SLAC-PUB-2103
April 1978

AN IBM 370/360 SOFTWARE PACKAGE FOR DEVELOPING STAND ALONE LSI-11 SYSTEMS*

R.L.A. Cottrell and C.A. Logg

Stanford Linear Accelerator Center
P. O. Box 4349
Stanford, California 94305

ABSTRACT

We have implemented a software package for our IBM 360/370 system at SLAC, which we use in the development of stand alone LSI-11 software systems. Included in this package are:

* XASM11 - a cross assembler which handles a substantial subset of MACRO-11
* PL-11 - a cross compiler for a structured programming language designed around the PDP-11 instruction set
* FORTRAN - FORTRAN code is compiled under RT-11, and the object modules are transmitted to the IBM 360/370 for translation and integration
* XLCVT - a conversion program which takes as input standard DEC DOS/RT-11 object modules or LIBR files, or object modules produced by PL-11 and XASM11; it converts them to a form suitable for input to XLINK11
* XLINK11 - a cross linker which takes in relocatable object modules and outputs a DEC DOS/RT-11 absolute loader format module, or a format suitable for an IBM 360/370 PDP-11 simulator
* REFORM/EPROM - conversion programs which take as input a DEC DOS absolute loader format module generated by XLINK11 and convert it to a format suitable for downloading into an LSI-11 or for burning into EPROM's
* XREF - a cross reference program which takes as input a PDS of XLINK11 type object modules, and outputs cross reference tables and information on each of the modules
* WYLBUR execute files - several files of commands which may be executed from the terminal to create jobs in response to simple prompts
* KERNEL - a set of MACRO-11 and PL-11 routines which have been burnt into 2708 EPROM's and include a terminal emulator, a down line loader, and several utility routines for the LSI-11

All of this was accomplished by using the facilities already available on our IBM 360/370 system. Most of this package has been in regular use since Spring 1976, although new features are still being added.

The manpower cost of researching, organizing, developing, and integrating the tools has paid off well in terms of programmer productivity, source code readability, accessibility, and integrity.

INTRODUCTION

In Group A at SLAC we have had a need for a highly portable, rugged, reliable, compact, yet versatile and easily reconfigured module for testing and controlling equipment. After some thought and research it was realized that building a system based on a mini or micro computer with a minimum of bulky peripherals could fulfill the need at a reasonable cost. The lack of peripherals limited the programming development tools that could be used on the computer itself. Instead, therefore, we looked for ways of developing programs on the large SLAC IBM 360/370 complex (the 'TRIPLEX' see Figure 1) and then loading them into the mini/micro computer for execution. The existence at SLAC of an IBM 360/370 cross assembler and linker for a PDP-11 prompted us to choose LSI-11's to provide the computer base for the test and control modules. Having made the decision to use the TRIPLEX for our programming development, it became apparent that there are many advantages compared to using, for example, a fully configured PDP-11 development system (for which we could neither afford the money, the maintenance, or the space). Not the least of these advantages was our familiarity with using the TRIPLEX tools, in particular, the

*Work supported by the Department of Energy
(Contributed to the DECUS Spring Mini/Midi Symposium, Chicago, Illinois, April 25-28, 1978)

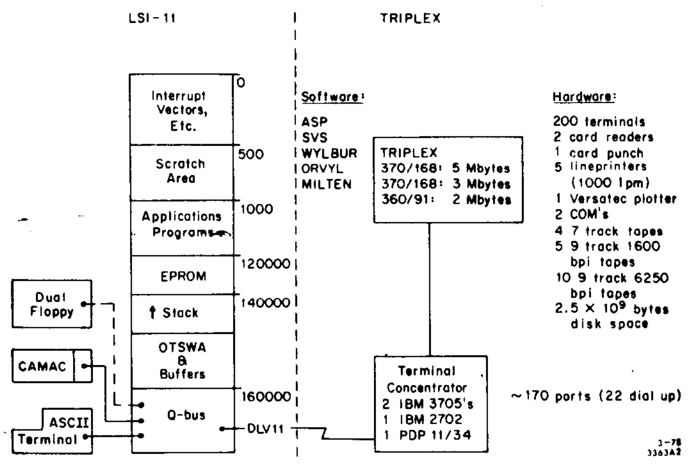


Figure 1: User oriented equipment on the SLAC TRIPLEX and a standard LSI-11 configuration.

powerful interactive editor and job entry system (WYLBUR[1]), the file system, data management facilities, and job control language. Other attractive facilities automatically available on the TRIPLEX are:

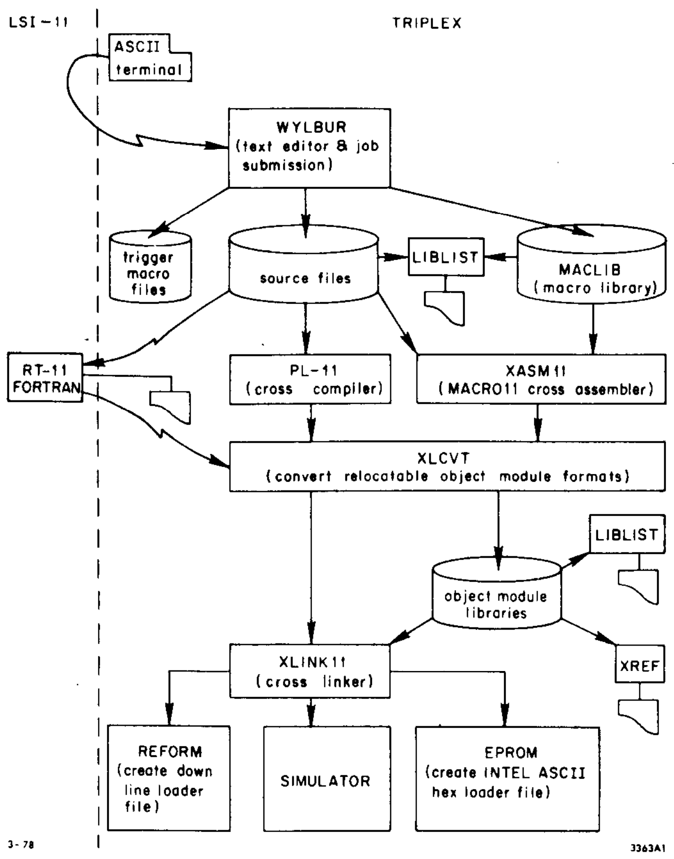
* permanently mounted mass storage facilities so that data sets are immediately and continuously accessible in a clean protected environment
* data set management facilities such as data set cataloging, archiving, auditing, date and time stamping, and backup
* availability of a full complement of peripherals such as high density (including 6250 bpi) tape drives, microfilm, microfiche, plotters, high speed printers with a variety of print trains, card readers and punches; these simplify transportation of programs, obtaining up to date listings, preparing documentation, etc.
* global access to the system from any site (in the laboratory, at home, in the office) with a telephone
* multi user facilities for program development
* widely accessible mass storage facilities for run time data acquisition storage

We therefore have developed an integrated package of software components for developing LSI-11 software on the TRIPLEX. The package includes a PL-11 cross compiler[2], a cross assembler, a cross linker, RT-11[3] support, and several utilities. The main components and their interrelations are shown in Figure 2. By providing the basic necessities in the EPROM's (a terminal emulator, serial interface device handler for communication to the TRIPLEX, and down line loader), we have at our disposal the full power of the TRIPLEX for system development anywhere we have an LSI-11 and a telephone. In addition the absence of LSI-11 peripherals increases the reliability of the system and reduces maintencance costs. All the hardware is standard and available commercially off the shelf. We now describe in more detail each of the major components of the software package.

THE EPROM KERNEL

The EPROM KERNEL is composed of a set of PL-11 and XASM11[4] routines which support the link to the TRIPLEX and the stand alone systems.

Included in the KERNEL are:



LSI-11
TRIPLEX
ASCII terminal
WYLBUR (text editor & job submission)
trigger macro files
source files
LIBLIST
MACLIB (macro library)
RT-11 FORTRAN
PL-11 (cross compiler)
XASM11 (MACRO11 cross assembler)
XLCVT (convert relocatable object module formats)
LIBLIST
object module libraries
XLINK11 (cross linker)
XREF
REFORM (create down line loader file)
SIMULATOR
EPROM (create INTEL ASCII hex loader file)
3-78
3363A1

Figure 2: Components of the TRIPLEX software support package for LSI-11 software development. The down line loader file is loaded into the LSI via a WYLBUR execute file.

* a terminal emulator program that has serial line (DLV11) device handlers for an ASCII terminal and the TRIPLEX link which supports both full and half duplex communication
* a down line loader which enables us to down line load an absolute load module created on the TRIPLEX
* a simple memory test and sizer routine which checks for simple problems such as loose chips in the memory boards and checks the size and location of the types of memory accessible (ROM, RAM, and device addresses)
* a boot strap for booting RT-11 from a floppy disk[5]
* ROLL - a facility for saving and restoring memory contents on a floppy disk
* a test to compute the check sum of each EPROM and compare it versus its recorded value
* several utility routines used by the EPROM programs and made available to stand alone systems via EMT and TRAP calls
* a facility for interpreting and executing various commands directed to the LSI-11 EPROM program; commands to execute EPROM programs are indicated by typing the ASCII escape character followed by the appropriate command character

These facilities enable the console terminal attached to the LSI-11 to be used as a normal terminal attached to the TRIPLEX and as an LSI-11 console terminal concurrently. They also provide basic support for the stand alone systems which are downloaded. The interrupt driven terminal emulator routines allow the user to continue communication with the

TRIPLEX while simultaneously running a stand alone system.

XASM11

The XASM11 assembler has been heavily modified to accept almost all of DEC's MACRO-11 language. The main restrictions are: immediate conditional assembly directives are not supported, and macros are treated differently.

Macros may be resolved either from source or from a macro library. The macros are expanded by passing the source through a modified version of IBM's level F assembler[6]. This means that macros are defined using IBM's syntax. The modifications mean that they can, however, be referenced in almost the standard DEC MACRO-11 fashion. The major modifications to IBM's level F assembler expansion facility were to allow:

* a colon in the label field
* an equals sign to be an operator
* the characters " ' & ! to appear in the operand field
* operator symbols to begin with a period or an equals sign

A few restrictions are placed on the MACRO-11 syntax by the macro facility. The most important restrictions are:

* To use an equals sign as an operator, it must be separated from the other fields by spaces, i.e. R1=%1 must be written as R1 = %1.
* Since the IBM assembler uses the & as an escape character to designate a macro time variable, when using & in constructs such as
  IFNDF EAE & FIS & EIS
  the & must be separated by spaces from the other symbols.

Using XASM11 with the macro expansion facility together with a WYLBUR execute file to change '=' to ' = ' in the operator field, we have successfully assembled 26,000 lines of DEC's RT-11 system code with changes being made to less than 20 lines. All changes were simple, and their necessity immediately apparent from the XASM11 error diagnostic. Also after the changes, the code was still compatible with DEC's MACRO-11 syntax. The assembler has also been used to generate code for a PDP-11/34 and a PDP-11/55.

PL-11

PL-11 is a block structured programming language with a rich set of structured facilities so that GOTO's can be virtually eliminated. It includes bit, byte, integer (16 bit), logical, real (32 bit), CAMAC type variables, arrays, FORTRAN like named common blocks, TEMPLATES to provide simple record structure facilities, local and global procedures, and the ability for global procedures to call or be called by DEC RT-11 and RSX/11 FORTRAN routines. A simple facility is also provided for separating the code and data sections of the procedures. All variables must be declared and full type checking is made by the compiler. It is designed around the PDP-11 instruction set for efficient compilation and execution. This allows the programmer to have the power, speed and data manipulatibility of assembly language and the readability of a higher level structured programming language. The cross compiler is written in ANSI FORTRAN and can be run on most large computers. A version of the compiler written in PL-11 also exists and can be run on a PDP-11.

We have slightly extended PL-11 with the addition of support for the Standard Engineering U type CAMAC crate controller, and a facility whereby comments can be inserted anywhere by enclosing them between " marks.

RT-11 FORTRAN SUPPORT

There are some applications where PL-11 or assembly language are insufficiently high level. One example was a relatively simple beam polarization measurement system that required DOUBLE PRECISION floating point to maintain accuracy. For this application the language of choice was FORTRAN due to its support of DOUBLE PRECISION arithmetic, our familiarity with it, and its availability. Since no FORTRAN cross compiler generating PDP-11 code existed, we decided to use the RT-11 FORTRAN compiler on one of our LSI-11's that had a floppy disk unit. Unfortunately the location where the beam polarization measurement was required was cramped, so it was undesirable to use valuable space for a floppy disk drive. Also the location was considered too dirty to allow a floppy disk unit to work reliably. Therefore we developed the source code using WYLBUR on the TRIPLEX, transferred the FORTRAN source files to RT-11 files on the LSI-11 with a floppy disk, compiled them, and transferred back the listings and object module files to the TRIPLEX. Object modules from the RT-11 FORTRAN object time system (OTS) library (FORLIB) and the system library (SYSLIB) were also transferred to the TRIPLEX. These RT-11 object modules were converted, saved, linked together and the resulting load module down line loaded into the LSI-11 by the methods outlined in the following sections.

In addition to the software development tools already on hand, all that was needed was a simple FORTRAN program (running under RT-11) to emulate a terminal, provide read and write access to RT-11 files, provide a terminal operator dialogue to request the file name and direction of transfer, and to encode/decode RT-11 .OBJ and .SAV files. This program performs file transfers between the WYLBUR active file and the named RT-11 file by sending commands to WYLBUR to either list the active file (transfer to RT-11) or to collect (in the active file) input from the terminal (transfer to WYLBUR).

In order not to invoke too much of RT-11, the following restrictions had to be observed in the polarization measurement program:

* All FORTRAN READ/WRITE statements are replaced by ENCODE/DECODE statements, with the I/O being performed by subroutine calls that in turn referenced EMT's contained in the EPROM's.
* The main program is in assembly language. It simply sets up the interrupt vectors for the errors (FIS interrupt, stack overflow, etc.) and calls the FORTRAN routine. This prevents some of the unwanted FORTRAN/RT-11 initializations.

In addition:

* The error EMT used to report execution time errors by the FORLIB routines had to be rewritten.
* The RT-11 FORTRAN OTS work area (OTSWA) is located in a location compatible with our memory utilization, initialized to zero, and the RT-11 OTS variable $AOTS is made to point to it.
* The FORTRAN OTS STOP and EXIT processors are simplified to type a message out and then HALT.

The main disadvantage with developing software this way is the file transfer time. For example at 2400 baud, it takes 7 minutes to transfer the 37 RT-11 blocks of SYSLIB.OBJ (481 lines of 80 hexadecimal characters each), or 35 seconds to transfer a FORTRAN source program of 88 lines (6 RT-11 BLOCKS). The number of file transfers during development can be considerably reduced by:

* using a compatible subset of IBM FORTRAN and RT-11 FORTRAN and then simulating the special input/output devices by separate subroutines using random number generators and testing the program to some level on the TRIPLEX
* using the RT-11 editor, FORTRAN compiler and LINKER, and executing on an LSI-11 with a floppy disk

Both of these techniques have drawbacks in that the first can't allow a full test, and the second ties up the single user LSI-11 for a long period. However, judicious use of these techniques has allowed us to integrate FORTRAN code with PL-11 and MACRO-11 code to create stand alone LSI-11 systems with reasonable success.

XLCVT

The object module format required as input to the cross linker (XLINK11^5) is not standard DEC DOS/RT-11 object module format. Thus object modules created by PL-11 or DOS/RT-11 programs must be converted before being input to XLINK11. XLCVT is a FORTRAN program that takes as input PL-11 or DOS/RT-11 object modules or RT-11 LIBR files and converts them to the XLINK11 format. It may optionally write the converted object modules, and XASM11 object modules into an OS partitioned data set (PDS) library specifically created for object modules. Each module is saved as a member with a member name equal to the object module name, and aliases equal to the entry points (if there are any). The modules are also stamped with the time, date, and the jobname which includes user identification. This information may be easily accessed via a batch job or from WYLBUR to enable users to ascertain the current status of any given object module.

The ability to handle RT-11 object modules has enabled object modules created by RT-11 FORTRAN to be transferred to the TRIPLEX, converted, saved in PDS libraries and/or to be linked on the TRIPLEX together with object modules produced by XASM11 and PL-11 on the TRIPLEX.

XLINK11

XLINK11 is a cross linker program that takes as input the PL-11, XASM11, or DOS/RT-11 object modules which have been translated and converted by XLCVT. XLINK11 processes object modules, relocates them, and assigns absolute addresses. During this process it may search through user specified PDS object module libraries to satisfy global references. The output of XLINK11 can be either in DOS absolute loader format or a format suitable for loading into a PDP-11 simulator^7 that runs on the TRIPLEX. A load map is also produced.

REFORM/EPROM

Since the serial line link to the TRIPLEX is set up for 7 bit character transfers, pure 8 bit binary DOS absolute loader modules cannot be transmitted over the link. Further, the terminal concentrator at the TRIPLEX (an IBM 3705) interprets certain 7 bit characters as control characters for its own use. The REFORM program overcomes this by converting each byte in a DOS load format module into 2 hexadecimal characters. Two different checksums are included in each record. The first is a checksum for the load address and the second covers the remaining bytes and comes at the end of the record. The use of two checksums means that the LSI-11 down line loader knows immediately that the load address is valid and so it can start loading the following data directly without buffering it (as it would if it had to wait until the end of the record to know if the load address was good). A further redundancy that the down line loader checks is that only the hex characters '0' through 'F' appear in the records. Since unsolicited messages may come from the TRIPLEX computer operator or other users during the loading procedure, it is important to be able to try and guarantee the integrity of the data transmitted. When an invalid character is noticed during down line loading, the remaining part of the record is printed on the console terminal. This allows the user to see any messages sent to him by other users even during down line loading. In such a case the down line loader then sends a checksum error record to the TRIPLEX so that the record can be retried.

The DOS absolute loader format modules may also be reformatted by the EPROM program into the Intel Hex ASCII format^8 and saved in an OS sequential file. In addition a one byte checksum of the contents of each EPROM is saved at the end of the EPROM's so that the contents of the EPROM's may be verified when they are in use. This file may then be used by any of several EPROM burners which are attached to the TRIPLEX, and Intel 2708 EPROM's burnt with its contents. EPROM can also be used to provide a hex or an octal dump of the core image that is created from a load module.

XREF AND OTHER UTILITY PROGRAMS

In a program containing many tens (and possibly hundreds) of separate modules referencing each other, it is very important to be able to understand the interrelationships between the modules. XREF is a program to aid in this understanding. It takes a PDS library of XLINK11 object modules, analyses each module, and provides an alphabetized table including for each module: its name, CSECT names and lengths, entry points, global symbol declarations and references, the time and date the module was last updated, and the user identification for the job that last updated the module. Next grid-like printouts are produced containing along one axis the global declarations (one table for procedures and entries, one for global data segments, and one for absolute data locations), and along the other axis the global references. Entries are made in the grid wherever a global procedure references a global declaration. Lists are also produced of any references which are not resolved from within the PDS library and any global definitions which are not referenced.

Up to date listings of programs are essential in most programming projects. LIBLIST is a program to read a PDS of source modules and list their contents together with an expansion of any text invoked by trigger macros. An index of lines containing selected text such as: .TITLE, .ENTRY, .GLOBAL, and trigger macros (see next section) is also produced; it is indexed by the module name the line occurs in, and the line number. Also included is the last date the source module was updated. LIBLIST can also produce formatted listings of a PDS of XLINK11 object modules.

Another utility (RT11TAPE) enables us to dump RT-11 and RXS-11 ASCII labeled tapes onto OS files on the TRIPLEX. This facility has been extremely useful in transporting programs to SLAC from other installations using DEC equipment. For example, our sources of the RT-11 system and MAINDEC diagnostics were all transferred from tape and now are saved in PDS's on the TRIPLEX.

EXECUTE FILES

Execute files are a sequence of WYLBUR commands that can be initiated and executed by typing a single WYLBUR command. They provide a simple way of putting together source code and JCL to create batch jobs to perform various tasks; e.g. update an object module (XLIB11), compile a complete source PDS and put the resultant object modules into an object module PDS (COMPILE), create a job to form a date stamped absolute load module (RUN11), list the contents of a source PDS (LIBLIST), or run the cross reference program on an object module library (XREF). The program we use on the TRIPLEX to transfer the reformatted absolute load module to the LSI-11 during the downloading phase is also an execute file (LOADER).

The execute files interface the user to the TRIPLEX facilities and speed up the program development since they eliminate much of the error prone typing of JCL.

The basic program development procedure we utilize entails:

* using WYLBUR to modify our MACRO-11 and/or PL-11 source code and to save it in an OS PDS
* executing XLIB11 to assemble MACRO-11 and/or to compile PL-11 and to save the resulting object modules in an OS PDS specifically created to hold the object modules
* executing RUN11 which creates a batch job to link together the object modules into an absolute load module suitable for downloading into an LSI-11
* downloading the LSI-11 via the LOADER execute file and the EPROM terminal handler and downloading program
* obtaining source listings via the LIBLIST execute file
* obtaining cross reference tables via the XREF execute file

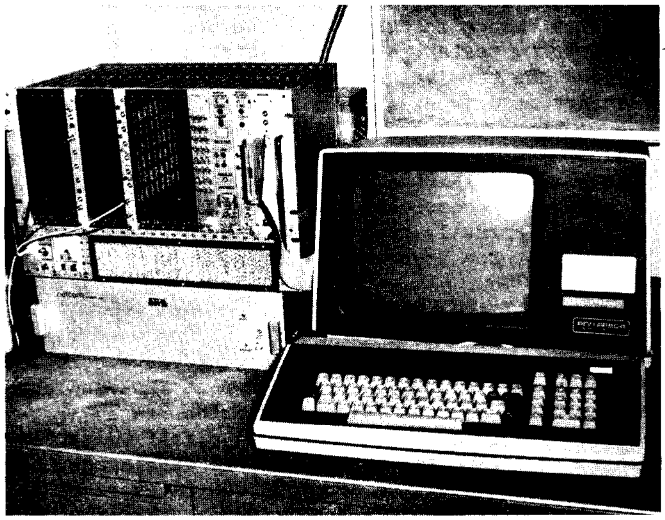
The execute files (COMPILE, XLIB11, and RUN11) have also enabled us to utilize a facility we refer to as a trigger macro. A trigger macro is a comment statement to the PL-11 compiler, however, to the execute files it is a command to copy in a piece of text from another OS data set. The form of the statement is:

"&COPY FROM $name"

where 'name' is the name of the test file to be copied in just after the "&COPY FROM $name" statement. Trigger macros enable us to maintain just one copy of a piece of code which may be used in several different places. Thus when changes are made to that text, one generally only has to recompile the routines which reference that particular trigger macro. Since the output from a LIBLIST contains a list of trigger macros referenced and where they are referenced, it is easy to find out which routines must be recompiled.

LSI-11 CONFIGURATION

The LSI-11's (see figure 3) are configured with a terminal, 2 serial line interfaces, a programmable



clock, 24K 16 bit words of RAM memory, 4K 16 bit words of EPROM, and a CAMAC interface. One of the LSI-11's also has a DSD dual floppy drive and supports RT-11. The lower 20K words of memory (0-117776 octal) is RAM, the next 4K (120000-137776 octal) is the EPROM area, and the upper 4K (140000-157776 octal) is RAM which is generally used for buffers and the system stack. The stand alone systems are loaded generally in the memory from 1000 to 117776 octal. They use a portion of the upper 4K (153776 down to 140000 octal) as their stack area and work space, and the rest of the upper 4K (154000 to 157776 octal) is reserved for clock counters, global flags, buffers, and EPROM program variables. This configuration of memory has several advantages:

* Since the EPROM contents are the same on all LSI-11's and do not cover the interrupt vectors, any stand alone system can run on any of the LSI-11's. This means that we always have readily available backup hardware.
* All stand alone systems are carefully designed so that they can be restarted without reinitializing everything. The EPROM's contain a routine (ROLL) to dump the contents of the program area (0-117776 octal) onto a floppy disk and restore that area from the floppy. Thus we can easily try a new version of a system out, and if it does not work, we can back up to the old version and continue on from where we left off.
* Our most frequent hardware problem has been caused by chips working loose on the EMM 16K RAM board. The EPROM's also contain a simple nondestructive memory test. Given a memory problem there is usually no way in which we can bring up RT-11 to run the diagnostics, and even trying would destroy the loaded program. By executing the EPROM memory test we can leave the contents of memory as it is, and then after we rule out a memory problem, we can still have the program available to use in our search for the problem.
* Also in general, if there is some kind of hardware problem, RT-11 simply will not work. By having the more detailed diagnostics on the TRIPLEX, we can down line load them via the loader in the EPROM's and try them out that way.
* Having the stack area above the EPROM's is very

advantageous when one is developing a system. If for some reason there is an error which would cause the stack to overflow, the stack simply runs into the EPROM area which causes a trap to location 4 or 10 and the program halts without destroying the program load. This gives one a much better chance to track down the problem and successfully restart a program after a crash.

* Since the terminal emulator in the EPROM is fully interrupt driven, it is available for use in communicating with the TRIPLEX even though a stand alone system may be executing. Systems requiring keyboard or TRIPLEX communication control need only assume control for the portion of the device drivers actually needed. In most systems this has been just the keyboard receiver.

DISCUSSION

Parts of the facilities described have been successfully used by 7 separate groups at SLAC and outside user groups from Yale and Florida State University. In the last one and one half years the authors have used the package in the development of three systems involving about 30,000 lines of PL-11 code and 3000 lines of assembler code. Briefly, the three systems are:

* CLOTHO which is a simple system that aids in the monitoring and calibrating of a polarized electron generator.
* LACHESIS which is a closed loop feedback system that reads and analyses beam position and energy monitors in the SLAC switchyard 'A' line. Based on this analysis it then adjusts the corresponding steering magnet currents and a klystron to hold the beam positions constant to a few tens of microns, and the energy constant to a few hundredths of a percent. This system has been in successful use for 9 months.
* ATROPOS which is a pulse height analysis system that contains facilities for reading, analysing and displaying data. In addition, ATROPOS can simultaneously log data to a job (ACQUIRE) running on the TRIPLEX and this TRIPLEX job can concurrently analyse the data and display its results either on the LSI-11 terminal or on the output facilities available on the TRIPLEX (the lineprinter and/or plotter). This system has been used by three different groups at SLAC for periods of several months in the development and testing of the components used in high energy particle detection equipment. We have also used it for developing and testing a hardware method of generating random bits.

The main disadvantages of the software development package currently are:

* The lack of hardware supporting a "transparent" mode of transmission to and from the TRIPLEX means that sending arbitrary binary data requires some software encoding and decoding.
* The limited transmission rates lead to long transmission delays. For example, when CLOTHO was originally connected to the TRIPLEX via a 300 baud modem on a telephone line, the down line loading took 32 minutes for 19K bytes. Fortunately most of our applications are located at sites which have twisted pair cables connecting them directly to ports on the TRIPLEX. Thus the normal line speed is 2400 baud at which rate CLOTHO can load in 4 minutes via an IBM 3705 port. More recently with the installation of a PDP-11/34 terminal concentrator front end to the TRIPLEX, we have been able to successfully connect to the TRIPLEX at 9600 baud and down line load CLOTHO in 1 minute 30 seconds. The transmission line speed and software decoding/encoding also limits the ATROPOS data logging rate to typically 60 bytes per second of logical data at 2400 baud.
* The performance of the TRIPLEX terminal support processors (MYLTEN/WYLBUR/ORVYL) under sustained full speed communication with computers instead of people may degrade their performance, particularly if several LSI-11's are logging data continuously at maximum rates.

Some of the transmission problems may be alleviated in about a year when plans call for the TRIPLEX to support a DECNET protocol via a PDP-11/60 front end.

In exchange for these disadvantages, the software package supports in a unified fashion multiple languages (MACRO-11, PL-11, FORTRAN, and work is currently in progress to integrate PASCAL, BCPL, and FORTH into the package). It also:

* has access to the sophisticated I/O devices of the TRIPLEX
* supports multiple users for many stages of the system development
* provides powerful TRIPLEX features such as WYLBUR, automatic archival, backup, and data set management
* is immediately available for use after powering up the LSI-11 since it uses EPROM's
* allows cheap hardware reproduction (including a CAMAC crate controller and interface, terminal, serial interfaces for the TRIPLEX line and terminal, LSI-11 power supplies, 24K RAM, 4K EPROM's and board, programmable clock, etc.) for less than $7000

ACKNOWLEDGEMENTS

We wish to acknowledge the help received in the development of this package from the following people: Sylvia Sund for her work on ACQUIRE and the EPROM program ROLL; Kenneth Eymann for his efforts to make the cross reference program XREF work on XLINK11 object module libraries; John Steffani for his extensive modifications of XASM11 and XLINK11; Tim Streater of CERN, Geneva, Switzerland, for making available manuals and providing distribution tapes of the PL-11 compiler; Bruce Hitson and Brigitte Bricaud for their work in integrating RT-11 FORTRAN into this package; Joe Zingheim for building and maintaining the LSI-11's and for useful discussions; and Professor Richard Taylor for providing support and encouragement. Work supported by the Department of Energy.

REFERENCES

(1) WYLBUR/370, The Stanford Timesharing Reference Manual, Third Edition (Nov. 1975).
(2) Robert Russell, PL-11: A Programming Language for the DEC PDP-11 Computer, Edited by T.C. Streater, CERN 74-24 (1974).
(3) RT-11 System Reference Manual, Digital Equipment Corporation, Maynard, Mass., (1976), Order No. DEC-11-ORUGA-C-D, DN1, DN2.
(4) S. Steppel and H.E. Syrett, XASM11/XLINK11, A PDP-11 Cross Assembler/Linker User's Manual, CGTM. No. 160 (1974).
(5) Diskette Memory System, General Product Descrip-

tion, Data Systems Design, Inc., Santa Clara, California.
(6) Assembler Language, IBM Systems Reference Library, order number GC28-6514-8.
(7) DEC PDP-11/20 Assembler and Simulator for the IBM System/370, PLAGO Project, Polytechnic Institute of New York, 333 Jay St., New York, NY 11201
(8) CGTM 190., On Connecting Computers to WYLBUR, Leonard Shustek, September 1977, Computation Research Group, SLAC, Stanford, CA.