DEVELOPMENT OF A PROGRAMMABLE ARRAY LOGIC PROGRAMMER USING A HOME COMPUTER

by GERT DANIEL JORDAAN

Dissertation submitted in compliance with the requirements for the

MASTER'S DIPLOMA in TECHNOLOGY

in the Department of Electronics at the

TECHNIKON O.F.S.

OCTOBER, 1988.

Supervisor: Prof. F.W. Bruwer

Co-supervisor: Mr. B. de Witt

ACKNOWLEDGEMENTS

I would like to thank the following persons without whose help this project could hardly have been completed:

The supervisor, prof. F.W. Bruwer, and co-supervisor, mr. B. de Witt, for help and guidance during the course of the project.

Mr. H.F. Coetzer for technical as well as philological assistance. It is really appreciated that time could be found in his very full schedule, for this assistance.

Dr. C.A.J. van Rensburg for his personal interest in the research project and for continuous encouragement and help.

Dr. J. van der Merwe for his assistance - in particular with respect to the registration and other administrative aspects of the project.

Miss M. du Toit who was largely responsible for the word processing.

For the guidance provided by my parents and the opportunities which they afforded me.

My children, Tania, Johan, Madelie and Lourens, whose main contribution was to have to forego much of my attention and time for such a long period.

Last, but not least, my wife, Christa, for her encouragement and understanding.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Recent Trends in Electronics

The development of programmable logic devices was a logical step in the ongoing process of increasing the integration of electronic circuitry by which the component count of electronic systems is decreased. The use of such programmable logic devices offers the following advantages:

1) Design cycles of systems are drastically reduced.
2) Production costs are reduced.
3) Inventory levels of chips are reduced.
4) Proprietory designs and signatures which cannot be copied by competitors (A Levy, April, 1986: 33).

The programmable integrated circuits are mass-produced, but small numbers of these can be programmed to fulfill the specialized functions required. This may prove to be a more economic solution than the development of customised integrated chips if fairly small numbers of these chips are required.

1.2 Problem Investigated

The main aim of this project is to investigate the programming principles and characteristics of programmable array logic devices, commonly referred to as PALs (although PAL is

a registered trademark of Monolithic Memories, it is used to describe any device within this family originating from any source), and to develop a relatively inexpensive PAL programmer that can be used in conjunction with a personal computer.

The only computer available for experimental use at the time of initiation of the project was a Commodore 64. For this reason it was decided to develop the programmer around the Commodore. It is appreciated that a computer such as the IBM PC, or a compatible, would have been much more suitable for this purpose. However, the Commodore proved to be quite up to the task with the only real problem being the slow execution speed of some of the programs written in BASIC. While this may have been unacceptable in a commercial model, it did not have any real bearing on the attainment of the goal of the project.

The programmer is capable of programming the majority of 20 pin programmable array logic devices, currently manufactured by Monolithic Memories and National Semiconductor. The design was limited to 20 pin chips but the same principles are valid for 24 pin devices. Incorporation of 24 pin devices in the project would only have resulted in the parallel development of two very similar sets of software, without really contributing towards the scope of the project.

Since the programming characteristics of PALs produced by other companies such as Texas Instruments, differ from those

mentioned, the field of application of the programmer is somewhat limited (IC Master, 1985, Vol 2: 5046).

1.3 Development of PAL Programmer

The development of the programmer can be considered under the following two headings:

1.3.1 Generation of Fuse Map

A programmable array logic chip contains a large number of fuses in a definite pattern. The desired logic operation can be realized by blowing some of these fuses in a predetermined pattern. This pattern is referred to as a fuse map.

A set of three programs, referred to as the fuse map generating software (and described in chapter 4), have been developed to generate such a fuse map according to the operator's instructions. The fuse map is saved on floppy disk and a hard copy of it can be printed for reference purposes.

1.3.2 Programming of Programmable Array Logic Devices

A PAL can be programmed to realize a logical operation as defined by a fuse map. Programming may be regarded as comprising two seperate problems as far as this project is concerned.

1.3.2.1 Programming Software

Software had to be developed to address and blow the fuses according to the fuse map. However, a PAL has to be pre-verified before programming to ensure that it is undamaged and can realize the desired Boolean expressions (fuse map). After programming, it should be verified that the required fuses have been blown successfully. If this is not the case, some corrective action should be taken by the programmer. These pre-verify and verify routines form part of the programming software as discussed in chapter 5.

1.3.2.2 Programmer Hardware

Although the computer can be programmed to generate the signals to blow the appropriate fuses as described, the voltage levels of these signals would not be suitable to be applied directly to the PAL.

For this reason a programmer capable of transforming these signals into suitable levels as required by the PAL was developed. This programmer is connected to the computer which exercises control over its functions.

1.4 Evaluation of Programmer

After developing the relevant software and constructing a programmer, it was then possible to design a number of PAL

circuits, to program the PALs and evaluate the performance of these devices. This would be an indirect evaluation of the programmer operation. This was carried out and satisfactory results were obtained.

The results of the project can however also be evaluated in a wider context than merely an increase in knowledge of programmable devices. A definite aim of the project was to realize the programmer design with a minimum of really sophisticated equipment and components - and at a minimum cost.

Although it is appreciated that the performance of such a system may fall far short of that of specialised imported equipment, it is felt that due to South Africa's unique position in the political and technological world such solutions to our problems should be investigated.

1.5 Importance of the Research Project

Smaller countries are experiencing difficulties in entering and maintaining a competitive basis in the electronics market. This is largely due to the very high degree of sophistication of the modern electronics industry and, in the case of South Africa, a serious lack of well trained manpower.

With the growing importance of field programmable logic techniques it is imperative that South African engineers and technicians keep abreast of these developments. This project

is primarily aimed at obtaining some specialized knowledge in this regard.

CHAPTER 2

PALs AS PROGRAMMABLE LOGIC DEVICES

The relatively recent development of a number of different families of custom and semi-custom logic chips, has caused what can only be described as a revolution in the electronic industry. A consequence of this was the development of programmable logic devices (PLD).

Programmable logic is an excellent choice of component if board size and complexity, system reliability, inventory considerations or speed is important (Meyer, February, 1988: 59). Functions with a large number of input variables fit particularly well into a PLD. Bus decoding and state machines both fall into this category (Meyer, March, 1988: 66). Although the scope of possible applications of programmable logic is virtually unlimited, the above defines in general terms its primary fields of application.

One measure of the efficiency of a PLD is an expression of how many fixed-function devices it can replace. In existing circuits a single programmable device can often replace four to six 7400 series chips. However, when designing new circuits incorporating programmable logic devices, a single chip can eliminate the need for 10 or more fixed-function chips (Jones, 1986: 437).

The use of large scale integration (LSI) introduced the importance of pin constraints on packaging. A system with

more gates but fewer packages might cost less than the minimal gate system (Klingman, 1982: 106).

The traditional practice of minimization of digital logic circuits using Boolean algebra, Karnaugh maps and for instance the Quine McCluskey method is therefore being limited to a certain extent. In this manner hazardfree circuits can be designed with a very slight increase in cost (Comer, 1984: 80). It certainly is preferable to design a hazard-free circuit by utilizing "don't care" terms in a PLD, rather than have an unreliable circuit and some unused gates in a PLD.

The following descussion of PLDs will be limited to programmable read only memories (PROM), programmable logic arrays (PLA) and programmable array logic (PAL) devices. All of these devices consist of a basic structure as shown in figure 2.1. The main characteristics of these components are as follows:

2.1 Programmable Read Only Memory (PROM)

PROMs are primarily used as memory elements rather than as combination logic circuits and was the earliest fuse-programmable device of the three under discussion. The OR array is provided with fusible links which are blown according to the data that the operator intends saving in the memory.

The input buffer and AND matrix constitute a full decoder - which, incidentally, is not the case with ordinary PLDs. Due to this, PROMs have difficulty accommodating large numbers of input variables since the size of the fuse matrix would become unacceptably large (IC Master, 1985: 455).

Due to the full decoder, the PROM is a universal logic solution in that all the product terms (minterms) of the input variables are generated, making it possible to implement any AND-OR funtion of these variables (Programmable Logic Databook by National Semiconductor Corporation, 1983: 24.6).

The maximum possible number of outputs are determined by the wordlength of the PROM. The Boolean expressions realized can consist of any number of terms, each term being a minterm (including all input variables).

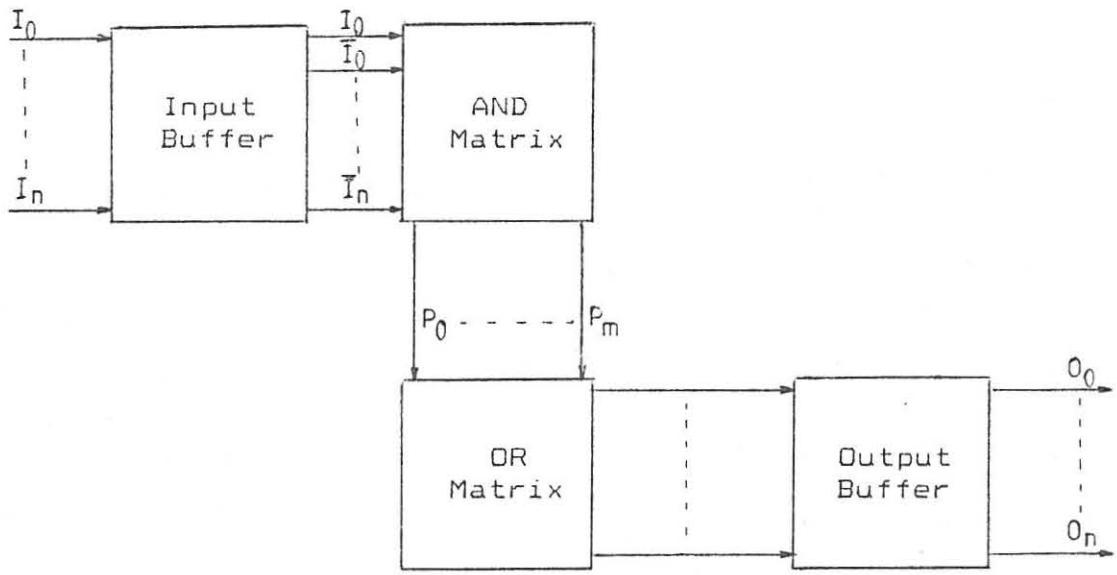


Fig. 2.1 - Blockdiagram of a PLD.

2.2 Programmable Logic Array (PLA)

This was the first fuse-programmable device aimed at the combination logic market, with the type 82S100, by Signetics Corporation, appearing in 1977. IBM was the first to incorporate PLAs in their computer designs - although these PLAs were metal-programmed, rather than fuse-programmed (Meyer, February, 1988: 62).

PLAs incorporate two fusible (programmable) arrays. The first steers inputs to the AND gates, while the second steers the product term outputs of the AND gates to the inputs of the OR gates. Both the AND and OR arrays in figure 2.1 are therefore programmable. Product terms are formed by opening the unwanted AND gate inputs by blowing the appropriate fuses (Triebel and Chu, 1982: 226). A product term with all fuses intact will "see" all inputs, and their inversions, and therefore remains low (Klingman, 1982: 91). Such a product term will not have any effect on the output.

The main disadvantage of PLAs is the fact that programming requires special equipment. The versatility of these devices is however beyond doubt. The recent improvements in PLA technology, and increased density, will result in an erosion of the bottom end of the gate array market (Putman, April, 1987: 25).

2.3 Programmable Array Logic (PAL)

The manufacture of PALs was a more recent development than that of PROMs and PLAs. One of the first major applications of PLAs was by Data General Corporation in their Eagle project during the late 1970's. At that time the supply of these components was still very limited with Monolithic Memories as leaders in this field.

The basic structure of a PAL can also be illustrated by figure 2.1. In this case the AND matrix can be programmed. The OR matrix is fixed and cannot be programmed. In a PAL, specifying the OR gate connection becomes a matter of device selection rather than of programming - as with PLAs (Programmable Logic Databook, National Semiconductor Corporation, 1983: 24.10).

The following initial observations should therefore be made with respect to the selection of a specific PAL device:

1) The number of inputs and outputs of the circuit should be fewer than, or equal to, the number of inputs and outputs of the PAL.
2) The number of, and distribution of, product terms available in the PAL must be considered (Jay, November, 1986: 67).

With PALs, the maximum possible number of inputs and outputs are not always fixed. The outputs of some devices, such as with the type 16L8, are fed back as inputs. Such an output

pin can be used as an input if the appropriate output inverter is in high impedance state (Programmable Array Logic, Elektor, May, 1985: 5.54). Obviously such an increase in inputs causes a decrease in the maximum possible number of outputs. It is however the versatility of the component that is of importance.

This feedback principle enables the utilization of a PAL in multi-level logic applications, and not only for realizing a number of sum-of-products. The feedback connections of registered outputs also allows the configuration of the PAL as a state machine. Since all register output cells within a single package share a common clock, it is ideal for the design of, for example, synchronous counters.

Although it is conceivable that higher densities of integration, and thus increased PAL capabilities, will become possible in future, studies have shown that if the inputs and outputs of PALs are increased, a law of diminishing return sets in. The more the chip is expanded, the smaller the useable proportion (Baker, April 24, 1987: 30).

Programmable array logic devices are not susceptible to causing glitches at its outputs, as is common with PROMs (Programmable Array Logic Leads to Flexible Application of 8-Bit Wide Memories, In: PAL Handbook by Monolithic Memories, 1983: 8.40). The three-state outputs can be used to avoid glitches due to loading differences of different gates by enabling them after the outputs have settled.

Some three-state outputs are controlled by a seperate product term. This term is used to enable the three-state buffer, which in turn gates the summation term (OR gate output) to the output pin (LSI Databook by Monolithic Memories, 1985: 5.8).

In order to protect the design of a programmed PAL, it is supplied with security fuses. Once the programming of a PAL is completed (and verified), these fuses can be blown. This disables the verification logic, providing a significant deterrent to potential copiers.

The normal operating voltage of PALs is 5 volt. The programming voltage, at 11,75 volt, is however significantly higher. This programming voltage level ensures that power supply glitches will not set the PAL in the programming mode (Meyer, March, 1988: 65).

PALs are especially suited for application as "glue chips" between standard blocks. A replacement ratio of one PAL for 5 to 10 small/medium scale integration chips is common. It has been reported by D. Carlson of DEC, in Electronics of October 6, 1982, that the use of PALs in the VAX 11/730 computer helped reduce board area for the central processing unit by a factor of 4, and halved component cost compared with MSI (Chakravarty and Gottlieb, February, 1986: 11).

Extremely fast switching speeds are achieved with PALs and propagation delays are very much shorter compared with the

discrete transistor-transistor-logic version of almost any medium complexity circuit (Barwise, January, 1987: 24).

2.4 PLD Technology

Until recently programmable logic devices have always been produced using fusible bipolar technology. A number of materials are used as fuses. The companies "Monolithic Memories" and "National Semiconductors" use titanium-tungsten (PAL Handbook by Monolithic Memories, 1983: 1.11 and Programmable Logic Databook by National Semiconductor, 1983: 24.11). Advanced Micro Devices uses platinum-silicide in their PALs (IC Master, 1985: 4901), while other manufacturers have opted for various other materials.

Some modern programmable devices are produced using CMOS technology. Erasable types, using ultra-violet light for erasure, use floating gates (Meyer, February, 1988: 66 and 67), while electrically erasable types are also being used. The advantages of these new developments are primarily lower power dissipation and an improved development cycle by means of the erasable types.

An alternative to PALs are hard array logic devices (HAL). HALs are identical to PALs except that it is mask-programmable and not fuse-programmable. PALs are then used during development of a system. Once the design has been finalized, the PAL data is supplied to the PAL/HAL manufacturer, who produces the HALs to the customer's specifications.

Existing PLDs come in a variety of package sizes with 20 and 24 pin versions being the most widely used today. These devices are equivalent to a 200 - 300 gate array. Upgrading to 40 and 84 pins in leadless chip carrier packages or pin grid arrays will permit gate densities of over 1000 gates. These PLDs will therefore usefully serve the lower end of the gate array market, especially when very large quantities are not required (see section 2.2). Standard cells may be a better alternative to PLDs for higher volume requirements and high gate counts.(Chakravarty and Gottlieb, February, 1986: 10 and 11).

Although gate arrays are also a potential solution for many complex applications, they pose their own set of challenges and limitations. Among these are high development and production cost, long turnaround times and the fact that the manufacturer must intercede between the system designer and array production (Putman, April 24, 1987: 25).

2.5 Programming of PLDs

Most modern PLD programmers are used in conjunction with programming software such as "PALASM". The software prepares a Jedec file which is transferred to, for instance, a PAL programmer through a RS232 interface.

Manufacturers of some programmers now build development software directly into their machines (Levy, April, 1986: 32). The programmer developed as part of this project, and

described in chapter 6, however, differs considerably from both these two solutions.

2.6 Summary

The PLD explosion, as currently experienced in the electronic industry, is opening many new and exciting avenues to the logic designer.

The development however, takes place in so many directions and on so many levels, that help for the designer becomes highly desireable. An interesting development in this regard is the "Isdata PLD database" which is claimed to be a help to the designer in selecting a specific device for a certain purpose (Intelligent Help for Circuit Designers Overcomes PLD Chaos, In: Dataweek, May 6, 1988: 11).

The above may not be regarded as a comprehensive discussion of PLDs but serves as a suitable introduction to the closest alternatives to PALs, while the main concern of this project is directly with PALs. Logic sequencers (FPLS), generic array logic (GAL) and logic cell arrays (LCA), to name but a few, are other possible elements which could be considered as PLD alternatives. Yet, PALs should certainly be considered as a worthwhile component, fullfilling an important function in the electronic industry of the day.

CHAPTER 3

PROGRAM CONSTRAINTS, STYLE AND TESTING

The following had to be considered before the programmer software could be developed:

1) What programs were required.
2) The inputs of each of these programs.
3) The outputs of each of these programs.
4) What standards and conventions were to be followed.
5) The availability of funds within the context of the project.

3.1 Constraints

The project had to be realized within the following constraints:

3.1.1 Finances

Since the aim was to investigate the viability of developing an inexpensive PAL programmer, attempts were made to keep the total cost of the programmer at an absolute minimum.

3.1.2 Hardware

Due to the financial constraints it was decided to develop the system around a Commodore 64 microcomputer. The

programmer is connected to the computer by means of the data bus extention socket on the back panel of the computer.

3.1.3 Language

A high level language should be selected for its ease of use and readability if the program designed in this language executes fast enough (Aron, 1974: 93). It was decided therefore to write the fuse map generating software in BASIC. The total processing time proved to be fairly short (between 3 and 6 minutes depending on the operator) and this was considered to be acceptable. However, due to certain pulse timing parameters which could not be met with BASIC, both BASIC and assembler language were used to prepare the programming software.

3.2 Program Goals

Certain goals were set with respect to the reliability, efficiency and generality of the software.

3.2.1 Reliability

Software is reliable if it meets its initial specifications and performs as specified (Sommerville, 1982: 7). This was accepted as a general principle. It was also accepted that the initial specifications of the programs were in fact complete and correct.

Reliable programs include code to detect the occurance of an exception (where an exception is the occurance of an error, such as invalid data of some kind) if it has been anticipated by the programmer. This should then indicate the need for corrective action to be taken. The code need not however be self-correcting. A programmer may decide to ignore rejection (if data is outside the acceptable range) if a system is to be used by one operator only (Van Tassel, 1978: 258). This principle certainly also holds true for this type of program which is to be used only by a certain knowledgable group of persons.

These principles were adhered to during the program development and the programs should act predictably if supplied with invalid data. As it was assumed that only operators with a fair amount of knowledge of Boolean algebra would utilize the system, basic Boolean conventions were adhered to (e.g. "+" sign denotes OR operation).

3.2.2 Efficiency

Efficiency should always be an important aim for any programmer, even though it may be considered to be less so than reliability. The main reasons why reliability is more important than efficiency are as follows:

1) The cost of equipment has decreased.
2) An unreliable program is practically useless.

3) Inefficiency is predictable, unreliability is not.

The preparation of Boolean expressions which are to be realized in a PAL is a time consuming process. The execution time of the fuse map generating software is in fact much shorter than the time required to prepare for its execution and is therefore not all that important. The program execution is only one step (and indeed the least time consuming) in the process of implementing a PAL in a circuit - hence the relative unimportance of efficiency.

3.2.3 Generality and Ease of Use

The programs were written in such a manner as to facilitate use both by the person generating the fuse map as well as the programmer making modifications to the programs. The first is catered for by leading him through the program with suitable prompts and requests. Modifications can only be made to a program if the programmer has a thorough understanding of the program. Free use is therefore made of comments (REM statements) which greatly improves the readability of the programs as well as identifying the different procedures in the programs.

Wherever possible variables were used as parameters in order to improve generality of the programs. It was therefore easy to modify the programs as the need arose during development.

In another attempt to maintain the generality a minimum

of machine dependant code was used even though it might have improved the structure of the program (this obviously does not hold true for the machine code routines). It should therefore be possible to run the fuse map generating software on most other microcomputers with a minimum of change. This is also the reason why use is not made of, for instance, Simon's Basic, which would have decreased the generality of the programs (but would have improved the program structure).

3.3 Structured Programming

Because BASIC is not considered to be suitable for a very structured program design, these programs may not be considered to be very well structured by, for instance, PASCAL programmers.

The number of GOTO statements (generally considered as indicating a lack of structure in a program) could have been decreased if two-armed-conditionals (if-then-else) were available. The Commodore 64 is however only capable of one-armed-conditionals.

Sommerville (1982: 123) holds that a circumstance where the use of a GOTO may be justified is to exit from a procedure in case of an exception. This principle was employed in a number of cases where program flow had to be diverted to certain program sections.

A top-down design strategy was applied in writing the

programs. The basic design was initially made by means of structure charts and the detail was added as the design progressed. The top-down design principle is considered to establish the logical structure of the solution before it decides on the detailed elements of the solution (Aron, 1974: 96). An advantage of top-down design is that it simultaneously provides complete design control and logical modularity (Aron, 1974: 128).

3.4 Modular Programming

Modular programming should be based on identifying all the things a program must do. With this in mind the programs were modularized by sequence of execution. This means breaking a problem into the major activities to be performed sequentially from the start of program execution.

Advantages derived from modular programming include:

1) Ease of testing.
2) Ease of reading.

Certain goals with respect to the modules have to be attained in order that these advantages may be realized. In particular;

1) Modules must be correct irrespective of context in which it is used.
2) Modules must be combined to form larger programs without knowing the content of each module.

Modules should be independant of the following:

1) Source of input.
2) Destination of output.
3) Past history of module.

The only disadvantage of this style of programming is that it may lead to a slight increase in the length of the programs. Modularization in this set of programs certainly improved the readability of the programs.

In connection with readability, a short explanation of why the variables were named so indescriptively follows: With the Commodore 64 variable names can be of any length but only the first two characters are considered significant in CBM BASIC (Commodore 64 Programmer's Reference Guide, 1984: 7). The author is of the opinion that the use of more descriptive variable names increases the danger of effectively giving more than one variable the same name (e.g. the Commodore 64 would not be able to distinguish between variable NAME and variable NATURE).

3.5 Program Testing

Testing should not be confused with debugging. Testing ensures correct results under all conditions, whereas debugging are the steps taken to correct errors becoming apparent during testing. Debugging may, for instance, entail the addition of statements which print intermediate results at appropriate places in the program. These statements are

removed eventually after debugging. If the program is written in a language with a trace facility, this may also be used for debugging to monitor program flow (the Commodore 64 instruction set does not include a TRACE instruction).

As with the design, a top-down approach was used in testing the programs. The advantages of top-down testing of programs are as follows:

1) More data is added as the program develops.
2) The main logic is tested early and continually.
3) Testing is distributed.

The two major classifications of software testing are functional and structional testing. Functional testing is conducted without any consideration of the internal structure of the program. Possible input data is divided into classes in such a manner that if one test case from a class is executed and fails to detect an error, then any other test case in this class would also fail to detect an error. This technique is designed essentially to reduce the number of tests necessary.

A major weakness of functional testing is that there is no way to be sure that testing is complete. The aim of structural testing is to ensure that the entire program code has been executed during testing (Herington et al, 1987; 14 and 15).

Each test case should represent a different class of data

and test data should obtain the maximum amount of information from each run (Van Tassel, 1978; 253). These two conditions can only be satisfied if test data is carefully selected, which in turn minimizes the number of test runs required. Van Tassel (1978: 272) underscores the importance of planned testing by stating that if an effort is not made to develop systematic test cases, it is doubtful that any true indication of status or success of testing will be found.

Eventually any programmer has to ask himself how well his program has been tested. Ideally every instruction should have been tested at least once. Another way of evaluating the testing performance is by determining:

1) The percentage of code executed.
2) The percentage of number of branches tested in both directions.

The programs developed as part of this project have been thoroughly tested and debugged. Care has been taken to ensure that all code was tested and the test cases were selected in such a manner as to verify the program performance under all possible input conditions.

As far as the fuse map generating software is concerned the testing had to ensure that:

1) The PAL File program should generate an accurate file of the appropriate PAL.

2) The Fuse Map 20 program should generate a fuse map realizing the Boolean expression supplied by the operator.

The set of programs under consideration were indeed subjected to such tests as outlined above and performed as intended. During execution of Fuse Map 20 some basic data regarding the specific type of PAL to be programmed are required. For this purpose a personality file has been compiled, and saved on floppy disk, for each type of PAL. A program, referred to as PAL File, was written and used to create these personality files.

Because of the importance of an accurate personality file, a seperate program, entitled "Testing PAL File", was written to check the personality file (see 4.2). At the same time this also proved the correctness of the PAL File program.

The programming software was evaluated by the operation of PALs programmed with it. Initially the programming waveforms generated by these programs were measured and evaluated (see figure 5.12). In this manner the total operation of the programmer plus software were found to be as required.

3.6 Program Documentation

Careful consideration went into determining the amount and format of program documentation to include in this document. When documenting a system for a user it is neccessary to

communicate with the right perspective by recognising the user's position and function (Katzin, 1985: 5).

A combination of flowcharts and pseudo code listings were included. Since the programs were to be coded in BASIC, the pseudo code format was intentionally written so as to approach BASIC in order to simplify coding (Dvorak and Musset, 1984: 7).

Detailed operator's instructions are not treated seperately because program execution generates prompts wherever the operator has to supply data, etc. Where neccessary the required data is described in the chapters dealing with the software.

CHAPTER 4

FUSE MAP GENERATING SOFTWARE

The fuse map generating software consists of a set of three programs, viz.:

1) PAL File
2) Testing PAL File
3) Fuse Map 20

4.1 Pal File

The aim of this program is to generate a personality file for each type of PAL with which the programmer is to be compatible. This file is accessed by the Fuse Map 20 program during generation of the fuse map. In this manner it was possible to develop a fuse map program which is independant of the type of PAL to be programmed.

This concept was decided upon since the set of programs would be compatible with new PAL's to be announced in future. It is now possible to extend the programmer's capabilities by generating personality files corresponding to such new PALs.

When initializing the Fuse Map 20 program, the operator has to identify the type of PAL he intends using. The relevant PAL data is now automatically read from the personality file. The personality file is in the form of a sequential

file since the running of Fuse Map 20 involves a uniform series of operations on all the file records (Brinkman, 1984: 230). The sequential file structure has the advantage of simplifying the Pal File program.

The data to be supplied whilst running the PAL File program is obtained from the appropriate data book. The data must be arranged as follows:

1) Control pins:

These are directly read off the appropriate data sheet.

2) Input pins:

It should be ascertained to which input line each input pin is connected. This input line should then be specified when prompted whilst compiling the personality file.

3) Output pins:

When identifying a pin as an output, the operator is prompted for the relevant product line. This is the smallest numbered product line input to the OR gate connected to this output pin. This is followed by the number of product lines connected to this OR gate. In those cases where the output has an enabling product line (e.g. the 16L8), care should be taken not to supply the product line number of this line instead.

If the output is supplied with feedback capabilities this should be stated, followed by the input line to which this feedback line is connected.

Although this may appear to be a complicated procedure, it

should be borne in mind that it is executed only once for any type of PAL. After this the file will be available for processing by Fuse Map 20.

Figure 4.1 is a flowchart of the Pal File program.

A listing of the PAL File program is supplied in Appendix A.

4.2 Testing PAL File

It is imperative that the personality file data must be correct if the Fuse Map 20 program is to prepare an accurate fuse map. The Testing PAL File program has been developed to verify the accuracy of the personality file.

Execution of this program reads and displays the sequential data file as stored during the PAL File program. The operator can now compare this with the PAL data as supplied by the manufacturer and if any errors are detected the PAL File program can be rerun.

The operator is guided through the program by suitable prompts. Figure 4.2 is a flowchart of this program whilst Appendix B is its listing.

4.3 Fuse Map 20

This is a relatively complex program requiring a fair amount of operator intervention during execution. It also accesses the PAL personality file generated beforehand (as

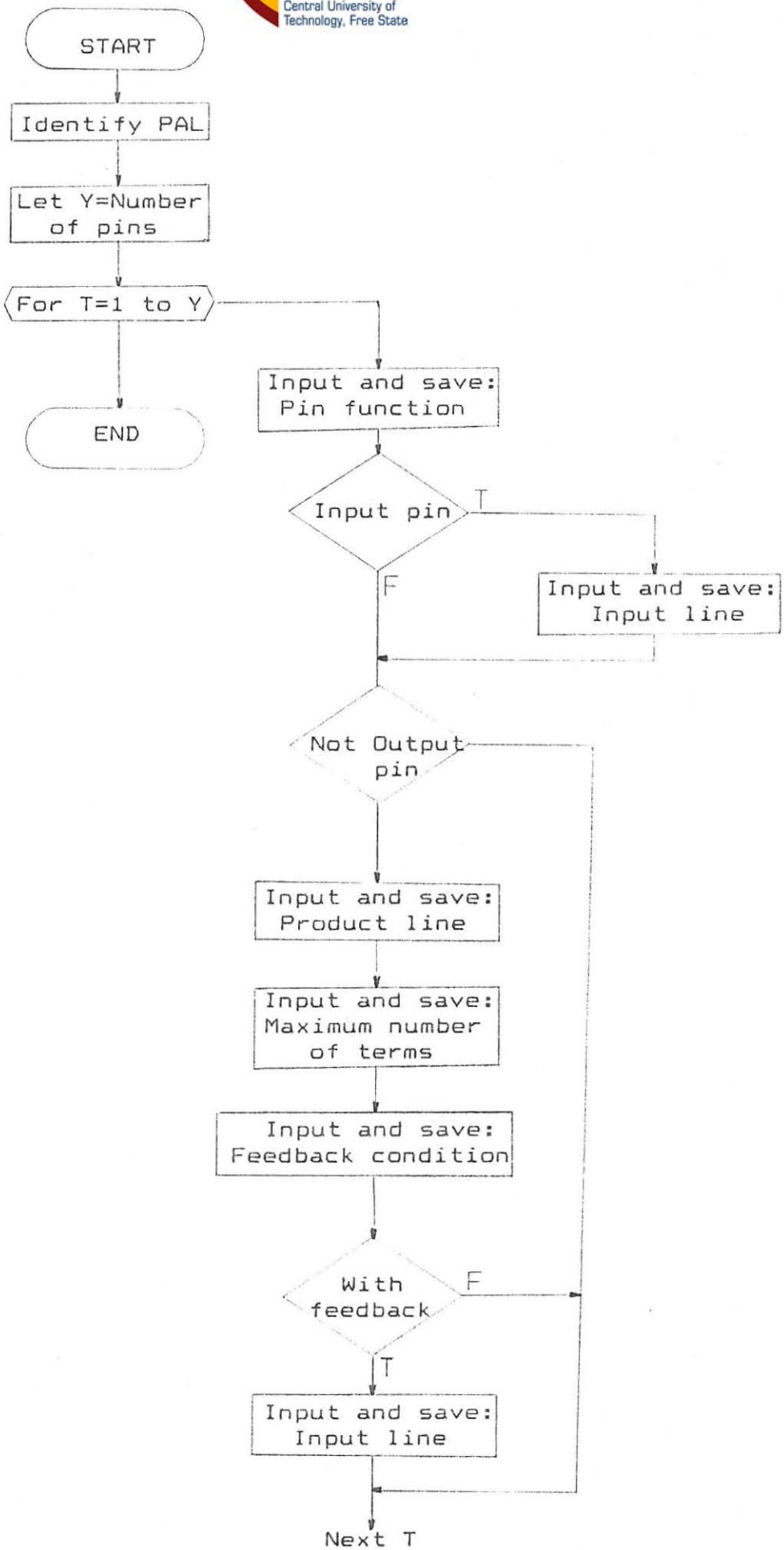


Fig 4.1 - Flowchart of PAL File

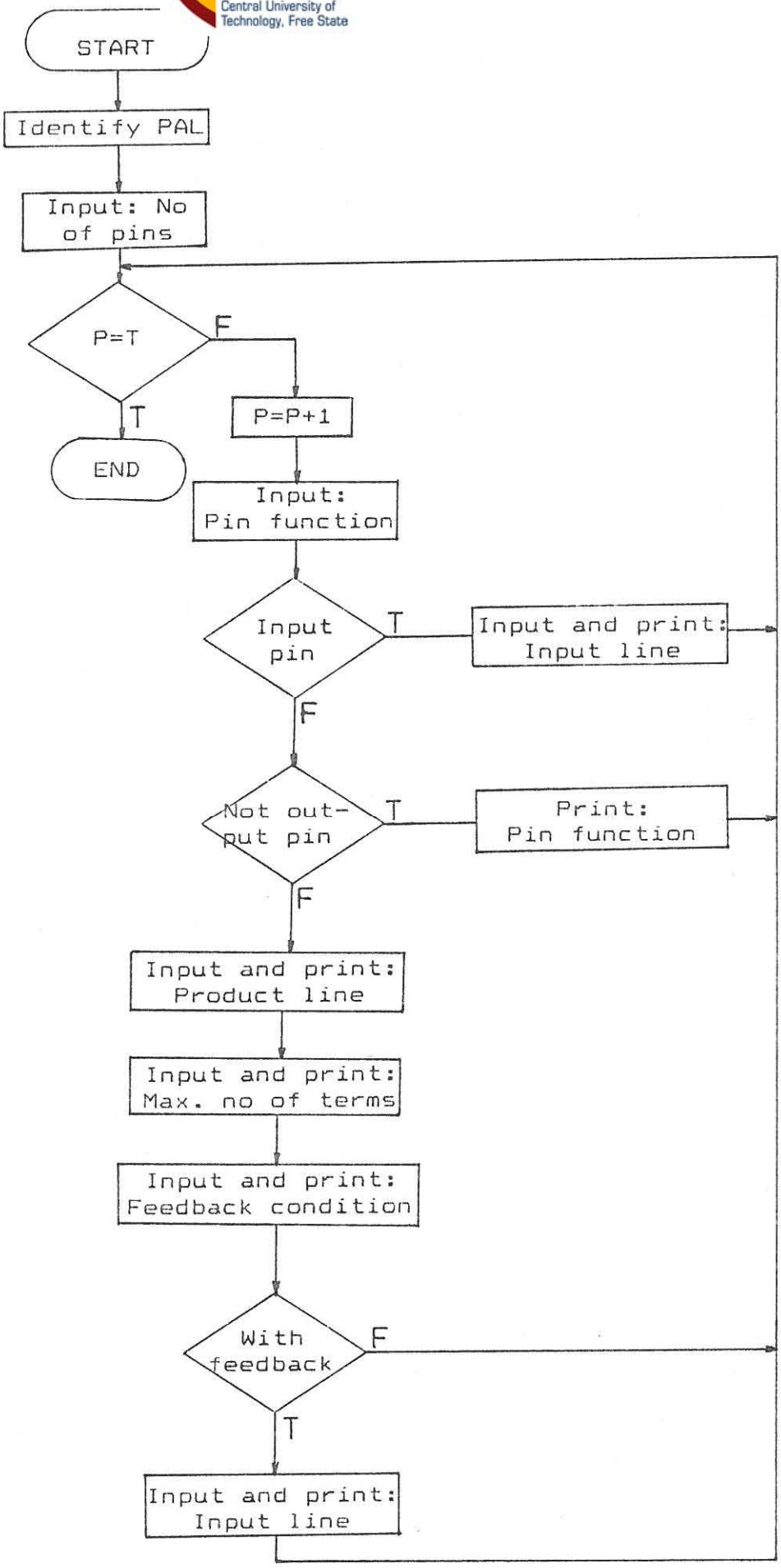


Fig. 4.2 - Flowchart of Testing PAL File.

discussed).

The basic structure of the Fuse Map 20 program is as shown in figure 4.3. From this it can be seen that the program consists of the following modules:

MODULE	LINE NUMBER
1) The PAL to be used is identified.	10-80
2) The circuit to be realized is identified.	90
3) The appropriate PAL personality file is accessed and variables are assigned to the pins of the chip.	340-370
4) A summary of pin variables is displayed.	260-330
5) The operator inputs the required Boolean expressions.	380-470
6) These expressions are transformed into a preliminary fuse map and checked for variable validity and that the maximum number of terms possible have not been exceeded. If any of these tests are not satisfied an error message is printed and a new Boolean expression is requested.	1720-2100
7) The fuse map is processed for don't care conditions and phantom fuses.	480-890
8) The fuse map is displayed.	900-1060
9) The fuse map is saved as a sequential file.	1070-1180
10) A hard copy of the fuse map is printed.	1190-1650
11) The program terminates.	1660-1670

Some effort went into displaying sufficient information regarding operator intervention during execution of the

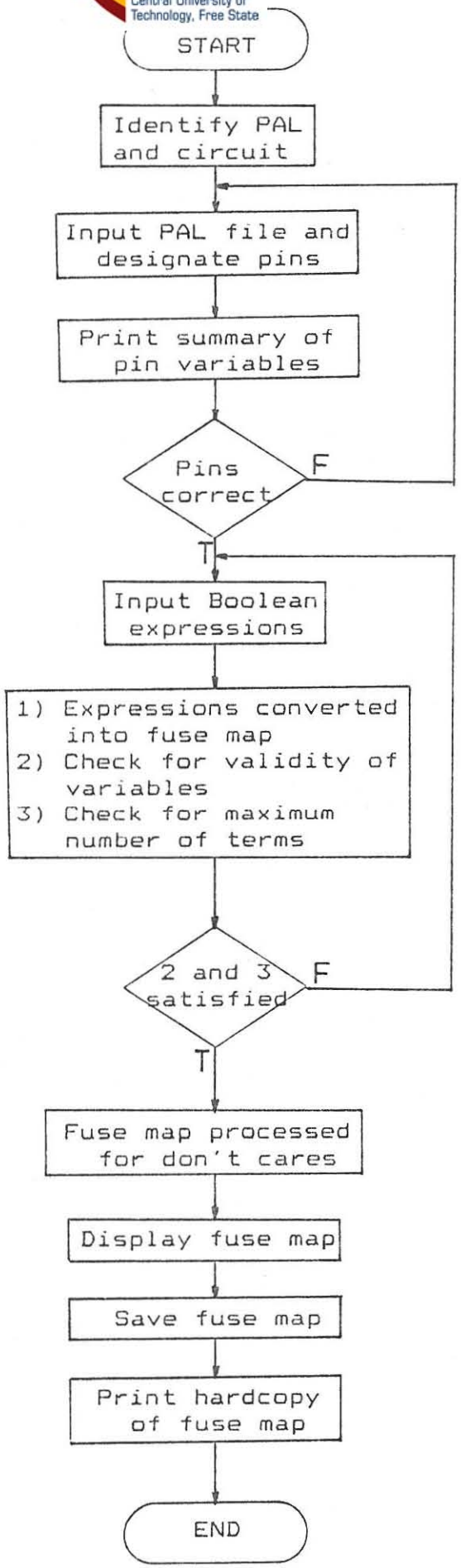


Fig 4.3 - Flowchart of Fuse Map 20.

program. Where neccessary further details will be supplied when the appropriate program segment is discussed.

4.3.1 PAL, Circuit and Pin Variables Defined

The first step in running this program entails the identification of the type of PAL to be used, naming the circuit and designating the pins of the PAL in this circuit.

The variables assigned to the PAL pins may be only one character long, except in those cases where active low output pins (such as the 16L8 or 16R4) are named. In such cases these outputs should be identified as active low (e.g. !X = NOT X). In this manner the deduction of the Boolean expressions becomes easier. If this practice is not adhered to in circuits using internal feedback, problems will arise with the generation of the fuse map. During this program segment the PAL personality file is also read as shown in the following pseudo code listing.

```
1.  Dimension variable arrays.
2.  Do until X>20
        2.1  Print X
        2.2  Input:  PAL pin function (A$)
        2.3  Print:  PAL pin function
        2.4  F$=A$
        2.5  If Input pin then     --Input pin routine
             True:  2.5.1  Input:  Input line
                    2.5.2  Input:  Pin variable
```

2.5.3 Goto 2

2.6 If Output pin then --Output pin routine

True: 2.6.1 Input: Product line

2.6.2 Input: Maximum number of terms

2.6.3 Input: Feedback condition

2.6.4 If with feedback then

True: 2.6.4.1 Input: Input line

2.6.5 Input: Pin variable

2.6.6 Goto 2

2.7 Print: PAL pin function -- Control pin routine

2.8 Goto 2

3. END of routine

Following this, a summary of the variables as assigned, is displayed. If the operator should now find that he is not quite satisfied with his selection, the program can repeat the procedure and new variables may be assigned.

4.3.2 Input and Conversion of Boolean Expressions

The Boolean expressions supplied by the operator during execution of this program segment should conform to the following specifications:

1) The normal Boolean algebraic convention of a "+" sign to denote an OR operation is maintained.

2) As is accepted practice in Boolean algebra, the absence of an operational sign is assumed to denote an AND operation (Tocci, 1980: 26).

3) An exclamation mark indicates an inversion, e.g.

"!A" denotes "NOT A".

4) Some PAL outputs need an enabling term (this is not to be confused with an ENABLE input pin on a PAL). An example where this may be required is with a type 16L8 PAL. The format of such an expression is as follows:

X=#AB(ACD+EFG)

This is equivalent to the Boolean expression

X=AB(ACD+EFG)

where AB is the enabling term.

Each Boolean expression is immediately converted and tested for validity. The first step in the conversion process is to identify the appropriate output pin and product line. The following is a pseudo code listing of this process.

```
1.  Input:  OP$                          -Input expression
2.  If OP$<>"END" then
    True:  2.1  X=11
           2.2  VAR$=LEFT$(OP$,1) -Identify variable
           2.3  If VAR$="!" then  -Active low output
                True: 2.3.1  VAR$=LEFT$(OP$,2)
                      2.3.2  AL=1
           2.4  If X>20 then
                True:  2.4.1 Print:"Undefined variable"
                       2.4.2  AL=1
           2.5  If VAR$<>P$(X) then
                True:  2.5.1  X=X+1
                       2.5.2  Goto 2.4
```

2.6 L=PL%(X) -Identify product line

2.7 OUT=X -Identify PAL pin

3. END of output pin routine

The input part of the expression can now be converted. Figure 4.4 is a flowchart representation of this process.

Each character is converted by identifying its input line. Whenever a "+" or "(" is identified it indicates the end of the term and the next term is then referred to the next product line. A "#" sign indicates an enabling term. This term is realized by means of a special product line.

In this manner all the expressions are read in and converted into a preliminary fuse map. However, no provision has been made for don't care fuses or phantom fuses at this stage.

4.3.3 Don't Care Fuses

Those product lines on which all the fuses are blown assume the logical high state. During compilation of the preliminary fuse map the program operates in such a manner as to switch all unused product lines to this condition. This process should however be reversed to prevent these PAL outputs from being latched in the active logic condition. The following program segment is designed to identify all such fuse locations and to change those fuses from "to be blown" to "not to be

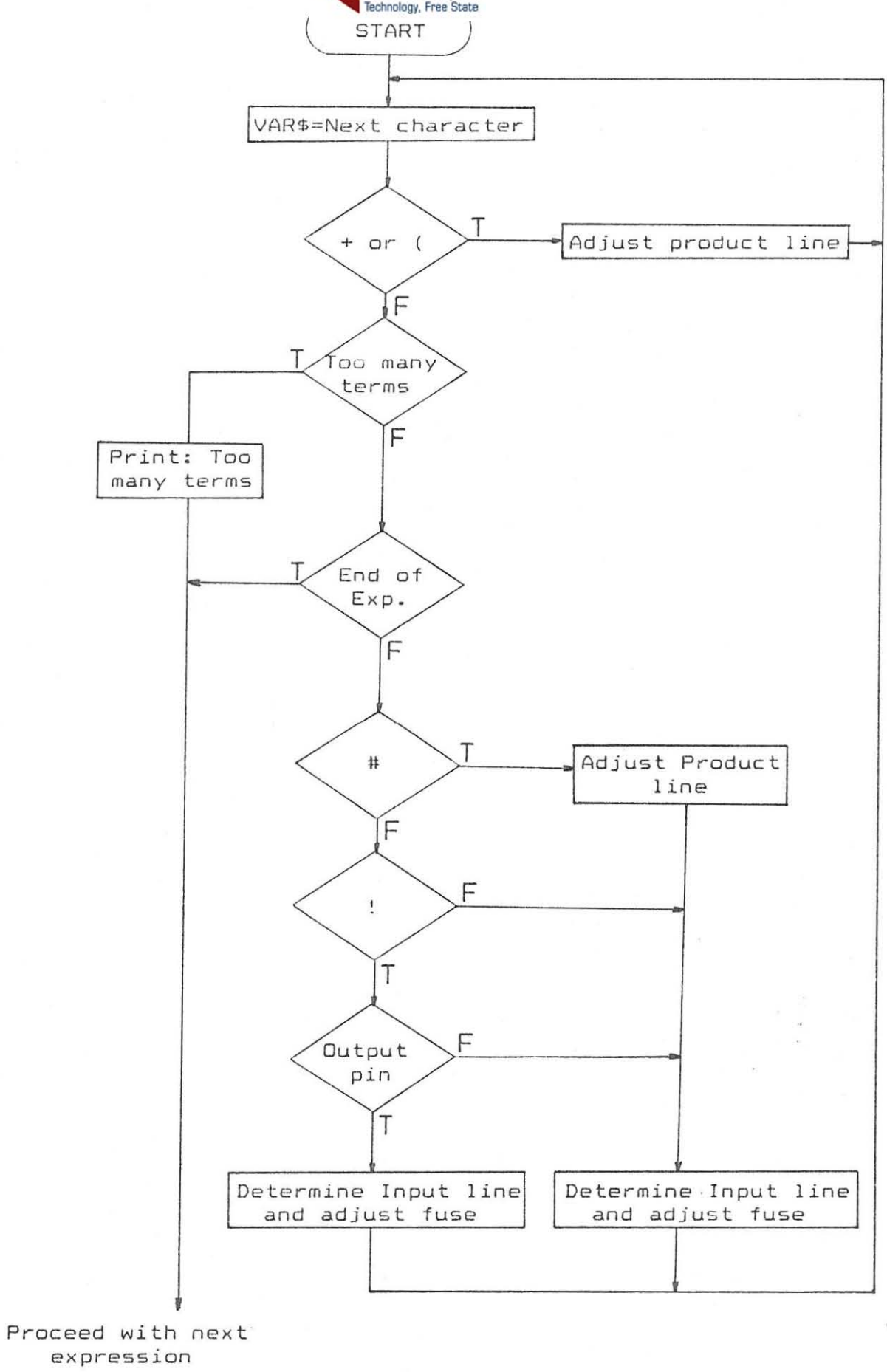


Fig. 4.4 - Flowchart of Input pin conversion.

blown". The following is a pseudo code listing of this routine:

```
1.  For T=0 to 63          -Check for each product line
    1.1  I=-1
    1.2  I=I+1
    1.3  If F%(I,T)=1 then   -Fuse not to be blown
         True:  1.3.1  Next T
    1.4  If I<31 then
         True:  1.4.1  Goto 1.2
    1.5  For I=0 to 31       -If all fuses to be blown
         1.5.1  F%(I,T)=2    -Change to not to be blown
         1.5.2  Next I
    1.6  Next T
```

4.3.4 Phantom Fuses

Phantom fuse locations are those locations where a fuse does not exist (Birkner and Coli, 1983: 4.3). These positions have to be identified during the fuse map generation phase since the programmer pre-verifies these fuses as blown. The next routine identifies these locations and modifies the fuse map so that the pre-verification software is able to identify these fuses as non-existent instead of assuming them to be blown.

Due to the relative complexity of this process, this routine will be examined by means of two flowcharts. The identification of phantom fuses entails two basic steps:

1) Many PALs are not supplied with the maximum possible number of input lines. The 10H8 PAL for example has only 20 of a possible maximum of 32 input lines. This obviously creates a number of phantom fuses. The first part of this routine deals only with these phantom fuses (as shown in figure 4.5).

2) Many PALs are supplied with less than the maximum possible number of product lines. The 10H8 PAL has for instance only 16 out of a possible maximum of 64 product lines. The second step in this fuse map modification process is therefore aimed at identifying all of these non-existent lines. Fig 4.6 is a flowchart of this second part of the routine.

4.3.5 Display of Fuse Map

Once the fuse map has been processed for don't cares and phantom fuses, the operator has the facility of displaying and examining the fuse map before saving it on floppy disk. This is a fairly straightforward routine and is presented in pseudo code.

```
1.  Input: "Fuse map display required?;FM$
2.  If "NO" then
       True:  Goto 4
3.  For T=0 to 63
       3.1  For I=0 to 31
```

```
3.1.1  If F%(I,T)=0 then
  True:  3.1.1.1  Print:  "-"
3.1.2  If F%(I,T)=1 then
  True:  3.1.2.1  Print:  "*"
3.1.3  If F%(I,T)=2 then
  True:  3.1.3.1  Print:  "+"
3.1.4  Next I
```

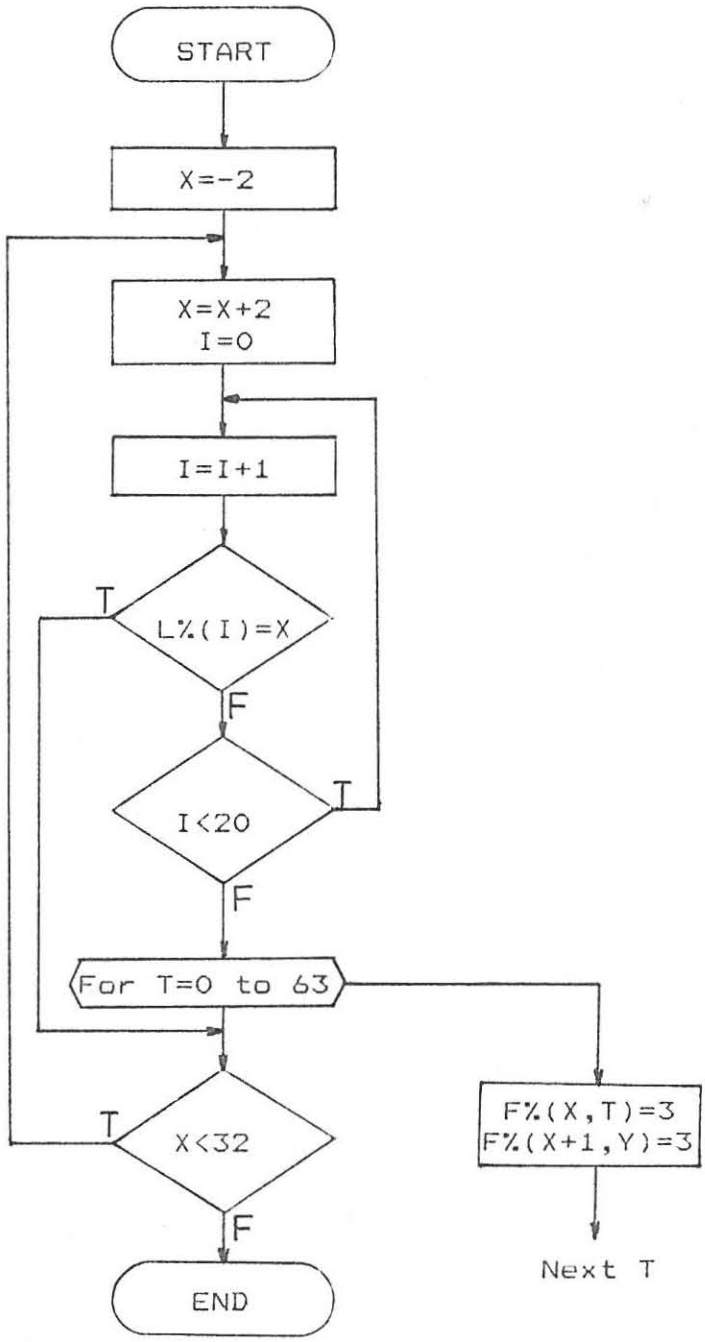


Fig. 4.5 - Flowchart of Input line modification.

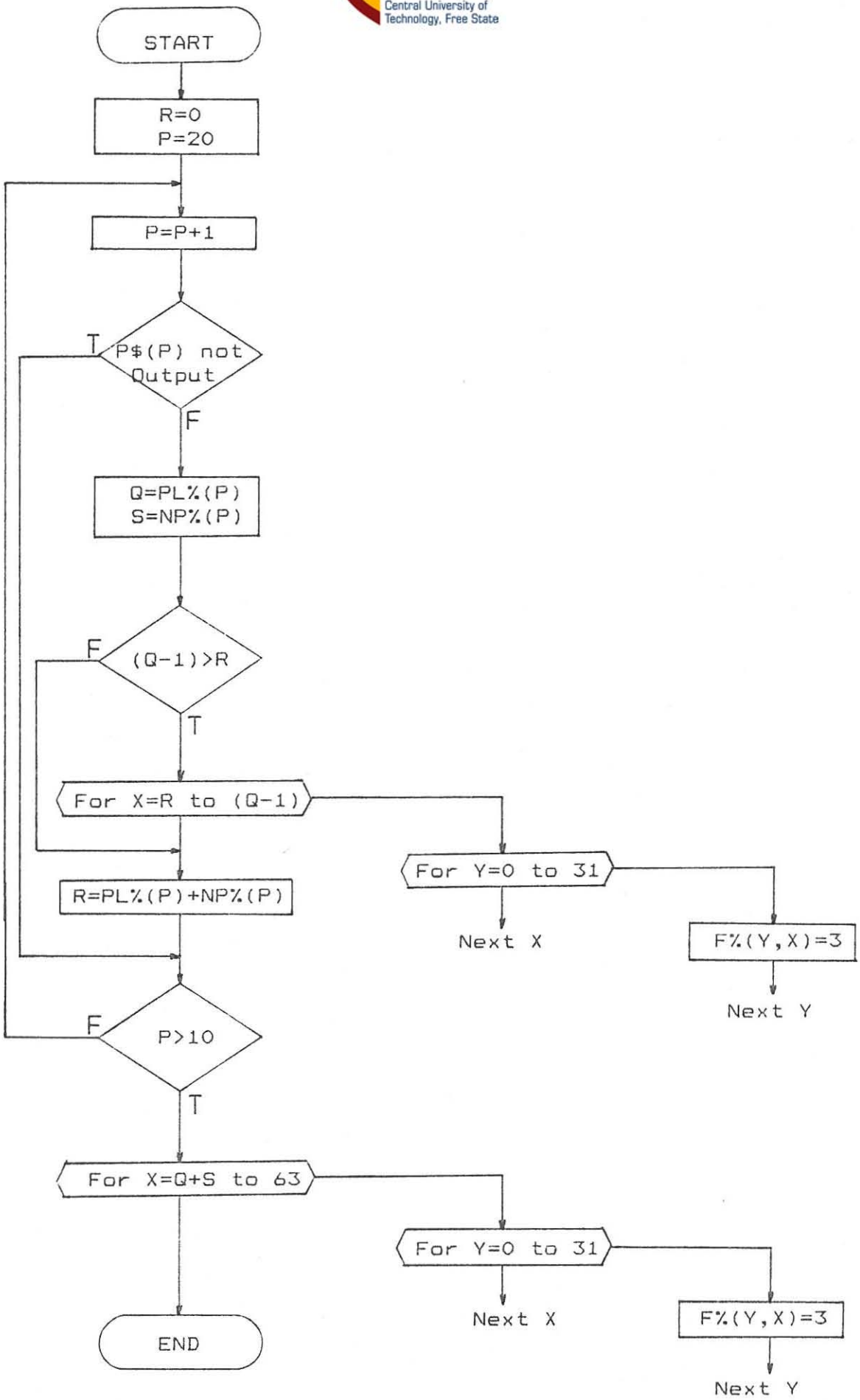


Fig. 4.6 - Flowchart of Product line modification.

3.2 Print: T

3.3 Next T

4. END of routine

The representation of fuse conditions in the fuse map, display mode and hard copy can be summarized as follows:

CONDITION	FUSE MAP	DISPLAY AND HARD COPY
Blown	0	-
Not blown	1	*
Don't care	2	+
Phantom fuse	3	(space)

4.3.6 Save Fuse Map

Once the operator has examined the fuse map, he can either reject the fuse map and repeat the whole procedure, or he can accept it and save it on floppy disk. The fuse map is saved in the form of a sequential file and will be accessed by the programming software at a later stage. The following is a pseudo code listing of this saving routine.

1. Input: "Fuse map to be saved";FM$
2. If "NO" then

   True: 2.1 Goto 6
3. Open fuse map file
4. For I=0 to 31

   4.1 For T=0 to 63

      4.1.1 Print: F%(I,T)

4.1.2 Next T

4.2 Next I

5. Close fuse map file

6. END of routine.

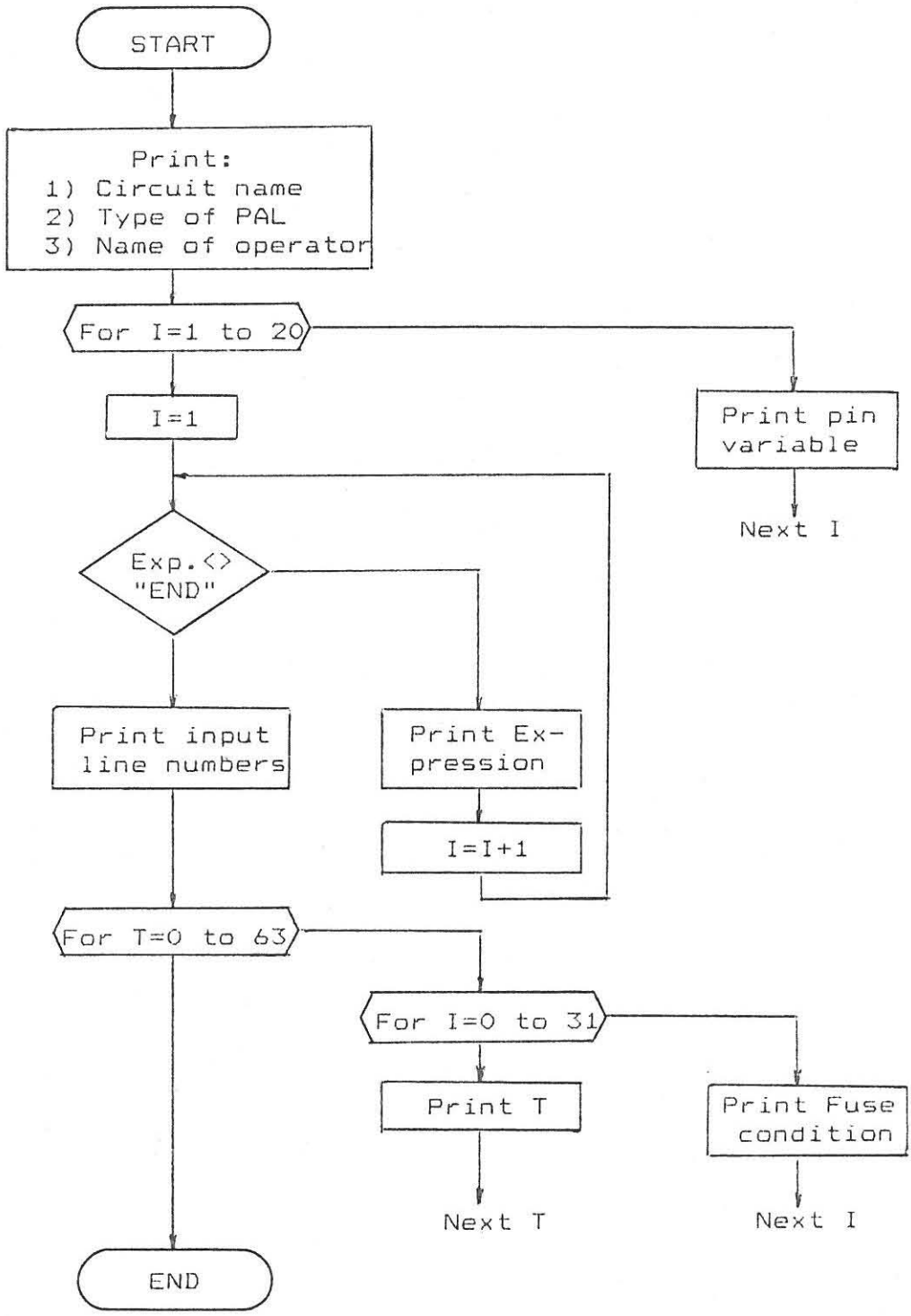


Fig. 4.7 - Flowchart of printing of fuse map.

4.3.7 Printing Fuse Map

At this stage the operator can decide whether he requires a hard copy of the fuse map as generated. If it is required, the routine as shown in figure 4.7 is executed. Otherwise the program terminates. This hard copy should be regarded as a record of any chip programmed with this fuse map. Without it, it is almost impossible to determine the nature of data programmed into the PAL.

4.4 Summary

The fuse map generating software consists of three programs.

1) "PAL File" is used to generate new PAL personality files whenever neccessary and saves it as a sequential file.

2) "Testing PAL File" is used to verify the accuracy of this personality file.

3) "Fuse Map 20" is used to generate a fuse map according to Boolean expressions supplied by the operator. During this process the personality file is accessed and variables are assigned to the pins of the PAL. This fuse map can be displayed on a monitor and is eventually saved on floppy disk. Hard copies of the fuse map can also be printed. This fuse map is accessed when the programming of a PAL takes place, as will be discussed in the next chapter.

CHAPTER 5

PROGRAMMING SOFTWARE

The programming software consists of a main program, Program PAL, written in BASIC, and a number of machine code routines. The same programming philosophy was followed in writing the main program as in the writing of the fuse map generating software.

During programming and verification of the PAL, certain stringent signal parameters are to be met. In order to meet these requirements, some of the signals have to be generated by means of machine code routines due to the possible higher execution speed. This led to the program design as shown in figure 5.1 (Programming - Subroutines overlay) with the machine code routines accessed from within the main program.

The following three points require special attention with regard to the subroutines:

1) Port extension.
2) Memory management.
3)Addressing modes.

5.1 Port Extension

The manner in which the port facilities of the Commodore 64 were extended is discussed in detail in chapter 6. With respect to the discussion of the software it should however

be mentioned that six 8 bit ports are used in the programmer. These are identified and addressed as follows:

PORT	Address in Decimal	Address in Hexadecimal
0A	56840	DE08H
0B	56842	DE0AH
1A	56848	DE10H
1B	56850	DE12H
2A	57088	DF00H
2B	57090	DF02H

Addressing of these ports is done by means of the I/O1 and I/O2 lines used in conjunction with address lines A3 and A4 (Commodore 64 Programmer's Reference Manual, 1983: 336). Memory-mapped input and output is used and therefore the in-

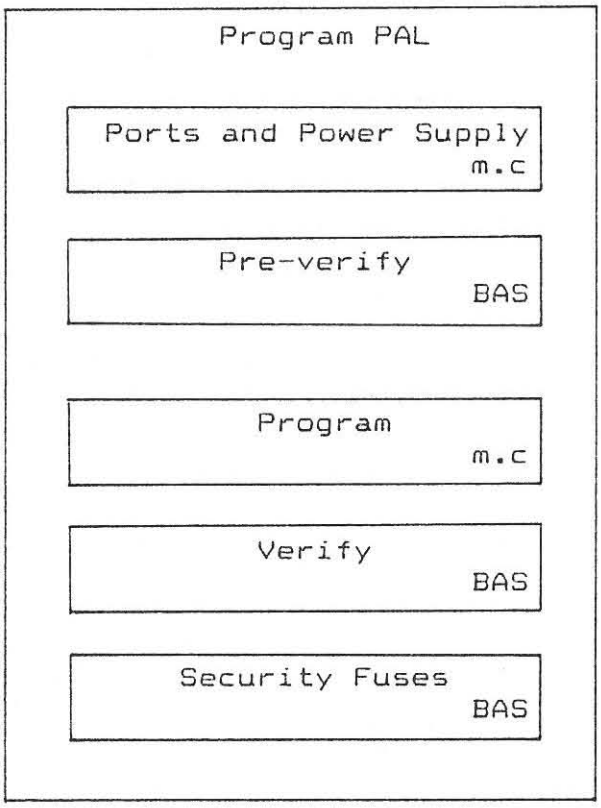


Fig. 5.1 - Program PAL - Subroutines overlay.

put and output of data in the machine code routines are realized using LDA (load accumulator) and STA (store accumulator) instructions respectively (Hall, 1983: 237). PEEK and POKE instructions were used in BASIC for this purpose.

5.2 Memory Management

The main program is loaded at the default memory locations. The fuse map file, as saved during execution of Fuse Map 20, is read by Program PAL and saved at locations 28672 to 30719 (7000H to 77FFH). The machine code routines are saved from memory location 32768 to 33188 (8000H to 81A4H). Locations 30720 to 30915 (7800H to 78C3H) are used as data tables and as a scratch pad by the subroutines.

5.3 Addressing Modes

The use of zero-page addressing was avoided in the machine code routines because of the switching between BASIC and machine code. Data required by the main program, in which zero-page addressing may well be used by the interpreter, will thus not be destroyed during execution of the machine code routines, or vice versa.

Throughout, a fairly linear program flow with uncomplicated modes of addressing (and a minimum of logical instructions) were adhered to. In this manner readability of the programs was improved even if this produced a less than optimum use of memory capacity - but then the Commodore 64 has more than sufficient capacity for the purpose of the programmer.

5.4 Program PAL - Main Program

The program flow of this program is as shown in figure 5.2. Top-down program design has the desirable effect of creating a program with the appearance of a level-one flow diagram, thus improving its readability (Koffman and Friedman, 1979: 147). As described in section 3.3 the top-down design principles were adhered to during the development of this program. The main program can be considered to consist of the following modules:

MODULE	LINE NUMBER
1) Program identification	10
2) Loading of programming routines	20-80
3) Loading of data table	90-140
4) Initialization of ports and power supply	150-160
5) Loading of fuse map	165-375
6) Pre-verification of PAL	380-445
	1000-1530
7) Programming of PAL	450-500
8) Verification of PAL	510-576
	1600-3560
9) Programming of security fuses	580-647

A fair amount of operator intervention is required during execution of this program. Self-explanatory prompts have been provided to enable the operator to select the desired operation, i.e. pre-verify, program, verify or program security fuses.

A listing of the main program, with its BASIC subroutines, is supplied in Appendix D.

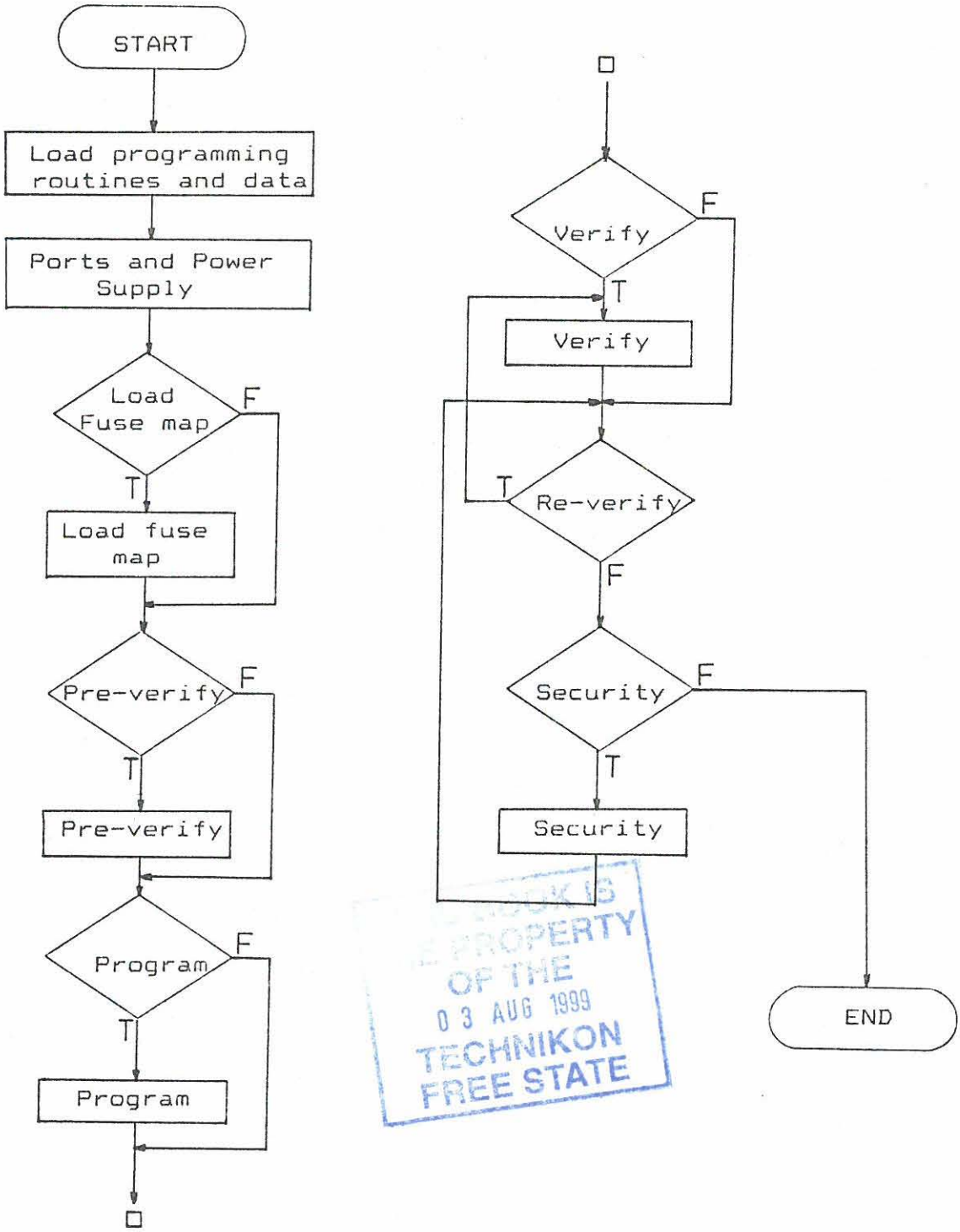


Fig. 5.2 - Flowchart - Programming software.

5.5 Ports and Power Supply

Immediately after power-up the three peripheral interface adapters (PIAs), used as port extention, have not yet been initialized. The programmer design is such that its power supply is deactivated until it is supplied with a suitable control signal by peripheral interface adapter PIA2. The PIAs and power supply must therefore be activated before the programmer can operate. This is then the purpose of this subroutine which is run automatically by the main program after loading the machine code routines.

The following is a short summary of the operation of this routine:

1) Set ports 0A to 2B to control mode.
2) Program ports 0A to 2B as output ports.
3) Set ports 0A to 2B to data mode.
4) Reset 0A to 1B and 2B to 00H.
5) Output 11H at port 2A to activate the power supply to 5 and 11,75 volt.

A listing of the machine language routines is supplied in Appendix E.

5.6 Pre-verification of PAL

Before programming a PAL, the operator has to verify that it is indeed an unprogrammed PAL which he is about to program.

This process is referred to as "pre-verification" and entails the testing of each fuse in the PAL to ensure that it has not been damaged before programming. If any fuses in a PAL have been blown beforehand, it may be impossible to realize the required Boolean expression with the chip and it should be rejected.

There may however be exceptions to this rule in that the utilization of the chip may take place in more than one step. Suppose for instance that not all the outputs of a PAL have been used initially. The user of the chip may now decide, at some later stage, to expand the capabilities of the chip by blowing some additional fuses that were regarded as don't cares in the initial application.

It was therefore decided to pre-verify only those fuses which should not be blown in order to realize the required fuse map, irrespective of any previous programming. This greatly reduced the execution time of the pre-verify program as well as increasing its versatility.

The operation of this program can be summarized as follows:

1) Set VCC at 5 volt.
2) Raise the Output Disable (OD) pin to 11,75 volt.
3) Address each fuse in turn by applying suitable signals to the Input and Product Lines as well as the LR pin.
4) Pulse the Clock pin and verify that the addressed fuse is indeed in the unblown condition. These levels are logic Ø for active high outputs and logic 1 for active

low outputs (Programmable Logic Databook by National Semiconductor, 1983: 24.24). If these levels are not correct, the PAL should be rejected.

Figure 5.3 is a first-level flowchart of this program.

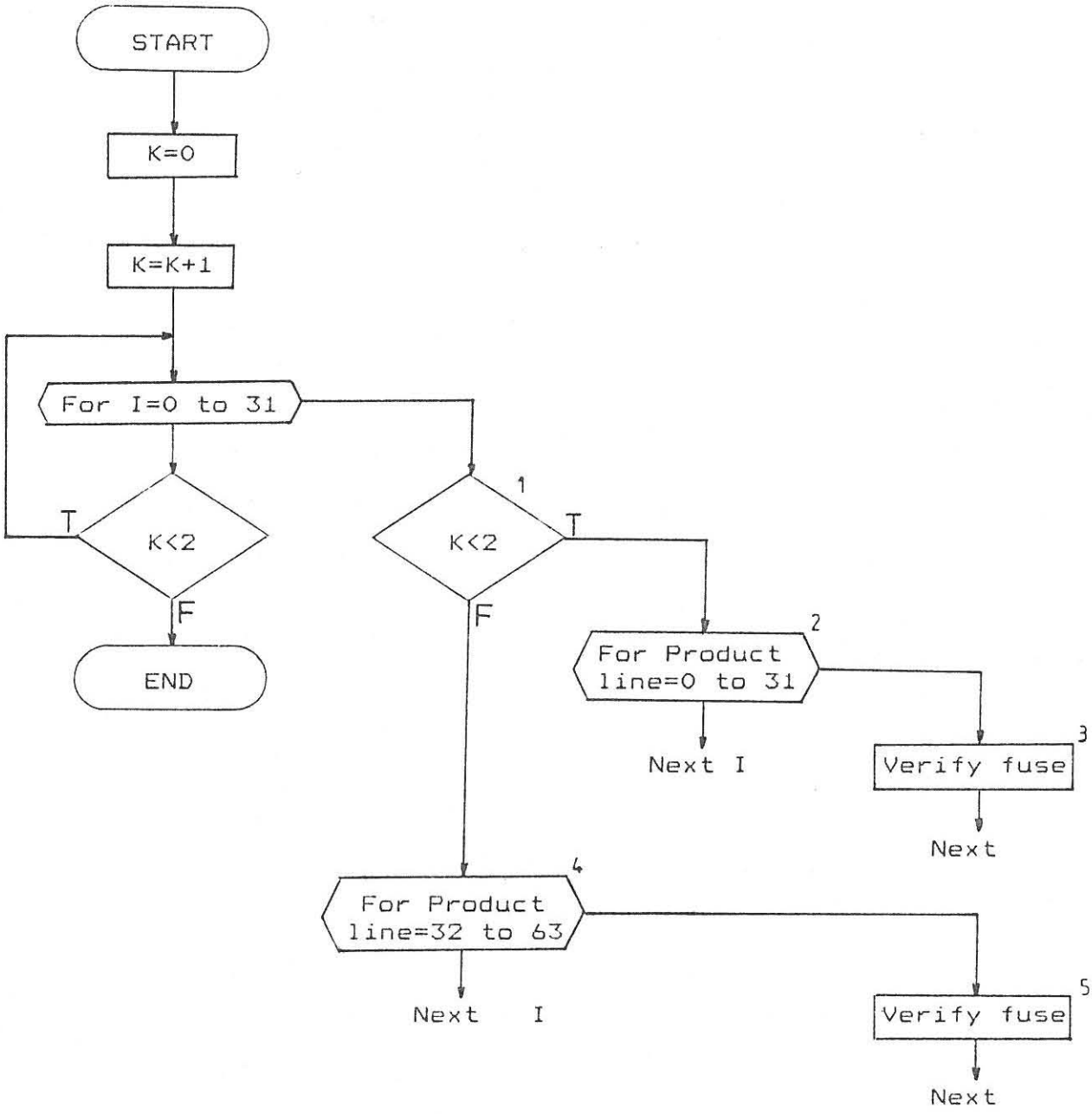


Fig 5.3 - Flowchart of pre-verify.

5.6.1 Addressing of Input Lines

The PAL input lines are addressed as follows:

1) A combination of VHH (11,75 volt) signals are required. Eight different combinations are used with any combination serving 4 input lines (see below).

2) Those input lines not at VHH are switched to either VH (5 volt) or high-impedance.

3) The LR pin is switched to 5 volt or to high-impedance.

The circuit has been designed in such a manner that port 0A switches VHH, port 0B switches VH and one bit of port 1A the LR pin. The addressing of input lines 0 to 7 would therefore require the following outputs at the indicated ports:

Input Line	Port 0A	Port 0B	LR Pin
0	FEH	00H	0
1	FEH	FFH	0
2	FEH	00H	1
3	FEH	FFH	1
4	FDH	00H	0
5	FDH	FFH	0
6	FDH	00H	1
7	FDH	FFH	1

From this it is obvious that:

1) Port OA remains unchanged in cycles of four input lines.

2) Port OB switches to OOH for the even numbered input lines and to FFH for the uneven numbered input lines.

3) LR is O for two input lines, 1 for the next two, and so on.

A refinement of step 1 in figure 5.3, regarding the addressing of the input lines, is as shown in figure 5.4.

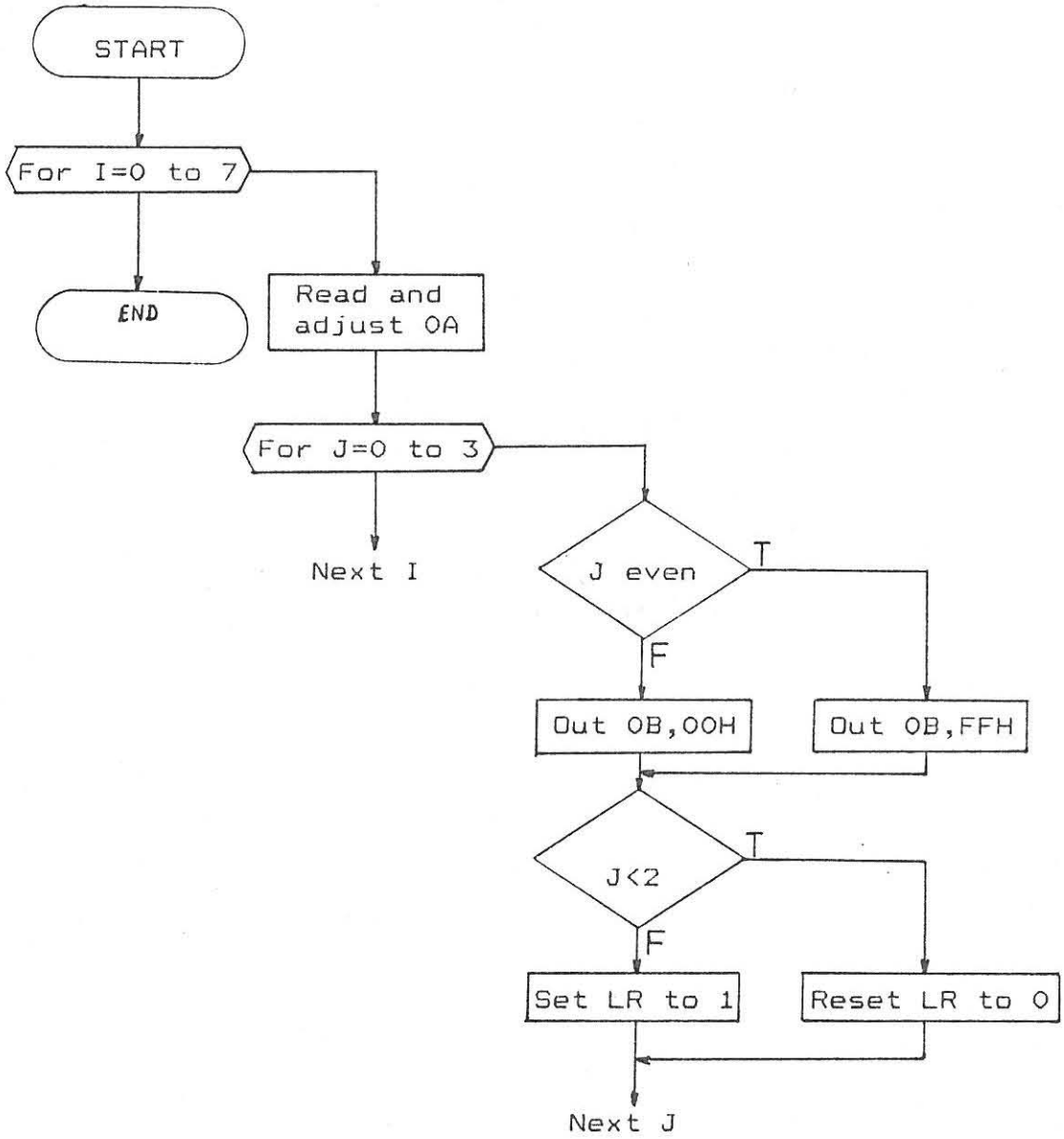


Fig. 5.4 - Flowchart - Addressing of input lines.

5.6.2 Addressing of Product Lines

The product lines are addressed by port 1A. At any moment in time 3 bits of this port are used to address a product line and a fourth is used to select the LR pin as explained in section 5.6.1. The other four bits of port 1A are used to read the condition of the addressed fuse. The LR pin has been discussed in section 5.6.1 and hence it will not be considered again.

Since the pre-verification routine involves both the reading and writing of data through port 1A, the PIA should be initialized to facilitate this. The multiplexers (see section 6.5.1) are also required to function according to the operation of the port - MUX0 as input and MUX1 as output and vice versa. The switching of the MUXs are however kept to a minimum by pre-verifying the fuses in two blocks, viz. product lines 0 to 31 for all input lines, followed by product lines 32 to 63 for all input lines.

Figure 5.5 is a refinement of blocks 2 and 3 of figure 5.3 with LR in the condition as described in section 5.6.1.

Addressing of product lines 32 to 63 is done in exactly the same manner except that A, in figure 5.5, loops from 0 to 14 in steps of 2. Because of the similarity with product lines 0 to 31, these will not be discussed in detail.

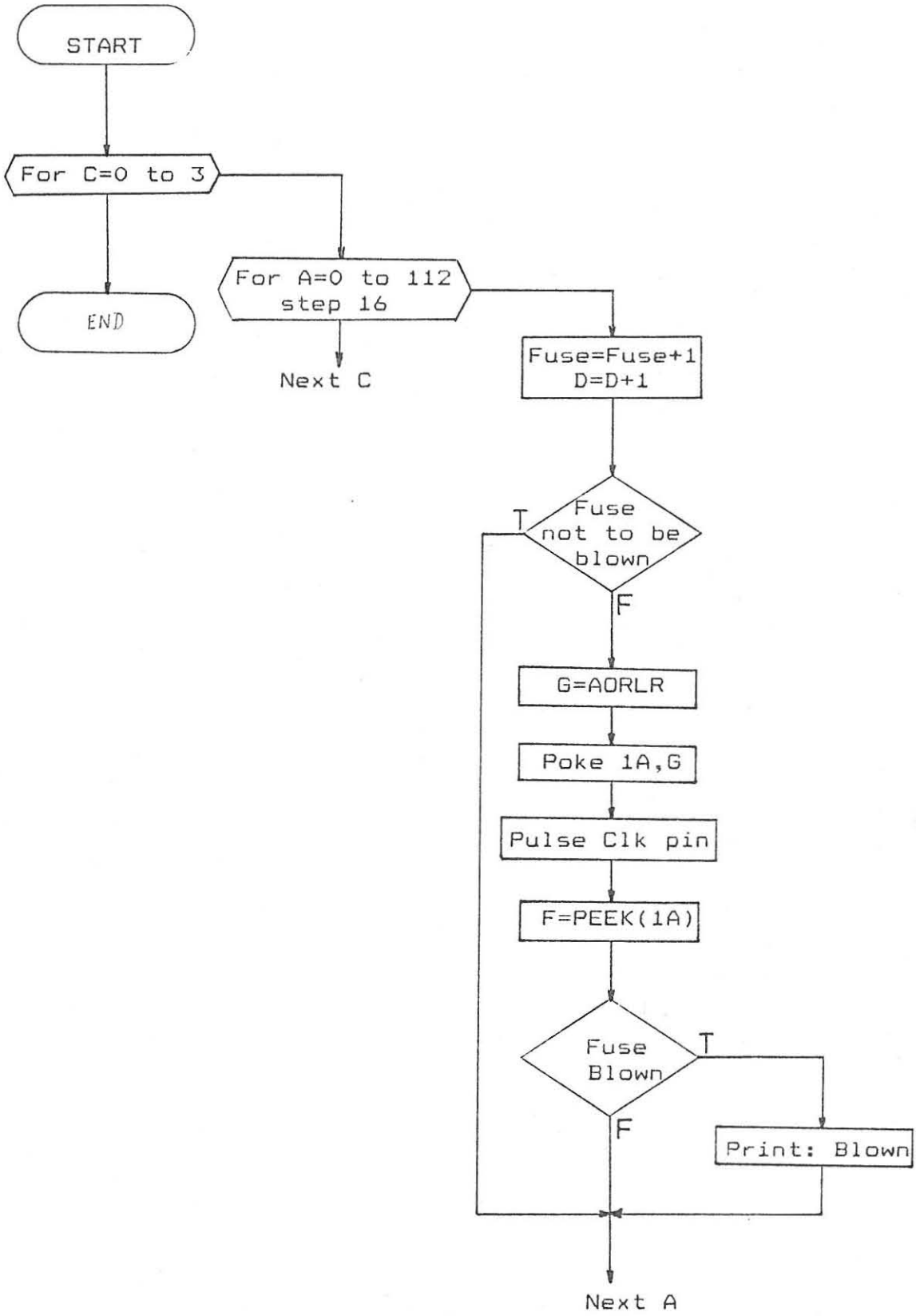


Fig 5.5 - Addressing of Product lines 0 to 31.

5.6.3 Verification of Fuse Condition

The logic level of an addressed fuse depends on its condition and the active logic level of the PAL. This can be summarized as follows:

Fuse Condition	Active High	Active Low
Unblown	Logic 0	Logic 1
Blown	logic 1	Logic 0

Considering this, a logical refinement of the "fuse blown?" decision in figure 5.5 is given in figure 5.6. Variable F is assumed to be the fuse condition as read through port 1A and R$ is either an H,L or R. An H

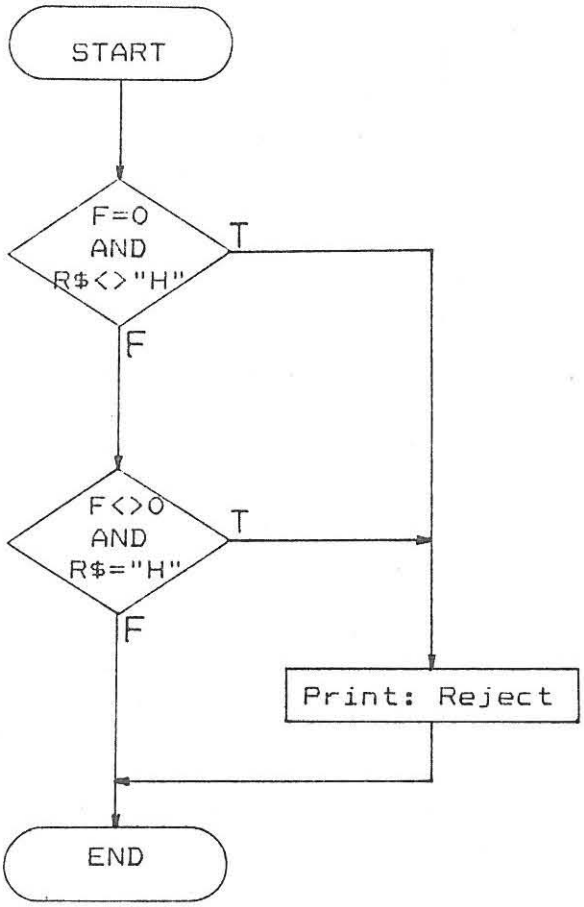


Fig 5.6 - Verify fuse as unblown.

indicates a PAL with an active high output, while an L or R indicates a device with an active low output.

5.7 Programming of PAL

Since the following two timing parameters could not be met in BASIC, the program routine is in machine code.

TP MAX ---- 50 microsecond

TCCP MAX -- 60 microsecond

The "Supermon64.V1" assembler program was used to develop the programming software. Figure 5.7 is a first-level flow-chart of this routine.

A data table is created at memory locations 7800H to 78BFH to be used by the programming subroutine. This data table consists of the following:

MEMORY LOCATION	PURPOSE
7800H-781FH	Select VHH on Input lines.
7820H-783FH	Select VH on Input lines.
7840H-785FH	Select A0-A2 on Product lines 0 to 31.
7860H-787FH	Select A0-A2 on Product lines 32 to 63.
7880H-789FH	Select program pulse (Product line < 32).
78A0H-78BFH	Select program pulse (Product line > 31).

5.7.1 Addressing of Input Lines

The input line loop in figure 5.7 can be refined to

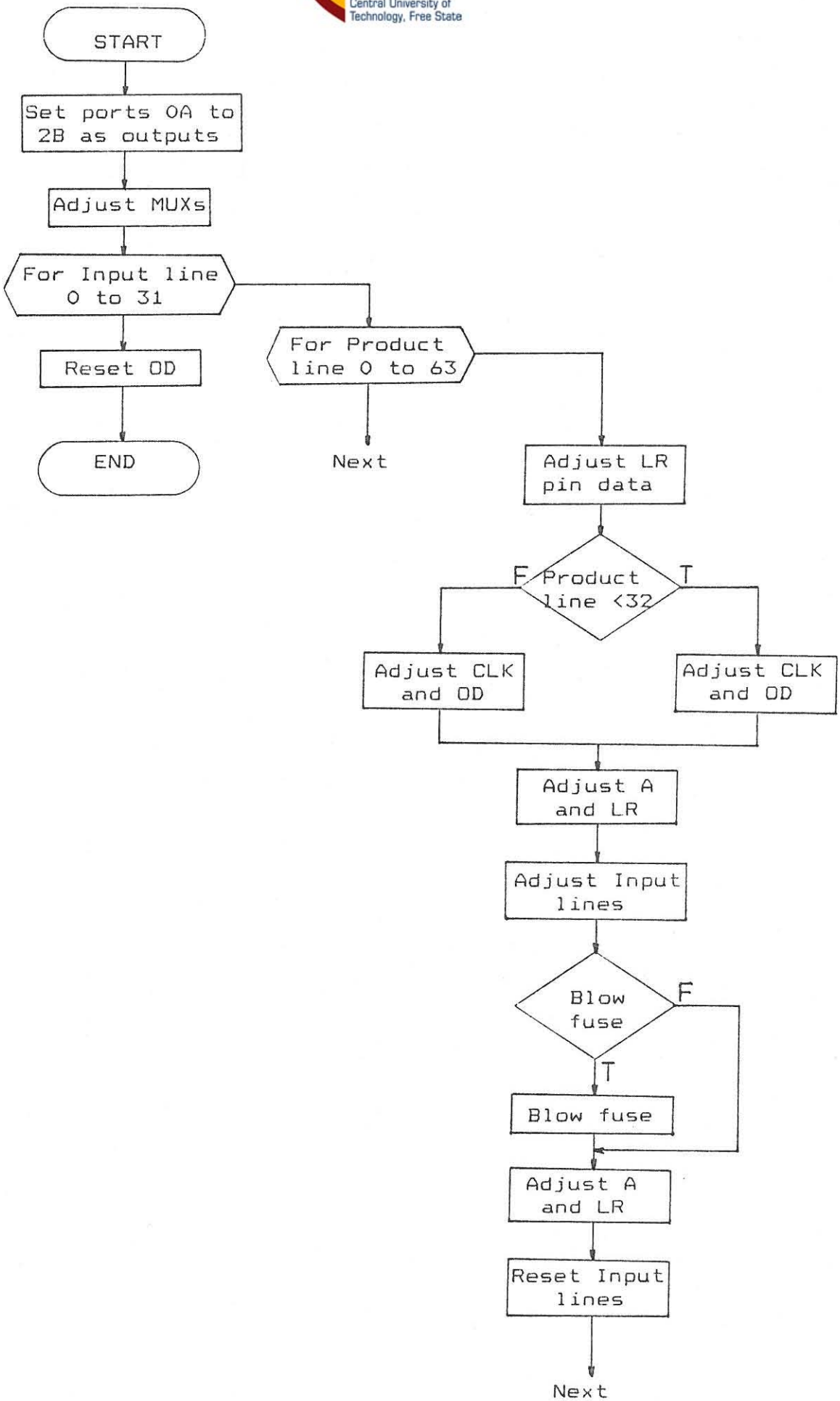


Fig 5.7 - Flowchart of program routine

figure 5.8. Index register X is used as pointer to identify the input line being addressed. In this manner indexed addressing could be used (Bayley, 1984: 107). Location 78C1H is initially loaded with 1FH, and is decreased as each input line is completed. The control of the LR pin will be discussed in section 5.7.2.

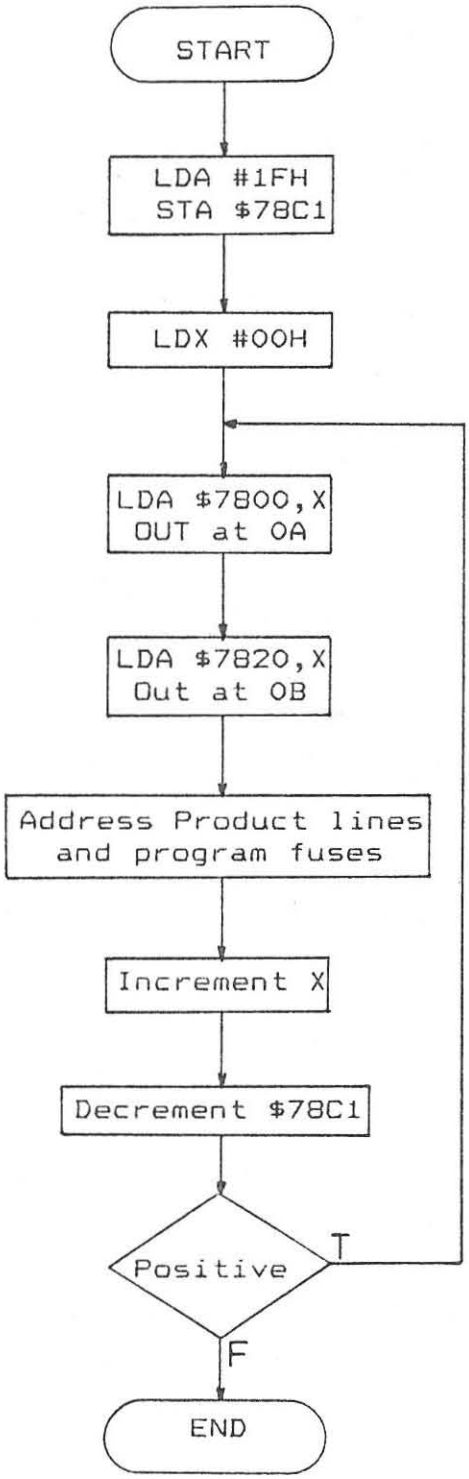


Fig. 5.8 - Flowchart of addressing of input lines.

5.7.2 Addressing of Product Lines and Adjusting LR

A similar approach has been used in developing the product line addressing loop to that followed with the input lines. In this case memory location 78C0H is used in conjunction with the Y index register as pointer.

Due to the multipurpose operation of some PAL pins - especially the CLK and OD pins - special care was taken in designing this program segment. The function of these two pins change around after product line 31. At the same time the LR pin switches from pin 12 to 19 - requiring 01H instead of 80H on port 1A whenever the pin is to be at VHH. Figure 5.9 shows a refinement of the product line addressing.

5.7.3 Control of Output Disable and Clock Pins

The manner in which the program controls the OD and CLK pins is closely related to the addressing of the product lines (as discussed in section 5.7.2). The Y index register is increased with the addressing of each successive product line, whilst the contents of memory location 78C0H is decreased from its initial setting of 3FH.

During each product line loop the contents of location 78C0H is compared with that of the Y index register. The contents of 78C0H is larger than that of the Y index register for product lines 0 to 31, but for product lines

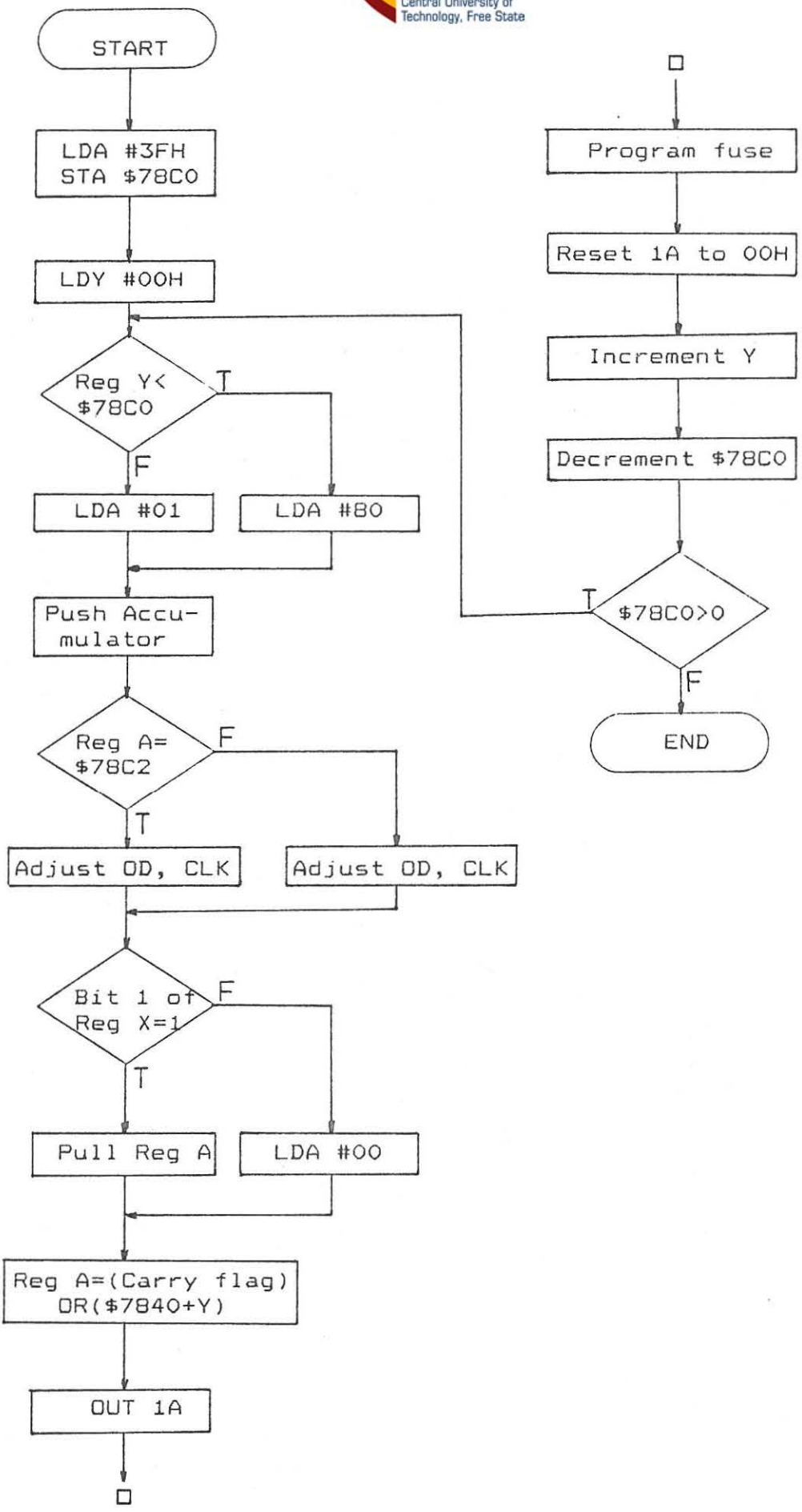


Fig. 5.9 - Adressing product lines and adjusting LR.

32 to 63 the contents of location 78C0 is smaller than that of the Y register. This condition is tested for, and is used to control the LR bit of port 1A - this bit also changes after product line 31 - as well as to control the OD and CLK pins of the PAL.

For product lines 0 to 31 pin 1 is at VHH and pin 11 at 0 volt. For the other product lines these conditions reverse and pin 11 becomes VHH and pin 1 ground potential. Figure 5.10 illustrates the control of the OD and CLK pins.

5.7.4 Programming of Fuses

Sections 5.7.1 to 5.7.3 dealt with the addressing of a fuse. It should however be ascertained from the fuse map whether the fuse has to be blown or not. The actual blowing of the fuse entails its addressing, switching of

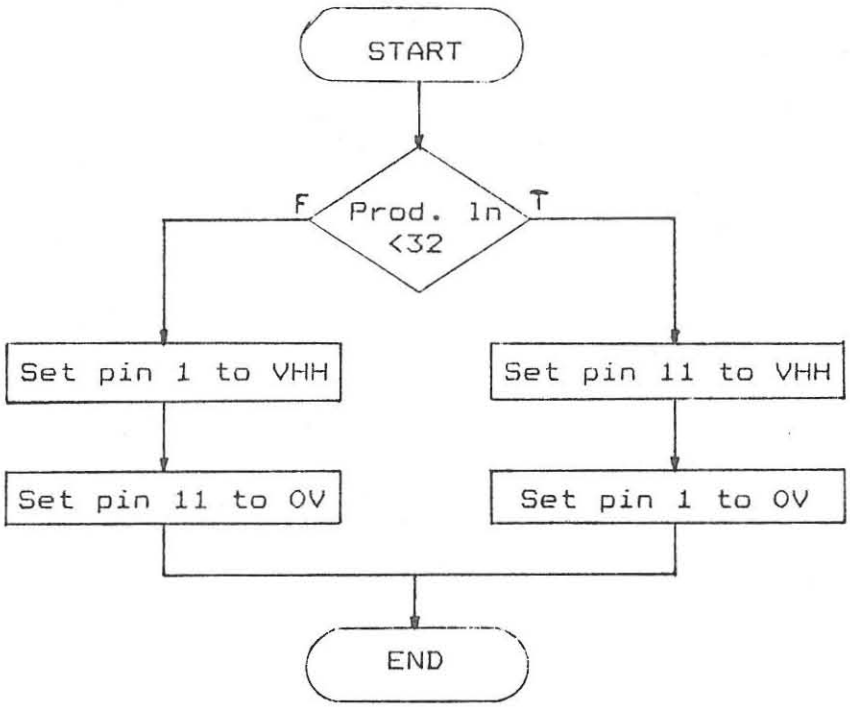


Fig 5.10 - Control of OD and Clk pins.

the PAL supply voltage, VCC, to 11,75 volt and the pulsing of an output pin - controlled by port 1A - to 11,75 volt.

The fuse map is compiled in such a way that those fuses which are to be blown are identified by a 0 in the appropriate memory locations. This feature is used to determine whether an addressed fuse should be blown or not. Figure 5.11 illustrates this process and the generation of the programming pulse.

The programming pulse is read from the data table using indexed addressing. The data is then ORed with the product line address as pushed onto the stack at an earlier stage.

The above routine resulted in a programming waveform as in figure 5.12. The timing notes are given in microseconds relative to the leading edge of the program pulse. The resultant waveform is well within the specified programming parameters for PALs manufactured by Monolithic Memories (Birkner and Coli, 1983: 3.20). PALs manufactured by National Semiconductor have similar programming characteristics (Programmable Logic Databook by National Semiconductor, 1983: 24.24) and can also be programmed with these signals.

5.8 Verification of Programmed PAL

Once the PAL has been programmed, it should be verified that

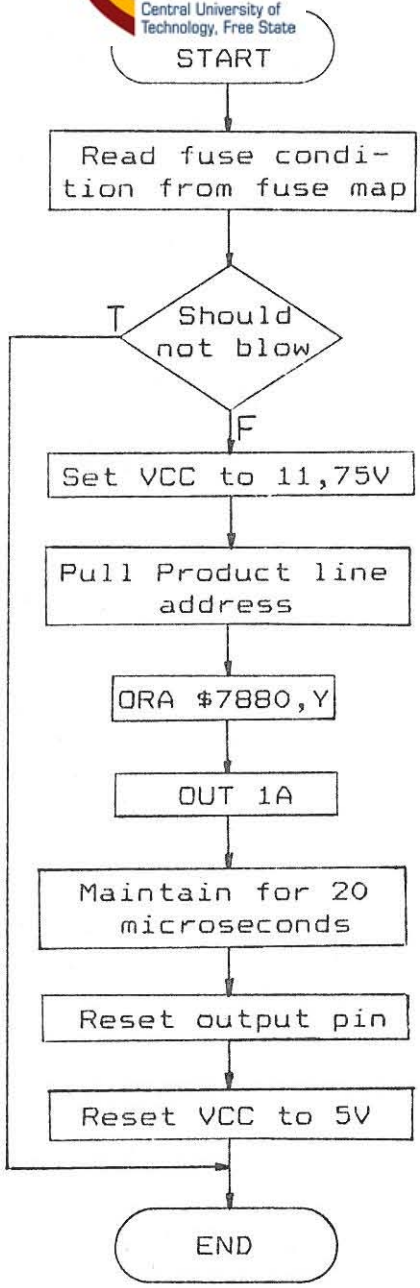


Fig. 5.11 - Program fuse.

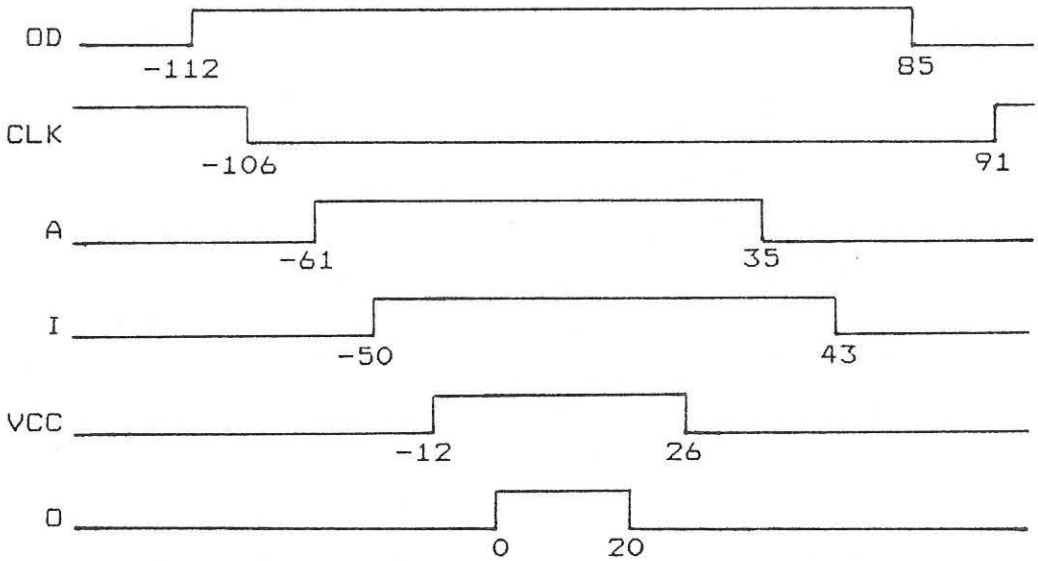


Fig. 5.12 - Programming waveforms.

all the required fuses have in fact been blown successfully. The routine for this is in BASIC, since no critical timing parameters need to be met.

The verification routine is virtually identical to that of pre-verification, with only minor differences. Verification can take place at different values of supply voltage. For this reason the operator is prompted to specify the preferred verification voltage. The voltage specified is referred to a variable, PV$, according to which the power supply is adjusted by port 2A. The following pseudo code listing illustrates this concept:

```
1. Print: "Select verification voltage"
2. Print: "First verification pass...5V...1"
3. Print: "High voltage verify.....5,5V...2"
4. Print: "Low voltage verify......4.5V...3"
5. Print: "Select 1, 2 or 3"
6. Get: VV
7. If VV=1 then
   True: 7.1 PV$="FV"
8. If VV=2 then
   True: 8.1 PV$="HV"
9. If VV=3 then
   True: 9.1 PV$="LV"
10. Gosub Verify
11. Input : "Further verification to take place?";P$
12. If YES then
   True: 12.1 Goto 1
13. END of verification.
```

During verification fuses are verified to be blown whereas in pre-verification they are verified as not blown. This has the effect of inverting the logic level of the verified fuses as compared to pre-verification (see section 5.6.3 for the relevant logic levels) and therefore the Pass/Fail logic of this routine is different from that of pre-verification.

One aspect in which verification differs considerably from pre-verification, is in what happens when a fuse fails to verify correctly. Should this occur during the first verification pass at 5 volt, another attempt is made to blow the fuse. In this manner two more attempts can be made at blowing the fuse before it is rejected as a faulty component. If however, a fuse verifies at 5 volt but not at 4,5 or 5,5 volt, it should also be rejected (Birkner and Coli, 1983: 3.19 and 3.20).

Figure 5.13 shows the program flow in these cases. In fact two such routines were developed, one for product lines 0 to 31 and another for product lines 32 to 63. This duplication was decided upon due to the changes in pin functions as described in section 5.7.3.

The reprogram cycle, given in figure 5.13, is executed in BASIC with the exception of the generation of the actual programming pulse. Due to the short duration of this pulse it has to be generated with machine code. Figure 5.14 illustrates the program flow during this process. Variable G in figure 5.14 is the data required on port 1A by the LR and A0 to A2 pins.

During verification fuses are verified to be blown whereas in pre-verification they are verified as not blown. This has the effect of inverting the logic level of the verified fuses as compared to pre-verification (see section 5.6.3 for the relevant logic levels) and therefore the Pass/Fail logic of this routine is different from that of pre-verification.

One aspect in which verification differs considerably from pre-verification, is in what happens when a fuse fails to verify correctly. Should this occur during the first verification pass at 5 volt, another attempt is made to blow the fuse. In this manner two more attempts can be made at blowing the fuse before it is rejected as a faulty component. If however, a fuse verifies at 5 volt but not at 4,5 or 5,5 volt, it should also be rejected (Birkner and Coli, 1983: 3.19 and 3.20)..

Figure 5.13 shows the program flow in these cases. In fact two such routines were developed, one for product lines 0 to 31 and another for product lines 32 to 63. This duplication was decided upon due to the changes in pin functions as described in section 5.7.3.

The reprogram cycle, given in figure 5.13, is executed in BASIC with the exception of the generation of the actual programming pulse. Due to the short duration of this pulse it has to be generated with machine code. Figure 5.14 illustrates the program flow during this process. Variable G in figure 5.14 is the data required on port 1A by the LR and A0 to A2 pins.

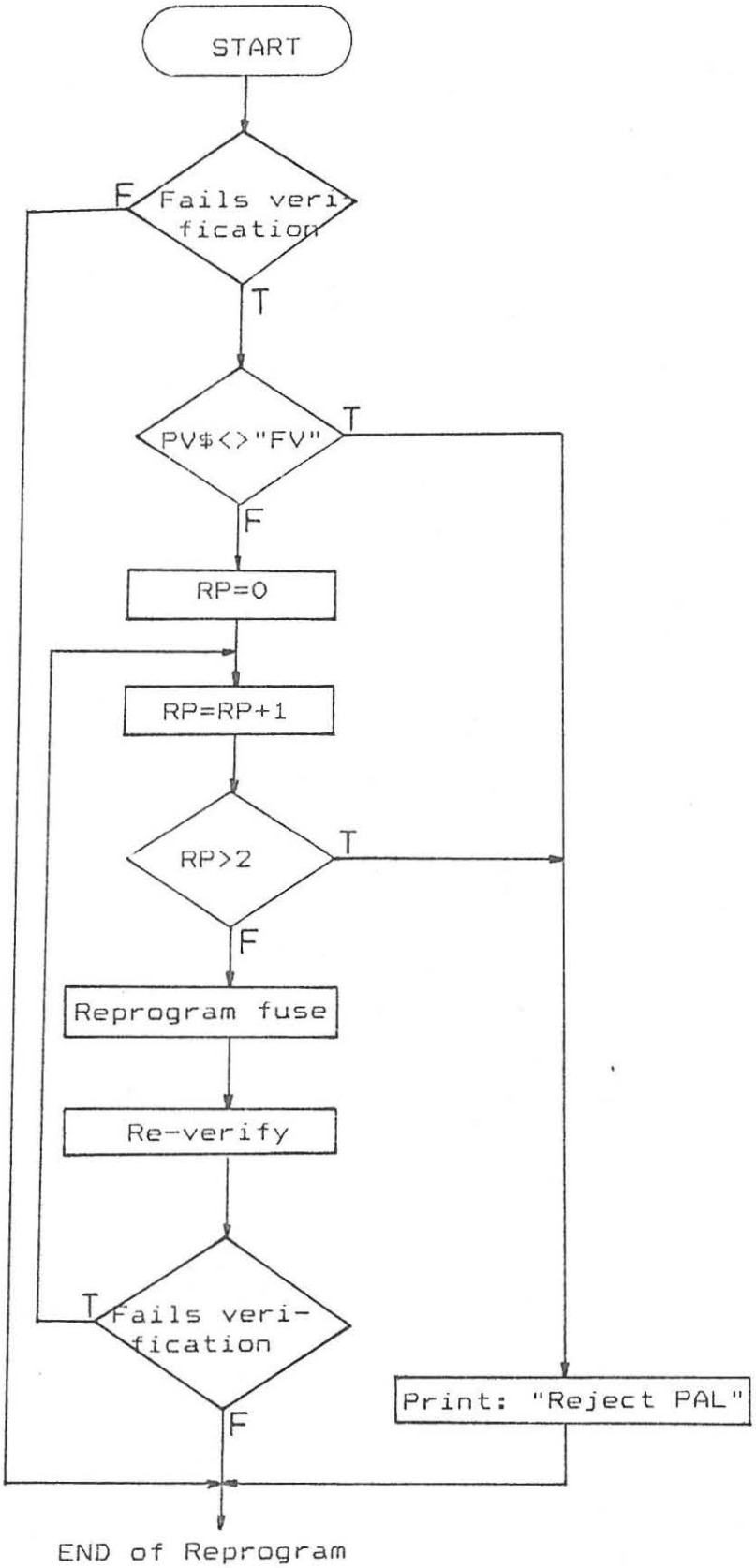


Fig. 5.13 - Address fuse for reprogramming during verification.

5.9 Security Fuses

Once the operator is certain that the PAL verifies correctly according to the fuse map, the security fuses can be programmed. Programming these would disable the verification logic of the chip.

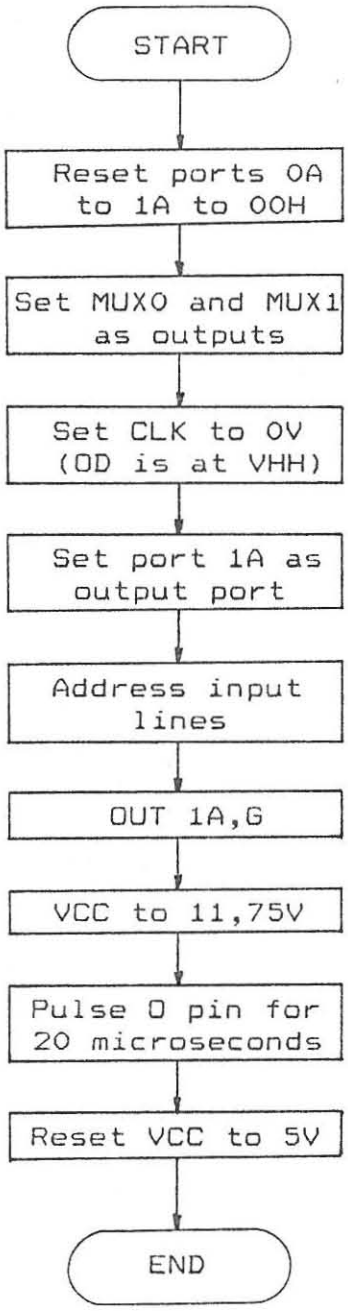


Fig. 5.14 - Reprogram fuse during verification.

Due to the signal timing parameters, which could not be met with BASIC, this is also a machine code routine saved at memory location 8161H to 8171H. This routine involves switching pins 1 and 11 to 18,5 volt for a duration of 40 microseconds with the PAL supply voltage at 6 volt. This process is repeated four times after which the program terminates.

5.10 Summary

The Program PAL software reads the fuse map generated by Fuse Map 20 and the machine code routines which form part of the programming software. During execution thereof the operator is guided through the program with suitable prompts. In this manner he can use the programmer to pre-verify, program, verify or to blow the security fuses.

The program execution speed is rather slow because the software is primarily in BASIC - on a machine with a clock frequency of only 1 megahertz. By contrast, the programming phase, which is in machine code, is very fast.

Appendix D is a listing of the main program with its subroutines, while appendix E is an assembly language listing of the machine code routines. No comments or labels are supplied with this listing since the assembler program used makes no provision for these.

CHAPTER 6

PAL PROGRAMMER HARDWARE

The programming system consists of a Commodore 64 computer and a programmer. The block diagram of the programmer is given in figure 6.1. Appendix F gives the circuit diagram of the programmer.

6.1 Block Diagram

The programmer is connected to the computer via the expansion connector on the back of the Commodore 64. This being a 44 pin female connector, a small printed circuit board adaptor and a flat ribbon cable connects the computer with the actual programmer.

6.1.1 Peripheral Interface Adapters

Three 6821 PIAs are used as interface between computer and programmer. The functions of these PIAs are summarized in table 6.1 (PAL pins and port connectors) and table 6.2 (Control of power supply).

The PIAs are identified as PIA0, PIA1 and PIA2 and addressed as described in section 4.1. Although ten port bits are not connected, it was impossible to decrease the number of PIAs to two.

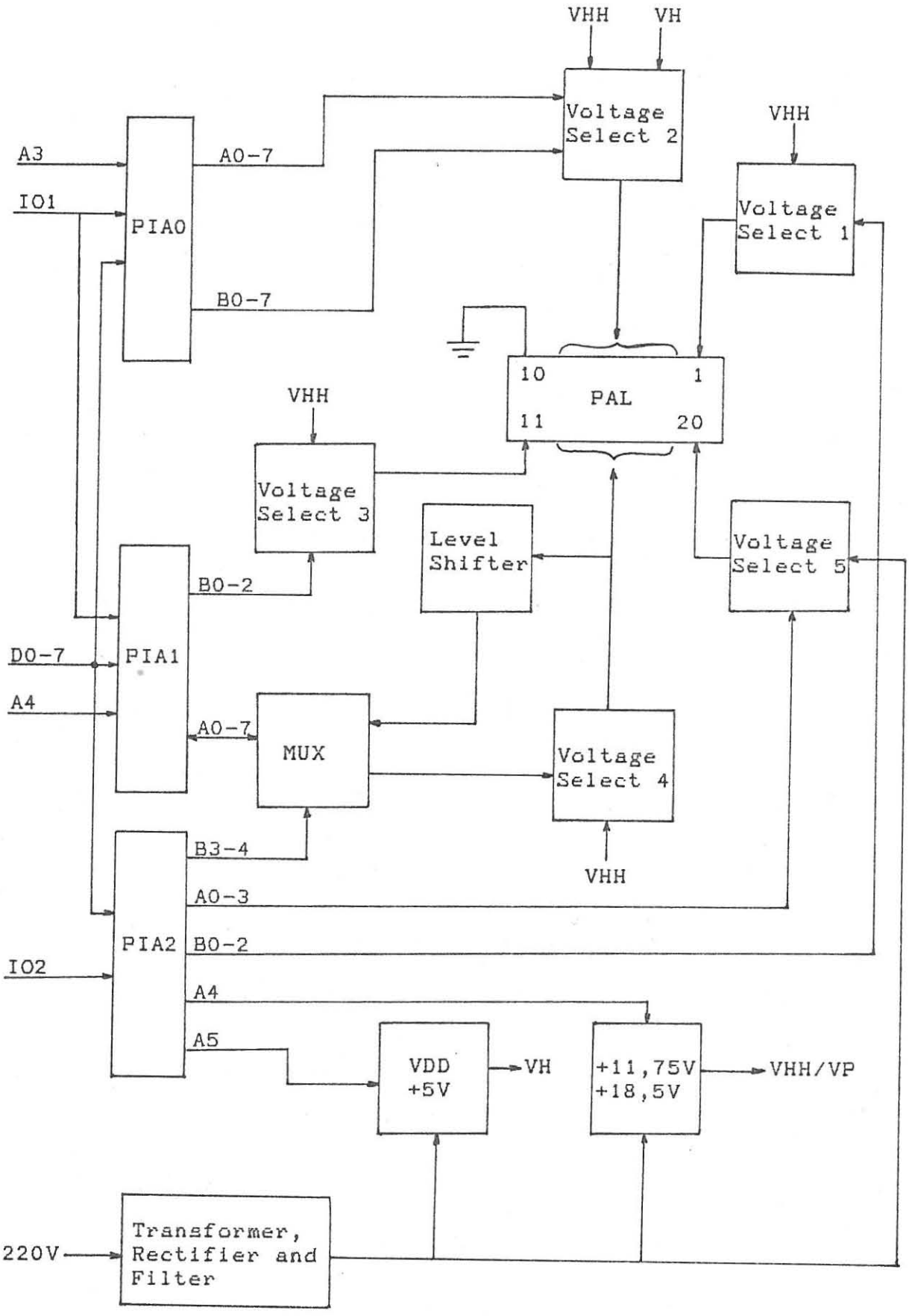


Fig. 6.1 - Block Diagram of PAL programmer.

6.1.2 Voltage Select 1 to 5

The voltages connected to the PAL pins depends upon the function of these pins during programming, and must be carefully controlled. At any moment in time, the power supply produces a range of voltages, which are connected to the voltage select circuits. In this manner a voltage

Table 6.1 - PAL pins and port connections

PAL pin	Function	VHH(11,75V)	VH(5V)	Ground	Miscellaneous
1	OD/CLK	2B0	2B1	2B2	
2	I0	0A0	0B0		
3	I1	0A1	0B1		
4	I2	0A2	0B2		
5	I3	0A3	0B3		
6	I4	0A4	0B4		
7	I5	0A5	0B5		
8	I6	0A6	0B6		
9	I7	0A7	0B7		
10	PAL ground pin				
11	CLK/OD	1B0	1B1	1B2	
12	LR/O3	1A7			2B3 - MUX1 (0 - Input) (1 - Output) (pins 12-15)
13	A2/O2	1A6			
14	A1/O1	1A5			
15	A0/O0	1A4			
16	O3/A2	1A3			2B4 - MUX1 (0 - Input) (1 - Output) (pins 16-19)
17	O2/A1	1A2			
18	O1/A0	1A1			
19	O0/LR	1A0			
20	PAL supply voltage (as in table 6.2)				

select circuit can for instance be supplied with VHH and VH.

One of these voltages is selected by the voltage select circuit to be applied to the PAL pin - according to the logic levels applied to the circuit by the appropriate PIA outputs.

6.2 Power Supply

The power supply may be regarded as consisting of four major sections. Each of these sections will be considered in some detail.

The power supply has been designed in such a way that its outputs are disabled during power up. This is done to prevent voltages, which may have disasterous effects on the PAL to be programmed, from being applied to it during power-up. This disabling is achieved by bit 2A5 of PIA2 biasing a transistor on and pulling the control pins of the regulators low. As is obvious from table 6.2, this condition is then reversed, and the power supply activated, by a logic Ø on bit 2A5 (see section 5.5 and the programmer circuit diagram).

6.2.1 Transformer, Rectifier and Filter

A 220 volt to 15 volt step-down transformer supplies a rectifier bridge. The fullwave rectified output of the bridge is smoothed with a 6800 microfarad, 25 volt,

capacitor.

6.2.2 VDD/V

The integrated circuits forming part of the programmer is supplied with 5 volt. The VH voltage required on some PAL pins during programming should also be approximately 5 volt (logic 1).

A 317T voltage regulator is used as regulating element, with a 2,2 kilo-ohm preset potentiometer, VR8, for calibration purposes.

6.2.3 VHH/VP

During programming, a voltage of 11,75 volt, referred to as VHH, is required on some pins of the PAL. During programming of the security fuses, this voltage is referred to as VP, and is increased to 18,5 volt (on pins 1 and 11). These voltages are supplied by a seperate section of the power supply.

Table 6.2 - Control of power supply

	Port pin						Output voltage	
	2A5	2A4	2A3	2A2	2A1	2A0	VHH/VP	PAL VCC
	1	X	X	X	X	X	Off	Off
D	0	0	0	0	0	0	18,5	11,75
A	0	1	0	0	0	1	11,75	5
T	0	1	0	0	1	0	11,75	5,5
A	0	1	0	1	0	0	11,75	4,5
	0	1	1	0	0	0	11,75	6

The output voltage is controlled, as shown in table 6.2, by bit 2A4 of PIA2. With an 18,5 volt output, two preset potentiometers (VR5 and VR6 on the circuit diagram) are connected in series between the control pin of a 317T regulator and ground. One of the potentiometers (VR5) is shorted by a transistor with a forward bias when the output voltage must be decreased to 11,75 volt.

6.2.4 PAL Supply Voltage

As is obvious from table 6.2, the voltage applied to pin 20 of the PAL should be controllable between 4,5 and 11,75 volt in a number of steps. These voltages are obtained as follows:

A 4,7 kilo-ohm preset potentiometer (VR7) is connected between the control pin of a 317T regulator and ground. Four other preset potentiometers (VR1 to VR4), each in series with a transistor, are connected in parallel with this. The transistors are controlled by bits 2A0 to 2A3. With all transistors off, a relatively high resistance between the control pin and ground results in a high output voltage, adjustable to 11,75 volt. By selectively switching on different transistors, the required voltages can be obtained.

6.3 Output Disable (OD) and Clock Pins

These pins are controlled by numbers 1 and 3 voltage select circuits in figure 6.1. From table 6.1 it is obvious that

in addition to VH and VHH, these pins are also to be grounded.

Each of these pins is provided with a circuit as shown in figure 6.2.

During normal operation, either Q1 and Q2, or Q3 and Q4, or Q5 will be biased on. With Q1 and Q2 on, it is effectively connected to VHH. If both combinations are switched on simultaneously, a reverse biased diode, D1, prevents VHH from being connected to the collector of Q4, possibly damaging the transistor.

Q5 should never be biased on simultaneously with any of the other combinations, since this would cause a virtual short circuit between VH or VHH and ground.

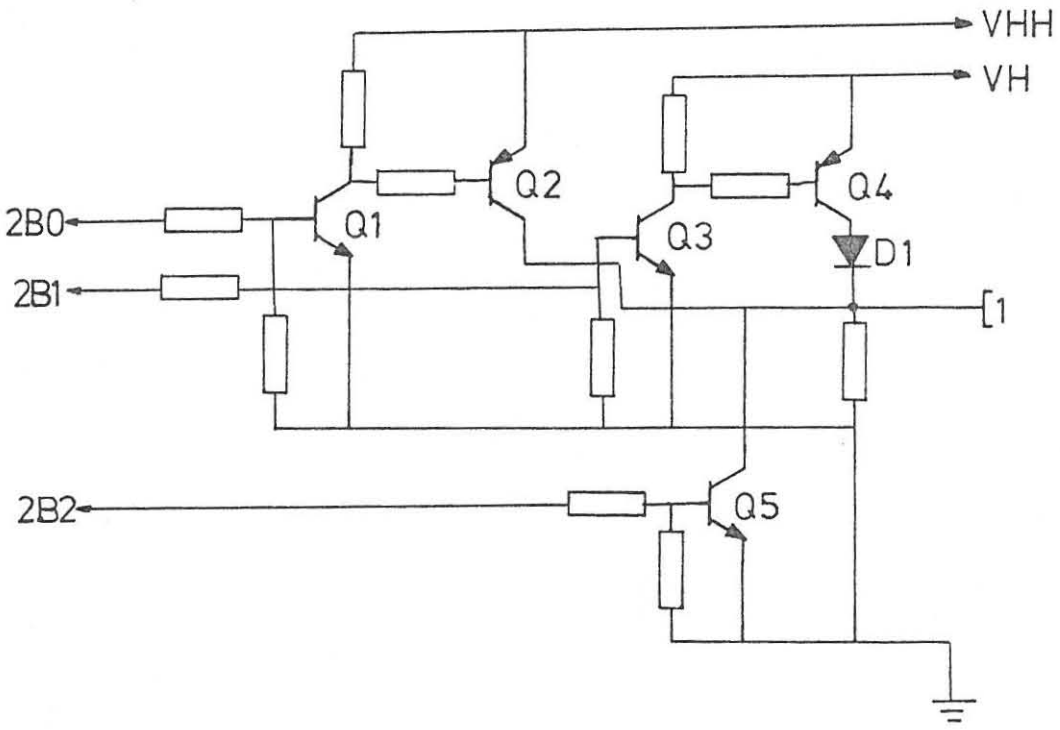


Fig. 6.2 - Circuit diagram of OD/CLK pins (no's 1 and 11).

One other possibility may also arise: At times the pin should be kept at a logic zero level (which is not to be confused with the virtual 0 volts, as controlled by Q5). A pull-down resistor is shown in figure 6.2 between the PAL pin and ground. The value of this resistor is such that the voltdrop across it (with no other input voltage activated), at the maximum low-level input current of 0,25 milliampere, is significantly below the maximum low-level input voltage of 0,8 volt (PAL Handbook by Monolithic Memories, 1983: 3.12).

6.4 Input Lines (Voltage Select 2)

The input lines are addressed during programming by means of pins 2 to 9 - as well as the LR pin (pin 12 or 19, depending on the product line), which is discussed in section 6.5.

These pins should be connected to a combination of VHH, VH and VL (logic 0). This is achieved by eight circuits (one for each pin) similar to that shown in figure 6.3. The relationship between port output and voltage applied to the PAL pin can be summarized as follows:

Port 0A	Port 0B	Voltage on pin
0	0	VL
0	1	VH
1	0	VHH
1	1	VHH

The operation of the circuit is exactly the same as that of the circuit discussed in section 6.3, with the exception

that these pins are never pulled down to zero volts.

6.5 Product Lines and L/R Pin (pins 12 to 19)

PAL pins 12 to 19 are used to address the product lines and the L/R pin. Any of these pins may be switched to either 11,75 volt, or to a high-impedance during programming. The high-impedance referred to is a voltage source of 5 volt, connected through a 10 kilo-ohm resistor (Programmable Logic Databook by National Semiconductor Corporation, 1983: 24.24).

Data is read from these pins during verification. For this reason it is preferable that the pins should also be entirely disconnected from any of these voltages.

Eight circuits, similar to the one given in Figure 6.4, are used to control these pins - the only difference being that only two transistors, and not eight, are used to disconnect the pins from the high-impedance - e.g. 10 kilo-ohm to 5V

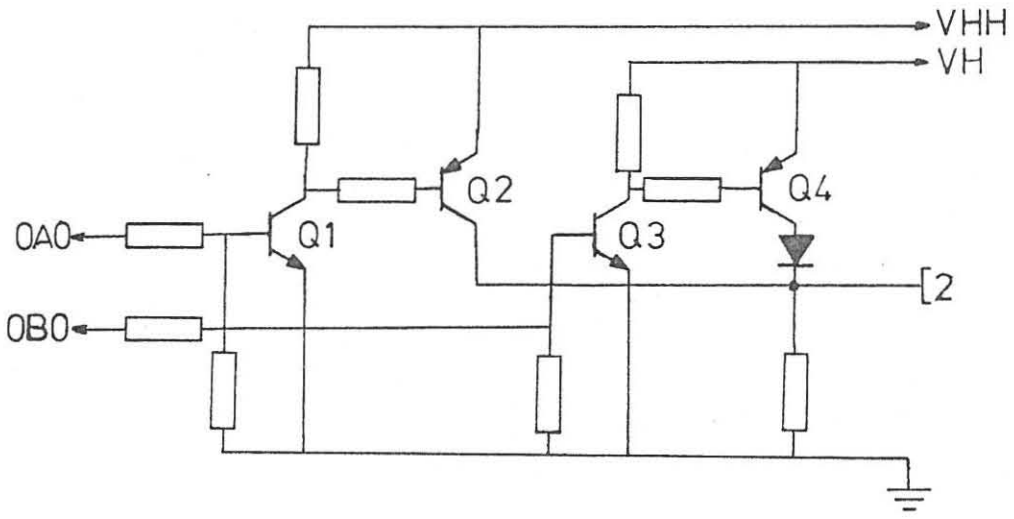


Fig. 6.3 - Control of input lines (pins 2 to 9).

(Birkner and Coli, 1983: 3.22). Transistor Q51 disconnects pins 12 to 15, and Q52 pins 15 to 19 (see Appendix F).

6.5.1 Multiplexers

As mentioned previously, port 1A is used to read data regarding the condition of the PAL fuses during the verification routine. This reading is done by means of eight MUXs, connected as shown in figure 6.4, controlled by bits 2B3 and 2B4 of port 2B. The control of the MUXs are as specified in table 6.1. The programmer is provided with four 4053 CMOS multiplexer chips for this purpose.

6.5.2 Level Shifter

During programming, a voltage of 11,75 volt may appear on the PAL pin, and would therefore be applied to the MUX. Since the voltage on the MUX input may not exceed the chip supply voltage of 5 volt by such a large margin,

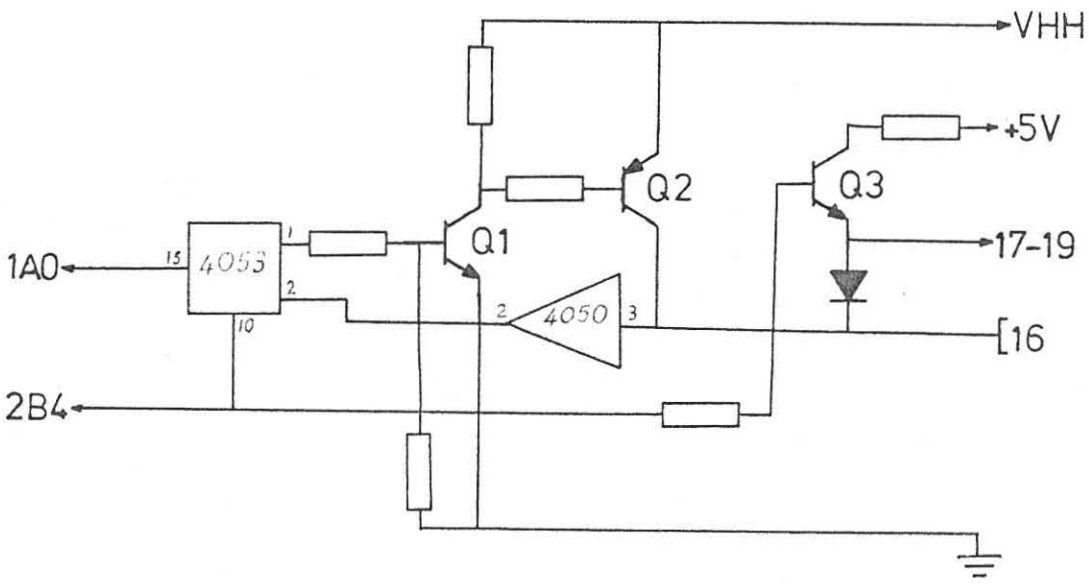


Fig. 6.4 - Control of product lines and L/R pin (pins 12-19)

a level shifter is inserted between the PAL pin and the MUX. This device limits the input voltage of the MUX to a maximum of 5 volt. The programmer is provided with two 4050 CMOS chips for this purpose.

6.6 Programmer Construction

The programmer consists basically of one printed circuit board. Another PC board is used, as described in section 6.1, to connect the computer and programmer. Yet another small printed circuit board is connected to the main board by means of a flat ribbon cable, and is provided with a 20 pin DIP socket. The PAL to be programmed is inserted into this socket.

The printed circuit layout was designed with "Tango PCB" and plotted with a Sekonic SPL 410 plotter.

6.7 Summary

The eventual cost of the programmer was a determining factor in the selection of components to be used in the design. This, for instance, is the reason why common, discrete transistors were used rather than transistor arrays. A considerable saving was realized with this approach - in fact more than half of the total component cost. Using discrete components in the programmer design, however, resulted in a relatively large printed circuit board - but not prohibitively so.

A very similar transistor arrangement, with the following characteristics, is used as part of every voltage select circuit:

1) Voltage gain achieved without a phase change between input and output.

2) A very low saturation voltage at fairly high currents is achieved.

Because the basic design meet these requirements exceptionally well, it was decided to use it throughout the programmer - even if other less complicated circuits may have sufficed.

Certain advantages may have been derived from developing an intelligent programmer, rather than using the computer to control the actual programming process. This would however, have resulted in a more complex, and certainly more expensive, design.

CHAPTER 7

Evaluation of Programmer

The evaluation of the performance of an electronic system such as the PAL programmer can be a very complicated process. This is especially true in this case since all the nodes of a programmed PAL cannot be reached to determine the exact effect that the programming operation had on the chip. Such an evaluation would in any case be far beyond the aim of the project. The important programmer performance criteria with respect to the project - which had to be considered - are the following:

1) Is the fuse map an exact realization of the Boolean expressions supplied during execution of Fuse Map 20?

--- This could be verified by checking a number of fuse maps by hand and by verifying the operation of a number of circuits programmed with these fuse maps.

2) Is the basic circuit operation under static and dynamic conditions as required?

--- Programs were developed to check the voltage levels and reaction times of the different circuit elements.

3) Is the fuse addressing and programming according to specifications?

--- This was checked in two ways. A logic analyser was

used to verify the addressing of the fuses, and a number of PALs were programmed successfully. This should be sufficient proof of the programmer's operation.

The rest of the chapter deals with two such circuits which were realized with the programmer as developed.

CIRCUIT	PAL
Full adder	14H4
Decade counter	16R8

Section 7.1 describes the design of the full adder circuit and section 7.2 that of the decade counter.

7.1 Full Adder

The design of the full adder can be considered with respect to a truth table of the circuit (see table 7.1). The following variables and pin functions were decided upon:

VARIABLE	I/O	FUNCTION	PIN NUMBER
A	Input	Bit A	1
B	Input	Bit B	2
C	Input	Carry In	3
X	Output	Sum	17
Y	Output	Carry Out	15

The following two Boolean expressions, written in the form of sum-of-products, can be derived from the truth table. The format of the expressions, as referred to in this chapter,

Table 7.1 - Truth table off full adder

Input			Output		
A	B	C	X	Y	
0	0	0	0	0	
0	0	1	1	0	!A!BC
0	1	0	1	0	!AB!C
0	1	1	0	1	!ABC
1	0	0	1	0	A!B!C
1	0	1	0	1	A!BC
1	1	0	0	1	AB!C
1	1	1	1	1	ABC

is as required by the Fuse Map 20 program). The expressions were not simplified since no real advantage would have resulted from any simplification (see also chapter 2).

Sum: X=!A!C+!AB!C+A!B!C+ABC

Carry: Y=!ABC+A!BC+AB!C+ABC

Figure 7.1 shows the hard copy output of the Fuse Map 20 program. From this the variables as defined above can be checked and even the Boolean terms as such can be verified before a PAL is programmed according to the fuse map.

The operation of the circuit proved to be as predicted by the truth table.

7.2 Decade Counter

The count table of the decade counter is used in the design

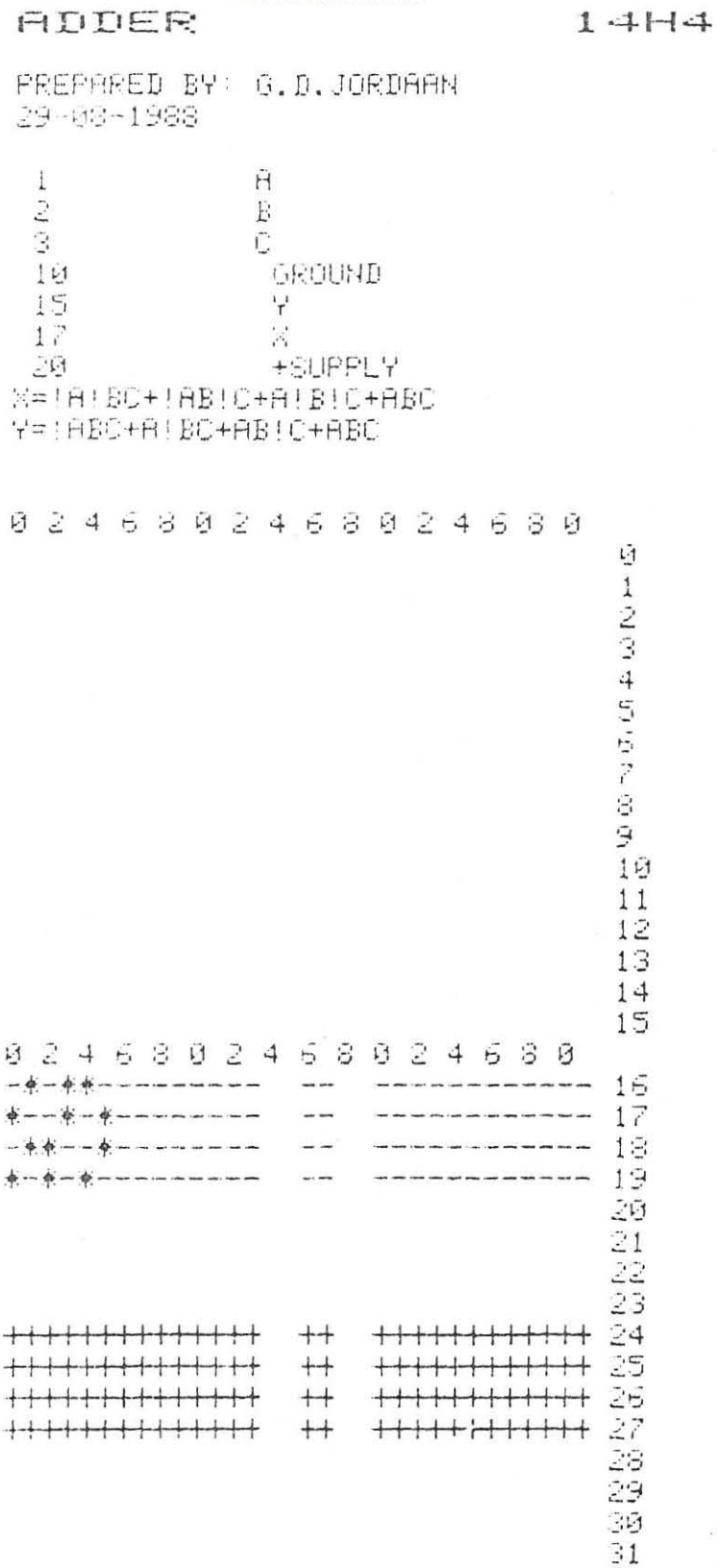


Fig. 7.1 - Hardcopy of full adder fuse map (continued on next page).

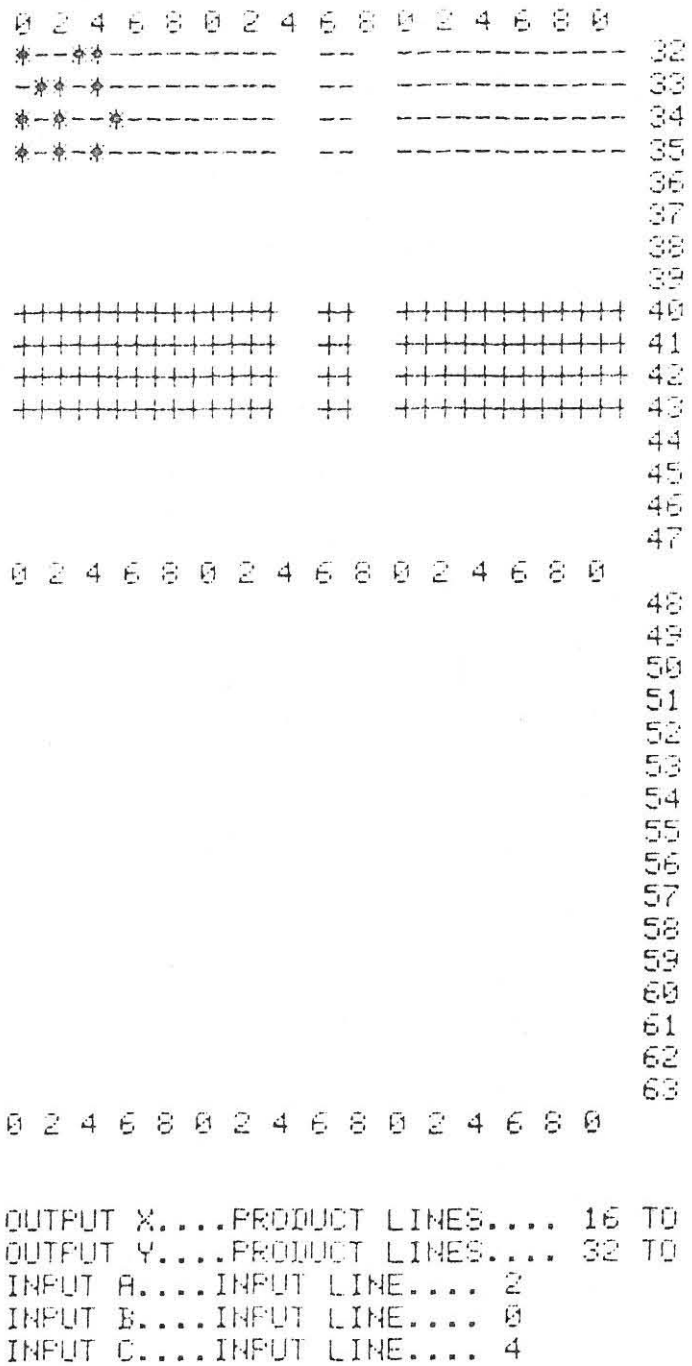


Fig. 7.1 (continued) - Hardcopy of full adder

of the circuit. Upon power-up the initial state of the output registers are unknown - and uncontrollable. This may force the counter into an "illegal" state (such as 1101 in binary). This eventuality was taken into consideration during the design of the counter in that the counter is forced to a count of zero by the first clock pulse if an illegal state should occur (Vafai, 1983: 8.81).

Table 7.2 is the count table of a decade counter. The table includes the required binary outputs and inversion of the outputs, from which the data required on the data inputs of the flip-flops are derived, as well as the data.

This rather complicated procedure is necessary because the 16R8 PAL is an active low output device, which is used as an active high in this application. From the count table, as shown in table 7.2, the relevant Karnaugh maps, can be drawn.

The Boolean expressions were encoded by excution of the Fuse Map 20 program. A PAL was programmed according to these and

Table 7.2 - Count table of decade counter.

!W	!X	!Y	!Z	W	X	Y	Z	DW	DX	DY	DZ
0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	1	1	0	1
0	0	1	0	1	1	0	1	1	1	0	0
0	0	1	1	1	1	0	0	1	0	1	1
0	1	0	0	1	0	1	1	1	0	1	0
0	1	0	1	1	0	1	0	1	0	0	1
0	1	1	0	1	0	0	1	1	0	0	0
0	1	1	1	1	0	0	0	0	1	1	1
1	0	0	0	0	1	1	1	0	1	1	0
1	0	0	1	0	1	1	0	0	1	0	1
1	0	1	0	0	1	0	1	1	1	1	1

YZ \ WX	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	1	0	1	1
10	1	1	1	1

!W=WX+!W!X+!W!Y+WZ+Y!Z

YZ \ WX	00	01	11	10
00	1	1	0	1
01	1	1	1	0
11	1	1	1	0
11	1	1	1	0

!X=!W+XZ+XY+!X!Y!Z

YZ \ WX	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	1	1	1	1
10	1	1	0	0

!Y=!W+!Y!Z+YZ

YZ \ WX	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	1	0	0	0
10	1	1	1	1

!Z=!Z+!W!Y+!W!X

Fig. 7.2 - Karnaugh maps - decade counter.

its operation proved to be satisfactory. Figure 7.3 is a hard copy of the decade counter as discussed.

7.3 Summary

The fuse map generating software was evaluated to a considerable extent during the program development cycle. However, no evaluation of the programming software would be complete without the actual programming of a number of PALs. This also substantiates the accuracy of the fuse map generating software.

```
PREPARED BY: G.D.JORDAAN
29-08-1988

 1              CLOCK
 10              GROUND
 11              ENABLE(L)
 16              !Z
 17              !Y
 18              !X
 19              !W
 20              +SUPPLY
!W=WX+!W!X+!W!Y+WZ+Y!Z
!X=!W+XZ+XY+!X!Y!Z
!Y=!W+!Y!Z+YZ
!Z=!Z+!W!Y+!W!X
```

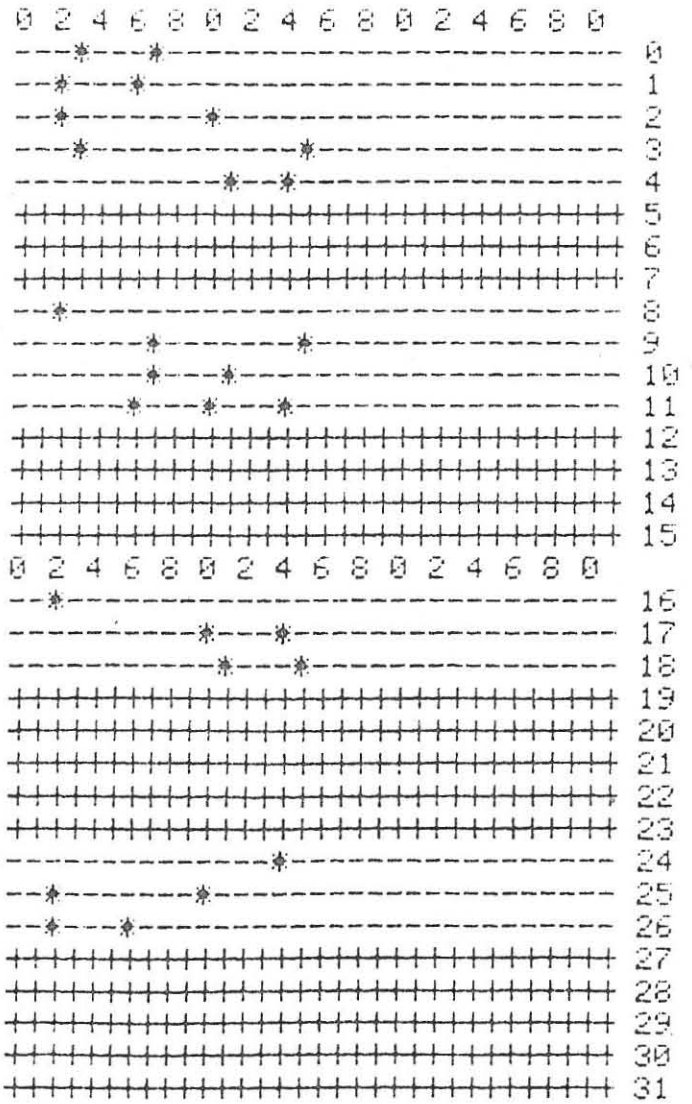
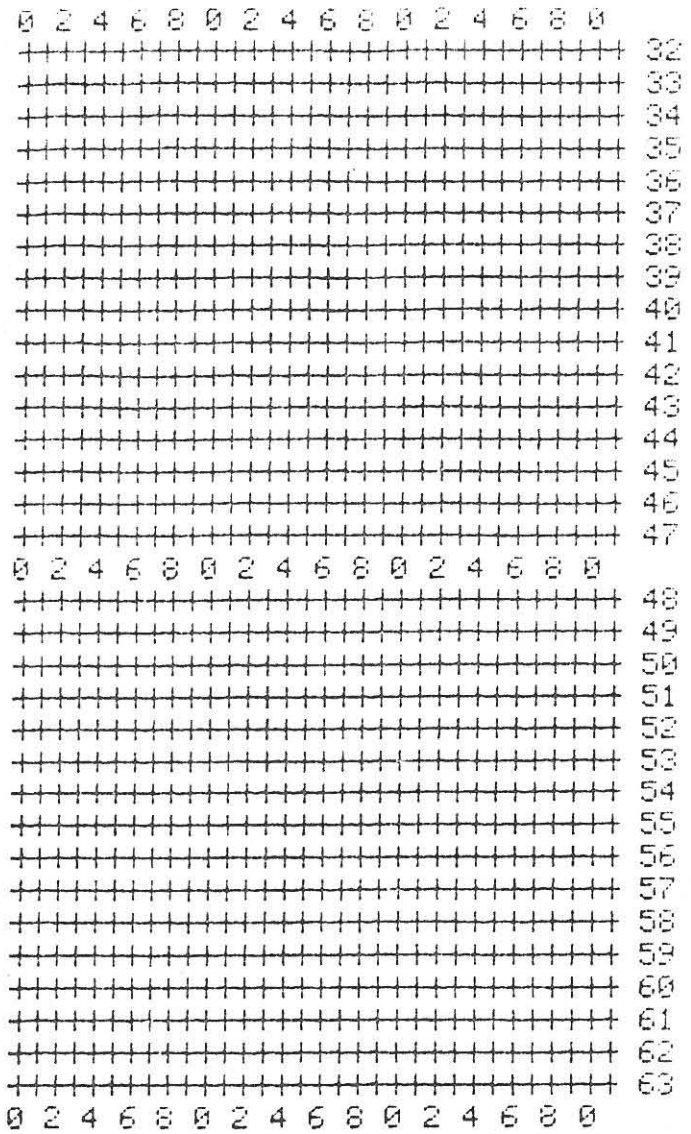


Fig. 7.3 - Hardcopy of decade counter fuse map (continued overleaf).




```
OUTPUT !W....PRODUCT LINES.... 0 TO 7
OUTPUT !X....PRODUCT LINES.... 8 TO 15
OUTPUT !Y....PRODUCT LINES.... 16 TO 23
OUTPUT !Z....PRODUCT LINES.... 24 TO 31
OUTPUT !Z....INPUT LINE.... 14
OUTPUT !Y....INPUT LINE.... 10
OUTPUT !X....INPUT LINE.... 6
OUTPUT !W....INPUT LINE.... 2
```

Fig. 7.3 (continued) - Hardcopy of decade counter fuse map.

The two circuits discussed can hardly be considered to be a comprehensive evaluation of the programmer performance, yet it verifies its basic operation. During evaluation a substantial number of PALs were programmed and no problems were experienced. In fact the programming was very successful.

CHAPTER 8

SUMMARY

The main aim of the project was the investigation of the programming principles and characteristics of programmable array logic devices. The development of the PAL programmer was used as a vehicle to facilitate these investigations.

The relative position of PALs as a programmable logic device is discussed in chapter 2. In this chapter they are compared in particular with PROMs and PLAs.

In particular the following three PAL characteristics were to be investigated.

1) Fuse layout and operation of PALs.
2) Addressing of these fuses.
3) Parameters of the pulses required to blow the fuses.

A number of programs were developed during the course of the project. The programming style used in these is discussed in chapter 3.

The fuse layout and operation of PALs were investigated by developing the fuse map generating software as described in chapter 4. By programming and testing a number of PALs, it was also possible to verify the accuracy of principles as understood during this process.

No PAL can be blown sucessfully unless the appropriate fuses are addressed. The manner in which the fuses are addressed was investigated by developing software for this purpose. This was done in machine code as well as in BASIC. These programs are described in chapter 5.

The parameters of the programming signals were investigated by designing the hardware of the programmer. A Commodore 64 was used as controlling element in the design. Although it is appreciated that an IBM PC, or compatible, computer (which incidentally were not available at the inception of the project), may have been a more acceptable choice, the Commodore was found to be quite suitable for the purpose of the project.

Chapter 6 deals with the design of the hardware external to the Commodore 64. The operation of the programmer as designed and constructed proved to be very satisfactory.

The design of a full adder and a decade counter, as implemented in a PAL, is described in chapter 7. These devices were programmed and its operation, as compared to its truth tables, proved to be as anticipated.

During the development phase the author became very much aware of the relative scarcity of documentation on PALs as compared to for instance PLAs. Although a fair amount of information on PALASM became available as the project progressed, this was not used. In this way it was ensured that all the principles involved were indeed understood rather

than carried over from another system.

The selection of the project proved to be very suitable as a means of investigating the characteristics of PALs. In this manner direction was given in studying the underlying principles of operation and programming of this member of the programmable logic family.

PAL File

```
10 REM CREATES DATA FILE ON SPECIFIED PAL
15 REM INSERT NUMBER OF PAL IN LINE NUMBER 2000
17 REM REMOVE @ IF NOT REQUIRED
20 PRINT"[illegible]DATA SHOULD BE SUPPLIED AS FOLLOWS"
30 PRINT"[illegible]________________________________________________"
40 PRINT:PRINT"1)CONSIDER PINS FROM NO. 1 TO 20 OR 24"
45 PRINT
50 PRINT"2)STATE FUNCTION OF PIN, E.G. INPUT"
55 PRINT
60 PRINT"  IF INPUT THEN"
70 PRINT"    FOLLOWED BY INPUT LINE"
75 PRINT
80 PRINT"  IF OUTPUT THEN"
90 PRINT"    FOLLOWED BY PRODUCT LINE
100 PRINT"    FOLLOWED IMMEDIATELY BY NUMBER"
110 PRINT"    OF PRODUCT LINES"
112 PRINT"    FOLLOWED BY YES OR NO DEPENDING ON:"
113 PRINT"    IF THE OUTPUT HAS FEEDBACK-YES
114 PRINT"                   NO FEEDBACK-NO
115 PRINT"    FOLLOWED BY INPUT LINE IF YES"
120 PRINT"  IF GROUND, +SUPPLY OR ENABLE
130 PRINT"    PROCEED TO THE NEXT PIN"
135 PRINT:PRINT"INPUT END WHEN FINISHED":PRINT
140 PRINT"         PRESS ANY KEY TO PROCEED"
150 GET A$:IF A$="" THEN GOTO 150
160 PRINT"[illegible]IDENTIFY PAL";:INPUT X$
170 INPUT"NUMBER OF PINS";Y
2000 OPEN 2,8,2,"@0:"+X$+",S,W"
2020 FOR T=1 TO Y
2030 PRINTT,
2040 INPUT A$
2050 PRINT#2,A$
2060 IF A$="INPUT" THEN INPUT"INPUT LINE";A$:GOTO 2160
2065 IF RIGHT$(A$,6)<>"OUTPUT" THEN GOTO 2170
2070 IF A$<>"OUTPUT"THEN GOTO 2170
2080 INPUT"PRODUCT LINE";A$
2090 PRINT#2,A$
2100 INPUT"NUMBER OF TERMS";A$
2110 PRINT#2,A$
2120 INPUT"WITH FEEDBACK";A$
2130 IF A$="NO" THEN 2160
2140 PRINT#2,A$
2150 INPUT"INPUT LINE";A$
2160 PRINT#2,A$
2170 NEXT T
2180 CLOSE 2

READY.
```

APPENDIX B

Test PAL File

```
5 REM CHECK ACCURACY OF PAL FILE
10 INPUT "IDENTIFY PAL";P$
15 INPUT "NUMBER OF PINS";T
20 OPEN 2,8,2,"0:"+P$+",S,R"
24 IF T=P THEN CLOSE 2:END
25 PRINT"PRESS ANY KEY TO CONTINUE":PRINT
26 GET K$:IF K$="" THEN GOTO 26
27 P=P+1
28 PRINT"PIN...";P
29 PRINT
30 INPUT#2,A$
40 IF A$="INPUT" THEN PRINTA$:INPUT#2,A$:PRINT"INPUT LINE...",A$:GOTO 24
50 IF A$<>"OUTPUT" THEN PRINTA$:GOTO 24
60 PRINTA$
70 INPUT#2,A$
80 PRINT"PRODUCT LINE...";A$
90 INPUT#2,A$
100 PRINT"NUMBER OF PRODUCTS...";A$
110 INPUT#2,A$
120 IF A$="NO" THEN PRINTA$;"FEEDBACK":GOTO 25
130 PRINT"WITH FEEDBACK"
140 INPUT#2,A$
150 PRINT"INPUT LINE...";A$
160 GOTO 24

READY.
```

APPENDIX C

Fuse Map 20

```
10 PRINT"FUSE MAP 20---GENERATING A PAL FUSE MAP"
20 PRINT"-------------------------------------------------------"
30 PRINT"THE PAL TO BE USED MUST FIRST BE "
40 PRINT"                IDENTIFIED"
50 PRINT
60 PRINT
70 INPUT "IDENTIFY PAL";X$
80 IF MID$(X$,3,1)="L"OR MID$(X$,3,1)="R"THEN PRINT"OUTPUT IS LOW WHEN ACTIVE"
90 PRINT:INPUT"IDENTIFY CIRCUIT";Y$:                        REM IDENTIFY CIRCUIT
100 DIM P$(20),F%(31,63),L%(20),PL%(20),NP%(20),OP$(20),F$(20),FB$(20)
110 OPEN 2,8,2,"0:"+X$+",S,R":                          REM READ PERSONALITY FILE
120 FOR I=1 TO 20                                          :REM RESET INPUT LINES
130 L%(I)=32:P$(I)=""
140 NEXT I
150 X=X+1:PRINTX;
160 INPUT#2,A$:PRINTA$;
170 F$(X)=A$
180 IF A$="INPUT" THEN INPUT#2,L%(X):INPUT P$(X):GOTO 230
190 IF A$="OUTPUT" THEN GOTO 1680
200 IF (A$="INPUT") OR (A$="OUTPUT") THEN PRINT A$:GOTO 220
210 PRINT
220 P$(X)=A$
230 IF X<20 THEN GOTO 150
240 CLOSE 2
250 PRINT:PRINT
260 PRINT"                SUMMARY":              REM DISPLAY SUMMARY
270 PRINT
280 PRINT"PIN";TAB(5)"DESCRIPTION";TAB(20)"PIN";TAB(25)"DESCRIPTION"
290 PRINT"---";TAB(5)"-----------";TAB(20)"---";TAB(25)"-----------":PRINT
300 FOR I=1 TO 10
310 PRINTI;TAB(5)P$(I),TAB(20)I+10;TAB(25)P$(I+10)
320 NEXT
330 PRINT
340 PRINT"ARE THE PIN FUNCTIONS AS REQUIRED?";
350 GET A$:IF A$="" THEN GOTO 350
360 PRINT
370 IF A$="N" THEN PRINT"HAVE ANOTHER GO AT IT!":X=0:GOTO 110
380 PRINT
390 PRINTTAB(6) "PLEASE SUPPLY AN EXPRESSION":      REM INPUT BOOLEAN EXPRESSIONS
400 PRINTTAB(13)"TO BE REALIZED"
410 PRINT:PRINTTAB(24)"END IF FINISHED"
420 PRINTTAB(24)"---"
430 I=1
440 INPUT OP$
450 OP$(I)=OP$
460 IF OP$<>"END" THEN GOSUB 1720
470 IF S=1 THEN S=0:GOTO 440
480 PRINT"UNUSED FUSES IDENTIFIED AND PROCESSED":        REM UNUSED FUSES
490 PRINT:PRINT"PLEASE WAIT....."
500 FOR T=0 TO 63
510 I=-1
520 I=I+1
530 IF F%(I,T)=1 THEN GOTO 590
540 IF I<31 THEN GOTO 520
550 FOR I=0 TO 31
560 F%(I,T)=2
570 NEXT I
580 D=0
590 NEXT T
600 X=-2                                         :REM ADJUST FOR PHANTOM FUSES
610 X=X+2
620 I=0
630 I=I+1
640 IF L%(I)=X THEN GOTO 700
650 IF I<20 THEN GOTO 630
660 FOR T= 0 TO 63
670 F%(X,T)=3
680 F%(X+1,T)=3
690 NEXT T
700 IF X<32 THEN GOTO 610
710 R=0
720 P=20
```

```
730 P=P-1
740 IF F$(P)<>"OUTPUT" THEN GOTO 830
750 Q=PL%(P):S=NP%(P)
760 IF Q-1<=R THEN GOTO 820
770 FOR X=R TO Q-1
780 FOR Y=0 TO 31
790 F%(Y,X)=3
800 NEXT Y
810 NEXT X
820 R=PL%(P)+NP%(P)
830 IF P>10 GOTO 730
840 FOR X=Q+S TO 63
850 FOR Y=0 TO 31
860 IF X>63 THEN GOTO 900
870 F%(Y,X)=3
880 NEXT Y
890 NEXT X
900 PRINT "FUSE MAP DISPLAY REQUIRED?":                    REM F/M DISPLAYED
910 GET DP$:IF DP$="" THEN GOTO 910
920 IF DP$="N" THEN GOTO 1070
930 PRINT
940 PRINT"0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0"
950 FOR T=0 TO 63
960 FOR I=0 TO 31
970 IF F%(I,T)=0 THEN PRINT"-";
980 IF F%(I,T)=1 THEN PRINT"*";
990 IF F%(I,T)=2 THEN PRINT"+";
1000 IF F%(I,T)=3 THEN PRINT" ";
1010 NEXT I
1020 PRINTT
1030 IF INT((T+1)/16)<>(T+1)/16 THEN GOTO 1060
1040 PRINT"0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0  (SPACE)
1050 GET T$:IF T$="" THEN GOTO 1050
1060 NEXT T
1070 INPUT "FUSE MAP TO BE SAVED";FM$:                     REM SAVE FUSE MAP
1080 IF LEFT$(FM$,1)<>"Y" THEN GOTO 1190
1090 PRINT"■■■■■■INSERT FUSE MAP DISK"
1100 PRINT:PRINT"PRESS ANY KEY WHEN READY"
1110 GET K$:IF K$="" THEN GOTO 1110
1120 OPEN 3,8,3,"@0:"+Y$+",S,W"
1130 FOR I=0 TO 31
1140 FOR T=0 TO 63
1150 PRINT#3,F%(I,T)
1160 NEXT T
1170 NEXT I
1180 CLOSE 3
1190 PRINT"HARDCOPY REQUIRED?"
1200 GET H$:IF H$="" THEN GOTO 1200
1210 IF H$="N" THEN GOTO 1660
1220 OPEN 4,4
1230 CMD 4
1240 PRINT#4,CHR$(14)Y$,X$
1250 PRINT#4,CHR$(15)
1260 INPUT"NAME OF OPERATOR";N$
1270 PRINT#4,"PREPARED BY: ";N$
1280 INPUT"DATE";N$
1290 PRINT#4,N$
1300 PRINT"■■PRINTING...."
1310 PRINT#4,CHR$(15)
1320 FOR I=1 TO 20
1330 IF P$(I)=""THEN NEXT I
1340 PRINT#4,I,P$(I)
1350 NEXTI
1360 I=1
1370 IF OP$(I)="END" THEN PRINT#4,CHR$(10):GOTO 1390
1380 PRINT#4,OP$(I):I=I+1:GOTO 1370
1390 PRINT#4,"0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0"
1400 FOR T=0 TO 63
1410 FOR I= 0 TO 31
1420 IF F%(I,T)=0 THEN PRINT#4,"-";
1430 IF F%(I,T)=1 THEN PRINT#4,"*";
1440 IF F%(I,T)=2 THEN PRINT#4,"+";
1450 IF F%(I,T)=3 THEN PRINT#4," ";
1460 NEXT I
1470 PRINT#4,T
1480 IF (T+1)/16<>INT((T+1)/16) THEN GOTO 1500
1490 PRINT#4,"0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0"
```

```
1500 NEXT T
1510 PRINT#4,CHR$(10)
1520 FOR K=19 TO 11 STEP -1
1530 IF P$(K)="" THEN 1570
1540 IF F$(K)<>"OUTPUT" THEN 1570
1550 IF FB$(K)="YES" THEN F=-1
1560 PRINT#4,F$(K);" ";P$(K);"....PRODUCT LINES....";PL%(K);"TO";PL%(K)+NP%(K)+F
1570 NEXT K
1580 FOR K=1 TO 20
1590 IF P$(K)="" THEN GOTO 1640
1600 IF F$(K)="INPUT"THEN GOTO 1630
1610 IF F$(K)<>"OUTPUT" THEN GOTO 1640
1620 IF FB$(K)<>"YES" THEN GOTO 1640
1630 PRINT#4,F$(K);" ";P$(K);"....INPUT LINE....";L%(K)
1640 NEXT K
1650 CLOSE 4
1660 PRINT"[illegible]END OF PROGRAM!"
1670 END
1680 INPUT#2,PL%(X),NP%(X),FB$(X)
1690 IF LEFT$(FB$(X),1)="Y" THEN INPUT#2,L%(X)
1700 INPUT P$(X)
1710 GOTO 230
1720 X=11:                                          REM DETERMINE PRODUCT LINE
1730 VAR$=LEFT$(OP$,1)
1740 IF VAR$="!" THEN VAR$=LEFT$(OP$,2):AL=1
1750 IF X>20 THEN PRINT"ILLEGAL VARIABLE. SUPPLY ANOTHER EXPRESSION.":GOTO 2000
1760 IF VAR$<>P$(X) THEN X=X+1:GOTO 1750
1770 L=PL%(X)
1780 OUT=X
1790 Y=3+AL
1800 X=1
1810 VAR$=MID$(OP$,Y,1)
1820 IF VAR$="+" OR VAR$="(" THEN GOTO 1950:                    REM NEXT TERM
1830 IF VAR$=")" OR VAR$="" THEN I=I+1:GOTO 2000:           REM NEXT EXPRESSION
1840 IF VAR$="#" THEN L=L-1:Y=Y+1:VAR$=MID$(OP$,Y,1):         REM ENABLING TERM
1850 IF VAR$="!" THEN GOTO 2020
1860 IF X>20 THEN PRINT"UNDEFINED VARIABLE.SUPPLY ANOTHER EXP.":GOTO 2000
1870 IF (VAR$<>P$(X)) AND (VAR$=RIGHT$(P$(X),1)) THEN P=P+1: GOTO 1890
1880 IF VAR$<>P$(X) THEN X=X+1:GOTO 1860:                       REM SET FUSE
1890 P=P+L%(X)
1900 PRINTL,P
1910 F%(P,L)=1
1920 P=0
1930 Y=Y+1
1940 GOTO 1800
1950 L=L+1
1960 Y=Y+1
1970 IF L<PL%(OUT)+NP%(OUT) THEN GOTO 1800
1980 PRINT"EXPRESSION MAY CONSIST OF ONLY";NP%(OUT);"TERMS"
1990 PRINT"ENTER A SUITABLE EXPRESSION"
2000 S=1
2010 RETURN
2020 VAR$=MID$(OP$,Y,2)
2030 T=11:                              REM IDENTIFY ACTIVE LOW OUTPUT AND SET FUSE
2040 IF VAR$=P$(T) THEN Y=Y+1:P=P+L%(T):GOTO 1900
2050 T=T+1
2060 IF T<20 THEN GOTO 2040
2070 P=P+1
2080 Y=Y+1
2090 VAR$=MID$(OP$,Y,1)
2100 GOTO 1860

READY.
```

APPENDIX D

Program PAL - Main Program

```
10 PRINT"■■■■■■■■■■■■■■■■■■PROGRAMMING PAL"
20 PRINT"■■■■■(LOADING PROGRAMMING ROUTINES)"
30 OPEN 2,8,2,"PROGRAM 17,S,R"
40 FOR I=32768 TO 33188
50 INPUT#2,D
60 POKE I,D
70 NEXT I
80 CLOSE 2
90 OPEN 2,8,2,"PROGRAM DATA,S,R"
100 FOR I=30720 TO 30911
110 INPUT#2,D
120 POKE I,D
130 NEXT I
140 CLOSE 2
150 SYS 32768
160 PRINT"■■■■PORTS AND POWER SUPPLY ACTIVATED"
165 PRINT"■■■■■■SHOULD THE FUSE MAP BE LOADED?";
167 GET L$:IF L$="" THEN GOTO 167
168 IF L$="N" THEN GOTO 375
170 PRINT"■■■■■■■■■■■■INSERT FUSE MAP DISK"
180 PRINT"■■■■■■■■(PRESS ANY KEY WHEN READY)"
190 GET A$:IF A$="" THEN GOTO 190
200 PRINT"■■■NAME OF CIRCUIT";
210 INPUT X$
220 PRINT:PRINT:PRINT TAB(3)" ";TAB(36)"| "
230 PRINT TAB(12)"■■LOADING FUSE MAP."
240 PRINT TAB(4)"■■■■";
250 OPEN 2,8,2,"0:"+X$+",S,R"
260 DIM F%(31,63)
270 Z=0
280 FOR I= 0 TO 31
290 FOR T= 0 TO 63
300 INPUT#2,F%(I,T)
310 POKE 28672+Z,F%(I,T)
320 Z=Z+1
330 NEXT T
340 PRINT"●";
350 NEXT I
360 PRINT"■■■■■■■■"
370 CLOSE 2
375 PRINT:PRINT:PRINT:PRINT"■■■■■TYPE OF PAL";:INPUT PAL$
380 PRINT:PRINT:PRINT:PRINT"PAL TO BE PREVERIFIED?"
390 GET P$:IF P$="" THEN GOTO 390
400 IF P$="N" THEN GOTO 450
410 PRINT"■PREVERIFYING..."
420 GOSUB 1000
440 IF UNP=0 THEN PRINT:PRINT"PAL VERIFIED UNPROGRAMMED":GOTO 450
445 PRINT"PAL CANNOT REALIZE FUSE MAP...."
450 PRINT:PRINT TAB(4)"PAL TO BE PROGRAMMED?"
460 GET P$:IF P$="" THEN GOTO 460
470 IF P$="N" THEN GOTO 510
480 PRINT:PRINT"PROGRAMMING PAL..."
490 SYS 32851
500 PRINT:PRINT"PROGRAMMING COMPLETE -"
510 PRINT:PRINTTAB(4)"VERIFICATION TO COMMENCE?"
520 GET P$:IF P$="" THEN GOTO 520
530 IF P$="N" THEN GOTO 580
540 PRINT:PRINT"■■■■VERIFYING..."
541 PRINT"■■■■■■■■■■■■■■■■"
542 PRINT"■SELECT VERIFICATION VOLTAGE"
543 PRINT"■FIRST VERIFICATION PASS..5V....1"
544 PRINT"■HIGH VOLTAGE VERIFY.....5,5V..2"
545 PRINT"■LOW VOLTAGE VERIFY.....4,4V..3"
546 PRINT"■■■■■■■■■SELECT 1, 2 OR 3
```

```
547 GET VV$:IF VV$="" THEN GOTO 547
548 IF VV$="1" THEN PV$="FV"
549 IF VV$="2" THEN PV$="HV"
550 IF VV$="3" THEN PV$="LV"
560 GOSUB 1600
565 PRINT"VERIFICATION RUN COMPLETE"
570 PRINT"FURTHER VERIFICATION TO PROCEED?"
575 GET P$:IF P$="" THEN GOTO 575
576 IF P$="Y" THEN GOTO 542
580 PRINT:PRINTTAB(4)"SECURITY FUSES TO BE PROGRAMMED?"
590 GET P$:IF P$="" THEN GOTO 590
600 IF P$="N"THEN GOTO 660
610 PRINT:PRINT"PROGRAMMING SECURITY FUSES..."
630 SYS 33138:REM PROGRAMMING SECURITY FUSES
640 PRINT:PRINT"SECURITY FUSES PROGRAMMED"
642 POKE 57088,17
643 POKE 56840,0
644 POKE 56842,0
645 POKE 56848,0
646 POKE 56850,0
647 POKE 57090,0
660 PRINT:PRINT TAB(25)"END OF PROGRAM"
999 END
1000 REM START PREVERIFICATION
1001 A0=56840:B0=56842:A1=56849:B1=56850:A2=57088:B2=57090:FUSE=28671
1006 R$=MID$(PAL$,3,1)
1009 POKE A2,17
1010 POKE B2,8:REM MUX0 AS I/P AND MUX1 AS O/P
1012 POKE 56849,0:REM A1 TO CONTROL MODE
1013 POKE 56848,240:REM SET A1
1014 POKE 56849,04:REM A1 TO DATA MODE
1015 POKE 56848,0:REM RESET A1
1020 POKE B2,9:REM PIN1 TO VHH
1030 K=K+1
1060 FOR I=0 TO 7
1065 READ INP
1067 POKE A0,INP
1070 FOR J=0 TO 3
1080 IF J/2=INT(J/2) THEN POKE B0,0:GOTO 1100:REM SET VH TO Z
1090 POKE B0,255:REM SET VH TO VH
1100 IF J<2 THEN LR=0:GOTO 1130
1110 IF K=1 THEN LR=128:GOTO 1130
1120 LR=1
1130 POKE A1,LR:REM SET L/R
1140 IF K=2 THEN GOTO 1300
1143 D=0:REM START OF PL<32
1145 FOR C=0 TO 3
1150 FOR A=0 TO 112 STEP 16
1155 FUSE=FUSE+1
1157 D=D+1
1160 IF PEEK(FUSE)<>1 THEN GOTO 1250
1170 G=AORLR:REM ADJUST A,LR
1180 POKE A1,G
1190 POKE B1,2:REM CLK TO VH
1200 POKE B1,0:REM CLK TO Z
1210 F=PEEK(A1)
1215 IL=INT((FUSE-28672)/64):PL=FUSE-28672-IL*64
1220 PRINTIL,PL,F,
1230 F=PEEK(30847+D)ANDF:PRINTF
1240 IF (F=0)AND(R$<>"H") THEN GOSUB 1510
1241 IF (F<>0)AND(R$="H") THEN GOSUB 1510
1245 PRINT
1250 NEXT A
1260 NEXT C
1265 FUSE=FUSE+32
1270 NEXT J
1280 NEXT I
1285 FUSE=28671+32
1286 RESTORE
1288 POKE 56849,0:POKE 56848,15:POKE 56849,4:REM SET A1
1290 IF K<2 THEN POKE B2,16:POKE B1,1:GOTO 1030:REM RESET OD,SET OD AND SET MUX
1295 POKE A0,0:POKE B0,0:POKE A1,0:POKEB1,0:POKEA2,17:POKE B2,0:REM RESET PORTS
1296 RESTORE
1297 RETURN
```

```
1300 D=0:REM START OF PL>31
1305 FOR C=0 TO 3
1310 FOR A=0 TO 14 STEP 2
1320 FUSE=FUSE+1
1325 D=D+1
1330 IF PEEK(FUSE)<>1 THEN GOTO 1420
1340 G=AORLR:REM ADJUST A/LR
1350 POKE A1,G
1360 POKE B2,18:REM CLK TO VH
1370 POKE B2,16:REM CLK TO Z
1380 F=PEEK(A1)
1385 IL=INT((FUSE-28672)/64):PL=FUSE-28672-IL*64
1390 PRINT IL,PL,F,
1400 F=PEEK(30879+D)ANDF:PRINTF
1410 IF (F=0)AND(R$<>"H") THEN GOSUB 1510
1411 IF (F<>0)AND(R$="H") THEN GOSUB 1510
1415 PRINT
1420 NEXT A
1430 NEXT C
1440 GOTO 1265
1500 DATA 254,253,251,247,239,223,191,127
1510 PRINT"REJECT":REM PAL FAILS PREVERIFICATION
1520 UNP=1
1530 RETURN
1600 REM START OF VERIFICATION
1601 A0=56840:B0=56842:A1=56848:B1=56850:A2=57088:B2=57090:FUSE=28671
1603 IF PV$="HV" THEN POKE A2,18:PS=18:GOTO 1610
1604 IF PV$="LV" THEN POKE A2,20:PS=20:GOTO 1610
1605 POKE A2,17
1606 PS=17
1610 RESTORE
1614 R$=MID$(PAL$,3,1)
1615 K=0
1620 POKE B2,8:REM MUX0 AS I/P AND MUX1 AS O/P
1630 POKE 56849,0:REM A1 TO CONTROL MODE
1640 POKE 56848,240:REM SET A1
1650 POKE 56849,4:REM A1 TO DATA MODE
1660 POKE 56848,0:REM RESET A1
1670 POKE B2,9:REM PIN1 TO VHH
1680 K=K+1
1690 FOR I=0 TO 7
1700 READ INP
1710 POKE A0,INP
1720 FOR J=0 TO 3
1730 IF J/2=INT(J/2) THEN POKE B0,0:GOTO 1750:REM SET VH TO Z
1740 POKE B0,255:REM SET VH TO VH
1750 IF J<2 THEN LR=0:GOTO 1780
1760 IF K=1 THEN LR=128:GOTO 1780
1770 LR=1
1780 POKE A1,LR:REM SET L/R
1790 IF K=2 THEN GOTO 2150
1800 D=0:REM START OF PL<32
1810 FOR C=0 TO 3
1820 FOR A=0 TO 112 STEP 16
1830 FUSE=FUSE+1
1840 D=D+1
1850 IF PEEK(FUSE)<>0 THEN GOTO 1970
1860 G=AORLR:REM ADJUST A/LR
1870 POKE A1,G
1880 POKE B1,2:REM CLK TO VH
1890 POKE B1,0:REM CLK TO Z
1900 F=PEEK(A1)
1910 IL=INT((FUSE-28672)/64):PL=FUSE-28672-IL*64
1920 PRINT IL,PL,F,
1930 F=PEEK(30847+D)ANDF:PRINTF
1940 IF (F<>0)AND(R$<>"H") THEN GOTO 3100:REM GOTO REPROGRAM  ROUTINE
1950 IF (F=0)AND(R$="H") THEN GOTO 3100:REM GOTO REPROGRAM ROUTINE
1960 PRINT
1970 NEXT A
1980 NEXT C
1990 FUSE=FUSE+32
2000 NEXT J
2010 NEXT I
2020 FUSE=28703:REM START OF PL>31
```

```
2030 RESTORE
2040 POKE 56849,0:REM ADJUST A1
2050 POKE 56848,15
2060 POKE 56849,4
2070 IF K<2 THEN POKE B2,16:POKE B1,1:GOTO 1680:REM SET OD AND SET MUX'S
2080 POKE A0,0:REM RESET PORTS
2090 POKE B0,0
2100 POKE A1,0
2110 POKE B1,0
2120 POKE A2,17
2130 POKE B2,0
2135 RESTORE
2140 RETURN
2150 D=0:REM START OF PL>31
2160 FOR C=0 TO 3
2170 FOR A=0 TO 14 STEP 2
2180 FUSE=FUSE+1
2190 D=D+1
2200 IF PEEK(FUSE)<>0 THEN GOTO 2320
2210 G=A0RLR:REM ADJUST A,LR
2220 POKE A1,G
2230 POKE B2,18:REM CLK TO VH
2240 POKE B2,16:REM CLK TO Z
2250 F=PEEK(A1)
2260 IL=INT((FUSE-28672)/64):PL=FUSE-28672-IL*64
2270 PRINT IL,PL,F,
2280 F=PEEK(30879+D)ANDF:PRINTF
2290 IF (F<>0)AND(R$<>"H") THEN GOTO 3300:REM GOTO REPROGRAM ROUTINE
2300 IF (F=0)AND(R$="H") THEN GOTO 3300:REM GOTO REPROGRAM ROUTINE
2310 PRINT
2320 NEXT A
2330 NEXT C
2340 GOTO 1990
3100 IF PS<>17 THEN PRINT"APAL FAILS VERIFICATION":GOTO 3500:REM REJECT
3101 DA=PEEK(30847+D)ORG
3102 POKE 30915,DA
3106 RP=0
3110 RP=RP+1
3120 IF RP>2 THEN PRINT"AREJECT PAL! FAILS VERIFICATION.":GOTO 2140
3125 PRINT"REPROGRAM FUSE";IL;PL
3126 POKE A0,0:POKE B0,0:POKE A1,0
3127 POKE B1,4:REM CLK TO 0V
3130 POKE B2,24:REM MUX0 AND 1 AS O/P
3140 POKE 56849,0:REM A1 0-7 AS O/P
3150 POKE A1,255
3160 POKE 56849,4
3161 POKE A0,INP
3162 IF J/2=INT(J/2) THEN POKE B0,0:GOTO 3170
3163 POKE B0,255
3170 POKE A1,G:REM SET A, L/R
3180 POKE A2,16:REM VCC TO VHH
3190 SYS 33121:REM PULSE O PIN
3200 POKE A2,17:REM VCC TO 5V
3201 POKE B2,8
3210 POKE 56849,0:REM ADJUST A1
3220 POKE 56848,240
3230 POKE 56849,4
3231 POKE B1,2:REM CLK TO VH
3232 POKE B1,0:REM CLK TO VL
3233 F=PEEK(A1)
3234 F=PEEK(30847+D)ANDF
3235 IF (F<>0)AND(R$<>"H") THEN GOTO 3110
3236 IF (F=0)AND(R$="H") THEN GOTO 3110
3250 GOTO 1960:REM RESUME VERIFICATION
3300 IF PS<>17 THEN PRINT"REJECT PAL!":GOTO 3500
3301 DA=PEEK(30847+D)ORG
3302 POKE 30915,DA
3306 RR=0
3310 RR=RR+1
3320 IF RR>3 THEN PRINT"REJECT PAL!":GOTO 2140:REM REJECT PAL
```

```
3321 PRINT"REPROGRAM FUSE";IL;PL
3322 POKE A0,0:POKE B0,0:POKE A1,0
3323 POKE B2,24:REM MUX0 AND 1 AS O/P
3330 POKE B2,28:REM CLK TO 0V
3340 POKE 56849,0:REM SET A1 0-7 AS 0/P
3350 POKE 56848,255
3360 POKE 56849,4
3361 POKE A0,INP
3362 IF J/2=INT(J/2) THEN POKE B0,0:GOTO 3370
3363 POKE B0,255
3370 POKE A1,0:REM SET A, L/R
3380 POKE A2,16:REM VCC TO VHH
3390 SYS 33121:REM PULSE O PIN
3391 POKE A2,17:REM VCC TO 5V
3392 POKE 56849,0:REM ADJUST A1
3393 POKE 56848,15
3394 POKE 56849,4
3401 POKE B2,16
3402 POKE B2,18
3403 POKE B2,16
3404 F=PEEK(A1)
3405 F=PEEK(30847+D)ANDF
3406 IF (F<>0)AND(R$<>"H") THEN GOTO 3310
3407 IF (F=0)AND(R$="H") THEN GOTO 3310
3450 GOTO 2310: REN RESUME VERIFICATION
3500 POKE A1,0
3510 POKE A0,0
3520 POKE B0,0
3530 POKE B1,0
3540 POKE B2,24
3550 POKE A2,17
3560 GOTO 2140

READY.
```

APPENDIX E

Program Fuse - Machine Code Routine

```
8000 A9 00      LDA #$00
8002 8D 09 DE   STA $DE09
8005 8D 0B DE   STA $DE0B
8008 8D 11 DE   STA $DE11
800B 8D 13 DE   STA $DE13
800E 8D 01 DF   STA $DF01
8011 8D 03 DF   STA $DF03
8014 A9 FF      LDA #$FF
8016 8D 08 DE   STA $DE08
8019 8D 0A DE   STA $DE0A
801C 8D 10 DE   STA $DE10
801F 8D 12 DE   STA $DE12
8022 8D 00 DF   STA $DF00
8025 8D 02 DF   STA $DF02
8028 A9 04      LDA #$04
802A 8D 09 DE   STA $DE09
802D 8D 0B DE   STA $DE0B
8030 8D 11 DE   STA $DE11
8033 8D 13 DE   STA $DE13
8036 8D 01 DF   STA $DF01
8039 8D 03 DF   STA $DF03
803C A9 00      LDA #$00
803E 8D 08 DE   STA $DE08
8041 8D 0A DE   STA $DE0A
8044 8D 10 DE   STA $DE10
8047 8D 12 DE   STA $DE12
804A 8D 02 DF   STA $DF02
804D A9 11      LDA #$11
804F 8D 00 DF   STA $DF00
8052 60         RTS
8053 A9 00      LDA #$00
8055 8D 11 DE   STA $DE11
8058 A9 FF      LDA #$FF
805A 8D 10 DE   STA $DE10
805D A9 04      LDA #$04
805F 8D 11 DE   STA $DE11
8062 A9 00      LDA #$00
8064 8D 10 DE   STA $DE10
8067 8D 12 DE   STA $DE12
806A 8D 08 DE   STA $DE08
806D 8D 0A DE   STA $DE0A
8070 A9 11      LDA #$11
8072 8D 00 DF   STA $DF00
8075 A9 18      LDA #$18
8077 8D 02 DF   STA $DF02
807A 09 01      ORA #$01
807C 8D 02 DF   STA $DF02
807F A9 1F      LDA #$1F
8081 8D C1 78   STA $78C1
8084 A9 04      LDA #$04
8086 8D 12 DE   STA $DE12
```

```
8089 A9 00      LDA #$00
808B 8D C2 78   STA $78C2
808E A2 00      LDX #$00
8090 A9 3F      LDA #$3F
8092 8D C0 78   STA $78C0
8095 A0 00      LDY #$00
8097 CC C0 78   CPY $78C0
809A 30 05      BMI $80A1
809C A9 01      LDA #$01
809E 4C A3 80   JMP $80A3
80A1 A9 80      LDA #$80
80A3 48         PHA
80A4 CD C2 78   CMP $78C2
80A7 F0 27      BEQ $80D0
80A9 08         PHP
80AA A9 18      LDA #$18
80AC 8D 02 DF   STA $DF02
80AF A9 00      LDA #$00
80B1 8D 12 DE   STA $DE12
80B4 28         PLP
80B5 10 0F      BPL $80C6
80B7 A9 19      LDA #$19
80B9 8D 02 DF   STA $DF02
80BC A9 04      LDA #$04
80BE 8D 12 DE   STA $DE12
80C1 68         PLA
80C2 48         PHA
80C3 4C D0 80   JMP $80D0
80C6 A9 01      LDA #$01
80C8 8D 12 DE   STA $DE12
80CB A9 1C      LDA #$1C
80CD 8D 02 DF   STA $DF02
80D0 8A         TXA
80D1 4A         LSR
80D2 4A         LSR
80D3 B0 06      BCS $80DB
80D5 68         PLA
80D6 A9 00      LDA #$00
80D8 4C DE 80   JMP $80DE
80DB 68         PLA
80DC EA         NOP
80DD EA         NOP
80DE EA         NOP
80DF 18         CLC
80E0 79 40 78   ADC $7840,Y
80E3 48         PHA
80E4 EA         NOP
80E5 8D 10 DE   STA $DE10
80E8 48         PHA
80E9 BD 00 78   LDA $7800,X
80EC 8D 08 DE   STA $DE08
80EF BD 20 78   LDA $7820,X
80F2 8D 0A DE   STA $DE0A
80F5 EA         NOP
80F6 B9 00 70   LDA $7000,Y
80F9 48         PHA
80FA 38         SEC
80FB E9 02      SBC #$02
80FD 10 23      BPL $8122
```

```
80FF 68          PLA
8100 4A          LSR
8101 B0 20       BCS $8123
8103 A9 10       LDA #$10
8105 8D 00 DF    STA $DF00
8108 68          PLA
8109 19 80 78    ORA $7880,Y
810C 8D 10 DE    STA $DE10
810F 99 00 60    STA $6000,Y
8112 EA          NOP
8113 EA          NOP
8114 EA          NOP
8115 EA          NOP
8116 68          PLA
8117 8D 10 DE    STA $DE10
811A A9 11       LDA #$11
811C 8D 00 DF    STA $DF00
811F 4C 25 81    JMP $8125
8122 68          PLA
8123 68          PLA
8124 68          PLA
8125 A9 00       LDA #$00
8127 8D 10 DE    STA $DE10
812A 8D 0A DE    STA $DE0A
812D 8D 08 DE    STA $DE08
8130 C8          INY
8131 CE C0 78    DEC $78C0
8134 10 28       BPL $815E
8136 E8          INX
8137 CE C1 78    DEC $78C1
813A 10 0B       BPL $8147
813C A9 18       LDA #$18
813E 8D 02 DF    STA $DF02
8141 A9 00       LDA #$00
8143 8D 12 DE    STA $DE12
8146 60          RTS
8147 A9 40       LDA #$40
8149 18          CLC
814A 6D F7 80    ADC $80F7
814D 8D F7 80    STA $80F7
8150 8D 10 81    STA $8110
8153 90 06       BCC $815B
8155 EE F8 80    INC $80F8
8158 EE 11 81    INC $8111
815B 4C 90 80    JMP $8090
815E 4C 97 80    JMP $8097
8161 AD C3 78    LDA $78C3
8164 8D 10 DE    STA $DE10
8167 EA          NOP
8168 EA          NOP
8169 EA          NOP
816A EA          NOP
816B EA          NOP
816C A9 00       LDA #$00
816E 8D 10 DE    STA $DE10
8171 60          RTS
8172 A9 08       LDA #$08
8174 8D 00 DF    STA $DF00
8177 A2 04       LDX #$04
```

```
8179 A9 01      LDA #$01
817B 8D 02 DF   STA $DF02
817E A0 04      LDY #$04
8180 88         DEY
8181 D0 FD      BNE $8180
8183 A9 00      LDA #$00
8185 8D 02 DF   STA $DF02
8188 CA         DEX
8189 D0 EE      BNE $8179
818B A2 04      LDX #$04
818D A9 01      LDA #$01
818F 8D 12 DE   STA $DE12
8192 A0 04      LDY #$04
8194 88         DEY
8195 D0 FD      BNE $8194
8197 A9 00      LDA #$00
8199 8D 12 DE   STA $DE12
819C CA         DEX
819D D0 EE      BNE $818D
819F A9 11      LDA #$11
81A1 8D 00 DF   STA $DF00
81A4 60         RTS
```

DESCRIPTION	QUAN.	COMPONENT NAME(S)
10k	30	R109,R111,R114,R118,R12,R122,R125,R16,R20 R24,R28,R32,R36,R4,R40,R44,R48,R52,R56,R60 R64,R68,R72,R76,R8,R80,R84,R88,R92,R96
15k	64	R10,R101,R102,R105,R106,R11,R113,R116,R117 R120,R121,R124,R14,R15,R152,R153,R154,R155 R156,R18,R19,R2,R22,R23,R26,R27,R3,R30,R31 R34,R35,R38,R39,R42,R43,R46,R47,R50,R51 R54,R55,R58,R59,R6,R62,R63,R66,R67,R7,R70 R71,R74,R75,R78,R79,R82,R83,R86,R87,R90 R91,R94,R95,R99
1k2	10	R103,R107,R157,R158,R159,R160,R161,R162 R163,R164
1N4004	2	D19,D20
1N4148	18	D1,D10,D11,D12,D13,D14,D15,D16,D17,D18,D2 D3,D4,D5,D6,D7,D8,D9
240E	3	R148,R150,R151
2k2	5	VR1,VR2,VR3,VR4,VR8
2N2222	32	Q1,Q11,Q13,Q15,Q17,Q19,Q21,Q23,Q25,Q27,Q29 Q3,Q31,Q33,Q35,Q37,Q39,Q41,Q43,Q45,Q47,Q49 Q5,Q50,Q51,Q52,Q53,Q55,Q57,Q59,Q7,Q9
2N4403	26	Q10,Q12,Q14,Q16,Q18,Q2,Q20,Q22,Q24,Q26,Q28 Q30,Q32,Q34,Q36,Q38,Q4,Q40,Q42,Q44,Q46,Q48 Q54,Q58,Q6,Q8
33k	32	R1,R100,R104,R112,R115,R119,R123,R13,R17 R21,R25,R29,R33,R37,R41,R45,R49,R5,R53,R57 R61,R65,R69,R73,R77,R81,R85,R89,R9,R93,R97 R98
4k7	26	R126,R127,R128,R129,R130,R131,R132,R133 R134,R135,R136,R137,R138,R139,R140,R141 R142,R143,R144,R145,R146,R147,R149,VR5,VR6 VR7
6800/25V	1	C1
BC177	2	Q56,Q60
CA3081	1	TA1
CD4050	2	U5,U6
CD4053	4	U1,U2,U3,U4
MC6821	3	PIA0,PIA1,PIA2

COMP No.	LABEL	PACKAGE	COMMENT
1	C1	RB.4/.8	6800/25V
2	D1	AXIAL0.4	1N4148
3	D10	AXIAL0.4	1N4148
4	D11	AXIAL0.4	1N4148
5	D12	AXIAL0.4	1N4148
6	D13	AXIAL0.4	1N4148
7	D14	AXIAL0.4	1N4148
8	D15	AXIAL0.4	1N4148
9	D16	AXIAL0.4	1N4148
10	D17	AXIAL0.4	1N4148
11	D18	AXIAL0.4	1N4148
12	D19	AXIAL0.4	1N4004
13	D2	AXIAL0.4	1N4148
14	D20	AXIAL0.4	1N4004
15	D3	AXIAL0.4	1N4148
16	D4	AXIAL0.4	1N4148
17	D5	AXIAL0.4	1N4148
18	D6	AXIAL0.4	1N4148
19	D7	AXIAL0.4	1N4148
20	D8	AXIAL0.4	1N4148
21	D9	AXIAL0.4	1N4148
22	PIA0	DIP40	MC6821
23	PIA1	DIP40	MC6821
24	PIA2	DIP40	MC6821
25	Q1	TRANS	2N2222
26	Q10	TRANS	2N4403
27	Q11	TRANS	2N2222
28	Q12	TRANS	2N4403
29	Q13	TRANS	2N2222
30	Q14	TRANS	2N4403
31	Q15	TRANS	2N2222
32	Q16	TRANS	2N4403
33	Q17	TRANS	2N2222
34	Q18	TRANS	2N4403
35	Q19	TRANS	2N2222
36	Q2	TRANS	2N4403
37	Q20	TRANS	2N4403
38	Q21	TRANS	2N2222
39	Q22	TRANS	2N4403
40	Q23	TRANS	2N2222
41	Q24	TRANS	2N4403
42	Q25	TRANS	2N2222
43	Q26	TRANS	2N4403
44	Q27	TRANS	2N2222
45	Q28	TRANS	2N4403
46	Q29	TRANS	2N2222
47	Q3	TRANS	2N2222
48	Q30	TRANS	2N4403
49	Q31	TRANS	2N2222
50	Q32	TRANS	2N4403

COMP No.	LABEL	PACKAGE	COMMENT:
51	Q33	TRANS	2N2222
52	Q34	TRANS	2N4403
53	Q35	TRANS	2N2222
54	Q36	TRANS	2N4403
55	Q37	TRANS	2N2222
56	Q38	TRANS	2N4403
57	Q39	TRANS	2N2222
58	Q4	TRANS	2N4403
59	Q40	TRANS	2N4403
60	Q41	TRANS	2N2222
61	Q42	TRANS	2N4403
62	Q43	TRANS	2N2222
63	Q44	TRANS	2N4403
64	Q45	TRANS	2N2222
65	Q46	TRANS	2N4403
66	Q47	TRANS	2N2222
67	Q48	TRANS	2N4403
68	Q49	TRANS	2N2222
69	Q5	TRANS	2N2222
70	Q50	TRANS	2N2222
71	Q51	TRANS	2N2222
72	Q52	TRANS	2N2222
73	Q53	TRANS	2N2222
74	Q54	TRANS	2N4403
75	Q55	TRANS	2N2222
76	Q56	TRANS	BC177
77	Q57	TRANS	2N2222
78	Q58	TRANS	2N4403
79	Q59	TRANS	2N2222
80	Q6	TRANS	2N4403
81	Q60	TRANS	BC177
82	Q7	TRANS	2N2222
83	Q8	TRANS	2N4403
84	Q9	TRANS	2N2222
85	R1	AXIAL0.4	33k
86	R10	AXIAL0.4	15k
87	R100	AXIAL0.4	33k
88	R101	AXIAL0.4	15k
89	R102	AXIAL0.4	15k
90	R103	AXIAL0.4	1k2
91	R104	AXIAL0.4	33k
92	R105	AXIAL0.4	15k
93	R106	AXIAL0.4	15k
94	R107	AXIAL0.4	1k2
95	R109	AXIAL0.4	10k
96	R11	AXIAL0.4	15k
97	R111	AXIAL0.4	10k
98	R112	AXIAL0.4	33k
99	R113	AXIAL0.4	15k
100	R114	AXIAL0.4	10k

COMP No.	LABEL	PACKAGE	COMMENT
101	R115	AXIAL0.4	33k
102	R116	AXIAL0.4	15k
103	R117	AXIAL0.4	15k
104	R118	AXIAL0.4	10k
105	R119	AXIAL0.4	33k
106	R12	AXIAL0.4	10k
107	R120	AXIAL0.4	15k
108	R121	AXIAL0.4	15k
109	R122	AXIAL0.4	10k
110	R123	AXIAL0.4	33k
111	R124	AXIAL0.4	15k
112	R125	AXIAL0.4	10k
113	R126	AXIAL0.4	4k7
114	R127	AXIAL0.4	4k7
115	R128	AXIAL0.4	4k7
116	R129	AXIAL0.4	4k7
117	R13	AXIAL0.4	33k
118	R130	AXIAL0.4	4k7
119	R131	AXIAL0.4	4k7
120	R132	AXIAL0.4	4k7
121	R133	AXIAL0.4	4k7
122	R134	AXIAL0.4	4k7
123	R135	AXIAL0.4	4k7
124	R136	AXIAL0.4	4k7
125	R137	AXIAL0.4	4k7
126	R138	AXIAL0.4	4k7
127	R139	AXIAL0.4	4k7
128	R14	AXIAL0.4	15k
129	R140	AXIAL0.4	4k7
130	R141	AXIAL0.4	4k7
131	R142	AXIAL0.4	4k7
132	R143	AXIAL0.4	4k7
133	R144	AXIAL0.4	4k7
134	R145	AXIAL0.4	4k7
135	R146	AXIAL0.4	4k7
136	R147	AXIAL0.4	4k7
137	R148	AXIAL0.4	240E
138	R149	AXIAL0.4	4k7
139	R15	AXIAL0.4	15k
140	R150	AXIAL0.4	240E
141	R151	AXIAL0.4	240E
142	R152	AXIAL0.4	15k
143	R153	AXIAL0.4	15k
144	R154	AXIAL0.4	15k
145	R155	AXIAL0.4	15k
146	R156	AXIAL0.4	15k
147	R157	AXIAL0.4	1k2
148	R158	AXIAL0.4	1K2
149	R159	AXIAL0.4	1k2
150	R16	AXIAL0.4	10k

COMP No.	LABEL	PACKAGE	COMMENT
151	R160	AXIAL0.4	1K2
152	R161	AXIAL0.4	1k2
153	R162	AXIAL0.4	1k2
154	R163	AXIAL0.4	1k2
155	R164	AXIAL0.4	1k2
156	R17	AXIAL0.4	33k
157	R18	AXIAL0.4	15k
158	R19	AXIAL0.4	15k
159	R2	AXIAL0.4	15k
160	R20	AXIAL0.4	10k
161	R21	AXIAL0.4	33k
162	R22	AXIAL0.4	15k
163	R23	AXIAL0.4	15k
164	R24	AXIAL0.4	10k
165	R25	AXIAL0.4	33k
166	R26	AXIAL0.4	15k
167	R27	AXIAL0.4	15k
168	R28	AXIAL0.4	10k
169	R29	AXIAL0.4	33k
170	R3	AXIAL0.4	15k
171	R30	AXIAL0.4	15k
172	R31	AXIAL0.4	15k
173	R32	AXIAL0.4	10k
174	R33	AXIAL0.4	33k
175	R34	AXIAL0.4	15k
176	R35	AXIAL0.4	15k
177	R36	AXIAL0.4	10k
178	R37	AXIAL0.4	33k
179	R38	AXIAL0.4	15k
180	R39	AXIAL0.4	15k
181	R4	AXIAL0.4	10k
182	R40	AXIAL0.4	10k
183	R41	AXIAL0.4	33k
184	R42	AXIAL0.4	15k
185	R43	AXIAL0.4	15k
186	R44	AXIAL0.4	10k
187	R45	AXIAL0.4	33k
188	R46	AXIAL0.4	15k
189	R47	AXIAL0.4	15k
190	R48	AXIAL0.4	10k
191	R49	AXIAL0.4	33k
192	R5	AXIAL0.4	33k
193	R50	AXIAL0.4	15k
194	R51	AXIAL0.4	15k
195	R52	AXIAL0.4	10k
196	R53	AXIAL0.4	33k
197	R54	AXIAL0.4	15k
198	R55	AXIAL0.4	15k
199	R56	AXIAL0.4	10k
200	R57	AXIAL0.4	33k

COMP No.	LABEL	PACKAGE	COMMENT
201	R58	AXIAL0.4	15k
202	R59	AXIAL0.4	15k
203	R6	AXIAL0.4	15k
204	R60	AXIAL0.4	10k
205	R61	AXIAL0.4	33k
206	R62	AXIAL0.4	15k
207	R63	AXIAL0.4	15k
208	R64	AXIAL0.4	10k
209	R65	AXIAL0.4	33k
210	R66	AXIAL0.4	15k
211	R67	AXIAL0.4	15k
212	R68	AXIAL0.4	10k
213	R69	AXIAL0.4	33k
214	R7	AXIAL0.4	15k
215	R70	AXIAL0.4	15k
216	R71	AXIAL0.4	15k
217	R72	AXIAL0.4	10k
218	R73	AXIAL0.4	33k
219	R74	AXIAL0.4	15k
220	R75	AXIAL0.4	15k
221	R76	AXIAL0.4	10k
222	R77	AXIAL0.4	33k
223	R78	AXIAL0.4	15k
224	R79	AXIAL0.4	15k
225	R8	AXIAL0.4	10k
226	R80	AXIAL0.4	10k
227	R81	AXIAL0.4	33k
228	R82	AXIAL0.4	15k
229	R83	AXIAL0.4	15k
230	R84	AXIAL0.4	10k
231	R85	AXIAL0.4	33k
232	R86	AXIAL0.4	15k
233	R87	AXIAL0.4	15k
234	R88	AXIAL0.4	10k
235	R89	AXIAL0.4	33k
236	R9	AXIAL0.4	33k
237	R90	AXIAL0.4	15k
238	R91	AXIAL0.4	15k
239	R92	AXIAL0.4	10k
240	R93	AXIAL0.4	33k
241	R94	AXIAL0.4	15k
242	R95	AXIAL0.4	15k
243	R96	AXIAL0.4	10k
244	R97	AXIAL0.4	33k
245	R98	AXIAL0.4	33k
246	R99	AXIAL0.4	15k
247	TA1	DIP16	CA3081
248	U1	DIP16	CD4053
249	U2	DIP16	CD4053
250	U3	DIP16	CD4053

COMP No.	LABEL	PACKAGE	COMMENT
251	U4	DIP16	CD4053
252	U5	DIP16	CD4050
253	U6	DIP16	CD4050
254	VR1	VR4	2k2
255	VR2	VR4	2k2
256	VR3	VR4	2k2
257	VR4	VR4	2k2
258	VR5	VR4	4k7
259	VR6	VR4	4k7
260	VR7	VR4	4k7
261	VR8	VR4	2k2

REFERENCES

Aron, J.D. 1974: *The Program Development Process*, Part 1. Reading, Mass., Addison-Wesley Publishing Company.

Baker, S. A New Programmable Logic Era Is Dawning In: *Dataweek*, Vol. 10, No. 8: 30. April 24, 1987.

Barwise, M. Hardware Design Concepts In: *Electronics Today International*, Vol. 16, No. 1: 22-24. February, 1987.

Bayley, B. 1984: *Commodore 64 Exposed*. Nashville, Melbourne House (Publishers) Ltd.

Birkner, J.M. and Coli, V.J. 1983: *Programmable Array Logic Handbook*, 3rd Ed. Santa Clara, Monolithic Memories Inc.

Brinkman, R. 1984: *Programming in Structured Basic*. New York, Macmillan Publishing Company.

Brafman, B. 1983: Programmable Array Logic Leads to Flexible Application of 8-bit Wide Memories In: *Programmable Array Logic Handbook* by Birkner, J.M. and Coli, V.J.: 8.40-8.41.

Chakravarty, D. and Gottlieb, E. Programmable Logic Devices In: *Pulse*: 9-11. February, 1986.

Comer, D.J. 1984: *Digital Logic and State Machine Design.* Tokyo, CBS College Publishing.

Commodore 64 Programmer's Reference Guide, 1982. Commodore Business Machines Inc. and Howard Sams and Co., Inc.

Dvorak, S. and Musset, A. 1984: *Basic in Action.* London, Butterworths.

Hall, D.V. 1983: Microprocessors and Digital Logic, 2nd. Ed. Singapore, McGraw-Hill International Book Co.

Herrington, D.E., Nichols, P.A. and Lipp, R.D. Software Verification Using Branch Analysis In: *Hewlett Packard Journal*, Vol. 38, No. 6: 13-22. June, 1987.

IC Master, Vol. 1 1985: Edited by Howell, D. Garden City, N.Y., Hearst Business Communications, Inc.

IC Master, Vol. 2 1985: Edited by Howell, D. Garden City, N.Y., Hearst Business Communications, Inc.

Intelligent Help For Circuit Designers Overcomes PLD Chaos In: *Dataweek*, Vol. 11, No. 9: 11. May 6, 1988.

Jay, C. Programmable Logic Design In: *Electronics and Wireless World*, Vol. 92, No. 1609: 65-69. November, 1986.

Jones, D.J. 1986: *Principles and Applications of Digital Electronics.* New York, Macmillan Publishing Company.

Katzin, E. 1985: *How to Write a Really Good User's Manual.* New York, Van Nostrand Reinhold Co., Inc.

Klingman, E.E. 1982: *Microprocessor Systems Design, Vol. 2.* Englewoord Cliffs, N.J., Prentice all, Inc.

Koffman, E.B. and Friedman, F.L. 1979: *Problem Solving and Structured Programming in Basic.* Reading, Mass., Addison-Wesley Publishing Company. Inc.

Levy, A. Programmable Logic - The Silent Revolution In: *Pulse*: 31-33. April, 1986.

LSI Databook, 6th Ed. 1985: Santa Clara, Monolithic Memories, Inc.

Meyer, E. Programmable Logic Devices, Part 1 In: *Radio-Electronics*, Vol. 59, No. 2: 59-64. February, 1988.

Meyer, E. Programmable Logic Devices, Part 2 In: *Radio-Electronics*, Vol. 59, No. 3: 63-67. March, 1988.

Programmable Array Logic In: *Electronics and Wireless World*, Vol. 92, No. 1609: 65-69. November, 1986.

Programmable Logic Databook, 1983: Santa Clara, National Semiconductor Corporation.

Putman, C. The State-of-the-Art in Semi-Custom In: *Dataweek*, Vol. 10, No. 8. April 24, 1987.

Sommerville, I. 1982: *Software Engineering*. London, Addison-Wesley Publishing Company.

Tocci, R.J. 1980: *Digital Systems*. London, Prentice-Hall International, Inc.

Triebel, W.A. and Chu, A.E. 1982: *Handbook of Semiconductor and Bubble Memories*. Englewood Cliffs, N.J., Prentice-Hall International, Inc.

Van Tassel, D. 1978: *Program Style, Design, Efficiency, Debugging and Testing*, 2nd Ed. Englewood Cliffs, N.J., Prentice-Hall, Inc.

Vafia, M. 1983: Testing Your PAL Devices In: *Programmable Logic Databook*, 3rd. Ed., by Birkner, J.M. and Coli, V.J.: 8.79-8.83.