임베디드 시스템을 위한 저비용 SIMD MAC/MAS 블록 설계

정회원 이 용 주*, 정 진 우*, 이 용 석**

The Design of low-cost SIMD MAC/MAS for Embedded Systems

Yong-Joo Lee*, Jin-Woo Jung*, Yong-Surk Lee** *Regular Members*

요 약

본 논문에서는 실생활에 많이 사용되는 멀티미디어의 연산에 꼭 필요한 명령어를 수행할 수 있는 저면적의 저전력 SIMD MAC/MAS(Single Instruction Multiple Data Multiply and ACcumulate/Multiply And Subtract)를 개발하였다. 개발의 목적이 이전에 개발된 64-bit의 고면적, 고성능 MAC/MAS를 저면적, 저비용화하면서 성능 저하를 최소화 하는 것이었기 때문에 이전에 개발된 구조와 비교함으로써 이번 연구의 성과를 판단하였다. 본 논문의 내용은 크게 SIMD MAC의 설계에 대한 내용, 본 설계가 이전의 설계와의 차별성, 그리고 합성 결과 및 결론으로 이루어져 있다. 설계 결과, 이전에 설계되었던 고성능의 64비트 SIMD MAC/MAS에 비해 전체적인 하드웨어의 크기는 32%로 축소되었다. 이는 임베디드 DSP(Digital Signal Processor)에 적합하도록 ISA(Instruction Set Architecture)를 개선하였고, 내부 데이터의 대역폭을 32비트로 줄였으며 하드웨어를 보다 최적화하여 설계하였기 때문으로 판단된다.

Key Words : DSP, SIMD, MAC, MAS

ABSTRACT

In this paper, we developed a low-area and low-cost SIMD MAC/MAS(Single Instruction Multiple Data Multiply and ACcumulate/Multiply And Subtract) for multimedia that is used much in real life. We compared the result of this research with a previously developed more large and high performance SIMD MAC/MAS. This paper is consist of 5 parts, which are an introduction, the contents of designing SIMD MAC/MAS hardware, a special qualities for previous works, the result of synthesis and conclusion. The design result reduced by size 32% of whole hardware than 64 bit SIMD MAC/MAS block of designed for high performance. This improved ISA (Instruction Set Architecture) to be suitable to embedded DSP(Digital Signal Processor), and shortened bit range of 64-bit data to 32-bit and implement more optimally.

I. 서 론

멀티미디어 분야는 현재 두 가지의 방향으로 나아가고 있다. 첫 째는 하드웨어가 연산 속도가 병목이 될 정도의 고화질을 추구하고 있는 고성능의 게임이나, 영상처리를 위한 하이엔드를 지향하는 분야가 있다. 이 분야는 점차 고성능화되고 있는 멀티미디어 프로세서 분야에서 많은 양의 연산을 효율적

*연세대학교 전기전자공학과 프로세서연구실(jjinu@dubiki.yonsei.ac.kr)
논문번호 : 040059-0203, 접수일자 : 2004년 2월 3일

으로 계산하기 위해 인텔의 MMX, SSE, SSE2 또는 AMD의 3D Now의 경우 동일한 하드웨어 구조를 여럿으로 쪼개어 하드웨어 사용의 효율을 증가시키는 방식인 SIMD(Single Instruction Multiple Data)의 구조를 사용하여 영상처리에서 가장 흔히 사용되는 연산인 MAC(Multiply and ACcumulate)의 경우 64-bit의 MAC을 이용한다면, 2 ~ 8배까지의 성능향상을 가져오고 있다. 하지만 이러한 응용은 수백~ 수천 MIPS를 요구하는 고사양의 컴퓨팅 환경에 적합하다. 이에 비해 두 번째의 경향은 현재의 멀티미디어 응용에서 수익성 및 미래의 유비쿼터스와의 연결 등으로 각광을 받고 있는 모바일의 소형 프로세서 응용분야가 있다. 이러한 모바일의 멀티미디어 응용에서는 고성능의 많은 전력을 필요로 하는 프로세서는 배터리에 의존해야 하는 모바일의 특성상, 채택하기가 힘들다. 또한 모바일의 개념을 충실히 수행하기 위해서는 저전력 뿐만 아니라, 휴대 및 이용을 간편화하기 위해 적은 크기의 단말기를 요구하기 때문에, 소형화하기 위해서는 저면적의 칩을 요구한다. 본 논문에는 멀티미디어 분야의 연산 속도를 향상시키면서 저면적, 저전력의 구조로서 멀티 비트 응용 환경을 지원함으로써 하드웨어 사용의 효율성을 높이여 모바일 환경에 적합하도록 설계한 SIMD MAC블록의 설계 내용을 담고 있다.

II. SIMD MAC

1. SIMD DSP

1.1 개 요

본 논문에서 다루는 SIMD MAC 블록은 이전에 개발되었던 64-bit SIMD DSP의 소형화와 저전력화를 목표로 개발된 SIMD DSP블록에서 가장 핵심이 되는 블록으로서 SIMD DSP에서의 가장 핵심이 되는 명령을 수행한다.

1.2 DSP 전체 수행 명령어

이 번에 개발된 32-bit SIMD DSP의 전체 명령어는 다음의 표 1과 같다. 표에서 보는 바와 같이 총 34개의 명령어를 지원하도록 설계가 되었다.

1.2.1 SIMD MAC 명령어

이전에 개발된 64-bit SIMD MAC의 경우에는 연산의 종류가 8, 16, 32, 64-bit의 네 가지 동작을 모두 수행할 수 있었는데 32-bit DSP에서는 MAC의 경우 단지 32-bit만의 명령어를 지원하게 바뀌었다. 하지만 이전의 64-bit SIMD MAC과의 비교를 위하여 수정을 하여 8, 16, 32-bit SIMD 연산을 지원하도록 MAC을 수정하였다.

표 1. SIMD DSP 명령어와 동작

명령어	동작
MIN	signed number minimum
MAX	signed number maximum
MULL	signed multiply long
MULU	unsigned multiply long
REV	bit reverse
CNTS	count leading sign bits
MAC	signed multiply long & accumulate
MAS	signed multiply long & subtract
EXTB	8bit sign extend
EXTS	16bit sign extend
EXTW	sign extend
CLSB	8bit signed clamp
CLSS	16bit signed clamp
CLSW	signed clamp
CLUB	8bit unsigned clamp
CLUS	16bit unsigned clamp
CLUW	unsigned clamp
PKU8	signed 16bit pack to unsigned 8bit
PKU16	signed 32bit pack to unsigned 16bit
PKS8	signed 16bit pack to signed 8bit
PKS16	signed 32bit pack to signed 16bit
UPKU8	unsigned 8bit unpack
UPKU16	unsigned 16bit unpack
UPKS8	signed 8bit unpack
UPKS16	signed 16bit unpack
CALDA	calculate xy memory next address
REVDA	reverse xy memory address
DLXY	load xy memory
MACD	multiply and accumulate
MASD	multiply and subtract
DLDX	load register from DSP x memory
DLDY	load register from DSP y memory
DSTX	store register from DSP x memory
DSTY	store register from DSP y memory

수정을 가하게 된 명령어는 MACD와 MASD로서 이들은 각각 MACD8, MACD16, MACD, MASD8, MASD16, MASD로 세분화되게 되었다. 다음의 1.3에서 블록에 대한 자세한 설명을 할 것이다. 각각은 8, 16, 32-bit의 MAC/MAS 명령을 수

행한다.

1.3 MXX 유닛

1.3.1 MAC 및 MAS 유닛

MAC과 MAS(Multiply And Subtract) 연산은 SIMD DSP 전체 블록에서 가장 핵심이 되는 블록으로서 수행 시간이 가장 길다. 또한 멀티미디어 데이터의 인코딩, 필터링 등의 동작에서 매우 빈번히 쓰이는 연산이기 때문에 이 유닛의 설계에 가장 시간을 많이 들였으며, 현재 나와 있는 많은 알고리즘을 적용하여 여러 가지로 구현을 해 보았고 그 중에 가장 효율적인 구조를 이용하여 설계를 하게 되었다.

MAC 연산과 MAS 연산의 동작은 식 (1), (2)과 같다.[3]

dst+1:dst=src3:src4+src1*src2(MAC) (1)

dst+1:dst=src3:src4-src1*src2(MAS) (2)

위의 식 (1)와 (2)에서보면, MAC과 MAS의 차이는 32비트 연산의 예를 들면, 64비트 숫자에 곱해지는 수가 더해지면 MAC 연산이 되고, 빼지면 MAS 연산이 된다. 이 때 빼는 연산에 대해 여러 가지 방법이 있다. 이는 MXX 유닛의 설계 부분을 마치고, SIMD MAC 특징의 단락에서 설명하겠다.

MXX 유닛의 설계 방식은 여러 논문을 참고한 결과 풀 커스텀(full custom) 방식 등의 회로(circuit) 기술을 사용하지 않고 스탠더드 셀(standard cell) 방식의 설계에 있어서는 주로 월리스 트리(wallace tree)를 이용한 수정 부스(modified booth)방식이 많이 쓰이고 잉여이진수체계(redundant binary number system)를 이용한 방식도 쓰이는 것으로 나타났다.[1]-[11]

이에 따라서 본 논문에서는 수정 부스 알고리즘을 이용하여 레이딕스(radix) 4, 레이딕스 8, 혼합 레이딕스(hybrid radix)[1]-[7][9][11] 방식을 구현하여 비교해 보았고, 또한 잉여이진수체계를 이용한 곱셈기[8][10]를 구성해 보았다.

(1) 수정 부스 알고리즘

수정 부스 알고리즘에서는 최초의 부스 알고리즘과 달리 전체 수의 값을 고려하는 것이 아니고, 레이딕스 4의 경우 2 개로, 레이딕스 8의 경우 3 개 등등의 작은 수 단위의 값으로 기록한다. 이렇게 비트를 고정시킴으로써 하드웨어적인 구현이 정규적으로 되고, 원래 알고리즘 상에서 1과 0이 연속될 때에 나타나는 1과 -1의 연속에 의한 하드웨어의 복잡도 또한 고정되게 된다. 따라서 각 블록 간의 수행시간을 통일 시킬 수 있고, 동기 회로로의 구현이 용이하다.

다음의 그림 1은 레이딕스 4 수정 부스 알고리즘에서 승수를 분할하여 기록하는 것을 표현하였다. 수정 부스 알고리즘을 적용하면, 레이딕스 4의 경우 부분곱이 17개, 레이딕스 8의 경우 부분곱이 11개가 되어 원래 기록을 하지 않은 것이 32개인 것에 비해 많이 줄어들게 된다.[3][9] 또한 레이딕스 4와 레이딕스 8을 혼합한 혼합 부스 방식을 사용하면 부분곱의 수는 늘지만 다음에 설명하게 될 레이딕스 8의 단점을 보완할 수 있다.

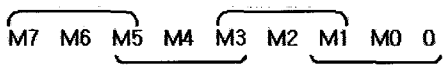

그림 1. 레이딕스 4 수정 부스 알고리즘을 통한 승수 분할

(3) 잉여이진수체계 알고리즘

잉여이진수 체계는 일반적인 이진 SD(Signed Digit)의 수 체계는 다음의 식 (3)과 같다.[8]

$$\text{digit-set} \in \{-1,0,1\} \quad (3)$$

이진 SD 수의 가산은 다음의 식 (4)과 같이 나타난다.[8][10]

$$(x_{n-1}, \ldots, x_0) \pm (y_{n-1}, \ldots, y_0) = (s_{n-1}, \ldots, s_0)$$

$$\text{단계 1. 중간합} : u_i = x_i + y_i - 2c_i \quad (4)$$

$$\text{단계 2. 최종합} : s_i = u_i + c_{i-1}$$

c_i는 다음의 식 (5)과 같이 결정된다.[10]

$$c_i = \begin{cases} 1 & \text{if } (x_i + y_i) \geq 1 \\ -1 & \text{if } (x_i + y_i) \leq -1 \\ 0 & \text{if } (x_i + y_i) \leq 1 \end{cases} \quad (5)$$

일반적인 이진 SD 수 체계의 경위 위의 식 (5)에

서 자리올림이 1이 되고 바로 상위의 비트 중간합이 1 또는 -1이 되는 경우, 자리올림이 계속적으로 전파되는 현상이 발생하게 되어, 덧셈의 수행시간이 최악 조건에서 비트 수만큼 전파되는 현상이 발생된다. 이러한 단점을 막기 위한 방식이 1984년 Takagi에 의해 제안되었다.[8][10] 제안된 동작은 1비트 상위 중간합과 자리올림이 동시에 1 또는 -1이 되는 현상을 방지함으로써, 최종합에서 자리올림의 전파를 막는 방법이다. 일반적으로 대부분의 수 체계가 이진 SD 체계이기 때문에 외부로부터 들어오는 이진 SD 수를 곱셈기 내부에서 사용되는 잉여이진 수로 기록이 필요하고, 이를 연산한 결과 역시 이여이진수이기 때문에 이를 다시 이진 SD로 바꾸어 주는 추가적인 하드웨어가 필요하다.

1.3.2 레이딕스 4, 8, 수정 레이딕스 곱셈기

(1) 레이딕스 4/8 곱셈기

레이딕스 4 부스 인코딩의 가능한 기록은 -2x, -x, 0, x, 2x의 5 가지만이 존재한다. 이 방식의 장점은 가능한 조합이 한정되어 있기 때문에 매우 간단하게 부분곱을 생성시킬 수가 있다. 음수의 경우 2의 보수가 아닌 1의 보수를 생성하데 그 이유는 1의 보수의 경우 단순한 xor게이트만으로 보수화가 가능하고, 자리올림의 전달이 없기 때문이다. 여기서 xor게이트를 사용하여 보수화하는 방법을 다음의 그림 2에서 보여주고 있다.

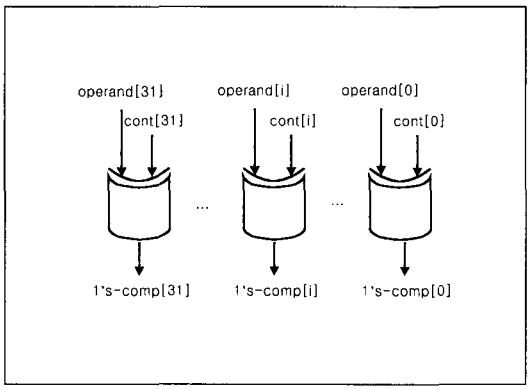

그림 2. XOR 게이트를 이용한 1의 보수 구현

하지만, 사용되는 수자는 2의 보수 숫자이기 때문에, 음수로 변환되는 경우 최하위 비트에 2의 보수화를 위한 1을 더해 주어야 한다. 이것을 자리올림의 전달이 없는 월리스 트리부분에서 더해주게 되어 실제로 자리올림의 전달 없이 2의 보수의 구현이 가능하도록 하였다. 다음의 그림 3은 월리스 트리로 들어가는 부분곱의 구성도이다.

그림 3에서 s는 부호(sign)의 약자로서 이는 피제수의 최상위 비트 또는 0을 채워준다. 이유는 설계된 곱셈기가 부호/비부호 숫자에 대한 곱셈을 모두 수행할 수 있도록 되어있기 때문이다. 아무런 변형을 가하지 않은 상태에서의 수는 부호 숫자로 인식되고 덧셈을 알고리즘이 2의 보수의 덧셈으로 수행되기 때문에, 부호 곱셈의 경우, 단순히 부호인 최상위 비트를 연장하여 주면 되지만, 비부호 곱셈 명령인 경우, 양수의 부호를 나타내는 0을 강제로 입력하여 줌으로써 현재의 값이 양수로 인식되도록 하면 비부호 곱셈이 가능하다.[9][11] 그리고, 나머지 1과 s의 역은 앞으로 설명할 부호 확장 알고리즘에 의해 추가되는 값들이다. 부호수의 경우, 올바른 값을 얻기 위해서는 부호 비트를 출력값과 같은 크기로 확장해야 한다.[8]

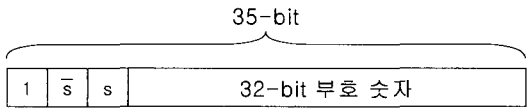

그림 3. 32-bit 연산의 부분곱의 구성

그러나, 이런 방식을 사용하게 되면, 소비 전력 및 하드웨어가 매우 커지게 된다. 이러한 단점을 극복하기 위해 부호 확장을 최소화하는 알고리즘을 이용하여 구현하였다.[8] 월리스 트리는 최대한의 추가 하드웨어를 없애는 방식으로 최적화하여 설계하였다.[8] 3:2 카운터를 사용하여 구현하였으며, 각 단계별로 줄어드는 부분곱의 수를 예상하여 최소한의 하드웨어를 이용하여 구현하였다. 그림 8에서는 수정 레이딕스 월리스 트리를 표현했는데 그 이유는 수정 레이딕스 월리스 트리는 레이딕스 4와 레이딕스 8이 모두 포함되었기 때문이다. 그림에서 맨 위의 4 블록이 레이딕스 4의 월리스 트리 부분이고 중간 및 밑 부분의 블록은 레이딕스 8의 부분이다. 그림 8의 중간의 월리스 트리 밑의 어두운 부분은 MAC/MAS를 위한 64 비트 입력 값이다. 그림의 어두운 부분을 밑의 월리스 트리처럼 월리스 트리에 흡수시킴으로써 추가 지연 없이 MAC/MAS구조를 설계하였다. 최종 덧셈단에서 이전단의 값을 더해주면, 이전단의 값과 월리스트리 출력인 합(sum), 자리올림(carry)의 값의 세 개가 더해진다면, 최종단

에서 또한 CSA(Carry Save Adder)를 사용해야 하므로 추가적인 수행시간 지연이 생기지만, MAC/MAS를 위한 이전단의 값을 월리스 트리에 넣음으로써 추가적인 수행시간 없이 MAC/MAS를 구현하였다. 그림 4에서 C는 2의 보수를 구현하기 위한 1의 값이다. 음수의 구현 시, 1의 보수를 사용하여 단순히 값의 역전을 하였지만, MAC/MAS 블록내의 수체계가 2의 보수 체계이기 때문에 음수의 경우 0번째 비트에 1을 넣어주어야 한다. 하지만, 자리 이전의 전파 때문에 부분곱에 직접 넣지 않고 다음단의 부분곱에 이전단의 최하위비트 위치에 넣어 줌으로써, 자리올림 전파 없이 구현하게 되었다. 명령어에 따른 mode를 입력으로 받아 SIMD 연산인 경우 mode를 0으로 하면, 부스 인코더의 입력값이 0이 되어 SIMD 연산을 위한 부분곱이 생성된다. 각 자리 올림의 부분을 AND 게이트를 이용하여 블록화하여 월리스 트리의 자리 올림을 mode 값으로 차단하면, 각각의 별개의 곱셈기로 동작시킬 수 있다.

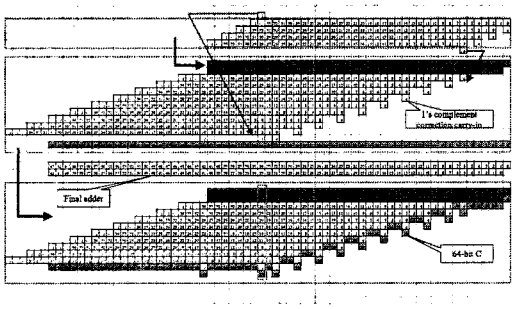

그림 4. 수정 레이딕스 월리스 트리

MAS의 동작을 위하여 부스 인코딩의 패턴을 바꾸었다. 표 4와 5는 단순한 MAC을 위한 인코딩 표이고 MAS를 위해서는 곱의 음수가 필요로 하기 때문에 MAC에서의 a × b를 a × (-b)로 인식하게 하는 인코딩을 수행하게 된다. MAS의 인코딩을 위해서는 표 1, 2의 인코딩에서 0x를 제외한 모든 경우에서 음과 양을 반전시키면 된다. 하지만, 월리스 트리의 부분곱의 생성 방식이 1의 보수로 계산을 하기 때문에, 월리스 트리 방식에 맞게 0일 경우에도 반전을 하게 설계하였고, 2의 보수를 위한 입력이 1이 되므로 결과에 영향을 주지 않고 설계를 간편화하였다. 다음의 표는 최종적인 레이딕스 4/8 곱셈, MAC, MAS를 위한 부스 인코딩 및 2의 보수 입력의 생성을 표로 나타내었다. 레이딕스 8의 곱셈기 구조의 단점은 레이딕스 4에는 없는 ±3x의 존재에 있다. 3x의 경우에는 부분곱 생성 시 필연적으로 x + 2x라는 더하기 연산을 필요로 하기 때문이다. 이 때문에, 3x 연산 지연이 발생하여 월리스 트리로 입력되는 임계 지연에 포함되게 된다. 따라서 이 3x 연산의 고속화가 레이딕스 8 곱셈기에 있어서 매우 중요한 역할을 한다. 구현 결과를 볼 때, 레이딕스 8의 곱셈기가 부분곱의 수가 줄기 때문에 월리스 트리의 크기에서 이득을 보지만, 부스 인코더가 커지고 3x를 위한 덧셈기를 추가하기 때문에, 레이딕스 8곱셈기는 임베디드 프로세서와 같은 구조에서는 이득이 없는 것으로 나타났다.

(2) 수정 레이딕스 곱셈기

수정 레이딕스 방식의 곱셈기는 이전에 언급한 바 있듯이 레이딕스 8의 부분곱 와 비슷한 크기에서, 3x 생성지연에 의한 단점을 보완한 구조이다. 3x 생성시 발생되는 지연시간에 3x 생성이 없는 레이딕스 4의 동작을 함으로써 부분곱을 수를 줄이고, 또한 하드웨어의 이용도를 극대화시키는 구조이다.[9]

표 1. 제안된 레이딕스 4 인코더 로직

Y_{i+1} Y_i Y_{i-1}	Multiply, MAC		MAS	
	부분곱	보수 보정	부분곱	보수 보정
000/111	34'b0	0	all 1	1
001/010	{X[32],X[32:0]}	0	~{X[32],X[32:0]}	1
011	{X[32:0],0}	0	~{X[32:0],0}	1
100	~{X[32:0],0}	1	{X[32:0],0}	0
101/110	~{X[32],X[32:0]}	1	{X[32],X[32:0]}	0

월리스트리를 구성은 4개의 레이딕스 4 부스 인코더와 9개의 레이딕스 8 부스 인코더, 총 13개의 부스 인코더를 이용하여 부분곱을 생성하였다. 이는 레이딕스 4가 17개, 레이딕스 8이 11개인 것에 비해 중간적이 숫자이다. 월리스 트리 생성에서 이전의 논문[4][11]은 레이딕스 4의 단계가 3 단계였던 것에 비해, 2 단계로 줄이고, 고성능의 CLA(Carry Look-ahead Adder)를 사용함으로써, 보다 낮은 성능 향상을 가져오게 되었다.

1.3.3 잉여이진 곱셈기

잉여이진수의 장점이 자리올림의 전파가 원천적으로 차단되는 특성이 월리스트리의 구조와 같은 점에 착안하여 설계되었다. 월리스 트리 부분이 잉여이진 수를 위한 인코더로 교체되었을 뿐, 레이딕스 4/8 인코더 로직은 그대로 사용할 수 있다. 하지만, 인코더에서 생성되는 값은 잉여이진의 값을 생성시켜야하는 차이가 있다. 또한 잉여이진 곱셈기의 장점은 레이딕스 8 이상에서는 모두 일정 지연을 통하여 부분곱을 생성할 수 있다. 레이딕스 8의 ±3x의 생성시간이나, 레이딕스 16의 ±3x, ±5x, ±7x의 생성에 소비되는 시간이 일정하게 된다. 이는 잉여이진 수 체계가 자리올림 전파가 없기 때문이다. 따라서 하드웨어적인 면에서 구속을 받지 않을 경우, 매우 효과적인 설계구조가 된다. 또한 구조가 매우 규칙적이어서, 풀 커스텀 방식의 설계에도 3:2 counter를 이용한 월리스트리에 비해 좋은 장점을 가질 것으로 기대된다.

표 2. 제안된 레이딕스 8 인코더 로직

$Y_{i+2}Y_{i+1}Y_i$ Y_{i-1}	Multiply, MAC		MAS	
	부분곱	보수 보정	부분곱	보수 보정
0000/1111	35'b0	0	all 1	1
0001/0010	{X[32],X[32],X[32:0]}	0	~{X[32],X[32],X[32:0]}	1
0011/0100	{X[32],X[32:0],0}	0	~{X[32],X[32:0],0}	1
0101/0110	3X	0	~3X	1
0111	{X[32:0],0,0}	0	~{X[32:0],0,0}	1
1000	~{X[32:0],0,0}	1	{X[32:0],0,0}	0
1001/1010	~3X	1	3X	0
1011/1100	~{X[32],X[32:0],0}	1	{X[32],X[32:0],0}	0
1101/1110	~{X[32],X[32],X[32:0]}	1	{X[32],X[32],X[32:0]}	0

III. SIMD MAC 특징

1. 저전력 설계

(1) 하드웨어 최소화

설계된 SIMD DSP는 저전력 구조를 구현하기 위해서 최대한 하드웨어의 중복을 피해 크기를 줄였다. 클램프 유닛의 설계에서 부호 카운트 명령이 있기 때문에 필연적으로 설계된 부호카운터 유닛을 그대로 이용하였고, 팩 유닛의 설계에서 클램프 동작에서 필요할 경우 이미 설계되어 있는 클램프 유닛을 공유하여 설계하였다. 또한 1의 보수를 생성할 때, 역전(invert)된 값과 이전 값을 MUX를 통해 구현하던 옛날 방식을 개선하여 XOR 게이트를 이용한 방식을 사용함으로써 하드웨어의 크기를 줄였다. 또한 많이 쓰이지 않는 32비트를 초과하는 정수연산 등의 명령어를 지원하지 않고 최소한의 DSP를 위한 최소한의 명령어를 축약함으로써, 전체 하드웨어의 크기를 줄일 수 있었다. 이전에 필자가 공동으로 연구하여 칩으로 구현된 64비트 MUSTANG2에서 고성능의 64비트 SIMD 연산을 수행하도록 설계하였는데, 수행시간 및 하드웨어의 복잡도가 상당히 높았지만, 이번의 32비트 DSP에서는 이러한 많은 기능을 단순화 시켜 하드웨어를 작게 하였다. 그림 5는 MUSTANG2의 칩 사진이다.[11] 현재 32-bit 프로세서는 레이아웃 사진이 현재 나와 있지 않다.

(2) 게이팅을 이용한 입력 고정

III장에서 열거한 모든 유닛의 입력 부분에 활성화 신호(enable signal)와 함께 입력 값을 곱게이트(AND gate)를 이용하여 연결함으로 연산이 없는 모든 유닛은 입력 값이 0으로 일정하게 유지되기 때문에, 다이내믹 파워(dynamic power)를 줄일 수 있도록 설계되었다. 일반적인 소비전력은 스태틱 파워(static power)와 다이내믹 파워로 나눌 수 있다. 이 때, 스태틱 파워는 공정의 개선이나 구동 전압의 저하를 이용하여 줄일 수 있다. 하지만, 스탠더드 셀 방식의 합성을 이용한 설계에서, 위의 방법으로의 구현은 저전력을 위한 라이브러리를 이용한 합성방식의 방법과 하드웨어의 크기를 줄이는 방법은

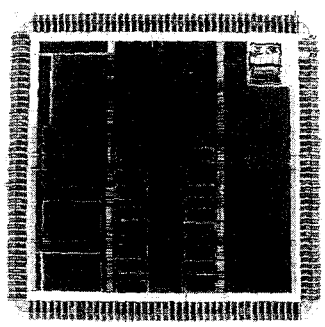

그림 5. MUSTANG2의 레이아웃 모습

이용될 수 있으나, 구조 개선의 방법으로는 스태틱 파워를 줄이기 힘들다. 따라서 필자는 곱게이트를 이용하여 동작하지 않을 때의 입력값을 0으로 고정시킴으로써 토글(toggle)이 일어나지 않게 하여 다이내믹 파워를 줄이도록 설계하였다.

Ⅳ. 합성 결과

본 논문에서 설계된 SIMD DSP 유닛은 Verilog-HDL (Verilog-Hardware Description Language)로 모델링 되었다. 모든 연산 유닛은 RTL (Register Transfer Level)로 설계되었기 때문에 합성 및 최적화하기에 매우 적합하다. 이 Verilog-HDL 모델은 C 프로그램으로 입력 벡터를 생성하고 이에 상응하는 결과 벡터를 발생시킨 후, 입력 벡터를 Verilog-HDL 모델에 가하여 얻은 결과와 비교하여 Verilog-XL tool을 이용하여 기능 검증을 하였다.[9] Verification을 한 후 Synopsys design compiler를 이용하여 삼성전자 0.35 미크론 공정의 표준 셀 라이브러리 (STD90)를 이용하여 합성하였다. 메모리 블록은 SRAM 라이브러리를 사용하여 구성하였으며, 메모리 블록의 합성은 Cubic tool을 이용하여 합성 결과를 구한다. 타이밍 시뮬레이션은 로직 합성으로 얻어진 net 리스트와 SDF (Standard Delay Format) 파일을 이용하여 이루어졌다. Synopsys tool을 통해 얻어진 STA (Static Time Analysis)로부터 최악 동작 조건 (3.0V, 85C), 최악 공정 (worst-case process)에서의 지연시간을 얻어 최대 동작 주파수를 결정짓는다.

Synopsys tool에서는 면적이 등가 게이트 숫자로 표현되는데 2-입력 nand 게이트가 4 개의 트랜지스터로 구성되어 있으므로 트랜지스터 개수는 대략적으로 등가 게이트 숫자에 4를 곱하면 구할 수가 있다. 연산 유닛의 지연시간은 합성된 회로를 정량적으로 분석하고 가장 긴 지연시간을 갖는 지연 경로를 구함으로써 얻을 수 있다. 이때 각 게이트의 팬인(fan-in)/팬아웃(fan-out) 문제를 고려하여야 한다. 비록 풀 커스텀 레이아웃 기술이 높은 집적도와 빠른 동작 속도를 얻는 설계에는 유리하지만 설계 시간과 복잡함에서 많은 비용을 들여야 하는 어려움이 있다. 따라서 제안된 DSP 유닛과 부동소수점 유닛을 합리적인 비용으로 표준 셀 방식으로 구현하는 것이 바람직하다.[12] 위의 표 3은 SIMD DSP 블록 중에서 가장 핵심이 되는 MAC/MAS 블록의 합성결과를 표시했다. 64비트 MUSTANG2의 MAC/MAS의 지연시간이 가장 빠른 이유는 하나의 64비트 MAC/MAS 유닛을 두 개의 32비트 레이딕스 4 MAC/MAS로 쪼갠 후 이를 3사이클에 걸쳐서 연산을 완성하기 때문이다.

표 3. 각종 MAC/MAS 유닛의 합성 결과

Unit	Delay (ns)	Area
레이딕스 4 MAC/MAS	10.18	16,084
수정 레이딕스 MAC/MAS	10.43	14,096
레이딕스 8 MAC/MAS	10.61	13,571
잉여이진 레이딕스 4 MAC/MAS	9.64	22,123
64비트 MUSTANG2 MAC/MAS	8.62	50,832

다음의 그림 9은 MUSTANG2의 MAC/MAS에서 3사이클에서 부분곱의 덧셈을 완성하는 과정을 보여준다. 최종적으로 위의 표 4의 내용을 참고하여 우리가 설계한 SIMD DSP의 MAC/MAS 유닛은 수정 레이딕스 알고리즘을 이용하여 설계하게 되었다. 빠르기 면에서는 잉여이진 레이딕스 4 연산기가 가장 좋았지만, 하드웨어의 크기 면에서 많은 차이를 보였기 때문이다. 다음의 표 4는 SIMD DSP의 유닛별 합성결과를 보여준다. 합성결과에서 알 수 있듯이 MXX 유닛이 전체 DSP 기능 유닛 중에서 80%이상을 차지하는 것으로 나타났다.

하지만, 메모리 유닛은 컨트롤러만을 포함하였기 때문에 합성이 안 되는 XY memory를 추가한다면, 64비트 SIMD DSP 블록을 기준으로 판단할 때, 40%정도가 될 것으로 예상된다. 64비트의 고성능 DSP 프로세서에 비하면, 설계된 32비트의 SIMD

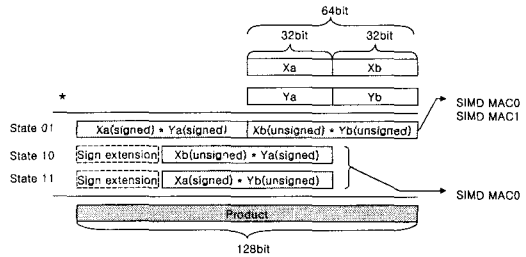

그림 6. MUSTANG2의 MAC/MAS 완성 알고리즘

DSP는 약 32%정도로 매우 작다. 이유는 내부 연산의 대역폭이 반으로 줄었고, 또한 고성능의 응용에 필요한 명령어의 수를 많이 줄이고, 이 후에 개발된 모델이기고, MAC/MAS의 설계에 대한 보다 많은 연구가 진행되었기 때문에 좀더 하드웨어적인 최적화를 수행했기 때문으로 판단된다.

다음의 표 5는 이전에 개발된 다른 논문에서의 MAC의 합성결과를 발췌해 온 것이다.[9][13][14]

표 4. SIMD DSP 유닛 별 합성결과

Unit	Delay (ns)	Area
MM 유닛	6.91	334
REV 유닛	0.37	120
CLCN 유닛	7.93	934
PKXX 유닛	7.78	400
MXX 유닛	10.43	14,096
ADR_GEN 유닛	9.96	1,663
Total		17,547
64비트 MUSTANG2 Total		54,080

V. 결 론

이번 논문은 SIMD DSP 설계에 있어서 가장 핵심이 되는 MAC/MAS 블록의 소형화에 대한 연구를 중점적으로 설계를 하였다. 또한 저전력에 대한 기본적인 연구를 수행하여 소비전력의 감소를 위한 기능을 추가하였다. XY memory가 현재 싱크로노스 듀얼 포트 SRAM으로 설계되었다. 이 메모리 구조는 매우 동작속도가 빠르고, 온칩화가 쉽기 때문에, SIMD DSP 블록을 사용하지 않을 때에는 시스템의 동작 성능을 높일 수 있도록 캐쉬(Cache)로 사용될 수도 있을 것으로 판단된다. 칩으로 제작된 64비트 프로세서의 경우, 그 응용분야가 실제적으로 매우 한정될 수밖에 없기 때문에, 32비트의 중성능 임베디드 SIMD DSP 모듈을 개발함으로써 앞으로의 프로세서 개발 분야의 활성화에 도움이 되었으면 한다.

표 5. 제안된 MAC/MAS와 타 논문의 구조와의 비교

	MAC	곱셈기	제안된 MAC/MAS
Process	0.25um	0.25um	0.25um
Design	full-custom	full-custom	full-custom
최대 동작 속도	100MHz	227MHz	93MHz
트랜지스터 수	67,000	100,200	56,384
파이프라인 단계	2	1	1
지원 포맷	15x15	54x54	32x32, 16x16,8x8

참 고 문 헌

[1] Ohsang Kwon; Nowka, K.; Swartzlander, E.E. "A 16-bit×16-bit MAC design using fast 5:2 compressors", Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on, 10-12 July 2000, Page(s): 235-243

[2] Richard J. Higgins, Georgia Institute of Technology, Digital Signal Processing In VLSI, Prentice Hall.

[3] Israel Koren, *Computer Arithmetic Algorithms* , Prentice Hall, 1993.

[4] 홍인표, "멀티미디어 데이터 처리에 적합한 SIMD 곱셈 누적 연산기의 설계", 연세대학교 석사 학위 졸업논문, pp. 8-14, 22-30, 34-36, 2001.

[5] Yong Surk Lee, "A 4 Clock Cycle64 X 64 Multiplier With 60 MHz Clock Frequency", 대한전자공학회 논문지, Vol. 2, No. 2, pp. 61-67, December 1991.

[6] Aamir a. Farooqui, Vojin G. Oklobdzija, "General Data-Path Organization of a MAC unit for VLSI Implementation of DSP Processors", IEEE, pp. 260-263, 1998.

[7] 박종환, "32비트 RISC/DSP 프로세서를 위한 17 X 17비트 곱셈기의 설계", 연세대학교 석사 학위 졸업논문, pp. 11-12, 16-18, 26-28, 1999.

[8] 홍종욱, "Redundant Binary 연산을 이용한 실수/복소수 승산기", 연세대학교 석사 학위 졸업논문, pp.12-19, 1999

[9] 김승철, "32-Bit 내장형 프로세서에 적합한 저비용 DSP/FPU 설계", 연세대학교 석사 학위 졸업논문, pp.6-13, 39-54

[10] Naofumi Takagi , et. al., "High-Speed VLSI Multiplication Algorithm with a Redundant

Binary Addition Tree," IEEE Trnas. on Computers, vol. C-34, no. 9, pp. 789-796, Sep.1985.

[12] 정우경, "고성능 내장형 마이크로프로세서를 위한 SIMD DSP/FPU 설계", 연세대학교 박사 학위 졸업논문, pp. 44-65, 86-98, 163-183, 2003.

[13] D.H.Allen, S.H.Dhong, H.P.Hofstee, J.Leenstra, K.J.Nowka, D.L.Stasiak, D.F.Wendel, "Custom circuit design as a driver of microprocessor performance", *IBM J. RES. DEVELOP*, vol. 44, No. 6, November 2000.

[14] Dusan Suvakovic, C.Andre T. Salama, "gA pipelined multiply-accumulate unit design for energy recovery DSP systems", *IEEE International Symposium on Circuits and Systems,* pp. I-16 - I-19, May, 2000.

[15] Norio Ohkubo, Makoto Suzuki, "gA 4.4 ns CMOS 54 x 54-b multiplier using pass-transistor multiplexer", *IEEE Custom Integrated Circuits Conference,* pp. 599-602, 1994.

이 용 주 (Yong-joo Lee) 정회원

1999년 8월 : 연세대학교 전자공학과 졸업

2001년 8월 : 연세대학교 전기전자공학과 석사학위 취득

2001년 9월 ~ 현재 : 연세대학교 박사과정

<관심분야> 전자공학, 마이크로프로세서, ASIC