DIGITÁLIS TECHNIKA

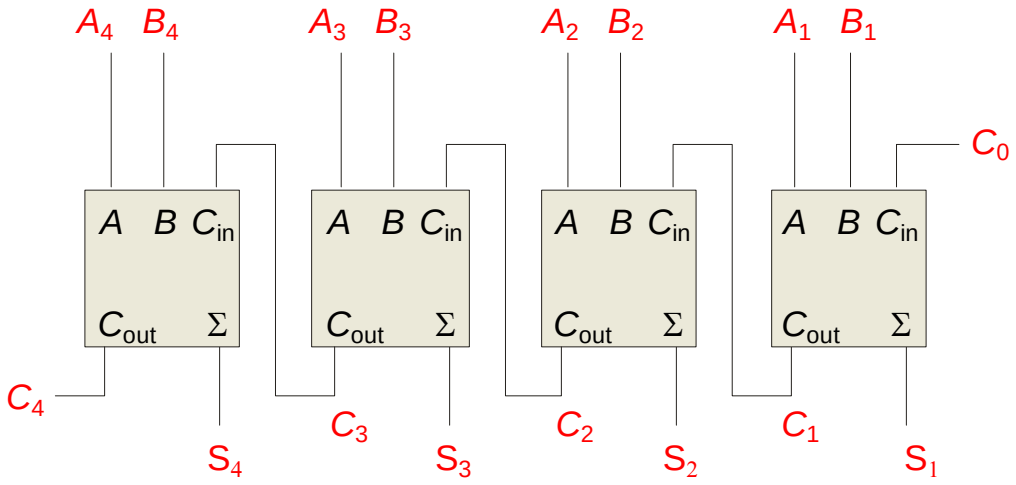
Dr. Oniga István

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta.
A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

Több bites összeadók I

A több bites számokat teljes összeadókból építhetjük meg

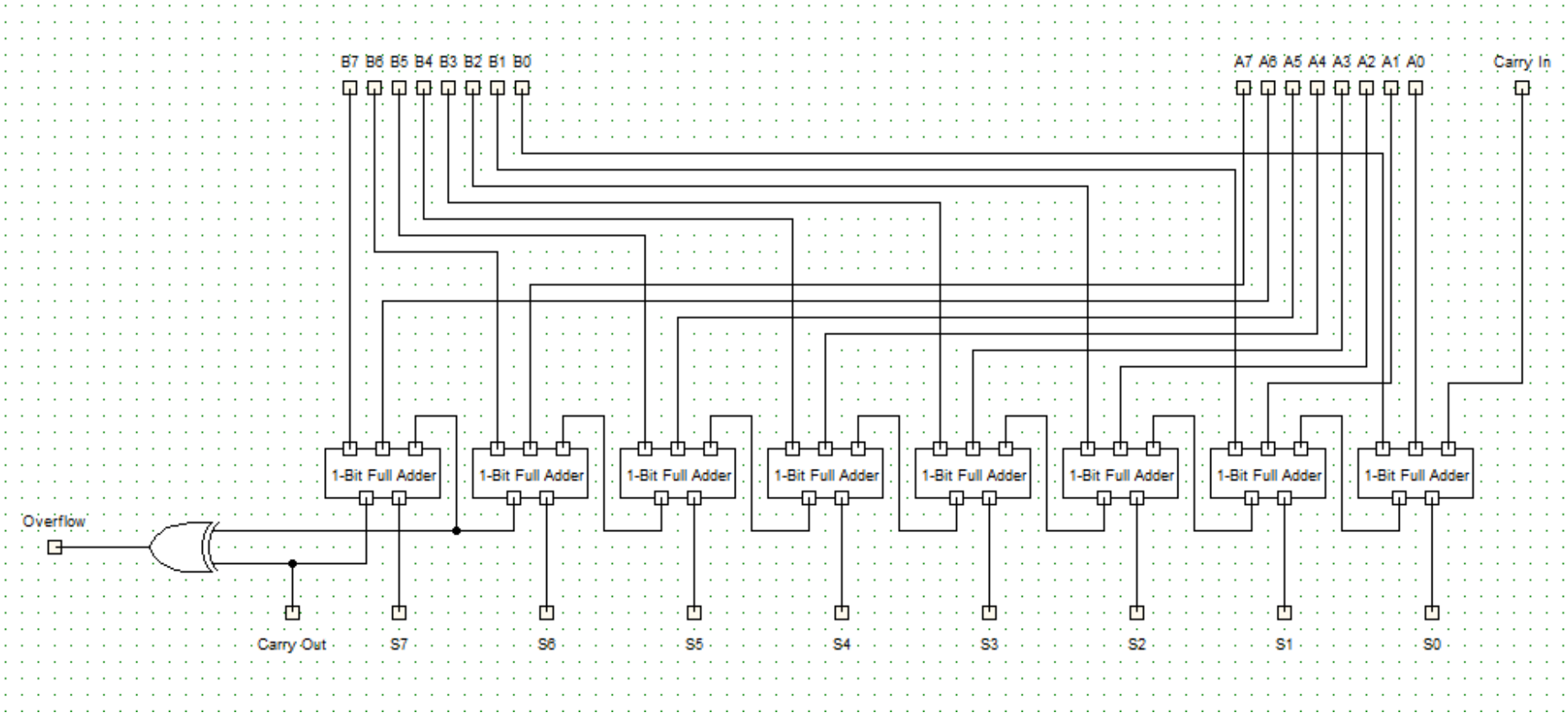
Soros átvitelű 4 bites összeadó (Ripple carry adder): 7483



Lassú

S_i és C_i eredményt csak azután kapjuk meg amikor C_{i-1} felvette végső értékét

Soros átvitelű 8 bites összeadó
(8-bit Ripple carry adder)

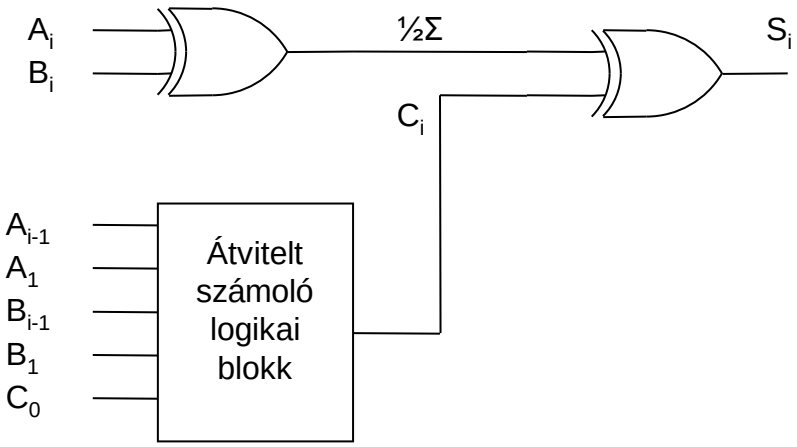


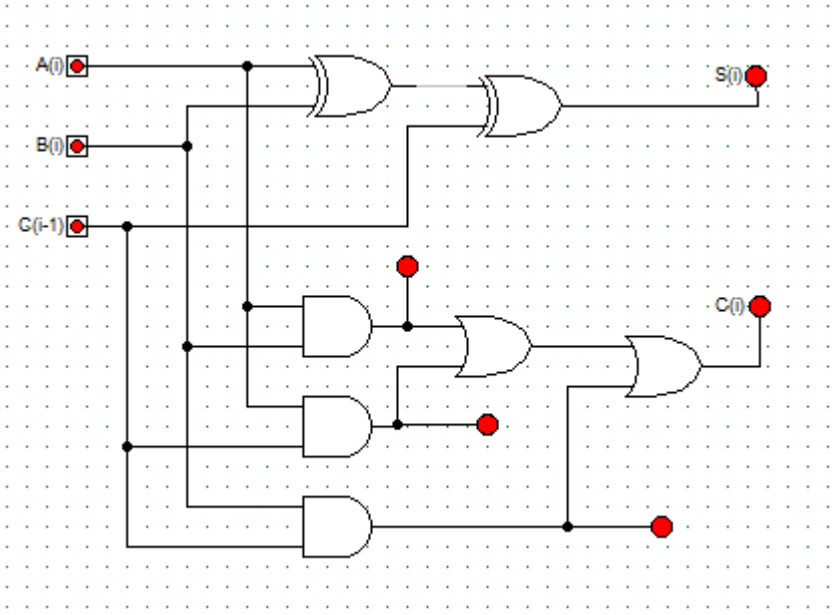
Több bites összeadók II

Párhuzamos átvitelű 4 bites összeadó (Look-ahead carry adder) = gyors átvitelképző

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = (A_i B_i + A_i C_{i-1}) + B_i C_{i-1}$$

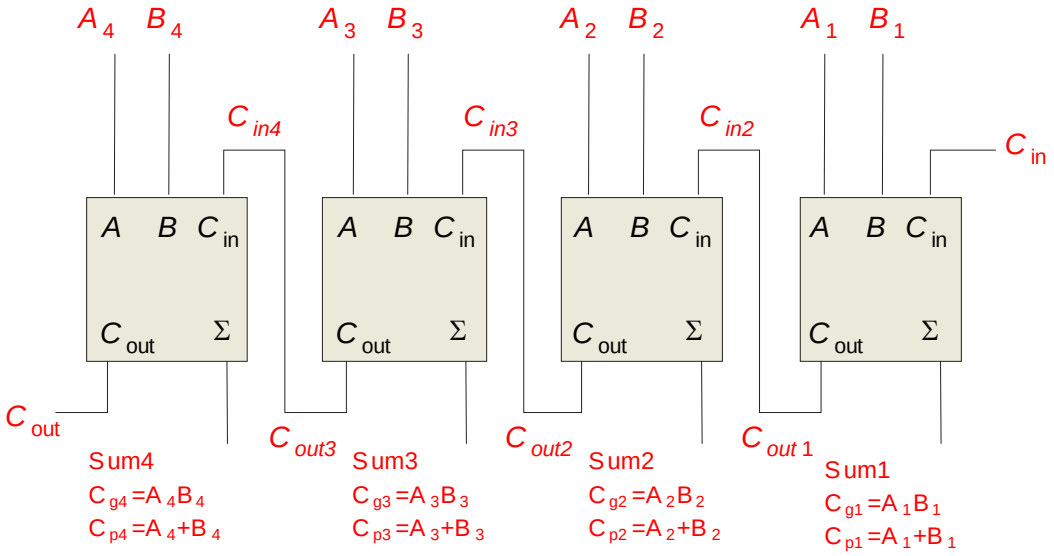
Generate carry — Cgi — Cpi — Propagate carry





Párhuzamos átvitelű 4 bites összeadó

Look-ahead carry adder = gyors átvitelképző



$C_{out1}=C_{g1}+C_{p1}C_{in1}$

$C_{in2}=C_{out1}$

$C_{out2}= C_{g2}+C_{p2}C_{in2}= C_{g2}+C_{p2}C_{out1}= C_{g2}+C_{p2}(C_{g1}+C_{p1}C_{in1})= C_{g2}+C_{p2}C_{g1}+ C_{p2}C_{p1}C_{in1}$

$C_{out2}=A_2B_2+(A_2+B_2)\ A_1B_1+(A_2+B_2)\ (A_1+B_1)C_{in1}$

$C_{in3}=C_{out2}$

$C_{out3}= C_{g3}+C_{p3}C_{in3}= C_{g3}+C_{p3}C_{out2}= C_{g3}+C_{p3}(C_{g2}+C_{p2}C_{g1}+C_{p2}C_{p1}C_{in1})= C_{g3}+C_{p3}C_{g2}+ C_{p3}C_{p2}C_{g1}+ C_{p3}C_{p2}C_{p1}C_{in1}$

$C_{in4}=C_{out3}$

$C_{out4}=C_{g4}+C_{p4}C_{in4}=C_{g4}+C_{p4}C_{out3}= C_{g4}+C_{p4}(C_{g3}+C_{p3}C_{g2}+C_{p3}C_{p2}C_{g1}+C_{p3}C_{p2}C_{p1}C_{in1}) = C_{g4}+C_{p4}C_{g3}+ C_{p4}C_{p3}C_{g2}+ C_{p4}C_{p3}C_{p2}C_{g1}+ C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$

Párhuzamos átvitelű 4 bites összeadó

$C_{out1}=C_{g1}+C_{p1}C_{in1}$

$C_{in2}=C_{out1}$

$C_{out2}= C_{g2}+C_{p2}C_{in2}= C_{g2}+C_{p2}C_{out1}= C_{g2}+C_{p2}(C_{g1}+C_{p1}C_{in1})= C_{g2}+C_{p2}C_{g1}+ C_{p2}C_{p1}C_{in1}$

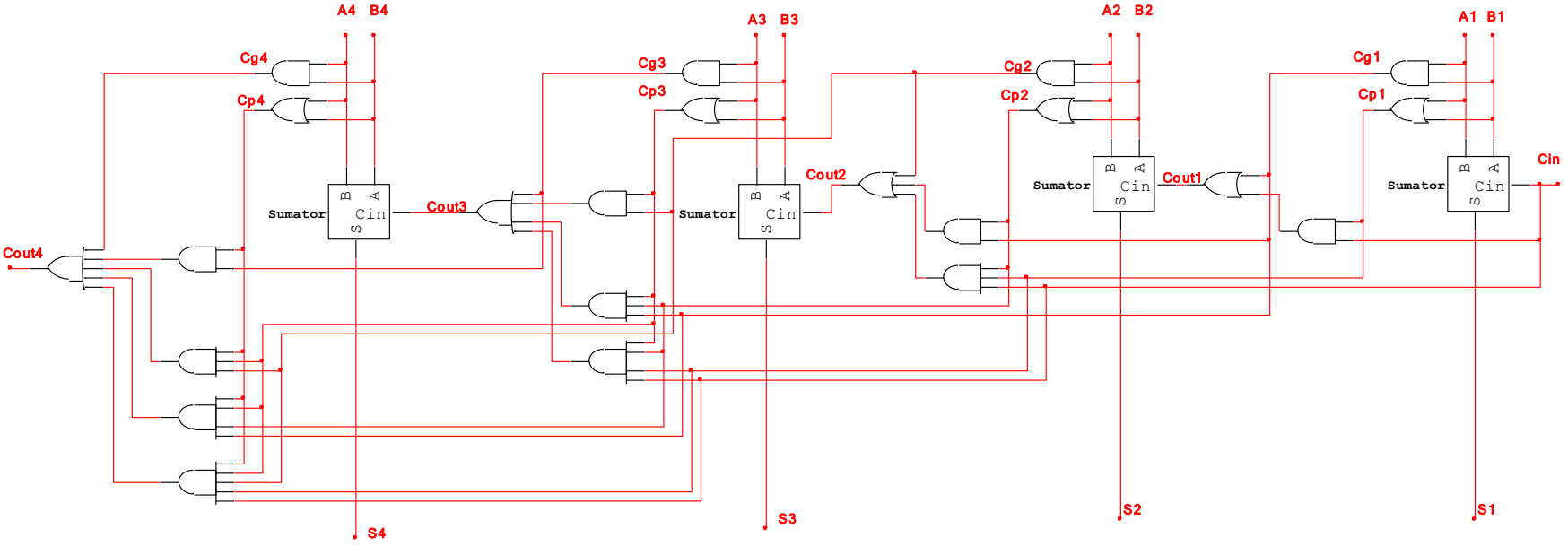
$C_{out2}=A_2B_2+(A_2+B_2)\ A_1B_1+(A_2+B_2)\ (A_1+B_1)C_{in1}$

$C_{in3}=C_{out2}$

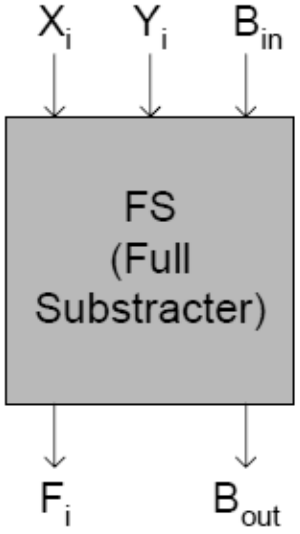
$C_{out3}= C_{g3}+C_{p3}C_{in3}= C_{g3}+C_{p3}C_{out2}= C_{g3}+C_{p3}(C_{g2}+C_{p2}C_{g1}+C_{p2}C_{p1}C_{in1})= C_{g3}+C_{p3}C_{g2}+ C_{p3}C_{p2}C_{g1}+ C_{p3}C_{p2}C_{p1}C_{in1}$

$C_{in4}=C_{out3}$

$C_{out4}=C_{g4}+C_{p4}C_{in4}=C_{g4}+C_{p4}C_{out3}= C_{g4}+C_{p4}(C_{g3}+C_{p3}C_{g2}+C_{p3}C_{p2}C_{g1}+C_{p3}C_{p2}C_{p1}C_{in1}) = C_{g4}+C_{p4}C_{g3}+ C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1}+ C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$

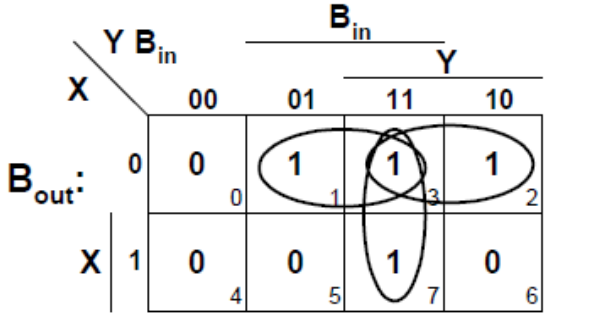


1 BITES TELJES KIVONÓ

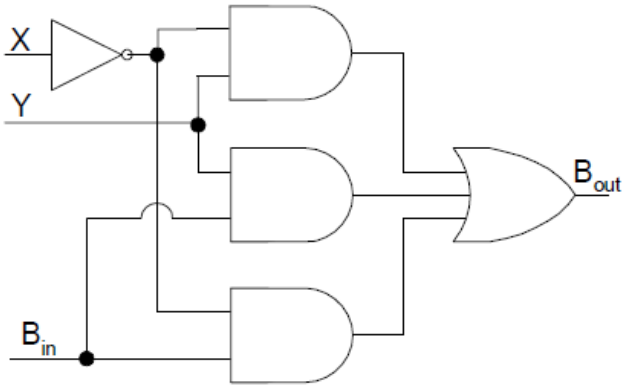


X_i	Y_i	Bin	F_i	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$F_i = X_i \oplus Y_i \oplus B_{in} :$$



$$B_{out} = \overline{X_i} \cdot Y_i + \overline{X_i} \cdot B_{in} + Y_i \cdot B_{in}$$



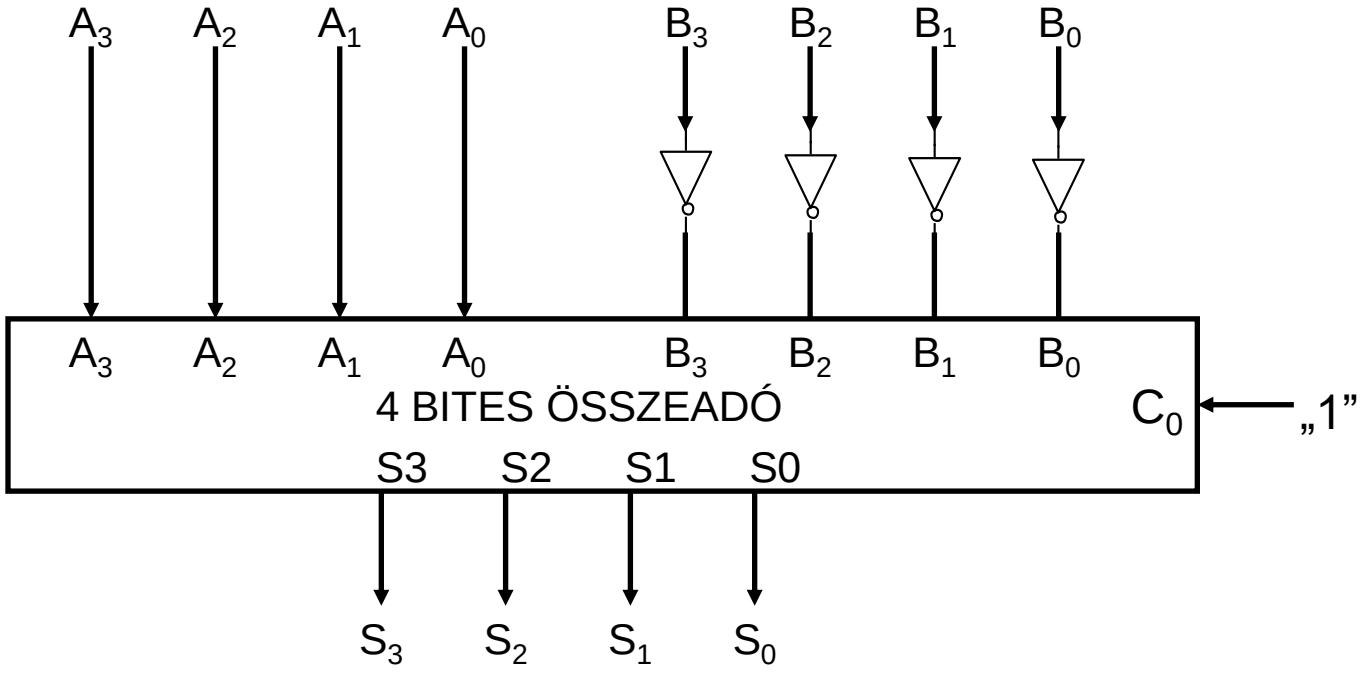
4-bites teljes kivonó

$A-B=A+(-B)$

$-B_N = B_N^{(2)}$

$A-B=A+ B_N^{(2)}$

$B^{(2)}=\overline{B}+1$

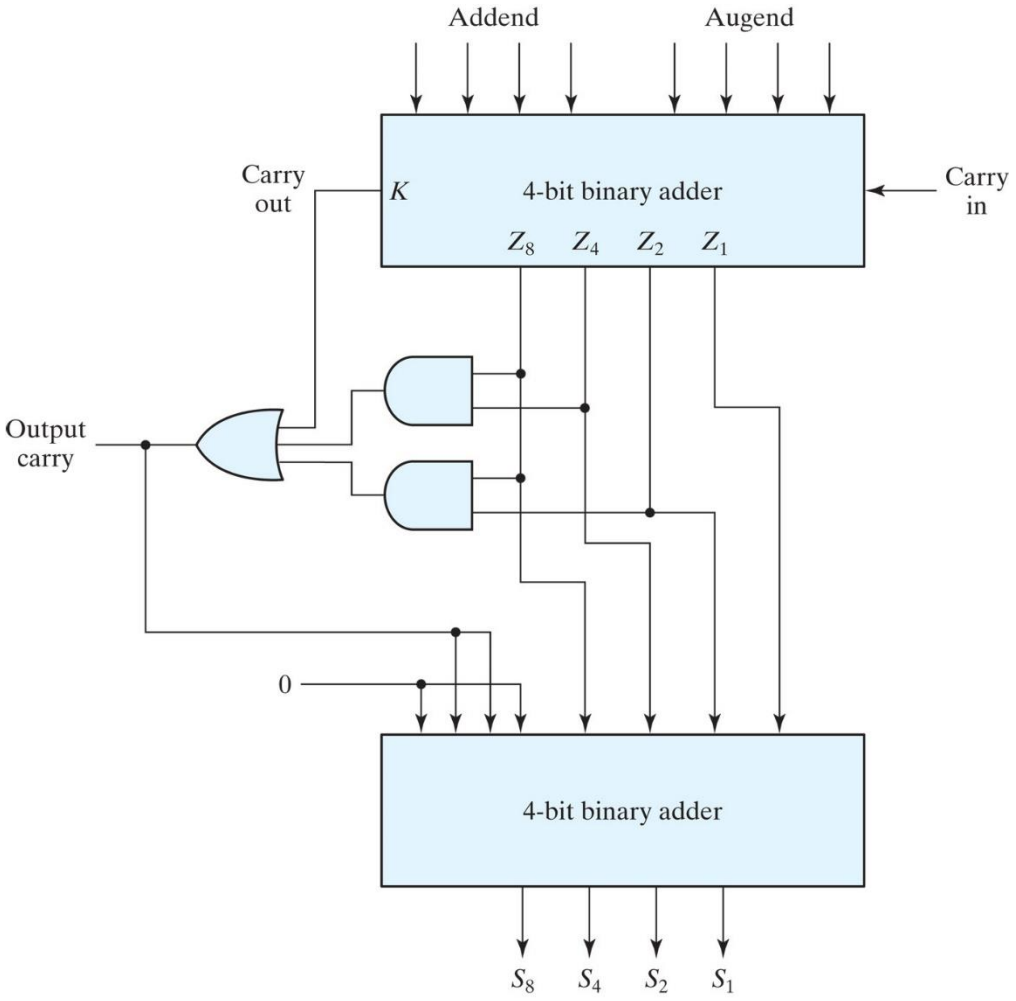


BCD összeadó

Table 4.5
Derivation of BCD Adder

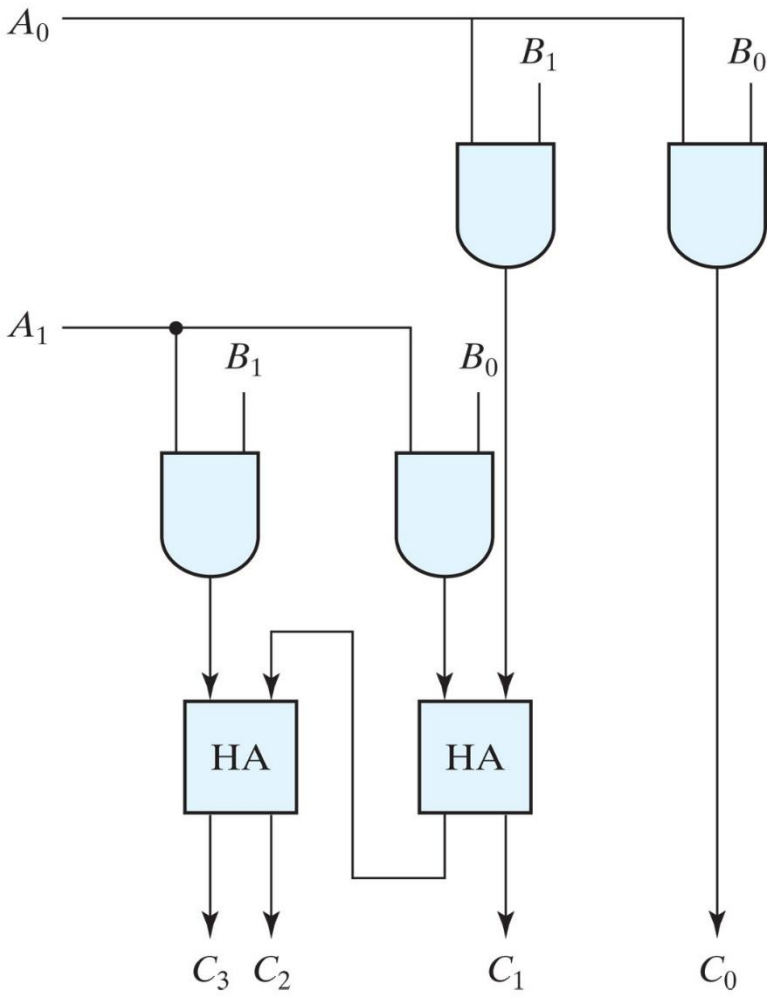
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD összeadó

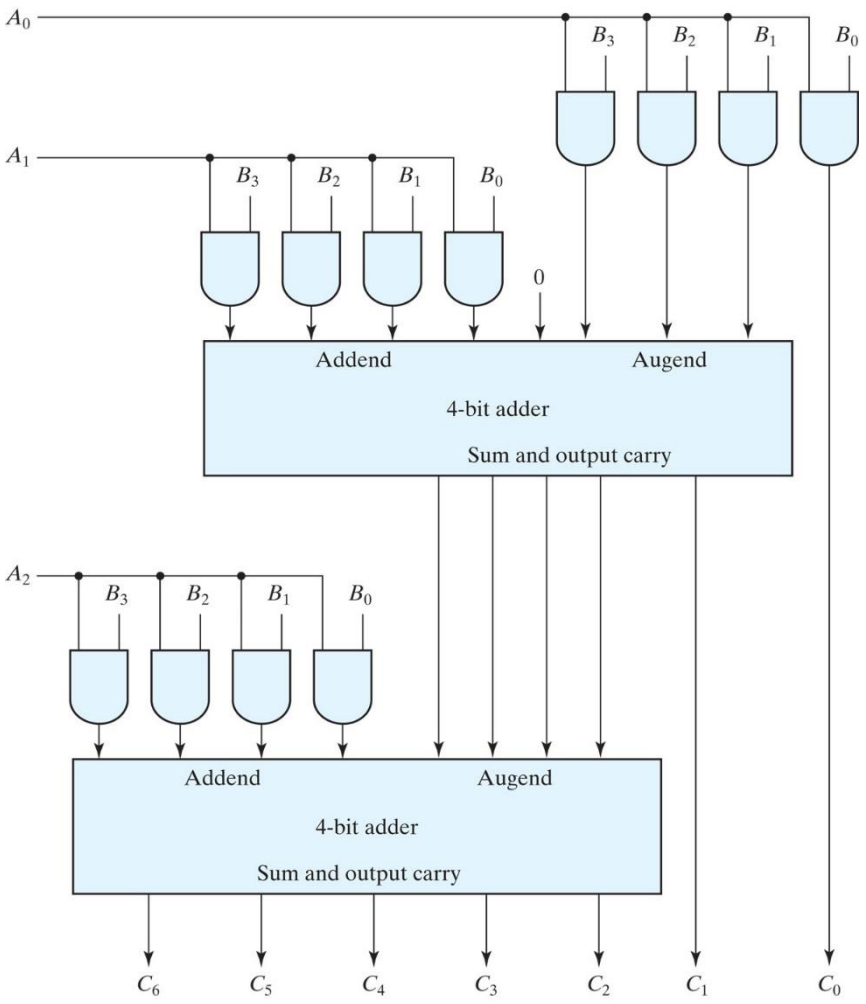


2 bites számok szorzása

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



Egy 4 bites szám és egy 3 bites szám szorzássá



ARITMETIKAI-LOGIKAI EGYSÉGEK (ALU)

ALU minden processzorban van, de önálló, diszkrét áramkörként is gyártják.

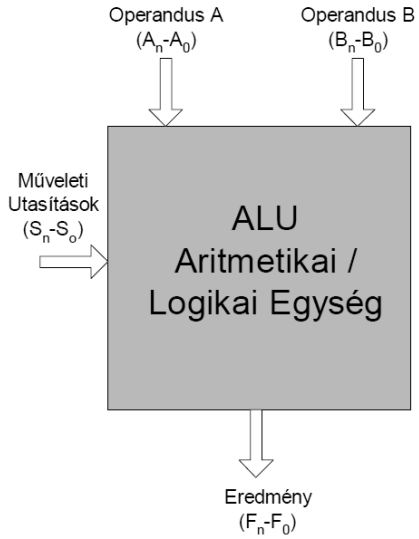
Az ALU egy kombinációs hálózat

- a bemeneteikre érkező két számmal (A és B)

- S bemeneteken megadott logikai vagy aritmetikai műveletet végzik el

- az eredményt az F kimeneteken jelenítik meg.

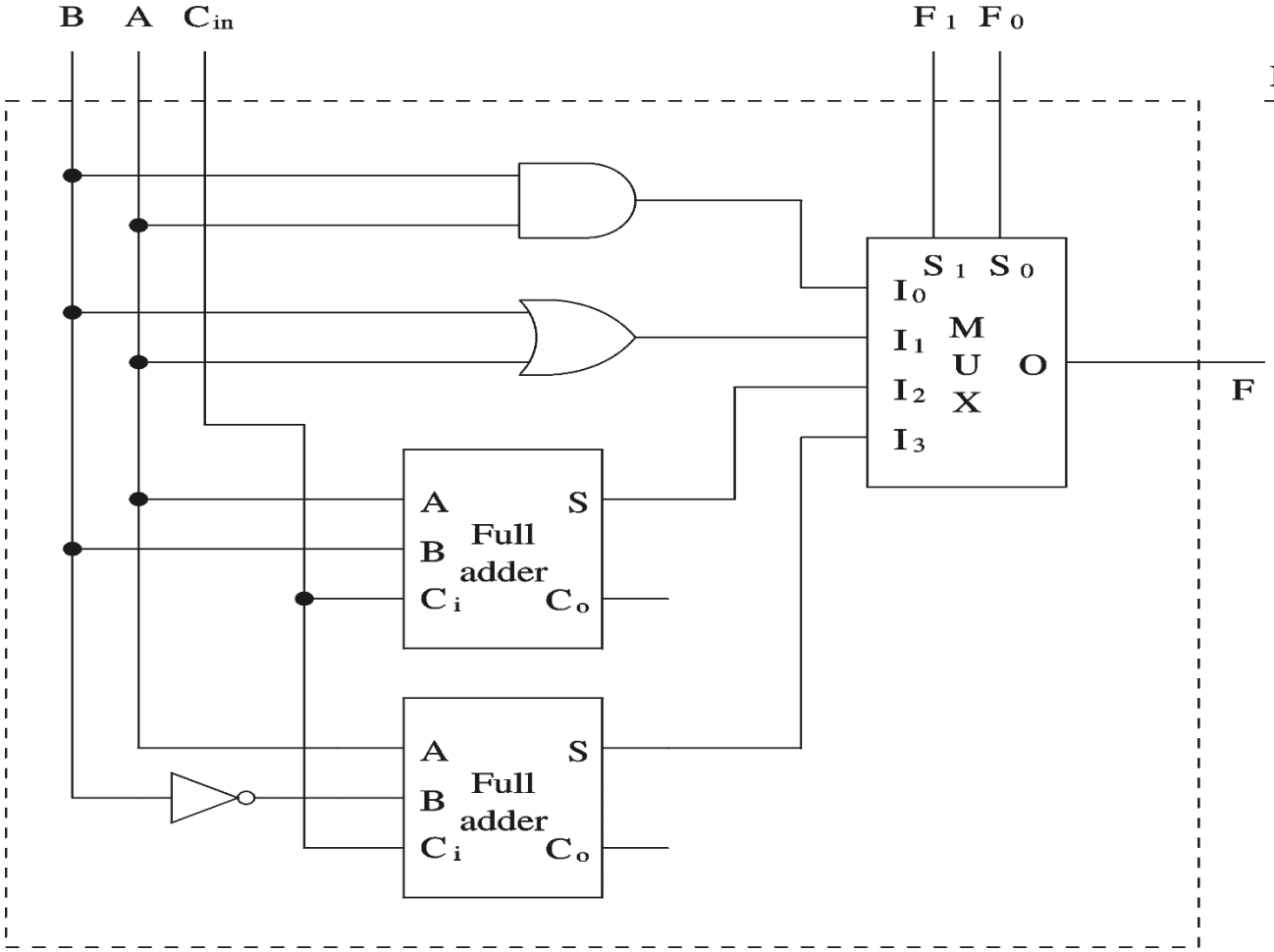
- Összeadás és kivonás művelet elvégzésekor figyelembe veszik az előző helyérték átvitelét (Cn), és az előállított átvitelt továbbítják a következő helyértékre (C).



Műveletek:
- bináris aritmetikai utasítások (összeadás, kivonás, stb.),
- logikai műveletek (AND, OR, stb.),
- regiszterműveletek (jobbra-balra léptetés, inkrementálás, dekrementálás),

1 bites ALU

(ALU = arithmetic + logic unit)



F_1	F_0	F
0	0	A and B
0	1	A or B
1	0	A + B
1	1	A − B

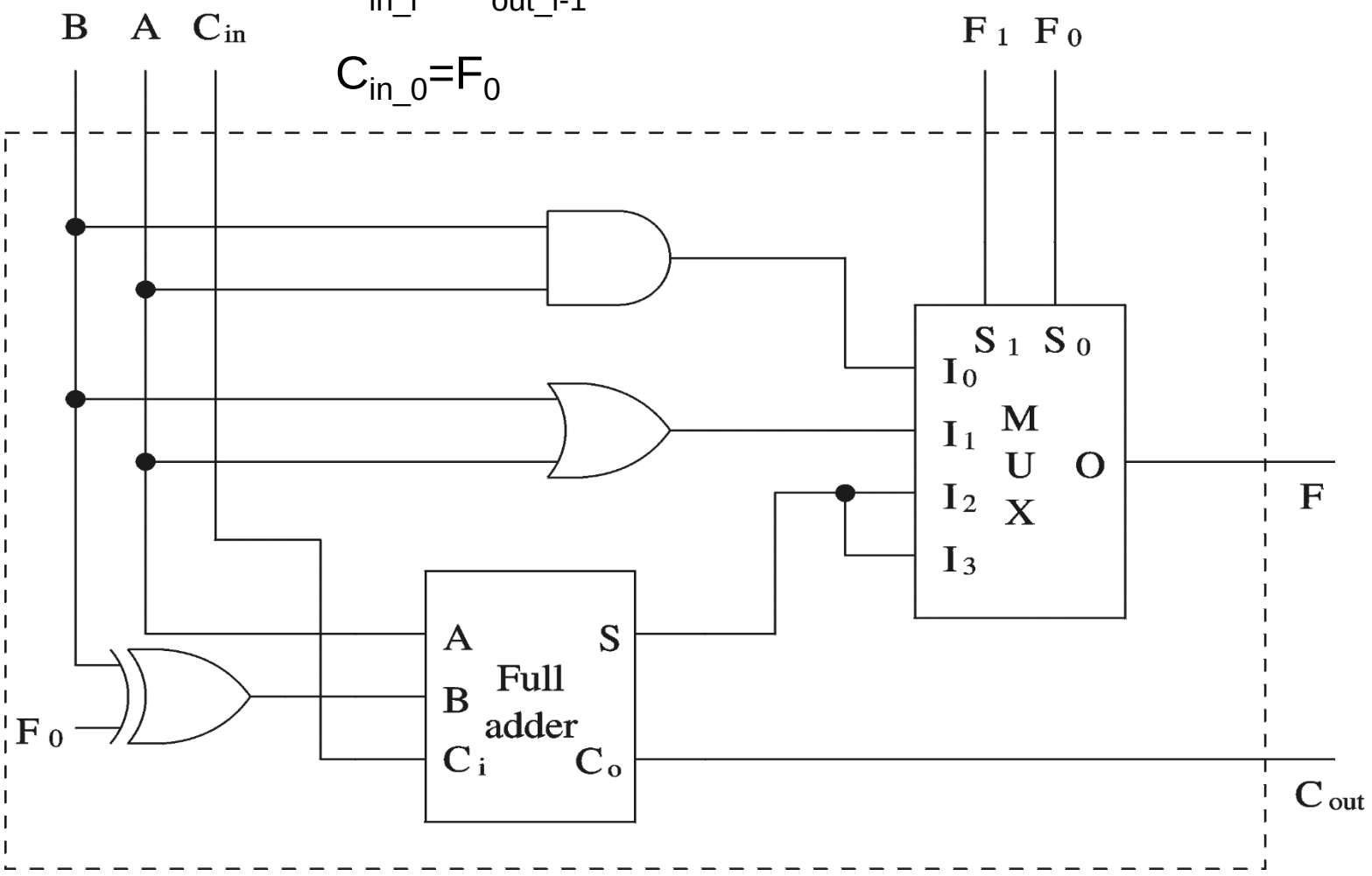
$C_{in\_i} = C_{out\_i-1}$

$C_{in\_0} = F_0$

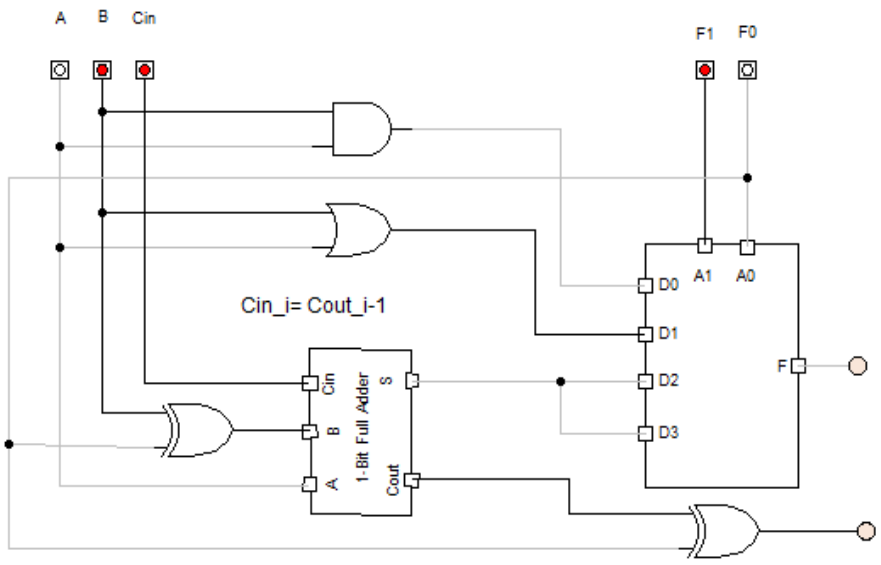
Optimalizált 1 bites ALU

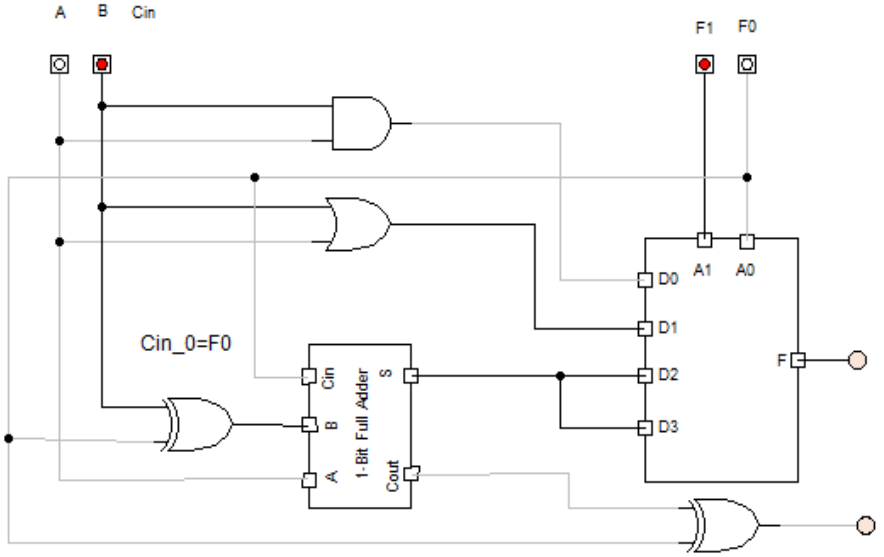
$C_{in\_i} = C_{out\_i-1}$

$C_{in\_0} = F_0$

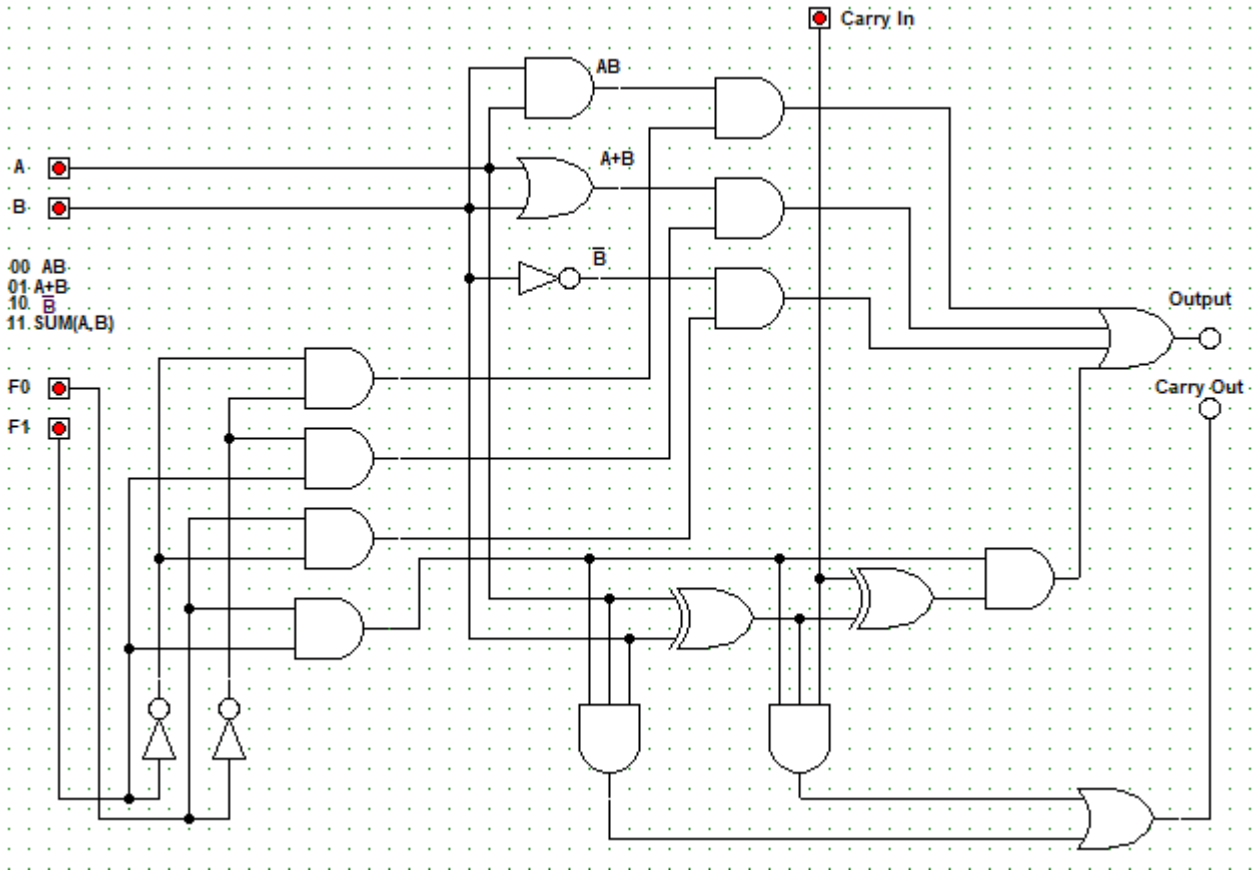


Optimalizált 1 bites ALU





1 bites ALU

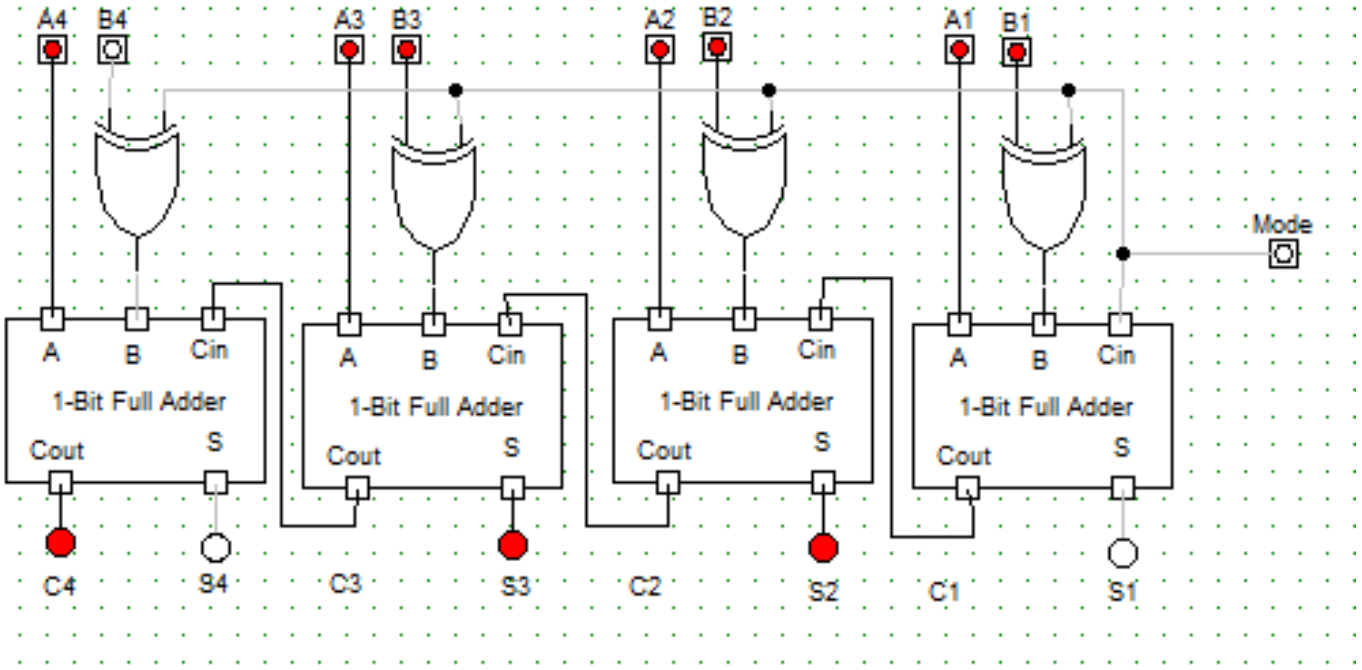


Simplified version of a circuit in Tanenbaum, Andrew S., Structured Computer Organization, Fourth Edition Prentice-Hall, 1999 [p.138]

Optimalizált 4 bites összeadó/kivonó

Összeadás: Mode = 0 => $b_i' = b_i$

Kivonás: Mode = 1 => $b_i' = \overline{b_i}$

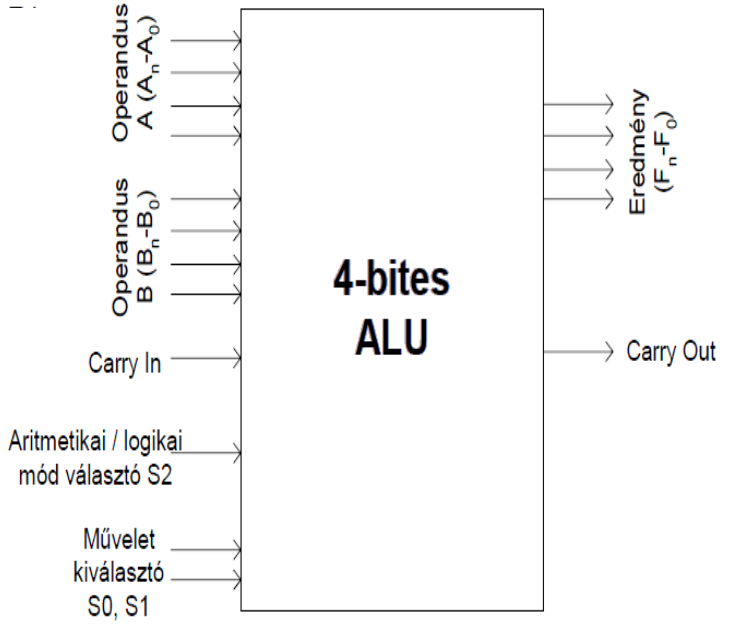


4-bites ALU - 74LS181

- Két 4-bites operandus (A, B)
- 4 bites eredmény (F)
- Átvitel: CarryIn/ Out
- S2: Aritmetikai/ logikai mód választó(MUX)
- S0, S1: művelet kiválasztó

Művelet kiválasztás:				Művelet:	Megvalósított függvény:
S2	S1	S0	Cin		
0	0	0	0	F=A	A átvitele
0	0	0	1	F=A+1	A értékének növelése 1-el (increment)
0	0	1	0	F=A+B	Összeadás
0	0	1	1	F=A+B+1	Összeadás carry figyelembevételével
0	1	0	0	$F = A + \overline{B}$	A + 1's komplemens B
0	1	0	1	$F = A + \overline{B} + 1$	Kivonás
0	1	1	0	F=A-1	A értékének csökkentése 1-el (decrement)
0	1	1	1	F=A	A átvitele
1	0	0	0	$F = A \wedge B$	AND
1	0	1	0	$F = A \vee B$	OR
1	1	0	0	$F = A \oplus B$	XOR
1	1	1	0	$F = \overline{A}$	A negáltja (NOT A)

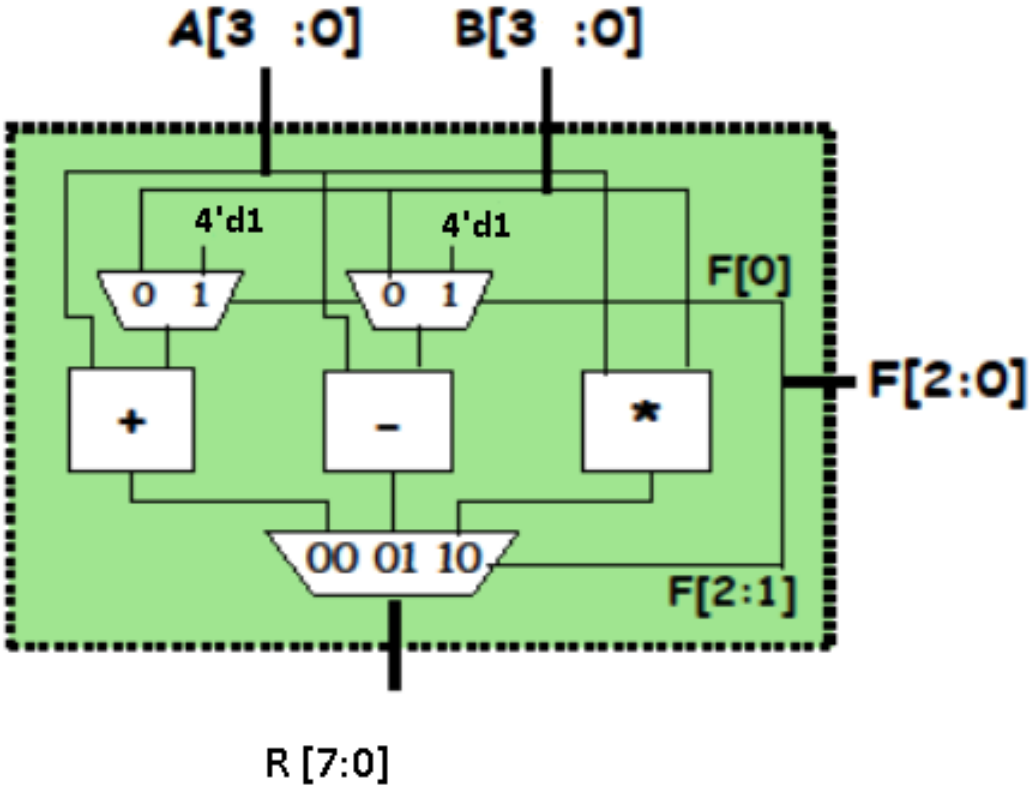
* L.Howard Pollard: Computer Desing and Architecture c. könyv függeléke (appendix)



Jelzőbitek:

- *carry-in, carry-out átviteleket,*
- *előjel bitet(sign),*
- *túlcsordulást(overflow),*
- *alulcsordulást(underflow).*

4-bites ALU – tervezése Verilog-ban



F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

Modulok leírása

```
module mux2_4(input [3:0] i0, i1, input sel, output [3:0] out);
          assign out = sel ? i1 : i0;
endmodule

module mux3_8(input [7:0] i0, i1, i2, input [1:0] sel, output reg [7:0] out);
     always @(i0 or i1 or i2 or sel)
          begin
               case (sel)
                    2'b00: out = i0;
                    2'b01: out = i1;
                    2'b10: out = i2;
                    default: out = 8'bx;
               endcase
          end
endmodule

module add4(input [3:0] i0, i1, output [7:0] sum);
          assign sum=i0+i1;
endmodule

module sub4(input [3:0] i0, i1, output [7:0] diff);
          assign diff=i0-i1;
endmodule

module mul4(input [3:0] i0, i1, output [7:0] prod);
          assign prod=i0*i1;
endmodule
```

ALU top modul leírása

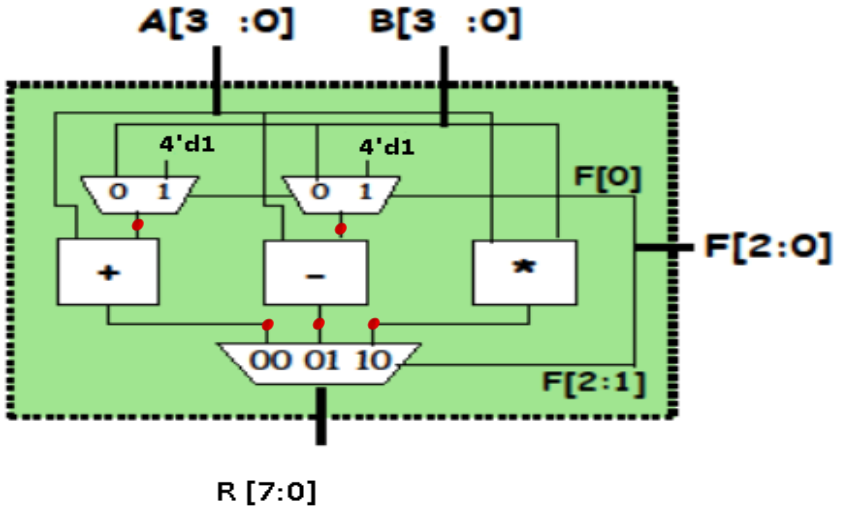
```
module alu (input [3:0] a, b, input [2:0] f, output [7:0] r);

wire [3:0] addmux_out, submux_out;
wire [7:0] add_out, sub_out, mul_out;

   mux2_4 adder_mux(b, 4'd1, f[0], addmux_out);
   mux2_4 sub_mux(b, 4'd1, f[0], submux_out);
   add4 our_adder(a, addmux_out, add_out);
   sub4 our_subtracter(a, submux_out, sub_out);
   mul4 our_multiplier(a, b, mul_out);
   mux3_8 output_mux(add_out, sub_out, mul out, f[2:1], r);

endmodule
```

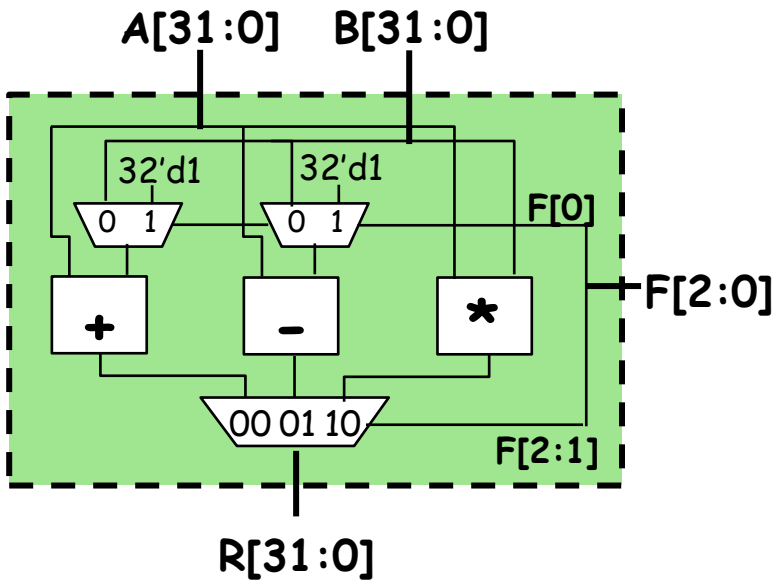
- 4 és 8 bites belső vezetékek



ALU modul viselkedési leírása

- Nem hierarhikus:

```
module alu(a, b, f, r);
  input [31:0] a, b;
  input [2:0] f;
  output [31:0] r;
  always @ (a or b or f)
    case (f)
      3'b000:  r = a + b;
      3'b001:  r = a + 1'b1;
      3'b010:  r = a - b;
      3'b011:  r = a - 1'b1;
      3'b100:  r = a * b;
      default: r = 32'bx;
    endcase
endmodule
```



- A szintézis után 2 összeadó és 2 kivonó lesz az eredmény, vagy mindegyikből egy-egy modul?