PROFIBUS에서 대역폭 할당 기법 구현 및 실험적 평가

Implementation and Experimental Evaluation of Bandwidth Allocation Scheme on PROFIBUS

홍 승 호, 김 유 철, 김 지 용
(Seung Ho Hong, Yu Chul Kim, and Ji Yong Kim)

Abstract : Fieldbus is the lowest level industrial network in the communication hierarchy of factory automation and distributed process control systems. Data generated from field devices are largely divided into three categories: time-critical, periodic and time-available data. Because these data share one fieldbus medium, it needs a method that allocates these data to the bandwidth-limited fieldbus medium. This paper introduces an implementation method of bandwidth allocation scheme on PROFIBUS. In order to implement bandwidth allocation scheme on PROFIBUS, the following functions need to be supplemented on the FDL(Fieldbus Datalink Layer) protocol: (i) separation of medium bandwidth into periodic and non-periodic intervals, (ii) synchronization of node timers over a local link. In order to examine the validity of bandwidth allocation scheme on PROFIBUS, this paper develops an experimental model of a network system. The results obtained from the experimental model show that the bandwidth allocation scheme satisfies the performance requirement of time-critical, periodic and time-available data.

Keywords : PROFIBUS, bandwidth allocation, implementation, experimental model, evaluation

I. 서론

필드버스는 공장자동화와 공정제어 시스템에서 자동화 장비들간에 양방향 디지털 직렬 통신기능을 제공하는 산업용 통신망이다. 필드버스에 연결된 각종 필드 장치들로부터 생성되는 데이터들은 크게 산발적 실시간 데이터와 주기적 데이터 및 비실시간 데이터의 세가지 종류로 구별된다. 산발적 실시간 데이터는 각종 이벤트와 경고 신호를 전달하기 위해서 사용되며, 일반적으로 데이터 길이가 짧고 가장 높은 우선순위로 전송된다. 주기적 데이터는 주로 피드백제어 시스템에서 센서 값을 주기적으로 샘플링하거나 주기적인 제어명령을 액튜에이터에 전달하기 위해서 사용되며, 반드시 샘플링 주기의 제한 시간이내에 데이터전송이 완료되어야 하는 실시간성을 요구한다. 비실시간 데이터는 전송 지연시간에 크게 제약을 받지 않으며, 여기에는 프로그램 또는 데이터 파일 등이 포함된다. 필드버스에서는 위의 세가지 종류의 데이터들이 하나의 네트워크 매체를 공유하게 되므로 네트워크의 트래픽이 적절히 관리되지 못하면 산발적 실시간 데이터나 주기적 데이터의 전송 지연시간이 제한 시간을 만족하지 못하는 경우가 발생할 수 있다[1].

접수일자 : 2000. 1. 29., 수정완료 : 2000. 8. 19.
홍승호 : 한양대학교 전자컴퓨터공학부
김유철 : 아이콘트롤스 기술연구소
김지용 : 원클릭테크놀로지스 기술연구소
※ 본 연구는 한국과학재단 목적기초연구(97-01-00-11-01-3) 지원으로 수행되었음.

일반적으로 필드버스의 매체제어방식은 크게 토큰 전달 방식과 중앙 제어 방식으로 분류되는데 토큰 전달 방식은 주로 비주기적 데이터의 전송에 적합한 반면 폴링에 의한 중앙 제어 방식은 주기적 데이터의 전송에 적합한 것으로 알려져 있다. 필드버스의 이러한 제한된 기능을 확장하기 위하여 기존의 프로토콜에 새로운 기능을 추가하는 연구가 진행되고 있다. Hong과 Kim[2]은 CAN 프로토콜에서 실시간 데이터의 전송 요구사항을 보장하기 위한 대역폭 할당 기법을 제시하였고, Cavalieri et. Al[3]는 IEC/ISA 필드버스에서 우선순위 기능을 사용하는 경우에 각각의 우선순위 데이터들에 대한 대역폭 할당 기법을 제시하였다. 또한, Kobayashi et al[4]은 802.4 토큰 버스와 802.5 토큰 링에서 실시간성과 빠른 에러 처리를 위한 기능을 추가한 IINET(Integrated Instrument Network) 이라는 새로운 프로토콜을 제시하고, 이를 LSI로 구현하였다. IINET은 토큰 링 프로토콜에서 토큰에 우선순위를 부여하여 우선순위가 높은 토큰을 가지고 있는 노드가 먼저 데이터를 전송하도록 함으로써 실시간 데이터의 전송지연 제한시간을 만족하도록 하였다.

본 논문에서는 토큰 전달 방식의 PROFIBUS[5][6]에 대역폭 할당 기법을 적용하는 방법을 제시하고, 이를 실제로 구현함으로써, 대역폭 할당 기법이 PROFIBUS에서 주기적 및 산발적 실시간 데이터의 전송지연 요구사항을 만족시키는 동시에 네트워크의 대역폭을 충분히 활용할 수 있음을 실험 모델을 통하여 제시한다. 본 논문의 II장에서는 PROFIBUS의 데이터링크

계층인 FDL (Fieldbus Datalink Layer) 프로토콜 구조에 대하여 간략히 기술한다. III장에서는 대역폭 할당 기법에 대해서 기술하고, IV장에서는 본 연구를 통하여 구현된 FDL 프로토콜의 하드웨어와 소프트 웨어에 대하여 기술한다. V장에서는 대역폭 할당기 법을 FDL계층에 적용하기 위한 방법을 제시하고, VI 장에서는 실험모델을 이용한 실험결과에 대해서 고찰 한다. 마지막으로 VII장에서는 본 논문의 결론 및 추후 연구 과제에 대하여 기술하도록 한다.

II. PROFIBUS FDL 프로토콜

일반적으로 필드버스는 OSI(Open Systems Interconnection) 참조 모델에서 제시하는 7 계층 통신망 구조 가운데 물리 계층, 데이터링크 계층, 응용 계층의 3계층으로 이루어진다. 이러한 구조는 데이터 전송의 실시간성을 보장하기 위해서 꼭 필요한 기능만 선별한 것이다. PROFIBUS의 데이터링크 계층인 FDL에서 지원하는 데이터 전송 서비스에는 원격 스테이션에게 데이터를 전송해주기 위한 SDA(Send Data with Acknowledge) 서비스, 원격 스테이션에게 데이터를 전송함과 동시에 데이터를 전송해 줄 것을 요구하는 SRD(Send and Request Data with Reply) 서비스, 모든 노드에게 데이터를 동시에 전송하는 브로드 캐스 트 전송이나 특정한 그룹에 속한 스테이션들 에게만 데이터를 보내는 멀티캐스트 전송에 사용되는 SDN (Send Data with No acknowledge) 서비스가 있다. 또한 주기적 데이터 전송 서비스를 위하여 SRD 서비 스를 이용한 CSRD(Cyclic Send and Request Data with Reply) 서비스를 제공한다.

FDL의 매체 접근 제어(Medium Access Control) 프로토콜은 토큰 전달 방식과 중앙 제어 방식을 지원한다. 토큰 전달 방식은 마스터 노드의 데이터 전송에 사용되며, 매체 접근 권한을 부여하는 토큰이 네트워크 내의 마스터 노드들을 차례로 방문한다. 토큰을 수신한 마스터 노드는 토큰을 소유하는 동안 전송큐에 대기하고 있는 데이터들을 전송한다. PROFIBUS는 high와 low의 두 가지 데이터 우선순위를 지원하며, 노드에서 생성되는 데이터를 실시간 전송 특성에 따라 high 또는 low 우선순위로 전송한다. 중앙 제어 방식은 슬레이브 노드의 데이터 전송에 사용되며, 마스터 노드가 토큰을 소유하는 동안 네트 워크 내의 슬레이브 노드들에게 데이터를 전송하도록 매체 접근 권한을 분배하는 방식이다.

PROFIBUS에서는 FDL 계층의 성능을 조절하기 위하여 타이머와 시간변수를 사용한다. 여기에는 Idle, Slot, Time-out, Syn Interval, Token Rotation, GAP Update 등 6개의 타이머가 있고, Syn Time, Syn Interval, Station Delay Time, Ready Time, Safety Margin Time, Idle Time, Transmission Delay Time, Slot Time, Time-out Time, Gap Update Time 등 10개의 시간 변수가 있다. 이 시간 변수들 중에서 Syn Time, Station Delay Time, Safety Margin Time 등은 실제 물리 계층과 데이터 연결 계층이 어떠한 하드웨어와 소프트웨어로 구현되었는가에 따라 다른 값을 가지게 된다.

PROFIBUS의 FDL에는 SD1, SD2, SD3, SD4, SC의 다섯 가지의 프레임이 있다. SD1은 프레임 크기가 6 바이트로 고정되어 있고 데이터가 없는 형태이다. SD2는 프레임 크기가 9~255바이트까지 가변할 수 있고 데이터가 0~246바이트까지 포함되어지는 형태 이다. SD3는 프레임 크기가 14바이트로 고정되어 있고 데이터가 8바이트 포함되어지는 형태이다. SD4는 토큰 프레임으로 3바이트의 크기를 가지고 있다. SC는 요구 프레임에 대해서 올바르게 수신했음을 알리는 응답 프레임의 역할만을 수행한다.

PROFIBUS FDL 계층은 Offline, Passive_Idle, Listen_Token, Active_Idle, Claim_Token, Use_Token, Await_Data_Response, Check_Access_Time, Pass_Token, Check_Token_Pass, Await_Status_Response 등의 11개의 상태를 가지고 있다. 여기서 마스터 노드는 Passive_Idle 상태를 제외하고 10개의 상태가 필요하며, 슬레이브 노드는 Offline, Passive_Idle의 2개의 상태만을 갖는다. 규격서에는 각 상태에서 다른 상태로 천이할 수 있는 조건들이 명시되어 있다. PROFIBUS FDL 프로토콜의 동작은 규격서[5]에 자세히 명시되어 있으므로 본 논문에서는 생략한다.

III 대역폭 할당 기법

필드버스의 매체를 통하여 전송되는 데이터는 크게 산발적 실시간, 주기적 및 비실시간 데이터로 구성된다. PROFIBUS의 FDL계층에서 지원하는 토큰 전달 방식은 우선순위 기능을 통하여 우선순위가 높은 데이터의 전송지연 시간을 단축시킬 수는 있으나 데이터 전송의 실시간성을 100% 보장하지는 못한다. 특히, 데이터 생성 주기 이내에 데이터 전송이 완료되어야 하는 주기적 데이터의 경우 전송지연시간의 요구조건이 보장되지 못하는 단점이 있다. 또한, PROFIBUS-DP에서 사용되는 중앙 제어 방식의 경우 슬레이브 노드에서 발생되는 주기적 데이터의 발생 주기를 고려하여 슬레이브 노드의 폴링 시간을 미리 스케듈링함으로써 주기적 데이터의 전송지연시간 요구조건을 만족시키는 방법을 생각할 수 있으나, 이 경우 마스터 노드는 하나만이 사용되어야 하며 매체의 대역폭이 충분히 활용되지 못하는 단점이 있다.

본 연구에서는 PROFIBUS FDL의 토큰 전달 방식에 적용할 수 있는 대역폭 할당 기법을 제시한다. 본 연구를 통하여 제시하는 대역폭 할당 기법은 주기적 및 산발적 실시간 데이터의 전송지연시간 요구조건을 만족시키는 동시에 매체 대역폭의 이용도를 최대화하여 전송지연시간에 구속 받지 않는 비실시간 데이터의 전송효율을 극대화한다. 대역폭 할당 기법은 기본적으로 각 전송큐가 한번에 한 개씩 의

데이터만을 전송하는 단일서비스 방식으로 동작 된다. 대역폭 할당 기법의 알고리듬은 참고문헌[1]에 자세히 기술되어 있으며, 본 논문에서 개념만을 요약 하여 기술한다.

NOTATION :

N : 전체 노드 개수
N_p : 주기적 데이터를 생성하는 노드 개수
N_c : 산발적 실시간 데이터를 생성하는 노드 개수
N_a : 비실시간 데이터를 생성하는 노드 개수
L_p : 주기적 실시간 데이터의 전송 시간
L_c : 산발적 실시간 데이터의 전송 시간
L_a^i : 노드 i 에서 비실시간 데이터의 전송 시간
L_a : 비실시간 데이터 패킷의 전송 시간
σ : 토큰 오버 헤드
R : $(=N\sigma)$ 토큰 순환 시간
$[\phi_i,\ i=1$ 에서 $N_p]$: 주기적 데이터의 최대 허용 지연 시간
ϕ_c : 산발적 실시간 데이터의 최대 허용 지연 시간
$[\lambda_c^i,\ i=1$ 에서 $N_c]$: 산발적 실시간 데이터 패킷의 평균 도착 빈도
$[\Lambda_a^i,\ i=1$ 에서 $N_a]$: 비실시간 데이터의 평균 도착 빈도
$[\lambda_a^i,\ i=1$ 에서 $N_a]$: 비실시간 데이터 패킷의 평균 도착 빈도
$[T_i,\ i=1$ 에서 $N_p]$: 노드 i 에서 주기적 데이터 생성 주기

대역폭 할당 기법에서는 산발적 실시간, 주기적 및 비실시간 데이터들을 처리하기 위해서 필드버스에서 이용할 수 있는 대역폭을 주기적 구간과 비주기적 구간으로 나눈다. 주기적 구간에서는 주기적 데이터들이 전송되며, 이들에 대한 대역폭 할당은 윈도우 스케듈링 알고리듬[7]을 적용한다. 비주기적 구간에서는 비실시간 데이터들이 전송된다. 산발적 실시간 데이터는 가장 우선적으로 전송되어야 하며, 따라서 주기적 및 비주기적 구간에 상관없이 데이터 전송 기회가 부여되는 즉시 데이터를 전송한다.

그림 1에는 대역폭 할당 기법을 통하여 필드버스의 대역폭을 적절히 분배하는 예가 나타나 있다.

주기적 구간에서 사용되는 윈도우 스케듈링 알고리듬의 기본 개념은 N_p 개의 주기적 데이터를 생성하는 노드가 주기적 구간에서 제공하는 $\gamma(\gamma \le N_p)$ 개의 윈도우를 서로 동적으로 공유하도록 하는 것이다. 즉, 주기적 데이터는 윈도우를 통하여 전송되며, 주기적 구간동안 생성되는 데이터의 개수가 γ 개의 윈도우 개수를 초과하지 않도록 각 노드에서 주기적 데이터의 생성 주기 T_i 를 스케듈링하는 것이다.

N_p 개의 주기적 데이터 생성 노드에서 데이터 생성 주기를 원소로 하는 벡터 T 와 최대 허용 지연 시간 벡터 Φ 는 다음과 같이 정의된다.

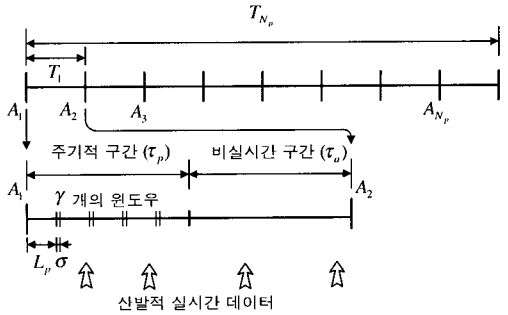


그림 1. 필드버스 대역폭 할당의 예.

Fig 1. An example of the fieldbus bandwidth allo Cation.

$$T = [T_1,\ T_2,\ T_3,\ ...,\ T_{N_p}] \qquad (1)$$
$$\Phi = [\phi_1,\ \phi_2,\ \phi_3,\ ...,\ \phi_{N_p}]$$

여기서 T 와 Φ 는 오름차순으로 정렬한다(즉, $T_i \le T_{i+1},\ \phi_i \le \phi_{i+1}$). T_1 은 N_p 개의 노드 중에서 최소 생성 주기이고 T_{N_p} 은 최대 생성 주기이다. T_1 은 주기적 데이터의 최대 허용 지연 시간 ϕ_i 중에서 최소값인 ϕ_1 로 결정된다. 나머지 노드들의 생성 주기 T_i 는 최대 허용 지연 시간 ϕ_i 을 초과하지 않는 범위 내에서 (2)의 관계를 만족시키는 최대값으로 지정되며 이 경우에 모든 생성 주기는 서로 배수의 관계를 갖는다.

$$\mathrm{Rem}\left[\frac{T_j}{T_i}\right] = 0,\ \forall i, j(j \ge i) \qquad (2)$$

따라서 네트워크 대역폭은 T_{N_p} 주기로 반복된다. 필드버스의 대역폭은 T_{N_p} 주기내에서 T_1 길이로 분할된다. T_1 의 주기내에서 주기적 구간 τ_p 동안 γ 개 이하의 주기적 데이터가 생성되도록 하기 위하여서는 N_p 개의 노드에서 생성 주기가 (3)의 조건을 만족하도록 설정되어야 한다.

$$\gamma = \lceil \alpha_K \rceil,\ \alpha_K = \sum_{i=1}^{N_p} \frac{T_1}{T_i} \qquad (3)$$

여기서 α_K 는 T_1 동안 생성되는 주기적 데이터 개수의 평균값이다. 주기적 데이터를 생성하는 노드에서 첫번째 주기적 데이터 생성 순간인 t_j 는 다음과 같이 설정된다.

$$t_j = \inf\left[A_l \ge A_{l-1} : u^i(A_l) \le \gamma\right],$$
$$i=1 \text{ 에서 } N_p,\ l=1 \text{에서 } k_{N_p} \qquad (4)$$

여기서 A_l 은 T_{N_p} 구간 내에서 $l-$번째 T_1 슬롯이 시작되는 시점이며, $u^i(A_l)$ 은 $i-$ 번째 노드에서 생성된 주기적 데이터를 포함하여 A_l 의 순간에 생성된 주기적 데이터의 개수이다. (4)의 조건은 T_1 구간 내에서 생성되는 주기적 데이터의 개수가 γ 를 초과하지 않도록 제한한다.

주기적 데이터는 T_1 의 구간 내에서 전송이 완료되어야 한다. 따라서 주기적 데이터 트래픽의 안정화 조건은 다음과 같다.

$$\gamma L_p + N_c L_c + R \le T_1 \tag{5}$$

여기서 $N_c L_c$ 는 주기적 구간동안 전송되는 산발적 실시간 데이터에 최대로 할당될 수 있는 대역폭이다.

그림 1의 T_1 대역폭 구간에서 주기적 구간 τ_p 를 제외한 나머지 대역폭은 비실시간 데이터의 전송을 위하여 할당된다. 비실시간 데이터는 전송 지연 시간 에 크게 구애를 받지 않는다. 따라서 어느 노드 에서든지 주기적 실시간 데이터가 생성되면 비실시간 구간 τ_a 는 중지되고 네트워크 대역폭은 다시 주기적 실시간 데이터의 전송을 위한 주기적 구간 τ_p 로 전환된다. PROFIBUS에 대역폭 할당 기법을 적용하기 위하여서는 대역폭을 주기적 구간과 비주기적 구간으로 구분할 수 있는 기능이 필요하며, 따라서 기존의 프로토콜을 변형시킬 필요가 있다. PROFIBUS에서 대역폭을 주기적 구간과 비실시간 구간으로 구분하는 기능은 V장에서 기술된다.

주기적 구간 τ_p 동안 N_p 개의 주기적 실시간 데이터 전송큐 가운데 γ 개 이하의 데이터만이 생성되도록 스케듈링 되어 있으므로 토큰이 네트워크 내의 모든 노드들을 순환하는 동안에 이러한 주기적 데이터들은 전송이 완료된다. 비실시간 데이터는 전송단에서 여러 개의 패킷으로 분할되어 전송되며, 수신단에서는 이를 재조립한다. 비실시간 데이터의 경우에는 주기적 및 산발적 실시간 데이터의 전송지연시간 요구조건을 만족시키기 위하여 필요에 따라 패킷전송 시간을 다음과 같이 제한한다.

$$L_a \le \frac{T_1 - (\gamma L_p + N_c L_c + R)}{N - \gamma + 1} \tag{6}$$

네트워크 시스템이 안정화하기 위하여서는 (5)에서 제시한 주기적 데이터 트래픽 안정화 조건과 더불어 산발적 실시간 데이터와 비실시간 데이터의 트래픽 부하가 네트워크의 용량을 초과하지 않아야 한다. Ibe와 Cheng이 제시한 station backlog 기법[8]에 의하여 산발적 실시간 데이터와 비실시간 데이터 트래픽의 안정화 조건을 구하면 다음과 같다.

$$\lambda_c^i \le \frac{1 - L_c \sum_{j=1}^{N_c} \min[\lambda_c^i, \lambda_c^j] - L_a \sum_{j=1}^{N_a} \min[\lambda_c^i, \lambda_a^j] - [(\alpha_K L_p + R)/T_1]}{R}$$

$$\lambda_a^i \le \frac{1 - L_a \sum_{j=1}^{N_a} \min[\lambda_a^i, \lambda_a^j] - L_c \sum_{j=1}^{N_c} \min[\lambda_a^i, \lambda_c^j] - [(\alpha_K L_p + R)/T_1]}{R} \tag{7}$$

대역폭 할당 알고리듬은 다음과 같다. Step 1에서 기호 $y = \langle x \rangle$ 는 y 가 x 를 초과하지 않는 2의 멱(즉, 2^{n_i}, $n_i \in \{0,1,2,\ldots\}$)임을 나타낸다.

GIVEN :

$N,\ N_p,\ N_c,\ N_a,\ L_p,\ L_c,\ L_a^i, \sigma,\ R,\ \phi_c,$

$[\phi_i,\ i=1 \text{to } N_p],\ [\lambda_c^i, i=1 \text{ to } N_c],\ [\lambda_a^i, i=1 \text{ to } N_a]$

STEP 1 : 주기적 실시간 데이터의 생성 주기 T_i 를 구한다.

$\phi_1 = \min[\phi_i,\ i=1$ 에서 $N_p]$

$T_1 = \phi_1$

$k_i = \left\langle \frac{\phi_i}{T_1} \right\rangle, \forall\, i=1$ 에서 N_p

$\alpha_K = \sum_{i=1}^{N_p} \frac{1}{k_i}$

$\gamma = \lceil \alpha_K \rceil$

If $\gamma L_p + N_c L_c + R > T_1$, then (주기적 실시간 데이터 트래픽의 과부하 $\Rightarrow$ N_p 또는 N_c 감소 : goto Step 1)

Else $T_i = k_i T_1, \forall i = 2$ to N_p

STEP 2 : 각 노드에서 첫 번째 주기적 실시간 데이터 생성 순간 t_j 를 구한다.

$t_1 = A_1 = 0$

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

$$t_j = \inf[A_l \ge A_{l-1}; \mu^j(A_l) \le \gamma]$$

STEP 3 : 비실시간 데이터의 데이터 패킷 길이 L_a 를 구한다.

$$L_a \le \begin{cases} \dfrac{\phi_c - (\gamma L_p + N_c L_c + R)}{N - \gamma + 1}, & \phi_c < T_1 \\ \dfrac{T_1 - (\gamma L_p + N_c L_c + R)}{N - \gamma + 1}, & \text{otherwise} \end{cases}$$

$$\lambda_a^i = \left\lceil \frac{L_a^i}{L_a} \right\rceil \Lambda_a^i$$

$$\lambda_c^i \ge \frac{1 - L_c \sum_{j=1}^{N_c} \min[\lambda_c^i, \lambda_c^j] - L_a \sum_{j=1}^{N_a} \min[\lambda_c^i, \lambda_a^j] - [(\alpha_K L_p + R)/T_1]}{R} \text{ or}$$

$$\text{If } \lambda_a^i \ge \frac{1 - L_a \sum_{j=1}^{N_a} \min[\lambda_a^i, \lambda_a^j] - L_c \sum_{j=1}^{N_c} \min[\lambda_a^i, \lambda_c^j] - [(\alpha_K L_p + R)/T_1]}{R}$$

Then (산발적 실시간 또는 비실시간 트래픽의 과부하 $\Rightarrow$ N_c 또는 N_a 감소: goto Step 3)

End

Ⅳ. PROFIBUS FDL 계층 구현

PROFIBUS의 FDL 계층을 구현하는 방법에는 하드웨어와 소프트웨어를 혼용하는 방법과 소프트웨어적으로 구현하는 방법이 있다. PROFIBUS 표준화 단체에서 초기에는 PROFIBUS의 FDL계층을 소프트웨어로 구현했다. 최근에는 데이터 전송 속도를 증가시키기 위하여 ASIC을 이용하여 프로토콜의 일부를 하드웨어화 시키는 것이 일반적인 추세이다. 본 논문에서는 기존의 PROFIBUS FDL 프로토콜에 대역폭 할당 기법을 적용한 수정된 프로토콜을 필요로 하므로,

상용화된 ASIC칩 및 소프트웨어를 이용하지 않고 FDL 프로토콜을 직접 소프트웨어로 구현하였다.

1. FDL 계층 보드 구현

본 논문에서 제작한 PROFIBUS FDL 보드는 PC나 외부 호스트 보드에 연결하여 FDL기능만 수행할 수 있고, 간단한 응용 프로그램을 포함하여 독립적으로도 동작할 수 있다. 그림 2는 PROFIBUS FDL 보드의 전체 구조를 나타낸다. 또한 그림 3에는 본 연구를 통하여 구현된 보드의 사진이 나와 있다. PROFIBUS FDL 보드의 CPU로는 모토롤라사의 MC68360[9]을 사용하였다. MC68360은 내부에 CPU32+ 코어를 가지고 있으며 별도로 통신을 위한 Communication Processor를 가지고 있다. FDL 보드는 시동프로그램을 저장하기 위한 1M비트 플래시 롬(AT29C010A)과 프로그램 실행 시 필요한 1M비트 S램(KM681002C)을 4개 사용하고, PC와의 인터페이스를 위한 듀얼포트램(CY7C139)을 장착하였다. 그리고 RS-485칩(MAX1487)을 사용하여 네트워크에 접속하며 모니터링을 위해서 RS-232칩(MAX233CPP)을 추가로 장착하였다.

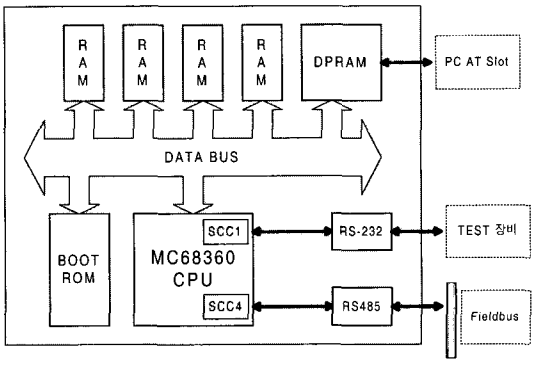


그림 2. PROFIBUS FDL 보드의 구조.

Fig 2. Structure of PROFIBUS FDL board.

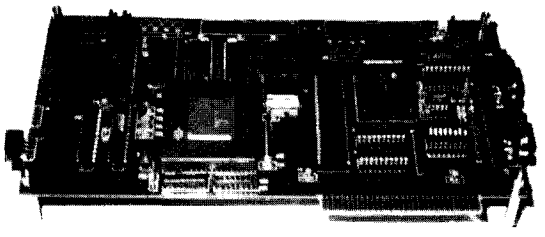


그림 3. PROFIBUS FDL 보드 사진.

Fig 3. Picture of PROFIBUS FDL Board.

MC68360에서는 4개의 SCC(Serial Communication Controller)를 통해 직렬 통신을 제공한다. SCC에는 CRC 종류, sync 비트 길이, 인코딩 종류, 프리앰블 종류 등의 물리계층에 필요한 파라미터들이 설정된다. 그리고 범용으로 사용되는 프로토콜의 경우 프로토콜 타입을 설정해주도록 되어 있다. RS-485와 RS-232의 경우에는 UART를 지정해주면 된다. PROFIBUS 표준안에서는 9600, 19200, 93750, 187500, 500000 bps를 지원하도록 하고 있다. MC68360에서는 BRG(BaudRate Generator) Configuration Register에서 보드레이트를 설정한다. MC68360은 4개의 16비트 General Purpose Timer와 16개의 RISC Timer를 제공한다. PROFIBUS의 FDL 계층 구현을 위하여 실제로 필요로 하는 타이머는 Token Rotation, Slot, Time-out, Sync Interval 타이머들이며 General Purpose Timer 3번과 4번을 이용하여 이들 타이머를 구현하였다. PROFIBUS 물리 계층은 주로 트위스트페어선을 이용하는RS-485를 사용한다. 그림 4는 본 논문에서 사용한 RS-485 의 연결도이다. PROFIBUS에서는 양 끝단의 터미널 저항을 150Ω으로 규정하고 있다. 풀업 저항과 풀다운 저항은 390Ω으로 규정하고 있다.

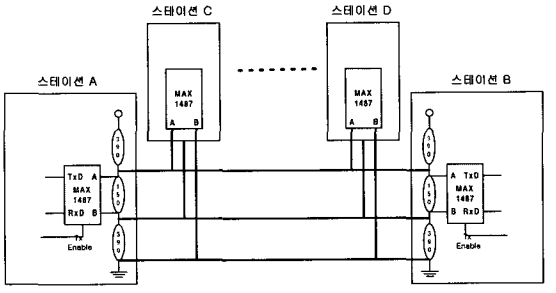


그림 4. RS485 신호선의 연결.

Fig 4. A link of RS485 signal lines.

2. FDL 계층 소프트웨어 구현

FDL 계층은 FDL의 사용자인 LLI(Lower Layer Interface) 계층에게 SRD, SDA, SDN, CSRD의 서비스를 제공한다. 각각의 서비스에 해당하는 알고리듬에 따라 프레임을 생성, 전송하고 매체로부터 수신되는 프레임을 분석하여 자신이 목적지인 프레임만을 선별 및 처리하여 LLI로 올려 보낸다. 그리고 노드들의 주소록(LAS) 작성 관리, 노드 추가를 위한 GAP Update, 전송권한(토큰) 관리 등의 기능도 수행한다. 본 논문에서 구현한 FDL 계층의 소프트웨어는 스타트업 코드부분과 인터럽트 처리 함수의 일부를 어셈블리어로 작성하고, 나머지는 C언어를 사용하여 작성하였다. FDL 프로그램은 표 1에서 처럼 LLI 인터페이스 함수, 인터럽트 함수, 프레임처리 함수, 상태천이 함수 등 4부분으로 나누어 진다.

LLI 인터페이스 함수는 FDL에서 LLI로 데이터를 읽거나 보내주는 기능을 수행한다. Receiving_LLI_Req_Res() 함수는 LLI interface queue에서 request 또는 response data를 읽어오는 함수이며, Receiving_LLI_Ind_Con() 함수는 LLI interface queue로 indication 또는 confirm data를 보내주는 함수이다. 인터럽트 함수는 각 인터럽트에 대한 처리 함수들이다. 이 함수들은 빠른 처리를 위하여 어셈블리어로 작성한

표 1. 함수의 분류.
Table 1. Classification of functions.

LLI 인터페이스 함수	인터럽트 함수	프레임 처리 함수	상태 천이 함수
Receiving_LLI_Req_Res() Sending_LLI_Ind_Con()	SCC4INT() Timer1() TimerPC0() Timer3() Timer4()	GetSD1Frame() GetSD2Frame() GetSD3Frame() GetSD4Frame() GetSCFrame() SendSD1Frame() SendSD2Frame() SendSD3Frame() SendSD4Frame() SendSCFrame() Send_HPS_frame() Send_LPS_frame() CheckSD() Tx_485() getch0()	main() RegisterLAS() FindNs() get_trt() Offline() Listen_Token() Active_Idle() Claim_Token() Use_Token() Await_Data_Response() Check_Access_Time() Pass_Token() Gap_Update() Check_Token_Pass() Await_Status_Response() Passive_Idle() App()

다. SCC4INT() 함수는 RS485 수신 인터럽트 처리 함수로서 인터럽트가 걸리면 수신된 문자를 큐에 집어 넣는다. Timer1() 함수는 MC68360 내부 타이머 1의 인터럽트 처리 함수로서 TRT 값을 계산하기 위해서 사용된다. TIMERPC0() 함수는 타이머 2에서 가장 높은 우선순위를 가지는1msec 타이머이며 가상의 메시지 발생시간을 제어한다. Timer3() 함수는 Slot 타이머, Sync Interval 타이머의 역할을 한다. Timer4() 함수는 Time-out 타이머의 역할을 한다.

프레임 처리 함수는 전송 큐의 메시지를 PROFIBUS의 프레임 구조로 만들어서 RS485를 통해서 전송하고, 수신된 프레임을 메시지로 만들어서 수신 큐에 복사하는 역할을 한다. Send_LPS_frame(), Send_HPS_frame() 함수는 각각 low 전송 큐, high 전송큐에서 frame 구조체를 읽어오고, 데이터의 종류에 따라 SendSD1Frame(), SendSD2Frame(), Send SD3Frame() 함수 중에서 선택하여 메시지를 전송한다. SendSD4Frame() 함수는 토큰을 전송할 때에 이용된다. SendSCFrame() 함수는 SC 프레임을 전송할 때에 이용된다. CheckSD() 함수는 수신된 문자들을 검색하여 SD(Start Delimiter)를 찾아낸다. SD에 따른 프레임의 수신은 GetSD1Frame(), GetSD2 Frame(), Get SD3Frame(), GetSD4Frame(), GetSC Frame() 함수들이 이용된다.

상태 천이 함수들은 FDL의 상태에 따른 각 상태 함수들과 상태 함수들을 호출하는 함수들이다. main() 함수에서는 FDL_state 구조체의 현재 상태에 따라 알맞은 상태함수들을 호출하고, 에러처리를 한다. RegisterLAS() 함수는 LAS에 마스터 스테이션을 등록하기 위한 함수이다. FindNS()는 현재의 NS(Next Station)가 응답이 없을 경우에 다음 NS를 LAS중에서 찾기 위한 함수이다. get_trt() 함수는 Use_Token() 상태 함수에서 Real Token Rotation Time 과 Token Hold Time을 계산하기 위한 함수이다. Offline(), Active_Idle(), ... , Passive_Idle() 등의 상태 함수들은 FDL의 상태에 따른 처리 기능을 수행한다. App() 함수는 대역폭 할당 기법의 타당성을 검증하기 위하여 추가로 구현한 FDL 계층의 사용자함수로서 주기적, 산발적 실시간 및 비실시간 데이터를 FDL_SDN(), FDL_SDA(), FDL_SRD() 함수들을 이용하여 전송한다. 그림 5는 데이터의 흐름에 따른 함수들의 호출 경로이며 FDL 소프트웨어의 전체 구조를 나타내고 있다.

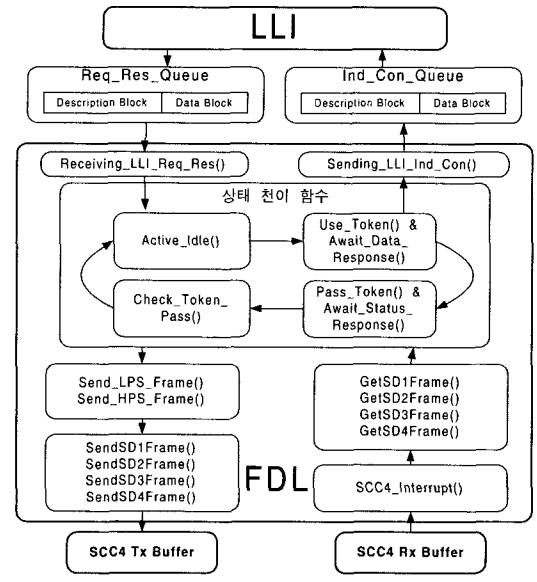


그림 5. FDL 프로그램의 전체 블록도.
Fig 5. Block diagram of FDL program.

V. 대역폭 할당 기법의 적용

본 장에서는 기존의 PROFIBUS FDL 프로토콜에 대역폭 할당 기법을 적용하는 방법을 제시한다. PROFIBUS에 대역폭 할당 기법을 적용하기 위해서는 i) 주기적 및 비실시간 대역폭 구분 기능과 ii) 노드들간에 시간 동기화 기능이 추가로 구현되어야 한다. 이를 위하여 제 IV장에서 기술된 FDL 계층의 소프트웨어를 변형할 필요가 있다. 표 2에는 표1에서 제시된 함수에서 수정 또는 추가 되어야 할 함수들이 나타나 있다.

1. 주기적 및 비실시간 대역폭 구분 기능

일반적인 필드버스를 가정한 대역폭 할당 기법[1]에서는 1바이트를 서버에 추가하여 주기적인 구간이 시작되었음을 알리는 방법을 제시하였다.

그러나 PROFIBUS에서는 토큰 프레임의 SA(Source Address), DA(Destination Address)가 0~6번 비트만 사용하므로, 사용하지 않는 7번 비트를 1로 이용하여 주기적 구간이 시작되었음을 알리는 방법을 사용하였다. 즉, 토큰을 받은 마스터 노드가 주기적인 데이터를 전송해야 될 경우에는 토큰 프레임의 SA, DA의 7DA의 7번 비트가 1로 지정된 토큰 프레임을 수신하면 주기적 구간이 시작되었음을 인식하고 자신이 가지고 있는 주기적 데이터만 전송한다. SA, DA의 7번 비트를 1로 설정한 노드가 토큰 프레임을번 비트를 1로 설정하여 전송한다. 다른 노드들은 SA, 다시수신하면 SA, DA의 7번 비트를 0으로 설정하고, 이 때부터 다

표 2. 대역폭 할당의 위해 수정 또는 추가되는 함수.
Table 2. Modified or added functions for bandwidth allocation.

User 함수	인터럽트 함수	프레임 처리 함수	상태 함수
없음	없음	GetSD4Frame()** SendSD4Frame()** GetSD1Frame()** GetSD2Frame()** Send_PERIOD()*	Use_Token()**

*: 추가, **: 수정

시 비주기적인 구간이 시작된다. 각 노드는 자신이 주기적 구간을 설정했음을 인식하기 위한 StartPeriod 상태변수와 현 구간이 주기적 구간임을 인식하기 위한 IsPeriod 상태변수를 갖는다.

토큰을 수신한 노드가 주기적 데이터를 가지고 있고, IsPeriod 변수가 0으로 설정되었으면 해당 노드는 StartPeriod 변수와 IsPeriod 변수를 1로 설정하고 주기적 데이터만을 전송한다. 각 노드는 토큰(SD4) 프레임을 전송하기 전에 현재 구간이 주기적 구간임을 나타내는 IsPeriodic 변수가 1인지 검사하여 1이면 SA, DA의 7번 비트를 1로 설정하고, IsPeriodic변수가 0이면 SA, DA의 7번 비트를 0으로 설정한 후 SD4 프레임을 전송한다. 이러한 기능들이 수정된 SendSD4 Frame() 함수에 구현되었다.

주기적 구간을 끝내고 비주기적 구간을 시작하거나 계속 주기적 구간을 유지하기 위해서 각 노드는 SD4 프레임을 수신한 후 SA, DA의 7번 비트가 1인지 검사하여 1이면 다시 자신이 시작한 주기적 구간임을 표시하는 StartPeriod변수가 1인지 검사한다. 만일 StartPeriod변수가 1이면 주기적 구간임을 표시하는 IsPeriodic 변수와 StartPeriod변수를 0으로 설정하고 주기적 구간을 종료한다. 그러나 StartPeriod 변수가 0이면 IsPeriodic변수를 1로 설정하여 주기적 구간이 유지되도록 한다. 이러한 기능이 수정된 GetSD4Frame() 함수에 구현되었다.

주기적 구간동안 주기적 데이터만이 전송되도록 하기 위하여 Use_Token() 함수를 수정하였다. 수정된 Use_Token 함수에서는 high 우선순위를 갖는 산발적 실시간 메시지를 처리한 후에 현재 구간이 주기적 구간인지를 나타내는 IsPeriodic변수가 1인지 검사한다. IsPeriodic 변수값이 1이면 주기적 데이터를 전송하기 위하여 추가로 구현된 Send_PERIOD 함수를 호출한 후에 low 우선순위로 설정되는 비실시간 데이터들은 전송하지 않고 바로 Pass_Token 상태로 천이한다.

2. 노드들 간에 시간 동기화 기능

대역폭 할당 기법에서는 네트워크내의 주기적 데이터를 생성하는 노드들간에 데이터 생성 시간을 동기화 시킬 필요가 있다. 데이터 생성 시간을 동기화 시키기 위해서는 먼저 각 스테이션의 노드 시간을 동기화 시켜야 한다. 노드들간에 시간 동기화는 MC68360 CPU에서 제공하는 1msec의 분해능을 가진 32비트 타이머인 Timer 2를 이용한다. 노드들간에 시간 동기화를 위하여서는 네트워크내에 기준시간을 제공하는 시간-마스터 노드가 하나 존재한다. 시간-마스터 노드는 SD1 및 SD2 프레임을 이용하여 시간 정보를 전달한다. 그림 6에는 시간 동기화 알고리듬의 타이밍도가 나타나 있으며, 알고리듬의 동작은 다음과 같다.

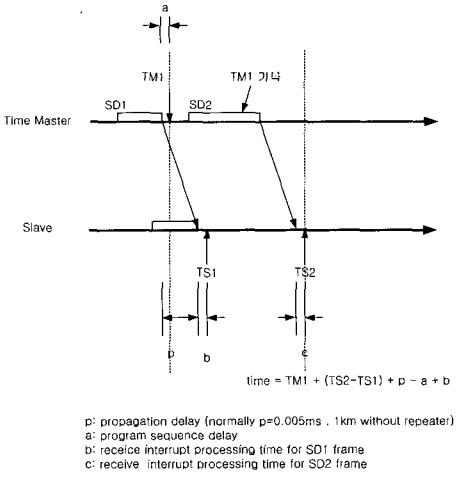


그림 6. 동기화 타이밍도.
Fig. 6. Timing diagram of synchronization function.

• 시간-마스터 노드는 주기적으로 시간 동기화를 위한 SD1, SD2 프레임을 연속해서 보낸다.

• 시간-마스터 노드는 SD1 프레임의 마지막 바이트를 보낸 직후 그때의 시간(TM1)을 기록하고, 이를 SD2 프레임을 통하여 시간-슬레이브 노드들에게 전송한다.

• SD1 프레임를 수신한 시간-슬레이브 노드들은 마지막 바이트를 받자 마자 그 때의 시간(TS1)을 기록한다.

• 시간-슬레이브들은 두번째 동기화 메시지인 SD2 프레임을 수신한 시간(TS2)을 기록해 놓는다.

시간 슬레이브들은 다음의 공식으로 현재의 시간을 결정한다.

$$time = TM1 + (TS2 - TS1) + p - a + b \quad (8)$$

여기서 p, a, b는 각각 프레임 전파시간, 프로그램 처리 시간, SD1 프레임 처리시간을 나타내며, 실제 실험에 의하면 p, a, b 는 아주 작은 값이고 서로 상쇄되므로 무시 가능하다.

위의 알고리듬을 구현하기 위해 시간-마스터 노드에는 App() 함수 내에서 시간 동기화 메시지를 전송하도록 하였다. 시간-마스터 노드의 App() 함수 에서는 동기화 메시지 전송 주기마다 TIME_ SYNC 프레임을 2번 보내는 기능을 수행한다. 시간-마스터 노드는 SD1 프레임의 FC(Frame Control)의 function을 "TIME_ SYNC"로 설정하여 전송하 고, 그 때의 시간

인 time 변수 값을 TM1 변수에 복사 한다. 다시 SD2 프레임의 FC의 function을 "TIME_ SYNC"로 설정하고 TM1 변수 값을 데이터에 복사한 후에 전송한다.

시간-슬레이브 노드들은 SD1 프레임을 수신한 후에 FC안의 function이 "TIME_SYNC"로 설정되었는지 검사하여 "TIME_SYNC"로 설정되어 있으면, 그 때의 시간인 time을 TS1 변수에 복사한다. 시간-슬레이브 노드들은 또한 SD2 프레임을 수신한 후에 역시 FC의 function이 TIME_SYNC로 설정되었는지 검사 한다. FC의 function이 TIME_SYNC로 설정되었으면 그 때 의 시간을 TS2로 설정한 후에 (8)의 공식을 통하여 time변수를 설정한다. 이러한 기능들이 수정된 GetSD1 Frame()함수와 GetSD2Frame() 함수에 구현 되 었다.

그림 7은 대역폭 할당기법을 적용한 후 변화된 FDL 소프트웨어의 전체 구조이다.

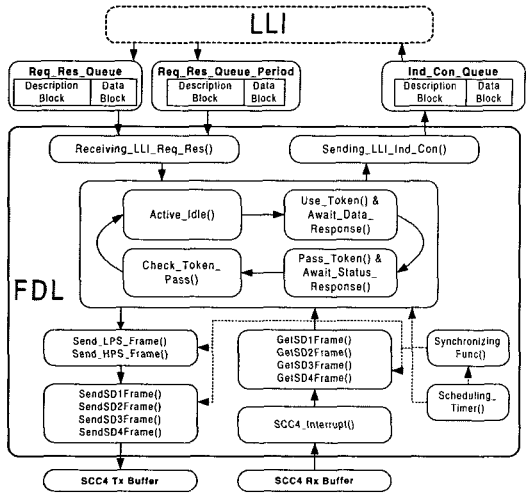


그림 7. 대역폭 할당 기법을 적용한 경우의 소프트웨어 구조.

Fig 7. Software structure with bandwidth alloction scheme.

VI. 실험 및 결과

본 장에서는 대역폭 할당 기법의 타당성을 실험 모델을 통하여 검증한 결과를 기술한다. 실험 모델은 열 개의 노드와 프레임을 모니터링하기 위한 모니터 보드로 구성된다. 그림 8에는 본 논문에서 사용된 실험 모델의 구조도가 나타나 있다. 노드 1, 3, 5, 7, 9에서는 산발적 실시간 데이터와 주기적 데이터가 생성되며, 노드 2, 4, 6, 8, 10에서는 비실시간 데이터와 주기적 데이터가 생성된다. 주기적 데이터는 10개의 노드에서 모두 생성된다. 그림 8의 구조도에서 10번 노드는 본래의 기능이외에 노드들간에 시간 동기화를 위한 시간-마스터 노드 역할을 병행하여 수행한다. 모니터는 매체를 통하여 전송되는 프레임의 상태를 모니터링하기 위한 노드이다. 모니터 보드는 매체를 통하여 전송되는 모든 프레임을 수신하여 이를 RS-232 통신을 통하여 PC에게 전달해주는 기능을 수행한다. PC에서는

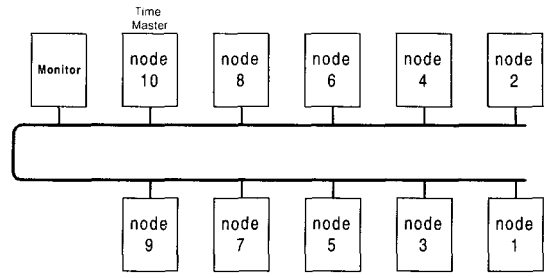


그림 8. 실험모델의 네트워크 시스템의 구조도.

Fig 8. Structure of the network system of ex perimental model.

이러한 프레임들을 분석함으로써 데이터 지연시간의 성능을 평가한다. 그림 9에는 실험 모델의 사진이 제시되어 있다.

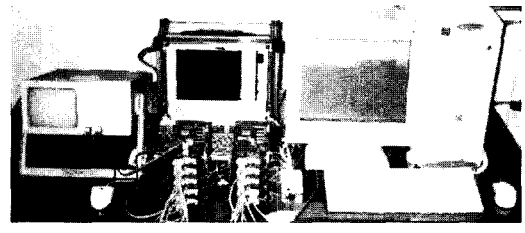


그림 9. 실험모델의 사진.

Fig 9. Picture of experimental model.

표 3에는 본 실험에 사용된 네트워크 트래픽의 조건이 나타나 있다. 구현된 FDL 프로토콜의 전송속도는 93750bps로 설정되었으며, 주기적 실시간 데이터의

표 3. 네트워크 트래픽 조건.

Table 3. Network traffic condition.

파라미터	값
전송속도	93750 (bps)
주기적 실시간 데이터 길이	85 (바이트)
산발적 실시간 데이터 길이	14 (바이트)
비실시간 데이터 길이	255 (바이트)
산발적 실시간 데이터의 도착 빈도	0.001 ($msec^{-1}$)
비실시간 데이터의 도착 빈도	0.002($msec^{-1}$)
토큰 오버 헤드	1 (msec)

길이는 85 바이트(헤더:11 바이트, 순수 데이터:74 바이트)이다. 실제 전송되는 데이터의 길이는 1바이트당 11비트이므로 935 비트 데이터 길이를 가지는 주기적 데이터의 전송시간 L_p는 처리시간 30비트 타임을 합하여 9.97msec(965비트 시간)이다. 여기서 처리시간이란 전송 큐에서 메시지를 꺼내오고 프레임을 만들어서 전송하는 데까지 소요되는 시간이다. 산발적 실시간 데이터의 길이는 14바이트(헤더:8 바이트, 순수 데이터:6 바이트)이다. 여기서 실제 전송되는 데이터의 길이는 154비트이고 따라서 산발적 실시간 데이터의

전송시간 L_c 는 처리시간을 합하여 1.96 msec(184비트 시간)이다. 비실시간 데이터의 길이는 일반적으로 가변적이지만 이 실험에서는 모든 노드에서 똑같이 프레임 최대 길이인 255바이트를 생성하는 것으로 가정한다. 각 노드에서 산발적 실시간과 비실시간 데이터의 도착 빈도는 각각 0.001msec^{-1}, 0.002msec^{-1}로 설정되었다. 토큰 오버 헤드는 토큰 처리 시간과 토큰 전송 시간을 합한 시간으로 본 연구를 통하여 소프트웨어적으로 구현된 프로토콜에서는 1msec가 소요된다. 산발적 실시간 데이터의 최대 허용 데이터 지연시간은 ϕ_c = 100msec이며, 10개의 노드에서 생성되는 주기적 데이터의 최대 허용 지연시간은 Φ = [100, 160, 200, 240, 400, 600, 800, 1000, 1600, 2000] msec로 주어지는 것으로 하였다.

이러한 트래픽 조건에 대하여 3장에서 제시한 대역폭 할당 알고리듬을 적용한 결과는 다음과 같다.

GIVEN :

N = 10, N_p = 10, N_c = 5, N_a = 5, L_p = 9.97msec, L_c = 1.96msec, L_a^i = 30.24msec, σ = 1msec, R = 10msec, ϕ_c =100msec, λ_c^i =0.001 msec^{-1}($i=1$에서 N_c),

Λ_a^i =0.002 msec^{-1}($i=1$ 에서 N_a),

Φ =[100,160,200,240,400,600,800,1000,1600,2000] msec

STEP 1 : 주기적 실시간데이터의 생성주기 T_i 를 구한다.

$\phi_1 = 100$

$T_1 = \phi_1 = 100$

$k_i = \left\langle \frac{\phi_i}{T_1} \right\rangle, \forall i=1$ 에서 N_p,

$\therefore k_i$ = [1, 1, 2, 2, 4, 4, 8, 8, 16, 16]

$\alpha_K = \sum_{i=1}^{N_p} \frac{1}{k_i}$ = 3.875

$\gamma = \lceil \alpha_K \rceil$ = 4

주기적 데이터의 안정화 조건을 점검한다.

$\gamma L_p + N_c L_c + R(=59.68) < T_1(=100)$ ⇒ 안정

$T_i = k_i T_1, \forall i = 2$ to N_p,

$\therefore T_i$ =[100,100,200,200,400,400,800,800,1600,1600]msec

STEP 2 : 각 노드에서 첫 번째 주기적 실시간 데이터 생성 시간 t_j 를 구한다.

$t_1 = A_1 = 0$

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

$$t_j = \inf[A_i \ge A_{i-1}; \mu^j(A_i) \le \gamma],$$

$\therefore$ t_j = [0, 0, 0, 0, 100, 100, 300, 300, 700, 700] msec

STEP 3 : 비실시간 데이터의 데이터 패킷 길이 L_a 를 구한다.

$$L_a \le \begin{cases} \dfrac{\phi_c - (\gamma L_p + N_c L_c + R)}{N - \gamma + 1}, & \phi_c < T_1 \\ \dfrac{T_1 - (\gamma L_p + N_c L_c + R)}{N - \gamma + 1}, & \text{otherwise} \end{cases}$$

$\phi_c = T_1$ 이므로 $L_a \le 5.76$,

$\therefore$ L_a = 5.13 msec (41byte: 오버헤드 포함 481bit times)

$$\lambda_a^i = \left\lfloor \frac{L_a^i}{L_a} \right\rfloor \Lambda_a^i = 0.012 \text{ msec}^{-1}$$

산발적 실시간 데이터와 비실시간 데이터 트래픽의 안정화 조건을 점검한다.

i) 산발적 실시간 데이터:

$$\lambda_c^i (=0.001) < \frac{1 - L_c \sum_{j=1}^{N_c} \min[\lambda_c^i, \lambda_c^j] - L_a \sum_{j=1}^{N_a} \min[\lambda_c^i, \lambda_a^j] - [(\alpha_K L_p + R)/T_1]}{R}$$

(=0.0478) ⇒ 안정

ii) 비실시간 데이터:

$$\lambda_a^i (=0.012) < \frac{1 - L_a \sum_{j=1}^{N_a} \min[\lambda_a^i, \lambda_a^j] - L_c \sum_{j=1}^{N_c} \min[\lambda_a^i, \lambda_c^j] - [(\alpha_K L_p + R)/T_1]}{R}$$

(=0.0196) ⇒ 안정

END

위에서 구한 결과에 따라 첫번째 T_{N_p} 주기 동안에 10개의 노드에서 생성되는 주기적 데이터의 데이터 생성 순간이 그림 10에 나타나 있다. 그림에서 보는 바와 같이 각각의 T_1 슬롯 내에서 생성되는 주기적 데이터의 개수는 $\gamma = 4$개를 초과하지 않으며, 이러한 데이터 생성 패턴은 T_{N_p} 의 주기로 반복된다.

네트워크 이용도 U 를 단위 시간당 네트워크에서 데이터 전송에 소요되는 시간의 비율로 정의하면, 산발적 실시간, 주기적 및 비실시간 데이터 트래픽으로 구성되는 필드버스 시스템에서 네트워크 이용도는 다음과 같이 나타내어진다.

$$U = \sum_{i=1}^{N_c} \lambda_c^i L_c + \sum_{j=1}^{N_p} \frac{L_p}{T_j} + \sum_{k=1}^{N_a} \lambda_a^k L_a \qquad (9)$$

실험모델의 예에서 산발적실시간, 주기적 및 비실시간 데이터의 네트워크 이용도는 각각 1%, 39%, 31%이며, 전체 네트워크의 이용도는 71%이다. 이러한 네트워크 이용도는 순수히 데이터를 전송하는 데 소요되는 시간만을 고려한 것이며, 토큰의 전송에 따르는 필수 불가결한 오버헤드 σ 는 제외시켰다. 따라서 실험모델에서 토큰 전달을 포함한 네트워크 부하는 90%를 초과하여 네트워크의 자원을 충분히 활용하는 경우이다.

본 연구에서 대역폭 할당 기법의 실험은 알고리듬에 의해서 산출된 각 파라미터를 적용한 경우(알고리듬)와 기존의 PROFIBUS 프로토콜을 그대로 사용한경우(PROFIBUS)에 대하여 비교 하였다. 표 4에는 (알고리듬)과 (PROFIBUS)에서 데이터 지연시간에 대한 성능 비교가 나타나 있다. 표 4에서 실패 빈도란 데이터 지연시간이 최대 허용 지연시간을 초과하는 경우를

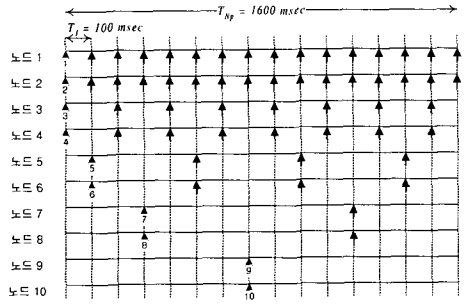


그림 10. 주기적 데이터의 생성 패턴.
Fig 10. Generation pattern of periodic data.

말한다. (PROFIBUS)의 경우에서 산발적 실시간 데이터는 high 우선순위로 전송되며, 주기적 데이터와 비실시간 데이터는 low 우선순위로 전송되는 것으로 하였다. PROFIBUS의 FDL 파라미터인 TRT(Token Rotation Timer) 값은 각 노드에서 최대 프레임 1개의 전송시간과 slot time을 더하고, 이를 스테이션 수로 곱한 값으로 설정하였다.

주기적 데이터의 경우 전송 지연 시간이 데이터 생성 주기보다 작아야 성공적으로 전송될 수 있다. 주기적 데이터의 전송큐 용량은 1로 제한되며, 전송지연시간이 데이터 생성 주기보다 큰 경우에는 전송큐 내에서 대기하고 있던 데이터가 새로이 생성된 데이터에 의하여 대체되어 데이터 전송 실패가 발생한다. 본 실험에서 (PROFIBUS)의 경우 주기적 데이터는 최대 허용 지연 시간의 주기로 생성되도록 하였다. 표 4에서 주기적 데이터의 경우를 보면, (PROFIBUS)의 경우에는 노드 1, 2, 3, 4에서 최대 전송 지연 시간이 167, 257, 478, 383msec으로 최대 허용지연 시간인 100, 160, 200, 240msec를 초과하여 최대허용 전송지연시간의 요구를 충족시키지 못하고 있으며, 이로 인하여 약 14%의 데이터가 전송 실패되는 현상이 발생한다. 반면에

표 4. 알고리듬, 표준 PROFIBUS의 성능 비교.
Table 4. Performance comparison for algorithm and PROFIBUS.

	평균전송지연		최대전송지연			실패율		
	알고리듬	PROFIBUS	알고리듬	PROFIBUS		알고리듬	PROFIBUS	
산발적실시간	239	218	32	31		0.0520	0.0504	
주기적	60.24	86.44	56	노드1	167	0.0074	노드1	3447/1238
				노드2	257		노드2	2137/1297
				노드3	478		노드3	97/646
				노드4	383		노드4	73/648
				노드5	177		노드5	0.020
				노드6	514		노드6	0.019
				노드7	108		노드7	0.0161
				노드8	471		노드8	0.0163
				노드9	408		노드9	0.061
				노드10	372		노드10	0.079
비실시간	112.71	94.73	99	736		0.0276	0.0268	

(알고리듬)의 경우에는 그림 10에서 보는 바와 같이 T_1 슬롯동안 발생되는 주기적 데이터의 개수를 γ (=4)개 이하가 되도록 데이터 생성 주기와 최초 데이터 생성 시간을 스케듈링하여 모든 노드에서 최대 전송 지연시간이 95ms를 초과하지 않으며, 따라서 전송 실패가 발생하지 않는다.

산발적 실시간 데이터의 경우에는 (알고리듬)과

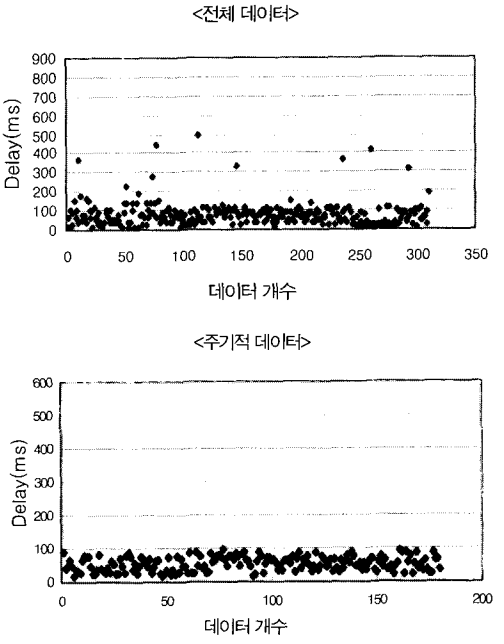


그림 11. 알고리듬의 지연시간 분포도.
Fig 11. Delay distribution in Algorithm.

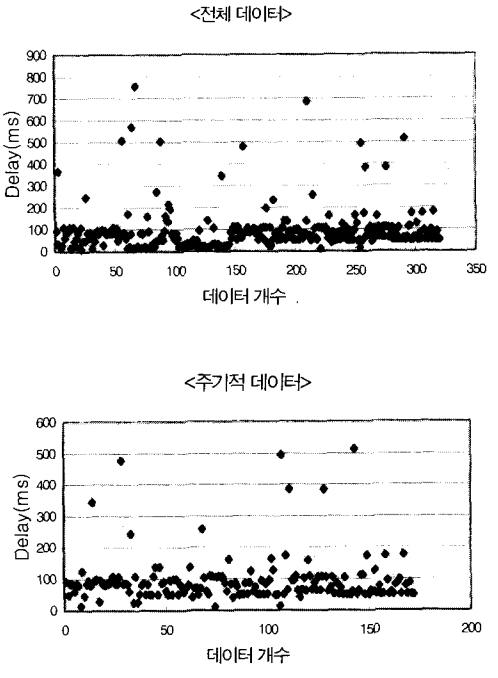


그림 12. PROFIBUS의 지연시간 분포도.
Fig. 12. Delay distribution in PROFIBUS.

(PROFIBUS)의 두 경우 모두 최대 전송지연 시간이

허용 지연시간의 한계인 100msec를 넘지 않는 것을 볼 수 있다. 이는 산발적 실시간 데이터가 두 경우에서 모두 high 우선순위로 처리되었기 때문이다. 비실시간 데이터의 경우에는 (알고리듬)의 경우에 데이터 지연 시간의 평균값이 (PROFIBUS)에 비하여 증가하는 것으로 나타나 있으나, 비실시간 데이터의 전송 지연 시간이 응용시스템에 미치는 영향은 중요하지 않다. 표 4에서 보는 바와 같이 대역폭 할당 기법은 데이터 지연시간이 중요하지 않은 비실시간 데이터의 전송지연 시간을 증가시키는 대신에, 실시간 데이터 전송을 요구하는 산발적 실시간 데이터와 주기적 데이터의 전송 요구사항을 만족시킨다.

그림 11과 12는 각 (알고리듬)과 (PROFIBUS)의 경우에 데이터 지연시간의 분포도로서 X축은 발생한 데이터의 빈도이고 Y축은 데이터 전송 지연 시간이다. (알고리듬)의 경우에는 주기적 데이터가 최대 지연시간이 100msec를 초과하지 않는 반면, (PROFIBUS)의 경우에는 (알고리듬)과 비교하여 주기적 데이터 지연시간의 분포가 상대적으로 넓음을 알 수 있다.

Ⅶ. 결론 및 추후 연구 과제

본 논문에서는 먼저 PROFIBUS의 데이터링크 계층인 FDL 프로토콜을 구현하였고, 제한된 대역폭에 산발적 실시간, 주기적 실시간 및 비실시간 데이터 트래픽을 적절히 할당할 수 있는 대역폭 할당 기법이 PROFIBUS에 실제로 적용될 수 있음을 실험을 통하여 검증하였다. PROFIBUS 데이터 연결 계층은 소프트웨어로 구현되었으며, 대역폭 할당 기법에서 제시하는 주기적/비주기적 구간을 구분하는 기능을 삽입하기 위해서 SD4 프레임을 수신하는 함수인 GetSD4Frame()과 SD4 프레임을 전송하는 함수인 SendSD4Frame() 함수를 수정하였다. 또한, 노드 시간 동기화 기능을 추가하기 위해서 인터럽트 처리 함수 TimerPC0(), Timer1()함수를 추가하였고, 주기적 데이터 처리 기능을 삽입하기 위해서 Use_Token() 함수를 수정하였다.

실험결과에 의하면 대역폭 할당 기법은 데이터 지연시간이 중요하지 않은 비실시간 데이터의 전송지연 시간을 증가시키는 대신에, 실시간 데이터 전송을 요구하는 산발적 실시간 데이터와 주기적 데이터의 전송 요구사항을 만족시키는 것으로 나타났다. 또한, 실시간 데이터의 전송 지연시간 요구조건을 만족시키면서 네트워크의 자원을 충분히 활용할 수 있음을 보였다.

본 연구에서 실험은 93750bps의 전송 속도로 수행하였다. 이는 문자단위로 처리하는 UART 전송방식의 특성상 수신 인터럽트가 자주 발생할 수밖에 없으며, 이를 소프트웨어로 처리하기에는 한계가 있기 때문이다. 그러나 UART 처리부분 만을 ASIC화한다면 본연구를 통하여 개발된 FDL 프로토콜은 PROFIBUS-FMS의 최대 전송 속도인 500kbps로 동작되는데 아무 문제가 없다. 본 연구를 통하여 구현한 필드버스 실험 모델에서 각 보드는 FDL 계층만을 포함하고 있으며, 실험을 위하여 인터럽트 처리 함수에 의해서 산발적 실시간, 주기적 및 비실시간 데이터가 전송 큐에 삽입되도록 하였다. 추후 연구로는 산발적 실시간, 주기적 및 비실시간 데이터의 생성 기능이 상위 계층인 필드버스의 사용자 계층에서 처리될 수 있도록 하는 시스템 관리 기능을 제시하는 것이다.

참고문헌

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

[2] S. H. Hong and W. H. Kim, "Bandwidth allocation scheme in the CAN protocol" *IEE Proceedings-Control Theory and Applications*, vol. 147, pp. 37-44, Jan, 2000.

[3] S. Cavalieri, A. Di Stefano, and O. Mirabella, "Optimization of acyclic bandwidth allocation exploiting the priority mechanism in the fieldbus data link layer," *IEEE Trans. Industrial Electron*, vol. 40, no. 3, pp. 297-306, June, 1993.

[4] K. Kobayashi, M. Kameyama, and T. Higuchi, "Communication network protocol for realtime distributed control and its LSI implementation", *IEEE Trans. Industrial Electron*, vol. 44, no. 3, pp 418~426, June, 1997

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

[6] Klaus Bender, "PROFIBUS the fieldbus for industrial automation" Prentice Hall, 1993

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

[8] O. C. Ibe and X. Cheng, "Stability conditions for multiqueue systems with cyclic service," *IEEE Trans. on Automatic Control*, vol. 33, no. 1, pp. 102-103, January, 1988.

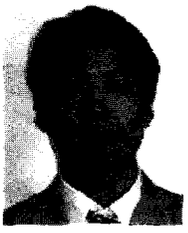
[9] Documentation Set for the 68000 Family Assembler/Linker/Librarian/C,C++ Compiler, Mi crotec Reaserch Inc., 1995.

홍 승 호

1956년 5월 31일생. 1982년 연세대학교 기계공학과(공학사). 1985년 Texas Tech Univ 기계공학과(공학석사). 1989년 Pennsylvania State Univ. 기계공학과(공학박사). 1992년~현재 한양대학교 전자컴퓨터공학부 부교수. 관심분야는 산업용 통신망.

김 지 용

1996년 한양대학교 제어계측공학과 졸업. 동 대학원 석사(1998). 1998년~2000 LG 산전 중앙연구소 연구원. 2000년~현재 원클릭테크놀로지스연구원. 관심분야는 필드버스 및 PLC.



김 유 철

1997년 한양대학교 제어계측공학과 졸업. 동 대학원 석사(2000). 2000년~현재 아이콘트롤스 연구원. 관심분야는 필드버스 하위계층 및 하드웨어분야.