CAN 통신 개요 및 파라미터 정리

2022-09-01

Contents

Contents

차량 내 통신 개요

- CAN
  - CSMA/CD+AMP, 차량 내에서 가장 많이 사용되는 통신방식
  - 저비용, 꼬인 2선 사용, 버스형 통신
- LIN
  - Polling 사용, UART 기반, CAN 보조 역할
  - 저비용, 단일선 사용, 버스형 통신
- FlexRay
  - TDMA 사용, 안전이 중요한 응용에 사용
  - 고비용, 꼬인 2선 사용, 버스형 or 별(star)형 통신, 백업 네트워크 사용
- Ethernet
  - 큰 대역폭과 빠른 속도를 필요로 하는 경우 적합
    - Ex) ADAS, 진단장치 등
  - 고비용, RJ45 케이블 사용, Point-to-Point 사용
- MOST
  - TDMA 사용, 고속 멀티미디어 데이터 전송
  - 고비용, 플라스틱광섬유 사용, 링 구조

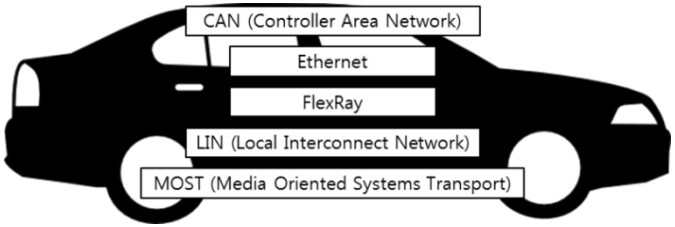


Fig. 차량 내에서 사용되는 다양한 통신방식

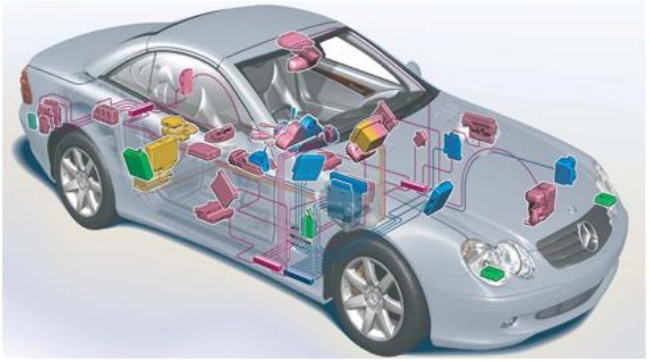


Fig. 통신이 필요한 차량 내 모듈

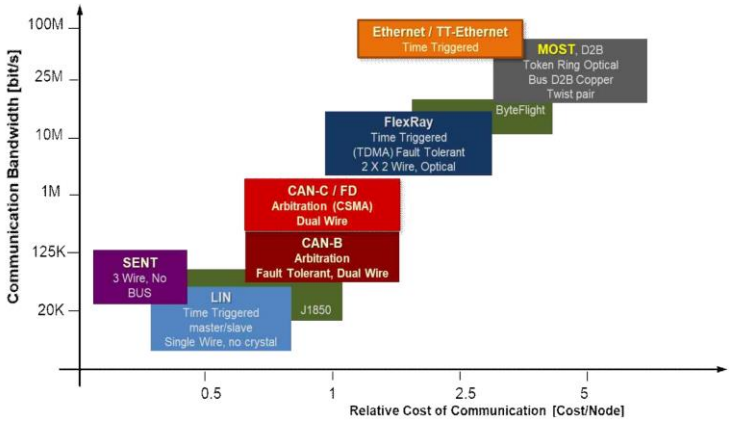


Fig. 대역폭, 비용에 따른 통신방식 분류

CAN 통신 계층 구조

- OSI 7계층 구조
  - 통신 프로토콜 단계 별 역할을 설명
  - 문제 발생 시 원인 파악 용이

- 주요 계층 소개
  - 물리 계층
  - 데이터 링크 계층
  - 네트워크 계층
  - 응용 계층

- CAN 통신 3계층 구조
  - 응용/데이터 링크/물리 계층으로 구성
  - ISO 11898에서 데이터 링크/물리 계층 표준을 정의

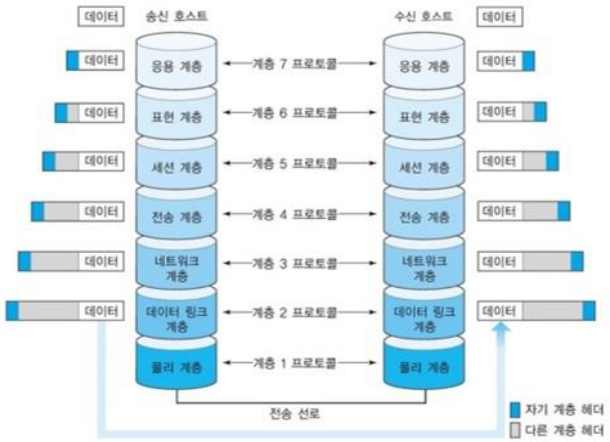


Fig. OSI 7계층 구조

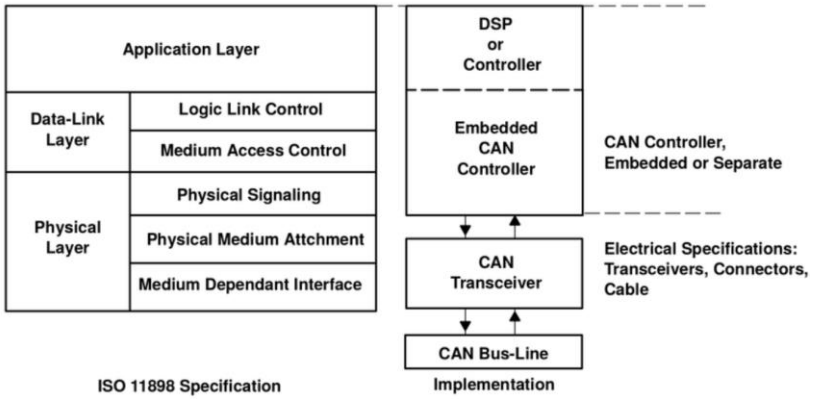


Fig. CAN통신에서의 OSI 계층 구조

CAN 통신 계층 구조 – 물리 계층 (1계층)

- 물리적 연결 방법
  - 2개의 전송 라인 사용 (CAN_{High} / CAN_{Low})
- 차동 신호
  - 정의: 크기가 같고 부호가 반대인 두 신호의 차이
    - (CAN_{High} – CAN_{Low}) when CAN_{High} = –CAN_{Low}
  - 두 신호 라인에 동일하게 유입되는 노이즈 제거에 유리함
- 꼬인 2선 구조(w/ 차동 신호)
  - 꼬인 2선으로부터 발생하는 노이즈 적음
  - 외부로부터 꼬인 2선으로 유입되는 노이즈 적음

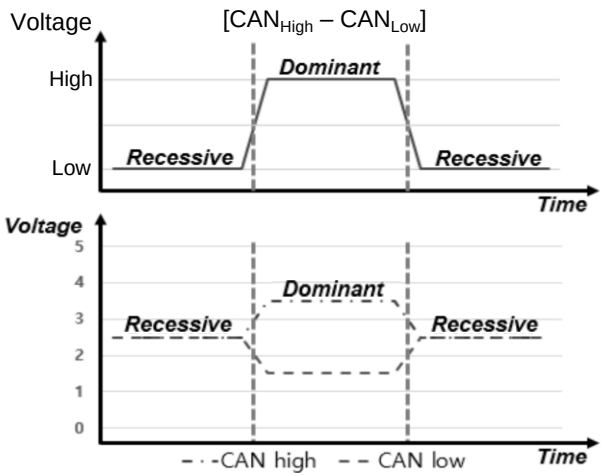


Fig. 차동 신호의 dominant, recessive

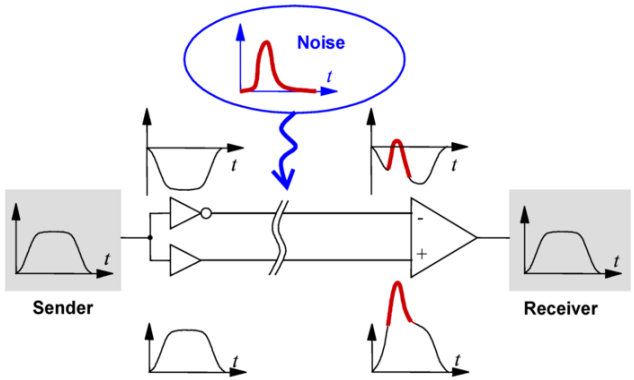


Fig. 차동 신호 구성 시 노이즈 제거 과정

Fig. 꼬인 2선 케이블 구조

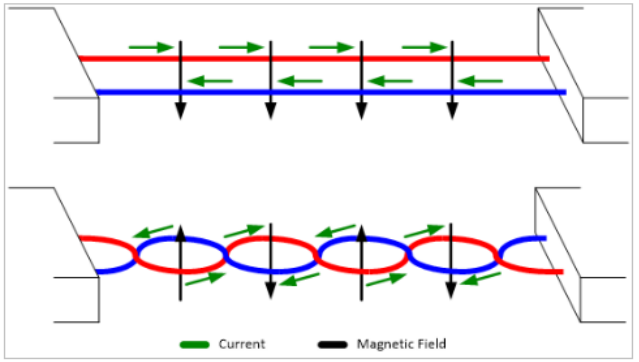


Fig. 꼬인 2선 구성 시 발생 노이즈

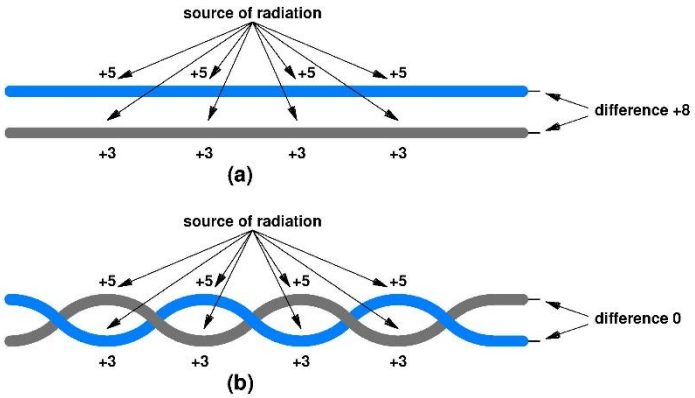


Fig. 꼬인 2선 구성 시 외부 노이즈 영향

CAN 통신 계층 구조 – 물리 계층 (1계층)

- 0(Low)과 1(High)의 정의
  - 차동 신호가 크면 dominant (0 또는 Low)로,
    작으면 recessive (1 또는 High)로 정의됨
- 트랜시버 회로 구성
  - dominant 송신: 스위치를 켬 → CAN_{High} / CAN_{Low}에 각각 5V, 0V 출력
  - recessive 송신: 스위치를 끔 → CAN_{High} / CAN_{Low} 모두 2.5V 출력
  - BUS에 recessive와 dominant가 동시에 송신될 경우, dominant만 송신
  - 다른 트랜시버와 연결되어 있으므로 보호 소자(제너 다이오드) 사용

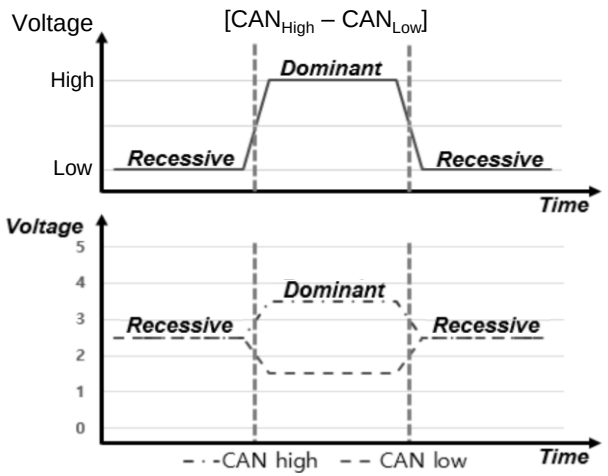


Fig. 차동 신호의 dominant, recessive

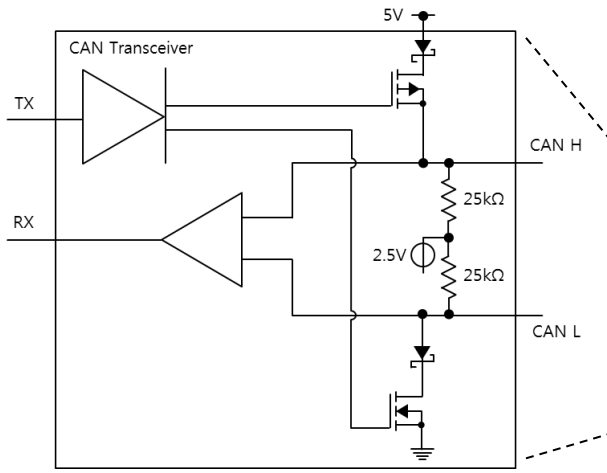


Fig. CAN 트랜시버 내부 회로도

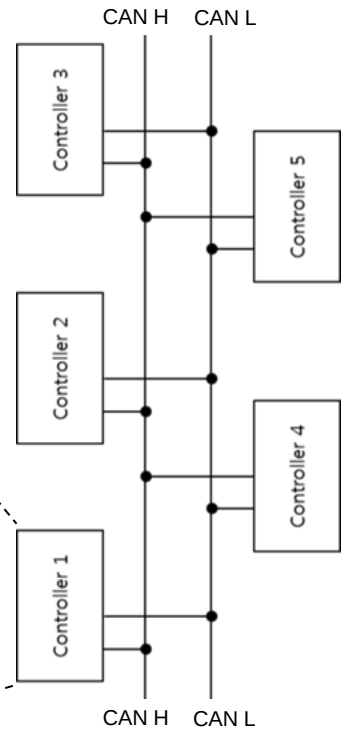


Fig. CAN 통신에 사용되는 버스 구조

CAN 통신 계층 구조 – 데이터 링크 계층 (2계층)

- 프레임 구조
  - 물리 계층을 통해 전송될 데이터 포맷
  - 프레임 길이는 가변 (제어 필드 값에 의해 결정)
- 필드 구성
  - SOF(1 bit), EOF(7 bit): 프레임의 시작과 끝
  - Identifier(11 bit or 29 bit): 메시지 식별자
  - IDE(1 bit): Identifier 형식 결정
  - DLC(4 bit): 데이터 필드의 바이트 수 결정
  - CRC(15 bit): 데이터 필드 에러 체크
  - DEL(1 bit): 필드와 필드 사이를 구분

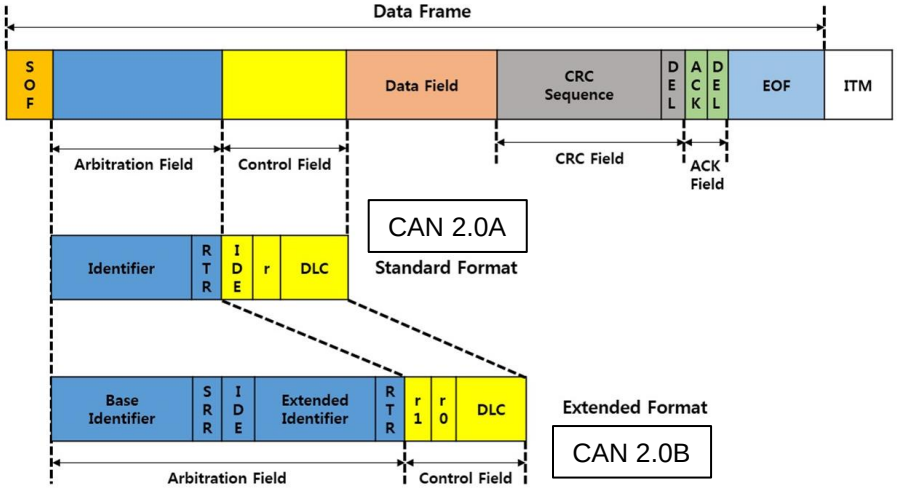


Fig. CAN 메시지 프레임 구조

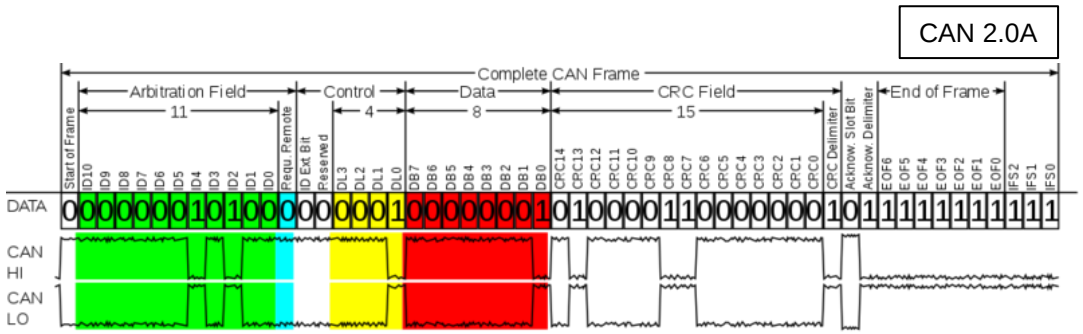


Fig. CAN 통신 메시지 전송 시 데이터 예시

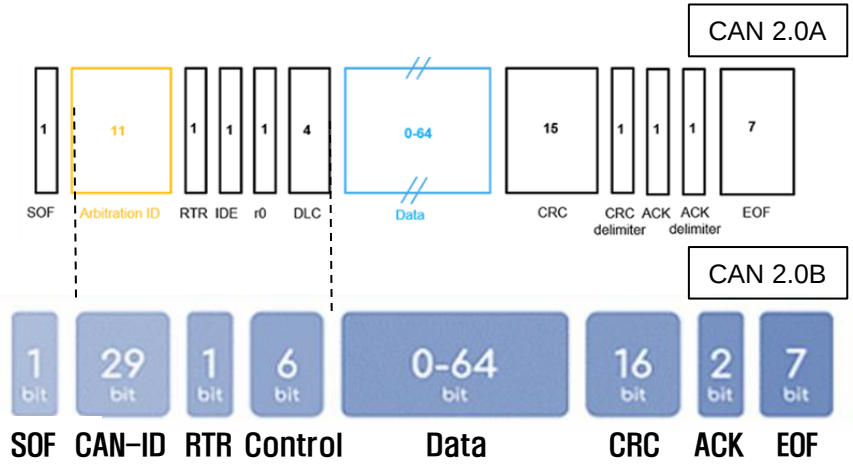


Fig. 메시지 프레임 필드 별 길이

CAN 2.0A, CAN 2.0B 이외의 CAN 통신 (추가 슬라이드)

- 저속 CAN vs 고속 CAN
  - 저속 CAN: 최대 125 kbps 속도로 통신 (CAN 2.0B)
  - 고속 CAN: 최대 1 Mbps 속도로 통신 (CAN 2.0C)
- CAN FD
  - 데이터 전송 시간을 줄이고 CAN 네트워크 부하를 낮춤
  - 데이터 필드의 길이를 최대 64 Byte까지 가질 수 있으며, 최대 5 Mbps 속도로 통신 가능
  - 제어비트
    - FDF(1 bit): CAN FD 프레임 여부 (비트가 '1' 이면 FD)
    - BRS(1 bit): 데이터 필드 전송 시 더 높은 비트 전송속도 사용 여부 (비트가 '1' 이면 더 빠른 속도 사용)
    - ESI(1 bit): '1' 이면 Error Passive, '0' 이면 Error Active

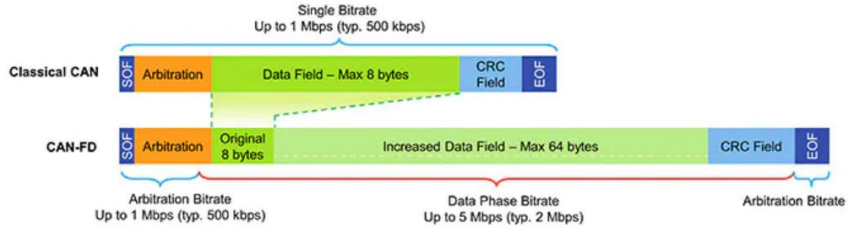


Fig. CAN FD통신에서 DLC 필드에 따른 데이터 전송 속도

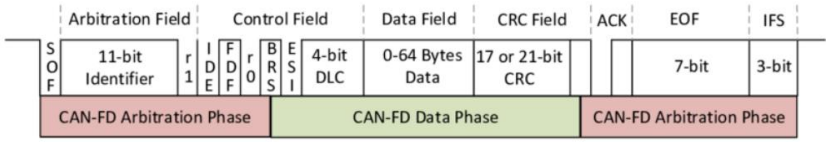


Fig. CAN FD 데이터 프레임 구조

DLC	Classic CAN	CAN FD
0	0 Byte	0 Byte
1	1 Byte	1 Byte
2	2 Byte	2 Byte
3	3 Byte	3 Byte
4	4 Byte	4 Byte
5	5 Byte	5 Byte
6	6 Byte	6 Byte
7	7 Byte	7 Byte
8	8 Byte	8 Byte
9	8 Byte	12 Byte
10	8 Byte	16 Byte
11	8 Byte	20 Byte
12	8 Byte	24 Byte
13	8 Byte	32 Byte
14	8 Byte	48 Byte
15	8 Byte	64 Byte

Table. CAN FD통신에서 DLC 필드에 따른 데이터 필드 길이

CAN 통신 계층 구조 – 데이터 링크 계층 (2계층)

- 중재 (Arbitration)
  - 둘 이상의 노드가 동시에 송신을 요청하는 경우
    →ID 우선순위가 높은 송신만 이루어짐
  - Identifier 전송하면서 피드백 확인
  - Recessive 전송 & Dominant 피드백 시
    →신호 전송 중지

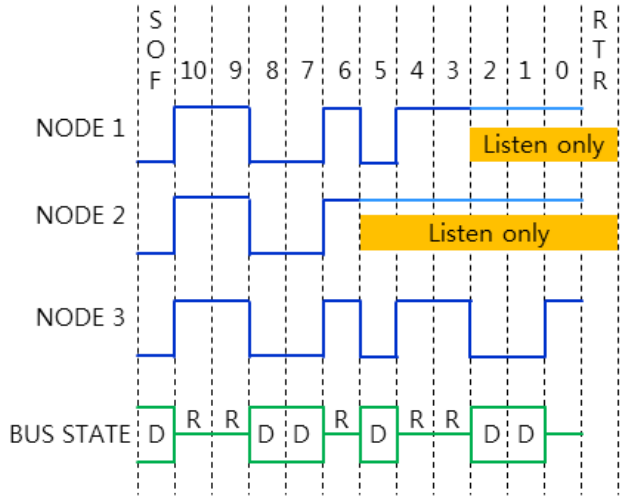


Fig. CAN 통신 Arbitration 과정

- 비트스터핑 (Bitstuffing)
  - 모든 제어기가 동일한 클락을 사용하지 않음
    →동기화 필요
  - Falling edge(R→D) 발생 시 동기화 수행
  - 5 bit 이상 동일한 데이터 전송 시 추가 bit삽입

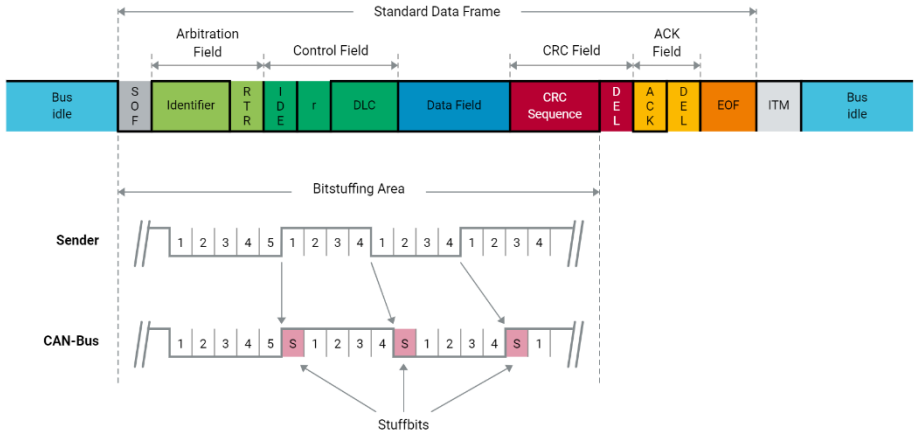


Fig. Bitstuffing 적용 조건 및 전 · 후 비교

Contents

Interaction Layer

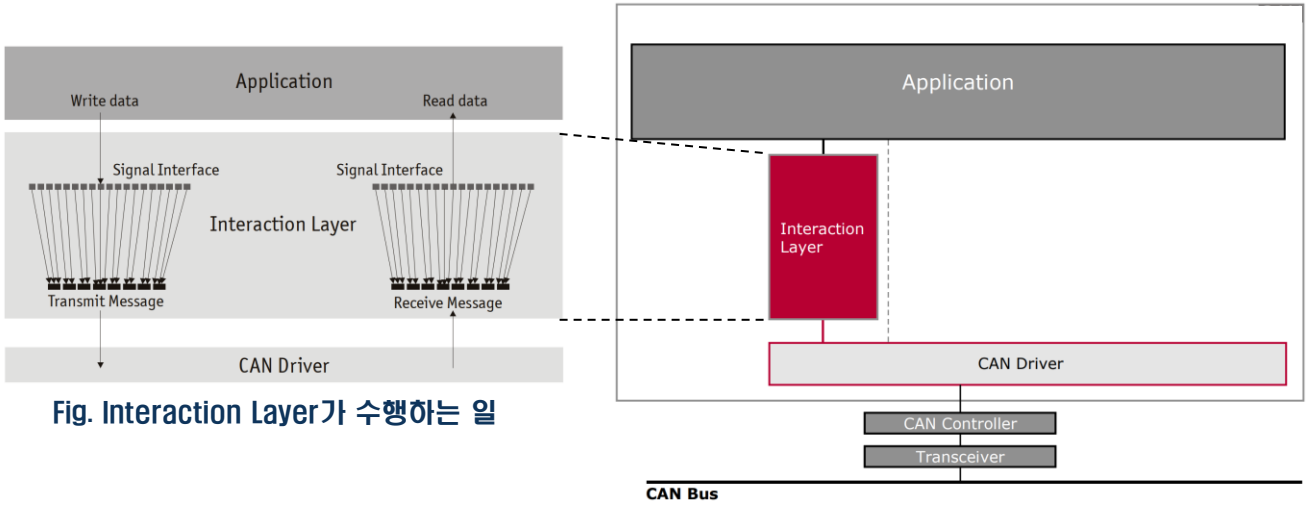


Fig. Interaction Layer가 수행하는 일

Fig. Application, Interaction Layer 및 CAN 드라이버 구조

VECTOR

Fig. Interaction Layer를 지원하는 VECTOR 社

- **Interaction Layer**
  - **역할:**
    - CAN message 송 · 수신 및 signal 변환
    - CAN message 송출방법 제어 가능
    - 전송 관련 속성 설정 가능
  - **사용 목적:**
    - CAN 드라이버를 직접 제어하는 것이 아니라, 한 단계 상위 계층인 Interaction Layer를 제어함으로써 개발 효율성을 높임
  - **Message**
    - 프레임 구조 특성 상 0~8 Byte의 크기를 가질 수 있음
    - CAN 통신에서 실제 데이터가 전송되는 단위
  - **Signal**
    - Application에서 전송하거나 수신받을 정보 단위 (bit 단위)
    - CAN 통신에서는 Message 단위로 전송하므로, 어떤 Message의 어떤 위치에 해당 Signal이 할당되는지 설정해야 함

DBC 파라미터

- **Message**
  - Name
  - Node
  - ID
  - ID-Format
  - DLC [Byte]
  - GenMsgILSupport
  - GenMsgDelayTime
  - GenMsgCycleTime
  - GenMsgSendType

- **Signal**
  - Name
  - Message
  - Startbit
  - Length [Bit]
  - Byte Order: Intel / Motorola
  - Value Type: Signed / Unsigned / IEEE Float / IEEE Double
  - Initial Value
  - Factor, Offset
    - Float형 데이터를 int형으로 변환함으로써 전송 효율을 높일 수 있음
    - [실제값] = [수신데이터]×[Scaling Factor] + [Offset]
  - Minimum, Maximum
  - GenSigSendType

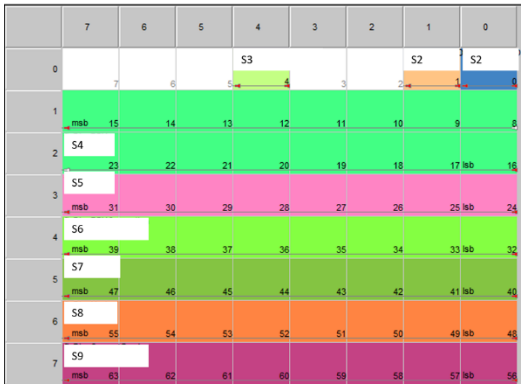


Fig. Message 내 Signal 배치 레이아웃

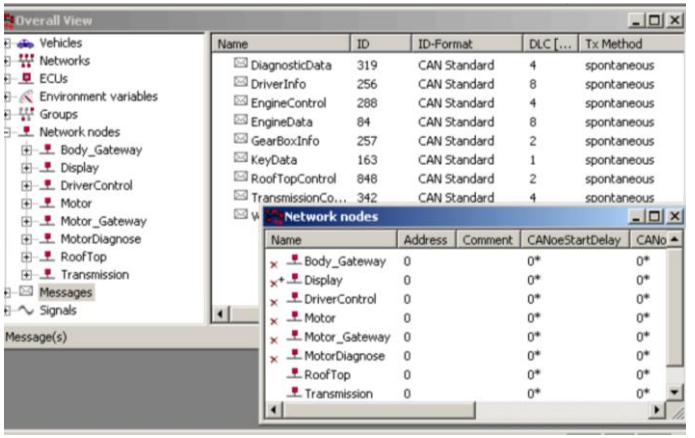


Fig. CAN db 설정화면

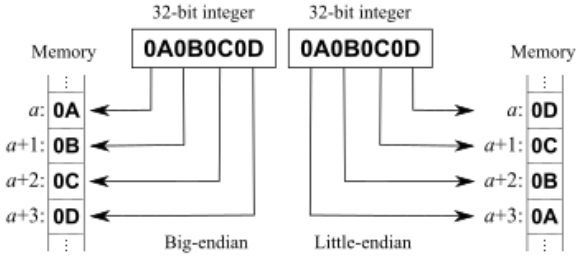
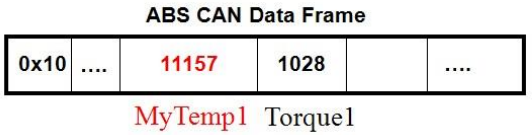


Fig. 빅엔디안, 리틀엔디안 메모리 저장 방법



Scaling Factor = 0.0312
Offset = -273
Units = °C
MyTemp1 = 11157 x 0.0312 - 273 = 75.1 °C

Fig. Factor, Offset 이용한 데이터 변환 예시

Interaction Layer – CAN 메시지 송출 방법

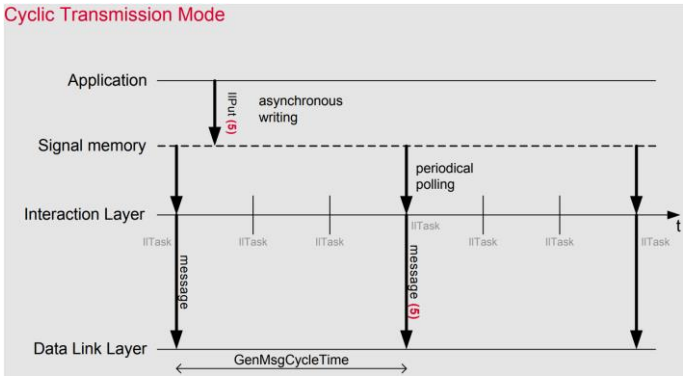


Fig. Cyclic 전송 모드 동작

전송 모드: Cyclic

- 설정방법
  - GenSigSendType: Cyclic
  - GenMsgSendType: NoMsgSendType (or Cyclic)
- 동작: 일정 주기마다 전송
  - GenMsgCycleTime 설정 필요
  - 변동이 잦은 값을 모니터링할 때 적합
    →Ex) 엔진 RPM 정보 전송
  - 주기 시간 동안 2번 이상 데이터 변화 시, 손실 발생

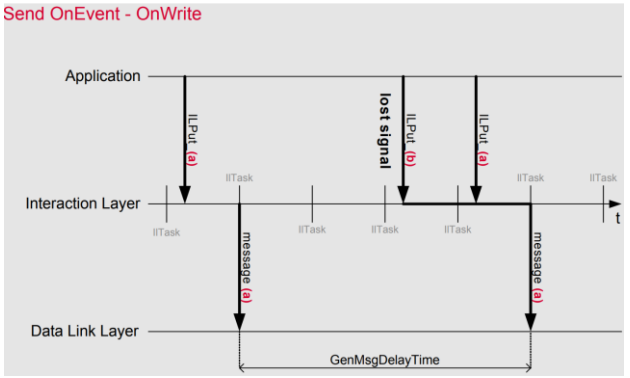


Fig. OnWrite 전송 모드 동작

전송 모드: OnWrite

- 설정방법
  - GenSigSendType: OnEvent
  - GenMsgSendType: NoMsgSendType
- 동작: ILPut함수 호출 시 전송
  - GenMsgDelayTime 설정 필요
  - 딜레이 시간 동안 2번 이상 데이터 변화 시, 손실 발생

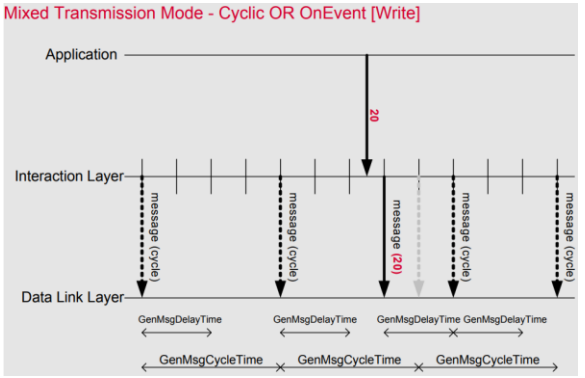


Fig. Cyclic + OnWrite 전송 모드 동작

전송 모드: Cyclic + OnWrite

- 설정방법
  - GenSigSendType: OnEvent
  - GenMsgSendType: Cyclic
- 동작: 일정 주기마다 전송하되, ILPut함수 호출 시 추가 전송
  - GenMsgCycleTime 및 GenMsgDelayTime 설정 필요

감사합니다.