Cover Page

Standard

			1 page of 17
TITAN 532L Cross Connect Generator		A	5/22/00
Subject	File Code	Rev.	Effective Date

Rev	Change Description	Date - Originator
AP1	Initial Version	5/22/00 - EPW
A	First Official Version	8/22/00 - EPW

1. Introduction

Introduction

The Cross Connect Generator (CCG) is a tool which helps users to provision a TITAN 532L digital cross connect system.

Purpose

The purpose of this document is to outline all of the requirements necessary to design, implement and test the CCS.

Scope

This document covers the requirements phase of the CCS life cycle.

Field of Application

This document is written for the Tellabs development and support groups, Dr. Aditya Mathur and the students of Purdue University’s CS406 / CS407 classes.

Revision History

Revision	Date	Revision Description
AP1	05/22/00	Initial Version
A	8/22/00	First Official Version

Definitions

Term	Definition
Port Module	One physical card that contains one or more NPCs.
Channel	A channel is one of the individual DS0s on an NPC. There are 24 channels on one NPC (or DS1).
port	Usually refers to a single DS1.
Feature Package	A feature package refers to the major, minor and revision level of software. The 532L is sold by feature package.

Acronyms and Abbreviations

Term	Definition
DS0	Digital Signal 0 - 64 Kb/s - one phone call
DS1 / T1	Digital Signal 1 - 1.544 Mb/s - 24 DS0s
E1	European Signal 1 - 2Mb/s - 32 DS0s.
DS3 / T3	Digital SIgnal 3 - 44.736 Mb/s - 28 DS1s
STS-1E	Synchronous Transport Signal - 51.84Mb/s - 28 DS1
PSC	Port Shelf Complex

Term	Definition
NPC	A generic term usually referring to one DS1 on a port module
SDF	Standard Data Format

Requirements Overview

When dealing with system requirements it is important to remember several key concepts:

- Requirements are driven by the product and not necessarily vice versa.
- Requirement change when it is most inconvenient.
- There are few wrong answers but some answers are more right than others.
- Some requirements can be a wish list.

2. The TITAN 532l Overview

Introduction

The TITAN 532L Digital Cross Connect System is a telecommunications network element that is used to transport, monitor and groom telephone traffic. The 532L is considered a narrow band cross connect because it only makes connections at the DS0, DS1 and DS3 levels (as opposed to a broadband cross connect like the TITAN 5500 cross connect which supports DS1, DS3, OC3, and OC12).

While this may seem to limit the usefulness of the 532L, there is a specific market for narrowband cross connects. A major use of the 532L is to provide the data paths for small networks. A major application of the 532L is to connect cell sites together. It is also used to connect ATMs (Automatic Teller Machines) to a bank. Another application is in 911 call centers. The 532L connects police, fire and hospitals together with the call center. It is also found in lottery networks connecting individual terminals to the main lottery computers.

The 532L currently supports up to 7,168 DS1s, 256 DS3s or 256 STS-1Es or any combination thereof. This is equivalent to 172,032 phone calls.

532L Architecture

There are three major systems (complexes) on the 532L. The first is the Administrative Complex (AC). The AC provides the user interface, non-volatile data storage for the system configuration, is a LAN hub for the rest of the major processors in the system and provides timing for the cross connects in the system.

The second system is the Network Switch Complex (NSC). The major functionality of the cross connect is to take data from point A and transport it to point B. The NSC is responsible for finding a path from A to B within three of the five stages of the switch network.

The third system is the Port Switch Complex (PSC). The PSC has two major responsibilities. Its first major duty is to provide an interface to the customer’s network. A PSC is comprised of many port modules. The data is connected to the PSC at the port modules. The second major duty of the PSC is to switch 2 of the 5 stages of the network.

Standard Data Format

The 532L supports many different kinds of customer traffic, DS1, DS3, STS-1E, etc. It would be impossible to cross connect them together if they were not converted to a single, common data type. The Standard Data Format (SDF) is used for this purpose. When data comes into the 532L at the PSC it is converted to SDF. As the data travels through the PSC and NSC it can be shuffled around in a single SDF or moved between SDFs depending on how it is cross connected. When the data finishes its journey through the network and has been cross connected it is once again converted to a format that the customer is expecting and is sent back out of the system. The 532L currently supports up to 256 SDFs. A PSC can support up to 16 SDFs.

3. The Port Shelf Complex

Introduction

While it is useful to understand the architecture of the entire 532L. The CCG is most concerned with the Port Shelf Complex. The PSC is comprised of two parts. There is the network portion and the port module portion. The network portion of the PSC is responsible for the first and last stage of the switching network.

The second (and more relevant) portion of the PSC is the port module. There are many different kinds of port modules. Each PSC supports one or more port module type but no PM type is supported by more than one PSC. Depending on the type, a PSC supports anywhere from 16 to 256 port modules.

Originally, the term port module referred to a single DS1 card. That term has evolved over time and now refers to a single module on a PSC. The module may be a single DS1 card, 8 DS1 card, DS3, STS-1E, etc. NPC is used to refer to a DS1 (or equivalent) on a port card.

Furthermore, every DS1 signal is comprised of 24 DS0's. These DS0's are referred to as channels.

As an example, 1 DS3 port module is comprised of 28 NPC's (DS1's) with 24 channels (DS0's) per NPC for a total of 672 channels.

Every PSC uses some number of SDFs to transport customer data. This means that each port module uses some fraction of those SDFs. There is a direct relationship between the SDF number, PSC number and port module number. Understanding this relationship is essential to implementing the CCG. This will be explained in greater detail in later sections and in additional documentation.

PSC Numbering

To be able to uniquely identify each PSC in a 532L it is given a number to identify its position in the network. There are two methods of doing this. The first method is to number each PSC sequentially. Each PSC will have a number between 1 and 16. This is known as sequential numbering. This is the numbering method for feature packages FP1.0 to FP5.0.

The second method is called SDF numbering. Each PSC's number will be its starting SDF number in the network. The valid range of SDFs is 1 to 256. This method is only valid for feature packages FP6.0 and later.

Even though development has replaced sequential numbering with SDF numbering both methods must be supported. Various testing, support and development groups routinely need to create cross connects on older feature packages and still need the PSC numbering method.

Continued on next page

3. The Port Shelf Complex (cont.)

Low Density PSC
LD100

The low density PSC type (LD100) was the first PSC type available for the 532L. It uses 16 SDFs and provides an interface to 256 DS1 modules, 8 DS3 modules or a combination of the two. This PSC is structured so that 16 PM's are seated in a single shelf and 3 DS3 modules are in a single shelf. Each PM shelf (with 16 PM cards) is one SDF.

Table 3-1 Low Density Port Modules

Name	Interface	Description
PM	1 DS1	Port Module - This card has a single DS1 interface. It is the oldest card supported by the 532L.
DPT1	2 DS1	Dual Port T1 - This card has two DS1 interfaces. As far as the CCG is concerned there is no distinction between the DPT1 and the PM.
DS3	1 DS3	DS3 - This card provides an interface to a single DS3.
Subrate	DS0	Subrate - The subrate cards are used for DS0 (or smaller) applications. They are not frequently used and are not supported by the CCG.

High Speed PSC
HS100

The high speed PSC type (HS100) is a single shelf which supports 16 SDFs. It provides an interface to 16 STS-1E modules, 16 HSDS3[1] modules or 8 of each type. Each port module on this PSC consumes 1 SDF. There are 448 DS1 equivalents on this PSC.

Table 3-2 : High Speed Port Modules

Name	Interface	Description
HSPM	1 STS-1E	High Speed Port Module - This card provides an interface to 28 DS1s.
HSDS3	1 DS3	High Speed DS3 - This card provides an interface to 28 DS1s. The data is not formatted in the same way as the HSPM.

Continued on next page

[1] Note that the DS3 interface on the HS100 PSC is different from the DS3 interface on the LD100 PSC.

3. The Port Shelf Complex (cont.)

Low Speed PSC
LS100

With SDF PSC numbering, low speed PSC type (LS100) is a single shelf that supports 8 SDFs. It provides an interface to 28 octal DS1 cards. Every 7 port cards split 2 SDFs. There are a total of 224 DS1 equivalents on this PSC.

With sequential PSC numbering low speed PSC type (LS100) is two identical shelves, each supporting 8 SDFs. They are numbered, PSC *X*, HIGH and PSC *X*, LOW. It provides an interface to 56 octal DS1 cards. Every 7 port cards split 2 SDFs. There are a total of 448 DS1 equivalents on this PSC.

Table 3-3 Low Speed Port Modules

Name	Interface	Description
LSPM	8 DS1	Low Speed Port Module - This card provides an interface to 8 DS1s.

4. TITAN 532L Provisioning

Introduction

One of the major purposes of the 532L is to be able to take traffic coming from one location and statically switch it to another location. This static switch is called a cross connect. The process of adding, removing or changing cross connects on the system is called provisioning. The 532L command to create a cross connect is called a TCON. The command to take down a cross connect is called a TDIS.

TCON

A cross connect is an electronic pathway that allows data to flow from one DS0 to another. Each one has a source and a destination. A cross connect can be made at the DS1 level or the DS0 level. They may also be made as a range of contiguous DS0s on a single DS1 (the range could be DS0 (01 - 24).

There are two types of cross connects (TCONs) that are supported by the CCG. There is the standard TCON and the broadcast conference. The standard TCON is a two way cross connect meaning that data flows in both directions. The conference cross connect has one host and can have one or more listeners. Data cannot travel from a listener to a host.

TDIS

As with everything in life, all good things must come to and end. Every cross connect whether its a standard TCON or conference, can be removed from service. This is known as a TDIS. Once a TCON is disconnected, the DS0s that used to be connected are once again available.

5. TITAN 532L Conclusions

The TITAN 532L is an extremely complicated piece of machinery and it takes a significant amount of time (years) to thoroughly understand its operation. A detailed knowledge of the 532L is not necessary to complete this project. The important aspects of the 532L are discussed in this document. To help in understanding this project a trip to Tellabs may be made to see the 532L and discuss its operation.

6. The Cross Connect Generator Overview

Need for the CCG

The TITAN 532L can be configured to carry up to 7,168 DS1s (that's 172,032 individual phone calls). It is often necessary to cross connect most or all of these ports during testing. Fully configuring the 532L can take well over 10,000 commands. It is not feasible to perform this configuration by hand.

To help alleviate this burden, the 532L can accept a command (CMD) file which is a list of commands. Each one of these commands are executed by the 532L as if it were typed in by the user. While the CMD file is a big help, creating a file that contains thousands of commands is not a trivial task either.

One of the benefits of configuring a system during development is that a 532L can be configured using an algorithm. For instance, one may decide that NPC 1 must be cross connected to NPC 513, NPC 2 must be cross connected to NPC 514, NPC 3 to 515 and so on. It is trivial to write a program in **C** or **Perl** to generate this configuration. In fact several small programs exist to do this. There are several problems with these programs though:

- One program can create a CMD file to configure the 532L in one way. If the user wants a different configuration or the 532L is not set up the same way as the program expected it, the program must be changed.
- Programs that are used to generate CMD files cannot simply describe the system configuration that they will be generating.
- People who do not have a programming background may need to configure systems. They often ask software developers to create the CMD files for them.

CCG Components

The CCG is comprised of many components. There is the generator, configuration command set, API and GUI. The generator portion of the CCG is responsible for generating the commands that will be executed on the 532L to create the user's configuration.

The user will specify a system configuration using the various commands in the configuration command set. The commands include information on what PSCs are added in the system, as well as which port modules and NPCs to use and how to provision the system.

The 532L is an ever changing beast and the CCG will have to change with it. To help facilitate this an API is required. The functions in the API will define the NPC/port module/PSC/SDF relationships as well as describe how cross connects are to be made.

The GUI should be a standalone program. Its only job will be to help the user create the configuration file that will be used by the CCG.

7. CCG Algorithms

Introduction

In order for the user to generate thousands of cross connects efficiently, the configuration command set defines a series of algorithms that are used on ranges of port modules. For instance, if a 532L has PSCs 1 and 17 (we're using SDF numbering here) to be HS100. We may wish to use the TCON Offset algorithm. We would apply this algorithm to the every NPC on PSC 1.

The TCON Offset algorithm takes a range and an offset. Since our range starts at NPC 1 and ends at NPC 448 our algorithm would look like:

```
offset = 512
for each NPC from 1 to 448
    from_npc = <cur_npc>
    end_npc = <cur_npc> + offset
```

This algorithm would result in commands:

TCON::**FROMDC** 1, **TODC** 513, **TRSP**
TCON::**FROMDC** 2, **TODC** 514, **TRSP**
TCON::**FROMDC** 3, **TODC** 515, **TRSP**
...
TCON::**FROMDC** 448, **TODC** 960, **TRSP**

Cross Connect Types

An algorithm can be applied to two types of cross connects. There is the standard two way cross connect (TCON) and there is the daisy chain broadcast conference.

The TCON is a very easy, straight forward cross connect. There are two sides. A *from* side and a *to* side. Convention states that the *from* side is always less than or equal to the *to* side. While regular TCONs are used, they are not used very often because it is very difficult to monitor that sample data is correctly flowing through the system (The reasons for this will become apparent later on).

The daisy chain broadcast conference (BCST) is the more useful of the two cross connect types. The typical use is to have many conferences with a single host and a single listener. What happens is that NPC 1 is a host that is listened to by NPC 2 which is a host that is listened to by NPC 3 which is a host that is listened to by NPC 4, etc. The BCST is used more often because the daisy chaining of the conferences allow the user to monitor the data network from a single point (This too will become more apparent later on).

7. CCG Algorithms (cont.)

TCON Self

This algorithm accepts a range of port modules as input. On each module in the range a TCON is cross connected to itself.

For instance:

```
for (i = 1 to 3)
    tcon::fromdc 1, todc 2, trsp
    tcon::fromdc 2, todc 2, trsp
    tcon::fromdc 3, todc 3, trsp
```

TCON Offset

This algorithm uses a range of port modules and an offset as input. On each module in the range, the cross connect is made from that port module to that port module + the offset.

For instance:

```
offset = 512
for (i = 1 to 3)
    tcon::fromdc 1, todc 513, trsp
    tcon::fromdc 2, todc 514, trsp
    tcon::fromdc 3, todc 515, trsp
```

TCON Invert

This algorithm takes a starting point and an ending point to define a range. The cross connect go from the start to the end and work their way to the middle.

For instance:

```
for (i = 1 to 5)
    tcon::fromdc 1, todc 5, trsp
    tcon::fromdc 2, todc 4, trsp
    tcon::fromdc 3, todc 3, trsp
```

Continued on next page

7. CCG Algorithms (cont.)

BCST Chain

The daisy chain broadcast conference algorithm takes a series of port modules and uses each element in the range to define the next host in the conference. The last element in the daisy chain is looped back to the first element so that traffic can be monitored. The conference is a bit more complicated than the standard TCON because it requires two commands as well as a conference number. Notice how the last conference is connected to the first conference.

For example:

```
grth::conf 1, bcst, host 1
tcon::conf 1, to 516, trsp, trb
grth::conf 2, bcst, host 516
tcon::conf 2, to 520, trsp, trb
grth::conf 3, bcst, host 520
tcon::conf 3, to 536, trsp, trb
grth::conf 4, bcst, host 536
tcon::conf 4, to 540, trsp, trb
grth::conf 5, bcst, host 540
tcon::conf 5,to 1,trsp,trb
```

8. CCG Configuration Commands

Introduction

The CCG configuration file is a series of simple commands that the user will use to describe a specific 532L configuration. The commands names are in bold and the parameters (if any) are in italics. If a parameter is underlined then that is the value to be used if the command is not present in the configuration file. None of the commands are case or whitespace sensitive.

Some commands are affected by their scope (whether or not they are located within **{}**. There are three scopes. File, Algorithm and PSC scope.

PSCNumbering

File Scope

PSCNumbering: < *SDF* | *Sequential* >

This optional command specifies whether PSCs are numbered sequentially or by SDF number. The default value is SDF numbering. This command may only appear once in a configuration file.

PSCMap

File Scope

PSCMap: { <*PSC #*>, <*PSC Type*> **... }**

This optional command specifies all of the PSCs that are configured into the 532L (The configuration file does not necessarily have to use them all). There are no default values for this command. It can safely be left out.

If **PSCNumbering** is Sequential, then the valid values are:

- *PSC* [1 - 16], [*HS100* | *LS100 LOW* | *LS100 HIGH* | *LS100 LOW HIGH* | *LD100 LOW* | *LD100 HIGH*]

If **PSCNumbering** is Sequential, then the valid values are:

- *PSC* [1 - 256] , [HS100 | LS100 | LD100 LOW | LD100 HIGH]

For Example:

```
PSCMap:
{   PSC 1, LS100 LOW HIGH,
    PSC 2, LS100 LOW,PSC 2, LS100 HIGH,
    PSC 3, HS100,
    PSC 4, LD100 LOW
}
```

```
PSCMap:
{
    PSC 1, LS100, PSC 9, LS100, PSC 17, LS100,
    PSC 25, LS100, PSC 33, HS100, PSC 49, LD100 LOW
}
```

Continued on next page

8. CCG Configuration Commands (cont.)

Algorithm
File Scope

Algorithm < *algorithm* > **{** ... **}**

This command begins the Algorithm scope section. More than one Algorithm section may appear in a file but they may not be nested. The CCG will start with a few algorithms but should allow for new algorithms to be easily added.

The valid values for *algorithm* are (See section 7. for more details):

- TCONSelf, TCONOffset, TCONInvert, BCSTChain

PM
Algorithm Scope

[**PM | LSPM | HSPM**] < *start* [**-** *end*] **:** *npc1* [, *npc2* **|** - *npc2*]

The **PM** commands in the algorithm scope specifies an port module number or a range of port modules followed by a list or range of NPCs to include in the algorithm.

The port module range can be specified as a single port module or a range of port modules. The NPC range can be a single NPC, a range of NPCs or a comma delimited list on NPCs.

The following are allowed:

```
HSPM 1:1, 2-3, 4, 5, 10-20
LSPM 1-8: 1-4
PM 1-2048
```

Note that the PM option does not support a range of NPCs since there is only one NPC on the card.

Port module and NPC ranges are defined in the following table::

Port Module Name	Port Module Range	NPC Range per Port Module
HSPM	1 - 256	1 - 28
LSPM	1 - 896	1 -8
PM	1 - 2048	Not allowed

There is no limit to the number of these commands which may exist in a file or scope. If an NPC is used twice in a file then an error must be generated.

Continued on next page

8. CCG Configuration Commands (cont.)

PSC
Algorithm Scope

PSC < *PSC #, [PSC Type]* **> { ... }**

With all of the port modules, NPCs, PSCs and SDFs in the system and can be difficult to try and figure out some of the numbers. The **PSC** command will allow the user to specify port modules ranges relative to the PSC number.

For instance:

```
   PSC 2, HS100 { PM 1: 2,4,6,8,10,12 }
```

in PSC scope, Is Equivalent to:

```
   HSPM 17: 2,4,6,8,10,12
```

In algorithm scope.

The *PSC Type* parameter is optional if and only if the **PSCMap** command identifies the type of *PSC #*. In addition, the *PSC Type* command may not be LS100 LOW HIGH.

PM
PSC Scope

Inside the PSC scope, the **PM** command works slightly differently. The **PSC** command tells all commands within its scope to use relative numbering, as is illustrated in the **PSC** command section.

Port modules with relative numbering must obey the guidelines outlined in the following table:

Port Module Name	Port Module Range per PSC	NPC Range per Port Module
HSPM	1 - 16	1 - 28
LSPM	1 - 28	1 -8
PM	1 - 2048	Not allowed

Host
Algorithm/PSC Scope

HOST < ***PM*** >

The **Host** command takes a **PM** command as its input. The **PM** command may only reference a single NPC. This designates the NPC in the **PM** command as the host of broadcast conference. This command is only useful with BCST algorithms and should generate an error if used elsewhere.

9. External Requirements

Platform

The CCG will primarily be run on the UNIX platform, specifically Solaris 2.6 or later. It will also be run from the Win 98 and NT platforms so these must also be supported. It would be beneficial to also support Macintosh but this is not required.

Implementation Language

The only restriction on the implementation language is that it is already installed at Tellabs or it can be easily installed. This project requires a GUI, string manipulation and complex data types. The language should be chosen accordingly.