CHAMELEON 32

X.25 C SIMULATOR

ENHANCED HDLC C SIMULATOR

Version 1.1

TEKELEC
26580 Agoura Road
Calabasas, California
91302

Part Number 910–3426

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION AND INSTALLATION

Introduction

The X.25/HDLC C Simulator package is an optional package which provides the following:

- Enhanced automatic HDLC simulator and HDLC library which enables you to access the simulator. This library provides a number of features not available in the standard HDLC library which comes with the C Development System.
- Automatic X.25 simulator and X.25 library which enables you to interact with the simulator.

In order to use this package, you must have the C Development System installed on your Chameleon 32.

Installation

To install the X.25/HDLC Simulation package on your Chameleon, do the following:

1. Turn on the Chameleon 32 and insert the disk labeled **The Installer** (930-3001-01) into the floppy disk drive.
2. When the first screen appears, the Chameleon 32 will automatically boot from the floppy drive.
3. A prompt appears which asks whether or not you want to format the hard disk. Press **n** (no) in response to this prompt.
4. A menu appears with three options, as follows:

   F1 **Install**, which displays the installation menu.

   F2 **Show Installation Status**, which displays the software packages that have been installed, and their version numbers.

   F3 **Identify Installation Disk**, which displays the title and version number of any installation disk in the floppy drive.

5. Remove **The Installer** from the floppy disk drive and press ***F1 Install***. The screen will display:

   Insert Disk #1 of package to be installed and press RETURN

6. Insert the **X.25/HDLC Simulator** disk into the drive and press ***Return***. The installation will proceed immediately and the screen will display:

   Installing *package name* **Disk 1 of** *x*.

   When the installation of the disk is complete, the Install Menu will be redisplayed.

7. You can then reset the Chameleon and use the new software.

Customer Support

If you have any problems installing software, please call Tekelec Customer Support at 1-800-441-9990. In California, call 1-818-880-5656.

CHAPTER 2
ENHANCED AUTO HDLC SIMULATOR

Description

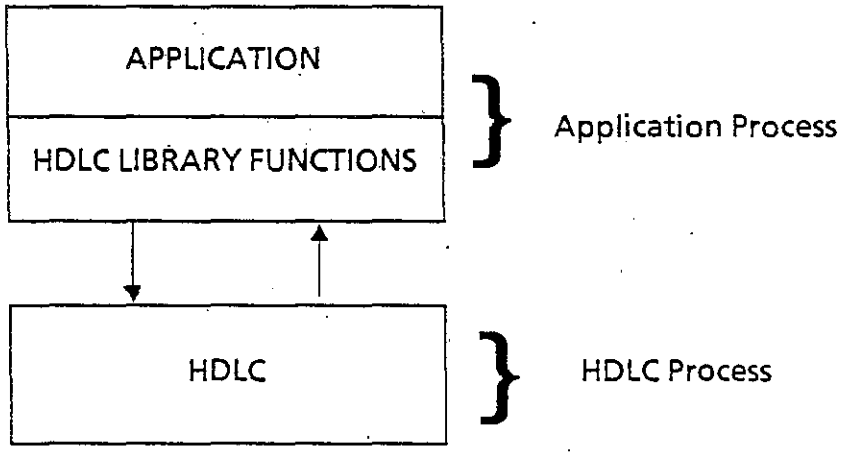
The Enhanced Auto HDLC Simulation C Library is called **libhdlci.a** and is located in the **\lib** directory. The Auto HDLC functions are listed on the next page so that you can locate them quickly within this section. Following the index, the functions are in alphabetical order with one function per page.

This version of the HDLC library is backward compatible with the standard C HDLC library. This means that you can use your existing HDLC simulation programs by relinking the applications with the Enhanced HDLC library.

The Enhanced HDLC library provides you with the ability to transmit and receive packets up to 8 Kbytes in length. This is substantially larger than the maximum packet size of 512 bytes which is allowed in the standard HDLC library. The Enhanced HDLC library also provides some extra functionality for greater testing capability. For example, you can transmit packets with illegal sequence numbers, frame types, and frame lengths. You can also turn the protocol timers on and off.

Note: This library runs on the same processor as the application, which causes packet transmission to be slower than with the standard HDLC library. This version of HDLC is therefore not suitable for load generation or time critical applications.

The Enhanced Auto HDLC library functions is illustrated below:



Enhanced HDLC Functions

The Enhanced HDLC Library functions are described on the following pages:

Default HDLC Values

The table below lists the default values for the Enhanced HDLC simulator. The HDLC protocol parameter values can be set using the hdlc_setup() function when the simulator is initialized and can be changed individually using the functions indicated in the table below.

ITEM	DEFAULT VALUE	FUNCTION
Acknowledgement	ON	HDLC_SETACK
K (window size)	7	SET_WINDOW
Modulo	8	HDLC_SET_MOD
N1	2000 bytes	SET_N1
N2	10	SET_N2
Port	Port A	SETPORT
Process Priority	10	HDLC_SETPRI
T1	2.5 sec	SET_T1
T2	0.05 sec	HDLC_SET_T2

Default HDLC Values

FLUSH

Declaration void flush()

Description This function clears all outstanding I-frames in the reception buffer.

GETPORT

Declaration int getport()

Description This function returns which port is currently communicating with the library. Use the setport function to select the port. The default is Port A.

Returns

0	Port A selected
1	Port B selected

HDLC_DATA_TRANS

Declaration

```
int   hdlc_data_trans (type)
int   type;
```

Range

type	0	ABSORB
	1	ECHO
	2	NORM

Description

This function specifies how I-frames are to be handled once received by HDLC.

ABSORB configures HDLC to discard all received I-frames (they are, however, acknowledged).

ECHO configures HDLC to retransmit all received I-frames.

NORM selects the default mode where I-frames are delivered to the application.

Returns

0	Successful
-1	Parameter error

HDLC_SETACK

Declaration

```
int     hdlc_setAck (val)
char    val;
```

Range

val:	0	ON
	1	OFF

Description

This function turns the acknowledgement of I-frames on or off.

Returns

0	Successful
-1	Parameter error

HDLC_SET_MOD

Declaration

```
int   hdlc_set_mod (val)
int   val;
```

Range

val:

1	Modulo 8
2	Modulo 128
3	Interim Modulo 128. This conforms to the interim Modulo 128 standard, in which unnumbered frames consist of two bytes, with the second byte being zero, except for the P/F bit.

Description

This function sets the modulo of communication for HDLC. The default mode is Modulo 8.

Returns

0	Successful
-1	Parameter Error

HDLC_SETPRI

Declaration

```
hdlc_setpri (priority)
unsigned char priority;
```

Range

priority 0 - 255 (255 = highest priority)

Description

This function sets the priority of the HDLC process. The priority can be changed for optimum speed performance depending on whether the HDLC process is mainly transmitting or receiving. If optimum performance is not required, this function need not be called.

The default priority is 10. Programs started from the C shell have priority 200.

Note

If this function is used, it must be called before initp1().

Returns

0 Successful

HDLC_SET_T2

Declaration

```
int hdlc_set_t2 (val)
int val;
```

Range

0 - 255	.01 second units
-1	Timer not used

Description

This function sets the value of the frame acknowledgement timer T2 in units of .01 seconds. If -1 is specified, the timer value is not used.

Returns

0	Successful
-1	Parameter error

HDLC_SETUP

Declaration

```
int hdlc_setup (n1,n2,t1,t2,k,mode)
int  n1;
int  n2;
int  t1;
int  t2;
int  k;
int  mode;
```

Range

n1	Size of HDLC frame in the range 1 - 8200 bytes
n2	Number of retransmissions in the range 1 - 255
t1	Retransmission timer in the range 1 - 255 in .01 second units. A value of -1 turns use of the timer off.
t2	Acknowledgement timer in the range 1 - 255 in .01 second units. A value of -1 turns use of the timer off.
k	Window size. For Mod 8, the range is 1 - 7. For Mod 128, the range is 1 - 127.
mode	Modulus of operation: 1 = Mod 8 2 = Mod 128 3 = Mod 128 Interim

Description

This function sets up of the HDLC system parameters using one function. Each of the parameters can be set separately as indicated in the table on page 2-2.

Returns

0	Successful
-1	Parameter error

HDLC_TRANSMITFRAME

Declaration

```
int hdlc_transmitFrame (type, p_f, c_r, e, mode, rFrame)
unsigned char type;
unsigned char p_f;
unsigned char c_r;
unsigned char e[];
int mode;
unsigned char rFrame;
```

Description

This function transmits the specified frame type in the specified mode. The link need not be established to transmit frames with this function.

Note that the parameters *e* and *rFrame* are relevant only when transmitting a frame reject.

type Specifies the type of frame being transmitted:

0x01 I_RR
0x02 I_RNR
0x03 I_REJ
0x04 I_SABM
0x05 I_SABME
0x06 I_DISC
0x07 I_FRMR
0x08 I_UA
0x09 I_DM

p_f: Sets the Poll/Final bit on or off:

0x00 PF_ON
0x01 PF_OFF

c_r: Sets the Command/Response bit on or off:

0x00 CR_OFF
0x01 CR_ON

e[5] This parameter is relevant only when transmitting a rejected frame:

e[0]:	0x00	CR0	Sets the C/R bit to 0
	0x01	CR1	Sets the C/R bit to 1
e[1]:	0x00	W0	Sets the W bit to 0
	0x01	W1	Sets the W bit to 1
e[2]:	0x00	X0	Sets the X bit to 0
	0x01	X1	Sets the X bit to 1
e[3]:	0x00	Y0	Sets the Y bit to 0
	0x01	Y1	Sets the Y bit to 1
e[4]:	0x00	Z0	Sets the Z bit to 0
	0x01	Z1	Sets the Z bit to 1

mode: Specifies the mode of transmission for the frame:

0	GOOD_CRC	Transmit with a good CRC
3	NR_GT	Transmit with an N(r) sequence number greater than expected
4	NR_LT	Transmit with an N(r) sequence number less than expected
5	NS_GT	Transmit with an N(s) sequence number greater than expected
6	NS_LT	Transmit with an N(s) sequence number less than expected
7	TOO_SHORT	Transmit too short a frame
8	TOO_LONG	Transmit too long a frame

rFrame: This parameter is relevant only when transmitting a rejected frame. It specifies the type of rejected frame to transmit:

0x00	C_I
0x01	C_RR
0x05	C_RNR
0x09	C_REJ
0x2F	C_SABM

0x6F C_SABME
0x43 C_DISC
0x87 C_FRMR
0x63 C_UA
0xDF C_DM

The valid error codes for a specific message type are described in the following diagram:

FRAME	GOOD	BAD CRC	ABORT	NR_GT	NR_LT	NS_GT	NS_LT	TOO LONG	TOO SHORT
I_RR		X	X			X	X		
I_REJ		X	X			X	X		
I_RNR		X	X			X	X		
I_SABM		X	X	X	X	X	X		
I_SABME		X	X	X	X	X	X		
I_DISC		X	X	X	X	X	X		
I_FRMR		X	X	X	X	X	X		
I_UA		X	X	X	X	X	X		
I_DM		X	X	X	X	X	X		

X = NOT RELEVANT MODES

Returns

-1	Parameter error
0	Successful
1	Front End Processor busy (transmitting previous packet)
2	initp1 not performed

HDLC_TRANSMITMODE

Declaration

```
#include <hdlc.h>

int     hdlc_transmitMode (buffer, length, mode)
char    *buffer;
int     length;
int     mode;
```

Range

buffer: Pointer to the buffer containing the data to transmit

length: Number of bytes to transmit

mode:
- 0 GOOD_CRC
- 1 BAD_CRC
- 2 ABORT_SEQ

Description

This function sends an I-frame using the specified mode. When the selected mode is GOOD_CRC, this function is equivalent to transmit().

Returns

0	Successful
1	Front End Processor is busy
2	Initp1 not performed
3	Link not established
-1	Parameter error

HDLC_TRANSMITTRP

Declaration

```
int     hdlc_transmitTrp (buffer, length, mode)
char    *buffer;
int     length;
int     mode;
```

Range

buffer: Pointer to the buffer containing the data to transmit

length: Number of bytes to transmit

mode:

0	GOOD_CRC
1	BAD_CRC
2	ABORT_SEQ

Description

This function sends an unformatted string on the line transparently to HDLC.

Returns

0	Successful
1	Parameter error

INITP1

Declaration

```
int   initp1(type1, type2, encode, bitrate)
char  type1;
char  type2;
char  encode;
unsigned long  bitrate;
```

Range

type1	0	DCE
	1	DTE
	2	ISDN
type2	0	Network
	1	Subscriber
encode	0	NRZ
	1	NRZI
bitrate	50 - 64000	

Description

This function starts the HDLC process and initializes the port specified by setport().

Returns

0	Successful
-1	Parameter error
-2	HDLC executable code not found
-3	Port already initialized

P1RESET

Declaration

```
int  p1reset(kind)
char kind;
```

Range

kind	0	Restart simulation
	1	Stop simulation

Description

This function either restarts the simulation or stops the simulation. The restart function brings HDLC to the same status (default values, etc.) as after an initp1(). The stop function stops the simulation for the specified port and a new initp1() can be issued. If both ports are stopped, the HDLC process is removed.

Returns

0	Successful
-1	Parameter error

RECEIVE

Declaration

```
int   receive(packet)
char  *packet;
```

Description

This function receives an I-frame from the Front End Processor and places the I-field frame starting at the address pointed to by the passed variable ***packet.***

The external global variable ***rxlen*** will be set to the length of the received frame. If *rxlen* = 0, then no I-frame was received.

Returns

0	Successful
1	Link not established
2	**initp1** not performed

RECEIVEWAIT

Declaration

```
#include <mtosux.h>

int     receiveWait (packet, waitTime)
char        *packet;
unsigned long   waitTime;
```

Range

packet: Pointer to receive buffer

waitTime: Maximum time to wait for a message, in the following units (t is the number of units):

SEC + t	Seconds
HMS + t	.1 seconds
TMS + t	.01 seconds
MS + t	.001 seconds
NOEND	Wait forever
IMONLY	Attempt to receive one time only

Description

This function will wait for the reception of an I-frame. The time it will wait is specified by waitTime. The frame will be put in the buffer pointed out by packet. The global variable rxlen will be set to the length of the received frame.

If rxlen = 0, then no I-frame was received.

Returns

0	Successful
1	Link not established
2	initp1 not performed
4	Timeout

SET_N1

Declaration

```
int  set_n1(val)
int  val;
```

Range

val 1 - 8200

Description

This function sets the value of N1 (maximum size of a frame in bytes). The default value is 2000.

Returns

0	Successful
-1	*val* outside of range

SET_N2

Declaration

```
int  set_n2(val)
int  val;
```

Range

val 1 - 255

Description

This function sets the value of N2 (the maximum number of retransmissions). The default value is 10.

Returns

0	Successful
-1	*val* outside of range

SETPORT

Declaration

```
int  setport(port)
int  port;
```

Range

port	0	Port A
	1	Port B

Description

This function sets the library to exchange information with either Port A or Port B. Use the getport function to determine which port is currently communicating with the library.

Returns

0	Successful
-1	Parameter out of range
-2	Attempted to select Port B on Chameleon with a single port (Port A)

SET_T1

Declaration

```
int  set_t1(val)
int  val;
```

Range

val	1 - 255	.01 second units
	-1	Turns off use of timer

Description

This function sets the value of the T1 frame level timer in .01 second units. The default value is 255 (2.5 seconds).

Returns

0	Successful
-1	*val* outside of range

SET_WINDOW

Declaration

```
int  set_window(val)
char  val;
```

Range

val	1 - 7	(Valid range for Modulo 8)
	1 - 127	(Valid range for Modulo 128)

Description

This function sets the window size for the frame level. The default value is 7. The modulus of operation (Mod 8 or Mod 128) must be set before setting the window size.

Returns

0	Successful
-1	*val* outside of range

SLOF

Declaration slof()

Description This function disconnects the link at the frame level by sending a **DISCONNECT.** Be sure to check the link status for the result of this command.

Returns None

SLON

Declaration slon()

Description This function attempts to establish a link at the frame level by sending a **SABM** or **SABME** depending on the modulus of operation. Be sure to use STATUS to ascertain that the link is established before you use transmit() to transmit data.

Returns None

STATUS

Declaration int status()

Description This function returns a value indicating the status of the frame level.

Returns

0	Disconnected
1	Link connection requested
2	Frame reject state
3	Link disconnection requested
4	Information Transfer State
5	Local Station Busy
6	Remote Station busy
7	Local and remote stations busy

TRANSMIT

Declaration

```
transmit (packet,length)
char  *packet;
int   length;
```

Description

This function transmits an I-frame with the I-field set to the number of bytes specified by the passed variable *length*, starting at the address pointed to by the passed pointer ***packet**.*

Returns

0	Successful
1	Front End Processor busy (transmitting previous packet)
2	**initp1** not performed
3	Link not established

CHAPTER 3
X.25 SIMULATOR

Introduction

The X.25 Simulator provides an automatic X.25 simulator and a library of protocol functions which enable a C application to access the simulator. The following features are supported:

- Multiple Logical Channels. (The maximum number of LCNs is determined by available memory and the packet layer window size.)
- SVCs and PVCs
- CCITT X.25 or TRANSPAC X.25
- Automatic handling of sequence numbering, flow control, and error recovery
- Complete control over the following parameters:
  - Called address
  - Calling address
  - User data
  - Facilities (negotiating is not supported in this release)
  - Cause
  - Diagnostic
  - Diagnostic explanation
  - M-bit, D-bit and Q-bit
  - Layer 2 parameters
- Support of the following X.25 packet types:
  - Call Request
  - Clear Request
  - Restart Request
  - Reset Request
  - Diagnostic Request
  - Data Request
  - Interrupt Request

Functions

The X.25 library functions are defined in the **libx25.a** file located in the the \lib directory. General instructions for using the X.25 library functions in a C program are provided on the following page. Each function is described fully on the pages indicated below.

Include Files

You must include the **x25lib.h** file when using X.25 library functions.

Note

If your program also uses functions from libraries provided in the base Chameleon C Development system, you may need to include the cham.h file, which is provided with the base Chameleon C package. If so, in your program, include the cham.h file before you include the x25lib.h file to ensure that the appropriate constants and variables are used. In other words, the include statements should be in the following order:

```
.
.
.
#include cham.h
#include x25lib.h
.
.
.
```

Description

The X.25 functions provide access to the X.25 automatic simulator. Some of the functions must be used in a specific sequence in a program in order to establish a call and transfer data. This sequence is described below.

Initializing the X.25 Simulator

You must first initialize the X.25 simulator. To do this, your application must call two functions:

- x25_Setport() selects the Chameleon port on which to initialize the simulator
- x25_Start() initializes the X.25 simulator using the specified HDLC and X.25 parameters. These parameters include the X.25 standard (CCITT or TRANSPAC) and the number of PVCs and SVCs that the X.25 simulator will accommodate.

  For PVCs, Logical Channel Numbers (LCNs) are assigned automatically starting with the lowest LCN used for the selected X.25 standard:

  - For CCITT X.25, PVCs are assigned LCNs beginning with LCN 1.
  - For TRANSPAC X.25, PVCs are assigned LCNs beginning with LCN 0.

To initialize the X.25 simulator on both Ports A and B, you must call x25_Setport() and then x25_Start() for each port.

When initialization is successful (x25_Start returns 0), the simulator is started, the physical layer is set up, and HDLC is initialized. The next step is to establish an X.25 link between the Chameleon and the Device Under Test (DUT).

Link Establishment

The link can be established by either the Chameleon or by the DUT, using the following functions:

- For the Chameleon to initiate link establishment, your application must call the function x25_LinkReq() with the Type parameter specifying to set the link ON.
- For the Chameleon to wait for the DUT to initiate link establishment, your application must call the function x25_Receive(). (This is the function which is used to receive data from the DUT.)

When a packet is received, x25_Receive() sets the parameters ReqType (which identifies the packet type) and RetVal (return value).

An incoming Call Request is indicated by the receipt of a Link Status (ReqType = aiLink) with a RetVal = 9 (Link is on) in response to the x25_Receive().

For PVCs, data transfer can then occur. For SVCs, a call must be established, as described below.

Call Establishment

For SVCs, a call can be established by either the Chameleon (outgoing call) or by the DUT (incoming call).

To establish an outgoing call, the following sequence must occur:

- You application must call the function x25_CallReq() which causes the simulator to transmit a Call Request to the DUT. A call to this function must be made for each call you are trying to establish. x25_CallReq() includes a RefNo parameter which enables you to assign a reference number to each call request. This enables you to match the assigned LCN to its corresponding Call Request.

- Generally, there are two responses to each Call Request that you make. Your application must call x25_Receive to receive the first response to your call request. The first response is from the X.25 simulator, and is generally an Assigned LCN (ReqType = aiCallLCN) which returns the Logical Channel Number assigned to the call. When the aiCallLCN message is received, store the LCN and LCGN so that these parameters can be used for additional messages relating to that call.

  If an LCN cannot be assigned to the call, you will receive an Outgoing Call Rejected (ReqType = aiCallRej) as the only response to the call request. This generally occurs if an LCN is not available for the call. The number of LCNs available is configurable in the x25_Start function and must be the same as the DUT.

- Your application must call x25_Receive a second time in order to receive a response from the DUT. If an Outgoing Call Accepted (ReqType = aiCallAcc) is received, the call request was accepted and the call is considered established.

If you receive an Outgoing Call Rejected (ReqType = aiCallReq), the call request was rejected by the DUT and the call cannot be established.

Use the following sequence to handle an incoming call:

- The application must call the function x25_Receive(). At any time this function is called, an incoming call could be received. An incoming call request is indicated by the receipt of an Incoming Call (ReqType = aiCallSetup), which also includes the LCN assigned to the call.
- To reject an incoming call, your application must call the function x25_ClearReq() for the indicated LCN.
- There are two methods available for accepting incoming call requests. The method used is determined by the Answer parameter in the function x25_Start() which was called to initialize the X.25 simulator.
  - If Answer is Automatic, incoming call requests are accepted automatically by the simulator; the application does not have to transmit a response to the DUT.
  - If Answer is Manual, your application must call the function X25_CallAcc() in order to accept the call for the indicated LCN.

For SVCs, once the call is established, data can be transmitted and received as described below.

Data Transfer

During data transfer, the X.25 simulator handles flow control, sequence numbering, and error recovery automatically for you.

- To transmit a Data packet from the Chameleon to the DUT, call the function x25_DataReq().
  - For PVCs, use the fixed LCN and LCGN corresponding to that PVC.
  - For SVCs use the LCN and LCGN received in the aiCallLCN message for that call.
- An Incoming Data packet from the DUT is indicated by the receipt of an Incoming Data (ReqType = aiData) in response to an x25_Receive().

Call Clearing

Call Clearing is relevant for SVCs only.

- To transmit a Clear Request packet from the Chameleon to the DUT, call the function x25_ClearReq().
- A Clear Request packet from the DUT is indicated by the receipt of a Call Cleared (ReqType = aiCallCleared) in response to an x25_Receive().

If the X.25 simulator has to clear a call (due to protocol requirements), it sends a Clear Request both to your X.25 application and to the DUT. Your application can determine whether the Clear Request was initiated by the X.25 simulator or by the DUT by checking the RetVal parameter of the received aiCallCleared.

Call Reset

Call Reset is valid only for PVCs or SVCs in data transfer state.

- To transmit a Reset Request packet from the Chameleon to the DUT, call the function x25_ResetReq().
- An Incoming Reset packet from the DUT is indicated by the receipt of an Incoming Reset (ReqType = aiResetLCN) in response to an x25_Receive().

Interrupt Data

Interrupt Data is valid only for PVCs or SVCs in data transfer state.

- To transmit an Interrupt Data packet from the Chameleon to the DUT, call the function x25_IntDataReq(). This function is valid only for PVCs or for SVCs currently in Data Transfer state.
- An Interrupt Data packet from the DUT is indicated by the receipt of an Incoming Interrupt Data (ReqType = aiIntrupt) in response to x25_Receive().

Restart

- To transmit a Restart Request packet from the Chameleon to the DUT, call the function x25_Restart(). A Restart affects all active channels.
- A Restart packet from the DUT is indicated by the receipt of an Incoming Restart (ReqType = aiRestart) in response to x25_Receive().

Diagnostic Packet

- To transmit a Diagnostic packet from the Chameleon to the DUT, call the function x25_DiagReq().
- A Diagnostic packet from the DUT is indicated by the receipt of an incoming Diagnostic (ReqType = aiDiag) in response to x25_Receive().

Stopping the X.25 Simulator

You must stop the X.25 simulator before you exit your application. If you use the C function exit(), this will be done automatically for the ports you have initialized. Otherwise, your application must call these two functions:

- x25_Setport() selects the Chameleon port on which to stop the simulator.
- x25_Stop() stops the X.25 simulator on the selected port.

To stop the X.25 simulator on both Ports A and B, you must call x25_Setport() and then x25_Stop() for each port.

Passing Data

The structure ACB (Application Contents Block) is used to pass data to X.25 for transmitting and receiving X.25 packets. Each function which uses structure ACB describes the parameters which are required for that function.

```
typedef    uch          unsigned char;
typedef    uint         unsigned int;
typedef    ush          unsigned short;

typedef struct {
           uch          Q_Bit;
           uch          D_Bit;
           uch          M_Bit;
           uch          LCGN;
           uch          LCN;
           uch          *pClgAd;
           uch          *pCldAd;
           uch          FacLen;
           uch          *pFacil;
           uint         RegLen;
           uch          *pReg;
           uint         UD_Len;
           uch          *pUserData;
           uch          Cause;
           uch          Diag;
           ush          DiagExpLen;
           uch          *pDiagExp;
           ush          RetVal;
           ush          ReqType;
           int          RefNo;
} ACB;
```

x25_CallAcc

Declaration

```
#include x25lib.h
int     x25_CallAcc(pACB)
ACB *pACB;
```

Description

This function accepts an incoming Call Request from the DUT. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required for this function are as follows:

D_Bit	0 = D-bit not set 1 = D-bit set
*pClgAd	Pointer to Calling DTE Address (maximum of 15 ASCII digits)
*pCldAd	Pointer to Called DTE Address (maximum of 15 ASCII digits)
FacLen	Facilities length
*pFacil	Pointer to Facilities
UD_Len	User Data length
*pUserData	Pointer to User Data

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to the X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_CallAcc()
{
  ACB uACB;
  unsigned char facil[109], userData[128];
  int result;

  facil[0]        = 0x32;
  facil[1]        = 0x42;
  facil[2]        = 0x5c;
  userData[0]     = 0x01;
  userData[1]     = 0x02;

  uACB.D_bit       = 0;
  uACB.pClgAd     = "12345";
  uACB.pCldAd     = "67890";
  uACB.FacLen     = 3;
  uACB.pFacil     = facil;
  uACB.UD_Len      = 2;
  uACB.pUserData  = userData;

  result = x25_CallReq(&uACB);

}
```

x25_CallReq

Declaration

```
#include x25lib.h
int   x25_CallReq(pACB)
ACB *pACB;
```

Description

This function transmits a Call Request packet from the Chameleon to the DUT. It enables the Chameleon to initiate call establishment. Structure ACB is defined on page 3-8.

Range

The parameters in structure ACB required by this function are as follows:

D_Bit	0 = D-bit not set 1 = D-bit set
*pClgAd	Pointer to Calling DTE Address (maximum 15 ASCII digits)
*pCldAd	Pointer to Called DTE Address (maximum 15 ASCII digits)
FacLen	Facilities length
*pFacil	Pointer to Facilities
UD_Len	User Data length
*pUserData	Pointer to User Data
RefNo	Call reference number to identify the call request. This parameter enables you to match the response from the simulator (indicating the assigned LCN) with its corresponding Call Request.

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_CallReq()
{
  ACB uACB;
  unsigned char facil[109], userData[128];
  int result;

  facil[0]        = 0x32;
  facil[1]        = 0x42;
  facil[2]        = 0x5c;
  userData[0]     = 0x01;
  userData[1]     = 0x02;

  uACB.D_bit      = 0;
  uACB.pClgAd     = "12345";
  uACB.pCldAd     = "67890";
  uACB.FacLen     = 3;
  uACB.pFacil     = facil;
  uACB.UD_Len     = 2;
  uACB.pUserData  = userData;
  uACB.RefNo      = 11;     /* This number is used to identify the */
                            /* Corresponding Call Accept/Reject    */
                           /* The user can select any number       */

  result = x25_CallReq(&uACB);

  /** Call x25_Receive to get the assigned LCN then     **/
  /** Call x25_Receive to get Call Accept/Call Reject   **/
  /** The Call Accept will contain LCN, LCGN and RefNo **/

}
```

x25_ClearReq

Declaration

```
#include x25lib.h
int   x25_ClearReq(pACB)
ACB *pACB;
```

Description

This function transmits a Clear Request packet from the Chameleon to the DUT. This function is valid for SVCs only. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required for this function are:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
Cause	Cause code
Diag	Diagnostic code

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_ClearReq(chan)
int chan;              /* High byte LCGN, low byte LCN */
{
  ACB uACB;
  int result;

  uACB.LCGN     = chan>>8;   /* Group number received in Call Accept */
  uACB.LCN      = chan;      /* Channel number received in Call Accept */
  uACB.Cause    = 0x16;
  uACB.Diag     = 0x11

  result = x25_ClearReq(&uACB);

}
```

x25_DataReq

Declaration

```
#include x25lib.h
int   x25_DataReq(pACB)
ACB *pACB;
```

Description

This function transmits a Data packet from the Chameleon to the DUT. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required for this function are:

Q_Bit	0 = Q-bit not set 1 = Q-bit set
D_Bit	0 = D-bit not set 1 = D-bit set
M_Bit	0 = M-bit not set 1 = M-bit set
LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
UD_Len	User data length. This specifies the maximum allowed length of user data in the packet. This is limited by the value specified for MaxFrmSizeN1 (HDLC maximum frame size) in x25Start(): • For Mod 8, UD_Len must be ≤ MaxFrmSizeN1 - 3. • For Mod 128, UD_Len must be ≤ MaxFrmSizeN1 - 4. This should be ≤ the link level frame size (3 for Mod 8 or 4 for Mod 128)
*pUserData	Pointer to User Data

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_DataReq(chan,data,dLen)
int chan;                    /* High byte LCGN, low byte LCN */
unsigned char *data;
int dLen;
{
  ACB uACB;
  int result;

  uACB.LCGN     = chan>>8;     /* Group number received in Call Accept */
  uACB.LCN      = chan;       /* Channel number received in Call Accept */
  uACB.Q_Bit      = 0;
  uACB.D_Bit      = 0;
  uACB.M_Bit      = 0;
  uACB.dLen       = dLen;
  uACB.pUserData  = data;

  result = x25_DataReq(&uACB);
  /* Call x25_Receive to check if Clear/Restart has been received. */
}
```

x25_DiagReq

Declaration

```
#include x25lib.h
int   x25_DiagReq(pACB)
ACB *pACB;
```

Description

This function transmits a Diagnostic packet from the Chameleon to the DUT. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required for this function are as follows:

Diag	Diagnostic code
DiagExpLen	Diagnostic Explanation Length
*pDiagExp	Pointer to Diagnostic Explanation

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_DiagReq()
{
  ACB uACB;
  unsigned char diagExp[32];
  int result;

  diagExp[0]      = 0x31;
  uACB.Diag       = 0x72;
  uACB.DiagExpLen = 1;
  uACB.pDiagExp   = diagExp;
  result = x25_DiagReq(&uACB);
}
```

x25_Getport()

Declaration

```
#include x25lib.h
int    x25_Getport()
```

Description

This function returns the port which is currently selected. (The SelectPort() function enables you to select a port.) This function is used on Dual Port machines when the X.25 simulator is initialized on both ports.

Returns

0 = Port A is active

1 = Port B is active

x25_IntDataReq

Declaration

```
#include x25lib.h
int   x25_IntDataReq(pACB)
ACB *pACB;
```

Description

This function transmits an Interrupt Data packet from the Chameleon to the DUT. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required for this function are as follows:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
UD_Len	User Data length
*pUserData	Pointer to User Data

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_IntDataReq(chan,data,dLen)
int chan;                    /* High byte LCGN, low byte LCN */
unsigned char *data;
int dLen;
{
  ACB uACB;
  int result;

  uACB.LCGN    = chan>>8;  /* Group number received in Call Accept   */
  uACB.LCN     = chan;     /* Channel number received in Call Accept */
  uACB.dLen      = dLen;
  uACB.pUserData  = data;

  result = x25_IntDataReq(&uACB);
  /** Call x25_Receive to check if Clear/Restart has been received. */
}
```

x25_LinkReq

Declaration

```
#include x25lib.h
int   x25_LinkReq(type,waitTime)
int   type;
int   waitTime;
```

Description

This function causes the Chameleon to initiate link establishment or disestablishment by setting the link on or off.

Range

type	0 = Set Link OFF 1 = Set Link ON
int waitTime	Time to wait for the link to be set on or off, in the range 0 - 250 seconds

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error
-8	Link not established within specified time

Sample Program

```
usr_LinkON()
{
  int ret;

  /* Set link on, wait max 30 seconds for link */

  if ( (ret = x25_LinkReq(1,30)) )  printf("Link on failed\n");
  else                              printf("Link established\n");
  return( ret );

}
```

x25_Receive

Declaration

```
#include x25lib.h
#include mtosux.h
int   x25_Receive(pACB, waitTime)
long  waitTime;
ACB *pACB;
```

Description

This function enables the Chameleon to receive data from the DUT or the X.25 simulator into a structure of type ACB (defined on page 3-8). This function should be called after each call to a request procedure in order to get information about incoming Clear, Restart and Link Status packets.

Structure ACB includes parameters which are pointers to buffers to be used for the data received by the Chameleon. The buffers must be initiated by the application program so that data can be copied to them. The alternative is to use NULL, in which case no data will be copied. These techniques are illustrated in the sample program.

Range

waitTime — The time to wait for a response, as follows:

SEC + t	Time in seconds
HMS + t	Time in .01 seconds
TMS + t	Time in .001 seconds
MS + t	Time in .0001 seconds
NOEND	Wait forever
IMONLY	Attempt to receive one time only

Do not use a waitTime < .0010 seconds.

The structure ACB parameters which are set by this function depend on the type of packet received from the DUT. The ReqType parameter is set in structure ACB for all received packets, as defined below.

ReqType — Specifies the type of X.25 packet received from the DUT or the X.25 simulator. The return values for ReqType and the constants defined in x25lib.h for those values are shown in the table on the next page.

ReqType Return Value (Decimal)	CONSTANT DEFINED IN x25lib.h	X.25 PACKET
61	aiCallSetup	Incoming Call
62	aiCallRej	Outgoing Call Rejected
63	aiCallCleared	Call Cleared
64	aiData	Incoming Data
65	aiResetLCN	Incoming Reset
67	aiRestart	Incoming Restart Request or Restart Confirmation or Timeout
68	aiCallAcc	Outgoing Call Accepted
70	aiDiag	Incoming Diagnostic
71	aiIntrupt	Incoming Interrupt Data
72	aiLink	Link Status
73	aiCallLCN	Assigned LCN
74	aiRegister	Incoming Register

ReqType Parameter Values (ACB Structure)

Once you identify the type of X.25 packet received, you can further examine the contents of the received packet by determining the values of the relevant parameters in structure ACB. For each X.25 packet type, the relevant ACB parameters are listed on the following pages.

ReqType = 61
aiCallSetup
(Incoming Call)

This message indicates an incoming Call Request from the DUT. The following parameters are set in structure ACB for aiCallSetup:

D_Bit	0 = D-bit not set 1 = D-bit set
LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
*pClgAd	Pointer to Calling DTE Address (maximum 15 ASCII digits)
*pCldAd	Pointer to Called DTE Address (maximum 15 ASCII digits)
FacLen	Facilities length (0 - 109)
*pFacil	Pointer to Facilities
UD_Len	User Data length (0 - 128 bytes)
*pUserData	Pointer to User Data
RetVal	Always zero

ReqType = 62
aiCallRej
(Outgoing Call
Reject)

This is one of the possible second messages received in response to an x25_CallReq and indicates that the Call Request is being rejected by the DUT. The following parameters are set in structure ACB for aiCallRej:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
FacLen	Facilities length (0 - 109)
*pFacil	Pointer to Facilities
UD_Len	User Data length (0 - 128 bytes)
*pUserData	Pointer to User Data
Cause	Cause code

Diag	Diagnostic code
RetVal	1 = Timeout 3 = Call collision 4 = Call rejected by DUT 7 = No more LCNs available
RefNo	Reference number corresponding to the Call Request when RetVal = 7(No more LCNs available). This value enables you to match the aiCallRej to its corresponding Call Request when multiple calls have been requested.

ReqType = 63
aiCallCleared
(Call Cleared)

This message indicates that a Call Cleared was received from the DUT or the X.25 simulator. The following parameters are set in structure ACB for aiCallCleared:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
FacLen	Facilities length (0 - 109)
*pFacil	Pointer to Facilities
UD_Len	User Data length (0 - 128)
*pUserData	Pointer to User Data
Cause	Cause code
Diag	Diagnostic code
RetVal	0 = Incoming Clear (from DUT) 2 = Internal Clear (from X.25 simulator)

ReqType = 64
aiData
(Incoming Data)

This message indicates that a Data packet was received from the DUT. The following parameters are set in structure ACB for aiData:

Q_Bit	0 = Q-bit not set 1 = Q-bit set
D_Bit	0 = D-bit not set 1 = D-bit set
M_Bit	0 = M-bit not set 1 = M-bit set
LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
UD_Len	User Data length. The range is 0 - 8192 bytes, but is limited by the maximum packet size and frame size specified when the simulator was initialized using x25_Start().
*pUserData	Pointer to User Data
RetVal	Always zero

ReqType = 65
aiResetLCN
(Incoming Reset)

This message indicates that a Reset packet has been received from the DUT. (The channel has been reset, but the call is still active.) The following parameters are set in structure ACB for aiResetLCN:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
Cause	Cause code
Diag	Diagnostic code
RetVal	Always zero

Reqtype = 67
aiRestart
(Incoming Restart Request/Restart Confirmation/ Timeout)

This message indicates that a Restart packet has been received from the DUT. (All SVC calls have been cleared and all PVCs has been reset.) The following parameters are set in structure ACB for aiRestart:

Cause	Cause code
Diag	Diagnostic code
RetVal	1 = Timeout 2 = Internal Restart 5 = Restart Confirm 6 = Incoming Restart

Reqtype = 68
aiCallAcc
(Outgoing Call Accepted)

This is one of the possible second messages received in response to an x25_CallReq and indicates that the Call Request is being accepted by the DUT. The following parameters are set in structure ACB for aiCallAcc:

D_Bit	0 = D-bit not set 1 = D-bit set
LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
*pClgAd	Pointer to Calling DTE Address (maximum 15 ASCII digits)
*pCldAd	Pointer to Called DTE Address (maximum 15 ASCII digits)
FacLen	Facilities length (0 - 109)
*pFacil	Pointer to Facilities
UD_Len	User Data length (0 - 128 bytes)
*pUserData	Pointer to User Data
RetVal	Always zero

ReqType = 70
aiDiag
(Incoming
Diagnostic)

This message indicates that a Diagnostic packet has been received from the DUT. The following parameters are set in structure ACB for aiDiag:

Diag	Diagnostic code
DiagExpLen	Diagnostic Explanation Length (0 - 3)
*pDiagExp	Diagnostic Explanation
RetVal	Always zero

ReqType = 71
aiIntrupt
(Incoming
Interrupt Data)

This message indicates that an Interrupt packet has been received from the DUT. The following parameters are set in structure ACB for aiIntrupt:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
UD_Len	User Data length
*pUserData	Pointer to User Data (0 - 32)
RetVal	Always zero

ReqType = 72
aiLink
(Link Status)

This message indicates that a change in link status packet has been received from the DUT. The following parameters are set in structure ACB for aiLink:

RetVal	9 = Link Up 8 = Link Down

ReqType = 73
aiCallLCN
(Assigned LCN)

This message is received from the X.25 simulator as a response to an x25_CallReq(). It indicates the LCN that will be assigned to the call. All subsequent packets for the call should have the LCN and LCGN set using the indicated values. The following parameters are set in structure ACB for aiCallLCN:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
RetVal	Always zero
RefNo	Reference number corresponding to the Call Request. This value enables you to match the LCN to the appropriate Call Request when multiple calls have been requested.

ReqType = 74
aiRegister
(Incoming
Register)

This message indicates that a Registration packet has been received from the DUT. The following parameters are set in structure ACB for aiRegister:

*pClgAd	Pointer to Calling DTE Address (maximum 15 ASCII digits)
*pCldAd	Pointer to Called DTE Address (maximum 15 ASCII digits)
RegLen	Registration length (0 - 109)
*pReg	Pointer to Registration
RetVal	Always zero

Sample Program

```
usr_Receive()
{
  ACB uACB;
  unsigned char clgAd[16];  /* Max address length + 1 for string terminator */
  unsigned char cldAd[16];  /* Max address length + 1 for string terminator */
  unsigned char facil109];  /* Max facility length                          */
  unsigned char uData[512]; /* Should be set to max packet size             */
  unsigned char Reg[109];   /* Max registration length                      */
  unsigned char diagExp[3]; /* Max diagnostic explanation length            */
  int result;

  uACB.pClgAd     = clgAd;   /* Parameters are copied to these buffers      */
  uACB.pCldAd     = cldAd;   /* If we are not interested in a parameter     */
  uACB.pFacil     = facil;   /* replace the buffer name with 0L             */
  uACB.pUserData  = uData;   /* uACB.pUserData = 0L;                        */
  uACB.pReg       = reg;     /* In this case, no buffer space has to be     */
  uACB.pDiagExp   = diagExp; /* reserved                                    */

  result = x25_ReceiveReq(&uACB,1+HMS);
  switch (uACB.ReqType)
    {
     case aiCallSetup:  handle_inc_call(&uACB);
                        break;
     case aiCallRej:    handle_call_rej(&uACB);
                        break;
     case aiCallCleared:handle_call_clear(&uACB);
                        break;
     case aiData:       handle_data(&uACB);
                        break;
     case aiResetLCN:   handle_reset(&uACB);
                        break;
     case aiRestart:    handle_restart(&uACB);
                        break;
     case aiCallAcc:    handle_call_acc(&uACB);
                        break;
     case aiDiag:       handle_diagnos(&uACB);
                        break;
     case aiIntrupt:    handle_interupt(&uACB);
                        break;
     case aiRegister:   handle_register(&uACB);
                        break;
     case aiLink:       handle_link(&uACB);
                        break;
     case aiCallLCN:    handle_LCN(&uACB);
                        break;
     default:
                        break;
    }
}
```

x25_ResetReq

Declaration

```
#include x25lib.h
int    x25_ResetReq(pACB)
ACB *pACB;
```

Description

This function transmits a Reset Request packet from the Chameleon to the DUT. Structure ACB is defined on page 3-8.

Range

The structure ACB parameters required for this function are as follows:

LCGN	Logical channel group number (0 - 16)
LCN	Logical channel number (0 - 256)
Cause	Cause code
Diag	Diagnostic code

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-5	Parameter error

Sample Program

```
usr_ResetReq(chan)
int chan;              /* High byte LCGN, low byte LCN */
{
  ACB uACB;
  int result;

  uACB.LCGN     = chan>>8;   /* Group number received in Call Accept */
  uACB.LCN      = chan;   /* Channel number received in Call Accept */
  uACB.Cause    = 0x16;
  uACB.Diag     = 0x72;

  result = x25_ResetReq(&uACB);

}
```

x25_RestartReq

Declaration

```
#include x25lib.h
int    x25_RestartReq(pACB)
ACB *pACB;
```

Description

This function transmits a Restart Request from the Chameleon to the DUT. A restart affects the state of all LCNs. The procedure will not return until the DUT has responded or timer T10 has timed out (60 seconds). Structure ACB is defined on page 3-8.

Range

The structure ACB parameters which are required by this function are as follows:

Cause	Cause code
Diag	Diagnostic code

Returns

0	Successful
-1	X.25 simulator is not started
-3	Queue to X.25 simulator is full; try again
-4	Timeout, no data received
-5	Parameter error
-6	No answer from Device Under Test

Sample Program

```
usr_RestartReq()
{
  ACB uACB;
  int result;

  uACB.Cause      = 0x16;
  uACB.Diag       = 0x72;

  result = x25_RestartReq(&uACB);

}
```

x25_Setport

Declaration

```
#include x25lib.h
int   x25_Setport(port)
int   port;
```

Range

port 0 = Port A
1 = Port B

Description

This function selects a port when the X.25 simulator is initialized on both ports. If x25_Setport is not used in the application program, the default is Port A.

Returns

0	Successful
-5	Parameter error

x25_Start

Declaration

```
#include x25lib.h
int         x25_Start(pLPB)
LPB         *pLPB;
```

Description

This function initializes the X.25 simulator on the selected port using the HDLC and X.25 parameters specified in the structure LPB. The structure LPB (Link Parameter Block) is as follows:

```
typedef struct {
        ulong       Baud;
        uch         Encoding;
        uch         DC_TE;
        uch         Net_Sub;
                    Interface;
        int         ReTxN2;
        uch         T1;
        int         Port;
        uch         PackWin;
        uch         LinkWin;
        int         MaxFrmSizeN1;
        uch         ModVal;
        ush         No_of_SVCs;
        ush         No_of_PVCs;
        uch         Standard;
        int         Answer;
} LPB;
```

Range

The parameters are as follows:

HDLC Parameters

Baud — Specifies the bitrate when a V-type interface is being used and the Chameleon is acting as the DCE. The valid range is 50 - 64000 bps.

Encoding — Specifies the encoding scheme to use:

0 = NRZ
1 = NRZI

Net_Sub	Specifies whether the Chameleon is simulating a Network or Subscriber 0 = Network 1 = Subscriber
Interface	Specifies the physical interface that will be used: 0 = DCE (V-type) 1 = DTE (V-type) 3 = ISDN (BRI or PRI)
MaxFrmSizeN1	Specifies the value of N1 (maximum frame size) in the range 1 - 512 bytes
ReTxN2	Specifies the value of N2 in the range 1 - 255
LinkWin	For modulo 8, this specifies the window size in the range 1 - 7.
T1	Specifies the value of T1 in the range 1 - 255
Port	This parameter is used internally. It is automatically set to the port selected with x25_Setport(port).

X.25 Parameters

DC_TE	Specifies whether the Chameleon is simulating a DCE or DTE device: 0 = DCE 1 = DTE
PackWin	Specifies the packet window size: Mod 8 1 - 7 Mod 128 1 - 127
MaxPackSize	Specifies the maximum allowed length of user data in a Data packet. This is limited by the value selected for HDLC N1 (MaxFrmSizeN1): • For Mod 8, MaxPackSize must be ≤ MaxFrmSizeN1 - 3. • For Mod 128, MaxPackSize must be ≤ MaxFrmSizeN1 - 4.

ModVal — Specifies the modulo:

0 = Modulo 8
1 = Modulo 128

No_of_SVCs — Specifies the number of SVCs that can be used in the simulation session. This value must match the number of SVCs supported on the DUT. The valid range is 0 - 4095. (No_of_SVCs + No_of_PVCs < = 4095)

No_of_PVCs — Specifies the number of PVCs that can be used in the simulation session. This value must match the number of PVCs supported on the DUT. The valid range is 0 - 4095. (No_of_SVCs + No_of_PVCs < = 4095)

LCNs are assigned automatically to PVCs, beginning with the lowest LCN. If the X.25 standard is CCITT, the first PVC will be assigned to LCN 1. If the X.25 standard is TRANSPAC, the first PVC will be assigned to LCN 0.

Standard — Specifies the X.25 standard being used:

0 = CCITT X.25 standard
1 = TRANSPAC X.25 standard

Answer — Specifies how the X.25 simulator will act when an incoming Call Request is received from the DUT.

0 = Automatic. All incoming Call Requests are accepted automatically by the simulator.

1 = Manual. The application must call x25_CallAcc to accept the Call Request or x25_ClearReq to reject the Call Request.

Returns

0	Successful
-1	X.25 simulator is not started
-2	X.25 simulator is already started
-3	Queue to X.25 simulator is full; try again
-4	Timeout, no data received
-5	Parameter error
-6	No answer from Device Under Test
-8	Link not established within specified time

Sample Program

```
usr_x25Start()
{
  LCB vLCB;
  int result;

  vLPB.ModVal         = MOD8;    /* Constants are defined ix x25lib.h */
  vLPB.PackWin        = 2;
  vLPB.LinkWin        = 7;
  vLPB.MaxFrmSizeN1   = 512;
  vLPB.No_of_SVCs     = 7;
  vLPB.No_of_PVCs     = 2;
  vLPB.DC_TE          = DTE;
  vLPB.Baud           = 9600L;
  vLPB.Encoding       = NRZ;
  vLPB.Net_Sub        = NETWORK;
  vLPB.ReTxN2         = 7;
  vLPB.T1             = 3;
  vLPB.Port           = PORTA;
  vLPB.Standard       = CCITT;
  vLPB.Answer         = AUTO_ANS;

  result = x25_Start(&vLCB);
}
```

x25_Stop

Declaration

```
#include x25lib.h
int    x25_Stop()
```

Description

This function stops the X.25 simulator on the selected port. Your application must stop the X.25 simulator on all ports before the application is exited.

To stop the X.25 simulator on both Ports A and B, you must call x25_Setport() and then x25_Stop() for each port.

Note

If your application calls the C exit() function, this function is called automatically for the ports you have initialized.

Returns

0 Successful