Embedded System Design을 위한 Real-Time System의 최적화된 Timing효과의 구현

박은정* · 정태경*

Timing Optimization of Real-Time System Design for Embedded Systems

Eunjung Park · Taikyeong Jeong

요 약

본 고에서는 임베디드 시스템을 위한 새로운 이벤트 중심의 실시간 시스템의 디자인의 최적화에 대한 연구를 논하고자 한다. 이를 위해서는 대단위의 분산처리시스템을 만족하기위한 종료시한 초과 처리기의 구현이 필수적이며, 이때 실시간 시스템에서의 타이밍을 조절하고 계측할 수 있는 종료시한이 사용되어야 한다. 이러한 타이밍의 요소들은 보통 모델과 실제 시스템사이의 인터페이스에서 동작하는데, 여기서 실시간 시스템의 디자인을 위한 다양한 그래픽 언어들이 종료시한을 효과적으로 처리하도록 제고한다. 이 연구를 통하여 사용자 수준에서의 임베디드 시스템을 위한 종료시한 초과 처리와 함께, RoseRT라는 툴을 이용한 최적화된 종료시한 초과 처리기를 증명해 보이고자한다.

ABSTRACT

This paper presents a new real-time system design methodology for embedded system as well as event-driven real time application. It is required to implement a deadline handling mechanism in order to satisfy a large-scale distributed real time application. When we design real time system, it has handled a deadline and is important to measure / control a timing issue. These timing constraints usually associated with an interface between model and system. There are many case tools that supporting a real time application, for example, UML, graphic language for designing real time system, but they cannot provide efficient way to handle deadline miss. Therefore, users have to design deadline handler manually when they need to use it. This paper contributes solving the problems of user-level deadline handling for an embedded system. Also, it also discusses an efficient deadline handler design mechanism using on RoseRT, which is a graphical CASE tool supporting from UML.

키워드

Real-time system, Deadline miss handler, Embedded system

* To whom correspondence should be addressed at the University of Delaware, Newark, DE 19716 USA;

접수일자 : 2005. 10. 19

Ⅰ. 서 론

분산되어지고 병렬 처리되어 계산되는 시스템이 무선 통신과 네트워크분야에서도 확대되어 일어남과 동시에, 실제 시스템의 개발과 함께 요구사항도 복잡해지면서 근래의 Real-time system을 디자인하고 개발하는 일은 점점 어려워지고 있다. 이러한 Real-time system의 설계가 최근 Embedded system에 적용되어 새로운 분야로 자리 잡고 있는데 이를테면, Intel사의 XScale Architecture 또는 CC-NUMA Multiprocessor가 지니고 있는 장점처럼 다수의 유저가 원격으로 접속하는 OS또는 컴파일러가 임베디드 되는 경우를 쉽게 찾아 볼 수가 있다 [1, 2]. 이를 기반으로 실시간으로 진행되는 임베디드 시스템의 구현이 가능하여지는데, 이러한 시스템을 디자인하는 단계에서 Timing issue는 결코 간과하지 못할 요소이다. 이를 위해 Real-time system의 중요한 timing constraints 중의 하나인 deadline (종료시한)을 처리하기 위한 기법을 살펴보고, 이를 임베디드 시스템에 적용함에 앞서 최적화 할 필요가 있다. 왜냐하면 Real-time system의 디자인에서 강조되던 miss rate을 줄이고, 시스템의 성능을 향상시키기 위한 새로운 알고리즘이 개발되지 않는 이상, 현재의 종료시한 초과 처리 기법이 임베디드 시스템에 적용되어 시스템의 성능을 향상시키기 힘들기 때문이다.

현재 Real-time system을 디자인하기 위한 다양한 툴이 존재하지만, 종료시한 초과 처리에 대한 일련의 과정과 명확한 기능을 규명하지 않고 있기 때문에, 사용자가 직접 디자인 단계에서 deadline miss handler(종료시한 초과 처리기)를 기술하기 위해 디자인 툴에서 제공하는 timing 기능을 사용해야 한다. 여기서 timing 기법이 최적화되어서 embedded될 경우 시스템의 성능을 높일 수 있음은 물론이고 사용자에게 편리함을 제공할 수 있기에, 이는 새로운 성과가 될 수 있다.

따라서 이 연구를 통하여 Real-time system이 가진 중요한 요소 중 하나인 timing 기능의 효과적인 구현을 살펴보고, 이와 관련되어 예상되는 문제점을 분석하여 적절한 종료시한 초과 처리의 방법론을 제안하고자 한다.

Ⅱ. 연구 동향

시스템 및 어플리케이션을 디자인하고, 시각화, 구조화, 문서화하며 수행을 가능하게하기 위해 제작된 모델링 언어로서 UML (Unified Modeling Language)이 상용되고 있다 [3]. 더욱이, UML-RT (UML-Real Time)는 Real-time system을 디자인하기에 적합한 모델링 언어이며, 실시간으로 진행되는 임베디드 시스템과 분산화된 시스템에 적용하기에 알맞다.

UML-RT에서 특정 작업을 수행할 수 있는 가장 기본적인 개체는 캡슐이라고 하는 개념이며, 이 캡슐 기반의 이벤트 처리를 할 때에는, 캡슐 인스턴스 하나에 해당하는 이벤트가 하나의 쓰레드에서 처리가 된다. 더 나아가서 캡슐 기반의 이벤트 처리 대신, "*시나리오*" 기반의 이벤트 처리를 할 때에는 하나의 시나리오에 해당하는 이벤트가 하나의 쓰레드에 각각 매핑된다 [3].

여기서 시나리오란 하나의 외부 메시지에 의해 시작되어 최종 output 메시지가 나오기까지의 일련의 과정이나 흐름을 말한다. 그러므로 캡슐 기반 멀티쓰레딩에서는 하나의 캡슐에 들어오는 여러 메시지들이 다른 쓰레드에서 처리되는 것이 불가능한 반면, "*시나리오 기반 다중 쓰레딩*" (Scenario-based Multithreading) 에서는 각각의 메시지가 아니라 완전한 메시지 Chain에 대해 쓰레드로 매핑하고 우선순위를 매기는 것이 가능하다 [5].

그림 1은 실제 시나리오 기반 멀티 쓰레딩을 UML-RT의 한 종류인 RoseRT라는 CASE 툴을 동작하도록 만든 내용이다. 이는 기존의 객체 지향 모델링 (Object-oriented modeling)을 real-time으로 확장한 것이다 [4]. 이때 우리가 시스템을 디자인하여 모델을 만들면, RoseRT에서 그 모델을 수행시키기 위한 코드를 생성한다. 그 생성된 코드로부터 Analyser라는 툴이 시나리오의 정보를 뽑아낸다 [3]. 이 정보가 담기는 파일이 시나리오 정보 파일(Scenario description file or Analysis file)이다. 이 파일에는 시나리오를 수행하기 위한 정보가 담기게 되고, Modifier라는 툴이 새로운 시나리오 기반의 정보를 적용하여 소스코드를 변

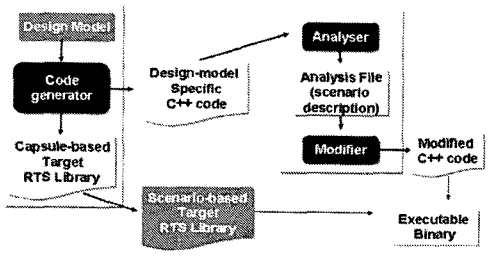

그림 1. A scenario-based Multithreading

경하여 수행하도록 되어있다 [3].

이러한 시나리오 기반 멀티 쓰레딩에서는 두 수준 스케줄링과 쓰레드 간 메시지 발송에 의한 blocking이 존재하지 않으며, 완전 수행 조건에 의한 blocking도 단 한번으로 한정 지을 수 있기에 기존의 캡슐 기반 멀티 쓰레딩보다 좋은 성능을 기대할 수 있다[5]. 본 논문에서는 이러한 시나리오 기반 멀티 쓰레딩을 기반으로 종료시한 초과 처리를 위한 메커니즘들을 제안하고, 이를 임베디드 시스템에 적용하고자한다.

우선 태스크(task)에 대한 정의가 필요하다. UML-RT에서는 태스크에 대한 정의가 없기 때문에 하나의 캡슐을 태스크로 볼 수도 있으나, 시나리오 기반 멀티 쓰레딩을 사용할 경우, 연속된 캡슐들로 이루어진 하나의 시나리오를 태스크로 볼 수도 있다. 본 연구에서는 태스크에 대한 정의를 시나리오로 정의하였다. 또한 UML-RT에서 종료시한이라는 timing constraints를 지원하지 않아서, 사용자가 직접 종료시한 초과에 대해 정의해야한다 [8]. 이에 대한 정의는 태스크의 정의에 따라 달라질 수 있다. 만약 캡슐을 태스크로 본다면 캡슐 안에서의 상태전이를 기준으로, 하나의 메시지로부터 상태전이가 제 시간 안에 이루어졌는지를 통해 종료시한 초과 여부를 판단한다. 그러나 본 연구에서와 같이 시나리오를 태스크로 본다면 하나의 외부 메시지로부터 시작된 시나리오가 모두 수행된 뒤, 마지막 발생하는 최종 출력 메시지가 제 시간 안에 발생했는지를 종료시한 초과를 판단할 수 있다.

이러한 가정들을 바탕으로, 사용자 수준에서 종료시한 초과 처리를 제공할 경우 발생할 수 있는 문제점들을 해결하기 위한 Run-Time system (RTS) 기반의 종료시한 초과 처리 메커니즘에 대해 살펴보고자 한다.

Ⅲ. 종료시한 초과 처리를 위한 Run-Time System의 디자인

3.1. 종료시한 초과의 분석

하나의 시나리오의 수행은, 시나리오의 시작이 되는 하나의 메시지가 시작캡슐에 전달됨으로써 시작된다. 이때 RTS레벨에서 사용되는 두 가지 종류의 메시 지 큐가 있는데 하나는 외부 메시지 큐로, 다중 멀티 쓰레딩을 사용하는 RTS에서 사용되는 메시지 큐로, 내부에서 발생할 수 있는 블로킹을 막기 위해 사용된다. 다른 하나는 내부 메시지 큐인데, 이는 기존의 UML-RT에서 사용하는 메시지 큐와 동일한 개념으로, 시나리오의 실제 수행에 필요한 메시지들을 포함한다. 따라서 하나의 시나리오의 수행이 요청되면, 하나의 시나리오의 시작이 되는 메시지는 시작캡슐에 전달되기 전 외부 메시지 큐로 전달된 뒤 내부 메시지 큐로 들어가 게 된다. 이 내부 메시지 큐에서 나와서 시작 캡슐에 전달되면서, 비로소 시나리오의 실제 수행이 시작된다.

따라서 이러한 과정에서, 종료시한 초과 처리를 위한 timer(또는 indicator)가 동작해야 하는 가장 정확한 시점은 그림 2와 같이 첫 번째 내부 메시지가 시나리오의 시작 캡슐에 전달되는 시점이 아닌 외부 메시지가 외부 메시지 큐로부터 나와 내부 메시지 큐로 들어가는 시점이 된다. 하지만 이 시점을 사용자 수준에서 정확히 알아내어 종료시한 초과 여부를 체크하기 시작하는 일은 어려운 일이다.

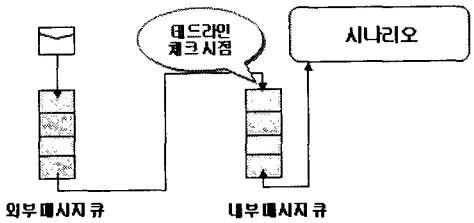

그림 2. Block Diagram of Enqueueing Check Point

따라서 정확한 종료시한 초과 여부를 판단하기 위해, 내부 메시지 큐에 시나리오 수행에 필요한 첫 번째 내부 메시지가 들어가는 시점에 timer를 등록하여 사용하는 방법을 제안한다. 이 때, timer를 등록하는 시점에 필요한 정보는 바로 종료시한이다. 또한 이 종료시한은 사용자로부터 얻어내야 하며 이를 위한 방법에 대하여 다음절에서 살펴본다.

3.2. 종료시한 초과 처리에 필요한 Information의 분석

시스템 안에서 사용될 종료시한은 사용자가 시스템을 디자인 할 때에 정해질 수도 있지만, 시스템을 구현한 뒤 실제 사용될 때 정해지는 경우도 있다. 따라서 종료시한을 처음 디자인 단계에서 값을 고정하는 것 보다, 이 값을 시스템 사용자로부터 넘겨받을 수 있는 방법을 생각하는 것이 더 바람직하다.

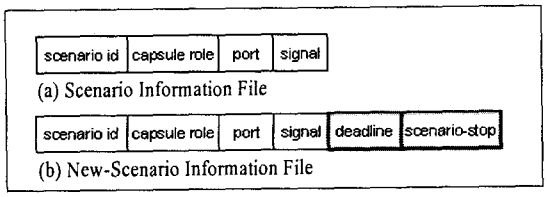

그림 3. Description of Information Files

시나리오 기반 멀티 쓰레딩 환경에서 시스템을 디자인한 뒤, 실제 코드를 생성하여 수행시키고자 할 때, 시나리오 기반 멀티 쓰레딩으로 변경하기 위하여 생성된 소스코드로부터 scenario description file (또는 analysis file)을 추출한다. [3]. 이 파일은 시나리오의 수행에 관련된 정보를 포함하며, 그림 3(a)와 같이 4개의 필드로 구성된다.

따라서 종료시한 초과 처리를 위해, 그림 3(b)와 같이 시나리오 정보 파일에 deadline 필드를 추가하여, 사용자로부터 원하는 종료시한 값을 읽어온 뒤, 이 시나리오 정보 파일에 포함시키는 방법을 제안한다. 추가로, 종료시한 값과 함께 종료시한 초과가 발생했을 때, 사용자가 원하는 처리를 해주기 위해 scenario-stop필드를 추가한다. 사용자 수준에서는 시나리오의 수행을 중간에 중단하기 위한 효과적인 방법이 존재하지 않기 때문에, 이 필드의 값에 따라 종료시한 초과가 발생했을 때 시나리오의 수행을 중단할 지에 대한 판단을 내린 뒤, RTS수준에서 적절한 처리를 제공하고자 한다.

3.3 종료시한 초과 처리를 위한 Information의 전달

시나리오 기반 멀티 쓰레딩에서 시나리오 정보파일의 값들이 실제 수행 모델에 적용되도록, 시나리오의 시작이 되는 하나의 외부 메시지가 외부 메시지 큐에 들어가기 전, 외부메세지의 필드로서 그 값들을 포함하도록 되어있다 [3].

따라서 앞에서 사용자로부터 얻은 종료시한 초과 처리에 필요한 정보들을 그림 4와 같이 시나리오의 시작이 되는 외부 메시지의 필드로서 전달해주는 방법을 제안한다.

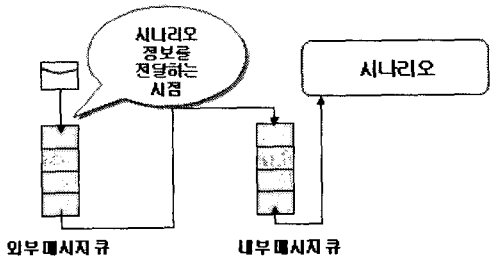

그림 4. Block Diagram of Information Transfer Point

이렇게 함으로써, 간단하게 종료시한 초과에 필요한 정보들을 전달할 수 있을 뿐 아니라, 시나리오가 시작되는 정확한 시점부터 종료시한 초과 여부를 체크를 시작할 수 있다.

Ⅳ. Embedded System 설계의 응용

4.1. 실시간 처리를 위한 메커니즘

종료시한 초과가 발생했을 때, 이를 실시간으로 처리하기 위해서 scenario-stop 필드의 값이 어떤 특정 값을 가질 경우, 사용자가 현재 종료시한 초과가 발생한 시나리오를 중단하기 원하는 경우로 판단하고 이에 대한 적절한 처리를 해줄 필요가 있다. 이를 위해 시나리오 수행의 중단에 대한 실시간 처리를 위해 시나리오와 관련된 내부 메시지 큐의 모든 메시지들을 삭제하는 방법을 제안한다. 왜냐하면 시나리오의 수행에 필요한 모든 메시지들은 그 시나리오가 매핑이 되어있는 쓰레드의 내부 메시지 큐에 들어오며, 내부 메시지 큐로부터 나온 메시지를 하나씩 처리하는 방법을 사용하여 시나리오를 수행하기 때문이다 [3]. 따라서 내부 메시지 큐에 메시지가 비어있다면, 그 시나리오는 더 이상 처리할 메시지들이 없어지므로, 수행을 진행할 수 없게 된다.

그러나 하나의 물리적인 쓰레드에 하나 이상의 시나리오가 매핑될 수 있기 때문에, 내부 메시지 큐의 모든 메시지들이 종료시한 초과가 발생한 시나리오와 관련된 메시지가 아닐 수 있다. 따라서 모든 내부 메시지를 삭제하는 대신, 중단하고자 하는 시나리오와 관련된 내부 메시지만을 삭제해야 한다. 이를 위해 우리는 시퀀스 번호라는 값을 사용한다. 이 값은 시나리오가 물리적인 쓰레드에 매핑되는 순간에 부여 받는 번호이며, 이 번호는 하나의 쓰레드에 매핑된 모든 시나리오마다 다른 값을 가지게 된다. 따라서 그림 5와 같이 종료시한 초과가 발생한 시나리오의 시퀀스 번호를 Xi 라고 하면, 그와 동일한 시퀀

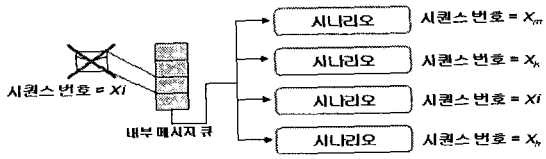

그림 5. Deleting an Internal Message using a Sequence Number

스 번호, *Xi*를 가지는 모든 내부 메시지를 검색하여 삭제하도록 한다면, 문제없이 해당 시나리오만을 멈출 수 있다.

4.2 실시간 처리를 위한 구현

본 절에서는 UML-RT를 지원하는 CASE 툴 중에서, IBM Rational사의 RoseRT를 선택하여, 기존의 RTS를 수정하고 실시간 임베디드 시스템에서의 timing의 실시간 처리를 위한 최적화 구현을 유도하였다.

우선, 종료시한 초과 처리를 위한 새로운 라이브러리를 그림 6과 같이 구현하였다. StartChecking()은 타이머 등록과 관련된 동작을 구현한 함수이다. 이 안에서 RoseRT에서 제공하는 Timer 포트를 등록하는데, 이 때 사용자로부터 얻은 deadline 값을 인자로 사용한다. MissHandling()은 종료시한 초과가 발생했을 때 실시간으로 처리해주기 위한 메커니즘을 실제 구현한 함수이며, stop-scenario값에 따라 다른 동작을 한다. 만약 stop- scenario 값이 0 외의 값을 가지면, 시나리오를 중단하기 위해, 내부 메시지 큐에서 해당 시나리오와 같은 시퀀스 번호를 가지는 모든 메시지를 지우는 동작을 해주고, 0을 가질 경우에는 단순히 체크만 해준 뒤 시나리오의 동작에는 영향을 미치지 않는다.

```
class RTDhandler{
    private:
        int deadline;
        int stop-scenario;
        int seqNo;
        RTController * controller;
        ...

    public:
        RTS_INLINE void RTDhandler( int, int,
                                    RTController *, int);
        ...
        RTS_INLINE void StartChecking();
        RTS_INLINE void MissHandling();
        ...
}
```

그림 6. Definition of RTDhandler

V. 실험의 결과

지금까지 종료시한 초과 처리를 위해 응용의 예로, 생산라인에서 물건을 조립하기 위한 assemble line을 디자인하여 종료시한 초과 발생 시 시나리오를 멈추게 하는 것을 살펴보면, 크게 로봇 팔 4개와 벨트 1개로 구성되며, 벨트와 로봇 팔은 서로 다른 시나리오로 동작하며, 동시에 하나의 물리적인 쓰레드에 매핑되어 동작한다.

5.1. Embedded System 을 위한 실시간 처리시스템

그림 7은 assemble line의 동작과정을 표현한 것이다. 화살표 방향으로 벨트가 진행되다 멈추면, 각각의 로봇 팔은 자신이 내려놓을 색깔의 부품을 내려놓게 된다. 만약 하나의 로봇 팔이 종료시한 안에 부품을 내려놓지 못하는 경우가 발생하여 오랜 시간 전체 시스템의 동작이 중단된다면, 제품의 생산성에 큰 타격을 입게 될 것이다, 따라서 특정한 값의 종료시한을 정해놓은 뒤, 그 시간 안에 수행이 완료되지 못했다면, 그 시나리오를 종료하여 다음 작업이 계속 진행될 수 있도록 해야 한다.

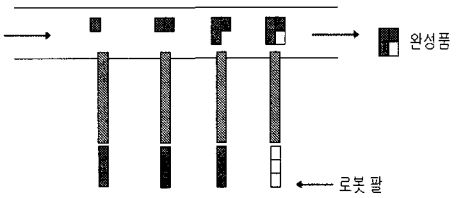

그림 7. An example of Deadline Miss Handling

이러한 시스템모델을 기반으로, 실험 환경을 설정하여 실험을 하였다. 실제적으로 특정 시나리오 안에 종료시한보다 길게, 혹은 짧은 값만큼 잠시 멈추도록 디자인했으며 이를 확인하기 위해 모델을 수행하는 동안 발생하는 출력 결과물을 분석하였다. 실험을 위해 위 모델에서의 로봇 팔과 벨트의 종료시한은 500msec로 정했다.

5.2. 실시간 처리시스템의 결과

사용자 수준에서 종료시한 초과 처리기를 디자인 할 경우, 정상적으로 동작하는지 체크하기 위하여, 새롭게 제안한 종료시한 초과 처리기를 사용하지 않고, 사용자 수준에서 단순한 timing기능만을 사용하여 종료시한 초과 처리 부분을 직접 추가하였다. 또한 종료시한 초과가 발생하는 경우를 만들기 위하여, 로봇 팔 중, 녹색 팔이 수행되는 시나리오 안에서 종료시한보다 긴 시간인, 1000 msec 동안 정지하도록 해 종료시한 초과를 유도하였다. 그리고 종료시한 초과가 발생한 경우 정상적으로 종료시한 초과 여부는 찾아냈지만, 한동안 정지된 상태가 지속되는 문제점을 보였다. 이는 해당 시나리오만을 중단하고 뒤에 처리해야 할 시나리오들을 제대로 처리해주지 못함을 의미한다.

따라서 새롭게 구현한 종료시한 초과 처리기를 적용하여, 종료시한 초과가 발생했을 때 성공적으로 이를 처리할 수 있는지 실험하였고, 그 결과는 그림 9와 같다.

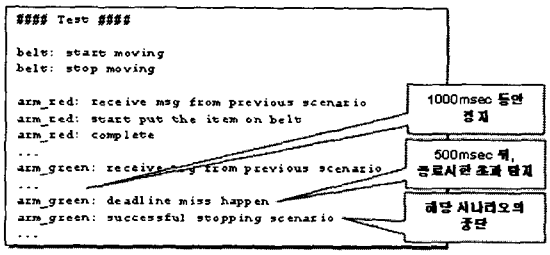

그림 9. Results of Deadline Miss Handling by Prosed Mechanism

이는 종료시한 초과가 발생할 경우 0.5sec 후, 해당 시나리오에서 deadline miss가 발생한 것을 감지하여, 즉시 시나리오를 중단하였고, 확인하기 위한 메시지를 출력하였다. 또한 해당 시나리오의 중단이 시스템 안의 다른 시나리오의 수행에 영향을 미치지 않음을 확인하였다.

VI. 결 론

본 고에서는 사용자가 직접 종료시한 초과 처리를 해줄 경우, 종료시한 초과 여부를 정확하게 판단하기 어려울 뿐 아니라 실험에서 증명된바와 같이 시나리오의 수행을 중간에 멈추는 것이 어렵다는 것을 보였다. 실시간 시스템을 디자인 할 때, 종료시한이라는 것은 대부분의 시스템에 고려해야 할, 실시간 시스템에서 중요한 제약조건이다. 따라서 종료시한 초과여부를 체크하고, 그에 대한 처리를 해주는 일은 대부분의 실시간 시스템에서 필요한 부분이다.

본 고에서 제시하고 있는 종료시한 초과 처리기는 종료시한 초과가 발생했을 때 즉시 시나리오의 중단이 가능할 뿐 아니라 재사용이 가능하기 때문에 실시간 시스템을 디자인하기에 보다 적합한 환경을 제공할 수 있을 뿐 아니라, 사용자에게 편리함을 제공해줄 수 있다. 이를 사용할 경우 예상되는 효과는, 정확한 시간에 종료시한 초과 여부를 체크함으로써 실시간 시스템에서의 최적화된 timing기능을 제공할 수 있다는 것이다. 이를 통하여 분산 처리를 요하는 실시간 시스템 뿐 아니라, 실시간으로 진행되는 임베디드 시스템을 위한 timing의 최적화 구현이 가능하게 된다.

참고문헌

[1] H. Yang, et al., "Improving Power Efficiency with Compiler-Assisted Cache Replacement", *Journal of Embedded Computing*, pp. 12-19, Dec., 2004.

[2] J. Chapman, et. al., "UNIX Performance on CC-NUMA Multiprocessor", *ACM Sigmetics Conf. on Measurement and Modeling of Computer System*, pp. 1-13, May 1995

[3] J. Masse, S. Kim, and S. Hong., "Tool Set Implementation for Scenario-based Multithreading of UML-RT Models and Experimental Validation.", *IEEE Real-Time/Embedded Tech. and App. Sym. (RTAS)*. pp. 70-77, Washington D.C., May 2003.

[4] S. Kim, M. Buettner, M. Hermeling, and S. Hong., "Experimental Assessment of Scenario-Based Multithreading for Real-Time Object-Oriented Models: A Case Study with PBX Systems.", *Int. Con. on Embedded and Ubiquitous Com. (EUC)*. Aizu, Japan, Aug., 2004.

[5] S. Kim, S. Hong, and N. Chang.,"Scenario-Based Implementation Architecture for Real-Time Object-Oriented Models", *2002 Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*. pp. 147-152, San Diego, CA, January 2002.

[6] T. Jeong, A. Ambler., "Design Trade-offs and Power Reduction Techniques for High Performance Circuits and Syste" Lecture Notes in Computer Science, ISSN: 0302-9743, May 2006

[7] M. Aron and P. Druschel, "Soft timers: efficient microsecond software timer support for network processing", *Pro. of the ACM Symp. on Operating Systems Principles*, 1999

[8] M. Saksena and P. Freedman and P. Rodziewicz, "Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems", *Pro. of IEEE Real-Time Systems Symp.*, pp. 240-251, June, 1997

저자소개

Eunjung Park received the B.S. degree from the Dep. of Computer Science, the Seoul Woman's University in 2002. After she graduated her M.S. degree from the Seoul National University, she joined the Dep. of Electrical and Computer Eng., at the Univ. of Delaware. Her research interests include Software engineering, Data communication, Computer networks, Sensor and Ad-hoc networks.

Taikyeong Jeong received the Ph.D. degree from the Dep. of Electrical and Computer Engineering, the University of Texas at Austin in 2004. He joined the University of Delaware, where he is now a research associate under the research grants of NASA (Grant No. NNG05GJ38G) in 2004, working on high performance VLSI design for next generation space robotics devices and system. His research interests include VLSI design, computer architecture, embedded system, and high performance system design.