工學碩士學位論文

페트리 네트를 이용한 필드버스 시스템의 성능 해석과 설계

李 在 壽

江原大學校 大學院

1995年 2月

朴 洪 聖　教 授 指 導

工 學 碩 士 學 位 論 文

페트리 네트를 이용한 필드버스 시스템의 성능해석과 설계

Performance Analysis of Fieldbus system using Petri-Net and Design

江 原 大 學 校　大 學 院

制御計測工學科 制御計測專攻

李　　在　　壽

李在壽의 碩士學位論文을

合格으로 判定함

1994 年 12 月

審査委員長 남 부 희 

委 員 박 홍 성 

委 員 최 명 환 

Performance Analysis of Fieldbus system using Petri-Net and Design

LEE JAE SOO

Department of Control and Instrumentation Engeering

Graduate School, Kangwon National University

Summary

This paper derives to the mean token rotation time and the mean service time of a station using an analytic methods in the fieldbus. These performance measures are represented in terms of the number of stations, the token holding time(THT), the token rotation time(TRT), and the mean arrival rate and mean service time of messages. This paper presents the Petri-Net(PN) model for a Datalink layer of the fieldbus. The mathematically forms of the mean token rotation time and the mean service time of the station are presented this PN model and the moment generating function.

목 차

그림 목차

표 목차

1. 서 론

현재 생산현장에서는 인간을 대신하여 수많은 자동화기기들이 대량생산을 위해 사용되고 있다. 그러나 이들 기기들은 각 회사별로 독특한 하드웨어나 소프트웨어들로 구현되어 있기 때문에, 기기 상호간의 접속이 원활하진 못하다. 이를 보완하기위해 MAP(Manufacturing Automation Protocol)[1]이 제안되어, 많은 생산라인에서 사용되고 있다. 그러나, 최근 스마트 센서가 생산라인에서 많은 비중을 차지하게 되어감에따라, 스마트 센서를 이용할 수 있는 통신 시스템이 필요하게 되었다. 즉, 맵(MAP)보다 더 낮은 레벨의 통신규약이 필요하게 되었다. 그리고, 그러한 요구를 충족시키기 위해 필드버스(FieldBus)가 제안되었다[2][3]. 각 통신규약별로 사용가능 레벨을 비교해보면 표1-1.과 같다.

표1-1. 통신규약별 처리 레벨 비교

Network의 종류	처 리 레 벨
MAP	factory controller
Mini-MAP	cell controller
FieldBus	device, sensor, actuator

필드버스는 국제적인 표준화 작업이 진행중에 있다[2][3]. 그러나, 서로 자국의 표준안을 국제 표준화 모델로 삼기위해 경쟁하고, 미래형 필드기기까지 고려하고 있기 때문에 국제 표준이 늦어지고 있다. 본 논문에서는 ISA(Instrument Society of America)/IEC(International Electrotechnical Comission)에서 자체적으로 작성하고 있는 표준안[2][3]을 토대로 필드버스 시스템을 해석한다.

미니맵(Mini-MAP)과 필드버스는 서로 유사한 특성을 갖는다. 먼저, 개방형 통신(OSI:Open Syaytem Interconection)참조 모델을 비교해보면 모두 3계층만(물리층, 데이타링크층, 응용층)을 포함하는 형태를 갖는다. 그러나, 미니맵에서는 응용계층에 MMS(Manufacturing Message Specification)만을 적용하며 기기제어용으로 사용한다.

그리고, 접속하는 기기들로는 PLC, 로봇, CNC/MC, 셀제어기등이 있다. 반면에 필드버스는 보다 생산라인에서 직접적으로 사용되는 센서나 액츄에이터 등과 같은 장비들과 접속하며 통신해야하기 때문에(이들 필드기기들은 데이타 길이가 짧으며(20바이트이내), 응답시간 역시 짧다(msec단위)) S/W 접속 및 계산 등에 많은 시간이 소요되는 MMS를 사용하기는 힘들다. 그래서, 다음과 같은 요구사항을 만족시키는 독창적인 방식을 제안하고 있다.

- 짧은 데이타전송에 비하여 전송 오버헤드가 상대적으로 작아야한다.
- 주기적 데이타와 비주기적 데이타의 동시 처리가 가능해야한다.
- 데이타에대한 에러 제어 기능이 있어야한다.
- 기기들간의 동기를 위한 타이밍 신호를 제공할 수 있어야한다.
- 충분한 대역폭(혹은 채널용량)을 가지고 있어야한다.
- 고장에 대비하여 예비 장비를 보유하여야한다.
- 신호선을 통하여 전력을 공급할 수 있어야한다.
- EMI 환경하에서도 동작되어야하며, 충분한 안정성이 유지되어야한다.

상기와 같은 기능 요구조건을 갖춘 필드기기간의 통신을 위한 필드버스가 요구된다.

1.1. 기존의 연구 결과

필드버스 데이타링크층을 페트리네트를 이용하여 모델링하거나, 전송율(throughput)과 접근시간(access delay)을 성능지표로써 평가한 연구들이 있다. Stefano의 경우[4], 주기적/비주기적 처리를 모두 하는 필드버스 시스템의 데이타링크층 모델을 제시하였다. 이 모델은 시스템의 안정성을 보이기 위한 것으로, 우선순위의 할당량을 변화시켜가며 각 상태별 전송율을 구하였다. Stefano는 모의실험을 통해 필드버스 설계시 각 우선순위의 특성을 알아보고, 각각의 할당량에 따른 특성을 보였다. 이 연구에서 Stefano는 시간제약이 큰 정보의 선달에는 DeT(Delegated Token)를 사용하고,

그 제약이 크지 않은 경우에는 CiT(Circulated Token)를 사용하는 것이 필드버스 시스템에 적합하다는 것을 보였다. 그러나, 이 결과는 수학적 결과가 아닌 제시한 모델을 이용한 모의실험 결과를 해석해서 얻은 것이다. Cavalieri[5]는 필드버스 데이타링크층의 접근 메카니즘에 전송율과 접근시간으로 정해진 우선순위를 이용하여, 가정한 각 상태들의 차이를 분석하여 필드버스의 성능해석을 하였다. Stefano와 마찬가지로 각 우선순위 할당량 조절을 이용하여 분석하였지만, 우선순위를 동적 할당과 정적 할당의 두가지 방법으로 우선순위를 할당하고 모의실험을 통해 비교하여 각 할당방법의 장점을 살피고, 대역폭 할당의 최적화 방법의 한 예로써 보였다. 그러나, Stefano의 경우와 마찬가지로 제시한 모델을 이용한 우선순위 할당법만을 보였으며, 모의실험만을 이용하여 비교·분석하였다. 위와같은 연구들에서는 모두 모의실험 방법을 이용하여 필드버스 시스템을 평가한 것으로 수학적 형태로의 해석은 하지 않았다.

시스템의 성능을 평가한 연구들을 살펴보면, Elnakhai[6]는 MAP에서의 성능 평가를 한 것으로 한 스테이션에서 토큰 이동시 지연되는 시간을 구하는 데 중점을 두었다. Elnakhai는 MAC(Medium Access Control)에서의 즉각 응답 옵션(immediate response option) 이용시의 장점을 나타내기 위해 서비스 요구 발생율과 네트웍에서 제공되는 통화량의 총합과 토큰 이동 시간(token walk time)을 이용하여 평균 토큰 회전시간과 대기시간을 구하였다. Valenzano[7]는 필드버스와 매우 비슷한 미니맵을 모델로 시스템에서의 서비스 시간을 구하였다. 여기서는 우선순위 메카니즘이 긴급(Urgent) 통화량에 대한 응답시간을 개선시킬 수 있음을 보였고, 이를 이용하여 미해결 요구의 최대치가 낮아지도록 모델을 제시하고 분석하였다. 각 스테이션에서의 서비스 요구 발생시간과 그 서비스 요구에대한 확인이 되돌려지는 시간이 모두 알고 있다는 가정하에서 모의실험만을 이용하여 각 값들을 구하였다. Montushi[8]는 TRT(Token Rotation Time), HPTHT(High-Priority Token Holding Time)과 TTRT(Target Token Rotation Time)의 관계를 이용하여 분석하였다. 동기적 메시지는 패턴의 특성이나 주기의 길이를 이용하는 데 제한을 두지 않는 일반적인 주기적 패턴으로 발생된다고 분석하였다. 또한 이 관계들을 이용하여, 각 우선순위하에서 스테이션이 토큰을 갖는 시간을 얻었다. On-Ching[9]의 경우, 맵에서 명시된 시간을 갖는 토큰 버스 네트웍에서의 시간지

연을 비대칭/대칭 경우에 대해 수학적으로 근사화하였다. On-Ching은 근사화된 값들을 맵에서 사용되는 타이머 값의 설정시의 참고자료로써 제시하였다. Guo[10]는 구성된 페트리네트 모델을 수학적으로 평가할 수 있는 방법을 제안하였다. 페트리네트 모델에서 얻을 수 있는 상태도는 XOR GERT(exclusive-OR Graphical Evaluation and Review Technique) 네트웍 모델과 동일하므로, XOR GERT에서 사용되는 정리들을 구성한 모델에 적용시킬 수 있다는 것을 제안하였다. 그리고, 이러한 정리에의해 각 상태의 전달함수를 구하여 시스템을 해석하고, 그 시스템을 다시 하나의 전달함수로 바꾸어서 전체 시스템을 해석하는 방법을 제시하였다.

1.2. 연구 방향

본 논문에서는 필드버스에서 사용되는 프로토콜[2][3]을 살펴, 그 데이타의 흐름과 동일한 형태의 모델을 설정하여 기존 연구[4][5]보다는 수학적 평가가 가능한 시스템으로 모델링해 나가겠다. 그러나, 필드버스에서 사용될 수 있는 모든 토큰에대한 모델 제시와 해석에는 많은 어려움이 있으므로, 본 논문에서는 필드버스에서 주기적 데이타교환시 사용되는 CiT(Circulated Token)에 대한 모델만을 제시하겠다. 그리고, 제시한 모델을 TRT(Token Rotation Time)와 THT(Token Holding Time), 그리고 Guo[9]가 제안한 성능해석 방법을 이용하여, 평균 서비스 시간과 평균 토큰 회전시간을 구해 보겠다. 페트리네트로 구성된 모델의 각 상태변환시 전달함수를 구하여, 그 상황에 대한 모멘트 발생함수를 얻는다. 구한 모멘틀 발생함수를 이용하여 보다 큰 상태변화에 대한 전달함수를 얻는다. 이러한 방법으로 제시된 모델에서 모든 토큰 전달시의 전달함수들을 메이슨의 정리를 이용하여 하나의 전달함수로 변환하고, 그때의 모멘트 발생함수를 이용하여 한 스테이션에서의 평균 서비스 시간을 구한다. 그리고, 시스템은 동일한 모델을 가진 스테이션들로 구성되기 때문에 평균 서비스 시간에 따른 전달함수로 바꿀 수 있다. 안정상태에서 각 스테이션들은 동일한 서비스 시간을 갖는다고 볼 수 있으므로, 평균 토큰 회전시간을 구할 수 있다. 본 논문에서는 이러한 방법으로 한 스테이션에서의 평균 서비스 시간과 평균 토큰 회전시간을 구한다.

그리고, 표준안[11]에 따라 직접 필드버스 데이타링크층을 설계해 보겠다.

1.3. 논문의 구성

본 논문은 다음과 같은 구성으로 되어 있다. 2장에서는 연구대상으로 삼은 필드버스에대한 각 계층별 동작이 설명되고, 3장에서는 성능해석시 사용한 방법인 페트리네트상에서의 모멘트 발생함수 이용에 대해 설명한다. 4장에서는 페트리네트에 대해 설명하고. 5장에서는 모델링에 필요한 가정을 밝히고, 페트리네트를 이용하여 한 스테이션에서의 필드버스 데이타링크층을 모델링한다. 6장에서는 5장에서 제시한 모델의 성능을 평가하고, 모의실험과 비교한다. 7장에는 필드버스 데이타링크층의 소프트웨어적 설계 방법이 나오고, 마지막으로 8장에서는 결론이 나오게 된다.

2. 필드버스의 개요

2.1. 필드버스의 각 계층별 개요

필드버스는 "필드에 설치된 필드기기(스마트 센서, 엑츄에이터, 단일 루프 조절계 같은 계측기기 모터나 접점의 제어기기)와 계기실에 설치된 상위의 제어기기 (DCS(Distributed Control System), PLC, operator station)간을 연결하는 디지탈 시리얼 방식의 통신 방법"이라고 정의할 수 있다. 이러한 필드버스는 필드 정보의 질과 양의 향상, 원격 유지 및 보수, Wiring의 간소화, 다른 필드기기들의 디지탈화, 보다 고도의 계측제어 시스템의 구축으로인한 안정한 플랜트 운용의 기대 등의 장점을 얻기위해 사용된다.

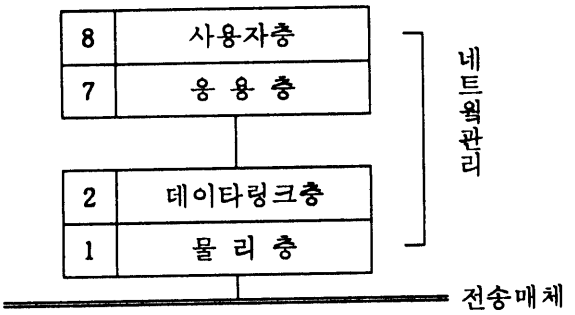


그림 2-1. 필드버스의 구조

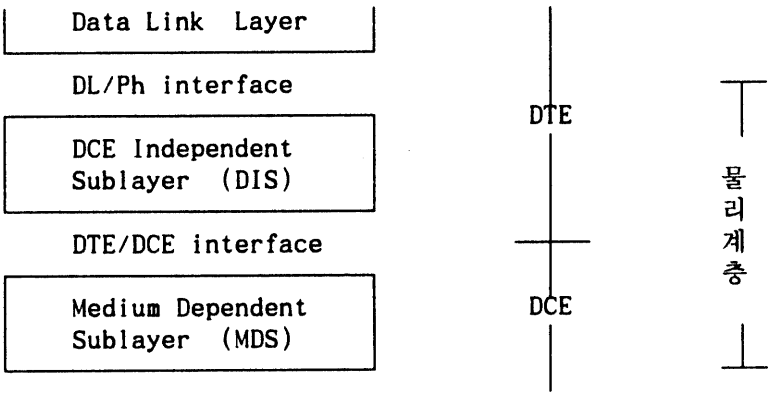
그림 2-1.에서 보는 바와 같이 필드버스는 4개의 구조로 되어 있다. 이 모델은 미니맵과 거의 비슷하다. 그러나, 사용하는 곳을 보면 미니맵은 제어기의 레벨까지 사용이 가능한 반면, 필드버스는 셀 레벨까지 사용이 가능하다. 그리고, 필드버스에는 다른 OSI 참조모델과는 달리 8계층(사용자층)이 있다.

각 계층별 구조를 살펴보면 다음과 같다.

1) 제1층 (물리층)

데이타 링크 계층으로부터 심볼을 받고 전송을 위하여 이들을 전기적 신호로

변환하며, 그 구조는 다음의 그림 2-2와 같다.



Data Terminal Equipment(DTE)
Data Communication Equipment(DCE)
DIS(DTE Independent Sublayer)
MDS(Medium Dependent Sublayer)

그림2-2. 물리층의 구조

여기서 각 계층의 역할을 살펴보면 다음과 같다.

DIS 부계층은 데이터링크층에 데이터 단위를 전달하고 인코딩된 물리계층 신호를 DIS/MDS 인터페이스를 통하여 MDS로 전달한다. 그리고, MDS 부계층은 도달한 물리적 신호를 DIS로 전달해주고, DIS에서 전달된 신호를 전송매체로 전달한다.

전송속도에따른 데이타 교환 방식을 살펴보면 다음과 같은 4가지가 제안되어 있다.[12] 각 비트당 전송율과 사용 신호의 종류에 따라 구분된다. 비트당 전송율에는 31.25 KBPS(bit/sec), 1.0 MBPS, 2.5 MBPS등이 있으며, 신호전달시 사용되는 기준은 전압차에 의한 방법과 전류의 크기에 의한 방법이 있다.

① 31.25 Kbit/s, 전압 모드(DC)인 경우

전 송 매 체 : twisted-pair cable

연결 장치의 수 :

- 전원을 공급하지 않는 경우 : 최대 32개
- 전원을 공급하지만 위험한 경우 : 최대 6개
- 전원을 공급하는 경우 : 최대 12개

케이블의 최대 허용 길이 : 1900 m

② 1.0 Mbit/s, 전압 모드(DC)

전 송 매 체 : shielded twisted-pair cable

연결 장치의 수 (전원 공급 안함) : 최대 32개

장치간 최대 허용 길이 : 750 m

③ 1.0 Mbit/s, 전류 모드(AC)

전 송 매 체 : shielded twisted-pair cable

연결 장치의 수 (전원 공급 여부 관계없이): 최대 32개

장치간 최대 허용 길이 : 750 m

정상적인 신호로 인정하는 전류 : 1.3 mA

④ 2.5 Mbit/s, 전압 모드(DC)

전 송 매 체 : shielded twisted-pair cable

연결 장치의 수 (전원 공급 안함) : 최대 32개

장치간 최대 허용 길이 : 500 m

2) 제2층 (데이타링크층)

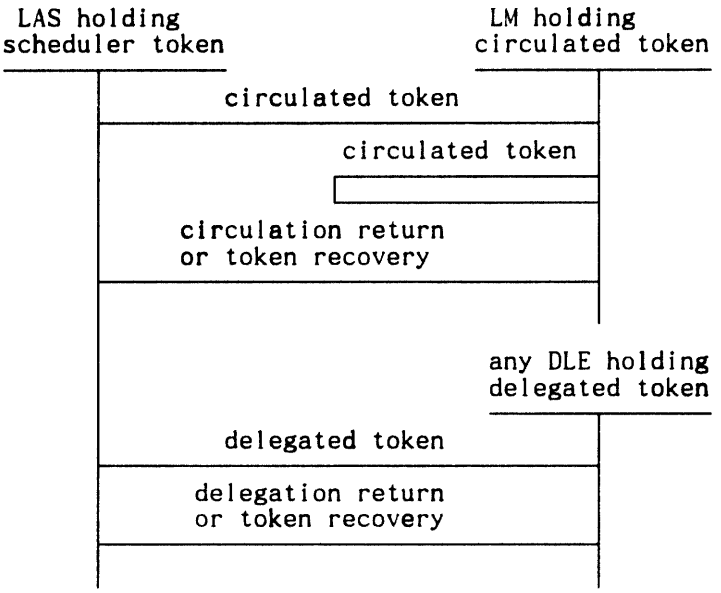
데이타 교환과 메시지전달을 제공하며, 세개의 부계층으로 구성된다. 연결 프레임 부계층(Link framing sublayer)에서는 상위 계층과 물리층사이의 프레임 검사 작업(frame check sequence)을 담당한다. 연결 접속 부계층(Link access sublayer)에서는 시작자(initiator)와 응답자(responder)의 기본적인 기능을 제공한다. 물리 매체 접속에 대한 메카니즘 운영을 맡는다. 연결 스케쥴링 부계층

(Link scheduling sublayer)은 데이타링크층의 활동을 스케쥴하는 데 필요한 기능들을 제공한다.

데이타링크층에서 제공되는 기능은 기본적으로 응답자(Responder)로의 기능, 시작자(Initiator)로의 기능, 브릿지(Bridge)로의 기능, 링크 마스타(Link Master)로의 기능등 4가지로 구분될 수 있다. 먼저 응답자로의 기능은 다른 스테이션으로부터 전달된 것에대한 응답을 담당하는 것이고, 시작자로의 기능은 다른 스테이션으로 통신을 시작하는 데 이용하는 것이다. 브릿지로의 기능은 현재 자신이 속한 필드버스 시스템이 아닌 다른 필드버스 시스템으로의 연결시 이용되며, 링크 마스타로의 기능은 데이타링크층에서의 작업수행이다. 여기서 링크 마스타로의 기능은 LAS(link active scheduler)로의 기능과 LM(Link Master)으로의 기능으로 구성된다. LAS는 자신이 가진 토큰을 이동시켜 연결된 LM들을 관리하며, LM은 LAS로부터 받은 토큰을 이용하여 할당된 작업을 수행한다.

작업수행시 사용되는 토큰에는 CiT(Circulated Token)와 DeT(Delegated Token)이 있다. CiT는 한번에 최대 8개의 스테이션들만이 이용가능하고, 그외의 스테이션들은 이 스테이션이 CiT를 이용하는 동안 DeT가 이용된다. CiT의 이용은 LAS로 CiT 요구가 발생했을 때 순차적으로 전달된다.

이와같은 토큰과 토큰 사용 상태를 다음과 같이 나타낼 수 있다.



3) 제7층 (응용층)

통신 지점에 있는 AP(application process)들 사이(제어소자와 필드소자들사이)의 정보교환 기능을 한다. 이 계층을 기능별로 분류하면 다음과 같이 두개의 응용 서비스 요소로 되어있다. 먼저 블럭 억세스 부계층은 대상(object)들의 발생, 소멸, 정의 및 사용하는 대상처리 동작을 수행하고, 기기 및 네트워크 블럭 영역에 있는 사용자 소자에 대한 접속을 한다. 통신 부계층은 PDU(protocol data unit) 발생, PD 해석 등과 같은 PDU 처리와 데이타링크층으로 접속하는 기능을 한다.

가장 중요한 특징은 현재 사용중이거나 앞으로 사용이 예상되는 모든 계측 및 제어에 관련된 필드장비들을 객체지향(object-oriented) 접근방식으로 모델링하여 공장자동화 및 공정제어시스템에 응용될 수 있는 폭을 넓힌 것이다. 응용계층에는 또한 네트워크의 구성, 성능, 고장등을 관리하는 네트워크 관리기능이 부과된다. 응용계층에서 생성되는 AL-PDU(Application Layer Protocol Data Unit)을 기술하기위한 추상구문과 전송구문으로는 각각 ASN.1 (ISO 8824)와 ASN.1/BER(ISO 8825)를 채택하였다.

4) 사용자층

OSI에서 제공하지는 않은 계층이지만, 사용자 입장을 고려하여 이미 사용하고 있던 제어 알고리즘을 변경없이 사용하거나 이식할 수 있게 하기위해 추가된 계층이다. 데이타, 정보의 종류를 지정하고 이것이 어떻게 이용되는 가를 지정한다. 즉, 기능블록(functional Block) 구현(analog I/O, 카운터, 타이머, PID, 보상, Discrete I/O등), 데이타베이스기능 구현, 실시간성 보장을 위한 기능 블럭 스케쥴링등을 말한다. 여기서는 기본적으로 유사한 필드 디바이스는 정해진 호스트 컴퓨터 상에서 동등한 기능을 갖아야 하기 때문에 디바이스류의 상호운용성(Interoperability)이 매우 중요하다. 이것은 유사한 필드 디바이스를 교환해도 기능상 변하지 않는 운용을 가능하게 하는 것을 보증하는 것이다.

2.2. 데이타링크층에서의 토큰 종류에 따른 동작

1) CiT(Circulated Token)

CiT는 한 주기의 기간동안 데이타링크 트랜잭션(transaction)을 시작할 권리를 제공한다. Circulated Token DLPDU(CTN, CTD)는 다음과 같이 CiT를 통과시키는 데 사용된다. 먼저 LAS DLE로부터 LM DLE(Data link entity)로 토큰이 전달되는 경우는, LM DLE가 CiT를 갖고있는 동안 마지막으로 CIRCULATION REPLY DLPDU를 보낸 LM DLE일 때와 LAS DLE로서 같은 노드에 있는 LM DLE의 초기화가 필요한 경우이다. 토큰은 현재 CiT를 갖고 있는 LM DLE로부터 순서상 다음 차례의 주소를 갖고있는 LM DLE로 전달된다. 단, CIRCULATED TOKEN DLPDU는 브릿지로는 전달되지 않는다.

CIRCULATED TOKEN DLPDU가 LAS DLE에의해 보내질 경우 LAS DLE는 다음과 같은 행동을 취한다. 스케쥴이 링크에 존재할 경우 LAS DLE는 CTD DLPDU에 존속기간 매개변수를 포함시킨다. 그 매개변수의 값은 CTD DLPDU가 실제로 전달되는 시간에서 CiT가 통과되거나 리턴되는 데 요구되는 시간량을 갖는다. 그러나, 링크상에 스케쥴이 존재하지 않을 경우 LAS DLE는 DLPDU로부터 존속기간 매개변수를 생략한 CTN DLPDU를 보낸다.

CiT DLPDU가 LM DLE에의해 보내질 경우, LM DLE는 LM이 현재 CiT를 갖고 있지만 토큰을 더이상 사용할 필요가 없을 경우 또는 다음 트랜잭션에서 요구되는 최대 링크 용량성(capacity)이 토큰 사용 가능 시간보다 클 경우 LAS로 토큰을 되돌린다.

토큰을 전달받는 DLE가 LM DLE이고 존속기간 매개변수를 가질 경우는 그 값을 변수에 저장하여 작업을 수행하는 동안 카운트해간다. 이 값이 현 LM이 사용해야할 시간보다 적은 경우, 토큰을 다음 LM이나 LAS로 넘긴다. 존속기간을 갖지 않을 경우는 현 LM의 토큰 사용 시간동안 작업을 수행하고, 다음으로 토큰을 넘긴다.

2) DeT(Delegated Token)

DeT는 주어진 시간동안 LAS로 그 사용을 요구한 LM으로 전달되어 사용된다. CiT 사용후 토큰사용이 계속 요구되거나, 특별한 서비스 요구에대한 서비스를 수행하고자 할 때 DeT가 요구된다.

LAS는 요구한 LM으로 토큰을 보내고, 링크가 활성화되는지를 검사한다. 만일 링크가 활성화되지 않는다면 다시 토큰을 보내고 활성화여부를 검사한다. 그래도 실패할 경우 DL-management에 실패임을 알리고 LAS로서의 동작을 재개한다. 링크가 활성화될 경우는 두 슬롯 시간동안 DeT가 되돌려지길 기다린다. 그러나 그 시간동안 토큰이 되돌려지지 않을 경우는 DL-management에 실패임을 알리고 LAS로서의 동작을 재개한다. 만일 DeT를 받는 LM이 CiT나 DeT를 이미 갖고 있었다면, 모든 서비스를 중단시키고 토큰을 제거시킨다.

3. 페트리네트 상에서의 모멘트 발생함수

Guo[9]가 제안한 성능 평가방법을 살펴보면 다음과 같다.

모멘트 발생 함수는

$$M(s) = \int_{-\infty}^{\infty} e^{st} f(t)\, dt = E(e^{st})$$

- s : 확장된 매개변수
- f(t) : 랜덤 변수 t의 확률밀도함수

$$M(0) = \int_{-\infty}^{\infty} f(t)\, dt = 1$$

로 정의된다. 그리고, n번째 모멘트는 모멘트 발생 함수의 n번째 편미분에 의해 구할 수 있다.

$$\frac{\partial^n}{\partial s^n} M(s)|_{s=0} = E(t^n)$$

트랜지션 t_k가 점화할 확률을 $P(t_k)$로 놓자. 그러면, 이 확률은 트랜지션 t_k를 통해서 P_I에서 P_J로 변환되는 트랜지션 확률이 된다. 활성화 상태에서 트랜지션 t_k의 점화까지 시간지연 τ_k가 필요하다면, 이 트랜지션의 시간지연 τ_k는 상수나 랜덤변수 중 하나가 된다. 이 랜덤변수의 모멘트 발생함수를 M(s)로 놓자. 그러면, 전달함수 $W_k(s)$는 $P(t_k)$와 M(s)의 곱으로 나타낼 수 있다.

$$W_k(s) = P(t_k)\, M(s) \qquad (3\text{-}1)$$

이 두개의 매개변수 $P(t_k)$와 τ_k를 가진 트랜지션 t_k는 다음의 그림 3-1에서와 같이 $W_k(s)$를 가진 t_k로 표현될 수 있다.

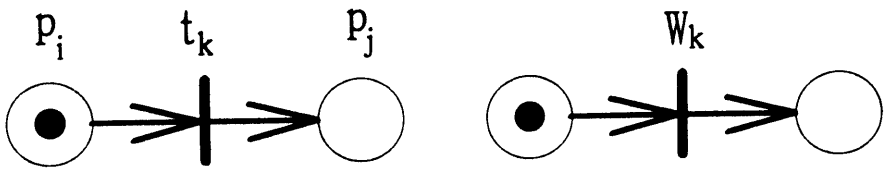


그림 3-1. 전달함수의 정의

그러므로, 각 상태별 전달함수는 도달가능 그래프에서 얻을 수 있다. 각 상태 전환시의 전달함수를 구하는 데 메이슨 정리를 이용할 수 있다. 확장 확률 페트리네트(ESPN: Expanded Stochastic Petri-Net)에서 주어진 플레이스 i와 j사이 트랜지션의 전달함수를 구하는 데 이용 가능하다.

먼저, 메이슨의 정리을 살펴보면 다음과 같다.

$$M = \sum_{k=1}^{N} \frac{M_k \Delta_k}{\Delta} \qquad (3\text{-}2)$$

- N : 전향경로의 총 수
- M_k: k번째 전향경로의 이득
- $\Delta = 1 - \sum_m P_{m1} + \sum_m P_{m2} - \sum_m P_{m3} + \ldots$
- P_{mr}: r개의 비접촉 루프의 가능한 m번째 조합의 이득 곱
- Δ = 1 - (모든 각각의 루프 이득의 합)
  +(2개의 비접촉 루프의 가능한 모든 조합의 이득곱의 합)
  -(3개의 ...) + ...
- Δ_k : k번째 전향경로와 접촉하지 않는 신호 흐름선도 부분에대한 Δ값

플레이스 i와 j사이 트랜지션의 전달함수 $W_{ij}(s)$는 다음과 같이 표현이 가능하다.

$$W_{ij}(s) = \frac{1}{\Delta}\sum_{k} W_k(s)\,\Delta_k \qquad (3\text{-}3)$$

$$\Delta = 1 - \sum_{m}\sum_{i}(-1)^m W_i(m,s) \qquad (3\text{-}4)$$

이 식 3-3은 식 3-2에서 M 대신 $W_{ij}(s)$를, M_k 대신 $W_k(s)$를 넣은 것이다. M_k는 k번째 전향경로의 이득이므로, 트랜지션 t_k의 전달함수와 같다. 메이슨 정리를 적용시키기 위해 폐루프로 만들어 Δ 값이 0이 되도록 그림 3-2와 같이 놓자. 여기서 $W_E(s)$는 소스(source)에서 싱크(sink)로의 전달함수이고, $W_C(s)$는 추가된 전달함수이다.

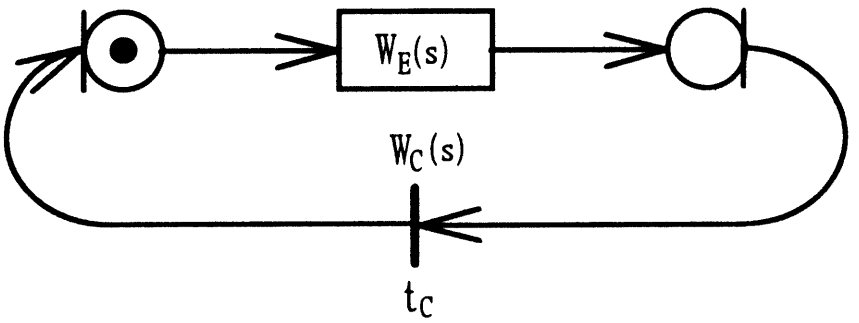


그림 3-2. $W_C(s)$를 가진 t_c가 추가된 폐루프 다이어그램

여기서 Δ는

$$\Delta = 1 - W_E(s)\cdot W_C(s) = 0 \qquad (3\text{-}5)$$

가 된다. 여기서 추가된 폐루프에서의 전달함수 $W_C(s)$를 얻을 수 있다.

$$W_C(s) = \frac{1}{W_E(s)} \qquad (3\text{-}6)$$

따라서, 식 3-1로부터 다음의 식을 얻을 수 있다.

$$W_E(s)|_{s=0} = P_{ij} \tag{3-7}$$

$$(\because\ M_E(0)=1,\ W_E(0)=P_{ij}\cdot M_E(0))$$

모멘트 발생함수 M(s)는

$$M(s) = \frac{W_E(s)}{W_E(0)} \tag{3-8}$$

이 된다.

4. 페트리네트(Petri-Net)

4.1 페트리네트의 정의

페트리네트의 종류에는 일반적인 상페트리네트(Ordinary PN), 플레이스나 트랜지션에 시간을 삽입한 시간 페트리네트(Timed PN), 일반화된 페트리네트(generalized PN), 확률 페트리네트(Stochastic PN), 다양한 정보 흐름을 용이하게 표현하는 착색 페트리네트(Colored PN), 일반 페트리네트의 표현력을 보강한 확장 페트리네트(Extended PN)등이 있다.

페트리네트는 다음과 같이 5개의 요소로 구성되어 있다.

$$N = \{P, T, I, O, M_0\}$$

$P = \{ P_1, P_2, \cdots, P_m \}$: 유한한 플레이스의 집합으로 원으로 표시된다.

$T = \{ t_1, t_2, \cdots, t_n \}$: 유한한 트랜지션의 집합으로 막대로 표시된다.

$I : P \times T \rightarrow \{ 0, 1 \}$: 플레이스에서 트랜지션으로 가는 입력함수로,
플레이스 P에서 트랜지션 T로 향하는 화살표로 표시된다.

$O : P \times T \rightarrow \{ 0, 1 \}$: 트랜지션에서 플레이스로 가는 출력함수로,
트랜지션 T에서 플레이스 P로 향하는 화살표로 표시된다.

M: 플레이스의 토큰의 수, 초기 마킹은 M_0

플레이스는 자원의 사용가능성 또는 공정상태등 조건(condition)을 나타낸다. 트랜지션은 공정의 시작과 끝을 나타내는 사건(event)을 의미한다. 시스템의 상태변화는 토큰의 위치로 나타내어지므로, 토큰이 이동하는 것은 시스템이 다음 상태로 전이함을 나타내게 된다.

페트리네트는 일정한 실행규칙에 따라 토큰이 이동하게 된다. 이 실행규칙에는 활성화 규칙(enabling rule)과 점화규칙(firing rule)이 있다. 활성화 규칙은 트랜지션의 모든 입력 플레이스에 토큰이 있을 때, 트랜지션은 활성화(enabled)되고 점화(fire)할 수 있다는 것이다. 점화규칙은 활성화된 트랜지션이 점화하면, 모든 입력 플레이스에서 하나의 토큰이 제거되고, 모든 출력 플레이스에 하나의 토큰이 더해진다는 것이다.

페트리네트는 다음과 같은 것들을 모델링 할 수 있다.

- 선행관계(precedence of events) : 순차적 실행
- 상충(conflict) : 동시에 두개 혹은 그 이상의 프로세스가 공통의 자원(resource)을 요구할 때 발생
- 병렬처리(concurrency or parallelism)
- 동기(synchronization)
- 합병(merging)
- 혼란(confusion) : 일관성과 상충이 동시에 존재
- 우선순위(priorities) : 금지아크(Inhibitor arc)를 사용 특정한 프로세스에 우선순위를 할당한다.

이들을 도식화하면 다음의 그림 4-1과 같다.

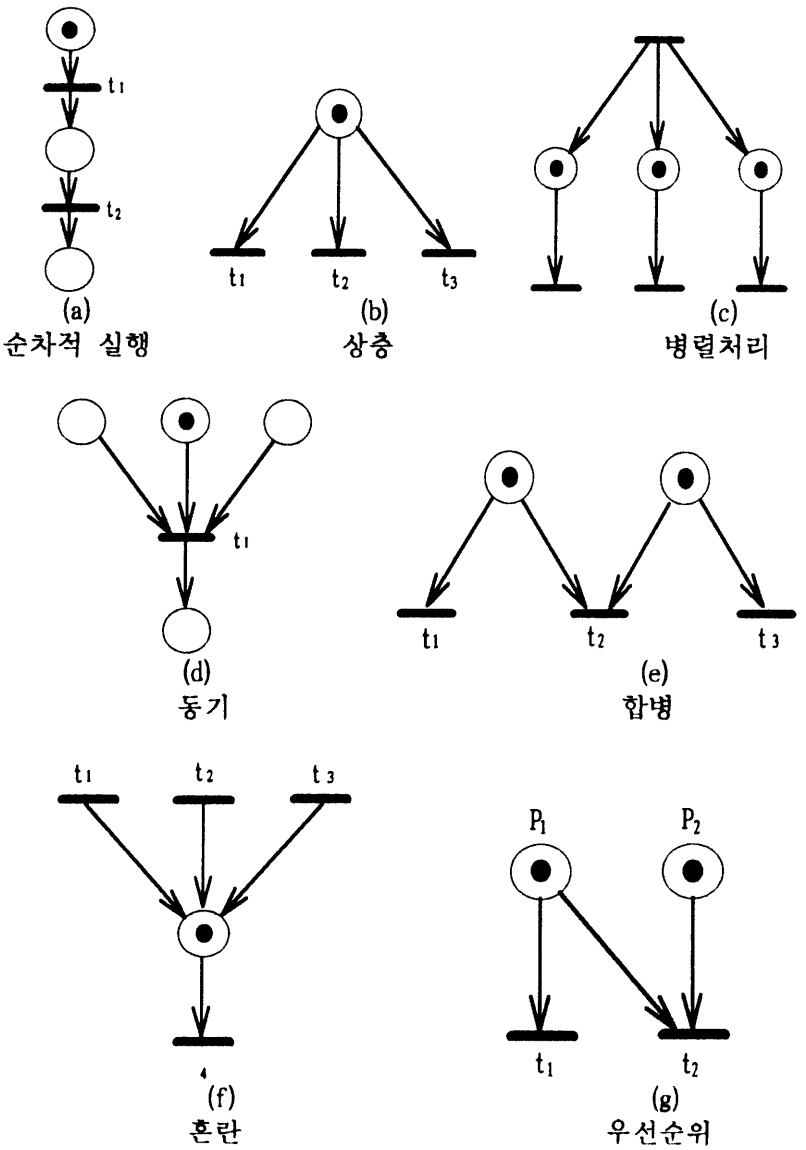


그림 4-1. 페트리네트를 이용한 시스템 모델링 예

4.2 페트리네트의 성질

1) 도달가능성(Reachability)

도달가능성은 어떤 시스템의 동적 거동을 해석하는 데에 가장 기본적인 성질로, 페트리네트 모델에서 상태를 초기조건에서 M_n에 도달하도록 하는 점화순서가 존재한다면 M_n은 M_0로 부터 도달가능하다(reachable)고 한다. 이것은 실제 시스템이 초기상태로부터 일련의 작업을 수행하여 현재 어떤 상태에 있는가를 나타내는 것이다. 활성화된 트랜지션 t_i가 점화되면 상태가 전이되어 새로운 마킹(marking) M'로 변화하는 데 그 변환을 수식으로 표현하면 다음과 같다.

$$M \xrightarrow{t_i} M' \quad : \quad M'(P) = M(P) + O(P, t_i) - I(P, t_i)$$

2) 유한 갯수(Boundedness)

$$M(P_i) \leq k \qquad \text{for any} \quad M \in R[M_0]$$

한 플레이스 P_i의 토큰 갯수가 양의 정수 k를 넘지 않는다면 k-bounded라 한다. 또한, k가 1이면 페트리네트는 안전하다고 한다. 페트리네트는 시스템의 레지스터(register), 버퍼(buffer)등을 나타내는 데, 실제시스템에서 페트리네트의 유한갯수 또는 안전성는 시스템내에 오버플로우가 존재하지 않음을 의미한다.

3) 생동성(Liveness)

이것은 초기 마킹 M_0로부터 어떤 마킹 M에 도달하였을 지라도 항상 활성 가능한 트랜지션이 있음을 의미한다. 시스템 교착상태(deadlock)는 어떤 마킹에서 어떤 트랜지션도 활성 가능하지 않게 되었을 때를 의미한다. 따라서, 페트리네

트에서의 생동성(Liveness)은 시스템내에 교착상태(deadlock)가 존재하지 않음을 의미한다.

4) 귀환성(Reversibility)

귀환성은 초기상태 M_0에서 출발하여 $M = M_0$가 되는 점화순서가 존재함을 나타내며, 시스템을 다시 초기 마킹 상태를 만들수 있음을 의미한다. 이것은 이상상태 복구(error recovery)가 가능하다는 것을 의미한다.

5) 일관성(Consistency)

초기 마킹 M_0에서 어떤 점화순서에 의하여 다시 M_0로 되돌아 갈 수 있다면 페트리네트는 일관적이다. 이는 시스템이 반복적인 작업(Cyclic activity)을 수행할 수 있음을 보여준다.

4.3. 페트리네트의 축소

페트리네트의 구조들은 다음과 같이 하나의 트랜지션으로 축소될 수 있다. 이 축소 규칙은 페트리네트의 어떤 클래스들을 간략화하는 데 사용되는 것으로, 각 바운드된 페트리네트들은 유한 상태 머신으로 변환할 수 있다. 그리고, 플레이스의 입/출력 천이는 XOR(exclusive-OR) 로직으로 구성할 수 있다. 이것은 페트리네트의 상태 머신이 XOR GERT(Graphical Evaluation and Review Technique) 네트웍 모델과 같다는 것을 의미한다. 그러므로, 모든 XOR GERT 네트웍에 대한 정리를 상태 머신 페트리네트에 적용시킬 수 있다.

플레이스를 흐름 그래프에서의 노드로 놓고, 트랜지션에 귀속되는 전달함수는 이득으로 놓는다면, 각 상태 머신들을 간략화 시킬 수 있다. 이 간략화는 메이슨의 정리에서의 이득을 구하는 방법과 동일하다. 그러므로, 이 간략화에 메이슨의 정리를 적용시킬 수 있다. 본 논문에서는 이러한 방법을 이용하여 제시한 모델을 간략화 시키고, 전체 시스템에 적용시킨다.

5. 필드버스 데이타링크층의 페트리네트 모델

토큰 사용 가능시간에 대한 매개변수를 가지지 않는 CiT(Circulated Token)가 이동하며, 서비스를 수행하는 것에 대해 다음의 그림 5-1과 같이 나타내었다. 토큰이 현 스테이션에 도착한 경우, 도착한 토큰의 사용 가능 여부를 판별한다. 먼저 버퍼에 저장된 메시지가 있는 지 조사한다. 저장된 메시지가 없는 경우에는 토큰을 사용하지 않고 다음 스테이션으로 넘겨야 한다. 저장된 메시지가 있는 경우, 제일 먼저 저장된 메시지에 대한 처리 시간을 미리 할당된 토큰 사용 가능 시간(THT: Token Holding Time)과 비교한다. 현재 처리할 메시지에 대한 시간이 토큰 사용 가능 시간보다 클 경우, 토큰을 다음 스테이션으로 넘긴다. 이때 메시지가 긴급을 요하는 경우(Urgent)에는 DeT(Delegated Token)을 요구하여 서비스하고, 그렇지 않은 경우는 다음에 다시 CiT가 도착할 때까지 버퍼에 저장된 상태로 있는다. 토큰 사용 가능시간이 메시지 처리 시간보다 클 경우만 메시지를 처리하며, 메시지를 처리하는 동안 메시지 처리속도와 같은 비율로 토큰 사용 가능시간을 감소시킨다. 한개의 메시지를 처리한 후 다시 버퍼에 저장된 메시지가 있는 지 조사한다. 저장된 메시지가 있는지 없는지에 따라 앞서와 같은 과정을 통해 토큰을 다음 스테이션으로 넘기거나 서비스를 수행한다. 이때 사용되는 토큰 사용 가능시간은 메시지 처리시 감소된 값이 사용된다.

이러한 동작을 함에 있어 각 시간들에 대해 다음과 같이 가정하였다. 평균 메시지 처리 시간은 $1/\mu$의 지수분포를 갖는다. 그리고, 메시지 발생율은 발생율 λ의 포아송 분포를 가지며, 발생된 메시지는 k개의 저장량을 가진 선입선출(First Come First Service) 형태의 버퍼에 저장된다고 가정한다.

이와 같은 동작을 페트리네트를 이용하여 다음과 같은 형태로 모델을 구성하였다. 먼저, 메시지 발생은 항상 가능해야하므로 이 부분(플레이스 P_6)에 토큰을 한개 부여하였다. 그러므로, 서비스를 하려면 두개의 토큰이 필요하게 된다. 이 메시지 발생 토큰은 트랜지션 T_2를 점화시키고 다시 플레이스 P_6로 돌아온다. 트랜지션 T_2 점화시 발생된 메시지는 저장량이 k개인 플레이스 P_5에 저장된다. 또한, 플레이스 P_5는 앞서 가정한 것과 같이 선입선출의 형태를 갖는다. 이 플레이스 P_5가 모두 찼을 경우(저장

된 메시지의 수가 버퍼 한계와 동일한 k개인 경우) 트랜지션 T_2의 점화를 금지시킨다.

현 스테이션에 토큰이 도착하면 플레이스 P_1이 토큰을 가지게 된다. 트랜지션 T_1은 시간지연이 없는 트랜지션이다. 이 트랜지션 T_1은 서비스 수행 가능 상태로 만듦과 동시에 서비스 수행 가능 조건을 조사하는 기능을 갖는다. 플레이스 P_2는 서비스 수행 가능 상태로, 플레이스 P_5와 P_4와 함께 트랜지션 T_3가 점화 가능하도록 한다. 트랜지션 T_3는 서비스 수행, 즉 메시지 처리 시간동안 지연되는 트랜지션이다. 단, 이 트랜지션은 플레이스 P_2와 P_5에 토큰이 있고, 플레이스 P_4에는 토큰이 없는 경우에만 점화가능하다. 여기서 플레이스 P_4는 서비스 수행 불가능 상태일 때 토큰을 가지므로, 토큰이 없는 경우에만 트랜지션 T_3를 활성화 시킨다.

트랜지션 T_1이 점화되었을 때 서비스 수행 가능 여부를 판별하는 상태인 플레이스 P_3가 토큰을 갖게되어, 서비스 수행 가능 여부를 조사한다. 플레이스 P_5에 저장된 메시지가 없는 경우, 트랜지션 T_4가 점화되어 플레이스 P_4가 토큰을 갖게된다. 플레이스 P_5에 저장된 메시지가 있어서 서비스를 수행해야 할 경우, 트랜지션 T_5의 점화 여부를 조사한다. 여기서 트랜지션 T_5는 토큰 사용 가능 시간 동안 지연시키는 트랜지션이다. CiT 동작에 따른 서비스 수행에는 한개의 메시지 처리시마다 현재의 토큰 사용 가능 시간과 비교해야 하지만, 변수값을 갖기 때문에 모델로써의 구성도 어렵고 수학적으로 그 확률을 구하기 어렵다. 그래서, 본 논문에서는 이 부분을 모델 오차로써 처리하여, 할당된 토큰 사용 가능시간동안 기다렸다가 이 토큰 사용가능 시간이 만료될 때 점화되도록 구성하였다.

더이상 현 스테이션의 서비스가 더이상 수행될 수 없거나(트랜지션 T_5가 점화:THT 만료) 수행될 필요가 없을 경우(플레이스 P_5에 저장된 메시지가 없는 경우), 플레이스 P_4가 토큰을 가지게 된다. 플레이스 P_4가 토큰을 가질 경우에는 트랜지션 T_3가 점화될 수 없다. 트랜지션 T_6는 토큰을 다음 스테이션으로 넘기기 위해 플레이스 P_6와 P_5의 토큰을 제외한 나머지 토큰, 즉 플레이스 P_2와 P_4의 토큰을 제거시키는 역할을 한다. 그리고, 플레이스 P_7은 토큰을 다음 스테이션으로 넘긴 상태가 된다. 마지막으로 트랜지션 T_7은 토큰이 다시 들어올 때까지 기다리는 것을 의미한다. 이 트랜지션 T_7이 다른 스테이션들이라고 볼 수 있다. 그리고, 이 트랜지션 T_7이 현 스테이션으로의 토큰 도착을 가리키는 플레이스 P_1에 연결된다고 볼 수 있다.

이상과 같이 토큰을 이용하여 서비스를 수행하는 스테이션들은 모두 같은 형태로 구성된다. 다음의 표 5-1은 그림 5-1에 대한 각 플레이스와 트랜지션 설명을 나타낸 것이다.

표 5-1. 필드버스 데이타링크층 페트리네트 모델에서의 플레이스와 트랜지션

플레이스	
P_1	현재의 스테이션이 토큰을 가짐
P_2	서비스 수행중
P_3	서비스 종료 조건 검사
P_4	서비스 종료
P_5	서비스 요구를 저장(k개의 저장량을 가짐)
P_6	서비스 요구 발생 가능 상태
P_7	토큰을 다음 스테이션으로 넘김

트랜지션	
T_1	현 스테이션에 토큰이 도달, 서비스 수행 시작
T_2	서비스 요구 발생(발생율의 λ의 포아송 분포)
T_3	서비스 수행 (평균 수행 시간은 1/μ의 지수분포)
T_4	서비스 요구가 없을 경우 서비스 종료
T_5	토큰 사용 가능 시간 동안 지연
T_6	서비스 종료
T_7	토큰이 다시 들어올 때까지 지연

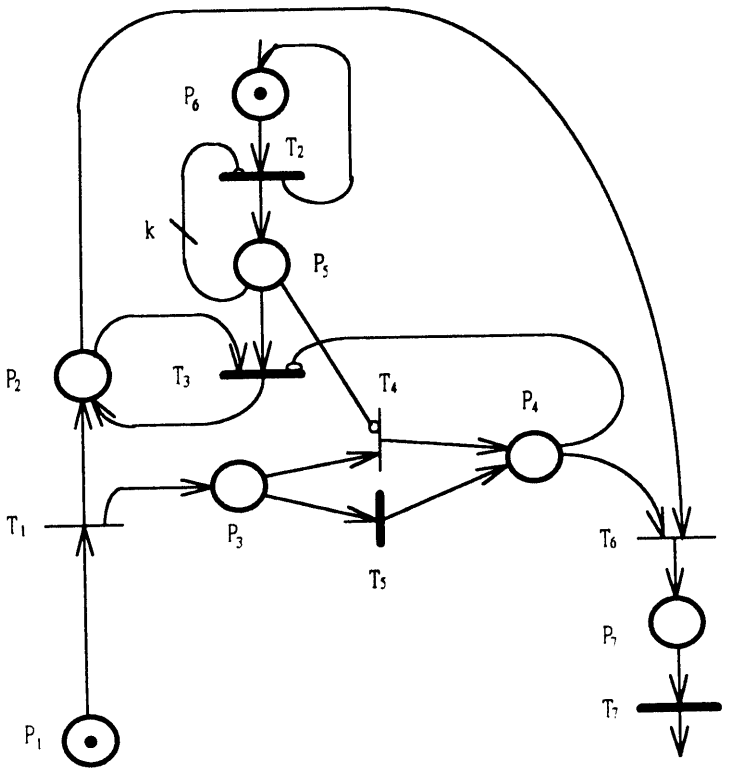


그림 5-1. 펌프제스 데이타링크층의 페트리네트 모델

6. 평균 토큰 회전 시간과 평균 서비스 시간

6.1. 메시지 저장량이 임의의 k개인 경우

플레이스 P_5를 임의의 k개로 제한을 둔 경우, 다음과 같은 도달 가능 그래프를 얻을 수 있다.

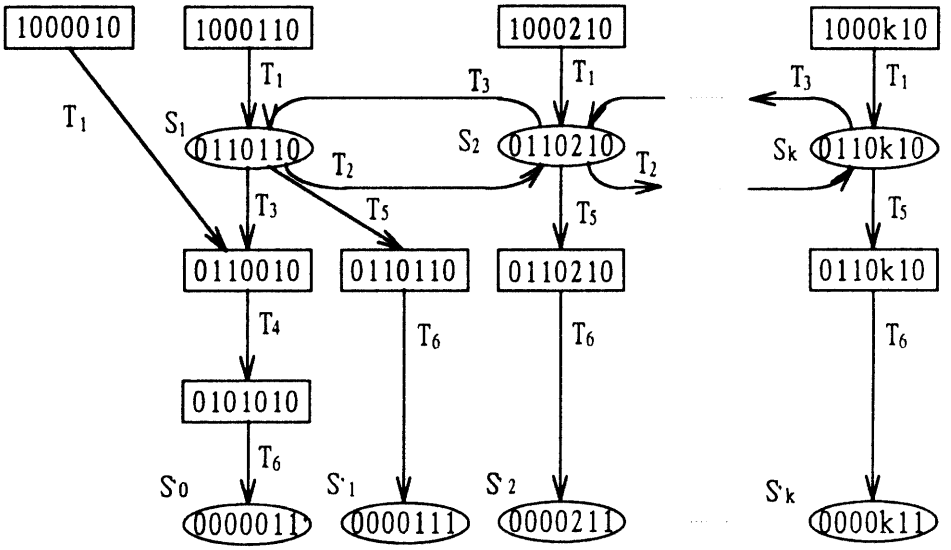
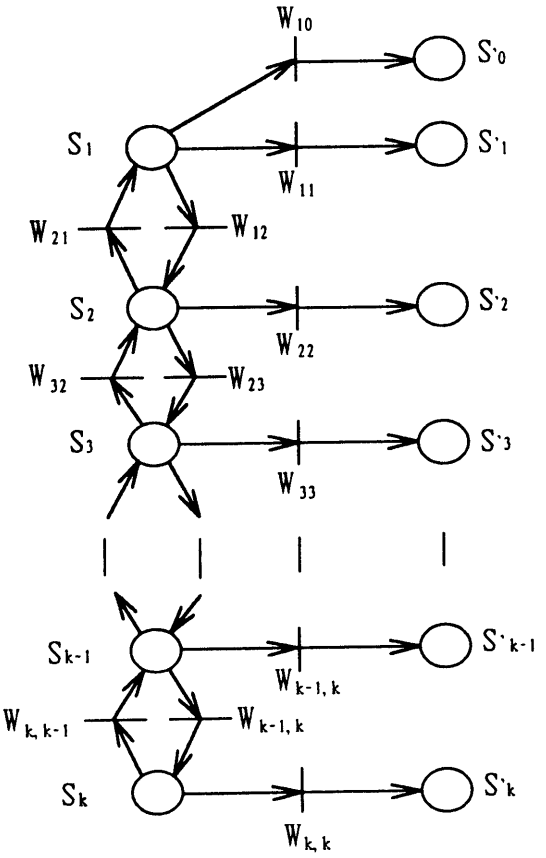


그림 6-1. 메시지 저장량이 k개인 경우의 도달 가능 그래프

여기서 사각형의 상태들은 시간지연이 없는(vanishing) 상태이고, 타원형의 상태들

은 시간지연을 가지는(tangible) 상태이다. 이 그림 6-1의 도달 가능 그래프에서 시간지연이 없는 상태를 제거하고, 시간지연이 있는 부분만 취하여 상태도를 얻으면 그림 6-2와 같다.



S_k : 토큰을 가지고 있으며, 저장된 메시지가 k개 있는 경우

S'_k : 토큰을 다음 스테이션으로 전달해야 하고,
저장된 메시지가 k개 있는 경우

W_{ij} : 상태 S_i에서 S_j로 변환될 경우의 전달 함수
단, W_{10}은 S_1에서 S'_0로 변환될 경우의 전달함수이다.

W_{11} : 상태 S_1에서 S'_1로 변환될 경우의 전달 함수

그림 6-2. 메시지 저장량이 임의의 k개인 경우의 상태도

같은 시간에 발생 가능한 트랜지션들을 $\{t_1, t_2, \ldots, t_d\}$, $d > 1$ 로 놓고, X_i를 $t_i(i=1,2,\ldots,d)$에서의 랜덤시간지연이라 놓는다. 그러면, t_i의 전단 함수 $W_i(s)$는 다음과 같이 나타낼 수 있다.

$$W_i(s) = P(t_i) \cdot M(t_i, s) \tag{6-1}$$

$$P(t_i) = P\{X_i \leq X_j,\ j \neq i\}$$

여기서 $M(t_i, s)$ 는 다음과 같이 사건 발생 중 시간지연이 작은 쪽에 대한 모멘트 발생 함수이다. 왜냐하면, 트랜지션은 시간지연이 작은 쪽으로 먼저 점화되기 때문이다.

$$M(t_i, s) = \text{Min}\{X_j,\ j = 1, 2, \cdots, d\} \tag{6-2}$$

이러한 방법으로 구한 모멘트 발생함수를 이용하여 다음과 같은 순서에 의해 평균 서비스 시간 T_S와 평균 토큰 회전시간 T_R을 구한다. 식의 유도 및 계산은 각 과정별로 진행해가며 설명한다. 그러면, 다음의 표 6-1.에서와 같은 순서에의해 평균 서비스시간과 평균 토큰 회전시간들을 구해보겠다.

표 6-1. 한 스테이션에서의 평균 서비스 시간과 평균 토큰 회전시간을 구하는 알고리즘

평균 서비스 시간 T_S와 평균 토큰 회전시간 T_R을 구하는 알고리즘
1. 토큰이 들어왔을 때의 확률 P_A를 구한다.
2. 상태 S_i에서 발생 가능한 각각의 트랜지션에 대한 전달 함수 $W_{ij}(s)$를 구한다.
3. 과정 2에서 구한 전달 함수들을 이용하여, 상태 S_i에서의 전달 함수 $W_{Ei}(s)$를 구한다.
4. 전달 함수 $W_{Ei}(s)$에서 모멘트 발생 함수 $M_{Ei}(s)$를 구한다. $M_{Ei}(s) = \frac{W_{Ei}(s)}{W_{Ei}(0)}$, 단, $M_{E0}(0)=1$
5. 모멘트 발생 함수 $M_{Ei}(s)$와 토큰 도착시 확률 P_A를 곱하여, 한 스테이션의 전달 함수 $W_{LM}(s)$를 구한다.
6. 한 스테이션의 전달함수 $W_{LM}(s)$를 이용하여, 한 스테이션에서의 모멘트 발생 함수 $M_{LM}(s)$를 구한다.
7. 모멘트 발생 함수 $M_{LM}(s)$를 이용하여, 평균 서비스 시간 T_S와 평균 토큰 회전시간 T_R을 구한다.

STEP 1. 도착 확률 P_A

그림 6-2의 상태도에서 토큰 도착시 상태들에대한 도착 확률 P_A은 다음과 같이 나타낼 수 있다.

$$P_A = [\ P_A(0)\ \ P_A(1)\ \ \cdots\ \ P_A(K)\ ]^T$$

$$P_A(i) = P(X_t = i)$$

$$P_D(i) = P(X_{t'} = i)$$

$a_{ij} = P(X_t = i | X_{t'} = j)$

$d_{ij} = P(X_{t'} = i | X_t = j)$

$X_t = i$: 시간 t에서의 메시지 수가 i인 경우

$X_{t'} = j$: 시간 t'에서의 메시지 수가 j인 경우

라 놓으면, 다음과 같이 도착 확률 P_A를 구할 수 있다.

$$P_A = A \cdot D \cdot P_A \qquad (6\text{-}3)$$

where,

$$A = \begin{bmatrix} a_{00} & 0 & 0 & \cdots & 0 & 0 \\ a_{10} & a_{11} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ a_{k0} & a_{k1} & a_{k2} & \cdots & a_{k,k-1} & 1 \end{bmatrix}$$

$$a_{ij} = \begin{cases} \dfrac{(\lambda t)^{(i-j)}}{(i-j)!} e^{-\lambda t} & (i \neq k) \\ 1 - \sum_{k=0}^{k=i-1} \dfrac{(\lambda t)^{(k-j)}}{(k-j)!} e^{-\lambda t} & (i = k) \end{cases}$$

* t : 토큰이 떠나서 다시 자신의 스테이션에 도달할 때까지의 시간

$$D = \begin{bmatrix} 1 & d_{01} & \cdots & d_{0k} \\ 0 & d_{11} & \cdots & d_{1k} \\ & \vdots & \vdots & \\ 0 & d_{k1} & \cdots & d_{k,k} \end{bmatrix}$$

$$d_{ij} = W_{ij}(0)$$

$W_{ij}(s)$: 상태 i에서 상태 j로 변할 때의 전달함수

식 6-3에 안정상태(Steady state)라는 가정하에서 $\sum_{i}^{all} P_A(i) = 1$을 대입하면, P_A를 구할 수 있게 된다.

STEP 2. 상태 S_i에서 상태 S_j로의 전달함수 $W_{ij}(s)$

앞서 나타낸 그림 6-1에서 시간지연을 가지는 트랜지션들에는 T_2, T_3, T_5가 있다. 각 트랜지션들의 확률 밀도함수들은 다음과 같다.

트랜지션 T_2 점화시의 확률 밀도함수 : $\lambda \cdot e^{-\lambda t}$ (6-4)

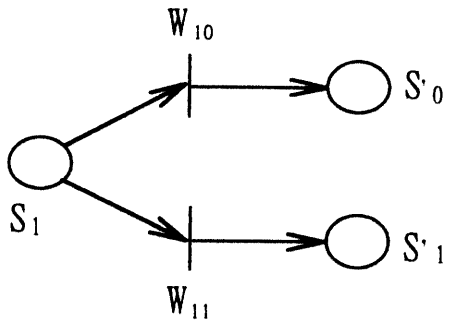
트랜지션 T_3 점화시의 확률 밀도함수 : $\mu \cdot e^{-\mu t}$ (6-5)

트랜지션 T_5 점화시의 확률 밀도함수 : $\delta(t - T_H)$ (6-6)

그림 6-2에서 각 상태별로 변환이 가능한 상태들에 대한 확률 밀도함수들에의해 그 최소치를 구하고, 모멘트 발생함수 $M_i(s)$를 구한다. 구한 모멘트 발생함수에 트랜지션 점화확률을 곱하면, 그것이 상태 S_i에서 상태 S_j로의 전달함수 $W_{ij}(s)$가 된다.

STEP 3. 상태 S_i에서의 전체 전달함수

STEP 2에서 구한 상태 S_i에서의 변환별 전달함수들을 메이슨의 정리를 이용하여 하나의 전달함수로 축소하면, 그것이 상태 S_i에서의 전체 전달함수가 된다. 예를들어, 다음과 같은 형태의 상태도가 주어졌을 경우, S_i에서의 전체 전달함수 $W1(s)$는 각 전달함수의 합과 같다.



$$W_1(s) = W_{10}(s) + W_{11}(s)$$

STEP 4. 상태 S_i에서의 모멘트 발생함수

식 3-7과 3-8에서와 같이 상태 S_i에서의 모멘트 발생함수는

$$M_{Ei}(s) = \frac{W_{Ei}(s)}{W_{Ei}(0)}, \qquad 단, \quad M_{E0}(0) = 1$$

가 된다.

STEP 5. 한 스테이션에서 서비스가 수행되는 시간에대한 전체 전달함수 $W_{LM}(s)$

위의 식 6-3을 이용하면, 토큰이 전달되는 순간의 플레이스 P5에 저장되어 있는 메시지의 갯수에 대한 확률 P_A가 구해진다. 이 확률 P_A와 모멘트 발생 함수 $M_{Ei}(s)$를 이용하여 한 스테이션의 전체 전달 함수 $W_{LM}(s)$를 구할 수 있다.

$W_{LM}(s)$: 한 스테이션에서의 전체 전달 함수 2
$M_{LM}(s)$: 한 스테이션에서의 모멘트 발생 함수

$$W_{LM}(s) = P_A^T \cdot M_E(s)$$

where,

$$M_E(s) = [\ M_{E0}(s) \ \cdots \ M_{EK}(s)\ ]^T$$

STEP 6. 한 스테이션에서의 모멘트 발생함수 $M_{LM}(s)$

STEP 4에서와 같은 방법으로 한 스테이션에서의 모멘트 발생함수를 구한다.

$$M_{LM}(s) = \frac{W_{LM}(s)}{W_{LM}(0)}$$

STEP 7. 한 스테이션에서의 평균 서비스 시간 T_S와 평균 토큰 회전시간 T_R

평균 서비스 시간 T_S은 앞서 구한 $M_{LM}(s)$을 이용2하여 다음과 같이 구할 수 있다.

$$T_S = \frac{d}{ds} M_{LM}(s)|_{s=0}$$

또한, 평균 토큰 회전 시간 T_R은 2

$$T_R = N \cdot (T_S + T_O)2$$

N : 시스템에서2 스테이션의 갯수

T_O : 오버 헤드 2시간(상수로 정해짐)

로 나타낼 수 있다. 안정상태에서의 평균값을 얻는 것이기 때문에 각 스테이션에서의 지연시간들이 일정하다고 볼 수 있다.

위에서와 같이 평균 서비스 시간 T_S와 평균 토큰 회전 시간 T_R은 이미 알고 있는 값들(스테이션의 총 수, 오버 헤드 시간, 평균 서비스 시간, 메시지 발생율, 토큰 사용 가능 시간)에의해 구해진다. 그러면, 이 알고리즘에 따라 k=1인 경우와 k=2인 경우에 대해 평균 서비스 시간과 평균 토큰 회전시간을 구해 보겠다.

6.2. k=1 인 경우

k를 1로 놓았을 경우 모델에서 얻을 수 있는 도달 가능 그래프는 다음과 같다.

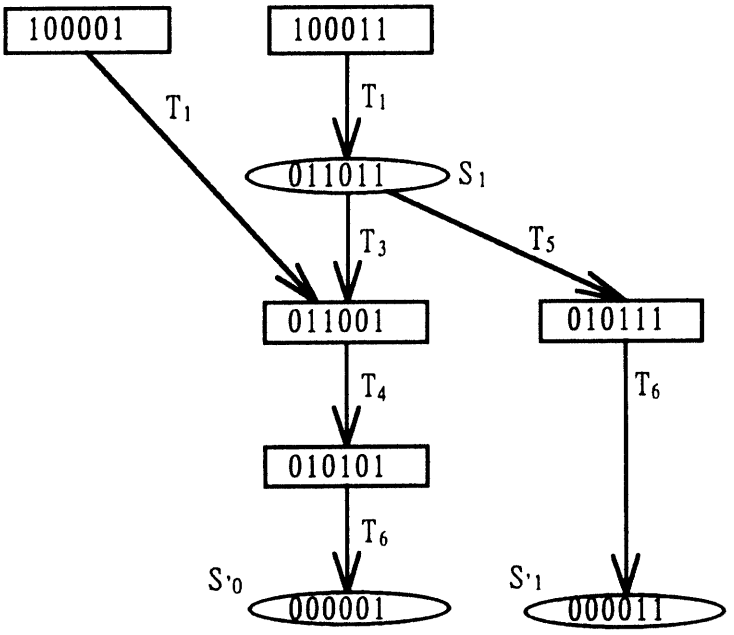


그림 6-3. k=1인 경우의 도달 가능 그래프

그림 6-3에서 얻은 도달 가능 그래프에서 시간지연이 없는 상태들을 제거한 상태도는 다음 그림 6-4와 같다.

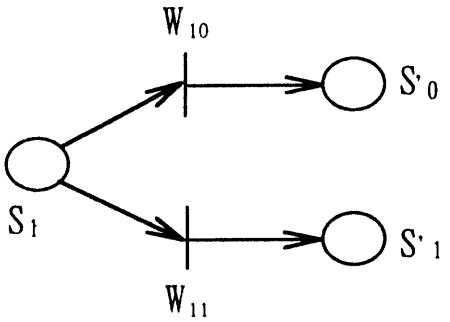


그림 6-4. k=1인 경우의 상태도

그림 6-4에서 각 상태들은 다음과 같은 상태를 의미한다.

S_1 : 토큰을 가지며, 서비스 요구가 존재하는 상태

S_0 : 서비스를 수행하여 남아 있는 서비스가 없고, 토큰을 다음 스테이션으로 넘겨야 하는 상태

S_1 : 토큰 사용 가능시간이 만료되어, 서비스를 하지 않고 토큰을 다음 스테이션으로 넘겨야 하는 상태

사용되는 변수들을 살펴보면, 다음과 같다.

λ : 서비스 요구 발생율

μ : 서비스 수행률($1/\mu$는 서비스 수행 시간)

T_H : 토큰 사용 가능 시간(THT)

T_R : 평균 토큰 회전 시간

T_S : (한 스테이션에서의) 평균 서비스 시간

T : $T_R - T_S$ (토큰을 넘긴 후 토큰을 다시 받을 때까지의 시간)

STEP 1. 도착 확률 P_A

$$P_A = \begin{bmatrix} e^{-\lambda T} & 0 \\ 1-e^{-\lambda T} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1-e^{-\mu T} \\ 0 & e^{-\mu T} \end{bmatrix} \cdot P_A$$

$$P_A(0)+P_A(1)=1$$

($P_A(i)$: 토큰 도착시 저장되어 있는 메시지의 수가 i개일 확률)

이 두 식을 이용하면, 다음과 같이 도착시의 확률을 구할 수 있다.

$$P(X_t=0) = \frac{(1-e^{-\mu T_H}) \cdot e^{-\lambda(T_R - T_S)}}{1 - e^{-\mu T_H} \cdot e^{-\lambda(T_R - T_S)}}$$

$$P(X_t=1) = \frac{1-e^{-\lambda(T_R-T_S)}}{1-e^{-\mu T_H}\cdot e^{-\lambda(T_R-T_S)}}$$

$X_t = i$: 시간 t(토큰이 도착했을 때)에서의 메시지 수가 i인 경우

$X_{t'} = j$: 시간 t'(토큰이 다음 스테이션으로 넘어갈 때)에서의 메시지 수가 j인 경우

STEP 2. 상태 i에서 상태 j로의 전달함수 $W_{IJ}(s)$

그림 6-4의 상태도에서 보면 S_1에서 발생 가능한 트랜지션은 T_3와 T_5가 있다. 트랜지션 T_3가 점화할 경우와 트랜지션 T_5가 점화할 경우의 확률 밀도함수는 식 6-5, 6-6과 같다.

트랜지션 T_3 점화시의 확률 밀도함수 : $\mu\cdot e^{-\mu t}$

트랜지션 T_5 점화시의 확률 밀도함수 : $\delta(t-T_H)$

이 두 식의 최소치는 다음과 같은 방법으로 구한다. 두 확률 밀도함수를

$$f_x(t) = \mu\cdot e^{-\mu t}, \quad f_y(t) = \delta(t-T_H)$$

로 놓자. 이 $f_x(t)$와 $f_y(t)$의 최소치는 다음과 같은 식에의해 구할 수 있다.

$$f_w(t) = f_x(t)(1-F_y(t)) + f_y(t)(1-F_x(t)) \qquad (6\text{-}7)$$

$f_w(t)$: $f_x(t)$와 $f_y(t)$의 최소치의 확률 밀도함수

$F_x(t), F_y(t)$: 각 확률 밀도함수에 대한 확률 분포함수

식 6-7에 의해 두 확률 밀도함수에 대한 최소치는

$$f_w(t) = \mu\cdot e^{-\mu t}\{1-U(t-T_H)\} + \delta(t-T_H)\cdot e^{-\mu t}$$

가 된다.

여기서 얻은 최소치의 확률 밀도함수 $f_w(t)$를 모멘트 발생함수 $M_1(s)$로 변환시키면, 다음과 같다.

$$M_1(s) = \int_0^{T_H} e^{st} \cdot f_w(t)\, dt$$
$$= \frac{\mu}{\mu - s} - \frac{s}{\mu - s} \cdot e^{(s-\mu)T_H} \qquad (6\text{-}8)$$

이 식 6-8을 그림 6-4의 상태도에 대해 적용시키면, 다음과 같이 각 트랜지션별 전달함수를 구할 수 있다.

$W_{10}(s)$: 서비스 수행 시의 전달함수(T_3가 점화)

$$W_{10}(s) = (1 - e^{-\mu T_H})\{\frac{\mu}{\mu - s} - \frac{s}{\mu - s} e^{(s-\mu)T_H}\}$$

$W_{11}(s)$: 토큰 사용가능시간 만료시의 전달함수(T_5가 점화)

$$W_{11}(s) = e^{-\mu T_H}\{\frac{\mu}{\mu - s} - \frac{s}{\mu - s} e^{(s-\mu)T_H}\}$$

STEP 3. 상태 S_1에서의 전체 전달함수

그림 6-4. 상태도에서 전체 전달함수 $W_{E1}(s)$는 다음과 같이 된다.

$$W_{E1}(s) = W_{10}(s) + W_{11}(s)$$

STEP 4. 상태 S_1에서의 모멘트 발생함수

전달함수 $W_{E1}(s)$를 이용하여 상태 S_1에서의 모멘트 발생함수 $M_{E1}(s)$를 구하면 다음

과 같다.

$$M_{E1}(s)=\frac{W_{E1}(s)}{W_{E1}(0)}=\frac{\mu}{\mu-s}-\frac{s}{\mu-s}\cdot e^{(s-\mu)T_H}$$

STEP 5. 한 스테이션에서 서비스가 수행되는 시간에 대한 전체 전달함수

$$W_{LM}(s)=[\ P_A(0)\quad P_A(1)\ ]\begin{bmatrix}1\\ M_{E1}(s)\end{bmatrix}$$

STEP 6. 한 스테이션에서의 모멘트 발생함수 $M_{LM}(s)$

$$M_{LM}(s)=\frac{W_{LM}(s)}{W_{LM}(0)}=W_{LM}(s)\qquad(\because\ W_{LM}(0)=1)$$

STEP 7. 한 스테이션에서의 평균 서비스 시간 T_S와 평균 토큰 회전 시간 T_R

$$T_S=\frac{d}{ds}M_{LM}(s)|_{s=0}$$
$$=\frac{1-e^{-\lambda(T_R-T_S)}}{1-e^{-\mu T_H}e^{-\lambda(T_R-T_S)}}\frac{1}{\mu}(1-e^{-\mu T_H}) \qquad (6\text{-}9)$$

$$T_R=N\cdot(T_S+T_O) \qquad (6\text{-}10)$$

N : 시스템에서 스테이션의 갯수

T_O : 오버헤드 시간 (상수)

위의 두 식 6-9과 6-10를 이용하여 각 평균 값들을 구할 수 있다.

6.3. k=2 인 경우

k를 2로 놓았을 경우 모델에서 얻을 수 있는 도달 가능 그래프는 다음과 같다.

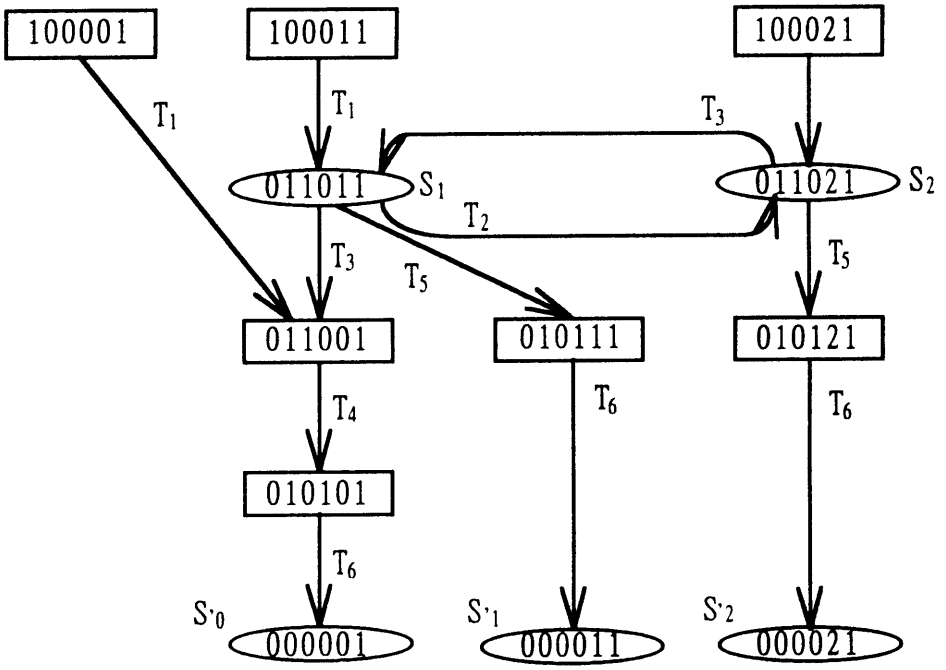


그림 6-5. k=2인 경우의 도달 가능 그래프

그림 6-3에서와 같은 형태로, 사각형은 시간지연이 없는 상태이고, 타원형은 시간지연을 갖는 상태이다. 다음의 그림 6-6은 시간지연 부분을 제거한 상태도로, 다음과 같은 상태들을 갖는다.

S_1 : 토큰을 가지며, 현재 저장된 메시지가 한개인 상태

S_2 : 토큰을 가지며, 현재 저장된 메시지가 두개인 상태

S'_0 : 남아 있는 서비스가 없고, 토큰을 다음 스테이션으로 넘겨야 하는 상태

S'_1 : 토큰 사용가능시간이 만료되어, 메시지가 한개 남아있지만 토큰을 다음 스테이션으로 넘겨야 하는 상태

S'_2 : 토큰 사용가능시간이 만료되어, 메시지가 두개 남아있지만 토큰을 다음 스테이션으로 넘겨야 하는 상태

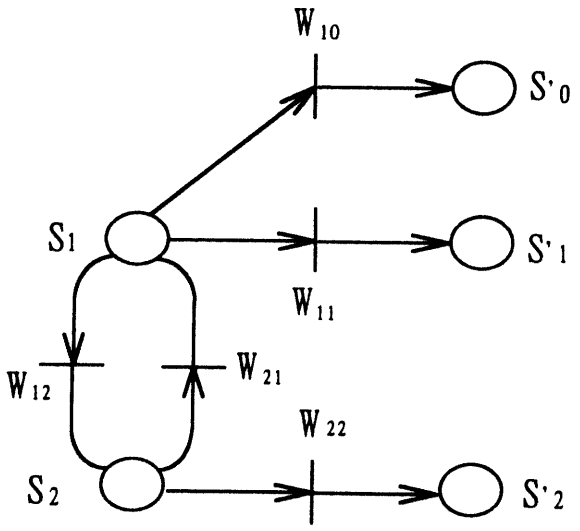


그림 6-6. k=2인 경우의 상태도

위의 상태도를 이용하여 제시한 알고리즘에 따라 k=1에서와 같이 해석과정을 살펴보겠다. 각 식에서 사용된 변수는 k=1의 경우에서 사용된 변수들과 동일하다.

λ : 서비스 요구 발생율

μ : 서비스 수행률(1/ μ는 서비스 수행 시간)

T_H : 토큰 사용 가능 시간(THT)

T_R : 평균 토큰 회전 시간

T_S : (한 스테이션에서의) 평균 서비스 시간

T : $T_R - T_S$

STEP1. 도착 확률 P_A

$$A=\begin{vmatrix} e^{-\lambda T} & 0 & 0 \\ \lambda \cdot T \cdot e^{-\lambda T} & e^{-\lambda T} & 0 \\ 1-(e^{-\lambda T}+\lambda \cdot T \cdot e^{-\lambda T}) & 1-e^{-\lambda T} & 1 \end{vmatrix}$$

$$D=\begin{vmatrix} 1 & d_{01} & d_{02} \\ 0 & d_{11} & d_{12} \\ 0 & d_{21} & d_{22} \end{vmatrix}$$

$$\begin{cases} d_{01}=\dfrac{\mu(1-e^{-(\mu+\lambda)T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \\ d_{11}=\dfrac{(\mu+\lambda)\cdot e^{-(\mu+\lambda)T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \\ d_{12}=\dfrac{\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot e^{-\mu T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \\ d_{20}=\dfrac{\mu\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \end{cases}$$

$$d_{21} = \frac{(\mu+\lambda)\cdot e^{-(\mu+\lambda)T_H}\cdot(1-e^{-\mu T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})}$$

$$d_{22} = \frac{(\mu+\lambda)\cdot e^{-\mu T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})}$$

$$P_A = A\cdot D\cdot P_A$$

$$P_A(0)+P_A(1)+P_A(2)=1$$

($P_A(i)$: 토큰 도착시 저장되어 있는 메시지의 수가 i개일 확률)

이 두 식을 이용하면, 토큰 도착시 상태에 대한 확률 P_A를 구할 수 있다.

STEP 2. 상태 S_I에서 상태 S_J로의 전달함수 $W_{IJ}(s)$

그림 6-6에서 각 상태들은 앞서 그림 6-4에서의 상태와 동일하다. k=1인 경우에는 모멘트 발생함수가 한개만이 존재했으나, 여기서는 모멘트 발생함수는 두가지 존재하게 된다. 그림 6-6에서와 같이 S_2에서의 모멘트 발생함수는 k=1인 경우와 같지만, S_1에서의 모멘트 발생함수는 다르다. 여기에는 메시지 발생이 더 추가되기 때문에 참조해야할 확률 밀도함수가 한가지 더 추가된다. 그러나, k=2이상인 다른 경우들에서는 k=2의 경우와 같이 두개의 모멘트 발생함수로 해석이 가능하다.

그림 6-6의 상태도에서 보면 S_2에서 발생 가능한 트랜지션은 그림 6-4의 S_1과 동일하다. 그러나, 그림 6-6에서의 상태 S_1에는 T_3와 T_5, 그리고 T_2가 연결되어 있다. 그러므로, 상태 S_1에서는 트랜지션 T_2가 점화할 확률이 더 첨가되어야 한다. 트랜지션 T_3가 점화할 경우와 트랜지션 T_5가 점화할 경우, 그리고 트랜지션 T_2가 점화할 경우의 확률 밀도함수들은 식 6-4, 6-5, 6-6에서와 같다.

트랜지션 T_2 점화시의 확률 밀도함수 : $\lambda\cdot e^{-\lambda t}$

트랜지션 T_3 점화시의 확률 밀도함수 : $\mu \cdot e^{-\mu t}$

트랜지션 T_5 점화시의 확률 밀도함수 : $\delta(t-T_H)$

상태 S_2에서의 모멘트 발생함수는 식 6-7과 같다. 그러므로, 상태 S_2에서의 모멘트 발생함수 $M_2(s)$는 다음과 같다.

$$M_2(s)=\frac{\mu}{\mu-s}-\frac{s}{\mu-s}\cdot e^{(s-\mu)T_H} \qquad (6\text{-}11)$$

위의 세가지 확률 밀도함수의 최소치는 다음과 같은 방법으로 구한다. 각 확률 밀도함수들을

$$f_x(t)=\mu\cdot e^{-\mu t},\quad f_y(t)=\lambda\cdot e^{-\lambda t},\quad f_z(t)=\delta(t-T_H)$$

로 놓자. 먼저 $f_x(t)$와 $f_y(t)$의 최소치를 구하고, 그 최소치의 확률 밀도함수 $f_w(t)$와 $f_z(t)$의 최소치를 구한다.

$$\begin{aligned} f_w(t) &= f_x(t)(1-F_y(t)) + f_y(t)(1-F_x(t)) \\ &= (\lambda+\mu)\cdot e^{-(\mu+\lambda)t} \end{aligned}$$

$f_w(t)$: $f_x(t)$와 $f_y(t)$의 최소치의 확률 밀도함수

$F_x(t), F_y(t)$: 각 확률 밀도함수에 대한 확률 분포함수

여기서 얻은 최소치의 확률 밀도함수 $f_w(t)$와 $f_z(t)$의 최소치는 위와 동일한 식에의해 다음과 같이 구할 수 있다.

$$f_v(t) = f_w(t)(1-F_z(t)) + f_z(t)(1-F_w(t))$$

$f_v(t)$: $f_w(t)$와 $f_z(t)$의 최소치의 확률 밀도함수

$F_w(t), F_z(t)$: 각 확률 밀도함수에 대한 확률 분포함수

대입하여 $f_v(t)$를 구하면 다음과 같다.

$$f_v(t) = (\mu+\lambda)e^{-(\mu+\lambda)t}\{1-U(t-T_H)\} + \delta(t-T_H)e^{-(\mu+\lambda)t}$$

이 $f_v(t)$를 이용하여, 모멘트 발생함수 $M_1(s)$로 변환시키면 다음과 같다.

$$\begin{aligned} M_1(s) &= \int_0^{T_H} e^{st} f_v(t)\,dt \\ &= \frac{\mu+\lambda}{\mu+\lambda-s} - \frac{s}{\mu+\lambda-s} e^{-(\mu+\lambda)T_H} e^{sT_H} \end{aligned} \qquad (6\text{-}12)$$

여기에 각 상태 변환 확률들을 곱하면, 상태 S_1에서의 트랜지션별 전달함수 $W_{1J}(s)$를 구할 수 있다.

$$W_{10}(s) = \frac{\mu(1-e^{-(\mu+\lambda)T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_1(s)$$

$$W_{11}(s) = \frac{(\mu+\lambda)\cdot e^{-(\mu+\lambda)T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_1(s)$$

$$W_{12}(s) = \frac{\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot e^{-\mu T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_1(s)$$

마찬가지의 방법으로 상태 S_2에서의 트랜지션별 전달함수 $W_{2J}(s)$를 다음과 같이 구할 수 있다.

$$W_{20}(s) = \frac{\mu\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_2(s)$$

$$W_{21}(s) = \frac{(\mu+\lambda)\cdot e^{-(\mu+\lambda)T_H}\cdot(1-e^{-\mu T_H})}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_2(s)$$

$$W_{22}(s) = \frac{(\mu+\lambda)\cdot e^{-\mu T_H}}{\mu+\lambda-\lambda\cdot(1-e^{-(\mu+\lambda)T_H})\cdot(1-e^{-\mu T_H})} \cdot M_2(s)$$

STEP 3. 상태 S_i에서의 전체 전달함수 (i=1, 2)

상태 S_1에서의 전체 전달함수 $W_{E1}(s)$는

$$W_{E1}(s) = W_{10}(s) + W_{11}(s) + W_{12}(s)$$

가 되고, 상태 S_2에서의 전달함수 $W_{E2}(s)$는

$$W_{E2}(s) = W_{20}(s) + W_{21}(s) + W_{22}(s)$$

가 된다.

S8TEP4. 상태 S_1에서의 모멘트 발생함수 $M_{E1}(s)$

상태 S_1과 S_2에서의 모멘트 발생함수 $M_{E1}(s)$, $M_{E2}(s)$는

$$M_{E1}(s) = \frac{W_{E1}(s)}{W_{E1}(0)} \quad , \qquad M_{E2}(s) = \frac{W_{E2}(s)}{W_{E2}(0)}$$

가 된다.

S8TEP 5. 한 스테이션에서 서비스가 수행되는 시간에대한 전체 전달함수 $W_{LM}(s)$

현 스테이션에서의 전체 전달함수 $W_{LM}(S)$는

$$W_{LM}(s) = P_A(0) \cdot 1 + P_A(1) \cdot M_{E1}(s) + P_A(2) \cdot M_{E2}(s)$$

가 된다.

STEP 6. 한 스테이션에서의 모멘트 발생함수 $M_{LM}(s)$

$$M_{LM}(s) = \frac{W_{LM}(s)}{W_{LM}(0)}$$

STEP 7. 한 스테이션에서의 평균 서비스 시간 T_S와 평균 토큰 회전 시간 T_R

$$T_S = \frac{d}{ds} M_{LM}(s)|_{s=0}$$

$$T_R = N \cdot (T_S + T_O)$$

N : 시스템에서 스테이션의 갯수

T_O : 오버헤드 시간 (상수)

이 식들을 이용하여 앞서 살펴보았던 k=1인 경우와 마찬가지로 평균 서비스 시간과 평균 토큰 회전시간을 구할 수 있다. 물론 평균 서비스 시간과 평균 토큰 회전시간은 모두 알고있거나, 주어진 값들을 변환시켜 얻을 수 있는 것들로 구성된다.

6.4 모의실험 결과와의 비교

다음은 앞서 살펴보았던 k=1인 경우에 대한 수학적 계산값과 모의실험 결과이다. 그림 6-7, 6-8, 6-9는 k=1인 경우에서 메시지 발생율 λ만을 변화시켰을 경우, 평균 메시지 처리시간만 변화시켰을 경우, 토큰 사용 가능 시간만 변화시켰을 경우에 대한 모의실험 결과와 해석 결과를 비교한 것이다. 여기서 스테이션의 갯수는 4개로, 오버헤드 시간은 10μsec로 놓았다. 기준으로 삼은 표준안[2][3]에는 해당 시간에대한 정의가 아직 내려져 있지 않았다. 그래서, 여러 논문을 참조하고, 맵의 경우도 참조하여 기준값을 정해 사용하였다. 서비스 발생율은 초당 메시지 발생갯수이며, 평균 메시지 처리시간과 토큰 사용 가능시간은 모두 μsec단위의 값을 갖는다. 모의 실험과 해석시 사용한 비트 전송율은 1M BPS(초당 전송비트)로 하였다. 그리고, 각 스테이션은 동일한 레벨의 우선순위를 가지며, 안정상태에서 모두 동일한 평균 서비스 시간을 갖는다.

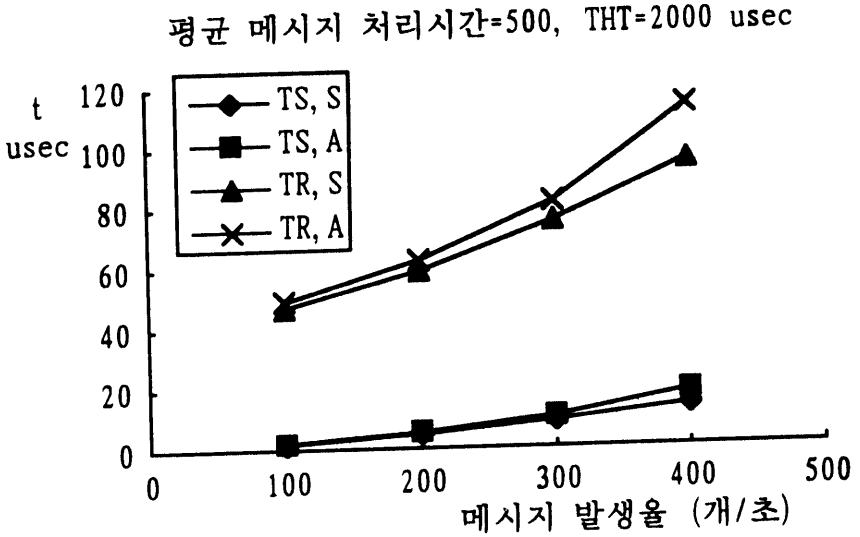
그림 6-7은 평균 서비스 시간($1/\mu$)을 500μsec, 토큰 사용 가능 시간을 2000μsec로 놓고, 메시지 발생율 λ를 100개/초, 200개/초, 300개/초, 400개/초, 500개/초

로 변화시켰을 때의 평균 서비스 시간과 평균 토큰 회전시간이다. 그리고, 그림 6-8은 앞서 그림 6-7에서 평균 서비스 시간($1/\mu$)을 600μsec로 바꿔 비교한 것이다. 그림 6-9는 토큰 사용 가능 시간을 1000μsec로, 메시지 발생율 λ를 100개/초와 200개/초로 놓고, 평균 메시지 처리 시간($1/\mu$)을 500, 600, 700, 800, 900μsec로 변화시킨 경우이다. 그림 6-10은 그림 6-9에서 토큰 사용 가능시간을 2000μsec로 바꾼 경우이다. 그림 6-11은 토큰 사용 가능 시간은 2000μsec로, 평균 메시지 처리 시간($1/\mu$)은 500μsec로 놓고, 메시지 발생율 λ를 50개/초로, 60개/초, 70개/초, 80개/초, 90개/초로 변화시킨 경우이다.

그림 6-7과 6-8에서 보면, 발생율 λ의 값이 작은 100개/초의 경우에는 오차가 1% 정도이고, 400개/초의 경우는 최대 오차가 약 30%정도이다. 그러나, 30%정도의 오차도 그 범위를 시간으로 보면 8.8μsec정도로 오버헤드시간(10μsec)보다도 작다. 또한, 보통 생산기기들의 메시지 처리시간과 비교할 때 상당히 작은 값이기 때문에 시스템 성능에 그리 크게 영향을 끼치지 않는다. 그리고, 모델오차의 영향이 가장 큰 경우가 k=1인 경우이기 때문에 k=N의 경우에서는 모델오차에 의해 발생되는 오차는 더욱 작아질 것이다. 그림 6-9와 6-10의 경우는 오차가 5%미만이다. 따라서, 앞서의 오차보다 더 작은 경우이므로 해석적 결과와 모의실험의 결과가 비슷하다고 볼 수 있다. 앞서 언급했듯이 이러한 발생율의 범위에 대해 정의된 곳이 없기 때문에 기타 다른 논문들에서 얻은 수치를 이용하여 대입하였다. 이 필드버스 시스템이 미니맵과 비슷하므로 여러 미니맵 성능평가 연구에서 사용한 값을 참조하고, 필드버스가 미니맵보다 더 낮은 레벨의 통신규약이기 때문에 미니맵에서의 발생율보다 큰 발생율을 적용시켰다. 미니맵 성능평가들[19][20][23][24]을 살펴보면, 모두 메시지 발생율을 10개/초에서 100개/초의 범위를 선택하고 있다. 따라서, 본 논문에서는 이 범위보다 더 많은 발생율을 고려하여 평가한 것이다. 그림 6-11은 이러한 조건에 대한 것을 살펴보고자 메시지 발생율을 50개/초에서 90개/초로 놓고 살펴본 것이다. 여기서 오차는 최대 5%의 오차가 발생하였다. 그리고, 그 오차의 크기는 0.06μsec에 불과하다.

오차에 대해 살펴본 결과, 다음과 같은 것을 얻을 수 있었다. 오차가 많이 발생하는 경우는 다른 여러 경우보다 메시지 발생량과 서비스된 메시지의 양사이에서의 비율이 더 낮은 것을 알 수 있었다. 이것은 모의실험에 의해 얻은 결과로써 서비스된

양이 75%보다 큰 경우에는 오차가 5%이하이고, 그렇지않은 경우는 그 이상의 오차가 발생하였다. 따라서, 원활한 시스템 운영을 위해서는 이 수치에 따른 발생율과 메시지 처리시간의 값이 조정이 필요하다. 메시지 처리시간이 증가되는 것은 한 스테이션이 서비스할 수 있는 시간이 줄어드는 것이므로, 비례적으로 토큰 사용 가능시간을 늘려주거나, 발생율을 줄여줘야 한다. 그러나, 토큰 사용 가능시간은 그리 많은 효과가 없기 때문에 발생율을 줄이는 것이 적합하다. 발생율을 줄이기 어려운 경우에는 스테이션의 수를 줄여 보완해야한다. 이러한 발생율과 처리시간, 토큰 사용 가능시간의 조절이 통신망의 성능에 많은 영향을 끼치게 된다. 미니맵의 경우에는 전체적인 성능해석과 더불어 이러한 사용되는 시간들의 조절에 대한 연구가 많이 나오고 있다. 따라서, 필드버스 시스템에서도 이러한 시간 조절에 대한 연구가 진행되어야 한다.



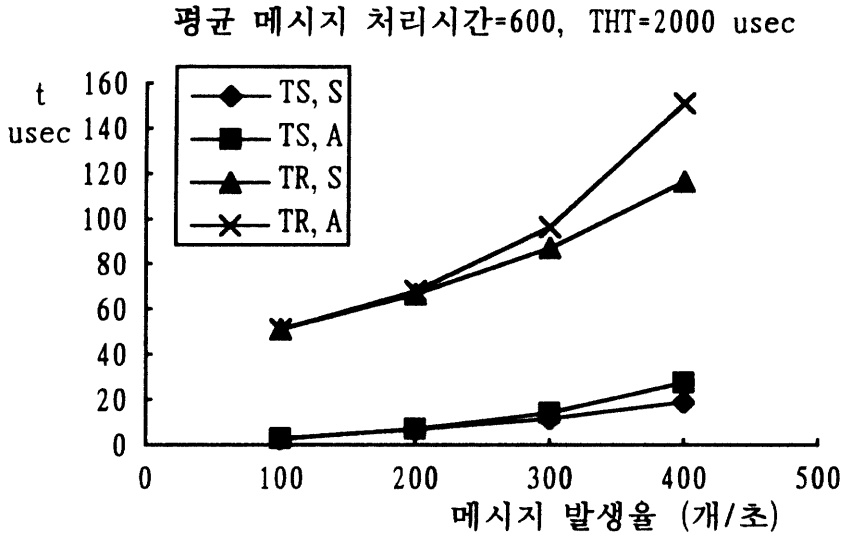
TS,S : 모의실험결과(평균 서비스 시간)

TS,A : 해석결과(평균 서비스 시간)

TR,S : 모의실험결과(평균 토큰 회전시간)

TR,A : 해석결과(평균 토큰 회전시간)

그림 6-7. 평균 메시지 처리시간=500 μsec, 토큰 사용가능시간 = 2000 μsec인 경우



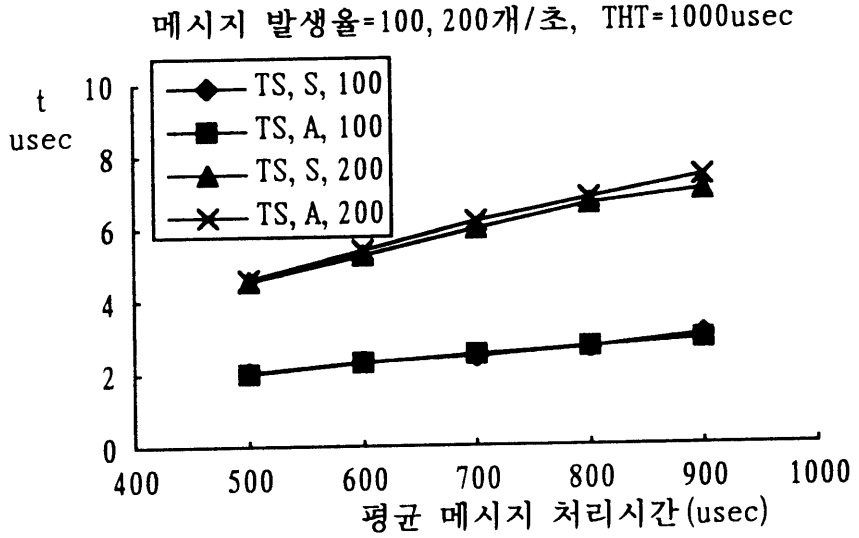
TS,S : 모의실험결과(평균 서비스 시간)

TS,A : 해석결과(평균 서비스 시간)

TR,S : 모의실험결과(평균 토큰 회전시간)

TR,A : 해석결과(평균 토큰 회전시간)

그림 6-8. 평균 메시지 처리시간=600 μ sec, 토큰 사용가능시간 = 2000 μ sec인 경우



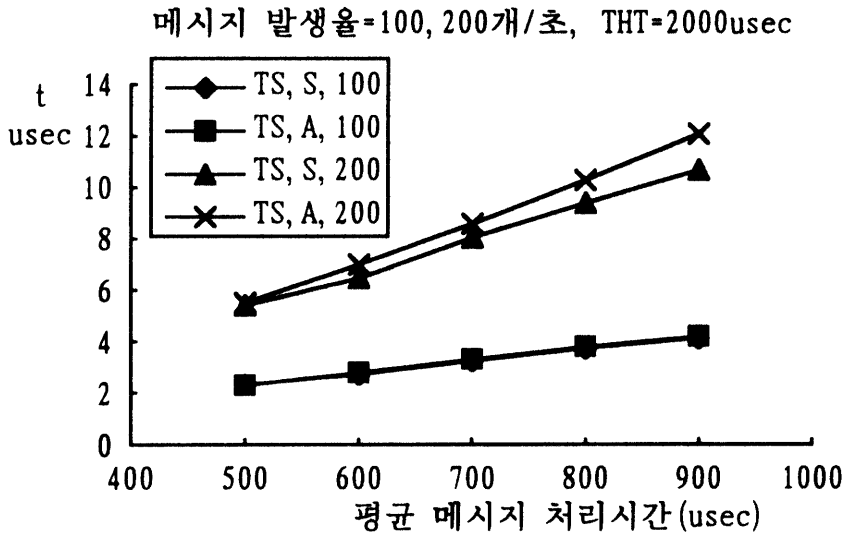
TS, S, 100 : 모의실험결과(평균 서비스 시간)

TS, A, 100 : 해석결과(평균 서비스 시간)

TR, S, 200 : 모의실험결과(평균 토큰 회전시간)

TR, A, 200 : 해석결과(평균 토큰 회전시간)

그림 6-9. 메시지 발생율=100, 200개/초, 토큰 사용가능시간=1000 μ sec인 경우



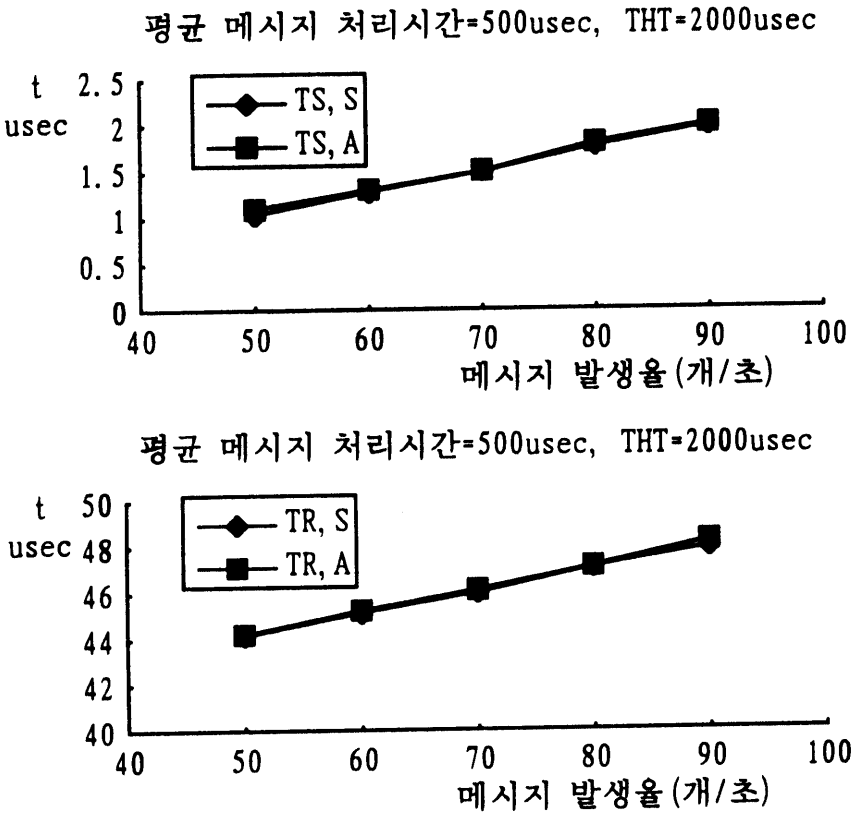
TS, S, 100 : 모의실험결과(평균 서비스 시간)

TS, A, 100 : 해석결과(평균 서비스 시간)

TS, S, 200 : 모의실험결과(평균 서비스 시간)

TS, A, 200 : 해석결과(평균 서비스 시간)

그림 6-10. 메시지 발생율=100, 200개/초, 토큰 사용가능시간=2000 μ sec인 경우



TS,S : 모의실험결과(평균 서비스 시간)

TS,A : 해석결과(평균 서비스 시간)

TR,S : 모의실험결과(평균 토큰 회전시간)

TR,A : 해석결과(평균 토큰 회전시간)

그림 6-11. 평균 메시지 처리시간=500 μ sec, 토큰 사용가능시간=2000 μ sec인 경우

7. 필드버스 테이타링크층의 설계

데이타링크층 소프트웨어는 응용계층으로부터 요구 프리미티브를 받아서 그 프리미티브에 해당되는 서비스제공에 필요한 PDU(Protocol Data Unit)을 만든다. 이렇게 만들어진 PDU는 물리층을 통해 다른 스테이션으로 전달된다. 또한 다른 스테이션으로부터 PDU를 받아 이를 분석, 처리하고 응용 계층에 indication 또는 confirm 프리미티브를 전달한다. 필드버스의 데이타링크층에서 구분되는 각 상태들은 다음의 그림 7-1과 같다.

7.1. 상태별 동작

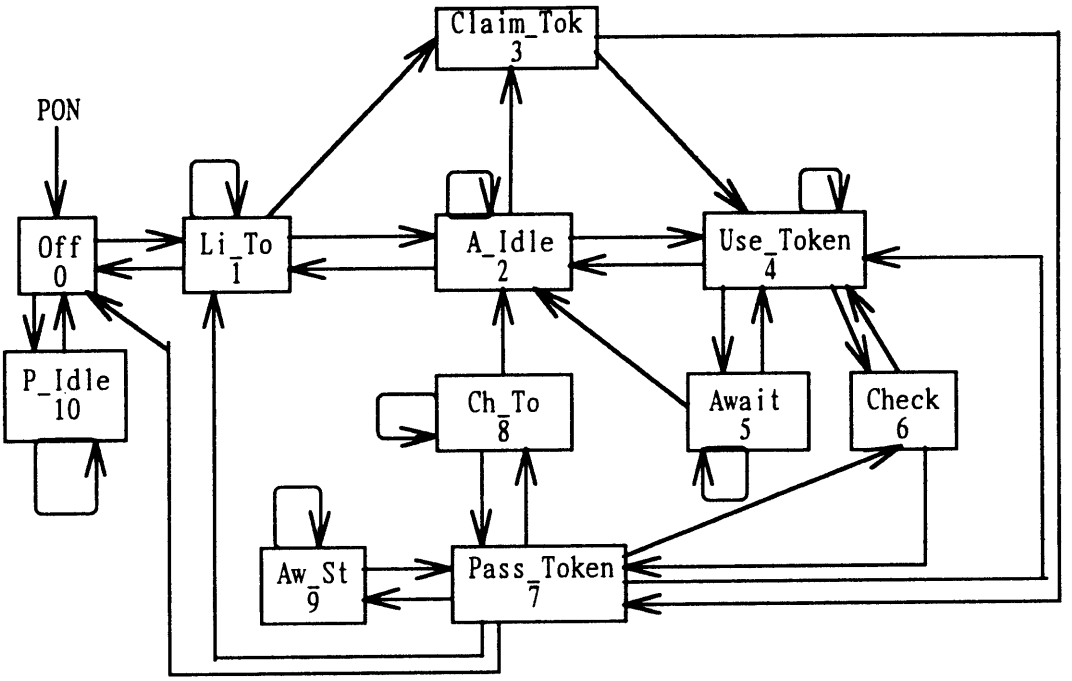
그림 7-1에 나타난 각 상태들에서의 동작들을 살펴보면 다음과 같다.

0 OFFLINE

전원을 인가했을 때의 초기 상태로 각 스테이션들은 자체검사를 수행한다. 자체테스트를 한 후, 요구되는 운영 매개변수들이 초기화될 때까지 "OFFLINE" 상태를 유지한다.

1 LISTEN_TOKEN

운영 매개변수들이 초기화된 후, 마스터 스테이션의 FDL은 "LISTEN_TOKEN" 상태로 들어간다. 논리적 링에 들어갈 준비가 되었을 경우, 마스터 스테이션의 FDL은 논리적 토큰링에서 이미 존재하는 마스터 스테이션들을 식별하기위해 라인을 모니터한다. 이전 스테이션(PS)에의해 전송된 "Request FDL Status"에 의해 어드레스될 때까지 "LISTEN_TOKEN"을 유지한다. 만일 "Request FDL Status"가 도달했을 경우, "ready to enter logical token ring"으로 응답한다. 그리고나서, "ACTIVE_IDLE" 상태로 들어간다.



PON Power On

0 OFFLINE

1 LISTEN_TOKEN

2 ACTIVE_IDLE

3 CLAIM_TOKEN

4 USE_TOKEN

5 AWAIT_DATA_RESPONSE

6 CHECK_ACCESS_TIME

7 PASS_TOKEN

8 CHECK_TOKEN_PASS

9 AWAIT_STATUS_RESPONSE

10 PASSIVE_IDLE

그림 7-1. 필드버스 데이타링크층의 상태도

만일 마스터 스테이션이 남아있을 때 토큰 프레임에서 소스 주소(SA)로써 자신의 어드레스를 인식한다면, 이미 링에 동일 주소를 가진 다른 마스터 스테이션이 존재한다는 것을 의미한다. 이때 FDL은 "OFFLINE" 상태로 되돌아가고, 관리자(FMA 1/2)에게 이 사실을 알린다.

만일 FDL이 타임아웃 주기동안 버스 활성 여부를 관측하지 못한다면, 논리적 토큰링의 초기화나 복원이 필요하다고 가정하여 "CLAIM_TOKEN" 상태로 들어가 토큰의 요구를 시도하고, 논리적 링의 (재)초기화를 시도하게 된다.

2 ACTIVE_IDLE

활성화되지 않은 버스 라인을 관찰한다. 만일 현 스테이션이 목적 주소가 되는 동작 프레임이 수신되었다면, 요구된 것에 대한 확인이나 응답을 한다. 자신에게 어드레스된 토큰 프레임을 받은 후, 스테이션이 논리적 토큰링에서 유지되길 원한다면 "USE_TOKEN" 상태로 들어가고, 그렇지 않은 경우는 "LISTEN_TOKEN" 상태로 되돌아간다. 만일 소스 주소가 현 스테이션의 주소가 되는 토큰 프레임이 수신되었다면, 에러로 처리한다.

만일 FDL이 타임아웃 주기동안 버스 활성 여부를 관측하지 못한다면, 논리적 토큰링의 복원이 필요하다고 가정하여 토큰의 요구를 시도하고, 논리적 링의 (재)초기화를 시도한다.

3 CLAIM_TOKEN

FDL은 타임아웃이 되었을 때 "ACTIVE_IDLE"과 "LISTEN_TOKEN" 상태에서 "CLAIM_TOKEN" 상태로 들어온다. 이 상태는 재초기화를 시도하거나 초기화 시작을 시도한다. 재초기화할 때, 스테이션의 상태 리스트(LAS와 GAPL)는 여전히 유용하다. 그러므로, "USE_TOKEN" 상태로 즉시 들어가게된다.

초기화를 하기 시작할 때, 다음 스테이션 주소가 현 스테이션의 주소인 경우에는 "PASS_TOKEN" 상태로 들어간다. 토큰 전송 후, 자신의 GAPL(GAP List)과 NS(다음 스테이션의 주소, Next Station)는 뒤따르는 스테이션 주소들에 요구하는 "Request FDL Staus" 방법에의해 "AWAIT_STATUS_RESPONSE" 상태에서 생성된다.

4 USE_TOKEN

FDL은 토큰을 수신한 후나, (재)초기화 후 "USE_TOKEN" 상태로 들어간다. 이것은 FDL이 높은 순위와 낮은 순위 메시지 싸이클들을 수행하는 상태이다.

이 상태에 들어가면, T_RR(Real Rotation Time)은 토큰 회전 타이머로 부터 읽히고, 타이머는 다시 동작된다. 높은 순위의 메시지 싸이클은은 항상 가능하다. 우선 순위에 상관없이 단지 실행 순간에 T_RR < T_TR (T_TR : Target Rotation Time) 일 경우만 수행된다.

동작 프레임이 각기 전송된 후, FDL은 "AWAIT_DATA_RESPONSE" 상태로 들어가고, 슬롯 타이머를 동작 시킨다. 전송된 프레임이 사용자에게 확인되자 마자 "USE_TOKEN" 상태로 들어간다.

FDL은 "USE_TOKEN" 시작시, 수행된 높은 순위 메시지 싸이클이 없거나 한개의 높은 순위나 낮은 순위 메시지 싸이클이 완료된 후라면 "CHECK_ACCESS_TIME" 상태로 들어간다.

5 AWAIT_DATA_RESPONSE

이 상태는 동작 프레임의 전송 후, 들어가게 된다. FDL은 확인이나 응답 프레임의 수신에대해 한 슬롯 시간동안 대기한다.

SDN(Send Data with No Ack.) 서비스의 경우(확인을 갖지 않은 데이타 서비스의 경우) 확인이 기다려지지 않는다. FDL은 발생 가능한 요구의 처리를 위해 "USE_TOKEN" 상태로 되돌아간다. 확인이나 응답을 요구하는 경우, FDL은 다음의 사건들중 한 가지를 기다린다.

① 확인이나 응답 프레임이 요구의 시작자(Initiator)에 어드레스되는 것

② 어떤 다른 유효 프레임(토큰 프레임이나 동작 프레임)이 수신되는 것

③ 유효하지 않은 프레임이나 슬롯 시간이 만료되는 것

확인이나 응답 프레임을 받아 처리한 후, FDL은 발생 가능한 요구를 처리하기 위해 "USE_TOKEN" 상태로 되돌아간다.

다른 유효 프레임의 수신은 에러 발생을 가리킨다. FDL은 "ACTIVE_IDLE" 상태로

들어가고, 수신된 프레임을 버린다.

만일 유효하지 않은 프레임이 수신되거나 슬롯 시간이 만료되었다면, FDL은 동작 프레임의 전송을 재시도한다. 재시도 후, 유효한 확인이나 응답이 수신이 되지 않는다면 그 사실을 사용자에게 통보하고, "USE_TOKEN" 상태로 되돌아간다. 에러의 경우 이 스테이션으로의 더이상의 요구는 정상적인 메시지 싸이클(Send/Request Data)이 수행될 때까지는 반복되지 않는다.

6 CHECK_ACCESS_TIME

이 상태에서 유용한 토큰 사용 가능시간은 TTR - TRR(TTR:Target Rotation Time, TRR:Real Rotation Time)로 계산된다. 토큰 사용 가능시간이 여전히 유용할 경우에만 "USE_TOKEN" 상태로 되돌아간다. 그렇지 않은 경우, "PASS_TOKEN" 상태로 들어간다.

7 PASS_TOKEN

"PASS_TOKEN" 상태에서 FDL은 논리적 링에서 다음 스테이션(NS)로의 토큰 패싱을 시도한다. 토큰 프레임 전송시, 전송부가 바르게 동작하고 있는지 연속적으로 모니터한다. 만일 그 자신의 토큰 프레임을 수신할 수 없다면, 전송·수신 채널에서의 치명적 에러이므로 FDL은 논리적 링에서 그것의 활성화를 금지시키고 "OFFLINE" 상태로 들어간다. 그리고 관리자(FMA 1/2)에 알린다.

FDL은 손상된 그 자신의 토큰 프레임을 수신한다면, 이것은 일시적 결함이 있는 전송부·수신부나 버스 라인에의해 발생되는 것으로 본다. 단지 토큰 프레임이 정확하게 수신된 후, FDL은 "CHECK_TOKEN_PASS" 상태로 들어간다. 토큰 프레임이 재전송되고(NS로부터 재동작하지 않기에) 잘못되었다고 조사된 경우에는 논리적 링에서 그 활성화 여부가 금지되며, "LISTEN_TOKEN" 상태로 들어간다. 그리고, 관리자(FMA 1/2)에 알린다.

GAP 갱신 시간이 만료되었지만 토큰 사용 가능시간 T_TH (Token Holding Time)가 유효한 경우, 토큰을 패싱하기 전에 필요하다면 논리적 링에서 그것을 포함하기 위해 GAP에서 가능한한 한개 새로운 스테이션을 저장하도록 시도한다. (여기서

GAP이란 논리적 토큰링에서 현 스테이션(TS)에서 다음 스테이션(NS)까지의 스테이션 어드레스들의 범위를 말한다.) FDL은 "Request FDL Status"를 전송하고, "AWAIT_STATUS_RESPONSE" 상태로 들어간다. 만일 새로운 한 마스터 스테이션이 논리적 링으로의 참여가 승인될 경우 FDL은 이 스테이션으로 토큰을 패스시킨다.

만일 상태 요구가 링에서 포함되길 원하지 않는 새로운 스테이션에의해 응답된다면, 그 스테이션은 GAPL(GAP list)에 들어가게 된다. 재시도 후에도 응답이 없는 경우에는 이 스테이션을 GAPL에서 제거한다.

만일 GAP 유지동안 새로운 마스터 스테이션이 응답하지 않는다면, FDL은 원래의 NS로 토큰을 통과시키고, "CHECK_TOKEN_PASS" 상태로 들어간다. 단지 성공여부가 알려지지 않을 경우만(버스상에 활성 스테이션만 있는 순간) 자신으로 토큰을 통과시키고 "USE_TOKEN" 상태로 되돌아간다.

8 CHECK_TOKEN_PASS

"CHECK_TOKEN_PASS"은 FDL이 토큰을 패스시키도록 스테이션의 재동작에대해 한 슬롯 시간동안 대기하는 상태이다. 이 대기 시간은 토큰 프레임의 수신과 어드레스된 스테이션의 전송 재동작을 발생시키는 것 사이의 지연을 허용한다.

만일 FDL이 한 슬롯 시간에서 유효한 프레임 헤더를 보장한다면, 토큰 패싱이 성공적이었다고 가정한다. 프레임은 "ACTIVE_IDLE" 상태에서 수신되었었던것 처럼 처리된다. 만일 유효하지 않은 프레임이 한 슬롯 시간에서 보장되었다면, FDL은 다른 스테이션이 활성화되었다고 가정하고 "ACTIVE_IDLE" 상태로 들어간다.

만일 FDL이 한 슬롯 시간안에 어떤 프레임도 받지 못한다면, "PASS_TOKEN" 상태로 들어간다.

9 AWAIT_STATUS_RESPONSE

이 상태에서 FDL은 확인 프레임에대해 한 슬롯 시간을 대기한다. 부적합한 프레임이 수신되거나 아무 프레임도 수신되지 않을 경우, "PASS_TOKEN" 상태로 되돌아간다.

만일 FDL이 확인 프레임 대신 어떤 다른 프레임을 수신할 경우(다중 토큰이 존

재한다는 것을 가리킴) "ACTIVE_IDLE" 상태로 들어간다.

10 PASSIVE_IDLE

매개변수들이 초기화된 후, 슬레이브 스테이션의 FDL은 "PASSIVE_IDLE" 상태로 들어가고, 라인을 관찰한다. 그리고, 전달된 요구에 따라 메시지 싸이클을 수행한다.

"Reset FDL" 서비스나 치명적 에러(예, 연속적인 전송)이 발생될 경우에 "OFFLINE" 상태로 되돌아 간다.

7.2. 서비스 프리미티브

여기서 .req는 .request를, .ind는 .indication을, .con은 .confirm을 의미한다.

1) FDL 사용자에서 FDL로의 서비스 프리미티브

프리미티브명	매개변수 의미	
XXXX.req	SSAP	(로컬 사용자의 소스 서비스 접근점)
	DSAP	(원격 사용자의 목적 서비스 접근점)
	Rem_add	(원격 주소)
	L_sdu	(링크 서비스 데이타 단위)
	Serv_class	(데이타 전달에대한 우선순위)
FDL_REPLY_UPDATE .req	SSAP	(로컬 사용자의 소스 서비스 접근점)
	DSAP	(원격 사용자의 목적 서비스 접근점)
	Serv_class	(데이타 전달에대한 우선순위)
	Transmit	(갱신 횟수)

* XXXX : FDL_DATA_ACK, FDL_DATA, FDL_DATA_REPLY

2) 원격 FDL에서 원격 사용자(remote user)로의 서비스 프리미티브

프리미티브명	매개변수 의미
XXXX.ind	SSAP (수신된 프레임의 SSAP) DSAP (수신된 프레임의 DSAP) Loc_add (수신된 프레임의 로컬 주소) Rem_add (수신된 프레임의 원격 주소) L_sdu (수신된 프레임의 L_sdu) Serv_class (수신된 프레임의 Serv_class)

3) FDL에서 로컬 사용자로의 서비스 프리미티브

프리미티브명	매개변수 의미
XXXX.con	SSAP (로컬 사용자의 SSAP) DSAP (원격 사용자의 DSAP) Rem_add (목적 FDL 주소) Serv_class (데이타 전달에 관계된 우선순위) L_status (현재 상태를 알림)

* XXXX : FDL_DATA_ACK, FDL_DATA, FDL_DATA_REPLY, FDL_REPLY_UPDATE

4) FMA1/2 사용자에서 FMA1/2로의 서비스 프리미티브

프리미티브명	매개변수 의미
FMA1/2_RESET.req	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) * SAP : Service Access Point
FMA1/2_SET_VALUE.req	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Variable_name (표 7-1 참조) Desired_value (표 7-2 참조)
FMA1/2_READ_VALUE.req	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Variable_name (표 7-1 참조)
FMA1/2_IDENT.req	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Rem_add(원격 스테이션의 주소)
FMA1/2_LSAP_STATUS.req	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) LSAP (링크 서비스 접근점) Rem_add(원격 스테이션의 주소)

5) FMA1/2에서 FMA1/2 사용자로의 서비스 프리미티브

프리미티브명	매개변수 의미
FMA1/2_RESET.con	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) * SAP : Service Access Point
FMA1/2_SET_VALUE.con	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Variable_name (표 7-1 참조) Desired_value (표 7-2 참조)
FMA1/2_READ_VALUE.con	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Variable_name (표 7-1 참조)
FMA1/2_IDENT.con	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) Rem_add(원격 스테이션의 주소)
FMA1/2_LSAP_STATUS.con	MSAP_0 (관리 서비스 접근점, 로컬 FMA1/2 사용자의 SAP) LSAP (링크 서비스 접근점) Rem_add(원격 스테이션의 주소)

7.3. 변수 및 기능함수 정의

상태천이 설계시 사용되는 변수들은 다음과 같은 것들이 있다.

1) Power_OK

전원이 인가되었음을 의미한다.

2) Service_type

현재 요구에대한 서비스 형태가 어떤 것인지를 나타낸다.

설정값 : SDA, SDN, SRD

이 변수값은 프레임 제어 영역(FC)에서 비트 1-4까지의 영역인 평션 코드에 의해 설정된다. 그 코드값을 살펴보면 다음의 표 7-3과 같다.

3) FATAL_ERROR

치명적인 에러가 발생되었을 경우를 의미한다. 예를들어, 센서가 고장났거나, 통신부가 계속적으로 잘못 동작할 경우를 말한다.

표 7-1. 변수명

운영 매개변수	
이 름	의 미
TS	이 스테이션의 FDL 주소
Baud_rate	비트당 전송속도
Medium_red	리던던트 매디아의 유용성
HW-Release	하드웨어 릴리즈 번호
SW-Release	소프트웨어 릴리즈 번호
TSL	슬롯 시간
min T_{SDR}	최소 스테이션 지연 시간
max T_{SDR} *	최대 스테이션 지연 시간
T_{QUI} *	전송부 대기시간/리피터 전환시간
T_{SET} *	셋업 시간
T_{TR} *	Target Rotation Time
G *	GAP 갱신 인자
in_ring_desired *	논리적 토큰 링으로의 추가나 종료 요구 여부
HSA *	최대 스테이션 주소
max_retry limit *	최대 재시도 횟수
카 운 터	
Frame_sent_count*	프레임 전송 횟수
Retry_count *	재시도 프레임의 갯수
SD_count	정확한 시작 구획의 횟수
SD_error_count	잘못전달된 시작 구획의 횟수
만일 카운터가 그 최대치에 도달할 경우, 이 카운터와 비교 카운터는 정지된다. 카운터가 0으로 리셋될 경우, 이 카운터는 다시 카운트할 수 있도록 놓여진다.	
* 마스터 스테이션만 가능	

표 7-2. 변수들에 대한 허용치

운영 매개변수들	
이 름	의 미
TS	1 옥테트 주소 영역, 범위는 0에서 126까지로 설정됨
Baud_rate	9.6, 19.2, 93.75; 187.5; 500 kbit/s
Medium_red	single/redundant
HW-Release	LE_HR; HW_release
SW-Release	LE_SR; SW_release
T_{SL}	2^0 에서 2^{16}-1사이의 값을 가짐 (Bit Times)
min T_{SDR}	2^0 에서 2^{16}-1사이의 값을 가짐 (Bit Times)
max T_{SDR} *	2^0 에서 2^{16}-1사이의 값을 가짐 (Bit Times)
T_{QUI} *	0 에서 2^8-1사이의 값을 가짐 (Bit Times)
T_{SET} *	2^0 에서 2^8-1사이의 값을 가짐 (Bit Times)
T_{TR} *	2^0 에서 2^{24}-1사이의 값을 가짐 (Bit Times)
G *	1 에서 100사이의 값을 가짐
in_ring_desired *	true; false
HSA *	2 에서 126까지의 값을 가짐
max_retry limit *	1 에서 8까지의 값을 가짐
카운터	
Frame_sent_count*	0
Retry_count *	0
SD_count	0
SD_error_count	0
만일 카운터가 그 최대치에 도달할 경우, 이 카운터와 비교 카운터는 정지된다. 카운터가 0으로 리셋될 경우, 이 카운터는 다시 카운트할 수 있도록 놓여진다.	
* 마스터 스테이션만 가능	

4) Rx_status

수신된 프레임이 갖고 있는 값들이 이 구조체에 저장된다. 그 형태는 다음과 같다.

DA : 목적 주소를 나타낸다.

SA : 소스 주소를 나타낸다.

status : 수신된 프레임이 유효한 프레임인지를 나타낸다. VALID일 경우, 유효 프레임이고, INVALID인 경우는 유효하지 않은 프레임을 나타낸다.

type : 프레임의 형태를 나타낸다. REQ_FRAME 일 경우, 현 프레임이 요구(request) 프레임이다. ACK_RESP_FRAME인 경우는 확인/응답 프레임임을 나타낸다.

new : 처리되지 않은 프레임일 경우, 즉 새로 수신된 프레임일 경우 TRUE로 세트된다. 그렇지 않을 경우 FALSE로 세트된다.

5) Rx_token_frame

전달된 프레임이 토큰 프레임일 경우 TRUE로 세트된다.

표 7-3에 나타난 것처럼 펑션이 "Request FDL status"인 경우 TRUE로 세트된다. (기능 코드에서 b7이 1이고, 코드번호가 9번인 경우 TRUE로 세트)

6) Token_trans_status

토큰 전달이 제대로 실행되지 않았을 경우 FAIL로 세트된다. 제대로 전달되었을 경우 OK로 세트된다.

7) want_in_ring

현 스테이션이 논리적 토큰 링에 들어가길 원할 경우 TRUE로 세트된다. 그렇지 않은 경우는 FALSE로 세트된다. 이 변수는 프레임 제어 영역(FC)에서 6, 5번째 비트의 값에의해 설정된다. 다음의 표 7-4에 나타나 있는 것과 같이 비트 6만 1일 경우 TRUE로 세트된다.

표 7-3 전송 기능 코드들

코드번호	기 능	
	Frame Type b7 = 1 (Request, Send/Request frame)	
0,1,2	IEC 870-5-2*, FC Code 0,1,2	
3	Send Data with Acknowledge low	
4	Send Data with No Acknow. low	
5	Send Data with Acknowledge high	
6	Send Data with No Acknow. high	
7	Reserved (Req. Diagnosis Data)	
8	IEC 870-5-2*, FC Code 8	
9	Request FDL Status with Reply	
10,11	Reserved	
12	Send and Request Data low	
13	Send and Request Data high	
14	Request Ident with Reply	
15	Request LSAP Status with Reply	
	(Code No 14 and 15: FMA1/2)	
	Frame Type b7 = 0 (Acknowledgement, Response frame)	
0	Acknowledgement positive	(OK)
1	Acknowledgement negative	
	FDL/FMA1/2 User Error	(UE)
2	Acknowledgement negative	
	no Resource for Send data (& no Response FDL Data)	(RR)
3	Acknowledgement negative	
	no Service activated	(RS)
4 to 7	Reserved	
8	Response FDL/FMA1/2 Data low (& Send Data ok)	
9	Acknowledgement negative	
	no Response FDL/FMA1/2 Data (& Send Data ok)	(NR)
10	Response FDL Data high (& Send Data ok)	(DH)
11	Reserved	
12	Response FDL Data low,	
	no Resource for Send Data	(RDL)
13	Response FDL Data high,	
	no Resource for Send Data	(RDH)
14,15	Reserved	
* 현재 제공되지 않는다.		

* 표 7-4. 프레임 제어 영역에서 스테이션의 형태와 FDL 상태

b6	b5	설 명
0	0	슬레이브 스테이션
0	1	논리적 토큰 링에 들어갈 준비가 되지 않은 마스터 스테이션
1	0	논리적 토큰 링에 들어갈 준비가 되어있는 마스터 스테이션
1	1	논리적 토큰 링에 있는 마스터 스테이션

8) in_ring

현 스테이션이 논리적 토큰 링에 있는 지를 나타낸다. 위의 표 7-4에서 보였듯이 프레임 제어 영역(FC)의 비트 6와 5가 모두 1인 경우에 TRUE로 세트된다.

9) want_remain_ring

현 스테이션이 논리적 토큰 링에 계속 머물러 있고자 할 경우 TRUE로 세트된다. 그렇지 않은 경우는 FALSE로 세트된다.

10) h_msg

현재 저장된 높은 순위의 메시지 수를 나타낸다.

11) msg_complete

메시지 처리가 완료되었을 경우 YES로, 그렇지 않은 경우는 NO로 세트된다.

12) retry_count

현재 재시도 횟수를 나타낸다.

13) exist_new_master

새로운 마스터 스테이션이 존재할 경우 YES로 세트된다.

14) Station_type

표 7-4에서 b6와 b5가 모두 0인 경우만 슬레이브 스테이션임을 나타내는

SLAVE를 갖고, 그렇지 않은 경우는 모두 MASTER를 갖는다.

15) NS

현 스테이션이 토큰을 넘길 때의 목적 주소로 GAPL에서 얻는다.

16) T_TH

현재 토큰 사용 가능시간을 나타낸다.

17) T_SL

현재 슬롯 시간을 나타낸다.

사용된 기능함수들은 다음과 같다.

18) Send_frame ( )

프레임을 목적주소로 전달하는 기능을 갖는다.

19) Timer_set(Timer, value)

전달된 타이머의 값을 다시 세트하는 기능을 갖는다.

20) Discard_frame()

현재 수신버퍼에 저장되어 있는 프레임을 제거시키는 기능을 갖는다.

21) Resend_frame()

현재 송신버퍼에 저장되어 보낸 적이 있는 프레임을 다시 재전송한다.

22) Report_to_FMA(Status)

관리자(FMA1/2)로 전달된 인수에 따른 현재 상태를 알려주는 기능을 한다.

23) Timer_read(Timer)

인수로 전달된 타이머의 현재 값을 읽는다.

7.4. 프레임의 구조

다음의 각 프레임들에서 사용되는 약어들은 다음과 같다.

SYN : 동기화 주기(Synchronization Period, 최소 33비트가 사용된다.)

SD1 : 시작 구획문자(Start Delimiter, 값은 10H이다.)

SD2 : 시작 구획문자(Start Delimiter, 값은 68H이다.)

SD3 : 시작 구획문자(Start Delimiter, 값은 A2H이다.)

SD4 : 시작 구획문자(Start Delimiter, 값은 DCH이다.)

* SD1에서 SD4는 각기 프레임의 종류를 구분하기위해 각기 다른 값을 갖는다.

DA : 목적 주소(Destination Address)

SA : 소스 주소(Source Address)

FC : 프레임제어(Frame Control)

FCS : 프레임 검사 작업(Frame Check Sequence)

LE : 옥테트 길이(Octet Length, 허용된 값은 4에서 249까지이다)

LEr : 반복된 옥테트 길이(Octet Length repeated)

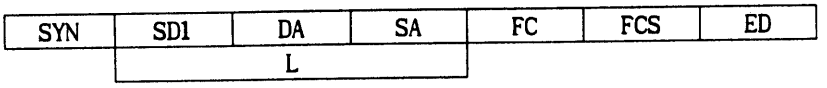
ED : 종료 구획문자(End Delimiter, 값은 16H이다.)

L : 정보 영역 길이(Information Field Length)

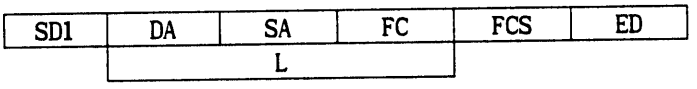
SC : 단일 문자(Single Character, 값은 E5H이다.)

1) 데이타 영역을 가지지 않는 고정된 길이의 프레임들

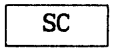
요구 프레임



확인 프레임

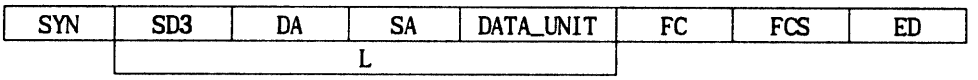


짧은 확인 프레임

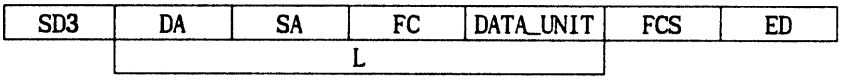


2) 고정된 길이의 데이타 영역을 가지는 프레임들

전송/요구 프레임

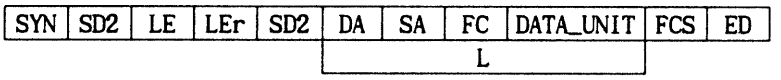


응답 프레임

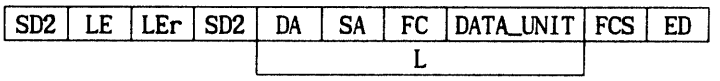


3) 가변적인 데이타 영역을 가진 프레임들

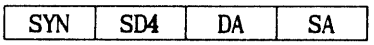
전송/요구 프레임



응답 프레임



4) 토큰 프레임



7.5 상태 천이표

현재상태 사건발생 조건 동작	사건명	다음상태
1. OFFLINE	Initialize_slave_station	PASSIVE_IDLE

슬레이브 스테이션의 초기화 완료시

```
Power_OK
AND Station_type == SLAVE

        FDL_SET_VALUE.request( TS         = TS,
                               Baud_rate  = Baud_rate,
                               Medium_red= Medium_red,
                               HW_Release= HW_Release,
                               SW_Release= SW_Release,
                               T_SL       = T_SL,
                               MIN_T_SDR  = MIN_T_SDR,
                               MAX_T_SDR  = MAX_T_SDR,
                               T_QUI      = T_QUI,
                               T_SET      = T_SET,
                               T_TR       = T_TR,
                               G          = G,
                               in_ring    = in_ring,
                               HSA        = HSA,
                               MAX_retry  = MAX_retry);
```

2. OFFLINE	Initialize_master_station	LISTEN_TOKEN

마스터 스테이션의 초기화 완료시

```
Power_OK
AND Station_type == MASTER

        FDL_SET_VALUE.request( TS         = TS,
                               Baud_rate  = Baud_rate,
                               Medium_red= Medium_red,
                               HW_Release= HW_Release,
                               SW_Release= SW_Release,
                               T_SL       = T_SL,
                               MIN_T_SDR  = MIN_T_SDR,
                               MAX_T_SDR  = MAX_T_SDR,
                               T_QUI      = T_QUI,
                               T_SET      = T_SET,
```

```
                                T_TR       = T_TR,
                                G          = G,
                                in_ring    = in_ring,
                                HSA        = HSA,
                                MAX_retry = MAX_retry);
```

---

3. PASSIVE_IDLE Ack_Resp_Request PASSIVE_IDLE

라인을 조사하여 동작 프레임이 수신되면, 그 요구에 따른 확인이나 응답을 함

```
        Rx_status.new         == TRUE
        AND Rx_status.type    == REQ_FRAME
        AND Rx_status.status == VALID

                if Service_type = SDA
                then
                        FDL_DATA_ACK.indication( SSAP = SSAP,
                                DSAP = DSAP,
                                Loc_add = Loc_add,
                                Rem_add = Rem_add,
                                L_sdu = L_sdu,
                                Serv_class = Serv_class);

                else if Service_type = SDN
                then
                        FDL_DATA.indication( SSAP = SSAP,
                                DSAP = DSAP,
                                Loc_add = Loc_add,
                                Rem_add = Rem_add,
                                L_sdu = L_sdu,
                                Serv_class = Serv_class);

                else if Service_type = SRD
                        AND FDL_DATA_REPLY.request
                then
                        FDL_DATA_REPLY.indication( SSAP = SSAP,
                                DSAP = DSAP,
                                Loc_add = Loc_add,
                                Rem_add = Rem_add,
                                L_sdu = L_sdu,
                                Serv_class = Serv_class
                                Update_status = update_status);

                else if Service_type = SRD
                        AND FDL_REPLY_UPDATE.request
                then
                        FDL_REPLY_UPDATE.confirm( SSAP = SSAP,
```

```
                                Serv_class = Serv_class,
                                L_status = OK );
```

---

4. PASSIVE_IDLE — Reset_FDL — OFFLINE

\# 사용자로 부터 Reset 명령이 도착했을 경우

```
FDL_RESET.request

        FDL_RESET.confirm();
```

---

5. PASSIVE_IDLE — Fatal_error — OFFLINE

\# 치명적인 에러 발생시

```
FATAL_ERROR

        FDL_RESET.request();
```

---

6. LISTEN_TOKEN — Token_rotation_listen — LISTEN_TOKEN

\# 목적 주소가 현 스테이션인 "Request FDL Status"가 전송될 때까지 기다린다

아무 행동도 취하지 않는다.

---

7. LISTEN_TOKEN — Request_status_sa_equ_ts — OFFLINE

\# 토큰 프레임을 살펴보았을 때 소스 주소(SA)가 자신의 주소인 경우,
\# 전송/수신부의 에러로 보고 관리자에게 알린다.

```
Rx_status.new        == TRUE
AND Rx_token_frame == TRUE
AND Rx_status.SA     == TS

        FDL_FAULT.indication( Fault_type = Faulty_transceiver );
```

---

8. LISTEN_TOKEN — No_bus_active_for_time_out — CLAIM_TOKEN

\# 타임아웃 주기동안 버스가 활성화되지 않았을 경우
\# (이 경우 논리적 링의 초기화나 복구가 필요하다고 본다.)

```
TIME-OUT

        FDL_FAULT.indication( Fault_type = Time_out );
```

---

9. LISTEN_TOKEN Request_FDL_status ACTIVE_IDLE

"Request FDL Status"가 수신될 경우,

```
Rx_status.new        == TRUE
AND Rx_token_frame == TRUE
AND want_in_ring    == TRUE
AND Rx_status.SA     != TS
```

send_frame(FC = token_confirm, NS = Rx_status.SA):

---

10. ACTIVE_IDLE Bus_line_listen ACTIVE_IDLE

활성화되는 것 없이 버스 라인을 검색한다.

아무 행동도 취하지 않는다.

---

11. ACTIVE_IDLE No_bus_active_for_time_out CLAIM_TOKEN

FDL이 타임아웃 주기동안 버스가 활성화되지 않을 경우,
논리적 토큰 링의 복구가 필요하다고 보고 관리자에게 알린다.

TIME-OUT

FDL_FAULT.indication(Fault_type = Time_out):

---

12. ACTIVE_IDLE Want_to_remain_ring USE_TOKEN

현 스테이션의 주소가 목적 주소와 같은 토큰 프레임을 수신하였고,
논리적 토큰 링에 남아있길 원할 경우

```
Rx_status.new          == TRUE
AND Rx_token_frame    == TRUE
AND want_remain_ring == TRUE
```

아무 행동도 취하지 않는다.

---

13. ACTIVE_IDLE Not_want_to_remain_ring LISTEN_TOKEN

현 스테이션의 주소가 목적 주소와 같은 토큰 프레임을 수신하였고,
논리적 토큰 링에 남아있길 원하지 않는 경우

```
Rx_status.new          == TRUE
AND Rx_token_frame    == TRUE
AND want_remain_ring == FALSE
```

아무 행동도 취하지 않는다

14. CLAIM_TOKEN | Reinit_init_complete | USE_TOKEN

논리적 링의 재초기화 완료나 초기화 완료시

```
FDL_SET_VALUE.confirm
AND Rx_status.DA != TS
```

아무 행동도 취하지 않는다.

---

15. CLAIM_TOKEN | NS_equ_TS | PASS_TOKEN

초기화 시, 다음 스테이션의 주소가 현 스테이션의 주소와 같은 경우

```
NS == TS
```

아무 행동도 취하지 않는다.

---

16. USE_TOKEN | Transmit_action_frame | AWAIT_DATA_RESPONSE

토큰 사용 가능시간이 유효할 경우, 동작 프레임을 전송한다.
그리고, 슬롯 타이머를 동작시킨다.

```
THT > 0

        if Service_type = SDA || SDN
        then
                send_data_frame( SSAP = SSAP,
                                 DSAP = DSAP,
                                 Loc_add = Loc_add,
                                 Rem_add = Rem_add,
                                 L_sdu = L_sdu,
                                 Serv_class = Serv_class);

        else if Service_type = SRD
        then
                send_data_frame( SSAP = SSAP,
                                 DSAP = DSAP,
                                 Loc_add = Loc_add,
                                 Rem_add = Rem_add,
                                 L_sdu = L_sdu,
                                 Serv_class = Serv_class
                                 Update_status = update_status);

        start_slot_timer();
```

---

17. USE_TOKEN Have_no_message CHECK_ACCESS_TIME

수행될 높은 순위의 메시지 싸이클이 없을 경우

```
h_msg == 0
```

아무 행동도 취하지 않는다.

---

18. USE_TOKEN Message_cycle_complete CHECK_ACCESS_TIME

메시지 싸이클을 완료한 경우

```
msg_complete == YES
```

아무 행동도 취하지 않는다.

---

19. AWAIT_DATA_RESPONSE Wait_ack_resp AWAIT_DATA_RESPONSE

확인이나 응답 프레임의 수신에 대해 한 슬롯 시간동안 대기한다.

```
Rx_status.new == FALSE
```

아무 행동도 취하지 않는다.

---

20. AWAIT_DATA_RESPONSE No_received_frame_after_retry USE_TOKEN

재시도 후, 유효한 확인이나 응답 프레임이 수신되지 않을 경우

```
Rx_status.new == FALSE
AND retry_count == 1
```

아무 행동도 취하지 않는다.

---

21. AWAIT_DATA_RESPONSE Other_valid_frame_receive ACTIVE_IDLE

확인/응답 프레임이 아닌 다른 유효 프레임이 수신될 경우,
수신된 프레임을 버린다.

```
Rx_status.new        == TRUE
AND Rx_status.status == VALID
AND Rx_status.type   != ACK_RESP_FRAME
```

Discard_frame();

---

22. AWAIT_DATA_RESPONSE Ack_resp_frame_receive USE_TOKEN

확인이나 응답 프레임이 수신될 경우

```
Rx_status.new          == TRUE
AND Rx_status.status   == VALID
AND Rx_status.type     == ACK_REAP_FRAME

        아무 행동도 취하지 않는다.
```

---

23. AWAIT_DATA_RESPONSE Invalid_frame_receive AWAIT_DATA_RESPONSE

유효하지 않은 프레임이 수신될 경우, 동작 프레임의 전송을 다시 시도한다.

```
Rx_status.new          == TRUE
AND Rx_status.status   == INVALID
AND retry_count        == 0

        Resend_frame();
```

---

24. CHECK_ACCESS_TIME THT_not_available PASS_TOKEN

토큰 사용 가능시간(THT)이 유효하지 않은 경우

```
THT < 0

        아무 행동도 취하지 않는다.
```

---

25. CHECK_ACCESS_TIME THT_available USE_TOKEN

토큰 사용 가능시(THT)가 유효한 경우

```
THT > 0

        T_RR = T_TR - T_RR;
        timer_set( T_RR );
```

---

26. PASS_TOKEN fatal_error OFFLINE

자신의 토큰 프레임을 수신하지 못할 경우, 전송/수신 채널의 치명적 에러로
보고 이 사실을 관리자(FMA1/2)에게 알린다.

```
Rx_status.SA != TS

        FMA_EVENT.indication( MSAP_1 = MSAP_1,
                              Fault  = FATA_ERROR);
```

---

27. PASS_TOKEN Token_retrans_incorrect LISTEN_TOKEN

만일 토큰 프레임이 재전송된 후 잘못된 것으로 모니터될 경우

```
        retry_count == 1
        AND Token_trans_status == FALSE

                Report_to_FMA( "Token retrans. error" );
```

---

28. PASS_TOKEN receive_own_token_frame_corrupted CHECK_TOKEN_PASS

\# 부적합한 자신의 토큰 프레임이 수신될 경우

```
        Rx_token_frame
        AND Rx_status.status == INVALID
        AND Rx_status.SA     == TS
```

        아무 행동도 취하지 않는다.

---

29. PASS_TOKEN Trans_request_FDL_status AWAIT_STATUS_RESPONSE

\# 새로운 마스터 스테이션이 있고, 논리적 토큰 링에 남아있길 원하며,
\# "Request FDL Status"을 전송한 경우

```
        exist_new_master == YES
        AND want_in_ring == TRUE
```

        아무 행동도 취하지 않는다.

---

30. CHECK_TOKEN_PASS Delay_for_reaction CHECK_TOKEN_PASS

\# 토큰을 통과시킨 스테이션의 재동작에 대해 한 슬롯 시간동안 기다린다.

```
        Rx_status.new == FALSE

                T_SL --;
```

---

31. CHECK_TOKEN_PASS Receive_invalid_frame ACTIVE_IDLE

\# 슬롯 시간안에서 유효하지 않은 프레임을 인식할 경우
\# (다른 스테이션이 활성화되었다고 가정한다.)

```
        Rx_status.new        == TRUE
        AND Rx_status.status == INVALID
```

        아무 행동도 취하지 않는다.

---

32. CHECK_TOKEN_PASS Not_receive_any_frame PASS_TOKEN

\# 한 슬롯 시간안에 어떤 프레임도 수신하지 못한 경우

Rx_status.new == FALSE
AND T_SL == 0

아무 행동도 취하지 않는다.

---

33. AWAIT_STATUS_RESPONSE Wait_ack_frame AWAIT_STATUS_RESPONSE

확인 프레임에 대해 한 슬롯 시간동안 기다린다.

Rx_status.new == FALSE

T_SL --;

---

34. AWAIT_STATUS_RESPONSE Receive_frame_except_ack_frame ACTIVE_IDLE

확인 프레임 대신 어떤 다른 프레임을 수신할 경우
(토큰이 여러개 존재한다는 것을 가리킴)

Rx_status.new == TRUE
AND Rx_status.type != ACK_RESP_FRAME

아무 행동도 취하지 않는다.

---

35. PASS_TOKEN NOT_exist_next_station USE_TOKEN

토큰을 넘기려하지만, 다음 스테이션의 주소가 자신인 경우

FMA1/2_LIVE_LIST.confirm
AND NS == TS

아무 행동도 취하지 않는다.

---

36. AWAIT_STATUS_RESPONSE No_or_corrupted_receive PASS_TOKEN

부적합한 프레임이 도착하거나, 아무 프레임도 수신되지 않을 경우

Rx_status.new == FALSE
OR Rx_status.new == TRUE AND Rx_status.status == INVALID

아무 행동도 취하지 않는다.

7.6. 마스터 스테이션에서의 서비스 흐름

다음의 그림 7-2는 마스터 스테이션에서의 전체 상태머신도이다. 이를 이용하여 각 상태머신별 변환과 상태머신내에서의 변환을 다음과 같이 구성하였다.

1) 상태머신간의 전달

1. 운영 매개변수들의 초기화 완료 (상태천이표 2)
2. 전달된 토큰 프레임의 소스 주소가 자신의 주소인 경우 (상태천이표 7)
3. 올바른 토큰 프레임을 전달받았을 경우 (상태천이표 12, 14)
4. SERVE 상태머신에서 수신된 프레임이 확인/응답 프레임이 아닌 경우 (수신된 프레임을 버림) (상태천이표 21)
5. SERVE 상태머신에서 토큰 사용 가능 시간이 유효하지 않은 경우(상태천이표 4)
6. PASS 상태머신에서 토큰을 다음 스테이션으로 넘기고자 하지만, 다음 스테이션의 주소를 알 수 없는 경우 (현 스테이션만이 논리적 토큰 링에서 활성화된 것을 의미) (상태천이표 35)
7. PASS 상태머신에서
   ① 토큰 프레임 재전송 후, 잘못으로 모니터되는 경우 (상태천이표 27)
   ② 토큰 프레임 전송 후, 슬롯타임안에 유효 프레임이 수신되지 않은 경우 (상태천이표 31)
   ③ 확인 프레임 대신 다른 프레임이 수신된 경우 (상태천이표 34)
8. IDLE 상태머신에서 논리적 링 (재)초기화시 다음 스테이션의 주소가 현 스테이션의 주소와 동일한 경우(상태천이표 15)
9. 전송한 토큰 프레임이 버스상에서 발견되지 않을 경우 (상태천이표 26)

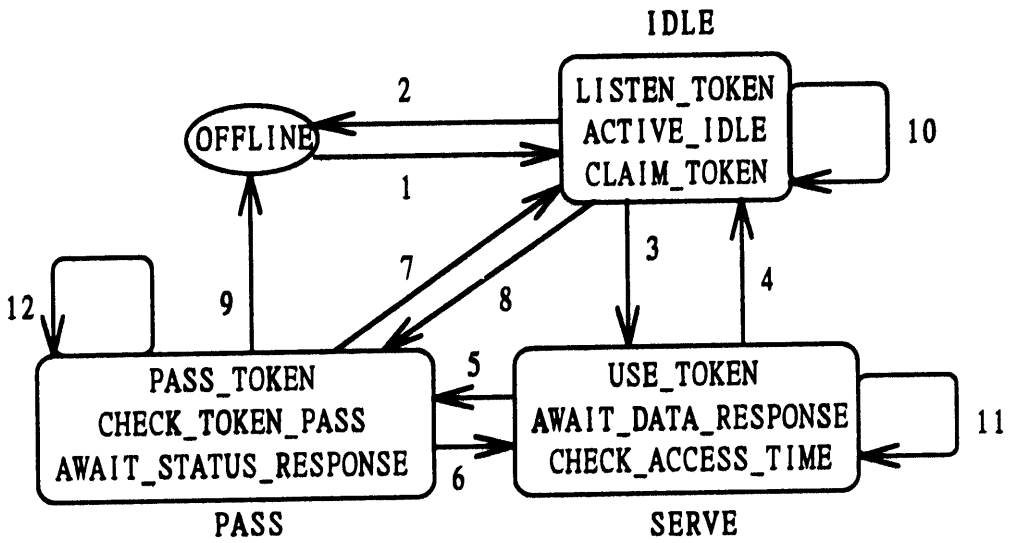
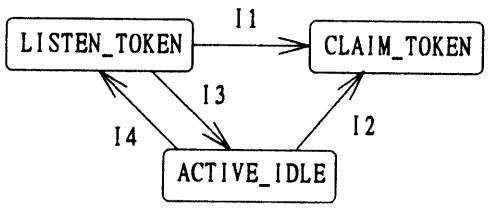
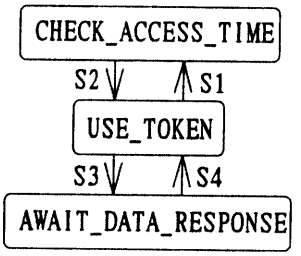


그림 7-2. 마스터 스테이션에서의 전체 상태머신도

IDLE 상태머신에서의 흐름도



SERVE 상태머신에서의 흐름도



PASS 상태머신에서의 흐름도

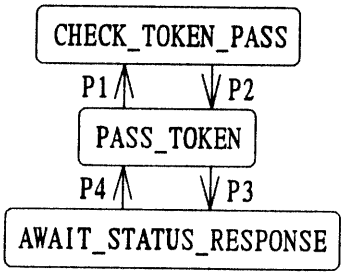


그림 7-3. 각 상태머신내에서의 서비스 흐름도

2) 상태머신 내에서의 전달

IDLE. 자신의 FDL 주소를 GAP List에 첨가하고, 토큰 프레임을 수신할 준비를 함

LISTEN_TOKEN : 이전 스테이션에서 FDL_LIVE_LIST.req가 전달되었을 때, 자신의 FDL address를 Live_List 엔트리에 포함시켜 FDL_LIVE_LIST.con을 전달한다.

ACTIVE_IDLE : 자신이 토큰을 받아야 할 경우, 토큰 프레임을 기다림

CLAIM_TOKEN : 논리적 토큰 링의 오류나 토큰 프레임 전달이 잘못되었을 경우로 논리적 토큰 링을 초기화하거나 복구시킨다.

I1.I2. 타임아웃 주기동안 버스가 활성화되지 않은 경우 (상태천이표 8, 11)

I3. 토큰 프레임이 전달되었고, 논리적 토큰 링에 남아있고자 할 경우 (상태천이표 9)

데이타 영역을 가지지 않는 고정된 길이의 Ack.프레임을 토큰을 보낸 스테이션으로전송한다.

10H	DA	SA	FC	FCS	16H

DA : 전달된 토큰 프레임의 SA(Source Address)

SA : 현 스테이션의 주소

I4. 토큰 프레임이 전달되었고, 논리적 토큰 링에 남아있길 원하지 않는 경우 (상태천이표 13)

SERVE. 토큰을 가져 메시지 처리가 가능한 상태

USE_TOKEN : 메시지 처리가 가능한 상태

① 다른 스테이션의 데이타를 얻고자 할 때(Master or Slave)

FDL_DATA_ACK.req, FDL_DATA.req, FDL_DATA_REPLY.req

② 현 스테이션에서의 데이타를 다른 스테이션으로 보내고자 할 경우

FDL_CYC_DATA_REPLY.req, FDL_SEND_UPDATE.req

CHECK_ACCESS_TIME : 모든 높은 순위의 메시지 처리가 끝난 경우, 토큰 사용 가능시간과 비교한다.

AWAIT_DATA_RESPONSE : 전송한 동작 프레임에 대한 확인/응답을 기다린다.

S1. 높은 순위의 메시지가 한개도 없는 경우(토큰 사용 가능시간이 유효한지 조사) (상태천이표 17)

S2. 토큰 사용 가능시간이 유효한 경우 (상태천이표 25)

S3. 동작 프레임을 전송하고, 전송한 동작 프레임의 응답이나 확인을 기다린다. (상태천이표 16)

S4. 동작 프레임을 재전송한 후, 유효한 응답/확인 프레임이 수신되지 않는 경우 (상태천이표 20)

PASS. 토큰을 다음 스테이션으로 넘기는 상태

PASS_TOKEN : FDL_LIVE_LIST.req를 전송하여 GAP List를 얻는다. 얻은 다음 스테이션의 주소에 토큰 프레임을 전달하고, 확인을 받는다.

CHECK_TOKEN_PASS : 전송한 토큰 프레임의 상태를 조사한다.

AWAIT_STATUS_RESPONSE : FDL_LIVE_LIST.req에 대한 응답(확인)을 기다린다.

P1. 토큰 프레임을 보낸 후, 버스상에 있는 토큰 프레임이 부적합한 자신의 토큰 프레임인 경우 (상태천이표 28)

P2. 토큰 프레임 전송후 한 슬롯타임안에 어떤 프레임도 수신하지 못한 경우 (상태천이표 32)

P3. 논리적 토큰 링에 참여하길 바라는 새로운 마스터 스테이션이 있는 경우 (상태천이표 29)

P4. 한 슬롯타임 동안 부적합한 프레임을 수신하거나, 수신된 프레임이 없는 경우 (상태천이표 36)

7.7 슬레이브 스테이션에서의 서비스흐름

PASSIVE_IDLE : 마스터 스테이션에서 전달되는 동작프레임에 대한 확인/응답만을 수행한다.

1. OFFLINE에서 운영변수들의 초기화가 완료되고, 스테이션이 SLAVE인 경우 (상태천이표 1)
2. 라인을 조사하여, 수신된 동작프레임에 대한 확인/응답을 함 (상태천이표 3)
3. 사용자로부터 RESET 명령이 전달될 경우, OFFLINE 상태로 변환 (상태천이표 4)
4. 치명적 에러발생시 OFFLINE 상태로 변환 (상태천이표 5)

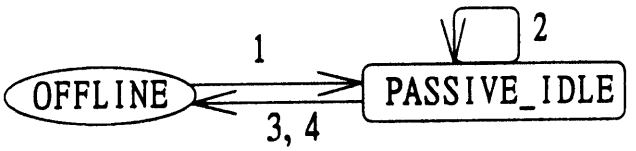


그림 7-4. 슬레이브 스테이션에서의 상태머신도

8. 결 론

본 논문에서는 필드버스 시스템의 데이타링크층을 모델링하고, 각 메시지들이 임의로 발생 및 수행된다는 가정하에서 성능해석을 해보았다. 확률(Stochastic) 페트리네트를 이용하여 필드버스 시스템 중 CiT(Circulated Token)가 서비스하는 부분에 대한 모델링을 하였다. 제시한 페트리네트 모델에 모멘트 발생함수를 이용한 성능해석 방법을 적용시켜 수학적 형태의 성능평가를 하였다.구성한 모델에 XOR GERT (eXclusive-OR Graphical Evaluation and Review Technique) 네트웍에 대한 정리를 적용시켜, 모델의 각 상태들을 간단한 형태로 변환하였다. 그리고, 여기에 모멘트 발생함수를 이용하여 변형된 상태를 해석하였다. 단, 안정상태하에서 각 스테이션들이 동일한 형태를 갖는다는 가정하에서 한 스테이션에서의 평균 서비스 시간과 평균 토큰 회전시간을 구하였다. 모델에서 각 상태 변환인 트랜지션 점화를 각기 전달함수로 바꾸고, 메이슨 정리를 이용하여 이 전달함수들을 간략화한다. 간략화는 그 상태변환시 전달함수를 이용하여 모멘트 발생함수를 구한다. 전달함수를 그 전달함수의 초기치로 나누면, 그것이 그 전달함수에 대한 모멘트 발생함수가 된다. 그 모멘트 발생함수에 상태변환시의 확률을 곱하면 축소된 상태에 대한 전달함수를 얻을 수 있다. 이러한 간략화를 반복하여 전체 시스템에 대한 전달함수를 구한다. 구한 모멘트 발생함수를 일차미분하여 그 초기치를 구한 것이 그 변환시의 평균 지연 시간이 된다. 이러한 방법으로 상태도를 하나의 전달함수로 변환시키고, 그 전달함수에서 모멘트 발생함수를 얻어 한 스테이션에서의 지연시간을 얻는다. 이 지연시간이 한 스테이션에서의 평균 서비스 시간이 된다. 앞서 가정했듯이 안정상태 조건을 첨가시키면, 이 한 스테이션에서의 평균 서비스 시간은 각 스테이션이 동일한 값을 갖기 때문에 평균 토큰 회전시간을 구할 수 있다. 구한 한 스테이션에서의 평균 서비스 시간과 평균 토큰 회전시간은 이미 알고 있고 부여한 값들로 수학적 형태의 표현이 가능하다. 표현된 수학적 형태는 시스템에서의 스테이션의 갯수, 오버헤드시간, 메시지 발생율, 평균 메시지 처리시간, 토큰 사용 가능시간등으로 나타내진다. 이러한 방법으로 필드버스 시스템에 대한 수학적 성능해석을 해보았으며, 얻은 수식에 따라 각 상황별(발생율을 변화시킨 경우, 메시지 처리시간을 변화시킨 경우, 토큰 사용 가능시간을 변화시킨 경우등) 서

비스 시간들을 모의실험의 결과와 비교해 보았다. 통화량이 200개/초 이하의 경우에서는 거의 오차가 발견되지 않아, 구한 수학적 형태가 바르다고 볼 수 있다. 그러므로, 이 결과는 다른 연구들에서도 사용되는 발생율 범위에서의 수학적 형태가 된다는 것을 의미한다. 그러나, 통화량이 200개/초 보다 큰 경우에서는 약간 큰 오차가 발생되었다. 이것은 제시한 모델이 다른 시스템들에서와 같이 한계가 있어, 그 통화량이나 메시지 처리시간등에 따른 다른 여러 환경들의 조절이 필요하다는 것을 보여준다. 그리고, 이렇듯 살펴본 필드버스 시스템의 데이타링크층을 설계해 보았다. 참조한 표준안에 따라 각 상태별 처리과정과 서비스 프리미티브를 살펴보았으며, 그것들에 따른 상태천이표를 작성하였다. 그리고, 작성한 상태천이표들에 입각하여 세가지 상태머신을 구성하고, 각 상태머신들을 설계하였다.

본 논문에서는 필드버스 시스템에서 사용되는 토큰들 중 한가지에 대해 연구하였지만, 앞으로 모든 종류의 토큰을 고려한 연구가 있어야 한다고 본다. 특히, 긴급 메시지의 처리가 통신망의 안정성에 많은 영향을 끼치므로 DeT(Delegated Token)의 비주기적 서비스에 대한 연구가 필요하다고 본다. 그리고, 앞서 언급한 것처럼 각 매개변수로써 사용되는 요소값들(메시지 발생율, 메시지 처리시간, 토큰 사용 가능시간등)의 배분에 대한 연구가 필요하다고 본다. 그리고, 이론적 연구만이 아닌 제품으로써의 구현이 빨리 이루어져야 한다고 본다.

참 고 문 헌

[1] General Motors, Manufacturing Protocol Specification Version 3.0, 1988

[2] Industrial Automation Systems - Systems Integraion and Communications - FieldBus Part 4: Data Link Protocol Specification Editor's Draft - Version 5 ISA/SP50-1991-360E

[3] Industrial Automation Systems - Systems Integraion and Communications - FieldBus Part 3: Data Link Service Definition Editor's Draft - Version 8 ISA/SP50-1991-359H

[4] Antonella Di Stefano and Orazio Mirabella, "Evaluating the FieldBus Data Link Layer by a Petri Net-Based Simulation", IEEE Tr. on Industrial Electronics, Vol.38, No.4, August 1991, pp.288-297

[5] Salvatore Cavalieri, Antonella Di Stefano, and Orazio Mirabella, "Optimization of Acyclic Bandwidth Allocation Exploiting the Priority Mechanism in the FieldBus Data Link Layer", IEEE Tr. on Industrial Electronics, Vol 40, No 3, June 1993, pp.297-306

[6] Abd E.Elnakhai and Helmut Rzehak, "Design and Performance Evaluation of Real Time Communication Architectures", IEEE Tr. on Industrial Electronics, Vol.40, No.4, AUGUST 1993, pp.404-411

[7] ADRIANO VALENZANO and LUIGI CIMINIERA, "Performance Ebaluation of MiniMAP Networks", IEEE Tr. on Industrial Electronics, Vol.37, No.3, JUNE 1990, pp.253-258

[8] PAOLO MONTUSHI, ADRIANO VALENZANO, LUIGI CIMIERA, "Selection of Token Holding Times in Timed-Token Protocol", IEEE Tr. on Industrial Electronics, Vol.37, No.6, DECEMEMBER 1990, pp.442-451

[9] On-Ching, Charles A. Brooks, "Performance of the Timed Token Scheme in MAP", IEEE Trans. on Communication, Vol.38, No.7, July, 1990, pp.1006-1012

[10] D.L. Guo, F. DiCesare, and M.C. Zhou," A Moment Generating Function Based Approach for Evaluating Extended Stochastic Petri Nets," IEEE Tr. on AC.,Vol. 38, No.2, 1993, pp.321-327

[10] J.D.Decotignie, P.Raja, "Fulfilling Temporal Constraints in FieldBus", Proc. of IECON'93, Maui, Hawai, Nov.15-19, 1993, pp.519-529

[11] DIN 19 245 Profibus Standard Part 1, 1993

[12] Field Bus Standard for Use in Industrial Control Systems Part 2: Physical Layer Specification and Service Definitions ISA/SP50-1992-236P

[13] P.Raja, K.Vijayanada and J.D.Decotignie, "Polling Algorithm and their Properties for FieldBus Networks", Proc. of IECON'93, Maui, Hawai, Nov.15-19, 1993, pp.530-534

[14] Gonzalo Ulloa, "FieldBus Application Layer and Real-Time Distributed Systems" Proc. of IECON'91, 1991, pp.1679-1683

[15] T.Murata, "Petri-Nets: Properties, Analysis and Applications", Proc. of the IEEE, Vol.77, No.4, 1989, pp.541-579

[16] N.Viswanadham, Y.Narahari, *Performance Modeling of Automated Manufacturing Systems*, Prentice-Hall, 1992

[17] Peter G. Harrison, Naresh M. Patel, *Performance Modeling of Communication Networks and Computer Architectures*, Addison-Wesley, 1993

[18] Stewart V. Hoover, Ronald F. Perry, *Simulation: A Problem-Solving Approach*, Addison-Wesley, 1990

[19] Athanasios Papoulis, *Probability, Random Variables and Stochastic Processes: Third Edition*, McGraw-Hill Inc. 1991.

[20] 문상용, 박홍성, 권욱현, "페트리네트를 이용한 IEEE 802.4 토큰 버스 규약의 성능 해석", 94' 한국 자동제어 학술회의 논문집, 1994.10.17, pp.661-665

[21] 김정호, 정태진, "산업 공정 환경에서 Mini MAP을 기준으로 한 실시간 네트워크의 성능해석", '91 한국 자동제어 학술회의 논문집, 1991.10.22-24, pp.342-346

[22] 김현기, 이전우, 황선호, 이혁희, 채영도, "Fieldbus 네트워크 접속기의 하위계층 구현", '91 한국 자동제어 학술회의 논문집, 1991.10.22-24, pp.337-341

[23] 홍지민, 이기동, 이범희, "일반화된 확률 페트리네트의 축소에 관한 연구", '93 한국 자동제어 학술회의 논문집, 1993.10.20-22, pp.324-329

[24] 심광현, 임종태, "우선순위 및 제한 서어비스를 갖는 대규모 토큰-패싱 네트워크의 점근적 성능해석 및 적용제어", '93 한국 자동제어 학술회의 논문집, 1993.10.20-22. pp.1000-1005

[25] 김덕우, 정범진, 추영열, 권욱현, "실시간 응용시 Mini MAP의 시뮬레이션에의한 성능해석에 관한 연구", '87 한국 자동제어 학술회의 논문집, 1987.10.16-17, pp.382-387

```
/*
   제     목: 필드버스 데이타링크층 함수 설계
   화 일 명: FDL.C
   목     적: 필드버스 데이타링크층의 소프트웨어적 구현
   사용언어: TC 2.0
   주     의: 1.인터럽트 61h에 여기서 사용되는 전역변수가 저장되어야 함
                LOADFDL.EXE 프로그램을 이용하여 램상주시킴.
                값을 바꾸고자 할 때는 FDL_SET.CFG를 custom.exe를 이용
                램상주된 형태는 OP_param이라는 구조체로 구성됨
                메모리상에 존재하지 않을 경우는 기본값을 사용한다.
              2.사용되는 타이머들은 전송부의 물리층에서 감소된다.
                타이머 활성화여부와 값을 세트하는 것은 여기서 담당
                이 타이머 값들은 물리층에서 계속적으로 감소한다.
              3.수신버퍼는 버스상의 모든 프레임들을 수신한다.
                수신된 프레임이 자신의 주소를 갖는 경우는 저장한다.
                그리고, 자신이 전송한 토큰 프레임인 경우도 조사한다.
                이 경우는 PASS_TOKEN에서 버스상에 자신이 보낸 토큰이 존재하는 지
                조사한다.
   작 성 일: 1995. 1. 2.
   작 성 자: 이 재 수
*/

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

/********** 형 선언 ************/
#define  BOOLEAN unsigned char
#define  BYTE    unsigned char
#define  WORD    unsigned int
#define  ULONG   unsigned long

/************  리턴될 상태 값에 대한 선언 ***************/
#define  YES     0x01
#define  NO      0x00
#define  TRUE    0x01
#define  FALSE   0x00
#define  OK      0x01

/************ 상태 종류 선언 ***************/
#define  OFFLINE                 0x00
#define  LISTEN_TOKEN            0x01
#define  ACTIVE_IDLE             0x02
#define  CLAIM_TOKEN             0x03
#define  USE_TOKEN               0x04
#define  AWAIT_DATA_RESPONSE     0x05
#define  CHECK_ACCESS_TIME       0x06
#define  PASS_TOKEN              0x07
#define  CHECK_TOKEN_PASS        0x08
```

```
#define  AWAIT_STATUS_RESPONSE  0x09
#define  PASSIVE_IDLE           0x0a

/************ SD의 종류에 따른 선언 ****************/
#define SD1  0x10     /* 데이타 영역을 갖지 않는 고정된 길이의 프레임 */
#define SD2  0x68     /* 가변적인 데이타 영역을 가진 프레임           */
#define SD3  0xa2     /* 고정된 길이의 데이타 영역을 가지는 프레임    */
#define SD4  0xdc     /* 토큰 프레임                                  */

/******* 수신된 프레임의 유효 여부 선언 **************/
#define  VALID    0x01  /* 프레임 유효              */
#define  INVALID  0x02  /* 프레임이 유효하지 않음   */

/******* BYTE Service_type 에 대한 서비스 형태 선언 ***************/
#define  SDA  0x00    /* Send Data with Ack.                        */
#define  SDN  0x01    /* Send Data with No ack.                     */
#define  SRD  0x03    /* Send and Request Data with reply           */
#define  CSRD 0x05    /* Cyclic Send and Request Data with reply    */

/******* BAUD rate 선언 *************/
#define B_96    0x00
#define B_192   0x01
#define B_9375  0x02
#define B_1875  0x03
#define B_500   0x04

/******* Medium 선언 ***************/
#define SINGLE      0x00
#define REDUNDANT   0x01

/************ 스테이션의 종류 선언 ****************/
#define  SLAVE   0x01  /* slave 스테이션          */
#define  MASTER  0x02  /* master 스테이션         */

/************ 프레임에 대한 선언 ******************/
#define BUF_NUM          8       /* 수신 버퍼의 갯수              */
#define REQ_FRAME        0xff    /* FC 생성시 AND 시키기 위한 값 */
#define ACK_RESP_FRAME   0xbf

/************ 타이머 세트 ****************/
#define T_SLOT_TIME     500
#define T_TIME_OUT      133000 /* slot time이 500일 때 해당하는 값 */
#define T_OUT           0x01
#define T_SLOT          0x02

/************ 수신 프레임의 형태 **************/
struct  _Rx_FRAME {
    BYTE SYN[33];      /* synchronization period, minimum 33 bit    */
    BYTE SD;           /* start delimiter                           */
```

```
    BYTE DA;            /* destination address                          */
    BYTE SA;            /* source address                               */
    BYTE FC;            /* frame control                                */
    BYTE FCS;           /* frame check sequence                         */
    BYTE LE;            /* octet length                                 */
    BYTE LEr;           /* octet length repeated                        */
    BYTE ED;            /* end delimiter, value = 0x16                  */
    BYTE L;             /* information field length                     */
    BYTE SC;            /* single character, value = 0xE5               */
    ULONG *DATA_UNIT;   /* data field                                   */
};

/************ 수신 프레임의 상태 저장 ***************/
struct  _Rx_status {
    BYTE SD;          /* 전송된 프레임의 종류 판별                                  */
    BYTE DA;          /* Destination address, FRAME.DA가 저장됨                     */
    BYTE SA;          /* Source address, FRAME.SA가 저장됨                          */
    BYTE status;      /* 수신된 프레임의 유효 여부를 저장, VALID/INVALID            */
                      /* FRAME.FCS를 이용하여 판별함                                */
    BYTE type;        /* FRAME.FC의 b7이 1이면 REQ_FRAME, 0일 경우 ACK_RESP_FRAME   */
    BOOLEAN new;      /* 프레임이 수신될 경우 TRUE로 세트됨                         */
};

/************* 전송 프레임의 형태 **************/
struct _Tx_Token {
    BYTE SD;
    BYTE DA;
    BYTE SA;
};

struct _Tx_Req_SD1 {
    BYTE SD;            /* start delimiter                              */
    BYTE DA;            /* destination address                          */
    BYTE SA;            /* source address                               */
    BYTE FC;            /* frame control                                */
    BYTE FCS;           /* frame check sequence                         */
    BYTE ED;            /* end delimiter, value = 0x16                  */
};

struct _Tx_Ack_SD1 {
    BYTE SD;            /* start delimiter                              */
    BYTE DA;            /* destination address                          */
    BYTE SA;            /* source address                               */
    BYTE FC;            /* frame control                                */
    BYTE FCS;           /* frame check sequence                         */
    BYTE ED;            /* end delimiter, value = 0x16                  */
};

struct _Tx_SC {
```

```
    BYTE SC;
};

struct _Tx_Req_SD3 {
    BYTE SD;            /* start delimiter                    */
    BYTE DA;            /* destination address                */
    BYTE SA;            /* source address                     */
    BYTE FC;            /* frame control                      */
    BYTE DATA_UNIT[8];  /* data field                         */
    BYTE FCS;           /* frame check sequence               */
    BYTE ED;            /* end delimiter, value = 0x16        */
};

struct _Tx_Res_SD3 {
    BYTE SD;            /* start delimiter                    */
    BYTE DA;            /* destination address                */
    BYTE SA;            /* source address                     */
    BYTE FC;            /* frame control                      */
    BYTE DATA_UNIT[8];  /* data field                         */
    BYTE FCS;           /* frame check sequence               */
    BYTE ED;            /* end delimiter, value = 0x16        */
};

struct _Tx_Req_SD2 {
    BYTE SD;            /* start delimiter                    */
    BYTE LE;            /* octet length                       */
    BYTE LEr;           /* octet length repeated              */
    BYTE SDD;
    BYTE DA;            /* destination address                */
    BYTE SA;            /* source address                     */
    BYTE FC;            /* frame control                      */
    ULONG *DATA_UNIT;   /* data field                         */
    BYTE FCS;           /* frame check sequence               */
    BYTE ED;            /* end delimiter, value = 0x16        */
};

struct _Tx_Res_SD2 {
    BYTE SD;            /* start delimiter                    */
    BYTE LE;            /* octet length                       */
    BYTE LEr;           /* octet length repeated              */
    BYTE SDD;
    BYTE DA;            /* destination address                */
    BYTE SA;            /* source address                     */
    BYTE FC;            /* frame control                      */
    ULONG *DATA_UNIT;   /* data field                         */
    BYTE FCS;           /* frame check sequence               */
    BYTE ED;            /* end delimiter, value = 0x16        */
};
/************* 운영 매개변수 설정 ***************/
```

```
struct  _OP_param {
    BYTE  TS;          /* Station Address(0-126)                          */
    BYTE  Baud_rate;   /* baud rate(9.6, 19.2, 93.75, 187.5, 500 kBPS) */
    BYTE  Medium_red;  /*single/redundant                                 */
    BYTE  HW_Release;   /* Hardware release                               */
    BYTE  SW_Release;   /* Software release                               */
    WORD  T_SL;         /* Slot time (0-65535)                            */
    WORD  MIN_T_SDR;    /* minimum T_SDR (0-65535), Station Delay Time */
    WORD  MAX_T_SDR;    /* maximum T_SDR (0-65535), Station Delay Time */
    BYTE  T_QUI;        /* Quiet Time (0-255)                             */
    BYTE  T_SET;        /* Setup Time (0-255)                             */
    BYTE  T_TR;         /* Target Rotation Time (0-255)                   */
    BYTE  G;            /* GAP Update Factor (1-100)                      */
    BOOLEAN in_ring;    /* Request entry into or exit out of the logical
                           token ring, TRUE/FALSE                         */
    BYTE  MAX_retry;    /* Maximum Number of retries (1-8)                */
    BYTE  Station_type;/* FRAME.FC의 b6와 b5가 모두 0인 경우 SLAVE로 세트
                          그렇지 않은 경우는 MASTER로 세트                */
    BYTE  c_magic;      /* 구조체의 식별자  */
};

/****************** 변수 정의 ***********************/
struct _OP_var {
   BYTE  Service_type;          /* 현재 요구에 대한 서비스 형태        */
   BYTE  Fatal_error;           /* 치명적 에러                         */
   BOOLEAN Rx_token_frame;      /* 전달된 프레임이 토큰 프레임일 경우 세트됨 */
   BOOLEAN Token_trans_status; /* 토큰 전달이 제대로 실행되지 않았 경우 FAIL로 세트
                                  토큰 전달시 confirm에 의해 세트됨   */
   BOOLEAN want_in_ring;        /* 논리적 토큰 링에 들어가길 원할 때 TRUE로 세트됨
                                  FRAME.FC의 b6가 1이고 b5가 0인 경우 TRUE  */
   BOOLEAN in_ring;             /* 현재 논리적 토큰 링에 있는 지를 알려줌
                                  FRAME.FC의 b6와 b5가 모두 1인 경우 TRUE   */
   BOOLEAN  want_remain_ring;  /* 현 스테이션이 논리적 토큰 링에 계속남아있길
                                  원하는 경우 TRUE로 세트됨                */
   BYTE  h_msg;                 /* High-priority message의 갯수 */
   BYTE  l_msg;                 /* Low-priority message의 갯수 */
   BOOLEAN msg_complete;        /* 한 개 메시지 처리 완료시마다 YES로 세트됨,
                                  새로운 메시지처리시 NO 로 세트           */
   BYTE  retry_count;           /* 재전송 횟수를 저장(0 에서 8까지 가능)     */
   BOOLEAN exist_new_master;    /* 새 마스터 스테이션이 존재할 경우 YES로 세트 */
   BYTE  NS;                    /* 토큰을 넘겨야하는 다음 스테이션의 주소
                          LIVE_LIST에서 다음 스테이션의 주소를 얻을 수 있다.    */
   WORD  T_TH;                  /* 토큰 사용 가능 시간을 나타낸다.           */
   WORD  T_SL;                  /* 현재 슬롯 시간을 나타낸다.                */
   BYTE  Station_num;           /* 현재 GAP list에 있는 스테이션의 총 수      */
};


/***************** 함수들에서 사용되는 전역 변수들  **************/
```

```
struct  _Tx_Token    Tx_Token;        /* 전송 버퍼들          */
struct  _Tx_Req_SD1  Tx_Req_SD1;
struct  _Tx_Ack_SD1  Tx_Ack_SD1;
struct  _Tx_SC           Tx_SC;
struct  _Tx_Req_SD3  Tx_Req_SD3;
struct  _Tx_Res_SD3  Tx_Res_SD3;
struct  _Tx_Req_SD2  Tx_Req_SD2;
struct  _Tx_Res_SD2  Tx_Res_SD2;
struct  _Rx_FRAME    Rx_F[BUF_NUM];  /* 수신 버퍼            */
struct  _Rx_status   Rx_status;      /* 현재 수신 상태       */
struct  _OP_param    OP_param;       /* 운영 매개변수        */
struct  _OP_var      OP_var;
BYTE    current_state = OFFLINE;    /* 현재 상태를 알려줌     */
int     Station_type = MASTER;      /* 현 스테이션의 형태를 저장 */
int     TIME_OUT = 0;               /* 각 타이머들이 만료될 때 세트됨 */
int     RESET=0;                    /* RESET 요구가 있을 때 세트됨 */
int     T_TO_s = 0, T_SL_s = 0;
WORD    T_TH,T_TR, T_RR;

/*
 함 수 명 : Trans_state()
 기    능 : 현재 상태를 인수에 따라 변환한다.
            DEBUG 모드에서는 현재 상태와 발생된 사건의 종류를 디스플레이
 전달인자 : bST ( 다음 상태 ),  event_num ( 발생된 사건 번호 )
*/
void Trans_state( BYTE bST, BYTE event_num )
{
    current_state = bST;
}

/*
  함 수 명 : Timer_set ()
  기    능 : 지정된 타이머에 값을 세트
  전달인자 : timer_kind(타이머 종류), value(세트할 값)
*/
void Timer_set( BYTE timer_kind, WORD value )
{
    TIME_OUT = 0;        /* time-out timer를 clear */
/*    지정된 타이머에 값을 세트함    */
}

/*
  함 수 명 : Timer_read ()
  기    능 : 지정된 타이머의 값을 읽어들임
  전달인자 : timer_kind(타이머 종류)
  리 턴 값 : 타이머의 값
*/
WORD Timer_read( BYTE timer_kind )
{
```

```
/*  지정된 타이머에서 값을 읽어들임  */
}

/*
  함 수 명 : Discard_frame()
  기    능 : 수신된 프레임을 버린다.
*/
void Discard_frame( )
{
    Rx_status.new = FALSE;
}

/*
  함 수 명 : Report_to_FMA()
  기    능 : FDL 관리자에게 현재 상태를 알려줌
  전달인자 : sTransStatus ( 전달하고자하는 상태 )
  리 턴 값 : 없음
*/
void Report_to_FMA( char *sTransStatus )
{
    printf( "\n REPORT: %s", sTransStatus );
}

/*
  함 수 명 : Variable_init()
  기    능 : 변수들의 초기화
*/
void Variable_init()
{
    OP_var.Fatal_error      = 0;
    OP_var.Rx_token_frame   = FALSE;
    OP_var.Token_trans_status = OK;
    OP_var.want_in_ring     = TRUE;
    OP_var.in_ring          = TRUE;
    OP_var.want_remain_ring= TRUE;
    OP_var.h_msg            = 0;
    OP_var.l_msg            = 0;
    OP_var.msg_complete     = NO;
    OP_var.retry_count      = 0;
    OP_var.exist_new_master= NO;
    OP_var.Station_num      = 1;
}
/*
  함 수 명 : Read_value()
  기    능 : 운영 매개변수들을 화일에서 읽어서 세트한다.
*/
void Read_value()
{
    struct _OP_param far *OP_temp;
```

```
    long far *para_pos;

    para_pos = (long far *)0x184;        /* int 61h의 인터럽트벡터 */
    OP_temp = *para_pos;

    if( OP_temp->c_magic == 0x17 )       /* 사용될 수있는 값인지 판별 */
              OP_param = *OP_temp;
    else{
         OP_param.TS = 0;                /* Station Address(0-126)                    */
         OP_param.Baud_rate = B_96;      /* baud rate(9.6,19.2,93.75,187.5,500 kBPS)*/
         OP_param.Medium_red= SINGLE;    /* single/redundant                          */
         OP_param.HW_Release= 0;         /* Hardware release                          */
         OP_param.SW_Release= 0;         /* Software release                          */
         OP_param.T_SL      = 1000;      /* Slot time (0-65535)                       */
         OP_param.MIN_T_SDR = 1000;      /* minimum T_SDR (0-65535)                   */
         OP_param.MAX_T_SDR = 6000;      /* maximum T_SDR (0-65535)                   */
         OP_param.T_QUI     = 50;        /* Quiet Time (0-255)                        */
         OP_param.T_SET     = 60;        /* Setup Time (0-255)                        */
         OP_param.T_TR      = 100;       /* Target Rotation Time (0-255)              */
         OP_param.G         = 1;         /* GAP Update Factor (1-100)                 */
         OP_param.in_ring   = TRUE;
         OP_param.MAX_retry = 2;         /* Maximum Number of retries (1-8)           */
         OP_param.Station_type = MASTER;
    }
}


/*
  함 수 명 : OP_LISTEN_TOKEN()
  기    능 : LISTEN_TOKEN 상태에서의 동작을 담당
*/
void OP_LISTEN_TOKEN()
{
    if( T_TO_s == TRUE ){              /* time-out 타이머가 active상태 */
        if( !TIME_OUT ){      /* time-out 타이머가 만료되지 않은 경우 */
            if( Rx_status.new == FALSE )
                return;       /* 수신된 프레임이 없는 경우 */
            else
            if( Rx_status.type == REQ_FRAME && Rx_status.SA != OP_param.TS ){
               /* FDL_LIVE_LIST.req가 도착한 경우 */
                FDL_LIVE_LIST_con( Rx_status.SA );
                T_TO_s = FALSE;
                Rx_status.new = FALSE;
                Trans_state( ACTIVE_IDLE, 9 );
                return;
            }else
                Rx_status.new = FALSE;
        } else {
            T_TO_s = FALSE;
```

```
            Trans_state( CLAIM_TOKEN, 8 );
        }
    }else{
        T_TO_s = TRUE;
        Timer_set( T_OUT, T_TIME_OUT );     /* time-out timer 세트 */
    }
}

/*
  함 수 명 : OP_ACTIVE_IDLE()
  기    능 : ACTIVE_IDLE 상태에서의 동작
*/
void OP_ACTIVE_IDLE()
{
    if( T_TO_s == TRUE ){           /* time-out 타이머가 active상태 */
        if( !TIME_OUT ){     /* time-out 타이머가 만료되지 않은 경우 */
            if( Rx_status.new == FALSE )
                return;      /* 수신된 프레임이 없는 경우 */
            else if( OP_var.Rx_token_frame == TRUE && Rx_status.SA != OP_param.TS )
            {       /* token frame이 도착한 경우 */
                Tx_Ack_SD1.DA = Rx_status.SA;
                Tx_Ack_SD1.FC = 0x39; /* 00111001B */
                Tx_ACK_SD1_F();

                OP_var.Rx_token_frame = FALSE;
                Rx_status.new = FALSE;
                T_TO_s = FALSE;
                Trans_state( USE_TOKEN, 12 );
                return;
            }
            else
                Rx_status.new = FALSE;
        } else{
            T_TO_s = FALSE;
            Trans_state( CLAIM_TOKEN, 11 );
        }
    }else {
        T_TO_s = TRUE;
        Timer_set( T_OUT, T_TIME_OUT );     /* time-out timer 세트 */
    }
}

/*
  함 수 명 : OP_CLAIM_TOKEN()
  기    능 : ACTIVE_IDLE 상태에서의 동작
*/
void OP_CLAIM_TOKEN()
{
    Logical_ring_init();            /* 논리적 토큰 링을 초기화 */
```

```
    if( OP_var.NS == OP_param.TS )  /* 다음 스테이션의 주소가 자신의 주소인 경우 */
        Trans_state( PASS_TOKEN, 15 );
    else
        Trans_state( USE_TOKEN, 14);
}

/*
  함 수 명 : OP_USE_TOKEN()
  기    능 : USE_TOKEN 상태에서의 동작
*/
void OP_USE_TOKEN()
{
    if( OP_var.msg_complete ){
        OP_var.msg_complete = NO;
        Trans_state( CHECK_ACCESS_TIME, 18 );
        return;
    }

    if( OP_var.h_msg > 0 ){ /* high-priority message가 있는 경우 */
        Msg_process();      /* 한개의 high-priority message 처리 */
        T_SL_s = TRUE;      /* slot timer 세트 */
        Timer_set( T_SLOT, T_SLOT_TIME);
        Trans_state( AWAIT_DATA_RESPONSE, 16 );
        return;
    } else
        Trans_state( CHECK_ACCESS_TIME, 17 );
}

/*
  함 수 명 : OP_AWAIT_DATA_RESPONSE()
  기    능 : AWAIT_DATA_RESPONSE 상태에서의 동작
*/
void OP_AWAIT_DATA_RESPONSE()
{
    if( OP_var.retry_count > 0  && TIME_OUT && Rx_status.new == FALSE ){
        TIME_OUT = FALSE;
        Trans_state( USE_TOKEN, 20 );
        return;
    }
    if( Rx_status.new == FALSE )        return;

    if( Rx_status.new == TRUE && Rx_status.status == VALID
        && Rx_status.type != ACK_RESP_FRAME ){
        Discard_frame();
        Trans_state( ACTIVE_IDLE, 21 );
        return;
    }

    if( Rx_status.new == TRUE && Rx_status.status == VALID
```

```
        && Rx_status.type == ACK_RESP_FRAME ){
        Rx_status.new = FALSE;
        Trans_state( USE_TOKEN, 22 );
        return;
    }

    if( Rx_status.new == TRUE && Rx_status.status == INVALID
        && OP_var.retry_count == 0 ){
        Rx_status.new = FALSE;
        Resend_frame();
        return;
    }
}

/*
  함 수 명 : OP_CHECK_ACCESS_TIME()
  기    능 : CHECK_ACCESS_TIME 상태에서의 동작
*/
void OP_CHECK_ACCESS_TIME()
{
    T_TH = T_TR - T_RR;

    if( T_TH > 0 )
        Trans_state( USE_TOKEN, 25 );
    else
        Trans_state( PASS_TOKEN, 24 );
}

/*
  함 수 명 : OP_PASS_TOKEN()
  기    능 : PASS_TOKEN 상태에서의 동작
*/
void OP_PASS_TOKEN()
{
    if( OP_var.Fatal_error != 0 ){
        OP_var.Fatal_error = 0;
        Report_to_FMA( "FATAL_ERROR" );
        Trans_state( OFFLINE, 26 );
        return;
    }

    if( Rx_status.new == TRUE && OP_var.retry_count > 0
        && OP_var.Token_trans_status == FALSE ){
        Rx_status,new = FALSE;
        OP_var.retry_count = 0;
        Report_to_FMA( "Token retrans. Error!" );
        Trans_state( LISTEN_TOKEN, 27 );
        return;
    }
```

```
    if( OP_var.Rx_token_frame && Rx_status.status == INVALID
        && Rx_status.SA == OP_param.TS ){
        OP_var.Rx_token_frame = FALSE;
        Rx_status.new = FALSE;
        Trans_state( CHECK_TOKEN_PASS, 28 );
        return;
    }

    if( OP_var.exist_new_master == YES && OP_var.want_in_ring == TRUE ){
        FMA_LIVE_LIST_req( );
        Trans_state( AWAIT_STATUS_RESPONSE, 29 );
        return;
    }

    if( OP_var.NS == OP_param.TS ){
        Trans_state( USE_TOKEN, 35 );
        return;
    }
}

/*
  함 수 명 : OP_AWAIT_STATUS_RESPONSE()
  기    능 : AWAIT_STATUS_RESPONSE 상태에서의 동작
*/
void OP_AWAIT_STATUS_RESPONSE()
{
    if( TIME_OUT && Rx_status.new == FALSE ){
        TIME_OUT = FALSE;
        Trans_state( PASS_TOKEN, 36 );
        return;
    }
    else if( Rx_status.new == TRUE && Rx_status.type != ACK_RESP_FRAME ){
        Rx_status.new = FALSE;
        Trans_state( ACTIVE_IDLE, 34 );
        return;
    }
}

/*
  함 수 명 : OP_CHECK_TOKEN_PASS()
  기    능 : CHECK_TOKEN_PASS 상태에서의 동작
*/
void OP_CHECK_TOKEN_PASS()
{
    if( TIME_OUT && Rx_status.new == FALSE ){
        TIME_OUT = FALSE;
        Trans_state( PASS_TOKEN, 32 );
        return;
```

```
    }

    if( Rx_status.new == FALSE )   return;

    if( Rx_status.new == TRUE && Rx_status.status == INVALID ){
        Rx_status.new = FALSE;
        Trans_state( ACTIVE_IDLE, 31 );
        return;
    }
}

/*
  함 수 명 : OP_PASSIVE_IDLE()
  기    능 : SLAVE 스테이션의 상태머신으로서 동작
*/
void OP_PASSIVE_IDLE()
{
    if( Rx_status.new == TRUE && Rx_status.type == REQ_FRAME
        && Rx_status.status == VALID ){
        Rx_status.new = FALSE;
        switch( OP_var.Service_type ){
            case SDA:   FDL_DATA_ACK_ind();
                        break;
            case SDN:   FDL_DATA_ind();
                        break;
            case SRD:   FDL_DATA_REPLY_ind();
                        break;
        }
        return;
    }

    if( RESET ){
        RESET = FALSE;
        Trans_state( OFFLINE, 4 );
        return;
    }

    if( OP_var.Fatal_error ){
        OP_var.Fatal_error = FALSE;
        Trans_state( OFFLINE, 5 );
        return;
     }

}

/***************** 메인 함수 ******************/
void  main( void )
{
    char ch;
```

```
Variable_init();  /* 사용되는 변수들을 초기화시킴                    */
Read_value();     /* 화일에서 운영 매개변수들의 값을 읽어들임 */
Station_type = OP_param.Station_type;

if( Station_type == MASTER )
   Trans_state( LISTEN_TOKEN, 1);  /* 상태를 LISTEN_TOKEN로 변환 */
else
   Trans_state( PASSIVE_IDLE, 2);  /* 상태를 PASSIVE_IDLE로 변환 */

 while( 1 ){
     switch( current_state ){
          case  LISTEN_TOKEN:            OP_LISTEN_TOKEN();
                                         break;
          case  ACTIVE_IDLE:             OP_ACTIVE_IDLE();
                                         break;
          case  CLAIM_TOKEN:             OP_CLAIM_TOKEN();
                                         break;
          case  USE_TOKEN:               OP_USE_TOKEN();
                                         break;
          case  AWAIT_DATA_RESPONSE:     OP_AWAIT_DATA_RESPONSE();
                                         break;
          case  CHECK_ACCESS_TIME:       OP_CHECK_ACCESS_TIME();
                                         break;
          case  PASS_TOKEN:              OP_PASS_TOKEN();
                                         break;
          case  CHECK_TOKEN_PASS:        OP_CHECK_TOKEN_PASS();
                                         break;
          case  AWAIT_STATUS_RESPONSE:   OP_AWAIT_STATUS_RESPONSE();
                                         break;
          case  PASSIVE_IDLE:            OP_PASSIVE_IDLE();
                                         break;
          case  OFFLINE:
                  printf("Do you want to exit?(y/N)");
                  scanf("%c", &ch );
                  if( ch == 'Y' || 'y' )
                      exit(0);
                  else{
                      Variable_init();
                      Read_value();

                      if( Station_type == MASTER )
                          Trans_state( LISTEN_TOKEN, 12 );
                      else
                          Trans_state( PASSIVE_IDLE, 13 );
```

감사의 글

그리길지 않게 느껴진 지난 2년간의 대학원 과정은 나에게 많은 것을 남겨주었다. 대학 4년동안 배우고 공부한 것보다 더 많은 것을 가르쳐 주었며, 생활방법까지도 바꿔주었다. 조금은 힘들게 지낸 이 시간들은 앞으로 내 인생에서 가장 중요한 부분으로 남아있을 것이다.

항상 편하게 생활할 수 있도록 보살펴주신 부모님께 감사드립니다. 집에서의 생활이 편했기에 늘 밝게 행동할 수 있었습니다. 그리고, 사고방식을 보다 프로로써 갖도록 지도해주신 박홍성 교수님께 감사드립니다. 교수님께 배운 지난 2년간은 다른 어떤 것보다 값진 것이였습니다. 또한, 부족한 저에 대해 늘 신경써주신 남부희 교수님, 김형중 교수님, 이정문 교수님, 김기택 교수님, 그리고, 최명환 교수님께 감사드립니다.

필요할 때는 늘 도움을 주었던 친구들, 선배, 후배들에게 고마움을 전합니다. 힘들때마다 받은 도움이 지금 이 순간을 만들 수 있었습니다. 그 동안 저를 도와주신 모든 분들께 다시한번 감사드립니다.