1999년도 대한전기학회 정기총회 및 추계학술대회 논문집 1999. 11. 20

IEEE 754 단정도 부동 소수점 연산용 곱셈기 설계

이 주 훈* ,정 태 상**
*중앙대 공대 제어계측공학과 , **중앙대 공대 전자전기공학부

Design of a Floating Point Multiplier for IEEE 754 Single-Precision Operations

Ju-Hun Lee* , Tae-Sang Chung**
*Dept. of Control and Instrumentation Eng.**School. of Electrical and Electronics Eng. Chung-Ang Unv.

Abstract - Arithmetic unit speed depends strongly on the algorithms employed to realize the basic arithmetic operations.(add, subtract, multiply, and divide) and on the logic design. Recent advances in VLSI have increased the feasibility of hardware implementation of float -ing point arithmetic units and microprocessors require a powerful floating-point processing unit as a standard option. This paper describes the design of floating-point multiplier for IEEE 754-1985 Single-Precision operation. Booth encoding algorithm method to reduce partial products and a Wallace tree of 4-2 CSA is adopted in fraction multiplication part to generate the 32×32 single-precision product. New scheme of rounding and sticky-bit generation is adopted to reduce area and timing. Also there is a true sign generator in this design. This multiplier have been implemented in a ALTERA FLEX EPF10K70RC240-4.

1. 서 론

컴퓨터에서 사용되고 있는 가장 대표적인 연산에는 소수점이 고정되어 있는 고정소수점 연산(Fixed Point Arithmetic) 과 소수점이 고정되어 있지 않은 부동 소수점 연산(Floating Point Arithmetic)이 있다. 고정소수점 연산의 경우 데이터를 표현하는 방법과 처리하는 방법이 간단하기 때문에 범용 마이크로프로세서에 쉽게 수행될 수 있지만, 부동 소수점 연산은 고정 소수점 연산을 하기 위한 부분과는 별도의 하드웨어가 필요하게 되었다. 본 논문의 주요 관점이 바로 이 별도의 하드웨어를 수행 해 보는 것이다. 부동 소수점 연산기에는 ALU, 곱셈기, 제산기등이 포함되어 있다. 본 논문에서는 많은 부동 소수점을 표현하기 위한 규약 중에 IEEE 754-1985 형식을 채택한 부동 소수점 곱셈기를 수행하는 것이다.

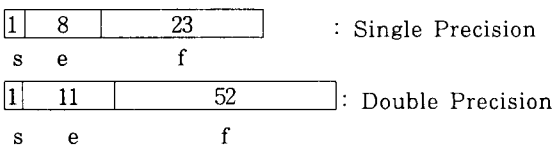


그림 1. IEEE standard single and double precision floating point.

2. 본 론

2.1 IEEE Standard 754-1985 단정도

IEEE Computer Society는 부동 소수점 표현법과 연산에 대한 제안된 표준을 발전시켰다. 1985년에 IEEE의 위원회에서 정한 이 표준은 그 후 설계된 대표적인 부동 소수점 연산기인 Intel사의 80387,Motorola사의 68881등 거의 모든 부동 소수점 연산기에 사용되고 있으며 이 표준에서 단정도(single precision) 또는 배정도(Double precision) 숫자는 그림 1에 보인 바와 같이 다음의 3가지로 구성되어있다.

$$\begin{cases} (1)1\text{bit 부호} & s \\ (2)\text{Bias된 지수(e)} & e = E + \text{Bias} \\ (3)\text{가수(Mantissa)} & f = .b_1 b_2 \cdots b_{p-1} \end{cases}$$

부동소수점 숫자는(ν) 다음과 같이 표시된다.

$$\nu = S_a r^{E_a} = (-1)^s \cdot (1.f) \cdot 2^E$$

넓게 수행되어지고 있는 IEEE standard 754 단정도 부동 소수점 수의 구성은 1부터 254의 범위를 가지는 8-bit bias된 정수 지수를 가진다. 지수는 그것의 실제 값이 저장된 값에서부터 127을 뺀 것으로 결정되어지는 excess-127 code로써 표현이 되어진다.
그러므로, 지수의 실질적인 값들의 범위는 각각 1에서 254까지의 저장된 값에 상응하는 -126에서 127까지이다. Bias의 값은 단정도 부동 소수점의 경우 127이다.

$$\nu = (-1)^s \cdot 2^{e-127} \cdot (1.f)$$

$$0 \le e \le 255,\ 2^{-127} \le |\nu| < 2^{+129}$$

표 1은 다양한 bit 패턴들에 할당되어진 값들을 나타낸 것이다.

2.2 곱셈기 구현

2.2.1 부동 소수점 곱셈의 알고리즘

부동 소수점 곱셈과 나눗셈은 덧셈과 뺄셈보다 훨씬 더 간단한 과정들이다. 그림 2은 부동 소수점 곱셈에서의 순서도 이다.

Single Precision (32 bits)

Exponent, e	Significand, f	Value
255	≠ 0	NaN
255	0	$(-1)^s\infty$
$0 < e < 255$	—	$(-1)^s 2^{e-127}(1.f)$
0	≠ 0	$(-1)^s 2^{-126}(0.f)$
0	0	$(-1)^s 0$

표 1, Values of IEEE Floating-Point Numbers

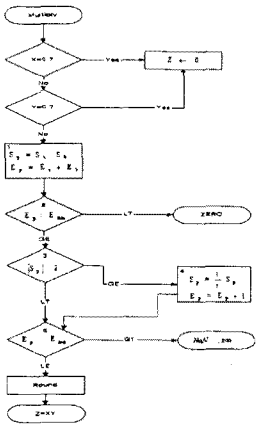


그림 2. 부동 소수점 곱셈의 순서도

2.2.2 하드웨어적인 수행

곱셈기는 크게 가수(Mantissa), 지수(Exponent), 부호(Sign)의 3 부분으로 구성이 되어서 동작을 한다. 첫째 가수부분은 다시 3단계로 구성이 되어있다. 먼저 첫째 단계에서는, 승수와 피승수 연산자들이 radix-4 encoding(Booth 2)과 CSA(carry save adder)을 이용한 곱셈 tree(wallace tree)를 거치게 된다. 부분 곱들의 중간에서의 합들과 tree의 결과는 carry-save 형식으로 나타난다. 두 번째 단계에서는 Conditional sum 방법의 덧셈기를 이용하여 결과를 carry-save 형식에서 binary 형식으로 바꾸면서, 곱의 결과를 결정하고, 반올림에 사용되는 bit들을 생성하는데 사용되었다. 세 번째 단계에서는 반올림(rounding)과 overflow 신호 생성을 수행한다. IEEE 754 표준에서의 가수부분의 bit들의 수는 23bit(mantissa bit들)+1bit(hidden bit)으로 구성이 되어 있고, Booth 2 방법에 의해 생성되는 부분 곱들의 수는 13개이다. 다음 그림 3에서 각 열(맨 마지막 열을 제외한)의 dot의 수는 encoding한 값이 피승수(Multiplicand)의 두 배일 때 왼쪽으로 이동하기 위하여 원래 dot의 수 보다 하나 더 추가된 형태이다.[1]

2.2.3 가수(Mantissa)들의 곱

2.2.3.1 I 단계 : Multiplication Tree

그림 4에서 보면 처음에 두 가수(Mantissa)의 입력을 받아서, booth2_total_mul의 블록에서 Booth 2 방법을 사용하고, CLA(carry look ahead) 28bit 덧셈기를 이용하여 부호확장과 상수(1)을(그림 3) 가지고 있는 부분 곱들을 생성한다. 생성된 각각의 부분 곱들은 Wallace Tree(Fraction 블록)를 통해 carry-save 형식을 가지는 결과(Carry와 sum)를 생성한다.

2.2.3.2 II단계 : 마지막 덧셈과 L,G,R,S bit들

2.2.3.2.1 Half Adder

그림 5에서의 C21신호와 23이나 24에서 일어나는 반올림 때문에 25의 위치에 두 개의 carry들이 들어갈 가능성이 있다. 하나의 carry만이 25bit에 전달된다는 것을 보증하기 위해서, 반가산기(half adder)들의 열이 C[46..21]과 S[47..22]에 사용이 되었다.

2.2.3.2.2 Conditional-Sum Adder

곱셈기의 마지막 더하기 단계는 단계 I 에서 구해진 Carry Save 형식의 결과를 입력으로 받아서, 23-bit

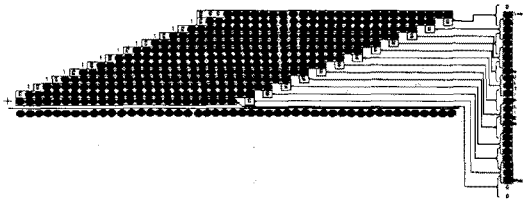


그림 3. IEEE 754 부동소수점 가수부분의 부분 곱

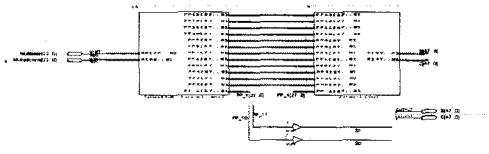


그림 4. Block Diagram of Stage 1

최소 지연을 위해 분배되어진 Conditional sum 덧셈기를 이용한다. 합과 합+1 둘 다의 결과를 만들 수 있는 conditional sum 덧셈기를 사용하는 것은, 반올림의 연산을 수행하는데 값을 증가시키기 위해 필요한 덧셈기의 필요를 없앴다. 단지 반올림 후에 맞는 결과를 선택하는 선택작업만이 필요하다.

2.2.3.2.3 Carry-Save 형식의 결과에서 Sticky -Bit 생성

정확한 반올림을 수행하기 위해 필요한 Sticky bit은 전형적인 방법으로는 carry-save 형식인 낮은 차수의 22bit들의 합을 취하고, 모두를 OR를 함으로써 구할 수가 있었다. 그러나, 본 논문에서는 합을 구하기 위해 22 bit 덧셈기를 사용하던 방법과는 달리 타이밍과 면적에서 굉장히 장점이 되는 carry-save 형식의 tree에서의 결과들에서부터 직접 sticky bit을 구하는 방법을 사용하였다. 다음과 같이 정의를 한다.

$$p_i = s_i \oplus c_i \qquad \text{(식 1)}$$

$$h_i = s_i + c_i \qquad \text{(식 2)}$$

s_i와 c_i는 carry save 형식의 결과를 낳는 tree에서의 합과 carry 결과들이다.

$$t_i = p_i \oplus h_{i-1} = s_i \oplus c_i \oplus (s_{i-1} + c_{i-1}) \qquad \text{(식 3)}$$

$$s_{-1} = 0 \text{ and } c_{-1} = 0$$

그런 다음, sticky bit은 다음과 같은 방법을 사용하여 직접 구해질 수 있다.

$$\text{sticky} = t_0 + t_1 + \cdots + t_{21} \qquad \text{(식 4)}$$

2.2.3.3 III단계 : Rounding과 옳은 결과 선택을 위한 Logic

단정도 부동소수점 곱셈에서는 carry-save 형식의 48-bit 곱(그림 3)이 booth 2 알고리즘에 의해 만들어진 후에, 소수점은 46과 45 bit들의 사이에 생기게 되고, 상위 24 bit들만이 단지 가수 결과로써 사용되어진다. 하위 24bit들은 옳은 반올림을 수행하는데 만 필요하다. 반올림이 수행된 후에, MSB가 1이면, 그때 가수는 47~24bit을 취하고, 지수의 값은 증가시켜준다. 그렇지 않고, 만약 MSB가 0면, 그때는 가수로 46~23bit들을 취하고, 지수의 값은 증가하지 않는다. 아래

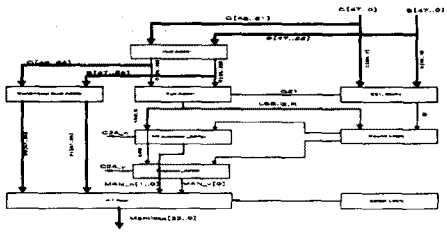
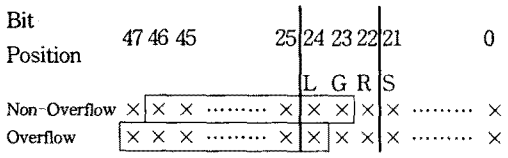


그림 5. Final Addition and Rounding

OF	C24_n	C24_v	Select
0	0	×	F0[46..25],MAN_n[1..0]
0	1	×	F1[46..25],MAN_n[1..0]
1	×	0	F0[47..25],MAN_v[0]
1	×	1	F1[47..25],MAN_V[0]

표 2. Select Logic

그림은 가수가 반올림 후에 47bit에 따라서 선택되어지는 것을 보여주고 있다. L,G,R,S는 각각 LSB와 guard와 round와 sticky bit들을 의미한다. 반올림을 수행한 후에 발생하는 overflow 경우를 올바로 취급하기 위해서, 마지막 선택 로직에서 필요한 C24_n과 C24_v의 신호를 만들기 위한 두 개의 덧셈기 (NoOverflow_Adder 와 Overflow_Adder)를 추가로 더 수행하고 있다.



C24_n과 C24_v는 각각 overflow가 발생하지 않았거나 발생했다고 가정했을 때 24bit에서 발생하여 25 bit 위치로 들어가는 carry들이다. L, G, R, S bit들은 두 개의 반올림 값들을 만들기 위한 반올림 로직(round logic)에서 사용되어 진다. 하나의 값은 가수가 overflow라고, 다른 하나는 가수가 overflow가 아니라고 가정한다. 이러한 반올림 bit들은 overflow (MAN_v)와 non_overflow (MAN_n)에 대한 결과적인 가수의 가장 차수가 낮은 한 bit 또는 두 bit들을 각각 구성하기 위해 L 과 {L, G} bit들에 더해진다.

2.2.3.3.1 마지막 단계에서의 옳은 결과 선택

마지막에 있는 선택 로직(4×1 MUX)은 최종적인 가수를 형성하기 위해 conditional sum 덧셈기로부터의 결과인 F0 와 F1를 MAN_v나 MAN_n로 서로 결합한다. 표 2은 선택 로직의 진리표이다. 이 표는 식 5에서 보여지는 Overflow 신호에 대한 표현이다.

$$\text{Overflow} = F0[47] + (C24\_n \cdot F1[47]) \quad \text{(식 5)}$$

마지막으로 반올림의 결과에 따라 최종적인 가수 부분의 옳은 곱의 결과를 구하기 위하여 위의 표 2 선택 로직에 따라서 4×1 MUX를 구성하면 총 세 부분 중에 첫 번째 가수 부분의 하드웨어적인 수행은 완료가 된다.

2.2.4 지수 부분의 수행

곱셈의 연산에서 지수들간의 연산은 덧셈의 과정이다. 입력 연산자들의 8bit bias된 지수들을 각각 e_1, e_2 라

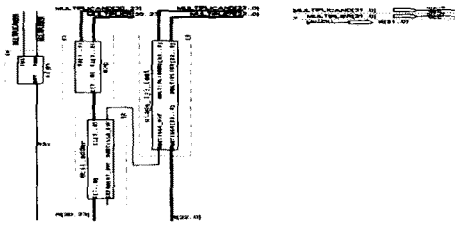


그림 6. IEEE 754 단정도 부동소수점 곱셈기의 전체 블록도

고하자. 그러면

$$\begin{cases} e_1 = E_1 + \text{Bias} \\ e_2 = E_2 + \text{Bias} \end{cases}$$ 여기서 Bias는 127이다.

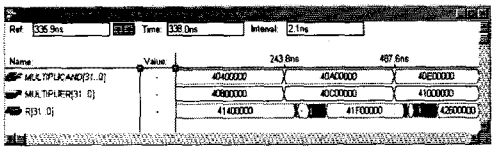
따라서, bias를 제외한 실제 지수인 E_1, E_2에 대한 곱셈 결과의 실제 지수는 E는 $E = E_1 + E_2$이고, 결과적인 지수는 $e = E + 127 = E_1 + E_2 + 127$이다. 따라서, 입력 연산자의 지수에서 바로 결과 지수를 얻으려면 $e = e_1 + e_2 - 127$ (식 4.6)
과 같은 연산으로 바로 결과적인 지수를 구할 수 있다.

2.2.5 Sign bit 부분의 수행

IEEE 단정도 부동소수점에서의 sign bit가 1이면 음수를 0면 양수를 표현한다. 즉, Sign bit의 하드웨어적인 구현은 XOR로써 구현이 가능하다.

3. 결 론

3.1 기능 검증과 Timing 분석



처음 입력 데이터인 3(40400000)과 4(40800000)의 곱의 결과 12(41400000)를 가지고 있다가 두 번째 입력 데이터인 5(40A00000)과 6(40C00000)가 들어오고 92.1ns 지연 후에 두 수의 안정된 곱의 결과 30(41F00000)가 나타난다. 세 번째 입력 데이터 7(40E00000)과 8(41000000)들의 곱의 결과 56(42600000)는 95.2ns 지연 후에 두 수의 안정된 곱의 결과가 나타난다. 고속의 CMOS 기술 반도체 기반의 하드웨어적인 수행이 아닌 속도 면에서 약세를 나타내고 있는 CPLD(또는 FPGA)에서의 하드웨어적인 수행 이였지만, MAX_PATH는 243.8ns인 반면에 MIN_PATH은 71.7ns로써 거의 3사이클 이상의 차이가 난다. 따라서 4.1MHz 클럭사이클 내에서만 동작을 보장하는 것이 아니라 그 보다 더 높은 클럭사이클(10MHz 정도)에서도 동작은 보장된다.

〔참 고 문 헌〕

〔1〕 Gray W.Bewick, "Fast Multiplication:Algorithms and Implementation", PhD thesis, Stanford Uni.1994
〔2〕 Israel Koren, "Computer Arithmetic Algorithms", John Wiley &Sons. Inc, 1993