Dialogic®

# Dialogic® DSI Signaling Servers

## SS7G41 SIU Developers Manual

## Copyright and Legal Notice



Publication Date: May 2012

## Contents

## Revision History

| Date | Issue No. | Description |
|---|---|---|
| May 2012 | 3 | Minor corrections. Guidelines for Dual Resilient Signaling Server operation move to the SS7G41 Operators manual. |
| September 2011 | 2 | General Availability |
| July 2011 | 1 | Initial version for Beta release |

# 1 Overview

## 1.1 General Description

This manual is applicable to the Dialogic® SS7G41 Signaling Server

***Note:*** Throughout this manual, the product is referred to as the Dialogic® DSI Signaling Server or as the Signaling Server. Where there is a need to identify this specific generation of Signaling Server the alphanumeric designation SS7G41 is used. In addition, the terms “SIU” and “Signaling Interface Unit” may be used to refer to a Dialogic® DSI Signaling Server being operated in SIU mode.

The Signaling Interface Unit (SIU) provides an interface to SS7 networks for a number of distributed application platforms via TCP/IP LAN. In this mode, the units implement the SS7 Message Transfer Part (MTP) and a number of User Parts (ISUP, BICC, TUP, SCCP, TCAP, MAP, IS41 and INAP). In addition, when fitted with Dialogic® DSI SS7 Boards, the SIU can be used to build high performance monitoring applications.

The Dialogic® SS7G41 Signaling Server may be purchased with AC or DC power and may use two different signaling boards. See the *Dialogic® Signaling Servers SS7G41 Hardware Manual* for a fuller description of the Signaling Server hardware.

The SS7G41 Signaling Servers are shipped in TEST Mode - without any operation mode license installed. To enable SIU functionality an appropriate license must be purchased and installed. See Section 2.4, “Multiple Network Support” on page 13 for the list of licenses supported. For information on license installation see the *Software Licensing* section in the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual*.

## 1.2 Related Information

Refer to the following for related information:

- *Dialogic® DSI Signaling Servers SS7G41 Operators Manual*
- *Dialogic® DSI Signaling Servers SS7G41 SWS Developers Manual*
- *Dialogic® DSI Signaling Servers SS7G41 Hardware Manual*
- *Dialogic® DSI Components Software Environment Programmer’s Manual (U10SSS)*
- *Dialogic® DSI Signaling Servers SNMP User Manual (U05EPP)*

The current software and documentation (including the product datasheet) supporting Dialogic® DSI Signaling Server products is available on the web at the following site:

http://www.dialogic.com/support/helpweb/signaling/

The following manuals should be read depending on the protocol options installed on the SIU:

- *ISUP Programmer’s Manual (U04SSS)*
- *SCCP Programmer’s Manual (U05SSS)*
- *TCAP Programmer’s Manual (U06SSS)*
- *MAP Programmer’s Manual (U14SSS)*

- *IS41 Programmer's Manual (U17SSS)*
- *INAP Programmer's Manual (U16SSS)*
- *SCTP Programmer's Manual (U01STN)*
- *M3UA Programmer's Manual (U02STN)*
- *M2PA Programmer's Manual (U03STN)*

## 1.3 Applicability

This manual is applicable to SS7G41 with SIU Mode Release 1.0.1 or later.

## 1.4 Signaling Overview

The signaling capability of the SIU depends on the number and type of signaling boards installed. Up to a maximum of 64 link sets and 256 signaling links are supported.

All link sets terminate at an adjacent signaling point, which may be a Signaling Transfer Point (STP), allowing the use of the quasi-associated signaling mode. When operating as a pair, resilience is provided at MTP3 through the use of a link set between the two units.

In addition to SS7 over TDM signaling, the SIU supports the SIGTRAN M2PA and M3UA protocols. A maximum 256 M2PA or M3UA links are configurable - depending on the license installed.

The SIU will also allow mixed configurations deploying SS7 over TDM, SS7 over ATM, SS7 over M2PA and SS7 over M3UA signaling. Resilience can be achieved using M2PA or M3UA links between a pair of units.

## 1.5 Functional Summary

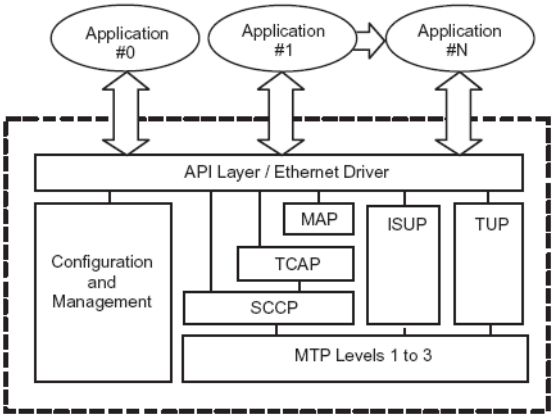
### 1.5.1 SIU Mode Overview

The Signaling Server, when operating in SIU Mode, provides an interface to SS7 networks for a number of distributed application platforms via TCP/IP LAN. In this mode, the unit implements a number of User Parts (ISUP, BICC, TUP, SCCP, TCAP, MAP, IS41 and INAP) operating over either M3UA or MTP3 utilizing Low Speed (LSL), High Speed (HSL), Asynchronous Transfer Mode (ATM), or M2PA SS7 Signaling Links.

The SIU supports multiple SS7 signaling links within the same PCM trunk interface or over multiple PCM trunks. The SIU examines the timeslots carrying the SS7 information and processes them accordingly, then outputs this data to the LAN using TCP/IP. Similarly, it takes commands from the TCP/IP LAN and converts those to SS7 signals for transmission to the SS7 network.

The SIU terminates the signaling and distributes the extracted information to multiple application platforms. In the case of circuit switched telephony, these are the platforms that manage the bearer (non-signaling) channels. Driver software manages communication between the application and the SIU.

**Figure 1. Structure of SIU**




Each SIU can optionally be used as one half of a pair of units operating in a dual resilient configuration. The two units are designated SIU A and SIU B and a single signaling point code is allocated to the SIU pair. See Chapter 6, “Development Guidelines” for more information.

For circuit-related operation, the SIU provides the ability to automatically distribute the call messaging between a number of physically independent application platforms, thus providing a degree of fault tolerance within the application space.

The Application Programming Interface (API) between the application and the SIU is message based. Each command issued by the application to the SIU is packaged in a message structure and sent to the SIU using the C-library functions and drivers provided. In the receive direction, information is conveyed to the user application in structured messages placed in a sequential queue.

The SS7 signaling may be presented from the network multiplexed in a timeslot on a T1 (1.544 Mbps, also known as DS1) or an E1 (2.048 Mbps) bearer.

### 1.5.2 Application Software

The SIU provides an SS7 interface for applications running on remote platforms (host computers). Each application may be implemented as a process within a multi-tasking operating system on the host computer, or, in the case of a non multi-tasking host, as a single application task (or program). An application may be any of the following:

- A User Part with direct access to MTP or M3UA
- A telephony application with access to the ISUP User Part
- A local sub-system using SCCP (Connectionless and Connection-oriented)
- A local-sub-system using TCAP (Transaction Capabilities)
- A local-sub-system using MAP (Mobile Application Part)
- A local-sub-system using IS41 (ANSI Mobile Application Part)
- A local-sub-system using INAP (Intelligent Network Application Part)

***Note:*** TCAP and applications above MAP, INAP and IS41 may be distributed using a Distributed Transaction Server (DTS), allowing a highly scalable architecture. See the *DTS User Guide* for further information.

This provides a flexible implementation for a number of SS7 functions such as Service Switching Point (SSP), Service Control Point (SCP), mobile HLR and Intelligent Peripheral (IP).

Each application task is assigned a unique module identifier (module ID) and communicates with other tasks in the system using a message based Inter-Process Communication (IPC) mechanism. The software library that manages communication between each SIU and the host reserves five module IDs for user applications, and a further module ID to receive management status and event indications from the SIU.

Examples of application modules and management functions are supplied in source code form for use on the host computer.

### 1.5.3 Fault Monitoring

The SIU is able to detect internal fault conditions and report these to the user. The internal faults are combined with external events, to provide an alarm reporting function, which has several possible interfaces to the user, and may be local or remote. For further information on alarm functions refer to “Alarms” on page 26.

#### 1.5.3.1 Diagnostic Log Files

The SIU is able to generate several diagnostic log files for use in the event of an unexpected system restart. The text files can be recovered from the unit using FTP. Refer to the *Diagnostics* section in the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual* for further details.

### 1.5.4 Management Interface

A management interface is provided and may be accessed either via a VT100-compatible terminal or remotely. Remote console based management is possible via telnet or SSH. In addition the Signaling Server can be managed via a web browser. All of these methods can be used to request information on the status of signaling links and PCM ports. The management interfaces also provides configuration information and activation of tracing. See the management interface section of the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual* for details.

### 1.5.5 IP Security

The SIU offers a number of security features for protection against unwarranted access on its IP interface. It is recommended that the user enables the optional Password Protection feature on the Management Interface port and on the FTP Server port.

For additional security, the SIU is equipped with Secure Shell (SSH) functionality, which supports the tunneling of **telnet** and **RSI** traffic, as well as Secure FTP.

Unused ports are disabled to increase security against unintentional or malicious interference.

Additional security may be gained by separating management and signaling IP traffic. This can be achieved by configuring specific Ethernet ports for traffic and utilizing other Ethernet ports for system management information. Signaling IP traffic security between the SIU and its hosts can be further enhanced by tunneling the IP traffic over SSH. See “Once the SIU has been configured, the host software should be installed and configured on each application platform as described in Chapter 3, “Host Software” for further information.

It should be understood that while the SIU has been designed with security in mind, it is recommended that the SIU accessibility over IP be restricted to as small a network as possible. The SIU supports a firewall and this together with other techniques is discussed further in the Security and Access control section of the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual.*

### 1.5.6 Monitoring

The monitoring capabilities of the Dialogic® Network Interface Board can be used in conjunction with the SIU to realize a high-performance protocol monitor supporting up to two boards, each monitoring a licensable number of links. Data from the monitored links can be transmitted to applications operating on multiple SIU hosts that may be selected on a per monitor link basis.

When used in a passive monitoring mode, the Network Interface Boards treats the signaling timeslot as an HDLC channel. When operating in monitoring mode, the 3rd and successive identical frames may be filtered. It is possible to configure monitoring and terminated SS7 links on the same signaling board.

For further information on the configuration and operation of Monitoring on the SIU, See the *Configuration Guidelines* section in the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual.* For further information on the configuration and operation of Monitoring on the SIU.

# 2 Architecture

## 2.1 Introduction

The SIU provides SS7 signaling capability to a telephony application implemented over distributed platforms. This chapter provides an overview of how the SIU integrates into a system.

## 2.2 Overview

An intelligent telephony network application can be considered as consisting of physical resources (such as voice circuits, databases, etc.), an interface to the signaling network and application programs. The SIU implements a (SS7) signaling interface that is physically independent of the applications that use it. Resilience may be built into a system by using a pair of SIUs in a dual resilient configuration.

Applications communicate with the SIU using a message-based Inter-Process Communication (IPC) mechanism implemented transparently over TCP/IP for inter-platform communication. The IPC mechanism provides a level of operating system independence in the sense that the message definitions are the same irrespective of the operating system used.

The SS7 configuration parameters are specified in a text file contained within each SIU. Refer to the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual.*

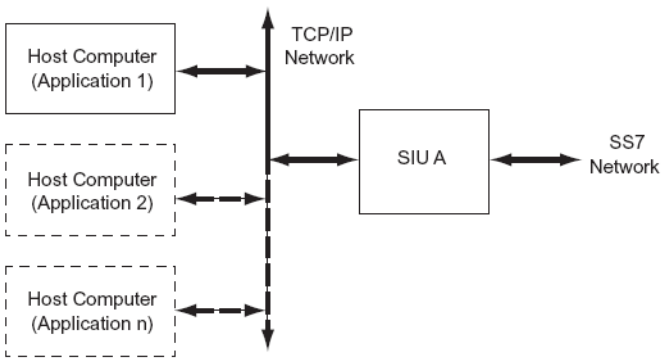
## 2.3 Signaling Topologies

A single SIU may be used standalone, or two units may be configured in a dual resilient configuration. Each SIU may support one or more application (host) computers.

The host computer contains the physical resources controlled by the signaling, such as voice circuits and databases. The SIU extracts SS7 information and conveys it to the application software, which can control the resource accordingly and issue the required responses to the SIU for transport over the SS7 network. In telephony applications, the voice circuits may be distributed between more than one application (or host) computer for resilience.

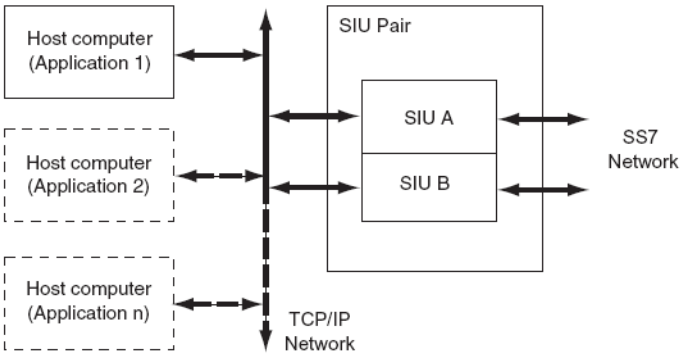
The minimal system consists of a single SIU connected to a single host via Ethernet as illustrated in Figure 2. Dashed lines indicate optional equipment.

**Figure 2. Signaling Paths in a Single SIU Configuration**



This system may be scaled up at initial system build time or later to a dual resilient configuration connected to the maximum number of hosts supported. See the figure below.

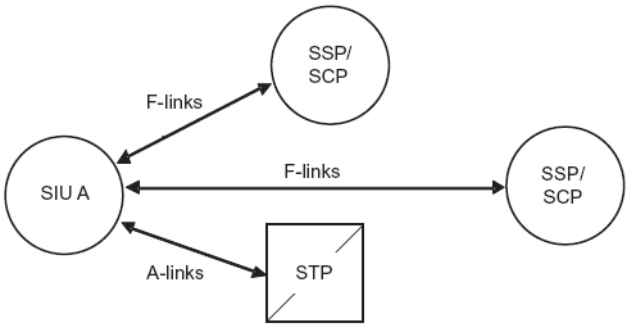
**Figure 3. Signaling Paths in a Dual Resilient Configuration**




The SIU may connect to a number of adjacent signaling points, the maximum number being limited only by the maximum number of link sets supported by the unit. The adjacent SS7 nodes may be Signaling Transfer Points (STPs), Signaling Switching Points (SSPs) or Signaling Control Points (SCPs). The following diagrams indicate possible configurations, although these are not exhaustive.

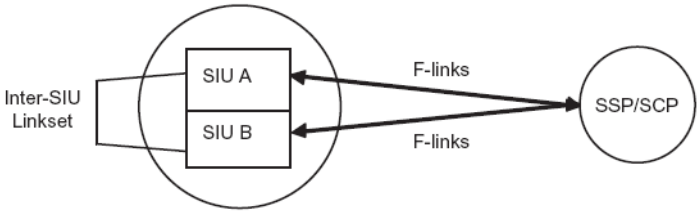
The figure below shows a single SIU connected to an adjacent SSP/SCP and/or STP.

**Figure 4. Single SIU Connected to SSP/SCP or STP**



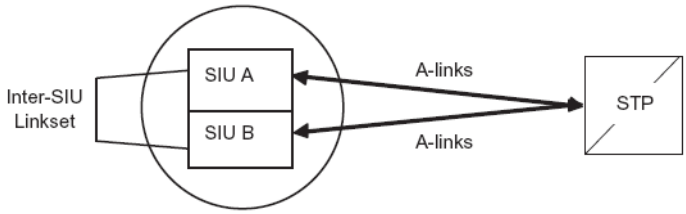

In a dual resilient configuration, the SIU pair *share the same SS7 point code*.The figure below shows an SIU pair connected to a single adjacent SSP/SCP.

**Figure 5. SIU Dual Configuration with Connections to SSP/SCP**



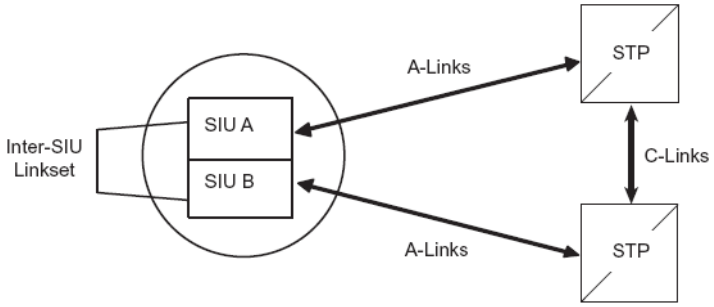
The SIU pair may also be connected to a single adjacent STP (or combination of SSP and STP) as shown in Figure 6.

**Figure 6. SIU Dual Configuration with Connections to STP**




Finally, Figure 7 shows an SIU pair connected to a "mated" STP pair. In this configuration, all the links from the first STP must be terminated at SIUA and all the links from the second STP must be terminated at SIUB.

**Figure 7. SIU Dual Configuration with Connections to Mated STP Pair**




## 2.4 Multiple Network Support

The SS7 Network Context together with a signaling point code uniquely identifies an SS7 node by indicating the specific SS7 network it belongs to. The Network Context may be a unique identifier for a physical SS7 network, for example, to identify an ANSI, ITU, International or National network, or it may be used to subdivide a physical SS7 network into logical sub-networks. An example of the use of logical networks is in provisioning, where the user requires 64 SS7 links between two point codes in a network. As the SIU supports 16 links in a link set, and one link set between two points in a network, only 16 links between two points would normally be achievable. However, if the network is divided into four logical Network Contexts, then up to four link sets may be created between the two point codes, one in each Network Context, thus allowing up to 64 SS7 links to be configured between the two points.

***Note:*** The Network Context has significance only to the configuration of the local node (including the hosts). No external messages include any indication of the Network Context and the configuration of remote systems is unaffected.

The SIU mode is able to support architectures in which a single SIU or dual resilient SIU pair are connected into one or more different SS7 networks. The SIU or SIU pair can also independently terminate multiple local point codes within the same network. "Support for Multiple Local Point Codes" on page 14 and "Support for Multiple Networks" on page 15 following describe these different architectures. Further details on the specific changes required to convert a configuration to use multiple Network Contexts can be found in the *Dialogic® DSI Signaling Servers SS7G41*

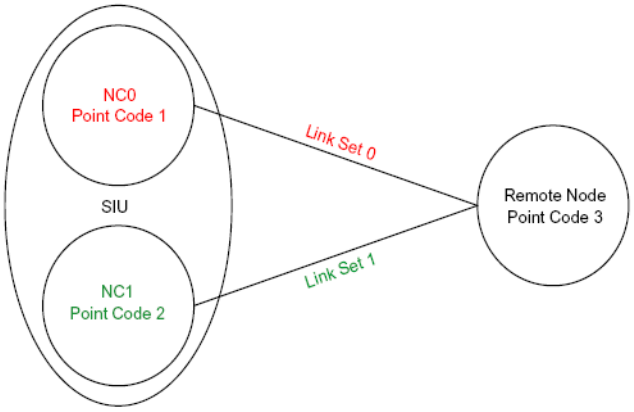
*Operators Manual.* The SIU can support up to four Network Contexts where each Network Context is a different network or different independent local point code within the same network. In the configuration commands or MMI commands, Network Contexts are designated NC0, NC1, NC2 or NC3. Network Context NC0 is also referred to as the default Network Context since this is the Network Context that is assumed if no other explicit value is specified within the command.

### 2.4.1 Support for Multiple Local Point Codes

In some situations, it is desirable to have an SIU terminate more than one local point code within the same SS7 network. Each local point code can have separate routes and associated pairs of link sets to a destination point code. This means that adding additional local point codes allows additional link sets to be used to send traffic to a destination point code. As link sets are limited to 16 links adding more link sets using multiple local point codes effectively allows a larger total number of links to carry traffic to any single destination point code.

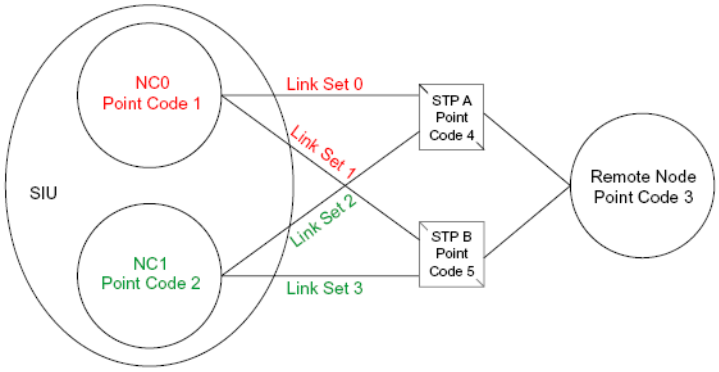
The figure below shows a simple configuration that uses two Network Contexts to allow a single SIU to connect to the remote node using two link sets from two independent local point codes. Link set 0 and 1 are configured in Network Contexts NC0 and NC1 respectively.

**Figure 8. Multiple Network Contexts to Support Multiple Local Point Codes**




The figure below extends the previous example to show a configuration with an STP pair. This configuration uses two Network Contexts to allow a single SIU to connect to the Remote Node using four link sets from two independent local point codes. An equivalent configuration using a dual resilient pair is also possible.

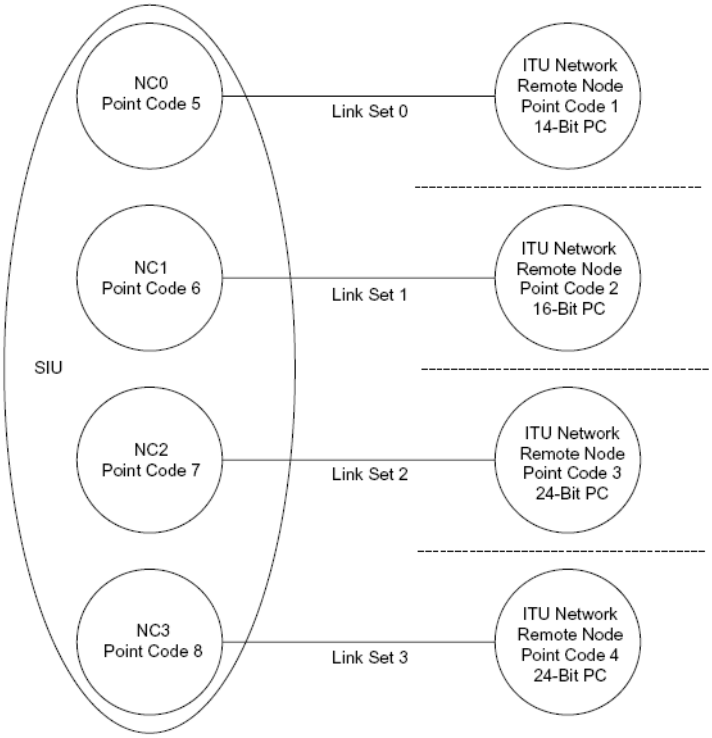
**Figure 9. Multiple Network Contexts with an STP Pair**




## 2.4.2 Support for Multiple Networks

The Network Context-based configuration of the SIU mode allows the settings and behavior to be configured independently for each Network Context. This allows a system to be configured with mixed ITU and ANSI network types or allows multiple networks of the same type to be configured with different settings.

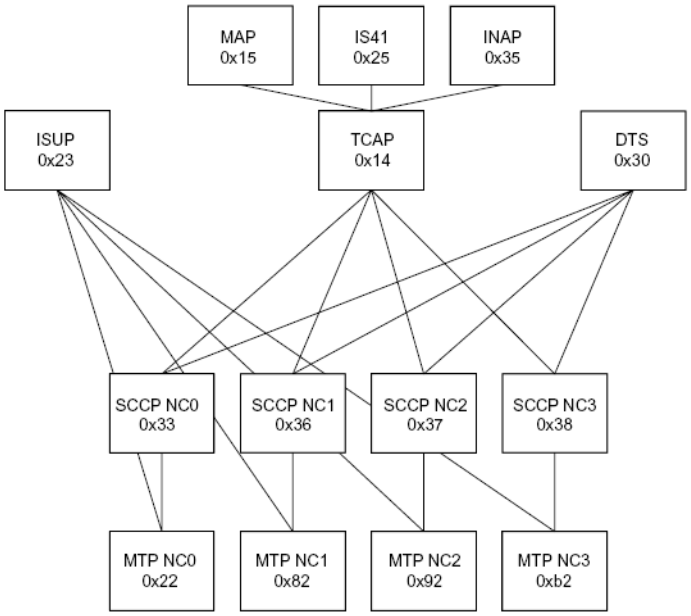
**Figure 10. Multiple Network Contexts Support for Multiple Network Types**



### 2.4.3 Protocol Handling for Multiple Network Contexts

The figure below shows the use of multiple Network Contexts from an application perspective and provides examples of the module IDs for the various application layers.

**Figure 11. Module IDs for Use with Multiple Network Contexts**




#### 2.4.3.1 MTP Applications

Since there is one instance of MTP3 for each Network Context, messages that are destined for a specific network must be sent to the correct MTP module ID as shown in Figure 11 above.

In most SIU configurations, MTP is not the highest protocol layer and the sending of messages to the correct module is handled by the higher layer modules without further user interaction.

#### 2.4.3.2 SCCP Applications

In the same manner as MTP3, there is one instance of SCCP for each Network Context; therefore, messages that are destined for a specific network must be sent to the correct SCCP module ID as shown in Figure 11.

When TCAP or DTS is used above SCCP, those modules handle the sending of messages to the correct module without further user interaction.

#### 2.4.3.3 ISUP Applications

ISUP applications do not need modification, the config.txt parameters are sufficient to identify the Network Context.

#### 2.4.3.4 TCAP, MAP, INAP and IS41 Applications

Where a dialog is initiated remotely, no change is required since TCAP, MAP, INAP and IS41 automatically determine which Network Context is appropriate. Where the dialog is initiated locally, the application must specify the Network Context to which the message is destined. This effectively indicates the point code to be used as the originating point code.

The Network Context should be indicated in the first message for the dialog being used. In the case of TCAP, this is in the first TCAP service request, typically an Invoke Req, using the TCPPN_NC parameter. For MAP, IS41 and INAP, the Network Context should be indicated in the Open Request message, instead of using the MAPPN_NC, IS41PN_NC and INAPPN_NC parameters respectively.

If a Network Context is not specified, the default Network Context, NC0 is assumed.

#### 2.4.3.5 DTS Applications

DTS users should follow the instruction in "TCAP, MAP, INAP and IS41 Applications" on page 17 above, which also apply when using DTS. The DTS_ROUTING_REQ message includes a **DTSPN_network_context** parameter that should be used to indicate the network and hence the local point code that a specified sub-system is part of. If this parameter is not specified, the default Network Context, NC0 is assumed.

To route messages to the correct SCCP instance, you must specify the DTC option, DTC_ROUTE_MSG_VIA_DTS. This option is set via bit 0 in the **options** field of the DTC_MSG_CONFIG (0x776c) configuration message.

## 2.5 Connection of Bearer Channels

In some applications, signaling channels are multiplexed onto the same physical bearer as the voice circuits being controlled by the application. In these circumstances, the signaling channels should be extracted by external equipment prior to connection to the SIU.

## 2.6 Software Environment

The SIU software environment is based on a number of communicating processes, each with its own unique identifier or “module ID”. Each module in the system runs as a separate task, process or program (depending on the type of operating system). Inter Process Communication (IPC) is message based and is achieved by using the set of library functions in Table 1.

**Table 1. Library Functions for Inter Process Communications**

| Function Name | Purpose |
|---|---|
| **getm( )** | Allocate an IPC message |
| **relm( )** | Release an IPC message |
| **GCT_send( )** | Send an IPC message to a process |
| **GCT_receive( )** | Blocking receive |
| **GCT_grab( )** | Non-blocking receive |
| **GCT_set_instance( )** | Indicate that a message is destined for a particular SIU |
| **GCT_get_instance( )** | Determine which SIU sent a message |

These functions are provided as a library to allow the application programs running on the host computer to communicate with the SIU. The IPC message (or MSG) is a “C” structure containing a fixed format header field and a data buffer for variable length parameter data. A complete description of the IPC functions and message structure is given in the *Software Environment Programmer’s Manual*.

In a dual resilient configuration, SIUA is identified by the IPC mechanism as **instance 0** and SIUB as **instance 1**.

## 2.7 Communication Between SIU and Host Application

Host software, in binary form, enabling user applications on Windows, Linux, and Solaris to communication with the SIU is available from your Dialogic support channel.

This software allows the environment described above to be established on the host operating system. This software consists of the IPC library functions and a number of processes that manage the communication between the host and each SIU.

The SIU identifies each unique host computer using a host identifier value (or **host_id**). Values range from zero to one less than the total number of hosts. The first host is assigned host_id 0.

A process running on the host computer manages the communication between the application and an SIU. The application issues two messages to this process to connect to (and use) an SIU. Each host connects to a different TCP/IP port on the SIU; hence the port selected assigns the host_id value to the computer requesting the connection. The host receives status indications (in the form of an IPC message) from each connected SIU indicating changes in the availability of this link. If a failure occurs, an event is sent to the host indicating that the connection to the SIU has been lost, allowing the host to take corrective action.

Once the SIU detects an active connection to a host computer, all configured SS7 links are activated. If all host connections fail, the signaling links are deactivated (until one or more host links become active).

## 2.8 Inter-SIU Communication

In a dual resilient configuration (one unit nominated as SIUA, the other as SIUB), two physically independent communication channels exist between the two units.

Control information is exchanged over the Ethernet. Signaling messages are exchanged (when necessary) over an inter-SIU SS7 link set, *which must be configured for correct dual resilient operation.*

The preferred route for messages transmitted from an SIU is over the links connecting that unit to the appropriate adjacent point code (a point code that is either the final destination or a route to the final destination). If no signaling link to an appropriate adjacent point code is available, the transmit traffic is passed to the other SIU via the inter-SIU link set. If the inter-SIU link set fails, transmit messages fall back to being passed over the Ethernet.

If the inter-SIU link set fails (causing the Ethernet link to be used for transmitted messages), message loss may occur at the point where the preferred route fails.

The SS7 network is free to deliver received messages to either SIU. Special processing at the User Part level (ISUP or TCAP) ensures that any message received for a call or transaction being handled by the other unit is routed over the Ethernet.

The inter-SIU link set is configured in the same manner as normal link sets, for details, refer to *Dialogic® DSI Signaling Servers SS7G41 Operators Manual.*

The inter-SIU link set may consist of one or more signaling links configured on one or more signaling ports (T1/E1), distributed between one or more signaling boards. Resilience on the Inter SIU link set may be achieved by configuring two links in the inter-SIU link set, each on a separate signaling board. The inter-SIU link may also be conveyed over M2PA or M3UA avoiding the requirement for a TDM link and cabling between the units.

## 2.9 Call Control Applications

### 2.9.1 Standalone Operation

When the SIU is not used in a dual resilient configuration, the circuits that are used by the application are activated on the SIU automatically at start-up. You are free to use these circuits for telephony once the connection between the host and SIU is active (and the SS7 links are in service).

### 2.9.2 Call Control Interface

Call control primitives are conveyed between ISUP running on the SIU and the host using IPC messages. One message type is used to send request messages from the user to the User Part module, while a second message type is used to send indications in the opposite direction. A third message type provides the user application with indications of the remote signaling point status.

The message types are:

- **ISP_MSG_TX_REQ**
  Conveys primitive from host to User Part.
- **ISP_MSG_RX_IND**
  Conveys primitive from User Part to host.
- **ISP_MSG_STATUS**
  Conveys remote signaling point status to the host.

The format of these messages is defined in the *ISUP Programmer's Manual*.

In a dual resilient configuration, it is necessary for the application to assign a group to a particular SIU before any call control or management primitives can be exchanged for circuits in that group. The host ensures that circuit-related messages are routed to the correct SIU by setting the destination instance of the IPC message using the **GCT_set_instance( )** library function. Messages issued by the SIU to a host computer are automatically delivered to the host_id that is specified as part of the per circuit group data in the SIU configuration.

An example telephony application is provided in "C" source code. This makes use of a library of interface functions that provide you with a "C" structured representation of the protocol primitives for convenience.

### 2.9.3 Circuit Supervision Interface

The Circuit Supervision Interface allows the host to carry out the following operations:

- Reset a circuit or circuit group
- Abort a reset cycle
- Block a circuit or circuit group (Maintenance, Hardware, or Software)
- Unblock a circuit or circuit group (Maintenance, Hardware, or Software)
- Abort a blocking or unblocking attempt

Commands originated by the host take the form of a Circuit Group Supervision Request. On completion of command execution, the host receives notification in the form of a Circuit Group Supervision Confirmation, indicating that the expected response or acknowledgement has been received from the network for the command. Events initiated at the remote end of the network are notified to the user in a Circuit Group Supervision Indication.

The message structure and parameters for each message are defined in relevant protocol *Programmer's Manual.*

### 2.9.4 ISUP Detection of Failed SIU Hosts

It is possible to configure ISUP to detect failed (or inactive) SIU hosts and initiate circuit group blocking to the network. This ensures that the network does not attempt to initiate calls on circuits for which there is no active application and calls would consequently fail.

The use of this feature requires the user application to respond to the CAL_MSG_HEARTBEAT message (see Section 4.6.11, "CAL_MSG_HEARTBEAT" on page 51) that is periodically issued by the ISUP module. In the event that no response is received within a pre-determined time, the ISUP module initiates hardware circuit group blocking to the network.

The feature is optional and is activated by setting bit 9 in the **<options2>** parameter of the ISUP_CFG_CCTGRP command. When the bit is set, heartbeat messages are generated and sent to the user_id configured for the circuit group. If the option is not set, automatic blocking of circuits is not performed for the circuit group and heartbeat messages are not sent to the user application.

When the feature is activated, the ISUP module periodically sends a heartbeat message, CAL_MSG_HEARTBEAT, to the user application, to determine status. A single heartbeat message is sent every 30 seconds regardless of the number of circuit groups configured per SIU host. The application must respond by confirming the message (using the **confirm_msg( )** function instead of releasing the message using the **relm( )** function).

If the user application fails to respond to a heartbeat message within 3 seconds, the ISUP module considers the application to be unavailable and out of service. Circuit groups associated with the application and for which autoblocking is configured are hardware blocked and a blocking message is sent to the network (CGB).

Once circuit groups have been blocked, ISUP continues to send heartbeat messages with the UIHB_FLAGS_CGRPS_BLOCKED flag (bit 0) set to a value of 1.

Following recovery, the application should clear the automatically imposed hardware blocking condition by requesting either a reset or hardware unblocking using the normal circuit supervision request mechanism and resuming to subsequent heartbeat messages.

## 2.10 Transaction-Based Applications

Applications that need to exchange non-circuit related information over the SS7 network (such as for the control of a Mobile Telephone Network or for an Intelligent Network application) do so by exchanging information between *sub-systems* using the services of SCCP. A sub-system is an entity that exchanges data with other entities by using SCCP.

***Note:*** TCAP and the higher layer protocol modules INAP, MAP, and IS41 may be supported on the SIU directly or may be distributed onto the host using the Distributed Transaction Server (DTS) functionality, allowing a highly scalable architecture. A DTS may also be used when MAP, INAP or IS41 operate on the SIU and the user wishes to distribute its applications across multiple hosts. For an overview of DTS operation, see the *Dialogic® SS7 Protocols DTS User Manual*.

The SIU provides the capability to configure local sub-systems and routing to remote resources. The intelligent functionality of each local sub-system is provided by the user application running on one or more host computers.

### 2.10.1 Management of Local SCCP Sub-Systems

The availability of local sub-systems is conveyed by sending SCCP management request messages of type **SCP_MSG_SCMG_REQ** as defined in the *SCCP Programmer's Manual* to the TCAP module on the SIU. These messages can be issued either by the individual local sub-system tasks (that is, **APPn_TASK_ID**) or by the management module (that is, **REM_API_ID**) on behalf of all the local sub-systems. This choice is left to you.

You may request the return of a confirmation message (using the **rsp_req** field) from the SIU to verify that the message has been received. Successful receipt of a confirmation message implies that the path through the protocol stack down to and including the SCCP module is operational.

***Note:*** The confirmation message is returned to the module that issued the original management request. This is not necessarily the local sub-system module.

### 2.10.2 Sub-System In Service

When the local sub-system task first starts running a User In Service (UIS) request should be issued to the SIU.

While the local sub-system remains operational, further UIS requests should be issued to the SIU on a periodic basis. It is recommended that a UIS request be issued approximately every 12 seconds.

On receipt of each UIS request, the SIU starts (or restarts) a 30-second timer. Should the timer expire before the next UIS request is received, then the SIU assumes that the local sub-system is no longer operational and reacts as if a User Out of Service (UOS) request had been received for the local sub-system.

### 2.10.3 Sub-System Out of Service

When the local sub-system task is to be taken out-of-service in a controlled manner a User Out of Service (UOS) request should be issued to the SIU.

As there is no requirement for the UOS request to be issued by the local sub-system to which it refers, another module may issue the UOS notification if a local sub-system goes out of service in an uncontrolled manner.

### 2.10.4 TCAP-Based Applications

For TCAP operations, each TCAP user (or sub-system) is implemented as a unique process running on one or more host computers. The module identifier for each local sub-system is assigned in the SIU configuration file.

Each dialog (or conversation) is uniquely identified by a dialog ID (identifier). Definitions in the SIU protocol parameter file reserve a separate contiguous block for incoming and outgoing dialogs. Each SIU can manage up to 65,535 simultaneous active dialogs.

In a dual resilient configuration, the maximum supported number of dialogs is available on each unit. A remotely-initiated dialog is handled by the SIU that received the first TCAP message from the remote TCAP entity. An outgoing dialog is handled by the SIU that processes the first dialog or component request from the user application. Each dialog is permanently associated with either SIUA or SIUB for its duration. The application should use the **GCT_set_instance( )** to address each TCAP primitive request to the correct SIU. The **GCT_get_instance( )** function enables the originating SIU for each primitive indication to be identified.

> ***Note:*** Systems using TCAP should ensure that the rsi routing algorithm is set to “-l1” to enable the full number of TCAP dialogs to be available to the host application and to cause the TCAP primitives to be routed according to the SIU instance.

### 2.10.5 TCAP Application Interface

The interface between the user application and the TCAP protocol is defined in the *TCAP Programmer’s Manual* and the *SCCP Programmer’s Manual*.

An example application is provided in “C” source code to demonstrate how a local sub-system application program can interface with the TCAP/SCCP protocols running on the SIU. This makes use of a library that provides the user with a “C” structured representation of the protocol primitives for convenience.

Each local sub-system task should notify the SIU as it becomes available using a User In Service N-STATE Request (UIS) and before it terminates using a User Out of Service N-STATE Request (UOS). If the local heartbeat detection mechanism is enabled, the local sub-system should also continue to issue UIS N-STATE Requests approximately every 12 seconds.

Maintenance events and software events from TCAP and SCCP are reported to the user’s own management module, which uses the REM_API_ID module ID. This module receives the following messages:

- **TCP_MSG_MAINT_IND**
  Maintenance event from TCAP.

- **TCP_MSG_ERROR_IND**
  Software event from TCAP.
- **SCP_MSG_MAINT_IND**
  Maintenance event from SCCP.
- **SCP_MSG_ERROR_IND**
  Software event from SCCP.

The user's management module is typically configured to receive SCCP management indications by configuring it as a concerned local sub-system.

***Note:*** While the user's management module is not a local sub-system in terms of sending and receiving protocol primitives, it is configured as a local sub-system on the SIU to allow it to receive SCCP management indications.

SCCP management indications use the following message:

- **SCP_MSG_SCMG_IND**
  Management indication from SCCP.

The user's management task may use the following message to set up default parameters within the TCAP module, although you may elect to insert default parameters prior to calling the TCAP library functions:

- **TCP_MSG_DEF_PARAM**
  Set up default parameter values.

None of the other messages described in the *TCAP Programmer's Manual* should be issued by the application because they would conflict with the internal operation of the SIU. In particular, the TCAP configuration message is issued by the internal SIU management task at initialization.

The only messages that may be sent directly to the SCCP module are the messages to read status and statistics and to add and remove entries in the Global Title Translation table. These are message types SCP_MSG_R_STATS, SCP_MSG_R_SSR_STATS, SCP_MSG_GTT_ADD and SCP_MSG_GTT_REM as described in the *SCCP Programmer's Manual*.

### 2.10.6 Multiple TCAP Application Hosts

Redundancy may be achieved in the application space by distributing the local sub-system application between more than one application platform (or host) connected to the SIU via the Ethernet. The parameter file on the SIU assigns ranges of dialog identifiers for incoming and outgoing dialogs for each host. The application program running on each host must therefore ensure that only dialog identifiers from the assigned range are used.

Incoming dialogs are distributed between multiple instances of the application using an algorithm defined in the SIU config.txt file. This may be set to load balance between hosts, cycle through each host in turn or fill the hosts in sequence.

### 2.10.7 MAP Application Interface

The Mobile Application Part (MAP) layer of the SS7 protocol enables the control of services within the GSM mobile telephone network.

The interface between the user application and the MAP protocol is defined in the *MAP Programmer's Manual*. An application interfacing to the MAP protocol requires the sub-system management procedures described above for SCCP and TCAP.

Distribution of the application above MAP between multiple hosts MAP can be achieved by running DTS above MAP on the SIU. See the *DTS User Guide* for further information.

### 2.10.8 IS41 Application Interface

The IS41 layer of the SS7 protocol enables the control of services within the ANSI mobile network.

The interface between the user application and the IS41 protocol is defined in the *IS41 Programmer's Manual*. An application interfacing to the IS41 protocol requires the sub-system management procedures described above for SCCP and TCAP.

Distribution of the application above IS41 between multiple hosts can be achieved by running DTS above IS41 on the SIU. See the *DTS User Guide* for further information.

### 2.10.9 INAP Application Interface

The Intelligent Network Application Part (INAP) layer of the SS7 protocol enables the control of services within the intelligent network.

The interface between the user application and the INAP protocol is defined in the *INAP Programmer's Manual*. An application interfacing to the INAP protocol requires the sub-system management procedures described above for SCCP and TCAP.

Distribution of the application above INAP between multiple hosts can be achieved by running DTS above INAP on the SIU. See the *DTS User Guide* for further information.

## 2.11 Resilience

### 2.11.1 IP Resilience

The SIU ships with four IP ports. These ports may be configured with IP addresses in separate IP networks to allow greater IP resilience on the SIU. IP addresses are configured using the IPEPS command. The IPGWI command allows configuration of the default gateway and additional gateways.

As the SIU supports static, rather than dynamic IP routing, the SIU may **not** be configured with different IP addresses within the same IP network. Instead, resilience between two IP ports within the same network can be achieved by using IP port bonding, which allows two physical IP ports to be bonded together in an active/standby configuration under a single IP address.

### 2.11.2 Dual Resilient Operation

In a dual resilient configuration, optimal performance is achieved by distributing control of the circuit groups evenly between SIUA and SIUB. The application requests a particular SIU to manage the signaling for each circuit group at run-time. Control of a circuit group may be transferred by the host from one SIU to the other at any time.

Both SIUA and SIUB contain the same circuit group data in the configuration file. The application activates a group by sending an Activate Group User Command to a particular SIU. The circuits in this group may then be used by the application for telephony. The application may deactivate the group by sending a Deactivate Group User Command to the controlling SIU. This group may then be activated on the other SIU if required. The format of the User Command is described in Chapter 4, "Application Programming Interface".

The application ensures that call primitives and activate/deactivate group commands are routed to the correct SIU by setting the destination instance of the IPC message that conveys the primitive. SIUA is instance 0 and SIUB is instance 1. The **GCT_set_instance( )** library function is provided for this purpose.

Unit failure is indicated to the host by receipt of a status message indicating that the connection to an SIU has been lost. If this occurs, the circuit groups managed by the failed unit may be activated on the available unit. This transfer does not affect calls in the speech/connected state. Calls that are currently being set-up fail; these calls should be attempted again once the transfer is complete.

The transferred circuits should be reset by the application once any call that was active during the transfer has completed. Idle circuits may be reset immediately following the transfer.

### 2.11.3 Fault Tolerance in Call Control Applications

See Chapter 6, "Development Guidelines" for information on building fault tolerant SS7 systems for call control applications using Signaling Gateways SIUs.

### 2.11.4 Fault Tolerance in Transaction Processing Applications

See Chapter 6, "Development Guidelines" for information on building fault tolerant SS7 systems for transaction processing applications using Signaling Gateways SIUs.

### 2.11.5 Use of Multiple Host Computers

For telephony, the circuits multiplexed on a single T1/E1 PCM trunk are usually controlled by the same host. The control of a number of T1/E1 trunks may be divided between more than one host.

The circuits available to the SS7 telephony User Part (ISUP) on the SIU are configured in groups. The configuration data for each group may include an optional host_id, specifying which host computer controls the physical resources. This ensures that protocol messages received for each circuit are routed to the correct host computer.

### 2.11.6 Backup Host Capability

The ability to configure backup hosts allows management and/or signaling messages to be redirected to a backup host application in the event of primary host failure. Backup hosts can be employed when configured for ISUP. Backup hosts may also be used for SCCP operation; however, they may not be used in configurations that utilize DTS/DTC.

When using ISUP for example, this mechanism allows continued use of circuits if the primary host for a circuit group were to fail. Once the primary host link has been recovered, messages are again sent to it from the SIU. See the SIU_HOSTS command.

## 2.12 Management Reporting

The SIU reports management alarms such as PCM trunk status and SS7 level 2 link status to a single software process, identified as module_id **0xef**, that exists by default on host 0. The user management application is responsible for interpreting the management messages (described in Chapter 4, "Application Programming Interface"), performing the appropriate action and distributing these messages to other hosts if required. The identity of the default management host is displayed by the DMHOST parameter on the CNSYP MMI command and may be changed through use of the CNSYS command.

Any host may assume the role of management by sending a management request (in the form of an API_MSG_COMMAND request). On receipt of this request, the SIU begins to send management events to the new host. Unlike the setting of the DMHOST through MMI setting of a management host, using the API_MSG_COMMAND will require the user to transmit the API_MSG_COMMAND with the desired management host identify each time the SIU is restarted.

An optional second management host may also be activated by sending a management request. On receipt of this request, the SIU sends management events to both management hosts.

The selection of which host is the manager, as well as the configuration of an additional management host, allows a user to build a resilient solution to meet their management event reporting needs.

The SIU also maintains a log of management messages for diagnostic purposes in the "syslog" subdirectory of the siuftp account. This log is maintained as a rolling log of up to 10 5MB files containing management messages transmitted to the management host as well as some further diagnostic data. The most recent maintenance log file will have the name “maint.log” the next most recent “maint.log.1” and then “maint.log.2” and so on.

## 2.13 Alarms

The Dialogic® DSI Signaling Server products are able to detect a number of events or alarm conditions relating to either the hardware or the operation of the protocols. Each alarm condition is assigned a severity/class (3=Critical, 4=Major, 5=Minor) and a category and ID, which give more detail about the alarm. There are a number of mechanisms described below, by which these conditions can be communicated to management systems, and ultimately to the system operator. See the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual.*

to for a list of alarm types, and their reporting parameters.

- Active alarms are indicated on the front panel of the Signaling Server, with two LEDs identifying severity; C Fault, M Fault. There is also an additional LED for power faults marked P Fault.
- Alarm events (occurrence and clearing, class, category and ID) may be reported via Management messages to the host application as detailed in Chapter 4, “API Commands”, thus permitting remote monitoring and/or logging.
- Alarm events may be reported to an SNMP manager.
- A system operator can obtain a listing of the current alarm status (CLA, CATEGORY, ID and TITLE) using the ALLIP management terminal command described in the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual*. Test Critical, Major, or Minor may be activated using the ALTEI management command and cleared using the ALTEE management command.

# 3 Host Software

## 3.1 Introduction

For reliability, redundancy and scalability, telephony applications may be implemented over a number of physically independent platforms. The SIU provides the SS7 processing component of such a system and communicates with one or more user applications that run on *host* computers using Ethernet. An SS7 Development Package is installed on each host platform that communicates with the SIU to provide transparent communication between the user application program(s) and the SIU. It is not necessary for the user application to provide any Ethernet or TCP/IP functionality.

The host software environment is based on a number of processes that communicate using messages and message queues. The SS7 Development Package extends the message passing mechanism to work over the Ethernet network and provides a library of interface functions for the user application.

This section of the user manual describes how to install the development package and run the various elements required on host platforms. See Section 5, “Host Utility and Command Syntax” on page 53 for information on the command options and syntax of the utilities provided by the SS7 Development Package.

## 3.2 Contents of the SS7 Development Package

The development package consists of a number of executable programs and libraries or C-source files that are linked with the user’s application. This software is available for multiple operating systems. All programs operate through a console interface.

The following components are included in all operating system implementations of the development package:

- **rsi** (Remote Socket Interface)
  Manages routing of messages between the host and SIU(s). The RSI process automatically starts an instance of **rsi_lnk** (Remote Socket Interface Link) for each SIU that the host is able to communicate with.
- **rsicmd** (Remote Socket Interface Command)
  A utility to configure and start-up a connection from a host computer to an SIU.
- **s7_log**
  A task that receives status and management indication messages from the SIU and displays these as text on the application console.
- **s7_play**
  Reads message contents from an ASCII text file (in a defined format) and sends these messages to the SIU.
- **gctload**
  A task that initializes the host system environment and starts up all other processes, such as RSI, deriving the process and message queue configuration from a text file.
- **tim**
  Receives periodic tick messages from tick and handles protocol timers for other processes.
- **tick**
  A task that interfaces to the operating system and host message passing environment for the purpose of generating periodic tick notification messages.

- **gctlib**
  A library containing the IPC functions that are used by the user application to exchange information with the SIU.
- **system.txt**
  A text configuration file used by gctload providing definitions required to establish the IPC environment. For further details of the format of the file system.txt, refer to the *Software Environment Programmer's Manual.*

Refer to *Chapter 5, "Host Utility and Command Syntax"* for further information on rsi, rsicmd, tick, tim, s7_log, s7_play and gctload.

For information on installation refer to the *Dialogic DSI Components - Software Environment Programmer's Manual.*

## 3.3 Application Operation

Each application runs as a separate task that communicates with the other entities in the system using the IPC library functions. These functions access the IPC environment consisting of message queues, messages and socket interfaces initialized by gctload. Each task within the system including each user application is assigned a unique identifier or *module_id*, which is defined in the system.txt file on the host. The values APPn_TASK_ID are defined in the system.txt file for use by user applications.

The module ID used by the example programs and utilities is shown in the following table. The module ID used by CTU, TTU, MTU and s7_log is set by a command line switch "-m". By convention, the following module IDs are used:

| Program | Value | Mnemonic |
|---|---|---|
| TTU example | 0x0d | APP0_TASK_ID |
| CTU example | 0x1d | APP1_TASK_ID |
| rsicmd | 0xfd | APP15_TASK_ID |
| s7_log | 0xef | REM_API_ID |

These values should be specified as the **user_id** parameter for the protocol configuration command that configures the layer that the example program interfaces with. For example, if CTU is used with ISUP, and CTU uses a module_id of 0x1d, the ISUP_CONFIG user_id parameter must also be set to 0x1d. This value must also appear against a LOCAL definition in the system.txt file on the host.

The operation of gctload and the format of the configuration file system.txt are defined in the *Software Environment Programmers Manual*.

The **rsi** process manages the connection between the host and each SIU. It takes several command line parameters and is normally spawned by an entry in the host's system.txt. The command syntax is given in Section 5.1, "rsi" on page 53.

The connection between the host and the SIU is configured and activated by the **rsicmd** command; the syntax for this is given in Section 5.2, "rsicmd" on page 54. Alternatively, the link may be activated from the application by sending two messages to the **rsi** process; a link configuration message RSI_MSG_CONFIG, followed by a link activation message RSI_MSG_UPLINK. These two messages are described in Chapter 4, "Application Programming Interface". If a host connects to SIUA and SIUB, **rsicmd** or the sequence of two messages should be repeated for each SIU.

The following is an example system.txt file for a Windows SIU host:

```
*
* Module Id's running locally on the host machine:
*
LOCAL           0x00        * timer Module Id
LOCAL           0xb0        * rsi Module Id
LOCAL           0xef        * REM_API_ID Module Id (s7_log)
LOCAL           0xfd        * rsicmd Module Id
LOCAL           0x1d        * ctu Module Id
LOCAL           0x3d        * siucmd Module Id
LOCAL           0x0d        * ttu Module Id
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT        0x20    0xb0    * SSD module Id
REDIRECT        0xdf    0xb0    * SIU_MGT module Id
REDIRECT        0x22    0xb0    * MTP3 module Id for NC0
REDIRECT        0x82    0xb0    * MTP3 module Id for NC1
REDIRECT        0x92    0xb0    * MTP3 module Id for NC2
REDIRECT        0xb2    0xb0    * MTP3 module Id for NC3
REDIRECT        0x14    0xb0    * TCAP module Id
REDIRECT        0x33    0xb0    * SCCP module Id for NC0
REDIRECT        0x36    0xb0    * SCCP module Id for NC1
REDIRECT        0x37    0xb0    * SCCP module Id for NC2
REDIRECT        0x38    0xb0    * SCCP module Id for NC3
REDIRECT        0x32    0xb0    * RMM module Id
REDIRECT        0x23    0xb0    * ISUP module Id
*
* Now start-up the Host tasks ....
*
FORK_PROCESS    .\tim_nt.exe
FORK_PROCESS    .\tick_nt.exe
FORK_PROCESS    .\rsi.exe -r.\rsi_lnk.exe  -l1
*
* Start the Host-SIU link:
* (This should only be done at this point if the user
* application is ready to read messages from its queue)
*
FORK_PROCESS    .\rsicmd.exe 0 0xef 0 123.124.125.126 9000
*
* Example application programs:
*
* FORK_PROCESS  .\ctu.exe -m0x1d -o0x1fff
* FORK_PROCESS  .\ttu.exe -m0x0d -n0x66
```

***Note:*** Some operating systems use "\" as the directory separator token, while others use "/". Care should be taken to use the appropriate separator for the operating system in use.

The first group of commands creates local IPC message queues for processes that run on this host. Each process that runs locally must have a **LOCAL** entry in the system.txt file.

The next command group ensures that messages sent from any host process to the listed destination module ID (in the left-hand column) are redirected via the TCP/IP link to the SIU. Any module that is present on the SIU that the user application sends IPC messages to must have a redirection entry in this section.

The final command group, **FORK_PROCESS**, starts processes running on the host. In this example, the **rsi** process is started, along with the example **s7_log** application. The examples provided on the *User Part Development Pack* diskette (CTU and TTU) are shown at the end of the file and are commented out using the asterisk "*" comment token character.

## 3.4 Host Link Operation

Once the host software is running (gctload, rsi and an application process), it is necessary to start the Ethernet connection between the host and the SIU. This may be done with a utility, rsicmd, supplied with the host software, or by sending configuration messages to the rsi task as described in Chapter 4, "Application Programming Interface".

The application receives a notification of the status of this connection (each time the availability of the link changes) from the **local** rsi task in the form of RSI_MSG_STATUS indications.

The SIU activates all configured SS7 links when it is able to communicate with one or more hosts. If the SIU is not able to communicate with any of the hosts, the SS7 links are deactivated.

Standard TCP/IP does not provide a mechanism to detect the failure of a physical link, hence the rsi task (the software that controls the Ethernet connection between the host and SIU) running on both the host and the SIU itself send periodic heartbeat information. This ensures that both the SIU and the host will receive data packets via the Ethernet within a preset time. If this does not occur after a certain number of time periods, either the SIU or the host (or both) consider the link as failed. If all the host links fail, the SIU deactivates all of the SS7 links.

### 3.4.1 Starting the Host Software

Before the SIU host software is started, it is necessary to verify that the host computer is able to communicate with the SIU over the Ethernet using the standard TCP/IP ping utility. If this does not succeed, check the IP address configuration and the physical cabling. If this appears to be correct, refer the relevant TCP/IP manual for your operating system to diagnose the fault.

The host software is started by running **gctload**. This should be done from the host directory containing the configuration file system.txt.

For Windows, to start the system in a new virtual console (DOS console window), type:

```
start gctload -Ci0xb0
```

This establishes the IPC environment and starts the tasks listed in system.txt, including the host link manager (rsi), the utility to start-up the link to the SIU (rsicmd) and any user application processes entered against a FORK_PROCESS command.

User application processes may be started from a FORK_PROCESS command, or after gctload has started running (manually). In both cases, there must be a LOCAL definition in the system.txt file for the module_id used by the application.

### 3.4.2 Startup Order and Congestion Control

The example system.txt file included with the host software starts up the connection between the SIU and the host with a FORK_PROCESS rsicmd. Once the rsi link between the host and the SIU is established, the SIU activates the SS7 links and begins to receive SS7 messages, causing traffic to be sent from the SIU to the message queues on the host. At this point, the application should also be running to service the message queue(s). If the system is such that the application is not able to service the message queue immediately, it would be possible for the receive messages to completely fill the message queues on the host, leaving no messages free for the application to communicate in the transmit direction with the SIU. If this point is reached, the rsi process is no longer able to generate the Ethernet heartbeat messages; the host link stops. The only way to recover from such a situation is to restart the host software (by shutting down and restarting gctload (and the user application if this was not started from within system.txt).

Once an application has been developed, the functionality provided by rsicmd should be integrated into that application. This allows the application to control the socket connection, and to only activate this once the application is in a position to read from its message queue.

In order to prevent overload from occurring, gctload is usually configured to inform the rsi task when the number of messages allocated on the host (that is messages that are waiting to be read from a queue) rises above a certain value. This causes rsi to stop reading from the Ethernet socket connection to the SIU, thus giving the application a chance to empty its message queue and

reduce the number of allocated messages. When the number of messages allocated falls below a threshold, the rsi task begins to read from the SIU again. In this way, the rsi task controls the number of messages read from the SIU and prevents overload on the host. Overload and control of the socket connection by rsi should not occur if a host is running correctly; it should be possible for the application on the host to be able to service its message queue at the maximum system capacity without overload.

The command line parameters provided by gctload to configure the congestion management are:

```
-Ci<module_id>
```

This command sets the task that will be informed of overload and overload abatement. This must be set to 0xb0 for rsi to prevent overload correctly.

```
-Co<onset>
```

This command sets the percentage of messages that must be allocated before the system is overloaded. By default, this is set to 50%.

```
-Ca<abate>
```

This command sets the percentage of messages that must be allocated before the overload is considered to have passed. By default, this is set to 20%.

```
-m<number of messages>
```

This command sets the number of messages available on a host. By default this is set to 200. See Section 5.5, “gctload” on page 58 to increase the number of messages.

Hence, in the examples shown above, for UNIX operating systems, to start gctload, the following command is entered:

```
gctload –Ci0xb0 &
```

### 3.4.3 Shutting Down a Host

The software may be shut down in a controlled manner by stopping the gctload process, by either sending the kill signal (SIGTERM) in the case of UNIX systems, or by closing the console for Windows. This deletes the GCT IPC environment and any processes spawned by gctload (any program specified with a FORK_PROCESS command in the system.txt file).

Any program not started from within the system.txt file continues to run after gctload is stopped.

## 3.5 Example Application Programs

The SS7 Development Package, along with the User Part Development Package contain the files to allow the you to develop applications. These consist of makefile definitions, C header files (.h files) and libraries.

A single definitions file is supplied (for each operating system) that contains the definitions relating to the user's own development environment. This file is then included in the make files for all other processes. The user may need to modify this definitions file to ensure that correct paths etc. are set up.

The definitions file is called one of the following, depending on the operating system:

```
makdefs.mnt        (Windows)
```

```
makdefs.mlx        (Linux)
makdefs.ms2        (Solaris)
```

The following library files should be linked with the user's application code:

```
gctlib.lib         (Windows using Microsoft compiler)
gctlibb.lib        (Windows using Borland compiler)
gctlib.lib         (Linux)
gctlib.lib         (Solaris)
```

Some simple example programs are supplied to illustrate the techniques for interfacing to the protocol stack although they are not intended to show a real application. Before starting to develop an application, you should familiarize yourself with the example programs and how they are built.

The following example programs are contained on the User Part Development Package.

- **upe**
  A framework for a User Part module and contains a worked example of exchanging messages with the MTP3 module. It loops back any MTP-TRANSFER-INDICATIONS messages that it receives and reports other MTP indications to the user.
- **mtpsl**
  An example of how to send messages to MTP3 to activate and deactivate signaling links. It can be used as a command line tool for this purpose initially. It is intended that the user builds the example code into the management application.
- **ctu**
  An example of how a user application can interface with the telephony user parts, for example, ISUP.
- **ttu**
  An example of how a user application can interface with the TCAP protocol module.

A makefile is included to allow you to build the application programs. To build the program, change to the appropriate directory and enter commands similar to the following. To build **ctu** for example:

```
nmake /f ctu.mnt        (Windows)
make -f ctu.mlx         (Linux)
make -f ctu.ms2         (Solaris)
```

# 4 Application Programming Interface

## 4.1 Introduction

In addition to the protocol primitives exchanged between the user application and SIU, the SIU also supports a number of management primitives to allow for event reporting and system control. This interface supports requests by the user to restart a board, activate and deactivate a signaling link, block and un-block a circuit group and request status information. It allows the SIU to report PCM events, level 2 state changes and level 3 events to the user's management module. The following messages are used over this interface:

| Type value | Mnemonic | Description |
|---|---|---|
| 0x0003 | MGT_MSG_TRACE_EV | Trace Event Indication |
| 0x7f80 | RSI_MSG_CONFIG | RSI Link Configuration Request |
| 0x7f81 | RSI_MSG_UPLINK | RSI Link Activate Request |
| 0x0f83 | RSI_MSG_LNK_STATUS | RSI Link Status Indication |
| 0x0201 | MGT_MSG_SS7_STATE | SS7 Level 2 Status Indication |
| 0x0301 | MTP_MSG_MTP_EVENT | MTP Protocol Event Indication |
| 0x0e01 | MVD_MSG_LIU_STATUS | PCM Trunk Status Indication |
| 0x0f0d | API_MSG_SIU_STATUS | SIU Status Indication |
| 0x070e | API_MSG_USER_EVENT | User Event Indication |
| 0x7f0f | API_MSG_COMMAND | User Command Request |
| 0x7718 | CAL_MSG_HEARTBEAT | Check Heartbeat |

Details of these messages are described later in this chapter.

## 4.2 Messages and Message Queues

The Application Programming Interface (API) is based on *messages* and *message queues*. Each application receives data from the SIU by reading from its own message queue, and sends data to an SIU by sending a message to the queue of another process or task running on the SIU.

The Inter Process Communication (IPC) is handled by the following set of functions provided by the gctlib library:

- **GCT_receive( )**
- **GCT_grab( )**
- **GCT_send( )**
- **GCT_set_instance( )**
- **getm( )**
- **relm( )**

These functions operate on a message structure, defined in the C programming language as **MSG**. These functions and the structure of a MSG are described in the *Software Environment Programmer's Manual.*

The messages that may be used on the SIU API are defined in "API Commands" on page 35 and also in the appropriate protocol Programmer's Manual.

Example application programs, supplied as part of the *User Part Development Package* together with the SIU, demonstrate the operation of the API.

## 4.3 Sending a Message to an SIU

An application wishing to send a message to the SIU must first allocate a message structure (MSG) using the **getm( )** function. The application should write the message parameters to this structure according to the MSG definition tables in this manual or the relevant protocol Programmer's Manual. The message is directed to a particular SIU using the **GCT_set_instance( )** library function. SIUA is instance 0 and SIUB is instance 1. Once the message parameters have been set the application calls **GCT_send( )** to send the message to the destination process (running on the SIU). If the **GCT_send( )** function fails to send the message, the application must release the message back to the system using the **relm( )** function. This happens only if the system has been configured incorrectly.

## 4.4 Receiving Messages From an SIU

The SIU software writes any messages addressed to the application to the application's message queue. These may be read by the application using either the **GCT_receive( )** or **GCT_grab( )** functions (depending on whether it wishes to block or not if no messages are available). The MSG parameters may then be extracted and processed.

When the application has finished processing the message it must be released back to the system using the **relm( )** function. In this way, it is ensured that each message is always released back to the system.

## 4.5 Requesting a Confirmation

Under certain circumstances, the application needs to know that the contents of a message received by an SIU was recognized as being valid, or that the requested operation has been completed. This is achieved by requesting a *Confirmation* when the message is sent to the SIU.

The SIU confirms that a message received from the application has been recognized and processed correctly by sending the message back to the application (via the application's message queue). The message is modified in two ways before being sent back to the application to identify the message as a confirmation:

- The **dst** field of the MSG header is set to the module_id of the application process
- Bit 14 of the MSG header **type** field is set to zero. For example, the SIU would confirm a message with type value 0x7123 by setting the type value to 0x3123.

A confirmation is requested by the application by setting one of the bits in the 16-bit **rsp_req** parameter of the MSG header. The bit number that should be used by the application for this purpose is identified by the least significant nibble (4-bits) of the application's own **module_id**. For example, if the application was assigned **module_id** 0x3d, a confirmation is requested by setting bit 13 of the **rsp_req** parameter, value 0x2000 (bit counting is zero-based).

The confirmation message contains a **status** value in the MSG header. For command requests, a status of zero normally indicates success.

Each message specification table details whether a confirmation may be (or must be) requested.

### 4.5.1 Congestion Management

When the host software is first run, a specified number (200 by default) of messages are allocated from host system resources which are then available for allocation for sending messages to the SIU by the application and the components of the host software package.

The function of the application is to read messages from its own input queue, (received from the TCP/IP connection with the SIU), extract the information from these messages then release the original message structure back to the system. Hence, under normal operating conditions, the host application works to ensure that its queue is almost empty, and that all the messages are available.

The message handling functions monitor the number of free messages available (messages that are not allocated). If this number falls below a pre-set threshold, the host is said to be in an overloaded or congested state, and if configured correctly, the host software stops reading from the TCP/IP socket. This provides a time period for the application to read messages stored in its input queue, to process then release these messages. This increases the number of free messages available, ultimately removing the congested state and enabling the host software to begin reading from the TCP/IP socket connection.

## 4.6 API Commands

The SIU may be configured to issue management indications to any single host using an API_MSG_COMMAND with **cmd_type** 15 (see Section 4.6.1, “API_MSG_COMMAND” on page 36), allowing these messages to be redirected following failure of the host that is currently processing this information. On power-up, the SIU issues management indications to host 0.

The Dialogic® DSI SS7G41 Signaling Servers support the commands described in the following subsections.

### 4.6.1 API_MSG_COMMAND – User Command Request

#### Synopsis

The API_MSG_COMMAND message is used to request execution of a user command.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | API_MSG_COMMAND (0x7f0f) |
| id | 0 |
| src | Sending module_id |
| dst | SIU_MGT_TASK_ID (0xdf) |
| rsp_req | Should be used to request a confirmation |
| hclass | 0 |
| status | 0 |
| err_info | 0 |
| len | 8 |

| Parameter Area | | |
|---|---|---|
| **Offset** | **Size** | **Name** |
| 0 | 2 | cmd_type |
| 2 | 2 | id |
| 4 | 4 | result |

#### Description

The API_MSG_COMMAND message is used by the application to request execution of a management command on the SIU. You should always request a confirmation message and should note that only one command can be executed at a time.

#### Confirmation Message

The module sending the message should always request that a confirmation is returned by the SIU when the message has been processed. This is achieved by setting the sending layer's bit in the **rsp_req** field, which causes a confirmation message of the same format to be returned. The **status** field in this message is zero on success or an error code as shown below otherwise.

***Note:*** In a dual resilient configuration, the application must use the **GCT_set_instance( )** library function to address this message to the correct SIU. SIUA is instance 0, SIUB instance 1.

| Value | Description |
|---|---|
| 1, 5 | Unable to process the command due to an internal error |
| 2 | Unrecognized command |
| 3 | Command is unacceptable in the current state (for example; may need to deactivate link first) |
| 4 | No resources (only one command can execute at a time) |
| 6 | Range error in supplied parameters |
| **NOTE:** All other values are reserved. | |

### Parameters

The API_MSG_COMMAND message includes the following parameters:

- **cmd_type**
  Command type that used in conjunction with the **id** instructs the SIU to perform a command as shown in the following table:

| cmd_type | id | Description |
|---|---|---|
| 1 | <bpos> | Restart a signaling board.<br>**Note**: All signaling links on the board must first be deactivated. |
| 2 | <link_id> | Activate a signaling link |
| 3 | <link_id> | Deactivate a signaling link |
| 4 | <link_id> | Read level 2 link state |
| 5 | <port_id> | Read PCM trunk status<br>As defined/configured by you in the LIU_CONFIG config.txt command. |
| 8 | <gid> | Activate group |
| 9 | <gid> | Deactivate group |
| 10 (0x0a) | <link_id> | Read level 3 state |
| 11 (0x0b) | <bpos> | Read board state |
| 12 (0x0c) | 0 | Read hardware alarms |
| 13 (0x0d) | 0 | Read inter-SIU Ethernet link state |
| 14 (0x0e) | <host_id> | Read host-SIU link state |
| 15 (0x0f) | <host_id> | Nominate a primary host to receive management messages |
| 16 (0x10) | 0 | Read Congestion Status |
| 17 (0x11) | 0 | Restart Unit |
| 18 (0x12) | 0 | SIU restart query; provides a count of the number of times the SIU has restarted. Once the host link becomes active, this command can be issued to determine whether or not the SIU was restarted while the host link was down. If the SIU had restarted, the host may want to initiate dynamic configuration (for example, for circuit groups). The result is the number of times the SIU has restarted. |
| 19 (0x13) | <host_id> | Activate an optional second host to receive management messages |
| 20 (0x14) | <host_id> | Deactivate an optional second host from receiving management messages |
| 21 (0x15) | 0 | System reference query. The result is the integer system reference (SYSREF) that identifies the unit. |
| 22 (0x16) | <link_id> | Activate a previously deactivated M3UA link. |
| 23 (0x17) | <link_id> | Deactivate an M3UA link. |
| 24 (0x18) | <link_id> | Return Layer 2 status:<br>• 0 Failed<br>• 1 Closed<br>• 2 Cookie wait<br>• 3 Cookie echoed<br>• 4 Established<br>• 5 Pending Shutdown<br>• 6 Sent Shutdown<br>• 7 Received Shutdown<br>• 8 Acknowledge Shutdown |
| 25 (0x19) | <link_id> | Requests a SS7_MSG_R_STATS message to be sent to the requesting module ID with statistics for the specified link. |
| 26 (0x1a) | <port_id> | Requests a LIU_MSG_R_STATS message to be sent to the requesting module ID with statistics for the specified port. |
| 43 0x2b | 0 | Traffic congestion |
| 44 (0x2c) | 0 | Traffic enforcement |
| 45 (0x2d) | 0 | Clearing traffic congestion and enforcement if active |

- **id**
  For <bpos>, <link_id> and <gid>, this is the index of the board, signaling link or group affected by the command, as defined in config.txt. <port_id> identifies a PCM port on a signaling board.
- **result**
  Additional status information returned by some commands is as follows:
  — Level 2 link state
  The value returned in the result field is the link state value as defined for the SS7 Level 2 State Indication message (MGT_MSG_SS7_STATE).
  — PCM trunk status
  The value returned in the result field is a bit mapped field with the following meanings:

| Bit | Mnemonic | Description |
|---|---|---|
| 0 | PCM_SF_PCM_LOSS | Loss of PCM detected |
| 1 | PCM_SF_AIS | AIS (all ones) detected |
| 2 | PCM_SF_SYNC_LOSS | Frame sync loss |
| 3 | PCM_SF_REM_ALARM | Remote alarm present |

  — Alarms
  The value returned in the result field is a bit mapped field with the following meanings:

| Bit | Name | Description |
|---|---|---|
| 0 | MGTSF_FAN_WARN | Fan warning |
| 1 | MGTSF_FAN_FAIL | Fan failure |
| 2 | reserved | reserved |
| 3 | reserved | reserved |
| 4 | MGTSF_TEMP | Temperature is outside preset threshold |
| 5 | MGTSF_PSU1_FAIL | Power supply failure |
| 6 | MGTSF_PSU2_FAIL | Power supply failure |
| 7 | reserved | reserved |
| 8 | MGTSF_PARSE | Syntax errors found in config.txt protocol configuration file |
| 9 | MGTSF_CONFIG | Protocol configuration failed |
| 10 | MSTSF_CONG | System congestion |

  — Board status
  The value returned in the result field indicates the corresponding board state as indicated in the following table:

| Value | Mnemonic | Description |
|---|---|---|
| 0 | SIMBS_INACTIVE | Board is inactive |
| 1 | SIMBS_RESETTING | Board is resetting |
| 2 | SIMBS_ACTIVE | Board is active |
| 3 | SIMBS_FAILED | Board has failed |

  — Level 3 status
  The value returned indicates the level 3 state according to the following table:

| Value | Mnemonic | Description |
|---|---|---|
| 0 | MGTL3S_UNAVAILABLE | Destination available |
| 1 | MGTL3S_AVAILABLE | Destination not available |

— Inter SIU Ethernet status

The value returned in the result field is the link state value as defined for the RSI link state Indication message (RSI_MSG_LNK_STATUS).

— Host-SIU and Inter SIU Ethernet status

The value returned in the result field is the link state value as defined for the RSI link state Indication message (RSI_MSG_LNK_STATUS). Bit 8 of the result is set to 1 to indicate that the host indicated by host_id is currently receiving management indications.

### 4.6.2 RSI_MSG_CONFIG – RSI Link Configuration Request

#### Synopsis

The RSI_MSG_CONFIG message is issued to the rsi to configure a link between a host and an SIU.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | RSI_MSG_CONFIG (0x7f80) |
| id | siu_id |
| src | Sending module ID |
| dst | RSI module ID (0xb0) |
| rsp_req | Used to request a confirmation |
| hclass | 0 |
| status | 0 |
| err_info | 0 |
| len | 68 |

| Parameter Area | | |
|---|---|---|
| **Offset** | **Size** | **Name** |
| 0 | 1 | reserved - must be set to zero |
| 1 | 1 | conc_id |
| 2 | 2 | flags |
| 4 | 2 | local_port |
| 6 | 2 | remote_port |
| 8 | 20 | local_addr |
| 28 | 20 | remote_addr |
| 48 | 20 | reserved - must be set to zero |

#### Description

The RSI_MSG_CONFIG message is used by the host application to configure a link to a single SIU. The requested link is configured in the idle (inactive) state.

#### Confirmation Message

The module sending the message may request that a confirmation is returned when the message has been processed by setting the sending layer's bit in the **rsp_req** field which causes a confirmation message of the same format to be returned. The **status** field in this message is zero on success.

#### Parameters

The RSI_MSG_CONFIG message includes the following parameters:

- **siu_id**
  Identifies the SIU that the link connects to. 0 indicates SIUA, 1 indicates SIUB.
- **conc_id**
  Specifies a module ID that will receive RSI link status indications. This module should exist on the host, such that when these status messages are issued by rsi, they are received and then released by this module.
- **flags**
  A 16-bit value specifying additional link configuration. All bits must be set to zero.
- **local_port**
  This field should be set to zero.

- **rem_port**
  Specifies the TCP/IP socket port that will be used to communicate with the SIU. Each host uses a different port number, starting at 9000 for the first host (ID 0) and incrementing by one for each additional host. Hence host ID 4 uses port 9004. If there is only one host, port 9000 should be used.
- **local_addr**
  This field should be set to zero.
- **rem_addr**
  Specifies the IP address of the SIU that the connection is to be made with, as defined in the SIU configuration. This should be entered as ASCII characters (for example to specify the IP address 123.124.125.126 the parameter should be 3132332e3132342e3132352e313236).

### 4.6.3 RSI_MSG_UPLINK – RSI Link Activate Request

#### Synopsis

The RSI_MSG_UPLINK message is sent by the application to the rsi to activate a link to an SIU.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | RSI_MSG_UPLINK (0x7f81) |
| id | siu_id |
| src | Sending module ID |
| dst | RSI module ID (0xb0) |
| rsp_req | Used to request a confirmation |
| hclass | 0 |
| status | 0 |
| err_info | 0 |
| len | 0 |

#### Description

The RSI_MSG_UPLINK message is issued by the host application to activate a previously configured TCP/IP connection to an SIU. The rsi process attempts to establish the link on receipt of this message. RSI Link Status Indications are issued to the host process identified by conc_id detailing the availability of the connection to the SIU.

#### Confirmation Message

The module sending the message may request that a confirmation is returned when the message has been processed by setting the sending layer's bit in the **rsp_req** field which causes a confirmation message of the same format to be returned. The **status** field in this message is zero on success.

#### Parameters

The RSI_MSG_UPLINK message includes the following parameter:

- **siu_id**
  Identifies the SIU to which the TCP/IP connection should be activated. 0 indicates SIUA, 1 indicates SIUB.

### 4.6.4 RSI_MSG_LNK_STATUS – RSI Link Status Indication

#### Synopsis

The RSI_MSG_LNK_STATUS message is issued by the rsi to notify the concerned host process (conc_id) of state changes in the link between the host and the SIU.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | RSI_MSG_LNK_STATUS (0x0f83) |
| id | siu_id |
| src | RSI module ID (0xb0) |
| dst | Concerned ID (See below) |
| rsp_req | 0 |
| hclass | 0 |
| status | LINK STATE (see below) |
| err_info | 0 |
| len | 0 |

#### Description

The RSI_MSG_LNK_STATUS message is issued by the rsi to a process identified by the **conc_id** (concerned ID) value specified when the RSI link was configured.

#### Parameters

The RSI_MSG_LNK_STATUS message includes the following parameter:

- **siu_id**
  Identifies the SIU to which the link has failed, as entered on the rsicmd command line.
- **LINK_STATE**
  The status value specifies the state of the link as follows:

| Value | Link state |
|---|---|
| 2 | Link to SIU lost |
| 1 | Link to SIU (re)established |

### 4.6.5 MVD_MSG_LIU_STATUS – PCM Trunk Status Indication

#### Synopsis

The MVD_MSG_LIU_STATUS message is used by the SIU to notify of changes of state on the PCM trunk.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | MVD_MSG_LIU_STATUS (0x0e01) |
| id | pcm_id |
| src | MVD_TASK_ID (0x10) |
| dst | REM_API_ID |
| rsp_req | 0 |
| hclass | 0 |
| status | LIU Status (see below) |
| err_info | 0 |
| len | 0 |

#### Description

The MVD_MSG_LIU_STATUS message is used by the SIU for every change of state on the PCM trunk interface. The **id** field indicates the identity of the PCM trunk to which the message refers.

The **LIU Status** contained in the **status** field of the message indicates the type of event. Possible values are listed in the following table.

| Value | Description |
|---|---|
| 10 | Frame sync loss |
| 11 | Frame sync OK |
| 12 | AIS detected |
| 13 | AIS cleared |
| 14 | Remote alarm |
| 15 | Remote alarm cleared |
| 20 | PCM loss |
| 21 | PCM restored |
| 22 | Frame Slip |

### 4.6.6 MGT_MSG_SS7_STATE – SS7 Level 2 Status Indication

**Synopsis**

The MGT_MSG_SS7_STATE message is used by the SIU to notify of changes of state of level 2 link state control.

**Format**

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | MGT_MSG_SS7_STATE (0x0201) |
| id | link_id |
| src | SS7_TASK_ID (0x71) |
| dst | REM_API_ID |
| rsp_req | 0 |
| hclass | 0 |
| status | LINK STATE (see below) |
| err_info | Reserved for future use. |
| len | 0 |

**Description**

The MGT_MSG_SS7_STATE message is issued by the SIU every time a change of state takes place at level 2. It is intended only for diagnostic use by system management. The level 2 link state control state machine is defined in Q.703.

The **LINK STATE** in the **status** field in the message header is used to indicate the state that has just been entered. It is coded as follows:

| Value | Mnemonic | State |
|---|---|---|
| 1 | S7S_IN_SERVICE | In Service |
| 2 | S7S_OUT_SERVICE | Out of Service |
| 3 | S7S_INIT_ALIGN | Initial Alignment |
| 4 | S7S_ALIGN_NOT_RDY | Aligned, Not Ready |
| 5 | S7S_ALIGN_READY | Aligned, Ready |
| 6 | S7S_PROC_OUTAGE | Processor Outage |

### 4.6.7 MTP_MSG_MTP_EVENT – MTP Protocol Event Indication

#### Synopsis

The MTP_MSG_MTP_EVENT message is used by the SIU to notify management of protocol events within the MTP.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | MTP_MSG_MTP_EVENT (0x0301) |
| id | Link_id |
| src | MTP_TASK_ID (0x22) |
| dst | REM_API_ID |
| rsp_req | 0 |
| hclass | 0 |
| status | Event code (see below) |
| err_info | 0 |
| len | 1, 2 or 4 (see below) |

#### Description

The MTP_MSG_MTP_EVENT message is issued by the SIU to indicate MTP events to the host management module. The **id** field contains the link number to which the event refers.

The **Event Code** contained in the **status** field of the message indicates the type of event. The EVENT_CODE coding and the meaning of the event specific parameters are given in the following table.

| Value | Mnemonic | Parameter | Description |
|---|---|---|---|
| 1 | MTPEV_CO | link | Changeover (link failure) |
| 2 | MTPEV_CB | link | Changeback (link restored) |
| 3 | MTPEV_REST | link | Restoration commenced |
| 4 | MTPEV_RPO | link | Remote processor outage |
| 5 | MTPEV_RPO_CLR | link | Remote processor outage cleared |
| 6 | MTPEV_CONG | link | Signaling link congestion |
| 7 | MTPEV_CONG_CLR | link | Congestion cleared |

**NOTES:**
1. link is indicated as (linkset_id * 256) + link_ref, (size = 2)
2. link set is indicated as linkset_id, (size = 1)
3. point code is a 4-byte value, (size = 4)

### 4.6.8 API_MSG_USER_EVENT – User Event Indication

#### Synopsis

The API_MSG_USER_EVENT message is issued to inform the nominated host of events within the SIU.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | API_MSG_USER_EVENT (0x0f0e) |
| id | Event ID (See below) |
| src | SIU_MGT_TASK_ID |
| dst | See below |
| rsp_req | 0 |
| hclass | 0 |
| status | Event code |
| err_info | 0 |
| len | 0 |

#### Description

This message is issued to inform the nominated host of events within the SIU.

The **Event Code** contained in the **status** field of the message indicates the type of event. Possible values are listed in the following table.

| Event | Error code | Event ID | Description |
|---|---|---|---|
| Circuit group conflict detected | 1 | <gid> | Sent to the module/instance handling the affected circuit group, as specified in config.txt, to indicate that a circuit group has become active on both SIUs (in a dual resilient configuration). The circuit group should be deactivated on the “non-preferred” unit using an API_MSG_COMMAND message. |

### 4.6.9 API_MSG_SIU_STATUS – SIU Status Indication

#### Synopsis

The API_MSG_SIU_STATUS message is issued to the nominated host to inform the application of a change in alarm status within the SIU.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | API_MSG_SIU_STATUS (0x0f0d) |
| id | See table below |
| src | SIU_MGT_TASK_ID (0xdf) |
| dst | REM_API_ID (0xef) |
| rsp_req | 0 |
| hclass | 0 |
| status | SIU status event (see below) |
| err_info | 0 |
| len | 0 |

#### Description

This message is issued to the nominated host to inform the application of a change in alarm status within the SIU.

The **SIU status event** in the **status** field of this message indicates the event being reported as shown in the following table. The **id** field is used by certain events to provide additional information.

| Value | Mnemonic | Event | id |
|---|---|---|---|
| 1A | SIUS_FAN_FAIL | Fan failure | 0 |
| 1B | SIUS_FAN_OK | Fan recovered | 0 |
| 1C | SIUS_BOARD_FAIL | Board Failure | BPOS |
| 1D | SIUS_BOARD_OK | Board recovered | BPOS |
| 1E | SIUS_HOST_FAIL | Host link failure | Host ID |
| 1F | SIUS_HOST_OK | Host link recovered | Host ID |
| 20 | SIUS_SIUL_FAIL | Inter SIU Ethernet link failure | 0 |
| 21 | SIUS_SUIL_OK | Inter SIU Ethernet link recovered | 0 |
| 22 | SIUS_CONGESTION | SIU is congested | 0 |
| 23 | SIUS_NO_CONGESTION | SIU congestion has cleared | 0 |
| 24 | SIUS_PSUx _FAIL | Power supply failure | PSU ID |
| 25 | SIUS_PSUx_OK | Power supply recovered | PSU ID |
| 26 | SIUS_PRO_OVER_TEMP | Processor over temp | CPU ID |
| 27 | SIUS_PRO_TEMP_OK | Processor temp recovered | CPU ID |
| 2A | SIUS_FAN_WARN | Fan failure | 0 |
| 2E | SIUS_DRIVE_UNAVAIL | Disk drive unavailable | Drive ID |
| 2F | SIUS_DRIVE_AVAIL | Disk drive available | Drive ID |

### 4.6.10 MGT_MSG_TRACE_EV – Trace Event Indication

#### Synopsis

Used by a protocol layer to report trace primitives as event indications to neighboring protocol layers.

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | MGT_MSG_TRACE_EV (0x0003) |
| id | 0 |
| src | Module_id that originated trace message |
| dst | Management module id (mgmt_id) |
| rsp_req | 0 |
| hclass | 0 |
| status | 0 |
| err_info | 0 |
| len | 18 + length of traced data |

| Parameter Area | | |
|---|---|---|
| **Offset** | **Size** | **Name** |
| 0 | 1 | source module id |
| 1 | 1 | destination module id |
| 2 | 2 | id |
| 4 | 2 | type |
| 6 | 2 | status |
| 8 | 4 | timestamp |
| 12 | 4 | pointer to the message being traced |
| 16 | 2 | data length |
| 18 | 0 to 280 | data – Data taken from the MSG parameter area. |

#### Description

The protocol software running on the SIU may be configured to report to primitives exchanged with the protocol layer above and below. This is useful for trace and debug purposes. Tracing is enabled by specifying individual bits in trace masks in the xxx_TRACE configuration commands. The traced primitives are reported as event indications as shown below.

#### Parameters

The MGT_MSG_TRACE_EV message includes the following parameter:

- **source module id**
  The source module ID of the traced message.
- **destination module id**
  The source module ID of the traced message.
- **id**
  The id parameter of the traced message.
- **type**
  The type parameter of the traced message.
- **status**
  The status parameter of the traced message.
- **timestamp**
  The timestamp parameter of the traced message.
- **pointer**
  A pointer to the message being traced.

- **data length**
  The length of the parameter area of the traced message.
- **data**
  The data taken from the MSG parameter area of the traced message.

### 4.6.11 CAL_MSG_HEARTBEAT – Check Heartbeat

#### Synopsis

This message is issued by the ISUP module as a heartbeat to determine availability of a particular user application as identified by module_id and instance (or host_id).

#### Format

| Message Header | |
|---|---|
| **Field Name** | **Meaning** |
| type | CAL_MSG_HEARTBEAT (0x7718) |
| id | 0 |
| src | ISUP module ID |
| dst | User Application module ID |
| rsp_req | Sending layer's bit must be set |
| hclass | 0 |
| status | 0 |
| err_info | 0 |
| len | 64 |

| Parameter Area | | |
|---|---|---|
| **Offset** | **Size** | **Name** |
| 0 | 2 | user instance id |
| 2 | 2 | state |
| 4 | 2 | flags |
| 6 | 58 | Reserved for future use - set to zero |

#### Description

It is possible to configure ISUP to detect failed (or inactive) SIU hosts and initiate circuit group blocking to the network. This ensures that the network does not attempt to initiate calls on circuits for which there is no active application and calls would consequently fail.

The use of this feature requires the user application to respond to this message that is periodically issued by the ISUP module. In the event that no response is received within a predetermined time, the ISUP module initiates hardware circuit group blocking to the network.

#### Parameters

The CAL_MSG_HEARTBEAT message includes the following parameters:

- **user instance id**
  The User instance (or SIU host_id)
- **state**
  The status of the user application

| Value | Meaning | Description |
|---|---|---|
| 0 | Unconfigured | No circuit groups have been configured. |
| 1 | Down | The user application is unavailable and out of service. The circuit groups have been hardware blocked. |
| 2 | Up | The user application is available and in service. |

- **flags**
  Set by the ISUP module

| Bit | Mnemonic | Description |
|---|---|---|
| 0 | UIHB_FLAGS_CGRPS_BLOCKED | If set in heartbeat messages from the ISUP module, this indicates to the user, that the ISUP circuit group(s) have been blocked. |

# 5 Host Utility and Command Syntax

This chapter describes in more detail the host utilities identified in Section 3.2, "Contents of the SS7 Development Package" on page 27. The utilities include:

- rsi
- rsicmd
- s7_log
- s7_play
- gctload
- tim
- tick

These are the utilities most applicable to the SIU. The complete development packages contains additional utilities discussed in detail in the *Dialogic DSI Software Environment Programmer's Manual.*

## 5.1 rsi

The **rsi** process manages the connection between the host and each SIU. It takes several command line parameters and is normally spawned by an entry in the host's system.txt file. The command line syntax is shown below:

```
rsi -p<pipe> -r<link_process> -l<link_selection>
```

where,

- <**pipe**>
  Specifies the pipe used for communication between rsi and rsi_lnk. If not specified, rsi attempts to use /tmp/pipe. This parameter is not required under Windows.
- <**link_process**>
  Specifies the location of the rsi_lnk process binary. If not specified, rsi assumes that the rsi_lnk binary is located in the current directory.
- <**link_selection**>
  Specifies the routing algorithm used by rsi to send a message (MSG) from a user application running on the host to an SIU. The following routing algorithms are supported:

| Value | SIU Selection Algorithm |
|---|---|
| 1 | Messages are routed to SIUA or SIUB depending on the setting of the message instance. SIUA is instance 0, SIUB is instance 1. The **GCT_set_instance( )** C-library function should be used to set the instance value for each MSG sent to either SIUA or SIUB. |
| 2 | Send all messages to SIUA. |

The following is an example rsi entry in a system.txt file on a Linux system:

```
FORK_PROCESS../BIN/rsi -p/tmp/rsilnk -r../BIN/rsi_lnk -l1
```

For Windows, the equivalent entry is:

```
FORK_PROCESS.\rsi.exe -r .\rsi_lnk -l1
```

## 5.2 rsicmd

The **rsicmd** command starts the Ethernet link between a host and an SIU. The syntax is common to all operating systems and is shown below:

```
rsicmd <siu_id> <conc_id> <link_type> <rem_addr> <rem_port>
```

- **<siu_id>**
  The local logical identifier to identify each link from a single host to each SIU as described in the following table:

| siu_id | Link |
|---|---|
| 0 | Between host and SIUA |
| 1 | Between host and SIUB |

  This parameter sets the instance value that must be used by the application in the call to the **GCT_set_instance( )** library function when directing an API message to either SIUA or SIUB in a dual resilient configuration.

- **<conc_id>**
  Specifies a module ID that will receive a message whenever the rsi link fails. This module should exist within the system, such that when these status messages are issued by rsi, they are received and then released by this module.

  ***Note:*** In a DOS system, the application receives all messages issued by the SIU system regardless of the destination module ID.

- **<link_type>**
  Must be set to 0.

- **<rem_addr>**
  Specifies the IP address of the SIU, as specified in the SIU configuration.

- **<rem_port>**
  Specifies the TCP/IP socket port that is used to communicate with the SIU. Each host uses a different port number, starting at 9000 for the first host (ID 0) and incrementing by one for each additional host. Hence host ID 4 uses port 9004. If there is only one host, port 9000 should be used.
  For example, to start a link to SIUA with an IP address 123.124.125.126 as host 0, nominating a module whose ID is 0xef to receive RSI status information, the command line is:

```
rsicmd 0 0xef 0 123.124.125.126 9000
```

  rsicmd may be run from system.txt by adding the appropriate FORK_PROCESS commands, hence to connect to both SIUA and SIUB as host ID 3, the following commands would be entered in the system.txt file on the host:

```
FORK_PROCESS  ..\RUN\rsicmd 0 0xef 0 123.234.345.456 9003
FORK_PROCESS  ..\RUN\rsicmd 1 0xef 0 123.234.345.456 9003
```

## 5.3 s7_log

### Description

The s7_log utility is a console application program that receives messages and displays them as text on the host console. Maintenance and status events are interpreted as text; other messages are typically displayed in hexadecimal format. The s7_log utility can optionally print the date and time of when a message is received by the utility.

### Syntax

```
s7_log [-m<module_id>] [-o<options>] [-f<filename>] [-t[t|d]]
```

### Command Line Options

The s7_log utility supports the following command line options:

- **-m<module_id>**
  Specifies the unique module identifier assigned to s7_log for the inter-process communication (IPC) environment. Any message sent to this module ID is displayed by the s7_log utility as text on the console. The module ID may be entered in decimal or hexadecimal (prefixed by "0x") format. If the module ID is not specified, s7_log uses a module ID of 0xef. The module ID that is assigned to s7_log must have a corresponding LOCAL entry in the host's system.txt file and must not be in use by any other process on the host.
- **-o<options>**
  A 16-bit value that specifies the type of message reporting that occurs. If not specified, a value of 0xaf0d is used. Each bit that is set to 1 enables reporting of a particular message group or parameter field as described in the following table:

| Bit | Function |
|---|---|
| 0 | Enable text interpretation of all recognized messages. |
| 1 | Display ALL received messages (including those interpreted as text) as hexadecimal. |
| 2 | Decode and display Management trace messages. |
| 3 | Decode and display Management Trace Event 'time stamp' field. |
| 4 | Decode message header **src** and **dst** fields as text if recognized. |
| 5 | Not used. Must be set to zero. |
| 6 | Not used. Must be set to zero. |
| 7 | Not used. Must be set to zero. |
| 8 | Display message **type** field. |
| 9 | Display message **id** field. |
| 10 | Display message **src** field. |
| 11 | Display message **dst** field. |
| 12 | Display message **rsp_req** field. |
| 13 | Display message **status** field. |
| 14 | Display message **err_info** field. |
| 15 | Display message **parameter** field. |

- **-f<filename>**
  Optionally specifies a file to which all screen output is written. If the specified file does not exist, it is created. If the specified file already exists, it is overwritten. The data is stored in the file in ASCII format.
- **-t[t|d]**
  Specifies the format of timestamp values derived from the host clock. The timestamp information is printed after the "S7L:" label in the log. The format options are:
  - — -tt specifies short timestamp format, that is, the time only
  - — -td specifies full timestamp format, that is, the date and time

***Note:*** Since the timestamps related to this option are derived from the host clock, values can be affected by host loading.

### Example

To run s7_log as module ID 0xef and enable all tracing options, the command line is:

```
s7_log -m0xef -o0xff1f
```

### Sample Output

Typical output from s7_log is as follows:

```
S7_LOG: Message monitor Copyright (C) Dialogic Corporation 1998-2007.  All Rights Reserved.
=====================================================
S7_log : mod ID=0xef, options=0xaf0d
S7L:I0000 RSI_MSG_LNK_STATUS : Link 0 now down
S7L:I0000 RSI_MSG_LNK_STATUS : Link 0 now up
S7L:I0001 RSI_MSG_LNK_STATUS : Link 0 now down
S7L:I0001 RSI_MSG_LNK_STATUS : Link 0 now up
S7L:I0000 LIU Status : id=0 IN SYNC
S7L:I0000 LIU Status : id=0 PCM OK
S7L:I0000 Level 2 State : id=0 INITIAL ALIGNMENT
S7L:I0000 LIU Status : id=0 IN SYNC
S7L:I0000 LIU Status : id=0 PCM OK
S7L:I0001 Level 2 State : id=0 INITIAL ALIGNMENT
S7L:I0000 Level 2 State : id=0 ALIGNED READY
S7L:I0000 Level 2 State : id=0 IN SERVICE
S7L:I0000 MTP Event : linkset_id/link_ref=0000 Changeback
S7L:I0000 MTP Resume, dpc=00000001
S7L:I0000 M t0708 i0000 f23 d1d s00 p000000007fff
S7L:I0000 M t0708 i0000 f23 d1d s00 p00007fff0000
```

Each line of text that corresponds to a received message is prefixed by S7L:I<instance>, the instance being recovered from the received message.

Messages that are not interpreted as text are displayed in hexadecimal format as follows:

```
M t<type> i<id> f<src> d<dst> s<status> e<err_info> p<param>
```

Each field contains the value of the corresponding message field in hexadecimal format.

## 5.4 s7_play

### Description

The s7_play utility is a console application that reads commands from an ASCII text file then executes the commands. Each command can specify either:

- a message to be sent to a destination process
- a delay to apply before the next command is executed

### Syntax

```
s7_play -m<module_id> -f<filename>
```

### Command Line Options

The s7_play utility supports the following command line options:

- **-m<module_id>**
  Specifies the unique module ID that is assigned to s7_play for the inter process communication (IPC) environment. Any message that is sent to this module ID is displayed by the s7_log utility as text on the host console. The module ID may be entered in decimal or hexadecimal (prefixed by “0x”) format. If the module ID is not specified, the s7_play utility uses a module ID of 0xef. The module ID assigned to the s7_play utility must have a corresponding LOCAL entry in the host’s system.txt file and must not be in use by any other process on the host.
- **-f<filename>**
  Specifies the text file that contains the commands to be executed by the s7_play utility.

### Example

To run s7_play with module ID 0x3d and accept commands from a file called cmd.txt, the command is:

```
s7_play -m0x3d -fcmd.txt
```

### Text File Format

Each line in the text file must begin with one of the command specifiers in the following table:

| Character | Function |
|---|---|
| M | Send a message |
| D | Delay |
| * | Ignore (comment line) |

The delay function takes a single parameter specifying the delay in either milliseconds (-m) or seconds (-s). Some examples:

```
D-s0001 * Delay for 1 second
D-m0001 * Delay for 1 millisecond
```

***Note:*** The delay value may be in the range 0000 to FFFF.

The send message function allows the fields of the message to be specified in the following format:

```
M-I<inst>-t<type>-i<id>-f<src>-d<dst>-r<rsp_req>-e<err_info>-s<status>-p<param>
```

The meaning of the various options is shown in the following table:

| Field Identifier | Length (in characters) | Message Field |
|---|---|---|
| I | 2 | Instance |
| t | 4 | type |
| i | 4 | id |
| f | 2 | src |
| d | 2 | dst |
| r | 4 | rsp_req |
| e | 8 | err_info |
| s | 2 | status |
| p | 2 to 640 (variable) | param |

Each field identifier is optional and causes the corresponding message field to be set to zero if not present. All values are entered in hexadecimal format. For example:

```
M-tc701-i0000-f1d-d23-s00-p0000ffffffff
```

The following command file sends a reset circuit group message to the first ISUP group, waits for 5 seconds, then sends a reset group message for group 1.

```
*
* Example s7_play command file
*
M-tc701-i0000-f1d-d23-s00-p0000ffffffff
*
D-s0005
*
M-tc701-i0001-f1d-d23-s00-p0000ffffffff
```

## 5.5 gctload

### Description

gctload is a task that initializes the host system environment and starts up all other processes (such as ssd), deriving the process and message queue configuration from a text file. For further details of the operation of gctload refer to the *Software Environment Programmer's Manual*. The gctload task derives its configuration from a text file, typically called system.txt.

The gctload task can be run on an active system to provide tracing information that indicates the system state (-t1, -t2 flags) and it can also be used to terminate an active system (-x flag). Users of Windows-based systems who wish to run gctload as a service should refer to Section 5.5.3, “Running gctload as a Service” on page 60.

### Syntax

```
gctload [-c<filename> -m<message pool size> -Ci<congestion module id>
-Co<congestion onset threshold> -Ca<congestion abatement threshold>
-d -v -t1 -t2 -x]
```

### Command Line Options

The gctload utility supports the following command line options:

- **-c<filename>**
  Specifies the system configuration file, <filename>. If not selected a default filename of system.txt is assumed.
- **-m<message pool size>**
  Specifies the message pool size, that is the number of messages available on the host. If this option is not defined, the default message pool size is 200.

  ***Note:*** For systems based on Dialogic® DSI SS7MD Network Interface Boards, a higher system throughput is expected, therefore the size of the pool should be increased to at least 10000.

  ***Note:*** For Linux systems, the kernel.msgmnb value may also have to be increased to provide stable operation.
- **-Ci<congestion module id>**
  Specifies the congestion-handling module ID. Must be set to 0xb0 (the module_id of rsi).
- **-Co<congestion onset threshold>**
  Specifies the congestion (overload) onset threshold, that is, the percentage of the total number of available messages that must be allocated before the system starts congestion procedures. The default is 50% of the messages in the message pool defined by the -m option. Once this threshold is reached, the congestion-handling module specified by the -Ci option is notified and should take steps to reduce the system loading.
- **-Ca<congestion abatement threshold>**
  Specifies the congestion abatement threshold, that is, the percentage of the total number of messages that must be available before the system stops congestion procedures. The default is 50% of the messages in the message pool defined by the -m option. Once the message pool size drops back below this threshold, the congestion-handling module, as specified by the -Ci option, is notified and can return the system to normal loading levels.
- **-t1**
  Display system trace information (short). See Section 5.5.1, “System Status (gctload -t1)” on page 59 for more information.
- **-t2**
  Display system trace information (long). See Section 5.5.2, “Show All Currently Allocated API messages (gctload -t2)” on page 59 for more information.

- **-v**
  Display version information.
- **-d**
  Enable diagnostic tracing.
- **-x**
  Terminate a running system. An active instance of the gctload module, together with any forked binaries, is terminated if a subsequent call of gctload binary is made with the -x parameter.

**Example**

To run gctload with the system.txt file as the configuration file, a congestion onset value of 70, a congestion abatement value of 30, and a message pool size of 2000, with the mandatory congestion-handling module ID set to 0xb0(the rsi module), the command is:

```
gctload -csystem.txt -Co70 -Ca30 -m2000 -Ci0xb0
```

### 5.5.1 System Status (gctload -t1)

For diagnostic purposes, it is possible to determine message queue statistics using gctload with an additional command line option. When a host is running (having already started gctload), run gctload a second time with either the -t1 or -t2 option to display message statistics to the console. The -t1 option causes gctload to print the current system statistics.

For example, the command:

```
gctload -t1
```

generates output similar to the following:

```
GCTLOAD System status:

  MSGs in system:         200
  MSGs allocated:         4
  MSGs free:              196
  Maximum MSGs allocated: 6
  Out of MSG count:       0
  Congestion module Id:   0xb0
  Congestion onset:       100
  Congestion abate:       20
  Congestion status:      0
  Congestion count:       0
```

A rising number of allocated messages indicates that there is a problem, for example, messages may be being sent to a non-existent queue, a queue that is not being read by any process in the system or a queue that is being read by an application that is failing to release the messages after processing them. The behavior of the system after it has run out of messages may be unstable and in these conditions, the gctload environment should be restarted. The contents of the currently allocated messages may be shown using the -t2 option, see Section 5.5.2 below.

### 5.5.2 Show All Currently Allocated API messages (gctload -t2)

***Caution:*** The gctload command with the -t2 option should not be used on live systems, since it locks the system until all messages have been printed out, an operation that can take a significant amount of time. The -t2 option is intended for use during fault finding on a system that has not been configured correctly.

Issuing the gctload command with the -t2 option generates a printout of all the currently allocated messages to the console. Messages are displayed in hexadecimal format as follows:

```
M t<type> i<id> f<src> d<dst> s<status> e<err_info> p<param>
```

where each field contains the value of the corresponding message field in hexadecimal format.

For example, the following command:

```
gctload -t2
```

generates output similar to the following:

```
M-t0f83-i0000-fb0-def-s02
M-t0f83-i0000-fb0-def-s01
M-t0f0d-i0000-fdf-def-s19
M-t0201-i0000-f71-def-s03
M-t0201-i0000-f71-def-s02
M-t0201-i0000-f71-def-s03
M-t0201-i0000-f71-def-s02
```

The output above indicates that there are messages sent to a destination module ID 0xef in the IPC system. Under normal operation, the message queues for destination tasks should either be empty or contain a small number of messages. If this is not the case, this may be due to one of the following reasons:

- No module has been configured to read messages for the listed destination queue.
- The destination task may have stopped reading from its message queue or may have stopped running.
- There may be a missing REDIRECT statement in the host's system.txt file to redirect messages from the listed destination to a running task.

## 5.5.3 Running gctload as a Service

The Development Pack for Windows can be configured to allow gctload to be automatically executed at system initialization. This is achieved by running gctload via a Windows service. The system, when run in this manner, can be stopped and restarted under software control and does not require a user to be logged in.

This allows:

- automatic invocation of gctload at system boot
- stopping and restarting of gctload via a standard programming interface
- a mechanism for remotely restarting the SS7 software

### 5.5.3.1 Installing the Service Software

The functionality uses the following executables:

- gctserv.exe - Service executable.
- servcfg.exe - Service configuration and installation tool.

These files are installed on the system when the Development Package for Windows is installed, but before the service itself can be installed, the executable must be copied to the system32 directory of the Windows installation using a command similar to the following:

```
copy gctserv.exe c:\winnt\system32
```

### 5.5.3.2 Installing the Service

The installation is performed using the executable servcfg.exe. You must have an account belonging to the "Administrators" user group to run the utility.

When installed, the service is identified by the name "Septel Startup Service" within the Windows Services tool.

The command line format for service installation is:

```
servcfg.exe install <service> <gctload> <system.txt> <start_dir>
```

Where:

- **<service>** is the full pathname for the service executable
- **<gctload>** is the full pathname for the gctload executable
- **<system.txt>** is the pathname for the system configuration file
- **<start_dir>** is the directory in which the service starts. All files referenced by the gctload executables (including the system.txt and all executables specified therein) must be specified with pathnames relative to this directory (or as absolute path names).

For example, if gctload.exe and system.txt are present in the c:\ss7 directory, the following command could be used to configure the service:

```
servcfg.exe install c:\winnt\system32\gctserv.exe c:\ss7\gctload.exe
system.txt c:\ss7
```

The locations of the required executables have to be specified with full pathnames including the drive letter.

***Note:*** Pathnames that contain spaces are **not** supported currently.

When the service is installed, by default the startup mode is set to “manual” mode. To configure the service to be automatically invoked at boot time, the service must be configured explicitly to “automatic” mode. This is achieved by running the services tool and setting the startup option to “automatic”.

Under Windows, the Services tool can be found in the Administrative Tools section of the control panel.

#### 5.5.3.3 Uninstalling the Service

The service is removed using the executable servcfg.exe that has the following syntax:

```
servcfg.exe remove
```

The service executable can then be removed from the system32 directory.

#### 5.5.3.4 Additional System.txt Commands

As a result of starting gctload as a service, it is no longer practical to pass command line parameters directly to gctload.

To enable you to configure the number of messages in the system and the correct congestion handling parameters, two new system.txt commands, NUM_MSGS and CONG_MSG, have been added.

The syntax of the NUM_MSGS command is as follows:

```
NUM_MSGS <num msgs>
```

where:

- **<num msgs>** is the number of messages globally allocated for use within the GCT environment (this replaces the gctload -m option)

The syntax of the CONG_MSG command is as follows:

```
CONG_MSG <module id> <onset> <abatement>
```

where:

- **<module id>** is the module ID of the module to which congestion is to be reported (this replaces the gctload -Ci option)

- **<onset>** is the level of congestion, expressed as a percentage of the buffer pool allocated, at which congestion procedures are to be initiated (this replaces the gctload -Co option)
- **<abatement>** is the level of congestion, expressed as a percentage of the buffer pool allocated, at which active congestion procedures are to be discontinued (this replaces the gctload -Ca option)

#### 5.5.3.5 Running the Service Manually

The service can be started manually using the Windows Service tool.

Select the required service, “Septel Startup Service”, and start the service (via the start icon in Windows). When the service has successfully started, the displayed status if the service changes to “started”. The service can also be stopped manually using the Windows Services tool, using the “stop” button or the stop icon. When the service has been successfully stopped, the displayed status of the service changes to “stopped”.

## 5.6 tim

### Description

The tim utility starts the tim process that receives periodic tick notification from tick processes and handles protocol timers for all other processes.

### Syntax

```
tim_xxx [-v]
```

where xxx is operating system specific, for example lnx for Linux. The syntax for Windows versions is tim_nt.

### Command Line Options

The tim utility supports the following command line options:

- **-v**
  Show version information.

### Example

The tim process is typically only started by forking a process using gctload by including the following line in the system.txt file:

```
FORK_PROCESS ./tim_lnx
```

## 5.7 tick

### Description

The tick utility starts the tick process that sends periodic tick notification to the tim process which in turn handles protocol timers.

### Syntax

```
tick_xxx [-v]
```

where xxx is operating system specific, for example lnx for Linux. The syntax for Windows versions is tick_nt.

### Command Line Options

The tick utility supports the following command line options:

- **-v**
  Show version information.

**Example**

The tick process is typically only started by forking a process using gctload by including the following line in the system.txt file:

```
FORK_PROCESS ./tick_lnx
```

# 6 Development Guidelines

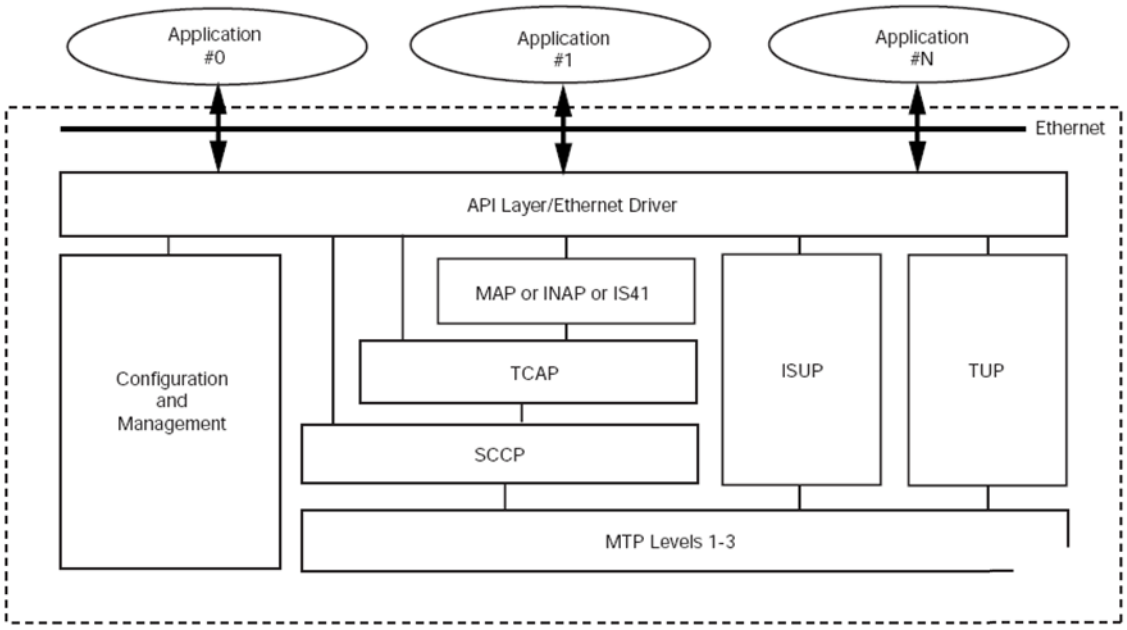
## 6.1 Overview of SIU Operation

The SIU is an SS7 network access product that provides a resilient interface to SS7 networks via a TCP/IP local area network (LAN). As shown in Figure 12, the SIU software includes SS7 protocol layers, as well as a configuration and management module. The SIU supports multiple SS7 signaling links within the same Pulse Code Modulation (PCM) trunk interface or over multiple PCM trunks.

The SIU examines the PCM timeslots carrying the SS7 information and processes them accordingly, outputting this data to the LAN using TCP/IP to be conveyed to the user application in structured messages placed in a sequential queue. Similarly, it takes messages from the application, via TCP/IP LAN, and converts those to SS7 signaling for transmission to the SS7 network.

For both circuit- and transaction-related operations, the SIU provides the ability to automatically distribute signaling information between a number of physically-independent application platforms, thus providing fault tolerance within the application space.

**Figure 12. SIU Structure**




The SS7 signaling may be presented from the network multiplexed in a timeslot on a T1 (1.544 Mbps, also known as DS1) or an E1 (2.048 Mbps) bearer.

### 6.1.1 Circuit-Switched API Operation

The message-based Application Programming Interface (API) operates transparently over TCP/IP Ethernet, using software modules provided by Dialogic. For circuit-switched (telephony) applications, each application platform terminates and hence controls a fixed range of physical circuits, or circuit identification codes (CICs). CICs are configured in groups of up to 32, each group normally equating to all the circuits in a single T1 or E1 trunk.

Each group is terminated on a fixed application platform or host processor, enabling the SIU to automatically direct API messages to the correct platform.

### 6.1.2 Transaction-Based API Operation

TCAP-based applications can be distributed on multiple application hosts using two different methods which are explained in more detail later in this chapter (see Section 6.2.10, “Failure of Application” on page 82). These methods imply running TC-user application parts (such as GSM-MAP, INAP, or IS-41) on each application host. When running any application part above TCAP on the SIU itself, the product allows operation of a single host application.

### 6.1.3 Management Interface

The SIU constantly monitors the state of its physical connections, PCM trunk inputs, the communication channel via TCP/IP Ethernet to the host processors and reports status information to an application process running on a user-defined host. The host elected to receive management messages can be selected by sending an API_MSG_COMMAND management request. Host 0 is used by default.

### 6.1.4 Potential Points of Failure

The most critical points of failure of an SIU-based system are:

- Failure of SS7 Links
- Failure of SS7 Routes
- Failure of Power Supply
- Failure of Signaling Interface Unit
- Failure of IP Subnetwork
- Failure of Application

For each of these points of failure, a solution is provided and the implementation details are given in the subsections that follow.

### 6.1.5 Failure of SS7 Links

**Problem**

With a single link to the adjacent signaling point, service is disrupted if the link goes down for some reason (for example, a layer 1 alarm or congestion).

**Solution**

Link resiliency is achieved by using multiple links between a local point code and an adjacent point code. By definition, such links are said to belong to the same link set, which can contain up to 16 links. Ideally, the links of a link set should not be carried over a unique physical medium (such as a T1 or E1 trunk) but, instead, should be split over independent physical trunks.

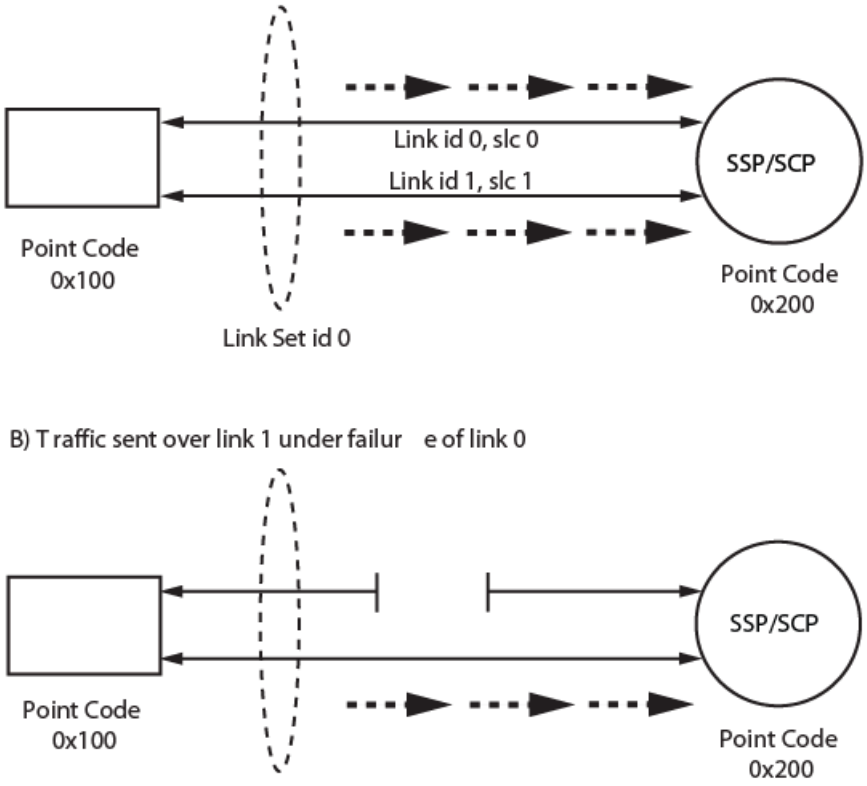
Link failure management is a standard MTP3 operation and is not an SIU-specific feature. In other words, failure between links in the same link set happens in a completely transparent way for the user application.

**Details**

In an SIU configuration, two commands are used in the config.txt file to configure link sets and links: MTP_LINKSET and MTP_LINK. In the following example, two SS7 links are defined between local point code 0x100 and adjacent point code 0x200 on time slot 16 of PCM ports 1 and 2. See also Figure 13, "SIU Connected to Adjacent Node with Two Links in a Link Set" on page 70.

```
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET 0 0x200 2 0x0000 0x100 0x08
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <bpos> <blink> <bpos2>
*<stream> <timeslot> <flags>
MTP_LINK 0 0 0 0 0 0-0 0 0 16 0x0006
MTP_LINK 1 0 1 1 0 0-1 0 1 16 0x0006
```

**Figure 13. SIU Connected to Adjacent Node with Two Links in a Link Set**




## 6.1.6 Failure of SS7 Routes

**Problem**

With a single route to a destination point code (DPC), service can be disrupted if all the links of the link set used to reach that signaling node fail.

**Solution**

To eliminate this single point of failure, an alternative link set can be provided in the SIU system configuration to reach the same DPC. Route failover is a standard MTP3 operation which does not require any specific action from the user application.

***Note:*** When an alternative route to a given DPC is defined in an SIU configuration file, a choice must be made between two different traffic modes: load sharing or failover. In load-sharing mode, traffic sent towards the remote signaling point is shared between the two link sets. In failover mode, all traffic sent towards the remote signaling point

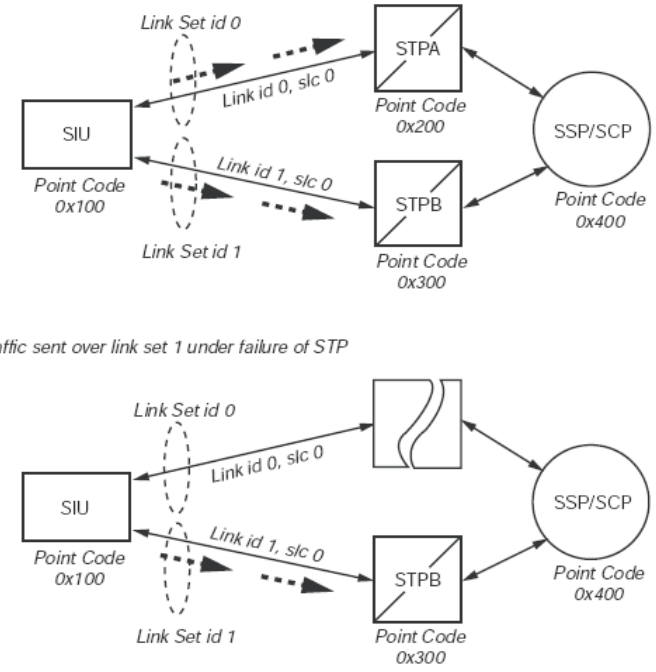
will normally be sent using the primary link set, unless this link set fails, in which case the traffic will use the alternative link set. See Section 7.6.7, “MTP_ROUTE” on page 157 for more information on the selection of traffic mode.

**Details**

The example following shows two link sets (each containing one link) being used in load-sharing mode to reach destination point code 0x400. See also Figure 14.

```
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET 0 0x200 2 0x0000 0x100 0x08
MTP_LINKSET 1 0x300 2 0x0000 0x100 0x08
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <bpos> <blink> <bpos2>
*<stream> <timeslot> <flags>
MTP_LINK 0 0 0 0 0 0-0 0 0 16 0x0006
MTP_LINK 1 0 1 1 0 0-1 0 1 16 0x0006
MTP_LINK 2 1 0 0 0 0-2 0 2 16 0x0006
MTP_LINK 3 1 1 1 0 0-3 0 3 16 0x0006
* MTP_ROUTE <route_id> <dpc> <linkset_id> <user_part_mask> <flags> <second_ls> *<reserved>
MTP_ROUTE 0 0x400 0 0x0020 0x0003 1 0
```

**Figure 14. SIU Connected to Mated STP Pair Providing Route Resiliency**




## 6.1.7 Failure of Power Supply

**Problem**

Ensuring that the unit survives the loss of one power supply and making it possible to replace a failed power supply without affecting the availability of the system.

**Solution**

The SIU can be optionally configured with a redundant and hot swappable power supply.

**Details**

See the *SS7G21 and SS7G22 Hardware Manual* to obtain part numbers for redundant power supplies for the SS7Gx (operating as an SIU).

For the Dialogic® DSI SS7G41 Signaling Server products, refer to the *Dialogic® DSI SS7G41 Signaling Servers Product Data Sheet* (navigate from http://www.dialogic.com/products/signalingip_ss7components/signaling_servers_and_gateways.htm) for a list of the ordering codes and definitions of the hardware variants.

### 6.1.8 Failure of Signaling Interface Unit

**Problem**

Since the SIU provides an SS7 interface to a distributed application, it is usually deployed for high-density service platforms. Should the SIU in a single SIU-based system fail, many resources (telephony circuits to TCAP dialogs) would become unavailable and would cause major service disruption.

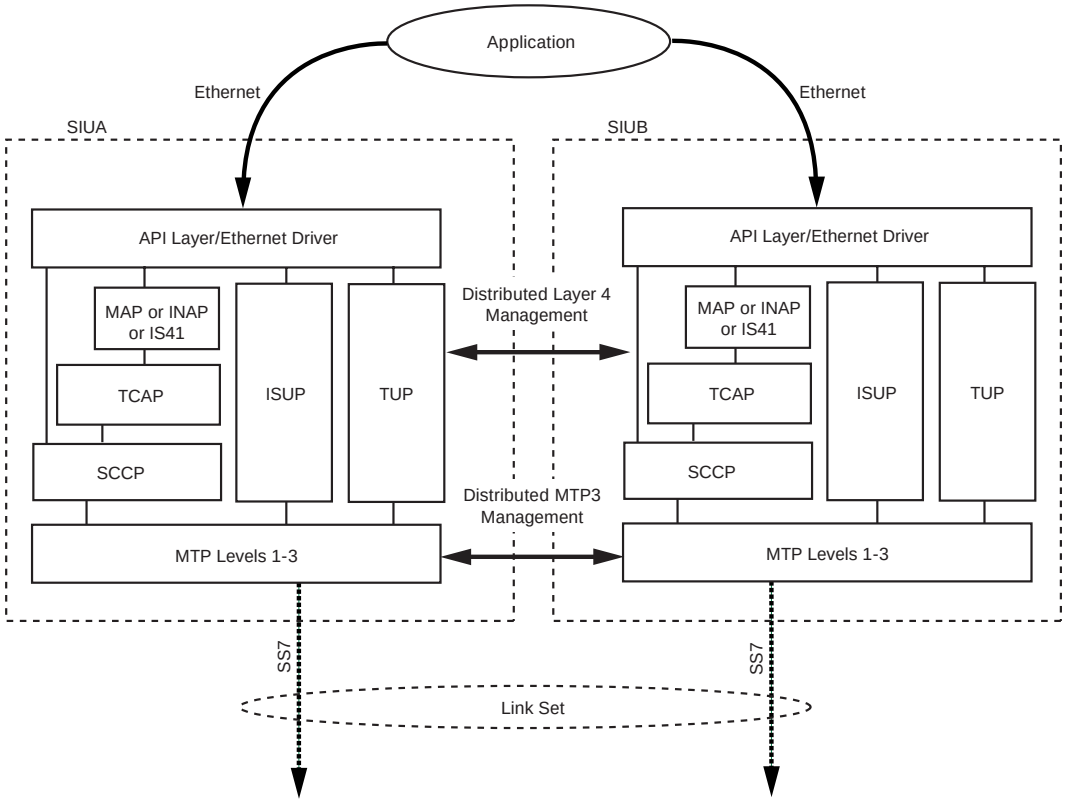
**Solution**

A major feature of the SIU architecture is that two units can be configured to operate as a single entity, sharing the same local SS7 point code. In normal operation, signaling can be shared between two units. In the event of a failure, signaling is maintained by the remaining unit.

**Details**

In a dual resilient configuration, one unit is assigned as SIUA, the other as SIUB. Under normal operation, the application uses both the resources of SIUA and SIUB. See Figure 15.

**Figure 15. Dual SIU Architecture**




The distributed layer 4 management is achieved using a LAN connection and allows SS7 messages for any transaction or call to be received by either unit, regardless of the unit that is actually processing the call or transaction. The distributed MTP3 functionality is achieved using a specially-configured inter-SIU link set, containing one or more signaling links. Transmit messages from

each SIU are load shared between links that connect to the local SIU, if these are available. If all local network-facing SS7 links have failed, transmit messages are relayed to the partner SIU across the inter-SIU link set and sent out to the adjacent signaling point by the partner unit. The inter-SIU link set also provides the capability of message retrieval and retransmission when a changeover operation occurs between the two units.

- For circuit-switched applications, the circuit groups are configured on both units, letting the application select which SIU controls each group. In normal operation, the control of circuit groups is distributed between both the SIUA and SIUB. In the event of failure of a unit (or for maintenance), the application can move control of each circuit group from one SIU to the other.
- For transaction-based applications, the transactions are shared equally between the two units.

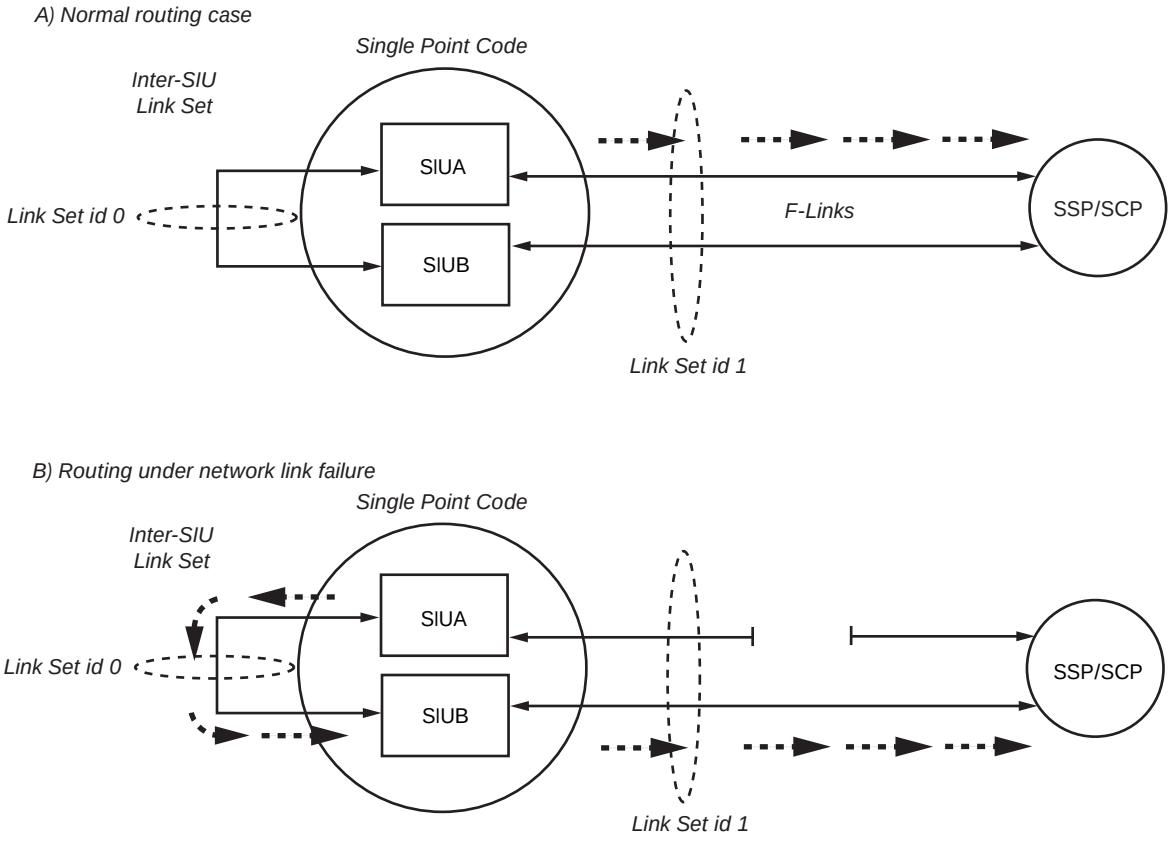
#### 6.1.8.1 Routing Architectures of a Dual-resilient SIU System

The routing options (that is, a straight connection to the DPC vs. an indirect connection via a pair of STPs) described in this section will vary based on the actual network architecture that is being supported for a particular application.

**Connection to a Single Adjacent Signaling Point**

Figure 16 shows two possible routing alternatives for SIUA routing to an adjacent SSP or SCP. Messages issued from SIUA are sent to the destination SSP or SCP using local SS7 links if available. If these fail, transmit messages are relayed through SIUB over the inter-SIU links. In this case, the DPC is also the adjacent signaling point. Although Figure 16 shows an SIU pair connected to a single adjacent signaling point, the pair may be connected to multiple destinations.

**Figure 16. Transmit Routing to a Single Destination**



The number of links allocated in the inter-SIU link set has to be carefully calculated. Since these links will be used for outgoing traffic only, they can be used at a higher capacity than network-facing links. Moreover, since only half of the traffic can potentially be routed through that link set, a common rule is to allocate a fourth of the total number of network-facing links in the inter-SIU link set. For example, in a pair of Signaling Servers with 12 links per unit, 24 links total, 16 links will be typically allocated in the network-facing link set and four links will be allocated in the inter-SIU link set for each SIU. Routes to destinations are configured such that there is a primary link set (the link set from the SIU to the DPC) and a secondary link set (the inter-SIU link set) which is used only when the primary link set has failed.

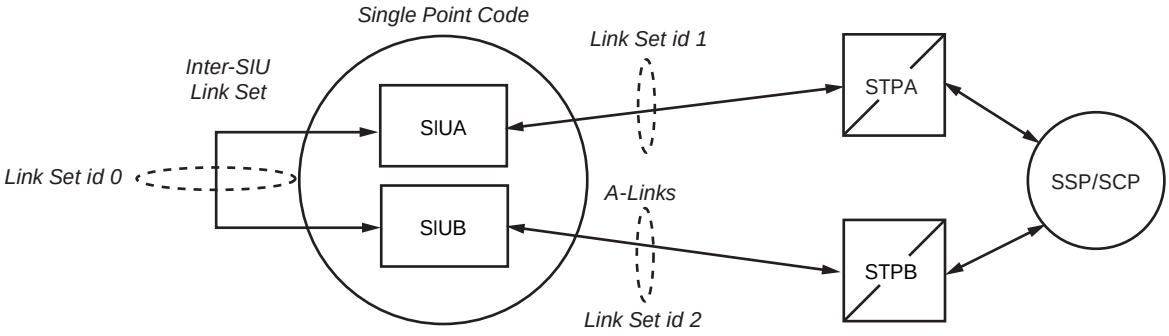
In Figure 16, case (A), since the F-links exist in the same link set, messages may be sent from the adjacent SSP/SCP to either SIUA or SIUB. If an SIU receives a message from the SS7 network for a circuit or transaction that it does not control, this receive message is passed automatically to the other SIU for processing via the TCP/IP LAN.

**Connection to an Adjacent Mated STP Pair**

There are two ways of connecting a pair of SIUs to a mated STP pair.

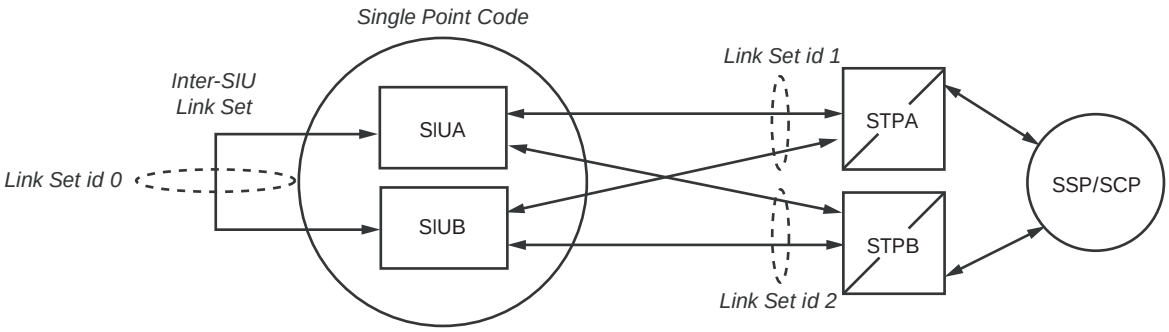
In the first method, the straight link configuration, each SIU is configured with two link sets: one link set containing STP-facing links, plus the inter-SIU link set. The straight link configuration consists in connecting SIUA to STPA and SIUB to STPB, as shown in Figure 17.

**Figure 17. Dual-resilient SIUs Connected to a Mated STP Pair in a Straight Link Configuration**




The second method, the crossed link configuration, consists of the addition of crossed link connections between SIUA and STPB and between SIUB and STPA to the previous mode. In a crossed link configuration, each SIU is configured with three link sets: two link sets containing the links towards each STP, plus the inter-SIU link set. See Figure 18. The configuration of the DPC will contain the link set IDs of the two link sets connected to the STPs. Load sharing can be enabled to take advantage of all the system resources. In such a configuration, the inter-SIU link set is not used for traffic failover, but only for synchronization of network management messages.

**Figure 18. Dual-resilient SIUs Connected to a Mated STP Pair in a Crossed Link Configuration**



Consequently, a single link within this link set is adequate enough, giving an increased bandwidth for network-facing links. The inter-SIU link must be carefully dimensioned since it needs to support the outgoing traffic of the SIU that would have lost its entire network-facing links, as shown in Figure 19. Again, a common practice is to allocate a fourth of the total number of network-facing links in the inter-SIU link set.

**Figure 19. Transmit Routing Through Mated STPs**

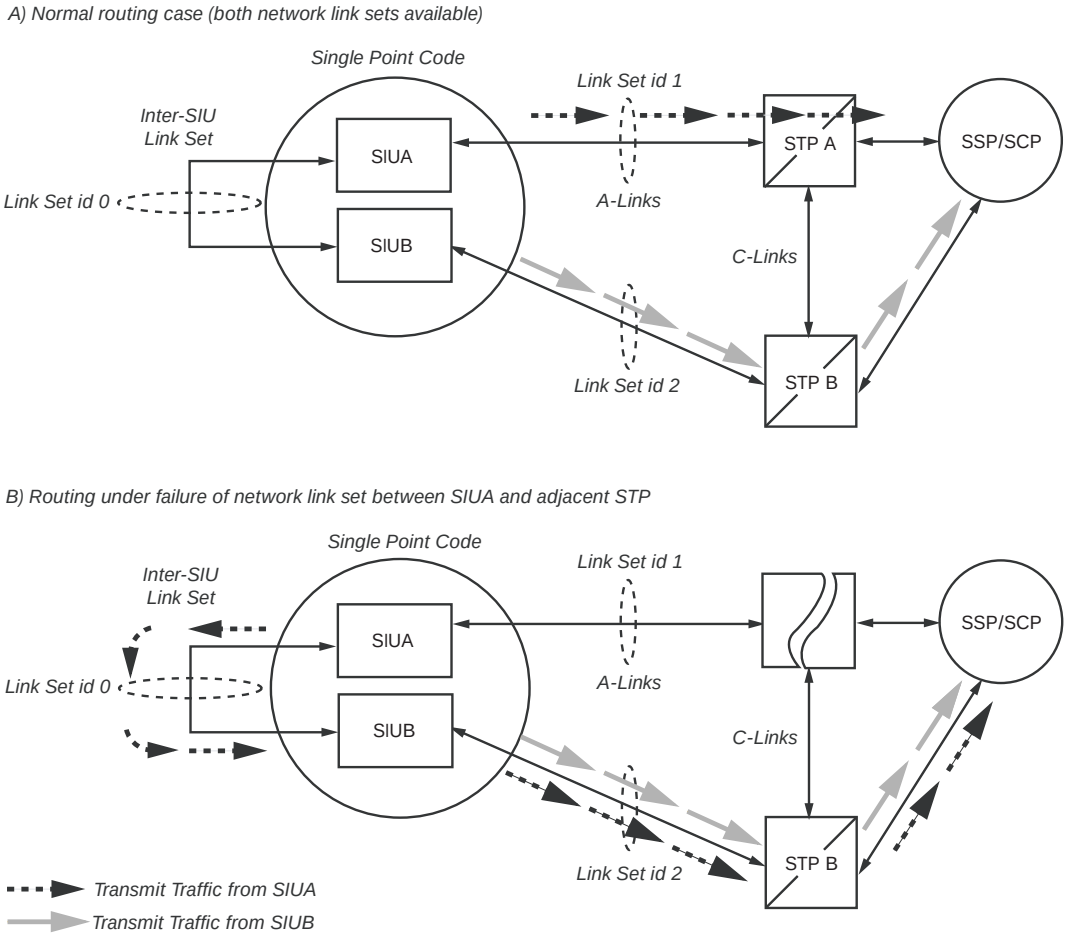


Table 2 compares the advantages and disadvantages of these methods.

**Table 2. Comparison of a Straight Link Configuration vs. Crossed Link Configuration**

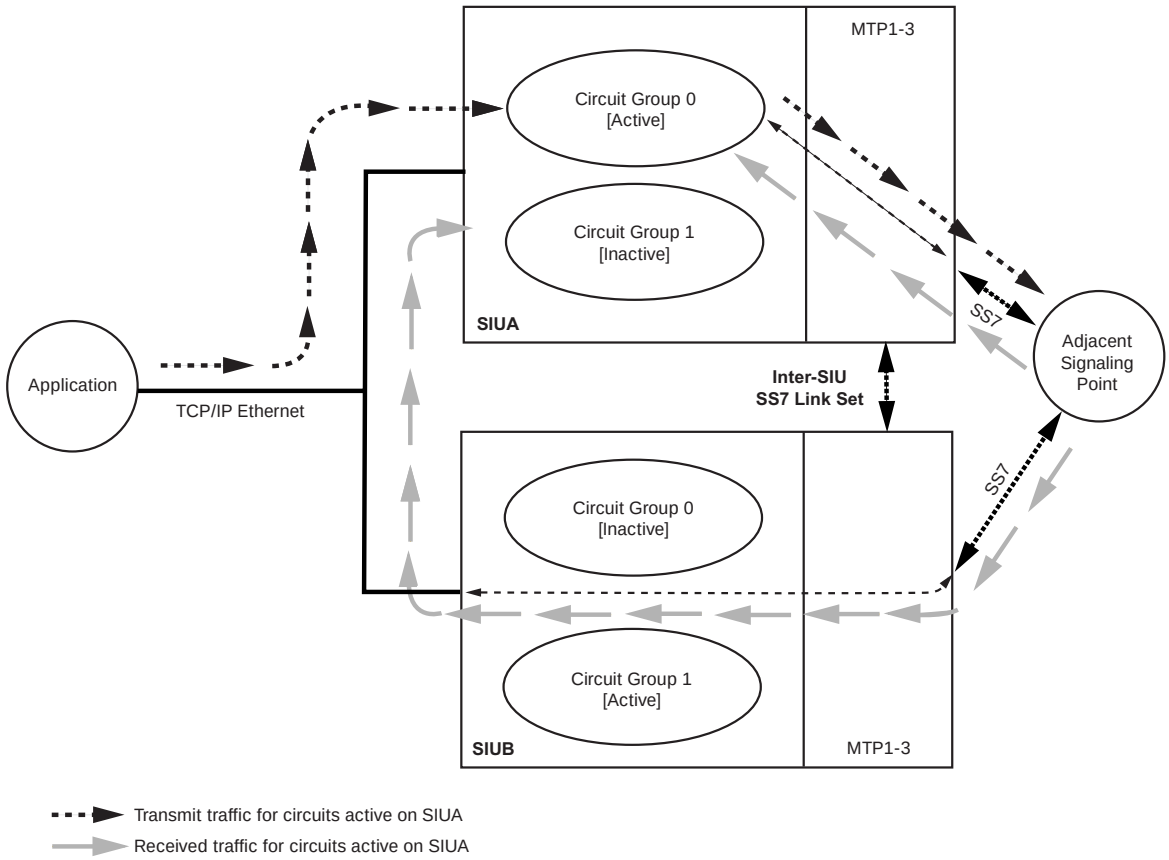
| Comparison Subject | Straight Configuration | Crossed Configuration |
|---|---|---|
| Load sharing | - STPA can only load share traffic for SIUA and vice-versa | + Each STP can load share between the two SIUs, optimizing the resource utilization |
| Network-facing links failure | + SIUA can rely on SIUB to send outgoing traffic upon failure of its entire network-facing links | - When an SIU loses its network-facing links, the application must activate circuit groups on the surviving SIU (for ISUP-based application) |
| Inter-SIU link set dimensioning | - Need to allocate 1/4 of all network facing links (for example, 16 network facing links and four inter-SIU links) | + Need to allocate a single link, maximizing the number of network-facing links (for example, 22 network facing links and one inter-SIU link). Allocating a single link means there are two single points of failure in the system. For best resilience, the inter-SIU link set should contain two links spread across two signaling boards in each SIU. |

#### 6.1.8.2 Dual SIU Architecture for Circuit-Switched Applications

Within the SIU environment, circuits are configured in groups, each group equating to all the circuits multiplexed onto a single T1 or E1 PCM trunk. The SIU provides the SS7 circuit control and the application platform (or host) terminates the physical channels, typically with a voice processor.

For normal operation, half the circuit groups are controlled by SIUA and half by SIUB. As each application platform starts up and connects to both SIUA and SIUB, the application must nominate which SIU is to control the signaling for each of the circuit groups it terminates. A circuit group activation command must be sent to the selected SIU for each circuit group. Any outgoing messages for circuits in this group must be sent to this SIU, as shown in Figure 20.

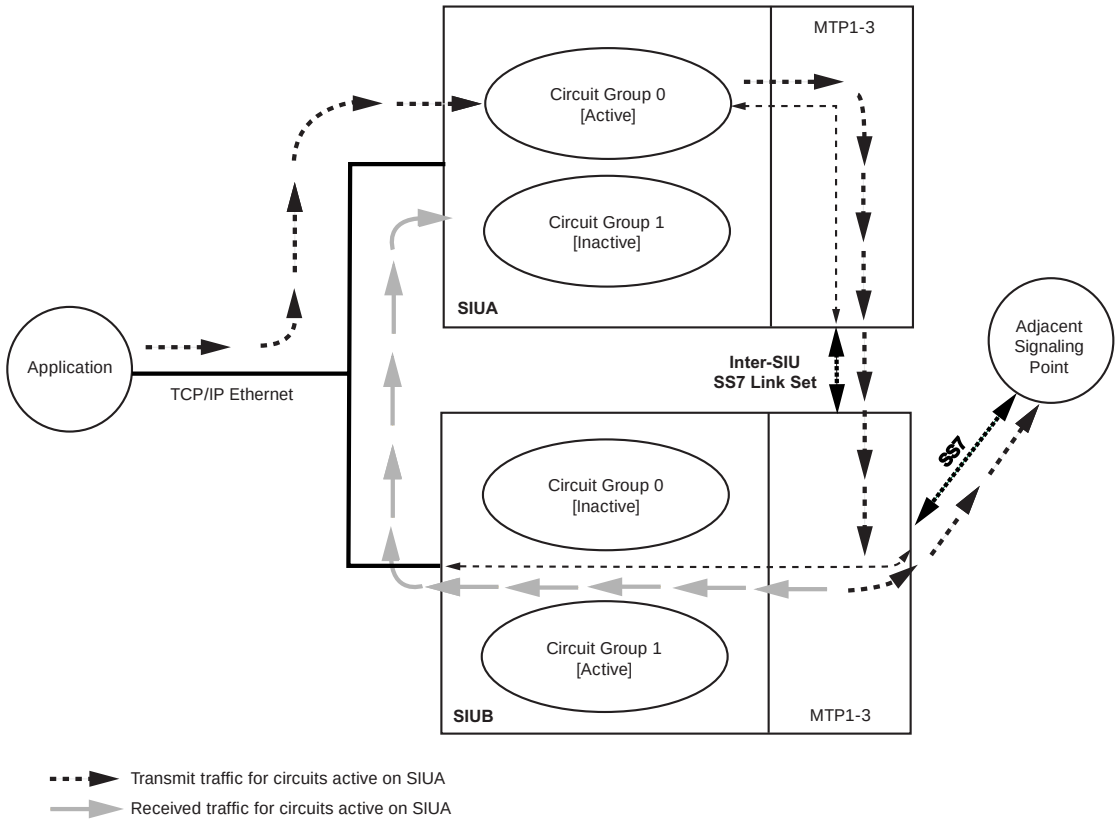
**Figure 20. Normal Routing for Circuit Group 0 When Controlled by SIUA**




The adjacent signaling point views the links connected to SIUA and SIUB as the same link set and, as such, is free to send messages for the circuits controlled by SIUA to either unit. In the case where SIUB receives a message for a circuit controlled by SIUA, the message is automatically routed to the correct controlling circuit group using the LAN Ethernet connection (shown by the shaded arrows in Figure 20).

If all the links between the controlling SIU and the adjacent signaling point fail, all transmit traffic is automatically routed to the adjacent signaling point through the inter-SIU link set as shown in Figure 21. The application should continue to use the SIU as before, directing all outgoing circuit requests to SIUA.

**Figure 21. Routing When All Local Links Have Failed, Group 0 Controlled by SIUA**

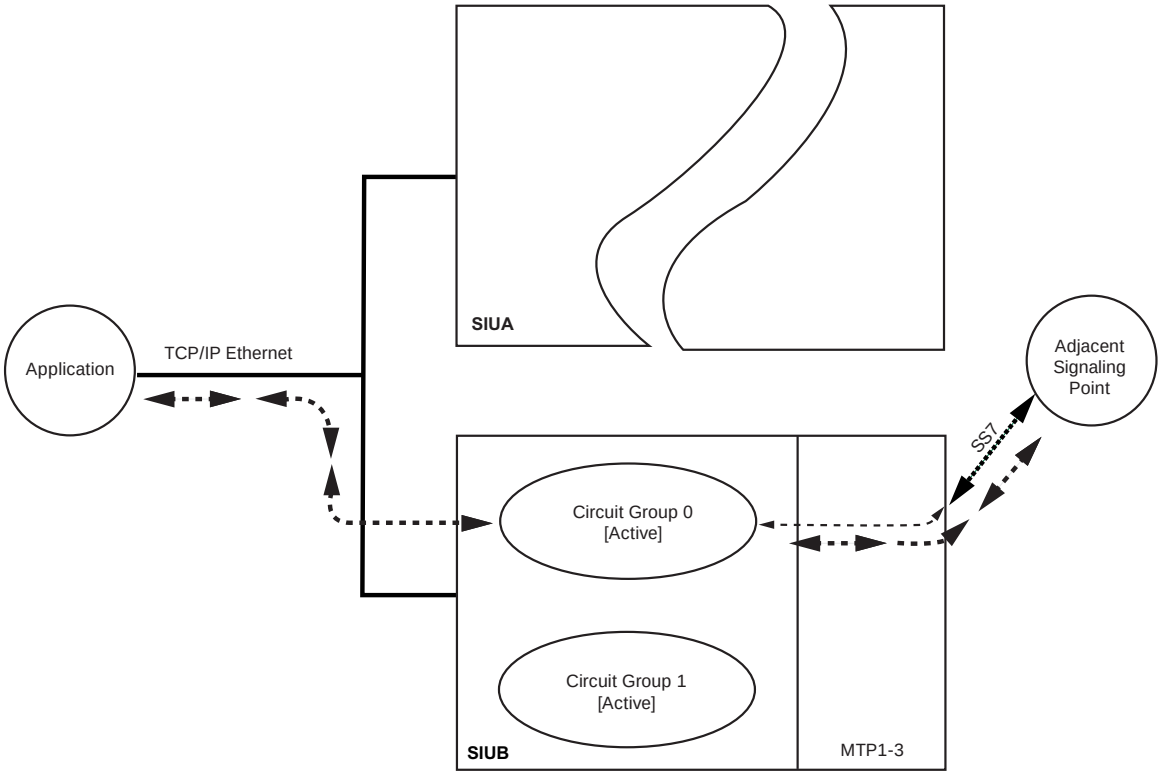



If the controlling SIU fails, the application is informed of the failure and should transfer control of the circuit group to the remaining unit. The following also apply:

- Calls in a steady state (speech) continue uninterrupted.
- Outgoing calls in a set-up state during the transfer should be reattempted.
- Incoming calls being set up by the interconnected SS7 equipment also fail and are reattempted remotely.

The circuit group control will then appear as shown in Figure 22. The user application software should reset all idle circuits following a transfer, and reset all remaining circuits as they become idle. When the failed unit recovers, control of the circuits may be transferred back by the application.

**Figure 22. Routing Following Failure of SIUA**




Circuit group control is transferred by the application in two stages. The group should be deactivated from one unit (if communication with that unit is possible) before being activated on the partner SIU. To protect against control being active on both units at the same time, the SIU automatically issues a deactivate command to the partner unit in response to an activate command from the host application and checks the status of each circuit group on the partner unit. An API management event indication is given if a dual resilient configuration is detected.

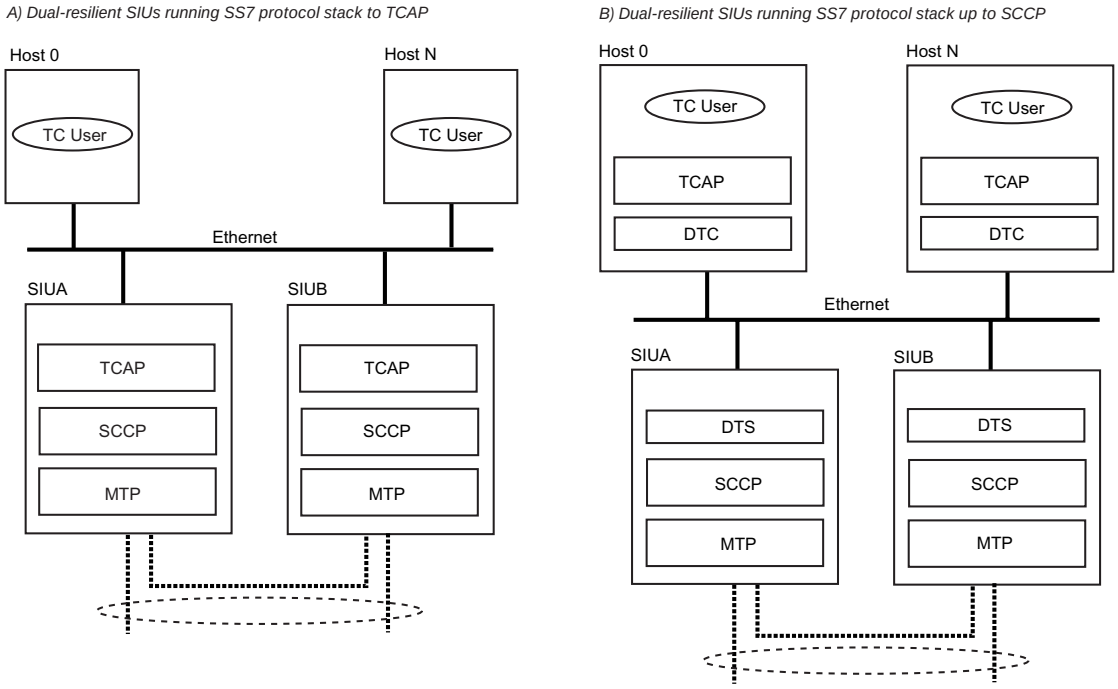
#### 6.1.8.3 Dual Resilient SIU Architecture for Transaction-based Applications

There are two ways of architecting a dual resilient SIU-based system for processing TCAP transactions.

- Running the SS7 stack up to SCCP on the SIUs
- Running the SS7 stack up to TCAP on the SIUs

In the first case, TCAP and potential layers sitting on top of TCAP run on the application host(s) and an additional piece of software is required on each SIU to distribute TCAP transactions to multiple application hosts. This software option is called distributed transaction server (DTS). Figure 23 depicts this architectural difference.

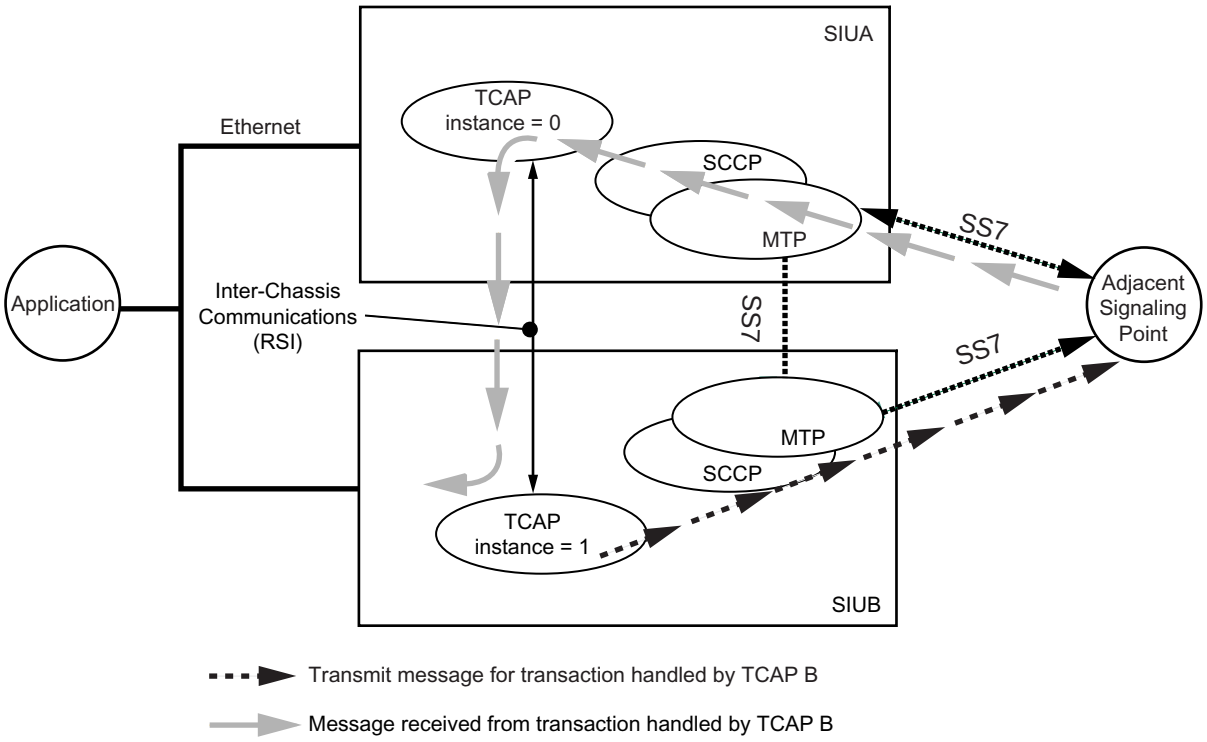
**Figure 23. Two Different Architectures for a TCAP Processing SIU System**




In the second case, each unit controls half of the total available transactions. The SIU supports up to 64k-1 transactions. A dual resilient configuration can consequently handle up to 128k-1 simultaneous transactions. Each transaction is processed for its entire duration by the SIU that processed the first TCAP message. The user application must therefore direct all messages for a transaction to the same SIU, and load balance outgoing dialogs between the two units. An incoming TCAP dialog message other than BEGIN or QUERY is handled by the SIU that processed the first TCAP message for that dialog received from the SS7 network. When an SIU receives a TCAP message that belongs to a transaction that was initiated on the other unit, it passes this message to its peer SIU over the RSI connection. This is shown in Figure 24. Failure of an SIU reduces the number of available transactions to one-half. In a dual resilient transaction-based SIU system running the SS7 stack up to SCCP on the SIU, and TCAP (and above) on the application host, each host controls a fixed number of transactions. Each transaction is processed for its entire duration by the application host that processed the first TCAP message. Upon failure of one SIU unit, the TCAP capacity and ongoing transaction are totally unaffected.

The architectural decision taken on where the TCAP module is running also has consequences on the level of application resiliency and total system capacity. These consequences are explained in more details in Section 6.2.10, “Failure of Application” on page 82.

**Figure 24. Message Flow on a Dual-resilient System Running the SS7 Stack up to TCAP**




## 6.1.9 Failure of IP Subnetwork

**Problem**

Should one subnetwork go down due to a network component failure, the hosts connected to the SIU over the other subnetworks will remain active and attempt to preserve half of the total system capacity.

**Solution**

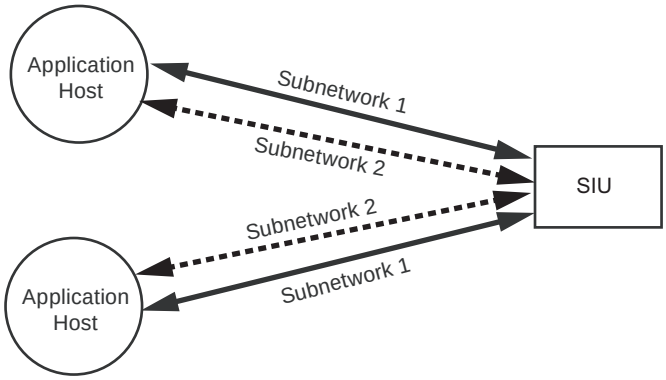
The SIU can be configured with up to six Ethernet ports using the NIC accessory. This allows the splitting of IP connections between the SIU and the application hosts onto a maximum of four physically-separated subnetworks. Figure 25 shows IP resilience within two subnetworks between the SIU and two hosts. Configuring the SIU to exist in multiple IP networks reduces the risk of losing all IP connectivity in the event of a switch/router/hub failure in the LAN.

If there is only a single IP network available, resilience can be achieved with IP port bonding. Using IP port bonding, two IP ports on the SIU are configured in an active/standby relationship underneath a single IP address. On failure of the primary IP port, the secondary IP port becomes active. See *IP Port Bonding* in the *Dialogic® DSI Signaling Servers SS7G41 Operators Manual* for further information.

**Details**

Section 6.2.10, “Failure of Application” on page 82 shows how to take advantage of the dynamic configuration features offered by the SIU to failover the affected application hosts to the surviving subnetwork.

**Figure 25. Dual LAN Operation on the SIU**




## 6.1.10 Failure of Application

**Problem**

The failure of an application host leads to the loss of a portion of system resources.

**Solution**

The most basic feature causing this is that the application can be deployed on multiple hosts. The SIU supports up to 128 hosts. For circuit-switched applications, failure of a host generally means loss of the physical trunk interface; hence, there is no need to transfer the logic to other (surviving) hosts. More sophisticated features are available to allow TCAP-based applications to failover to other hosts.

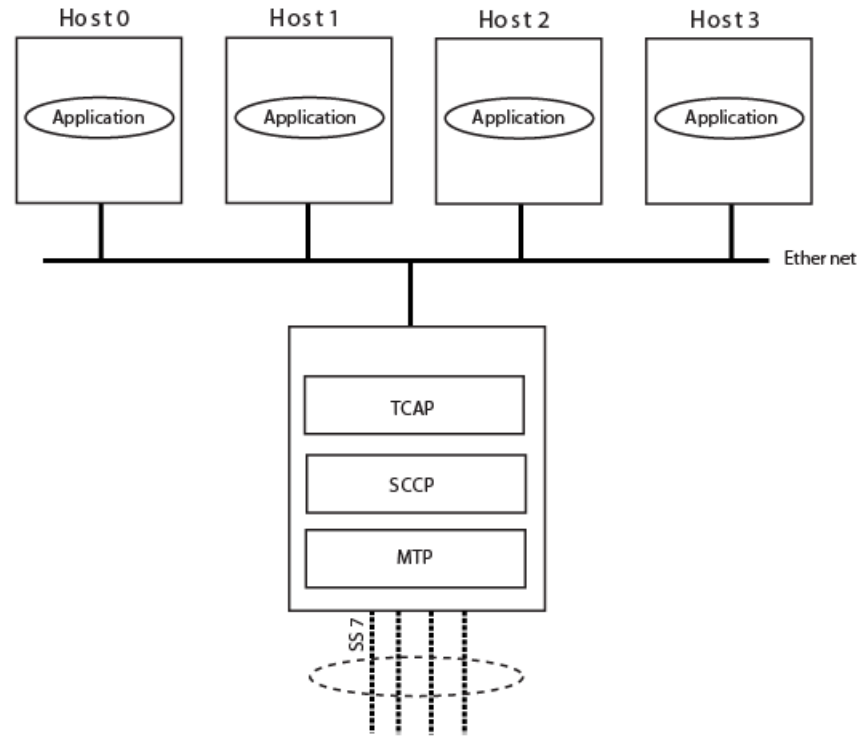
**Details**

For TCAP-based applications, the SIU allows operation of multiple application hosts interfacing directly to TCAP, hence giving a certain level of resiliency in the user application space. Two methods are available for this purpose; both of which are explained in the following subsections.

### 6.1.10.1 TCAP Resiliency Based on Dialog Groups

Fixed ranges of TCAP dialogs can be created in the SIU configuration file and assigned to different application hosts. TCAP dialog groups are defined using the TCAP_CFG_DGRP command in the config.txt file. See Section 7.10.3, “TCAP_CFG_DGRP” on page 184 for more information. The application program running on each host must therefore ensure that only dialog identifiers from the assigned range are used. Optionally, a TCAP-user layer such as MAP, INAP, or IS41 can run on each application host to provide some application part functionalities. Figure 26 describes such a distributed architecture, where TCAP transactions are handled by four different hosts, each of them running MAP and a MAP application. The total number of TCAP dialogs for the whole system is 65,535 and this number does not depend on the number of hosts.

**Figure 26. TCAP Dialog Groups Example**




#### 6.1.10.2 TCAP Resiliency Based on DTS/DTC Option

Alternatively, it is possible to distribute SCCP traffic to multiple application hosts using the Distributed Transaction Server/Distributed Transaction Client (DTS/DTC) software option, as shown on Part B of Figure 23. In this architecture, TCAP and potential application parts run on each application host. Consequently, the total dialog capacity of such a system depends on the number of application hosts multiplied by the number of dialogs supported by the TCAP module on each individual host. See the *Dialogic® SS7 Protocols DTS User Guide* for more information on the DTS/DTC software option.

## 6.2 Dual Resilient SIU Operation

The following aspects of a dual resilient SIU-based system are described:

- Connecting a Host to Two SIUs
- Communicating with Both SIUA and SIUB
- Transferring Control of a Circuit Group Between SIUs

### 6.2.1 Connecting a Host to Two SIUs

In a dual resilient SIU system, each host should connect to both SIUA and SIUB at start-up. This is achieved using the rsicmd utility twice: first, with an siu_id of 0 for SIUA and second, with an siu_id of 1 for SIUB. For example, if SIUA has an IP address of 192.168.0.1 and SIUB an IP address of 192.168.0.2, the entry in the host's system configuration file, system.txt, would be as follows:

```
FORK_PROCESS rsicmd 0 0xef 0
```

```
192.168.0.1 9000

FORK_PROCESS rsicmd 1 0xef 0
192.168.0.1 9000
```

The “concerned module id” (0xef in this case) receives status indications from the rsi process for both connections. The **id** field of the MSG header is set to the siu_id to identify the link that each status indication relates to.

The ability to communicate between a host and an SIU is indicated by RSI_MSG_STATUS messages received by the conc_id application process (0xef in this example).

### 6.2.2 Communicating with Both SIUA and SIUB

The user application exchanges information with the SIU via API messages (MSG). In a dual resilient SIU system, each time the user application sends a message to the SIU, it should be directed to either SIUA or SIUB using the library function **GCT_set_instance( )**. In the receive direction, the application can determine the SIU that sent a MSG using the library function **GCT_get_instance( )**.

Function definitions may be found in the sysgct.h header file. The definitions are given here for convenience:

**GCT_set_instance( )**

```
int GCT_set_instance(unsigned int instance, HDR *h);
```

This function sets the destination instance number (SIU identity or siu_id) prior to sending a message and returns 0 on success, non-zero otherwise (currently no failure conditions are defined). SIUA is instance 0 and SIUB is instance 1, assigned by the **siu_id** parameter provided to the rsicmd utility. This function should be called immediately before the **GCT_send( )** function.

**GCT_get_instance( )**

```
unsigned int GCT_get_instance(HDR *h);
```

This function returns the instance number (SIU identity or siu_id) after receiving a message. The parameter **h** is a pointer to the HDR structure at the start of the received MSG. The returned value is either 0 or 1. SIUA is instance 0 and SIUB is instance 1, as assigned by the **siu_id** parameter provided to the rsicmd utility.

### 6.2.3 Transferring Control of a Circuit Group Between SIUs

The transfer of control of circuit groups between SIUs is described under the following topics:

- “Circuit Group Configuration”
- “Activating and Deactivating Circuit Groups”
- “System Initialization”
- “Failure Detection”
- “Transferring the Circuit Group”
- “Resynchronization of Circuit State Information”
- “Recovery of the Failed Unit”
- “Transferring Control Back”
- “Circuit Group Conflict”

#### 6.2.3.1 Circuit Group Configuration

For dual resilient operation, each signaling server should contain identical circuit group declarations using the appropriate ISUP_CFG_CCTGRP command. These circuit group configurations do not become active on either unit until an Activate Circuit Group API command (API_MSG_COMMAND with cmd_type = 8) has been issued to a particular signaling server.

#### 6.2.3.2 Activating and Deactivating Circuit Groups

At run time, the application running on each host should select which SIU is currently in control of each group by “activating” and “deactivating” groups on a particular SIU.

Circuit groups are activated and deactivated using the API_MSG_COMMAND message (type 0x7f0f), with a **cmd_type** of 8 to activate a group and a **cmd_type** of 9 to deactivate a group. The format of this message is described in Section 4.6.1, “API_MSG_COMMAND” on page 38. This message should be issued with a request for a response (an acknowledgement); this will be returned to the requesting application with a status value of zero (indicating “success”) or non-zero values (indicating “busy” or “failure”).

#### 6.2.3.3 System Initialization

When the system starts, the host establishes communication with both SIUA and SIUB, either by using the rsicmd utility or by issuing RSI configuration API messages directly from within the application.

When the communication between the host and the SIU is established, the RSI task on the host issues an RSI_MSG_LNK_STATUS API message with a status value set to 1 (link to SIU recovered) to a destination task conc_id on the host (conc_id is set when the RSI link was started). This message is only received by the application if the RSI link is configured with the conc_id set to the application’s module ID.

The ID field of this message is set to 0 to indicate SIUA and 1 to indicate SIUB. When the link to the SIU that normally controls a circuit group (the primary SIU) becomes active, the application should issue an activate group command to that SIU, specifying that circuit group (using its group ID). The SIU processes API commands sequentially and the application must wait for an acknowledgement of this command before proceeding. An acknowledgement that indicates “busy” should cause the application to reattempt the activate command.

The application should wait for a period of time sufficient to establish communication to the preferred SIU before deciding that the preferred unit is not available and activating circuit groups on the non-preferred or secondary SIU.

Once the acknowledgement of the activation of a circuit group is received, the circuits should be reset to force the circuits into a known, idle state. This is achieved using the Circuit Group Status Control (CGSC) Request API message. The circuit reset is acknowledged by the terminating exchange; this acknowledgement is passed to the user application as a circuit group status confirmation API message. On receipt of this, the application may commence using the associated circuits for calls. See the *ISUP Programmer’s Manual* for details on the API messages mentioned here.

#### 6.2.3.4 Failure Detection

The event that triggers the application to transfer circuit groups from one SIU to another is loss of communication between the application and the SIU. When the failure occurs, the RSI task on the affected host detects the loss of communication and issues an RSI_MSG_LNK_STATUS API message with a status value set to 2 (link to SIU lost) to a destination task conc_id on the host

(conc_id is set when the RSI link was started, optionally by the rsicmd utility). This message is only received by the application if the RSI link is configured with the conc_id set to the application's module ID.

At the same time, the affected SIU (if it can), issues an API_MSG_SIU_STATUS message with a status value of 30 (decimal) indicating a host link failure on the specified host ID. This message is sent to the host configured to receive management messages (host 0 by default).

There are two failure modes that can cause loss of communication:

- Complete failure of one SIU in a dual resilient configuration
- Partial TCP/IP failure causing loss of communication between the host and one SIU of the pair via the TCP/IP LAN

From the application's point of view, there is no difference in these cases since the RSI link fails in either case. From a system point of view, the main difference is that in the second case, the inter-SIU communication may still be functioning.

If the affected SIU loses communication with all of its hosts, it automatically deactivates all SS7 signaling links, preventing any messages from being processed by any remaining active circuit groups.

#### 6.2.3.5 Transferring the Circuit Group

If any of the circuit groups terminating on the host are currently active on the affected SIU, the host application must transfer control of each affected circuit group from the failed SIU (the primary SIU) to the remaining SIU (the secondary SIU). Transferring a circuit group normally involves deactivating the group on the controlling SIU then reactivating it on the other. However, since the host is unable to communicate with the failed SIU, the application is only required to send an API_MSG_COMMAND message to the secondary SIU with **cmd_type** of 8 (activate circuit group) for each affected group.

The activate circuit group command should be issued with a request for a response and the application should not send any call processing or circuit management commands until the response (acknowledgement) has been received from the secondary SIU.

The SIU processes single commands in sequence; therefore, if an activate command is received while a previous command is executing, the response is received with a non-zero status (in this case, a value of 4 indicating "equipment busy"). The application should reattempt the activate command on receipt of a response indicating "busy".

Since the failure may affect SIUA and SIUB, the host may choose to wait for a period of time following notification of the failure to determine if communication with the other unit remains stable. The circuit groups may then be transferred after this timeout if the communication to the secondary unit remains active.

#### 6.2.3.6 Resynchronization of Circuit State Information

Once the circuit group activation(s) are acknowledged from the secondary SIU, the application needs to resynchronize the circuit state information based on the application's knowledge of the current circuit state. This is achieved by sending three CGSC requests to the secondary SIU.

Circuits that were in a call set-up state or idle (that is, any circuit that was not in the steady state "speech" or "answered") should be RESET. Circuits that were in the speech stage of an incoming call should be forced to INCOMING ACTIVE; circuits that were in the speech state of an outgoing

call should be forced to OUTGOING ACTIVE. The forcing of the circuit state to either INCOMING ACTIVE or OUTOING ACTIVE is achieved using the CGSC Request API message, with **ptype** set to 14 (decimal) for INCOMING ACTIVE and 15 (decimal) for OUTGOING ACTIVE.

Calls that were in outgoing set-up prior to the transfer should be reattempted following successful completion of the transfer. The application should be able to reattempt a failed outgoing call attempt, as this may occur under normal operating conditions. The originating switch automatically reattempts calls that were in incoming setup. When these commands are acknowledged, the application may resume normal call activity. See the *ISUP Programmer's Manual* for details on the messages mentioned here.

#### 6.2.3.7 Recovery of the Failed Unit

The host application is informed of recovery of the communication to the SIU with the same method used for notification of the failure. The RSI_MSG_LNK_STATUS message in this case contains a status value of 1 (link to SIU recovered).

The host nominated to receive management indications (normally host 0) also receives an API_MSG_SIU_STATUS message with status value 31 (decimal) indicating a host link has recovered to the specified host ID.

#### 6.2.3.8 Transferring Control Back

Immediately following reestablishment of communication with the primary SIU, the application should send deactivate circuit group messages to this SIU to ensure that groups are only active on the secondary unit (this may be the case if the inter-SIU link had also failed). If the primary unit has recovered from a complete failure, no circuit groups will be active. This deactivate command fails if the groups were not active and the application should ignore any acknowledgement of this command with a status value indicating processing failure. A "busy" response should cause the application to reattempt the deactivate operation.

When communication with the primary SIU has been reestablished, the application should allow sufficient time to ensure that the communication is stable, thus avoiding repeatedly transferring circuits between units. After this time has expired, the application should transfer control of the affected circuit groups back to the original SIU. This is achieved by deactivating the groups on the secondary SIU and re-activating the ones on the primary SIU. However, before the groups are deactivated, the circuits in that group should be maintenance blocked (using the Circuit Group Supervision Control API message as described in the *ISUP Programmer's Manual*). This does not affect any calls in progress, but prevents these circuits from being selected for any new incoming calls. The application should also ensure that none of the affected circuits are selected for new outgoing calls.

When all existing calls are completed (all the circuits are therefore idle), the application should deactivate the circuit group by sending an API_MSG_COMMAND message with a **cmd_type** of 9 to the secondary unit. When the acknowledgement that this command has been successfully processed is received, the groups should be activated on the primary unit by sending an API_MSG_COMMAND message with a **cmd_type** of 8.

Once the acknowledgement of the activation has been received by the application, all affected circuits should be reset. This forces the circuits to a known (idle) state and remove the blocking status. When the reset is acknowledged from the terminating switch (by receipt of a circuit group supervision control confirmation message) the application may begin exchanging call traffic with the SIU.

#### 6.2.3.9 Circuit Group Conflict

Both SIUs in a dual resilient configuration periodically poll each other to determine which circuit groups are active on each unit. If a group is active on both units at the same time, an API_MSG_USER_EVENT message is issued by the unit that detects the conflict, indicating the group ID of the affected circuit group. The controlling application host should issue a deactivate command to the SIU that should not be controlling the circuit group and resynchronize the circuits in the group (on the correct SIU) by issuing a reset.

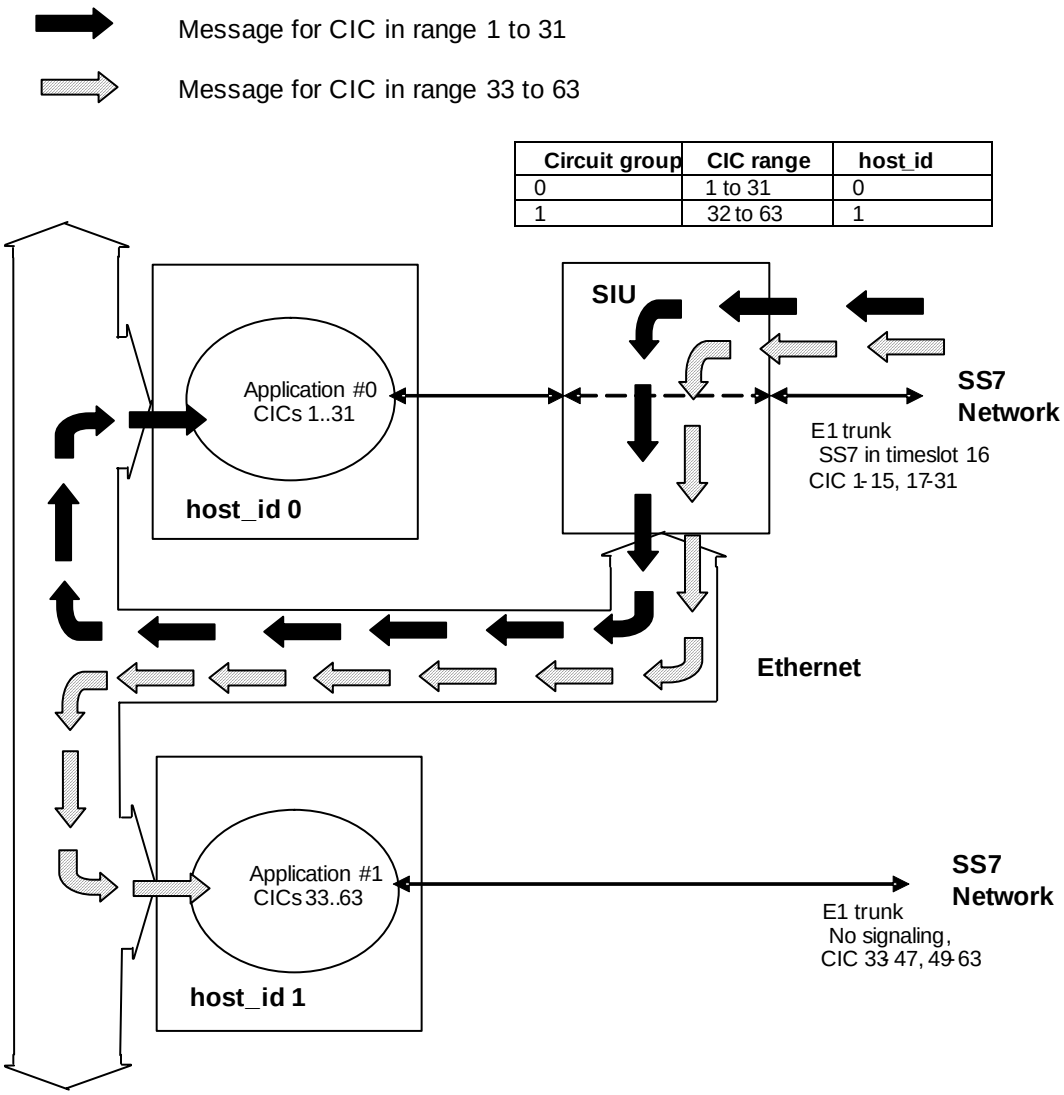
The SIUs prevent this situation from arising by automatically sending a deactivate circuit group command to the other unit on receipt of the activate command. If the nature of the failure is such that inter-SIU communication is lost, it may be impossible for the SIU to issue the automatic deactivate command. In this case, when the failed SIU recovers, circuit groups may be active from the time before the initial failure. This situation is handled by the application sending a deactivate command for all of the previously active groups immediately following restoration of the communication with the SIU.

## 6.3 System Operation

### 6.3.1 Telephony API Operation

The message based API operates transparently over TCP/IP Ethernet, using software drivers provided by Dialogic. For Telephony (circuit-switched applications), each application platform terminates and hence controls a fixed range of physical circuits, or Circuit Identification Codes (CICs). CICs are configured in groups of up to 32, each group equating to all the circuits in a single E1 or T1 bearer. Each group is terminated on a fixed application platform or host processor, enabling the SIU to automatically direct API messages to the correct platform.

**Figure 27. Receive Message Flow for a Two-Host System**




Each application platform or host is uniquely identified with a host identifier or host_id. The SIU architecture provides the ability to configure up to 128 hosts, with a host_id range of 0 to 127. Each circuit to a particular destination is uniquely identified by a Circuit identification Code (CIC). Internally, the SIU maps a local logical circuit reference, cid to a CIC and destination, cid values being unique to each SIU installation. (CIC values may be repeated where routing is possible to more than one destination).

The circuits are configured on the SIU in blocks or “circuit groups” of up to 32, each block corresponding to all the circuits in a single T1 or E1 PCM trunk. Each circuit group is assigned a host_id, allowing the received messages for the circuits in the group to be issued to the correct application platform as shown in Figure 30.

### 6.3.2 Programming Model

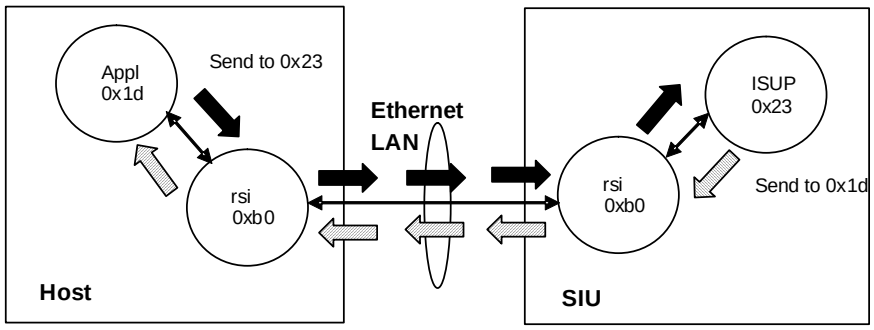
The SIU programming model is based on tasks (or processes), each identified by a unique 8-bit number or **module_id**, and the message queues that are used for inter-task communication.

A task communicates with another by sending a message (MSG) to the message queue identified by the module_id of the destination task. The inter-task communication is managed by a program “**gctload**” and statically configured by a text file “**system.txt**”. These files are present on all platforms and operating systems supported by the SIU host software.

For example, the ISUP SS7 protocol layer is implemented as task 0x23 (**ISP_TASK_ID**) running on the SIU. Hence, when the application (which has its own module_id) wishes to make an outgoing call, this is achieved by sending a message (containing set-up request parameters) to destination module 0x23.

The system.txt file provides the ability to define a local message queue using the **LOCAL** keyword, indicating that the named task is running on the local platform, hence any messages sent to this task should be queued locally. Messages to a destination may be intercepted or redirected to an inter-platform driver to allow a message to be passed to a queue that exists on a physically different platform.

**Figure 28. Redirecting Messages between ISUP and the Application**




In the SIU environment, the **rsi** task, which runs as module_id 0xb0 manages communication via the LAN between the SIU and each host application platform. Hence, from the host's viewpoint, the ISUP module runs on a physically remote machine, and any messages sent to ISUP must therefore be redirected via rsi to the SIU. This is achieved with a **REDIRECT** statement as follows:

```
LOCAL 0xb0  * Local rsi message queue
REDIRECT 0x23 0xb0* Redirect ISUP message via rsi to SIU
```

The system.txt file provides a third feature, the ability to start a task using a **FORK_PROCESS** statement.

### 6.3.3 Connecting a Host

The connection between the host and the SIU (and also an SIU pair) is managed by the rsi task. A connection between a host platform and an SIU is started using the rsicmd utility as described in the in section Section 3.3, “Application Operation” on page 30, of this manual, for example:

```
rsicmd 0 0xef 0 192.168.0.1 9000
```

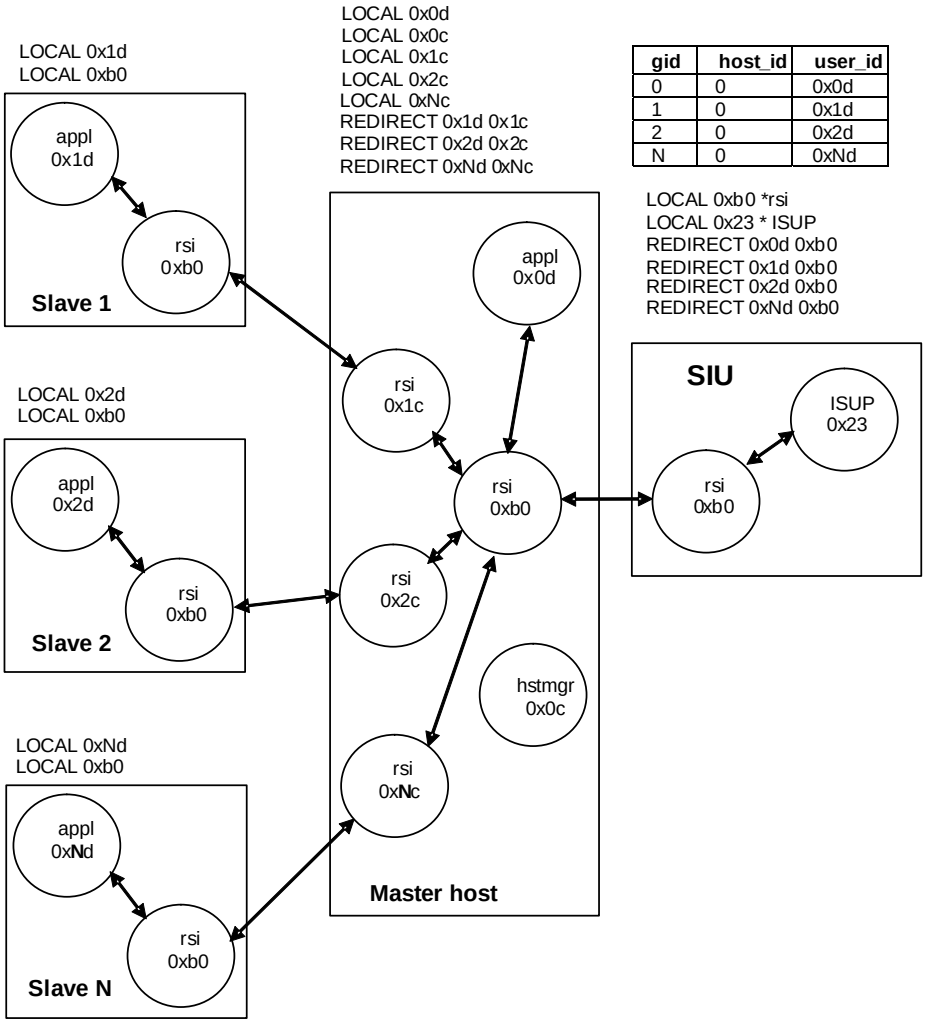
This connects a host to an SIU that has an IP address of 192.168.0.1 using port 9000, identifying this host as host_id 0. Hence, all messages for circuits configured in groups assigned to host_id 0 will be issued by the SIU to this platform.

The 0xef parameter indicates which local task should be informed of changes in state of the connection between the host and the SIU.

### 6.3.4 Clustering Host Platforms

Host clustering extends the use of the rsi task and the REDIRECT functionality to increase the SIU capability beyond the 64-application platform limit. In addition to **host_id**, it is also necessary to specify the destination module id (or **user_id**) used for all incoming indications for each circuit group. It is therefore possible to direct messages for a number of circuit groups to the same host, each group having a different destination module identifier. If the destination module identifier is declared as LOCAL on the master host platform, messages sent to this module id will be processed locally. However, it is possible to redirect the destination to a rsi task controlling a link via Ethernet to a slave host platform, as shown in Figure 32 on page 95.

**Figure 29. Message Redirection in Host Clustering**




Each slave platform terminates E1 or T1 PCM trunks in the same way as the master host, which may itself terminate PCM trunks and process voice circuits. The figure above shows that for each slave host connected to the master, an additional rsi task is started.

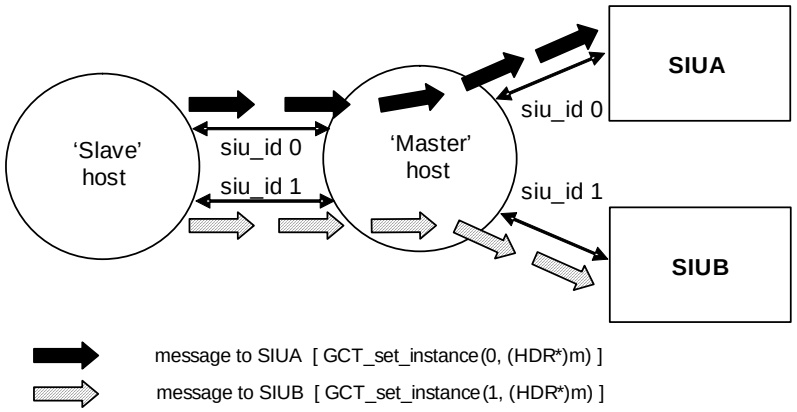
The links between the master and slave hosts are started using the rsicmd utility in a similar manner to that used to start the connection between the master host and the SIU.

### 6.3.5 Dual SIU Operation

Dual SIU operation is achieved by implementing two rsi connections between each slave and master host, identified by siu_id values 0 and 1. The application directs messages to SIUA and SIUB using the GCT_set/get_instance library functions in the same manner as a system that does not use master and slave hosts.

A message sent from a slave to SIUA should be directed to instance 0 and will travel down links assigned siu_id value 0. A message sent from a slave to SIUB should be directed to instance 1 and will travel down links assigned siu_id value 1, as shown in Figure 33 on page 96.

**Figure 30. Directing Messages to SIUA and SIUB**




# 6.4 Configuration Parameters

## 6.4.1 Circuit Group Configuration for Host Clustering

Circuit groups are configured using the same method for a system that does not implement host clustering, with the addition of identifying the user application module id for each circuit group. The complete circuit group configuration syntax is shown below, where xxx is either ISUP or TUP.

```
xxx_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options> <host_id> <user_id> <opc> <ssf>
```

The **host_id** parameter uniquely identifies the host cluster, the user_id uniquely identifies each member within the cluster.

The values that must be used for the **user_id** parameter are shown in the following table.

| Host Platform | User_id |
|---|---|
| Master | 0x0d |
| Slave #1 | 0x1d |
| Slave #2 | 0x2d |
| Slave #N | 0xNd |

## 6.4.2 Configuring the Master Host

Each rsi task on the master host takes a unique module id within this host cluster, this must be declared in the system.txt file on the master host with a LOCAL definition and started with a **FORK_PROCESS** command. The rsi program takes its module id as an optional command line parameter, prefixed by "-m", for example:

```
./rsi -m0xc0
```

A **REDIRECT** statement must also be inserted in the system.txt file for the API messages sent from the SIU to the slave host platforms.

| rsi task | rsi module id | FORK_PROCESS | REDIRECT |
|---|---|---|---|
| To SIU | 0xb0 | rsi -r.\RSI_LNK.EXE -l1 | None |

| rsi task | rsi module id | FORK_PROCESS | REDIRECT |
|---|---|---|---|
| To slave #1 | 0x1c | rsi -r.\RSI_LNK.EXE -l1 -m0x1c | REDIRECT 0x1d 0x1c |
| To slave #2 | 0x2c | rsi -r.\RSI_LNK.EXE -l1 -m0x2c | REDIRECT 0x2d 0x2c |
| To slave #**N** | 0x**N**c | rsi -r.\RSI_LNK.EXE -l1 -m0x**N**c | REDIRECT 0x**N**d 0x**N**c |

If a slave platform is not present, the declarations listed in the table should not be made within the system.txt file.

The rsi links between the master and each slave host is activated using the rsicmd utility. The syntax for the rsicmd utility is shown below.

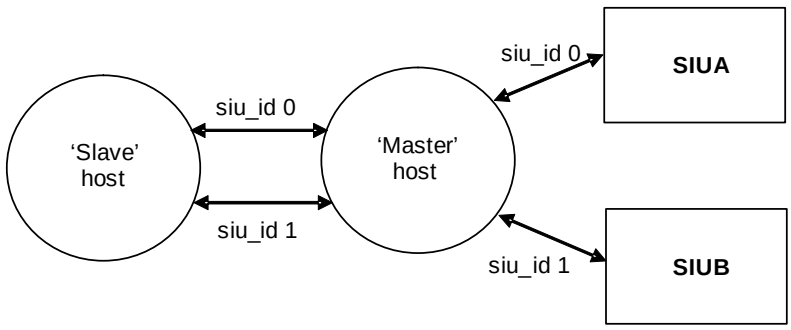
```
rsicmd <siu_id> <conc_id> <link_type> <rem_addr> <rem_port> [<rsi_id>]
```

<**siu_id**> this parameter assigns the destination of a connection as being either SIUA or SIUB and is the 'instance' value used by the GCT_set_instance/GCT_get_instance functions when directing a message to either SIUA or SIUB in a dual SIU system. The values should be set as shown in the following table.

| siu_id | Link |
|---|---|
| 0 | Between host and SIUA |
| 1 | Between host and SIUB |

For a single SIU system, only siu_id 0 will be present. For a dual SIU system, both siu_id values 0 and 1 will be present between the master host and SIUA and SIUB and also between the master host and each slave host, as shown in Figure 34.

**Figure 31. Use of siu_id values**




<**conc_id**> specifies a module ID that will receive a message whenever the specified rsi connection fails. For the connection to the SIU, this must be set to 0x0c, the module identifier assigned to the hstmgr program. For the other links, this should be set to 0xb0 (the module identifier of the rsi controlling the link to the SIU).

<**link_type**> and <**rem_addr**> should be set according to the following table.

| Connection Type | rem_addr | link_type value |
|---|---|---|
| Master host to SIU | IP address of SIUA or SIUB | 0 (client) |
| Master to slave host | 0 | 1 (server) |

<**rem_port**> specifies the TCP/IP socket port that used for the connection. For master host to SIU connections, this port value uniquely identifies each host and corresponds to the host_id parameter held within the SIU parameter file. Each slave connection must take a unique port value, starting from 9000.

<**rsi_id**> is optional and identifies the rsi module and hence the slave link that will receive the activation command. For links from the master host to the SIU, this parameter may be omitted and the default rsi module id of 0xb0 will be used. For the remaining rsi links, this parameter must be set to the module identifier of the rsi task that is managing the connection.

A summary of the rsicmd parameters are shown in the table below.

| Destination | siu_id | conc_id | link_type | rem_addr | rem_port | rsi_id |
|---|---|---|---|---|---|---|
| SIUA | 0 | 0x0c | 0 | SIUA IP address | 9000 + host_id | (0xb0) |
| SIUB | 1 | 0x0c | 0 | SIUB IP address | 9000 + host_id | (0xb0) |
| Slave #1 link A | 0 | 0xb0 | 1 | Slave #1 IP address | 9000 | 0x1c |
| Slave #1 link B | 1 | 0xb0 | 1 | Slave #1 IP address | 9001 | 0x1c |
| Slave #2 link A | 0 | 0xb0 | 1 | Slave #2 IP address | 9002 | 0x2c |
| Slave #2 link B | 1 | 0xb0 | 1 | Slave #2 IP address | 9003 | 0x2c |
| | | | | | | |
| Slave #N link A | 0 | 0xb0 | 1 | Slave #N IP address | 90xx | 0xNc |
| Slave #N link B | 1 | 0xb0 | 1 | Slave #N IP address | 90xx + 1 | 0xNc |

#### 6.4.2.1 The hstmgr (Host Manager) Program

To ensure that the link status between the master host and the SIU is conveyed correctly to each slave host platform, and additional task, hstmgr, is required on the master host only. This task also ensures that congestion is handled correctly between the rsi tasks that exist within the system.

This program takes several command line parameters, the syntax is shown below.

```
hstmgr -n<num_slave> -c<conc_id> [-m<module_id>]
```

<**module_id**> specifies the module identifier used by this task. If omitted, the default identifier of 0x0c used. This identifier must also be defined as LOCAL within the system.txt file.

<**num_slave**> specifies the number of slave application platforms that are connected to this master host, in the range 1 to 3.

<**conc_id**> specifies a local module ID that will receive a message whenever the rsi link fails. This module should exist on the master host, such that when these status messages are issued by rsi, they are received and then released by this module.

For example, in a system with two slave hosts which requires a module 0xef to be informed of the availability of the connection to the SIU, the command line would be:

```
hstmgr -n2 -c0xef
```

The hstmgr program must be informed of state changes in the Ethernet link between the master host and the SIU, hence the rsicmd entry for starting the master host to SIU link(s) must specify the module identifier of hstmgr as the conc_id. For example, to connect as host cluster 0 to SIUA with an IP address of 192.168.0.1, the following command would be used:

```
rsicmd 0 0x0c 0 192.168.0.1 9000
```

### 6.4.3 Configuring the Slave Host

The slave host is configured almost identically to a standard SIU host (in a system that does not employ host clustering).

An application program running on host N should use the module identifier 0xNd, which must be declared in the system.txt file with a LOCAL definition. The first slave host is slave 1, hence the application must use the module identifier 0x1c.

The rsi connection between the slave and the master host is activated using the rsicmd utility. This must specify the same rem_port value used by the master host on this link.

## 6.5 Example Configuration

This section presents the system.txt configuration files for each node in a system consisting of a single master host serving three slave hosts. Each host processes 60-voice circuits carried in two E1 PCM trunks (each host therefore manages two circuit groups).

The SIUs are deployed in a dual fault tolerant configuration. The IP address of SIUA is192.168.0.1, SIUB 192.168.0.2 and the master host 123.234.345.458.

**Figure 32. Logical View of Clustered Host System**

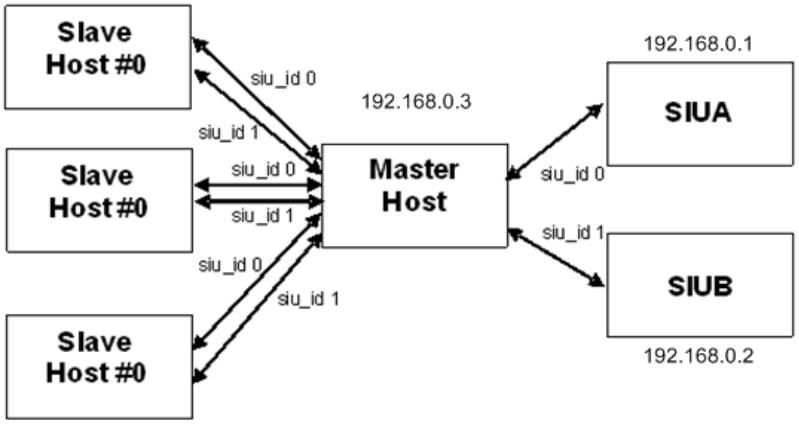
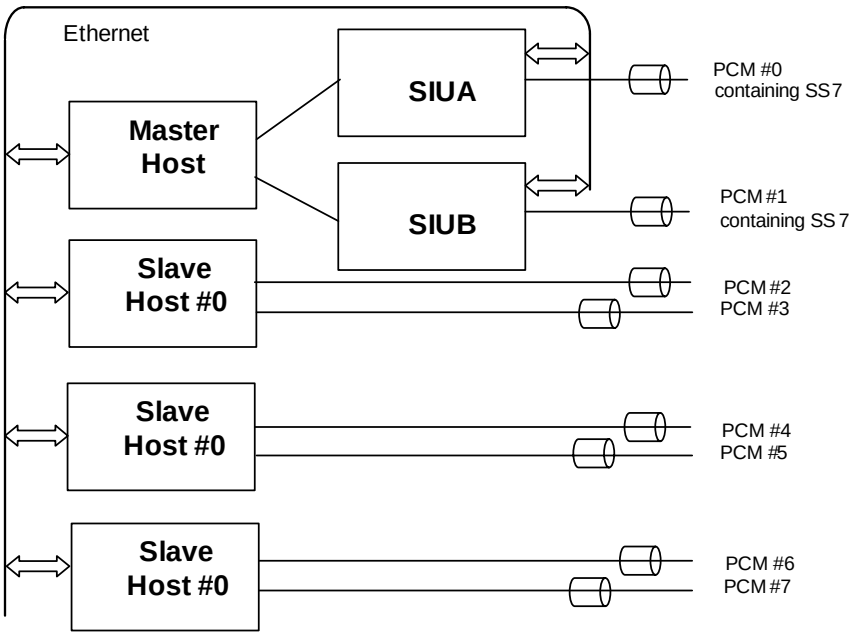



**Figure 33. Physical View of a Clustered Host System**



### For the master host:

```
*
* Module Id's running locally on the host machine:
*
LOCAL0xb0* rsi Module Id to SIUA/SIUB
LOCAL0x0c* hstmgr (Master only)
LOCAL0x1c* rsi Module Id to Slave #0
LOCAL0x2c* rsi Module Id to Slave #1
LOCAL0x3c* rsi Module Id to Slave #2
LOCAL0xef* REM_API_ID Module Id (s7_log)
LOCAL0xfd* rsicmd Module Id
LOCAL0x0d* Local application task Module Id
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT0xdf0xb0* SIU_MGT module Id
REDIRECT0x220xb0* MTP3 module Id
REDIRECT0x140xb0* TCAP module Id
REDIRECT0x330xb0* SCCP module Id
REDIRECT0x320xb0* RMM module Id
REDIRECT0x230xb0* ISUP module Id
REDIRECT0x4a0xb0* TUP/NUP module Id
*
* Redirection to slave host platforms
*
REDIRECT 0x1d0x1c* To slave #0
REDIRECT 0x2d0x2c* To slave #1
REDIRECT 0x3d0x3c* To slave #2
*
* Start-up the master host tasks ....
*
FORK_PROCESS.\s7_log.exe
FORK_PROCESS.\hstmgr.exe -n3 -c0xef
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1 -m0x1c
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1 -m0x2c
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1 -m0x3c
*
* Start the Master host to SIUA/SIUB rsi connection
*(note that hstmgr is the 'conc_id')
*
FORK_PROCESS.\rsicmd.exe 0 0x0c 0 192.168.0.1 9000
FORK_PROCESS.\rsicmd.exe 1 0x0c 0 192.168.0.1 9000
*
* Start the master to slave host links
*
FORK_PROCESS.\rsicmd.exe 0 0xb0 1 0 9000 0x1c
FORK_PROCESS.\rsicmd.exe 1 0xb0 1 0 9001 0x1c
FORK_PROCESS.\rsicmd.exe 0 0xb0 1 0 9002 0x2c
FORK_PROCESS.\rsicmd.exe 1 0xb0 1 0 9003 0x2c
FORK_PROCESS.\rsicmd.exe 0 0xb0 1 0 9004 0x3c
FORK_PROCESS.\rsicmd.exe 1 0xb0 1 0 9005 0x3c
*
* Start example 'telephony' application :
*
* FORK_PROCESS.\ctu.exe -m0x0d -o0x1fff
```

### For the first slave host (slave #1):

```
*
* Module Id's running locally on the host machine:
*
LOCAL0xb0* rsi Module Id to master host
LOCAL0xef* REM_API_ID Module Id (s7_log)
LOCAL0xfd* rsicmd Module Id
LOCAL0x1d* Local application task Module Id
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT0xdf0xb0* SIU_MGT module Id
REDIRECT0x220xb0* MTP3 module Id
REDIRECT0x140xb0* TCAP module Id
REDIRECT0x330xb0* SCCP module Id
REDIRECT0x320xb0* RMM module Id
REDIRECT0x230xb0* ISUP module Id
REDIRECT0x4a0xb0* TUP/NUP module Id
*
* Start-up slave host tasks ....
*
FORK_PROCESS.\s7_log.exe
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1
*
* Start the slave to master host rsi connections
*
FORK_PROCESS.\rsicmd.exe 0 0xef 0 192.168.0.3 9000
FORK_PROCESS.\rsicmd.exe 1 0xef 0 192.168.0.3 9001
*
* Start example 'telephony' application :
*
* FORK_PROCESS.\ctu.exe -m0x1d -o0x1fff
```

### For the second slave host (slave #2):

```
*
* Module Id's running locally on the host machine:
*
LOCAL0xb0* rsi Module Id to master host
LOCAL0xef* REM_API_ID Module Id (s7_log)
LOCAL0xfd* rsicmd Module Id
LOCAL0x2d* Local application task Module Id
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT0xdf0xb0* SIU_MGT module Id
REDIRECT0x220xb0* MTP3 module Id
REDIRECT0x140xb0* TCAP module Id
REDIRECT0x330xb0* SCCP module Id
REDIRECT0x320xb0* RMM module Id
REDIRECT0x230xb0* ISUP module Id
REDIRECT0x4a0xb0* TUP/NUP module Id
*
* Start-up slave host tasks ....
*
FORK_PROCESS.\s7_log.exe
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1
*
* Start the slave to master host rsi connections
*
FORK_PROCESS.\rsicmd.exe 0 0xef 0 192.168.0.3 9002
FORK_PROCESS.\rsicmd.exe 1 0xef 0 192.168.0.3 9003
*
* Start example 'telephony' application :
*
* FORK_PROCESS.\ctu.exe -m0x2d -o0x1fff
```

**For the third slave host (slave #3):**

```
*
* Module Id's running locally on the host machine:
*
LOCAL0xb0* rsi Module Id to master host
LOCAL0xef* REM_API_ID Module Id (s7_log)
LOCAL0xfd* rsicmd Module Id
LOCAL0x3d* Local application task Module Id
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT0xdf0xb0* SIU_MGT module Id
REDIRECT0x220xb0* MTP3 module Id
REDIRECT0x140xb0* TCAP module Id
REDIRECT0x330xb0* SCCP module Id
REDIRECT0x320xb0* RMM module Id
REDIRECT0x230xb0* ISUP module Id
REDIRECT0x4a0xb0* TUP/NUP module Id
*
* Start-up slave host tasks ....
*
FORK_PROCESS.\s7_log.exe
FORK_PROCESS.\rsi.exe -r.\rsi_lnk.exe  -l1
*
* Start the slave to master host rsi connections
*
FORK_PROCESS.\rsicmd.exe 0 0xef 0 192.168.0.3 9004
FORK_PROCESS.\rsicmd.exe 1 0xef 0 192.168.0.3 9005
*
* Start example 'telephony' application :
*
* FORK_PROCESS.\ctu.exe -m0x3d -o0x1fff
```

The config.txt file on SIUA and SIUB would contain circuit group definitions similar to the following, where xxx is either ISUP or TUP:

```
*   Define xxx circuit (groups) :
*   xxx_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options> <host_id> <user_id> <opc>
<ssf>
*
xxx_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0003 0x00 0x0d 2 0x8
xxx_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x0003 0x00 0x0d 2 0x8
xxx_CFG_CCTGRP  2  1  0x41  0x41  0x7fff7fff  0x0003 0x00 0x1d 2 0x8
xxx_CFG_CCTGRP  3  1  0x61  0x61  0x7fff7fff  0x0003 0x00 0x1d 2 0x8
xxx_CFG_CCTGRP  4  1  0x81  0x81  0x7fff7fff  0x0003 0x00 0x2d 2 0x8
xxx_CFG_CCTGRP  5  1  0xa1  0xa1  0x7fff7fff  0x0003 0x00 0x2d 2 0x8
xxx_CFG_CCTGRP  6  1  0xc1  0xc1  0x7fff7fff  0x0003 0x00 0x3d 2 0x8
xxx_CFG_CCTGRP  7  1  0xe1  0xe1  0x7fff7fff  0x0003 0x00 0x3d 2 0x8
*
*
```

## 6.6 Frequently Asked Questions

**How can a slave host tell if an SIU has failed?**

The link between the slave and master host that carries traffic to the failed SIU will fail (this is achieved by the hstmgr task on the master host), indicated by a RSI_MSG_STATUS link down indication presented to the locally concerned module.

**What happens if the master host fails?**

The slave hosts will lose communication with both SIUA and SIUB.

**How many hosts can such a system support?**

The maximum number of master hosts for a SIU is 64. Each master host can support up to three slave hosts (hence the total for SIU is 256).

**How can I tell if an SIU fails?**

The status of the communication between the host and the SIU is indicated by RSI_MSG_LNK_STATUS messages.

**What happens if an SS7 message for a circuit or transaction is received by an SIU that does not control that circuit or transaction?**

The message is automatically passed to the partner SIU using the TCP/IP LAN.

**If a single SS7 link to the network fails on one of the SIUs, is any action required?**

No. If there are other links remaining on that unit, the traffic will changeover to one of these; otherwise, the traffic is automatically passed to the other unit via the inter-SIU SS7 link set.

**If all of the SS7 links to one of the SIUs fail, is any action required?**

No. The SIU automatically re-routes transmit traffic via the inter-SIU SS7 link set.

**If an SIU fails, is any action required?**

Yes. For switched-circuit applications, the circuit groups controlled by the failed unit should be activated on the remaining unit. Details are provided in this manual.

**What happens if an inter-SIU SS7 link fails?**

The SIU will changeover to use other SS7 links in the inter-SIU link set.

**What happens if the Ethernet interfaces fail on an SIU?**

This causes failure of all of the connections between the hosts and the SIU. The SIU reacts by deactivating all connected SS7 links, preventing any more signaling information from being received from the SS7 network. The host applications receive an indication of failure of the SIU and should transfer any circuit groups controlled by this SIU to the remaining unit, the effective result being as though the complete unit had failed.

**What causes No System Resources alarm and how can I rectify the situation.**

The No System Resources alarm has been observed immediately following system startup and indicates that the unit has not yet completed internal startup up procedures correctly - therefore it cannot process any MMI commands. Although the condition should eventually clear itself, you can restart the unit to rectify the situation.

**The alarm log indicates a PSU failure alarm, but the LED on the back of the affected PSU is “green”.**

The PSU may have genuinely failed or may be incorrectly seated or may be operating outside of its normal operating ranges for input voltage and temperature. The LED on the rear of the PSU is not currently supported or used.

**What causes the System Overload condition and how can it be resolved.**

The most common cause of system overload is that messages are being queued for a user application which is unable to process these at a sufficient rate, therefore the number of outstanding messages exceeds the congestion thresholds set on the gctload command line. The gctload -t1 and -t2 options can assist in identifying modules accumulating messages within the system.

**Can the Signaling Server be employed in dual mode - paired to an SS7G3x.**

Yes the Signaling Server can be paired with the **SS7G3x** as a dual pair.

**What is the CPU Warning alarm.**

The CPU warning indicates that the identified CPU is operating above or below its expected operating temperature. This may eventually lead to CPU failure. You should verify that the system is correctly sited, that input voltages are correct and that the unit has adequate cooling and ventilation.

**I cannot communicate with the unit using SSH.**

The unit does not provide a Secure Shell session connection. Your SSH client may need additional configuration to allow SSH tunneling without a session connection.

**The PCM status is cycling between PCM OK and SYNC LOSS - what does this mean.**

If the unit cannot synchronize properly with an adjacent switch it will abandon synchronization and attempt to regain it again - causing the PCM to cycle between OK and SYNC LOSS. Check the clocking in the network configuration, it is usual, although not always the case, that the Signaling Server will be configured to receive the clocking signal on an E1/T1 connected to an adjacent switch. Also check the clocking configuration on the unit board configuration and the frame format and syncpri parameters on the LIU configuration.

# Glossary

| | |
|---|---|
| A-link | An “access” link that connects a signaling end point (for example, an SCP or SSP) to an STP. Only messages originating from or destined to the signaling end point are transmitted on an A-link. |
| AIS | Alarm Indication Signal (Blue alarm). |
| BER | Bit Error Rate. |
| blink | The index of the logical signaling processor (SP) channel (on the board) allocated for this signaling link. For a link on a SS7LD board the blink is a value between 0 to 3 and for a SS7MD board the blink is a value in the range 0 to 123. When the SS7 link is to be conveyed over M2PA, the blink parameter identifies the SIGTRAN link and is in the range 0 to 255. |
| CCITT | Consultative Committee on International Telegraphy and Telephony |
| config.txt | A text file used for protocol configuration. |
| CPU | Central Processing Unit |
| CSSR | A concerned SCCP sub-system resource, that is, a sub-system resource that wants to receive state change information about another SCCP sub-system or signaling point. |
| ctu | An example program that demonstrates how a user application can interface with a telephony user part, such as ISUP. |
| DPC | Destination Point Code. Identifies the address (point code) of the SS7 network node to which a Message Signal Unit (MSU) should be directed. |
| DSI | Distributed Signaling Interface. |
| dual resilient | A term used to describe a system that consists of two SIUs configured as a single point code in the SS7 network. Under normal circumstances, both SIUs share the load. If one unit fails, the partner unit maintains operation of the node. |
| F-link | An “fully-associated” link that connects two signaling end points (for example, SSPs and SCPs). F-links are not usually used in networks with STPs. In networks without STPs,<br>F-links directly connect signaling points. |
| FTP | File Transfer Protocol |
| MAP | Mobile Application Part (MAP). An SS7 stack layer supporting messages sent between mobile switches and databases to support user authentication, equipment identification, and roaming. |
| MTP | Message Transfer Part. Layers 1 to 3 of the SS7 protocol stack broadly equivalent to the Physical, Data Link and Network layers in the OSI protocol stack. See also MTP1, MTP2, and MTP3. |
| MSU | Message Signal Unit. A data unit that carries signaling information for call control, transaction processing, network management and maintenance. Typically, the MSU is carried in the Signaling Information Field (SIF) of SS7 messages. |
| gctload | A program that handles the initialization sequence and creates inter-process communication. |
| INAP | Intelligent Network Application Part. An SS7 stack layer that defines the messages and protocol used to communicate between applications (deployed as subsystems) in SS7 nodes. INAP uses the Transaction Capabilities Part (TCAP). See TCAP below. |
| IS41 | An ANSI signaling standard used in mobile networks. |
| ISUP | ISDN User Part. A SS7 stack layer that defines the messages and protocol used in the establishment and tear down of voice and data calls over the public switched network, and to manage the trunk network on which they rely. |
| LIU | Line Interface Unit. |
| Link | A physical and logical connection between two signaling points. |
| Link set | One or more signaling links that are connected to adjacent signaling points. |
| mtpsl | An example utility that can also be used to activate and deactivate signaling links. |
| OPC | Originating Point Code. A signaling point code that identifies the signaling point at which a message originated. |

| | |
|---|---|
| RAI | Remote Alarm Indication (Yellow alarm). |
| route | An MTP3 concept that determines how signaling is distributed over link sets. A route consists of a destination point code and the link set ID of one or two link sets over which traffic to the destination node should be routed. When two link sets are provided, you can choose to load share traffic or treat the link sets as primary and secondary. |
| rsi | A process manages the connection between the host and each SIU. It takes several command line parameters and is normally spawned by an entry in the host's system.txt file. |
| rsicmd | A command that starts the Ethernet link between a host and an SIU. |
| s7_play | A utility that can be used to generate messages from a text file and send them to the system. Typically used for diagnostic purposes. |
| s7_log | A utility that enables messages received from the protocol stack to be logged in a text file. Typically used for diagnostic purposes. |
| SCCP | Signal Connection Control Part. An SS7 stack layer that allows a software application at a specific node in an SS7 network to be addressed. |
| SIU | Signaling Interface Unit |
| SP | Signaling Processor |
| SP channel | The logical processing channel, within the signaling processor hardware, that conducts the processing of a signaling link. |
| SS7 | Signaling System Number 7 |
| SS7MD | SS7MD  Short form of the name for  Dialogic® DSI SS7MD Network Interface Board. |
| SS7 Protocol Stack | A set of software modules that implement the various layers of the SS7 protocol stack. |
| SSH | Secure Shell |
| SSP | Service Switching Point |
| STP | Signaling Transfer Point |
| SSR | An SCCP sub-system resource. This can be a local sub-system, a remote sub-systems or a remote signaling point. |
| system.txt | A text file used for system configuration. |
| TCAP | Transaction Capabilities Application Part. An SS7 stack layer that enables the deployment of intelligent network and mobile services by supporting non-circuit related information exchange between signaling points using the SCCP connectionless service. |
| ttu | An example program that demonstrates how a user application can interface with the TCAP protocol module. |
| timeslot | The smallest, switchable data unit on a TDM bus. For T1 and E1 technologies, one time slot is equivalent to a data path with a bandwidth of 64 Kbps. |
| upe | A worked example of exchanging messages with the MTP3 module. |