ESD Electronic Systems Design

Synthesis Verification Testing Power Communication

Espresso
Two-level Boolean minimization

Dott. Luigi Di Guglielmo

Prof. Tiziano Villa

University of Verona

Dep. Computer Science

Italy

UNIVERSITÀ DEGLI STUDI DI VERONA

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
    - OFF-set is the complement of the ON-set.
    - The DC-set is empty
  - 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 tool developed by the CAD group at U.C. Berkeley (software developer: Richard L. Rudell)
- Current release is the #2.3
  - Release date 01/31/1988
- *espresso* is a program that
  - Minimizes a sum of product representation of a Boolean function

espresso – Boolean Minimization

- *espresso* takes as input:
  - A sum-of-product (SOP) representation of a two-valued (or multi-valued) Boolean function
- and produces:
  - a minimal equivalent SOP representation
- How to use *espresso*:
  - espresso [options] <input file>

espresso – Input file

- *espresso* accepts as input a character *matrix* with *keywords* embedded
  - keywords specify:
    - the size of the matrix
    - the format of the input function
  - comments:
    - allowed using **#**
  - whitespaces:
    - Blanks, tabs … are ignored

espresso – Input file format (I)

- Semantics of input part
  - The format of the function
    - 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 the function
    - type *f*:
      - for each output, a **1** means this product term belongs to the *ON-set*, and **0** or **–** means this product term has *no meaning* for the value of this function
    - type *fd*:
      - for each output, a **1** means this product term belongs to the *ON-set*, **–** implies this product term belongs to the *DC-set* and a **0** means this product term has *no meaning* for the value of this function
      - it is the default type

espresso – Input file format (III)

- 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
- 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

espresso – Input file keywords (I)

- 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
  - **.type** [s]
    - specifies the logical interpretation of the output part of the character matrix
    - this keyword must come before any product term
    - [s] is one of *“f” “fd” “fr” “fdr”*
  - **.e**
    - optionally marks the end of the description

espresso – Input file keywords (II)

```
# example of sqrt function
.i 8
.o 5

00000000 00000
00000001 00001
00000010 00001
00000011 00010
00000100 00010
00000101 00010
…
11111101 10000
11111110 10000
11111111 10000
.e
```

espresso – Input file keywords (III)

- **.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
- **.symbolic** [s0]..[sN];
  - the binary variables named [s0] thru [sN] must be considered as a single multiple-valued variable
  - [s0] is the most significant bit, [sN] is the least significant bit

espresso – Input file keywords (IV)

- **.phase** [b1] [b2] .. [bn]
  - specifies the phase of each output
    - positive (1) or negative (0)
  - must come after **.i** and **.o**
  - as many tokens as output variables
- **.p** [d];
  - specifies the number [d] of products
  - optional

espresso – Input file keywords (V)

```
# example of multiple-valued variable
.i 4
.o 2
.ilb in1<1> in1<0> in2<1> in2<0>
.ob out<1> out<0>
.symbolic in1<1> in1<0>;
.symbolic in2<1> in2<0>;
.symbolic out<1> out<0>;

0000 01
0001 01
…
010- 00
-010 10
.e
```

espresso – Input file keywords (V)

- **.mv** [num_var] [num_bin_var] [d1] [dN]
  - specifies the number *num_var* of variables, the number *num_bin_var* of binary variables and the size of each of the multiple-valued variables (*d1* through *dN*)
- **.kiss**
  - sets up for a *kiss*-style minimization

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

University of Verona - ESD release

- The latest linux binary of the tool is available at
  - ~ldg/lectures/daes20112012/lesson01/espresso-64.bin
- Latest sources of the tool are available at
  - ~ldg/lectures/daes20112012/lesson01/archives/espresso.src.tar.gz
- Several examples are available at
  - ~ldg/lectures/daes20112012/lesson01/espresso.lesson-examples
- Man pages are available at
  - ~ldg/lectures/daes20112012/lesson01/espresso-64.bin/man/
- A repository is also available on the e-learning site

U.C. Berkeley – Official release

- Official *espresso* release is available at http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm
  - Source code
  - Examples
  - Man pages for *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

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.

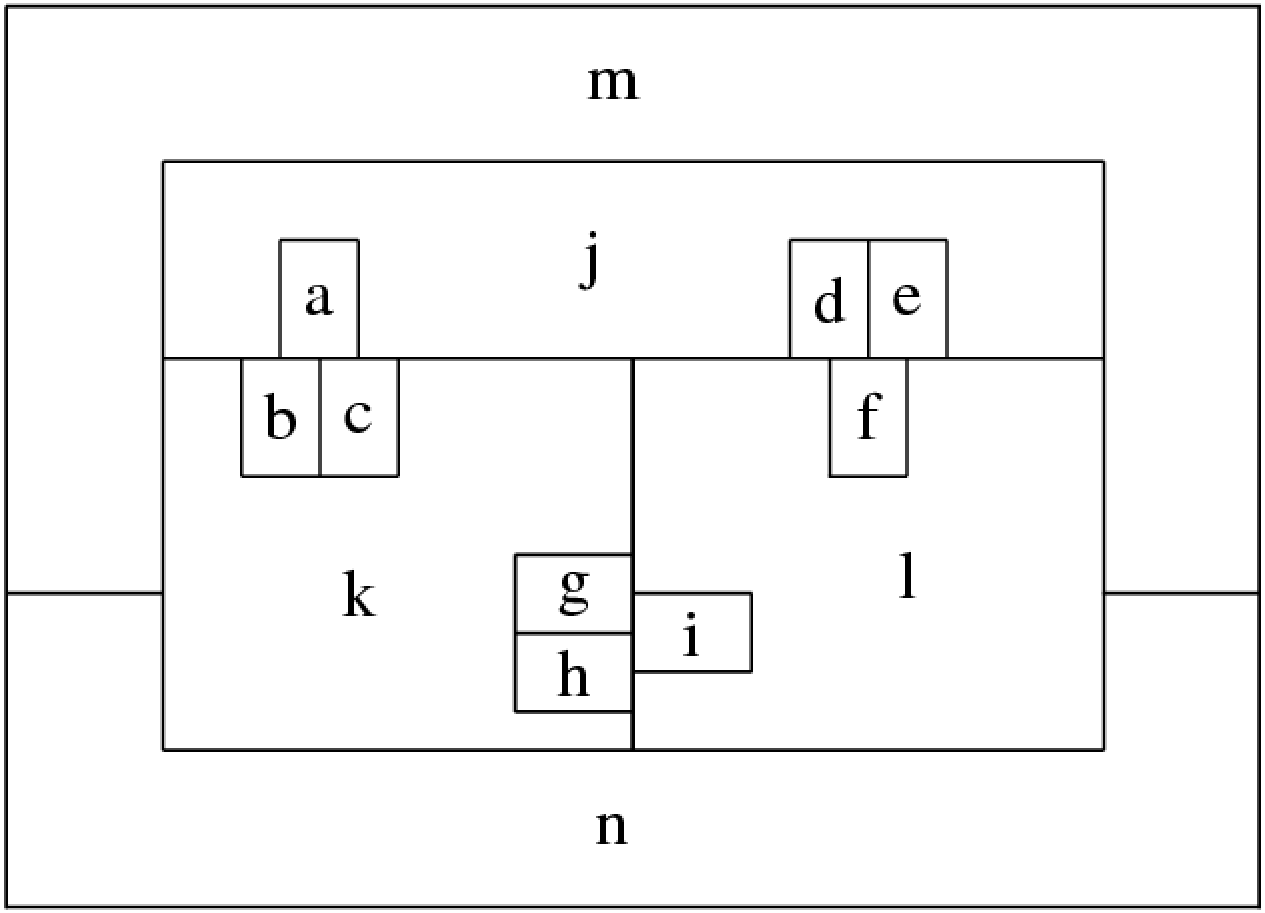
Exercise 1 (II)

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)


m
j
a
d
e
b
c
f
k
g
h
i
l
n