논문 2012-50-3-7

제한된 범위의 Signed-Digit Number 인코딩을 이용한 병렬 십진 곱셈기 설계

(Design of Parallel Decimal Multiplier using Limited Range of Signed-Digit Number Encoding)

황 인 국*, 김 강 희*, 윤 완 오*, 최 상 방**

(In-Guk Hwang, Kanghee Kim, WanOh Yoon, and SangBang Choi)

요 약

본 논문에서는 제한된 범위의 Signed-Digit number 인코딩과 축약 단계를 이용한 고정소수점 병렬 십진 곱셈기를 제안한다. 제안한 병렬 십진 곱셈기는 승수와 피승수를 제한된 범위의 SD number로 인코딩하여 캐리 전달 지연 없이 빠르게 부분곱을 생성한다. 인코딩에 사용하는 숫자의 범위를 줄임으로써 SD number 다중 피연산자 덧셈의 한번에 연산 가능한 피연산자의 개수가 늘어나게 되고, 이에 따라 부분곱 축약 단계의 연산을 빠르게 수행 할 수 있다. 제안한 병렬 십진 곱셈기의 성능 평가를 위해 Design Compiler에서 SMIC사의 180nm CMOS 공정 라이브러리를 이용하여 합성한 결과 기존의 Signed-Digit number를 이용한 병렬 십진 곱셈기보다 전체 지연시간은 4.3%, 전체 면적은 5.3% 감소함을 확인 하였다. 전체 지연시간 및 면적에서 부분곱 축약 단계가 차지하는 비중이 가장 크므로 부분곱 생성 단계에서 약간의 지연시간 및 면적 증가가 있음에도 불구하고 전체 지연시간과 면적이 감소하는 결과를 얻을 수 있다.

Abstract

In this paper, parallel decimal fixed-point multiplier which uses the limited range of Singed-Digit number encoding and the reduction step is proposed. The partial products are generated without carry propagation delay by encoding a multiplicand and a multiplier to the limited range of SD number. With the limited range of SD number, the proposed multiplier can improve the partial product reduction step by increasing the number of possible operands for multi-operand SD addition. In order to estimate the proposed parallel decimal multiplier, synthesis is implemented using Design Compiler with SMIC 180nm CMOS technology library. Synthesis results show that the delay of proposed parallel decimal multiplier is reduced by 4.3% and the area by 5.3%, compared to the existing SD parallel decimal multiplier. Despite of the slightly increased delay and area of partial product generation step, the total delay and area are reduced since the partial product reduction step takes the most proportion.

Keywords : Parallel Decimal Multiplier, IEEE 754-2008, Signed-Digit Number

Ⅰ. 서 론

십진 부동소수점(Decimal Floating-Point, DFP) 연산은 금융, 과학 어플리케이션과 같이 정밀도가 매우 중요한 분야에서 주목받고 있다. 이는 이진 부동소수점(Binary Floating-Point, BFP) 연산이 특정 부동소수점 수를 정확한 값으로 표현할 수 없기 때문이다. 십진 부동소수점 연산을 위해 IBM의 decNumber library[1], Java의 BigDecimal library[2], Intel의 DFP Math library[3]와 같은 소프트웨어 라이브러리 형태로 지원되고 있지만, 십진 부동소수점 연산에 대한 보다 높은 성능을 얻기 위해서는 하드웨어적인 측면에서 십진 부동소수점 연산기를 설계하여 이를 프로세서에 내장하는

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
※ 이 논문은 정부(교육과학기술부)의 재원으로 한국연구재단의 중점연구소 지원사업으로 수행된 연구임 (2012-0005858)
접수일자: 2012년10월31일, 수정완료일:2013년2월27일

것이 더욱 바람직하다[4]. IBM의 POWER6[5], z9[6] 이후의 아키텍처(architecture)에서는 DFU (Decimal Floating-point Unit)를 포함 하고 있고, SilMinds사에서는 십진 부동소수점 연산 IP (Intellectual Property)[7]를 개발하여 판매하고 있다.

십진 부동소수점 연산에 대한 중요성이 부각됨에 따라, 미국 전기 전자 학회(Institute of Electrical and Electronics Engineers, IEEE)에서는 기존의 이진 부동소수점 연산만을 다뤘던 IEEE 754 표준안[8]을 개정하여 십진 부동소수점 연산을 추가 도입한 IEEE 754-2008 표준안[9]을 제정하였다. IEEE 754-2008 표준안에서는 십진수를 저장하기 위한 포맷으로 DPD (Densely Packed Decimal) 코드와 BID (Binary Integer Decimal) 코드를 제안하였는데, 연산을 위해서는 BCD (Binary Coded Decimal)와 같이 십진수를 이진수로 표현하기 위한 방법이 필요하다. BCD는 4비트의 이진수 표기와 같은 방법으로 쉽게 십진수를 표현 가능하지만, 0xA~0xF의 숫자를 사용하지 않음으로 인한 redundancy를 갖게 된다. 이러한 redundancy를 피할 수 있는 4비트 Signed-Digit (SD) number를 이용하여 표현가능한 모든 숫자의 범위를 내부 연산에 사용하여 연산의 효율성을 높일 수 있다.

십진 부동소수점 곱셈기는 여러 가지 세부 유닛들로 이루어지며, 이중에서 연산의 핵심이 되는 부분은 십진 고정 소수점 곱셈기이다. 본 논문에서는 십진 고정 소수점 곱셈기의 성능을 향상시키기 위해 개선된 SD number 인코딩(encoding)과 축약(reduction) 단계를 이용한 병렬 십진 곱셈기를 제안한다.

제안하는 병렬 십진 곱셈기는 승수(multiplier)와 피승수(multiplicand)를 제한된 숫자 범위의 SD number로 인코딩하여 캐리 전달 지연(carry-propagation delay)없이 빠르게 부분곱(partial product)을 생성한다. 인코딩에 사용하는 숫자의 범위를 줄임으로써 SD number 다중 피연산자 덧셈(multi-operand addition)의 한번에 연산 가능한 피연산자의 개수를 늘려 부분곱 축약 단계를 개선하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 병렬 십진 곱셈기의 연산과정을 살펴보고 본 논문의 기초가 되는 Liu Han의 SD number를 이용한 병렬 십진 곱셈기[10]에 대해 설명 한다. Ⅲ장에서 제안한 병렬 십진 곱셈기의 부분곱 생성, 부분곱 축약, 최종합 단계에 대해 설명하고, Ⅳ장에서는 제안한 병렬 십진 곱셈기를 구현하여 ASIC 환경에서 지연시간 및 면적에 대해 비교 분석한다. 마지막으로 Ⅴ장에서는 본 논문에서의 연구 내용을 정리한 후 최종 결론을 제시한다.

Ⅱ. 관련 연구

이 장에서는 병렬 십진 곱셈에 대한 기본 연산 과정 및 SD number를 이용한 병렬 십진 곱셈기에 대해 살펴본다.

2.1 병렬 십진 곱셈기 개요

본 논문에서는 IEEE 754-2008 표준안에서 정의된 decimal64 포맷을 만족하는 정밀도 16의 유효수 부분을 연산하기 위한 고정소수점 곱셈기에 대해 다루도록 한다. 고정소수점 병렬 십진 곱셈기는 그림 1과 같이 크게 3단계의 연산과정을 갖는다.

그림 1의 (a)단계는 부분곱 생성(partial product generation) 단계이다. 부분곱 생성 단계는 전체 피승수에 승수의 각 자릿수를 곱해 부분곱을 만들어 내는 단계이다. 십진 곱셈의 경우 승수의 각 자릿수는 0~9 이므로, 피승수에 대해 $0X$~$9X$의 배수(multiple)를 병렬로 모두 생성한 후 승수의 각 자릿수에 맞게 부분곱을 선택하는 방법을 주로 사용한다. $0X$와 $1X$는 와이어(wire) 연결만으로도 피승수의 배수를 얻을 수 있다. 그리고 $2X$와 $5X$에 해당하는 배수는 간단한 로직 게이트를 통해 얻을 수 있는 방법이 연구되어 있으며[11], $4X = 2 \times 2X$, $8X = 2 \times 2 \times 2X$ 도 같은 방법으로 고정 시간내에 얻을 수 있다. 하지만 나머지 $\{3X, 6X, 7X, 9X\}$ 배수에 대해서는 $3X = 1X + 2X$

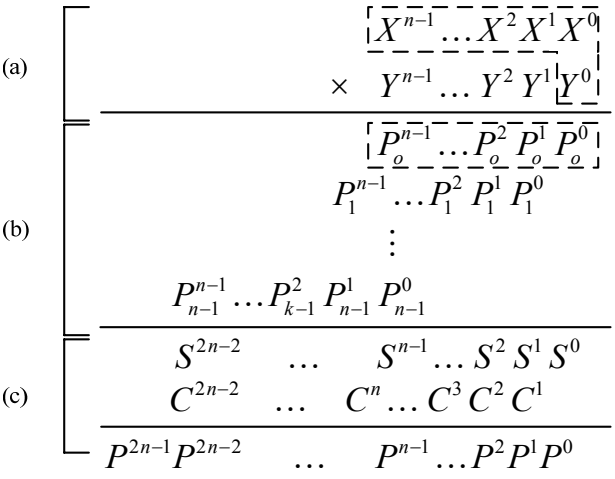

그림 1. 병렬 십진 곱셈의 3가지 단계
Fig. 1. The three step of parallel decimal multiplication.

와 같이 캐리 전달 지연이 필요한 덧셈을 통해서 얻을 수 있다. 이때 선택되어진 부분곱은 곱해진 승수의 자릿수를 고려하여 다음 단계에 정렬되어야 한다.

그림 1의 (b)단계는 부분곱 축약(partial product reduction) 단계이다. 부분곱 생성 단계에서 만들어진 승수의 각 자릿수에 해당하는 부분곱은 승수의 자릿수를 고려하여 더해져야 한다. 승수의 자릿수 개수 만큼의 다중 피연산자 덧셈 연산을 수행해야 하기 때문에 연산 시간이 가장 오래 걸리는 단계이다. 보통의 경우 캐리 전달 지연을 피하기 위해 이진 CSA (Carry Save Adder) 트리를 십진수에 맞게 변형하여 연산을 수행하거나 십진 가산기를 이용한 CS (Carry Save) 구조를 이용한다. 부분곱 축약 단계를 통해 합 벡터(sum vector)와 캐리 벡터(carry vector)를 얻을 수 있고 그림 1의 (c)에 해당하는 최종합 단계에서 이 두 벡터를 더해 최종 결과값을 얻는다. 최종합 단계에서는 십진 덧셈을 위한 BCD 가산기를 사용하거나, 이진 가산기에 사전교정(pre-correction)과 후교정(post-correction)을 수행하여 최종곱(final product)을 얻게 된다.

2.2 SD number를 이용한 병렬 십진 곱셈기

이 절에서는 본 논문의 기초가 되는 Liu Han의 SD number를 이용한 병렬 십진 곱셈기에 대해 살펴본다.

2.2.1 부분곱 생성 단계

SD number를 이용한 병렬 십진 곱셈기는 승수와 피승수를 $[-8,8]$의 범위를 갖는 SD number로 인코딩한다. 피승수의 $1X \sim 4X$ 배수는 이전 자릿수의 캐리를 고려하여 인코딩되지만, $5X$ 배수는 두 자릿수 전의 캐리까지 고려하여 4비트 SD number로 인코딩된다. 따라서 최대 $(n+2)$자릿수의 배수가 생성된다. SD number로 인코딩된 $1X \sim 5X$의 각 배수는 몇 단계의 논리게이트를 통해 고정 시간 내에 얻을 수 있다. 이렇게 생성된 $1X \sim 5X$의 배수는 인코딩된 승수의 선택 신호를 통해 부분곱 선택 로직에서 $(n+1)$개의 부분곱으로 선택 된다. 음수배수가 선택될 경우 9의 보수를 취하게 되는데, 10의 보수를 얻기 위해서는 마지막 자릿수에 1을 더해야 한다. 더해야 하는 1의 개수는 승수의 인코딩된 부호가 음수인 자릿수의 개수와 같으므로 이를 부분곱 축약 단계에서 카운터(counter)를 통해 SD number로 변환하여 더한다. 이 같은 방법을 통해 피승수에 승수의 각 자릿수를 곱한 부분곱은 $\{\pm 1X, \pm 2X, \pm 3X, \pm 4X, \pm 5X\}$ 중 하나로 선택된다 .

2.2.2 부분곱 축약 단계

부분곱 생성 단계에서 선택되어진 $(n+1)$개의 부분곱은 곱해진 승수의 자릿수를 고려하여 부분곱 축약 단계에서 더해져야 한다. 이때 배치된 $(2n+1)$개의 열 중에 $(n+1)$, $(n+2)$번째 열에서 최대 17개의 부분곱 자릿수를 더해야 하고, 해당 열은 임계경로(critical path)가 된다.

십진수의 모든 자리를 표현하려면 최소 $[-5,4]$의 범위가 필요하며, 4비트 SD number의 표현 범위인 $[-8,7]$내에서 연산하기 위해 최대 캐리는 $[-3,3]$범위 내에서 발생해야 하고, 한번에 최대 $[-35,34]$의 범위까지 연산이 가능하다. $[-8,8]\times 4 = [-32,32]$이므로 부분곱 생성 단계에서 $[-8,8]$의 범위로 인코딩된 각 부분곱은 한번에 최대 4개의 부분곱을 축약 가능하다. 이에 해당하는 로직이 부분곱 축약의 1단계 연산이다.

4개의 부분곱을 더하는 로직을 2단계로 구성할 경우 16개의 부분곱 연산이 가능한데, 임계경로에서 최대 17개의 부분곱을 더해야 한다. 이 때문에 1단계를 추가하여 3단계로 부분곱 축약 트리를 구성할 경우 긴 지연시간을 초래하므로 $[-35,34]$ 범위까지 연산 가능함을 이용해 17번째 부분곱을 4부분으로 작게 나누어 1단계에 포함하여 연산을 수행한다. 각 단계의 연산이 수행된 후 각 자릿수는 $[-5,4]$의 범위를 갖는 합 벡터와 $[-3,3]$의 범위를 갖는 캐리 벡터로 리코딩(recoding)되어 다음 단계에 전달된다.

부분곱 축약의 2단계 연산은 1단계 연산 후 리코딩된 4개의 합 벡터와 캐리 벡터를 축약하는 단계로, $[-8,7]\times 4 = [-32,28]$이므로 역시 축약 가능하다. 최종적으로 축약되어 리코딩된 합 벡터와 캐리 벡터는 최종합 단계로 전달된다.

2.2.3 최종합 단계

부분곱 축약 단계에서 더해진 결과인 합 벡터와 캐리 벡터는 최종합 단계에서 더해져 BCD 포맷으로 변환되어야 한다.

부분곱 축약 단계에서 각 자리의 합 벡터와 캐리 벡터의 합은 $[-8,7]$ 범위이므로, 합이 음수이면 빌림(borrow)이 발생하게 된다. 빌림을 감지하기 위한 빌림 생성(generate)신호와 빌림 전달(propagate)신호를 캐리

프리픽스 네트워크(prefix-network)를 이용해 빠르게 전달하고, 결정된 각 자릿수의 빌림 신호를 고려한 조건부 상수 가산기(conditional constant adder)에 의해 BCD로 변환된다.

Ⅲ. 제한된 범위의 SD 인코딩을 이용한 병렬 십진 곱셈기

이 장에서는 제안하는 병렬 십진 곱셈기에 대해 설명한다. 2.2절에서 설명한 SD number를 이용한 병렬 십진 곱셈기를 바탕으로 성능을 개선하기위한 방법을 제안한다.

3.1. 개요

제안하는 구조는 내부적으로 4비트 SD number를 이용하여 효율적으로 부분곱 인코딩 및 축약을 수행하는 병렬 십진 곱셈기이다. 부분곱 생성 단계에서 부분곱은 $[-6,6]$의 범위로 인코딩된 피승수의 $1X \sim 5X$ 배수들과 $[-5,5]$로 인코딩된 승수를 통해 $(-5X) \sim 5X$로 선택된다. 이때 $1X \sim 5X$의 배수는 논리 게이 트를 통해 고정된 시간 내에 얻을 수 있고, $(-1X) \sim (-5X)$를 얻기 위해 부분곱 선택(partial product selection) 로직을 통해 Yn_i가 1일 경우 비트 반전으로 9의 보수를 얻도록 한다. 10의 보수를 구하기 위해 마지막 자릿수에 1을 더하는 동작은 기존의 방법과 마찬가지로 부분곱 축약 단계에서 카운터를 이용해 더하도록 한다.

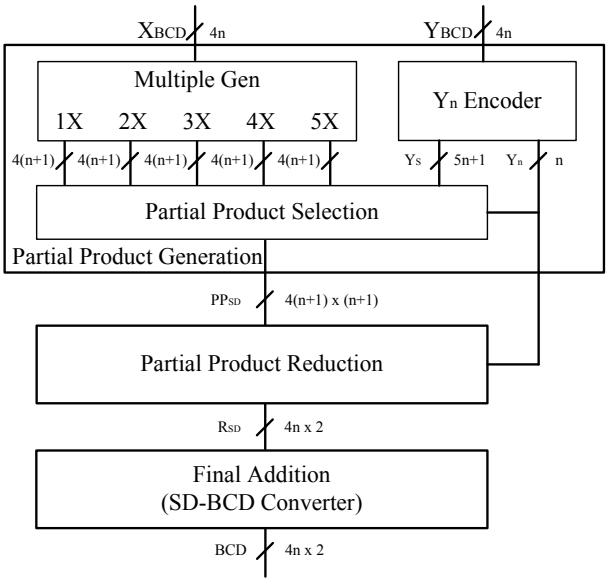

그림 2. 제안하는 병렬 십진 곱셈기의 전체 구조
Fig. 2. Top level architecture of proposed parallel decimal multiplier.

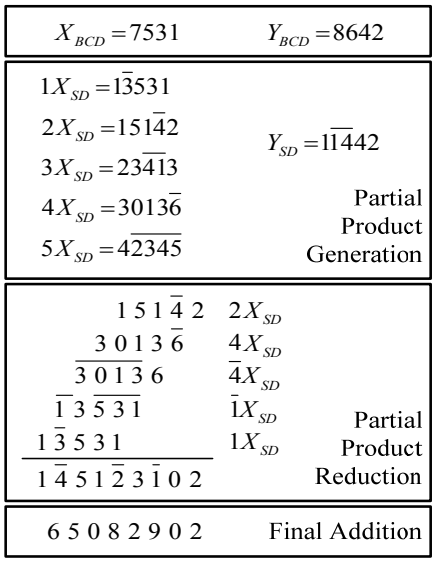

그림 3. 4x4 곱셈의 예
Fig. 3. Example of 4x4 multiplication.

부분곱 축약 단계에서는 $(n+1)$자릿수를 가지는 $(n+1)$개의 부분곱 축약을 수행한다. 제안하는 1단계 축약에서 한 번에 5개의 SD number를 더한 후 리코더(recoder)를 통해 각각의 비트 가중치를 고려하여 합 벡터 $w_i \in [-5,4]$와 캐리 벡터 $t_i \in [-3,3]$로 리코딩을 수행한다. 1단계 축약에서 최대 20개의 부분곱을 더 할 수 있으므로, IEEE 754-2008 표준안의 decimal64 포맷의 유효수 정밀도를 만족하는 경우 기존의 SD number를 이용한 방법에 비해 특별한 처리가 필요 하지 않다. 2단계 축약에서는 기존의 방법과 마찬가지로 1단계 축약에서 리코딩된 4개의 합 벡터와 캐리 벡터를 더한 후 두번째 리코더를 통해 하나의 합 벡터와 캐리 벡터를 구한다.

부분곱 축약 단계에서 하나로 축약된 합 벡터와 캐리 벡터를 최종합 단계에서 캐리 프리픽스 네트워크를 통해 BCD로 변환하여 최종 결과값을 얻는다. 제안하는 병렬 십진 곱셈기의 전체 구조는 그림 2와 같고, 연산에 대한 예는 그림 3과 같다.

3.2. 부분곱 생성 단계

제안하는 병렬 십진 곱셈기의 부분곱 생성은 표 1과 같이 승수와 피승수 모두 SD number를 이용하여 $[-6,6]$의 범위로 인코딩을 수행한다. 기존의 SD number를 이용한 인코딩 방법에 비해 $5X$를 $T_i + W_i$만으로 표현하여 부분곱의 자릿수를 $(n+1)$로 한자릿

표 1. 제안하는 승수와 피승수 인코딩
Table 1. Proposed encoding of Multiplier and Multiplicand.

BCD Operand	$1X_i$		$2X_i$		$3X_i$		$4X_i$		$5X_i$		Y_i	
	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	2	0	3	1	-6	1	-5	0	1
2	0	2	1	-6	1	-4	1	-2	1	0	0	2
3	0	3	1	-4	1	-1	1	2	2	-5	0	3
4	0	4	1	-2	1	2	2	-4	2	0	0	4
5	0	5	1	0	2	-5	2	0	3	-5	1	-5
6	1	-4	1	2	2	-2	3	-6	3	0	1	-4
7	1	-3	1	4	2	1	3	-2	4	-5	1	-3
8	1	-2	2	-4	3	-6	3	2	4	0	1	-2
9	1	-1	2	-2	3	-3	4	-4	5	-5	1	-1
$T_i + W_i$	$[-4,6]$		$[-6,6]$		$[-6,6]$		$[-6,6]$		$[-5,5]$		$[-5,5]$	

수 줄일 수 있고, 표현 범위를 $[-8,8]$에서 $[-6,6]$으로 제한함으로써 부분곱 축약 단계에서 한번에 더할 수 있는 부분곱의 개수를 5개로 늘릴 수 있다.

피승수의 $1X, 2X, 4X$ 배수 인코딩과 승수의 인코딩은 기존의 방법을 동일하게 사용하고, 제안하는 피승수의 $3X$와 $5X$ 배수 인코딩은 다음과 같다.

• $3X$ 인코딩

기존의 SD number를 이용한 방법과 같이 피승수의 $3X$ 배수를 SD number를 이용해 직접 인코딩하지만 사용하는 숫자의 범위는 $[-6,6]$이다. 제안하는 $3X$ 배수의 인코딩은 식 (1)과 같다.

$$
\begin{aligned}
{}_3T_i^1 &= X_{i-1}^2(X_{i-1}^0 + X_{i-1}^1) + X_{i-1}^3 \\
{}_3T_i^0 &= (X_{i-1}^3 + X_{i-1}^2 + X_{i-1}^1)(\overline{X_{i-1}^2} + \overline{X_{i-1}^1})(\overline{X_{i-1}^2} + \overline{X_{i-1}^0}) \\
{}_3W_i^3 &= \overline{X_i^2}X_i^1(\overline{X_i^0} + \overline{T_i^1}\,\overline{T_i^0}) + (X_i^1\overline{X_i^0}\,\overline{T_i^1}) \\
&\quad + (X_i^2\overline{X_i^1}X_i^0) + X_i^3(\overline{X_i^0} + \overline{T_i^1} + \overline{T_i^0}) \\
{}_3W_i^2 &= \overline{X_i^2}X_i^1(\overline{X_i^0} + \overline{T_i^1}\,\overline{T_i^0}) + \overline{X_i^0}(X_i^1\overline{T_i^1} + X_i^3X_i^1) \\
&\quad + X_i^2T_i^1(\overline{X_i^1} + X_i^0T_i^0) \\
&\quad + X_i^0(\overline{X_i^3}\,\overline{X_i^1}T_i^0 + \overline{X_i^1}T_i^1\overline{T_i^0} + X_i^3\overline{T_i^1}) \\
{}_3W_i^1 &= \overline{X_i^3}T_i^1(\overline{X_i^2}\,\overline{X_i^0} + \overline{X_i^1}X_i^0T_i^0) + X_i^2\overline{T_i^1}(\overline{X_i^0} + X_i^1T_i^0) \\
&\quad + X_i^0\overline{T_i^0}(X_i^2X_i^1T_i^1 + X_i^3T_i^1 + \overline{X_i^3}\,\overline{X_i^2}\,\overline{T_i^1}) \\
&\quad + X_i^3\overline{T_i^1}(\overline{X_i^0} + T_i^0) \\
{}_3W_i^0 &= (\overline{X_i^0}T_i^0) + (X_i^0\overline{T_i^0})
\end{aligned}
\tag{1}
$$

• $5X$ 인코딩

피승수의 $5X$ 배수 역시 SD number로 직접 인코딩을 수행한다. 기존의 방법과의 차이점은 두 자릿수 이전 캐리를 제거하여 $T_i + W_i$만으로 인코딩 한다는 것이다. 이로 인해 기존의 방법에 비해 부분곱의 자릿수가 한자리 줄어들게 된다. 제안하는 $5X$ 배수의 인코딩은 식 (2)와 같다.

$$
\begin{aligned}
{}_5T_i^2 &= (X_{i-1}^2X_{i-1}^1X_{i-1}^0) + X_{i-1}^3 \\
{}_5T_i^1 &= (\overline{X_{i-1}^2}X_{i-1}^1X_{i-1}^0) + X_{i-1}^2(\overline{X_{i-1}^1} + \overline{X_{i-1}^0}) \\
{}_5T_i^0 &= (\overline{X_{i-1}^1}X_{i-1}^0) + (X_{i-1}^1\overline{X_{i-1}^0}) \\
{}_5W_i^3 &= X_i^0(\overline{T_i^2} + \overline{T_i^0}) \\
{}_5W_i^2 &= T_i^2(\overline{X_i^0} + \overline{T_i^0}) + X_i^0(\overline{T_i^2}T_i^0 + T_i^1) \\
{}_5W_i^1 &= T_i^1(\overline{X_i^0} + T_i^0) + (X_i^0\overline{T_i^1}\,\overline{T_i^0}) \\
{}_5W_i^0 &= (\overline{X_i^0}T_i^0) + (X_i^0\overline{T_i^0})
\end{aligned}
\tag{2}
$$

3.3. 부분곱 축약 단계

부분곱 생성 단계에서 만들어진 $(n+1)$자릿수의 부분곱 $(n+1)$개를 축약하기 위해 승수의 자릿수를 고려하여 부분곱을 배치한다. 이때 기존의 SD number를 이용한 방법과 같이 특별한 배치를 하지 않고, 자릿수만을 고려하여 그림 1의 (b)와 같이 평행사변형 모양으로 배치한다. 부분곱 생성 단계에서 인코딩을 변경하여 각 부분곱의 자릿수가 $(n+1)$로 한자릿수 줄어들었지만, 최대 $(n+1)$개의 부분곱을 더해야 하는 $(n+1)$번째 열이 임계 경로임은 동일하다. 하지만 기존의 SD number를 이용한 방법과 달리 인코딩에 사용되는 숫자의 범위를 줄임으로써 한번에 5개의 SD number를 더할 수 있어 기존의 방법에 비해 17번째 부분곱에 대한 특별한 처리가 필요없다.

표 2. $[-6,6]$ 범위의 SD number에서 덧셈 가능한 피연산자의 개수 분석
Table 2. Analysis of the number of operands of SD addition with $[-6,6]$.

Number of operands	Range of ps_i	Range of t_i	Range of w_i	Range of s_i
2	[-12, 12]	[-1, 1]	[-5, 4]	[-6, 5]
3	[-18, 18]	[-2, 2]	[-5, 4]	[-7, 6]
4	[-24, 24]	[-2, 2]	[-5, 4]	[-7, 6]
5	[-30, 30]	[-3, 3]	[-5, 4]	[-8, 7]
6	[-36, 36]	[-4, 4]	[-5, 4]	[-9, 8]

제안하는 부분곱 인코딩을 이용해 다중 피연산자 덧셈에서 4비트 SD number로 연산 가능한 최대 피연산자의 개수는 표 2와 같이 5개이다. 인코딩하는 SD number의 범위를 $[-6,6]$으로 제한하여, $[-8,7]$의 표현범위 내에서 한번에 더할 수 있는 부분곱의 개수를 5개로 늘릴 수 있다.

이에 따라 제안하는 부분곱 축약 단계의 1단계인 5개의 SD number 덧셈에 해당하는 로직은 그림 4와 같다. 그림 4의 P_I가 가리키는 화살표는 10의 보수시 마지막 자릿수에 1을 더하는 개수를 세는 4:3 카운터, 점선 사각형들은 반가산기(Half Adder, HA), 전가산기(Full Adder, FA), 4:2 컴프레서(compressor), 두줄 선은 CLA(Carry Look-ahead Adder) 로직, 굵은 선은 리코더를 의미하고, 표현 방법은 [12]에 제안된 WBS (Weighted Bit-Set) 방법을 응용하여 사용하였다. 1단계 축약에서 5개의 부분곱을 더하는 경우는 17개의 부분곱을 모두 더해야 하는 임계경로의 경우에만 해당된다. 이때 1단계 축약에서 $\{5,4,4,4\}$개의 부분곱을 더해야하며, 5개의 부분곱을 더하는 경우에 17번째 부분곱이 포함되어야 한다. 17번째 부분곱은 $0X$ 또는 $1X$이므로 음수 부분곱은 선택되지 않는다. 따라서 임계경로에서 음수 부분곱 선택시 1을 더하는 개수는 4개로 동일하다. 나머지 자릿수에 대해서는 5개의 부분곱을 더하는 경우가 발생하지 않는다.

각 단계의 축약이 끝날 때마다 리코더를 통해 연산 가능 범위인 $s_i \in [-8,7] = t_i + w_i$, $t_i \in [-3,3]$, $w_i \in [-5,4]$로 각각의 비트 가중치를 고려하여 리코딩을 수행한다. 기존의 SD number를 이용한 방법과 다르게 1단계 연산 결과의 범위가 $(-5X) \sim 5X$의 부분곱 4개와 $0X$ 또는 $1X$ 부분곱 1개 즉, $[-6,6] \times 4 + [-4,6] = [-28,30]$으로 변함에 따라 해당 리코더 로직도 변경되어야 한다.

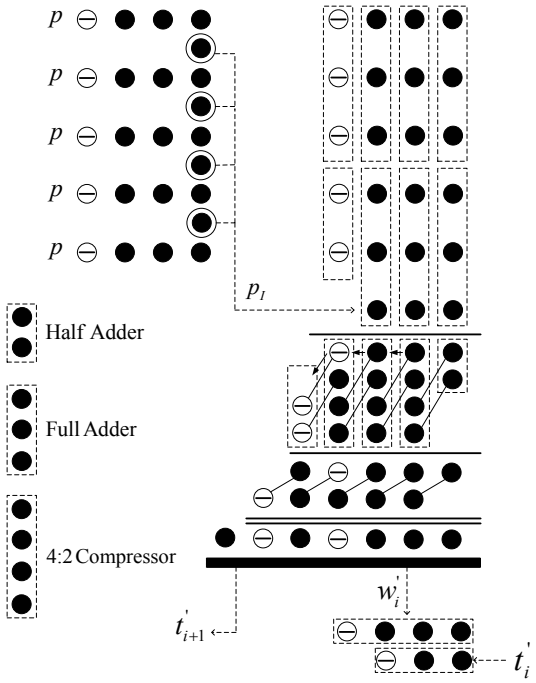

그림 4. 제안하는 1단계 다중 SD number 덧셈
Fig. 4. 1st level of proposed multi-operand SD number additions.

1단계 연산후 리코딩된 4개의 $t_i' + w_i'$를 축약하는 단계인 부분곱 축약 단계의 2단계는 기존의 SD number를 이용한 방법과 동일하다. 최종적으로 축약되어 리코딩된 합벡터 w_i와 캐리벡터 t_i는 최종합 단계에 전달된다.

임계경로의 1단계 축약단계의 지연시간을 비교해 보면 임계 경로에서 기존의 SD number를 이용한 1단계 축약 단계는 3개의 FA, 1개의 3비트 CLA, 1개의 $[-34,34]$리코더를 거치므로 5.75 FA 지연시간을 갖는다. 반면, 제안하는 병렬 십진 곱셈기의 부분곱 축약 단계의 임계경로는 1개의 FA, 1개의 4:2 컴프레서, 1개의 4비트 CLA, 1개의$[-28,30]$리코더를 거치므로 5.35 FA 지연시간을 갖고 2단계 축약 단계는 기존의 SD number를 이용한 방법과 동일하다. 제안하는 병렬 십진 곱셈기가 기존의 SD number를 이용한 병렬 십진 곱셈기보다 임계경로에서 약 0.4 FA 지연시간만큼 향상됨을 확인할 수 있다. 이때, 3비트 CLA의 지연시간은 1개의 FA, $[-34,34]$리코더는 1.75개의 FA, 4:2 컴프레서는 1.5 FA, 4비트 CLA는 1.25 FA, $[-28,30]$리코더는 1.6 FA 지연시간과 같다.

3.4. 최종합 단계

최종합 단계의 합벡터 w'_i과 캐리벡터 t'_i을 BCD로 변환하는 알고리즘은 기존의 SD number를 이용한 방법과 같고, 부분곱 축약단계의 지연시간 분포도 거의 동일하므로 프리픽스 네트워크 구조 또한 동일하게 사용한다.

Ⅳ. 실험 및 성능평가

4.1. 검증 및 시뮬레이션

본 논문에서 제안한 병렬 십진 곱셈기는 Verilog 언어를 이용하여 설계하였으며, 올바른 동작을 수행하는지 확인하기 위한 RTL(Register Transfer Level) 시뮬레이션(simulation)은 Mentor Graphics사의 Modelsim을 이용하여 수행하였다. 하위 모듈에 대한 시뮬레이션 검증을 마친 후 최종적으로 최상위 모듈에 대해 테스트 벡터(test vector) 10000개를 이용하여 동작을 검증하였다.

4.2 실험 결과

제안한 병렬 십진 곱셈기의 성능 평가를 위해 기존의 SD number를 이용한 병렬 십진 곱셈기와 동일한 ASIC 환경에서 합성하여 이를 비교하였다. 설계된 디자인의 합성은 Synopsys사의 Design compiler를 사용하였고, 논리 합성을 위해 SMIC사의 180nm CMOS 공정 라이브러리를 사용하였다.

표 3은 기존의 SD number를 이용한 병렬 십진 곱셈기와 제안한 병렬 십진 곱셈기를 동일한 환경과 컴파일 옵션을 이용하여 합성한 결과이다. 합성 결과 기존의 SD number를 이용한 병렬 십진 곱셈기의 임계경로 지연시간은 3.88ns, 면적은 $549050.2\mu m^2$를 얻을 수 있었으며, 제안한 병렬 십진 곱셈기의 임계경로 지연시간은 3.72ns, 면적은 $520618.0\mu m^2$를 얻을 수 있었다.

그림 5는 부분곱 생성단계, 부분곱 축약 단계, 최종합

표 3. 합성 결과
Table 3. Systhesis result.

Architecture	Delay		Area	
	(ns)	Ratio	(μm^2)	Ratio
[10] Liu Han et al.	3.88	1.046	549050.2	1.055
Proposed	3.72	1.000	520618.0	1.000

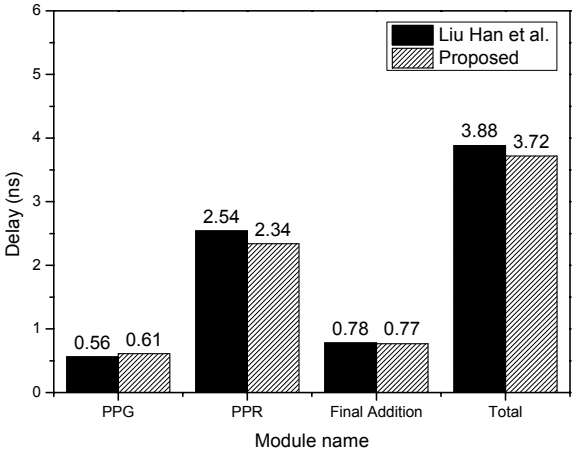

그림 5. 각 하위 모듈의 지연시간 비교
Fig. 5. Delay comparison of each sub-module.

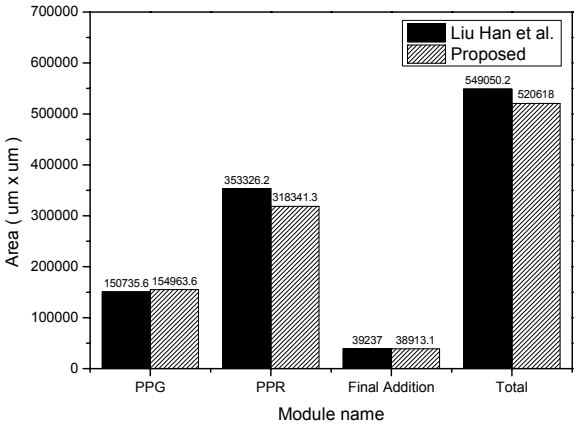

그림 6. 각 하위 모듈의 면적 비교
Fig. 6. Area comparison of each sub-module.

단계를 나누어 지연시간을 분석한 결과이며, 그림 6은 각 단계별 면적을 분석한 결과이다. 제안한 부분곱 생성 단계에서 $3X$와 $5X$의 인코딩 변경으로 인해 논리식이 복잡해져 지연시간이 약 9.2%, 면적이 약 2.7% 증가하였다. 하지만 인코딩을 변경하여 부분곱 축약 단계에서 한 번에 5개의 부분곱을 축약 가능하기 때문에 부분곱 축약 단계에서의 지연시간은 2.54ns에서 2.34ns로 약 8.5% 감소하였고, 17번째 부분곱 인코딩 로직과부분곱의 자릿수를 줄임으로써 면적은 약 10.9% 줄어듦을 확인할 수 있다. 전체 지연시간 및 면적 중에서 부분곱 축약 단계가 차지하는 비중이 60% 이상으로 가장 크기 때문에 부분곱 생성 단계에서 약간의 지연시간 및 면적 증가가 있었지만, 전체 지연시간이 4.6%, 면적이 5.5% 감소한 결과를 얻을 수 있다.

다른 병렬 십진 곱셈기 아키텍처와의 비교를 위하여 합성한 결과를 FO4 지연시간과 NAND2 등가게이트

면적으로의 변환이 필요하다. FO4는 지연시간을 팬아웃(fan-out)이 4인 인버터의 지연시간 개수로 전체 지연시간을 표현하는 것이고, NAND2 등가게이트 면적은 NAND2 게이트의 셀 면적으로 전체 면적을 나누어 면적을 나타내는 것이다. FO4 지연시간과 NAND2 등가게이트 면적은 CMOS 공정에 독립적인 비교 단위가 되므로 서로 다른 환경에서 합성한 결과 간의 공정한 비교가 가능하다.

SMIC사의 180nm CMOS 공정 라이브러리의 FO4는 90ps, NAND2 게이트 셀의 면적은 $10.976\mu m^2$로 계산하여 변환하면 기존의 SD number를 이용한 병렬 십진 곱셈기는 43.11 FO4 지연시간, 50000 NAND2 등가게이트 면적을 얻을 수 있고, 제안한 병렬 십진 곱셈기는 41.33 FO4 지연시간, 47400 NAND2 등가게이트 면적과 같다. [10]에서 기존의 SD number를 이용한 아키텍처는 43.1 FO4 지연시간, 49900 NAND2 등가게이트 면적으로 표기되어 있는데, 거의 동일하게 구현하여 비교하였음을 확인할 수 있다. FO4 지연시간과 NAND2 등가게이트 면적으로 변환한 수치를 이용해 다른 병렬 십진 곱셈기와의 성능 비교를 수행한다.

표 4는 비교대상이 되는 병렬 십진 곱셈기들의 합성 결과를 FO4 지연시간과 NAND2 등가게이트 면적으로 비교한 결과이다. 이때 기존 부호화 자릿수를 이용한 병렬 십진 곱셈기의 FO4 지연시간과 NAND2 등가게이트 면적은 [10]의 결과를 따르므로 실제 구현하여 합성한 결과와 약간의 차이가 있을 수 있다. FO4 지연시간과 NAND2 등가게이트 면적으로 다른 병렬 십진 곱셈기와 비교한 결과 제안한 병렬 십진 곱셈기가 [15]의 SD Radix-5 아키텍처와 비교하여 지연시간은 12.6%, 면적은 4.9% 향상되었으며, [10]과 비교하여 지연시간은 4.3%, 면적은 5.3% 향상됨을 확인하였다. 따라서 제안한 병렬 십진 곱셈기가 최근 연구된 다른 병렬 십진 곱셈기보다 효율적인 아키텍처라 할 수 있다.

표 4. 다른 병렬 십진 곱셈기와의 비교
Table 4. Comparison with another parallel decimal multiplier.

Architecture	Delay		Area	
	(FO4)	Ratio	(NAND2)	Ratio
[13] T. Lang et al.	58.89	1.425	68000	1.456
[14] G. Jaberipur et al.	53.53	1.295	79600	1.679
[15] A. Vázquez et al. SD Radix-10	46.94	1.136	44500	0.939
[15] A. Vázquez et al. SD Radix-5	46.53	1.126	49700	1.049
[10] Liu Han et al.	43.11	1.043	49900	1.053
Proposed	41.33	1.000	47400	1.000

V. 결 론

십진 부동소수점 연산의 중요성이 부각됨에 따라 십진 부동소수점 연산기의 성능을 향상시키기 위한 많은 연구가 진행되고 있다.

본 논문에서는 고정 소수점 병렬 십진 곱셈기의 성능을 향상시키기 위해 향상된 SD number 인코딩과 축약 단계를 이용한 병렬 십진 곱셈기를 제안한다. 승수와 피승수를 제한된 숫자 범위의 SD number로 인코딩하여 캐리 전달 지연 없이 빠르게 부분곱을 생성하고, 인코딩에 사용하는 숫자의 범위를 줄임으로써 한번에 축약 가능한 피연산자의 개수를 늘려 부분곱 축약 단계를 개선하였다.

제안한 병렬 십진 곱셈기의 성능 평가를 위해 Design Compiler에서 SMIC사의 180nm CMOS 공정 라이브러리를 이용하여 합성한 결과 기존의 SD number를 이용한 병렬 십진 곱셈기와 비교하여 지연시간은 4.3%, 면적은 5.3% 감소함을 확인하였다. 인코딩을 변경하여 부분곱 생성 단계에서 약간의 지연시간 및 면적 증가가 있음에도 불구하고 전체 지연시간 및 면적에서 부분곱 축약 단계가 차지하는 비중이 가장 크기 때문에 전체 지연시간 및 면적이 감소함을 확인하였다.

참 고 문 헌

[1] M. F. Cowlishaw, *The decNumber C Library v3.68*, IBM Corportation, http://speleotrove.com/decimal/decnumber.pdf, 2011.
[2] Sun Microsystems, *BigDecimal class, java 2 plaform standard edition 5.0, API specification*, http://java.sun.com/j2se/1.5.0/docs/api/java/math/BigDecimal.html, 2011.
[3] Intel Corportation, *Intel Decimal Floating-Point Math Library*, http://software.intel.com/en-us/articles/intel-decimal-floating-point-math-library/, 2011.
[4] 이창호, 김지원, 황인국, 최상방, "이중 경로 십진 부동소수점 가산기 설계," 전자공학회 논문지, 제 49권 9호, 183-195쪽, 2012년.

[5] L. Eisen, J. W. Ward III, H. W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, "IBM POWER6 Accelerators: VMX and DFU," *IBM J. Research and Development*, vol. 51, no. 6, pp. 663-684, 2007.
[6] A. Y. Duale, M. H. Decker, H. G. Zipperer, M. Aharoni, and T. J. Bohizic, "Decimal Floating-Point in z9: An Implementation and Testing Perspective," *IBM J. Research and Development*, vol. 51, no. 1/2, pp. 217-228, 2007.
[7] Decimal IP, SilMinds [Online]. Available: http://www.silminds.com/decimal-products.
[8] IEEE, *IEEE 754 Standard for Binary Floating-Point Arithmetic*, 1985.
[9] IEEE, *IEEE 754-2008 Standard for Floating-Point Arithmetic*, 2008.
[10] Liu Han, Seok-Bum Ko, "High Speed Parallel Decimal Multiplication with Redundant Internal Encodings," *IEEE Trans. on Computers, Early Access Articles*, 2012.
[11] R.K. Richards, "Arithmetic Operations in Digital Computers", *Van Nostrand*, 1955.
[12] G. Jaberipur and B. Parhami, "Constant-time addition with hybrid-redundant numbers: Theory and implementations," *Integration, the VLSI journal*, vol. 41, pp. 49-64, 2008.
[13] T. Lang and A. Nannarelli, "A Radix-10 Combinational Multiplier," *40th Asilomar Conference on Signals, Systems and Computers*, pp. 313-317, Oct. 2006.
[14] G. Jaberipur and A. Kaivani, "Improving the Speed of Parallel Decimal Multiplication," *IEEE Trans. on Computers*, vol. 58, No. 11, pp. 1539-1552, Nov. 2009.
[15] A. V´azquez, E. Antelo, and P. Montuschi, "Improved Design of High-Performance Parallel Decimal Multipliers," *IEEE Trans. on Computers, vol. 59*, No. 5, pp. 679-693, May 2010.

저 자 소 개

황 인 국(학생회원)
2011년 인하대학교 전자공학과 학사 졸업.
2011년~현재 인하대학교 전자공학과 석사과정.
<주관심분야 : 컴퓨터 아키텍처, SoC 설계>

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

윤 완 오(학생회원)
2000년 경기대학교 전자공학과 학사 졸업.
2002년 인하대학교 전자공학과 석사 졸업.
2010년 인하대학교 전자공학과 박사 졸업.
2010년~현재 인하대학교 정보전자공동연구소 연구교수.
<주관심분야 : 병렬 및 분산 처리 시스템, 컴퓨터 아키텍처>

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>