논문 2010-47SD-7-11

SDR용 기저대역 프로세서를 위한 프로그래밍 모델

(Programming Model for SODA-II: a Baseband Processor for Software Defined Radio Systems)

이 현 석*, 이 준 환**, 오 혁 준*

(Hyunseok Lee, Joonhwan Yi, and Hyukjun Oh)

요 약

이 논문은 SDR 시스템용으로 개발된 기저대역 프로세서인 SODA-II[1]를 활용하는데 필요한 프로그래밍 모델에 대한 것이다. SODA-II는 4개의 프로세서로 구성되는 멀티코어 시스템으로 한 코어에는 SIMD 데이터패스와 직렬 데이터패스가 모두 구현되어 있어 벡터 연산과 직렬 연산이 혼재하는 기저대역 신호처리 동작에 적합하다[2]. SODA-II에 대한 프로그래밍 모델은 C 언어 라이브러리 형태를 가진다. 라이브러리 함수가 SODA-II의 SIMD 데이터패스를 구동시키는데 필요한 세부적인 제어동작을 모두 처리하므로 사용자는 SIMD 데이터패스 구조에 대한 자세한 이해 없이 기저대역 신호처리 알고리즘을 구현할 수 있다. 이 논문에서는 기저대역 신호처리의 핵심 연산들이 SODA-II에서 어떤 형태로 구현되는지 설명하고 응용의 예로 W-CDMA 다중 경로 탐색기와 OFDM 복호기 동작을 SODA-II에서 구현한 결과를 살펴본다.

Abstract

This paper discusses the programming model of SODA-II that is a baseband processor for software defined radio (SDR) systems. Signal processing On-Demand Architecture II (SODA-II) is an on-chip multiprocessor architecture consisting of four processor cores and each core has both an wide SIMD datapath and a scalar datapath. This architecture is appropriate for baseband processing that is a mixture of vector computations and scalar computations. The programming model of the SODA-II is based on C library routines. Because the library routines hide the details of complex SIMD datapath control procedures, end users can easily program the SODA-II without deep understanding on its architecture. In this paper, we discuss the details of library routines and how these routines are exploited in the implementation of baseband signal processing algorithms. As application examples, we show the implementation result of W-CDMA multipath searcher and OFDM demodulator on the SODA-II.

Keywords : Baseband Processor, Software Defined Radio (SDR), Low Power, Digital Signal Processing (DSP)

I. 서 론

SDR (Software Defined Radio)은 무선통신 시스템을 동일한 하드웨어 플랫폼 상에서 소프트웨어 변경만으로 다수의 통신규격을 지원할 수 있게 하는 기술이다. 당초 군용 기술로 개발되었으나 이동통신 규격의 수가 증가함에 따라 하나의 기기로 여러 종류의 통신방식을 지원하도록 하기 위해서 상용화 연구가 진행되어왔다.

SDR 방식의 단말기나 기지국을 개발하는데 있어서 핵심이 되는 문제 중 하나는 특정 통신 규격에 특화된

* 정회원, 광운대학교 전자통신공학과
(Dept. of Electronics and Communications Engineering, Kwangwoon Univ.)
** 정회원, 광운대학교 컴퓨터공학과
(Dept. of Computer Engieering, Kwangwoon Univ.)
※ 이 논문은 서울시 산학연협력사업(10560), 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업(정보통신)[KI002145, 차세대 광통신용 디지털 신호처리 기반 초고속 CMOS회로 설계 기술], 교육과학기술부/한국연구재단 기초연구 사업(과제번호 2009-0088455), 반도체설계교육센터 (IDEC)의 지원을 받아 수행되었다.
접수일자: 2010년2월28일, 수정완료일: 2010년7월1일

부품들을 여러 종류의 통신 규격을 지원할 수 있도록 만드는 것이며 특히 ASIC (Application Specific Integrated Circuit) 형태로 구현되는 RF (Radio Frequency) 신호처리부와 기저대역 (Baseband) 신호처리부가 주된 연구 대상이다. RF 신호처리부와 기저대역 신호처리부는 요구되는 연산량이 매우 높으면서 소비전력에 대한 제약도 동시에 높아서 프로세서와 (Processor) 같은 프로그램 가능한 형태의 하드웨어로 구현하는 것이 매우 어렵다. 이들 RF 신호처리부와 기저대역 신호처리부 모두 SDR 시스템을 구현하는데 있어서 매우 중요하지만 이 논문은 기저대역 신호처리부만을 논의의 대상으로 한다.

SODA-II (Signal Processing On-Demand Architecture - II)는 SDR용 통신시스템의 기저대역 신호처리를 목적으로 설계된 프로세서이다[1~2]. 이 프로세서는 4개의 프로세서 코어들로 구성되며 각 코어에는 벡터(Vector) 연산에 적합한 SIMD (Single Instruction Multiple Data) 데이터패스 (Datapath)와 일반적인 제어연산에 적합한 스칼라 데이터패스 (Scalar datapath)가 모두 구현되어 있다. 이와 같은 코어의 구조는 널리 사용되는 TDMA (Time division multiple access), CDMA (Code division multiple access), OFDMA (Orthogonal frequency division multiple access) 방식의 통신시스템들의 기저대역 신호처리부의 연산 특성을 분석한 결과를 바탕으로 설계된 것이다[3].

SIMD 구조는 연산량과 소비전력 측면에서는 매우 효율적이지만 적합한 프로그래밍 환경이 존재하지 않는 경우 사용이 어려운 단점이 있다. 이는 사용자가 SIMD 데이터패스의 구조와 동작방식을 세부적으로 이해해야 최대의 성능을 얻을 수 있기 때문이다. 이런 이유로 SIMD 형태의 데이터패스를 가지는 프로세서를 위해서 다양한 형태의 컴파일러들이 제안되었다[4~6]. 하지만, 이 논문에서 다루는 SODA-II의 경우 SIMD 데이터패스에 기저대역 신호처리를 위해 복합 명령어 (Instruction chaining), SIMD 파이프라인 (SIMD pipelining), 순차적 메모리 접근 (Staggered memory access), 멀티사이클링 (Multicycling) 등과 같은 차별화된 기능들이 추가되어 효과적인 컴파일러의 개발이 용이하지 않았다.

이 논문에서는 새로운 컴파일러를 개발하는 대신에 일반적인 GPP (General Purpose Processor)용 C 컴파일러를 이용해서 SODA-II의 SIMD 데이터패스용 실행 프로그램을 효과적으로 개발하는 방안을 제안한다. 제안되는 방식은 SIMD 데이터패스의 동작을 정밀하게 제어하는 부분은 C 라이브러리 함수 형태로 구현하고 사용자는 라이브러리 함수들에 적정한 입력 파라미터를 입력하여 사용하는 형태를 가진다. 이는 신호처리 알고리즘 개발에 많이 사용되는 Matlab 환경에서 사용자가 모든 기능을 직접 구현하는 대신에 기본적으로 제공되는 신호처리 함수들을 이용해 시스템의 동작을 구현할 수 있는 것과 같은 원리이다. SODA-II에서는 이와 같은 형태의 접근 방법을 적용하여도 라이브러리의 개발과 관리에 어려움이 크지 않다. 그 이유는 [2]에서 보인 것과 같이 SODA-II의 SIMD 데이터패스에서 처리되는 벡터 연산의 종류가 10개 내외로 많지 않기 때문이다.

제안하는 프로그래밍 모델의 적정성을 입증하기 위해서 구현의 예로 기저대역 신호처리부의 동작에서 빈번하게 사용되고 그 구조가 간단하여 설명이 용이한 상관도 (Correlation) 계산 라이브러리 함수와 그 사용법을 상세히 살펴본다. 또한 응용의 예로 W-CDMA 수신기의 다중경로 탐색기와 (Multipath searcher) OFDM 복조기의 구현 결과를 보여서 제안하는 프로그래밍 모델의 구현 용이성과 효율성을 입증한다.

II. 본 론

1. SODA-II

가. 구조

SODA-II는 벡터연산이 주를 이루는 기저대역 신호처리에 적합하도록 설계되어 있다. SODA-II는 그림 1에서 보인 것과 같이 4개의 프로세서 코어들과, 1개의 GPP, 1개의 공유메모리 (Shared memory) 로 구성되어 있다. 4 개의 프로세서 코어들에는 기저대역 신호처리 연산들이 할당되고 GPP에는 관리 동작들이 할당된다. 공유메모리는 프로세서 코어들 사이의 통신과 대량의 데이터 저장에 사용된다.

한 프로세서 코어는 SIMD 데이터패스, 스칼라 데이터패스, 명령어 메모리, 데이터 메모리, 시스템 제어장치, 버스 정합장치 등의 6가지 장치로 구성된다. SIMD 데이터패스는 벡터 연산을 담당하고 스칼라 데이터패스는 병렬처리가 어려운 순차적인 연산을 담당한다. 시스템 제어장치는 명령어 분석기, 인터럽트 제어기, 프로그램 카운터로 구성되며 SIMD 데이터패스와 스칼라 데

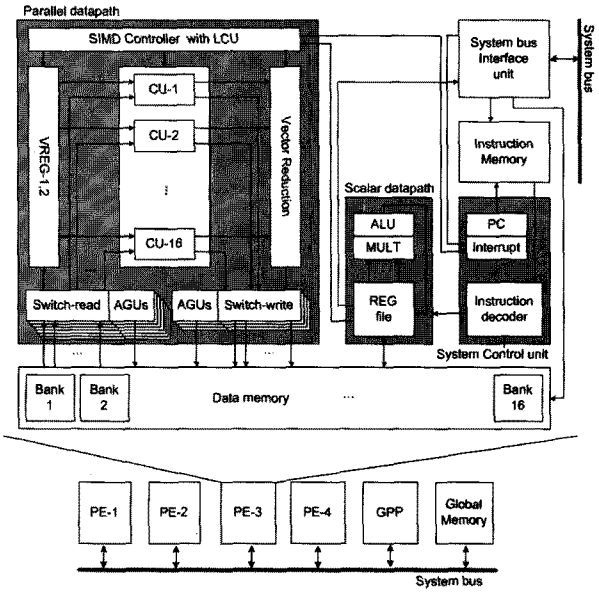


그림 1. SODA-II의 구조
Fig. 1. Structure of SODA-II.

이터패스를 모두 제어의 대상으로 한다. 버스 정합장치는 다른 프로세서 코어들과의 통신을 위해 사용된다. 명령어 메모리와 데이터 메모리는 각각 명령어와 사용자 데이터의 저장에 사용된다. 데이터 메모리는 내부에 16개의 서브뱅크(sub-bank)를 가지고 있어서 최대 16개의 데이터를 동시에 읽거나 쓸 수 있다.

SIMD 데이터패스는 2개의 벡터 레지스터(Vector register), 16개의 연산장치, 1개의 벡터-스칼라 변환 장치 (Vector reduction unit), SIMD 제어장치, 11개의 주소 생성기 (AGU: address generation unit)와, 1 개의 루프 제어 장치 (LCU: Loop control unit), 11개의 읽기/쓰기 스위치들로 구성되어 있다. 이들은 필요한 경우

표 1. 시스템 제어장치가 SIMD 제어 장치로 전달하는 명령어들
Table 1. Instructions transferred from system control unit to SIMD control unit.

Instruction	Description
FIR	Finite impulse response filter
COR	Correlation computation
MIN	Minimum value searching
MAX	Maximum value searching
Pattern	Pattern matching
FFT	Fast fourier transform
BMC	Viterbi branch metric computation
ACS	Viterbi add compare selection
VREGLD	Vector register load

동시에 동작될 수 있으며 파이프라인을 (Pipeline) 형성한다. 벡터 레지스터는 벡터 연산 장치에 입력 데이터를 공급하고 이들의 연산 결과를 임시 저장하는 역할을 수행한다. 벡터-스칼라 변환 장치는 벡터 연산 장치의 연산 결과를 처리하여 스칼라 형태의 데이터로 변환한다. SIMD 제어장치는 SIMD 데이터패스의 동작에 필요한 제어 신호를 생성한다. 시스템 제어장치는 표 1에 정리된 것 같은 간단한 명령어들을 이용해서 SIMD 데이터패스에서 수행될 명령어를 SIMD 제어장치에 알린다. SIMD 제어장치는 이 명령어를 이용해서 표 2에 정리된 것과 같은 보다 상세한 SIMD 명령어들을 생성시켜 SIMD 연산장치에서 목적하는 알고리즘에 대한 처리가 이루어지도록 한다. 이는 CISC (Complex instruction set computer) 컴퓨터의 한 명령어가 다수의 마이크로 명령어 (micro instruction) 들을 이용하여 구현되는 것과 유사한 원리이다. AGU는 데이터 메모리에서 데이터를 읽고 쓸 때 필요한 주소를 자동으로

표 2. SIMD 제어장치가 SIMD 데이터패스를 제어하는 명령어들
Table 2. Instructions for SIMD datapath control.

Instruction	Description
DCCAD RD,R1,R2,R3,R4	Double conditional complements and addition: T1=R1 ? R2:-R2; T2=R3 ? R4:-R4; RD = T1+T2
DMLAD RD,R1,R2,R3,R4	Double multiplication and addition: T1=R1XR2; T2=R3XR4; RD=T1+T2
DMLSB RD,R1,R2,R3,R4	Double multiplication and subtraction: T1=R1XR2; T2=R3XR4; RD=T1-T2
TMIN RD,R1,R2,R3,R4	Triple min. value search: T1=(R2>R1)?R1:R2; T2=(R4>R3)?R3:R4; RD=(T2>T1)?T1:T2
TMAX RD,R1,R2,R3,R4	Triple max. value search: T1=(R2>R1)?R2:R1; T2=(R4>R3)?R4:R3; RD=(T2>T1)?T2:T1
DBMCA RD,R1,R2,R3,R4	Double branch metric computations and addition: T1=abs(R1-R2); T2=abs(R3-R4); RD=T1+T2
BMCA RD,R1,R2,R3	Branch metric computation and addition: T1=abs(R1-R2); RD=T1+R3
BMC RD,R1,R2	Branch metric computation: RD=abs(R1-R2)
ACS RD,R1,R2,R3,R4	Add, compare, and select: T1=R1+R2; T2=R3+R4; RD=(T1>T2) ? T1:T2
ADD RD, R1, R2	Addition, RD=R1+R2
SUB RD, R1, R2	Subtraction, RD=R1-R2

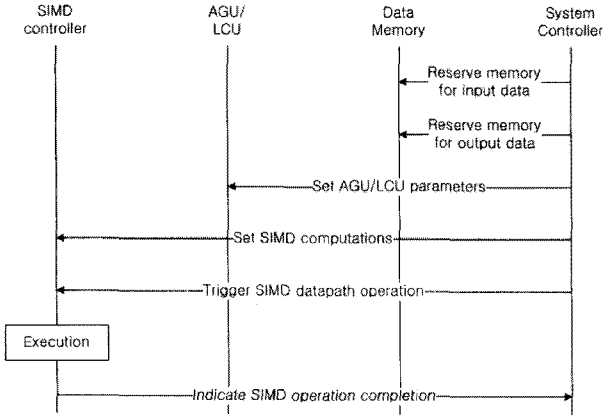


그림 2. SODA-II에서 SIMD 데이터패스를 이용하여 벡터연산을 수행하는 과정

Fig. 2. Vector computation procedure performed in the SIMD datapath of SODA-II.

생성하는데 사용된다. 주소 계산이 자동적으로 이루어짐으로써 벡터 연산 장치들의 사용 효율을 보다 높일 수 있다. LCU는 SIMD 데이터 패스의 동작 회수를 제어하는 역할을 수행하는 일종의 계수기 (Counter)이다. LCU의 값이 사전에 설정해 놓은 값에 도달하면 SIMD 데이터페스는 동작을 멈추고 그 사실을 시스템 제어장치에 인터럽트 (Interrupt) 신호를 발생시켜서 알린다.

이와 같은 SODA-II의 구조는 [2]에서 보인 기저대역 연산이 가지는 특징 가운데 하나인 벡터 파이프라인을 소비전력 최소화와 최대 처리량 증대에 활용한 결과다. SODA-II에 대한 보다 자세한 내용은 [1~2]를 참조하면 된다.

나. SODA-II의 동작과정

SODA-II의 시스템 제어부는 RISC (Reduced instruction set computer) 형태의 명령어를 사용한다. 시스템 제어부가 스칼라 연산을 수행하는 경우에는 직접 스칼라 데이터패스를 제어하며 그 과정은 일반적인 GPP의 경우와 동일하다. 하지만 시스템 제어장치가 벡터 연산을 수행하고자 하는 경우에는 SIMD 제어장치의 도움을 받아 SIMD 데이터패스를 간접적으로 제어하며 그 자세한 절차는 다음과 같다.

시스템 제어장치는 SIMD 데이터패스를 이용해서 연산을 수행하기 위해서는 다음과 같은 절차를 수행하여야 한다.

가) 데이터 메모리에 SIMD 데이터패스가 연산 중에 사용할 입력데이터를 준비한다.

나) 데이터 메모리에 SIMD 데이터패스의 연산결과를 저장할 공간을 확보한다.

다) AGU들에 주소를 자동으로 생성하는데 필요한 정보를 설정한다. 예를 들면 읽기/쓰기 동작 여부, 시작 주소, 주소 증감 방식, 읽고 쓰는 데이터의 폭 등이 그 대상이다.

라) LCU를 설정하여 몇 주기 동안 연산을 수행할 것인지 SIMD 데이터패스에 알린다.

마) SIMD 제어장치에 명령어를 입력한다. 이때 사용되는 명령어들은 표 1에 정리된 것이다.

바) SIMD 데이터패스의 연산을 시작시킨다. 이 과정은 메모리 영역에 할당된 제어 레지스터의 특정 비트를 설정하는 형태로 구현된다.

사) 시스템 제어장치는 SIMD 데이터패스의 연산이 완료될 때까지 대기하거나 다른 스칼라 연산들을 수행한다.

SIMD 제어장치는 시스템 제어장치의 입력을 바탕으로 다음과 같이 동작한다.

가) 시스템 제어장치에서 시작 신호를 보낼 때까지 아무런 동작을 수행하지 않고 대기한다.

나) 시스템 제어장치가 시작 신호를 보내면 설정된 명령어를 분석하여 관련을 가지는 장치들에 동작 인가 신호를 보낸다. 이때 시스템 제어장치에 의해서 사전에 설정된 SIMD 제어 명령어를 분석하여 벡터 연산기 제어에 사용할 명령어들을 생성한다. 생성되는 명령어들은 표 2에 정리된 것들이다.

다) AGU들은 매 주기 마다 주소를 생성하여 데이터들을 자동으로 읽고 쓸 수 있도록 한다.

라) LCU는 매 주기 마다 계수기의 값을 증가하고 그 값이 사전에 설정된 값과 같으면 SIMD 데이터패스의 동작을 중단시킨다.

마) 동작이 완료되면 그 사실을 시스템 제어장치에 인터럽트 신호를 발생시켜 알리고 다시 처음과 같은 대기 상태에 다시 들어간다.

다. SODA-II의 SIMD 데이터패스 제어를 위한 시스템 라이브러리

일반적인 사용자가 앞서 설명한 것과 같은 복잡한 과정을 이해하여 SODA-II의 SIMD 데이터패스를 효율적

으로 사용하는 것은 매우 어렵다. 따라서 많은 경우 고급언어 (High level programming language)를 사용하여 원하는 동작을 기술하고 컴파일러가 자동으로 하드웨어에 적합한 명령어를 만들어내는 방식이 사용된다. 하지만 컴파일러가 자동 생성한 기계어는 SIMD 데이터패스를 가진 하드웨어에서 최적의 결과를 가져다주지 않는다. 소비전력과 최대처리량에 대한 제약이 매우 높은 기저대역 신호처리에서 SIMD 데이터패스를 위한 컴파일러를 사용함으로써 발생하는 비효율성은 간과하기 어려운 수준이다. 이런 측면에서 컴파일러를 사용하지 않으면서도 사용자가 편리하게 응용프로그램을 개발할 수 있는 환경을 구축하는 것이 필요하다.

앞서 서론에서 언급한 것과 같이 기저대역 신호처리의 경우 SIMD 데이터패스에서 처리되는 벡터 연산의 종류가 한정되어 있기 때문에 인간에 의한 최적화가 개발 시간과 유지 보수 측면에서 크게 문제되지 않는다. 반면에 컴파일러를 사용하지 않음으로서 얻는 소비 전력과 최대 처리량 증대 측면에서 얻을 수 있는 이득은 인간에 의한 최적화에 따르는 어려움보다 더 크다고 할 수 있다. 따라서 SODA-II에서는 SIMD 데이터패스에 특화된 컴파일러 대신에 라이브러리 루틴들을 사용한다. SIMD 데이터패스를 제어하기 위해 개발된 라이브러리 루틴에는 다음과 같은 것들이 있다.

- SIMD_fir() : FIR 여과기
- SIMD_cor() : 상관도 계산기
- SIMD_min() : 최소값 찾기
- SIMD_max() : 최대값 찾기
- SIMD_pattern() : 패턴 정합
- SIMD_FFT() : FFT 연산
- SIMD_BMC() : 비터비 BMC 연산
- SIMD_ACS() : 비터비 ACS 연산
- SIMD_VRegLoad() : 벡터 레지스터 값 탑재
- SIMD_status() : SIMD 연산 상태 확인

이들은 이전의 연구를 통해 얻어진 기저대역 신호처리 동작에 대한 분석결과를 바탕으로 선정된 것이다[3]. 개발된 라이브러리 루틴들은 SIMD 데이터패스 제어에 필요한 제어 동작들을 내부적으로 수행하고 사용자에게는 동작에 필요한 기본적인 계수들만을 요구한다. 이렇게 함으로써 사용자가 개발하는 응용 프로그램의 복잡도를 낮출 수 있다.

2. SODA-II에서 라이브러리 루틴을 이용한 핵심 연산 구현의 예

가. 라이브러리 루틴의 구현 예

그림 3은 상관도 계산을 위해 구현된 라이브러리 루틴의 예이다. 설명에 앞서 시스템 제어장치에서 SIMD 데이터패스를 제어하기 위해 사용되는 레지스터들은 시스템 제어장치가 접근할 수 있는 메모리 공간에 할당되어 있다. 이 예에서는 간략화를 위해 제어 레지스터들의 주소 값을 정의한 부분은 생략하였다.

첫 번째 동작은 입력 데이터를 벡터 연산 장치로 읽어오는데 사용되는 AGU를 설정하는 과정으로 줄번호 6~11에 위치한다. AGU의 시작 주소는 입력 변수로 받은 "idata_addr"을 이용하여 설정한다. 이 예에서 AGU 제어 레지스터의 설정되는 값은 2 바이트 (16bit) 데이

```
1  void SIMD_cor(int idata_addr, int odata_addr, int length)
2  {
3
4    volatile int *ptr;
5
6    // SET AGU for input data
7    ptr = (int *)AGU1_ADDR_REG_1;
8    *ptr = idata_addr;
9    ptr = (int *)AGU1_CTRL_REG_1;
10   *ptr = AGU_2BYTE | AGU_READ |
11          AGU_LINEAR_MODE | AGU_INC_4B;
12
13   // SET AGU for output data
14   ptr = (int *)AGU2_ADDR_REG_1;
15   *ptr = odata_addr;
16   ptr = (int *)AGU2_CTRL_REG_1;
17   *ptr = AGU_4BYTE | AGU_WRITE |
18          AGU_LINEAR_MODE | AGU_INC_4B;
19
20   // SET Loop counter
21   ptr = (int *)LCU_COUNTER_REG;
22   *ptr = length;
23
24   // SET SIMD computation
25   ptr = (int *)SIMD_FUNC_REG;
26   *ptr = SIMD_COR;
27
28   // TRIGGER SIMD datapath operation
29   ptr = (int *)SIMD_CONT_REG;
30   *ptr = SIMD_START;
31 }
```

그림 3. 상관도 계산을 위해 SIMD 데이터패스를 제어하는 라이브러리 루틴

Fig. 3. Example of library routine which controls SIMD datapath for correlation computation.

```
1  DCCAD R4, R0, R2, R1, R3
```

(a) 상관도 연산

```
1  BMC.D  R10, R0, R3, R1, R3, 0
2  BMC.D  R11, R0, R3, R1, R3, 4
3  BMC.D  R12, R0, R4, R1, R4, 0
4  BMC.D  R13, R0, R4, R1, R4, 4
5  BMCA   R10, R2, R3. R10, 2
6  BMCA   R11, R2, R3. R11, 6
7  BMCA   R12, R2, R4. R12, 2
8  BMCA   R13, R2, R4. R13, 6
```

(b) 비터비 BMC 연산

그림 4. SIMD 데이터패스 제어를 위해 사용되는 명령어들의 예

Fig. 4. Example of instructions for SIMD datapath control (a) correlation (b) Viterbi BMC.

터 (AGU_2BYTE), 데이터 읽기 (AGU_READ), 선형적인 주소 증가 (AGU_LINEAR_MODE), 4 바이트의 주소 증가 (AGU_INC_4B)를 의미한다. 출력 데이터를 위한 AGU의 설정도 (줄번호 13~18) 유사한 방식으로 이루어진다. 줄 번호 20~22에서는 상관도 연산의 반복 횟수를 설정하며 이 값은 입력데이터의 길이(length)에 의해서 결정된다. SIMD 연산장치에서 이루어지는 연산의 종류는 줄 번호 24~26에서 설정된다. SIMD_COR는 SIMD 연산이 상관도 계산임을 나타낸다. 마지막으로 SIMD 데이터패스가 연산을 시작하도록 제어 레지스터를 설정하는 과정이 필요하며 이것은 줄번호 28~30 사이에 이루어진다.

그림 4(a)는 상관도 계산을 위해 SIMD 데이터패스에서 사용된 명령어들을 보여준다. 그림 3의 줄번호 26에서 시스템 제어장치가 설정한 "SIMD_COR" 명령어는 SIMD 데이터패스의 명령어 가운데 하나인 "DCCAD R4, R0, R2, R1, R3"로 변환되어 실행된다. 이 명령어는 두 개의 조건부 보수 (complement) 연산을 수행하여 결과를 합해서 R4에 저장하는 것이다. 상관도 계산의 경우 계속적으로 동일한 명령이 수행되므로 한 가지 마이크로 코드만이 사용되었다. 다른 알고리즘을 수행하는 예로 비터비-BMC 연산을 처리하는 과정을 보인 것이 그림 4(b) 이다. 이 경우 그림 4(a)와 달리 다수의 SIMD 명령어들이 사용되었다.

나. 사용자 프로그림의 구현 예

그림 5는 사용자 프로그램에서 라이브러리 루틴들을 사용하는 예를 보여준다. 이 프로그램은 상관도 계산을

```
1  int main()
2  {
3    int icoef_addr = 0x0000;
4    int idata_addr = 0x1000;
5    int odata_addr = 0x2000;
6    int length     = 1024;
7   // CONFIGURE SIMD datapath
8    SIMD_VregLoad(VREGA, icoef, 2, 2, 32);
9    SIMD_cor(idata_addr, odata_addr, length);
10   // wait SIMD completion
11   SIMD_status(BLOCKING);
12   // dump output
13   for(i=0; length; i++)
14    printf("y[%d] = %x\n", i, odata_addr + (i*2)));
15 }
```

그림 5. 상관도 계산용 라이브러리 루틴을 이용하는 사용자 프로그램의 예

Fig. 5. Example of application program which uses correlation library routine.

수행하고 그 결과를 출력하는 프로그램이다. 줄 번호 7~9에서 보인 것과 같이 SIMD 데이터패스를 설정하는 과정이 가장 처음에 이루어진다. 줄 번호 8의 SIMD_VRegLoad()는 벡터 레지스터를 초기화하는 라이브러리 루틴으로 상관도 계산에 사용되는 계수들을 레지스터에 탑재하는데 사용되었다. 줄번호 9의 SIMD_cor()는 그림 3에서 보인 라이브러리 루틴을 호출하여 SIMD 데이터패스에서 상관도 연산이 수행되도록 하는 부분이다. SIMD 데이터패스에서 연산이 수행되는 동안 시스템 제어장치는 독립적인 연산이 가능하지만 이 예에서는 SIMD 데이터패스의 연산이 종료될 때까지 기다리도록 하였다. 줄 번호 11의 SIMD_status (BLOCKING)은 시스템 제어장치가 SIMD 데이터 패스의 동작이 완료될 때까지 기다리게 하는데 사용된다. 만일 종료 때까지 기다리지 않고 상태만을 확인하려면 계수를 BLOCKING에서 NON_BLOCKING 으로 바꾸어주면 된다. 간략화를 위해 그림 5의 예에서 보이지는 않았지만 SIMD 데이터패스는 연산이 완료되면 인터럽트 신호를 발생시키고 이 신호는 시스템 제어장치가 ISR (interrupt service routine)에 의해서 처리된다. 이 과정을 통해 시스템 제어장치는 SIMD 데이터패스의 연산이 종료되었음을 알 수 있다.

다. 시스템 에뮬레이션의 지원

SODA-II와 같은 특수한 형태의 시스템에서 응용프로그램을 개발하는 경우 구현 결과를 하드웨어에서 직접 검증하는 것은 매우 어려운 일이다. 이런 문제를 최

소화하기 위해서 시스템 에뮬레이션 (System emulation) 기능을 지원한다. 시스템 에뮬레이션 기능은 타겟 하드웨어인 SODA-II에서 실행되는 프로그램과 기능적인 면에서는 동일한 프로그램을 호스트 시스템에서도 구동할 수 있도록 한다. 이 환경을 통해 사용자는 보다 손쉽게 응용 프로그램을 개발할 수 있다.

시스템 에뮬레이션 환경은 신호처리용 라이브러리를 두 가지 형태로 구현함으로써 구축할 수 있다. 라이브러리의 첫 번째 구현 형태는 그림 3에 보인 것과 같이 작업이 SIMD 데이터패스 상에서 수행될 수 있도록 제어 레지스터들에 동작 파라미터들을 설정하도록 하는 것이다. 이 경우 SODA-II를 타겟 하드웨어로 하는 크로스 컴파일러 (Cross compiler)가 사용되며 연산의 결과는 SIMD 데이터패스에 의해서 생성된다. 두 번째 구현 형태는 응용프로그램과 라이브러리 루틴들이 호스트 컴퓨터에서 실행되도록 만드는 것이다. 이 경우 SIMD 데이터패스에 연산을 할당하지 않고 호스트 컴퓨터가 동일한 연산 결과가 얻어지도록 연산을 수행한다. 응용 프로그램과 라이브러리 루틴들은 호스트 컴퓨터의 컴파일러를 사용하여 실행 파일로 만든다.

```
1  #ifndef TARGET
2  void SIMD_cor(int idata_addr, int odata_addr, int
length)
3  {
4
5     int i, j;
6     short *idata;
7     int   *odata;
8     int   sum;
9
10    idata = (short *)idata_addr;
11    odata = (int *)odata_addr;
12
13    for(i=0; i<length; i++) {
14      // load new input data
15      for(j=VECTOR_WIDTH-1;j>0;j--) {
16        vregB[j]=VregB[j-1];
17      }
18      vregB[0] = idata[i*step];
19      // compute correlation
20      sum = 0;
21      for(j=0; j<VECTOR_WIDTH; j++) {
22        if(vregA[j] == 1) sum -= vregB[j];
23        else sum += vregB[j];
24      }
25      // store result
26      odata[i] = sum;
27    }
28 }
29 #endif
```

그림 6. 시스템 에뮬레이션을 위한 라이브러리의 예 (상관도 계산)

Fig. 6. Structure of library routine additionally concerning system level emulation.

이를 통해서 SODA-II에서 수행되던 프로그램을 호스트 컴퓨터에서 수행시킬 수 있게 되어 기능 측면에서 오류가 없는지 미리 검증을 할 수 있다. 호스트 컴퓨터에서 제공하는 다양한 형태의 디버깅 환경을 활용 할 수 있어 응용 프로그램의 검증이 용이해진다. 시스템 에뮬레이션 기능의 제한은 그 동작에 걸리는 시간이 타겟 시스템과 다르다는 것이다.

라이브러리를 컴파일하여 생성되는 실행파일이 SODA-II에서 실행될 프로그램인지 호스트 컴퓨터에서 실행될 프로그램인지에 대한 선택은 C 언어의 #ifdef 와 같은 프리프로세싱 (Preprocessing) 기능을 활용하여 구현하였다. 그림 6은 그림 3에서 보인 상관도 계산용 라이브러리를 호스트 시스템에서 구동시킬 수 있도록 추가적으로 만든 시스템 에뮬레이션 라이브러리의 예를 보여준다. 이 예에서는 TARGET이라는 MACRO 변수를 선언하면 라이브러리 루틴이 SODA-II에서 실행 가능한 형태로 만들어지고 이 변수를 선언하지 않으면 호스트 시스템에서 실행 가능한 형태로 컴파일된다.

III. 실 험

제안된 프로그래밍 모델의 효율성을 입증하기 위해서 W-CDMA 시스템의 다중경로 탐색기와 OFDM 시스템의 복조기를 SODA-II 상에서 구현하였다[7]. 이들은 기저대역 신호처리 과정에서 가장 많은 연산을 필요로 하는 알고리즘 가운데 하나로 기저대역 프로세서에서 효과적인 처리가 이루어져야 한다.

실험을 위해 구현되는 W-CDMA 다중경로 탐색기와 OFDM 복조기의 사양은 표 4에 보인 것과 같다. 이 표

표 4. 구현된 기저대역 신호처리 알고리즘들의 사양

Table 4. Specification of baseband signal processing algorithms used in the experiment.

다중경로 탐색기	
탐색창의 길이	5120 샘플
스크램블링 코드 길이	320
최대 검출 다중 경로의 수	10 개
OFDM 복조기	
iFFT 길이	1024 샘플
입력데이터	복소수

에서 "탐색창의 길이"는 다중경로 탐색기가 경로 탐색을 위해 사용하는 입력 데이터의 크기를 의미한다. "스크램블링 코드 (Scrambling Code)의 길이"는 다중 경로 검출을 위해 입력 데이터와 스크램블링 코드 사이에 상관도를 계산하는데 사용되는 스크램블링 코드의 길이를 의미한다. "최대 검출 다중 경로"의 수는 다중 경로 탐색기에서 검출하여 관리할 수 있는 신호 전송 경로의 최대 수이다. 표 4의 계수들이 커지면 수신기의 성능은 향상되지만 그에 따라 연산량이 급격하게 늘어나게 된다. 실험에서 사용하는 값들은 성능 저하를 최소화하면서도 연산량을 줄일 수 있도록 분석하여 결정한 것이다. OFDM 복조기의 동작은 iFFT (Inverse FFT) 연산이 주를 이룬다. 평가를 위한 구현한 iFFT 연산은 1024개의 복소수 데이터를 입력으로 사용한다.

가장 좋은 성능 비교 방법은 다른 상용 SDR 프로세서들에 동일한 작업을 수행시켜 서로의 수행 시간을 직접 비교하는 것이다. 그렇지만, 대부분의 상용 SDR 프로세서들은 성능 평가용 시스템이 아직 일반에 공개되지 않기 때문에 직접적인 비교가 어렵다. 이런 이유로 SODA-II와 일반적인 스칼라 프로세서를 서로 비교한다. 실험을 위해 구현된 SODA-II는 ARM926EJ-S를 스칼라 제어장치로 사용한다. 성능의 비교는 동일한 기저대역 신호처리 작업을 ARM926EJ-S 만을 사용하여 처리한 경우와 SIMD 데이터패스도 사용하여 처리한 경우의 처리 시간들을 서로 비교하여 이루어 졌으며 표 5는 그 비교 결과를 보여준다.

표 5. ARM926과 SODA-II의 성능 비교
Table 5. Performance comparison between ARM926 and SODA-II.

		Cycles	성능향상
다중경로 탐색기	ARM926	287749106	48.08 배
	SODA-II	5983966	
OFDM 복조기	ARM926	6778489	710.9배
	SODA-II	9535	

표 6. SODA-II 동작 상세 내역
Table 6. Detailed execution results of SODA-II.

다중경로 탐색기		
수행시간	Cycles	비율 (%)
스칼라 데이터패스 수행시간	5793326	96.8%
SIMD 데이터패스 수행시간	183040	3.05%
SIMD 데이터패스 설정시간	7600	0.01%
SODA-II 전체 수행시간	5983966	
OFDM 복조기		
수행시간	Cycles	비율 (%)
스칼라 데이터패스 수행시간	0	0%
SIMD 데이터패스 수행시간	5120	53.7%
SIMD 데이터패스 설정시간	4415	46.3%
SODA-II 전체 수행시간	9535	

또한 이 논문에서 설명된 프로그래밍 모델의 효율성을 보이기 위해서 SIMD 데이터패스의 실제적인 동작에 앞서 이를 설정하는데 소비되는 시간이 전체 알고리즘 수행시간 가운데 얼마 정도의 비중을 차지하는지 분석한다. 이는 프로그래밍 모델이 사용하기 편리하더라도 실제적인 동작 시간이 너무 길어 전체 처리량을 낮춘다면 좋은 프로그래밍 모델이 아니기 때문이다.

표5의 결과를 살펴보면 SODA-II는 ARM926에 비해 CDMA 다중경로 탐색기를 약 48배 그리고 OFDM 복조기를 710배 더 빠르게 처리함을 알 수 있다. 이는 앞서 설명한 것과 같이 SIMD 데이터패스에서 벡터 연산, 벡터 데이터 정렬, 데이터 읽기/쓰기, 주소의 발생 등의 연산을 모두 병렬로 처리하기 때문이다.

표 6의 결과를 살펴보면 SODA-II는 다중 경로 탐색 기능을 5983966 주기 안에 완수하였다. 이 가운데 SIMD 프로세서가 동작하는 시간의 비율은 3.05 %로 대부분의 동작 시간이 스칼라 데이터패스에 의해서 처리된다. 이는 SIMD 데이터패스가 병렬처리를 통해 벡터 연산들의 처리 시간을 낮춘 결과다. 이는 표 5의 ARM926과의 결과를 비교해 보면 당초 전체 연산의 2% (=5793326/287749106) 정도에 해당하던 작업이 SIMD 데이터패스를 사용한 병렬처리의 결과로 전체 작업 시간의 약 97%를 차지하는 작업으로 변한 것이다. 또한 SODA-II에서 SIMD 데이터패스를 제어하는데 사용된 연산 시간은 전체 연산시간의 0.01%로 제안된 프로그래밍 모델을 통해 적은 부하로 SIMD 데이터패스를 효과적으로 제어 할 수 있다. OFDM 복조기의 경우 벡터 연산인 FFT 연산이 주를 이루므로 다른 스칼라 연산들이 포함되지 않았다. 이런 결과로 SIMD 데이터패스의 설정 시간이 4415 cycle로 전체 연산시간의 46.3%에 이른다. 하지만 이는 ARM926에서 처리되는 시간과 비교하여 약 0.06% (=4415/6778489) 에 해당하는 처리시간으로 매우 적은 부하임을 알 수 있다. 이와 같은 결과들을 종합해보면 본 논문에서 제안한 프로그래밍 모델을 이용하여 SODA-II의 SIMD 데이터패스를 효과적으로 제어할 수 있음을 알 수 있다.

Ⅳ. 결 론

본 논문에서는 SDR용 기저대역 프로세서인 SODA-II의 프로그래밍 모델에 대해서 살펴보았다. SODA-II는 SIMD 파이프라인에 기반을 둔 기저대역 프로세서로 기저대역 신호의 특성을 여러 측면에서 활용하여 성능은 우수하지만 그 구조가 복잡하여 효율적인 컴파일러의 구현이 어려운 측면이 있었다. 본 논문에서는 기저대역 신호처리에서 빈번히 나타나는 연산의 종류가 제한적이라는 점에 착안하여 라이브러리 루틴 형태의 프로그래밍 모델을 제안하였다. 이 방법을 적용하여 기저대역 신호처리 알고리즘을 효과적으로 구현할 수 있음을 실험으로 보였다.

참 고 문 헌

[1] H. Lee, et. al., "A Low Power DSP for Wireless Communications," IEEE Trans. Very Large Scale Integrated Systems, Accepted

[2] H. Lee, "A Baseband Processor for Software Defined Radio Terminals," University of Michigan, Ann Arbor Ph.D Dissertation, 2007.

[3] H. Lee et al, "Software Defined Radio: A High Performance Embedded Challenge," HiPEAC 2006.

[4] H. Chang et. al., "Performance evaluation of an SIMD processor with a vector memory unit," SiPS 2006.

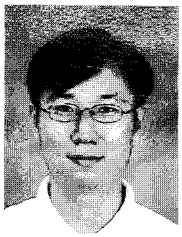
[5] J. Xiong, et. al., "SPL: A Language and Compiler for DSP Algorithms", pp. 298-308, PLDI 2001.

[6] Dorit Naishlos, et. al., "Compiler Vectorization Techniques for a Disjoint SIMD Architecture," IBM Research Report H-0146, 2002.

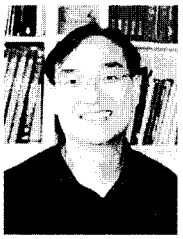
[7] T. Rappaport, "Wireless Communications: Principles and Practice," Prentice Hall, pp. 391, 2002.

저 자 소 개

이 현 석(정회원)
1992년 KAIST 전기및전자공학과 공학사.
1995년 POSTECH 전자전기과 공학석사.
2007년 Univ. of Michigan, Computer Science and Engineering (CSE), Ph.D.
1992년~2008년 삼성전자 통신연구소 수석연구원
2008년~현재 광운대학교 전자통신공학과 조교수
<주관심분야: Software defined radio, 통신용 DSP구조, Embedded system, VLSI>



오 혁 준(정회원)
1999년 한국과학기술원 전기 및 전자공학과 공학박사
1999년~2000년 미국 Stanford 대학교 박사후과정
2000년~2004년 미국 Qualcomm사 3GPP UMTS CSM/MSM 개발
2004년~현재 광운대학교 전자통신공학과 부교수
<주관심분야: 차세대 이동통신, 통신 소프트웨어, 통신 모뎀 SoC 설계, 디지털신호처리, 레이더 신호처리>



이 준 환(정회원)
1991년 연세대학교 전자공학과 학사
1998년 Univ. of Michigan, Electrical Engineering and Computer Science (EECS) 석사
2002년 Univ. of Michigan, EECS, 박사
1991년~1995년 삼성전자 시스템LSI 연구원
2003년~2008년 삼성전자 통신연구소 수석연구원
2008년~현재 광운대학교 컴퓨터공학과 조교수
<주관심분야 : SoC/MPSoC 구조설계, Computer Vision, 반도체설계, 저전력설계>