Proceedings of the 12th
KACC, October 1997

PROFIBUS 에서 대역폭 할당 기법 구현

김지용*, 홍승호**

*한양대학교 제어계측공학과(Tel:+82-345-400-4084; Fax:+82-345-406-6639; E-mail: sandrain@shinbiro.com)
**한양대학교 제어계측공학과 제어계측공학과(Tell:+82-345-400-5213;Fax:+82-345-406-4632;E-mail: shhong@hyunp2.hanynag.ac.kr)

Abstracts Fieldbuses are used as the lowest level communication network for real-time communication in factory automation and process control systems. Data generated from field devices can be devided into three categories: sporadic real-time, periodic real-time and non real-time data. Since these data share one fieldbus network medium, it needs a method that allocate the limited bandwidth of fieldbus network to the sporadic real-time, periodic real-time and non real time traffic. This paper introduces an implemetation method of bandwidth allocation scheme introduced in [5] on PROFIBUS. Using the modified PROFIBUS FDL(Fieldbus Data Link layer), the bandwidth allocation scheme introduced in [5] is verified by the experiments.

Keywords fieldbus, PROFIBUS, badwidth, allocation, sporadic real-time, periodic real-time, non-real-time

1. 서론

필드버스는 공장 자동화나 분산 제어 설비에서 사용되는 컴퓨터 통신망 구조중에서 최하위 계층의 필드 장비들간에 실시간 통신을 담당하는 산업용 네트워크이다. 필드에서 생성되는 데이터들은 산발적 실시간, 주기적 실시간, 비실시간 데이터의 세 가지 종류로 구별할 수 있다. 필드버스에서는 이러한 데이터들이 하나의 미디엄을 공유하게 되므로 데이터들을 관리하고 제어하기 위한 적절한 방법이 필요하게 된다.[6]

본 연구에서는 [5]에서 제시한 대역폭 할당 기법을 PROFIBUS[2,3]에 적용하기 위한 방법을 제시하고, 이를 통하여 대역폭 할당 기법의 타당성을 실험적으로 검증하고 분석 한다. 본 논문 2장에서는 대역폭 할당 기법의 알고리듬을 기술한다. 3장에서는 PROFIBUS를 기반으로 하여 대역폭 할당 기법을 적용한 FDL(Fieldbus Datalink Layer)을 구현한 하드웨어 및 소프트웨어의 구조에 대해서 기술한다. 4장에서는 본 연구를 통하여 구현된 FDL을 통하여 대역폭 할당 기법에 대한 타당성을 실험적으로 검증하고, 5장에서는 결론 및 추후 연구 사항에 대해서 기술한다.

2. 대역폭 할당 기법

[5]에서 제시한 대역폭 할당 기법에서는 산발적 실시간, 주기적 실시간, 비실시간 데이터들을 처리하기 위해서, 필드버스에서 이용할 수 있는 대역폭을 주기적 구간과 비주기적 구간으로 나눈다. 주기적 구간에서 주기적 실시간 데이터들에 대한 네트워크 대역폭 할당 문제는 윈도우 스케쥴링 알고리듬[1]을 적용하고, 비 주기적 구간에서는 비실시간 데이터들을 처리한다. 산발적 실시간 데이터가 생성된 경우에는 가장 우선적으로 전송하도록 한다. 윈도우 스케쥴링 알고리듬은 N개의 주기적 실시간 데이터를 생성하는 노드가 $r(r \le N)$개의 윈도우를 서로 동적으로 할당하면서 공유하도록 하여 주기적 구간 동안 샘플링 되는 데이터의 개수가 r개의 윈도우 개수를 초과하지 않도록 각 제어 루프에서의 샘플링 주기를 스케쥴링하는 것이다.

본연구는 한국과학재단 핵심 전문 연구비 지원에 의하여 수행되었음 (과제번호:961-0924-138-2)

[5]에서 제시한 알고리듬을 정리하면 아래와 같다.

Given:

N: 전체 노드 개수

M: 제어 루프 개수

N_c: 산발적 실시간 데이터를 생성하는 노드 개수

N_a: 비실시간 데이터를 생성하는 노드 개수

N_p: ($=2M$) 주기적 실시간 데이터를 생성하는 노드개수

L_c: 산발적 실시간 데이터의 전송 시간

L_a: 비실시간 데이터의 전송 시간

δ: 서버 오버헤드

R: ($=N\sigma$) 서버 순환 시간

[Φ_i, i=1에서 Nc]: 최대 허용 루프 지연 시간

Φ_c^m: 산발적 실시간 데이터의 최대 허용 데이터 지연시간

[λ_c^i, i=1에서 N_c]: 산발적 실시간 데이터의 패킷 도착 빈도

[λ_a^i, i=1에서 Na]: 비실시간 데이터의 패킷 도착 빈도

[T_i, i=1에서 M]: 제어 루프 i에서 데이터 샘플링 주기

Step 1: 주기적 실시간 데이터의 샘플링 주기 T_i를 구한다.

$$\Phi_1 = \min[\Phi_i, i=1 \text{에서} M]$$

$$T_1 = \frac{\Phi_1 + L_p}{3} \quad (1)$$

$$k_i = \left\langle \frac{\Phi_i - (T_i - L_p)}{2T_1} \right\rangle, \forall i = 1 \text{에서} M \quad (2)$$

$$\alpha_K = 2\sum_{i=1}^{M} \frac{1}{k_i} \quad (3)$$

$$r = \lceil \alpha_K \rceil \quad (4)$$

If $rL_p + N_c L_c + R > T_1$, then

(주기적 실시간 데이터 트래픽의 과부하

⇒ M 또는 N_c 감소: goto Step 1)

Else

$$T_i = k_i T_1$$

Step 2: 각 노드에서 첫 번째 주기적 실시간 데이터 샘플링 순간 t_i를 구한다.

$$t_1 = A_1 = 0$$

For ($j=2; l=1;\ j \le N_p,\ i \le N_p,\ l \le k_M,\ j=j+1,\ l=l+1$)

$$t_j = inf\,[A_i \geq A_{i-1};\ u^j(Al) \leq r]$$

Step 3: 비실시간 데이터의 데이터 패킷 길이 L_a를 구한다.

$$L_a = \begin{cases} \dfrac{\Phi_c^m - (rL_p + N_c L_c + R)}{N - r + 1}, \Phi_c^m < T_1 \\ \dfrac{T_1 - (rL_p + N_c L_c + R)}{N - r + 1}, otherwise \end{cases} \quad (5)$$

$$\lambda_a' = \lceil L_a'/L_a \rceil \Lambda_a' \quad (6)$$

$$\lambda^m = \max[\lambda_c', \lambda_a', i = 1 \text{에서} N_c, j = 1 \text{에서} N_a]$$

$$\text{If } \lambda^m \geq \frac{1 - L_c \sum_{i=1}^{N_c} \lambda_c' - L_a \sum_{j=1}^{N_a} \lambda_a' - [(\alpha_K L_p + R)/T_1]}{R}, \text{ then} \quad (7)$$

(산발적 실시간 또는 비실시간 트래픽의 과부하

⇒ N_c 또는 N_a 감소: goto Step 3)

(주기적 데이터 전송큐 공존

⇒ 해당 노드 감소: goto Step 1)

End

제시한 알고리듬은 필드버스에서 제공하는 대역폭안에서 산발적 실시간, 주기적 실시간, 비실시간 데이터가 원하는 제한 시간 이내에 전송되도록 하기 위하여 주기적 데이터의 샘플링 주기와 비 주기적 데이터의 패킷 길이를 적절히 조정한다. 그림 1에서는 대역폭 할당 기법으로 필드버스에서 이용할 수 있는 대역폭을 적절히 분배한 예를 보여 준다.

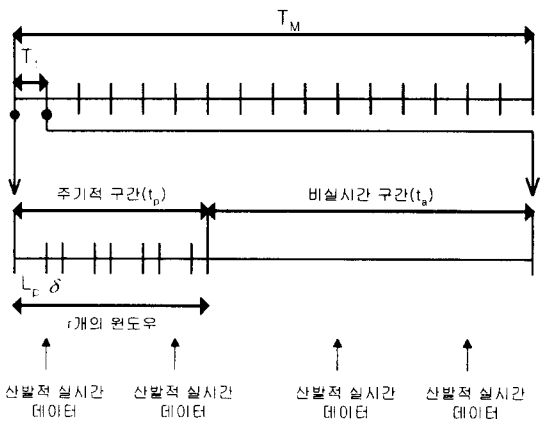

그림 1. 필드버스 대역폭 할당의 예

Fig. 1. a example of the fieldbus bandwidth allocation

3. 프로토콜의 구현

본 연구에서 FDL 프로토콜은 소프트웨어로 구현되었으며, 이를 통하여 PC 보트와 센서/엑튜에이터용 독립 보드를 개발하였다. FDL 프로토콜은 PROFIBUS의 FDL 프로토콜을 그대로 구현하고, 이에 대역폭 할당 기법에 관련된 기능을 추가하였다.

3.1 소프트웨어의 구조

PROFIBUS FDL은 일반적으로 ASIC을 이용한 하드웨어와 소프트웨어를 혼합한 형태로 구현된다[4]. 그러나 본 연구에서는 수정된 프로토콜을 사용하기 때문에 상용화된 ASIC을 이용하지 않고, 소프트웨어적으로 프로토콜을 구현하는 방법을 사용하였다.

3.1.1 데이터 전송 서비스

PROFIBUS FDL 데이터 전송 서비스에는 SDA, SDN, SRD 전송 서비스가 있다. 소프트웨어로는 FDL 사용자가 전송하고자 하는 요구를 아래의 frame 구조체로 만들어서 Low 우선순위 전송큐 또는 High 우선순위 전송큐에 삽입되도록 구현 하였다. FDL은 또한 물리계층에서 수신된 프레임을 FDL은 frame 구조체로 정리한 다음, Low 우선순위 수신큐와 High 우선순위 수신큐에 넣는다. FDL 사용자는 Low 우선순위 수신큐 카운터와 High 우선순위 수신큐 카운터를 체크하고, 카운터가 0이 아니면 큐에서 frame 구조체를 꺼내온다.

```
typedef struct frame_tag
  {
  unsigned char Service;      //-- 전송 서비스의 종류
  unsigned char SSap;         //-- SAP
  unsigned char DSap;         //-- 상대편 SAP
  unsigned char Rem_add;      //-- 상대편 주소
  unsigned char Loc_add;      //-- 주소
  unsigned char Update_Status; //-- for SRD
  unsigned char L_Status;     //-- 에러 표시
  unsigned char Length;       //-- 데이터 길이
  unsigned char Data[246];    //-- 데이터
  } frame;
```

3.1.2 타이머

PROFIBUS FDL에서는 Token Rotation Timer, Gap Update Timer, Idle Timer, Slot Timer, Syn Interval Timer, Time-out Timer의 타이머가 필요하다. 본 연구의 프로토콜 구현에 사용된 V25는 2개의 타이머를 가지고 있으므로 0번 타이머를 Token Rotation Timer로 사용하고, Gap Update Timer는 Token Rotation Time의 배수로 동작하므로 0번 타이머의 인터럽트 함수에서 처리하도록 한다. 1번 타이머는 Slot Timer로 사용하고, Syn Interval Timer와 Time-out Timer는 Slot Time의 배수로 동작하므로 1번 타이머의 인터럽트 함수에서 처리하도록 하였다. Idle Timer는 [2]에서는 Idle인 상태가 시작되면 8254의 카운터를 시작해서 Idle Timer가 소모되기 전까지 Idle인 상태를 유지하지 못하면 인터럽트를 걸도록 구현하였다. 본 연구에서는 소프트웨어적으로 1개의 프레임을 전송하고 난후에 Idle Time만큼 지연시간을 줌으로서 구현하였다. Idle Time은 [2]에서는 33 bit time으로 사용하도록 하고 있지만 Idle Timer의 값은 하드웨어에 종속적인 값이기 때문에 구현한 하드웨어와 소프트웨어의 T_{SM}(Safety Margin Time)값을 계산해서 Idle Time의 값이 조정 되도록 하였다.

타이머의 상태 및 인터럽트 함수는 아래의 timer 구조체를 사용해서 처리한다.

```
typedef struct timer_tag
{
  unsigned long count;
  struct bits_tag
  {
    unsigned char active  : 1; // 0=no active  , 1= active
    unsigned char expired : 1; // 0=no expired , 1= expire
    unsigned char type    : 1; // 0=one_shot   , 1=periodic
    unsigned char incr    : 1; // 0=decrease   , 1=increase
    unsigned char reserved : 4 ;
  }
  bits;
  void (*func)(void);
} timer;
```

3.1.3 프레임의 구조

FDL은 5가지 프레임의 형태를 가지고 있다. FDL은 전송큐에서 전송할 프레임을 꺼내와서 SD1, SD2, SD3, SD4, SC 프레임 전송 버퍼중에서 알맞은 전송버퍼에 넣는다. 재전송 요구가 생기

면 버퍼를 참조하여 재 전송한다. 아래는 가변 데이터 길이 프레임인 SD2의 구조체의 예이다.

```
typedef struct FDL_frame_SD2_tag
  {
    unsigned char SD1;
    unsigned char LE1;
    unsigned char LE2;
    unsigned char SD2;
    Addr DA;
    Addr SA;
    FC_struct FC;
    SD2_DATA_UNIT DATA;
    unsigned char FCS;
    unsigned char ED;
  } FDL_frame_SD2;
```

수신시에는 SD(Start Delimiter)의 종류를 판별하여 SD1, SD2, SD3, SD4, SC 프레임 수신 버퍼중에서 알맞은 수신버퍼에 넣는다.

3.1.4 프로토콜 상태 천이

FDL의 상태는 모두 Offline, Passive_Idle, Listen_Token, Claim_Token, Active_Idle, Use_Token, Await_Data_Response, Check_Access_Time, Check_Token_Pass, Await_Status_Response, Pass_Token등의 11개의 상태로 나뉘지며, 본 연구에서는 이를 11개의 함수로 구현하였다. 아래는 메인함수이다.

```
void main(void)
{
  상태변수 초기화;
  while(1)
  {
    (*FDL_state_func[FDL_state.state])();
    if(이벤트가 발생하였나)
      이벤트 처리 함수();
  }
}
```

상태변수 초기화에서는 FDL_STATE 구조체로 선언되어 있는 변수들을 초기화한다.

```
typedef struct FDL_state_tag
  {
    unsigned char prevstate;
    unsigned char state;
    unsigned int  prevevent;
    unsigned char error;
  } FDL_STATE;
```

본 연구를 통하여 소프트웨어로 구현된 프로토콜에서는 FDL_state.state의 값에 따라서 0~10까지의 상태에 따른 상태함수를 실행한다. 상태함수안에서는 최대한 대기하면서 무한루프를 돌고 있는 구조를 배제함으로서 이벤트 처리에 지연시간이 최소한이 되도록 설계하였다. 여기서 이벤트는 FDL에서 FDL 사용자에게 보내는 프리미티브가 된다. 이벤트 처리함수는 FDL 사용자 프로그램이 된다.

3.1.5 에러의 처리

에러와 예외상황을 올바르게 처리함으로서 안정된 네트워크 시스템을 유지할 수 있다. PROFIBUS에서 생성되는 에러와 예외상황의 종류는 아래와 같다[2].

- ㅇ여러개의 토큰이 생성된 경우
- ㅇ토큰을 잃어버린 경우
- ㅇ토큰이 전달 도중에 사라진 경우
- ㅇ여러 스테이션이 같은 주소를 가지고 있는 경우
- ㅇ스테이션에 잘못된 전송기와 수신기를 사용한 경우
- ㅇ다른 스테이션이 논리적인 링에 삽입되거나 삭제되는 경우

각 상황마다의 에러처리는 [2]에 자세히 설명되어 있으며 본 연구를 통하여 개발된 프로그램에서는 FDL_state.error에 그 상태에서 발생한 에러의 종류를 기입하여 다음 상태에서 그 에러에 알맞은 처리를 하도록 구현하였다. FMA1/2에게 보고하도록 되어 있는 에러의 경우에는 FDL user로부터 전송 요구가 내려오면 이를 거부하고 현재 에러의 상태를 응답에 실어서 FDL user에게 전달한다.

3.2 하드웨어의 구조

하드웨어의 기본 구성 요소는 V25+(16MHz), 128K(byte) 램, 64K(byte) 롬, 8254를 사용한다. RS485 통신을 위해서 SN75176을 사용한다. FDL User 보드와의 데이터통신은 CY7C139 4k(byte) DPRAM(Dual-Port RAM)을 사용하여 구현하였다. 그림 2에는 본 연구를 통하여 개발된 하드웨어의 구조가 나타나 있다. 센서와 엑츄에이터 역할을 할 수 있는 보드는 하나의 보드로 구현하고 제어기를 하나의 보드로 구현하여, 1개의 제어 루프는 2개의 하드웨어 보드로 구성된다. 또한 미디엄상에 전송되는 프레임을 관찰하기 위한 PC 모니터링 보드를 구현하였으며, 구현된 보드에는 FDL 사용자 보드를 따로 붙일 수 있는 형태로 제작하였다.

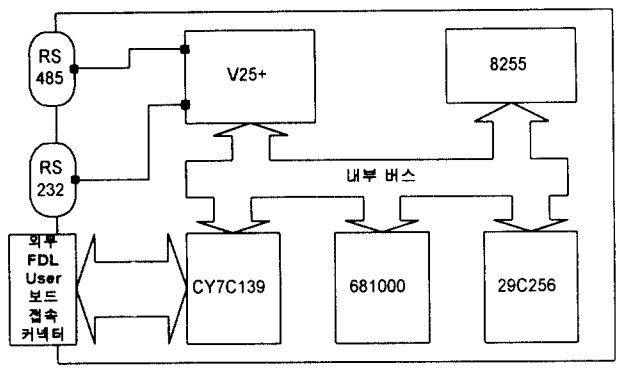

그림 2. 하드웨어의 구조

Fig. Hardware structure

PC 모니터링 보드는 외부 FDL 사용자 보드 접속커넥터에 ISA 버스 접속 보드를 연결하여 사용한다. 센서,엑츄에이터 역할을 수행하는 보드는 외부 보드를 사용하지 않고 내부에서 모든 작업이 처리되도록 하였다.

3.3 대역폭 할당 기법의 적용

PROFIBUS에 대역폭 할당 기법을 적용하기 위하여서는 필드버스 대역폭중에서 주기적인 구간이 시작되었음을 알리는 방법이 필요하다. [5]에서는 일반적인 필드버스를 가정하여 1바이트를 토큰에 붙여서 주기적인 구간이 시작되었음을 알리는 방법을 제시하였으나, PROFIBUS에서는 토큰 프레임의 SA,DA가 1~7번째 비트만 사용하므로 사용하지 않는 8번째 비트를 1로 설정함으로써 주기적인 구간이 시작되었음을 알리는 방법을 사용하였다. 즉, 토큰을 받은 마스터 스테이션이 주기적인 데이터를 전송해야 될 경우에는 토큰 프레임의 SA, DA의 8번째 비트를 1로 설정하여 전송한다. 다른 마스터 스테이션들은 SA, DA의 8번째 비트가 1로된 토큰 프레임을 수신하면 자신이 가지고 있는 주기적인 데이터만 전송한다. SA, DA의 8번째 비트를 1로 만든 마스터 스테이션이 토큰 프레임을 다시 수신하면 SA, DA의 8번째 비트를 0으로 만들고 다시 비주기적인 구간이 시작된다.

본 연구에서는 주기적 실시간 데이터를 위해서 용량이 1인 주기적 실시간 데이터 전송 큐를 따로 구현하였다. 샘플링시에 이 큐에 데이터를 전송하고, 만일 다음 샘플링 데이터가 도착했는데도 큐에 있는 데이터를 전송하지 못했으면, 먼저 데이터를 덮어쓰기한다. 대역폭 할당 기법에서는 TRT(Token Rotation Timer)를 사용하지 않으나 샘플링 시간을 알려주는 타이머와 비실시간 데이터 및 산발적 데이터의 전송 주기를 관리하는 타이머가 필

요하다. 이를 위하여 8254를 사용하여서 2개의 타이머를 추가하였다.

4. 실험 및 고찰

4.1 알고리듬에 의한 필드버스 시스템의 구성

2장에서 기술된 알고리듬을 검증하기 위한 실험용 네트워크 시스템은 다섯 개의 제어루프로 구성된다. 각 제어루프당 주기적 실시간, 산발적 실시간, 비 실시간 데이터 전송큐의 개수는 2, 1, 1개이다. 구현된 FDL 프로토콜의 전송속도는 95238bps이며, 주기적 실시간 데이터의 길이는 135 바이트(헤더:11바이트, 순수 데이터:135바이트)이다. 실제 전송되는 데이터의 길이는 1바이트당 11비트이므로 1606 비트 데이터 길이의 L_t는 34.86msec(전송시간(16.06 msec)+처리시간(18msec))이다. 산발적 실시간 데이터의 길이는 14 바이트(헤더:8 바이트, 순수 데이터: 6 바이트)이다. 여기서 실제 전송되는 데이터의 길이는 154 비트이고 따라서 L_c=19.62msec(전송시간(1.62msec) + 처리시간(18msec))이다. 서버 오버 헤드는 서버 처리 시간과 토큰 전송 시간을 합한 시간으로 본 연구를 통하여 소프트웨어적으로 구현된 프로토콜에서는 19msec가 소요된다. 그러므로 서버 순환 시간은 190msec(10노드×19msec)이다. 다섯 개의 제어 루프는 최대 허용 루프 지연시간은 $[\Phi_1, \Phi_2, \Phi_3, \Phi_4, \Phi_5]$=[2066, 4200, 8500, 17000, 34000]msec로 주어지는 것으로 한다. 산발적 실시간, 비 실시간 데이터의 도착 빈도는 각각 0.001msec^{-1}, 0.001msec^{-1}로 2장에서 제시된 알고리듬의 Step 1에서는 주기적 실시간 데이터의 샘플링 주기를 구하며, 여기서 T_1을 700msec로 선정한다.[1] 그러므로 샘플링주기는 $[T_1, T_2, T_3, T_4, T_5]$ = [700, 1400, 2800, 5600, 11200]msec이다. Step 2에서는 각 노드의 첫 번째 주기적 실시간 데이터가 샘플링되는 순간인 t_j를 구한다. 제시된 알고리듬은 본 예에 적용하면 T_1의 구간내에 윈도우의 개수는 r=4가 되며, 그림 3에서 나타난 것처럼 [t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}]=[0, 0, 0, 0, 700, 700, 2100, 2100, 4900, 4900]msec이 된다. Step 3에서는 비실시간 데이터의 패킷 길이를 구하여 (5)식에 의하여 La=24.908msec이다. 즉, PROFIBUS에서 가변 데이터 전송 프레임을 사용하여 649 bit, 59 바이트(헤더:11 바이트, 데이터:48바이트)의 길이로 잡으면 L_a=24.814msec(전송시간(6.814)+처리시간(18msec))가 된다. 산발적 데이터와 비 실시간 데이터의 도착 주기는 (7)식을 만족해야 하며, (7)식으로부터 λ'''=0.001<0.0016가 만족됨을 알수 있다.

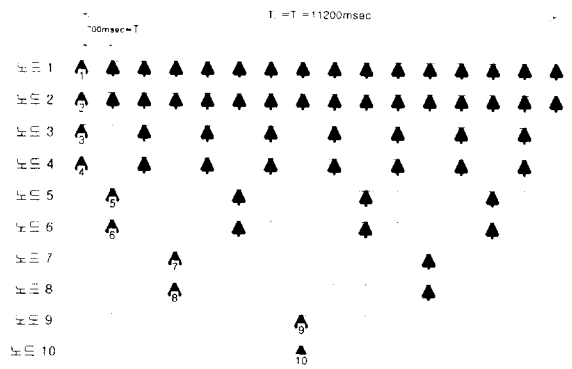

그림 3. 샘플링 주기 할당

fig. 3. Sampling time allocation

4.2 실험

실험은 알고리듬에 의해서 산출된 각 파라미터를 적용한 경우와 4.1장에서 산출한 비실시간 데이터의 패킷 전송 시간이 적절히 설정되지 못한 경우(CASE 1)로 비실시간 데이터의 길이를 255바이트(헤더:11바이트, 순수데이터:244바이트)로 설정한 경우에 대하여 비교하였다.

표1. 알고리듬,CASE 1의 성능 비교

Table 1. Performance comparison for Algorithm, CASE 1

	평균 데이터 전송 지연 시간		최대 데이터 전송 지연 시간		실패 데이터 개수 (실패/생성데이터)	
	알고리듬	CASE 1	알고리듬	CASE 1	알고리듬	CASE 1
산발적 실시간	169.033	161.5	691.21	1003.43	0/1662	0/1665
주기적 실시간	377.332	616.653	665.267	993.311	0/1487	569/1487
비실시간	337.35	225.742	788.475	1003.83	0/1662	0/1666

표1에서 산발적 실시간 데이터의 경우를 보면, 알고리듬의 경우에는 691.21msec로 최대 허용 데이터 지연 시간인 700msec보다 작아서 성능 요구 사항을 만족하고 있다. CASE 1은 1003.43msec로 초과해서 성능 요구 사항을 만족하지 못한다. 또한 전송큐의 용량이 1인 주기적 실시간 데이터의 알고리듬을 적용한 경우에는 생성된 데이터가 모두 전송되었으나, CASE 1의 경우에는 약 40%의 데이터가 전송되지 못하였다.

5. 결론 및 추후 연구 사항

본 논문에서는 제한된 필드버스 대역폭에 산발적 실시간, 주기적 실시간 및 비실시간 데이터 트래픽을 적절히 할당할 수 있는 대역폭 할당 기법을 PROFIBUS에 적용하였다. 실험을 통하여 적용된 대역폭 할당 기법에 의하여 산발적, 주기적 실시간 및 비실시간 데이터들이 데이터 전송 지연 시간의 요구 사항을 만족하고 있음을 알 수 있다. 본 논문에서 구현한 필드버스 시스템을 실제 제어 시스템 실험 모델에 적용하여 제어시스템이 원하는 성능 요구 사항을 만족하는가에 대한 연구를 수행할 계획이다.

참고문헌

[1] S. H. Hong, "Scheduling Algorithm of data sampling times in the intergrated communication and control systems," IEEE Trans. on Control Systems Technology, vol. 3, no. 2, pp.255- 230, June, 1995

[2] DIN 19 245 Profibus Standard Part 1,2

[3] Klaus Bender, "PROFIBUS The Fieldbus for Industrial Automation" Prentice Hall, 1993

[4] Thomas Hadlich, Thorsten Szczepanski, "Joint Hardware/ Software Fieldbus Implementation based on SDL", Fieldcomms, 1996

[5] 홍승호, "대역폭 할당 기법에 의한 필드버스 네트워크의 트래픽 관리 및 제어", 제어.자동화.시스템 공학회지, 제3권, 제1호, pp. 77-88, 1997. 2

[6] 홍승호, 김기암, 김지용, 고성준, "분산제어 및 자동화 시스템과 필드버스", 제어・자동화・시스템공학회지 제2권, 제4호, pp. 19-29, 1996. 7