論文99-36C-3-3

새로운 Bit-serial 방식의 곱셈기 및 나눗셈기 아키텍처 설계

(Design of a New Bit-serial Multiplier/Divider Architecture)

邑 壽 煥 *, 鮮 于 明 勳 *

(Soohwan Ong and Myung Hoon Sunwoo)

요 약

본 논문에서는 기존의 bit-serial 방식 곱셈기 및 나눗셈기의 하드웨어 부담을 줄이고 동일한 연산 사이클 수를 갖는 새로운 bit-serial 방식의 곱셈기 및 나눗셈기 아키텍처를 제안한다. 제안하는 bit-serial 곱셈기 및 나눗셈기 아키텍처는 부분곱 또는 부분나머지를 구하기 위해 레지스터 및 가감산기의 비트 수를 2배 확장하지 않기 때문에 기존의 아키텍처에 비해 하드웨어의 부담을 줄였다. 또한 덧셈/뺄셈과 Shift 연산을 동시에 수행하므로써 N 비트 곱셈 및 나눗셈 연산에 각각 N, $N+2$ 사이클을 소모하며 이는 기존의 아키텍처와 동일한 연산 사이클 수를 지원한다. 제안하는 bit-serial 곱셈기 및 나눗셈기 아키텍처는 SliM Image Processor에 적용하여 실제 칩으로 구현하였으며 그 성능을 입증하였다.

Abstract

This paper proposes a new bit-serial multiplier/divider architecture to reduce the hardware complexity significantly and to maintain the same number of cycles compared with existing architectures. Since the proposed bit-serial multiplier/divider architecture does not extend the number of bits in registers and an adder/subtractor to calculate a partial product or a partial remainder, the hardware overhead can be greatly reduced. In addition, the proposed architecture can perform an addition/subtraction and a shift operation in parallel and the number of cycles for N-bit multiplication and division for the proposed circuits is N and $N+2$, repectively. Thus, the number of cycles for multiplication and division is the same compared with existing architectures. The SliM Image Processor employs the proposed multiplier/divider architecture and proves the performance of the proposed architecture.

I. 서 론

최근 VLSI 칩의 집적도가 높아짐에 따라 곱셈기나 나눗셈기를 하나의 VLSI 칩으로 구현하는 경우가 많으나 곱셈기나 나눗셈기는 Die 면적을 많이 차지하기 때문에 칩 가격이 높아지는 단점이 있다. 신호 및 영상처리용 칩[1-3], SIMD 프로세서[4-8] 등과 같이 다수의 곱셈기 또는 나눗셈기를 필요로 하는 경우에는 이들을 집적시키기 어렵다. 특히 SIMD 프로세서의 경우 성능 향상을 위해 많은 수의 PE를 하나의 칩으로 집적시켜야 하기 때문에 곱셈기나 나눗셈기를 포함하고 있는 경우가 거의 없다[4-8]. 따라서 Die 면적을 줄이기 위해 다양한 bit-serial 방식의 곱셈기들이 개발되고 있다[9-11]. 그러나 이러한 곱셈기들은 상대적으로 하드웨어 부담이 큰 단점이 있다. 기존의 bit-serial 방식 곱셈기로는 Quasi-Serial 곱셈기[9-10

* 正會員, 亞州大學校 電機電子工學府
(School of Electronical and Electronic Eng., Ajou Univ.)

接受日字:1998年10月26日, 수정완료일:1999年3月2日

]와 COPE(COProcessor Element) 아키텍쳐[11]가 있다.

Qusasi-Serial 곱셈기[9-10]는 먼저 $2N$ - 1 비트 Shift 레지스터의 MSB 부분의 N 비트에 피승수를 저장하고 나머지 LSB 부분에 "000...0"을 저장한다. $2N$ - 1 비트 Shift 레지스터의 하위 N 비트와 승수를 AND 시킨 후 그 결과에서 '1'의 개수를 Count 한다. 다음 사이클에서 피승수가 저장된 $2N$ - 1 비트 Shift 레지스터를 Shift Right시키고 다시 같은 형태로 AND 시킨 후 1의 개수를 Count 한다. 이러한 방식으로 각 자리수로 연산된 부분곱들은 동일한 자리수에 해당하는 부분의 '1'의 개수를 Count하여 더해주므로써 곱셈 연산을 수행한다. 따라서 N-비트 곱셈인 경우 N번의 Shift 사이클과 N번의 Count 사이클이 소모되어 총 $2N$ 사이클이 소모되며 상대적으로 연산 속도가 느리다. 또한 나눗셈을 처리할 수 없으며 15-비트 이하의 곱셈 연산시 덧셈기를 이용한 일반적인 곱셈기에 비해 속도가 느린 단점을 가지고 있다[9-10].

COPE 아키텍쳐[11]는 SIMD 프로세서에서 곱셈 연산을 가속화 시키는 Co-processor의 형태로 개발되었다. COPE 아키텍쳐는 기본적으로 Add-and-Shift 연산과 Sub-and-Shift 연산을 이용하여 곱셈 및 나눗셈 연산을 수행한다. COPE 아키텍쳐는 N 비트 곱셈에 대해 피승수(젯수)를 bit-serial 방식으로 입력하기 위한 N 번의 Shift 연산과 N 번의 Add-and-Shift 연산이 필요하기 때문에 총 연산수는 $2N$ 사이클이다. 나눗셈 연산은 피젯수를 2의 보수를 취하여 더하는 것과 Sub-and-Shift 연산 후 한 번의 수정 과정이 필요한 외에는 곱셈 연산 방식과 유사하며 N 비트 데이타의 나눗셈 연산은 $2N$ + 1 사이클이 소모된다. 만약 피승수(젯수)를 Bit-parallel로 입력한다면 곱셈과 나눗셈은 각각 N, N + 1 사이클이 소모된다. 그러나 이 방법은 부분곱과의 연산을 위해 피승수 및 젯수의 비트 수를 2배 확장하여 연산하기 때문에 N 비트 데이타의 곱셈 및 나눗셈 연산을 수행하기 위해서는 N 비트 Shift 레지스터 1개와 $2N$ 비트 레지스터 2개, 그리고 하나의 $2N$ 비트 덧셈기가 필요하므로 하드웨어 부담이 크다.

본 논문에서는 상대적으로 하드웨어 부담이 적은 새로운 bit-serial 방식의 곱셈기 및 나눗셈기 아키텍쳐를 제안한다. 제안하는 곱셈기 및 나눗셈기 아키텍쳐는 범용 RISC 프로세서 또는 SIMD 프로세서에 적용이 용이하며 곱셈과 나눗셈 연산이 모두 가능하기 때문에 기존의 전용 곱셈기와는 차이가 있다. 또한 비트 수를 2배 확장할 필요가 없기 때문에 COPE 아키텍쳐[11]에 비해 하드웨어의 크기를 줄이는 동시에 COPE 아키텍쳐와 동일한 연산 수로 곱셈 및 나눗셈 연산이 가능하다.

본 논문은 다음과 같이 구성된다. 2장에서는 제안하는 bit-serial 방식 곱셈기 및 나눗셈기 아키텍쳐에 대해 기술하고 3장에서는 제안하는 아키텍쳐에 적용되는 곱셈 및 나눗셈 연산 알고리즘에 대해 기술한다. 4장에서는 성능 평가를 하고 끝으로 5장에서는 결론을 맺는다.

Ⅱ. 제안하는 bit-serial 방식 곱셈기 및 나눗셈기 아키텍쳐

일반적으로 곱셈기 및 나눗셈기는 VLSI 면적을 많이 차지하기 때문에 많은 수의 곱셈기 및 나눗셈기를 하나의 칩으로 집적시키는 것은 고가, 고전력의 원인이 된다. 따라서 다수의 곱셈기 및 나눗셈기를 요하는 신호 및 영상처리용 칩, SIMD 프로세서 등의 대부분은 전용의 곱셈기 및 나눗셈기를 사용하지 않고 하드웨어 크기를 줄인 bit-serial 방식의 곱셈기 및 나눗셈기를 사용한다. 이 장에서는 기존의 bit-serial 방식 곱셈기 및 나눗셈기에 비해 하드웨어 크기를 줄인 새로운 bit-serial 방식 곱셈기 및 나눗셈기 아키텍쳐에 대해 기술한다.

먼저 기존의 곱셈기 및 나눗셈기 아키텍쳐인 COPE 아키텍쳐[11]에 대해 살펴본다. COPE 아키텍쳐[11]는 SIMD 프로세서인 DAP[4]과 GAPP[5]에 적용되는 co-processor로 개발되었으며 그림 1은 COPE의 아키텍쳐를 나타낸다. N 비트 Shift 레지스터인 MQ(Multiplicand-Quotient) 레지스터는 피승수 또는 젯수를 저장하기 위한 레지스터이며 2의 보수기는 전가산기를 이용하여 뺄셈 연산을 수행하기 위한 것이다. $2N$ 비트 L-S 레지스터는 부호가 확장된 피승수 또는 제수를 저장하며 $2N$ 비트 전가산기는 $2N$ 비트 Accumulator에 저장된 부분곱 또는 부분나머지의 값과 피승수 또는 젯수(L-S 레지스터)를 더하여 또다른 부분곱(부분나머지)를 구한다. $2N$ 비트 2 x 1 MUX는 덧셈을 하기 전에 부분곱(부분나머지)의 자리 수를 맞추기 위한 Shift 연산을 수행한다. 이 외에도 최종

결과를 출력하기 위한 부호 회로와 제어 회로로 구성되어 있다. COPE 아키텍처는 N 비트 데이타의 곱셈 연산에 N 사이클이 필요하며 나눗셈 연산에 $N+1$ 사이클이 필요하다.

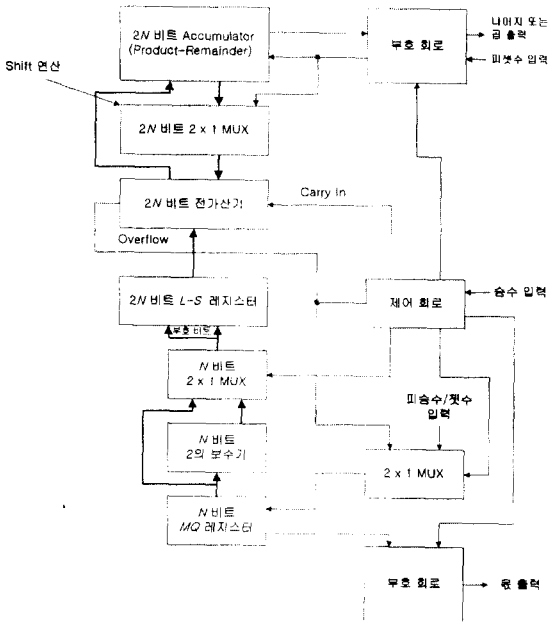


그림 1. COPE 아키텍처
Fig. 1. The Architecture of COPE.

그림 1에서 알 수 있듯이 COPE 아키텍처에서 필요한 하드웨어는 N 비트 Shift 레지스터, 2의 보수기, $2N$ 비트 레지스터 2개, $2N$ 비트 전가산기, N 비트 2 x 1 MUX, $2N$ 비트 2 x 1 MUX와 부호 회로 및 제어 회로로 구성되어 있다. 특히 승수의 비트 수를 2배 확장하기 때문에 레지스터 및 가산기의 비트 수가 $2N$으로 증가하여 하드웨어 부담이 크다.

제안하는 bit-serial 방식의 곱셈기 및 나눗셈기는 승수 및 젯수의 비트 수를 확장하지 않고 곱셈 및 나눗셈 연산이 가능하다. 그림 2는 제안하는 곱셈기 및 나눗셈기의 아키텍처를 나타낸다. 그림에서 보는 바와 같이 제안하는 아키텍처는 N 비트 레지스터(MQ), $2N$ 비트 ACC(Accumulator), N 비트 가감산기, N 비트 2 x 1 MUX인 Condition MUX, 음의 최소값 검출기 및 제어 회로로 구성된다. N 비트 레지스터와 ACC의 상위 N 비트인 $ACCH$는 N 비트 가감산기의 입력이 되고 가감산기의 출력은 Condition MUX를 거쳐 $ACCH$로 입력된다.

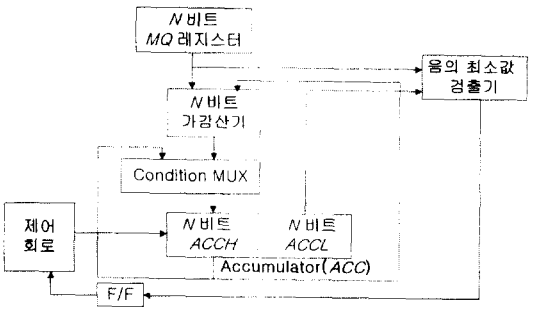


그림 2. 제안하는 아키텍처
Fig. 2. The Proposed Architecture.

$2N$ 비트 ACC는 두개의 N 비트 레지스터로 구성되며 상위 N 비트 레지스터인 $ACCH$는 데이타를 Load하면서 동시에 Shift하는 레지스터이며 하위 N 비트 레지스터인 $ACCL$은 일반 Shift 레지스터로 승수 또는 피젯수가 입력된다. 이러한 구조는 N 비트 프로세서에서 $2N$ 비트를 처리하는 Double Precision 연산을 할 경우 매우 유용하게 사용될 수 있다. Condition MUX는 제어 회로에 의해 제어받으며 $ACCH$의 입력으로 가감산기의 출력과 $ACCH$로부터 궤환되는 값 중 하나를 선택하여 출력한다. 제어 회로는 곱셈의 경우 ACC의 LSB, 나눗셈의 경우 가감산기의 부호 비트를 참조하여 Condition MUX의 선택 신호 및 Shift 연산의 종류를 결정한다.

음의 최소값끼리 연산을 할 경우 제안하는 아키텍처 상에서 부호 비트의 오류가 발생한다. 이를 막아주기 위해 음의 최소값 검출기가 필요하다. 음의 최소값 검출기는 N개의 NOR 게이트, N 입력 AND 게이트를 사용하여 피승수와 승수가 모두 음의 최소값인 경우를 검출한다. 음의 최소값끼리의 연산인 경우 곱셈 연산의 마지막 단계 Shift 연산의 종류를 제어한다. 이에 대한 자세한 설명은 다음 장에 나와있다.

COPE 아키텍처와 달리 제안하는 아키텍처는 승수 또는 젯수의 비트 수를 2배 확장할 필요가 없기 때문에 하드웨어 부담을 크게 줄일 수 있다. 또한 N 비트 곱셈 및 나눗셈을 위해 각각 N, $N+2$ 사이클이 필요하므로 연산 수는 거의 동일하다. 또한 제안하는 아키텍처는 일반 프로세서의 ALU 연산 구조를 조금 변형시킨 구조이기 때문에 간단한 MUX와 제어 회로를 일부 추가하므로써 SIMD 프로세서나 범용 마이크로프로세서에 적용이 가능하다. 실제로 SIMD 프로세서인 SliM Image Processor [12-14]에 제안하는 아키텍

쳐를 적용하여 실제 칩으로 구현하였으며 그 성능을 입증하였다.

Ⅲ. 제안하는 아키텍쳐 상의 곱셈 및 나눗셈 연산 알고리즘

이 장에서는 제안하는 아키텍쳐 상에서의 곱셈 및 나눗셈 연산 알고리즘에 대해 기술한다. 곱셈은 Add-and-Shift 연산, 나눗셈은 Sub-and-Shift 연산을 기본으로 하였으며 하드웨어의 크기를 줄이기 위해 곱셈 및 나눗셈 연산 알고리즘을 개선시켰다.

1. 곱셈 알고리즘

일반적인 Add-and-Shift 곱셈 연산 알고리즘[15]은 승수의 LSB부터 MSB까지를 한 비트씩 점검하여 부분곱을 생성한다. 만약 승수의 비트가 '1'이면 덧셈을 수행하고 '0'이면 덧셈을 수행하지 않는다. 덧셈이 끝난 후 자리수를 맞추어 주기 위해 Shift 연산이 필요하다. 부호 비트에 대한 연산은 Correction Step[15]으로 명칭되며 다른 비트들과는 다른 연산을 필요로 한다. 만약 승수와 피승수가 모두 음수일 경우, 덧셈 대신 뺄셈이 수행되며, 피승수가 양수이고 승수가 음수이면 피승수에 대해 1의 보수를 취한 후 덧셈을 수행한다.

Quasi-Serial 곱셈기는 동일한 자리수에 해당하는 1의 개수를 Count하는 방식이며 COPE 아키텍쳐는 부호를 확장하는 Add-and-Shift 알고리즘을 사용하였다. 제안하는 아키텍쳐는 하드웨어 부담을 줄이기 위해 Add-and-Shift 곱셈 연산 알고리즘을 개선하여 적용하였다. 개선된 알고리즘의 특징은 Shift 연산을 수행할 때 실제로는 부호를 확장하지 않으면서 부호 확장된 효과를 거둘 수 있도록 하는 것이다. 승수 $X = x_{n-1}x_{n-2}...x_0$와 피승수 M을 곱한 결과인 P는 식 (1)과 같이 표현된다.

$$P = M \cdot x_0 + M \cdot x_1 \cdot 2 + M \cdot x_2 \cdot 2^2 + \ldots - M \cdot x_{n-1} \cdot 2^{n-1} \quad (1)$$

X의 부호 비트는 X의 다른 비트와 달리 덧셈 대신 뺄셈을 수행하여 승수가 음수일 경우에 발생하는 연산 오류를 정정한다[15]. 식 (1)에서 승수 M이 음수인 경우 부분곱 $M \cdot x_i$는 음수여야 한다. 따라서 자리수가 낮은 부분곱과 높은 자리수의 부분곱을 연산하기 위해 자리수가 낮은 부분곱의 부호가 확장되어야 한다. 부호를 확장할 경우 하드웨어 크기가 두배 커지기 때문에 부분곱간의 자리수를 맞추어 주는 Shift 연산을 제어하므로써 하드웨어 크기를 줄일 수 있다.

부분곱의 부호는 전적으로 피승수의 부호에 의존하므로 피승수가 양수일 경우 LSR을 수행하여 부분곱을 항상 양수로 유지시킨다. 피승수가 음수인 경우 Carry가 발생하게 되면 Shift 연산을 LSR로 수행하여 Carry를 입력받아 부분곱의 부호를 '1'로 유지하고 Carry가 발생하지 않는 경우에는 ASR을 수행하여 부호를 유지시킨다. 따라서 부분곱을 2배로 부호 확장하지 않고 조건적인 연산을 통해 동일한 연산 결과를 얻을 수 있다. 이를 제안한 아키텍쳐에 적용하기 위한 개선된 곱셈 알고리즘의 flow 차트는 그림 3에 나와 있다.

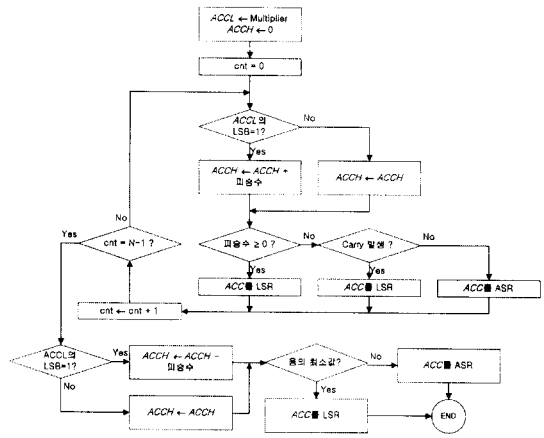


그림 3. 개선된 곱셈 알고리즘
Fig. 3. The Enhanced Multiplication Algorithm.

먼저 $ACCH$는 Clear되어 있고 $ACCL$에 승수가 저장되어 있으며 MQ 레지스터에 피승수가 저장되어 있다고 가정한다. 첫번째 부분곱을 만들기 위해 승수의 LSB인 $ACCL$의 LSB를 점검하여 '1'이면 $ACCH$와 MQ를 더하고 '0'이면 더하지 않는다. 이 값은 피승수의 부호에 따라 양수이면 ACC를 LSR(logical shift right) 연산을 수행하고, 음수이면 Carry의 생성 여부에 따라 LSR 또는 ASR(arithmetic shift right) 연산을 수행하여 ACC에 저장한다. 피승수의 값이 양수이면 부분곱이 음수가 될 수 없으므로 LSR을 수행하여 부호 비트가 항시 '0'이 되도록 유지한다. 피승수가 음수이면 부분곱은 항시 음수이기 때문에 부호 비트를 '1'로 유지해야 한다. 따라서 Carry가 발생

하면 LSR을, 발생하지 않는 경우는 ASR을 수행하여 부호 비트를 '1'로 유지한다. 이와같이 피승수의 부호와 Carry에 따른 조건적인 연산은 피승수를 부호 확장하지 않고 부분곱의 부호를 올바로 계산할 수 있기 때문에 피승수와 덧셈기의 비트 수를 2배 확장하지 않아도 된다. 따라서 하드웨어 부담을 감소시킨다. *ACCH*는 가감산기의 출력을 Shift시키며 Load할 수 있으므로 덧셈 연산과 Shift 연산은 동시에 수행된다.

이러한 과정을 N - 1번 수행한 후 승수의 부호에 대한 연산을 수행한다. 첫번째 과정부터 N - 2번째 과정까지는 피승수의 부호를 유지하며 부분곱을 생성하였기 때문에 승수가 양수인 경우에는 ASR 연산을 통해 부호만 유지하면 옳은 결과를 얻을 수 있다. 그러나 승수가 음수인 경우에는 뺄셈을 수행하여 부호를 보정한 후 ASR 연산을 수행해야 한다. 예외적인 경우로 피승수와 승수가 모두 음의 최소값을 갖는 경우 뺄셈의 연산 결과가 '100...0'이 된다. 이때 ASR을 하면 곱셈 결과가 음수가 되는 오류를 발생한다. 이를 제어하기 위해 음의 최소값 검출기는 승수와 피승수가 모두 음의 최소값인지를 검출하여 이러한 경우에는 ASR 대신 LSR 연산을 수행하도록 제어한다. N 사이클 후 곱셈 결과의 상위 N 비트는 *ACCH*에, 하위 N 비트는 *ACCL*에 각각 저장된다.

그림 4는 제안한 알고리즘을 이용한 4 비트 곱셈의 예를 보여준다. 피승수 '-6'은 *MQ* 레지스터에 저장되어 있고 승수 '-5'는 *ACCL*에 저장되어 있다. 첫번째 사이클에서 *ACCL*의 LSB가 '1'이므로 피승수는 *ACCH*와 더해진다. Carry가 발생하지 않았으므로 ASR 연산이 수행되어 *ACC*에 저장된다. *ACCL*의 LSB에는 승수의 LSB에서 두번째 비트가 저장된다. 두번째 싸이클에서 *ACCL*의 LSB가 '1'이므로 덧셈이 수행되며 이때 Carry가 발생한다. 이는 *ACC*의 부호 비트가 바뀌었음을 의미하므로 부호를 유지하기 위해 Carry를 Shift 입력으로 받는 LSR 연산을 수행하여 *ACC*에 저장한다. 세번째 사이클에서 *ACCL*의 LSB는 '0'이므로 덧셈이 수행되지 않고 *ACC*가 ASR 연산을 수행한다.

네번째 사이클은 부호 비트에 대한 연산으로 *ACCH*에서 피승수를 빼준 후 ASR 연산을 수행한다. 네 사이클 후 *ACCH*에는 곱셈 결과의 상위 4 비트가 저장되고 *ACCL*에는 곱셈 결과의 하위 4 비트가 저장된다.

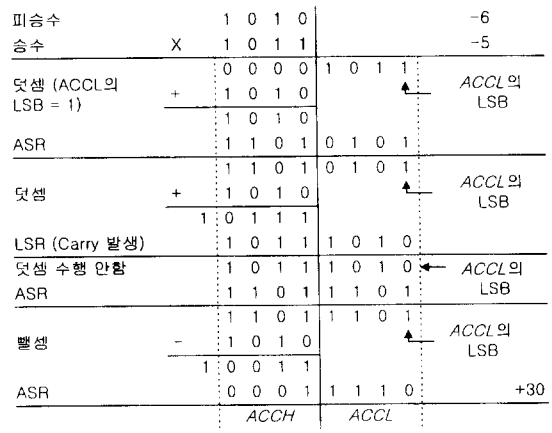


그림 4. 4 비트 곱셈 연산의 예
Fig. 4. An Example of 4-bit Multiplication.

2. 나눗셈 알고리즘

일반적으로 나눗셈은 많은 연산 사이클을 요구한다. 제안하는 아키텍처는 Restoring 나눗셈 알고리즘[15]을 개선하여 적용하였다. 제안하는 아키텍처는 양수 값에 대해서만 처리가 가능하기 때문에 젯수와 피젯수는 반드시 양수로 변환하여 연산하여야 한다. 연산된 결과도 젯수와 피젯수의 부호에 따라 부호 변환이 필요하다. 피젯수와 젯수가 모두 음수인 경우에는 나머지를 음수로 변환해야 하며 피젯수가 음수, 젯수가 양수인 경우에는 몫과 나머지를 모두 음수로 변환해야 한다. 또한 피젯수가 양수, 젯수가 음수인 경우에는 몫을 음수로 변환해야 한다. 이러한 부호 변환과정은 제안하는 아키텍처를 범용 프로세서에 적용하기 위해 가감산기를 ALU로 변환하면 조건적인 명령어를 이용하여 처리가 가능하다.

피젯수를 X, 젯수를 D, 몫을 Q, 그리고 나머지를 R이라고 하면 Restoring 나눗셈 알고리즘에서 i번째 사이클에 발생하는 부분나머지는 식 (2)와 같이 표현된다.

$$r_i = 2r_{i-1} - q_i \cdot D \qquad i = 1,2,\ldots,N \tag{2}$$

여기에서 r_i는 i번째 사이클에 발생하는 부분나머지이며 r_i - 1은 i - 1번째 사이클에 발생한 부분나머지이다. q_i는 몫의 i번째 비트를 의미하며 N은 젯수, 피젯수의 비트수를 나타낸다. 첫번째 부분나머지 $r_0 = X$이며 몫은 $Q = q_N q_{N-1}\ldots q1$이다. i번째 사이클에서 부분나머지는 젯수와 비교하여 $2r_i$ - 1이 D보다 크면 q_i는 '1'이 되고 뺄셈이 수행된다. 만약 $2r_i$ - 1이 D보

다 작다면 q_i는 '0'이 되고 뺄셈은 수행되지 않는다.
그림 5는 제안하는 아키텍쳐에 적용한 개선된 나눗셈 알고리즘의 flow 챠트를 나타낸다. 먼저 $ACCH$는 Clear되어 있으며 $ACCL$에 피젯수가 저장되어 있고 MQ 레지스터에 젯수가 저장되어 있다고 가정한다.

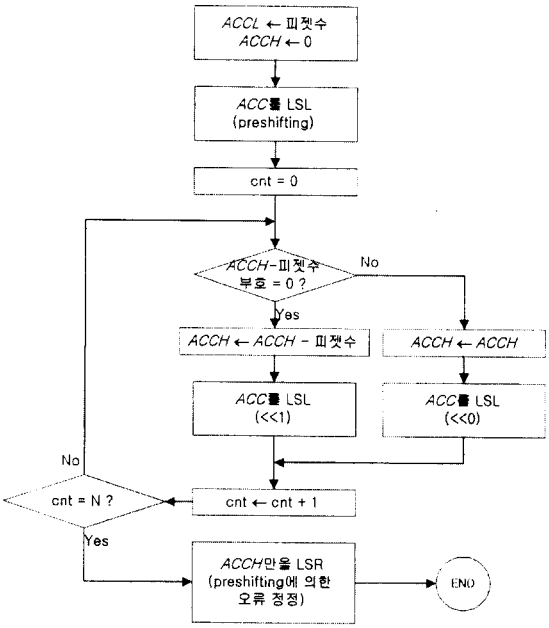


그림 5. 개선된 나눗셈 알고리즘
Fig. 5. The Enhanced Division Algorithm.

첫번째 사이클에서 $2r_0$를 구하기 위해 ACC의 값을 한 비트 LSL(logical shift left : Preshifting)한다. 두번째 사이클부터는 뺄셈을 수행하여 뺄셈 결과의 부호 비트를 점검하여 q_i를 결정한다. 가감산기 출력의 부호 비트가 '0'이면 q_i가 '1'이 되어 뺄셈을 수행한 결과를 $ACCH$로 보내고 가감산기 출력의 부호 비트가 '1'이면 q_i는 '0'이 되어 뺄셈을 수행하지 않은 결과가 $ACCH$로 보내진다. 이와 동시에 ACC는 LSL 연산을 수행한다. 이때 ACC의 LSB로 입력되는 값은 몫으로 가감산기 출력의 부호 비트가 입력된다. N번의 Sub-and-Shift 연산을 수행하면 $ACCL$에는 몫이 저장된다. $ACCH$에는 첫번째 사이클에 Preshifting (LSL)을 수행하였기 때문에 $2R$이 저장된다. 따라서 $ACCH$는 LSR을 수행하여 나머지의 값을 조정해야 한다.

나눗셈 연산은 한번의 Preshifting, N번의 Sub-and-Shift, 나머지 값의 보정을 위한 LSR 연산을 필요로 하기 때문에 총 $N + 2$ 사이클이 소모된다.

그림 6은 4 비트 나눗셈 연산의 예를 나타낸다. 피젯수 '7'은 $ACCL$에 저장되어 있으며 젯수 '2'는 MQ 레지스터에 저장되어 있다. 첫번째 사이클에 ACC의 데이타에 대해 LSL 연산을 수행하여 $2r_0$를 구한다. 두번째 사이클에는 $ACCH$와 MQ 레지스터를 뺀 후 부호가 음수이기 때문에 뺄셈을 수행하지 않은 결과를 $ACCH$로 입력하고 ACC는 LSL 연산을 수행한다. 세번째 사이클은 두번째 사이클과 같은 연산을 수행하며 네번째 사이클에서는 뺀 결과가 양수이기 때문에 뺄셈을 수행한 결과를 $ACCH$로 입력하고 ACC에 대해 LSL 연산을 수행한다. 다섯번째 사이클도 뺄셈을 수행하고 LSL 연산을 수행한다. 여섯번째 사이클에서는 나머지를 보정하기 위해 $ACCH$에 대해 LSR 연산을 수행한다. 최종적으로 $ACCH$에 저장된 '1'은 나머지이며 $ACCL$에 저장된 '3'은 몫을 의미한다. 따라서 4 비트 나눗셈 연산에 6 사이클이 소모된다.

0111(7)/0010(2) = 0011(3:몫) ... 0001(1:나머지)

젯수 (MQ)		0	0	1	0				←피젯수
		0	0	0	0	0	1	1	1
LSL		0	0	0	0	1	1	1	0
	−	0	0	1	0				
음수 →		1	1	1	0				
뺄셈 수행 안함		0	0	0	0	1	1	1	0
LSL		0	0	0	1	1	1	0	0
	−	0	0	1	0				
음수 →		1	1	1	1				
뺄셈 수행 안함		0	0	0	1	1	1	0	0
LSL		0	0	1	1	1	0	0	0
	−	0	0	1	0				
뺄셈		0	0	0	1	1	0	0	0
LSL		0	0	1	1	0	0	0	1
	−	0	0	1	0				
뺄셈		0	0	0	1	0	0	0	1
LSL		0	0	1	0	0	0	1	1
ACCH만 LSR		0	0	0	1	0	0	1	1
		ACCH				ACCL			
		나머지				몫			

그림 6. 4 비트 나눗셈 연산의 예
Fig. 6. An Example of 4-bit Division.

Ⅳ. 성능 비교

제안하는 bit-serial 방식 곱셈기 및 나눗셈기 아키텍쳐는 기존의 bit-serial 방식 곱셈기 및 나눗셈기 회로에 비해 하드웨어 크기의 부담을 감소시켰으며 동일한 연산 수를 갖는다. 표 1은 기존의 아키텍쳐인

Quasi-Serial 곱셈기[9-10] 및 COPE 아키텍처[11]와 제안하는 곱셈기 및 나눗셈기 아키텍처의 하드웨어 크기 및 연산 수를 비교한 표이다. N은 입력 데이타의 비트 수로 N 비트 곱셈 및 N 비트 나눗셈을 가정하였다.

표 1. 기존 아키텍처와의 비교
Table 1. Comparisons with Existing Architectures.

	Quasi-Serial[9-10]	COPE[11]	제안하는 아키텍처
레지스터 비트 수	$\log_2(2N)$	$4N$	N
Shift 레지스터 비트 수	$5N-1$	N	$2N$
가감산기/Counter 비트 수	$\log_2 N$	$2N$	N
2 x 1 MUX 및 게이트	N개의 AND 게이트	$3N$ (MUX)	N (MUX)
기타	-	부호 회로 제어 회로	음의 최소값 검출기 제어 회로
곱셈 연산 사이클	$2N$	N	N
나눗셈 연산 사이클	불능	$N+1$	$N+2$

Quasi-Serial 곱셈기[9-10]는 $\log_2(2N)$ 비트 레지스터, $(5N-1)$ 비트 Shift 레지스터, N-input Counter, N개의 AND 게이트가 필요하다. 연산기에 대한 하드웨어 부담은 가장 적으나 Shift 레지스터의 개수가 많기 때문에 상대적으로 하드웨어 부담이 크다. 또한 곱셈 연산에 필요한 사이클 수는 N번의 Shift 사이클과 N번의 Count 사이클을 합해 $2N$ 사이클이며 나눗셈 연산은 수행할 수 없는 단점을 가지고 있다. COPE 아키텍처[11]는 $4N$ 비트 레지스터, N 비트 Shift 레지스터, N 비트 2의 보수기, $2N$ 비트 전가산기, $3N$ 비트 2 x 1 MUX와 부호 회로 및 제어 회로로 구성되어 있다. COPE 아키텍처는 N 비트 곱셈 및 나눗셈 연산에 각각 N, $N+1$ 사이클이 소모되어 연산수의 부담이 적다. 그러나 부분곱 또는 부분 나머지를 구하기 위해 가산기와 레지스터의 비트 수를 2배 확장해야 하기 때문에 하드웨어 부담이 큰 단점을 가지고 있다.

제안하는 아키텍처는 N 비트 레지스터, $2N$ 비트 Shift 레지스터, N 비트 가감산기, N 비트 2 x 1 MUX와 음의 최소값 검출기 및 제어 회로로 구성된다. N 비트 곱셈 연산에 필요한 사이클 수는 N 사이클이며 N 비트 나눗셈 연산을 위해 $N+2$ 사이클이 소모된다. 표 1에서 볼 수 있듯이 제안하는 bit-serial 방식 곱셈기 및 나눗셈기 아키텍처는 기존의 아키텍처에 비해 하드웨어 부담이 매우 적고 연산 수는 동일함을 알 수 있다.

본 논문에서 제안한 곱셈기 및 나눗셈기 아키텍처는 SliM Image Processor[12-14]에 실제 적용하여 칩으로 구현하였다. 25개의 PE(Processing Element)를 포함하고 있는 SliM 칩은 0.8 μm 표준 셀 라이브러리(v8r4)를 이용하여 구현하였으며 55,255개의 게이트와 25개의 128 x 9 비트 SRAM으로 이루어져 있다. 다이 크기는 326 x 313 mil^2 이며 동작 주파수는 25 MHz이다. 그림 7은 구현한 SliM 칩의 사진이다.

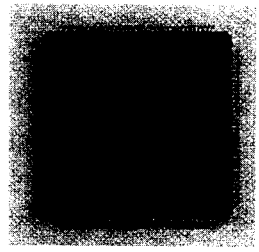


그림 7. SliM 칩 사진
Fig. 7. Photograph of the SliM Chip.

표 2. SIMD 프로세서들의 곱셈 및 나눗셈 연산 사이클 비교
Table 2. Comparisons with SIMD Processors for the Number of Cycles.

종류	8 비트 곱셈 (사이클수)	8 비트 나눗셈 (사이클수)
DAP[4]	225	560
GAPP[5]	252	459
MGAP[6]	135	-
IMAP[7]	11	-
VIP[8]	16	448
SliM[12-14]	9 (9 비트)	11 (9 비트)

SliM은 본 논문에서 제안한 아키텍처로 인해 기존의 SIMD 프로세서에 비해 곱셈 및 나눗셈 연산에서 특히 월등한 성능을 보였다. 표 2는 SIMD 프로세서들의 곱셈 및 나눗셈 연산의 성능을 사이클 수로 비교한 것이다. 기존의 SIMD 프로세서인 DAP[4], GAPP[5], MGAP[6], IMAP[7] 및 VIP[8]은 8 비트 곱셈 연산에 대해 각각 225, 252, 135, 11 및 16

명령어 사이클을 필요로 하며 DAP[4], GAPP[5] 및 VIP[8]는 8 비트 나눗셈 연산에 대해 각각 560, 459 및 448 명령어 사이클을 필요로 한다. 이에 반해 SliM은 9 비트 곱셈 연산에 9 명령어 사이클, 9 비트 나눗셈 연산에 대해 11 사이클만을 필요로 한다. 따라서 제안하는 아키텍쳐는 곱셈기 및 나눗셈기로서의 성능 뿐 아니라 실제 프로세서에 적용하였을 경우에도 성능이 월등함을 알 수 있다.

V. 결 론

본 논문에서는 하드웨어 부담을 크게 줄이고 연산 수는 기존의 아키텍쳐와 동일한 새로운 곱셈기 및 나눗셈기 아키텍쳐를 제안하였다. 제안한 아키텍쳐는 덧셈과 Shift 연산을 동시에 수행하므로써 기존의 Quasi-serial 곱셈기에 비해 연산 사이클 수를 반으로 줄였으며 COPE 아키텍쳐와는 동일한 연산 사이클 수를 갖는다. 또한 부분곱 및 부분나머지 연산을 위해 레지스터 및 가감산기의 비트 수를 2배 확장하지 않으므로 기존의 COPE 아키텍쳐에 비해 하드웨어 부담을 반으로 줄였으며 일반 프로세서의 ALU 연산 구조에 쉽게 적용이 가능한 장점을 가지고 있다. 제안한 아키텍쳐는 SIMD 프로세서인 SliM Image Processor[12-14]에 실제 적용하여 칩으로 구현하였고 그 성능을 입증하였다.

참 고 문 헌

[1] Vijai K. Madisetti, *VLSI Digital Signal Processors*, Butterworth-Heinemann, 1995, pp. 121-129.

[2] Neil H.E. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addson Wesley, 1993, pp. 317-325.

[3] HARRIS semiconductor Inc., *Digital Signal Processing*, 1994.

[4] DAP Series Technical Overview. Active Memory Technology Inc., 1989.

[5] R. Davis and D. Thomas, "Systolic array chip matches the pace of high-speed processing," *Electronic Design*, vol. 32, no. 22, pp. 207-218, Oct., 1984.

[6] R. S. Bajwa, R. M. Owens and M. J. Irwin, "Image processing with the MGAP: a cost effective solution," in *Proc. 7th Int. Parallel Processing Symposium*, Apr. 1993, pp. 439-443.

[7] S. Okazaki, Y. Fujita and N. Yamashita, "A compact real-time vision system using integrated memory array processor architecture," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, no. 5, pp. 446-452, Oct., 1995.

[8] M. Hall, "A study of parallel image processing on the VIP architecture," Ph. D. Thesis, Dep. of Electrical Engineering, Linkoping University, Sweden, Oct., 1995.

[9] Earl. E. Swartzlander, Jr., "The Quasi-Serial Multiplier," *IEEE Trans. Comput.*, vol. C-22, pp. 317-321, Apr. 1973.

[10] T. G. MaDaneld and R. K. Guha, "The Two's Complement Quasi-Serial Multiplier," *IEEE Trans. Comput.*, C-24, pp. 1233-1235, 1975.

[11] Robert E. Morley, Jr., Gray E. Christensen, Thomas J. Sullivan, Orly Kamin, "The Design of a bit-serial Coprocessor to Perform Multiplication and Division on a Massively Parallel Architecture," in *Proc. IEEE, The 2nd Symposium on the Frontiers of Massively Parallel Computation*, Fairfax, U.S.A., pp. 419-422, Oct. 1998.

[12] Soohwan Ong, Surim Ryu, M. H. Sunwoo and S. Lee, "A Parallel Image Processor Chip for Real-Time Applications," in *Proc. International Symposium on Circuits and Systems*, Atlanta, U.S.A., May. 1996, pp. 356-359.

[13] 옹 수환, 선우 명훈, "SliM 이미지 프로세서 칩 설계 및 구현," 전자공학회논문지, 제33권 A편 제10호, pp. 186-194, 1996, 10

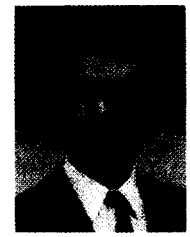
[14] Soohwan Ong and Myung H. Sunwoo, "Implementation of a Sliding Memory Plane Image Processor," in *Journal of Parallel and Distributed Computing*, Nov. 1998.

[15] K. Hwang, *Computer Arithmetic*, John Wiley & Sons, 1979.

저 자 소 개

邕 壽 煥(正會員)

1994년 2월 아주대학교 전자공학 학사. 1996년 2월 아주대학교 전자공학 석사. 1996년 2월 ~ 현재 아주대학교 전자공학 박사과정. 주관심분야는 영상, 멀티미디어, 통신 및 신호처리용 ASIC 설계



鮮 于 明 勳(正會員)

1980년 2월 서강대학교 전자공학 학사. 1982년 2월 한국과학기술원 전자공학 석사. 1982년 3월 ~1985년 8월 한국전자통신연구소(ETRI) 연구원. 1985년 9월 ~ 1990년 8월 Univ. of Texas at Austin 전자공학 박사. 1990년 8월 ~ 1992년 8월 Motorola, DSP Chip Division, 미국. 1992년 8월 ~ 1996년 10월 아주대학교 전기전자공학부 조교수. 1996년 10월 ~ 현재 아주대학교 전기전자공학부 부교수. 주관심분야는 VLSI 및 Parallel Architecture, 통신, 영상 및 신호처리용 ASIC 설계