한국자동차공학회논문집 제8권 제3호, pp.98~109

〈논문〉 SAE NO. 2000-03-0050

객체지향 프로그래밍 기법을 이용한 엔진제어시스템에 관한 연구

A Study on an Engine Control System using an Object Oriented Programming Method

윤 팔 주*, 이 상 준**, 선 우 명 호***
Paljoo Yoon, Sangjun Lee, Myungho Sunwoo

ABSTRACT

A new PC-based Engine Control system (ECS) is developed using an object oriented programming method. This system provides more convenient environment for engine tests, easier user interface and extended functions. A Windows-based ECS software is developed with class, and the class structure is built on encapsulation and abstraction. The closed-loop engine control scheme can be easily constructed by using dynamic link library and multitasking. This means that a user can perform desired experiments without clear knowledge of the hardware structure of the ECS. Also this system allows a user to individually control the ignition and fuel injection for each cylinder in a simple manner such as through a keyboard/mouse or in a real-time operation from a closed-loop control program.

주요기술용어 : Engine Control System (엔진제어시스템), Object Oriented Programming (객체지향 프로그래밍), Abstraction (추상화), 동적 연결 라이브러리(Dynamic Link Library, DLL), 장치관리자 (Device Driver), DOS-based ECS (ECSd)

1. 서 론

자동차에서 발생되는 유해 배출물로 인한 환경오염 문제가 심각한 사회적 관심사로 대두되면서, 세계 각국은 환경보호를 위해 유해 배출가스에 대한 규제를 한층 강화하는 동시에 이의 저감 기술의 개발에 더욱 노력을 경주하고 있다. 이와 같이 엄격해 지고 있는 유해 배출물에 대한 규제를 만족시키기 위해서는 저공해 엔진 자체의 개발과 더불어 이를 정밀하게 제어할 수 있는 제어기술의 개발이 병행되어야 한다.

이와 같은 요구에 부응하기 위하여 국내외의 많은 연구그룹에서 엔진을 효과적으로 제어할 수 있는 PC 기반의 엔진제어시스템(Engine Control System, ECS)에 대한 연구를 수행하여

* 회원, 한양대학교 대학원
** 회원, (주) 만도
*** 회원, 한양대학교 자동차공학과

왔다[1),2)]. PC 기반의 엔진제어시스템은 다양한 엔진의 종류에 대해서도 비교적 용이하게 적용이 가능할 뿐만 아니라, 엔진개발의 초기단계에서 실험환경의 구축 또는 상용 엔진제어시스템에서 구현이 곤란한 고급 제어이론을 이용한 엔진제어 실험 등에 유용하게 사용될 수 있다.

최근 들어 PC의 하드웨어 사양들이 고급화되어 방대한 저장공간과 뛰어난 연산능력을 갖춘 PC가 보편화되었으며 윈도(MS-Windows)와 같은 멀티태스킹 운영체제를 기반으로 하는 편리한 GUI(Graphic User Interface) 환경이 널리 사용되고 있다. 그러나 MS-DOS를 운영체제로 개발된 대부분의 기존의 PC 기반의 엔진제어시스템(ECS for DOS: ECSd)들은 PC의 고성능 하드웨어 자원들을 효과적으로 활용할 수 없는 단점이 있으며, 기존의 시스템을 응용하여 다양한 제어 알고리즘을 구현하고자 할 경우, 엔진제어시스템의 하드웨어에 대한 깊은 이해가 필요하다.

이 연구에서는 위와 같은 문제점을 해결하기 위하여 기존의 엔진제어시스템을 소프트웨어적인 측면에서 객체(Object) 개념을 도입하여 접근함으로써, 기존 시스템이 가지고 있는 하드웨어 자원 활용의 비효율성을 극복하였다. 이를 위하여 객체지향 프로그래밍(Object Oriented Programming, OOP) 기법을 적용하여 제어 대상인 엔진 제어시스템을 추상화(Abstraction)하여 구현하고, 대부분의 PC에서 사용되고 있는 멀티태스킹 운영체제인 Microsoft사의 윈도를 기본 운영체제로 삼아 다른 응용프로그램과의 다중작업성을 높이고, 보다 향상된 사용자 인터페이스를 구축하였다. 이를 통하여 다양한 제어 알고리즘의 적용이 가능하도록 함으로써 보다 나은 실험 환경을 제공하는 동시에 엔진제어시스템의 유연성을 향상시키는 것이 가능하다. 또한, 엔진제어시스템의 하드웨어에 대한 기본적인 지식만으로도 폐루프 제어시스템을 구성할 수 있도록 동적 연결 라이브러리(Dynamic Link Library, DLL)의 지원이 가능하도록 하였다. 하드웨어적 측면에서는 기존 엔진제어시스템이 PC의 하드웨어 자원을 많이 점유하는 문제점을 해결하기 위하여 상대적으로 적은 양의 하드웨어 자원을 점유하도록 함으로써, 복잡한 수학연산을 필요로 하는 고급 제어 알고리즘을 실시간에 적용하여 폐루프 엔진제어시스템을 구현하였다.

2. ECS의 하드웨어 개요[1)]

PC 기반 엔진제어시스템의 하드웨어는 다음과 같은 Subsystem으로 이루어져 있다. 엔진과의 동기를 위하여 캠과 크랭크 각도 신호에 의해 발생되는 기준신호 발생부(Reference Signal Generation Unit)와 기준신호로부터 엔진제어신호를 출력하는 엔진제어신호 발생부(Control Signal Generation Unit)가 있으며, 엔진의 폐루프 제어와 실험 데이터 수집을 위한 A/D 변환기가 장착되어 있다. 또한 각종 입출력 신호처리와 외부로부터 유입되는 전기적 노이즈 차단을 목적으로 신호처리부(Signal Processing Unit)가 추가되어 있다. 이 엔진제어시스템은 PC의 AT-BUS를 통하여 쉽게 PC에 접속하여 사용할 수 있으며, Fig. 1은 엔진제어시스템의 하드웨어의 블록도이다.

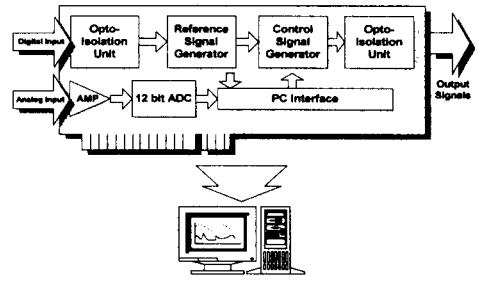

Fig. 1 Block diagram of ECS

2.1 하드웨어의 구성

엔진제어신호를 발생시키기 위하여 35개의 16-bit 타이머/카운터가 이용된다. 외부로부터 입력되는 아날로그 신호들은 최대 8 Channel 까

지 차동입력으로 받아들일 수 있으며 이 신호들은 증폭기와 필터를 거쳐 Multiplexor를 통하여 12-bit A/D 변환기로 입력된다. A/D 변환기는 6000rpm 시 크랭크 각 1° 마다 자료획득이 가능하도록 충분히 빠르나 Multiplexor의 Switching Delay와 PC BUS의 속도를 고려하여 사용목적에 따라 적절히 조절할 수 있다. 또한 이 엔진제어시스템은 각 실린더별로 점화시기와 연료분사량을 독립적으로 제어할 수 있으며, 다양한 종류의 엔진에 적용이 가능하다.

2.2 기존엔진제어시스템(DOS-based ECS: ECSd)과의 차이점

윈도 환경에서 작동되는 새로운 엔진제어시스템(Windows-based ECS: ECSw)의 하드웨어는 기존의 시스템과 하드웨어 구성 면에서 동일하다. 단지 아래 Table 1에서와 같은 몇 가지 차이점을 가진다.

1) 기존의 DOS용 엔진제어시스템은 5개의 제어신호 발생용 Interrupt와 1개의 A/D 변환기의 샘플시간 설정용 Interrupt를 포함하여 총 6개 Interrupt를 사용한다. 이는 총 15개의 PC 하드웨어 Interrupt 중 PC 자체에서 사용하는 부분을 제외한다면 나머지 모든 Interrupt를 점유하는 것이 된다. 따라서 새로운 시스템에서는 시스템의 활용도를 높이고 다양한 환경에 적용이 가능하도록 하기 위하여 180 ° CA 마다 발생하는 하나의 Interrupt만을 사용하여 엔진 제어신호를 발생시키도록 설계하였다. 이에 따라 하나의 Interrupt 신호만을 이용하여 여러 개의 Interrupt를 구별해야 하며, 이를 위하여 새로운 시스템에서는 1번 실린더의 TDC에 동기된 신호를 별도의 디지털 입력 포트를 통하여 입력받아, 이를 Interrupt 신호와 AND 연산을 통하여 각각의 Interrupt를 구별하게 된다. Fig. 2는 각 Interrupt 신호의 발생과정을 나타낸 그림이다.

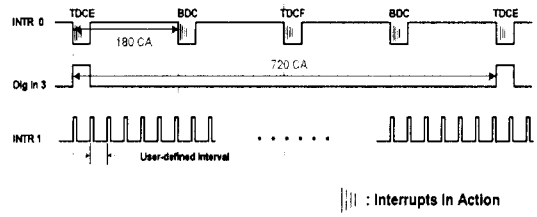

Fig. 2 Timing diagram of interrupt signals

2) 엔진의 회전수는 크랭크 축이 180 ° 회전하는 시간을 측정하여 구하게 된다. ECSd에서는 이 시간을 측정하기 위하여 Interrupt service routine에서 Software Trigger 방식으로 매 180° 마다 시간을 측정하였다. 그러나 이와 같은 연산방식은 프로그램 수행 시에 시간적인 손실을 가져오며, 멀티태스킹 운영체제 하에서 정확한 타이밍에 프로그램이 수행되지 않을 가능성도 배제할 수 없으므로 그 신뢰성과 정확도가 떨어진다. 따라서 ECSw에서는 Hardware

Table 1 Comparison of IRQ between ECSd and ECSw

ECSd				ECSw			
No.	NAME	IRQ	FUNCTION	No.	NAME	IRQ	Function
1	INTR0	10	Data Sync.	1	INTR0	9	Data Sync. Timer Control
2	INTR1	11	Timer Control				
3	INTR2	12	Timer Control	2	INTR1	5	A/D Sampling
4	INTR3	15	Timer Control				
5	INTR4	3	Timer Control				
6	INTR5	9 / 5	A/D Sampling				

Trigger방식으로 시간을 측정하며, 이 값을 타이밍에 무관하게 측정하고 읽을 수 있도록 하였다.

3. 객체지향기법의 적용과 하드웨어 추상화[3),4),5),6)]

3.1 절차식 방법과 객체지향방법

객체란 사물에 대한 추상적인 개념으로서, 해결해야 될 어떤 문제를 나타내는 자료의 형태이다. 객체 지향 방법이란 문제를 해결해 가는 과정에 있어서 어떤 방식이나 절차보다 객체라는 자료처리의 대상에 더 중점을 두고 각 객체들간 혹은 객체와 사용자간의 상호작용을 통해 문제를 해결하는 방식이다.

일반적으로 프로그램을 작성하는 방법에는 절차식 방법과 객체지향 방법의 두 가지가 있다. 절차식 방법은 어떤 문제에 대하여 작업의 순서를 정한 후 그에 따라 코드를 작성, 자료를 처리해 나가는 방법이다. 이에 비하여 객체지향 방법은 문제를 해결하는데 있어서 순서보다는 작업의 대상과 목적을 파악한 후, 각각의 대상들을 처리하는 방법을 결정, 이를 조합하여 문제를 해결해 가는 방식을 뜻한다.

엔진제어시스템에서는 제어신호 발생, 제어 알고리즘의 수행, 관련 정보의 획득 및 표시와 같은 일련의 과정들이 시간기준(Time-base)으로 처리되는 것이 아니라 크랭크 각도기준(Angle-base)으로 동시 다발적으로 처리되어진다. 이에 따라서 제어신호를 발생시키고 자료를 획득하는 등의 일을 엔진제어시스템의 각각의 부분들이 독립적으로 동작, 처리해야 하며, 전체 엔진제어시스템은 이러한 각 부분들의 독립된 작업들의 조합으로 이루어진다. 따라서 사용자적 측면에서 보았을 경우 기존의 절차식 방법보다는 객체 지향 방법이 개념적인 면에서 더욱 타당한 접근 방법으로 판단된다.

3.2 클래스와 장치 독립적 특성

이 연구에서는 객체 지향 방법을 구현하기 위해서 기존의 C 언어를 확장한 C++ 언어를 사용하였다. C++는 기존의 구조체(Structure)를 확장한 클래스(Class)라는 사용자 정의 데이터형을 통하여 객체를 추상화하여 구현할 수 있도록 확장되어 있다. 이러한 클래스 구조는 부적당한 접근으로부터 자료를 보호하고, 실제의 사용자가 그 내부구조를 알 필요가 없도록 한다(Encapsulation, Data Hiding). 또한 C++의 상속성(Inheritance)과 재활용성(Reusability)은 기존의 코드를 이용함으로서 구조를 확장하기 쉽고 코드들의 유용성을 증대시킨다. 상속성은 클래스 구조를 확장 할 경우 클래스 전체를 재 프로그래밍 할 필요성을 줄여 주며, 재활용성은 이미 완성된 클래스를 다른 프로그램의 일부로서 다시 이용함으로서 활용성을 높일 수 있음을 뜻한다.

하드웨어를 나타내는 클래스 구조를 만들 경우 하드웨어를 정확히 알지 못해도 그 하드웨어를 사용 가능 할 수 있게 된다. 또한 하드웨어의 기능을 변경하거나 추가할 경우에도 새로이 작성하는 것이 아니라 상속을 통하여 기존의 클래스를 재활용 할 수 있고, 하드웨어 전체를 하나의 구조로 나타냄으로 개념적 접근이 쉬워지는 장점이 있다.

3.3 추상화(Abstraction)와 하드웨어 클래스 설계

일반적으로 어떤 대상에 대해 객체지향 방법을 적용하고 클래스 구조를 만들고자 하는 경우 다음과 같은 순서에 의해 설계된다. 먼저 문제로 삼고자 하는 대상에서 원하는 객체들을 인식하고, 각 객체들의 속성을 파악한다. 그 후 각각의 객체들 간의 관계를 규정한 후, 하나 혹은 그 이상의 여러 객체들을 다루는 방법을 만드는 방식으로 클래스 구조를 설계한다. 이 연구에서는 이러한 방법을 동원하여 엔진제어시스템의 하드웨어를 나타내고 그것을 제어할 수 있는 방법을 갖춘 클래스를 설계하였다. 객체 지향 개념을 도입하여 설계된 ECS Class 구조는 Fig. 3과 같다.

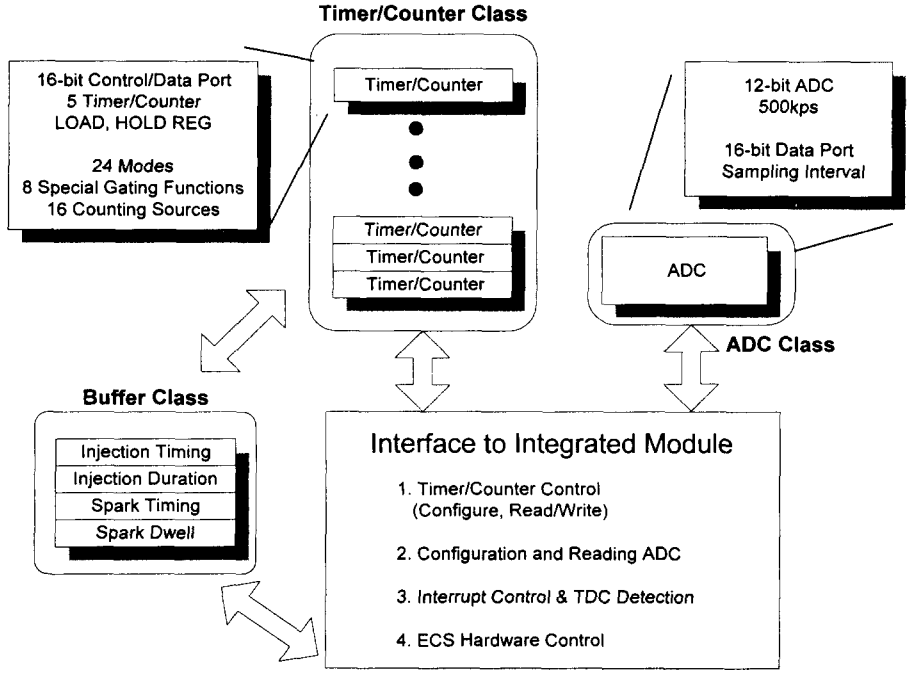

Fig. 3 Structrue of hardware class

3.3.1 객체와 그 속성의 파악

엔진제어시스템의 하드웨어 구성 중 별도의 속성을 가지며 사용자가 직접 접근하고 제어할 수 있는 부분은 타이머/카운터와 A/D 변환기이다. ECS 전체에는 추가적인 디지털 입출력 포트들이 있고, 전체적인 시스템은 제어신호 발생과 아날로그 데이터 샘플링을 위하여 각각 1개씩의 Interrupt 신호를 사용한다.

1) 타이머/카운터는 두 개의 연속된 16bit의 제어 포트와 데이터 포트를 가지고 있으며, 하나의 타이머/카운터 IC 내에 5개의 타이머/카운터를 가지고 있다. 각각의 타이머/카운터는 LOAD, HOLD 및 MODE의 3개의 래지스터를 가지고 있다.

2) A/D 변환기는 하나의 16-bit 입 · 출력 포트에 연결되어 있으며, 사용자가 설정한 타이밍에 따라 데이터를 입력받는다.

3) 디지털 입 · 출력 포트는 3-bit의 보조신호 입력부와 엔진제어를 위한 기준신호 발생부로 입력되는 엔진종류 설정신호 출력부로 구성된다.

3.3.2 각 객체들 간의 관계규정

전체 ECS의 객체는 타이머/카운터 객체의 배열과 A/D 변환기 객체를 포함하는 또 하나의 큰 객체로서, 엔진에 제어신호를 출력하거나 데이터 획득 및 사용자와의 정보 교환 등을 담당한다.

1) 타이머/카운터 객체는 각각 독립적으로 작동되며, 7개의 구성요소를 가지는 배열이다.

2) ADC 객체는 데이터 획득 주기에 따라 가변적인 크기를 가지는 데이터 저장 객체이다.

3) ECS 객체는 타이머/카운터 객체와 ADC의 객체를 포함하며, 사용자와의 자료교환을 목적으로 하는 BUFFER 객체를 가진다. 또한, 전체 하드웨어를 제어하기 위한 데이터를 포함한다.

3.3.3 처리방법의 설계

1) 타이머/카운터는 16-bit의 데이터 포트와

제어 포트를 통하여 5개의 카운터에 있는 각각의 래지스터에 접근 가능하다. 제어 포트를 통하여 원하는 래지스터를 선택하고 데이터 포트로 자료를 보내거나 읽는 순서로 작동한다.

2) ADC는 16-bit 데이터 포트로 접근하며 순차적으로 배열에 저장된다. 타이밍을 정해주면 반복적으로 작동하며, 기본적으로 읽기 전용이다.

3) BUFFER는 실제로 하드웨어에 제어신호를 내보내기 위한 값을 저장하며, 이는 일정한 타이밍에 사용자의 영역으로부터 하드웨어의 영역으로 옮겨진다.

4) ECS는 위의 모든 객체들을 포함하며 각각의 객체에 접근 할 수 있는 통로를 제공한다. 각각의 객체들은 직접적인 사용자의 접근은 불가능하며 ECS를 통해서만 가능하다. 또한 하드웨어 자원을 할당받고 실제 하드웨어를 초기화하는 기능을 가진다.

4. 엔진제어시스템의 소프트웨어 구성

4.1 윈도 환경에서의 하드웨어 제어

기존 PC 기반의 하드웨어 제어 프로그램들은 운영체제를 MS-DOS 기반으로 하였기 때문에 하드웨어에 대한 자유로운 접근이 가능하였다. 이 경우는 제작하기가 쉽고 간단하게 구현이 가능하다는 장점이 있다. 하지만 작업의 효율과 PC 자원의 효과적인 사용이란 측면에서는 바람직하지 못하다.

ECSw의 소프트웨어는 Microsoft사의 윈도에서 수행되는 프로그램이다. 기존의 Real Mode에서 작동되던 DOS용 프로그램과는 다르게 INTEL 계열 CPU의 Protected Mode에서 작동된다[7]. 이는 다시 말하면 응용프로그램이 하드웨어에 직접 접근하는 것을 허락하지 않는다는 것을 뜻한다. 윈도 환경 하에서 하드웨어에 직접 접근을 하기 위해서는 기존의 DOS 모드와는 다르게 장치관리자(Device Driver)라는 것을 통하여 가능하다. 기존의 단일작업 환경 하에서 장치관리자는 응용프로그램과 운영체제나 하드웨어를 연결시켜주는 기능뿐이었다. 그러나 다중작업 환경에서의 장치관리자는 여러 작업들간의 작업의 범위와 순서를 관리하고 하드웨어자원을 분배하는 역할을 담당한다. 이러한 장치관리자는 다음과 같이 세 가지로 분류된다.

1) DOS 장치관리자

기본적으로 DOS 장치관리자(DOS Device Driver)는 모든 하드웨어 자원의 제어권을 응용프로그램에 넘겨주어 사용자가 제한 없이 사용가능하도록 되어 있으며, 현재는 하위 호환성의 이유 외에는 거의 사용되지 않고 있다. Fig. 4는 DOS 환경에서의 장치관리자의 구조를 표시한 것이다.

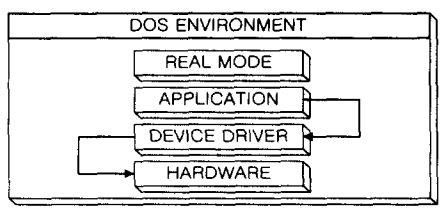

Fig. 4 DOS device driver

2) 표준 장치관리자

표준 장치관리자(Standard Device Driver)는 기존의 16-bit 환경에서 사용되던 장치관리자이다. Fig. 5에 나타낸 표준 장치관리자는 기본적인 함수(Standard API)들을 지원하며 이를 통하여 응용프로그램과 운영체제가 하드웨어 자원을 나누어 사용하도록 되어있다. 현재 32-bit 환경에서는 하위 호환성을 위하여 단순히 가상장치관리자를 호출하는 기능만을 수행한다.

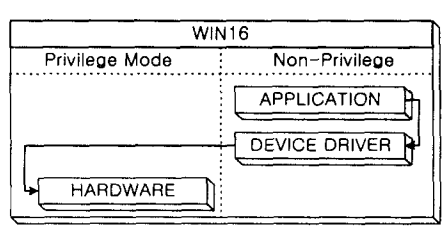

Fig. 5 WIN16 device driver

3) 가상 장치관리자

일반적으로 32-bit 환경에서 주로 사용되고 있는 장치관리자는 가상 장치관리자 (Virtual Device Driver: VxD)이다. 이것은 Fig. 6에 나타낸 바와 같이 CPU의 보호모드 상에서 작동되며 각각의 하드웨어에 대한 독점적인 제어권을 가지고 운영체제 및 응용프로그램에 사용권만을 할당함으로써 안정성을 향상시킨 것이다. 응용프로그램이 운영체제에 필요한 하드웨어 자원을 요구하면 운영체제는 장치관리자로부터 하드웨어를 가상으로 할당받아 그 사용권을 응용프로그램에 넘겨주게 된다.

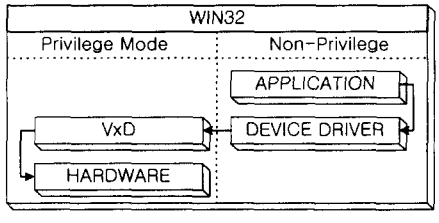

Fig. 6 WIN32 device driver

ECSw에서는 32-bit 가상장치관리자인 VxD를 사용하였다. 이는 여러 번의 함수 호출을 거치지 않고 VxD를 통하여 직접적으로 하드웨어에 접근이 가능하기 때문이다. 또한 하드웨어 자원은 운영체제를 통하여 VxD가 제어하여 주며, CPU의 보호모드를 사용하기 때문에 하드웨어에 대해 잘못된 접근이 수행되어도 이를 CPU 레벨에서 통제 할 수 있다.

4.2 WinDriver의 개요

이 연구에서는 KrfTech사의 WinDriver[8]라는 Skeleton Structure 만을 가진 개발도구를 사용하여 하드웨어에 대한 장치관리자를 제작하여 사용하였다. Fig. 7은 WinDriver의 기본구조를 나타낸 그림으로, 이 제품은 VxD의 기본 구조만을 가지고 있으며 사용자가 VxD와의 연결 부분을 작성하여 사용할 수 있다. 이는 VxD 전체를 작성하는 것보다 사용이 편리하며, 손쉽게 작성할 수 있는 장점이 있다. 따라서 하드웨어에 관한 정보의 구조가 고정된 것이 아니라 실행 시에 하드웨어의 정보를 등록하게 되어 있기 때문에, 하드웨어의 변경에도 용이하게 대처 할 수 있다. 또한 간단한 형태의 ANSI C Style의 구조체를 기본으로 지원하기 때문에 편리하게 사용할 수 있으며, 그에 따라 사용자가 하드웨어에 용이하게 접근할 수 있다

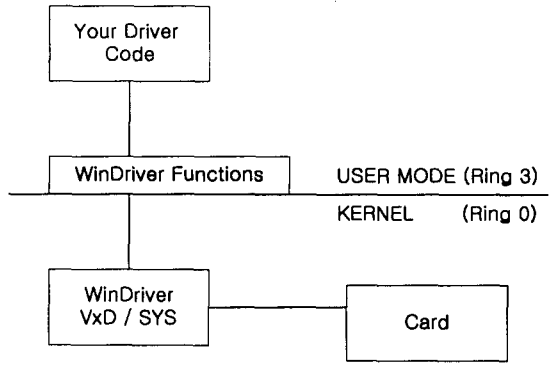

Fig. 7 Structure of WinDriver

4.3 Interrupt Service Routine (ISR)

4.3.1 윈도 환경에서 Interrupt 처리

윈도 환경에서 Interrupt를 사용하는 것은 기존의 방식과는 다른 점이 있다. 윈도는 Multi-tasking과 함께 Multi-thread를 지원한다. Thread란 CPU로부터 자원을 할당받아 수행되는 프로그램의 가장 작은 단위로서 CPU의 Time Scheduling과 가장 관계가 깊다. 이를 운영체제가 직접적으로 지원하며 작업의 Priority를 설정함에 따라 시간할당을 조절할 수 있다. 따라서 다른 모든 작업들이 정지한 상태에서 ISR만을 동작시키기보다는 Interrupt를 담당하는 Thread를 생성한 후, 이 Thread에서 Interrupt 신호를 Polling한다. 또한 이를 장치관리자와 연결하여 Interrupt 신호가 입력될 때에만 Thread를 활성화시키면 CPU Time을 전혀 사용하지 않게 된다. 이것은 Interrupt의 개념과 동일하게 구현된 것이다. Fig. 8은 윈도 상

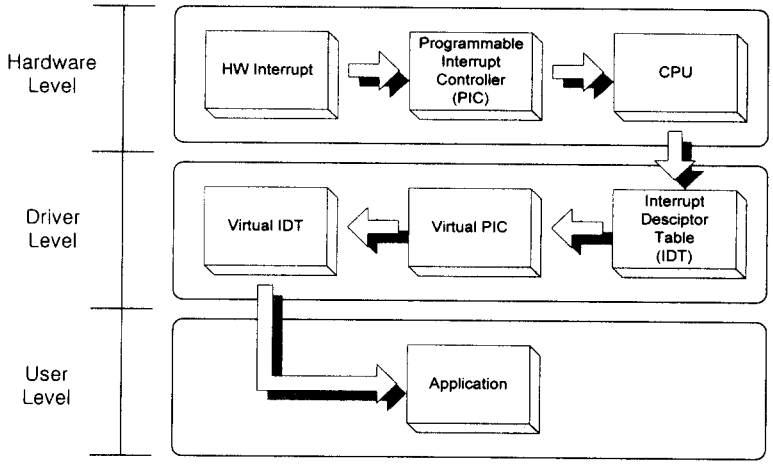

Fig. 8 Interrupt process in Windows environment

에서의 인터럽트를 처리하는 순서를 나타낸다.

1) BUS를 통하여 Interrupt 신호가 들어오면 PIC(Programmable Interrupt Controller)를 통하여 Interrupt의 종류와 Priority가 CPU에 전해지면서 하드웨어 레벨의 Interrupt가 설정된다. CPU는 IDT(Interrupt Descriptor Table)를 참조하여 Device Driver를 구동시키고 하드웨어 Interrupt를 종료한다.

2) 장치관리자는 VPIC(Virtual PIC)를 만들어서 어떤 응용프로그램이 Interrupt를 필요로 하는지를 파악한다. 이를 바탕으로 각 프로그램마다의 VIDT(Virtual IDT)를 통하여 각 프로그램의 Interrupt Routine Thread를 활성화시킨다. 즉, Thread Priority를 증가시켜 다른 Thread에 비하여 많은 CPU Time을 할당받는다.

3) 응용프로그램의 Thread가 수행되면 Interrupt Process를 한번 수행 후 다시 Idle 모드로 돌아가서 장치관리자의 호출을 기다린다. Interrupt를 처리하는 Thread는 엔진의 작동과 동시에 한번만 생성되며 따라서 이를 생성하기 위하여 자원을 할당받는 등의 시간 손실은 무시될 수 있다.

4.3.2 Interrupt의 발생과 처리

엔진의 크랭크축에 장착된 인코더 신호와 정확히 동기화 되어 제어 신호를 출력하기 위하여, 엔진제어시스템에서는 Interrupt 신호를 사용한다. 기존 시스템(ECSd)에서 6개의 Interrupt를 사용하는 것에 비하여 새로운 시스템(ECSw)에서는 2개의 Interrupt만을 사용한다. 이것은 최근 PC의 성능 향상에 따라 다른 하드웨어나 프로그램에서 많은 자원들을 필요로 하기 때문에 다른 작업과의 자원충돌을 막기 위한 것이 가장 큰 이유이다.

1) Fig. 2에 나타낸 바와 같이 INTR 0(IRQ9)은 매 180 ° 마다 발생되며, 엔진의 이벤트 확인을 위하여 이와 동기되어 720 ° 마다 발생되는 추가적인 디지털 입력신호를 통하여 1번 실린더의 TDC 신호를 입력받는다. 기존의 5개 Interrupt 신호를 이용한 방법은 여러 다른 하드웨어와의 상호 충돌 문제로 이용할 수 없다.

2) INTR 1(IRQ5)은 A/D 변환을 위하여 사용하며 사용자가 원하는 데이터 획득 주기에 따라 그 발생주기를 설정하여 사용할 수 있다.

4.4 페루프 제어를 위한 확장 모듈

엔진제어시스템의 용도를 개루프 제어기에서

페루프 제어기로 전환해야 할 경우 직접 프로그램의 소스 코드 레벨에서의 수정 방법보다는 확장성이 좋은 Dynamic Link Library(DLL)를 사용하는 방법을 선택하였다. DLL은 프로그램이 만들어질 때 연결되는 것이 아니라 실행 시에 연결되는 함수 모듈로서 이미 완성된 프로그램의 확장이 용이하며 동일한 함수를 여러 프로그램에서 사용할 경우는 저장공간의 이용효율을 향상시킬 수 있는 이점이 있다. Fig. 9는 기존의 함수모듈과 DLL의 차이점을 보여준다.

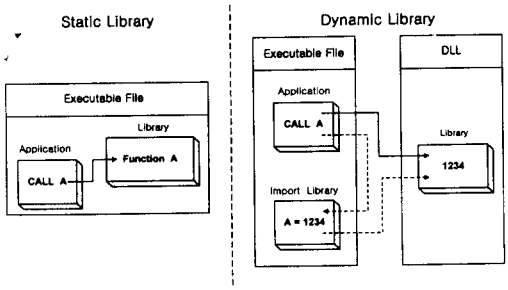

Fig. 9 Static library vs DLL

4.5 ECSw의 소프트웨어 개요

ECSw의 외형은 기존의 MDI(Multi Document Interface)와 유사하다. 그러나 단일의 하드웨어를 제어하는 프로그램이므로 다중이라는 개념은 올바르지 않다. 따라서 프로그램의 전체구조는 하나의 하드웨어 즉, 하나의 Document Class에 여러 형태의 View Class가 결합된 SDMVI(Single Document Multi View Interface)를 채택하고 있다. 전체 프로그램은 앞에서 구성한 하드웨어 클래스를 기본으로 하여 Microsoft사의 Visual C++를 사용하여 작성되었다. 엔진제어시스템 소프트웨어의 주요 기능은 사용자에게 엔진제어에 필요한 여러 기능을 최대한 지원함으로서 시스템의 유연성과 확장성의 향상에 중점을 두었다. 즉, 기존의 엔진제어 시스템의 기본 기능이외에 DLL을 이용하여 폐루프 제어기로 확장이 가능하도록 하였으며, 이 과정에서 사용자의 하드웨어에 대한 깊은 이해는 필요치 않다.

4.5.1 ECSw 소프트웨어의 구조

이 연구에서 구현된 ECSw의 소프트웨어는 Fig. 10과 같이 구성된다. 전체 프로그램은 가상으로 구현된 하드웨어를 통하여 시스템을 제어하고, 시스템의 정보를 화면에 표시하며, 사용자의 입력을 처리한다. 또한 외부 DLL을 이용하여 별도의 제어 알고리즘을 구현할 수 있다.

4.5.2 제어창의 구성

ECSw의 제어창은 Fig. 11에 나타낸 바와 같이 현재 엔진의 사양과 운전상황 등을 표시하며

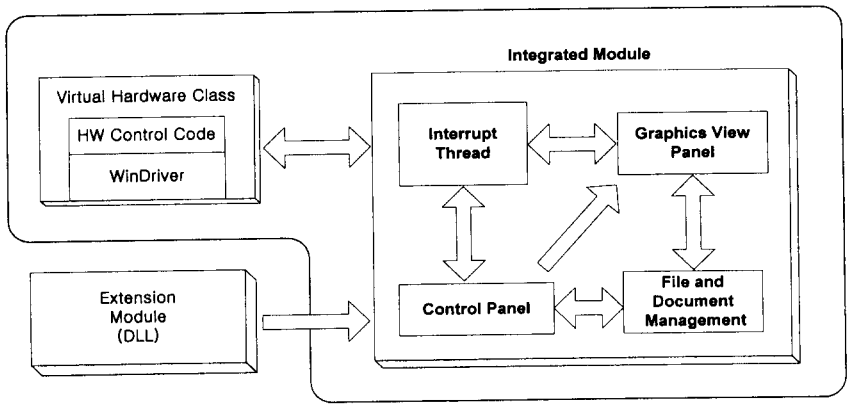

Fig. 10 Software block diagram of ECSw

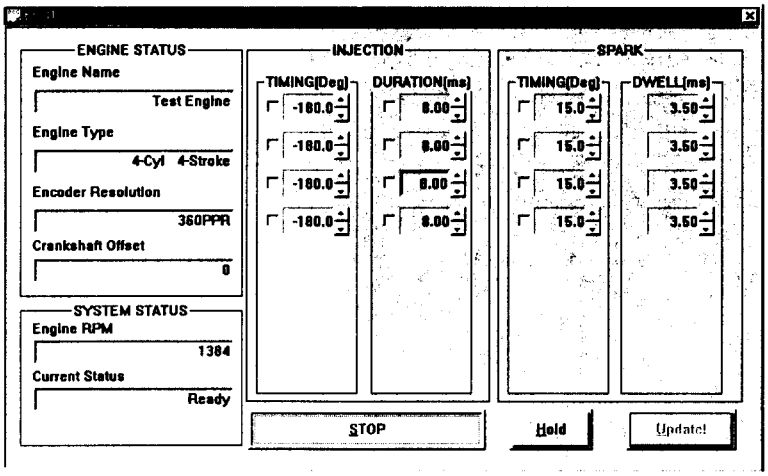

Fig. 11 Screen structure of the ECSw

주며 개루프 제어기로 사용될 경우 기본적인 제어값(연료 분사시기 및 분사량, 점화시기 및 충전시간)을 실린더별로 직접 변경하도록 제작되었다. 또한 엔진의 시동과 정지에 관련된 조작이 가능하며, 변경된 값의 적용 여부를 조절 할 수 있도록 HOLD와 UPDATE 기능을 보유하고 있다. HOLD는 엔진제어시스템에서의 설정값이 엔진에 영향을 미치지 않는 상태를 나타낸다. 즉 엔진제어변수를 변경하는 동안 엔진에 영향을 주지 않고 모든 변수값이 설정된 후, 이 값을 동시에 엔진에 영향을 미치게 하고자 할 때 사용된다.

5. 제어성능 실험

다음의 Fig. 12와 Fig. 13은 2000RPM, 0.4bar의 운전조건에서 직렬 4기통 엔진의 연료분사 및 점화제어 신호를 측정한 결과이다. 연료분사신호는 분사시기(Injection Timing) ATDC 180°, 분사기간(Injection Duration) 4.0msec이며, 점화신호는 점화시기(Spark Timing) BTDC 25°, 충전시간(Dwell Time) 3.5msec인 경우이다. 연료분사신호와 점화 신호가 주어진 조건을 정확히 유지하면서 안정적으로 작동되고 있음을 볼 수 있다.

각 제어신호의 정확성을 검증하기 위하여 2번 실린더의 연료분사신호 및 1번 실린더의 점화신호를 Fig. 14와 Fig. 15에 각각 나타내었다. 2번 실린더의 연료분사신호는 분사시기가 ATDC 180° 이므로 1번 실린더의 압축 TDC에서 분사가 개시되며, 정확히 4ms간 지속되는 것을 알 수 있다. 또한 점화신호의 경우 3.5msec의 충전시간을 가지고 TDCF에서 약 2ms정도 앞서 방전되는 것을 볼 수 있다. 이는 실험 조건인 2000rpm에서 크랭크 각도 25° 에 해당한다.

Fig. 16, 17, 18은 엔진제어시스템을 Table 2와 같이 여러 가지 엔진 종류에 대하여 같은 조건으로 작동시킨 후 HP 16500B Logic Analysis System을 이용하여 그 제어 신호를 측정한 것이다. 이 실험은 실제 엔진 운전상태가 아니라 ECSw의 기능중 하나인 인코더 신호발생기를 이용하여 2000[rpm]의 실제 운전상황을 모사하였다. 각 엔진의 경우 모두 점화순서에 따라 정확한 점화 및 분사시기에 신호가 출력되고 있음을 알 수 있다.

Table 2 Engine control parameters for various types of engine

(Simulated Engine Speed = 2000 RPM)

Engine Type	Spark Control		Injection Control		Firing Order
	Timing [°BTDC]	Dwell [ms]	Timing [°BTDC]	Duration [ms]	
L4	15	3.5	-180	8	1342
V6					123456
V8					18436572

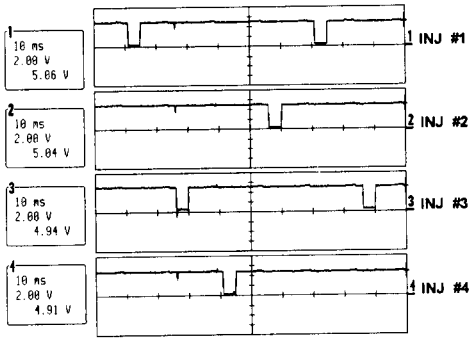

Fig. 12 Fuel injection control signals

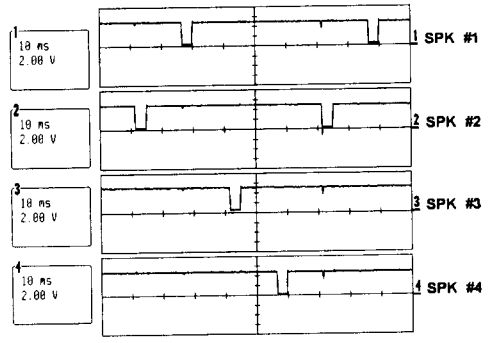

Fig. 13 Spark control signals

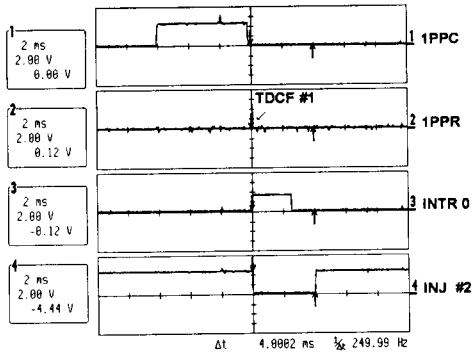

Fig. 14 Measured injection timing and duration of cylinder #2

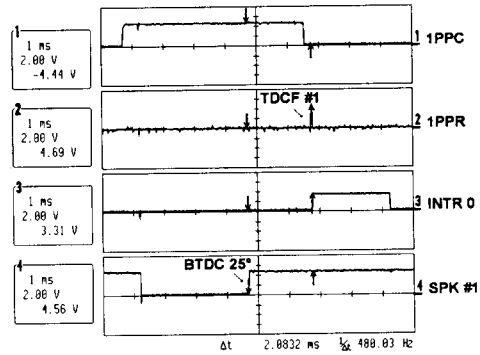

Fig. 15 Measured spark timing and dwell of Cylinder #1

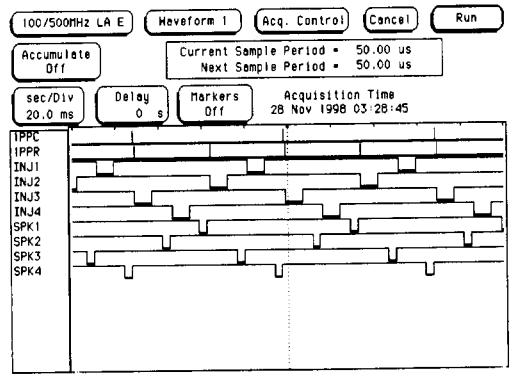

Fig. 16 Waveform of control signals (L4)

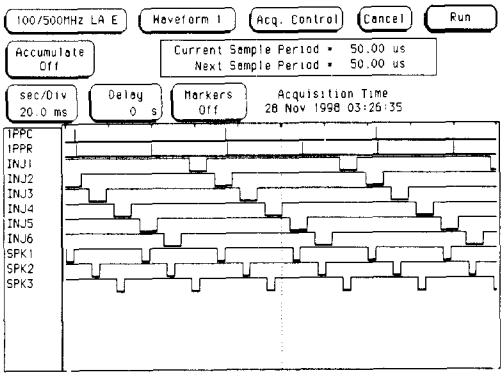

Fig. 17 Waveform of control signals (V6)

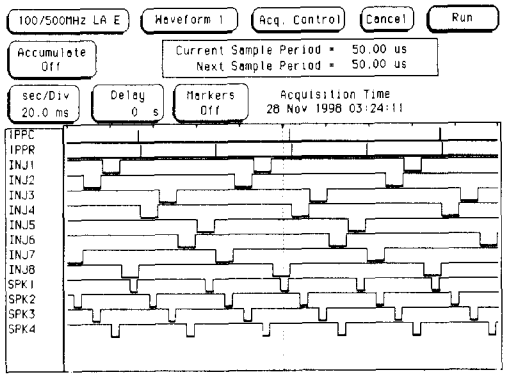

Fig. 18 Waveform of control signals (V8)

6. 결 론

이 연구에서 개발된 MS-Windows 기반의 엔진제어시스템(ECSw)은 기존의 MS-DOS를 운영체제로 하는 엔진제어시스템(ECSd)에 비해 최소한의 하드웨어 변경을 통하여 확장성과 사용자의 편의성이 한층 향상되었으며, 이를 이용하여 저공해 엔진과 그 제어 알고리즘 개발 및 연소 특성 해석을 보다 효과적으로 수행할 수 있는 환경을 제공할 수 있다. 엔진제어시스템의 개발에 대한 주요 연구결과는 아래와 같다.

1) 객체지향 프로그래밍 기법을 이용하여 하드웨어에 대한 개념을 추상화함으로써, 제어시스템에 대한 직접적인 이해 없이도 엔진 제어실험을 효과적으로 수행할 수 있다.

2) DLL을 이용함으로써 외부에서 제어시스템을 편리하게 확장할 수 있게 되었으며, 이를 통하여 폐루프 엔진제어세스템을 용이하게 구현할 수 있다.

3) 하드웨어 장치관리자의 설계를 통하여 윈도 환경에서 하드웨어 자원의 효율적인 관리가 가능하게 되었다. 이에 따라 엔진 제어신호 발생과정에 5개의 Interrupt를 사용해야 했던 기존 시스템에 비해 1개의 Interrupt를 사용하여 엔진제어 신호를 출력할 수 있다.

4) 연소기관 해석에 필수적인 다양한 엔진 실험을 효과적으로 수행할 수 있는 환경을 제공하며, GUI 작업 환경에서 손쉽게 필요한 자료를 획득하고 후처리를 할 수 있다.

참 고 문 헌

1) 윤팔주, 김명준, 선우명호, "크랭크 각 기준의 엔진제어시스템 설계· 제작에 관한 연구", 한국자동차공학회 논문집, 6권, 4호, 1998.

2) Lillelund, J. and Hendricks, E., "A PC Engine Control Development System (ECDS)", SAE Paper 910259, 1991.

3) Bjarne Stroustrup, "The C++ Programming Language Second Edition", Addison - Wesley Publishing Company, 1991.

4) Stephan Prata, "C++ Primer Plus 2nd Edition", Waite Group Press, 1996.

5) Robert Lafore, "Object-Oriented Programm -ing in C++ 2nd Edition", Waite Group Press.

6) David Thielen and Bryan Woodruff, "Writing Windows Virtual Device Drivers", Addison-Wesley Publishing Company, 1994.

7) Charles Petzold & Paul Yao, " Programming Windows 95", Microsoft Press, 1996.

8) "WinDriver V3.0 Developer's Guide", KrfTech.