GREENTECH SYSTEM

CAN 통신

1. CAN 통신이란
2. CAN 통신프로그램
3. CAN 통신을 하면서 발생하였던문제점

- (주) 그린텍 시스템 개발 2과 -

CAN 통신이란

1. 시작

CAN이란 무엇인지 CAN 통신은 어떤 식으로 구성되어 있는지 알아본다.

1. CAN 통신이란

1-1. 개요

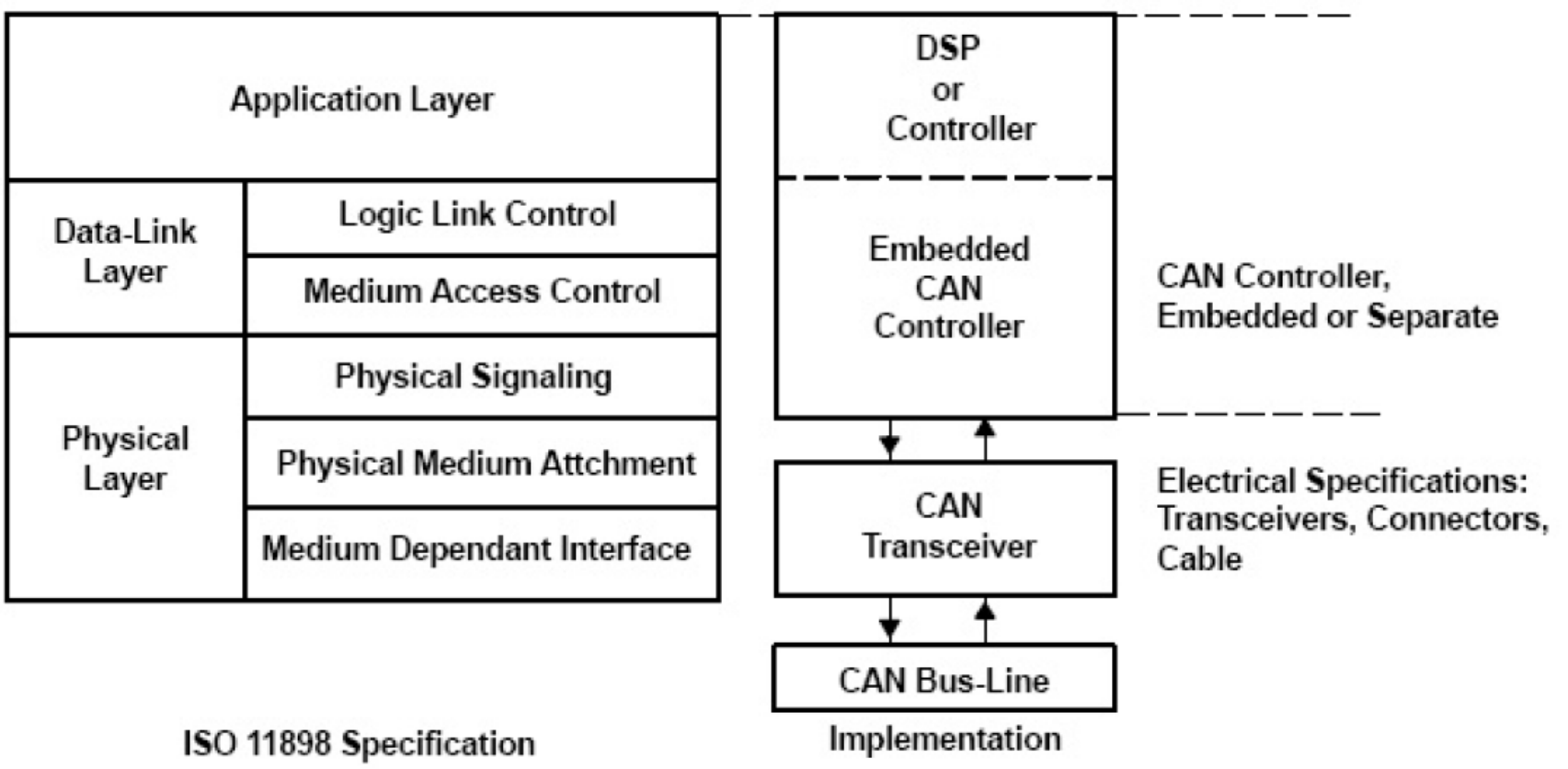
- CAN통신은 “Controller Area Network”의 약자로 여러개의 ECU(Electroinc Control Unit)를 병렬로 연결하여 데이터를 주고받는 통신방법을 말한다.

- 통신 선은 2개의 버스를 가지며 통신선상에 데이터를 띄어놓고 필요한 데이터를 가져다 사용하는 방식이다.

- CAN통신은 두 개의 버스 전압차이로 데이터를 읽기 때문에 만약 두 개중 하나라도 단선이 된다면 통신이 불가하다는 단점이 있다.

- CAN 통신은 ISO 11898 통신방식을 채택하고 있다.

1. CAN 통신이란

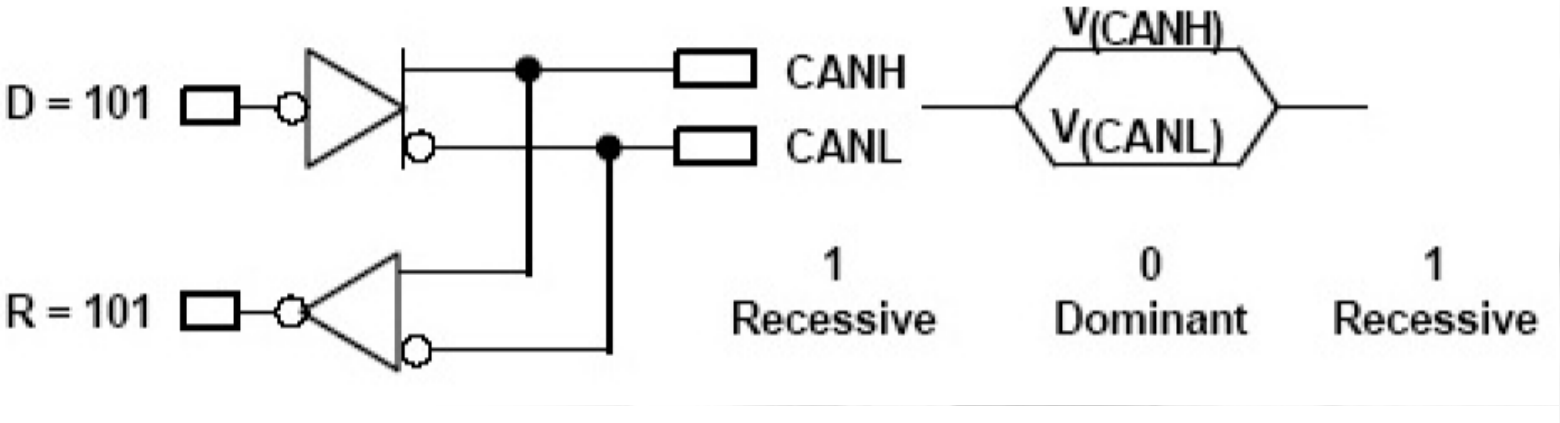

Application Layer
Data-Link Layer
Logic Link Control
Medium Access Control
Physical Layer
Physical Signaling
Physical Medium Attchment
Medium Dependant Interface
ISO 11898 Specification
DSP or Controller
Embedded CAN Controller
CAN Controller, Embedded or Separate
CAN Transceiver
Electrical Specifications: Transceivers, Connectors, Cable
CAN Bus-Line
Implementation

1. CAN 통신이란

1-2. CAN 통신의 이해

1-2-1. 데이터 라인 구조

- CAN 통신에서는 데이터 통신을 할 때 CAN H, CAN L 두 가닥을 이용해서 정보를 처리 한다.
- 즉 전위차가 있느냐 없느냐로 데이터의 논리 값 “0” 과 “1”을 구분한다.



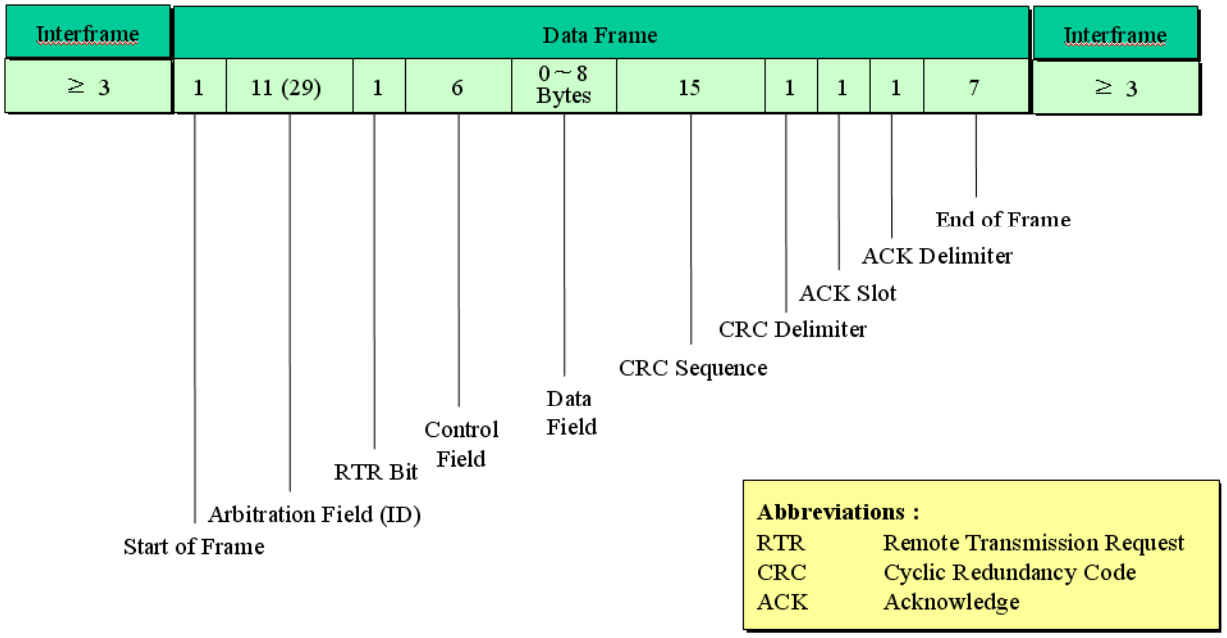
1. CAN 통신이란

1-2-2. 속도

◆ CAN 통신은 최대 11M bps에 달하는 고속 통신을 제공한다.

◆사용자 선택에 따라 50K bps 또는 그 이하의 속도도 설정 가능 하다.

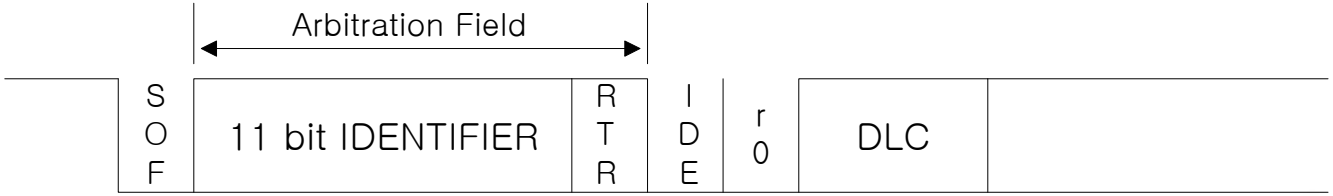
1-2-3. 데이터의 구조



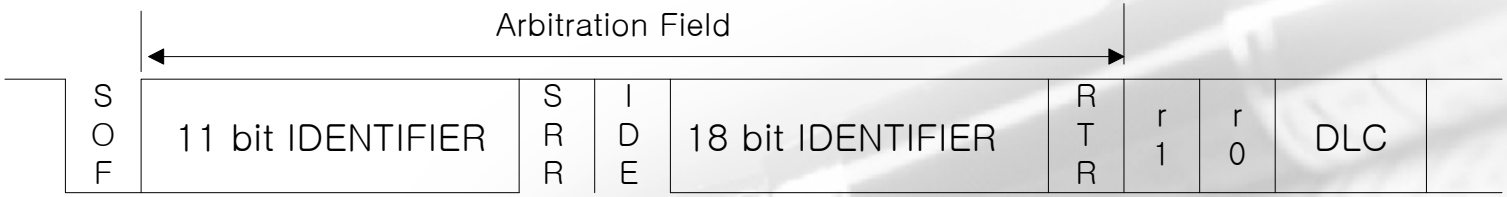
1. CAN 통신이란

Arbitration Field 구조

- CAN 2.0 Part A – 11bit arbitration field

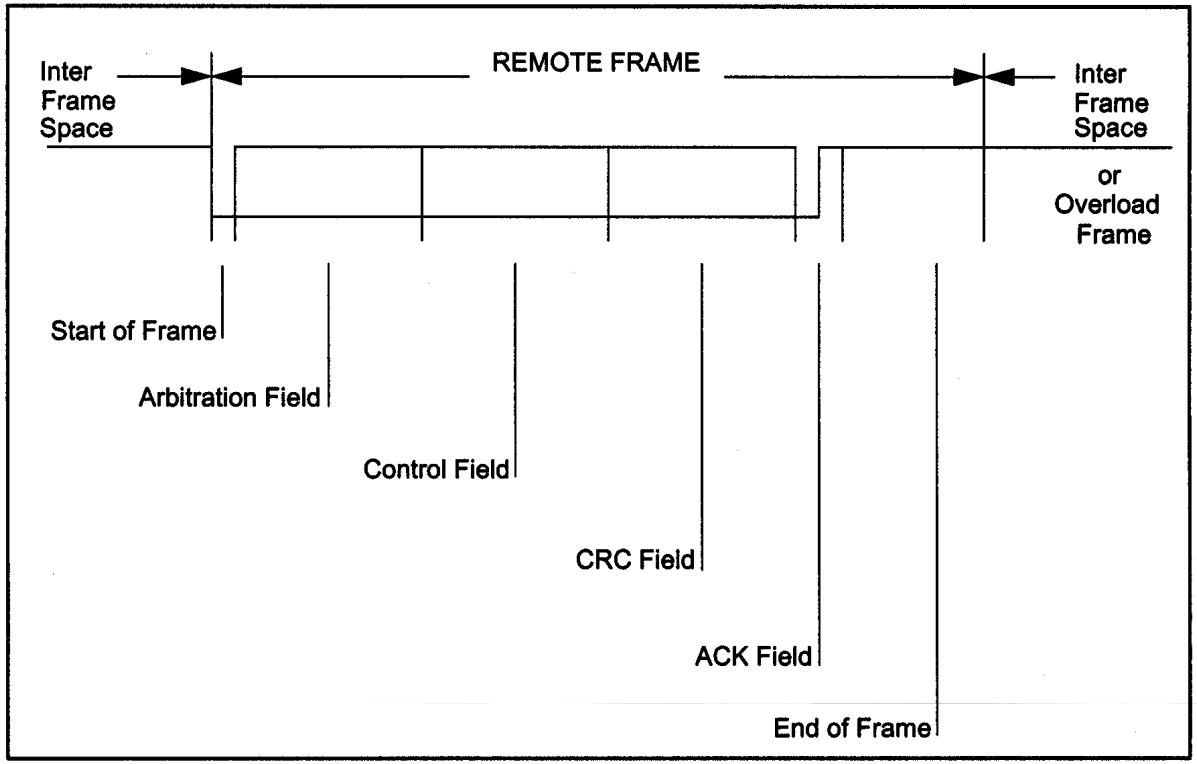


– CAN 2.0 Part B – 29bit extended arbitration field



Remote Frame

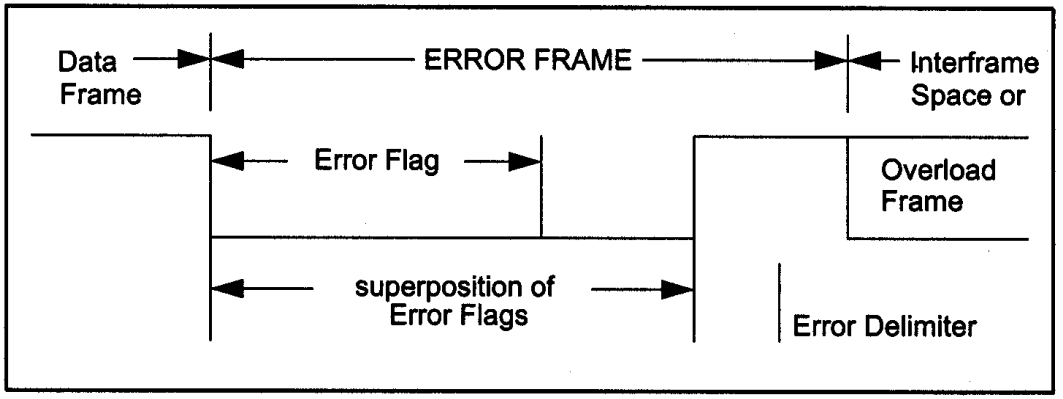
- Receiver가 transmitter에 data를 요청할 때 출력하는 frame
- Arbitration field에는 transmitter의 ID가 포함되며 data field가 없는 것이 특징



1. CAN 통신이란

Error Frame

- Error 상황을 알리기 위한 frame
- Error를 감지한 node가 6 dominant bit를 출력하면
- 다른 node는 이에 반응 (stuff error)하여 6 dominant bit를 출력
- 6~12 dominant bits (Error Flag) + 8 recessive bits (Error delimiter)로 구성
- Superposition(중첩) : Node들이 출력하는 error flag의 순서에 따라서 중첩될 수 있음
- Error Delimiter : Error frame의 종료. 8 recessive bit로 구성



Overload Frame

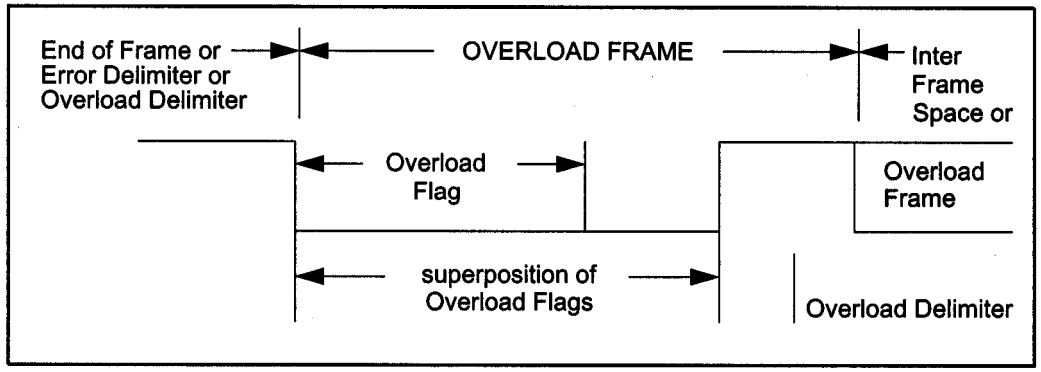
- 다음의 상황에서 overload frame을 전송 (bus 상태를 안정화하기 위해서)

  1) Receiver의 내부 조건에 의해서 다음 data/remote frame 사이에 delay가 필요한 경우

  2) Frame 사이의 intermission의 1,2 bit가 dominant bit인 경우

  3) Error delimiter 또는 overload delimiter의 마지막 bit가 dominant bit 인 경우

- 6 dominant bits (Overload Flag) + 8 recessive bits (Overload delimiter)로 구성



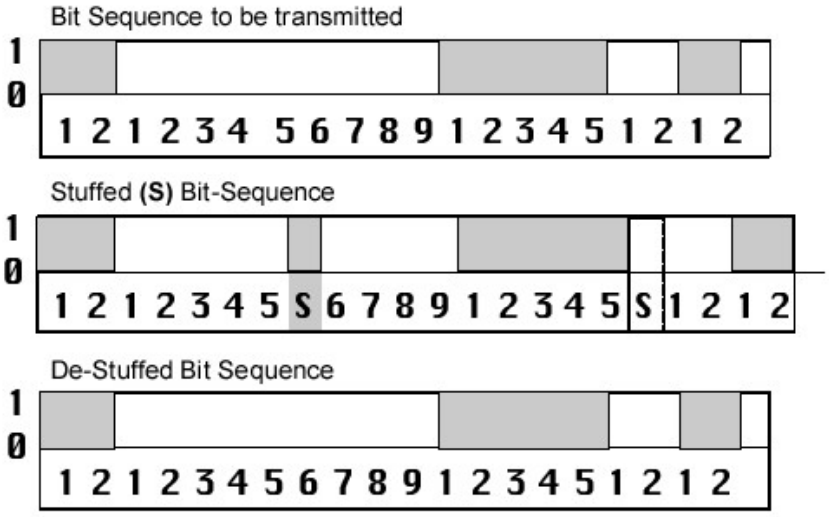
Bit Stuffing

- 전송할 내용이 같은 bit가 5개 연속이면 6번째 bit는 반대의 bit를 끼워넣음

  1) 비동기식 통신이므로 각 bit의 check 시점이 달라지는 오차를 줄이기 위한 방법

  2) Transmitter에 의해 만들어지며, receiver는 이를 빼고 해석

  3) Bit Stuff Area : SOF, Arbitration Field, Control field, Data field, CRC field



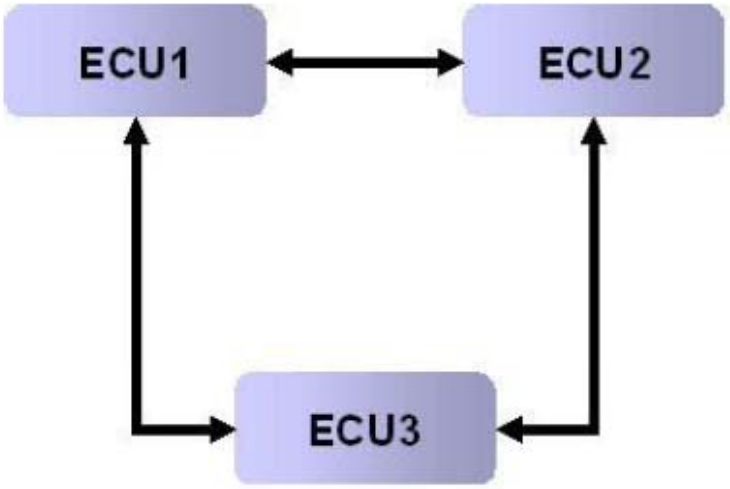
1-3. CAN 통신의 분류

1-3-1. CAN 통신은 크게 HIGH CAN 과 LOW CAN으로 나뉘어 진다

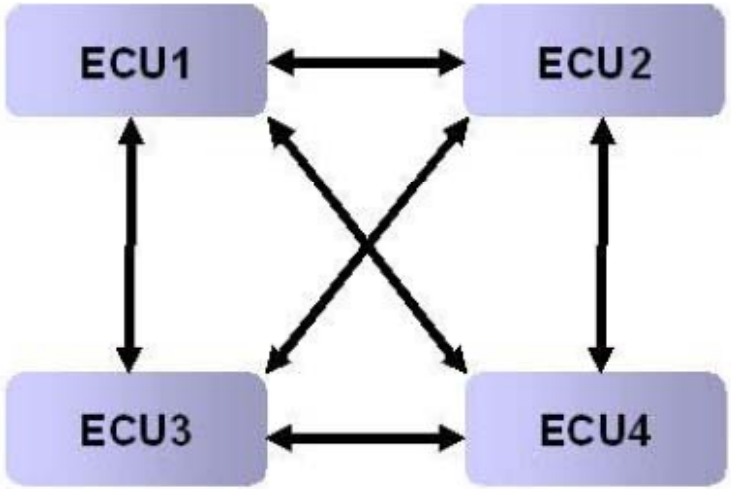
- ◆ HIGH CAN
  - ▪ 2.5V ~ 3.5V 전압을 이용한다.
  - ▪ 200 Kbit/Sec 이상의 고속 통신을 한다.

- ◆ LOW CAN
  - ▪ 1.5V ~ 2.5V 전압을 이용한다.
  - ▪ 33.33 Kbit/Sec 이상 에서 125Kbit/Sec 이하의 저속 통신을 한다.

1-4. CAN 통신의 필요성



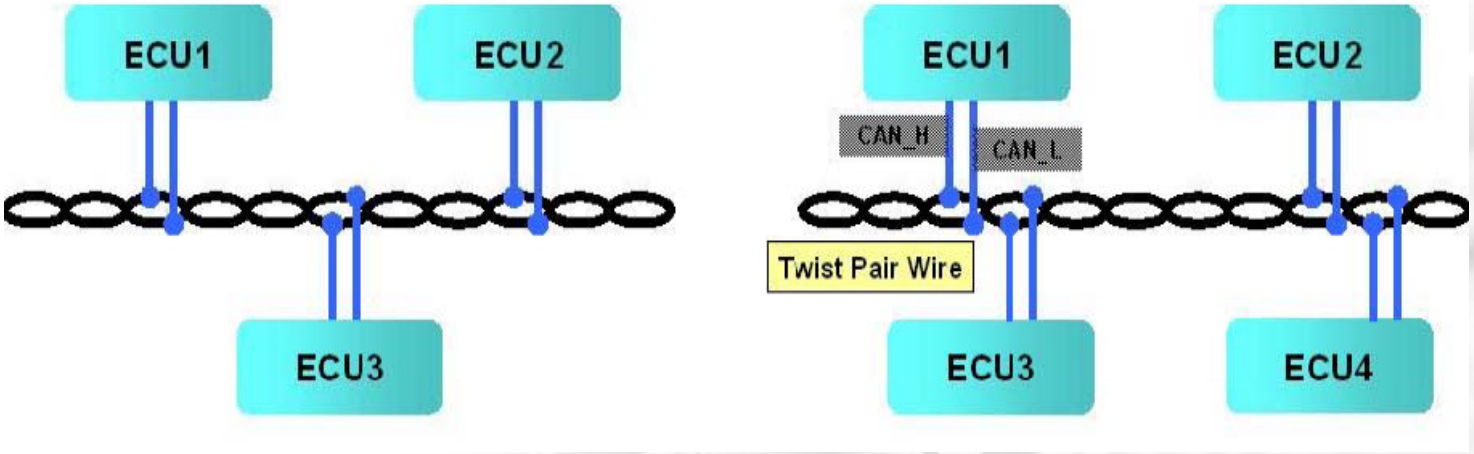
< UART Communication >



< UART Communication >

1. CAN 통신이란

- 위 그림에서 보는 것과 같이 CAN 통신이 등장하기 전 Device 간의 통신은 1 : 1 방식이었다.
- 단순하게 Device를 두고 필요한 배선을 연결만 하면 통신환경이 구성되는 아주 간단한 구조였다.
- 그러나 기술의 발전에 따라 Deivce의 개수가 늘어나게 되었으며 Device가 늘어남에 따라 Wire의 수는 (n^2 – n) / 2 –- 계수의 -- 개수 만큼 늘어나게 되었다.
- 위의 문제를 해결하기 위해 여러 가지 통신 방식이 만들어지게 되었는데 그 중 하나가 CAN 통신 방식이다



1. CAN 통신이란

- 위 그림은 CAN 통신 방식의 Block Diagram 이다.
- 위 Block Diagram을 보면 알 수 있듯이 CAN BUS를 통해 Device들이 연결되어 있는 구조를 보인다.
- 이로 인해 물리적 환경이 간소해지는 장점이 있다.
- 또한 CAN BUS상에 Device를 추가하여 바로 사용할 수 있는 구조이다.

1-5. CAN 통신의 장점

1-5-1. CAN 프로토콜은 Multi Master 통신을 한다.

통신버스를 공유하고 있는 CAN컨트롤러들은 모두가 Master역할을 하여 언제든지 버스를 사용하고 싶을 때 사용할 수 있다.

1-5-2. 노이즈에 매우 강하다.

레벨은 다르지만 RS485와 비슷하게 twist pair 2선을 사용하여 전기적 Differential 통신을 하여 전기적인 노이즈에 매우 강하다

1-5-3 표준 프로토콜이므로 시장성이 뛰어나다.

이로 인해 많은 업체들이 경쟁적으로 CAN칩을 제작하고 있으며 비용 또한 비교적 저렴하다. CAN 프로토콜은 호스트 CPU에 내장된 CAN 컨트롤러 칩이나, 호스트 CPU에 장착된 CAN 주변장치에서 실행된다.

1-5-4 통신속도가 빠르다.

최대 1 Mbps까지 사용할 수 있다.

1-5-5 먼 거리를 통신할 수 있다.

최대 1000 m 까지도 40 kbps로 통신할 수 있다.

1-5-6 하드웨어적인 오류보정이 있다.

하드웨어적으로 설정된 ID만을 골라 수신 받을 수 있다.

CAN에는 수신필터가 있어 필터를 어떻게 설정하느냐에 따라 정해진 ID, 특정 그룹 또는 전체 수신을 할 수 있다.

1. CAN 통신이란

1-5-7 기타

실시간 메시지 통신을 할 수 있다.
한번에 8개의 데이터를 전송하는 HW 패킷을 제공한다. 보통 RS232/RS485
통신에서는 패킷 통신을 위해 사용자가 일일이 패킷 형식을 만들어 주고
수신 받을 때도 그런 해석이 필요하지만 CAN은 8바이트 데이터를 담는
HW 패킷 통신을 기본으로 한다. 따라서 사용자는 데이터 버퍼에 데이터를 쓰고
전송만 하면 그 외 모든 처리는 HW가 알아서 한다. 분산제어 분야 적용이 용이하다.
우선순위가 있다.
사용되는 전선의 양을 획기적으로 줄일 수 있다.
PLUG & PLAY 를 제공한다.

2. CAN 통신 프로그램

이 장에서는 통신 프로그램에 관해서 설명하며 통신 프로그램은 VQ 최종 검사기에 사용된 프로그램을 구성되어 있다.

2. CAN 통신프로그램

2-1. 통신 프로그램의 구성

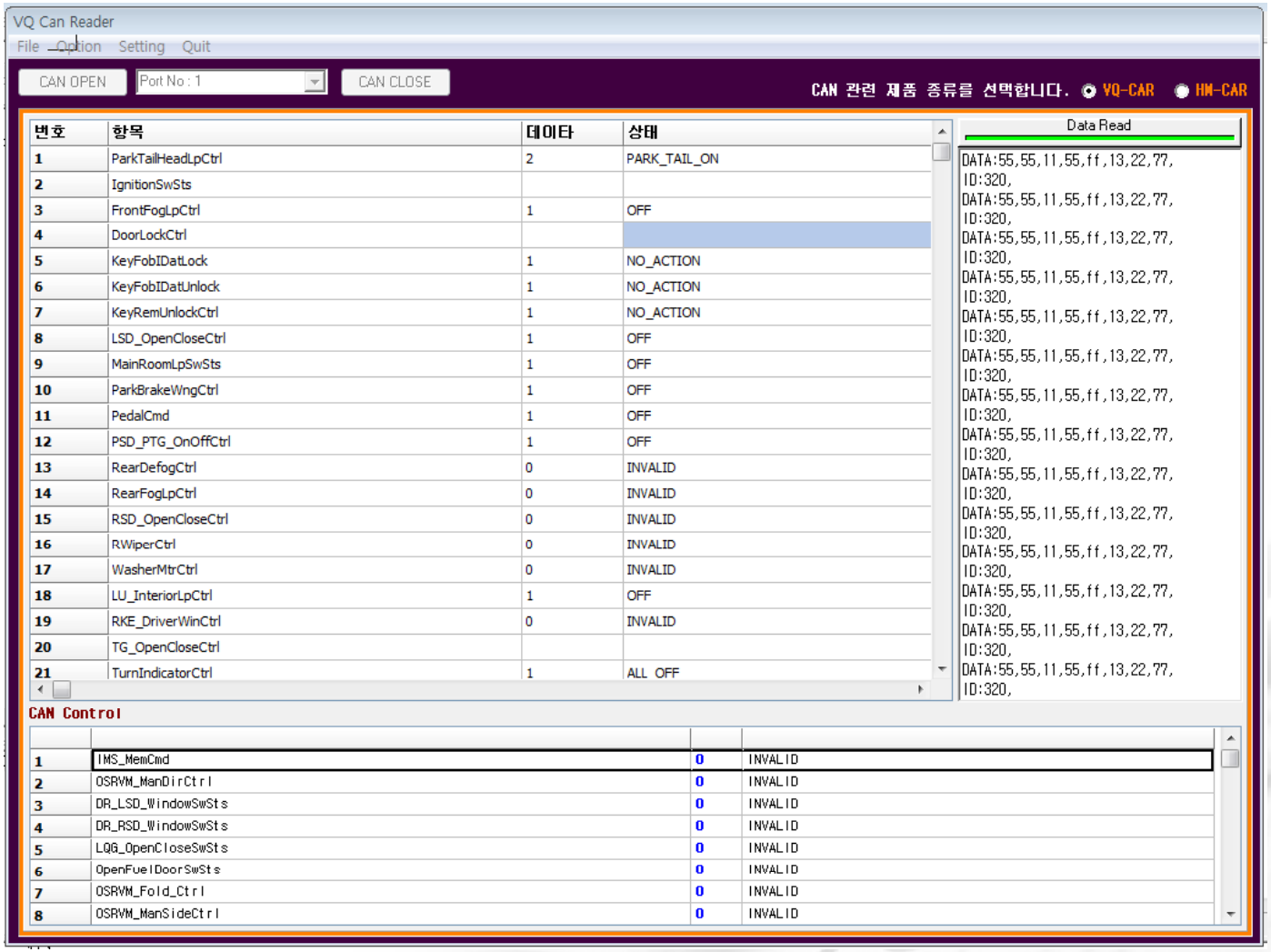
2-1-1. 구성

- ◆ 통신 프로그램은 크게 두 가지로 구성되어 진다.
- ◆ 첫 번째는 실제로 통신을 하면서 입력 과 출력을 관장하는 프로그램이다.
- ◆ 두 번째는 CAN의 여러가지 항목들은 등록하는 프로그램이다.

2-1-2. 기능별 설명

- ◆ CAN 입력 과 출력
  - ▪ CAN 보드에 출력을 보내 거나 입력을 받아 어떤 데이터가 들어왔는지 분석하고 판정 하며 사용자에게 보여주는 기능을 한다.
  - ▪ 위 입/출력 제어는 통신 프로그램을 작성하는 목적이기도 하다.
  - ▪ 입력을 받는 것은 크게 어렵지는 않으나 이 받은 데이터를 어떻게 가공하여 사용자에게 정확하게 보여주고 쉽게 받은 데이터를 가지고 제품을 분석 할 수 있게 하느냐가 이 프로그램을 키 포인트 이다.
  - ▪ 아무리 화려하고 보기 좋게 화면을 구성할 지라도 입력된 데이터를 정확하게 분석하지 못하고 사용자로 하여금 데이터의 신뢰를 줄 수 없다면 통신 프로그램은 그 의미를 잃어 버리게 될 것이다.
  - ▪ 마찬가지로 출력 또한 중요하다 원하는 출력을 제품에 주어 동작을 시켜야 하는데 기 능에 충실하지 못하면 사용할 수 없는 통신 프로그램이 될 것이다.
  - ▪ 그 만큼 통신을 하는 것 보다 그 데이터를 편집하고 가공하는 작업이 CAN 통신 프로그램에서는 중요하고 어려운 } 작업이다.

2. CAN 통신프로그램

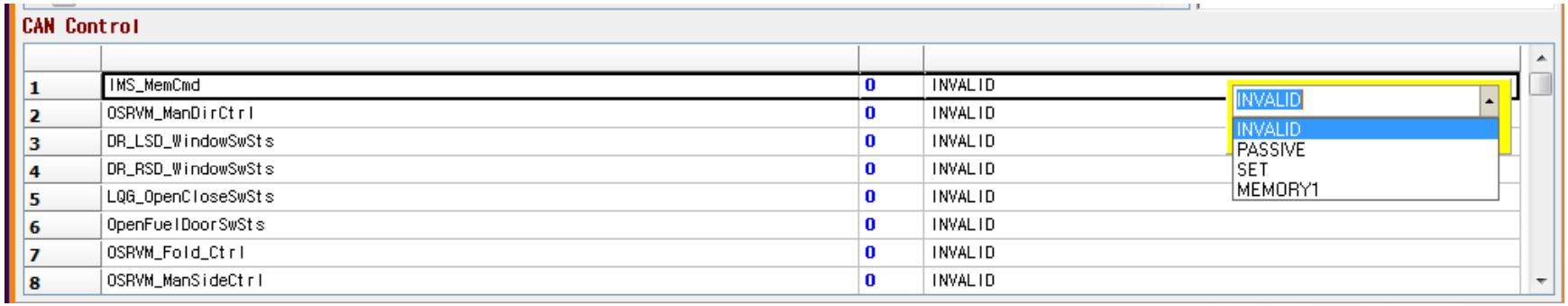


번호	항목	데이타	상태
1	ParkTailHeadLpCtrl	2	PARK_TAIL_ON
2	IgnitionSwSts		
3	FrontFogLpCtrl	1	OFF
4	DoorLockCtrl		
5	KeyFobIDatLock	1	NO_ACTION
6	KeyFobIDatUnlock	1	NO_ACTION
7	KeyRemUnlockCtrl	1	NO_ACTION
8	LSD_OpenCloseCtrl	1	OFF
9	MainRoomLpSwSts	1	OFF
10	ParkBrakeWngCtrl	1	OFF
11	PedalCmd	1	OFF
12	PSD_PTG_OnOffCtrl	1	OFF
13	RearDefogCtrl	0	INVALID
14	RearFogLpCtrl	0	INVALID
15	RSD_OpenCloseCtrl	0	INVALID
16	RWiperCtrl	0	INVALID
17	WasherMtrCtrl	0	INVALID
18	LU_InteriorLpCtrl	1	OFF
19	RKE_DriverWinCtrl	0	INVALID
20	TG_OpenCloseCtrl		
21	TurnIndicatorCtrl	1	ALL OFF

```
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
DATA:55,55,11,55,ff,13,22,77,
ID:320,
```

1	IMS_MemCmd	0	INVALID
2	OSRVM_ManDirCtrl	0	INVALID
3	DR_LSD_WindowSwSts	0	INVALID
4	DR_RSD_WindowSwSts	0	INVALID
5	LQG_OpenCloseSwSts	0	INVALID
6	OpenFuelDoorSwSts	0	INVALID
7	OSRVM_Fold_Ctrl	0	INVALID
8	OSRVM_ManSideCtrl	0	INVALID

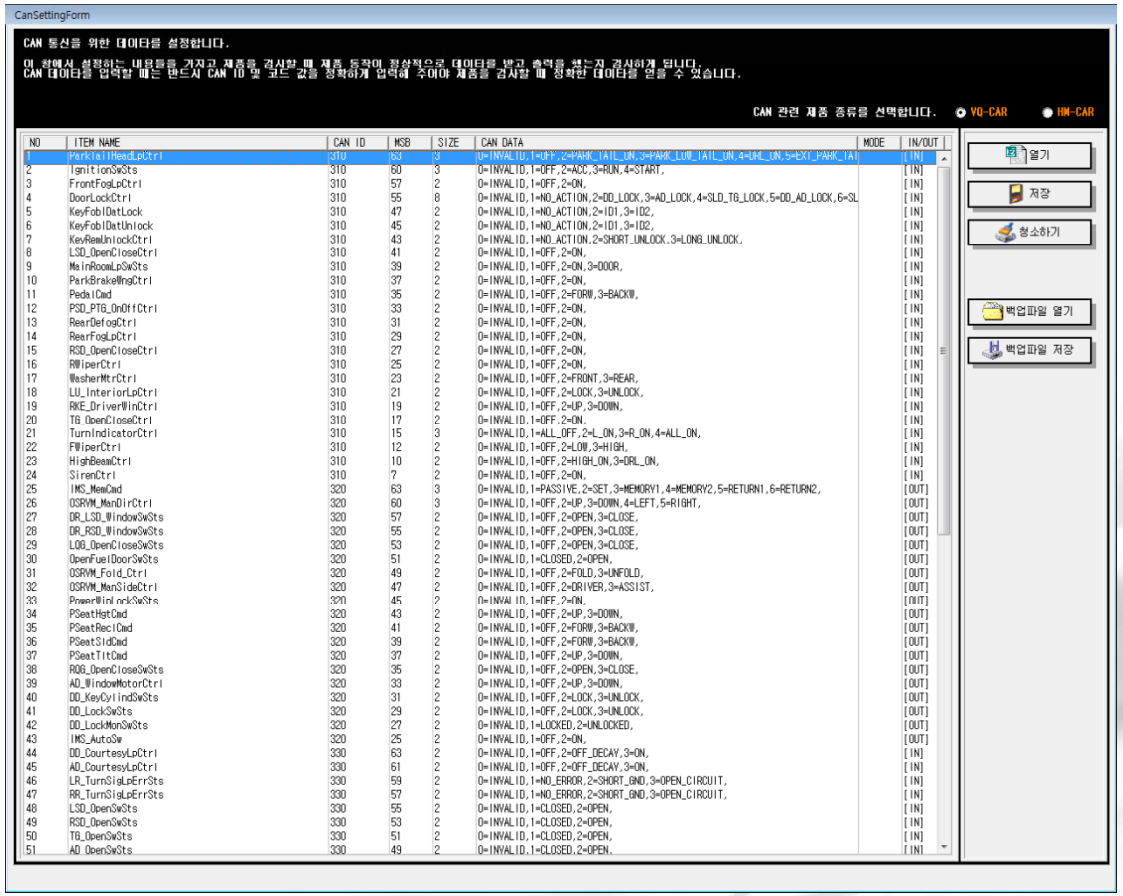
2. CAN 통신프로그램


CAN Control
1 IMS_MemCmd 0 INVALID
2 OSRVM_ManDirCtrl 0 INVALID
3 DR_LSD_WindowSwSts 0 INVALID
4 DR_RSD_WindowSwSts 0 INVALID
5 LQG_OpenCloseSwSts 0 INVALID
6 OpenFuelDoorSwSts 0 INVALID
7 OSRVM_Fold_Ctrl 0 INVALID
8 OSRVM_ManSideCtrl 0 INVALID
INVALID
INVALID
PASSIVE
SET
MEMORY1

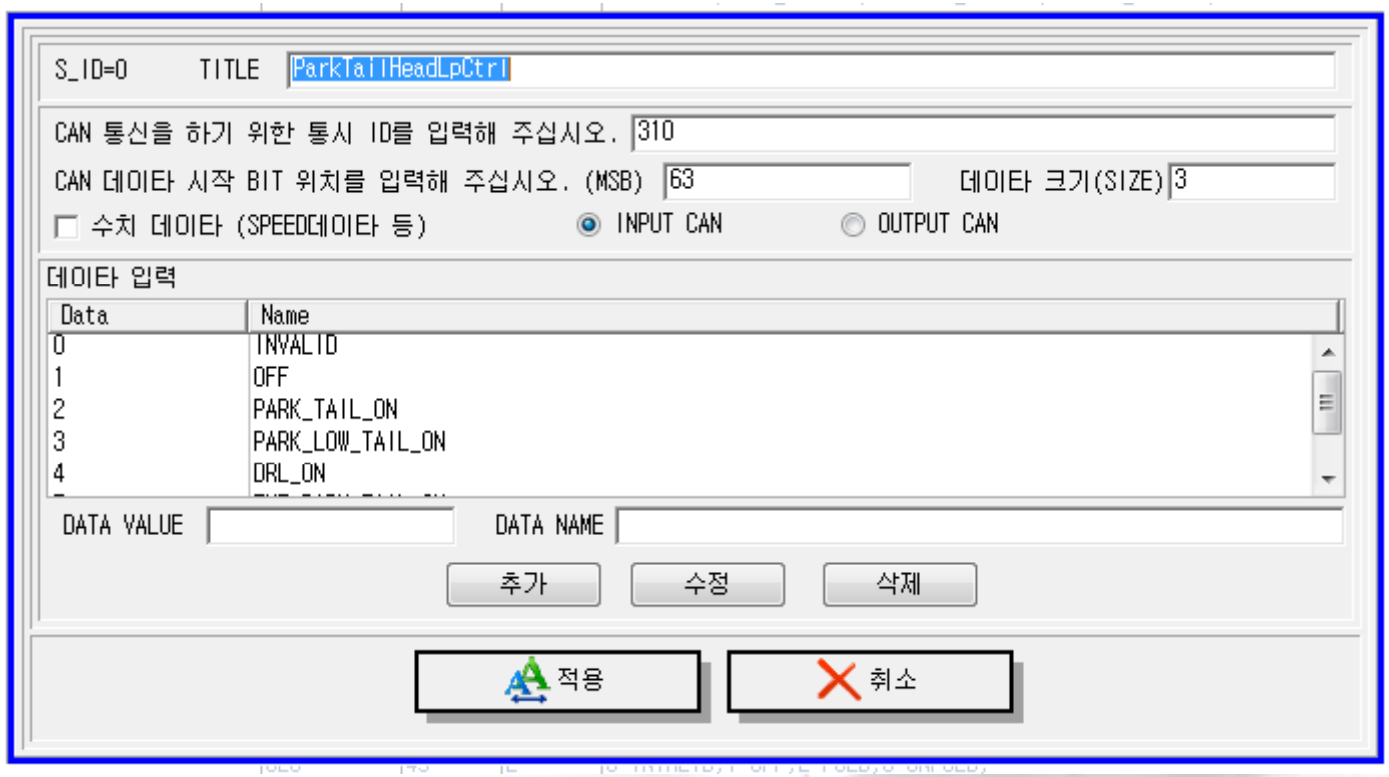
2. CAN 통신프로그램

◆ **CAN 항목 등록하기**

- 아무리 통신 프로그램을 잘 만들었다 하더라도 통신 관련 항목들을 설정할 수 없다면 소용없을 것이다.
- 통신 항목을 등록할 때 아이디 및 데이터의 항목 별 데이터 와 타이틀 등을 등록하여 통신을 할 때 이 설정 데이터를 이용하여 출력을 보낸다던가 입력을 분석하는 데 사용하게 된다.



2. CAN 통신프로그램


S_ID=0
TITLE
ParkTailHeadLpCtrl
CAN 통신을 하기 위한 통시 ID를 입력해 주십시오.
310
CAN 데이타 시작 BIT 위치를 입력해 주십시오. (MSB)
63
데이타 크기(SIZE)
3
수치 데이타 (SPEED데이타 등)
INPUT CAN
OUTPUT CAN
데이타 입력
Data
Name
0
INVALID
1
OFF
2
PARK_TAIL_ON
3
PARK_LOW_TAIL_ON
4
DRL_ON
DATA VALUE
DATA NAME
추가
수정
삭제
적용
취소

2-1-3. 통신 프로그램 소스 분석

◆ **READ** 한 통신 데이터 분석 및 표시

```
for(i1 = 0;i1 < CanItem.ItemCounter;i1++)
{
    if(CanItem.Item[i1].InOut == true) // 입력 can 일경우
    {
      if(CanRead.Addr == CanItem.Item[i1].CanID) // 입력된 CAN 아이디와 내가 원하는 CAN의 아이디가 일치 하는지 확인
      {
        xData = 0x00;
        i2 = 0;
        yData = 0x00;
        for(i2 = 0;i2 < CanItem.Item[i1].Size;i2++) yData |= HEX_DATA[i2]; // 마스킹을 위한 데이타를 구한다.
       // xData 변수에 원하는 위치에서 데이터를 축출하는 작업을 하여 데이터를 준다.
        xData = CanRead.Data[7 - (CanItem.Item[i1].StartBit / 8)] >> ((CanItem.Item[i1].StartBit % 8) –
                        (CanItem.Item[i1].Size - 1));
        xData &= yData; // 쓸데없는 데이타 마스킹한다.
        for(i2 = 0;i2 < CanItem.Item[i1].DataCounter;i2++)
        {
          if(CanItem.Item[i1].Data[i2].Data == xData) // 읽혀진 데이터가 어떤  의미를 지닌것 인지 화면에 표시해 준다.
          {
            CanDisplayForm->CanDis1->Cells[1][Row] = xData;
            CanDisplayForm->CanDis1->Cells[2][Row] = CanItem.Item[i1].Data[i2].Title;
            CheckCanData.Data[i1].Data[i2] = true;
            break;
          }
        }
      }
      Row++;
    }
}
```

0X01, 0X02, 0X04, 0X08, 0X10, 0X20, 0X40, 0X80

2. CAN 통신프로그램

◆ **CAN 데이터 출력**

```
for(Count = 0;Count < CanItem.ItemCounter;Count++) // 출력을 내 보내기 를 원하는 항목을 찾기 위해 루프를 돈다.
{
    Flag = false;
    if(DefaultCanData.Can[Count].Check == true) // 출력을 보내기 원하는 CAN 항목일 경우
    {
       for(j = 0;j < CanItem.ItemCounter;j++)
       {
          if(DefaultCanData.Can[Count].ID == CanItem.Item[j].S_ID) // 선택한 항목과 CAN 등록 항목이 같을 경우
          {
             Flag = true;
             ID = CanItem.Item[j].CanID; // CAN 아이디 저장

             Msb = CanItem.Item[j].StartBit; // 데이터 시작 위치 저장
             Length = CanItem.Item[j].Size; // 데이터 길이 저장
             break;
          }
       }

       if(Flag == true) // 출력 항목을 찾았으면 아래의 작업을 진행한다.
       {
          Flag = false;
          for(i = 0;(i < TCanIO.Counter) && (i < 50);i++) // 현재 출력 제어가 진행된 ID 수 만큼 루프를 돌아 같은
          {                                               // ID가 존재하는지 찾는다.
             if(ID == TCanIO.Can[i].ID) // 출력을 보내고자 하는 ID와 동일한 ID가 존재한다면 아래 작업진행
             {
```

CAN ID 및 데이터가 시작 위치 , 길이 등을 알고 있어야만 원하는 위치에 원하는 데이터를 출력할 수 있다.
CAN은 한 개의 주소와 **8 BYTE**의 데이터 구조를 같는 관계로 위치를 찾아서 원하는 **BIT**에 데이터를 등록해 주어야 한다.

2. CAN 통신프로그램

```
        Masking = 0x00;
        switch(Length)
        {
           case 1 : Masking = 0x01; break;
           case 2 : Masking = 0x03; break;
           case 3 : Masking = 0x07; break;
           case 4 : Masking = 0x0f; break;
           case 5 : Masking = 0x1f; break;
           case 6 : Masking = 0x3f; break;
           case 7 : Masking = 0x7f; break;
           case 8 : Masking = 0xff; break;
        }
        TCanIO.Can[i].Data[7 - ((Msb - (Length - 1)) / 8)] &= ~(Masking << ((Msb - (Length - 1)) % 8)); // MASKING 작업
        // 출력할 데이터를 등록하는 작업을 진행한다.
        TCanIO.Can[i].Data[7 - ((Msb - (Length - 1)) / 8)] |= DefaultCanData.Can[Count].Item << ((Msb - (Length - 1)) % 8);
        Flag = true;
        break;
     }
  }
  if(Flag == false)
  {
     TCanIO.Can[TCanIO.Counter].ID = ID;
     Masking = 0x00;
     switch(Length)
     {
        case 1 : Masking = 0x01; break;
        case 2 : Masking = 0x03; break;
        case 3 : Masking = 0x07; break;
        case 4 : Masking = 0x0f; break;
```

MASKING 작업은 **CAN** 출력을 내 보내는 작업 만큼이나 중요한 작업이다.
해당 위치에 어떤 값이 들어가 있을 경우 **MASKING** 작업에 따라 아주 이상한 값이 출력 될 수도 있기 때문에 데이터를 길이만큼 어떤 데이터로 **MASKING**을 해야할 것인지를 구해서 **MASKING** 후 데이터를 출력해 주어야 한다.

```
                case 5 : Masking = 0x1f; break;
                case 6 : Masking = 0x3f; break;
                case 7 : Masking = 0x7f; break;
                case 8 : Masking = 0xff; break;
              }
              TCanIO.Can[TCanIO.Counter].Data[7 - ((Msb - (Length - 1)) / 8)] &= ~(Masking << ((Msb - (Length - 1)) % 8));

              TCanIO.Can[TCanIO.Counter].Data[7 - ((Msb - (Length - 1)) / 8)] |= DefaultCanData.Can[Count].Item <<
                                    ((Msb - (Length - 1)) % 8);
              TCanIO.Counter++;
            }
          }
        }
      }
```

3. CAN 통신을 하면서 발생하였던 문제점

이 장에서는 **CAN** 통신 프로그램을 작성하면서 발생하였던 여러 가지 문제점과 해결 방법에 대해서 알아본다**.**

3. CAN 통신을 하면서 발생하였던 여러 가지 문제점

3-1 제품 특성에 대한 이해 부족

- **CAN** 작업을 진행하면서 가장 문제가 되었던 부분이기도 하지만 가장 해결 하기 어려웠던 부분이기도 하다.
- 제품 특성을 파악하기 위해서는 우선적으로 제품 관련 자료를 최대한 많이 보고 제품을 분석하는 방법 외에는 해결 방법이 없을 것이다.
- 특히 대성전기 **(VQ-CAR** 및 **HM-CAR** 최종 검사기**)** 의 **CAN** 적용 프로그램을 진행하면서 **K-LINE** 통신 방식에서 보았던 제품의 특성과 **CAN**으로만 작업했을 경우의 제품 특성은 상당히 많은 차이를 보였다.

- 제품 특성 파악을 위해서 최대한 많은 자료를 분석한 후 제품을 최대한 많이 **TEST**해서 그 특성을 파악하여 프로그램을 진행하는 방법으로 진행해야 한다.
- 물론 위 제품과 같이 특성 분석이 힘든 제품도 있겠지만 그렇지 않은 제품들에 대해서는 자료 분석 만이라도 필히 해 놓아야 한다.

3-2 자료의 부족

- 어떤 장비를 제작하더라 제품에 관한 자료 분석은 필수 일 것이다.
- 하지만 **CAN** 또는 **LIN**과 같은 통신 관련 자료는 상당히 구하기가 힘들다.
- **VQ / HM** 장비를 진행하면서도 **CAN** 관련 제품별 어떤 형식으로 통신이 이루어져야 하는지에 대한 자료가 없어 일일이 제품에서 데이터를 취득해서 분석하는 작업을 해야만 했다.

- ❖ 프로그램을 진행하는 담당자는 자료를 최대한 제작 의뢰 업체에 요청하고 그 자료를 구하기가 어려울 경우 제품을 최대한 빨리 구해서 하드웨어 담당자의 도움을 받아 제품을 분석하고 데이터를 취득 해야 할 것이다.
- ❖ 위 와 같은 작업을 하지 않고 프로그램을 진행할 경우 상당한 시행 착오를 거쳐야 할 것이다.

3-3. CAN 통신 CHIP 이해 부족

- ◆ **CAN** 통신 프로그램을 작성하면서 가장 많은 시간 동안 시행 착오를 하게 만든 문제점이 바로 **CAN** 통신의 이해 부족일 것이다.
- ◆ 프로그램 및 하드웨어 설계 담당자도 이 문제를 빨리 파악하지 못하여 시간을 낭비한 **CASE** 이다.
- ◆ 즉 **HIGH SPEED CAN CHIP** 과 **LOW SPEED CAN CHIP** 의 차이로 인한 것이다.
- ◆ 현재 자동차 관련 제품에는 대부분이 **LOW SPEED CAN** 통신 방식으로 구성되어 있으며 몇몇 제품에만 **HIGH SPEED CAN**을 사용한다.
- ◆ **HIGH SPEED CAN CHIP** 의 경우 고속의 통신 으로 많은 데이터를 짧은 시간 동안 통신할 수 있는 장점이 있는 반면에 **LOW SPEED CAN** 통신에 사용하였을 경우 데이터를 잃어 버리는 문제점을 않고 있다.
- ◆ 위 문제를 빨리 **CATCH** 했더라면 **3**개월 가까운 시간을 낭비 하지 않아도 되었을 것이다.

❖ 하드웨어 담당자 및 프로그램 담당자는 앞으로 **CAN** 통신 작업을 진행할 때 **HIGH SPEED CAN CHIP** 을 사용할 것인지 **LOW SPEED CAN CHIP**을 사용할 것인지를 정확하게 판단하여 작업을 진행해야만 이번과 같은 시행 착오를 하지 않을 것이다.

3-4. 앞으로의 과제

- ◆ 좀더 윤활하게 **MICOM** 과 **PC** 간 통신이 이루어 지도록 **MICOM** 및 **PC** 프로그램을 개선해야 하는 문제점이 남아 있다.
- ◆ **PC** 프로그램에서는 앞으로 **CAN** 통신을 이용한 제품 검사 장비가 많을 것으로 판단된다. 다른 프로그램에 쉽게 적용할 수 있도록 라이브러리 형식으로 구성하는 작업을 해야 할 것이다.
- ◆ **CAN** 통신 특성과 관계없이 회사에서 보유하고 있는 **CAN** 보드를 이용 해서 제품에서 생성되는**CAN** 정보를 쉽게 파악할 수 있는 **TEST** 용 **PROGRAM** 을 만드는 작업이다.