1999년도 대한전기학회 하계학술대회 논문집 1999. 7.19-21

지능형 센서의 데이터 처리 모듈 개발

김인욱, 임동진
한양대학교 제어계측공학과

Development of data processing module of intelligent sensor

Kim In Uk, Lim Dong Jin
Dept. of Control and Instrumentation Engineering at Hanyang University

Abstract - In the case of using sensor in the industrial control systems, the location of sensor is not close to the system which utilizes the sensor data. Two main functions of intelligent sensor are data processing and communication. In this paper, we will show that the developed result of intelligent sensor, which process the sensor data inside of the sensor module, except for the communication function. For this, we refered to the Profibus and Fieldbus Foundation standard.

1. 서 론

본 논문에서는 지능형 센서 모듈의 기능들 중 통신 기능을 제외한 나머지 필요한 기능들을 개발한다. 이러한 기능들은 센서로부터 얻어지는 데이터를 사용 목적에 알맞게 변환, 처리하여 통신 모듈에 전달하는 역할을 담당한다. 전처리 기능의 종류에는 하드웨어와 연결되어 동작하는 A/D 또는 D/A conversion기능, 노이즈 제거를 위한 filtering 기능, 센서로부터의 데이터 값을 선형화하는 linearization기능 그리고 전처리된 데이터를 engineering unit으로 바꾸어 주는 scaling 기능 등이 있다. 또한, 센서 모듈들과 master station 으로 구성된 전체 시스템을 운용하기 위한 응용 시스템 소프트웨어를 개발하였고, 전체 시스템의 성능을 평가하기 위해서 사용될 제어 대상 시스템으로서 온도 제어시스템을 구성하였다. 각 항목 별 개발 결과를 상세히 기술하면 다음과 같다.

2. 본 론

1. 센서 모듈의 구현

1.1 하드웨어의 구성

그림 1-1은 Main Board의 구조에 대한 그림이다. Main Board의 Block에는 CPU 및 메모리, RS-485통신부, AD/ DA 변환부로 이루어진다. CPU는 Motorola사의 MC68360RC-33을 사용하는데 이는 크게 CPU32+, SIM60, CPM의 세 부분으로 구성되어 있다. 이 CPU는 내, 외부 모두 32bit의 버스를 가지며, 33MHz로 동작되는데, 현재 산업용 네트워크 장비에 많이 사용되고 있는 MC68302보다 개선된 모델이다.

AD/DA 변환부는 지능형 센서에서 생성되는 아날로그 데이터를 처리하기 위한 ADC(Analog to Digital Converter)와 지능형 actuator에 제어 신호를 처리하기 위한 DAC(Digital to Analog Converter)로 구성되어 있다. MC68360은 3개의 Port를 지원하는데 이 중 Port B는 Parallel data 입출력용으로 사용하였고 AD/DA에서

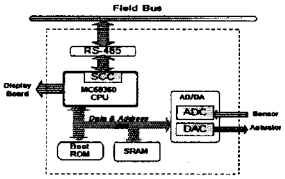

그림 1-1 Main Board의 구조

는 이 Port B(PB0 -PB17)를 이용하여 Sensor & Actuator Board의 데이터를 입출력하도록 설정하였다. 현재 Main Board의 모든 칩에 대한 Test가 끝난 상태이며 RS-485통신부의 Test만을 남겨두고 있다. 통신부는 노드간의 통신을 위해 필드버스 표준인 RS-485(MAX1487)을 이용하여 MC68360의 SCC(Serial Communication Controller)에 의해 제어된다.

메모리는 필드버스 프로토콜 프로그램 저장용 1M비트 ROM(AT29C010A)과 데이터 저장을 위한 1M비트 SRAM(681000)을 4개로 구성되어 있다. 메모리 구조는 가장 하위주소에 인터럽트 벡터 테이블이 들어가며 여기에는 RESET 벡터와 Stack Pointer, Program Counter, 그리고 각종 인터럽트 벡터가 있다. 실제 프로그램의 base는 0x400번지부터 시작된다. SRAM은 0x80000 번지에서 시작하며 처음 0x80FFF까지는 인터럽트 벡터 복사 영역으로 사용하며, 0x81000번지부터 정적 데이터(전역변수), 동적 데이터(지역변수), Heap, Stack의 순서로 메모리가 구성되어 진다. 또한 68360의 기능을 제어하기 위한 각종 레지스터의 번지가 0x01001000부터 2k바이트의 어드레스에 할당하였다.

1.2 전체적인 소프트웨어의 구성

본 연구에서 구현된 FBA(Function Block Application)을 포함한 센서 모듈의 전체적인 소프트웨어의 구조는 그림 1-2와 같다. FMS 블록을 포함한 그 아래의 블록들은 통신을 위한 모듈로서 네트웍 메시지를 생성하거나 전달받는 역할을 담당한다. System Management는 서로 다른 필드 디바이스들 사이의 동기화를 위해 system time을 제공하여 전체 시스템의 시간을 관리하고 FBA 내부에 있는 FB(Function Block)들의 실행을 스케줄링하며 Network Management는 FBA의 네트웍 I/O인 Unscheduled traffic과 Scheduled traffic을 관리한다. 본 연구에서는 통신모듈과 system 및 네트웍관리기능을 제외한 나머지 부분-즉, 센서의 전처리 기능을 담당하는 FBA를 구현한다.

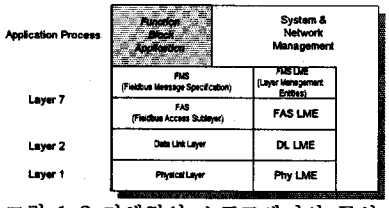

그림 1-2 전체적인 소프트웨어의 구성

구현된 FBA의 구조는 그림 1-3에 나타내었고 내부의 Block과 Object들의 기능과 특징은 다음과 같다.

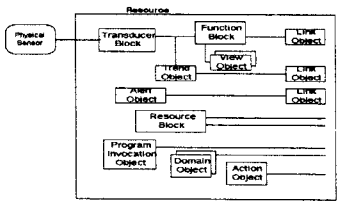

그림 1-3 FBA의 구조

Resource는 device 내의 하드웨어 혹은 소프트웨어 내부의 논리적인 단위이며 process와 communication interface로부터 data와 events를 받아 처리한다. FB (Function Block)은 I/O device와 네트워크와 독립적으로 동작하는 application에 의해 수행되는 기본적인 함수를 말한다. 각 FB은 특유의 알고리듬과 contained parameter에 따라서 input parameter를 처리한다. 그 종류로는 센서로부터 데이터를 읽어들이는 Input Function Block, 하드웨어로의 출력을 담당하는 Output Function Block, 그리고 제어를 위한 Control Function Block과 계산을 위한 Calculation Function Block등으로 구성 되어있다. Transducer Block은 센서, 액츄에이터, 스위치 등과 같은 외부 I/O 기기로부터 FB을 분리하고 transducer block의 data를 사용하는 FB에 부담을 덜 주면서 센서로부터 data를 얻기 위해서 필요할 때마다 실행되며 function block의 가장 중요한 기능인 센서 data의 전처리 기능 (linearization, filtering)을 수행한다. Link Object는 resources와 communication network를 통해 교환되는 정보 사이의 mapping을 제공하고 resource내 또는 resources사이에서 FB간에 교환되는 process data와 event들을 정의하고 trend reporting과 alert notification의 지원을 위한 정보 교환을 정의한다. Resource Block은 필드 디바이스의 하드웨어적인 특성을 나타낸다. View & Trend objects는 FBA내에서 parameter data를 access하고 resource내에 수집되고 저장된 short term history data 제공하며 interface device가 FMS read 서비스를 사용하면 언제든지 읽혀질 수 있다. Alert objects는 interface device와 다른 필드 devices에게 event reporting을 제공한다.

2 센서 모듈 firmware의 구현

2.1 Data type and structure 정의

Firmware 소프트웨어의 구현을 위해서 우선 블록과 Object들에서 사용되는 데이터의 형식과 구조를 먼저 정의하였다. 그리고 위와 같이 정의된 데이터형과 structure는 블록과 object들의 formal model 형식에 맞추어 FBA내부의 블록 및 object들을 구현하는데 사용하였다.

2.2 전처리 기능의 구현

Transducer block은 센서, 액츄에이터, 그리고 스위치 등과 같은 입출력 기기와 FB(Function Block)의 기능이 분리될 수 있도록 하기 위해서 정의된 소프트웨어 블록이다. 또한 Transducer Block은 filtering, linearization 등의 전처리 기능을 담당하여 실제로 data를 사용하는 FB의 부담을 덜어주게 된다. 그리고 transducer block의 실행은 올바른 동작을 위하여 외부에서 스케줄되는 것이 아니라 자체적으로 적절한 실행주기를 가지고 실행되며 사용자에 의해서 임의대로 바뀔 수 없다.

Transducer block은 크게 Input Transducer Block, Output Transducer Block, Display Transducer Block의 세 가지 형태로 나뉘어 구현되었다. 구현된 Transducer block의 기능에 대한 블록 다이어그램을 보면 아래와 같다.

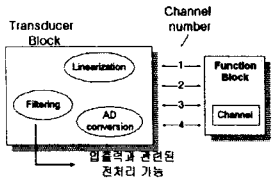

그림 1-4 Transducer Block의 구조 및 기능

위의 그림에서처럼 센서의 데이터를 전처리 하는데 필요한 기능들은 transducer block의 내부에 함수의 형태로써 존재한다. 구현된 전처리 기능 중에서 filtering은 아래와 같은 식으로 표현되는 1차 ARMA (AutoRegressive Moving Average) filter인 exponential filter를 사용하였다. 여기서 h는 sampling interval이고 $\hat{y}$는 filtering된 output이며 y는 센서로부터 읽어들인 input 값이다. 그리고 α는 0에서 1사이의 값을 갖는다.

$$\hat{y}(kh) = \alpha\hat{y}[(k-1)h] + (1-\alpha)y(kh)$$

Linearization 기능은 사용되는 센서 마다 각각 다른 식을 사용하여야 하므로 센서종류별로 구현을 해야 한다. 구현된 온도센서(thermocouple)에 대한 선형화 방법의 예를 Pseudo code로 나타내면 다음과 같다.

```
IF Unity(Default) : OUT = INPUT
IF LINEAR         : OUT = INPUT*M + B
IF TYPE S         : OUT = F(INPUT)
IF TYPE J         : OUT = F(INPUT)
IF TYPE K         : OUT = F(INPUT)
IF TYPE T         : OUT = F(INPUT)
```

위에서 보는 바와 같이 사용자가 센서의 데이터를 선형화 하는 방법을 온도센서의 종류에 따라 지정할 수 있도록 하였다. Default값은 input을 그대로 출력하는 것이며 LINEAR인 경우는 기울기 M과 offset B값을 정하여 output을 생성하고, thermocouple의 type을 정하는 경우는 각각의 온도-전압 특성에 따라 함수 F를 결정하여 output을 계산하는 방법을 사용하였다.

2.3 Function Block의 구현

2.3.1 AI(Analog Input) Block

구현된 AI Block은 다음과 같이 동작한다. 현재 BLOCK의 모드가 MAN이면 사용자가 입력한 값을 output으로 내보내고 AUTO이면 transducer block을 통해 값을 읽는다.

```
If mode = MAN then
        OUT = user input
Else If mode = AUTO then
        Read from channel value
```

그리고 나서 SIMULATE 의 상태가 ON 이면 사용자의 입력 값을 channel value로 설정하고 OFF이면 원래의 channel value를 사용한다.

```
If SIMULATE = ON then
channel value = user input
```

이렇게 얻어진 값은 L_TYPE이라는 parameter를 통해 Direct, Indirect, Indirect sqr root의 세 가지 방법으로 계산되고 스케일링된다.

$$FIELD\_VAL = 100 \times \frac{channel\ value - EU@0\%}{EU@100\% - EU@0\%} \quad [XD\_SCALE]$$

$$Direct: PV = channel\ value$$

$$Indirect: PV = \frac{\frac{FIELD\_VAL}{100}}{EU@100\% - EU@0\%} + EU@0\% \quad [OUT\_SCALE]$$

$$Ind\ Sqr\ Root: PV = \sqrt{FIELD\_VAL} \times (EU@100\% - EU@0\%) + EU@0\% \quad [OUT\_SCALE]$$

PV는 PV_FTIME이라는 time constant를 가진 exponential filter에 의해 filtering된다. 최종적으로 PV는 OUT block에 저장되는데 이 값을 HI/LO Alarm을 적용한다.

```
OUT = a*OUT_OLD + (1-a)*PV
```

If (OUT > HI_LIM) or (OUT < LO_LIM) then ALARM

AI block의 전체 동작을 순서도로 나타내면 다음과 같다.

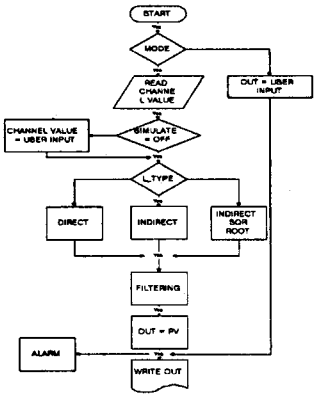

그림 1-5 AI Block의 순서도

2.3.2 AO(Analog Output) Block

구현된 AI Block은 다음과 같이 동작한다. 현재 block의 mode에 따라 CAS_IN과 RCAS_IN을 받아들여서 Setpoint block에서 SP_RATE_DN이나 SP_RATE_UP을 이용하여 SP의 rate를 조절하고 Limit alarm을 적용한다.

If mode = CAS then IN = CAS_IN
Else If mode = RCAS then IN = RCAS_IN
*IN = SP_RATE_UP*IN or SP_RATE_DN*IN*
If IN > SP_HI_LIM then Hi Alarm
If IN < SP_LO_LIM then Lo Alarm

그리고 Out Convert block을 거쳐서 scaling 된다.

$$Temp = \frac{SP - EU@0\%}{EU@100\% - EU@0\%}[PVscale]$$

$$OUT = Temp \times (EU@100\% - EU@0\%) + EU@0\%[XDscale]$$

$$Temp = \frac{READBACK - EU@0\%}{EU@100\% - EU@0\%}[XDscale]$$

$$PV = Temp \times (EU@100\% - EU@0\%) + EU@0\%[PVscale]$$

생성된 OUT값은 바로 출력되는 것이 아니라 SIMULATE의 상태에 따라 user input값이 출력되거나 OUT값이 그대로 출력된다.

If SIMULATE = ON then channel value=user input
Else channel value = OUT

AO block의 전체 동작의 순서도는 다음과 같다.

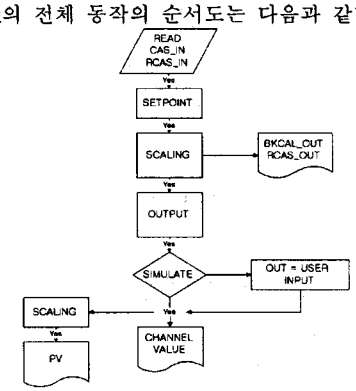

그림 1-6 AO Block의 순서도

2.3.3 PID Control Block

구현된 PID Block은 다음과 같이 동작한다. Block의 모드에 따라 CAS_IN 혹은 RCAS_IN을 입력 값으로 받아 SP_IN으로 사용하고 IN을 입력받아 PV_IN으로 사용한다.

If mode = CAS then SP_IN = CAS_IN
Else If mode = RCAS then SP_IN = RCAS_IN
PV_IN = IN

Setpoint block에서 SP_RATE_DN이나 SP_RATE_UP을 이용하여 SP_IN의 rate를 조절하고 Limit alarm을 적용한다.

*SP = SP_RATE_UP*SP_IN or SP_RATE_DN*SP_IN*
If SP > SP_HI_LIM then Hi Alarm
If SP < SP_LO_LIM then Lo Alarm

PV_IN은 PV_FTIME이라는 time constant를 가진 exponential filter에 의해 filtering되고 이 값에 HI/LO Alarm을 적용한다.

*PV = α*PV_OLD + (1-α)*PV_IN*
If (PV > HI_LIM) then Hi Alarm
If (PV < LO_LIM) then Lo Alarm

계산되어진 SP와 PV는 control 알고리즘을 계산하기 위해 Control block으로 입력되고 다음과 같은 PID 계산식을 거치고 Feedforward block을 통해 OUT값이 생성되고 이 값에 HI/LO Alarm을 적용한다.

$$u(n) = u(n-1) + K_P(e_n - e_{n-1}) + K_P e_n T + \frac{K_D}{T}(e_n - 2e_{n-1} + e_{n-2})$$

error = SP-PV
OUT = OUT_old + GAIN(error - error_old)*
*+ RESET*error*Time*
+ RATE(error-2*error_old+error_old_2)/Time*
error_old_2 = error_old
error_old = error

*OUT = OUT + FF_VAL*FF_SCALE*FF_GAIN*
If (OUT > OUT_HI_LIM) then Hi Alarm
If (OUT < OUT_LO_LIM) then Lo Alarm
ROUT_OUT = OUT

PID block의 전체 동작을 순서도로 나타내면 다음과 같다.

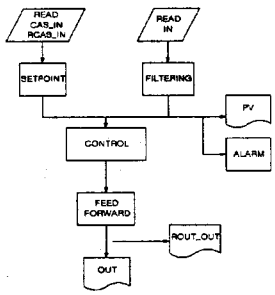

그림 1-7 PID Block의 순서도

3. 결 론

본 논문에서는 지능형 센서 모듈의 기능들 중 통신 기능을 제외한 센서의 데이터 신호를 사용 목적에 알맞게 변환, 처리하여 통신 모듈에 전달하는 전처리 하는 기능인 A/D 또는 D/A conversion기능, 노이즈 제거를 위한 filtering 기능, 센서로부터의 데이터 값을 선형화 하는 linearization기능 그리고 전처리된 데이터를 engineering unit으로 바꾸어 주는 scaling 기능 등을 개발하였다.

또한, 센서 모듈들과 master station으로 구성된 전체 시스템을 운용하기 위한 응용 시스템 소프트웨어를 개발하였고, 전체 시스템의 성능을 평가하기 위해서 사용될 제어 대상 시스템으로서 온도 제어시스템을 구성하였다.

[참 고 문 헌]

[1] "Function Block Application Process - Part 1 and Fieldbus Foundation, 1996

[2] "User Layer Technical Report for the Fieldbus Stand Instrument Society of America Standards Committee,

[3] Gustaf Olsson, Gianguido Piani, "Computer System For Automation And Control", Prentice-Hall, p143 1992