



Some additional examples



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 presents some additional examples of combinational synthesis, exploiting the design methodologies presented in lectures 6.4-6.5

6.6

2

Prerequisites

- Lectures 6.4 and 6.5

6.6

3

Homework

- Students are warmly encouraged to solve the proposed exercises

6.6

4

Further readings

- No particular suggestion

6.6

5

Outline

- Address decoder
- "I gieug d' la mora" (the "morra" game)
- Keyboard encoder

6.6

6

Address decoder

Design a device to be connected to a 12-bit address bus $A[11:0]$, whose output U is asserted whenever:

$$03F < A[11:0] < 800$$

6.6

7

RT level solution

$$03F < A[11:0] < 800$$

6.6

8

RT level solution

$$03F < A[11:0] < 800$$

$A[11:0]$

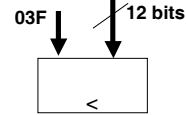
6.6

9

RT level solution

$$03F < A[11:0] < 800$$

$A[11:0]$



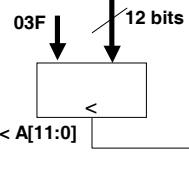
6.6

10

RT level solution

$$03F < A[11:0] < 800$$

$A[11:0]$



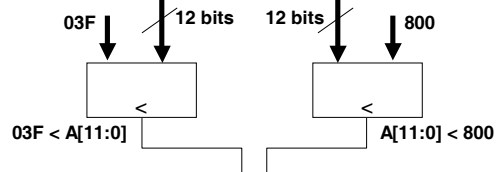
6.6

11

RT level solution

$$03F < A[11:0] < 800$$

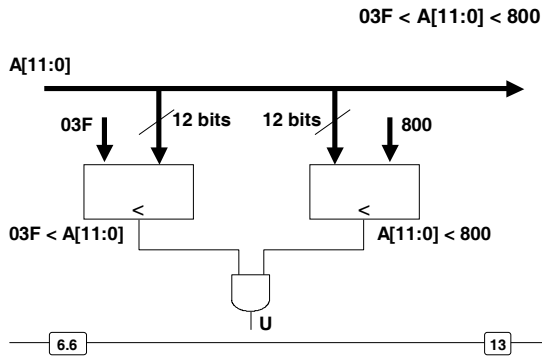
$A[11:0]$



6.6

12

RT level solution



6.6

13

Logic level solution

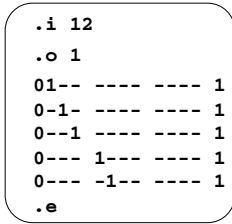
$03F < A[11:0] < 800$ is equivalent to:
 $040 \leq A[11:0] \leq 7FF$ and thereof:
 $0000\ 0100\ 0000 \leq A[11:0] \leq 0111\ 1111\ 1111$

6.6

14

Logic level solution

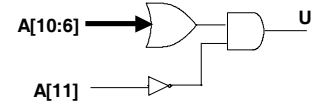
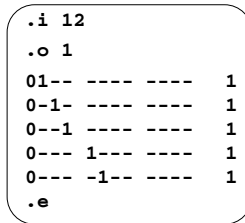
$03F < A[11:0] < 800$ is equivalent to:
 $040 \leq A[11:0] \leq 7FF$ and thereof:
 $0000\ 0100\ 0000 \leq A[11:0] \leq 0111\ 1111\ 1111$



6.6

15

Logic level solution (cont'd)



6.6

16

Outline

- Address decoder
- “l gieug d’ la mora” (The “morra” game)
- Keyboard encoder

6.6

17

The “morra” game



We have to design a circuit capable of detecting the winners in the “morra” play.

Such a play involves 2 players. Each player provides:

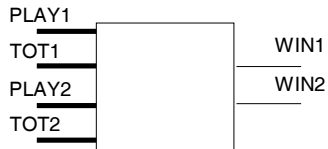
- A number in the range 0-5, called PLAY
- A number in the range 0-10, called TOT

6.6

18

The “morra” game

The circuit has 2 outputs WIN1 and WIN2, which are asserted when the player 1 or 2 is winning, respectively:



6.6

19

The “morra” game

Each player aims at guessing the value:
 $PLAY1 + PLAY2$.

Thus:

- if $TOT1 = PLAY1 + PLAY2$, player 1 wins
- if $TOT2 = PLAY1 + PLAY2$, player 2 wins

6.6

20

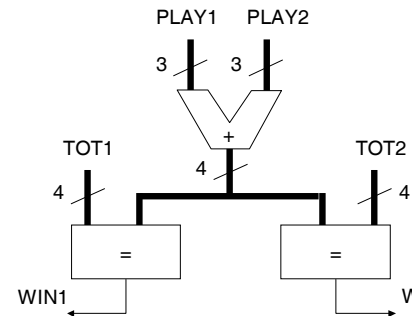
Solution

- $WIN1 = 1$ iff $TOT1 = PLAY1 + PLAY2$
- $WIN2 = 1$ iff $TOT2 = PLAY1 + PLAY2$

6.6

21

- $WIN1 = 1$ iff $TOT1 = PLAY1 + PLAY2$
- $WIN2 = 1$ iff $TOT2 = PLAY1 + PLAY2$



6.6

22

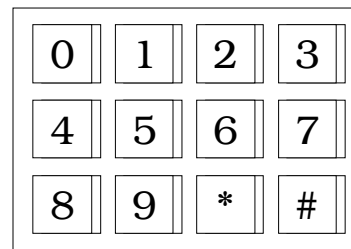
Outline

- Address decoder
- “l gieug d’ la mora” (the “morra” game)
⇒ *Keyboard encoder*

6.6

23

Keyboard encoder



6.6

24

Keyboard encoder

0	1	2	3
4	5	6	7
8	9	*	#

The decoder get in input:

- 3 signals R_1, \dots, R_3 , one for each key row
- 4 signals C_1, \dots, C_4 one for each key column

Each signal is asserted whenever a key belonging to the corresponding row (or column) is pressed

6.6

25

Keyboard encoder

0	1	2	3
4	5	6	7
8	9	*	#

The decoder provides:

- An output **VALID**, asserted whenever one (and just one) key is pressed
- An encoding, on 4 bits, of the pressed key (* and # are coded as 10 and 11, respectively).

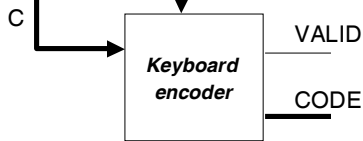
6.6

26

Keyboard encoder

0	1	2	3
4	5	6	7
8	9	*	#

0	1	2	3
4	5	6	7
8	9	*	#

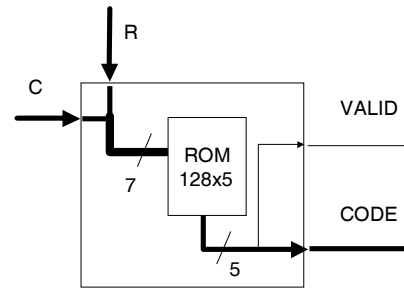


6.6

27

Solution #1 – Implementation by a ROM

0	1	2	3
4	5	6	7
8	9	*	#

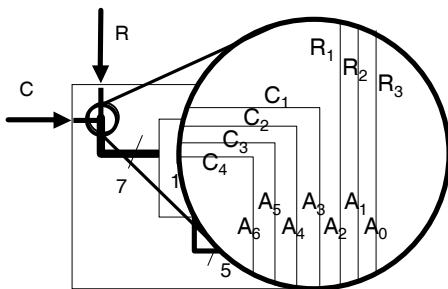


6.6

28

Solution #1 – Implementation by a ROM

0	1	2	3
4	5	6	7
8	9	*	#



6.6

29

Solution #2 – Implementation by Espresso

0	1	2	3
4	5	6	7
8	9	*	#

6.6

30

```
.i 7
.o 5
```

0	1	2	3
4	5	6	7
8	9	*	#

6.6

31

```
.i 7
.o 5
```

0	1	2	3
4	5	6	7
8	9	*	#

```
000 ---- 0 ----
--- 0000 0 ----
```



6.6

32

```
.i 7
.o 5
```

0	1	2	3
4	5	6	7
8	9	*	#

```
000 ---- 0 ----
--- 0000 0 ----

11- ---- 0 ----
-11 ---- 0 ----
1-1 ---- 0 ----
```

6.6

33

```
.i 7
.o 5
```

0	1	2	3
4	5	6	7
8	9	*	#

```
000 ---- 0 ----
--- 0000 0 ----

11- ---- 0 ----
-11 ---- 0 ----
1-1 ---- 0 ----

--- 11-- 0 ----
--- 1-1- 0 ----
--- 1--1 0 ----
--- -11- 0 ----
--- -1-1 0 ----
--- --11 0 ----
```

6.6

34

```
.i 7
.o 5
```

0	1	2	3
4	5	6	7
8	9	*	#

```
001 0001 1 0000
001 0010 1 0001
001 0100 1 0010
001 1000 1 0011

010 0001 1 0100
010 0010 1 0101
010 0100 1 0110
010 1000 1 0111

100 0001 1 1000
100 0010 1 1001
100 0100 1 1010
100 1000 1 1011

.e
```

6.6

35

```
.i 7
.o 5
.p 12
0010001 10000
0010100 10010
0010010 10001
1000001 11000
0100001 10100
0011000 10011
1000100 11010
0100100 10110
1000010 11001
0100010 10101
1001000 11011
0101000 10111

.e
```

6.6

36

**Solution #3 –
RT level implementation**

Let's design the portion related to the generation of the VALID output first.

0	1	2	3
4	5	6	7
8	9	*	#

6.6

37

**Solution #3 –
RT level implementation**

Let's design the portion related to the generation of the VALID output first.

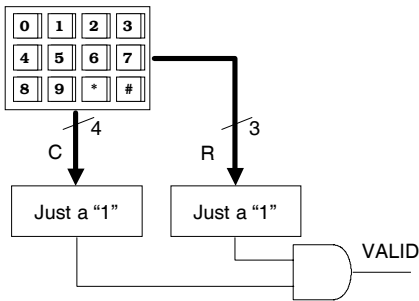
VALID=1 iff:

- just a 1 on C_1, C_2, C_3, C_4 &
- just a 1 on R_1, R_2, R_3

0	1	2	3
4	5	6	7
8	9	*	#

6.6

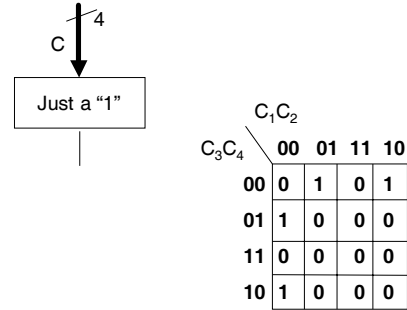
38



0	1	2	3
4	5	6	7
8	9	*	#

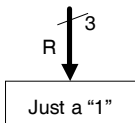
6.6

39



6.6

40



	$R_1 R_2$			
R_3	00	01	11	10
0	0	1	0	1
1	1	0	0	0

0	1	2	3
4	5	6	7
8	9	*	#

6.6

41

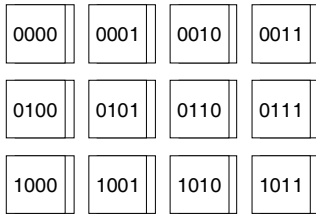
Let's now target the generation of the CODE output.

0	1	2	3
4	5	6	7
8	9	*	#

6.6

42

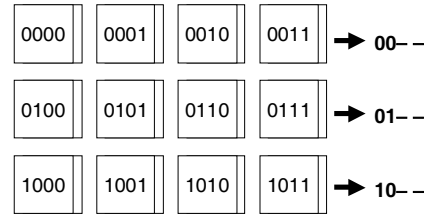
Let's now target the generation of the CODE output.



6.6

43

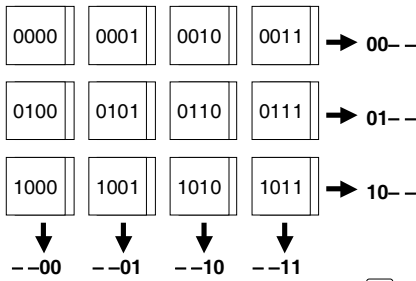
Let's now target the generation of the CODE output.



6.6

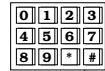
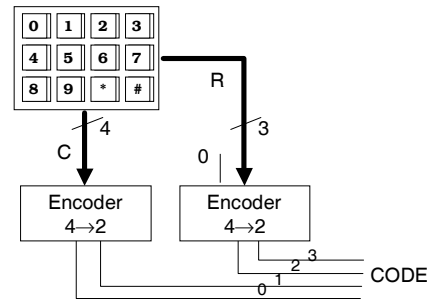
44

Let's now target the generation of the CODE output.



6.6

45



6.6

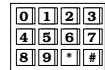
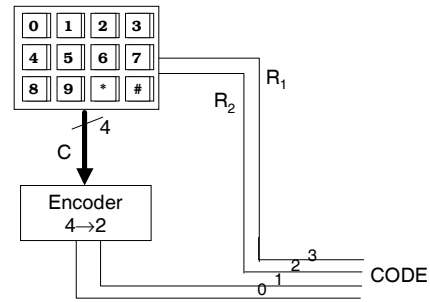
46

Optimization

The row encoder behaves as follows:

R ₁	R ₂	R ₃	outputs
0	0	1	00
0	1	0	01
1	0	0	10

It can thus be avoided by resorting to the following connection:



6.6

47

6.6

48

