임베디드 소프트웨어 개발을 위한 빠른 기능-시간 시뮬레이션 기법

김호근*, 정은진*, 박해우*, 하순회*
*서울대학교 전기컴퓨터공학부
e-mail : hkim@iris.snu.ac.kr

A Method for Fast Timed-functional Simulation of Embedded Software

Hokeun Kim*, EunJin Jeong*, Hae-woo Park*, Soonhoi Ha*
*Dept. of Electrical Engineering and Computer Science, Seoul National University

요 약

임베디드 시스템의 설계 복잡도가 높아지고, 설계 확인 및 수정 비용이 증가하면서 목표 아키텍처가 결정되기 전, 즉 시스템의 설계 초기 단계에서의 적절한 소프트웨어 검증이 더욱 더 중요해지고 있다. 이러한 초기 단계의 시뮬레이션은 높은 성능을 보이면서도 기능과 타이밍, 하드웨어 아키텍처, 태스크 할당, 그리고 스케줄링 정책 등을 잘 반영해야 한다. 특히 외부와의 시간 의존적인 상호 작용이 있는 경우 이를 반영하는 것은 어려운 문제인데 기존 연구[1]에서는 반복 시뮬레이션을 통해 이를 반영할 수 있도록 하였으나, 시뮬레이션 결과의 수렴 속도에 따라 총 시뮬레이션 시간이 증가한다는 약점이 있었다. 본 연구에서는 시스템의 명세 조건에 따라, 가능한 경우 반복 시뮬레이션을 피하고 단일 시뮬레이션만으로 시뮬레이션 결과를 얻음으로써 시뮬레이션 시간을 크게 단축할 수 있는 방법을 제시하였다. 연구의 결과는 반복 시뮬레이션 예제, 로봇 예제, 센서 네트워크 예제 등의 예제를 통해 검증되었으며 시뮬레이션 성능은 평균적으로 약 2.31 배 향상되었다.

1. 서론

최근 임베디드 시스템이 보편화되고, 이에 요구되는 기능이 다양화 됨에 따라 임베디드 시스템 디자인의 복잡도가 증가하고 있다. 또한 설계 단계가 진행됨에 따라 설계의 확인, 수정 비용이 급격히 증가하기 때문에 목표 아키텍처에 대한 시뮬레이션 모델이 정해지기 전의 초기 검증이 점점 중요해지고 있다. 응용 프로그램의 검증을 위해서는 기능적 검증 외에 시간적 검증 역시 병행되어야 하는데, 시간적인 정확도를 높게 유지하기 위해서는 긴 검증 시간을 요구하기 때문에 합리적인 정확도를 요구한 채 빠른 검증을 수행하는 방법에 대해서는 다각도에서 연구가 진행되고 있다.

Transaction level Model (TLM) 시뮬레이션은 주로 SystemC [2] 나 SpecC [3]와 같은 명세 언어를 사용하여 기술된 시스템을 검증하는 데 사용된다. TLM 은 모델링을 추상화한 정도에 따라 여러 모델로 나뉘는데, 그 중 기능-시간 (timed functional) 모델[4] 은 TLM 에 있는 여러 모델들 중 한 가지의 추상화 형태로 component-assembly [5] 모델이라고도 한다.

본 연구에서 제안하는 시뮬레이터는 시스템을 TLM 의 기능-시간 모델과 유사한 수준에서 추상화하여 검증을 수행하나 코드 내에 시간 정보를 삽입하는 대신 별도의 시간 정보를 기술하고 이를 토대로 시간 검증을 수행한다는 점에서 차이가 있다.

본 연구에서는 기존의 초기 검증용 범용 반복 기능-시간 시뮬레이터[1]를 개선한 새로운 기능-시간 시뮬레이션 기법을 제시하고자 한다. 여기에서 제시하는 기법은 기존의 반복 기능-시간 시뮬레이션과 동일하게 응용 프로그램의 동작 형태가 태스크 도식으로 주어지며, 태스크는 선점 불가능 하고, 각각의 프로세서에서의 태스크의 실행 시간은 이미 알고 있다고 가정하고 있다.

시뮬레이션 성능을 개선하기 위해 본 연구에서는 태스크의 실행 시 채널을 통한 입력을 센서 입력보다 먼저 받아들이도록 한다는 제약 조건을 추가하여 기존의 반복 기능-시간 시뮬레이션에서 이루어졌던 예비 시뮬레이션과 다수의 반복 시뮬레이션을 단 한 번의 시뮬레이션을 통해 완료할 수 있도록 하였다.

본 논문의 2 장에서는 관련 연구를 소개하고, 3 장에서는 기존의 반복 기능-시간 시뮬레이션을 간단히 소개한다. 4 장에서는 새롭게 제안되는 단일 실행 기능-시간 시뮬레이션 기법에 대해서 소개한 후에 5 장에서는 몇 가지 예제와 실험을 통해 단일 실행 기능-시간 시뮬레이션 기법의 유효성과 성능에 대해 기술하였다. 끝으로 6 장에서는 단일 실행 기능-시간 시뮬레이션 기법에 대한 결론을 맺는다.

2. 관련 연구

하드웨어-소프트웨어 통합 시뮬레이션은 임베디드 시스템의 성공적인 통합설계 개발을 위한 필수 과정이다. 가상 프로토타이핑 시스템[6]은 시뮬레이션을 통해 하드웨어 없이 소프트웨어를 개발하고, 목표 아키텍처의 성능, 전력, 비용 등의 설계 제약들을 만족하도록 하는 설계 공간 탐색에 주로 사용된다.

실제 시스템 구현에 들어가기 전 단계에서의 마지막 검증은 기능뿐만 아니라 통신 구조, 자원 경합에 따라 발생하는 시간제약도 고려해야 한다. 따라서 가상 프로토타이핑 시스템도 주로 실제 프로세싱 컴포넌트를 반영하는, 시간을 고려한 컴포넌트 시뮬레이터들로 구성되며, ISS (Instruction Set Simulator)는 전형적인 프로세서 시뮬레이터[7]의 예이다.

소프트웨어를 개발하는 경우 이러한 가상 프로토타이핑 시스템을 검증이 완료될 때까지 태스크 분할 등 여러 설정을 바꾸면서 반복하여 사용하게 된다. 소프트웨어 개발의 초기 단계에서는 기능적 검증을 주로 요하기 때문에 시간적 정확도보다는 속도와 유연성이 중요하게 되는데, 일반적인 ISS 는 이러한 목적으로 쓰기에는 너무 느리기 때문에 이진파일 변환(binary translation)이나 시간주석(time annotation) 같은 기법이 개발되었다[8][9][10].

[11]에서는 소스 코드의 라인별로 소요 시간에 관해 주석을 달아, SystemC 로 기능-시간 시뮬레이션을 수행하는 방법을 제안하였다. 가상 프로토타입 프로세서 위에서 소스 코드를 실행하면서 시간 정보를 얻어내기에 시간 정확도는 20% 수준으로 나온다[1]. 이 논문에서 제안하고 있는 기법에서도 태스크의 내부 실행 시간은 비슷한 방법으로 얻어낼 수 있다.

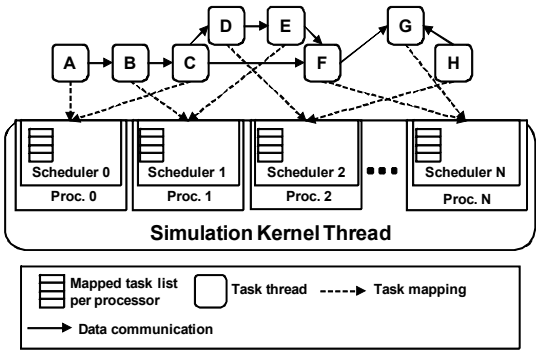
[12]에서는 MPSoC 에 대한 기능-시간 시뮬레이션을 하는 데 있어 SystemC 시뮬레이터를 사용하였다. [12]에서는 가상 프로토타입 프로세서를 사용하지 않고, 컴파일러 기법을 통해 태스크의 실행 시간을 추정하고, wait()문으로 소요 시간을 모델링 하였다. 각 태스크의 시간 정보는 wrapper 모듈을 통해 SystemC 시뮬레이터로 전달되도록 하였는데, 여기서 wrapper 모듈은 상위 수준의 가상 플랫폼이라고 지칭되며, 프로세서의 운영체제 동작을 모델 하도록 하였다.

3. 기존의 기능-시간 시뮬레이션

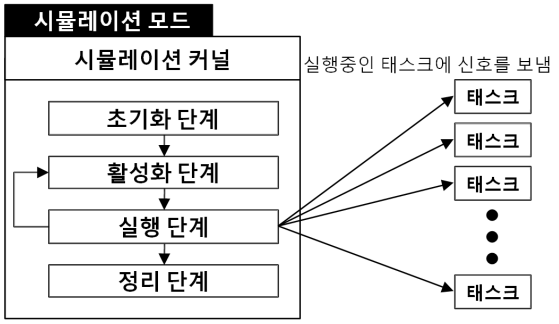
3.1. 기능-시간 시뮬레이터 구조

본 논문에서 제안하는 시뮬레이터는 시뮬레이션 커널 역할을 하는 한 개의 커널 스레드와 태스크 모델에서의 각각의 태스크를 나타내는 태스크 스레드로 그림 1 과 같이 구성된다. 태스크 스레드는 태스크의 실제 수행에 해당하는 함수를 포함하고 있다.

시뮬레이션 커널은 그림 1 과 같이 여러 가지 프로세서 모델들을 포함하고 있다. 각각의 프로세서는 할당된 태스크의 관리 목록을 가지고 있다. 시뮬레이션 커널이 전역 시간을 관리하는 반면, 프로세서들은 각각의 지역 시간을 관리한다. 그림 2 는 시뮬레이션 커널의 작업 흐름을 나타낸다. 초기화 단계에서는 태스크와 프로세서의 자료구조를 초기화하고, 활성화 단계에서는 프로세서별로 수행 가능한 태스크를 찾는다. 실행 단계에서는 프로세서 별로 태스크를 하나 실행한다. 모든 태스크의 실행이 완료되면 정리 단계를 수행한다.



(그림 1) 시뮬레이션 구조([1])



(그림 2) 시뮬레이션 커널의 수행 단계([1])

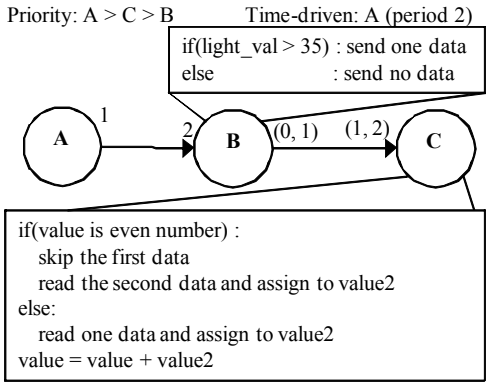
3.2. 반복 기능-시간 시뮬레이션 기법

일반적으로는 태스크가 필요로 하는 데이터의 개수를 미리 알 수 없다. 따라서, 기존 방법에서는 태스크가 필요로 하는 데이터의 개수를 알기 위하여 예비 시뮬레이션을 수행하였다. 태스크가 필요로 하는 데이터가 채널에 없는 경우 태스크는 블로킹이 된다. 블로킹 이전의 수행시간과 블로킹 이후의 수행시간을 구별하지 않기 때문에 예비 시뮬레이션에서는 시간 시뮬레이션을 정확히 수행하지 못한다. 대신 각 태스크가 필요로 하는 데이터의 개수를 기록한다.

예비 시뮬레이션 이후에, 시간 시뮬레이션이 수행되는데, 이 때에는 각 태스크가 필요로 하는 데이터가 채널이 존재하는 경우에만 태스크를 활성화시킨다. 따라서 한번 활성화된 태스크는 블로킹 없이 수행이 되고 태스크의 수행시간을 이용하여 시간 시뮬레이션을 수행할 수 있다. 그러나 여기에서 문제는 응용 프로그램의 동작은 비동기적으로 일어나는 외부 이벤트에 의해 달라질 수도 있다는 점이다. 여기서 이러한 문제를 “외부 이벤트 문제”라고 부른다. 그림 3 의 반복 시뮬레이션 예제는 그러한 예 중 하나이다.

그림 3 의 태스크 B 는 호출되는 시점에 따라 읽는 센서의 값이 달라진다. 따라서, 외부 이벤트에 의하여 태스크 수행이 바뀌면 태스크가 필요로 하는 데이터의 개수가 변할 수 있으므로 예비 시뮬레이션과 같이 블로킹 현상이 생길 수 있다. 이 경우, 필요한 데이터

의 양이 바뀐 것을 다시 기록하고, 시간 시뮬레이션을 다시 반복하여 실행한다. 이렇게 반복이 필요한 횟수는 외부 이벤트에 의하여 태스크의 동작이 바뀌는 횟수에 따라 결정이 된다. 만약 태스크가 필요로 하는 데이터의 수가 변화가 없다면 시간 시뮬레이션을 한번만 수행하면 되지만, 그렇지 않다면 최악의 경우, 외부 이벤트의 횟수만큼 반복을 해야 한다.



(그림 3) 반복 시뮬레이션을 요하는 예제 태스크 정보([1])

4. 제안하는 단일 실행 기능-시간 시뮬레이션

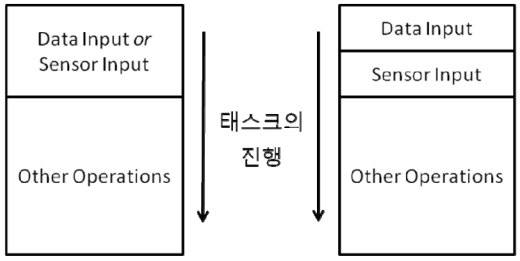
기존의 반복 기능-시간 시뮬레이션은 외부 이벤트의 영향으로 인해 시뮬레이션 수렴 속도가 달라지며, 이로 인해 시뮬레이션 시간이 크게 증가할 수 있다는 약점을 가지고 있다. 이에 본 연구에서는 응용 프로그램의 명세에 약간의 제약 조건을 추가하여 수렴할 때까지 수행하던 시뮬레이션 반복을 제거하고 단일 횟수 시뮬레이션만으로 기능-시간 결과를 얻는 방법을 개발하였다. 또한 추가된 조건은 대다수의 응용 명세에서는 개발자의 의도를 크게 제약하지 않는다.

기본적인 동작 방법은 시뮬레이션이 예비 시뮬레이션의 활성화 방법과 마찬가지로 채널에 데이터가 하나라도 있는 경우, 태스크를 활성화시키게 된다. 예비 시뮬레이션에서와 마찬가지로 블로킹이 발생하게 되는데, 이 때의 처리가 기존 연구와 구분된다. 즉, 예비 시뮬레이션의 경우 필요한 데이터를 파일로 기록하는 반면, 단일 실행 시뮬레이션에서는 현재 접근하려는 채널과 읽으려는 데이터 양을 태스크마다 추가로 만든 구조체에 저장하게 된다. 또한, 예비 시뮬레이션이 시간을 바로 증가시키는 것과 달리, 태스크가 스케줄 되지 않았다고 본다. 태스크 스레드의 상태는 블로킹, 유휴, 실행 상태 3 가지로 나뉘어져 있는데, 블로킹으로 분류된 태스크는 시뮬레이션 커널 동작의 4 단계 중 2 번째 단계인 활성화 단계에서 블로킹이 된 태스크에 한해 태스크의 구조체에 기록한 채널과 크기 정보를 확인하여 재활성화를 해주게 된다.

단일 실행 시뮬레이션의 구현상 특이점은 태스크 활성화와 블로킹 이후의 재활성화가 같은 명령을 통해 이루어지게 함으로써 블로킹된 태스크를 마치 새로 실행하는 것과 같이 재활성화 명령을 내릴 수 있다는 것이다. 즉, 3 번째 단계인 행동 단계에서 태스크 스케줄러의 결정에 따라 블로킹된 태스크를 실행해주는데, 만약 태스크가 블로킹 된 경우에는 프로세서가 스케줄러를 통해 활성화된 다른 태스크를 실행시킬 수 있다. 태스크가 정상적으로 종료가 된 이후에나 프로세서의 지역 시간을 증가시키게 되고, 이 때 함께 스케줄 결과도 기록한다.

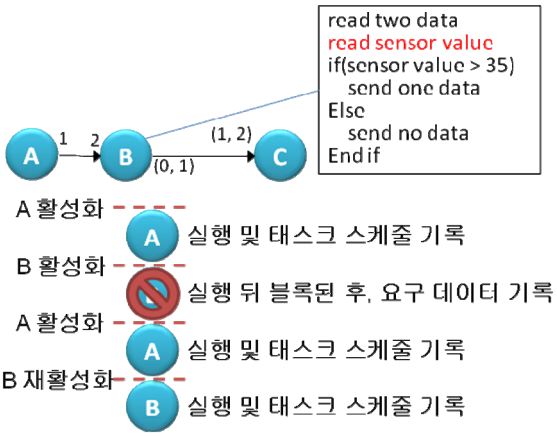
이렇게 수정된 방법을 사용하기 위해서는 한가지 추가적인 조치가 필요하다. 그것은 외부 이벤트인 센서의 입력과 데이터의 입력의 순서를 조정하는 일이다. 센서를 먼저 읽게 되면, 데이터의 블로킹 때문에 태스크의 수행이 지연되는 경우 잘못된 센서의 값을 읽을 수가 있다. 따라서 외부 이벤트에 의하여 태스크의 수행이 바뀌지 않는 것을 보장하기 위해서는 센서의 값을 데이터보다 나중에 읽어야 한다.

그림 4 는 이전 시뮬레이션의 입력 순서 가정과 제안하는 시뮬레이션의 입력순서 가정의 차이를 보여주는 그림이다. 이러한 제약을 추가함으로써 제안한 기법과 다른 시뮬레이션을 추가로 개발하였다.



(그림 4) 기존 연구[1]의 태스크 모델과 제안하는 기능-시간 시뮬레이션의 태스크 모델 가정의 차이

그림 5 는 앞의 그림 3 의 예제를 제안하는 시뮬레이션에 맞게 수정하고 그 시뮬레이션 과정을 간략히 나타낸 것이다.



(그림 5) 제안하는 기능-시간 시뮬레이션에 맞추어 수정된 기존 반복 시뮬레이션 예제

단일 실행 기능-시간 시뮬레이션은 입력 포트를 센서 입력보다 먼저 오게 하도록 제약을 가하면서 여러 번의 시뮬레이션을 하는 대신 한 번의 시뮬레이션만으로 스케줄 결과를 얻을 수 있다는 점에서 기존 기능-시간 시뮬레이션에 비해 성능상 이점이 있다.

5. 예제 및 실험

이 장에서는 기존 기능 시뮬레이션 및 반복 기능-시간 시뮬레이션 기법과 제안하는 단일 실행 기능-시간 시뮬레이션 기법의 오버헤드 및 성능을 예제를 통하여 비교한다. 실험은 3GHz 클럭 주파수를 가지는 듀얼 코어 Intel Xeon 5120 프로세서 두 개와 4GB 메모리, 그리고 SMP Linux 2.6.26 버전의 운영체제를 갖춘 플랫폼에서 수행되었다.

5.1. MPEG4 예제

이 절에서는 복잡한 MPEG4 simple profile 디코더 예제를 이용하여 기능 시뮬레이션에 비해 기능-시간 시뮬레이션이 가지는 오버헤드를 측정한다. 예제는 총 29 개의 태스크와 90 개의 채널로 이루어진 복잡한 멀티미디어 예제로 대상 플랫폼은 5 개의 프로세서를 포함한다고 설정하였다. 예제를 돌리는 총 시뮬레이션 시간은 40,000,000 us 으로 설정하였고, 측정 소요 시간은 시뮬레이션을 각 5 회 수행하여 얻은 소요 시간의 평균으로 하였다. 결과는 표 1 에 기록하였다.

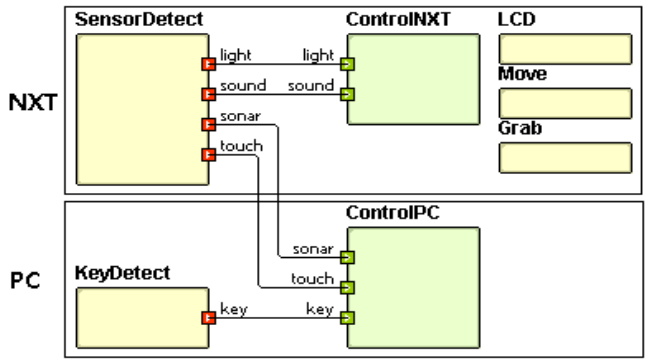
시뮬레이션 종류		실행시간(초)
기능		11.952
반복 기능-시간	예비	17.018
	반복	31.533
	합계	48.551
단일 실행 기능-시간		23.826

(표 1) MPEG4 디코더 예제 시뮬레이션 종류별 실행시간

실험 결과 제안하는 시뮬레이션은 단순 기능 시뮬레이션에 비해 50%정도의 성능을 가지는 것으로 나타났다. 본 예제는 센서 입력이 없는 예제로서 기존의 반복 기능-시간 시뮬레이션에서는 반복을 총 1 회 하였고, 제안하는 시뮬레이션에서는 이보다 약 2.04 배 향상된 성능을 보여주었다.

5.2. NXT 로봇 예제

이 절에서는 LEGO MINDSTORM NXT[13] 로봇 응용 프로그램을 사용하여, 제안된 기능-시간 시뮬레이션 기법의 효과를 검증하였다. 그림 6 은 본 예제의 태스크 도식이다.



(그림 6) NXT 로봇 예제의 태스크 도식

SensorDetect 와 KeyDetect 는 각각 입력 파일로부터 센서와 키보드 입력을 읽는 태스크이고, ControlNXT 와 ControlPC 는 데이터 입력을 받아 LCD, Move, Grab 등의 태스크를 제어하는 태스크이다. 태스크 가장자리의 작은 사각형들은 포트를 나타내며 실선들은 각 포트를 연결하는 채널을 나타낸다. 태스크들 외곽의 NXT 와 PC 는 태스크의 수행 플랫폼을 표현하고 있다. 표 2 는 이 예제를 사용하여 2,000,000,000 us 의 시뮬레이션을 각 5 회 수행하여 소요 시간의 평균을 기록한 것이다.

시뮬레이션 종류		실행시간(초)
반복 기능-시간	예비	1.168
	반복 1	1.306
	반복 2	1.374
	합계	3.848
단일 실행 기능-시간		1.499

(표 2) NXT 로봇 예제 시뮬레이션 종류별 실행시간

본 예제는 센서 입력이 존재하는 예제로서 실험 결과 기존의 시뮬레이션은 반복을 총 2 회 하였고, 제안하는 시뮬레이션은 이에 비해 약 2.57 배 빠르게 수행되었음을 알 수 있다.

6. 결론

본 논문에서는 기존의 기능-시간 시뮬레이션 기법보다 빠르고 효율적인 새로운 기능-시간 시뮬레이션 기법을 제시하였다. 예제를 이용한 실험에서 본 기법은 임베디드 소프트웨어의 기능-시간 시뮬레이션에 대해 기존의 기법보다 평균적으로 약 2.31 배 향상된 성능을 나타내었다.

참고문헌

[1] EunJin Jeong, Hae-woo Park, Soonhoi Ha, Hyunok Oh, “Task-level Timed-functional Simulation for Multi-core Embedded Systems”, ESTIMedia'10, 2010 (게재 예정)
[2] OSCI, Open SystemC Initiative http://www.systemc.org.
[3] Daniel Gajski *et al.*, "SpecC: specification language and methodology," Kluwer Academic Publishers, 2000
[4] Thorsten Grotker *et al.*, "System design with SystemC." Kluwer Academic Publishers, 2002
[5] Lukai Cai and Daniel Gajski. "Transaction level modeling: an overview." CODES+ISSS '03, 19-24, 2003
[6] Randy Young *et al.*, "Virtual Prototyping System," Defense Technical Information Center, 2004
[7] CoWare Inc. CoWare Virtual Platform. http://coware.com/PDF/products/VirtualPlatform.pdf
[8] CoWare Inc. CoWare Processor Designer. http://coware.com/PDF/products/ProcessorDesigner.pdf
[9] CoMET, VaST, http://www.vastsystems.com
[10] Marius Gligor *et al.*, "Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation," CODES+ISSS'09, 71-80, 2009
[11] Trevor Meyerowitz et al., "Source-level timing annotation and simulation for a heterogeneous multiprocessor." DATE '08, 276-279, 2008
[12] Jianjiang Ceng *et al.*, "A high-level virtual platform for early MPSoC software development". CODES+ISSS '09, 11-20, 2009
[13] LEGO® MINDSTORMS® NXT http://mindstorms.lego.com