SLAC – PUB – 3454
September 1984
(I)

UNIVERSAL FILE PROCESSING PROGRAM FOR FIELD PROGRAMMABLE INTEGRATED CIRCUITS*

DIETRICH R. FREYTAG
DAVID J. NELSON
Stanford Linear Accelerator Center
Stanford University
Stanford, California 94305

Abstract

A computer program is presented that translates logic equations into promburner files (or the reverse) for programmable logic devices of various kinds, namely PROMs FPLAs, FPLSs and PALs. The program achieves flexibility through the use of a database containing detailed information about the devices to be programmed. New devices can thus be accommodated through simple extensions of the database. When writing logic equations, the user can define logic combinations of signals as new logic variables for use in subsequent equations. This procedure yields compact and transparent expressions for logic operations, thus reducing the chances for error. A logic simulation program is also provided so that an independent check of the design can be performed at the software level.

1. Introduction

Programmable logic devices (PLDs) in the form of PROMs FPLAs, FPLSs and PALs are very useful and rich building blocks for electronic logical systems. The range and performance of available devices is increasing every year and so is the complexity of the problems that can be solved with programmable logic.

A powerful method of describing a logic system is by means of logic equations using true/false signals and the logical operations of 'and' and 'or'. Thus a software package which translates logic equations directly into a list of fuses to be burned is an immensely useful tool in the development of logic systems up to the complexity of a modern micro- or minicomputer.

Manufacturers of programmable logic offer software packages which perform this translation, *e.g.*, the PALASM program from Monolithic Memories for PALs[1] or a similar program available from Signetics for FPLAs. Since the various devices on the market have different areas of strength, the designer may decide in the course of development to transfer certain logic functions from one device type to another. It is therefore advantageous to have a program that addresses all PLDs in an identical manner. With the rapidly expanding line of programmable logic, it is imperative to have a program structure that can easily accommodate new devices.

It is desirable to have the capability of translating the fuse pattern back into logic equations in order to keep track of the contents of programmed devices. This operation is also very helpful for the understanding of undocumented devices.

Finally, as an independent check of the functionality of the design, it is useful to be able at the software level to step through logic sequences using test vectors.

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.

2. Description of the Program

2.1 SOFTWARE STRUCTURE

The program described here (named 'PROMise' at SLAC) consists of an interactive section for input of information at execution time and the processing section, which is written in FORTRAN. This latter section uses a very basic instruction set for maximum compatibility with different computers. The FORTRAN program compiles without changes on an IBM or a Digital computer.

The interactive section is written in REX executive language for the IBM machine and in Digital's interactive FORTRAN for the VAX computer.

2.2 FILE TYPES

The program performs translations between several equivalent descriptions of the device at various hierarchical levels of language. The highest level is that of logic equations. An intermediate level, which is often useful for checking purposes or which may be used as a source file, is a representation in the form of a fuse table. The description needed for burning the device is a binary list of fuses to be blown. The types of descriptions and the transformations between them may be schematically represented as follows:

$$(\text{Logic}) \leftrightarrow (\text{Fuse}) \leftrightarrow (\text{Hex}) \leftrightarrow (\text{Intel}) \leftrightarrow (\text{Prom Burner})$$

For convenience, a transformation within the binary (Hex or Intel) files is included in the package. This consists of the breakup of a string of specified length (*e.g.*, a 48-bit wide instruction word) into bytes (8 bits) or halfbytes (4 bits) for purposes of partitioning the instruction into several PROMs.

In order to minimize the amount of information that has to be provided at the time of execution of the program, each design file contains directions in a header specifying what operations are to be performed. The header block contains the device name, the source and destination file types, and (for binary files) whether a string of bits is to be partitioned into bytes or halfbytes. The header also contains one line of descriptive text, which will be repeated in the destination file and in the listing for purposes of identification. The device name serves as a key to all the necessary information about the device, which is contained in a master data file read by the program. This database is the tool for expanding the program to new devices. As long as the new device performs basically the same functions as the ones already included, a new entry in the table suffices. However, when a device has new features, the program itself must be extended to accommodate the new functions.

Presented at the Nuclear Science Symposium, Orlando, Florida, October 31 – November 2, 1984

2.3 Logic Equations

Logic equations are written in standard form combining signal names or their complement by logical 'AND' and 'OR' operations. The equations are written in terms of true or false for the signals regardless of the type of logic used (positive or negative logic). False is indicated by a '/' preceding the signal name. The type of logic is defined for each signal independently in the list assigning a name to each device pin. A signal which is negative true carries a leading '/'. In this way positive or negative true signals may be mixed, or the definitions changed during the design phase, without any effect on the form of the logic equations, thus minimizing the possibility for error. Examples for production files of logic equations including substitutions (2.4) and the resulting fuse table (2.5) are shown in Table 1.

When writing logic equations for FPLAs or FPLSs, the program automatically takes care of the merging of expressions with identical input conditions into one term driving multiple outputs. This reduction process, which maximizes the amount of logic that can be packed into a device, can be quite difficult if done without the help of a computer. Another special feature of the program has to do with the possibility of editing PLDs by burning out an entire fuse line and replacing it with a new one. A special code ('VCC=') has been defined for logic equations, generating an inactive fuse line, thus enabling the user to represent the PLD through all iterations of editing.

2.4 Substitutions

In order to simplify the writing of complicated expressions, certain logical combinations of signals may be defined and given a new name for use in subsequent equations. A particular form of such a building block helpful in all kinds of counting and sequencing applications is a function defining a group of signals as a binary number. This group of bits may subsequently be called by the function name specifying a value (decimal or hex), thus eliminating the need to assign true or false to the individual signals. It is also possible to generate "don't care" bits using the function definition. This is accomplished by specifying as argument of the function the 'OR' of all values resulting from the given bits being unspecified.

When programming PROMs using logic equations, the binary function definition described above can be particularly useful. Signal names not appearing in a logic equation for a PROM ("don't care" bits) cause all addresses allowed by the undefined bits to be generated and identical output terms burned at these addresses.

2.5 Fuse Tables

Fuse tables have been retained as an integral part of the programming package because they are still widely used as a means of defining programmable devices, and are also quite useful for checking a design because they are made to resemble the electronic structure of the device. The program listing produces a fuse table annotated with comments derived from the logic equations. When a PROM is used as a logic device, a fuse table resembling an FPLA is printed. Here a "don't care" bit means that both addresses (with the bit in question reading 0 or 1) are programmed to give identical output data.

2.6 Binary Files

Two different types of binary files are supported by the program. One type is the Intel format carrying address information and checksum for transmission to the PROM burner. The other type is a straight hex format which in some applications may serve as a source file.

2.7 Reverse Translations

Normally a device is defined in a high language (*e.g.*, logic equations) and the program provides the conversion into a binary file for burning. Occasionally the reverse translation is useful in order to determine the contents of a programmed device. For older designs generated in fuse form, it may be advantageous to generate a new definition in logic form in order to make the function of the device more transparent. The logic equations generated by the program refer to pin numbers which then should be replaced by signal names (using a text editor) to make the equations easily readable.

2.8 Documentation

Each of the file types described above defines a device completely, shown by the fact that conversion can proceed from logic to binary and back to logic forms. It is thus a matter of convenience which file type is kept for documentation. Since logic equations are most easily understood and can be richly annotated, they constitute the preferred form for keeping records.

3. Controlling the Prom Burner

The programming package also performs all the routine control functions for the prom burner. This involves a three-way communication between terminal, host computer and prom burner, which is handled by a microprocessor (Intel 8085) located in a self-contained box near the prom burner. Data received at any of the three serial ports of the interface are modified if necessary and routed to the currently selected receiving device. The program stored in the microprocessor recognizes flags in the incoming data stream and sets up one of the three possible two-way communications, *i.e.*,

$$\begin{aligned} \text{(Terminal)} &\leftrightarrow \text{(Host)} \\ \text{(Host)} &\leftrightarrow \text{(Prom Burner)} \\ \text{(Terminal)} &\leftrightarrow \text{(Prom Burner)} \end{aligned}$$

Filtering the data through the microprocessor is a convenient way to match the host computer to the prom burner. A further advantage of this approach is that it provides a standardized procedure when more than one host computer and prom burner are utilized. (At SLAC an IBM 3081 or a VAX may serve as a host while different models of DATA I/O prom burners are used in the various laboratories). By modifying the resident program in the microprocessor, other host computers or prom burners can be accommodated.

Table 1. Sample Input and Output Files

```
;-------------------------------------------------
;   Header for 74S288 PROM
;   RAMWRITE is the signal name at pin 1, high true.
;   RAMREAD  is the signal name at pin 2, low  true.
;-------------------------------------------------
74S288    LOGIC TO INTEL
TEST FOR CODING OF 32 X 8 PROM
RAMWRITE /RAMREAD DATAOUT /ENABLEY O5 O6 O7 GND
O8 A0 A1 A2 A3 A4 /E1 VCC  ;  End pin list.
;-------------------------------------------------
;   Logic equations input to program.
;-------------------------------------------------
ADDR(A4,A3,A2,A1,A0)        ;  Address function
DATAIN = A0*A1              ;  Substitution
RAMWRITE=  ADDR(AH+BH)      ;  AH = decimal 10
RAMREAD=   ADDR(8+9)        ;  A0 is "don't care"
DATAOUT=   DATAIN*A3+/A2
ENABLEY=   ADDR(1)+ADDR(CH+DH)+ADDR(EH)+ADDR(11H)
;-------------------------------------------------
;   Fuse table output generated by program.
;-------------------------------------------------
;         HHHHLHLH
;   ADDRS ( DATA )
;   43210 76543210
00  LHLH- .......A RAMWRITE=          ADDR(AH+BH)
01  LHLL- ......A. RAMREAD=           ADDR(8+9)
02  -H-HH .....A.. DATAOUT=           DATAIN*A3+
03  --L-- .....A..                    /A2
04  LLLLH ....A... ENABLEY=           ADDR(1)+
05  LHHL- ....A...                    ADDR(CH+DH)+
06  LHHHL ....A...                    ADDR(EH)+
07  HLLLH ....A...                    ADDR(11H)
;-------------------------------------------------
;   Header for Signetics FPLA 82S100
;   Setup to generate the same functions as above.
;-------------------------------------------------
FPLA100   LOGIC TO INTEL
TEST FOR CODING OF SIGNETICS FPLA 82S100
X A0 A1 A2 A3 A4 X X X RAMWRITE /RAMREAD DATAOUT GND
/ENABLEY X X X X X X X X X X X X VCC
;-------------------------------------------------
;   Logic equations follow as before.   The program
;   generates fuse table and binary file for 82S100.
;-------------------------------------------------
```

```
;-------------------------------------------------
;   Header for Signetics 82S105 FPLS
;-------------------------------------------------
FPLS105   LOGIC TO INTEL
TEST FOR 82S105 FPLS
CLOCK             ; Signal name at pin 1
I7 I6 QM2 QM1 IM2 IM1 MPC START F7 F6 F5 F4 GND
F3 F2 F1 F0 PRESET RESP BSFIN EMS I12 LAST CAMQ CAMI X VCC
CON P0 P1 P2 P3 P4 P5 N0 N1 N2 N3 N4 N5 ; Internal signals
NE(N5,N4,N3,N2,N1,N0)               ; Function definition
FF(F5,F4,F3,F2,F1,F0)               ; Function definition
PP(P5,P4,P3,P2,P1,P0)               ; Function definition
ILLL=/EMS*/I12*/LAST                ; Substitution
NE(0)=FF(0)= F6=/F7= PP(63)*START    ; First equation
F5= F6= N5=/P5
NE(24)=FF(24)= F6= P5* BSFIN* LAST*CON
NE(24)=FF(24)= F6= P5*/I12* LAST*CON
NE(30)=FF(30)= F6=/F7=PP(60)* BSFIN*/MPC
CON= P5* P4* P3
FUNCTION TABLE
BSFIN EMS I12 LAST MPC START N5 N4 N3 N1 N0 F7 F6 F5 F0
-----------------------------------------------------
;   -   -   -   -   -     H  L  L  L  L  L  L  H  L  L
;   H   -   -   L   -     -  H  L  L  L  H  L  H  L  H
;-------------------------------------------------
;   Fuse table output generated by program.
;   (The annotations to the fuse table are truncated)
;-------------------------------------------------
OE=H,PR=L   (82S104/5)                          L
;   C 111111 (INPUTS)   PREV.  NEXT  (OUTPUT)
;   D 54321098765432l0 543210 543210 76543210
00  - ---------------H HHHHHH LLLLLL LHLLLLLL NE(0)=  PP(6
01  - ---------------- L------ H------ -HH----- F5=F6=  /P5
02  . -H--H----------- H------ LHHLLL -HLHHLLL NE(24)= P5*
03  . ---LH----------- H------ LHHLLL -HLHHLLL          P5*
04  - -H------------L- HHHHLL LHHHHL LHLHHHHL NE(30)= PP(6
05  A ---------------- HHH--- ------ -------- COM=    P5*P
SIMULATION
TEST FOR 82S105 FPLS
TEST VECTOR   1 TERMS   0
ERROR IN VECTOR  2  EXPECT = H  TABLE = L PIN = F5
ERROR IN VECTOR  2  EXPECT = L  TABLE = H PIN = F0
ERROR IN VECTOR  2  EXPECT = L  TABLE = H PIN = N0
TEST VECTOR   2 TERMS   1
```

Control of the prom burner proceeds through the interactive part of the program. The binary file created in the first pass as an output file now serves as the input file to be loaded into the prom burner. The information contained in the file header greatly simplifies the operation of burning, because all the necessary information about pin-out, chip size *etc.*, is automatically picked up from the database. The interactive program allows these default values to be modified by the operator as needed in special applications.

4. List of Devices

The following is a list of devices as currently handled by the program. The device name 'FILE' is used for downloading a file containing multiple PLDs for sequential burning of several devices.

File 2708 256X4 256X4ECL 512X8 2532 2716 2732 2564
18S22 28L22 63S441 82S137 18S030 74S288 6309

2764 27S18 74188TI 7602 5330 74S188NA 10139

82S100 82S105 82S147 82S153 82S157 82S159

PAL10H8	PAL12H6	PAL14H4	PAL16H2	PAL10L8
PAL12L6	PAL12L10	PAL14L4	PAL16L2	PAL16A4
PAL16X4	PAL16R4	PAL16R6	PAL16R8	PAL16L8
PAL16C1	PAL18L4	PAL20C1	PAL20L10	PAL20X4
PAL20X8	PAL20X10			

Acknowledgements

The original version of the microprocessor program handling the communication between terminal, host computer and the prom burner was designed by John P. Steffani of SLAC. The treatment of logic equations and the design check using test vectors was adapted from MMI's PALASM programs.[1]

Reference

High Level Language for Programmable Array Logic

John M. Birkner
Programmable Logic Planning
Monolithic Memories
1165 East Arques Avenue
Sunnyvale,California 94086