Integrity Instruments Addressable RS-485 to RS-232 Converter

RS-485

Data Rate	600 – 115200 bps
Max Packet Size	2048 bytes
Max Payload Size	2040 bytes
Error Protection	16-bit CRC

RS-232

Data Rate	600-115200 bps
Receive Data Buffer	2048 bytes
Transmit Data Buffer	2048 bytes
Options:	
Flow Control	RTS/CTS Hardware Handshaking
Parity	Even/Odd
Data Bits	7/8
Stop Bits	1/2

RS-485 Functions

Version	Returns the Firmware Version information Major Version/Minor Version
Send RS-232 Data	Places data in the RS-232 transmit buffer
Transmit Buffer Count	Returns the number of characters in the transmit buffer
Get RS-232 Data	Retrieves the data in the RS-232 receive buffer
Receive Buffer Count	Retrieves the number of characters in the receive buffer
Reset Buffers	Resets all error conditions and RS-232 buffer states.

Packet Format

Dest Address 8 bits	Source Address 8 bits	Packet Length 16 bits	Sequence Number 8 bits	Command Response 8 bits	Data …	CRC 16 bits

Packet Fields

Packet Field	Description
Dest Address	8 bit packet destination address 0x00: Master address

	0x01-0xFE: Slave address 0xFF: Broadcast address
Source Address	8 bit packet source address
Packet Length	16 bit packet length: # of bytes following starting with and including Sequence Number
Sequence Number	Calculated by Master and returned in Slave response. 0x00 is reserved for the starting sequence number whereby the Slave will not check sequencing on a 0x00.
Command Response	8 bit command/response value
Data	0…x bytes of data. **OPTIONAL**
CRC	16 bit CRC of the entire packet.

Commands & Responses

Master Command	**Slave Response**	**Data Bytes**	**Description**
0x01		0	Send RS-232 data. Sequence number of checked by the slave. (The only command sequence check)
	0x81	0	RS-232 data sent
	0x82	0	RS-232 data not sent (xmit buffer overrun)
0x03		0	Transmit buffer count
	0x83	2	Transmit buffer count Data: Bytes in transmit buffer
0x04		2	Get RS-232 data. Data: Max. number of bytes to transfer
	0x84	0…x	Get RS-232 data. Data: Receive buffer contents
0x05			Receive buffer count
	0x85	2	Receive buffer count Data: Bytes in receive buffer
0x06			Firmware version
	0x86	2	Firmware version Data: Major-Minor version number
0x07			Reset buffers
	0x87		Reset buffers

CRC Example

```
struct TYPE_485 {
   // Packet Variables and buffers
   unsigned char   destination;          //Destination address
   unsigned char   source;               //Source Address
   unsigned int    length;               //Payload length
   unsigned char   data[BUFFERSIZE - 4];   //Payload buffer

   // Internal packet system variables
   unsigned long   pTimer;               //Packet Timer
   unsigned int    pStatus;              //Packet byte count
   unsigned int    CRC;                  //Used to calculate CRC
```

```
    unsigned int    dataPtr;            //Used to refrence data buffer

}modBus;


/* Table of CRC values for high-order byte */
unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40
, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40 } ;

/* Table of CRC values for low-order byte */
unsigned char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40} ;

int calcCRC( void ) {
    unsigned int i, index, loops, rVal;
    unsigned int chkSumHi, chkSumLo;
    unsigned char uchCRCHi, uchCRCLo;

    // retreive the packet checksum
```

```
   chkSumHi = modBus.data[modBus.length-2];
   chkSumLo = modBus.data[modBus.length-1];

   //Setup checksum accumulator
   uchCRCHi = 0xFF;
   uchCRCLo = 0xFF;

   index = uchCRCHi ^ modBus.destination;              //Include,
   uchCRCHi = uchCRCLo ^ auchCRCHi[index] ;              //... Destination
   uchCRCLo = auchCRCLo[index] ;

   index = uchCRCHi ^ modBus.source;                  //Include
   uchCRCHi = uchCRCLo ^ auchCRCHi[index] ;              //... Source
   uchCRCLo = auchCRCLo[index] ;

   index = uchCRCHi ^ (modBus.length>>8);               //Include
   uchCRCHi = uchCRCLo ^ auchCRCHi[index] ;              //... Length High Byte
   uchCRCLo = auchCRCLo[index] ;

   index = uchCRCHi ^ (modBus.length & 0xFF);          //Include
   uchCRCHi = uchCRCLo ^ auchCRCHi[index] ;              //... Length Low byte
   uchCRCLo = auchCRCLo[index] ;

   //Setup Loop counter
   loops = modBus.length-2;
   i =0;
   while(loops--)  {                              //Calc the data
      index = uchCRCHi ^ modBus.data[i++];
      uchCRCHi = uchCRCLo ^ auchCRCHi[index] ;
      uchCRCLo = auchCRCLo[index] ;
   }

   modBus.CRC = (uchCRCHi << 8) + uchCRCLo;

   if (uchCRCHi == chkSumHi && uchCRCLo == chkSumLo) {
      rVal = 1;
   }
   else {
      rVal = 0;
   }
   return rVal;

}
```