# PLC를 위한 PROFIBUS-DP 구현에 관한 연구

최명수

삼성항공산업(주) 정밀기기연구소

## 서 론

### 1. 연구 배경

최근 수년간에 걸쳐 이루워진 센서 기술의 발달과 컴퓨터를 이용한 Data 처리 능력의 발달은 4-20mA의 아나로그 신호들을 그대로 전송하거나, RS-232등과 같은 Serial 통신장비들에 의하여 point-to-point 방식으로 제어 되던 Sensor,Actuator Level의 자동화에 일대 변화를 요구 하기 시작 했다[1]. 이러한 변화는 Sensor, Actuator Level에 네트워크 시스템 적용을 가속화 시켰다. 그러나 기존의 국제 표준 네트워크인 Mini-MAP은 비용과 실시간성에서 부적합 하였고 SIEMENS의 SINEC, AB의 Data-Highway, MITSUBISHI의 MELSEC 등은 상이한 규격으로 호환성에 문제가 있었다. 따라서 Sensor, Actuator Level를 위한 새로운 국제 표준 네트워크이 필요했고 그 결과 Field-Bus 시스템이 정의 되었다. Field-Bus는 설치에 많은 비용과 노력이 소요 되며 시스템의 유지 보수도 어렵던 종래의 Sensor, Actuator Level 제어 시스템에 Bus 형태의 분산 처리가 가능한 통신 시스템의 사용을 가능케 하여 사용자들에게 다음과 같은 이득을 제공 한다.

(1) 여러 기기들간에 단일 케이블을 사용함으로써 배선 작업이 용이 해져 설치 비용이 절감 된다.
(2) 시스템의 확장및 변형이 용이하여 시스템 운용에 유연성을 제공한다.
(3) 케이블 단선등의 결함을 쉽게 탐지 함으로써 시스템의 신뢰도가 증대 된다.
(4) 네트워크를 통하여 하나의 데이터를 여러개의 노드에 동시에 전송 함으로써 데이타의 일관성이 유지 된다.
(5) 디지탈 통신 기술을 사용함으로써 노이즈에 의한 데이타 결함을 크게 완화 한다.

Field-Bus의 이러한 장점 때문에 1980년대 중반 부터 유럽,미국,일본등 기술 선진국에서 Field-Bus에 관한 많은 연구가 수행 되어, 다양한 규격이 제정 발표 되었으나 현재 PROFIBUS-DP, CAN, INTERBUS-S등이 주요 시장을 형성 하고 있다.

### 2. 연구의 범위

PROFIBSUBUS-DP의 상품화는 반드시 다음의 4 항목을 만족 시켜야 한다.

(1) Conformance
(2) Interoperability
(3) Performance
(4) Convenience

Interoperability는 구현된 표준 네트워크 제품간에 상호 동작성을 의미하며Conformance는 PROFIBUS-DP를 대상 기기에 접속(Interface)하기 위해 표준안에서 정의한 내용들이 표준안의 정의 대로 정확히 구현 되었는가를 판단 하는 항목이며 Performace는 Field-Bus의 가장 중요 항목인 실시간(Real-Time) 요소가 허용 가능한수준 으로 구현 되었는가를 판단 하는 항목이다. Convenience는 사용상 편리성및 다양한 Application에 쉽게 적용 가능한가를 판단 하는 항목으로 상품화시 구현자가 표준안의 Conformance을 저해 하지 않는 범위에서 정의한 부가 기능이 포함 된다.

본 논문에서는 상기 요소를 만족하는 PROFIBUS-DP 네트워크을 구현 하였으며 그결과를 PROFIBUS-DP Class 2 Master 기능과 PLC CPU Memory Dump 기능을 하도록

제작한 C-Tool과 상용화된 Bus Monitor를 이용 하여 검증 하였다.

# 본 론

## 1. Conformance 설계

### 1.1 Data 중심 네트워크

공장 자동화의 피라미드 계층 구조에서 Shop Level 이상에서 사용되는 네트워크는 "Service"중심의 네트워크(이하 S-Network) 이다. 따라서 Mini-MAP이나 MAP에서는 공장 자동화 시스템에 사용되는 기기들을 제어하기 위해 86개의 Service를 정의한 MMS(Manufacturing Message Service)를 핵심으로[2] 으로 하고 있으며 Shop Level이상에서 사용되는 MAP이외의 모든 표준 네트워크들도 MMS를 포함하고 있다. 반면 공장 자동화의 피라미드 계층 구조에서 Shop Level밑에 위치하는 Sensor,Actuator Level을 위한 Field-Bus는 "Data" 중심의 네트워크(이하 D-Network) 이다.

S-Network은 대상 기기 제어를 위한 Transaction의 생성이 사용자 Application 책임하에 있게 된다. 따라서 네트워크 응용에 대한 사용자의 유연성이 대단히 높은 반면 실시간성이 떨어 진다. D-Network은 반대로 Transaction의 생성이 사용자 Application에 있지 않고 Network Kernel의 책임 하에 있게 된다. 따라서 네트워크 응용에 대한 사용자의 유연성이 떨어지는 반면 실시간성이 매우 향상 된다. 이러한 차이는 S-Network이 "Event-Driven"방식의 제어에 적합하도록 설계되었고 D-Network이 "Polling"방식에 의한 제어에 적합하도록 설계 된것에 기인한다.

D-Network의 Data Polling은 Input Data와 Output Data로 구성 되며 D-Network의 사용자는 Polling되는 I/O Data에 의해 D-Network System에 연결된 제어 대상 기기(이하 Slave)들을 제어 하게 된다. D-Network인 Field-Bus의 모든 정의는 I/O Data의 Polling을 위한 것이다. 따라서 Field-Bus의 Conformance설계는 I/O Data Interface를 의미 하며 아울러 I/O Data Polling의 관리,보수를 위한 보조 Data Interface를 포함 한다. 한편 Field-Bus 규약에 따라 I/O Data를 통한 Slave 제어를 보조하기 위해 극히 제한적인 Service Interface가 포함된다.

### 1.2 PROFIBUS-DP의 Conformance 설계

PROFIBUS-DP의 Conformance 설계는 Data Interface와 Service Interface를 모두 포함 하며 Data Interface는 Parameter Data Interface, I/O Data Interface, Diagnostic Data Interface로 구성 되고 Service Interface는 Global-Control Service와 Mark Service로 구성된다[3].

#### 1.2.1 Data의 물리적 이동

**(1) 물리적 Data 이동 방법의 결정**

PROFIBUS-DP의 Data Interface를 위해서 먼저 PROFIBUS-DP Master Module(이하 DPM)과 PLC CPU Module(이하 CPU)과의 물리적 Data 이동 방법(이하 Data 이동)을 정의 해야 한다. PROFIBUS-DP의 Data 이동은 Dual Port RAM을 이용하는 방법이 일반적이나[1] 본 논문에서는 DPM이 CPU Memory를 직접 Access하는 방법을 채택 하였다. 이는 Dual Port RAM을 사용하는 경우 보다 Data의 Consistency 확보가 용이 할뿐 아니라 Dual Port RAM 사용시 DPM의 재료비 상승및 일반적으로 Dual Port RAM이 온도(저온) 특성이 불량한점을 고려 했기 때문이다.

**(2) Data Consistency 보장**

Data 이동시의 Consistency는 임의의 순간에 Data Interface 영역을 DPM과 CPU가 동시에 Access하지 못하도록 하는 것이다. 동시 Access하는 경우 서로가 완전히 새롭게 갱신 되지 못한 Data를 얻게 되어 사용자 Application에 논리적 Error를 초래 하게 된다. 즉 Input Data의 경우 CPU Memory에 대한 DPM의 Write 동작이 끝나기 전에 CPU가 Read하는 경우 CPU는 New Data와 Old Data가 섞인 Input Data를 얻게 되어 Consistency가 깨지게 된다. 따라서 DPM과 CPU의 Data Interface 영역의 Access는 배타적 이어야 한다. 이를 위해 본 논문에서는 CPU의 Scan Code를 이용 하였다. CPU는 여러개의 단계로 구성되는 Loop를 계속적으로 Scan 하는 구조이며 매 단계마다 해당 Scan Code를 PLC System Bus에 내놓는다. DPM은 CPU가 Data Interface를 위한 영역을 Access하지 않는 단계에 있음을 CPU가 내놓는 Scan Code로 알 수 있고 이때 CPU Memory를 Access하므로써 CPU와 DPM간의 Data 이동시 Consistency를 보장 할 수 있다.

**(3) CPU Scan Code 사용을 위한 비동기 문제 해결**

CPU Scan Code를 이용하기 위해서는 DPM과 CPU의 비동기 문제를 해결 해야 한다. CPU의 Scan Code는 DPM의 동작 단계와 독립적으로 생성되므로 DPM이 Data Interface영역을 Access하고자 할때 CPU가 원하는 Scan Code를 내놓기를 기다려야 한다. 그러나 이러한 방법은 DPM의 Performance를 저하 하기 때문에 본 논문에서는 Interrupt 방식을 사용 하였다.

그림 1 CPU Interrupt 발생 구조처럼 DPM은 원하는 CPU Scan Code 값을 미리 예약하고 PLC System Bus를 통해 CPU가 내놓는 Scan Code를 감시 하다 Scan Code가 일치 하는 경우 Interrupt를 발생 시킨다 LN-DPM은 Interrupt Handler에서 Interrupt를 발생 시킨 Scan Code에

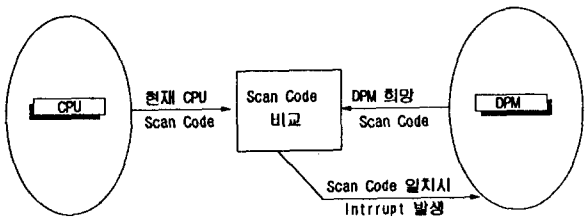



그림 1. CPU Interrupt 발생 구조.

해당하는 Data Interface를 실시한다.

### 1.2.2 Parameter Data Interface

Parameter Data는 DPM이 Slave들과 I/O Data를 교환하기 이전에 우선 설정 되야 할 값들로 Slave Parameter와 Bus Parameter가 있다[3]. Slave Parameter는 DPM이 Polling할 Slave와 I/O Data에 관한 정보 이며, Bus Parameter는 PROFIBUS-DP 네트워크를 통한 Data 송수신시 Bus 제어에 관한 정보이다. Parameter Data 표현에 대해서는 Slave Parameter의 Address Table를 제외 하고는 모두 표준안에 정의 되어 있다.

I/O Data와 Diagnostic Data는 DPM이 "On-Line"시 사용자와 무관하게 DPM Kernel에 의해 갱신 되는데 반해 Parameter Data는 "Off-Line"에서 Console을 통해 입력 된다.

#### (1) Address Table 정의

DPM이 Polling 하는 I/O Data와 CPU Memory를 연결(Mapping) 하는 방법에 대한 정의로 Convenience에 가장 밀접한 항목이다. 본논문에서는 I/O Data의 CPU Memory 할당시 사용자의 혼돈을 최소화 하기 위해 "Module"의 개념을 정의 하였다.

Module은 8 Word의 크기를 갖는 I/O Data CPU Memory 할당의 최소 단위이다. Module의 크기는 PLC I/O Card의 최대 크기를 기준으로 하였다. 따라서 PLC의 모든 I/O Card는 하나의 Module로 Mapping 된다.

(a) Module 단위 지정의 장단점및 채택 배경

I/O Data Interface의 영역은 중복 할당 되서는 안된다. 따라서 Address Table정의시 구현자는 중복 할당의 가능성을 최소화 할 수 있는 설계를 해야 한다. 본 논문에서 정의한 Module 단위 할당은 이러한 요구사항을 만족 시킨다.

CPU Memory의 최소 단위인 Word(16점)단위 지정은 사용자가 16점 이상의 I/O Card 사용시 Word의 Offset을 기억 해야 한다. Word 1에 128점의 I/O Card를 할당 한경우 사용자는 Word 2 부터 Word 8까지를 다른 I/O Card에 할당 할 수 없음을 기억 해야 한다. 즉 사용자는 항상 할당된 Word 번호와 I/O Card의 크기로 사용할 수 없는 Word를 계산 해야 한다. 이는 사용자 입장에서 매우 귀찮은 요소가 되며 작업 능률을 저하 시킨다.

Module은 사용할 수 있는 I/O Card의 최대 크기를 갖기 때문에 사용자는 CPU Memory 할당시 I/O Card의 크기에 따라 사용 할 수 없는 Module을 계산 하지 않아도 된다. 단 Module 사용시 128점 미만의 I/O Card에 대해서는 사용되지 않는 Memory 발생으로 I/O Data 이동시 Firmware에서 Module의 Offset을 계산 해야 하는 부담과 Word단위 사용시 보다 CPU의 I/O Data 영역이 커지는 손해가 발생하는데 이는 Memory 증가로 인한 재료비 상승 요인이 없고 편리성을 중요시 하는 사용자 경향을 고려 하여 Module 단위 할당을 채택 하였다.

#### (2) Console Interface

Console Interface는 표준안의 범주에 있지 않은 내용으로 PROFIBUS-DP 상품화시 구현자가 제품의 특성에 맞춰 정의해야 할 부분이다. Console Interface는 RS-232 Port를 이용하여 구현 되는데 본 논문에서는 사용자 편리성을 위해 CPU의 RS-232 Port를 이용 하도록 설계 하였다.

LN-DPM이 RS-232 Port를 내장 하는 경우 PLC 사용자는 CPU Program시와 LN-DPM Parameter Data설정시 Computer의 COM Port를 두개 사용 해야 하는 불편이 있으며 Note Book PC 등 하나의 COM Port를 갖는 Computer사용시 PLC쪽의 RS-232 Connector를 CPU와 LN-DPM에 번갈아 연결 해야 하는 불편이 따른다.

Console Interface는 Parameter Data 설정 기능 뿐만 아니라 DPM과 Slave의 Run, Stop을 원격제어 하여 PROFIBUS-DP 시스템 Tuning에 사용 되는 DP Class 2 Master의 기능을 PROFIBUS-DP 네트워크를 통하지 않고 RS-232 Port를 이용하여 사용 할 수 있도록 고안 되었다. 따라서 사용자가 DP-Class 2 Master 사용을 위한 비용 부담 없이 DP-Class 2 Master 기능을 이용 할 수 있도록 하였다 (DPM은 DP Class 1 Master임)

사용자가 Console을 이용하여 I/O Data 설정시 각각의 Slave가 가지고 있는 I/O 정보를 자동으로 인식 할 수 있다면(이하 Auto-Configuration) I/O Data 설정이 매우 용이해진다. 이 기능은 DP Class 2 Master에만 허용 되는데 본 논문에서 구현한 Console Interface가 DP-Master 2 기능을 갖기 때문에 LN-DPM 사용자는 Auto-Configuration 기능을 사용 할 수 있어 사용자 편리성을 한층 보강 하였다.

### 1.2.3 I/O Data Interface

DPM은 Parameter Data에 의해 정의된 I/O Data를 그림 2 I/O Data Flow에서 처럼 CPU, Slave와 교환 한다.

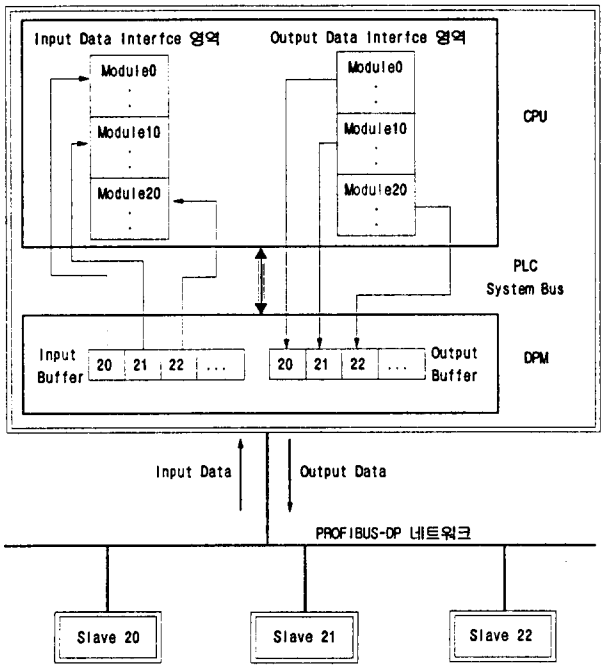



그림 2. I/O Data Flow.

### (1) I/O Data 크기 결정

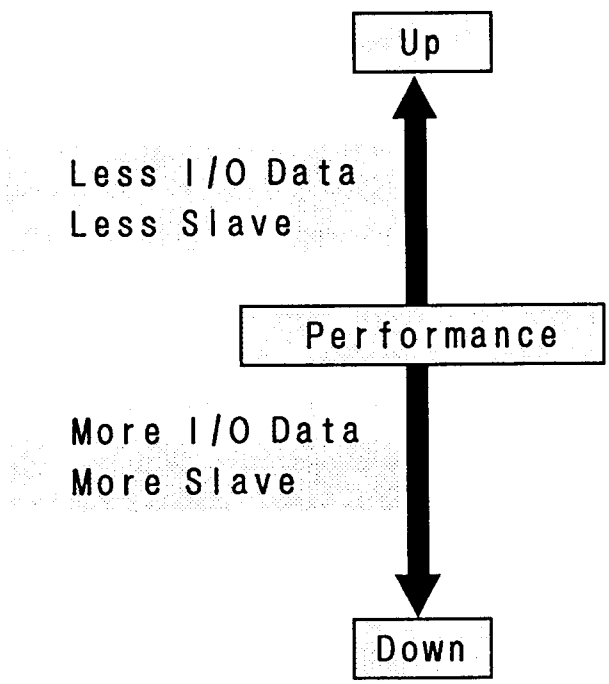
DPM이 Polling할 수 있는 I/O Data의 최대크기는 PROFIBUS-DP 시스템의 용량을 결정 하는 중요 요소이다. 본 논문에서는 PROFIBUS-DP 운용 권고 사항에 준하여 I/O Data 크기를 결정 하였다.

PROFIBUS-DP 시스템의 운용 권고 사항은 "Master는 최대 32 Byte Input과 32 Byte Output을 갖는 Slave를 최대 32대 까지 Polling 것이 효율적이다"[4] 라고 밝히고 있다. 이는 Polling하는 Slave의 수와 I/O Data의 양이 PROFIBUS-DP Performance를 결정하는 중요 요소이기 때문이다.

따라서 물리적으로 가능한 최대 Slave(126대)와 I/O Data 크기(Input 30744 Byte, Output 30744 Byte)를 갖도록 하는 것은 의미 없는 일이다. 본 논문에서는 권고 사항에 따라 DPM이 32 Byte Input과 32 Byte Output를 갖는 Slave를 32대 까지 Polling 할 수 있는 I/O Data 크기를 선택 하였다.

Input Data 크기 : 32 Byte * 32 Slave = 1024 Byte (8192점)

Output Data 크기 : 32 Byte * 32 Slave = 1024 Byte (8192점)




### (2) I/O Data Consistency

a. 출처 : PROFIBUS-DP Technical Description by PNO

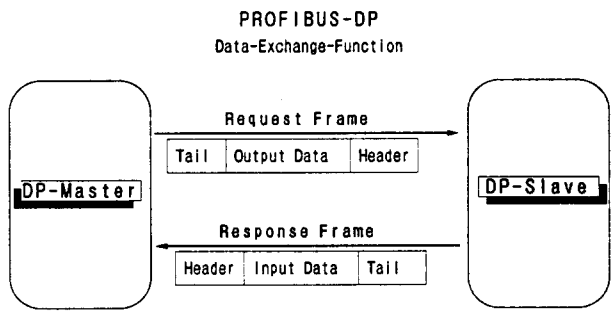



그림 3. User Data Exchange at PROFIBUS-DP.

Output Data는 CPU Interrupt Handler에의해 CPU Memory로 부터 DPM Output Buffer로 이동 되고 Data-Exchange Function의 Request Frame으로 DP-Slave에게 전달 된다. Input Data는 Data-Exchange Function의 Response Frame으로 DP-Slave로 부터 DP-Master인 DPM에게 전달 된다. DPM은 수신한 Input Data를 Input Buffer의 해당 영역에 저장 하고 CPU Interrupt Handler에 의해 CPU Memory로 이동 된다.(그림 1,2,3 참조) 이 과정에서 DPM의 I/O Buffer Handler(이하 Buffer Handler)와 CPU Interrupt Handler사이에 Data Consistency의 문제가 발생 된다. 즉 두개의 DPM Task가 동시에 I/O Buffer를 Access할 경우 "1.2.1 Data의 물리적 이동"에서 발생한 CPU와 DPM의 "Consistency 깨짐" 현상과 동일한 현상이 발생 하게 된다. 본 논문에서는 Buffer Handler와 CPU Interrpt Handler사이의 Consistency 보장을 위해 3

Buffer Mechanism을 도입 하였다.

(A) 3-Buffer Mechanism for Input Data

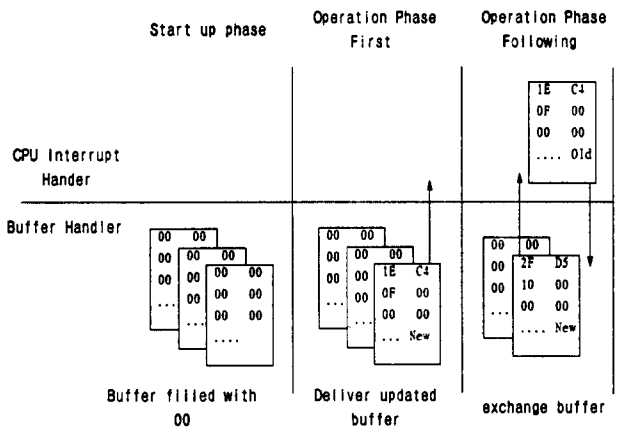



그림 4. 3 Buffer Mechanism for input data.

Input Data 3 Buffer Mechanism을 위해 본 논문에서는 i_ready, i_read, i_transmit, i_free의 buffer pointer 변수를 사용 하였다.

표 1. Input Buffer Pointer의 용도.

| Input Buffer | 용 도 | 초 기 치 |
|---|---|---|
| i_ready | Buffer Handler와 CPU Handler 간의 Bridge Pointer | NULL |
| i_read | Buffer Handler가 Slave로 부터 수신한 Input Data 를 저장 하는 Pointer | Buffer_1 |
| i_transmit | CPU Handler가 Data 이동을 위해 사용 하는 Pointer | Buffer_2 |
| i_free | Dummy Pointer | Buffer_3 |

Input Data를 위한 3 Buffer Mechanism의 알고리즘은 다음과 같다.

- Buffer Handler의 알고리즘

```
Begin
  Store Input Data in i—read
  Disable Interrupt
  if i_ready
  Then
  Begin
    i_free : = i_ready
  End
  i_ready : = i_read
  i_read : = i_free
  i_free : = NULL
  Enable Interrupt
End
```

- CPU Interrupt Handler의 알고리즘

```
Begin
  if i_ready
  Then
  Begin
    i_free : = i_transimit
    i_transmit : = i_ready
    Move Input Data from i_tranmit to CPU
      Memory
    i_ready : = NULL
  END
END
```

상기 알고리즘을 통해 Buffer Handler는 i_tranmit pointer를 CPU Interrupt Handler는 i_read pointer를 Access하지 않음로서 Input Data의 Consistency가 보장 된다.

(B) 3-Buffer Mechanism for Output Data

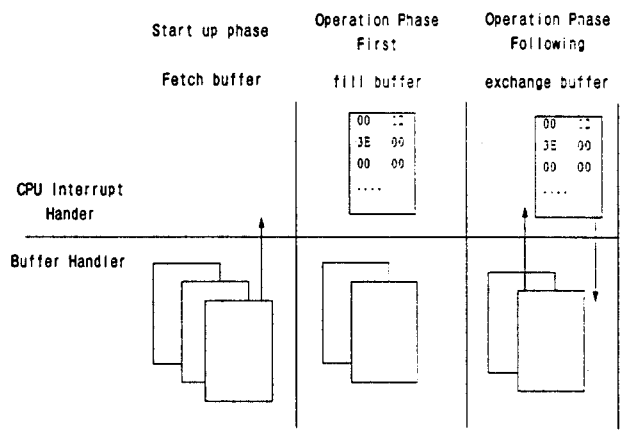



그림 5. 3 Buffer Mechanism for output data.

Output Data 3 Buffer Mechanism을 위해 본 논문에서는 o_ready, o_receive, o_write, o_free의 buffer pointer 변수를 사용 하였다.

Output Data를 위한 3 Buffer Mechanism의 알고리즘은 다음과 같다.

- Buffer Handler의 알고리즘

```
Begin
  Disable Interrupt
  if o_ready
```

표 2. Output Buffer Pointer의 용도.

| Output Buffer | 용 도 | 초 기 치 |
|---|---|---|
| o_ready | Buffer Handler와 CPU Handler 간의 Bridge Pointer | NULL |
| o_receive | CPU Interrupt Handler가 CPU Memory로 부터 읽어온 Output Data 를 저장 하는 Pointer | Buffer_1 |
| o_write | Buffer Handler가 Slave에게 Output Data를 전송하기 위해 사용 하는 Pointer | Buffer_2 |
| o_free | Dummy Pointer | Buffer_3 |

```
    Then
    Begin
      o_free : = o_write
      o_write : = o_ready
      o_ready : = NULL
    End
    Enable Interrupt
    Move Output Data from o_write to DP-Slave
  End
```

● CPU Interrupt Handler의 알고리즘

```
  Begin
    if o_ready = NULL
    Then
    Begin
      Move Output Data from CPU Memory to
        o_receive
      o_ready : = o_receive
      o_receive : = o_free
      o_free : = NULL
    END
  END
```

상기 알고리즘을 통해 Buffer Handler는 o_receive pointer를 CPU Interrupt Handler는 o_write pointer를 Access 하지 않음로서 Output Data의 Consistency가 보장 된다.

3 buffer mechanism은 Data Consistency 보장뿐 아니라 Performance 향상에도 기여 한다. 이는 2장 Performance 설계에서 설명 한다.

### 1.2.4 Diagnostic Data Interface

PROFIBUS-DP Application Program 작성시 사용자는 네트워크 비정상 상태로 인한 오동작 방지를 할 수 있어야 한다. 본 논문에서는 사용자가 네트워크 동작 상태를 쉽게 파악 할 수 있도록 표 3와 같은 Diagnostic Data를 CPU에 제공 한다.

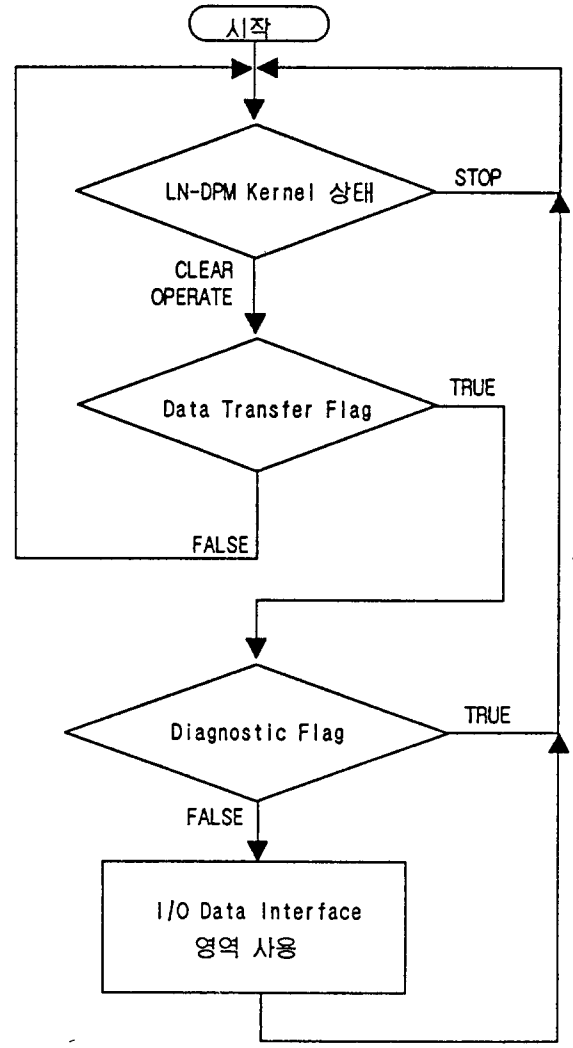



그림 6. Diagnostic Data을 이용한 사용자 Program.

표 3. Diagnostic Data 일람.

| Diagnostic Data | 의 미 |
|---|---|
| DPM Kernel 상태 | – STOP : I/O Polling을 하지 않음.<br>– CLEAR : Read Input Data<br>Write Output Data with "0"<br>– OPERATE : Read Input Data<br>Write Output Data |
| Diagnostic Flag | – Slave에 Diagnostic 발생을 표시<br>– 126개의 Flag로 구성 되어 어떤 Slave에 Diagnostic이 발생 하였는지 구별 가능 |
| Data Transfer Flag | – Slave와 I/O Data 교환 상태 표시<br>– 126개의 Flag로 구성 되어 어떤 Slave와 교환중인지 구별 가능 |
| I/O Update Time | – LN-DPM의 I/O Update 시간을 표시 |
| Bus 상태 Flag | – 네트워크의 Bus 제어 상태 표시<br>In-Ring Flag<br>Duplicate-Address Flag<br>Ring-Midified Flag<br>Time-Out Flag |

본 논문에서 제공하는 Diagnostic Data는 그림-6의 방법으로 자기진단 Programming에 사용 된다.

### 1.2.5 Service Interface

PLC 사용자들이 네트워크 명령어에 익숙치 않은 점을 고려 하여 본논문에서는 Service Interface를 CPU Status Register를 이용 하여 구현 하였다. CPU Register의 내용은 Data Interface와 동일하게 CPU Interrupt에 의해 DPM으로 전달 된다.Service Interface는 CPU Scan Code 4를 사용 한다.

(A) Global_Control Service

Global_Control Service는 Input Data의 입력동기, Output Data의 출력 동기 제어에 사용 한다.

표 4. Global_Control Service의 Register 구성.

| 구 성 | 의 미 |
|---|---|
| Primitive Register | – Request : To issue Global_Control Service<br>– Confirm : To terminate Global_Control transaction |
| Slave_Address Register | – Destination Address 지정 |
| Status Register | – Success, Fail 표시 |
| Command Register | – 입력 동기, 출력 동기 지정 |
| Group_Select Register | – Broadcasting, Multicasting 지정 |

(B) Mark Service

PLC Program과 DPM의 I/O Data Polling 동기를 위해 사용한다.

표 5. Mark Service의 Register 구성.

| 구 성 | 의 미 |
|---|---|
| Primitive Register | – Request : To issue Mark Service<br>– Confirm : To terminate Mark transaction |
| Status Register | – OK : Error 없는 I/O Polling 종료<br>– NOK : Error 있는 I/O Polling 종료 |

## 2. Performance 설계

PROFIBUS-DP의 Performance는 DP-Master가 I/O Data 갱신을 얼마나 빠르게 할 수 있는가로 귀결된다. I/O Data 갱신주기는 그림-3의 "Data-Exchange Function" Frame의 전송 시간과 LN-DPM의 I/O Data 처리 시간및 Bus 제어 시간의 합으로 결정 된다. 이중 Frame의 전송 시간은 Baud Rate, Data의 크기및 Slave의 수에 따라 산술적으로 계산 되는 값으로 최적화의 대상은 아니다. 따라서 본 논문에서는 DPM I/O Data 처리 시간및 Bus 제어 시간를 최적화의 대상으로 한다.

### 2.1 CPU와 I/O Data 갱신 주기의 관계

PLC(중형)의 주 사용 Scan Time 범위는 20ms 에서 30ms 이다. 따라서 20 ms 이하의 갱신 주기를 갖는 DPM의 I/O Data는 실제로 CPU 사용 범위안에 있지 않기 때문에 의미 없게 된다. 한편 사용 될 수 없는 I/O Data 갱신을 위한 "Data-Exchange Frame"은 불필요한 Traffic을 증가시켜 네트워크 오류 가능성을 증가 시키므로 본 논문에서는 DPM의 논리적 최소 I/O Data 갱신 주기를 10ms로 제한하였다. 이는 PROFIBUS-DP의 I/O Data Polling 시작 제어 Timer인 "Min_Slave_Interval Timer"의 값을 10ms로 설정 하므로써 가능 하다.

### 2.2 I/O Data 처리 시간의 최적화

I/O Data 처리 시간의 최적화는 그림-2 I/O Data Flow에서 처럼 이동 하는 I/O Data 이동 시간의 최소화및 CPU Interrupt 주기 조절로 이루워 진다.

(A) Data 이동 시간의 최소화

Data 이동 시간의 최소화는 Data 이동 개체간에 동일 Data의 복사 회수를 줄이는 것과 중복 이동을 방지 하는 것이 관건이다. I/O Data 이동의 개체는 CPU, DPM Buffer, DPM Kernel Buffer(DP의 실제 송수신 Buffer)로 구성된다. 3 개체간 Data 이동은 1.2.3 I/O Data Interface에서 설명한 3 Buffer Mechanism 사용시 실제 Data 복사가 CPU와 DPM Buffer간에 1회만 발생한다. 즉 1회의 Data 복사 횟수를 줄임으로써 최대 I/O Data Polling시 매 주기마다 약 3.6 ms 의 절감 효과가 있다.

3 Buffer Mechanism 사용으로 CPU Interrupt Handler는 i_ready와 o_ready (CPU Interrupt Handler 알고리즘 참조)의 값으로 CPU Interrpt Instance마다 CPU와의 Data 이동의 필요성을 판단 할 수 있다. 즉 Input Data의 경우 i_ready가 NULL이면 새로운 Input Data가 수신 되지 않은 경우 이므로 CPU Interrput Handler는 이전의 Instance에서 이동된 Input Data를 중복 이동 하지 않는다. Output Data의 경우 o_ready가 NULL이 아니면 이전의 Instance에서 이동한 Ouput Data가 출력 되지 않은 경우 이므로 CPU Interrupt Handler는 CPU로 부터 Output Data를 중복 이동 하지 않는다.

본논문에서 사용한 3 Buffer Mechanism은 간단한 Code로 구현이 가능하여 알고리즘 수행의 Over-Head를 발생치 않고 I/O Data Consistency 보장및 I/O Data 이동 시간을 절감 함으로써 DPM Conformance와 Performance 설계에 중요한 역할을 한다.

(B) CPU Interrupt 발생 주기 조절

LN-DPM의 I/O Data 갱신 주기 보다 빠른 CPU Interrupt 주기는 갱신될 I/O Data가 없는 상황에서 CPU Interrupt Handler가 CPU Bus 제어권및 DPM u-processor 사용권을 점령 하기 때문에 CPU와 DPM의 Performance를 저하 시킨다. 본논문에서는 불필요한 CPU Interrupt의 발생을 방지 하기 위해 CPU Interrupt의 휴면(SLEEP) Mode를 사용 하였다.

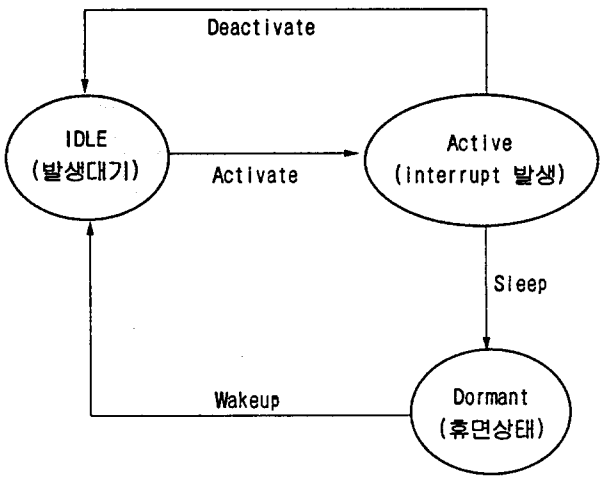
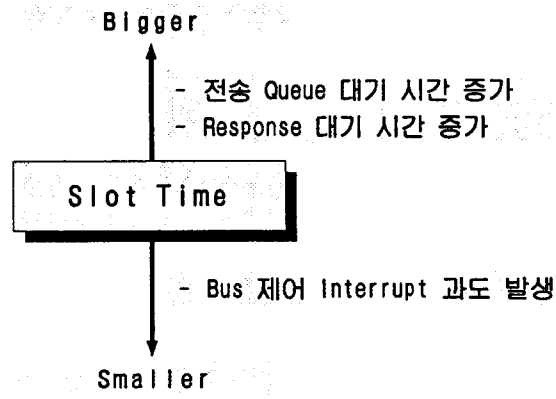



그림 7. CPU Interrupt 상태 변환.

### 2.3 Bus 제어 시간의 최적화

Bus 제어 시간은 Bus Parameter중 Slot Time에 의해 제어 된다. Slot Time은 LN-DPM의 Request Frame 전송 주기및 Response Frame 대기 시간을 제어 한다. Slot Time이 매우 작은 경우 DPM은 전송할 Request Frame이 없는 상황에서 전송 Queue 검색을 위한 Bus 제어 Interrupt의 빈번한 발생으로 DPM 다른 Task의 수행 속도가 떨어져 DPM Performance가 저하 된다. 한편 Slot Time이 매우 큰 경우 Bus 제어 Interrupt 발생 지연으로 인한 Request Frame의 전송 Queue 대기 시간이 길어 지고 한편 Response 대기 시간도 증가 하여 역시 DPM의 Performance를 저하 시킨다.

본 논문에서는 500 kbps에서 DPM의 최대 I/O Data 양을 8대의 Slave에 분산 배열한 구성에서 Slot Time을 1 ms 에서 8 ms 까지 변경 하며 시험한 결과 2 ms 에서 가장 좋은 I/O Update Time을 얻을 수 있었다. 2 ms 이하 에서




는 과도한 Bus 제어 Interrupt를 경험 했고 2 ms 이상 에서는 전송 Queue 대기 시간이 증가함을 경험 했다. 한편 실험 결과 전송 Queue 대기 시간 증가 요인 보다는 Bus 제어 Interrupt의 과도한 발생이 Performance 저하에 더 치명적임을 목격 했다. 따라서 본 논문에서는 DPM의 최저 Slot Time을 2 ms로 제한 하였다.

## 3. CPU Link 구현

PROFIBUS-DP 네트워크은 그림-2에서 처럼 CPU가 DP-Master(LN-DPM)을 통해 Slave와 I/O Data 교환을 가능케 하여 준다. 그러나 실제 사용자 Application에서는 I/O Data Link 뿐만 아니라 CPU와 CPU 사이에 CPU Memory를 공유 할 수 있는 CPU Link도 필요하다. 일반적으로 PLC 네트워크는 I/O Link와 CPU Link를 위한 별개의 네트워크로 제공 된다.

CPU Link는 "Event-Driven"방식의 Service 중심인 S-Network 형태로 구현 되는 것이 옳으나 실제로 활용도 측면에서 보면 Data 중심의 D-Network으로도 충분한 예가 많기 때문에 CPU Link도 D-Network 형태로 구현 되는 경향이 있다.따라서 Vendor 입장에선 동일한 형태의 네트워크을 두종 개발 하는 Burden이 있고 사용자 입장에선 두종의 네트워크 구입으로 인한 Cost증가및 Application이 복잡해지는 Burden이 있다.

본 논문에서는 Parameter Data 설정시 CPU Module을 64 Word Input과 64 Word Output를 갖는 특수 I/O Module로 정의 가능토록 하여 DPM이 CPU의 Memory를 마치 I/O Module 처럼 인식 하므로써 CPU Link를 가능 토록 하였다. 이를 위해 위해 DC-Slave의 개념을 도입 하였다.

### 3.1 DC-Slave 구별

DPM은 DP-Slave Module(이하 DPS)로 부터 I/O Data를 Polling 한다. 이때 DPS는 자신이 I/O Data Link를 위한 I/O Slave인지 CPU Link를 위한 DC-Slave 인지를 식별 해야 한다. DC-Slave와 IO-Slave의 구분은 사용

자의 목적에 따르기 때문에 이의 구별도 역시 사용자가 해야 한다. 그러나 이 구별을 위한 DIP Switch 선택등은 사용자에게 불편을 주기 때문에 본논문에서는 LN-DPS의 Slot 위치로 자동 인식 하도록 하였다. 즉 Slot 0에 장착된 DPS는 DC-Slave로, 그외의 Slot에 장착된 DPS는 IO Slave로 동작 한다.

### 3.2 DC-Slave의 특성

CPU Link은 일반적으로 "Peer-to-Peer"인 반면 DC-Slave를 이용한 CPU Link는 "Master-Slave"의 Polling 방식만 가능 하다. 즉 DC-Slave는 DP-Master의 Polling에 의하지 않고는 자의적으로 Data를 송신 할 수 없다. 그러나 Data 중심의 CPU Link에서는 "Peer-to-Peer" 방식이 Critical 요인이 되지 않으며 LN-DPM의 I/O Data를 이용하는 CPU Program에 의해 "Peer-to-Peer" 방식을 모사 할 수 있기 때문에 DC-Slave는 CPU Link로 충분히 사용 가능 하다. 더우기 DC-Slave는 DPM에 의해 제어

되는 Bus상에 직접 접속 되기 때문에 동일한 배선으로 CPU Link와 I/O Link를 동시에 제공 하여 사용자의 네트워크 사용 비용 절감및 용이성을 증대 시킨다.

## 결 과

본논문에서는 결과 고찰을 위하여 DP-Class 2 Master 기능과 CPU Memory Dump 기능을 하는 Tool(이하 C-Tool)을 제작 하였으며 사용화된 Bus Monitor를 사용 하였다.

## 1. Conformance 구현 고찰

### 1.1 Parameter Data Interface

#### 1.1.1 고찰 항목및 방법

- Parameter Data가 LN-DPM의 Console Interface를 통하여 LN-DPM에 설정 되는가 ?

  방법 : C-Tool을 이용 하여 작성한 Parameter Data를 CPU RS-232 Port로 전송시 DPM이 정상적으로 반응 하는가를 고찰 한다.

- DPM이 설정된 Parameter Data에 준거 하여 I/O Data를 Polling 하는가?

  방법 : Bus Monitor로 Capture한 Frame을 분석 하여 설정된 I/O Data 양과 Slave수, Baud Rate, Slot TIme과 일치 하는 Traffic을 DPM이 발생 시키는가를 고찰 한다.

#### 1.1.2 고찰 결과

(A) Console interface 동작

C-Tool을 이용 하여 작성된 다양한 조합의 Parameter Data가 DPM에 설정 되는데 문제가 발생 하지 않았다. 즉 Parameter Data 설정을 위해 정의한 Console Interface가 항상 정상 동작 한다는 결과를 얻었다.

(B) Parameter Data에 준거한 I/O Data Polling

표 6 시험용 Parameter Data를 DPM에 설정한후 Bus Monitor의 Capture Frame으로 DPM의 동작 여부를 판단 할 수 있다.

표 6. 시험용 Parameter Data.

| Parameter Data | 설 정 값 |
|---|---|
| Slave 별 I/O Data 크기 | Slave 21 (15H) - Output 2 byte<br>Input 2 byte<br>Slave 22 (16H) - Output 3 byte<br>Input 3 byte<br>Slave 23 (17H) - Output 2 byte<br>Input 2 byte<br>Slave 24 (18H) - Output 3 byte<br>Input 3 byte<br>Slave 25 (19H) - Output 2 byte<br>Input 2 byte |
| Baud Rate | 500 kbps |
| Slot Time | 2 ms |

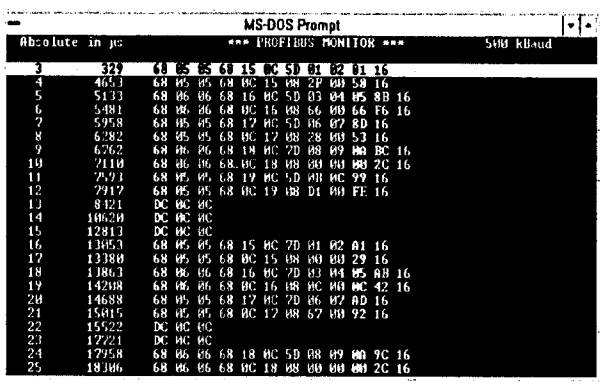



그림 8. Bus Monitor Capture Frame (I/O Data는 "68"(offset 0)로 시작하는 Frame에서 Offset 7부터 시작 되고 마지막 두 byte는 제외 된다.)

PROFIBUS-DP의 Frame Length는 Data 길이에 9를 더하면 된다. 따라서 Slave 21 관련 Frame은 11 byte, Slave 22 관련 Frame은 12 byte, Slave 23 관련 Frame은

11 byte,Slave 24 관련 Frame은 12 byte, Slave 25 관련 Frame은 11 byte 로 구성 되면 정상 이다. 이 결과를 그림-8 Bus Monitor Capture Frame 에서 확인 할 수 있다. Frame Number 3,4는 Salve 21, Frame Number 5,6는 Salve 22, Frame Number 7,8는 Salve 23, Frame Number 9,10는 Salve 24, Frame Number 11,12는 Salve 25의 Frame으로 모두 정상적인 Frame이다. 한편 그림-8의 우측 상단에서 500 kbps상의 Data 전송임을 확인할수 있고 (DC 0C OC)로 구성 되는 Token Frame이 약 2ms 간격으로 발생 됨으로 LN-DPM이 2ms로 설정된 Slot Time으로 동작함을 알 수 있다. 이러한 결과는 LN-DPM이 설정된 Parameter Data에 의해 정상 동작함을 증명 한다.

### 1.2 I/O Data Interface

#### 1.2.1 고찰 항목및 방법

- CPU Memory의 Output Data가 정확히 Request Frame에 표현 되는가?
  방법 : C-Tool을 이용 하여 CPU Memory에 쓴 Output Data가 Bus Monitor에 Capture된 Request Frame에 정확히 표현 되있는가를 고찰 한다.
- Response Frame의 Input Data가 CPU Memory에 정확히 쓰여 지는가?
  방법 : Bus Monitor로 Capture한 Response Frame의 Input Data가 CPU Memory에 쓰여 졌는가를 C-Tool의 CPU Memory Dump 기능으로 확인 한다.

상기 고찰로 I/O Data의 물리적 이동및 I/O Data Consistency를 확인 할 수 있다.

#### 1.2.2 고찰 결과

(A) Output Data 이동및 Consistency

C-Tool로 CPU Memory에 쓴 (1,2), (3,4,5), (6,7), (8,9,A), (B,C)의 Data가 Slave 21부터 Slave 25의 Request Frame에 정확히 표현 되고 있음을 그림-8 에서 확인 할 수 있다. 따라서 Output Data의 물리적 이동및 Output Data에 대한 3 Buffer Mechanism이 정확히 동작함을 알 수 있다.

(B) Input Data 이동및 Consistency

그림 8의 Response Frame에 표현된 Input Data (2F, 00), (66,00,66), (28,00), (00,00,00), (D1,00)이 CPU Memory Input영역의 정확히 쓰여짐을 C-Tool의 CPU Memory Dump기능으로 확인 하였다. 따라서 Input Data의 물리적 이동및 Input Data에 대한 3 Buffer Mechanism이 정확히 동작함을 알 수 있다.

## 2. Service Interface 구현 고찰

### 2.1 고찰 항목및 방법

- 표-4의 Global_Control Service 구성 Register의 조작으로 입력 동기및 출력 동기를 제어 할 수 있는가?
  방법 : C-Tool을 이용 하여 해당 Register 설정후 Slave 출력 Card의 동시 출력 상황및 CPU Input 영역의 동시 갱신을 관찰 한다.
- 표 5의 Mark Service 구성 Register의 조작으로 Polling 주기를 식별 할 수 있는가?
  방법 : C-Tool을 이용 하여 해당 Register 설정후 Confirm Primitive 수신이 가능한가를 고찰 한다.

### 2.2 고찰 결과

(A) Global_Control Service

C-Tool을 이용한 Global_Control Transaction 발생으로 Slave Output Card의 동시 출력을 관찰 할 수 있었고 Input Data의 동시 입력도 관찰 되었다. 반복 되는 시험으로 이과정에서 특이한 Error가 발생 하지 않으므로 Global_Control Service Interface가 정상적으로 동작함을 알 수 있다.

(B) Mark Service

C-Tool을 이용한 Mark Transaction 발생후 Primitive Register를 통해 Confirm Primitive를 정상적으로 수신 하였고 Error 상황 연출시 Status Register로 NOK Code를 수신 할 수 있었다. 한편 정상 상황에서 OK 이외의 Code를 수신 할 수 없었다. 반복 되는 시험으로 이과정에서 특이한 Error가 발생 하지 않으므로 Mark Service Interface가 정상적으로 동작함을 알 수 있다.

## 3. Performance 구현 고찰

PROFIBUS-DP의 Performance는 I/O Data의 갱신 주기로 판단한다. PROFIBUS-DP의 I/O Data 갱신 주기는 사용자의 네트워크 구성에 따라 가변적이므로 일괄적으로 Performance의 범주를 정의 할 수는 없으나 Small, Medium, Large System으로 구분하여 500 kbps 상에서 갱신 주기가 표 7의 범주안에 있으면 중형 PLC의 Field-Bus System으로 실시간성을 만족 한다고 할 수 있다[3].

### 3.1 고찰 항목및 방법

- 각 System의 갱신 주기가 표-7의 목표 갱신 주기를 만족 하는가?
  방법 : 각 System 요건에 맞는 네트워크를 구성하여

표 7. System 크기에 따른 목표 갱신 주기.

| 시스템 구분 | Slave 수 | I/O Data 크 기 | 목 표 갱신 주기 | 비 고 (AB PLC) |
|---|---|---|---|---|
| Small System | 5 | 2560점 | 20 ms | 127 ms |
| Medium System | 16 | 8192 점 | 45 ms | 387 ms |
| Large System | 32 | 16384점 | 100 ms | 1071 ms |

갱신 주기를 Bus Monitor의 Time Stamp로 확인한다. 단 Medium과 Large System은 8개의 Salve에 해당 I/O 크기를 설정한후 부족한 Slave수에 대해서는 Salve당 지연 시간을 가산하여 실제 갱신 주기를 구한다.

Slave 당 지연 시간 =

((Slave 당 Over-Head Byte 수 * UART Bit 수) * BIT Time) * Frame 수 = ((9 * 11) * 2) * 2= 396 us (약 0.4 ms)

### 3.2 Small System 고찰 결과

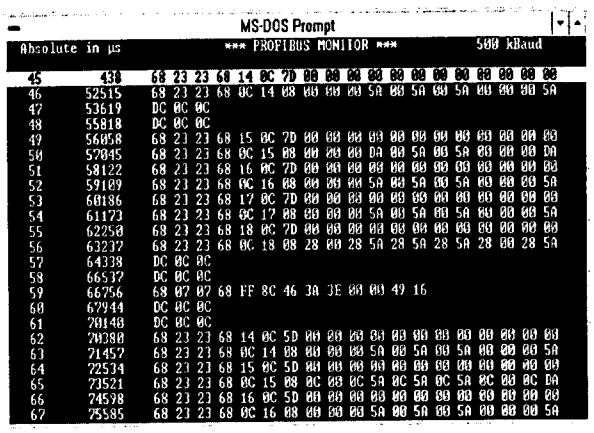



그림 9. Small System Bus Monitor Capture Frame.

그림-9의 Frame Number 62와 45의 시간 간격이 Small System의 갱신 주기가 된다.

Small System 갱신 주기 =

70380 us - 51438 us = 18942 us (약 19 ms)

본논문에서 구현한 PROFIBIS-DP 시스템은 Small System의 목표 갱신 주기를 만족한다. 따라서 10ms 단위의 고속의 처리 속도를 요구 하는 Sensor, Actuator 제어에 사용 할 수 있다.

### 3.3 Medium System 고찰 결과

그림-10의 Frame Number 60과 38의 시간 간격에 Slave당 지연 시간의 합을 가산 한것이 Medium System의

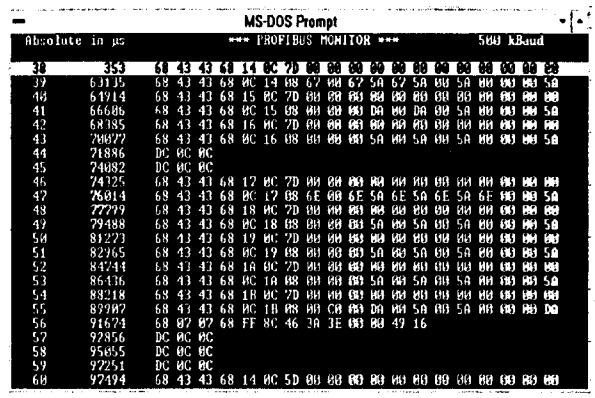



그림 10. Medium System의 Bus Monitor Capture Frame.

갱신 주기가 된다.

Medium System의 갱신 주기 =

(97494 us - 61353 us) + (0.4 * 8) = 39341 us (약 40 ms)

본논문에서 구현한 PROFIBUS-DP System은 Medium System의 목표 갱신 주기를 만족 한다. 대부분의 상용화된 Remote I/O System들은 PROFIBUS-DP Medium System 규모를 제어 하기 위해 100ms 이상이 소요 되는 것을 감안 하면 본 시스템의 갱신 주기는 매우 뛰어나다고 할 수 있다.

### 3.4 Large System 고찰 결과

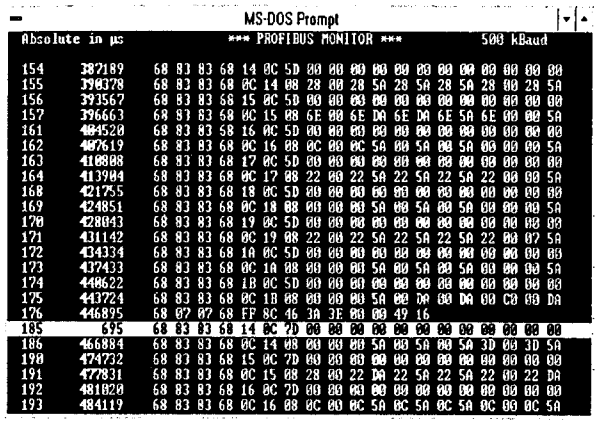



그림 11. Large System의 Bus Monitor Capture Frame.

그림 11의 Frame Number 185와 154번의 시간 간격에 Slave당 지연 시간의 합을 가산 한것이 Large System의 갱신 주기가 된다.

Large System의 갱신 주기 =

(463695 us - 387189 us) + (0.4 us * 24) =

86106 us (약 86 ms)

본논문에서 구현한 PROFIBUS-DP System은 Large System의 목표 갱신 주기를 만족 한다. 따라서 16000점 대의 대규모 용량 Application을 실시간으로 제어 가능 하다.

Performance 고찰 결과 본 시스템은 Small System에서 Large System에 걸쳐 실시간 제어가 가능한 것이 관찰 되었다.

## 참고문헌


[1] Klaus Bender, The Fieldbus for Industrial Automation, Prentice Hall, 1993

[2] ISO 9506 MMS, ISO/TC 184/SC 1, 1990

[3] DIN 19245 Part 3 PROFIBUS-DP, PNO, 1994

[4] PROFIBUS Technical Description, PNO, 1993

[5] AB Processor Manual, ALLEN-BRADLEY


## 저 자 소 개


**최 명 수**

1989년 한양 대학교 전자 통신공학 졸업

1989~현재 삼성항공 정밀기기연구소 근무

1990년 : IEEE 802.2 LLC 개발

1992년 : Mini-MAP MMS 개발

1993년 : FAIS-MAP MMS 개발

1995년 : PROFIBUS-DP 개발

공장 자동화용 국제 표준 네트워트,공정 제어등이 주관심 분야임.

(462-121) 경기도 성남시 중원구 상대원1동 145

TEL. 0342) 40-8471 / FAX. 0342) 40-8411