論文2003-40SD-9-12

작은 룩업테이블을 가지는 새로운 파이프라인 나눗셈기

(A New Pipelined Divider with a Small Lookup Table)

鄭 雄 *, 朴 祐 贊 **, 郭 承 浩 *, 梁 薰 模 *, 鄭 喆 浩 **,
韓 鐸 敦 **, 李 文 基 *

(Woong Jeong, Woo-Chan Park, Sung-Ho Kwak, Hoon-Mo Yang,
Cheol-Ho Jeong, Tack-Don Han, and Moon-Key Lee)

요 약

기존의 나눗셈 연산기들은 대부분 반복적인 방식으로 연산을 수행하여 왔으나, 최근에는 파이프라인드 나눗셈 연산기에 대한 연구가 시도되고 있다. 현재 발표된 파이프라인 나눗셈 연산기는 큰 사이즈의 룩업테이블을 필요로 하기 때문에 면적을 크게 차지한다는 단점이 있다. 본 논문에서는 기존의 파이프라인드 나눗셈 연산기에 비해 룩업테이블을 크게 줄여, 비용에 효과적인 파이프라인 나눗셈 연산기를 제안한다. 제안하는 나눗셈 연산기는 단정밀도에서 3 사이클의 지연시간를 가지며, P. Hung의 방식에 비하여 약 30퍼센트 정도의 면적을 줄일 수 있다.

Abstract

Generally, dividers have been designed to use iteration, but recently the research on the pipelined divider is underway. It is a difficult point in the known pipelined division unit that a large lookup table is required. In this paper, the cost-effective pipelined divider is proposed, that needs a lookup table smaller than that of the other pipelined divider. The latency of the proposed divider is 3 cycles. We obtain a 30% reduced area than that of P. Hung.

Keyword : computer arithmetic, division, floating point

Ⅰ. 서 론

나눗셈 연산은 다른 산술 연산에 비하여 그 빈도수가 낮다는 특성을 가지고 있다. 이런 특성 때문에 나눗셈 연산기는 작은 면적을 차지하도록 설계되어 왔고, 반복적인 방식으로 대부분 구현되어 왔다. 이에 따라 다른 연산에 비해 나눗셈 연산을 수행할 때의 지연시간은 상대적으로 긴 편이다. 하지만, 낮은 빈도수에도 불구하고, 긴 지연시간이 전체 시스템에 끼치는 영향은 작지 않다[1].

최근 들어 VLSI의 집적도가 높아지고 고성능을 필요로 하는 응용분야가 커지면서, 이터레이션 없이 나눗셈을 수행하는 구조가 제안되고 있다. 특히, 3차원 그래픽 처리가 프로세서의 주요한 어플리케이션으로 부각되었으며, 3차원 그래픽을 고속으로 처리하기 위하여 높은 처리율을 가진 파이프라인으로 구성된 나눗셈 연산기가 필요하게 되었다[2, 3].

P. Hung은 테일러 급수 전개 방식을 수정하여, 파이프라인드 나눗셈 연산기를 제안하였다[2]. 이 방식은 제수의 역수의 테일러 급수 2차항까지를 하나의 룩업테

* 正會員, 延世大學校 電氣電子工學科
(Dept. Electrical and Electronics, Yonsei University)
** 正會員, 延世大學校 컴퓨터科學科
(Dept. Computer Science, Yonsei University)
接受日字:2002年3月21日, 수정완료일:2003年9月4日

이블에 저장하며 룩업테이블 참조와 2번의 곱셈 연산만으로 나눗셈 연산을 수행할 수 있다. 이로 인하여 나눗셈 연산기를 파이프라인 형태로 구성할 수 있다. 기존 방식보다 룩업테이블의 크기를 다소 줄였다는 장점을 가지고 있다. 그러나 이 방식은 여전히 큰 사이즈의 룩업테이블을 필요로 하기 때문에 면적을 크게 차지한다는 단점이 있다. 또한 정밀도가 높아질수록 그 문제는 더 심각하다.

본 논문에서는 P. Hung이 제안한 파이프라인드 나눗셈 연산기의 알고리즘을 수정하여, 지연시간을 1 사이클 증가시키나, 룩업테이블 크기를 크게 줄일 수 있는 방법을 제안한다. 제안하는 알고리즘은 단정밀도에서 3 사이클의 지연시간을 가지며 전체 크기를 P. Hung 방식에 비해 30퍼센트 정도 줄일 수 있다.

본 논문의 구성은 다음과 같이 이루어져 있다. Ⅱ장에는 지금까지 진행되어 왔던 관련된 작업들에 대해 쓰여 있다. Ⅲ장에는 제안하는 알고리즘과 아키텍쳐에 대하여 설명한다. Ⅳ장은 에러 분석을 통하여 최대 에러를 계산하고 나눗셈기 내부 유닛들의 비트폭을 결정한다. Ⅴ장은 다른 나눗셈 알고리즘과 비교를 한다. 마지막으로 Ⅵ장은 결론으로 구성되어 있다.

Ⅱ. Related Works

나눗셈 연산기는 크게 SRT 등의 단위 회귀 방식과 multiplicative 방식으로 나눌 수 있다[4]. 이중 multiplicative 방식은 approximation 방식으로써 곱셈기와 룩업테이블을 이용한다. 곱셈기를 이용하는 방법 중에 잘 알려진 알고리즘으로는 뉴튼랩슨과 급수전개법이 있는데, 둘 다 룩업테이블에서 가져온 역수를 곱셈기를 통하여 연산하여 몫을 계산하는 방법이다.

이들 방법들의 단점은 고속처리를 위해서 큰 사이즈의 룩업테이블을 필요로 한다는 것이다. 예를 들면, 16비트 시드를 가지는 뉴튼-랩슨 방식이나 급수 전개 방식의 나눗셈 연산기는 총 64KB의 룩업테이블 크기를 필요로 한다[5]. 이 방식들은 2번의 이터레이션을 통하여 단정밀도의 나눗셈 연산을 수행할 수 있다. 8비트의 시드를 사용하는 경우에는 룩업테이블의 크기는 128B로 작지만, 3번의 이터레이션을 수행해야 되기 때문에 지연시간이 상당히 늘어난다는 단점을 지니게 된다. 테일러 급수를 이용한 accurate quotient approximation 방식은 여러 개의 룩업테이블을 이용하는 방식인데, 한 번의 iteration에 단정도의 연산을 수행하려면 19.5KB의 룩업테이블을 필요로 한다[6].

최근에 제안된 파이프라인으로 구성된 나눗셈기에는 A. Liddicoat이 제안한 구조와 P. Hung이 제안한 구조가 있다[2, 3]. A. Liddicoat의 나눗셈 연산기는 뉴튼-랩슨 알고리즘의 3차항까지를 계산하는데, 병렬성을 이용하여 지연시간을 줄이는 방식이다. P. Hung의 나눗셈 연산기는 테일러 급수 전개를 이용한 방식인데, 구조가 간단하고 룩업테이블의 크기가 기존의 방식보다 작다는 장점을 가지고 있다.

P. Hung이 제안한 알고리즘을 보면 테일러 급수 전개에 의하여 나눗셈 연산을 다음의 식으로 정리할 수 있다.

$$\frac{X}{Y}=\frac{X}{Y_h+Y_l}=\frac{X}{Y_h}(1-\frac{Y_l}{Y_h}+(\frac{Y_l}{Y_h})^2-\cdots) \quad (1)$$

여기서 Y_h는 Y의 상위 p 비트까지의 값이고, Y_l은 Y에서 Y_h를 뺀 값이다. X와 Y는 정규화되어진 고정소수점수이고, 따라서 각 값들은 식 (2)와 같은 경계조건을 갖는다.

$$\begin{aligned} &1\le X,Y<2\\ &1\le Y_h<2-2^{-p+1}\\ &0\le Y_l<2^{-p+1}\end{aligned} \quad (2)$$

식 (1)에서테일러 급수 2차항까지만으로 근사화하면, 식 (3)와 같이 된다.

$$\frac{X}{Y}\approx\frac{X(Y_h-Y_l)}{Y_h^2} \quad (3)$$

식 (3)에 따라 나눗셈 연산은 피제수 X와 제수에서 얻어낸 Y_h-Y_l을 곱하고 룩업테이블에서 불러온 $1/Y_h^2$을 곱함으로써 수행될 수 있다. <그림 1>은 P. Hung의 나눗셈 연산기가 수행되는 과정을 블록다이어그램으로 나타낸 것이다. 여기서 Y_h-Y_l는 실제 뺄셈을 수행할 필요 없이 X와 Y_h-Y_l을 곱하는 데 사용되는 곱셈기의 부쓰 인코딩을 수정하는 것만으로 가능하다[7].

P. Hung이 제안한 알고리즘은 하나의 룩업테이블을 이용하여 $1/Y_h^2$을 근사화한다. 이를 통하여 기존의 방식보다 룩업테이블의 크기를 다소 줄일 수 있다. 그럼에도 불구하고, P. Hung의 방식에 의한 나눗셈 연산기 역시 기존의 반복적인 알고리즘으로 구현된 나눗셈 연

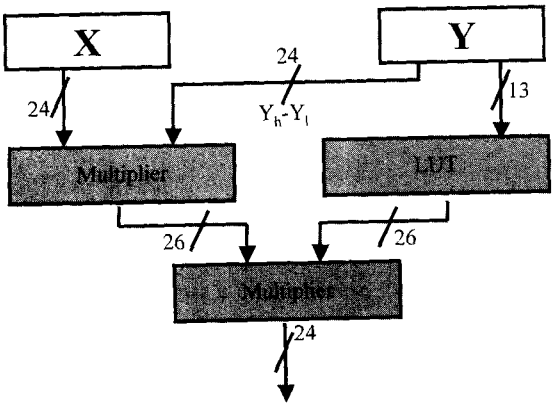


그림 1. P. Hung의 나눗셈 연산 수행 과정 (단정도)
Fig. 1. Block Diagram of P.Hung's Division Algorithm in Single Precision.

산기에 비해 면적이 많이 크다는 단점을 지니게 된다. 단정도의 경우에는 13KB 정도의 룩업테이블 면적을 차지하며, 쌍정도인 경우 사실상 구현이 불가능하다. 따라서 파이프라인 형태의 나눗셈 연산기에서 룩업테이블의 크기를 줄이는 것은 매우 중요한 이슈가 된다.

Ⅲ. Proposed Divider Architecture

본 논문의 중요한 목표는 비반복적인 나눗셈 연산기의 가장 큰 단점이 되는 룩업테이블의 크기를 줄이는 것이다. 훨씬 더 작은 룩업테이블을 가지고 P. Hung의 알고리즘과 동일한 연산을 수행하면, P. Hung의 알고리즘에 비해coarse quotient가 계산된다. 이 coarse quotient와 제수를 곱한 것을 피제수에서 빼면 나머지 값을 계산해 낼 수 있다. 이 나머지값을 가지고 P. Hung의 알고리즘과 동일한 연산을 한번 더 수행하고, 이 두번의 연산에 의하여 계산된 몫을 더하면 최종적인 해를 계산해낼 수 있다. 이 과정은 복잡해보이지만, 수식을 다시 정리하여 redundant한 연산을 없애면, P. Hung의 나눗셈 연산기보다 수십분의 일 크기의 룩업테이블과 P. Hung에 비해 precison이 작은 4개의 곱셈기만으로 연산이 가능하다.

1. Proposed Algorithm

제안되는 알고리즘은 다음과 같다. (3)번 식에서 Y_h의 비트폭을 줄이면, coarse quotient $\tilde{Q}$는 다음과 같이 정의할 수 있다.

$$\tilde{Q} \approx \frac{X(Y_h - Y_l)}{Y_h^2} \tag{4}$$

이 coarse quotient에 의해서 아래 식과 같이 계산을 하면, 나머지값을 계산할 수 있다.

$$\tilde{X} \approx X - Y \cdot \tilde{Q}$$

이 나머지 값을 다시 한번 식 (4)과 동일한 계산을 하고, 두개의 해를 더하면 최종적인 quotient가 구해진다.

$$\tilde{\tilde{Q}} \approx \frac{\tilde{X}(Y_h - Y_l)}{Y_h^2}$$

$$\frac{X}{Y} \approx \tilde{Q} + \tilde{\tilde{Q}}$$

위 식은 다음과 같이 정리될 수 있다.

$$\begin{aligned} \frac{X}{Y} &\approx \tilde{Q} + \tilde{\tilde{Q}} \\ &= \frac{(X + \tilde{X})(Y_h - Y_l)}{Y_h^2} \\ &= (X + \tilde{X})A \\ &= (2X - Y\tilde{Q})A \\ &= (2X - AYX)A \\ &= (2 - AY)AX \end{aligned} \tag{5}$$

식 (5)에서 A 값은 다음과 같다.

$$A = \frac{(Y_h - Y_l)}{Y_h^2}$$

2. Proposed Hardware Architecture

제안하는 알고리즘을 수행하는 하드웨어 블록다이어그램은 <그림 2>와 같다. 식 (5)을 계산하는 과정은 네 단계로 나누어진다. 첫번째 단계에서는 룩업테이블에서 $1/Y_h^2$를 계산하고, 두번째 단계에서는 $1/Y_h^2$와 Y_h-Y_l를 곱하여 A를 계산한다. 세번째 단계에서는 AX와 AY를 계산하는데, 두 곱셈은 병렬로 수행한다. 그리고 네번째 단계에서 최종적인 수식을 얻어내게 되는데, 여기서 (2-AY)의 계산은 AY를bit-inversion하여 얻어낼 수 있다. 예를 들면 1.00101을 bit-inversion하면 0.11010이 되는데, 이는 2에서 1.00101을 뺀 값인 0.11011보다 1 ulp가 작은 값이다. Bit-inversion을 사용하는 경우 실

제 뺄셈을 수행하는 것에 비해 덧셈기를 없앨 수 있지만 에러값이 더해진다. 이 에러값이 어떠한 영향을 미치는지는 4장에서 분석하고 있다.

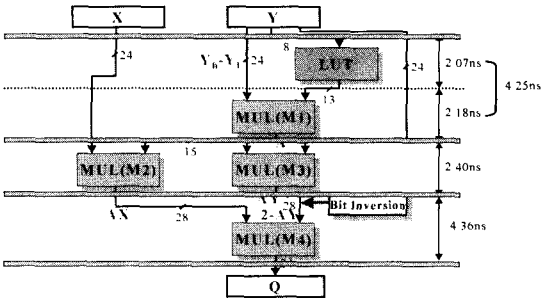


그림 2. 하드웨어 아키텍쳐 블록 다이어그램 (단정도)
Fig. 2. Block Diagram of the proposed Algorithm in Single Precision.

제안하는 알고리즘을 수행하는 데에는 총 하나의 룩업테이블과 4개의 곱셈기를 요구한다. 여기서 두개의 곱셈은 평행하게 수행될 수 있기 때문에, 레이턴시는 1 LUT + 3 MUL을 갖게 된다. 4개의 곱셈기 중에서 3개의 곱셈기는 [2]에 비해서 승수의 비트폭이 작아서 면적과 지연시간을 상당히 줄이게 된다. 단정밀도인 경우에는 최종단의 곱셈기는 28×28이지만, 나머지 세 곱셈기는 24×15 이거나 24×13이다. 또한 룩업테이블의 크기가 매우 작아서, 아주 짧은 시간에 룩업테이블의 값을 가져올 수 있다. 따라서, 단정밀도인 경우 이 나눗셈 연산기는 3 사이클만에 나눗셈 연산을 수행할 수 있다. 룩업테이블에서 읽어와서 곱셈까지 수행하여 A를 계산하는 것을 첫번째 사이클에서 수행하고, 두번째 사이클에서는 두개의 평행한 곱셈을 수행하며, 세번째 사이클에서는 최종적인 곱셈을 수행하는 방식으로 나눗셈 연산을 처리할 수 있다.

<그림 2>에는 각 기능 유닛의 지연시간이 Samsung 0.25um ASIC 공정의 컴파일드 매크로를 기준으로 제시되었다[8]. 각 곱셈기의 비트폭과 룩업테이블의 엔트리 수, 각 워드의 비트폭에 대한 것은 Ⅳ장 에러분석을 통하여 제시하였다.

Ⅳ. Error Analysis

제안하는 알고리즘의 에러 분석은 나눗셈 연산기의 설계의 가장 근간이 되는데, 이는 에러 분석이 룩업테이블의 크기와 곱셈기의 비트 폭을 결정하기 위한 기본 데이터를 제공하기 때문이다. 또한 에러 분석을 통하여 다른 나눗셈 알고리즘과의 비교가 가능하게 된다. 본 논문에서 제안하는 알고리즘에 대한 에러 분석시 고려해야 되는 항목은 총 4가지가 있다. 첫번째는 룩업테이블의엔트리 제한으로 인한 에러인데, 이 에러는 Y_h의 비트 길이가 p비트로 제한되기 때문에 발생하는 에러이다. 식 (5)는 곱셈기와 룩업테이블이 무한대의정밀도를 가진다고 보았을 때의 수식이기 때문에, 식 (5)에는 룩업테이블 제한으로 인한 에러를 제외한 다른 에러가 없다고 가정할 수 있다. 따라서, 첫번째 에러는 이상적인 몫에서 식 (5)를 빼서 얻을 수 있다. 두번째는 룩업테이블 비트폭으로 인한 에러인데, 이 에러는 룩업테이블의 정밀도에 의하여 최대값이 결정된다. 세번째는 곱셈기의 라운딩으로 인한 에러인데, 이 에러 역시 곱셈기의 정밀도에 의하여 결정된다. 마지막으로 bit-inversion으로 인한 에러가 있는데, 이 에러는 항상 1 ulp의 값을 가지게 된다.

1. 룩업테이블 엔트리 제한으로 인한 에러

룩업테이블의 엔트리 개수는 Y_h의 비트폭에 의하여 결정되는데, Y_h는 정규화되어 있으므로 엔트리 개수는 2^{p+1}이 된다. 이렇게 제한된 엔트리의 개수에 의하여 에러가 발생하게 된다. 이 에러값은 이상적인 몫에서 제안하는 알고리즘에서 X와 Y의 비트수가 무한하고, 곱셈기 역시 무한한 정밀도를 갖는다고 가정하였을 때의 몫을 빼면 계산할 수 있다.

$$
\begin{aligned}
E_{Table-entry} &= \frac{X}{Y} - (2 - AY)AX \\
&= X\{\frac{1}{Y} - \frac{Y_h - Y_l}{Y_h^2}(2 - \frac{Y_h - Y_l}{Y_h^2}Y)\} \\
&= X\{\frac{1}{Y} - \frac{Y_h - Y_l}{Y_h^2}(2 - \frac{Y_h^2 - Y_l^2}{Y_h^2})\} \\
&= X\{\frac{1}{Y} - \frac{Y_h - Y_l}{Y_h^2}\frac{Y_h^2 + Y_l^2}{Y_h^2}\} \\
&= X\{\frac{Y_h^4 - Y(Y_h - Y_l)(Y_h^2 + Y_l^2)}{YY_h^4}\} \\
&= \frac{X}{Y}\{\frac{Y_h^4 - (Y_h^2 - Y_l^2)(Y_h^2 + Y_l^2)}{Y_h^4}\} \\
&= \frac{X}{Y}\frac{Y_l^4}{Y_h^4}
\end{aligned}
\tag{6}
$$

식 (2)의 경계조건에 따라, 엔트리 제한으로 인한 에러의 최대값은 식 (7)과 같다.

$$E_{Table-entry:MAX} = \frac{X}{Y}\frac{Y_l^4}{Y_h^4}\Big|_{\substack{X,Y_l=MAX\\ Y,Y_h=MIN}}$$
$$< 2\cdot 2^{-4p+4}$$
$$= 2^{-4p+5} \quad (7)$$

또한 식 (5)는 식 (6)에 의하여 다음과 같은 등가식으로 표현할 수 있다.

$$\frac{X}{Y} \approx (2-AY)AX = \frac{X}{Y}(1-\frac{Y_l^4}{Y_h^4}) \quad (8)$$

2. 룩업테이블의 제한된 비트 폭에 의한 에러

룩업테이블의 비트 폭은 일정한 비트로 제한되어야 한다. 이로 인하여 에러가 발생하게 되는데 이 에러값이 항상 양의 아닌 수가 되도록 하였다. 그 이유는 다른 에러들은 양수이므로 룩업테이블 비트 폭 제한으로 인한 에러를 양이 아닌 수로 결정하면 전체 에러를 줄일 수 있기 때문이다. 룩업테이블의 비트폭이 q 라고 가정하면 룩업테이블에 저장되는 값은 $1/Y_h^2$ 값의 상위 q 비트까지의 값에 라운딩한 결과이다. 라운딩 모드는 round-to-plus-infinity로 한다. Round-to-plus-infinity로 라운딩을 하게 되면 라운드 비트와 스티키 비트가 모두 0인 경우를 제외하고는 올림을 하게 되는데, 이에 따라 아래와 같이 에러값이 항상 양이 아닌 수가 된다.

$$-2^{-q+1} < E_{Table-bitwidth} \le 0 \quad (9)$$

3. 곱셈기의 라운딩에 의한 에러

제안하는 알고리즘은 도합 4개의 곱셈기를 사용한다. 곱셈의 결과에 대하여 일정한 위치에서 라운딩을 수행해야 하는데, 그렇지 않다면 한 번의 곱셈을 수행할 때마다 비트 폭이 두 배로 늘어나게 되어 면적과 성능에서 큰 문제를 야기하게 된다. 곱셈기의 라운딩 모드를 round-to-nearest로 하는 경우에는 최대 에러 값을 줄일 수는 있지만, 곱셈기의 출력단에 덧셈기가 들어가야 한다. 라운딩을 위한 덧셈기를 제거하기 위하여 각 곱셈기의 rounding 모드를 round-to-zero로 하였다. Round-to-zero는 반올림의 결과가 항상 버림이며 반올림을 위한 별도의 하드웨어를 필요로 하지 않는다[9]. 각각 *M1, M2, M3, M4* 비트까지 결과값을 나타낸다면 에러는 식 (10)와 같다.

$$0 \le E_{M1} < 2^{-M1+1}$$
$$0 \le E_{M2} < 2^{-M2+1}$$
$$0 \le E_{M3} < 2^{-M3+1}$$
$$0 \le E_{M4} < 2^{-M4+1} \quad (10)$$

4. Bit-inversion에 의한 에러

*2-AY*를 계산할 때, 뺄셈 연산을 하지 않고 bit-inversion을 함으로써 항상 1 ulp의 에러가 발생되게 된다. 이 값은 M3 곱셈기에서 결과값의 비트폭에 의하여 결정되게 된다.

$$E_{bit-inversion} = 2^{-M3+1} \quad (11)$$

5. 전체 에러 합산

각 에러항을 포괄하는 전체 에러를 계산하려면, 연산 수행 과정에서 어느 부분에서 에러가 발생하여 누적되는지를 살펴 보아야 한다. 전체 에러의 최대값을 계산하게 되면, 룩업테이블의 엔트리 개수와 비트폭, 각 곱셈기의 비트폭을 결정할 수 있게 된다.

식 (5)의 $1/Y_h^2$ 항에는 룩업엔트리 개수 제한으로 인한 에러가 포함되어 있으며, A 값을 계산할 때 룩업테이블의 비트폭 제한으로 인한 에러와 *M1* 곱셈기의 라운딩에 의한 에러가 발생하게 된다. AX를 계산할 때 *M2* 곱셈기의 라운딩에 의한 에러가 발생하며, AY를 계산할 때 *M3* 곱셈기의 라운딩에 의한 에러가 발생한다. *2-AY*를 계산할때 bit-inversion으로 인한 에러가 발생하며, 최종적으로 *AX(2-AY)*를 계산할 때에 *M4* 곱셈기의 라운딩으로 인한 에러가 발생한다.

A를 계산할 때 발생되는 룩업테이블 비트폭 제한으로 인한 에러와 *M1* 곱셈기의 라운딩에 의한 에러를 수식으로 표현하면,

$$[A]_{include-error} = (\frac{1}{Y_h^2} - E_{Table-bitwidth})(Y_h - Y_l) - E_{M1}$$
$$= A - E_{Table-bitwidth}(Y_h - Y_l) - E_{M1}$$

같은 방식으로 다른 항들을 표현하면,

$$[AX]_{include-error} = [A]_{include-error}X - E_{M2}$$
$$= AX - E_{Table-bitwidth}(Y_h - Y_l)X - E_{M1}X - E_{M2}$$

$$[AY]_{include-error} = [A]_{include-error} Y - E_{M3}$$
$$= AY - E_{Table-bitwidth}(Y_h - Y_l)Y - E_{M1}Y - E_{M3}$$

$$[2-AY]_{include\ error} = 2 - AY + E_{Table-bitwidth}(Y_h - Y_l)Y$$
$$+ E_{M1}Y + E_{M3} - E_{bit-inversion}$$

위의 각 항을 바탕으로, 에러를 포함한 몫을 계산하면 아래 수식과 같다.

$$[Q]_{include-error} = (AX - E_{Table-bitwidth}(Y_h - Y_l)X - E_{M1}X - E_{M2}) \times$$
$$(2 - AY + E_{Table-bitwidth}(Y_h - Y_l)Y + E_{M1}Y + E_{M3} - E_{bit-inversion}) - E_{M4}$$

위 식에서 에러끼리 서로 곱해진 항은 그 값이 매우 작기 때문에 무시할 수 있다.

$$[Q]_{include-error} \approx Q - E_{Table-entry} +$$
$$E_{Table-bitwidth}(AXY(Y_h - Y_l) - X(2 - AY)(Y_h - Y_l))$$
$$+ E_{M1}(AXY - X(2 - AY)) - E_{M2}(2 - AY) + AXE_{M3}$$
$$- AXE_{bit-inversion} - E_{M4}$$

위 식은 에러의 2차 항을 없애서 근사화하고, 식 (8)에 의하여 $AX(2-AY)$를 $Q-E_{table\ entry}$로 대체한 식이다. 위 식에서 전체 에러를 구하면 아래 식과 같다.

$$E_{Total} \approx E_{Table-entry} + E_{Table-bitwidth}(2X(Y_h - Y_l)(1 - AY))$$
$$+ E_{M1}(2X(1 - AY)) + E_{M2}(2 - AY) - E_{M3}AX$$
$$+ AXE_{bit-inversion} + E_{M4}$$

위 식에서 AY에 $(Y_h^2 - Y_l^2)/\ Y_h^2$를 대입하면,

$$E_{Total} \approx E_{Table\ entry} + E_{Table\ bitwidth} 2X\frac{Y_l^2}{Y_h^2}(Y_h - Y_l) + E_{M1}\{2X\frac{Y_l^2}{Y_h^2}\}$$
$$+ E_{M2}\frac{Y_h^2 + Y_l^2}{Y_h^2} - E_{M3}X\frac{Y_h - Y_l}{Y_h^2} + E_{bit\ inversion}X\frac{Y_h - Y_l}{Y_h^2} + E_{M4} \quad (12)$$

식 (12)에서 $E_{table\ entry}$와 $E_{bit\ inversion}$, E_{M1}, , E_{M2}, E_{M3}, E_{M4}는 양의 에러이며, $E_{table\ bitwidth}$는 음의 에러이다. 식 (12)의 여러 항 중에서 E_{M2}항만 ( ) 부호를 지니기 때문에, 양의 최대에러는 E_{M2}가 최소값을 가지고 다른 에러항이 최대값을 가질 때이며, 음의 최대에러는 E_{M2}가 최대값을 가지고 다른 에러항이 최소값을 가질 때이다. 식 (12)에 식 (7), 식 (9), 식 (10), 식 (11)를 대입하고, X=2, Y=1, $Y_l^2/\ Y_h^2$=2^{p+2}를 넣어 근사화하면 다음과 같다.

$$-(2^{-2p-q+5} + 2^{-M3+2}) < E_{Total} < 2^{-4p+5} + 2^{-2p-M1+5} + 2^{-p-M2+3}$$
$$+ 2^{-M2+1} + 2^{-M3+2} + 2^{-M4+1}$$

최대 허용 에러값은 수체계의 정밀도와 라운딩의 정확도에 따라서 달라지게 된다. 정확한 라운딩을 하려면 최소 정밀도의 두 배 이상의 정밀도로 계산하거나 부가적인 연산이 요구된다[10]. 이는 Multiplicative 방식과 같은 approximation 방식에서 난점으로 작용하게 된다. 제안하는 구조는 1 ulp의 오차를 허용한다고 가정한다면, 최대에러는 다음과 같은 허용범위를 갖게 된다.

$$2^{-4p+5} + 2^{-2p-M1+5} + 2^{-p-M2+3} + 2^{-M2+1} + 2^{-M3+2} + 2^{-M4+1} < 2^{-m+1} \quad (13)$$

$$2^{-2p-q+5} + 2^{-M3+2} < 2^{-m+1} \quad (14)$$

위 식에서 m은 주어진 수체계에서의 정밀도를 뜻하는 비트수로 적용하려는 시스템의 사양에 의하여 결정되는 수이고, p, q, $M1$, $M2$, $M3$, $M4$는 결정하고자 하는 수이다.

6. 룩업테이블 엔트리 수와 비트폭 결정

식 (12)에서 p, q, $M1$, $M2$, $M3$, $M4$는 본 논문에서 제안하는 나눗셈 연산기를 구성하는 룩업테이블과 곱셈기의 크기를 결정하는 요소가 된다. 이 수 중에서 p는 룩업테이블을 참조하기 위한 어드레스 비트 수가 된다. p는 식 (13)의 여러 파라미터 중에서 기수가 가장 높고 여러 항에 사용되는 지배적인 수이기 때문에 p를 먼저 결정해야 한다. 특히 p가 1 증가할 때 룩업테이블의 엔트리의 개수가 두 배로 증가하기 때문에, p 값이 최소가 되도록 파라미터들을 결정해야 한다. 단정도의 시스템인 경우에는 m=24이므로, 식 (12)에 의하여 p는 8로 정할 수 있다. <표 1>는 각 m의 값에 따른 p의 값을 나타낸 것이다. <표 1>에는 엔트리의 개수도 나타냈는데, Y는 정규화되어져 있기 때문에 엔트리의 개수는 2^{p-1}이 된다.

룩업테이블의 비트폭인 q는 p가 결정된 후 식 (14)에 의하여 결정할 수 있는데 이 값에 의하여 $M1$ 곱셈기의 크기가 결정된다. q 값은 $M3$ 값을 고려하여 결정하여야 하며 다양한 variation이 가능한데, $2^{-2p-q+5} \le 2^{-M3+2}$인 경우 반대의 경우에 비하여 면적이 다소 증

가하게 됨을 분석을 통하여 알게 되었다. 본 논문에서는 $2^{-2p-q+5} > 2^{-M3+2}$ 라고 가정하고, 이때 q 값은 아래 식에 의하여 결정하였다.

$$2^{-2p-q+5} \le 2^{-m}$$

위 과정에 의하여 결정된 p, q 값을 <표 1>에 나타내었다.

표 1. 제안하는 알고리즘에서의 룩업테이블의 크기
Table 1. Lookup Table Size in the Proposed Algorithm.

m	p	# of entry	q	ROM size (Byte)	비고
15	6	32	8	32	
16	6	32	9	36	
23	7	64	14	112	
24	8	128	13	208	단정도
25	8	128	14	224	
52	15	16K	27	54K	
53	15	16K	28	56K	쌍정도

7. 곱셈기의 라운딩 위치 결정

*M1, M2, M3, M4*는 각 곱셈기의 라운딩 위치를 결정하는 비트인데, 이 4 개의 파라미터 중에서 *M1*은 *M2*곱셈기와 *M3* 곱셈기의 크기를 결정하는 파라미터이고, *M2* 곱셈기와 *M3*는 *M4* 곱셈기의 크기를 결정하는 파라미터이며, *M4*은 나눗셈기의 결과값이라서 고정적인 수이다. 따라서, *M1*을 네 개의 파라미터 중에서 우선적으로 결정해야 되고, *M1*을 결정한 다음에 *M2*와 *M3*를 결정해야 한다.

제안하는 구조에서는 단정도의 경우, p=8, q=13일 때, *M1=15, M2=28, M3=28, M4=25*로 결정할 수 있다. 이와 같이 결정하였을 경우, 단정도의 시스템에서 제안하는 나눗셈기를 구성하는 각 곱셈기의 크기는 24×13, 24×15, 24×15, 28×28이 되며, 룩업테이블의 엔트리 개수는 128개, 룩업테이블의 각 워드의 비트폭은 13비트가 되게 된다.

V. Comparison with other algorithms

이번 장에서는 P. Hung의 알고리즘과 다양한 비교를 통해서 제안한 알고리즘의 장단점을 살펴보았다. 또한, 대표적인 나눗셈 알고리즘인 단위 회귀 방식이나 Newton-Raphson, 급수 전개 방식 등과의 비교를 통하여 제안하는 알고리즘의 특성과 성능을 살펴 보았다.

1. P. Hung의 알고리즘과의 비교

Hung의 알고리즘은 <그림 1>과 같이 하나의 룩업테이블과 두 개의 곱셈기로 구성되어 있다[2]. 그리고, Hung의 알고리즘은 룩업테이블이 임계경로에 있을 때에는 1 LUT + 1 MUL의 latency, 그렇지 않을 때에는 2 MUL의 latency를 갖는다. 곱셈기의 latency는 요구되어지는 정밀도가 증가함에 따라 선형적으로 증가하며, 반면에 룩업테이블의 latency는 기하급수적으로 증가한다. 따라서, 정밀도가 낮은 경우에는 2 MUL이 전체 latency가 되며, 정밀도가 높은 경우의 전체 latency는 1 LUT + 1 MUL이다. 보통 단정도의 경우에는 룩업테이블의 latency가 곱셈기보다 더 짧기 때문에 전체 latency가 2 MUL 이며, 쌍정도의 경우에는 룩업테이블의latency가 훨씬 길기 때문에 전체 latency는 1 LUT + 1 MUL이 된다.

제안하는 알고리즘의 latency는 1 LUT + 3 MUL이다. 하지만, LUT의 크기가 Hung의 알고리즘보다 훨씬 작고, 세 개의 곱셈기 중 두 개는 승수의 비트폭이 작아 latency가 상대적으로 짧기 때문에 실제로 단정도인 경우에는 Hung의 알고리즘보다 약간 latency가 길지만, 쌍정도의 경우에는 훨씬 짧은 편이다.

두 알고리즘은 모두 파이프라인이가능한 형태(pipeline-able)이며, 파이프라인 레지스터로 단계를구분하는 경우에는 모두 1 사이클의 처리율을 갖게 된다. 이렇게 한 경우에는 latency를 다시사이클 단위로 계산해야 되는데, 28×28 곱셈기를 한 스테이지에 넣는다고 가정하면, 단정도인 경우에 latency는 Hung의 알고리즘시 2사이클이고 제안하는 알고리즘시 3사이클이다. 쌍정도인 경우에 53×28 곱셈기가 2 사이클이고 58×58 곱셈기가 3사이클이라고 가정하면, 제안하는 알고리즘의 latency는 8사이클이나, Hung의 알고리즘시 룩업테이블의 크기가 기하급수적으로 증가하여 사실상 구현이 불

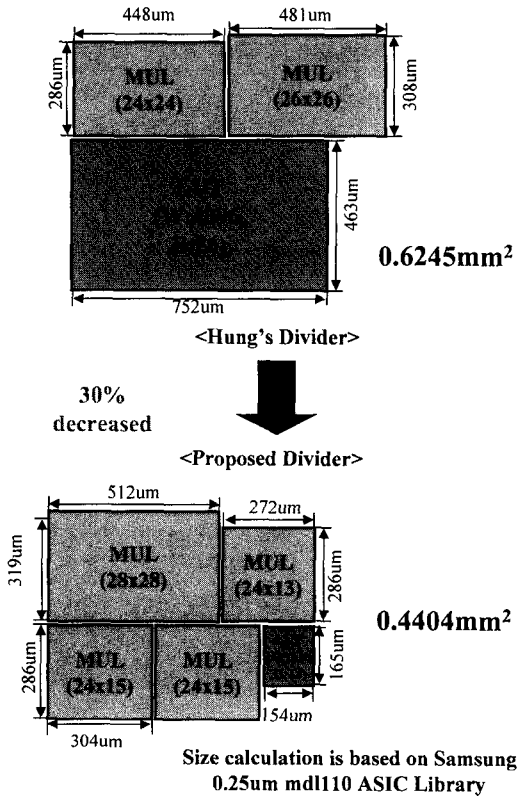


그림 3. Hung의 나눗셈 연산기와의 면적 비교
Fig. 3. Area Comparison of Pipelinable Division Algorithms.

표 2. P. Hung과 제안하는 알고리즘과의 비교
Table 2. Comparison of Pipelinable Division Algorithms.

		P. Hung's	Proposed
개관	Latency	2 MUL or	1 LUT + 3 MUL
	Throughput	1 MUL + 1 LUT	1 cycle
	LUT	1 cycle	Very smaller than Hung's
	MUL	1 (# of entries : 2m/2,	4
단정도	Latency (ns)	bit-width : m+2)	11.01
	Latency (cycles)	2	3
	LUT	7.6	208B
	MUL	2	24×13, 24×15, 24×15, 28×28 혹은 24×13, 24×17, 24×17, 27×27
쌍정도	Latency (ns)	13KB	8
	LUT		26×26, 24×24
	MU		구현 자체가 불가능

가능하게 된다.

<그림 3>은 두 알고리즘의 차지면적을 비교하기 위하여, 특정한 ASIC 테크놀로지 라이브러리를 바탕으로 산출한 단정도에서의 두 알고리즘의 차지면적을 보여주고 있다[8]. <그림 3>는 파이프라인시 필요한 플립플롭의 차지면적과 라우팅에 의한 오버헤드를 제외한 상태에서 비교한 것이다. 두 알고리즘을 비교한 결과, 제안하는 알고리즘이 Hung의 알고리즘보다 약 30%의 면적 감소가 있음을 알 수 있다. 제안하는 알고리즘과 Hung의 알고리즘과의 비교는 <표 2>에 정리되어 있다.

차지면적의 경우 처리해야 할 데이터의 정밀도가 높아질수록 제안하는 알고리즘이 유리하게 된다. 이는 곱셈기의 면적은 정밀도가 증가함에 따라 선형적으로 증가하지만, 룩업테이블의 면적은 정밀도가 증가함에 따라 기하급수적으로 증가하기 때문이다.

2. 다른 나눗셈 알고리즘과의 비교

이번 절에서는 제안하는 나눗셈 연산 알고리즘과 기존의 대표적인 나눗셈 연산 알고리즘에 대한 다양한 비교를 하였다. 이러한 비교는 매우 중요하지만 상당히 어려운 문제이다. 각각의 알고리즘에 들어가는 블록이 다양하고, 각 알고리즘이 어떤 정밀도로 구현되어지나 따라 면적과 latency가 달라지기 때문에 일정한 기준 아래 면적을 비교하는 것은 쉽지 않으며, 또한 latency 역시 수평적인 비교가 어렵다. 하지만 정확한 비교가 불가능하더라도 어느 정도의 오차를 갖는 비교는 가능하며, 이런 대략적인 비교는 필요하다. 이러한 비교를 위하여 다음과 같은 가정을 하였다.

(1) 여러가지 정밀도 중에서 단정도와 쌍정도에서만 비교한다.
(2) latency는 사이클 단위로 계산한다.
(3) 28×28 곱셈기는 한 사이클에 동작하며, 그보다 더 latency가 긴 곱셈기나 룩업테이블은 두 사이클 혹은 세 사이클에 동작한다.
(4) 24×24 곱셈기와 4KB의 룩업테이블이 면적 산출의 기본 단위로 이들의 면적을 1로 정한다[8].
(5) 곱셈기의 면적은 승수와 피승수의 비트수의 곱에 선형적으로 비례한다.
(6) 룩업테이블의 면적은 바이트 단위에 선형적으로 비례한다.

(7) SRT 나눗셈 연산기의 면적은 P. Soderquist의 분석에 의한 값을 사용한다[10].

(8) 룩업테이블의 비선형성에 대한 보정은 P. Soderquist의 분석에 의한 값을 사용한다[10].

(9) 제어유닛이나 라운딩회로는 면적 산출에서 제외한다.

(4)의 가정은 <그림 3>에서 면적 비교를 하였던 라이브러리 상에서 24×24 곱셈기와 4KB의 룩업테이블이 거의 비슷한 면적을 차지하고 있다는 점을 바탕으로 한 것이다.

(5)과 (6)의 가정은 상당한 오차를 발생시킬 수 있는 가정이다. 곱셈기의 경우에는 비트 폭의 차이가 많지 않으므로 그 오차가 그다지 크지 않으나, 룩업테이블 같은 경우에는 크기가 128B에서 64KB까지 다양하기 때문에 오차가 상당히 크게 발생할 수 있으며, 그 오차를 보정해 주어야 한다. P. Soderquist는 8 비트의 초기값을 지니는 급수전개 방식과 16비트의 초기값을 지니는 급수전개 방식 사이에 면적이 22배 차이가 나는 것을 밝혔다. 이에 따라, 본 논문에서는 (9)와 같은 가정을 하였으며, 이에 의하여 면적을 산출하였다.

표 3. 단정도에서 다른 알고리즘과의 비교
Table 3. Comparison with Other Algorithms in Single Precision.

알고리즘 및 구현방식		Latency (cycles)	Pipeline ability	Accurate remainder	# of iterations	Area	# of MULs	Size of LUT
SRT Radix-4		12	×	○	12	1.5		
Newton Raphso	8b seed	6	×	×	3	1.0	1	128B
	16b see	4	×	×	2	22	1	64KB
Series expansio	8b seed	7	×	×	3	1.0	1	128B
	16b see	5	×	×	2	22	1	64KB
Accurate quotient approximation		3	×	○	1	9.8	3	19.5KB
Hung's		2	○	×		6.8	2	13KB
Propose		3	○	×		4.6	4	240B

표 4. 쌍정도에서 다른 알고리즘과의 비교
Table 4. Comparison with Other Algorithms in Double Precision.

알고리즘 및 구현방식		Latency (cycles)	Pipeline ability	Accurate remainder	# of iterations	Area	# of MULs	Size of LUT
SRT Radix-4		27	×	○	27	1.5		
Newton Raphso	8b seed	22	×	×	1	1.0	1	128B
	16b see	16	×	×	22	22.0	1	64KB
Series expansio	8b seed	25	×	×	1	1.0	1	128B
	16b see	19	×	×	22	22.0	1	64KB
Accurate quotient approximation		9	×	○	2	9.8	3	19.5KB
Hung's							2	440MB
Propose		8	○	×		35.9	4	56KB

<표 3>은 단정도에 대하여 각 알고리즘을 비교한 결과이고, <표 4>는 쌍정도에 대하여 각 알고리즘을 비교한 결과이다. <표 3>과 <표 4>는 [5]에서 나눗셈 연산 알고리즘을 비교한 것을 바탕으로 latency를 산출하고 나눗셈기를 구성하는 각 기능유닛의 개수를 결정하였다. 특히 accurate quotient approximation 알고리즘은 두 개의 룩업테이블을 사용하는 방식을 기준으로 산출하였다[6]. 단정도의 경우에는 제안하는 알고리즘은 latency가 5 사이클 이내에 들어가는 알고리즘 중에서 가장 면적을 작게 차지한다. 따라서, 짧은 latency를 요구하는 시스템에 적용하는 경우, 효율적으로 사용할 수 있다. 특히 나눗셈의 빈도수가 높은 경우에는 높은 처리율을 가지는 특성 때문에 유용하게 사용할 수 있을 것으로 판단된다. 반면 제안하는 알고리즘은 SRT radix-4 알고리즘이나 8비트 시드를 지니는 뉴튼-랩슨 혹은 급수전개 방식보다 더 넓은 면적을 차지한다는 점에서 나눗셈의 빈도수가 낮은 시스템에서는 제안하는 알고리즘의 효율성이 적다. 쌍정도의 경우에는 P. Hung에 비하면 면적과 성능 모두 우월하지만, 나머지 알고리즘과 비교하면 면적이 너무 크다. 하지만, 높은 처리율을 요구하는 경우에는 유용하게사용할 수 있다. 따라서, 제안하는 알고리즘은 단정도나 쌍정도 모두에서 높은 처리율을 요구하는 경우에는 유용하게 사용될 수 있으며, 단정도의 경우에는 높은 처리율을 요구하지 않는다 하더라도, 면적을 줄이면서 latency를 단축하기 위하여 유용하게 사용될 수 있다.

VI. Conclusion

본 논문에서는 이터레이션 없이 파이프라인 형태로 나눗셈을 수행하는 새로운 알고리즘을 제안하였다. 또한 다른 알고리즘과 다양한 비교를 하였다. 제안하는 알고리즘은 단정밀도의 경우 4개의 곱셈기와 하나의 룩업테이블로 이루어져 있으며, 기존의 파이프라인 형태의 나눗셈 알고리즘보다 면적을 줄일 수 있는 장점이 있다. 제안한 나눗셈 연산기는 3차원 그래픽 가속기 분야와 같이 나눗셈의 빈도수가 높은 시스템에서 사용할 수 있다[11]. 제안하는 나눗셈 연산기는 수식을 이용한 해석적인 에러 분석과 상위 수준의 언어를 이용한 시뮬레이션이 수행되었다. 또한, 동작적인 수준에서

HDL로 구현되어 있다. 향후 게이트 수준의 시뮬레이션과 합성을 통하여 칩으로 구현할 계획이다.

참 고 문 헌

[1] S. F. Oberman and M. J.Flynn, "Design Issues in Division and Other Floating Point Operations,", IEEE Transactions on Computers, Vol. 46, No. 2, pp 154-161, Feb. 1997.

[2] P. Hung, H. Fahmy, O. Mencer and M. J. Flynn, "Fast division algorithm with a small lookup table," Conference Record of the Thirty- Third Asilomar Conference on Signals, Systems and Computers, Vol. 2, May pp 1465-1468, 1999.

[3] A. A. Liddicoat and M. J. Flynn, "Pipeline-able Division Unit," Technical Report No. CSL-TR-00-809, Computer Systems Laboratory, Stanford University, pp 11-28, Sep. 2000.

[4] I. Koren, Computer Arithmetic Algorithms, Prentice Hall, pp 153-161, 1993.

[5] S. F. Oberman and M. J. Flynn, "Division Algorithms and Inplementations," IEEE Transactions on Computers, Vol. 46, No. 8, pp 833-854, Aug. 1997.

[6] D. Wong and M. J. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations," IEEE Transactions on Computers, Vol. 41, No. 8, pp 981-985, Aug. 1992.

[7] C. N. Lyu and D. Matula, "Redundant Binary Booth Recoding," Proceedings of IEEE Symposium on Computer Arithmetic, pp 50-57, Jul. 1995.

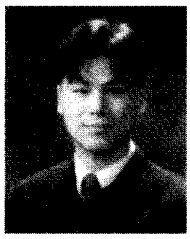
[8] Samsung Electronics Co. Ltd., MDL110 0.25um 2.5V CMOS Standard Cell Library for Pure Logic/MDL Products, 1999.

[9] ANSI/IEEE standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[10] P.Soderquist and M.Leeser, "Division and Square Root choosing the right implementations," IEEE Micro, Vol. 17, pp 56-66, Jul./Aug. 1997.

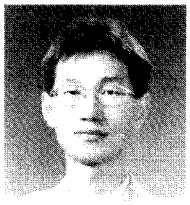
[11] A. Kugler, "The Setup for Triangle Rasterization," 11th Eurographics Workshop on Computer Graphics Hardware, pp 49-58, Aug. 26 1996 .

저 자 소 개



鄭 雄(正會員)

2000년 2월 : 연세대학교 전자공학과 공학사. 2002년 2월 : 연세대학교 전기전자공학과 공학석사. 현재 : LG전자 디지털미디어연구소 Diznet 그룹 연구원

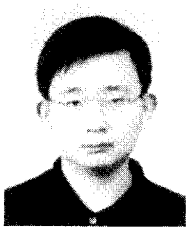


郭 承 浩(正會員)

1994년 2월 : 연세대학교 전자공학과 공학사. 1996년 8월 : 연세대학교 전자공학과 공학석사. 현재 : 연세대학교 전자공학과 박사과정

朴 祐 贊(正會員)

1993년 2월 : 연세대학교 전산과학과 공학사. 1995년 2월 : 연세대학교 전산과학과 공학석사. 2000년 2월 : 연세대학교 컴퓨터과학과 공학박사. 현재 : 세종대학교 컴퓨터공학부 교수



梁 薰 模(正會員)

1994년 2월 : 연세대학교 전자공학과 공학사. 1996년 8월 : 연세대학교 전자공학과 공학석사. 현재 : 연세대학교 전자공학과 박사과정

鄭 喆 浩(正會員)

1996년 2월 : 숭실대학교 소프트웨어공학과 공학사. 1998년 2월 : 연세대학교 컴퓨터과학과 공학석사. 현재 : 연세대학교 컴퓨터과학과 박사과정

韓 鐸 敦(正會員)

1978년 2월 : 연세대학교 전자공학과 공학사. 1983년 8월 : Wayne State Univ. 컴퓨터공학과 공학석사. 1986년 9월 : Massachusetts Univ. 컴퓨터공학과 공학박사. 현재 : 연세대학교 컴퓨터과학과 교수

李 文 基(正會員)

1965년 2월 : 연세대학교 전기공학과 공학사. 1967년 2월 : 연세대학교 전자공학과 공학석사. 1973년 2월 : 연세대학교 전자공학과 공학박사. 1980년 5월 : Oklahoma Univ. 전자공학과 공학박사. 현재 : 연세대학교 전기전자공학과 교수