ESD Electronic Systems Design
Synthesis Verification Testing Power Communication

Espresso
Two-level Boolean minimization

Alessandro Danese
Tiziano Villa

University of Verona
Dep. Computer Science
Italy

Agenda

- Introduction
- *espresso* – two-level Boolean minimization
- *espresso* - Input file
  - description format
  - keywords
- *espresso* - Options
- Exercises

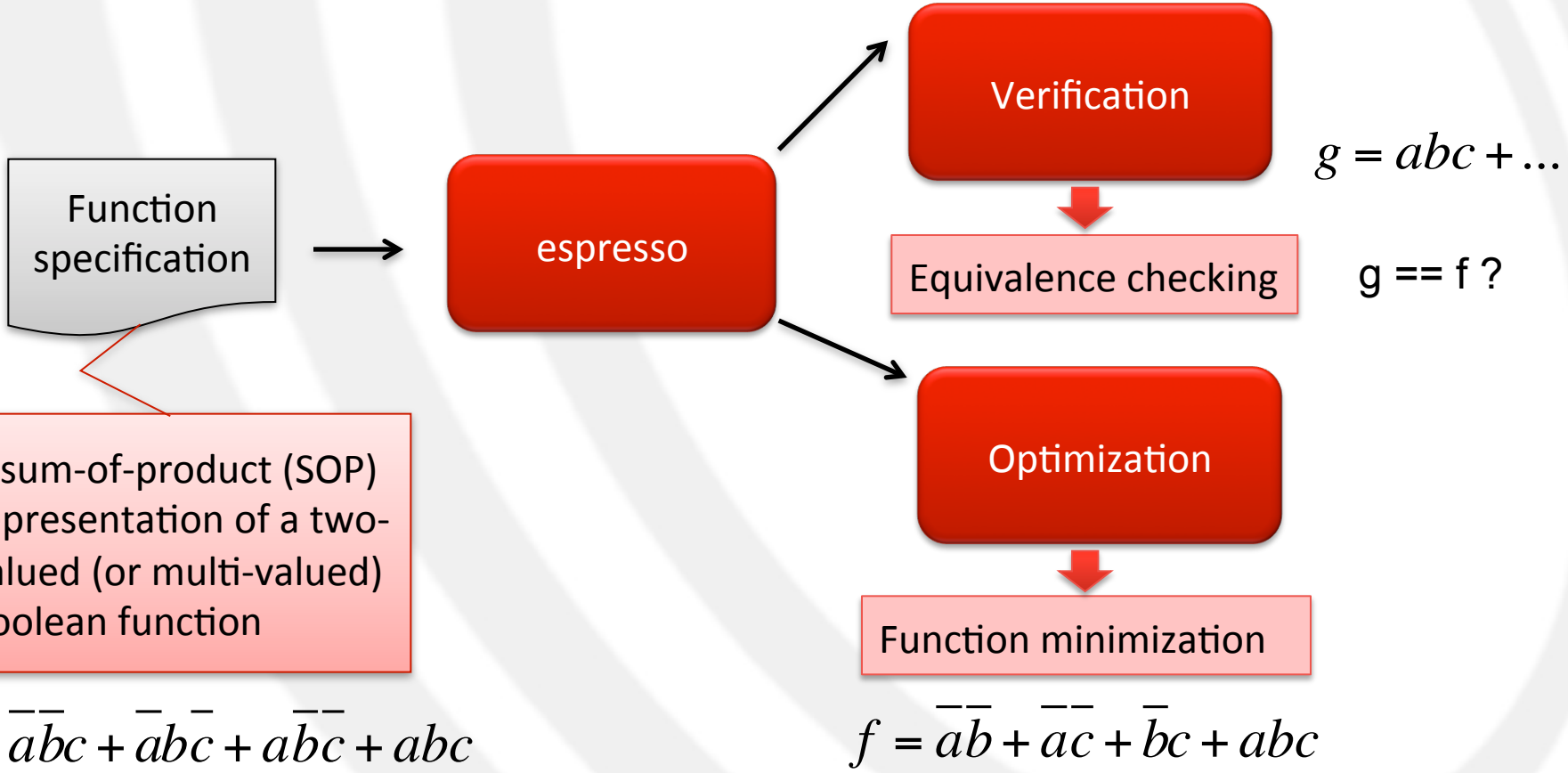
Introduction

- A Boolean function can be described providing:
  - ON-set
    - The DC-set is empty
    - OFF-set is the complement of the ON-set.
  - ON-set and DC-set
    - OFF-set is the complement of the union of ON-set and DC-set
  - ON-set and OFF-set
    - DC-set is the complement of the union of ON-set and OFF-set
- A Boolean function is completely described by providing its **ON-set**, **OFF-set** and **DC-set**.

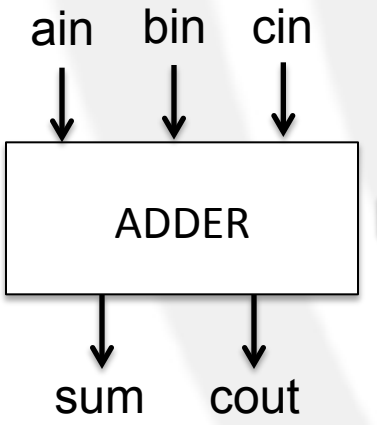
Espresso - U.C. Berkeley

- *espresso* is a program for **two-level Boolean minimization** developed by the CAD group at U.C. Berkeley (software developer: Richard L. Rudell)
- Official release is available at http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm
  - Source code
  - Examples
  - Man pages for *espresso*

What can we do with espresso ?



Running example - Adder



ain	bin	cin	sum	cout
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1
1	1	0	0	1
0	1	1	0	1
1	0	1	0	1

$$sum = \overline{ain} * \overline{bin} * cin + \overline{ain} * bin * \overline{cin} + ain * \overline{bin} * \overline{cin} + ain * bin * cin$$

$$cout = ain * bin * \overline{cin} + \overline{ain} * bin * cin + ain * bin * cin + ain * \overline{bin} * cin$$

espresso – Basic usage

$>*espresso* [options] [in_file] [out_file]

- Reads the *in_file* provided
  - Or the standard input if no file is specified
- Writes the minimized results in *out_file*
  - Or to the standard output if no file is specified

espresso – Input file format (V)

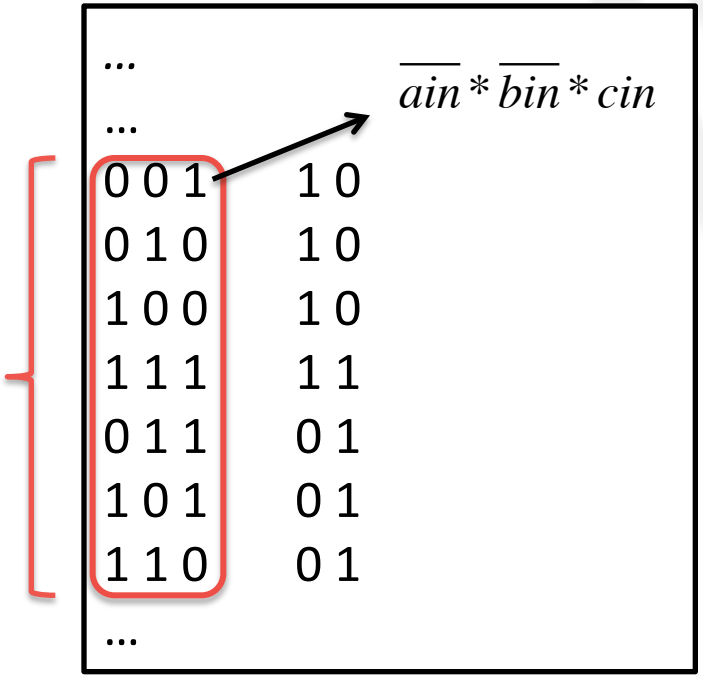
```
# num of input vars
# e.g., ain, bin, cin
.i 3
# num of output functions
# e.g., sum, cout
.o 2
...
...
...
...
...
...
...
...
.e
```

- The following keywords are recognized by *espresso*:
  - **.i** [d]
    - specifies the number "d" of input variables
  - **.o** [d]
    - specifies the number "d" of output variables
  - **.e**
    - optionally marks the end of the description

espresso – Input file format (I)

ain	bin	cin	sum	cout
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1
1	1	0	0	1
0	1	1	0	1
1	0	1	0	1

Matrix format



- each position in the input matrix corresponds to an input variable where:
  - 0 implies the corresponding input literal appears complemented in the product term
  - 1 implies the input literal appears uncomplemented in the product term
  - - implies the input literal does not appear in the product term

espresso – Input file format (II)

- Semantics of output part
  - Specifying the format of each function
    - type *f*:
      - a **1** means this product term belongs to the *ON-set*, and **0** or **–** means this product term has *no meaning*. (specified ON-set, empty DC-set, OFF-set is the complement of ON-set).
    - type *fd (default type)*:
      - a **1** means this product term belongs to the *ON-set*, **–** implies this product term belongs to the *DC-set.* **0** means this product term has *no meaning*. (specified ON-set and DC-set, OFF-set is the complement of their union).

```
...
.type [f|fd|fr|fdr]
0 0 1    1 0
0 1 0    1 0
1 0 0    1 0
1 1 1    1 1
0 1 1    0 1
1 0 1    0 1
1 1 0    0 1
...
```

espresso – Input file format (III)

- Semantics of output part
  - Specifying the format of each function
    - type *fr*:
      - 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*. (specified ON-set and OFF-set, DC-set is complement of their union)
    - type *fdr*:
      - 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*. *(all sets specified)*

```
...
.type [f|fd|fr|fdr]
0 0 1    1 0
0 1 0    1 0
1 0 0    1 0
1 1 1    1 1
0 1 1    0 1
1 0 1    0 1
1 1 0    0 1
...
```

espresso – Input file format (IV)

```
# num of input vars
# e.g., ain, bin, cin
.i 3
# num of output functions
# e.g., sum, cout
.o 2
.type fr
0 0 1      1 0
0 1 0      1 0
1 0 0      1 0
1 1 1      1 1
0 1 1      0 1
1 0 1      0 1
1 1 0      0 1
.e
```

- The following keywords are recognized by *espresso*:
  - comments
    - allowed using #
  - whitespaces:
    - Blanks, tabs … are ignored

espresso – Input file format (VI)

```
# num of input vars
# e.g., ain, bin, cin
.i 3
# num of output functions
# e.g., sum, cout
.o 2
.type fr
0 0 1      1 0
0 1 0      1 0
1 0 0      1 0
1 1 1      1 1
0 1 1      0 1
1 0 1      0 1
1 1 0      0 1
.e
```


```
~$ espresso adder_espresso.txt
# num of input vars
# e.g., ain, bin, cin
# num of output functions
# e.g., sum, cout
.i 3
.o 2
.p 7
111 10
-00  10
0-0  10
00-  10
-11  01
1-1  01
11-  01
.e
```

espresso – Input file keywords (VII)

- **.phase** [b1] [b2] .. [bn]
  - It specifies which polarity of each output function should be used for the minimization
    - (1): specifies that the ON-set of the corresponding output function should be used;
    - (0): specifies that the OFF-set of the corresponding output function should be used;
  - Optional

espresso – Input file format (VI)

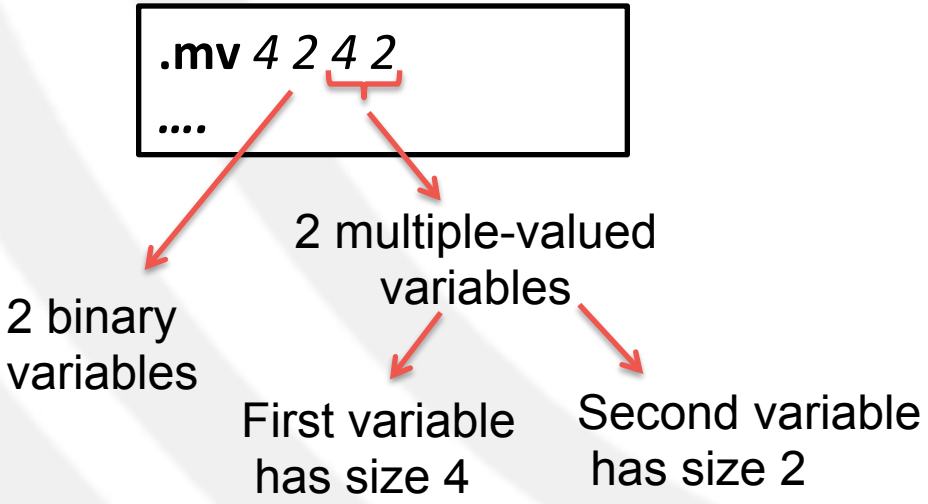
```
.i 3
.o 2
.ilb ain bin cin
.ob sum cout
.type fr
0 0 1    1 0
0 1 0    1 0
1 0 0    1 0
1 1 1    1 1
0 1 1    0 1
1 0 1    0 1
1 1 0    0 1
.e
```

- The following keywords are recognized by *espresso*:
  - **.ilb** [s1] [s2] .. [sn]
    - gives the names of the binary-valued variables
    - must come after **.i** and **.o**
    - as many tokens as input variables
  - **.ob** [s1] [s2] .. [sn]
    - gives the names of the output function
    - must come after **.i** and **.o**
    - as many tokens as output variables

espresso – Input file keywords (VIII)

- **.mv** [num_var] [num_bin_var] [d1] . . . [dN]
  - specifies the number of variables (num_var), the number of binary variables (*num_bin_var*) and the size of each of the multiple-valued variables (*d1* through *dN*)

- example



espresso – Input file keywords (VIII)

- **Example: Single primary output of an FSM**
  - 3 inputs: state variable (S), two inputs (c_1, c_2)
  - 4 states: s_0, s_1, s_2, s_3
  - y is 1 when:
    - ($S=s_0$) and c_2
    - ($S=s_0$) or ($S=s_2$) and not c_1
    - ($S=s_1$) and not c_2 and c_1
    - ($S=s_3$) or ($S=s_2$) and c_1

espresso – Input file format (VI)

$$y = S^{\{0\}} * c_2 + S^{\{0,2\}} * \overline{c_1} + S^{\{1\}} * \overline{c_2} c_1 + S^{\{2,3\}} * c_1$$

```
.mv 4 2 4
1- 1000
-0 1010
01 0100
-1 0011
.e
```

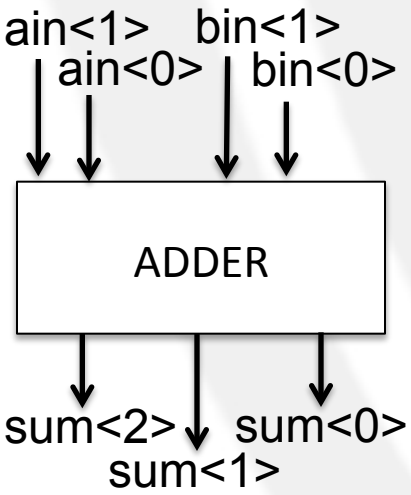

```
~$ espresso automaton.txt
.mv 3 2 4
.p 3
11 1011
01 0111
-0 1010
.e
```

$$y = S^{\{0,2,3\}} * c_2 c_1 + S^{\{1,2,3\}} * \overline{c_2} c_1 + S^{\{0,2\}} * \overline{c_1}$$

espresso – Input file keywords (VIII)

- **.symbolic** [s0]..[sN] ; [t0] .. [tM] ;
  - the binary variables named [s0] thru [sN] must be considered as a single multiple-valued variable
    - variable with 2^N parts corresponding to the decodes of the binary-valued variables
  - [s0] is the most significant bit, [sN] is the least significant bit
  - [t0] .. [tm] provide the labels for each decode of [s0] thru [sN]

espresso – Input file keywords (IX)




```
.i 4
.o 3
.ilb ain<1> ain<0> bin<1> bin<0>
.ob sum<2> sum<1> sum<0>

.symbolic ain<1> bin<1> ; ;
.symbolic ain<0> bin<0> ; ;

0 0  0 0  0 0 0
0 0  0 1  0 0 1
0 0  1 0  0 1 0
0 0  1 1  0 1 1
0 1  0 0  0 0 1
```

```
...

0 1  0 1  0 1 0
0 1  1 0  0 1 1
0 1  1 1  1 0 0
1 0  0 0  0 1 0
1 0  0 1  0 1 1
1 0  1 0  1 0 0
1 0  1 1  1 0 1
1 1  0 0  0 1 1
1 1  0 1  1 0 0
1 1  1 0  1 0 1
1 1  1 1  1 1 0
.e
```

espresso – Options (I)

- Interesting options for running *espresso* are:
  - **-D***check*
    - checks that ON-set, OFF-set, DC-set are disjoint
  - **-D***exact*
    - performs exact minimization (potentially expensive)
  - **-D***many*
    - reads and minimizes all PLA defined into the input file
  - **-D***opo*
    - performs output phase optimization, i.e., reduce the number of terms needed to implement the function or its complement

espresso – Options (II)

- **-D*verify***
  - checks for Boolean equivalence of two functions
  - requires two filenames from command line
- **-D*equiv***
  - identifies output variables which are equivalent
- **-D*so***
  - minimizes each function one at time as a single-output function
- **-epos**
  - swaps the ON-set and OFF-set of the function after reading the function
  - useful for minimizing the OFF-set of a function

espresso – Options (II)

- **-v**
  - verbose debugging details
  - activates all details
- **-d**
  - enables debugging
- **-o** [type]
  - selects the output format
  - type can be:
    - *f*: only On-set
    - *fd*: ON-set and DC-set
    - *fr*: ON-set and OFF-set
    - *fdr*: ON-set, OFF-set and DC-set

Exercise 1 (I)

- The Indian society of Natchez, who lived in North America, was divided into four groups: *Suns*, *Nobles*, *Honorables*, *Stinkards*.
  In this society, marriages were allowed according to specific rules, and the corresponding progeny belongs to a particular group as described in the following table:

Mother	Father	Progeny
Sun	Stinkard	Sun
Noble	Stinkard	Noble
Honorable	Stinkard	Honorable
Stinkard	Sun	Noble
Stinkard	Noble	Honorable
Stinkard	Honorable	Stinkard
Stinkard	Stinkard	Stinkard

- Other combinations are not allowed.

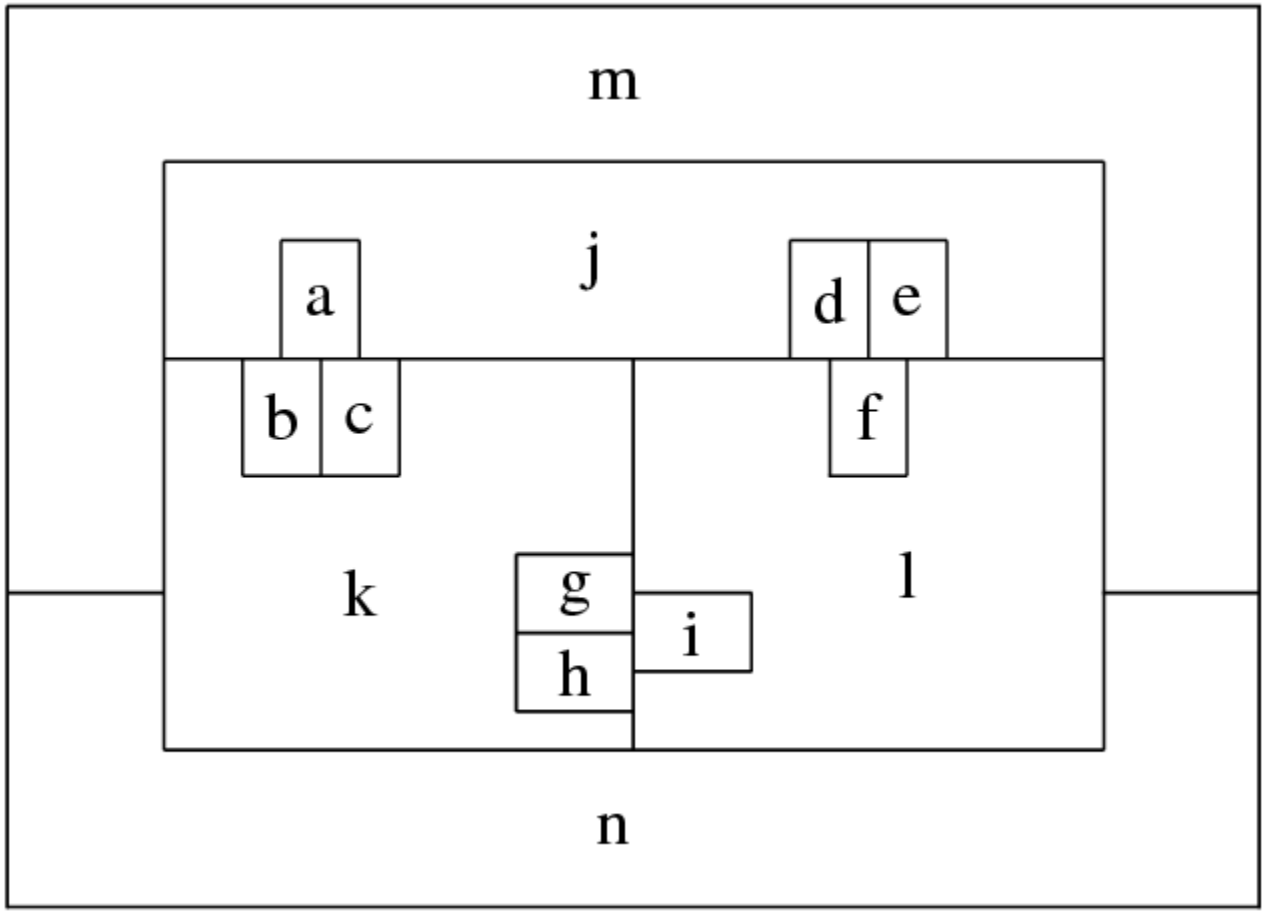
Exercise 1 (II)

1. Represent the condition that characterizes the progeny of type Stinkard using a multi-valued single product.
2. Represent, using the minimum number of multi-valued products, the illegal marriages.
3. Represent using the minimum number of multi-valued products the illegal marriages and progeny group.

Exercise 2 (I)

- Formulate the minimum map coloring problem (coloring a map with the minimum number of colors such that adjacent regions don't have the same color) as a logic minimization problem.
- Apply your formulation to the following map and use *espresso* to find a minimum coloring for the map.

Exercise 2 (II)



University of Verona - ESD release

- The latest version of the tool is installed in
  - /opt/EDA_Software/sse/espresso
- To set environment variabiles
  - source /opt/EDA_Software/start_eda.bash

  then select option 19 (SSE Tools)
- Several examples are available at
  - /opt/EDA_Software/sse/espresso/examples
- Man pages are available
  - man espresso

Man pages

- PLA format manual (espresso.5)
  - see examples
    - #1, a two bit adder
    - #2, multi-valued function
    - #3, multi-valued function setup for *kiss*-style minimization
- *espresso* usage manual (espresso.1)
  - List options by espresso -h