논문 2012-07-06

FlexRay 프로토콜에서 플랫폼 구성 변수의 자동 생성에 관한 연구

(Study on Automatic Generation of Platform Configuration Register in FlexRay Protocol)

양 재 성, 박 지 훈, 이 석*, 이 경 창, 최 광 호
(Jae-Sung Yang, Jee-Hun Park, Suk Lee, Kyung-Chang Lee, Gwangho Choi)

Abstract : Recently, FlexRay was developed to replace controller area network (CAN) protocol in chassis networking systems, to remedy the shortage of transmission capacity and unsatisfactory real-time transmission delay of conventional CAN. FlexRay network systems require correct synchronization and complex scheduling parameters. However, because platform configuration register (PCR) setting and message scheduling is complex and bothersome task, FlexRay is more difficult to implement in applications than CAN protocol. To assist a network designer for implementing FlexRay cluster, this paper presents an analysis of FlexRay platform configuration register and automatic generation program of PCR. To demonstrate the feasibility of the automatic generation program, we evaluated its performance using experimental testbed.

Keywords : In-vehicle network, FlexRay, Platform configuration register, Automatic generation program

Ⅰ. 서론

2000년 이후 세계적으로 개발된 자동차의 샤시 및 바디 도메인 관련 기술은 상당 부분 전자제어장치(Electronic Control Unit, ECU)에 의존하고 있다. 1990년대 전반까지는 주로 ABS(anti-lock brake system), 에어백, 자동변속기와 같은 기본적인 안전 시스템 관련 기술이 주로 개발되었고, 1990년대 후반에는 무선도어잠금장치(keyless entry system), ESC(electronic stability control)와 같은 안전 및 편의 시스템 관련 기술이 주로 개발되었다. 하지만 2000년대 이후 운전자의 안전 및 편의성을 향상시키기 위해서 차량거리제어시스템, 충돌피해경감시스템, 자동주차시스템, 나이트비전, 타이어공기압측정시스템과 같은 적극적인 안전기술이 탑재되고 있다 [1-3]. 이로 인해 2000년대 초 20~30%에 머물던 차량의 전자화 비중은 현재 약 40%에 이를 정도로 증가되었다. 특히, 최근에는 배기가스에 대한 규제와 고유가로 인한 차량의 연비 개선, 모바일 오피스의 구현, 능동형 안전 관련 기술의 적극적인 도입, 운전자의 편의성 향상 기술 등이 상품을 서로 차별화하는 역할을 하면서 차량의 전자화 경향은 더욱 확대되고 있는 추세에 있다 [3].

최근에는 운전자의 안전 운전을 보조하기 위한 ADAS(advanced driver assistance system)와 정보 제공을 위한 IVIS(in-vehicle information system)을 구축하기 위해서, 별도의 ECU와 고성능 센서 등이 추가적으로 탑재되는 추세에 있다 [4]. 실제로 최신의 고급형 자동차에서는 운전자의 편의성 및 안전성 개선을 이유로 많은 수의 ECU가 사용되고 있다. 예를 들어, 토요타의 렉서스와 같은 고급차인 경우 약 100개의 ECU가 사용되며, 현대

* 교신저자(Corresponding Author)
논문접수 : 2011. 09. 07., 채택확정 : 2011. 11. 17.
양재성, 박지훈, 이석 : 부산대학교 기계공학부
이경창 : 부경대학교 제어계측공학과
최광호 : ETRI 대경권연구센터
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2010-0011046)

ISSN : 1975-5066

의 제네시스에는 약 70개의 ECU가 사용되고 있다고 알려져 있다 [5]. 하지만 ECU의 사용이 증가되면서 배선의 길이는 1995년 약 45m에서 2000년 이미 4Km에 이를 정도로 증가하였으며, 이는 배선의 무게 증가에 따른 연비감소와 유지보수의 문제로 이어졌다. 이러한 문제는 지금까지 CAN(Control Area Network) 네트워크를 사용함으로써 비교적 효과적으로 해결되고 있었다. 그러나 ECU의 개수 증가에 따라 CAN 네트워크의 처리 능력이 문제로 대두되었고, 복잡한 기능은 다수의 ECU 간의 협조와 분산이라는 문제를 야기하게 되었다 [6].

그림 1은 최근 들어 개발이 추진되고 있는 지능형 자동차의 샤시 통합 제어시스템(integrated chassis control system)을 도식적으로 나타낸 것이다. 그림 1에서는 나타낸 것과 같이 운전자의 조향 각도를 알 수 있는 조향(steering) ECU와 전후방 충돌감시를 위한 레이더(radar)와 초음파(ultrasonic)의 데이터를 수집하는 ADAS(advanced driver assistance system) ECU로 구성이 되어있다. 각각의 ECU는 담당하는 기능에 따라 센서 및 액추에이터를 제어하고, 그 정보는 네트워크를 통해 다른 ECU에 공유된다. 그림 1에서 보는 것과 같이 각 제어시스템들을 단순히 결합하기보다는 각 시스템간의 관계와 차량에 미치는 영향들을 고려하여, 상호보완적으로 ECU를 결합함으로써 샤시 시스템의 제어 성능을 극대화하는 통합 샤시 통합 제어 ECU가 별도로 탑재되는 형태의 개발이 추진되고 있다 [7-8]. 샤시 통합 제어시스템과 같은 실시간 제어시스템은 제한된 시간 내에 주어진 태스크가 수행되어야 하기 때문에 네트워크의 성능이 차량 전체 시스템의 성능과 밀접한 연관을 가지고 있다. 따라서 최대 1Mbps의 전송속도와 비 확정적인 전송지연에 따른 불확실한 성능을 보이는 CAN 네트워크를 대체하기 위하여 새로운 차량용 프로토콜인 FlexRay가 개발되었다 [9].

FlexRay 네트워크는 CAN의 낮은 전송속도와 비확정적 전송지연 문제를 개선하기 위해 2000년 BMW, Daimler-Chrysler, Motorola, Philips를 주축으로 개발되었다. FlexRay 네트워크는 확정적 전송지연을 보장하는 TDMA(time-division multiple access) 방식과 CAN과 같은 우선순위 전송을 보장하는 FTDMA(flexible TDMA) 방식을 병행하여 사용하며 최대 10Mbps의 전송속도를 제공 한다 [10]. FlexRay 프로토콜은 차세대 차량용 전자시스템(by-wire system)에 최적화되어 개발되었다는 평가를 받으며, 샤시 네트워크에 적용하기 위한 많은 연구가 진행되고 있다 [11]. FlexRay 프로토콜은 네트워크에서 TDMA 방식이 가지는 장점을 효과적으로 사용하기 위해서 면밀한 네트워크 클러스터의 설계와 높은 수준의 ECU간 동기화가 요구된다. 하지만 프로토콜 규격을 분석하여 FlexRay 클러스터에 관한 수많은 네트워크 구성 변수(configuration register)를 계산하는 것은 기존의 CAN 네트워크 설계자에게 매우 어려운 일이며, FlexRay 프로토콜의 적용을 느리게 만드는 주요 원인이 되고 있다.

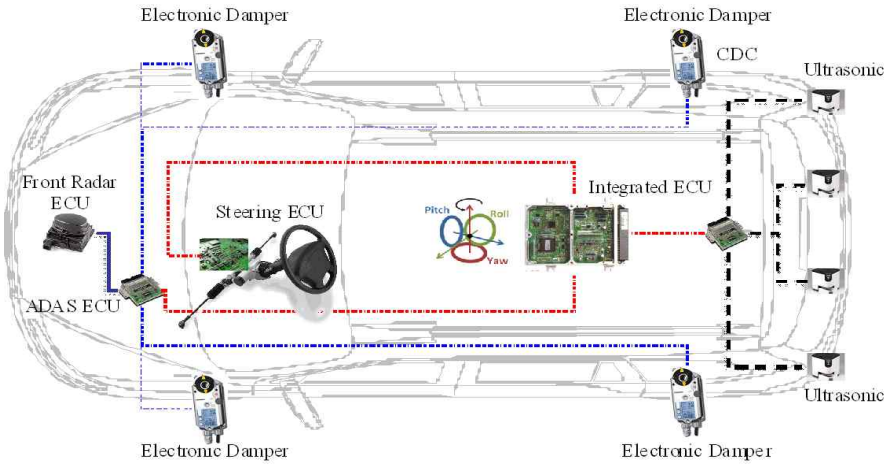

그림 1. 샤시 통합 제어 시스템의 예
Fig. 1 Example of Chassis Integrated Control System

본 논문에서는 자동차 네트워크 설계자가 컨트롤러와 통신모듈로 구성되어 있는 노드들의 집합체인 클러스터를 설계하는 데 도움을 주기 위하여, FlexRay 프로토콜의 규격에 따라 설정해야 하는 53개의 클러스터 구성 변수를 분석하고, 구성 변수를 자동으로 생성할 수 있는 프로그램을 설계한다. 또한, 테스트베드를 구성하여 개발된 FlexRay 클러스터 구성 변수 자동 생성 프로그램의 적용 가능성을 검증한다.

II. FlexRay 프로토콜의 클럭 동기화

FlexRay 프로토콜[12]은 정적 구간(static segment), 동적 구간(dynamic segment), 심볼 윈도우(symbol window), 네트워크 유휴시간(network idle time)을 포함한 4부분으로 구성되어 있다. 정적 구간과 네트워크 유휴시간은 클러스터를 구성할 때 반드시 사용되어야 하며, 동적 구간 및 심볼 윈도우는 설계자의 필요에 따라 선택될 수 있다. 그림 2는 정적 구간과 동적 구간에서의 매체 접근 제어 방식을 나타내고 있다.

정적 구간은 TDMA 방식을 사용하여 데이터를 송수신 한다. 정적 구간은 여러 개의 정적 슬롯(static slot)으로 구성되며, 하나의 정적 슬롯의 크기와 한 사이클에서 할당할 수 있는 정적 슬롯의 개수는 클러스터 내의 모든 노드에서 동일하게 설정되어야 한다. 처음부터 순서대로 1번 정적 슬롯이라 부르며, TDMA 방식에 따라 정해진 시간에 맞춰 각각의 정적 슬롯 구간에서 데이터를 송수신 할 수 있는 노드가 정해져 있다.

다음으로 동적 구간에서는 FTDMA 방식을 사용하여 데이터를 송수신한다. 동적 구간은 여러 개의 미니 슬롯(mini slot)으로 구성되며, 정적 슬롯과 마찬가지로 모든 클러스터 내의 슬롯의 길이와 개수는 동일하여야 한다. 하지만 정적 구간과는 다르게 노드별 우선순위를 지원하며, 노드별로 전송할 수 있는 데이터의 길이를 다르게 할 수 있다. 우선순위가 높게 설정된 노드에서 데이터 전송을 시작하면, 다른 노드는 먼저 전송된 데이터의 수신이 완료된 후 데이터를 송신할 수 있다. 이때, 데이터 송신을 위해 필요한 동적 구간의 길이가 남아있지 않다면, 송신을 포기하고 다음 사이클을 기다려야 한다.

심볼 윈도우는 프로토콜에서 정의하고 있는 CAS (collision avoidance symbol), MTS(media access test symbol), WUS(wake-up symbol)와 같은 특정 패턴을 전송하는 구간이다. 통신 사이클 당 하나의 패턴만 전송 할 수 있으며, 설계자의 필요에 따라 사용된다. 마지막으로 네트워크 유휴시간은 노드 간 동기화를 위해 사용되는 구간이다. 이 구간에서는 동기화 노드 간의 알고리즘을 처리하고 계산된 보정 값을 적용하여 동기화를 조정하는 절차가 수행된다.

여러 개의 ECU가 네트워크를 통해 연결되어 있는 분산 시스템에서 하나의 ECU는 개별적으로 자신의 클럭(clock)을 가지고 있으며, 이것은 오실레이터의 성능이나 전압의 변동과 같은 이유로 정확

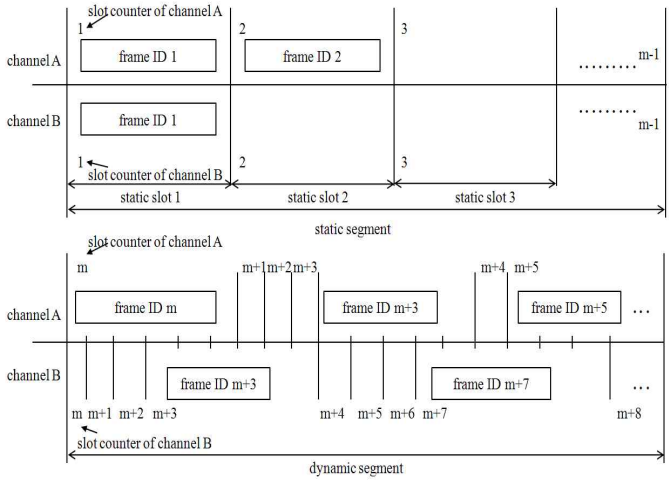

그림 2. FlexRay의 매체 접근 제어
Fig. 2 Media access control of FlexRay

하게 일치하지 않는다. 따라서 여러 개의 ECU가 연결되어 특정 태스크를 수행해야 하는 시스템에서는 ECU간 클럭 동기화는 중요한 일이며, FlexRay 프로토콜은 규격에서 이를 지원하기 위한 별도의 절차를 제공하고 있다. 본 장에서 기술하고 있는 FlexRay 프로토콜에서의 클럭 동기화 절차(clock synchronization process, CSP)는 각 노드간의 클럭을 동기화하기 위한 핵심 부분이다.

그림 3은 FlexRay 프로토콜의 클럭 동기화 절차를 도식적으로 나타낸 것이다. 그림 3에서 노드간 클럭 동기화를 위한 기본 동작을 살펴보면, CSP의 측정 구간(measurement phase)에서는 프레임의 시작시간과 도착시간의 편차를 측정하는 동기화 프레임(sync. frame)을 기준으로 클러스터에서의 정적 슬롯 시간과 자신의 정적 슬롯 시간을 비교하여 오차를 측정한다. CSP에서는 오차 측정값(measurement value)을 사용하여 각 사이클의 시간 오차를 수정하기 위해 매 사이클마다 오프셋 수정 값(offset correction value)을 계산하고 홀수 사이클에서 반영한다. 그리고 각 노드 간의 사이클 시간 오차를 수정하기 위해 홀수 사이클에서는 비율 수정 값(rate correction value)을 계산하여 각 사이클에서 반영한다. 이와 함께, 각각의 오차 보상을 통해 계산된 수정 값을 통해 매크로틱 생성절차(macrotick generation process, MTG)를 수행하고, 생성된 매크로틱을 기준으로 MAC(media access control)에서 데이터를 입출력하게 된다. 이러한 과정을 사이클마다 지속적으로 반복하면서 노드 간의 클럭을 동기화 하며, 데이터를 송수신하게 된다.

FlexRay 프로토콜에서 사용되는 클럭 동기화 절차를 앞에서 간략히 설명하였으나, 실제로 해당 절차를 수행하기 위해서는 복잡한 상호 관계에 있

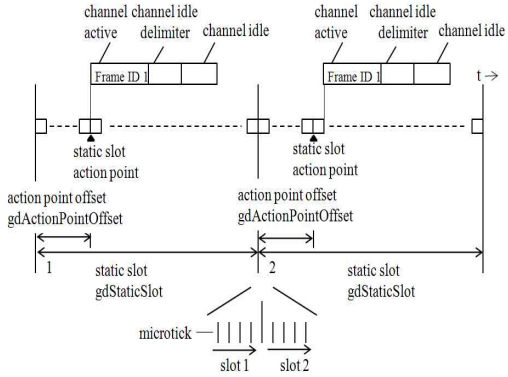

그림 4. 동기화를 위한 정적 슬롯 변수
Fig. 4 Parameters of static slot synchronization

는 수 많은 관련 구성 변수를 규격에 정의된 바에 따라 정확히 계산해야 한다. 예를 들어, 클럭 동기화를 위해 계산되는 정적 슬롯의 크기는 그림 4에서 보는 것과 같이 모두 8개의 관련 변수의 산술적인 계산으로 결정된다. 그림 4에서 나타낸 "gdActionPointOffset", "cChannelIdleDelimiter", "gdBitMax"와 같은 변수들은 프로토콜에서 클러스터에 참여하는 모든 노드에서 정해진 값을 가지도록 정의하는 것도 있고, 설계자의 필요에 따라 노드별로 다른 값을 가지는 변수도 있다.

관련된 구성 변수는 크게 시스템 변수와, 클러스터 설정 변수로 구분된다. 먼저 시스템 변수는 프로토콜에 의해 값이 일정하게 정해져 있기 때문에, 특별한 계산과정 없이 규격에서 정의한 값을 사용할 수 있다. 다음으로 클러스터 설정 변수는 설계자의 클러스터 설정에 따라 값이 변동되며, 복잡한 계산을 통해 직접 선정해야 한다. 클러스터 설정 변수는 다시 전역 클러스터 변수(global cluster parameter)와 노드 변수(node parameter)

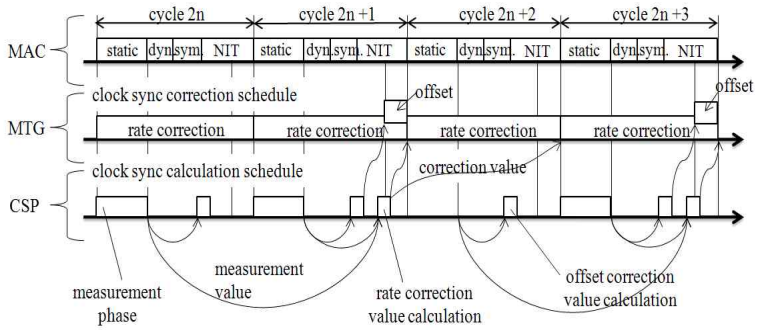

그림 3. FlexRay의 클럭 동기화 절차
Fig. 3 Clock synchronization process of FlexRay

로 구분된다. 전역 클러스터 변수는 노드의 동기화를 위해 클러스터에 참여하고 있는 모든 노드에서 반드시 동일한 값을 사용해야 한다. 반면에 노드 변수는 개별 노드마다 다른 값을 사용해도 동기화에 영향을 미치지 않는다. 전역 클러스터 설정 변수는 채널 A, B에 공통적으로 사용되는 설정변수를 포함하여 모두 53가지이다. 이를 프로토콜 규격에서는 플랫폼 구성 변수(platform configuration resister, PCR)라고 한다. PCR은 규격에서 정한 제약 조건이 만족되도록 사용자가 임의로 설정해야 하는 25개의 PCR과, 부록 A에서 나타낸 바와 같이 사용자 정의 PCR을 이용하여 계산되는 28개의 PCR이 있다. 특히, 사용자가 임의로 설정해야 하는 PCR을 주어진 FlexRay 네트워크 설계 조건(예로 노드 수나 메시지의 개수 등)에 맞추어서 제대로 설정하지 않는다면, 클럭 동기화가 이루어지지 않으며 FlexRay 통신이 성공하지 못한다.

Ⅲ. 플랫폼 구성 변수의 분석

앞 절에서 설명한 것과 같이 FlexRay 프로토콜을 사용하기 위해서는 높은 수준의 노드별 클럭 동기화가 요구되며, 이는 클러스터 변수와 노드 변수를 포함하여 규격에서 정의하고 있는 53개의 PCR에 의해서 결정된다. 당연히 이를 위해서는 네트워크 설계자가 FlexRay 프로토콜의 규격을 정확히 파악하여 최대의 성능이 나오도록 관련 변수를 직접 결정하는 것이 가장 이상적이다. 그러나 기존의 차량용 네트워크 설계자가 FlexRay 프로토콜의 규격을 이해하고 직접 동기화를 위한 설정 변수를 계산하는 것은 쉬운 일이 아니며, 이런 이유로 FlexRay 네트워크는 기존에 자동차에 널리 적용되어 있는 CAN 네트워크에 비해서 상대적으로 기술적 장벽이 높은 것으로 인식되고 있다.

그림 5는 FlexRay 프로토콜 규격 2.1 버전에서 규정하고 있는 PCR간의 상관관계를 도식적으로 나타낸 것이다. 본문에서는 PCR의 기능을 변수의 속성에 따라 정적 구간 관련 변수, 동적 구간 관련 변수, 심볼 윈도우 관련 변수, 기타 변수와 같이 4부분으로 구분하여 분석하였다. 우선 그림 5에서 실선으로 표현된 도형 안의 변수는 전역 클러스터 변수로 모든 노드에서 동일한 값을 가지는 변수이며, 점선으로 표현된 도형 안의 변수는 노드 변수를 나타낸 것이다. 다음으로 원형으로 표시된 부분은 수식관계에 의해서 계산되는 변수이며, 사각형으로 표시된 부분은 변수 값 결정을 위해서 외부의 입력이 요구되는 변수를 나타낸 것이다. 마지막으로 도식화상의 편의를 위해 중복으로 표현된 변수는 음영으로 나타내었다.

첫째, 통신 사이클(Input5)을 결정하기 위해서는 설계자의 입력을 받으면 된다. 하지만 사이클의 매크로틱의 크기를 결정하기 위해서 “Macro Per

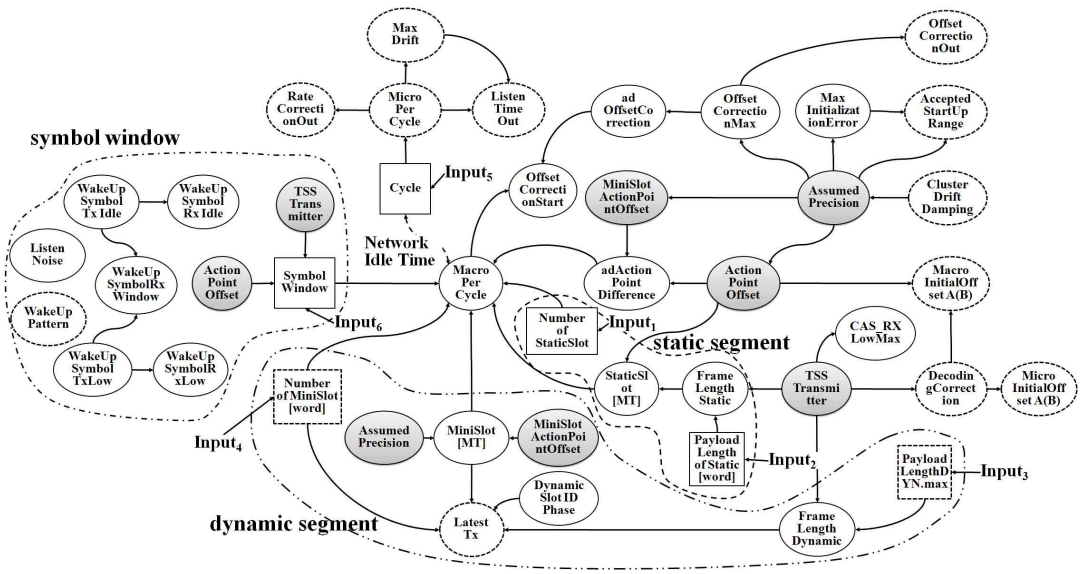

그림 5. 플랫폼 구성 변수간의 상관관계

Fig. 5 Schematic diagram of relationship of platform configuration register

Cycle(사이클 내의 매크로틱 수)"의 영향을 받게 되며, 또한 "Rate Correction Offset(최대 허용 비용 보정값의 크기)", "Max Drift(네트워크 최대 드리프트 값)", "Listen Time Out(start up과 wake up의 시간 초과 값)"의 값을 결정하기 위해 "Micro Per Cycle(지역 노드의 마이크로틱 수)"의 값을 결정하는데 영향을 미친다.

둘째, 그림 5에서 가운데에 위치하고 있는 정적 구간을 결정하기 위한 구성 변수는 4가지로서, "Payload Length of Static(정적 슬롯의 데이터 길이)", "Frame Length Static(정적 슬롯의 프레임 길이)", "Static Slot[MT](정적 슬롯)", "Number of Static Slot(정적 슬롯의 개수)"이다. 이때 관련된 변수를 계산하기 위한 "Action Point Offset(정적 슬롯과 심볼 윈도우의 시작으로부터 오프셋을 맞추는 역할)"과 "TSS Transmitter(프레임 구성 중 반드시 필요로 하는 변수, 시작을 알리는 연속적인 bit의 수를 나타냄)"를 같이 참조하게 된다. 그림 5에서 정적 슬롯의 개수(Input1)와 정적 슬롯의 데이터 길이(Input2)는 별도의 설계자의 입력이 요구되는 변수이며, 설명한 4개의 구성 변수는 모두 클러스터 설정변수로 클러스터에 참여하고 있는 모든 노드에서 동일하게 적용되어야 한다.

셋째, 그림 5에서 아래 부분에 위치하고 있는 동적 구간을 결정하기 위한 구성 변수는 6가지로서, "Payload Length DYN. Max(동적 슬롯의 데이터 길이)", "Frame Length of Dynamic(동적 슬롯의 프레임 길이)", "Latest Tx(동적구간의 마지막 미니 슬롯)", "Dynamic Slot Idle Phase(동적 슬롯에서 idel phase의 지속 시간)", "Number of Mini Slot(미니 슬롯의 개수)", "Mini Slot[MT](미니 슬롯)"이다. 정적 구간과 마찬가지로 관련 변수의 계산을 위해 "Assumed Precision(네트워크의 드리프트나 전송지연 등의 예상되는 지연을 나타냄)", "Mini Slot Action Point Offset(미니 슬롯의 시작으로부터 오프셋을 맞추는 역할)"를 같이 참조하게 된다. 그림 5에서 동적 슬롯의 최대 데이터 길이(Input3)와 미니 슬롯의 개수(Input4)는 설계자의 입력이 요구되는 변수이며, 실선으로 표현된 3개의 변수는 클러스터 설정변수이며, 점선으로 표현된 나머지 3개의 변수는 노드 설정변수로 노드마다 다르게 적용해도 무방하다.

넷째, 그림 5에서 좌측에 위치하고 있는 심볼 윈도우는 모두 8개의 구성변수로 구성되어 있다. 앞에서 설명한 것과는 다르게 심볼 윈도우는 대부분의 구성변수와 관계를 가지지 않는다. 다만, 동기화를 위해 "Symbol Window(심볼 윈도우)"를 사용할 경우, "Action Point Offset"과 "TSS Transmitter"를 참조하여 계산한다. 그림 5에서 심볼 윈도우의 크기(Input6)만 설계자의 입력이 요구되며, 나머지 7개의 구성변수는 모두 프로토콜 규격에서 정의한 값을 사용하여 계산된다.

마지막으로 별도로 분류를 하지 않은 모든 구성변수는 클럭의 동기화를 위해 사용되거나, 다른 변수의 계산을 위해서 별도로 계산되는 변수를 나타낸 것이다. 이런 구성변수는 모두 FlexRay 프로토콜의 전송속도에 결정되는 값이다.

표 1. FlexRay의 선택적인 PCR
Table 1. Optional PCR of FlexRay

Network management	Max Without Clock Correction Passive
	Max Without Clock Correction Fatal
	Network Management Vector Length
	Allow Halt due to Clock
	Allow Passive to Active
	Extern Offset Correction
	Extern Rate Correction
Protocol defined	Cold Start Attempt
	Sync Node Max
	Delay Compensation A(B)
User specified	Channel
	Wake Up Channel
	KeySlot Header CRC
	KeySlot ID
	KeySlot Used For Sync
	KeySlot Used For Startup
	SingleSlot Enabled

표 1은 FlexRay 프로토콜에서는 정의되어 있으나, 그림 5에 표시되지 않은 구성변수를 정리한 것이다. 표 1에서 나타낸 구성변수는 PCR의 일부이지만, 다른 변수들과 상관관계를 가지지 않고 설계자의 요구에 따라서 결정된다. 표 1에서 네트워크 관리(network management)를 위해 사용되는 "Max Without Clock Correction Passive(비율이나 오프셋 값을 계산할 수 없을 때 변수만큼 값이 유지되면 POC가 normal passive 상태로 천이)", "Max Without Clock Correction Fatal(비율이나 오프셋 값을 계산할 수 없을 때 변수만큼 값이 유지되면 POC가 halt 상태로 천이)", "Network Manag- ement Vector Length(클러스터 내의 네트워크 관리 변수의 길이를 나타냄)"와 프로토콜 정의(protocol defined)에서 사용되는 "Cold Start Attempt(클러스터의 동기화를 초기화 하면서 네트워크의 시작을 시도하는 역할)", "Sync Node Max

(동기화 프레임을 전송할 수 있는 최대 노드 수)"는 클러스터 설정변수로 모든 노드에서 동일한 값을 적용해야 한다. 그 외의 경우는 모두 설계자의 선택에 따라서 노드별로 다른 값을 적용해도 무방한 노드 변수이다. 표 1에서 나타내고 있는 변수들은 기본 값(A, B 채널의 동시 사용, 키 슬롯(key slot)을 싱크와 시작프레임으로 허용 등)을 가지고 있으며, 다른 값으로 선택하더라도 FlexRay 프로토콜의 통신이나 동기화에는 영향을 미치지 않는다.

Ⅳ. PCR 자동 생성 프로그램의 구현

본 장에서는 앞 장에서 분석한 PCR의 특성 및 상호 관계성을 이용하여 PCR 자동 생성 프로그램을 개발하고 적용 가능성을 평가하였다. 이미 전술한 것과 같이 비록 PCR은 복잡한 상호 관계를 가지고 있지만, 설계자의 클러스터 설계에 따라서 모든 변수의 값이 변경되는 것이 아니기 때문에 최소한의 입력(Input1~Input5)만을 사용하였다. 프로그램의 설계 시 설계자가 필수적으로 결정해야 하는 5개의 구성변수를 선정하였는데, 이는 "정적 슬롯의 개수(number of static slot, NSS)(Input1)", "정적 슬롯의 데이터 길이(payload length of static slot, LSS)(Input2)", "동적 슬롯의 최대 데이터 크기(payload length of DYN.max, LDS)(Input3)", "미니 슬롯의 개수(number of mini slot, NDS)(Input4)", "통신 주기(base cycle, TC)(Input5)"이다. 심볼 윈도우(Input6)는 설계자의 입력이 필요하지만, 반드시 필요한 구성 요소가 아니기 때문에 필수 구성 변수로 사용하지 않았다. 통신 주기를 초과하지 않는 범위에서 설계자가 설계한 5개의 변수를 입력하면 이를 통해서 나머지 48개의 변수를 규격과 기본 값을 통해서 자동으로 생성하도록 프로그램을 구성하였다.

그림 6은 본 논문에서 설계한 FlexRay 플랫폼 구성 변수 자동 생성 프로그램의 동작 흐름도를 나타낸 것이다. 프로토콜의 통신 속도는 기본 값인 10Mbps로 설정하였다. 프로그램에서는 5개 구성 변수의 입력을 받으면, 가장 먼저 사용자가 지정한 NSS , LSS , NDS , LDS를 이용하여 정적 구간의 길이(TSS)와 동적 구간의 길이(TDS)를 계산한다. 심볼 윈도우는 기본값을 0으로 설정하였다. 다음으로 통신 주기와 세그먼트의 길이, 심볼 윈도우의 길이를 고려하여 네트워크 유휴시간(TNIT)을 계산한다. FlexRay 프레임을 구성하기 위한 4가지의

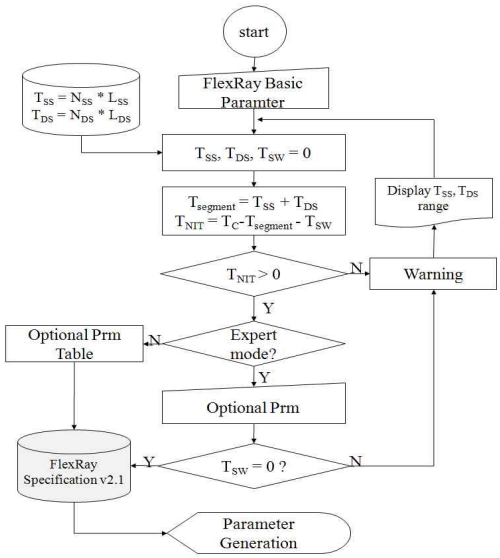

그림 6. PCR 자동 생성 프로그램의 흐름도
Fig. 6 Flowchart of PCR Auto Generation S/W

구성요소(TSS, TDS, TSW, TNIT)의 길이의 합이 사용자가 입력한 통신 주기(TC)와 같으면, 선택적 구성 변수를 설정하는 단계로 넘어간다. 만약 네트워크 유휴시간의 크기가 0보다 작으면, 경고 메시지를 팝업(pop-up)시켜 사용자에게 알리고, 현재 통신 주기에 맞게 슬롯의 최대값을 나타내고 사용자가 직접 수정할 수 있도록 입력 루틴으로 돌아간다.

선택적 구성 변수를 설정하는 단계에서는 설계자의 필요에 따라 통신 채널이나 CRC, 키 슬롯 등을 변경할 수 있다. 사용자가 선택적 구성 변수를 직접 설계하고 나면, 가장 먼저 심볼 윈도우의 사용 유무를 판단한다. 만약 심볼 윈도우를 사용한다면 통신 주기(TC)를 초과하기 때문에 이를 사용자에게 알려주고, 네트워크 유휴시간을 변경한다. 네트워크 유휴시간을 변경하였음에도 불구하고 통신 주기를 초과하면 동적 구간의 미니 슬롯 개수를 줄이도록 하였다. 그러나 사용자가 선택적 구성변수로 기본 값을 사용하거나, 심볼 윈도우를 사용하지 않으면 프로토콜 규격 2.1 버전에서 정의하고 있는 기본 값에 의해서 나머지 변수를 생성한다. 이후 생성된 파라미터는 사용자의 가독성을 고려하여 각각의 값을 변수 이름과 함께 나타내었다.

그림 7은 구현한 FlexRay 클러스터 구성 변수의 자동 생성 프로그램을 나타낸 것이다. 구현된 소프트웨어는 기능적으로 "Basic Prm(기본 변수부)

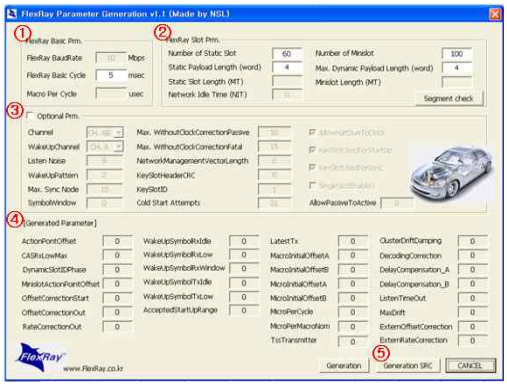

그림 7. PCR 자동 생성 프로그램
Fig. 7 FlexRay PCR Auto Generation S/W

", "Slot Prm(설계 변수부)", "Optional Prm(선택적 변수부)", "Generated Prm(생성 변수부)", "Generation SRC(외부 코드생성)"과 같이 5부분으로 구성하였다. 먼저 "Basic Prm"는 FlexRay의 통신 속도와 통신 주기를 설정하는 부분이며, 통신 속도는 기본 값인 10Mbps로 설정하였다. "Slot Prm"는 FlexRay의 네트워크 설계자가 설계한 정적 슬롯의 개수와 길이, 동적 슬롯의 개수와 길이를 입력하는 부분이다. 이 부분에서는 입력된 정보를 기준으로 정적 구간과 동적 구간의 크기를 결정하고 네트워크 유휴시간을 계산한다. "Optional Prm"는 설계자가 선택적 구성변수를 설정하는 부분으로, 필요에 의해서 심볼 윈도우의 사용 유무, 통신 채널, 시작 시도 횟수, 싱크와 시작프레임 전송을 위한 키 슬롯 ID, CRC 등을 설정할 수 있다. "Generated Prm"는 입력된 구성변수를 이용하여 FlexRay의 나머지 구성 변수를 계산해서 출력하는 부분이다. 마지막 "Generation SRC"에서는 생성된 PCR을 펌웨어에 입력하기 위해 그림 8과 같은 소스코드 형태로 파일을 생성하는 역할을 한다.

본 논문에서는 PCR 생성 프로그램의 적용 가능성을 검증하기 위하여 MC9S12XF512 모듈 2개와 MPC5517모듈 1개를 그림 9와 같이 구성하였다. 또한, 네트워크 상태를 관측하기 위해 네트워크 모니터링 장비인 CANoe. FlexRay v7.0을 이용하였다.

본 논문에서는 임의로 선택된 정적 슬롯의 개수(Input1)와 최대 데이터 길이(Input2), 동적 슬롯의 최대 데이터 길이(Input3)와 미니 슬롯의 개수(Input4)를 PCR 자동 생성 프로그램에 입력하여 자동 생성된 PCR을 이용하였을 경우, FlexRay 네

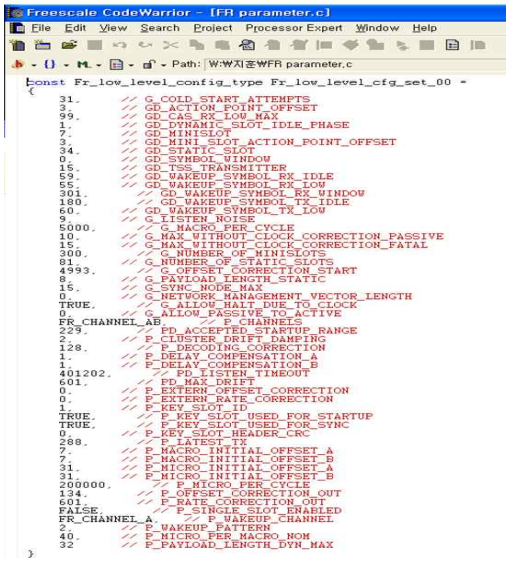

그림 8. PCR 자동 생성 프로그램의 출력 형태
Fig. 8 Output type of PCR auto generation S/W

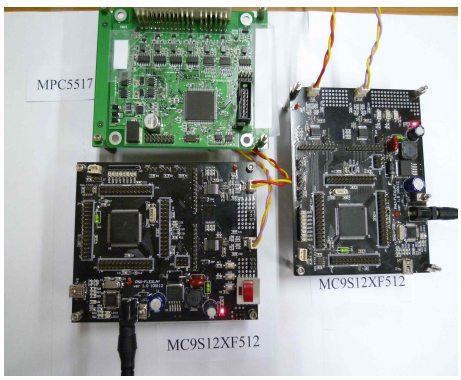

그림 9. 실험 환경 testbed 구현
Fig. 9 Implementation of testbed

트워크의 클럭 동기화와 통신이 정상적으로 이루어지는 지를 검사하기 위하여 표 2와 같이 두 가지 실험 구성 변수를 선택하였다. 두 가지 실험 구성 변수에서 통신 사이클(Input5)와 심볼 윈도우(Input6)은 동일하게 설정하였다. CASE (I)의 구성 변수는 정적 구간에서 8word 크기를 가지는 81개의 정적 슬롯을 사용하였으며, 동적 구간에서는 최대 16word 크기의 데이터 전송과 300개의 미니 슬롯을 사용하였다. CASE (II)의 구성변수는 정적 구간에서 4word의 크기를 가지는 138개의 정적 슬롯을 사용하였으며, 동적 구간에서는 최대

표 2. 클러스터 구성변수
Table 2. Cluster variable

	CASE(I)	CASE(II)
BaudRate	10Mbps	10Mbps
cycle(T_c)	5ms	5ms
payload length of static(L_{SS})	8word	4word
payload length of DYN.max(L_{DS})	16word	32word
number of static(N_{SS})	81	138
number of mini(N_{DS})	300	200
symbol window	0	0
max. sync node	15	15
channel	A, B	A, B
wake up channel	A	A

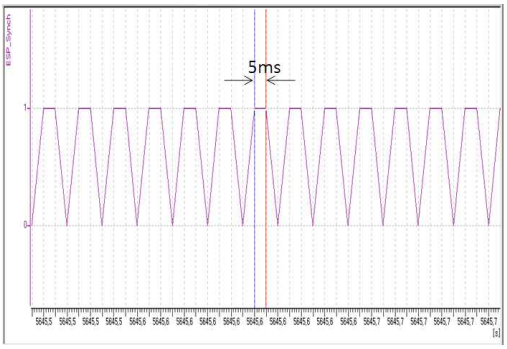

그림 10. 자동 생성된 PCR을 이용한 경우의 통신 성능 평가
Fig. 10 Evaluation of FlexRay communication in case of auto generated PCR

32word의 크기를 가지는 미니 슬롯 200개를 사용하였다. 선택적 구성 변수중의 일부인 심볼 윈도우는 사용하지 않았으며, 통신 채널과 키 슬롯 ID등은 본 실험에 적합하게 변경하였다. 클러스터에 참여하는 노드에서는 매 사이클 마다 하나의 싱크 프레임을 전송하여 동기화를 유지하게 하였다.

그림 10은 자동 생성된 PCR을 입력하여 FlexRay 통신을 수행하였을 때의 모니터링 결과를 나타내고 있다. CANoe에서 확인한 결과 전송되는 데이터의 오류나 프레임 손실 등이 관측되지 않았으며, 전송 주기와 설계한 슬롯의 크기와 개수도 동일하였다. 그래픽 창을 통해서도 출력하는 모든 내용이 스케줄링 정보와 동일하다는 것을 확인할 수 있으며, 사이클 지연 없이 모든 메시지들이 정상적으로 송수신 되는 것을 확인 할 수 있었다. 그림 10에서 나타내고 있는 결과는 구성변수를 CASE (I)과 CASE(II)로 설정한 환경에서 동일하게 관측되었다.

그림 11은 표 2와 같이 설계한 상태에서 CASE(I)과 CASE(II)의 노드별 동기화 편차를 나타낸 것이다. 실험을 위해서 MPC5517모듈은 1번째 슬롯(Frame ID 1)에서 송신을 하고, MC9S12XF512 모듈 2개는 2번째 정적 슬롯(Frame ID 2)에서 수신하도록 프로그램 하였다. 이때, MC9S12XF512 모듈에서 2번째 정적 슬롯(Frame ID 2)을 수신할 때 외부 I/O를 "High" 상태에서 "Low" 상태로 트리거 하도록 한 상태에서 오실로스코프로 차이를 측정하였다. 실험은 네트워크 설계에서 필수 PCR만 변경하고, 나머지 구성 변수는 모두 동일한 상태에서 수행되었다. 실험에서 2번째 정적 슬롯을 수신한 모듈 2개의 동기화 지연은 1ns이하로 측정되었으며, 이는 필수 구성 변수를 변경한 상태에서도 동일하게 1ns이하로 측정되었다.

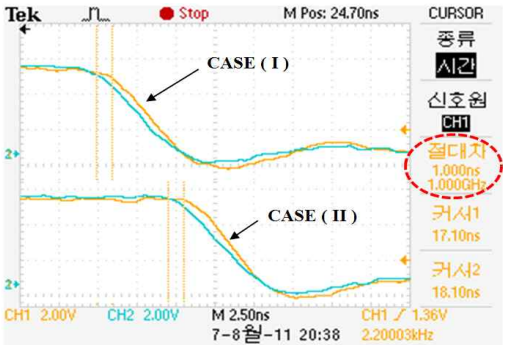

그림 11. 자동 생성된 PCR을 이용한 경우의 FlexRay 동기화 성능 평가
Fig. 11 Evaluation of FlexRay synchronization in case of auto generated PCR

Ⅴ. 요약 및 결론

본 논문에서는 FlexRay 프로토콜을 사용해야 하는 기존의 차량용 네트워크 설계자의 기술적 진입을 용이하게 하기 위한 목적으로 FlexRay 플랫폼 구성 변수 자동 생성 프로그램을 설계하고 적용 가능성을 검증하였다. 이를 위하여, FlexRay 프로토콜에서 사용하는 클럭 동기화 절차와 53개 PCR의 상호관계성에 대해 분석하였다. 이를 토대로 설계자가 반드시 입력해야 하는 필수 구성 변수와 선택적 구성 변수를 구분하였으며, PCR 자동 생성 프로그램을 개발하였다. 마지막으로 개발된 프로그

램의 적용 가능성을 평가하기 위하여, 기본적인 통신 테스트와 동기화 성능테스트를 수행하였다. 이를 통해서 다음과 같은 결론을 도출할 수 있었다.

1) FlexRay 프로토콜은 TDMA 기반의 매체 접근 방식을 사용하기 때문에, 클러스터에 참여하고 있는 모든 노드에 대해서 높은 수준의 클럭 동기화가 요구되며, 프로토콜 규격에서는 복잡한 상호관계에 있는 53개의 PCR을 정의하고 있다. 본 논문에서는 PCR의 분석을 통해서 동기화와 직접적인 관련이 있는 변수를 도출할 수 있었으며, 설계자의 입력에 따라 영향을 미치는 구성 변수의 관계를 확인하였다.

2) 본 논문에서는 FlexRay 프로토콜을 단순히 사용하고자 하는 일반적인 차량용 네트워크 설계자를 위해서 필수 구성 요소와 선택적 구성 변수로 PCR을 구분하였다. 차량용 네트워크 설계자는 정적 슬롯의 개수와 최대 데이터 길이, 동적 슬롯의 최대 데이터 크기와 미니 슬롯의 개수, 통신 사이클과 같이 5개의 필수 구성 요소만 입력하고 다른 구성 변수를 변경하지 않아도 통신의 성능이나 동기화에는 영향을 미치지 않음을 실험적으로 확인하였다.

그러나, 본 연구는 FlexRay 프로토콜의 성능 개선의 목적보다는 기존의 차량용 네트워크 설계자의 기술적 진입을 용이하게 하기 위한 목적으로 진행되었다. 따라서, 본 연구를 통해 개발된 클러스터 구성 변수의 자동 생성 프로그램은 테스트 환경에서 사용하기 위한 기능의 구현에만 초점을 맞추고 있다. 보다 실용적으로 사용되기 위해서는 변수 최적화와 같은 연구가 필요하다. 예를 들어, 차량용 네트워크 설계자가 CAN 네트워크에서 사용하고 있는 메시지 표준 포맷인 실차 CANdb를 입력하게 되면, 5개의 필수 PCR을 자동으로 결정해 주는 최적화 알고리즘에 대한 연구가 요구된다. 뿐만 아니라, 하나의 실차 CANdb를 이용하여 CAN 네트워크와 FlexRay 네트워크를 구성하였을 경우 네트워크 성능 변화에 대한 실용적인 연구도 필요하다. 마지막으로 본 연구에서는 프리스케일사의 CC(communication controller) 내장형 마이크로컨트롤러만을 대상으로 적용 가능성을 평가하였다. 향후에는 다양한 제조사의 마이크로컨트롤러에도 적용될 수 있도록 자동 생성 프로그램의 보완이 필요하다.

참고문헌

[1] 김현구, 정호열, 박주현, “적응형 헤드 램프 컨트롤을 위한 야간 차량 인식,” 대한임베디드공학회논문집, Vol. 6, No. 1, pp.8-15, 2011.

[2] 정슬, T.C.Hsia, "Intelligent technique application for autonomous lateral position control of an unmanned 4 wheel steered snow plow robotic vehicle," 대한임베디드공학회논문집, Vol. 6, No. 3, pp.132-138, 2011.

[3] T. Yoshida, H. Kuroda, T. Nishigaito, “Adaptive driver-assistance systems,” Hitashi review, Vol. 53, pp.212-216, 2004.

[4] M. Houtenbos, H.M. Jagtman, M.P. Hagenzieker, P.A. Wieringa, A.R. Hale "Understanding road users' expectations: an essential step for ADAS development," European Journal of Transport and Infrastructure Research, Vol. 5, No. 4, pp.253-266, 2005.

[5] http://www.carlife.net/bbs/board.php?bo_table = school&wr_id=2725

[6] G. Leen, D. Hefferman, A. Dunne, "Digital networks in the automotive vehicle," Computer & Control Engineering Journal, Vol. 10, No. 6, pp.257-266, 1999.

[7] A. Trachtler, "Integrated vehicle dynamics control using active brake, steering and suspension systems," International Journal of Vehicle Design, Vol. 36, No. 1, pp.1-12, 2004.

[8] T.H. Hwang, K. Park, S.J. Heo, S.H. Lee, J.C. Lee, "Design of integrated chassis control logics for AFS and ESP," International Journal of Automotive Technology, Vol. 9, No. 1, pp.17-27, 2008.

[9] 박기홍, "지능형자동차 통합샤시제어기술," 한국정밀공학회논문지, Vol. 23, No. 9, pp.15-22, 2006.

[10] S. Ding, X. Yin, H. Xu, S, Zhang, "A hybrid GA-based scheduling method for static segment in FlexRay systems," Proceedings on 2010 Chinese Control and Decision Conference, pp. 1548-1552, 2010.

[11] P.K. Mishra, S. Naik, "Distributed control system development for FlexRay-based systems," Proceedings on 2005 SAE World

Congress & Exhibition, No. 2005-01-1279 2005.
[12] FlexRay Consortium, FlexRay Communication System Protocol Specification Version 2.1 Revision A, 2005.

저 자 소 개

양 재 성 (Jae-Sung Yang)

2010년 : 부경대 제어계측공학과 학사. 2012년 : 부산대 기계공학부 석사. 현재, LG전자 연구원.
관심분야: 차량용 네트워크.
Email: jsyang@pnu.eud

박 지 훈 (Jeehun Park)

2004년 : 부산대 기계공학부 학사. 2012년 : 부산대 기계공학부 박사. 현재, 부산대학교 전임연구원.
관심분야: Cyber-Physical System, 차량용 네트워크, 로봇용 네트워크.
Email: network@pusan.ac.kr

이 석 (Suk Lee)

1984년 : 서울대 기계공학과 학사. 1985년 : 펜실바니아주립대 석사. 1990년 : 펜실바니아주립대 박사. 현재, 부산대학교 기계공학부 교수.

관심분야: 산업용 네트워크, 차량용 네트워크, 센서 네트워크.
Email: slee@pusan.ac.kr

이 경 창 (Kyung-Chang Lee)

1996년 : 부산대 생산기계공학과 학사. 1998년 : 부산대 생산기계공학과 석사. 2003년 : 부산대 지능기계공학과 박사. 현재, 부경대학교 제어계측공학과 교수.

관심분야: Cyber-Physical System, 차량용 네트워크, 로봇용 네트워크.
Email: gclee@pknu.ac.kr

최 광 호 (Gwang-Ho Choi)

2007년 : 부경대 제어계측공학과 학사. 2009년 : 부산대 기계공학부 석사. 현재, ETRI 대경권연구센터 연구원.
관심분야: 임베디드 시스템, 디바이스 드라이브.
Email: xingdoli@etri.re.kr

Appendix : 28개 PCR의 계산식

[1] gdActionPointOffset[MT]>=ceil((gAssumedPrecision[μs]-gdMinPropagationDelay[μs])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)))
[2] gdCASRxLowMax[gdBit]=ceil(2*(gdTSSTransmitter[gdBit]+cdCAS[gdBit])*(1+cClockDeviationMax)/(1-cClockDeviationMax)+2*dBDRxai[μs]/gdBitMin[μs/gdBit])
[3] gdMinislot[MT]>=gdMinislotActionPointOffset[MT]+ceil((gdMaxPropagationDelay[μs]+gAssumedPrecision[μs])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)))

[4] gdMinislotActionPointOffset[MT]>=ceil((gAssumedPrecision[μs]-gdMinPropagationDelay[μs])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)))

[5] gdStaticSlot[MT]=2*gdActionPointOffset[MT]+ ceil(((aFrameLengthStatic[gdBit]+ cChannelIdleDelimiter[gdBit])*gdBitMax[μs/gdBit]+ gdMinPropagationDelay[μs]+ gdMaxPropagationDelay[μs])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)))

[6] gdSymbolWindow=2*gdActionPointOffset[MT]+ ceil(((gdTSSTransmitter[gdBit]+ cdCAS[gdBit]+ cChannelIdleDelimiter[gdBit])*gdBitMax[μs/gdBit]+ dBDRxai[μs]+ gdMinPropagationDelay[μs]+ gdMaxPropagationDelay[μs])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)))

[7] gdTSSTransmitter[gdBit]>=ceil((gdBitMax[μs]+ dBDRxia[μs]+ max({x| x=nStarPathM,N})*dStarTruncation[μs])/gdBitMin[μs/gdBit]

[8] gdWakeupSymbolRxIdle[gdBit]=floor((gdWakeupSymbolTxIdle[gdBit]*(1-cClockDeviationMax)-gdWakeupSymbolTxLow[gdBit]*(1+ cClockDeviationMax))/(2*(1+ cClockDeviationMax)))

[9] gdWakeupSymbolRxLow[gdBit]=floor((gdWakeupSymbolTxLow[gdBit]*gdBitMin[μs/gdBit]-(max({x|=nStarPathM, N})*dStarTruncation[μs]+ dBDRxia[μs]))/gdBitMax[μs/gdBit])

[10] gdWakeupSymbolRxWindow[gdBit]=ceil((2*gdWakeupSymbolTxLow[gdBit]+ gdWakeupSymbolTxIdle[gdBit])*(1+ cClockDeviationMax)/(1-cClockDeviationMax))

[11] gdWakeupSymbolTxIdle[gdBit]=ceil(cdWakeupSymbolTxIdle[μs]/gdBit[μs/gdBit])

[12] gdWakeupSymbolTxLow[gdBit]=ceil(cdWakeupSymbolTxLow[μs]/gdBit[μs/gdBit])

[13] gMacroPerCycle[MT]=gdStaticSlot[MT]*gNumberOfStaticSlots+ adActionPointDifference[MT]+ gdMinislot[MT]*gNumberOfMinislots+ gdSymbolWindow[MT]+ gdNIT[MT]

[14] gOffsetCorrectionStart[MT] = gMacroPerCycle[MT]-adOffsetCorrection[MT]

[15] pdAcceptedStartupRange[μT]>=ceil((gAssumedPrecision[μs]+ gdMaxInitializationError[μs])/(pdMicrotick[μs/μT]*(1-cClockDeviationMax)))

[16] pClusterDriftDamping[μT]<=gdMaxMicrotick[μs]/pdMicrotick[μs]*gClusterDriftDamping[μT]

[17] pDecodingCorrection[μT]=round(((gdTSSTransmitter[gdBit]+ cdFSS[gdBit]+ 0.5*cdBSS[gdBit])*cSamplesPerBit[samples/gdBit]+ cStrobeOffset[samples]+ cVotingDelay[samples])/pSamplesPerMicrotick[samples/μT])

[18] pMacroInitialOffset[Ch][MT]=gdActionPointOffset[MT]+ ceil((pDecodingCorrection[μT]+ pDelayCompensation[Ch][μT])/pMicroPerMacroNom[μT/MT])

[19] pdListenTimeout[μT]=2*(pMicroPerCycle[μT]+ pdMaxDrift[μT])

[20] pdMaxDrift[μT]=ceil(pMicroPerCycle[μT]*2*cClockDeviationMax/(1-cClockDeviationMax))

[21] pExternOffsetCorrection[μT]<=pOffsetCorrectionOut[μT]

[22] pExternRateCorrection[μT]<=pRateCorrectionOut[μT]

[23] pLatestTx[Minislot]<=floor(gNumberOfMinislots[Minislot]-(((aFrameLengthDynamic[gdBit]+ vDTSLowMin)*gdBitMax[μs/gdBit])/(gdMacrotick[μs/MT]*(1-cClockDeviationMax)*gdMinislot[MT/Minislot]))-gdDynamicSlotIdlePhase[Minislot])

[24] pMacroInitialOffset[Ch][MT]=gdActionPointOffset[MT]+ ceil((pDecodingCorrection[μT]+ pDelayCompensation[Ch][μT])/pMicroPerMacroNom[μT/MT])

[25] pMicroInitialOffset[Ch][μT]=(pMicroPerMacroNom[μT]-((pDecodingCorrection[μT]+ pDelayCompensation[Ch][μ T])(modulo)pMicroPerMacroNom[μT]))(modulo)pMicroPerMacroNom[μT]

[26] pMicroPerCycle[μT]=round(gdCycle[μs]/pdMicrotick[μs/μT])

[27] pOffsetCorrectionOut[μT]=ceil(gOffsetCorrectionMax[μs]/(pdMicrotick[μs/μT]*(1-cClockDeviationMax)))

[28] pRateCorrectionOut[μT]=ceil(pMicroPerCycle[μT]*2*cClockDeviationMax/(1-cClockDeviationMax))