

# Using espresso

## 1 Introduction

**espresso** is part of the OctTools suite of integrated-circuit design tools developed at the University of California at Berkeley. Its main function is to find simple SOP expressions for arbitrary truth tables, possibly including don't-cares and for possibly multi-output functions.

As pointed out in class, **espresso** is *not* designed to find absolutely minimal SOP forms due to the potentially staggering computational cost of doing this for “bad” large problems. Instead, it is guaranteed to find an **irredundant cover** of the ON-set of a function and uses heuristics to try to make this cover simple. For small problems, however, (with up to five or six inputs, say), it is *very* unlikely that **espresso**'s solution is not minimal.

### 1.1 Accessing espresso

**espresso** is available on Kettering's network of Sun SPARCstations running the Solaris II operating system. This includes nova, the CAD labs, and the ME Truck lab. In order to use it (as well as other OctTools programs we'll use later), you must log in to one of these systems and set your `cs`h environment appropriately. To do this, you can just execute the following three commands at a `cs`h prompt:

```
setenv OCTTOOLS      /home/faculty/mcdonald/octtools-5.2
setenv PATH          ${PATH}:${OCTTOOLS}/bin
setenv MANPATH       ${MANPATH}:${OCTTOOLS}/man
```

Alternatively—and better—you can include the above three lines in the `.cshrc` file in your home directory; then these commands will be executed and the OctTools will be available to you automatically each time you log in.

### 1.2 For More Information

This handout gives only a brief introduction to **espresso**. Much more information is available:

- On the SPARCs, read the UNIX `man` page on **espresso** by typing `man espresso` or, for complete information on its input file format, `man -s 5 espresso`.
- HTML-ized versions of the `man` pages are linked to the course Web page.
- A tutorial and copies of the `man` pages (in PostScript format and viewable using `pageview`) are linked to the course Web page.
- User and reference manuals (also in PostScript format) for the entire OctTools suite are available at

```
/home/faculty/mcdonald/octtools-5.2/docs/userman.ps and
/home/faculty/mcdonald/octtools-5.2/docs/refman.ps,
```

respectively.

## 2 Basic Use

**espresso** is a UNIX command-line program that operates on an input file, or on text fed to its **standard input**, and produces text output on its **standard output**, which is usually connected to the terminal but can be redirected through the UNIX shell to an output file if desired. Details of its operation can be controlled using options included on the command line. The command synopsis is

```
espresso [options] [file]
```

The square brackets indicate optional arguments. If the **file** argument is not included, **espresso** operates instead on its standard input (i.e., it will wait for you to type data into it, or you can use the shell's redirection capability to **pipe** data to it). If the **options** argument is not included, **espresso** will operate on its input data in the default way and produce its output on its standard output in the default format.

### 2.1 Single-Output Functions

**espresso** expects its input in **Berkeley PLA format**, which is simply a way of specifying the truth table of one or more Boolean functions. It is *very* flexible, allowing incompletely specified functions, multi-output functions, and multivalued logic, amongst other things. However, we'll start simple with a single-output function.

**Example.** You can specify a Boolean function like

$$Y = \sum m(0, 3, 5, 12, 13) + \sum d(1, 2, 15)$$

in PLA format as follows:

```
#####
# An example PLA-format file
#####
# number of inputs
.i 4
# number of outputs
.o 1
# names of inputs
.ilb A B C D
# names of outputs
.ob Y
# ON-set
0000 1
0011 1
0101 1
1100 1
1101 1
# DC-set ("- " means don't-care)
0001 -
0010 -
1111 -
.e
```

Note that lines beginning with **#** are comments. Note also that only the ON-set and DC-set of the function are specified—**espresso** assumes that the OFF-set is everything else. An optional

Keyword	Meaning
.type f	specified ON-set, empty DC-set, OFF-set is complement of ON-set
.type r	specified OFF-set, empty DC-set, ON-set is complement of OFF-set
.type fd	specified ON-set and DC-set, OFF-set is complement of their union (default)
.type fr	specified ON-set and OFF-set, DC-set is complement of their union
.type dr	specified DC-set and OFF-set, ON-set is complement of their union
.type fdr	all sets specified (can cause error)

Table 1: *espresso*'s .type keyword

.type keyword can be used to change this interpretation so that functions can be entered as compactly as possible. Table 1 summarizes the possibilities.

If the above PLA description is in a file called, say, `ex1.pla`, you can simplify it as follows:

```
{nova}> espresso ex1.pla
```

This will produce the following output on your terminal (where I've omitted the comments, which are passed through in a bunch at the beginning):

```
.i 4
.o 1
.ilb A B C D
.ob Y
.p 3
-101 1
00-- 1
110- 1
.e
```

This is again in PLA format, and means that the simplest form for  $Y$  that *espresso* could find is

$$Y = B\bar{C}D + \bar{A}\bar{B} + AB\bar{C}$$

Note that each output row corresponds to a single product term. In the left-hand columns, a 1 means the corresponding variable appears in the term, a 0 means it appears complemented, and a - means it doesn't appear at all.

If you wanted your output in a file called, say, `ex1-out.pla` instead of on the terminal, you could invoke *espresso* as follows:

```
{nova}> espresso ex1.pla > ex1-out.pla
```

to redirect standard output to the desired file.

To get the output in perhaps a more readable form, use the “-o eqntott” option, which produces output in the Boolean-expression-like **eqntott format**:

```
{nova}> espresso -o eqntott ex1.pla
:      (comments)
Y = (B&!C&D) | (!A&!B) | (A&B&!C);
```

Note that, in eqntott format, “&” means AND, “|” means OR, and “!” means NOT.

## 2.2 Multi-Output Functions

PLA format can be used to express multi-output functions for simplification by *espresso*, which produces a multi-output PLA-format result. Interpreting these can be tricky until you get the hang of it ...

**Example.** Here's the PLA-format description of the three-output function we looked at in class:

```
.i 4
.o 3
.ilb A B C D
.ob f1 f2 f3
0010 110
0011 110
0101 110
0110 011
0111 111
1000 101
1001 101
1010 110
1011 110
1101 101
1110 011
1111 111
.e
```

Each row represents a minterm (although it could represent any product term) and, in the output columns, a 1 means the minterm belongs to the ON-set of the corresponding function and a 0 means it does not. The OFF-sets of the three output functions are simply the complements of their ON-sets (since there are no don't-cares in this problem), and the 0s in the output columns have nothing to do with it! This can be *very* confusing at first.

In any case, the above input can be simplified as follows (assuming it's in a file `ex2.pla`):

```
{nova}> espresso ex2.pla
.i 4
.o 3
.ilb A B C D
.ob f1 f2 f3
.p 5
11-1 101
100- 101
01-1 110
-01- 110
-11- 011
.e
```

This just means that

$$f_1 = ABD + A\bar{B}\bar{C} + \bar{A}BD + \bar{B}C$$

$$f_2 = \bar{A}BD + \bar{B}C + BC$$

$$f_3 = ABD + A\bar{B}\bar{C} + BC$$

**Separate Minimization.** By default, *espresso* tries to reduce the *total number of product terms* required to minimize all outputs of a multi-output function simultaneously. This is not always what is desired.

The three outputs of our example function can be minimized *separately* as follows:

```
{nova}> espresso -Dso ex2.pla
.i 4
.o 3
.ilb A B C D
.ob f1 f2 f3
.p 8
10-- 100
-01- 100
-1-1 100
01-1 010
--1- 010
1-01 001
100- 001
-11- 001
.e
```

which means that the simplest expressions *espresso* could find *separately* for the three outputs are

$$\begin{aligned}f_1 &= A\bar{B} + \bar{B}C + BD \\f_2 &= \bar{A}BD + C \\f_3 &= A\bar{C}D + A\bar{B}\bar{C} + BC\end{aligned}$$

Note that, while each *individual* output is simpler than in the default case, this solution requires a total of eight unique product terms, as opposed to five.

## 2.3 Obtaining POS Forms

*espresso* is oriented entirely toward SOP forms, but it can be used to obtain simple POS forms.

**Example.** When the *-epos* option is used, *espresso* finds a simple SOP form for the *complement* of a function. Here's an example:

```
{nova}> espresso -epos ex1.pla
.i 4
.o 1
.ilb A B C D
.ob Y
#.phase 0
.p 3
10-- 1
-11- 1
01-0 1
.e
```

This means that

$$\bar{Y} = A\bar{B} + BC + \bar{A}B\bar{D}$$

Using DeMorgan's Theorem twice we find:

$$\begin{aligned}
 Y = \bar{Y} &= \overline{A\bar{B} + BC + \bar{A}B\bar{D}} \\
 &= (\overline{A\bar{B}})(\overline{BC})(\overline{\bar{A}B\bar{D}}) \\
 &= (\bar{A} + B)(\bar{B} + \bar{C})(A + \bar{B} + D)
 \end{aligned}$$

### 3 Options

We've seen a few of *espresso*'s simplest and most frequently used options, but there are many more. Using them, you can get *espresso* to do some nice things:

- Dcheck Checks that the function is a partition of the entire space (i.e., that the ON-set, OFF-set and DC-set are pairwise disjoint, and that their union is the Universe).
- Dexact Exact minimization algorithm (guarantees minimum number of product terms, and heuristically minimizes number of literals). Potentially expensive.
- Dmap Draw the Karnaugh maps for a binary-valued function.
- Dverify Checks for Boolean equivalence of two PLA's. Reads two filenames from the command line, each containing a single PLA.
- efast Stop after the first EXPAND and IRREDUNDANT operations (i.e., do not iterate over the solution).

The above, excerpted from the *espresso* man page, is only a partial list.