Ezi-IO® RS-485

Input/Output Module

사용설명서

통신 기능편

(**Rev.03**)

FASTECH

목 차

1 . 통신 프로토콜

1 - 1 . 통신 기능

Ezi-IO RS-485 제품 군은 RS-485 (2 선식)에서의 데이지체인 링크에 의해 최대 16 개까지 제어가 가능합니다.

⚠ 주의	**Windows 가 대기 모드 및 절전 모드로 들어갈 경우 기본적으로 Serial 통신이 두절됨을 주의 하십시오. 이 모드에서는 복구 후에 Serial 통신을 재연결 해야 합니다. 이 사항은 제공되는 라이브러리에도 동일하게 적용됩니다.**

1 - 1 - 1 . 통신 사양

통신 규격	RS-485
통신 방식	비동기식
	반이중 통신
전송속도 [bps]	9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600
데이터 형식	8 비트 ASCII 코드, HEX
Parity bit	No
Stop bit	1bit
CRC 검사	있음
최대 배선 길이 (컨버터<->끝단 드라이브)	30 m 이내
보드간 최소 배선 길이	60 cm 이상
최대 연결 보드 수	16 (No. 0~F)

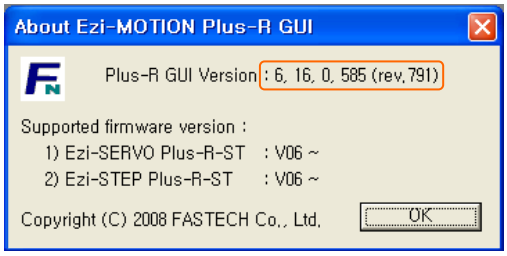
1 - 1 - 2 . RS-485 통신 프로토콜 (Ver6)

사용자 프로그램(GUI)의 Version 에는 다음의 2 가지가 지원되며, 본 매뉴얼은 버전 6 에 해당됩니다.

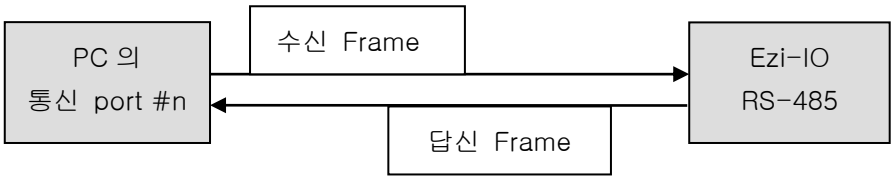
종류	펌웨어 버전	호환	사용자(GUI) 버전
1	6 번대 (V06.0x.0xx.xx)	<->	6 번대 (6.xx.x.xxx)
2	8 번대 (V08.xx.0xx.xx)	<->	8 번대 (8.xx.x.xxx)

현재 사용중인 버젼은 다음과 같이 확인할 수 있습니다.

사용자 프로그램(GUI)과 통신 연결 후 'Help' 메뉴 중 'About Plus-R GUI...'메뉴를 클릭하면 뜨는 오른쪽 창으로 확인



1) 통신 FRAME 의 개요



2) FRAME 의 기본 구조

Header	*Frame Data*	Tail
0xAA 0xCC	4~252 bytes	0xAA 0xEE

① 0xAA : 구분 표시용 byte

② 0xAA 0xCC : Frame 의 시작(Header)임을 표시합니다.

③ 0xAA 0xEE : Frame 의 종료(Tail)임을 표시합니다.

④ Frame Data 항의 데이터 중 실제 데이터가 '0xAA'인 경우에는 바로 뒤에 '0xAA'를 추가하여 줍니다. (byte stuffing)

⑤ '0xAA'다음의 데이터가 '0xAA', '0xCC' 또는 '0xEE'이 아니면 에러가 발생한 상황입니다.

1 - 1 - 3 . 수신 Frame Data 구조

수신 ***Frame Data*** 항의 세부 구성은 다음과 같습니다.

Slave ID	Frame type	Data	CRC	
1 byte	1 byte	0 ~ 248 bytes.	2 bytes	
			Low byte	High byte

① Slave ID : PC 의 통신 port 에 연결된 드라이브 모듈의 번호(0~15)입니다.

② Frame type : 해당 Frame 의 명령어 종류를 지정합니다.
그 종류는 아래의 「Frame type 과 Data 구성」절을 참조하십시오.

③ Data : 이 항의 데이터 구조 및 길이 등은 Frame type 에 따라 정해집니다.
그 자세한 구조는 아래의 「Frame type 과 Data 구성」절을 참조하십시오.

④ CRC : 통신중 에러 상태를 확인하기 위하여 CRC(Cyclic Redundancy Check) 규격중 다항식(polynomial) 인자값으로 CRC16(2 byte)의 '0xA001'을 사용하거나 CRC-16-IBM 의 생성 다항식 'X16+X15+X2+1'을 사용합니다.
CRC 계산은 CRC 항 이전의 모든 항(Slave ID, Frame type, Data)에 대해 수행합니다.

1 - 1 - 4 . CRC 계산 예제

아래의 프로그램 소스 내용은 제품과 함께 제공된 파일 (파일명 : CRC_Checksum.c)에 포함되어 있습니다.

1) CRC16 의 '0xA001'사용 방법

```
const unsigned short TABLE_CRCVALUE[] =
{
  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
};

unsigned short CalcCRC(unsigned char* pDataBuffer, unsigned long usDataLen)
{
  unsigned char nTemp;
  unsigned short wCRCWord = 0xFFFF;
```

```
    while (usDataLen--)
    {
            nTemp = wCRCWord ^ *(pDataBuffer++);
            wCRCWord >>= 8;
            wCRCWord ^= TABLE_CRCVALUE[nTemp];
    }
    return wCRCWord;
  }
```

2) CRC-16-IBM 의 'X16+X15+X2+1'사용 방법

```
unsigned short CalcCRCbyAlgorithm(unsigned char* pDataBuffer, unsigned long usDataLen)
{
    const unsigned short POLYNOMIAL = 0xA001;
    unsigned short wCrc;
    int iByte, iBit;

    /* Initialize CRC */
    wCrc = 0xffff;

    for (iByte = 0; iByte < usDataLen; iByte++)
    {
        /* Exclusive-OR the byte with the CRC */
        wCrc ^= *(pDataBuffer + iByte);

        /* Loop through all 8 data bits */

        for (iBit = 0; iBit <= 7; iBit++)
        {
            /* If the LSB is 1, shift the CRC and XOR the polynomial mask with the CRC */

          // Note - the bit test is performed before the rotation, so can't move the << here
            if (wCrc & 0x0001)
            {
                    wCrc >>= 1;
                    wCrc ^= POLYNOMIAL;
            }
            else
            {
                    // Just rotate it
                    wCrc >>= 1;
            }
        }
    }
    return wCrc;
    }
```

1 - 1 - 5 . 답신 Frame Data 구조와 통신 에러 (Ver6)

송신측 명령에 대한 답신측의 Frame 기본 구조는 동일합니다. 다만 아래의 ***Frame Data*** 항에서의 차이점은 '통신 상태'가 추가되어 다음과 같습니다.

Slave ID	Frame type	Data		CRC	
1 byte	1 byte	1 byte	0 ~ 247bytes	2 bytes	
		통신 상태	답신 데이터	Low byte	High byte

① Slave ID : 수신 Frame 과 동일.
(수신시의 데이터와 일치하지 않으면 에러 상태로 인식합니다)

② Frame type : 수신 Frame 과 동일.
(수신시의 데이터와 일치하지 않으면 에러 상태로 인식합니다.)

③ Data : 단순 실행 명령등의 송신에는 이 항목의 데이터는 없을 수 있습니다.
그러나 답신에는 통신 상태(에러/정상 여부)를 나타내는 1 byte 의 데이터가 포함됩니다.

통신 상태를 나타내는 byte 의 값에 대한 내용은 아래의 표와 같습니다.

Hexa 값	Decimal 값	내 용
0x00	0	통신이 정상 상태입니다.
0x80	128	Frame type 에러 : 수신한 Frame type 명령을 인식할 수 없습니다.
0x81	129	데이터 에러, ROM 데이터 읽기, 쓰기 에러 : 수신한 데이터의 값이 정해진 범위외의 데이터입니다.
0x82	130	수신 Frame 에러 : 수신된 Frame 이 규격에 맞지 않는 데이터입니다.
0xAA	170	CRC 에러 : 수신된 Frame 의 data 가 주변 노이즈등의 영향으로 CRC 에러가 발생했습니다. 이 경우에는 송신측 DLL 라이브러리에서 자동으로 통신을 1 회 더 시도합니다.

⚠ 주의	**1) 송신 Frame 의 'Header'와 'Slave ID'값이 정상이 아니면, 드라이브측에서 답신을 하지 않습니다.** **2) 통신 상태 '130'번의 에러가 발생한 경우에는 답신 데이터의 크기는 0 byte 입니다.**

1 - 2 . Frame 의 구성

1 - 2 - 1 . Frame type 별 송수신 내용

(1) 다음표는 Frame type 값에 따른 Data 항의 내용과 구성에 대한 설명입니다.

<table>
<tr><th>Frame type</th><th>라이브러리 명</th><th>내용</th></tr>
<tr><td>0x01
(1)</td><td>FAS_
GetSlaveInfo</td><td>
연결된 slave 의 종류와 프로그램 버전 정보를 요청.

송신 : 0 byte

수신 : 1~248 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td><td>0~246 bytes</td></tr>
<tr><td>통신 상태</td><td>Slave 종류</td><td>ACII string with NULL byte
(strlen() + 1 byte)</td></tr>
</table>
◆ Slave 종류
- 150 : Ezi-IO-RS-I16 시리즈
- 155 : Ezi-IO-RS-I8O8 시리즈
- 160 : Ezi-IO-RS-O16
</td></tr>
<tr><td>0xC0
(192)</td><td>FAS_GetInput</td><td>
Input / Latch 상태를 읽어 들임.

송신 : 0 byte

수신 : 9 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>통신 상태</td><td>Input 상태</td><td>Latch 상태</td></tr>
</table>
수신 DATA 구조
MSB LSB
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>47~40</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Input15</td><td>Input14</td><td>Input13</td><td>Input12</td><td>Input11</td><td>Input10</td><td>Input9</td><td>Input8</td><td>15~8</td></tr>
<tr><td>Input7</td><td>input6</td><td>Input5</td><td>Input4</td><td>Input3</td><td>Input2</td><td>Input1</td><td>Input0</td><td>7~0</td></tr>
</table>
0: Off
1: On

*Ezi-IO-RS-I16 시리즈 : Input 0~15, Latch 0~15 상태를 읽어 들임.
*Ezi-IO-RS-I8O8 시리즈 : Input 0~7, Latch 0~7 상태를 읽어 들임.
</td></tr>
</table>

<table>
<tr>
<td>0xC1
(193)</td>
<td>FAS_ClearLatch</td>
<td>
해당 bit 의 Latch 를 Clear(Reset)함.

송신 : 4 bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>

송신 Data 구조
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>15~8</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>7~0</td></tr>
</table>

0 : 현재 상태 유지
1 : Latch Clear(Reset)

*Ezi-IO-RS-I16 시리즈 : Latch 0~15
*Ezi-IO-RS-I8O8 시리즈 : Latch 0~7

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xC2
(194)</td>
<td>FAS_GetLatchCount</td>
<td>
Latch Count 를 읽어 들임.

송신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Data</td></tr>
</table>
Data : Latch bit 번호.
*Ezi-IO-RS-I16 시리즈 : 0~15
*Ezi-IO-RS-I8O8 시리즈 : 0~7

수신 : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>통신 상태</td><td>Data</td></tr>
</table>
Data : 0~2^{32}
</td>
</tr>
</table>

<table>
<tr>
<td>0xC3
(195)</td>
<td>FAS_GetLatchCount
All</td>
<td>
Latch Count 전체를 읽어 들임.

송신 : 0 byte

수신 : 65 bytes
<table>
<tr><td>1 byte</td><td>64 bytes</td></tr>
<tr><td>통신 상태</td><td>Data</td></tr>
</table>
수신 Data 구조
<table>
<tr><td colspan="2">MSB</td><td colspan="4"></td><td colspan="2">LSB</td></tr>
<tr><td>COUNT15</td><td>COUNT14</td><td>COUNT13</td><td>・・・</td><td>・・・</td><td>COUNT2</td><td>COUNT1</td><td>COUNT0</td></tr>
<tr><td>4 bytes</td><td>4 bytes</td><td>4 bytes</td><td></td><td></td><td>4 bytes</td><td>4 bytes</td><td>4 bytes</td></tr>
</table>
*Ezi-IO-RS-I16 시리즈 : COUNT 0~15
*Ezi-IO-RS-I8O8 시리즈 : COUNT 0~7
</td>
</tr>
<tr>
<td>0xC4
(196)</td>
<td>FAS_ClearLatch
Count</td>
<td>
해당 bit 의 Latch Count 를 Reset 함.

송신 : 4 bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
송신 Data 구조
<table>
<tr><td colspan="2">MSB</td><td colspan="6"></td><td>LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>15~8</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>7~0</td></tr>
</table>
*Ezi-IO-RS-I16 시리즈 : Latch 0~15
*Ezi-IO-RS-I8O8 시리즈 : Latch 0~7

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>0xC5
(197)</td>
<td>FAS_GetOutput</td>
<td>
Output / Trigger(Run/Stop) 상태를 읽어 들임.

송신 : 0 byte

수신 : 9 bytes
<table>
<tr><td>1 byte</td><td>8 bytes</td></tr>
<tr><td>통신 상태</td><td>Data</td></tr>
</table>

수신 Data 구조
<table>
<tr><td>MSB</td><td></td><td></td><td></td><td></td><td></td><td></td><td>LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Run/Stop15</td><td>Run/Stop14</td><td>Run/Stop13</td><td>Run/Stop12</td><td>Run/Stop11</td><td>Run/Stop10</td><td>Run/Stop9</td><td>Run/Stop8</td><td>47~40</td></tr>
<tr><td>Run/Stop7</td><td>Run/Stop6</td><td>Run/Stop5</td><td>Run/Stop4</td><td>Run/Stop3</td><td>Run/Stop2</td><td>Run/Stop1</td><td>Run/Stop0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Output15</td><td>Output14</td><td>Output13</td><td>Output12</td><td>Output11</td><td>Output10</td><td>Output9</td><td>Output8</td><td>15~8</td></tr>
<tr><td>Output7</td><td>Output6</td><td>Output5</td><td>Output4</td><td>Output3</td><td>Output2</td><td>Output1</td><td>Output0</td><td>7~0</td></tr>
</table>

0 : OFF (Trigger Stop)
1 : ON (Trigger Run)

*Ezi-IO-RS-O16 : Output 0~15, Trigger(Run/Stop) 0~15 상태를 읽어 들임.
*Ezi-IO-RS-I8O8 시리즈 : Output 8~15, Trigger(Run/Stop) 8~15 상태를 읽어 들임.
</td>
</tr>
</table>

<table>
<tr>
<td>0xC6
(198)</td>
<td>FAS_SetOutput</td>
<td>
Output 을 설정 함.(Set : On, Reset : Off)

송신 : 8 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>Set Output</td><td>Reset Output</td></tr>
</table>

송신 Data 구조
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Reset Output15</td><td>Reset Output14</td><td>Reset Output13</td><td>Reset Output12</td><td>Reset Output11</td><td>Reset Output10</td><td>Reset Output9</td><td>Reset Output8</td><td>47~40</td></tr>
<tr><td>Reset Output7</td><td>Reset Output6</td><td>Reset Output5</td><td>Reset Output4</td><td>Reset Output3</td><td>Reset Output2</td><td>Reset Output1</td><td>Reset Output0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Set Output15</td><td>Set Output14</td><td>Set Output13</td><td>Set Output12</td><td>Set Output11</td><td>Set Output10</td><td>Set Output9</td><td>Set Output8</td><td>15~8</td></tr>
<tr><td>Set Output7</td><td>Set Output6</td><td>Set Output5</td><td>Set Output4</td><td>Set Output3</td><td>Set Output2</td><td>Set Output1</td><td>Set Output0</td><td>7~0</td></tr>
</table>

1 : Set Output or Reset Output 실행
(Set Output 과 Reset Output 의 같은 bit 가 1 일 경우 Reset 으로 실행)
0 : Set Output or Reset Output 실행하지 않음

*Ezi-IO-RS-O16 : Set Output 0~15, Reset Output 0~15 설정가능
*Ezi-IO-RS-I8O8 시리즈 : Set Output 8~15, Reset Output 8~15 설정가능

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>0xC7
(199)</td>
<td>FAS_SetTrigger</td>
<td>
Trigger Out 설정.

송신 : 13 bytes
<table>
<tr><td>13 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
송신 Data 구조
<table>
<tr><td colspan="3">MSB</td><td colspan="3">LSB</td></tr>
<tr><td>Count</td><td>Blank</td><td>On time</td><td>Blank</td><td>Period</td><td>Output No.</td></tr>
<tr><td>4 bytes</td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>1 byte</td></tr>
</table>
Output No. : 트리거 설정 출력 번호
*Ezi-IO-RS-O16 : 0~15
*Ezi-IO-RS-I8O8 시리즈 : 8~15
Period : 트리거 출력 주기, 단위 [ms],
On time 보다 반드시 큰 값을 입력(2~65,535)
On time : 트리거 출력 On time, 단위[ms],
Period 보다 반드시 작은 값을 입력 (1~65,534)
Count : 트리거 출력 반복 횟수 (1~4,294,967,295)

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xC8
(200)</td>
<td>FAS_SetRunStop</td>
<td>
해당 bit 의 Trigger 출력을 Run or Stop 함.
Run 중 Stop bit 가 On 되면 count 만큼 출력되지 않아도 Stop 함.

송신 : 13 bytes
<table>
<tr><td>13 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
송신 Data 구조
<table>
<tr><td colspan="8">MSB</td><td>LSB</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Stop15</td><td>Stop14</td><td>Stop13</td><td>Stop12</td><td>Stop11</td><td>Stop10</td><td>Stop9</td><td>Stop8</td><td>47~40</td></tr>
<tr><td>Stop7</td><td>Stop6</td><td>Stop5</td><td>Stop4</td><td>Stop3</td><td>Stop2</td><td>Stop1</td><td>Stop0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Run15</td><td>Run14</td><td>Run13</td><td>Run12</td><td>Run11</td><td>Run10</td><td>Run9</td><td>Run8</td><td>15~8</td></tr>
<tr><td>Run7</td><td>Run6</td><td>Run5</td><td>Run4</td><td>Run3</td><td>Run2</td><td>Run1</td><td>Run0</td><td>7~0</td></tr>
</table>
*Ezi-IO-RS-O16 : Run 0~15, Stop 0~15
*Ezi-IO-RS-I8O8 시리즈 : Run 8~15, Stop 8~15
수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>0xC9
(201)</td>
<td>FAS_GetTrigger</td>
<td>현재의 Trigger 출력 횟수를 읽어 들임.

송신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Data</td></tr>
</table>

Data : Trigger 출력 bit 번호
*Ezi-IO-RS-O16 : 출력 bit 번호 0~15
*Ezi-IO-RS-I8O8 시리즈 : 출력 bit 번호 8~15

수신 : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>통신 상태</td><td>Data</td></tr>
</table>
Data : 0~2^{32}</td>
</tr>
<tr>
<td>0xCA
(202)</td>
<td>FAS_GetIOLevel</td>
<td>Input or Output Active Level 을 읽어 들임.

송신 : 0 byte

수신 : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>통신 상태</td><td>Data</td></tr>
</table>

수신 Data 구조
<table>
<tr><td colspan="8">MSB</td><td>LSB</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>LEVEL15</td><td>LEVEL14</td><td>LEVEL13</td><td>LEVEL12</td><td>LEVEL11</td><td>LEVEL10</td><td>LEVEL9</td><td>LEVEL8</td><td>15~8</td></tr>
<tr><td>LEVEL7</td><td>LEVEL6</td><td>LEVEL5</td><td>LEVEL4</td><td>LEVEL3</td><td>LEVEL2</td><td>LEVEL1</td><td>LEVEL0</td><td>7~0</td></tr>
</table>

0 : Low Active Level (Latch : Falling edge)
1 : High Active Level (Latch : Rising edge)

*Ezi-IO-RS-O16 : Output Active Level 0~15 을 읽어 들임.
*Ezi-IO-RS-I16 시리즈 : Input Active Level 0~15 을 읽어 들임.
*Ezi-IO-RS-I8O8 시리즈 : Input Active Level 0~7, Output Active Level 8~15 을 읽어 들임.</td>
</tr>
</table>

<table>
<tr>
<td>0xCB
(203)</td>
<td>FAS_SetIOLevel</td>
<td>
Input or Output Active Level 을 설정.

송신 : 4 bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>

송신 Data 구조
MSB LSB
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>LEVEL15</td><td>LEVEL14</td><td>LEVEL13</td><td>LEVEL12</td><td>LEVEL11</td><td>LEVEL10</td><td>LEVEL9</td><td>LEVEL8</td><td>15~8</td></tr>
<tr><td>LEVEL7</td><td>LEVEL6</td><td>LEVEL5</td><td>LEVEL4</td><td>LEVEL3</td><td>LEVEL2</td><td>LEVEL1</td><td>LEVEL0</td><td>7~0</td></tr>
</table>
0 : Low Active Level (Latch : Falling edge)
1 : High Active Level (Latch : Rising edge)

*Ezi-IO-RS-O16 : Output Active Level 0~15 을 설정.
*Ezi-IO-RS-I16 시리즈 : Input Active Level 0~15 을 설정.
*Ezi-IO-RS-I8O8 시리즈 : Input Active Level 0~7, Output Active Level 8~15 을 설정.

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xCC
(204)</td>
<td>FAS_
LoadIOLevel</td>
<td>
ROM 메모리 Active Level 을 읽어 들임.
Board 의 RAM data 를 읽어 들인 data 로 변경함.

송신 : 0 byte

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xCD
(205)</td>
<td>FAS_
SaveIOLevel</td>
<td>
현재 설정된 Active Level 값을 ROM 메모리에 저장함. 이는 전원을 OFF 해도 해당 값들이 저장되도록 합니다.

송신 : 0 byte

수신 : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>통신 상태</td></tr>
</table>
</td>
</tr>
</table>

1 - 3 . 프로그램의 종류

Ezi-IO RS-485 를 사용하기 위한 프로그램 방법은 두 가지가 있습니다.

첫 번째는 일반적으로 사용하는 방법으로서 PC 의 window system 에서 Visual C++ 언어를 사용하는 방법입니다. 이 때에는 제품과 함께 제공된 라이브러리 (「2. PC 프로그램용 라이브러리」를 참조)를 사용합니다.

두 번째는 라이브러리 함수를 사용하지 않고 사용자가 직접 명령어 (command character)를 전송 하는 방법입니다. Protocol Test 프로그램과 같이 low level 의 protocol program 을 사용자가 직접 작성해야 하며, 주로 상위 제어기로 PLC 등을 사용할 경우에 적용됩니다.
구체적인 프로그램 방법은 제품과 함께 제공된 「3. PLC 프로그램용 프로토콜」을 참조하시고, 'ProtocolTest_PlusR.exe'라는 사용자 GUI 프로그램으로 시험 Test 를 하실 수 있습니다.

2 . PC 프로그램용 라이브러리 (Ver6)

2 - 1 . 라이브러리의 구성

이 라이브러리를 사용하기 위해서는 C++ header file(*.h)와 library file(*.lib or *.dll) 이 필요합니다. 이 파일들은 "₩₩FASTECH₩₩EziMOTION PlusR₩₩include₩₩에 있으며 개발용 source file 에 다음 내용을 포함 시키십시오.

#include "₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩FAS_EziMotionPlusR.h"
#include "₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩ReturnCodes_Define.h"
#include "₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩MOTION_DEFINE.h"
#include "₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩COMM_Define.h"

또한 라이브러리 파일은 다음과 같습니다.
"₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩EziMotionPlusR.lib"
"₩₩FASTECH₩₩ EziMOTION PlusR₩₩include₩₩EziMotionPlusR.dll"

이 라이브러리를 사용한 sample program source 등이
"₩₩FASTECH₩₩EziMOTION PlusR₩₩Examples₩₩"폴더에 포함되어 있습니다.

(1) 다음의 표는 각 라이브러리 함수 사용시 리턴되는 값들에 대한 설명입니다.
이 리턴 값들은 라이브러리(DLL) 함수에서만 확인할 수 있고 프로토콜을 사용한 프로그램 방식에서는 지원되지 않습니다.

구분	명칭	리턴값	내용
정상	FMM_OK	0	함수가 정상적으로 명령을 수행하였습니다
입력 에러	FMM_NOT_OPEN	1	잘못된 Port 번호를 입력하였습니다.
	FMM_INVALID_PORT_NUM	2	연결되지 않은 Port 번호입니다.
	FMM_INVALID_SLAVE_NUM	3	잘못된 Slave 번호를 입력하였습니다.
연결 에러	FMC_DISCONNECTED	5	해당 Board 가 연결 해제 되었습니다
	FMC_TIMEOUT_ERROR	6	정해진 시간(100msec)동안 응답이 없습니다.
	FMC_CRCFAILED_ERROR	7	통신중 Checksum 에러가 발생하였습니다.
	FMC_RECVPACKET_ERROR	8	받은 패킷에서 프로토콜 레벨의 에러가 발생하였습니다.

(2) 다음의 표는 모든 라이브러리에 공통적으로 포함되는 리턴값으로서 드라이브에서 판단한 결과(통신 상태, 운전 상태)를 확인할 수 있는 기능입니다. 라이브러리(DLL)를 사용하는 경우와 프로토콜을 사용한 프로그래밍 모두에 지원 됩니다.

구분	명칭	리턴값	내용
정상	FMP_OK	0	통신이 정상적으로 수행되었습니다.
입력 에러	FMP_FRAMETYPEERROR	128	모듈이 인식하지 못하는 명령어 입니다.
	FMP_DATAERROR	129	입력한 데이터가 범위를 벗어났습니다.
연결 에러	FMP_PACKETERROR	130	모듈이 받은 패킷에서 프로토콜 레벨의 에러가 발생하였습니다.
	FMP_PACKETCRCERROR	170	모듈이 받은 패킷의 CRC 계산값이 일치하지 않습니다.

2 - 2 . 모듈 연결함수

함수명	내용
FAS_Connect	모듈과 통신 연결을 시도합니다. : 성공적으로 접속했다면 TRUE 를, 접속에 실패를 했다면 FALSE 를 리턴합니다.
FAS_Close	모듈과 통신 해지를 시도합니다.
FAS_GetSlaveInfo	모듈의 종류와 프로그램 버젼을 읽어들입니다. : 드라이브 종류와 버젼 정보를 리턴합니다.
FAS_IsSlaveExist	해당 드라이브의 존재 여부를 확인합니다. : 존재하면 TRUE 를, 접속에 실패를 했다면 FALSE 를 리턴합니다.
FAS_EnableLog	통신 오류 관련 Log 의 출력을 제어합니다 : 존재하면 TRUE 를, 접속에 실패를 했다면 FALSE 를 리턴합니다.
FAS_SetLogPath	출력될 Log 가 저장될 경로를 설정합니다 : 존재하면 TRUE 를, 접속에 실패를 했다면 FALSE 를 리턴합니다.

FAS_Connect

FAS_Connect 함수는 Ezi-IO RS-485 를 접속하는 함수입니다.

Syntax

```
BOOL FAS_Connect(
    BYTE nPortNo,
    DWORD dwBaud
);
```

Parameters

nPortNo

접속하려는 Serial Port 번호를 선택합니다.

dwBaud

Serial Port 의 Baudrate 를 입력합니다.

Return Value

성공적으로 접속했다면 TRUE 를, 접속에 실패를 했다면 FALSE 를 리턴합니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInit()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        DWORD dwBaudrate = 115200;          // Baudrate 값. (설정에 의해 변경될 수 있습니다)
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        char lpBuff[256];
        int nBuffSize = 256;
        BYTE nType;
        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, dwBaudrate) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        if (FAS_IsSlaveExist(nPortNo, iSlaveNo) == FALSE)
```

```
        {
                // 해당 Slave 번호는 존재하지 않습니다.
                // Ezi-IO GATE PlusR 의 Slave 번호를 확인하십시오.
                return;
        }

        nRtn = FAS_GetSlaveInfo(nPortNo, iSlaveNo, &nType, lpBuff, nBuffSize);
        if (nRtn != FMM_OK)
        {
                // 명령이 정상적으로 수행되지 않았습니다.
                // ReturnCodes_Define.h 를 참조하십시오.
        }

        printf("Port : %d (Slave %d) ₩n", nPortNo, iSlaveNo);
        printf("₩tType : %d ₩n", nType);
        printf("₩tVersion : %d ₩n", lpBuff);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

FAS_Close

FAS_Close

사용하던 Serial Port 를 연결 해제 합니다.

Syntax

```
void FAS_Close(
    BYTE nPortNo
);
```

Parameters

nPortNo

연결을 해제할 Port 번호.

Remarks

Example

FAS_Connect 라이브러리 참조.

See Also

FAS_Connect

FAS_GetSlaveInfo

해당 Board 의 버젼 정보 문자열을 받아옵니다.

Syntax

```
int FAS_GetSlaveInfo(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE pType,
    LPSTR lpBuff,
    int nBuffSize
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

pType

해당 Board 의 Type 번호.

lpBuff

Version 정보 문자열을 입력받을 Buffer Pointer.

nBuffSize

lpBuff 의 메모리 할당 크기 값.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.
FMM_NOT_OPEN : Board 를 연결하기 전 입니다.
FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.
FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

FAS_Connect 라이브러리 참조.

See Also

FAS_IsSlaveExist

현재 모듈이 연결 상태인지를 확인합니다.

Syntax

```
BOOL FAS_IsSlaveExist(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

Return Value

TRUE : 연결 상태.

FALSE : 해지 상태.

Remarks

이 함수는 라이브러리에서만 제공되며 프로토콜 프로그램 방식에는 지원되지 않습니다.

Example

FAS_Connect 라이브러리 참조.

See Also

FAS_Connect

FAS_EnableLog

통신 오류 관련 Log 의 출력을 제어합니다.

Syntax

```
void FAS_EnableLog(BOOL bEnable);
```

Parameters

bEnable

Log 를 출력할지 설정합니다.

Remarks

현재 Process 에서 Ezi-MOTION Plus-R 함수를 사용하는 중 발생하는 Log 의 출력을 제어합니다. 이 설정은 다른 Process 혹은 다른 프로그램의 Log 출력에 영향을 끼치지 않습니다.
Log 는 FAS_Connect 부터 발생하며, FAS_Close 를 하여 현재 연결된 모든 Port 를 닫으면 Log 의 출력은 종료됩니다. Log 출력의 기본 설정 값은 TRUE 입니다.

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcDisableLog()
{
	BYTE nPortNo = 1;  // COMM Port 번호

	FAS_EnableLog(FALSE);

	// 이 후 함수들의 Log 는 출력되지 않습니다.

	// 연결을 시도합니다.
	if (FAS_Connect(nPortNo, 115200) == FALSE)
	{
		// 연결이 실패하였습니다.
		// 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
		return;
	}

	// 연결을 종료합니다.
	FAS_Close(nPortNo);
}
```

See Also

FAS_SetLogPath

FAS_SetLogPath

출력될 Log 가 저장될 경로를 설정합니다.

Syntax

BOOL FAS_SetLogPath(LPCTSTR lpPath);

Parameters

lpPath

Log 가 저장될 절대 경로 문자열.

Return Value

입력된 경로가 존재하지 않거나 접근할 수 없을 경우 FALSE 를 리턴합니다.

Remarks

이 함수는 FAS_Connect 함수를 사용하기 전에 호출되어야 합니다.
lpPath 값이 NULL 이거나 길이가 0 인 문자열을 입력할 경우 Log 경로는 현재 Ezi-MOTION Plus-R Library 를 사용하는 프로그램이 존재하는 폴더로 설정됩니다.
Log 경로의 기본값은 NULL 로서 현재 프로그램이 존재하는 폴더입니다.

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcEnableLog()
{
	BYTE nPortNo = 1;  // COMM Port 번호

	// Log 를 출력합니다.
	FAS_EnableLog(TRUE);  // 사용하지 않아도 됩니다.

	if (!FAS_SetLogPath(_T("C:₩₩Logs₩₩"))) // C:₩Logs 폴더가 존재하여야 합니다.
	{
		// Log 경로가 존재하지 않습니다.
		Return;
	}

	// 모든 함수들의 Log 는 C:₩Logs 폴더에 출력됩니다.

	// 연결을 시도합니다.
	if (FAS_Connect(nPortNo, 115200) == FALSE)
	{
		// 연결이 실패하였습니다.
		// 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
		return;
	}

	// 연결을 종료합니다.
	FAS_Close(nPortNo);
}
```

See Also

FAS_EnableLog

2 - 3 . Active Level 관련 함수

함수명	내용
FAS_GetIOLevel	Input or Output Active Level 을 읽어 들입니다.
FAS_SetIOLevel	Input or Output Active Level 을 설정 합니다.
FAS_ SaveIOLevel	현재의 Input Active Level or Output Active Level 을 ROM 에 저장합니다. 전원 OFF 후에도 data 가 보존 되도록 저장합니다.
FAS_LoadIOLevel	Input Active Level or Output Active LeveL 을 ROM 영역에서 RAM 영역으로 읽어들입니다.

FAS_GetIOLevel

Input or Output Active Level 을 읽어 들입니다.

Syntax

```
int FAS_GetInputLevel(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned long* ulOLevel
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

ulOLevel

Active Level 값이 저장될 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.
FMM_NOT_OPEN : Board 를 연결하기 전 입니다.
FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.
FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcIOStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uInputLevel;

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
```

```
            // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
            return;
        }

        nRtn = FAS_GetInputLevel(m_nPortNo, iSlaveNo, &uIOLevel);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
    }
```

See Also

FAS_SetIOLevel

Input or Output Active Level 을 설정 합니다.

Syntax

```
int FAS_SetIOLevel(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned long ulOLevel
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

ulOLevel

Active Level 을 설정할 값.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.
FMM_NOT_OPEN : Board 를 연결하기 전 입니다.
FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.
FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long ulOLevel = 0x0000FF00;
                                // 0~7: low active,   8~15 : high active

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
```

```
            // 연결이 실패하였습니다.
            // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
            return;
        }

        nRtn = FAS_SetIOLevel(m_nPortNo, iSlaveNo, uIOLevel);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
    }
```

See Also

FAS_SaveIOLevel

현재의 Input or Output Level 값을 ROM 영역에 저장합니다.

Syntax

```
Int FAS_SaveIOLevel(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcModifyParameter()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }
        // ROM 에 저장합니다.
        nRtn = FAS_SavetIOLevel(nPortNo, iSlaveNo)
        _ASSERT(nRtn == FMM_OK); // 명령이 정상적으로 수행되지 않았다면 멈춥니다.

        FAS_Close(nPortNo);
}
```

See Also

FAS_LoadIOLevel

ROM 영역에 저장되어 있는 Input or Output Active Level 값을 불러옵니다.

Syntax

```
int FAS_LoadIOLevel(
    BYTE nPortNo,
    BYTE iSlaveNo,
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcModifyParameter()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

         // ROM 에서 Input or Output Level 값을 ROM 에서 RAM 으로불러옵니다
        nRtn = FAS_LoadIOLevel(nPortNo, iSlaveNo)
        _ASSERT(nRtn == FMM_OK); // 명령이 정상적으로 수행되지 않았다면 멈춥니다.

        FAS_Close(nPortNo);
}
```

See Also

2 - 4 . Input 제어 함수

함수명	내용
FAS_GetInput	Input / Latch 상태를 읽어 들입니다.
FAS_ClearLatch	해당 bit 의 Latch 를 Clear 합니다.
FAS_GetLatchCount	해당 bit 의 Latch Count 를 읽어 들입니다.
FAS_GetLatchCountAll	Latch Count 전체를 읽어 들입니다.
FAS_ClearLatchCount	해당 bit 의 Latch Count 를 Reset(0)합니다.

*Ezi-IO-RS-I16 시리즈 : 0~15

*Ezi-IO-RS-I8O8 시리즈 : 0~7

FAS_GetInput

Input / Latch 상태를 읽어 들입니다.

(Ezi-IO-RS-I16 시리즈 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 0~7 번)

Syntax

```
int FAS_GetInput(
    BYTE nPortNo,
    BYTE iSlaveNo
    unsigned long* uInput,
    unsigned long* uLatch
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uInput

Input 값이 저장될 변수 포인터.

uLatch

Latch 값이 저장될 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uInput;
        unsigned long uLatch;

        int nRtn;
```

```
        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        nRtn = FAS_GetInput(m_nPortNo, iSlaveNo, &uInput, &uLatch);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

FAS_ClearLatch

해당 bit 의 Latch 를 Clear 합니다.

(Ezi-IO-RS-I16 시리즈 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 0~7 번)

Syntax

```
int FAS_ClearLatch(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned long uLatchMask
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uLatchMask

Latch 를 Clear 할 bitmask 값.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uLatchMask = 0x0000FFFF;   // (Latch 0~15 Clear)

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
```

```
            // 연결이 실패하였습니다.
            // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
            return;
        }

        nRtn = FAS_ClearLatch(m_nPortNo, iSlaveNo, uLatchMask);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
    }
```

See Also

FAS_GetLatchCount

Latch Count 를 읽어 들입니다.

(Ezi-IO-RS-I16 시리즈 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 0~7 번)

Syntax

```
int FAS_ClearLatchCount(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned char iInputNo,
    unsigned long* uCount
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

iInputNo

Latch count 를 읽어 들일 Latch 번호

uCount

Latch count 값이 저장될 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned iInputNo =0;  // (Latch count 번호, 0~15)
        unsigned long* uCount;
```

```
        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        nRtn = FAS_GetLatchCount(m_nPortNo, iSlaveNo, iInputNo, &uCount);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

FAS_GetLatchCountAll

Latch Count 전체를 읽어 들입니다.

(Ezi-IO-RS-I16 시리즈 : 0~15 번 /Ezi-IO-RS-I8O8 시리즈 : 0~7 번)

Syntax

```
int FAS_GetLatchCountAll(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned long** ppuAllCount
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

ppuAllCount

Latch count 값(0~15)이 저장될 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long Latch_All_Count[16] = 0 ; //Latch count 0~16 을 저장할 변수

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
```

```
        {
            // 연결이 실패하였습니다.
            // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
            return;
        }

        nRtn = FAS_GetLatchCountAll(m_nPortNo, iSlaveNo,
                (unsigned long**) Latch_All_Count));

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
    }
```

See Also

FAS_ClearLatchCount

해당 bit 의 Latch Count 를 Reset(0)합니다.

(Ezi-IO-RS-I16 시리즈 : 0~15 번 /Ezi-IO-RS-I8O8 시리즈 : 0~7 번)

Syntax

```
int FAS_ClearLatchCount(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned long uInputMask
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uInputmask

Latch count 를 Reset(0)할 Latch 번호의 Bitmask 값

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uInputmask = 0x000000FF ;  //Latch count 0~7 을 Reset

        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
```

```
        {
            // 연결이 실패하였습니다.
            // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
            return;
        }

        nRtn = FAS_ClearLatchCount (m_nPortNo, iSlaveNo, uInputMask);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
    }
```

See Also

2 - 5 . Output 제어 함수

함수명	내용
FAS_GetOutput	Output 과 Trigger(Run or Stop)상태를 읽어 들입니다.
FAS_SetOutput	Output 을 설정합니다.
FAS_SetTrigger	Trigger 에 대한 설정을 합니다.
FAS_SetRunStop	Trigger 의 실행 또는 정지 설정을 합니다.
FAS_GetTriggerCount	선택한 번호의 Trigger 횟수를 읽어 들입니다.

*Ezi-IO-RS-O16 : 0~15
*Ezi-IO-RS-I8O8 시리즈 : 8~15

FAS_GetOutput

Output 과 Trigger(Run or Stop)상태를 읽어 들입니다.
(Ezi-IO-RS-O16 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 8~15 번)

Syntax

```
int FAS_GetOutput(
    BYTE nPortNo,
    BYTE iSlaveNo
    unsigned long* uOutput,
    unsigned long* uStatus
);
```

Parameters

nPortNo
해당 Board 의 Port 번호.
iSlaveNo
해당 Board 의 Slave 번호.
uOutput
Output 값이 저장될 변수 포인터.
uStatus
Trigger Run/Stop 값이 저장될 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.
FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.
FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uOutput;
        unsigned long uStatus;

        int nRtn;
```

```
        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        nRtn = FAS_GetOutput(m_nPortNo, iSlaveNo, &uOutput, &uStatus);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

FAS_SetOutput

Output 을 설정합니다.

(Ezi-IO-RS-O16 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 8~15 번)

Syntax

```
int FAS_SetOutput(
    BYTE nPortNo,
    BYTE iSlaveNo
    unsigned long uSet,
    unsigned long uClear
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uSet

Output 을 On 할 bitmask 값.

uClear

Output 을 Off 할 bitmask 값.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

uSet 과 uClear 가 같은 bit 를 설정할 경우 uClear 로 처리합니다.

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uSet = 0x0000FF00; (Output 8~15 번을 On 합니다.)
        unsigned long uClear = 0x000000FF; (Output 0~7 번을 Off 합니다.)
```

```
        int nRtn;

        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        nRtn = FAS_SetOutput(m_nPortNo, iSlaveNo, uSet, uClear);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

FAS_SetTrigger

Trigger 에 대한 설정을 합니다.

(Ezi-IO-RS-O16 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 8~15 번)

Syntax

```
int FAS_SetTrigger(
    BYTE nPortNo,
    BYTE iSlaveNo,
    unsigned char uOutputNo
    TRIGGER_INFO* pTrigger
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uOutputNo

Trigger 를 설정하는 Output 번호

pTrigger

Trigger 를 설정할 구조체 변수.(Count, OnTime, Period)

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_NOT_OPEN : Board 를 연결하기 전 입니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Ontime 은 [ms]단위입니다.

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        TRIGGER_INFO Trg_info;
        unsigned char uOutputNo = 0;

        int nRtn;
```

```
Trg_info.wCount   = 100; //Trigger 출력 횟수 설정
Trg_info.wOnTime   = 10; // On 시간 설정
Trg_info.wPeriod   = 30 ; //주기 설정, 30[ms]마다 On : 10[ms] Off : 20[ms]
Trg_info.wReserved1 = 0;
Trg_info.wReserved2 = 0;

// 연결합니다.
if (FAS_Connect(nPortNo, 115200) == FALSE)
{
        // 연결이 실패하였습니다.
        // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
        return;
}

nRtn = FAS_SetTrigger(m_nPortNo, iSlaveNo, uOutputNo, &Trg_info);

_ASSERT(nRtn == FMM_OK);

// 연결을 종료합니다.
FAS_Close(nPortNo);
}
```

See Also

FAS_SetRunStop

Trigger 의 실행 또는 정지 설정을 합니다.

(Ezi-IO-RS-O16 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 8~15 번)

Syntax

```
int FAS_SetRunStop(
    BYTE nPortNo,
    BYTE iSlaveNo
    unsigned long uRun,
    unsigned long uStop
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uRun

Trigger 를 실행할 bitmask 값.

uStop

Trigger 를 정지할 bitmask 값.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

uRun 과 uStop 이 같은 bit 를 설정할 경우 uStop 으로 처리합니다.

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned long uRun = 0x0000FF00; (Trigger 8~15 번을 실행합니다.)
        unsigned long uStop = 0x000000FF; (Trigger 0~7 번을 정지합니다.)
```

```
int nRtn;

// 연결합니다.
if (FAS_Connect(nPortNo, 115200) == FALSE)
{
        // 연결이 실패하였습니다.
        // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
        return;
}

nRtn = FAS_SetRunStop(m_nPortNo, iSlaveNo, uRun, uStop);

_ASSERT(nRtn == FMM_OK);

// 연결을 종료합니다.
FAS_Close(nPortNo);
}
```

See Also

FAS_GetTriggerCount

선택한 번호의 Trigger 횟수를 읽어 들입니다.

(Ezi-IO-RS-O16 : 0~15 번 / Ezi-IO-RS-I8O8 시리즈 : 8~15 번)

Syntax

```
int FAS_GetTriggerCount(
    BYTE nPortNo,
    BYTE iSlaveNo
    unsigned char uOutputNo,
    unsigned long* uCount
);
```

Parameters

nPortNo

해당 Board 의 Port 번호.

iSlaveNo

해당 Board 의 Slave 번호.

uOutputNo

Trigger count 를 읽어 들일 Output 번호.

uCount

Trigger Count 값을 저장할 변수 포인터.

Return Value

FMM_OK : 명령이 정상적으로 수행되었습니다.

FMM_INVALID_PORT_NUM : 연결한 Port 중 nPort 는 존재하지 않습니다.

FMM_INVALID_SLAVE_NUM : 해당 Port 에 iSlaveNo 의 Slave 는 존재하지 않습니다.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusR.h"

void funcInputStatus()
{
        BYTE nPortNo = 1;  // COMM Port 번호
        BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
        unsigned char uOutputNo = 1; (Output 번호. 0~15)
        unsigned long* uCount;

        int nRtn;
```

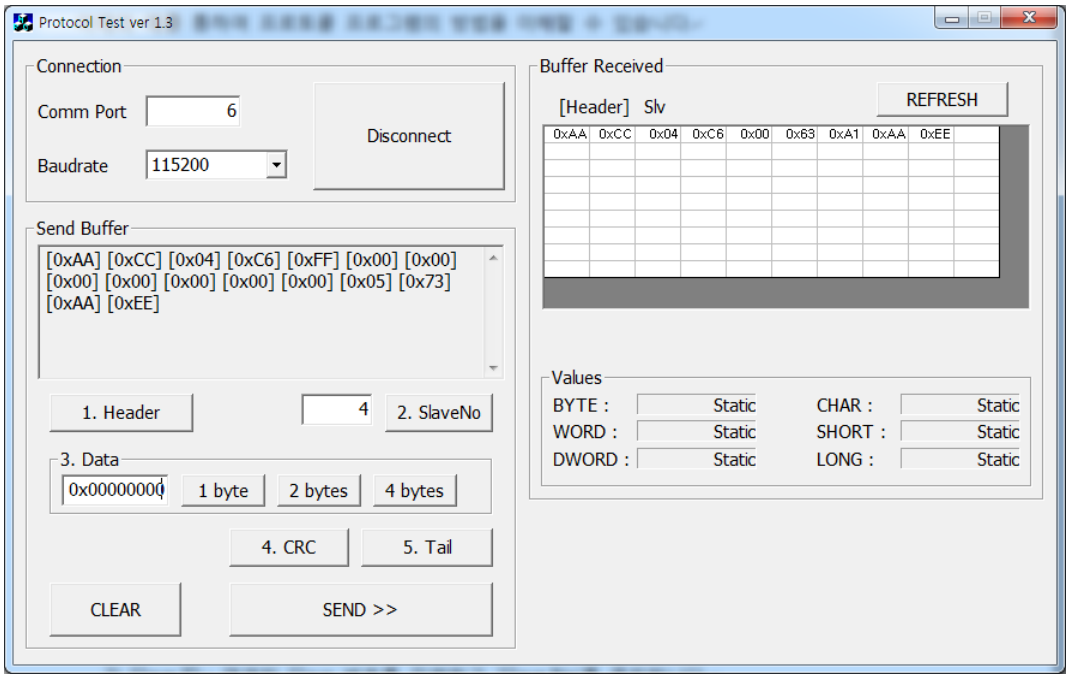
```
        // 연결합니다.
        if (FAS_Connect(nPortNo, 115200) == FALSE)
        {
                // 연결이 실패하였습니다.
                // 연결된 Port 가 아니거나 Baudrate 가 틀릴 수 있습니다.
                return;
        }

        nRtn = FAS_GetTriggerCount(m_nPortNo, iSlaveNo, uOutputNo, &uCount);

        _ASSERT(nRtn == FMM_OK);

        // 연결을 종료합니다.
        FAS_Close(nPortNo);
}
```

See Also

3 . PLC 프로그램용 프로토콜

사용자 프로그램(GUI)이 설치된 폴더에서 Protocol Test_Plus R.exe 를 실행하면 다음과 같은 창이 활성화 됩니다.
아래의 예를 통하여 프로토콜 프로그램의 방법을 이해할 수 있습니다.

(1) FAS_SetOutput 명령용 프로토콜



모든 프로토콜에는 Header, Slave No., Frame type, Data, CRC, Tail 의 내용이 포함되어 있어야 합니다.

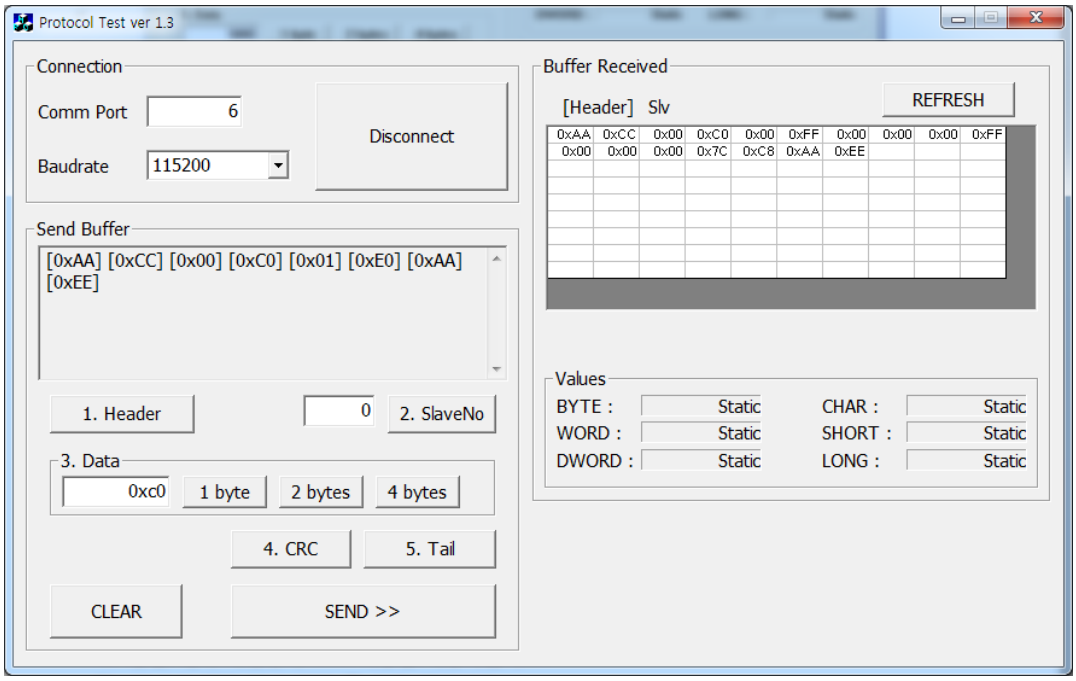
1) 'Comm Port'와 'Baudrate'를 선택한 후 'Connect'를 클릭합니다.
2) Header : 'Header'를 클릭하면 '[0xAA][0xCC]'가 'Send Buffer'창에 나타납니다.
3) Slave ID : 연결된 Slave 번호를 입력하고 'Slave No'를 클릭합니다.
4) Frame type : 명령의 종류를 선택합니다.
   '1-2-1. Frame type 별 송수신 내용'에서 FAS_SetOutput 명령에 대한 내용을 다음표로 확인할 수 있습니다.

Frame type	DLL Library name	Data
198 (0xC6)	FAS_SetOutput	「1-2-1. Frame type 별 송수신 내용」참조

   따라서 '0xC6' 를 입력하고 송신 Frame type 의 크기가 1 byte 이므로, '1 byte'를 클릭합니다.
5) Data : Set Output data (0x000000ff)를 입력하고 4bytes 클릭, Reset Output data (0x00000000)를 입력하고 4bytes 를 클릭합니다.
6) CRC : 이것을 클릭하면 자동으로 checksum 값을 계산하여 2 bytes 크기로 표시됩니다.
7) Tail : 'Tail'을 클릭하면 '[0xAA][0xEE]'가 표시됩니다.

8) 마지막으로 'Send'를 클릭하면 'Send Buffer' 창에 입력된 data 들이 Ezi-IO Plus-R 로 전달됩니다. 이 때 출력을 LED 점등 상태로 확인할 수 있습니다.
9) 명령이 Ezi-IO Plus-R 에 정상적으로 전달되고, Ezi-IO Plus-R 로부터 정상적으로 받은 내용은 'Buffer Received' 창에 표시됩니다.

(2) FAS_GetInput 명령용 프로토콜



1) Header
2) Slave No.
3) Frame type : 'FAS_GetInput' 명령을 위해서는 '0xC0'을 입력합니다. 1byte 클릭
4) CRC
5) Tail
6) Send : 현재 Input 상태를 읽어 들이며, Buffer Received 창에 표시된 내용으로 확인할 수 있습니다.

(3) PLC Programming

실제 프로토콜 프로그램과 상기의 예는 다음의 차이가 있습니다.
상기의 시험용 GUI 프로그램에서는 다음의 기능이 자동적으로 수행되고 있습니다.
1) Header 와 Tail 의 확실한 구분을 위해 'Byte stuffing'기법을 사용 합니다.
「1-1-2. RS-485 통신 프로토콜」을 참조 하십시오.
2) 통신 에러 확인용의 CRC 기능을 사용하십시오.
3) 「1-1-3. CRC 계산 예제」를 참조 하십시오.

FASTECH

Fast, Accurate, Smooth Motion

FASTECH Co., Ltd.
경기도 부천시 원미구 약대동 193번지
부천테크노파크 401동 1202호 (우)420-734
TEL : 032)234-6300,6301 FAX : 032)234-6302
E-mail : fastech@fastech.co.kr
Homepage : www.fastech.co.kr