BrianHetrick.com Application Note AN-6

Introduction to Asynchronous Serial Communications

Contents

Goal and Overview

Communication between and among computing devices is a ubiquitous need. In robotics and home automation, functions are typically allocated among dispersed computing elements and overall activities require cooperation among the computing elements. This coordination requires communications. Serial communication pathways can be simple, reliable, and fast.

Most data communications pathways are serial: bits are sent one at a time across some medium. This Application Note discusses a particular type of serial communication, asynchronous (or start-stop) character communication.

Concepts

The fundamental capability provided by asynchronous serial communication is the transmission and reception of a *stream* of *characters*. A character is a fixed-length sequence of *bits*. Bits are binary digits 0 and 1. Characters become available to a receiver as they are produced by a sender: if no sender is producing characters, there is nothing to receive. Characters are transmitted between sender and receiver by modulating a signal. The signal has two states, called *mark* and *space*. The space state is used for an *idle* line, that is, when there is nothing being transmitted.

To send a single character, the transmitter first sends a *start bit*. This bit must be a mark so its presence can be distinguished from the idle state. The transmitter then sends the bits of the character one at a time, in a specified order. The transmitter might then send a single-bit character checksum (parity bit). Finally, the transmitter returns the signal to the idle state for a period of time. The amount of time the signal spends representing each bit (the start bit and the data bits) is called the *symbol time* or bit time; the number of symbols per second is called the *symbol rate*, or the *baud rate*. (In the

© 2016 Brian Hetrick. This document may be used and copied in accordance with the terms of the Creative Commons Attribution-ShareAlike 3.0 United States license. The text of this license is available at: http://creativecommons.org/licenses/by-sa/3.0/us/

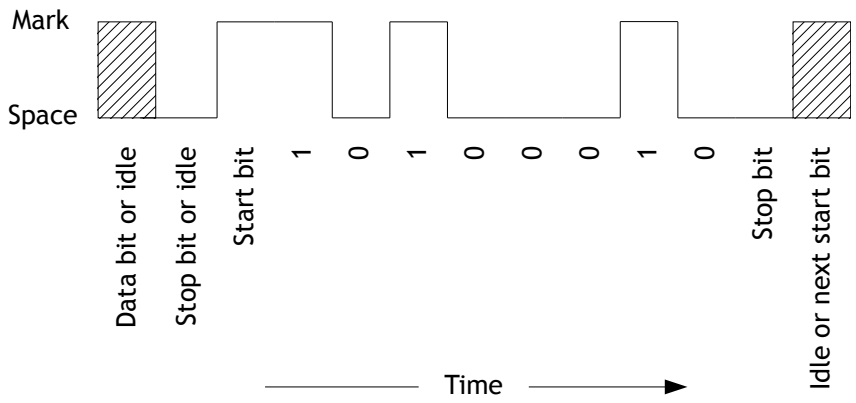

Figure 1. Asynchronous serial representation.

case of binary encoding, as with mark and space states, the two terms are equivalent.)

The sender and receiver must have a common understanding of several aspects of this communication scheme for communication to be successful. These aspects are: the conditions that constitute the mark state and the space state; the correspondence between mark and space and the 0 and 1 data bits; the number of data bits composing a character; the type of parity bit used (even, odd, 0, 1, or none); the order in which character bits are transmitted; the baud rate; and the length of the post-character idle time.

In what might be called "the" asynchronous serial protocol, mark is used for the data bit 1 and space for the data bit 0, characters are commonly 8 bits, parity bits are commonly omitted, character bits are transmitted in sequence from the lowest order bit to the highest order bit, and commonly the post-character idle time is one symbol time.

Historically, characters have required anywhere from 5 to 9 data bits; current systems typically deal with *octets*, or bytes, of 8 bits[1]. A wide variety of mark and space conditions are in current use: single voltages, differential voltages, presence or absence of current, and presence or absence of light are some. A wide variety of baud rates are common: 300, 9.6k, 115.2k, and 10M bits per second are representative. Typical post-character idle times are 1 and 2 symbol times, although 1.5 symbol time also finds use. The post-character idle time is sometimes called the *stop bit* or stop bits.

As an example, consider the character "E" (Unicode point U+0045, or ASCII character 4/5, binary 0100 0101 in both ASCII and UTF-8). When transmitted on an asynchronous serial line as an 8-bit character with no parity and one stop bit, the character "E" becomes a time-varying signal as shown in Figure 1.

Note the bits of the character are transmitted from low order to high order, and a mark is used to represent a 1 bit.

The asynchronous serial signal is implicitly clocked: there is no separate clock signal as with some other ("synchronous") technologies, and the signal uses levels rather than edges to encode bits. The timing of signals is therefore critical in this arrangement. The transmitter generates mark and space conditions with durations equal to the signal time. The receiver typically samples the communications medium somewhere midway through the expected time window allocated to the bit; the process starts at the detected start of the start bit. The absolute maximum clock rate difference (called *clock skew*) between the sender and receiver is ½ bit for each character; with one start bit, eight data bits, no parity bit, and one stop bit, the absolute maximum clock skew possibly permitting communication is 5%. A clock skew of under 2% allows bit samples to occur about 2/3 of the way through the expected bit time, which in turn allows the signal to survive signal edge degradation. The 2% limit also allows the sender or receiver to approximate a baud rate it cannot generate exactly. A 2% clock skew is typically regarded as acceptable, although clock skew under 1% is preferred.

A clock rate mismatch between the sender and the receiver typically results in an expected stop bit not being the space condition. This condition is known as a

1 But see the section TIA/EIA-485 for a common exception.

framing error, and is typically reported by the serial line hardware. The presence of framing errors is highly suggestive of a clock rate mismatch, but not definitive. Other causes of framing errors include loss of synchronization between the transmitter and receiver, where the receiver misidentifies the start bit, and excessive noise on the communications line. One or two character times of idle line will typically resynchronize the transmitter and receiver, but will not affect clock rate mismatches. Excessive noise can render a serial link unusable.

Framing errors can also be deliberately introduced by imposing a mark condition continuously for an entire character time (including start and stop bit times) or longer. When introduced deliberately, such a framing error is called a *break*. In data communications protocols using a break for routine signaling, the break time is commonly 1 to 4 symbol times longer than a character. When used as an extraordinary attention symbol, break times of 100 to 500 ms are typical.

The asynchronous serial protocol described above is directly supported by devices called Universal Asynchronous Receiver/Transmitters (*UART*s). These devices are typically implemented as an on-chip peripheral device in microcontrollers, and are commonly available to PCs either on the motherboard or as a USB device.

Very low-end microcontrollers might not include an on-chip UART peripheral; in such cases, UART peripherals are available as separate packages using SPI and I^2C to communicate with the microcontroller. Also, the availability of an interrupting programmable timer permits implementation of the protocol entirely in software through "bit-banging" GPIO lines. Bit-banging is typically effective for baud rates up to about 20k baud. A microcontroller with a 5 MHz instruction rate and a 25 instruction interrupt overhead would be 40% consumed simply by sampling the serial line four times per bit at 20k baud. The use of a UART would lower the interrupt load to 1%.

By the nature of the asynchronous serial protocol, the line idle condition is transparent to the character stream transmitted. At 115.2k baud, a half-character time idle is 43 µs. This condition is detectable when specifically looked for, but inconspicuous and would be invisible to typical data transfer. Idle condition modulation of a communications line can serve as a covert data channel. The availability of a timing side channel is a property shared with a great many technologies, perhaps all technologies. Nonetheless, this property may affect the suitability of the asynchronous serial communication technology in high-security environments.

Standards and Practices

TIA/EIA-232

The TIA/EIA-232[2] standard, also known as "RS-232"[3], is an electrical, signaling, and physical standard for point to point asynchronous serial communications. The equivalent international standards are ITU V.24 (signal descriptions and names) and ITU V.28 (electrical).

The conceptual model underlying the standard is one of two devices of a class called data terminal equipment (*DTE*) communicating through some medium; the medium is interfaced to with devices of a class called data communication equipment (*DCE*). Thus the DTE device communicates directly with a DCE device which somehow communicates with another DCE device which directly communicates with another DTE device, and the effect is that the DTE devices communicate with one another. Typically DTE devices are computers or display terminals, and DCE devices are modems. The full standard provides for primary and secondary communication pathways and a variety of special signals used for establishing and tearing down the communications pathways. Revision D of this standard established the DB-25 connector standard; revision F, the currently ubiquitous DE-9 connector[4] standard. Male connectors are assigned to DTE; female to DCE.

In TIA/EIA-232, one signal is used to send data from the DTE to DCE, and a separate signal for data from DCE to DTE. The two signals are logically independent (and may run at different baud rates, although this is uncommon) but are relative to a common signal ground line. A space is represented by a voltage of +3 to +25 volts relative to signal ground, and a mark by a

2 TIA stands for Telecommunications Industry Association, at http://www.tiaonline.org. EIA stands for Electronic Industries Alliance, now defunct.

3 RS stands for "recommended standard."

4 The DE-9 connector is commonly misidentified as "DB-9" due to it functionally replacing the previously common DB-25 connector. The "B" and "E" are connector sizes, the two connectors have different sizes, and DE-9 is correct. It's a DE-15 for a VGA connector, too.

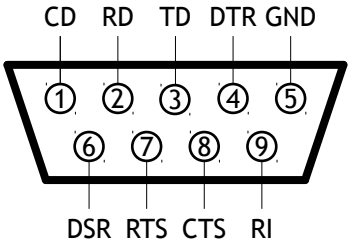

Figure 2. DE-9 connector TIA/EIA-232 assignments.

voltage of -3 to -25 volts relative to signal ground. The standard calls for transmitters to generate ±15V to ensure the receiver sees at least ±3V; common single-supply transceivers actually generate about ±9V. The mark state corresponds to a 1 data bit, the space state to a 0 data bit. Control signals also use these voltages but are deasserted in the mark state and active or asserted in the space state. Transceivers typically pull up their input lines so that when disconnected the data line appears idle and the control signals appear asserted.

This means a 1 is represented by a negative voltage, a 0 by a positive voltage, the waveform in Figure 1 is upside-down, and control signals are properly named with the active low overline (such as $\overline{\text{DSR}}$). Welcome to telecommunications.

TIA/EIA-232 transceivers are typically not galvanically isolated from their equipment: the signal ground is tied to the chassis ground. This implies that the chassis grounds of the DTE and DCE must be at the same potential to avoid ground loops.

Connecting DTE and DCE

A minimal bidirectional TIA/EIA-232 connection uses three wires: signal ground, transmit data (TD or TxD, DTE origin), and receive data (RD or RxD, DCE origin). With the addition of two additional lines, ready to receive (RTR, which replaces the earlier request to send [RTS], DTE origin) and clear to send (CTS, DCE origin), each side can signal its ability to accept data. This can be used as the basis of a flow control discipline, where each side avoids sending data when the other side is unable to process it[5]. Alternately, the signals data terminal ready (DTR, DTE origin) and data set ready (DSR, DCE origin) are sometimes used for flow control.

Pin	Origin	Signal
1	DCE	Carrier Detect (CD). Asserted by the modem when connection has been established.
2	DCE	Receive Data (RD or RxD). Data inbound through the modem.
3	DTE	Transmit Data (TD or TxD). Data outbound through the modem.
4	DTE	Data Terminal Ready (DTR). Asserted to instruct a modem to accept an inbound connection attempt. Sometimes used with direct connections for flow control.
5	Common	Signal Ground (GND). The potential against which the other signals are measured.
6	DCE	Data Set Ready (DSR). Asserted by the modem to indicate the modem is powered on. Sometimes used with direct connections for flow control.
7	DTE	Request to Send (RTS) or Ready to Receive (RTR). Used with half-duplex modems to request the data direction be from this terminal to the remote terminal. RTR is used with direct connections for flow control.
8	DCE	Clear to Send (CTS). Used with half-duplex modems to indicate the data direction is from this terminal to the remote terminal. Used with direct connections for flow control.
9	DCE	Ring Indicator (RI). Asserted by the modem to indicate an inbound connection attempt. Typically DTR would be asserted to indicate the modem should answer.

Figure 3. DE-9 TIA/EIA Signals

The common 9-pin connector assigns signals as shown in Figure 2. This figure shows the view looking into the DE-9 male connector used by DTE. Looking into the DE-9 female connector used by DCE, the pins would be reversed left to right. The meanings of these signals are shown in Figure 3.

The signals are asymmetric: three signals go to the modem, five signals come from it. This is the reason for

5 This is termed *hardware* flow control. See the section Transparency Issues for an alternative termed software flow control.

TIA/EIA-232 transceivers with three drivers and five receivers or the reverse: these support "full modem control."

TIA/EIA-232 uses single-ended signals, rather than differential signals, and does not call for these signals to be terminated. It therefore has limited distance and data rates for which it is useful. The standard specifies a maximum signal rate of 20k baud, a maximum cable length of 50 feet, and a maximum cable capacitance of 2500 pF. However, the signaling scheme is often used outside these limits with success. Over a short distance (0-2 meters), data rates of 115.2k baud can be routinely achieved, and both UARTs and transceivers commonly support this data rate. With low-capacitance cables and data rates of 2.5k baud, distances of 1 km have been achieved. Nonetheless, TIA/EIA-232 was designed for, and would typically be considered only for, low- to moderate-speed local communication.

For historical reasons having to do with modem technology, recently common baud rates are powers of 2 times 300: 300, 1.2k, 9.6k, 19.2k. The rate 9.6k seems more or less traditional for a "moderate" speed link. The rate 115.2k is nonstandard but not uncommon for a short "fast" link.

Connecting DTE and DTE

There is no standard for directly connecting two DTE devices. There are, however, a number of practices.

What is called a "null modem cable" can be used to connect two DTE devices each using the standard DE-9 male connector. This type of cable typically interchanges pins 2 and 3 (thus connecting each device's TD line to the other device's RD line) and 7 and 8 (thus connecting each device's RTS/RTR line to the other device's CTS line) between the two ends, while pin 5 (ground) is wired straight through. In this arrangement, data exchange is possible when each device has only one driver and one receiver, and data exchange with flow control is possible when each device has only two drivers and two receivers. This arrangement is typically used to communicate between two PCs or between a PC host and a microcontroller.

The PLC manufacturer Unitronix has cleverly used the telephony RJ-25[6] jack and plug to produce a fixed gender, role-independent connector for serial lines. The six conductors of the 6P6C RJ-25 jack are assigned as shown in Figure 4. There are four significant features of the use of this connector and these assignments:

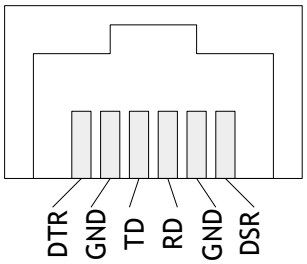

Figure 4. Unitronix RJ-25 connector TIA/EIA-232 assignments.

- The RJ-25 jack and plugs are ubiquitous, smaller, and much less expensive than DE-9 connectors.
- There is no distinction between the roles of DTE and DCE: all devices are treated as DTE. Thus any device asserts DTR when it can receive data, transmits data on TD, receives data on RD, and observes DSR for outbound flow control. The DTR and DSR lines are used as if they were RTR and CTS, respectively.
- A four-conductor RJ-14 telephony cable provides communication, a 6-conductor RJ-25 cable communication and flow control.
- The connecting cable is straight-through with a reflection. The reflection is easily introduced into flat cable (such as indoor satin telephony cable or ribbon cable) by putting on one connector "upside

6 The multipair RJ series of telephony jacks and plugs are commonly misidentified as all RJ-11. RJ-11, RJ-14, and RJ-25 are a series of telephony jacks and corresponding plugs that provide one, two, and three wire pairs, respectively. The jacks and plugs are physically identical and interchangeable except for the number of connections provided. There are six possible pin positions: RJ-11 populates the innermost two, RJ-14 the innermost four, and RJ-25 all six. The jacks and plugs are generically described as 6P2C, 6P4C, and 6P6C respectively; P and C refer to positions and conductors. The Unitronics arrangement requires six conductors in a telephony jack and plug, hence RJ-25.

Another popular practice, the Yost arrangement, is frequently described (even by Yost himself) as using RJ-45 plugs and jacks. RJ-45 is an 8P8C telephony connector, but is incompatible with the 8P8C connector used for Ethernet. The Yost arrangement uses Ethernet jacks and plugs, not telephony plugs. It does not help that Ethernet jacks and plugs are typically called "RJ-45" even by vendors, although the actual standard describing them is TIA/EIA-568-B. Actual telephony RJ-45 jacks and plugs are sometimes called "telephone" RJ-45 to distinguish them from "data" (Ethernet) connectors that are called, but are not, RJ-45.

down." The half-twist serves to reverse the order of the signals and put signals driven on the right side of one connector to be observed on the left side of the other connector.

Appendix A of this Application Note describes a circuit providing a USB serial peripheral using TIA/EIA-232 signal levels presented through a Unitronics arrangement connector.

20mA

A practice has arisen to allow moderate speed point to point asynchronous serial connections to be extended to and beyond 1 km. This practice is to use a four-wire cable providing one 20 mA current loop in each direction. Equipment at each end of the connection imposes a 20 mA current on one pair of wires to send a mark condition, and detects a 20 mA current on the other pair as the receipt of a mark condition. This arrangement permits full-duplex data communication between the endpoints, although without hardware flow control. As current rather than voltage is used for signaling, the connection is almost immune to induced noise, but is more vulnerable to poor splices than is a voltage-based arrangement.

Typically outdoor telephone cable is used for the connection, each end uses an operational amplifier arranged as a controlled current source as a transmitter, and each end uses a photocoupler as a receiver. Conveniently, the current requirement of a typical photocoupler is 20 mA. The use of photocouplers provides galvanic isolation between the stations so ground potential differences of several hundred volts can be accommodated. Connectors are typically screw terminals.

Variations on this arrangement include the use of 40 mA and 60 mA current loops instead of 20 mA current loops.

TIA/EIA-485

The TIA/EIA-485 standard, also known as RS-485, is an electrical (only) standard used for asynchronous serial communications. This standard uses *differential signaling*: that is, there are two signal lines called A and B, the mark condition (binary 1) is defined by the voltage of the B line being at least 200 mV higher than that of the A line, while the space condition (binary 0) is defined by the voltage of the A line being at least 200 mV higher than that of the B line[7]. When used with terminated twisted pair cable, this signaling is highly resistant to electrical noise from induced currents. Transceivers typically allow voltage differences of -7 to +12 V from local ground; this means a signal ground connection, while called for in the standard, is typically not necessary.

The TIA/EIA-485 arrangement permits much higher data rates than that of TIA/EIA-232: the standard supports data rates of up to 35M baud at 10m and 100k baud at 1200m.

TIA/EIA-485 can be used for full-duplex point-to-point communications by using four wires, one pair for data in each direction, each pair terminated at the receiver for that pair. This arrangement is commonly used when the distance or data rate required of a point-to-point link cannot be met by TIA/EIA-232 or 20 mA. Transceivers with separate transmitters and receivers are readily available to implement this variation.

Appendix B of this Application Note describes a circuit providing a USB full-duplex point-to-point communications peripheral using TIA/EIA-485 signal levels and including both line termination and line biasing. The signals are presented through either screw terminals or an RJ-14 connector.

Point-to-point TIA/EIA-485 connections also find use in extending what would otherwise be an impossibly long TIA/EIA-232 connection. This requires a modem to translate the TIA/EIA-232 connection and flow control to and from a TIA/EIA-485 connection and flow control. Appendix C of this Application Note describes such a modem. Longer point-to-point links can be arranged with repeaters; Appendix D of this Application Note describes such a repeater.

However, the use envisioned by the standard is a network rather than a point-to-point link. Typically TIA/EIA-485 is used as a linear bus[8]: up to 35 "unit

7 Beware: a certain manufacturer cleverly mislabels the A and B connections of its transceivers as B and A, respectively, while other manufacturers use non-standard terminology such as "Data+" and "Data-" to avoid the mess caused by the first manufacturer.

8 Topologies other than a line—for example star arrangements—are ill-advised from a signal integrity standpoint and disallowed by the standard. Non-linear topologies can be created with repeaters, however, as long as each segment is linear.

load" devices may be attached. Common currently available transceivers present ¼ "unit load" to the bus, allowing up to 140 devices to be connected to a single bus. The bus is typically terminated at both ends by connecting the A and B lines through a resistor with a value of the characteristic impedance of the cable. The bus is typically biased into the space condition by a voltage divider arrangement near its center; this also prevents the bus common potential from drifting far away from the devices' ground potentials. Only one device at a time may transmit on the bus; other devices must effectively disconnect their transmitters. Transceivers with disconnectable transmitters on the same lines as the receivers are readily available to implement this arrangement.

This bus is typically implemented with a single *master* device near the physical bus center, supplying biasing, with remaining devices taking the role of *slave* devices. The bus is used in an application protocol where the master device interrogates a slave device; after a short delay (typically one character time, to permit the master to disconnect its transmitter from the bus) the slave device responds. Each slave device must have some way of distinguishing when it, rather than some other slave device, is being interrogated and thus granted the right to transmit on the bus. Typically this is achieved by assigning each slave device an identifier that is unique among those used on the bus, called an *address*, and the master incorporating the address of the intended slave device at a specified point in the interrogation message.

TIA/EIA-485 networks typically use the same asynchronous serial protocol (start bit, data bits in low order to high order sequence, stop bit) as does TIA/EIA-232. However, a variation in common use is to use 9 bit characters rather than 8 bit octets. The extra high order bit is used solely to mark the address for which a master-generated message is intended, and this address is positioned as the first character in each message. This permits slave devices to quickly recognize when they are not addressed and to simply ignore all subsequent data traffic until another character with high order bit set is received. This variation is so widespread in TIA/EIA-485 networks it might be assumed unless disclaimed.

Typically a standard UART is used with a TIA/EIA-485 transceiver to drive a TIA/EIA-485 line: a station asserts a signal to enable the transmitter when it wishes to send data to the bus, and deasserts it otherwise. It is critical that a station know when its transmission is complete so that it can deassert the signal attaching the transmitter to the bus and permit other stations to send data. This is not typically a problem for microcontrollers that receive an interrupt when the UART peripheral drains its transmit buffer. It is more commonly an issue for PCs, where an operating system might delay the notification of transmission completion or even require the application to poll to determine transmission completion.

TIA/EIA-485 does not specify the connectors to be used. Commonly used connectors include screw terminal blocks, DE-9 connectors, and RJ series telephony connectors, with a variety of signal assignments.

Appendix E of this Application Note describes a circuit providing a master station for an 8- or 9-bit character TIA/EIA-485 network. A PC (for example) communicates with the master station over an 115.2k baud full duplex point-to-point TIA/EIA-485 communication link, such as that described in Appendix B. The master station provides network line biasing but not termination, uses a network baud rate set by the PC, engages its network transmitter only on demand, and uses a protocol described in Appendix G to carry binary network data over the character connection to the PC.

Logic Levels

Close attention to the TIA/EIA-232 standard reveals that the forbidden zone of -3 to +3 V is a constraint on signals, not receivers: the receiver can do what it pleases with signals in that range. Some bright person somewhere realized "what it pleases" might very well be recognizing a space at +2V or thereabouts and above, and a mark otherwise. And thus was born the simple comparator as a TIA/EIA-232 receiver.

This in turn gave rise to some other bright person realizing logic levels would be compatible with that particular arrangement. And thus was born the NOT gate as a transmitter.

And from that came the demand on UART and microcontroller vendors to be able to tell a UART device to invert all signals on the '232 side, so not even a NOT gate was needed.

Inside a device, there is no particular need to throw negative voltages around for signaling. There is no particular need to obey the "high space, low mark" convention. The signal police will not come after you if

you connect the transmit and receive lines of two microcontrollers' UARTs with a simple crossover, instead of using standard-conformant transceivers in the middle. The standard is for interoperation of devices designed in isolation from one another. Inside a single device, there in no need to interoperate with strange devices.

But basing products intended to interoperate with standard serial line products on these practices is a Very Bad Idea. That has not stopped people from doing it, however.

Particularly in the USB serial adapter space, a device may be presented as a TIA/EIA-232 device when it in fact generates logic levels and will not interoperate with a device requiring actual TIA/EIA-232 levels. Typically such devices will be marketed as a "serial interface," or will say "compatible with most motherboards." But sometimes the vendors simply lie and say "RS-232." Such devices cost less than $5, and typically incur more frustration than they save in purchase price.

The very low-cost devices also commonly use counterfeit parts, which in turn causes both manufacturers and buyers substantial headaches.

Feel free to use logic level signals inside a device. Feel free to use logic levels between devices. But recognize "asynchronous serial interface" is a rather broad style of interfacing, "EIA/TIA-232" is a specific standard for interoperability, and a $3 USB eBay "serial port" might not be useful.

LIN

A relative newcomer to the serial communications networking scene, the Local Interconnect Network (*LIN*) protocol provides moderate speed networking among physically close devices using asynchronous serial communications. LIN networks consist of a master node and up to 15 slave nodes. A single signal wire is used; all devices must share a common ground; each node supplies partial biasing of the signal line. A master-slave arrangement is used: slaves transmit only in response to interrogation from the master, preventing collisions. Typical baud rates are 9.6k and 19.2k. Similarly to I^2C and CAN, LIN networks have the concept of a "dominant" and a "recessive" bit: line idle is the recessive bit, and any node can transmit a dominant bit by pulling the line low. As the network was developed as an inexpensive in-vehicle network, the space state (line idle and recessive bit) is typically 12V, although 5V is also used in non-vehicle networks.

A unique feature of LIN is its inclusion of a synchronization byte in messages. This byte allows slaves using RC oscillators to determine the network baud rate in terms of their current oscillator frequency at each message.

LIN networks use a sophisticated protocol embodying attention breaks, synchronization bytes, addresses or message identifiers, packet payloads, and checksums; yet the only hardware needed is a UART, and even that can be dispensed with via software and bit-banging. Many microcontrollers (particularly those from Microchip Technology, Freescale, and ST Microelectronics) contain UART devices directly supporting the LIN 13-bit time attention break and automatic baud rate setting on the LIN synchronization byte. In addition to its use as a sub-bus in automotive settings, LIN is finding use as an in-device bus in appliances such as air quality sensors and annunciators in home and building automation.

Appendix F of this Application Note describes a circuit providing a master station for a LIN network. A PC (for example) communicates with the master station over an 115.2k baud full duplex point-to-point TIA/EIA-485 communication link, such as that described in Appendix B. The master station uses a network baud rate set by the PC, engages its transmitter only on demand, and uses the protocol described in Appendix G to carry binary network data over the character connection to the PC.

Transparency Issues

The previous sections of this Application Note have discussed the transmission of characters between devices, but it has not considered a "character" as other than a sequence of bits. While there has been a distinction drawn between multi-station networks ("binary") and point to point connections ("character"), the reason for this distinction might not be clear.

A character is indeed a sequence of bits. But characters have something sequences of bits lack: meaning. Characters have semantics. This section discusses the effect characters and the transport medium may have on one another.

Unicode, as well as its predecessor national and international standards, reserves certain characters as *control characters* and assigns semantics to them. The C0

Code	Character	Semantics
0x00	NUL	Filler; may be freely introduced and removed
0x01	SOH	Indicates message header follows
0x02	STX	Terminates message header, indicates message text follows
0x03	ETX	Terminates message text
0x04	EOT	Indicates end of transmission
0x05	ENQ	Request for receiving station to identify itself
0x06	ACK	Indicates successful receipt of message
0x07	BEL	Request for operator attention signal
0x08	BS	Move cursor position one cell left; used for character composition
0x09	HT	Move cursor position horizontally to next tabulation stop
0x0A	LF or NL	Move cursor position vertically one line down (LF) or move cursor to start of next line (NL) (prior agreement among parties required)
0x0B	VT	Move cursor position vertically to next vertical tabulation stop; may also return cursor to left margin (prior agreement among parties required)
0x0C	FF	Move cursor position vertically to start of next page; may also return cursor to left margin (prior agreement among parties required)
0x0D	CR	Move cursor position horizontally to left margin
0x0E	SO	Switch to an alternate character set (prior agreement among parties required)
0x0F	SI	Return to the normal character set
0x10	DLE	Indicate the following characters address the data link itself rather than the remote interchange party; return to communication with remote party is data-link dependent
0x11	DC1	Device control 1 (device on)
0x12	DC2	Device control 2 (device on)
0x13	DC3	Device control 3 (device off)
0x14	DC4	Device control 4 (device off)
0x15	NAK	Indicates failure to receive message
0x16	SYN	Idle character for synchronous links
0x17	ETB	Indicates the end of a block for transmissions divided into blocks
0x18	CAN	Previous data is to be disregarded
0x19	EM	The end of the recording medium has been reached
0x1A	SUB	Substitute for an invalid character or a character that does not exist in the character set
0x1B	ESC	Indicates the start of a control sequence; allows access to C1 set using only 7-bit characters
0x1C	FS	Indicates the start of a new file
0x1D	GS	Indicates the start of a new group of records in a file
0x1E	RS	Indicates the start of a record in a file or a group
0x1F	US	Indicates the start of a new unit (item) in a record
0x20	SP	Indicates a graphic space; may also be used to indicate the start of a new sub-unit within a unit
0x7F	DEL	Substitute for a character deleted from the medium

Figure 5. C0 control characters.

set of control characters are shown in Figure 5. In addition to these C0 control characters, there is a set of C1 control characters. The C1 control characters are assigned codes in the range 0x80 to 0x9F. However, the C1 characters are not as well known or as widely interpreted as are the C0 characters.

There is a long history here, hinted at by the SI and SO characters in the C0 set and the SS2 and SS3 characters in the C1 set. The history includes 7- and 8-bit character sets and 95- and 96-character repertoires and 32- and 33-character control sets and escape sequences to invoke character sets into character code ranges and locking and single character shifts between graphic sets and control sets and an entire panoply of similar complications that make the meaning of a character code dependent on context. Nobody uses those standards anymore. Unicode and UTF-8 did away with all that, and a good thing too, even if now we have to deal with things such as the basic multilingual plane, surrogate character pairs, and combining diacritics.[9]

But the old ways never fully go away, and there is always the question of whether a data path is "transparent." Typically the term "transparent" means a character sent by the transmitter is received unchanged by the receiver and has no side-effects on the transmission medium. Typical non-transparencies include transmission medium sensitivity to the high order bit (the bit may be stripped, the bit may be set or reset based on a parity rule, the entire character may be dropped based on a parity rule, and so forth) and transmission medium sensitivity to certain control characters. The characters NUL, ENQ, DLE, DC1, DC3, and ESC are particularly risky.

The characters DC3 and DC1 deserve special mention, as they are the "software flow control" characters. When used for flow control, these characters are known an XOFF (transmitter off) and XON (transmitter on) respectively. These characters are properly used as follows: XOFF and XON can be emitted by any party at any time regardless of any data exchange taking place. XOFF is emitted by a party when its input buffer becomes close to full, and XON is emitted when the input buffer subsequently becomes close to empty. The communication partner honors an XOFF by ceasing transmission until an XON is received. Receipt of an XON should be acknowledged with an XON or XOFF, as the device is or is not ready to accept data, unless the side receiving the XON has "recently" sent an XON. This simple discipline permits flow control over a serial line without hardware flow control and even permits a party to inquire whether flow control is still in effect.

Note the "party" wording: this includes the endpoints, obviously, but it also includes the transmission medium itself. Any intermediate devices handling the data stream may generate and handle flow control requests independently. It is entirely legitimate for a terminal server or other relay to handle flow control for each side of a serial conversation separately and with respect to its own buffers. An implication of this is that the XOFF and XON characters transmitted or received by an endpoint are independent of the XOFF and XON characters received or transmitted by the other endpoint.

Flow control aside, why might one worry about control characters and transparency? A wire obviously does not care what particular bit patterns are transmitted over it. But wires are the special, simple case. In contrast to a wire, a modem cares; a terminal server cares; a telnet connection cares; a multi-user time sharing system cares; a PC's serial port cares (and Linux and Windows care differently); and a connection that happens to otherwise be a multi-user computer's operator's console cares very much indeed. Break conditions and various control characters and character sequences are used to address various components along the data path. It is wise to remember that over one-quarter of all possible character bit patterns have defined semantics that some component of the data path may interpret and act upon.

A variety of techniques have been invented to use non-control characters to represent byte values that would otherwise be interpreted as control characters. The KERMIT protocol, for example, uses the # character to "quote" control characters into the printable range @ to _ and, if needed, the & character to "quote" characters with the 0x80 bit set into their corresponding equivalents with the 0x80 bit clear (and there are rules for quoting the quote characters themselves). URIs, which typically use ISO-8859-1 or UTF-8 but are themselves encoded in ASCII, represent non-alphanumeric octets as %xx where "xx" is the hexadecimal representation of the character. The "HDLC-like framing" of PPP uses the } character (0x7d) to "quote" protocol-meaningful byte values and byte values that might be interpreted or intercepted by the data link into non-meaningful equivalents.

In the absence of a connection known to be transparent under all circumstances, it is prudent to prepare for non-transparency. Quoting arrangements are popular, but an alternative is to MIME base-64 encode all

9 You didn't think a Unicode code point and a Unicode character were the same thing, did you?

High 2 bits	Low 4 bits															
	0x_0	0x_1	0x_2	0x_3	0x_4	0x_5	0x_6	0x_7	0x_8	0x_9	0x_a	0x_b	0x_c	0x_d	0x_e	0x_f
0x0_	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
0x1_	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f
0x2_	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
0x3_	w	x	y	z	0	1	2	3	4	5	6	7	8	9	+	/

Figure 6. MIME base-64 encoding characters.

binary data. The MIME base-64 encoding organizes the bits to be transmitted in groups of six, rather than octets, and then represents the values in the range 0-63 resulting from this as characters in the ranges A-Z, a-z, 0-9, and + and / (and = as a null pad to a total encoding length a multiple of 4). This approach is illustrated in Figure 7; the characters used to represent each six-bit quantity are given in Figure 6.

While both quoting and base-64 encoding allows a byte stream to survive transport on a serial line intact, neither addresses byte ordering of larger constructs such as 16-bit or 32-bit integers. Common practice is to use "network byte order," sending multi-byte integers high order bytes first. Practice for non-integers (such as floating point values) is mixed.

When designing a protocol that might possibly ever be used over serial lines, think hard before sending "bytes." By definition, unless there are special arrangements in place, *the things transmitted on an asynchronous serial line are not bytes*. They are characters, even if they have 8 bits.

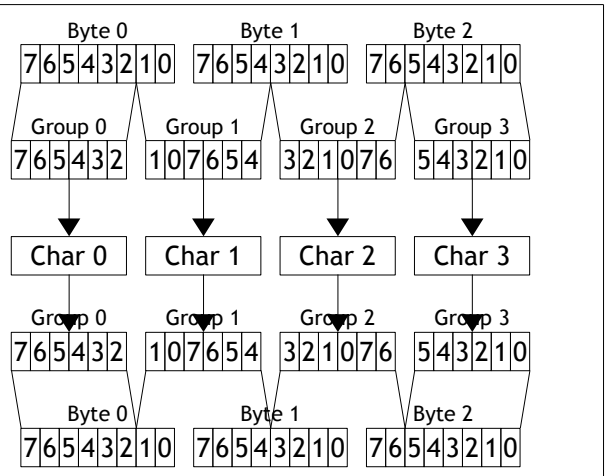

Figure 7. MIME base-64 encoding.

Data Structuring Facilities

The control characters presented in Figure 5 have semantics that interfere with transparent data communications. But those semantics are functions which can be used to structure and label data. These structuring and labeling facilities are typically underutilized. The HL7 data interchange protocol, for example, reserves the graphic characters | (vertical bar), ^ (circumflex), and & (ampersand) as field, component, and sub-component separators. The CSV file format reserves CR and LF both separately and together as record terminators, the comma character as the field separator, and assigns special semantics to the quote character.

So the word "coûter," property encoded[10] in standard ASCII as "cou←^ter" (where "←" is used to indicate the backspace "character composition" character), simply cannot appear in HL7; and "coöperate," for which the ASCII encoding includes the sequence 'oo←"p' has to be quoted in a CSV file even though it contains only alphabetic characters.

Such circumlocutions and failures result when control characters are ignored and graphic characters are assigned special semantics. These semantics, however, are typically already present in the control characters themselves. The FS, GS, RS, and US characters are in-

10 There is a perhaps subtle distinction here between a character code and a character encoding. The letter "û" does not have a *code* in ASCII: there is no 7-bit value assigned to represent this particular character. It does, however, have an *encoding*: a standard-defined way to represent the character. In ASCII and its related international standards and other countries' national standards, encodings of letters with diacriticals without assigned codes involves the backspace (BS) character. The meaning of the BS character is to combine the immediately preceding character with the immediately following diacritic. So, representing the BS character with the graphic ←, "û" is encoded as "u←^", "ç" is encoded as "c←,", and "ñ" is encoded as "n←~", among others.

tended for grouping and separating functions. The BEL character is intended for use as an urgency signal. The characters SOH, STX, and ETX structure a stream (or a portion of a stream: see ETB) into a header and data; the ACK and NAK characters indicate error-free and erroneous reception; the character FF structures a stream into pages; the character VT structures pages into sections; the CR and LF character pair, or the NL character, structures sections into lines; the character HT structures lines into areas; and so forth. Using control characters for device control and data structuring functions is neither kluge nor hack: it is their intended use. It is why they are there.

Keep the control characters in mind when you need to control devices and structure data. Your local character set weenie will appreciate it.

Further Information

This Application Note, the design files and source code associated with the devices described in this Application Note, and source code for applications to be used in conjunction with the devices described in this Application Note, are available at http://www.brianhetrick.com/an/AN-6.zip.

An excellent explanation of the electrical properties of TIA/EIA-232 communication is the Dallas Semiconductor technical note 83, *Fundamentals of RS-232 Serial Communication*, at http://www.fte.com/docs/max-232.pdf.

An introduction to TIA/EIA-485 networks and their implementation is Analog Devices' Application Note 960, *RS-485/RS-422 Circuit Implementation Guide*, available at http://www.analog.com/media/en/technical-documentation/application-notes/AN-960.pdf. A book-length discussion of the myriad details associated with deploying TIA/EIA-485 networks is the B+B SmartWorx *RS-422 and RS-485 Applications eBook - A Practical Guide to Using RS-422 and RS-485 Serial Interfaces*, available at http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/RS-422-and-RS-485-Applications-eBook.aspx.

A good overview of the LIN bus is National Instruments' *Introduction to the Local Interconnect Network (LIN) Bus* white paper, available at http://www.ni.com/white-paper/9733/en/. The LIN specification package itself is available at, among other places, http://www.cs-group.de/fileadmin/media/Documents/LIN_Specification_Package_2.2A.pdf. The 2.2A specification is the final one from the LIN Steering Group; current development is now in the hands of ISO, where LIN has become ISO 17987. Note LIN and ISO 17987 are in fact incompatible with one another.

The semantics of the C0 control characters are transcribed from ISO IR-001, a registry associated with the ISO-646 international character set, apparently by Mike Brown, at https://skew.org/iso-ir-001/. See also the Wikipedia article *C0 and C1 control codes* at https://en.wikipedia.org/wiki/C0_and_C1_control_codes.

The no-charge ExpressSCH and ExpressPCB software used to produce the schematic diagrams and PCB layouts of this Application Note is available from ExpressPCB at https://www.expresspcb.com/free-cad-software/.

The no-charge Visual C# Express 2010 software used to develop the Windows software in this Application Note is available from Microsoft at https://go.microsoft.com/?linkid=9709969. (The 2010 edition is the most recent Visual Studio edition that supports Windows XP as a development platform. The current Express and Community versions of Visual Studio, requiring Windows 7 or later, are available at https://www.visualstudio.com/downloads/download-visual-studio-vs.)

The no-charge Atmel Studio 7 software used to develop the Atmel microcontroller software in this Application Note is available from Atmel at http://www.atmel.com/tools/atmelstudio.aspx. (This software requires Windows 7 or later. It also needs to be the only Visual Studio 2015-based product installed on the development system. But what's another VM and Windows license between friends?)

The LibreOffice office suite used to create this Application Note itself is available at http://www.libreoffice.org.

This Application Note uses the Book Antiqua font for body text and the Source Sans Pro fonts for headings, headers, and footers. The Book Antiqua font can be licensed from Monotype at http://catalog.monotype.com/family/monotype/book-antiqua and is included in several of Microsoft's products. The Source Sans Pro font is available at https://adobe-fonts.github.io/source-sans-pro/.

Disclaimer

The author believes all content of this Application Note and any software and design it describes is a straightforward application of existing well-known concepts and techniques and is evident to anyone versed in the state of the art. Despite this, the author makes no claim or warranty of any kind that any item developed based on this Application Note, the software it describes, the designs it describes, or any portion of any or all, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country.

Copyright and Grant of License

This Application Note is Copyright © 2016 Brian Hetrick. It may be used and copied in accordance with the terms of the Creative Commons Attribution-Share-Alike 3.0 United States license. The text of this license is available at: http://creativecommons.org/-licenses/by-sa/3.0/us/.

The software presented or described in this Application Note or contained in the ZIP file accompanying this Application Note is Copyright © 2016 Brian Hetrick. It may be used and copied in accordance with the GNU Affero General Public License, version 3, or any later version at your option. The text of this license is available at: http://www.gnu.org/licenses/agpl-3.0.en.html.

The copyright holder grants you the above licenses because the copyright holder regards the social benefit arising from your compliance with the terms of the licenses as adequate consideration for the licenses themselves. You might not wish to comply with the terms of the licenses granted to you above. Contact the author to make arrangements for licensing under other terms.

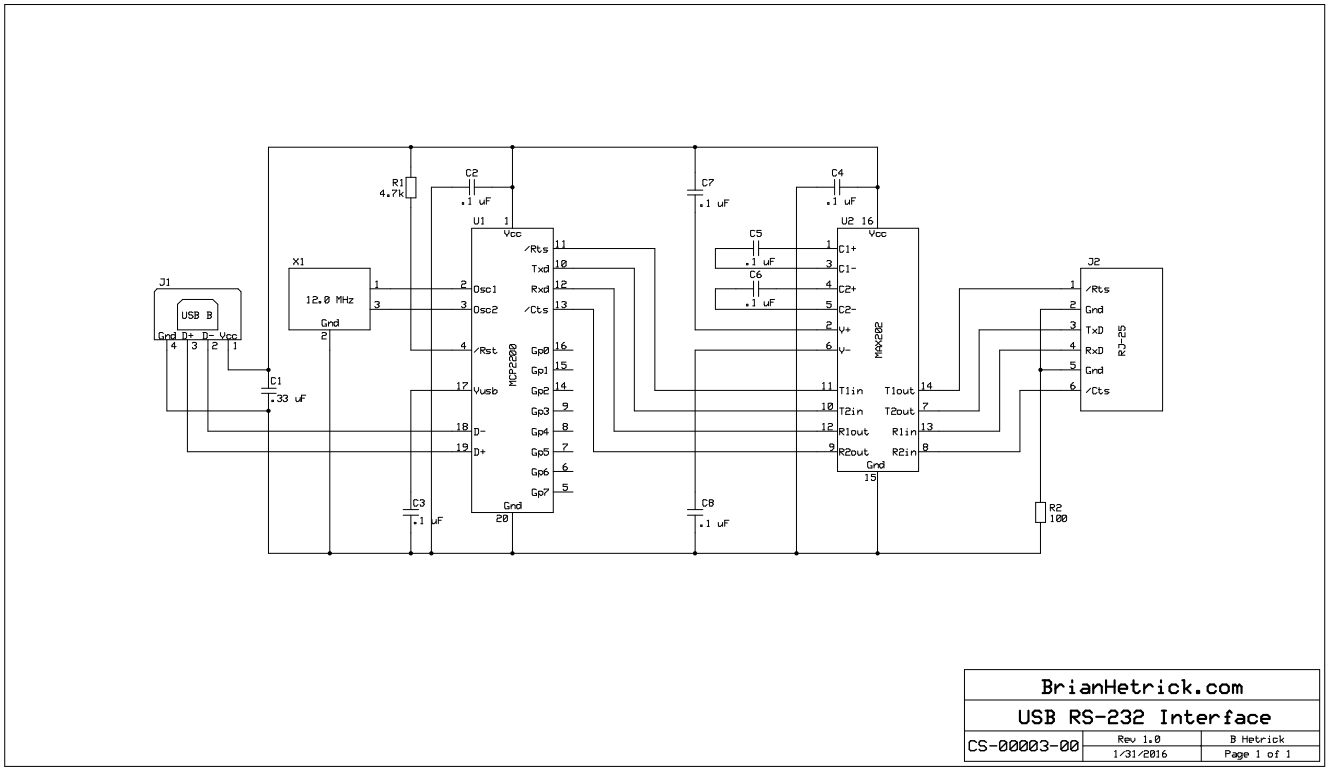

Figure A-1. Schematic circuit diagram for USB TIA/EIA-232 interface.

Appendix A: USB to TIA/EIA-232 point to point interface

As discussed in the Logic Levels section, there are a number of commercial USB adapters for TIA/EIA-232 serial lines with full modem control readily available. However, one that actually implements TIA/EIA-232 is typically $15-30. In hobby robotics and home control networking, full modem control is rarely required. A less expensive alternative intended for use on the workbench, between a host and a device, would be welcome. Additionally, USB serial adapter chips typically provide additional functionality (such as general purpose digital I/O lines) which can prove useful. This appendix shows the design of a device that provides bidirectional data and flow control, but not full modem control.

A convenient IC package that provides a 12 Mbps USB interface to a serial line with flow control is the Microchip MCP2200 USB 2.0 to UART Protocol Converter with GPIO. The MCP2200 part directly supports RTR/CTS hardware flow control. To translate the logic level serial signals used by the protocol converter to the TIA/EIA-232 levels needed on the wire, the Maxim MAX202 +5V supply, 2 transmitter 2 receiver RS-232 transceiver, is convenient. The +5V power for these parts is supplied by the USB connection. The MAX202 part includes charge pumps to create ±10V from the +5V USB power supply. Connectors and a handful of discrete parts complete the circuit. The design line speed of this device is up to 115.2k baud.

The device is intended as a fixed PC interface to a serial line; therefore neither space, USB connect/disconnect wear, nor power consumption are at any particular premium. Durability, however, is important, so the circuit is implemented as a PCB. The overall device power constraint is 500 mW, the amount of power provided that can be provided by the USB host when power negotiation is performed. The MCP2200 can be configured to perform this negotiation.

Modem control signals are neither generated nor used by the device. A six conductor RJ-25 connector is used for the TIA/EIA-232 level signals. This connector uses the Unitronics assignment (with DTR and DSR replaced by RTR and CTS, respectively) and is compatible with other RJ-25 TIA/EIA-232 signal connectors described in the appendices to this Application Note.

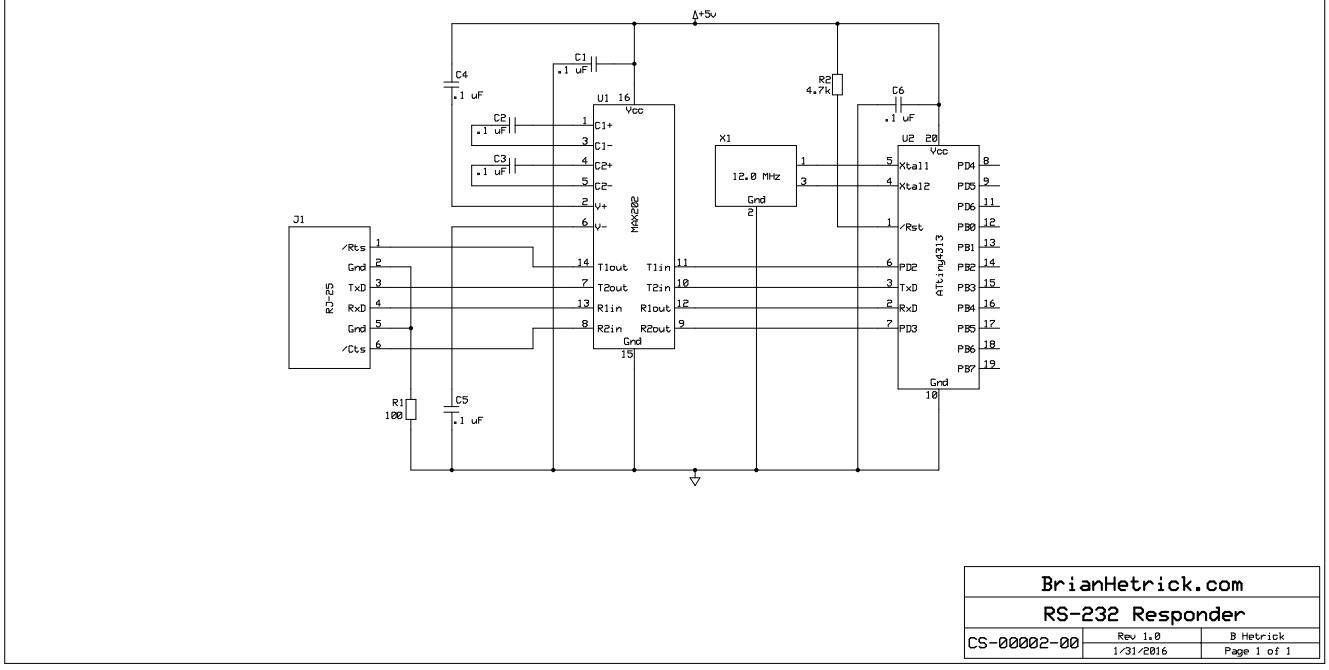

Figure A-2. Schematic circuit diagram for TIA/EIA-232 responder.

The schematic circuit diagram for this device is given in Figure A-1.

The MCP2200 includes 8 general purpose, bidirectional digital I/O lines. These might be used for discrete LED indicators, switches or buttons, or a low speed SPI or I^{2}C-like interface. They are unused in this circuit.

To test the circuit, known communication must be performed. This can be performed with two of these devices, or with one of these devices, a connector adapter, and an existing serial port. Source code for a .NET test driver exercising a pair of serial devices is included in the .ZIP file accompanying this Application Note. Alternately, a special purpose responder device can be used, and this approach is also explored here.

The responder device uses another Maxim MAX202 to convert TIA/EIA-232 signals to and from logic signals, and the built-in serial port peripheral of an Atmel ATtiny4313 microcontroller to enable programmatic response.

The schematic circuit diagram for the responder is given in Figure A-2. The source code for the .NET host testing driver is included in the .ZIP file accompanying this Application Note. The source code for the microcontroller firmware is included in the .ZIP file accompanying this Application Note.

The ATtiny4313 must be programmed with response firmware. The ATtiny4313 is programmed through an SPI arrangement, so an SPI interface must be acquired. Such interfaces can be purchased for about $20; once again the hobbyist response is to build one for less.

The Microchip MCP2210 serves as a convenient USB adapter for SPI communication. This SPI adapter will also be used to program microcontrollers in other circuits in these appendices to the Application Note, so it is made more elaborate than is needed for this partuclar application: it includes four LEDs for status indication and a pushbutton to indicate an operation should start. The schematic circuit diagram for this SPI interface is given in Figure A-3. A programming driver is also needed. The source code for a .NET host driver for Atmel device programming using the SPI interface is included in the .ZIP file accompanying this Application Note.

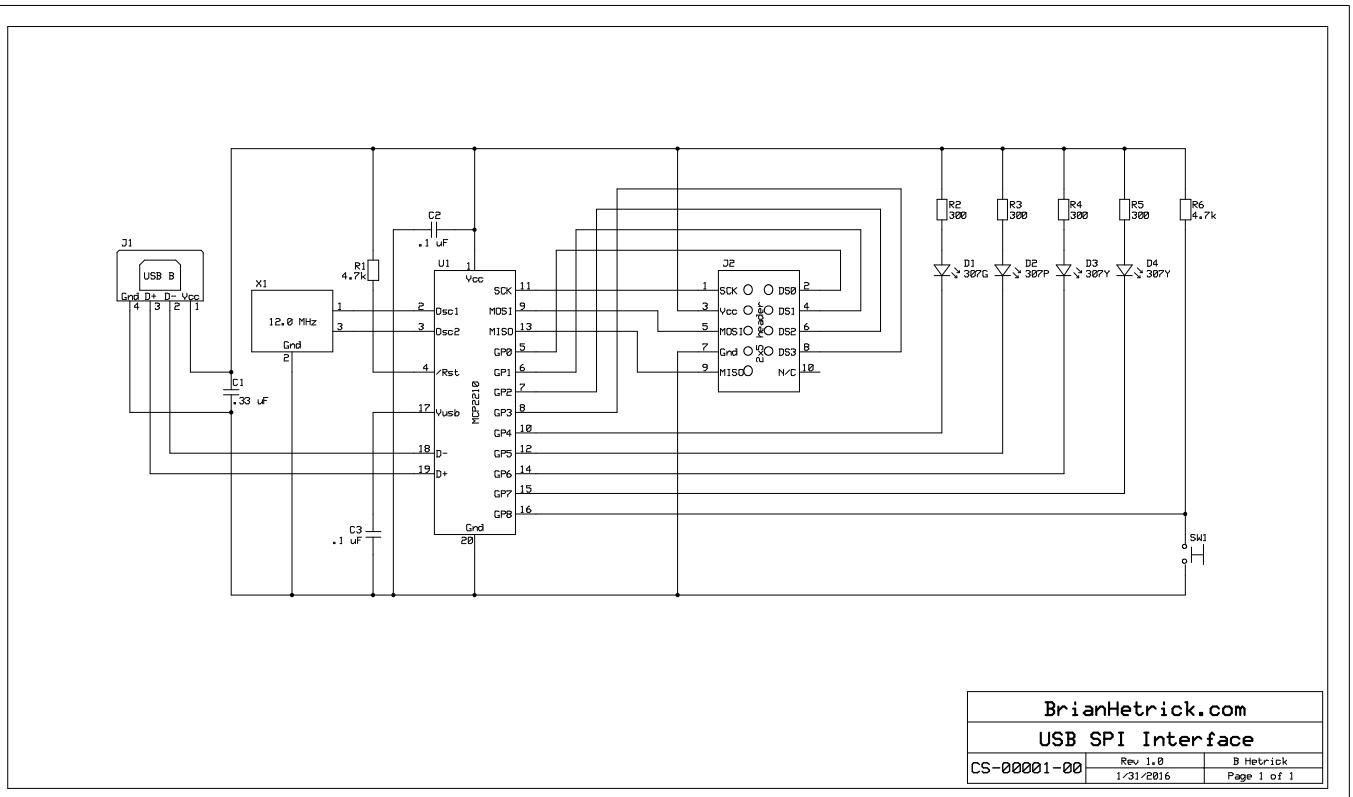

Figure A-3. Schematic circuit diagram for USB SPI interface.

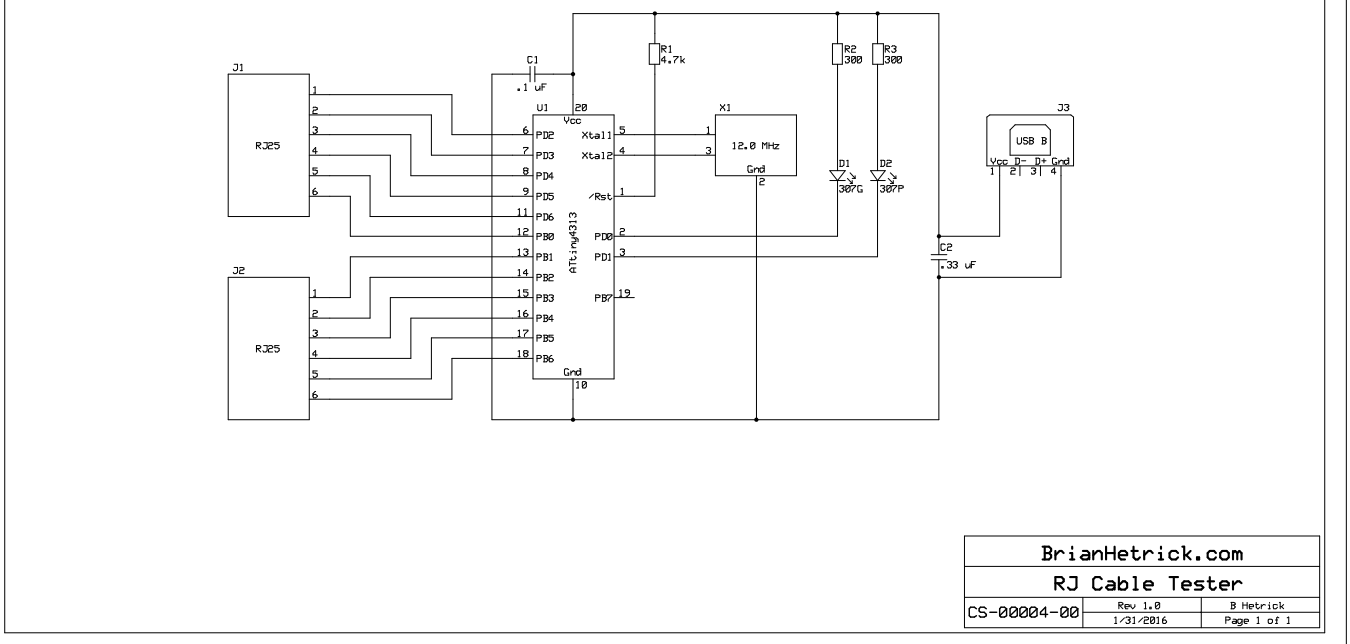

Figure A-4. Schematic circuit diagram for RJ cable tester.

Finally, it is convenient to be able to test the cables used for communications between devices. The devices described in these appendices to this Application Note use telephony RJ series connectors and cables for data transmission, and up to six lines are used. A simple cable tester that uses a handful of LEDs, resistors, and push button switches to indicate which pin of one connector is connected to which pin of a second connector is easily constructed.

Perhaps unsurprisingly in these digital times, however, the device cost is substantially reduced by using a microcontroller instead of a handful of resistors, LEDs, and push button switches; once again the Atmel ATtiny4313 serves well. The schematic circuit diagram for the cable tester is given in Figure A-4. The circuit is powered through a USB connector; this is for power only, the device is not a USB device. A wall wart and the venerable 7805 could easily be used for power instead. The source code for the microcontroller firmware is included in the .ZIP file accompanying this Application Note. The circuit and firmware indicate whether the cable is wired mirrored (what is typically required), straight through, or some other arrangement.

Thus, to validate the one circuit, three additional circuits and three software programs are needed. The SPI interface will be needed in other contexts, and is therefore implemented in a PCB. The cable tester circuit will be needed in other contexts also, so it too is implemented in a PCB. The responder circuit is needed only once, and so is never advanced past the breadboard stage. Cleverly, many parts from the evanescent responder circuit can be reused in the cable tester circuit or in other circuits described in these appendices to the Application Node.

Note both the MCP2200 and MCP2210 contain general-purpose storage accessible through the USB interface. The ATtiny4313 stores its program in reprogrammable flash memory. These properties may have implications for the use of these devices in a high-security environment.

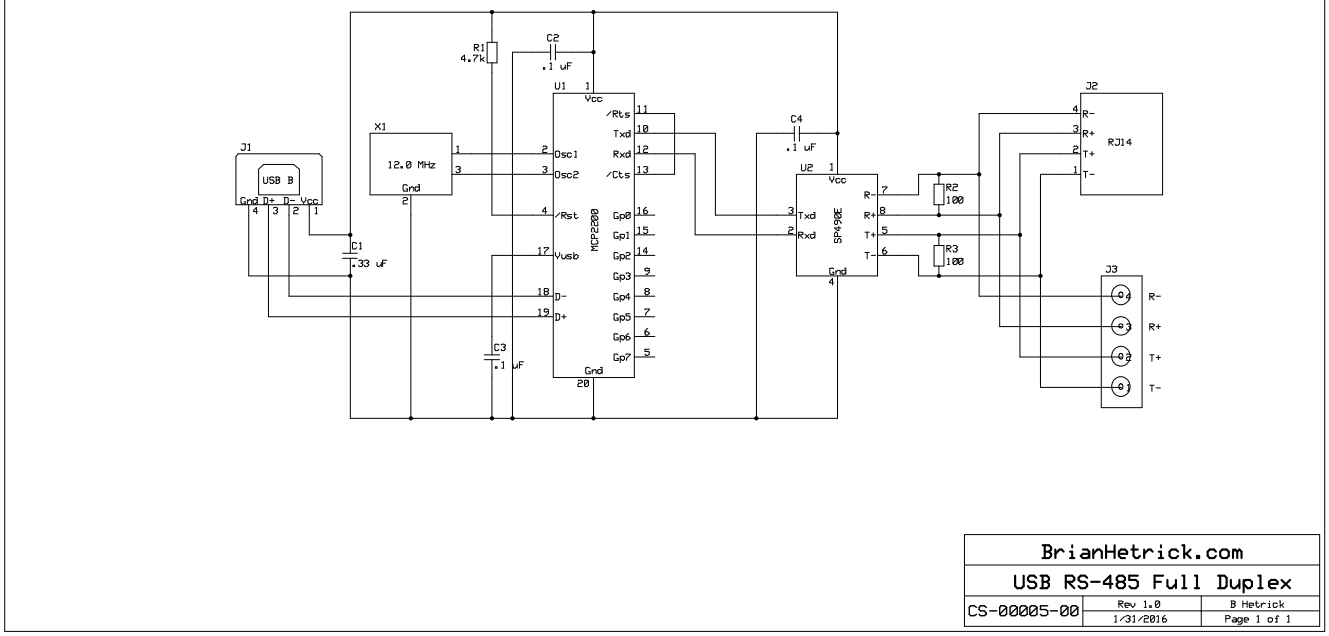

Figure B-1. Schematic circuit diagram for USB TIA/EIA-485 interface.

Appendix B: USB to bidirectional TIA/EIA-485 point to point interface

A variation on the peripheral of Appendix A is to use a bidirectional TIA/EIA-485 transceiver rather than a TIA/EIA-232 transceiver. Adapting the circuit to use the Exar SP490E bidirectional transceiver device results in a point-to-point data transceiver lacking hardware flow control but supporting data rates up to 1M baud. This then can provide a 1M baud serial link of up to about 100 m in length, or a 115.2k baud serial link up to approximately 1 km in length.

The device provides both a telephony RJ-14 (four conductor) connector and screw terminals for connection to the transmission line. Only one of these should be in use at a time. The RJ-14 telephone connector uses a symmetric, reversible pinout. Two of these devices can be connected by a 4-conductor telephony cable with RJ-14 plug ends, with the cable having a half-twist between the ends (equivalent to one RJ-14 plug being crimped on "upside-down").

The device is intended as a fixed PC interface to a serial line; therefore neither space, USB connect/disconnect wear, nor power consumption are at any particular premium. The overall device power constraint is 500 mW, the amount of power provided by the USB host after power negotiation is performed. The MCP2200 can be configured to perform this negotiation.

As this device is intended to be used as an endpoint in a fixed point-to-point link, the TIA/EIA-485 link is terminated on both the receiver and transmitter sides. The transmitter side is not "fail-safe" biased as the SP490E transmitter is always active. If this device is used to directly interface with the TIA/EIA-485 side of the modem of Appendix C, the application software must be prepared to deal with the quoting convention used by that device.

The schematic for this variation is given in Figure B-1. Testing arrangements are the obvious adaptations of those of the USB RS-232 interface of Appendix A.

Note the MCP2200 contains general-purpose storage accessible through the USB interface. This may have implications for the use of this device in a high-security environment.

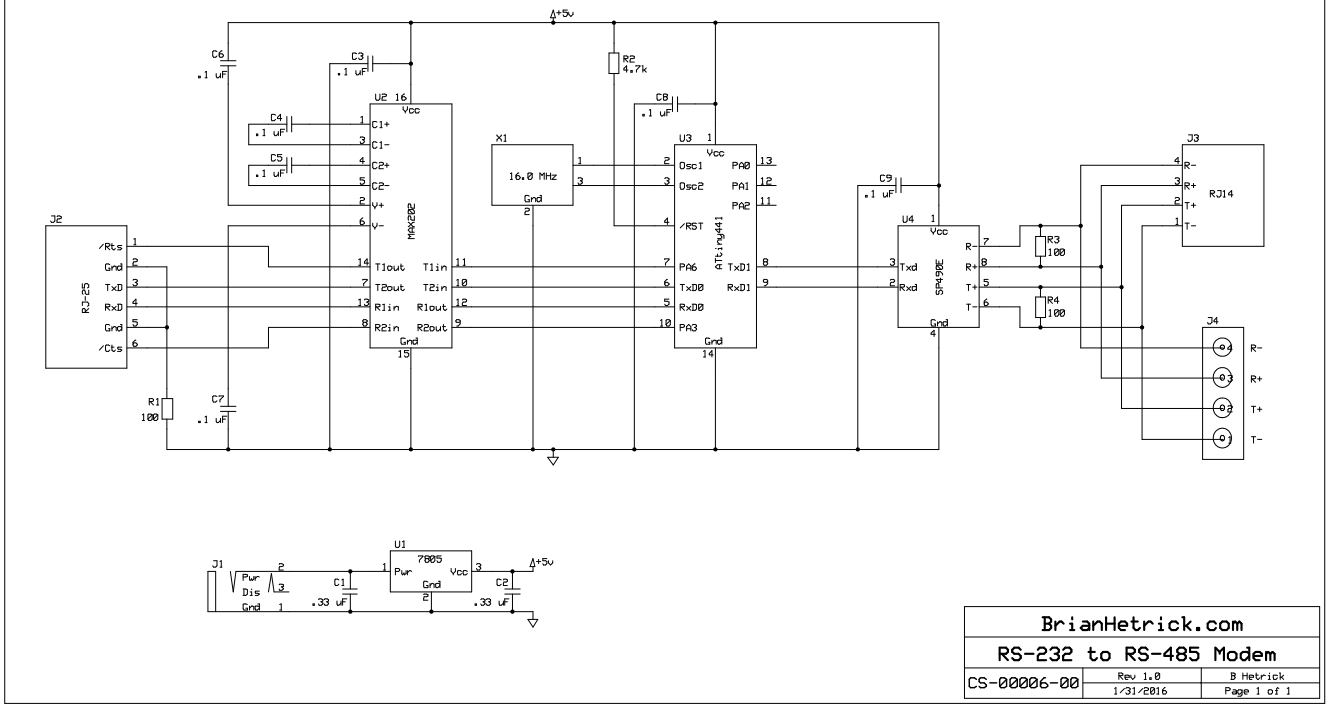

Figure C-1. Schematic circuit diagram for TIA/EIA-232 to/from TIA/EIA-485 modem.

Appendix C: TIA/EIA-232 to full duplex TIA/EIA-485 modem

In home networking, it is sometimes useful to have existing devices located an outbuilding participate in the home network. If the device can be fitted with a TIA/EIA-485 transceiver, all is well. If the device supports only TIA/EIA-232, however, some adjustments must be made.

A reasonably effective way to adapt a TIA/EIA-232 device to a TIA/EIA-485 long haul line is to use what effectively is a modem. This section shows the design of a full-duplex modem; when two instances of this design are paired, they use a bidirectional TIA/EIA-485 connection with software flow control to implement a link that appears to each participant as a transparent TIA/EIA-232 connection with hardware flow control. The TIA/EIA-485 segment can extend up to approximately 1 km.

The TIA/EIA-485 segment, lacking hardware flow control, must of necessity use software flow control. As the data stream may include XON and XOFF control characters, the device cannot simply inject and remove these. To provide a fully transparent medium, it is necessary to encode the characters on the ’232 connection for transport on the ’485 connection. Therefore this device adopts the PPP convention of representing some data characters as a pair of line characters: a prefix character of “}” (0x7d) followed by the data character with bit 0x20 inverted. This convention represents XOFF (0x11) as “}1”, XOFF (0x13) as “}3”, and } itself as “}]”. Only the three characters DC1 (XON), DC3 (XOFF), and } are encoded like this for transmission on the ’485 connection, but the quoting convention is applied to all characters received through the ’485 connection.

While this encoding discipline may reduce effective bandwidth by half (if the data stream consists entirely of the three affected characters), a data stream of random 8-bit characters is transmitted at 98.8% of line speed, and of printable and typical control characters at 100% of line speed.

However, a property of this design choice is that one of these characters emitted by one side will be received by the other side only after a three character time delay. This delay must be accounted for should software flow control be used on a data connection routed through a pair of these devices. Situations such

as this are why flow control disciplines typically quench the source when the input buffer is half to three-quarters full, rather than waiting for the input buffer to fill completely.

This design is based around a Maxim MAX202 +5V supply, 2 transmitter 2 receiver TIA/EIA-232 transceiver, an ATiny441 microcontroller, and an Exar SP490E TIA/EIA-485 full-duplex transceiver. The microcontroller is used to re-edge and re-clock the data transmission, convert from the hardware flow control protocol to the software flow control protocol, and convert to and from the quoted representation. A major criterion in the choice of microcontroller is that it have two UART devices. One UART device, along with two GPIO lines for flow control signals, are used for the TIA/EIA-232 line. The other UART is used for the TIA/EIA-485 line.

The device is powered through a 9V "wall wart" power supply. All baud rates are fixed at 115.2k.

As this device is intended to be used as an endpoint in a fixed point-to-point link, the TIA/EIA-485 link is terminated on both the receiver and transmitter sides. The transmitter side is not "fail-safe" biased as the SP490E transmitter is always active.

The device uses an RJ-25 connector for the TIA/EIA-232 level connection, and provides both an RJ-14 and screw terminal block for the TIA/EIA-485 line. These two connection arrangements connect to the same TIA/EIA-485 transceivers: only one should be in use at any one time. The device can be used directly with the TIA/EIA-232 device of Appendix A, the TIA/EIA-485 device of Appendix B, a second instance of this device, and the TIA/EIA-485 relay of Appendix D.

The schematic of the device is given in Figure C-1. The source code for the ATtiny441 firmware is included in the ZIP file containing this Application Note.

A pair of these devices can be tested with two instances of the TIA/EIA-232 interface of Appendix A, or with a TIA/EIA-232 interface of Appendix A combined with a TIA/EIA-485 interface of Appendix B. A .NET software test driver in included in the .ZIP file associated with this Application Note.

Note the ATtiny441 microcontroller stores its program in reprogrammable Flash memory. This may have implications for the use of this device in a high-security environment.

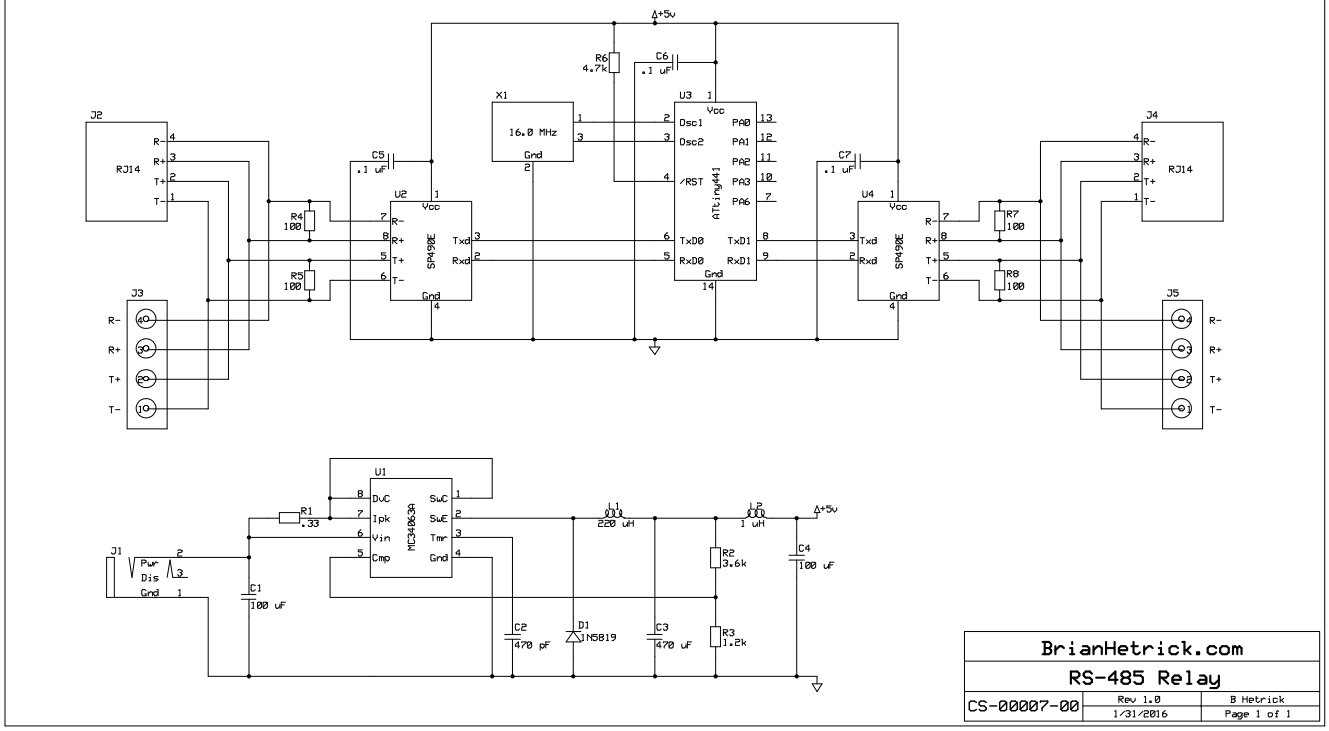

Figure D-1. Schematic circuit diagram for TIA/EIA-485 relay.

Appendix D: Full duplex TIA/EIA-485 relay

In home networking, it is sometimes useful to have devices located in an outbuilding further than 1 km from the network "home" participate in the home network. This might occur with a remote garage, barn, pump house, or water treatment arrangement. A reasonably effective way to provide such connectivity is through a series of relays 1 km apart. This section shows the design of a bidirectional TIA/EIA-485 relay running at 115.2k baud.

This design is based around two Exar SP490E TIA/EIA-485 full-duplex transceivers and an Atmel ATtiny441 microcontroller. The microcontroller is used to re-edge and re-clock the data transmission and to provide flow control. The ATtiny441 includes two UART devices, which are used for the two serial lines. The device provides termination resistors on all signal pairs. The device is powered through a DC power supply of at least 7.5V (a 9V "wall wart" or solar-recharged battery, for example).

This device is intended as a field device, potentially powered by a battery, rather than a device used in a structure. Power consumption is critical. For this reason, power is regulated with a switching buck regulator rather than the linear regulator or USB connections of previous devices described in these appendices. The device draws approximately 250 mW, or 6 Wh per day. This low power usage will enable a small solar panel to keep a 50 Wh battery fully charged, even in Seattle; or a quarterly maintenance visit to recharge a 1 kWh battery might be used.

The device introduces a 1-character delay in each direction: a character must be received fully before it can be retransmitted. The device also produces and consumes XON and XOFF characters as control flow characters, so an encoding or quoting scheme such as that used by the modem of Appendix C must be used to provide data transprency.

The schematic of the device is given in Figure D-1. The source code for the ATtiny441 firmware is given in a file included in the ZIP file containing this Application Note.

Note the ATtiny441 microcontroller stores its application program in reprogrammable Flash memory. This may have implications for the use of this device in a high-security environment.

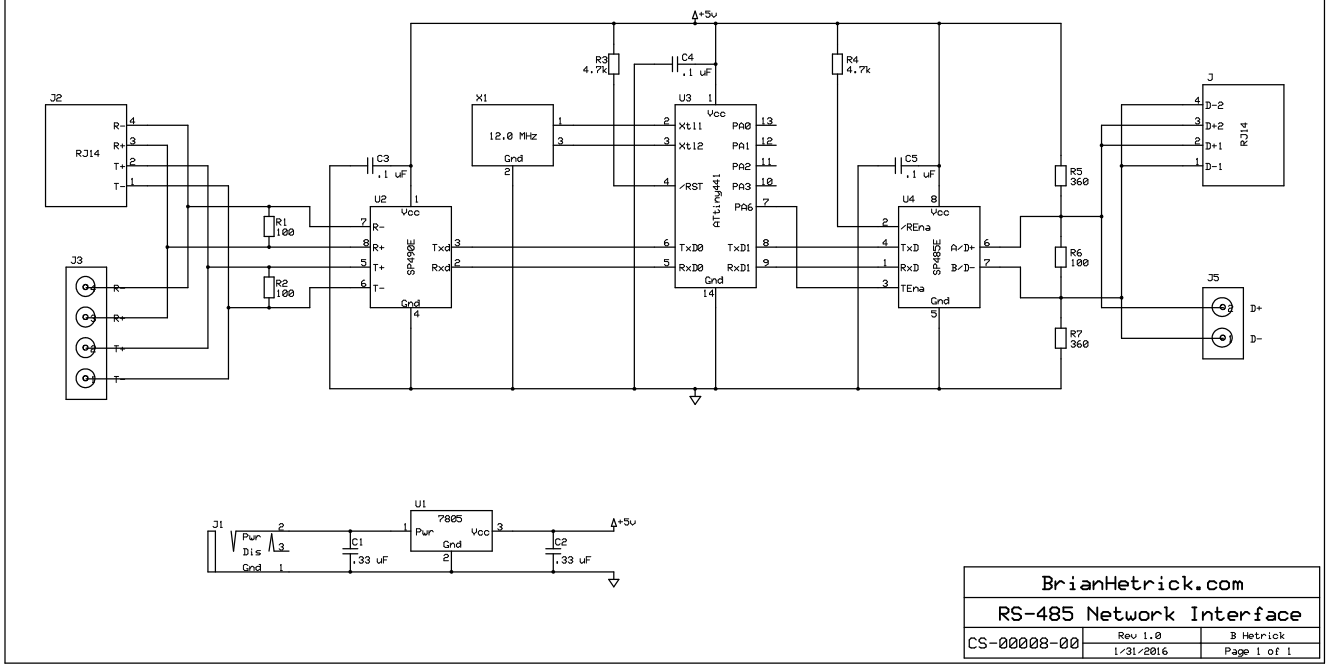

Figure E-1. Schematic circuit diagram for TIA/EIA-485 to TIA/EIA-485 half duplex network interface.

Appendix E: TIA/EIA-485 to TIA/EIA-485 binary packet network interface

A relatively common use of TIA/EIA-485 signals is in a half-duplex network with binary traffic used as a bus with a linear topology. Typically a master device or station addresses a slave device or station, either to give the slave device instructions or to query the slave device for information. After a delay during which the master device releases the line, the slave station responds. Slave stations otherwise must typically remain silent. In some networks, slave stations are permitted to transmit a break to inform the master station that a slave station requires service. All devices transmit on the same signal pair; the master/slave arrangement avoids collisions.

Typically network traffic consists of bytes (one of the few times this is true of a serial line) or nine-bit characters containing a high order flag bit and a low order byte value. In nine-bit networks, the high order flag bit is typically used only by the master, and only for an address byte indicating the slave device for which a message is intended. This permits slave devices to ignore traffic to and from other devices.

The circuit described in this appendix lets a PC serve as a master device on a half-duplex TIA/EIA-485 linear bus. The PC communicates with the circuit over a full duplex TIA/EIA-485 connection; the circuit accepts commands to transmit on the bus from, and returns received bus data to, the PC. The circuit can use 8- or 9-bit characters on the bus.

The connection from the PC to the circuit is point-to-point, so the RJ-14 cabling used in appendices B, C, and D is used for this connection. The connection from the circuit to the network, however, shares a single wire pair with all other stations. Typically network transceivers are placed directly on the network cables. The circuit contains a pair of terminal blocks for this use. However this can be inconvenient when a new station is to be added to an existing network.

To enable the transceiver to stand off from the network cable, an extension arrangement must be used. To maintain the linear topology, the bus must be brought to and then returned from the transceiver. To enable this, the device includes an RJ-14 connector and duplicates the network on the two sides. This permits the installation of a network tap on an existing two-wire network and extension to the location where the

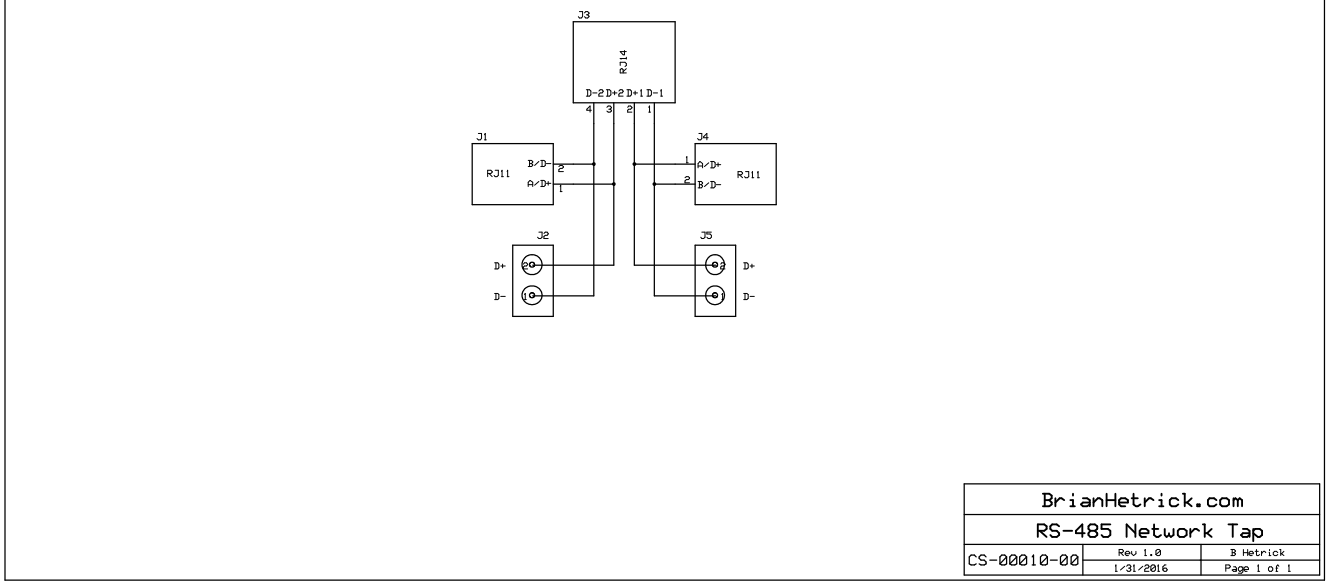

Figure E-2. Schematic circuit diagram for TIA/EIA-485 half duplex network tap.

transceiver is placed: two extensions, one to and one from the transceiver, are packaged into a single cable and single connector.

The circuit provides idle biasing for the network through a voltage divider. The resistor values used assume line termination at each end of the linear bus with 100 Ω resistors.

The schematic diagram for the device is given in Figure E-1. The schematic diagram for the network tap is given in Figure E-2. The command and data communications protocol between the PC and the device is described in Appendix G. Source code for a .NET PC interface, and source code for the ATtiny441 microcontroller are given in the .ZIP file associated with this Application Note.

Note the ATtiny441 microcontroller stores its application program in reprogrammable Flash memory. This may have implications for the use of this device in a high-security environment.

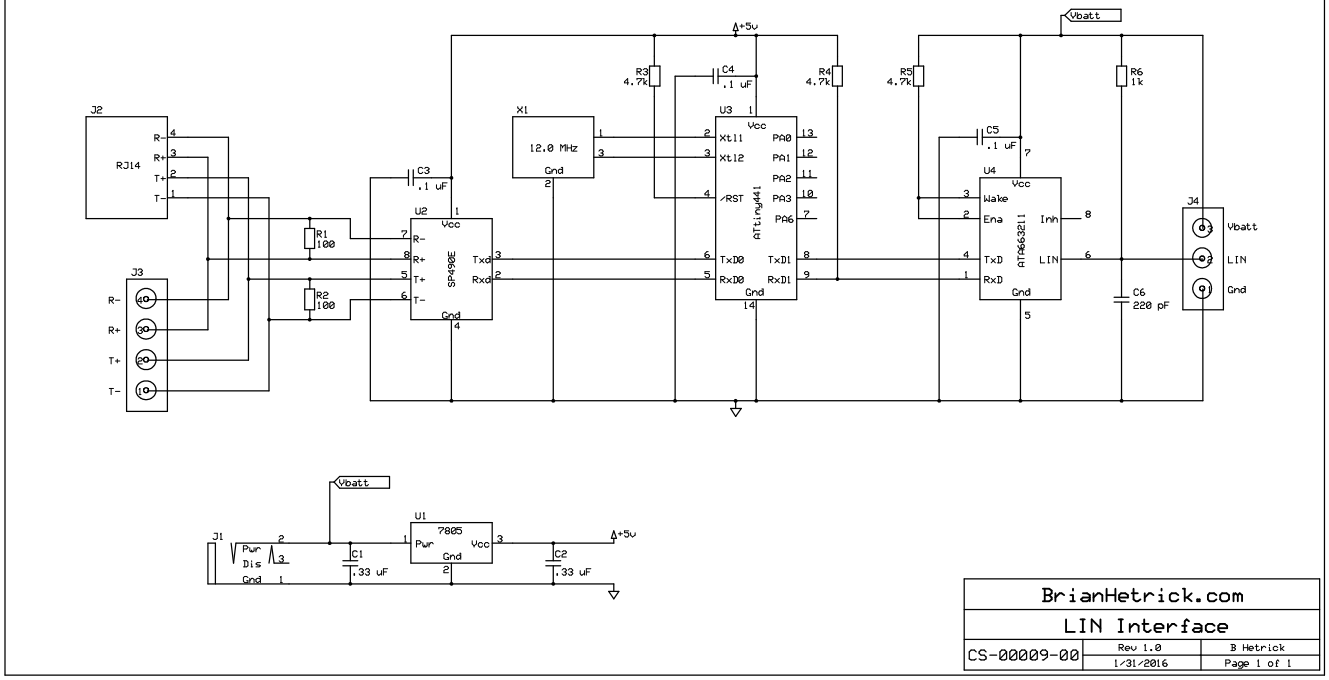

Figure F-1. Schematic circuit diagram for TIA/EIA-485 to LIN interface.

Appendix F: TIA/EIA-485 to LIN binary packet network interface

A final circuit in these appendices is a minor variation on that of Appendix E. With firmware changes and a transceiver change, the arrangement of Appendix E can be used to drive a LIN binary packet network. This appendix gives a circuit by which a PC can be a master node on a LIN. The PC communicates with the circuit with an 115.2k baud TIA/EIA-485 connection.

The LIN physical layer is a single wire: a default space condition is provided by a pull-up resistor. The pull-up is typically 1 kΩ at the master. Pull-ups may or may not be needed at the slaves, and if so are typically at least 10 kΩ. LIN transceivers typically are built essentially as a pair of FETs pulling output lines to ground so the digital and network sides can have independent voltages. Thus the received data line requires a pull-up resistor from the logic supply as it either floats high or is driven low.

In this circuit, the LIN is runs at 12 V; the LIN voltage is set by the supply (V_{batt}) to the pull-ups on the master transceiver. This may be replaced with a 5 V supply with no effect on bus performance, as long as all nodes use a consistent voltage.

It is possible to run low-power devices directly from the LIN, using a diode and capacitor to capture energy when the LIN bus is in its space state. One imagines this appeals to the same sort of designer that powers devices from the RTS line of TIA/EIA-232 connections. If this is done, the pull-up resistor on the LIN connection must be adjusted to accommodate the power draw of the connected devices.

The schematic diagram for the device is given in Figure F-1. The command and data communications protocol between the PC and the device is described in Appendix G. Source code for a .NET PC interface, and source code for the ATtiny441 microcontroller are given in the .ZIP file associated with this Application Note.

Note the ATtiny441 microcontroller stores its application program in reprogrammable Flash memory. This may have implications for the use of this device in a high-security environment.

Appendix G: Binary packet network interface communications protocol

TIA/EIA-485 networks and LINs are atypical in their use of a serial line: the line is not point to point, but rather shared among a number of network nodes; there is not a single node transmitting, but rather a number of stations; there is typically no explicit line turnaround request, but rather simply a pause in transmission; and characters are commonly larger than a byte. While microcontroller UARTs commonly support the 9-bit characters commonly used in TIA/EIA-485 networks, typical PC UARTs and operating systems do not. It should come as no surprise, then, that interfacing a PC to such a line requires some special arrangements.

The PC to TIA/EIA-485 binary packet interface of Appendix E and the PC to LIN binary packet interface of Appendix F describe microcontroller-based hardware devices which serves to translate between a full duplex 8-bit character serial data path at 115.2k baud and a half-duplex 8- or ('485 only) 9-bit character bus at 4.8k, 9.6k, or 19.2k baud. It receives commands from the PC and returns acknowledgments, status, received data, and line error condition reports to the PC. Communication with the PC is through a protocol described in this appendix.

The protocol uses only printable ASCII characters and the CR and LF characters, with XON and XOFF used for flow control; all of which typically behave well with character-oriented intermediaries such as relays, telnet connections, and operating systems. The protocol is packet oriented, where a packet is represented by a sequence of adjacent characters in the stream present on the full duplex path between the PC and the device.

The entire binary packet is packaged in radix-64 and terminated with CR and LF characters for transmission between the PC and the device.

Each packet is a byte stream. A packet consists of a one-byte command or response code (the message ID), followed by a one-byte sequence number, followed by a type-specific data payload, followed by a 16-bit CRC of all previous bytes. The CRC-16 is in network (big endian) byte order.

Packets from the PC to the device are termed commands. Packets from the device to the PC are termed responses. There need not be a command for there to be a response: the device will send unsolicited response packets when data is received from the TIA/EIA-485 network or when line errors occur.

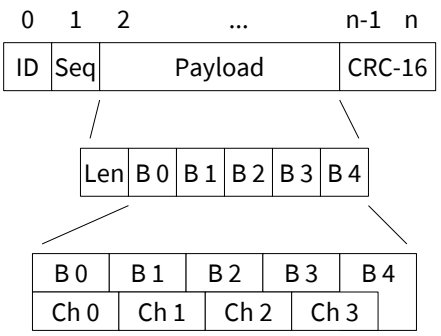

Figure G-1. Packet structure.

Sequence numbers in commands are unconstrained: they need not be distinct or have values in any particular sequence. However the command sequence number is included in responses to the command, so the application may wish to use distinguishable values. Sequence numbers in responses start at 1 at device power-on, increment by 1 for each response, and wrap from 255 to 1; response sequence number 0 is unused.

When the data payload contains a possibly variable number of characters, the data part of the packet consists of a one-byte character count followed by the characters. The characters are presented as a bit stream, from high order to low order in both characters and bytes, with byte boundaries occurring at each multiple of 8 bits. Excess bits in the last byte are ignored in commands and set to zero in responses. This structure is represented in Figure G-1. In the figure, four 9-bit characters are packed into five bytes; the length would therefore be 4. When the network character size is 8 bits, characters and bytes are synonymous and there are no excess bits.

The CRC-16 is computed in what Wikipedia calls reverse order[11]: bits contribute to the CRC in transmission order (low order bit first). The particular CRC polynomial used is that of CRC-16-ANSI. The CRC is computed as if through the C code:

```
unsigned int crc
   (const unsigned char * data,
    int data_length)
{
    int bitindex;
```

11 https://en.wikipedia.org/wiki/Cyclic_redundancy_check.

```
    unsigned int crc;

    crc = 0xffffu;
    while (data_length -- > 0)
    {
        crc ^= * data ++;
        for (bitindex = 0;
             bitindex < 8;
             bitindex ++)
        {
            crc = (crc >> 1) ^
               ((crc & 1u)
                    ? 0xa001u
                    : 0u);
        }
    }

    return crc ^ 0xffffu;
}
```

Note the initial value is all 1 bits, and the CRC is complemented at the end of the computation.

Commands

Messages sent from the PC to the device are commands.

Inquire Status

The inquire status command has ID 0 (0x00) and no payload. The device replies with a Status Report response.

Set Network Parameters

The set network parameters command has ID 1 (0x01). The data payload is: a one-byte role and character size indicator followed by a three-byte network baud rate in bits per second. The one-byte role and character size indicator has as its high-order bit an indicator of the station's role on the network: a 1 bit indicates the station is the master, a 0 bit indicates the station is a slave. The low order four bits of the role and character size indicator are the network character size in bits. Other bits are reserved and must be zero.

The TIA/EIA-485 network interface may (or may not) respond to an indication of role by imposing or not imposing biasing on the network. The circuit of Appendix E always imposes biasing regardless of role. The LIN interface may (or may not) respond to an indication of role by imposing or not imposing a pull-up on the network. The circuit of Appendix F always imposes a pull-up regardless of role. The LIN interface responds to an indication of role by prefacing transmissions with an attention break and a synchronization byte when the station is the network master. The TIA/EIA-485 network interface will accept network character sizes of 8 and 9 bits. The LIN interface will accept only a network character size of 8 bits.

The device replies with a Positive Acknowledgment or Negative Acknowledgment response.

Transmit Block

The transmit block command has ID 2 (0x02). The data payload is: a counted variable number of characters as described in the text. The number of characters to be transmitted must be between 1 and 32, inclusive. The device replies with a positive acknowledgment response upon receipt of the command and with a data transmit complete response upon transmission of the characters.

Responses

Messages sent from the device to the PC are responses.

Positive Acknowledgment

The positive acknowledgment response has ID 128 (0x80). The data payload is: a one-byte sequence number of the command being acknowledged.

Negative Acknowledgment

The negative acknowledgment response has ID 129 (0x81). The data payload is: a one byte sequence number of the command being rejected, if that can be determined, followed by a one byte rejection code. The rejection code is a bitwise OR of the following reason codes:

0x01: invalid radix-64 encoding

0x02: invalid CRC-16 value

0x04: invalid command ID

0x08: invalid role and character size indicator

0x10: invalid network speed

0x20: invalid data length.

Status Report

The status report response has ID 130 (0x82). The data payload is: a one-byte command sequence number of the inquire status command being responded to, followed by a one-byte protocol version number (cur-

rently 1), followed by a one-byte role and character size indicator, followed by a three-byte network baud rate in bits per second. The role and character size indicator and the network baud rate in bits per second are as in the set network parameters command.

Data Transmission Complete

The data transmission complete response has ID 131 (0x83). The data payload is: a one-byte command sequence number of the transmit block command for which the transmission has completed.

Data Received

The data received response has ID 132 (0x84). The data payload is: a counted variable number of characters as described in the text.

Because the bus speed is considerably slower than the line between the PC and the device, multiple character transmissions from devices on the bus are typically presented in a series of short data received responses each holding only a few characters. The PC software must combine these fragments and determine when the device on the bus has stopped transmitting.

Framing Error Received

The framing error received response has ID 133 (0x85). There is no data payload. This response is sent when a framing error is received on the bus. Multiple framing errors with no received data separating them may be combined into a single report.

Overrun

The overrun response has ID 134 (0x86). There is no data payload. This response is sent when data from the bus has been lost, typically because the device has been quenched through flow control. Multiple overruns with no received data separating them may be combined into a single report.

The device imposes a minimum of 1 character time between transmissions arising from different transmit block commands. This allows the host to implement a (simulated) network device. The host first transmits the (simulated) network device attention sequence, then the (simulated) network device response. The one-character delay permits other network devices interested in the (simulated) device response to set up for receiving the (simulated) device response. Simulated network devices may include clocks, sensors, effectors, and so forth.

At power on, the device initializes the network parameters to 9.6k baud and 9-bit characters (for the EIA/TIA-485 network interface) or 8-bit characters (for the LIN interface).