World Transactions on Engineering and Technology Education
Vol.2, No.2, 2003

Java program implementation of HDLC protocol for serial data communication

Faisal Kaleem, Kang K. Yen & Amaury A. Caballero

Florida International University
Miami, United States of America

ABSTRACT: This paper presents a Graphic User Interface (GUI)-based implementation of serial data communication using High-level Data Link Control (HDLC) protocol. This Java-based program has been implemented for educating students in order to provide them with insights into how HDLC can be utilised to perform a data transfer on a serial basis. The program can be used to analyse HDLC data packets being transferred from one machine to another. This practical approach is totally different from the theoretical approach, whereby students only read the book and try to understand the different layers of a protocol without actually using or implementing the protocol.

INTRODUCTION

Because of the possibility of transmission errors and the need to regulate the arrival rate of data to the receiver, synchronisation and interfacing techniques alone are not sufficient for error-free data transmission. As such, it is necessary to impose a layer of control in each communicating device to provide functions such as flow control, error detection and error control. This layer of control is known as a data link control protocol and the transmission medium between systems is called the data link. For effective serial data communication, two directly connected stations require: frame synchronisation; flow control; error control; addressing and link management [1]. A data link protocol must satisfy all of the above requirements. Several data link protocols have been proposed and are currently in use [2][3]. The following discussion will concentrate on the High-level Data Link Control (HDLC) protocol [4].

A developed program in Java to show insight into how HDLC can be used to perform data transfer on a serial basis and to analyse HDLC data packets being transferred from one machine to another is presented in this article. It demonstrates the exchange of I-Frames, S-Frames and U-Frames between two end stations, during initialisation, data exchange and disconnection, as well as how bit stuffing is performed. In an end station, the transmission and reception of data packets are displayed in separate windows in a binary format, as well as in a hexadecimal format. Along with strings of zeros and ones, readable statements are also inserted to describe the purpose of a stream of data that is being sent or received. In order to see all of the frames involved in the data transfer of a single byte of character, the user can utilise the *live mode* where, as soon as a key is pressed, the program constructs all of the necessary packets and sends it to the other end. This practical approach is different from that of the theoretical approach, which only shows the different layers of a protocol without actually using or implementing it.

HDLC PROTOCOL

HDLC, a layer 2 protocol of the OSI model, is a popular ISO-standard, bit-oriented, link layer protocol [5][6]. This protocol was derived from Synchronous Data Link Control (SDLC) and specifies an encapsulation method of data on synchronous serial data links. The HDLC protocol defines three communicating station types: primary, secondary and combined; two link configurations: unbalanced and balanced; and three modes of data transfer: Normal Response Mode (NRM), Asynchronous Balanced Mode (ABM), and Asynchronous Response Mode (ARM) [6].

The HDLC operation consists of the exchange of I-Frames, S-Frames and U-Frames between the two end stations. There are three phases involved in the HDLC operation [7][8]. These are described below:

- *Data Link Initialisation*: Initialisation may be requested by any station using any of the six set modes of the U-Frame. This set mode initialisation serves three purposes, namely:
  - It signals the other side that initialisation is requested.
  - It specifies the mode of operation (NRM, ABM and ARM).
  - It specifies whether the control byte uses a 3-bit or 7-bit sequence number. If the receiving station accepts this request, then it acknowledges this by transmitting an Unnumbered Acknowledgement (UA) frame to the transmitting station and establishing a logical connection between the end stations. If the request is rejected, then a Disconnect Mode (DM) frame is sent.
- *Data Exchange*: After the logical link is established between the two stations, both sides may begin to send user data using I-Frames, starting with sequence number 0.

The N(S) and N(R) are numbered sequentially using modulo 8 or 128 depending on whether 3- or 7-bit sequencing is used. N(S) contains the current frame number to be sent by a station, while N(R) contains the next frame that a receiver expects from the other station during a data transfer.

Sometimes S-frames are also used for flow and error control. The Receive Ready frame (RR) in S-Frame acknowledges the last frame received by indicating the next expected I-Frame, while Receive Not Ready (RNR) not only acknowledges the received I-Frame, but also informs the transmitting station to halt the transmission momentarily. In the meantime, the entity that receives the RNR may poll the other station to find out whether it is ready to receive the next frame by transmitting an RR frame with a P bit and the expected frame sequence number. If the receiver is still not ready, it again issues the RNR frame; otherwise, it issues an RR frame indicating its willingness to accept the next frame and the data exchange resumes again.

- *Disconnecting the Link*: Any HDLC station can initiate a disconnect mode by sending a DISC (disconnect) frame. The receiver responds with a UA frame. At this time, the link is logically disconnected and any outstanding I-Frame may be lost.

IMPLEMENTATION

HDLCDemo is a program developed in Java to simulate serial data communication between two end stations using HDLC protocol. The program also utilises the powerful communication Application Programming Interface (API) from Sun MicroSystems to perform serial communications between two stations connected through serial ports via a null modem [9]. The steps needed to executive HDLCDemo, with regard to installing the communication API, are as follows:

- Download and install the API for the operating system from the Sun MicroSystems Website;
- Unzip the file javacomm20-win32.zip in the C: drive and JDK is installed in C:\jdk1.4;
- Copy win32com.dll to the <JDK>\bin directory;
- Copy comm..jar to the <JDK>\lib directory;
- Copy javax.comm.properties to the <JDK>\lib directory;
- Add comm..jar to the classpath;
- Reboot the system.

Some of the features of HDLCDemo will work without any hardware set-up, but to use the two-way communications feature, a *null modem cable* is required. The cable can either be looped back between the serial ports on a single machine or used to connect the serial ports on two different machines.

All of the Java files for HDLCDemo reside in a single directory. This makes the program very easy to build. In order to compile HDLCDemo, one requires a single command.

PROGRAM DESCRIPTION

Figure 1 depicts the HDLC Demo window. The main window contains a configuration panel that lets the user select the serial port parameters. These parameters must be selected before the user opens the port for active transfer. The program automatically identifies any available serial port on the system. The user can then select any port from the list. The open port and closed port buttons are used to open and close the selected serial port. The send text command is used to send any text the user has typed to the other station on a semi-live basis.

The main window, excluding the configuration panel, can be divided into two main sections: the TX section for transmission and an RX section to receive data packets. Both the TX and RX

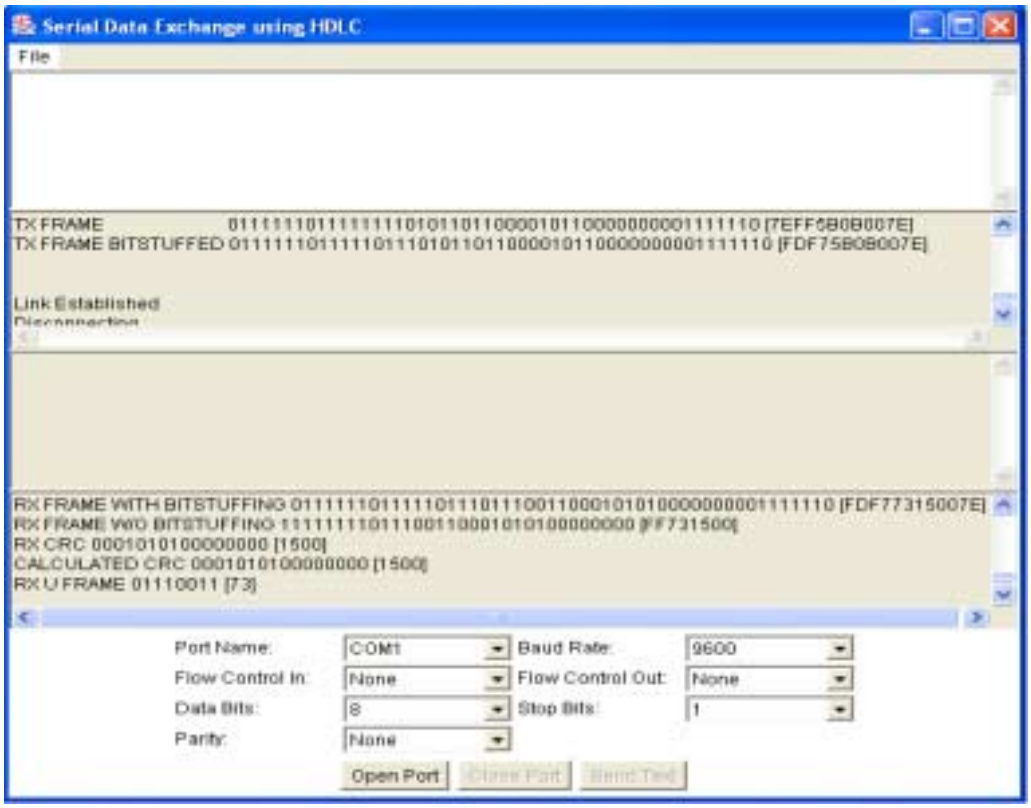

Figure 1: HDLC demo window and link establishment.

sections have two subsections: one to display the original information in a readable format and the other to show the HDLC operations by displaying frames in a binary and a hexadecimal format, as depicted in Figure 1. The only editable window that lets the user type and transmit the text is the TX window. The rest of the windows are non-editable.

The program also uses some of the function keys (F1-F5) to perform certain actions. None of the function keys work if the port is not opened. The following explains what the function keys do:

F1: Initialises the link and sets the mode of operation to Asynchronous Balanced Mode (SABM).
F2: Initialises the link and sets the mode of operation to Normal Response Mode (SNRM).
F3: Initialises the link and sets the mode of operation to Asynchronous Response Mode (SARM).
F4: Toggles between the live and semi-live modes. In the live mode, whatever the user is typing on the transmitting side is received straightaway by the receiver. The user does not have to press the *send* key to transmit the text to the other side. In the semi-live mode, after typing the text, the user has to press the *send text* button to send the data to the other side.
F5: Toggles between busy and non-busy conditions. Pressing F5 indicates to the transmission side that the receiver is not ready to receive any new frame by sending an RNR command. Pressing F5 again generates the RR frame, indicating its willingness to receive the next frame in the sequence.

The following steps establish the communication and data exchange:

- Execute the program on both stations (assuming that the above-mentioned requirements have been fulfilled).
- Select the serial parameters and press the open port button on the configuration panel on both stations. Data exchange cannot be performed by just opening the port; doing so will generate an error message.
- Initiate a data transfer from any station by pressing the function keys (F1-F3). Press F1 key to establish SABM 3 bit sequence mode; this will initialise all the sequence counters and indicates that the link between the two stations has been established (see Figure 1 again).
- Select the mode of transfer, live or semi-live, by pressing the F4 key. As described above, the live mode transmits the data as the user types, while the semi-live mode requires the user to press the *send text* button to transmit the information.
- Assuming live mode, start typing; the construction of the packets in binary and hexadecimal format in both the TX and the RX windows of the transmitting station and the receiving station, respectively, can be seen. This is shown in Figures 2 and 3.

It should be noted that if the link is established and the system detects inactivity for 10 seconds on any station, it sends an RR frame. The program does this three times and, after that, it assumes some problem in the link and sends a DISC command indicating a logical disconnection. At this time, both systems close their ports.

The following examples are some of the snapshots of raw data that are transferred from one side to the other.

Initialisation TX Side:
Initialisation Using SABM mode
U Control Byte 01011011 [5B]
TX CRC 0000101100000000 [0B00]
TX FRAME
01111110111111110101101100001011000000000001111110 [7EFF5B0B007E]
TX FRAME BITSTUFFED
01111110111111011101011011000010110000000000011111110 [FDF75B0B007E]

Frame received at RX Side:
RX FRAME WITH BITSTUFFING
01111110111111011101011011000010110000000000011111110 [FDF75B0B007E]
RX FRAME W/O BITSTUFFING
11111111010110110000101100000000 [FF5B0B00]
RX CRC 0000101100000000 [0B00]
CALCULATED CRC 0000101100000000 [0B00]

JAVA CLASSES IMPLEMENTED

The developed program, HDLCDemo, is comprised of four main Java classes, as listed below:

- HDLCDemo: This class executes the main program and is responsible for the Graphical User Interface (GUI).
- Serial Connection: a comprehensive class that is responsible for setting the serial parameters, opening the port and transmitting and receiving serial data wrapped in HDLC frame.
- CRC16: This class calculates the 16-bit CCITT-CRC-based check sum for error identification and error correction.
- HDLC: This is the main class that is responsible for controlling all HDLC-based operations involving initialisation, frame construction, frame reconstruction at the receiving end, maintaining the N(S) and N(R) sequence numbers, bit stuffing and other things necessary to the HDLC protocols.

CONCLUSIONS

HDLCDemo has been developed as a Java-based program that simulates the HDLC operations, excepting the functions of frame rejection and recovery process. It is believed that this implementation provides an environment to visualise the insight about how HDLC operates. The display of the binary and the hexadecimal-based frame in the window is a powerful feature, which can be used to demonstrate the operations of the HDLC frames. In its current form, the HDLC class contains all of the necessary methods for possible extensions.

REFERENCES

1. Stallings, W., *Data and Computer Communications.* Englewood Cliffs: Prentice-Hall (2000).
2. Held, G., *Understanding Data Communications.* New York: John Wiley & Sons (2000).
3. Forouzan, B.A., *TCP/IP, Protocol Suite.* New York: McGraw-Hill (2000).

4. Bux, W., Kuemmerle, K. and Truong, H.L., HDLC performance: comparison of normal response mode and asynchronous balanced mode of operation. *Proc. NTC Conf. Rec. National Conf. on Telecommunications in a New Decade*, 1, 1-15 (2000).
5. Bochmann, G.V. and Chung, R.J., Formalized specification of HDLC classes of procedures. *Proc. Conf. Rec. of National Telecommunication Conf.*, 1, 03A.2.1-03A.2.11 (1977).
6. Sczittnick, M. and Uhl, T., Aggregation of systems with HDLC protocol. *European Trans. on Telecommunications and Related Technologies*, 4, **1**, 107-112 (1993).
7. Bryant, S.F. and Johnson, K.E., Protocol simulation using SDL tools. *Proc. 6th IEE Inter. Conf. on Software Engng. for Telecommunication Switching Systems*, 259, 52-56 (1986).
8. Goyal, R., Lai, S., Jain, R. and Durresi, A., Laboratories for data communications and computer networks. *Proc. 28th Annual Frontiers in Educ. Conf.*, 3, 1113-1118 (1998).
9. Sun MicroSystems, Java Communication API, www.sun.com

Figure 2: Sending data from one station.

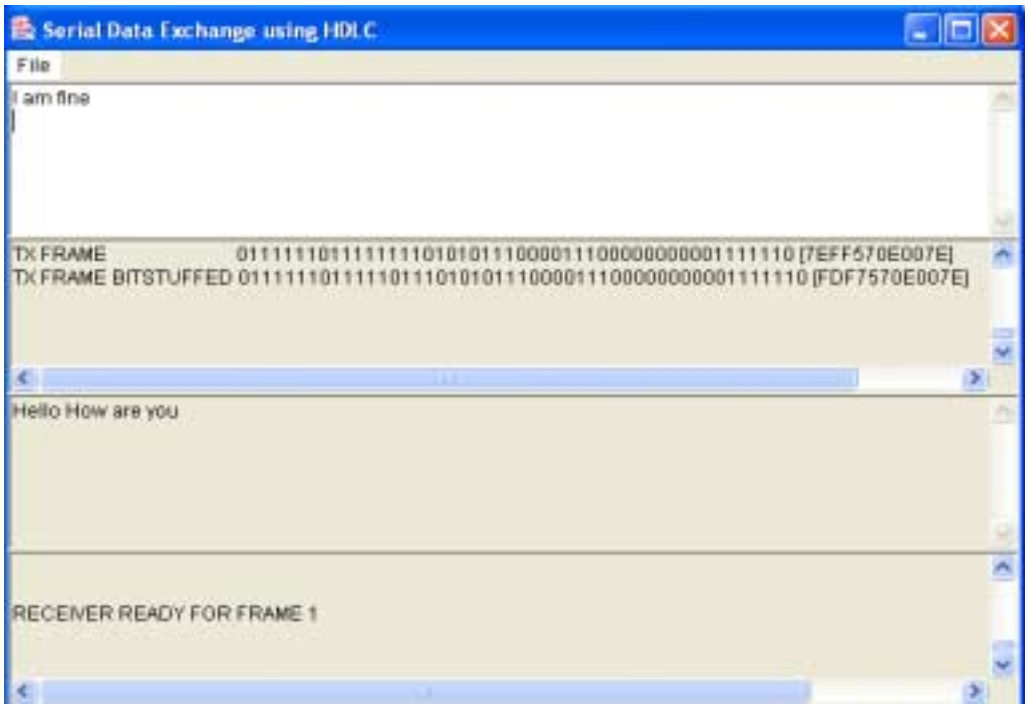

Figure 3: Receiving data at another station.