논문 2007-44SD-4-8

멀티미디어 무선 단말기를 위한 재구성 가능한 코프로세서의 설계

(Design of Reconfigurable Coprocessor for Multimedia Mobile Terminal)

김 남 섭*, 이 상 훈**, 금 민 하**, 김 진 상***, 조 원 경***

(Namsub Kim, Sanghun Lee, Minha Kum, Jinsang Kim, and Wonkyung Cho)

요 약

본 논문에서는 멀티미디어 무선단말기에 적합한 코프로세서를 설계하였다. 멀티미디어 무선단말기는 많은 양의 멀티미디어 데이터를 실시간으로 처리하기 때문에 고속 멀티미디어 연산을 지원하는 코프로세서가 요구된다. 따라서 본 논문에서는 재구성 가능한 구조를 사용하여 고속 연산이 가능한 코프로세서의 구조를 제안하고 이를 설계하였다. 제안된 코프로세서는 재구성이 가능할 뿐만 아니라 PE(Processing Element)들을 그룹 단위로 묶어서 응용분야에 따라 확장이 가능하도록 하였으며 곱셈기를 사용하지 않고 곱셈처리가 가능하도록 하였다. 또한 메인 프로세서의 시스템 I/O 버스에 연결되도록 하였기 때문에 모든 프로세서에 연결이 가능하도록 하였다. 제안된 코프로세서는 VHDL을 이용하여 설계되었으며 설계된 코프로세서를 기존의 재구성 가능한 코프로세서 및 상용 임베디드 프로세서와 구조비교 및 성능비교를 하였다. 비교 결과, 제안된 코프로세서는 기존의 재구성 가능한 코프로세서에 비해 융통성 및 하드웨어 크기 면에서 우수함을 나타내었고, 실제 DCT 응용분야에서 상용 ARM 프로세서에 비해 26배의 속도증가를 보였으며 고속 DCT코어를 탑재한 ARM프로세서와의 비교에서 11배의 속도증가를 나타내었다.

Abstract

In this paper, we propose a novel reconfigurable coprocessor for multimedia mobile terminals. Because most of multimedia operations require fast operations of large amount of data in the limited clock frequency, it is necessary to enhance the performance of the embedded processor that is widely used in current multimedia mobile terminals. Therefore, we proposed and have designed the coprocessor which had the ability of fast operations of multimedia data. The proposed coprocessor was not only reconfigurable, but also flexible and expandable. The proposed coprocessor has been designed by using VHDL and compared with previous reconfigurable coprocessors and a commercial embedded processor in architecture and speed. As a result of the architectural comparison, the proposed coprocessor had better structure in terms of hardware size and flexibility. Also, the simulation results of DCT application showed that the proposed coprocessor was 26 times faster than a commercial ARM processor and 11 times faster than the ARM processor with fast DCT core.

Keywords : Coprocessor, Reconfigurable Architecture, Multimedia, Mobile Terminal

I. 서 론

멀티미디어 무선 단말기는 많은 양의 멀티미디어 데이터를 처리하기 때문에 기존의 프로세서만을 사용한 멀티미디어 데이터 처리방식으로는 그 한계점이 있다. 따라서 코프로세서(coprocessor)를 사용하여 프로세서

* 정회원, 한림대학교 정보통신공학부
(Div. of Information Engineering and Telecommunications, Hallym University)
** 학생회원, *** 정회원, 경희대학교 전자정보학부
(School of Electronics and Information, Kyung Hee University)
※ 이 논문은 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임
(과제번호 : DD00525(R05-2004-000-12487-0))
접수일자: 2007년2월2일, 수정완료일: 2007년3월15일

의 성능을 높일 수 있으며 제한된 클록 주파수에서 최대한의 성능을 내기 위한 방법으로 재구성 가능한 구조가 연구되어 왔다[1]. 재구성 가능한 구조는 재구성을 위한 로직의 입도(granularity)에 따라 fine-grained와 coarse-grained로 나뉜다. FPGA와 같은 fine-grained 형태는 일반 커스텀 IC(custom IC)에 비하여 속도가 느리기 때문에 성능 향상을 위한 코프로세서에는 적합하지 않다. 따라서 coarse-grained 형태의 재구성 가능한 구조가 많이 연구되어 왔으나 기존의 재구성 가능한(reconfigurable) 구조들은[2][3] 멀티미디어 연산처리에 필수적인 곱셈 연산을 위하여 큰 용량의 곱셈기를 사용하기 때문에 이에 따른 하드웨어의 용량이 커진다는 단점을 갖고 있다. 따라서 본 논문에서는 곱셈기 없이 고속 연산이 가능한 재구성 가능한 코프로세서를 제안한다.

제안하는 코프로세서는 곱셈 연산을 위하여 DA (Distributed Arithmetic)[4]를 적용하여 곱셈기 없이 곱셈연산을 수행할 수 있도록 하였으며 연산처리를 블록 단위로 나누어 처리 용량에 따라 하드웨어의 크기가 조절이 될 수 있도록 하였다. 즉, 적은 용량의 데이터 처리에는 적은 수의 블록을 사용하여 연산 작용을 할 수 있고, 고 용량의 데이터 처리에는 많은 수의 블록을 이용하여 연산 작용을 할 수 있도록 하였으며 블록의 수에 상관없이 연산속도의 차이가 없게 하였다. 또한 병렬처리(parallel processing)를 이용하여 많은 양의 멀티미디어 데이터를 고속으로 처리할 수 있도록 하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 기존에 연구된 재구성 가능한 구조의 코프로세서들에 대하여 고찰하고 Ⅲ장에서는 제안하는 코프로세서의 구조에 대하여 설명한다. Ⅳ장에서는 제안하는 코프로세서의 동작 및 연산 작용에 대하여 설명하고 Ⅴ장에서는 설계한 코프로세서의 실험 및 결과에 대하여 고찰한다. Ⅵ장에서는 제안된 코프로세서의 실제 멀티미디어 응용 방법 및 예에 대하여 기술하며 Ⅶ장에서 결론을 맺는다.

Ⅱ. 재구성 가능한 코프로세서에 대한 기존 연구

재구성 가능한 구조를 이용하여 코프로세서를 설계한 대표적인 예들을 살펴보면 다음과 같다.

J. R. Hauser와 J. Wawrzynek는 1997년에 coarse-grained 형태의 Garp라 명명된 재구성 가능한 코프로세서를 제안하였다[5]. Garp는 MIPS 프로세서와 연동하여 동작하도록 되어 있고 같은 형태의 UltraSparc 프로세서와 비교하여 응용분야에 따라 적게는 2배에서 크게는 24배에 가까운 속도를 낼 수 있다[6]. 또한 C언어를 이용하여 자동 컴파일(compile)이 가능하다[7][8]. 그러나 Garp는 고정된 형태의 구조를 갖고 있기 때문에 사용자가 요구하는 응용분야에 따른 변환이 쉽지 않고 2비트 단위로 구성이 되기 때문에 하드웨어의 구조가 복잡하다는 단점을 갖고 있다.

Takashi Miyamori와 Kunle Olukotun은 REMARC라 명명된 멀티미디어 전용 코프로세서를 제안하였다[9]. REMARC는 nano processor라 불리는 64개의 프로그래머블(programmable) 로직 블록들로 구성되어 있으며 16비트 단위로 데이터를 처리하도록 되어있다[10]. REMARC는 각각의 nano processor들 안에 16비트의 ALU(Arithmetic Logic Unit)를 포함하고 있기 때문에 하드웨어의 크기가 크다는 단점을 갖고 있으며 고정된 16비트를 갖고 있기 때문에 비트단위의 연산이 불가하다는 단점이 있다.

Ⅲ. 제안하는 코프로세서의 구조

1. Architecture Overview

그림 1은 제안하는 코프로세서가 일반적인 프로세서와 연동하여 동작하는 블록 다이어그램을 나타낸다. 제안하는 프로세서는 메인 프로세서로부터 연산에 필요한 데이터를 인계받아 자체 메모리에 저장 후 연산을 수행하여 연산결과를 메인 프로세서로 전송하도록 되어 있다. 이와 같은 동작을 위하여 제안하는 재구성 가능한 코프로세서는 자체에 데이터 전송 및 기본적인 제어 기능만을 수행하는 16비트 프로세서가 내장되어 있고 메모리 맵 IO(memory mapped IO) 방식을 사용하여 최종

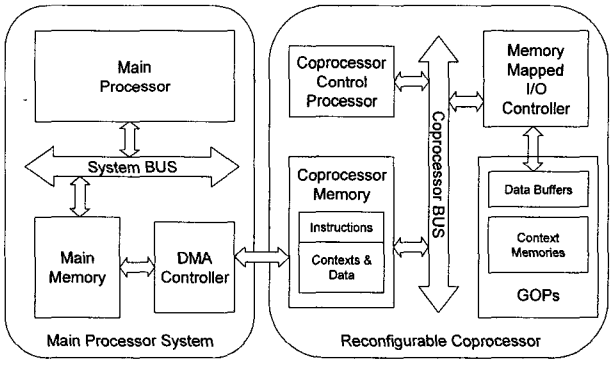


그림 1. 일반적인 프로세서와 연동되어 동작하는 제안하는 코프로세서의 블록 다이어그램

Fig. 1. Block Diagram of a conventional microprocessor system with proposed coprocessor.

연산을 수행하는 GOP(Group of Processing element)들에게 연산 데이터 및 재구성을 위한 콘텍스트(context)를 전송한다.

2. 제안하는 코프로세서의 구조

제안하는 재구성 가능한 코프로세서는 그림 2와 같이 크게 4개의 블록으로 구성되어 있다. 전체적인 연산 기능을 담당하는 GOP는 Memory Mapper를 통하여 자체 내장된 메모리로부터 재구성에 필요한 콘텍스트와 연산에 필요한 데이터를 인계 받는다. 인계된 데이터는 연산과정을 거쳐 그 결과를 자체 메모리 맵의 일정한 부분에 저장하게 된다.

하나의 GOP는 4×1 벡터간의 내적(inner product) 연산을 16비트 단위로 수행할 수 있다. 이때 곱셈기를 사용하지 않고 DA 연산을 이용하여 연산 작용을 수행한다. 따라서 멀티미디어 연산에서 가장 널리 사용되는 행렬의 곱셈 연산을 수행하기 위해서는 하나의 GOP를

이용하여 여러 번의 반복연산을 통해 모든 연산이 가능하다. 그러나 이와 같은 여러 번의 반복 연산은 연산속도를 저하시키기 때문에 제안하는 코프로세서는 응용분야에 따라 GOP의 개수를 가변적으로 구성할 수 있도록 하였으며 그림 3에 8개의 GOP를 사용하여 코프로세서를 구성한 예를 나타내었다. 여러 개의 GOP들을 사용할 경우 제안하는 코프로세서는 각각의 GOP들이 병렬처리 방식으로 연산이 수행이 되기 때문에 연산 속도에 큰 변화가 없이 연산을 수행할 수 있다. GOP에 재구성을 위한 콘텍스트 전송 및 연산 데이터의 전송은 코프로세서내의 RA Controller가 담당하며 이것은 16비트 크기의 마이크로프로세서 형태로 되어 있다.

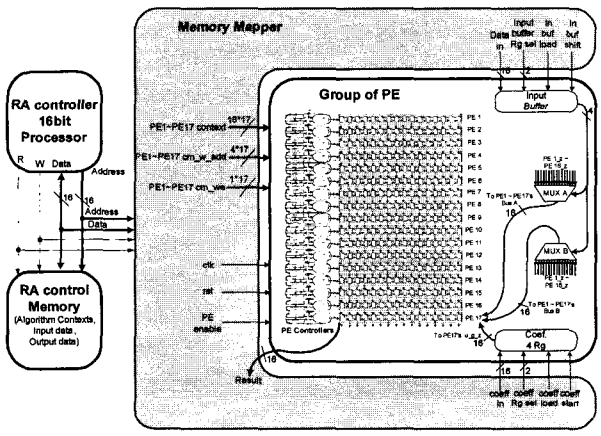


그림 2. 제안하는 코프로세서의 전체구조
Fig. 2. Overall architecture of proposed coprocessor.

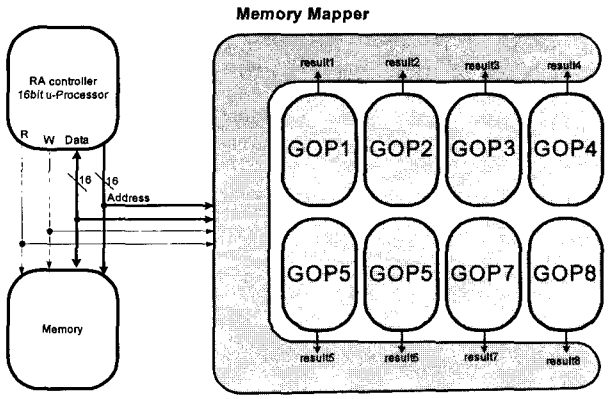


그림 3. 8개의 GOP를 사용한 코프로세서 구성 예
Fig. 3. Example of the proposed coprocessor with 8 GOPs.

일반적인 마이크로프로세서는 다양한 응용분야에 맞게 동작하기 위하여 많은 인스트럭션(instruction)과 복잡한 구조를 갖고 있으나 제안하는 코프로세서에서 사용한 마이크로프로세서는 데이터 전송 및 GOP간의 덧셈 연산을 주로 하기 때문에 총 8개의 인스트럭션과 다이렉트 어드레싱(direct addressing)만을 지원하는 간단한 구조로 설계하였다.

3. GOP의 구조

그림 4에 나타낸 바와 같이 GOP는 총 17개의 PE(Processing Element)들로 구성되어 있다. 하나의 PE는 16개의 RLC(Reconfigurable Logic Core)들로 구성되어 있다. 각각의 PE에는 그림 4와 같이 PE Controller가 부착되어 있어서 Context Memory에 저장된 내용에 따라 PE의 연산을 결정하는 재구성 작용을 하게 된다.

재구성을 위한 콘텍스트의 구조는 그림 5와 같으며 콘텍스트의 내용에 따라 논리 및 산술 연산을 수행할 수 있다. 최상위 비트가 0이면 PE는 논리 및 산술연산

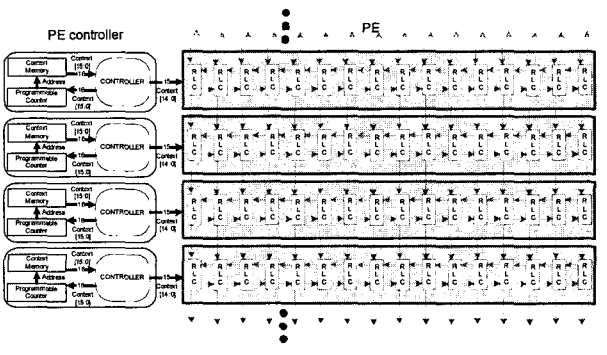


그림 4. GOP의 구조
Fig. 4. Block diagram of GOP.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	mul	Function configuration			sub	S reg control						B input control bits			
0	Serial Multiplier	Logic/Arithmetic Operation configuration			0 : + 1 : -	00: A 01: B 10 : Left S 11 : Static S		A Register Input Selection				B Register Input Selection			
1	Programmable operation Instruction Operation . Number of loop operation														

그림 5. 콘텍스트의 구조
Fig. 5. Structure of the context.

을 수행하고 1이면 콘텍스트의 재사용을 위해 메모리의 내에서 점프(jump) 및 루프(loop) 기능을 수행한다.

4. RLC의 구조

RLC는 그림 6과 같이 ALU와 이를 제어하기 위한 회로들로 구성된다. RLC는 16비트 크기의 A, B와 그림 5에 나타난 최상위 비트를 제외한 15비트의 콘텍스트를 입력으로 받는다.

A, B 입력은 표 1에 나타난 바와 같이 상하좌우의 RLC 및 내부 레지스터(register)로부터 피드백(feedback)된 입력을 나타내며 콘텍스트의 정보에 따라 MUX를 통해 필요한 1 비트의 입력을 받도록 하였다. 따라서 콘텍스트 입력을 제외한 RLC간의 연결은 그림 7과 같이 구성된다.

RLC의 연산의 종류는 그림 5에 나타난 콘텍스트에 따라 결정되며 그 결과는 Z 레지스터에 저장된다. 16 비트 입력 중 최종 비트의 결정은 A in sel과 B in sel이 결정하고 C in은 덧셈 연산 시에는 캐리입력을 나타낸다.

표 1. A, B 입력단의 구성요소
Table 1. Inputs components of A and B.

Bits	A input components	B input components
LSB	Left RLC's A reg.	Left RLC's B reg.
2nd bit	Right RLC's A reg.	Right RLC's B reg.
3rd bit	Reserved	Reserved
4th bit	Reserved	Reserved
5th bit	Internal A reg.	Internal B reg.
6th bit	Left RLC's Z reg.	Left RLC's Z reg.
7th bit	internal Z reg.	internal Z reg.
8th bit	GND	GND
9th bit	Upper RLC's A reg.	Upper RLC's B reg.
10th bit	Upper RLC's Z reg.	Lower RLC's Z reg.
11th ~ 15th bits	Reserved	Reserved
MSB	VDD	VDD

각각의 RLC는 1비트 단위의 연산을 수행하며 연산 결과를 주변의 RLC에게 전송할 수 있기 때문에 비트단위의 쉬프트 연산도 가능하다. 즉 제안된 구조는 fine-grained 재구성 가능한 구조의 주 장점인 비트 단위의 연산조작을 가능하게 하며 시스톨릭(systolic) 형태의 연산도 수행할 수 있다.

ALU는 비트단위의 연산을 수행하는 기본 블록으로 NOT, AND, NAND, OR, NOR, XOR, XNOR, ADDITION과 같은 8가지 연산을 수행한다. 연산의 종류는 f_con입력 신호에 따라 결정되며 minus_con 신호에 따라 뺄셈연산이 가능하도록 하였다.

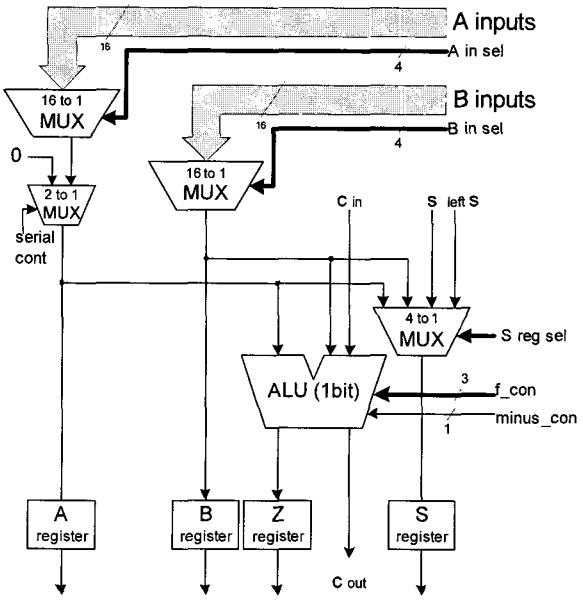


그림 6. RLC의 구조
Fig. 6. Block diagram of RLC.

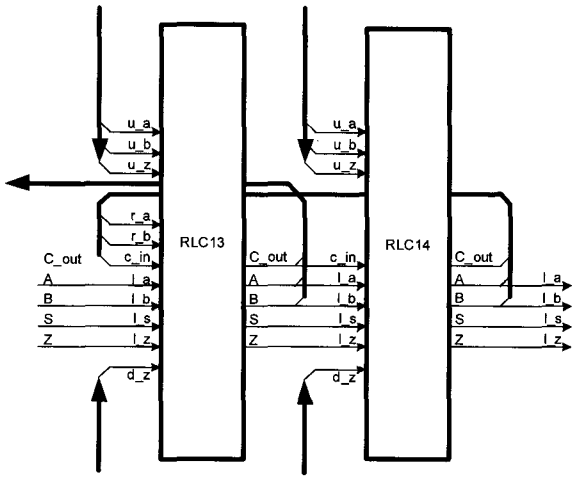


그림 7. RLC간의 연결
Fig. 7. Interconnection between RLCs.

S 레지스터는 제안하는 코프로세서의 비트시리얼 곱셈연산을 위하여 데이터의 임시 저장 역할을 한다.

5. RA Controller

RA Controller는 16비트 마이크로프로세서이다. 일반적인 마이크로프로세서는 많은 종류의 명령(instruction)을 갖는 복잡한 구조를 갖고 있다. 그러나 제안하는 코프로세서에서의 마이크로프로세서는 데이터의 전송, 제어 및 덧셈만을 수행하기 때문에 표 2와 같은 간단한 명령을 갖고 있는 마이크로프로세서를 설계하였다.

메인 프로세서로부터 전송되어온 연산에 필요한 데이터 및 콘텍스트는 RA Controller를 통하여 코프로세서 메모리에 저장이 된다. RA Controller는 다이렉트 어드레싱(direct addressing) 모드만을 지원하며 내부의 레지스터로서 accumulator만을 내장하고 있다.

6. Memory Mapper

RA Controller가 GOP들을 제어하는 방식은 메모리 맵 I/O방식을 사용하기 때문에 이를 제어하기 위하여 Memory Mapper를 설계하였다. Memory Mapper는 RA Controller가 메모리 맵 상의 특정 번지에 데이터를 쓰면 이를 해당되는 GOP로 전달하고 그 연산 결과를 메모리 맵 상에 저장하는 기능을 하며 Memory Mapper의 전체적인 입출력 관계를 그림 8에 나타내었다.

Memory Mapper는 RA Controller의 Address BUS로부터 특정한 번지에 들어오는 데이터를 각 GOP에 전달한다. 또한 내부에 메모리 맵 상의 제어 비트 정보를 판단하여 GOP의 연산 수행 시작과 종료를 결정하며 GOP로부터 연산된 결과 데이터를 내부의 레지스터에 저장함으로써 RA Controller가 그 결과를 메모리 맵을 이용하여 읽어낼 수 있도록 한다.

표 2. RA Controller의 명령
Table 2. Instructions of RA Controller.

Instructions	Operation
HLT	Halt
SIZ	Skip if zero
ADD	Add
AND	And
XOR	Exclusive OR
LDA	Load from memory to AC
STA	Store from AC to memory
JMP	*Jump to Address*

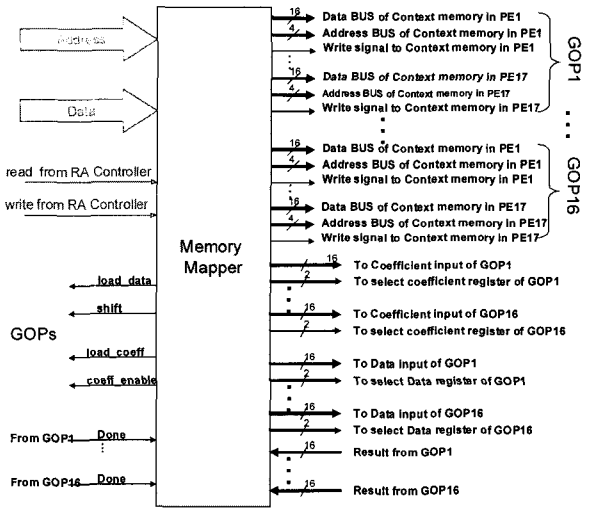


그림 8. Memory Mapper의 입출력
Fig. 8. Inputs and outputs of Memory Mapper.

Ⅳ. 제안하는 코프로세서의 연산 작용

본 장에서는 제안하는 구조의 코프로세서에서 수행되는 연산 수행 과정 및 여러 가지 연산의 적용 방법에 대하여 살펴본다.

1. 연산 수행 과정

제안하는 구조의 연산 수행은 크게 재구성 단계와 연산단계로 나뉜다. 재구성 단계에서는 연산에 필요한 재구성 콘텍스트들을 GOP에 인가하여 해당되는 연산을 수행할 수 있도록 재구성하는 단계로 RA Controller에서 순차적으로 콘텍스트들을 GOP에 전달한다. 또한 정해진 계수(coefficient) 값에 따른 행렬 연산 경우 각 계수 값도 미리 GOP에 전달한다.

연산단계는 전달된 피연산자들을 로딩(loading)하는 과정과 GOP에서 실제 연산을 수행하는 과정 및 결과를

표 3. 제안하는 구조의 기본 연산을 위한 비트할당
Table 3. Bit allocation for basic logical and arithmetic operation.

Function Configuration Bits	Sub Bit	Operation
000	N/A	NOT A
001	N/A	A AND B
010	N/A	A NAND B
011	N/A	A OR B
100	N/A	A NOR B
101	N/A	A XOR B
110	N/A	A XNOR B
111	1	Addition
	0	Subtraction

저장하는 과정으로 나누어진다. 전달된 피연산자들은 직렬로 4개의 입력 버퍼에 전송이 되며 전송된 RA Controller로부터 연산 시작 명령을 받아 연산을 수행한다. 연산이 끝나면 그 결과는 RA Memory에 순차적으로 저장이 된다.

2. 논리 연산과 덧셈 및 뺄셈

제안하는 구조는 7가지의 논리연산과 덧셈 및 뺄셈을 수행할 수 있으며 이를 표 3에 나타내었다. 덧셈의 경우 맨체스터 캐리 체인(Manchester Carry Chain)[4] 방식의 덧셈을 수행할 수 있다. 그림 6의 RLC안의 MUX를 통하여 캐리를 전달할 수 있기 때문에 16비트의 덧셈이 가능하며 뺄셈의 경우 반전된 입력 값을 통해 계산 후 결과를 다시 반전하여 뺄셈의 결과를 얻을 수 있도록 하였다.

3. 곱셈 및 내적

제안하는 구조는 그림 5의 콘텍스트의 내용에 따라 RLC로 들어오는 입력 및 나가는 출력을 자유자재로 조절할 수 있기 때문에 곱셈의 경우 쉬프트 후 덧셈하는 방식을 사용하여 곱셈이 가능하다.

하나의 PE는 곱셈기 없이 곱셈 연산을 할 수 있으나 내적과 같이 반복된 곱셈과 덧셈의 연산은 처리속도를 느리게 할 수 있다. 따라서 제안하는 구조는 17개의 PE를 사용하였으며 이를 이용하여 16비트 크기의 4×1 벡터간의 내적 연산을 수행하도록 하였다.

내적 연산은 DA 알고리즘을 사용하여 연산을 수행한다. PE1부터 PE16까지는 DA연산에 필요한 Look-up Table을 병렬처리로 생성하며 PE17에서 결과를 생성하여 최종 내적 계산이 수행되어진다.

5. 행렬-벡터 곱셈

멀티미디어 데이터 처리는 대부분은 행렬-벡터 연산에 기초하고 있기 때문에 이를 고속으로 수행하기 위하여 제안하는 구조는 여러 개의 GOP를 사용할 수 있도록 하였다.

예를 들어 식 (1)과 같은 4×4 행렬과 4×1 벡터의 곱셈은 결과를 요소별로 풀어쓰면 식 (2)과 같다.

$$\begin{bmatrix} c_{00} c_{01} c_{02} c_{03} \\ c_{10} c_{11} c_{12} c_{13} \\ c_{20} c_{21} c_{22} c_{23} \\ c_{30} c_{31} c_{32} c_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} \quad (1)$$

$$\begin{aligned} X_0 &= c_{00}x_0 + c_{01}x_1 + c_{02}x_2 + c_{03}x_3 \\ X_1 &= c_{10}x_0 + c_{11}x_1 + c_{12}x_2 + c_{13}x_3 \\ X_2 &= c_{20}x_0 + c_{21}x_1 + c_{22}x_2 + c_{23}x_3 \\ X_3 &= c_{30}x_0 + c_{31}x_1 + c_{32}x_2 + c_{33}x_3 \end{aligned} \quad (2)$$

식 (2)에서와 같이 각각의 요소의 결과는 내적과 같으므로 GOP하나를 사용할 경우 4번의 반복 연산이 필요하다. 그러나 제안하는 구조는 각각의 GOP가 병렬로 처리되도록 하였기 때문에 4개의 GOP를 사용할 경우 하나의 내적을 계산할 때와 동일한 속도로 연산결과를 얻을 수 있다.

8×8 행렬과 8×1벡터의 곱셈과 같이 GOP 하나가 처리할 수 없는 경우 식 (3)과 같이 분할행렬을 사용하여 두개의 행렬-벡터 곱셈의 덧셈으로 그 결과를 얻을 수 있고 16개의 GOP를 사용할 경우 덧셈의 각항의 결과는 동시에 출력되기 때문에 덧셈을 제외한 시간은 요소의 수에 상관없이 동일하다.

$$\begin{bmatrix} C_{00} C_{01} C_{02} C_{03} C_{04} C_{05} C_{06} C_{07} \\ C_{10} C_{11} C_{12} C_{13} C_{14} C_{15} C_{16} C_{17} \\ C_{20} C_{21} C_{22} C_{23} C_{24} C_{25} C_{26} C_{27} \\ C_{30} C_{31} C_{32} C_{33} C_{34} C_{35} C_{36} C_{37} \\ C_{40} C_{41} C_{42} C_{43} C_{44} C_{45} C_{46} C_{47} \\ C_{50} C_{51} C_{52} C_{53} C_{54} C_{55} C_{56} C_{57} \\ C_{60} C_{61} C_{62} C_{63} C_{64} C_{65} C_{66} C_{67} \\ C_{70} C_{71} C_{72} C_{73} C_{74} C_{75} C_{76} C_{77} \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{bmatrix} = \begin{bmatrix} C_{00} C_{01} C_{02} C_{03} \\ C_{10} C_{11} C_{12} C_{13} \\ C_{20} C_{21} C_{22} C_{23} \\ C_{30} C_{31} C_{32} C_{33} \\ C_{40} C_{41} C_{42} C_{43} \\ C_{50} C_{51} C_{52} C_{53} \\ C_{60} C_{61} C_{62} C_{63} \\ C_{70} C_{71} C_{72} C_{73} \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} + \begin{bmatrix} C_{04} C_{05} C_{06} C_{07} \\ C_{14} C_{15} C_{16} C_{17} \\ C_{24} C_{25} C_{26} C_{27} \\ C_{34} C_{35} C_{36} C_{37} \\ C_{44} C_{45} C_{46} C_{47} \\ C_{54} C_{55} C_{56} C_{57} \\ C_{64} C_{65} C_{66} C_{67} \\ C_{74} C_{75} C_{76} C_{77} \end{bmatrix} \begin{bmatrix} x4 \\ x5 \\ x6 \\ x7 \end{bmatrix} \quad (3)$$

V. 실험 및 성능평가

1. 소프트웨어 실험 환경

RA Controller의 소프트웨어는 그림 9와 같이 어셈블러를 사용하여 프로그램을 만든 후 자체 제작된 어셈블러를 이용해서 실행 가능한 기계어로 변환하여 실험하였다.

그림 9에 나타낸 바와 같이 제어를 위한 소프트웨어와 콘텍스트 및 연산에 필요한 데이터를 자체 제작된 어셈블러를 통하여 컴파일 한 후 최종 기계어 코드를 생성해 내었으며 생성된 코드를 그림 10과 같이 RA Control Memory상에 장착하여 실험하였다.

RA Control Memory상의 데이터는 Instruction에 따라 우측의 메모리 맵(Memory Map)에 이동되고 이동된 메모리 맵의 데이터는 Memory Mapper를 통해 해당 GOP 블록으로 전송되어 진다. 연산이 완료된 결과는 GOP로부터 메모리 맵의 Outputs 번지에 결과가 저장되어 지며 저장된 결과는 RA Controller에 의해 Output Data 번지에 순차적으로 저장되게 하였다.

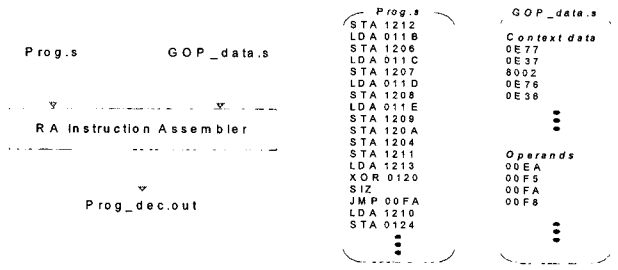


그림 9. 소프트웨어 환경 및 예
Fig. 9. Software environment and example.

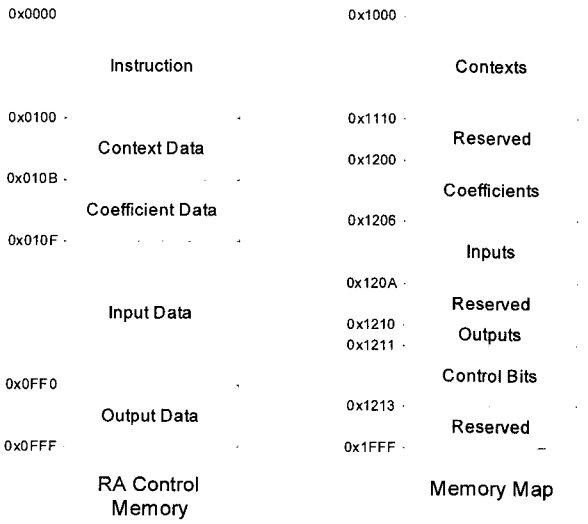


그림 10. 하나의 GOP를 사용한 경우의 메모리 맵 구조
Fig. 10. Structure of memory map with one GOP.

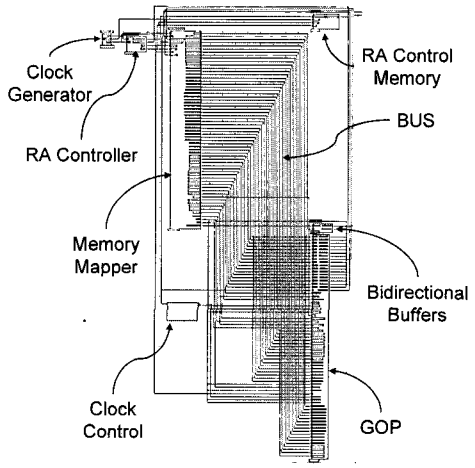


그림 11. 제안된 코프로세서의 구현
Fig. 11. Implementation of proposed coprocessor.

2. 하드웨어 실험 환경

제안된 구조는 Aldec사의 Active-HDL과 VHDL을 이용하여 설계하였고 완성된 회로를 그림 11에 나타내었다.

설계는 각 세부 블록들을 VHDL로 코딩 후 시뮬레이션을 수행하였고 최종적으로 Active-HDL의 BDE (Block Diagram Editor)를 이용하여 전체회로를 구성하였다. 구성된 회로로부터 VHDL 코드를 추출하여 전체 시뮬레이션을 수행하였다.

3. 연산속도 고찰

4개의 요소를 갖는 두 벡터의 스칼라 곱 연산을 하나의 GOP를 사용한 경우에 소요되는 전체 연산시간을 그림 12에 나타내었다. 코프로세서 재구성으로부터 연산결과가 최종 메모리까지 저장되는 시간은 100MHz 클록으로 시뮬레이션 한 결과 7210ns였다. 그러나 다수의 연산을 반복할 경우 제안하는 구조는 한 번의 재구성으로 반복된 연산이 가능하기 때문에 실제 연산 소요시간은 740ns에 불과하다. 따라서 반복된 연산 작용이 주를 이루는 멀티미디어 연산처리에 적합함을 알 수 있다.

또한 제안된 코프로세서는 그림 13과 같이 여러 개의 GOP를 사용하여 연산속도를 줄일 수 있다. 그림 13에 나타난 연산속도는 8×8 행렬과 8×1벡터의 곱셈을 GOP의 개수에 따라 연산결과를 측정한 것이다. 즉, GOP를 많이 사용할수록 연산속도를 감소시킬 수 있다.

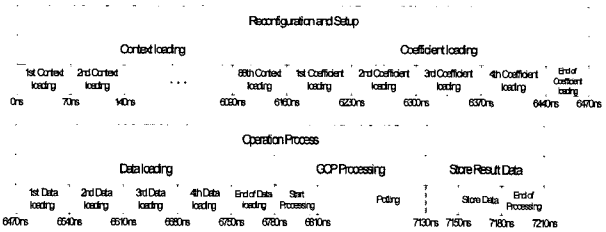


그림 12. 100MHz 클록을 사용한 경우의 전체 연산시간
Fig. 12. Total operation time with 100MHz clock frequency.

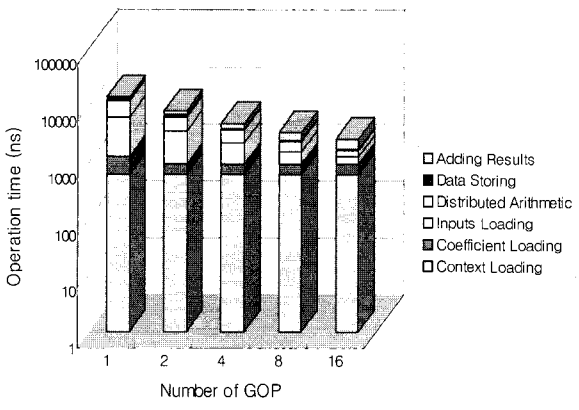


그림 13. GOP 개수에 따른 연산속도 비교
Fig. 13. Comparison of operation time according to number of GOP.

표 4. 기존의 재구성 가능한 코프로세서와의 구조비교
Table 4. Architecture comparison between previous coprocessor and proposed coprocessor.

	Proposed	REMARC	Garp
Main Processor Interconnection Topology	Coupled to I/O System BUS	Coupled to Local BUS	Coupled to CPU
Num. of Processing Elements	Variable Small (1,2,4,8,16 GOPs)	Fixed Medium (64 processors)	Very Large (at least 768 processors)
Processor Size	Medium	Very Large	Medium
Multiplier	Non-exist	16bit Multiplier	Non-exist
Reconfiguration	Dynamic	Static	Static
Execution Control	Controlled by instruction	Controlled by instruction	Controlled by instruction
Control Mechanism	Based on Microprocessor	Hard-wired Control Logic	Heuristic routing
Interconnection	Coprocessor BUS	4 neighbors, VBUS and HBUS	Vertical wires and Horizontal wires
Target Application	Multimedia processing for Mobile Application	Acceleration of Multimedia Processing	Loop acceleration
Bit manipulation	Possible	Impossible	Partially possible
Application domain	Embedded Application	Data-parallel applications	Bit-level computations
Programming Environment	Simple Assembler	Global and Nano Assembler	Garpcc (C-based compiler)

4. 구조 비교

제안된 코프로세서와 기존에 발표된 재구성 가능한 코프로세서의 구조적인 차이점 및 장단점을 비교하여 표 4에 나타내었다.

먼저 제안된 코프로세서는 기존의 코프로세서와는 달리 메인 프로세서의 시스템 버스를 이용하기 때문에 모든 프로세서와의 연결이 가능하다는 장점이 있다.

Garp의 경우 MIPS 프로세서에 국한되어 사용되기 때문에 다른 종류의 프로세서와의 연결이 불가능하며 REMARC 또한 메인 프로세서의 지역버스(local bus)를 이용하기 때문에 융통성 측면에서 단점을 포함하고 있다. 또한 제안된 코프로세서는 응용분야에 따라 코프로세서의 크기를 결정할 수 있고 곱셈기를 사용하지 않는 점은 하드웨어 크기 및 전력 소모 면에서 큰 장점이며 마이크로프로세서를 이용한 제어 기법은 다양한 응용분야에 확대적용이 가능함을 나타낸다. 따라서 제안된 코프로세서는 멀티미디어용 모바일 단말기에 적합함을 알 수 있다.

VI. 제안된 코프로세서의 응용 예

본 장에서는 제안된 코프로세서를 사용하여 2차원 멀티미디어 연산에서 널리 사용되고 있는 DCT (Discrete Cosine Transform) 연산을 수행하는 방법 및 속도에 대하여 고찰한다.

1. 2차원 DCT

2차원 DCT를 수식으로 나타내면 식 (4)와 같다.

$$X(u,v)=\frac{2}{N}C(u)C(v)\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}x(i,j)\cos\left(\frac{(2i+1)u\pi}{2N}\right) \quad (4)$$
$$where,\ C(0)=1/\sqrt{2},\ C(u)=C(v)=1\ for\ u,v\neq 0$$

실제 식 (4)의 연산은 연산 시 많은 데이터를 처리해야 하기 때문에 연산소요시간이 길다. 따라서 본 응용에서는 Chen-DCT[4]를 사용하였으며 8×8 윈도우(window)를 사용한 경우의 Chen-DCT는 식 (5), (6)과 같다.

$$\begin{bmatrix}X_0\\X_2\\X_4\\X_6\end{bmatrix}=\frac{1}{2}\begin{bmatrix}A & A & A & A\\B & C & -C & -B\\A & -A & -A & A\\C & -B & B & -C\end{bmatrix}\begin{bmatrix}x_0+x_7\\x_1+x_6\\x_2+x_5\\x_3+x_4\end{bmatrix} \quad (5)$$

$$\begin{bmatrix}X_1\\X_3\\X_5\\X_7\end{bmatrix}=\frac{1}{2}\begin{bmatrix}D & E & F & G\\E & -G & -D & -F\\F & -D & G & E\\G & -F & E & -D\end{bmatrix}\begin{bmatrix}x_0-x_7\\x_1-x_6\\x_2-x_5\\x_3-x_4\end{bmatrix} \quad (6)$$

$$where, A=\cos(\pi/4),\ B=\cos(\pi/8),\ C=\sin(\pi/8),$$
$$D=\cos(\pi/16),\ E=\cos(3\pi/16),\ F=\sin(3\pi/16),\ G=\sin(\pi/16)$$

2. 제안된 코프로세서의 적용

2차원 DCT를 수행하기 위하여 1차원 DCT를 한 후 그 결과 행렬을 전치하여 다시 1차원 DCT를 수행하여 최종 결과를 얻어 내었다. 이때 제안된 구조는 행렬의

표 5. 하나의 블록에 대한 연산속도
Table 5. Operation cycle time per block.

8×8 Block DCT Operations	Cycle
Context Loading	616
Coefficient Loading(32 coeff.)	248
1D-DCT(N=8,Chen) 16 times	3040
Transpose Memory (64 data)	448
Total	4352

표 6. 640×480 영상의 DCT 연산 시 소요시간
Table 6. Operation cycle time for 640×480 image.

640*480 Image Operations	Cycle
Context Loading	616
Coefficient Loading(32 coeff.)	248
DCT Operations of 640×480	16742400
Total	16743184

표 7. 상용프로세서와의 연산속도 비교
Table 7. Time comparison between ARM processor and proposed coprocessor.

Comparison	Computing Time (sec@25MHz working frequency)		
	pixel (600×400)	pixel (800×600)	pixel (1024×768)
ARM922T (pure software)	11.9	23.6	39.6
ARM922T + AHB DCT/IDCT core [11]	5.3	10.7	17.6
Proposed coprocessor	0.52	1.04	1.49

전치를 단순히 코프로세서내의 메모리 데이터의 재배열만으로 가능하며 하나의 블록을 연산한 속도를 표 5에 나타내었다. 표 5의 결과는 8개의 GOP를 이용하여 속도를 계산하였으며 SVGA 영상(640×480) 전체에 대하여 DCT를 수행할 경우 소요되는 연산을 평가한 결과 표 6과 같은 시간이 소요되었다.

제안된 코프로세서의 성능을 비교하기 위하여 멀티미디어 무선단말기에서 널리 사용되고 있는 상용 프로세서인 ARM922T 및 논문 [11]에서 제안한 고속 DCT 코어(core)와 연산속도를 측정하여 제안된 코프로세서와 비교하였으며 동일하게 25MHz의 클록 주파수로 동작했을 때 소요되는 시간을 측정한 결과 표 7과 같으며 제안된 구조는 기존의 프로세서에 비해 월등히 우수한 결과를 나타냄을 확인하였다.

Ⅶ. 결론 및 향후 연구과제

본 논문은 멀티미디어 무선단말기에 적합한 재구성 가능한 코프로세서를 설계하였다. 설계된 코프로세서는 응용분야에 따라 그 크기 및 연산방식을 달리할 수 있으며 곱셈기를 사용하지 않고 빠른 속도의 연산이 가능하다. 또한 기존의 코프로세서에 비해 하드웨어의 크기가 작고 시스템 버스를 사용하여 메인 프로세서와 연결하기 때문에 대부분의 프로세서와 연결이 가능하여 멀티미디어용 무선 단말기에 적합하다.

차후 연구과제로는 순차적인 메모리 데이터의 전송을 병렬로 처리하여 제안된 코프로세서에서 발생하는 레이턴시(latency)를 최대한으로 줄이는 연구를 진행할 예정이다.

참 고 문 헌

[1] Katherine Compton and Scott Hauck, "Reconfigurable computing: a survey of systems and software", ACM Computing Surveys (CSUR), Vol. 34, Issue 2, pp. 171-210, 2002.

[2] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiD-recongurable pipelined datapath", Proc. of International Workshop on Field Programmable Logic and Applications, pp. 126-135, 1996.

[3] H. Singh, Ming-Hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho., "MorphoSys: a Reconfigurable Architecture for multimedia applications", Proc. of IEEE Integrated Circuit Design, pp. 134-139, 1998.

[4] Mahesh Mehendale, Sunil D. Sherlekar, *VLSI SYNTHESIS OF DSP KERNELS*, Kluwer Academic Publishers, 2001.

[5] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Co-processor", Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines, pp. 12-21, 1997.

[6] J. R. Hauser, *Augmenting a Microprocessor with Reconfigurable Hardware*, Ph. D thesis of Univ. of California, Berkeley, Fall 2000.

[7] T. J. Callahan, J. R. Hauser, J. Wawrzynek, "The Garp architecture and C compiler Computer", Computer, Vol. 33, Issue 4, pp. 62-69, 2000.

[8] Tim Callahan, "Kernel formation in GarpccField-Programmable Custom Computing Machines", Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines, pp.

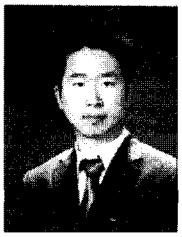
308-309, 2003.
[9] Takashi Miyamori and Kunle Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor", FPGA'98 (as poster paper), 1998
[10] Takashi Miyamori and Kunle Olukotun, "A quantitative analysis of reconfigurable coprocessors for multimedia applications," Proc. of FPGAs for Custom Computing Machines, pp. 15-17, 1998.
[11] Kai-Yuan Jan, Chih-Bin Fan, An-Chao Kuo, Wen-Chi Yen, Youn-Long Lin,"A platform based SOC design methodology and its application in image compression", International Journal of Embedded Systems, Vol. 1, No. 1-2, pp. 23-32, 2006.

저 자 소 개

김 남 섭(정회원)
1990년 경희대학교 전자공학과 학사
1992년 경희대학교 전자공학과 석사
2006년 경희대학교 전자공학과 박사
2007년～현재 한림대학교 전자공학과 강의전임교수
<주관심분야 : SoC 설계 및 테스트, 멀티미디어 ASIC, Computer Vision, Ubiquitous Health Care>

금 민 하(학생회원)
2006년 경희대학교 전자공학과 학사
2006년～현재 경희대학교 전자공학과 석사과정
<주관심분야 : VLSI 회로설계, 컴퓨터시스템>

조 원 경(정회원)
1971년 경희대학교 전자공학과 학사
1973년 한양대학교 전자공학과 석사
1986년 한양대학교 전자공학과 박사
1980년～현재 경희대학교 전자공학과 정교수
<주관심분야 : VLSI설계, 마이크로프로세서>



이 상 훈(학생회원)
2006년 경희대학교 전자공학과 학사
2006년～현재 경희대학교 전자공학과 석사과정
<주관심분야 : VLSI 회로설계, Reconfigurable Architecture>

김 진 상(정회원)
1985년 경희대학교 전자공학과 학사
1987년 경희대학교 전자공학과 석사
2000년 콜로라도 주립대 전기 및 컴퓨터공학과 박사
2001년～현재 경희대학교 전자공학과 조교수
<주관심분야 : 차세대 통신시스템, SoC 설계>