Operations Manual
February 4, 2000

X.25 SUPPORT FOR SANGOMA CARDS

Hardware Interface and Operations Manual

LIMITED USE LICENSE AGREEMENT

Sangoma Technologies Inc. provides the computer software program contained on the medium in this package (hereinafter called the Program) and licenses its use.

THE LICENSEE SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE ATTEMPTING TO USE THIS PRODUCT. INSERTION OF ANY OF THE DISKETTES IN THIS PACKAGE INTO ANY MACHINE INDICATES THE LICENSEE'S ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF THE LICENSEE DOES NOT AGREE WITH THE TERMS AND CONDITIONS, THE LICENSEE SHOULD PROMPTLY RETURN THE PACKAGE WITHIN 15 DAYS UNUSED AND UNCOPIED IN ANY WAY SHAPE OR FORM, AND MONIES WILL BE REFUNDED.

0.2 LICENSE:

a. The purchaser of this license (hereinafter called the Licensee) is granted a personal, non-exclusive license to use the Program in accordance with the terms and conditions set out in this agreement.

b. The Program may be used only on a single computer per license granted.

c. The Licensee and the Licensee's agents and employees shall protect the confidentiality of the Program and shall not distribute or make available the Program or documentation to any third party.

d. The Licensee may copy the programs into machine readable or printed form for backup or modification purposes only in support of the Licensee's use on a single machine. The Licensee must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

e. Any portion of the Program merged into or used in conjunction with another program will continue to be subject to the terms and conditions of this agreement.

f. The Licensee may not assign or transfer the license or the program to any third party without the express prior consent of Sangoma Technologies Inc.

g. The licensee acknowledges that this license is only a limited license to use the Program and documentation, and that Sangoma Technologies Inc. retains full title to the program and documentation.

h. The Licensee shall not use, copy, modify or transfer the Program or documentation or any copy, modification or merged portion, in whole or in part, except as expressly provided for in this license. If the Licensee transfers possession of any copy, modification or merged portion of the program to a third party, the license is automatically terminated under this agreement.

0.3 TERM:

The license is effective until terminated. The licensee may terminate the license at any time by destroying the Program together with all copies, modifications and merged portion in any form. The Licensee agrees upon such termination to destroy the Program together with all copies, modifications and merged portion in any form.

0.4 LIMITED WARRANTY:

The Program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the performance of the Program is with the Licensee. Should the Program prove defective, the Licensee (and not Sangoma Technologies Inc. or an authorized dealer) shall assume the entire cost of all necessary servicing, or correction. However, Sangoma Technologies Inc. warrants the diskettes on which the Program is furnished will be free of defects in materials or workmanship under normal use for a period of 90 days from the date of delivery to the Licensee. In no event will Sangoma Technologies Inc. be liable for any damages, including incidental or consequential damages arising out of the use or inability to use the Program, even if Sangoma Technologies Inc. or an authorized dealer have been advised of the possibility
of such damages, or for any claim by any other party.

The Licensee acknowledges that the Licensee has read this agreement, understands it, and agrees to be bound by its terms and conditions. The Licensee further agrees that it is the complete and exclusive statement of the agreement between the parties and supersedes any proposal or prior agreement, oral or written, and any other communications between the parties relating to the subject matter of this agreement.

1. Introduction

This document described X.25 support on the S502, S503 and S508 cards.

The hardware/software solution handles the link and packet levels autonomously, without PC intervention. The PC accesses the system as required to send or receive data and to initiate packe level pragmatics such as call set-up and clearing.

The implementation of HDLC and X.25 on the S502 adapter corresponds to the specifications of the following ISO documents:

> ISO 7776: "Information processing systems - Data Communication - High-level data link control procedures- Description of X.25 LAPB-compatible DTE data link procedures", (first edition 1986-12-15).
>
> ISO 8208: "Information technology - Data Communications - X.25 Packet Layer Protocol for Data Terminal Equipment", (second edition 1990-03-15).

1.1 Conventions used in this manual

Programming conventions used are as follows:

> Variables described with an **0x** prefix are hexadecimal values. All other variables are decimal.
>
> For bit mapping, the **least significant** (**low**) bit is denoted as bi **0**.

This manual refers to asynchronous X.25 packets. These packets include the following types:

> Call Request/Incoming Call
> Call Accepted/Call Connected
> Clear Request/Clear Indication
> Clear Confirmation
> Reset Request/Reset Indication
> Reset Confirmati
> Restart Request/Restart Indication
> Restart Confirmati
> Interrup
> Interrupt Confirmation
> Diagnosti
> Registration Request
> Registration Confirmation

2. Hardware

2.1 General

Sangoma supports X.25 on the S502 v 3.0, S503 and S508 adapters.

2.2 S502 v 3.0

The Sangoma S502 v 3.0 SDLA adapter is compatible with an ISA or Microchannel bus. The hardware configuration is as follows:

POST setup for MCA version:

Copy the .ADF file provided with your software onto the setup disk for your MCA machine. Use the automatic setup. Use the address 360 (default) unless it clashes with one of the other cards in the machine.

Clock speed:

This is factory set by Jumper **JP3** on the ISA card and **JP1** on the MCA card. Do not change without consulting your Sangoma dealer.

I/O port address:

This is set by Jumpers **JP1** and **JP2** on the ISA card.

JP1	JP2	Selection
Jumpered	Jumpered	IO Address 250-252 (Hex)
Not Jumpered	Jumpered	IO Address 300-302 (Hex)
Jumpered	Not Jumpered	IO Address 350-352 (Hex)
Not Jumpered	Not Jumpered	IO Address 360-362 (Hex)†

† Factory default.

Cable pinout

RS232

Function	Pin #	
TxD	2	
RxD	3	
GND	7	
RTS	4	
CTS	5	
DTR	20	
DSR	6	
DCD	8	
TxC	15	
RxC	17	
BxC	24	(On board clock source)
Power (+12v)	10	

2.3 S503

This is a short 4 layer card, compatible with the ISA bus and it supports hardware interrupts as well as operating in a passive polled mode. The RS232 or V.35/X.21 interface is jumper selectable.

I/O port address:

This is set by Jumper **JP3**.

Pins 5-6	Pins 3-4	Pins 1-2	I/O Address Selection
Not Jumpered	Jumpered	Jumpered	250-252 (Hex)
Jumpered	Jumpered	Jumpered	254-256 (Hex)
Not Jumpered	Jumpered	Not Jumpered	300-302 (Hex)
Jumpered	Jumpered	Not Jumpered	304-306 (Hex)
Not Jumpered	Not Jumpered	Jumpered	350-352 (Hex)

Pins 5-6	Pins 3-4	Pins 1-2	I/O Address Selection
Jumpered	Not Jumpered	Jumpered	354-356 (Hex)
Not Jumpered	Not Jumpered	Not Jumpered	360-362 (Hex)†
Jumpered	Not Jumpered	Not Jumpered	364-366 (Hex)

† Factory default.

IRQ Selection

The optional IRQ is set using **JP2**.

Pins 1-2	Pins 3-4	Pins 5-6	Pins 7-8	Pins 9-10	Selection
In	Out	Out	Out	Out	IRQ 2
Out	In	Out	Out	Out	IRQ 3
Out	Out	In	Out	Out	IRQ 4
Out	Out	Out	In	Out	IRQ 5
Out	Out	Out	Out	In	IRQ 7†

† Factory default.

Interface Level Selection

This is set by Jumper **JP3**.

Pins 9-10	Interface Level
Jumpered	RS-232
Not Jumpered	V.35

2.4 S503 and S508 Cable Pinouts

RS232

Pin #	Function	
2	TxD	
3	RxD	
7	GND	
4	RTS	
5	CTS	
20	DTR	
6	DSR	
8	DCD	
15	TxC	
17	RxC	
24	BxC	(On board clock source)

V.35/X.21

Pin #	Function
4	RTS
5	CTS
6	DSR
7	GND
8	DCD
10	TxA
9	TxB
12	RxA
11	RxB
19	Tx Clock A
20	DTR (V10 signal)
13	DTRA (V11 signal)
14	DTRB (V11 signal)
21	Tx Clock B
22	RI
23	Rx Clock A
25	Rx Clock B
18	Aux. Clock A (On board clock source)

Pin #	Function
16	Aux. Clock B (On board clock source)

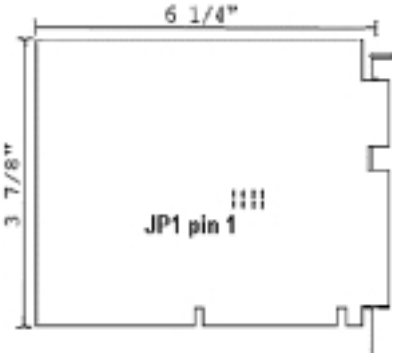
2.5 S503 and S508 Cable Pinouts

S508 I/O port Jumper Settings

Table 3-1 **JP1 - I/O Address Selection**

JP1-1	JP1-2	JP1-3	Selection
CLOSED	CLOSED	CLOSED	I/O Address 250-253 (Hex)
OPEN	CLOSED	CLOSED	I/O Address 270-273 (Hex)
CLOSED	OPEN	CLOSED	I/O Address 280-283 (Hex)
OPEN	OPEN	CLOSED	I/O Address 300-303 (Hex)
CLOSED	CLOSED	OPEN	I/O Address 350-353 (Hex)
OPEN	CLOSED	OPEN	I/O Address 360-363 (Hex)†
CLOSED	OPEN	OPEN	I/O Address 380-383 (Hex)
OPEN	OPEN	OPEN	I/O Address 390-393 (Hex)

† Factory default.



3. Software modules

3.1 General

This manual describes the hardware interfaces to the card that are applicable to any operating system (Windows, Windows NT, OS/2, Linux, Unix etc.) In particular, for simple MS-DOS implementations, we provide two interfaces which may be used to interface with the X.25 code running on the card hardware:

1. The shared memory interface where the SDLA card is operated by reading and writing structures to positions in the shared memory window. Usage of thi interface is described in a later section.

2. Under PC-DOS, XIP.COM may be used to provide a simpl interface to avoid the complexities of direct memory accesses. The application interfaces with the XIP by means of a Software Interrupt and the XIP, in turn, reads and writes to the shared memory areas. Usage of this interface is described later.

Software support for X.25 consists of the following modules:

XLOAD.EXE loads the X.25 microcode onto the SDLA adapter, configures it, runs through a self test and starts the program.

X25.502 is the run time X.25 downloadable module loaded by XLOAD.EXE.

SDLA_TST.502 is a module used for testing the SDLA hardware before loading X25.502.

X25.SDL is a file defining the HDLC and X.25 configuration parameters.

XIP.COM is a PC-DOS TSR program which provides a simpl method of interfacing with the adapter by means of a Software Interrupt.

X25_TEST.EXE is a high-level interface to the X.25 code which allows the user to perform individual interface commands.

3.2 XLOAD.EXE

XLOAD.EXE is a PC-DOS program and has the following command line syntax:

XLOAD -c<CODE> -f<CONFIG>

where:

CODE	is the name of the file containing the SDLA executable code for the X.25 implementation, e.g. X25.502.
CONFIG	is the name of the file containing the X.25 and HDLC configuration information, e.g. X25.SDL

Example:

XLOAD -c\X25\X25.502 -f\X25\X25.SDL

XLOAD will perform a system test, read the CODE and CONFIG files and load and configure the adapter. If XLOAD does not execute successfully, an error message will be displayed and an exit code will be returned. XLOAD may also display configuration warning messages. A description of these messages and corresponding exit codes are described in Section "Error Messages".

3.3 X25.502

This is the X.25 support code which is loaded onto the card and establishes the link and packe level communications.

X25.502 is **NOT** a PC-DOS program and is not executable under DOS.

3.4 X25.SDL

This file contains the X.25 code configuration data. X25.SDL is an ASCII text file and may be edited using any standard editor.

The default parameters are as follows:

```
io_port                 = 0x360
mem_segment         = 0xD000
mem_window              = 0
line_speed              = 9600
DTE                     = 1
T1                      = 1
```

T2	= 0
T4	= 10
N2	= 10
k	= 7
auto_modem_err_fixup	= 1
auto_HDLC	= 1
HDLC_config_options	= 0x01
packet_window	= 2
default_packet_size	= 128
maximum_packet_size	= 128
lowest_PVC	= 0
highest_PVC	= 0
lowest_incomming_channe	= 0
highest_incomming_channe	= 0
lowest_two_way_channe	= 1
highest_two_way_channe	= 3
lowest_outgoing_channe	= 0
highest_outgoing_channe	= 0
X25_config_options	= 0x00
X25_response_options	= 0x00
genl_facilities_supported_1	= 0x0000
genl_facilities_supported_2	= 0x0000
CCITT_facilities_supported	= 0x0000
non_X25_facilities_supported	= 0x0000
CCITT_compatibility	= 1988
T10_T20	= 30
T11_T21	= 30
T12_T22	= 30
T13_T23	= 30
T16_T26	= 30
T28	= 30
R10_R20	= 5
R12_R22	= 5
R13_R23	= 5

io_port

This corresponds to the base port address as set by the jumpers on the SDLA adapter.

mem_segment

This is the 64K PC memory segment in which the shared memory window will reside. Valid values are 0xA000, 0xC000, 0xD000 and 0xE000.

mem_window

This defines the 8k window of the PC memory segment which the PC sees as being occupied by the SDLA card. Valid values and their meanings are as follows:

Value	PC memory window
0	0000H to 1FFFH
2	2000H to 3FFFH
4	4000H to 5FFFH
6	6000H to 7FFFH
8	8000H to 9FFFH
A	A000H to BFFFH
C	C000H to DFFFH

line_speed

The X.25 link normally takes the line clocking from the modem. However, the cards w provide clocking signals on the pins marked Aux. Clock.. This means that in a local attach or test situation, there is no need for the expense or complexity of a modem eliminator. Valid values are :

Value (bps)	Actual Line Speed (bps)	Comments
0	----	No clocking signal provided
1200	1202	
2400	2405	
4800	4760	
9600	9727	
19200	18643	
38400	37286	
45000	44744	
56000	55930	
64000	55930	
74447	74573	
112000	111860	
128000	111860	

DTE

Defines the station's link level and packet level configuration and should be set t **1** for a **DTE** and **0** for a **DCE**.

T1

The period timer T1 which is used for various link level retransmission and recovery procedures.

Valid values are **1** to **30** seconds and a typical value is **3** seconds.

T2

T2 is the amount of time available at the station before an acknowledging frame must be initiated in order to ensure its receipt by the remote station, prior to timer T1 running ou at that station.

Valid values are **0** to **29** seconds and is generally set to **0**.
This value must always be less than T1.

T4

Once the HDLC link is in the asynchronous balanced mode (ABM) information transfer phase, there is no need for any HDLC Supervisory frame traffic to occur between the two devices on the link if no Information frames are being transmitted.

The **T4** parameter is used to allow the Sangoma HDLC code to issue a link leve Supervisory frame (and to elicit a response from the remote device) periodically during this 'quiescent' ABM link phase.

If this parameter is set to **0**, no unnecessary Supervisory frames will be generated. Otherwise a Supervisory frame will be issued at a period o

T4 x T1 (seconds)

For example, if the T1 timer is 3 seconds and T4 is 10, a Supervisory frame will be sent every 30 seconds.

Valid values for this parameter are **0** to **240**.

N2

This is the maximum number of transmissions and retransmissions of an HDLC frame a T1 intervals before a state change occurs.

Valid values are **1** to **30** and a typical value is **10**.

k

This parameter defines the size of the frame level window, i.e. the maximum number o sequentially numbered Information frames that may be unacknowledged at any given time.

Valid values are **1** to **7** and a typical value is **7**.

auto_modem_err_fixup

This parameter controls the reporting of modem failures to the application. If this function is enabled, then a particular modem error will only be reported once to the application.

Otherwise, this error must be manually cleared by using the **SET_GLOBAL_VARIABLES** command.

Valid entries are 0 and 1, 1 enables the automatic error correction.

Note that if **auto_HDLC** is enabled, then this parameter is also automatically enabled.

auto_HDLC

If this parameter is enabled, then the link level connection is automatically established and the application need only deal with packet level pragmatics such as call establishment and clearing.

The **auto_HDLC** value should only be set to **0** when the user wishes to fully control the set-up and disconnection of the HDLC link.

In general, this parameter should be set t **1** for X.25 code usage.

HDLC_config_options

This value describes the configuration options for the Sangoma HDLC code and the bi settings are as follows:

bit 0 - used for **link setup** control.

If this bit is set to **0**, then this station will issue SABM frames when link setup occurs. If set to **1**, then this station will not issues SABMs, but will reac appropriately to SABMs issued by the remote station.

bit 1 - If this bit is reset, then all Information frames aborted during transmission will be immediately re-transmitted. If this bit is set, then no automatic recovery on abort will occur and the link level protocol will be used to trigger Information frame re-transmissions.

bits 2-7 reserved.

packet_window

This parameter defines the default window for each direction of transmission at the DTE/DCE interface. Note that an optional window size may be selected for a period o time by using the flow control negotiation facility for SVCs or by manually configuring PVCs through the **CONFIGURE_PVC** command.

Valid values are **1** to **7**, and a typical value is **2**.

default_packet_siz

This parameter defines the default number of octets in the data field of a Data packet.

Note that an optional user data field length may be selected for a period of time by using the flow control negotiation facility for SVCs or by manually configuring PVCs through the **CONFIGURE_PVC** command.

Valid values are **16**, **32**, **64**, **128**, **256**, **512** and **1024** bytes.

maximum_packet_size

This parameter defines the maximum number of octets in the data field of a Data packe and is used to limit the data field length selected during flow control negotiation.

Valid values are **16**, **32**, **64**, **128**, **256**, **512** and **1024** bytes,
but this parameter must be greater than or equal to the **default_packet_siz** .

lowest_PVC
highest_PVC
lowest_incomming_channel
highest_incomming_channel
lowest_two_way_channel
highest_two_way_channel
lowest_outgoing_channel
highest_outgoing_channel

These parameters are used to define the logical channel configuration as assigned in agreement with the administration at the time of subscription to the X.25 service.

Valid values are between **0** and **4095**, with **0** indicating that no logical channels are assigned to that particular channel configuration.

X25_config_options

This value describes the configuration options for the Sangoma X.25 code and the bit settings are as follows:

bit **0** used for Registration pragmatics.

If this bit is set to **0**, then Registration pragmatics are not supported.

If this bit is set to **1**, then Registration pragmatics are supported.

bit **1** used to control the issuing of Diagnostic packets when the station is configured as a DCE.

If this bit is set to **0**, then Diagnostic packets will be issued by a DCE station in response to error conditions.

If this bit is set to **1**, then Diagnostic packets will not be issued.

Bit **2** used to control the issuing of a Restart request/Indication once the Data Link Layer has completed its initialization procedures.

If this bit is set to **0**, then a Restart Request/Indication will automatically be issued.

If this bit is set to 1, then no Restart Request/Indication will be issued on link initialization.

bit **3** controls whether or not diagnostic fields are to be included in Clear, Reset and Restart Request/Indication packets. Note that this parameter pertains to both outgoing and incoming packets.

If this bit is set to **0**, then diagnostic fields are always included in the above mentioned packets.

If this bit is set to **1**, then diagnostic fields are not included.

bit **4** controls the usage of D-bit procedures.

If this bit is set to **0** then D-bit usage may be established during virtual call setup on a per-logical-channel basis and D-bit pragmatics may be used on all Permanent Virtual Circuits.

If this bit is set to **1** then D-bit usage will not be permitted in call setup procedures and Data packets received with the D-bit set will be treated as an error condition.

bit **5** this bit controls the use of Flow Control Parameter Negotiation facilities and only has meaning if this particular facility has been enabled (see **genl_facilities_supported_1**).

If this bit is set to **1** then both packet and window size Flow Control Parameter Negotiation facilities will always be included in outgoing Call Request/Incoming Call and Call Accepted/Connected packets, even if the application has not passed these facilities in the **PLACE_CALL** and **ACCEPT_CALL** commands.

If this bit is set to **0** then the application is responsible for inserting these facilities when using the **PLACE_CALL** and **ACCEPT_CALL** commands.

bit **6** controls the inclusion of the Address Length and Facility Length fields in Call Accepted/Connected packets. Note that this parameter pertains to both outgoing and incoming packets.

If this bit is set to **0**, then the Call Accepted/Connected packets always include the Address Length and Facility Length fields, even if the Address and Facility fields are not included in the packet.

If this bit is set to **1**, then the Call Accepted/Connected packets do not include the Address Length and Facility Length fields, unless the Address and Facility fields are included in the packet.

bit **7** controls which facilities are permitted in incoming packets.

If this bit is set to **0**, then only the facilities included in the **genl_facilities_supported_1**, **genl_facilities_supported_2** and **CCITT_facilities_supported** parameters will be accepted in incoming packets - any other facility will invoke an error procedure.

If this bit is set to **1**, then all facilities will be accepted and passed to the application for processing.

response_options

This parameter defines how the Sangoma X.25 code will respond to various incoming packets. The bit settings are as follows:

bit **0** used for **flow control**.

If this bit is set to **0**, then an RR packet will only be issued in response to an incoming Data packet if the packet window is full (or if the D-bit is set in the received Data packet).

If this bit is set to **1**, then an RR packet will be issued in response to each incoming Data packet regardless of whether or not the packet window is full.

bit **1** used to define the response to incoming **Clear Indication**/**Request** packets.

If set to **0**, a Clear Confirmation packet will be automatically issued. If set to **1**, then the application must issue the appropriate response.

bit **2** used to define the response to incoming **Reset Indication**/**Request** packets.

If set to **0**, a Reset Confirmation packet will be automatically issued. If set to **1**, then the application must issue the appropriate response.

bit **3** used to define the response to incoming **Restart Indication**/**Request** packets.

If set to **0**, a Restart Confirmation packet will be automatically issued. If set to **1**, then the application must issue the appropriate response.

bit **4** used to define the response to incoming **Interrupt** packets.

If set to **0**, an Interrupt Confirmation packet will be automatically issued. I set to **1**, then the application must issue the appropriate response.

Note that the application will be informed that an asynchronous packet has been received, whether the response is automatic or not.

genl_facilities_supported_1
genl_facilities_supported_2

These parameters describe the general facilities which are supported during Virtual Ca establishment.

Each bit listed below represents a facility. If the bit is set to **1**, then the facility is supported. If the bit is set to **0**, then the facility is not supported and received packets which include this facility will invoke the appropriate error procedure.

The bit settings for **genl_facilities_supported_1** are as follows:

bit **0** flow control parameter (packet and window size) negotiation.

bit **1** throughput class negotiation.

bit **2** reverse charging.

bit **3** fast select (extended packet format).

bit **4** NUI selection.

bit **5** closed user group selection, basic format.

bit **6** closed user group selection, extended format.

bit **7** closed user group with outgoing access selection, basic format.

bit **8** closed user group with outgoing access selection, extended format.

bit **9** bilateral closed user group selection.

bit **10** RPOA selection, basic format

bit **11** RPOA selection, extended format.

bit **12** call deflection selection.

bit **13** call redirection or deflection notification.

bit **14** called line address modified notification.

bit **15** transit delay selection and indication.

The bit settings for **genl_facilities_supported_2** are as follows:

bit **0** charging information - requesting service.

bit **1** charging information - indicating monetary unit

bit **2** charging information - indicating segment count.

bit **3** charging information - indicating call duration.

bits
4-15 reserved.

CCITT_facilities_supported

These parameters describe the CCITT facilities which are supported during Virtual Ca establishment.

Each bit listed below represents a facility. If the bit is set to **1**, then the facility is supported. If the bit is set to **0**, then the facility is not supported and received packets which include this facility will invoke the appropriate error procedure.

The bit settings for are as follows:

bit **0** calling address extension.

bit **1** called address extension.

bit **2** minimum throughput class negotiation.

bit **3** end-to-end transit delay negotiation.

bit **4** priority.

bit **5** protection.

bit **6** expedited data negotiation.

bits
7-15 reserved.

non_X25_facilities_supported

These parameters describe non-CCITT facilities which are supported during Virtual Ca establishment.

The bit settings for are as follows:

bits
0-15 reserved.

CCITT_compatibility

The packet layer procedures implemented in the Sangoma X.25 code are compatible with the 1988 CCITT Recommendation X.25.

This parameter has been included for DTEs needing to operate with the 1980 or 1984 versions of Recommendation X.25 and may be set to **1988**, **1984** or **1980**.

T10_T20
T11_T21
T12_T22
T13_T23
T16_T26
T28

Timeout values for expected responses to various X.25 packets issued as follows:

T10_T20	timeout on **Restart Indication**/**Request** packets
T11_T21	timeout on **Incoming Call**/**Call Request** packets
T12_T22	timeout on **Reset Indication**/**Request** packets
T13_T23	timeout on **Clear Indication**/**Request** packets
T16_T26	timeout on **Interrupt** packets
T_28	timeout on **Registration Request** packets (DTE only)

These timeouts are in **1 second** units and valid values are between **1** and **255**.

R10_R20
R12_R22
R13_R23

Retransmission counts for various X.25 packets as follows:

R10_R20 retransmission count for **Restart Indication**/**Request** packets

R12_R22 retransmission count for **Reset Indication**/**Request** packets

R13_R23 retransmission count for **Clear Indication**/**Request** packets

Valid values for these retransmission counts are between **0** and **250**.

3.5 XIP.COM

A PC-DOS interface is provided so as to avoid the complexities of direct memory access. The application interfaces with the **XIP** by means of a Software Interrupt and the **XIP**, in turn, reads and writes to shared memory area.

XIP.COM is loaded with the following command line:

XIP -f<CONFIG> -i<CALLING INTERRUPT> -r

where:

CONFIG is the name of the file containing configuration information for the X.25 code, e.g. X25.SDL. **The configuration file named here must be the same as that listed in the command line arguments for XLOAD.**

CALLING INTERRUPT is the software interrupt used by the application program to communicate with XIP. The default interrupt is **6F hex**.

The "-r" parameter is included when removing a resident XIP from memory.

e.g.

XIP -fX25.SDL -i7A

Load the XIP with configuration file X25.SDL and use the software interrupt 7A (hex) to access the XIP.

The selected software interrupt is used to activate the XIP. When executing this call, the application must set the DX and BX registers to point to a control block.

If XIP does not execute successfully, an error message will be displayed and an exit code will be returned. A description of the error messages and corresponding exit codes are described in Section "Error Messages".

3.6 X25_TEST.EXE

X25_TEST is a high-level interface to the X.25 code which allows the user to perform individua interface commands.

If X25_TEST does not execute successfully, an error message will be displayed and an exit code returned.

4. The programmer's interface

4.1 Using the X.25 Shared Memory Interface

The SDLA card is operated by reading and writing structures to positions in the shared memory window. For details of moving the structures to/from the board, see the code example later in this document.

The application program accesses the X.25 software by completing the required parameters within the control block defined below and then setting the OPP_FLAG. The SDLA processor will carry out the defined command and then update this control block with the required return code and, if applicable, the associated data buffer, data length etc. When the command has been completed, the OPP_FLAG will once again be reset.

There are two control block areas within this shared memory window:

- the SEND control block is at offset **0x16B0** from the base address of the memory window and is used for all commands except the X25_READ and HDLC_READ commands.

- the RECEIVE control block is at offset **0x1AD0** from the base address of the memor window and is only used for the X25_READ and HDLC_READ commands.

Both these control blocks are of the same format.

The shared memory control block structure is as follows:

Parameter	Off-set	Lth	Remarks
OPP-FLAG	00H	1	A flag set by the application to inform the SDLA processor that a COMMAND is pending. This flag is in turn reset by the processor when the COMMAND has been completed.
COMMAND	01H	1	Command code.
BUFFER_LENGTH	02H	2	Length of the data buffer associated with this call.

Parameter	Off-set	Lth	Remarks
RETURN_ CODE	04H	1	Result of the previous command.
RESERVE D	05H	1	Reserved for HDLC interface.
LOGICAL_ CHANNEL	06H	2	The X.25 logical channe concerned with this command.
Q_D_M_BI TS	08H	1	The Q, D and M-bit settings in the X.25 packet to be transmitted or in the received packet.
CAUSE	09H	1	The X.25 cause field in the packet to be transmitted or in the received packet.
DIAGNOS-TIC	0AH	1	The X.25 diagnostic field in the packet to be transmitted or in the received packet.
X25_ PACKET_ TYPE	0BH	1	The X.25 packet type - used for reporting exception conditions such as asynchronous X.25 packets received.
TIME_ STAMP	0C H-0FH	4	A time stamp - used when reporting asynchronous X.25 occurrences.
DATA	10H	104 0	This is the transfer area for passing data to and from the application level.

4.2 Using the XIP TSR Interface

If you are running the card under DOS, you can use our DOS code and utilities. The application interfaces with the XIP by means of a Software Interrupt and the XIP, in turn, reads and writes to the shared memory areas. For details of using the XIP, see the code example later in this document.

The application program will access the XIP software by means of the following commands:

```
MOV BX, <data segment of control block>
MOV DX, <offset of control block>
INT <CALLING INTERRUPT>
```

The XIP software will transfer the control block (and associated data buffer, if applicable) fro the application software and carry out the defined command. It will then update this control block at the application level with the required return code and, if applicable, the associated data buffer, data length etc.

The XIP Control Block Structure is as follows:

Parameter	Off-set	Lth	Remarks
COMMAND	00H	1	Command code.
BUFFER_LENGTH	01H	2	Length of the data buffer associated with this call.
RETURN_CODE	03H	1	Result of the previous command.
RESERVED	04H	1	Reserved for HDLC interface.
LOGICAL_CHANNEL	05H	2	The X.25 logical channe concerned with this command.
Q_D_M_BITS	07H	1	The Q, D and M-bit settings in the X.25 packet to be transmitted or in the received packet.
CAUSE	08H	1	The X.25 cause field in the packet to be transmitted or in the received packet.

Parameter	Off-set	Lth	Remarks
DIAGNOS-TIC	09H	1	The X.25 diagnostic field in the packet to be transmitted or in the received packet.
X25_ PACKET_ TYPE	0AH	1	The X.25 packet type - used for reporting exception conditions such as asynchronous X.25 packets received.
TIME_ STAMP	0B H-0EH	4	A time stamp - used when reporting asynchronous X.25 occurrences.
RESERVE D	0FH	1	
DATA	10H	104 0	This is the transfer area for passing data to and from the application level.

5. COMMAND Codes

There are two levels of commands available when using the X.25 interface - HDLC and X.25 level commands. A limited number of the HDLC level functions are described in this document, as they may be useful in debugging any link level problems which may occur. Users wishing to fully control the HDLC link (or to interface at the HDLC level) should contact their Sangoma Technologies representative for additional interface documentation.

The valid commands are:

5.1 HDLC-Level commands

0x06	LINK_STATUS
0x07	READ_HDLC_STATISTICS
0x08	FLUSH_HDLC_STATISTICS
0x09	READ_COMMS_ERR_STATS
0x0A	FLUSH_COMMS_ERR_STATS
0x0C	READ_MODEM_STATUS
0x0B	SET_GLOBAL_VARIABLES
0x15	READ_CODE_VERSION
0x14	OPERATE_DATALINE_MONITOR
0x16	READ_TRACE_DATA

5.2 X.25-Level commands

0x22	X25_READ
0x23	X25_WRITE
0x30	PLACE_CALL
0x31	ACCEPT_CALL
0x32	CLEAR_CALL
0x33	CONFIRM_CLEAR
0x34	RESET_LOGICAL_CHANNEL
0x35	CONFIRM_RESET
0x36	RESTART
0x37	CONFIRM_RESTART
0x38	INTERRUPT
0x39	CONFIRM_INTERRUPT
0x3A	REGISTRATION_REQUEST
0x3B	REGISTRATION_CONFIRMATION
0x40	IS_DATA_AVAILABLE
0x41	INCOMMING_CALL_CONTROL
0x42	CONFIGURE_PVC
0x43	READ_ACTIVE_CHANNELS
0x44	READ_CHANNEL_CONFIGURATION
0x45	FLUSH_X25_DATA_BUFFERS
0x46	READ_X25_HISTORY_TABLE
0x47	X25_HISTORY_TABLE_CONTROL
0x48	READ_TX_D_BIT_STATUS
0x49	READ_X25_STATISTICS
0x4A	FLUSH_X25_STATISTICS
0x50	READ_HDLC_X25_CONFIGURATION
0x51	SET_HDLC_X25_CONFIGURATION

5.3 LINK_STATUS (0x06)

This command returns the HDLC status of the link, the number of Information frames queued for transmission and reception, the rotating Supervisory frame reception count and the station configuration.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The link is in the **Disconnected mode**.

0x01 The link is in the **Asynchronous Balanced Mode** (ABM), permitting data transfer.

0x06, 0x07, 0x08, 0x0A
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to 0x05 if **RETURN_CODE** of **0x00** or **0x01** is received.

DATA (valid if a **RETURN_CODE** of **0x00** or **0x01** is received):

Offset 0x00: the number of Information frames queued for transmission (0 t frames).

Offset 0x01: the number of incoming Information frames queued for reception by the X.25 protocol layer code (0 to 7 frames).

Offset 0x02: the station configuration as defined in the configuration file (or as set in the **CONFIGURE_LINK** command).

0x0 configured as a **DTE**.

0x02 configured as a **DCE**.

Offset 0x03: reserved.

Offset 0x04: the rotating Supervisory frame reception count. This count is incremented on the reception of every Supervisory frame and is useful in monitoring link activity. This value rotates between 0 and 255.

5.4 READ_HDLC_STATISTICS (0x07)

Retrieve HDLC level statistics on the operation of this station.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x22** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

Note that each value listed below is a two byte unsigned short value and is set with **the low byte first**.

Offset
0x00,0x01: number of Information frames received and made available for reception b the X.25 protocol layer code.

Offset
0x02,0x03: number of Information frames received out of sequence.

Offset
0x04, 0x05: number of Information frames received with no data field present.

Offset
0x06, 0x07: number of incoming frames discarded due to one of the following reasons:

The frame type was unsupported.

The frame was shorter than four bytes in length.

The frame was of an S or U format but had an illegal Information fiel attached.

Offset
0x08, 0x09: number of incoming frames whose data field exceeded the maximu configured data length.

Offset
0x0A, 0x0B: number of frames received with an invalid HDLC address.

Offset
0x0C, 0x0D: number of Information frames transmitted and acknowledged.

Offset
0x0E, 0x0F: number of Information frames re-transmitted.

Offset
0x10, 0x11: number of T1 timeouts.

Offset
0x12, 0x13: number of SABM frames received.

Offset
0x14, 0x15: number of DISC frames received.

Offset
0x16, 0x17: number of DM frames received.

Offset
0x18, 0x19: number of FRMR frames received.

Offset
0x1A, 0x1B: number of SABM frames transmitted.

Offset
0x1C, 0x1D: number of DISC frames transmitted.

Offset
0x1E, 0x1F: number of DM frames transmitted.

Offset
0x20, 0x21: number of FRMR frames transmitted.

5.5 FLUSH_HDLC_STATISTICS (0x08)

The current values of the variables accessed by the **READ_HDLC_STATISTICS** command are reset to zero.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

5.6 READ_COMMS_ERR_STATS (0x09)

Retrieve the communications error statistics for the link.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x0A** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

Offset 0x00: number of receiver overrun errors.

Offset 0x01: number of receiver CRC errors.

Offset 0x02: number of abort frames received.

Offset 0x03: number of frames discarded at the interrupt level due to buffering constraints.

Offset 0x04: number of abort frames transmitted.

Offset 0x05: number of transmit underruns.

Offset 0x06: number of missed transmit underrun interrupts.

Offset 0x07: reserved for later use.

Offset 0x08: number of times DCD was found to be unexpectedly inactive.

Offset 0x09: number of times CTS was found to be unexpectedly inactive.

5.7 FLUSH_COMMS_ERR_STATS (0x0A)

The current values of the variables accessed by the **READ_COMMS_ERR_STATS** command are reset to zero.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set by XIP on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

5.8 SET_GLOBAL_VARIABLES (0x0B)

Set the status of DTR.

Control Block values to be set on entry:

BUFFER_
LENGTH: Set to **0x03**.

DATA: Offset 0x00 reserved - set to 0x00.

Offset 0x01 DTR status - set to **0x01** to lower DTR.

set to **0x02** to raise DTR.

Offset 0x02 reserved - set to 0x00.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

5.9 READ_MODEM_STATUS (0x0C)

Read the current CTS and DCD status.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x01** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

Offset 0x00: The current CTS and DCD status.

If **bit 5** is set, then **CTS** is **high**.

If **bit 3** is set, then **DCD** is **high**.

5.10 READ_CODE_VERSION (0x15)

Return the code versions for both the X.25 code and the XIP.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x09** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

The code versions are of the format:

main-version.sub-versi

For example, XIP code version 2.01.

Offset 0x00 - 0x03: the version of the X.25 code running on the SDLA adapter.

Offset 0x04 - 0x07: the XIP code version (if resident).

Offset 0x08: reserved.

5.11 OPERATE_DATALINE_MONITOR (0x14)

Set the configuration of the dataline monitor or read the current monitor configuration.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x02**.

LOGICAL_
CHANNEL: If this value is set to zero, then packets will be traced for all logical channels. I this value is non-zero and a logical channel is specified, then only packets received and transmitted on this selected channel will be traced.

DATA:

Offset 0x00 miscellaneous monitor configuration bits as follows:

bit 0 if reset, then the dataline monitor is deactivated. If set, then the dataline monitor is activated.

bit 1 if set, then each traced frame will include a millisecond time stamp. Note that this time stamp rotates between 0 and 65535 milliseconds.

bit 2 if set, then the trace delay mode is activated. The trace delay mode prevents trace frames from being discarded due to limited on-board trace buffering. In effect, if the delay mode is enabled, then the actua transmission of HDLC frames is slowed down on the adapter to keep pace with the rate at which the application is reading trace frames from the board.

bit 3 if set, then X.25 Data packets will be traced.

bit 4 if set, then X.25 Supervisory packets will be traced.

bit 5 if set, then X.25 Asynchronous packets will be traced.

bit 6 if set, then all HDLC level frames will be traced.

bit 7 if set, then the current trace configuration will be returned to the application. Note that the trace configuration will not be set if this bit is enabled.

Offset 0x01 the trace deactivation timer. This timer is used in conjunction with the trace delay mode and automatically deactivates the trace in the case where the application does not read trace frames from the board at least once per defined deacitivation period.

This value of this timer may be set from 1 to 8 seconds.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x01** if is the current trace configuration was requested.

DATA (valid if the current trace configuration was requested):
Offset 0x00 miscellaneous monitor configuration bits as defined above.

5.12 READ_TRACE_DATA (0x16)

Read a trace frame from the SDLA adapter.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE:
- 0x00 The action has been performed successfully and trace data for pick up in the DATA area of this interface structure.
- 0x01 No trace frames are currently available for the application.
- 0x02 The trace has not been activated.
- 0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to the length of the received trace structure if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

The received trace structure in the following format:

Offset
0x00-0x01: the length of the actual trace data.

Offset 0x02: the type of trace is indicated in the low 2 bits as follows:

- 0x00 the trace is of a received frame
- 0x01 the trace is of a transmitted frame
- 0x02 the trace is of an HDLC error frame, in which case the trace byte is decoded as follows:
  - 0x12 the trace is of a received abort frame

0x22 the trace is of a received frame with a CRC error

0x32 the trace is of a received frame with an overrun error

0x42 the trace is of a frame received of excessive length.

0x72 the trace is of an aborted transmitted frame (missed transmit interrupt)

0x82 the trace is of an aborted transmitted frame (missed transmit underrun interrupt)

Offset 0x03: the number of trace packets discarded since the last trace frame was passed to the application.

Offset 0x04-0x08: valid only if the XIP interface is being used, and is the time at which this transaction occurred (in binary coded decimal). The format of this four-byte time stamp is as follows:

Offset	Parameter
0x04	Month
0x05	Day
0x06	Hours
0x07	Minutes
0x08	Seconds

Offset 0x09-0x0A: the millisecond time stamp for the frame (valid only if time stamping has been activated). This time stamp rotates between 0 and 65535 milliseconds.

Offset 0x0B-0xNN: the actual contents of the frame.

Note that this is the complete HDLC frame including the address field, the control field and the two CRC bytes. The HDLC flags are not included and each CRC byte for a transmitted frame is represented by the character 0xFF.

5.13 X25_READ (022)

Receive an X.25 Data packet from the network on a selected logical channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The selected logical channel.

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE:

0x00 Data has been received and is available for pick up in the DATA area o this interface structure.

0x02 The link is not currently in the ABM and data transfer is not possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The selected logical channel is not in the data transfer state and therefore no Data packets may be transferred.

0x33 No data is currently available on the selected logical channel.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH (valid if a **RETURN_CODE** of **0x00** is received):
The length of the data message received.

Q_D_M_BITS (valid if **RETURN_CODE** of **0x00** is received):
The status of the Q, D and M bits (bits 2, 1 and 0 respectively) in the received Data packet .

DATA (valid if a **RETURN_CODE** of **0x00** is received):
The actual data that has been received on the selected logical channel.

5.14 X25_WRITE (0x23)

Send a data message to the card for onward transmission to the network on the selected logica channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The selected logical channel.

BUFFER-
LENGTH: The length of the data packet to be transmitted. The maximum data length is **1024** bytes and is dependent on the **maximum_packet_size** defined in the configuration file OR the packet size negotiated for the particular Switched Virtual Circuit OR the packet size set for a Permanent Virtual Circuit by means of a **CONFIGURE_PVC** command.

Q_D_M_BITS: The status of the Q, D and M-bits (bits 2, 1 and 0 respectively) in the Data packet to be transmitted. In general, this parameter should be set to **0x00**.

DATA: The data to be transmitted.

Control Block values set on return:

RETURN_
CODE: 0x00 The data has been queued for transmission.

0x02 The link is not currently in the ABM and data transfer is not possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The selected logical channel is not in the data transfer state and therefore no data may be transferred.

0x32 The length of the data message to be transmitted exceeded the maximu permitted data length for that logical channel. The data was **not** transmitted.

0x33 The data was not queued due to the fact tha

a) the packet window for the logical channel window is closed

OR

b) all the transmit buffers are currently occupied.

The application is attempting to send data to the X.25 network at a rate faster than the network is able to receive the data. **No data was sent to the S502 card and this same data should be re-sent in its entirety.**

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.15 PLACE_CALL (0x30)

This command causes an X.25 Call Request/Incoming Call packet to be issued on the nex available switched virtual circuit.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to the length of the call argument string.

Q_D_M_BITS: Bit 1 is set if the D-bit should be set in the call packet. Note that if D-bit pragmatics are to be supported, then the appropriate bit must be reset in the **X25_config_options** configuration parameter.

DATA: The call data is entered as an ASCII string in the following format -

-d[called DTE address] -s[calling DTE address]

-f[facilities] -u[user data]

The call data is entered as a series of ASCII strings, for example:

-d81200012 -f0101 -uC2

(Call DTE 81200012 with facility number 1 set to 1 and user data of 0xC2).

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The call packet was not transmitted as there was no logical channe available on which this outgoing call could be placed.

0x32 The call arguments are too long.

0x33 The call was not placed as the asynchronous buffer was in use.

0x38 The D-bit was set in the Call packet, but D-bit pragmatics are not supported according the **X25_config_options** configuration parameter.

0x39 A facility included in the Call packet is not supported according the **genl_facilities_supported** configuration parameter.

0x3A The call arguments are erroneous as no destination address was included.

0x3B An odd number of bytes was included in the User Data Fields. This field is reformatted in BCD, and therefore an even number of characters must be processed

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

LOGICAL_CHANNEL (valid if a **RETURN_CODE** of **0x00** is received):
The logical channel on which the Call Request/Incoming Call was issued.

5.16 ACCEPT_CALL (0x31)

This command causes an X.25 Call Accepted/Connected packet to be issued in response to a Ca Request/Incoming Call packet previously received.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the Call Request/Incoming Call was logged (and on which this packet should be issued).

BUFFER-
LENGTH: Set to the length of the Call Accepted argument string (normally set to **0x00**).

Q_D_M_BITS: Bit 1 is set if the D-bit should be set in the Call Accepted packet (in response to the D-bit being set in the received Call Request/Incoming Ca packet). In general, this parameter is set to **0x00**.

DATA: The call data (if required) is entered as an ASCII string in the same format as tha for the **PLACE_CALL** command.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x31 The Call Accepted/Connected packet was not transmitted as no Ca Request/ Incoming Call was logged on the selected logical channel.

0x32 The packet arguments included are too long.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x34 The incoming Call Request/Incoming Call included the Fast Select Facility with restriction on response. Therefore, this call may not be accepted, bu must be cleared.

0x38 The D-bit was set in the Call Accepted/Connected packet, but D-bit pragmatics are not supported according the **X25_config_options**

configuration parameter, **OR**, the incoming Call Request/Incoming Call did not have the D-bit set

0x39 A facility included in the packet is not supported according the **genl_facilities_supported** configuration parameter.

0x3B An odd number of bytes was included in the User Data Fields. This field is reformatted in BCD, and therefore an even number of characters must be processed.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.17 CLEAR_CALL (0x32)

This command causes an X.25 Clear Request/Indication packet to be issued on the selected logical channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the call should be cleared.

BUFFER-
LENGTH: Set to the length of the clear argument string (normally set to **0x00**).

Q_D_M_BITS: Controls the flushing of the X25 data buffers. If set to **0x00**, then all receive and transmit buffers (for the selected logical channel) will be flushed before the Clear packet is issued. If set to **0x80**, then the Clear packet w **not** be issued if there are any queued data buffers on this logica channel. Instead, a return code o **0x35** will be passed to the application to indicate the existence of this queued data. The **X25_READ** or the **FLUSH_X25_DATA_BUFFERS** commands may be used as desired to clear the data buffers.

CAUSE: The X.25 cause field.

DIAGNOSTIC: The X.25 diagnostic field.

DATA: The Clear data (if required) is entered as an ASCII string in the same format as that for the **PLACE_CALL** command.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The Clear Request/Indication packet was not transmitted as the selected logical channel is not configured as an SVC.

0x31 The Clear Request/Indication packet was not transmitted as the state of the selected logical channel is not valid for the issuing of such a packet.

0x32 The packet arguments are too long.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x35 The packet was not issued as there is queued data on the selected logica channel (see **Q_D_M_BITS** above).

0x39 A facility included in the packet is not supported according the **genl_facilities_supported** configuration parameter.

0x3B An odd number of bytes was included in the User Data Fields. This field is reformatted in BCD, and therefore an even number of characters must be processed.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.18 CONFIRM_CLEAR (0x33)

This command causes an X.25 Clear Confirmation packet to be issued on the selected logica channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the Call Confirmation packet should be issued.

BUFFER-
LENGTH: Set to the length of the Clear Confirmation argument string (normally set to **0x00**).

DATA: The data (if required) is entered as an ASCII string in the same format as that for the **PLACE_CALL** command.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The Clear Confirmation packet was not transmitted as the selected logica channel is not configured as an SVC.

0x31 The Clear Confirmation packet was not transmitted as no Clear Request/Indication packet was previously received on the selected logica channel.

0x32 The packet arguments are too long.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x37 User data field is not permitted.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40,0x41,0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.19 RESET_LOGICAL_CHANNEL (0x34)

This command causes an X.25 Reset Request/Indication packet to be issued on the selected logical channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the call should be reset.

BUFFER-
LENGTH: Set to **0x00**.

Q_D_M_BITS: Controls the flushing of the X25 data buffers.

If set to **0x00**, then all receive and transmit buffers (for the selected logica channel) will be flushed before the Reset packet is issued.

If set to **0x80**, then the Reset packet w **not** be issued if there are any queued data buffers on this logical channel. Instead, a return code o **0x35** will be passed to the application to indicate the existence of this queued data. The **X25_READ** or the **FLUSH_X25_DATA_BUFFERS** commands may be used as desired to clear the data buffers.

CAUSE: The X.25 cause field.

DIAGNOSTIC: The X.25 diagnostic field.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The Reset Request/Indication packet was not transmitted as the state of the selected logical channel is not valid for the issuing of such a packet.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x35 The packet was not issued as there is queued data on the selected logica channel (see **Q_D_M_BITS** above).

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.20 CONFIRM_RESET (0x35)

This command causes an X.25 Reset Confirmation packet to be issued on the selected logica channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the Reset Confirmation packet should be issued.

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The Reset Confirmation packet was not transmitted as no Reset Request/Indication packet was previously received on the selected logica channel.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.21 RESTART (0x36)

This command causes an X.25 Restart Request/Indication packet to be issued.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Q_D_M_BITS: Controls the flushing of the X25 data buffers. If set to **0x00**, then **all** receive and transmit buffers will be flushed before the Restart packet is issued. If set to **0x80**, then the Restart packet w **not** be issued if there are any queued data buffers for **any** logical channel. Instead, a return code o **0x35** will be passed to the application to indicate the existence of this queued data. The **X25_READ** or the **FLUSH_X25_DATA_BUFFERS** commands may be used as desired to clear the data buffers.

CAUSE: The X.25 cause field.

DIAGNOSTIC: The X.25 diagnostic field.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x35 The packet was not issued as there is queued X.25 data (see **Q_D_M_BITS** above).

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.22 CONFIRM_RESTART (0x37)

This command causes an X.25 Restart Confirmation packet to be issued.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x31 The Restart Confirmation packet was not transmitted as no Restart Request/Indication packet was previously received.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.23 INTERRUPT (0x38)

Send an Interrupt packet to the network on the selected logical channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The selected logical channel.

BUFFER-
LENGTH: The length of the Interrupt user data to be transmitted. The maximum data length is **32** bytes if configured to support CCITT 1984/88, or **1** byte if CCITT 1980 compatibility is required.

DATA: The Interrupt data to be transmitted.

Control Block values set on return:

RETURN_
CODE: 0x00 The Interrupt packet has been queued for transmission.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The selected logical channel is not in the data transfer state and therefore no Interrupt packets may be transferred.

0x32 The length of the Interrupt data to be transmitted was zero or exceeded the maximum permitted length.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.24 CONFIRM_INTERRUPT (0x39)

This command causes an X.25 Interrupt Confirmation packet to be issued on the selected logica channel.

Control Block values to be set on entry:

LOGICAL_
CHANNEL: The logical channel on which the Interrupt Confirmation packet should be issued.

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x31 The Interrupt Confirmation packet was not transmitted as no Interrupt packet was previously received on the selected logical channel.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.25 REGISTRATION_REQUEST (0x3A)

This command causes an X.25 DTE Registration Request packet to be issued.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to the length of the Registration Request argument string.

DATA: The Registration data is entered as an ASCII string in the same format as that for the **PLACE_CALL** command.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x35 The X.25 device may not issue a Registration Request packet as it is configured as a DCE.

0x36 Registration pragmatics are not supported according to the **X25_config_options** configuration parameter.

0x3A The Registration field is invalid and the packet was not issued.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.26 REGISTRATION_CONFIRMATION (0x3B)

This command causes an X.25 DCE Registration Confirmation packet to be issued.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to the length of the Registration Confirmation argument string.

DATA: The Registration data is entered as an ASCII string in the same format as that for the **PLACE_CALL** command.

Control Block values set on return:

RETURN_
CODE:

0x00 The action has been performed successfully.

0x02 The link is not currently in the ABM and no transfer of X.25 packets is possible.

0x31 The Registration Confirmation packet was not transmitted as no Registration Request packet was previously received.

0x33 The packet was not issued as the asynchronous buffer was in use.

0x35 The X.25 device may not issue a Registration Confirmation packet as it is configured as a DTE.

0x36 Registration pragmatics are not supported according to the **X25_config_options** configuration parameter.

0x3A The Registration field is invalid and the packet was not issued.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

5.27 IS_DATA_AVAILABLE (0x40)

Interrogate the interface to find out if any X.25 Data packets are available for forwarding to the application level.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 Data is available on the logical channels listed in the data buffer area.

0x33 No Data packets are currently available for reception by the application.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Represents double the number of logical channels on which data is currently available (as each logical channel is represented by a two byte value).

DATA: A list of logical channels on which data has been received. Each channel is represented as a **two byte hexadecimal** number. **X25_READ** calls should now be made with the **LOGICAL_CHANNEL** parameter set to the value of each of the logical channels listed here.

For example, the **BUFFER_LENGTH** may be equal to 0x06 and the **DATA** area may contai

0x01 0x00 0x06 0x00 0xA1 0x03

This implies that three logical channels (1, 6 and 929 decimal) have data available.

5.28 INCOMMING_CALL_CONTROL (0x41)

Control the handling of received X.25 Call Request/Incoming Call packets.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x01**.

DATA: The call control parameter is situated at offse **0x00** in the data buffer area and is defined as follows:

Set to **0x00** all received Call Request/Incoming Call packets will be automatically **cleared**, by the issuing of a Clear Request/Indication packet.

Set to **0x01** the application is notified of received Call Request/ Incoming Ca packets and may then respond as desired by accepting or clearing the call.

Set to **0x02** all received Call Request or Incoming Call packets will be automatically **accepted**, by the issuing of a Call Accepted/Connected packet.

Note that if this command is not performed after loading the X.25 code, then, by default, the application is notified of incoming calls and these calls are **not automatically cleared or accepted**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x31 The defined call control parameter is invalid.

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

5.29 CONFIGURE_PVC (0x42)

Configure a Permanent Virtual Circuit to have a packet or window size other than the defaults defined in the X.25 configuration file.

Note - it is important that if you re-configure a permanent virtual circuit after Data packets have been transferred on this logical channel, then a Reset Request/Indication or a Restar Request/Indication should be issued so as to reset the windowing pragmatics.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x02**.

LOGICAL_
CHANNEL: The logical channel to be configured.

DATA: The parameters for the direction of incoming data are defined at offse **0x00** of the data area and those for outgoing data are at offse **0x01**.

The **window size** is defined by the **low four bits** of the hexadecimal character and may be between the values o **0x01** and **0x07**.

The **packet size** is defined by the **high four bits** of the hexadecimal character as follows:

Bit setting	Packet size
0x40	16
0x50	32
0x60	64
0x70	128
0x80	256
0x90	512
0xA0	1024

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x30 The selected logical channel is invalid (i.e., it was not included in the range of Permanent Virtual Circuits defined in the configuration file).

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

5.30 READ_ACTIVE_CHANNELS (0x43)

Interrogate the XIP to find out which logical channels are in the data transfer state i.e., a list o logical channels on which data may be transferred with **X25_READ** and **X25_WRITE** commands.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The logical channels listed in the data buffer area are in the data transfer state.

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Represents double the number of logical channels which are currently in the data transfer state (as each logical channel is represented by a two byte value). Note that if this value is set to 0 then no channels are in the data transfer state.

DATA: A list of logical channels on which data may be transferred. Each channel is represented as a **two byte hexadecimal** number.

5.31 READ_CHANNEL_CONFIGURATION (0x44)

Read the range of logical channels used for virtual calls and permanent virtual circuits, or read the current configuration of a particular logical channel.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

LOGICAL_
CHANNEL: Set to **0x00** if the application wishes to establish the range of logical channels used for virtual calls and permanent virtual circuits.

OR

Set to a particular logical channel number if the application wishes to establish the configuration (including the packet and window sizes) of that channel.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Valid if the RETURN_CODE is **0x00**.
Set to **0x10** if the selected logical channel was **0x00**, otherwise set to **0x03**.

DATA: If the selected logical channel was set to **0x00**, the returned data is as follows:

Offset 0x00, 0x01 - Lowest PVC
Offset 0x02, 0x03 - Highest PVC
Offset 0x04, 0x05 - Lowest Incoming Channe
Offset 0x06, 0x07 - Highest Incoming Channe
Offset 0x08, 0x09 - Lowest Two-way Channe
Offset 0x0A, 0x0B - Highest Two-way Channe
Offset 0x0C, 0x0D - Lowest Outgoing Channe
Offset 0x0E, 0x0F - Highest Outgoing Channe

If the selected logical channel was not set to **0x00**, the returned data is as follows:

Offset 0x00 - The channel configuration in **bits 0 to 4** as follows:

0x00 - Channel not configured
0x01 - PVC
0x03 - Incoming only
0x07 - Two-way
0x0B - Outgoing onl

Bits 5, **6** and **7** pertain to Switched Virtual Circuits only, and are as follows:

Bit **5** if set, then D bit usage has been negotiated during the call setup.

Bits **6,7**

Fast Select configuration as follows:

If bit 7 is set, then Fast Select has been requested for the logica channel.

If bit 6 is reset, then there is no restriction on response.

If bit 6 is set, then there is restriction on response.

Offset 0x01 The packet and window sizes for **transmission** defined as follows:

The low three bits (bits **0, 1, 2**) define the packet window and may be between the values o **0x01** and **0x07**.

Bits **3, 4** and **5** define the packet size as follows:

Bit setting	Packet size
0x00	16
0x08	32
0x10	64
0x18	128
0x20	256
0x28	512
0x30	1024

Offset 0x02 The packet and window sizes for **reception** defined as for transmission above.

5.32 FLUSH_X25_DATA_BUFFERS (0x45)

Flush any queued X.25 transmit and receive data buffers.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

LOGICAL_
CHANNEL: Set to **0x00** if the application wishes to flush the data buffers for **all** logica channels.

Set to a particular logical channel number if the application wishes to only flush the data buffers of the selected channel.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x06, 0x07, 0x08, 0x0A, 0x09
See Section "Notes on Return Codes" for further details.

5.33 READ_X25_HISTORY_TABLE (0x46)

Read a table listing the asynchronous X.25 transactions which have occurred.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Valid if the RETURN_CODE is **0x00**. Set to the length of the returned history data.

DATA: The X.25 history data in **11** byte records as follows:

Offset 0x00 - The direction of transmission and cause of the asynchronous transaction, defined as follows:

- 0x00 The asynchronous packet was an incoming packet.
- 0x01 The asynchronous packet was sent as a result of a command **generated by the application**, e.g., a **CLEAR_CALL** command.
- 0x02 The packet was sent as an **automatic response to an incoming asynchronous packet**.
- 0x04 The packet was issued as a result of a **protocol error**.
- 0x08 The packet was generated due to an **X.25 time out** occurring.
- 0x10 The packet was issued to reinitialize the packet level after **link level recovery** procedures.

Offset 0x01, 0x02 - The logical channel concerned with this transaction.

Offset 0x03 - The asynchronous packet type defined as follows:

0x0B - Call Request/Incoming Call
0x0F - Call Accepted/Connected
0x13 - Clear Request/Indication
0x17 - Clear Confirmation
0x1B - Reset Request/Indication
0x1F - Reset Confirmati
0x23 - Interrup
0x27 - Interrupt Confirmation
0xFB - Restart Request/Indication
0xFF - Restart Confirmation
0xF1 - Diagnosti
0xF3 - DTE Registration Request
0xF7 - DCE Registration Confirmation

Offset 0x04 - The X.25 cause field (if applicable).

Offset 0x05 - The X.25 diagnostic field (if applicable).

Offset 0x06 to 0x0A (5 bytes) - valid only if the XIP interface is being used, and is the transaction time-stamp in **binary coded decimal** (with the **month** being at offset **0x06**, the **day** being at offset **0x07**, the **hours** being at offset **0x08**, the **minutes** at offset **0x09** and the **seconds** being at offset **0x0A**.

For example, a time stamp of 0x09, 0x01, 0x15, 0x46, 0x03 represents the 1st of September (9th month) at 3:46:03 PM.

Note that **this time stamp is inserted by the XIP and is therefore dependent on the accuracy of the PC time clock**.

5.34 X25_HISTORY_TABLE_CONTROL (0x47)

Flush the table listing the asynchronous X.25 transactions which have occurred.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x01**.

DATA: The byte at offse **0x00** of the data area controls the operation of the X.25 histor table and is configured as follows:

bit 0: If **set**, the history table will be **flushed**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

5.35 READ_TX_D_BIT_STATUS (0x48)

This command is used to establish whether or not a Data packet transmitted with the D-bit set has been acknowledged by the remote station.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

LOGICAL_
CHANNEL: Set to the logical channel on which a D bit Data packet has previously been transmitted.

Control Block values set on return:

RETURN_
CODE:

0x00 No D bit acknowledgement is outstanding for the selected logical channel.

0x30 The selected logical channel is invalid (i.e., it was not included in the logical channel range defined in the configuration file).

0x33 A D bit acknowledgement is still outstanding for the selected logica channel.

0x06, 0x07, 0x08, 0x09, 0x0A
See Section "Notes on Return Codes" for further details.

5.36 READ_X25_STATISTICS (0x49)

Retrieve X.25 level statistics on the operation of this station.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER-
LENGTH: Set to **0x40** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):

Note that each value listed below is a two byte unsigned short value and is set with **the low byte first**.

Offset	Parameter Count
0x00-0x01	Restart Request/Indication packets transmitted
0x02-0x03	Restart Request/Indication packets received
0x04-0x05	Restart Confirmation packets transmitted
0x06-0x07	Restart Confirmation packets received
0x08-0x09	Reset Request/Indication packets transmitted
0x0A-0x0B	Reset Request/Indication packets received
0x0C-0x0D	Reset Confirmation packets transmitted

Offset	Parameter Count
0x0E-0x0F	Reset Confirmation packets received
0x10-0x11	Call Request/Incoming Call packets transmitted
0x12-0x13	Call Request/Incoming Call packets received
0x14-0x15	Call Accepted/Call Connected packets transmitted
0x16-0x17	Call Accepted/Call Connected packets received
0x18-0x19	Clear Request/Indication packets transmitted
0x1A-0x1B	Clear Request/Indication packets received
0x1C-0x1D	Clear Confirmation packets transmitted
0x1E-0x1F	Clear Confirmation packets received
0x20-0x21	Diagnostic packets transmitted
0x22-0x23	Diagnostic packets received
0x24-0x25	Registration Request packets transmitted
0x26-0x27	Registration Request packets received
0x28-0x29	Registration Confirmation packets transmitted
0x2A-0x2B	Registration Confirmation packets received
0x2C-0x2D	Interrupt packets transmitted
0x2E-0x2F	Interrupt packets received
0x30-0x31	Interrupt Confirmation packets transmitted
0x32-0x33	Interrupt Confirmation packets received
0x34-0x35	Data packets transmitted

Offset	Parameter Count
0x36-0x37	Data packets received
0x38-0x39	RR packets transmitted
0x3A-0x3B	RR packets received
0x3C-0x3D	RNR packets transmitted
0x3E-0x3F	RNR packets received

5.37 FLUSH_X25_STATISTICS (0x4A)

The current values of the variables accessed by the **READ_X25_STATISTICS** command are reset to zero.

Control Block values to be set on entry:

BUFFER-
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

5.38 READ_HDLC_X25_CONFIGURATION (0x50)

Read the current HDLC/X.25 configuration parameters.

Control Block values to be set on entry:

BUFFER_
LENGTH: Set to **0x00**.

Control Block values set on return:

RETURN_
CODE: 0x00 The action was performed successfully.

0x0A See Section "Notes on Return Codes" for further details.

BUFFER_
LENGTH: Set to **0x37** if a **RETURN_CODE** of **0x00** is received.

DATA (valid if a **RETURN_CODE** of **0x00** is received):
The data is in the same format as that for the SET_HDLC_X25_CONFIGURATION command.

5.39 SET_HDLC_X25_CONFIGURATION (0x51)

Set the HDLC/X.25 configuration parameters.

Note that no checking of the validity of the passed configuration data is performed by the X.25 code and the user must therefore ensure that this data is correct. Before using this command, it is recommended that the READ_HDLC_X25_CONFIGURATION command be performed to check the accuracy of the configuration structure defined in the application code.

A Restart Request/Indication packet will automatically be transmitted after the completion of the SET_HDLC_X25_CONFIGURATION command.

Control Block values to be set on entry:

BUFFER_
LENGTH: Set to **0x37**.

DATA: This area contains the configuration parameters. A detailed description of these parameters may be found in Section 3.

Offset	Size	Parameter (bytes)
0x00	1	A number corresponding to the baud rate generated by the SDLA card on pin 24 as follows:

Value	Baud Rate (bps)
0x00	External Clocking
0x01	1200
0x02	2400
0x03	4800
0x04	9600
0x05	19200
0x06	38400
0x07	45000
0x08	56000
0x09	64000
0x0A	74000
0x0B	112000
0x0C	128000

0x01	1	The period timer T1.

0x02	1	The timer T2.
0x03	1	The N2 counter.
0x04	2	The maximum length of the HDLC Information frame data field. This parameter should be set to zero, as it will be calculated fro the configured maximum packet length.
0x06	1	The size of the frame level window (k).
0x07	1	The T4 parameter.
0x08	1	The 'auto_modem_err_fixup' parameter.
0x09	1	The 'auto_HDLC' parameter.
0x0A	1	The 'HDLC_config_options'.
0x0B	1	Defines the station's link level and packet level configuration and is set to **0x01** for a **DTE** and **0x00** for a **DCE**.
0x0C	1	The X.25 packet window.
0x0D	2	The default X.25 Data field size.
0x0F	2	The maximum X.25 Data field size which may be selected during flow control negotiation.
0x11	2	The lowest PVC.
0x13	2	The highest PVC.
0x15	2	The lowest incomming channel.
0x17	2	The highest incomming channel.
0x19	2	The lowest two-way channel.
0x1B	2	The highest two-way channel.
0x1D	2	The lowest outgoing channel.
0x1F	2	The highest outgoing channel.
0x21	2	The 'X25_config_options'.

0x23	1	The 'X25_response_options'.
0x24	2	The 'genl_facilities_supported_1' parameter.
0x26	2	The 'genl_facilities_supported_2' parameter.
0x28	2	The 'CCITT_facilities_supported' parameter.
0x2A	2	The 'non_X25_facilities_supported' parameter.
0x2C	2	The compatibility version for the X.25 packet layer procedures.
0x2E	1	The T10/T20 Restart Indication/ Request timeout.
0x2F	1	The T11/T21 Incomming Call/Call Request timeout.
0x30	1	The T12/T22 Reset Indication/Request timout.
0x31	1	The T13/T23 Clear Indication/Request timeout.
0x32	1	The T16/T26 Interrupt timeout.
0x33	1	The T28 Registration Request timeout.
0x34	1	The R10/20 Restart Indication/Request retransmission count.
0x35	1	The R12/22 Reset Indication/Request retransmission count.
0x36	1	The R10/20 Clear Indication/Request retransmission count.

Control Block values set on return:

RETURN_ CODE:	0x00	The action was performed successfully.
	0x0A	See Section "Notes on Return Codes" for further details.

6. Notes on Return Codes

There are return codes common to specific commands which require additional discussion. These return codes (hexadecimal values) are:

0x05 The command used is invalid.

0x06 An Unnumbered frame (SABM, DISC, DM, UA) was unexpectedly received while the link was in the Asynchronous Balanced Mode, or a UI frame has been received.

The **BUFFER_LENGTH** will be set to **0x01** (plus the length of the UI Data field if applicable) and the byte at offse **0x00** of the XIP structure **DATA** area indicates the type of Unnumbered frame received as follows:

0x01 - a **SABM** command frame was received.

0x02 - a **DISC** command frame was received.

0x03 - a **DM** response frame was received.

0x04 - a **UI** frame was received.

0x05 - a **UA** response frame was received.

If the received frame is a **UI**, then the UI Data field is at offset 0x01 of the XIP structure **DATA** area. If the received frame is a **DISC** of a **DM**, the Sangoma HDLC code will automatically attempt to re-enter the ABM by issuing **SABMs**.

Note: the command executed when this return code was passed to the application was no performed (i.e. any returned data is invalid) and this call should be repeated.

0x07 The link is in the **Frame Reject mode**.

The **BUFFER_LENGTH** will be set to **0x05** and the interface structure **DATA** area ma be decoded as follows:

Offset 0x00 the source of the FRMR frame
0x01 - loca
0x02 - remote

Offset 0x01: the reason for the FRMR
0x01 control field invalid or not implemented.

0x03 an S or U frame received with an illegal I-field attached.
0x04 an I-frame received whose data length exceeded the defined maximum.
0x08 an invalid Nr count.

Offset
0x02-0x05 the FRMR Information field.

The Sangoma HDLC level code will automatically attempt to reset the link when FRMR responses are received.

Note: the command executed when this return code was passed to the application was not performed (i.e. any returned data is invalid) and this call should be repeated.

0x08 A **modem failure** occurred - DCD and /or CTS were found to be unexpectedly low.

The cause of this failure is returned at offse **0x00** of the XIP data buffer and may be one of the following:

0x01 - DCD was found to be unexpectedly low.
0x02 - CTS was found to be unexpectedly low.

Executing a **READ_GLOBAL_STATS** command will indicate the count of the various modem error types and the **READ_MODEM_STATUS** command will return the curren status of DCD and CTS.

Note: the command executed when this return code was passed to the application was no performed (i.e. any returned data is invalid) and this call should be repeated.

0x09 The **N2 retry limit** has been exceeded, i.e. an unnumbered frame has been issued N2 times, but the remote device has still not responded.

The **BUFFER_LENGTH** will be set to **0x01** and the byte at offse **0x00** of the XIP structure **DATA** area indicates the type of Unnumbered frame concerned with this retr limit:

0x01 - a **SABM** command retry limit occurred.

0x02 - a **DISC** command retry limit occurred.

0x0A A **SDLA card timeout** occurred.

The XIP call to the board was not processed in the specified time, indicating a hardware failure. This may be caused by a clash in the shared memory window occupied by the card. Make sure that the 8k shared memory window specified in the configuration is reserved for the S502 card. If this error persists, contact your Sangoma dealer.

0x40 An asynchronous X.25 packet was received.

On reception of this return code, the following XIP structure parameters are significant:

LOGICAL_CHANNEL - the logical channel on which the packet was received.

BUFFER_LENGTH - the length of the data associated with this asynchronous packet.

Q_D_M_BITS - the Q and D-bit settings in the received packet.

CAUSE, **DIAGNOSTIC** - the X.25 cause and diagnostic fields associated with the incoming packet (if applicable).

X25_PACKET_TYPE - the received packet type.
If the high bit in this parameter is set, then the Sangoma X.25 code has automatically responded to the incoming packet (according to the **response_options** in the configuration file) and there is no need for the application to issue an appropriate asynchronous response.

The **high bit should be masked** and then the X25_PACKET_TYPE is decoded as follows:

0x02 Clear Request/Indication
0x04 Reset Request/Indication
0x08 Restart Request/Indication
0x10 Interrup
0x20 DTE Registration Request
0x30 Call Request/Incoming ca
0x31 Call Accepted/Connected
0x32 Clear Confirmation
0x33 Reset Confirmati
0x34 Restart Confirmati
0x35 Interrupt Confirmation
0x36 DCE Registration Confirmation
0x37 Diagnosti
0x38 Call Request/Incoming Call (**Call automatically cleared**)
0x39 Call Request/Incoming Call (**Call automatically accepted**)

TIME_STAMP is valid only if the XIP interface is being used, and is the time a which this transaction occurred (in binary coded decimal). The format of this four-byte time stamp is as follows:

Offset within time stamp definition	Parameter
0x00	Day
0x01	Hours
0x02	Minutes
0x03	Seconds

DATA is the data associated with this asynchronous packet (if the BUFFER_LENGTH is non-zero).

Note that for a Diagnostic packet, the diagnostic code reflects the diagnostic code transmitted in the packet. The data area contains both the diagnostic code and the diagnostic explanation fields.

This data is in an ASCII string format as follows:

-d[called DTE address] -s[calling DTE address] -f[facilities] -u[user data]

For example, a received Call Request/Incomming Call packet may be passed to the application with the string::

-d12345678 -f430202 -uC3

0x41 A protocol violation occurred and an X.25 asynchronous packet was automatically issued to notify the remote device of this violation.

On reception of this return code, the following structure parameters are significant:

LOGICAL_CHANNEL is the logical channel on which the asynchronous packet was issued.

BUFFER_LENGTH is the length of the data associated with this asynchronous packet.

Q_D_M_BITS is the Q and D-bit settings in the transmitted packet.

CAUSE, **DIAGNOSTIC** is the X.25 cause and diagnostic fields associated with the packet issued (if applicable).

X25_PACKET_TYPE is the transmitted packet type.

If the high bit in this parameter is set, then a Restart Request/Indication packet was issued to reset the packet level after link level recovery procedures.

The **high bit should be masked** and then the X25_PACKET_TYPE is decoded as follows:

0x32 Clear Request/Indication
0x33 Reset Request/Indication
0x34 Restart Request/Indication
0x37 Diagnosti

TIME_STAMP is the time at which this transaction occurred (in binary coded decimal), as described for **RETURN_CODE** 0x40 above.

DATA is the data associated with this asynchronous packet (if the BUFFER_LENGTH is non-zero).

Note that for a Diagnostic packet, the diagnostic code reflects the diagnostic code transmitted in the packet. The data area contains both the diagnostic code and the diagnostic explanation fields.

0x42 A **time out occurred** for a previously issued X.25 packet and appropriate recovery action has been undertaken.

On reception of this return code, the following structure parameters are significant:

LOGICAL_CHANNEL is the logical channel on which the time-out occurred.

BUFFER_LENGTH is the length of the data associated with this asynchronous packet.

DIAGNOSTIC is the X.25 cause and diagnostic fields associated with the transmitted 'recovery' packet (if applicable).

X25_PACKET_TYPE is the packet type originally transmitted (and for which the time-out occurred) as follows:

0x03 Restart Request/Indication
0x05 Call Request
0x08 Clear Request/Indication
0x0A Reset request/Indication
0x0C Interrup

TIME_STAMP is the time at which this transaction occurred (in binary coded decimal), as described for **RETURN_CODE** 0x40 above.

DATA is the data associated with this asynchronous packet (if the BUFFER_LENGTH is non-zero).

If a timeout occurs for a Call Request packet, a Clear Request/Indication packet will be issued. If a timeout occurs for an Interrupt packet, then a Reset/Reste Indication packe will be transmitted. However, for Clear, Reset and Restart packets, the original packe type will be re-issued on a timeout, unless the device is configured as a DCE and the **config_options** have been set to send Diagnostic packets.

Note: the command executed when this return code was passed to the application was no performed (i.e. any returned data is invalid) and this call should be repeated.

0x43 The **retry limit was exceeded** for a previously issued X.25 packet.

On reception of this return code, the following structure parameters are significant:

LOGICAL_CHANNEL is the logical channel on which the time-out occurred.

X25_PACKET_TYPE is the packet type originally transmitted (and for which the retr limit was exceeded) as follows:

0x03 Restart Request/Indication
0x05 Call Request
0x08 Clear Request/Indication
0x0A Reset request/Indication

TIME_STAMP is the time at which this transaction occurred (in binary coded decimal), as described for **RETURN_CODE** 0x40 above.

Note: the command executed when this return code was passed to the application was no performed (i.e. any returned data is invalid) and this call should be repeated.

7. PC/SDLA Interface Bytes

There are a number of bytes within the shared PC/SDLA memory area which may be useful for programmers using the shared memory interface. These bytes are:

The MISCELLANEOUS_X25_HDLC_BITS (offset 0x1F00 from the defined memory window base address). This byte is set as follows:

bit 0 set to 0 if the HDLC link is disconnected.
set to 1 if the HDLC link is in the asynchronous balanced mode (ABM).

bit 1 if this bit is set to 1, then there are incoming X.25 Data packets queued for reception by the application.

bit 2 - reserved.

bit 3 - if this bit is set to 1, then there is trace data available for the application.

bit 4 reserved.

bit 5 if this bit is set to 1, then the application must be notified of an asynchronous occurrence. The application should issue an appropriate interface command to establish the type of packets issued/received.

bit 6 if this bit is set to 1, then data transmit buffer space is available on the SDL adapter.

bit 7 reserved.

The X25_LOGICAL_CHANNEL_BITS (offset 0x1F01 to 0x1FFF from the defined memor window base address). Each of these 255 bytes represent one of the 255 logical channels supported on the SDLA adapter and is set as follows:

bits 0-5 these bits indicate the number of incoming Data packets queued for this logical channel and available for reception by the application.

bit 6 if this bit is set to 1, then the X.25 transmit window is open and Data packets ma be issued for this logical channel by using the X25_WRITE command.

bit 7 if this bit is set to 0, then this logical channel is not in the data transfer mode. If this bit is set to 1, then this logical channel is in the data transfer mode and Data packets may be transmitted and received by using the X25_WRITE and X25_READ commands.

The mapping of these X25_LOGICAL_CHANNEL_BITS to the actual logical channels is established as follows:

If the highest logical channel set in the configuration file is less than or equal to 255, then the channel-to- X25_LOGICAL_CHANNEL_BITS mapping is a one-to-one mapping, with logical channel number 1 being represented by the byte at offset 0x1F01 and logica channel number 255 being represented by the byte at offset 0x1FFF.

If the highest logical channel set in the configuration file is greater than 255, then the channel-to- X25_LOGICAL_CHANNEL_BITS mapping uses CHANNEL_TO_STRUCTURE mapping values to establish the actual offset of the X25_LOGICAL_CHANNEL_BITS on the adapter. These CHANNEL_TO_STRUCTURE mapping values are unsigned short values located a offsets 0x1EF0 to 0x1EF7 from the defined memory window base address as follows:

0x1EF0-0x1EF1	the PVC mapping value
0x1EF2-0x1EF3	the Incoming Channel mapping value
0x1EF4-0x1EF5	the Two-way Channel mapping value
0x1EF6-0x1EF7	the Outgoing Channel mapping value

The application must establish the logical channel configuration by performing a READ_CHANNEL_CONFIGURATION command. Then, if the application wishes to examine the X25_LOGICAL_CHANNEL_BITS for a particular channel, the procedure is as follows:

Establish the type of logical channel selected and read the associated mapping value. For example, if the logical channel is a Two-way SVC, then the Two- wa mapping value is applicable.

Subtract the mapping value from the actual logical channel number, and then add 0x1F00. The result indicates the offset of the relevan X25_LOGICAL_CHANNEL_BITS byte from the defined memory window base address.

For example, assume our configuration file includes the following logical channe definitions:

```
lowest_PVC = 1
highest_PVC = 10
lowest_incomming_channel = 0
```

highest_incomming_channel = 0
lowest_two_way_channel = 1000
highest_two_way_channel = 1003
lowest_outgoing_channel = 0
highest_outgoing_channel = 0

The resultant PVC mapping value will be 0x0000 and the Two-way Channel mapping value will be 0x3DD (989 decimal).

The X25_LOGICAL_CHANNEL_BITS for PVC number 1 are found at offset 1 - 0x0000 + 0x1F00 = 0x1F01.

The X25_LOGICAL_CHANNEL_BITS for Two-way SVC number 1003 are found at offset 1003 - 989 + 0x1F00 = 0x1F0E.

These bytes are useful as they permit the application to make intelligent decisions before issuing interface commands, resulting in improved efficiency of both the X.25 microcode and the application running on the PC. Some examples of the usage of these interface bytes are as follows:

There is no use in issuing an IS_DATA_AVAILABLE command if MISCELLANEOUS_X25_HDLC_BITS do not indicate that the HDLC link is in the ABM and that there is incoming X.25 Data packets queued for reception by the application.

If the application wishes to issue an X25_WRITE command, then the following should be checked so as to avoid a non- zero (error) return code:

The MISCELLANEOUS_X25_HDLC_BITS should indicate that the HDLC link is in the ABM and that data transmit buffer space is available on the SDLA adapter.

The X25_LOGICAL_CHANNEL_BITS should indicate that the logical channel is in the data transfer mode and that the X.25 transmit window is open for this channel.

When using the XIP interface, the PC time stamp is periodically downloaded to the board by the XIP. This information is used for time stamping asynchronous packet transactions as well as trace frames. If the application is using the shared memory interface and not the XIP, then the application is responsible for supplying this date and time information, if required. This time stam is in a binary coded decimal format as follows:

Offset from window base address	Parameter
0x1EF8	month
0x1EF9	day
0x1EFA	hour
0x1EFB	minutes
0x1EFC	seconds

For example, a time-stamp of

0x09, 0x01, 0x15, 0x46, 0x03

represents the 1st of September (9th month) at 3:46:03 PM.

8. Adapter/PC Interrupt Usage

There are two interface commands for interrupt usage. These are:

0x17	SET_INTERRUPT_TRIGGERS
0x18	READ_INTERRUPT_TRIGGERS

8.1 SET_INTERRUPT_TRIGGERS (0x17)

Set the occurrences which will cause the SDLA adapter to trigger a hardware interrupt on the PC.

Control Block values to be set on entry:

BUFFER_
LENGTH: Set to 0x01.

DATA: Offset 0x00 defines the interrupt triggers as follows:

bit 0 - the receive interrupt bit.
If this bit is set, then an interrupt will be triggered if there is an incoming Data packet available for reception by the application.
The receive interrupt may be used in two different ways, in combination with bit 7 of the interrupt trigger byte as follows:

a) If bit 7 is reset, then when a receive interrupt interrupt is triggered, the application needs to establish on which LCN to perform the X25_READ command so as to receive the incomming data (if multiple LCNs are in use). This may be done by examining the X25_LOGICAL_CHANNEL_BITS as described in the Section "PC/SDLA Interface Bytes" or by using the IS_DATA_AVAILABLE command.

b) If bit 7 is set, then the user must examine the receive mailbox on a receive interrupt. This mailbox will have been filled in as if the application had already performed a X25_READ command i.e., the return code, buffer length, Q/D/M bits and the data will be valid, and the logical channel will have been set to that on which the Data packet was received.

bit 1 - the transmit interrupt bit.
The transmit interrupt may be used in two different ways:

a) LCN specific transmit interrupt, where an interrupt will be triggered if a Data packet may be transmitted on the LCN specified in the 'LOGICAL_CHANNEL' area of the mailbox when performing this SET_INTERRUPT_TRIGGERS command.

b) non-LCN specific transmit interrupt, where an interrupt will be triggered if at least one X.25 transmit buffer is available on the adapter. LCN zero is specified in the logical channel definition area.

The use of this non-LCN specific interrupt is complicated by the fact tha there is no guarantee that a subsequent X25_WRITE command will be successfull. For example, the transmit window for the selected logica channel may be closed, causing a return code of 0x33. The user may be satisfied with this interrupt scheme or may otherwise set the 'Q_D_M_bits' parameter to 0x01 when selecting the non-LCN specific transmit interrup in the SET_INTERRUPT_TRIGGERS command. The application mus then set bt 5 of the X25_LOGICAL_CHANNEL_BITS associated with the logical channels on which data is to be transmitted. A transmit interrup will now only be generated if a successfull X25_WRITE command can be executed on one of the selected LCNs.

bit 2 - the modem status interrupt.
If this bit is set, then an interrupt will be triggered when a change in the state o CTS or DCD occurs. The details of this modem status change may be established by using a LINK_STATUS command to illicit a return code of 0x08 and then examining the byte at offset 0x00 in the structure data area.

bit 3 - the 'command complete' interrupt bit.
If bit 3 is set, then an interrupt will be triggered on completion of an interface command, i.e. when the 'opp_flag' has been reset.

bit 4 - the asynchronous transaction interrupt bit.
If bit 4 is set, then an interrupt will be triggered on the occurrence of an X.25 asynchronous transaction. The details of this status change may be established b using an IS_DATA_AVAILABLE command to illicit a return code of 0x40, 0x41, 0x42 or 0x43 as documented in the section "Notes on Return Codes".

bits 5, 6 - reserved for later use.

bit 7 - the direct receive interrupt bit.
See the setting of bit 0 for further details.

Control Block values set on return:

RETURN_
CODE: 0x00 The action has been performed successfully.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

8.2 READ_INTERRUPT_TRIGGERS (0x18)

Read the current interrupt trigger configuration as set in the SET_INTERRUPT_TRIGGERS command.

Control Block values to be set on entry:

BUFFER_
LENGTH: Set to 0x00.

Control Block values set on return:

RETURN
_CODE: 0x00 The action has been performed successfully.

0x06, 0x07, 0x08, 0x09, 0x0A, 0x40, 0x41, 0x42, 0x43
See Section "Notes on Return Codes" for further details.

BUFFER_
LENGTH: Set to 0x01 if a RETURN_CODE of 0x00 is received.

DATA: The interrupt trigger configuration as defined in the SET_INTERRUPT_TRIGGERS command.

8.3 Interrupt Usage

Once the interrupt vector and the PIC are initialized on the PC, the application informs the X.25 code of the required interrupt configuration by using the SET_INTERRUPT_TRIGGERS command. Thereafter, the following procedure must be used to enable the SDLA hardware to trigger the interrupt:

For the S502E adapter:
write 0x03 out to the SDLA base port address
write 0x07 out to the SDLA base port address

For the 503 adapter:
write 0x7F out to the SDLA base port address

Note that the SDLA base port address is defined as ‘io_port’ in the X25.SDL configuration file.

When an interrupt occurs on the PC, the interrupt handler should examine the INTERRUPT_INTERFACE_BYTE at offset 0x1EFD from the defined memory window base address. This byte is as set as follows:

If bit 0 is set, then the interrupt has been triggered due to a Data packet being available for reception by the application.

If bit 1 is set, then a transmit interrupt has been triggered.

If bit 2 is set, then an interrupt has been triggered due to a change in the state of CTS or DCD.

If bit 3 is set, then an interrupt has been triggered due to the completion of the las interface command, i.e. the 'opp_flag' has been reset.

If bit 4 is set, then the interrupt has been triggered due to an X.25 asynchronous packe transaction.

To reset the interrupt and to permit the next interrupt to be triggered, the interrupt handler should:

Reset the PIC on the PC

For the S502E adapter:
write 0x03 out to the SDLA base port address
write 0x07 out to the SDLA base port address

Reset the INTERRUPT_INTERFACE_BYTE byte (i.e., set this byte to 0x00)

Note that no writes to the SDLA base port address are required for the S503 adapter.

Once the interrupts have been enabled by using the SET_INTERRUPT_TRIGGERS command, the interrupts may be temporarily disabled by using the INTERRUPT_PERMISSION_BYTE a offset 0x1EFE from the defined memory window base address. The interrupt bit map for the INTERRUPT_PERMISSION_BYTE is the same as that used for the SET_INTERRUPT_TRIGGERS command, i.e,

bit 0 is used for frame reception
bit 1 is used for frame transmission
bit 2 is used for modem transitions
bit 3 is used for command completions
bit 4 is used for asynchronous packet transactions

If the bit is reset by the application, then an interrupt of that type w **not** occur until the bit is se again.

An example of the usage of this interface byte is as follows:

Assume that the X.25 code is configured for transmit, receive and 'command complete' interrupts and the application is passed a transmit interrupt. In the transmit interrup handler, both receive and transmit interrupts are now reset (disabled) in the INTERRUPT_PERMISSION_BYTE and an X25_WRITE command is used to send the Data packet. This ensures that the next interrupt to the PC if a 'command complete' interrupt and is not due to another transmit or receive interrupt. The 'command complete' interrupt handler would then once again set (enable) the transmit and receive interrupts in the INTERRUPT_PERMISSION_BYTE.

9. General Programming Notes

Once completion of an interface command has occurred (the XIP returns to the calling application or the 'opp_flag' has been reset in the shared memory area), the application should examine the **RETURN_CODE** and other parameters relevant to the particular command. Note that the

RETURN_CODE is most important, as it may indicate that the interface command was no successful. For example, if the command is an **X25_WRITE** and the **RETURN_CODE** is 0x33, then the Data packet was not sent to the adapter and this same **X25_WRITE** should be re-issued.

The return codes 0x40, 0x41, 0x42 and 0x43 (Section **Notes on Return Codes**) are all concerned with the handling of asynchronous X.25 packet pragmatics such as call setup, call clearing, rese and restart procedures. It is most important that the application should be notified of such an asynchronous transaction as soon as possible after its occurrence as, for example, a Rese Request/Indication received or transmitted may mean the loss of Data packets on that particular logical channel. For this reason, once an asynchronous transaction has been recorded, **the next call to the interface with the following commands will always pass the appropriate asynchronous return code to the application**:

X25_READ
X25_WRITE
IS_DATA_AVAILABLE
PLACE_CALL
ACCEPT_CALL
CLEAR_CALL
CONFIRM_CLEAR
RESET_LOGICAL_CHANNEL
CONFIRM_RESET
RESTART
CONFIRM_RESTART
INTERRUPT
CONFIRM_INTERRUPT
REGISTRATION_REQUSET
REGISTRATION_CONFIRMATION

Note that this return code will be passed irrespective of the **LOGICAL_CHANNEL** used in the interface call. For example, if a Reset Request was received on logical channel 3 and the application executes an **X25_READ** command on logical channel 1, the return code of 0x40 would still be passed to the application. These asynchronous return codes indicate that the interface command was not successful and should be repeated.

If the link level is active, then data transfer may take place on any PVC's by using the **X25_READ** and **X25_WRITE** commands. In addition, the **IS_DATA_AVAILABLE** command is used to establish if there is incoming data available for reception by the application.

Use the **PLACE_CALL** command to issue a call on a SVC. At some later stage, a return code o **0x40** should be received on an interface call, indicating that an incoming asynchronous X.25 packet has been logged. The data associated with this return code will indicate whether your

outgoing call has been accepted or cleared. If the call has been accepted then data transfer ma commence as described for PVC's above.

Clear the call by issuing a **CLEAR_CALL** command on the selected logical channel. As described above, a return code of **0x40** will notify the application of the reception of a Clear Confirmation packet received from the remote device (or the occurrence of an X.25 time-out).

10. Example Code

10.1 Example code for interfacing with the XIP TSR

(WRITTEN FOR BORLAND C++, Version 4.0)

/*

To execute an XIP command, the following steps should be performed:

Set all the parameters in the XIP control block structure parameters required for the particular command. For example, if the command is an X25_WRITE, then the LOGICAL_CHANNEL, the BUFFER_LENGTH, the Q_D_M_BITS and DATA areas must be completed.

Set the BX register to the data segment and the DX register to the offset of the XIP control block structure.

Call the XIP software interrupt.

The XIP will carry out the defined command. It will then update the XIP control block at the application level with the required return code and, if applicable, the associated data buffer, data length etc.

Once control is returned to your application, examine the RETURN_CODE and other parameters relevant to the particular XIP command.

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <mem.h>
#include <dos.h>


/* XIP commands */

#define X25_READ             0x22     /* receive data */
#define X25_WRITE            0x23     /* transmit data */
#define PLACE_CALL           0x30     /* issue a Call Request packet */
#define IS_DATA_AVAILABLE    0x40     /* is incomming data available? */
#define READ_ACTIVE_CHANNELS 0x43     /* return a list of active channels */


#define ASY_PKT_RECEIVED     0x40     /* XIP return code for an asynchronous */
                                      /* packet received */

#define XIP_INTERRUPT        0x6F     /* the XIP software interrupt */

/* the XIP calling structure */
typedef struct {
  char command;                       /* XIP command */
  unsigned short buffer_length;       /* buffer length */
  char return_code;                   /* return code */
  char reserved_HDLC;                 /* reserved for HDLC use */
  unsigned short logical_channel;     /* X.25 logical channel */
  char Q_D_M_bits;                    /* X.25 Q/D/M bits */
  char cause;                         /* X.25 cause field */
  char diagnostic;                    /* X.25 diagnostic field */
  char X25_packet_type;               /* X.25 packet type */
  char time_stamp[4];                 /* time stamp for asynchronous packets */
  char reserved[1];                   /* reserved for later use */
  char data[1040];                    /* data area */
  } XIP_CALL_STRUCT;
XIP_CALL_STRUCT XIP_struct;

unsigned logical_channel_in_use;
int i;

void place_X25_call(void);
void wait_for_call_result(void);
```

```
void check_active_channels(void);
void send_X25_Data_packet(void);
void receive_X25_Data_packet(void);
void execute_interface_command(void);
void exit_from_example(char, char);


void main()
{
   /* issue an X.25 Call Request packet */
   place_X25_call();

   /* wait for the result of this Call Request */
   wait_for_call_result();

   /* check for active logical channels */
   check_active_channels();

   /* send an X.25 Data packet */
   send_X25_Data_packet();

   /* receive an X.25 Data packet */
   receive_X25_Data_packet();
}



/*** issue an X.25 Call Request packet ***/

void place_X25_call()
{
   /* set the command */
   XIP_struct.command = PLACE_CALL;

   /* set the Q/D/M bits */
   XIP_struct.Q_D_M_bits = 0x00;

   /* set the call data in the data area */
   sprintf(XIP_struct.data, "%s", "-d12345678 -s98765432 -f0101");

   /* set the length of the call data */
   XIP_struct.buffer_length = 28;

   /* perform the interface command */
   execute_interface_command();

   /* exit on a bad return code */
   if(XIP_struct.return_code)
      exit_from_example((char)PLACE_CALL, XIP_struct.return_code);

   /* the call was successfully placed */
   printf("\nCall placed on logical channel = %d\n", XIP_struct.logical_channel);
}



/*** wait for the result of a previously placed Call Request ***/

void wait_for_call_result()
{
   /*
   Poll the interface until the result of the outgoing call is received. Note       that in this
example we use the IS_DATA_AVAILABLE command to poll the
   interface. In practice, the return code informing the application of the
   reception of an asynchronous X.25 packet would also be passed to the
   application on an X25_READ and X25_WRITE command, as well as any command
   which sends an asynchronous packet.
   */
   do {
      /* set the command */
      XIP_struct.command = IS_DATA_AVAILABLE;
```

```
      /* set the length of the data buffer */
      XIP_struct.buffer_length = 0x00;

      /* perform the interface command */
      execute_interface_command();

      /* exit on a keyboard hit */
      if(kbhit())
        _exit((int)0x00);

      /* loop until we see that an asynchronous X.25 has been received */
   } while(XIP_struct.return_code != ASY_PKT_RECEIVED);

   /* display the asynchronous packet details */
   printf("\nX.25 asynchronous packet received on logical channel %d",
   XIP_struct.logical_channel);
   printf("\nX.25 packet type (%x): ", XIP_struct.X25_packet_type & 0x7F);

   switch(XIP_struct.X25_packet_type & 0x7F) {

      case (0x02):
         printf("Clear Request");
         /* display the cause and diagnostic fields */
         printf("\nX.25 cause field = %x", XIP_struct.cause);
         printf("\nX.25 diagnostic field = %x", XIP_struct.diagnostic);
         break;

      case (0x31):
         printf("Call Accept");
         break;

      default:
         printf("See X.25 manual");
   }

   /* display any data in the packet */
   if(XIP_struct.buffer_length) {
      printf("\nData associated with this incomming packet: ");
      for(i = 0; i < XIP_struct.buffer_length; i ++)
         printf("%c", XIP_struct.data[i]);
   }
}



/*** check for active logical channels ***/

void check_active_channels()
{
   /* check which logical channels are active */
   do {
      /* set the command */
      XIP_struct.command = READ_ACTIVE_CHANNELS;

      /* set the length of the data buffer */
      XIP_struct.buffer_length = 0x00;

      /* perform the interface command */
      execute_interface_command();

   /* loop until the command is successful */
   } while(XIP_struct.return_code);

   printf("\n\n%u logical channels are active", XIP_struct.buffer_length / 2);

   /* display a list of active logical channels */
   if(XIP_struct.buffer_length) {
      printf("\nThese are: ");
      for(i = 0; i < XIP_struct.buffer_length; i ++)
         printf("%u ", *(unsigned *)&XIP_struct.data[i++]);
   }
}
```

```
/*** send an X.25 Data packet ***/

void send_X25_Data_packet()
{
   /* set the command */
   XIP_struct.command = X25_WRITE;

   /* send the Data packet on the first logical channel in the previously */
   /* returned list of active channels */
   XIP_struct.logical_channel =  *(unsigned *)&XIP_struct.data[0x00];

   /* set the Q, D and M-bits */
   XIP_struct.Q_D_M_bits = 0x00;

   /* set the actual data to be sent */
   sprintf(XIP_struct.data, "%s", "X.25 TEST PACKET");

   /* set the length of the data */
   XIP_struct.buffer_length = 16;

   /* perform the interface command */
   execute_interface_command();

   if(XIP_struct.return_code)
      printf("\n\nData packet not sent (return code = %x)",
      XIP_struct.return_code);
   else
      printf("\n\nData successfully sent");
}



/*** receive an X.25 Data packet ***/

void receive_X25_Data_packet()
{
   /* check to see if there is any incomming data available for reception */

   /* set the command */
   XIP_struct.command = IS_DATA_AVAILABLE;

   /* set the length of the data buffer */
   XIP_struct.buffer_length = 0x00;

   /* perform the interface command */
   execute_interface_command();

   /* if there is data available, then print a list of the logical channels */
   /* which have packets queued for reception by the application */
   if(!XIP_struct.return_code) {

      printf("\n\n%d logical channels have incomming data available",
      XIP_struct.buffer_length / 2);
      /* display a list of the channels on which data has been received */
      printf("\nThese are: ");
      for(i = 0; i < XIP_struct.buffer_length; i ++)
         printf("%u ", *(unsigned *)&XIP_struct.data[i++]);

      /* receive a packet on the first logical channel in this list */
      logical_channel_in_use = *(unsigned *)&XIP_struct.data[0x00];
      do {
         /* set the command */
         XIP_struct.command = X25_READ;

         /* set the length of the data buffer */
         XIP_struct.buffer_length = 0x00;

         /* set the logical channel */
         XIP_struct.logical_channel = logical_channel_in_use;
```

```
         /* perform the interface command */
         execute_interface_command();

      /* loop until the call is successful */
      } while(XIP_struct.return_code);

      /* display the received data */
      printf("\n\n%d bytes of data received on logical channel %d",
      XIP_struct.buffer_length, logical_channel_in_use);
      printf("\nData: ");
      for(i = 0; i < XIP_struct.buffer_length; i ++)
         printf("%x ", XIP_struct.data[i]);
   }
}


/*** perform an interface command ***/

void execute_interface_command()
{
union REGS inregs;

   /* set the BX register to our Data Segment */
   inregs.x.bx = _DS;

   /* set the DX register to point to the XIP structure */
   inregs.x.dx = (unsigned short)&XIP_struct.command;

   /* call the XIP with a software interrupt */
   int86(XIP_INTERRUPT, &inregs, &inregs);
}



/*** exit to the operating system ***/

void exit_from_example(char command, char exit_code)
{
   printf("\n\nEXITING DUE TO RETURN CODE %x FOR COMMAND %x", exit_code,
   command);
   _exit((int)0x00);
}
```

10.2 Example code showing use of the shared memory area

(Written for BORLAND C++, Version 4.0)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <mem.h>
#include <dos.h>

/* XIP commands */
#define X25_READ             0x22     /* receive data */
#define X25_WRITE            0x23     /* transmit data */
#define PLACE_CALL           0x30     /* issue a Call Request packet */
#define IS_DATA_AVAILABLE    0x40     /* is incomming data available? */
#define READ_ACTIVE_CHANNELS 0x43     /* return a list of active channels */


#define ASY_PKT_RECEIVED     0x40     /* XIP return code for an asynchronous */
                                      /* packet received */

/*
   Assume that the X.25 code was loaded with the configuration file defining
   the memory address as 0xD000 and the memory window as 0. The physical
   address of the SEND interface mailbox is 0xD000:0x16B0 and the address of
   the RECEIVE interface mailbox 0xD000:0x1AD0.
*/
#define SEND_MAILBOX_ADDRESS              0xD00016B0
#define RECEIVE_MAILBOX_ADDRESS           0xD0001AD0


/* for opp_flag use */
#define RESET                  0x00
#define SET                    0x01
#define ZERO                   0x00


/* the mailbox structure */
typedef struct {
  char opp_flag;                      /* the opp flag */
  char command;                       /* XIP command */
  unsigned short buffer_length;       /* buffer length */
  char return_code;                   /* return code */
  char reserved_HDLC;                 /* reserved for HDLC use */
  unsigned short logical_channel;     /* X.25 logical channel */
  char Q_D_M_bits;                    /* X.25 Q/D/M bits */
  char cause;                         /* X.25 cause field */
  char diagnostic;                    /* X.25 diagnostic field */
  char X25_packet_type;               /* X.25 packet type */
  char time_stamp[4];                 /* time stamp for asynchronous packets */
  char data[1040];                    /* data area */
} MAILBOX_INTERFACE_STRUCT;

/* define far pointers to the SEND and RECEIVE interface structures */
MAILBOX_INTERFACE_STRUCT _far *SEND_mailbox_ptr;
MAILBOX_INTERFACE_STRUCT _far *RECEIVE_mailbox_ptr;

unsigned logical_channel_in_use;
int i;
char call_arguments[] = "-d12345678 -s98765432 -f0101";
char test_data[] = "X.25 TEST PACKET";
char utility_buffer[1040];

void place_X25_call(void);
void wait_for_call_result(void);
void check_active_channels(void);
void send_X25_Data_packet(void);
```

```
void receive_X25_Data_packet(void);
void execute_interface_command(MAILBOX_INTERFACE_STRUCT _far *);
void exit_from_example(char, char);


void main()
{
   /* set the SEND mailbox pointer to the physical address on the adapter */
   SEND_mailbox_ptr = (MAILBOX_INTERFACE_STRUCT _far*)SEND_MAILBOX_ADDRESS;

   /* set the RECEIVE mailbox pointer to the physical address on the adapter */
   RECEIVE_mailbox_ptr = (MAILBOX_INTERFACE_STRUCT
    _far*)RECEIVE_MAILBOX_ADDRESS;

   /* issue an X.25 Call Request packet */
   place_X25_call();

   /* wait for the result of this Call Request */
   wait_for_call_result();

   /* check for active logical channels */
   check_active_channels();

   /* send an X.25 Data packet */
   send_X25_Data_packet();

   /* receive an X.25 Data packet */
   receive_X25_Data_packet();
}



/*** issue an X.25 Call Request packet ***/

void place_X25_call()
{
   /* set the command */
   SEND_mailbox_ptr->command = PLACE_CALL;

   /* set the Q/D/M bits */
   SEND_mailbox_ptr->Q_D_M_bits = 0x00;

   /* set the call data in the data area */
   _fmemcpy(SEND_mailbox_ptr->data, call_arguments, 28);

   /* set the length of the call data */
   SEND_mailbox_ptr->buffer_length = 28;

   /* perform the interface command */
   execute_interface_command(SEND_mailbox_ptr);

   /* exit on a bad return code */
   if(SEND_mailbox_ptr->return_code)
      exit_from_example((char)PLACE_CALL, SEND_mailbox_ptr->return_code);

   /* the call was successfully placed */
   printf("\nCall placed on logical channel = %d\n", SEND_mailbox_ptr
   ->logical_channel);
}



/*** wait for the result of a previously placed Call Request ***/

void wait_for_call_result()
{
   /*
   Poll the interface until the result of the outgoing call is received. Note
   that in this example we use the IS_DATA_AVAILABLE command to poll the
   interface. In practice, the return code informing the application of the
   reception of an asynchronous X.25 packet would also be passed to the
   application on an X25_READ and X25_WRITE command, as well as any command
```

```
   which sends an asynchronous packet.
   */
   do {
      /* set the command */
      SEND_mailbox_ptr->command = IS_DATA_AVAILABLE;

      /* set the length of the data buffer */
      SEND_mailbox_ptr->buffer_length = 0x00;

      /* perform the interface command */
      execute_interface_command(SEND_mailbox_ptr);

      /* exit on a keyboard hit */
      if(kbhit())

        _exit((int)0x00);

      /* loop until we see that an asynchronous X.25 has been received */
   } while(SEND_mailbox_ptr->return_code != ASY_PKT_RECEIVED);

   /* display the asynchronous packet details */
   printf("\nX.25 asynchronous packet received on logical channel %d",
   SEND_mailbox_ptr->logical_channel);
   printf("\nX.25 packet type (%x): ", SEND_mailbox_ptr->X25_packet_type &
   0x7F);

   switch(SEND_mailbox_ptr->X25_packet_type & 0x7F) {

      case (0x02):
         printf("Clear Request");
         /* display the cause and diagnostic fields */
         printf("\nX.25 cause field = %x", SEND_mailbox_ptr->cause);
         printf("\nX.25 diagnostic field = %x", SEND_mailbox_ptr->diagnostic);
         break;

      case (0x31):
         printf("Call Accept");
         break;

      default:
         printf("See X.25 manual");
   }

   /* display any data in the packet */
   if(SEND_mailbox_ptr->buffer_length) {
      printf("\nData associated with this incomming packet: ");
      _fmemcpy(utility_buffer, SEND_mailbox_ptr->data, SEND_mailbox_ptr
       ->buffer_length);
      for(i = 0; i < SEND_mailbox_ptr->buffer_length; i ++)
         printf("%c", utility_buffer[i]);
   }
}



/*** check for active logical channels ***/

void check_active_channels()
{
   /* check which logical channels are active */
   do {
      /* set the command */
      SEND_mailbox_ptr->command = READ_ACTIVE_CHANNELS;

      /* set the length of the data buffer */
      SEND_mailbox_ptr->buffer_length = 0x00;

      /* perform the interface command */
      execute_interface_command(SEND_mailbox_ptr);

   /* loop until the command is successful */
```

```
   } while(SEND_mailbox_ptr->return_code);

   printf("\n\n%u logical channels are active", SEND_mailbox_ptr->buffer_length     / 2);

   /* display a list of active logical channels */
   if(SEND_mailbox_ptr->buffer_length) {
      _fmemcpy(utility_buffer, SEND_mailbox_ptr->data, SEND_mailbox_ptr
      ->buffer_length);
      printf("\nThese are: ");
      for(i = 0; i < SEND_mailbox_ptr->buffer_length; i ++)
         printf("%u ", *(unsigned *)&utility_buffer[i++]);
   }
}


/*** send an X.25 Data packet ***/

void send_X25_Data_packet()
{
   /* set the command */
   SEND_mailbox_ptr->command = X25_WRITE;

   /* send the Data packet on the first logical channel in the previously */
   /* returned list of active channels */
   SEND_mailbox_ptr->logical_channel =  *(unsigned *)&utility_buffer[0x00];

   /* set the Q, D and M-bits */
   SEND_mailbox_ptr->Q_D_M_bits = 0x00;

   /* set the actual data to be sent */
   _fmemcpy(SEND_mailbox_ptr->data, test_data, 16);

   /* set the length of the data */
   SEND_mailbox_ptr->buffer_length = 16;

   /* perform the interface command */
   execute_interface_command(SEND_mailbox_ptr);

   if(SEND_mailbox_ptr->return_code)
      printf("\n\nData packet not sent (return code = %x)",
      SEND_mailbox_ptr->return_code);
   else
      printf("\n\nData successfully sent");
}



/*** receive an X.25 Data packet ***/

void receive_X25_Data_packet()
{
   /* check to see if there is any incomming data available for reception */

   /* set the command */
   SEND_mailbox_ptr->command = IS_DATA_AVAILABLE;

   /* set the length of the data buffer */
   SEND_mailbox_ptr->buffer_length = 0x00;

   /* perform the interface command */
   execute_interface_command(SEND_mailbox_ptr);

   /* if there is data available, then print a list of the logical channels */
   /* which have packets queued for reception by the application */
   if(!SEND_mailbox_ptr->return_code) {

      printf("\n\n%d logical channels have incomming data available",
      SEND_mailbox_ptr->buffer_length / 2);
      /* display a list of the channels on which data has been received */
      _fmemcpy(utility_buffer, SEND_mailbox_ptr->data, SEND_mailbox_ptr
      ->buffer_length);
      printf("\nThese are: ");
```

```
      for(i = 0; i < SEND_mailbox_ptr->buffer_length; i ++)
         printf("%u ", *(unsigned *)&utility_buffer[i++]);

      /* receive a packet on the first logical channel in this list - note */
      /* that the RECEIVE mailbox is used for the X25_READ command */
      logical_channel_in_use = *(unsigned *)&utility_buffer[0x00];
      do {
         /* set the command */
         RECEIVE_mailbox_ptr->command = X25_READ;

         /* set the length of the data buffer */
         RECEIVE_mailbox_ptr->buffer_length = 0x00;

         /* set the logical channel */
         RECEIVE_mailbox_ptr->logical_channel = logical_channel_in_use;

         /* perform the interface command */
         execute_interface_command(RECEIVE_mailbox_ptr);

      /* loop until the call is successful */
      } while(RECEIVE_mailbox_ptr->return_code);

      /* display the received data */
      printf("\n\n%d bytes of data received on logical channel %d",
      RECEIVE_mailbox_ptr->buffer_length, logical_channel_in_use);
      printf("\nData: ");
      _fmemcpy(utility_buffer, RECEIVE_mailbox_ptr->data, RECEIVE_mailbox_ptr
       ->buffer_length);
      for(i = 0; i < RECEIVE_mailbox_ptr->buffer_length; i ++)
         printf("%x ", utility_buffer[i]);
   }
}
```

```
/*** perform an interface command ***/

void execute_interface_command(mailbox_ptr)
MAILBOX_INTERFACE_STRUCT _far *mailbox_ptr;
{
long opp_flag_loops = 0x00;

   /* set the 'opp_flag' to initiate the processing of the command */
   mailbox_ptr->opp_flag = SET;

   /* Read the 'opp_flag' from the shared memory area and, when it has been
      reset, then the command has been completed. Note that the application
      should check to see that the 'opp_flag' is reset within a defined time
      period after being set. A suitable time out would be approximately one
      second (most commands would be completed with 1/100th of a second).
   */
   do {
      if((++ opp_flag_loops) == 200000) {
         printf("\nAdapter dead. Contact your Sangoma representative.");
         _exit((int)ZERO);

      }
   /* loop until the 'opp_flag' has been reset */
   } while(mailbox_ptr->opp_flag != RESET);
}



/*** exit to the operating system ***/

void exit_from_example(char command, char exit_code)

{
   printf("\n\nEXITING DUE TO RETURN CODE %x FOR COMMAND %x", exit_code,
   command);
   _exit((int)0x00);
}
```

11. HDLC Implementation

11.1 HDLC Protocol Specifics

This section is only of interest to those users interfacing at the HDLC level.

The implementation of the HDLC protocol on the S502 adapter corresponds as closely as possible to the specifications of the ISO 7776 document "Information processing systems - Data communication - High-level data link control procedures - Description of X.25 LAPB-compatible DTE data link procedures", 15 December 1986, with the following restrictions:

Link Initialization

After issuing a **CONFIGURE_LINK** command and a subsequent **HDLC_OPEN**, the station issues DM responses at a period of T1 seconds. Also, all incoming frames with the P bit set will be responded to with a DM frame.

Once the **LINK_SETUP** command is issued, the station will issue the DM frame N2 times and then send SABM commands. Incoming SABMs will be responded to with a UA, setting the link in the ABM.

Note that if **auto_HDLC** is enabled in the configuration file (as is usually done for an X.25 implementation), then the **CONFIGURE_LINK**, **HDLC_OPEN** and **LINK_SETUP** commands are automatically issued on code startup.

Link Disconnection

When the **LINK_DISCONNECT** command is used, a DISC command is issued on the link at a period of T1 until a UA response is received. The station then issues DM responses at a period o T1 seconds.

Frame Reject Mode

The HDLC station will begin Frame Reject transmission under the following circumstances:

The Control field in the received frame does not allow an information field to be included wit the frame, but an information field is present.

The function specified by the Control field has not been implemented.

An Information frame is received with an I-field which exceeds the maximum defined length.

When receiving a FRMR during the ABM, this station will attempt link re-initialization by issuing a SABM command.

11.2 X.25 Protocol Specifics

The implementation of the X.25 protocol on the S502 adapter corresponds as closely as possibl to the specifications of the ISO 8208 document "Information technology - Data communications - X.25 Packet Layer Protocol for Data Terminal Equipment", 15 March 1990, with the following restrictions:

The maximum data packet length is restricted to 1024 bytes.

The TOA/NPI address format is not supported.

The code is limited to a sequence numbering scheme of 8.

Reject packet pragmatics are not supported.

12. Error messages

12.1 XLOAD.EXE

If XLOAD does not execute successfully, an error message will be displayed and an exit code w be returned. The error messages and corresponding exit codes (DOS ERRORLEVEL) are as follows:

"**A command line error was found when executing XLOAD**"
An invalid command line argument was used or the **CODEFILE** or **CONFIG** arguments were omitted (exit code of 1).

"**The file FILENAME was not found**"
A filename listed in the command line arguments was not found in the defined directory (exit code of 2).

"**The parameter PARAMETER was not found in the configuration file**"
There is an error in the X.25 configuration file and the listed **PARAMETER** could not be located (exit code of 3).

"**The parameter PARAMETER read from the configuration file is invalid**"
There is an error in the X.25 configuration file as the listed **PARAMETER** is invalid (exit code of 3).

"**The code running on the adapter is not the same as the original downloaded code**"
There is a memory or I/O port address conflict in your PC. Change the I/O port address and/or the memory segment and memory window parameters (exit code of 4).

"**The downloaded code is not running on the adapter**"
The SDLA CPU has halted. Contact your Sangoma representative (exit code of 5).

"**The S502 adapter is configured with the incorrect serial communications chip**"
Contact your Sangoma representative (exit code of 6).

12.2 XIP.COM

If XIP does not execute successfully (exit code o **0x00**), an error message will be displayed and a non-zero exit code will be returned. The error messages and corresponding exit codes are as follows:

"**THE XIP TSR IS ALREADY RESIDENT**"
Multiple copies of this TSR may not be run at the same software interrupt address (exit code of 0x01).

"**A COMMAND LINE ERROR WAS FOUND**"
An invalid command line argument was used and the valid arguments are displayed (exit code of 0x02).

"**THE DEFINED X.25/HDLC CONFIGURATION FILE HAS NOT BEEN FOUND**"
The configuration filename listed in the command line arguments (or the default configuration file) was not found in the defined directory (exit code of 0x03).

"**ERROR IN READING THE X.25/HDLC CONFIGURATION FILE**"
The configuration file could not be opened for reading (exit code of 0x04).

"**THE X.25/HDLC CONFIGURATION FILE IS OF EXCESSIVE LENGTH**"
The configuration file could not be read into the allocated buffer space (exit code of 0x05).

"**THE I/O PORT ADDRESS OR THE MEMORY CONFIGURATION PARAMETERS WERE NOT FOUND IN THE CONFIGURATION FILE**"
Check the defined configuration file for the presence of these two parameters (exit code of 0x06).

12.3 X25_TST.EXE

"**A COMMAND LINE ERROR WAS FOUND**"
An invalid command line argument was used (exit code of 1).