리눅스 기반의 무인항공기를 위한 ARINC 653 프로세스 지원

(Support for ARINC 653 Processes over Linux-based Unmanned Aerial Vehicles)

한 상 현 † 이 상 헌 †
(Sang-Hyun Han) (Sang-Hun Lee)

진 현 욱 ††
(Hyun-Wook Jin)

요 약 항공 시스템에 사용되는 소프트웨어는 높은 신뢰성과 생산성이 요구된다. 이와 같은 요구로 인하여 IMA(Integrated Modular Avionics)의 파티션 OS 개념을 가진 ARINC 653 같은 항공 시스템 운영체제와 응용프로그램 간 인터페이스를 정의한 표준이 등장하였다. ARINC 653을 사용한 운영체제나 유인 항공기의 예는 많다. 하지만 아직까지 무인 항공기를 위한 리눅스 기반의 ARINC 653은 연구된 바 없다. 리눅스는 항공 ARINC 653의 요구사항을 충분히 충족시킬 수 있는 잠재력을 가지고 있다. 본 논문에서는 리눅스 기반의 ARINC 653 프로세스 모델을 위한 설계를 제안하고 초기 버전을 구현한다. 구현된 결과물을 통해 제시된 리눅스 기반 ARINC 653이 무인 항공기에 서 충분히 활용 가능함을 보인다.

키워드 : ARINC 653, 프로세스 모델, 무인항공기, 리눅스

Abstract The software running on avionic systems is required to be highly reliable and productive. Due to these demands, the standard such as ARINC 653 has been suggested, which includes the abstraction of resource partitioning and defines interfaces between avionic operating system and applications. Though there are many manned aerial vehicles employing ARINC 653 based operating systems, Linux-based ARINC 653 for unmanned aerial vehicles has not been studied yet. In this paper, we propose the design of Linux-based ARINC 653 process model and present preliminary implementation. The experiment results present that the implementation is enough to support control software of unmanned helicopter.

Key words : ARINC 653, Process model, UAV, Linux

1. 서 론

항공 시스템에 사용되는 소프트웨어는 높은 신뢰성과 생산성이 요구된다. 이와 같은 요구로 고 신뢰성과 안전성을 최우선으로 여기는 특수한 상황에 적합한 운영체제가 연구되고 있다[1]. 특히 통합 모듈 항공전자(IMA, Integrated Modular Avionics)라는 개념이 등장하여 많은 연구가 진행되었고[2,3], ARINC 653[4]과 같은 항공 소프트웨어 표준에 적용 되었다. ARINC 653은 항공 전자기기를 위한 실시간 운영체제와 그 위에서 작동하는 응용 프로그램 간의 인터페이스를 규정한다. 현재 ARINC 653은 VxWorks, LynxOS 등의 RTOS에 구현되어 항공 전자기기 및 시스템을 대상으로 적용되고 있다. 이런 ARINC 653을 적용한 시스템들은 대부분 규모가 큰 항공시스템으로 DO-178B[5] 항공기 시스템 소프트웨어 개발 공정 표준에 따라 개발되었다.

리눅스 역시 이런 항공시스템에 사용될 가능성이 충분히 높다. 현재 리눅스는 높은 안전성을 인정받아 24시간 무중단 시스템인 고성능 서버에도 많이 사용되고 있다. 또한 리눅스는 오픈소스 프로젝트이기에 사용하기 위해 별다른 로열티나 비용이 들지 않는다. 그리고 사용자가 시스템을 확장하거나 수정하기에 용이하다. 이러한 이유로 점차 리눅스를 항공기에 적용시키는 노력들이 생겨나고 있다[6,7]. 일부 무인 비행체(UAV, Unmanned Aerial Vehicles)의 제어 시스템이나 대형 항공기의 객실 시스템에 리눅스 시스템이 적용되고 있고 이런 사실들은 리눅스 시스템이 실제 항공기에 적용될 수 있음을 보여준다. 리눅스를 적용한 연구들 중에 ARINC 653을 지원하기 위한 연구는 아직까지 수행되지 않고 있으나, 리눅스 시스템이 ARINC 653표준을 지원할 수 있는 잠

· 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업(NIPA-2010-C1090-1031-0003)과 교육과학기술부의 세계수준의 연구중심대학 육성사업(WCU-2009-A419-0115)의 연구 결과로 수행되었음
· 이 논문은 2010 한국컴퓨터종합학술대회에서 '무인항공기 소프트웨어를 위한 리눅스 기반의 ARINC 653 프로세스 모델'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 건국대학교 컴퓨터공학부
whacker@konkuk.ac.kr
mir1004@konkuk.ac.kr
†† 종신회원 : 건국대학교 컴퓨터공학부 교수
jinh@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2010년 8월 12일
심사완료 : 2010년 10월 5일

재력은 충분하다. 본 논문에서는 리눅스 시스템에서 ARINC 653의 프로세스 모델을 지원하기 위한 방안을 연구한다.

본 논문은 다음과 같이 구성되어 있다. 서론에 이은 2장에서는 본 논문의 관련 연구에 대해 알아본다. 3장에서는 ARINC 653 프로세스 모델을 지원하기 위한 리눅스의 확장을 제안한다. 4장에서는 3장에서 제시한 설계를 구현한 방법과 이를 사용하여 실험한 실험결과를 보여준다. 마지막 5장에서는 본 논문의 결론과 향후 계획에 대하여 기술한다.

2. 관련연구

본 장에서는 관련 연구에 대해서 논한다.

2.1 관련연구

항공기에 ARINC 653을 적용한 예는 많이 존재한다. ARINC 653 표준 규격을 만족하는 VxWorks 653[8]은 보잉 787 드림라이너를 포함한 40가지 이상의 기체에서 180개 이상의 서브시스템에 적용되어 사용되고 있다. 그리고 미국의 전폭기인 JSF(Joint Strike Fighter)에 사용된 Green Hill의 INTEGRTITY, 보잉 777기에 사용된 LYNUX WORKS의 LynxOS-178 RTOS와 Lynx OS-SE RTOS 등이 있다. 하지만 이런 운영체제들을 사용하기 위해서는 많은 로열티를 지불해야 하며 커널의 구성과 확장을 쉽게 수행할 수 없는 문제가 있다. 이런 점을 해결하기 위하여 다양한 응용프로그램과 디바이스 드라이버를 제공하면서 무료로 사용할 수 있는 리눅스 커널을 이용하여 항공기를 제어하기 위한 연구들이 진행되어 왔다[6,9]. 이러한 연구는 주로 무인비행체를 위해 작고 경량화된 기체를 제어하기 위한 용도로 연구되었다. 하지만 ARINC 653과 같은 표준 인터페이스를 지원하지 않아 항공 소프트웨어의 재사용이 용이하지 않고 다른 소프트웨어와 호환성을 유지하기 힘들다. ARINC 653의 통신 인터페이스를 리눅스에 구현하는 연구는 현재 진행 중이다[7]. 연구는 ARINC 653의 Inter-Partition Communication의 Queueing mode와 Sampling mode를 지원하며 임베디드 리눅스 기반 경량 프로토콜[10]을 이용하여 리눅스 환경에서 지원한다.

지금까지 살펴본 것과 같이 현재까지 리눅스 시스템 기반의 ARINC 653 프로세스 모델을 지원하는 연구는 이루어지지 않았다. 이러한 면에서 본 논문의 연구는 기존의 연구와 차별화 된다고 할 수 있다.

3. 리눅스 기반의 ARINC 653 프로세스 지원

본 장에서는 리눅스 시스템에서 ARINC 653 프로세스를 지원할 수 있는 방안을 제안한다.

3.1 전체구조

본 논문에서 제안하는 설계는 그림 1과 같은 형태를 가지고 있다. 리눅스 환경에서 ARINC 653 표준을 지원하기 위하여 커널 영역에 ARINC 모듈, Earliest Deadline First(EDF) 스케줄러를 추가하였다. 그리고 사용자 영역에 ARINC 653 라이브러리와 StartUP 프로세스를 구성하였다. ARINC 커널 모듈은 시스템에서 수행될 파티션 정보와 프로세스들의 전체 정보를 가지고 있다. 그리고 파티션에 포함되어 있는 프로세스들의 주기와 실행시간 값을 EDF 알고리즘으로 계산하여 프로세스가 실행될 수 있는지 여부를 판단한다. EDF 스케줄러는 하나의 파티션에 속해 있는 프로세스의 스케줄링을 담당한다. EDF 스케줄러를 선택한 이유는 구현이 쉽고 동적 프로세스 삽입이 용이하기 때문이다.

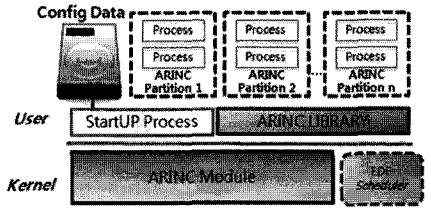

그림 1 리눅스 기반 ARINC 653 프로세스 지원

사용자 영역의 ARINC 653 라이브러리는 ARINC 653 표준 라이브러리를 프로세스들에게 제공한다. 그리고 라이브러리가 제공하는 API를 통해 프로세스들은 커널의 ARINC 모듈과 통신할 수 있다. 이를 통해 사용자 영역의 파티션 정보나 프로세스 정보를 커널 영역의 ARINC 모듈에 전달이 가능하게 된다. ARINC 653은 파티션 생성과 파티션 주기, 파티션 수행시간, 파티션 이름 등 많은 정보들을 시스템 개발자가 미리 설정하도록 정의하고 있다. 따라서 이런 정보들을 읽어 들여 실제로 파티션을 생성하고 스케줄러에서 알 수 있도록 전달해 주는 프로세스가 필요하다. 이를 위해 StartUP 프로세스는 파티션의 생성, 환경설정파일 로딩 등 일련의 작업을 담당한다.

3.2 파티션 생성

ARINC 653의 규격에 따르면 모든 파티션은 시스템이 구동될 때 미리 생성된다. 그리고 파티션의 이름, 실행주기 등 환경 변수들은 전체 시스템의 구성자가 시스템이 구동되기 전에 미리 설정하도록 규정하고 있다. 따라서 ARINC 653의 파티션은 시스템이 수행되고 있는 시간에 동적으로 할당이 불가능하며 모든 파티션의 정보와 파티션들의 요구사항을 미리 알 수 있다. 이와 같은 특징을 가지고 있는 ARINC 653의 파티션을 리눅스 환경에서 구성하기 위해 StartUP 프로세스와 ARINC 커널 모듈이 파티션을 생성한다. 파티션 생성 과정은 시

스템의 구동 시 한번만 이루어지며 시스템이 운용되는 중간에 동적으로 변경이 불가능하다. 시스템 구성자가 파티션들의 정보를 파일로 작성한다. 그 후 시스템이 구동될 때 StartUP프로세스가 설정파일을 읽어 들여 파티션에 관한 정보를 획득한다. 획득한 파티션 정보를 ARINC 653 라이브러리를 통해 커널에 존재하는 ARINC 모듈에 전달하고 설정파일에 저장되어 있던 정보들이 이상이 없으면 ARINC 커널 모듈은 StartUP 프로세스에게 전달받은 데이터들을 저장한 후 StartUP 프로세스에게 파티션을 생성해도 좋다는 결과를 전달한다. 마지막으로 StartUP 프로세스는 config 파일에 정의된 파티션 정보에 따라 ARINC 파티션을 생성한다.

3.3 프로세스 생성

ARINC 653은 프로세스를 생성하고 실행할 때 2개의 함수가 호출되며 전자는 CREATE_PROCESS이며 후자는 START 함수이다. CREATE_PROCESS 함수는 프로세스가 생성될 수 있는지 확인한다. 그 후 프로세스가 생성될 수 있는 환경이면 ARINC 653에서 사용하는 PID (ARINC PID)를 반환하고 Context와 스택을 리셋한다. 그리고 반환된 PID를 이용하여 START 함수를 호출하여 실제 프로세스를 생성한 후 실행시킨다.

그림 2는 ARINC 653 프로세스를 생성하기 위해서 본 논문이 제시한 단계를 표현한다. 먼저 생성 할 프로세스의 설정파일을 파티션이 읽어 들여 ARINC 653 라이브러리에 존재하는 CREATE_PROCESS함수를 호출한다. 그 후 ARINC 653 라이브러리는 전달받은 프로세스 정보를 ARINC 모듈에게 전달한다.

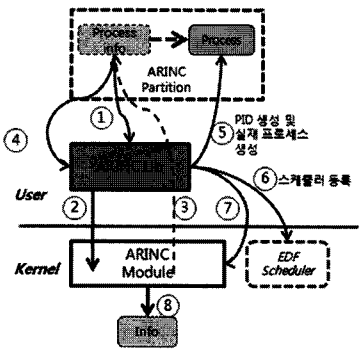

그림 2 프로세스 생성 과정

이때 전달하는 정보에 프로세스의 실행 주기, 우선순위 등 프로세스를 생성하기 위한 데이터가 포함되어 있다. 이를 통해 ARINC 653모듈은 생성할 프로세스가 주기와 마감시간을 지켜 스케줄링이 가능한지 등을 판단하고 ARINC PID와 결과를 반환한다. 그 후 파티션은 프로세스를 실행시키기 위해 START함수를 호출한다. START함수는 ARINC 653 라이브러리에 정의되어 있으며 함수가 호출되면 리눅스 시스템 상에서 실제 프로세스를 생성한다. 이 과정에서 리눅스에서 사용되는 PID를 얻고 ARINC PID와 맵핑하여 스케줄러에 등록하는 과정을 수행한다. ARINC 653 라이브러리는 스케줄러에 PID를 등록하고 커널 모듈에게 ARINC PID와 프로세스의 정보를 커널모듈에 저장하고 리스트 형으로 관리하도록 지시한다. ARINC 모듈에 저장된 ARINC 653 프로세스 정보들은 EDF 스케줄러가 프로세스를 스케줄링 할 때 이용된다.

3.4 스케줄링

ARINC 653의 스케줄링을 위한 구조와 요구사항은 일반 리눅스와는 다르다. 이는 파티션과 프로세스의 주기적 실행, 마감시간 존재 등의 이유로 기인한다. 따라서 주기와 마감시간을 인지할 수 있는 EDF 스케줄러를 리눅스에 추가한다. 또한 2.1절에서 설명된 바와 같이 ARINC 653은 파티션에 대한 스케줄링을 필요로 한다. 그러므로 ARINC 653의 프로세스 스케줄러보다 상위에 존재하는 파티션 간 스케줄링을 담당하는 스케줄러에 대한 고려가 필요하다.

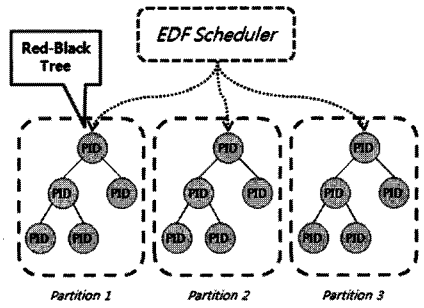

그림 3 프로세스 스케줄링

그림 3은 본 논문에서 제안하는 스케줄링 구조를 나타낸다. EDF 스케줄러는 프로세스가 생성되면 ARINC 653 커널 모듈에 존재하는 정보들을 토대로 파티션 당 하나의 Red-Black Tree를 구성한다. Tree를 구성할 때 이를 위한 추가적인 메모리 할당이나 데이터 복사는 거의 일어나지 않는다. 이를 가능케 하는 이유는 ARINC 653 커널 모듈이 모든 프로세스 정보와 파티션 정보를 커널에 리스트 형으로 저장하고 있기 때문이다. 하나의 파티션 안에서 프로세스들을 EDF 알고리즘을 통해 스케줄링 하다가 해당 파티션의 주기가 끝나면 다음 파티션의 Red-Black Tree의 헤더를 포인팅 한다. 이렇게 되면 EDF 스케줄러는 항상 하나의 Red-Black Tree만을 보게 되고 파티션이 스케줄링 될 때 마다 새롭게 Red-Black Tree를 구성하는 추가적인 오버헤드 없이

자연스럽게 다른 파티션의 프로세스들을 스케줄링 할 수 있다. 본 논문에서는 연구의 초기단계로 프로세스 간 스케줄링에 한하여 구현한다. 초기 버전으로 구현한 설계가 단일 파티션이지만 4.3절에서 제어 위주의 간단한 무인비행체를 지원하는 것에 큰 무리가 없음을 보인다.

프로세스가 생성되고 실행될 때 프로세스가 정확한 주기와 마감시간을 준수하지 못할 수 있다. 이와 같은 시스템의 오류를 확인하고 회복하기 위해서 ARINC 653은 Health Monitor(HM)를 제공한다. HM은 지정된 API를 통해 오류내용을 사용자 영역으로 전달하거나 오류를 대체할 수 있는 방안을 사용자가 정의할 수 있도록 한다.

4. 구현 및 성능측정

본 논문에서는 연구의 초기 단계로 하나의 파티션만을 지원하는 ARINC 653 프로세스 모델을 구현하였다. 그리고 파티션 안의 프로세스 스케줄링을 위한 EDF 스케줄러로 EVIDENCE[11]사의 SCHED_DEADLINE을 사용하였다. 구현 대상 시스템은 Ubuntu 리눅스 (커널 버전 2.6.32.7)를 사용하였다. 구현된 소프트웨어는 다음과 같이 구성되어 있다.

StartUP: 시스템 구동 시 정적으로 설정된 config파일을 읽어 커널 모듈에 올려주는 역할을 한다.

ARINC_LIB: ARINC 653의 API가 구현되어 있다. 각 API안에는 SCHED_DEADLINE에서 사용되는 시스템 콜이 호출되고 프로세스 구조체와 스케줄러에서 사용될 프로세스 정보를 매핑 한다. 그리고 HM의 오류 상황을 확인하고 적합한 핸들러를 등록할 수 있는 API도 제공된다.

ARINC_MODULE: ARINC 모듈은 파티션과 프로세스의 정보를 저장하고 이것들이 시스템에 실행될 수 있는지를 검사하는 역할을 한다. 먼저 모듈이 실행되면 모든 파티션의 정보를 저장할 수 있는 리스트를 생성한다. ARINC 653에서 규정하고 있는 최대 파티션의 개수는 32개 이므로 모든 파티션에 대해 저장 공간을 미리 설정해도 큰 오버헤드가 되지 않으며 미리 저장 공간을 설정해 놓음으로 인해 config 파일에 설정된 파티션들이 시스템에 생성 가능한지 여부를 미리 알 수 있다. 그리고 프로세스가 실행될 때 프로세스의 오버런을 체크하기 위해 리눅스의 hrtimer(High-Resolution Timer)를 프로세스 예상 종료 시간으로 설정한다. 그리고 프로세스가 주기 내에서 종료될 때 설정한 hrtimer를 해제한다. 이렇게 함으로써 hrtimer 이벤트가 발생하면 오버런이 발생한 것을 알 수 있다.

성능 측정은 Intel Mobile 1.4GHz를 장착하고 있는 ADVENTECH PCM-9584 산업용 임베디드 보드에서 수행했다.

4.1 수행 가능 최소 주기

본 논문에서 구현한 리눅스 기반 ARINC 653 프로세스 설계가 지원할 수 있는 최소 실행 가능 주기를 측정하기 위하여 다음과 같은 실험방법을 사용하였다. 하나의 ARINC 653 프로세스를 생성하여 프로세스가 실행되는 주기를 변경해 가며 실험을 진행하였다.

그림 4는 최소 실행 가능 주기 측정 결과를 보여준다.

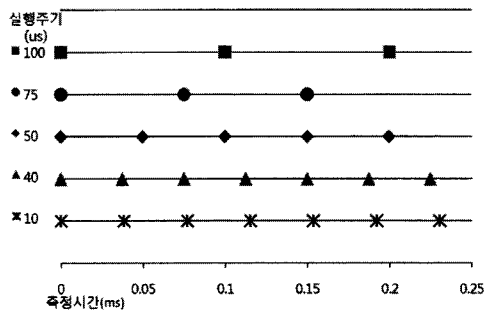

그림 4 최소 실행 가능 주기 측정 결과

가로축은 실험을 진행한 시간을 나타내며 ms의 단위를 가지고 세로축은 프로세스가 실행된 주기를 나타내며 us의 단위를 가진다. 실험의 결과를 보면 50us까지 프로세스는 정상적인 주기에 수행되며 실행주기 마다 약 0.7us~2.4us 사이의 평균 오차를 보인다. 하지만 그 미만의 주기부터는 프로세스의 실행 주기가 정확하게 지켜지지 않는다. 따라서 본 논문에서 제시한 구현은 측정 환경에서 프로세스의 주기를 50us의 시간 해상도까지 지원해줄 수 있음을 알 수 있다.

4.2 수행시간 정확도 분석

본 논문에서 제시한 ARINC 653 프로세스 구현이 어느 정도의 정확성을 가지는지 알아보기 위해서 100us의 주기와 50us의 주기별 수행시간으로 설정된 프로세스를 수행한다. 그리고 인위적으로 수행시간을 초과시킬 경우 프로세스가 시작한 시간을 측정하고 오버런을 체크하기 위한 타이머가 호출된 시간을 측정한다.

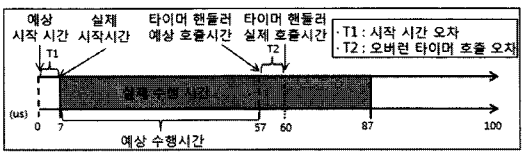

그림 5 수행시간 정확도 분석

그림 5는 측정한 결과이다. 시간 0은 프로세스가 시작되길 기대하는 예상 시작 시간이다. 그림의 T1은 이 예상 시작 시간과 프로세스의 실제 시작 시간의 차이로서

약 7us을 보인다. 그리고 실제 프로세스의 실행시간인 50us이 지난 57us에 오버런 타이머가 호출되길 기대 했으나 3us가 지난 60us에 오버런 타이머가 인지되었다. 이것은 오버런 타이머의 오차시간 T2는 약 3us 정도임을 의미하며, 이는 매우 작은 값으로 판단된다. 이것은 HM이 프로세스의 오버런이 발생할 경우 매우 빠르게 프로세스의 오버런을 인지할 수 있다는 것을 보여준다. 무인 비행체에서 수행되는 프로세스의 주기가 10ms 이상인 것을 고려할 때 본 논문에서 제안한 ARINC 653 프로세스 구현의 수행시간 정확도는 매우 높다고 볼 수 있다.

4.3 무인헬기 제어 소프트웨어

ARINC 653은 항공기를 위한 표준이다. 때문에 실 항공기에 적용된 소프트웨어를 사용하여 항공기에 적용된 소프트웨어 주기를 설정하여 실험을 진행하였다. 실험에서 참고한 프로세스의 주기 및 작업은 한국항공우주연구원에서 개발된 무인헬기[12]에 탑재된 항공제어 소프트웨어를 참고하였다. 이 소프트웨어는 5개의 프로세스를 가지고 있다. 20ms의 주기를 가지고 실행되는 항법 프로세스와 100ms의 주기를 가지고 센서에서 시리얼포트를 통해 데이터를 읽어 들이는 센싱 프로세스 4개로 구성되어 있다. 이 프로세스들을 실험용 ARINC 653 프로세스로 변경하여 실험을 진행하였다.

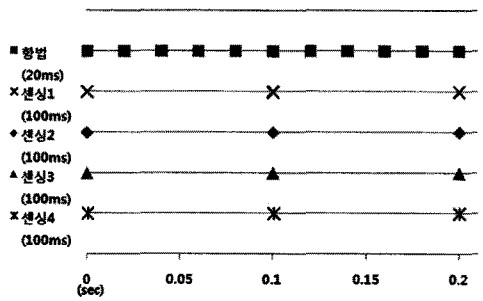

그림 6 실 항공 소프트웨어 시나리오 적용 실험 결과

그림 6은 실험 결과를 보여준다. 그래프를 보면 각 프로세스의 실행주기가 정확히 준수되는 것을 볼 수 있다. 이를 통해 본 논문에서 제시한 ARINC 653을 위한 프로세스 지원이 실제 무인 항공기에 적용 가능함을 알 수 있다.

5. 결론 및 계획

본 논문에서는 항공 소프트웨어 표준인 ARINC 653의 프로세스 모델을 리눅스 환경에서 효과적으로 지원할 수 있는 설계를 제안했다. 그리고 초기 단계로 실제 리눅스환경에 구현하고 이것의 성능을 측정하였다. 측정결과 구현된 결과물은 충분히 무인 항공기에 활용 가능함을 보였다. 향후 여러 개의 파티션을 지원하고 효율적인 파티션 간 스케줄링 기법을 제공할 수 있는 방안을 연구할 계획이다. 아울러 HILS에 적용할 계획이다.

참 고 문 헌

[1] 김대영, 도윤미, 김주홍, 조혜영, 성종우, "실시간 운영체제 연구 개발 동향", *전자공학회지*, 제29권, 9호, pp.78-86, 2002.

[2] "Design Guidance for Integrated Modular Avionics," ARINC report 651, Aeronautical Radio Inc, Annapolis, MD, 1991.

[3] D. Kim, Y.-H. Lee and M. Younis, "SPIRIT-mKernel for Strongly Partitioned Real-Time Systems," *IEEE Real-Time Computing Systems and Applications*, 2000.

[4] Airlines Electronic Engineering Committee, "ARINC SPECIFICATION 653 1-2," 2005.

[5] http://ko.wikipedia.org/wiki/DO-178B

[6] R. Nardi and O. Holland, "UltraSwarm: A Further Step Towards a Flock of Miniature Helicopters. Swarm Robotics," *Lecture Notes in Computer Science*, vol.4433, 2006.

[7] S.-H. Lee, H.-W. Jin and K-Y. Bang, "Communication Support for Collaborative Embedded Controllers in Unmanned Aerial Vehicles," International Conference on Intelligent Unmanned Systems, 2010.

[8] http://www.windriver.com/

[9] 김명현, 문승빈, 홍성경, "리눅스 기반 무인항공기 제어 애플리케이션 개발", 제어·로봇·시스템학회논문지 제12권 제1호, 2006.

[10] S.-H. Lee and H.-W. Jin, "Communication Primitives for Real-Time Distributed Synchronization over Small Area Networks," *In Proc. of IEEE ISORC 2009*, 2009.

[11] http://www.evidence.eu.com/content/view/313/390/

[12] 김성필, 이장호, 김봉주, 권형준, 김응태, 안이기, "위치예측에 기반한 무인헬기 점항법 유도법칙 개발", *항공우주기술*, vol.5, no.2, pp.1-7, 2006.