SychCharts를 이용한 임베디드 시스템을 위한 RTOS Scheduling 검증

*이수영[O] 안영아 김진현 류갑상 최진영
고려대학교 컴퓨터학과
{sylee[O], ellaahn, jhkim, gsryu , choi} @formal.korea.ac.kr

RTOS Scheduling Verfication for Embedded system by SyncCharts

SuYoung Lee[O] YoungAh Ahn, JinHyun Kim, GabSang Ryu, JinYoung Choi
Dept. of Computer Science Korea University

요 약

Mission-Critical한 실시간 반응형 내장 시스템들의 설계과정에 있어 high-level abstraction과 formal(software 기반) modeling은 없어서는 안될 중요한 부분이다. 실시간 반응형 내장 system의 OS는 정형 명세 기법을 이용하여 시스템의 주요 component들을 설계하고 OS의 Formal model들을 모든 가능한 input들 아래 OS의 behavior를 엄격하게 검증함으로써 error가 없는 완벽한 OS를 개발할 수 있다. 본 논문에서는uC/OS-II의 OS Scheduling 부분을 반응형 시스템 언어인 Esterel의 SyncCharts로 명세, 명세한 시스템의 요구조건을 정형기법을 이용해서 검증해보고자 한다.

1. 서 론

근래에 실시간 반응형 내장 system들의 사용이 점차 늘어나고 있다. 그런 시스템들이 mission-critical하다면 high-level abstraction과 formal modeling은 내장형 system 설계과정에 있어 없어서는 안될 부분이다. 그러한 실시간 반응형 내장 system의 OS를 정형 명세 기법을 이용하여 시스템의 주요 component들을 설계하고 OS Formal model들을 모든 가능한 input들 아래 OS의 behavior를 엄격하게 검증함으로써 error가 없는 완벽한 OS를 개발할 수 있다. 게다가 대부분 그런 model들은 deterministic하고 검증과정이 자동적으로 이루어진다. OS formal model의 형태는 "명확하고 수학적인 명세"[1]를 가져야 한다. 이것은 보증된 도구들을 이용하여 모델들을 검증하는 것을 가능하게 한다. 그러한 수학적으로 정확한 명세를 가능하게 하기 위해 동기적 정형 검증 언어인 Esterel SyncCharts를 이용한다. SyncCharts로 명세한 모델들은 완벽하게 deterministic하고 수학적으로 명확함을 보장한다.[3]

본 논문에서는 uC/OS-II의 OS Scheduling 부분을 정형 명세 언어인 Esterel SyncCharts를 이용하여 명세하고 SyncCharts의 검증기법을 이용하여 모델 체킹을 함으로 사용자의 요구를 시스템이 이룰 수 있는 어떤 상태에서도 충족시키는가를 실험할 것이다. 2장은 정형 명세 언어인 Esterel SyncCharts에 대해 설명하고, 3장에서는 OS Scheduling 모듈을 SyncCharts로 명세 및 검증 4장에서 결론 및 향후 과제를 제시한다.

2. SyncCharts 소개

SyncCharts는 Reactive 시스템의 동기적 프로그램을 작성하는데 사용되는 동기적 언어로 Esterel technology 사에서 만든 graphical한 언어이다.[2] 동기적 프로그램 언어는 완벽한 동기적 동시성 모델에 근거하며, 이러한 모델 내에서는 동시적 프로세스가 0시간 내에 계산과 정보의 교환을 일으킨다. SyncCharts는 일종의 결정적 reactive 시스템을 프로그래밍하는 정형적 의미를 지닌 언어로써 동시적인 입력 set을 기다리고 출력을 계산하며 생산하는 것으로 입력에 대한 반응을 하고 정지한 다음, 새로운 입력을 기다린다. SyncCharts는 동기적 가설을 기반으로 하기 때문에 입력 set에 대한 모든 반응은 시간을 소모하지 않고 instantaneous한 것으로 간주된다. SyncCharts의 프로그램 모델은 컴포넌트 혹은 모듈의 명세와 이를 평행하게 실행하는 것으로 이루어져 있다. 모듈들은 시그널을 통하여 각각의 모듈끼리 혹은 외부 세계와 통신을 한다. 이러한 시그널은 임의의 타입의 값을 지닐 수 있으며 전체 모듈에게 broadcasting된다. 이러한 동기적 가설을 충족시키도록 시그널의 출력과 입력은 시간적 소모가 없는 것으로 간주된다. SyncCharts는 모든 반응에 대한 입력은 전체적으로 그 반응에 대한 출력을 결정하며 나머지 프로그램의 입-출력 행위 또한 결정한다. 동기적 가설을 충족시키도록 통신과 선점은 결정성을 유지한다. 그에 더하여 모든 내부 통신은 컴파일되고 시그널 결정적 유한 상태 기계가 컴파일러에 의해 만들어진다. SyncCharts내의 동시성은 프로그램의 편리를 위해 구조적 도구이며 런타임 오버헤드를 일으키지 않는다.

3. OS Scheduling 모델 명세 및 검증

3.1 OS Scheduling의 SyncCharts 명세

본 논문에서 다루는 실시간 반응형 내장 시스템은 태스크 관리와 스케줄러 기능을 가진 간단한 RTOS를 내장한 시스템으로 정형 명세 및 검증의 대상은 이 RTOS만으로 한정한다.

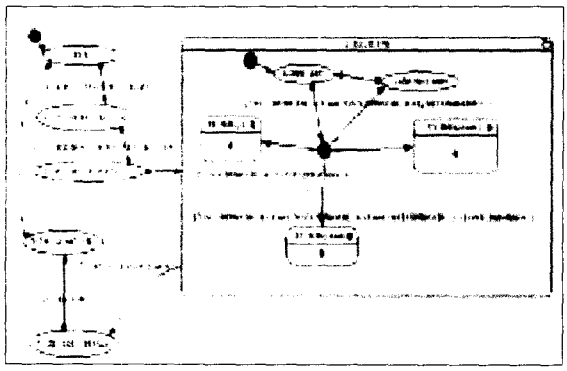

그림 1 OS Scheduling module

RTOS의 전체 구성 요소는 3개의 task들과 스케줄러로 구성되며 스케줄러 알고리즘은 우선순위 스케줄링 기법을 이용하여 각 task 마다 우선순위를 지정하여 높은 우선순위의 task를 먼저 실행하도록 스케줄링한다. 먼저 SyncCharts를 이용하여 task, 스케줄러 등과 같은 RTOS 내부의 주요 컴포넌트들을 설계한다. SyncCharts로 명세한 우선순위 스케줄링 알고리즘은 그림 1과 같다. 이 스케줄러는 ready 상태에 있는 task 가운데 우선순위가 가장 높은 task를 선택하여 스케줄링한다. 그림 1에서 이러한 스케줄러가 스케줄링하는 idle task, periodic task, sporadic task들은 각각 run sub module로 만들어 간단하게 명세하였다.

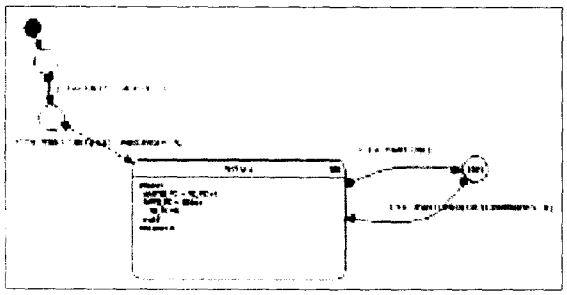

그림 2 TASK 0

본 논문에서 명세할 운영체제가 운영하는 시스템은 idle task, periodic task, sporadic task 이상 3개가 운영된다. idle task는 다른 태스크 모두가 dormant 상태일 때, 수행을 시작하여 다른 태스크가 스케줄링 되기 전까지 스케줄링되는 가장 우선순위가 낮은 태스크이다. 그림 2의 TASK 0과 같이 다른 태스크들이 스케줄링 되기 전과 수행이 끝난 후에 더 이상 스케줄링 될 태스크들이 reday 상태에 있지 않을 때 스케줄링되는 태스크로 idle task는 tick을 발생하면서 기다리다 Interrupt가 들어오면 interrupt 서비스 루틴으로 상태전이하게 명세하였다.

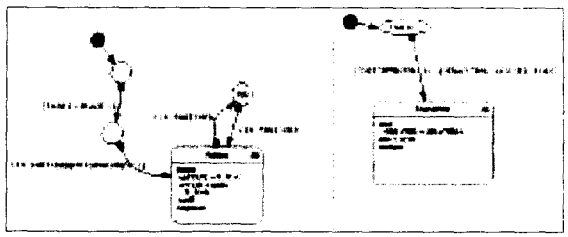

그림 3 TASK 1

Periodic task는 다양한 인스턴스 혹은 반복을 주기적으로 수행하는 task로 같은 task의 이어지는 수행은 반드시 고정된 주기를 포함해야 한다. Periodic task를 명세한 그림 3을 보면 Timedely 모듈에서 정의한 tick을 주기로 task가 수행되게 명세하였다.

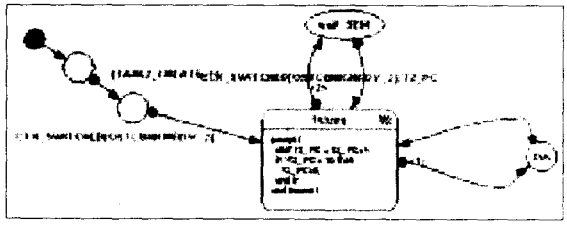

그림 4 TASK 2

sporadic task는 0 또는 그 이상의 인스턴스를 갖는 task로 같은 태스크의 이어지는 수행은 최소한 구분간격이 존재해야 한다. 그림 4와 같이 sporadic task는 wait semaphore 모듈에서 발생하는 P,V 신호에 따라 task를 수행하도록 명세하였다.

OS 스케줄링 모듈은 uC/OS-II의 OS 스케줄러 소스를 참고로 초기화 상태에서 OS 스케줄러가 시작되면 interrupt를 disable시켜 스케줄링이 되는 동안 어떤 interrupt도 발생하지 못하도록 lock을 걸도록 명세하였다. 또한 uC/OS-II는 interrupt가 중첩적으로 발생가능한 OS로 interrupt nesting check와 Lock nesting check를 한 후 우선순위 알고리즘에 따라 task들을 스케줄링한다. SyncCharts에서는 Valued signal을 이용하여 각 task들이 갖는 priority 값을 0,1,2로 표현하여 signal들의 방출 유무에 따라 스케줄링을 명세하였다.

3.2 OS Scheduling의 정형 검증

명세된 OS Scheduling 모듈은 시뮬레이션과 정형검증의 두 가지 방법을 사용하는데 시뮬레이션은 시스템에 어떤 입력 값을 주고 그 입력 값에 따른 상태 다이어그램의 상태 변화와 출력되는 시그널을 통해서 결과가 처음에 의도한대로 올바르게 나오는지를 확인해 보는 방법으로 Esterel SyncCharts의 시뮬레이션 도구를 이용하여 validation을 시행한다. 정형검증은 모델체킹[3]기법으

로 명세한 시스템을 시스템 사용시 발생 가능한 모든 상태를 표현하는 유한 상태 기계(Finite state machine)로 바꾼다. 이 유한 상태 기계는 latch를 지닌 Boolean equation으로 표현되는 함축적 유한 상태 기계이다. SyncCharts의 컴파일러는 이 유한 상태 기계로 변환된 모델을 검증하기 위해 우선 유한 상태 기계를 돌아다니며 구별하기 힘든 관계에 있는 상태들을 연결시키는 동가 관계 기법과 상태 폭발을 막기 위한 상태 감소 기법인 BDD(Binary Decision Diagram)를 사용하여 유한 상태 기계의 크기를 최소화 시킨다. 그 후에 최소화된 유한 상태 기계의 모든 상태를 찾아다니며 출력 시그널의 상태를 확인하는 상태 탐색 기법을 사용한다. 즉 특정 조건 하에서 특정 신호가 발생하는가를 관찰함으로 시스템이 따라야 하는 조건의 만족여부를 검증할 수 있다. 이 요구 조건들은 만족해야 할 시제 논리로 작성한 Observer를 통해 이를 위반하거나 만족하는 상황을 감지할 수 있다.

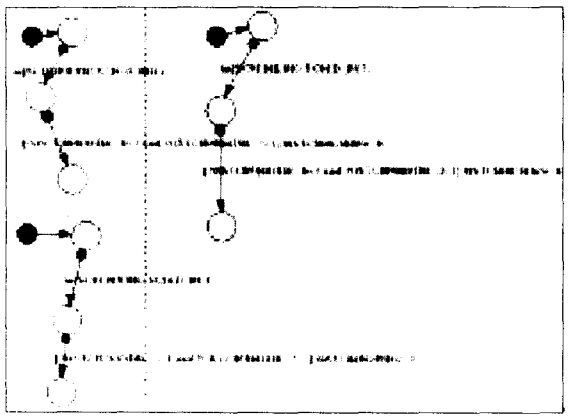

그림 5 Observer

설계한 시스템이 만족해야 할 요구 조건은 어떤 상황에서도 3개의 task들 중 가장 높은 우선순위를 가진 하나의 task만이 스케줄링 되어야 한다. 주어진 요구 조건은 그림 5와 같이 작성된 obsever를 통하여 검증하였다. 3개의 모듈로 구성된 observer는 입력 시그널에 따른 출력 시그널을 항상 관찰하면서 요구조건이 만족 되는 경우 또는 위반되는 경우를 감지한다. 요구조건을 만족하는 경우에는 시스템이 본래의 의도대로 제대로 설계되었다는 것을 의미한다. 그림 5의 observer는 모든 상황에서 task0, task1, task2가 갖는 priority를 비교하여 우선 순위가 높은 task 하나만을 스케줄링 하는지를 검증하도록 명세하였다.

4. 결론 및 향후 연구 과제

mission-critical 시스템의 내장형 소프트웨어의 구현은 시스템이 복잡해짐에 따라 높은 신뢰성을 검증하는 여러가지 방법이 개발 적용되고 있다. 하지만 대부분의 테스트 케이스를 추출하여 검사하는 기법이 대부분으로 그러한 테스팅 기법은 테스트 케이스의 한계로 인해 오류를 포함할 가능성이 있다. 따라서 본 논문에서는 정형 명세 언어인 Esterel SyncCharts를 이용하여 구현함으로써 설계시의 오류를 없애고 Esterel SyncCharts의 정형 검증 기법을 통해 설계 명세를 검증하였다.

향후 연구과제로는 Esterel SyncCharts로 명세 구현한 OS 모듈을 SyncCharts가 아닌 textual 언어인 Esterel로 더 자세히 명세 구현하여 검증, 자동 생성된 code를 실제 임베디드 시스템 타겟 플랫폼에 이식할 수 있는 방법을 제시할 계획이다.

5. 참고 문헌

[1] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. In Proceedings of the IEEE, 1997

[2] Charles Andre, "SyncChart: A Visual Representation of Reactive Behavior", Technical report RR-95-52, 13S, 1995

[3] Micheal Gunzert. Building Safety-Critical Real-Time Systems with Synchronous Software Components. Technical Report, Institute of Industrial Automation and Software Engineering(IAS), 1999

[4] Edmund M. Clake, Orna Grumberg, Doron A. Peled. Model Checking. The MIT press. 1999