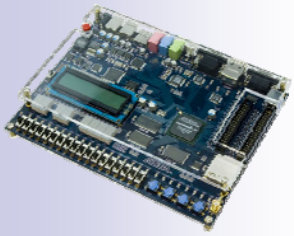
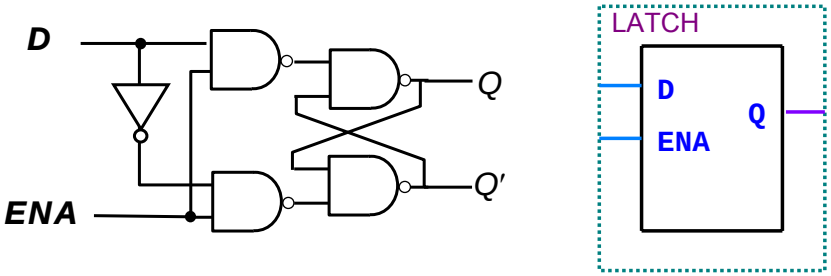
Struktury počítačových systémů

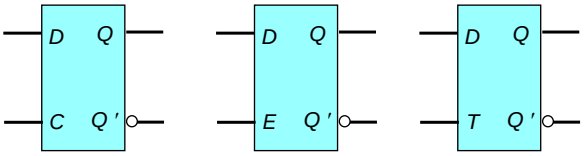
Přednáší: Richard Šusta



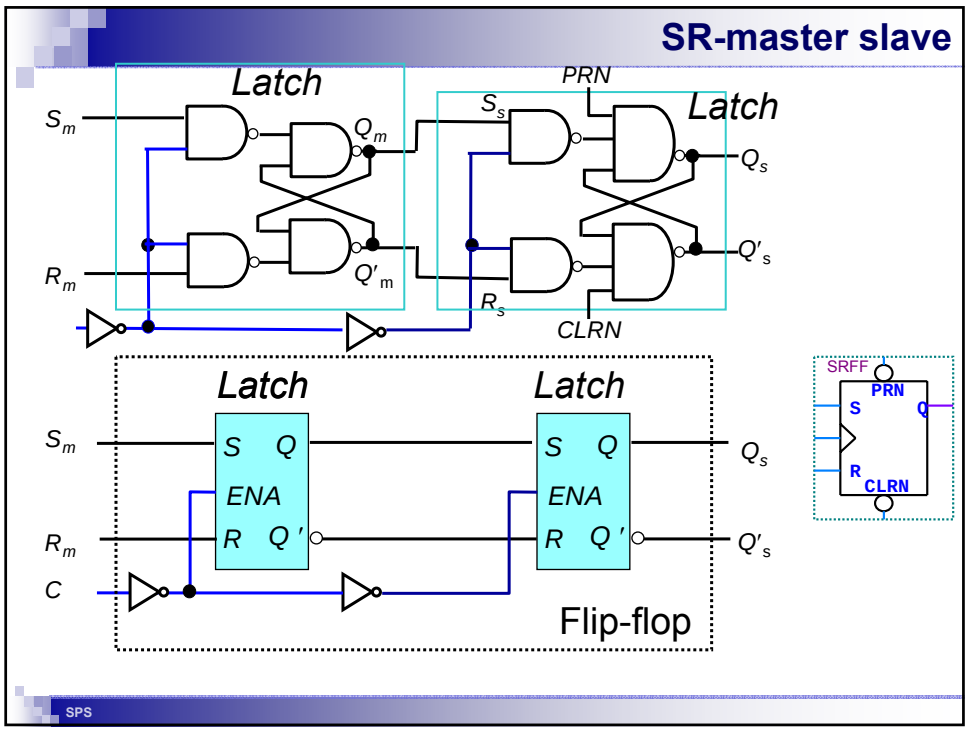
ČVUT-FEL v Praze – kód předmětu A0B35SPS

Quartus Latch





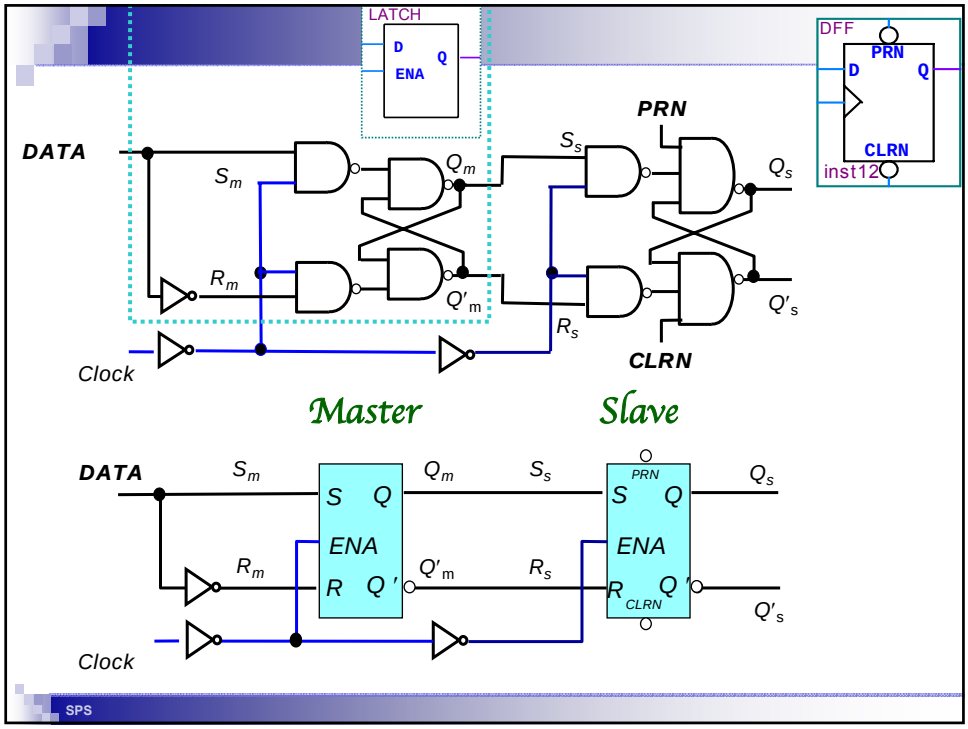
Alternatives in schematics

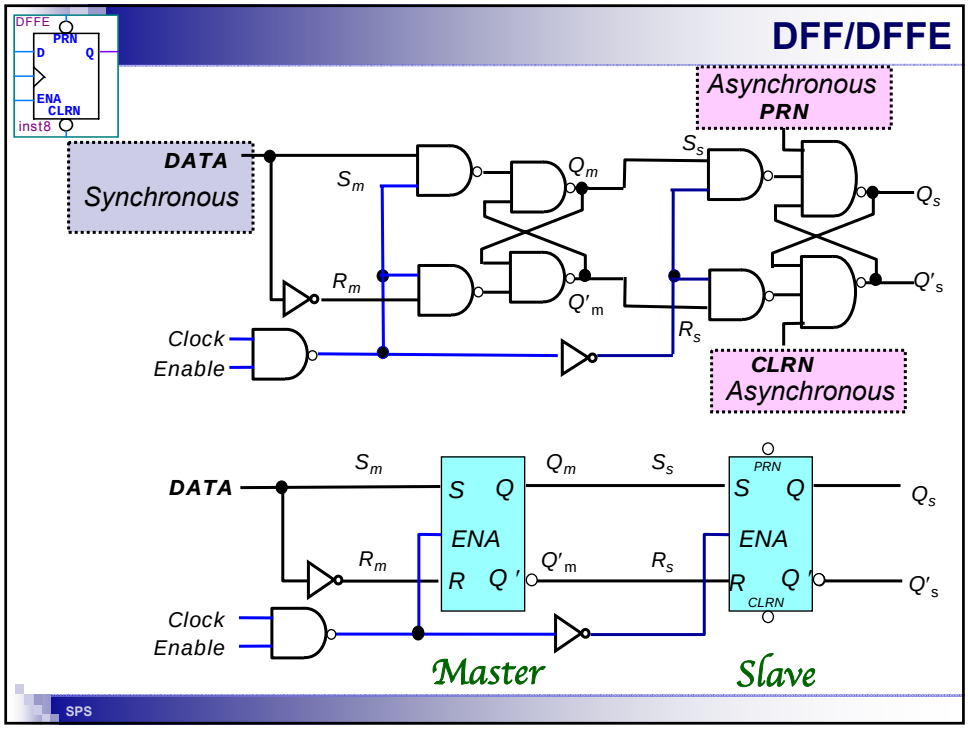


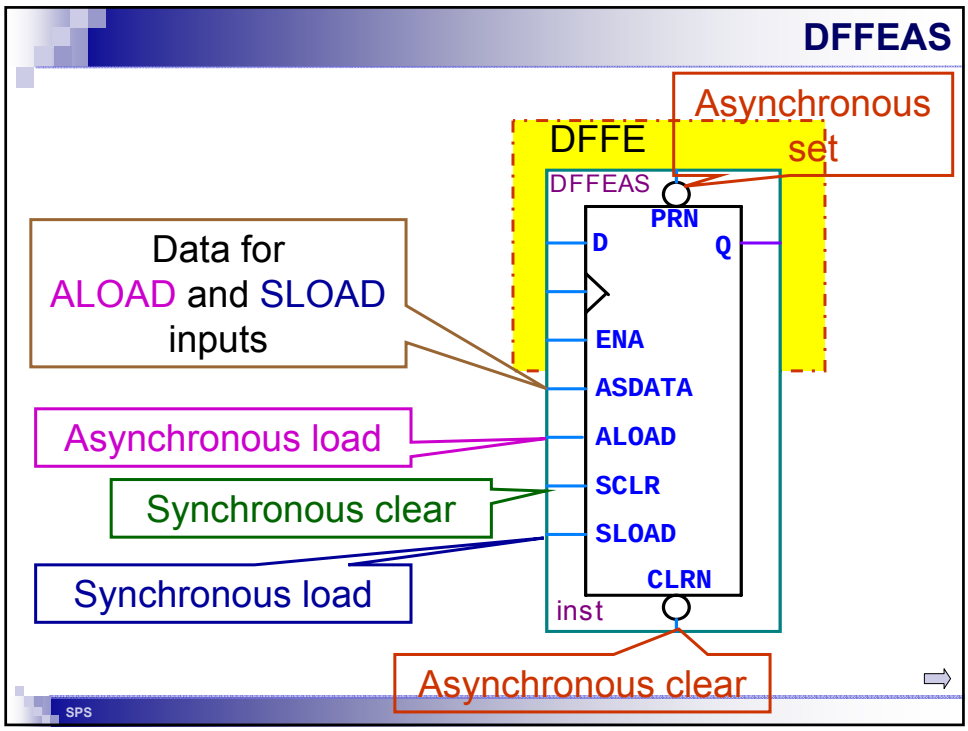
Latch – Flip-flop v angličtině

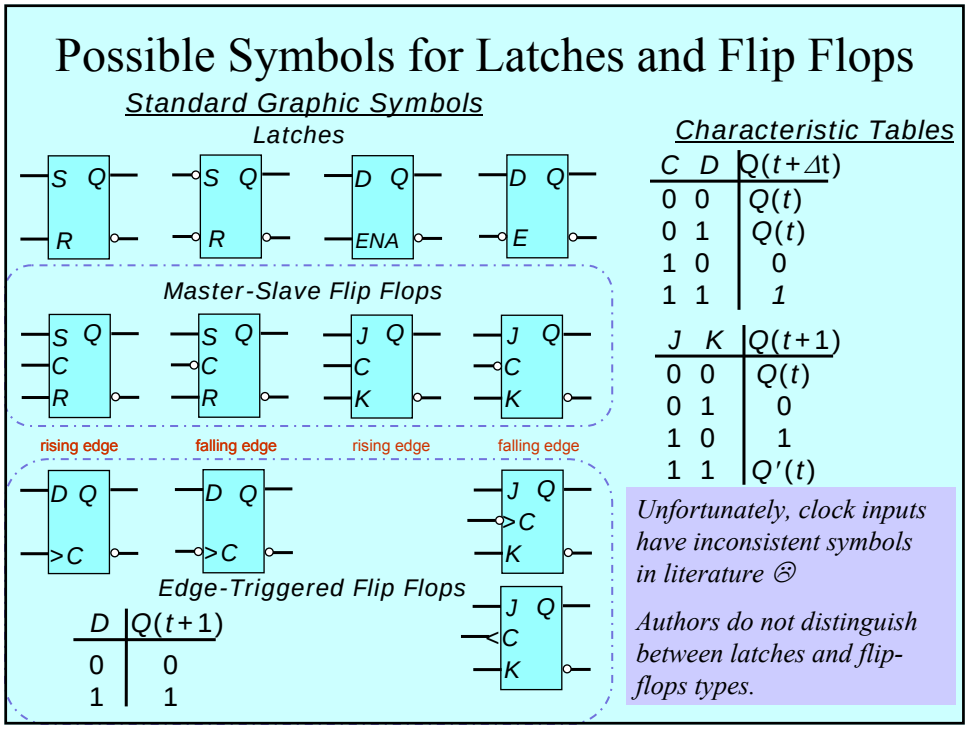
- **Latch** – *někdy i jako* "**level triggered flip-flop**"
  (*původně petlice, závora, západka -> západkové relé, první elektrický paměťový prvek*)
  = jednobitová paměť bez hodinového signálu,
  český termín: úrovňový klopný obvod,
  přehled: http://en.wikipedia.org/wiki/Latch_(electronics)

- **Flip-flop** – *(původně přemet nazad, obrat o 180 stupňů)*
  = klopný obvod řízený hodinovým signálem,
  přehled: http://en.wikipedia.org/wiki/Flip-flop_(electronics)
  Pozn. Odpovídající česká wiki-stránka nerozlišuje mezi latch a flip-flop


LATCH
D
ENA
Q
DFF
PRN
D
Q
CLRN
inst12
DATA
PRN
S_m
Q_m
S_s
Q_s
R_m
Q'_m
R_s
Q'_s
CLRN
Clock
Master
Slave
S
Q
ENA
R
Q '
CLRN
SPS


DFFE
PRN
D
Q
ENA
CLRN
inst8
DFF/DFFE
Asynchronous
PRN
DATA
Synchronous
S_m
Q_m
S_s
Q_s
R_m
Q'_m
Q'_s
R_s
Clock
Enable
CLRN
Asynchronous
DATA
S_m
Q_m
S_s
PRN
Q_s
R_m
Q'_m
R_s
Q'_s
Clock
Enable
S
Q
ENA
R
Q '
CLRN
Master
Slave
SPS

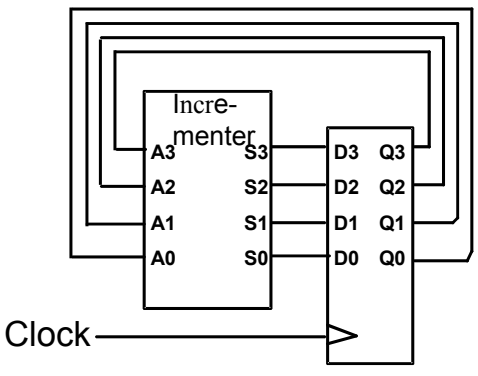

DFFEAS
Asynchronous set
DFFE
DFFEAS
PRN
D
Q
ENA
ASDATA
ALOAD
SCLR
SLOAD
CLRN
inst
Data for ALOAD and SLOAD inputs
Asynchronous load
Synchronous clear
Synchronous load
Asynchronous clear
SPS


Possible Symbols for Latches and Flip Flops
Standard Graphic Symbols
Latches
Master-Slave Flip Flops
rising edge
falling edge
rising edge
falling edge
Edge-Triggered Flip Flops
D Q(t+1)
0 0
1 1
Characteristic Tables
C D Q(t+Δt)
0 0 Q(t)
0 1 Q(t)
1 0 0
1 1 1
J K Q(t+1)
0 0 Q(t)
0 1 0
1 0 1
1 1 Q'(t)
Unfortunately, clock inputs have inconsistent symbols in literature ☹
Authors do not distinguish between latches and flip-flops types.

Synchronous Counters

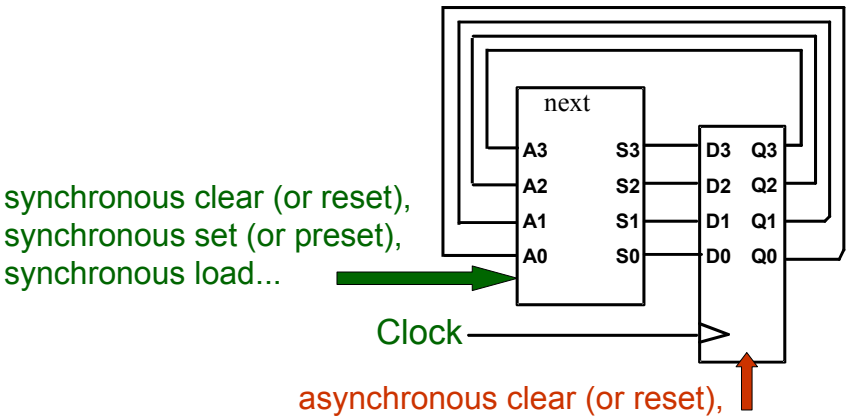
Synchronous Counters

- To eliminate the "ripple" effects, use a common clock for each flip-flop and a combinational circuit to generate the next state.
- For an up-counter, use an incrementer =>

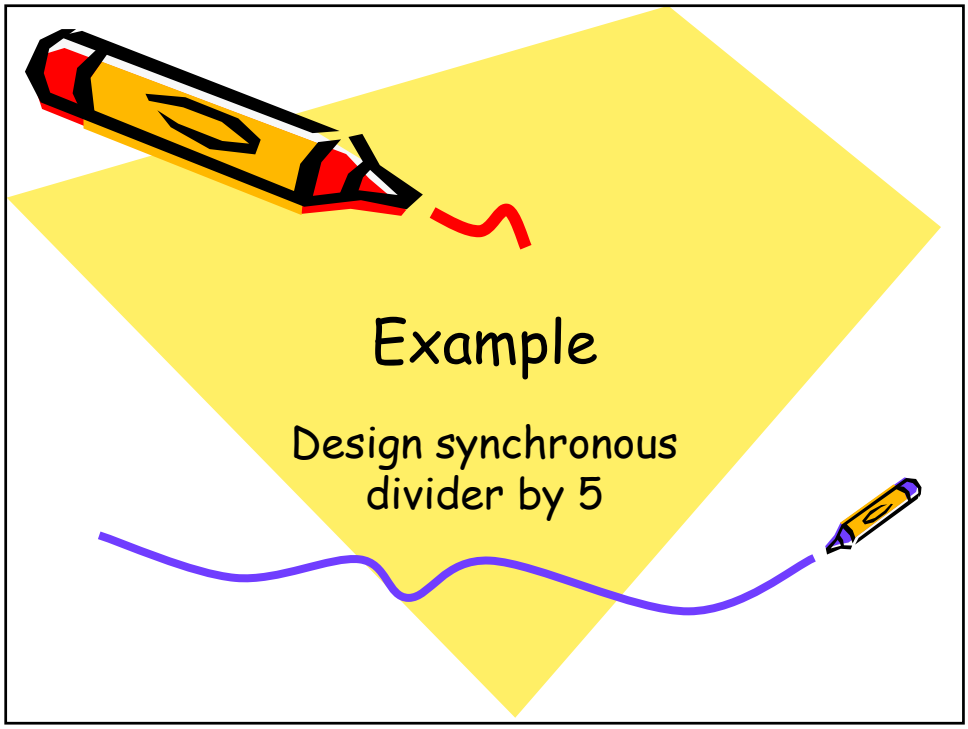


Synchronous Counters

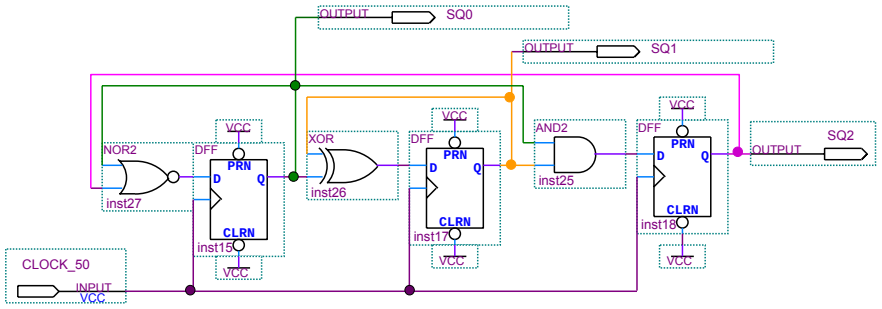
- Learn to read schematics - inputs of synchronous counters



SPS



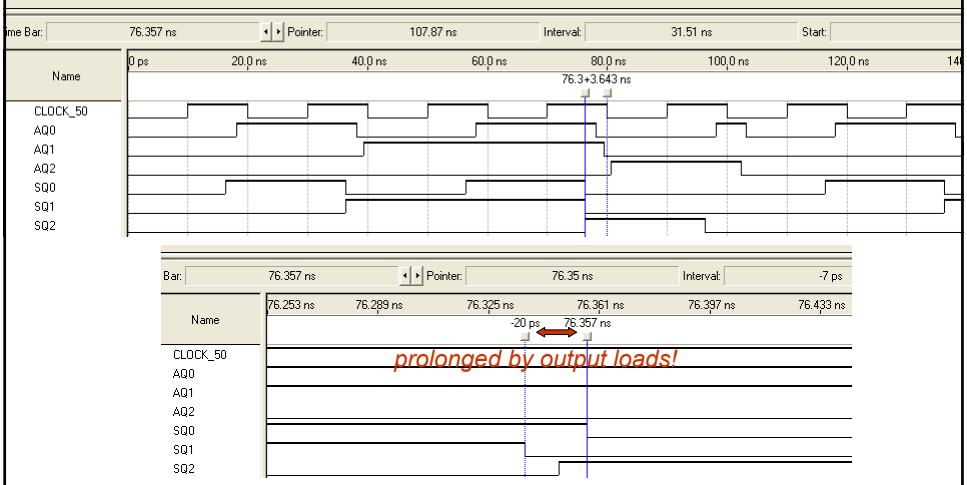
Result: Synchronous Counter 5 / Divider 5



- All output clock transitions happen at the same time.
- Clocks are connected together.
- Input connections are more complicated.

SPS

Time Simulation

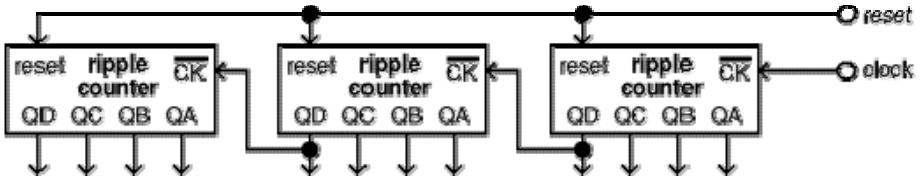


- Synchronous flip-flop receive clock request to change their outputs at one instance of time but they certainly change them at very near instances of time.

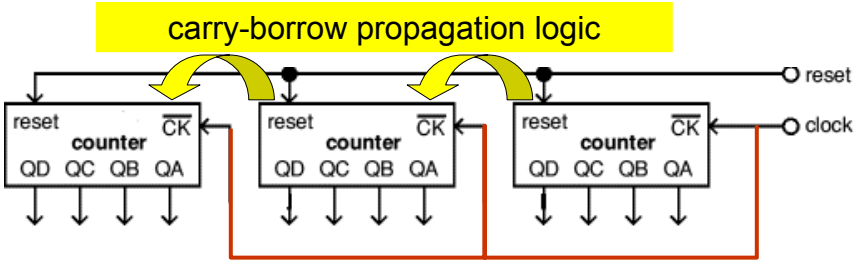
SPS

Cascading Counters

- Asynchronous: Main Scale Bit (QD) of a lower counter is usually connected to the clock of a higher counter



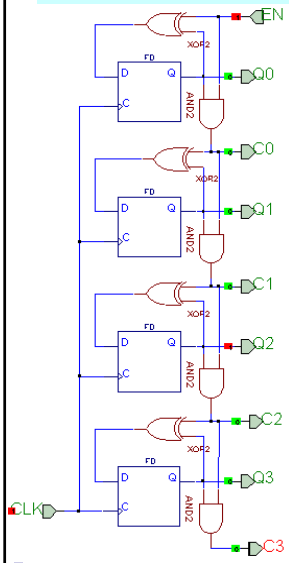
- Synchronous: common clock + carry/borrow propagation logic



SPS

Chaining 1 bit Synchronous Counters

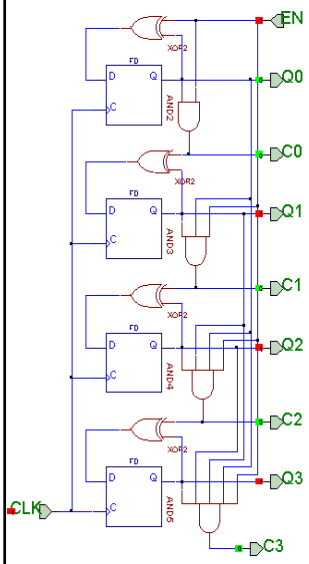
ripple carry logic



- Clocks are connected together.
- All output clock transitions happen at the same time.
- Ripple Carry
- Input connections are more complicated.

SPS

Counter with Carry Look-ahead



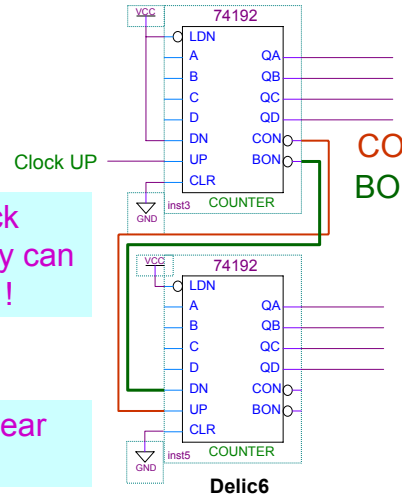
- Carries sent forward to eliminate carry propagation delay.
- In large counters, carry logic becomes major part of counter complexity.

Cascading Counters

- Cascading of synchronous counters can be performed by many ways, see next slides, e.g.
  - Carry/Borrow (74192, 74193),
  - MinMax + Ripple Carry (74190),
  - Common Clk + ENT/ENP gates (74162)

Cascading Decadic Counters 74192

by propagating carry/borrow



CON: carry
BON: borrow

One clock input only can be in "0" !

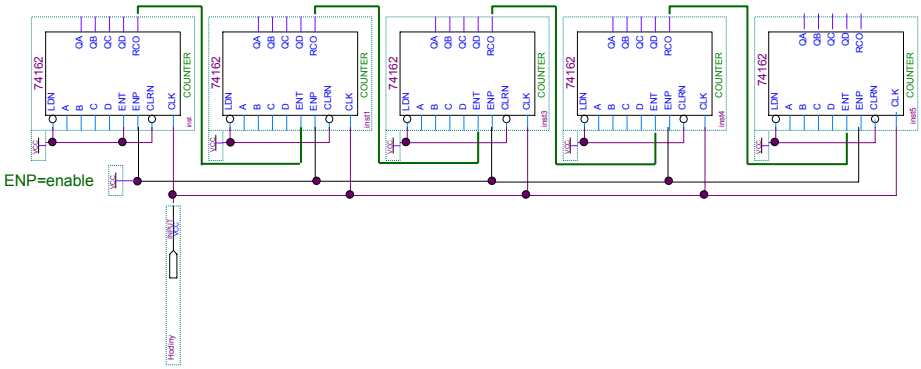
Notice clear on "1" !

Delic6

SPS

Cascading Decadic Counters 74162

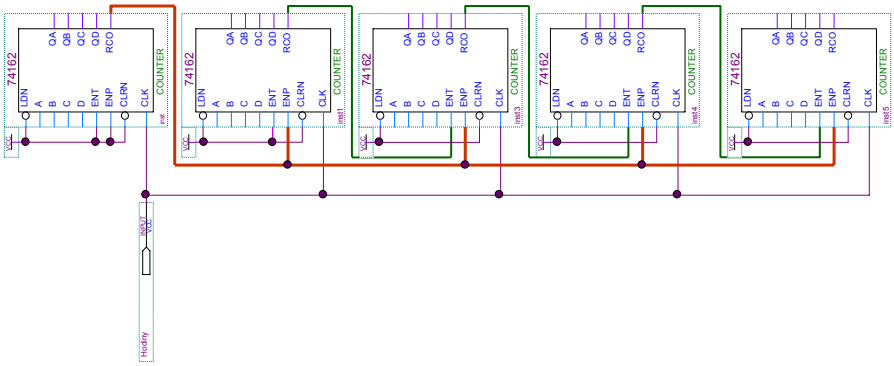
with ripple mode of carry



SPS

Advance Cascading Decadic Counters 74162

with the carry look-ahead



SPS

Synchronous Counter/Divider

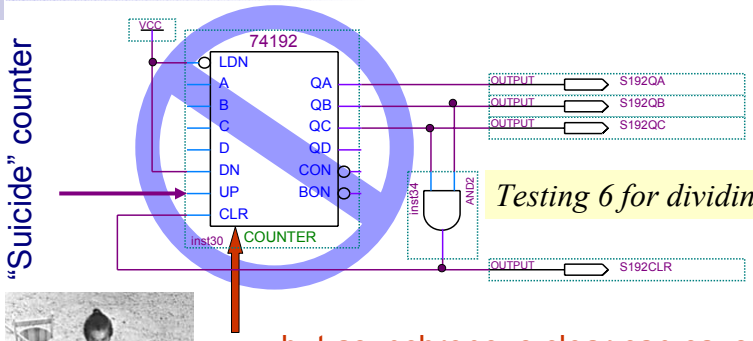
- Outputs can appear in different times…
- Some possible changes of outputs 1110→0001
  - 1110→1111→0001
  - 1110→1111→1101 →0101→0001
  - 1110→1010→1100 →0000→0001
  - 1110
    → any outputs from 0000 to 1111 are possible→ 0001

It is impossible to designed hazard free combination circuits for outputs of synchronous counters.

SPS

Decadic Counter 74192 with Asynchronous Clear

"Suicide" counter



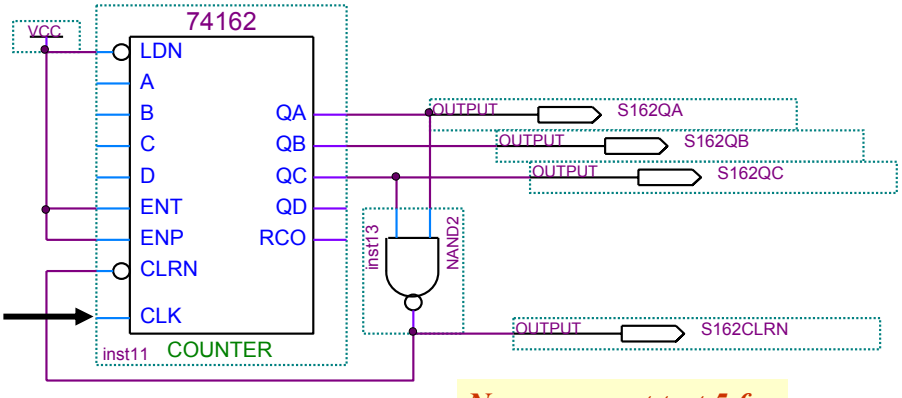
Testing 6 for dividing by 6!

...but asynchronous clear can cause random hazards, especially on higher frequencies

Asynchronous clear surely resets all flip-flops to 0, i.e. kills them all to 0, but the designer's job may be dead as well!

SPS

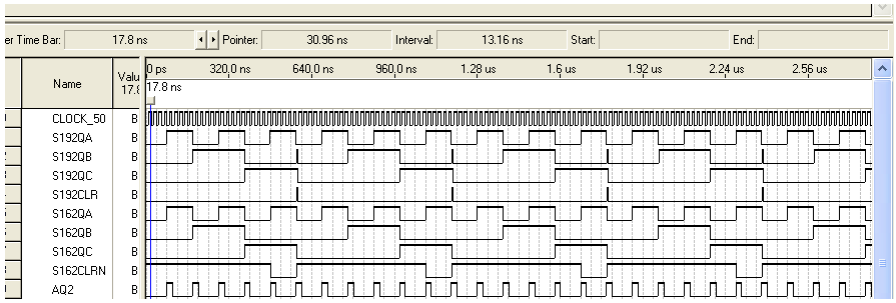
Cascading decadic counters 74162 with much better synchronous clear



Now, we must test 5 for dividing by 6!

SPS

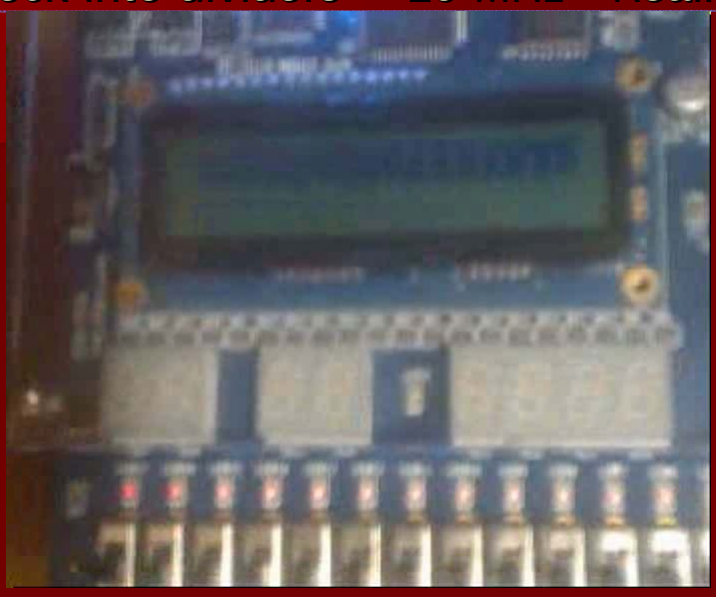
Clock into dividers = 10 MHz - simulation



The both dividers beautifully divide by 6 in simulations...

SPS

Clock into dividers = 10 MHz - Reality

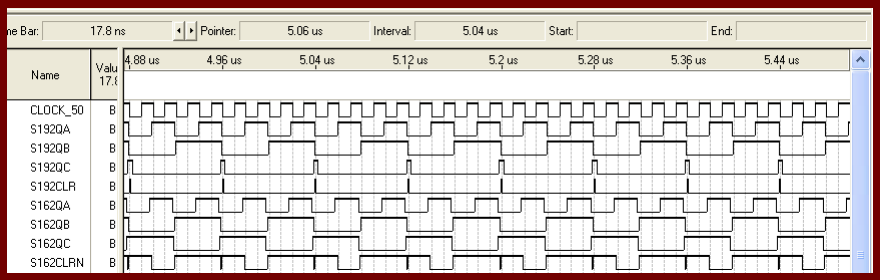


Left - 162 with synchronous clear | Right 192 with asynchronous clear

SPS

Clock 50 MHz

The both dividers give false outputs even in simulations.



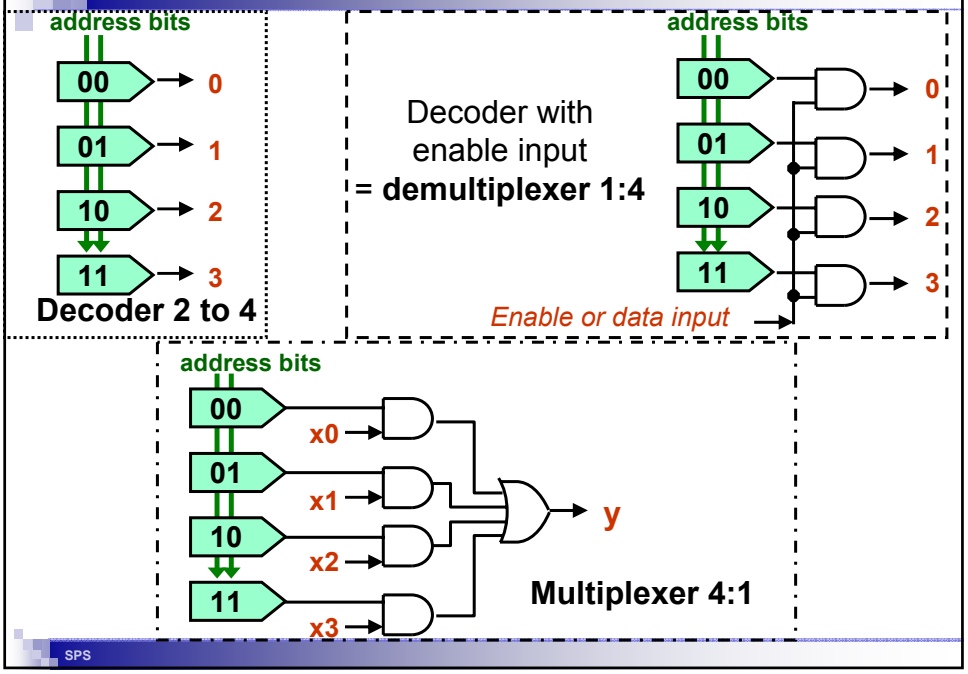
Max2 library has 25 MHz frequency limit

SPS

Minimization

of Logic Functions

SoP minterms in Circuits



SPS

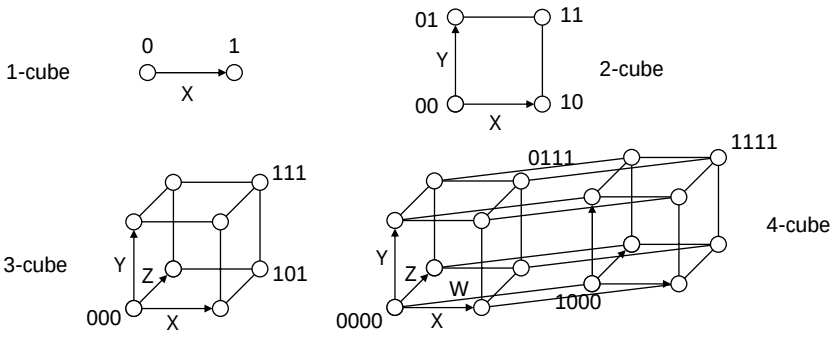
C++ Analogy

```
bool * decoder(int addr)
{   bool out [4];
    switch(addr&3)
    {case 0: out[0]=true;  out[1]=false; out[2]=false; out[3]=false; break;
     case 1: out[0]=false; out[1]=true;  out[2]=false; out[3]=false; break;
     case 2: out[0]=false; out[1]=false; out[2]=true;  out[3]=false; break;
     case 3: out[0]=false; out[1]=false; out[2]=false; out[3]=true;  break;
    }
        return out;
}
bool * demultiplexer(int addr, int enable)
{   bool out [4];
    switch(addr&3)
    {case 0: out[0]=enable; out[1]=false;  out[2]=false;  out[3]=false;  break;
     case 1: out[0]=false;  out[1]=enable; out[2]=false;  out[3]=false;  break;
     case 2: out[0]=false;  out[1]=false;  out[2]=enable; out[3]=false;  break;
     case 3: out[0]=false;  out[1]=false;  out[2]=false;  out[3]=enable; break;
    }
    return out;
}
bool multiplexer(int addr, bool x[4])
{switch(addr&3)
    {case 0: return x[0]; case 1: return x[1];
     case 2: return x[2]; case 3: return x[3];
    }
}
```

SPS

Boolean cubes

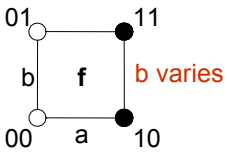
- n input variables = n-dimensional "cube"



Minimization in Boolean Cubes

Sub-cubes of on nodes can be used for simplification.
On-set – filled-in nodes, off-set – empty nodes

a	b	f	g
0	0	0	1
0	1	0	0
1	0	1	1
1	1	1	0



a asserted & unchanged

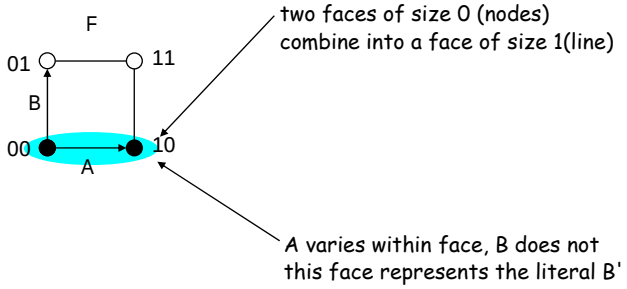
Uniting theorem

ab + ab' = a

Mapping truth tables onto Boolean cubes

- **Uniting theorem** combines two "faces" of a cube into a larger "face"
- Example:

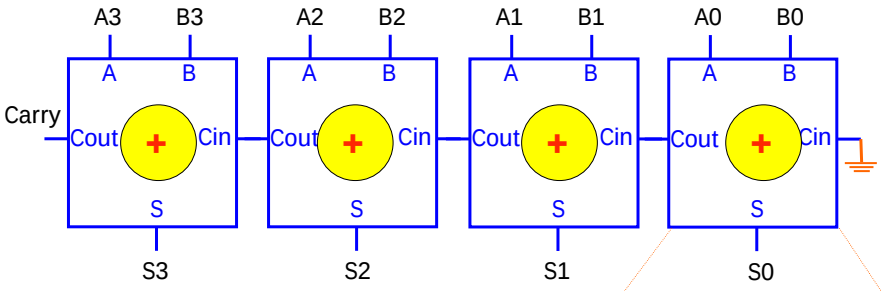
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0



Uniting theorem

a'b' + ab' = b'

Sčítačka

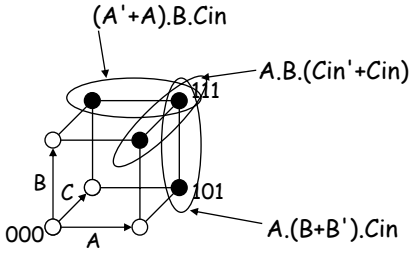


Jednobitová úplná sčítačka

Příklad návrhu: Sčítačka na krychli

- Jednobitová úplná sčítačka

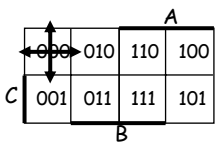
A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

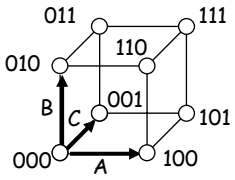


Cout = B.Cin+A.B+A.Cin

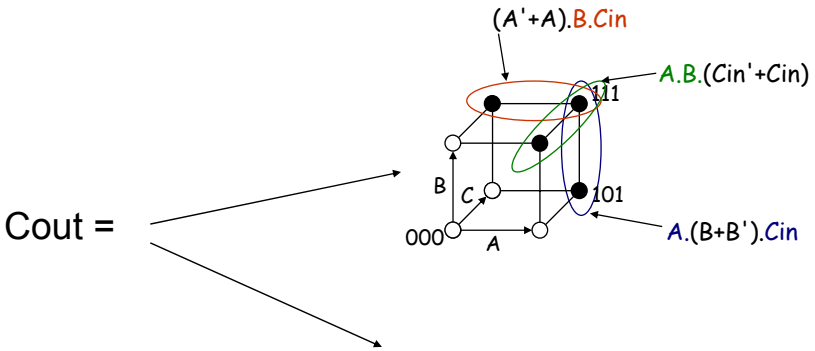
Karnaugh maps

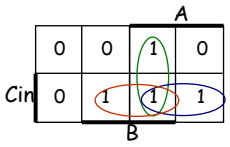
- Flat map of Boolean cube
- Wrap–around at edges
  - Wrap from first to last column
  - Wrap top row to bottom row





Karnaugh map versus cube

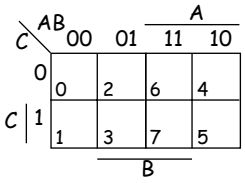


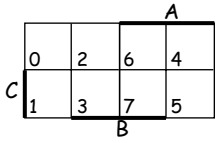


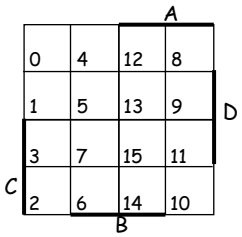
A.B+ A.Cin+ B.Cin

Karnaugh maps (cont'd)

- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10
  - Only a single bit changes in code for adjacent map cells



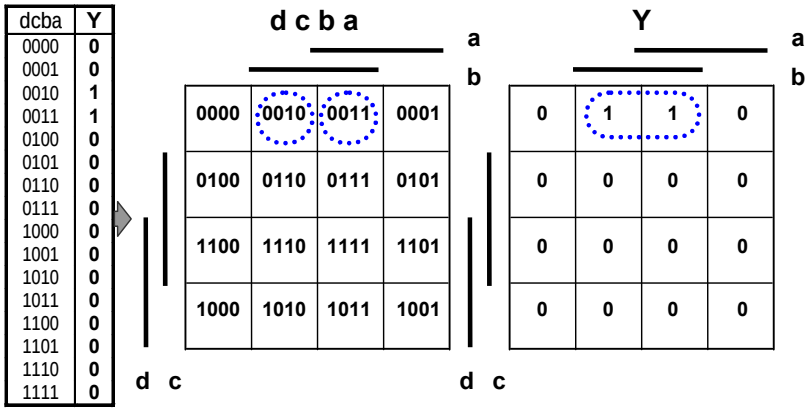




13 = 1101= ABC'D

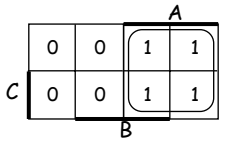
Příklad: Karnaughova mapa a logická tabulka

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

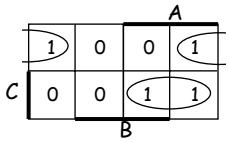


$$F = \overline{d}.\overline{c}.b.a + \overline{d}.\overline{c}.b.\overline{a} = \overline{d}.\overline{c}.b.\left(a + \overline{a}\right) = \overline{d}.\overline{c}.b$$

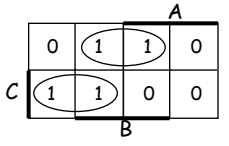
Příklady Karnaughových map



$G(A,B,C) = A$



$F(A,B,C) = \Sigma m(0,4,5,7) = AC + B'C'$



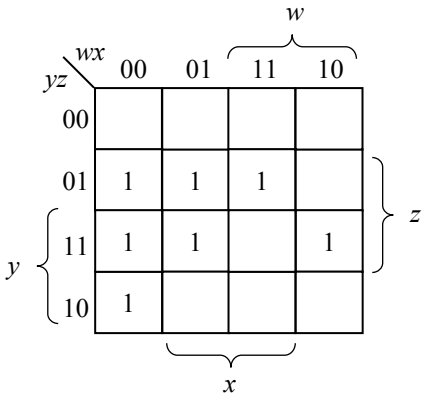
$F'(A,B,C) = \Sigma\, m(1,2,3,6) = BC' + A'C$

Miniotázka: Nakreslete odpovídající Booleovské krychle

Miniotázka: Minimalizujte mapu detektoru prvočísel

$F(w,x,y,z) = \Sigma(1,2,3,5,7,11,13)$

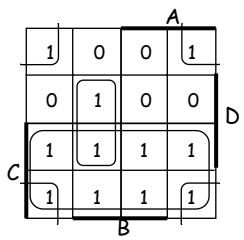
m_0	m_4	m_{12}	m_8
m_1	m_5	m_{13}	m_9
m_3	m_7	m_{15}	m_{11}
m_2	m_6	m_{14}	m_{10}

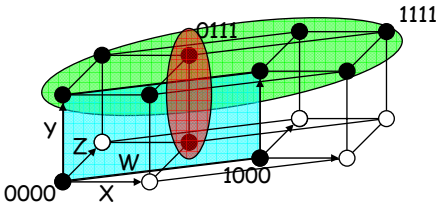


Karnaughova mapa: 4-proměnné

- F(A,B,C,D) = Σm(0,2,3,5,6,7,8,10,11,14,15)

  F = C + A'BD + B'D'

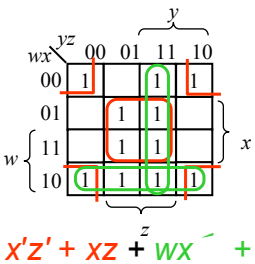


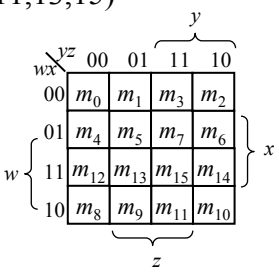


Hledáme minimální množinu, která pokryje funkci

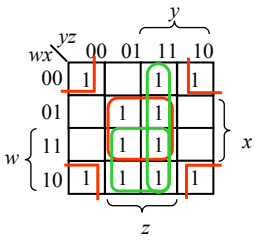
Může existovat více řešení

$F(w,x,y,z) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$

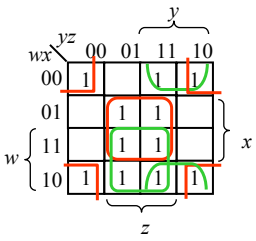




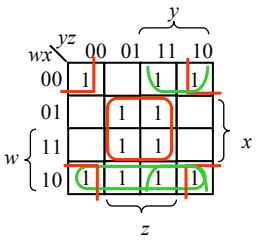
$x'z' + xz + wx' + yz$



$x'z' + xz + wz + yz$



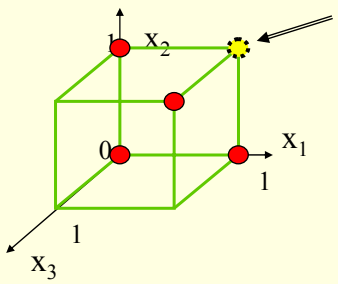
$x'z' + xz + wz + x'y$



$x'z' + xz + wx' + yz$

Multi-Valued Logic

- MVL (Multi-Valued Logic) – binární logika obohacená o informaci o stavu signálu, jako odpojení (vysoká impedance), neurčitost, apod.



don't care
tj. hodnota není nespecifikována při návrhu, neboť pro zamýšlené použití logického obvodu je bezvýznamné, zda bude 1 nebo 0

Example: binary coded decimal increment by 1

- BCD digits encode decimal digits 0 – 9 in bit patterns 0000 – 1001

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	0	-	-	-	-
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

off-set of W

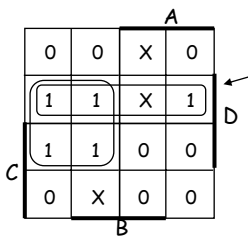
on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice – "don't care" about associated output values, can be exploited in minimization

Karnaugh Maps and Multivalue Logic: 0, 1, x

- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - f = A'D + B'C'D without don't cares
  - f = A'D + C'D with don't cares

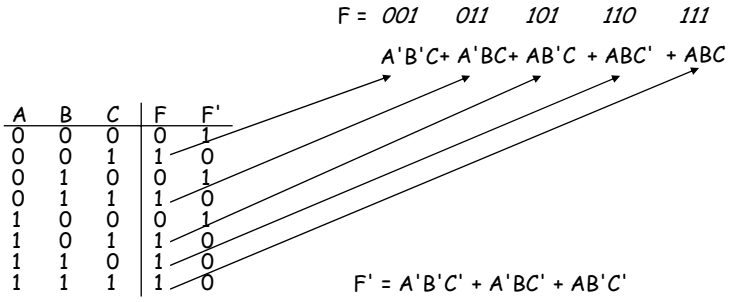


by using don't care as a "1" a 2-cube can be formed rather than a 1-cube to cover this node

don't cares can be treated as 1s or 0s depending on which is more advantageous

Opakování: Disjunktivní normální forma 1/2

- známa také jako
- **Sum-of-Products = S-o-P**
- **mintermy**



Disjunktivní normální forma 2/2

A	B	C	minterms	
0	0	0	A'B'C'	m0
0	0	1	A'B'C	m1
0	1	0	A'BC'	m2
0	1	1	A'BC	m3
1	0	0	AB'C'	m4
1	0	1	AB'C	m5
1	1	0	ABC'	m6
1	1	1	ABC	m7

zkratka pro mintermy
3 proměnných

F v kanonickém tvaru:

$F(A, B, C) = \Sigma m(1,3,5,6,7)$
$= m1 + m3 + m5 + m6 + m7$
$= A'B'C + A'BC + AB'C + ABC' + ABC$

Kanonická forma ≠ minimalní formy

$F(A, B, C) = A'B'C + A'BC + AB'C + ABC + ABC'$
$= (A'B' + A'B + AB' + AB)C + ABC'$
$= ((A' + A)(B' + B))C + ABC'$
$= C + ABC'$
$= ABC' + C$
$= AB + C$

Konjunktivní normální forma 1/2

- známa také jako
- **Products-of-Sum = P-o-S**
- **maxtermy**

F = *000* *010* *100*
F = (*A* + *B* + *C*) (*A* + *B*' + *C*) (*A*' + *B* + *C*)

A	*B*	*C*	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')

Konjunktivní normální forma 2/2

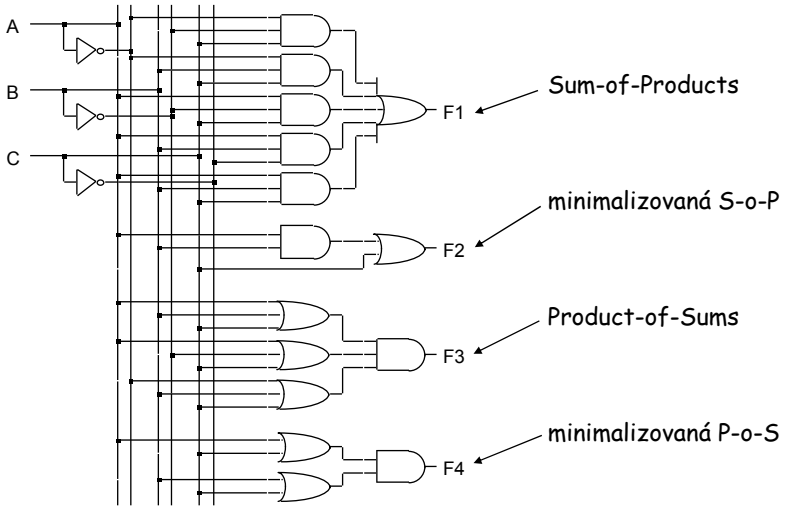
A	*B*	*C*	maxterms	
0	0	0	*A*+*B*+*C*	M0
0	0	1	*A*+*B*+*C*'	M1
0	1	0	*A*+*B*'+*C*	M2
0	1	1	*A*+*B*'+*C*'	M3
1	0	0	*A*'+*B*+*C*	M4
1	0	1	*A*'+*B*+*C*'	M5
1	1	0	*A*'+*B*'+*C*	M6
1	1	1	*A*'+*B*'+*C*'	M7

zkratky pro maxtermy
3 proměnných

F v kanonickém tvaru:
F(*A*, *B*, *C*) = ΠM(0,2,4)
= M0 · M2 · M4
= (*A* + *B* + *C*) (*A* + *B*' + *C*) (*A*' + *B* + *C*)

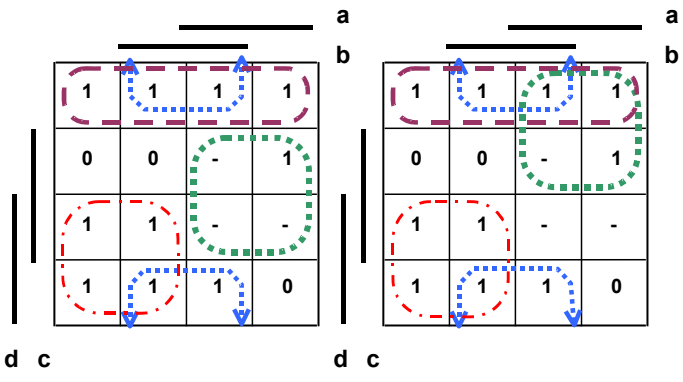
kanonický tvar ≠ minimální tvar
F(*A*, *B*, *C*) = (*A* + *B* + *C*) (*A* + *B*' + *C*) (*A*' + *B* + *C*)
= (*A* + *B* + *C*) (*A* + *B*' + *C*)
(*A* + *B* + *C*) (*A*' + *B* + *C*)
= (*A* + *C*) (*B* + *C*)

4 různé implementace funkce F = A.B + C



Karnaugh maps: don't cares

dcba	Y
0000	1
0001	1
0010	1
0011	1
0100	0
0101	1
0110	0
0111	X
1000	1
1001	0
1010	1
1011	1
1100	1
1101	X
1110	1
1111	X



$$F_{left}(x) = \bar{c}.\bar{d} + b.\bar{c} + a.c + \bar{a}.d \qquad F_{right}(x) = \bar{c}.\bar{d} + b.\bar{c} + a.\bar{d} + \bar{a}.d$$

$$F_{zeros}(x) = (a + \bar{c} + d).(\bar{a} + b + \bar{d})$$

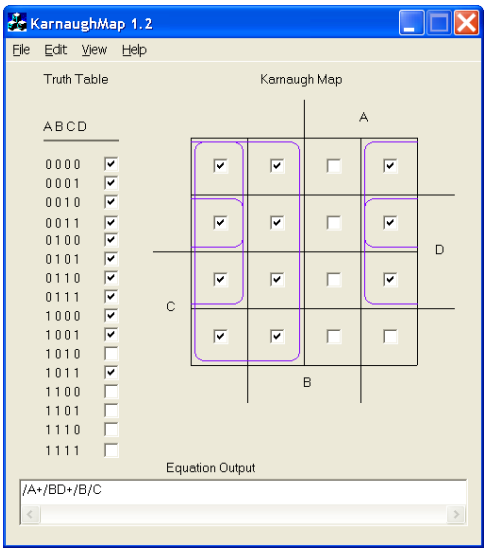
$$F_{left}(x) = F_{right}(x) = F_{zeros}(x);\ [\forall x; F_n(x) = 0 \vee 1]$$

S-o-P, P-o-S a de Morganův teorém

- Sum-of-products
  - F' = A'B'C' + A'BC' + AB'C'
- Aplikujeme de Morganův teorém
  - (F')' = (A'B'C' + A'BC' + AB'C')'
  - F = (A + B + C) (A + B' + C) (A' + B + C)

- Product-of-sums
  - F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')
- Aplikujeme de Morganův teorém
  - (F')' = ((A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C'))'
  - F = A'B'C + A'BC + AB'C + ABC'

Example 3-5 using kmap12



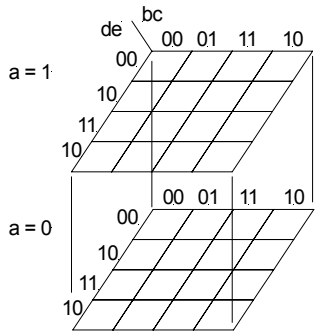
$F = y' + w'z' + xz'$

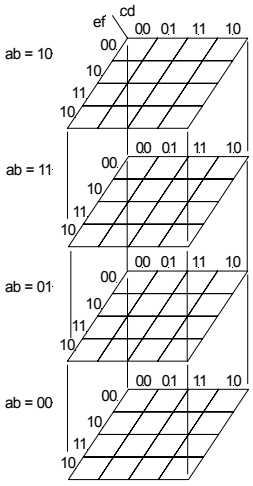
symbol names

$A = y$
$B = z$
$C = w$
$D = x$

This map is transpose of way Mano does it.

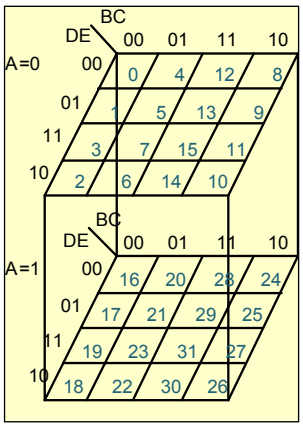
Higher Dimensional K-maps

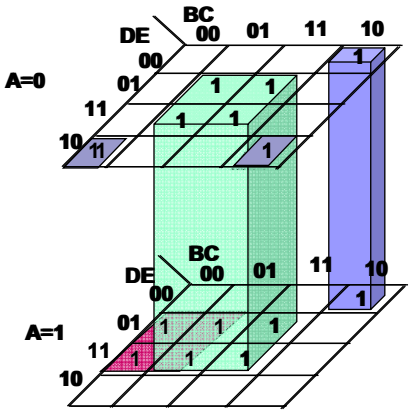




Gate Logic: Two-Level Simplification

5-Variable K-maps





$f(A,B,C,D,E) = \Sigma m(2,5,7,8,10, 13,15,17,19,21,23,24,29\ 31)$

$= C\ E + A\ B'\ E + B\ C'\ D'\ E' + A'\ C'\ D\ E'$

Karnaugh maps 8 variables