플랫폼 기반 시스템 설계 방법론 제안 및 구현

윤덕용[O1], 기안도[2], 유우석[3], 하순회[1]
[1]서울대학교 전기컴퓨터공학부 통합설계및병렬처리연구실
{zion2k, sha}@iris.snu.ac.kr
[2]서울대학교 전기컴퓨터공학부 실시간운영체제연구실
wsyoo@filewood.snu.ac.kr
[3](주) 다이나릿
adki@garden.dynalith.com

Platform based System design methodology and Implementation

Dukyoung Yun[O1], Ando Ki[2], Wooseok Yoo[3], Soonhoi Ha[1]
[1]CAP Lab, School of EECS, Seoul National Univ.
[2]RTOS Lab, School of EECS, Seoul National Univ.
[3]Dynalith Inc.

요 약

이 논문에서는 시스템 수준의 가상프로토타이핑 환경을 자동으로 생성하고 이를 위한 소프트웨어 환경을 생성하는 프레임워크를 제안한다. 가상프로토타이핑 자동 생성 프레임워크는 Y-chart 접근법을 기반으로 한다. 알고리즘의 명세를 위해서는 SDF 모델 기반의 방법과 사용자가 직접 Task를 기술할 수 있는 방법을 제공하고 아키텍처 명세를 위해서는 플랫폼 기반의 기술 방법을 제공한다. 플랫폼 명세는 시스템에 사용되는 모듈에 대한 인터페이스 파일을 명세하고 그래픽 기반의 플랫폼 기술을 제공하여 사용자가 쉽게 플랫폼 구성을 변경할 수 있도록 했다. 인터페이스 파일에 모듈을 사용하기 위한 디바이스 드라이버의 정보를 명세하여 소프트웨어 생성 시 모듈을 사용하기 위한 코드가 자동으로 삽입되도록 하였다. 프로세서의 시뮬레이션은 빠른 소프트웨어의 기능 개발과 설계 공간 탐색을 위해 지연시간이 기술된 코드를 호스트에서 직접 수행하는 방법과 검증을 위하여 컴파일 된 이미지를 ISS를 사용하여 시뮬레이션 하는 두 가지 방법을 제공한다. 실험에서는 JPEG decoder를 기술하고 가상프로토타이핑에서 수행해봄으로 해당 프레임워크가 효과적으로 사용될 수 있음을 보였다.

1. 서 론

최근 기술이 발전함에 따라 제품의 주기가 짧아지고 제품의 개발 주기 또한 짧아지게 되었다. 프로토타이핑 시스템을 하드웨어로 만들고 그 위에서 소프트웨어를 개발 및 검증하고 하드웨어를 다시 수정하는 기존의 방법은 개발 주기가 길기 때문에 하드웨어와 소프트웨어를 동시에 개발하는 통합설계방법론이 제안되었다.

통합설계에 대한 많은 연구들은 Y-chart 접근방법[1]을 바탕으로 하고 있다. Y-chart 접근방법은 아키텍처에 독립적인 알고리즘을 명세하고 아키텍처를 명세한 후 알고리즘을 해당 아키텍처에 매핑하고 이를 시뮬레이션해서 성능을 검증하고 제약조건을 만족하지 못하는 경우에는 아키텍처와 매핑을 변경해서 원하는 조건을 만족하게 하는 접근법이다. Y-chart 접근법은 아키텍처에 독립적으로 알고리즘을 기술하기 때문에 다른 아키텍처에 알고리즘을 재사용 할 수 있는 점과 시뮬레이션을 통해 요구조건을 만족하는지를 미리 확인할 수 있기 때문에 다양한 설계 공간 중 주어진 어플리케이션에 적합한 설계 공간을 쉽게 찾을 수 있다는 장점을 가진다.

또한 요구되어지는 많은 제품들이 비슷한 기능을 사용하기 때문에 IP의 재사용 또한 중요해졌다. 그래서 아키텍처를 새로 설계하기보다는 많이 사용되는 모듈을 포함하는 기본적인 아키텍처를 플랫폼으로 지정하고 여기에 애플리케이션에서 요구되는 전용 하드웨어를 추가, 변경하는 방식을 통해 아키텍처를 설계한다. 이러한 접근방법은 쉽고 빠르게 아키텍처를 설계할 수 있으며 검증된 설계를 기반으로 시작하기 때문에 새롭게 아키텍처를 설계하는 방법에 비해 발생할 수 있는 에러가 적다.

시스템 시뮬레이션은 통합설계의 큰 연구 분야 중 하나로 주어진 시스템을 정확하고 빠르게 시뮬레이션하기 위해 많은 방법들이 연구되고 있다. 시스템 시뮬레이션은 다양한 수준에서 수행될 수 있고 일반적으로 정확도와 시뮬레이션 성능은 반비례하는 모습을 보인다. RTL 수준의 시뮬레이션은 signal 단위로 시뮬레이션을 하는 방법으로 정확하지만 느리다는 단점을 가지고 있고 TLM 수준의 시뮬레이션은 RTL 수준의 시그널들을 묶어서 Transaction단위로 시뮬레이션을 하는 방법으로 RTL 수준보다는 정확도가 떨어지나 시뮬레이션 성능이 빠르다는 장점을 가진다.

프로세서의 시뮬레이션은 다양한 방법으로 수행될 수 있다. RTL 수준은 정확한 시뮬레이션이 가능하고 하드웨어를 디버깅할 수 있다는 장점을 가지지만 프로세서가 복잡하기 때문에 시뮬레이션에 많은 시간이 소요된다.

ISS(Instruction set simulator)를 사용한 사이클 수준의 시뮬레이션은 RTL 수준에 비해 빠르고 프로세서의 상태를 확인할 수도 있다. ISS처럼 하나의 명령어를 읽어 들여 해석하고 수행하여 프로세서 내부 상태를 바꾸는 방법은 호스트에서 많은 사이클이 소모되기 때문에 호스트에서 동일한 명령을 수행하고 이로 인한 지연시간을 기술하는 방법이 제안되었다. 이전 방법에 비해 시간 정확도가 떨어지지만 프로세서의 동작을 빠르게 시뮬레이션 할 수 있기 때문에 소프트웨어를 기능을 검증하거나 설계 공간을 탐색할 때 많이 사용된다.

이 논문에서는 Y-Chart 접근방법을 기본으로 하며 Y-chart의 아키텍처 명세를 위해 플랫폼 기반의 아키텍처 명세를 제공하도록 하였다. 시스템 시뮬레이션을 위해 SystemC를 사용하여 TLM 레벨에서 시뮬레이션을 하도록 했으며 프로세서 시뮬레이션을 위해 ISS (Instruction Set Simulator)를 이용한 시뮬레이션과 일반화된 RTOS를 모델링한 GRTOS(Generic Real Time OS)를 사용하여 OS를 시뮬레이션 할 수 있는 환경을 지원한다.

이 논문은 다음처럼 구성된다. 2장에서는 이전 관련 연구들을 간단하게 살펴볼 것이다. 3장에서는 가상 프로토타이핑환경 자동 생성을 위한 전체 프레임워크에 대해 간단히 설명하고 각 세부 사항들에 대해 설명하도록 하겠다. 4장에서는 실험 결과를 보이고 5장에서 결론을 맺도록 하겠다.

2. 관련 연구

2.1 가상 프로토타이핑 환경

시스템 레벨의 가상 프로토타이핑 환경에 관한 연구는 많이 있었고 이에 대해 여러 가지 상용 제품도 존재한다. SoCDesigner[2]는 ARM®의 상용 제품으로 아키텍처를 명세하는 단계와 명세된 아키텍처의 프로세서 위에 컴파일 된 이미지를 올리고 시뮬레이션 하는 단계로 나뉜다. 이 툴은 TLM 레벨에서 하드웨어-소프트웨어 시뮬레이션을 수행한다. Platform Express™[3]은 플랫폼 기반의 설계를 제공하는 Mentor Graphics®의 상용 제품으로 HDL로 명세된 컴포넌트들을 그래픽 환경에서 pin과 버스 인터페이스를 연결하여 ISS와 HDL 시뮬레이터를 사용하여 시뮬레이션할 수 있도록 해준다. CoWare Platform Architect™은 전체 플랫폼을 만들고 시뮬레이션하고 분석 및 디버깅 할 수 있는 SystemC 기반 환경이다. 초기 플랫폼 위에 서드파티에서 제공되는 RTL 구현을 연결하고 검증하는 것도 가능하다. 하지만 대부분의 상용 툴들은 시스템의 시뮬레이션에 중점을 두고 있다. 이 논문에서 제안되는 환경은 가상 프로토타이핑 시스템의 생성 외에 소프트웨어 생성을 지원하기 때문에 기술된 플랫폼의 정보를 고려하여 이에 적합한 소프트웨어를 생성할 수 있다는 장점을 가진다.

2.2 소프트웨어 시뮬레이션

시간을 고려한 소프트웨어 시뮬레이션 방법은 크게 2가지로 구분된다. (1) 기능검증을 위한 코드에 지연시간을 삽입하여 호스트에서 수행하는 방법과 (2) ISS를 사용하여 실제 프로세서에 올라갈 컴파일 된 이미지를 수행하는 방법이다. ISS를 이용한 시뮬레이션은 높은 정확도를 가지며 프로세서의 PC값이나 레지스터를 확인하는 것이 가능하기 때문에 시스템 프로그래밍을 시뮬레이션하고 디버깅하는 것이 가능하지만 느리다는 단점이 있다. 기능 검증을 위한 코드에 지연시간을 삽입하는 방식은 정확도는 떨어지지만 수행속도가 빠르기 때문에 일반적인 소프트웨어를 개발하는데 편리하고 대충의 시간정보도 확인해 볼 수 있다는 장점이 있다.

OS의 시뮬레이션은 (1) 호스트에서 OS를 수행하는 방법과 (2) 가상의 OS를 이용하여 실제 OS의 기능을 시뮬레이션 하는 방법, 그리고 (3) OS의 시간 지연 모델을 세워서 시뮬레이션 하는 방법이 있다.[8] 본 연구에서는 일반화된 가상의 OS를 모델링하여 각 부분에 시간 지연을 넣고 가상의 OS가 제공하는 일반화된 OS API를 사용하여 호스트에서 태스크의 기능과 대략적인 시간을 시뮬레이션 한다. 그리고 이를 자동으로 타겟 OS에 맞게 변환하여 ISS 위에서 정확한 시간 등을 검증하는 환경을 제공한다.

어플리케이션에서 사용되는 알고리즘을 명세하는 방법에 대해서도 많은 연구가 진행되고 있다. 기존의 C와 같은 프로그래밍 언어를 사용한 알고리즘 기술은 사용자에게 높은 자유도를 허락하고 높은 성능을 낼 수 있지만 코드가 프로그램에 특화되는 경우가 많고 이로 인해 재사용이 어려워지고 중간에 오류가 발생하는 경우가 많아진다. 이를 위해 정형화된 모델을 사용하여 프로그램을 기술하려는 연구가 진행되었다. Dataflow의 확장된 형태 중의 하나인 SDF(Synchronous Dataflow)[4]의 경우 각 Block에서 데이터가 일정한 비율로 입출력되기 때문에 정적분석이 가능하다는 장점을 가지고 있다. 외부의 입력이나 이벤트에 의해 상태가 변화되고 이에 따라 동작하는 알고리즘의 경우에는 Finated State Model과 같은 모델을 사용하여 알고리즘을 기술하고 분석한다. 또한 UML[6]과 같은 언어를 이용하여 소프트웨어를 기술하고 이로부터 코드를 생성하는 연구도 많이 진행되고 있다. 본 연구에서는 SDF 모델을 사용한 알고리즘 기술 방법[7]을 지원하고 있으며 모델로 표현되기 어려운 부분이나 기존 코드의 활용을 위해 사용자가 태스크의 코드를 직접 기술할 수 있도록 해준다.

3. 시스템 설계 흐름

시스템 레벨의 가상 프로토타이핑 개발을 위해 본 연구에서는 그림 1과 같이 Y-chart 접근법에 플랫폼 기반 설계를 적용한 설계 플로우를 사용하고 있다. SDF(Synchronous Dataflow)나 Task Model 같이 정형화된 모델을 기반으로 알고리즘을 명세한다. 아키텍처

명세를 위하여 기본 플랫폼을 그리고 추가로 사용할 IP들을 등록하여 플랫폼에 연결한다. 기술된 알고리즘 블록들의 성능 예측 데이터를 이용하여 각 알고리즘 블록을 프로세서, 또는 전용 하드웨어에 매핑한다.

프로세서에 매핑 된 알고리즘 블록은 코드 자동생성과정을 통해서 태스크 형태의 소프트웨어 코드로 생성이 된다. 생성된 태스크는 GRTOS 환경 위에서 수행된다. 이때 태스크의 생성, 스케줄, 태스크 간의 통신, 인터럽트 등은 GRTOS에서 제공되는 Generic API를 이용한다.

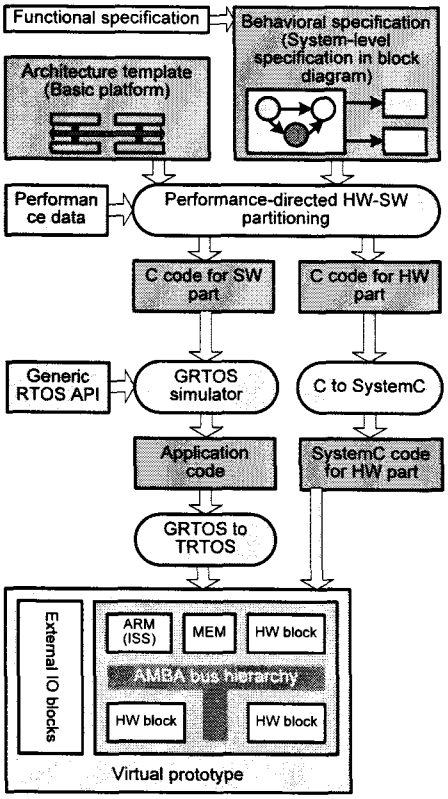

그림 1 Design flow

Generic API를 사용한 코드는 변환과정을 통해 해당 플랫폼에서 사용하고자 하는 Target RTOS 위에서 수행할 수 있다. 가상프로토타이핑 시스템 위에서 프로세서는 ISS를 사용하거나 지연시간을 삽입한 호스트에서의 실행 모델로 시뮬레이션 된다.

하드웨어로 매핑 된 블록들은 자동 인터페이스 생성을 통해 정형화된 프로토콜을 가지는 SystemC 모듈로 생성이 되고 앞에서 기술한 플랫폼에 추가되어 가상 프로토타이핑 시스템을 만들게 된다.

알고리즘 기술과 플랫폼 기술, 소프트웨어 생성 및 시뮬레이션에 대해서 자세하게 설명하도록 하겠다.

3.1 알고리즘 명세

알고리즘의 명세를 위해 본 연구에서는 두 가지 방법을 제안한다.

하나는 SDF 모델 기반의 명세 방법으로 잘 정의된 정적인 비율의 데이터 입출력을 가지는 SDF 블록을 데이터의 흐름에 따라 연결하여 원하는 동작을 할 수 있도록 기술하는 방법이다. 이 경우 하나 이상의 SDF 블록이 모여 하나의 태스크를 만든다.

다른 하나는 본 연구에서 새롭게 추가된 방법으로 태스크 단위로 명세를 하는 방법이다. 사용자가 원하는 코드를 C언어로 기술하여 태스크 단위로 명세한다. 사용자는 태스크 형태로 코드를 기술하기 위해 다음과 같은 정보를 기술해야한다.

- initialize code
- main code
- wrap-up code
- task period
- input, output port

태스크의 초기화를 위한 initialize 코드와 태스크의 동작을 기술한 main 코드, 태스크 종료 시 수행할 wrap-up 코드, 그리고 태스크가 수행되어야 할 주기와 데이터 입출력을 위한 포트들이 정의되어야 한다.

3.2 플랫폼 명세

전체 시스템을 명세하기 위해 시스템에서 사용되는 각 모듈에 대한 인터페이스 파일을 정의하였다. 인터페이스 파일은 해당 모듈이 가지는 입출력(pin, bus 인터페이스)과 모듈의 사용을 위해 설정해야하는 파라미터들과 SystemC 모듈의 이름, 그리고 모듈을 사용하기 위한 디바이스 드라이버 코드의 내용으로 구성된다. 다음은 Timer 모듈에 대한 인터페이스 파일의 예이다.

```
defstar{
    name {DynsocTimers_apb}
    ...
    hinclude { "DynsocBus.h" }
    input { name {PCLK} type {int} }
    input { name {slave_port} type {message} }

    constructor {
        pName.setInitValue("timers_apb");
        slave_port.setBusType(
            new DynsocAPBSlaveIn());

        ...
    }
    defstate {
        name { size }
        type { int }
        default { "0x100" }
        attributes { A_NONSETTABLE }
    }
    setup {
        addInclude("\"timers_apb.h\"");
        addArgElement(name(),"string");

        setDDVar("TIMER0_BASE","#REF_BASE");
        setDDIncludeFile("timer.h");
        setDDInitCode("TIMER_reset();");
    }
}
```

그림 2 Timer 모듈의 인터페이스 예

name에서는 블록의 이름을 설정하고 input, output 등을 통해 포트를 정의한다. constructor에서는 인터페이스 객체가 생성될 때 처리해야 할 작업으로 모듈의 이름, 버스 포트의 설정 등을 처리한다. 만약 모듈 내에서 설정가능한 파라미터가 있다면 defstate를 사용하여 파라미터 값을 입력받을 수 있도록 설정한다. 파라미터를 defstate를 통해 등록하면 GUI에서 해당 값을 입력할 수 있다. setup에서는 모듈이 생성 될 때 필요한 인자들을 설정하고 소프트웨어에서 사용하기 위한 디바이스 드라이버 관련 헤더 파일과 설정 값, 초기화를 위해 실행되어야 하는 코드 등을 설정한다.

모듈의 인터페이스에 대한 명세가 끝나면 GUI에 등록이 가능하다. GUI에 등록이 끝난 후에는 모듈들을 그리고 연결하여 플랫폼을 명세할 수 있다. 여러 신호를 포함하고 있는 버스 연결의 경우는 내부적으로 자료구조를 만들어서 버스 연결이 가지는 세부 연결들에 대한 정보와 연결될 수 있는 버스 연결 타입에 대해 알 수 있도록 하였다. 또한 각 모듈의 메모리 영역을 지정하기 위해 버스의 slave 연결의 경우 base address를 설정할 수 있도록 하였다. 그림3은 등록된 모듈을 사용하여 명세한 플랫폼의 예이다.

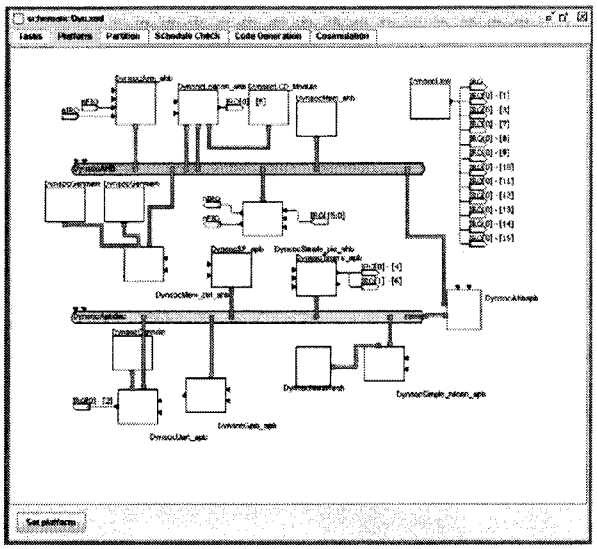

그림 3 DynSoC의 플랫폼 명세

3.3 소프트웨어 코드 생성

프로세서로 매핑 된 알고리즘 블록들은 코드 생성과정을 통하여 GRTOS에서 사용될 수 있는 태스크 형태의 코드를 생성한다. 태스크는 기본적으로 앞에서 언급한 주기, 초기화 코드, 메인 코드, 종료 코드, 통신을 위한 포트의 정보를 가지고 있고 그림4와 같은 형태로 실행된다.

```
task->init();     // initialize code
while(!done) {
    task->go();     // main code
    wait_next_period();
}
task->wrapup();     // finish code
```

그림 4. 태스크 실행 형태

플랫폼의 일부 모듈은 소프트웨어에서 사용하려면 초기화나 관련된 디바이스 드라이버가 필요하다. 이를 위해 각 플랫폼 모듈의 인터페이스 파일에 디바이스 드라이버와 관련된 정보를 명세하고 소프트웨어 생성 시 이들의 정보를 읽어 초기화를 해주고 태스크에서 사용할 수 있도록 관련 헤더 파일을 추가해준다. 초기화와 태스크 생성은 그림5와 같은 순서로 실행된다.

```
initialize_devices();
initialize_channels()

for(i=0;i<tasknum;i++)
    create_task(taskinfo[i].task_id);
```

그림 5. 초기화와 태스크 생성

3.4 소프트웨어 시뮬레이션

시스템에서 프로세서의 시뮬레이션은 GRTOS를 이용한 host에서의 시간 지연 모델을 사용하거나 ISS를 사용하여 시뮬레이션 할 수 있다. 그림6은 프로세서의 모델이다. 인터페이스 파일을 두 가지 방식의 프로세서에 대해 만들어 원하는 시뮬레이션 방식에 따라 플랫폼에서 그림을 바꿔주면 사용이 가능하다.

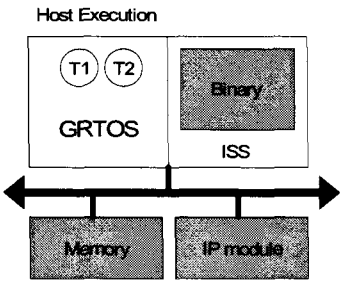

그림 6 프로세서 모델

GRTOS를 사용하는 경우 일반적인 계산 처리는 Host에서 직접 수행되고 일정 단위로 지연 시간을 처리하는 함수를 불러서 시간을 진행시킨다. SDF 모델 기반의 경우에는 각 SDF 블록의 수행마다 성능예측 값만큼을 진행하도록 하였다. 또한 태스크 간의 통신을 위한 공유 메모리로의 접근이나 하드웨어의 접근은 GRTOS의 API를 통해 버스 모델을 호출하여 데이터를 전달하고 시간 지연이 가능하도록 하였다. 그림7는 GRTOS를 사용한 경우 두 태스크 사이의 정보전달이 실제 어떤 과정을 통해 일어나는지를 보여준다. Task1이 Task2에 정보전달을 하기 위해서는 write_port(...)라는 함수를 통해 GRTOS가 관리하는 message_queue에 데이터를 쓰고 GRTOS에서는 이를 실제 버스의 함수 ctb_Bfmwrite(...)를 통해 메모리 모듈에 데이터를 쓴다. 그리고 Task2에서는 read_port(...)를 통해 message_queue에서 데이터를 읽고 GRTOS에서는 내부적으로 ctb_Bfmread(...)를 통해 메모리 모듈에서 값을 읽어서 Task2에게 돌려준다. 만약 message queue가 가득차서 데이터를 쓰지 못하거나 비어서 데이터를 읽을 수 없는 경우에는 태스크를 블록 시켰다가 write 또는 read가 일어나 처리가 가능하게 되면 해당 태스크를 깨운다.

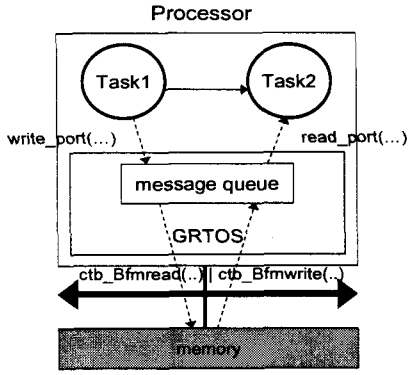

그림 7 GRTOS 시뮬레이션 모델

4. 실험

실험에 사용된 알고리즘은 JPEG decoder이다.

JPEG decoder는 그림 8과 같이 두 개의 태스크로 구성된다.

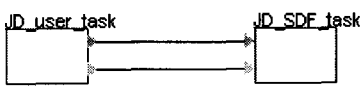

그림 8 JPEG 디코더 태스크

설계에서 사용된 두 가지 방법이 잘 동작하는 것을 보이기 위해 JD_user_task은 사용자가 직접 기술하는 방식을 사용하고 JD_SDF_task는 SDF 모델으로 기술을 하였다. JD_user_task에서는 jpeg파일을 읽어서 각 마커들을 파싱해서 jpeg에 대한 정보를 읽고 이후 처리할 작업에 대한 테이블을 만들어준다. 그리고 매크로 블록의 개수만큼 반복하며 JPEG에서 사용된 허프만 디코딩을 수행해서 JD_SDF_task에서 처리할 데이터를 만들어준다. JD_SDF_task는 그림9와 같은 SDF 블록으로 구성되어있다.

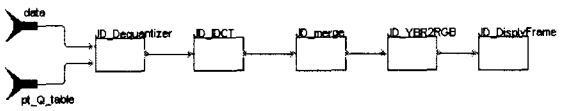

그림 9 JD_SDF_task

JD_user_task에서 처리된 data와 Quantizer에 대한 테이블을 받아 Dequantizer - IDCT에 대해 수행하고 Y, Cr, Cb에 대한 블록들을 모아 RBG로 변경한 후, 가상 LCD로 출력한다.

사용된 플랫폼은 Dynalith에서 제공된 TLM 수준의 플랫폼인 DynSoC으로 그림3과 같은 구조를 가지고 있다.

프로세서로는 ARM7이 사용되고 플랫폼은 LCD 컨트롤러와 가상 LCD, 메모리 컨트롤러, 메모리, 인터럽트 컨트롤러, 타이머, NAND 플래시 컨트롤러, NAND 플래시, UART 컨트롤러, 가상 콘솔, GPIO로 이루어져 있고 커뮤니케이션을 위해 AHB와 APB버스가 사용되었고 연결을 위해 AHB-APB 브릿지가 사용되었다. 생성된 플랫폼은 xml의 형태로 만들어지고 플랫폼에 대한 top module을 생성하는 과정에서 dynalith에서 제공한 top module generator인 scalet를 위한 스크립트를 생성한다. 표1은 생성된 플랫폼의 결과이다. 차지하는 양이 그리 크지 않음을 알 수 있다.

사용된 모듈 개수	19개(버스 2개)
xml의 크기	23,154 bytes
script의 크기	13,277 bytes

표 1 플랫폼 생성 결과

실험에는 QCIF 크기의 이미지가 사용되었다. ISS에 비해 native가 빨리 수행되는 것을 확인할 수 있었다. ISS의 경우에는 디코더 태스크를 하나로 만들어 수행하였다.

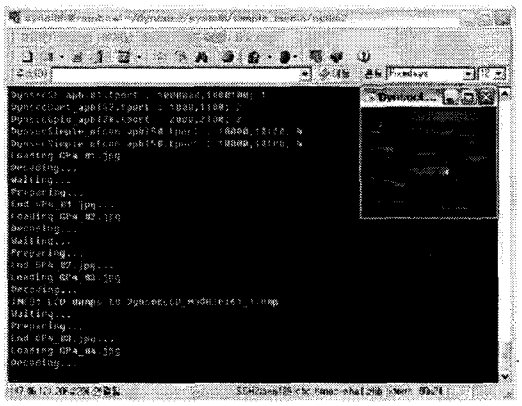

그림 10 시스템 시뮬레이션 결과

5. 결론

이 논문에서는 가상 프로토타이핑 시스템의 자동생성 및 이를 고려하여 소프트웨어를 생성하는 환경을 개발하고 DynSoC 플랫폼 위에서 JPEG 디코더를 수행하였다. 본 프레임워크에서는 플랫폼이 수정되는 경우 이에 대해 수정이 필요한 소프트웨어 코드를 자동으로 생성하기 때문에 초기 플랫폼 위에서 설계공간탐색을 위해 프로그램을 수행하고 변경하는 것이 쉽다는 장점을 가진다.

여기서는 설계공간탐색을 위해 TLM 수준에서의 시뮬레이션만을 수행하여 빠르지만 정확도가 떨어진다. 만약 주어진 모듈에 대한 RTL 수준의 라이브러리도 존재한다면 최종 검증을 위해서는 느리지만 좀더 정확한 RTL 수준의 시뮬레이션도 가능할 것이다.

감사의글

본 연구는 BK21 프로젝트, 과학기술부 도약연구지원사업(R17-2007-086-01001-0), 시스템집적반도체기반기술개발사업에 의해 지원되었다. 또한 서울대학교 컴퓨터신기술연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 또한 한국전자통신연구원의 SoC 핵심설계인력양성사업에 의해 부분적으로 지원되었다.

참고문헌

[1] B. Kienhuis et.Al. "An approach for quantitative analysis of application specific dataflow architectures", Proc. Int'l Conf. Application-Specific Systems, Architectures and Processors, pp. 14-16, Jul 1997.

[2] Coware, "Coware Platform Architect", available at http://www.coware.com/products/virtualplatform.php

[3] Mentor Graphics, "Platform Express Professional", available at http://www.mentor.com

[4] E. A. Lee and D. G. Messerschmitt, "Synthronous Dataflow," IEEE Proceedings, Sep 1987.

[5] S. Yoo, G. Nicolescu, L. Gauthier, A. A. Jerraya, "Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design", Proc. Design Automation and Test in Europe, 2002

[6] Hans-Erik Eriksson, Magnus Penker, "UML toolkit", John Wiley & Sons, 1998

[7] omitted for blind review