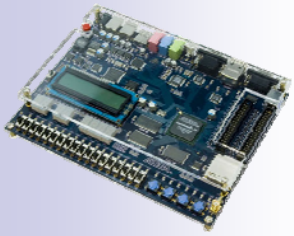
Struktury počítačových systémů



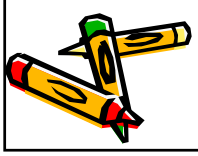
Přednáší: Richard Šusta

13. října 2011 verze 1.0

ČVUT-FEL v Praze – kód předmětu A0B35SPS

K-Map Example

- Synchronous divider/counter by 5



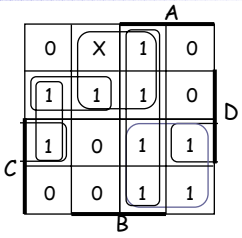
K-maps

- Implicants -

Two-level K-map simplification

- Implicant
  - it covers "1"s
- Prime implicant
  - larger as possible
- Essential prime implicant
  - it alone covers a "1"
- Method:
  - Grow implicants into prime implicants
  - Cover all "1" (minimize number of product terms)

Example of Implicants



6 prime implicants:
A'B'D, BC', AC, A'C'D, AB, B'CD

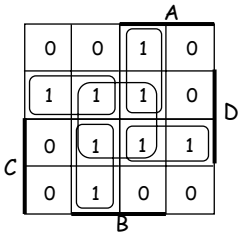
essential

minimum cover: AC + BC' + A'B'D

5 prime implicants:
BD, ABC', ACD, A'BC, A'C'D

essential

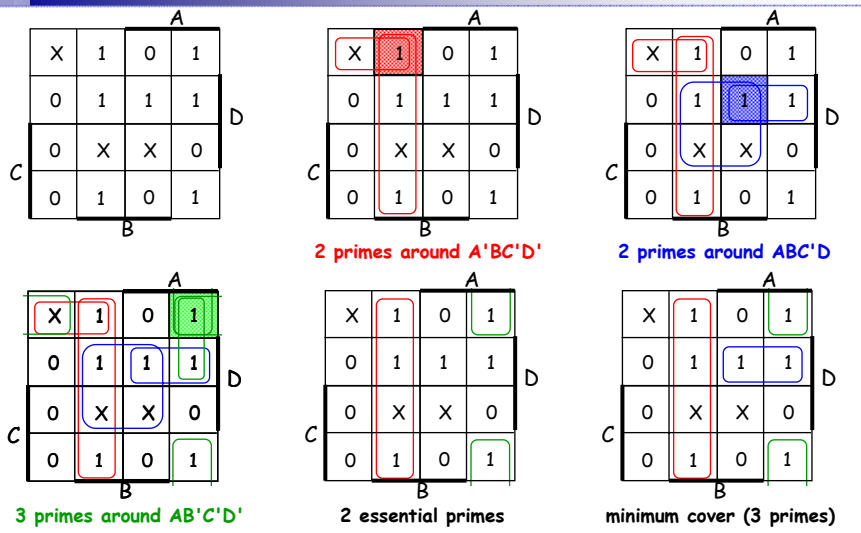
minimum cover: 4 essential implicants



Algorithm for two-level simplification

- K-map algorithm
  - Step 1: choose a "1"
  - Step 2: maximize covering implicants to prime implicants *(number of elements must be a power of 2)*
  - Repeat Steps 1 and 2 until finding all prime implicants
  - Step 3:
    - select essential implicants
  - Step 4:
    - cover remaining "1" by the smallest number of prime implicants

f(A,B,C,D) = m(4,5,6,8,9,10,13) + d(0,7,15)



[Seungryoul M.: Combination Logic, KAIST 2002]

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

f(A,B,C,D) = Σm(4,5,6,8,9,10,13) + Σd(0,7,15)

Stage 1: Find all prime implicants

Step 1: Fill Column 1 with ON-set and DC-set minterm indices. Group by number of 1's.

Step 2: Apply Uniting Theorem

E.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Implication Table

Column I	Column II	Column III
0000♠	0_00*	01__*
	_000*	
0100♠		_1_1*
1000♠	010_♠	
	01_0♠	
0101♠	100_*	
0110♠	10_0*	
1001♠		
1010♠	01_1♠	
	_101♠	
0111♠	011_♠	
1101♠	1_01*	
1111♠	_111♠	
	11_1♠	

Repeat until no further combinations can be made.

Stage2: Cover ON-set

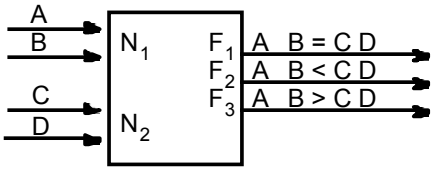
[Seungryoul M.: Combination Logic, KAIST 2002]

Summary

- Boolean Algebra provides framework for logic simplification
- De Morgans transforms between gate types
- Uniting to reduce minterms
- K-maps provide visual notion of simplifications

Gate Logic: Two-Level Simplification

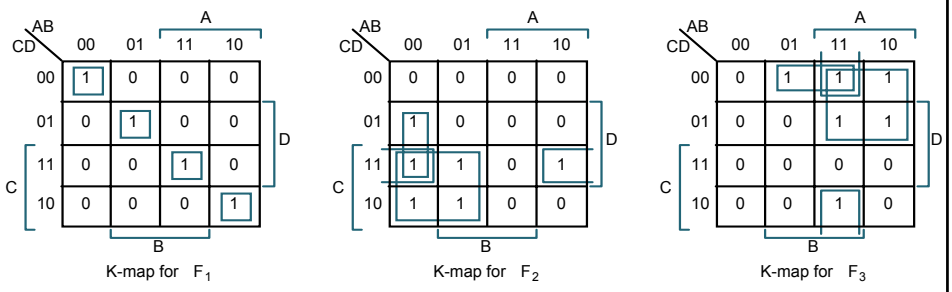
Block Diagram



Truth Table

A	B	C	D	F_1	F_2	F_3
0	0	0	0	1	0	0
		0	1	0	1	0
		1	0	0	1	0
		1	1	0	1	0
0	1	0	0	0	0	1
		0	1	1	0	0
		1	0	0	1	0
		1	1	0	1	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	1	0	0
		1	1	0	1	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	1	0	0

Gate Logic:Two-Level Simplification



F1 = A' B' C' D' + A' B C' D + A B C D + A B' C D'

F2 = A' B' D + A' C

F3 = B C' D' + A C' + A B D'

F1 = F2' . F3'

Skupinová minimalizace

- Group Minimization -

BCD-to-Seven-Segment Decoder

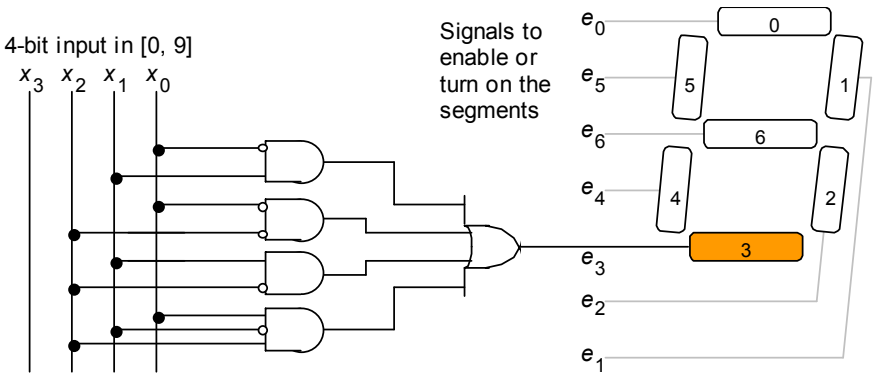
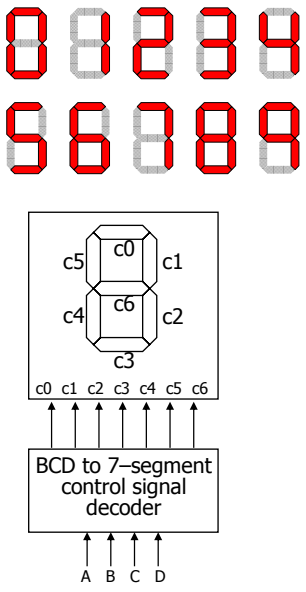


Figure 1.8 The logic circuit that generates the enable signal for the lowermost segment (number 3) in a seven-segment display unit.

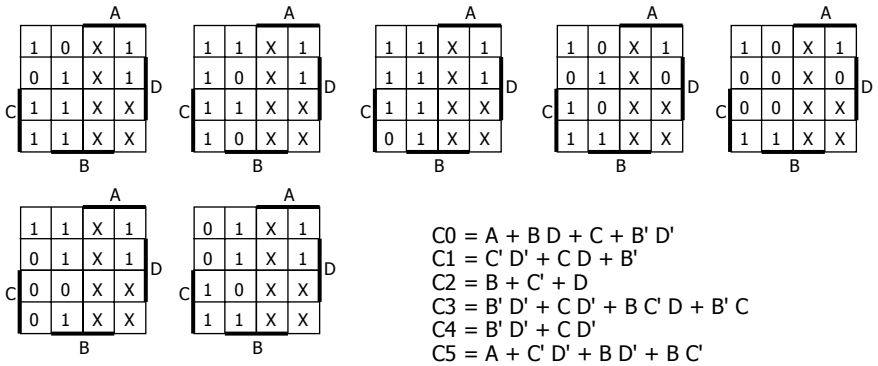
BCD / 7-segment



A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

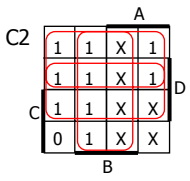
Karnaughovy mapy

- Implementace přes Karnaughovy mapy



C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

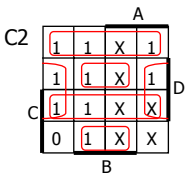
Skupinová minimalizace



C0 = A + B D + C + B' D'
C1 = B' + C' D' + C D
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

9 mintermů

+ 6 sdílených



C0 = A +B C' D + C D + B' D' + B C D'
C1 = B' D + C' D' + C D + B' D'
C2 = B' D + B C' D + C' D' + C D + B C D'
C3 = B' D' + B C D'+B C' D + B' D
C4 = B' D' + B C D'
C5 = A + C' D' + B C D' + B C' D
C6 = A + B C D' + B' C + B C'

2 mintermy v C6

+ 6 sdílených

Závěr

- Karnaughovy mapy představovaly skvělý nástroj, když lidé prováděli zjednodušování logických funkcí
- Dnes sice zjednodušují počítače,...
- ...ale Karnaughovy mapy stále dovedou dát náhled na chování funkce a kvůli tomu se dodnes používají

Odkaz:

M. Karnaugh, "The Map Method for Synthesis of Combinatorial Logic Circuits", *Transactions of the American Institute of Electrical Engineers, Communications and Electronics*, Vol. 72, pp. 593-599, November 1953.

Hazard

Limits in Design
of Sequential Circuits

Hazard

hazard (noun) anglická výslovnost: haz(r)d

hazard {hra založená jen na náhodě, riskantní čin},
hazardní, hazardér, hazardovat, prohazardovat, zahazardovat si.

Staročeština: *hazart* {druh hry v kostky}.
Přes **němčinu** ze **starofrancouštiny** *hasart*
do ní ze **španělštiny** *azar,*
a to z **arabštiny** *az-zahr* {hra v kostky}.
*Nynější významy se vyvinuly ve starší **francouštině** a přišly k nám opět přes **němčinu***

Ve výpočetní technice: nežádoucí výstupní signál logického prvku vzniklý nepřesnou synchronizací mezi vstupními signály

LEDA etymologický slovník jazyka českého

Circuit's Behavior

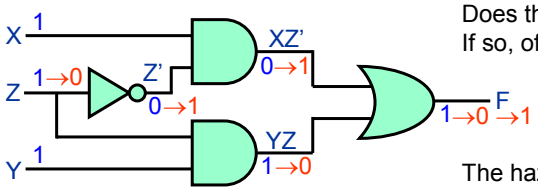
The **steady-state behavior** of a circuit is the value of the output after the inputs have been stable for a long time.

The **transient behavior** of a circuit is the value of the output while (or soon after) the inputs change.

The **glitch** is a (often undesirable) short pulse produced in the output during a transient phase.

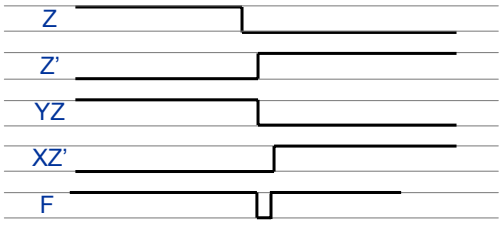
If a circuit has the possibility of producing a glitch, **the circuit has a hazard**.

Example



Does this circuit have a hazard?
If so, of what kind?

The hazard occurs in the transition from X,Y,Z = 111 to X,Y,Z = 110



static-1 hazard

Static-1 Hazard

A **static-1 hazard** is a set of two input combinations X_a and X_b such that:

(i) X_a and X_b differ in only one input variable;

(ii) both X_a and X_b produce a 1 output;

but it is possible for a momentary 0 to appear in the output when the input transits from X_a to X_b or from X_b to X_a

i.e., a static-1 hazard is a possibility of a 0 glitch when we expect a steady 1 output.

Static-0 Hazard

A **static-0 hazard** is a set of two input combinations X_a and X_b such that:

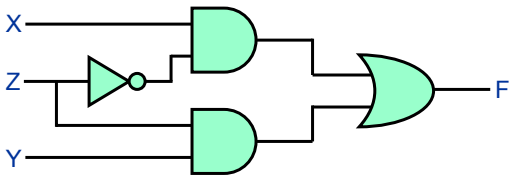
(i) X_a and X_b differ in only one input variable;

(ii) both X_a and X_b produce a 0 output;

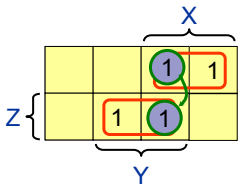
but it is possible for a momentary 1 to appear in the output when the input transits from X_a to X_b or from X_b to X_a

i.e., a static-0 is a possibility of a 1 glitch when we expect a steady 0 output.

Static Hazards in Karnaugh Maps



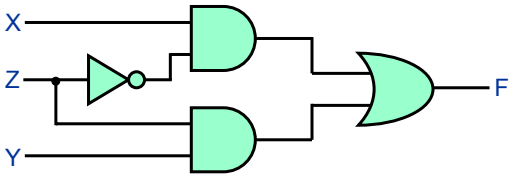
F = X•Z' + Y•Z



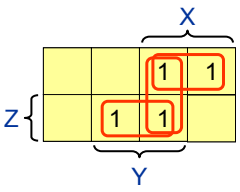
How can we identify a static-1 hazard in this Karnaugh map?

Two adjacent 1's that are not in the same term cause a static-1 hazard.

Static Hazards in Karnaugh Maps



$F = X \cdot Z' + Y \cdot Z$



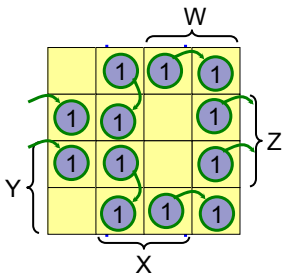
How can we eliminate the hazard?

We can add one extra term to F.

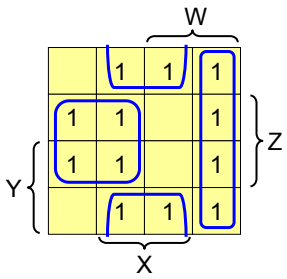
$F = X \cdot Z' + Y \cdot Z + X \cdot Y$

Consensus Term

Static Hazard - Another Example 1/2



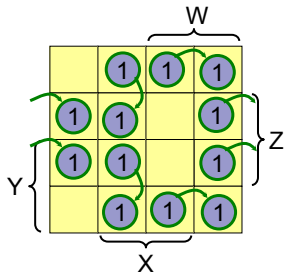
$F = W' \cdot Z + X \cdot Z' + X' \cdot W$



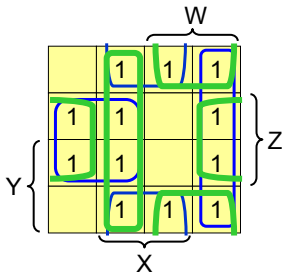
$F = W' \cdot Z + X \cdot Z' + X' \cdot W$

1. Write minimal form for F
2. Identify static-1 hazards
3. Eliminate static-1 hazards

Static Hazard - Example 2/2



F = W'•Z + X•Z' + X'•W



F = W'•Z + X•Z' + X'•W + X•W' + W•Z' + X'•Z

1. Write minimal form for F
2. Identify static-1 hazards
3. Eliminate static-1 hazards

Dynamic Hazards

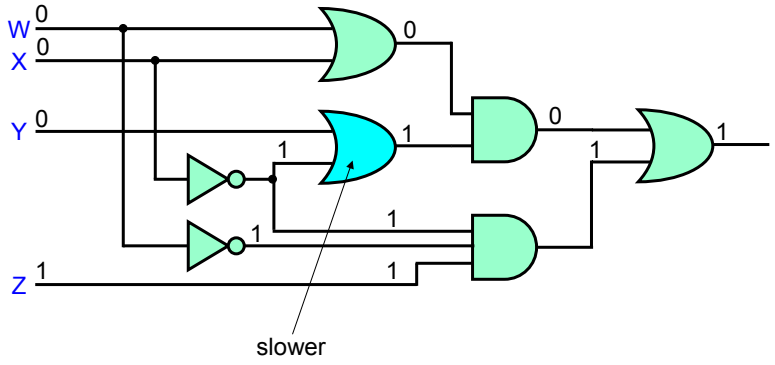
A **dynamic hazard** is the possibility of an output changing more than once as the result of a single transition.

Dynamic hazards exist when there are multiple paths with different delays from the changing input to the changing output.

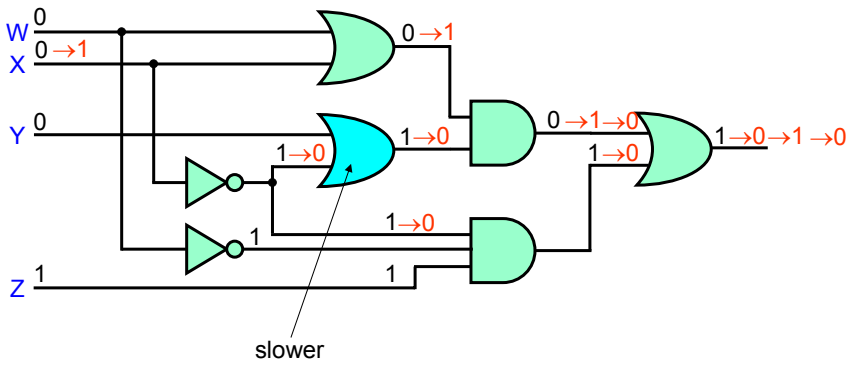
Dynamic hazards do not occur in properly designed two level AND-OR or OR-AND circuits.

PS: *A two level AND-OR or OR-AND circuit is properly design if a variable and its complement are never input to the same first level gate.*

Dynamic Hazard - Example 1/2



Dynamic Hazard - Example 2/2



A dynamic hazard occurs when oscilation may occur when a single transition is expected.

Závěr

- Kdy nám hazard vadí?

-> *pokud následující obvod dokáže zpracovat i krátký signál (třeba hodinový vstup), musíme odrušit hazardy synchronizací - blíže další přednášky*

LUT – Look Up Table
- multiplexory v FPGA

Shannon (Boole) Cofactors

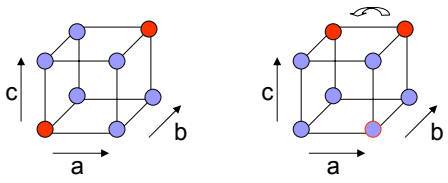
Let $f : B^n \rightarrow B$ be a Boolean function, and $x= (x_1, x_2, \ldots, x_n)$ the variables in the support of f.

The cofactor f_a of f by a literal $a=x_i$ or $a=\overline{x_i}$ is

$$f_{x_i}(x_1, x_2, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$
$$f_{\overline{x_i}}(x_1, x_2, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

Example:

$f = \overline{a}\,\overline{b}\,\overline{c} + abc$ $\qquad$ $f_a = bc$



Warning: The computation of the cofactor is a fundamental operation in Boolean reasoning !!!! and one of my fundamental questions for exams:-)

Shannon Expansion

$$f : B^n \rightarrow B$$

Shannon Expansion:

$$f = x_i f_{x_i} + \overline{x}_i f_{\overline{x}_i}$$

Theorem: F is a cover of f. Then

$$\tilde{F} \equiv x_i F_{x_i} + \overline{x}_i F_{\overline{x}_i}$$

We say that f (F) is **expanded** about x_i.

x_i is called the **splitting** variable.

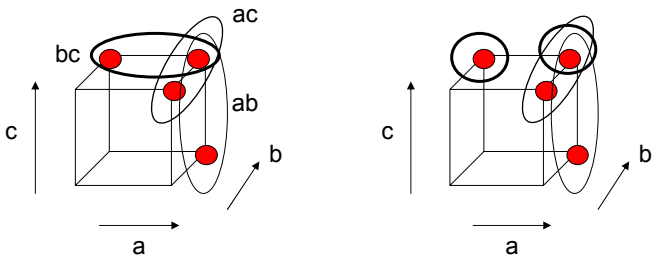
Shannon Expansion (cont.)

Example

$$F = ab + ac + bc$$

$$\tilde{F} = aF_a + \bar{a}F_{\bar{a}} = a(b + c + bc) + \bar{a}(bc)$$

$$= ab + ac + abc + \bar{a}bc$$



Cube bc got split into two cubes

Example

$$f(x_1, x_2, \ldots, x_n) = x_i.f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) + \overline{x}_i.f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots x_n)$$

f

x4 x3 \ x2 x1	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	0	0	0	1
10	0	1	1	0

g

x4 x3 \ x2	0	1
00	0	0
01	0	0
11	0	0
10	1	1

h

x4 x3 \ x2	0	1
00	1	1
01	0	1
11	0	1
10	0	0

$$f = Fg + \bar{F}h$$

$$F(x_i) = x_1$$

$$g = \bar{x}_3 x_4$$

$$h = (\bar{x}_3 \bar{x}_4 + x_3 x_2)$$

Příklad 1/2

…rozložte funkci na jednodušší

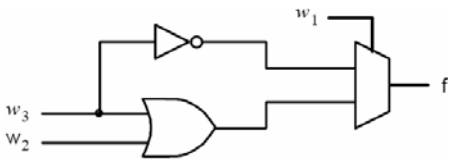
- $f(x_1, x_2,\ldots x_n) = x_1'.f_a(0,x_2,\ldots,x_n) + x_1.f_b(1,x_2,\ldots,x_n)$

$f(w_1,w_2,w_3) = w_1w_2 + w_1w_3 + w_2w_3$ *(3 vstupová majorita)*

$= w_1'(w_2w_3) + w_1(w_2 + w_3)$

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	w_2+w_3

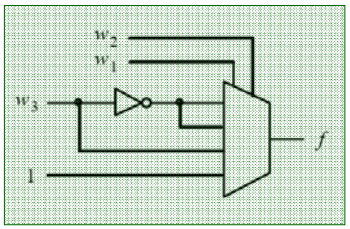


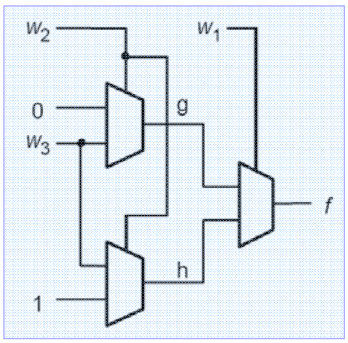
Rozpůlení Booleovské krychle podle 1 proměnné

Příklad 2/2

$f(w_1,w_2,w_3) = w_1'(w_2w_3) + w_1(w_2 + w_3) = w_1'g + w_1'h$

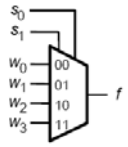
$= w_1'w_2'(w_3') + w_1'w_2(w_3') + w_1w_2'(w_3) + w_1w_2(w_3' + w_3)$





Lze snížit složitost funkce tak, aby se dala realizovat dostupnými multiplexory, tj LUT

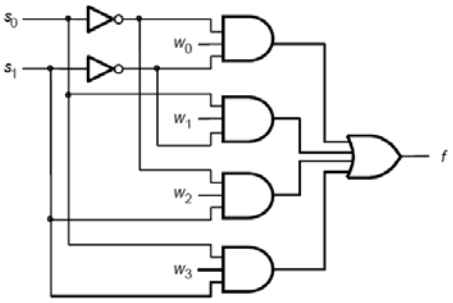
4 bitový multiplexer



Schematická značka v FPGA

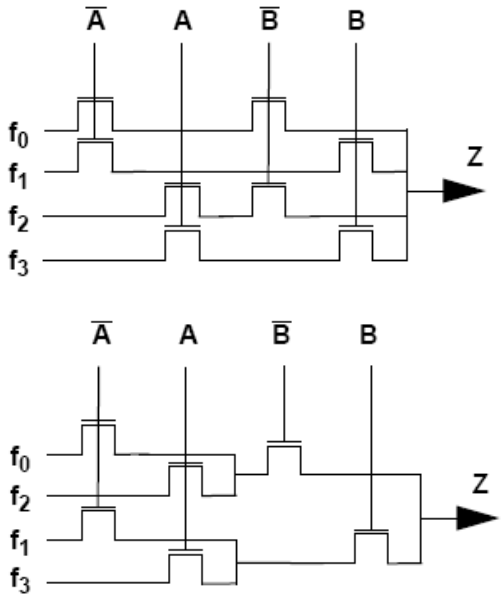
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

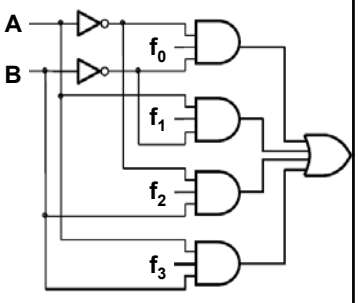
Pravdivostní tabulka



Zapojení

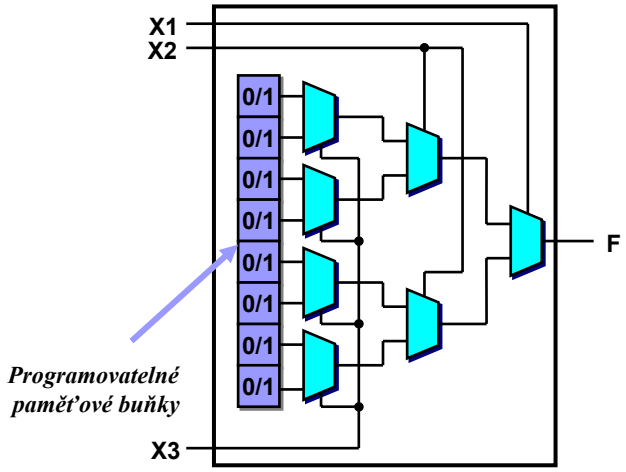
Některé ASIC implementace multiplexoru





Zapojení z hradel

Příklad implementace logické funkce



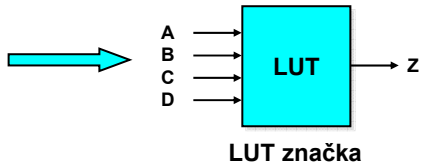
LUT – Look Up Table

LUT - Look Up Tables

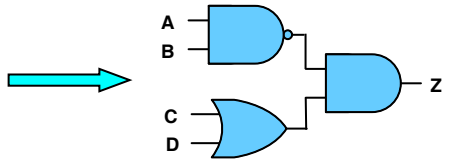
- N-vstupová LUT umožňuje implementovat kombinační logiku
- LUT se programuje pomocí pravdivostní tabulky

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0

Pravdivostní tabulka



LUT značka

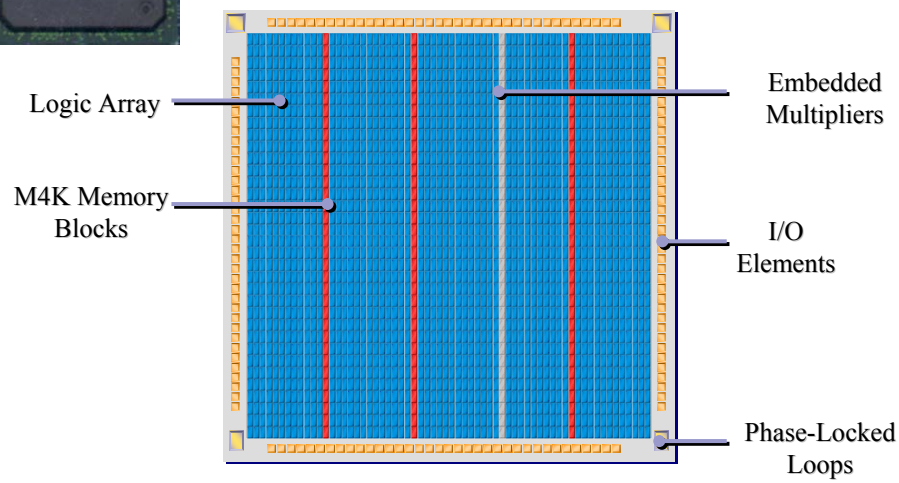


Implementace z hradel

Více o RTL syntéze

v Cyclone II

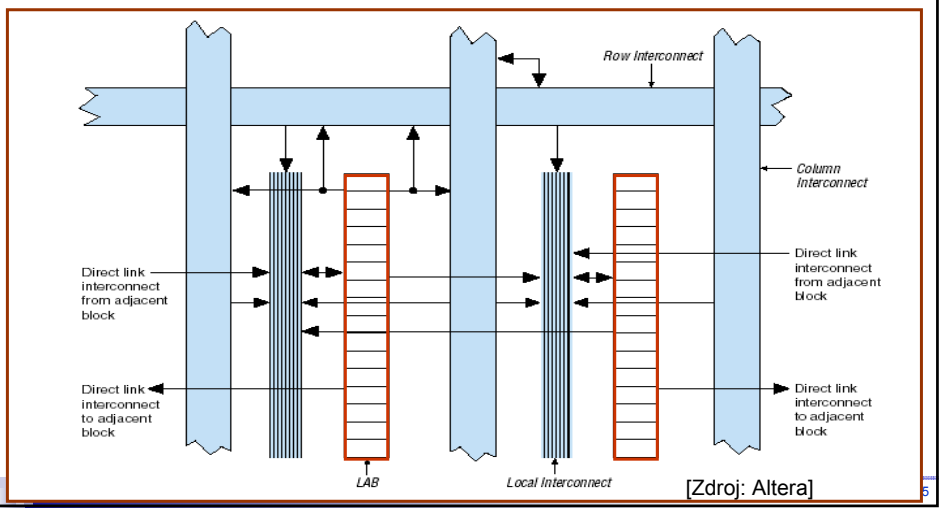
Cyclone II Internals



[Zdroj: Altera]

Cyclone II Logic Array

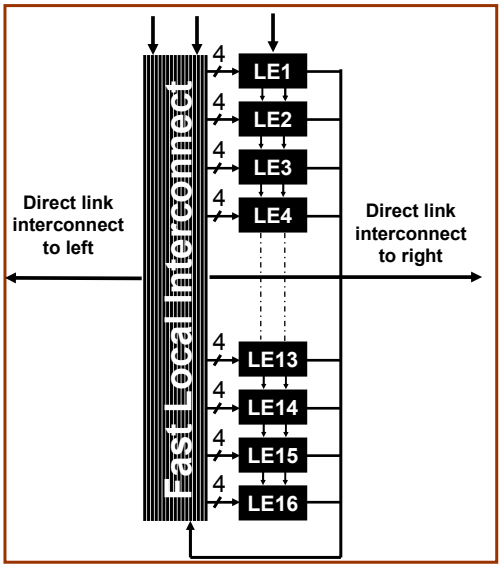
- Build of LABs (logic array blocks) and reconfigurable interconnect



[Zdroj: Altera]

Cyclone II Logic Array Block (LAB)

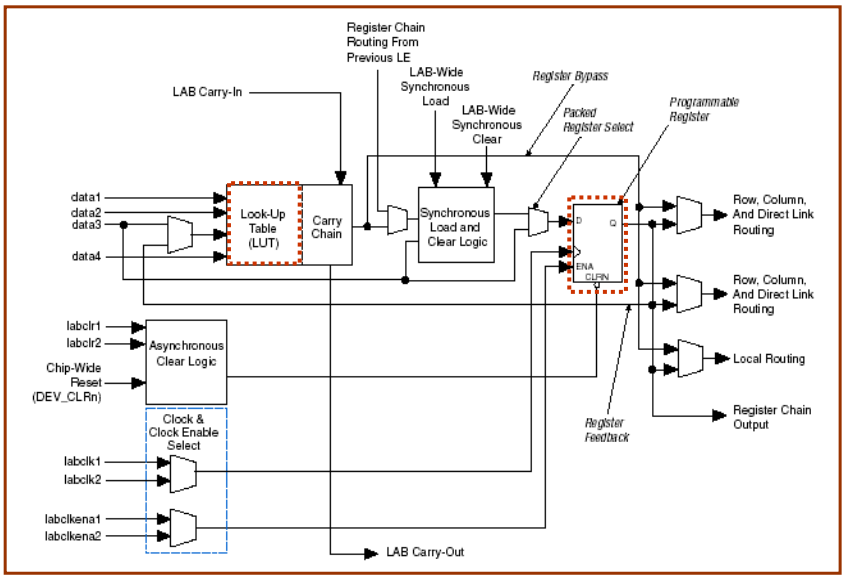
- 16 LEs
- Local Interconnect
- LE carry chains
- Register chains
- LAB Control Signals
  - 2 CLK
  - 2 CLK ENA
  - 2 ACLR
  - 1 SCLR
  - 1 SLOAD



[www.altera.com]

[Zdroj: Altera]

Cyclone II Logic Element (LE)



[Zdroj: Altera]

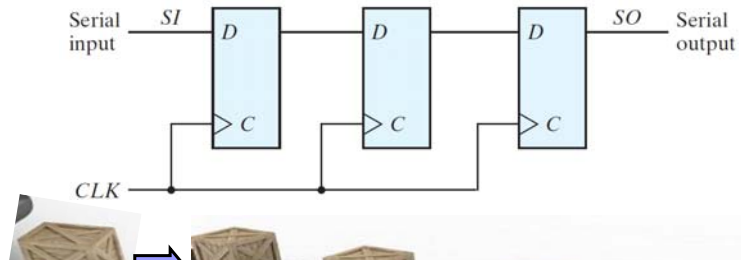
Závěr

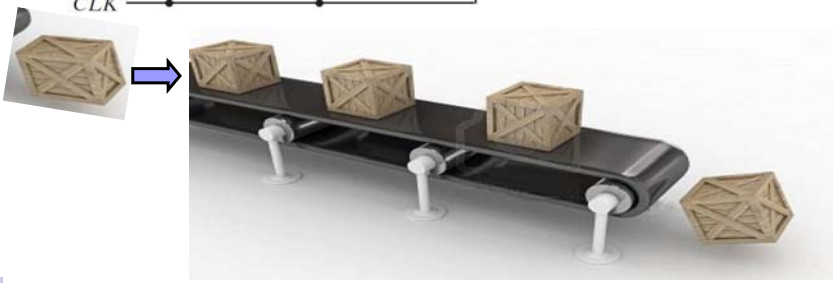
- V FPGA se logické funkce generují přes tabulky –> nelze navrhnout funkci, která by negenerovala hazard
- Obvody prosté hazardu lze navrhnout pouze v semi-programovatelných nebo uživatelských ASIC
- V FPGA se doporučuje nevést výstupy z kombinačních obvodů na hodinové vstupy - je to riskantní a neefektivní

Shift Registers

Shift Register

- shifting the binary information held in each cell to its neighboring cell...



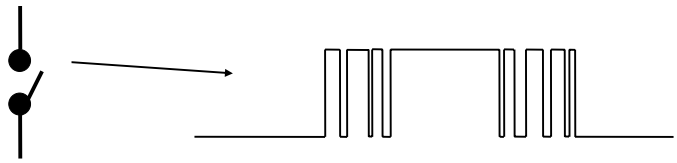


Debouncing Switches

What is switch bounce?

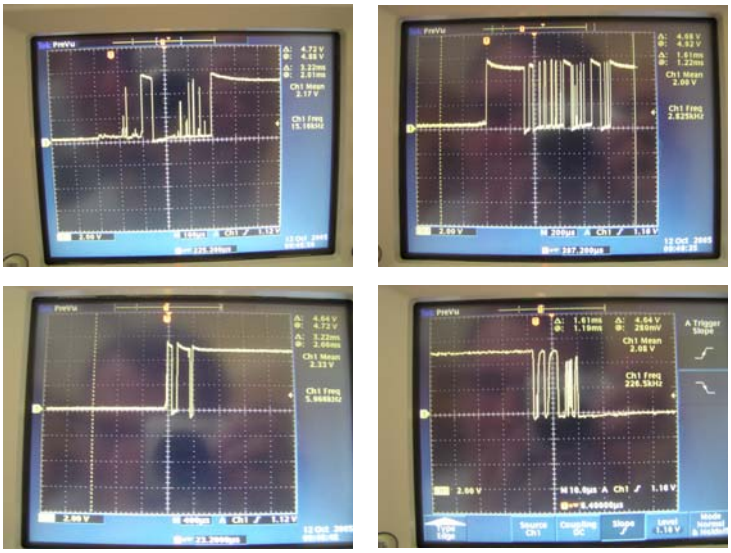
-The non-ideal behavior of the contacts that creates multiple electrical transitions for a single user input.

Cz: bounce = odrazit, odrážet se - v logice se překládá jako zakmitávání, debounce switch = odrušené či ošetřené tlačítko



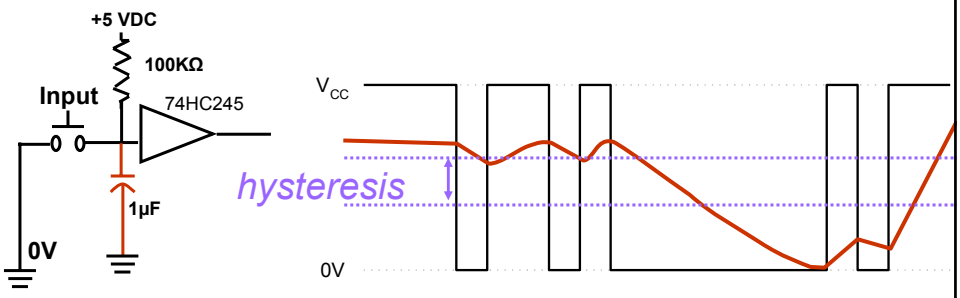
Debouncing Switches

Switch bounce from a single depress/release of switches



[Source: L. Traylor: Computer Techniques, Oregon 2009]

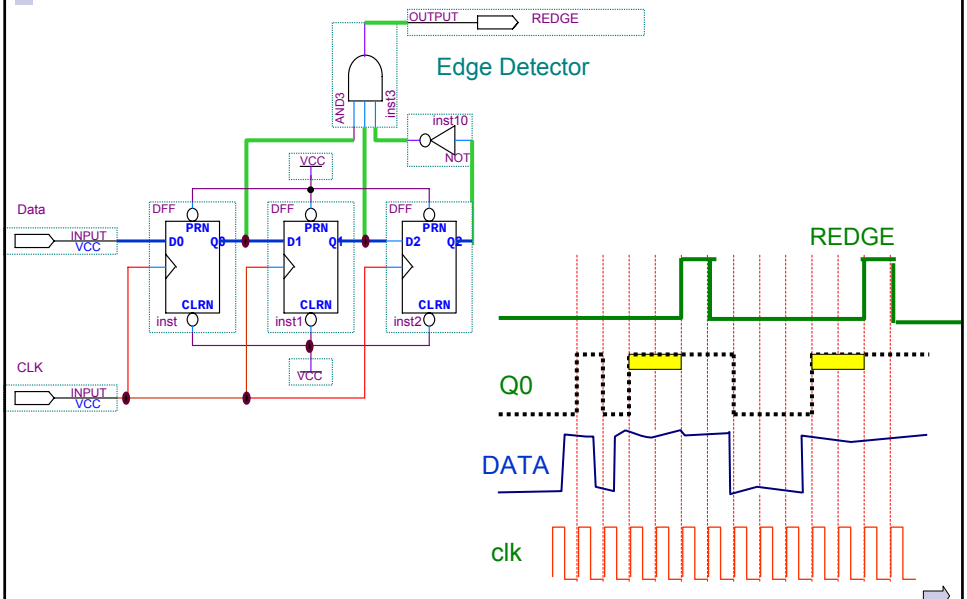
Simple Debouncing DE2 - KEY[3..0]

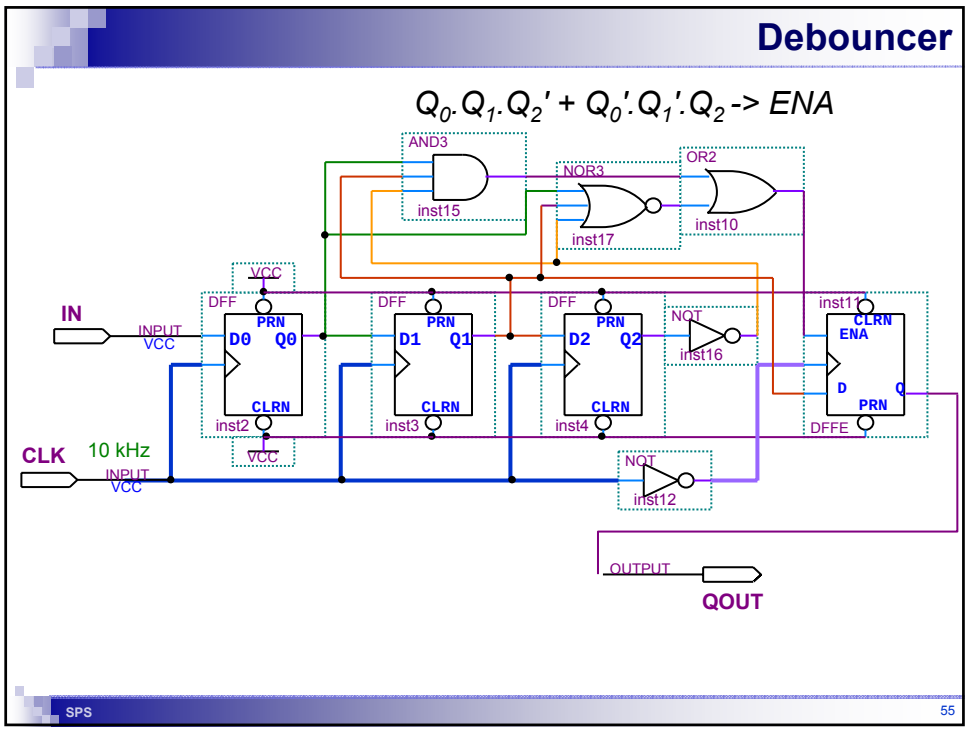


- RC Integrator debouncer and initiator...

After DE2 board is switched on KEY[3..0] will hold "0" state for several milliseconds until their capacitors are charged.
KEY inputs are useful for automatic initialization of your logic!

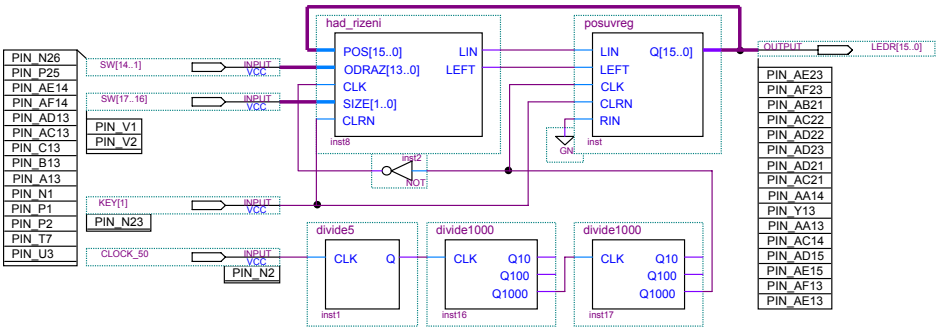
Flip-flop with 2bit Debouncer 1/2




Debouncer
$Q_0.Q_1.Q_2' + Q_0'.Q_1'.Q_2$ -> ENA
AND3
inst15
NOR3
inst17
OR2
inst10
VCC
IN
INPUT
VCC
DFF
PRN
D0
Q0
CLRN
inst2
D1
Q1
inst3
D2
Q2
inst4
NOT
inst16
inst11
CLRN
ENA
D
Q
PRN
DFFE
CLK
10 kHz
INPUT
VCC
NOT
inst12
OUTPUT
QOUT
SPS
55


How to make snake?

LED Snake



- 60LE total 4bit snake
  - 33LE frequency divider
  - 14LE snake with variable length from 1 to 4 bits
  - 13LE logic for bounding from switches