INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Automated Test System for ADSL Equipment

By

Tariq Syed

A thesis submitted to
the School of Graduate Studies and Research
in partial fulfillment of the
requirements for the degree of

Master of Applied Science (M.A.Sc.)

Ottawa-Carleton Institute
School of Information Technology And Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario

24/10/2000

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Automated Test System for ADSL Equipment

submitted by Tariq Syed
in partial fulfillment of the requirements for
the degree of Master of Applied Science (M.A.Sc.)

_____
Thesis Supervisor

_____
Department

Abstract

To achieve a competitive edge, reduced development cost, timely product delivery and product quality are the essential requirement for today's organization. Manual testing, requires skill workers which increases cost; increases testing time which delays product delivery; reduces number of test cases which affects product quality, is not a desirable choice. The low cost PC based automated system helps to get an edge by fulfilling the today's organization requirement.

The automated testing system described in this thesis is developed to support functional testing of all testing phases of Nortel's 1-Meg Modem System (SUT). The SUT is an inherently complex ADSL product and its testing is far more complex than just verification of process faults. The complexity of ADSL system, renders automated test system an important and imperative part of ADSL testing.

The requirement for an automated system has greatly increased due to the realization that manual testing is associated with additional resources and staffing constraints, which takes 15 to 90 minutes to run a single test on the SUT. Manual testing also requires additional time to perform re-execution for debugging and validation for fault correction. Whereas automation can do multiple re-execution of each test thereby increasing the amount of regression testing dramatically and improving the quality of feature testing on the SUT.

This thesis shows the indispensable requirement of automated test system for ADSL testing and its advantageous of developing a competitive edge for the organization.

Acknowledgement

I would like to take this opportunity to express my gratitude to Professor Sunil Das, whose guidance and suggestions helped me in putting this thesis together. My special thanks to my manager, Chris Davis, at Nortel who allowed me to schedule time-off to work on my thesis. My warmest gratitude to my parents whose prayers and support helped me in achieving my target and this degree. Finally, I would like to thank my family and my son Hamza for the time, which I was supposed to spend with him instead, was spent on this thesis.

Table Of Contents

CHAPTER 3 SYSTEM'S AUTOMATION

CHAPTER 4 HARDWARE TESTING

CHAPTER 5 SOFTWARE TESTING

List Of Figures

CHAPTER 1

1.1 INTRODUCTION

System diagnosis identifies faulty system components given assertions regarding intended or erroneous system behavior. The act of diagnosis is normally carried out in an iterative fashion as part of a system debugging / validation cycle. The problem solved in this dissertation is to verify the hardware and software development domains by diagnosing through the automated test system.

The automated test system diagnoses the problem by verifying the deviation of application's implementation from its specification. Automating the process of knowing where this deviation occurs facilitates the job of re-synchronizing the application and the specification. This helps ensure that the application is implemented according to specification and the document used is current.

Most software projects use a development process that incorporates: requirements analysis, design, specification, implementation, and testing. The requirements analysis captures the testing requirements. During design, models are built which describe how the application will be realized. At the specification, "black box" or functional test cases are derived. Implementation involves the actual coding. Testing verifies that the application was implemented correctly.

In most of the projects, implementation deviates from specification. The deviation may be intentional or accidental. Whether it is developer, integration, or regression testing, it presents an opportunity to validate the running application against the

specification. Integration testing, in some cases, is the first time that many of the components interact with each other. It is usually difficult for developers to do interoperability testing of their components earlier in the application development. This is because it requires other components that may not have been developed yet.

The designer/developer continuously improve and enhance the application by iterating implementation. An iterative development environment benefits from automating the process by comparing the specification to the implementation. Reducing the amount of work on the developer's part provides him/her with more incentive to keep cutting the specification / implementation iterative cycle.

Today, most projects only perform what is called black box testing. This is where the application / system is treated as a black box. Testers input information into the application / system and compare the output against expected results [Rop94]. This testing verifies that the application conforms to the specifications.

Developers usually do not have tools to assist them in validating design compliance. Most projects rely on code reviews to do the job. These manual reviews verify that the source code is complying with static definitions. Code reviews are not appropriate to predict the behavior of a running application. These code inspections tend to find the obvious bugs only. Subtle errors are not easy to spot by looking at the code, but show up at runtime. By using information in the specification, automation to validate the behavior can be done.

Asymmetric Digital Subscriber Line (ADSL) system's testing is far more complex than just a verification of process faults. Interoperability, conformance and performance testing involves many parameters for evaluating operation of the transmitter, receiver,

affects of the ADSL link over POTS and vise versa. As well as, immunity to noise intrusion of different sorts requires a detailed automated solution. Performance testing of ADSL system also requires testing all the Internet application as ADSL's strongest application is Internet connectivity.

In this thesis we present the automated testing performed on the Nortel Networks ADSL product, 1-Meg Modem system. The automated test system is designed to be non-intrusive. It runs on the PC based Window platform. It provides a graphical user interface (GUI) to the user to test the various feature of SUT (1 Meg Modem System). The automated test system uses the serial interface and standard SNMP interface to communicate with the SUT (1 Meg Modem System). Unlike other automated system, it tests hardware, software, conformance, functionality, and the performance of the SUT.

1.2 MOTIVATION

The last decade has seen Internet as the fastest growth segment within the telecommunication market. Ways are being sought to end the familiar "World Wide Wait", market segments are demanding an increase in speed for quick access to today's multimedia-rich content on the web and corporate / campus local area networks (LANs).

Before the advent of Internet, most of the local telephone companies (Bell Canada, Pacific Bell, etc) had the network usage predictable, revenues were plentiful, and things pretty much revolved around the relatively routine tasks involved in supporting the typical three-to-six-minute telephone call made by the subscriber. Since the explosion of Internet and World Wide Web, life has become much more interesting for these local telephone companies. One of the biggest changed involved the interaction between the

local telephone companies and the Internet. As seen in many other fields, the Internet has completely changed the rules of the voice networking game.

On Monday, January 6, 1997, a PSTN central office (CO) at the East Bay area of San Francisco browned out (congested) as many users logged in to the Internet at the same time through their dial up modem. Alexander Graham Bell's telephone based exchange was not made for Internet where users held the telephone lines for more than half an hour and sometimes even for the whole day. This created a need to separate the Internet traffic from the existing voice traffic.

Many solutions are possible for the congestion caused by the limited bandwidth and the Internet data traffic on the existing infrastructure of copper local loops:

One possible solution is faster dial-up modem. This increase in speed would not solve the congestion problem as user may browse some more pages and keep the PSTN circuitry holding time high. Hold time, not the data rate, is the main culprit of the dial-up modem technology.

Cable TV companies have been playing around with "cable modem" for a while now. These devices essentially bypass the local loop altogether and use the CATV network to provide high-speed Internet connections. The disadvantage of using this technology is this that cable is not available every where and therefore it can not provide a complete alternative solution for accessing Internet.

In the wireless arena, Internet traffic can be separated by using the wireless cable TV network. However, this solution is not applicable, as local loop is still required for upstream Internet traffic.

Satellite service providers have also considered providing Internet and thus separating the traffic from voice call. Satellite cost is high and it is not possible for every common user to afford the price.

The technology that has generated the most interest in terms of solving the local loop issues once and for all is Asymmetric Digital Subscriber Line (ADSL). It has been around in the laboratory for about more than ten years. Many vendors were attracted towards the ADSL solution and had generated an intense local competition. Since this technology encompasses the data and voice therefore it forces the telephone exchange manufacturing companies, Lucent, Nortel, Alcatel, etc, to leverage their expertise in the data site and integrate the ADSL technology in their telephone exchange product line. Similarly data equipment manufacturing companies like Cisco, NewBridge, etc, already have expertise in data and therefore consider ADSL a revenue generating technology. In this competitive race, the obvious winner will be the one who delivers the tested ADSL product first to the market. Nortel was one of the competitors. For delivering the product earlier by verifying the functionality of the product quickly, it was important to have proper testing tools. The ADSL product was not commercialized at that time so not many tools were available for its testing. Though all sort of tools were available for testing but they either tested data, voice or loops. ADSL testing requires concurrent use of data, voice and loop test equipment but in a specific sequence and at a particular time. By doing automation, it was possible to make an interaction between the individual test equipment in a way that tested the functionality of ADSL- based equipment quickly. Other obvious advantage of automated testing is that it speeds up the testing process. Both these advantages motivated me to select automation of ADSL testing for my

research work. The automated test system built for Nortel Networks ADSL equipment proved later that Nortel has taken the lead by launching its ADSL product earlier.

1.3 OUTLINE

The next chapter will cover the basic review of ADSL technology. This review is important because Nortel Networks 1-Meg Modem technology, which is being used as a SUT, is ADSL equipment. Being the proprietary product (SUT) of Nortel, it is not possible to discuss its functionality in detail. The automated test system does the functional testing of hardware and software of the SUT. Since most of the SUT features are standard ADSL features, therefore the general discussion of ADSL technology would help in understanding the functional testing of SUT.

Chapter 3 describes the architecture and design of the automated test system. The beginning portion discusses the objective of Nortel's automated system. Later, the methodology, architecture and individual components of an automated system are described. In the methodology, four of its phases (Operation Analysis, Operation Modeling, Automation Architecture and Automated System) are explained. Since Automation Architecture is a big topic so it is discussed separately. In the components of an automated system, hardware and software are discussed. In the hardware components, Personal Computer (PC) based hardware platform is discussed. In this section, the advantages of PC based platform over the other hardware based platforms are described. In the software components, the coding language – Visual Basic, Win32 API-Dynamic Link Libraries (DLLs), Windows Socket, Windows SNMP, SNMP Manager API, FTP Client API and ICMP Application (ICMP.DLL) are discussed. In the end Testing

Environment is described which explained briefly the connection and integration of the individual components.

Chapter 4 describes the hardware testing of the SUT carried out by the automated test system. It starts with an introduction, in which it describes the advantages of hardware testing for the complex systems like ADSL system. In introduction, the progress of hardware testing, i.e. from a small system to a complex multiple module system is also outlined. The next topic describes the Hardware / System functional testing and their advantageous over other testing technique. This is followed by the examples of various implementation of functional testing in different hardware areas. Usually, the examples are covered in the beginning chapters of thesis. Since this thesis covers both Hardware and Software testing techniques and in order to maintain a relationship between the previous research work and my work, the examples of previous hardware testing techniques are covered in this chapter. In this section, the recent industrial automated system and Nortel automated test system in the perception of functional testing are also discussed. The section on Nortel automated test system described the two suits of hardware test cases. In order to understand the suits of functional test cases, it was important to comprehend the functionality and architecture of the SUT. Therefore, in the next section the general hardware architecture and the functions of the SUT are explained. This section also describes a very important element of the functional testing i.e., by testing the function of a particular hardware component implicitly, other hardware components are tested explicitly. The last section in this chapter deals with implementation. It first covers the different components of an automated test system and then their implementation. Actual coding is given as an example to show the interaction

of these components with the SUT. This is followed by the two test cases. The first test case, Stuck Bit Test, shows the different bit patterns to check any stuck bit in the SUT. The patterns used are – all zeros, all ones, alternate one and zero and vice versa, marching zeros and ones and galloping ones and zeros. The graphical user interface and operation flow are also discussed in the end of first test case section. The second test case is the data path test of the SUT. Packet counters located across the data path confirms the sanity of the data path by confirming the establishment of data path and increasing the counter. The GUI and operation flows of the test case 2 are also discussed at the end of this section.

In line with chapter 4 (Hardware Testing), chapter 5 also first discusses the specification / functional based testing in the background of software testing. In the next section, 'Related Work', the other automated software testing tools are covered. The reason of keeping this topic here is the same as described earlier. Next, the general software architecture of the SUT is summarized. This section is important as in order to understand the specification based test cases described later, the function of the software modules of the SUT should be known. It also briefly describes the SUT software modules. In the implementation section, first the components of an automated test system that are responsible for the software testing, along with their actual coding, are discussed. The software components of an automated test system covered are – Manager SNMP-API, FTP Client API, ICMP (Ping) Application, and MSComm ActiveX control for accessing the serial port. Next section describes the various available formats for presenting the test result. The automated test system provides test result in DOS, Excel and GUI format. It discusses the system providing results in three different formats and

also explains the conditions of selecting one format over the other. Unlike the GUI of Hardware Testing, the GUI of Software testing is discussed in a separate section. The reason for discussing it separately is due to its rich set of options. Single GUI is used for all test cases. The single GUI covers the SNMP Agent and FTP Server IP address; Start, Exit, Load Configuration and Save Configuration buttons; selection of PC COM ports, slot#, line loops, downstream and upstream target speed and bandwidth and test setup time on each loop. Each elements of GUI is described in detail with their actual coding. The last section covers the seven Test Cases.

Chapter 6 concludes the thesis with future extension to our system and further research issues.

CHAPTER 2
xDSL TECHNOLOGY AND APPLICATION

2.1 INTRODUCTION

After years of steady but slow evolution in the voice-dominated telecommunications industry, the Internet phenomenon forced a dramatic change of pace. The rapidly rising proportion of data in the overall traffic mix on public networks has required new thinking and urgent action from network planner.

Capacity in core switching and transmission facilities has grown in leaps and bounds with the deployment of technologies such as dense wavelength division multiplexing. But until recently, the access network has been the biggest single obstacle to the progress of the Internet revolution.

The "content-rich" applications (e.g., graphics, multimedia, etc.) require much more bandwidth at the residential and branch-office level than in the past. High-capacity requirements are now typical of knowledge-based telecommuters, branch offices, schools, medical, clinics, etc. Users of Internet want fast, and always-on access. The local telephone network, with its bandwidth ceiling of 64 kbps per line and dial-up connectivity model, does not meet the requirement. However, the new generation of X-Digital Subscriber Line (xDSL) technology being rolled out by service provider all over the world offers an excellent solution. Here, X stands for variety of DSL technologies like HDSL, ADSL, VDSL etc and these technologies provide different performance levels.

xDSL makes use of the existing copper pairs in the telephone access network to deliver far more bandwidth than a regular phone line. Furthermore, xDSL provide direct data connectivity to the Internet, bypassing the local telephone switch and offering always-on access to Internet users.

An xDSL circuit connects an xDSL modem on each end of a twisted-pair telephone line, creating three information channels -- a high speed downstream channel, a medium speed duplex channel, depending on the implementation of the xDSL architecture, and a POTS (Plain Old Telephone Service) or an ISDN channel [ADS99-1]. The POTS/ISDN channel is split off from the digital modem by filters, thus guaranteeing uninterrupted POTS/ISDN, even if xDSL fails. Each channel can be submultiplexed to form multiple, lower rate channels, depending on the system.

In general the xDSL technologies provide a few megabits per second to 50 Mbps in one or both direction [Min98]. Data rates depend on a number of factors, including the length of copper line, its wire gauge, presence of bridged taps, and cross-coupled interference. Line attenuation increases with line length and frequency, and decreases as diameter increases. While the measure varies from telco to telco, these capabilities can cover up to 95% of a loop plant depending on the desired data rate. Customers beyond these distances can be reached with fiber-based digital loop carrier systems.

To upgrade the existing infrastructure to fiber is still many years away, and the cost is quite high, xDSL technology allows megabits-per-second rates on copper loops at a reasonable cost. xDSL refers to a set of similar technologies that provide high bandwidth over copper twisted pair without the use of amplifiers or repeaters on the loop (T1 systems, for example, require a repeater every one mile.). Use entails adding electronics

at both ends of the loop to provide for robust signal encoding; those encoding schemes, which are more sophisticated than traditional bipolar encoding used on T1 lines, support increased bandwidth.

xDSL signals are designed to maximize the rate of transmission of digital signals through subcategories of nonloaded twisted pairs, making full use of bandwidths on the wire. *Such bandwidth can be greater than 1 MHz; this is much more than 3,000 Hz normally used for voice transmission* [Min98]. It is then clear why xDSL signals will not pass through the loaded loops and may or may not be able to be incorporated into digital loop carrier and fiber-to-the-curb systems; loading reduces the available bandwidth – see Figure 2.1.

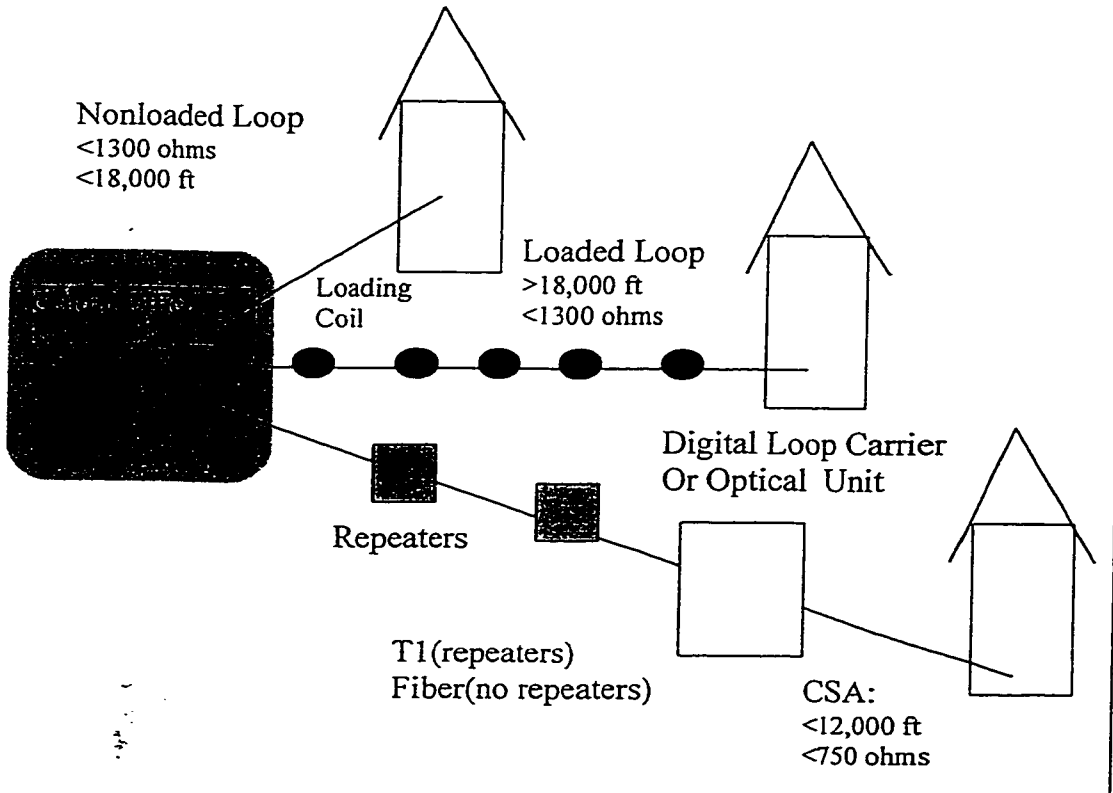

Figure 2.1: Various Loop Arrangements

The enterprise branches' networks are interconnected through corporate extended LAN. Observers estimate that 80 percent of the traffic on such LAN infrastructure is either directed to or coming from remote locations. The interconnection of these remote LANs over a set of traditional meshed T1s, frame-relay links, or ATM facilities may be too expensive. Therefore, xDSL technology can be used to interconnect such LAN at a lower cost.

2.2 XDSL LOOP

The various cable configurations between the telephone companies (CO) and the customer premises are known as "Local Lops". There are different types of local loops available today for xDSL transmission. Some loops are straight loops with minimum length and few bridge taps. These loops are considered to be simple loops. *CSA # 0 and ANSI #15 loops fall under the category of simple loop* [Con99]. Some loops are complex loops. These loops comprise of mixture of wire gauges, multiple bridge taps and extended lengths. *ANSI #3 & ANSI #6 are the complex loop examples* [Con99]. There are more than 29 types of loops for xDSL transmission. It is beyond the scope of this thesis to discuss each loop individually. However, these loops are summarized in the appendix-A. In this section we will deal with evolution of the loops, and the impairment attached with these loops.

2.2.1 EVOLUTION

Last century saw the introduction of underground telephone cable which used twisted pairs #18 American Wire Gauge (AWG). *The capacitance per unit length of the pairs was chosen to be 0.083 microF/mile, which is still used to the present day in the loop network* [ADS99-2]. Wire gauges of 19, 22, 24, and 26 AWG have come to be used over the years in the loop plants. At the turn of the century, in the absence of electronic mechanisms to aid transmission, three techniques were used:

- Improvement in the electro-acoustical efficiency of telephone set
- Use of the coarser gauge.
- Band limits on voice signal via the use of periodically spaced inductors, called loading coils, that add to the native inductance of the wire pair to reduce attenuation at frequencies below about 3,300 Hz.

The bandwidth restriction was judged to be satisfactory for voice transmission. Subjective tests determined intelligibility of telephone speech to be adequate. In the 1950s, AT&T issued engineering rule to systemize the design of wire loops serving about 80 percent of the customers in the United States. *These rules placed an upper bound of 1,300 ohms of resistance on maximum-length loops. Wire gauges were required to be no finer than 26 AWG, and loops longer than 18,000 feet were required to be "loaded" with 88 MHz inductors placed at 6,000-foot interval starting 3,000 feet from the Central Office (CO). Unterminated branch pairs or bridges taps were constrained to be less than 6,000 feet in total length when added together* [Haw97].

When Pulse Code Modulation (PCM) was introduced in the early 1960s, voice signals were assumed to be band-limited to 4,000 Hz, with 8 bits encoding of 8,000 samples per second, result in 64,000 bps digitized voice signals. *In the early1970s, T1 carrier lines were introduced into the local loop* [Min98]. The T1 link was terminated by digital loop carrier system terminals: one in the CO (soon to be incorporated into the digital switch) and the other, a remote terminal, installed within 12,000 feet of nonloaded twisted pair length to end users. The 12,000-foot design rule beyond the remote terminal is called Carrier Serving Area (CSA) design. In the early 1980s, optical fiber began to be installed between digital loop carrier terminal in lieu of T1 lines, but with no change in the CSA design rules.

As a result of this development of loops in the U.S. telephone network, there are now three types of connection between end users and the telephone central office. The first types are nonloaded twisted pairs up to 18,000 feet and 70 percent of the loops come under this category. The second type of loops are loaded loops which are greater than 18,000 feet in length. 15 percent of the loops fall in this category. The third type of the loops are the derived loops with up to 12,000 feet of nonloaded twisted pair connected to a remote terminal of a digital loop carrier system. The remaining 15 percent of loops belong to this category.

2.2.2 COPPER LOOP IMPAIRMENTS

The Impairments that high-speed digital services must deal with on the copper loop are well understood. *All xDSL schemes increases the normal, modest analog voice band, to 400 kHz or so and as far as 1.1 MHz* [Gor99]. The impairments at these frequencies fall into two main categories: physical and electrical.

(i) Physical Impairment:

These impairments arise from the local lop being engineered and optimized for analog voice. As shown in the Figure 2.2, the physical impairments are:

Load Coils:

Service providers added load coils to extend the reach of voice passband but at the same time it choke off higher frequencies. In many cases mixed gauges were used instead of loaded coils for the same purpose.

Bridged Taps:

These are unterminated branches at the lateral side of the cable. The advantage is this that the customer can be added and deleted easily. However, it reflects signals and causes interference at the higher xDSL frequencies.

Loop Splicing:

Splices between different wires bend and flex in the wind, causing "micro outage" of short duration. And splices are not always done properly among pairs within the same bundle. Although each pair is color-coded, various factors make it possible to splice wires from separate pair together. Such split pairs are seldom fatal for voice, but a real problem for digital services.

Untwisted Drop Wires:

Short but untwisted drop cables to the premises had minimal effect on overall voice quality, but are a concern in xDSL environment.

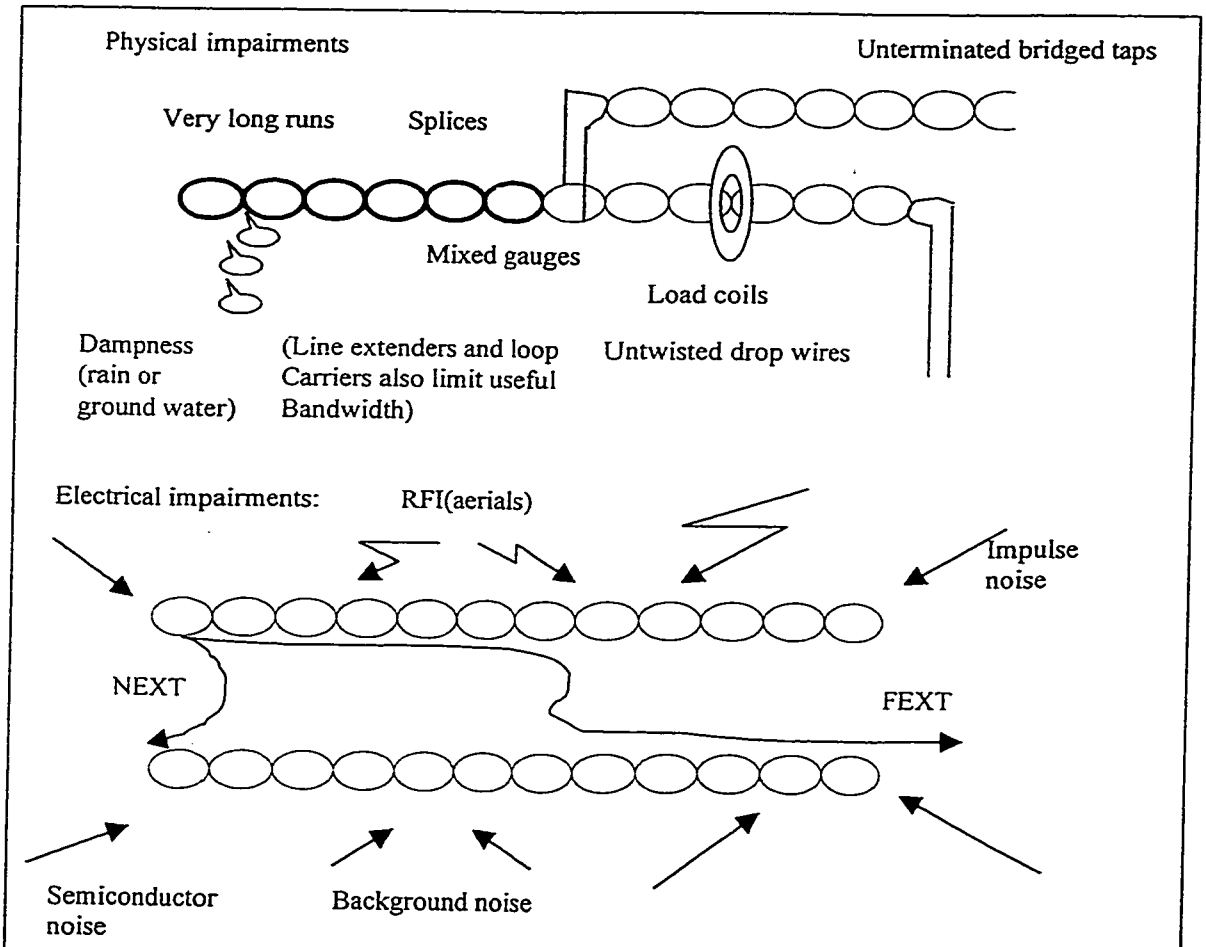

Figure 2.2: Loop impairments

Dampness:

It is always a problem, whether it is caused by rain which affect the aerial cable or it is caused by ground water which effect the cable buried in it.

Line Extenders & Loop Carriers:

The use of proprietary line extender and digital loop carrier (DLC) pose additional problems for xDSL mainly due to their operation only in the voice passband.

(ii) Electrical Impairment:

Electrical loop impairments are also called interferers or disturbers and these are shown in Figure 2.2:

Radio Frequency Interference (RFI):

At xDSL frequency, RFI is a major concern. Much Amplitude Modulation (AM) radio station broadcast in the same range as xDSL operates. Aerial wires are much more at risk, especially where AM stations are dense. Even amateur radio can be a concern for xDSL schemes that operate on aerial cable above 1 MHz or so.

Crosstalk:

The unwanted electrical coupling of transmitter and receivers is called crosstalk. *There is both near end crosstalk (NEXT) from a receiver pair's adjacent pair transmitter and far end crosstalk (FEXT) from a remote pair's transmitter* [Lin88]. There is also self-crosstalk (SNEXT) from the same types of xDSL running in the same cable binder and foreign crosstalk (FNEXT) from other adjacent types of xDSL.

Background Noise:

Background noise could be thermal noise, semiconductor noise and impulse noise. These noises can be generated from local exchange equipment and premises equipment such as light dimmers and refrigerator compressors. Proposals have been made to place surge protectors such as power strips on common household appliances to minimize these noise effects.

2.3 THE xDSL FAMILY

There are some ten xDSL technologies that can be deployed for high speed local access. ADSL, RADSL, VDSL, VADSL, HDSL, DSL, SDSL, BDSL – enough for several dizzy spells [ADS99-2]. Most of these acronyms have relatively clear definitions, but they often suffer confusion, with one another and with other acronyms.

The basic acronyms for all DSL arrangements came from Bellcore, and the basic confusion is between a line and its modems. In general we say that DSL signifies a modem, or a modem pair, and not a line at all. A modem pair applied to a line creates a digital subscriber line, but when a telephone company buys DSL, or ADSL, or HDSL, it buys modems, quite apart from the lines, which they already own. So, DSL is a modem, not a line. This confusion becomes quite important to avoid when we talk about prices. A "DSL" is one modem; a line requires two.

As shown in Figure 2.3, *the xDSL technologies can be divided into three major categories, symmetric methods (HDSL), very high rate DSL (VDSL) and asymmetric methods (ADSL)* [Gor99]. The first two technologies will be discussed here briefly

whereas the last one –ADSL will be discussed in detail. Since the SUT belong to ADSL technology and being proprietary product of Nortel, its functionality can not be discussed due to the security reason. However, as it belongs to ADSL technology, most of SUT functionality is common to ADSL. Therefore major portion of SUT functionality is covered automatically under ADSL discussion. Later, this knowledge of ADSL functionality will also be required during the discussions on hardware / software testing.

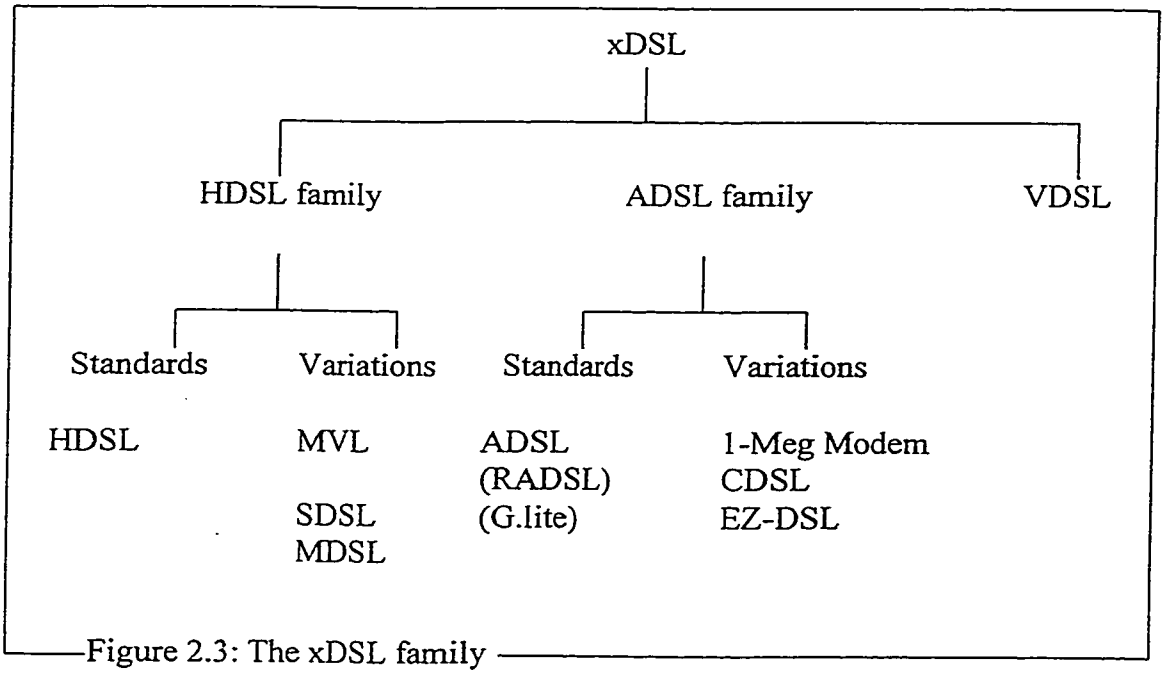

Figure 2.3: The xDSL family

2.3.1 SYMMETRIC METHOD

Symmetric method technologies are following:

2.3.1A HDSL -- High data rate Digital Subscriber Line:

HDSL is simply a better way of transmitting T1 or E1 over twisted pair copper lines. It uses less bandwidth and requires no repeaters. *Using more advanced modulation techniques, HDSL transmits 1.544 Mbps or 2.048 Mbps in bandwidths ranging from 80 kHz to 240 kHz, depending upon the specific technique, rather than the greedy 1.5 MHz absorbed by AMI* [ADS99-2]. HDSL provides such rates over lines up to 12,000 feet in length (24 ga), the so-called Carrier Serving Area (CSA), but does so by using two lines for T1 and three lines for E1, each operating at half or third speed.

Most HDSL will go into the feeder plant, which connect subscribers after a fashion, but hardly in the sense of an individual using a phone service.

Typical applications include PBX network connections, cellular antenna stations, digital loop carrier systems, interexchange POPs, Internet servers, and private data networks. HDSL2 is the "next generation" of HDSL. All of the limitations of current HDSL products are intended to be addressed in HDSL2.

As HDSL is the most mature of DSL technologies with rates above a megabit, it will be used for early-adopter premises applications for Internet and remote LAN access, but will likely give way to ADSL and SDSL in the near future.

2.3.1B SDSL -- Single line Digital Subscriber Line:

On its face SDSL is simply a single line version of HDSL, transmitting T1 or E1 signals over a single twisted pair, and (in most cases) operating over POTS, so a single

line can support POTS and T1/E1 simultaneously. However, SDSL has the important advantage compared to HDSL that it suits the market for individual subscriber premises which are often equipped with only a single telephone line. SDSL will be desired for any application needing symmetric access (such as servers and power remote LAN users), and it therefore complements ADSL. It should be noted, however, that *SDSL will not reach much beyond 10,000 feet, a distance overwhich ADSL achieves rates above 6 Mbps* [Sum99].

2.3.1C Other HDSL Family

Multiple Virtual Line **(MVL)** is a variation of HDSL scheme. *MVL is a Paradyne solution that scales from 128 to 768 kbps in N x 64 increments up to 24,000 ft (7.32 km)* [Gor99]. It, in contrast to HDSL and HDSL2, will allow analog plain old telephone services (POTS) "lifeline" support without a special premises splitter to isolate the analog voice passband, along with up to eight digital MVL devices. "Lifeline" POTS is a minimal telephony quality which acknowledges that at times POTS will interfere with digital services (especially during on-hook/off-hook cycles) and vice versa.
Moderate bit rate **DSL (MDSL)** is an another variation of HDSL scheme. *It usually runs at 768 kbps on one pair up to 21,000 ft (6.41 km)* [Gor99].

Finally, integrated services digital network (ISDN) **DSL (IDSL)** is also a part of HDSL family. It provides ISDN basic rate interface (BRI) 2B+D "channels" for 160 kbps service over up to 18,000 ft (5.49 km), but not to an ISDN switch port. *Loop resistance must be less than 1300 ohms, and the total IDSL loop loss must be less than –39 dB* [Gor99].

2.3.2 VDSL -- Very high data rate Digital Subscriber Line:

VDSL began life being called VADSL, because at least in its first manifestations, VDSL will be asymmetric transceivers at data rates higher than ADSL but over shorter lines. *While no general standards exist yet for VDSL, discussion has centered around the following downstream speeds* [ADS99-2]:

12.96 Mbps	(1/4 STS-1)	4,500 feet of wire
25.82 Mbps	(1/2 STS-1)	3,000 feet of wire
51.84 Mbps	(STS-1)	1,000 feet of wire

Upstream rates fall within a suggested range from 1.6 Mbps to 2.3 Mbps. The principal reason T1E1.4 decided against "VADSL" was the implication that VDSL would never be symmetric, when some providers and suppliers hope for fully symmetric VDSL someday, recognizing that line length will be compromised.

In many ways VDSL is simpler than ADSL. Shorter lines impose far fewer transmission constraints, so the basic transceiver technology is much less complex, even though it is ten times faster. VDSL only targets ATM network architectures, obviating channelization and packet handling requirements imposed on ADSL. And VDSL admits passive network terminations, enabling more than one VDSL modem to be connected to the same line at a customer premises, in much the same way as extension phones connect to home wiring for POTS.

However, the picture clouds under closer inspection. VDSL must still provide error correction, the most demanding of the non-transceiver functions asked of ADSL. As public switched network ATM has not begun deployment yet, and will take decades to

become ubiquitous, VDSL will likely be asked to transmit conventional circuit and packet switched traffic. (Indeed, a recent telephone company RFQ describes a VDSL type transceiver with three circuit-switched video channels and a single ATM channel.) And passive network terminations have a host of problems, some technical, some regulatory, that will surely lead to a version of VDSL that looks identical to ADSL (with inherent active termination) except its capability for higher data rates.

VDSL will operate over POTS and ISDN, with both separated from VDSL signals by passive filtering.

2.3.3 ASYMMETRIC METHOD

Asymmetric method technology is a new modem technology, converts existing twisted-pair telephones into access paths for multimedia and high speed data communications [ADS99-1]. It supports transmission of "high speed" data communications traffic, in conjunction with telephony Plain Old telephone Service (POTS), over a single pair of standard-gauge copper wire for distances around 18,000 feet.

The vendor specific variations of Asymmetric method technologies are Nortel's **1-Meg Modem**, Rockwell's Consumer CSL (**CDSL**), Cisco's **EZ-DSL**. The G.lite (G.992.2) is a standard ADSL technology. It will establish a "least common denominator" service sometimes called universal ADSL (**UADSL**). It is intended to use as a "splitterless" but with POTS services. Debate over G.lite specifics continues, especially over whether a remote splitter is allowed and just where it should be. *G.lite*

will use a subset of the 255 DMT bins and is intended to interoperate with "full" ADSL modems at either end in the future [Gor99]. Asymmetric technologies are:

2.3.3A Asymmetric Digital Subscriber Line - ADSL

In spite of the confusion over the relationships between DSLs, xDSLs, and ADSL, one thing is clear that ADSL is the most standardized of all, in terms of available documentation, service trials, and open specifications. *It is widely anticipated that many xDSL services will begin with ADSL and may even end with ADSL* [Gor98]. *ADSL work started in the mid-1980s and a history that is already 12 years old* [Gor99]. It followed on the heels of HDSL, but is really intended for the last leg into a customer's premises.

As its name implies, ADSL is asymmetrical. The downstream speed is much greater, sometimes ten times greater, than the upstream speed. A maximum speed downstream of 8.192 Mbps is defined for ADSL. However, this speed may be quite difficult to achieve in practice, not due to ADSL limitations, but due more the limitations in throughput given the current Internet architecture and backbones. So for the foreseeable future, most equipment vendors and service providers would settle for 4 Mbps to 6 Mbps as a maximum. *Following are the ADSL downstream speed ranges at different loop length* [ADS99-2].

Up to 18,000 feet 1.544 Mbps (T1)

16,000 feet 2.048 Mbps (E1)

12,000 feet 6.312 Mbps (DS2)

9,000 feet 8.448 Mbps

Upstream speeds range from 16 kbps to 640 kbps. All of these arrangement operate in a frequency band above POTS, leaving POTS service independent and undisturbed, even if a premise ADSL modem fails.

The ADSL technology is generally destined for residential users, small office/home office (SOHO), and small business users. The majority of target applications for digital subscriber services are asymmetric. Video on demand, home shopping, Internet access, remote LAN access, multimedia access, specialized PC services all feature high data rate demands downstream, to the subscriber, but relatively low data rates demands upstream. The IP protocols for Internet or LAN access push upstream rates higher, but a ten to one ratio of down to upstream does not compromise performance in most cases. MPEG movies with simulated VCR controls, for example, require 1.5 or 3.0 Mbps downstream, but can work just fine with no more than 64 kbps (or 16 kbps) upstream. As ADSL transmits digitally compressed video, among other things, it includes error correction capabilities intended to reduce the effect of impulse noise on video signals. Error correction introduces about 20 msec of delay, which is much too much for LAN and IP-based data communications applications. Therefore ADSL must know what kind of signals it is passing, to know whether to apply error control or not.

2.3.3B Rate Adaptive DSL -- RADSL

Rate Adaptive DSL is another DSL technology that automatically rate adapts according to the condition of line or can be manually adjusted to different ADSL speeds [Gor98]. The whole concept is similar to the idea of "self-equalizing", which balances attenuation over the whole frequency spectrum and accordingly increases or decreases speed.

All of the other properties of RADSL essentially mimic ADSL in terms of maximum speeds and distances. RADSL is a natural progression of ADSL, and all ADSL equipment should evolve into RADSL in the future. For equipment based on Discrete Multitone (DMT), RADSL is an inherent capability that appears as a result of the way the technique functions. Although RADSL operation is not impossible in CAP-based ADSL, such operation is difficult to achieve and adds a lot of circuitry and procedural overhead to the CAP devices. The biggest difference is that with CAP, the signal spectrum changes. Nevertheless, several CAP-based ADSL vendors have introduced RADSL equipment.

2.3.3C Consumer DSL -- CDSL

Early trials with ADSL and RADSL uncovered a rather serious situation, regarding the customer's premises. ADSL / RADSL required the installation and maintenance of a remote device—the splitter. The premises splitter's main function was to allow the simultaneous use of existing analog telephony and faxing devices with the data access (internet) devices in the home or SOHO location. However, besides introducing complexity, the presence of the splitter also raised issues about the premises wiring and configuration. There was a real concern about the continued care and feeding of the remote splitter device.

Near the end of 1997, Rockwell Semiconductor Systems introduced CDSL, a variation of ADSL, to address these concerns and limitations. CDSL allows simultaneous voice telephone calls and Internet and Web access over the same local loop without any splitter [Gor98]. Rockwell also proposed its method for standardization before the ITU, as something called "G.adsl lite," which shows the close relationship between

ADSL/RADSL and CDSL. In fact, the only significant difference between ADSL/RADSL and CDSL besides the absence of the premises splitter and wiring concerns is a restricted operating speed range (most importantly, 1-Mbps downstream as opposed to about 8 Mbps with ADSL). By the end of 1997, Nortel Networks, Microsoft, Compaq, and Intel had all made CDSL support announcements.

CDSL is still ADSL and is dependent on line conditions for maximum speeds, but the more modest maximum should allow the maximum to be reached in more varied circumstances than ADSL or RADSL. More importantly, CDSL is not intended as a *replacement* for ADSL or RADSL at all. CDSL's intention is to *complement* service providers' ADSL and RADSL deployments into places where higher speeds are not particularly mandated or feasible, and where there are concerns about remote splitter and wiring installation. ADSL and RADSL are still expected to be popular services in and of themselves.

NORTEL NETWORK'S 1-MEG MODEM

The automated test system, which is proposed in this thesis uses the Nortel's 1-Meg Modem system as its System Under Test (SUT). The ADSL automated system which is being presented in this thesis has been developed for 1-Meg Modem. *Nortel Networks 1-Meg Modem service is a CDSL technology that promises an exciting new way to surf the Internet at blazing speeds. Supporting data rates of 1.280 Mbps downstream and 320 kbps upstream, 1-Meg Modem service delivers cost-effective bandwidth ideally suited for bandwidth-hungry Internet subscribers and teleworkers* [Nor99-2].

1-Meg Modem's splitter-less technology is designed to eliminate the need of truck roll, and changes to outside plant or homewiring. Its always available connection

excludes dialing in, busy signals, and slowdowns. As shown in the Figure 2.4, the easy-to-install digital modem with simultaneous analog voice services using any existing telephone jack. A passthrough RJ-11 jack on the unit permits a telephone, answering machine or fax to operate.

For service providers, 1-Meg Modem technology uses the existing DMS line peripheral infrastructure and access to the existing copper loop. The operational, logistical, and engineering impacts to the central office, outside plant, and subscriber's home are greatly minimized.

Congestion on the switch from long-duration data calls is relieved because data traffic is routed off the twisted pair at the line concentrating module (LCM) and directed, for example, to an Internet service provider (ISP).

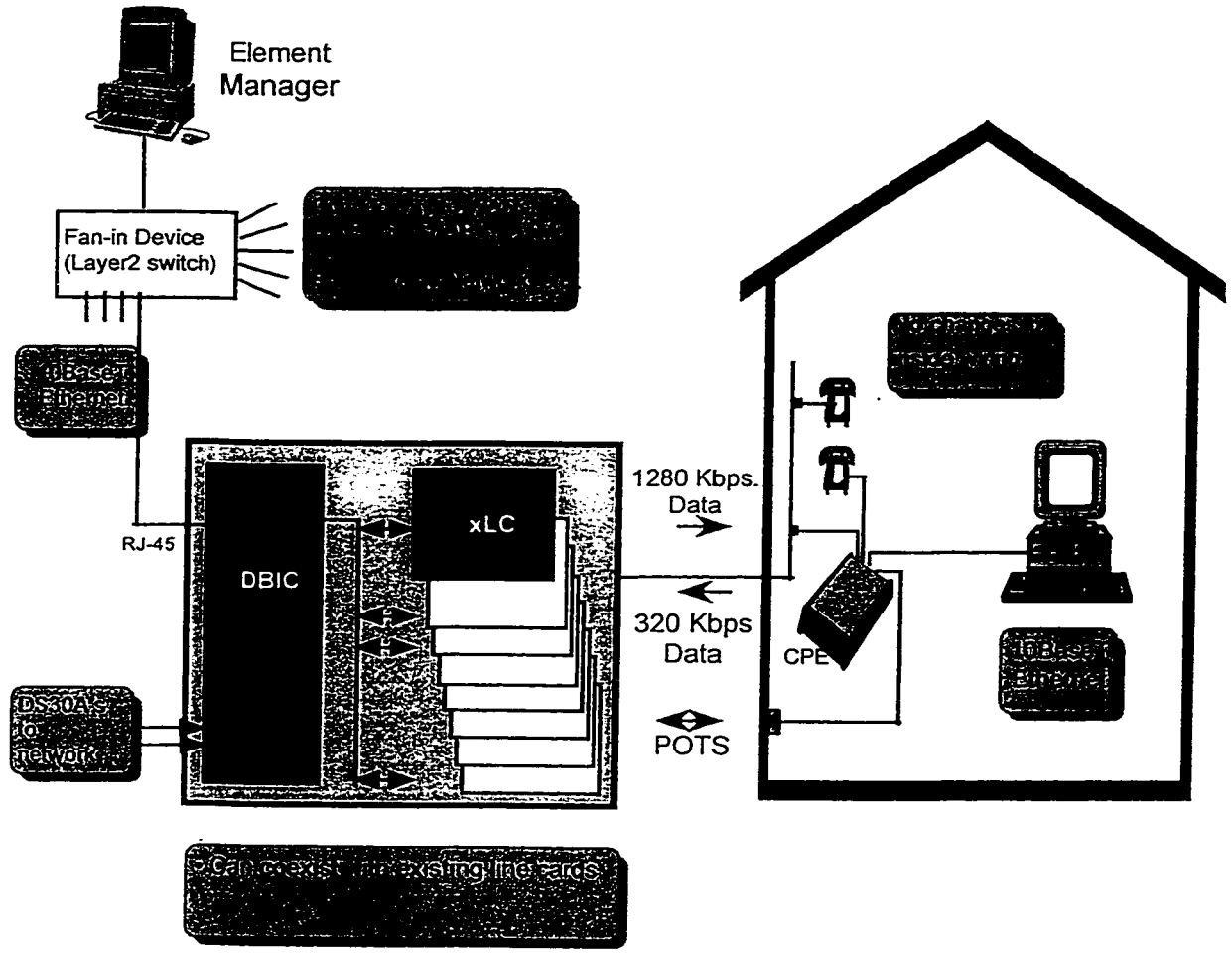

Figure 2.4: 1-Meg Modem Components

2.3.3.1 ADSL Technology:

ADSL depends upon advanced digital signal processing and creative algorithms to squeeze so much information through twisted-pair telephone lines. In addition, many advances have been required in transformers, analog filters, and A/D converters [ADS99-1]. Long telephone lines may attenuate signals at one megahertz by as much as 90 dB, forcing analog sections of ADSL modems to work very hard to realize large dynamic ranges, separate channels, and maintain low noise figures. On the outside, ADSL looks simple, -- transparent synchronous data pipes at various data rates over ordinary telephone lines. On the inside, where all the transistors work, there is a miracle of modern technology.

*ADSL products use carrierless amplitude / phase modulation (**CAP**), quadrature amplitude modulation (**QAM**), and discrete multitone (**DMT**) technology for its line coding* [TR-98]. Whatever line coding technique is used, whenever the same two wires in a pair are used for full-duplex operation, either the frequency range must be split into upstream and downstream bandwidths or echo cancellation must be used. In ADSL, both FDM and echo cancellation techniques can be combined and the frequency ranges may overlap without coinciding.

DMT coding use either FDM or echo cancellation to achieve full-duplex operation. FDM is a simpler method to implement. Echo cancellation is always vulnerable to the effect of near end crosstalk, where a receiver pickups up signals that are being transmitted on an adjacent system. The other system may be another pair of wires or may be the transmitter of the same system. Following are the details of the line coding which are being used in the ADSL system. However, before discussing the line coding it is

important to understand its two important component which are echo cancellation and FDM.

(i) Echo Canceling and FDM:

Some form of echo control is required whenever the same frequency range is used for sending signals in both direction at the same time on the same physical path. Echoes commonly arise due to impedance mismatches on the signal path as some of the signal is reflected back to the sender at these points. When the same frequency range is used in both directions, this signal reflection could easily be mistaken for a signal originating at the remote end of the circuit. Echo cancellers electronically subtract the signal sent from the signal received, allowing any signal actually sent from the remote end to be distinguished more easily.

One way to accomplish echo control is to separate the frequency range into upstream and downstream bandwidth by frequency division multiplexing or FDM. In FDM method no echo control is needed in the end devices.

The CAP-Based ADSL devices typically use the FDM approach whereas DMT based ADSL devices typically use the echo cancellation approach , although there are exceptions. This echo cancellation approach is called an echo –FDM "combination" due to the asymmetrical nature of the devices [Gor98]. Basically, there are "FDM ADSL" and "echo cancelled ADSL" systems and equipment.

(ii) CAP versus DMT:

CAP and DMT are the line codes that are most frequently used in ADSL products (ATU-R and ATU-C). Lines codes determine how the zeroes and ones of the digital signal are sent and received. One is carrierless amplitude/phase modulation (CAP) and the other

is discrete multitone (DMT).

No other controversy exists in ADSL that is at once so vital to the technology and yet so confusing to non-specialists. Sometimes the debate about the relative merits of each assumes almost religious fervor, but both work just fine. The debate revolves around other issues, such as performance, cost-effectiveness and signal processing delay. In other words, which is *better*.

CAP is closely related to QAM, and many treat the two as virtually indistinguishable. DMT is more complex than CAP/QAM and is the open standard chosen by ANSI for ADSL (T1.413-1995) [Sum99].

With regard to the CAP/DMT debate, most observers agree that both have strengths in different areas that affect ADSL performance and deployment scenarios. For example, it is generally agreed that DMT is better at rate adaptation, varying loop conditions, handling noise (digital "crosstalk"), and subcarriers (for voice and other purposes). CAP is seen to enable more simple echo cancellation (although many CAP products use FDM!), latency (about 25 percent of DMT processing delay is claimed), maturity (based on QAM, which has been around for years), and simplicity.

In spite of the presence of DMT in the T1.413 standard, an active and vocal group within ANSI is pushing for CAP as an acceptable "optional" line code for ADSL [Gor98]. In fact, there is little in the standard that is tied to DMT exclusively; both can be used in ADSL. Some have proposed adding CAP circuits to DMT devices, and vice versa. Even straight QAM has been proposed as an ADSL line code since this would eliminate the need for the CAP rotation function, making devices slightly more cost effective and simpler.

2.3.3.2 Proposed ADSL Test Requirement for Conformance and Performance Testing:

Conformance and performance testing of ADSL products are much complex than other product's testing.

(i) Conformance testing:

Conformance testing verifies electrical standards and allowable product limitations [Dao98]. It addresses two areas, Physical Layer Conformance and Protocol Conformance. Conformance and Performance are supported by the following parameters.

***Electrical Characteristics*:**

The Electrical requirements of the ADSL products are following:

a) DC Characteristics:

Testing and verification are require to check that the DC, Power feeding, Ring-Trip conditions, Ringing Voltage of the POTS splitter should not affect the performance and operation of the modem.

b) Voice-band Characteristics:

There should not be an interference of voice parameters into ADSL band and visa versa.

c) ADSL-band Characteristic:

ADSL band parameters like Return Loss, Total Power, Power Spectral Density, Total Harmonic Distortion, Timing Characteristic, Jitter, etc, should not affect voice band application.

(ii) Performance Testing:

Bit Error Rate is a parameter of the performance testing. ADSL provides mechanism to correct limited burst of errors, and to maintain a specific Bit Error Rate with a noise margin in the overall bandwidth of loop. ADSL can mask Bit errors at the physical layer. Therefore BER test alone is not sufficient for measuring the performance. Additional metrics like line attenuation, noise margin, line status, transmit blocks, corrected blocks, uncorrected blocks, counter for loss of signal, loss of frame & loss of power, interleave delay, and rate adaptation are also required to test the performance.

CHAPTER 3
SYSTEM'S AUTOMATION

3.1 INTRODUCTION

Automation is a process designed to follow automatically a predetermined sequence of operations or respond to encoded instructions. Increasing levels of automation may help address the needs of modern system and the organizations that manage them. Due to industrial competitiveness and budget constraints, organizations face increased pressure to reduce operations costs while increasing operating effectiveness. Though small when considered as a part of total life-cycle costs, operations costs are a substantial part of the annual operating budget for many systems. Thus, once a system has been developed and fielded, reducing expensive skilled worker via automation tools and functions may be an effective method to reduce operating costs.

Light-out automation is a term coined in the 1980's in the area of manufacturing operations [JAI86] to describe a *fully automated manufacturing facility in which robots, intelligent work cells, automated material handling systems, and computer-based controllers produced goods without human supervision or intervention* [SHA85, WAR85]. Other domains especially systems' testing are now using the 'light-out operations' in which human tester will not be in attendance during certain shifts and automated systems will execute regression tests, respond to anticipated anomalies, and report on the success or failure of tests.

Manual testing is expensive and unreliable due to human factor. Tester committed error even with proper documentation. Automated testing removes human factor and thus

creates a reproducible measurement technique independent of operator skill or fatigue level. It also drastically reduces the testing time and increases the product quality.

3.2 ISSUES WITH AUTOMATION

The introduction of any new technology carries with it associated benefits and costs. Sometimes, *rather than transforming system control into a more manageable function as intended, automated system caused problems in both system performance and human interaction* [Bil97, Par96, Zsa97]. *Users impediment to introducing an automated system are* [Som92]:

- questions regarding the sanity of the automated test system
- questions regarding the semantic correctness of test scripts
- questions regarding the automated result processing

While all three questions are valid areas of concern, each can be alleviated in a fairly straight manner.

Confidence in the sanity of the automated test system itself is built both by demonstrated reliability, and by user awareness of the way in which the automated system is connected to the SUT (system under test). If the non-intrusion rule is truly observed, the worst the system might introduce an event across the natural SUT links or the test system might miss or ignore events from the SUT. In the former case, it is due to a non-previously planned test case and in the latter, it is due to a non-fully executed test case. In either case, a well designed automated test system will flag the test as requiring analysis or as a failure.

Confidence in test scripts can be gained by showing that they contain all elements a tester would observe during manual execution. When developing new tests, all those new tests should be first executed manually. The automated script should be altered to correct any deficiencies found during the manual execution.

Confidence in test results comes easily when the test system uses pessimistic test analysis techniques. The test philosophy in SUT testing is that no test be passed if there is any question of its correct execution. Erroneous test results must be avoided at all costs. As long as the user can quickly identify the known passed/failed tests, and have confidence that all the questionable results are separately classified, gaining trust is directly proportional to the user trust in their own test conversion.

One concern is the psychological cost of automated system. In an environment where fully manual testing has been the norm, introducing of automated testing is usually viewed as a threat to test engineers. Conversely the belief that the system should be capable of performing all levels of testing otherwise it is not useful, permeates. The problem with introducing automation is to balance the cost of developing system in a timely fashion with the needs and expectations of the end user community. The system should be designed and implemented in such a fashion that the end user has a significant amount of input into the feature content of the system, and the order in which the features are provided. It is much easier to implement the system in phases such that incremental "stable" automation is provided, than to attempt to support all possible automation scenarios in one "shot". One must also attempt to avoid trying to automate testing of features which change with such rapidity that the automated test system is constantly out of date, or one step behind the requirements of the end user.

3.3 OBJECTIVE OF AUTOMATED TESTING

The design of Nortel Network Automated System considered the following objectives for effective testing.

Reliable and Cost-Effective Testing:

To achieve reliability and cost-effectiveness it is important to include:

- thoroughly tested tools that speed up and fully support automated testing.
- relevant information about tests performed, revision and configuration tested etc.
- test specification following a sound and reliable test methodology.

Test Evaluation:

The evaluation of the test results was important to give a system's reliability estimation. This evaluation was an integral and continuous part of testing, which supported flexible and effective planning and resource allocation to attain reliable systems on time.

Easy Maintenance of Tests and Test Environment:

Adequate maintenance is necessary for reliable and cost-effective testing. It was ensured that the changes in SUT functionality would not make automation useless.

Reuse of Spent Efforts:

Utilizing of previous work is useful to save resources. Several test projects and test phases use the same instrument and simulators, general command sequences implementing standards are labeled, parameterized and stored for common usage.

As the Test Specifications consist of general commands and a test phase specific test Environment Specification exists, the general commands can be implemented in accordance with the current environment. This means that Test Specification can be taken from one environment into another, where they can be used for testing.

It is important to note that each test phase needs individual analysis, therefore reuse must be done carefully and adaptation is always required.

3.4 AUTOMATION METHODOLOGY, ARCHITECTURE AND COMPONENTS

3.4.1 AUTOMATED SYSTEM METHODOLOGY

An operation-centered methodology in which domain practitioners are viewed as 'master craftperson' have been used in the Nortel Network automated test system. The goal is to provide a process during which the designer clarifies, integrates, and synthesizes the system through observation of human performance. Through this process, the designer broadens its set of experiences and thus reduces the brittleness characteristic of most automated systems.

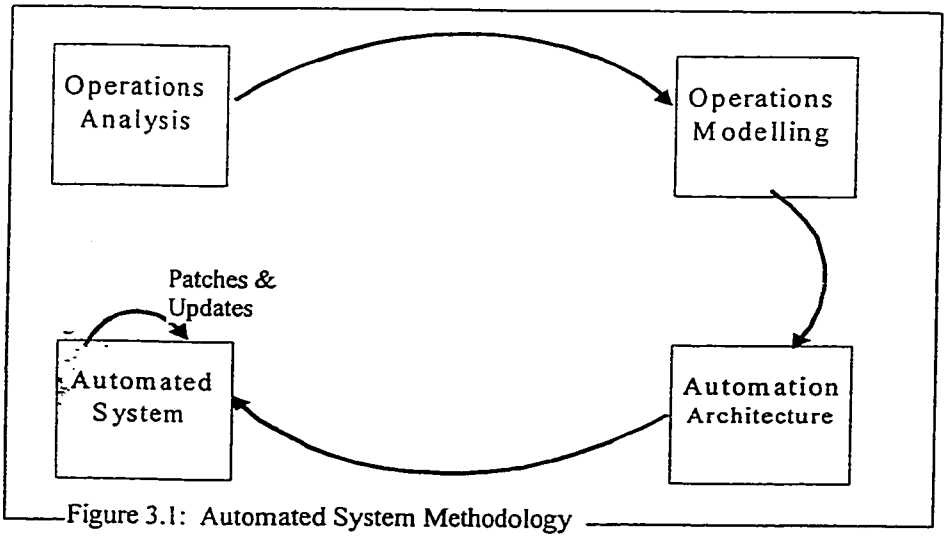

Figure 3.1: Automated System Methodology

The automation methodology consisted of four steps shown above in Figure 3.1. First, an *operations analysis* of domain operations was conducted. This systems-level analysis provided insights into the activities of operators and systems in the domain. The goal was to understand and identify the depth and breadth of operator activities, the system components involved, technological limitations in the domain, and typical operations scenarios. This process produced requirement specification for operations automation.

The requirement specification determined:

- what will be automated; and
- who will maintain the automation.

The goal of analysis was to ensure that any 'automation' be deliberate rather being accidental. Automation requiring human bridges would make any attempt of lights-out automation fruitless.

In step 2, based on the analysis discussed above, an operations model was constructed with the three components shown in Figure 3.2. The systems model represented the physical and functional structure of system components and their relationships.

The activity model organized information about typical operator activities in the domain, including the purposes of activities and how they were carried out by skilled human operators. Connections between the activities in the activity model and components in the systems model identified dependencies between the system components or functions and typical operator activities. In this implementation, *the activity model was an extension of the operator function model* [Mit96]. *Extensions*

included the addition of a plan level to represent pre-specified sequence of activities for meeting operational goals and representations of information flow among activities [Thu94, Thu95]. The plan archive contained information about previously useful sequences or sets of activities. Plans in the archive indexed into the activity model to document the operator activities used to respond to pre-specified system states.

The operations model is a tool for gathering, organizing, and documenting knowledge about the domain. Its form allows domain practitioners to review, correct, and extend the representation. The operations model was constructed through a process of conceptualization, evaluation and refinement. It required a computational form of the model and automation architecture with capabilities to both execute and update the model through the automated process.

Step three produced a computational implementation of the operations model. A proposed implementation is discussed in the next section -- automated system architecture.

The final step in the automation methodology was the automated test system. The automation architecture was placed in the domain and used to automate testing activities. The automated system performs testing for which it has requisite knowledge and integrate new test cases through operator observation and intervention.

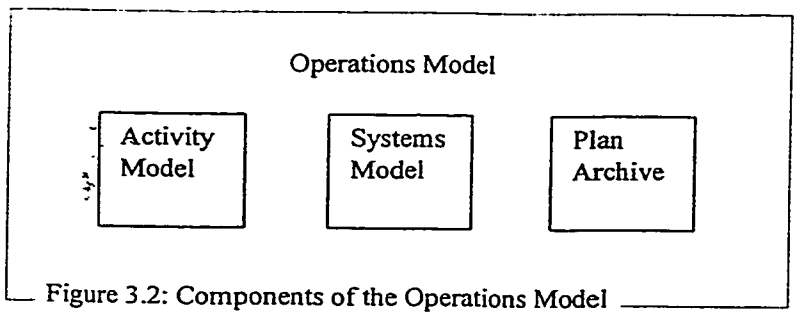

Figure 3.2: Components of the Operations Model

3.4.2 AUTOMATED SYSTEM ARCHITECTURE

Figure-3.3 shows the automated system architecture. The hierarical diagram of the automated system shows all the levels of the test system from the highest to the lowest level as follows:

The test management program, supports test program editing, file management, flow control, and data logging. Thus, allowing the test programmer to concentrate on creating actual tests rather than managing the program development task.

The user-level application program implements and specifies the test sequence. It is written in a general purpose programming language, and calls on the instrument driver and on the application test library. This modular structure maximizes user written functions over multiple test applications.

The application test libraries consist of a set of high-level routines which perform various test functions and measurement operations. It includes the default limit for the various tests as specified by the standards committees.

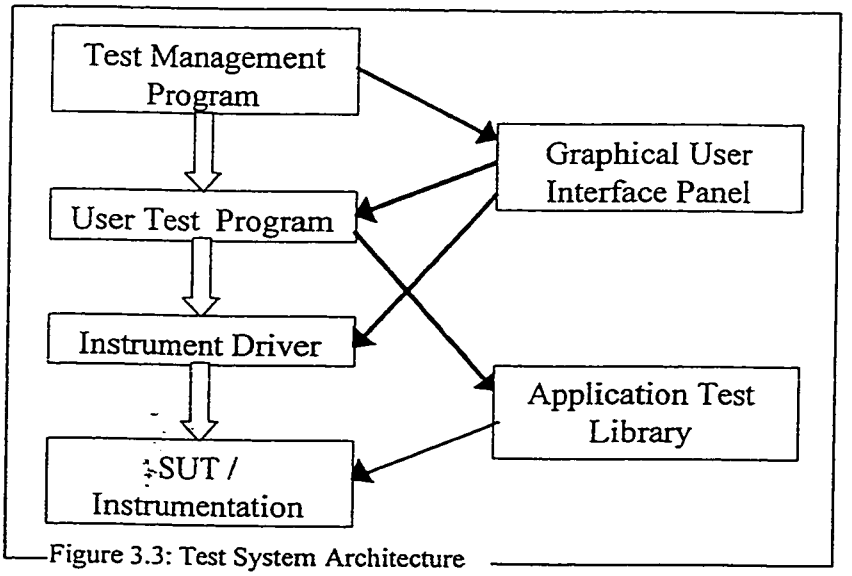

Figure 3.3: Test System Architecture

The instrument drivers control the hardware, and present a common function interface to higher levels. The drivers are used by either the application test library or the user to create customized measurements.

3.4.3 AUTOMATED SYSTEM COMPONENTS

Previous section describes the methodology and architecture of Nortel Network ADSL's automated test system. This section will describe the components which are responsible for achieving the previously described methodology and architecture of the Nortel Network ADSL's automated test system. As shown in the Figure 3.4, the two main components of an automated tests system are **Hardware** and **Software** platforms and their selection depends on many factors. The choice of the hardware mostly depends on the choice of the software. If the performance is an issue, then usually the high-end computers are used. Sometimes the dedicated systems are used for the critical tests. Availability is an another criteria in choosing the platform. Following sections discussed the hardware and software platforms and their components that comprise them.

3.4.3A Hardware Platform

The automated systems have been developed on different hardware platform. In the earlier days, the mainframe and miniframe computers were used for the automated systems. The hardware selection of Nortel Network test system depends upon the availability, the resources, the software which is being used, the performance, and the cost.

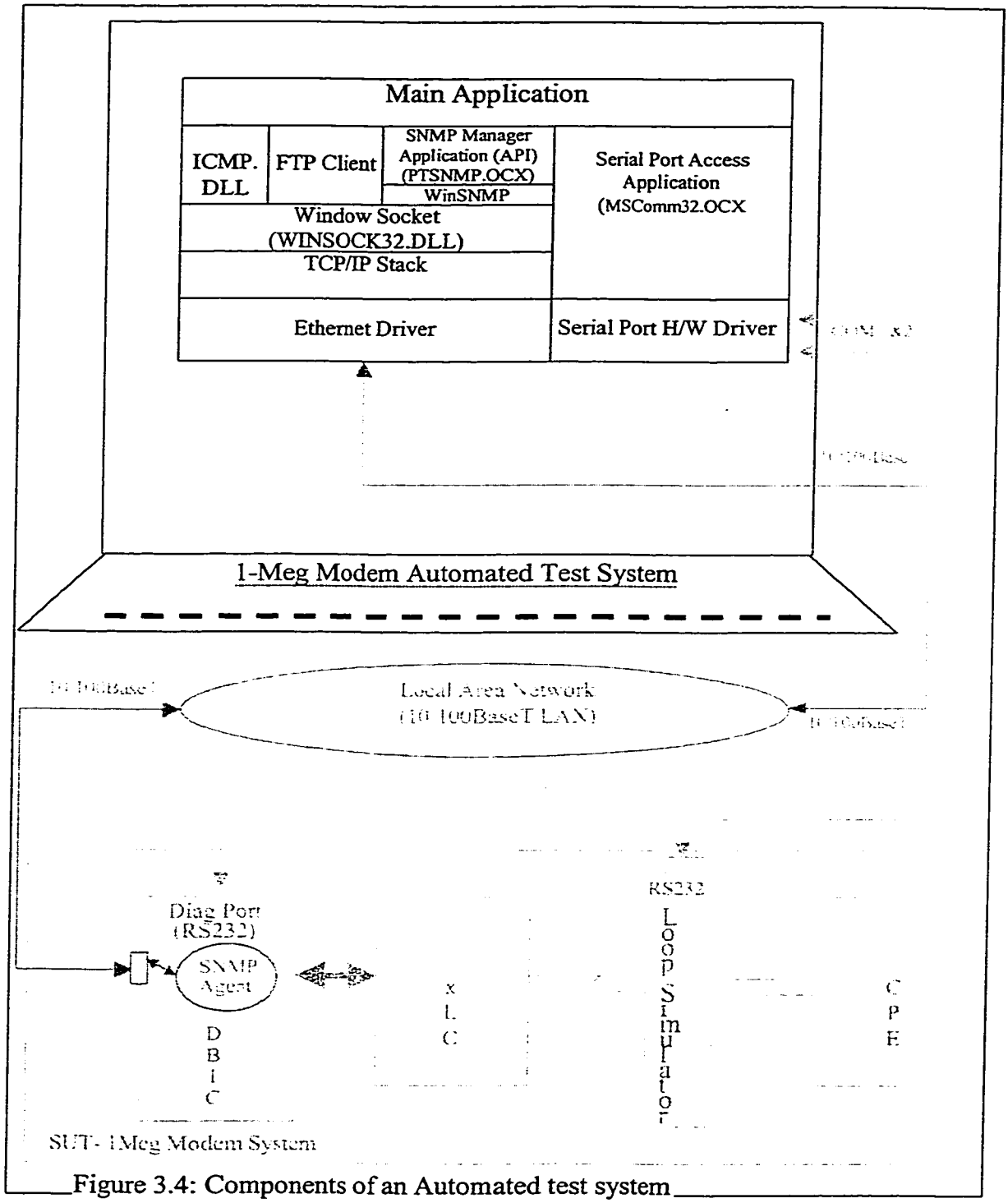

Figure 3.4: Components of an Automated test system

Siemens automated test system, DARTS [MAL94], which was developed in 1994, uses the Unix box and IBM mainframe for its automated test platform. It uses this platform because the IBM mainframe database tools ADABAS and UNIX system V processes were powerful system during that time. A wide variety of applications were also available on those platforms. IRWindows [IRW98] automated system, which is developed in 1998, used Personal Computer (PC) for their automation platform. Scada [MIT98] automated system also developed in 1998, and they also used PC for the automation. Intel's decision support system (DSS) [PIL88] was developed in 1988. It uses the PC and Unix for its system. Nortel automated test system is designed to run on a standard Windows based PC hardware.

It can be deduced from the above paragraph that the systems that were developed during the Pentium era of PC, used PC. Whereas the other system used hardware platform which were strong during that period like IBM mainframes, etc. Due to the distinct advantages of PC, almost all of the automated systems are based on PC platform. To discuss every hardware platform is out of the scope of this thesis. Since the NortelNetworks ADSL's automated test system is based on PC so our discussion will be restricted on the advantageous and the selection criteria of the PC based platform only.

PC Based Hardware Platform:

PC is playing an increasingly significant role in the field of automation technology. *The Open Control User Group projecting the PC – based automation solution* [Bag97]. They propose to use the standard interface to communicate with the SUT, Intermediate Device (ID) and Instruments. Their first step is the implementation of an open bus

system. Their second step is toward the multi-vendor capability i.e. to use PC architecture as an alternative to a manufacture-specific PLC solution and the implementation of standard software.

The advantages of PC are not only its hardware availability but also its expertise and software availability at any part of the world. In the past couple of years lot of improvement has been made in the PC hardware and software. The concepts of main and mini-frame computers are diminishing and PCs are replacing the main/mini-frame computers. Power full PC hardware is available and similarly tons of software available on PC platform. Microsoft Window based operating system and programming languages readily available everywhere in the world. Every sort of APIs are available on this platform especially APIs related with the Internet connectivity either through LAN or through serial COM port (dialup connectivity).

Rigorous competition has brought down the PC price fairly low. *A rough estimation is the automation on PC compare to PLC is 40% cheaper* [Baag97]. The standard data interface save a lot of time as exchange of information between design tools of different manufacturer on a non standard data interface is extremely time consuming. Because of the open control architecture, acceptance of PC based automation is growing steadily in North America, as well as in Asia and Europe.

The use of PC in the Nortel Network automated system, helped the organization to port the system from one testbed to another testbed. Standard interfaces with the SUT and other instruments also helped to avoid any strange problems. Due to the PC based platform, the organization managed to develop a powerful system at a cheap cost.

3.4.3B Software Platform:

The automated systems have been developed on numbers of software platform. Siemen's automated test system, DARTS [MAL94], has been programmed in Unix. Its system Fault Report System is written in a 4th generation language called Natural. Intel's DSS [PIL88] uses Lotus 1-2-3 and AutoCAD on its PC and AutoMOD and AutoGRAM on its UNIX box.

Selecting software platform is difficult due to the availability of large number of software. However, if hardware has been chosen, then selection depends upon only to those software which are compatible with the chosen hardware. The general criteria of selecting the software for automated test system are:

- ease of integration with other software
- availability of APIs
- strong Graphical User Interface (GUI)
- rapid application development
- less cost / high performance

As mentioned above, tons of software are available in this era. Nortel Networks chooses the Visual Basic (VB) for its automated test system's software platform. APIs and other lower level software components are also being used by the Nortel automated test system. This section discuss the advantage of VB over other software platforms and the justification of selecting VB for the Nortel Networks automated test system's software platform. Later the APIs, DLL files and other drivers that are used in the

automated system are also discussed. Note: The details of VB implementation and its usage with ActiveX control, APIs, DLL files (either directly or indirectly through APIs), and the drivers are covered in Chapter 5 – Software Testing.

(i) Visual Basic – the Coding Language:

Microsoft's Visual Basic (VB)— a new programming language in the past six years —has evolved into a major development environment that *covers every aspect of programming, from automated application to database programming, and from financial application to developing Internet components* [PET97]. These features made VB to quickly rise to a dominant position in software development. *Languages like Ada, Cobol, Fortran, Pascal, and RPG are also being refurbished with new graphical user interfaces (GUIs) and visual programming IDEs* [Sur99] just like Java and Visual Basic.

Visual Basic, along with Paradox and Dbase, was one of the first languages designed to take advantage of GUIs with visual programming in mind [Sur99]. It allows building menus, forms and reports for a program through its visual drag-and-drop operations. The skeleton program, which was developed while creating the prototype of a program's interface, can be completed by filling the remaining codes.

Visual Basic was the first language to use the third-party components such as VBX and now ActiveX (e.g. ptsnmp32.ocx and ptftp32.ocx). The slightly simplified form of VB called Visual Basic for Applications (VBA) is now the macro language for Windows applications (e.g. MS Excel). Therefore, it is the best rapid application development (RAD) programming languages available today.

Today, two primary trends in programming are being noted. The first is the programming complexity of distributed processing over wide area networks (WAN) is finally falling out of the realm of high-risk, and specialist-only development into a demanding but achievable task. The second major being the object-oriented (OO) languages.

Visual Basic has taken the slow-but-sure approach, so OO techniques and methods have been gradually incorporated into the language. Polymorphism and dynamic linking, followed by encapsulation and hiding, were added in Visual Basic 3.0 through 5.0. *Inheritance may be added in version 7.0 or 8.0 (or may be a part of a new Microsoft language code-named Cool)* [Sur99]. The bottom line is that Visual Basic is a hybrid OO language like C++.

As discussed above, visual programming and the use of components is one of the pioneering strengths of Visual Basic. OO techniques make GUI and general program development using components even easier. But these techniques have brought forth two very real needs. The first need is to reduce the cost associated with the much greater complexity of distributed programs. Distributed processing has brought two real benefits as well: the elimination of both isolated islands of information and inefficient use of data and program resources.

A second need identified is to develop software much faster. One of the essential advantages of Visual Basic is that *programmers can develop a wide range of programs so much faster than compared with other languages like 4GLs (fourth-generation languages) e.g., Focus, PowerBuilder and Uniface* [Sur99]. These languages once held

an advantage in ease and speed of development but at the cost of poor runtime performance.

Java and Visual Basic (3GLs - third-generation languages), with their visual programming, components and use of clever wizards or third-party design tools, have matched if not surpassed 4GLs for speed of development and have superior runtime performance. Compared with other 3GLs, such as Ada, C/C++, Cobol, Fortran, and Pascal, Visual Basic has following compelling advantages:

- extensive APIs for every aspect of Web, client/server, and n-tier programming;
- several modes of deployment including EXEs, components;
- rapid development through visual IDEs; and
- superior visual programming development environs with many third-party suppliers of tools and components.

The runtime performance of Visual Basic has steadily improved to the point that it is now approaching within 10 to 30 percent of the fastest programs developed in C/C++, Cobol, Fortran or Pascal.

Visual Basic (VB) runs on Windows 9x, NT (partly on Alpha versions), 2000 (the new name for NT 5.0), and CE (partly). However, Visual Basic no longer supports Windows 3.x. Visual Basic has become, through VBA, the scripting language for Windows applications. This is important because more systems will be built directly into key applications such as AutoCAD, Excel, Notes, Project, SAS, SAP, Word and hundreds of other major Windows programs.

Visual Basic is making tough Windows programming very approachable. And the text editor and debugger in Visual Basic's IDE are the best in the business for client-side

development. For PC-based applications, where Windows dominates the desktop, Visual Basic is flexible to interface with the existing commercial or homegrown Windows applications through Net, browser interface, COM and other components.

Visual Basic has been selected because of the above mentioned advantages, especially power full GUI, a rich set of APIs and OLE (Object Linking and Embedding) property. As mentioned earlier, the implementation of GUI, usage of APIs and ActiveX component and their impacts on the automated system are discussed in Chapter 5, "Software Testing". Before discussing the high level APIs, which are directly used by the main VB application, it is important to discuss first, some of the lower level APIs (DLL files) that are frequently used by either the main application or its high level APIs.

(ii) Win32 API – Dynamic Link Libraries (DLLs):

Win32 APIs are lower level APIs and are a set of functions that any programmer can use to access all of the services of Windows [Boc98]. No matter what language is being used, API is an essential part of the program. No language could provide all functions. If the function exists in API, it is a quick and easy way of doing what is required. Sometimes, languages provide necessary function by calling themselves the APIs. Windows groups these APIs into Dynamic Link Libraries (DLLs).

According to Microsoft, *DLLs are files that contains one or more functions that are compiled, linked, and stored separately from the processes those use them* [DLL-1]. The operating system maps the DLLs into the address space of the calling process when the process is starting or while it's running.

DLLs are required because of the problem in the **static linking**. Basically, when an application is compiled, all of its code is linked into the executable (EXE) that Windows can use. However, if the changes are made in the code, these changes can not be seen by the executables unless the application is recompiled. But the problem is this that only executables are distributed and they reside only on the user's computer. No doubt that this gives a complete control on the source code. But, in the world of business application development, these users may be anywhere in the globe and this could lead to big distribution problem. The chances are that the different programs use the different version of executable. These problems are solved in the **dynamic linking** (DLLs). *DLLs are centralized in a binary file that has a specialized executable format that Windows can read* [DLL-1]. No one else can change these DLLs file. Once the file is on the PC and the program need to use these DLLs file, they will all use the same file. They do not need to recompile the code to generate the same executables.

DLLs are required to run all Windows programs, including Windows, but they run in the background and we never see them. When a program requires a DLL to run, and can't find it, it will not run because it will miss the DLL to perform that particular task. Following are the details of specific DLLs which are used in the automated test system.

Windows Sockets:

Windows Sockets, or Winsock.DLL, is a DLL which allows application to talk over a network, usually the Internet. The layer position of Windows Socket in the automated test system is shown in Figure 3.4 and can be seen that the FTP, SNMP and ICMP application of the automated test system access their corresponding applications on

the network through Windows Socket APIs. Windows Socket specification defines a network programming interface for Microsoft Windows which is based on the "socket" paradigm.

The Windows Sockets Specification is intended to provide a single API to which application developers can program and multiple network software vendors can conform. Furthermore, in the context of a particular version of Microsoft Windows, it defines a binary interface (ABI) such that an application written to the Windows Sockets API can work with a conformant protocol implementation from any network software vendor. This specification thus defines the library calls and associated semantics to which an application developer can program and which a network software vendor can implement.

Applications which are capable of operating with any "Windows Sockets Compliant" protocol implementation will be considered as having a "Windows Sockets Interface" and will be referred to as "Windows Sockets Applications".

Windows SNMP:

The Windows SNMP API (WinSNMP.DLL) specification defines a programming interface for network management applications running under the Microsoft Windows family of GUI/operating system products, enabling those applications to make use of a logically external SNMP engine or service layer [Nat95]. Figure 3.4 shows Manager SNMP-API uses the WinSNMP to access the SNMP agent and vice versa.

The simple network management protocol (SNMP) is developed to provide a basic network management with no-frill service for TCP/IP based environment. It is a request-response protocol used to transfer management information between entities acting in a manager role (automated test system) and entities acting in an agent role (SUT).

The Windows SNMP API specification introduces no constraints on the use of SNMPv1 or SNMPv2, nor on the functionality supported by those protocols as prescribed in the relevant Internet RFCs. SNMPv1 is seen as a subset of SNMPv2.

The purpose of WinSNMP is to promote the development of SNMP-based network management applications running under the Microsoft Windows family of GUI/operating system products. WinSNMP provides a single interface to which application developers can program and multiple SNMP software vendors can conform. This specification thus defines the procedure calls, data types, data structures, and associated semantics to which an application developer can program and which an SNMP software vendor can implement.

A process may contain one or more threads of execution. WinSNMP supports multi-threaded Windows processes.

WinSNMP offers these major benefits--all intended to accelerate the development, dissemination, and use of SNMP network management applications [Nat95]:

- SNMP enabling technology for functional network management applications (i.e., "hides" ASN.1, BER, and SNMP protocol details);
- SNMP service provider independence. A WinSNMP application will run against any compliant WinSNMP implementation; and
- uniform SNMPv1 and SNMPv2 support. A WinSNMP application does not have to know the SNMP version level of the target SNMP entities acting in an agent role. The WinSNMP implementation will perform any and all necessary mappings between SNMPv1 and SNMPv2 in accordance with the appropriate RFCs.

The softwares which conform to WinSNMP specification are WinSNMP compliant software. In the specification, there is no stress for any particular implementation. The specification allows following overlapping SNMP implementation:

- Level 0 = Message encoding/decoding only
- Level 1 = Level 0 + interaction with SNMPv1 agents
- Level 2 = Level 1 + interaction with SNMPv2 agents
- Level 3 = Level 2 + interaction with other SNMPv2 managers

Windows ICMP:

Microsoft's ICMP.DLL is an undocumented API for sending ICMP echo packets, also called "pings," after the submariner's term for sonar signals. This API works fine and is present on all current Windows boxes with Microsoft Winsocks. The implementation details of ICMP are discussed in Chapter 5 "Software Testing".

(iii) High Level APIs:

High Level APIs provides an explicit programming interface that is easy to use and hides many details of the communication protocols actually taking place by Win32 API. The main application of the automated test equipment uses high level APIs for accessing the SNMP and other network protocols by avoiding the details of WinSNMP and WinSock libraries. Generally these high level APIs automatically taking care:

- Setting up connections (creating socket, resolving names, setting up active and passive connections, etc.)
- Closing connections (destroying sockets and buffers, etc.)
- Data buffering and application flow control
- Processing of user interface messages

- Safe error recovery
- Optimized design with event notification

Manager SNMP and FTP client are the two major High Level APIs which are extensively used in the automated test system. These two APIs provides an easy interface to the main VB application.

Manager SNMP-API:

Chapter 5, "Software Testing" discussed the detail implementation of Manager SNMP-API so here we only define briefly the Manager SNMP-API.
The main VB application uses the Manger SNMP-APIs to access the SNMP agent. Manager SNMP-API in its turn uses the WinSNMP DLL file for accessing the SNMP agent.

PTSNMP ActiveX control (ptsnmp32.ocx) is responsible for providing the SNMP Manager-API. Like all other ActiveX control, it also has methods, events, and properties.

FTP Client API:

Chapter 5, "Software Testing" also discussed in detail about the implementation of FTP Client API therefore we also discuss it briefly here.

FTP client API provides an easy method to main VB application for doing FTP and the file transfer rate. PTFTP ActiveX control (ptftp32.ocx) is responsible to access files stored on the FTP server. In order to store and retrieve a file from the server, it establishes two connections with server. One is the control connection by using Telnet and other data connection by using TCP. It uses the WinSock DLL file to do the Telnet and TCP.

3.5 TESTING ENVIRONMENT

So far, our discussion has revolved around the different components of the automated system. This section will deal with the connection & integration of the components with each other. Figure 3.5 explains the automated system test environment. Each component of the test environment is described below.

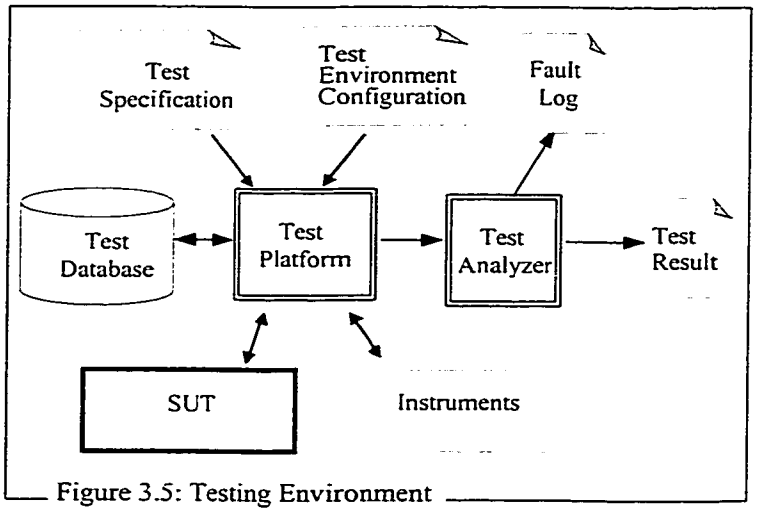

Figure 3.5: Testing Environment

Test Specification:

Information about the test cases to be performed is kept in the test specification. It specifies objectives, description, and the expected results of each test case. It provides references to the Test Environment Configuration for extracting the test configuration for every test case and the option to select individual test case or multiple test cases.

Test Environment Configuration:

It contains environmental configuration of the test cases. It specifies which hardware, software, instruments and simulator functionality to be used & how these are interconnected in the system.

Test Database:

It contains the implementation of the test cases which includes SUT, instruments and simulators. It also does the variable-to-value substitution.

Instruments/Simulator:

Traffic Generator, Performance Monitoring, Loop Simulator, etc are the example of instrumentation/simulator.

Test Analyzer:

It processes the data and produces a test report and a fault log.

Test Result:

It contains information on the executed tests. It records the number of tests executed, number of times it passed and failed, resources used, and time taken to complete the test, etc.

Fault Log:

It logs the failures occurred during test execution. These logs help to rectify the system's fault.

Test Platform:

The platform has been discussed earlier for the automated system. The chosen platform coordinates the testing by:

- extracting the test case(s) from the test specification and evaluating it;
- extracting the correct test configuration from the test environment configuration, and evaluating it;
- extracting the implementation of SUT, instruments & simulator from the test database, checking all the user's selected parameters and then executing the implementation; and
- verifying and generating the test result and fault log.

CHAPTER 4
HARDWARE TESTING

4.1 INTRODUCTION

ADSL systems are built with increasing complexity, which renders fault testing an important and indispensable part of the manufacturing and maintenance process. As per the classical theory, the fault detection and fault location are the two sub categories of the fault system. Three types of fault that most likely to occur on fault-model/system-fault-model are: Stuck bit (s-a-0, s-a-1), Coupling and Pattern Sensitivity.

A number of basic analytic and heuristic methods, viz. the fault table method, the path sensitizing and equivalent normal form (ENF) method, the Karnaugh map method and tabular method, the ENF-Karnaugh map method, the spoof method, etc. exist for the solution of the fault detection and location problem...[Das97]. Based on these methods, computer programs can be written to generate complete fault detection test sets for checking combinational circuits. Two computer programs that are used to test the combinational circuits are DALG – II and Test Detector.

The testing of digital systems have progressed from the state of testing the operations codes of the machine instruction set in the early days, to the current hardware assisted software aids for speedy determination and identification of faulty elements. In the Very Large Scale Integration/Large Scale Integration (VLSI/LSI) era, the problem of diagnosis is shifted from locating the faulty discrete component to the identification of the replaceable module (chip/board) in which the faulty component is resident. There are hundreds of automated test systems available to test LSII, VLSI, and multiple modules

systems. 1-Meg Modem automated-hardware-test-system is also an effort to test the hardware of Nortel's 1-Meg Modem system.

1-Meg modem is a complete system using multiple modules. At the time of development of 1-Meg Modem system, there was a requirement to quickly test the functions of prototype module. 1-Meg Modem automated test system was developed to fulfill the testing requirement discussed above. Later on the automated test system was extended to test the design, integration, regression and production testing.

4.2a Hardware Functional Testing

The multiple module system provides more powerful function. These complex device modules must be tested for correct functional operation to obtain certain assured reliability before being adapted in the system. The high complexity of VLSI on individual modules makes Gate-Level testing very difficult and expensive. Techniques for design testability or built-in-self-test consider the testing problem during design stage of digital devices as these approaches can not be applied on off-the-shelf-components.

Typical failure characteristic or mean time between failure data are not always readily available [Bon94]. Functional-Level testing is the natural approach to assure that a system with off-the-shelf-components is working properly.

Functional-Level testing uses higher level of system representation than Gate-Level testing. In Functional-Level testing, functional faults with respect to specification are tested instead of signal faults at the inputs and outputs of a logic gate or interconnections among gates in Gate-Level testing. Figure 4.1 shows the contrast between Gate-Level and Functional-Level testing. When Functional-Level testing is to be

performed, only the functions of modules and their interactions are tested. The functional faults of f_1 (e.g. incorrect addition) and f_2 (e.g. incorrect control signal decoding) in Figure 4.1, are examples of assumed functional faults for data operation and control modules respectively. Functional-Level testing in testing Microprocessors usually means the testing of each instruction in the instruction set based on its specification.

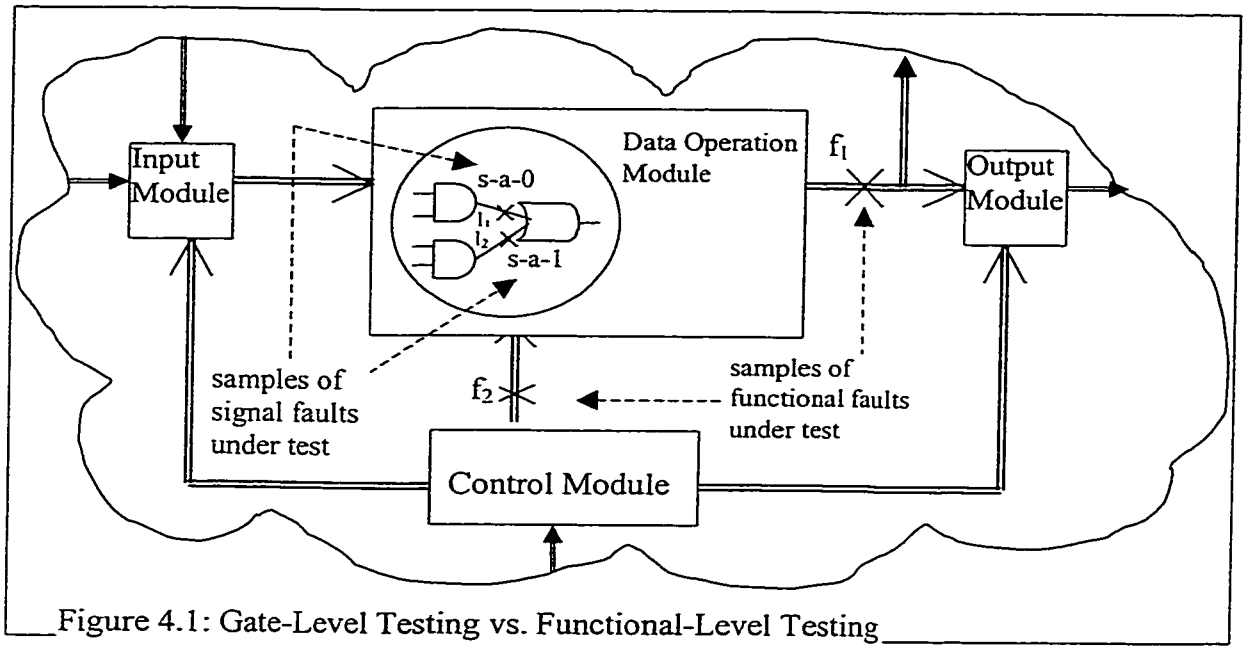

Figure 4.1: Gate-Level Testing vs. Functional-Level Testing

The development of hardware specification is another domain since a fault in a specification is not due to worn out component but is instead due to a perceived inconsistency between the actual specified behavior and the intended specified behavior [Bon94]. In many cases, systems in this domain (hardware specification) may also satisfy. One can assume that behavior of a specification itself is completely observable. However, when a specification does not have specification of its own and the tester is not in a position to answer all queries regarding intended behavior then one can conclude that

knowledge of the specified system behavior is incomplete. A complete functional testing effort includes generally the following steps:

1) describe the system under test in terms of a well- defined description medium;
2) set up a suitable Functional-Level fault model(s);
3) develop a testing algorithm to test the comprehensive set of functional test patterns (or test procedures); and
4) validate the effectiveness (e.g., fault coverage and testing complexity) of the functional tests by a fault simulator.

4.2b System's Functional Testing

In general, the systems available to users can be categorized into the following types:

1) microprocessors
2) peripheral supporting modules
3) special-purpose functional modules
4) memory/register

Functional-Level testing receives different degree of attention among these four types according to their diverse functional complexity. As shown in Figure 4.2, microprocessors usually have the highest functional complexity among these four types. Peripheral supporting modules perform interfacing and inter-communication with other systems and also have high complexity. Special purpose functional modules, such as modules designed for fast execution, have medium complexity. The memories/registers receive the least emphasis in functional testing due to their low functional complexity.

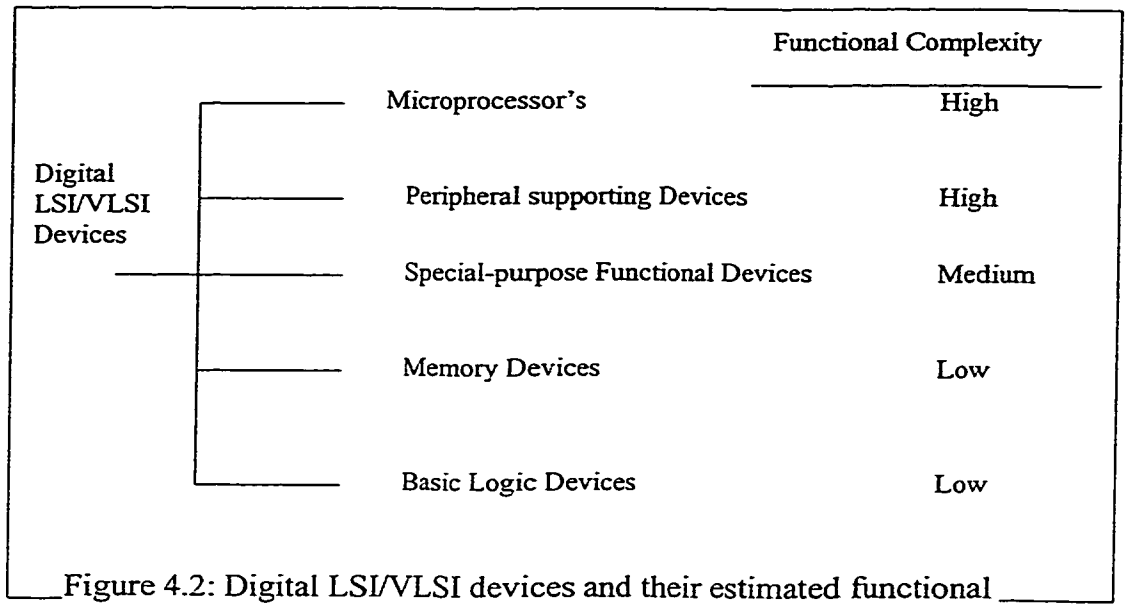

Figure 4.2: Digital LSI/VLSI devices and their estimated functional

Testing of modules can be performed using different philosophies by different people based on how much implementation information is available and how much confidence of fault detection is required. In various kinds of applications, especially those requiring critical reliability, users must assure the correct functional operations of modules.

The order of test sequence is of major concern during functional testing. The order is usually derived from the functional description of system, which may be structural or behavioral.

4.3 FUNCTIONAL TESTING TECHNIQUES

After advent of LSI/VLSI, a lot of work has been done on the testing of these devices especially in late 70s and early 80s. This section highlights some of the techniques and approaches, which were adopted in the functional testing of VLSI/multiple modules systems. The idea of discussing these techniques is to show that initially the hardware testing was confined to the component level. With the advancement of technology, new components are fabricated, the sizes of components are reduced, integrated chips are developed and the circuit board densities are increased. As the more components are placed on the circuit board, the functionality of board increased and the component level testing is not only become difficult but also time consuming. Among several approaches, one approach is to test explicitly the functionality of the LSI/VLSI, module, and multiple modules system and in this way the individual component will be tested implicitly.

(i) Abstract Execution Graph Technique:

This technique is used for the functional testing of microprocessors. Roboch and Saucier [Rob80] proposed that, each instruction of a microprocessor is represented by an "abstract execution graph". Test generation of microprocessor can be performed based on the set of abstract execution graphs derived from its instruction set.

An example of "ADDA N, X" from the MC6800 microprocessor is given in Figure 4.3. This instruction first computes the effective address by adding N to X, using this address for fetching the operand. The operand is then added to register 'A' and the sum is stored in register 'A'. The Abstract Execution Graph for every instruction in MC6800 can be derived similarly.

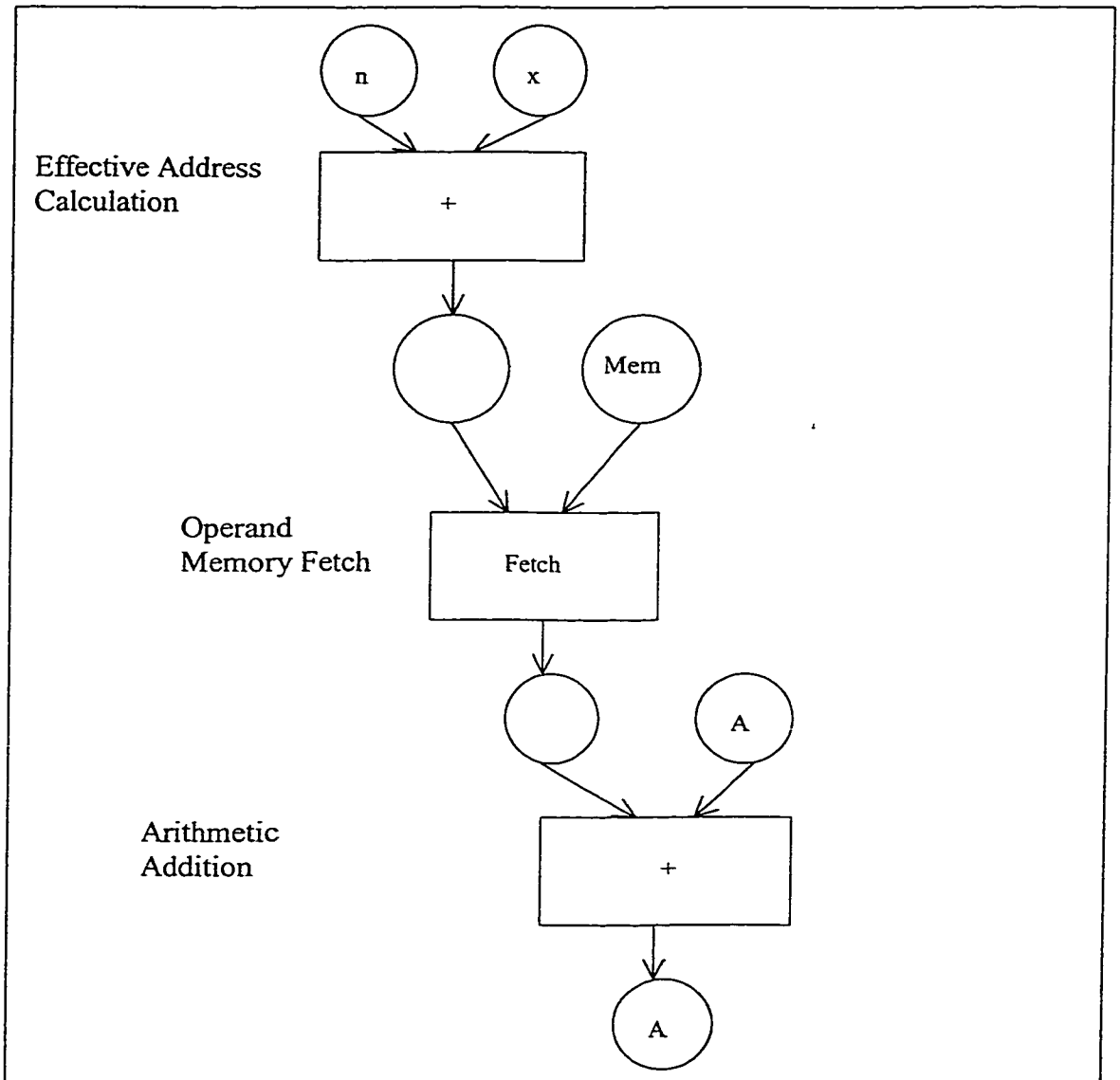

Figure 4.3: An illustrative example of "abstract execution graph" of instruction

(ii) System Graph Technique:

With a different graph theoretic approach, Thatte and Abraham [Tha80] proposed another method which uses register transfer level description of microprocessors. Instructions are classified into three types: Transfer (1) type, Manipulation (m) type, and Branch (b) type. A directed Graph called System-Graph (S- Graph) quite different from the 'Abstract Execution Graph" is derived based on the functional, structural information and instruction set.

An example of an S-Graph based on part of a hypothetical microprocessor is shown in Figure 4.4. The superscript in each edge indicates the subinstruction sequence. I_2^1, for example, represents the first subinstruction of I_2.

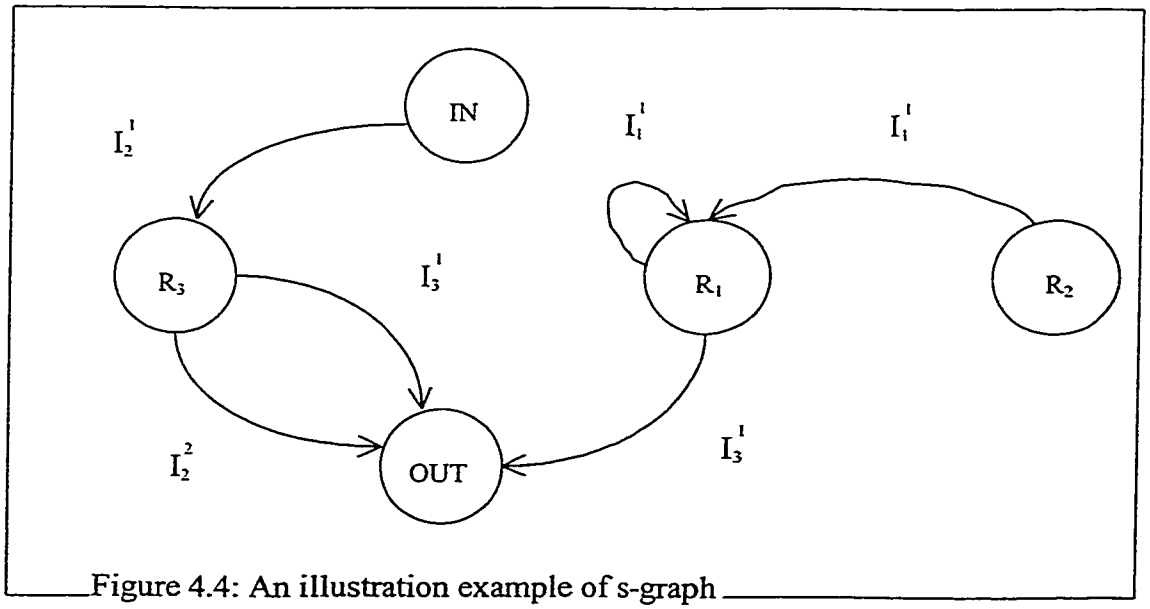

Figure 4.4: An illustration example of s-graph

(iii)State Transformation Graph Technique:

Lai [Lai81] proposed an ambitious functional testing methodology for general digital systems. A new graph language called State Transformation Graph (STG) language was designed to describe the digital system to be tested.

The STG has its root in data flow graphs and can describe digital systems at all levels ranging from user-defined primitives of arbitrary complexity down to logic gates. In Figure 4.5, a rough STG description of the PDP-8 mini-computer is shown.

(iv)Register Transfer Language (RTL) Technique:

The RTL description of a digital system can be reasonably derived from the function description and user-available information (e.g., user manuals and application notes) of the system. Using RTL, the behavior of an LSI/VLSI device is comprehensively described, and functional faults derived from them are studied. Su and Hsieh [Hsi82] proposed two approaches for functional testing based on the RTL description. The first approach constructs a data graph from RTL description and uses existing algorithm such as the D-algorithm or path sensitizing method to generate the test inputs for functional faults. In the second approach, the symbolic execution technique is employed and a test generation algorithm called S-algorithm is developed to generate test input patterns for detecting functional faults in the RTL description.

(v) The Microprogram-Oriented Technique:

This technique is proposed by Annaratone and Sami [Ann82] mainly for functional testing of microprocessors with microprogrammed control. It functionally describes a microprocessor as a set of microprograms derived from information available to users.

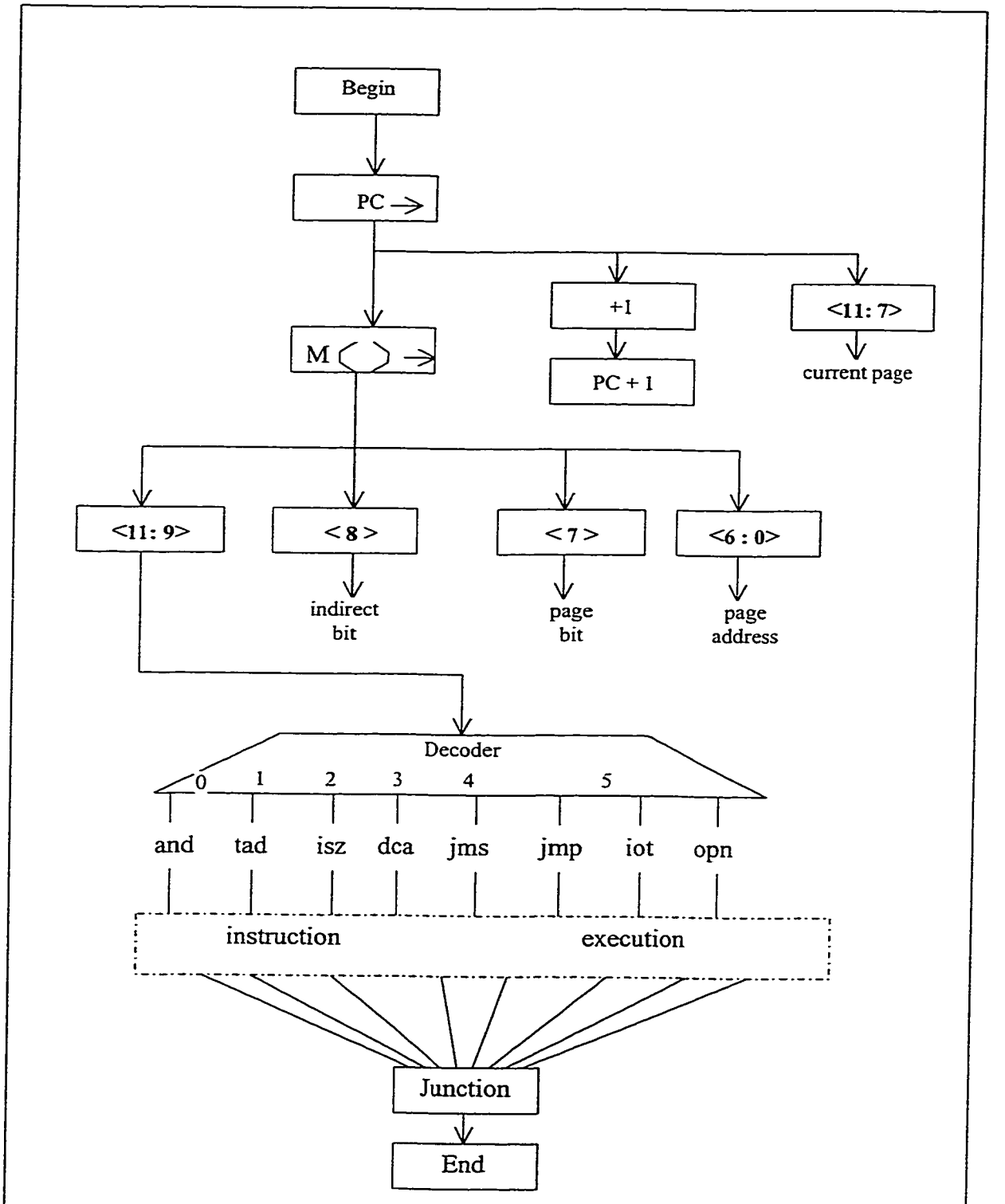

Figure 4.5: State Transformation Graph of PDP-8: instruction decoding

As an illustrative example, the instruction "NOP" in Z-80 microprocessor will be tested first since the microprogram of this instruction only consists of instruction fetching and decoding phase, and hence its cardinality is least in the whole instruction set.

(vi) The Extended D-algorithm Technique:

The extended D-algorithm technique is designed for the Functional-Level testing of general digital systems. It is a direct application of D-algorithm into digital systems described by a high-level hardware description language.

The approach proposed by Levendel and Menon [Lev82] uses computer hardware description language (CHDL) constructs (e.g., if-then-else, case) and functional operators (e.g., addition) to describe the behavior of a general logic network.

Breuer and Friedman [Bre80] proposed a system which may generate tests for complex sequential circuits composed of functional primitive logic elements such as counters or shift registers. Using this concept, a digital circuit may be described as interconnections of basic logic gates, flip flops, and functional primitive elements such as counters. Then, a path sensitization type test generation algorithm like D-algorithm can be applied for testing of each test point within the "mixed-component" circuit.

(vii) Behavioral-Level Technique:

Behavioral-Level description is the abstract description of digital systems using high level hardware description statements like "if-then-else", "do-while", "repeat-until"" and "case". Using behavioral-level model, the total device function is described by a relatively abstract behavioral description. Behavioral-Level testing is to test digital systems by considering faults which may only occur in behavioral-level model [Das97].

(viii) Memory Functional Testing Techniques:

Work has been done in the area of static functional testing of LSI/VLSI semiconductor memories. The functional testing of memories from users' point of view is the application of a sequence of "read" and "write" operations to check the storage function of each memory cell.

There are several other special fault models developed for testing memories for sensitivity to multiple-bit patterns. Figure 4.6 lists some test techniques and their corresponding complexity as a function of the number of bits.

TEST	Complexity(in terms of bits)
Checker board of 1's and 0's	N
Marching 1's and 0's	N
Shifted diagonal	$N^{3/2}$
Walking 1'0 and 0's	N^2
Galloping 1's and 0's	N^2

Figure 4.6: Some popular memory testing techniques and their testing complexity

(ix)Current Industrial Automated Test System:

The five industrial automated test systems, Sanata Barbara's IRWindows test system [IRW98], Ericsson Telecom automated test system [PER94], Siemen's DARTS – automated system [MAL94], Intel Automated System [PIL88], SCADA automated system [MIT98], which are being used as reference systems, do not perform hardware

tests apart from Ericsson Telecom automated test system. However, the detail of hardware testing of Ericsson test system is not available except that

> *...hardware reliability is according to classical theory* [PER94].

Excluding the hardware testing in the industrial automated test system raises the point of 'why the industrial automated test systems are not incorporating the hardware testing?.' Is it due to

> *The large number of testing technique developed is not supported by a large body of empirical evidence describing the relative merits of each technique. Many of the techniques are not prescriptive in their selection of test data* [RON95].

Or, in this high competitive era, the organizations do not start to build hardware from scratch. They usually upgrade their existing product by adding additional circuitry/modules or they get different modules from related technology and integrate them by adding minimum circuitry. In this way they do not test the whole hardware. They only need to test the part of the hardware. To develop an automated test system for that small portion of hardware may not be desirable as it may be an expensive solution or the time taken to test this small portion of hardware is less than the time taken to develop the automated test system.

(x) 1-Meg Modem Automated Hardware Test System:

1-Meg Modem (SUT) is a complete system. It comprises of multiple modules. Automated test system considers it a-system-fault-model and applies the Functional-Level testing on it to find any hardware problem in/among the module(s).

Two sets of hardware tests are performed by the automated system. In the first one, it tests the stuck-bit-test on all the direct registers of modules by performing marching

test, galloping test, write & read all zeros, write & read all ones, write & read alternate one & zero, etc. The second set is performed on the indirect/remote/programmable registers, which contain the instruction word (opcode) to execute specific functions on different modules. In this test, macro command (set of instruction) is executed step by step on the programmable registers to activate the different functionality of 1-Meg Modem in such a way that finally the complete 1-Meg Modem system is up and operational. Since different commands are responsible for different modules functionality so it is a good test to test the functionality of each module, functionality among modules and the functionality of the whole system. Later, end to end datapath is tested by using the various packet counters of the datapath.

4.4 GENERAL HARDWARE ARCHITECTURE OF 1- MEG MODEM

In order to understand that how the Functional-Level testing work, it is important to know the general hardware architecture of 1-Meg Modem system. In this section, we will describe the major blocks of 1-Meg Modem hardware architecture and information flow among these blocks.

1-Meg Modem (SUT) is a proprietary product of Nortel Network. Therefore, due to security reason it is not possible to discuss its architecture in detail. However, general hardware information of ADSL product architecture is available in the literature and some hardware information of 1-Meg Modem is available publicly [Nor99-1, Nor99-2, GOR98, Net99, Sum99, Int96, TR998, CMP98, Inf98]. By using this information, the general block level hardware architecture is shown in the Figure 4. 7. From the hardware testing point of view this information is enough to understand that how 1-Meg modem

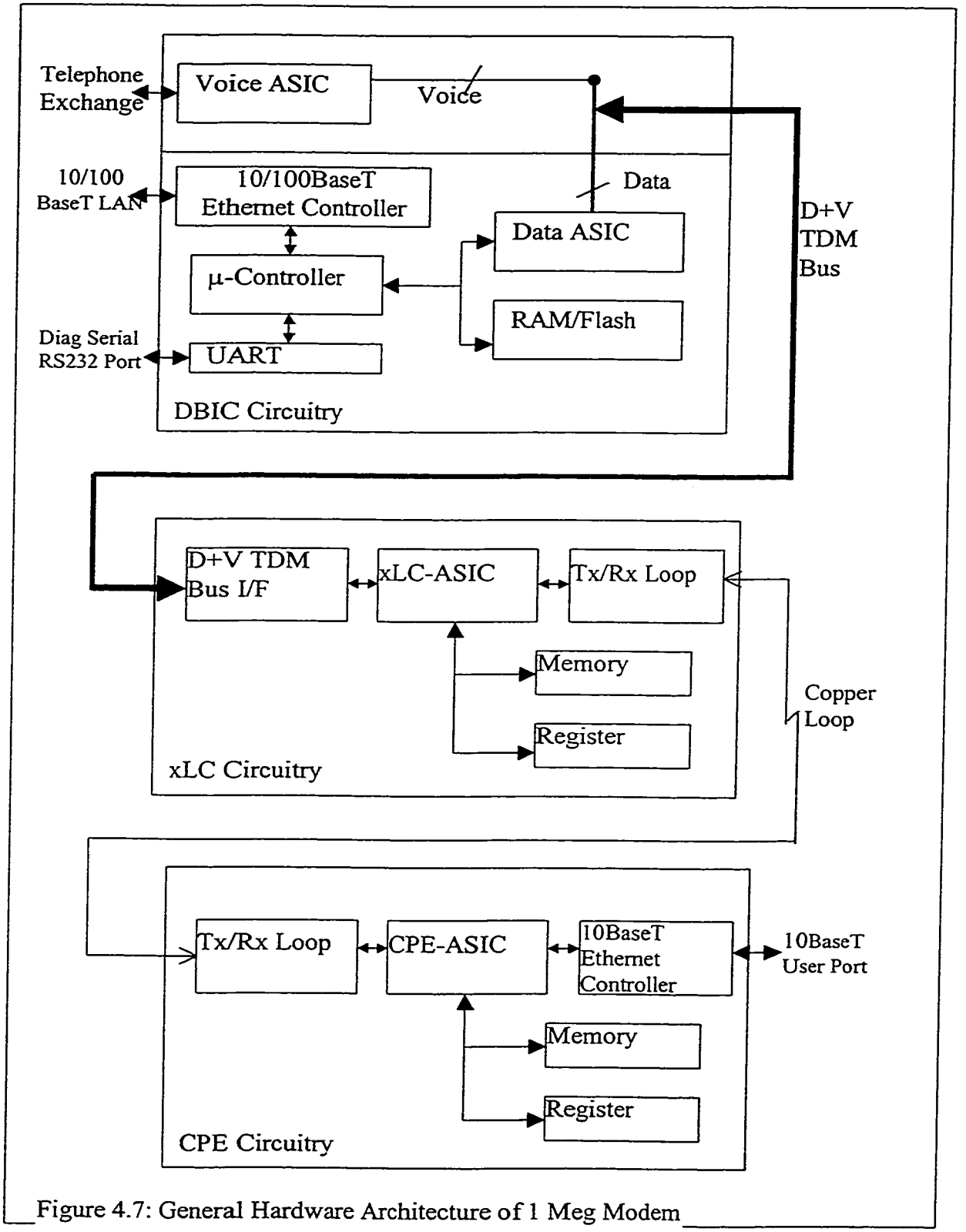

Figure 4.7: General Hardware Architecture of 1 Meg Modem

system works and how its different modules/components interact with each other. Secondly we are using the Function-Level testing so information of each and every component is not required. Block level diagram and basic knowledge of electronic component/module are enough to understand which component/modules are involved in the execution of one instruction. As can be seen that the execution of one instruction involves most of the major hardware components. The writing and reading on a line card register involves the following operations and components/modules.

1. Automated Test system sends a command to write a particular pattern of bits into the line card register through the 1-Meg Modem diag (serial) port;
2. Micro-controller communicated with universal asynchronous receiver transmitter (UART) to fetch the write command. The component involved in fetching the write command from the serial port include the following shown in Figure 4.8:
    a. μ-controller
    b. internal data bus
    c. address decoder
    d. programmable clock chip
    e. UART
    f. RS232 cable

Note: All these components are indirectly tested as our testing is a functional-level testing. If any of the components fail/misbehave, the system will not perform that particular function (writing particular pattern of bits in the line card register) and the fault will be detected;

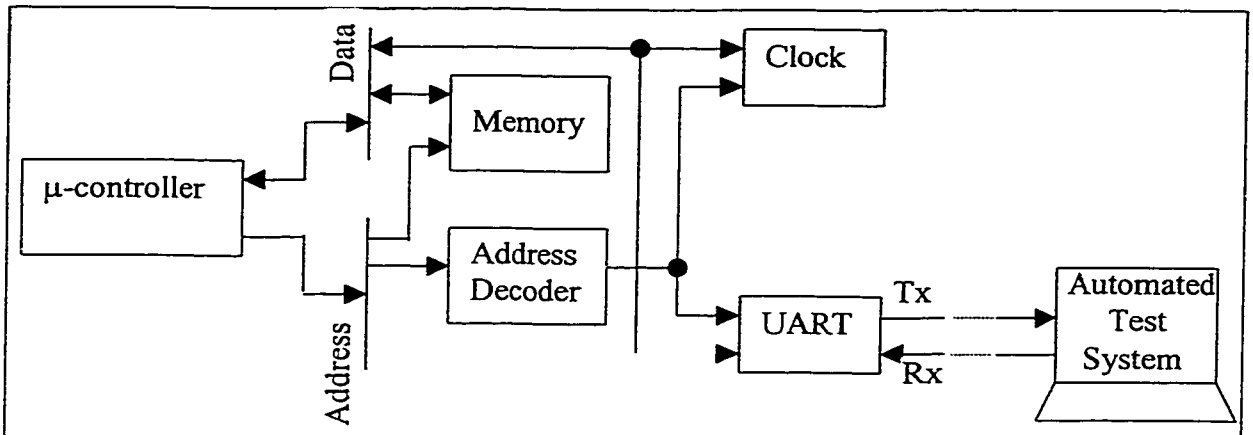

Figure 4.8: Serial Communication Block Diagram

3. µ-controller store the receive data into memory. The components involve in this execution are:

    a. µ-controller

    b. memory

    c. data bus

    d. address bus

    e. control lines

    Note: If there is any test like marching tests, galloping test, etc, is executed on the line card register, these tests implicitly will be executed on the some portion of the memory;

4. Data-ASIC handshakes with µ-controller and reads the data from memory and store it in its buffer. It then handshakes with the Voice-ASIC and detect any empty slot on the Data+Voice TDM bus (D+V TDM bus). Insert the data on the D+V TDM bus empty slot. The components involved in this cycle are:

a. Data-ASIC

b. μ-controller

c. data bus

d. address bus

e. control lines

f. Voice-ASIC

g. Data+Voice TDM bus

During the execution of this cycle, multiple processors (μ-controller, Data-ASIC & Voice-ASIC) and their interactions are tested along with data bus, address bus and control lines. D+V TDM bus is also tested; and

5. Refer to line card (xLC) architecture block diagram – Figure 4.7. The xLC-ASIC retreat data from the D+V TDM bus and store it in the register. The components involve in this execution are:

   a. D+V TDM bus

   b. xLC-ASIC

   c. data bus on xLC module

   d. address bus on xLC module

   e. control lines on xLC module.

   f. xLC Register

   In this cycle, multiple module interactions are tested along with D+V TDM bus, data bus, address bus, control lines & registers.

   In order to check that the data stored in the line card register is intact and no bit position is changed or corrupted, a read operation is required. When the automated test

system gives read command, almost same 1-Meg Modem components/blocks are used mentioned above in the write cycle. Automated test system compares the data, which it retrieves from the read cycle with the data, which it has written in write cycle. If there is any change in the bit pattern, an error is generated.

4.5 IMPLEMENTATION

4.5.1 1-MEG MODEM AUTOMATED HARDWARE-TEST-SYSTEM ARCHITECTURE

The detailed architecture and components of automated test system are explained in the previous section 3.4. Here we only highlight those components of automated test system that are used in the SUT hardware testing. As shown in the Figure 4.9, the PC-based automated test system is connected with SUT's diagnostic port through RS232 (serial) port. The main processes of automated test system which are involved in the hardware testing of SUT are Visual Basic based main application, serial port communication application (MSCOMM control) and the serial port drivers. The MSComm control provides serial communications for the main application by allowing the transmission and reception of data through a serial port. The main application pools for events and errors by checking the CommEvent property after sending commands to SUT. The communication drivers (RS232) are at the lowest layer. MSComm interface with the RS232 driver and send & receive data through it.

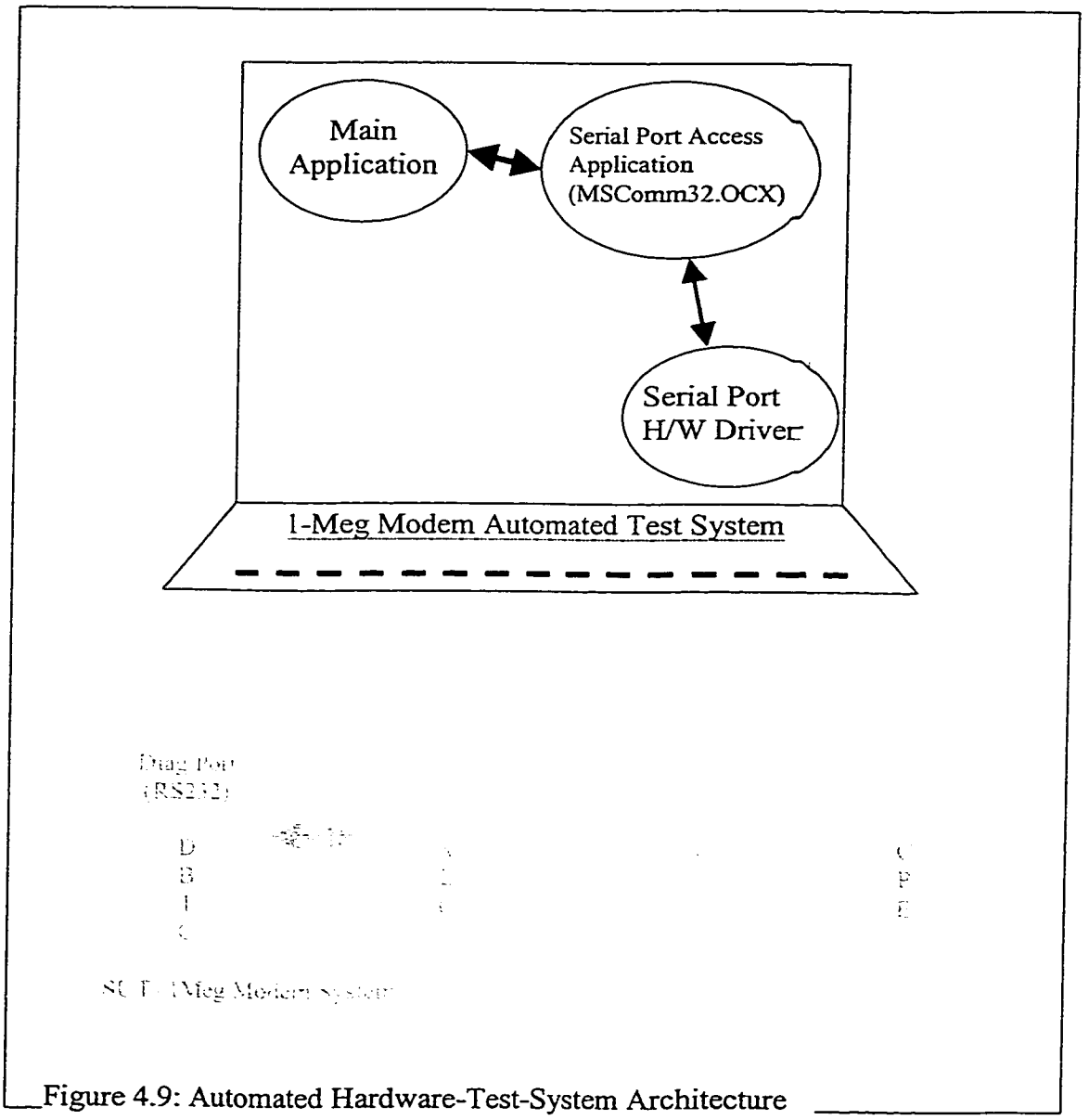

Figure 4.9: Automated Hardware-Test-System Architecture

MSComm control has many properties but few are the primary properties that are responsible for establishing the rudimentary serial communication. Following are the primary properties of MSComm and their usage are shown in Appendix B, Figure B.7.

Properties	Description
CommPort	Sets and returns the communications port number.
Settings	Sets and returns the baud rate, parity, data bits, and stop bits as a string.
PortOpen	Sets and returns the state of a communications port. Also opens and closes a port.
Input	Returns and removes characters from the receive buffer.
Output	Writes a string of characters to the transmit buffer.

RS232 driver uses full duplex mode and asynchronous communication to send and receive data to/from RS232 physical cable. The configuration of RS232 port and cable configuration are shown in the Figure 4.10 & Figure 4.11 respectively:

Port Settings	
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Figure 4.10: Port Configuration

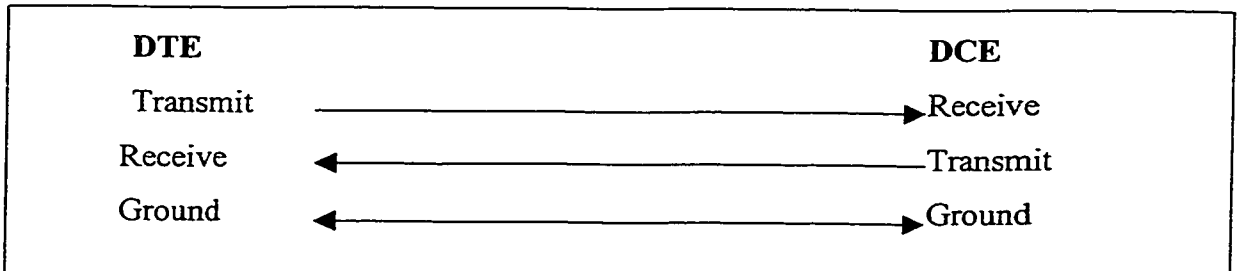

Figure 4.11: Cable configuration

The RS232 interface is extremely slow interface compared to the internal bus of 1-Meg Modem (SUT). The commands are sent at 9600 bps through RS232 interface whereas these commands were executed in microsecond by SUT. To overcome the RS232 speed problem, the designer provided the macro commands for most of the test cases.

4.5.2 COMMAND FORMAT

The command format is based on a client-server architecture. The SUT acts as the server, and waits for request from the client (automated test system). The SUT never initiates communication, and only send messages in response to requests received from the automated test system.

Commands are composed of a set of ASCII characters. The formats of commands are following:

- Read operation on individual register
  opcode + module no + address
- Write operation on individual register
  opcode + module no + address + data
- Macro commands
  command word (e.g. "adapt")

The advantage of ASCII command format is:

1. *ASCII commands are portable.* ASCII format is a standard representation of characters. The ASCII character values represent the same information across the different implementation platform.
2. *ASCII commands are readable/interpretable.* Commands can be understood/interpreted directly by the user, without requiring translation from special format.

Using ASCII characters for the content of command request/response is not an efficient representation for volume data. However, it is an acceptable method for the commands request/response between automated test system & SUT since the commands request/response are short and do not carry large amount of data.

4.5.3 USER INTERFACE, OPERATION FLOW & TESTING ALGORITHM

The prototypes of our system's graphical user interface (GUI), using Visual Basic, are shown in Figure 4.12, 4.13 & 4.17. As mentioned earlier that Visual Basic, a visual programming language, using OO techniques & Active X components, is ideal to develop GUI. Forms and controls (Active X components) are the basic elements/objects of GUI in any Windows application. These objects have properties and methods which are used in developing GUI interface and reacting to external events. The user interface are those objects with which user interact. There are some objects that do not interact directly with user so they do not appear on GUI during application running. For example, MSComm object that is responsible for serial communication does not appear in the above mentioned Figures.

The automated test system executes three tests to check registers / buffers stuck bits, the operational functionality and packet/bytes counters of the SUT. Following are these test cases:

4.5.3A Test Case-1 (Stuck Bit Test)

As the name implies, it explicitly tests the stuck bits, coupling and pattern sensitivity of the registers / buffers by using different set of bit patterns. Interestingly, as indicated in the section 4.4, while testing the registers / buffers of the line card module, the test case1 implicitly test many other components of the SUT. Following are the list of components that are tested explicitly and implicitly in the test case-1.

a) registers / buffers of line card module (xLC)

b) μ-controller, Data-ASIC, VOICE-ASIC of the main module (DBIC)

c) xLC-ASIC of xLCs

d) inter processing among μ-controllers on the DBIC and between μ-controller of DBIC & μ-controller of xLC

e) memory in DBIC/xLC

f) UART of DBIC

g) internal / external data/address/control bus of DBIC / xLC

h) address decoder of DBIC

i) Data+Voice TDM bus on DBIC / xLC

As can be deduced from the above component list that test case1 almost cover all major components of SUT. However, since test case1 is only explicitly testing the registers/buffers of xLC so it can only detect & locate the fault of registers/buffers. The fault on any other component (choice b to i) can only be detected. However, it can be located on these components at the module level by simply replacing modules.

The test patterns that are used in this test case1 are presumably detect all faults related to stuck bit test, coupling, & pattern sensitivity. We presume that all the

definitions & corollaries of classical theory pertaining to stuck bit tests (various bit patterns) are fulfilled in the tested hardware. The test patterns are:

a) *test pattern-0* (00 0000) : This test pattern is used to check explicitly any bit(s) of registers/buffers s-a-1.

b) *test pattern-3F* (11 1111): This test pattern checks explicitly any bit(s) of registers / buffers s-a-0.

c) *test pattern-2A* (10 1010): This test pattern is used to check explicitly every alternate bit(s) s-a-1 & s-a-0 respectively.

d) *test pattern-15* (01 0101): This pattern checks explicitly every alternate bit(s) s-a-0 & s-a-1 respectively.

e) *marching test*: This test algorithm is popular because of its simplicity. One type of marching test is the *modified algorithmic test sequence* (MATS+) which is considered here. This test scans all memory in ascending then descending order. Each scan consists of reading the registers/buffers cell for a preloaded expected value, writing the complemented value, and then reading it a second time.

The idea is that, while scanning the registers/buffers in the ascending order, if there is a problem with coupling, the write operation may affect a higher address registers/buffers location. This would be detected during the ascending portion of the test. While scanning in the descending order, any write that might affect a lower address registers/buffers would be detected.

This test detects all stuck-at-faults and address decode faults. However it does not detect a coupling fault with the data lines.

f) *galloping test*: The galloping ones and zeros test is quite often used in industry. This test verifies that all registers/buffers cells exist, that there are no stuck at bits in the data buffer, and that no coupling exists between registers/buffers cells.

The major disadvantage of this test is the long completion time. This makes it very impractical for use with large number of registers/buffers.

This test verifies that all cells are addressable and that there is no coupling between the cells. It also guarantees that there are no stuck-at-bits in the decoder, data buffer, or registers/buffers.

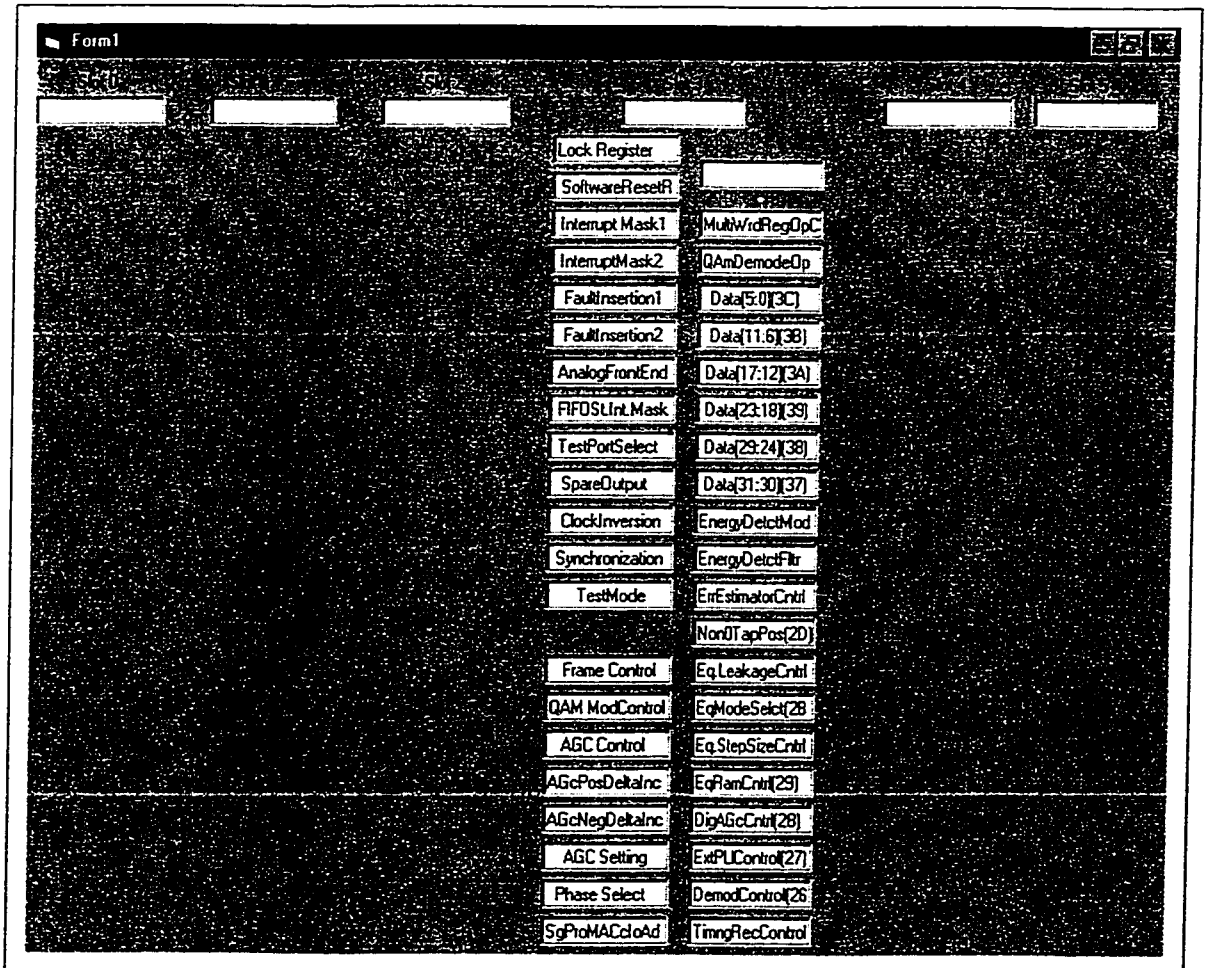

Figure 4.12: test case-1 GUI

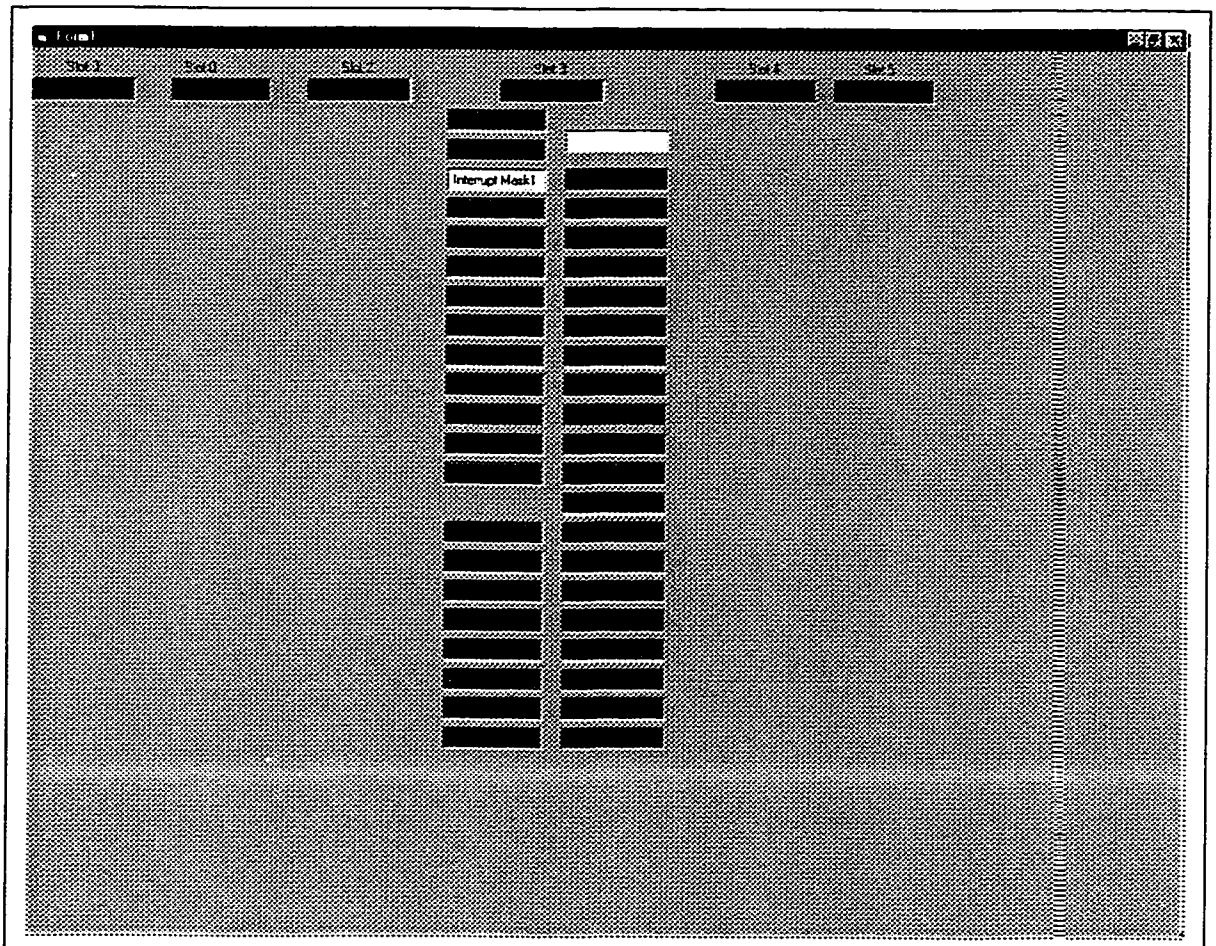

Figure 4.13: test case-1 GUI result

(i) Graphical User Interface:

As mentioned earlier, in test case 1, stuck bits (s-a-0, s-a-1) on different slot modules are tested and its GUI is very simple - see Figure 4.12. The GUI has o ne form and on which arrays of labels and textbox objects are placed. The labels are marked as Slot 0, Slot 1, Slot 2, Slot 3, Slot 4, & Slot 5 by using the caption property of label objects. Similarly, the textbox are assigned the name of different registers/buffers by using the Text property of textbox objects. The result of the test case 1 is shown in the

Figure 4.13. It is the same GUI of Figure 4.12 except the color properties of textboxes are used. The red color of the textboxes which are marked as Slot-#, indicates that the module is either not present or fails to detect. The green color indicates that the module is detected in that slot #. The registers of the detected slot module indicate that the test case 1 (stuck bit test) passes if green and fails if red.

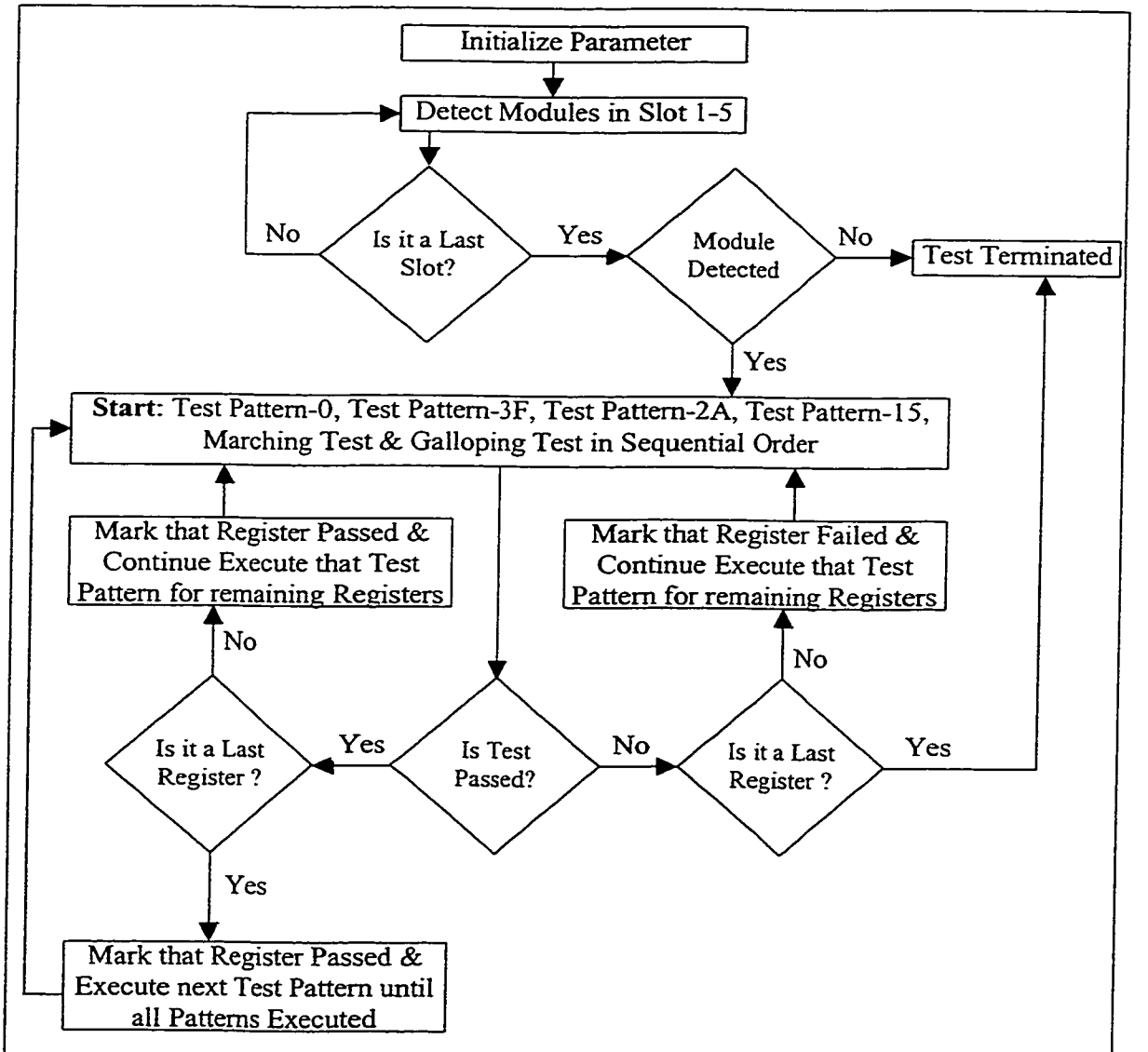

Figure 4.14: Flow Chart of Test Case -1

(ii) Operation Flow:

As shown in the Figure 4.14, flow chart of test case 1, the application:

1. initializes parameters;
2. detects modules in the different slot of SUT whose storing elements, register/buffer, are tested by the stuck bit test method;
3. marks the slots green where the modules are present and mark them red if modules are absent;
4. skip the register/buffer if it is read only;
5. execute test pattern-0 (00 0000) algorithm, shown in Appendix B, Figure B.1;
6. *if test pattern-0 is failed*, & it is not a last registered

   stop the test, mark the register red (failed), and

   go to step-4 to execute the test patterns on the next register

   else

   mark the register red (failed), and terminate the test

   if test pattern-0 is passed,

   execute the test pattern-3F (11 1111) algorithm, shown in Appendix B, Figure B.2;
7. *if test pattern-3F is failed*, & it is not a last register

   stop the test, mark the register red (failed), and

   go to step-4 to execute the test patterns on the next register

   else

   mark the register red (failed), and terminate the test

if test pattern-3F is passed,

execute the test pattern pattern-2A (10 1010) algorithm, shown in Appendix B, Figure B.3;

8. *if test pattern-2A is failed, & if it is not a last register*

stop the test, mark the register red (failed), and

go to step-4 to execute the test patterns on the next register

else

mark the register red (failed), and terminate the test

if test pattern-2A is passed,

execute the test pattern pattern-15 (01 0101) algorithm, shown in Appendix B, Figure B.4;

9. *if test pattern-15 is failed, & it is not a last register*

stop the test, mark the register red (failed), and

go to step-4 to execute the test patterns on the next register

else

mark the register red (failed), and terminate the test

if test pattern-15 is passed,

execute the test pattern marching-test algorithm, shown in Appendix B, Figure B.5;

10. *if test pattern-Marching-1-test is failed, & it is not a last register*

stop the test, mark the register red (failed), and

go to step-4 to execute the test patterns on the next register

else

mark the register red (failed), and terminate the test

if test pattern-Marching is passed,

execute the test pattern galloping test, shown in Appendix B, Figure B.6;

11. *if test pattern-galloping test is failed, & it is not a last register,*

stop the test, mark the register red (failed), and

go to step-4 to execute the test patterns on the next register

else

mark the register red (failed), and terminate the test

if test pattern-galloping test is passed, & it is not a last register,

mark the register green (passed), and go to step-4 to execute the test patterns on the next register

else

mark the register green (passed), and terminate the test

4.5.3B Test Case 2 – Systems Synchronization & In-service Data Path Integrity Test

This test case is used to test:

1. Systems' Synchronization: the hardware blocks and their integration by bringing the entire system up using the macro commands of assembly language via serial port
2. In-service data path integrity test:

1. Systems' Synchronization:

The automated system run the macro commands to test the different blocks of 1-Meg modem and their integration by invoking the functionality of these blocks in a

particular order & with particular delay. The result of this test appears in the form of 1-Meg modem system up and operational. However, this operational 1-Meg modem system would be a basic system that has no intelligence as it does not react with any changes in the environment such as rate adaptation with changes in loop condition. The other ADSL feature like Forward Error Correction (FEC), Off/On-Hook detect, etc, are also not invoked in the basic 1-Meg modem system. The basic system acts as a data pipe which takes data from one end and pass it to other end and vice versa.

The section 4.4 explained about the DBIC & xLC blocks and their interaction that how DBIC communicate with xLC and how it access xLCs' registers/buffers. Test case-1 tested these components explicitly & implicitly. This test case test the function of entire 1-Meg Modem system including CPE and the Tx/Rx Block of xLC which are not covered in test case-1. The analogue portion of xLC and CPE deals with ADSL frequency spectrum using QAM modulation. As mentioned in the beginning of this chapter that 1-Meg-Modem automated test system performs the functional testing. It does not test individually QAM modulation (constellation, power, gain, etc.) or ADSL frequency spectrum, etc. Emphasizing the points that these analogue components are the main core of ADSL technology and since the 1-Meg-Modem is an ADSL system so confusion may develop that automated system tests the analogue components explicitly. All blocks of 1-Meg Modem system are implicitly tested in this test case. The errors can be observed when the system is not up. The approximate location of errors can be deduced from the debug port messages. The 1-Meg-Modem system in response to macro command from the automated system replies the state of events that it processes on the debug (serial) port. The automated system captures the events in another form shown in Figure 4.15.

These debug messages give a good idea of problematic area and since the SUT is a modular system, so the fault can be located immediately by replacing that problematic area module.

Following are the list of blocks/components that are verified in this test case.

a) registers/buffers of xLC & CPE

b) μ-controller, Data-ASIC, VOICE-ASIC of the main module (DBIC)

c) xLC-ASIC of xLCs

d) CPE-ASIC of CPE

e) inter processing among μ-controllers on the DBIC, between μ-controller of DBIC & μ-controller of xLC and between μ-controller of xLC & μ-controller of CPE

f) memory in DBIC/xLC/CPE

g) UART of DBIC

h) internal/external data/address/control bus of DBIC/xLC/CPE

i) address decoder of DBIC

j) Data+Voice TDM bus on DBIC/xLC

k) Digital/Analogue Loop controller circuitry (DAC, ADC, QAM constellation, transmitter/receiver, etc) of xLC & CPE

l) Power, line, & data status display (LEDs & its circuitry)

```
adapt
Hit any key to stop...

      00 - - -04 - - -08 - - -0c - - -10 - - -14 - - -18 - - -1c - - -
xlbus:      s
ds:         4
            w

us:         4
            3


      00 - - -04 - - -08 - - -0c - - -10 - - -14 - - -18 - - -1c - - -
       . . . - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
xlif 3
Initializing XLBUS
Initializing MIX
Initializing rx
Wait <= 6 sec for energy
Energy detected
123
Level=3
SetRateNoAck
Waiting for AGC to stabalize...done
Waiting for timing...done
SetRate...Ack
errpwr=6 <= keep=24

Level=2   errpwr=6 <= drop=10
SetRateAtLevel 2...Ack
Sync Attempt 1
Waiting for AGC to stabalize...done
Waiting for timing...done
errpwr=29 <= keep=24

Level=3
SetRateAtLevel 3...Ack
Sync Attempt 1
Waiting for AGC to stabalize...done
Waiting for timing...done
errpwr=9 > keep=24
  US QAM:4 in sync
  DS QAM:4 40kHz in sync
```

Continue on the next Page.....

```
SetRate ds 64 160kHz hi
CPE acked + tx set
Wait 10s for CPE
  US QAM:4 in sync
  DS QAM:64 160kHz in sync

SetRate us 64
Waiting for AGC to stabalize...done
Waiting for timing...done
CPE acked + rx set
  US QAM:64 in sync
  DS QAM:64 160kHz in sync
s
     00 - - -04 - - -08 - - -0c - - -10 - - -14 - - -18 - - -1c - - -
xlbus:      s
ds:        64
            w
             s
us:        64
            3
            s

      00 - - -04 - - -08 - - -0c - - -10 - - -14 - - -18 - - -1c - - -
      . . . s - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 4.15: Debug Port message during system up

2. <u>In-service data path integrity test</u>:

Once the basic system is up and ready to carry data from user to network & vice versa, the packet counters help to test the entire data path. The packet counters are located in different modules at different position in the entire data path to count good & bad frames of data packets. The automated system monitors these data packet at the entire data path of 1MM system through the packet counters and determines the quality of data path.

The data path comprises of different level of frames/protocols. Again, due to security reason, the details of these frames/protocols, the name of some frame fields and their hierarchical layers are not given. However, transformation of these frames at different hardware modules shown in Figure 4.16 is sufficient to understand the data path integrity test. Figure 4.16 not only shows the position of packet counters in the data path at different modules but also explains pictorially the functions of these packet counters in that data path. It is important to note that the data format is same in the upstream and downstream directions even though the data rates are asymmetrical. Details of frames/protocols transformation captured in Figure 4.16 are explained below for downstream:

1. the system (Dbic module) receives the Ethernet packet and converts it into Protocol-1 frames to send it over to xLC module via D+V TDM bus;
2. The xLC modules receive the Protocol-1 frames and transform it into Protocol-2 frames which is heavily based on HDLC;
3. xLC module further transforms the Protocol-2 frame into Protocol-3 frames which is responsible for modem (CPE) synchronization;
4. Finally xLC transform the Protocol-3 frame into Protocol-4 which encompasses details relating to the QAM modulation used to carry data over the copper loop;
5. The CPE (modem) on the other side of the loop receive Protocol-4 and extract the Protocol-3 frame;
6. The CPE then extract the Protocol-2 frames from Protocol-3 frames; and
7. The CPE then further process the Protocol-2 frame and extract the Ethernet frame from it and pass it over to user interface.

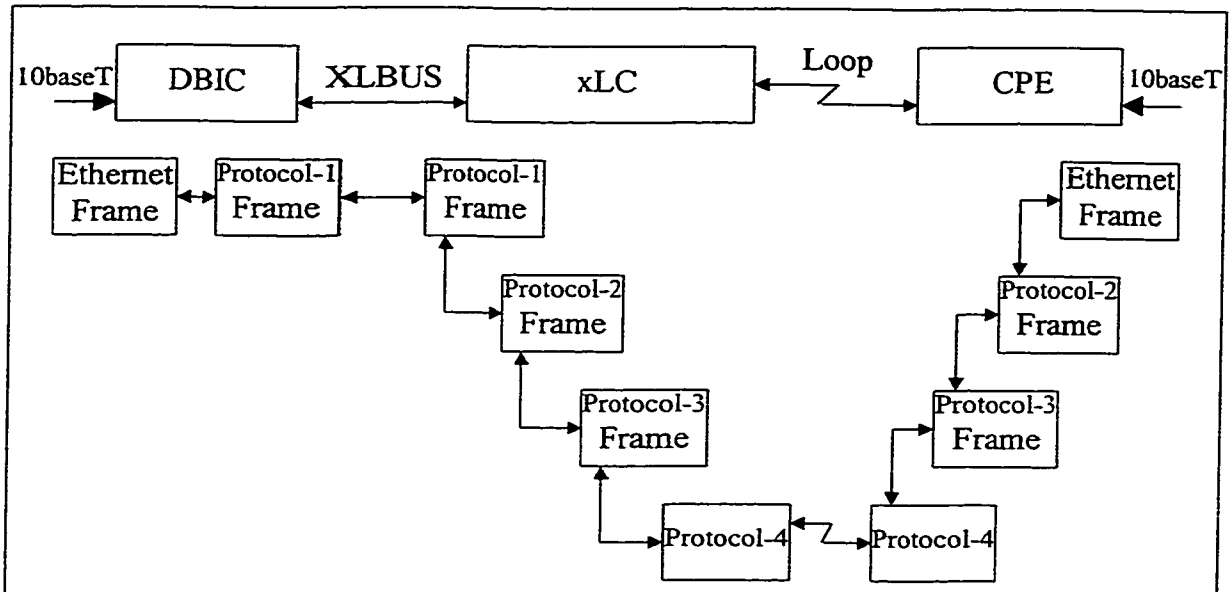

Figure 4.16: End to End Data Path Frames

Since the data format is same for upstream and down stream directions so the transformation of frames/protocols in the upstream direction is merely a reverse order of the downstream transformation i.e. step 6 to step 1.

The hardware components that are explicitly involved in this test case are data packet counters. It is important to note that if any bad (CRC error) frame is received in the entire data path, the system will drop that frame and increase its corresponding layer packet counter. However, if the data is transmitted faster than the capacity of SUT, the frames will be ultimately dropped and captured by edge counters, dbicEthernet drop counter, of data path. In this test case we presumed that the data would be transferred within the capacity of SUT. The list of packet counters is given below with their function and their usage in locating the data path fault. In the list, 'a' to 'e' are the packet counters capturing the downstream traffic and 'f' to 'i' are the packet counters capturing the upstream traffic. The list is in the serial order in which data comes and leaves.

a) dsXLBUS Err Counter at xLC-ASIC: increase if the received downstream Protocol-1 frame at xLC is corrupted. It indicates that the interface circuitry of D+V TDM bus at DBIC & xLC and D+V TDM bus are suspected faulty data path area;

b) dsXDLC TX Frame Counter: increase when receiving the good downstream Protocol-2 frame at xLC. It indicates the data path from the Ethernet interface of DBIC to xLC (not included the Tx/Rx block) is intact;

c) dsXDLC Rx Good Byte Counter: increase when receiving the good Protocol-2 frame on the loop at CPE. It shows that the data path between xLC and the entire CPE excluding Ethernet port of CPE is through;

d) dsXLINIK CRC Err Counter: increase when receiving the bad Protocol-3 frame on the loop at CPE. The most possible reason is the noisy external loop lines. From the component of view, the Tx/Rx block of xLC and CPE are questionable areas;

e) dsCPE XDLC Err counter: increase when receiving the bad Protocol-2 frame at CPE in the downstream direction. It indicates that the suspected area is entire CPE excluding Tx/Rx block and Ethernet interface;

f) usXDLC Rx Good Byte Counter: increases when receive the good Protocol-2 frame at xLC in the upstream direction. It indicates that the data path from CPE Ethernet interface circuitry to xLC is faultless;

g) usXLINK CRC Err Counter: increases when receiving the corrupted frame on the loop. The suspected areas would be upstream Tx/Rx block of CPE and

xLC circuitry and the loop itself. Among the above-mentioned areas, the most suspected area is the noisy loop;

h) usXDLC Bad Frame Counter: increases when receive the xLC's XDLC bad frame in the upstream direction. The suspected areas would be the entire xLC XLINK circuitry excluding D+V TDM Bus I/F circuitry; and

i) usXLBUS Err Counter: increases when receive the faulty XLBUX frame at DBIC in the upstream direction. The suspected area would be xLC-ASIC circuitry, xLC's D+V TDM Bus I/F circuitry, D+V TDM bus and DBIC's Data ASIC circuitry.

(i) Graphical User Interface:

Figure 4.17 shows the GUI of test case 2. The GUI has one form, which contains a CPE picture, option buttons with their color properties, arrays of labels with different fonts, arrays of text box, Command Button, Clocks, and a MSComm object. The CPE picture and options buttons indicate the result of system-up-test and arrays of labels & text boxes indicate the result of data path test.

The CPE picture object is added through the Picture Control. The CPE picture which is added to the form is a bit map (*.bmp) file. The option button's color property imitates the Power, Sync & Speed light of CPE. The option button's "Power" red light indicates that the SUT Power is ON; "Sync" green lights indicates that the system is up otherwise it will be red (system is down); and the "Speed" green light indicates the line speed in kbps.

Labels are marked with "Down Stream" & "Up Stream" heading with higher font and counter names with lower font along with the text boxes. The text boxes are used to display the integer value of good/bad frame count.

The command button is an "EXIT" command button that terminates the existing application. The clocks are used for multitasking and can not be seen during application running. Similarly MSComm object is also a hidden object and its function is similar to the one that is explained in test case1.

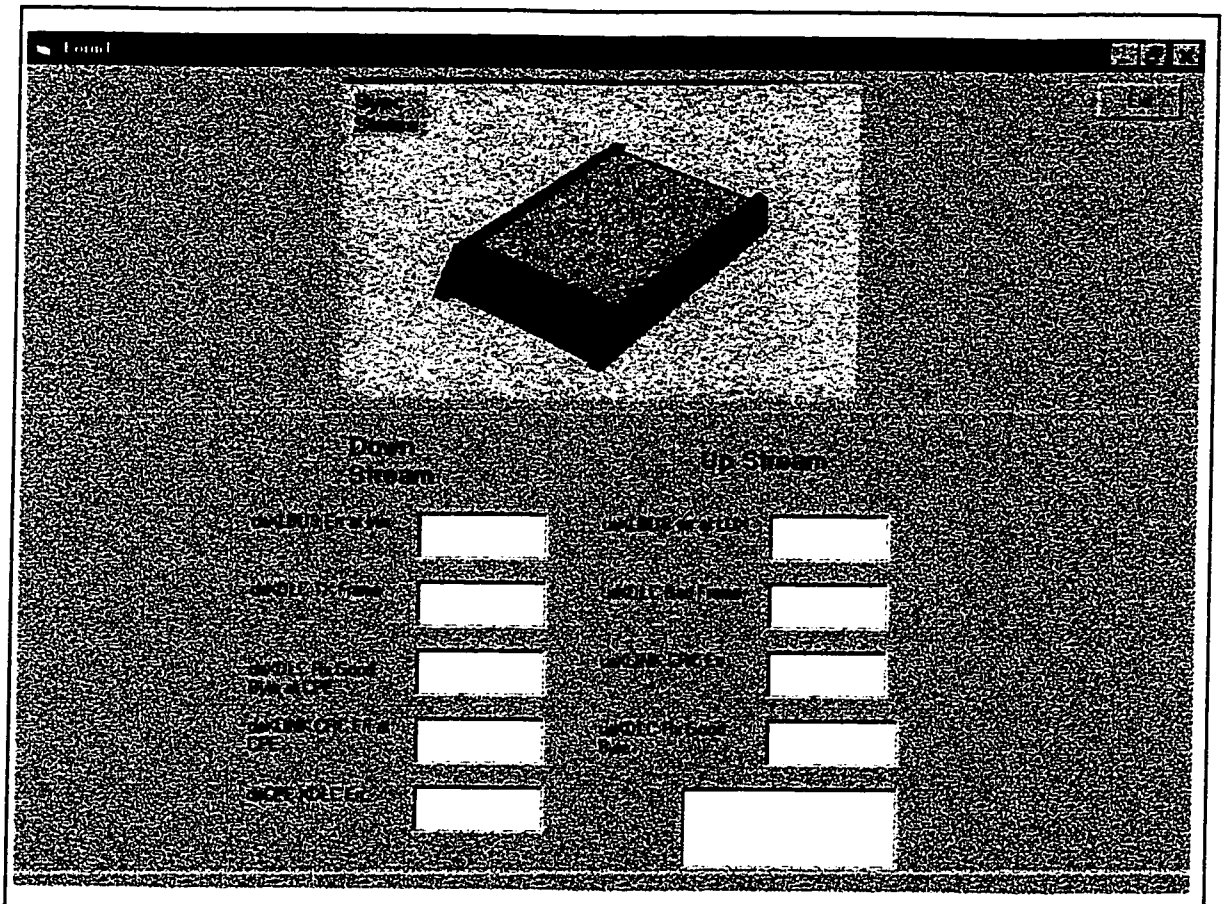

Figure 4.17: test case-2 GUI

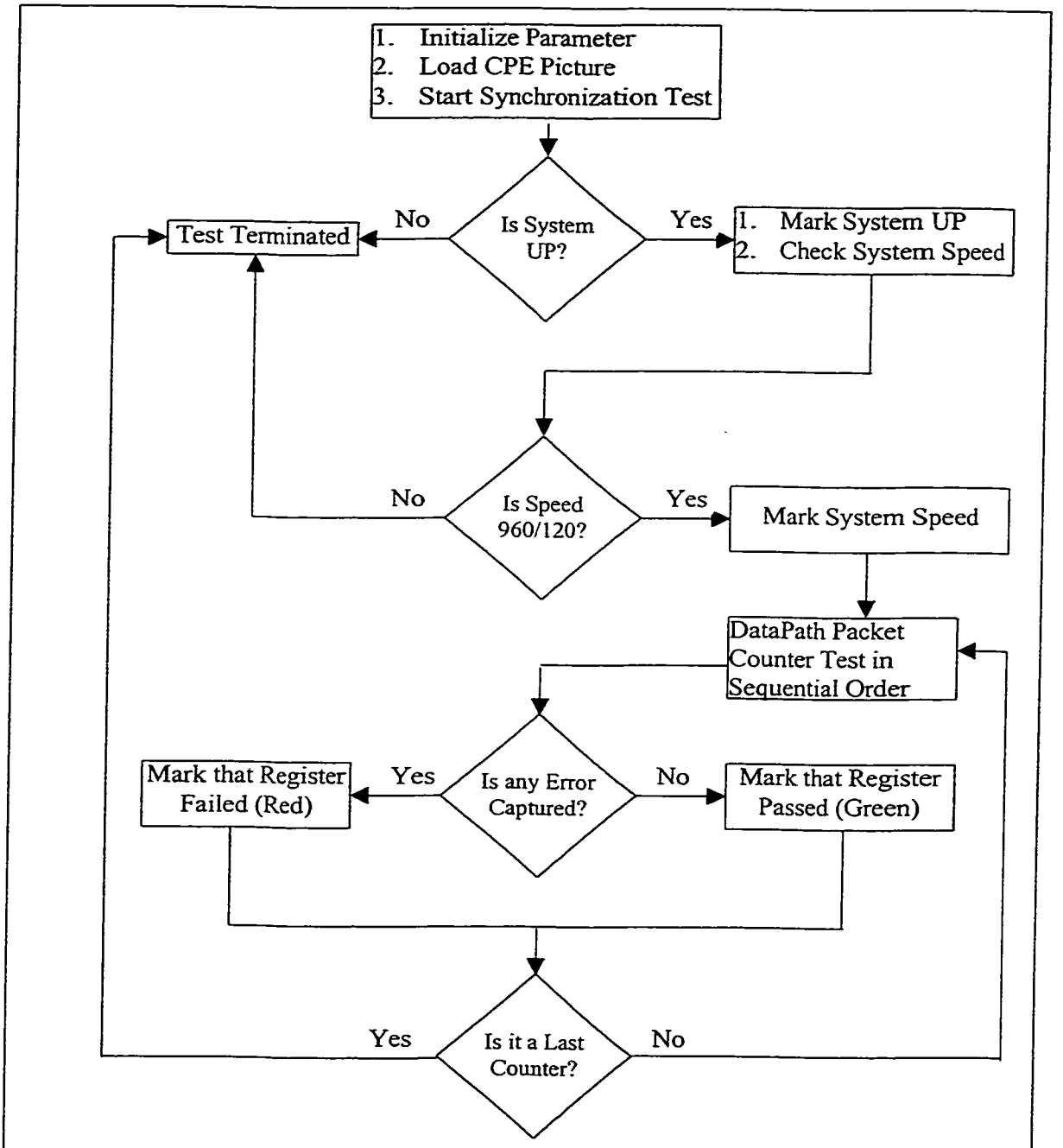

Figure 4.18: Flow Chart of Test Case -2

(ii) Operation Flow:

The flow chart of test case-2 is shown in Figure 4.18. The automated test case 2 application:

1. initializes parameters;
2. loads CPE picture from local hard drive;
3. sends macro command to bring up the system;
4. reads messages from SUT. Parsing the messages to determine the state of the system i.e. Up or Down;
5. if the system is Up, it

    makes the CPE sync light green

    checks the max speed. If max speed is 960/120 kbps

        makes the CPE speed light green

        starts the data path test

    else

        makes the CPE speed light red

        terminates the test

  else

    makes the CPE sync light red

    terminates the test

6. data path test : start generating traffic through data generator or via ping;
7. start capturing those data packet by the packet counters; and
8. if any error is captured by the error packet counter, an error is generated by changing that counter text box to red.

CHAPTER 5
SOFTWARE TESTING

5.1 INTRODUCTION

Software testing only came to be treated as a serious research topic in the late sixties when the 'software crisis' spawned the concept of software engineering. To clarify matters from the start, it is appropriate to consider the various definitions that have been given to software testing. Adrion, *et al.* [Adr82] define it as:

> *Examination of the behavior of a program by executing the program on sample data sets.*

while Glass [Gla79] defines it as:

> *...the process of executing computer software in order to determine whether the results it produces are correct.*

Myers [Mye79] is of the opinion that testing adds some 'value metric' to the program – the discovery and removal of errors increase the reliability and quality of the program. His definition of testing is:

> *...the process of executing a program with the intent of finding errors.*

This almost aggressive approach to the subject can be seen as a response to Dijkstra's, (Dah72] comment some years earlier that,

> *Program testing can be used to show the presence of bugs, but never their absence.*

This, however, is not true. As will be seen later, there are testing methods that are able to show certain—but not *all*—*'bugs'* (or faults) to be absent. A contrast to Myers' approach

is made by Hennell et al. [Hen84] who argue that:

> *the aim . . . is not to discover errors but to provide convincing evidence that there are none*

or,

> *...to show that particular classes [of fault] are not present.*

Hetzel devotes a large section of *The Complete Guide to Software Test* [Het85] to the definition of testing and concludes:

> *Testing is any activity aimed at evaluating an attribute or capability of a program or system. Testing is the measurement of software quality.*

Testing plays a very distinct role and its purpose is to detect bugs. Testing is more usually a basic set of checks which determines the system's bug (failure). Historically there have been different classifications of testing techniques.

Static techniques were those that examined the software without executing it and encompassed activities such as inspection, symbolic execution, and verification. Whereas, Dynamic techniques examined the software with a view to generate test data for execution by the software.

Another set of technique used is the *black-box/white-box* dichotomy. Test cases that were derived without reference to the construction of the program (i.e. they were created by reference to the specification, or some other description of what the software should do) were termed 'black-box' techniques. That is, the software was treated as a black box and its functionality was determined by supplying it with different combinations of inputs. In contrast to this, test cases that were derived by examination of the construction of the program (its 'internal workings') were then termed 'white-box'. Other terms have been introduced over the years and now black-box techniques are

sometimes called 'functional' or 'specification-based' and white-box techniques are referred as 'structural' or 'code-based' or even 'glass-box'. Most of the software projects only perform black box testing. Industry papers and books started to document and show by example that test cases designed from code were not reliable failure finders [Good75, Howd76, Howd80, Beiz82, Beiz90, and Patr91]. In this thesis, the software testing of 1-Meg modem (SUT) is performed on the basis of functional (specification/black box) testing, along the lines of hardware testing.

5.2 RELATED WORK

Automated software testing is an ideal way to maximize yield of testing. Tools that support this feature provide the ability to reapply tests frequently and with little effort when changes are constantly made to a system. Different tools support different features. An automated tool could excel the black-box testing and/or white-box testing. Irrespective of the type of testing applied, "automated testing leads to a higher reliability by allowing more testing" [Bin96].

At the time this thesis was started, ADSL was in its nascent stage. The product like G.Lite, which is similar to 1-Meg Modem was not available in the market. Its standards along with the 1-Meg modem were being defined. None of the tools was available to test the entire ADSL (G.Lite/1-Meg Modem) features. Some of the proprietary telecommunications' automated test systems were available to test the software of their telecom products.

Following are few tools/automated-test-system which are being used for testing software on basis of functional (specification/black box) testing.

(i) Probe-Advance Connect:

IC&C GmbH developed this software product. It automatically generates source code from the design model. It also parses the source code and generates structural design model. It is capable of monitoring the messages sent in a running application and capturing this information. It works with Smalltalk and provides its own design modeling capabilities [IC&97].

(ii) Rational Rose:

Rational Software Corporation developed a product called Rational Rose. It automatically generates code from the design models. It also parses the source code and generates structural design model. It does not monitor the behavior of the implemented application and generates the corresponding behavioral models. [Alg94]

(iii)Together/Professional:

This software product was developed by Object International software. It automatically generates code from the design models. It also parses the source code and generates structural design model. It does not monitor the behavior of the implemented application. [Obj97]

(iv)The Smalltalk Test Mentor:

SilverMark developed this software. It provides automated testing capabilities. The user can record and play back user interface interactions with the application being used. The user can also write scripts to automatically execute tests. It keeps track of which application code was executed/tested. It does not monitor/capture message events between objects.

(v) ASTOOT:

ASTOOT (A Set of Tools for Object-Oriented Testing) is a research system developed for testing programs in the Eiffel environment [Doo94]. It automates class testing, which is needed because manual inspection of the state of complex objects can be slow and difficult. ASTOOT tries to automatically determine if a sequence of messages applied to object leaves that object in a correct state. Methods can be applied in many different sequences, but whatever the order, the object under consideration must always be left in a correct state. The ASTOOT approach adapts ADT (Abstract Data Type) testing strategies to object-oriented software. The ASTOOT system automatically generates paired sequences of method operations, which should yield the same observable results. Both sequences are executed, and the two resulting objects are compared. If the objects are not equivalent, then the test has successfully revealed a fault. In order for this approach to work each class must have an equivalence method developed, one that compares two instances of that class and determines if they are equivalent objects. What drives this tool is a standalone driver that must initialize the object state, run a test sequence, then report the resulting state. ASTOOT demonstrates the ability to automate **black-box testing**.

(vi)Scada in Automated Test and Measurement:

Supervisory Control and Data Acquisition System (SCADA) is a control software and is used in the measurement automation at the laboratory environment, see Figure 5.1. This automated system was developed in 1988 on PC platform. The system supports recipe handling which incorporates the capability to configure the channels. There are also provisions of gain control, range selection, change of polarity in the data acquisition

system (DAS). The measured variables are stored in a database along with time. The system also supports short time archiving showing past performance of the process.

As shown in the Figure 5.1, SCADA system also supports on-line signal processing and database management on the two-dimensional database file consisting of measured variable against time. The system has provisions of display management. The display supports real time database of the measured signal, calculated signal as well as processed signal. The number of windows including the size and position of each window can be adjusted. The sample data points can be highlighted, the line type of the graph and the background color of the window can also be configured. Instead of graphical form the output can also be displayed through virtual instrumentation. A hard copy can also be printed out.

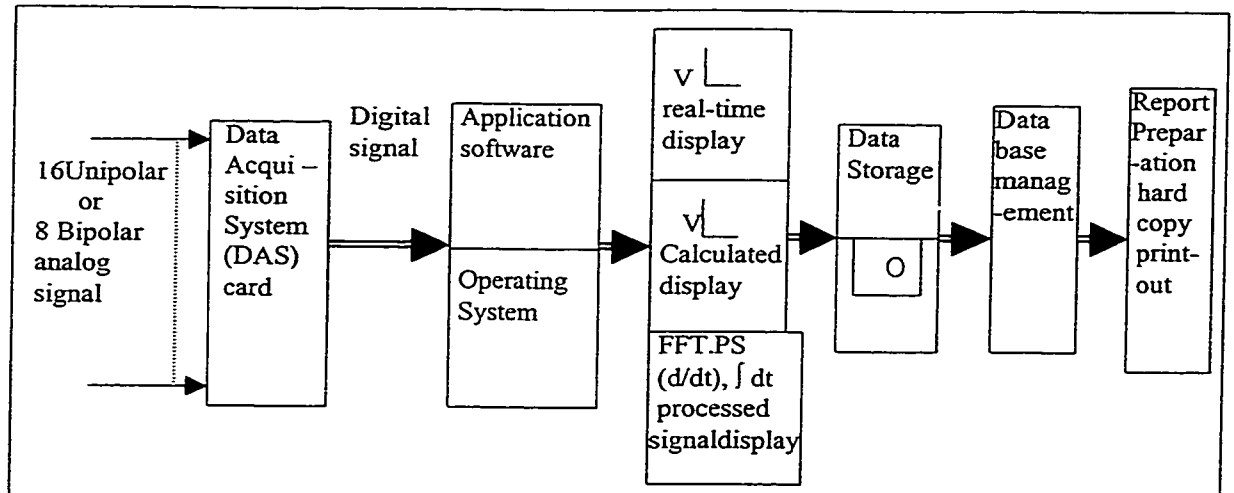

Figure 5.1: Software Capabilities of the Control software used in the Measurement Automation

(vii) Automated Testing for Transport Network Systems (TNS) of Ericsson Telecom:

The automated test system is used to test the software reliability of TNS system. It tests the transport network feature Synchronous Digital Hierarchy (SDH) and Synchronous Digital Cross Connects. The system uses the Basic model and Logarithmic Poisson model for estimating the software reliability. The model parameters are selected by point or interval estimation. Normally both are used simultaneously.

It supports the test analyzer that provides a basis for test management decisions, e.g. resource allocation,. The test analyzer processes the data and produces a fault log, test and reliability report. These reports can be produced in graphical or detailed information form.

(viii) IRWindows:

The IRWindows software provides an accurate automated infrared minimum resolvable temperature difference (MRTD) measurement. It was developed in 1996 on the PC platform. As shown in the Figure 5.2, the automated IRWindows measurement system consists of a unit under test (UUT), an infrared target projector, a data acquisition system, and a PC containing the control, measurement, and analysis software IRWindows. The infrared target projector consists of a relative (two-mirror) collimator, a computer controlled black-body controller and target wheel, a black-body, and the various opaque targets. The data acquisition system consists of a commercial RS-170 compatible digitizer.

IRWindows system is customized to measure key parameters of an infrared system. It estimates the relative performance with the help of a predefined constant described by a trained human observer. It allows the operator to configure the parameters

necessary to run a particular measurement or load parameters from a previous test. It has a graphical type interface, and is easy to control and modify the measurement parameters. Test results are displayed to the operator in a tabular or graphical display, or can either be exported to a text file or dynamically linked to a master control program. The test plan can also be predefined as macros and can run by clicking a button from an operator's menu.

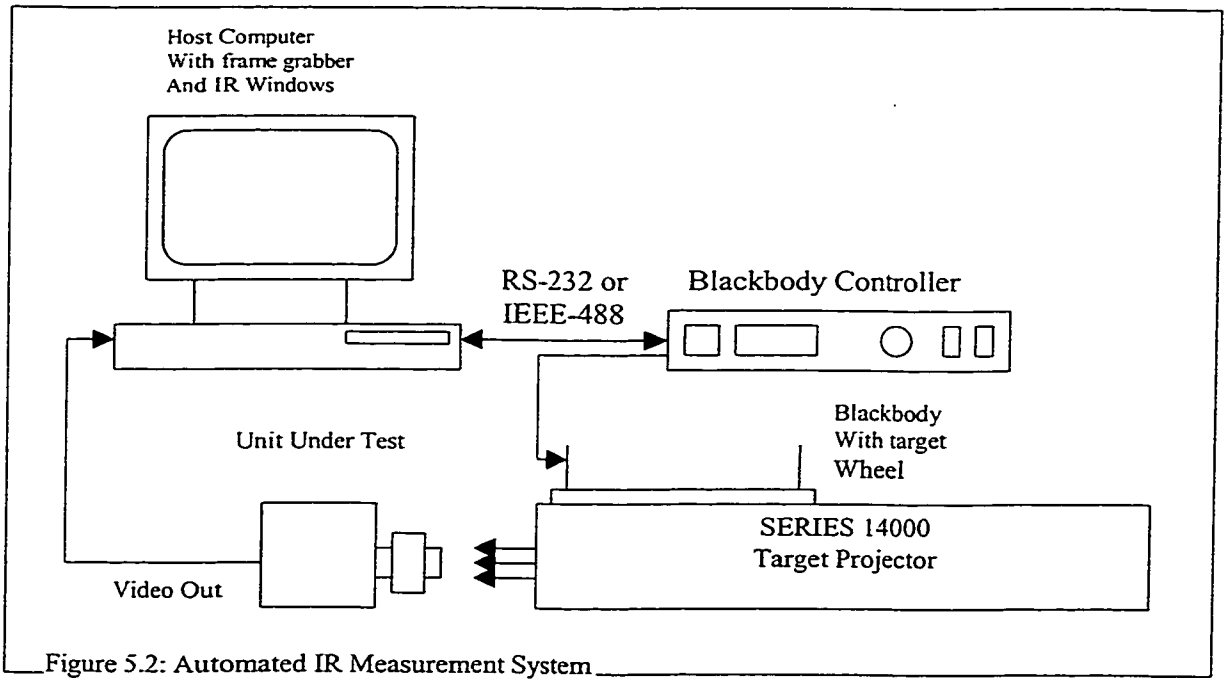

Figure 5.2: Automated IR Measurement System

(ix)Distributed Automated Real-time Test System (DARTS):

DART was developed in 1994 using UNIX box and IBM mainframe for its automated test platform as shown in Figure 5.3. It was developed to support feature testing of a Digital Central Office Switching System, the Siemens EWSD. The Siemens EWSD is a real-time digital central office telephone switch combining software,

firmware and hardware components. The EWSD consists of five major building blocks: Digital Line Units (DLU), Line/trunk Group (LTG), Switching Network (SN), Common Channel Network Controller (CCNC) and Coordinate Processor (CP). The software for the EWSD is written in CHILL (CCITT High Level Language) and is distributed across multiple processors and has millions of line codes.

The DARTS is non-intrusive and runs under SCO Open Desk Top Version 2.0 operating system. The user interface to the DARTS runtime software is provided through an X-terminal. As shown in Figure 5.4, DARTS is designed to be modular. Each functional component of DARTS is responsible for specific tasks. The modular structure facilitates expansion to support additional Commercial Test Devices (CTDs), or extension to the existing CTD driver software. Each CTD driver is isolated from the rest of the system and the interactions between it and the other DARTS software components are clearly defined. Each DARTS component consists of one or more UNIX system V processes communication using files, pipes, streams, message queues, semaphores and shared memory.

Being the automated test system, DARTS supports unattended feature testing, exception handling, the ability to recover and continue testing in "less stable" SUT environments, bypassing of blocking output or events, and handling of resultant data. Users write their "test scripts" in the DARTS Test Language (DTL). Each test submitted to DART as input via DTL. Each test case is broken down into steps, and each step is executed, analyzed, and its results (passed, failed, blocked or analysis) are logged before next step is started. A single automated test case is comprised of one or more test steps.

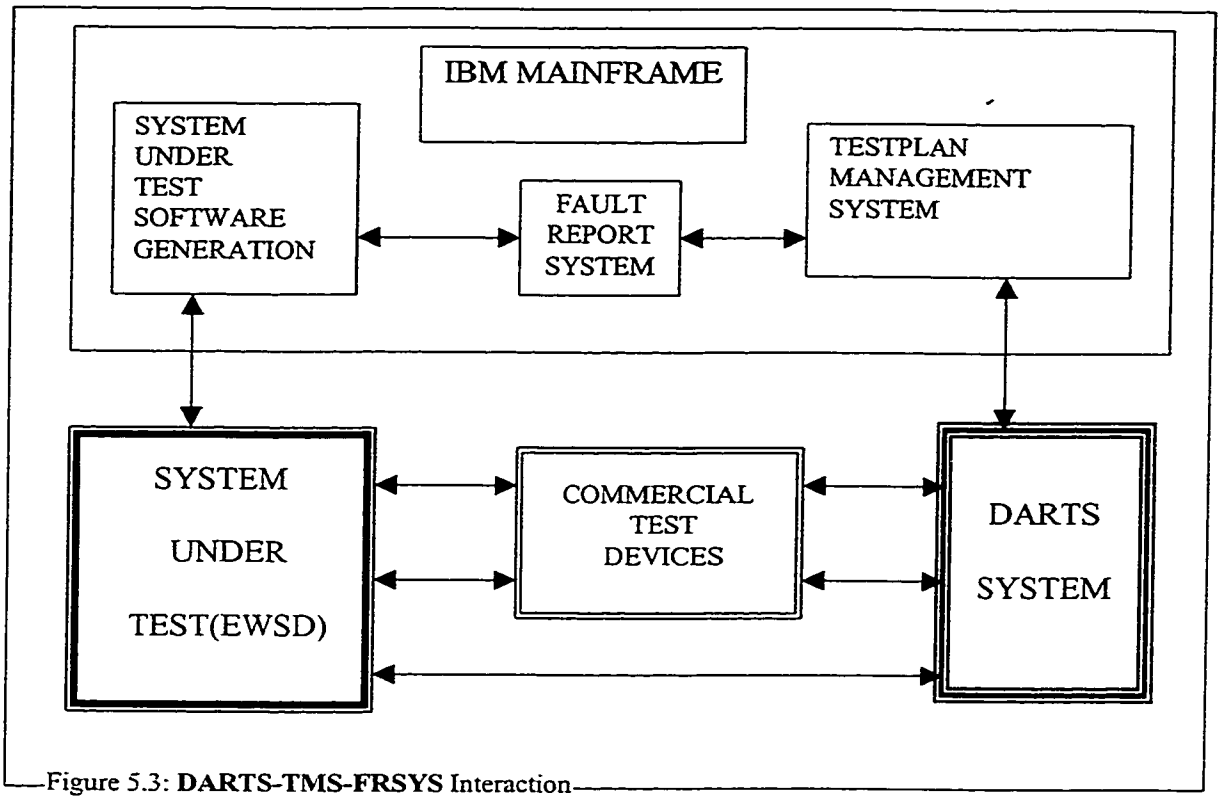

Figure 5.3: **DARTS-TMS-FRSYS** Interaction

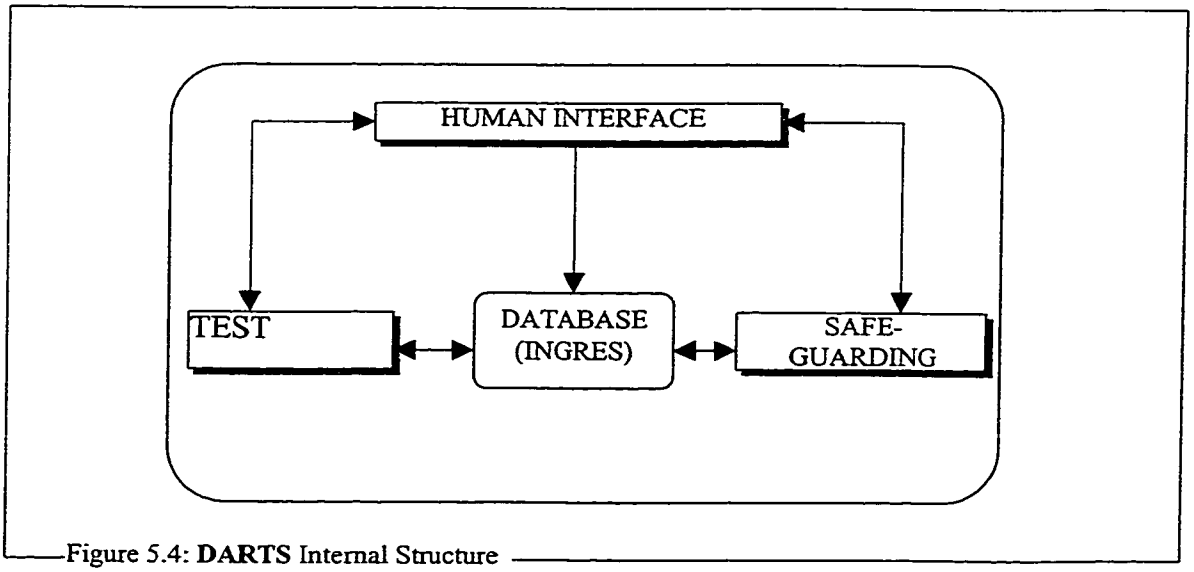

Figure 5.4: **DARTS** Internal Structure

5.3 GENERAL SOFTWARE ARCHITECTURE OF 1MEG MODEM

SUT (1-Meg Modem) is a Nortel Networks proprietary product. Therefore due to the security reason, it is not possible to discuss the software modules in detail. The discussion has been restricted only to the general functionality of these software modules. Since the automated test system is based on functional testing therefore the following brief functionality of the required software modules is enough to understand the software functional testing.

As shown in Figure5.5, the main firmware elements running on the SUT include LoopConfig Process, Line Monitor, Line Interface, SNMP Agent, and Datapath.

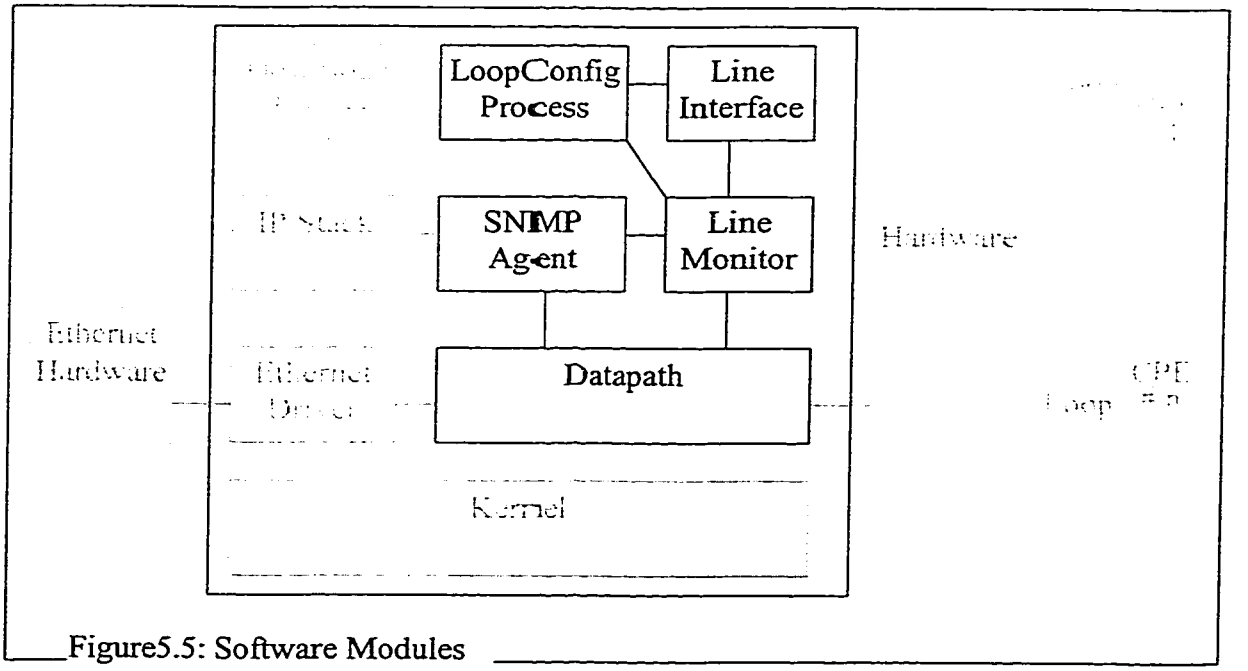

Figure5.5: Software Modules

a) SNMP Agent:

The SNMP agent is a standard agent. It uses the UDP protocol of the IP stack to communicate to the Network Management Station (Automated Test System). It defines the SNMPv1 agent. The brief functionality is summarized below:

- Access to MIB variables is via asynchronous function calls;
- Accessing MIB variables using GET, Get Next, Set, Test;
- Creates queues for Line Monitor messages and
- Create processes for management including Test Process, Trap Processes, and Trap Handler.

b) LoopConfig Process:

The LoopConfig Process handles management of all loops concurrently. However, each loop is processed independently. Its functionality can be divided into following main areas:

- Setup and configure the loops up to a point where data connection is ready to be established;
- Identification if the installed hardware release version of Dbic, xLC and CPE;
- Establishment and maintenance of the connection in the upstream direction and
- Establishment and maintenance of the connection in the downstream direction.

c) Line Monitor:

Its functionality is defined below:

- Channel polling – poll each line card (xLC) and update share memory with the current hardware status;
- Channel setting up – set up xLC sync when line card is detected;

- Line card initializing – initialize line card as long as xLC sync is set up;
- SNMP message processing – process message from SNMP and
- LoopConfig message processing – send a message to LoopConfig to do rate adaptation.

d) Line Interface:

The functionality of line interface is following:

- Hardware access – access Dbic/xLC;
- Mutex access – any process must get a mutex in order to access hardware;
- CPE message processing – send CPE message on behalf of any process and pass the response received from CPE to the requested process and
- API for LoopConfig process to do rate adaptation.

e) Datapath:

The datapath refers to the element which are directly involved in moving and bridging data frames, not including the Ethernet driver. Its purpose is to move Ethernet frames between the Ethernet driver and the hardware. Its functionality is:

- Source Learning – Learn the source MAC address of upstream frames;
- Source Filtering – permits a fixed number of hosts to transmit upstream on a given line;
- Broadcast filtering – Broadcast frame will be filtered except in case of ARPs and Boot/DHCP;
- ARP limiting – total number of ARPs are limited to a given amount per second for upstream and downstream;

- SNMP counters – the counters for SNMP (octets received/transmitted, frames received/transmitted, etc.) must be updated;
- Unicast steering – Downstream unicast frames will only be sent on the line where the destination host is known to be; and
- Connection table aging – Entries will be removed from the connection table if no upstream frames have been received from that for a given time limit.

5.4 IMPLEMENTATION

5.4.1 1-MEG MODEM AUTOMATED SOFTWARE-TEST-SYSTEM ARCHITECTURE:

The architecture of 1-Meg modem automated test system is explained in section 3.4. Here we highlight only those components of automated test system, which participates in the functional testing of 1-Meg Modem software. As shown in the Figure 5.6, the automated test system communicates with SUT for verifying its functionality through 10/100 BaseT Ethernet Port (LAN). The protocol used to communicate between automated test system and the SUT is the standard network management protocol, SNMP. The communication between the automated test system and SUT is through LAN so there is no distance limitation between these systems. For some special purpose, like system's dump, the automated system also communicates with SUT through serial port. In that case both the systems can not be far apart more than 50 feet (serial cable limitation). The loop simulator which is responsible for simulation different loop is also interfaced with the automated system via serial cable. For system performance and

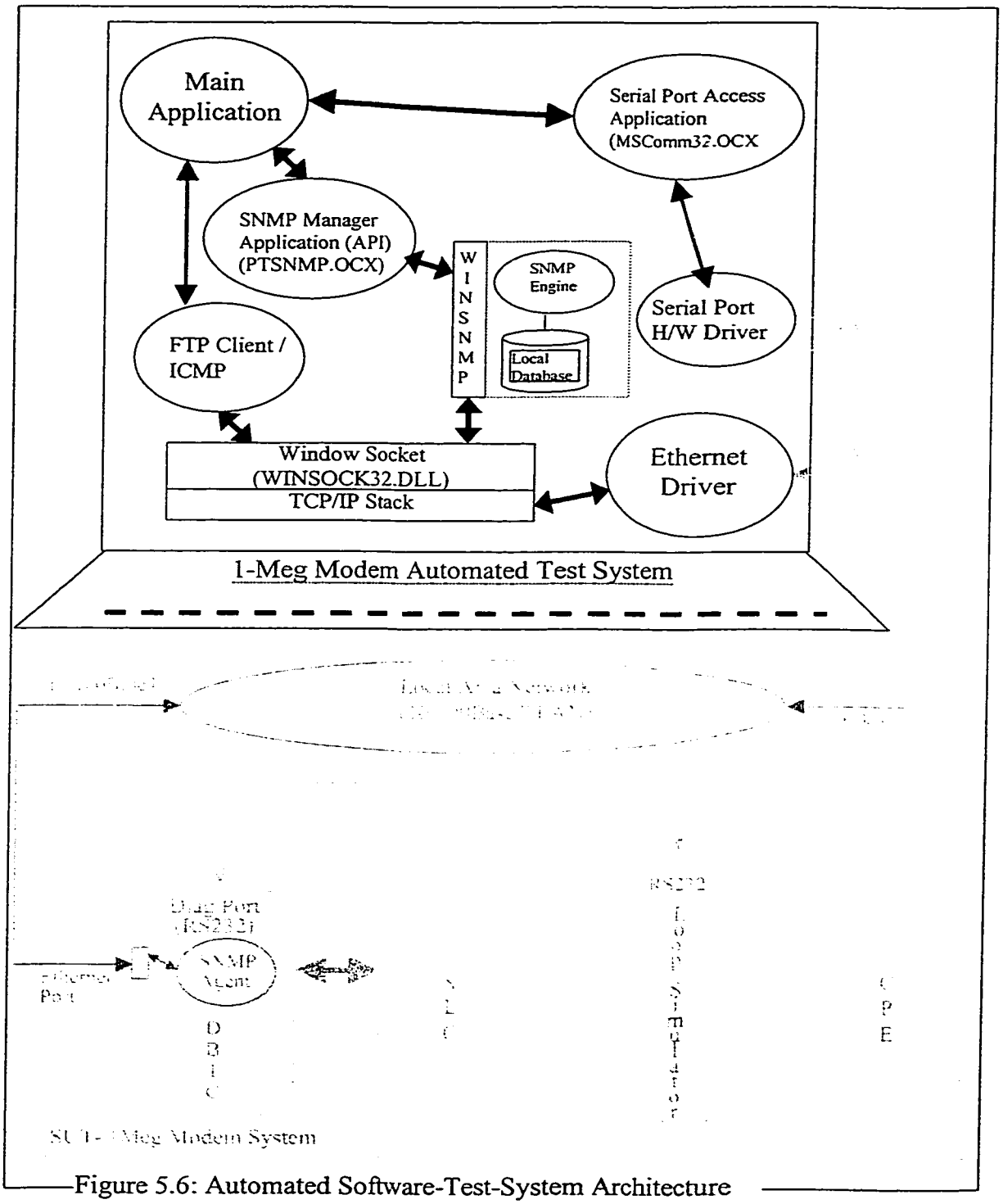

Figure 5.6: Automated Software-Test-System Architecture

Forward error correction (FEC) test, automated test system uses the FTP & ICMP (Ping) applications.

The processes of automated test system, involved in the software testing of SUT are Visual Basic based main application, Manager SNMP-API (PTSNMP control (ptsnmp32.ocx)), WinSNMP, FTP Client (ptftp32.ocx), ICMP/Ping (ICMP.DLL), Winsock, TCP/IP stack, Ethernet Driver, serial port communication application (MSCOMM control) and the serial port drivers.

Manager SNMP-API:

The main VB application accesses (Get/Set) the SUT MIB objects through the SNMP Manager API. PTSNMP ActiveX control (ptsnmp32.ocx) is responsible for providing the SNMP Manager API and have Methods, Events and Properties. For event notification, PTSNMP ActiveX control signals an important protocol event by firing an appropriate event with necessary arguments that indicate the state of the session in progress. PTSNMP ActiveX control does not allow the application to generate more than one event at any one time. Therefore for debugging purpose, if break point is inserted than no other events is fired. Similarly, if MessageBox is used, all other communication events are disabled while the MessageBox captures the thread of execution. Also, if a DoEvents loop is placed within any event, all other communication events will be disabled.

Figure 5.7 illustrates the flow of information between automated system (SNMP Manager) and SUT (SNMP Agent). The SNMP Manager supports SNMP version 1. The Manager uses any local port for initiating a GetRequest, GetNextRequest or SetRequest (the request will normally be addressed to port 161, the SNMP Agent standard port). The

Agent will always reply to the source address of the packet. In the case of SetRequest, a GetResponse packet is used to confirm the updated information on the Agent. For Traps, the Manager uses local port 162 and makes it open to get the Trap message from the Agent.

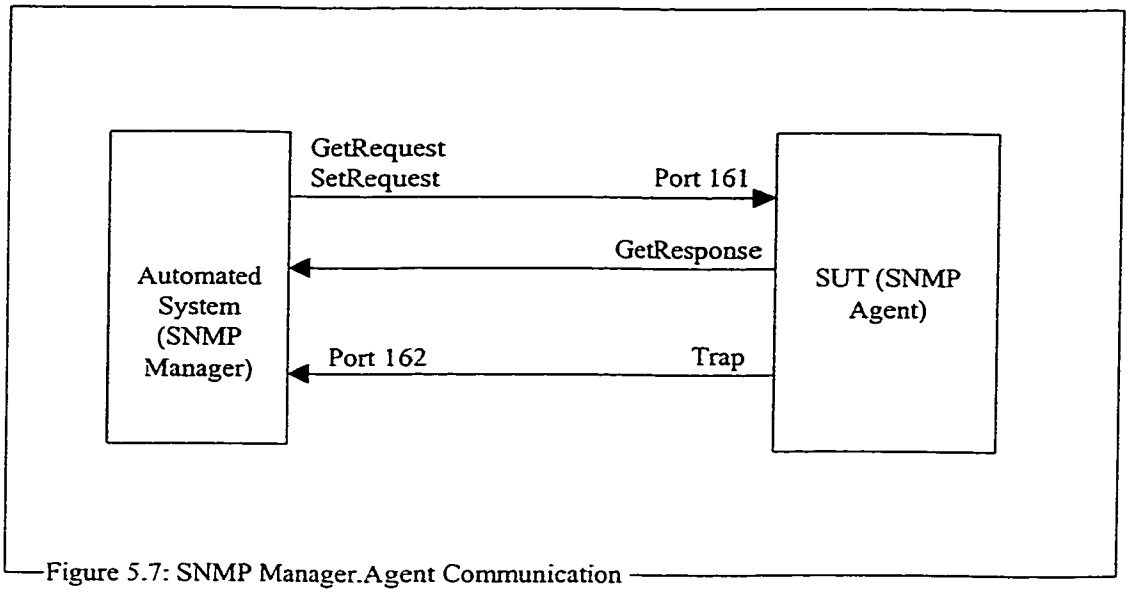

Figure 5.7: SNMP Manager.Agent Communication

SNMP Manager API takes all responsibility of accessing the SUT MIB by hiding all the details of SNMP protocol. It helps in building the network management application in Microsoft Windows, a much easier task by providing an asynchronous, event driven interface to the messages used by SNMP. The SNMP interface:

1. sets the SNMP Manager session by using the **Open** method – (refer to Appendix C, Figure C.1 for syntax and its usage in the application).
2. handles the building and encoding of outbound messages through the **SendGetRequest**, and **SendSetRequest** methods – (refer to Appendix C, Figure C.2 for methods).
3. handles the parsing and decoding of inbound messages and notifies the application by firing the **OnRcvSnmp** event – (refer to Appendix C, Figure C.3 for methods).

FTP Client API:

FTP stands for File Transfer Protocol. It provides a standard method for transferring files between computers. FTP can be used for many file-related activities such as:

- getting directory listings
- retrieving files
- uploading files

An FTP client (ptftp32.ocx) accesses files stored on a server. An FTP server responds to FTP clients, providing directory and file data. FTP uses both TCP and Telnet, so FTP differs from other protocols in that it actually consists of two connections—a control connection which uses Telnet, and a data connection which uses TCP.

An FTP connection is generally made between a client and server. However, a client can assist in connecting two servers together to transfer files between them. FTP Client API uses the standard FTP-RFC 959.

During login, a user name and password must be given. In addition to standard user accounts, many FTP servers support anonymous FTP, where the user name is "anonymous" and the password is the user's email address. Once the login is complete, commands can be sent across the control connection. Typical commands are RETR (retrieve a file), STOR (upload a file), and QUIT (logoff).

Data can be transferred over the data connection in many different modes i.e. File type, format control, structure, and transmission mode. However, the most commonly used mode is file type. There are four types of data for file type and the two most commonly used are:

- **ASCII.** This is the default mode. Different computer systems implement new-line characters in different ways. Many UNIX systems use the ASCII character 13 (CR). Microsoft Windows uses ASCII character 13 followed by 10 (CR-LF). When ASCII mode is enabled, proper conversions are made between the server and client. This mode should be used only for text files. Using this mode on binary files may cause them to become corrupted.
- **Image (binary).** This sends an exact replica of the file across the connection, preserving each byte with no conversions. It should be used for all binary files, which include all programs and many document files.

Client FTP API always ensures the correct data type is set before any data transfer is initiated.

When an FTP session is initiated with the Login method, a control connection is formed. The control connection is used for login, sending commands, and receiving information. The control connection utilizes the Telnet protocol. It usually uses the standard FTP port.

Once a data connection has transferred its data, it is closed. The control connection always exists, but the data connection is dynamic. Note that only one data connection can exist at a time.

FTP not only provides the data stress but also presents a real environment. The automated test system uses the FTP utility in many test cases. All the data path test cases use the FTP utility for data stressing. The SUT performance test case uses the FTP capability for measuring the SUT performance. The main application takes the following action to perform the FTP (Refer to Appendix C, Figure C.4 that shows the retrieving and

storing files through FTP Client API.):

1. The main application contacts the Client FTP API to retrieve or load file from desired FTP server.
2. The client chooses an unused port number to accept the connection on, because the client is always in control of file transfers.
3. The client waits for a connection on that port.
4. The client sends the port number to the server over the control connection, using the PORT command.
5. The client sends a command, which transfers data across a data connection (such as STOR, RETR, or LIST).
6. The server connects to the correct port using port 20.
7. Once the data connection is established, the main application retrieves (RETR) or stores (STOR) a file from/to the FTP server.

ICMP (Ping) Messages:

The Microsoft ICMP.DLL is an undocumented API for sending ICMP echo packets, also called "pings". Ping (ICMP) is a very useful tool to determine the network connectivity. When ping succeeds, it provides the RoundTripTime in milli second (ms) for the ping to complete, the returned data (NULL terminated), the size of the returned data, and the remote host IP address.

In some of the data path and performance related tests of SUT, it is required to check the data path first before doing any further testing. The automated test system uses ICMP application for determining the availability of data path across the SUT. The echo property of the ICMP packet allows checking both upstream and downstream data path.

The main parameters required to assemble the ICMP packet are:

- file handler
- remote Host IP address (Destination Address)
- message that needs to be echoed
- length of message
- time to live i.e. how many hops to pass before die
- time out if the message does not receive during this time

The main application interfaces with the ICMP.DLL file and passes the necessary parameters. ICMP.DLL assembles the packet and sends it out. It informs the main application that whether the ping was successful or not. Appendix C, Figure C.5 shows formation of ICMP packet and determination of data connectivity.

Serial Communication:

As mentioned earlier, the automated system uses both of its serial port (COM1 & COM2) to communicate with SUT and Loop Simulator. Loop Simulator provides a perfect test bed for ADSL product by simulating 30 twisted pair copper cables. It provides GPIB (IEEE 488) & RS232 interface to integrate with test systems. In the automated test system, it is integrated through its RS232 port. The port setting and cable configuration of RS232 through which it communicates with the automated system are shown in Figure 5.8 and Figure 5.9 respectively.

In section 4.5.1, the MSComm control provides serial communications for the main application by allowing transmission and reception of data through a serial port. The main application pools events and errors by checking the CommEvent property. The automated system uses the same method and message format to communicate with SUT

which it uses in the Hardware Testing. However, different message format is used for Loop Simulator. To communicate with Loop Simulator, the method is the same but the message format is different. Appendix C, Figure C.6 shows a basic module by which the automated test system sets simulated loop in the Loop Simulator.

Port Settings	
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Figure 5.8 : Port Configuration

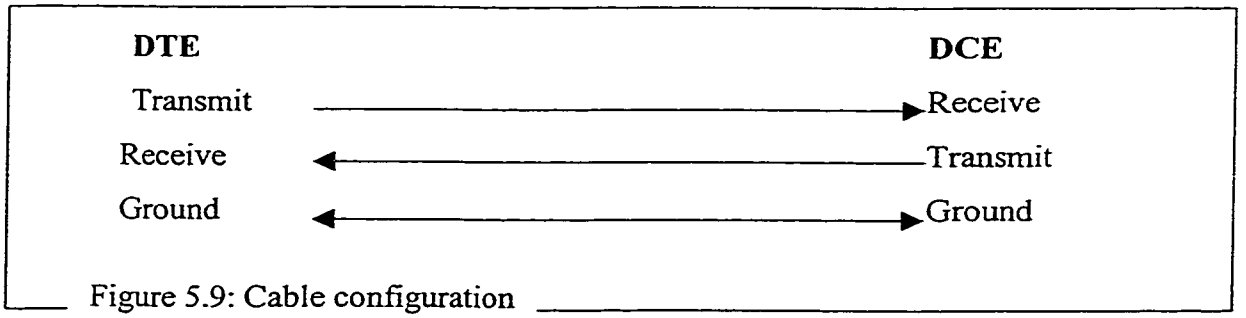

Figure 5.9: Cable configuration

5.4.2 COMMAND FORMAT

Message sent to Loop Simulator through serial interface for simulating a loop must be terminated with line feed character (decimal 10, hex 0A, LF). To ensure that no characters are left in the receiving buffer of the Loop Simulator from a previous incomplete command, a line feed character may be sent before sending the new commands. The carriages return character is not a valid terminator, and it invalidates the

last command. To avoid this problem a **semi-colon** ";", is appended after the string or at the end of command (inside the quote) as shown below:

```
transmit "command"+CHR$(10);
transmit "command;"
```

A Command can be a Common Command or a Device Dependant Command. Common Commands are preceded by the character "*" and Device Dependant Command are preceded by a colon, with a colon separating each level of the command. Commands may be either in upper or lower case. The multiple commands can be concatenated by separating each command by semi-colons. The format of a multiple command and its example is shown below and usage is shown in Appendix C, Figure C.6.

:SET:CHAN:LOOP <Loop Name>;TAP_A <NRF>;LINE <NRF>;TAP_B <NRF>

:SET:CHAN:LOOP VARIABLE_26_AWG; TAP_A 500;LINE 10k;TAP_B 500

In order to confirm that the desired loop setting has been set, the query command is used. When Loop simulator receives the Query Command, it replies with the actual loop setting value. The format of the Query command is the same as the normal loop setting Command except that the data (e.g. Loop Name, NRF,) is replaced by a question mark "?". The example of "Query" command is shown below and its usage is shown in Appendix C, Figure C.6.

:SET:CHAN:LOOP?; TAP_A?;LINE?;TAP_B?

5.4.3 TEST RESULT PRESENTATION

It was desirable to present the test result in some standard format of PC based application so that it can be ported, integrated in the verification report document, attached with e-mail, and viewed in the same format by all group members, project team

and management team. Microsoft Excel was an excellent choice to display the : standard formatted result not only in the real time but also without doing any transformation. However, it is found that some of the automated test system's PCs in the Lab were not equipped with Microsoft Excel. The alternative solution was to display the real time result on GUI shown in Figure 5.10, made by the VB ActiveX component and stored the results in DOS format with a "comma" delimiter. Later, this DOS format file can be opened in Microsoft Excel format. To fulfill the above requirements the automated test system provides the option to display the result in any format. The above mentioned test result formats are shown in Figure 5.10.

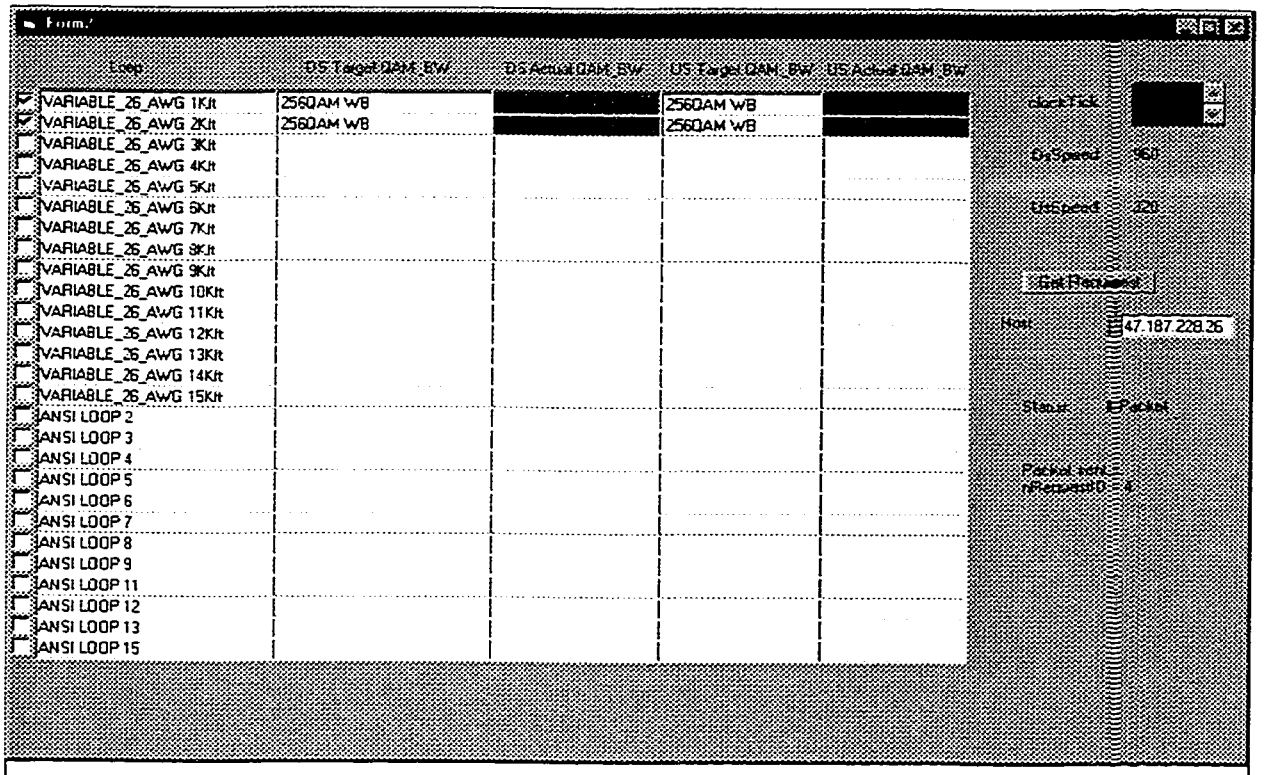

Figure 5.10: GUI format result

Result in Excel Format:

Object Linking & Embedding (OLE) automation allows the automated test system to load and open the Microsoft Excel program and write the results on the Excel sheet directly in real time shown in Figure 5.11. The Excel Sheet acts as an object that is exposed by Excel program. Automated test system (VB) application controls the Excel Sheet through properties and methods exposed by the OLE Server application (Excel Program).

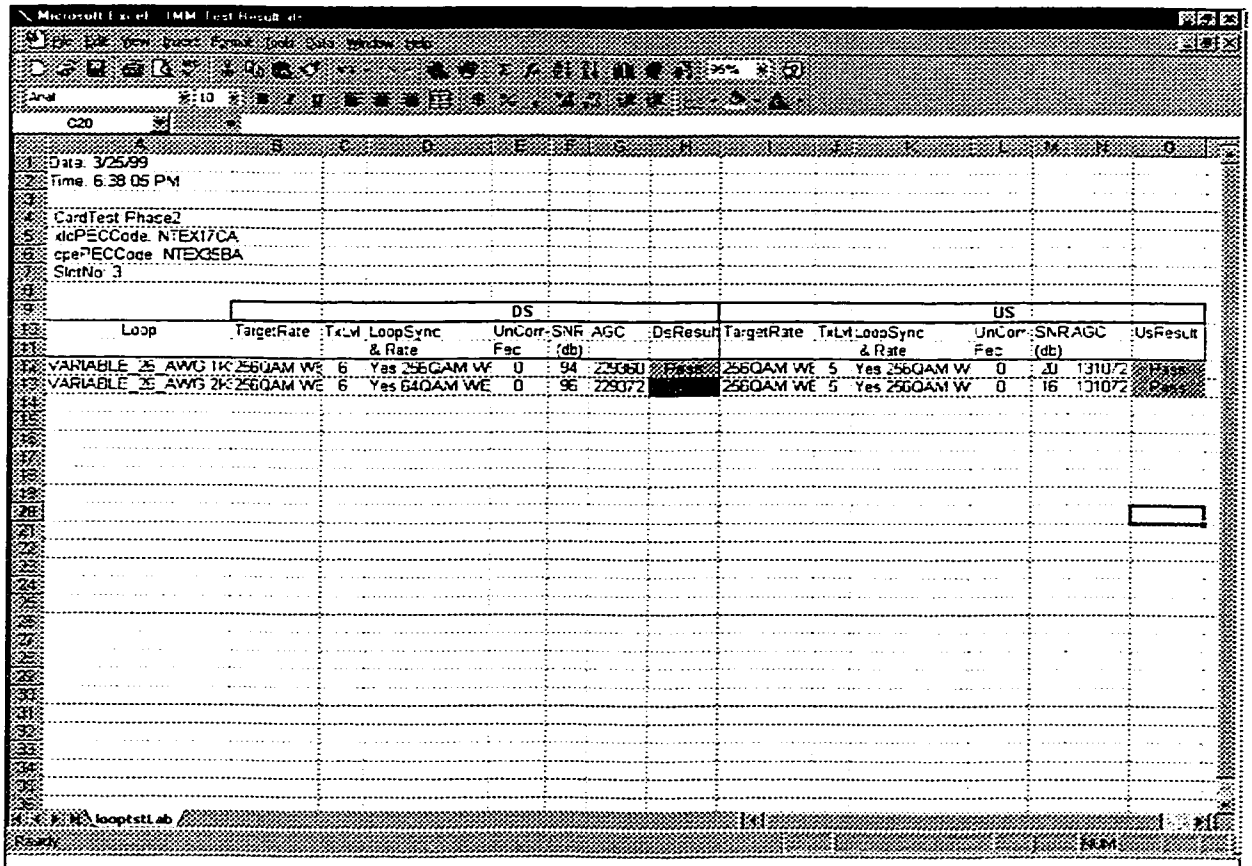

Figure 5.11: Result in Excel

"CreateObject" method opens the new worksheet (OLE Automation Object) and "GetObject" method opens the existing worksheet. "Cell(row, column)" is used to access (read/write) any cell of a given row and column. By using these & other methods and

properties, certain desired operations are performed on the Excel sheet. Appendix C, Figure C.7 shows all these objects, methods and properties which are actually used by the automated test system for presenting the test result.

Result in DOS Format:

Result in DOS format can not be seen in the real time. The application saves the result as text file. The result in DOS format is required only when the Microsoft Excel application is not installed on the automated test system PC. As shown in the Figure 5.12, "comma" delimiter separates each information in a row. Some applications like Microsoft Excel transform these ".txt" file into its format. Microsoft Excel also transforms these ".txt" files into the ".xcl" format. Excel uses "comma" delimiter and substitutes it with columns. Figure below shows the DOS format text file. "LineFeed" & "CarriageReturn" represent each new row and " comma" represents each column in a row.

Date: 3/25/99
Time: 6:38:05 PM
CardTest:Phase2
xlcPECCode: NTEX17CA
cpePECCode: NTEX35BA
SlotNo: 3

,,,, DS,,,,,,,, US
Loop,TargetRate,TxLvl,LoopSync,UnCorr-,SNR,AGC,DsResult,TargetRate,TxLvl,LoopSync,UnCorr-,SNR,AGC,UsResult
,,,& Rate,Fec,(db),,,,,& Rate,Fec,(db)
VARIABLE_26_AWG 1Kft,4QAM NB,6,Yes 4QAM NB,0,94,229360,Pass,4QAM WB,5,Yes 64QAM NB,0,20,131072,Pass
VARIABLE_26_AWG 2Kft,4QAM WB,6,Yes 4QAM NB,0,96,229372,Fail,4QAM NB,5,Yes 64QAM NB,0,16,131072,Pass

Figure 5.12: DOS Format

Result in VB GUI:

The Automated test system's PC which are not equipped with Microsoft Excel application save the result in DOS format file and later on transform it into the standard Excel format for viewing and distributing the result. It was felt that during the test run, the testers want to view the result in real time as the initial test result confirms them that the test bed setup is proper, the intermediate equipment are working normally, and the new SUT load is in order. At certain occasions, especially during regression testing when the tester leave the test for overnight soak, the visual monitoring of test result saves their time by displaying errors.

Figure 5.10 shows the GUI of Automated Loop Testing Result. It displays the features & other parameters with their values on which the loop-testing feature is dependent. The GUI uses the form on which Labels and Text Boxes are nicely placed. The Labels are used for headings and Text Boxes are used to display the features and parameters and their corresponding values. As can be seen that the first column heading "Loop" is written by using the Caption property of Label. Similarly the heading DS & US features are written on the remaining columns using Label control. Under the first column heading "Loop", different standards loops are written by using the "Text" property of Text Boxes. Similarly under the heading DS and US features, and in front of each Loop row, many Text Boxes are placed to represent the feature and parameters with their values. The automated test system checks the feature's value against an 'oracle'. If the value is lower, the test fails otherwise it passes. The automated test system change the color of Text Box to Red (fail) or Green (pass) according to the result by using the BackColor property of Text Box.

One of the important properties of GUI is that all of its controls are properly aligned. If they will not be properly aligned, the form will look crooked, and out of whack. If the control is not perfectly aligned with rows, one may pick the parameter values, which belonged to other rows and this could lead to confusion. Similarly, confusion may be developed if the controls are not vertically aligned. Visual Basic IDE provides numerous commands for aligning controls, which are all listed on the format menu. By using these commands all the controls are properly aligned and resized.

5.4.4 GRAPHICAL USER INTERFACE OF SOFTWARE TESTING

Figure 5.13 shows the Graphical User Interface (GUI) of Software Testing. As can be seen that GUI is very flexible, it provides a rich set of options therefore is used for all software tests.

Using different GUI for different tests can lead to wrong setting selection. The automated test system provides a single GUI system to reduce errors. One of the features of the automated test system is that it can also anticipate mistakes and before starting up the test, it checks all setting. Normally, default setting has been provided for different configuration for avoiding mistakes. However, if the tester uses any specific setting for any particular reason, the automated system checks the setting and generate error message if any discrepancy is found. Generally, the system generates two types of error messages. In one type of error message the system does not allow the tester to alter that particular setting if the SUT does not support that feature. For example, the tester set the option 256QAM, WB, for phase1 SUT hardware, not supported by the SUT hardware.

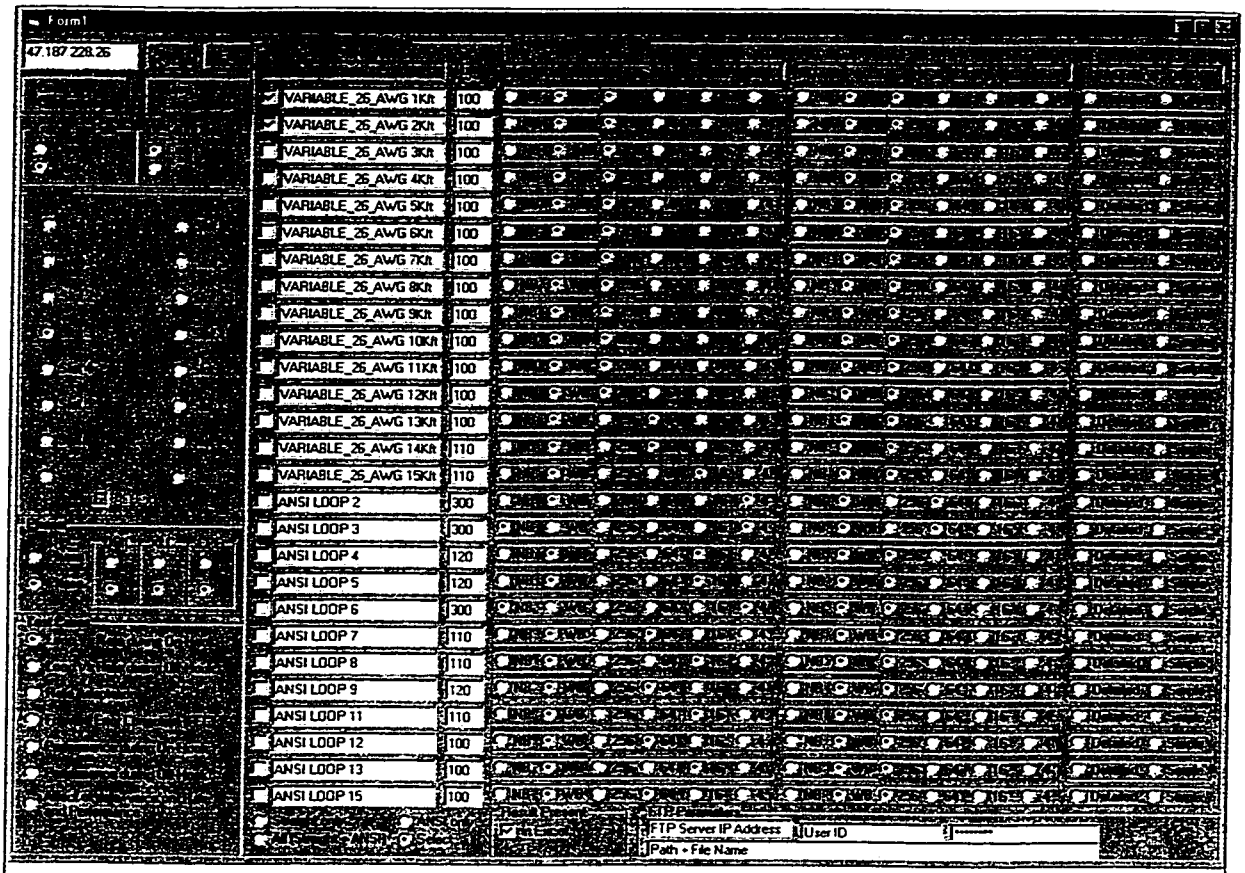

Figure 5.13: Main GUI

The error message is generated against that faulty setting, shown in Figure 5.14. The error message warns the tester that this setting is illegal. The automated system changes the setting to default setting for that particular SUT configuration. In the other type, the system allows that setting, however it is not a desirable setting. For example, during the loop test, the tester set a lower rate on any particular loop, whereas a higher rate can be achieved on that particular loop. In this scenario, the automated test system generates a message as a reminder that a higher rate can be achieved on this particular loop and inquires the tester either to accept the new setting by clicking "Yes" button or go back to the default setting by clicking the "No" button. This type of error message is shown in Figure 5.15.

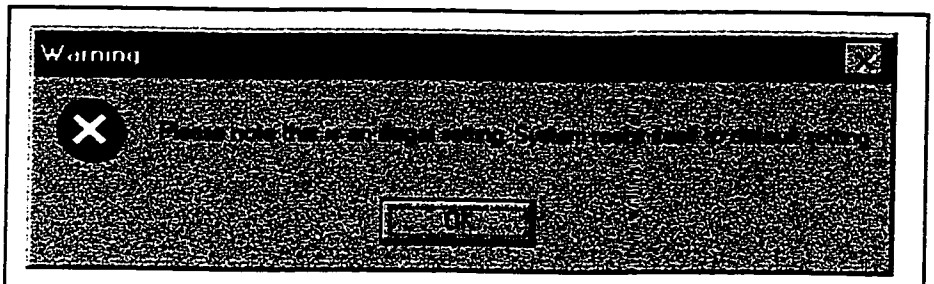

Figure 5.14: Msg Box for "Illegal Setting"

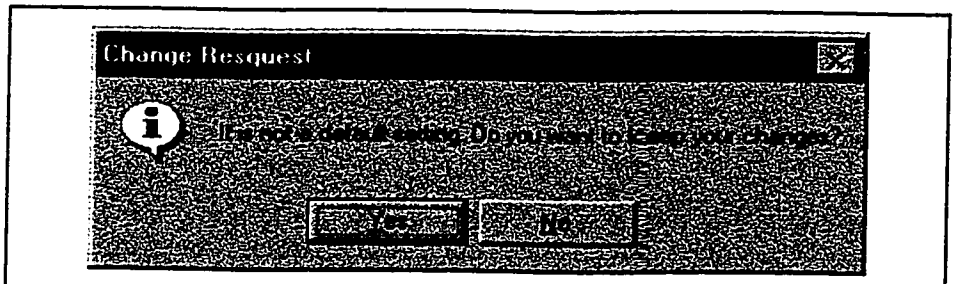

Figure 5.15: Msg Box for "not a default setting"

The GUI is made up from the basic Visual Basic ActiveX control. It has a Form, Frames and arrays of Labels, Text Boxes, Buttons, Option Buttons, Check Boxes. All these controls are placed on one Form. As can be seen, many sets of Option Buttons are placed on a form. However, as per the rule of Option Button, only one Option Button can be selected at any one time. Frame is an another Control, which carries other Controls. Due to Frame, many sets of Option Buttons can be selected on a GUI. The GUI uses the graphical power of visual basic. Visual Basic provides richness and alignment of controls and therefore, accommodates 155 number of Frames, 28 number of CheckBoxes, 418 number of OptionButtons, 59 number of TextBoxes, 4 number of Labels and 4 CommandButtons on a single 11880 x 8535 units size of the given GUI form. The functional description of GUI is following:

1. On the top left there is Text Box which carry an IP address of SUT. The Text property of the Text Box allows writing the IP Address in the Text Box. As shown in Appendix C, Figure C.8, SNMP Manager API uses this address to establish a session with SNMP Agent. The automated test system supports only one IP address so only one device (SUT) can be accessed at any one time. However, there is no limitation for any class of IP addresses. The IP address could be class A, class B, class C and classless, depending on SUT to which network it belongs. In future, the automated system could be upgraded to control more than one SUT by accepting more IP addresses.

2. Next to the IP address Text Box, a command button named "Start" (cmdStart) is located. The style of command button is standard (i.e. set to 0) and the caption property hold the text "Start" with default font. As the name implies, it starts the test. However, as mentioned above, it checks three important components of the software test. The first component is slot#, i.e. which line to be tested. The second component is loop setting, i.e. which type of loop is to be set at the Loop Simulator for the above mentioned line (slot#). Remember that the Loop Simulator is an important component of the software testing as it triggers the software component to act according to the configured loop. The third component is TestCase#. If any of the above three component is not set, the automated test system will not start the test and generate a message as a reminder for setting up these two components. The checking of the three components and the error message, if any of the three components are not selected, are shown in Appendix C, Figure C.9.

3. "Exit" command button lies next to the "Start" command-button. Its style is also standard and its caption property holds the text "Exit". It is used to terminate the application.

4. Below the "IP address " text box, there is another button which is twice the size of the "Start" command button named "Load Configuration". "Load Configuration" command button is a standard command button with the caption property set as "Load Configuration". As the name implies, the "Load Configuration" command button loaded the pre-configured configuration, which is stored (storing is described in the next section) by the tester during its previous testing. This is a very useful feature of automated test system as it saves considerable time by directly loading the desired configuration in no time. It is noted that the tester usually has two to three different type of configuration, which they use to track some software bugs.

   The File, which is loaded, it should be in the DOS text format. The automated test system extracts the configuration from the text file by parsing it through the Visual Basic "InStr( )" built-in function.

   For loading the file, an important Active-X control, "CommonDialog" control, is introduced. The "CommonDialog" control provides a standard set of dialog boxes for operation such as opening (loading) and saving files. While loading (opening) the file, the File Open Dialog Box that would appear is shown in Figure 5.16. It can be seen that, it works exactly as it does in Microsoft Word and other Windows applications. The tester can select a pathname or a filename, change the view by clicking one of the View buttons, open the file as read only, and even select another drive or computer to choose from by clicking the Look in drop-down list box.

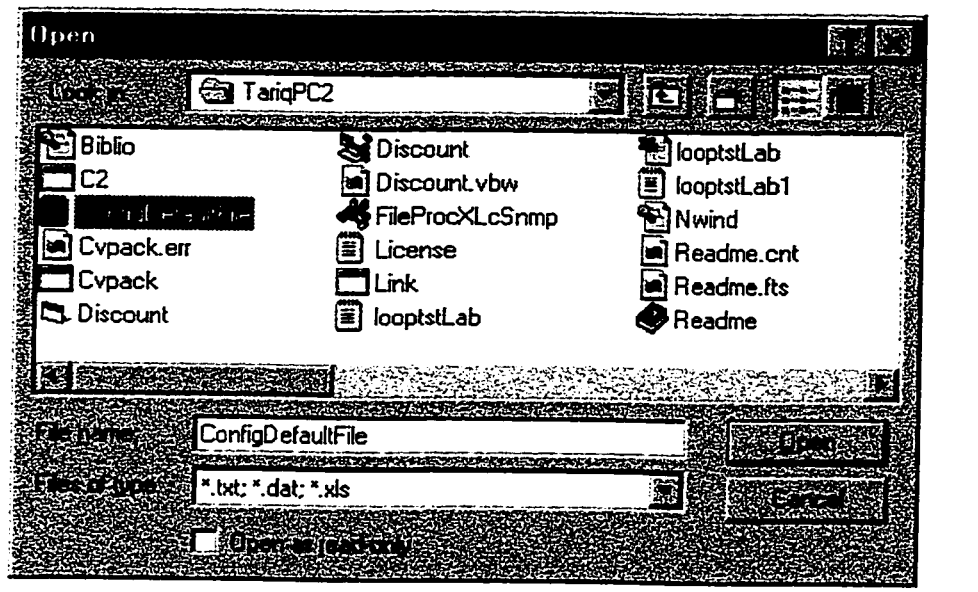

Figure 5.16: Shows common File Open Dialog box

The CommonDialog control provides an interface between Visual Basic and the routines in the Microsoft Windows dynamic-link library Commdlg.dll. To create a dialog box using this control, Commdlg.dll must be in the Microsoft Window SYSTEM directory. During the design time, the CommonDialog control is displayed as icon like other controls. However, its icon size can not be changed, as during run time, the icon is not displayed. When the dialog box is displayed it can be displayed in anywhere on the screen; there is no way to specify where a dialog box is displayed. The way the automated system uses the CommonDialog control for loading the file is shown in Appendix C, Figure C.10.

5. "Save Configuration" command button lies next to the "Load Configuration" command button. It is of the same size as "Load Configuration" command button except that its caption property set to "Save Configuration". As the name implies, it is used to save different configurations in the DOS format text files to be used later.

"Save Configuration" command-button and "Load Configuration" command-button work identically except that the former saves the DOS formatted configuration files and the later load these DOS formatted configuration files. "Save Configuration" command button also uses the "CommonDialog" Active-X control for saving the file. It uses the File Save Dialog Box, shown in Figure 5.17. File Save Dialog Box also works similar to other Window application and provides the same facility i.e. selecting pathname or filename, changing view, opening file as read only, selecting another drive etc, as provided by File Open Dialog Box. The way the automated system uses the CommonDialog control for saving the file is shown in Appendix C, Figure C.11.

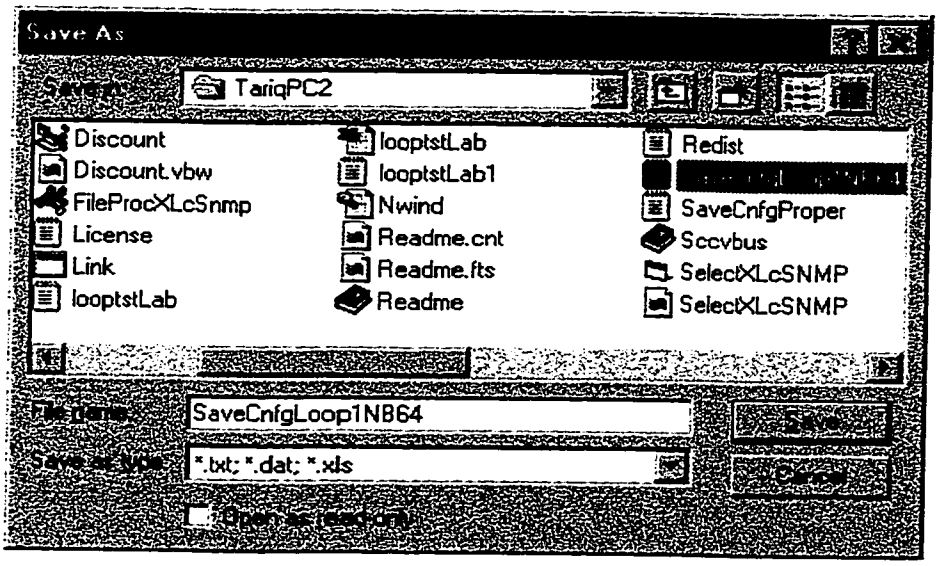

Figure 5.17: Shows the common File Save Dialog box

6. Below the "Load Configuration and Save Configuration" command-buttons, there are two frames captioned with "TraceComPort" & "DLS400ComPort". The advantage of Frame caption property can be seen that it eliminates the usage of additional Label controls by providing headings to Frames.

As mentioned in section 5.4.1, the automated test system are connected with Loop Simulator and SUT's debug port through RS232 (COM) port. The DLS400ComPort" and "TraceComPort" frames are responsible for that serial (Com) port connectivity by allowing the Loop Simulator and SUT's debug (trace) port to connect with either Com1 or Com2 port of the automated test system. Both the frames have two Option buttons, captioned with Com1 and Com2. The reason of using the two frames is because the two options buttons are needed to select for serial connectivity with two devices. A check mechanism has been provided to avoid selecting the same Com port for both the devices at any one time, shown in Appendix C, Figure C.12. If tester selects the Com1 for Loop Simulator, the Com2 will automatically be selected for SUT's debug port and vice versa.

7. The "Slot#" frame is located below the TraceComPort and DLS400ComPort. The heading "Slot#" shows that its caption property is set to "Slot#". It can be seen that the white boundary line around the Frame makes it appear as a separate piece of GUI which is very well fitted with the remaining pieces of GUI. Again frame is used here to select more than one option button on a form. It contains sixteen Option Button marked with Slot number from 0 to 15 and a Check Box marked with All Slot. The Option button allows selecting one slot (or one line) at a time to be tested. Selecting the slot Option button is a mandatory requirement of the test system. The test system gets the information, regarding the line (slot) to be tested through these slot-Option buttons. Appendix C, Figure C.9 shows how the system extracts the selected slot. If any of the slot-Option buttons is not selected, the system generates an error message as a reminder, shown in Appendix C, Figure C.9.

The CheckBox, marked as All Slot, has been grayed out for future purpose. The automated system can be upgraded with minimum changes to test all slots simultaneously. However, in that case the slot-Option button has to be replaced with CheckBoxes so that all slot or more than one slot can be selected simultaneously.

8. Another Frame captioned as Phase is situated below the Slot# Frame. New software-features/hardware-enhancements have been developed in the SUT. As of today, there are two types of SUT's hardware i.e. Phase-1 DBIC/xLC/CPE and Phase-2 DBIC/xLC/CPE. All these hardware are backward and upward compatible. However, different features are supported on different combination. Therefore, the system must know which hardware combination is being used so that it can test the SUT functionality accordingly. This Phase frame is responsible for getting the correct phase of hardware.

   The Phase Frame contains three more Frames, which are captioned as "DBIC", "xLC", and "CPE". Again, Frames are used wherever more than two Option button are required. The Phase Frame also contains two Option buttons named Phase1 and Phase2. Similarly, the remaining three frames also have two Option buttons. However, names have not been given to these Option buttons. These option buttons are placed in the same row as of Phase1 and Phase2 Option button of Phase Frame. So the Option buttons in the Phase1 Option button row belong to Phase1 hardware and the Option buttons in the Phase 2 Option button row belong to Phase 2 hardware. As mentioned above, the test system does not simply takes the tester selected values, but rather checks to determine the hardware's phase by interrogating with the SUT on SNMP platform. If the hardware configuration is found to be different from that of

the tester's selected value, it generates a message informing the tester about the correct hardware setting. This avoids any confusion by informing the tester that what output results to be expected against this hardware configuration. Appendix C, Figure C.13 shows how the system determines the phases and generates the message for the tester.

9. "Test Cases" frame is located below the "Phase Frame". The "Test Case" heading implies that its captioned property set to "Test Cases". Again due to the same reason, the frame is used to select more than one option button on a form. The frame carries seven option button each with marked as different test case.

   To test the various features of SUT, the testing is divided onto seven test cases. The test case frame represents those seven test cases in the form of seven option buttons. The test case frame allows selecting one test case at a time by clicking one of its test-case-option button. The selection of test-case-option button is mandatory. The test system gets the information from this frame that which test is to be executed and accordingly set the environment to test features in that test case. Appendix C, Figure C.9 shows how the test system extracts the selected test case and generates error message if any test case option button is not selected.

10. The Frame, which is captioned as Loop/TimeGivenToSync, is a large frame and is responsible for setting up Loop(s) and time for each Loop. It consists of Labels, Option Buttons, and arrays of CheckBox, TextBox. On the top of this Frame, two Label controls are placed, with the captioned property set for "Loop" and "Time (sec)". Under the "Loop" Label, there are twenty-four arrays of CheckBox and TextBox controls placed side by side. Similarly, under the "Time(Sec)" Label, there

are twenty-four arrays of TextBoxes, which are placed adjacent to the right side of the Loop-Label's TextBoxes and these Time-Label TextBoxes are almost $1/4^{th}$ the size of Loop-Label's TextBoxes.

The TextBoxes under the Label "Loop" represent the twenty-four standard loops and each of their Text property is set for one of the standard loop names. Similarly, the TextBoxes under the Label "Time" represent the time for which the SUT can attain the max speed for loops mentioned in the Loop-label TextBoxes. The time mentioned in the Time-Label Textboxes can be changed in the real time and this changed configuration can be saved through the SaveConfiguration button. The CheckBoxes are responsible for selecting the loops and the duration of time, which are written on the Loop-Label TextBox and Time-Label TextBox respectively. The selection of CheckBox is a mandatory requirement similar to the selection of slot number. If any of the CheckBox is not selected, the testing will not start and the testing system will generate the same error message, shown in Appendix C, Figure C.9. The CheckBoxes, the Loop-Labels' TextBoxes and Time-Labels' TextBoxes, placed side by side to each other, form a one set of information. That is, if first CheckBox is selected than its mean that the test system sets the 26 AWG, 1Kft, (first array of Loop-Label TextBox) loop in the Loop simulator for 100 seconds (first array of Time-Label TextBox).

To select all loops or straight loops or ISDN (ANSI) loops, one by one takes time. The test system provides options of selecting loops one by one; or all loops; or all straight loops; or all ISDN loops in one mouse key-click. At the bottom of Loop/TimeGivenToSync Frame, four Option buttons are provided which are

responsible for the above-mentioned features. The captions of the Option buttons reveal their functionality. Appendix C, Figure C.14 shows how these one clicked Option button select and deselect "all", "straight-only", "ISDN (ANSI)" and "selected" loops.

11. On the bottom right hand side of the Loops/TimeGivenToSync frame, there is a frame captioned as "Result Present". The frame has a CheckBox which is captioned as "in Excel". This frame is responsible to display the result in Microsoft Excel by enabling the "in Excel" CheckBox.

    As mentioned in section 5.4.3 that all PCs in the lab are not equipped with Microsoft Excel application. Therefore, despite representing the result in Excel standard format is a desirable choice, it is not possible. However, to cover the non availability of Excel application drawback, an alternate choice is provided to display the result in real time on the VB GUI and store the result in the DOS format with comma "," delimiter to transform the result in the Excel format. By enabling the CheckBox "in Excel", the result will be displayed directly in the Excel application (Appendix C, Figure C.7). The default value of the Check Box "in Excel" is set deliberately to false. The idea is to remind the tester to check the availability of Excel application on the automated system PC prior to assume that the result will be displayed in Excel format. The confirmation of assumption has saved lot of time as it is observed several times that the tester started the overnight test and left the Lab, assuming that the Excel application is present. When he came back to retrieve the result he found the system error message. This way, the system forces the tester to confirm the availability of Excel application before expecting the result in Excel format.

12. Next to the "Result Present" frame, there is another frame captioned as "FTP Parameter". As the name implies this frame carries the FTP parameters required in transferring file (FTP). As mentioned in the section 5.4.1 – FTP Client API, the FTP Server IP address, the path and file name of the retrieving file, the user ID and the password are required to accomplish the FTP. Four TextBox in the "FTP" frame carries the above mentioned requirement and their text property shows the parameter for which it responsible to. The last bottom right side TextBox of the FTP frame is responsible to carry the password parameter. Since the password is a secure feature which should not be displayed on the screen, therefore the password property of the TextBox is used which displays the control character, asterisk (*), instead of actual password string.

    FTP is not implemented in every test case. It only uses where the datapath or the performance is required to be tested. Forward Error Correction (FEC), Performance during On-Hook, Performance during Off-Hook and Affect of Data + Voice on Rate and Performance test cases are the only test cases which incorporate FTP in their testing. Appendix C, Figure C.4 shows the code which uses the above mentioned TextBox in the FTP application.

13. The last frame of the GUI is the biggest frame and its heading is "TargetRate/Trace_On". The Frame "TargetRate/Trace_On" contains Labels, numerous Frames to carry many Option buttons. As mentioned in section 3.3.3C, the SUT is an ADSL equipment and it supports Full Duplex Downstream (DS) and Upstream (US) rate at its loops. On the top of "TargetRate/Trace_On" Frame, there are Labels, which are marked as "DS", "US" and "Trace On". Under the "DS" and

"US" Labels, there are targets rates and is indicated by the Option buttons. These Option Buttons are in the same row as the Loop-Labels TextBoxes and indicate the rate of the corresponding loop. These targets rates are determined by a group of people whose expertise are in ADSL loops. On a perfect ADSL system, it turns out that their calculation is 99.99% correct. The test system uses these target rates in its default configuration for phase2 equipment. The tester can change the target rate and store the new configuration by using the "SaveConfiguration" command button. The target rates are mentioned in QAM and Bandwidth where NB stands for narrow band and WB stands for wide band.

Under the "DS" and "US" Label, in each row, there are two frames. One frame represents the bandwidth and has two option buttons which, are captioned as "NB" and "WB" respectively. The other frame represents speed in QAM and has four option buttons, which are marked as "256", "64", "16", and "4" respectively. There are total 24 rows under the "DS" & "US" Label which depicts the same information. Under the third Label "Trace On", there are 24 rows of frames and each frame carry two Option buttons which are captioned as "Detailed" and "Simple". The SUT is equipped with trace facility and can be enabled through its debug (serial) port. There are two types of traces available, Detailed and Simple. When the trace is enabled, the SUT start dumping its trace information on its serial port where the test system collect the information and store it in the file for later designer review. The Detailed tracing option dump more data than the Simple tracing option. By default, the tracing Option is disabled, as it is more a designer requirement than the functional testing requirement.

5.4.5 TEST CASES

The automated test system tests the SUT software by tracking the software cycle of different SUT's sub processes. It monitors the software responses by changing the external environment usually by loop simulation and call/data generation. Later, it verifies the responses against an "oracle" to confirm that the responses are in accordance with the changes or not.

As mentioned earlier, the automated test system mostly tracks the software responses by accessing the SUT's MIB objects using SNMP packets on the Ethernet segment. The 10/100 MB speed makes it possible to track the software in real time. This was not possible with the 9600 debug serial port to track the software instantly. However, the load provides an option (can be chosen from the GUI), to dump its traces through the serial port. These traces are exclusively for the designer. Neither the tester nor the automated test system can take any advantage of it. The automated test system can not extract any useful information from these traces and thus can not verify the software functionality or software sanity.

In hardware testing, hardware is closely coupled. To test one specific hardware explicitly, other hardware is also tested implicitly. Similarly, the software modules of SUT are also tightly integrated. Generally, the software modules can be divided explicitly into two, Loop Controller & Data Path. Within the Loop Control Modules, the modules are closely integrated and it is very difficult to test explicitly one particular module of Loop controller.

There are many test cases performed by the automated-system fully/semi automatically. Most of the recommended ADSL related tests proposed by Fadi [Dao98],

have already been incorporated in the automated system. However, the proposed DC characteristics test have not been included in the test system as SUT is an integral part of Nortel Networks DMS telephone switch. DMS built-in-tests checks these DC characteristics features. Following are the detailed test cases, which are related to the software functionality testing.

Test Case1- Loop Vs Achieved Rate during On-Hook:

(i) Introduction:

There are different types of loop available in the field and the customer premises are at different distance from the Central Office (CO). Ideally, each CO covers a radius of 18000-ft distance, adequate to reach almost all customer premises. This test case, test the maximum loop reach against the maximum achieved speed. The test case verifies the max rate achieved by the SUT against the 24 standard loops mentioned in appendix A. The result of this test justifies the implementation by showing the number of users accessing this technology on different types of loops.

In this test-case, the Loop Controller Modules are responsible broadly for achieving the maximum rate against different Loops. The module, which is explicitly involved for achieving the maximum rate against the different loops, is LoopConfig Process. Other modules which, participated implicitly are Line Monitor, Line Interface, SNMP Agent, Download Process, IP Stack, and Ethernet Driver.

The maximum rate achievement on different loops depends on two factors. One is the Error Power value and the other is CRC/FEC error rate. However, the later factor is not involved initially. It is involved when the data path is established and data has started across the SUT. Other parameters, which are involved in syncing up the loop and then

ultimately setting for maximum rate are AGC and Transmit Power Level. As mentioned earlier, the automated test system tracks the above mentioned parameters along with the achieved maximum rate parameter through SNMP protocol and presents these parameter results to the tester by GUI or by Excel sheet. The test system also compares the achieved downstream and upstream speed separately with the targeted value set by default or by tester. If the achieved downstream or upstream speed values or both are equal to or greater than the targeted speed values, the system declares it pass and highlights the achieved downstream and upstream speed GUI- TextBox / Excel-Cell background 'Green'. If any of the downstream or upstream speed values or both are less than the targeted speed values, the test system declares fail and marks that particular GUI-TextBox / Excel-Cell 'Red'. The other parameter values in the result help to determine that (i) if test is passed, all the Loop Control Modules are working properly (ii) if test is failed, which module(s) is/are responsible for the failure.

(ii) Operation Flow:

The automated test system:

1. initializes all parameters, checks the hardware phase and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 1, target rate and starts the selected test.
3. informs the tester to select the Slot#, Loops and Test Case if not selected earlier. If the target rate is not supported, it informs the tester that the selected target rate is illegal otherwise it starts the test.

4. resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch and so does the software modules.

   The SUT has a feature through which it remembers its previous state and when the loop changes, it tries to set the previous rate on the changed loop. Resetting the SUT, clears the previous information and the SUT establishes the loop synchronization from the start. Ethernet Driver, IP Stack, Download Process, SNMP Agent modules and Line Interface module of SUT accomplish this task.

5. sets the selected simulated loop on the Loop Simulator through its serial port.

6. waits for the time set by the tester in the main GUI for allowing loop to be synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules are involved in syncing up loop.

7. starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, Error Power, Power Level, AGC by accessing the SUT MIB through SNMP. The SUT modules which are involved at this stage are Ethernet Driver, IP Stack and SNMP Agent

8. converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

9. compares the achieved DS/US speed with the "oracle" (target speed).

   If the achieved speed ≥ target speed

   Test is passed. Mark the achieved speed Text Box at GUI or Cell at Excel Green

Else

Test is failed. Mark the achieved speed Text Box at GUI or Cell at Excel Red

End If

10. writes S/N ratio, Power Level and AGC in the output result.

11. if more loops are selected, go to step 4 until all selected loops are finished.

Test Case2- Loop Vs Achieved Rate during Off-Hook:

(i) Introduction:

A feature "Off-HookDetect" provided by SUT optimizes the data transmission level (Power Level) and rates for the off-hook condition. It has been found that when customer uses cheap telephone set which does not have the proper filter in it, an audible hissing noise may leak into the telephone set and can be heard by the customer. The parameter "Off-HookDetect" has been provided on each line to enable and disable this feature depending on the type of telephone set being used by the customer. Enabling this feature would cause increase in voice quality on the cheap telephone set at the expense of data rate drop when the telephone is off-hook. However, data rate will not drop for every loop. It may drop slightly (1QAM lower to existing speed) for harder loop. LoopConfig process is responsible for detecting the off-hook state. Off-Hook Detect feature works in the following manners:

- monitors the loop at both CPE and xLC end for an off-hook indication;
- reduces the transmit power level automatically in order to minimize the probability of interference occurring in the customer phone when an off-hook occurs;

- optimizes the data rates for the lower power level; and
- returns to previously established rates when on-hook occurs.

(ii) Operation Flow:

Test Case2 is an extension of Test Case 1. Initial steps are mostly common.

The automated test system:

1. initializes all parameters, checks the hardware phase and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, and Test Case 2.
3. sets the off-hook control value "enable" in the appropriate MIB through SNMP and sets the default target rate for off-hook control immediately after which the tester selects the Test Case 2 option button.
4. allows the tester to change the default target value (if not agree with the default target rate) and Start the selected test.
5. if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it. If the target rate is not supported, it informs the tester that the selected target rate is illegal otherwise it starts the test.
6. resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch and so does the software modules.
7. sets the selected simulated loop on the Loop Simulator through its serial port.
8. waits for the time set by the tester in the main GUI for allowing loop to by synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.

9. makes the line off-hook by sending the off-hook commands to the dialup modem, which is connected to its serial port.

10. allows the same time for syncing up the loop which is given in step 8.

11. starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, Error Power, Power Level, AGC by accessing the SUT MIB through SNMP. The SUT modules which are involved at this stage are Ethernet Driver, IP Stack and SNMP Agent.

12. makes the line on-hook by sending the on-hook commands to its serial port connected dialup modem.

13. allows the same time for syncing up the loop which is given in step 8.

14. converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

15. compares the achieved DS/US speed with the "oracle" (target speed).

If the achieved speed ≥ target speed

Test is passed. Mark the achieved speed Text Box at GUI or Cell at Excel Green

Else

Test is failed. Mark the achieved speed Text Box at GUI or Cell at Excel Red

End If

16. writes S/N ratio, Power Level and AGC in the output result.

17. if more loops are selected, go to step 6 until all selected loops are finished.

Test Case3 – Rate Adaptation Test:

(i) Introduction:

One more feature, which SUT supports, is automatic rate adaptation. This allows the loop to adjust dynamically its rate that best suits the condition of a line. If the loop encounters severe plant conditions or significant EMI, the service dynamically adjusts to a lower rate and if the loop encounters an improvement in the conditions, the service dynamically adjusts to a higher rate. It also allows the subscriber to receive the maximum data rate at an impressive bit error rate of 10^{-7}. Multiple downstream and upstream speeds provide a wide range of service options. Loop Configurator module is explicitly responsible for rate adaptation. However, other modules participating in the rate adaptation testing are Ethernet Driver, IP stack, SNMP Agent, and Line Monitor.

(ii) Operation Flow:

The rate adaptation test is a loop related test so it is an extension of Test case1. Most of the steps are common.

The automated test system:

1. initializes all parameters, checks the hardware phase, sets the loop simulator in the bypass mode and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 3, target rate (if not agree with the default target rate) and starts the selected test.
3. if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it. If the target rate is not supported, it informs the tester that the selected rate is illegal otherwise it starts the test.

4. sets the selected simulated loop on the Loop Simulator through its serial port. This change in loop would change the existing loop condition and will force the SUT to adapt the rate according to the changed loop.

5. waits for the time set by the tester in the main GUI for allowing loop to by synced. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.

6. starts getting the DS & US object values of Loop Sync states and maximum achieved Speed by accessing the SUT MIB through SNMP. The SUT modules which are involved at this stage are Ethernet Driver, IP Stack and SNMP Agent

7. converts the speed into QAM and Bandwidth and by using the algorithmic function converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

8. compares the achieved DS/US speed with the "oracle" (target speed).

   If the achieved speed ≥ target speed

   Test is passed as SUT adapt the rate according to the changed loop. Mark the achieved speed Text Box at GUI or Cell at Excel Green

   Else

   Test is failed as SUT did not adapt the rate according to the changed loop. Mark the achieved speed Text Box at GUI or Cell at Excel Red

   End If

9. If more loops are selected, go to step 4 until all selected loops are finished.

TestCase 4 – Forward Error Correction (FEC):

(i) Introduction:

FEC is another SUT feature, which is responsible for correcting up to 8 error bytes within a 233-byte frame without re-transmitting the frame. This forward correction of frame can be translated into increased data communication performance. However, the SUT throughput would decrease if the loop was error free as FEC uses some control bytes on the actual frame. Its use justifies when the loop is noisy and thus the forward error correction reduces the retransmission of frames. FEC is applied in both upstream and downstream direction. In order for the FEC feature to operate properly, FEC should be ON at both DBIC and CPE end.

FEC testing is related to the data path software module. The LoopControl module is also responsible as the data path can only be established on the proper sync loop. To generate data, the automated test system builds the ICMP (ping) messages using the Windows DLL libraries. The test system verifies the data path through Ping messages. The MIB objects FEC corrected Frames and FEC uncorrected Frames at US and DS are responsible to check the functionality of FEC features. There is no oracle available for FEC against which the output of the test result can be compared and the result can be marked pass or fail. Result only displays the corrected FEC frames and uncorrected FEC frame against different loop along with the maximum speed achieved. By knowing the characteristic of each loop and the number of corrected/uncorrected frames against those loops, the tester/designer can determine that the FEC is functioning properly.

Davies et al [Dav95] proposed some tedious calculation to verify FEC. SUT firmware already calculates FEC, therefore, the test system uses this information and

displays it on the result. By seeing the FEC counts increasing or decreasing on the hard and easy loops respectively depicts that the FEC feature is working.

(ii) Operation Flow:

The automated test system:

1. initializes all parameters, checks the hardware phase and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 4, & target values (if not agree with the default target rate).
3. sets the FEC control MIB object value to "enable" immediately after the Test Case 4 option button is selected by the tester and Start the selected test.
4. if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it.
5. resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch; software modules release all resources, empty their message queues and all the counters including FEC reset to 0.
6. sets the selected simulated loop on the Loop Simulator through its serial port.
7. waits for the time set by the tester in the main GUI for allowing loop to be synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.

8. starts generating Ping to the remote PC connected with the CPE and confirms that data path is established. The SUT data path module is responsible for establishing the data path within the SUT.

9. abandons the test if pinging fails and generates the message that data path can not be established. If pinging succeeds, the remaining test is executed.

10. opens an FTP session and starts uploading & downloading the File

11. starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, corrected FEC, Uncorrected FEC, Error Power, Power Level, AGC by accessing the SUT MIB through SNMP. Although the Error Power, Power Level & AGC is not required for this test but since the output result GUI is same therefore these values are also displayed. Tester usually ignores these values for this test case. The SUT modules which are involves at this stage are Ethernet Driver, IP Stack and SNMP Agent

12. converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

13. writes DS & US speed, corrected FEC, uncorrected FEC, S/N ratio, Power Level and AGC in the output result.

14. if more loops are selected, go to step 5 until all selected loops are finished.

Test Case 5 – Performance Test during On-Hook:

(i) Introduction:

After verifying the hardware and software (loop & data path), the automated system checks the SUT throughput. It performs the same method which Computer Reseller News [Cmp98] did in their testing of SUT except that their test was manual whereas the automated test system does everything automatically. Computer Reseller News used the FTP to measure the DS and US performance of the ADSL products (SUT) and so does the automated test system. Data Communication magazine [3] also tested the performance of ADSL products including SUT. Their testbed was similar to the automated test system and Computer Reseller News testbed. However, instead of using FTP to generate the traffic, they used "BLAST" for raw data measurement and "loadrun" for application (HTTP) measurement.

Throughput calculation used by the Computer Reseller News is different from that of the automated test systems. This difference in calculation may appear in slightly different result. Computer Reseller News does not include the framing packet in its throughput calculation and they directly divide the size of the file with the time it takes to transfer this file. Automated test system complies the FTP-RFC 959 and includes the packet frame in its throughput calculation.

Automated test system uses the 5Meg file in DS & 1 Meg file US for FTP. The automated test system is connected at the DBIC end, shown in Figure 5.6, therefore, it uses the FTP "STOR" (equivalent to "Put") command for DS and FTP "RETR" (equivalent to "Get") command for US. Data path software module is mainly responsible in the performance testing. However, implicitly the LoopControl module is also responsible as data path can only be established on the synced loop.

(ii) Operation Flow:

The automated test system:

1. initializes all parameters, checks the hardware phase and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 5, & target values (if not agree with the default target rate) and Start the selected test.
3. if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it.
4. resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch, software modules release all resources and empty their message queues.
5. sets the selected simulated loop on the Loop Simulator through its serial port.
6. waits for the time set by the tester in the main GUI for allowing loop to be synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.
7. starts generating Ping to the remote PC connected with the CPE for confirming the establishment of data path. The SUT data path module is responsible for establishing the data path within the SUT and ICMP module is responsible for testing the Ping.
8. abandons the test if pinging fails and generates the message that data path can not be established. If pinging succeeded, the remaining test is executed.

9. opens an FTP session, start downloading the File & calculates the downstream throughput. The throughput calculation is shown in Appendix C, Figure C.4.

10. starts uploading the File & calculate the upstream throughput. The throughput calculation is the same as the downstream calculation and is shown in Appendix C, Figure C.4.

11. starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, corrected FEC, Uncorrected FEC, Error Power, Power Level, AGC by accessing the SUT MIB through SNMP. Although the Error Power, Power Level & AGC is not required for this test but since the output result GUI is same therefore these values are also displayed. Tester usually ignores these values for this test case. The SUT modules which participates at this stage are Ethernet Driver, IP Stack and SNMP Agent

12. converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

13. writes DS/US speed, corrected FEC, uncorrected FEC, S/N ratio, Power Level, AGC, and FTP throughput in the output result.

14. if more loops are selected, go to step 4 until all selected loops are finished.

Test Case 6 – Performance Test during Off-Hook:

(i) Introduction:

Test Case 1 and Test Case 2 operations are almost identical except that in the Test Case 2, the loops are Off-Hook. Similarly, Test Case 5 and Test Case 6 operations are almost identical except that in the Test Case 6, the loops are Off-Hook. Test Case 5 measures the throughput during On-Hook in which some loops are sync at better rate than the loop sync during Off-Hook. This reduction in sync speed also reflects reduction in throughput. The result of Test Case 2 and Test Case 6 is very important for the next Test Case "Affect of data and voice (include ringing) on the achieved rate & throughput".

(ii) Operation Flow:

The automated test system:

1. initializes all parameters, checks the hardware phase and sets the default target rate.
2. opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 6.
3. sets the off-hook control value "enable" in the appropriate MIB through SNMP and sets the default target rate for off-hook control immediately after which the tester selects the Test Case 6 option button.
4. allows the tester to change the default target value (if not agree with the default target rate) and Start the selected test.
5. if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it.
6. resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch, software modules release all resources and empty their message queues.

7. sets the selected simulated loop on the Loop Simulator through its serial port.

8. waits for the time set by the tester in the main GUI for allowing loop to be synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.

9. makes the line off-hook by sending the off-hook commands to the dialup modem, which is connected to its serial port.

10. allows the same time for syncing up the loop which is given in step 6.

11. starts generating Ping to the remote PC connected with the CPE for confirming the data path. The SUT data path module is responsible for establishing the data path within the SUT and ICMP module is responsible for testing the Ping.

12. abandons the test if pinging fails and generates the message that data path can not be established. If pinging succeeds, the remaining test is executed.

13. opens an FTP session, starts downloading the File & calculates the downstream throughput.

14. starts uploading the File & calculates the upstream throughput. The throughput calculation is the same as the downstream calculation.

15. starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, corrected FEC, Uncorrected FEC, Error Power, Power Level, AGC by accessing the SUT MIB through SNMP. Although the Error Power, Power Level & AGC is not required for this test but since the output result GUI is same therefore

these values are also displayed. Tester usually ignores these values for this test case. The SUT modules which are involved at this stage are Ethernet Driver, IP Stack and SNMP Agent.

16. makes the line on-hook by sending the on-hook commands to its serially connected dialup modem.

17. converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

18. writes DS/US speed, corrected FEC, uncorrected FEC, S/N ratio, Power Level, AGC, DS/US FTP throughput in the output result.

19. if more loops are selected, go to step 4 until all selected loops are finished.

Test Case 7 - Affect of data & voice (include ringing) on the Achieved Rate & on the Performance:

(i) Introduction:

This is the last Test Case of the Software testing. This test case is important because of its closeness to the actual environment. In this test case, data and voice operate simultaneously. As the name implies, this test case tests the affect of data and voice on the:

(i) achieved rate or loop reachable for different loops

(ii) SUT performance (FTP throughput) on different loops

In this test case additional test equipment, Call Generator, is used. The call generator has its own GUI through which the Call Generator is configured. The Call

Generator is programmed manually and is required to be started at the beginning before the test case starts. The Call Generator simulates the voice call and works in the following way.

The Calling (CO) site Call-Generator module	**The Called (CPE) site Call Generator module**
• makes the line off-hook • dials the called site telephone number • generates the ringer	
	• receives the ringer • answers the call by going off-hook
• stops the ringer • starts generating & receiving messages	
	• starts generating & receiving messages
• drops the call • goes on-hook for pre programmed time	
	• drops the call

Data is generated through FTP using the same 5Meg file in the downstream direction and 1Meg file in the upstream direction.

(ii) Operation Flow:

Program the Call Generator to generate calls for 25 Sec off-hook and 25 Sec on-hook.

Start the Call Generator.

The automated test system:

1 initializes all parameters, checks the hardware phase and sets the default target rate.

2 opens the main GUI and allows the tester to select the Slot #, Loops, Test Case 7, & target values (if not agree with the default target rate) and Start the selected test.

3 if finds that Slot#, Loops and Test Case have not been selected, it intimates the tester to select it.

4 resets the SUT (by resetting the CPE & DBIC) by setting the reset value of the appropriate MIB objects using SNMP. The resetting would cause the system start from scratch, software modules release all resources and empty their message queues.

5 sets the selected simulated loop on the Loop Simulator through its serial port.

6 waits for the time set by the tester in the main GUI for allowing loop to be synced. The tester usually gives less time for syncing the easy loops and more time for syncing the difficult loops. LoopConfig Process, Line Interface and Line Monitor modules involve in syncing up loop.

7 starts generating Ping to the remote PC connected with the CPE for confirming the data path. The SUT data path module is responsible for establishing the data path within the SUT and ICMP module is responsible for testing the Ping.

8 abandons the test if pinging fails and generates the message that data path can not be established. If pinging succeeds, the remaining test is executed.

9 opens an FTP session, starts downloading the File & calculates the downstream throughput.

10 starts uploading the File & calculates the upstream throughput. The throughput calculation is the same as the downstream calculation.

11 starts getting the DS & US object values of Loop Sync states, maximum achieved Speed, corrected FEC, Uncorrected FEC, Error Power, Power Level, AGC by

accessing the SUT MIB through SNMP. The SUT modules which, participates at this stage are Ethernet Driver, IP Stack and SNMP Agent.

12 converts the speed into QAM and Bandwidth and by using the algorithmic function, converts the Error Power value into the decibel (db) value of Signal to Noise (S/N) ratio.

13 writes DS/US speed, corrected FEC, uncorrected FEC, S/N ratio, Power Level, AGC, DS/US FTP throughput in the output result.

14 if more loops are selected, go to step 4 until all selected loops are finished.

CHAPTER 6
CONCLUSION

6.1 SUMMARY

This chapter concludes my research work and discusses possible future enhancement. This dissertation was set out to verify the hardware and software development domain of the ADSL product (Nortel 1-Meg Modem) by automating diagnosis. The automated test system diagnoses the problem by verifying the deviation of application's implementation from its specification. The black-box testing has been proposed for the automated test system.

By automating the verification process, the specification / implementation / test iterative loop has been facilitated. This makes it easier for the designer / developer to ensure that the specification documentation reflects the implementation, and vise versa. This reduces the bugs and helps ensure that the documentation is up to date.

The automated test system application presented the PC based testing solutions for the complex ADSL system. The application has provided a robust, stable automated testing platform with a common vehicle which can perform attended and unattended feature regression testing. The PC based automated system has proven to be highly portable across organizations due to the PC's open system architecture and VB's powerful graphical user interface.

It has been proved that the system is more efficient than manual regression and feature testing. Therefore, it has resulted in increased number of tests reflecting quality. The system is capable of performing result evaluation and the test results can be directly

exported to the Excel sheet without requiring file conversion. The consistent level of sensitivity to test results and the SUT environment has proven to be a valuable aid in early problem detection. The automated system has greatly expanded the organization test coverage without a proportional increase in staff level.

Automation has resulted in many productivity gains. The robustness of the system and the consistent repeatable correctness of the test have resulted in diminished labor intensive task. It has freed the testing community from the bulk of the regression testing duties so that they can perform more creative and free form testing, translating into greater quality of the product.

The automated test system is designed with many useful tools. It uses the MSComm ActiveX control to communicate with the serial debug port of SUT for hardware testing and for capturing the SUT's software traces during software testing. Its Manager SNMP-API interfaces it with the WinSNMP agent and allows it to access SUTs' MIBs during software testing. FTP Client API helps in testing the performance of MIB by uploading and downloading file on each loop and calculating the time (throughput) for transferring the file. The application interfaces with ICMP.DLL to check the datapath of the SUT. The CommonDialog ActiveX control Open and Save file in a similar way as the Microsoft Word and other Windows application. Due to this feature the files names are not required to be hard coded and the tester can save file with any name, in any directory and is also capable of loading file from any directory.

6.2 FUTURE ENHANCEMENT

This section looks at some future enhancements that could be made to the automated test system. Although the automated test system working is flawless in the development, verification and production lab, further enhancements would facilitate the specification / implementation cycle in an iterative development process. Enhancement areas are:

(i) Currently the application is tightly integrated with one set of specification. Its scope can be extended by making the system flexible to include other specifications. Record and playback could be added to the system to make it flexible. During the testing phase, the user's keystrokes, mouse movements and the output of the application could be recorded. Later, regression tests can be run automatically by playing back user's recorded interactions and comparing the application's output against the original recording.

(ii) Due to cost association with the numbers of loop simulators, the automated test system test one loop line at a time. The infrastructure is ready to test multiple loop line simultaneously by making small changes in the code. Currently the interface with the loop simulator is through the serial interface. To control many loop simulators through serial ports for multiple loop lines testing is not a proper choice because many serial ports are required to control each loop simulator which are inherently slow, thereby causing unnecessary delay. However, a more effective way is to use the GPIB (IEEE 488) port. In this configuration the system can control many loop simulator connected in daisy chain by one GPIB port. From the coding point of view, the serial port communication module will be replaced with GPIB port communication module.

Several GPIB's APIs are available so it will not be difficult to replace the serial port communication module with GPIBs communication modules. In the main GUI of software testing, under the slot# frame, options buttons are provided for each slot (loop line). For multiple loop lines testing these options buttons will be replaced with the check boxes so that more slots (loop lines) can be selected. "'All loop" check box has been grayed out. This can be enabled to select all slots at once.

(iii) For very few test cases, the automated test act as a semi automated test system. To make it fully automated test system, the system has to interface with some external test equipment. In test case 7, the call generator, Crescendo, is controlled manually by using its own GUI. The automated test system can be upgraded to control the call generator by itself. By integrating the Call generator control in the automated test system, the call generation time can be changed and this would provide a more thorough voice and data and voice alone tests. The automated test system can integrate the call generator utility by either using the call generator API or the telnet utility.

Similarly in test case 5, 6 and 7, the SUT performance is measured through the FTP tool. FTP is real application therefore being the proper tool to measure SUT performance. However, for stressing the data, data generator, SmartBit, is used. By using the SmartBit available API or by doing telnet into the SmartBit, the automated test system can control the SmartBit and therefore can generate data at different rate. The integration of SmartBit into the automated test system will help the automated test system to test the SUT's stress test.

The ADSL forum proposed the Physical Layer Conformance and Protocol Conformance. The automated test system did not integrate the Physical Layer

Conformance test as the Nortel DMS telephone switch for which the SUT is part of has the ability to test physical layer tests. By interfacing with the DMS test suites, many physical as well as voice related test can be performed. Again by either through API of DMS or telnet into the DMS, will help the automated test system to perform physical layer as well voice related tests.

(iv) The automated test system has the ability to support all levels of SNMP communications. The automated test system can be upgraded to act as a SNMP browser by enabling it to compile MIB. This will allow the system to be more flexible as any MIB can be browsed at any given time.

(v) The automated system GUI is very dense especially the Software GUI. The automated system application is a PC based application running on Windows platform. All the windows application have the menu bar and menu options. By adding the menu bar and menu options in the GUI, will make the application appearance more professional and Windows flavor. However, this option will not increase in test capability. This would be more aesthetic type work on the GUI.

(vi) The last proposed enhancement is to add the help feature and expert adviser. In the help feature, the procedure of starting up the application, selecting any options, loading and saving files, the manual of APIs, etc can be included.

The expert adviser is more than the help feature. It is another extensive piece of research which may require another thesis. The expert adviser would analyze problems experienced by automated system. It would try to give the solution by highlighting the areas causing problems and the procedure to check those areas. For example, if the FTP fails, the expert adviser may ask to check the following areas:

- ip address of the FTP server;
- data path (both upstream and downstream) to that FTP server;
- path and the file name of the file which is being transferred.

Bibilography

[Adr82] W. R. Adrion, M. A. Branstad and J. C. Cherniavsky, ***Validation, Verification and Testing of Computer Software***, ACM Computing Surveys, pp. 159-192, June 1982.

[ADS98] ***Working Text 27 – Overview of ADSL Testing***, ADSL forum Testing & Interoperability Working group, June 19, 1998.

[ADS99-1] ***ADSL Tutorial***, http://www.adsl.com/adsl_tutorial.html.

[ADS99-2] ***General Introduction to Copper Access Technologies***, http://www.adsl.com/general_tutorial.html.

[Alg94] C. J. Alguire, ***On-line Dynamic Views of ApplicationsWritten in Smalltalk***, M.C.S. Thesis. Ottawa, Ontario: School of Computer Science, Carleton University, 1994.

[Alv89] A. M. Alvarez and N. V. Telindus, ***Echo canceller design and implication of a CCITT V32 modem: software solution***, Acoustics, Speech, and Signal Processing, 1989. ICASSP-89, Vol.2, May 1989, pp 1384-1387.

[Ame98] ***CRS-A Analog Load Generator Instruction Manual***, Ameritec Corporation, California, 1998.

[Ann82] M. A. Annarartone and M. G. Sami, ***An approach to functional testing of microprocessors***, Proc. 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA, pp. 158-164, June 1982.

[Bag97] A. Baginski and G. Covarrubias, ***Open Control – The Standard For PC-Based Automation Technology***, Proceedings. 1997 IEEE International Workshop on Factory Communication Systems, 1997, pp. 329 - 333. Conference held in Oct 1-3, 1997. IEEE Catalog Number 97TH8313

[Bar91] M. Barton, ***On the performance of an asymetrical digital subscriber lines QAM transceiver***, Global Telecommunications Conference, 1991. GLOBECOM'91, Vol.3, December 1991, pp 2002-2006.

[Ber 97] Philip A. Bernstein,***The Microsoft Repository***,conference Athens,Greece,1997. http://msdn.microsoft.com/library/backgrnd/msdn_repositwp.htm.

[Berg95] Ken Bergmann, ***Client/Server Solutions: Coding Guideline***, Sep 29,1999. http://msdn.microsoft.com/library/techart/msdn_csfdcode.htm

[Berg95-1] Ken Bergmann,***Client/Server Solutions:The Basics***, Sep 29,1999. http://msdn.microsoft.com/library/techart/msdn_csfdidea.htm

[Berg95-2] Ken Bergmann,***Client/Server Solutions:The Architecture Process***, Sep 29,1999. http://msdn.microsoft.com/library/techart/msdn_csarctop.htm

[Bil97] C. E. Billing, ***Aviation Automation: The search for a Human-Centered Approach***, Mahwah, NJ: Erlbaum, 1997.

[Bin96] R. Binder, ***Testing Object-Oriented Software: a Survey***, Software Testing, Verification and Reliability, Vol. 6, pp. 87-110, 1996.

[Boc98] J. Bock, ***Visual Basic 6 WIN32 API Tutorial***, Wrox Press Ltd., Birmingham, 1998

[Bon94] G. W. Bond, ***Logic Programs for Consistency-Based Diagnosis***, Ph.D. Thesis, Carleton University, Canada, August 1994.

[Boo99] C. Booth, ***ADSL Discounts Have Far-Reaching Implications***, Data Communications, January 1999. http://www.data.com/story/DCM19990601S0001

[Bre80] M. A. Breuer and A. D. Friedman, ***Functional level primitives in test generation***, IEEE Transactions on Computers, Vol. c-29, No. 3, pp. 223-235, March 1980.

[Cat98] ***An Introduction to TCP/IP Programming with Socket Wrench***, Catalyst Development Corporation, 1998. http://www.catalyst.com

[Che94] W. Y. Chen and D. L. Waring, ***ADSL Noise Environment and Potential System Performance***, Communication, 1994. ICC'94, SUPERCOMM/ICC'94, IEEE International Conference, Vol.1, 1994, pp. 451 – 454. Conference held in May 1-5, 1994. IEEE Catalog Number 94CH3403-3.

[CMP98] ***CRN Test Center***, Computer Reseller News, July 6, 1998. . http://www.infoexpress.com/reviewtracker/reprints.asp?page_id=114

[Con99] ***Operating Manual DLS 400 Wire line Simulator***, Consultronics Inc., Revision 9, 1999.

[Coo92] D. E. Cooke, ***Issues Surrounding Specification Languages For Software Automation***, Computer-Aided Software Engineering, 1992. Proceedings., Fifth International Workshop, July 1992, pp 120-123.

[Cor97] Gary Cornell and David Jezak, ***Matching developer skillsets and application requirements with the appropriate data access interface***, February 1997. http://msdn.microsoft.com/vbasic/technical/articles/create/default.asp.

[Cow97] S. Cowan, ***Automated White-Box Cluster Testing Using Message Sequence Charts'96***, Master's Thesis, Carleton University, Canada, December 1997.

[Dah98] K. Dahlquist and R. Giusto, ***Beyond Today's Boundaries: The Evolving Portable PC Platform***, International Data Corporation, IDC report #17161, October 1998.

[Dah72] O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, ***Structured Programming***, Academic Press, 1972.

[Dao98] F. H. Daou, ***Overview of ADSL test requirement towards conformance, performance and Interoperability***, "AUTOTESTCON '98. IEEE Systems Readiness Technology Conference, 1998, pp. 413 - 420. Conference held in Aug 24-27, 1998. IEEE Catalog Number 98CH36179.

[Das97] S. R. Das, ***Design and Testing of Reliable Digital System***, Course Notes – ELG 5194, 1997.

[Dav95] W. S. Davies and M. S. Peacock, ***Simple test of DMT ADSL interleaved FEC Provisions***, Electronics Letters, 1995, Vol. 31 (25), pp. 2152 – 2153

[DLL-1] ***"What are DLLs?"*** http://solo.abac.com/dllarchive/define.html

[Doo94] R. K. Doonga and P. G. Frankl, ***The ASTOOT approach to test object-oriented programs***, ACM Translations on Software Engineering and Methodology, Vol. 3, No. 4, pp 101-130.

[Exc98] ***Attach Custom Toolbars to Microsoft Excel Workbooks and Templates***. March 6,1998. http://www.microsoft.com/exceldev/articles/toolbatt.htm.

[Get99] Ken Getz and Mike Gilbert, ***Advance Class Module***, April8.1999. http://www.microsoft.com/officedev/Articles/AdvC1Mod.htm.

[Gla79] R. L. Glass, ***Software Reliability Guidebook***, Prentice-Hall, 1979.

[Gor98] W. Goralski, ***ADSL and DSL Technologies***, McGraw-Hill, New York, 1998

[Gor99] W. Goralski, ***xDSL loop qualification and testing***, IEEE Communications Magazine, Vol. 37, Issue 5, May 1999.

[Haw97] G. T. Hawley, ***System Consideration for the Use of xDSL Technology for Data Access***, IEEE Communication Magzine, March 1997.

[Hen84] M. A. Hennell, D. Hedley and I. J. Riddell, ***Assessing a Class of Software Tools***, In Proceeding of the IEEE 7th International Conference on Software Engineering, IEEE, pp. 266-277, March 1984.

[Het85] W. C. Hetzel, ***The Complete Guide to Software Testing***, Collins, 1985.

[Hsi82] S. Y. H. Su and Y. I. Hsieh, ***Testing functional faults in digital systems described by register transfer language***, Journal of Digital Systems, Vol. 6, No. 2, pp. 161-183, 1982.

[Hyj97] J. Hyjek, ***Automation of Object Behavior Validation***, Master's Thesis, Carleton University, Canada, November 1997.

[IC&97] Ref: ***Probe-Advance Connect***, Elmshorn, Germany: IC&C GmbH, 1997.

[Inf98] ***Nortel's topology simplest of all***, http://www.infoexpress.com/reviewtracker/reprints.asp?page_id=114

[Int96] ***CAP standardization for ADSL ignites passionate debate***, Internet Telephony, July 1996.

[IRW98] A. Irwin, ***Standard Software for Automated Testing of Infrared Imagers, IRWindows™ in Practical Applications***, AUTOTESTCON '98. IEEE System Readiness Technology Conference, 1998, pp. 561 – 571. Conference held in August 24-27, 1998. IEEE Catalog Number 98CH36179.

[Isf98] A. Isfan, ***1 Meg Modem Phase II Integration Test Plan***, Dept 5H52, Nortel Networks, 1998. (Not available externally).

[Jac97] Reed Jacobson, ***Using Dialog Box Controls on a Worksheet***, April 21,1997. http://www.microsoft.com/ExcelDev/Articles/exsxsc10.htm.

[JAI86] R. Jaikumar, ***Post Industrial Manufacturing***, Harvard Business Review, November_December, 1986, pp 69-76.

[Kir96] R. H. Kirby, ***Results of performance simulations of various ADSL configurations***, The Institution of Electrical Engineers, 1994.

[Lai81] K. W. Lai, ***Functional testing of digital systems***, Ph. D thesis, Computer Science Dept., Carnegie-Mellon University, Dec. 1981.

[Lev82] Y. H. Levendal and P. Mennon, ***Test generation algorithms for computer hardware description language***, IEEE Transaction on Computers, Vol. C-31, No. 7, pp. 577-588, July 1982.

[Lin88] N. Lin, J. Tzeng, ***Full-duplex data over local loops***, IEEE Communication Magazine, Vol. 26, Feb. 1988, pp 31-42.

[Lin98-1] ***Line Monitor High Level Design***, Nortel Networks, Issue 0.5, April 1998. (Not available externally)

[Lin98-2] ***Line Interface High Level Design***, Nortel Networks, Issue 0.5, April 1998. (Not available externally)

[Loo98] ***Loop Configurator Design Document***, Nortel Networks, Issue 1.4, November 1998. (Not available externally).

[Luo] Gang Luo, Rachida Dssouli, Test Generation For The Distributed Test Architecture.

[MAL94] S. L. Malaney, ***DARTS: An Automated Feature Test System FOR A Digital Central Office Switching System***, Proceedings from Third Symposium on Assessment of Quality Software Development Tools, 1994, pp. 112 - 124. Conference held in June 7-9, 1994. IEEE Catalog Number 94TH0635-3.

[MEO] ***Microsoft Excel Object***
http://www.microsoft.com/officeDev/Articles/OPG/004/004.htm

[Min98] D. Minoli, ***xDSL Technology and Application***,
http://47.2.3.239/DataproHTML/51534-1.htm

[Mit96] C. M. Mitchell, ***GT-MSOCC: Operator models, model-based displays, and intelligent aiding***, Human/technology interaction in complex system, Vol.8, pp. 67-172, Greenwitch, CT: JAI Press Inc., 1996.

[MIT98] T. K. Mitra, ***Scada in Automated Test and Measurement***, Power Quality '98, 1998, pp. 35 – 40. Conference held in1998.

[Mye79] G. J. Myers, ***The Art of Software Testing***, Wiley, 1979.

[Nal98] S. Naldrett, ***Datapath High Leven Design***, Nortel Networks, Issue 1.1, August 1998. (Not available externally).

[Nat95] B.Natale, ***An Open Interface for Programming Network Management Applications using the Simple Network Management Protocol under Microsoft Windows, WinSNMP/Manager API***, Windows SNMP version 1.1a, August 1995.

[Net99] ***A Tutorial for Evaluating the Performance of ADSL Networks using Netcom system Smartbits Performance Analyzer and Software***, www.netcomsystems.com.

[New98] D. Newman, M. Carter, and H. Holzbaur, ***DSL: Worth Its Wait***, Data Communications, June 1998. http://www.data.com/lab_tests/wait.html

[Nor99-1] ***1-Meg Modem Service***, www1.nortelnetworks.com, pcn,1megmodem

[Nor99-2] ***1-Meg Modem: Next Generation Data Access***, March 1999. http://www1.nortelnetworks.com/pcn/1megmodem/intro/home.html

[Obj97] Ref: ***Together/Professional. Stuttgart***, Germany: Object International Software, 1997.

[Ole95] ***OLE Remote Automation: Synopsis And Analysis***, December 18,1995. http://msdn.microsoft.com/vbasic/technical/articles/remauto/default.asp.

[Oul86] M. A. Ould and C. Unwin, ***Testing in Software Development***, Cambridge University Press, New York, 1986.

[Par96] R. Parasuraman and M. Mouloua, ***Autimation and Human Performance***, Mahwah, NJ: Erlbaum, 1996.

[Pat97] D. Patterson, ***XNET Test & Integration Plan***, Dept 5H52, Nortel Networks, 1996. (Not available externally).

[Pat98] D. Patterson, ***1Meg Modem Product Specification***, Issue 01.5, March 8, 1999.

[Pat99] Ted Pattison, ***Understanding Interface-based programming***, January 1999. http://msdn.microsoft.com/library/techart/IfaceBased.htm.

[Per95] W.Perry, ***Effective Methods for Software Testing***, A Wiley-QED Publication, New York, 1995.

[Per97] G. Perry, ***teach yourself Visual Basic 5 in 24 hours***, Sams Publishing, Indiana, 1997.

[Pet97] E. Petroutsos, ***Mastering Visual Basic 5***, Sybex Inc, California, 1997.

[PIL88] D. Pillai, ***Development A Decision Support System For Optimizing Automated Wafer Fabrication***, Fifth IEEE/CHMT International on Design-to-Manufacturing Transfer Cycle, Electronic Manufacturing Symposium, 1988, pp. 170 – 176. Conference held in Oct. 10-12, 1988.

[Pos96] R. M. Poston, ***Automating Specification-Based Software Testing***, IEEE Computer Society Press, California, 1996.

[Qua96] J. A. Quayle and M. Heath, ***Development of the Access Network***, The Institution of Electrical Engineers, 1996.

[Raj94] Jeffery A. Rajhel, ***The Application of an Automated Tool for Modeling Test Processes***, IEEE 1994.

[Riv98] D. Rivaud, ***Loop Characterization and Register Status***, Dept 5H52, Nortel Networks, April 1998. (Not available externally).

[Riv99] D. Rivaud, ***1MM QAM DSP***, Dept 5H52, Nortel Networks, March 1999. (Not available externally).

[Rob80] C. Robach and G. Saucier, ***Microprocessor functional testing***, Digest of papers, 1980 International Test Conference, pp 433-443, 1980.

[RON95] Ronald E. Howland, ***Computer Hardware Diagnostics For Engineers***, McGraw-Hill Inc, New York, 1995.

[Rop94] M. Roper, ***Software Testing***, Mcgraw-Hill Book Company, New York, 1994.

[SHA85] H. Shaiken, ***The automated factory: The view from the shop floor***, Technology Review, January, 1985, pp 17-25.

[Smith] Jeff Smith, ***A Developer's Perspective;A comparision of Powersoft Powerbuilder 5.0 and microsoft Visual Basic 5.0.*** http://msdn.microsoft.com/vbasic/t...1/articles/vbvspwrbldr/default.asp

[Soc98] ***Windows socket Programming***, http://www.goodnet.com/~esnible/winsock.html

[Som92] I. Sommerville, ***Software Engineering***, Addision-Wesley, ISBN 0-201-56529-3, 1992.

[Sum99] C. K. Summers, ***ADSL Standards, Implementation, and Architecture***, CRC Press, New York, 1999.

[Sur99] J. Surveyer, ***Java and Visual Basic: programming's new breed***, Canada Computer Paper Inc., June 1999.

[Sye98] T. Syed, ***1-Meg Modem Verification Test Plan***, Wireline Access, Nortel Networks, 1998. (Not available externally).

[Tay99M] M. Taylor, ***DSL slashes dial-up bottleneck***, EE Times, Issue 1066, June21, 1999, pp 142–156.

[Tay99] T. Taylor, ***1 Meg Modem Loop Startup, Rate Adaptation, and Off/On Hook Handling***, *Nortel Networks, December 1998.* (Not available externally).

[TCP97] ***PowerTCP Development Guide***, Dartcom Incorporated, New York, 1997.

[Tha80] S. M. Thatte and J. A. Abraham, ***Test generation for microprocessors***, IEEE Transactions on Computers, Vol. c-29, No. 6, pp. 429-441, June 1980.

[Thu94] D. A. Thurman and C. M. Mitchell, ***A methodology for the design of interactive monitoring interfaces***, In Proceeding of the 1994 IEEE International Conference on Systems, Man, Cybernetics, San Antonio, Tx, 1994.

[Thu95] D. A. Thurman and C. M. Mitchell, ***A design methodology for operator displays of highly automated supervisory control systems***, In Proceedings of the 6th IFAC/IFIP/IFOR/SEA Symposium on Analysis, Design, and Evaluation of Man Machine Systems, Boston, MA, 1995.

[Thu97] D. A. Thurman, D. M. Brann and C. M. Mitchell, ***An Architecture To Support Incremental Automation Of Complex System***, IEEE Transactions on System, 1997.

[Tol95] K. Tolly and D. Newman, ***TCP/IP Applications: Easy Does It***, Data Communications, February 1995. http://www.data.com/lab_tests/tcp-ip_applications.html

[TR998] ***Channelization for DMT and CAP ADSL Line Codes: Packet Mode***, ADSL Forum Packet Mode Technical Report DRAFT, February 1998.

[Val94] C. F. Vallenti and K. Kerpez, ***Analysis of wideband noise measurement and implications for signal processing in ADSL systems***, Communication, 994. ICC'94, SUPERCOMM/ICC'94, IEEE International Conference, Vol.2, 1994, pp. 826 – 832. Conference held in May 1-5, 1994. IEEE Catalog Number 94CH3403-3.

[Vau96] William R. Vaughn, ***Remote Server Connectivity***, February 1996. http://msdn.microsoft.com/vbasic/technical/articles/remote/default.asp.

[Vis98] ***The New Stuff in Microsoft Visual Basic 6.0***, 1998.

[Vis98-1] ***The New Stuff in Microsoft Visual Basic 6.0***, 1998 http://www.appdev.com/free98/freestuff.htm.

[WAR85] H. J. Warnecke & R. Steinhilper, ***Flexible Manufacturing***, IFS Publication Ltd., UK.

[Wel98] Eric Wells, ***Advanced Spreadsheet Programming With Microsoft Excel 97***. September 18,1998. http://www.microsoft.com/exceldev/articles/movs104.htm.

[Wil98] Mike Wills,***Visual Basic Live Techniques***,december1998 http://msdn.microsoft.com/library/techart/Vblive.htm.

[Zie99] P. Ziemann, ***ADSL line qualification tests***, Wandel & Goltermann, Application Note 52, 1999.

[Zsa97] C. E. Zsambok and G. Klein, ***Naturalistic Decision Making***, Mahwah, NJ: Erlbaum, 1997.

APPENDIX A

TEST LOOPS DESCRIPTION

The following list represents a series of test loops to be used to qualify the 1 Meg modem system operation. Straight and ANSI test loops have been included to get the maximum allowed length in the none, one, or two bridged taps conditions. All loops are defined starting from the CO and moving towards the subscriber.

1- VARIABLE _26_AWG 1 Kft : 1000ft 26 AWG

2- VARIABLE_26_AWG 2 Kft: : 2000ft 26 AWG

3- VARIABLE_26_AWG 3 Kft: : 3000ft 26 AWG

4- VARIABLE_26_AWG 4 Kft: : 4000ft 26 AWG

5- VARIABLE_26_AWG 5 Kft: : 5000ft 26 AWG

6- VARIABLE_26_AWG 6 Kft: : 6000ft 26 AWG

7- VARIABLE_26_AWG 7 Kft: : 7000ft 26 AWG

8- VARIABLE_26_AWG 8 Kft: : 8000ft 26 AWG

9- VARIABLE_26_AWG 9 Kft: : 9000ft 26 AWG

10- VARIABLE_26_AWG 10 Kft: : 10000ft 26 AWG

11- VARIABLE_26_AWG 11 Kft: : 11000ft 26 AWG

12- VARIABLE_26_AWG 12 Kft: : 12000ft 26 AWG

13- VARIABLE_26_AWG 13 Kft: : 13000ft 26 AWG

14- VARIABLE_26_AWG 14 Kft: : 14000ft 26 AWG

15- VARIABLE_26_AWG 15 Kft: : 15000ft 26 AWG

16- ANSI 2 :13500ft 26 AWG + 3000ft 24 AWG + 150ft BT 24 AWG

17- ANSI 3 : 7500ft 26 AWG + 6000ft 24 AWG + 500ft BT 24 AWG + 1500ft 24 AWG + 1000ft BT 22 AWG + 1000ft 22 AWG + 1500ft BT 24 AWG + 500ft BT on BT 24 AWG + 1000ft BT on BT 24 AWG

18- ANSI 4 : 7500ft 26 AWG + 4500ft 24 AWG + 2000ft 22 AWG + 3000ft 26 AWG

19- ANSI 5 : 9000ft 26 AWG + 6000ft 24 AWG + 1500ft BT 26 AWG

20- ANSI 6 : 4500ft 26 AWG + 12000ft 24 AWG + 500ft BT 24 AWG + 1000ft 24 AWG + 500ft BT 24 AWG

21- ANSI 7 : 13500ft 26 AWG

22- ANSI 8 : 9000ft 24 AWG + 1000ft 22 AWG + 6000ft 26 AWG + 1000ft BT 24 AWG

23- ANSI 9 : 3000ft 26 AWG + 1500ft BT 26 AWG + 6000ft 26 AWG + 1500ft BT 26 AWG + 1500ft 26 AWG + 1500ft BT 26 AWG

24- ANSI 11 : 2000ft 26 AWG + 1500ft BT 26 AWG

25- ANSI 12 : 7500ft 26 AWG + 4500ft 24 AWG + 1500ft 26 AWG

26- ANSI 13 : 9000ft 26 AWG + 2000ft 24 AWG + 1500ft BT 26 AWG + 500ft 24 AWG + 1500ft BT 26 AWG + 500ft 24 AWG

27- ANSI 15 : 12000ft 26 AWG

APPENDIX B

HARDWARE TESTING - TEST PATTERNS ALGORITHM

```
Public Function Execute_test-pattern0 () As String
        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
          ' Write 00 0000 in the register address intLoop
           Write_Data(intLoop) = 00 0000
          ' Read 00 0000 from the register address intLoop
           Read_Data(intLoop)
          ' Compare write value with the read value
          If (Write_Data(intLoop) = Read_Data(intLoop)) Then
              ' Test passed
              Execute_test-pattern0 = "Passed"
          Else
              ' Test Failed
              Execute_test-pattern0 = "Failed"
          End If
        Next intLoop
```

Figure B.1: **Test Pattern-0**

```
Public Function Execute_test-pattern-3F () As String
        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
          ' Write 11 1111 in the register address intLoop
           Write_Data(intLoop) = 11 1111
          ' Read 11 1111 from the register address intLoop
           Read_Data(intLoop)
          ' Compare write value with the read value
          If (Write_Data(intLoop) = Read_Data(intLoop)) Then
              ' Test passed
              Execute_test-pattern-3F = "Passed"
          Else
              ' Test Failed
              Execute_test-pattern-3F = "Failed"
          End If
        Next intLoop
```

Figure B.2: **Test Pattern-3F**

```
Public Function Execute_test-pattern-2A () As String
        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
          ' Write 10 1010 in the register address intLoop
           Write_Data(intLoop) = 10 1010
          ' Read 10 1010 from the register address intLoop
           Read_Data(intLoop)
           ' Compare write value with the read value
           If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                ' Test passed
                Execute_test-pattern-2A = "Passed"
           Else
                ' Test Failed
                Execute_test-pattern-2A = "Failed"
           End If
        Next intLoop
```

Figure B.3: **Test Pattern-2A**

```
Public Function Execute_test-pattern-15 () As String
        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
          ' Write 01 0101 in the register address intLoop
           Write_Data(intLoop) = 01 0101
          ' Read 01 0101 from the register address intLoop
           Read_Data(intLoop)
           ' Compare write value with the read value
           If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                ' Test passed
                Execute_test-pattern-15 = "Passed"
           Else
                ' Test Failed
                Execute_test-pattern-15 = "Failed"
           End If
        Next intLoop
```

Figure B.4: **Test Pattern-15**

```
Public Function Execute_marching-test () As String
        ' Fill zeros in all registers
        For intLoop = intFirstRegister To intLastRegister
          ' Write 00 0000 in the register address intLoop
           Write_Data(intLoop) = 00 0000
        Next intLoop              ' End For Loop
        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
          ' Read 00 0000 from the register address intLoop
           Read_Data(intLoop)
           ' Compare write value (Hex 0) with the read value (Hex 0)
           If (Write_Data(intLoop) <> Read_Data(intLoop)) Then
                ' Test Failed
                Execute_marching-test = "Failed"
           Else
                ' Write 11 1111 in the register address intLoop
                Write_Data(intLoop) = 11 1111
                ' Read 11 1111 from the register address intLoop
                Read_Data(intLoop)
                 ' Compare write value (Hex 1) with the read value (Hex 1)
                If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                        ' Test Passed
                        Execute_marching-test = "Passed"
                Else
                        ' Test Failed
                        Execute_marching-test = "Failed"
                End If
           End If
        Next intLoop ' End Ascending Loop
        ' Set a loop from last register to first register (descending order)
        For intLoop = intLastRegister To intFirstRegister Step -1
          ' Read 11 1111 from the register address intLoop
           Read_Data(intLoop)
           ' Compare write value (Hex 1) with the read value (Hex 1)
           If (Write_Data(intLoop) <> Read_Data(intLoop)) Then
                ' Test Failed
                Execute_marching-test = "Failed"
           Else
                ' Write 00 0000 in the register address intLoop
                Write_Data(intLoop) = 00 0000
                ' Read 00 0000 from the register address intLoop
                Read_Data(intLoop)
                 ' Compare write value (Hex 0) with the read value (Hex 0)
                If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                        Execute_marching-test = "Passed" ' Test Passed
                Else
                        Execute_marching-test = "Failed"  ' Test Failed
                End If
           End If
        Next intLoop ' End Descending Loop
```

Figure B.5: **Marching Test**

```
Public Function galloping-test () As String
        ' Fill zeros in all registers
        For intLoop = intFirstRegister To intLastRegister
            Write_Data(intLoop) = 00 0000    ' Write 00 0000 in the register address intLoop
        Next intLoop                         ' End For Loop

        ' Set a loop from first register to last register
        For intLoop = intFirstRegister To intLastRegister
            Read_Data(intLoop)               ' Read 00 0000 from the register address intLoop

            ' Compare write value (Hex 0) with the read value (Hex 0)
            If (Write_Data(intLoop) <> Read_Data(intLoop)) Then
                Execute_galloping -test = "Failed" ' Test Failed
            Else
                Write_Data(intLoop) = 11 1111   ' Write 11 1111 in the register address intLoop
                Read_Data(intLoop)              ' Read 11 1111 from the register address intLoop

                ' Compare write value (Hex 1) with the read value (Hex 1)
                If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                    Execute_galloping -test = "Passed"    ' Test Passed
                    'Check all above Registers kept the 0s values
                    For intNextLoop = intLoop To intLastRegister
                        Read_Data(intLoop) ' Read 00 0000 from the register address intLoop
                        ' Compare write value (Hex 0) with the read value (Hex 0)
                        If (Write_Data(intNextLoop) <> Read_Data(intNextLoop)) Then
                            Execute_galloping -test = "Failed" ' Test Failed
                        Else
                            Execute_galloping -test = "Passed"    ' Test Passed
                    Next intLoop

                Else
                        Execute_galloping -test = "Failed"    ' Test Failed
                End If
            End If
        Next intLoop      ' End Ascending Loop

        ' Set a loop from last register to first register (descending order)
        For intLoop = intLastRegister To intFirstRegister Step -1
            Read_Data(intLoop)      ' Read 11 1111 from the register address intLoop
            ' Compare write value (Hex 1) with the read value (Hex 1)
            If (Write_Data(intLoop) <> Read_Data(intLoop)) Then
                Execute_galloping -test = "Failed" ' Test Failed
            Else
                Write_Data(intLoop) = 00 0000    'Write 00 0000 in the register address intLoop
                Read_Data(intLoop)               'Read 00 0000from the register address intLoop

                ' Compare write value (Hex 0) with the read value (Hex 0)
                If (Write_Data(intLoop) = Read_Data(intLoop)) Then
                    Execute_galloping -test = "Passed"    ' Test Passed

                                                                    Continued....
```

```
                    'Check all lower Registers kept the 1s values
                    For intNextLoop = intLoop To intFirstRegister Step -1
                        Read_Data(intLoop) ' Read 11 1111 from the register address intLoop
                        ' Compare write value (Hex 1) with the read value (Hex 1)
                        If (Write_Data(intNextLoop) <> Read_Data(intNextLoop)) Then
                            Execute_galloping -test = "Failed"  ' Test Failed
                        Else
                            Execute_galloping -test = "Passed"    ' Test Passed
                    Next intLoop


                Else
                    Execute_galloping -test = "Failed"    ' Test Failed
                End If
            End If
        Next intLoop      ' End Descending Loop
```

Figure B.6: **Galloping Test**

```
Public Function Read_Data () As String
        ' Buffer to hold input string
        Dim Instring As String
        ' Use COM1.
        MSComm1.CommPort = 1
        ' Set 9600 baud, no parity, 8 data, and 1 stop bit.
        MSComm1.Settings = "9600,N,8,1"
        ' Tell the control to read entire buffer when Input is used.
        MSComm1.InputLen = 0
        ' Open the port.
        MSComm1.PortOpen = True
        ' Send all D-registers read (macro) command to 1-Meg Modem (SUT).
        MSComm1.Output = "mmd " + strSlotNo + " 00" + ChrS(13)


        ' Wait for data to come back to the serial port.
Do
                'DoEvents
Loop Until MSComm1.InBufferCount >= MSComm1.InBufferCount
        ' Read the DBIC response data in the serial port.
        Instring = MSComm1.Input

        ' Close the serial port.
        MSComm1.PortOpen = False
End Function
```

Figure B.7: **Shows Serial Communications with SUT**

APPENDIX C

SOFTWARE TESTING - SOURCE CODE SAMPLES

1. Manager SNMP-API:

Sets the SNMP Manager session by using the **Open** method.

Syntax: [Status =] *Object.Open* ( )

Status: A Boolean value. TRUE if the Object is in the Closed state and the connection process is started. FALSE if the Object is not in the Closed state or if the system network access fails for any reason.

Remarks:
If successful, application calls the OnOpen event, otherwise application calls the OnException event.

The Debug, LocalPort, LocalDotAddr, OemLicense, and ShowIcon properties must be set to their desired values prior to calling the Open method.

Usage:

```
Private Sub Form_Load()
   Dim ObjectID As String
   gRequestID = 0

   ' Allocates socket resources and establishes a local SNMP session. If
   ' successful, you will receive a Connect event. If not, you will receive an
   ' Exception Event.
   SNMP1.LocalPort = 162
   SNMP1.Open
   ' Allow time to receive Events
   Call DelayTimer(int1Sec)
End Sub

' Connect Event
Private Sub SNMP1_OnOpen(ByVal sSessionDotAddr As String. ByVal nSessionPort As
Long. ByVal sLocalName As String. ByVal nMaxByteCnt As Long)
   'Winsock is initialize, we have a port open
  lblStatus.Caption = "Port opened"
  SNMPstatus = "Port opened"
End Sub

' Exception Event
Private Sub SNMP1_OnException(ByVal nErrorCode As Integer. ByVal sErrorDesc As
String)
  lblStatus.Caption = "ErrorCode: " & nErrorCode & " received.  " & sErrorDesc
End Sub
```

Figure C.1: "Open" method Syntax and its Usage

Handles the building and encoding of outbound messages through the **SendGetRequest,** and **SendSetRequest** methods.

SendGetRequest Method & SendSetRequest Method
Sends a **get-request** message to an SNMP agent, requesting the value of one or more objects. Sends a **set-request** message to an SNMP agent, requesting one or more objects to be set to a new value.

Syntax: [Status =] Object.SendGetRequest (RemoteHost, RequestID, ObjectID)
[Status =] Object.SendSetRequest (RemoteHost, RequestID, ObjectID, ObjectValue, ObjectType)

Status : A **Boolean** value. TRUE if the Object is in the **snmpOpen state**, otherwise FALSE.
RemoteHost: A **String** value that identifies the name of the remote host.
RequestID : An expression that evaluates to a **Long** value. SNMP Manager API makes a copy of the RequestID parameter and passes it back when it calls the **OnSend** event.
ObjectID : A **Variant** expression that evaluates to a **String** or array of **Strings** that specifies one or more object identifiers that are to be used for fetching (GetRequest)/setting (SetRequest) object values.
ObjectValue : A **Variant** value or array of **Variant** values that describe one or more objects. The type of each element is specified in the *ObjectType* parameter.
ObjectType : A **Variant** expression that evaluates to an **Integer** or array of **Integers** that specifies the type of one or more objects.

Remarks:
1. Requests are normally sent to UDP port 161, so the RemotePort property should normally be set to this value.
2. If an array of ObjectIDs is created, the dimension statement specifies the upper bound.
3. The Community and RemotePort properties should be set to their desired values prior to calling this method.

Usage:

```
Const sRemoteHosts = Form1.txtRemoteIP_Address.Text   ' SUT IP Address
' SendGetRequest
        Private Function GetRequest(ObjectID) As String
  gRequestID = gRequestID + 1
  SNMP1.SendGetRequest sRemoteHosts, gRequestID, ObjectID
  Call DelayTimer(intISec)
End Function
' SendSetRequest
Private Sub SetRequest(ObjectID As String, ObjectValue As String)
  gRequestID = gRequestID + 1
  'Setting OID's value
  If ObjectValue <> "" Then
    SNMP1.SendSetRequest sRemoteHosts, gRequestID, ObjectID & (strSlotNo + 1),
    ObjectValue, snmpInteger
  End If
  SNMP1.Tag = "Set_Value"
  Call DelayTimer(intISec)
End Sub
```

Figure C.2: Shows SendGet/SetRequest methods & their Usage

Handles the parsing and decoding of inbound messages and notifies the application by firing the **OnRcvSnmp** event.

OnRecvSnmp Event
SNMP Manager API calls this function when a session has received an SNMP message and the control's **VarDataType** property is set to String or ByteArray

Syntax: **Object_OnRecvSnmp** (sCommunity, nRequestID, nSnmpError, nErrorIndex, nMessageType, vObjectID, vObjectValue, vObjectType, sRemoteDotAddr, nRemotePort)

sCommunity	A **String** value that specifies the community name which is often used as a pseudo-password. Typical values include "public" and "private".
nRequestID	A **Long** value that was passed as the user-defined RequestID argument in the Object's **SendGetRequest**, or **SendSetRequest** method.
nSnmpError	An **Integer** value that specifies an error code returned from the SNMP agent.
nErrorIndex	An **Integer** value that specifies the index to the variable in error within the packet.
nMessageType	An **Integer** value that specifies the type of message received.
vObjectID	A **Variant** expression that evaluates to a **String** or array of **Strings** that specifies one or more object identifiers.
vobjectValue	A **Variant** value or array of **Variant** values that specifies one or more object values. The exact type of each element depends on the corresponding element in the *ObjectType* parameter.
vObjectType	A **Variant** expression that evaluates to an **Integer** or array of **Integers** that specifies the type of one or more objects.
sRemoteDotAddr	A **String** value that specifies the dot address of the remote host.
nRemotePort	A **Long** value that specifies the port number of the remote host.

Remarks:
This function is called automatically when a session receives an SNMP message on the port used in the Open method. Application will receive one OnRecvSnmp event for each UDP packet received that is of the type get-request, set-request.
NOTE: There is a one-to-one correspondence between each element in the ObjectID, ObjectValue, and ObjectType parameters.

Usage:
Private Sub SNMP1_OnRecvSnmp(ByVal sCommunity As String, ByVal nRequestID As Long, ByVal nErrSnmp As Integer, ByVal nErrIndex As Integer, ByVal nMsgType As Integer, ByVal vObjectID As Variant, ByVal vObjectValue As Variant, ByVal vObjectType As Variant, ByVal sRemoteDotAddr As String, ByVal nRemotePort As Long)

```
    Dim num_char As Integer
Dim temp_str As Variant
Dim PhysicalAddr As String
Dim i As Integer

'Let user know we got a packet
lblStatus.Caption = "Packet received"
```

Continue on next Page

Continue OnRecvSnmp Event

```
Select Case SNMP1.Tag

'if the user did a Set, lets update our info with what was returned.
Case "xlcDesiredValue":
  lblSysObjectID.Text = vObjectValue(1)

' Parsing and decoding of received message
Case "GetMib"
  strSyncStatus(intDLS400Loop) = vObjectValue(0)
  strxlcDSSpeed(intDLS400Loop) = vObjectValue(1)
  lblSysObjectID.Text = vObjectValue(1)
  strxlcUSSpeed(intDLS400Loop) = vObjectValue(2)
  lblSysUpTime = vObjectValue(2)
  strxlcSignleQuality(intDLS400Loop) = vObjectValue(3)
  strxlcAGCIn(intDLS400Loop) = vObjectValue(4)

  If strxlcFECState = "1" And strSystemPhaseSet = "Phase2" Then
     'FEC Count at xLC (US)
     strxlcFECCountOrCRC8(intDLS400Loop) = vObjectValue(5)
  Else
     'CRC Count at xLC (US)
     strxlcFECCountOrCRC8(intDLS400Loop) = vObjectValue(5)
  End If

  strxlcEnergyValue(intDLS400Loop) = vObjectValue(6)
  strxlcTxPower(intDLS400Loop) = vObjectValue(7)
  strcpeSignleQuality(intDLS400Loop) = vObjectValue(8)
  strcpeAGCIn(intDLS400Loop) = vObjectValue(9)
  strcpeTxPower(intDLS400Loop) = vObjectValue(10)

  If strxlcFECState = "1" And strSystemPhaseSet = "Phase2" Then
    'FEC Count at xLC (DS)
    strcpeFECCountOrCRC8(intDLS400Loop) = vObjectValue(11)
  Else
    'CRC Count at xLC (DS)
    strcpeFECCountOrCRC8(intDLS400Loop) = vObjectValue(11)
  End If

End Select

'Decode received message into QAM and BW
Call ConvertDsQamAndBW(strxlcDSSpeed())
Call ConvertUsQamAndBW(strxlcUSSpeed())

'Reset Tag Value
SNMP1.Tag = ""
End Sub
```

Figure C.3: Shows OnRecvSnmp Event methods & their Usage

2. FTP Client API:

Retrieving and storing files and calculating its FTP rate through FTP Client API.

```
' Login into FTP Server
Private Sub intLogin()
   ' Initiate Login
   FTPClient.Login strFTPserver, strUserID, strUserPasswd, strAccount
End Sub

Private Sub fileCommand(strFileCmd, txtRemoteFile, txtLocalFile, txtBufferSize1)
     Dim txtRestartZero As String

   ' Set the Time out timer. If no Reply is received within the defined time,
   ' connections will be closed
     xferTimer.Interval = CLng(60) * 1000
     xferTimer.Enabled = True

    ' set Binary mode
     gXferType = ftpImage

   ' Choose command either to retrieve or store
   Select Case strFileCmd
     Case "RETR"        ' Retrieve (GET) file
          txtRestartZero = 0
        gStart = Timer
        gFileNum = 1
        Open txtLocalFile For Output As #1
        FTPClient.Retrieve "", txtRemoteFile, Val(txtBufferSize1), gXferType
     Case "STOR"        ' Store (PUT) file
          txtRestartZero = 0
        gAbort = False
        ' Set the Restart Marker
        FTPClient.Restart Val(txtRestartZero)
        If txtLocal.Text <> "" Then
          FTPClient.Store txtLocal.Text, txtRemote.Text, Val(txtBufferSize.Text), gXferType
        Else
          txtReply.SelText = "No local name given" & vbCrLf
        End If
  End Select
End Sub
```

Continue on Next Page

```
Private Sub FtpClient_OnRecv(ByVal vRecvData As Variant)

   Dim count As Integer
   Dim speed As Long

   Select Case gFileNum
      Case 0

      Case 1
         ' Put data into open file
         If Len(vRecvData) = 0 Then ' Transfer is complete
            Close #1
            cmdCommand.Enabled = True
            gFileNum = 0
            DetermineRate
         Else
            Print #1, vRecvData;
         End If
   End Select
End Sub

' Calculate the FTP rate
Private Sub DetermineRate()
   Dim speed As Long
   Dim rate As Long

   speed = Timer - gStart
   If speed > 0 Then
      rate = Val(lblByteCount.Caption) / speed
   Else
      rate = 0
   End If

   If rate > 0 Then
      lblTransferRate.Caption = Str(rate) & " Bytes/Sec"
   Else
      lblTransferRate.Caption = "Could not Resolve"
   End If

End Sub
```

Figure C.4: Shows Retrieval & Storing file through FTP and their FTP rate

3. ICMP (Ping) Messages:

Formation of ICMP packet and determination of data connectivity.

```
'declares for function to be used from icmp.dll
Private Declare Function IcmpCreateFile Lib "icmp.dll" () As Long
Private Declare Function IcmpCloseHandle Lib "icmp.dll" (ByVal IcmpHandle As Long) As Long
Private Declare Function IcmpSendEcho Lib "icmp.dll" (ByVal IcmpHandle As Long, _
      ByVal DestinationAddress As Long, ByVal RequestData As String, _
      ByVal RequestSize As Integer, RequestOptions As ip_option_information, _
      ReplyBuffer As icmp_echo_reply, ByVal ReplySize As Long, ByVal Timeout As Long) As Long

Private Declare Function WSAStartup Lib "wsock32" (ByVal wVersionRequested As Integer, _
      lpWSAData As tagWSAData) As Integer
Private Declare Function WSACleanup Lib "wsock32" () As Integer

'start pinging
Private Sub startPing(strIPAddress, strtxTTL)
   Dim hFile, lRet, lIPAddress, lPingRet  As Long
   Dim strMessage, strtxTTL  As String
   Dim pOptions    As ip_option_information
   Dim pReturn     As icmp_echo_reply
   Dim iVal        As Integer
   Dim pWsaData    As tagWSAData
   Dim strIPAddress As String

   strMessage = "Echo this string of data"
   iVal = WSAStartup(&H101, pWsaData)

   ' convert the IP address to a long
   lIPAddress = ConvertIPAddressToLong(strIPAddress)
   ' open up a file handle for doing the ping
   hFile = IcmpCreateFile()
   ' set the TTL from the text box, try values of 1 to 255
   pOptions.Ttl = strtxTTL
   ' Call the function that actually does the ping.
   lRet = IcmpSendEcho(hFile, lIPAddress, strMessage, Len(strMessage), _
               pOptions, pReturn, Len(pReturn), PING_TIMEOUT)

   If lRet = 0 Then
      ' the ping failed for some reason, hopefully the error is in the return buffer
      lbReturn.AddItem "Ping failed with error " & pReturn.Status
   Else
      If pReturn.Status <> 0 Then
         lbReturn.AddItem "Error -> Ping failed to complete, code = " & pReturn.Status
      ' The ping succeeded
      Else
         lbReturn.AddItem "Success -> completion time is " & pReturn.RoundTripTime & "ms."
      End If
   End If

   ' close the file handle that was used
   lRet = IcmpCloseHandle(hFile)
   iVal = WSACleanup()
End Sub
```

Continue on next page

```
' Converts a dotted IP address (eg: "123.234.2.45") to a long integer for use in sending a ping.
' This routine converts the string.
' Essentially we take the 4 numbers, flip them around and make a long by shifting all the parts
' into the correct byte. We do it here by making a hex string and converting it to a long.
' When we get in "a.b.c.d" what we want out is Val(&Hddccbbaa).

Function ConvertIPAddressToLong(strAddress As String) As Long

   Dim strTemp          As String
   Dim lAddress         As Long
   Dim iValCount        As Integer
   Dim lDotValues(1 To 4)  As String

   ' set up the initial storage and counter
   strTemp = strAddress
   iValCount = 0

   ' keep going while we still have dots in the string
   While InStr(strTemp, ".") > 0
      iValCount = iValCount + 1   ' count the number
      lDotValues(iValCount) = Mid(strTemp, 1, InStr(strTemp, ".") - 1)   ' pick it off and convert it
      strTemp = Mid(strTemp, InStr(strTemp, ".") + 1) ' chop off the number and the dot
   Wend

   ' the string only has the last number in it now
   iValCount = iValCount + 1
   lDotValues(iValCount) = strTemp

   ' if we didn't get four pieces then the IP address is no good
   If iValCount <> 4 Then
      ConvertIPAddressToLong = 0
      Exit Function
   End If

   '  take the four value, hex them, pad to 2 digits, make a hex string and then convert
   '  the whole mess to a long for returning
   lAddress = Val("&H" & Right("00" & Hex(lDotValues(4)), 2) & _
            Right("00" & Hex(lDotValues(3)), 2) & _
            Right("00" & Hex(lDotValues(2)), 2) & _
            Right("00" & Hex(lDotValues(1)), 2))

   '  set the return value
   ConvertIPAddressToLong = lAddress

End Function
```

Figure C.5: ICMP packet generation and its "echo" reply

4. Serial Communication:

```
Public Function SetLoop(strSetLength, intDLS400Loop, intDls400ComPort) As String
' Buffer to hold input string
   Dim strLoopLoopTmp As String
   Dim strLoopLoopTmpParse As String
   Dim strLoopLength As String
   Dim strLoopLengtQuery As String

    ' Use COM1 or COM2.
          MSComm2.CommPort = intDls400ComPort
   ' 9600 baud, no parity, 8 data, and 1 stop bit.
   MSComm2.Settings = "9600,N,8,1"
   ' Tell the control to read entire buffer when Input is used.
          MSComm2.InputLen = 0
   ' Open the port.
          MSComm2.PortOpen = True
   ' Loops to Set
   Select Case strSetLength
   Case "VARIABLE_26_AWG 1Kft":
      'set command to set DLS400 to 26AWG X Length
      strLoopLength = ":SET:CHAN:LOOP VARIABLE_26_AWG:LINE 1K" + ChrS(10)
     'set query command to check length currently simulated by DLS400
      strLoopLengthQuery = ":SET:CHAN:LOOP?:LINE?" + Chr$(10)




   Case "ANSI LOOP 15":
      'set command to set DLS400 to ANSI Loop 15
      strLoopLength = ":SET:CHAN:LOOP ANSI_#15" + ChrS(10)
     'set query command to check length currently simulated by DLS400
      strLoopLengthQuery = ":SET:CHAN:LOOP?:LINE?" + Chr$(10)

   End Select

     'Send command to DLS400 (Loop Simulator) to set Straight/ANSI Loop
      strLoopLength = ReadDataCom3(strLoopLength)
      Call DelayTimer(int1Sec)
      strLoopLength = ReadDataCom3(strLoopLengthQuery)

   ' Close the serial port.
   MSComm2.PortOpen = False
   'SetRate = strSetRate
   SetLoop = strLoopLength

End Function
```

Note: Application supports 27 Loops. In the above example, only two Loops, *VARIABLE_26_AWG 1Kft and ANSI Loop 15*, are shown.

Figure C.6: Example of serial communication with loop simulator

5. Transforming Result in Microsoft Excel Format:

```
If Form1.CheckExcel.Value = chkEnabled Then
    Dim XLSheet As Object

    'Open Excel. Change path name for your hard disk
    Set XLSheet = GetObject("C:\My Documents\1MM Test Result.xls", "Excel.Sheet")

    'Make Excel visible
    XLSheet.Application.Visible = True
    XLSheet.Windows("1MM Test Result.xls").Visible = True

    'set the zoom to 75%
    XLSheet.Windows("1MM Test Result.xls").Zoom = 75
    'Select Cells of Sheet1
    Set XLObjCell = XLSheet.Sheets("Sheet1")

    'Set the Column Size
    XLObjCell.Columns("A:A").ColumnWidth = 21.29
    XLObjCell.Columns("B:B").ColumnWidth = 10.71
      .
      .
    XLObjCell.Columns("O:O").ColumnWidth = 9

    'Set the cell to wrap the text
    XLObjCell.Cells(10, 4).WrapText = True
    XLObjCell.Cells(10, 5).WrapText = True
    XLObjCell.Cells(10, 6).WrapText = True
    XLObjCell.Cells(10, 11).WrapText = True
    XLObjCell.Cells(10, 12).WrapText = True
    XLObjCell.Cells(10, 13).WrapText = True

    'Write into the cells
    XLObjCell.Cells(1, 1).Value = "Date: " + strDate
    XLObjCell.Cells(2, 1).Value = "Time: " + strTime
    XLObjCell.Cells(10, 15).Value = "UsResult"
      .
      .
    'Change Size & Bold
    XLObjCell.Cells(9, 5).Font.Size = 14
    XLObjCell.Cells(9, 5).Font.Bold = True
      .
      .
    XLObjCell.Cells(10, 15).Font.Bold = True

    'Set the central alignment
    XLObjCell.Range("B11:O37").HorizontalAlignment = xlCenter
```

Continue on Next Page

```
        'set DS borders
        With XLObjCell.Range("B9:H9").Borders()
           .LineStyle = xlContinuous
           .Weight = xlThick
           .ColorIndex = xlAutomatic
        End With
        'Remove internal vertical lines
        XLObjCell.Range("B9:G9").Borders(xlRight).LineStyle = xlNone
        XLObjCell.Range("C9:H9").Borders(xlLeft).LineStyle = xlNone

        'set US borders
        With XLObjCell.Range("I9:O9").Borders()
           .LineStyle = xlContinuous
           .Weight = xlThick
           .ColorIndex = xlAutomatic
        End With
        'Remove internal vertical lines
        XLObjCell.Range("I9:N9").Borders(xlRight).LineStyle = xlNone
        XLObjCell.Range("J9:O9").Borders(xlLeft).LineStyle = xlNone

    'Start setting line loops and retrieving values
    For intDLS400Loop = 1 To intMaxLoop

        'Set the first line of Loop row
        intExcelRow = intDLS400Loop + intStartExcelLine

        'Start writing the DS results in the appropriate cells
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 1)).Value = strDsTargetQam(intDLS400Loop) _
        + " " + strDsTargetBw(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 2)).Value = strxlcTxPower(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 3)).Value = strSyncStatus(intDLS400Loop) _
        + strxlcDSSpeed(intDLS400Loop) + " " + strxlcDSBW(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 4)).Value = strcpeFECCountOrCRC8(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 5)).Value = strcpeSignleQuality(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 6)).Value = strcpeAGCIn(intDLS400Loop)
        XLObjCell.Cells(intExcelRow, (intStartExcelCol + 7)).Value = strDsPass(intDLS400Loop)
        With XLObjCell.Cells(intExcelRow, (intStartExcelCol + 7)).Interior
                                              .ColorIndex = intDsColor(intDLS400Loop)
                                               .Pattern = xlSolid
                                              .PatternColorIndex = xlAutomatic

        End With
    Next intDLS400Loop

'End CheckExcel if statement
End If

'Clear variable
 Set XLObjCell = Nothing
 Set XLSheet = Nothing
```

Figure C.7: Sample Code to display Result in Excel format

6. GUI's Functionality

```
Private Sub Form_Load()

    Dim ObjectID As String
    Dim strcpeReset As String
    Dim strxlcDesiredStateEnabled As String
    Dim strxlcDesiredStateDisabled As String

    strcpeReset = 2
    gRequestID = 0
    Show


    ' Allocates socket resources and establishes a local SNMP session.  If
    ' successful, you will receive a Connect event.  If not, you will receive an
    ' Exception Event
    SNMP1.LocalPort = 162
    SNMP1.Open

    Call DelayTimer(int1Sec)


    ' Get an IP address from Form1.txtRemoteIP_Address.Text TextBox
    sRemoteHosts = Form1.txtRemoteIP_Address.Text
    Call SetDesiredStateDisabledEnabled


    'cpeState = 1(InSync), 9(out of Sync)
    While strSyncStatus(intDLS400Loop) <> "1"
        Call GetSyncStatus
        Call DelayTimer(int1Sec)
    Wend


    Call Run_Test


End Sub
```

Figure C.8: SNMP agent using the IP address to access the SUT

```
Private Sub cmdStart_Click()
    Dim intIndex As Integer
    Dim strChkSlotSetTrue As String
    Dim intChkLoopSet As Integer
    Dim intChkLoopSetTrue As Integer
    intChkLoopSetTrue = 1
    'Check which array of Slot#(Option6(intIndex) button) is selected
    For intIndex = 0 To (intMaxSlot - 1)
       If (Option6(intIndex).Value = True) Then
          strSlotNo = intIndex
          strChkSlotSetTrue = True
          intIndex = 15
       End If
    Next intIndex
    'Check which Loop(s)Check1(intIndex) box) is(are) selected
    For intIndex = 1 To intMaxLoop
       If (Check1(intIndex).Value = intChkLoopSetTrue) Then
          intChkLoopSet = intChkLoopSetTrue
       End If
    Next intIndex
   'Check which array of Test Case#(Option14(intIndex) button) is selected
    For intIndex = 0 To (intMaxTestCase - 1)
       If (Option14(intIndex).Value = True) Then
          strTestNo = intIndex
          strChkTestCaseTrue = True
          intIndex = 7
       End If
    Next intIndex
       'Check mandatory item Slot#(Option6 button), Loop(s)(Check6 box) & Test
       'Case# are selected
              If ((strChkSlotSetTrue = "True") And (intChkLoopSet =
         intChkLoopSetTrue) And + -
                 (strChkTestCaseTrue = "True") ) Then
          'Proceed to run test
          Hide
          Form2.Show
       Else
          strLoopRequest = MsgBox("Please set Slot#, Loop and TesCaset#", vbInformation +
                vbOKOnly, "Setting Request")
       End If
End Sub
```

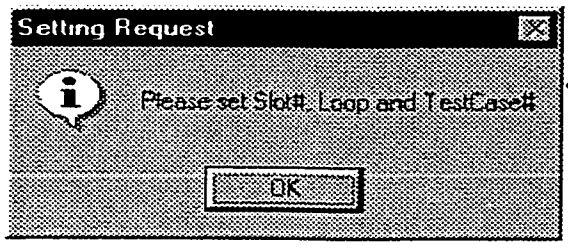

Figure C.9: Checking of Slot & Loop Setting and the error message

```
' Following function open the "File Open Dialog Box" and retrieve the name of config file.
Function FileOpenProc()
   Dim intRetVal
            On Error Resume Next
   Dim strOpenFileName As String
   Form1.CMDialog1.Filename = ""
   ' Use Common Dialog Box (CMDialog1) to open "File Open Dialog Box"
   Form1.CMDialog1.ShowOpen
   If Err <> 32755 Then    ' User chose Cancel.
     ' Retrieve the config file name to be loaded
     strOpenFileName = Form1.CMDialog1.Filename
     ' If the file is larger than 65K, it can't be opened, so cancel the operation.
     If FileLen(strOpenFileName) > 65000 Then
       MsgBox "The file is too large to open."
       Exit Function
     End If
     OpenFile (strOpenFileName)
     FileOpenProc = strConfigFileContents
   End If
End Function

' Following function retrieve the file content
Sub OpenFile(Filename)
   Dim fIndex As Integer

   On Error Resume Next
   ' Open the selected file.
   Open Filename For Input As #1
   If Err Then
     MsgBox "Can't open file: " + Filename
     Exit Sub
   End If
   ' Change the mouse pointer to an hourglass.
   Screen.MousePointer = 11

   ' Change the form's caption and display the new text.
   Form1.Tag = fIndex
   Form1.Caption = UCase(Filename)
   ' Retrieve the file content
   strConfigFileContents = Input(LOF(1), 1)
   ' Close the selected file.
   Close #1
   ' Reset the mouse pointer.
   Screen.MousePointer = 0
End Sub
```

Figure C.10: Module showing File Open Dialog Box and retrieval of config-file

```
'Following function extract the existing configuration for saving it into a file
Private Sub cmdSaveConfig_Click()
    Dim intLoopNo, intQAMLoopNo, intBW, intNB As Integer
    Dim strSaveFileName, strDefaultName, strConfigFileContent As String

    intWB = 2
    intNB = 1
    int256 = 1
    int64 = 2
    int16 = 3
    int4 = 4
    For intLoopNo = 1 To intMaxLoop

        'Extracting the existing DS BW setting into a variable string
        If (Option7(intLoopNo & intWB).Value = optEnabled) Then
            strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intWB)
        Else
            strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intNB)
        End If
        'Extracting the existing DS QAM rate into a variable string
        For intQAMLoopNo = 1 To 4
            If (Option8(intLoopNo & intQAMLoopNo).Value = optEnabled) Then
                strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intQAMLoopNo)
            End If
        Next intQAMLoopNo
        'Extracting the existing US BW setting into a variable string
        If (Option9(intLoopNo & intWB).Value = optEnabled) Then
            strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intWB)
        Else
            strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intNB)
        End If
        'Extracting the existing US QAM rate into a variable string
        For intQAMLoopNo = 1 To 4
            If (Option10(intLoopNo & intQAMLoopNo).Value = optEnabled) Then
                strConfigFileContent = strConfigFileContent + ", " + CStr(intLoopNo & intQAMLoopNo)
            End If
        Next intQAMLoopNo
        'Extracting the existing Loop Test Time into a variable string
        strConfigFileContent = strConfigFileContent + ", " + Text3(intLoopNo).Text

    Next intLoopNo

    'Add a "," at the end of Stream so that InStr can get info in LoadNewConfig function
    strConfigFileContent = strConfigFileContent + ", "

    UpdateContent (strConfigFileContent)

    ' Get the filename, and then call the save procedure, strSaveFileName.
    strSaveFileName = GetFileName("Untitled.txt")
    If strSaveFileName <> "" Then SaveFileAs (strSaveFileName)

End Sub
```

Continue on Next Page

```
' Following function open the "File Save Dialog Box" and retrieve the path and name of
' file to be saved.
Function GetFileName(Filename As Variant)

   ' Display a Save As dialog box and return a filename. If the user chooses Cancel, return
    an empty string.
   On Error Resume Next
   Form1.CMDialog1.Filename = Filename
   Form1.CMDialog1.ShowSave

   If Err <> 32755 Then    ' User chose Cancel.
      GetFileName = Form1.CMDialog1.Filename
   Else
      GetFileName = ""
   End If


End Function


' Following function save the existing configuration
Sub SaveFileAs(Filename As String)
   On Error Resume Next
   Dim strContents As String

   ' Open the file.
   Open Filename For Output As #1
   strContents = strConfigFileContents

   ' Display the hourglass mouse pointer.
   Screen.MousePointer = 11

   ' Write the existing configuration to a saved file.
   Print #1, strContents
   Close #1

   ' Reset the mouse pointer.
         Screen.MousePointer = 0

   ' Set the form's caption.
   If Err Then
      MsgBox Error, 48, App.Title
   Else
      Form1.Caption = UCase(Filename)
   End If


End Sub
```

Figure C.11: Extraction of existing config, Save As Dialog Box opening, Saving file


```
'Setting Loop Simulator (DLC400)  Com Port
Private Sub optDLS400ComPort_Click(Index As Integer)
    For intOpt = 1 To 2

      If (intOpt = 1 And optDLS400ComPort (intOpt).Value = optEnabled) Then

         'Setting Trace Com Port to COM2
         optTraceComPort(2).Value = optEnabled

      End If

      If (intOpt = 2 And optDLS400ComPort (intOpt).Value = optEnabled) Then

          'Setting Trace Com Port to COM1
         optTraceComPort (1).Value = optEnabled

      End If

    Next intOpt

End Sub

'Setting Loop Trace (SUT Debug) Com Port
Private Sub optTraceComPort _Click(Index As Integer)
    For intOpt = 1 To 2

      If (intOpt = 1 And optTraceComPort (intOpt).Value = optEnabled) Then

         'Setting Loop Simulator to COM2
         optDLS400ComPort (2).Value = optEnabled

      End If

      If (intOpt = 2 And optTraceComPort (intOpt).Value = optEnabled) Then

         'Setting Loop Simulator to COM1
         optDLS400ComPort (1).Value = optEnabled

      End If

    Next intOpt

End Sub
```

Figure C.12: Shows if COM1 is selected by one device, COM2 will automatically be selected by other device

```
' Send request to get the xLC & CPE PEC code. PEC code determines
' the hardware phases
Private Sub GetSyncStatus()
   Dim ObjectID(3) As String

   ObjectID(0) = cpeStateOID & (strSlotNo + 1)
   ObjectID(1) = xlcTypePECOID & (strSlotNo + 1)
   ObjectID(2) = cpeProductCodeOID & (strSlotNo + 1)
   ObjectID(3) = xlcFECStateOID & (strSlotNo + 1)
   gRequestID = gRequestID + 1

   SNMP1.SendGetRequest sRemoteHosts, gRequestID, ObjectID
   SNMP1.Tag = "SyncStatus"
   Call DelayTimer(int1Sec)
End Sub

' Response received with the PEC code values
Private Sub SNMP1_OnRecvSnmp(ByVal sCommunity As String, _
        ByVal nRequestID As Long, ByVal nErrSnmp As Integer, _
        ByVal nErrIndex As Integer, ByVal nMsgType As Integer, _
        ByVal vObjectID As Variant, ByVal vObjectValue As Variant, _
        ByVal vObjectType As Variant, _
        ByVal sRemoteDotAddr As String, ByVal nRemotePort As Long)

   Select Case SNMP1.Tag

   Case "SyncStatus"
      strSyncStatus(intDLS400Loop) = vObjectValue(0)

      If strSyncStatus(intDLS400Loop) = 1 Then
         strxlcPecCode = vObjectValue(1)
         strcpePecCode = vObjectValue(2)
         strxlcFECState = vObjectValue(3)
         Call CheckPhase(strxlcPecCode, strcpePecCode)
      End If

      ' Reset Tag Value
      SNMP1.Tag = ""

   End Select
End Sub
```

Continue on the next page

```
Private Sub CheckPhase(strxlcPecCode, strcpePecCode)
    ' Check phases set at the GUI
    If Form1.Option1(1).Value = True Then
       strDbicSet = "Phase1"
    Else
       strDbicSet = "Phase2"
    End If

    ' Check xLC PEC code at the GUI
    If Form1.Option3(1).Value = True Then
       strxlcPecCodeSet = "Phase1"
    Else
       strxlcPecCodeSet = "Phase2"
    End If

    ' Check xLC PEC code at the GUI
    If Form1.Option4(1).Value = True Then
       strcpePecCodeSet = "Phase1"
    Else
       strcpePecCodeSet = "Phase2"
    End If


 ' Check actual Hardware Phases
 'Process the xLC PEC code value to determine the xLC Phase
 Select Case strxlcPecCode

    Case "1"
       strxlcPecCode = "Unknown"

    Case "2"
       strxlcPecCodePhase = "Phase1"
       strxlcPecCode = "NTEX17AA"
       Form1.Option3(1).Value = True
       strxlcPecCode = "Phase1"

    Case "3"
       strxlcPecCodePhase = "Phase2"
       strxlcPecCode = "NTEX17BA"
       Form1.Option3(2).Value = True

    Case "4"
       strxlcPecCodePhase = "Phase2"
       strxlcPecCode = "NTEX17CA"
       Form1.Option3(2).Value = True

 End Select
```

Continue on the next page

```
'Process the CPE PEC code value to determine the xLC Phase
   Select Case strcpePecCode

      Case "1"
         strcpePecCode = "Unknown"

      Case "NTEX35AA"
         strcpePecCodePhase = "Phase1"
         Form1.Option4(1).Value = True

      Case "NTEX35BA"
         strcpePecCodePhase = "Phase2"
         Form1.Option4(2).Value = True

   End Select

'Compare the actual phase with the set phase. If different, generate a message
   If strxlcPecCodePhase = "Phase2" And strcpePecCodePhase = "Phase2" Then
      Form1.Option1(2).Value = True
      strSystemPhaseSet = "Phase2"

   Else

      Form1.Option1(1).Value = True
      strSystemPhaseSet = "Phase1"
   End If

   If strxlcPecCodeSet <> strxlcPecCodePhase Or strcpePecCodeSet <>
   strcpePecCodePhase Then
      strLoopRequest = MsgBox("Please note that system is " & strSystemPhaseSet & "
      System", vbInformation + vbOKOnly, "Information")
   End If
End Sub
```

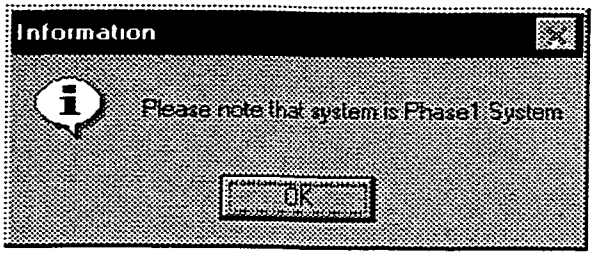

Figure C.13: Shows (i) phase detection. (ii) message acknowledgement

```
Private Sub optSelectLoop_Click(Index As Integer)
' optSelectLoop  isOption5 (Index)
   Dim intStraightISDNLoop As Integer
   Dim intStraightLoop As Integer
   Dim intISDNLoop As Integer
   Dim intIndex As Integer
  intStraightLoop = 15
  intISDNLoopStart = 16

   Select Case Index
        'Select Straight Loop only by single click
        Case "1":
           'Enabled loop
           For intIndex = 1 To intStraightLoop
              Check1(intIndex).Value = chkEnabled
           Next intIndex
           'Disable other than Straight Loop
           For intIndex = intISDNLoopStart To intMaxLoop
              Check1(intIndex).Value = chkDisabled
           Next intIndex

        'Select ISDN Loop only by single click
        Case "2":
          'Enabled loop
           For intIndex = intISDNLoopStart To intMaxLoop
              Check1(intIndex).Value = chkEnabled
           Next intIndex
           'Disable other than ISDN Loop
           For intIndex = 1 To intStraightLoop
              Check1(intIndex).Value = chkDisabled
           Next intIndex

        'Select All Loop by single click
        Case "3":
          'Enabled loop
           For intIndex = 1 To intMaxLoop
              Check1(intIndex).Value = chkEnabled
           Next intIndex

        'Select "selected" loop one by one
        Case "4":
          'Disable all loop for selecting the "selected" loop one by one
           For intIndex = 1 To intMaxLoop
              Check1(intIndex).Value = chkDisabled
           Next intIndex
   End Select

End Sub
```

Figure C.14: Shows select/deselect all, straight-only, selected and ISDN loops by one clicked