論文99-36C-12-5

백플레인에 기반한 제어 부분과 데이터 처리 부분의 통합적 명세

(Co-specification for control and dataflow based on the codesign backplane)

金 度 亨 *, 河 舜 會 *

(Do-Hyung Kim and Soon-Hoi Ha)

요 약

내장형 시스템의 기능에 대한 요구가 증가할수록 전체 시스템 설계의 복잡도는 높아진다. 복잡한 시스템을 설계하기 위해서는 동일한 환경에서 데이터 처리부분과 제어부분을 대등하게 표현할 수 있는 정형적 설계 환경이 필요하다. 본 논문에서 데이터의 처리는 데이터플로우 모델로 묘사하고, 데이터의 흐름의 제어를 Finite State Machine(FSM)모델로 기술한 후 두 모델의 통신으로 시스템을 표현하였다. 일반적으로 시스템을 기술할 때는 분리된 환경에서 각 부분을 만든 후 이를 최종 단계에서 결합하게되어 동작을 검증할 수 없었던 반면, 본 환경에서는 초기 설계 단계에서 부분에 적합한 다른 모델로 설계를 한 후 동일한 환경에서 동작을 검증하고 제품을 생산할 수 있다. 특히 논문에서는 두 모델 사이의 신호의 교환 방법을 제시하고 예제를 통해 유효성을 검증해보았다.

Abstract

As the requirements of embedded systems increase, the design complexity of the system becomes higher. The formal design methodology is required which supports well-balanced specification for control and dataflow to design a complex system. In this paper, control modules and function modules are separately described with FSMs and dataflow graphs respectively, and integrated into a system specification via inter-model communications. In previous approaches, the system could not be verified until control modules and dataflow modules are combined at the final design stage. However our approach enables us to design each part as the proper model of computation at early stage, and to verify the compositions and to co-synthesize the system effectively in the same framework. Especially this paper focuses on the communication protocols between control and dataflow models. Preliminary experiments show practicality of the proposed technique.

I. 서 론

현재 내장형 시스템의 성능은 빠른 속도로 증가하고 사용자는 다양한 기능을 요구하고 있다. 내장형 시스템에 대한 평가도 수행 속도나 정확성뿐만 아니라 실시간 수행에 대한 요구 등의 다양한 기준에 의거한다. 이러한 내장형 시스템에 대한 요구 사항의 증가는 시스템 설계의 복잡도를 증가시킨다. 설계 복잡도의 증가에 대응하기 위한 연구 방법론으로 내장형 시스템의 설계 자동화에 관한 통합 설계 방법론에 대한 연구가 활발히 진행되고 있다[1].

통합 설계 방법론은 시스템을 정형적 언어로 묘사하여 알고리즘을 상위 수준에서의 검증하도록 하고 이를 하드웨어/소프트웨어부분으로 자동 분할하며 각각에 적

* 正會員, 서울大學校 컴퓨터工學科

(Dept. of Computer Eng., Seoul Nat'l Univ.)

※ 본 논문은 1998년 한국과학재단의 핵심전문연구(No. 971-0906-040-2)에 의하여 연구되었습니다.

接受日字:1999年5月24日, 수정완료일:1999年11月9日

합한 코드를 합성하여 빠른 제품의 설계와 생산을 가능하게 한다. 정형적인 상위 수준의 언어를 사용하면 구현과 무관하게 알고리즘을 묘사할 수 있고, 잘 정의되어 있는 수학적 정의를 사용하여 실행 이전에 동작을 검증할 수 있다. 또한 통합 설계에서는 상위 수준의 검증이 합성 단계에 이르러서도 유효하게 되어 설계 및 합성의 반복된 과정에 걸리는 시간을 줄여준다. 통합 설계에 관한 이전의 연구들은 시스템의 기능을 하나의 정형적인 언어로 표현하고 이를 분할하고 자동화된 코드를 생성하는, 분할과 합성에 중심을 둔 하드웨어와 소프트웨어의 통합설계가 중심을 이루었다. 최근에 와서 아날로그 시스템과 디지털 시스템간의 통합설계나 프로세서와 메모리간의 통합설계와 같은 새로운 분야에 대한 연구가 진행되고 있다. 이러한 연구의 연장선상에서 본 논문에서는 제어 부분과 데이터 처리 부분간의 통합 설계에 대하여 살펴보려고 한다.

통합 설계에서 시스템을 표현하기 위해 사용되는 언어는 응용 분야에 따라 다르다. 디지털 신호처리와 같이 계산량이 많은 응용에서는 데이터의 흐름으로 표현되는 알고리즘을 묘사하기 쉽기 때문에 많은 사람들이 데이터플로우(dataflow) 모델을 사용했다. 특히 dataflow 모델 중에 synchronous dataflow(SDF)[2]나 cyclo-static dataflow (CSDF)[3] 모델은 dataflow의 표현을 제한하여 명세할 수 있는 응용의 대상을 줄이는 대신 정형적 분석을 통하여 알고리즘의 엄밀한 검증을 가능케 하였고 코드를 효율적으로 합성할 수 있도록 하였다. 따라서, 디지털 신호처리 응용을 위한 시스템 설계 도구에서 많이 사용된다.

이와는 반대로 제어 부분이 많은 리액티브(reactive) 시스템에서는 유한상태기(Finite State Machine)나 이를 확장한 형태가 시스템을 표현하기 위한 언어로 많이 사용되었다. FSM 모델은 수학적으로 잘 정의되어 있기 때문에 동작의 특성이나 성능에 관한 정형적 분석이 가능하고 효율적인 합성이 용이하다.

지금까지 통합설계의 연구는 응용 분야의 특성에 따라 두 분야로 구분될 수 있다. 디지털 신호처리와 같은 계산 위주의 통합설계 연구에서는 dataflow 모델을 사용하였고 자동제어 시스템과 같은 제어 위주의 응용에서는 FSM 모델이 주가 되었다. 하지만 네트워크의 응용과 멀티미디어 응용이 확대되고 기술이 발전함에 따라 시스템의 표현에 데이터 처리 부분과 제어 부분 모두가 중요한 경우가 증가하였다. 디지털 핸드폰의 예를 들어보면 통화 연결(Call Processing)이나 다중 접근 프로토콜(Multiple Access Protocol)을 처리하기 위해서는 제어 부분이 필요하고 음성 신호를 주고받기 위해서는 많은 양의 계산을 수행하여야 한다. 아날로그로 처리되는 부분을 무시한다 하더라도 이러한 시스템을 설계하기 위해서는 제어 부분과 데이터 처리 부분을 동시에 표현할 수 있는 통합설계 환경이 필요하다.

본 논문에서는 통합 설계(co-design) 환경을 위한 통합 명세 방법(co-specification)을 제안하였다. 제안된 통합 명세 방법에서는 제어부분과 데이터 처리 부분을 통합하여 하나의 시스템을 표현하며 제어 부분은 FSM 모델로, 데이터 처리 부분은 SDF 모델로 설계한 후 두 모델을 결합하기 위한 통신 방법을 제공하여 준다. 이 통신 방법은 통합 설계 백플레인 위에서 구현되어 통합 설계 백플레인이 제공해주는 통합 시뮬레이션[4]을 통해 전체 시스템의 동작을 검증할 수 있다. 이와 유사한 접근 방법으로는 i-Logix inc.에서 제안한 STATEMATE[5] 방법과 U. C. Berkely 대학의 Ptolemy[6] 방법이 있다. 그러나, STATEMATE에서는 표현의 확장으로 언어의 정형성을 보장하기 못하여 정형적 언어의 장점을 이용할 수 없었고 Ptolemy에서는 신호 전달 방법을 한정하여 두 모델간의 통신 방법을 제한하였다. 또한 두 연구들은 데이터 처리 부분과 제어 부분을 대등하게 표현해주지 못했기 때문에 시스템의 구현이 특정 표현에 종속되었다. 하지만 우리의 환경에서는 정형적 언어의 사용으로 인한 상위 수준의 검증과 대등한 표현에 의한 표현과 구현의 독립을 달성할 수 있었다.

2장에서는 통합된 표현에 관한 이전의 연구와 우리의 연구의 주안점을 비교하고 3장에서는 제어 부분과 데이터 처리 부분의 표현에 사용되는 FSM 모델과 SDF 모델에 대하여 살펴본 후 이 두 모델을 결합하기 위해 필요한 제어신호를 정의한다. 4장에서는 통합 명세의 기반이 되는 통합설계 백플레인(Backplane)의 소개와 제어신호의 표현 및 구현에 대하여 설명한다. 이어진 5장에서는 제안된 백플레인을 이용해서 숫자 증가 예제와 MPEG I Layer 3 음악 파일을 디코딩 하는 연주기를 통해 제안된 방법의 유효성을 검증해보고 6장에서는 본 연구의 중요성과 이후의 연구에 대하여 이야기한다.

II. 관련 연구

여기서는 데이터 처리 부분과 제어 부분을 하나의 환경에서 동시에 표현해 줄 수 있는 이전의 접근방법들로 statechart[7]에 기반을 둔 STATEMATE 환경과 Ptolemy 환경을 살펴보도록 하겠다.

STATEMATE[5]는 복잡한 제어 중심의 시스템을 표현하기 위한 설계 환경으로 시스템의 물리적 구조를 보여주는 module-chart, 기능들의 결합과 정보의 흐름을 표현한 activity-chart, 시간의 흐름에 따른 시스템의 동작을 기술한 statechart, 이렇게 세 개의 그래프로 구성된다. STATEMATE에서는 전통적 FSM을 확장한 statechart가 모든 시스템의 기능 블록들을 제어하게 하고, 우리의 백플레인과 유사한 activity-chart에서는 시스템의 데이터 처리 부분을 블록 별로 구분하고 이들 사이의 데이터의 흐름으로 데이터 처리 부분을 표현하였다.

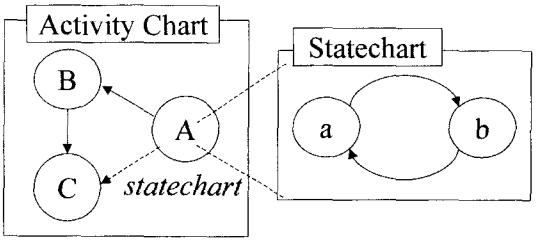


그림 1. STATEMATE에서의 신호의 표현
Fig. 1. specification of interactions in STATEMATE.

그림 1은 STATEMATE에서의 신호 전달을 보여준다. STATEMATE에서는 제어부분과 데이터 처리부분을 연결하기 위해 동기 제어신호와 비동기 제어신호를 모두 지원해주며 activity-chart위에서 동기 신호는 실선으로 비동기 신호는 점선으로 표현된다. 각 제어신호의 의미에 관해서는 3장에서 자세히 설명하기로 하겠다.

STATEMATE는 제어 중심의 시스템을 표현하기 위한 매우 강력한 설계 환경을 제공해주지만 activity-chart에는 실제 데이터가 언제 전달되고 어떻게 데이터 처리 블록이 실행될 지에 관해서는 어떠한 정보도 포함되지 않고 모든 실행은 statechart의 제어에 의해 이루어진다. 이러한 제어 중심의 시스템 표현은 데이터 처리 부분이 독립적으로 동작하는 시스템의 설계를 어렵게 하고 데이터 처리 부분의 구현이 제어 부분의 표현과 구현에 종속되어 통합설계 시스템 표현으로써는 한계를 보여준다. 또한 데이터 처리 부분은 정형적 모델로 표현되지 않고 제어 부분을 나타내는 statechart도 표현의 확장으로 인해 FSM이 가지는 정형적 모델의 성질을 가지지 않는다.

Ptolemy[6] 환경은 계층성을 가진 계산 모델(models of computation)간의 신호 교환 방법을 정의해 주어 여러 모델을 하나의 틀에서 표현하고 시뮬레이션을 통한 검증을 가능하게 해준다. Ptolemy에서 각 모델은 계산을 담당하는 블록들과 이들을 연결하는 선으로 이루어진 그래프로 표현되고 하나의 모델로 구성된 그래프를 도메인(domain)이라고 부른다. 각 도메인은 계산모델에 따라 그래프의 의미를 다르게 해석한다. 다른 계산모델들을 연결하기 위해서는 하나의 계산모델로 구성된 그래프가 다른 그래프의 한 블록 내에서 계층적으로 사용되고 이 블록을 특별히 웜홀(wormhole)이라고 한다. 웜홀은 다른 도메인간의 신호를 교환하는 방법을 정의해주어 외부 도메인에서 오는 신호를 받아 자신의 내부의 도메인으로 전달시켜준다. Ptolemy에서 제어 부분과 데이터 처리 부분의 표현은 이러한 웜홀을 이용하여 이루어진다. 그림 2와 같이 FSM 모델은 데이터플로우 모델 속에 포함될 수 있고 그 반대의 표현도 가능하다. 특히 Ptolemy에서는 FSM을 이용할 때 갖는 어려움의 하나인 상태폭발문제(state explosion problem)를 극복하기 위해 다양한 병렬적인 모델 속에서 계층적 FSM을 표현하는 방법을 제안하였고[8], 이는 언어의 정형성을 보전하면서 제어 부분과 데이터 처리 부분의 표현을 가능하게 해주었다.

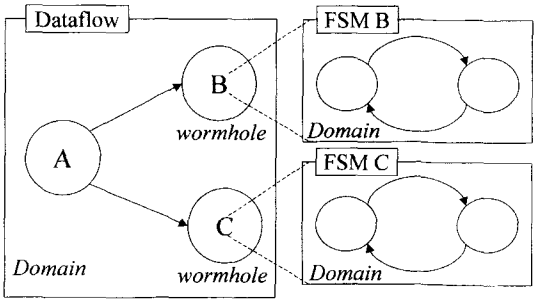


그림 2. Ptolemy에서 계층적 FSM의 표현
Fig. 2. Hierarchical FSM in Ptolemy.

그림 2는 Ptolemy에서의 계산모델간의 신호 교환 방법을 보여준다. 데이터플로우 블록 A에서 블록 B로 데이터가 전달되면 데이터를 받는 블록 B는 FSM 그래프를 포함한 웜홀이므로 외부에서 받은 신호를 내부의 FSM B 그래프로 전달시켜준다. 이는 데이터에 동기화

되어 동작하므로 동기 제어신호의 형태를 지닌다.

Ptolemy에서는 다양한 정형적 모델의 결합으로 시스템을 표현하여 시스템의 설계의 복잡도를 줄이려고 시도하였다. 하지만 다양한 모델간의 연결에서 웜홀을 사용한 통신방법은 동기 제어신호만을 표현 가능하고 3장에서 보일 제어부분과 데이터 처리부분을 연결하기 위해 필요한 통신 방법인 비동기 제어신호를 지원하지 않는다. 때문에 데이터 처리부분의 실행흐름을 FSM 모델에서 제어하기 위해서는 FSM 모델의 한 노드로 데이터플로우 모델을 표현해야 한다. 하지만 이 방법도 데이터 처리부분을 멈추고 시작하는 정도만 지원할 수 있다. 이러한 제한에 의해 Ptolemy 환경은 제어부분과 데이터 처리부분의 분리된 표현을 달성할 수 없으며 시스템의 설계가 복잡해지고 코드의 합성이 어렵게 된다.

이에 반해 제안된 방법은 STATEMATE에서 지원해주는 표현의 확장을 포함하고 Ptolemy에서 지원하는 정형적 언어의 표현을 가능하게 하였다. 제어 부분에는 정형적 성질 지닌 FSM을 사용하고 데이터 처리 부분에서는 SDF와 같은 정형적 언어를 사용하여 상위 수준에서의 검증을 가능하게 하고, 분리된 표현을 연결시켜 주기 위한 동기 제어신호와 비동기 제어 신호를 통합 설계 백플레인 위에서 제공하여 전체 시스템을 명세 할 수 있게 했다. 또한 통합설계 백플레인이 제공해주는 통합 시뮬레이션을 통하여 시스템 전체를 검증할 수 있는 방법을 제공해준다. 통합설계 백플레인은 STATEMATE의 activity-chart와 유사하게 데이터 처리 블록과 제어 부분의 블록들, 외부 환경에 대한 블록들이 존재하고 각 블록들 사이의 데이터의 흐름을 표현하여 준다. 하지만 activity-chart와는 다르게 언제, 어떻게 데이터가 전달될지의 정보를 가지고 있어 데이터 처리 부분과 제어 부분을 대등하고 독립적으로 표현해준다. 또한 기존의 Ptolemy에서 제공해주는 통신 방법을 확장하여 비동기 제어 신호를 제공하여 줌으로써 제어 부분과 데이터 처리 부분을 독립적으로 설계하고 결합할 수 있는 환경을 구축하였다.

Ⅲ. 데이터 처리 부분과 제어 부분의 통합 표현

본 장에서는 데이터 처리 부분과 제어 부분에 사용할 정형적 언어인 SDF와 FSM 모델에 대하여 설명하고 이를 통합하기 위한 통신 요구 사항을 분석하여 FSM 모델에서 SDF 모델을 제어할 수 있는 신호들을 정의하고 각 신호들의 특징에 대하여 살펴보겠다.

데이터 처리 부분을 표현해 주는 dataflow 모델에서는 계산을 수행하는 노드와 데이터의 흐름을 나타내는 연결선으로 시스템을 표현해 주고 실행 시에 각각의 노드는 각 입력으로부터 일정한 수의 데이터를 입력에서 받아 일정한 수의 데이터를 출력으로 보내준다. (그림 3참고)

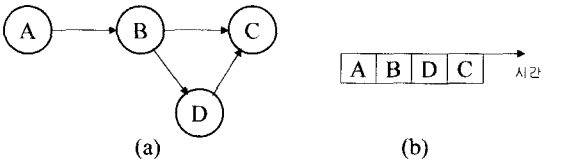


그림 3. (a) dataflow graph (b) static scheduling
Fig. 3. (a) dataflow graph (b) static scheduling.

SDF 그래프는 각 노드들 간의 상대적인 우선 순위를 정의하기 때문에 컴파일 시점에 정적으로 노드들의 수행 순서를 결정할 수 있다는 장점이 있다. 그림 3(a)은 네 개의 노드를 가진 SDF 그래프를 보여준다. 노드들은 수행을 위한 단위가 되고 화살표는 노드들 간의 데이터의 전달을 나타낸다. 각 노드들이 하나의 입력을 받고 하나의 출력을 내보낸다면 그림 3(a)의 그래프를 수행할 수 있는 효과적인 스케줄링은 그림 3(b)와 같이 결정된다. 그림 3(b)는 시간에 따른 노드들의 수행 순서를 보여주고 한 번의 수행을 마치면 처음과 같은 상태로 돌아간다.

표 1. FSM 모델의 수학적 표현
Table 1. Formal representation of FSM.

$F = (Q, \Sigma, \triangle, \delta, q_0)$

Q : 상태를 나타내는 유한의 값들의 집합

Σ : 가능한 입력을 나타내는 값들의 집합

$\triangle$: 가능한 출력을 나타내는 값들의 집합

δ : $Q \times \Sigma$에서 $Q \times \triangle$로의 상태변화 함수

q_0 : $q_0 \in Q$ 를 만족하는 초기 상태

제어 부분을 위한 FSM 모델은 일반적으로 표 1과 같이 수학적으로 나타내고 이를 그래프나 텍스트로 표

현한다. FSM 모델은 시간의 흐름에 따른 입력 값과 출력 값의 관계를 나타내기에 적당하여 reactive 시스템의 표현에 많이 사용되었다. 정적으로 수행에 필요한 메모리의 요구량이나 수행시간을 예측할 수 있고 유한한 상태를 가지는 특성 때문에 이론적으로는 모든 노드들을 검사하여 잘못된 동작을 사전에 찾아낼 수 있다. 이러한 FSM 모델을 이용한 통합 설계 방법론으로는 Vincentelli[9]등이 제안한 Codesign FSM(CFSM) 표현이 있다. 여기서는 여러 개의 FSM을 네트워크로 구성하여 이들의 통신으로 시스템을 표현해 주었고 CFSM 모델이 FSM 모델로 변환될 수 있음을 보였다.

하지만, 전통적인 FSM 모델은 메모리나 병렬성을 가지는 시스템에서 상태의 개수가 비약적으로 증가하는 상태폭발문제가 발생한다. 이를 극복하기 위하여 Harel은 FSM 모델에 계층성과 병렬성을 부여한 statechart[7] 모델을 제안하였다. 그러나, statechart 모델은 표현의 확장으로 인하여 FSM 모델이 가지는 정형적 성질을 상실하였고 상태 변화 시의 동작을 정확히 정의하지 않아 모호성을 가지기 때문에 많은 변형들이 제안되었다[10]. 우리가 지원하는 FSM 모델은 statechart 모델과 같이 계층성과 병렬성을 지원하지만 statechart 모델의 표현을 제한하여 정형성을 유지하고 모호성을 배제하였다. 이에 관한 내용은 본 논문의 내용을 벗어나므로 차후의 논문에서 다루도록 하고 여기에서는 설명을 간단히 하기 위하여 전통적인 FSM 모델을 사용하기로 한다.

데이터 처리 부분과 제어 부분의 통신 요구 사항을 정의하기 위해서는 일반적인 내장형 시스템에서 두 모델이 주고받는 신호의 종류를 살펴보아야 한다. 그림 4는 외부환경과 데이터 처리 부분, 제어부분의 신호 교환을 나타낸다. 데이터 처리 부분은 입력으로 외부의 신호나 다른 데이터 처리 부분으로부터 입력을 받을 수 있고 또는 제어 부분으로부터도 입력을 받을 수 있다. 그리고 출력을 외부나 다른 데이터 처리 부분, 제어 부분으로 내보낸다. 제어 부분에서도 상황은 비슷하게 외부의 신호나 데이터 처리 부분, 다른 제어 부분에서 입력을 받고 출력을 내보낸다.

데이터 처리 부분과 제어 부분이 주고받는 신호는 제어 부분에서 데이터 처리 부분으로 가는 제어 신호와 데이터 처리 부분에서 제어 부분으로 가는 신호가 있다. 후자의 경우에는 제어 부분의 입력으로 들어오는 신호로 외부 환경과 주고받는 신호와 그 형태가 같다. 하지만, 전자의 경우에는 데이터 처리 부분을 제어하기 위한 신호로써 본 논문에서 가장 중요하게 다루게 되는 데이터 처리 부분과 제어 부분의 통신 인터페이스를 담당한다.

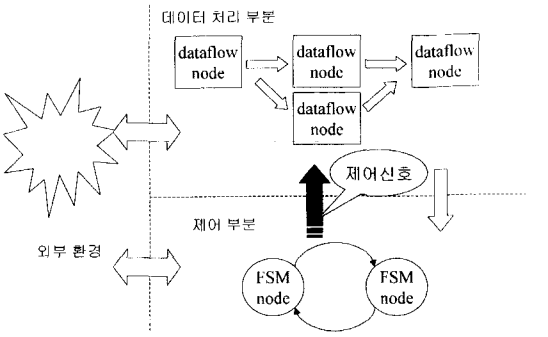


그림 4. 제어 부분과 데이터 처리 부분의 통신
Fig. 4. Communications of control and dataflow modules.

제어신호는 FSM에서 SDF에 영향을 미치기 위해, 즉 제어 부분에서 데이터 처리 부분을 조정하기 위해 발생시키는 신호이고 두 모델이 통신하는 데에 필요한 인터페이스의 역할을 해준다. 보통의 신호의 전달은 대상이 되는 노드가 입력에 데이터가 들어오면 값을 읽어 수행하여 결과를 내보내 주지만 제어신호의 경우에는 대상이 되는 노드의 수행의 흐름을 조절하기 위해 데이터 처리 부분의 외부 스케줄러가 관여되거나 노드 내부의 값을 조절해주는 것과 같이 일반적인 데이터 전달의 통로 외의 방법이 필요하다. 내장형 시스템에서 요구되는 제어신호의 패턴을 살펴보면 크게 데이터와 동기화 되어 동작하는 신호와 비동기적으로 발생하여 처리되는 신호로 나누어 볼 수 있다.

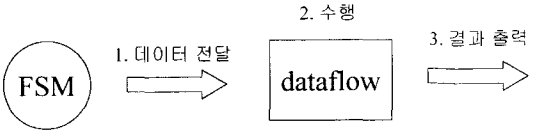


그림 5. 동기 제어 신호의 전달
Fig. 5. Transmission of synchronous control signal.

동기 제어신호의 경우 제어 부분과 데이터 처리 부분은 데이터에 의해 동기화 되어 통신한다. 데이터 처리 부분에서는 제어 부분에서 오는 데이터를 기다리며 수행을 준비하고 제어 부분에서는 데이터를 데이터 처리 부분으로 전달시켜 수행시킨다. 데이터 처리 부분은

제어 부분에서 데이터가 도착하면 동작을 시작하고 데이터에 대한 계산을 마치면 동작을 끝낸다. 그림 5는 동기 제어 신호의 전형적인 예를 보여주는데 FSM이 동기 제어 신호를 발생시키면 dataflow는 데이터를 받아 수행을 시작하고 수행이 끝나면 결과를 출력해 준다.

비동기 제어신호는 제어 부분이 데이터 처리 부분의 상태를 조절할 때 사용된다. 이의 대표적인 예로 그림 6(a)와 같이 데이터 처리 부분의 스케줄링을 조절해 줄 때 비동기화 제어신호를 사용할 수 있다. 프로세서의 동작을 보면 수행이 되고 있는 active 상태, 외부의 신호를 기다리고 있는 wait 상태, 수행을 멈춘 halt 상태로 나누어 볼 수 있다. 비동기 제어 신호는 데이터 처리 부분의 각 블록의 수행 상태를 조절해 주는 역할을 하게 된다. 비동기 제어 신호의 다른 형태는 그림 6(b) 처럼 제어 블록이 데이터 처리 블록의 변수 값을 비동기적으로 바꾸어 주고 싶은 경우가 있다. 예를 들어 음악이 연주되고 있는 도중에 사용자가 소리의 크기를 조절했을 경우에는 제어 부분에서는 이 신호를 받아 데이터 처리 부분에서 소리 출력을 담당해 주는 블록의 "gain" 변수를 비동기적으로 바꾸어준다.

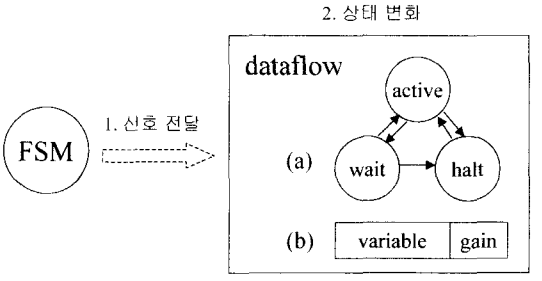


그림 6. 비동기 제어신호의 전달
Fig. 6. Transmission of asynchronous control signal.

실제 시스템에서는 제어 블록의 역할에 따라 한쪽의 제어신호가 우세한 경우가 많다. 만약 FSM이 데이터 처리 블록을 함수와 같이 호출해 주는 제어 구조라면 동기화 제어 신호가 많이 사용될 것이고 FSM이 연속적으로 동작하는 데이터 처리 블록의 스케줄링을 조절해 주는 역할을 담당한다면 비동기 제어신호가 우세할 것이다. 따라서 FSM과 dataflow를 통합해서 표현하기 위해서는 동기 제어신호와 비동기 제어신호를 모두 지원해 줄 수 있어야 한다.

Ⅳ. 통합설계 백플레인

통합설계 백플레인 환경은 서로 다른 계산 모델간의 통신을 지원해주는 통신의 백본(backbone)으로 동작하고 그 위에서 동작하는 모든 시스템의 부분 모듈들에 대한 제어권을 가지고 있다. 즉, 모든 모듈간의 통신은 백플레인을 통해서 이루어진다. 통합설계 백플레인은 통합설계를 위한 통합 명세, 통합 시뮬레이션 그리고 통합 합성을 지원하고 이 논문에서는 백플레인의 통합 명세 부분에 관심을 두고 있다. 본 논문에서 제안하는 통합 명세 방법은 제어 부분과 데이터 처리 부분을 각각 FSM 모델과 SDF 모델로 표현한 후 이들을 연결하기 위한 동기 제어 신호와 비동기 제어 신호를 통합설계 백플레인 위에서 구현하여 준다. 그리고, 통합설계 백플레인에서 제공해주는 통합 시뮬레이션[4]을 이용하여 통합 명세 된 시스템의 동작을 검증할 수 있다.

통합설계 백플레인은 우리 연구실에서 개발중인 PeaCE(Ptolemy extension as a Codesign Environment) 환경을 위해 구현되었고 통합 명세를 위하여 제어 부분과 데이터 처리 부분을 연결해 줄 수 있게 동기 신호와 비동기 신호의 표현을 지원한다. PeaCE는 통합설계 환경으로써 Ptolemy를 확장한 도구이며 통합설계 백플레인은 Ptolemy에 구현된 DE(Discrete Event) 모델[11]을 기반으로 설계하였다. DE 모델에서 각 노드는 태스크를 수행하는 데에 소요되는 시간이라는 정보를 보유하고 있다. 각 노드는 어느 입력으로부터이든지 샘플이 도착하면 그 샘플을 사건(Event)의 발생으로 간주하여 동작을 시작한다. 이를 위하여 백플레인 도메인의 스케줄러는 전역적인 Event Queue를 이용하여 블록도의 각 노드에서 발생하는 샘플(혹은 사건)을 그의 발생 시간이 작은 순서대로 정렬하며 가장 빠른 사건을 먼저 처리하는 방식으로 동작을 수행하게 된다.

1. 동기 제어신호의 표현

동기 제어신호는 백플레인에서 계층적으로 구성된 제어 부분과 데이터 처리 부분간에 직접 연결로 표현된다. 그림 7에서 보는 것처럼 백플레인에서 계층적인 블록으로 FSM 모델과 SDF 모델이 표현되고 동기 제어신호는 두 모델사이를 백플레인을 통해 전달된다. FSM이 X 상태에 있을 때 외부에서 "a" 신호가 도착하면 Y 상태로 상태천이를 하면서 "out"이라는 동기 제어신호를 백플레인을 통해 SDF로 전달해준다. 입력값을 받은 데이터 처리 부분은 수행을 시작하여 출력에 값을 내보내고 실행을 멈춘다. 동기 제어신호에 의

한 데이터 처리 부분의 제어는 제어 부분이 데이터를 데이터 처리부분에 보내어 주어 함수의 호출과 같이 이루어진다.

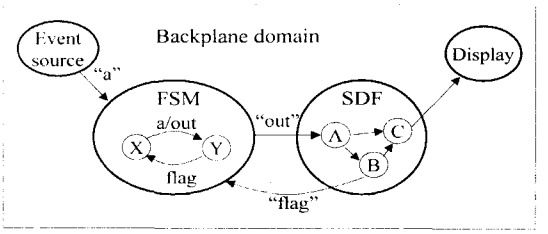


그림 7. 통합설계 백플레인에서의 동기 제어신호의 전달
Fig. 7. Synchronous control signal in the codesign backplane.

그림 7에서 Event Source와 Display 블록은 백플레인 위에서 특별한 기능을 수행한다. Event Source 블록은 시스템을 시뮬레이션하기 위한 테스트 벡터를 발생시킨다. 또는 시뮬레이션 상에서 시스템과 사용자간의 통신을 위한 사용자 환경을 제공해주는 역할을 할 수 있다. Display 블록은 데이터 처리 부분으로부터 오는 시뮬레이션의 결과를 출력시켜서 값을 확인할 수 있게 해준다. 백플레인 방법의 사용은 제어부분과 데이터 처리부분의 통합된 표현을 가능하게 할뿐만 아니라 동일한 환경에서 시뮬레이션을 위한 외부 입력과 출력을 묘사해줄 수 있는 부가적인 이점이 있다.

2. 비동기 제어신호의 표현

비동기 제어신호를 표현할 수 있는 방법은 여러 가지가 존재할 수 있다. 우리는 통합설계 백플레인 위에서 FSM 모델의 상태 속에 script를 써주어 비동기 제어신호를 표현하였다. Script는 무어 FSM 모델의 형태로 비동기 제어 신호를 표현하여주고 데이터 처리 부분의 모듈을 이름으로 직접 호출하는 형태를 지닌다. 비동기 제어 신호는 동기 제어 신호와는 달리 블록간에는 명시적인 연결이 없고 제어를 위한 대상 블록에 이름을 부여하고 이를 FSM 노드의 script에서 불러주면 신호를 받은 백플레인이 대상 블록으로 전달해준다.

표 2는 우리 환경에서 지원해주는 비동기 제어신호의 종류를 보여준다. 처음에 있는 세 가지 신호는 데이터 처리 부분의 실행 상태를 조절해주는 제어신호이고 마지막 신호는 비동기적으로 데이터 처리 부분에 있는 변수의 값을 조절해주기 위한 제어신호이다. 실행 상태를 조절하기 위한 비동기 제어신호 중에 suspend와 stop은 비슷한 의미를 지니지만 suspend는 run 상태로 이동하였을 때 자신이 멈추었던 곳에서부터 수행을 시작하고 stop은 처음 상태로 돌아가서 수행을 진행시킨다.

표 2. 비동기 제어신호를 위한 scripts
Table 2. Scripts for asynchronous control signal.

script	동 작
run *n_name*	n_name 블록의 수행을 진행
suspend *n_name*	n_name 블록의 수행을 잠시 멈춤
stop *n_name*	n_name 블록의 수행을 멈춤
set *n_name* parameter value	n_name 블록의 parameter 변수를 value라는 값으로 설정

통합설계 백플레인에서 비동기 제어신호는 그림 8에서 보이는 것과 같이 전달된다. FSM 모델에서 script를 이용해 비동기 제어신호를 백플레인으로 넘겨주면 백플레인은 script를 해석하여 *n_name*에 지정된 블록으로 신호를 전달시켜 준다. 그러나 비동기 제어신호는 백플레인에서 그래프간의 선의 연결로 나타내지 않고 FSM의 script에서는 블록의 이름을 지정해주면 신호를 전달받은 백플레인에서는 지정된 이름을 가진 블록을 찾아 비동기 제어 신호를 전달시켜준다.

비동기 제어신호를 받은 데이터 처리 블록은 그 내용에 따라 자신의 수행상태를 조절하거나 자신이 가지고 있는 변수를 지정된 값으로 바꾸어 준다. 그림 8에서 X 상태에 있을 때 Event Source 블록으로부터 "a"라는 신호가 전달되면 상태변화가 일어나면서 Y 상태에 지정되어 있는 "stop foo" 스크립트가 백플레인으로 전달되고 백플레인은 foo라는 이름을 가진 데이터 처리 블록으로 비동기 제어신호를 전달시켜 준다. stop 제어신호를 받은 SDF 블록은 초기 상태로 자기 자신을 되돌린 다음 수행을 중단한다.

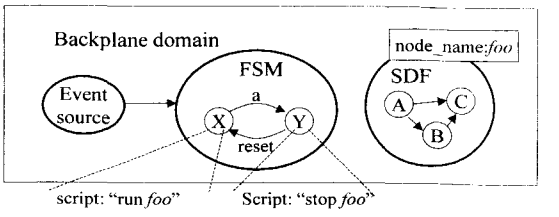


그림 8. 통합설계 백플레인에서 비동기 제어신호의 전달
Fig. 8. Asynchronous control signal in the codesign backplane.

통합설계 백플레인에서 데이터플로우 모델과 FSM 모델과의 신호전달은 실행 환경에 따라 구현 방법은 다르게 이루어진다. 순수한 시뮬레이션에서는 일반적인 함수 호출로 이루어지고 통합 시뮬레이션[4] 환경에서는 TCP/IP 환경을 이용한다. 따라서, 그림 8에서 데이터 처리 부분인 SDF 블록을 통합 시뮬레이션을 위한 C 코드로 합성할 때는 비동기 제어 신호를 처리해 주기 위한 부분도 같이 생성시켜 주어야 한다. 그림 9는 SDF 그래프가 C 코드로 합성될 때의 비동기 제어 신호의 처리와 백플레인과의 연결을 보여준다. SDF 블록을 합성하여 수행시킬 때에는 합성된 코드 앞에 제어 정보를 처리해 주기 위한 코드가 붙고 통합설계 백플레인과는 TCP/IP 연결을 통해 통신한다. 그림 9의 합성된 C 코드에서는 run 상태에 있을 때 non-blocking 읽기를 사용하여 새로운 제어신호가 도착했는가를 검사하고 아무런 신호가 도착하지 않았을 때는 데이터 처리 부분의 수행을 진행한다. 만약 제어신호가 도착하였을 때는 자신의 상태를 바꾸고 수행을 처음부터 다시 시작하게 한다. 이러한 방법으로 run 상태에서 데이터 처리 부분은 백플레인과 병렬적으로 수행된다. 만약 stop 상태로 바뀌었을 때는 데이터 처리 부분의 수행을 멈추고 다음 제어신호의 도착을 기다려야 하므로 blocking 읽기를 사용하여 새로운 제어신호를 기다리며 수행을 중지한다.

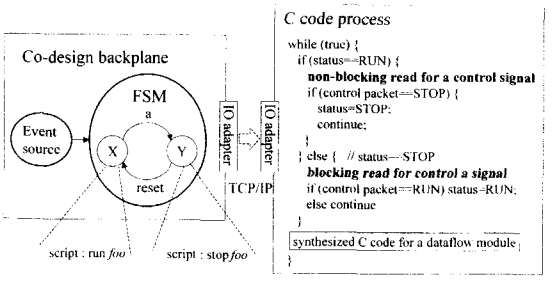


그림 9. 백플레인에서 합성된 SDF 블록과 FSM 모델 간의 비동기 제어신호의 전달

Fig. 9. Asynchronous control signal between a synthesized code and FSM in the codesign backplane.

V. 구현 및 실험

이번 장에서는 위에서 설계된 두 가지의 예제를 준비했다. 첫 번째로 간단한 숫자 증가 예제를 만들어 제안된 비동기 신호가 우리의 환경에서 어떻게 동작할 수 있는지 보여주고 두 번째로 MPEG I (audio) Layer 3를 연주해주는 연주기를 만들어 실제 응용에서 제안한 통합설계 환경의 실용성을 증명하였다.

1. 숫자 증가 예제

숫자 증가 예제는 실행을 시작하면 그림 10과 같이 화면의 아래쪽에서 계속 증가하는 숫자를 볼 수 있고 사용자는 toggle 버튼을 눌러 숫자의 증가를 멈추거나 다시 진행시킬 수 있다. 숫자가 증가하고 있을 때 reset 버튼을 누르면 숫자가 0에서부터 다시 증가하기 시작한다.

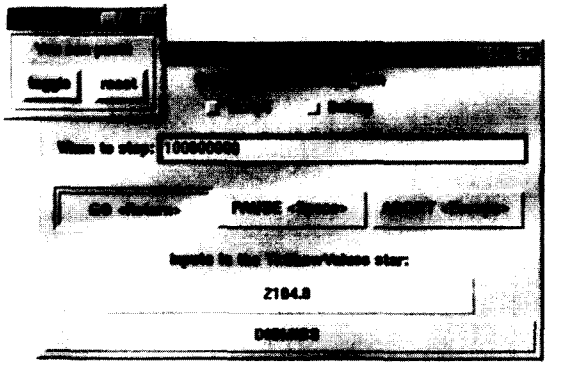


그림 10. 숫자 증가 예제의 실행 화면

Fig. 10. Execution display of a counter example.

그림 11은 우리의 통합 설계 환경인 PeaCE에서의 숫자 증가 예제를 보여준다. 제일 상위 수준이 통합설계 백플레인에서의 설계 화면이다. 계층적으로 구성된 counter_fsm 블록은 내부에 FSM을 가지고 toggle 신호에 의해 start 상태와 halt 상태 사이에 상태 천이가 일어난다. toggle 신호는 TclScript 블록에 의해서 발생하며 TclScript 블록은 사용자의 입력을 받아서 FSM에게로 전달시켜주는 사용자 인터페이스를 담당한다. halt 상태로 들어오면 제어 부분인 FSM에서는 "suspend counter"라는 스크립트를 해석하여 백플레인으로 신호를 전달시켜주고 백플레인은 counter 노드의 실행 상태를 바꾸어 숫자의 증가를 멈추어준다. 숫자를 증가시켜 주는 역할을 담당하는 Ramp 노드는 node_name의 값을 counter로 초기에 설정하여 FSM에서 발생한 비동기 제어신호가 백플레인에 의해 Ramp 노드에 전달될 수 있도록 해준다. toggle 버튼을 다시

누르면 halt 상태에서 start 상태로 넘어가면서 "run counter" 라는 스크립트를 실행시켜 다시 숫자가 증가하게 된다. start 상태는 계층적 FSM으로 구성되며 reset 상태를 내부에 가지는데 여기서는 숫자가 증가하고 있을 때 reset 신호가 TclScript 블록에서 전달되면 Ramp 블록에 있는 value라는 변수의 값을 0으로 되돌려 처음부터 다시 시작되게 한다.

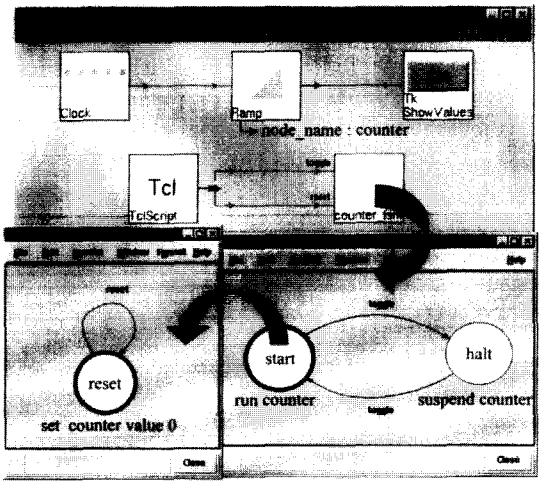


그림 11. 백플레인에서의 숫자 증가 예제
Fig. 11. A counter example in the codesign backplane.

2. MPEG I Layer 3 (MP3) 연주기

MPEG I Layer 3 연주기는 MP3 파일을 디코딩하여 스피커로 보내주는 데이터 처리 부분과 이를 연주를 멈추고 시작해 주는 제어 부분으로 구성한 예제이다. (그림 12) 예제에서 데이터 처리 부분인 MP3 디코딩 부분은 SDF 모델로 알고리즘이 설계되고 실행 시에는 그림 9에서 보인 것과 같은 C 코드로 합성되어 통합설계 백플레인과 병렬적으로 통합 시뮬레이션을 수행하여 준다.

예제는 그림 12에 보는 것과 같이 크게 데이터 처리 부인 mp3cgc 블록과 제어를 담당하는 fsm_07블록, 그리고 사용자의 입력을 받아 제어부로 전달해 주는 TclScript 블록으로 구성된다. MP3 디코딩을 담당하는 mp3cgc 블록은 파일에서 mp3 파일을 읽어 Huffman 디코딩 후에 주파수 대역의 소리로 만든 후에 이를 다시 시간 축으로 변환해 스피커로 출력해준다[12]. 실제 설계 화면은 두 개의 계층으로 구성되어 있고 화면도 복잡하기 때문에 간략화 된 그림으로 표시하였다. 그리고 node_name이라는 변수에 "mp3"라는 블록 이름을 설정하여 FSM 모델에서 발생시킨 제어신호를 백플레인을 통해 받을 수 있게 해준다. SDF 모델로 설계되어 있는 MP3 디코더 부분은 수행 시에 C 코드로 합성되어 통합설계 백플레인과 TCP/IP 연결로 제어신호를 주고받으며 병렬적으로 수행된다. 제어 부분인 fsm_07 블록은 FSM 모델로 설계되었고, start 상태와 stop 상태와 입력 신호로 TclScript에서 오는 "toggle" 신호를 지닌다. 사용자가 입력하는 "toggle" 신호에 의해 start 상태와 stop 상태로 상태전이가 발생하고, start 상태에서는 "run mp3"라는 script를 백플레인에 전달하여 mp3cgc 블록의 실행을 진행시키고 "stop" 상태에 있을 때는 "suspend mp3" script를 전달하여 mp3cgc 블록의 실행을 잠시 멈추게 하여준다. 사용자의 입력을 받는 TcpScript 블록은 toggle 버튼을 사용자에게 보여주고 버튼을 눌렀을 때 제어 부분인 fsm_07 블록으로 "toggle" 신호를 전달하여 준다.

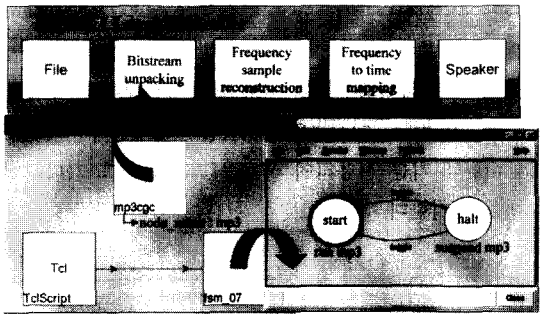


그림 12. MPEG I (Audio) Layer 3 연주기 : 실행시간에 합성된 C 코드로 실행되는 데이터 처리 부분과 백플레인 위의 제어 부분간의 통합시뮬레이션 예제
Fig. 12. MPEG I (Audio) layer 3 player : a co-simulation example of a software simulator and the codesign backplane.

수행 시에는 MP3 디코더 부분은 C 코드로 합성된 후 컴파일 되어 다른 프로세스로 실행되고 백플레인과는 TCP/IP로 연결되어 제어신호를 주고받는다. 수행이 시작되면 MP3 파일이 연주되고 사용자가 toggle 버튼을 눌러 연주를 멈추거나 다시 진행시킬 수 있다. 본 예제는 Sun Ultra1 기계에서 수행하였을 때 실시간으로 MP3 음악을 들려주었다. 하지만 실행이 시작되면 C 코드로 작성된 디코더 쪽의 속도가 빨라서 스피커의 버퍼에 데이터가 많이 쌓여 시간이 조금만 지나면 통

합 설계 환경 쪽에서 제어 신호를 보내어도 쌓여 있는 버퍼만큼의 음악은 연주되고 동작을 멈추는 현상이 발생한다. 이 현상은 MP3는 원래 일정 비율로 수행하게 정해져 있고 이에 따라 1초에 처리해야 하는 데이터의 양은 정해져 있지만 SDF에는 실행의 흐름만을 묘사할 수 있기 때문에 수행의 주기는 기계의 속도에 의해 결정되어서이다. 이를 통하여 실시간 응용을 위해서는 실행의 흐름뿐만 아니라 실행 주기도 표현할 수 있어야 하고 실제 코드를 생성시켜 줄 때도 이를 반영하여야 한다.

VI. 결론 및 향후 연구

본 논문에서는 데이터 제어 부분이나 데이터 처리 부분을 시스템을 기술할 때 각각의 모델에 대하여서는 표현 방법이나 검증방법들이 잘 정의되어 있는 반면에 이 모델들을 통합하여 표현하는 문제에 있어서는 많은 연구들이 이루어지지 않았다는 사실에 주목하였다.

이에 제어 부분과 데이터 처리 부분을 동일한 환경에서 통합하여 표현할 수 있는 설계 환경을 현재 개발중인 통합 설계 환경인 PeaCE의 일부분으로 구축하였다. 통합설계 환경에서는 제어 부분은 FSM 모델로 표현하고 데이터 처리 부분을 SDF 모델을 사용하여 정형적 언어가 가지는 장점을 이용할 수 있게 하였고, 통합설계 백플레인 위에서 두 모델의 동등한 표현과 동기 제어신호와 비동기 제어신호 통신을 가능하게 했다. 다른 관련된 연구와의 비교와 실제 예제를 통하여 통합 설계 백플레인 방법은 제어 부분과 데이터 처리 부분의 통합 표현 문제를 훌륭하게 해결할 수 있음을 보여주었다. 우리의 환경에서는 각각의 모델들이 정형적 언어로 표현되기 때문에 통합된 표현에서 자동적으로 효율적인 코드의 합성을 이끌어 내는 것은 어려운 문제가 아니라고 예측된다.

본 논문은 통합설계의 시스템의 표현 단계에서 다양한 이질적인 환경을 다른 언어 모델로 표현하고 이를 위한 통신 방법을 제공하여 복잡성을 줄일 수 있다는 것을 보여 주었다. 하지만, 연구의 초기 단계이기 때문에 시뮬레이션을 위한 환경만을 구현하였고 통합설계에서 검증이나 실제 합성 단계에서 통합된 표현 방법을 적용시키기 위해서는 고려해 주어야 할 사항이 많다. 제어 부분과 데이터 처리부분이 통합된 시스템 표현에서 하나의 통합된 코드를 합성하기 위해서는 FSM 모델의 코드 합성과 통합설계 백플레인 자체를 작은 운영체제로 합성해내는 것은 아직 진행하여야 할 과제들이다. 그리고, 마지막 응용을 통하여 문제로 등장한 SDF에서의 실시간 응용을 위한 주기의 표현과 이를 위한 코드의 생성도 이 후의 연구로 남아있다.

참 고 문 헌

[1] S. Edwards, L. Lavagno, E. Lee, A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Model, Validation, and Synthesis.", *In Proceedings of the IEEE*, vol. 85, (no.3), March 1997. p.366-90.

[2] E. A. Lee and D. G. Messerchmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, Sept. 1987.

[3] G. Bilson, M. Engels, R. Lauwereins, "Cyclostatic dataflow," *IEEE Trans. on Signal Processing*, Vol. 44, No. 2, pp. 397-408, February. 1996.

[4] Wonyong Sung and Soonhoi Ha, "Hardware Software Cosimulation Backplane with Automatic Interface Generation" 1998 *Asia South Pacific Design Automation Conference*, Yokohama, Japan, February, 1998.

[5] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "Statemate: A working environment for the development of complex reactive systems", *IEEE Trans. on Software Engineering*, vol. 16, no. 4, Apr. 1990.

[6] J.Buck, S.Ha, E.Lee, and D.Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, Vol. 4, pp.155-182, 1994.

[7] D. Harel, "Statecharts: A visual formalism for complex systems", Sci. *Comput. Program.*, vol. 8, pp. 231-74, 1987.

[8] A. Girault, B. Lee, and E. A. Lee, ''Hierarchical

Finite State Machines with Multiple Concurrency Models," April 13, 1998 (revised from Memorandum UCB/ERL M97/57, *Electronics Research Laboratory, University of California, Berkeley*, CA 94720, August 1997).

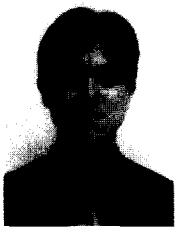
[9] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli, "Synthesis of software programs from CFSM specifications, in *Proc. of the Design Automation Conf.*, June 1995.

[10] M. von der Beeck, "A comparison of statecharts variants," *Proceedings of Formal Techniques in Real Time and Fault Tolerant Systems*, LNCS 863, pp. 128-148, Springer-Verlag, Berlin, 1994.

[11] C. Cassandras, "Discrete event systems, modeling and performance analysis," Irwin, Homewood IL, 1993.

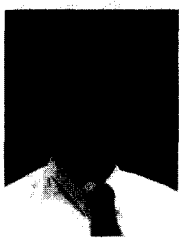
[12] D. Pan, "A Tutorial on MPEG/Audio Compression," *IEEE Multimedia*, Summer 1995, pp.60-74.

저 자 소 개



金 度 亨(正會員)

1997년 2월 서울대학교 컴퓨터공학과 졸업(공학사). 1999년 2월 서울대학교 대학원 컴퓨터공학과 석사과정 졸업(공학석사). 1999년 3월~현재 서울대학교 대학원 컴퓨터공학과 박사과정. 주요 관심 분야는 하드웨어/소프트웨어 codesign, 멀티미디어 시스템, Java



河 舜 會(正會員)

1985년 2월 서울대학교 전자공학과 졸업(공학사). 1987년 2월 서울대학교 대학원 전자공학과 석사과정 졸업(공학석사). 1992년 캘리포니아(Berkeley) 대학교 대학원 전기 및 컴퓨터공학과 박사과정 졸업(공학박사), 1993년~1994년 현대전자 근무. 1994년~현재 서울대학교 컴퓨터공학과 조교수. 주요 관심 분야는 하드웨어/소프트웨어 codesign, 신호처리용 설계 방법론, 병렬처리, 마이크로 프로세서 구조