# 효율적인 부분곱의 재배치를 통한 고속 병렬 Floating-Point 고속연산기의 설계

김 동 순, 김 도 경, 이 성 철, 김 진 태, 최 종 찬
전자부품연구원
전화 : 031-610-4323 / 핸드폰 : 016-310-6649

# Design of Fast Parallel Floating-Point Multiplier using Partial Product Re-arrangement Technique

Dong-Sun Kim, Doh Kyung Kim, Sung Chul Lee, Jin Tae Kim, and Jong Chan Choi
Korea Electronic Technology institute
E-mail : dskim@nuri.keti.re.kr

## Abstract

Nowadays ARM7 core is used in many fields such as PDA systems because of the low power and low cost. It is a general-purpose processor, designed for both efficient digital signal processing and controller operations. But the advent of the wireless communication creates a need for high computational performance for signal processing. And then This paper has been designed a floating-point multiplier compatible to IEEE-754 single precision format for ARM7TDMI performance improvement..

## I. 서론

곱셈연산은 신호처리 및 프로세서와 같은 많은 응용분야, 즉 Ray tracing 과 같은 3D 그래픽 처리, AAC와 같은 음원의 압축 및 복원 및 디지털 필터와 같은 다양한 신호처리시스템뿐만 아니라 특수 목적용 하드웨어의 임계 경로를 결정하는데 중요한 영향을 미치는 부분이다. 인텔의 33MHz 80386의 경우 곱셈연산을 위한 별도의 연산기가 없이 ALU를 통해 곱셈연산을 수행하도록 설계되었는데, 32bit 정수의 곱셈을 위해서 44 clock cycle이 소요되어 연산 량이 많은 응용부분에 있어서 연산의 병목 현상을 초래하게 되었으며, 효과적인 연산기 구조의 중요성이 대두되었다. 이와 함께 최근에 PDA와 같은 embedded CPU를 이용한 독립형 기기들의 개발이 다양한 방법으로 이루어지고 있으며, 휴대용 단말기의 전력소모를 줄이기 위해 ARM사의 RISC CPU를 사용하는 경우가 증가하고 있다. 하지만, ARM사의 대표적인 CPU인 ARM7TDMI의 경우 내부에 수정 Booth 알고리즘을 사용한 전용 곱셈기 블록이 존재하지만 32x32연산을 위해서는 4 cycle이 필요하며, 영상이나 음성의 처리를 위해 floating-point 연산을 하드웨어에서 지원하지 못하여 추가 연산시간이 필요하게 되어 효과적인 실시간 처리에 있어서 부적절한 구조를 가지는 단점이 있다. 본 논문은 실시간 신호처리를 위한 효과적인 IEEE 754의 single precision floating-point 곱셈기의 설계에 관해 기술하였으며, 이의 검증을 위해 Altera FPGA 10K100을 이용하여 구현하였다. 또한 ARM7TDMI의 보조연산기로 사용하기 위해 ARM7TDMI의 버스인터페이스 로직을 추가 설계되었으며, 이의 검증을 위해 MP3 부호화 알고리즘에 적용하였다.

## II. 본론

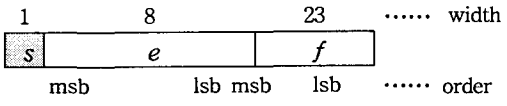
### 2.1 IEEE754-1985 Floating Point 규격

IEEE 754-1985규격은 크게 single precision과 double precision으로 나눌 수 있다. 실제 실수를 표현하기 위해서 sign bit $s \in 0, 1$, $e$는 양의 정수이고, significant $f$는 양의 실수라 하면, 임의의 실수 $N(s, e, f) = (-1)^s \cdot 2^e \cdot f$ 와 같이 표현할 수 있으며, 이에 따른 IEEE754-9185에서 정의한 규격은 아래 도표1과 같다.

표 1 IEEE754-1985 규격 정의

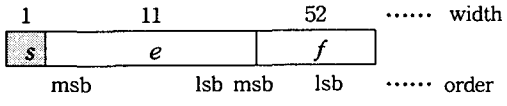
| | Single | | Double | |
|---|---|---|---|---|
| Parameter | Normal | Extend | Normal | Extend |
| Bit number of $f$ | 24 | ≥32 | 53 | ≥64 |
| $E_{max}$ | +127 | ≥+1023 | +1023 | ≥+16383 |
| $E_{min}$ | -126 | ≤-1022 | -1022 | ≤-16382 |
| Exponent bit | +127 | - | +1023 | - |
| Exponent width | 8 | ≥11 | 11 | ≥15 |
| Format width | 32 | ≥43 | 64 | ≥79 |

(1) single precision




- If $e = 255$ & $f \neq 0$, then N is NaN(Not a Numbe -r) regardless of $s$
- If $e = 255$ & $f = 0$, then $N = (-1)^s \cdot \infty$
- If $0 < e < 255$, then $N = (-1)^s \cdot 2^{e-127} \cdot (1 \cdot f)$
- If $e = 0$ and $f \neq 0$, $N = (-1)^s \cdot 2^{e-126} \cdot (0 \cdot f)$ $\simeq$ *denormilized number*
- If $e = 0$ & $f = 0$, $N = (-1)^s \cdot 0 \simeq$ *zero*

(2) double precision




- If $e = 2047$ & $f \neq 0$, then N is NaN(Not a Numb -er) regardless of $s$
- If $e = 2047$ & $f = 0$, then $N = (-1)^s \cdot \infty$
- If $0 < e < 2047$, then $N = (-1)^s \cdot 2^{e-1023} \cdot (1 \cdot f)$
- If $e = 0$ and $f \neq 0$, $N = (-1)^s \cdot 2^{e-1022} \cdot (0 \cdot f)$ $\simeq$ *denormilized number*
- If $e = 0$ & $f = 0$, $N = (-1)^s \cdot 0 \simeq$ *zero*

(3) Round to Nearest

연산된 값을 최대 정밀도에 근접한 값을 갖도록 조정하는 방법을 말하며, 이 경우 인접한 값이 2개가된다면, LSB가 "0"이 되는 값을 선택하도록 규정되어 있으며, $2^{E_{max}}(2 - 2^{-p})$의 경우는 부호의 변화 없이 $\infty$로 전체 값을 조절하게 된다[3].

- Case 1. below half : G=0, (R or S) = 1

```
  1.11110 00001 01001 00001 00 | 1 GRS
+                            0 |
-------------------------------+
  1.11110 00001 01001 00001 00 | 1
```

- Case 2. above half : G=1, (R or S) = 1

```
  1.11110 00001 01001 00001 00 | 1 GRS
+                            1 |
-------------------------------+
  1.11110 00001 01001 00001 01 | 0
```

- Case 3. exactly half with LSB = 1,G=1.R=S=0

```
  1.11110 00001 01001 00001 lsb | GRS
+                           100 |
-------------------------------+
  1.11110 00001 01001 00001 11 | 0
```

- Case 4. exactly half with LSB = 0, G=1, R=S=0

```
  1.11110 00001 01001 00001 01 0 | GRS
+                              0 |
---------------------------------+
  1.11110 00001 01001 00001 01 | 0
```

(4) Directed Round

Rounding의 또 다른 방법으로 Direct Round방식이 있으며, 이 방법은 G,R,S값이 모두 zero일 경우, Rounding 값을 직접 사용자가 선택적으로 $+\infty / -\infty / 0$의 값을 갖도록 하는 방법으로 비교적 간단히 값을 처리할 수 있다는 장점이 있다[3].

- Case 1. Toward 0

```
  1.11110 00001 01001 00001 00 | 1 GRS
+                            0 |
-------------------------------+-----
  1.11110 00001 01001 00001 00 | 1
```

· Case 2. Toward $+\infty$

```
  1.11110 00001 01001 00001 00 | 1 GRS
+                            1 |
-------------------------------+-----
  1.11110 00001 01001 00001 01 | 0
```

· Case 3. Toward $-\infty$

```
  1.11110 00001 01001 00001 00 | 1 GRS
+                            0 |
-------------------------------+-----
  1.11110 00001 01001 00001 00 | 1
```

## 2.2 Floating Point 곱셈기의 구현

(1) 수정 Booth 알고리즘

Booth 알고리즘은 연속된 "1" 또는 "0"의 곱셈 특성을 이용하여 전체 부분 곱의 수를 줄이는 것이다. 우선, 두 이진보수의 곱 Z는 아래와 같다[4].

$$Z = x_{N-1}\, y_{N-1}\, 2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} x_i\, y_j\, 2^{i+j} - 2^{2N-1} + 2^N + \sum_{j=0}^{N-2}\overline{x_{N-1}\, y_j}\, 2^{N-1+j} + \sum_{i=0}^{N-2}\overline{x_i\, y_{N-1}}\, 2^{N-1+i}$$

여기서, X를 다음과 같이 바꾸어 표현하면,

$$X = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i = \sum_{k=0}^{N/2-1}(-2x_{2k+1} + x_{2k} + x_{2k-1})\cdot 2^{2k}$$

와 같이 표현되고, 이를 이용하면 전체 곱 Z는 아래와 같이 표현된다.

$$Z = X\cdot Y = \sum_{k=0}^{N/2-1} Y\cdot Q(k)\cdot 2^{2k}$$

따라서, multiplier의 비트를 이용해 3bit단위로 처리할 수 있게 되는 특징을 가지게 된다. 이렇게 함으로써 전체 부분곱의 수를 N/2로 줄일 수 있는 특징을 가지게 되나, Booth recoder와 Multiple selector에 의한 면적이 소요되는 단점을 가지게 된다. 이러한 수정 Booth 알고리즘을 순서대로 정리하면 다음과 같이 나타낼 수 있다.

i) 승수 X의 LSB 오른쪽에 '0'을 추가 ( $x_{-1} = 0$)

ii) 승수 X의 비트 수가 홀수이면, MSB를 1-bit 확장

iii) $x_{-1}$에서부터 3-bit씩 Booth recoding

iv) 부분곱 생성 및 가산

(2) Multi-bit Recoding 알고리즘(MRA)

N-bit의 승수 X에 대해 다음과 같이 표현할 수 있다.

$$X = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i = \sum_{k=0}^{N/(M-1)-1} Q(k)\cdot 2^{(M-1)k}$$

$$Q(k) = x_{(M-1)k-1} + \left(\sum_{j=1}^{M-2} x_{(M-1)k+j-1}\cdot 2^{j-1}\right) - x_{(M-1)(k+1)-1}\cdot 2^{M-2}$$

여기서, 피승수 Y에 관한 곱 Z를 위에 기술한 X에 대해 표현하게 되면,

$$Z = X\cdot Y = \sum_{k=0}^{N/(M-1)-1} Y\cdot Q(k)\cdot 2^{(M-1)k}$$

$$Y\cdot Q(k) \in \{0, \pm Y, \pm 2Y, \pm 3Y, \cdots, \pm 2^{M-2}Y\}$$

위와 같이 나타낼 수 있으므로, M-bit형태의 부호화가 가능하게 되고 이에 따라 부분곱의 수가 N/(M-1)로 감소하는 효과를 가질 수 있게 된다. 여기서, 4bit의 경우에 대한 MRA(M=4)를 예를 들면 다음과 같이 표현되어진다.

$$X = \sum_{k=0}^{N/3-1}(-4x_{3k+2} + 2x_{3k+1} + x_{3k} + x_{3k-1})\cdot 2^{3k}$$

$$Z = X\cdot Y = \sum_{k=0}^{N/3-1} Y\cdot Q(k)\cdot 2^{3k}$$

하지만, +3X, -3X와 같은 multiple 생성을 위해 별도의 가산기와 감산기가 필요하게 되는 문제점이 있다. 본 연구에서는 MRA에 의해 전체 부분곱의 수를 줄임과 동시에 별도의 가산기와 감산기에 의한 연산량은 부분곱의 재배치와 효과적인 최종 가산단을 사용함으로써 고속의 연산이 가능한 floating-point 곱셈기를 설계하였다.

(3) 부분곱 연산기 및 전체 곱셈기 구조

곱셈기의 부분곱을 계산하는 과정은 Carry Save Adder Tree를 사용하여 Carry의 전파를 축소하였으며, CLA(Carry Look Ahead)구조를 CSA(Carry Select Adder)와 함께 적용하여 최적화시킴으로서 고속 곱셈기를 구현하였다. 그림 1은 Floatng Point 곱셈기의 블록 다이어그램이다. 24-bit Unsigned Multiplier와 Exponent Unit, 그리고 Sign Unit과 후처리기로 이루어져 있다. Sign Unit의 경우 XOR 게이트 하나로 구현되며, Exponent Unit는 8-bit 덧셈기로, 후처리기는 Mux set으로 구성되어 있다.

위 구조에서 보면 Booth Encoder와 Booth Decoder에서 부분 곱을 생성시키면, 그 부분 곱을 CSA Adder

Tree에서 부분 곱의 수를 2개로 줄이게 된다. 일반적으로 CSA Adder Tree는 역 피라미드 형태로 구성되어 있기 때문에 CSA의 단수를 줄일 수 있다[2]. 일반적인 CSA를 쓰게 되면 12개의 CSA와 1개의 CLA를 필요로 하게 된다. Tree 구조로 구현할 경우 사용되는 CSA의 수는 같지만 총 거치게 되는 CSA의 단은 $\log_{\frac{3}{2}} 13 \fallingdotseq 6.33$ 이 되므로 6개의 CSA단과 1개의 CLA 단만 거치면 되므로 속도를 보다 높일 수 있게 된다. 이러한 CSA의 구현은 부분곱의 재배치 알고리즘을 써 보다 효율적으로 연산 가능하도록 하였다. 부분곱의 수가 증가하고 들어오는 입력 패턴에 따라 지연시간이 달라짐과 동시에 전체 회로의 라우팅이 매우 복잡하다는 문제점이 발생하게 된다. 따라서, 하드웨어의 복잡성을 보다 줄이기 위해 반가산기와 전가산기를 이용하여 부분곱을 재배치함으로써 연산의 효율성을 증가시킬 수 있다. 반가산기는 그 동작 특성상 (2,2) 카운터로 표현 할 수 있다. 본 연구에서는 위 그림 2와 같이 (3,2) 카운터를 기반으로 한 재배치 기법에 의한 트리구조를 이용해 전체 하드웨어의 복잡도를 줄임과 동시에 보다 빠르게 연산을 수행할 수 있다.

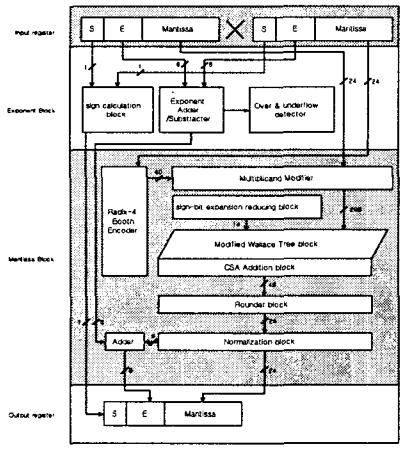



그림 1 제안된 floating-point 연산기

고속 연산을 위하여 CLA(Carry Look Ahead)구조를 Carry Select Adder와 함께 적용하였다. CLA는 bit수가 커질수록 Fan out이 커지고 차지하는 면적이 커지게 되므로 크게 만들기 어렵다. 따라서, 본 논문에서는 이러한 단점을 보완하기 위해 Carry Select Adder를 채용하여 Critical Path를 줄였다. 즉, 전단의 Carry를 기다리는 것이 아니라 전단의 Carry가 전달되기 전에 미리 결과를 계산해 놓았다가, 전단의 Carry가 전달되는 즉시 Mux를 통해서 결과를 내보내는 구조로 되어 있다. 전단의 Carry는 '0'과 '1' 두 가지 경우이므로 두가지 경우의 계산을 먼저 해놓고 전단의 Carry가 넘어오기를 기다리는 것이다. 일반적인 ripple carry adder가 48개의 FA에 비해 하드웨어는 2배 필요하지만, Critical Path는 14FA로 70%정도 향상된다.

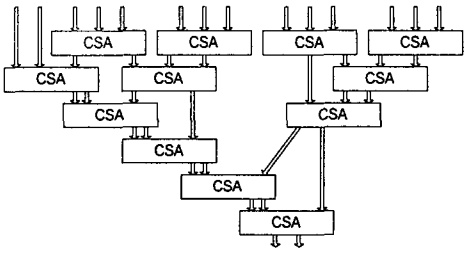



그림 2 제안된 재배치에 의한 Tree 구조

## III. 결론

본 논문에서는 IEEE-754규격의 Floating-point 연산기를 설계하고 이의 동작을 ARM7TDMI를 이용하여 검증하였다. 또한 내부 연산기의 효과적인 구조를 위해 MRA 및 부분곱 재배치를 통한 가산단을 설계함으로써 보다 고속의 연산이 가능하도록 하였다.

ARM7TDMI의 내부 연산기는 Booth 알고리즘을 사용하였으나 fixed point 연산을 수행하므로, 곱셈을 위해서는 최대 4cycle이상이 소요되게 된다, 따라서, 본 연구에서 제안한 보조연산기의 경우 1cycle에 floating-point연산을 수행하므로 보다 많은 연산을 수행할 수 있게 된다. 따라서, 본 연구에서 제안한 연산기는 MCU 및 DSP등의 성능 향상 및 다양한 신호처리부분에 사용되어 질 것으로 판단된다.

## 참고문헌

[1] Alan Y. Kwentus, Hing-Tsun Hun, and Alan N. Willson, Jr,, "An Architecture for High-Performance/Small-Area Multipliers for use in Digital Filtering Applications," IEEE joural of solid state Circuits, Vol. 29, No 2, Feburary 1994., pp. 117-121.

[2] C. S. Wallace, "A Suggestion for a fast Multiplier," IEEE Transactions on Electronic Computers, Feburary 1964, pp. 14-17.

[3] 이용석, "고성능 마이크로프로세서 부동소수점 연산기 구조," March. 1999.

[4] A. D. Booth, "A Signed binary multiplication technique," Quart. J. Math., Vol. IV, 1951, pp 2.