SCX160 SCxbus Adapter User's Guide

for Windows

05-1126-004

COPYRIGHT NOTICE

Publication Date: April, 2000

Part Number: 05-1126-004

Dialogic, an Intel Company
1515 Route 10
Parsippany NJ 07054
U.S.A.

For **Technical Support**, visit the Dialogic support website at:
http://support.dialogic.com

For **Sales Offices** and other contact information, visit the main Dialogic website at:
http://www.dialogic.com

Table of Contents

List of Tables

List of Figures

1. How to Use This Manual

1.1. Products Covered By This Guide

This guide is for users who have a Dialogic SCX160 SCxbus Adapter and related software installed on a host computer (PC) operating in a Windows environment. Product terminology conventions and the organization of this guide are described in this chapter.

The SCX160 SCxbus Adapter refers to the Dialogic SCxbus switchless adapter boards designed to implement an SCSA Multi-Node Architecture (MNA) that brings together a similar or dissimilar system into a single system image at the application level. The SCX160 SCxbus Adapter provides an interface between the local node SCbus time slots and the external SCxbus.

1.2. Product Terminology

The following product naming conventions are used throughout this guide:

D/41ESC refers to the Dialogic 4 channel voice board with play/record, tone, and call progress and 4 on-board analog loop start interface.

D/80SC refers to the Dialogic 8-channel voice board with play/record, tone, and call progress for use with an SCbus network interface board.

D/80SC-4LS refers to the Dialogic 8-channel voice board with play/record, tone, and call progress and 4 on-board loop start interfaces.

D/160SC refers to the Dialogic 16-channel voice board with play/record, tone, and call progress for use with an SCbus network interface board.

D/160SC-LS refers to the Dialogic 16-channel voice board with play/record, tone, and call progress and 16 on-board analog loop start interfaces.

D/240SC refers to the Dialogic 24-channel voice board with play/record, tone, and call progress for use with a network interface board.

D/240SC-T1 refers to the Dialogic 24-channel voice board with play/record, tone, and call progress and on-board T-1 digital interface. Provides T-1/DSX-1 and ISDN Primary Rate connectivity.

D/240SC-2T1 refers to the Dialogic DualSpan board with 24 ports of voice processing and 48 ports of network interface.

D/300SC-E1 refers to the Dialogic 30-channel voice board with play/record, tone, and call progress and on-board E-1 digital interface. Provides E-1 and ISDN Primary Rate connectivity.

D/320SC refers to the Dialogic 32-channel voice board with play/record, tone, and call progress for use with a network interface board.

D/480SC-2T1 refers to the Dialogic DualSpan board with 48 ports of voice processing and 48 ports of network interface.

DIALOG/HD or SpanCard refers to voice and telephone network interface resource boards that communicate via the SCbus. For example, D/160SC, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1, D/320SC, DTI/240SC, DTI/241SC, DTI/300SC and DTI/301SC.

DTI/240SC refers to the 24-channel Dialogic digital telephony interface board providing E-1/DSX-1 and ISDN Primary Rate connectivity to SCbus-based computer telephony systems.

DTI/241SC refers to the 24-channel Dialogic digital telephony interface board with tone and call progress, providing E-1/DSX-1 and ISDN Primary Rate connectivity SCbus-based computer telephony systems.

DTI/300SC refers to the 30-channel 75 or 120 Ohm Dialogic digital telephony interface board providing E-1 and ISDN Primary Rate connectivity SCbus-based computer telephony systems.

DTI/301SC refers to the 30-channel 75 or 120 Ohm Dialogic digital telephony interface board with tone and call progress for E-1 telephony standards. Connects E-1 networks to compatible voice processing boards.

D/xxxSC refers to voice and telephone network interface resource boards that communicate via the SCbus. For example, D/41ESC, D/160SC-LS, D/240SC, D/240SC-T1, D/300SC-E1 and D/320SC.

LSI/81SC refers to the Dialogic 8-channel analog loop start interface board with tone and call progress for the SCbus.

LSI/161SC refers to the Dialogic 16-channel analog loop start interface board with tone and call progress for the SCbus.

MSI/SC refers to the Dialogic 24-channel station interface product for the SCbus. Supports conferencing applications.

MSI/80SC refers to the Dialogic 8-channel station interface product for the SCbus. Supports conferencing applications.

MSI/80SC-R refers to the Dialogic 8-channel station interface product with power ringing and automatic ring trip for the SCbus. Supports conferencing applications.

MSI/160SC refers to the Dialogic 16-channel station interface product for the SCbus. Supports conferencing applications.

MSI/160SC-R refers to the Dialogic 16-channel station interface product with power ringing and automatic ring trip for the SCbus. Supports conferencing applications.

MSI/240SC refers to the Dialogic 24-channel station interface product for the SCbus. Supports conferencing applications.

MSI/240SC-R refers to the Dialogic 24-channel station interface product with power ringing and automatic ring trip for the SCbus. Supports conferencing applications.

SCbus is the TDM (Time Division Multiplexed) bus connecting SCSA (Signal Computing System Architecture) voice, telephone network interface and other technology resource boards together.

SCX160 SCxbus Adapter refers to the Dialogic switchless adapter board that extends the SCbus into other nodes by converting SCbus TTL signals into SCxbus SCSI-3 RS485 type signals, and vice-versa.

SCxbus refers to the standard SCSA bus for communicating between nodes. This bus uses RS485 type signals and a SCSI-3 external interface to transfer SCbus time slot transmissions to other PC hosts.

SpanCard - same as DIALOG/HD.

VFX/40E is a Dialogic SCbus voice and fax resource board with on-board analog loop-start interfaces. The VFX/40E board consists of a D/41E baseboard and a FAX/40E daughterboard that provides 4-channels of enhanced voice and fax services in a single slot.

VFX/40SC is a Dialogic SCbus voice and fax resource board with on-board analog loop-start interfaces. The VFX/40SC board consists of a D/41ESC baseboard and a FAX/40 daughterboard that provides 4-channels of enhanced voice and fax services in a single slot

VFX/40ESC is a Dialogic SCbus voice and fax resource board with on-board analog loop-start interfaces. The VFX/40ESC board consists of a D/41ESC baseboard and a FAX/40E daughterboard that provides 4-channels of enhanced voice and fax services in a single slot. Throughout this document, all references to the D/41ESC board apply to the D/41ESC baseboard component of the VFX/40ESC board.

VFX/40ESCplus is a Dialogic SCbus voice and fax resource board with on-board analog loop-start interfaces. The VFX/40ESCplus board consists of a D/41ESC baseboard enhanced with 128K of DSP RAM and a FAX/40ESC daughterboard that provides 4-channels of enhanced voice and fax services in a single slot.

For additional information on these products, refer to the Dialogic publications listed in *Appendix C - Related Publications*.

1.3. Organization Of This Guide

This guide provides an overview of the Dialogic SCX160 SCxbus Adapter, defines the information required for configuring a multi-node system interconnected via the SCxbus, provides an overview of the functions used to control SCX160 SCxbus Adapter operations and a detailed description of each of these functions, provides error handling information and describes the data structures used by the functions.

Chapter 2 presents an overview of the SCX160 SCxbus Adapter in terms of its features, its operation and how it works to interconnect a multi-node system via the SCxbus.

Chapter 3 provides an overview of the configuration process for a multi-node system and describes the functions performed when configuring each node.

Chapter 4 describes the various SCX160 SCxbus Adapter clock designations and clock states.

Chapter 5 provides an overview of each class of functions and a short description of these functions as a class.

Chapter 6 contains a detailed description of each function used to control SCX160 SCxbus Adapter operations, a programming example for each function, procedures for retrieving error codes returned, and a list of related functions. The functions are presented in alphabetical order.

Chapter 7 describes error handling and retrieval and lists the error codes that can be returned.

Chapter 8 contains information about the data structures used by the various SCX160 SCxbus Adapter functions.

Chapter 9 contains information and suggestions to guide programmers in designing and coding a Dialogic SCX160 SCxbus Adapter application.

Chapter 10 provides instructions for using the demonstration program.

Chapter 11 provides instructions for using the SCX160 InfoTool.

Appendix A describes changing the configuration files so that the master clock node is automatically setup at Dialogic Service startup for data communications between nodes.

Appendix B describes step-by-step procedures for configuring a multi-node system using data stream blocking.

Appendix C lists related publications for further information on the SCX160 SCxbus Adapter and other Dialogic products.

A **Glossary** and an **Index** follow the appendices.

1.4. Installation and Release Notes

The SCX160 SCxbus Adapter Package Release Notes contain information and procedures not documented in the manuals. A paper copy of these Release Notes and an ASCII text file are included with your software package.

The following steps indicate the order in which an SCX160 SCxbus Adapter and Dialogic software for Windows should be installed, checked, and programmed for each node.

1. Prepare and install the SCX160 SCxbus Adapter in your PC host following the procedure in the Quick Install for the SCX160 SCxbus Adapter card.
2. Install the necessary software and device drivers in the PC host as described in the installation procedure.
3. Refer to this SCX160 SCxbus Adapter User's Guide for Windows to develop application programs.

To use software for other Dialogic devices, refer to the appropriate software reference for specific instructions (see *Appendix C - Related Publications*).

2. Product Overview

The SCX160 SCxbus Adapter introduces a hardware and software platform that forms the basis for SCSA Multi-Node Network Architecture (MNA). MNA allows you to interconnect similar or dissimilar voice processing systems to form a single distributed system image at the application level, thus breaking the single host limitation.

SCbus products or devices communicate with each other via the SCbus. All such products have one or more transmit (TX) channels and receive (RX - listen) channels. At Dialogic Service startup, each SCbus time slot is assigned to a corresponding SCxbus transmit time slot. Therefore, only a single device channel transmits on a single SCbus transmit time slot. To receive data, a device listens to the SCbus time slot connected to another device's transmit channel.

The SCX160 SCxbus Adapter incorporates a single stage, non-switching architecture. This architecture connects each data stream (bundle) of SCbus time slots on a one-to-one basis across the SCxbus so that each node can communicate with every other node by listening to each other's transmit time slot. This architecture takes advantage of the inherent switching capabilities of the SCbus. Included run-time drivers monitor operations in real-time and provide fault detection and troubleshooting functions.

The SCX160 SCxbus Adapter interfaces the SCbus transmit time slots to corresponding numbered SCxbus transmit time slots so all data transmitted on these time slots can be received at all other nodes in the system. Likewise, the SCX160 SCxbus Adapter interfaces the transmissions received from other nodes via the SCxbus to like numbered time slots on the node's internal SCbus. Therefore, any device can listen to any other device in any node by listening to the SCbus transmit time slot assigned to this other device.

2.1. SCX160 SCxbus Adapter Features

The SCX160 SCxbus Adapter converts internal SCbus TTL signals into external SCxbus RS485 type signals, and vice-versa, for communications between SCSA devices installed in up to 16 PC hosts. Each SCX160 SCxbus Adapter is assigned a unique node identification number.

The SCX160 SCxbus Adapter platform incorporates the following:

- 1024 channel multi-node single stage SCxbus interconnect that enables developers to run an application across multiple PCs (nodes)
- Switchless design that takes advantage of the inherent switching capabilities of the SCbus
- Standard Dialogic ISA bus host interface with Board Locator Technology (BLT) circuits
- SCxbus compliant interface with transceivers capable of connecting up to 16 nodes within an overall SCSI-3 cable length of 15 meters
- Node hot plugability, which allows a node to be attached to or removed from the SCxbus without corrupting live traffic or disturbing the entire system
- Controlled slip stream buffers
- Fault tolerant clocking, which provides robust network synchronization
- SCxbus dual clock scheme, which increases system available time
- Background diagnostics that continually monitor clock, frame and data integrity
- Fault reports when user-defined thresholds are exceeded
- Programmable interrupts
- A shared RAM interface

The SCX160 SCxbus Adapter supports the following:

- Quick access to board parameter information with the SCX160 InfoTool
- Downloadable parameter file

- SCbus and SCxbus time slot assignment under control of the Dialogic Service Startup program
- Data channels (time slots) where the full bandwidth is used for data transmission; no data bandwidth is wasted for synchronization, signaling, or administration tasks
- Real time diagnostics that run continuously
- Data stream blocking if the total time slot requirements of your multi-node system exceeds the 1024 time slot limitation of the SCxbus

At Dialogic Service startup the SCX160 SCxbus Adapter runs a start-up test program that verifies on-board functionality. The start-up test program executes the following tests:

- Memory test
- Program signature and CRC check
- Sanity check with the host computer

In addition to these diagnostics, the SCX160 SCxbus Adapter continuously monitors operations and reports alarms when they are software enabled and when user-defined thresholds are exceeded. Alarms are transmitted to the application to be handled by the application or the system administrator. Alarms report the following kinds of problems:

- Data transmission and reception integrity tests based on user-defined thresholds on both the SCbus and the SCxbus
- SCbus and SCxbus clock integrity
- Non-recoverable SCxbus or SCbus clock error
- Excessive slip: more occurrences than a user-defined threshold
- Loss of primary SCxbus clock
- Loss of SCbus clock
- Initiation of system clock recovery

2.1.1. Specifications

The following table provides the technical specifications for the SCX160 SCxbus Adapter.

Table 1. SCX160 Technical Specifications

Max. nodes/system	16
Resource sharing bus	SCbus and SCxbus
Control microprocessor	80C186
HOST INTERFACE:	
Bus compatibility	IEEE P996 ISA compatible (IBM PC AT)
Bus speed	12.5 MHz maximum
Bus mode	Automatically configures to 8 or 16 bit transfer mode
Shared memory	32 Kbyte page
Base addresses	8000h, on 32K boundaries. All Board Locator Technology (BLT) boards share the same base address. Shared memory is page mapped in/out dynamically as needed.
Interrupt level	IRQ 2/9, 3, 4, 5, 6, 7, 10, 11, 12, 14, 15, software selectable. Only one IRQ line must be shared by all BLT boards.
I/O ports	None
POWER REQUIREMENTS:	
+5 VDC	3.40 A max.
-5 VDC	0.170 A max.
+12 VDC	1.275 A max.
-12 VDC	0.255 A max.
Operating temperature	0 °C to +50 °C
Storage temperature	-20 °C to +70 °C
Humidity	5% to 85% non-condensing
Form factor	PC AT, 13.3 in. long, 0.793 in. wide (total envelope), 4.5 in. high (excluding edge connector)

SAFETY & EMI CERTIFICATIONS:

United States	FCC Part 15, Subpart B, Class A UL: 1950 (E96804)
Canada	UL: CSA C22.2 No. 950
European Community	EN55022 Class B
World-Wide	For specific country approval designation, see the Dialogic International Approvals data sheet or contact a Sales Engineer.
Warranty	3 years standard

2.2. SCX160 SCxbus Adapter in a Multi-Node System

The SCX160 SCxbus Adapter converts SCbus TTL signals into SCxbus RS485 type signals, and vice-versa, for inter-node communications between SCSA devices. This adapter uses the inherent switching capabilities of SCSA products (along with the SC2000 ASIC) to perform the necessary switching functions.

For illustrative purposes, the call processing, shared resource, multi-node call switching system shown in *Figure 1. Multi-node Example* is used to highlight PC host communications via the SCxbus.

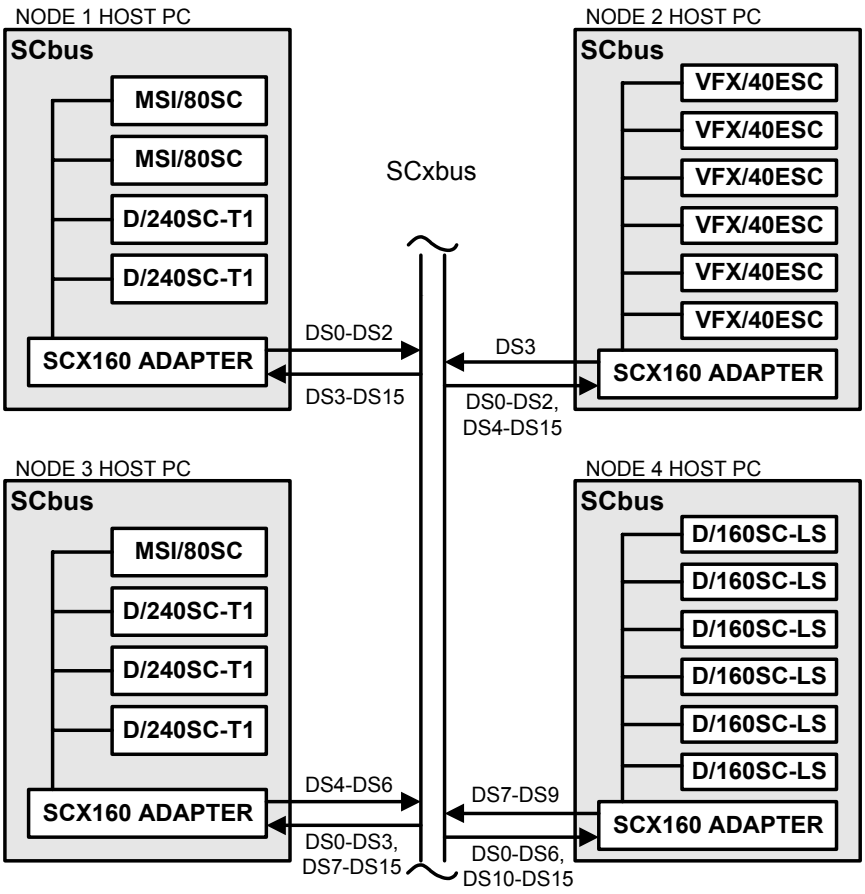

Figure 1. Multi-node Example

In the example presented in *Figure 1. Multi-node Example*, the nodes are defined as follows:

- Node 1 comprises an SCX160 board, 2 MSI/80SC boards (80 time slots; 64 for conferencing and 16 for stations) and 2 D/240SC-T1 boards (96 time slots; 48 for the voice devices and another 48 for the E-1 digital time slots) for a total of 176 time slots.
- Node 2 comprises an SCX160 board and 6 VFX/40ESC boards setup in resource mode (analog devices on these boards are disabled) for a total of 48 time slots (24 for the voice/FAX devices and 24 for the disabled analog devices).
- Node 3 comprises an SCX160 board, 1 MSI/80SC board (40 time slots) and 3 D/240SC-T1 boards (72 voice time slots and 72 E-1 digital time slots) for a total of 184 time slots.
- Node 4 comprises an SCX160 board and 6 D/160SC-LS boards for a total of 192 time slots (96 for the voice devices and 96 for the analog devices).

Time slots on the SCbus and SCxbus are transmitted in bundles of 64 across each of 16 lines for a total of 1024 time slots/bus. Each bundle of time slots is handled as a data stream (DS0 through DS15) by the SCX160 SCxbus Adapter. The SCX160 SCxbus Adapter maps the SCbus transmit data streams onto corresponding numbered SCxbus data streams, and vice-versa, and then buffers the transmit data streams between the two buses. No data streams or time slots are switched by the SCX160 SCxbus Adapter.

At Dialogic service startup, the transmit time slots required to support the SCSA products installed or allocated to each node are determined and then the appropriate quantity of data streams (bundles of 64 time slots each) are assigned to each node. In our example, the configuration established could be as follows:

- Node 1 uses 176 transmit time slots, so DS0, DS1 and DS2 comprising time slots 0 through 191 are assigned to this node.
- Node 2 uses 48 time slots, so DS3 comprising time slots 192 through 255 is assigned to this node.
- Node 3 uses 184 transmit time slots, so DS4, DS5 and DS6 comprising time slots 256 through 447 are assigned to this node.

- Node 4 uses exactly 192 transmit time slots, so DS7, DS8 and DS9 comprising time slots 448 through 639 is assigned to this node.

NOTE: DS10 through DS15 are not used on the SCxbus in this example.

In the example above, the number of data streams allocated for each node are the minimum number required. However, you can allocate more data streams than are required to leave room for future expansion.

See *Table 2. Data Stream Versus Time Slot Bundling* for the time slot bundles assigned to each Data Stream (DS); these numbers are not used, nor needed, when programming your application.

Table 2. Data Stream Versus Time Slot Bundling

Data Stream	Time Slot Bundle	Data Stream	Time Slot Bundle
DS0	0 to 63	DS8	512 to 575
DS1	64 to 127	DS9	576 to 639
DS2	128 to 191	DS10	640 to 703
DS3	192 to 255	DS11	704 to 767
DS4	256 to 319	DS12	768 to 831
DS5	320 to 383	DS13	832 to 895
DS6	384 to 447	DS14	896 to 959
DS7	448 to 511	DS15	960 to 1023

An SCSA device in any node connects to an SCSA device in any other node by listening to the transmit time slot assigned to that SCSA device. The user application is expected to know the configuration of each node in the system and to coordinate the transfer of SCbus transmit time slot information among nodes.

For example, to connect a call on channel 1 of the first D/240SC-T1 board (assume SCbus transmit time slot 0) in node 1 to channel 4 of the first VFX/40ESC fax resource board (assume SCbus transmit time slot 195) in node 2,

the application would make these SCSA devices listen to each other by issuing individual SCbus routing function calls in each node[1]. The SCbus convenience functions cannot be used in a multi-node configuration due to the need to communicate time slot information across nodes. The following example describes how to establish communications between these devices:

- At node 1, initialize the SC_TSINFO structure with the required information.
- At node 1, get the SC_TSINFO structure that contains the number of the SCbus time slot connected to digital transmit channel 1 on the first D/240SC-T1 board by issuing a **dt_getxmitslot( )** call.
- Pass this SC_TSINFO structure to node 2; for example by using a socket message.
- At node 2, connect FAX receive channel 4 of the first VFX/40ESC fax resource board to digital transmit channel 1 [SCbus transmit time slot 0] on the first D/240SC-T1 board in node 1 by issuing an **fx_listen( )** call. This function uses as input the information contained in the SC_TSINFO structure returned by the node 1 **dt_getxmitslot( )** call and passed to node 2.
- At node 2, get the SC_TSINFO structure that contains the number of the SCbus time slot connected to FAX transmit channel 4 of the first VFX/40ESC fax resource board by issuing an **fx_getxmitslot( )** call.
- Pass this SC_TSINFO structure to node 1; for example by using a socket message.
- At node 1, connect digital receive channel 1 of the first D/240SC-T1 board to FAX transmit channel 4 [SCbus transmit time slot 195] on the first VFX/40ESC fax board in node 2 by issuing a **dt_listen( )** call. This function uses as input the information contained in the SC_TSINFO structure returned by the node 2 **fx_getxmitslot( )** call and passed to node 1.

When these functions return, full duplex communications between the FAX and digital time slots will be established.

---

[1] The SCbus convenience functions cannot be used in a multi-node configuration due to the need to communicate time slot information across nodes.

2.3. Blocking Data Streams

If the total time slot requirement of your multi-node system exceeds the 1024 time slot limitation of the SCxbus, SCbus data streams (bundles of time slots) could be blocked within a node from being transmitted onto the SCxbus. In this blocking scenario, up to 4,096 time slots can be made available in the multi-node system.

The resources (e.g., voice, analog or digital) whose time slots are blocked from transmitting onto the SCxbus can be shared locally within their local node. The SCxbus data streams that correspond to these locally blocked time slots are not used. Therefore, no data is transmitted or received on these data streams by the SCX160 SCxbus Adapter.

NOTE: The data stream blocking scheme increases the complexity of resource management and should only be used for multi-node systems that exceed the 1024 time slot limitation. Blocking only increases the effective number of time slots in the multi-node system and does not change the 1024 time slot limitation of the SCbus in each node.

In *Figure 2. Blocking Example,* voice resource time slots at each of the four nodes are blocked from transmitting onto the SCxbus. Since Node 3 requires the largest number of data streams (three) to block the 144 time slots used by the local voice resources, three data streams must be blocked in all nodes. In this example, all voice resources are limited to resource sharing in their local nodes while all other resources can communicate and be shared among all nodes.

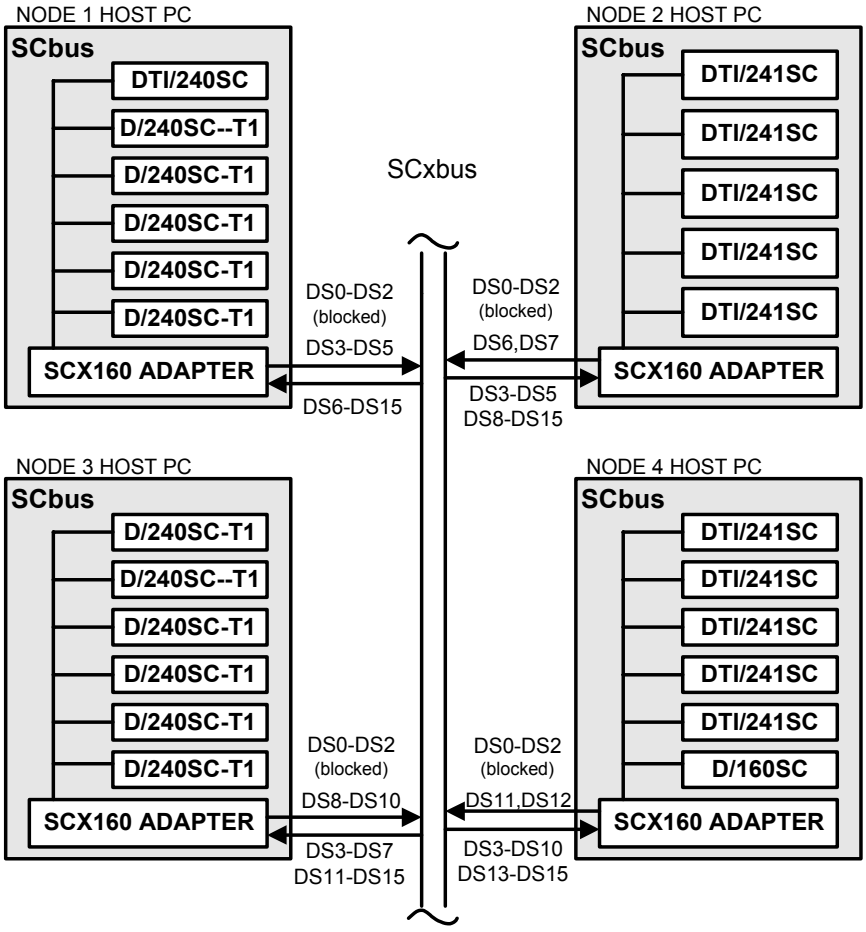

Figure 2. Blocking Example

2.3.1. Blocking Considerations

To configure blocking requires knowing:

- The configuration of each node (the boards installed and resources available on each board). Refer to the Dialogic Node Configuration Chart in the SCX160 SCxbus Adapter Package Release Notes.
- The resources to be blocked in each node from the SCxbus (e.g., voice, analog or digital).
- The number of data streams to be blocked at each node.

Refer to *Appendix B - Dialogic SCX160 Configuration Program* for procedures on how to set up blocking. The following considerations apply to blocking:

- Only the time slots for one resource (e.g., voice, analog or digital) can be blocked per node.
- When blocking time slots for a resource in a node, the number of data streams blocked must include all the time slots being used for that resource (e.g., if five D/240SC - T1 boards reside in Node 1 and voice resources are blocked then two data streams (128 time slots) must be blocked from the SCxbus because the voice resources on these boards require 120 time slots).
- Time slots for a different resource can be blocked in each node (e.g., Node 1 can block time slots for the voice resource and Node 2 can block time slots for the digital resource).
- The number of data streams to be blocked from the SCxbus must be the same for each node in the system. This number must be equal to or greater than the largest number of data streams to be blocked in any node.

2.3.2. Blocking Example

The configuration presented in *Table 3. Multi-node Configuration Without Blocking Example* illustrates a multi-node system that exceeds the SCxbus 1024 time slot limitation. The multi-node system comprises four nodes that contain one or more voice and network boards in addition to an SCX160 Adapter board. As *Table 3. Multi-node Configuration Without Blocking Example* illustrates, without blocking, the multi-node system exceeds the 16 data stream (1024 time slots) limitation of the SCxbus.

Table 3. Multi-node Configuration Without Blocking Example

Node	Node Configuration	DTI Time Slots	Voice Time Slots	Total Time Slots / Data Streams Used	# of SCxbus Data Streams
1	5xD/240SC-T1 + 1xDTI/240SC + 1xSCX/160	5x24+ 1x24=144	5x24=120 (2 data streams)	144+120=264 (**5** data streams)	**5**
2	5xDTI/241SC + 1xSCX/160	5x24=120	5x24=120 (2 data streams)	120+120=240 (**4** data streams)	**4**
3	6xD/240SC-T1 + 1xSCX/160	6x24=144	6x24=144 (3 data streams)	144+144=288 (**5** data streams)	**5**
4	5xDTI/241SC + 1xD/160SC + 1xSCX/160	5x24=120	5x24+ 1x16=136 (3 data streams)	120+136=256 (**4** data streams)	**4**
				TOTAL =	**18**

However, all the resources of this example can be used by the multi-node system by blocking the voice resources from the SCxbus in each node. Since the largest number of data streams required for voice resources is three data streams in Nodes 3 and 4 (refer to *Table 4. Determining Number of Data Streams to Block*), three data streams must be blocked in each node in the system.

Table 4. Determining Number of Data Streams to Block

Node	# of Voice Time Slots Used	# of Data Streams Required
1	120	2
2	120	2
3	144	3
4	136	3

In *Table 5. Multi-node Configuration With Blocking Example*, the voice resources are blocked from the SCxbus in each node. With blocking, all desired resources can be used while only using ten SCxbus data streams. Ten data streams are actually transmitting on the SCxbus for a total of 640 time slots and three data streams (equivalent to 192 time slots) are blocked in each node. In this system, 1,048 time slots are actually being used by the currently installed resources. However, the system has the potential to use up to 832 (13 data streams x 64) SCxbus time slots plus 768 local SCbus time slots (192 x 4 nodes) for a total of 1,590 time slots.

Table 5. Multi-node Configuration With Blocking Example

Node	Node Configuration	DTI Time Slots	Voice Time Slots	Total Time Slots / Data Streams Used	# of SCxbus Data Streams
1	5xD/240SC-T1 + 1xDTI/240SC + 1xSCX/160	5x24+ 1x24=144	5x24=120 (2 data streams)	144+120=264 (**5** data streams)	**3**

Node	Node Configuration	DTI Time Slots	Voice Time Slots	Total Time Slots / Data Streams Used	# of SCxbus Data Streams
2	5xDTI/241SC + 1xSCX/160	5x24=120	5x24=120 (2 data streams)	120+120=240 (**4** data streams)	**2**
3	6xD/240SC-T1 + 1xSCX/160	6x24=144	6x24=144 (3 data streams)	144+144=288 (**5** data streams)	**3**
4	5xDTI/241SC + 1xD/160SC + 1xSCX/160	5x24=120	5x24+ 1x16=136 (3 data streams)	120+136=256 (**4** data streams)	**2**
				TOTAL =	**10**

2.4. Functional Description of an SCX160 SCxbus Adapter

Time slots on the SCbus and SCxbus are transmitted in bundles of 64 across each of 16 lines for a total of 1024 time slots/bus. Each bundle of time slots is handled as a data stream by the SCX160 SCxbus Adapter. Each of these 16 data streams is interfaced to the SCX160 SCxbus Adapter by one of 16 data transceivers (a driver and receiver pair). Therefore, each transceiver handles 64 time slots.

As shown in *Figure 3*, SCbus data stream 0 outbound (DS0 OUT) comprising time slots 0 to 63, enters the SCX160 SCxbus Adapter via receiver 1A. All interface receivers are permanently enabled. DS0 OUT is then applied to an SCxbus FIFO (First-In, First-Out) elastic buffer and then via an SCxbus interface driver 1B to the SCxbus.

Data streams transmitted across the SCxbus are handled in a similar fashion. As shown in *Figure 3*, SCxbus data stream 1 inbound (DS1 IN) comprising time slots 64 to 127, enters the SCX160 SCxbus Adapter via receiver 2B. These interface receivers are also permanently enabled. DS1 IN is then applied to an SCbus FIFO elastic buffer and then via an SCbus interface driver 2A to the SCbus.

2.4.1. FIFO Elastic Buffers

The SCbus and SCxbus FIFO elastic buffers minimize any impact due to slip by controlling frame slip when a node uses independent clock. These buffers provide elasticity by absorbing clock drift so that slip is restricted to one PCM (Pulse Code Modulated) frame added or dropped. The FIFO buffer design incorporates dual-port zero fall-through time, asynchronous or coincident read and write clocks, and a FIFO depth of two PCM frames. A FIFO control circuit locks both the read and write clocks of the FIFO elastic buffers in both phase and frequency (synchronous operation) to their respective bus.

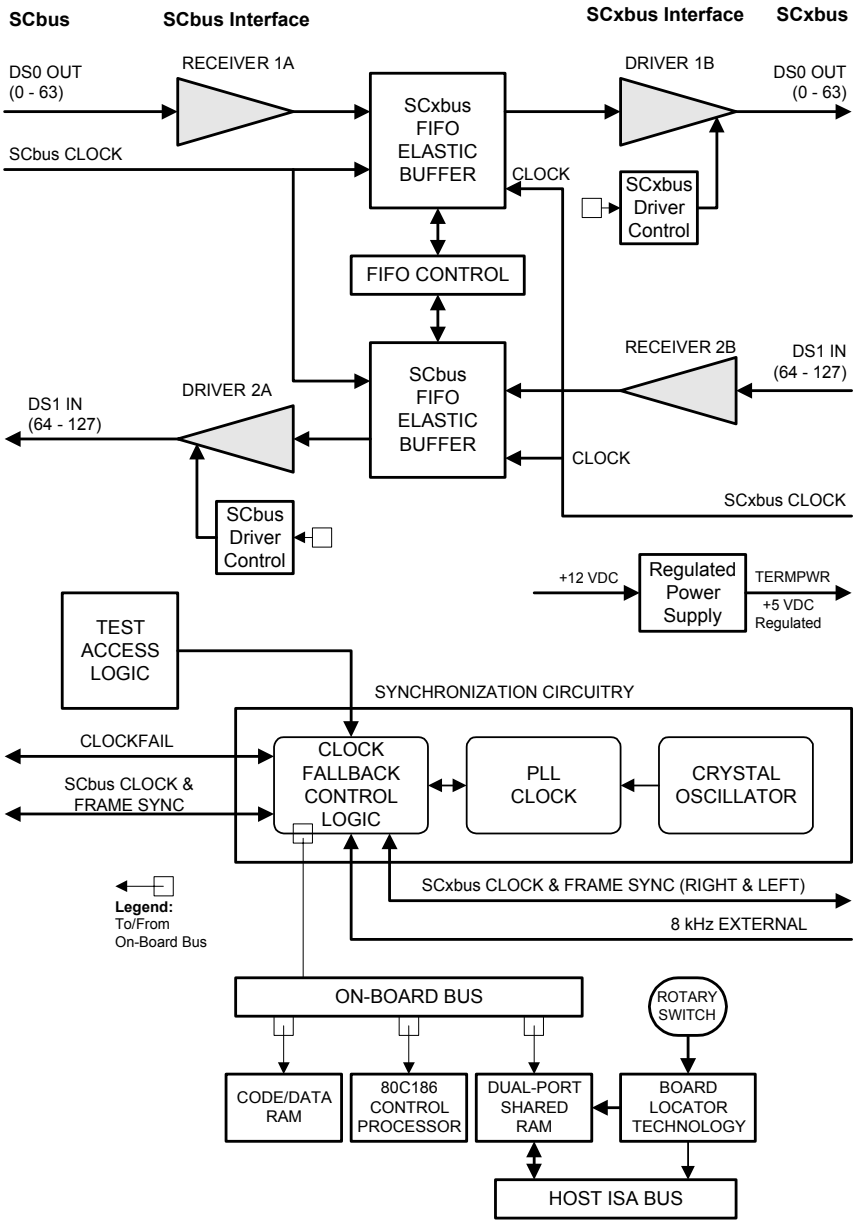
SCbus
SCbus Interface
SCxbus Interface
SCxbus
DS0 OUT
(0 - 63)
RECEIVER 1A
SCxbus
FIFO
ELASTIC
BUFFER
DRIVER 1B
DS0 OUT
(0 - 63)
SCbus CLOCK
CLOCK
SCxbus
Driver
Control
FIFO CONTROL
RECEIVER 2B
DS1 IN
(64 - 127)
SCbus
FIFO
ELASTIC
BUFFER
DRIVER 2A
DS1 IN
(64 - 127)
CLOCK
SCxbus CLOCK
SCbus
Driver
Control
+12 VDC
Regulated
Power
Supply
TERMPWR
+5 VDC
Regulated
TEST
ACCESS
LOGIC
SYNCHRONIZATION CIRCUITRY
CLOCKFAIL
SCbus CLOCK &
FRAME SYNC
CLOCK
FALLBACK
CONTROL
LOGIC
PLL
CLOCK
CRYSTAL
OSCILLATOR
SCxbus CLOCK & FRAME SYNC (RIGHT & LEFT)
8 kHz EXTERNAL
Legend:
To/From
On-Board Bus
ON-BOARD BUS
ROTARY
SWITCH
CODE/DATA
RAM
80C186
CONTROL
PROCESSOR
DUAL-PORT
SHARED
RAM
BOARD
LOCATOR
TECHNOLOGY
HOST ISA BUS

Figure 3. SCX160 SCxbus Adapter Functional Block Diagram

If a FIFO elastic buffer fills, normal operation continues after the start of the next frame. During this frame all writes are inhibited, thus bringing the FIFO elastic buffer to about half full and one full frame is thrown away (positive slip). The FIFO elastic buffer is now about half full and normal operations are resumed.

If a FIFO elastic buffer empties, a full frame of data is repeated (negative slip) to bring the buffer back to about half full.

Each FIFO elastic buffer is continuously tested by inserting an on-board generated test pattern in unused data stream 17 and checking the test pattern at the output of the FIFO elastic buffer.

2.4.2. SCxbus Interface

In addition to the transceivers, the SCxbus interface comprises SCSI-3 compliant receiver fail-safe circuits, EMI protection, term power and a 68-pin female connector mounted on the module's faceplate. A regulated +5 VDC is provided to the TERMPWR pin of the SCxbus connector for power sharing among all active SCX160 SCxbus Adapters connected to the SCxbus.

2.4.3. On-Board Control Processor

An on-board 80C186 Control Processor manages all operations of the SCX160 SCxbus Adapter via an on-board bus and interprets and executes commands from the host PC. This microprocessor handles real-time events, communications and interrupts with the host PC to provide faster system response time. Communications between this Control Processor and the host PC are via the dual port shared RAM that acts as an input/output buffer. This RAM interfaces to the host PC via the AT (ISA) bus. All operations are interrupt driven to meet the demands of real-time systems. When the node is initialized, firmware to control all board operations is downloaded from the host PC to the on-board code/data RAM. This downloadable firmware gives the board all of its intelligence and enables easy feature enhancement and upgrades.

2.4.4. Synchronization Circuitry

The synchronization circuitry comprises a Crystal Oscillator, a PLL Clock and Clock Fallback Control Logic circuits. The Crystal Oscillator provides a local clock source with a ±20 ppm stability to be used as a reference frequency in the event of a clock reference failure.

The PLL Clock maintains clock synchronization and frame synchronization (frame alignment) using one of the following as the reference clock source:

- Right SCxbus clock
- Left SCxbus clock
- 8 kHz external reference clock from the SCxbus
- SCbus clock
- Local Crystal Oscillator (default)

The on-board Control Processor generates the control signals that select the appropriate reference clock source. An alarm event is generated if an error is detected in any clock.

At Dialogic Service startup, one node must be designated as the SCxbus master clock node. This node drives the Right SCxbus clock line. Other boards in this node, including the SCX160 SCxbus Adapter, can be designated as SCxbus fallback masters for the SCbus master clock. The SCbus master clock should be designated as the clock reference to be used by the SCX160 SCxbus Adapter to synchronize clock sent over the Right SCxbus clock line.

Another node should be designated as the SCxbus fallback master clock node. This node phase locks to the Right SCxbus clock and drives the Left SCxbus clock line. The SCX160 SCxbus Adapter in all other nodes synchronize to the Right SCxbus clock. Typically, a network module in this node is designated to provide SCbus master clock.

NOTE: For more on clock designations, see *Chapter 4. Clocking*.

A clockfail signal from the SCbus in the master clock node indicates the status of the SCbus master clock source. If this clock fails, then the SCxbus master clock node stops driving the Right SCxbus clock line if the Left SCxbus clock is present. When the other nodes in the system detect loss of Right SCxbus clock, they automatically switch their clock synchronization to the Left SCxbus clock. The SCxbus fallback master clock node continues to drive the Left SCxbus clock independent of the Right SCxbus clock.

Right and Left Frame Synchronization signals are also supported on the SCxbus and are in phase synchronization. A Frame Synchronization signal is also available from the SCbus. These Frame Synchronization signals are used by the system clock circuits and the FIFO Control circuits.

The 8 kHz external reference clock can be used as an independent clock source. Typically, this clock is a composite station clock as seen in a CO (Central Office) environment.

2.4.5. Board Locator Technology Circuit

The Board Locator Technology circuit operates in conjunction with a rotary switch to determine and set Shared RAM memory base address in the host PC and non-conflicting slot (board identification number from 0 to 31) and IRQ interrupt-level parameters. This feature eliminates the need to set jumpers or DIP switches. A board locating sequence can locate each SCX160 SCxbus Adapter in the system and assign each to a unique address in the host PC or to the same base address. If the SCX160 SCxbus Adapter detects a board enable sequence that does not match its board identification, the SCX160 SCxbus Adapter automatically disables its Shared RAM. This feature allows the host PC to enable a specific SCX160 SCxbus Adapter for host PC communications without having to disable previously enabled boards.

2.5. SCbus/SCxbus Data Stream Connections

Each of the 16 data streams on the SCbus is interfaced to the SCX160 SCxbus Adapter via a corresponding data transceiver, as shown in *Figure 4*. Each SCbus interface transceiver comprises individually controlled tri-state drivers and permanently enabled receivers. These drivers are controlled by the on-board 80C186 Control Processor. The firmware only enables those drivers that will transmit onto the SCbus or the SCxbus.

Each of the 16 data streams on the SCxbus is interfaced to the SCX160 SCxbus Adapter via a corresponding, individually controlled, RS-485 data transceiver (tri-state drivers and permanently enabled receivers) (see *Figure 4*).

At Dialogic service startup, each data stream (DS0 through DS15) is configured to transmit data from the SCbus to the SCxbus, or to transmit data from the SCxbus to the SCbus.

To illustrate, assume that SCbus data stream 0 outbound (DS0 OUT), comprising time slots 0 to 63, enters the SCX160 SCxbus Adapter via SCbus data transceiver receiver 1A (see *Figure 4*). DS0 OUT is then applied to the SCxbus FIFO elastic buffer and then via the SCxbus data transceiver driver 1B to the SCxbus. Since DS0 is transmitting into the SCbus interface of the SCX160 SCxbus Adapter, the SCbus data transceiver driver 1A is disabled by the Control Processor to guard against inadvertently corrupting this data stream. At the SCxbus data transceiver interface, both driver 1B and receiver 1B are enabled. Driver 1B transmits the data stream onto the SCxbus and to receiver 1B. Receiver 1B output is monitored by the Test Access Logic.

To illustrate transmitting data from the SCxbus to the SCbus, assume that SCxbus data stream 1 inbound (DS1 IN), comprising time slots 64 to 127, enters the SCX160 SCxbus Adapter via SCxbus data transceiver receiver 2B (see *Figure 4*). DS1 IN is then applied to the SCbus FIFO elastic buffer and then via the SCbus data transceiver driver 2A to the SCbus. Since DS1 is transmitting into the SCxbus interface of the SCX160 SCxbus Adapter, the SCxbus data transceiver driver 2B is disabled by the Control Processor to guard against inadvertently corrupting this data stream. At the SCbus data transceiver interface, both driver 2A and receiver 2A are enabled. Driver 2A transmits the data stream onto the

SCbus and to receiver 2A. Receiver 2A output is monitored by the Test Access Logic.

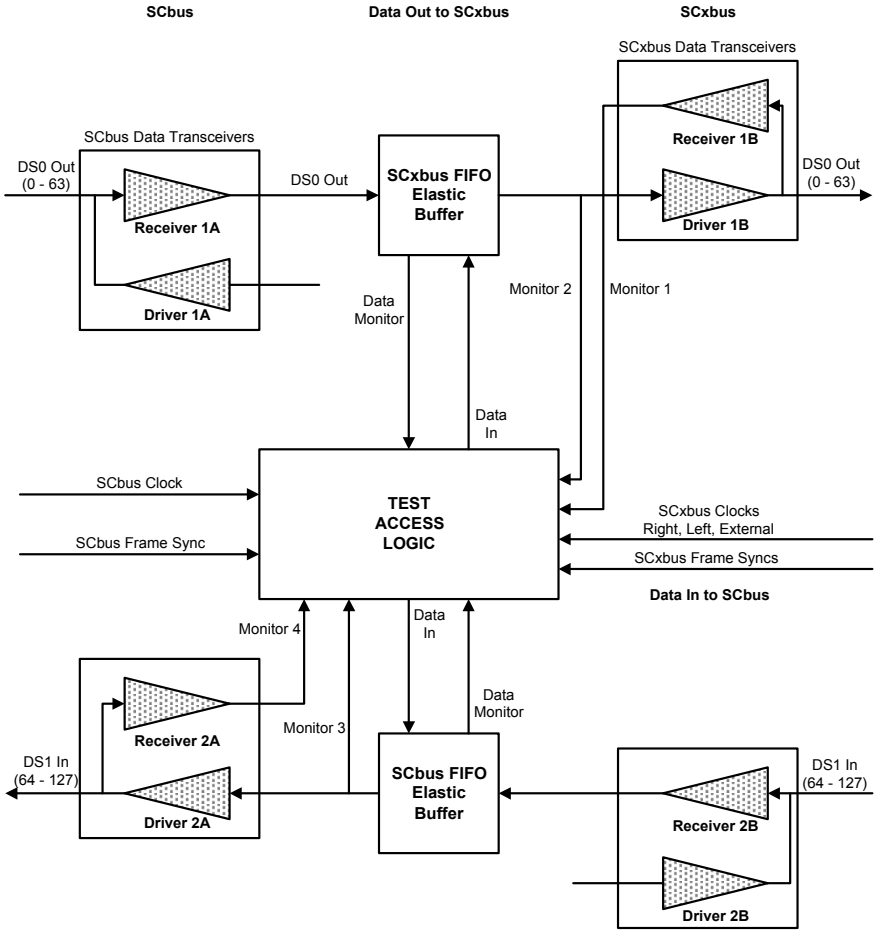

Figure 4. SCbus/SCxbus Data Stream Connections & Test Access

2.6. Test Access Logic

The Test Access Logic circuits perform the following functions:

- Continuously testing for errors in the SCbus and SCxbus data streams at the interface drivers and at the FIFO elastic buffers
- Monitoring the status of the synchronization circuitry

The Test Access Logic circuits compare data transmitted over each SCbus and SCxbus driver line with data echoed on a corresponding receiver line to continuously verify data integrity for each data stream. If different, an error counter is incremented and this count is saved until read and cleared by the Control Processor. In addition, these circuits monitor the status of clock signals and various programming status indicators.

Using the interface circuits as an example, the Test Access Logic connections can be configured as shown in *Figure 4*. Then data transmissions can be monitored as follows:

- To the SCxbus: DS0 transmissions from the SCbus are transmitted to the SCxbus by SCxbus data transceiverdriver 1B and are also input to receiver 1B. A monitor 1 line inputs the DS0 input of driver 1B (output of the SCxbus FIFO elastic buffer) to the Test Access Logic circuits where it is compared on a bit-by-bit basis with the DS0 data output received via the monitor 2 line from receiver 1B. If different, the error counter is incremented.
- From the SCxbus: DS1 transmissions from the SCxbus are transmitted to the SCbus by SCbus data transceiver driver 2A and also input to receiver 2A. A monitor 3 line inputs the DS1 input to driver 2A to the Test Access Logic circuits where it is compared on a bit-by-bit basis with the DS1 data output received via monitor 4 line from receiver 2A. If different, the error counter is incremented.

The Test Access Logic circuits monitoring capabilities as illustrated above apply to all enabled SCbus and SCxbus data transceivers.

2.7. Troubleshooting the SCxbus Cabling

Communications and clocking on the SCxbus can be monitored for troubleshooting purposes by connecting your monitoring device to the connector pins listed in *Table 6. SCxbus Pin-Outs*. This table lists the SCxbus signals versus connector pin-outs and provides a description of each signal. All cabling, connectors and signaling conform to the requirements defined in The SCSA Hardware Model Specification, Version 3.0, available from Dialogic Corporation. The binary state of line is as follows:

- 1 = "- *SCxbus signal*" terminal is negative with respect to the "+ *SCxbus signal*" terminal
- 0 = "- *SCxbus signal*" terminal is positive with respect to the "+ *SCxbus signal*" terminal.

The SCX160 SCxbus Adapters are connected to each other via a 34 signal twisted pair SCxbus Y-cable. Both ends of the cable must be terminated to meet RS485 requirements. All signals on the SCxbus cable are common between all connected devices. See *Figure 5. SCxbus Cabling, Typical*, for typical SCxbus cabling between nodes.

Table 6. SCxbus Pin-Outs

Pin	SCxbus Signal	Pin	SCxbus Signal	Description
1	- SCxSD0	35	+ SCxSD0	BSDS (Bi-directional Serial Data Stream) #1
2	- SCxSD1	36	+ SCxSD1	BSDS #2
3	- SCxSD2	37	+ SCxSD2	BSDS #3
4	- SCxSD3	38	+ SCxSD3	BSDS #4
5	- SCxSD4	39	+ SCxSD4	BSDS #5
6	GND	40	GND	Connect to PCB
7	- SCxSD5	41	+ SCxSD5	BSDS #6
8	- SCxSD6	42	+ SCxSD6	BSDS #7
9	- SCxSD7	43	+ SCxSD7	BSDS #8
10	- SCxSD8	44	+ SCxSD8	BSDS #9
11	- SCxSD9	45	+ SCxSD9	BSDS #10
12	- SCxSD10	46	+ SCxSD10	BSDS #11
13	- SCxSD11	47	+ SCxSD11	BSDS #12
14	- SCxSD12	48	+ SCxSD12	BSDS #13
15	- SCxSD13	49	+ SCxSD13	BSDS #14
16	N.C.	50	GND	Connect to PCB
17	TERMPWR	51	TERMPWR	Termination power
18	TERMPWR	52	TERMPWR	Termination power
19	N.C.	53	N.C.	No Connection
20	- KREF	54	+8 KREF	8 kHz reference clock

Pin	SCxbus Signal	Pin	SCxbus Signal	Description
21	GND	55	GND	Connect to PCB
22	- L_SCxCLK	56	+ L_SCxCLK	System clock, left
23	- L_SCxFSYNC	57	+ L_SCxFSYNC	Frame sync, left
24	- SCxMC	58	+ SCxMC	Message channel
25	- SCxSD20	59	+ SCxSD20	BSDS #21
26	- SCxSD19	60	+ SCxSD19	BSDS #20
27	- SCxSD18	61	+ SCxSD18	BSDS #19
28	- SCxSD17	62	+ SCxSD17	BSDS #18
29	- SCxSD16	63	+ SCxSD16	BSDS #17
30	GND	64	GND	Connect to PCB
31	- SCxSD15	65	+ SCxSD15	BSDS #16
32	- SCxSD14	66	+ SCxSD14	BSDS #15
33	- R_SCxCLK	67	+ R_SCxCLK	System clock, right
34	- R_SCxFSYNC	68	+ R_SCxFSYNC	Frame sync, right

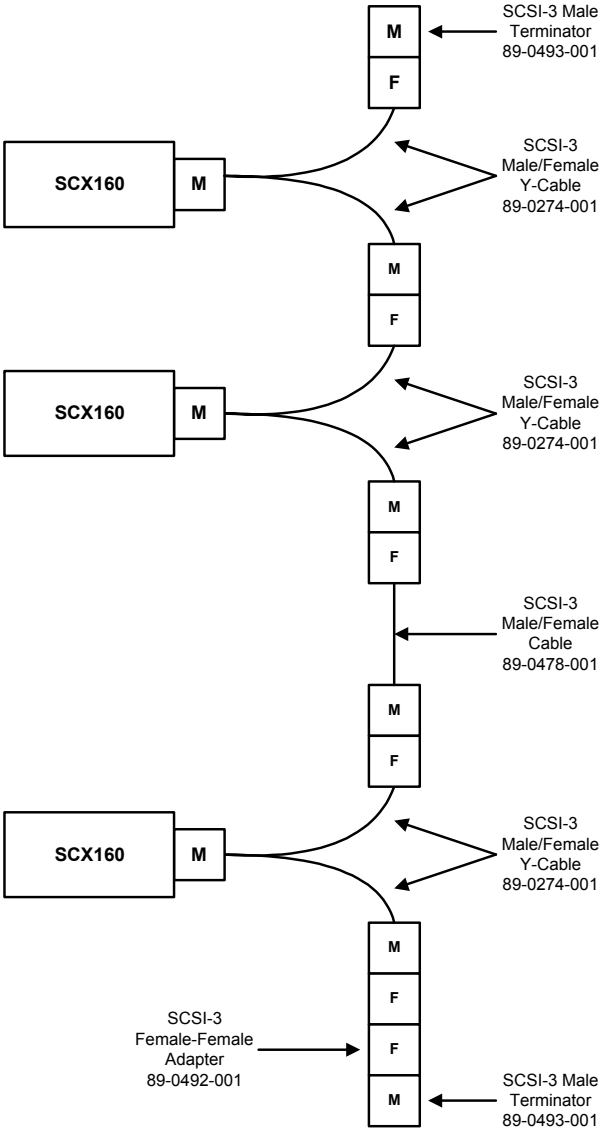

Figure 5. SCxbus Cabling, Typical

2.8. Diagnostic Tools

The following diagnostic tool is available to use for the SCX160 SCxbus Adapter:

- The SCX160 InfoTool displays SCbus and SCxbus clock status and data stream activity status. This information aids in troubleshooting system problems and failures. The SCX160 InfoTool can be run on an as needed basis during normal system operations.

For details on this diagnostic tool, refer to *Chapter 11. SCX160 Diagnostics*.

3. Configuration Overview

An overview of the process for installing the hardware and software and for configuring each node in a multi-node system, including the assignment of SCbus time slots, is presented in this chapter.

3.1. Overview of Installation Procedures

Step-by-step procedures for installing the software and configuring each node are located in the SCX160 SCxbus Adapter Package Release Notes. Procedures for installing the hardware are located in the Quick Install for the SCX160 SCxbus Adapter card.

The SCX160 SCxbus Adapter installation process includes the following tasks:

- Installation of the hardware for all nodes in the system, including any SCSA boards and the SCX160 SCxbus Adapter.
- Installation of the software at each node in the system, which may include the following activities:
    - Stopping the Dialogic Service to stop all the boards in an existing system, or installing the software for a new installation
    - Installing the SCxbus Adapter Package Software on each node in the system, in sequential order
- Configuration of each node in the system, in sequential order, which consists of the following activities:
    - Determining if the system will use a Global or a Local Map File. Global Map Files (scxmap.dat) are used in systems that are networked and contain information for all the nodes in the system. Local Map Files (scxmap.dat) are used in systems where each node operates independently and uses its own directory. The Local Map File contains the data stream map for all previously installed nodes.
    - Preparing a Local Map Transfer Disk, if using a Local Map File. The Local Map File from each node is copied onto this disk.

- Running the Dialogic Configuration Manager (DCM) which will subsequently run the Dialogic SCX160 Configuration Program. Choose **Yes** when prompted to configure the SCX board in order to run the *scxconfig* program.
- Determining if the multi-node system will use the *Blocking* or *Non-Blocking Configuration*. If the time slot requirements of your multi-node system exceeds the 1024 time slot limitation of the SCxbus, select **Blocking Configuration** when prompted by the Dialogic SCX160 Configuration Program (*scxconfig*). See *Appendix B - Dialogic SCX160 Configuration Program* for procedural information.
- Starting the Dialogic Service to start all the boards in the system.

The activities that comprise the node configuration process, including functions performed by the software that are transparent to the user, are described in more detail in the following sections.

3.2. Dialogic Configuration Manager

The open architecture of the SCbus provides the framework for configuring SCSA products or devices from different manufacturers so that all SCSA products installed in a system work together.

The Dialogic Configuration Manager (DCM) adds the SCX160 SCxbus Adapter board and any newly installed SCSA boards to the system configuration. The Dialogic Configuration Manager (DCM) detects the presence of an SCX160 board in the system and prompts the user to configure the SCX160 Board. If the node has not previously been configured, or if a change was made to the system configuration, select **Yes** to configure the SCX160 Board.

3.2.1. Dialogic SCX160 Configuration Program

The configuration of each node depends on the previously configured nodes (lower numbered) in the system. The Dialogic SCX160 Configuration program uses the requirements file and the information about the other nodes that is stored in the Map file and generated by the Master Assignment Program to determine the following:

- The number of time slots currently installed in the node
- The minimum number of data streams required
- The maximum number of data streams that can be allocated to the node
- The resources that can be blocked in the case of a Blocking Configuration
- The minimum number of data streams required to block a resource in the case of a Blocking Configuration

The Dialogic SCX160 Configuration program helps you to determine the number of data streams that need to be allocated to the node. More SCbus time slots than are required for the currently installed boards can be allocated to reserve the time slots for future use. If additional time slots are reserved, then additional boards may be added to these nodes without reconfiguring the system, until you run out of reserved SCbus time slots.

3.2.2. Dialogic Service Startup

The Dialogic Service needs to be started at each node after installing the software and then running the Dialogic Configuration Manager (DCM). Starting the Dialogic Service automatically initiates the following functions:

- Downloading of the existing boards in the node
- Generation of a requirements file
- Startup of all Dialogic boards in the system
- Running of the Master Assignment Program

The following sections describe the information received by the requirements file and how the Master Assignment Program uses this information to allocate time slots.

Requirements File

The following information is output to the requirements file:

- The number of transmit time slots required for the product type
- The name of the Time Slot Assignment Program that assigns SCbus time slots to that product type
- Any optional command line arguments

This information is used by the Master Assignment Program to calculate the number of time slots required by the node, as described in the following section.

Master Assignment Program

The Master Assignment Program reads all SCbus time slot requirements information stored in the Requirements file, adds up the time slot requirements, reads the starting (base) data stream from the Global or Local Map file, and then allocates SCbus transmit time slots in non-overlapping sets or bundles of 64. Each bundle of 64 time slots is handled as a data stream (DS0 through DS15) by the SCX160 SCxbus Adapter. The SCX160 SCxbus Adapter maps the SCbus transmit data streams onto corresponding numbered SCxbus data streams, and vice-versa, so that all SCbus transmit time slots are available to all PC hosts connected to the SCxbus. No data streams or time slots are switched by the SCX160 SCxbus Adapter.

The Master Assignment Program then calls each Time Slot Assignment Program and passes the following arguments:

- A base time slot number (the base SCbus transmit time slot number (default = 0) is read from the Map file)
- The number of time slots to be assigned
- Any optional command line arguments stored in the Requirements files

The Master Assignment Program then waits for notification that all Time Slot Assignment Programs executed and ran successfully.

The Time Slot Assignment Programs disconnect all SCbus time slots in a node and then connect each device transmit channel to an SCbus transmit time slot in accordance with the time slot information received from the program. A single Time Slot Assignment Program may assign SCbus transmit time slots to different types of SCbus products.

The assignment of SCbus (and concurrently SCxbus) transmit time slots uses a common mechanism to ensure that each device transmit channel in each node is assigned to a unique SCbus transmit time slot from 0 through 1023 for a total of 1024 transmit time slots per system. SCbus time slots are assigned at each Dialogic Service startup.

CAUTION

Do not manually execute the product group's Time Slot Assignment Programs. To do so could cause two transmit channels to be assigned to a time slot. Hardware can be damaged if two boards transmit on the same SCbus time slot.

4. Clocking

4.1. Overview of Clocking

To achieve communication between nodes, each SCX160 SCxbus Adapter interconnects, manages and controls the SCbus and the SCxbus interfaces. The software architecture for the SCX160 SCxbus Adapter incorporates a PLL (Phase Locked Loop) process in the SCX160 SCxbus Adapter's firmware that handles all clocking events and executes the SCbus and the SCxbus clock state changes.

This chapter provides the following information regarding clocking:

- Designating clock modes - information for determining which nodes will provide master and fallback master clock reference for all the other nodes
- Clock synchronization - a description of the various sources that can be used as clock references and of synchronous and plesiochronous clocked systems
- Required clock designations - a summary of clocking mode designations and selections, including examples of synchronous and plesiochronous multi-node systems
- Clock states - a discussion of clock states, clock alarms, and clock state transitions

4.2. Designating Clock Modes

Clocking designations are determined by evaluating total system resources and the system configuration, including SCSA boards other than the SCX160 SCxbus Adapter installed in each node. For each node, you must determine whether the clock for a particular board will be the master clock source or slaved to a clock reference provided on the SCbus or the SCxbus.

The SCX160 SCxbus Adapter clock mode(s) must be determined independently for both the SCbus and the SCxbus interfaces.

The clock designations for the SCbus are as follows:

- Master clock mode: typically, a network interface board provides System Wide Clock Master (SWCM) that becomes the clock reference for all SCSA boards and the SCX160 SCxbus Adapter in that node.
- Fallback master clock mode: takes over as the master clock if the board operating in master clock mode fails.
- Slave mode: SCbus boards operating in this mode take master clock from the SCbus as their clock reference.

The clock designations for the SCxbus are as follows:

- Master clock mode: the SCX160 SCxbus Adapter operating in master clock mode provides master clock via Right SCxbus Clock to all other SCX160 SCxbus Adapters in the system via the SCxbus.
- Fallback master clock mode: the SCX160 SCxbus Adapter operating in fallback master clock mode provides fallback master clock via Left SCxbus Clock to all other SCX160 SCxbus Adapters in the system via the SCxbus.
- Slave mode: SCX160 SCxbus Adapters operating in this mode take master clock from the SCxbus to synchronize its SCxbus interface circuits.

Unless otherwise configured by the downloadable parameters file at Dialogic Service startup, the SCX160 SCxbus Adapter clocking defaults to the following:

- For the SCxbus interface, Slave state
- For the SCbus interface, Slave state
- For the default clock when clock is not available from one of the above, the local crystal oscillator

To enable communication between nodes, one node must be designated as the SCxbus master clock node. Clocking designations can be accomplished at Dialogic Service startup by modifying the downloadable parameter file as described in *Appendix A - Downloadable Parameters* . Otherwise, all clock modes must be set by issuing an **scx_setbrdparm( )** function (see *Chapter 6. Function Reference*). The application can change the clocking configuration during system operation.

4.3. Clock Synchronization

The SCX160 SCxbus Adapters in the system maintain clock synchronization using one of the following as the reference clock source:

- Right SCxbus clock
- Left SCxbus clock
- 8 kHz external reference clock from the SCxbus
- SCbus clock
- Local Crystal Oscillator

The on-board Control Processor generates the control signals that select the appropriate reference clock source. An alarm event is generated if an error is detected in any clock (see *Section 4.5.1. Clock Alarms* for more information).

Typically, the SCbus clock is used as the reference clock source for SCX160 SCxbus Adapters in the nodes that have been designated as either the master clock or the fallback master clock. For all other nodes/SCX160 SCxbus Adapters, the SCxbus clock is used as the reference clock source.

For more on clock synchronization, see *Section 2.4.4. Synchronization Circuitry*.

4.3.1. Synchronous and Plesiochronous Operation

An SCX160 SCxbus Adapter multi-node system can operate as either a synchronous or plesiochronous clocked system. In a synchronous system, all nodes other than the Master Clock and Fallback Master Clock nodes operate in SCxbus slave mode. In a plesiochronous system all nodes other than the Master Clock and Fallback Master Clock nodes operate in SCbus slave mode AND in SCxbus slave mode simultaneously. For examples of each of these types of systems, see *Section 4.4.1. Clocking Examples*.

4.3.2. Clock Fallback

The application can change the SCX160 SCxbus Adapter clocking configuration during system operation, especially in the event of multiple failures. This level of clock fallback needs to be incorporated into a Clock Fallback List to be handled by the application or the system administrator.

If the SCbus Master Clock in the SCxbus Master Clock node fails, the next entry on this node's clock fallback list (typically another Network Interface Board) provides SCbus Master Clock. If the SCbus Master Clock falls back to the SCX160 SCxbus Adapter, the SCX160 SCxbus Adapter initiates system clock recovery by looking for the Left SCxbus clock. If detected, the Right SCxbus clock is suppressed to initiate fallback in the other system nodes.

The master clock node then tries to clock from the Left SCxbus clock. If unsuccessful (Left SCxbus clock not present), this node uses its local oscillator as the clock reference and provides both Right SCxbus clock and the SCbus Master Clock signal. If successful, this node provides the local SCbus Master Clock signal.

If an alternate SCxbus Master Clock Node is to be designated by the application or the system administrator, then un-designate the current SCxbus Master Clock Node and designate the new SCxbus Master Clock Node. This newly designated node starts driving the free Right SCxbus Clock and Frame Synchronization line after synchronizing to its designated reference. Once Right SCxbus Clock is detected by the other nodes, they will automatically synchronize to the Right SCxbus Clock, thus completing overall system clock recovery.

4.4. Required Clock Designations

The following clock designations are required in an SCxbus multi-node system.

For the SCxbus Master Clock Node, make the following selections:

- Select the Network Interface Board that will provide SCbus Master Clock
- Select any remaining Network Interface Board(s) and/or the SCX160 SCxbus Adapter to be used as an alternate fallback clock for the SCbus Master Clock source and place these boards into a preferred fallback order. The SCX160 SCxbus Adapter should be last.
- For the SCX160 SCxbus Adapter:
    - Designate slave mode for the SCbus interface
    - Designate Master Clock for the SCxbus interface
    - Designate a reference clock source for the SCX160 SCxbus Adapter, typically, the SCbus

For the SCxbus Fallback Master Clock Node, make the following selections:

- If a Network Interface Board is available in this node to provide SCbus Master Clock, then designate this board as the source of SCbus Master Clock
- If additional Network Interface Boards are available in this node that can provide fallback clock for the SCbus Master Clock source, then designate these boards as a fallback SCbus Master Clock source and place these boards into a preferred fallback order
- For the SCX160 SCxbus Adapter:
    - If the source of SCbus Master Clock is a Network Interface Board as described above, then designate slave mode for the SCbus interface
    - If the source of SCbus Master Clock is the SCX160 SCxbus Adapter (no Network Interface Board in this node is designated to provide SCbus Master Clock), then designate Master Clock Mode for the SCbus interface
    - Designate Fallback Master Clock for the SCxbus interface
    - Designate a reference clock source for the SCX160 SCxbus Adapter, typically, the SCbus

For all other SCxbus Nodes/SCX160 SCxbus Adapters, make the following selections:

- For synchronous clocking:
  - Designate Synchronous Slave mode for the SCxbus interface
  - Designate Master Clock mode for the SCbus interface
  - Designate a reference clock source for the SCX160 SCxbus Adapter, typically, the SCxbus
- For plesiochronous clocking:
  - Designate Plesiochronous Slave mode for the SCxbus interface
  - Designate slave mode for the SCbus interface

4.4.1. Clocking Examples

The following examples illustrate the interactions between the designated Master Clock node, the Fallback Master Clock node and the slave nodes in both a synchronous and a plesiochronous clocked system.

Synchronous Clocking Example

Figure 6. Synchronous Clocking Example provides an illustration of synchronous clocking for a multi-node system.

While referring to the figure, assume the following:

- Node 1 is designated as the source of SCxbus Master Clock
- Node 2 is a fallback node designated as the source of SCxbus fallback Master Clock
- Nodes 3 through 16 are designated as Synchronous Slave Mode nodes

The remainder of this section describes the operation of the nodes in the example.

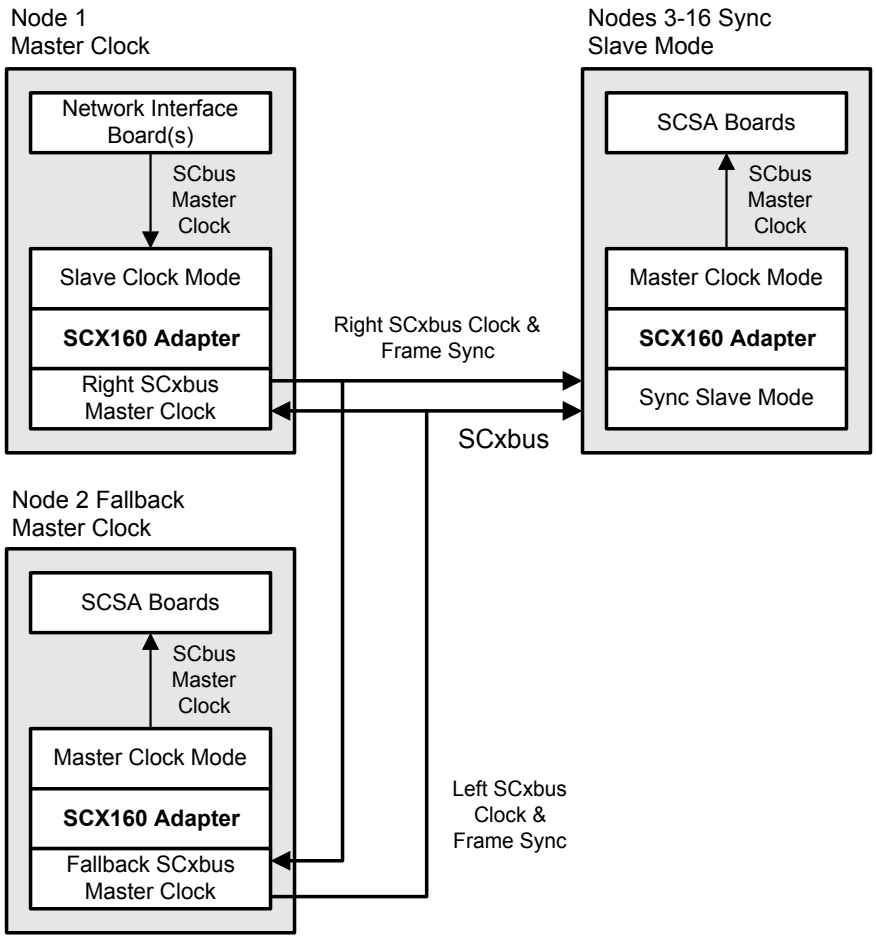

Figure 6. Synchronous Clocking Example

Node 1: SCxbus Master Clock

Node 1 includes one or more Network Interface Boards that provide a System Wide Clock Master (SWCM) reference for the entire multi-node system. This SWCM drives the SCbus Master Clock signal which is applied to the SCX160

SCxbus Adapter SCbus interface. In this configuration, the SCX160 SCxbus Adapter SCbus interface operates in Slave Clock Mode.

The SCX160 SCxbus Adapter uses the SCbus master clock as the clock source for the SCxbus and thus for the entire multi-node system by operating its SCxbus interface in SCxbus Master Clock Mode.

For this example, the SCX160 SCxbus Adapter synchronizes the multi-node system with the Right SCxbus Clock and Frame Synchronization signal. This clock reference is sent to all other SCX160 SCxbus Adapters via the SCxbus.

Node 2: SCxbus Fallback Master Clock

In this example, Node 2 is designated as the fallback node. In this configuration, the SCX160 SCxbus Adapter SCxbus interface operates in SCxbus Fallback Master Clock mode.

The SCX160 SCxbus Adapter uses the Right SCxbus Clock and Frame Synchronization signal from Node 1 to phase synchronize the Left SCxbus Clock and Frame Synchronization signal. This signal is then sent to all other SCX160 SCxbus Adapters via the SCxbus to be used if the Master Clock from Node 1 fails.

The SCX160 SCxbus Adapter SCbus interface can operate in either of the following modes:

- Master Clock Mode (synchronous operation)
- Slave Clock Mode (plesiochronous operation)

To have a fully synchronized system where all nodes and SCSA boards clock from the master clock reference in the master clock node (node 1 in this example), the SCbus interface of the node 2 SCX160 SCxbus Adapter would be set to Master Clock Mode. In this mode, the SCX160 SCxbus Adapter provides the SCbus master clock to drive all boards connected to the SCbus.

For the fallback master clock node (node 2 in this example), plesiochronous operation may be more advantageous. If the Right SCxbus Clock is lost, then all slaved nodes automatically switch to the Left SCxbus Clock for their clock

reference. The fallback master clock node (node 2) will switch to an application designated reference clock selected from one of the following, if available:

- SCbus clock
- Local Crystal Oscillato
- 8 kHz external reference clock from the SCxbus

If the designated reference clock is not available, then the fallback node automatically switches to its Local Crystal Oscillator.

Typically, the most stable clock reference would be clock from a locally installed network board. If a network board is providing SCbus master clock, then the SCX160 SCxbus Adapter can fallback to the SCbus master clock as its reference clock and continue to provide a highly stable clock to the SCxbus network. In this configuration, the SCbus interface of the SCX160 SCxbus Adapter operates in slave clock mode so that the SCbus master clock is always available to the SCX160 SCxbus Adapter.

Nodes 3 to 16: Synchronous Slave Mode

In this example, these nodes use the master clock reference from the SCxbus. In this configuration, the SCX160 SCxbus Adapter SCxbus interface operates in Synchronous Slave Mode.

The SCX160 SCxbus Adapter uses Right SCxbus Clock and Frame Synchronization signal from Node 1 as its master clock source. If the Master Clock in Node 1 fails, the SCX160 SCxbus Adapter automatically falls back to the Left SCxbus Clock and Frame Synchronization signal from Node 2.

The SCX160 SCxbus Adapter SCbus interface in these nodes is set to operate in Master Clock Mode so that all SCSA boards installed in these nodes are clocked from the SCX160 SCxbus Adapter.

Plesiochronous Clocking Example

Figure 7. Plesiochronous Clocking Example provides an illustration of plesiochronous clocking in a multi-node system.

While referring to the figure, assume the following:

- Node 1 is designated as the source of SCxbus Master Clock
- Node 2 is a fallback node designated as the source of SCxbus fallback Master Clock
- Nodes 3 through 16 are designated as Plesiochronous Slave Mode nodes

The remainder of this section describes the operation of the nodes in the example.

Node 1: SCxbus Master Clock

Node 1 includes one or more Network Interface Boards that provide a SWCM reference for the entire multi-node system. This SWCM drives the SCbus Master Clock signal which is applied to the SCX160 SCxbus Adapter SCbus interface. In this configuration, the SCX160 SCxbus Adapter SCbus interface operates in Slave Clock Mode.

The SCX160 SCxbus Adapter uses the SCbus master clock as the clock source for the SCxbus and thus for all SCX160 SCxbus Adapters in the multi-node system by operating its SCxbus interface in SCxbus Master Clock Mode.

For this example, the SCX160 SCxbus Adapter synchronizes all SCX160 SCxbus Adapters in the multi-node system with the Right SCxbus Clock and Frame Synchronization signal via the SCxbus.

Node 2: SCxbus Fallback Master Clock

In this example, Node 2 is designated as the fallback node. In this configuration, the SCX160 SCxbus Adapter SCxbus interface operates in SCxbus Fallback Master Clock Mode.

The SCX160 SCxbus Adapter uses the Right SCxbus Clock and Frame Synchronization signal from Node 1 to phase synchronize the Left SCxbus Clock and Frame Synchronization signal. This signal is then sent to all other SCX160 SCxbus Adapters via the SCxbus to be used if the Master Clock from Node 1 fails.

The SCX160 SCxbus Adapter SCbus interface is set to operate in Slave Clock Mode (plesiochronous operation).

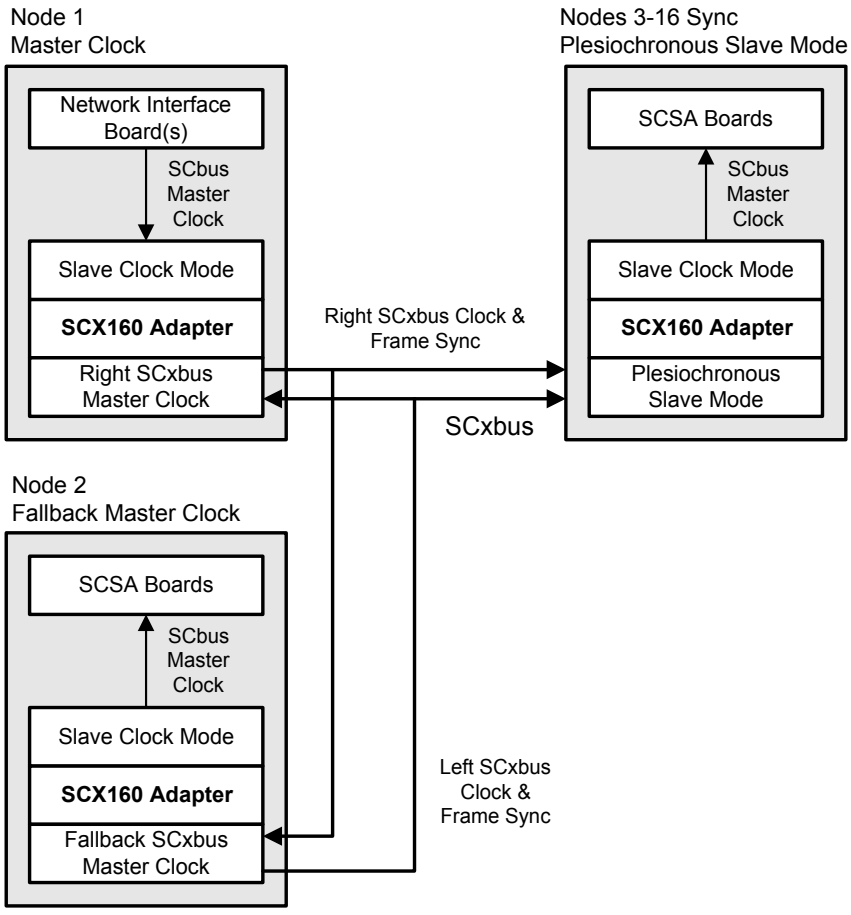

Figure 7. Plesiochronous Clocking Example

If the Right SCxbus Clock is lost, the fallback master clock node (node 2) will switch to an application designated reference clock selected from one of the following, if available:

- SCbus clock
- Local Crystal Oscillator
- 8 kHz external reference clock from the SCxbus

If the designated reference clock is not available, then the fallback node automatically switches to its Local Crystal Oscillator.

Typically, the most stable clock reference would be clock from a locally installed network board. If a network board is providing SCbus master clock, then the SCX160 SCxbus Adapter can fallback to the SCbus master clock as its reference clock and continue to provide a highly stable master clock to the SCxbus network. In this configuration, the SCbus interface of the SCX160 SCxbus Adapter operates in slave clock mode so that the SCbus master clock is always available to the SCX160 SCxbus Adapter.

Nodes 3 to 16: Plesiochronous Slave Mode

In this example, the SCX160 SCxbus Adapter in each node uses the master clock reference from the SCxbus to synchronize SCxbus activities. In this configuration, the SCX160 SCxbus Adapter SCxbus interface operates in Plesiochronous Slave Mode.

The SCX160 SCxbus Adapter uses Right SCxbus Clock and Frame Synchronization signal from Node 1 as its master clock source. If the Master Clock in Node 1 fails, the SCX160 SCxbus Adapter automatically falls back to the Left SCxbus Clock and Frame Synchronization signal from Node 2.

The SCX160 SCxbus Adapter SCbus interface in these nodes is set to operate in Slave Clock Mode so that the SCX160 SCxbus Adapter SCbus interface and all SCSA boards installed in the node are clocked from the SCbus.

4.5. Clock States

The clock state of each node is initially set to Slave state for the SCxbus. This clock state can be changed in any of the following ways:

- At Dialogic Service startup by adding the SCX160 SCxbus Adapter to the board configuration file
- By the application at any time during system operations by issuing a **scx_setbrdparm( )** function
- By an alarm event indicating a clock failure

Clock alarms, clock state machines and clock state transitions are discussed in the following sections.

4.5.1. Clock Alarms

The PLL (Phase Locked Loop) process gathers all PLL statistics and forwards all PLL alarm conditions to an Alarm process. This Alarm process passes the alarm messages to the host computer in accordance with the alarm masks that have been set. Alarm masks enable or disable the internal alarms. Each internal alarm has its own mask and is mapped to a host alarm event and a host alarm state byte. The default is to disable all alarms.

When a clock failure is detected the following occur:

- A hardware interrupt (maskable) is generated
- A TSCX_ALARM alarm event is generated
- A clock failure timer is started

Every 40 msec, this timer initiates a poll of the clock status. If the failed clock has recovered, then a clock recovered event is posted.

Alarm information is stored in the SCXBP_CLK_HISTORY clock history bitmask parameter (see the **scx_getbrdparm( )** function in *Chapter 6. Function Reference* for details). The SCXBP_CLK_HISTORY bitmask defines the specific clock event that occurred. This information is retrieved using the **scx_getbrdparm( )** function.

See *Table 7. Clock Alarm Events* for a list of SCX160 SCxbus Adapter clock alarm events and their corresponding parameter values. A clock failure event may cause a transition to a different clock state.

Table 7. Clock Alarm Events

Clock Failure Event	SCXBP_CLK_HISTORY Value
Right SCxbus clock	SCXST_SCX_RCLOCK
Left SCxbus clock	SCXST_SCX_LCLOCK
SCbus clock	SCXST_SC_CLOCK
Clockfail signal from SCbus	SCXST_SC_CLOCK_FAIL_SIG
External reference clock from SCxbus	SCXST_EXTCLOCK
PLL out-of-lock	SCXST_PLL_REF

4.5.2. SCxbus Clock State Machine

The SCxbus clock state machine controls the clocking on the SCxbus interface of the SCX160 SCxbus Adapter and also determines the clock source used for the PLL clock. Its objective is to ensure that the SCxbus always has a clock and to ensure that a clock signal is never erroneously placed on a clocked SCxbus line.

If the clock reference for the master SCxbus clock node fails, the Master clock state at this node automatically changes to the Wait Slave state (see *Section 4.5.3. SCxbus Clock State Transitions*) and all nodes automatically start clocking from the SCxbus left clock provided by the Fallback Master node.

NOTE: The return to the master clocking state at the original master node is either automatic due to recovery of the clock reference or the application must issue an **scx_setbrdparm( )** call to designate a new master clock reference.

The SCxbus clock state machine changes from an existing state to a new state in response to a command from the application or an event such as a clock failure. See *Table 8. SCxbus Clock State Transition Events* for a summary of the

commands and events that can cause a clock state transition in the SCxbus state machine. To retrieve the values listed, use the **scx_getbrdparm( )** function.

Table 8. SCxbus Clock State Transition Events

Parameter Value	Description
SCXCM_MASTER	SCXBP_SCX_CLOCK_MODE parameter designates node to be SCxbus master - i.e., to provide right SCxbus clock
SCXCM_SLAVE	SCXBP_SCX_CLOCK_MODE parameter designates node as slave - i.e., to receive SCxbus clock from another SCxbus device
SCXCM_FALLBACK_ MASTER	SCXBP_SCX_CLOCK_MODE parameter designates node to be SCxbus fallback master - i.e., to receive right SCxbus clock from another SCxbus device and to provide left SCxbus clock to be used by other SCxbus devices if the right SCxbus clock is unavailable
SCXST_SCX_RCLOCK	SCxbus right clock failed, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_SCX_LCLOCK	SCxbus left clock failed, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_SC_CLOCK	SCbus clock failed, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_SC_CLOCK_FAIL _SIG	SCbus clockfail signal detected, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_EXTCLOCK	SCxbus external clock failed, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_PLL_REF	PLL out-of-lock, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_CLK_RECOVER	Indicates failed clock recovered, use SCXBP_CLK_HISTORY parameter to retrieve

When the PLL process selects a clock source for a node, the following priorities apply in the order listed:

- For the node operating in the SCxbus Master state:
  - The clock source defined in the database (typically, the SCbus, the 8 kHz external reference clock from the SCxbus, or the local crystal oscillator)
  - The local crystal oscillator
- For the node operating in the SCxbus Fallback Master state:
  - SCxbus right clock
  - The clock source defined in the database
  - The local crystal oscillator
- For nodes operating as SCxbus slaves:
  - SCxbus right clock
  - SCxbus left clock
  - SCbus clock
  - Local crystal oscillator

4.5.3. SCxbus Clock State Transitions

The SCxbus clock state transitions for each node are summarized in *Figure 8. SCxbus Clock States Transition Diagram*. *Table 9* defines the clock states in the diagram.

By starting with a current clock state, the diagram in *Figure 8* illustrates the events and/or commands that can cause a transition from one state to another. To illustrate clock state transitions, the following description focuses on the clock transitions initiated by events or by application command for the following:

- Master state clock node
- Wait Master state clock node
- Fallback Master state clock node
- Wait Fallback Master state clock node

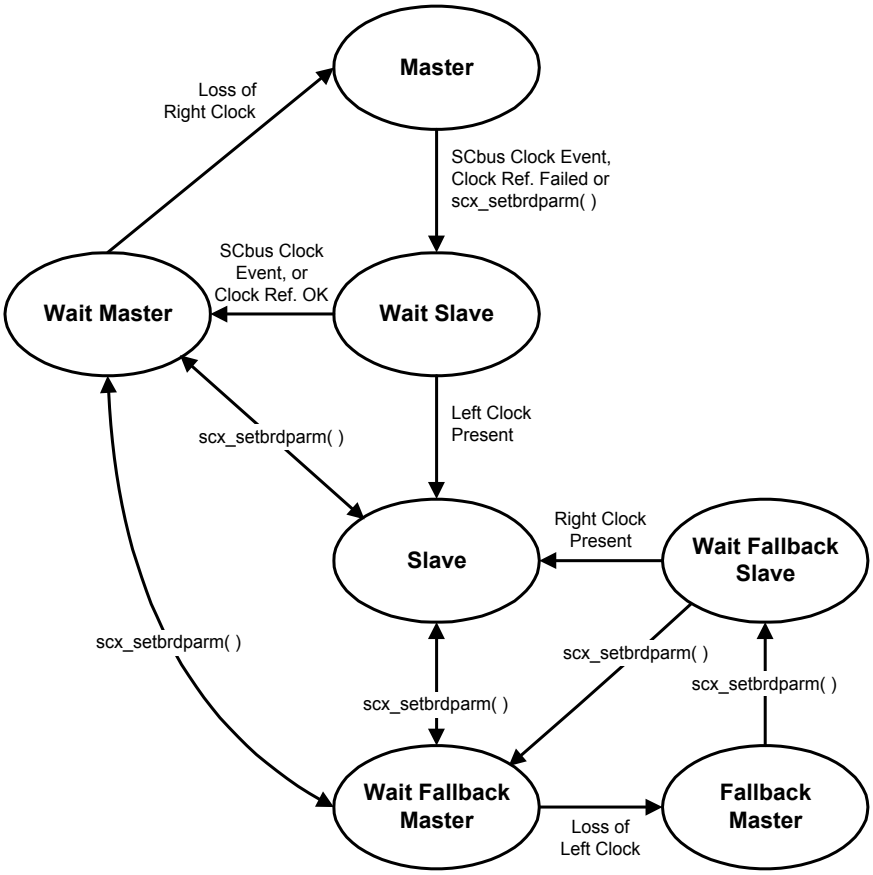

Figure 8. SCxbus Clock States Transition Diagram

Table 9. SCxbus Clock States

Clock State	Description
Master	The SCX160 SCxbus right clock is enabled, the left clock is disabled and the PLL clock is locked to the database or default value. If this master clock source fails or if the clocking configuration is changed by the **scx_setbrdparm( )** function or if a SCbus clock event prevents the SCX160 SCxbus Adapter from being a viable SCxbus Master, then this node automatically switches to Wait Slave state.
Wait Master	The SCX160 SCxbus right and left clocks are disabled. If the SCxbus right clock is not present, this node automatically switches to Master state and provides SCxbus right clock to all other nodes.
Wait Slave	The SCX160 SCxbus right clock is enabled, the left clock is disabled and the PLL clock is locked to the database value or to the local crystal oscillator. In this state, the node is waiting for the SCxbus left clock to recover, at which point this node will automatically switch to Slave state.
Fallback Master	The SCX160 SCxbus right clock is disabled, the left clock is enabled and the PLL clock is locked to either the SCxbus right clock, the database value or the local crystal oscillator.
Wait Fallback Master	The SCX160 SCxbus right and left clocks are disabled. The PLL clock is locked to either the SCxbus right clock, the database value or the local crystal oscillator. If the SCxbus left clock is not present, this node automatically changes to Fallback Master state and starts providing SCxbus left clock.
Wait Fallback Slave	This SCX160 SCxbus right clock is disabled, the left clock is enabled and the PLL clock is locked to the database value or to the local crystal oscillatorIn this state, the node is waiting for the SCxbus right clock to recover, at which point this node will automatically switch to Slave state and disable the left SCxbus clock.
Slave	The SCX160 SCxbus right and left clocks are disabled. The PLL clock is locked to the highest priority good clock.

SCxbus Master State Clock Node Transitions

The node designated as the source of master clock to the SCxbus has its SCxbus right clock enabled and its left clock disabled. This node continues to provide SCxbus right clock until one of the following occurs:

- Its reference clock fails
- An SCbus state machine event occurs that indicates that this node can no longer provide reliable clock
- An application command is issued (using the **scx_setbrdparm( )** function)

When any of the above events occur, this node transitions to the Wait Slave state as shown in *Figure 8*. The Wait Slave state is a transitory state during which the master clock node continues to provide SCxbus right clock until the node verifies the presence of the SCxbus left clock. If the SCxbus left clock is present, then this node transitions to the Slave state, as shown in *Figure 8*.

Alternatively, if during this period, the node's reference clock recovers or a SCbus state machine event indicates that this node is capable of providing master clock, then the node transitions to the Wait Master state, as shown in *Figure 8*.

In the Wait Master state, the node continually checks for the presence of the SCxbus right clock. When this clock is not detected, then this node automatically transitions to the Master state and provides master clocking on the SCxbus right clock line. This node will not take over as the master clock source as long as SCxbus right clock is being provided by another node.

SCxbus Wait Master State Clock Node Transitions

A node operating in the Slave state or the Wait Fallback Master state can be commanded by the **scx_setbrdparm( )** function to transition to the Wait Master state, as shown in *Figure 8*.

In the Wait Master state, the node continually checks for the presence of the SCxbus right clock. When this clock is not detected, then this node automatically transitions to the Master state and provides master clocking on the SCxbus right clock line. This node will not take over as the master clock source as long as SCxbus right clock is being provided by another node.

SCxbus Fallback Master State Clock Node Transitions

The node designated as the source of fallback master clock to the SCxbus has its SCxbus right clock disabled and its left clock enabled. This node continues to provide SCxbus left clock until commanded to change to the Wait Fallback Slave state by the application.

The Wait Fallback Slave state is a transitory state during which the fallback master node continues to provide SCxbus left clock until the node verifies the presence of SCxbus right clock. If the SCxbus right clock is present, then this node transitions to the Slave state, as shown in *Figure 8*.

Alternatively, this node can be commanded to transition from the Wait Fallback Slave state to the Wait Fallback Master state. In this state, the node is slaved to the SCxbus clock; the SCxbus right and left clocks are disabled. In the Wait Fallback Master state, the node continually checks for the presence of the SCxbus left clock. When this clock is not detected, then this node automatically transitions to the Fallback Master state and provides fallback master clocking on the SCxbus left clock line. This node will not take over as the fallback master clock source as long as SCxbus left clock is being provided by another node.

SCxbus Wait Fallback Master State Clock Node Transitions

A node operating in the Slave state, the Wait Master state or the Wait Fallback Slave state can be commanded by the s**cx_setbrdparm( )** function to transition to the Wait Fallback Master state, as shown in *Figure 8*.

In the Wait Fallback Master state, the node continually checks for the presence of the SCxbus left clock. When this clock is not detected, then this node automatically transitions to the Fallback Master state and provides fallback master clocking on the SCxbus left clock line. This node will not take over as the fallback master clock source as long as SCxbus left clock is being provided by another node.

4.5.4. SCbus Clock State Machine

The SCbus clock state machine controls the clocking on the SCbus interface of the SCX160 SCxbus Adapter. Its objective is to ensure that the SCbus always has a clock and to ensure that the SCX160 SCxbus Adapter never places a clock signal

on a clocked SCbus. The SCbus clock state machine can automatically provide SCbus clock upon failure of the SCbus master clock.

The SCbus clock state machine changes from an existing state to a new state in response to a command from the application or an event such as a clock failure. See *Table 10. SCbus Clock State Transition Events* for a summary of the commands and events that can cause a clock state transition in the SCbus state machine. To retrieve the values listed, use the **scx_getbrdparm( )** function.

Table 10. SCbus Clock State Transition Events

Parameter Value	Description
SCXCM_MASTER	SCXBP_SC_CLOCK_MODE parameter designates SCX160 to provide SCbus clock to the SCbus interface
SCXCM_SLAVE	SCXBP_SC_CLOCK_MODE parameter designates SCbus interface of SCX160 to receive SCbus clock, i.e., to become SCbus slave
SCXCM_FALLBACK_ MASTER	SCXBP_SC_CLOCK_MODE parameter designates SCX160 to receive SCbus clock from another SCbus device and to provide fallback SCbus clock to be used by other SCbus devices if the master SCbus clock fails
SCXST_SC_CLOCK	SCbus clock faile, use SCXBP_CLK_HISTORY parameter to retrieve
SCXST_SC_CLOCK_FAIL _SIG	SCbus clockfail signal was asserted, use SCXBP_CLK_HISTORYparameter to retrieve
SCXST_CLK_RECOVER	Indicates failed clockrecovered, use SCXBP_CLK_HISTORY parameter to retrieve

4.5.5. SCbus Clock State Transitions

The SCbus clock state transitions for each node are summarized in *Figure 9. SCbus Clock States Transition Diagram*. *Table 11* defines the clock states in the diagram.

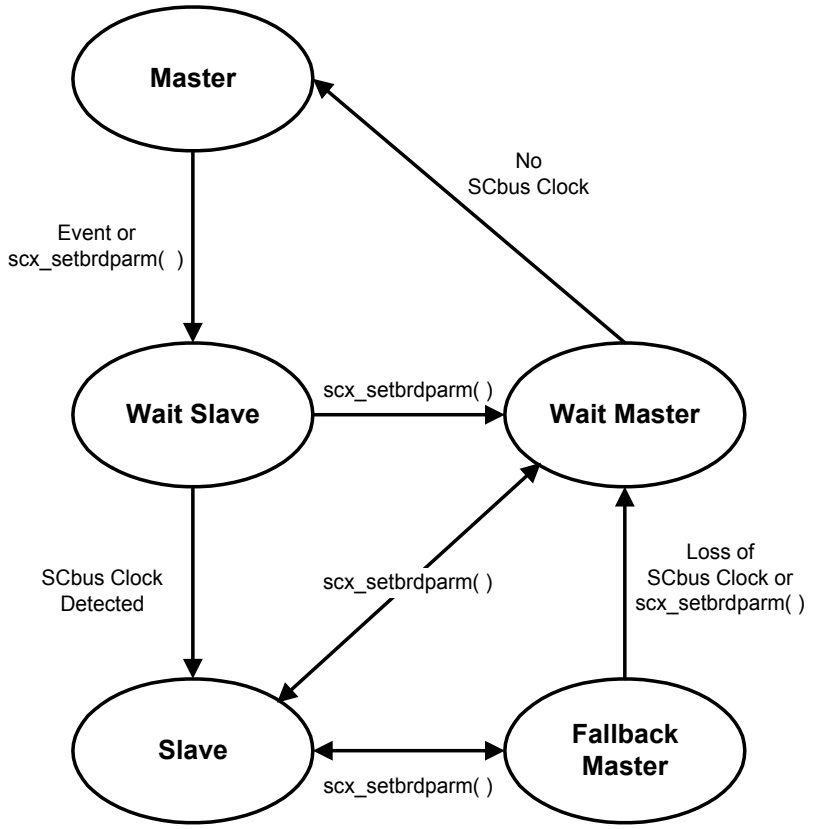

Figure 9. SCbus Clock States Transition Diagram

Table 11. SCbus Clock States

Clock State	Description
Master	The SCX160 SCxbus Adapter's SCbus interface provides clock to the SCbus.
Fallback Master	The SCX160 SCxbus Adapter's SCbus clock driver is disabled (interface operates as a slave) until the master SCbus clock fails, at which point the SCX160 SCxbus Adapter automatically provides SCbus clock.
Wait Master	The SCX160 SCxbus Adapter's SCbus clock driver is disabled until the SCbus clock is no longer present, at which point the SCX160 SCxbus Adapter automatically provides SCbus clock.
Wait Slave	The SCX160 SCxbus Adapter's SCbus interface provides SCbus clock to the SCbus until another SCbus device uses the clockfail signal to indicate that it is ready to take over the task of providing SCbus clock.
Slave	The SCX160 SCxbus Adapter's SCbus interface is slaved to the SCbus clock. This state is changed by issuing a **scx_setbrdparm( )** function that changes the SCXBP_SC_CLOCK_MODE parameter.

By starting with a current clock state, the diagram in *Figure 9* illustrates the events and/or commands that can cause a transition from one state to another. To illustrate clock state transitions, the following description focuses on the clock transitions initiated by events or by application command for the following nodes:

- Master state clock node
- Wait Master state clock node
- Fallback Master state clock node

Since the SCxbus and the SCbus interfaces are relatively independently clocked, a node can operate with its SCbus interface in the Master State, the Fallback Master state or in the Slave state.

SCbus Master State Clock Node Transitions

In the SCbus Master state, the SCX160 SCxbus Adapter SCbus interface provides clock for all devices connected to the SCbus. The SCX160 SCxbus Adapter continues to provide SCbus clock until an SCxbus state machine event indicates that the SCX160 SCxbus Adapter can no longer provide reliable clock or by issuing an application command (using the **scx_setbrdparm( )** function).

When either of the above events occurs, the SCbus interface transitions to the Wait Slave state (see *Figure 9*). The Wait Slave state is a transitory state during which the SCX160 SCxbus Adapter continues to provide clock to the SCbus until another SCbus device takes over this task. When the SCbus clockfail signal indicates that another SCbus device is ready to provide SCbus Clock, the SCbus interface transitions to the Slave state.

Alternatively, by application command, the SCbus interface can transition to the Wait Master state. In this state, the SCX160 SCxbus Adapter continually monitors the SCbus clockfail signal for loss of SCbus clock. When the SCbus clock is not present, the SCbus interface automatically transitions to the Master state and provides SCbus clock.

SCbus Wait Master State Clock Node Transitions

A SCbus interface operating in the Slave state, the Wait Slave State or the Fallback Master state can be commanded by the **scx_setbrdparm( )** function to transition to the Wait Master state, as shown in *Figure 9*.

In the Wait Master state, the SCbus interface continually monitors the SCbus clockfail signal for the absence of SCbus clock. When this clock is not present, then the SCbus interface automatically transitions to the Master state and provides SCbus clocking. The SCbus interface will not take over as the master clock source as long as SCbus clock is being provided by another device connected to the SCbus.

SCbus Fallback Master State Clock Node Transitions

The SCX160 SCxbus Adapter SCbus interface can be designated as fallback master source for SCbus clock. When the SCbus interface is operating in the Fallback Master state, the SCX160 SCxbus Adapter continually monitors the SCbus clockfail signal for the presence of SCbus clock. When SCbus clock is not present, then the SCbus interface automatically transitions to the Wait Master state, as shown in *Figure 9*.

In the Wait Master state, the SCX160 SCxbus Adapter checks for the absence of clock on the SCbus. If no clock signal is detected, then the SCbus interface transitions to the Master state.

5. Library Function Overview

5.1. SCX160 SCxbus Adapter library functions

The Dialogic SCX160 SCxbus Adapter for Windows library functions provide the building blocks for creating SCX160 SCxbus Adapter applications for multi-node systems. An overview of these functions, grouped into the following categories, is presented in this chapter:

- Configuration functions - sets device parameters
- Diagnostic functions - tests the communications interface of the device
- Attribute functions - gets device specific information

Detailed function descriptions are provided in *Chapter 6. Function Reference*.

5.2. Configuration Functions

Configuration functions retrieve, change and set SCX160 SCxbus Adapter device parameters and open or close an SCX160 board.

scx_close( )	• closes a previously opened SCX160 Adapter device
scx_clrevtmsk( )	• stops the generation of alarm events
scx_getbrdparm( )	• retrieves board parameters
scx_getevtmsk( )	• retrieves alarm event mask
scx_open( )	• opens an SCX160 board device
scx_setbrdparm( )	• changes SCX160 board level parameters
scx_setevtmsk( )	• enables alarm events specified in the event mask

5.3. Diagnostic Functions

Diagnostic functions check the communications interface of the SCX160 SCxbus Adapter with its host computer.

scx_tstcom( )	• runs a test that verifies communications between the host computer and the SCX160 SCxbus Adapter
scx_tstdat( )	• runs a data test to ensure that data is successfully passed from the host computer to the SCX160 SCxbus Adapter

5.4. Attribute Functions

Attribute functions retrieve specific firmware and hardware information about the SCX160 SCxbus Adapter.

scx_getctinfo( )	• gets information about SCX160 SCxbus Adapter
scx_gtbdattr( )	• gets SCX160 board attributes

6. Function Reference

6.1. Include Files

A list of the include files and a detailed description of each SCX160 SCxbus Adapter function, presented in alphabetical order, are contained in this chapter.

The following lines must be included in application code prior to calling library functions:

```
#include   <windows.h>
#include   <srllib.h>
#include   <dxxxlib.h>
#include   <dtilib.h>
#include   <scxlib.h>
```

NOTE: These include statements should be listed in the exact order shown.

Function prototypes for the SCX160 SCxbus Adapter and equates are defined in the *scxlib.h* file.

The library files used to link an application using the SCX160 SCxbus Adapter are as follows:

```
libsrlmt.lib
libdxxmt.lib
libdtimt.lib
```

The *libsrlmt.lib* and *libdxxmt.lib* library files are installed with System Release 4.25 SC or Dialogic System Software and SDK for Windows 96.11 or 97.01; *libdtimt.lib* is replaced by the SCX160 SCxbus Adapter Package Release. All library files are located in the *%DIALOGICDIR%\dialogic\lib* directory.

6.2. Error Codes

The library functions return one of the following values to indicate the success or failure of a function call:

- 0 indicates successful function completion
- -1 indicates function failure

NOTE: The **scx_open( )** function returns the device handle instead of 0 on success and, like the other functions, returns -1 on failure.

6.3. Alphabetical Function Descriptions

The Dialogic SCX160 SCxbus Adapter library functions for the SCX160 SCxbus Adapter are listed alphabetically in the remainder of this chapter and provide the following information:

Function Header	Lists the function name, function syntax, input parameters, outputs or returns, include files, category classification, and mode (synchronous). The function syntax and inputs are shown using standard C language syntax.
Description	Provides a detailed description of the function operation, including parameter descriptions.
Cautions	Provides warnings and reminders.
Example	Provides one or more C language coding examples showing how the function can be used.
Errors	Lists functions used to obtain error codes and error messages.
See Also	Provides a list of related functions.

Name: int scx_close(devh)
Inputs: int devh • SCX160 SCxbus Adapter device handle
Returns: 0 on success
-1 on failure
Includes: scxlib.h
Category: Configuration
Mode: synchronous

■ Description

The **scx_close( )** function closes a previously opened SCX160 device. This function releases the device handle and breaks the link between the calling process and the device.

Parameter	Description
devh:	SCX160 SCxbus Adapter device handle

■ Cautions

This function will fail if an invalid device handle is specified. Never close a Dialogic device using the Windows **close( )** function.

Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int  devh;                          /* SCX160 device handle */
unsigned short event_mask;          /* Event Mask           */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n ", errno);
     exit(1);
}

/*
 * Disable the reporting of all clocking alarms on the SCbus
 * and on the SCxbus.
 */
event_mask = (SCXEV_SCXBUS_CLOCK_FAIL    | SCXEV_SCXBUS_CLOCK_FAIL |
              SCXEV_EXT_CLOCK_FAIL);
if (scx_clrevtmsk(devh, event_mask) == -1) {
     printf("ERROR: scx_clrevtmsk( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}

/* Close the SCX160 Board */
if (scx_close(devh) == -1)   {
   printf ("Cannot close SCX160 Board Device.  errno=%d \n", errno);
   exit(1);
}
```

Errors

If the function returns -1, the specific error that occurred is returned in the errno global variable.

See Also

- **scx_open( )**

Name:	int scx_clrevtmsk(devh, mask)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	unsigned short mask	• Event mask for alarms
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **scx_clrevtmsk( )** function stops the generation of the alarm events specified in the mask parameter. Only alarm events are supported by this function.

Parameter	Description
devh:	SCX160 SCxbus Adapter device handle
mask:	Indicates one or more of the following event mask equates: • SCXMM_EXT_CLOCK - external clock event mask • SCXMM_FIFO - SCX160 FIFO event mask • SCXMM_SC_CLOCK - SCbus clock event mask • SCXMM_SC_DRIVER - SCbus driver event mask • SCXMM_SCX_CLOCK - SCxbus clock event mask • SCXMM_SCX_DRIVER - SCxbus driver event mask

■ Cautions

- This function disables alarm notification only for the event(s) specified. If notification for other events was previously enabled or disabled, their status of notification does not change.
- This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int   devh;                         /* SCX160 device handle */
unsigned short event_mask;          /* Event Mask           */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n ", errno);
     exit(1);
}

/*
 * Disable the reporting of all clocking alarms on the SCbus
 * and on the SCxbus.
 */
event_mask = (SCXMM_SCX_CLOCK | SCXMM_SC_CLOCK);
if (scx_clrevtmsk(devh, event_mask) == -1) {
     printf("ERROR: scx_clrevtmsk( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

■ See Also

- **scx_getevtmsk( )**
- **scx_setevtmsk( )**

Name:	int scx_getbrdparm(devh, parm, valuep)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	unsigned char parm	• parameter defined name
	void *valuep	• pointer where value will be returned
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **scx_getbrdparm( )** function retrieves board parameters. Parameters may be informative, defaults or values previously set by the scx_setbrdparm( ) function. Each parameter has a symbolic name defined in the scxlib.h file.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
parm:	Specifies the name of the parameter. See *Table 12. SCX160 SCxbus Adapter Parameters* below for a complete list of parameters. The user selectable parameters, designated by a "W" (write) in the "R/W" (read/write) column, can be set using the **scx_setbrdparm( )** function.
valuep:	Points to the buffer where the parameter value will be returned.

■ Cautions

Ensure that the valuep buffer is large enough to receive the parameter value, see *Table 12. SCX160 SCxbus Adapter Parameters* below. This function will fail if an invalid device handle is specified.

Table 12. SCX160 SCxbus Adapter Parameters

Define	B	Default	R/W	Description
SCXBP_SCX_ CLOCK_MODE	1	Default SCxbus clocking configured at download time.	R/W	Specifies clocking for the SCxbus interface of the SCX160 SCxbus Adapter. The SCxbus interface can be set to any of the following: • SCXCM_MASTER - provides right SCxbus clock • SCXCM_SLAVE - receives clock from another SCxbus device • SCXCM_FALLBACK_ MASTER - receives right clock from another SCxbus device, and generates left SCxbus clock to be used by other SCxbus devices if the right SCxbus clock is unavailable
SCXBP_CLOCK_ REF	1	Default SCX160 Clock Reference Signal configured at download time.	R/W	Specifies source of the SCX160 SCxbus Adapter clock reference signal. The clock source can be any one of the following: • SCXCS_INTERNAL - use Internal Clock • SCXCS_SCBUS - derive clock from SCbus • SCXCS_SCXBUS_RIGHT - derive clock from SCxbus Right Clock • SCXCS_SCXBUS_LEFT - derive clock from SCxbus Left Clock

Define	B	Default	R/W	Description
				• SCXCS_EXTERNAL - use External 8 kHz Clock
SCXBP_FIFO_ ERR_THRESH	4	0xFFFFFFFF	R/W	Specifies FIFO buffer error threshold that must be exceeded before an alarm is generated: • Lower word is Over Flow/Under Flow (Slip) Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF) • Upper word is Frame Error Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF)
SCXBP_DRIVER _ERR_THRESH	4	0xFFFFFFFF	R/W	Specifies SCX160 driver error threshold that must be exceeded before an alarm is generated: • Lower word is SCbus Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF) • Upper word is SCxbus Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF)
SCXBP_SC_ CLOCK_MODE	1	Default SCbus clocking configured at download time.	R/W	Specifies clocking for SCbus interface of the SCX160 SCxbus Adapter. The SCbus interface can be set to any of the following: • SCXCM_MASTER - provides SCbus clock • SCXCM_SLAVE - receives clock from another SCbus device • SCXCM_FALLBACK_ MASTER - receives clock

Define	B	Default	R/W	Description
				from another SCbus device. However, it is armed to provide master SCbus clock if the master clock fails.
SCXBP_SC_MAP	2	SCbus data stream mapping configured at download time.	R	The SCbus data stream bitmap. The values are as follows: • 0x0000 indicates that no SCbus data streams are assigned • 0x0001 indicates that the first of 16 SCbus data streams is assigned • 0xFFFF indicates that all SCbus data streams are assigned
SCXBP_SCX_ MAP	2	SCxbus data stream mapping configured at download time.	R	The SCxbus data stream bitmap. The values are as follows: • 0x0000 indicates that no SCxbus data streams are assigned • 0x0001 indicates that the first of 16 SCxbus data streams is assigned • 0xFFFF indicates that all SCxbus data streams are assigned
SCXBP_CLK_ HISTORY	1	0	R	SCX160 Clock History Bitmask. Upon detection of a SCX160 Clock event, the value of this parameter may be retrieved to determine the specific clock event that was detected. When the SCXST_CLK_RECOVER bit is set in conjunction with

Define	B	Default	R/W	Description
				another set bit, then the clock with the set bit is recovered. Otherwise, when the SCXST_CLK_RECOVER bit is 0, then any other set bit indicates that the clock with the set bit failed. The value of this parameter is set to 0 after the parameter is read. Possible values may be any combination of the following: • SCXST_SCX_RCLOCK - SCxbus Right Clock failed or recovered • SCXST_SCX_LCLOCK - SCxbus Left Clock failed or recovered • SCXST_SC_CLOCK - SCbus Clock failed or recovered • SCXST_SC_CLOCK_FAIL_SIG - SCbus Clock Fail Signal • SCXST_EXTCLOCK - SCX160 External Clock failed or recovered • SCXST_PLL_REF - SCX160 PLL Reference failed or recovered • SCXST_INTCLOCK - SCX160 Internal Clock failed or recovered • SCXST_CLK_RECOVER - Recovery of Clock
SCXBP_CLK_STATE_BYTE	1	0	R	SCX160 Clock State Bitmask. The value of this parameter

Define	B	Default	R/W	Description
				may be retrieved to determine the current clock state of the SCX160 see (*Chapter 4. Clocking*). When set, indicates loss of clock. Possible values may be any combination of the following: • SCXST_SCX_RCLOCK - SCxbus Right Clock state • SCXST_SCX_LCLOCK - SCxbus Left Clock state • SCXST_SC_CLOCK - SCbus Clock state • SCXST_SC_CLOCK_FAIL_SIG - SCbus Clock Fail Signal state • SCXST_EXTCLOCK - SCX160 External Clock state • SCXST_PLL_REF - SCX160 PLL Reference state • SCXST_INTCLOCK - SCX160 Internal Clock state
SCXBP_SCX_FIFO_STAT	1	0	R	SCX160 FIFO Fail Bitmask. Upon detection of a SCX160 FIFO Error, the value of this parameter may be retrieved to determine the specific FIFO error detected. Possible values may be any combination of the following: • SCXFS_SC_POS_SLIP - SCbus FIFO Positive Slip Exceeded • SCXFS_SC_NEG_SLIP - SCbus FIFO Negative Slip Exceeded

Define	B	Default	R/W	Description
				• SCXFS_SCX_POS_SLIP - SCxbus FIFO Positive Slip Exceeded • SCXFS_SCX_NEG_SLIP - SCxbus FIFO Negative Slip Exceeded • SCXFS_SC_SKEW - SCbus FIFO Skew Exceeded • SCXFS_SCX_SKEW - SCxbus FIFO Skew Exceeded
SCXBP_SCX_BER_STAT	1	0	R	SCX160 Bit-Error-Rate Bitmask. Upon detection of a SCX160 Bus Driver Error, the value of this parameter may be retrieved to determine the specific error detected. Possible values may be any combination of the following: • SCXBE_SC_BER - SCbus BER Threshold Exceeded • SCXBE_SCX_BER - SCxbus BER Threshold Exceeded
SCXBP_SCX_BER_HISTORY	4	0	R	SCX160 Bit-Error-Rate History Bitmask. Upon detection of an SCX160 Bus Driver Error, the value of this parameter may be retrieved to determine the specific error detected. Possible values may be any combination of the following: The first two bytes provide the SCbus Bus Driver Error bitmap. The values are as

Define	B	Default	R/W	Description
				follows: • 0x0000 indicates that no SCbus data streams exceeded the BER threshold • 0x0001 indicates that the first of 16 SCbus data streams exceeded BER threshold, and • 0xFFFF indicates that all SCbus data streams exceeded BER threshold The second two bytes provide the SCxbus Bus Driver Error bitmap. The values are as follows: • 0x0000 indicates that no SCxbus data streams exceeded BER threshold • 0x0001 indicates that the first of 16 SCxbus data streams exceeded BER threshold, etc.
SCXBP_SC_DR_ ERR_CNT	32	none	R	SCbus Driver Error Count for each data stream is stored in a 16 bit counter.
SCXBP_SCX_DR_ ERR_CNT	32	none	R	SCxbus Driver Error Count for each data stream is stored in a 16 bit counter.
SCXBP_SC_FIFO _OVER_CNT	24	none	R	SCbus FIFO Overflow Error Count. 12 - two byte hourly error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SCX_ FIFO_OVER_CNT	24	none	R	SCxbus FIFO Overflow Error Count. 12 - two byte hourly

Define	B	Default	R/W	Description
				error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SC_FIFO _UNDR_CNT	24	none	R	SCbus FIFO Underflow Error Count. 12 - two byte hourly error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SCX_ FIFO_UNDR_CNT	24	none	R	SCxbus FIFO Underflow Error Count. 12 - two byte hourly error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SC_FIFO _SKEW_CNT	24	none	R	SCbus FIFO Skew Error Count. 12 - two byte hourly error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SCX_ FIFO_SKEW_CNT	24	none	R	SCxbus FIFO Skew Error Count. 12 - two byte hourly error count windows: • window 0 is error count for current hour • window 11 is hourly error count from 11 hours ago
SCXBP_SCX_	1	none	R	SCX160 Board Status Bitmask.

Define	B	Default	R/W	Description
BOARD_STAT				This parameter may be retrieved to determine the current status of the SCX160 board (see *Chapter 4. Clocking*). Possible values may be any combination of the following: SCXBS_SELF_TEST - SCX160 Board Self Test Results (Bit 1): • 0 - Passed Self Test • 1 - Failed Self Test SCXBS_DOWNLOAD - SCX160 Board Download State (Bit 2): • 0 - Not Downloaded • 1 - Downloaded SCXBS_RUN_STATE - SCX160 Firmware Run State (Bit 3): • 0 - Not Started • 1 - Started SCXBS_SCX_CLOCK_ STATE - SCxbus Clock State (Bits 4-6); possible values are: • SCX_MASTER - operating as SCxbus Clock Master • SCX_WAIT_SLAVE - operating as SCxbus Clock Master. Waiting to become SCxbus Clock Slave • SCX_SLAVE - operating as SCxbus Clock Slave • SCX_WAIT_FB_MASTER - operating as SCxbus Clock

Define	B	Default	R/W	Description
				Slave. Waiting to become SCxbus Clock Master • SCX_FB_MASTER - operating as SCxbus Clock Fallback Master • SCX_WAIT_MASTER - operating as SCxbus Clock Fallback Master. Waiting to become SCxbus Clock Master • SCX_FB_WAIT_SLAVE - operating as SCxbus Clock Fallback Master. Waiting to become SCxbus Clock Slave • SCXBS_SC_CLOCK_ STATE - SCbus Clock State (Bits 7-8); possible values are: • SC_MASTER - operating as SCbus Clock Master • SC_WAIT_MASTER - Waiting to become SCbus Clock Master • SC_SLAVE - operating as SCbus Clock Slave • SC_FB_MASTER - operating as SCbus Clock Fallback Master

Legend:
- **B** (2nd) column lists the length in Byte(s)
- **R/W** column: **R** = Read; **W** = Write

Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int  devh;                          /* SCX160 device handle */
unsigned char clock_mode;           /* SCxbus Clock Mode    */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n", errno);
     exit(1);
}

/* Get the SCxbus clock mode */
if (scx_getbrdparm(devh, SCXBP_SCX_CLOCK_MODE, &clock_mode) == -1) {
     printf("ERROR: scx_getbrdparm( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}

printf("SCX160 SCxbus clock mode is %s \n",
       (clock_mode == SCXCM_MASTER) ? "MASTER" :
       ((clock_mode == SCXCM_SLAVE) ? "SLAVE" : "FALLBACK MASTER"));

/* Close the SCX160 Board */
scx_close(devh);
```

Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

See Also

- **scx_setbrdparm( )**

Name:	int scx_getctinfo(devh, ct_devinfop)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	CT_DEVINFO *ct_devinfop	• pointer to device information structure
Returns:	0 on success -1 on failure	
Includes:	dxxxlib.h scxlib.h	
Category:	Attribute	
Mode:	synchronous	

■ Description

The **scx_getctinfo( )** function gets Adapter device information.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
ct_devinfop:	Specifies a pointer to the CT_DEVINFO structure that will contain the SCX160 SCxbus Adapter device information.

On return from the function, the CT_DEVINFO structure contains the relevant information and is declared as follows:

```
typedef struct {
     unsigned long   ct_prodid;
     unsigned char   ct_devfamily;
     unsigned char   ct_devmode;
     unsigned char   ct_nettype;
     unsigned char   ct_busmode;
     unsigned char   ct_busencoding;
     unsigned char   ct_rfu[7];          /* reserved for future use */
} CT_DEVINFO;
```

Valid values for each of the members of the CT_DEVINFO structure are defined in the *dxxxlib.h* and *scxlib.h* header files.

The *ct_prodid* field contains a valid Dialogic product identification number for the device:

- **CT_PIDSCX160** - identifies the product as an SCX160 SCxbus Adapter

The *ct_devfamily* specifies the device family and contains the following:

- **CT_DFSCX** - specifies an SCX160 SCxbus Adapter device

The *ct_devmode* field does not apply. The *ct_nettype* field specifies the network interface used by the SCX160 SCxbus Adapter. The only valid value is:

- **CT_NTNONE** - specifies NO interface to the telephone network

The *ct_busmode* specifies the bus architecture used to communicate with other devices in the system:

- **CT_BMSCBUS** - SCbus architecture

The *ct_busencoding* field is not valid for SCX160 SCxbus Adapters.

■ Cautions

This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <scxlib.h>

int       devh;               /* SCX160 device handle */
CT_DEVINFO ct_devinfo;        /* Device information structure */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open scxB1.  errno = %d \n", errno);
     exit(1);
}

/* Get Device Information */
if (scx_getctinfo(devh, &ct_devinfo) == -1) {
     printf("ERROR: scx_getctinfo( ) failed, %s\n", ATDV_ERRMSGP(devh));
     exit(1);
}

printf("%s Device Information:\n", ATDV_NAMEP(devh));
printf("Product ID = 0x%lX, Device Family = %d\n",
          ct_devinfo.ct_prodid, ct_devinfo.ct_devfamily);
printf("Network Mode = %d, Bus Mode = %d\n",
          ct_devinfo.ct_nettype, ct_devinfo.ct_busmode);

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

Name:	int scx_getevtmsk (devh, bitmask)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	unsigned short *bitmask	• pointer to where the event bitmask will be returned
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **scx_getevtmsk( )** function retrieves alarm event mask. This function identifies the events enabled for a specified station.

Parameter	Description
devh:	SCX160 SCxbus Adapter device handle
bitmask:	Specifies the pointer to the buffer where the bitmask will be returned. The bitmask retrieved as a result of this call may be any combination of the following event mask equates: • SCXMM_EXT_CLOCK - external clock event mask • SCXMM_FIFO - SCX160 FIFO event mask • SCXMM_SC_CLOCK - SCbus clock event mask • SCXMM_SC_DRIVER - SCbus driver event mask • SCXMM_SCX_CLOCK - SCxbus clock event mask • SCXMM_SCX_DRIVER - SCxbus driver event mask

When an SCX160 SCxbus Adapter alarm event is generated, the event code returned by the **sr_getevttype( )** function is TSCX_ALARM. The data field returned by the **sr_getevtdatap( )** function contains the specific alarm information and is two bytes long. The first byte represents one of the following bitmask values:

Equate	Description
SCXEV_SCBUS_CLOCK	SCbus clock event
SCXEV_SCXBUS_CLOCK	SCxbus clock event
SCXEV_EXT_CLOCK	External clock event
SCXEV_SCBUS_DRIVER	SCbus driver event
SCXEV_SCXBUS_DRIVER	SCxbus driver event
SCXEV_FIFO	SCX160 FIFO event

The second byte provides additional details about the event reported by the first byte as follows. If any clock event is returned, then the second byte represents one of the following:

Equate	Description
SCXST_SCX_RCLOCK	SCxbus Right Clock failed/recovered
SCXST_SCX_LCLOCK	SCxbus Left Clock failed/recovered
SCXST_SC_CLOCK	SCbus Clock failed/recovered
SCXST_SC_CLOCK_FAIL_SIG	SCbus Clock Fail Signal failed/recovered
SCXST_EXTCLOCK	SCX160 External Clock failed/recovered
SCXST_PLL_REF	SCX160 PLL Reference failed/recovered
SCXST_INTCLOCK	SCX160 Internal Clock failed/recovered
SCXST_CLK_RECOVER	Recovery of Clock

If any driver event is returned, then the second byte represents either of the following:

Equate	Description
SCXBE_SC_BER	SCbus BER Threshold Exceeded
SCXBE_SCX_BER	SCxbus BER Threshold Exceeded

When a FIFO event is returned, then the second byte represents one of the following:

Equate	Description
SCXFS_SC_POS_SLIP	SCbus FIFO Positive Slip Threshold Exceeded
SCXFS_SC_NEG_SLIP	SCbus FIFO Negative Slip Threshold Exceeded
SCXFS_SCX_POS_SLIP	SCxbus FIFO Positive Slip Threshold Exceeded
SCXFS_SCX_NEG_SLIP	SCxbus FIFO Negative Slip Threshold Exceeded
SCXFS_SC_SKEW	SCbus FIFO Skew Threshold Exceeded
SCXFS_SCX_SKEW	SCxbus FIFO Skew Threshold Exceeded

See *Section 9.2. Handling Events* for an example of handling unsolicited events.

■ Cautions

This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int  devh;                          /* SCX160 device handle */
unsigned short event_mask;          /* Event Mask           */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n", errno);
     exit(1);
}

/* Get the SCX160 Event Mask */
if (scx_getevtmsk(devh, &event_mask) == -1) {
     printf("ERROR: scx_getevtmsk( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}

/* Print out the SCX160 Event Mask */
printf("SCxbus Clock Fail Alarm Reporting is %s \n",
       (event_mask & SCXMM_SCX_CLOCK) ? "ENABLED" : "DISABLED");
printf("SCbus Clock Fail Alarm Reporting is %s \n",
       (event_mask & SCXMM_SC_CLOCK) ? "ENABLED" : "DISABLED");
printf("External Clock Fail Alarm Reporting is %s \n",
       (event_mask & SCXMM_EXT_CLOCK) ? "ENABLED" : "DISABLED");
printf("SCX160 Fifo Alarm is Reporting %s \n",
       (event_mask & SCXMM_FIFO) ? "ENABLED" : "DISABLED");
printf("SCX160 SCbus Driver Fail Alarm Reporting is %s \n",
       (event_mask & SCXMM_SC_DRIVER) ? "ENABLED" : "DISABLED");
printf("SCX160 SCxbus Driver Fail Alarm Reporting is %s \n",
       (event_mask & SCXMM_SCX_DRIVER) ? "ENABLED" : "DISABLED");

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

■ See Also

- **scx_clrevtmsk( )**
- **scx_setevtmsk( )**

Name:	int scx_gtbdattr(devh, pbrdst)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	tSCX_BRDST *pbrdst	• Pointer to a board attribute structure
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Attribute	
Mode:	synchronous	

■ Description

The **scx_gtbdattr( )** function gets board attributes of hardware and firmware information for a specific SCX160 SCxbus Adapter. The returned information is contained in the tSCX_BRDST data structure:

```
typedef struct {
     unsigned char board_status;
     unsigned char self_test_code;
     unsigned char board_type;
} tSCX_BRDST;
```

Valid values for each member of this structure are defined in the scxlib.h header. The *board status* field contains a bitmap of the current status of the SCX160 SCxbus Adapter.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle
pbrdst:	Points to where the board attributes information is stored in the tSCX_BRDST data structure

The possible values of the fields in the tSCX_BRDST data structure can be any combination of the following:

Parameter	Description	Bit(s)	Possible Values
SCXBS_SELF_ TEST	SCX160 Board Self Test Results	1	• 0 - passed self test • 1 - failed self test
SCXBS_ DOWNLOAD	SCX160 Board Download State	2	• 0 - not downloaded • 1 - downloaded
SCXBS_RUN_ STATE	SCX160 Firmware Run State	3	• 0 - not started • 1 - started
SCXBS_SCX_ CLOCK_ STATE	SCxbus Clock State	4 - 6	• SCX_MASTER - operating as SCxbus Clock Master • SCX_WAIT_SLAVE - operating as SCxbus Clock Master; waiting to become SCxbus Clock Slave • SCX_SLAVE - operating as SCxbus Clock Slave • SCX_WAIT_FB_MASTER - operating as SCxbus Clock Slave; waiting to become SCxbus Clock Master • SCX_FB_MASTER - operating as SCxbus Clock Fallback Master • SCX_WAIT_MASTER - operating as SCxbus Clock Fallback Master; waiting to become SCxbus Clock Master • SCX_FB_WAIT_SLAVE - operating as SCxbus Clock Fallback Master; waiting to become SCxbus Clock Slave
SCXBS_SC_	SCbus Clock	7 - 8	• SC_MASTER - operating as

Parameter	Description	Bit(s)	Possible Values
CLOCK_ STATE	State		SCbus Clock Master • SC_WAIT_MASTER - waiting to become SCbus Clock Master • SC_SLAVE - operating as SCbus Clock Slave • SC_FB_MASTER - operating as SCbus Clock Fallback Master

■ Cautions

This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

const char *sc_state[] = {
     "SC_MASTER",
     "SC_WAIT_MASTER",
     "SC_SLAVE",
     "SC_FB_MASTER"
};

const char *scx_state[] = {
     "SCX_MASTER",
     "SCX_WAIT_SLAVE",
     "SCX_SLAVE",
     "SCX_WAIT_FB_MASTER",
     "SCX_FB_MASTER",
     "SCX_WAIT_MASTER",
     "SCX_WAIT_FB_SLAVE"
};

int  devh;                  /* SCX160 device handle         */
tSCX_BRDST brdst;           /* SCX160 Board Attribute Block */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n ", errno);
     exit(1);
```

```
}

/* Get the SCX160 board attributes  */
if (scx_gtbdattr(devh, &brdst) == -1) {
     printf("ERROR: scx_gtbdattr( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}
printf("SCX Clock State is %s \n",
       scx_state[((brdst.board_status & SCXBS_SCX_CLOCK_STATE) >> 3)]);
printf("SC Clock State is %s \n",
       sc_state[((brdst.board_status & SCXBS_SC_CLOCK_STATE) >> 6)]);

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

Name: int scx_open(name, oflags)

Inputs:
- char *name — • SCX160 SCxbus Adapter logical board or device name
- short oflags — • open attribute flags; reserved for future use

Returns: device handle on success
-1 on failure

Includes: scxlib.h

Category: Configuration

Mode: synchronous

■ Description

The **scx_open( )** function opens an SCX160 device and returns a unique Dialogic handle to identify the device. All subsequent references to the opened device must use the returned device handle.

Opening an SCX160 SCxbus Adapter does not alter the state of the device. A device may only be opened once and cannot be re-opened by the current process or any other process until the device is closed. A command is only accepted while a device is idle.

NOTE: If a parent process opens an SCX160 SCxbus Adapter and enables events, there is no guarantee that the child process will receive a particular event. You should only enable events in the process that opened the SCX160 device.

Parameter	Description
name:	Points to an ASCIIZ string that contains the name (scxB1) of the SCX160 SCxbus Adapter.
oflags:	Reserved for future use: set to 0.

■ Cautions

This function will fail if an invalid device handle is specified or if the device is already open.

Never open a Dialogic device using the Windows **open( )** function.

Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int  devh;                          /* SCX160 device handle */
unsigned short event_mask;          /* Event Mask           */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
     printf("Cannot open board scxB1.  errno = %d \n ", errno);
     exit(1);
}

/*
 * Disable the reporting of all clocking alarms on the SCbus
 * and on the SCxbus.
 */
event_mask = (SCXEV_SCXBUS_CLOCK_FAIL    | SCXEV_SCXBUS_CLOCK_FAIL |
              SCXEV_EXT_CLOCK_FAIL);
if (scx_clrevtmsk(devh, event_mask) == -1) {
     printf("ERROR: scx_clrevtmsk( ) failed, %s \n", ATDV_ERRMSGP(devh));
     exit(1);
}

/* Close the SCX160 Board */
scx_close(devh);
```

Errors

If the function returns -1, the specific error that occurred is returned in the errno global variable.

See Also

- **scx_close( )**

Name: int scx_setbrdparm (devh, param, valuep)

Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	unsigned char param	• parameter name
	void *valuep	• pointer to the location of the parameter value
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **scx_setbrdparm( )** function changes board level parameters.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
param:	Specifies the name of the parameter, see *Table 13. SCX160 User Selectable Parameters*, below for parameter defines and their description.
valuep:	Points to the location of the parameter value.

The SCXBP_FIFO_ERR_THRESH parameter specifies the FIFO buffer error threshold. Most FIFO errors occur due to clock reference drift, maintenance activities on the network and any other activity that impacts the clock reference signal. Therefore, error rates will vary depending on the environment in which the SCX160 SCxbus Adapter system operates. Each user must determine an acceptable error threshold for their system. A typical rate of 255 slips in a 24 hour period (error threshold setting of 11 slips/hour) would not be uncommon.

The SCXBP_DRIVER_ERR_THRESH parameter specifies the SCX160 bus driver error threshold. Driver errors reflect impairments due to transmission line inefficiency, bus contention, excessive cable length, improper SCSI-3 termination and similar phenomena. Typically, copper-based transmission lines can be expected to offer a Bit Error Rate (BER) of better than 1 x 10-6 (1 error per 1 million bits). Each user must determine an acceptable BER threshold for their system while keeping in mind that each data stream runs at 4.096 M bits/sec. A

threshold of 14,000 errors/hour (less than 1 error per million bits) may be appropriate for a network with moderate to heavy traffic between nodes.

Table 13. SCX160 User Selectable Parameters

Define	B	Default	R/W	Description
SCXBP_SCX_ CLOCK_MODE	1	Default SCxbus clocking configured at download time.	R/W	Specifies clocking for the SCxbus interface of the SCX160 SCxbus Adapter. The SCxbus interface can be set to any of the following: • SCXCM_MASTER - provides right SCxbus clock • SCXCM_SLAVE - receives clock from another SCxbus device • SCXCM_FALLBACK_MAS TER - receives right clock from another SCxbus device, and generates left SCxbus clock to be used by other SCxbus devices if the right SCxbus clock is unavailable
SCXBP_CLOCK_ REF	1	Default SCX160 Clock Reference Signal configured at download time.	R/W	Specifies source of the SCX160 SCxbus Adapter clock reference signal. The clock source can be any one of the following: • SCXCS_INTERNAL - use Internal Clock • SCXCS_SCBUS - derive clock from SCbus • SCXCS_SCXBUS_RIGHT - derive clock from SCxbus Right Clock • SCXCS_SCXBUS_LEFT - derive clock from SCxbus

Define	B	Default	R/W	Description
				Left Clock • SCXCS_EXTERNAL - use External 8 kHz Clock
SCXBP_FIFO_ ERR_THRESH	4	0xFFFFFFF F	R/W	Specifies FIFO buffer error threshold that must be exceeded before an alarm is generated: • Lower word is Over Flow/Under Flow (Slip) Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF) • Upper word is Frame Error Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF)
SCXBP_DRIVER _ERR_THRESH	4	0xFFFFFFF F	R/W	Specifies SCX160 driver error threshold that must be exceeded before an alarm is generated. • Lower word is SCbus Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF) • Upper word is SCxbus Threshold (Valid Range: 0 ≤ Thresh ≤ 0xFFFF)
SCXBP_SC_ CLOCK_MODE	1	Default SCbus clocking configured at download time.	R/W	Specifies clocking for SCbus interface of the SCX160 SCxbus Adapter. The SCbus interface can be set to any of the following: • SCXCM_MASTER - provides SCbus clock • SCXCM_SLAVE - receives clock from another SCbus device • SCXCM_FALLBACK_ MASTER - receives clock from another SCbus device,

Define	B	Default	R/W	Description
				however, is ready to provide master SCbus clock if the master clock fails.

Legend: **B** (2nd) column lists the length in Byte(s);
R/W column: **R** = Read; **W** = Write

■ Cautions

This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int      devh;                                  /* SCX160 device handle */
unsigned   char   parm_val = SCXCM_MASTER;      /* Parameter Value      */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
      printf("Cannot open board scxB1.  errno = %d \n", errno);
      exit(1);
}

/* Set SCxbus clock mode to Master */
if (scx_setbrdparm(devh, SCXBP_SCX_CLOCK_MODE, &parm_val) == -1) {
      printf("ERROR: scx_setbrdparm( ) failed, %s \n", ATDV_ERRMSGP(devh));
      exit(1);
}

/* Close the SCX160 Board */
scx_close(devh);
```

Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

See Also

- **scx_getbrdparm( )**

Name:	int scx_setevtmsk(devh, mask)	
Inputs:	int devh	• SCX160 SCxbus Adapter device handle
	unsigned short mask	• Event mask
Returns:	0 on success -1 on failure	
Includes:	scxlib.h	
Category:	Configuration	
Mode:	synchronous	

■ Description

The **scx_setevtmsk( )** function enables alarm events specified in the event mask. Calling this function may cause an immediate event reflecting the current state of the SCX160 SCxbus Adapter if that state is part of the message mask being enabled.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
mask:	Specifies the event mask. Possible event masks may be any combination of the following event mask equates: • SCXMM_EXT_CLOCK - external clock event mask • SCXMM_FIFO - SCX160 FIFO event mask • SCXMM_SC_CLOCK - SCbus clock event mask • SCXMM_SC_DRIVER - SCbus driver event mask • SCXMM_SCX_CLOCK - SCxbus clock event mask • SCXMM_SCX_DRIVER - SCxbus driver event mask

The event code returned by the **sr_getevttype( )** function is TSCX_ALARM. The data field returned by the **sr_getevtdatap( )** function contains the event detected and is two bytes long. The first byte represents one of the following bitmask values:

Equate	Description
SCXEV_SCBUS_CLOCK	SCbus clock event
SCXEV_SCXBUS_CLOCK	SCxbus clock event
SCXEV_EXT_CLOCK	External clock event
SCXEV_SCBUS_DRIVER	SCbus driver event
SCXEV_SCXBUS_DRIVER	SCxbus driver event
SCXEV_FIFO	SCX160 FIFO event

The second byte provides additional details about the event reported by the first byte as follows. If any clock event is returned, then the second byte represents one of the following:

Equate	Description
SCXST_SCX_RCLOCK	SCxbus Right Clock failed/recovered
SCXST_SCX_LCLOCK	SCxbus Left Clock failed/recovered
SCXST_SC_CLOCK	SCbus Clock failed/recovered
SCXST_SC_CLOCK_FAIL_SIG	SCbus Clock Fail Signal failed/recovered
SCXST_EXTCLOCK	SCX160 External Clock failed/recovered
SCXST_PLL_REF	SCX160 PLL Reference failed/recovered
SCXST_INTCLOCK	SCX160 Internal Clock failed/recovered
SCXST_CLK_RECOVER	Recovery of Clock

If any driver event is returned, then the second byte represents one of the following:

Equate	Description
SCXBE_SC_BER	SCbus BER Threshold Exceeded
SCXBE_SCX_BER	SCxbus BER Threshold Exceeded

When a FIFO event is returned, then the second byte represents one of the following:

Equate	Description
SCXFS_SC_POS_SLIP	SCbus FIFO Positive Slip Threshold Exceeded
SCXFS_SC_NEG_SLIP	SCbus FIFO Negative Slip Threshold Exceeded
SCXFS_SCX_POS_SLIP	SCxbus FIFO Positive Slip Threshold Exceeded
SCXFS_SCX_NEG_SLIP	SCxbus FIFO Negative Slip Threshold Exceeded
SCXFS_SC_SKEW	SCbus FIFO Skew Threshold Exceeded
SCXFS_SCX_SKEW	SCxbus FIFO Skew Threshold Exceeded

See *Section 9.2. Handling Events* for an example of handling unsolicited events.

■ Cautions

This function enables notification only for the events specified. If notification for other events was enabled or disabled previously, the status of the notification will not change.

This function will fail if an invalid device handle is specified.

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int      devh;                    /* SCX160 device handle  */
unsigned   short event_mask;      /* Event Mask            */

/* Open SCX160 board */
if ((devh = scx_open("scxB1", 0)) == -1) {
      printf("Cannot open board scxB1.  errno = %d \n ", errno);
      exit(1);
}

/*
 * Enable the reporting of all clocking alarms on the SC bus
 * and on the SCX bus
 */
event_mask = (SCXMM_SCX_CLOCK | SCXMM_SC_CLOCK | SCXMM_EXT_CLOCK);
if (scx_setevtmsk(devh, event_mask) == -1) {
      printf("ERROR: scx_setevtmsk( ) failed, %s \n", ATDV_ERRMSGP(devh));
      exit(1);
}

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

■ See Also

- **scx_clrevtmsk( )**
- **scx_getevtmsk( )**

Name: int scx_tstcom(devh, tmo)
Inputs: int devh • SCX160 SCxbus Adapter device handle
short tmo • time-out value
Returns: 0 on success
-1 on failure
Includes: scxlib.h
Category: Diagnostic
Mode: synchronous

■ Description

The **scx_tstcom( )** function runs a communication test to verify that the SCX160 SCxbus Adapter can communicate with the host computer. Usually performed during Dialogic Service startup, the test sends a single byte to the SCX160 SCxbus Adapter and waits for a single-byte response. A return code of 0 indicates successful completion of the test.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
tmo:	Specifies the maximum amount of time in seconds that the function will block while waiting for a response from the SCX160 SCxbus Adapter. If a response is not received within tmo seconds, an error is returned. A suggested setting is 5 seconds.

■ Cautions

This function fails if:

- The device fails to responds within tmo seconds
- An invalid device handle is specified
- A hardware problem exists on the SCX160 SCxbus Adapter
- A configuration problem exists; for example, an IRQ conflict

NOTE: Device configuration information is located in the appropriate hardware installation card.

Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#include <srllib.h>
#include <scxlib.h>

int devh;               /* SCX160 Device Handle */
int rcode;              /* Function Return Code */

/* Open SCX160 Board */
if ((devh = scx_open("scxB1", 0)) == -1) {
   printf("Cannot open board scxB1. errno=%d \n", errno);
   exit(1);
}

/* Perform Communications Test with SCX160 Board */
rcode = scx_tstcom(devh, 5);
if (rcode == -1) {
   printf("ERROR: SCX/160 Communications Test Failed on Board scxB1: %s\n",
          ATDV_ERRMSGP(devh));
} else {
   printf("SCX/160 Communications Test was Successful on Board scxB1\n");
}

/* Close the SCX160 Board */
scx_close(devh);
```

Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

See Also

- **scx_tstdat( )**

Name: int scx_tstdat(devh, tmo)
Inputs: int devh • SCX160 SCxbus Adapter device handle
short tmo • time-out value
Returns: 0 on success
-1 on failure
Includes: scxlib.h
Category: Diagnostic
Mode: synchronous

■ Description

The **scx_tstdat( )** function runs a data test to verify the integrity of the host computer-to-SCX160 SCxbus Adapter Shared RAM interface for the requested SCX160 SCxbus Adapter board. Usually performed during Dialogic Service startup, the data test sends a series of bytes to the SCX160 SCxbus Adapter. The same data is returned to the host computer and checked to verify the integrity of the interface. A return code of 0 indicates successful completion of the test.

Parameter	Description
devh:	Specifies the SCX160 SCxbus Adapter device handle.
tmo:	Specifies the maximum amount of time in seconds that the function will block while waiting for a response from the SCX160 SCxbus Adapter. If a response is not received within tmo seconds, an error is returned. A suggested setting is 5 seconds.

■ Cautions

This function fails if any of the following occur:

- The device fails to responds within tmo seconds
- The test data is corrupted
- An invalid device handle is specified
- A hardware problem exists on the SCX160 SCxbus Adapter
- A configuration problem exists; for example, an IRQ conflict

■ Example

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <srllib.h>
#include <scxlib.h>

int devh;               /* SCX160 Device Handle */
int rcode;              /* Function Return Code */

/* Open SCX160 Board */
if ((devh = scx_open("scxB1", 0)) == -1) {
   printf("Cannot open board scxB1. errno=%d \n", errno);
   exit(1);
}

/* Perform Data Test with SCX160 Board */
rcode = scx_tstdat(devh, 5);
if (rcode == -1) {
   printf("ERROR: SCX/160 Data Test Failed on Board scxB1: %s\n",
          ATDV_ERRMSGP(devh));
} else {
   printf("SCX/160 Data Test was Successful on Board scxB1\n");
}

/* Close the SCX160 Board */
scx_close(devh);
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code, or you can use the **ATDV_ERRMSGP( )** function to obtain a descriptive error message. The error codes returned by **ATDV_LASTERR( )** function are listed in *Chapter 7. Errors*.

■ See Also

- **scx_tstcom( )**

7. Errors

A description of error handling and retrieval, including a summary of possible errors returned, is presented in this chapter.

7.1. Error Handling and Retrieval

All library functions return a value to the application that indicates the success or failure of the function call. A function can return either of the following:

- 0 indicating success
- -1 indicating the function failed

NOTE: The **scx_open( )** function returns the device handle on success instead of 0. However, like the other functions, **scx_open( )** returns -1 on failure.

When a function returns a -1, the specific error code indicating the cause of the failure can be retrieved by the **ATDV_LASTERR( )** function found in the standard runtime library. The **ATDV_ERRMSGP( )** SRL function may be called to obtain the descriptive ASCII error message returned by the function that failed.

7.2. Possible Error Codes

Error codes returned for SCX160 SCxbus Adapter functions are listed in *Table 14. SCX160 SCxbus Adapter dtilib.h Error Codes*.

Table 14. SCX160 SCxbus Adapter dtilib.h Error Codes

Error Returned	Description
EDT_BADBRERR	Board missing or defective
EDT_BADCMDERR	Invalid or undefined command
EDT_BADDEV	Invalid device error
EDT_BADGLOB	Invalid global parameter number
EDT_BADVAL	Invalid value
EDT_DATTO	Data reception timed out

Error Returned	Description
EDT_FWERR	Firmware returned an error
EDT_INVBD	Invalid board device
EDT_INVCFG	Invalid configuration area
EDT_INVMSG	Invalid message
EDT_NOMEMERR	Invalid map or memory allocation
EDT_PARAMERR	Invalid command parameter
EDT_SH_BADEXTTS	External time slot unsupported at current clock rate
EDT_SH_BADINDX	Invalid switch handler library index number
EDT_SH_BADLCLTS	Invalid local time slot
EDT_SH_BADTYPE	Invalid local time slot type
EDT_SH_LCLDSCNCT	Local time slot is already disconnected from SCbus
EDT_SH_LCLTSCNCT	Local time slot is already connected to SCbus
EDT_SH_LIBBSY	Switch handler library busy
EDT_SH_LIBNOTINIT	Switch handler library is uninitialized
EDT_SH_MISSING	Switch handler is not present
EDT_SH_NOCLK	No switch handler clock source present
EDT_SYSTEM	Windows system error (actual error in errno global variable)
EDT_TMOERR	Timed out waiting for a response from firmware

8. Data Structure Reference

The SCX160 SCxbus Adapter defines provided in the scxlib.h file and the following data structures, which are used by SCX160 SCxbus Adapter functions, are described in this chapter:

- Board Status (tSCX_BRDST) Data Structure
- Channel/Time slot DEVice INFOrmation (CT_DEVINFO) Structure

8.1. Board Status (tSCX_BRDST) Data Structure

The tSCX_BRDST data structure is used by the **scx_gtbdattr( )** function to retrieve status information about the SCX160 SCxbus Adapter. On return from the function, the tSCX_BRDST data structure contains the relevant information and is defined as follows:

```
typedef struct {
     unsigned char board_status;
     unsigned char self_test_code;
     unsigned char board_type;
} tSCX_BRDST;
```

Valid values for each member of this structure are defined in the scxlib.h header:

- The *board_status* field contains a bitmap of the current status of the SCX160 SCxbus Adapter. See the SCXBP_SCX_BOARD_STAT parameter in *Table 12. SCX160 SCxbus Adapter Parameters* of the **scx_getbrdparm( )** function for possible values
- The *self_test_code* field contains the results of the self test; 0 = passed, 1 = failed the self test
- The *board_type* field is always 0

8.2. Channel/Time slot DEVice INFOrmation (CT_DEVINFO)

The CT_DEVINFO structure supplies information about the SCX160 SCxbus Adapter device. This structure is used by the SCbus routing functions identified

by the suffix, **_getctinfo( )**. On return from the function, the CT_DEVINFO structure contains the relevant information and is defined as follows:

```
typedef struct {
      unsigned long      ct_prodid;
      unsigned char      ct_devfamily;
      unsigned char      ct_devmode;
      unsigned char      ct_nettype;
      unsigned char      ct_busmode;
      unsigned char      ct_busencoding;
      unsigned char      ct_rfu[7];          /* reserved for future use */
} CT_DEVINFO;
```

Valid values for each of the members of the CT_DEVINFO structure are defined in the *dxxxlib.h* and *scxlib.h* header files.

The *ct_prodid* field contains a valid Dialogic product identification number for the device [length: 4 (unsigned long)]:

- **CT_PIDSCX160** - identifies the product as an SCX160 SCxbus Adapter

The *ct_devfamily* specifies the device family [length: 1 (unsigned char)] and contains the following:

- **CT_DFSCX** - specifies an SCX160 SCxbus Adapter device

The *ct_devmode* field does not apply. The *ct_nettype* field specifies the type of network interface for the device [length: 1 (unsigned char)]. The only valid value is:

- **CT_NTNONE** - specifies NO interface to the telephone network

The *ct_busmode* specifies the bus architecture used to communicate with other devices in the system [length: 1 (unsigned char)]:

- **CT_BMSCBUS** - SCbus architecture

The *ct_busencoding* field describes the PCM encoding used on the bus [length: 1 (unsigned char)] and is not valid for SCX160 SCxbus Adapters.

9. Application Guidelines

This chapter offers advice and suggestions to guide programmers in designing and coding a Dialogic SCX160 SCxbus Adapter application for a Windows environment.

9.1. Writing an Application

This chapter offers advice and suggestions to guide programmers in designing and coding a Dialogic SCX160 SCxbus Adapter application for a Windows environment.

This section provides SCX160 SCxbus Adapter programming guidelines:

- General Guidelines
- Initialization

NOTE: These guidelines are not intended as a comprehensive guide to developing or debugging SCX160 SCxbus Adapter applications.

9.1.1. General Guidelines

General guidelines for writing Dialogic applications are explained in the following sections:

- Check return codes
- Use symbolic defines
- Include header files

Check Return Codes

Most SCX160 SCxbus Adapter library functions return an error code if they fail. Any call to an SCX160 SCxbus Adapter library function should therefore check for a return value indicating an error. This can be done using a format similar to the following:

```
if   (scx_xxxxx(devh, arguments) == -1)   {
      printf("ERROR: scx_xxxxx( ) failed: %s (0x%X)\n"'
               ATDV_ERRMSG(devh), ATDV_LASTERR(devh));
      /* Error Handling Routine */
      .
      .
}
/* Successful Function Call   -   Continue Processing ... */
```

Use Symbolic Defines

Applications containing numerical values for defines are subject to obsolescence and require frequent modification. In general, do not use a numerical value in your application when an equivalent symbolic define is available.

Include Header Files

Various header files provided with the Package Release must be included in your application. These files provide equates, structures and prototypes needed to compile application programs.

9.1.2. Initialization

Before an SCX160 SCxbus Adapter application performs any processing, it should initialize the SCX160 SCxbus Adapter hardware to reflect the physical configuration of your system and set the parameters necessary to support the application. Tasks performed as part of initialization generally include:

- Set hardware configuration
- Set event masks for SCX160 SCxbus Adapter alarm to be reported

The tasks above are accomplished using the following library functions:

- **scx_clrevtmsk( )**
- **scx_setbrdparm( )**
- **scx_setevtmsk( )**

Set Hardware Configuration

Use **scx_setbrdparm( )** to set hardware configuration, clocking configuration and error threshold levels. See **scx_setbrdparm( )** function in *Chapter 6. Function Reference* for specific settings.

Set Event Mask on SCX160 SCxbus Adapter

Use the **scx_setevtmsk( )** function for setting and the **scx_clrevtmsk( )** function for clearing the mask for alarm events.

9.2. Handling Events

Events can be one of two types: solicited or unsolicited. Solicited events occur when a multitasking function terminates. Unsolicited events are typically generated by an occurrence external to the application, such as a clock error or exceeding error thresholds. Your application should be set up to handle both kinds of events.

The following example illustrates handling unsolicited events such as processing SCX160 clocking alarms:

```
#include <windows.h>
#include <stdio.h>
#include <srllib.h>
#include <dtilib.h>
#include <scxlib.h>


int ProcessClockAlarms(short ScxDev)
{
   unsigned char        ClockState;
   unsigned char        ClockEvent;
   unsigned char        EventType;
   unsigned char        *EventDatap;
   unsigned short int   EventMask;


   /* Enable reporting of SCX160 Clock Alarms */
   EventMask = SCXMM_SC_CLOCK | SCXMM_SCX_CLOCK | SCXMM_EXT_CLOCK;
   if (scx_setevtmsk(ScxDev, EventMask) == -1) {
      printf("ERROR: scx_setevtmsk( ) failed: %s (%d)\n",
             ATDV_ERRMSGP(ScxDev), ATDV_LASTERR(ScxDev));
      return(-1);
   }


   /* Loop Forever processing SCX160 Clock Alarm Events */
   while (1) {
      /*   Block until an event is retrieved */
      sr_waitevt(-1);


      /* Wait for a SCX160 Alarm Event - Ignore all other events */
      if ((sr_getevtdev( ) != ScxDev) || (sr_getevttype( ) != TSCX_ALARM)) {
         continue;
      }


      /* Perform processing dependent upon the type of the event */
      EventDatap = (unsigned char *)sr_getevtdatap( );
      EventType = *EventDatap++;
      switch (EventType) {
         case SCXEV_SCBUS_CLOCK:
            printf("SCbus Clock Event Detected\n");
            break;


         case SCXEV_SCXBUS_CLOCK:
            printf("SCxbus Clock Event Detected\n");
            break;


         case SCXEV_EXT_CLOCK:
            printf("SCxbus EXTERNAL Clock Event Detected\n");
            break;


         default:
            printf("Unknown SCX160 Clock Event: 0x%X\n", EventType);
            return(-1);
```

```
}

/* Determine specific SCX160 Clock Failure/Recovery. */
ClockEvent = *EventDatap;
if (ClockEvent & SCXST_CLK_RECOVER) {
   printf("Recovery of ");
}
switch (ClockEvent) {
   case SCXST_SCX_RCLOCK:
      printf("SCX160 SCxbus Right Clock Failure\n");
      break;


   case SCXST_SCX_LCLOCK:
      printf("SCX160 SCxbus Left Clock Failure\n");
      break;


   case SCXST_SC_CLOCK:
      printf("SCX160 SCbus Clock Failure\n");
      break;


   case SCXST_SC_CLOCK_FAIL_SIG:
      printf("SCX160 SCbus Clock Fail Signal\n");
      break;


   case SCXST_EXTCLOCK:
      printf("SCX160 SCxbus External Clock Failure\n");
      break;


   case SCXST_PLL_REF:
      printf("SCX160 PLL Reference Failure\n");
      break;


   case SCXST_INTCLOCK:
      printf("SCX160 Internal Clock Failure\n");
      break
}

/* Display the current status of all SCbus/SCxbus Clocks */
if (scx_getbrdparm(ScxDev, SCXBP_CLK_STATE_BYTE, (void *)&ClockState) == -1) {
   printf("ERROR: scx_getbrdparm( ) failed: %s (%d)\n",
          ATDV_ERRMSGP(ScxDev), ATDV_LASTERR(ScxDev));
   return(-1);
}

printf("\nCurrent State of SCbus/SCxbus Clocks:\n");
printf("     SCX160 SCxbus Right Clock:      %s\n",
      (ClockState & SCXST_SCX_RCLOCK) ? "FAIL" : "OKAY");


printf("     SCX160 SCxbus Left Clock:       %s\n",
      (ClockState & SCXST_SCX_LCLOCK) ? "FAIL" : "OKAY");


printf("     SCX160 SCbus Clock:             %s\n",
      (ClockState & SCXST_SC_CLOCK) ? "FAIL" : "OKAY");


printf("     SCX160 SCbus Clock Fail Signal: %s\n",
```

```
            (ClockState & SCXST_SC_CLOCK_FAIL_SIG) ? "FAIL" : "OKAY");

      printf("     SCX160 SCxbus External Clock:   %s\n",
            (ClockState & SCXST_EXTCLOCK) ? "FAIL" : "OKAY");

      printf("     SCX160 PLL Reference:           %s\n",
            (ClockState & SCXST_PLL_REF) ? "FAIL" : "OKAY");

      printf("     SCX160 Internal Clock:          %s\n",
            (ClockState & SCXST_INTCLOCK) ? "FAIL" : "OKAY");

   }  /* End of WHILE */

   return(0);
}
```

9.3. Aborting

If you abort an SCX160 SCxbus Adapter application by pressing Ctrl C or Ctrl Break keys, the operating system will terminate the current process but may leave devices in an unknown state. The next time you run your application you may encounter errors.

To avoid errors of this type, your application should trap the *Ctrl C* or *Ctrl Break* signal and terminate the application in an orderly manner.

9.4. Terminating the Application

When your process completes, devices should be closed in an orderly fashion.

9.5. Compiling and Linking

The SCX160 SCxbus Adapter application must be compiled and linked with the library files listed in *Section 6.1. Include Files*, which are contained on the Package Release diskettes.

When compiling, byte alignment must be maintained in order for the data structures used by the SCX160 SCxbus Adapter to be accessed correctly. Use caution if you are using a compiler that performs automatic word alignment of data.

10. Demonstration Programs

This appendix contains instructions for using the Dialogic SCX160 SCxbus Adapter Demonstration programs for Windows, a Console Demonstration (*SCxDemo*) Program and a Graphical User Interface (GUI) Demonstration (*WinSCxDemo*) Program. These programs demonstrate how to develop a multi-node system using the SCX160 SCxbus Adapter.

The Demonstration programs were designed using Client/Server architecture. The Client node provides the network interface, including call setup and teardown. Multiple clients can share different resources on multiple servers. Each server can be dedicated to one kind of resource (e.g., voice, fax, etc.). These demonstration programs support voice only.

Each program demonstrates how a Client node and a Server node can be linked by the SCX160 SCxbus Adapter to allow calls to come into the Client machine and be handled by the Server. The Server can play files, record files, get digits, and display the status of each channel. Channel status is displayed at the Client node as well.

NOTE: The Demonstration programs were tested with two nodes only. If you are using more than two nodes, make sure that the client nodes have the same network interface; all digital or all analog for the GUI Demonstration (*WinSCxDemo*) program or all analog for the Console Demonstration (*SCxDemo*) Program.

10.1. Hardware Requirements

The following hardware components are required to set up and run either of the Demonstration Programs:

Client machine with the following boards:

- Digital network interface board(s); such as, DTI/240SC or DTI/241SC for T1 interface, DTI/300SC or DTI/301SC for E1 interface, or analog network interface board(s); such as, LSI/160SC or LSI/161SC

NOTE: The client machine may use any network interface boards, with or without voice resources, but only the network resources are used for demonstration purposes.

- SCX160 Adapter
- Network adapter board

Server machine with the following boards:

- Voice resource board(s); such as, a D/240SC, D/320SC, or D/160SC

  NOTE: The server machine may use any voice resource boards, with or without network interfaces, but only the voice resources are used for demonstration purposes.

- The SCX160 Adapter
- Network adapter board

Station adapter and Type 2500 telephone set for analog interface

Channel bank for digital interface

NOTE: The client and server machines must be on the local area network running a protocol that supports sockets.

10.2. Software Requirements

- Visual C++ 4.0 or later is only required to compile the demonstration program.
- All Demonstration Program files (including voice files such as INTRO.VOX, GOODBYE.VOX) must be in the same directory. The files are loaded when the SCX160 SCxbus Adapter Package software is installed (see the SCX160 SCxbus Adapter Package Release Notes for installation instructions).
- Networking software with a protocol that supports sockets (e.g., TCP/IP) configured on the client and server machines.

10.3. Overview of Demonstration Programs

An overview of the sequence of events demonstrated by the Console and GUI Demonstration Programs and the various channel and application states is presented in the following paragraphs.

Figure 10. Application and Channel States illustrates the application states while waiting for and then processing an inbound call. After receiving an inbound call, the application plays an introductory message and listens for an access code to be entered. If the application does not detect an access code, a message can be recorded; otherwise, if an access code is detected, it is validated and if valid, the previously recorded message is played. The application then plays a goodbye message and sets up for the next call.

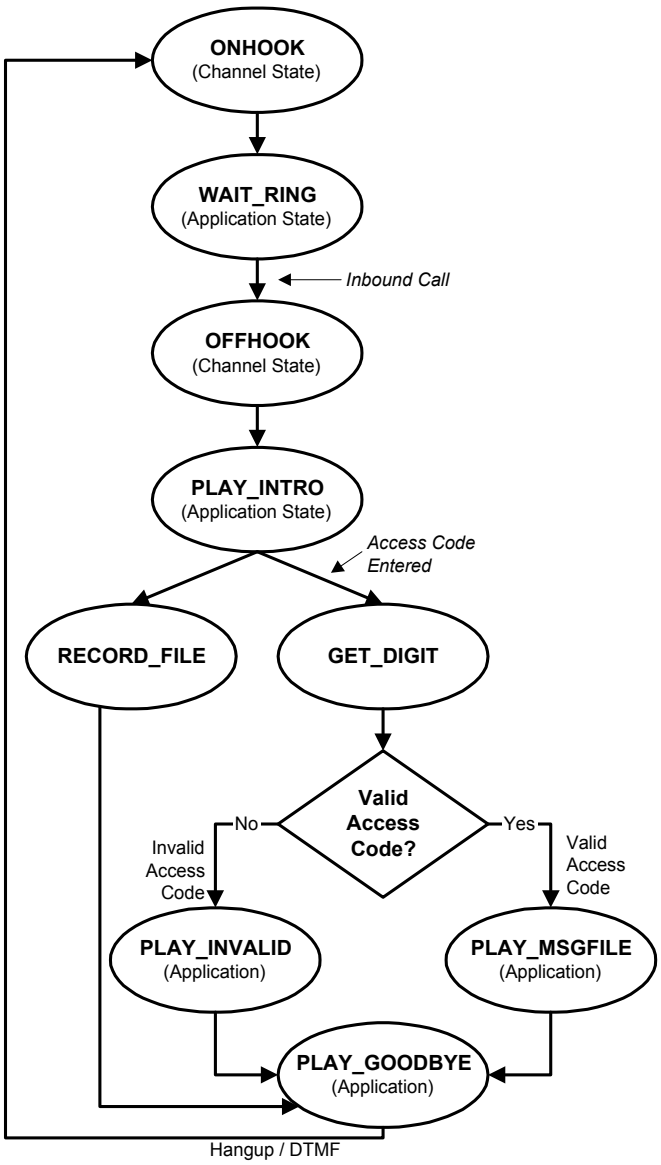

Figure 10. Application and Channel States

Table 15. Demonstration Program States

State	Type	Description
ONHOOK	Channel	Channel is onhook.
WAIT_RING	Application	Application is waiting for an inbound call.
OFFHOOK	Channel	When application detects an inbound call, the channel goes OFFHOOK.
PLAY_INTRO	Application	After receiving an inbound call, application plays an introductory message (i.e., INTRO.VOX file). Channel must be in OFFHOOK state.
GET_DIGIT	Application	If application detects digits being entered while playing the introductory message, it collects and validates the digits.
PLAY_INVALID	Application	Invalid access code was entered and a message (i.e., INVALID.VOX file) is played.
RECORD_FILE	Application	If application does not detect digits being entered after playing the introductory message, a message can be recorded. Channel must be in OFFHOOK state.
PLAY_MSGFILE	Application	If a valid access code was entered after playing the introductory message and GET_DIGIT state, then the previously recorded message is played.
PLAY_GOODBYE	Application	A goodbye message is played (i.e., GOODBYE.VOX file) after one of the following states - PLAY_INVALID, RECORD_FILE, or PLAY_MSGFILE.

10.4. Running the Console Demonstration (SCxDemo) Program

The following procedure describes setting up and running the Console Demonstration (*SCxDemo*) Program. The Console Demonstration Program must be run on the Server node first and then on the Client node.

NOTES: 1. If a clocking event occurs while running the Console Demonstration Program, the program displays the failure and recovery of the SCbus and SCxbus clock (OKAY or FAIL).

NOTES: 2. The only interface type supported is analog. T-1 and E-1 network interfaces are not supported.

1. If not already started, start the Dialogic Service at each node by clicking on the **Set Dialogic Service Startup Mode** icon**.**
2. From the Command Prompt:, enter:

```
scxdemo -n[channel_number] -t[server/client]
 -s[server_name]
```

where:

Option	Description
-n	number of D/4x (i.e., voice) channels to use. The minimum is 1, the maximum is 16, and the default is 4. **NOTE:** The number of channels specified for a client node cannot be greater than the number specified for the server node. If a larger number is specified, the client node will use the number of channels specified for the server.
-t	node type - server or client.
-s	server name - alphanumeric name of the server node; enter at client node only.

For example:

```
scxdemo -n4 -tclient -sbigbox
```

starts the demo program with 4 channels on a client node running an analog front-end and the server name is **bigbox**.

3. A summary screen of the information entered appears on the Server node.

   Select **Y** to confirm the choices.

   Otherwise, select **N** to exit and repeat procedure from step 1.

   When **Y** is selected on the server node, the following message appears:

   ```
   "Waiting for client connection…"
   ```

   The message indicates that the SCxDemo Program can be started on the client node.

   The screen in *Figure 11. Console Demonstration (SCxDemo) Program on Server Node* illustrates starting the Demonstration Program on a server node with five voice channels, using an Analog network interface, on a machine named KEYBOX.

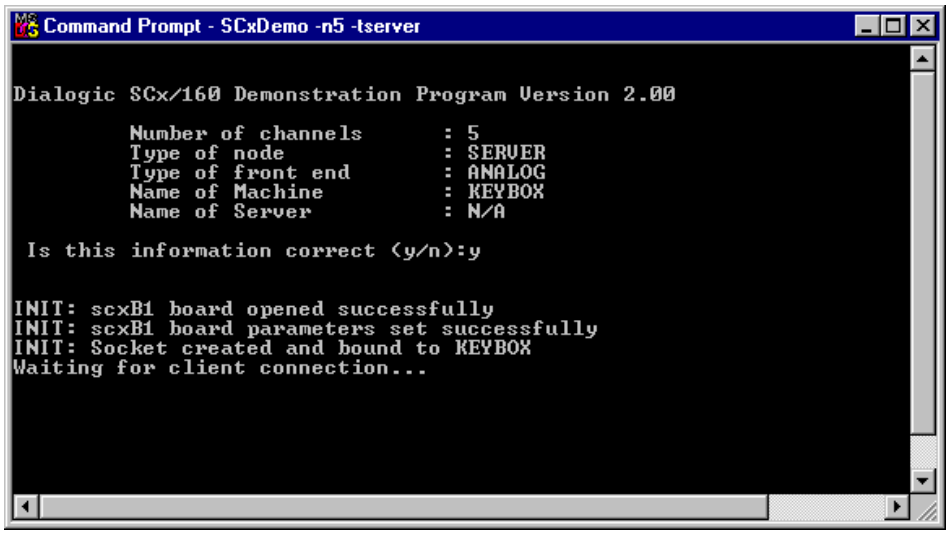

Figure 11. Console Demonstration (SCxDemo) Program on Server Node

4. Start the SCxDemo Program on the client node by repeating steps 1 and 2. The number of channels specified for a client node cannot be greater than the number specified for the server node. If a larger number is specified, the client node will only use the number of channels that was specified for the server. The maximum number of channels that can be specified is 16. The default is 4.

   NOTE: When starting the SCxDemo Program on the client node, ensure that you enter the -s option to identify the server (e.g., **-skeybox**) as illustrated in *Figure 12. Console Demonstration (SCxDemo) Program on Client Node*.

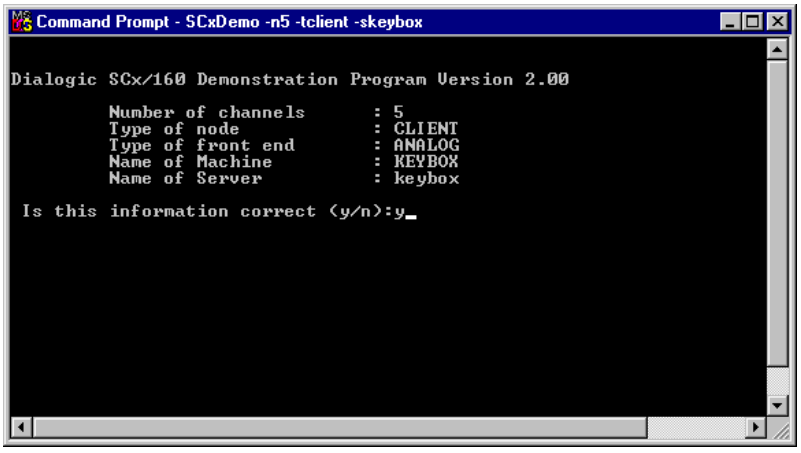

Figure 12. Console Demonstration (SCxDemo) Program on Client Node

After all the channels in the client node are activated, the server node screen displays information such as shown in *Figure 13. Messages Displayed on Server Node*.

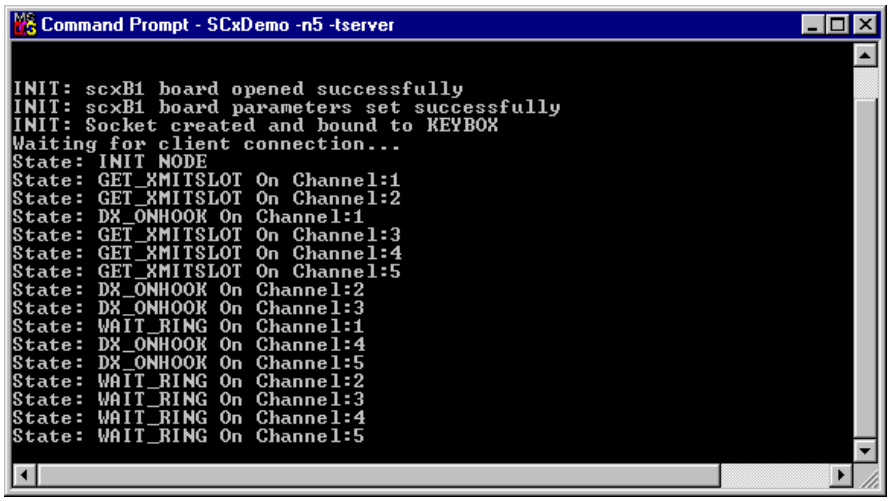

Figure 13. Messages Displayed on Server Node

Information, such as shown in *Figure 14. Messages Displayed on Client Node,* is displayed as each board in the client node is started.

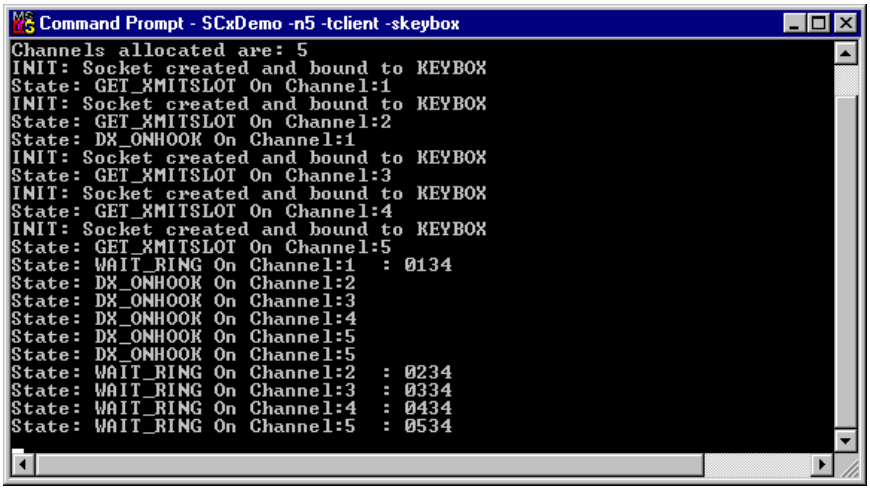

Figure 14. Messages Displayed on Client Node

NOTE: The first two digits (e.g., 01) of the 4-digit number (e.g., 0134) that follows the WAIT_RING message is the channel number. The entire number is also used as the access code that can be entered to playback a recorded message (see Step 6).

After all channels in both nodes are activated, the Console Demonstration (SCxDemo) Program is ready to handle calls.

5. To continue the Console Demonstration (SCxDemo) Program, generate a ring at the client node by making an inbound call.

   When a ring is detected, the client node goes off-hook and notifies the server node to play the INTRO.VOX file in the specified time slot. You will hear the messages on the client node as they are played at the server node, and you can record a message when asked to do so. Both the client and server node display the changes in status when the call is answered, when messages are played and recorded, and when the channel goes on-hook or off-hook.

   For example, *Figure 15. Application Status Messages on Client Node*, illustrates application status changes that occur when a channel receives a call and records a message:

   The same status changes are also displayed for the server node.

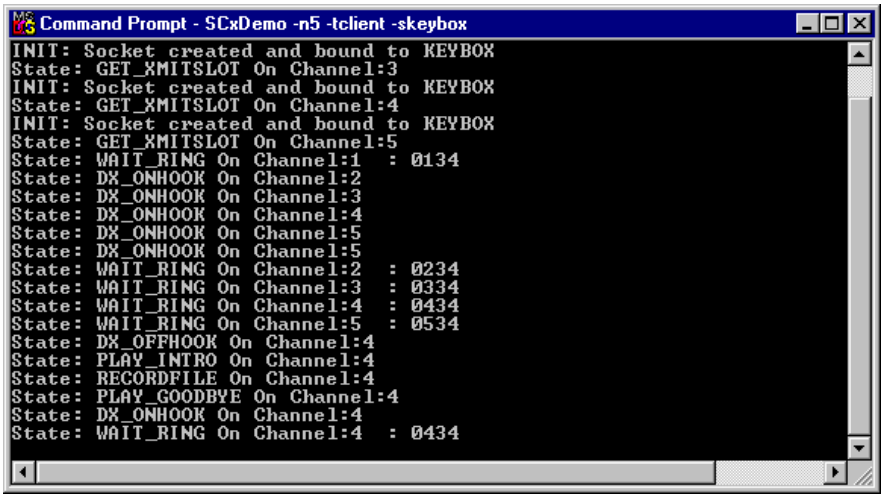

Figure 15. Application Status Messages on Client Node

6. To play a previously recorded message on a given channel:

   - Generate a ring on the same channel at the client node by making an inbound call
   - While the introductory message is played, dial the access code for the given channel (e.g., 0134 for Channel 1)

   The client retrieves the access code digits and sends a socket message to the server node. The server node then plays the recorded message on the given channel. See *Figure 16. Playing Back Recorded Message* for an example of the status changes displayed when playing a recorded message on channel 1.

   NOTE: If an invalid access code was entered, a "PLAY_INVALID" status message is displayed. Otherwise, the server node is instructed to play the recorded message and the status changes are shown on both nodes.

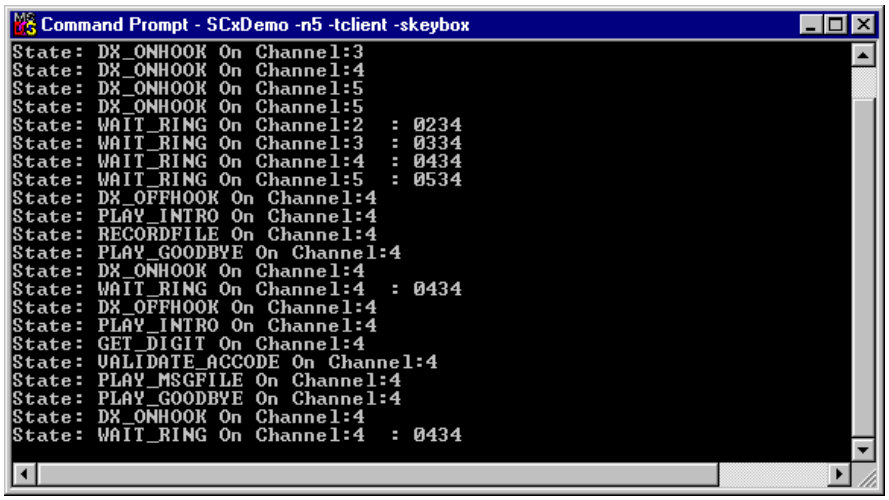

Figure 16. Playing Back Recorded Message

7. To terminate the SCxDemo Program at any node, press the **Ctrl** and **C** keys.

10.5. Running the GUI Demonstration (WinSCxDemo) Program

The following procedure describes setting up and running the GUI Demonstration (WinSCxDemo) Program. The GUI Demonstration (WinSCxDemo) Program needs to be started on the server node first and then on all client nodes.

NOTE: If a clocking event occurs while running the GUI Demonstration (*WinSCxDemo*) Program, the program displays the failure and recovery of the SCbus and SCxbus clock (OKAY or FAIL).

1. If not already started, start the Dialogic Service at each node by clicking on the **Set Dialogic Service Startup Mode** icon.
2. Click on the WinSCxDemo icon in the Dialogic Development Package or the Dialogic System Software Folder/Program Group. The SCX160 Bus Adapter Demo window, as shown in *Figure 17. SCX160 Bus Adapter Demo Window*, is displayed.

   NOTE: Start the GUI Demonstration (*WinSCxDemo*) Program at the Server node first and then at each client node.

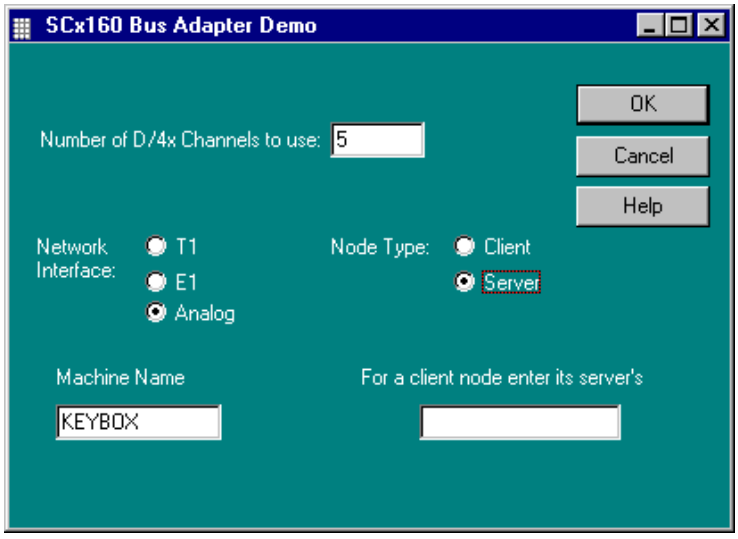

Figure 17. SCX160 Bus Adapter Demo Window

The machine name of this node is automatically displayed in the Machine Name field. Enter the number of D/4x (i.e., voice) channels you want to use. The minimum number is 1, the maximum is 16, and the default is 4.

NOTE: The number of channels specified for a client node cannot be greater than the number specified for the server node. If a larger number is specified, the client node will use the number of channels specified for the server.

Select the network interface type used in your client node at both the server and client(s) nodes.

Select the Node Type: (Client or Server). If this is the server node, leave the "For a client node enter its server's" field blank. If this is a client node, enter the name of the server node.

3. Click the **OK** button. A summary of the information entered is displayed, such as shown in *Figure 18. Starting the Server Node*. At the server node, activate the channels by clicking on the **Start** button shown in *Figure 18*.

NOTE: The server node must be started first.

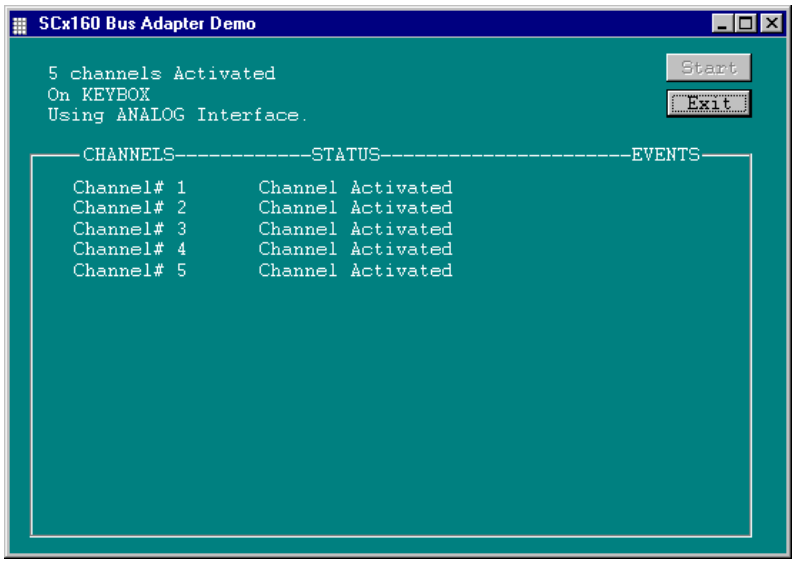

Figure 18. Starting the Server Node

4. Repeat steps 1 - 3 for the client node. At the client node, activate the channels by clicking on the **Start** button. Information, such as shown in *Figure 19. Starting the Client Node,* is displayed as the channels in the node are activated.

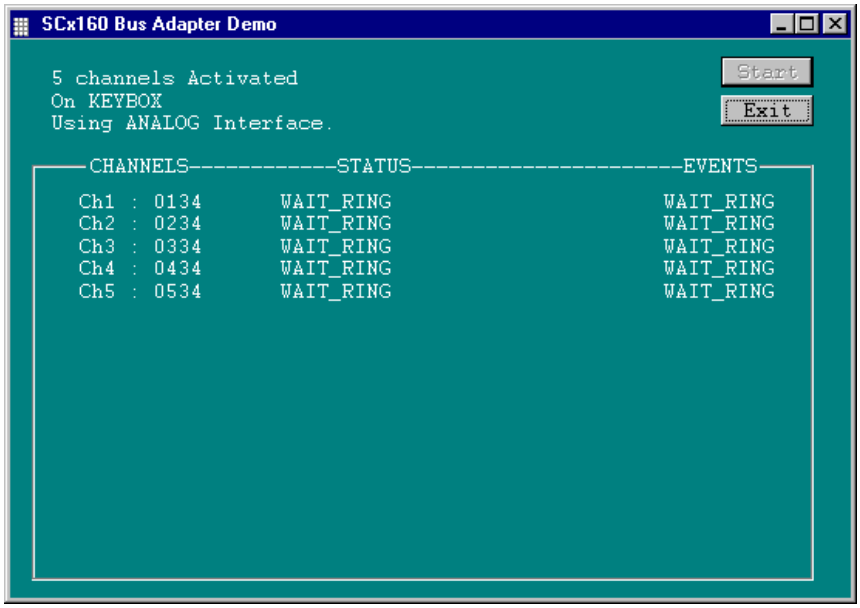

Figure 19. Starting the Client Node

NOTE: The first two digits (e.g., 01) of the 4-digit number (e.g., 0134) appearing before the WAIT_RING message is the channel number. The entire number is also used as the access code that can be entered to play a recorded message (see Step 6).

When all the channels in the client node are activated, the server node screen displays information, such as shown in *Figure 20. Server Node In WAIT_RING State*:

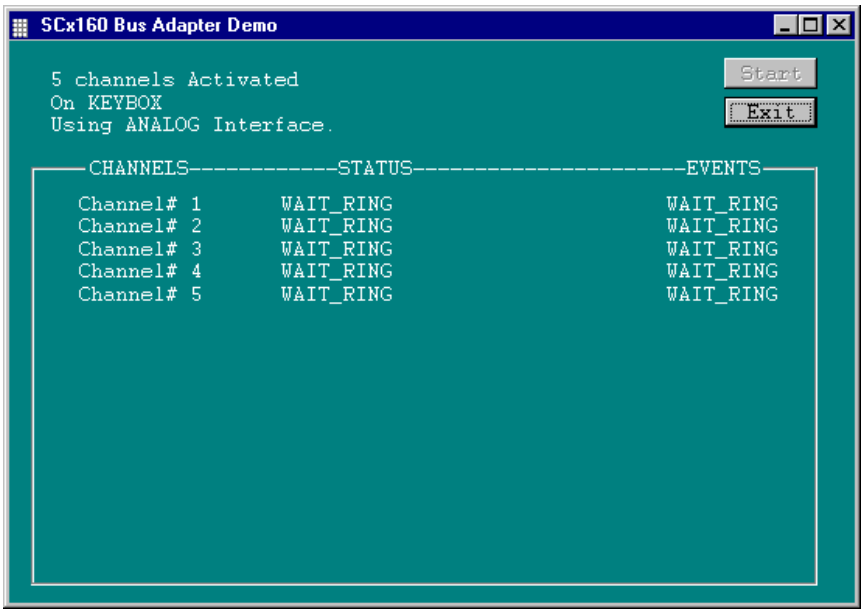

Figure 20. Server Node In WAIT_RING State

After all channels in both nodes are activated, the *WinSCxDemo* Program is ready to handle calls.

5. To continue the WinSCxDemo demonstration, generate a ring at the client node by making an inbound call.

   When a ring is detected, the client node goes off-hook, notifies the server node to play the INTRO.VOX file on the specified channel and displays information such as shown in *Figure 21. Client Node in PLAY_INTRO State*.

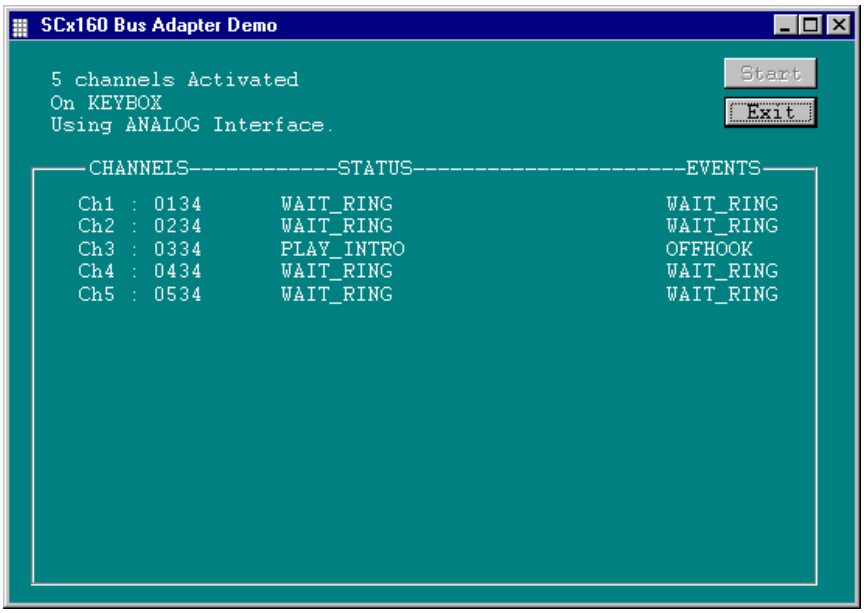

Figure 21. Client Node in PLAY_INTRO State

You will hear the messages on the client node as they are played at the server node, and you can record a message when asked to do so.

Both the client and server nodes display the changes in status when the call is answered, when messages are played and recorded, and when the channel goes on-hook or off-hook.

6. To play back a previously recorded message on a given channel:

   - Generate a ring on the given channel at the client node by making an inbound call
   - While the introductory message plays, dial the access code for the given channel (e.g., 0134 for Channel 1)

   The client node retrieves the access code digits and sends a socket message to the server. The server node then plays the recorded message on the given channel (e.g., channel 1).

NOTE: If an invalid access code was entered, a "PLAY_INVALID" status message is displayed. Otherwise, the server node is instructed to play the recorded message and the status changes are shown on both nodes.

7. To terminate the *WinSCxDemo* Program, click on the **Exit** button.

11. SCX160 Diagnostics

This chapter describes utilities for diagnosing and displaying information for the SCX160 SCxbus Adapter:

- SCX160 InfoTool: displays board parameter information (e.g., clocking, SCbus and SCxbus data streams, etc.) for the SCX160 SCxbus Adapter.

11.1. SCX160 InfoTool

Once a node has been initialized, the SCX160 InfoTool can be used to check how the node is configured and its current status. The SCX160 InfoTool will display the following information at the node whenever the InfoTool is run:

- System level hardware information
- Board level SCbus and SCxbus clock state and mode
- SCX160 SCxbus Adapter clock status
- SCbus and SCxbus data stream assignment

To start the SCX160 InfoTool, click on the **SCX160 InfoTool** icon in the Dialogic Development Package Folder/Program Group. The SCX160 InfoTool window shown in *Figure 22. System Tab* is displayed.

11.1.1. Viewing System Information

The **System** tab displays the hardware interrupt and device name for the SCX160 SCxbus Adapter installed in this node (see *Figure 22. System Tab*). The fields in this tab are described in *Table 16. System Tab Fields*.

Figure 22. System Tab

Table 16. System Tab Fields

Field/Value	Description
Hardware Interrupt	The interrupt assigned to all Dialogic Board Locator Technology (BLT) boards in the node.
SCX160 Device Name	The device name assigned to the SCX160 SCxbus Adapter when it was configured.

11.1.2. Viewing Board Information

The **Board** tab displays the self test results and current clock status (see *Figure 23. Board Tab*). The fields in this tab are described in *Table 17. Board Tab*.

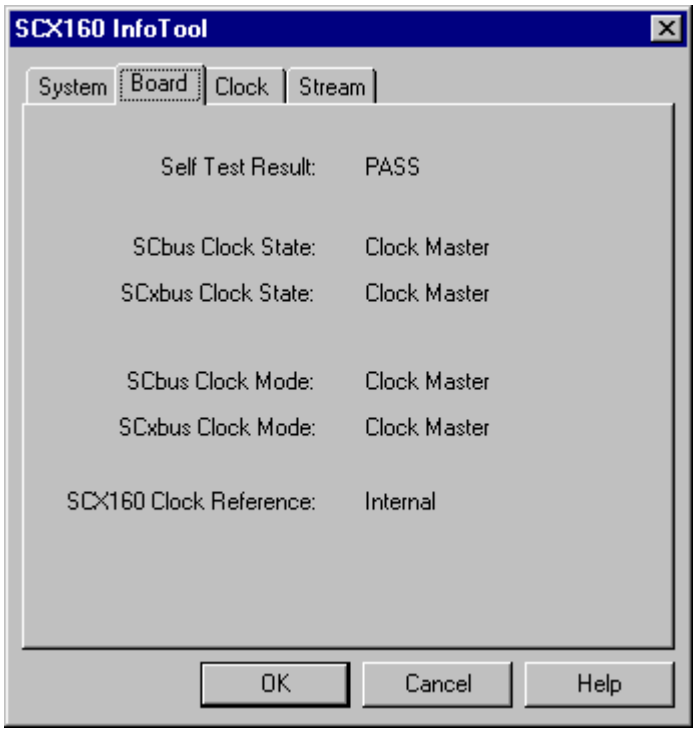

Figure 23. Board Tab

Table 17. Board Tab

Field/Value	Description
Self Test Result	The results of the on-board self test performed on the SCX160 SCxbus Adapter for this node.
Value: • PASS • FAIL	
SCbus Clock State	SCbus real-time clock state for this node. Refer to *Table 11. SCbus Clock States* for a description of all states.
Value: • Master • Fallback Master • Wait Master • Wait Slave • Slave	
SCxbus Clock State	SCxbus real-time clock state for this node. Refer to *Table 9. SCxbus Clock States* for a description of all states.
Value: • Master • Wait Master • Wait Slave • Fallback Master • Slave • Wait Fallback Master • Wait Fallback Slave	
SCbus Clock Mode	SCbus clocking configured at download time for this node. Refer to *Section 4.2. Designating Clock Modes* for a description of all modes.

Field/Value	Description
Value: • Master Clock • Fallback Master Clock • Slave	
SCxbus Clock Mode	SCxbus clocking configured at download time for this node. Refer to *Section 4.2. Designating Clock Modes* for a description of all modes.
Value: • Master Clock • Fallback Master Clock • Slave	
SCX160 Clock Reference	SCX160 clock reference configured at download time for this node. Refer to *Section 4.3. Clock Synchronization* for a description of clock sources.
Value: • Internal - The Local Crystal Oscillator provides the clock. If the node is configured as an SCxbus Slave node and the SCxbus clock fails, then the clock for the node falls back to the Local Crystal Oscillator when the SCX160 clock reference is set to Internal. • SCbus • SCxbus Right • SCxbus Left • External	

11.1.3. Viewing Clock Information

The **Clock** tab displays node clock information (see *Figure 24. Clock Tab*). The fields in this tab are described in *Table 18. Clock Tab*.

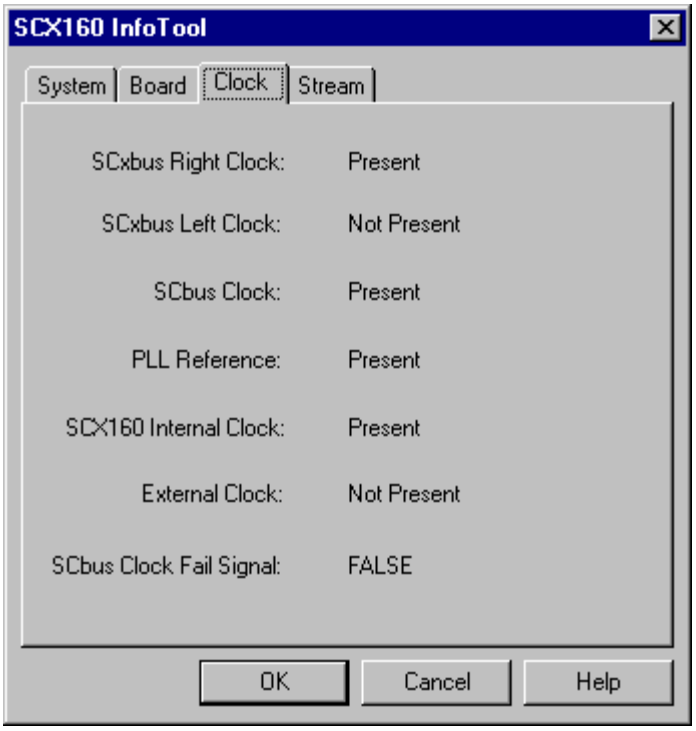

Figure 24. Clock Tab

Table 18. Clock Tab

Field/Value	Description
SCxbus Right Clock	Indicates whether Right SCxbus clock is present in this node. **NOTE:** The Right and/or Left SCxbus clock must be present at each node for proper operation of the multi-node system. Refer to *Chapter 4. Clocking* for more information.
Value: **Present** - Typically, Right SCxbus clock is Present in all nodes in the system. **Not Present** - If Right SCxbus clock is Not Present: • ensure that only one node in the system is configured as the SCxbus Master clock node. • ensure that the SCxbus cabling is properly installed. Refer to *Figure 5. SCxbus Cabling, Typical*	
SCxbus Left Clock	Indicates whether a Left SCxbus clock is present in this node. **NOTE:** The Right and/or Left SCxbus clock must be present at each node for proper operation of the multi-node system. Refer to *Chapter 4. Clocking* for more information.

Field/Value	Description
Value: **Present** - Typically, Left SCxbus clock is Present in all nodes in the system if one of the nodes is designated as the Fallback master Clock node. **Not Present** - if Left SCxbus clock is Not Present and one of the nodes is designated as a Fallback Master Clock node: • ensure that only one node in the system is configured as the SCxbus Fallback Master clock node. • ensure that the SCxbus cabling is properly installed. Refer to *Figure 5. SCxbus Cabling, Typical*.	
SCbus Clock	Indicates the presence of SCbus clock. Refer to *Chapter 4. Clocking* for more information.
Value: **Present** - SCbus clock must be Present in each node for proper operation of the SCbus. **Not Present** - if the SCbus clock is Not Present: • ensure that the designated SCbus clock source is a valid source. • ensure that the SCxbus cabling is properly installed. Refer to *Figure 5. SCxbus Cabling, Typical*	
PLL Reference	Indicates the presence of a valid PLL (Phase Locked Loop) Reference clock reference source. Refer to *Chapter 4. Clocking* for more information.
Value: **Present** - Indicates the presence of a valid clock reference source for the PLL Clock. **Not Present** - Indicates loss of clock synchronization and/or frame synchronization with respect to the reference clock source.	

Field/Value	Description
SCX160 Internal Clock	Indicates presence of the Local Crystal Oscillator clock in this node. Refer to *Chapter 4. Clocking* for more information.
Value: **Present** - Indicates that the Local Crystal Oscillator is generating a clock signal. **Not Present** - Indicates that the Local Crystal Oscillator is no longer generating a clock signal.	
External Clock	Indicates presence of external clock. Refer to *Chapter 4. Clocking* for more information.
Value: **Present** - Indicates external 8KHz clock reference is available at this node. **Not Present** - Typically, this clock is not used as a reference.	
SCbus Clock Fail Signal	Indicates if the SCbus clock has failed.
Value: **FALSE** - A valid SCbus clock reference is present on the board designated as the SCbus master clock source. **TRUE** - The SCbus clock master board no longer has a valid clock reference.	

11.1.4. Viewing Data Stream Information

The **Stream** tab displays information about the SCbus and SCxbus data stream mapping that is configured at download time (see *Figure 25. Data Stream Tab*). The fields in this tab are described in *Table 19. Data Stream Tab*.

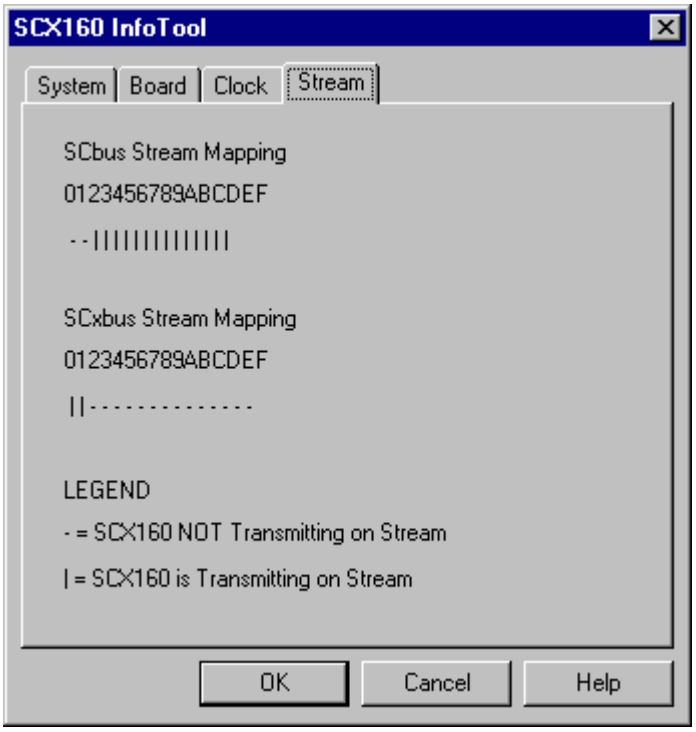

Figure 25. Data Stream Tab

Table 19. Data Stream Tab

Field/Value	Description
SCbus Stream Mapping	Indicates the SCxbus data streams from 0 to F (hex) that the SCX160 SCxbus Adapter transmits onto the SCbus.
SCxbus Stream Mapping	Indicates the SCbus data streams from 0 to F (hex) that the SCX160 SCxbus Adapter transmits onto the SCxbus.

Appendix A - Downloadable Parameters

The following information illustrates the changes that need to be made to the configuration files and parameters to assure that your master clock node is set up properly. The master clock node needs to be automatically setup to provide master clock to the SCxbus network and thus enable data communications between nodes when the system is initialized.

A master clock node must be selected before data can be passed between nodes. For each node, you have the option of selecting whether the clock for a particular board will be either the master clock source or slaved to a clock reference provided on the SCbus or the SCxbus.

The SCX160 SCxbus Adapter clock mode(s) must be determined independently for both the SCbus and the SCxbus interfaces.

Changing Configuration Files and Downloadable Parameters

Each SCX160 SCxbus Adapter must be configured as either a master clock source or as a slave clock node (default) for the SCxbus. The following procedure describes the steps for changing the configuration files to use downloadable parameters.

1. Edit the scxbus.prm file, located in the *%DIALOGICDIR%\data* directory, to uncomment the PARM[253] and the PARM[254] lines and set the PARM[253] value to the desired clock mode. For example, to select the SCxbus Master Clock mode for this node, set:

   PARM[253] = 0010H

   and then uncomment the PARM[254] line and set:

   PARM[254] = 0001H

   NOTE: To select the SCbus as the clock reference for this node; see the Example below.

At all other system nodes, you can use the default parameters established at Dialogic Service startup. The defaults will set the SCX160 SCxbus Adapter to SCxbus clock slave mode with automatic fallback to its internal clock when SCxbus clock is not detected. To change these defaults (e.g., to set up a fallback master node), change the configuration files and parameters at each node in the manner described above for the master clock node (only one node can be the master clock node at a time). See the **scx_setbrdparm( )** function in *Chapter 6. Function Reference* for clock selections and descriptions. Also see *Chapter 4. Clocking* for a description of the various clocking configurations, designations and modes.

Example of the scxbus.prm file

The following is an example of the scxbus.prm file edited to specify that this node be the SCxbus Master Clock node (see the PARM[253] line) and to use the SCbus clock as the clock reference for this node (see the PARM[254] line).

```
AREA=NETWORK
SIZE=WORD
BASE=HEX


; ------------------------------------------------------------------------
; |      Dialogic SCX160 Sample Download Parameter File                  |
; |      Version 1.00-PRODUCTION                                         |
; |      Copyright 1996 Dialogic Corp.                                   |
; |      All Rights Reserved                                             |
; ------------------------------------------------------------------------
;
; All SCX160 parameter values must be specified as word values.
;
;---
;--- NOTE:  The SCbus Clock Mode of the SCX160 board cannot be configured
;---        through download parameters.  Use the download configuration
;---        file to specify the SCbus Clock Mode of the SCX160 Board.
;---
;--- SCxbus Clock Mode  (Parameter 253)
;--- Defines what clock mode to set the SCxbus Interface of the SCX160
;--- SCxbus Adapter to.
;--- Valid Modes that may be specified are as follows:
;---    0010H = SCxbus Clock Master Mode
;---    0020H = SCxbus Clock Slave Mode
;---    0030H = SCxbus Fallback Clock Master Mode
;--- The default SCxbus Clock Mode is:
;---    0020H (SCxbus Clock Slave Mode).
PARM[253] =0010H
;---
;--- SCX160 Clock Reference (Parameter 254)
;--- Defines what reference should be used for the SCX160 PLL.
;--- Valid Clock References that may be specified are as follows:
;---    0000H = Use Internal Clock
;---    0001H = Derive Clock From SCbus
;---    0002H = Derive Clock From SCxbus Right Clock
;---    0004H = Use External Clock
;--- The default SCX160 Clock Reference is:
;---    0000 (Use Internal Clock).
PARM[254] =0001H
```

Appendix B - Dialogic SCX160 Configuration Program

Step-by-step procedures for running the Dialogic SCX160 Configuration Program are presented in this appendix. The Dialogic SCX160 Configuration Program is used to select the type of configuration, i.e., Non-Blocking or Blocking. The program is used to determine the number of data streams that need to be allocated to the node, and it is used to select the SCbus data streams that will be blocked from the SCxbus in the case of a Blocking Configuration. For a description and an example of SCbus time slot blocking, refer to *Section 2.3. Blocking Data Streams*.

The following considerations apply to blocking:

- Only the time slots for one resource (e.g., voice, analog or digital) can be blocked per node.
- When blocking time slots for a resource in a node, the number of data streams blocked must include all the time slots being used for that resource (e.g., if 5 D/240SC-T1 boards reside in Node 1 and voice resources are blocked, then two data streams (128 time slots) must be blocked from the SCxbus because the voice resources on these boards require 120 voice time slots).
- The time slots for a different resource can be blocked in each node (e.g., Node 1 can block time slots for the voice resource and Node 2 can block time slots for the digital resource).
- The number of data streams blocked from the SCxbus must be the same for each node in the system. This number must be equal to or greater than the largest number of data streams to be blocked in any node.

NOTE: The number of time slots used by a resource in a given node can be calculated from *Table 20. Time Slots Required Per Board* and the configuration information entered on the Dialogic Node Configuration Chart in the *SCX160 SCxbus Adapter Package Release Notes* when the SCX160 SCxbus Adapter was installed.

Prerequisites

For each node in the multi-node system:

- The hardware (SCSA boards and SCX160 SCxbus Adapter) must be installed. Refer to the *Quick Install for the SCX160 SCxbus Adapter Card* for details.
- The software (Dialogic System Software and SDK for Windows DNA Version 2.0, and the SCxbus Adapter Package) must be installed. Refer to the *SCX160 SCxbus Adapter Package Release Notes* for details.
- Determine the maximum number of data streams to be blocked for any one node in the entire multi-node system. See the next section, *Determining Number of Blocked Data Streams*.
- After ensuring that all of the above prerequisites are met, run the Dialogic SCX160 Configuration Program as described in the *Running the Dialogic SCX160 Configuration Program* paragraph in this appendix.

Determining Number of Blocked Data Streams

Before running the Dialogic SCX160 Configuration Program, determine the number of data streams to be blocked in the multi-node system. To determine this number, enter the following information into *Table 21. Dialogic Blocking Configuration Chart* and make the indicated calculations:

- Node #
- Resource to be blocked in this node
- Boards installed in this node
- Total number of time slots to be blocked in this node. Determine this number as follows:

    1. Locate the model number of an installed board in *Table 20. Time Slots Required Per Board* and record the *Time Slots Required* quantity for the resource to be blocked in the *Time Slots Blocked* column of *Table 21. Dialogic Blocking Configuration Chart*. See *Table 22. Example Blocking Configuration Chart* for a sample chart.
    2. Repeat step 1 for each board installed in this node.
    3. (optional) Add an unused/reserved-for-future-expansion entry and a corresponding number of time slots to be blocked. Reserving additional time slots may enable future expansion without reconfiguring the system.
    4. Add all the *Time Slots Blocked* entries for this node to get the sum.
       Divide this sum by 64 to get a quotient.
       Round up this quotient to the next highest integer.
       Enter this integer in the *Data Streams Blocked* column as the total number of data streams that must be blocked in this node.
    5. Repeat procedure from Step 1 for each node.

- Select the largest numerical value listed in the *Data Streams Blocked* column and use/enter this value as the *Number of data streams to be blocked in EACH node*.

Table 20. Time Slots Required Per Board

Board	Resource(s)	Time Slots Required
D/41ESC	Voice Analog	4 4
D/80SC	Voice	8
D/80SC-4LS	Voice Analog	8 8
D/160SC	Voice	16
D/160SC-LS	Voice Analog	16 16
D/240SC	Voice	24
D/240SC-T1	Voice Digital	24 24
D/240SC-2T1	Voice Digital	24 48
D/480SC-2T1	Voice Digital	48 48
D/300SC-E1	Voice Digital	30 30
D/300SC-2E1	Voice Digital	30 60
D/320SC	Voice	32
D/600SC-2E1	Voice Digital	60 60
DCB/320SC	Digital	32
DCB/640SC	Digital	64

Board	Resource(s)	Time Slots Required
DCB/960SC	Digital	96
DTI/240SC	Digital	24
DTI/241SC	Voice Digital	24 24
DTI/300SC	Digital	30
DTI/301SC	Voice Digital	30 30
LSI/81SC	Analog Voice	8 8
LSI/161SC	Analog Voice	16 16
MSI/80SC	Digital	40
MSI/160SC	Digital	48
MSI/240SC	Digital	56
VFX/40SC	Voice/Fax Analog	4 4
VFX/40ESC	Voice/Fax Analog	4 4
VFX/40ESCplus	Voice/Fax Analog	4 4

Table 21. Dialogic Blocking Configuration Chart

Node #	Resource Blocked	Boards Installed	Time Slots Blocked	Data Streams Blocked (Time Slots/64)

Node #	Resource Blocked	Boards Installed	Time Slots Blocked	Data Streams Blocked (Time Slots/64)
Number of data streams to be blocked in EACH node:				

Table 22. Example Blocking Configuration Chart

Node #	Resource Blocked	Boards Installed	Time Slots Blocked	Data Streams Blocked (Time Slots/64)
1	Voice	DTI/240SC	0	
		D/240SC-T1	24	
		D/240SC-T1	24	
		D/240SC-T1	24	
		D/240SC-T1	24	
		D/240SC-T1	24	
		Total:	120	2 (120/64 = 1.9)
2	Voice	7 DTI/241SC	24 each	
		Total (7 x 24):	168	3 (168/64 = 2.6)
Number of data streams to be blocked in EACH node:				**3**

Running the Dialogic SCX160 Configuration Program

Step-by-step procedures for running the Dialogic SCX160 Configuration Programs are included in this appendix.

Configuring Node 1 For The First Time

1. If the Dialogic Service is running, stop the service and exit from the Dialogic Configuration Manager (DCM) before starting the SCX160 Configuration Program.
2. Start the Dialogic Configuration Manager (DCM) by clicking on the **Start** button (located at the bottom left of the screen) and choose:

   - **Programs**
   - **Dialogic System Software**
   - **Dialogic Configuration Manager**

   When the DCM starts, the program detects the presence of an SCX160 board in the system and displays a dialog box, giving the user a chance to configure the SCX160 board. *Figure 26. Configuration Information Window* displays.

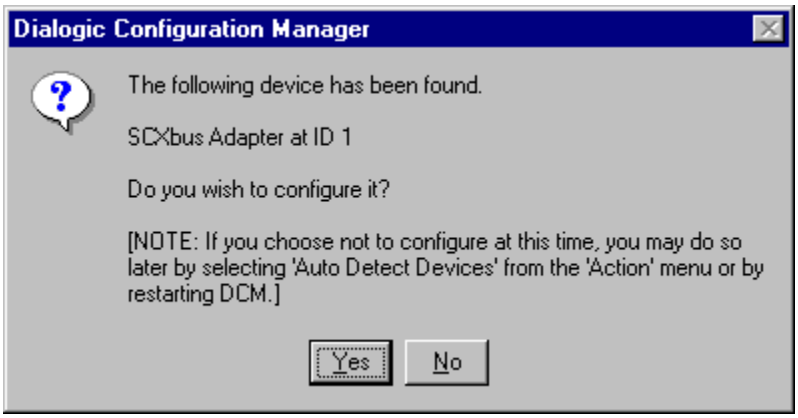

Figure 26. Configuration Information Window

3. Click on **Yes** to configure the device. The DCM initiates the SCX160 Configuration Program.

4. *Figure 27. Map File Information Window - Global Map File Flag* displays. Select either LOCAL or GLOBAL for the *MapFileFlag* and then enter the *MapFilePath* (the path of an existing directory) in the space provided. This is the directory where the MapFile, *scxmap.dat*, will be created.

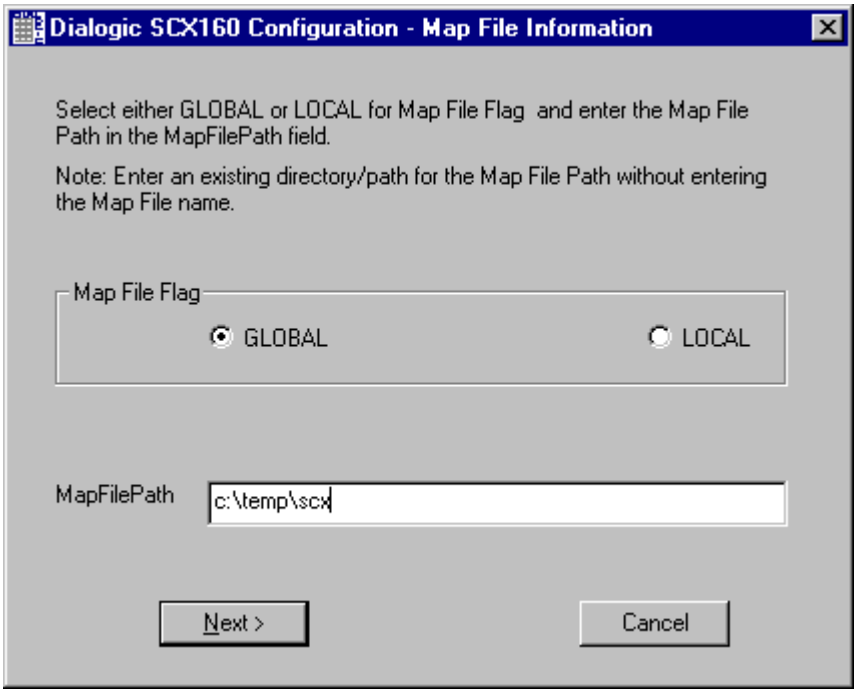

Figure 27. Map File Information Window - Global Map File Flag

5. Click on **Next>** to proceed to the next screen.

   If **GLOBAL** was chosen for the MapFileFlag, continue with *Step 6*. For **LOCAL**, skip to *Step 8*.

6. **For Global MapFile only:**

   Enter a name for Node 1 (i.e., NodeOne) and then enter an existing directory/path for a backup MapFile. This directory will have a copy of the MapFile and will be used in case the original MapFile is not accessible (*Figure 28*).

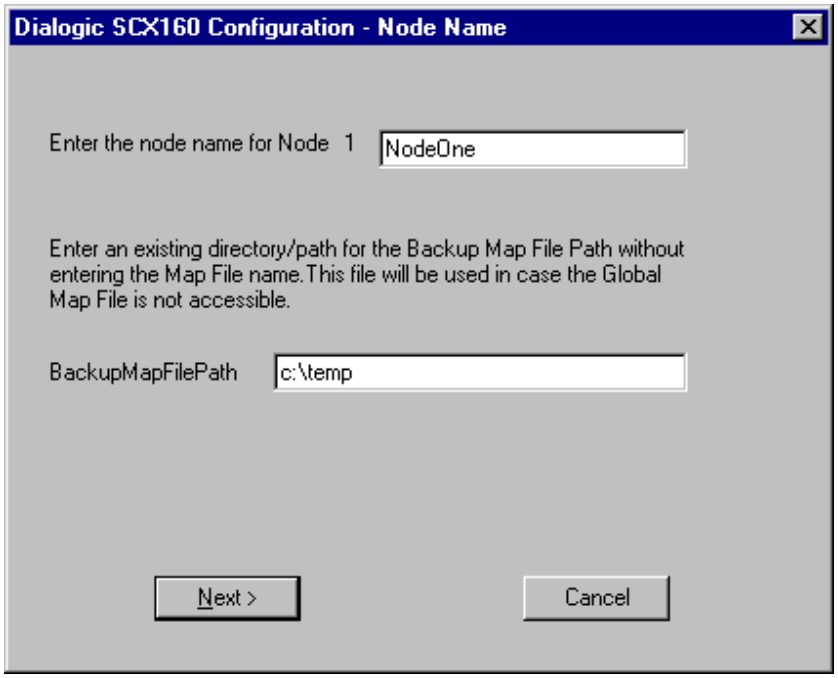

Figure 28. Node Name Window (Global MapFileFlag)

7. After the information has been entered, click on **Next>** to continue. Skip to *Step 12*.

8. **For Local MapFile only:**

   If **LOCAL** was chosen for the MapFileFlag, *Figure 29. Scxconfig Window* displays and asks the user if this is the first node being installed.

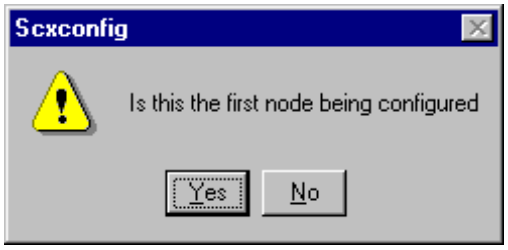

Figure 29. Scxconfig Window

9. Click on **Yes** since this is the first node being installed.
10. Enter the name of the node in the space provided (see *Figure 30. Node Name Window (Local MapFileFlag)*).

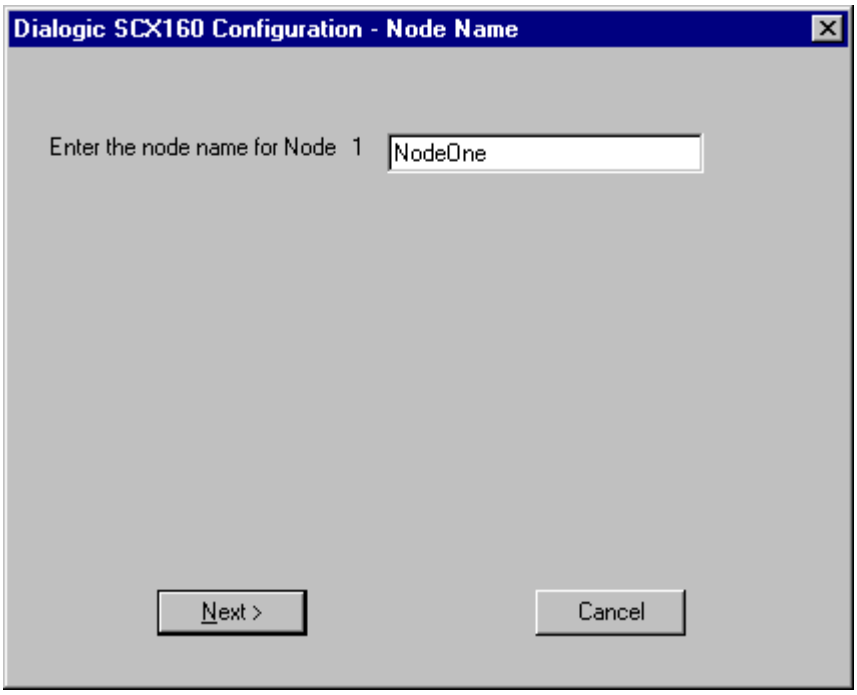

Figure 30. Node Name Window (Local MapFileFlag)

11. Click on **Next>** to continue. This opens *Figure 31. Configuration Type Window*.

12. *Figure 31. Configuration Type Window* prompts the user to select either the Non-Blocking (Standard) Configuration or the Blocking Configuration. For the Standard Configuration, continue with *Step 13*. Otherwise, skip to *Step 17,* Blocking Configuration.

    NOTE: Refer to *Section 2.3. Blocking Data Streams* for more information on Blocking Configurations.

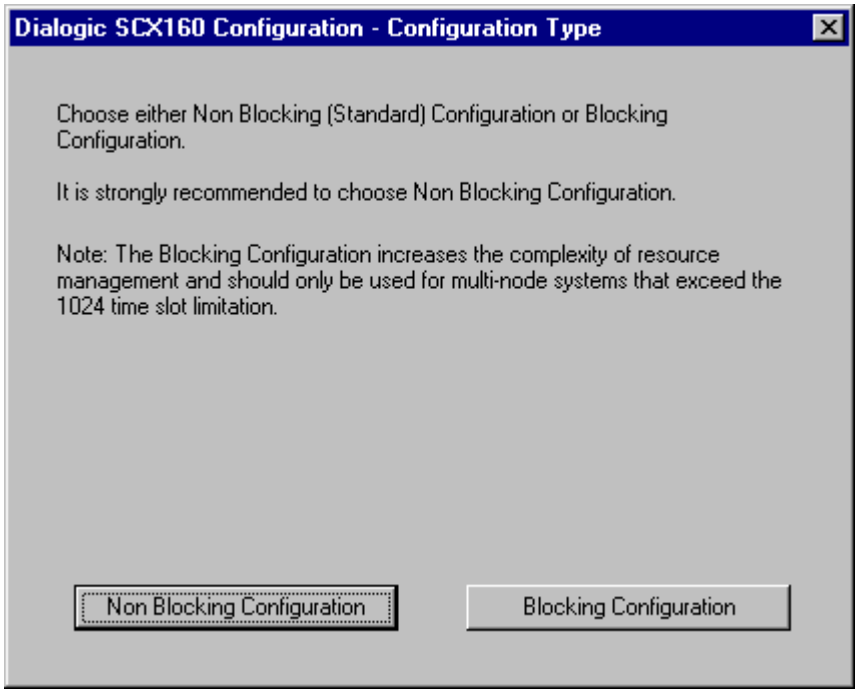

Figure 31. Configuration Type Window

13. **Non-Blocking Configuration:**

    Enter the number of data streams required to support the time slots used by the currently installed boards (*Figure 32. Streams Information Window*).

The system responds with information regarding the number of time slots in the node and the minimum number of data streams required.

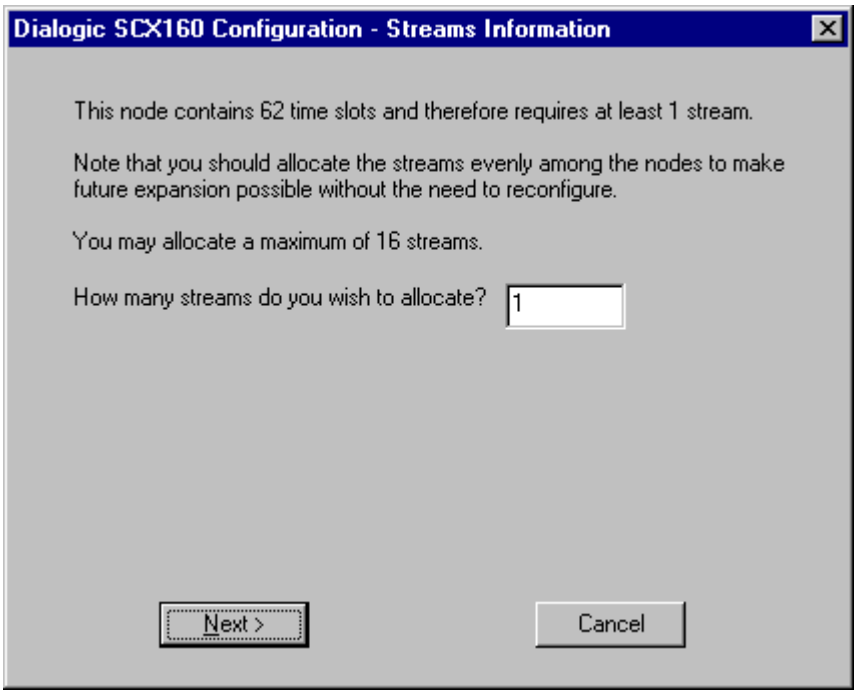

Figure 32. Streams Information Window

NOTE: It is possible to allocate more SCbus time slots than are required for the currently installed boards for future use. If additional time slots are reserved, additional boards may be added to the nodes without reconfiguring the system. Once reserved SCbus time slots have been utilized, the system would need to be reconfigured when adding additional boards.

14. Click on the **Next>** button to proceed. This opens the *Summary Window (Non-Blocking Configuration)* window (*Figure 33*).

15. The configuration values listed in *Figure 33. Summary Window (Non-Blocking Configuration)* are:

    - *Data Streams Transmitted from the SCX160 board onto the SCbus* : Indicates that the SCxbus data streams (between 0 and 15) on the SCX160 SCxbus Adapter transmit onto the SCbus.
    - *Data Streams Transmitted from the SCX160 board onto the SCxbus* : Indicates the SCbus data streams (between 0 and 15) on the SCX160 SCxbus Adapter transmit onto the SCxbus.

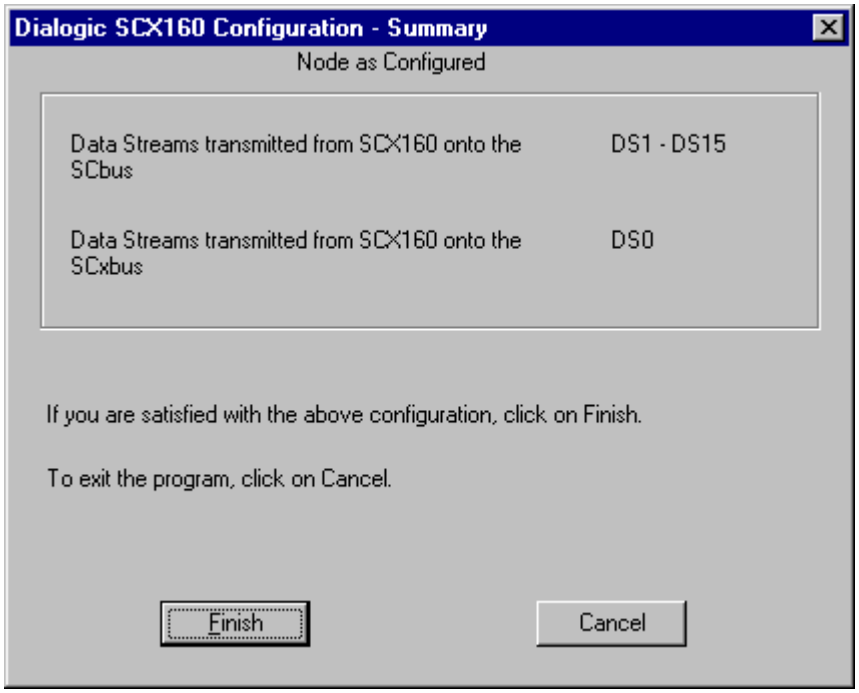

Figure 33. Summary Window (Non-Blocking Configuration)

16. Review the summary information displayed (*Figure 33. Summary Window (Non-Blocking Configuration)*) and either:

    - Click the **Finish** button to save the selections, exit the program and return to the Dialogic Configuration Manager Window, or;

- Click the **Cancel** button to discard the selections and exit the program.

Go to *Step 25*.

17. **Blocking Configuration:**

    At the *Configuration Type Window*, *Figure 31*, select **Blocking Configuration** and *Figure 34. Blocked Streams Information Window* displays:

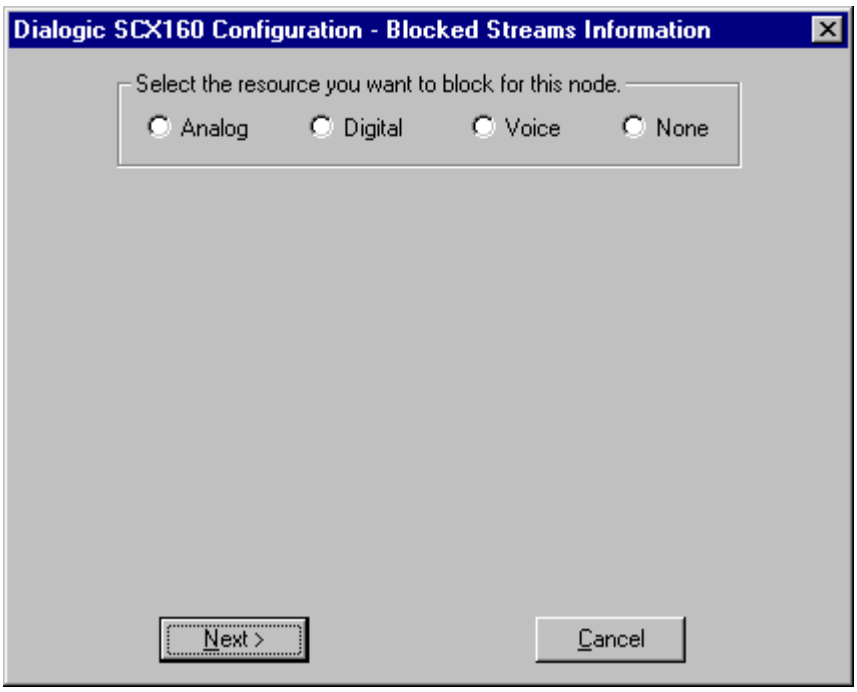

Figure 34. Blocked Streams Information Window

18. Select one resource to block at this node (i.e., digital) and click on **Next>** to continue.

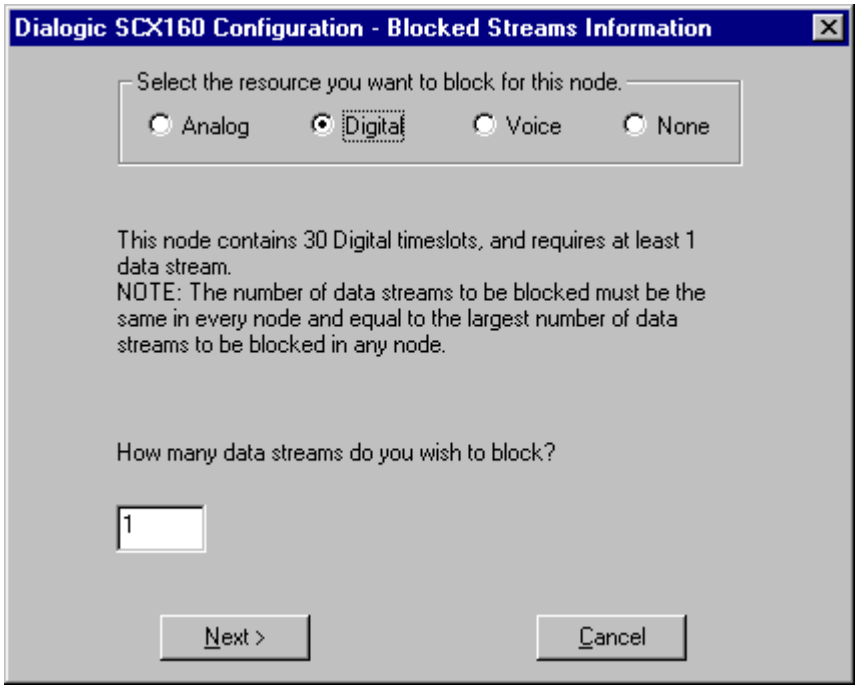

Figure 35. Number of Blocked Streams Window

19. The number of data streams required to block the selected resource is automatically calculated (*Figure 35*). A larger number of data streams must be blocked if this number is not the largest number of data streams that will be blocked at another node (see *Section 2.3. Blocking Data Streams*).

    NOTE: The number of data streams to be blocked can only be entered when configuring Node 1. Blocked resources will only be available to resources located in this node and are not shared across the SCxbus.

    If you do not wish to block a resource on a particular node, select **None** (*Figure 35*).

20. Since this is the first node configured, enter the largest number of data streams to be blocked in any node in the system (see *Section 2.3. Blocking Data Streams*).

21. Click the **Next>** button to continue. This opens *Figure 36. Streams Information Window (Blocked Configuration)*.

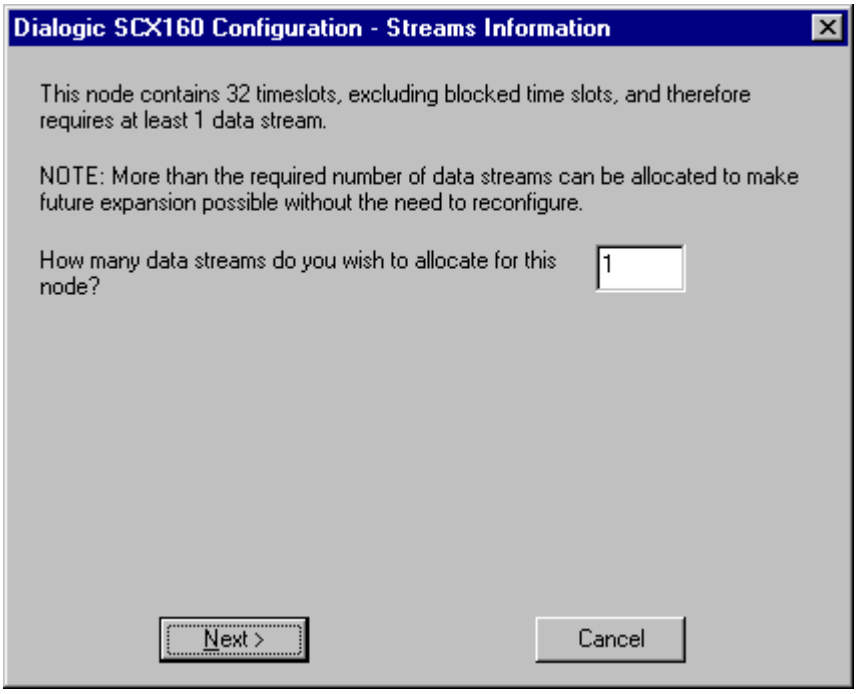

Figure 36. Streams Information Window (Blocked Configuration)

22. In *Figure 36*, the number of data streams required to support the time slots used by the non-blocked resources installed is automatically calculated. More data streams than are required for the currently installed resources can be entered to reserve data streams for future expansion (the addition of new resources). Reserving additional data streams will enable future expansion without reconfiguring the system.

    Enter the number of data streams required to support the time slots used by the non-blocked resources.

    Click on the **Next>** button to proceed.

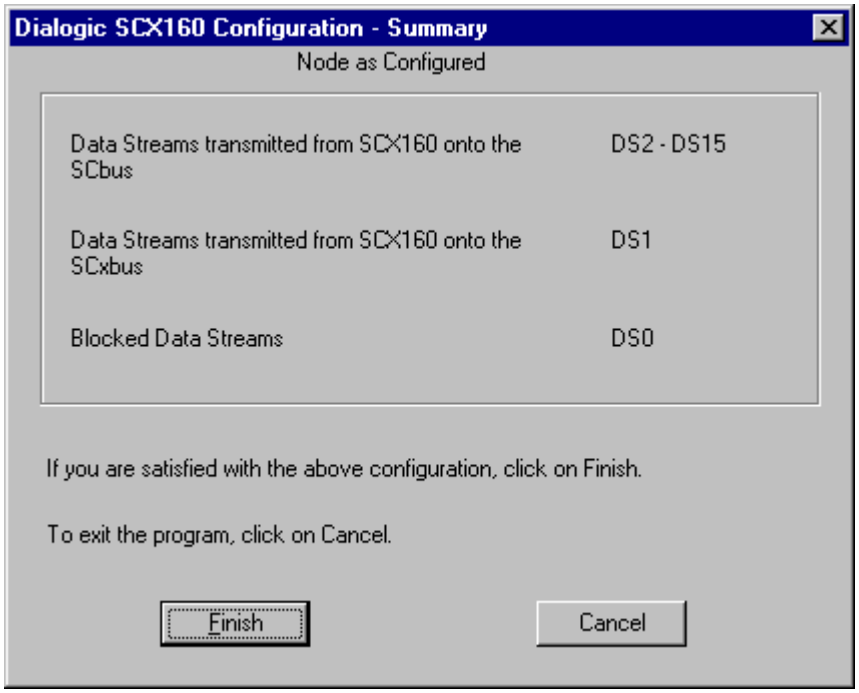

Figure 37. Summary Window (Blocked Configuration)

23. The configuration values listed in *Figure 37. Summary Window (Blocked Configuration)* are:

    - *Data Streams Transmitted from the SCX160 board onto the SCbus* : Indicates the SCxbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCbus.
    - *Data Streams Transmitted from the SCX160 board onto the SCxbus* : Indicates the SCbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCxbus.
    - *Blocked Data Streams*: Indicates the data streams that are blocked from transmitting onto the SCxbus.

24. Review the summary information displayed and either:

- Click the **Finish** button to save the selections, exit the program and return to the Dialogic Configuration Manager Window, or;
- Click the **Cancel** button to discard the selections and exit the program.

25. The program now returns to the Dialogic Configuration Manager window (see *Figure 38*). Double-click on the SCxbus Adapter and the Properties for SCxbus Adapter screen displays (*Figure 39. Properties for SCxbus Adapter screen*).

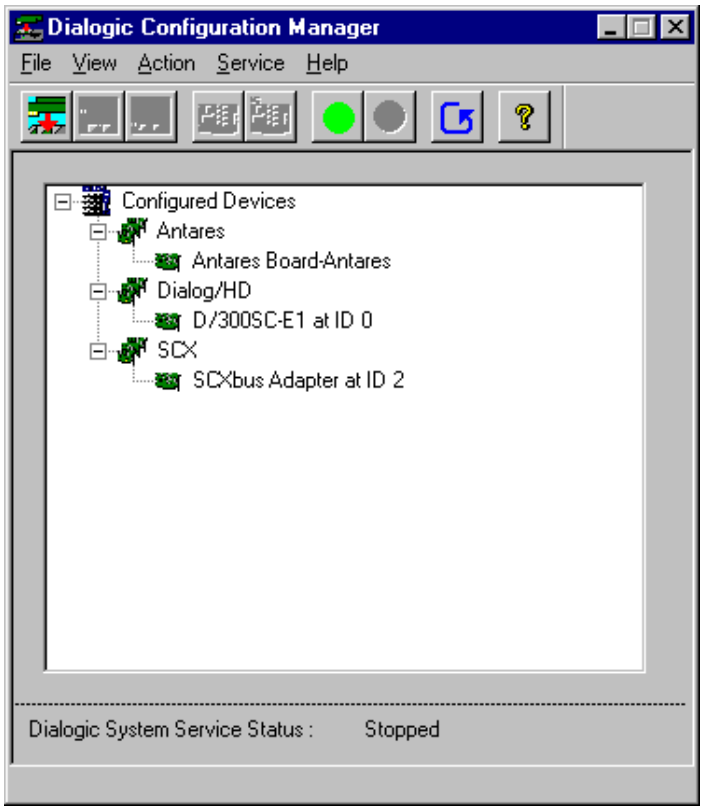

Figure 38. Dialogic Configuration Manager Window

26. Click on the SCxbus tab to display the current SCxClock settings (*Figure 39*).

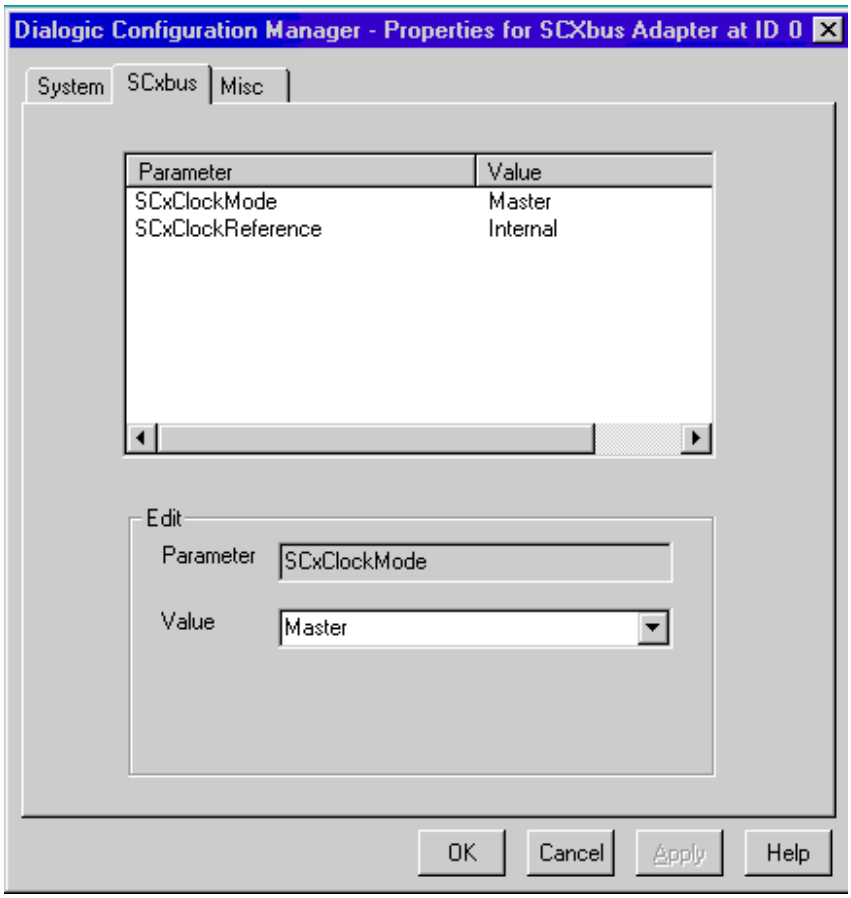

Figure 39. Properties for SCxbus Adapter screen

NOTE: When configuring a multi-node system, at least one node must be set to Master. The default value is Slave mode.

27. Change the value to Master, then Apply to save the changes. If the parameters and values are correct, click **OK**.

28. The system returns to the Dialogic Configuration Manager window. Start the Dialogic Service by clicking on the **Green** button at the top of the screen (*Figure 38*).

    NOTE: If at any time during the SCX160 Configuration, **Cancel** was selected to exit out of the program and you wish to return to the SCX160 Configuration Program, do the following:

    - Exit out of the Dialogic Configuration Manager by clicking on **File** and selecting **Exit**.
    - Start the Dialogic Configuration Manager. There will be an option to configure the SCX160. Choose **Yes** at this point to enter the SCX160 Configuration.

      or

    - Choose **Auto Detect Devices** from the action drop down menu in the Dialogic Configuration Manager (*Figure 38*).
    - There will be an option to configure the SCX160 board. Choose **Yes** to enter the SCX160 Configuration Program.

Configuring Node 2 or a Higher Node For The First Time

For a LOCAL map file, the *scxmap.dat* file must be copied from the last node configured to this node **before** starting the Dialogic Configuration Manager. Otherwise, the Dialogic SCX160 Configuration Program will prompt you to insert the disk that contains the *scxmap.dat* file.

1. Start the Dialogic Configuration Manager (DCM) by clicking on the **Start** button (located at the bottom left of the screen) and choose:

   - **Programs**
   - **Dialogic System Software**
   - **Dialogic Configuration Manager**

   When the DCM starts, it detects the presence of an SCX160 board in the system and displays a dialog box, giving the user a chance to configure the SCX160 board (see *Figure 40*).

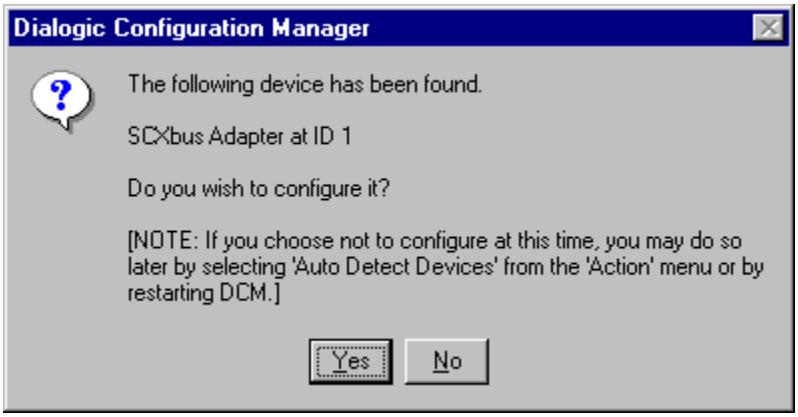

Figure 40. DCM Configuration Window

2. Click on **Yes** to start the Dialogic SCX160 Configuration Program.
3. Select either **GLOBAL** or **LOCAL** for the *MapFileFlag* and enter the *MapFilePath* in the space provided (*Figure 41. Map File Information Window*).

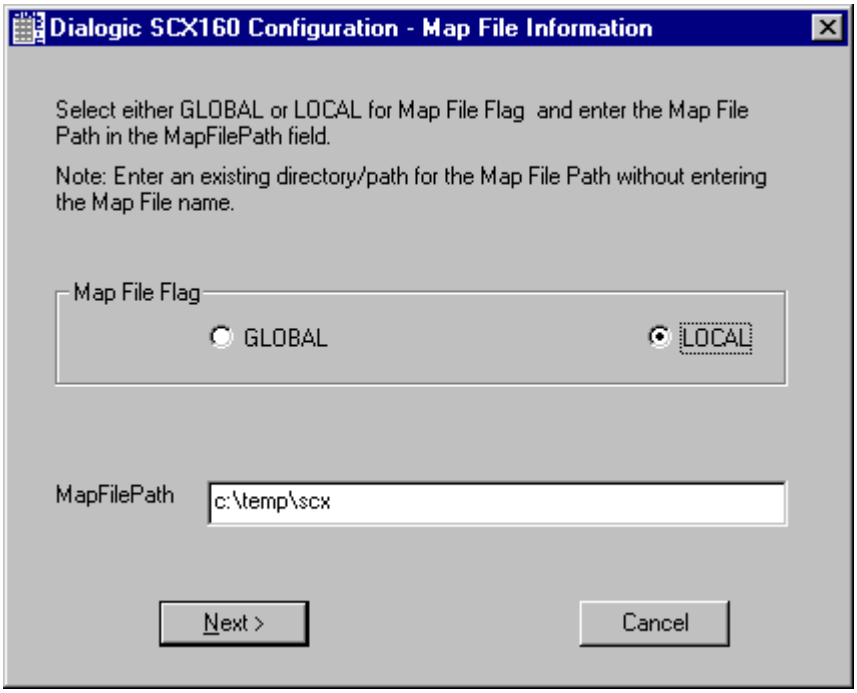

Figure 41. Map File Information Window

For a LOCAL MapFileFlag, enter the path of an existing directory for *MapFilePath*. This is the directory where the *scxmap.dat* MapFile will be placed. If the *scxmap.dat* file was copied from the previous node, enter that path.

For a GLOBAL MapFileFlag, enter the directory path for the *scxmap.dat* file that was created while configuring Node 1.

NOTE: You must enter the same MapFileFlag that was entered when configuring Node 1.

4. Click on **Next>** to proceed.

   If **GLOBAL** was selected for MapFileFlag, skip to *Step 11*. For a **LOCAL** MapFileFlag, continue with *Step 5*.

5. **Local MapFileFlag:**

   If the *scxmap.dat* file was copied from the previous node to this node before starting the DCM, skip to *Step 10*.

   Otherwise, *Figure 42. Scxconfig Window* asks the user if this is the first node being installed. Since this is not the first node being installed, click on **No**.

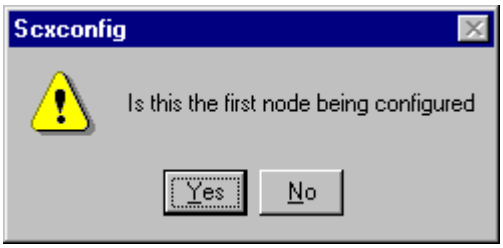

Figure 42. Scxconfig Window

6. When **No** is selected in *Figure 42*, the dialog box shown in *Figure 43. Copy MapFile Window* is displayed. If you wish to copy the MapFile now, click **Yes** to continue.

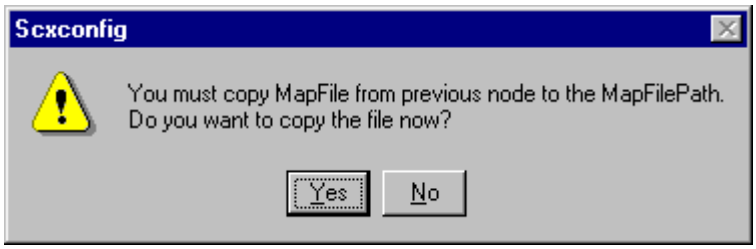

Figure 43. Copy MapFile Window

7. In *Figure 44. Insert Map File Disk Window,* the DCM prompts you to insert the disk containing the Map File. Insert the disk containing the *scxmap.dat* file and click **OK** to proceed to the next screen.

Figure 44. Insert Map File Disk Window

8. *Figure 45. Enter MapFile Path Window* opens and the user is prompted to enter the entire path of the MapFile, including the disk drive letter of where to copy the file from (i.e., **a:\scx\scxmap.dat**).

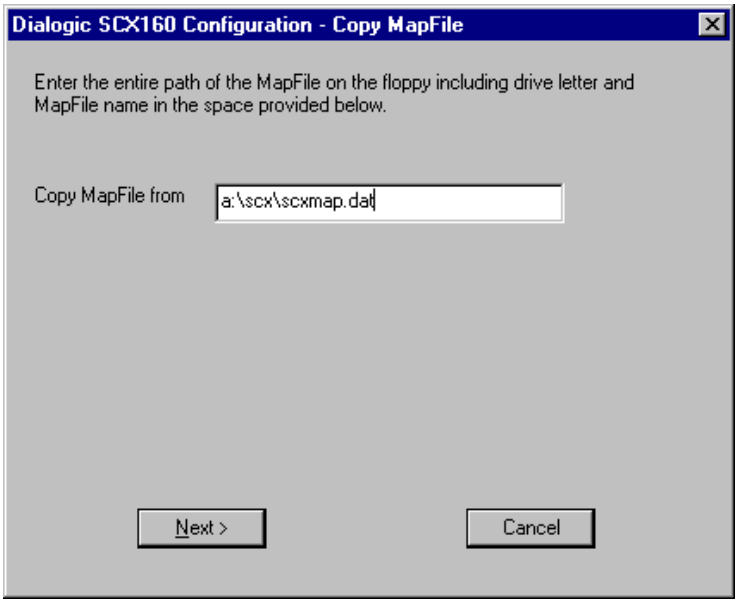

Figure 45. Enter MapFile Path Window

9. Click on **Next>** to proceed to the next screen.

10. Enter the name of the node in the space provided (see *Figure 46*).

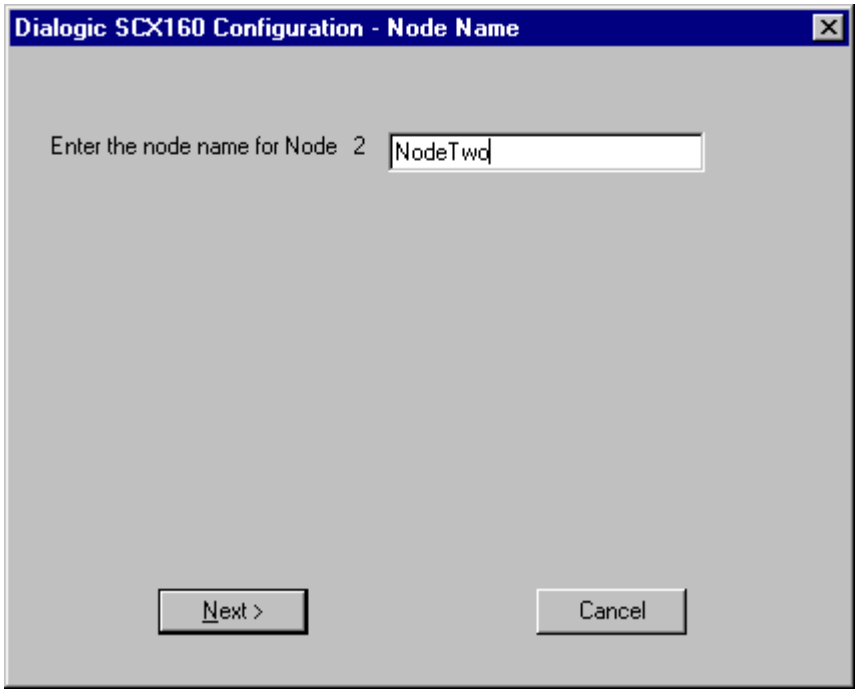

Figure 46. Node Name Window

Click on **Next>** to continue and go to *Step 12*.

11. Enter the name of the Node and enter an existing directory/path for a backup MapFile. This directory will have a copy of the MapFile and will be used in case the original MapFile is not accessible. See *Figure 41* for an example of the MapFile Information Window.

    Click on **Next>** to continue.

12. If you chose Non-Blocking (Standard) Configuration while configuring Node 1, proceed with *Step 13*. If the **Blocking Configuration** was selected, go to *Step 17*. See *Running the Dialogic SCX160 Configuration Program* , *Configuring Node 1 For The First Time*, *Step 12*.

13. *Figure 47. Streams Information Window* displays and prompts the user to enter the number of data streams required to support the time slots used by the currently installed boards.

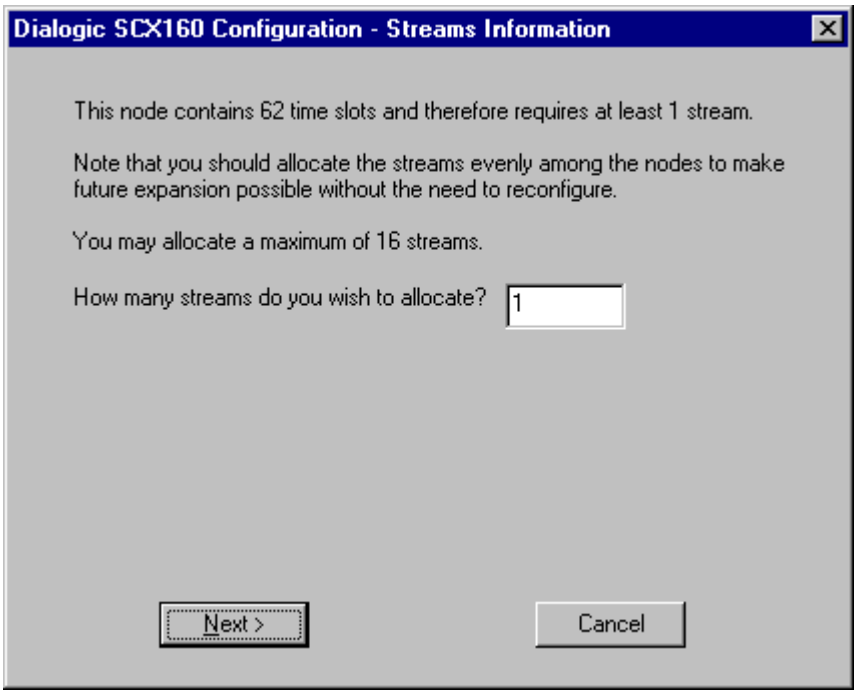

Figure 47. Streams Information Window

The system automatically calculates the minimum number of streams required for the node, however, more SCbus time slots than are required for the currently installed boards can be allocated to reserve the time slots for future use. If additional time slots are reserved, additional boards may be added to these nodes without reconfiguring the system until you run out of reserved SCbus time slots.

14. Click on the **Next>** button to proceed.

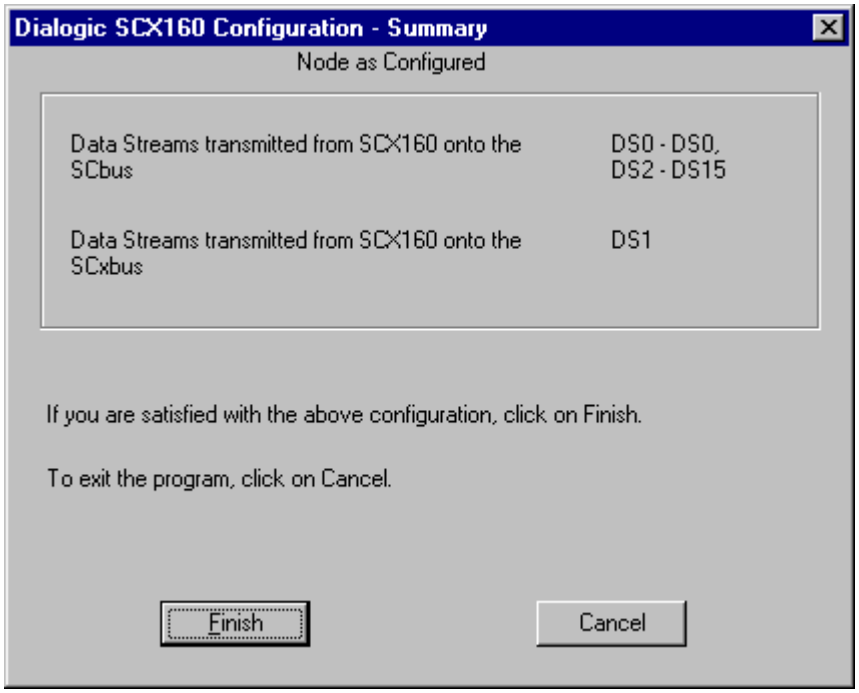

Figure 48. Node as Configured - Summary Window

15. The configuration values listed in *Figure 48. Node as Configured - Summary Window* are:

    - *Data Streams Transmitted from the SCX160 board onto the SCbus* : Indicates the SCxbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCbus.
    - *Data Streams Transmitted from the SCX160 board onto the SCxbus* : Indicates the SCbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCxbus.

16. Review the summary information displayed and either:

    - Click the **Finish** button to save the selections and exit the program, or;
    - Click the **Cancel** button to discard the selections and exit the program.

    Go to *Step 24*.

17. Blocking Configuration:

    When the Blocking Configuration has been selected, *Figure 49. Blocked Streams Information - Resource Selection Window* displays.

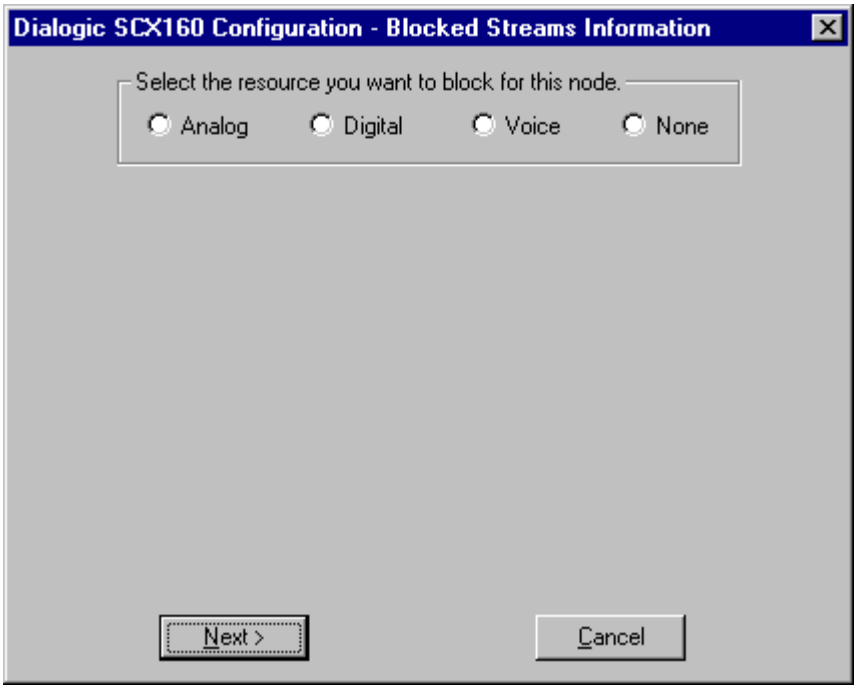

Figure 49. Blocked Streams Information - Resource Selection Window

18. Select one resource to block at this node (analog, voice, digital, etc.) as shown in *Figure 50. Blocked Streams Information Window*. If you do not wish to block a resource on a node, select **None**.

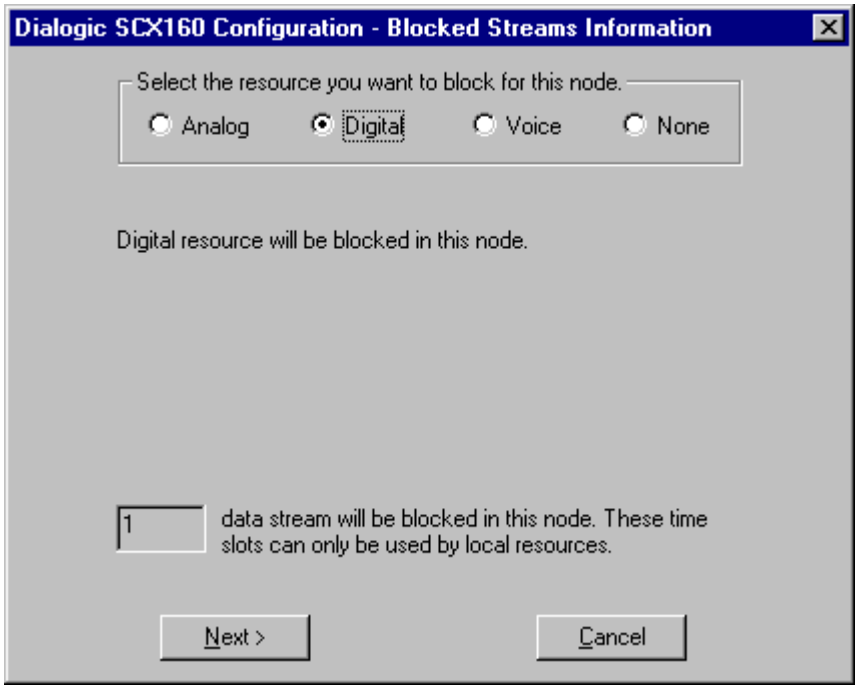

Figure 50. Blocked Streams Information Window

19. If this node requires more than the number of data streams than were actually blocked at Node 1, you will not be allowed to block this resource. You either have to choose another resource or reconfigure from Node 1 to increase the number of blocked streams. If the number of streams blocked at Node 1 is enough to block this resource, the program allows you to go to the next screen when you click **Next>**. The *Streams Information Window (Blocked Configuration)* displays.

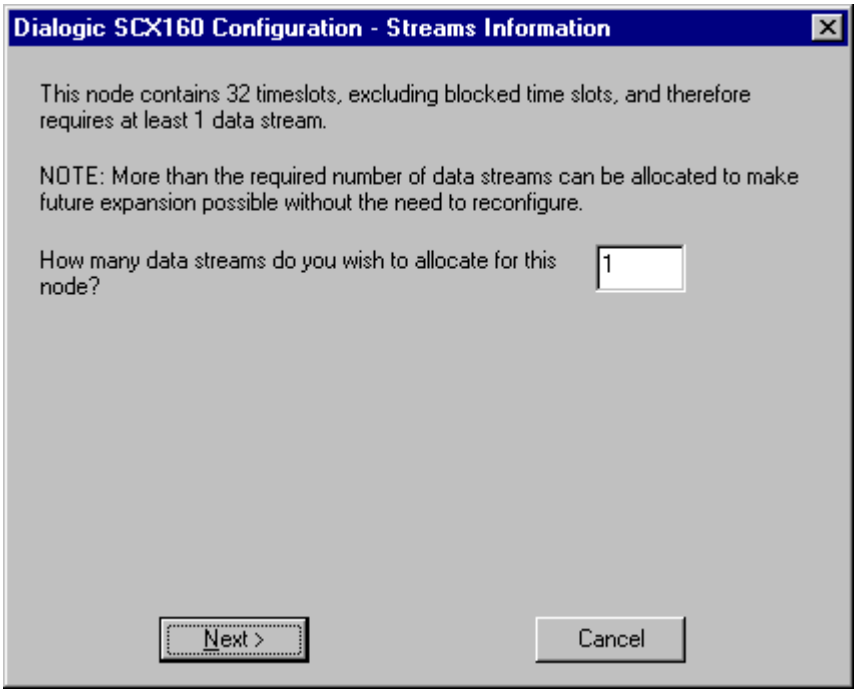

Figure 51. Streams Information Window (Blocked Configuration)

20. In *Figure 51. Streams Information Window (Blocked Configuration)*, the number of data streams required to support the time slots used by the Non-Blocked resources installed is automatically calculated. More data streams than are required for the currently installed resources can be entered to reserve data streams for future expansion (the addition of new resources). Reserving additional data streams will enable future expansion without reconfiguring the system.
21. Enter the number of data streams required to support the time slots used by the Non-Blocked resources.
22. Click on the **Next>** button to proceed and *Figure 52. Dialogic SCX160 Configuration - Summary Window* displays.

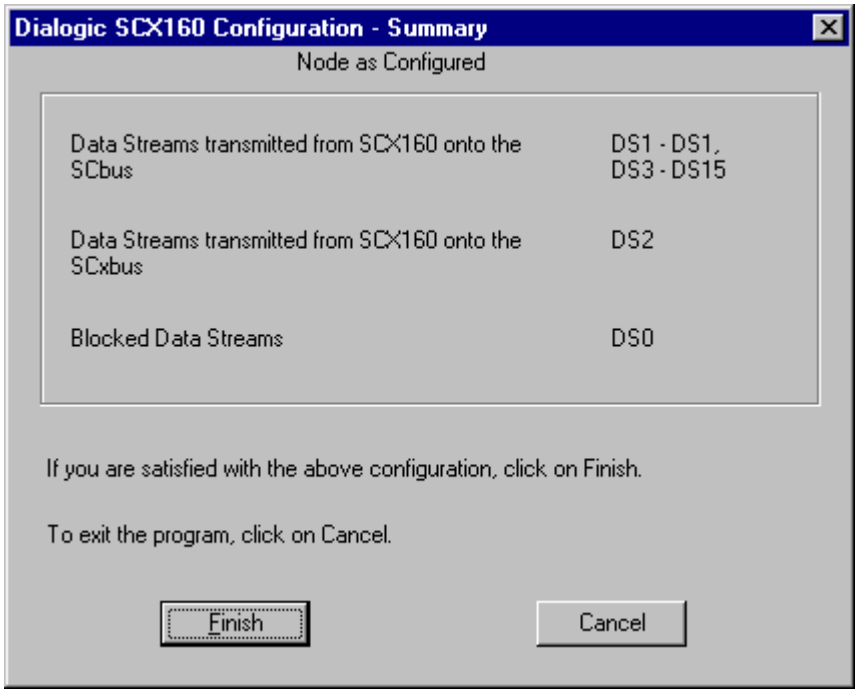

Figure 52. Dialogic SCX160 Configuration - Summary Window

The configuration values listed in *Figure 52. Dialogic SCX160 Configuration - Summary Window* are:

- *Data Streams Transmitted from the SCX160 board onto the SCbus* : Indicates the SCxbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCbus.
- *Data Streams Transmitted from the SCX160 board onto the SCxbus* : Indicates the SCbus data streams (between 0 and 15) that the SCX160 SCxbus Adapter transmits onto the SCxbus.
- *Blocked Data Streams*: The data streams that are blocked from transmitting onto the SCxbus.

23. Review the summary information displayed and either:
    - Click the **Finish** button to save the selections and exit the program, or;
    - Click the **Cancel** button to discard the selections and exit the program.
24. The program now returns to the Dialogic Configuration Manager window (*Figure 53*).

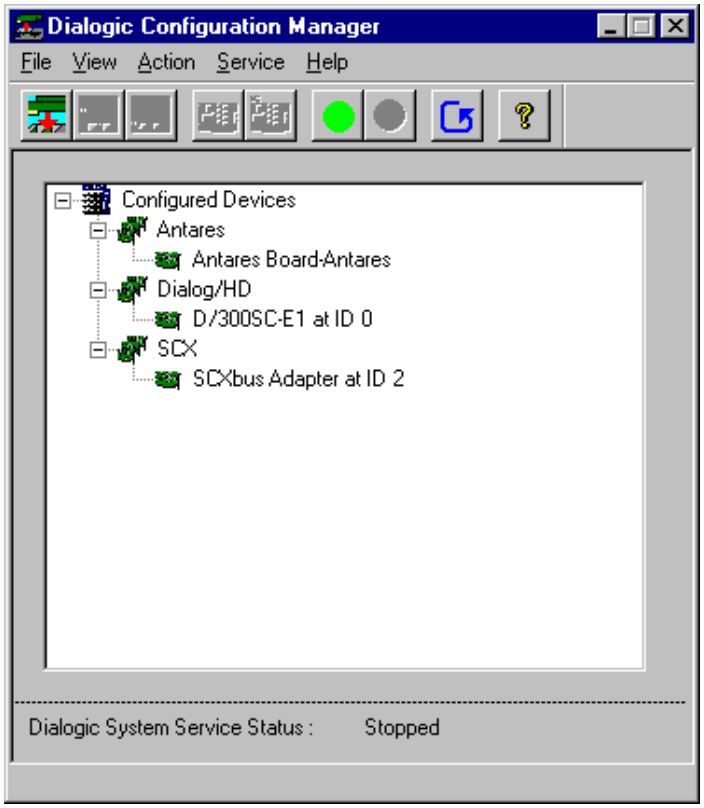

Figure 53. Dialogic Configuration Manager Window

25. Double-click on the SCxbus Adapter and the Properties for SCxbus Adapter screen displays (*Figure 54. Properties for SCxbus Adapter screen*).

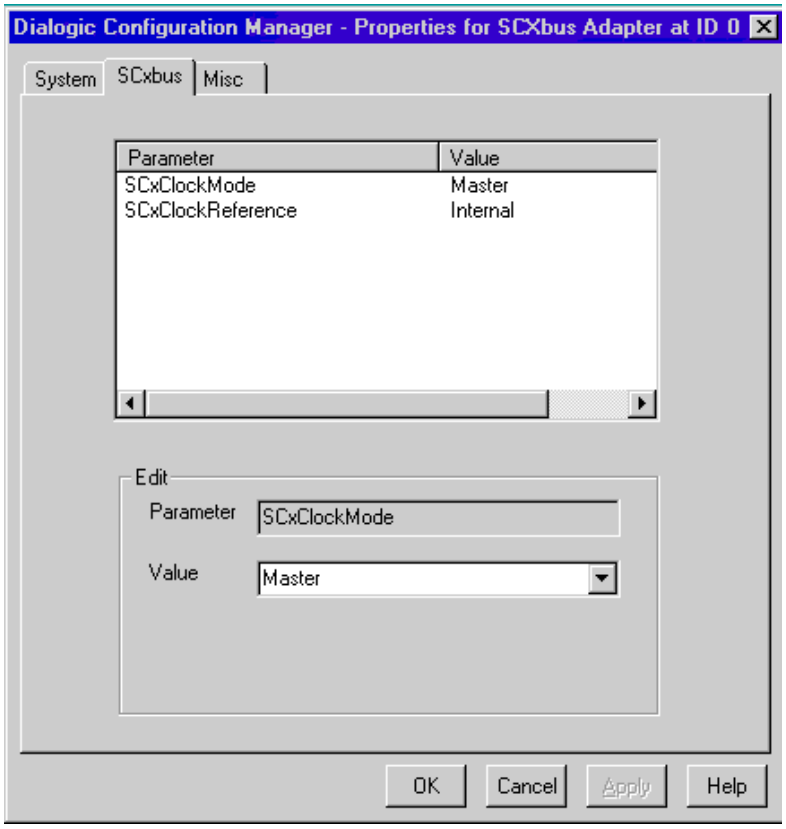

Figure 54. Properties for SCxbus Adapter screen

26. Click on the SCxbus tab to display the current SCxClock settings.

    NOTE: When configuring a multi-node system, at least one node must be set to Master. The default value is Slave mode.

27. Since this is the second node or higher being configured, the value may be set to either Master or Slave, then click Apply to save the changes. If the parameters and values are correct, click OK.

28. Start the Dialogic Service by clicking on the **Green** button at the top of the screen (see *Figure 53*).

NOTE: If at any time during the SCX160 Configuration, **Cancel** was selected to exit out of the program and you wish to return to the SCX160 Configuration Program, do the following:

- Exit out of the Dialogic Configuration Manager by clicking on **File** and selecting **Exit**.
- Start the Dialogic Configuration Manager. There will be an option to configure the SCX160. Choose **Yes** at this point to enter the SCX160 Configuration.

  or

- Choose **Auto Detect Devices** from the action drop down menu in the Dialogic Configuration Manager (*Figure 53*).
- There will be an option to configure the SCX160 board. Choose **Yes** to enter the SCX160 Configuration Program.

Second Time Configuration on any Node - No Configuration Changes

When the Dialogic Configuration Manager (DCM) is started and detects the presence of an SCX160 board, it gives the option to configure the SCX160 board. Since this node is already configured and there are no changes (i.e., the addition or removal of boards after the SCX160 board was configured), the user may choose **No** at this point. However, if the user chooses **Yes**, the SCX160 Configuration Program is invoked and the program exits and returns control to the DCM.

Start the Dialogic Service by clicking n the Green button in the Dialogic Configuration Manager window (see *Figure 38*).

Second Time Configuration on any Node - Configuration Changes

When the Dialogic Configuration Manager (DCM) is started and detects the presence of an SCX160 board, it gives the option to configure the SCX160 board. Since this node is already configured and there are changes (i.e., there was an addition or removal of boards *after* the SCX160 board was configured), the user may choose **Yes** at this point. However, if the user chooses **No**, the node will not be properly configured and the Dialogic Service may fail. Hence, the user MUST select **Yes**.

The SCX160 Configuration Program checks for changes in the configuration and if the number of streams allocated for the node is less than the current configuration requirements, then the program displays an error message and returns control to the Dialogic Configuration Manager (DCM).

However, if the streams requirement for the current configuration is less than the number of streams allocated for the node, the SCX160 Configuration Program updates the *scxmap.dat* file, the *sctscustom* file(in the case of a Blocking Configuration) and returns to the Dialogic Configuration Manager.

Start the Dialogic Service by clicking the Green button in the Dialogic Configuration Manager window (see *Figure 53. Dialogic Configuration Manager Window*).

Appendix C - Related Publications

This appendix lists publications you should refer to for additional information on Dialogic products and SCbus or SCxbus technology.

Dialogic References

- *Quick Install For The SCX160 SCxbus Adapter Board*
- *SCX160 SCxbus Adapter Package Release Notes*
- *SCbus Routing Guide*
- *SCbus Routing Function Reference for Windows*
- *SCbus Configuration Planning Guide*
- *Voice Software Reference for Windows*
- *Voice Software Installation Reference for Windows*
- *Digital Network Interface Software Reference for Windows*
- *FAX Software Reference for Windows*
- *MSI/SC Software Reference for Windows and Linux*
- *Dialogic Hardware Diagnostics Guide for Windows*

General Publications

- Bellamy, John, *Digital Telephony*. 2nd ed. New York: John Wiley & Sons, 1991.
- Newton, Harry, *Newton's Telecom Dictionary (12th Edition)*. Flatiron Publishing, Inc., 1997, ISBN 1-57820-008-3.

Glossary

ACD: Automatic call distributor. An automated (usually software-driven) system that connects incoming calls to agents based on a distribution algorithm. The system also gathers traffic-analysis statistics, such as number of calls per hour, average time holding, and call length.

agent: An operator, transcriber, telemarketing or sales representative, or other employee. In this guide, agent refers to any person using an analog station device who can be connected to a caller or recorded message through the MSI/SC board.

A-Law: A pulse-code modulation (PCM) algorithm used in digitizing telephone audio signals in E-1 areas.

analog: In this guide, analog refers to agent communications between a headset and the MSI/SC or to the *loop-start* type of network interface.

asynchronous function: Allows program execution to continue without waiting for a task to complete. See *synchronous function*.

automatic call distribution: See *ACD*.

baseboard: A term used in voice processing to mean a printed circuit board without any daughterboards attached.

blocking mode: When a telephone call cannot be completed, it is said that the call is "blocked." In blocking mode, it is said that the caller is "receiving a busy."

channel: 1. When used in reference to a Dialogic digital expansion board, a data path, or the activity happening on that data path. 2. When used in reference to the CEPT telephony standard, one of 32 digital data streams (30 voice, 1 framing, 1 signaling) carried on the 2.048 MHz/sec E-1 frame. (See *time slot*.) 3. When used in reference to a bus, an electrical circuit carrying control information and data.

data structure: C programming term for a data element consisting of fields, where each field may have a different type definition and length. The elements of a data structure usually share a common purpose or functionality, rather than being similar in size, type, etc.

daughterboard: In the context of this guide, the MSI/SC daughterboard assembly. The daughterboard enables the MSI/SC hardware to interface to analog station devices.

device: Any computer peripheral or component that is controlled through a software device driver.

digital: Information represented as binary code.

DIP switch: A switch usually attached to a printed circuit board with two settings- on or off. DIP switches are used to configure the board in a semipermanent way.

driver: A software module that provides a defined interface between a program and the hardware.

DTMF: Dual Tone Multi-Frequency. DTMF refers to the combination of two tones which represents a number on a telephone key pad. Each push-button has its own unique combination of tones.

E-1: Another name given to the CEPT digital telephony format devised by the CCITT that carries data at the rate of 2.048 Mbps (DS-1level). This service is available in Europe and some parts of Asia.

event: An unsolicited communication from a hardware device to an operating system, application, or driver. Events are generally attention-getting messages, allowing a process to know when a task is complete or when an external event occurs.

Extended Attribute functions: Class of functions that take one input parameter (a valid Dialogic device handle) and return device-specific information.

full-duplex: Transmission in two directions simultaneously, or more technically, bi-directional, simultaneous two-way communications.

host PC: The system PC in which Dialogic hardware and software are installed and applications are run and/or developed.

IRQ: Interrupt request. A signal sent to the central processing unit (CPU) to temporarily suspend normal processing and transfer control to an interrupt handling routine. Interrupts may be generated by conditions such as completion of an I/O process, detection of hardware failure, power failures, etc.

loop start interfaces: Devices, such as an analog telephones, that receive an analog electric current. For example, taking the receiver off hook closes the current loop and initiates the calling process.

Mu-Law: The PCM coding and companding standard used in Japan and North America (T-1 areas).

MSI/SC: Modular Station Interface. An SCbus-based Dialogic expansion board that interfaces SCbus time slots to analog station devices.

PC: Personal computer. In this guide, the term refers to an IBM Personal Computer or compatible machine.

PCM: Pulse Code Modulation. The most common method of encoding an analog voice signal into a digital bit stream. PCM refers to one technique of digitization. It does not refer to a universally accepted standard of digitizing voice.

rfu: Reserved for future use.

SCbus: Signal Computing bus. A hardwired connection between Switch Handlers (SC2000 chips) on SCbus-based products for transmitting information over 1024 time slots to all devices connected to the SCbus.

SCbus routing functions: Setup communications between devices connected to the SCbus. These functions enable an application to connect or disconnect (make or break) the receive (listen) channel of a device to or from an SCbus time slot.

SCSA: Signal Computing System Architecture. A generalized open-standard architecture describing the components and specifying the interfaces for a signal processing system for the PC-based voice processing, call processing and telecom switching industry.

Signal Computing System Architecture: See *SCSA*.

SpringBoard: A Dialogic expansion board using digital signal processing to emulate the functions of other products. The SpringBoard is a development platform for Dialogic products such as the D/121B.

SRL: Standard Runtime Library containing Event Management functions, Standard Attribute functions, and data structures that are used by all Dialogic devices.

Standard Attribute functions: Class of functions that take one input parameter (a valid Dialogic device handle) and return generic information about the device. The Dialogic SRL contains Standard Attribute functions for all Dialogic devices. Standard Attribute function names are case-sensitive and must be in capital letters. See *Extended Attribute functions*.

synchronous function: Blocks program execution until a value is returned by the device. Also called a blocking function. See *asynchronous function*.

T-1: The digital telephony format used in North America and Japan that carries data at the rate of 1.544 Mbps (DS-1 level).

time slot: In a digital telephony environment, a normally continuous and individual communication (for example, someone speaking on a telephone) is (1) digitized, (2) broken up into pieces consisting of a fixed number of bits, (3) combined with pieces of other individual communications in a regularly repeating, timed sequence (multiplexed), and (4) transmitted serially over a single telephone line. The process happens at such a fast rate that, once the pieces are sorted out and put back together again at the receiving end, the speech is normal and continuous. Each individual pieced-together communication is called a time slot.

ziptone: Short burst of a specified tone to an ACD agent headset usually indicating a call is being connected to the agent console.

Index

E

F

H

I

L

T

U

V

W

NOTES

NOTES

NOTES