SX-Stack: Internet Protocols Implementation

SCENIX

Application Note 35

May 8, 2000

1.0 INTRODUCTION

A complete Internet protocol stack demo has been implemented, called the SX-Stack. It runs on a demo board that comes with the evaluation kit (EPAK-TCP/PPP01-xx). The User's Guide (literature number SXL-UG01-xx) included in the evaluation kit gives detailed instructions on how to set up and run the demo board connected to a PC.

The SX-Stack demo program combines five different demonstrations:

- UDP Demo
- iSX Web Server Demo—responds to requests to view web pages stored in an EEPROM.
- eSX E-mail Appliance Demo—sends and receives e-mail.
- Java Sprinkler Demo—not discussed in this manual.
- A/D Converter Demo—not discussed in this manual.

The demo is selected by commenting out various assembler directives in the source code. At the beginning of the source code, there are three areas which you may need to change before compiling the program:

- Target SX—select the processor type. For the demo board, the line which says "SX48_52" must not be commented out.
- Assembler Used—select the assembler. For the Parallax SX-Key assembler, the line which says "SX_Key" must not be commented out. If you are using the SASM assembler, this line must be commented out. (SASM is the default.)
- Options—select the demo to be run by commenting out the defines for the other demos. These demos are mutually exclusive (e.g., you can't be sending and receiving e-mail simultaneously). The defines are listed in Table 1-1.

Table 1-1. Defines For Running Demos

Demos	Required Defines
UDP	UDP, UDPDEMO
iSX Web Server	TCP, HTTPDEMO
eSX E-Mail Appliance (sending)	TCP, SMTPDEMO
eSX E-Mail Appliance (receiving)	TCP, POP3DEMO, POP3DEBUG

2.0 PROGRAM STRUCTURE

Table 2-1 describes the organization of the SX-Stack demo program. Mutually exclusive sections of code are placed together, to accommodate the page structure of program memory. The POP3 and UART Virtual Peripheral code is broken into several chunks, to fit it into available program memory. Most programs for the SX communications controllers are not nearly so complex as this one, so they won't usually require breaking up the code of a Virtual Peripheral.

Because the only physical layer interface is a UART Virtual Peripheral, the ISR code is simply the ISR of the UART VP.

Table 2-1. Structure of the SX-Stack Demo Program

Section	Description
defines	Used to specify the target SX, assembler used, and options, as described in Section 1.0.
constant declarations	For all modules.
global variable declarations	For all modules.
jump to ISR	When an interrupt occurs, execution begins at this location in program memory (address = 0). The instruction at this location then jumps to the ISR.
jump table	Jumps to the entry points for all Virtual Peripheral API calls.
UART Virtual Peripheral	Code for `GetByte`, `SendByte`, and `SerialInit`.
PPP Virtual Peripheral	Code for `PPPOpen`, `PPPRxData`, `PPPInit`, `PPPSendConfReq`, `PPPStartIPPacket`, `PPPSendTermReq`, `PPPSendPacket`, `PPPXxFCSInit`, `PPPCheckFCS`, `PPPFCSData`, `PPPReceive`, `PPPClosePacket`, `PPPClose`, `PPPSendCodeReject`, `PPPSendConfAck`, `PPPSendConfRej`, `PPPSendPartialPacket`, and `ModemConnect`.
POP3 Virtual Peripheral	Code for `AppInit`.
IP Virtual Peripheral	Code for `IPStartPacket`, `IPCheckSum`, `IPRxHeader`, and `IPReceivePacket`.
UDP Virtual Peripheral	Code for `UDPStartPacket` and `UDPRxHeader`.
TCP Virtual Peripheral	Code for `TCPCompare32`, `TCPPassiveOpen`, `TCPActiveOpen`, `TCPClose`, `TCPRxHeader`, `TCPSendEmptyHeader`, `TCPSendHeader`, `TCPClosePacket`, `TCPProcessPacket`, `TCPTransmit`, `TCPSendSyn`, `TCPSendAck`, `TCPSendFin`, `TCPSendReset`, `TCPInitChecksum`, `TCPChecksum`, `TCPTxByte`, `TCPAddRCV_NXT`, and `TCPAddSND_NXT`.
POP3 Virtual Peripheral	Code for `get_tens`, `get_hundreds`, and `times_ten`.
EEPROM File System Virtual Peripheral	Code for `E2Start`, `E2Stop`, `E2WriteToRead`, `E2Write`, `E2ReadAck`, `E2ReadNotAck`, `E2DelaySCLLow`, `E2DelaySCLHigh`, `E2WriteStart`, `E2WriteData`, `E2WriteComplete`, `E2ReadStart`, `E2ReadData`, `E2ReadComplete`, `E2OpenFile`, `E2CloseFile`, and `E2ReadFile`.
HTTPimplementation of calls required by TCP Virtual Peripheral API	Code for `AppInit`, `AppBytesToSend`, `AppPacketOK`, `AppPacketBad`, `AppBytesAvailable`, `AppTxByte`, `AppRxByte`, `AppNak`, and `AppAck`.
SMTPimplementation of calls required by TCP Virtual Peripheral API	Code for `AppPacketOK`, `AppInit`, `AppBytesToSend`, `AppBytesAvailable`, `AppTxByte`, `AppRxByte`, `AppNak`, `AppAck`, and `AppPacketBad`. The `AppPacketOK` code contains the packet transmitted in response to a ping.
POP3 implementation of calls required by TCP Virtual Peripheral API	Code for `AppPacketOK`, `AppRxByte`, `AppBytesToSend`, `AppBytesAvailable`, and `AppTxByte`.

Table 2-1. Structure of the SX-Stack Demo Program (Continued)

Section	Description
Application Code	Includes the Main Loop.
UART Virtual Peripheral	Code for PhyRxByte, PhyTxByte, and PhyRxTest.
POP3 Virtual Peripheral	Code for AppNak, AppAck, and AppPacketBad.
ISR	ISR code for the UART Virtual Peripheral.

3.0 PROGRAM OPERATION

The operation of the SX-Stack demo program is very similar for each of the demos. Figure 3-1 is a state diagram for the operation of the demo program.

One of the first calls made by the demo program is to PPPOpen, to open up a PPP connection to the host computer. If PPPOpen fails, the program continues to loop on PPPOpen until a connection is made.

The iSX web server calls AppInit before entering the main loop, and again on every pass through the loop. The eSX e-mail appliance in POP3 mode only calls AppInit once, before entering the main loop. In SMTP mode, it only calls AppInit when it receives an ICMP packet, which it assumes to be a ping packet. (ICMP packets are used for diagnostic messages, such as ping packets. ICMP packets can be distinguished from TCP and UDP packets in the protocol number field of the IP header.)

If a TCP packet is received, TCPProcessPacket is called. This routine implements the TCP state machine. TCPProcessPacket may return to the main loop or call other routines, depending on the type of the TCP packet. For example, if a SYN packet is received, TCPProcessPacket updates its state machine and returns to the main loop. If an ACK packet is received, AppAck is called to inform the application that an acknowledgement of a previous data transmission has occurred. If the packet is data, AppBytesAvailable is called to warn the application it is about to receive some number of bytes. AppRxData is called once for each byte received.

After the packet is received, AppPacketOK or AppPacketBad are called, depending on whether the packet passed its frame check sequence (FCS). The application doesn't know whether the data it is receiving is valid until after the FCS has been verified, so it must not make any changes that cannot be undone until after AppPacketOK is called. If AppPacketBad is called, the received data is discarded and the application prepares to receive another transmission attempt.

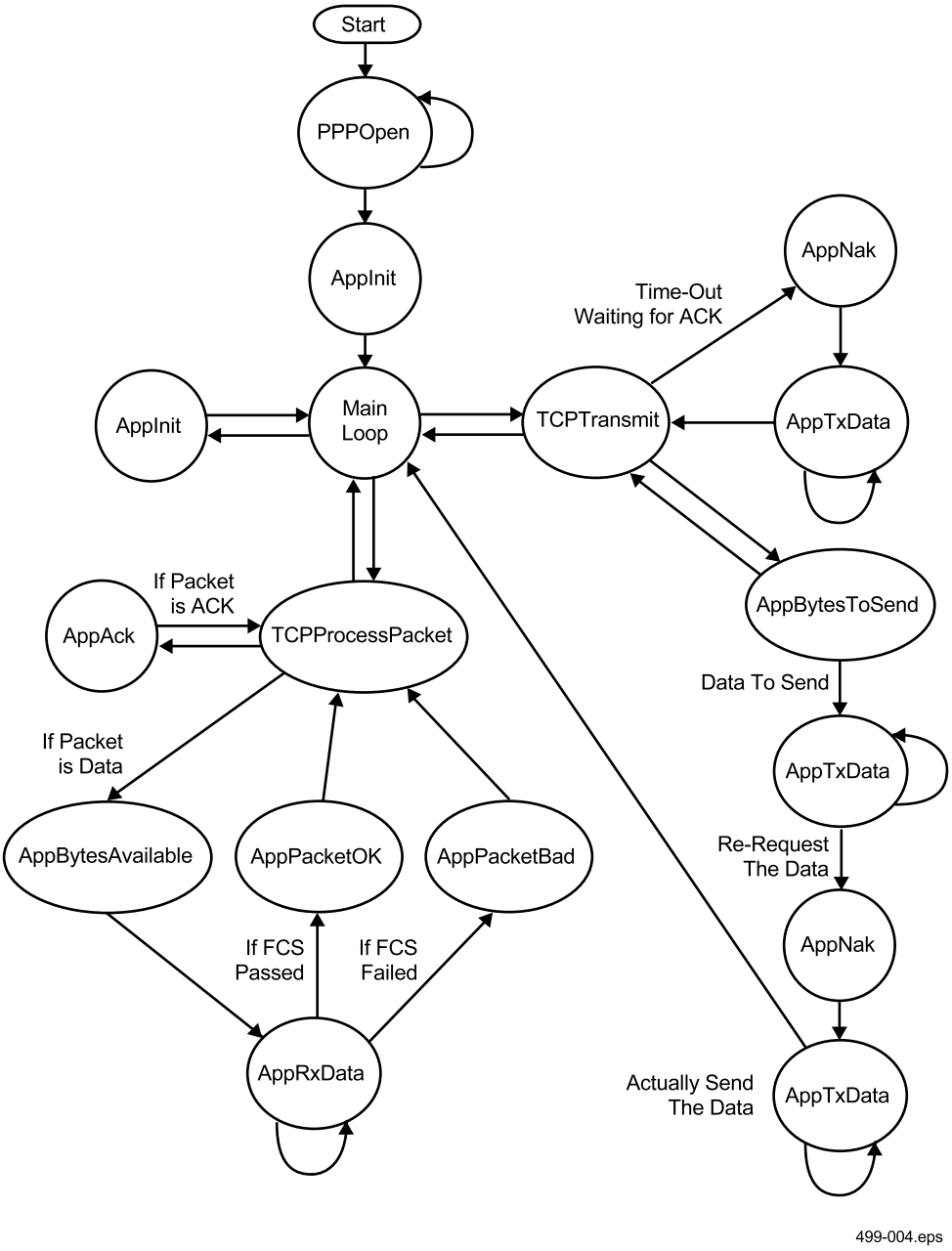

Figure 3-1. SX-Stack Demo Program State Diagram

Periodically, the main loop calls `TCPTransmit` to handle any transmit requests from the application. If `TCPTransmit` has timed out waiting for an acknowledgement of a previous transmission, `AppNak` is called to inform the application that the transmission has failed and must be retried. `AppTxData` is called once for each byte of the retransmission attempt. There can be only one outstanding request to transmit a packet.

`TCPTransmit` calls `AppBytesToSend` to check for new transmit requests. If there is no data to be transmitted, it just returns. If there is data, `AppTxData` is called to get the data from the application. The TCP header is required to include a checksum of the data. Because there isn't enough space in RAM to buffer the whole packet, the application is called to produce the data twice. On the first pass, the checksum is computed and saved. Then, `AppNak` is called to force the application to produce the data again. On the second pass, the packet is transmitted with the saved checksum. The saved checksum is also used if the packet must be retransmitted because of a time-out while waiting for acknowledgement.

4.0 IP Addresses

The IP address of the SX board is specified by constant declarations as 192.168.11.1 in the "IP constants" section. For the eSX demos, the SX board attempts to connect to a server at address 192.168.11.2. This address is specified by constant declarations in either the "SMTP constants" or "POP3 constants" sections.

5.0 iSX Web Server Demo

The iSX web server demo is used with a web browser running on a PC to demonstrate the HTTP protocol. HTTP defines a client-server relationship, in which all actions are initiated from the client side. The iSX server waits for an HTTP client to make a GET request, then responds with either the requested file or a 404 error message indicating that the requested file was not found.

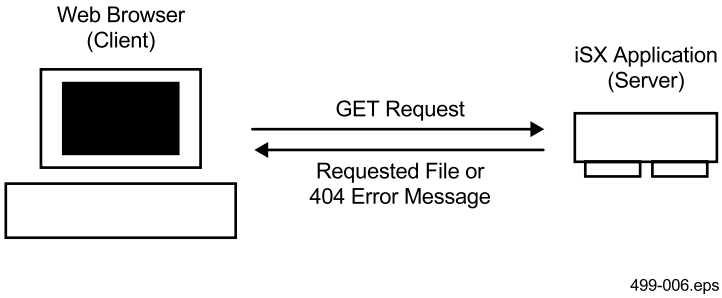

Figure 5-1. HTTP Client-Server Interaction

A GET request consists of the GET keyword, a space character, a Uniform Resource Identifier (URI), another space character, and the HTTP version number. An example is shown below:

```
GET /index.html HTTP/1.0
```

The iSX server looks for the first letter of the keyword, and it ignores any request that doesn't begin with "G". Then, it looks for the space characters and captures the URI. The version number and any following information such as request modifiers are ignored.

Only one response is sent for each GET request. An HTML file may reference other files, such as images embedded in the web page. In this case, the web browser may make additional GET requests to download those files.

The iSX server uses a serial EEPROM file manager Virtual Peripheral to access web pages stored in an external memory chip. A hashing function is used to look up the requested URI in the file system. If the web page is not found, a page containing the standard 404 error message is transmitted.

Text is stored as a sequence of ASCII bytes, and GIF images are stored as binary data. If the file exceeds the maximum packet size, it is transmitted as a series of maximum-size packets, followed by a packet that may be smaller than the maximum size as shown in Figure 5-2.

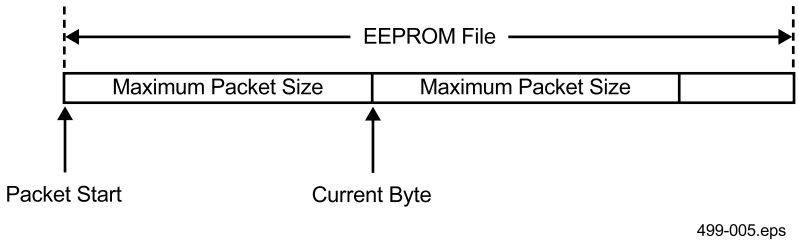

Figure 5-2. EEPROM File System

Two pointers are used to access the file. Initially, both pointers reference the beginning of the file. As each byte is read, the Current Byte pointer is incremented. After the first packet is transmitted, the pointers will address the locations shown in Figure 5-2.

If `AppNak` is called after the packet is transmitted, the packet was not acknowledged, and the Current Byte pointer is set equal to the Packet Start pointer. If `AppAck` is called, the transmission was successful, and the Packet Start pointer is set equal to the Current Byte pointer. The second packet can then be transmitted.

The maximum packet size that can be transmitted is 256 bytes, plus the header size. This limitation is imposed by the use of 8-bit pointers. The code could be rewritten to use larger pointers, however the only advantage of doing so would be increased throughput. Embedded Internet applications using the SX communications controllers typically handle short messages that are not likely to be throughput-limited.

6.0 eSX E-Mail Appliance

6.1 SMTP DEMO

The eSX SMTP demo is an SMTP client, with the EServ program running on a PC acting as the SMTP server. The SMTP demo transmits a message in response to receiving a ping packet. A ping packet is an ICMP echo message. Normally, an ICMP-compliant host responds to a ping packet with an ICMP echo reply message. But for this demonstration, the response to a ping packet (or any other kind of ICMP message) is to send an e-mail message in addition to the echo reply message.

The client-server interaction proceeds in lock-step, with each command from the client generating a reply from the server, as shown in Figure 6-1.

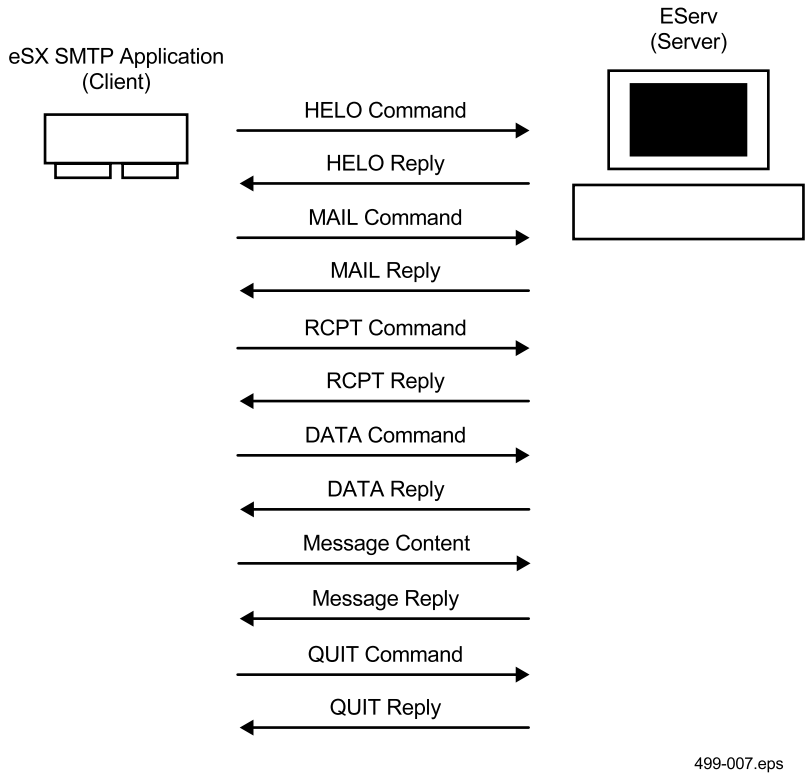

Figure 6-1. SMTP Client-Server Interaction

Replies from the server begin with three-digit code numbers. These numbers completely define the reply, but they are usually followed with descriptive text for the benefit of anyone viewing the message traffic as raw ASCII text. If the first digit of the code number is 2 or 3, the command was accepted. Otherwise, an error occurred and the transmission is aborted.

As commands are issued by the client and command replies are received from the host, the SMTP demo runs through a state machine. These states are listed in the constant declarations in the "SMTP constants" section, and they are described in Table 6-1.

Table 6-1. SMTP State Descriptions

Constant Name	State Number	State Description
SMTPStateClosed	0	Initial state of the SMTP state machine.
SMTPStateHello	1	Send HELO command.
SMTPStateHelloAck	2	Receive reply to HELO command.
SMTPStateMail	3	Send MAIL command.
SMTPStateMailAck	4	Receive reply to MAIL command.
SMTPStateRcpt	5	Send RCPT command.
SMTPStateRcptAck	6	Receive reply to RCPT command.
SMTPStateData	7	Send DATA command.
SMTPStateDataAck	8	Receive reply to DATA command.
SMTPStateMesg	9	Send message content.
SMTPStateMesgAck	10	Receive reply to sending the message content.
SMTPStateQuit	11	Send QUIT command.
SMTPStateQuitAck	12	Receive reply to QUIT command.
SMTPStateFinished	13	Close connection.

The session begins with the HELO command from the client and a reply from the server, which serve to identify the client to the server and the server to the client. It also confirms that both client and server are in their initial state (no transaction in progress, etc.).

The MAIL command indicates that the client is making a request to send mail. It includes the address of the sender of the e-mail.

The RCPT command specifies address of the recipient of the e-mail. This command may be repeated to specify multiple recipients, however the SMTP demo only sends to one recipient.

The DATA command is used to specify the message content. After receiving a reply to the DATA command, the client sends several lines of text. The text prior to the first blank line is the header of the e-mail, which shows the sender's e-mail address, the recipient's e-mail address, and the subject of the e-mail.

The text following the first blank line is the body of the message. This is terminated by a line consisting only of ".". The line containing the "." alone is stripped from the actual message content passed along to the e-mail recipient. The server sends a reply after receiving the line with the "." alone.

Finally, a QUIT command issued by the client. After receiving the QUIT reply, the client closes the connection.

If a second ping packet arrives while the STMP demo is processing the first ping packet, the first transmission is aborted and a second transmission is started. The standard ping command should be used with the "-n 1" option to make it send just one ping packet. By default, the ping command sends four ping packets, which can have various effects on the SMTP demo.

The message transmitted by the SMTP demo resides in program memory, in the SMTP implementation of the AppPacketOK call. You can edit this packet to change the name of the e-mail recipient, the contents of the message, etc. Be careful not to increase the size of the message by more than a few bytes, or some of the code will be pushed onto the next page of program memory.

7.0 POP3 Demo

The eSX POP3 demo is a POP3 client, with the EServ program running on a PC acting as the POP3 server. The POP3 demo waits for a PPP connection to be made, then it retrieves its e-mail and closes the connection. The only action taken with the e-mail is to copy it to the second serial port (i.e. the debug port) on the demo board. After closing the connection, the POP3 demo waits for another PPP connection.

The client-server interaction proceeds in lock-step, with each command from the client generating a reply from the server, as shown in Figure 7-1. All replies begin with a status indicator, which is either “+OK” or “-ERR”. The POP3 demo closes the connection if it receives a “-ERR” status indicator (or any response that does not begin with “+”).

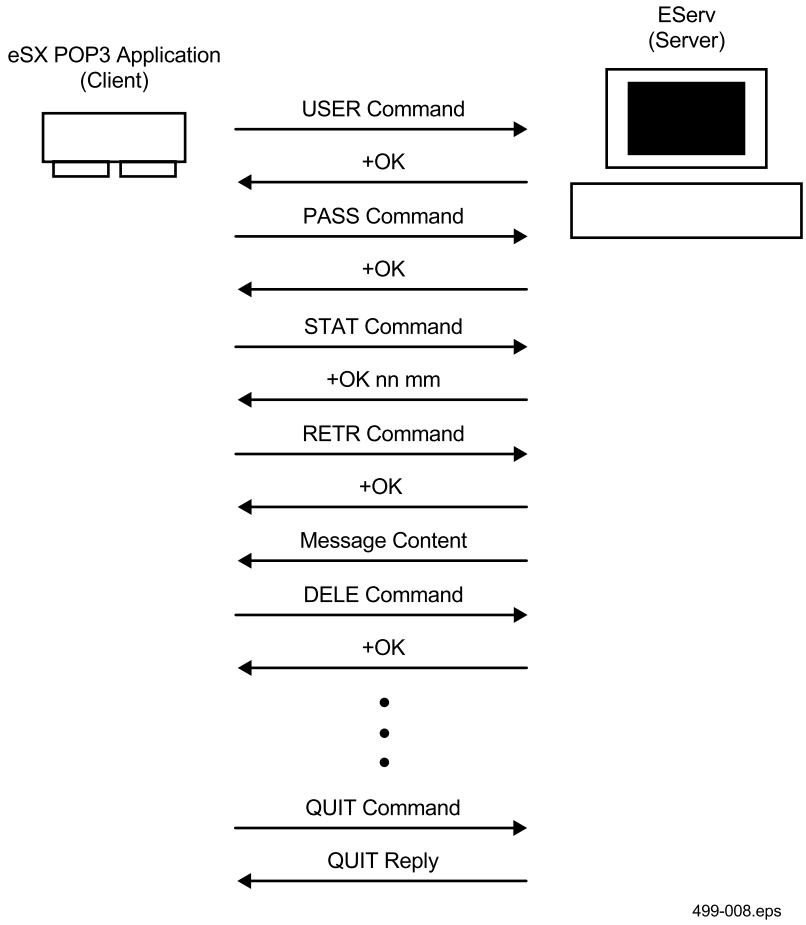

Figure 7-1. POP3 Client-Server Interaction

The USER and PASS commands are used to send a user name and password. These are stored in program memory in the PPP Virtual Peripheral, between the code for the PPPFCSData and PPPReceive functions.

The reply to the STAT command reports two numbers (shown as nn and mm in Figure 7-1): the number of messages waiting to be downloaded and a byte count for the messages. For each message, the client issues a RETR command to retrieve it and a DELE command to request deleting it.

After retrieving and requesting deletion of all its e-mail, the POP3 demo issues a QUIT command. This indicates to the server that it may delete all the messages previously requested to be deleted. After receiving the reply to the QUIT command, the POP3 demo closes the connection.

Lit #: SXL-AN35-01

Sales and Tech Support Contact Information

For the latest contact and support information on SX devices, please visit the Scenix Semiconductor website at www.scenix.com. The site contains technical literature, local sales contacts, tech support and many other features.

SCENIX

1330 Charleston Road
Mountain View, CA 94043
E-mail: sales@scenix.com
Web site:scenix.com
Tel.: (650) 210-1500
Fax: (650) 210-8715