Booth 알고리즘을 이용한 새로운 VQB 제산/제곱근 연산기의 설계

이 성 연, 이 태 영, 이 용 석
연세대학교 전자공학과
서울시 서대문구 신촌동 134 번지
Tel 02)361-2872, Fax 02)312-4584
lsy@dubiki.yonsei.ac.kr

New VQB divide/square root operator that uses Booth algorithm

Sungyoun Lee, Taeyoung, Yongsurk Lee
Dept. of Elec. Eng., Yonsei University
134, Shinchon-dong, Seodaemun-gu, Seoul
Tel 02)361-2872, Fax 02)312-4584
lsy@dubiki.yonsei.ac.kr

요 약

본 논문은 Booth 알고리즘을 사용하는 새로운 VQB 제산기를 제안한다. 본 논문은 Macsorley 의 제산 알고리즘에 기본 원리가 같은 제곱근 알고리즘을 추가하였으며, 이를 VQB 알고리즘이라고 명명하였다. 본 논문은 VQB 제산기의 두 가지 설계를 구현하였다. 하나는 계수를 사용하지 않는 설계 (A) 이며, 둘은 [1/2,2]의 계수군을 사용하는 설계 (B)이다. 설계 (A)는 순환할 때마다 2.54 비트의 부분 몫을 결정하며, 설계 (B)는 2.74 비트를 결정한다. 본 논문은 VQB 제산기의 성능지표를 좌우하는 제곱근을 위주로 하여, SRT 제산기와의 비교를 시도하였다. VQB 는 처리량과 설계 노력 면에서 SRT 를 앞서며, 면적과 임계지연 면에서는 SRT 와 서로 견줄만한 수준이다. 표준셀 0.35μ m CMOS 공정으로 구현될 때, 설계 (A)의 임계지연은 9.69ns 이며, 설계 (B)는 11.05ns 이다.

1. 서 론

제산과 제곱근은 사용 빈도가 가산과 승산에 비해 매우 낮음에도 불구하고, 부동소수점연산기(FPU)의 필수적인 명령어중 하나이다. 또한 대기 시간(latency)이 길기 때문에, 부동소수점연산기의 성능에 미치는 영향은 사용 빈도에 비하여 크다. 따라서, 제산과 제곱근은 대기 시간의 단축이 중요하다.

최근, 대부분의 고성능 부동소수점연산기는 SRT 제산기를 채택하고 있다. SRT 제산기는 뺄셈방식이므로, 곱셈기의 사용으로 인한 부동소수점연산기의 성능감소가 발생하지 않으며, 이와 동시에 큰 기수(radix)로 설계함으로써, 대기 시간이 짧게 구현할 수 있다는 장점 있다. 그러나 최근에는 부동소수점연산기의 설계에 있어서, 임계 지연(critical delay)의 단축이 가장 중요한 목적이 되었기 때문에, 기수의 증가에 따라 임계 지연도 길어지는 SRT 제산기의 경우, 작은 기수 쪽으로 설계 방향을 선회하는 경우가 많다. 또한, SRT 제산기는 복잡한 이론과, 테이블기반의 구조 때문에, 부동소수점연산기내에서의 중요도에 비해서, 오랜 설계 시간을 필요로 한다.

우리는 제산의 몫 결정량을 증가시킬 수 있는 간단한 알고리즘을 1960 년대 Macsorley 의 논문에서 발견하였다. 그러나 Macsorly 가 제시한 제산 알고리즘과 관련하여, 제곱근에의 적용을 다룬 논문이나 집적회로 구현을 통해 성능지표를 제시한 논문이 발표된 바가 없다. 따라서 우리는 Macsorly 의 알고리즘을 이용하며, 더욱이 제곱근을 수행하는 새로운 제산기를 집적회로로 구현하였다. 결론부터 말하자면, 제안된 제산기는 동급의 SRT 보다 대기 시간이 짧으며, 설계하는데 필요한 노력은 적다.

2. VQB 알고리즘

최근, 대부분의 승산기는 Modified Booth 알고리즘을 이용하여 구현되고 있다. 제산은 승산의 역과정이기 때문에 Booth 알고리즘은 제산, 더불어 제곱근에도 적용될 수가 있다. Booth 알고리즘이 '0'의 문자열과 '1'의 문자열의 처리를 통해 연산의 횟수를 줄이듯이, 제산과 제곱근 역시 이 한가지 이진숫자로 이루어진 문자열(이하 문자열)을 이용한다. VQB 제산기는 매 순환마다 문자열의 길이에 따라 결정되는 부분 몫의 길이가 가변적이다. 따라서 우리는 이 알고리즘을 VQB(Variable Quotient Bit) 제산과 제곱근이라고 명명하였다.

피연산자(operand)들의 범위는 (1)과 같다.

1≤ |부분 피제수|, 제수, 레지듀얼 < 2 (1)

제수 또는 레지듀얼(residual)의 배수가 부분 피제수로부터 가/감 된다. 부분 몫(이하 q)은 가/감산의 결과인 부분 나머지로부터 구해지며, 지금까지 결정된 총 몫(이하 Q)에 덧붙여진다. 부분나머지의 부호로부터 q 의 최상위비트가 결정되며, q 의 나머지 비트들은 이어지는 문자열로부터 쉽게 결정된다. 그러면, 영의 문자열과 일의 문자열이 어떻게 처리되는지 살펴보도록 하자. 다음의 두 소 단원은 문자열의 길이가 3 비트인 경우를 예로 들어, 처리 과정을 설명한다.

(1) 0 의 문자열

양의 부분 나머지는 부호 비트 '0', 부호확장 비트, 0 의 문자열((2)에서 밑줄 표시)을 순서대로 가진다(2). 제수에는 계수가 곱해지기 때문에, 가장 큰 계수가 곱해진 제수를 표시할 수 있을 만큼의 부호확장 비트가 필요하다. 피연산자들의 범위가 (1)과 같기 때문에, 결과인 부분 나머지는 항상 1 비트 이상의 영의 문자열을 가진다. 부분 나머지가 양이라는 것은 제수로부터 피제수가 완전히 감해진 것을 의미하므로 q 의 최상위 비트는 1 이 된다. 영의 문자열은 그 길이가 길수록 부분나머지가 제수보다 작다는 것을 의미한다(2)(3). 따라서, 부분 나머지는 영의 문자열의 길이만큼 왼쪽으로 쉬프트되어 다음 순환에서 (1)의 범위에 있는 부분 피제수가 되며(4), 이 때, 영의 문자열에서 최상위 0 을 제외한 부분이 추가의 몫으로 결정된다(5). 결국, 구해진 q 의 길이와 문자열의 길이가 같다. (2)(5)

*000.00**1100111**…* ; 부분나머지 (2)
000.010100101… ; 제수 (3)
*000.100111*010… ; 쉬프트된 부분 나머지 (4)
{1(최상위비트),00(추가의 몫) }; q (5)

(2) 일의 문자열

음의 부분 나머지는, 부호 비트 '1', 부호확장 비트, 일의 문자열((6)에서 밑줄표시)을 순서대로 가진다. 부분 나머지의 음의 부호는 제수가 피제수로부터 완전히 감해질 수 없음을 의미하므로, q 의 최상위 비트는 '0'으로 결정된다. 그리고, 실제로 빼어진 양은 q 의 최상위 아래의 비트들에서 나타난다. 일의 문자열의 길이는 그 길이만큼 부분나머지의 절대값이 제수보다 작다는 것을 의미한다(6)(7). 따라서 일의 문자열 어느 위치에서 제수를 더하더라도 그 결과는 양이 될 것이며, 이 위치까지의 몫은 1 로 결정될 것이다. 이는 Booth Algorithm 에서의 일의 문자열의 처리와 방향은 반대이지만 내용은 같은 것이다. 따라서 부분 나머지는 일의 문자열의 크기만큼 왼쪽 shift 되어 (1)의 범위에 있는 부분 피제수(8))가 된며, 1 의 문자열에서 최상위 1 을 제외한 부분이 추가의 몫으로 결정된다(9) 결국, 구해진 q 의 길이와 문자열의 길이가 같다(6)(9).

1*1**111**0011000…* ; 부분나머지 (6)
001010100101… ; 피제수 (7)
*1100110001*01… ; 쉬프트된 부분 나머지 (8)
{0(MSB),11(additional quotient) } ; q (9)

(3) 몫 결정

위의 두 단원에서 설명한 몫 결정을 요약하면 다음과 같다.

a) 결정 규칙 I : q 의 최상위 비트는 부분나머지의 부호의 부정(not)이다.

b) 결정 규칙 II : 추가의 몫은 문자열에서 최상위 비트를 제외한 부분으로 결정된다.

q = {부분 나머지의 부호의 부정, 문자열에서 최상위 비트를 제외한 부분}
(10)

2) 제수 조정

위의 논의로부터, 부분 나머지의 문자열의 길이가 바로 q 의 길이가 됨을 알았다. 따라서, 제수를 가/감하기 이전에 제수에 어떠한 계수를 곱하여 그 크기를 조정함으로써 결과인 부분 나머지의 절대값을 작게 한다면 더 많은 비트의 q 를 결정할 수 있을 것이다. 따라서 제수를 조정하기 위해 어떠한 계수들을 사용하느냐에 따라서, 성능이 개선되는 정도가 달라진다. VQB 에서의 제수 조정은 문자열의 길이를 늘리기 위한 것이라는 점에서, 제수를 1 에 근사하도록 조정하는 'pre-scaling 제산기'와는 다르다.

우리는 두 가지 설계를 하였다. 하나는 제수조정을 하지 않는 설계 (A)이며, 둘은 [1/2,2]의 계수군을 사용하는 설계 (B)이다.

3) 제곱근

이 단원에서는 VQB 제곱근 알고리즘을 소개한다. 제곱근에서 VQB 가 SRT 와 크게 다른 점은 q 를 알기 이전에 레지듀얼을 생성하여야 한다는 점이다. 따라서 우리는 정확한 q 와는 다른 임시적인 q 를 사용하게 되며, 임시적인 q 는 부분 피제수의 부호를 가지는 제수 조정 계수(부호화된 계수)로 결정한다. 레지듀얼을 생성하는 과정은 다음과 같다. 먼저 부호화된 계수가 Q 에 붙여진다. 다음, 레지듀얼은 십진수로 계산되며, 따라서, Q 보다 한 비트 아래에 위치하는 '부호화된 계수'는 십진 식에서는 절반이 되어 나타난다(11). 계산된 레지듀얼은 쉬프트를 포함한 이진 형태로 표현되며, Q 에 생성부분이 덧붙여진 형태로 나타난다(12).

{Q, 부호화된 계수} -> Q + 부호화된 계수/2 (11)

{Q, 부호화된 계수}의 레지듀얼
= 피제곱근중에서 *부호화된 계수*로 인한 양
= 2*Q*(*부호화된 계수*/2) + (*부호화된 계수*/2)^2
= ± (Q + 2*생성부분) * 2^i
= {Q, 생성부분} << i (12)

다음 각각의 경우에서, 부호화된 계수는 굵은 글씨로, 절반이 된 부호화된 계수는 이탤릭체로 표시하였다.

(1) 레지듀얼이 감해질 때

임시적인 q 를 양의 계수로 결정한다.

계수가 1 일 경우, 임시적인 q 도 **1** 로 간주되며, 결과 생성부분은 01 이다.

{Q,**1**}의 레지듀얼 = 2*Q**1/2* + (*1/2*)^2
= Q + 1/4 = {Q,01} (13)

계수가 1/2 일 경우, 임시적인 q 도 **1/2** 이며, 생성부분

은 001 이다.

$\{Q,\mathbf{1/2}\}$의 레지듀얼 $= 2*Q*1/4 + (1/4)^2$
$= (Q+1/8)*2^{-1} = \{Q,001\}>>1$ (14)

계수가 2 일 경우에는, 다시 두 가지 경우를 각각 계산해 보아야 한다. 즉, Q 의 최하위 비트가 각각 '0', '1' 인 경우이다. 계수 2 는 이진 형태일 때, Q 의 최하위비트와 같은 위치에 있기 때문에, Q 에서 최하위비트를 제외한 부분인 Q_{LSB} 를 기준으로 레지듀얼을 계산한다. 이 때 Q 의 최하위비트는 계수와 함께 상수로 취급된다(15)(16).

i) 최하위비트 = 0
$\{Q_{LSB},0,\mathbf{2}\}$의 레지듀얼 $= \{Q_{LSB},\mathbf{1}\}<<1$ 의 레지듀얼
$= \{2*Q_{LSB}*1/2+ (1/2)^2\}*2 = (Q_{LSB}+1/4)*2$
$= \{Q_{LSB},01\}<<1 = \{Q,1\}<<1$ (15)

ii) 최하위비트 = 1
$\{Q_{LSB},1,2\}$의 레지듀얼
$= \{Q_{LSB},\mathbf{2}\}$의 레지듀얼<<1 - $\{Q_{LSB},\mathbf{1}\}$의 레지듀얼<<1
$= \{Q_{LSB}*1+ 1^2\}*2 - \{Q_{LSB}*1/2 + (1/2)^2\}*2$
$= (Q_{LSB}+3/4)*2 = \{Q_{LSB},11\}<<1 = \{Q,1\}<<1$ (16)

두 가지 경우의 결과 생성부분은 같다(15)(16).

(2) 레지듀얼이 더해질 때
이 경우, Q 는 실제로 이루어진 동작을 표시하는 '동작 몫'보다 항상 1 만큼 작게 결정되어 있는 상태이므로, 레지듀얼을 계산한 때에는, Q 가 아닌 Q+1 을 사용하여야 실제 동작과 맞다. 임시적인 q 는 음의 계수로 결정된다.
계수가 -1 일 때에는, 임시적인 q 도 -1 로 결정되며, 결과 생성부분은 11 이다.

$\{Q+1,\mathbf{-1}\}$의 레지듀얼 $= 2*(Q+1)*-1/2 + (-1/2)^2$
$= -(Q+3/4) = \{Q,11\}$ (17)

계수가 -1/2 일 때에는, 임시적인 q 는 -1/2 이며, 생성부분은 111 이다.

$\{Q+1,\mathbf{-1/2}\}$의 레지듀얼 $= 2*(Q+1)*-1/4 + (-1/4)^2$
$= -(Q+7/8) *2^{-1} = \{Q,111\}>>1$ (18)

계수가 -2 일 때에는 계수가 2 인 경우와 마찬가지 이유로, 두 가지 경우에서 계산한다(19)(20). 임시적인 q 는 -2 이며, 생성부분은 1 이다.

i) 최하위비트 = 0
$\{Q_{LSB},0+1,\mathbf{-2}\}$의 레지듀얼
$= \{Q_{LSB},\mathbf{0}\}$의 레지듀얼<<1 - $\{Q_{LSB},\mathbf{1}\}$의 레지듀얼<<1
$= 0 - \{2*Q_{LSB}*1/2+ (1/2)^2\}*2 = -(Q_{LSB}+1/4)*2$
$= \{Q_{LSB},01\}<<1 = \{Q,1\}<<1$ (19)

ii) 최하위비트 = 1
$\{Q_{LSB},1+1,\mathbf{-2}\}$의 레지듀얼 = $\{Q_{LSB},2,\mathbf{-2}\}$의 레지듀얼
$= \{Q_{LSB},\mathbf{1}\}<<1$ 의 레지듀얼 - $\{Q_{LSB},\mathbf{2}<<1\}$의 레지듀얼
$= \{Q_{LSB}*1/2 + (1/2)^2\}*2 - \{Q_{LSB}*1+ 1^2\}*2$
$= -(Q_{LSB}+3/4)*2 = \{Q_{LSB}, 1\}<<1$
$= \{Q,1\}<<1$ (20)

두 가지 경우의 결과 생성부분은 같다(19)(20).

3. VQB 제산과 제곱근을 위해 제안된 하드웨어 구조

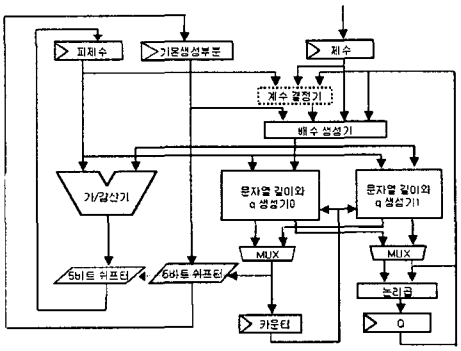

그림 1. 전체 블록도

그림 1 에서 점선으로 표시한 계수 결정기를 생략하면 설계 (A)의 전체 블록도가 된다. 먼저 계수 결정기는 피제수의 부호 비트와 피제수, 제수, Q 각각의 소수점 이하 2 비트를 입력으로 받아, 피제수와 제수(제곱근시에는 Q)의 크기를 대략적으로 비교하여, 얼마의 계수를 사용할 지 결정한다.

	피제수 [54:51]	제수 [53:51]	계수
계수 결정기	01.00	1.11	1/2
	10.11	1.11	1/2
	01.11	1.00	2
	10.00	1.00	2
	기타의 경우		1

표 1. 계수결정기의 진리표

표 1 에서 제수가 상대적으로 크면, 제수를 절반으로 줄이기 위해 계수 1/2 을 제수가 적으면, 두 배로 늘리기 위해 계수 2 를 출력한다. 그림 2 의 배수 생성기는 레지듀얼을 생성하며, 제수나 레지듀얼에 계수를 곱한다. 두 개의 '문자열 길이와 q 생성기' 그리고 가/감산기는 동시에 동작한다. 알고리즘에서는 가/감산이 끝난 후, 문자열 길이와 q 의 생성이 시작되지만, 이 경로는 지연이 너무 길다. 또한 그림 3 에서 문자열 길이와 q 생성기는 부분 나머지중에서 부호 확장 비트와 뒤따르는 5 비트만을 필요로 한다. 따라서, 두 개의 병렬 블록은 입력 올림수(carry)가 각각 '0'과 '1'인 경우에 대해서 미리 이 6 비트 를 계산하여, 문자열 길이 검출과 q 생성에 이용한다. 이 때 실제의 가/감산도 동시에 이루어진다

표 2 은 그림 2 내에 있는 문자열 길이 생성기의 진리표이며, 그림 4 는 q 생성기의 진리표이다. 본 구현에서는 피제수와 제수의 길이가 56 비트이므로, 48 번째 비트에서 49 번째로 넘어가는 올림수가 위의 두 병렬

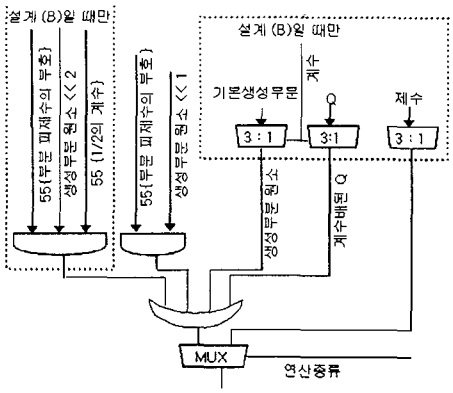

그림 2. 배수 생성기

	6비트 입력	q 마스크	문자열 길이
문자열 길이 검출기	11.1111	11111	101
	00.0000	11111	101
	11.1110	11110	100
	00.0001	11110	100
	11.110x	11100	011
	00.001x	11100	011
	11.10xx	11000	010
	00.01xx	11000	010
	11.0xxx	10000	001
	00.1xxx	10000	001
	the others	10000	000

표 2. 문자열 길이 검출기의

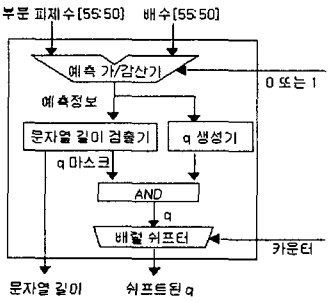

그림 3. 문자열 길이와 q 생성기

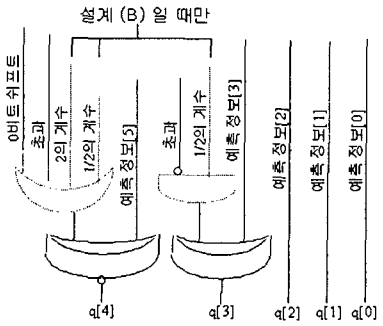

그림 4. q 생성기

블록의 문자열 길이와 q 중 각각 옳은 것을 선택한다. 그림 1 의 논리합(OR) 게이트는, 선택된 q 를 Q 에 축적한다. 선택된 문자열 길이는 카운터에 더해지며, 카운터는 정밀도에 따라 언제 순환을 끝낼지를 알린다. 부분 나머지는 문자열 길이만큼 왼쪽으로 쉬프트되어 다음 사이클의 부분 피제수가 되며, '기본생성부분'은 문자열 길이만큼 오른쪽으로 쉬프트 되어 다음 사이클에 레지듀얼의 생성에 쓰인다.

4. 결 론

우리는 감산 방식의 새로운 VQB 제산/제곱근을 제안한다. 제안을 위해서, 우리는 Macsorley 가 제시한 기본 알고리즘에 제곱근 알고리즘과 구현에 필요한 알고리즘을 추가하여 VQB 알고리즘이라고 명명하였다. 우리는 최초의 세부 구조와 성능 지표를 제시함으로서, 기수 4 의 수준에서는 VQB 가 충분히 SRT 의 대안이 될 수 있음을 입증하였다. 서론에서, 우리는 제산기 설계의 두 가지 방향, 즉 처리량 향상과 계발 기간의 단축을 제의한 바 있다. 제안된 VQB 제산기는 이러한 방향에 부합하는 것으로서, 기수 4 SRT 제산기에 비해 성능이 27%~37% 향상되었으며, 모든 주요 설계 과정에서 소요 시간이 적다. 기수 8 SRT 제산기와 동급인 [3/4,2/3] 계수군의 VQB 제산기 설계가 추후에 연구되어야 할 것이다.

5. 참 고 문 헌

[1] O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proceedings of the Institute of Radio Engineers*(now the IEEE), 49:67-91, 1961.

[2] 이원, 권호경, 이용환, 이용석, "Radix-4 SRT 알고리즘을 사용한 나눗셈/제곱근 연산기에 관한 연구," *전자공학회논문지*, 제 33 권, A 편, 제 9 호, 1996 년 9 월.

[3] 이용석, "RISC Microprocessor Overview," 고성능 마이크로프로세서 구조와 설계 비디오 강좌 시리즈, 1997, http://mpu.yonsei.ac.kr/yslee/RiscOverview.html

[4] Israel Koren, *Computer Arithmetic Algorithms*, Prentice Hall, 1993.

[5] Milos D. Ercegovac, Tomas Lang, and Paolo Montuschi, "*Very-high radix division with prescaling and selection by rounding,*" IEEE trans. On Computers, Vol.43, No.8, pp.909~918, Aug.1994

[6] Peter Soderquist and Miriam Leeser, "Division and Square Root: choosing the right implementation," *IEEE Micro*, Vol. 17, No. 4, pp. 56-66, July/August 1997.