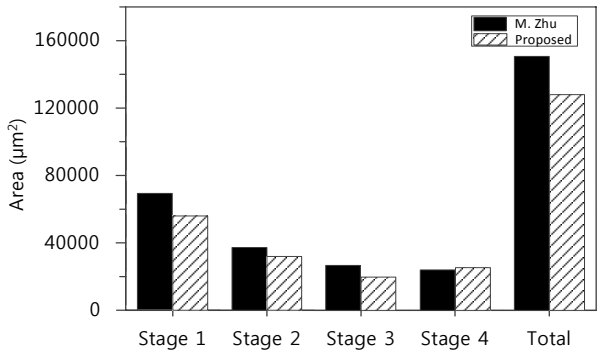


그림 7. 각 하위 모듈의 면적 비교
Fig. 7. Area comparison of each sub-module.

지연시간보다 마지막인 4단계에서 각 자리의 결과 값이 더 느리게 연산된다. 그 결과 지연시간은 일반 방법인 24-digit CLA에 비해 약 70.6% 증가하였고 면적은 약 3.1% 증가함을 확인할 수 있다. 그러나 전체적으로 보았을 때 총 지연시간은 약 15.6%, 총 면적은 약 16.2% 감소하는 결과를 얻을 수 있다.

다른 부분곱 축약 단계를 비교하기 위해 합성된 결과를 FO4 지연시간과 NAND2 등가게이트 면적으로 변환해야 한다. 하지만 부분곱 생성 단계에 따라 축약 단계의 아키텍처가 많이 다를 수 있기 때문에 축약 단계 사이의 비교는 의미가 없어 본 논문에선 생략한다.

V. 결 론

십진수를 근사치가 아닌 정확한 수치를 표현할 수 있는 십진 부동소수점 연산의 중요성에 따라 십진 연산을 더 효율적으로 수행하기 위한 많은 연구가 진행되고 있다.

본 논문에선 병렬 십진 곱셈기의 축약 단계의 면적과 지연시간을 감소시켜서 성능을 향상시키기 위해 다중 피연산자 십진 CSA과 개선된 십진 CLA를 이용한 부분곱 축약 단계를 제안한다. 제안한 부분곱 축약 단계는 십진수 부분곱에 대해 일반 십진 CSA를 사용하지 않고, 다중 피연산자 십진 CSA인 4to2, 3to2, 2to2 CSA를 사용하여 빠르게 부분곱을 축약한다. 각 CSA에서는 리코딩에 입력의 범위가 특정적이기 때문에 리코더의 가장 간단한 로직을 얻는다. 그리고 각 CSA는 지연시간을 균등하게 하는 아키텍처 트리의 특정한 위치에서 제한된 범위의 십진수를 더하기 때문에 부분곱 축약 단계의 연산을 효율적으로 수행할 수 있다. 또한, 최종 합을 얻기까지 사용되는 십진 CLA의 로직을 개선하여 BCD 결과를 빠르게 얻을 수 있다.

제안한 십진 부분곱 축약 단계의 성능의 평가를 위해 Synopsys사의 Design Compiler를 통해 SMIC사의 180nm CMOS 공정 라이브러리를 이용하여 합성하였다. M. Zhu의 부분곱에 대해 제안한 부분곱 축약 단계의 앞 세 단계에서 특정한 다중 피연산자 십진 CSA는 특정한 위치에서 사용되기 때문에 기존의 일반 CLA를 이용한 방법에 비해 지연시간과 면적이 감소하였다. 마지막 십진 CLA의 두 개의 피연산자 중 하나의 피연산자에 대해서 비트 수를 줄일 수 있지만 일반 24-digit CLA보다 32-digit CLA의 연산에 관한 자릿수가 많아지므로 면적과 지연시간의 증가함을 확인할 수 있다. 32-digit CLA를 이용하여 최종 결과를 얻을 때 십진 CLA의 지연시간과 면적 증가가 있음에도 불구하고 전체 지연시간은 약 15.6%, 전체 면적은 약 16.2% 감소하였다.

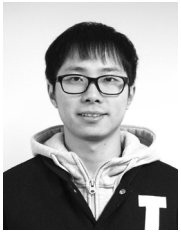
제안한 방법을 다른 곱셈기에서도 사용할 수 있는 것을 검증하기 위해서 T. Lang의 부분곱에 대해 제안한 축약 단계에 적용하여 일반 CSA를 이용한 방법과 비교하였더니 지연시간이 1.7%, 면적이 24.5% 줄어듦을 확인할 수 있었다.

REFERENCES

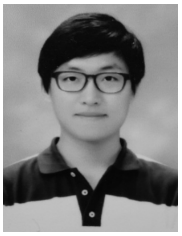
[1] E. A. a. P. A. Vazquez, “A New Family of High-Performance Parallel Decimal Multipliers,” 18th IEEE Symposium on Computer Arithmetic. (ARITH'18), June. 2007.
[2] B. J. H. a. M. J. S. Mark A. Erle, “Decimal Floating-Point Multiplication,” IEEE Transaction on Computers, vol. 58, no. 7, July. 2009.
[3] H. A. H. Fahmy. M. Y. Hassan, Y. Farouk, and R. R. Eissa, “Decimal Floating Point for future processors,” IEEE International Conference Microelectronics (ICM '09), pp. 443-446, Dec. 2010.
[4] IEEE, IEEE 754 Standard for Binary Floating-Point

Arithmetic, 1985.
[5] IEEE, IEEE 754-2008 Standard for Floating-Point Arithmetic, 2008.
[6] A. Kaivani, Liu Han, and S. Ko, “Improved Design of High-Frequency Sequential Decimal Multipliers,” Electronics Letters, pp. 558-560, Mar. 2014.
[7] A. Vazquez, E. Antelo, and P. Montuschi, “A New Family of High-Performance Parallel Decimal Multiplier,” Proc. 18th IEEE Symp. Computer Arithmetic, pp. 195-204, June. 2007.
[8] A. Vazquez, E. Antelo, and J. D. Bruguera, “Fast Radix-10 Multiplication Using Redundant BCD Codes,” IEEE Transaction on Computers, vol. 63, no. 8, August. 2014.
[9] I. K. Hwang, K. H. Kim, W. O. Yoon, and S. B. Choi, “Design of Parallel Decimal Multiplier using Limited Range of Signed-Digit Number Encoding,” Journal of the Institute of Electronics Engineers of Korea, vol. 50, no. 3, pp. 50-58, Mar. 2013.
[10] M. Zhu and Y. Jiang, “An Area-Time Efficient Architecture for 16×16 Decimal Multiplication,” Information Technology: New Generations, ITNG 2013, April. 2013.
[11] T. Lang and A. Nannarelli, “A Radix-10 Combination Multiplier,” 40th Asilomar Conference on Signals, System and Computers, pp. 313-317, Oct. 2006.
[12] A. Weinberger. and M. S. Schmookler, “High Speed Decimal Addition,” IEEE Transaction on Computers, vol. c-20, no. 8, August. 1971.

저 자 소 개



이 양(학생회원)
2011년 Northeastern Univ. 학사 졸업.
2016년 인하대학교 전자공학과 석사 졸업.
<주관심분야 : 컴퓨터 구조, SoC, 연산기>



박 태 신(학생회원)
2015년 인하대학교 전자공학과 학사 졸업.
2016년～현재 인하대학교 전자공학과 석사과정.
<주관심분야 : 컴퓨터 네트워크, 무선 센서 네트워크, 임베디드 시스템>

김 강 희(학생회원)
2011년 인하대학교 전자공학과 학사 졸업.
2013년 인하대학교 전자공학과 석사 졸업.
2013년～현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 컴퓨터 네트워크, 무선 센서 네트워크, SoC>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년～1986년 LG 정보통신(주).
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년～현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>