Transactions of KSAE, Vol. 16, No. 2, pp.8-17 (2008)

1225-6382/2008/092-02

실시간 운영체제를 적용한 제어시스템의 모델기반 설계 및 검증

연 제 명[1] · 마 주 영[1] · 선 우 명 호*[2] · 이 우 택[3]

한양대학교 자동차공학과 대학원[1] · 한양대학교 자동차공학과[2] · 창원대학교 메카트로닉스공학부[3]

Model Based Design and Validation of Control Systems using Real-time Operating System

Jeamyoung Youn[1] · Jooyoung Ma[1] · Myoungho Sunwoo*[2] · Wootaik Lee[3]

[1]Department of Automotive Engineering, Graduate School, Hanyang University, Seoul 133-791, Korea
[2]Department of Automotive Engineering, Hanyang University, Seoul 133-791, Korea
[3]Department of Mechatronics Engineering, Changwon National University, Gyeongnam 641-773, Korea
(Received 4 October 2006 / Accepted 9 November 2007)

Abstract : This paper presents the Matlab/Simulink-based software-in-the-loop simulation(SILS) environment which is the co-simulator for temporal and functional simulations of control systems. The temporal behavior of a control system is strongly dependent on the implemented software and hardware such as the real-time operating system, the target CPU, and the communication protocol. The proposed SILS abstracts the system with tasks, task executions, real-time schedulers, and real-time networks close to the implementation. Methods to realize these components in graphical block representations are investigated with Matlab/Simulink, which is most commonly used tool for designing and simulating control algorithms in control engineering. In order to achieve a seamless development from SILS to rapid control prototyping (RCP), the SILS block-set is designed to support automatic code generation without tool changes and block modifications.

Key words : Co-simulation(연계 모의실험), RTOS(Real-time operating system: 실시간 운영체제), SILS(Software-in-the-loop simulation: 소프트웨어를 포함한 시뮬레이션), Scheduling(스케줄링)

1. 서 론

자동차, 로봇, 항공기 등에 적용되는 실시간 내장형 제어시스템의 기능이 점차 증가되고 복잡화 되는 추세이다. 또한, 각 기업은 개발 기간의 단축과 비용 절감을 통하여 제품의 경쟁력을 높여야 하는 상황에 직면하고 있다. 이와 같은 제품 개발 환경의 변화에 대응하기 위하여 제안된 것이 모델기반의 제어시스템 개발방식이다.

모델기반의 제어시스템 개발방식에서 제어기 설계자는 컴퓨터 시뮬레이션 도구를 이용하여 제어 요구사항을 만족시키기 위한 제어 알고리즘을 설계하고 검증한다. 제어 알고리즘의 설계 결과로부터 얻어진 제어기의 기능적, 시간적 요구사항을 바탕으로 제어 소프트웨어의 설계가 이루어진다. 모델기반의 개발방식에서는 제어기 설계자와 소프트웨어 설계자 사이에서 발생할 수 있는 오류를 방지하고 개발 기간 및 비용을 절감하기 위하여, 자동코드생성방식을 채택하고 있다. 자동코드생성 방식을 통하여 최종적으로 얻어지는 제어 소프트웨어는 Hardware-in-the-loop simulation(HILS)이나 실제 플랜트에 적용되어 제어성능과 신뢰성을 검증받는다.[1)]

*Corresponding author, E-mail: msunwoo@hanyang.ac.kr

그러나 모델기반 제어기 개발을 위하여 이용되고 있는 대부분의 개발 환경은 제어기의 기능적 측면만을 고려한 시뮬레이션을 지원하고 있다. 결과적으로 시뮬레이션 단계에서 검증된 제어 결과는 구현 후의 제어 성능과 차이를 보이게 되며, 이를 줄이기 위한 보정과정에 추가적인 시간과 비용이 소요된다. 최근 이러한 문제점을 극복하기 위하여, 구현 소프트웨어를 포함한 제어기 플랫폼의 환경으로부터 발생하는 시간 지연 문제를 제어기 설계자가 설계 단계에서 고려하여 제어 알고리즘을 시뮬레이션 할 수 있는 설계 도구들이 개발되고 있다.[2,3] 하지만, 기존에 개발된 설계 도구들은 소프트웨어 설계 요소의 표현 방식이 제어기 설계자가 접근하기에 용이하지 않으며, 자동코드생성에 적합하지 않다는 문제점이 있다.

본 연구에서는 제어기 설계자가 설계 단계에서 제어기 플랫폼의 환경을 용이하게 고려할 수 있는 방법을 제시함으로써, 효과적인 제어 알고리즘의 설계와 검증이 가능하도록 한다. 또한, 제안된 방법을 이용하여 설계된 제어 알고리즘으로부터 구현 소프트웨어를 자동으로 생성하도록 함으로써, 소프트웨어 설계의 효율성 및 강건성을 높이도록 한다.

2. 연구 배경 및 필요성

컴퓨터를 이용한 제어기 설계(Computer Aided Control System Design, CACSD)는 제어기 설계분야의 보편적인 설계 방법이 되었다. 최근에는 모델기반의 제어시스템 개발에 CACSD를 이용하기 위한 다양한 연구가 진행되고 있다.[2-6]

Mathworks사의 Matlab/Simulink는 제어기 설계분야의 대표적인 CACSD 도구로 모델기반의 제어시스템 개발을 지원한다. Simulink는 제어 알고리즘을 블록 다이어그램 형태로 표현할 수 있는 환경을 제공하며, Simulink 모델로부터 구현 코드를 자동으로 생성할 수 있는 환경을 제공한다. 이와 같은 개발 환경을 이용하면 제어 알고리즘의 설계 단계와 구현 단계를 유기적으로 연계할 수 있으며, 설계한 제어 알고리즘을 제어기 플랫폼에 빠르게 구현해봄으로써 실제 작동환경에서의 제어기 성능을 검증할 수 있다.[7] 하지만, 현재 제공되고 있는 시뮬레이션 환경으로는 최근 적용이 증가되고 있는 실시간 운영체제 기반의 제어기 플랫폼의 특성을 고려한 제어시스템의 설계가 용이하지 않다. Simulink가 실시간 운영체제의 설계 단위인 태스크(task)를 기반으로 하는 제어 알고리즘의 설계와 구현 코드의 생성을 지원하지 않기 때문이다. 또한, 제어기 플랫폼의 성능에 따라 달라지는 태스크의 실행 속도를 설계 단계에서 고려할 수 없어, 제어기 플랫폼에 최적화된 제어 알고리즘을 설계하는데 어려움이 있다. 이와 같은 문제는 Simulink 뿐만 아니라, 대부분의 CACSD 환경이 가지는 공통적인 문제이다. 반면, 실시간 운영체제의 적용은 제어기의 기능적, 시간적 성능을 보장할 뿐만 아니라 제어기 플랫폼의 정형화를 통한 소프트웨어의 재사용성 및 관리의 효율성 향상을 위하여 그 적용이 확대될 전망이다. 따라서 제어기의 설계와 구현을 유기적으로 연계할 수 있도록, 실시간 운영체제 기반의 제어기 플랫폼의 특성을 설계 단계에서 고려할 수 있는 환경이 요구된다.

Fig. 1은 한 개의 제어기 플랫폼에서 수행되는 제어 알고리즘의 시뮬레이션 환경을 비교한 것이다. 왼쪽은 Simulink에서 기본적으로 제공하는 제어 알고리즘의 시뮬레이션 환경으로, 제어 알고리즘의 실행시간이 '0' 또는 단순한 시간지연으로 고려됨을 알 수 있다. 반면, 오른쪽은 Software- in-the-loop simulation (SILS) 환경을 이용하여 태스크 기반으로 설계된 제어 알고리즘을 실시간 운영체제를 고려하여 시뮬레이션 하는 환경이다.

또한, 실시간 분산 제어시스템에 제어 알고리즘을 적용하는 경우, 실시간 네트워크의 사용으로 인하여 발생하는 시간 지연 문제를 고려한 설계가 요

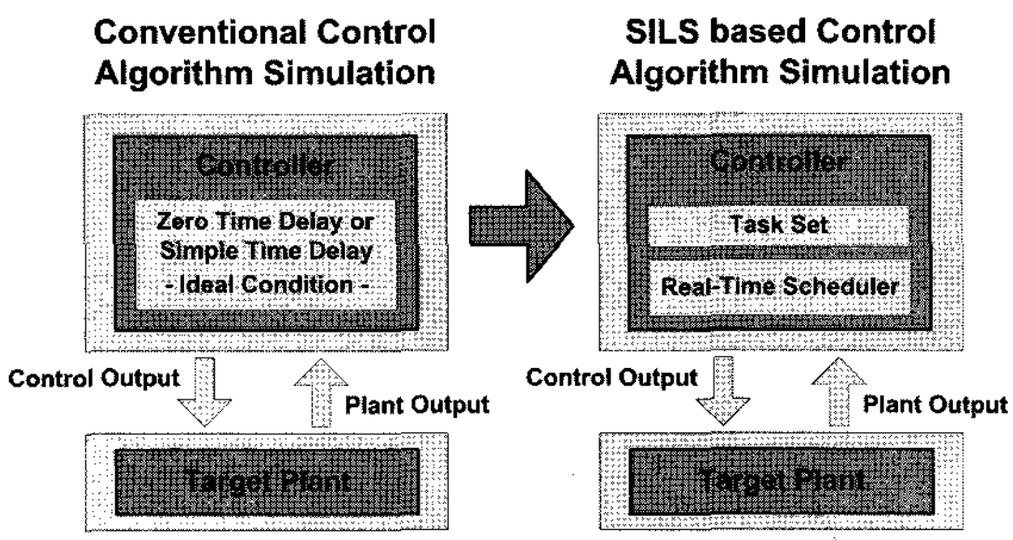

Fig. 1 Control algorithm simulation with single-node

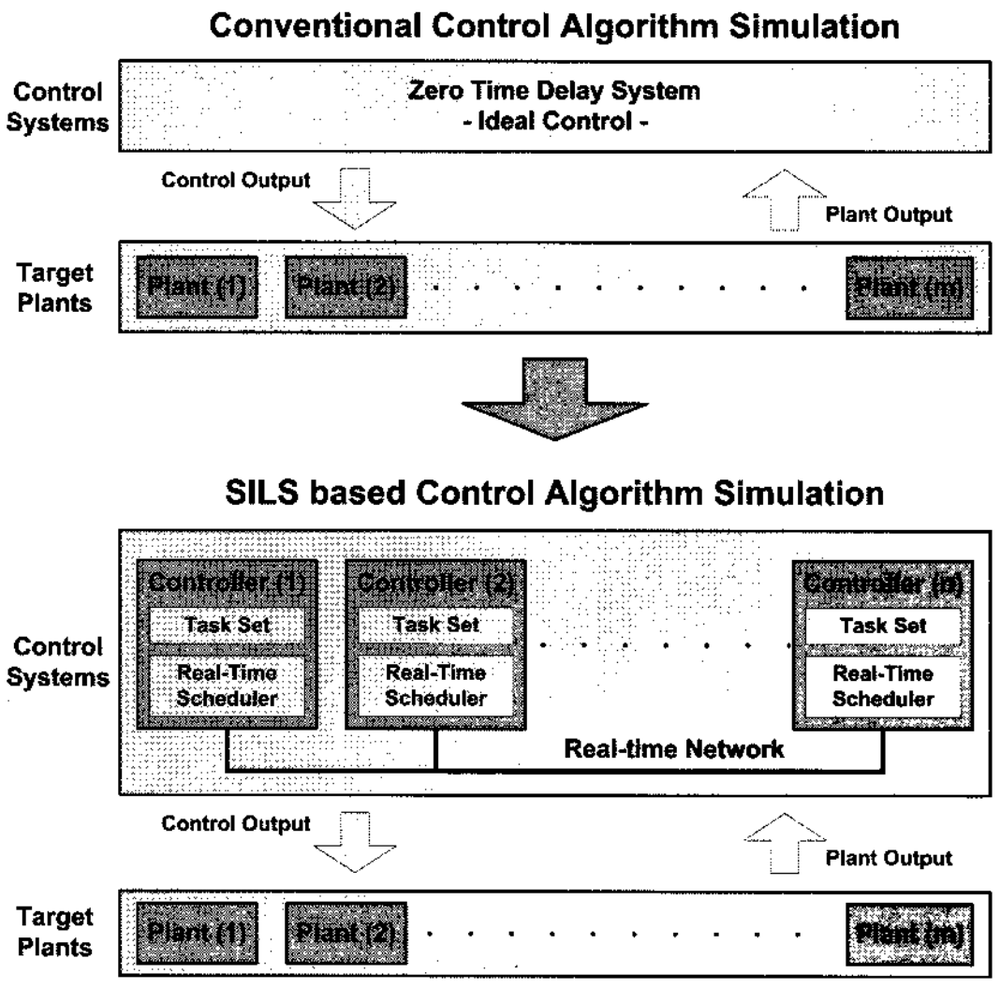

Fig. 2 Control algorithm simulation with multi-nodes

구된다. Fig. 2는 기존의 Simulink에서 제공하는 시뮬레이션 환경과 SILS에서 제공하는 실시간 운영체제 및 실시간 네트워크를 고려하는 시뮬레이션 환경을 비교한 것이다.

본 연구를 통하여 태스크의 실행시간과 실시간 운영체제 및 실시간 네트워크의 사용으로 인한 시간지연을 고려하여 태스크 단위의 제어 알고리즘을 설계하고 검증할 수 있는 SILS 환경을 개발하도록 한다.

3. SILS 환경 구축

제어기 플랫폼에 적용되는 실시간 운영체제와 실시간 네트워크로 인하여 발생하는 시간문제를 고려하기 위해서는 다음의 내용을 고려한 SILS 환경이 요구된다.

- 스케줄링 알고리즘을 고려한 실시간 운영체제 커널(kernel) 모델
- 태스크 실행시간을 고려한 블록 다이어그램 형태의 태스크 모델
- 네트워크 프로토콜의 특성을 고려한 실시간 네트워크 커널 모델

3.1 스케줄링 커널 모델링

본 연구에서는 정적 시분할 방식의 스케줄링 알고리즘과 우선순위 기반의 스케줄링 알고리즘을 가지는 스케줄링 커널에 대하여 모델링 한다.

3.1.1 정적 시분할 방식의 스케줄러 모델

정적 시분할 방식의 스케줄러는 태스크의 실행 순서와 실행 시기에 대한 정보를 가지는 스케줄 표에 따라 태스크를 실행시킨다. 스케줄 표는 실행되는 태스크의 기능적, 시간적 요구사항을 반영하여 설계단계에서 작성되며, 일정한 주기를 가지고 반복적으로 실행된다. 스케줄러에 의하여 CPU의 점유권을 할당받아 실행되는 태스크는 Fig. 3과 같이 실행상태로 전환되며, 태스크의 실행시간이 경과한 이후에 종료상태로 전환된다. 결과적으로, 정적 시분할 방식의 스케줄러 모델은 스케줄 표에 따라 태스크를 실행 시키는 기능만을 수행하게 된다.

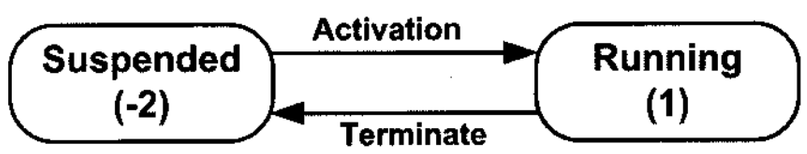

Fig. 3 Task state transition of static cyclic scheduler

3.1.2 우선순위 기반의 스케줄러 모델

우선순위 기반의 스케줄러는 이벤트 방식의 스케줄링 알고리즘으로, 태스크의 실행 요청을 기반으로 CPU의 점유권을 할당하는 스케줄링 방식이다. 스케줄러가 동시에 여러 개의 태스크로부터 실행을 요청받는 경우, 태스크에 부여된 우선순위에 따라 실행 순서를 결정한다. 우선순위 기반의 이벤트 방식에서 태스크는 종료, 준비, 실행의 세 가지 상태를 가진다. 종료상태에 있는 태스크가 이벤트의 발생에 의하여 실행을 요청하게 되면 CPU 점유권을 얻기 위하여 준비상태로 전환된다. 준비상태에 있는 태스크는 스케줄링 커널로부터 CPU 점유권을 할당받아 실행 상태로 전환된다. 태스크가 실행 중에 스케줄러에 의하여 CPU 점유권을 다른 태스크에게 넘겨 주어야하는 경우, 태스크는 준비상태로 전환된다. 한번 실행된 태스크는 CPU를 점유하는 시간이 태스크 실행시간과 동일할 때까지 지속적으로 CPU 점유를 요청하게 된다. 실행시간만큼의 시간 동안 CPU를 점유한 태스크는 종료상태로 전환된다. Fig. 4는 우선순위 기반의 태스크가 가지는 상태를 도시한 것이다.

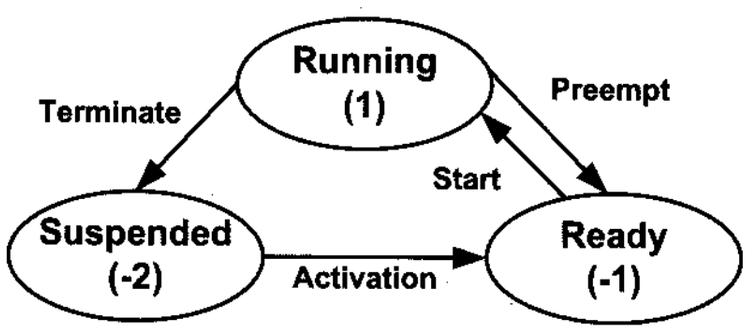

Fig. 4 Task state transition of priority based scheduling

우선순위 기반의 스케줄러는 정적 시분할 방식의 스케줄러와 달리 태스크의 실행이 이벤트의 발생으로 이루어지므로, 이벤트를 발생시키기 위한 운영체제 서비스를 모델링 하여야 한다. 대부분의 실시간 운영체제에서 제공하는 이벤트는 시간과 관련된 이벤트와 시간과 무관한 이벤트로 구분된다. 두 운영체제 서비스를 구별하기 위하여 본 연구에서는 시간과 관련된 이벤트를 알람(alarm) 이벤트로 정의한다.

이벤트는 이벤트의 발생으로 이벤트에 연결된 태스크를 준비상태로 전환하도록 모델링 하며, 알람 이벤트는 이벤트가 발생하고 일정 시간이 경과한 이후에 이벤트에 연결된 태스크를 준비상태로 전환하도록 모델링 한다. 두 운영체제 서비스를 스케줄링 커널이 관리하여 태스크의 실행을 결정한다.

3.2 태스크 모델링

실시간 운영체제 기반의 제어기 플랫폼에서는 스케줄링 커널에 의하여 관리되는 태스크를 기본 실행단위로 하므로, 실제 태스크의 실행을 모사할 수 있는 시뮬레이션 모델이 요구된다.

본 연구에서는 태스크가 입력 데이터를 이용하여 출력 데이터 값을 계산하는 데에 고정된 시간이 소요되는 것으로 모델링 한다. 태스크의 입력 데이터는 태스크의 실행이 시작되는 시점에서 동기화 되어 출력 데이터를 계산하는데 이용되는 것으로 고려한다. 출력 데이터는 태스크의 실행시간이 경과하여 태스크의 실행이 종료되는 시점에 동기화 되어 다른 태스크나 플랜트에 전달된다고 가정한다. 결과적으로, 태스크의 모델은 입력 데이터를 이용하여 출력 데이터를 계산하는 태스크 계산(task computation) 부분과 태스크의 실행시간이 경과한 이후에 출력 데이터를 내보는 태스크 출력(task output) 부분으로 구성된다. 시뮬레이션에서 태스크

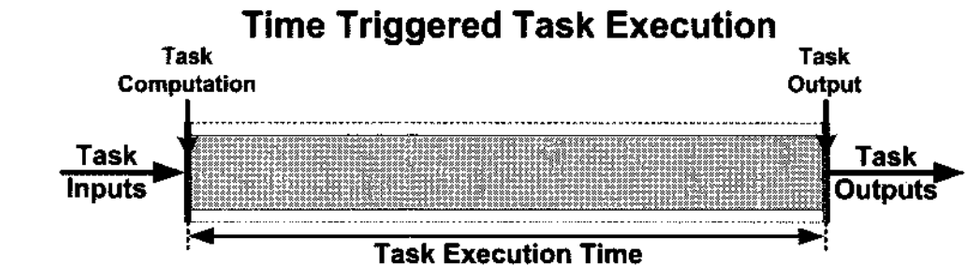

Fig. 5 Task execution model at static cyclic scheduler

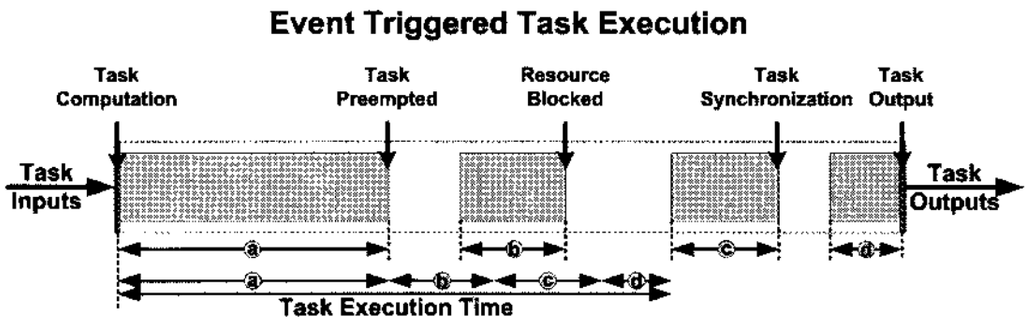

Fig. 6 Task execution model at priority based scheduling

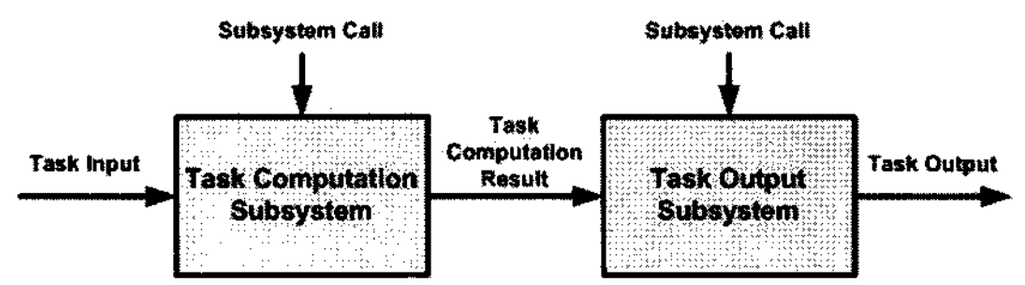

Fig. 7 Simulink block model of task execution

의 계산 부분과 출력 부분의 실행은 스케줄링 커널에 의하여 관리된다. 이는 스케줄링 방식에 따라 태스크 실행이 연속적이지 않을 수 있기 때문이다.

시분할 방식에서는 태스크가 실행되는 동안 실행시간 이상의 CPU 점유권을 보장받기 때문에 Fig. 5와 같이 태스크 실행이 연속적인 것으로 모델링 할 수 있다. 반면에, 이벤트 방식의 스케줄링의 경우 Fig. 6과 같이 태스크의 실행이 우선순위가 높은 태스크나 공유자원의 사용으로 인하여 중단될 수 있어, 태스크의 실행이 시작되고 실행시간 이상의 시간이 경과한 이후에 태스크가 종료된다. 따라서 Fig. 7과 같이 태스크 계산 부분과 출력 부분의 실행을 독립적으로 모델링함으로써, 태스크 실행을 스케줄링 커널에서 유연하게 관리할 수 있도록 한다.

3.3 네트워크 모델링

제어기 간의 정보공유를 위하여 사용되는 네트워크는 분산제어 방식에서 매우 중요한 부분을 차지한다. 네트워크의 사용으로 인하여 발생하는 시간지연이 전체 제어시스템의 성능 저하를 야기할 수 있기 때문이다. 따라서 네트워크 기반의 제어시스템을 설계할 때에는 네트워크 이용에 따른 시간 지연 문제를 고려하여야 한다.

네트워크에 의하여 야기되는 시간 지연은 적용되는 네트워크 프로토콜의 종류에 따라 달라지므로, 네트워크 프로토콜의 특성을 고려한 메시지 송·수신 모델을 이용하여 시간지연 문제를 고려하여야 한다. 네트워크 모델링에는 메시지의 송·수신 방식과 전송시간에 대한 모델이 포함된다. 본 연구에서는 자동차 분야에서 가장 보편적으로 사용되고 있는 이벤트 방식의 Control Area Network(CAN) 프로토콜과 시간 시분할 방식 Local Interconnect Network(LIN) 프로토콜에 대한 네트워크 모델링을 수행하도록 한다.

3.3.1 CAN 네트워크 모델링

3.3.1.1 CAN 메시지 전송

CAN은 CSMA(Carrier Sense Multiple Access) 방식의 통신 프로토콜로, 이벤트 방식으로 메시지를 전송하며 네트워크에 연결된 모든 노드에서 전송되는 메시지를 수신할 수 있다.[8] CAN 메시지는 메시지의 식별을 목적으로 고유의 아이디를 할당 받는다. 네트워크에 연결된 노드들은 전송되는 메시지의 아이디를 모니터링 하여 메시지의 수신여부를 결정한다. 또한, CAN 네트워크는 한 번에 한 개의 메시지 전송만을 허용하므로, 동시에 여러 개의 메시지로부터 전송을 요청받는 경우 아이디 값을 이용하여 메시지의 우선순위를 결정한다.

Fig. 8은 CAN 네트워크의 모델을 도식화 한 그림이다. 본 연구에서는 FullCAN을 사용하는 CAN 네트워크를 모델링 하였으며, 메시지의 송·수신이 다음과 같이 이루어진다.

- FullCAN 제어기가 송신버퍼에서 대기 중인 메시지 가운데 우선순위가 가장 높은 메시지를 결정하여, CAN 메시지 스케줄러로부터 할당받은 전송버퍼에 메시지를 저장한다.
- CAN 메시지 스케줄러는 CAN 버스의 동작을 모델링 한 것으로, 각 노드에 할당한 전송버퍼에 저장되어 있는 메시지 가운데 우선순위가 가장 높은 메시지를 선정하여 네트워크에 연결되어 있는 FullCAN 제어기에 메시지를 전달한다. 이때, 메시지는 3.3.1.2의 전송시간 모델로부터 계산된 시간만큼 지연되어 전달된다.
- 메시지를 수신한 FullCAN 제어기는 메시지 아이디를 모니터링 하여 메시지의 수신여부를 결정하고, 해당되는 수신버퍼에 메시지를 저장한다.

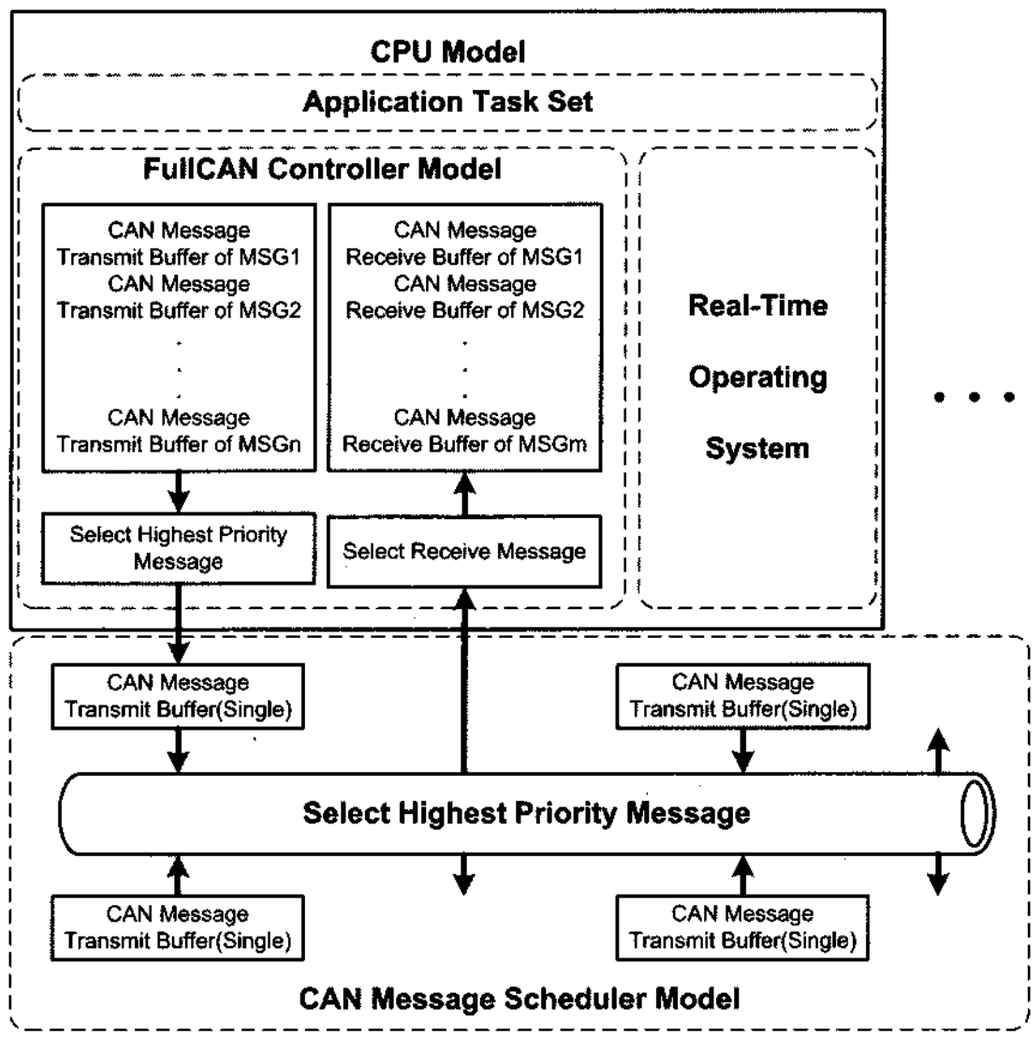

Fig. 8 CAN communication modeling

3.3.1.2 CAN 메시지 전송시간

CAN 네트워크에서는 한 개의 메시지 프레임에 최대 여덟 바이트(byte)의 데이터를 전송할 수 있으며, 메시지 프레임의 구성을 위하여 데이터 외에, 메시지의 시작, 식별, 동기화, 오류확인, 종료 등을 위하여 최소 47 비트(bit) 이상을 추가한다. 또한, CAN 프로토콜에서 s 바이트의 데이터를 포함하는 메시지의 최대 스터프 비트(stuff bits)는 식 (1)과 같다. 한 개의 비트를 전송하는데 소요되는 비트전송 시간을 τ_{bit}으로 정의하면, CAN 메시지 프레임의 최대 전송시간 C는 식 (2)와 같이 표현된다. 식 (2)를 이용하여 CAN 네트워크 시뮬레이션에 사용되는 CAN 메시지 전송시간을 계산하도록 한다.[9]

$$(Stuffbits)_{\max} = \left\lfloor \frac{34+8s-1}{4} \right\rfloor \quad (1)$$

$$C = \left(8s+47+\left\lfloor \frac{34+8s-1}{4} \right\rfloor\right)\tau_{bit} \quad (2)$$

3.3.2 LIN 프로토콜 모델

3.3.2.1 LIN 메시지 전송

LIN은 마스터/슬레이브(master/slave) 방식의 네트워크 프로토콜로, 마스터 태스크의 전송 스케줄

에 따라 메시지가 전송 된다. 마스터 태스크는 메시지의 전송 주기와 전송 시간에 기초하여 시분할 방식으로 작성된 메시지 스케줄 표에 따라 헤더(header) 프레임을 전송한다. 헤더 프레임에는 메시지의 식별을 목적으로 메시지에 할당하는 아이디가 포함된다. 네트워크에 연결된 노드들의 슬레이브 태스크는 수신 받은 헤더 프레임의 아이디를 모니터링 하여 응답(response) 프레임의 송·수신여부를 결정한다. Fig. 9는 LIN 네트워크의 모델을 도식화한 그림이다. LIN 네트워크 모델에서는 메시지의 송·수신이 다음과 같이 이루어진다.

- 마스터 태스크가 주기적으로 실행되면서, 메시지 스케줄 표에 따라 헤더 프레임을 슬레이브 태스크에 전달한다. 이때, 헤더 프레임은 3.3.2.2의 헤더 프레임 전송시간 모델로부터 계산된 시간만큼 지연되어 전달된다.
- 네트워크에 연결된 슬레이브 태스크들이 헤더 프레임의 아이디를 모니터링 하여 응답 프레임의 송·수신을 결정한다.
- 한 개의 슬레이브 태스크에서 헤더 프레임이 전송되고, 메시지를 수신하는 슬레이브 태스크들이 메시지를 해당되는 메시지 버퍼에 저장한다. 이때, 응답 프레임은 3.3.2.2의 응답 프레임 전송시간 모델로부터 계산된 시간만큼 지연되어 전송된다.

3.3.2.2 LIN 메시지 전송시간

LIN 네트워크의 헤더 프레임은 메시지의 동기화와 식별을 위하여 전송되며, 식 (3)과 같이 네 개의 시간상수와 전송속도(τ_{bit})의 관계식으로 표현된다.

$$T_{header} = 34\tau_{bit} + P_{INTER} + P_{SYNBRK} + P_{SYNDEL} + P_{ID} \quad (3)$$

LIN 메시지의 응답 프레임은 최대 여덟 바이트의 데이터를 전송할 수 있으며, 데이터 전송 오류 확인을 위하여 한 바이트의 체크섬(checksum)을 추가한다. 결과적으로, 응답 프레임의 전송시간은 식 (4)와 같이 두 개의 시간상수와 전송속도, 데이터 크기(s 바이트)의 관계식으로 표현된다.[10]

$$T_{response} = 10(s+1)\tau_{bit} + P_{INFRAME} + P_{INTERBYTE} \times s \quad (4)$$

3.4 SILS Simulink 블록

Fig. 10, 11, 12는 앞서 모델링한 SILS 설계 요소를 Simulink 블록으로 표현한 것이다. Matlab/Simulink에서 제공하는 s-function을 사용하여, 시뮬레이션과 구현 코드의 자동생성을 지원하도록 블록을 구현한다.[11] 우선순위 기반의 스케줄러의 코드 생성은 자동차 분야에서 표준으로 자리 잡고 있는 실시간 운영체제인 OSEK을 지원하도록 한다.[12] 또한 이전의 연구를 통하여 개발한 Rapid Control Prototype(RCP)

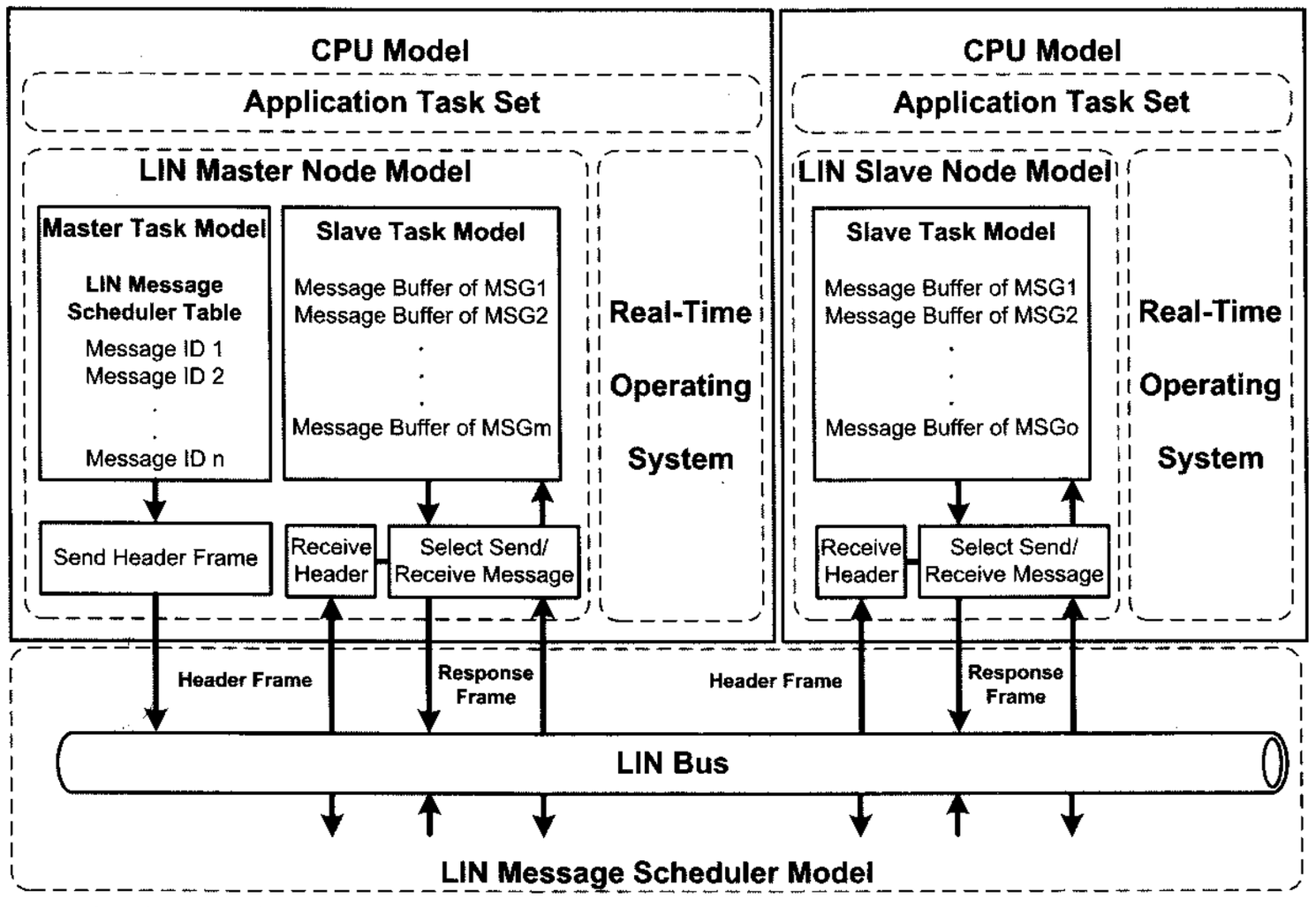

Fig. 9 LIN communication modeling

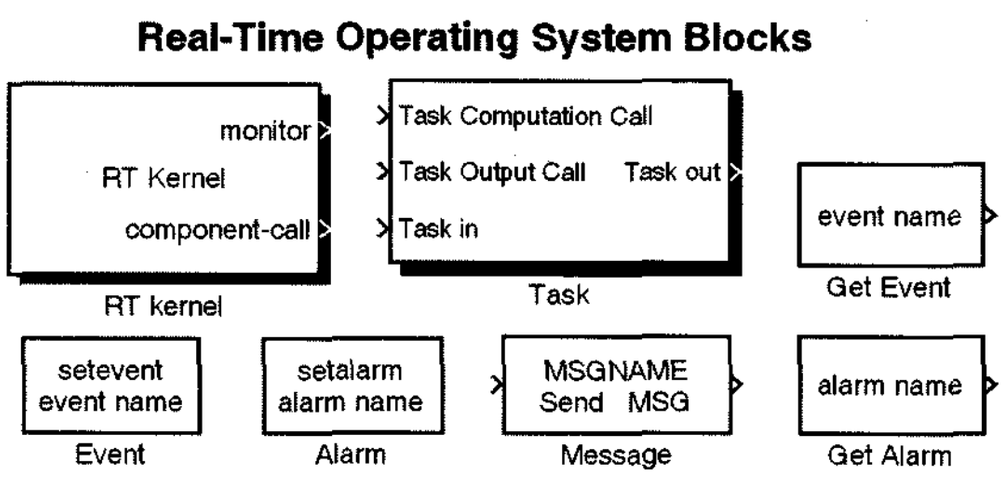

Fig. 10 Simulink blocks for RTOS implementation

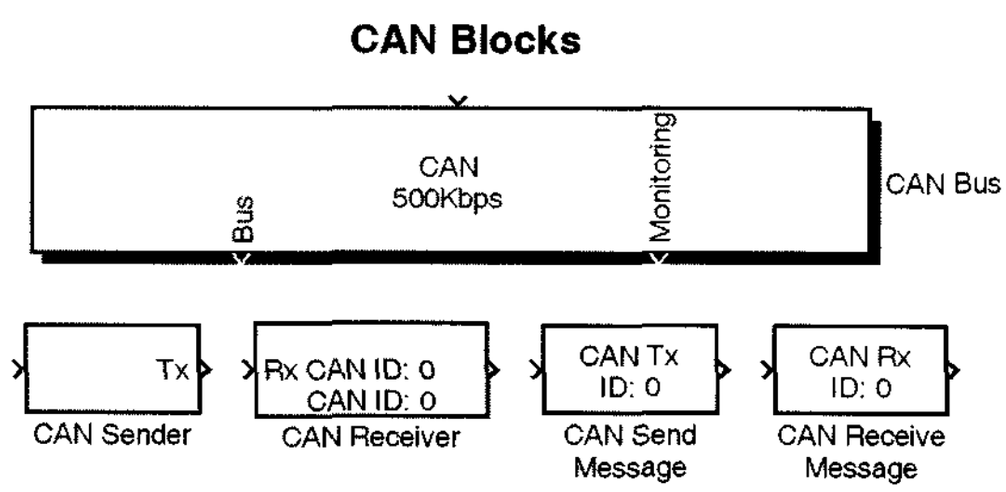

Fig. 11 Simulink blocks for CAN implementation

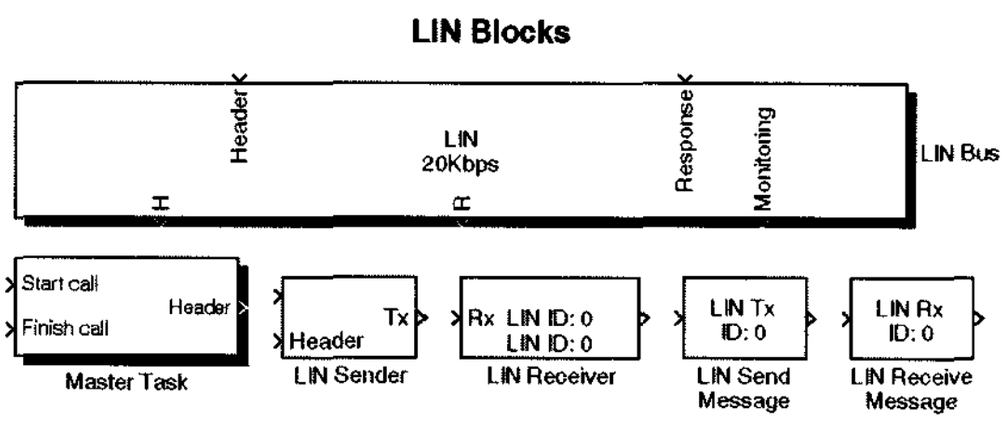

Fig. 12 Simulink blocks for LIN implementation

환경과의 연계를 통하여, 제어 알고리즘의 설계에서부터 구현까지의 과정을 단일화하도록 한다.[13)]

4. SILS 유용성 검증

Fig. 13은 SILS의 유용성을 검증하기 위한 실험환경을 도식화한 것이다. 제어기 플랫폼에 제어로직을 구현함으로써 발생하는 제어성능의 변화를 중점적으로 확인하고, 플랜트 모델링 오차 및 외부환경으로 인하여 발생하는 제어성능의 변화를 배제하기 위하여 HILS 환경을 이용한다.

실험에 사용되는 제어시스템은 제어기와 플랜트 사이의 물리적 정보가 CAN을 통하여 이동하는 스마트 센서·액추에이터기반의 분산제어시스템이고, 한 개의 제어기를 이용하여 Fig. 14와 같은 세 개의 도립진자를 동시에 제어하도록 구성한다.

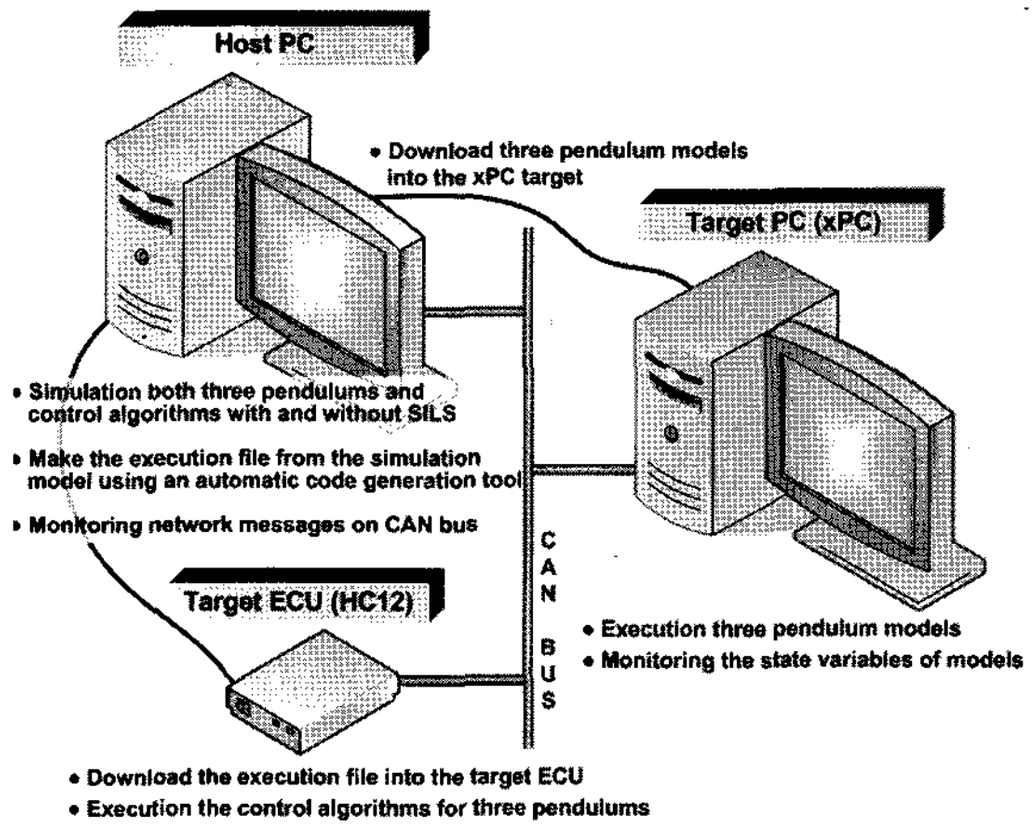

Fig. 13 HILS system for the evaluation of SILS environment

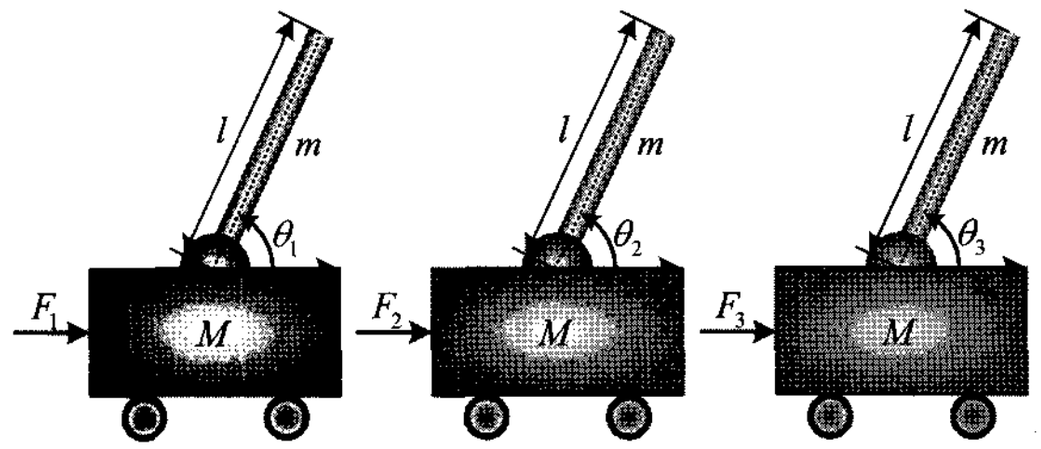

Fig. 14 Inverted pendulums controlled by a networked control system

제어기 설계 및 검증을 위한 시뮬레이션 및 HILS에서 사용되는 도립진자시스템의 수학적 모델은 다음 식 (5), (6)과 같다.

$$\ddot{x}=\frac{-mg\sin\theta\cos\theta+ml\dot{\theta}^2\sin\theta}{M+m\sin^2\theta}+\frac{1}{M+m\sin^2\theta}F \tag{5}$$

$$\ddot{\theta}=\frac{(M+m)g\sin\theta-ml\dot{\theta}^2\sin\theta\cos\theta}{Ml+ml\sin^2\theta}-\frac{\cos\theta}{Ml+mlsin^2\theta}F \tag{6}$$

여기서, 카트의 위치(x)와 도립진자의 각도(θ)를 상태변수로 정의하고, 카트에 가해지는 힘(F)을 이용하여 두 변수를 제어하는 PID와 LQR 제어 로직을 설계한다.

SILS 및 구현에 사용하는 OSEK과 CAN의 구성은 Table 1, 2와 같다. 각 도립진자는 센서입력 처리 및 상태변수 추정을 위한 태스크와 제어 출력 계산을 위한 태스크, 총 두 개의 태스크에 의하여 제어된다.

전체 노드 및 태스크의 구성은 Fig. 15와 같다. 각 도립진자에 스마트 센서와 액추에이터가 한 개씩 연결되어 있다. ECU로부터 CAN통신을 통하여 액

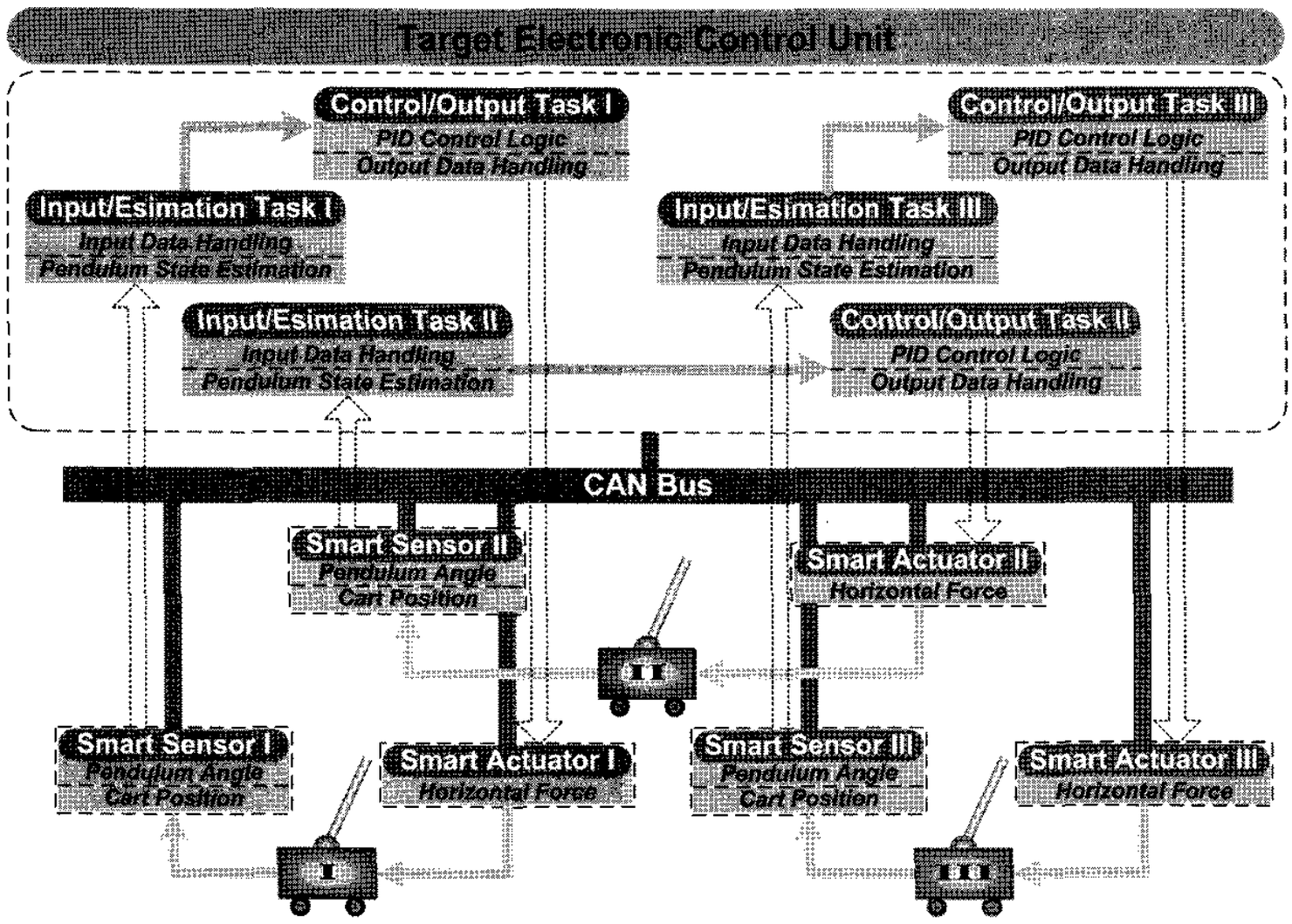

Fig. 15 Nodes and tasks configuration

Table 1 Task information

Task name	Period [ms]	exe. time [ms]	Preemption	Priority	
				Case I	Case II
Pend1_In_T	10	0.7	yes	13	13
Pend1_Out_T	10	0.8	yes	8	12
Pend2_In_T	10	0.7	yes	12	11
Pend2_Out_T	10	0.8	yes	9	10
Pend3_In_T	10	0.7	yes	11	9
Pend3_Out_T	10	0.8	yes	10	8

Table 2 CAN information: 500kbps

Message name	ID	Data length [bytes]	Data type	Period [ms]
Cart1_pos_msg	1	8	Double	10
Pend1_ang_msg	2	8	Double	10
Cart1_input_msg	3	8	Double	10
Cart2_pos_msg	4	8	Double	10
Pend2_ang_msg	5	8	Double	10
Cart2_input_msg	6	8	Double	10
Cart3_pos_msg	7	8	Double	10
Pend3_ang_msg	8	8	Double	10
Cart3_input_msg	9	8	Double	10

추에이터 노드가 액추에이터 정보를 받아 도립진자를 구동하고, 센서 노드가 도립진자의 위치와 각도 정보를 CAN통신을 이용하여 ECU에 제공한다.

SILS에서의 시뮬레이션 결과와 RCP/HILS를 이

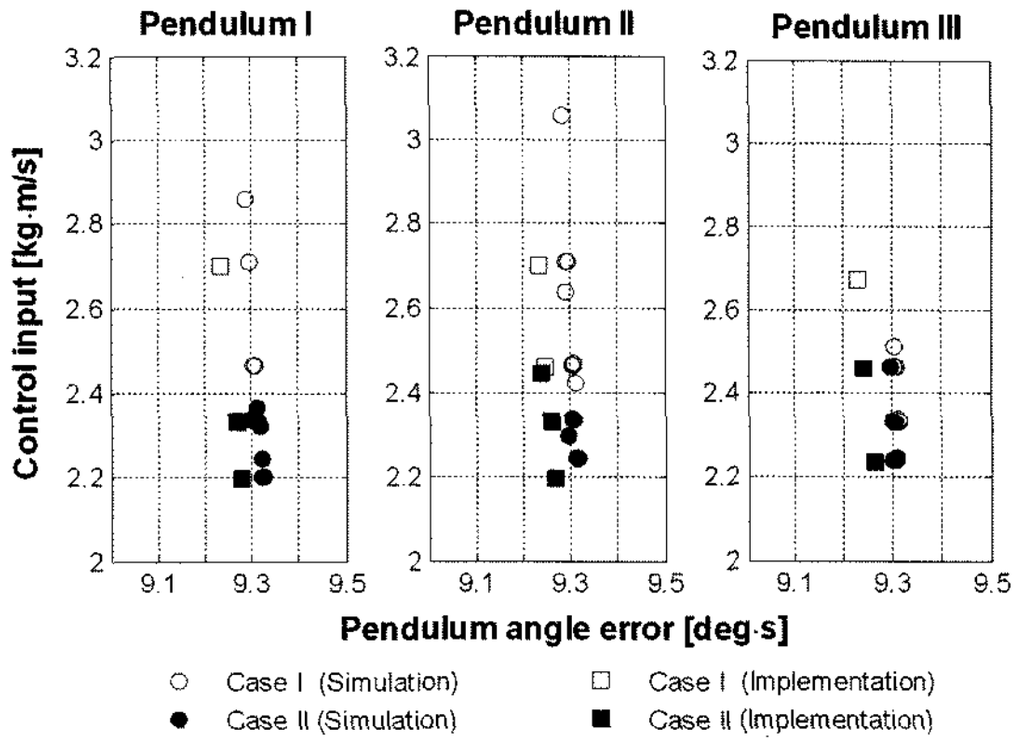

Fig. 16 Control performance versus priority assignment

용한 구현결과를 비교평가하기 위하여 식 (7)과 같은 제어 입력(F)의 적분 값을 이용한다.[14)]

$$\int_{t_0}^{t_f} |F| dt \tag{7}$$

태스크의 우선순위 할당에 따른 제어 성능 변화를 비교하기 위하여 Table 1의 Case I, II의 경우에 대한 SILS와 구현 결과를 Fig. 16에 도시하였다. 보다 정확한 성능 비교를 위하여 SILS의 경우 각 노드의 시작 시간을 임의로 변경하며 반복적으로 시뮬레이션을 하였다. Case I에 비하여 Case II의 경우, 모든 도립진자에 대하여 제어 입력의 크기가 작아지는

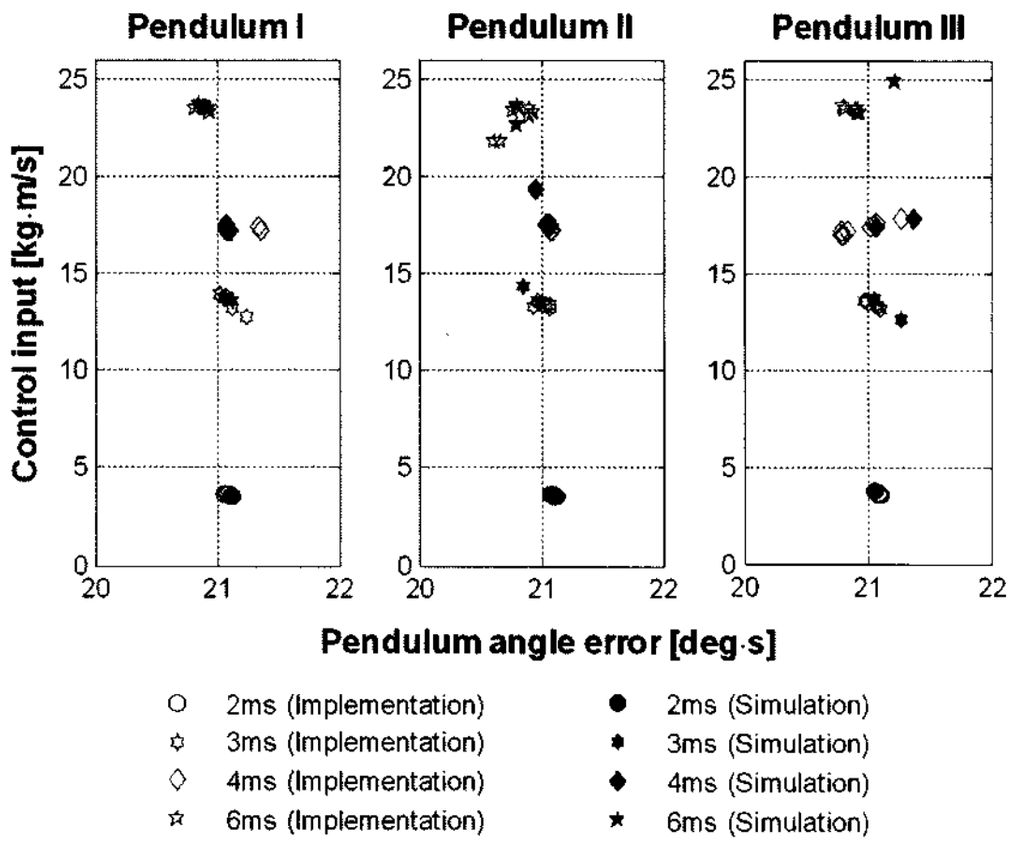

Fig. 17 Control performance versus sampling period

것을 확인할 수 있으며, 이러한 경향이 SILS의 결과와 구현결과에서 유사하게 나타남을 확인할 수 있다. 센서의 샘플링 주기에 대한 제어 성능 변화를 비교하기 위하여 Fig. 17과 같이 주기를 2ms, 3ms, 4ms, 6ms로 변경하면서 SILS와 구현 결과를 비교하였다. Fig. 16의 경우와 같이 SILS의 결과가 실제 구현 결과와 유사하게 나타남을 확인할 수 있다. 이와 같이 SILS를 기반으로 제어 로직을 설계하면, 구현후의 결과와 유사한 제어 성능을 설계 단계에서 검증할 수 있으므로, 보다 효과적인 제어 로직의 설계가 가능하다.

5. 결 론

본 연구를 통하여 실시간 운영체제와 실시간 네트워크의 특성을 고려한 SILS 환경을 제시하였으며, RCP 환경과의 연계를 통하여 설계와 구현 과정을 단일화하였다. 특히, 제시한 SILS를 RCP와 동일한 플랫폼에서 구현함으로써 기존의 연구와 차별화하였다.

제어기 설계자가 설계 단계에서 제어기 플랫폼의 환경을 용이하게 고려할 수 있는 SILS 환경을 제시함으로써, 효과적인 제어 알고리즘의 설계와 검증을 가능하게 하였다. 또한, 제안된 방법을 이용하여 설계된 제어 알고리즘으로부터 구현 소프트웨어를 자동으로 생성할 수 있는 RCP 환경을 제공하여, 소프트웨어 설계의 효율성을 높였다. 결과적으로, 제어 알고리즘 설계에서 구현까지의 일련의 개발과정을 동일한 모델과 환경으로부터 가능하도록 함으로써, 개발 과정의 단축과 모델의 재사용성을 향상 시켰다.

후 기

본 연구는 과학기술부의 국가지정연구실사업(NRL)의 지원에 의하여 수행되었다.

References

1) W. Lee, S. Park and M. Sunwoo, “Towards a Seamless Development Process for Automotive Engine-control System,” Control Engineering Practice, Vol.12, pp.977-986, 2004.
2) Z. Gu, S. Wang, J. Kim and K. Shin, “Integrated Modelling and Analysis of Automotive Embedded Control Systems with Real-Time Scheduling,” SAE, pp.115-122, 2004.
3) D. Henriksson, A. Cervin and K. Arzen, “True Time: Simulation of Control Loops Under Shared Computer Resources,” 15th IFAC World Congress on Automatic Control, 2002.
4) M. Shin, W. Lee and M. Sunwoo, “Implementation-conscious Rapid Control Prototyping Platform for Advanced Model-based Engine Control,” SAE 2003-01-0355, 2003.
5) S. Toeppe, S. Ranville, D. Bostic and Y. Wang, “Practical Validation of Model Based Code Generation for Automotive Applications,” IEEE, 1999.
6) G. Hodge, J. Ye and W. Stuart, “Multi-Target Modeling for Embedded Software Development for Automotive Applications,” 2004 World Congress SAE, 2004.
7) MathWorks, Target Language Compiler Reference Guide Ver.4, 2004.
8) M. Sunwoo, M. Shin, W. Lee and S. Han, “Development of a Body Network System with OSEK/VDX Standards and CAN Protocol,” Transactions of KSAE, Vol.10, No.4, pp.175-180, 2002.
9) K. Tindell, Real Time System by Fixed Priority Scheduling, Ph.D. Dissertation, Department of

Computer Science, University of York, 1994.
10) J. Youn, M. Shin, W. Lee and M. Sunwoo, "A Study on Timing Model and Analysis of LIN Protocol," Transactions of KSAE, Vol.13, No.6, pp.48-55, 2005.
11) The MathWorks, http://www.mathworks.com
12) Motorola, OSEK/VDX Operating System Ver. 2.1, 2000.
13) J. Ma, J. Youn, M. Shin and M. Sunwoo, "SILS/ RCP: Integrated Model based System Development Tool for Automotive Embedded Control System Design," Spring Conference Proceeding, KSAE, pp.143-149, 2004.
14) F. Lian, J. Moyne and D. Tilbury, "Network Design Consideration for Distributed Control Systems," IEEE Transactions on Control Systems Technology, Vol.10, No.2, pp.297-307, 2002.