높은 자릿수를 이용한 고속 나눗셈 연산기의 최적화 연구 및 변환 요소 전처리를 위한 설계

이병석, 안성용, 홍승완, 이정아
조선대학교 전자계산학과

Implementation Schemes to Optimize Very-High Radix Dividers in Pre-processing Scaling Factor Design

Byeong-Seok Lee, Seong-Yong Ahn, Seung-Wan Hong, Jeong-A Lee
Dept. Computer Science, Chosun University

요 약

나눗셈 알고리즘은 다른 덧셈이나 곱셈 알고리즘과 비교하여 복잡하고, 수행 빈도수가 적다는 이유로 그동안 고속 나눗셈의 하드웨어 연구는 활발하지 않았다. 그러나 멀티미디어의 발전으로 고속 나눗셈의 필요성 및 전체적인 수행 시간 향상을 위해 고속 나눗셈 연산기의 중요성은 더욱 부각되고 있다. 그러나 칩의 크기는 제작 단가와 깊은 관련이 있기 때문에 고속 나눗셈 연산기를 칩으로 제작할 때 요구되는 성능과 비용을 만족하기 위한 적절한 분석이 필요하다.

본 논문은 자릿수 순환(Digit Recurrence)알고리즘에서 속도가 빠른 높은 자릿수 이용(Very-High Radix) 알고리즘을 기반으로 최적화된 자릿수(Radix) 범위를 제시하였다. 그리고 변환 요소(Scaling Factor)를 전처리(Pre-processing)하여 연산의 주기를 감소하고, 크기 문제를 해결하기 위해서 상수표 대신 제어(Control) 방법으로 값을 구하는 방법을 설계하였다.

1. 서 론

나눗셈 알고리즘은 덧셈이나 곱셈 알고리즘과 비교하여 비교적 복잡하여 소프트웨어로 처리하는 경우 연산 수행 시간이 길어지는 문제점이 있었으나 지금까지는 그 빈도수가 다른 연산작업에 비하여 적다는 이유로 고속 나눗셈 하드웨어의 연구가 활발하지는 않았다. 그러나 멀티미디어의 발달로 나눗셈이 차지하는 비중이 점점 더 커지게 되었고, 나눗셈은 단위 연산 수행 시간이 길기 때문에 빈도수는 적어도 전체 수행 시간에서는 성능에 영향을 미치는 연산이 되어 고속 나눗셈 연산기의 중요성이 더욱 더 부각되고 있다.

현재 가장 널리 사용하고 있는 자릿수 순환(Digit Recurrence) 나눗셈 알고리즘은 2진수(Ordinal), 4진수(Radix-4), 영역 변환(Pre-scaling), 높은 자릿수 이용(Very-High Radix)등이 있다[2] 이 중에서도 높은 자릿수 이용 알고리즘의 연산 속도가 빠르다. 높은 자릿수 이용 나눗셈은 자릿수(Radix)가 크면 클수록 빠른 연산이 가능하지만 그 크기 또한 커지게 된다. 이렇게 되면 칩 내의 나눗셈 부분이 차지하는 공간 문제가 발생할 수 있다.

본 논문은 2장에서 높은 자릿수 이용 알고리즘을 고찰하고, 3장에서는 자릿수 값에 대한 속도 및 크기에 관한 최적화를, 4장에서는 변환 요소(Scaling Factor)인 M을 위한 성능 향상의 위한 설계를, 5장에서는 결론 및 향후 연구 방향에 대해서 논한다.

본 논문에서의 함수 수준의 시뮬레이션은 Active VHDL 3.1, 함수 수준의 시뮬레이션 및 합성(Synthesis)은 Synopsys 98.02로 하였다.

2. 높은 자릿수 이용(Very-High Radix) 나눗셈

높은 자릿수 이용 나눗셈 알고리즘의 특징은 하나의 주기(Cycle)에 여러 개의 q 비트를 찾아낸다[3]. 그리고 이 알고리즘은 $q = x/d$ 일 때 다음 조건(1), (2)를 만족해야 한다.

$$1 > d \geq \frac{1}{2} \quad (1)$$

$$d > x \quad (2)$$

Radix-r에 대한 $\log_2 r = b$인 알고리즘에서 몫 선택(Quotient Select)을 단순하게 하기 위하여 다음 식으로 표현되는 변환 요소(Scaling Factor)인 M 값(3)을 구하여 제수와 피제수의 영역을 변화시킨다.

$$M = -\widehat{\gamma_1} d_k + \widehat{\gamma_2} \quad \text{where,} \quad k = b + 5 \quad (3)$$

위 식의 γ_1 값과 γ_2 값은 다음 식으로 구한다.

$$\gamma_1 = \frac{1}{d_r^2 + d_r 2^{-r} + 2^{-k}} \quad (4)$$

$$\gamma_2 = \frac{2d_r + 2^{-r}}{d_r^2 + d_r 2^{-r} + 2^{-k}} \quad (5)$$

$$\text{where, } r = \left(\left\lfloor \frac{b}{2} \right\rfloor + 2 \right)$$

M값을 제수와 피제수(Dividend)에 곱하여 Scaling된 z(Scaled Divisor)값(6)과 w(Scaled Dividend)값(7)을 구한다.

$$z = Md \quad (6)$$

$$w[0] = Mx \quad (7)$$

중간 나머지(Partial Remainder)에서 상위 b+2비트 값인 $\hat{y}$을 구하여 q_{j+1}(8)을 얻는다. 여기서 q_{j+1}때문에 시간이 늘어나는 것을 방지하기 위해서 새로운 형태로 변환(Recoding)한다.

$$q_{j+1} = \left\lfloor \hat{y} + \frac{1}{2} \right\rfloor, \ (-r < q < r) \quad (8)$$

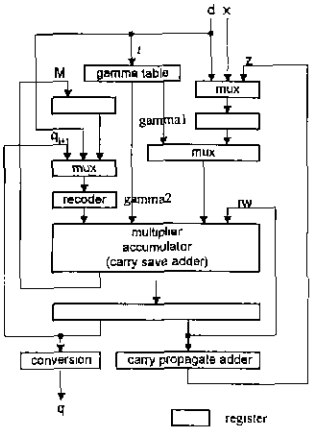
입력 비트 n에서 $\left\lceil \frac{n}{b} \right\rceil$만큼 다음 식(9)을 반복하여 몫을 구한다.

$$w[j+1] = rw[j] - q_{j+1}z \quad (9)$$

여기서 $w[i]$ 값과 와 q_{j+1} 값은 항상 고정된 연산시간을 위해서 캐리 저장(Carry-Save) 형식으로 표현하며, 리코딩(Recoding)을 위해서 Radix-4 형식으로 변환하며[1], 최종 몫은 반올림 및 On-The-Fly 변환으로 얻는다.

본 논문에서는 IEEE754 부동 소수점 표준에서 배정도(Double Precision)의 가수 값인 52비트를 제수와 피제수의 입력으로 한다. 여기서 가수의 MSB는 1비트가 생략되었기 때문에 각각 1비트를 추가하여야 하며, 피제수는 제수보다 작아야 함으로(2) 피제수에는 1비트를 추가하여, 총 제수 53비트, 피제수 54비트를 입력한다. γ_1값과 γ_2 값은 미리 계산하여 Gamma Table에 저장, d_r값에 의해 출력한다.

위 알고리즘에서는 M값을 구하기 위한 연산(3)과 순환식 계산을 위한 연산(9)을 위하여 2개의 누적 연산기(Multiplier Accumulator)가 필요하다. 여기서 누적 연산기를 공유하게 되면 공간을 절약할 수 있다.[4] 높은 자릿수 이용 나눗셈의 블록 다이어그램은 <그림 1>과 같다.



<그림 1> Very-High Radix Divider 블록 다이어그램

3. 자릿수 값에 대한 속도 및 크기의 최적화

높은 자릿수 이용 나눗셈은 자릿수 r에 대한 b비트만큼 q비트를 출력한다. 그러나 r값이 크다고 속도가 향상된다고는 볼 수 없다. 오히려 주기(Cycle)는 줄어들지는 않으면서 Netlist 길이의 및 크기가 증가함으로서 속도는 저하되면서 크기만 커지는 경우도 발생한다. 몫을 구하는데 필요한 주기는 다음과 같이 구한다

$$N_{cycles} = T_M + T_{Md,Mz} + \left\lceil \frac{n}{b} \right\rceil + T_{postcorr-round}$$

$$= \left\lceil \frac{n}{b} \right\rceil + 4$$

$$\text{where, } T_M = 1, \ T_{postcorr-round} = 1, \ T_{Md,Mz} = 2$$

감마(γ) 값은 식의 복잡성에 따라 미리 Table에 저장을 하게 된다. 이에 따라 자릿수의 증가에 따라서 Gamma Table의 크기도 달라지며, Gamma Table의 크기는 $2^r \times (2b+11)$비트이다. 각 자릿수에 대하여 몫을 구하는 주기 및 Gamma Table 크기를 보면 <표 1>와 같다.

$\log_2 r$	4	5	6	7	8	9	10	11
cycle	18	15	13	12	11	10	10	9
bit	152	168	368	400	864	928	1984	2112
$\log_2 r$	12	13	14	15	16	17	18	19
cycle	9	9	8	8	8	8	7	7
bit	4480	4736	9984	10496	22016	23040	48128	50176

<표 1> 각 자릿수에 대한 전체 주기 및 Gamma Table 크기

<표 1>의 자료를 보면 2^5이하는 주기수가 너무 길고, 2^{19}이후는 주기의 변화는 거의 없으면서 크기만 더욱 커지게 되므로 비효율적이다. 그리고 Gamma Table은 r 값에 크게 영향을 받으므로 크기의 변화가 적은 방향으로 성능을 향상시켜야 한다. 종합적으로 살펴보면 최상의 범위는 $2^7 \le r \le 2^{18}$이 된다.

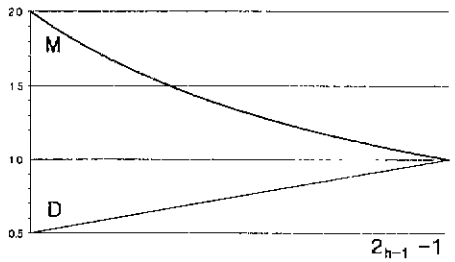
높은 자릿수 이용 나눗셈 알고리즘의 최적화 방향은 크기의 변화가 적으면서 속도를 향상시키는 방법과 속도의 변화가 적으면서 크기를 줄이는 방법이 있다. 이 속도와 크기 문제는 칩을 설계하는데 있어 가장 중요하면서 쉽게 처리할 수 없는 요소이다 속도를 향상시키려면, 연산하는 부분의 최소화, 고속 누적 연산기 설계 등이 있고, 크기를 줄이려면 공간의 최적화된 누적 연산기, 저장장소(Register, ROM)의 크기 감소, 공간 최적화된 모듈 설계 등이 있다. 문제는 속도와 크기는 서로 상반관계이므로 목적에 따른 선택 및 희생이 필요하다

4. 변환요소(Scaling Factor)의 전처리(Pre-Processing)

<그림 1>의 나눗셈 주기 중 변환 요소인 M을 계산하는 주기가 있다. 여기서 M을 미리 계산하면 <그림 1>에서 MUX, Gamma Table은 필요하지 않게 되며, 누적 연산기의 구조도 더 간단해진다. 그리고 M을 계산하지 않게 되므로 T_M 시간이 필요 없게 되며 Radix-r은 한 단계씩 성능이 향상된다.

문제는 M_Table의 크기에서 발생된다. Gamma_Table의 크기는 $2^r \times (2b+11)$ 비트이지만, M_Table의 크기는 $2^h \times (b+2+4)$ 비트이다. 이렇게 되면 크기는 기하급수적으로 증가하기 때문에 속도에 따른 크기 문제에서 완전히 비효율적이다. 이에 대한 해결 방법은 비

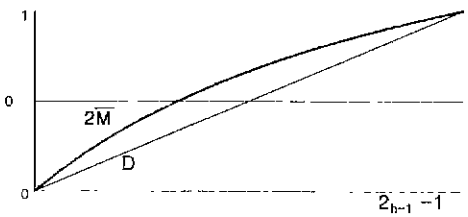
숫한 비트를 생략하여 입/출력 비트의 수를 줄여야 한다. 조건식(1)을 보면 d는 0.5와 1 사이에 있고, M의 범위는 $1<M<2$ 이다. 이것은 d의 MSB는 항상 '1'비트이고, M의 정수 비트 영역에는 항상 '01'비트이다. 항상 존재하는 비트를 생략하면 M_Table의 크기는 $2^{h-1}\times(b+4)$ 비트로 줄어든다. 그러나 입력 비트가 1비트밖에 줄어들지 않음으로 입력 비트가 상대적으로 적은 Gamma_Table과의 크기는 많은 차이가 나며, 자릿수가 커질수록 그 차이는 기하 급수적으로 커진다. 여기서 입력 d_h(D) 값에 대한 출력 M값을 그래프로 보면 <그림 2>와 같다.



<그림 2> d_h(D)에 따른 M의 관계

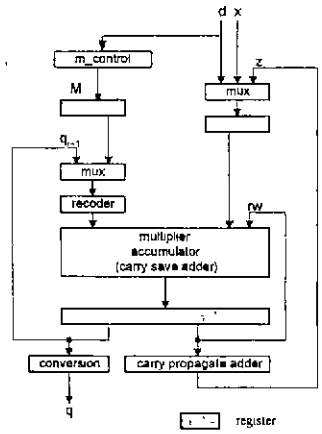
<그림 2>에서 보면 d_h의 값이 클수록 M값은 작아지며, 1에 가까운 M값이 인접한 D값과 비슷하거나 같은 값을 갖는 경우가 발생한다. 이런 현상이 발생한 이유는 레지스터에 저장하는 값이 한정된 정확도이기 때문이다. 즉, M 값은 서로 다르지만, $(b+4)$비트 이하의 비트는 버리게 됨으로, M의 인접 값들이 비슷하거나 같게 된다.

여기서 M값을 쉽게 처리하기 위하여 M비트 값에 1의 보수를 하게 되면 D값이 증가함에 따라 M값 또한 증가하게 된다. 그리고 2를 곱하게 되면 D비트의 크기와 M비트의 크기는 같아진다. 이에 대한 그래프는 다음과 같다.<그림 3>



<그림 3> 변환된 d_h(D)에 따른 M의 관계

<그림 3>의 그래프를 보면 d_h와 $2\overline{M}$은 0에서는 서로 중복된 값을 갖거나 거의 비슷한 값을 갖고 있다. 그리고 $2\overline{M}$은 $2_{h-1}-1$에 근접할수록 서로 인접한 값의 중복 비율이 높다. 이러한 현상을 이용하여 M값을 ROM에 저장하는 M_Table방식이 아닌, 중복된 값과 인접한 값을 직접 제어하는 M_Control로 구현하면 크기는 크게 늘이지 않으면서 빠른 높은 자릿수 이용 나눗셈을 설계할 수 있다. 그리고 제어 게이트는 ROM보다는 작기 때문에 M_Table 보다는 크기가 크게 줄어든다. 또한 크기 비교 대상이 동급이 아닌 한 등급 위와 비교를 하므로 크기에 대한 제약성이 어느 정도 해결된다. M 제어 방법을 이용한 높은 자릿수 이용 나눗셈의 블록 다이어그램은 <그림 4>와 같다.



<그림 4> M_Control을 사용한 Very High Radix Divider 블록 다이어그램

5. 결론

본 논문에서는 Radix-r과 γ값을 통하여 높은 자릿수 영역 나눗셈의 최적화 할 수 있는 r의 범위를 제시하였다. 그러나 실제 구현에서 툴(Tool)의 종류, 셀의 라이브러리(Sell Library), 설계 과정(Design Flow), 옵션(Option) 및 최적화(Optimization) 명령에 따라서 큰 차이가 난다.

그리고 속도의 향상을 위해서 M을 미리 처리하는 방법을 제안하였다. M을 미리 처리하면 주기의 감소로 속도는 향상되지만, 그에 따른 크기의 증가가 문제가 된다. 그러나 M의 크기 문제를 해결하면 높은 자릿수 이용 나눗셈의 성능은 향상된다. 이 모든 것은 현재 구현 중이며 M_Control은 복잡성에 의한 합성(Synthesis)의 어려움으로 복잡성 제거에 현재 중점을 두고 있다.

현재 높은 자릿수 이용 나눗셈을 실제 물리적 설계(Physical Design)를 통한 속도와 크기를 비교한 최적의 모델 제시 연구가 진행 중이며, 다른 자릿수 순환 나눗셈 알고리즘과의 비교 연구를 통한 최적화 모델 제시 및 제안된 M크기의 큰 원인 d_h을 적은 비트로서 M을 찾는 연구가 필요하다.

참 고 문 헌

[1] M.D. Ergegovac, T. Lang, "*Simple Radix-4 Division with Operands Scalling*", IEEE Trans, Comput., vol. 39, pp.1204~1208, 1990.

[2] Israel Koren, "*Computer Arithmetic Algorithm*", Prentice-Hall, Inc , 1993.

[3] M.D. Ergegovac, T. Lang, and P.Montuschi, "*Very-High Radix Division with Prescaling and Selection by Rounding*", IEEE Trans. Comput., vol. 43, pp.909~918, 1994.

[4] A Nannarelli, "*Implementation of a Radix-512 Divider*", Master thesis, Univ. of Calif., 1995.