Journal of The Institute of Electronics and Information Engineers Vol.52, NO.4, April 2015 http://dx.doi.org/10.5573/ieie.2015.52.4.115
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2015-52-4-15

동시연산 다중 digit을 이용한 직렬 십진 곱셈기의 설계

(Design of Serial Decimal Multiplier using Simultaneous Multiple-digit Operations)

유 창 헌*, 김 진 혁*, 최 상 방**

(ChangHun Yu, JinHyuk Kim, and SangBang Choi©)

요 약

본 논문에서는 직렬 십진 곱셈기의 성능을 향상시키는 방안을 제안하고 다중 digit을 동시에 연산하는 방안을 제안한다. 제안하는 직렬 십진 곱셈기는 부분 곱 생성단계의 2배수, 4배수를 생성하기 위한 인코딩 모듈을 없애고 쉬프트 연산만으로 부분 곱을 생성해 지연시간을 감소시킨다. 또한 다중 digit 연산을 이용해 연산의 횟수를 줄인다. 제안하는 직렬 십진 곱셈기의 성능을 평가하기 위해서 Synopsys사의 Design Compiler를 이용하여 SMIC사의 110nm CMOS 공정 라이브러리로 합성하였다. 그 결과 제안한 곱셈기는 기존의 직렬 십진 곱셈기와 비교해 전체 면적은 4% 증가하였지만, 전체 지연시간은 5% 감소함을 보였다. 또한 동시 연산 수가 증가함에 따른 제안한 다중 digit 곱셈기의 면적과 지연시간의 trade-off를 확인하였다.

Abstract

In this paper, the method which improves the performance of a serial decimal multiplier, and the method which operates multiple-digit simultaneously are proposed. The proposed serial decimal multiplier reduces the delay by removing encoding module that generates 2X, 4X multiples, and by generating partial product using shift operation. Also, this multiplier reduces the number of operations using multiple-digit operation. In order to estimate the performance of the proposed multiplier, we synthesized the proposed multiplier with design compiler with SMIC 110nm CMOS library. Synthesis results show that the area of the proposed serial decimal multiplier is increased by 4%, but the delay is reduced by 5% compared to existing serial decimal multiplier. In addition, the trade off between area and latency with respect to the number of concurrent operations in the proposed multiple-digit multiplier is confirmed.

Keywords : Decimal Multiplier, IEEE 754-2008, Multiple-Digit

Ⅰ. 서 론

최근의 컴퓨터는 이진 부동소수점(Binary Floaint Point, BFP) 연산과 십진 부동소수점(Decimal Floating Point, DFP) 연산을 지원한다. 이진 연산은 간단한 논리 연산을 위한 어플리케이션에서 유용하게 활용되지만 높은 정밀도가 요구되는 산술 연산 어플리케이션에서는 수의 표현에 있어 충분하지 않다. 이진 부동소수점 연산으로는 소수점 이하의 수를 정확히 표현할 수가 없고 십진수에 대한 정밀한 라운딩을 제공하지 못하기 때문이다. 즉 이진 연산을 사용할 때 발생하는 오차 때문에 이진 부동소수점은 환전, 세금계산, 전자상거래 등의 상업적인 기능을 수행하는 어플리케이션에 부적합하다. 이러한 배경에서 십진수를 근사치가 아닌 정확한 수치로 표현하고 불필요한 과정 없이 십진수를 직접적으로 처리할 수 있는 십진 부동소수점 연산에 대한 연구가 진행돼왔다[1~2]. 이처럼 기존의 이진 연산을 대체할 수

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering Inha University)
© Corresponding Author(E-mail: sangbang@inha.ac.kr)
Received ; December 12, 2014 Revised ; March 20, 2015
Accepted ; April 1, 2015

있는 십진 연산의 사용이 요구됨에 따라서 미국 전기 전자 학회(Institute of Electrical and Electronics Engineers, IEEE)에선 기존의 이진 부동소수점 연산만 다룬 IEEE-754 standard[3]를 개정해 십진 부동소수점 연산을 도입한 IEEE 754-2008 standard을 제정하였다[4]. 십진 연산기에서 원활한 연산을 위해선 십진수를 이진수로 표현할 수 있는 BCD (Binary Coded Decimal) 코드가 필요하다. BCD 코드는 0부터 9까지만 표현 가능하고 10부터 15까지 6개의 수를 사용하지 않기에 생기는 redundancy를 갖는다. 각 자리의 가중치에 따라 BCD-8421, 4221, 5221로 구분되는데 기존의 연구에서 덧셈 연산을 수행할 시 8421의 덧셈 연산이 나머지 두 경우에 비해서 더 효율적이라고 알려져 있다[5]. 또한 4비트 부호화 자릿수를 이용하면 기존의 BCD가 가진 단점인 redundancy를 피할 수 있고 표현 가능한 모든 숫자의 범위를 연산에서 사용할 수 있기에 십진 연산기 구현에 있어 효율이 좋다[6].

십진 곱셈기는 피승수에 따른 승수의 배수를 생성하는 부분 곱 생성 단계, 생성된 배수를 연산하는 부분 곱 축약 단계, 연산의 결과로 나온 부호화 자릿수를 BCD 코드로 표현해 주는 최종 합 단계로 이루어져 있으며 각 연산기마다 처리 방법에 따라서 부가적인 하드웨어가 추가된다. 즉 배수 생성을 간단히 해서 필요한 하드웨어를 최소화하고 생성된 부분 곱을 얼마나 효율적으로 축약하느냐에 따라서 직렬 십진 곱셈기의 성능이 결정된다. 본 논문에서는 직렬 십진 곱셈기의 성능을 향상시키는 방안에 대해 제안한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 직렬 십진 곱셈기의 개요에 대해 설명하고 M. A. Erle이 제안한 입력 조합을 최소화한 직렬 십진 곱셈기[7]와 L. Han의 배수 생성을 줄인 직렬 십진 곱셈기[8]에 대해 설명한다. Ⅲ장에서는 제안하는 직렬 십진 곱셈기의 부분 곱 생성 및 축약, 최종 변환 단계에 대해 설명한다. Ⅳ장에서는 Ⅲ장에서 제안한 곱셈기를 다중 digit 연산을 하도록 확장하는 방안에 대해 설명한다. Ⅴ장에서는 제안하는 직렬 십진 곱셈기를 구현하여 동작을 검증하고 ASIC 환경에서의 면적과 지연시간을 기존의 곱셈기와 비교해 성능을 분석하고 다중 digit 곱셈기의 digit 수에 따른 성능을 분석한다. 마지막으로 Ⅵ장에서 본 논문의 연구 내용을 요약하고 최종 결론을 제시한다.

Ⅱ. 관련 연구

2.1 직렬 십진 곱셈기 개요

십진 곱셈기는 크게 부분 곱 생성단계, 부분 곱 축약 단계, 최종 합 단계로 이루어져 있고 이중 제일 첫 단계인 부분 곱 생성 단계의 구조와 특성에 따라서 하위 구조가 바뀌게 된다. 부분 곱 생성(Partial Product Generation, PPG)단계에서는 피승수에 승수의 각 자릿수를 곱해 부분 곱을 만드는 단계이다. 이 단계를 통해서 십진 곱셈기의 연산 방식이 정해지게 된다.

십진 곱셈의 경우 승수의 각 자릿수는 $0 \sim 9$이므로, 피승수에 대해 $0X$부터 $9X$까지의 배수(multiple)를 생성해 낸 뒤 승수의 각 자릿수에 맞게 부분 곱을 선택하는 방식을 주로 사용한다. $0X$와 $1X$는 와이어(wire) 연결만으로도 피승수의 배수를 얻을 수 있다. 간단한 인코딩을 통해 얻을 수 있는 $[2X, 5X]$배수와는 달리 $[3X, 6X, 7X, 9X]$ 배수는 캐리지연이 불가피하다. 배수 생성 시 이 배수들을 덧셈을 통해 생성해주면 지연시간이 증가하므로 연산은 부분 곱 생성 단계가 아닌 부분 곱 축약(Partial Product Accumulation, PPA) 단계에서 해주는 것이 효율적이다. 선택된 부분 곱은 곱해진 승수의 자릿수를 고려해 다음 단계에 정렬해야 한다.

기존에 제안된 입력 값인 승수와 피승수를 적은 범위로 축소하여 필요한 배수의 수를 줄이는 방식은 입력 조합과 고유의 곱이 적다는 장점이 있지만 부분 곱 축약 단계의 중간 결과 값이 나오기 위해 여러 번 다른 조합의 연산단계를 거쳐야 한다는 단점이 있기에 이런 점을 보완하는 연구들이 진행되었다.

부분 곱 생성단계를 통해 생성된 승수의 각 자릿수에 해당하는 부분 곱은 해당하는 자릿수를 고려해 가산돼야 한다. 승수의 자릿수 개수만큼의 다중 피연산자 덧셈(Multi-Operand Addition) 연산을 수행해야 하므로 병렬 방식을 사용할 경우 입력 값의 길이에 따라 면적의 변동 폭이 크다. 직렬 방식을 사용할 경우 연산을 순차적으로 처리하고 한꺼번에 모든 부분 곱을 연산하지 않기 때문에 한 번 연산하는데 걸리는 시간은 병렬 연산에 비해 적다. 보통의 경우에 캐리 전달 지연을 피하기 위해서 이진 캐리 저장 가산기(Carry Save Adder, CSA) 트리를 십진수에 맞게 변형해 연산을 하거나 십진 가산기를 이용한 CS (Carry Save)구조를 사용한다. 부분 곱 축약단계를 통해 합 벡터와 캐리 벡터를 얻을

수 있다. 최종 합 단계에선 이 두 벡터를 더해서 결과 값을 얻는다. 부분 곱 축약 연산에서 부호화 자릿수를 사용했다면 이 단계에서 부호화 자릿수(Signed Digit Number)를 BCD로 변환해주므로 최종 합 단계를 최종 변환 단계라 하기도 한다.

직렬 십진 곱셈기는 병렬 방식과는 달리 피승수 $Y^{n-1}...Y^2Y^1Y^0$ 에 대해 $0X$부터 $9X$까지의 배수를 병렬로 한 번에 생성하지 않고 피승수의 각각 한 자릿수에 대한 배수를 생성하고 이를 매 클럭마다 누적 가산시켜서 최종 결과 값을 얻는다. 그렇기에 병렬 십진 곱셈기에 비해서 임계 경로가 짧고 하드웨어의 면적이 작다. 이런 특징을 갖기에 직렬 방식의 곱셈기는 높은 처리량을 요구하는 것이 아닌 전력 소모를 최소화 하는 어플리케이션에 적합하다. 각 자릿수에 대해서 하나의 Y값에 대해 배수를 생성하고 이를 이전까지의 중간 합과 합한다. 중간 합은 다음 연산의 입력이 되어 누적 연산되고 순차적으로 실행되는 연산에 따라 나온 결과 값은 BCD로 변환돼 최종 결과 값이 된다.

2.2 입력 조합을 최소화한 직렬 십진 곱셈기

입력 값의 범위에 따른 곱셈연산의 복잡도를 파악하고 입력의 조합과 고유의 결과가 가장 적은 조합을 선택해 연산을 진행하는 이 곱셈기는 입력 값인 승수와 피승수를 $[2,5]\times[2,5]$의 범위의 digit-set로 변환하고 해당하는 배수를 생성하는 유닛을 통해 부분 곱을 생성한다.

$[2,5]\times[2,5]$의 경우엔 입력 조합이 16개, 고유한 곱이 10개, 곱의 항의 총 개수는 15개를 갖기 때문에 다른 입력 조합과 비교했을 때 가장 적은 조합 수를 가진다. 하지만 2부터 5까지의 4가지 수의 조합으로 모든 배수를 만들어 내야 하기 때문에 음수변환이 필수적이다. 또한, 굳이 캐리 지연이 없어도 되는 배수의 경우도 불필요한 연산을 통해서 만들어야 한다는 단점이 있고, 부분 곱 축약 단계의 중간 결과 값이 반영되어 다음 부분 곱이 생성되어야 한다는 점에서 효율적이지 못한 특성을 가진다.

2.3 생성 배수를 줄인 직렬 십진 곱셈기

가. 부분 곱 생성 단계

기존의 연구에서 십진 곱셈기는 부분 곱 생성 단계에서 승수에 따라서 배수의 종류를 결정한다. 일반적으로 $[1X, 2X, 3X, 4X, 5X]$와 같은 배수를 생성하고 이들의 조합하여 $0\sim9X$의 부분 곱을 생성할 수 있다[9]. 기존 연구에서 $3X$ 또는 $7X$ 등의 배수는 기본 단위 배수의 조합으로 생성되기 때문에 캐리 지연이 불가피하다. 이러한 단점을 없애기 위해 $3X$ 배수를 직접 인코딩 하는 방식의 연구가 제안되었으며, 부분 곱 생성단계에서의 불필요한 캐리지연을 줄여 단위 시간당 처리량을 증가시켰다[10]. 하지만, 배수 생성 모듈의 수가 많으면 하드웨어 구조가 복잡해지는 단점이 있다. 하드웨어 구조를 간단히 하기위해선 배수 생성 모듈의 수를 최소화해야 하고 이를 이루기위해선 음수의 배수를 생성 하는 것도 고려해야 한다. 배수의 개수를 감소시키기 위해서 승수 Y를 $[-4,5]$의 범위를 갖는 digit set으로 구성한다. 이 경우에 $2X, 4X$ 두 개 배수만 생성해 모든 조합을 만들 수 있다. 그러므로 두 개의 배수 생성 모듈만 필요하고, $1X$는 별도의 인코딩 없이 와이어 연결만으로 생성가능하다.

나. 부분 곱 축약 단계

부분 곱 연산 단계 에서는 이전 단계인 부분 곱 생성 단계의 출력 값인 $U_i(1X)$와 $V_i(\pm2X, \pm4X)$ 그리고 이전 연산 결과 값인 $P[i]$, 즉 W_i와 T_i를 포함하여 총 4개의 피연산자가 입력 값이 된다. 식 (1)은 부분 곱 축약단계의 연산을 나타낸다.

$$P[i+1] = 0.1P[i] + U_i + V_i + c_i \quad (1)$$

십진수 표현에서 한 개의 digit은 4비트이므로 다음 자릿수 연산 시 4비트씩 쉬프트 한다. 식 (1)에서의 $0.1P[i]$는 다음 자릿수를 연산하기 위해 4비트 쉬프트 된 이전 결과 값을 의미한다. c_i는 음수 배수를 10의 보수로 취하기 위한 비트이다.

4개의 오퍼랜드를 가산한 결과 값인 중간 합은 $[-12,21]$의 범위를 갖는다. 이 중간 합은 부호화 자릿수이면서 다음 연산의 입력으로 사용할 수 없는 6비트이기 때문에 이를 연산 가능한 범위의 값으로 재인코딩 해야 한다. 재인코딩 후에 중간 합은 W_i 와 T_i가 되어 다음 연산의 입력으로 들어간다. 이전과 동일한 연산을 수행해야 하므로 가중치를 고려하여 W_i 와 T_i는 식 (2)가 나타내는 범위를 갖게 된다.

$$P[i] = W_i + T_i \in [-6,6]$$
$$W_i \in [-5,4],\ T_i \in [-1,2] \quad (2)$$

4개의 4:2 컴프레서와 1개의 4비트 CLA (Carry-Lookahead Adder)를 이용하여 부분 곱을 사이클 마다 연산한다. U_i를 제외한 나머지 3개의 피연산자는 부호화 자릿수이므로 부호화 자릿수 연산을 수행한다. 4:2 컴프레서와 CLA의 연산 결과 값은 6비트가 된다. 이 값은 다음 연산의 입력이 돼야 하므로 W_i와 T_i로 재인코딩 되어야 한다. 재인코딩 후 다음 클럭에서 입력이 되어 연산을 반복 수행하게 된다.

다. 최종 변환 단계

직렬 곱셈기의 특성상 부분 곱 축약이 수행될 때마다 최하위 4비트는 부호화 자릿수 결과 값이 된다. 변환 단계에서는 이를 매 사이클마다 순차적으로 BCD-8421 값으로 변환해서 최종 결과의 하위 절반(P_{low})를 얻는다. 부분 곱 축약이 종료된 후 나온 최종 결과의 상위 절반(P_{high}) 또한 부호화 자릿수-BCD 변환을 통해 BCD로 변환된다. 축약 단계에서의 사이클마다 한 자릿수씩 완성되는 하위 절반 값과 축약 단계가 종료된 후에 변환되는 상위 절반 값이 연산의 최종 결과 값을 나타낸다.

Ⅲ. 효율적인 배수 생성을 이용한 직렬 십진 곱셈기

이번 장에서는 본 논문에서 제안하는 직렬 십진 곱셈기에 대해 설명한다. 2.3절에서 설명한 기존의 생성 배수를 줄인 직렬 십진 곱셈기를 바탕으로 성능을 향상시키기 위한 방안을 제안한다.

3.1 개요

제안하는 곱셈기는 생성해야 할 배수를 최소화 한 기존의 배수 생성 모듈에서 $2X$, $4X$를 만들 때 인코딩 방법을 개선한다. 생성한 배수의 조합은 기존과 동일하기 때문에 피승수 X의 모든 배수를 표현 가능하며 생성된 부분 곱을 다중 피연산자 부호화 가산기를 통해 연산한다. 직렬 방식이므로 매 사이클의 연산 결과 값인 중간 합에 부분 곱을 누적하여 결과 값을 얻는다.

매 사이클 마다 하위 4비트의 부호화 자릿수를 BCD로 변환하고 최종 부분 곱 축약이 끝나면 나머지 상위 절반 값을 BCD로 변환하여 최종 결과 값을 얻는다. 기존의 방식과 비교했을 때 가장 큰 차이점은 $2X$ 와 $4X$ 배수를 나타내는 V_i값이 각 해당 자리에서만 처리 된다는 것이다. 또한 n 자리의 입력 값을 연산할 시 $n+1$의 자리까지 V_i가 생성됐다면 제안하는 방식에선 n자리에서만 V_i가 생성된다. 그렇기 때문에 부분 곱 축약단계에서 중간합의 범위는 증가하지만 연산은 각각 해당 자리에서만 이루어져 효율적이다.

3.2 부분 곱 생성 단계

제안하는 직렬 십진 곱셈기는 간단한 배수생성 방식과 동일하게 $2X$, $4X$만 생성하고 $1X$는 와이어 연결로 처리한다. 인코딩을 통해 배수를 생성하는 기존 방법과는 달리 짝수 배수만 필요하다는 점에 착안하여 쉬프트 연산만으로 배수를 생성하는 방법을 제안한다. 기존의 배수 생성 모듈을 전부 없애고 쉬프트 레지스터로 대체한다. 생성될 수 있는 배수의 최대치인 36 ($X=4$ 일 때 $4X$)을 표현하기위해 V_i값이 저장될 레지스터의 길이가 3비트 증가한다. 7비트의 길이를 갖는 V_i는 최상위 비트를 부호화 비트로 갖는다. 기존의 $2X$, $4X$를 인코딩하는 방식은 다음 배수를 만들기 위해 이전의 캐리인 T_i 값이 필요하다는 단점이 있는데 제안하는 방법에서 각 자릿수는 해당 자리에서만 처리되므로 필요한 하드웨어 복잡도가 감소하며, 향상된 처리 속도를 갖는다. 음수배수를 생성하는 경우에는 보수만 취해주고 따로 c_i비트를 통해서 부분 곱 축약 단계에서 이를 가산해 음수로 만들어 준다.

3.3 부분 곱 축약 단계

부분 곱 생성 단계에서 만들어진 n자릿수의 부분 곱을 연산하는 단계로 U_i ,V_i ,W_i ,T_i 총 4개의 피연산자를 사용한다. 4개의 피연산자를 2개의 부분 합으로 나타내기 때문에 4:2 컴프레서를 사용한다. 각 입력에 대한 범위는 식 (3)과 같다.

$$U_i \in [0,9]$$
$$V_i \in [-36,36] \quad (3)$$
$$W_i \in [-5,4],\ T_i \in [-4,5]$$

$$P[i] = W_i + T_i \in [-9,9]$$

V_i를 인코딩 없이 쉬프트 연산으로 생성하기 때문에 기존의 방법보다 범위가 증가하게 되며, 이를 고려하여 중간 합의 범위도 늘어나게 된다. 하지만 V_i를 인코딩을 통해 생성했을 시 나오는 다음 자릿수가 필요 없기에 전달 지연을 줄일 수 있어 성능 향상을 보인다. 그림 1에서 제안하는 부분 곱 축약 단계의 연산과정을 점 표현(dot notation)으로 나타낸다.

그림 1에서 하얀 점은 음수 비트이므로 이를 고려하여 부호화 자릿수 가산을 한다. 연산 결과 나온 중간 합의 5번째와 7번째 비트는 반전하여 계산하여 결과 값을 얻게 된다. 8비트의 중간 합이 다음 연산의 W_i ,T_i로 바뀌어야 하므로 재인코딩을 실시해야 한다. 이전 자릿수의 결과로 나온 T_i값과 현재 연산에서 나온 W_i가 부분 곱 생성 단계에서의 출력 값 U_i V_i와 함께 해당 자릿수의 다음 연산 입력으로 들어가게 된다. 그림 2는 제안하는 부분 곱 축약 단계의 블록 다이어그램을 나타낸다.

그림 1에서 확인 할 수 있듯이 V_i의 범위가 늘어나므로 기존 4비트의 CLA대신 6비트의 CLA를 사용한다. 이때, CLA의 최 상위 입력은 V_6 하나만 들어가기 때문에 최종 캐리를 look-up하는 로직은 일반적인 6비트 CLA로직에 비해 간단하다.

중간 합의 재인코딩 결과인 T_{i+1}과 W_i는 각각 다음 연산의 앞 자릿수와 해당 자릿수의 입력으로 들어가 연산이 반복 수행된다. 피연산자의 길이가 제각각인 부호화 자릿수는 부호화 전가산기 방식을 이용해 연산을 수행하게 된다. 피승수 Y의 자릿수에 따라서 반복의 횟수가 정해지고 반복 연산이 끝나면 결과 값의 상위 절반이 최종 부호화 자릿수가 된다.

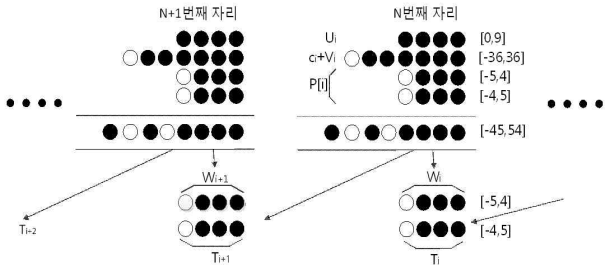


그림 1. 제안하는 부분 곱 축약 단계의 처리과정
Fig. 1. Process of the proposed PPA stage.

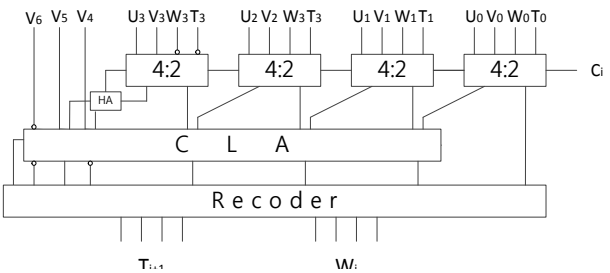


그림 2. 제안하는 부분 곱 축약 단계의 블록 다이어그램
Fig. 2. Block diagram of the proposed PPA block diagram.

3.4 최종 변환 단계

부분 곱 축약단계에서 매 사이클 연산 시 나오는 하위 4비트는 기존의 곱셈기와 동일한 범위를 가지므로 순차적으로 BCD로 변환하여 하위 절반(P_{low})를 얻는다. 부분 곱 축약 연산이 종료된 후 나온 상위 절반(P_{high}) 부분의 합 벡터인 W_i는 $[-5,4]$의 범위를 가

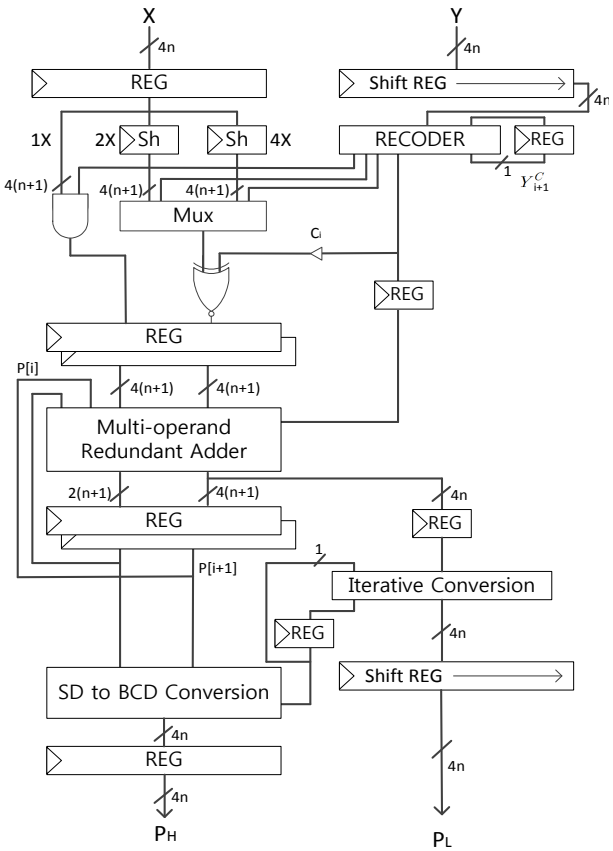


그림 3. 제안하는 직렬 십진 곱셈기의 전체구조
Fig. 3. Top level architecture of proposed serial decimal multiplier.

지므로 기존의 직렬 십진 곱셈기와 동일하지만 캐리벡터인 T_i는 $[-4,5]$의 범위를 갖기에 4비트가 된다. 따라서 나올 수 있는 값의 범위는 $W_i + T_{i-1} \in [-9,9]$가 된다. 기존 $[-8,7]$ 범위를 변환하는 변환단계에서 $-9,8,9$ 세 가지 경우를 고려해 로직을 확장하여 동일한 방법으로 부호화 자릿수-BCD 변환을 해서 결과 값을 얻는다. 그림 3은 제안하는 직렬 십진 곱셈기의 전체 구조를 나타낸다.

Ⅳ. 다중 digit을 이용한 직렬 십진 곱셈기

이번 장에서는 Ⅲ장에서 제안한 직렬 십진 곱셈기를 바탕으로 승수를 1 digit씩이 아닌 다중 digit을 연산하도록 확장하는 방안에 대해 설명한다.

4.1 개요

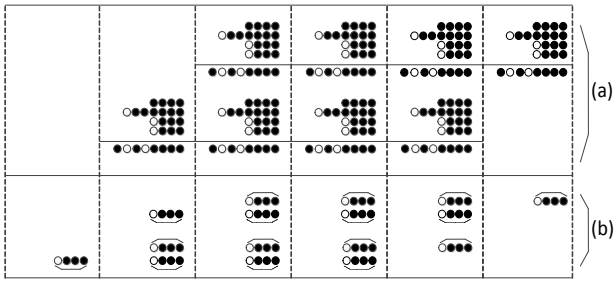
피승수 X에 대해 승수 Y의 1digit 씩 연산하는 기본적인 직렬 곱셈방법에서 동시에 다중 digit을 처리하도록 개선하면 반복 연산 수가 크게 감소해 총 지연시간이 감소하게 된다.

동시에 다중 digit을 처리할 수 있도록 개선하기 위하여는 직렬 십진 곱셈기의 배수 생성 로직을 동시에 여러 수를 처리할 수 있도록 확장하고 동시에 연산하는 digit 수에 따라서 요구되는 레지스터의 길이를 증가시켜야 한다. 또한 처리 digit 수에 따라서 부분 곱 축약 단계에서 다중 피연산자 가산기와 재인코딩을 하는 리코더(Recoder)를 추가하여야 한다.

이러한 십진 곱셈기의 개선은 향상된 처리속도를 제공하지만 추가되는 하드웨어의 영향으로 복잡도 및 면적이 크게 증가하는 특징이 있다. 따라서, 최적화된 성능을 보장하기 위해서는 등가교환 관계에 있는 총 지연시간과 하드웨어 복잡도를 적절히 선택해야 한다.

4.2 승수의 n digit 동시 연산

$n=2$인 경우에 생성되는 배수의 개수는 기존의 2배가 된다. 그러므로 두 개의 digit을 동시에 처리하는 부분 곱 생성 단계에선 승수 Y를 $[-4,5]$의 범위를 갖도록 인코딩하는 로직과 쉬프트 레지스터를 추가해서 두 자리에 해당되는 U_i, V_i 벡터를 동시에 생성한다. 따라서 부분 곱 축약 단계의 첫 입력은 두 개의 U_i, V_i



(a) Operation for 2 digit of multiplier Y
(b) Wi, Ti vector

그림 4. 제안하는 다중 digit 연산 과정
Fig. 4. Process of the proposed multiple-digit operation.

벡터가 된다. 한번 연산 과정을 통해서 2 digit씩 처리되기 때문에 8비트의 중간 합 벡터 또한 두 개가 된다. 따라서 중간 합이 다음 연산의 입력이 되기 위해 4비트의 W_i ,T_i로 재인코딩 되어야 하고 이때 두 자리가 동시에 재인코딩 되어야하므로 리코더 수는 2배가 된다. 그림 4는 제안하는 다중 digit($n=2$) 부분 곱 축약 단계의 점 표현을 나타낸다.

그림 4의 (a)의 각 자리는 3장에서 제안한 직렬 십진 곱셈기의 부분 곱 축약단계를 의미한다. 생성된 두 개의 중간 합 벡터는 기존의 재인코딩 방법을 통해서 (b)에서 나타내는 4비트의 W_i와 T_i 벡터가 되고 이전 자릿수의 결과로 나온 T_i값과 현재 연산에서 나온 W_i가 한 사이클 뒤 부분 곱 생성 단계 출력 값 U_i V_i와 함께 해당 자릿수의 다음 연산 입력으로 들어가게 된다. 기존의 직렬 십진 곱셈기에서 연산 반복 시에 다음 자릿수를 연산하기 위해서 1digit 쉬프트 했다면 다중 digit 곱셈 방식에선 동시에 연산한 digit 수만큼 쉬프트 해야 한다.

부분 곱 축약단계에서 매 사이클 연산 시 최하위 두 digit에 해당되는 값이 나온다. 두 개의 digit 중 하위 값은 기존과 동일하게 $[-5,4]$의 범위를 가지며, 상위 값은 기존 변환 단계의 $W_i + T_{i-1} \in [-9,9]$에 앞자리 T_i값이 추가되어 $[-13,14]$의 범위를 갖는다. 이를 고려하여 하위 digit과 상위 digit을 변환하는 로직을 따로 구분해서 사이클 마다 순차적으로 2digit 씩 BCD 변환하여 하위 절반(P_{low})를 얻는다. 부분 곱 축약 연산이 종료된 후 나온 상위 절반(P_{high}) 부분의 각 자리 범위는 $2\times[-9,9]$로 $[-18,18]$이 된다. 이 또한 기존의 부호화 자릿수-BCD 변환 알고리즘을 동일하게 이용한

변환 과정을 통해 상위 절반(P_{high})의 BCD값을 얻어 최종 결과 값을 얻는다.

$n=2$인 경우를 확장하여 $n>2$인 경우에도 동시에 처리되는 digit 수만큼 해당 로직이 추가되고 연산이 한 번 수행될 때마다 수행되어야 하는 쉬프트 수가 늘어난다. 최종 변환 단계에서의 하위 절반(P_{low})을 순차적으로 BCD값으로 변환하는 로직이 동시에 처리하는 digit 수만큼 필요하고 처리 범위 또한 크게 증가하므로 많은 면적 용량을 요구하게 되지만 총 연산 반복수가 크게 감소한다.

Ⅴ. 성능 분석

5.1 시뮬레이션 및 동작 검증

본 논문에서 제안한 직렬 십진 곱셈기를 VerilogHDL을 사용해 구현하였고, 동작을 검증하기 위해서 RTL (Register Transfer Level)시뮬레이션을 Mentor Graphics사의 Modelsim을 이용해 수행하였다. 하위 모듈을 먼저 설계하고 RTL 시뮬레이션을 통해 동작을 검증한 후 최 상위 모듈을 구성하는 방식으로 설계를 진행하였다. 4비트의 BCD코드가 입력의 한 자릿수가 되는데 각 스테이지의 세부 동작을 확인 할 수 있도록 충분한 길이인 64비트 즉, 16digit의 테스트 벡터를 선정해 동작을 검증하였다.

5.2 합성 결과

이번 절에서는 성능 평가를 위해 기존의 간단한 배수 생성을 이용한 직렬 십진 곱셈기와 제안하는 직렬 십진 곱셈기를 동일한 ASIC 환경에서 합성(synthesis)하여 비교한다. 또한 다중 digit을 이용한 직렬 십진 곱셈기의 처리 digit수에 따른 성능 비교를 진행한다. 설계한 모듈의 논리 합성을 위해 Synopsys사의 Design compiler를 이용하였고 SMIC $110nm$ CMOS 공정 라이브러리를 사용하였다.

표 1은 기존의 배수 생성을 줄인 L. Han의 직렬 십진 곱셈기와 제안한 직렬 십진 곱셈기를 동일한 환경과 컴파일 옵션을 이용해 합성한 결과이다.

합성 결과 기존의 간단한 배수생성을 이용한 직렬 십진 곱셈기의 면적은 $39400\mu m^2$, 지연시간은 $2.57ns$를 얻을 수 있었고, 제안하는 직렬 십진 곱셈기의 면적은

표 1. 합성 결과
Table 1. Synthesis result.

Architecture	Delay		Area	
	ns	Ratio	μm^2	Ratio
L. Han	2.57	1.066	39400	0.960
Proposed	2.41	1.000	41030	1.000

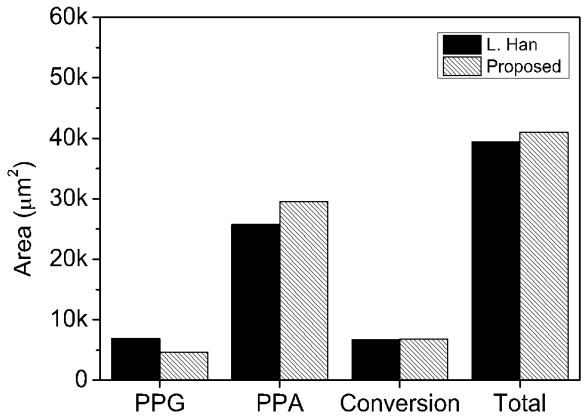


그림 5. 각 하위 모듈의 면적 비교
Fig. 5. Area comparison of each sub-module.

$41030\mu m^2$, 지연시간은 $2.41ns$를 얻을 수 있었다. 합성의 결과로 나온 수치를 바탕으로 제안한 직렬 십진 곱셈기가 면적 대비 약 4% 증가했고, 지연시간 대비 약 6% 감소함을 확인하였다.

그림 5는 기존의 직렬 십진 곱셈기와 제안하는 직렬 십진 곱셈기의 각 단계별 면적을 분석한 결과이고 그림 6은 각 단계별 소요되는 지연시간에 대해 분석한 결과이다.

전체 모듈의 각 구성 요소의 면적을 비교해보면 기존의 $2X\sim5X$를 만들기 위한 인코딩 모듈이 있던 것과 비교해 제안하는 배수생성 단계는 단지 $2X, 4X$를 만들기 위한 쉬프트레지스터만 있으면 되기에 면적이 32% 감소함을 확인 할 수 있다.

하지만 쉬프트를 통해 생성된 배수의 늘어난 범위 때문에 레지스터의 길이가 길어지고 이에 따라 부분 곱 축약단계에서 연산을 통해 나온 중간 합의 범위가 기존의 곱셈기보다 증가하기 때문에 CLA 모듈이 복잡해진다. 즉 부분 곱 생성단계가 개선 됐지만 부분 곱 축약단계의 다중 피연산자 부호화 가산기와 재인코딩을 실시하는 리코더의 면적이 커져 기존보다 14%의 증가를 보인다. 또한 최종 변환 단계에서도 약 2%의 면적 증가가 있다. 모든 요소가 반영된 총 면적은 4% 증가한

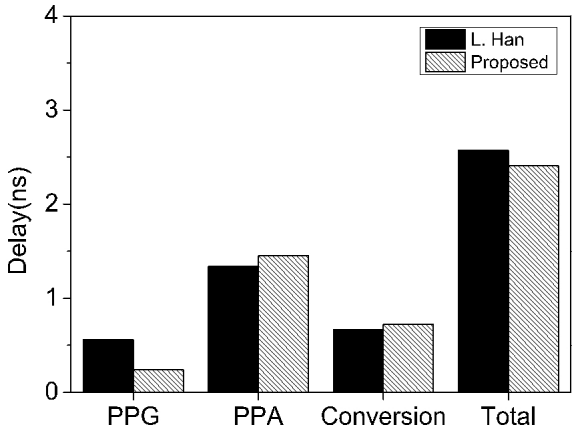


그림 6. 각 하위 모듈의 지연시간 비교
Fig. 6. Delay comparison of each sub-module.

$41030\mu m^2$ 가 된다.

부분 곱 생성단계에는 기존의 방식에서 실시되는 인코딩이 제안하는 곱셈기에선 필요 없기 때문에 지연시간이 약 57% 라는 큰 감소 폭을 보인다.

하지만 연산 과정에서 쉬프트로 생성된 배수의 범위가 커지고, 이에 따라 부분 합의 범위가 늘어나 다중 피연산자 연산과 재인코딩을 하는 부분 곱 축약 단계의 지연시간이 약 8% 증가하고, 기존 2비트인 T_i가 4비트로 늘어남에 따라 최종 변환 단계의 지연시간도 약 6% 증가하게 된다. 기존의 직렬 십진 곱셈기에서 인코딩이 부분 곱 생성 단계와 부분 곱 축약단계에서 2번 실시되는 것과 비교해서 제안하는 직렬 십진 곱셈기에선 인코딩이 부분 곱 축약단계에서 한번만 실시되기 때문에 총 지연시간은 약 6% 감소함을 확인하였다.

표 2는 비교대상이 되는 직렬 십진 곱셈기들의 합성 결과를 FO4 지연시간과 NAND2 등가게이트 수로 비교한 결과이다. FO4 지연시간과 NAND2 등가게이트의 수를 기준으로 기존의 직렬 십진 곱셈기와 비교했을 때 제안한 직렬 십진 곱셈기의 지연시간은 M. A. Erle 곱셈기와 비교해 약 10% 향상됐고 면적은 약 49% 감소

표 2. 기존 직렬 십진 곱셈기와의 비교
Table 2. Comparison with existing serial decimal multiplier.

	Area (NAND2)	Ratio	Latency (FO4)	Ratio
Proposed	9325	1	264	1
L. Han	8960	0.96	280	1.06
M. A. Erle	18550	1.98	294	1.11

표 3. 동시 연산 수에 따른 면적 비교
Table 3. Area comparison with respect to the number of concurrent operations.

	Area (μm^2)			
	n=1	n=2	n=4	n=8
PPG	4630	9270	18520	37120
PPA	29550	59120	118330	236580
Converasion	6850	10740	28170	59810
Total	41030	79230	165020	333510

되어 지연시간과 면적에서 성능이 뛰어남을 확인 할 수 있고 L. Han 곱셈기와 비교하면 지연시간은 약 5% 향상됐고 면적은 약 4% 증가해 지연시간에서 성능 향상이 있음을 확인하였다. 따라서 제안한 직렬 십진 곱셈기가 최근의 다른 직렬 십진 곱셈기와 비교해 지연시간에서 더 높은 성능을 가진다고 할 수 있다.

표 3은 제안하는 다중 n digit 곱셈기의 동시 연산 수 n값에 따른 각 스테이지 별 면적을 나타낸다. 부분 곱 생성 단계와 부분 곱 축약 단계의 경우 동시에 처리되는 digit 수만큼 해당 로직이 추가되고 레지스터가 늘어난다. 최종 변환 단계에서는 digit 수가 늘어남에 따라 절반(P_{low})을 순차적으로 BCD값으로 변환하는 로직이 증가폭이 점점 커진다. 그러므로 동시에 처리하는 digit 수가 커진다면 곱셈기의 면적이 크게 늘어나 비효율적이다.

표 4는 동시 연산 수에 따른 곱셈기의 총 지연시간을 나타낸다. 동시 n digit 연산 곱셈기의 사이클 수는 식 (4)와 같다.

$$Number\ of\ cycle = (\frac{input\ digit}{n}) + 4 \qquad (4)$$

입력이 16digit 이므로 $n=1$인 곱셈기는 사이클 수

표 4. 동시 연산 수에 따른 총 지연시간 비교
Table 4. Total latency comparison with respect to the number of concurrent operations.

	n=1	n=2	n=4	n=8
The number of cycles	20	12	8	6
Latency (FO4)	264	186	135	116
Ratio	1	0.70	0.51	0.43

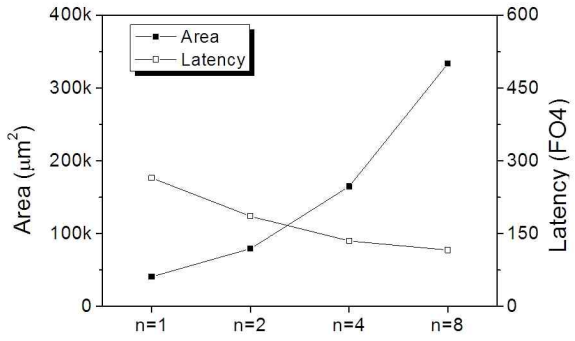


그림 7. 동시 연산 수에 따른 면적&지연시간 비교
Fig. 7. Comparison of area&Latency with respect to the number of concurrent operations.

가 20이 되고, 다중 연산 시 $n=2$인 경우 12, $n=4$인 경우 8, $n=8$인 경우 6의 사이클 수를 가진다. 직렬 십진 곱셈기의 각 단계별로 수행되어야 하는 고정 연산 때문에 다중 연산의 digit 수에 반비례하여 사이클 수가 줄어들지 않는다. 즉 동시에 더 많은 digit을 처리하는 것이 무조건 효율이 좋다고 볼 수 없다.

그림 7은 다중 digit 곱셈기의 동시 연산 수에 따른 면적과 총지연시간의 관계를 나타낸다. $n=1$인 곱셈기와 비교했을 시 연산 속도는 각각의 다중 digit 연산 당 1.4배, 1.95.배, 2.27배 빨라졌고, 면적은 각각 1.9배, 4.02배, 8.12배 늘어났다. 즉 승수 Y의 2개 digit $n=2$을 동시에 연산하는 직렬 곱셈기가 면적대비 지연시간 면에서 약간의 성능 향상을 보이고 그 이상의 경우는 비효율적이라 볼 수 있다. 본 시뮬레이션의 입력 값의 길이는 16 digit으로 그림 7에서 보이듯 n이 입력 값의 길이에 근접함에 따라 면적은 급격하게 증가함에도 불구하고 지연시간의 성능은 크게 나아지지 않는다. 입력 값의 digit 수가 늘어나면 면적 대비 지연시간은 더 향상될 것으로 예상된다.

Ⅵ. 결 론

십진 부동소수점은 십진수를 근사치가 아닌 정확한 수치로 표현이 가능하고 직관적으로 수치를 확인할 수 있기에 이진 연산보다 효율적이다. 이러한 배경에서 십진 연산의 성능을 향상시키기 위한 많은 연구가 진행되고 있다.

본 논문에서는 직렬 십진 곱셈기의 부분 곱 생성단계를 개선하여 성능을 향상시키는 방안을 제안하고 이를 통해 다중 digit을 연산하도록 확장하는 방안을 제안하였다. 생성되는 배수의 개수를 줄이는 방법은 병렬 방식에서는 부분 곱 축약단계의 피연산자가 증가하기 때문에 비효율적이지만 직렬 방식에서는 해당 자릿수마다 연산이 고정되어 있으므로 연산 과정의 개선을 이룰 수 있다.

제안하는 직렬 십진 곱셈기는 부분 곱 생성단계의 $2X$, $4X$ 배수를 생성하기 위한 인코딩 모듈을 없애고 쉬프트 연산만으로 이를 생성해 빠르게 부분 곱을 생성할 수 있다. 또한 연산이 진행 될 때 이전 자리에서 생기는 캐리를 반영해야하는 방법과 달리 각 자릿수는 해당 자리에서 연산이 되므로 효율적이다. 또한 이를 바탕으로 한 다중 digit 곱셈은 피승수 X에 대해 승수 Y의 1digit 씩 연산하는 기본적인 직렬 곱셈방법에서 동시에 다중 digit을 처리하도록 개선해 반복 연산 수가 크게 감소해 총 지연시간이 감소하게 된다.

제안하는 직렬 십진 곱셈기의 성능을 평가하기 위해서 Synopsys Design Compiler를 이용하여 SMIC사의 $110nm$ CMOS 공정 라이브러리로 합성하였다. 제안하는 배수생성 단계는 인코딩 모듈이 없고 단지 쉬프트레지스터만 있으면 되기에 면적이 감소하고 지연시간 또한 감소한다. 하지만 필요한 레지스터의 길이가 길어지고, 부분 곱 축약단계에서 연산을 통해 나온 중간 합의 범위가 기존의 곱셈기보다 크기 때문에 연산 단계에서 다중 피연산자 연산기가 복잡해진다. 결과적으로 총 면적은 기존의 곱셈기와 비교했을 때 4% 증가하지만 총 지연시간은 6% 감소함을 확인해 지연시간에서 나은 성능을 가진다고 할 수 있다. 또한 다중 digit을 이용한 곱셈을 진행할 시 동시 연산 수가 증가함에 따른 총 지연시간의 감소 폭에 비해 면적의 증가 폭이 크므로 승수 Y의 2개 digit을 동시 연산 할 수 있는 직렬 곱셈기가 면적 대비 지연시간에서 성능 향상을 보이고 그 이상의 경우는 비효율적이라 볼 수 있다. 입력 값의 digit 수가 늘어나면 다중 연산의 성능이 더 향상될 것으로 예상된다.

REFERENCES

[1] H. A. H. Fahmy, M. Y. Hassan, Y. Farouk, and R. R. Eissa, “Decimal Floating Point for future processors,” IEEE International Conference

Microelectronics(ICM '09), pp. 443-446, Dec. 2010.
[2] H. A. H. Fahmy, R. Raafat, A. M. Abdel-Majeed, R Samy, and Y. Farouk, "Energy and Delay Improvement via Decimal Floating Point Units," *In proceedings of 19th IEEE International symposium Computer Arithmetic (ARITH '09)*, pp. 221-224, Jun 2009.
[3] IEEE, IEEE 754 Standard for Binary Floating-Point Arithmetic, 1985.
[4] IEEE, IEEE 754-2008 Standard for Floating-Point Arithmetic, 2008.
[5] C. Coulston and V. Dave, "Addition of BCD Digits using Non-Standard Codes," *22nd International Conference*, pp. 148-152, Feb. 2012.
[6] J. Ghassem and P. Behrooz "Constant-time addition with hybrid-redundant numbers: theory and implementations," *Integration, the VLSI Journal*, Vol. 41, no. 1, pp. 49-64, Jan. 2008.
[7] M. A. Erle, E. M. Schwarz, and M. J. Schulte, "Decimal multiplication with efficient partial product generation," *In proceedings of 17th IEEE symposium on computer arithmetic*, pp. 21-28, Jun. 2005.
[8] L. Han, A. Kaivani, and S. B. Ko, "Area Efficient Sequential Decimal Fixed-point Multiplier," *Journal of Signal Processing Systems*, Vol. 75, no. 1, pp. 39-46, Apr. 2013.
[9] L. Han and S. B. Ko, "High speed parallel decimal multiplication with redundant internal encodings", *IEEE Transactions on Computers*, Vol. 62, no. 5, pp. 956-968, May. 2013.
[10] I. K. Hwang, K. H. Kim, W. O. Yoon, and S. B. Choi, "Design of Parallel Decimal Multiplier using Limited Range of Signed-Digit Number Encoding," *Journal of The Institute of Electronics Engineers of korea*, Vol. 50, no. 3, pp. 50-58, Mar. 2013.

저 자 소 개

유 창 헌(학생회원)
2013년 인하대학교 전자공학과 학사 졸업.
2015년 인하대학교 전자공학과 석사 졸업.
<주관심분야: 컴퓨터 구조, 병렬 및 분산 컴퓨팅>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주).
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수.
<주관심분야: 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>

김 진 혁(학생회원)
2009년 인하대학교 전자공학과 학사 졸업.
2011년 인하대학교 전자공학과 석사 졸업.
2011년~현재 인하대학교 전자공학과 박사 과정.
<주관심분야: 컴퓨터 구조, 멀티미디어 통신, 컴퓨터 네트워크, 병렬 및 분산 컴퓨팅>