시스템-온-칩의 하드웨어-소프트웨어 통합 시뮬레이션을 위한 다목적 설계 프레임워크

(A Multipurpose Design Framework for Hardware-Software Cosimulation of System-on-Chip)

주 영 표 † 윤 덕 용 † 김 성 찬 †† 하 순 회 †††
(Young-Pyo Joo) (Dukyoung Yun) (Sungchan Kim) (Soonhoi Ha)

요 약 SoC(System-on-Chip)를 설계함에 있어서 칩의 복잡도 증가로 인하여, RTL(Register Transfer Level)에 기반한 기존의 시스템 성능 분석 및 검증 기법 만으로는 점차 짧아지는 '시장 적기 출하(time-to-market)' 요구에 효율적으로 대응할 수 없게 되었다. 이를 극복하기 위하여 설계 초기 단계부터 지속적으로 시스템을 검증하기 위한 새로운 설계 방법이 요구되었으며, TLM(Transaction Level Modeling) 추상화 수준을 가진 하드웨어-소프트웨어(HW-SW) 통합 시뮬레이션이 이러한 문제를 해결하기 위한 방법으로 널리 연구되고 있다. 그러나 대부분의 HW-SW 통합 시뮬레이터들은 다양한 추상화 수준 중 일부만을 지원하고 있으며, 서로 다른 추상화 수준을 지원하는 툴들 간의 연계도 쉽지 않다. 이를 극복하기 위하여 본 논문에서는 HW-SW 통합 시뮬레이션을 위한 다목적 설계 프레임워크를 제안한다. 제안하는 프레임워크는 소프트웨어 응용의 설계를 포함하는 체계적인 SoC 설계 플로우를 제공하며, 각 설계 단계에서 다양한 기법들을 유연하게 적용할 수 있는 동시에, 다양한 HW-SW 통합 시뮬레이터들을 지원한다. 또한 플랫폼을 추상화 수준과 모델링 언어에 독립적으로 설계할 수 있어, 다양한 수준의 시뮬레이션 모델 생성이 가능하다. 본 논문에서는 실험을 통하여, 제안하는 프레임워크가 ARM9 기반의 상용 SoC 플랫폼을 정확하게 모델링 할 수 있는 동시에, MJPEG 예제의 성능을 44%까지 향상시키는 성능 최적화를 수행할 수 있음을 검증하였다.

키워드 : 시스템-온-칩, 하드웨어-소프트웨어 통합 시뮬레이션, 임베디드 시스템, 설계 플로우

Abstract As the complexity of SoC (System-on-Chip) design increases dramatically, traditional system performance analysis and verification methods based on RTL (Register Transfer Level) are no more valid for increasing time-to-market pressure. Therefore a new design methodology is desperately required for system verification in early design stages, and hardware software (HW SW) cosimulation at TLM (Transaction Level Modeling) level has been researched widely for solving this problem. However, most of HW-SW cosimulators support few restricted abstraction levels only, which makes it difficult to integrate HW-SW cosimulators with different abstraction levels. To overcome this difficulty, this paper proposes a multipurpose framework for HW SW cosimulation to provide systematic SoC design flow starting from software application design. It supports various design techniques flexibly for each design step, and various HW-SW cosimulators. Since a platform design

· 본 연구는 BK21 프로젝트와 교육과학기술부 도약연구 지원사업(R17-2007-086-01001-0)의 지원을 받아 진행 되었다. 또한 서울대학교 컴퓨터신기술연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 한국전자통신연구원의 SoC 핵심설계인력양성사업의 부분적인 지원을 받았다.

† 학생회원 : 서울대학교 전기컴퓨터공학부
youngpyo@iris.snu.ac.kr
zion2k@iris.snu.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학부
sungchan.kim@iris.snu.ac.kr
††† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
sha@iris.snu.ac.kr

논문접수 : 2008년 3월 4일
심사완료 : 2008년 6월 18일

정보과학회논문지: 시스템 및 이론 제35권 제10호(2008.10)

is possible independently of abstraction levels and description languages, it allows us to generate simulation models with various abstraction levels. We verified the proposed framework to model a commercial SoC platform based on an ARM9 processor. It was also proved that this framework could be used for the performance optimization of an MJPEG example up to 44% successfully.

Key words : System-on-Chip, Hardware-software cosimulation, Embedded system, Design flow

1. 서 론

SoC(System-on-Chip) 설계에 있어서 칩의 복잡도 증가와 함께, 날이 갈수록 짧아지는 시장 적기 출하(time-to-market)의 요구는 시스템 설계 주기의 단축의 강한 압력 요인이 되고 있다. 이는, 소프트웨어 응용 및 플랫폼의 설계가 완전히 끝난 이후에 이루어지는 기존의 RTL(Register Transfer Level)에 기반한 시스템 성능 분석 및 검증 기법만으로는 이러한 요구에 효율적으로 대응할 수 없음을 의미한다. 따라서 설계의 초기 단계부터 지속적으로 시스템을 검증하기 위한 새로운 설계 방법이 요구되었으며, 그 해결책으로 다양한 추상화 수준을 가진 하드웨어-소프트웨어(HW-SW) 통합 시뮬레이션이 널리 연구되고 있다[1].

HW-SW 통합 시뮬레이션에 대한 연구들은 시뮬레이션의 가속을 위한 다양한 최적화에 초점을 맞추고 있는데, 시뮬레이션 자체를 가속하는 기법에 대한 연구와, 시뮬레이션의 추상화 수준을 높여서 빠른 시뮬레이션을 수행하는 연구가 주를 이루고 있다. 시뮬레이션의 추상화 수준은 예전에는 주로 RTL에 기반하고 있었지만, 최근에는 TLM(Transaction Level Modeling)에 기반하여 컴포넌트들 간의 통신을 시그널 단위가 아니라 트랜잭션 단위로 모델링하고 있다. 또한 TLM 수준의 추상화도 다양하게 세분화되어, 사이클 수준, 핀 수준의 정확도를 갖는 구현 모델(implementation model)부터, 아무런 세부 명세를 갖지 않는 명세 모델(specification model)까지 다양하게 나뉘어진다[2].

그러나 현재의 HW-SW 통합 시뮬레이터들은 이와 같이 다양한 추상화 수준 중 각기 일부만을 지원하고 있으며, 서로 다른 추상화 수준을 지원하는 툴들 간의 연계도 쉽지 않다. 일부 툴들이 이를 위하여 표준화된 명세 기법을 지원하고 있지만, 아직까지 많은 제약이 있다[3]. 이와 같은 한계를 가진 기존의 HW-SW 통합 시뮬레이터들로는 TLM과 같은 높은 추상화 수준으로 플랫폼을 설계하여, 이를 RTL과 같은 정확한 수준으로 구체화 해 나가는 시스템 설계의 진행이 어렵다. 각 설계 단계에서 새로운 HW-SW 통합 시뮬레이터를 위하여 플랫폼을 매번 새로 모델링하고 이를 검증하는 것은 설계 과정의 세분화로 인한 부가적인 노력을 발생시킬 뿐만 아니라, 새로운 오류의 원인이 되고 있다. 이를 극복하기 위하여 본 논문에서는 HW-SW 통합 시뮬레이션을 위한 다목적 설계 프레임워크를 제안한다.

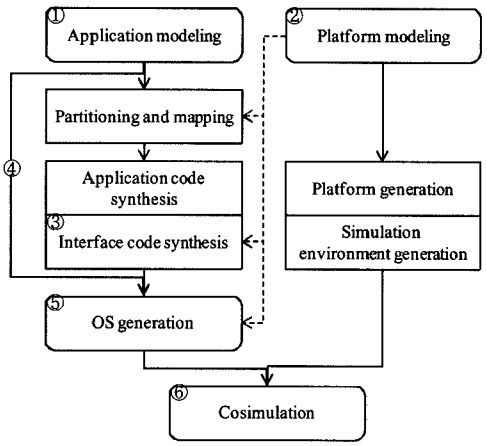

그림 1 제안하는 HW-SW 통합 설계 플로우

제안하는 프레임워크는 그림 1과 같이 소프트웨어 응용의 설계를 포함하는 체계적인 SoC 설계 플로우를 제공한다. 하드웨어-소프트웨어 통합 설계 방법론에 기반한 이 설계 플로우는 소프트웨어를 포함한 응용의 설계와 이를 구동하는 플랫폼의 설계를 하나의 툴에서 진행하여, 최종 시뮬레이션까지 보다 체계적인 SoC 설계 과정을 거쳐 시스템을 설계할 수 있도록 한다. 또한, 그림 2와 같이 각 설계 단계별로 다양한 기법들을 유연하게 적용할 수 있는 동시에, 다양한 추상화 수준을 지원하는 HW-SW 통합 시뮬레이터들을 지원한다. 예를 들어 소프트웨어 응용의 참조 코드가 전혀 없고 외부에서 하드웨어 가속기를 도입할 수 있는 상황이라면, 모델에 기반하여 소프트웨어 응용을 새로 설계하고 이를 외부에서 도입한 하드웨어 가속기와 함께 구동하는 최적화된 설계 과정의 진행이 가능하다. HW-SW 통합 시뮬레이터의 지원은 확장 가능한 구조를 갖고 있기 때문에, 새로운 HW-SW 통합 시뮬레이터의 지원이 가능하다. 즉, 기존에 사용하던 특정 HW-SW 통합 시뮬레이터에서, 새로운 시뮬레이터로의 체계적인 이전이 가능하다. 또한 플랫폼을 추상화 수준과 모델링 언어에 독립적으로 설계할 수 있어, SystemC나 VHDL과 같은 다양한 언어로 설계된 다양한 추상화 수준의 시뮬레이션 모델을 생성할 수 있다.

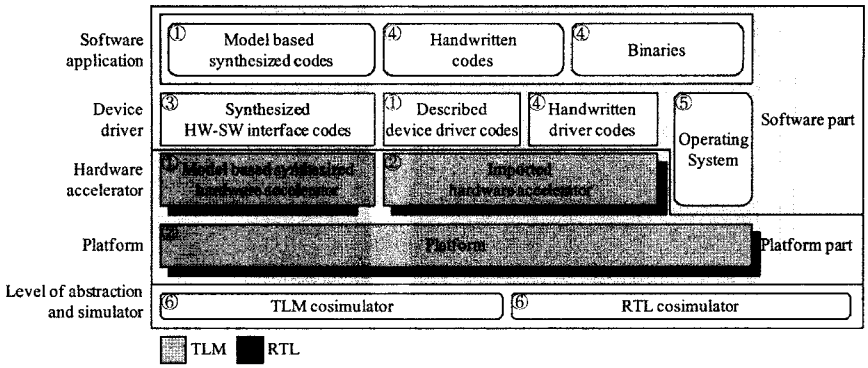

그림 2 본 프레임워크에서 지원하는 설계 단계별 기법들

본 논문의 구성은 다음과 같다. 다음 장에서 관련 연구 및 툴들을 소개하고, 본 연구의 의의를 설명한다. 3장에서는 소프트웨어 응용, 디바이스 드라이버, 그리고 운영 체제를 포함하는 소프트웨어 설계에 대하여 소개하고, 4장에서는 하드웨어 가속기를 포함하는 플랫폼 컴포넌트들의 모델링과 플랫폼의 설계 및 생성 과정에 대하여 설명한다. 제안하는 프레임워크의 검증을 위한 설계 시나리오들에 대한 실험 결과들을 5장에서 소개하며, 6장에서 결론을 맺는다.

2. 관련 연구

다양한 추상화 수준을 지원하는 플랫폼 설계 및 시뮬레이션 툴들은 몇 종류가 존재한다. ARM의 SoC Designer는 사이클 근사 모델(cycle approximate model)부터 사이클 정확 모델(cycle accurate model)까지의 다양한 추상화 수준을 지원한다[4]. 또한, 이 툴은 SPIRIT 컨소시엄의 SPIRIT 1.0을 따르기 때문에, IP-XACT 명세를 지원하는 플랫폼 컴포넌트와 플랫폼의 RTL 구현으로의 이전을 지원한다. SoC Designer의 IP-XACT 지원 기능은, 이 툴로 플랫폼을 설계하고 이를 TLM으로 시뮬레이션 한 후에, 이를 다시 RTL 기반의 HW-SW 통합 시뮬레이터들을 통하여 검증하는 설계 단계의 진행을 통하여 검증된 바 있다[5]. 하지만, SoC Designer는 SystemC만을 지원하기 때문에 다른 언어를 이용하여 모델링 할 수 없으며, 컴포넌트의 추상화 수준이 달라진다는 것은 곧 다른 컴포넌트를 사용하는 것을 의미하므로, 플랫폼의 추상화 수준 변경을 위해서는 플랫폼을 처음부터 재설계하는 노력이 필요하다. 또한 이 툴은 소프트웨어 개발 환경을 제공하지 않고 다른 도구에서 만든 바이너리를 사용하기 때문에, 소프트웨어 개발자는 응용 예제를 툴의 도움을 받지 않고 구동 가능한 형태까지 개발하여야 한다.

CoWare의 Platform Architect는 SystemC를 사용하는 TLM 기반의 플랫폼 설계 및 시뮬레이션 툴이며, RTL과의 혼합-수준(mixed-level) 시뮬레이션을 지원하고 있지만, 완전한 RTL 기반의 시뮬레이션을 수행할 수는 없다[6]. 플랫폼의 모델링을 TLM을 기준으로 하고 있기 때문에 RTL의 지원은 컴포넌트 단위에 머물고 있다. Virtual Platform은 Synopsys의 가상 프로토타이핑 환경이며, SystemC로 작성된 TLM을 가정하고 있다[7]. 이와 비슷한 기능을 가진 Mentor Graphics의 Platform Express는 RTL 수준의 모델링 도구이며[8], 시뮬레이션 환경으로는 같은 회사의 Seamless CVE를 가정하고 있다[9]. 위의 가상 프로토타이핑 환경들은 추상화 수준을 변경할 수 없다는 단점이 있다.

본 연구에서 제안하는 프레임워크는 위의 툴들과 비교하여 표 1과 같은 장점들을 가진다. 우선, 플랫폼을 TLM, RTL과 같은 추상화 수준에 관계없이 설계할 수

표 1 다른 HW-SW 통합 시뮬레이션 툴과의 비교

Tools	Model-based application modeling	Hardware accelerator device driver modeling	OS modeling	Platform modeling	Level of abstraction	Mixed-level simulation
SoC Designer [4]	-	-	-	O	TLM/RTL	-
Platform Architect [6]	-	-	-	O	TLM/RTL	O
Virtual Platform [7]	-	-	-	O	TLM	-
Platform Express [8]	-	-	-	O	RTL	-
Seamless CVE [9]	-	-	-	-	RTL	-
Proposed framework	O	O	O	O	TLM/RTL	-

있으며, SystemC, VHDL, Verilog와 같은 시스템 명세 언어에 의존하지 않은 설계가 가능하다. 이와 같이 설계한 플랫폼은 사용자가 원하는 모델링 언어와 추상화 수준으로 자동 생성된다. 또한 다른 플랫폼 기반의 HW-SW 통합 시뮬레이터들과는 달리, 모델링에 기반하여 응용을 설계하고, 이를 앞서 설계한 플랫폼에서 구동하는 자동화된 과정을 지원한다. 이와 같이 설계된 응용은 플랫폼 의존적인 정보들이 변경되는 경우에도 이를 체계적으로 수용할 수 있기 때문에 효율적인 설계가 가능하다. 본 프레임워크는 이와 같이 플랫폼과 응용의 설계를 구분하여 시스템을 설계하고, 설계 이후에 이를 통합하는 Y-차트 접근법에 기반하고 있으며[10], 해당 접근법에서 요구하는 설계 과정을 지원하고 있다. 마지막으로, 본 프레임워크를 통해서 설계한 플랫폼은 다양한 HW-SW 통합 시뮬레이터들에서 구동 가능한 형태로 자동 생성된다.

3. 소프트웨어 설계

3.1 소프트웨어 응용의 설계

본 프레임워크에서는 소프트웨어 응용의 설계 기법을 그림 2와 같이 모델에 기반한 설계, 코드의 직접 작성, 그리고 바이너리 도입의 세 가지로 가정하고 있다. 이 중에서 모델에 기반한 기법은 그림 3(a)와 같이 기능 블록 단위의 그래프로 응용을 모델링 하는 것을 의미한다. 이렇게 모델링 된 그래프의 각 기능 블록들(그림 3(a)의 A~D)은 그림 3(b)와 같이 이를 구동할 PE (Processing Element) 상에 분할 및 할당되어야 하는데, 여기서 PE는 응용을 직접 수행하는 CPU, DSP, 그리고 하드웨어 가속기와 같은 컴포넌트들을 의미한다.

어떤 기능 블록이 CPU나 DSP와 같이 소프트웨어를 구동하는 PE 상에 할당되었다는 것은 이 기능 블록이 소프트웨어로 자동 합성된다는 것을 의미하고, 그 외의 하드웨어 가속기로 할당되었다는 것은 이 기능 블록이 별도의 하드웨어 가속기로 자동 합성되거나 플랫폼 상에 존재하는 하드웨어 가속기 중 하나를 사용한다는 것

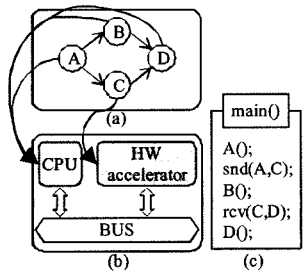

그림 3 (a) 소프트웨어 응용의 모델링, (b) PE 분할 및 할당, (c) 합성된 소프트웨어 코드

을 의미한다. 소프트웨어로 할당된 기능 블록들은 이들이 할당된 PE 단위로 묶인 뒤, 스케줄 분석 기법[11]에 의하여 그들 간의 구동 순서가 정해져 그림 3(c)와 같은 형태의 코드로 최종적으로 합성된다.

합성된 코드에는 다른 PE에 할당된 기능 블록과의 통신을 위하여 snd/rcv와 같은 통신 코드가 포함되는데, 이때 통신하는 PE가 하드웨어 가속기인 경우에는 이들 통신 코드가 디바이스 드라이버가 된다. 이와 같은 소프트웨어 응용의 모델링 및 합성은 직접 개발한 하드웨어-소프트웨어 통합 설계 툴[12]을 통하여 이루어진다. 이와 같은 응용의 모델링 과정을 통하여 사용자는 체계적으로 응용을 플랫폼에 이식할 수 있게 된다.

모델에 기반한 설계 기법 외에도 코드를 직접 작성하는 방법이 있는데, 이는 소프트웨어 응용 코드를 직접 작성하고, 프레임워크가 제공하는 makefile 템플릿에 소스 코드의 정보와 추가적인 빌드 정보를 추가하는 것을 의미한다. 이러한 과정이 기존의 바이너리만 도입하는 방식과 차별화되는 점은, 제공되는 makefile 템플릿을 사용하기 때문에 추가적인 노력 없이 본 프레임워크가 플랫폼에 맞게 자동으로 빌드 한 운영 체제와 그에 따르는 빌드 환경을 그대로 사용할 수 있다는 것이다.

3.2 디바이스 드라이버의 설계

디바이스 드라이버는 하드웨어로 할당된 기능 블록들을 대상으로 하는데, 해당 블록이 별도의 하드웨어 가속기로 자동 합성되는 경우와, 플랫폼 상에 존재하는 하드웨어 가속기 중 하나를 사용하는 경우에 대하여 각기 다른 기법이 이용된다. 전자의 경우에는 통합 설계 방법론의 하드웨어-소프트웨어 인터페이스 코드 자동 합성 기법을 이용한다[13]. 자동 합성된 인터페이스는 데이타 송수신과 동기화를 위한 하드웨어 로직, 그리고 소프트웨어 드라이버로 구성된다.

자동 합성된 인터페이스 하드웨어 로직은 그림 4의 오른쪽에 표현되어 있는데, 이 중 하드웨어 블록 B에 인접한 snd/rcv가 이 블록의 데이타 송수신을 중개한다. 이들 snd/rcv는 Synchronization Controller를 통하여 CPU에 할당된 블록들의 snd/rcv와 동기화를 수행하며, Bus Wrapper를 통하여 버퍼에 접근하여 블록 B의 데이타 송수신을 대행한다. 이때 자동화된 동기화가 가능한 이유는, 앞서 응용을 그래프로 모델링 하고 이를 합성하는 과정에서 각 기능 블록의 수행 순서가 결정되고 이들간의 공유 데이타의 크기 및 위치 등에 대하여 그림 5(a)와 같은 정보가 자동 생성되기 때문이다. 소프트웨어 디바이스 드라이버는 그림 4의 오른쪽에서 CPU에 배치된 snd/rcv 블록들에 해당하며, 이들의 pseudo 코드는 그림 5(b)와 같은 형식을 갖는다. 이 드라이버 코드에서 4, 7행에 해당되는 인터럽트 관련 코드는 그림 4

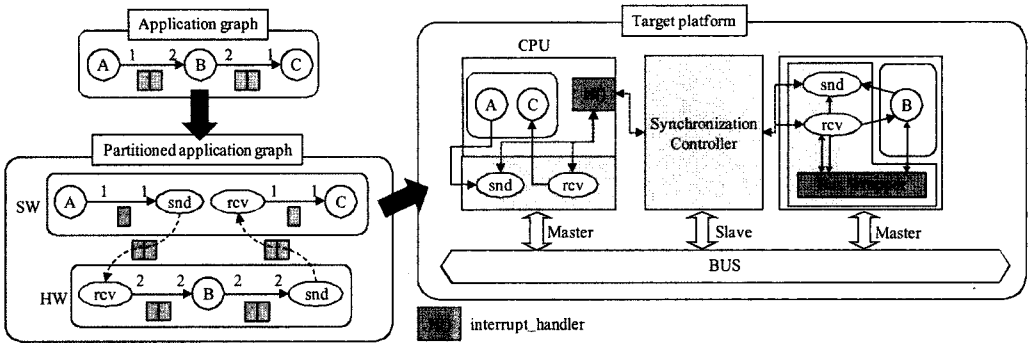

그림 4 자동 합성된 하드웨어 가속기를 위한 하드웨어-소프트웨어 인터페이스의 구조

Write (Send)	SndPtr	Pointer for data sending
	SndSize	Data size of one write operation
	SndIntrPtr	Interrupt register address (for software block)
Read (Receive)	RcvPtr	Pointer for data receiving
	RcvSize	Data size of one read operation
	RcvIntrPtr	Interrupt register address (for software block)
Buffer	ID	Channel ID
	BufferStart	Start address of the buffer on the channel
	BufferSize	Whole size of the buffer on the channel
	InitDelay	Initial data size on the buffer

(a)

```
Device driver for data sending:
1)  if this is a first execution
2)    SndPtr = BufferStart + InitDelay
3)  end if
4)  wait until SndIntrPtr is raised
5)  write data to SndPtr (size: SndSize)
6)  SndPtr += SndSize
7)  clear SndIntrPtr signal
```

```
Device driver for data receiving:
1)  if this is a first execution
2)    RcvPtr = BufferStart
3)  end if
4)  wait until RcvIntrPtr is raised
5)  read data from RcvPtr (size: RcvSize)
6)  RcvPtr += RcvSize
7)  clear RcvIntrPtr signal
```

(b)

그림 5 (a) 하드웨어-소프트웨어 인터페이스의 자동 합성을 위한 정보, (b) 자동 합성된 드라이버의 pseudo 코드

의 H()에 해당하는 인터럽트 핸들러에 의해 수행된다.

플랫폼 상에 존재하는 하드웨어 가속기를 위한 디바이스 드라이버의 모델링은 이와는 다른 과정을 통하여 이루어진다. 이 하드웨어 가속기는 그 내부 동작을 알 수 없기 때문에, 자동 합성된 경우와 같이 정해진 크기의 데이타를 주고 받는 자동화된 동기화 메커니즘을 통하여 제어할 수 없다. 그래서 이를 해결하기 위하여 쓰레드 기반의 디바이스 드라이버 핸들러를 만들고, 여기에 사용자가 모델링 한 디바이스 드라이버를 등록하는 방식을 사용한다[14]. 각각의 디바이스 드라이버는 1) 하드웨어 가속기의 구동 조건 확인, 2) 하드웨어 가속기의 구동(데이타 송신), 3) 하드웨어 가속기의 종료 조건 확인, 4) 하드웨어 가속기의 종료(데이타 수신)에 해당하는 네 개의 함수로 만들어지며, 각 함수는 그림 6의 오른쪽 아래에 있는 check_firing_condition(), fire(), check_is_end(), wrap_up()에 해당한다. 이와 같은 구조로 작성된 드라이버들은 드라이버 핸들러 쓰레드에 등록되어 하드웨어 가속기들을 제어하게 된다. 이 메커니즘은 쓰레드를 지원하는 운영체제가 필요하다는 단점이 있지만, 소프트웨어가 하드웨어 가속기로 인하여 데드락에 빠지는 현상을 막아준다.

이와 같은 디바이스 드라이버 설계 기법들의 사용은 다음과 같은 장점이 있다. 우선, 드라이버 설계에 플랫폼 의존적인 정보들을 사용하지 않으므로 하드웨어 가속기가 플랫폼 상에서 메모리 맵이 바뀌거나 버스 상의 위치가 바뀌는 경우에도 그대로 사용이 가능하며, 새로운 플랫폼에 적용될 때에도 별도의 수정 과정이 필요치 않다. 만약 플랫폼 의존적인 정보가 필요한 경우에는 본 프레임워크에서 제공하는 플랫폼의 파라미터들을 사용할 수 있기 때문에, 결과적으로 디바이스 드라이버의 재사용성을 높일 수 있다.

3.3 운영 체제의 설정 및 빌드

운영 체제의 자동화된 빌드를 위하여 본 프레임워크에서는 설정 가능한 운영 체제인 VPOS를 가정하였다. VPOS는 SoC 플랫폼인 SoCBase[15]와 함께 개발되었는데, POSIX 쓰레드(Pthread)의 부분 집합이 되는 쓰레드 통신 API를 갖고 있다. VPOS는 사용자가 작성한 설정 파일의 정보를 기반으로 하여 빌드 되는데, 본 프레임워크에서는 모델링된 플랫폼의 정보를 분석하여 여기에 필요한 설정 파일을 자동으로 생성한다. 설정 파일

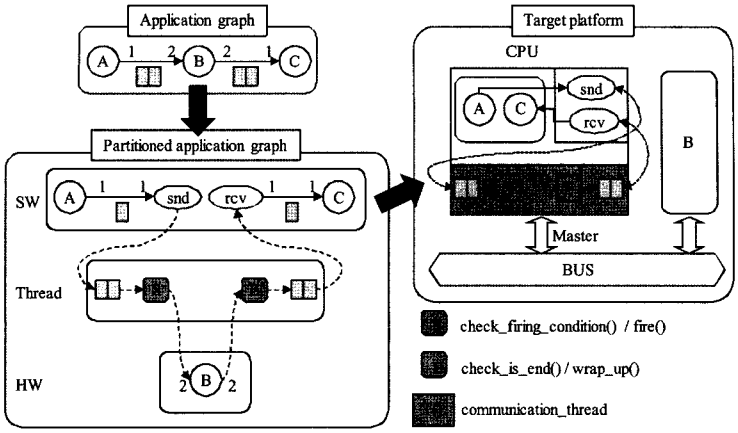

그림 6 플랫폼 상에 있는 하드웨어 가속기를 위한 디바이스 드라이버의 구조

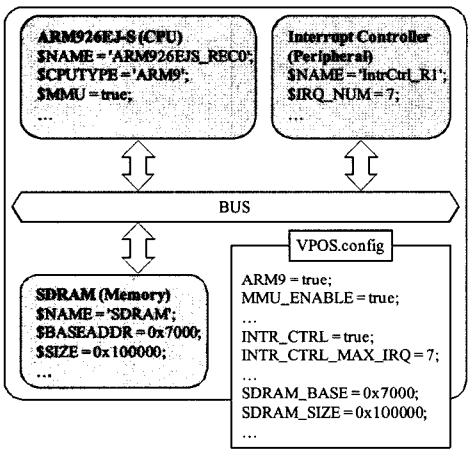

그림 7 운영 체제의 자동 설정

의 생성은 그림 7에서와 같이 플랫폼 상에 존재하는 컴포넌트 모델들의 타입과 그들의 실제 컴포넌트의 이름을 분석하여 운영 체제가 요구하는 컴포넌트의 존재 여부를 확인하고, 존재하는 컴포넌트로부터 원하는 파라미터를 추출하는 과정으로 이루어진다.

이와 같이 플랫폼에 맞게 운영 체제의 빌드를 자동으로 수행하는 것은 사용자가 플랫폼의 정보들을 직접 분석하는 노력과, 그 과정에서 발생할 수 있는 오류의 가능성을 줄여준다. 자동으로 빌드 된 운영 체제는 어플리케이션과 함께 링크되어 하나의 바이너리로 생성되므로, 응용 프로그램의 컴파일 및 링크를 위한 makefile을 포함하고 있다. 이 makefile은 템플릿의 형식을 갖고 있기 때문에, 본 프레임워크에서는 이를 활용하여 모델에 기반한 설계나 코드를 직접 작성하는 과정을 통해 만들어진 소프트웨어 응용을 빌드 한다. 따라서 소프트웨어의 빌드는 플랫폼의 명세가 끝난 이후에 이루어지며, 운영 체제의 빌드를 먼저 수행하고 그 다음에 디바이스 드라이버 및 응용 코드를 빌드를 하는 순서로 진행된다.

4. 플랫폼 설계

4.1 모델링된 하드웨어 가속기를 위한 통신 매커니즘

하드웨어로 매핑된 알고리즘 블록은 그림 4의 오른쪽과 같이 메모리 및 버스 접근 인터페이스가 없기 때문에 자동 합성되는 send/receive 블록과 Bus wrapper를 통하여 데이타를 주고 받는다. Send/receive 블록은 소프트웨어로 매핑된 블록과의 동기화된 메모리 통신 프로토콜을 갖고 있다. Bus wrapper는 플랫폼 독립적인 send/receive의 메모리 접근 프로토콜을 타깃 버스 구조에 맞게 변환하는 역할을 맡고 있으며, 라이브러리 형태를 갖는다. 이 경우 Bus wrapper가 파라미터로 취하게 되는 값은 send/receive의 개수와 공유 메모리 주소의 오프셋이다.

4.2 컴포넌트 모델링

모델에 기반한 설계에 의하여 별도의 하드웨어 가속기로 자동 합성된 기능 블록들은 본 프레임워크를 통하여 만들어진 것이므로 모델링이 필요치 않다. 하지만, 이를 제외한 하드웨어 가속기들과 플랫폼 컴포넌트들은 모델링이 필요하다. 모델링은 크게 두 가지 목표를 갖고 이루어지는데, 첫째는 언어와 추상화 수준에 의존적이지 않아야 한다는 것이다. 둘째는, 그림 8(a)에서 TLM 컴포넌트와 RTL 컴포넌트가 동일한 하나의 컴포넌트를 의미하듯이, 여러 컴포넌트들이 동일한 것임을 표현할 수 있어야 한다는 것이다. 이는 점차 다양한 수준으로 세분화되고 있는 설계 단계들을 반영하기 위해서 매우 중요하다. 본 프레임워크는 이를 위하여 다음과 같은 모델링을 위한 명세 구조를 정의하였다.

4.2.1 컴포넌트 정보 및 파라미터 설정

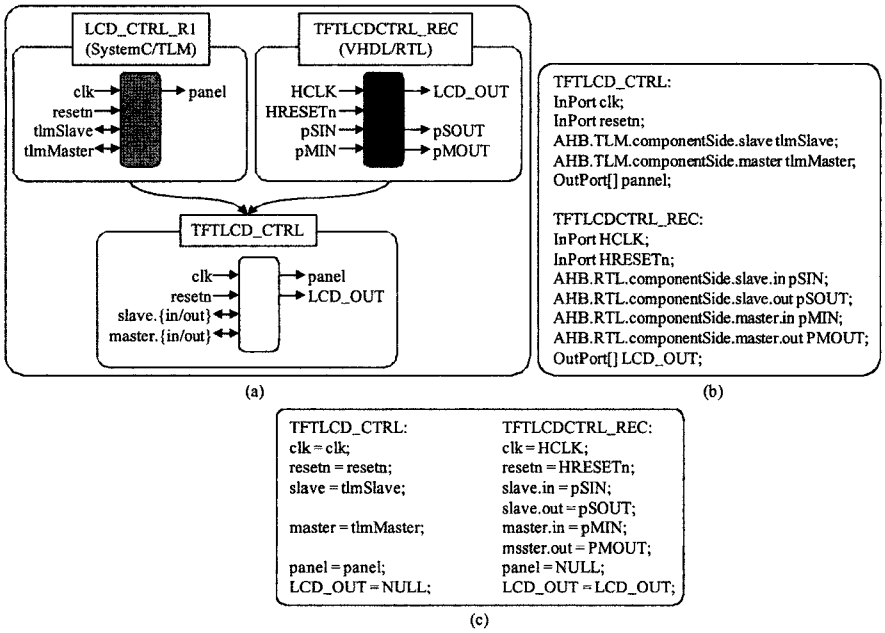

그림 8 (a) 실제 컴포넌트와 컴포넌트 모델의 관계, (b) 컴포넌트의 포트의 입출력과 타입 명세, (c) 컴포넌트 모델과 실제 컴포넌트 사이의 매핑

컴포넌트 정보에는 다음과 같은 내용들이 포함된다. 1) 컴포넌트 모델이 프레임워크 상에서 가지게 될 이름(그림 8(a)에서 TFTLCD_CTRL), 2)이 모델이 표현하는 실제 컴포넌트들의 이름(LCD_CTRL_R1, TFTLCDCTRL_REC) 3), 실제 컴포넌트들의 명세 언어 및 추상화 수준(SystemC/TLM, VHDL/RTL).

한편, 파라미터는 컴포넌트가 가지고 있는 변수들을 나타내고 설정하기 위한 것이다. 예를 들어, 시뮬레이션 과정에서 사용하는 대부분의 SRAM 메모리 컴포넌트들은 그 크기를 가변적으로 설정하여 사용할 수 있다. 이와 같은 변수들은 크게 컴포넌트 마다 가지는 고유한 변수와 컴포넌트 타입 단위로 기본적으로 가지는 변수들로 나뉜다. 컴포넌트 고유한 변수에는, 특정 컴포넌트만의 설정, 예를 들어 클럭 생성기의 클럭 주기와 같은 변수들이 포함되며, 컴포넌트 타입 단위로 가지게 되는 변수들은 CPU의 컴파일 옵션과 같이 해당 타입의 컴포넌트라면 모두 갖게 되는 변수들을 의미한다.

4.2.2 컴포넌트 포트 명세 및 매핑

현실적인 모델링을 위하여 그림 8(a)와 같이 실제 컴포넌트들이 서로 다른 개수와 타입의 포트들을 가질 수 있다고 가정한다. 컴포넌트들의 포트 명세는 그림 8(b)와 같이 이루어지는데, 여기에서 AHB 개체로 명세 된 포트들은 AHB 버스 포트를 의미한다. 버스 포트의 경우에는 포트가 컴포넌트 상에 위치할 때와 버스 상에 위치할 때 그 입출력 타입이 달라지며, 그림 8 (a)에서 pSIN과 pSOUT이 하나의 slave 포트를 구성하듯이, 한 개 이상의 포트들의 조합으로 버스 포트가 구성되는 경우도 있기 때문에 이와 같이 명세하게 된다. 이러한 명세의 장점은, 다양한 언어와 추상화 수준으로 구현된 실제 컴포넌트들의 버스 포트들을 하나로 통일하여 모델링할 수 있게 한다는 점이다. 이와 함께 모델링 된 컴포넌트 상의 버스 포트 표현이 단순화되어 가독성이 높다는 장점도 있다. 명세가 끝난 포트들은 그림 8(c)와 같이 컴포넌트 모델 상의 가상의 포트와 연결하여야 한다.

4.2.3 시뮬레이션을 수행하기 위한 스크립트 파일 생성

HW-SW 통합 시뮬레이션을 수행하기 위해서는 메인 시뮬레이션 엔진을 구동하는 것과는 별개로 1) 별도의 프로세스로 이루어진 시뮬레이터를 실행하거나, 2) 필요한 파일 및 디렉터리들을 생성하거나, 3) 환경 변수를 설정하는 것과 같은 추가적인 작업들이 동반되어야 하는 경우가 많다. 이와 같은 작업들은 기본적으로 쉘 명령을 수행하는 것이기 때문에, 필요한 경우 각 컴포넌트는 순서 있는 리스트 형태의 쉘 명령 집합을 가지게 된다.

4.3 버스 인터페이스와 버스 컴포넌트의 모델링

다양한 추상화 수준들을 구분할 때 하나의 기준이 되는 것이 버스의 모델링 차이가 될 정도로, 각 추상화 수준마다 버스와 버스 포트의 구현은 매우 다르다. 그림 9는 그림 8에서 버스 포트를 명세할 때 사용되었던 자료

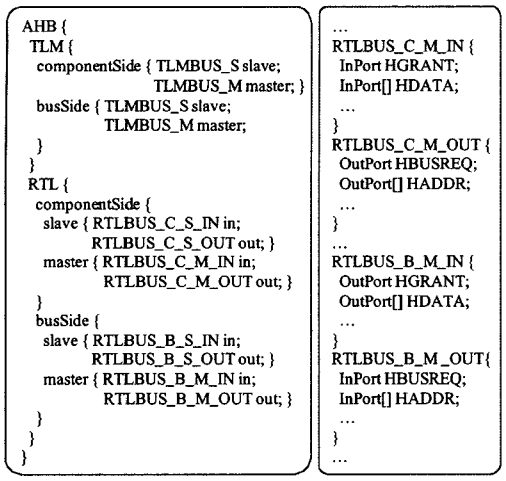

그림 9 버스 인터페이스의 정의

구조로, 다양한 추상화 수준 및 구현들을 수용하기 위하여 설계 되었다. RTL과 TLM은 버스의 구성이 다르며, RTL의 경우는 포트가 버스 쪽에 위치하는 경우와 컴포넌트 쪽에 위치하는 경우가 포트 입출력이 반대로 된다. 이와 같은 복잡한 구현들을 모두 표현할 수 있고, 새로운 추상화 수준을 추가할 수도 있도록 그림과 같이 버스 인터페이스를 정의하는 자료 구조를 정의하였다. 예를 들어, 그림 9에서 AHB 버스의 컴포넌트 쪽 slave 포트 명세는 TLM의 경우 하나만 존재하지만, RTL의 경우에는 두 개가 존재한다. 이는 RTL의 버스 포트 명세가 입력 레코드 타입과 출력 레코드 타입으로 나뉘어 명세 되었기 때문인데, 해당 레코드 타입의 내부 구조는 그림 9의 오른쪽과 같다.

버스 컴포넌트의 버스 포트 모델링은 일반 컴포넌트의 경우와 크게 다르다. 실제 버스 컴포넌트들은 버스의 master/slave 포트 개수가 정해져 있는 경우가 대부분이지만, 본 프레임워크에서는 그 개수에 제한을 두지 않는다. 따라서 플랫폼을 명세 하는 과정에서 버스 컴포넌트를 버스 포트 개수에 따라 계속 교체하거나, 불필요하게 많은 버스 포트가 있는 컴포넌트를 사용하는 문제를 발생시키지 않는다. 또한 모델링 된 버스 컴포넌트의 형태가 매우 간결해져 명세한 플랫폼의 가독성이 높다는 장점도 있다.

버스 포트에 대한 모델의 가정과 실제 컴포넌트 간의 불일치는 플랫폼의 생성 과정에서 해결한다. 플랫폼이 생성될 때, 본 프레임워크는 플랫폼 상에 존재하는 컴포넌트들의 라이브러리를 확인하는데, 이 때 다른 컴포넌트들은 사용자가 설정한 이름을 그대로 검색하지만, 버스 컴포넌트에 대해서는 사용자가 이름에 AHB_M{M}_S{S}와 같이 master/slave 포트 개수 정보를 설정할 수 있도록 하고 있다. 만약, 위의 버스가 3개의 master 포트와 7개의 slave 포트를 사용하도록 플랫폼 상에 배치했다면, AHB_M3_S7이라는 이름의 컴포넌트가 된다.

4.4 플랫폼의 설계 및 생성

플랫폼의 설계는 그림 10(a)와 같은 GUI 기반의 환경에서 이루어지며, 1) 플랫폼 파라미터의 설정, 2) 컴포넌트의 등록 및 배치, 3) 버스 컴포넌트의 배치 및 설정, 4) 포트 간의 연결, 5) 메모리 맵 설정과 같은 과정들로 이루어진다. 플랫폼 파라미터는 그림 10(b)와 같이 플랫폼의 시뮬레이션을 위한 정보들을 의미한다. 한편, 메모리 맵은 그림 10(c)와 같이 버스 단위로 slave 포트들의 기본 주소(base address)와 크기(size) 파라미터 정보를 사용하여 자동 생성되며, 이는 최종적으로 소프트웨어 컴파일 옵션에 추가된다.

이와 같은 과정을 거쳐 설계된 플랫폼은 그림 11(a)와 같은 XML 파일로 자동 생성된다. 이 XML 파일은 그림 11(b)와 같은 구조를 가졌으며, 이는 IP-XACT나 Platform Express에서 정의하는 구조와 유사하다. 이 구조는 크게 1) 환경 변수, 2) 전역 파라미터, 3) 컴포넌트 리스트, 4) 연결 구조로 이루어져 있으며, 이와 같은 XML 파일을 분석하여 자동 생성한 TLM과 RTL 플랫폼 구조는 1) 컴포넌트 선언, 2) 포트 간 연결을 위한 시그널 선언, 3) 컴포넌트 인스턴스 생성, 파라미터 설정 및 포트 연결의 세 단계로 구성된다.

5. 실 험

제안한 프레임워크의 효용성을 검증하기 위하여 플랫폼과 응용의 개발에 대한 세 가지의 실험 시나리오를 표 2와 같이 설정하였다. 모든 실험은 2.4GHz로 구동되는 단일 인텔 펜티엄 4 프로세서와 1GB의 주 메모리로 구성된 시스템 상에서 이루어졌으며, 사용된 운영 체제는 리눅스 커널 버전 2.6.18의 데비안 배포판 버전 4.0이다.

본 실험에서는 TLM 수준 HW-SW 통합 시뮬레이터로 SoCBaseSC를 사용하였는데, 이는 SoCBase 플랫폼[15]을 SystemC를 사용하여 TLM 수준으로 모델링 하는 시뮬레이터이다. SoCBaseSC는 CPU 시뮬레이터로 GDB를 지원하고 있으며, untimed TLM과 timed TLM의 두 가지 추상화 수준을 지원한다. 이 때, untimed TLM은 컴포넌트 내부의 딜레이 모델링을 하지 않기 때문에 시간 정확도를 보장하지 않지만, 보다 빠른 시뮬레이션을 가능하게 한다.

5.1 플랫폼 개발

본 실험은 프레임워크의 표현 능력을 확인하는 것이 목적이므로 새로운 플랫폼을 설계하지 않고, 기존의 SoCBase 플랫폼[15]을 충실하게 모델링 하는 것을 목

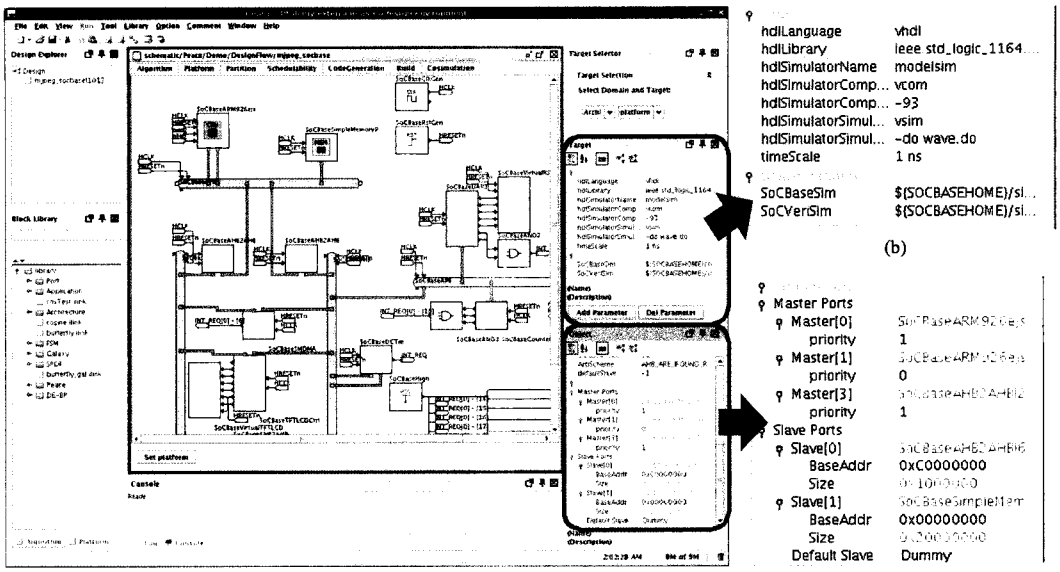

그림 10 (a) 제안하는 프레임워크의 플랫폼 설계 환경, (b) 플랫폼의 RTL 시뮬레이션을 위한 파라미터, (c) AHB 버스의 메모리 맵 설정 및 생성

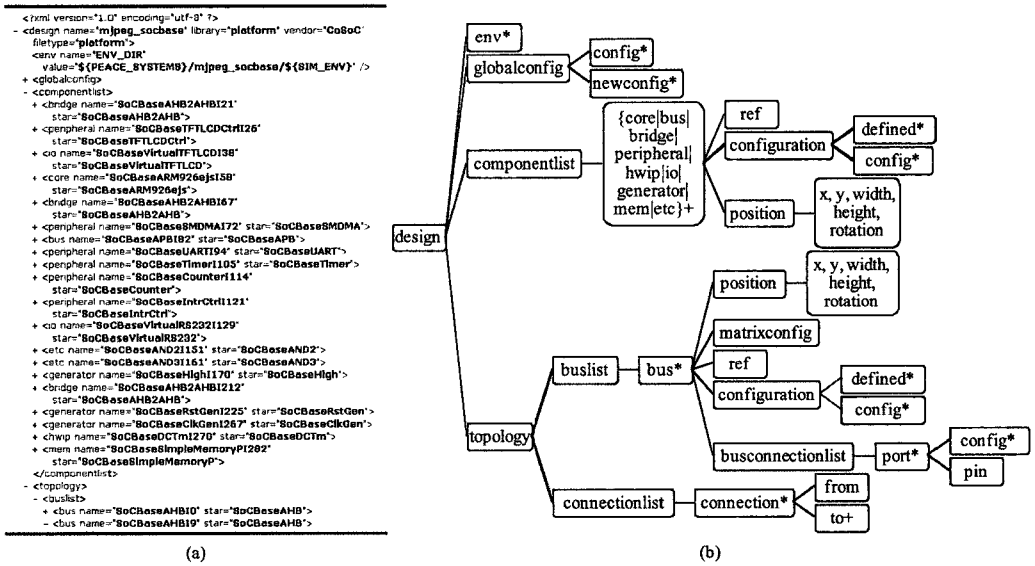

그림 11 (a) 플랫폼 명세로부터 자동 생성된 XML 파일 예제, (b) XML 파일의 구조

표로 하였다. 그에 따라, SoCBase 플랫폼 예제 중 하나인 ARM926EJ-S 프로세서에 기반한 플랫폼을 선정하여 1:1로 모델링 하였다. 이 플랫폼은 그림 12(a)와 같이 세 개의 AHB 버스와 한 개의 APB 버스로 구성되어 있고, LCD, UART와 같은 외부 입출력 인터페이스를 포함하고 있다. 프로세서와 주 메모리 간에는 큰 대역폭이 요구되므로 이들은 하나의 AHB 버스에 배치되었으며, 나머지 컴포넌트들 중에서 상대적으로 대역폭이 큰 DMA와 LCD 컨트롤러 역시 별도의 AHB 버스에 배치되었다. APB 버스에는 상대적으로 큰 대역폭을 차지하지 않는 주변 장치들이 배치되는데, 이러한 배치는 이들 컴포넌트들의 버스 인터페이스에 의해 일차적으로 결정된 것이다.

이와 같이 설계한 플랫폼 위에서 20×20 행렬 곱셈 소

표 2 실험 시나리오 별 HW-SW 통합 시뮬레이션 결과

		Application software model	Hardware IP	Abstraction level	Cosimulator	Simulated time (ns)	Simulation time (sec)
Exp.1	Platform development scenario*	Handwritten code	-	Untimed TLM	SoCBaseSC	-	15
				Timed TLM		5,174,794	31
				RTL	Seamless CVE	5,614,224	75
Exp.2	OS generation scenario**	-	-	Untimed TLM	SoCBaseSC	-	13
				Timed TLM		4,395,148	27
				RTL	Seamless CVE	4,328,320	50
Exp. 3	MJPEG example optimization scenario ***	Model based	-	Untimed TLM	SoCBaseSC	-	418
				Timed TLM		127,940,656	739
				RTL	Seamless CVE	154,771,360	2104
			Imported IP (IDCT)	Untimed TLM	SoCBaseSC	-	326
				Timed TLM		104,735,518	600
				RTL	Seamless CVE	86,554,000	1359

* Application: 20x20 matrix multiplication example
** Simulated time = OS boot-up time
*** Application: 350x240 MJPEG decoding example, Simulated time = 1 frame decoding time

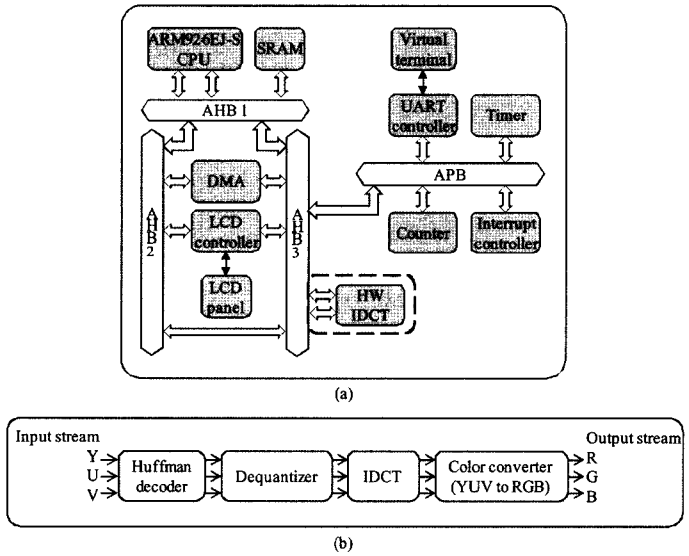

그림 12 (a) ARM926EJ-S에 기반한 SoCBase 플랫폼, (b) MJPEG 예제

프트웨어 예제를 테스트하였다. 표 2의 첫 번째 실험이 이에 해당하는데, 이 과정을 통하여 플랫폼의 기본적인 구동과 프로세서, 메모리, 가상 터미널, 버스와 같은 컴포넌트들의 간략한 검증을 수행할 수 있다.

5.2 운영 체제의 자동 생성

이 실험은 첫 번째 실험을 통하여 검증된 플랫폼에 대하여 최적화된 운영 체제를 자동으로 생성하고, 생성된 운영 체제와 플랫폼에 대한 보다 구체적인 검증을 수행하는 과정이다. 실험의 결과가 되는 시뮬레이션 시간은 운영체제가 초기 부팅 과정을 마칠 때까지 걸리는 시간이며, 이 과정에서 생성된 운영 체제는 물론, 플랫폼의 메모리 접근 및 인터럽트 구동에 대한 다양한 검증도 함께 이루어지게 된다. 이와 같은 시나리오의 진행을 통하여, 플랫폼을 설계하고 이를 운영 체제를 구동할 수 있는 수준까지 검증하는 과정을 본 프레임워크를 통하여 수행할 수 있음을 확인하였다.

5.3 MJPEG 예제 최적화

이 시나리오는 첫 번째 실험에서 개발하고 두 번째 실험을 통해서 운영 체제와 함께 검증된 플랫폼 상에 MJPEG 예제를 이식하고, 그 성능을 최적화하는 개발 과정이다. 이 실험에서 예제를 위한 입력은 350×240 크기로 인코딩된 MJPEG 동영상이며, 표 2에 제시된 시뮬레이션 시간은 한 프레임을 디코딩한 시간이다.

먼저 부가적인 하드웨어 가속기를 사용하지 않는

MJPEG 예제를 그림 12(b)와 같이 3.1장에서 설명한 모델 기반의 설계 과정을 통하여 설계하고, 이를 다양한 수준의 시뮬레이션 모델 상에서 구동하는 과정을 거쳐 검증하였다. 여기에 그림 12(a)의 점선 부분과 같이, 성능 향상을 위해 외부로부터 제공받은 하드웨어 IDCT 가속기를 플랫폼에 추가하고, 이를 구동하는 드라이버를 모델링 하여, 역시 다양한 수준의 시뮬레이션 모델을 통하여 검증하는 과정을 수행하였다. 그 결과, 시뮬레이션 모델마다 그 편차가 있지만, 가장 정확한 RTL 시뮬레이션의 경우에 154.771 ms에서 86.554 ms로 약 44%의 성능 향상을 얻었음을 확인하였다.

이러한 개발 시나리오의 진행을 통하여 본 프레임워크가 응용을 개발하는 과정에서 새로운 하드웨어 가속기를 도입하고, 이를 지원하기 위하여 응용의 모델링을 변경하는 것과 같은 다양한 설계 단계의 진행을 뒷받침할 수 있음을 확인할 수 있었다. 실험 결과가 반영하듯이 각 시뮬레이션 모델 간에는 성능 결과의 오차가 발생하며, 시뮬레이션 모델을 구체화하는 과정에 따른 부가적인 노력 또한 요구된다. 그러나 본 프레임워크가 제공하는 이와 같은 설계 단계를 활용하면, 큰 부가 노력 없이 설계 초기 단계에서 응용과 플랫폼에 대한 검증과 성능 분석을 빠르게 수행할 수 있고, 이를 그대로 RTL 수준으로 이전할 수 있다는 장점이 있다.

5.4 시뮬레이션 결과 분석

Untimed TLM의 경우에는 딜레이 모델링을 하지 않기 때문에 그 시간 정확도를 보장할 수 없지만, timed TLM은 딜레이를 모델링하고 있음에도 불구하고 표 2에서 RTL 대비 최대 20%의 큰 오차를 보이고 있다. 이는 SoCBaseSC HW-SW 통합 시뮬레이터가 CPU 시뮬레이션을 위하여 사용하는 GDB가 부정확하기 때문이다. GDB의 내장 ARM 프로세서 시뮬레이터는 메모리 접근 외의 명령어 수행에 대한 사이클 진행 정보를 얻을 수 없기 때문에, 사이클 단위 정확도를 보장하는 ARMulator에 비하여 예제의 크기가 커질수록 그 오차도 커지게 된다.

한편, 표 2에서 TLM 시뮬레이션은 RTL에 비하여 2배 내외의 속도 개선밖에 보이지 못하고 있는데, 이는 시뮬레이터의 내부 구현 문제와 시뮬레이션 대상이 된 플랫폼의 문제가 복합적으로 작용한 결과이다. SystemC 시뮬레이터는 컴포넌트의 프로세스를 구동하기 위하여 SC_METHOD, SC_THREAD, SC_CTHREAD 중 하나의 구동 방식을 사용하는데, SoCBasSC는 모든 컴포넌트가 쓰레드를 직접 구동하는 SC_THREAD를 사용한다. 이러한 구조는 모든 컴포넌트가 쓰레드 단위로 생성되기 때문에 지나치게 많은 쓰레드가 호출되어 시뮬레이션 시간이 오래 걸리는 문제가 있다. 그 외에도, 플랫폼에서 사용하는 AHB/APB 버스의 컴포넌트 모델은 사이클 수준의 정확도를 보장하는 복잡한 모델이기 때문에, 상대적으로 시뮬레이션 시간이 오래 걸린다. 이러한 문제점들을 확인하기 위하여 동일한 플랫폼을 복잡도가 낮은 다른 버스로 설계한 경우, 하드웨어 가속기를 사용하지 않는 MJPEG 예제의 시뮬레이션에 42초가 걸려, AHB 버스의 untimed TLM의 결과와 비교하여 약 1/10의 시뮬레이션 시간이 걸리는 것을 확인할 수 있었다. 이는 AHB 버스 모델에서 포트 마다 버스 어댑터가 존재하고 시뮬레이션 때 이들이 각각의 쓰레드로 생성되던 것과, AHB 버스 모델의 복잡한 구현이 사라졌기 때문이다.

즉, 시뮬레이션 모델의 정확도나 시뮬레이션에 소요되는 시간이 가지는 문제는 시뮬레이터와 관련된 문제이며, 본 프레임워크와는 관계가 없다.

6. 결 론

본 논문에서는 시스템-온-칩의 HW-SW 통합 시뮬레이션을 위하여 상위 수준의 다양한 설계 단계와 기법들을 통합하여 지원하는 프레임워크를 제안하였다. 이 프레임워크는 Y-차트 접근법에 따라 모델링에 기반하여 소프트웨어 응용을 설계하고, 이를 주어진 플랫폼 상에서 구동하기 위한 자동화된 플로우를 제공한다. 이와 같은 자동화를 위하여 운영 체제의 자동화된 빌드 과정을 지원하는 동시에, 기존의 툴들이 디바이스 드라이버 개발 환경을 제공하지 않았던 것에서 벗어나, 여러 가지의 디바이스 드라이버 개발 기법들을 지원하고 있다. 본 프레임워크는 이와 같은 소프트웨어 설계의 지원 외에도 플랫폼을 추상화 수준과 모델링 언어에 의존적이지 않게 설계할 수 있는 모델링 기법도 제안하였다. 제안하는 프레임워크를 통하여 설계한 플랫폼은 다양한 추상화 수준으로 자동 생성되며, 상용의 SoC 플랫폼의 1:1 모델링과 운영 체제의 자동 생성, 그리고 MJPEG 예제의 최적화 과정에 대한 세가지 실험 시나리오를 통하여 그 효용성을 검증하였다.

이와 같은 결과를 기반으로 하여 추가적인 연구를 통해, 보다 빠르고 정확한 TLM HW-SW 통합 시뮬레이터를 새롭게 지원하는 것을 목표로 하고 있다. 또한, 현재 지원하는 VPOS 외에 멀티 프로세서 환경을 지원하는 알려진 임베디드 운영 체제를 지원하는 것을 향후 목표로 하고 있다.

참 고 문 헌

[1] J. A. Rowson, "Hardware/Software Co-Simulation," in Proc. Design Automation Conference, pp. 439-440, June 1994.

[2] L. Cai and D. Gajski, "Transaction level modeling: an overview," in Proc. International Conference on Hardware/Software Codesign and System Synthesis, pp. 19-24, October 2003.
[3] The SPIRIT Consortium, "References for using IP-XACT," Online document. Available at http://www. spiritconsortium.org/tech/refs.
[4] ARM, "RealView SoC Designer," Online document. Available at http://www.arm.com/products/DevTools/SoCDesigner.html.
[5] C. Shin, P. Grun, N. Romdhane, C. Lennard, G. Madl, S. Pasricha, N. Dutt, and M. Noll, "Enabling heterogeneous cycle-based and event-driven simulation in a design flow integrated using the SPIRIT consortium specifications," in Design Automation for Embedded Systems, Vol.11, No.2-3, pp. 119-140, September 2007.
[6] CoWare, "CoWare Platform Architect," Online document. Available at http://coware.com/products/platformarchitect.php.
[7] Synopsys, "Virtual Platforms," Online document. Available at http://www.synopsys.com/products/designware/virtual_platforms.html.
[8] Mentor Graphics, "Platform Express Professional," Online document. Available at http://www.mentor.com/products/esl/platform_based_design/px_pro/index.cfm.
[9] Mentor Graphics, "Seamless Co-Verification," Online document, Available at http://www.mentor.com/products /fv/hwsw_coverification/seamless/index.cfm.
[10] B. Kienhuis, E. Deprettere, K. Vissers, P. van der Wolf, "An approach for quantitative analysis of application specific dataflow architectures," in Proc. of International Conference Application-specific Systems, Architectures and Processors, pp. 14-16, July 1997.
[11] C. Lee and S. Ha, "Hardware-Software Cosynthesis of Multitask MPSoCs with Real-Time Constraints," in Proc. International Conference on ASIC, Vol.2, pp. 919-924, October 2005.
[12] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. Joo, "PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems," in ACM Transactions on Design Automation of Electronic Systems, Vol.12, No.3, Article 24, August 2007.
[13] 주영표, 양회석, 하순회, "데이타 플로우 모델로부터 합성 가능한 하드웨어-소프트웨어 인터페이스의 자동 생성", 한국정보과학회 추계학술대회, Vol.34, pp. 232-237, October 2007.
[14] 권성남, 주영표, 하순회, "HW/SW간 인터페이싱을 위한 쓰레드 기반 통신 기법", 한국정보과학회 추계 학술대회, Vol.34, pp. 249-250, October 2007.
[15] S. Park, M. Kim, and S. Chae, "SoCBase: An Integrated solution for platform based design", in Proc. of International SoC Design Conference, pp. 329-332, October 2004.

주 영 표
2002년 서울대학교 컴퓨터공학과 학사
2002년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 하드웨어-소프트웨어 통합설계, MPSoC의 성능 분석 및 구조 최적화

윤 덕 용
2004년 서울대학교 생물자원공학부 학사
2004년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 하드웨어-소프트웨어 통합설계, MPSoC 통합 시뮬레이션, MPSoC 내장형 소프트웨어 설계

김 성 찬
1998년 서울대학교 재료공학부 학사. 2000년 서울대학교 컴퓨터공학부 석사. 2005년 서울대학교 전기컴퓨터공학부 박사
2005년~2006년 삼성전자 반도체총괄 SystemLSI 사업부 책임연구원. 2006년~현재 서울대학교 전기컴퓨터공학부 박사 후 과정. 관심분야는 하드웨어-소프트웨어 통합설계, SoC의 성능 분석 및 구조 최적화

하 순 회
1985년 서울대학교 전자공학과 학사. 1987년 서울대학교 전자공학과 석사. 1992년 미국 UCB 전기컴퓨터공학과 박사. 1993년~1994년 현대전자 근무. 1994년~현재 서울대학교 전기컴퓨터공학부 교수
관심분야는 하드웨어-소프트웨어 통합설계, 내장형 시스템을 위한 설계 방법론, MPSoC 내장형 소프트웨어 설계