NAME

espresso -- input file format for espresso(1)

DESCRIPTION

Espresso accepts as input a two-level description of a Boolean switching function. This is described as a character matrix with keywords imbedded in the input to specify the size of the matrix and the logical format of the input function. Comments are allowed within the input by placing a pound sign (#) as the first character on a line. Comments and unrecognized keywords are passed directly from the input file to standard output. Any white-space (blanks, tabs, etc.), except when used as a delimiter in an imbedded command, is ignored. It is generally assumed that the PLA is specified such that each row of the PLA fits on a single line in the input file.

KEYWORDS

The following keywords are recognized by *espresso*. The list shows the probable order of the keywords in a PLA description. [d] denotes a decimal number and [s] denotes a text string.

.i [d] Specifies the number of input variables.

.o [d] Specifies the number of output functions.

.type [s] Sets the logical interpretation of the character matrix as described below under "Logical Description of a PLA". This keyword must come before any product terms. [s] is one of f, r, fd, fr, dr, or fdr.

.phase [s] [s] is a string of as many 0's or 1's as there are output functions. It specifies which polarity of each output function should be used for the minimization (a 1 specifies that the ON-set of the corresponding output function should be used, and a 0 specifies that the OFF-set of the corresponding output function should be minimized).

.pair [d] Specifies the number of pairs of variables which will be paired together using two-bit decoders. The rest of the line contains pairs of numbers which specify the binary variables of the PLA which will be paired together. The binary variables are numbered starting with 1. The PLA will be reshaped so that any unpaired binary variables occupy the leftmost part of the array, then the paired multiple-valued columns, and finally any multiple-valued variables.

.kiss Sets up for a *kiss*-style minimization.

.p [d] Specifies the number of product terms. The product terms (one per line) follow immediately after this keyword. Actually, this line is ignored, and the ".e", ".end", or the end of the file indicate the end of the input description.

.e (.end) Marks the end of the PLA description.

LOGICAL DESCRIPTION OF A PLA

When we speak of the ON-set of a Boolean function, we mean those minterms which imply the function value is a 1. Likewise, the OFF-set are those terms which imply the function is a 0, and the DC-set (don't care set) are those terms for which the function is unspecified. A function is completely described by providing its ON-set, OFF-set and DC-set. Note that all minterms lie in the union of the ON-set, OFF-set and DC-set, and that the ON-set, OFF-set and DC-set share no minterms.

The purpose of the *espresso* minimization program is to find a logically equivalent set of product-terms to represent the ON-set and optionally minterms which lie in the DC-set, without containing any minterms of the OFF-set.

A Boolean function can be described in one of the following ways:

1) By providing the ON-set. In this case, *espresso* computes the OFF-set as the complement of the ON-set and the DC-set is empty. This is indicated with the keyword ".type f" in the input file, or "-f" on the command line.

2) By providing the ON-set and DC-set. In this case, *espresso* computes the OFF-set as the complement of the union of the ON-set and the DC-set. If any minterm belongs to both the ON-set and DC-set, then it is considered a don't care and may be removed from the ON-set during the minimization process. This is indicated with the keyword ".type fd" in the input file, or "-fd" on the command line.

3) By providing the ON-set and OFF-set. In this case, *espresso* computes the DC-set as the complement of the union of the ON-set and the OFF-set. It is an error for any minterm to belong to both the ON-set and OFF-set. This error may not be detected during the minimization, but it can be checked with the subprogram "-do check" which will check the consistency of a function. This is indicated with the keyword ".type fr" in the input file, or "-fr" on the command line.

4) By providing the ON-set, OFF-set and DC-set. This is indicated with the keyword ".type fdr" in the input file, or "-fdr" on the command line.

If at all possible, *espresso* should be given the DC-set (either implicitly or explicitly) in order to improve the results of the minimization.

A term is represented by a "cube" which can be considered either a compact representation of an algebraic product term which implies the function value is a 1, or as a representation of a row in a PLA which implements the term. A cube has an input part which corresponds to the input plane of a PLA, and an output part which corresponds to the output plane of a PLA (for the multiple-valued case, see below).

SYMBOLS IN THE PLA MATRIX AND THEIR INTERPRETATION

Each position in the input plane corresponds to an input variable where a 0 implies the corresponding input literal appears complemented in the product term, a 1 implies the input literal appears uncomplemented in the product term, and - implies the input literal does not appear in the product term.

With logical type *f*, for each output, a 1 means this product term belongs to the ON-set, and a 0 or - means this product term has no meaning for the value of this function. This logical type corresponds to an actual PLA where only the ON-set is actually implemented.

With logical type *fd* (the default), for each output, a 1 means this product term belongs to the ON-set, a 0 means this product term has no meaning for the value of this function, and a - implies this product term belongs to the DC-set.

With logical type *fr*, for each output, a 1 means this product term belongs to the ON-set, a 0 means this product term belongs to the OFF-set, and a - means this product term has no meaning for the value of this function.

With logical type *fdr*, for each output, a 1 means this product term belongs to the ON-set, a 0 means this product term belongs to the OFF-set, a - means this product term belongs to the DC-set, and a ˜ implies this product term has no meaning for the value of this function.

Note that regardless of the logical type of PLA, a ˜ implies the product term has no meaning for the value of this function. 2 is allowed as a synonym for -, 4 is allowed for 1, and 3 is allowed for ˜. Also, the logical PLA type can also be specified on the command line.

MULTIPLE-VALUED FUNCTIONS

Espresso will also minimize multiple-valued Boolean functions. There can be an arbitrary number of multiple-valued variables, and each can be of a different size. If there are also binary-valued variables, they should be given as the first variables on the line (for ease of description). Of course, it is always possible to place them anywhere on the line as a two-valued multiple-valued variable. The function size is described by the imbedded option ".mv" rather than ".i" and ".o".

.mv [num_var] [num_binary_var] [s1] . . . [sn]

Specifies the number of variables (num_var), the number of binary variables (num_binary_var), and the size of each of the multiple-valued variables (s1 through sn). A multiple-output binary function with *ni* inputs and *no* outputs would be specified as ".mv *ni+1 ni no*." ".mv" cannot be used with either ".i" or ".o" − use one or the other to specify the function size.

The binary variables are given as described above. Each of the multiple-valued variables are given as a bit-vector of 0 and 1 which have their usual meaning for multiple-valued functions. The last multiple-valued variable (also called the output) is interpreted as described above for the output (to split the function into an ON-set, OFF-set and DC-set). A vertical bar "|" may be used to separate the multiple-valued fields in the input file.

If the size of the multiple-valued field is less than zero, than a symbolic field is interpreted from the input file. The absolute value of the size specifies the maximum number of unique symbolic labels which are expected in this column. The symbolic labels are white-space delimited strings of characters.

To perform a *kiss*-style encoding problem, either the keyword **.kiss** must be in the file, or the **-kiss** option must be used on the command line. Further, the third to last variable on the input file must be the symbolic "present state", and the second to last variable must be the "next state". As always, the last variable is the output. The symbolic "next state" will be hacked to be actually part of the output.

EXAMPLE #1

A two-bit adder which takes in two 2-bit operands and produces a 3-bit result can be described completely in minterms as:

```
# 2-bit by 2-bit binary adder (with no carry input)
.i 4
.o 3
.type fr
.pair 2 (1 3) (2 4)
.phase 011
00 00      000
00 01      001
00 10      010
00 11      011
01 00      001
01 01      010
01 10      011
01 11      100
10 00      010
10 01      011
10 10      100
10 11      101
11 00      011
11 01      100
11 10      101
11 11      110
.end
```

The logical format for this input file (i.e., type fr) is given to indicate that the file contains both the ON-set and the OFF-set. Note that in this case, the zeros in the output plane are really specifying "value must be zero" rather than "no information".

The imbedded option *.pair* indicates that the first binary-valued variable should be paired with the third binary-valued variable, and that the second variable should be paired with the fourth variable. The function will then be mapped into an equivalent multiple-valued minimization problem.

The imbedded option *.phase* indicates that the positive-phase should be used for the second and third outputs, and that the negative phase should be used for the first output.

EXAMPLE #2

This example shows a description of a multiple-valued function with 5 binary variables and 3 multiple-valued variables (8 variables total) where the multiple-valued variables have sizes of 4 27 and 10 (note that the last multiple-valued variable is the "output" and also encodes the ON-set, DC-set and OFF-set information).

```
.mv 8 5 4 27 10
0 - 0 1 0 | 1 0 0 0 | 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 1 0 0 0 0 0 0 0
1 0 - 1 0 | 1 0 0 0 | 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 0 0 0 0 0 0 0 0 0
0 - 1 1 1 | 1 0 0 0 | 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 0 0
0 - 1 0 - | 1 0 0 0 | 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 | 1 0 0 0 | 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 | 1 0 0 0 | 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 1 0 0 0 0 0 0 0
0 1 0 0 1 | 1 0 0 0 | 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 1 0
0 1 0 1 - | 1 0 0 0 | 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0
0 - 0 - 0 | 1 0 0 0 | 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 | 1 0 0 0 | 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 | 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 1 0 0 0 0 0 0 0
1 0 - 1 0 | 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 | 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 1 0 0 0 0 0 0 0
                                  .
                                  .
                                  .
1 1 1 1 1 | 0 0 0 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 0 0 0 0
```

EXAMPLE #3

This example shows a description of a multiple-valued function setup for *kiss*-style minimization. There are 5 binary variables, 2 symbolic variables (the present-state and the next-state of the FSM) and the output (8 variables total).

```
.mv 8 5 -10 -10 6
.type fr
.kiss
# This is a translation of IOFSM from OPUS
# inputs are      IO1 IO0 INIT SWR MACK
# outputs are     WAIT MINIT MRD SACK MWR DLI
# reset logic
--1--     -        init0    110000
# wait for INIT to go away
--1--     init0    init0    110000
--0--     init0    init1    110000
# wait for SWR
--00-     init1    init1    110000
--01-     init1    init2    110001
# Latch address
--0--     init2    init4    110100
# wait for SWR to go away
--01-     init4    init4    110100
--00-     init4    iowait   000000
# wait for command from MFSM
0000-     iowait   iowait   000000
1000-     iowait   init1    110000
01000     iowait   read0    101000
11000     iowait   write0   100010
01001     iowait   rmack    100000
11001     iowait   wmack    100000
--01-     iowait   init2    110001
# wait for MACK to fall (read operation)
--0-0     rmack    rmack    100000
--0-1     rmack    read0    101000
# wait for MACK to fall (write operation)
--0-0     wmack    wmack    100000
--0-1     wmack    write0   100010
# perform read operation
--0--     read0    read1    101001
--0--     read1    iowait   000000
# perform write operation
--0--     write0   iowait   000000
.end
```