



Tackling circuit complexity (1)



Paolo PRINETTO
 Politecnico di Torino (Italy)
 University of Illinois at Chicago, IL (USA)

Paolo.Prinetto@polito.it
 Prinetto@uic.edu
 www.testgroup.polito.it

Goal

- This lecture is the former part of a group of 2 lectures aiming at presenting methods used in manual combinational synthesis to overcome the limitations of the purely manual approach presented in lectures 6.1-6.3
- It eventually shows how minimization tools can be used to exploit circuit complexity.

6.4

2

Prerequisites

- Lectures 5.2 and 6.3

6.4

3

Homework

- Students are warmly encouraged to solve the proposed exercises

6.4

4

Further readings

- No particular suggestion

6.4

5

Outline

- Tackling the complexity
- ROM-based synthesis
- K-map partitioning
- Partially automated approach.

6.4

6

Tackling the complexity

Purely manual synthesis can be applied when the # of PIs is very limited, only.
 To overcome such a very strong limitation, several approaches are possible, characterized by different applicability – cost trade-offs.

6.4

7

Tackling the complexity

We shall consider the following ones:

- ROM-based synthesis
- K-map partitioning
- Partitioning based techniques
- RT level minimizations
- Synthesis by iterating basic cells.

6.4

8

Outline

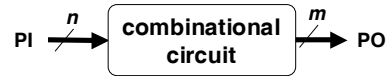
- Tackling the complexity
 ⇒ ROM-based synthesis
- K-map partitioning
- Partially automated approach

6.4

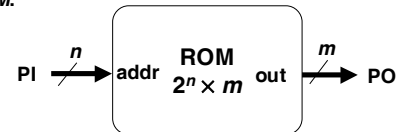
9

Rom-based Synthesis

Any n -inputs / m -outputs combinational circuit

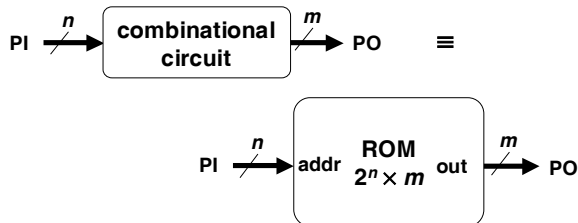


can easily be implemented resorting to a $(2^n \times m)$ ROM.



Rom-based Synthesis

The value that the m -outputs of the target circuit get when the value j is applied at the circuit's n -inputs must be stored in the m bits of the j -th cell of the ROM.



Examples

For sake of clarity, we shall consider some of the examples already seen in lecture 6.3.

6.4

12

Exercise #6.3.1

Design a combinational circuit with 1 output U and an input X (3 downto 0) that, when it receives on X an unsigned *hexadecimal* digit X, provides, on U, a logical value 1 iff:

$$X < 4 \text{ or } X > 8.$$



6.4

13

**Solution
(ROM 16x1)**

Address	Content	Address	Content
0000	1	1000	0
0001	1	1001	1
0010	1	1010	1
0011	1	1011	1
0100	0	1100	1
0101	0	1101	1
0110	0	1110	1
0111	0	1111	1



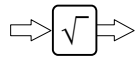
6.4

14

Exercise #6.3.2

Design a combinational circuit with a 4-bit input and a 3-bit output such that, when it receives on its input a *hexadecimal* digit X, provides, on its output:

$$\lceil \text{sqrt}(X) \rceil$$

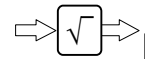


6.4

15

**Solution
(ROM 16x3)**

Address	Content	Address	Content
0000	000	1000	011
0001	001	1001	100
0010	010	1010	100
0011	010	1011	100
0100	010	1100	100
0101	011	1101	100
0110	011	1110	100
0111	011	1111	100



6.4

16

Look-up tables

ROM-based synthesis is particularly suited for combinational circuits acting as *look-up table*, i.e., tables storing values to be randomly read.

6.4

17

Outline

- Tackling the complexity
- ROM-based synthesis
- ⇒ *K-map partitioning*
- Partially automated approach

6.4

18

**K-map partitioning
or
MUX-based synthesis**

Such an approach stems from the Boole's expansion theorem (slide # 55 of lecture 3.1):

every Boolean function $f : B^n \rightarrow B$:

$$f(x_1, x_2, \dots, x_n)$$

can be expressed as:

$$f(x_1, x_2, \dots, x_n) = x_1' \cdot f_0(0, x_2, \dots, x_n) + x_1 \cdot f_1(1, x_2, \dots, x_n) \quad \forall (x_1, x_2, \dots, x_n) \in B$$

6.4

19

Basic idea

The expression

$$f(x_1, x_2, \dots, x_n) = x_1' \cdot f_0 + x_1 \cdot f_1$$

can obviously be interpreted as:

If $x_1 = 0$ (i.e., $x_1' = 1$)

then $f = f_0$

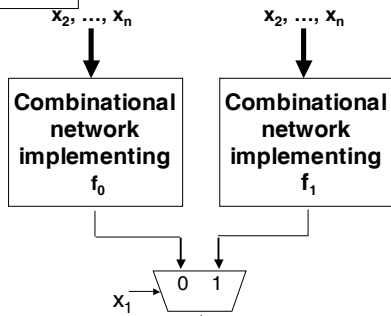
else $f = f_1$

which leads to the following implementation:

6.4

20

If $x_1 = 0$ (i.e., $x_1' = 1$)
then $f = f_0$
else $f = f_1$



6.4

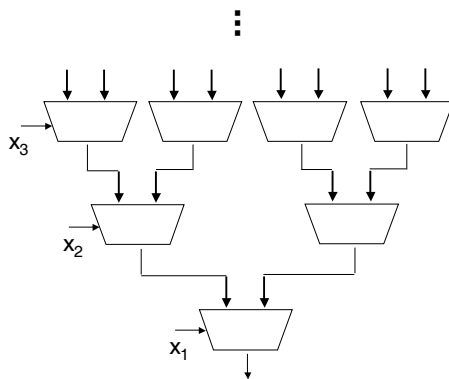
21

... and so on ...

The process can be easily iterated...

6.4

22



6.4

23

Example

Consider the following map:

		ab			
c		00	01	11	10
0		1	1	0	0
1		1	0	0	1
		U			

6.4

24

Example

Consider the following map:

	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

It can be partitioned as follows:

6.4

25

Example

Consider the following map:

	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

It can be partitioned as follows:

6.4

26

If $c=0$ then

	ab			
c	00	01	11	10
0	1	1	0	0

U

If $c=1$ then:

	ab			
c	00	01	11	10
1	1	0	0	1

U

Example

Consider the following map:

	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

It can be partitioned as follows:

6.4

27

If $c=0$ then

	ab			
c	00	01	11	10
0	1	1	0	0

U

a'

If $c=1$ then:

	ab			
c	00	01	11	10
1	1	0	0	1

U

b'

Example

Consider the following map:

	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

It can be partitioned as follows:

6.4

28

If $c=0$ then $U = a'$
if $c=1$ then $U = b'$

Example

Consider the following map:

	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

It can be partitioned as follows:

6.4

29

If $c=0$ then $U = a'$
if $c=1$ then $U = b'$

Alternative implementation

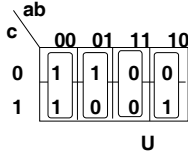
	ab			
c	00	01	11	10
0	1	1	0	0
1	1	0	0	1

U

6.4

30

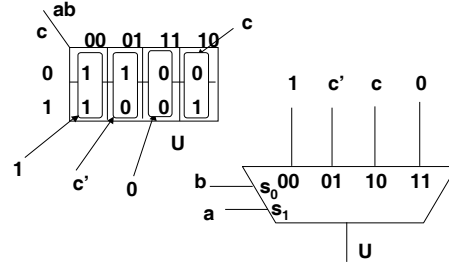
Alternative implementation



6.4

31

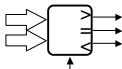
Alternative implementation



6.4

32

Exercise #6.3.4



Design a comparator which, receiving in input:

- A (1 downto 0) and B (1 downto 0)
- a signal 2C/~UM specifying the coding used for A and B (1=2's complement, 0=unsigned magnitude)

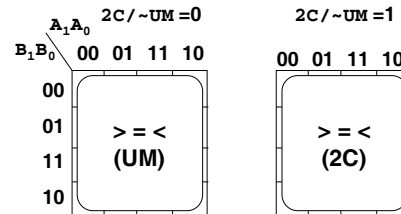
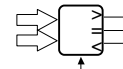
provides 3 outputs A_GT_B, A_EQ_B and A_LT_B such that:

- A_GT_B = 1 iff A > B
- A_EQ_B = 1 iff A = B
- A_LT_B = 1 iff A < B.

6.4

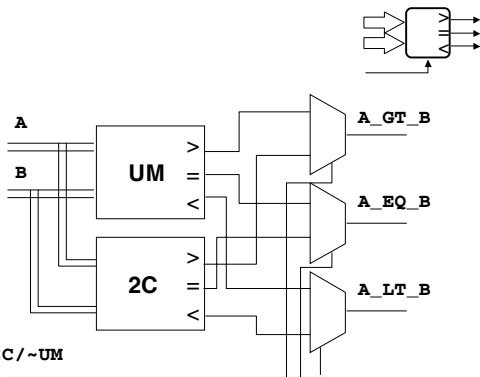
33

Solution



6.4

34



6.4

35

Outline

- Tackling the complexity
 - ROM-based synthesis
 - K-map partitioning
- ⇒ *Partially automated approach*

6.4

36

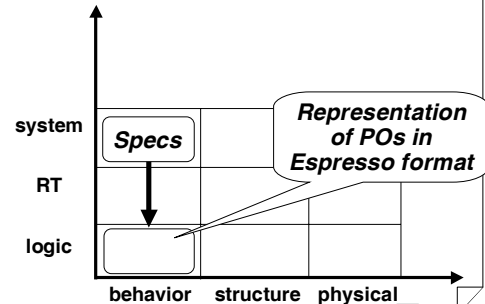
Partially automated approach

Logic level refinement and optimization can be easily performed resorting to the logic minimizer Espresso, developed at U.C. Berkeley (USA).

6.4

37

Step #3.1.1: Logic level refinements



6.4

38

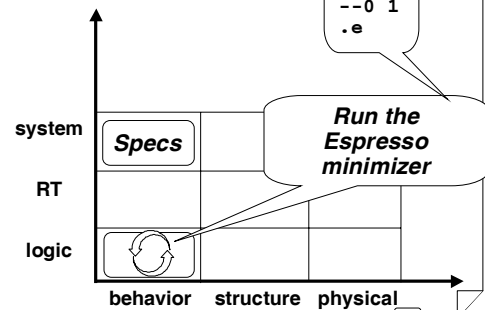
Espresso Format

```
.i 3      # of inputs
.o 1      # of outputs
00- 1    input_cube output_value
1-0 1    input_cube output_value
11- 1    input_cube output_value
.e       end of file
```

6.4

39

Step #3.1.2: Logic level minimization



6.4

40

Exercise #6.3.1



Design a combinational circuit with 1 output U and an input X (3 downto 0) that, when it receives on X an unsigned hexadecimal digit X, provides, on U, a logical value 1 iff:

$$X < 4 \text{ or } X > 8.$$

6.4

41

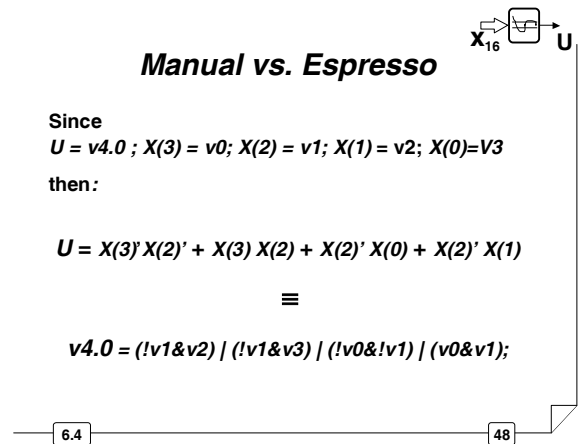
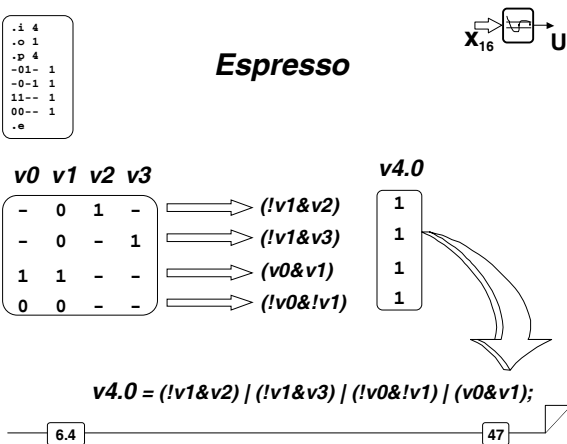
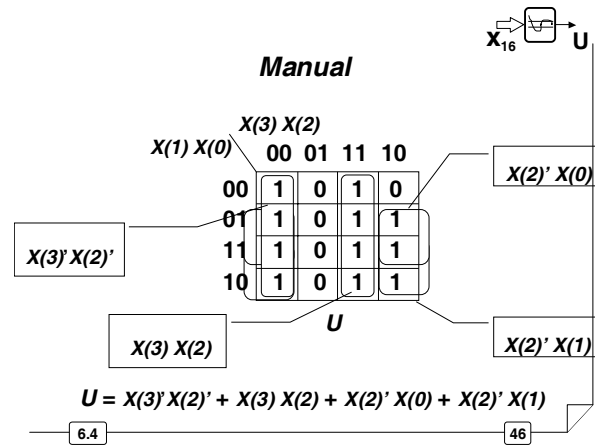
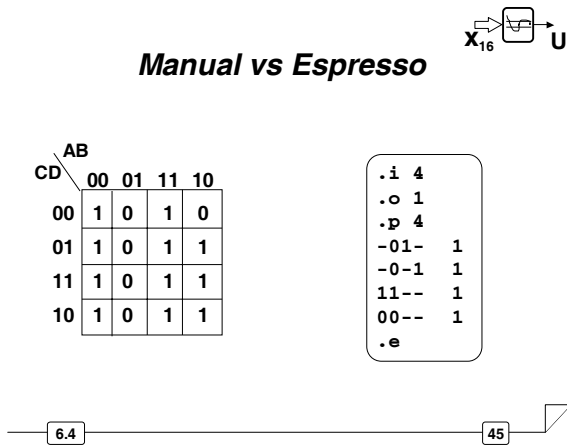
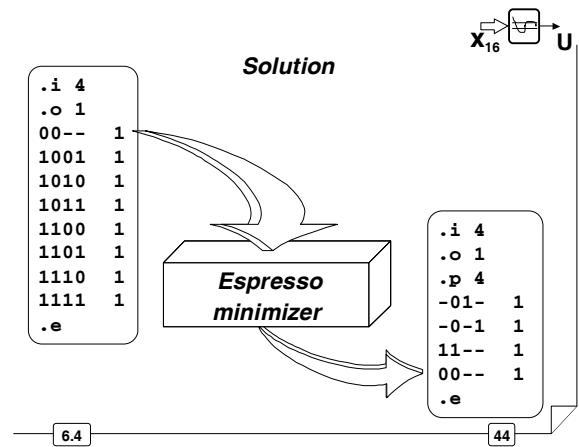
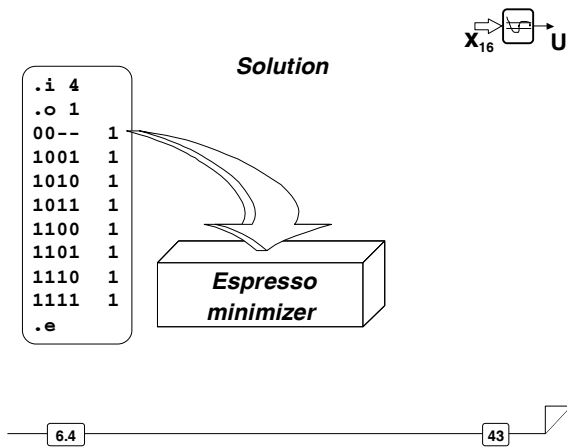
Solution

```
.i 4
.o 1
00-- 1
1001 1
1010 1
1011 1
1100 1
1101 1
1110 1
1111 1
.e
```



6.4

42



Exercise #6.3.2

Design a combinational circuit with a 4-bit input and a 3-bit output such that, when it receives on its input a *hexadecimal* digit X, provides, on its output:

$$\lceil \text{sqrt}(X) \rceil$$

6.4

49



```
.i 4
.o 3
0 0 0 0 0 0 0
0 0 0 1 0 0 1
0 0 1 - 0 1 0
0 1 0 0 0 1 0
0 1 0 1 0 1 1
0 1 1 - 0 1 1
1 0 0 - 0 1 1
1 0 1 - 1 0 0
1 1 - - 1 0 0
.e
```

Solution



```
.i 4
.o 3
.p 7
0-01 001
100- 011
011- 001
1-1- 100
11-- 100
0-1- 010
01-- 010
.e
```

6.4

50



Exercise #6.3.3

Similar to the example #1, with the only difference that the input digit X is, in this case, a *decimal* one.

6.4

51



```
.i 4
.o 1
00-- 1
1001 1
1010 -
1011 -
1100 -
1101 -
1110 -
1111 -
.e
```

Solution



```
.i 4
.o 1
.p 2
-0-1 1
00-- 1
.e
```

6.4

52



Note

The input file for Espresso can, in some cases, be easily generated by an ad-hoc program.

6.4

53

Exercise #6.3.4

Design a comparator which, receiving in input:

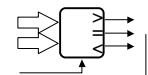
- A (1 downto 0) and B (1 downto 0)
- a signal 2C/~UM specifying the coding used for A and B (1=2's complement, 0=unsigned magnitude)

provides 3 outputs A_GT_B, A_EQ_B and A_LT_B such that:

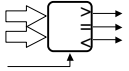
- A_GT_B = 1 iff A > B
- A_EQ_B = 1 iff A = B
- A_LT_B = 1 iff A < B.

6.4

54



Solution



```
#include <stdlib.h>
#include <stdio.h>

typedef int bit ;

void main(int argc, char **argv)
{
    bit A1, A0, B1, B0, SEL ;
    int A, B ;

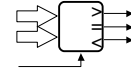
    printf(".i 5\n") ;
    /* 5 inputs: A1 A0 B1 B0 SEL */

    printf(".o 3\n") ;
    /* 3 outputs: AeqB AgtB AltB */
}
```

6.4

55

Solution #4.3



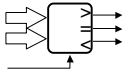
```
/* uguaglianza: xyxy- 100 */
for(A1=0; A1<=1; A1++)
for(A0=0; A0<=1; A0++)
for(B1=0; B1<=1; B1++)
for(B0=0; B0<=1; B0++)
for(SEL=0; SEL<=1; SEL++)
{
    /* conversione in decimale */
    if (SEL==0)
    {
        A = A1*2 + A0 ;
        B = B1*2 + B0 ;
    }
    else

```

6.4

56

Solution #4.3



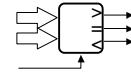
```
{
    A = -A1*2 + A0 ;
    B = -B1*2 + B0 ;
}

/* confronti */
if (A == B) printf("%d%d%d%d%d 100\n",
    A1, A0, B1, B0, SEL) ;
if (A > B) printf("%d%d%d%d%d 010\n",
    A1, A0, B1, B0, SEL) ;
if (A < B) printf("%d%d%d%d%d 001\n",
    A1, A0, B1, B0, SEL) ;
}
printf(".e\n") ;
exit(0) ;
}
```

6.4

57

Solution #4.3

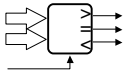


The previous C program generates the following input file:

6.4

58

```
.i 5
.o 3
00000 100
00001 100
00010 001
00011 001
00100 001
00101 010
00110 001
00111 010
01000 010
01001 010
01010 100
01011 100
01100 001
01101 010
01110 001
01111 010
.e
```

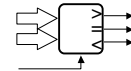


Which, in turn, can be minimized by Espresso as follows:

6.4

59

```
.i 5
.o 3
.p 12
0100- 010
1110- 010
0001- 001
1011- 001
0000- 100
1010- 100
0101- 100
1111- 100
1-0-0 010
0-1-1 010
0-1-0 001
1-0-1 001
.e
```



6.4

60

