Fuel Cell System용 내장형 제어기의 소프트웨어 플랫폼 개발

임채홍*(1), 김진우(1), 이우택(2)
(1)국립 창원대학교 대학원, (2)국립 창원대학교 제어계측공학과

Development of Software Platform of Embedded Controller for Fuel Cell System

Chaehong Lim*(1), Jinwoo Kim(1), Wootaik Lee(2)
(1)Graduated School Changwon National University
(2)Dept. of Control and Instrumentation Engineering Changwon National University

Abstract - This paper describes the development of software platform of embedded controller for Fuel Cell System. The fuel cell system is complex which needs an embedded controller to execute multiple tasks. The software organized by modualarization and layered architecture can perform complicated control algorithms. By development of the software platform with architectural software, the fuel cell system's embedded controller has a reusability and a scalability. And the developed software platform guarantees a execution of multiple tasks.

1. 서 론

에너지 문제와 환경 문제로 인하여 고효율 시스템과 신재생에너지 시스템에 관한 연구가 활발히 진행되고 있다. 특히 연료전지 시스템은 높은 에너지 효율과 배출가스가 거의 없다는 장점을 가지고 있어 활발한 연구가 진행되고 있다. 고용량의 출력과 장시간 사용이 가능하고 폐열을 이용한 열병합 발전이 가능한 점으로 볼 때 연료전지 시스템은 작은 발전 시스템으로 간주될 수 있다. 연료전지 시스템의 구성은 크게 연료전지 스택에 수소를 공급하여 전기를 생성하는 부분과 생성된 전기를 변환하는 부분으로 나누어지고, 이를 바탕으로 많은 연구 개발이 이루어지고 있다.

전기를 생성하는 부분에서는 연료전지 스택에 공급할 수소를 생산하는 개질기 부분과 연료전지 스택이 전기를 생산할 수 있는 환경을 만들어 주를 부분으로 구성이 된다. 전기를 변환하는 부분에서는 연료전지에서 나오는 저전압 고전류의 출력을 상용전원으로 변환하는 PCS(Power Conversion System)로 구성된다. 연료전지 시스템을 제어하기 위하여 100여개에 달하는 온도, 습도, 압력, 유량 등의 센서를 통하여 시스템의 상태를 파악하며 유량제어기, 송풍기, 압축기, 모터 등의 제어를 통하여 시스템의 출력을 결정하게 된다. 연료전지 시스템의 개발 단계에서는 많은 데이터의 입출력으로 인하여 PC와 PLC를 기반으로 하는 제어 시스템을 사용하고 있다. 하지만 MCU를 사용하여 내장형 제어시스템을 구현한다면 시스템의 크기를 줄이고 보다 안정된 시스템을 구현할 수 있을 것이다.

기존의 MCU를 사용한 내장형 제어시스템은 간단한 제어 알고리즘을 수행하기 위하여 사용되어왔다. 연료전지 시스템의 경우에는 여러 아날로그와 디지털 입출력 신호를 가지며 시스템의 입력에서 출력을 얻기까지 많은 제어루프를 가지는 복잡한 시스템이다. 연료전지 시스템처럼 복잡한 내장형 제어 시스템을 구현하려면 고성능의 MCU가 필요하고 복잡한 제어 알고리즘을 효과적으로 제어하고 시스템의 성능을 최적화 하려면 잘 구조화된 소프트웨어가 필요하다. 구조화된 소프트웨어는 모듈화를 통하여 하드웨어 독립적인 소프트웨어를 구현하여 재사용성을 증가시키고, 스케줄러를 이용하여 복잡한 제어 알고리즘들을 효과적으로 수행할 수 있으며 시스템 변경에 최소한의 영향을 받도록 구성된다[2]. 연료전지 시스템용 내장형 제어기의 소프트웨어 플랫폼을 구조화된 소프트웨어를 사용하여 구현하고 그 플랫폼을 사용하여 제어알고리즘을 구현하도록 하였다. 본론에서는 구현된 소프트웨어 플랫폼의 개념과 구조에 대하여 설명하겠다.

2. 본 론

2.1 MCU 선정

연료전지 시스템은 개질기에서 연료전지 스택에 이르기까지 많은 온도, 압력, 습도 등의 센서들이 존재한다. 연료전지 시스템의 제어기는 많은 센서 입력을 받아 제어 연산을 수행한 후 송풍기, 압축기, 모터를 구동하기 위한 출력을 내보내야 한다. PC기반의 제어시스템이라면 큰 문제가 되지 않겠지만 내장형 제어 시스템을 구현한다면 입출력 채널의 개수, 내부 디바이스들의 구성 및 성능, 외부 디바이스로의 확장성, 제어 알고리즘의 수행을 위한 연산능력 등을 고려해야만 한다.

Freescale에서 출시된 MPC5554는 이러한 조건을 만족시키기에 충분하다. PPC(PowerPC) Core를 이용한 32bit MCU인 MPC5554는 충분한 입출력 채널의 개수, 스마트한 내부 디바이스, 외부 디바이스로의 확장성, 큰 메모리 용량 등으로 인하여 연료전지 시스템의 내장형 제어기로 적합하다. <표 1>에서는 MPC5554의 특징을 나타내었다.

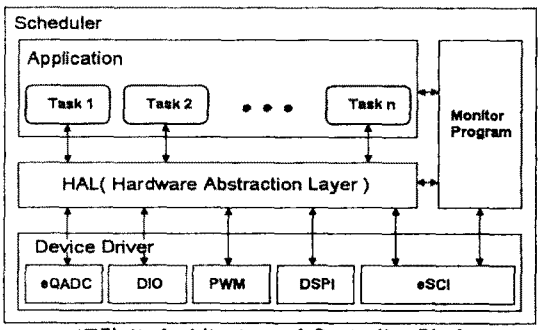
MPC5554는 64k byte의 SRAM과 2Mbyte의 내부 플래시를 가지고 있으며 EBI(External Bus Interface)를 이용하여 외부 메모리의 추가도 가능하다. 부동 소수점연산을 지원하여 정밀도가 높은 연산이 가능하다. MCU선정의 기준이 되었던 아날로그 입출력 채널수의 경우에는 QADC 모듈에서 총 40채널을 사용할 수 있으며 MUX를 사용하거나 외부 ADC 모듈을 SPI 통신을 이용하여 사용함으로써 100 채널 이상의 아날로그 입출력 채널 사용도 가능하다. eTPU와 eMOIS 모듈을 이용하여 PWM 파형 등의 타이밍에 관련된 신호를 생성 할 수 있으며 SCI, SPI, Flex CAN 모듈을 이용한 외부 디바이스와의 통신으로 확장성이 높은 장점도 가진다. 이러한 특징으로 개발 단계에서의 변동사항, 향후 시스템 발전에 따른 확장성을 고려할 때 MPC5554를 사용하여 연료전지 시스템의 제어기를 개발하는 것은 적합하다.

<표 1> MPC5554 Features

Feature		MPC5554
CPU	Core	e200z6 40~132Mhz
	Cache	64k
	MMU	32entry
	Crossbar	3×5
eDMA		64 channels
Memory	SRAM	64k
	Flach	2M
	eTPU/ParameterRAM	16k/3k
Communication I/O	SCI	2
	Flex CAN	3
	DSPI	4
Timed I/O	eMIOS	24 channels
	eTPU	64channels
Analog I/O	QADC	40channels

2.2 Software Architecture

연료전지 시스템의 제어기 플랫폼을 개발함에 있어 가장 중요한 것은 연료전지 시스템의 복잡한 제어 알고리즘을 효율적으로 구현하게 하는 것에 있다. 모듈화 된 디바이스 드라이버는 소프트웨어의 재사용성을 증가 시키고 확장성을 가진다. 하드웨어에 독립적으로 구현된 응용계층과 모듈화된 디바이스 드라이버는 개발과정에서의 설계 변경에 의한 소프트웨어 플랫폼의 변경을 최소화시킨다. 구조화된 소프트웨어 개념을 도입하여 내장형 시스템의 제어기 플랫폼을 구성함으로써 복잡한 제어 알고리즘이 효과적으로 구현하게 하였고 시스템 개발 후 검증도 용이하게 하였다. <그림 1>에서 연료전지 제어기 플랫폼의 소프트웨어 구조를 나타내었다.



<그림 1> Architecture of Controller Platform

각각의 디바이스 드라이버와 태스크들을 서로 의존성이 없도록 모듈화하고 이를 계층화하여 구성한 후 스케줄러를 사용하여 태스크들의 수행시간과 주기를 통괄하도록 구성하였다. HAL(Hardware Abstraction Layer)을 구성하여 디바이스 드라이버와 어플리케이션 사이에서 데이터를 추상화 하여 주고받을 수 있도록 구현을 하였고 SCI통신을 이용하여 PC와의 인터페이스를 통한 모니터 프로그램을 구성하였다.

<그림 1>과 같은 구조화된 소프트웨어 플랫폼에서 데이터의 흐름은 다음과 같다. 하드웨어에서 입력받는 신호는 디바이스 드라이버를 통하여 HAL로 전달된다. HAL에서는 디바이스 드라이버에서 입력받은 신호를 응용계

층에서 사용하기 쉽게 추상화하여 응용계층의 태스크로 전달한다. 예를 들면 eQADC를 통하여 하드웨어에서 5V의 입력을 받으면 HAL에는 ADC 분해능에 따른 16진수의 데이터를 입력받게 된다. 이 데이터를 응용계층으로 전달 할 때 HAL에서 다시 5V값으로 데이터를 변환하여 제어 알고리즘 내에서 사용하도록 한다. 반대의 경우로 PWM신호를 출력하고자 할 때 제어 알고리즘에서 0~100%사이의 듀티 값을 HAL로 전달하고 HAL에서는 PWM디바이스 드라이버에서 사용할 수 있는 16진수로 변환하여준다. PWM디바이스 드라이버에서 변환된 데이터를 사용하여 PWM신호를 생성하게 된다.

2.2.1 Device Driver

MCU와 외부의 하드웨어와 인터페이스는 디바이스 드라이버를 통하여 이루어진다. 디바이스 드라이버는 각 내부 주변장치 별로 모듈화 되며 각 디바이스 드라이버는 서로 독립적으로 구현된다. 모듈화 된 디바이스 드라이버는 재사용성과 확장성이 증가하고 시스템의 하드웨어 의존도를 낮춰준다. 그러므로 개발과정에서 발생하는 설계변경에 의한 소프트웨어 플랫폼의 변화를 최소화 할 수 있다. <표 2>는 디바이스 드라이버의 구성을 나타내었다.

<표 2> Device Driver의 구성

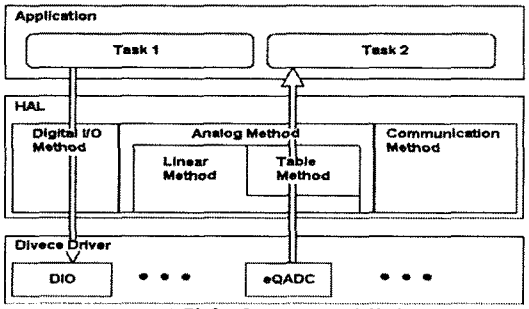
Classification	function
Member function	API(Application Programming Interface)
Attribute	Configuration

각 디바이스 드라이버는 크게 환경설정 부분과 API 부분으로 나누어진다. 환경설정에서는 디바이스 드라이버의 기능을 설정한다. API는 응용계층에서 사용하고자 하는 디바이스 드라이버의 기능을 구현하였다. 디바이스 드라이버는 기능에 적합한 자료구조를 가지고 내부적으로 하드웨어에 의존하는 함수를 가지게 된다.

이러한 구조의 디바이스 드라이버는 하드웨어의 변화에 따라 디바이스 드라이버의 API에는 변화가 없이 내부적으로 하드웨어에 의존하여 변하게 된다. API를 통하여 디바이스 드라이버와 연결되는 HAL과 응용계층에는 하드웨어의 변화가 주는 영향이 없게 된다.

2.2.2 HAL(Hardware Abstraction Layer)

HAL은 응용계층과 디바이스 드라이버 사이에 위치하여 데이터의 추상화를 담당한다. 또한 독립된 디바이스 드라이버 간의 데이터 전송은 HAL을 통하여 이루어지게 된다. HAL에서는 API를 통하여 디바이스 드라이브에서 가져온 데이터를 응용계층에서 쉽게 사용할 수 있도록 의미를 부여하고 데이터를 가공한다. <그림 2>에서는 HAL의 구조에 대하여 나타내었다.

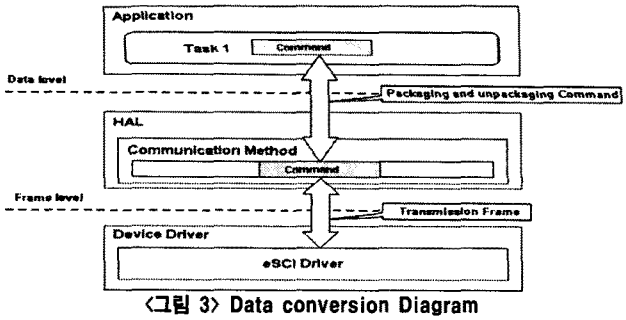


<그림 2> Structure of HAL

연료전지 시스템용 소프트웨어 플랫폼에서서의 HAL은 데이터의 성격에 따라 크게 세 가지로 구분될 수 있다. 디지털 입출력 데이터, 아날로그 입출력 데이터, 통신 프레임이 그것이다. 디지털 입출력 데이터는 간단한 스케일링이나 명명에 의하여 데이터에 의미를 부여 할 수 있다. 아날로그 입출력 데이터는 내부적으로 선형적인 변환을 하는 방법과 비선형적인 변환을 고려하기위한 테이블을 이용한 방법으로 나누어진다. 아날로그 입출력 데이터는 외부 하드웨어에 따라 데이터 특성이 선형적인지 비선형적인지 결정이 되므로 이를 모두 고려하기 위하여 위의 두 방법을 사용하여 응용계층과 데이터를 주고받을 수 있게 하였다. 선형적인 변환은 아날로그 입출력 신호를 <수식 1>과 같이 간단한 선형 보간법을 통하여 데이터를 변환하여 응용계층에서 사용할 수 있도록 하였다. 테이블을 이용한 방법은 하드웨어에 따른 아날로그 신호의 입출력 특성을 테이블로 작성하고 두 인덱스 사이의 값은 선형보간법을 사용하여 수치해석적으로 계산하도록 구현하였다.

$$Y = aX + b \qquad \text{<수식 1>}$$

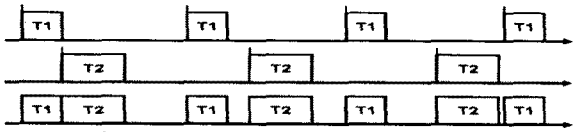
통신 디바이스와 데이터를 주고받기 위해서는 프로토콜에 맞는 데이터 프레임이 필요하다. <그림 3>에서 나타낸바와 같이 HAL에서는 응용계층에서 받은 데이터를 통신 프로토콜에 맞게 프레임을 만들어 디바이스 드라이버로 전송하고 반대로 디바이스 드라이버에서 전송받은 프레임에서 데이터를 분리하여 응용계층으로 보내는 역할을 한다. 응용계층에서는 통신 디바이스에 상관없이 데이터만 전송하므로 통신 프로토콜에 의존하지 않게 된다.



<그림 3> Data conversion Diagram

2.2.3 Scheduler

복잡한 제어 알고리즘을 가지는 연료전지 시스템에서 주기적으로 수행 되어야하는 제어 알고리즘의 수행 시간을 보장하기 위하여 스케줄러를 구현하였다. 스케줄러는 여러 개로 구성된 태스크들을 일련의 수행방법에 따라 순차적으로 수행시키는 역할을 한다. 각 태스크는 서로 독립적으로 수행하도록 설계 되어야 스케줄러를 이용하여 효과적으로 수행이 가능하다. 스케줄러를 사용하면 연료전지 시스템의 복잡한 제어 알고리즘들을 효율적으로 수행 할 수 있다. <그림 4>에서 순차적으로 수행되는 태스크들을 나타내었다.



<그림 4> Take Sequence of Cyclic Scheduler

연료전지 시스템에서 주기적으로 수행 되어야하는 여러 태스크들의 수행을 보장하기위해 Cyclic Task Scheduler를 구현하였다. Cyclic Task Scheduler는 크게 두 가지 방법으로 디자인될 수 있다. 하나는 태스크들의 수행주기와 우선순위를 이용하여 우선순위가 높은 태스크를 순차적으로 수행하는 방법이고 다른 하나는 각 태스크들의 수행시간과 시간 마진 을 계산하여 시간 축 위에서 태스크들의 나열하고 수행하는 방법이다. Cyclic Task Scheduler는 두 개의 모트를 사용할 수 있도록 설계 되었다. 모드의 변환은 연료전지 시스템에서 수행되는 태스크들의 수행주기와 우선순위 등의 특성을 바꿈으로써 시스템의 상태에 능동적으로 대처할 수 있다. 예를 들어 연료전지 시스템의 정상동작 상태에서 필요한 태스크들과 비상 상태시에 필요한 일련의 태스크들을 모드 변환을 통하여 수행할 수 있어 비상 상태에 대해 유연하게 시스템의 제어 알고리즘을 수행 할 수 있다.

2.2.4 Monitor Program

시스템 운전과정에서 필요한 정보를 모니터링하기 위하여 MCU와 호스트 PC와의 인터페이스 환경을 구축하였다. Freescale에서 제공하는 FreeMaster 플랫폼을 기반으로 연료전지 시스템의 내장형 제어기에 적합한 MPC5554에 맞게 수정하여 사용하였다.

3. 결 론

시스템의 제어기를 개발과정에서 간단한 구조의 내장형 시스템은 8-bit MCU를 사용한 내장형 제어기를 구현하고, 복잡한 구조의 시스템의 경우에는 PC를 기반으로 하는 제어 시스템을 구현한다. 하지만 연료전지 시스템의 경우는 시스템은 복잡하면서도 내장형 제어 시스템이 필요하기 때문에 앞선 두 제어 시스템의 장점이 모두 필요하다. 게다가 아직 시스템 개발 초기 단계이기 때문에 앞으로 많은 설계 변경이 이루어진다는 것도 제어 시스템을 선택함에 있어 중요한 변수이다. 이러한 문제를 고성능의 MCU를 이용하여 내장형 제어 시스템의 플랫폼을 만들고 그 위에 제어 알고리즘을 구현한다면 앞의 문제점을 해결할 수 있다.

연료전지 시스템용 내장형 제어 시스템을 구현하기위하여 소프트웨어를 기능에 따라 모듈화하고 계층화를 이루어 구조적 설계를 함으로써 연료전지 시스템에 적합한 내장형 제어기의 소프트웨어 플랫폼을 구현하였다. 소프트웨어의 구조적 설계는 복잡해지는 소프트웨어의 개발을 용이하게 하고 확장성과 재사용성을 키우며 시스템에 대한 검증이 편리다. 내장형 제어기의 소프트웨어 플랫폼에 스케줄러를 사용하여 제어 알고리즘의 수행을 보장하였다.

[참 고 문 헌]

[1] Jane W.S. Liu, Real-Time Systems, Prentice Hall, p.85~114, 2000

[2] Joonwoo Son, Wootaik Lee, Ivan Wilson, "Model Based Design System Development for In-Vehicle Network System", SAE, Electronics, p. 2006

[3] MPC5553_MPC5554 Reference Manual

[4] www.freescale.com

[5] Simon Bennett, John Skelton, Ken Lunn, "Schaum's Outline of UML", Mc Graw Hill, 2005