DIBD 프로토콜의 특수 목적 패킷

다빛채와 DIBD는 기본적으로 메시지 패킷만으로 통신하시지만, 잡음이 많은 산업현장이나 통신선로에서 사용되는 경우에는 통신 에러 검출/처리를 위한 특수 목적 패킷을 적용할 수 있습니다.
여기서는 특수 목적 패킷의 구조, 종류 및 사용 방법을 설명합니다.

1. 특수목적 패킷 구조

메시지 패킷 Length 의 하위 0bit~11bit(4096)는 데이터 길이이고, 상위 12bit~15bit 를 응답메시지의 반환여부 제어, CRC16 에러검출 방법, DLE 바이트 스토핑과 같은 특수한 목적 패킷으로 사용됩니다.

메시지 패킷

패킷		설명
DLE(10H)	STX(02H)	Start of a message
DST		Destination Station of the Message
High-LEN	**Low- LEN**	**Number of Bytes to be Transferred**
CMD		Message Command Code
DATA		Data
High-CRC16	Low-CRC16	Cyclic Redundancy Check(Option)
DLE(10H)	ETX(03H)	End of a message

명령어부(Command Part)의 메시지 패킷

명칭	설명
DLE STX	모든 메시지는 DLE(10H) STX(02H)로 시작합니다.
DST	Destination 은 목적지 주소를 의미합니다.
LEN	Bit0~Bit11 Bit0~Bit11 으로 4095 개까지의 Length (데이터 길이)를 표시합니다. Length는 메시지에 의해 전송된 Command와 Data의 크기이고, Command Type에 따라서 크기가 다양합니다. 상위 바이트에 이어서 하위 바이트를 전송합니다. *CRC16 에러검출방법을 특수목적 패킷을 사용하면 Length는 CRC16 값 2byte를 더해야 합니다. Bit12~Bit15 메시지의 패킷 상태를 특수목적으로 사용할 수 있습니다. 현장 상황에 따라서 필요한 경우에만 적용합니다. Bit12 – 사용안함 Bit13이 1이면 DIBD에서 응답메시지(Reply Message)를 보내지 않습니다. Bit14이 1이면 CRC16 에러검출방법을 사용합니다. Bit15이 1이면 DLE 바이트 스터핑(Byte Stuffing)을 사용합니다.
CMD	Command에서는 Message 유형과 기능을 정의합니다.

DATA	Data 는 Command Code에 대한 메시지의 내용을 의미합니다.
CRC16 (Option)	DST, LEN, CMD, DATA의 블록 합을 검사하고, 검출 값을 Low Byte, High Byte순서로 전송합니다. LEN의 특수목적 비트에서 Bit14이 1이면 CRC16 바이트를 메시지 패킷에 삽입하고, 0이면 CRC16 바이트를 메시지 패킷에서 삭제합니다.
DLE ETX	모든 메시지는 DLE(10H) ETX(03H)로 종료한다.

응답부(Reply Part)의 메시지 패킷

명칭	설명
DLE STX	모든 메시지는 Start symbol을 의미하는 DLE STX로 시작합니다.
DST	DIBD의 목적지 주소는 마스터 주소인 0을 반환합니다.
LEN	명령어부 메시지 패킷의 특수목적 비트가 1로 적용되어 수신되면 응답메시지도 동일한 특수목적이 적용됩니다.
CMD	Command에서는 Message 유형과 기능을 정의합니다.
DATA	Data 는 Command Code에 대한 메시지의 내용을 의미합니다.
CRC16	명령어부의 메시지 패킷과 동일합니다. (Option)
DLE ETX	모든 메시지는 DLE(10H) ETX(03H)로 종료합니다.

2. 응답메시지(Reply Message) 제어

일반적인 메시지 전송순서는 명령메시지를 전송하고 나서 응답메시지를 수신 받고, 다시 다음 명령메시지를 전송합니다. 시리얼 멀티 통신에서 여러 대의 컨트롤러에 명령메시지를 실시간으로 전송하고자 할 때 전송속도를 향상시키기 위해서 응답 메시지를 받지 않을 수 있습니다. 응답 메시지를 받지 않으면 현장에서 통신상태를 확인하는데 더 많은 주의가 필요하므로 권장하지 않습니다.
명령메시지의 LEN 의 특수목적 Bit13 이 1 이면, 응답메시지를 처리하지 않을 것이므로 반환하지 말라는 의미의 메시지입니다. 이 메시지를 받은 DIBD 는 수신 받은 메시지를 처리하고 나서 응답메시지를 전송하지 않습니다.

다빛채에서는 모든 명령 메시지에 대한 응답메시지를 받지 않도록 응답메시지 제어비트를 1 로 설정할 수 있습니다.
"davitche.ini" 파일을 열고, [USER]항목의 변수인 "FlagReplyPacket = 1"로 설정합니다.
다빛채 프로그램을 열고, 설정 창에서 "전원 켜기" 등의 전송 버튼을 누르면, 명령 메시지 패킷을 메인화면 하단의 로그 창에서 확인할 수 있습니다. 응답 메시지가 로그 창에 나타나지 않습니다.

다빛채(마스터)와 여러대의 DIBD(슬레이브)를 RS-485 멀티통신 방식으로 연결해서 실시간으로 전송하는 경우에 적용될 수 있습니다.
RS-485 통신 방식은 1 쌍의 통신라인으로 마스터에서 슬레이브로 메시지를 보내고 나서 다시 동일한 회선으로 응답을 받는 반이중 통신 방식입니다. 마스터가 첫 번째 슬레이브에 메시지를 보내고 나서 응답시간을 기다리지 않고 바로 두 번째 슬레이브에 메시지를 보내면 첫 번째 슬레이브의 응답데이터와

충돌이 발생되므로, 반드시 응답을 받고 다음 명령메시지를 전송해야 됩니다. 따라서, 전체 슬레이브에 데이터를 보내기 위해서 필요한 시간은 명령 메시지를 보내는 시간과 응답메시지를 받는 시간을 합한 값에 슬레이브 수를 곱한 값이 됩니다. 슬레이브가 많을수록 동시에 메시지를 처리하는데 걸리는 시간은 비례해서 늘어납니다. 실시간으로 데이터를 표시해야 하는 경우에 부담이 되므로, 슬레이브의 응답 메시지를 보내지 않도록 설정하면 메시지 처리시간은 상당히 줄어들 수 있으므로 응답메시지 제어비트를 1 로 설정합니다.

Command Message 의 패킷: (PC → DIBD)

DLE	STX	DST	LEN	CMD	DATA	DLE	ETX
10H	02H	00H	**20H** 03H	41H	01H 01H	10H	03H

- 목적지 주소 0 인 DIBD 의 전원제어: LEN 의 특수목적 Bit13 이 1 이므로, 전원켜기 명령 메시지를 전송하고 나서 응답메시지를 받지 않습니다.

3. CRC16 에러검출

데이터를 전송할 때 전송된 데이터에 오류가 있는지를 확인하기 위한 체크 값을 결정하는 방식을 말합니다.

데이터를 전송하기 전에 주어진 데이터의 값에 따라 CRC값을 계산하여 데이터에 붙여 전송하고, 데이터 전송이 끝난 후 받은 데이터의 값으로 다시 CRC 값을 계산하게 됩니다. 이어서 두 값을 비교하고, 이 두 값이 다르면 데이터 전송 과정에서 잡음 등에 의해 오류가 덧붙여 전송된 것 임을 알 수 있습니다.

명령메시지의 LEN의 특수목적 Bit14가 1이면, CRC16이 적용된 메시지입니다.

DIBD 는 수신 받는 메시지에서 DST(목적지주소), LEN(Command 와 Data 크기), CMD, DATA 의 CRC16 값을 계산하고, 메시지로부터 받은 CRC16 값과 비교해서 에러검출을 합니다.

DIBD 의 응답 메시지도 LEN 의 특수목적 Bit14 를 1 로 설정하고, CRC16 에러 검출방법을 적용해서 전송합니다.

다빛채에서는 모든 명령 메시지 패킷에 CRC16 에러 검출방법을 적용해서 전송할 수 있습니다.

"davitche.ini" 파일을 열고, [USER]항목의 변수인 "FlagCrcPacket = 1"로 설정합니다.

다빛채 프로그램을 열고, "전원 켜기" 등의 전송 버튼을 누르면, CRC16 이 적용된 메시지 패킷을 메인화면 하단의 로그 창에서 확인할 수 있습니다.

다빛채와 DIBD 에서 사용하는 실제 CRC16 소스코드는 "첨부 1. CRC16 소스코드"를 참조하세요.

Command Message 의 패킷: (PC → DIBD)

DLE	STX	DST	LEN	CMD	DATA	CRC16	DLE	ETX
10H	02H	00H	**40H** 04H	41H	01H	**A5H 91H**	10H	03H

- 목적지 주소 0 인 DIBD 의 전원제어: LEN 의 특수목적 Bit14 가 1 이므로, CRC16 이 적용된 전원켜기 명령(41H) 메시지를 전송합니다.

Reply Message 의 패킷: (DIBD → PC)

DLE	STX	DST	LEN	CMD	DATA	CRC16	DLE	ETX
10H	02H	00H	**40H** 04H	41H	00H	**64H 51H**	10H	03H

• LEN 의 특수목적 Bit14 가 1 이므로, CRC16 이 적용된 응답 메시지를 반환합니다.

4. DLE 바이트 스터핑(Byte Stuffing)

HDLC 프로토콜에서 프레임의 시작과 끝을 알리는 방법으로, ASCII 코드 중에 DLE STX 및 DLE ETX라는 특정 문자들을 사용하는데, 실제 데이터 몸체에서도 동일 문자가 나오는 것을 방지하기 위해, 송신측에서 우연히 나타나는 DLE 문자 바로 직전에 여분의 DLE를 삽입하게 되는데 이를 바이트 스터핑(Byte Stuffing)이라 합니다. – 정보통신기술 용어해설 참조

10H 02H 라고 전송해 줌으로써 데이터의 시작을 알려주고, 10H 03H 으로 데이터의 종료임을 알려 주는 방법입니다. 데이터에 10H 가 있다면 2 번 연속해서 10H 10H 로 전송합니다.

Header	Command + Data + CRC16	Tail
DLE STX DST LEN	데이터 중에 10H 가 있다면 2 번 연속해서 전송한다.	DLE ETX

수신부는 데이터에 10H가 수신되면 구분코드로 판단합니다. 다음 바이트도 10H 이면 데이터로서 구분코드 10H는 버리고 두 번째 10H만을 데이터로 저장합니다.

	DLE	STX	DST	LEN	CMD	DATA	DLE	ETX
수신데이터	10H	02H	00H	80H 00H	54H	31H 10H 10H 32H 33H	10H	03H
수신버퍼	10H	02H	00H	80H 00H	54H	31H 10H 32H 33H	10H	03H

DLE Byte Stuffing 참조 사이트 : http://forum.falinux.com/zbxe/?document_srl=406028

명령메시지의 LEN 의 특수목적 Bit15 가 1 이면, DLE 바이트 스토핑이 적용된 메시지입니다. DIBD 에서는 메시지를 처리하고, DLE 바이트 스토핑을 적용한 응답 메시지를 반환합니다. 이 때 데이터의 크기는 사용하지 않으므로 00H 로 고정입니다.

"davitche.ini" 파일을 열고, [USER]항목의 변수인 "FlagDlePacket = 1"로 설정합니다.

다빛채 프로그램을 열고, "DIBD 프로토콜 전송 창"에서 전송버튼을 누르면, DLE 바이트 스토핑이 적용된 메시지를 로그 창에서 확인할 수 있습니다.

다빛채에서는 "FlagDlePacket"를 1 로 설정하면 DIBD 연결, 전원 켜기/끄기, 시간동기, DIBD 시간, 긴급문구, 일반문구의 명령 메시지에 DLE Byte Stuffing 을 적용해서 전송합니다.

Command Message 의 패킷: (PC → DIBD)

DLE	STX	DST	LEN	CMD	DATA	DLE	ETX
10H	02H	00H	**80H** 00H	6AH	10H 10H 31H 10H 10H	10H	03H

• 목적지 주소 0 인 DIBD 의 Echo 모드: LEN 의 특수목적 Bit15 가 1 이므로, DLE 스토핑이 적용된 Echo 모드 명령 메시지를 전송합니다. DIBD 의 수신버퍼에서는 구분문자 10H 를 제거하고 데이터 "10H 31H 10H"를 추출합니다.

Reply Message 의 패킷: (DIBD → PC)

DLE	STX	DST	LEN	CMD	DATA	DLE	ETX
10H	02H	00H	**80H** 00H	6AH	10H 10H 31H 10H 10H	10H	03H

- LEN 의 특수목적 Bit15 가 1 이므로, DLE 스토핑이 적용된 응답 메시지를 반환합니다. 응답 데이터를 전송할 때는 다시 구분문자 10H 를 삽입해서 전송합니다.

5. 특수목적 패킷 예제

명령 메시지 중에서 [Echo 모드(6AH)]는 수신 받은 명령메시지를 응답메시지로 되돌려 송신하는 메아리(Echo) 기능을 수행합니다.
시리얼 통신 테스트 프로그램으로 아래의 명령메시지를 전송하고 나서, 응답 메시지를 확인합니다.
특수목적 패킷(LEN Bit13~15)의 설정 값에 따라서 메시지 패킷의 구조가 변화되는 것을 볼 수 있습니다.

	통신방향	DLE	STX	DST	LEN	CMD	DATA	CRC16	DLE	ETX
1	응답메시지: 0, CRC16: 0, DLE Stuffing: 0									
	PC→	10H	02H	00H	00H 06H	6AH	31H 32H 33H 34H 35H		10H	03H
	← DIBD	10H	02H	00H	00H 06H	6AH	31H 32H 33H 34H 35H		10H	03H
2	응답메시지: 0, CRC16: 0, DLE Stuffing: 0									
	PC→	10H	02H	00H	00H 07H	6AH	10H 10H 31H 32H 10H 10H		10H	03H
	← DIBD	10H	02H	00H	00H 07H	6AH	10H 10H 31H 32H 10H 10H		10H	03H
3	응답메시지: 1, CRC16: 0, DLE Stuffing: 0									
	PC→	10H	02H	00H	20H 07H	6AH	10H 10H 31H 32H 10H 10H		10H	03H
	← DIBD	응답없음								
4	응답메시지: 0, CRC16: 1, DLE Stuffing: 0									
	PC→	10H	02H	00H	40H 09H	6AH	10H 10H 31H 32H 10H 10H	5BH C3H	10H	03H
	← DIBD	10H	02H	00H	40H 09H	6AH	10H 10H 31H 32H 10H 10H	5BH C3H	10H	03H
5	응답메시지: 0, CRC16: 0, DLE Stuffing: 1									
	PC→	10H	02H	00H	80H 00H	6AH	10H 10H 31H 32H 10H 10H		10H	03H
	← DIBD	10H	02H	00H	80H 00H	6AH	10H 10H 31H 32H 10H 10H		10H	03H
6	응답메시지: 0, CRC16: 1, DLE Stuffing: 1									
	PC→	10H	02H	00H	C0H 00H	6AH	10H 10H 31H 32H 10H 10H	FAH 6FH	10H	03H
	← DIBD	10H	02H	00H	C0H 00H	6AH	10H 10H 31H 32H 10H 10H	FAH 6FH	10H	03H
7	응답메시지: 1 CRC16: 1, DLE Stuffing: 1									
	PC→	10H	02H	00H	E0H 00H	6AH	10H 10H 31H 32H 10H 10H	63H AEH	10H	03H
	← DIBD	응답없음								

첨부 1. CRC16 소스코드

CRC16은 송수신에 대한 오류 검색을 정확하게 판단할 수 있는 방법을 제공하지만 같은 CRC16을 사용한다고 해도 소스마다 계산하는 방법이 다르거나, CRC 테이블이 틀린 경우가 있으므로 아래의 DIBD에서 사용하는 CRC16 함수와 테이블을 사용하시기 바랍니다.

u08은 unsigned char로 8bit이고, u16은 unsigned short int로 16bit 입니다.

```
u16   fnGetCrc16(u08   *nData,   u16   wLength)
{
        static const   u16   wCRCTable[] = {
                0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
                0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
                0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
                0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
                0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
                0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
                0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
                0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
                0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
                0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
                0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
                0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
                0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
                0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
                0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
                0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
                0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
                0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
                0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
                0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
                0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
                0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
                0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
                0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
                0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
                0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
                0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
                0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
                0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
                0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
                0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
                0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040 };
        u08 nTemp;
        u16 wCRCWord = 0;           //0: CRC16,   0xFFFF: CRC16(Modbus)

                while (wLength--)  {
                        nTemp = (u08)(*nData++ ^ wCRCWord);
                        wCRCWord >>= 8;
                        wCRCWord  ^= wCRCTable[nTemp];
                }
                return wCRCWord;
}
```