論文2003-40SD-3-4

# NST 알고리즘을 이용한 비동기식 16비트 제산기 설계

# (Design of Asynchronous 16-Bit Divider Using NST Algorithm)

李 祐 錫 *, 朴 錫 在 **, 崔 滉 鎔 ***

(Woo-Seok Lee, Seok-Jae Park, and Ho-Yong Choi)

요 약

본 논문에서는 NST (new Svoboda-Tung) 알고리즘을 이용한 비동기식 제산기의 효율적 설계에 관해 기술한다. 본 제산기설계에서는 비동기 설계방식을 사용하여 제산연산이 필요할 때에만 동작함으로써 전력소모를 줄이도록 설계한다. 제산기는 비동기식 파이프라인 구조를 이용한 pre-scale부, iteration step부, on-the-fly converter부의 세부분으로 구성된다. Pre-scale부에서는 새로운 전용 감산기를 이용하여 적은 면적과 고성능을 갖도록 설계한다. Iteration step부에서는 4개의 division step을 갖는 비동기식 링 구조로 설계하고, 아울러 크리티컬 패스(critical path)에 해당하는 부분만을 2선식으로, 나머지 부분은 단선식으로 구성하는 구현방법을 채택하여 하드웨어의 오버헤드를 줄인다. On-the-fly converter부는 iteration step부와 병렬연산이 가능한 on-the-fly 알고리즘을 이용하여 고속연산이 되도록 설계한다. 0.6㎛ CMOS 공정을 이용하여 설계한 결과, 1,480 × 1,200um$^2$의 면적에 12,956개의 트랜지스터가 사용되었고, 41.7ns의 평균지연시간을 가졌다.

**Abstract**

This paper describes an efficient design of an asynchronous 16-bit divider using the NST (new Svoboda-Tung) algorithm. The divider is designed to reduce power consumption by using the asynchronous design scheme in which the division operation is performed only when it is requested. The divider consists of three blocks, i.e. pre scale block, iteration step block, and on-the-fly converter block using asynchronous pipeline structure. The pre-scale block is designed using a new subtracter to have small area and high performance. The iteration step block consists of an asynchronous ring structure with 4 division steps for area reduction. In other to reduce hardware overhead, the part related to critical path is designed by a dual-rail circuit, and the other part is done by a single-rail circuit in the ring structure. The on-the-fly converter block is designed for high performance using the on-the-fly algorithm that enables parallel operation with iteration step block. The design results with 0.6㎛ CMOS process show that the divider consists of 12,956 transistors with 1,480 × 1,200um$^2$ area and average-case delay is 41.7ns.

**Keywords** : asynchronous, divider, NST algorithm, ring structure, subtracter

* 正會員, 忠北大學校 半導體工學科
(Dept. of Semiconductor Eng., Chungbuk Nat'l Univ.)
** 學生會員, *** 正會員, 忠北大學校 電氣電子컴퓨터工學部
(School of Electrical & Computer Eng.)

※ 본 연구는 한국과학재단 목적기초연구(R05-2000-000-00254)와 IDEC의 툴 지원으로 수행되었음.
接受日字:2002年5月13日, 수정완료일:2003年2月18日

## Ⅰ. 서 론

반도체 제조기술의 발달로 인해 소자의 크기가 미세화 되면서 시스템 원칩화(SOC, system on a chip)가 가능하게 되었다. 그러나 많은 소자들이 하나의 칩에 집적화 되면서 상대적으로 배선의 길이가 길어지게 되고있다. 이에 따라 전역클록 (global clock)에 의해 시스템이 제어되는 기존의 동기식 시스템에서는 클록스큐(clock-skew)라는 물리적인 제약으로 제조기술의 발달에 따른 시스템의 성능향상이 한계에 이르고 있다. 최근 이러한 문제점을 해결하기 위해 비동기식 설계방법을 이용한 시스템 설계가 활발히 연구되고 있다[1-2].

비동기식 설계방법은 전역클록을 사용하지 않기 때문에 클록스큐에 의한 문제가 발생하지 않으며, 모든 블록의 평균지연에 의해 시스템의 속도가 결정되는 평균지연성능을 갖는다. 또한 연산이 필요한 부분에서만 회로가 동작하기 때문에 저전력 소모의 장점을 가지고 있다.

한편, 최근 3D 그래픽 처리 등의 고속연산을 위해서는 고성능의 연산처리기가 필요하며, 이에 따라 고성능 제산기 설계의 중요성이 점점 증가하고 있다[3].

제산연산을 위한 고속 알고리즘들이 다수 제안되고 있고, 주요 알고리즘으로는 SRT 알고리즘과 NST (new Svoboda-Tung) 알고리즘이 있다[4-6]. SRT 알고리즘은 감산과 이동연산에 의해 제산 연산을 수행한다. 피제수(부분 나머지)와 제수의 값을 모두 고려하여 몫을 결정하여, 몫을 결정하기 위한 연산이 복잡하다는 단점을 갖는다[5]. NST 알고리즘 등의 Svoboda계열 알고리즘의 경우는 제수의 범위를 제한함으로써 제수를 고려하지 않고 피제수 또는 부분 나머지의 값만을 가지고 몫을 결정한다[6]. 몫 결정을 위한 연산이 간단하다는 장점을 가지고 있으나, 제수의 범위를 한정하기 위한 pre-scale 연산과정이 추가되어 설계 구현을 할 경우 면적이 증가하고 성능이 저하되는 단점이 있다.

본 논문에서는 Svoboda 계열의 NST (new Svoboda-Tung) 알고리즘을 이용한 비동기식 제산기의 효율적인 설계를 제안한다. 기본적으로는 제산기 전체가 비동기식 설계방식을 이용하므로, 제산연산이 필요할 때에만 동작함으로써 전력소모를 줄이도록 설계된다. 제산기는 비동기식 파이프라인 구조를 이용한 pre-scale부, iteration step부, on-the-fly converter부로 구성된다. 본 제산기에는 비동기식 제어방법과 전용 감산기, 비동기식 링 구조, dual-rail과 single-rail의 혼합회로 등의 효율적인 설계기법들을 사용함으로써 면적감소와 성능향상을 꾀한다.

본 논문의 구성은 다음과 같다. 2장에서는 NST 알고리즘을 기술하며, 3장에서는 비동기식 시스템 모듈을 이루는 DCVSL회로와 파이프라인 구조에 관해 설명한다. 4장에서는 비동기식 제산기의 설계를 pre-scale부, iteration step부, on-the-fly converter부로 나누어 설명하고, 5장과 6장에서는 설계 결과 및 결론을 기술한다.

## Ⅱ. NST 알고리즘

NST (new Svoboda-Tung) 알고리즘은 제수의 값에 관계없이 나머지 값만을 고려하여 연산을 수행할 수 있도록 고안된 제산 알고리즘이다. 식(1)은 이 알고리즘의 기본식이다.

$$R_{j+1} = R_j - q_j \cdot X \cdot r^{-(j+1)} \quad (1)$$

여기서 Rj(j=0,1,2,…)는 j번째 단계에서의 부분 나머지를 나타내며, r은 radix-index, $R_0$는 피제수, $q_j$는 몫 Q의 j번째 자리의 값을 나타낸다. 또한 연산수(N)가 $1.0 \le N < 2.0$인 범위를 사용하며, 제수는 1+X 형태로 표현하여 X의 값만을 연산에 사용한다. 이때 X는 $X < 1/r$로 한정한다.

표 1. Radix-2 NST 알고리즘
Table 1. Radix-2 NST algorithm.

| MSDs | operation |
|---|---|
| 00 | no operation |
| 01 | no operation |
| 0(-1) | no operation |
| 10 | subtraction |
| 11 | subtraction |
| 1(-1) | rewrite to 01 |
| (-1)0 | addition |
| (-1)1 | rewrite to 0(-1) |
| (-1)(-1) | addition |

Radix-2인 NST 알고리즘을 사용할 경우 나머지의 MSD(most significant digit) 2비트만을 고려하여 현재

division step에서의 몫과 연산을 결정하게 된다. <표 1>은 MSD 2비트와 연산과의 관계를 정리한 것이다. MSD의 상위 값이 '0'이면 no-operation이고, '1'이면 subtraction, '-1'이면 addition으로 결정된다. 단, MSD 2비트가 모두 '0'이 아니고 부호가 다를 경우에는 rewrite라는 연산과정을 거치며, 이때는 no-operation이 된다.

예제 1에 NST 알고리즘을 이용하여 피제수 $0.1001_{(2)}$와 제수 $1.0101_{(2)}$에 대한 제산 연산을 수행하는 과정을 나타낸다. 여기서 피제수와 부분 나머지에서 밑줄 친 부분은 MSD 2비트를 가리키며, n비트의 연산수에 대한 연산에서 몫은 최종 나머지의 상위 n비트가 된다.

예제 1) $0.\underline{10}01_{(2)} \div 1.0101_{(2)}$ X= $0.0101_{(2)}$

step 1: 0.1001-0.00101=0.1$\underline{0(-1)}$1(-1)

step 2: 0.10$\underline{(-1)1}$(-1)

step 3: 0.100$\underline{(-1)(-1)}$

step 4: 0.100(-1)(-1)+0.00000101

=0.100(-1)(-1)101

result = 0.100(-1)(-1) = 0.01101

remainder = 0.00000101

## Ⅲ. 비동기식 시스템 모듈

비동기식 시스템 모듈은 <그림 1>과 같이 데이터의 흐름을 제어하기 위한 두 레지스터(register), 즉 데이터를 송신하는 송신자(sender)와 데이터를 수신하는 수신자(receiver)로 구성된다. 이들간의 데이터 전송은 송신자에 입력 데이터가 도착했음을 알리는 request (Req)신호와 수신자에 출력 데이터가 저장되었음을 알리는 acknowledge (Ack)신호를 가지고, 이들 사이의 핸드쉐이크 프로토콜(handshake protocol)에 의해 이루어진다.

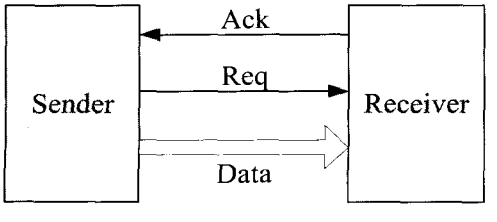



그림 1. 비동기식 시스템 모듈
Fig. 1. Asynchronous system module.

비동기식 시스템은 데이터 전송방식에 따라 단선(single-rail)에 의한 데이터 비트 다발과 지연소자를 이용하여 제어신호를 생성하는 다발 데이터(bundled data) 전송 방식과 한 비트의 데이터를 복선(dual-rail)으로 구성하고 이를 이용하여 제어신호를 생성하는 데이터 코딩 방식으로 나누어진다. 본 논문에서 사용되는 데이터 코딩 방식에 의한 DCVSL (differential cascode voltage switching logic) 비동기식 파이프라인은 다음과 같다.

### 1. DCVSL회로

DCVSL회로는 정로직과 부로직으로 구성되어 서로 반대되는 연산결과를 출력하도록 만들어진 회로이다. 서로 반대되는 출력을 갖는 특성으로 연산완료 신호의 생성이 용이하다는 장점을 가지지만, 하나의 기능블록을 정로직과 부로직으로 나누어 표현하기 때문에 하드웨어 오버헤드가 크다는 단점을 가지고 있다[7].

<그림 2>는 DCVSL 회로를 나타낸다. PCB는 DCVSL회로의 pre-charge신호로서, '0'일 경우에는 DCVSL회로가 pre-charge 상태에 있게되고, '1'의 값을 가질 경우에는 evaluation 상태에 있게 된다. Pre-charge 상태에서는 출력 단(NOT-게이트)의 입력에 해당하는 node에 전하가 충전되어 '1'의 값을 가지므로 DCVSL회로의 출력이 "00"의 무효(invalid) 데이터를 갖게 되며, 이를 spacer라고 부른다. Evaluation 상태는 회로가 동작을 시작하여 연산을 진행하는 상태로서 정로직과 부로직 중 하나의 로직이 충전된 전하를 방전시킴으로써 NOT-게이트의 출력을 '1'로 하여 DCVSL회로의 출력을 "10" 또는 "01"의 서로 다른 값을 갖게 한다.

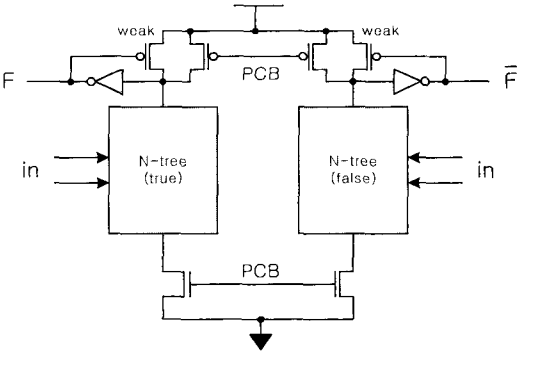



그림 2. DCVSL 회로도
Fig. 2. Circuit diagram of DCVSL.

본 DCVSL 회로는 출력단에 래치구조를 가지고 있어 연산결과를 저장하기 위한 별도의 래치가 필요없다는 장점을 갖는다.

2. DCVSL회로를 이용한 비동기식 파이프라인

<그림 3>은 DCVSL회로가 연산 수행을 위한 dual-rail회로로 사용된 데이터 코딩 방식의 비동기식 파이프라인 구조를 나타낸다. 이 파이프라인은 연산완료 신호를 생성하기 위한 연산완료 감지회로와 DCVSL 회로의 제어를 위한 C-소자로 구성되어 있다. 이러한 비동기식 파이프라인은 DCVSL회로의 출력 단에 데이터 저장을 위한 래치구조를 가지므로 데이터 저장을 위한 별도의 레지스터가 필요 없는 것이 특징이다.

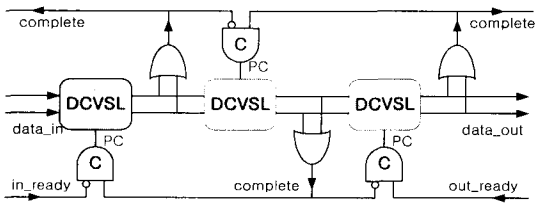



그림 3. DCVSL회로를 이용한 비동기식 파이프라인
Fig. 3. Asynchronous pipeline using DCVSL circuit.

Dada_in에 데이터가 도착하고 입력 데이터가 도착했음을 알리는 신호인 in_ready신호가 '1'이 되면, DCVSL회로의 pre-charge 신호인 PC신호는 '0'의 값을 갖게되고, DCVSL회로는 연산이 가능한 evaluation 상태가 된다. DCVSL회로에서 연산이 종료되면 서로 다른 값의 두 출력을 가지며, OR-게이트를 통해 이를 감지하여 연산완료 신호인 complete신호를 발생시킨다. 이때 발생된 complete신호는 외부로 전달되어 data_in에 다음 입력이 도착하도록 하며, 다음 단으로 전달되어 다음 단의 PC값을 '0'으로 만들면서 DCVSL회로가 현재 단의 출력을 받아서 연산을 시작할 수 있도록 한다. 다음 단의 연산이 종료되고 연산완료 신호가 발생하면, 이는 현재 단에 있는 C-소자의 입력으로 들어가 PC신호를 '1'로 만들어 DCVSL회로를 pre-charge 상태인 초기상태가 되게 한다.

## Ⅳ. 비동기식 제산기의 설계

본 논문에서는 RSD 수 표현 체계를 이용한 NST 알고리즘을 사용하고, 비동기식 파이프라인 구조를 이용하여 제산기를 구현한다. <그림 4>는 제산기의 전체 블록도로 pre-scale부, iteration step부, on-the-fly converter부의 세 부분으로 나누어진다. 각 부분은 연산완료 신호를 생성하여 다음 부분에서 연산이 시작되는 시기를 알려줌으로써 연산이 필요한 부분에서만 동작이 이루어지도록 되어있다. pre-scale부는 제수의 값을 일정범위로 한정시키기 위한 연산을 수행하며, iteration step부는 제산 연산에서 몫과 나머지 연산을 수행하고, on-the-fly converter부는 RSD 수 표현을 이진수 표현으로 바꿔주는 기능을 수행한다.

입력에 제수(divisor)와 피제수(dividend)가 도착한 후, start 신호가 발생하면서 연산이 시작되고, 3단의 파이프라인 구조인 pre-scale부, iteration step부, on-the-fly converter부를 차례로 거치면서 연산이 이루어지고 최종연산완료 신호인 done 신호와 함께 연산결과(result)를 출력하면서 제산연산을 마치도록 구성된다.

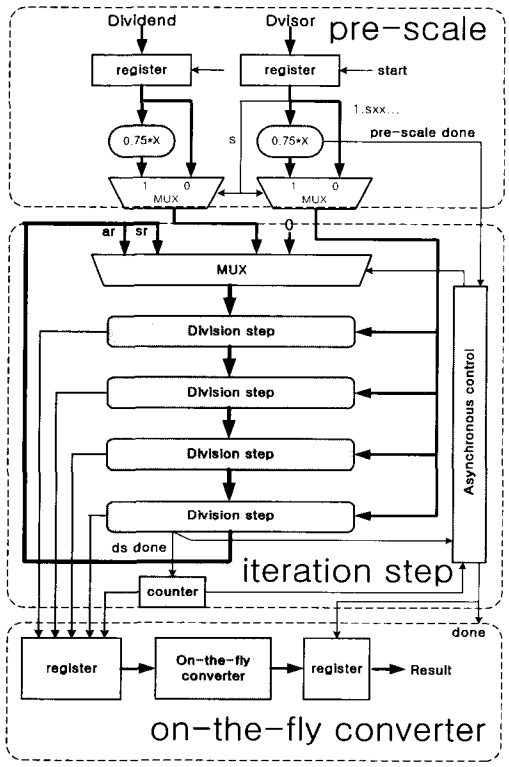



그림 4. 제산기 전체 블록도
Fig. 4. Block diagram of divider.

1. Pre-scale부

Pre-scale부에서는 제수의 값을 제한하기 위해 제수

와 피제수에 대해 제수의 값이 1.5보다 큰 값일 경우, 두 입력에 대해 0.75를 곱하여 제수를 1.5보다 작은 수로 만들어주는 기능을 수행한다.

<그림 5>는 pre-scale부의 피제수 연산부를 나타낸다. 16비트의 피제수입력을 저장하기 위한 레지스터인 REG16에 start신호에 의해 입력 데이터가 저장되면, 전용 감산기인 X075는 초기의 mode값을 결정하는 M_CON의 신호를 받아 감산연산을 수행한다. MUX32는 제수값에 따라 본래의 값과 pre-scale 연산을 거친 값 중 하나를 선택하여 출력하고, SX_GEN은 제수와 피제수의 X075 블록으로부터 발생되는 연산완료 신호와 제수값을 고려하여 다음 부분인 iteration step부에 연산의 시작 신호를 생성하면서 pre-scale부의 연산을 마친다.

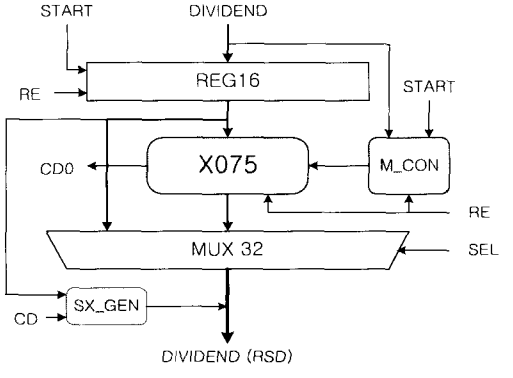



그림 5. Pre-scale부의 피제수 연산부
Fig. 5. Dividend calculation block of pre-scale block.

표 2. 제안한 감산연산 진리표
Table 2. Proposed truth table of subtraction.

| mode0 | | | | mode1 | | | |
|---|---|---|---|---|---|---|---|
| X | Y | Z | M | X | Y | Z | M |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

종전의 감산연산에서는 감산연산을 위한 2의 보수 형태로 변환과정이 필요하며, DCVSL형태의 구현으로는 많은 면적이 소요되는 문제점이 있다. 본 논문에서는 이를 위해 감산시 2의 보수 형태로의 변환이 필요 없고, 적은 면적으로 빠른 연산이 가능한 새로운 감산기를 이용하여 설계한다.

제안하는 감산기는 상위 bit에서 1을 빌려오는 경우와 그렇지 않은 경우를 각각 mode1과 mode0으로 하고 현재 단의 mode 입력에 대해 비트별 연산을 수행하여 결과를 얻는다. <표 2>는 제안된 감산기에 대한 진리표를 나타낸다.

감산연산(X-Y=Z)에 대해 X, Y는 입력을 나타내고 Z는 출력을 나타낸다. M은 다음 단으로 출력되는 mode의 값을 나타낸다.

<그림 6>은 1비트 감산기의 mode 값을 결정하는 m_sel과 감산연산결과를 출력하는 bit_op에 대한 회로이다. 따라서, 16비트 전용 감산기(X075)는 이러한 1비트 감산기 16개로 구현되며, 4비트마다 DCVSL회로로 만들어진 m_sel의 출력을 감지하여 연산완료 신호를 생성하도록 구현된다.

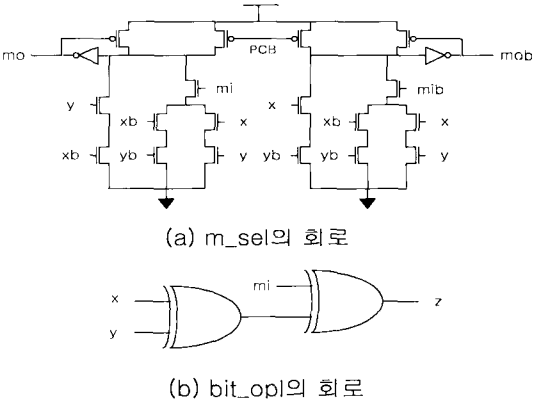



그림 6. 1비트 감산기
Fig. 6. 1-bit subtracter.

여기서, Xb, Yb 등은 X, Y 등의 신호와 반대되는 값을 가진 신호이며, Y는 본래의 16비트 연산수인 X의 0.25배수를 만들기 위해 오른쪽으로 두 번 자리이동한 수를 뜻한다.

제수에 대한 부분은 iteration step부에 대한 연산시작 신호의 생성이 필요없기 때문에 SX_GEN이 없는 것을 제외하고는 <그림 5>와 같은 구조를 가진다.

2. Iteration step부

제산 연산에서 몫과 나머지에 대한 연산을 수행하는 부분으로 비동기식 링 구조를 사용하여 구현하며, division step은 dual-rail과 single-rail의 혼합형태로

구현한다.

(1) 링 구조

링 구조는 제산 연산에서 필요로 하는 step을 모두 구현하지 않고 수 개의 step만을 구현하여 이를 반복하여 연산을 수행하도록 설계된다[8]. 본 논문에서는 링 구조를 비동기식으로 설계하므로써 동기식 링 구조와 달리, 데이터 저장을 위한 별도의 래치가 필요없도록 효율적으로 구현한다.

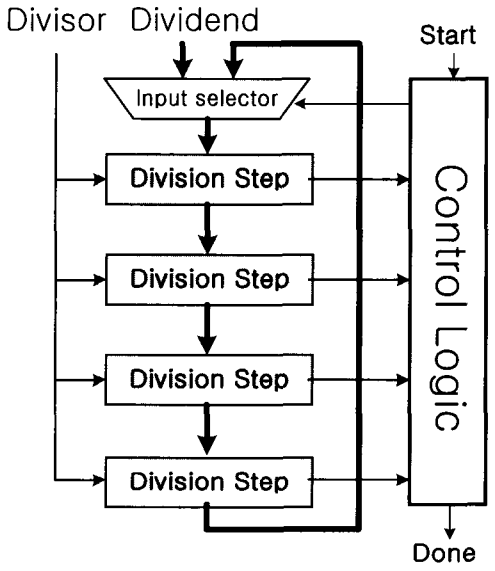



그림 7. 비동기식 링 구조
Fig. 7. Asynchronous ring structure.

<그림 7>은 4개의 division step을 가진 비동기식 링 구조이다. Input selector는 처음 연산에서는 피제수를 출력으로 선택하고 반복 연산중에는 부분 나머지를 출력으로 선택하는 기능을 수행한다. Start와 done은 각각 연산의 시작과 종료를 알리는 신호이다.

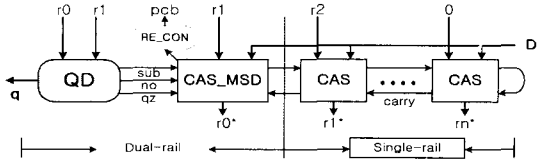



그림 8. Division step의 블록도
Fig. 8. Block diagram of division step.

(2) Division step

Division step은 여러번의 step을 통해 이루어지는 제산 연산중 한 step에 대한 몫과 부분 나머지를 구하는 기능을 수행하는 부분으로 <그림 8>과 같이 구성된다. 몫을 결정하기 위한 QD (quotient determination)와 부분 나머지를 결정하기 위한 CAS (controlled add/subtract), 제어신호를 생성하기 위한 RE_CON으로 이루어진다. 크리티컬 패스에 해당되는 CAS의 MSD (CAS_MSD)와 QD 부분만을 dual-rail 형태의 DCVSL로 구현하여 연산완료 신호를 생성하며, 나머지는 single-rail 형태로 CMOS 회로보다 제어가 용이한 도미노 로직(domino logic)을 사용하여 제산기의 효율성을 높인다.

입력 r0와 r1은 각각 RSD수로 표현된 partial remainder의 최상위비트와 그 아래 비트를 나타낸다.

① QD (quotient determination)의 설계

QD는 제산연산의 몫을 결정하고, CAS의 연산에서 필요한 제어신호를 발생시키는 부분으로 DCVSL 회로로 구성되어 연산완료 신호를 생성한다.

<그림 9>는 QD의 블록다이어그램을 나타낸다.

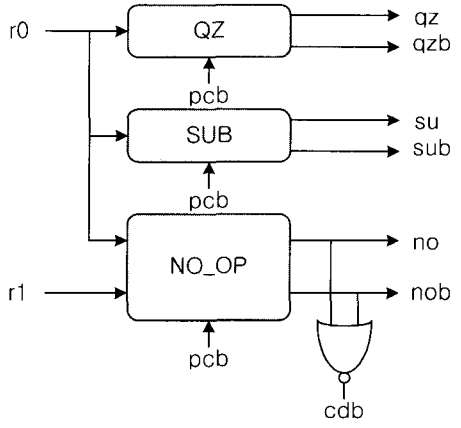



그림 9. QD의 블록도
Fig. 9. Block diagram of QD.

QZ는 몫이 0인지 아닌지를 판단하는 부분으로 몫이 0일 경우, 출력 qz의 값이 '1'이 된다. SUB는 감산연산의 유무를 판단하는 부분으로 감산연산이 발생할 경우, su의 값이 '1'이 된다. NO_OP는 가산/감산 연산이 일어나지 않는 no-operation을 판단하는 부분으로 no-operation이 발생할 경우, 출력 no값이 '1'이 된다. 또한 이 부분이 QD의 임계경로가 되므로 그 출력을

감지하여 QD의 연산완료신호(cdb)를 생성한다.

② CAS (controlled add/subtract)의 설계

CAS는 부분 나머지(partial remainder) 연산을 수행하는 부분으로 QD에서 발생하는 sub신호에 의해 입력값에 대한 가산 또는 감산 연산이 이루어지며, 크리티컬 패스에 대해 dual-rail 형태의 CAS_MSD를 사용하고 나머지 부분에 대해 single-rail 형태 CAS를 사용하므로써 하드웨어 오버헤드를 효율적으로 줄인다.

<그림 10>은 CAS의 최상위 비트인 CAS_MSD에 대한 블록도이다.

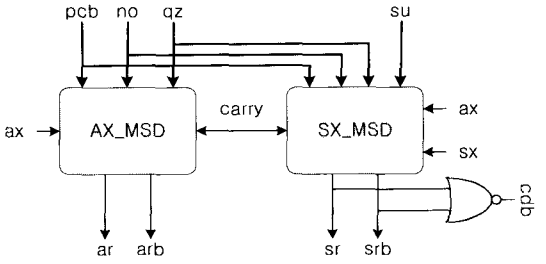



그림 10. CAS_MSD의 블록도
Fig. 10. Block diagram of CAS_MSD.

RSD수의 입력 ax, sx에 대해 AX_MSD와 SX_MSD는 가산 또는 감산연산을 수행하며, 각각 dual-rail 형태의 RSD수인 ar, arb, sr, srb를 결과로 출력한다. 그림에 나타난 신호들 중에 no-operation을 나타내는 no 신호, 몫이 '0'임을 나타내는 qz 신호, 감산연산을 나타내는 sub 신호는 QD로부터 받는 제어신호이며, carry는 이전 단으로부터 받는 carry신호이다. 이 연산부는 크리티컬 패스인 SX_MSD의 출력만을 이용하여 연산완료 신호를 생성하므로 불필요한 하드웨어 오버헤드를 줄인다.

<그림 11>은 CAS에 대한 블록도이다. AX_OP와 SX_OP는 CAS_MSD의 AX_MSD, SX_MSD와 같은 기능을 가지며, 면적이 작고 제어가 쉬운 single-rail 형태의 도미노 로직으로 구현되어 있다. CA_OP는 carry에 대한 연산을 수행하는 부분으로 dual-rail과 single-rail의 결합을 위해 CAS_MSD의 바로 아래 비트에 해당하는 부분은 DCVSL회로로 구현되며, 나머지는 도미노 로직으로 구현된다.

제어신호들이 나타내는 의미는 CAS_MSD의 신호와 같고, d는 각 비트별 연산에 대해 해당 비트에 대한 제수 입력을 뜻한다.

3. On-the-fly converter부

On-the-fly converter부는 RSD 수로 표시된 수를 보통의 이진수로 변환하는 기능을 수행하며, 상위 비트부터 연산이 이루어지는 on-the-fly 알고리즘을 사용하여 iteration step부와 병렬처리가 가능하기 때문에 고속의 연산처리가 가능하다는 장점을 갖는다.

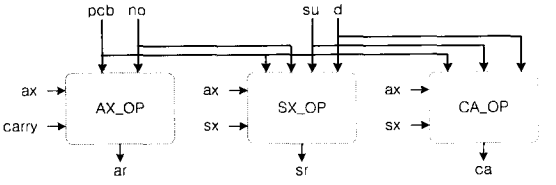



그림 11. CAS의 블록도
Fig. 11. Block diagram of CAS.

On-the-fly conversion은 <표 3>의 변환표에 의해 수행된다. X의 입력(X[k+1])이 0인 경우, A의 값(A[k+1])은 이전상태 A값(A[k])의 맨 끝자리에 0을 덧붙이고, B의 값(B[k+1])은 이전상태 B값(B[k])의 맨 끝자리에 1을 덧붙인다. 입력이 1인 경우, A의 값은 이전상태 A값의 맨 끝자리에 1을 덧붙이고, B의 값은 0을 덧붙인다. 입력이 -1인 경우, A의 값은 이전상태 B값의 맨 끝자리에 1을 덧붙이고, B의 값은 0을 덧붙인다. 연산의 결과는 최종 나머지의 값이 양수이면 A의 값이 되고, 음수이면 B의 값이 된다.

표 3. On-the-fly 변환표
Table 3. On-the-fly conversion table.

| X[k+1] | 0 | 1 | 1 |
|---|---|---|---|
| A[k+1] | (A[k],0) | (A[k],1) | (B[k],1) |
| B[k+1] | (B[k],1) | (A[k],0) | (B[k],0) |

예제 2) $10(-1)(-1)110_{(2)} \Rightarrow 0101110_{(2)}$

| 입력 | A[k] | B[k] |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 10 | 01 |
| -1 | 011 | 010 |
| -1 | 0101 | 0100 |
| 1 | 01011 | 01010 |
| 1 | 010111 | 010110 |
| 0 | 0101110 | 0101100 |

예제 2는 이 알고리즘을 사용하여 RSD 수로 표현된 $10(-1)(-1)110_{(2)}$을 이진수 표현으로 변환하는 과정을 나타낸다.

<그림 12>는 on-the-fly converter의 블록도를 나타낸다. $q_i$는 RSD수로 표현된 1비트의 몫을 뜻하고($q_0$가 MSD), D_CELL은 입력에 따라 A와 B에 각각 덧붙여지는 수를 결정하는 부분이고, M_CELL은 A와 B에 저장된 값을 입력에 따라 이동시키는 역할을 수행하는 부분이다.

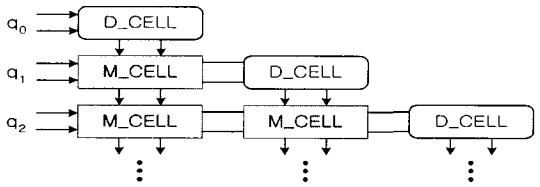



그림 12. On-the-fly converter의 블록도
Fig. 12. Block diagram of on-the-fly converter.

<그림 13>은 M_CELL과 D_CELL의 내부회로를 나타낸다.

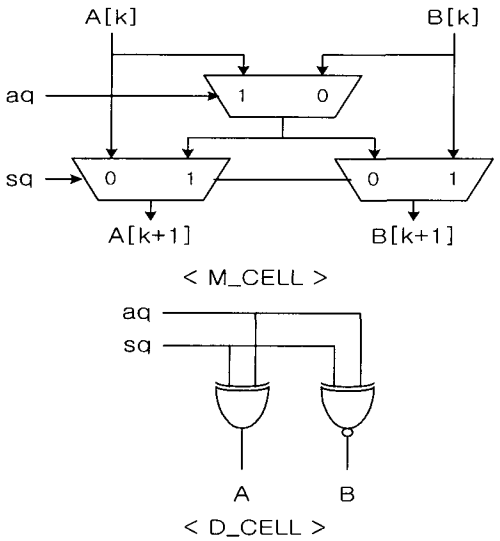



그림 13. On-the-fly converter의 회로도
Fig. 13. Circuit diagram of on-the-fly converter.

## V. 설계 결과

제안된 구현방법을 이용하여 full-custom 방식으로 제산기를 설계하였으며, Hynix 0.6㎛ 공정 라이브러리를 이용하여 HSPICE에서 회로 시뮬레이션을 하였다. <그림 14>는 제어신호에 대한 시뮬레이션에 대한 결과를 나타낸다.

<그림 4>와 <그림 14(a)>의 pre-scale부는 피제수와 제수의 입력에 대해 외부의 ①신호(start)를 받아 pre-scale연산을 시작하며, 연산을 수행하는 감산기로부터 각각 피제수와 제수에 대한 연산이 끝났음을 알리는 연산완료 ②신호(pre-scale done)를 생성하고 있다. 연산완료 ②신호(pre-scale done)는 iteration step부의 연산 시작 신호이기도 하다. <그림 14(b)>의 iteration step부는 비동기식 링 구조의 마지막 division step에서 발생하는 연산완료 ③신호(ds done)와 이 신호를 받아서 반복연산 회수(C1C0)를 카운트하는 카운터의 출력에 대한 결과 파형을 나타내고 있으며, 이들로부터 총 4번의 반복연산이 이루어지고 있음을 알 수 있다. <그림 4>의 iteration step부 마지막 단의 Division step에서 ③신호(ds done)가 발생되고, 이 신호를 아래 카운터의 clock으로 입력하여 카운터를 동작시킨다. 카운터의 출력 신호(C1C0)가 "11"이 되는 ④신호일 때, asynchronous control 블록에서 최종 연산완료를 알리는 ⑤신호(done)를 발생한다. <그림 14(c)>의 on-the-fly converter부는 최종연산완료를 알리는 ⑤신호(done)에 의해 이진수로 표현된 연산값

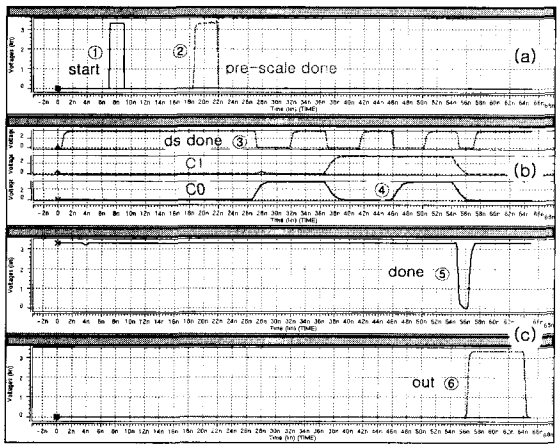



그림 14. Hspice 시뮬레이션 결과
(a) Pre-scale 블록,
(b) Iteration step 블록,
(c) On-the-fly converter 블록
Fig. 14. Results of Hspice simulation.
(a) Pre-scale block,
(b) Iteration step block,
(c) On-the-fly converter block.

⑥(out)을 출력하고 있다. 이와 같은 시뮬레이션 결과 파형을 통해 설계된 제산기가 올바로 동작하는 것을 확인할 수 있다.

<표 4>는 3.3V 공급전압에서의 블록별 지연시간을 나타낸다. 평균지연 (average)시간은 pre-scale 부는 2.6ns, iteration step 부는 37.7ns, on-the-fly converter부는 1.4ns를 가지고, 제산기의 총 평균지연시간은 41.7ns이다. 표에 나타난 worst-case는 지연변동율이 가장 큰 pre-scale부가 carry의 전파로 인해 나타나는 최대지연시간이며, best-case는 제수로 1의 값을 가질 경우에 나타나는 최소지연 시간이다. 또한 average는 worst-case와 best-case를 포함한 random 데이터에 대한 20회의 시뮬레이션을 거친 평균지연시간이며, 특히 pre-scale부의 평균지연시간에 대한 결과는 pre-scale 연산이 실행될 경우와 그렇지 않을 경우를 각각 50%로 하여 산출한 결과이다.

표 4. 블록별 지연시간
Table 4. Delay times of blocks.

| | best-case | worst-case | average |
|---|---|---|---|
| pre-scale | 1.0ns | 9.2ns | 2.6ns |
| iteration | 36.9ns | 37.5ns | 37.7ns |
| on-the-fly | 1.4ns | 1.4ns | 1.4ns |
| total | 39.3ns | 48.1ns | 41.7ns |

<그림 15>는 본 논문에서 제안한 제산기의 core부분에 대한 레이아웃 설계결과를 나타낸 것으로 1,480 × 1,200㎛²의 크기를 갖는다.

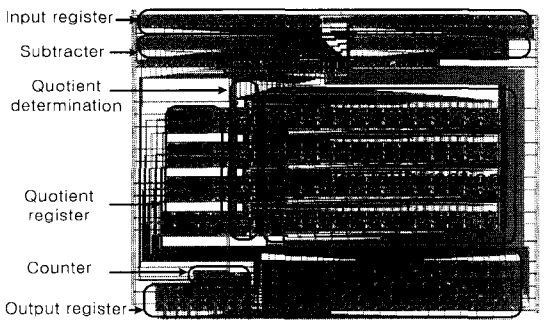



그림 15. 제산기 레이아웃
Fig. 15. Layout of divider

Core부분에 사용된 트랜지스터의 총 개수는 12,956개 이며, division step을 dual-rail로만 설계하였을 경우에 소요된 트랜지스터 수는 16,216개 이다. 본 설계에서 사용된 방법인 혼합회로(dual-rail+single-rail)에 의한 설계는 dual-rail로만 설계된 회로에 비해 약 20% 면적감소 되었다.

표 5. 혼합회로에 대한 성능 비교
Table 5. Comparision of mixed-circuit peformance.

| | 트랜지스터 수 |
|---|---|
| 혼합회로 | 12,956 개 |
| Dual-rail회로 | 16,216 개 |

<표 6>은 Carry-skip adder를 사용한 감산기와 본 설계의 Pre-scale부에 사용된 전용 감산기와의 비교이다. 트랜지스터 수가 1,261개에서 608개로 감소하면서 면적이 약 50% 감소되었고, 최대지연시간이 11.7ns에서 9.2ns로 단축되면서 약 20%의 성능향상이 되었다.

표 6. 제안한 감산기에 대한 성능 비교
Table 6. Comparisions of proposed subtracter performance.

| | 트랜지스터 수 | 최대지연시간 |
|---|---|---|
| 전용 감산기 | 608 개 | 9.2 ns |
| Carry-skip adder 를 사용한 감산기 | 1,261 개 | 11.7 ns |

## Ⅵ. 결 론

본 논문에서는 RSD 수 표현 체계를 적용한 radix-2 NST 알고리즘과 비동기식 파이프라인 구조를 이용하여 16비트 비동기식 제산기를 설계하였다. 제산기는 전용 감산기를 사용한 pre-scale부와 4개의 division step을 갖는 비동기식 링 구조를 사용한 iteration step부, 그리고 on-the-fly 알고리즘을 이용한 on-the-fly converter부로 구성되었으며, 각 블록은 연산 시작 신호와 연산 완료신호를 갖는 비동기식 설계방법을 이용하여 설계되었다.

본 제산기는 비동기식 제어방법과 전용 감산기, 비동기식 링 구조, dual-rail과 single-rail의 혼합회로 등의 효율적인 설계기법들을 사용함으로써 제산기의 면적을 줄이면서 성능을 향상시킬 수 있도록 설계되었다. 설계된 제산기는 12,956개의 트랜지스터를 사용하여

Hynix 0.6㎛ 공정에서 1,480 × 1,200㎛²의 core면적을 가지며, HSPICE 시뮬레이션 결과, 3.3V의 공급전압에서 41.7ns의 평균 지연시간을 가졌다. 이 결과는 dual-rail회로만으로 구성된 제산기와 비교했을 때, 약 20%의 면적 감소를 가져온 것이며, pre-scale 블록에서 사용된 전용 감산기는 carry-skip adder를 이용한 감산기와 비교할 때, 약 50%의 면적 감소와 20%의 성능향상을 가져왔다.

## 참 고 문 헌


[1] Scott Hauck, "Asynchronous design methodologies : an overview," *Proceedings of the IEEE*, Vol. 83, No. 1, pp. 69~93, Jan. 1995.

[2] Al Davis and Steven M. Nowick, "An introduction to asynchronous circuit design," *Technical Report UUCS-97-013*, Department of Computer Science, University of Utah, pp. 1~57, sept. 1997.

[3] M. Suzuoki, et al., "A microprocessor with a 128-bit CPU, ten floating-point MAC's, four floating-point dividers, and an MPEG-2 decoder," *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 11, pp. 1608~1618, Nov. 1999.

[4] S. F. Obermann and M.J. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers*, Vol. 46, No. 8, pp. 833~854, Aug. 1997.

[5] Israel Koren, *Computer arithmetic algorithm*, Prentice-Hall, Inc., New Jersey, pp. 127~133, 1993.

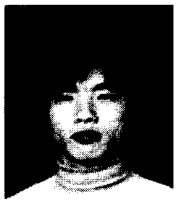
[6] L. A. Montalvo, K.K. Parhi, and A. Guyot, "New Svoboda-Tung division," *IEEE Transactions on Computers*, Vol. 47, No. 9, pp. 1014~1020, Sept. 1998.

[7] D. Somasekhar and K. Roy, "Differential current switch logic: a low power DCVS logic family," *IEEE Journal of Solid- State Circuits*, Vol. 31, No. 7, pp. 981~991, July 1996.

[8] T. E. Williams and M.A. Horowitz, "A zero-overhead self-timed 160-ns 54-b CMOS divider," *IEEE Journal of Solid -State Circuits*, Vol. 26, No. 11, pp. 1651~1661, Nov. 1991.

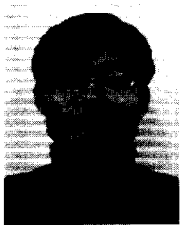
[9] N. Burgess, "A fast division algorithm for VLSI," *Proceedings of ICCD '91*, pp. 560~563, Oct. 1991.


## 저 자 소 개

**李 祐 錫(正會員)**

2000년 2월 충북대학교 전기전자공학부 졸업 (공학사). 2002년 2월 충북대학교 반도체공학과 졸업(공학석사). 2002년 4월-현재 @Lab 연구원. <주관심분야 : 연산회로설계>



**朴 錫 在(正會員)**

2002년 2월 충북대학교 전기전자공학부 졸업 (공학사). 2003년 3월-현재 충북대학교 반도체공학 석사과정. <주관심분야 : VLSI설계, GALS system설계>

**崔 淏 鎔(正會員)**

1980년 2월 서울대학교 전자공학과 졸업 (공학사). 1982년 2월 한국과학기술원 전기및전자공학과 졸업 (공학석사). 1994년 3월 Osaka University 졸업 (공학박사). 1982년-1985년 삼성반도체 연구소 연구원. 1985년-1996년 부경대학교 부교수. 1996년-현재 충북대학교 전기전자및컴퓨터공학부 교수. <주관심분야 : VLSI설계, VLSI테스트, 비동기회로 및 시스템 설계/테스트>