A Maxim Integrated Products Brand

73M2901CE
V.22 *bis* Single Chip Modem

APPLICATION NOTE

AN_2901CE_009 **JULY 2005**

Implementing Host Based HDLC Using Quasi-Synchronous Mode

Quasi-Synchronous Mode Basics

The Teridian 73M2901CE supports the use of a host-based software module for V.42 / MNP 2-4 Error correction Protocol and SDLC operation. These protocol operations require a special mode from the 73M2901CE called quasi-synchronous. This mode of operation is enabled via the ATY4 command.

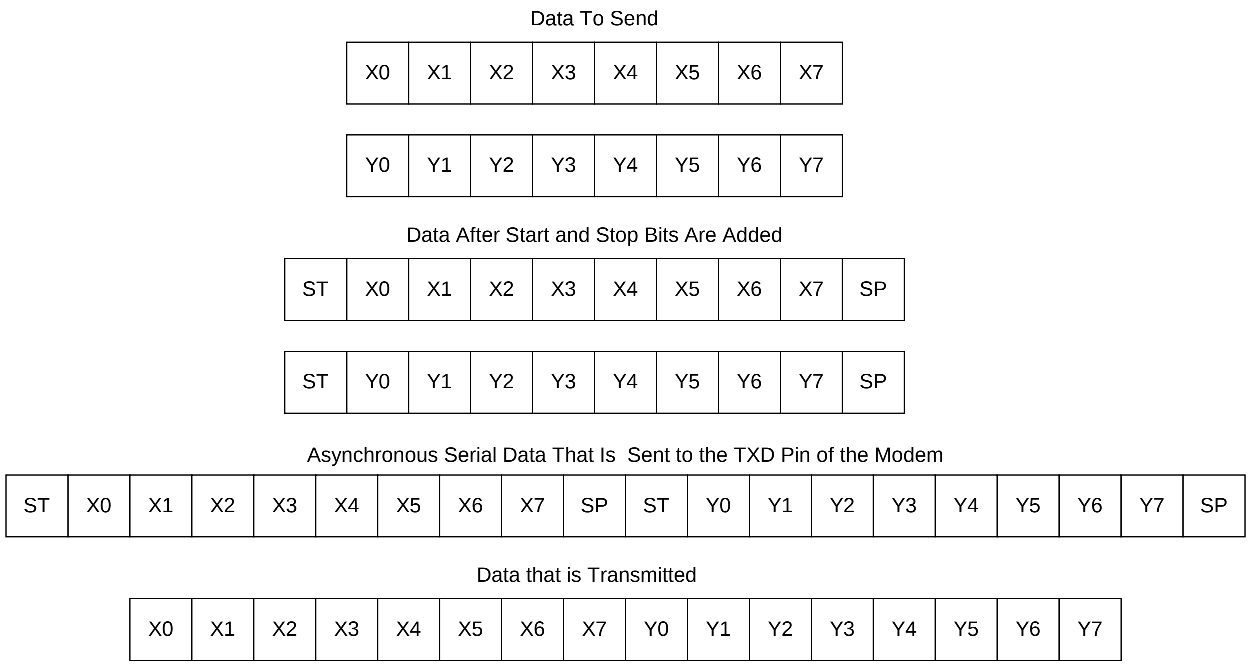
Quasi-synchronous operation (sometimes called the pseudo-synchronous) is a feature of the 73M2901CE modem to perform an asynchronous to synchronous conversion (and vice versa) to the data going over the DTE interface to and from the Host system. The following discussion describes the operation of this mode.

HDLC, SDLC, as well as V.42 and MNP Error control protocols, use a synchronous data format. The data is sent via 8 bit octets synchronized by the Transmit and Receive Clocks. A problem arises when sending data to and from the host system because most hosts do not have a synchronous connection to the modem data pump. These systems generally only have an asynchronous UART for sending and receiving serial data. A UART does not use the Transmit and Receive Clocks from the modem; instead it transfers data as characters using extra bits called Start and Stop bits to convey timing information. If the UART is not sending data, the Transmit pin is held in a high, or in a “Marking” state. If the UART wishes to transmit a byte (Octet) of data, a “Start Bit” is transmitted first. A Start Bit or low state is sent for one bit time before the first bit of data is transmitted. After the 8 bits of data are transmitted, the UART returns to the high, or “Marking” state for an additional bit time before more data can be transmitted. This trailing “Marking” time is called the “Stop Bit”. These extra bits added to the data stream represent 20% more data than what would normally be transmitted with the data bits only. Synchronous data, without a protocol, does not add overhead to the actual data.

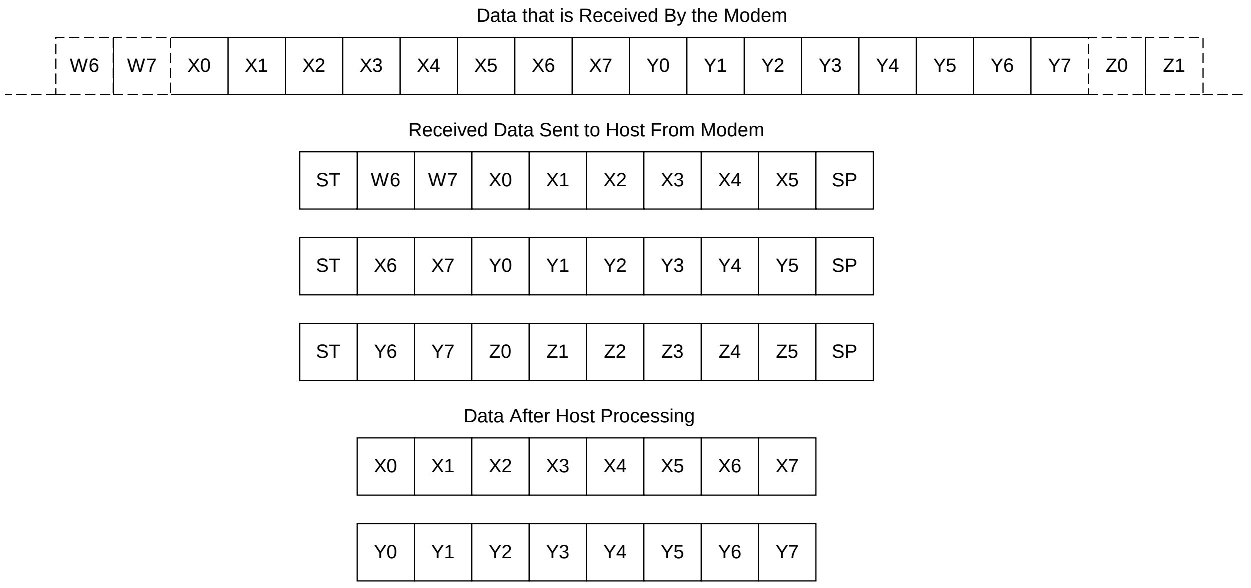
If the modem needs to send and receive synchronous data, there is a fundamental compatibility problem communicating with a UART interface that expects data framed by start and stop bits. This is where the quasi-synchronous mode becomes useful. In quasi-synchronous mode, after the initial V.22 or V.22 *bis* connection is established, the modem will automatically remove these extra start and stop bits from the transmitted data stream. This allows the Host system to use the more commonly found UART to connect to the modem, yet still send and receive synchronous data with the other modem. A visual representation of the data transfer across the UART to the modem is illustrated on the following page.

The sending process is easy to comprehend, but the receiving side is somewhat harder. The problem here is that the receiving modem no long can look for a start bit to tell where the octet data boundaries were in the transmitted data.

Quasi-Synchronous Process for Transmission


Data To Send
X0 X1 X2 X3 X4 X5 X6 X7
Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
Data After Start and Stop Bits Are Added
ST X0 X1 X2 X3 X4 X5 X6 X7 SP
ST Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 SP
Asynchronous Serial Data That Is Sent to the TXD Pin of the Modem
ST X0 X1 X2 X3 X4 X5 X6 X7 SP ST Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 SP
Data that is Transmitted
X0 X1 X2 X3 X4 X5 X6 X7 Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7

Quasi-Synchronous Process for Reception


Data that is Received By the Modem
W6 W7 X0 X1 X2 X3 X4 X5 X6 X7 Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Z0 Z1
Received Data Sent to Host From Modem
ST W6 W7 X0 X1 X2 X3 X4 X5 SP
ST X6 X7 Y0 Y1 Y2 Y3 Y4 Y5 SP
ST Y6 Y7 Z0 Z1 Z2 Z3 Z4 Z5 SP
Data After Host Processing
X0 X1 X2 X3 X4 X5 X6 X7
Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7

A quasi-synchronous modem receiving data needs to do the opposite actions of the transmitting channel by re-inserting the missing start and stop bits so the data can be send over the UART receive path. In the example for the receive path, the data is no longer aligned with the original data boundaries. But synchronous data has unique data patterns called “flags” (7E hex) that mark the boundaries of the HDLC (High-Level Data Link Control) frames and these are what the host-based HDLC protocol will be looking for when it processes the data. Once the flags have been identified, the rest of the data can be aligned to the original boundaries. This receive path example illustrates that the received data may no longer appear to be the same as the octets that were sent since the receive sync to async conversion process does not know or care where the beginnings of the octets were that were sent. This all looks a little strange at the serial interface, but the original data can easily be recognized by the protocol. This flag pattern is not allowed in the data fields and “zero insertion” is used after any sequences of 5 ones in any field except for flags so a “fake flag” cannot be sent. These extra zeros are removed after the flags are detected in the receiver. The extra zeros are not used in the CRC computation.

Teridian Quasi-Synchronous HDLC Software Module

HDLC or SDLC (IBM's Synchronous Data Link Control, from which HDLC is derived) are protocols that are used to send data reliably between two points. There are a number of other data link protocols, but most are similar in operation to HDLC. A key element of implementing this protocol is framing and de-framing the data. The 73M2901CE does not have hardware a HDLC packetizer but it can support this operation with a very small and efficient piece of host based software in combination with the "Quasi-Synchronous" operation mode of the 73M2901CE.

A HDLC frame is shown is Figure 1. All of the fields in between the opening Flag and the closing Flag are typically comprised of zero inserted data previously discussed. The Frame Check Sequence (FCS) block is a 16-bit CRC that is computed on the address, control, and data fields.

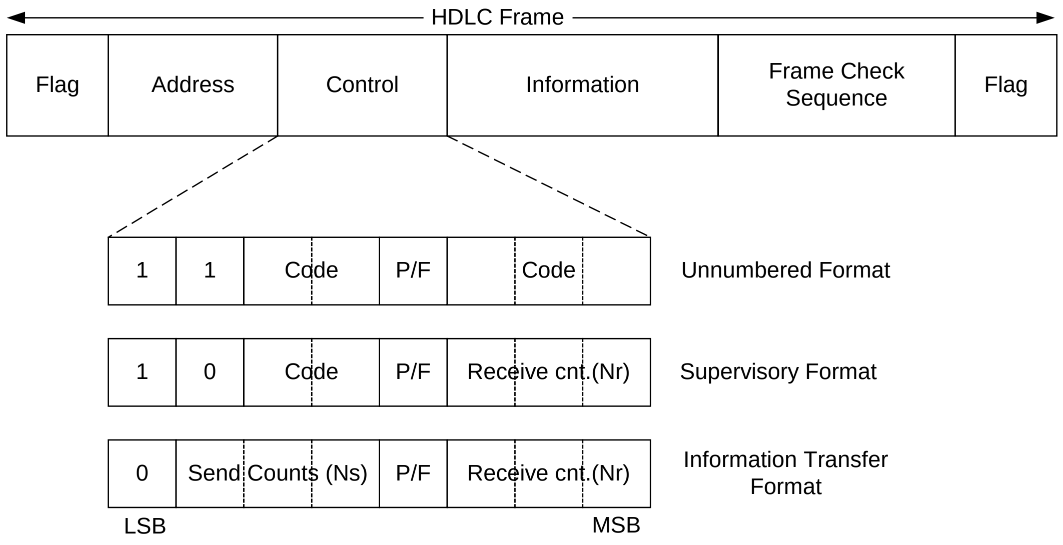


Figure 1

The host software described in this document is designed to perform the framing and de-framing function required by the HDLC protocol. The software makes no assumptions about the content of the combined Address, Control, and Data fields of the packet but simply treats all three fields as a payload. The content of the entire payload (Address, Control, and Data fields) is the responsibility of users application. The software described here will perform two functions, HDLC framing, and HDLC de-framing.

A few more points about flags should also be noted. One of the advantages of HDLC is that large blocks of data can be send with a minimum of overhead. The larger the payload relative to the overhead of Address, Control, and Flag fields, the more efficient the transfer. There are a few ways the flags can be sent to help reduce this overhead. The flag is sent as an 8-bit octet of 01111110 (7E hex). If an end flag and start flag are sent at the juncture of two frames, this now becomes 0111111001111110. Sometimes the end 0 of the end flag is combined with the start 0 of the next start flag to become 011111101111110, so this pattern must also be recognized. Sometimes the end flag of the first frame is also the start flag for the next frame, so only a single flag is sent between frames. All these occurrences must be detected. Also, when there is no data, flags may be sent to keep the link active.

How is the software used?
Very simply, when the host has a payload to send it is first sent to the framer to be packetized. The returned packet is then ready to be sent to the 73M2901CE via the asynchronous UART channel. When bytes are received from the 73M2901CE in the asynchronous UART channel they are sent to the de-framer function. When a complete de-framed packet is received the de-framer notifies the host and the application can retrieve the de-framed packet.

Remember that the data going to the modem from the host will not all be sent; 20% of the data is start and stop bits that will be stripped off. For this reason, the modem DTE-DCE interface should be run at a rate greater than the data over the line and flow control must be used so there are not gaps in the data that is transmitted. On the receiving side there will be 20% more bits sent to the host than received due to the added start and stop bits. This also requires that the DTE-DCE interface is able to send the extra data without running out of time.

SDLC framing:
The framing function performs zero insertion and CRC calculation on the entire payload and adds the opening and closing flags. This is all done with a single function call. Your application supplies the payload and the framing function will return a complete SDLC frame to pass on to your applications UART transmit function(s).

SDLC de-framing:
The de-framing function is also implemented with a single function call. The de-framer has more work to do than the framer because data bytes arriving from the 73M2901CE have no inherent alignment to the decoded bytes. That is to say, the modem is just demodulating bits and when it gets 8 bits they are framed between a start and stop bit and transmitted asynchronously to the host processor. Because of this the opening Flag could be received in 7 possible rotations across byte boundaries in addition to the 1 in 8 chance of being contained within a single byte. This is illustrated in the following Table 1.

Table 1: Eight Possible Bit Rotations of a Received Flag

Byte 0	Byte 1	
11111011	01111110	Flag in Byte 1
11111010	1111110x	Crosses Byte 0 to Byte 1
11111001	111110xx	Crosses Byte 0 to Byte 1
11110011	11110xxx	Crosses Byte 0 to Byte 1
11110111	1110xxxx	Crosses Byte 0 to Byte 1
11101111	110xxxxx	Crosses Byte 0 to Byte 1
11011111	10xxxxxx	Crosses Byte 0 to Byte 1
10111111	0xxxxxxx	Crosses Byte 0 to Byte 1

The de-framer must therefore first perform synchronization with the incoming bit stream. This is followed by removal of any inserted zeros and CRC calculation performed on the payload once the closing flag is detected. Notice that there is a zero whenever there are 5 consecutive ones. These inserted zeros will get removed after the flag detection is performed and before the address, control and data fields are processed.

This entire process is performed by making a single function call to the de-framer function on each byte received by the host applications Uart receive function. The de-framer will notify the application when a complete packet has been de-framed with its return value. A demo application has been provided to demonstrate the use of the framer de-framer functions.

SDLC Demo Application:
An example version of the quasi-synchronous code is available for a PC environment. This application demonstrates usage of the framing and de-framing functions. The program performs the following functions:

- Read in a message to be packetized using standard I/O.
- Process the message through the framer function.
- Display the complete framed packet in raw hex notation.
- Feed the framed packet back into the de-framer one byte at a time.
- Display the de-framer status on a byte-by-byte basis.
- Display the de-framed message in raw hex notation.
- Display the de-framed message in ASCII notation.

As implemented the program can work with payloads as large as 1024 bytes. This is only a limitation of the user defined max packet size and can be as large as required up to that of the 16 bit limitation of 65535 bytes. The program was compiled with Borland C++ 5.02 compiler for a 16-bit target. It will execute in a standard dos window. The program uses file I/O redirection for the input and display. The RunDemo.BAT program will execute Hdlc.exe demo application in a dos window using the file TDKTEST.txt as the input data and generate the HDLCOUT.txt file:

Hdlc.exe <TDKTEST.txt>HDLCOUT.txt

This means the scanf() function will read input from the TDKTEST.txt file and all printf() output will go to the HDLCOUT.txt file. Alternately you could execute the program with out redirection and input data at the keyboard terminating your input with <ctrl>Z. In this case all output would be to the screen.

It is important to note that the function main and all of the source code in the main.c file are only for demonstration purposes. This file is not intended for use in your application. It is only intended to show how to make the function calls to the framer and the de-framer. The code that would be integrated into your application is less than 1K words of ROM, and the RAM required is approximated by the formula (3.2*MaxPacketSize + 48) bytes. The following modules are needed to support the framer and de-framer.

• HDLC.H	; Header file for SDLC processing, max packet size is defined here.
• CCITTCRC.C;	; This module contains the ROM table for table based CRC calculation.
• SENDHDLC.C	; This module contains the Framer functions.
• RECVHDLC.C	; This module contains the De-Framer functions.

The source code is well commented to help the end user understand its function.

The following files are provided in addition but will not be required for integration into your application:

• HDLC.IDE	; Borland 5.02 project file used to build HDLC.EXE.
• MAIN.C	; Demonstrates the calls to the Framer & De-Framer functions.
• HDLC.EXE	; The demo application, uses STDIN I/O.
• RunDemo.BAT	; Use this batch file to execute the demo in a DOS window.
• TDKTEST.txt	; This plain ASCII text file is used for input to the demo application.

Framer De-Framer Functions:

Framer Syntax:
#include “hdlc.h”
TDK_INT16U TDK_build_hdlc_frame(TDK_INT8U *tx_in_buf, TDK_INT16U tx_in_length, TDK_INT8U *tx_out_buf)

Description:
This function is used to frame an HDLC packet prior to sending it to the 73M2901CE. The function is called once for each HDLC frame to be created. The arguments are defined as follows:

- tx_in_buf – pointer to buffer of 8bit unsigned values where the payload to be framed is located.
- tx_in_length –16 bit unsigned variable that indicates the number of bytes placed in tx_in_buf.
- tx_out_buf – pointer to buffer of 8bit unsigned values where the framed packet is deposited.

Return Value:

- This function returns the length of the resulting HDLC frame in bytes.

De-Framer Syntax:
#include “hdlc.h”
TDK_INT16U TDK_hdlc_recv_processor(TDK_INT8U inchar, TDK_INT16U *rx_cnt, TDK_INT8U *rx_ptr)

Description:
This function is used to de-frame a received HDLC packet. The function is called once for each byte of received data the arguments are defined as follows:

- inchar – an 8 bit unsigned value representing received byte of raw data from the 73M2901CE.
- rx_cnt – pointer to a 16 bit unsigned variable that indicates the number of bytes de-framed.
- rx_ptr- pointer to buffer of 8bit unsigned values where the de-framer will deposit the payload.

Return Value:

This function returns three possible values:

- TDK_HDLC_PROC – 0x0000 - Input byte accepted, still processing.
- TDK_HDLC_RCVD – 0x0001 - Input byte accepted, De-Framed packet ready.
- TDK_HDLC_BAD_CRC – 0x0002 - This packet aborted, init new search.