DIGITÁLIS TECHNIKA

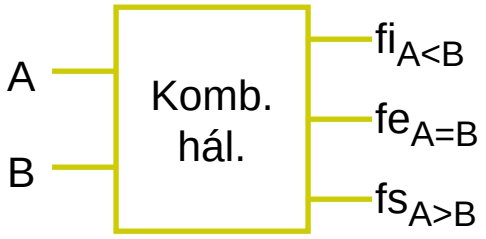
Dr. Oniga István

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.

Digitális komparátorok

- Két szám közötti relációt jelzi, (*egyenlő, kisebb, nagyobb)*.
- A három közül csak egy igaz

Egy bites komparátor

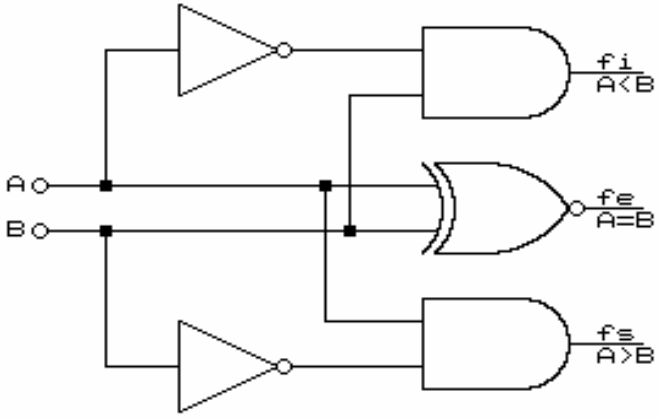


A	B	$fi_{A<B}$	$fe_{A=B}$	$fs_{A>B}$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$f_e = \overline{A \oplus B} = \overline{A \bullet \overline{B} + \overline{A} \bullet B} = A \bullet B + \overline{A} \bullet \overline{B}$$

$$f_i = \overline{A} \bullet B$$

$$f_s = A \bullet \overline{B}$$



Két bites komparátor I

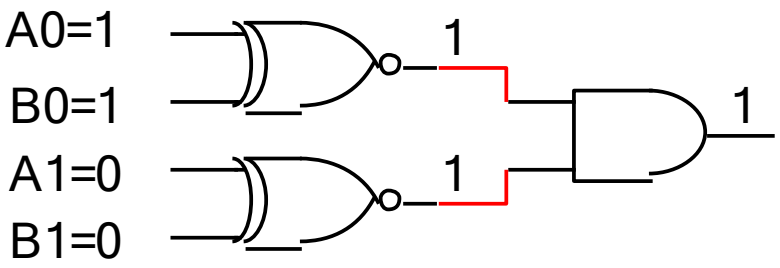
- Legyen a két szám:

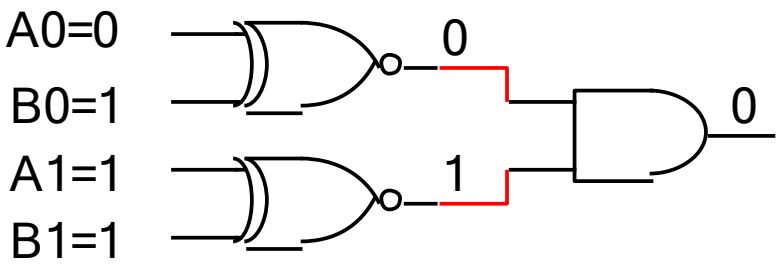
$$A = A_1 \cdot 2^1 + A_0 \cdot 2^0$$

$$B = B_1 \cdot 2^1 + B_0 \cdot 2^0$$

- Két bites számok akkor egyenlők, ha az azonos helyértékű bitek egyenlők

$$F_{A=B} = \overline{A_1 \oplus B_1} \bullet \overline{A_0 \oplus B_0} = (A_1B_1 + \overline{A_1}\,\overline{B_1})(A_0B_0 + \overline{A_0}\,\overline{B_0})$$





Két bites komparátor II

- Az egyenlőtlenségi relációkat jelző logikai függvények:

- A < B akkor igaz, ha A1 < B1,
  ill. ha A1 = B1 és A0 < B0

$$F_{A<B} = \overline{A_1} B_1 + (A_1 B_1 + \overline{A_1}\,\overline{B_1})\,\overline{A_0} B_0$$

- A > B akkor igaz, ha A1 > B1,
  ill. ha A1 = B1 és A0 > B0

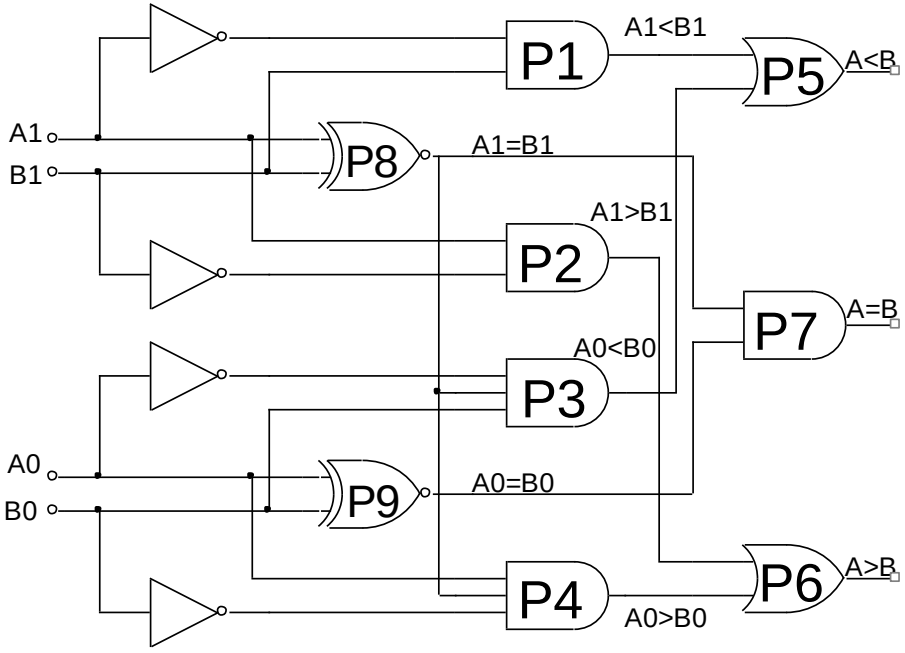
$$\mathrm{F_{A>B}} = \mathrm{A_1}\,\overline{\mathrm{B_1}} + (\mathrm{A_1 B_1} + \overline{\mathrm{A_1}}\,\overline{\mathrm{B_1}})\,\mathrm{A_0}\,\overline{\mathrm{B_0}}$$

Két bites komparátor III

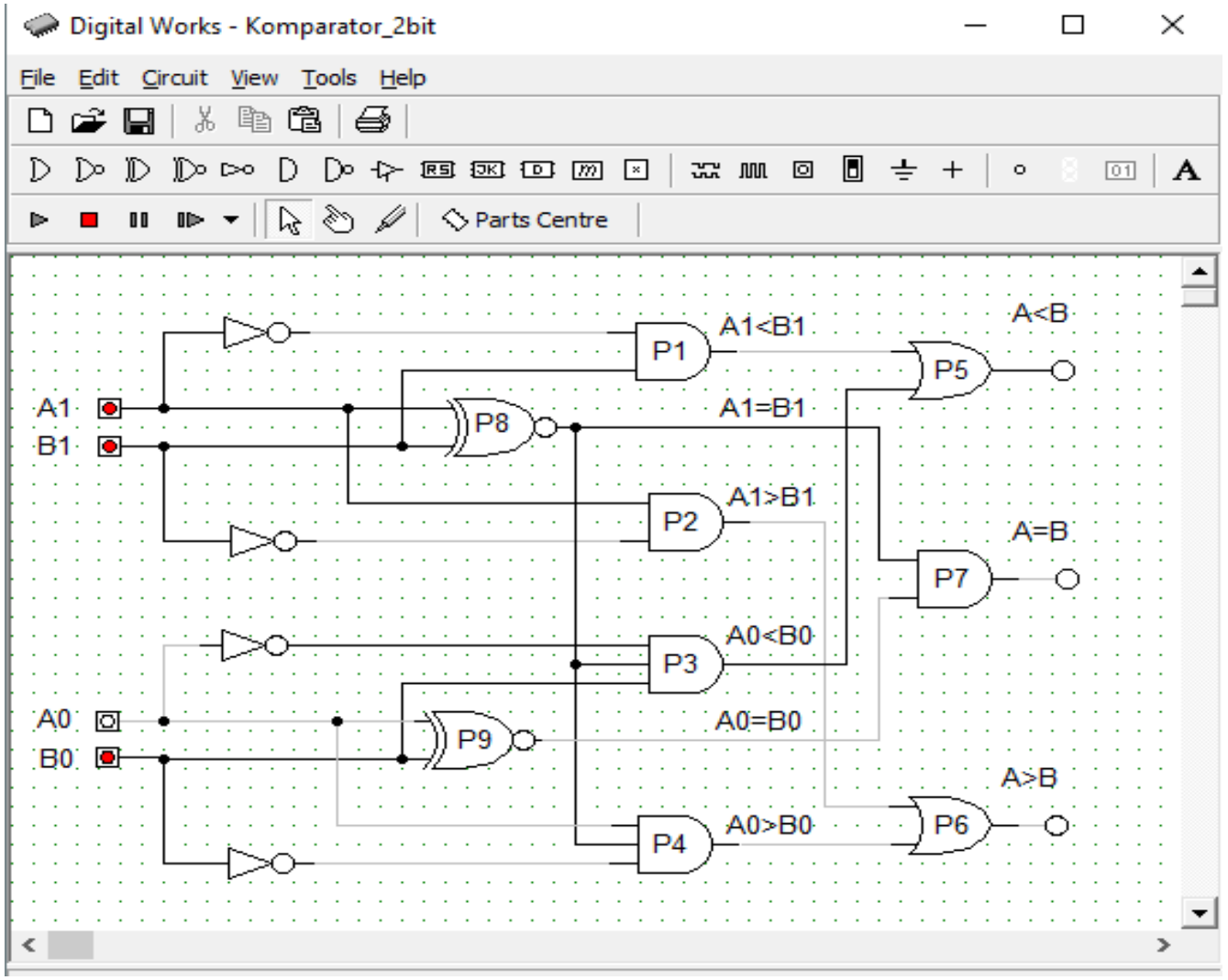
$$F_{A=B} = \overline{A_1 \oplus B_1} \bullet \overline{A_0 \oplus B_0}$$

$$F_{A>B} = A_1 \overline{B_1} + (A_1 B_1 + \overline{A_1}\,\overline{B_1}) A_0 \overline{B_0}$$

$$F_{A<B} = \overline{A_1} B_1 + (A_1 B_1 + \overline{A_1}\,\overline{B_1}) \overline{A_0} B_0$$

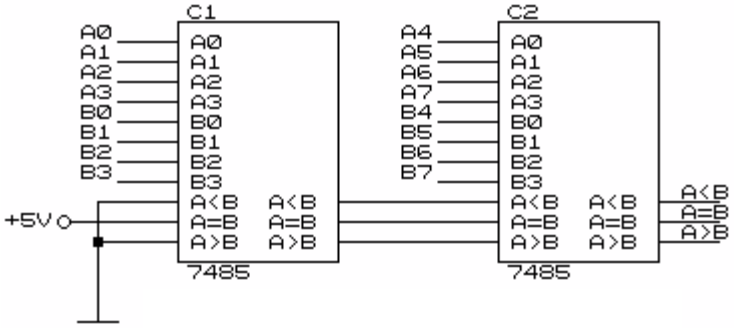


Két bites komparátor szimuláció



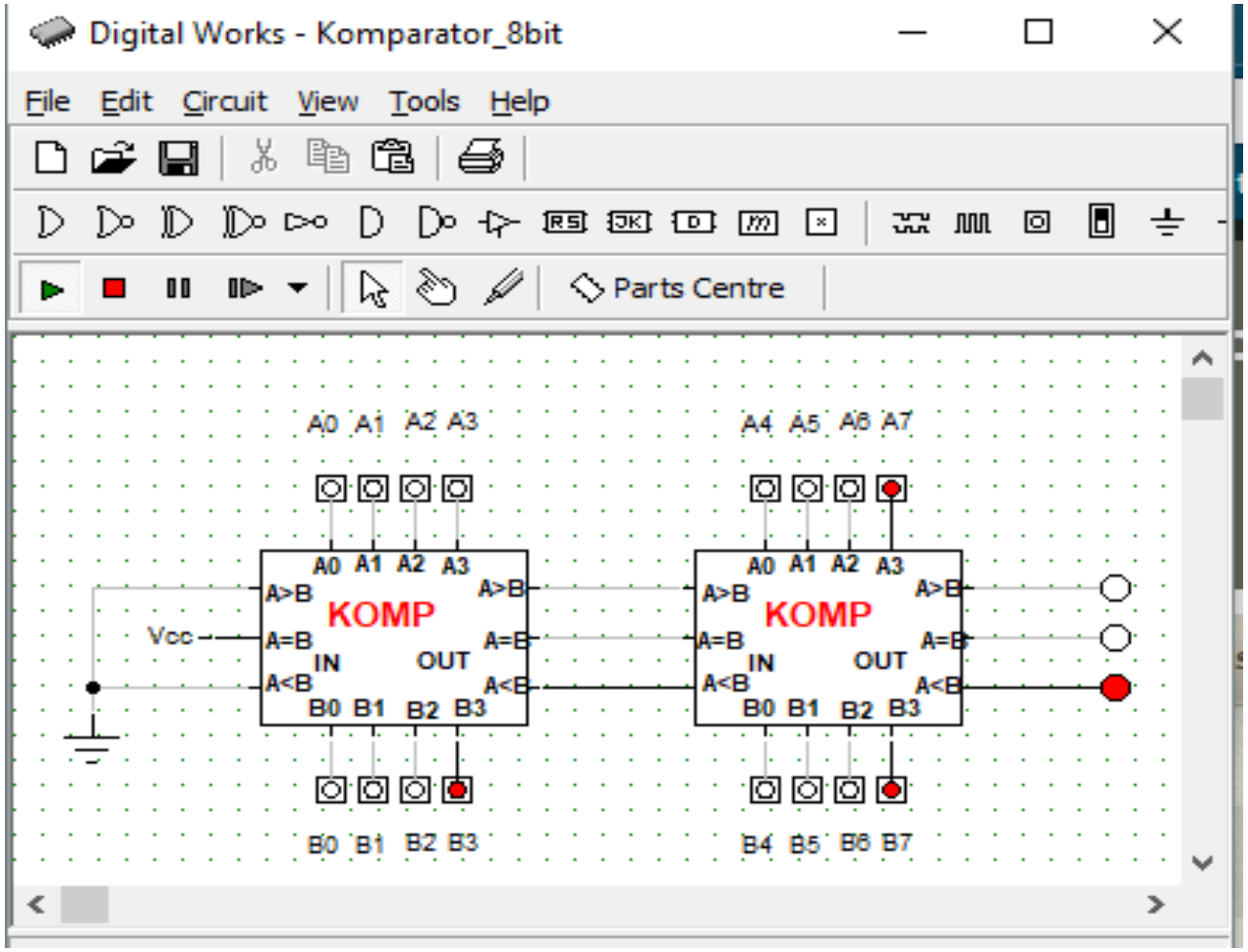
Több bites komparátorok

- Négybites nagyság komparátor
  - SN 7485 típusú áramkör
  - Bemenetei
    - a két összehasonlítandó szám bitjei (A_0, A_1, A_2, A_3 és B_0, B_1, B_2, B_3)
    - **a bővítő bemenetek** $A_i<B_i$, $A_i=B_i$, $A_i>B_i$, amelyekre az alacsonyabb helyértékű négy bit összehasonlításának eredményét kell adni.
  - Kimenetei a relációkat jelzik (A<B, A=B, A>B).
  - Komparátorok soros bővítése:



Több bites komparátorok szimuláció

A=1000 0000
B=1000 1000



Paritásképzés és ellenőrzés

HIBAFELISMERŐ ÉS HIBAJAVÍTÓ KÓDOK

Legegyszerűbb hibafelismerési eljárás:
- paritásbit átvitele

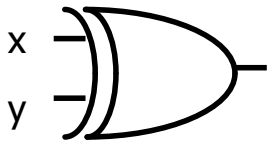
A paritáselemes kód elve - egy adott kód kódszavát kiegészítjük úgy, hogy a kiegészített kódszóban az 1-esek száma páros, vagy páratlan legyen.

- páros paritás
- páratlan paritás

Páros paritás		Páratlan paritás	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	0100	0	0100
0	1001	1	1001

A paritás képzés hátrányai:
– Nem tudjuk kijavítani a hibát, ha detektáljuk is
– Ha egyszerre több bit hibásodik meg, nem biztos, hogy a paritásellenőrzés felfedezi, mert lehet, hogy egyszerre két (vagy páros számú) bit is megváltoztatja értékét.

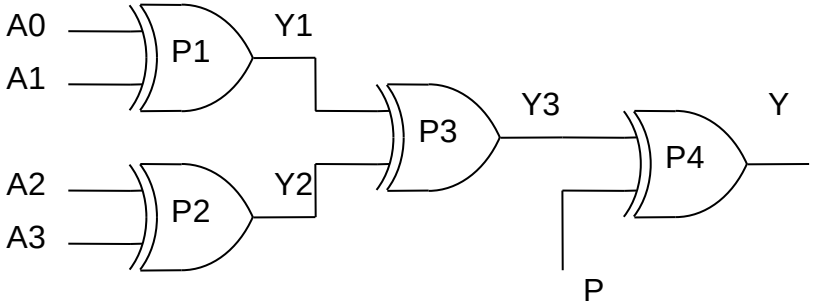
x y	x XOR y
0 0	0
0 1	1
1 0	1
1 1	0

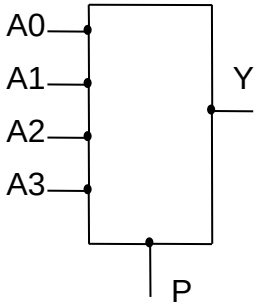


Legegyszerűbb paritásképző: XOR kapu

Paritásellenőrző áramkör

Paritásellenőrző áramkör 4 bites szavak részére





A_0	A_1	A_2	A_3	Y_1	Y_2	Y_3
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	0	0

P – Paritás típus beállító jel:

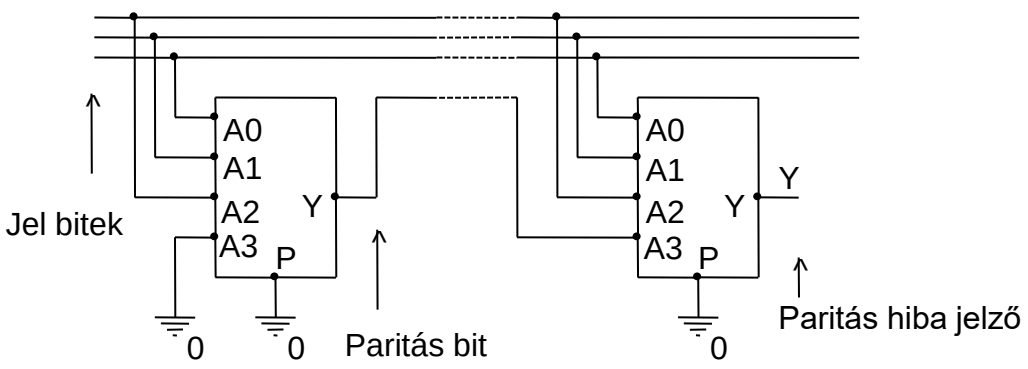
Y_3	P	Y
0	0	0
1	0	1
0	1	1
1	1	0

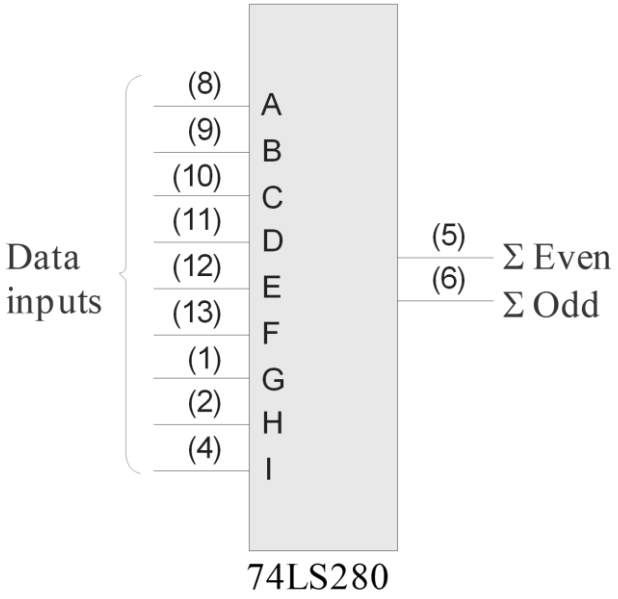
- P = 0 => Y = Y_3 => páros paritás generáló
- P = 1 => Y = $\overline{Y_3}$ => páratlan paritás generáló

Adatátviteli rendszer paritásellenőrzéssel

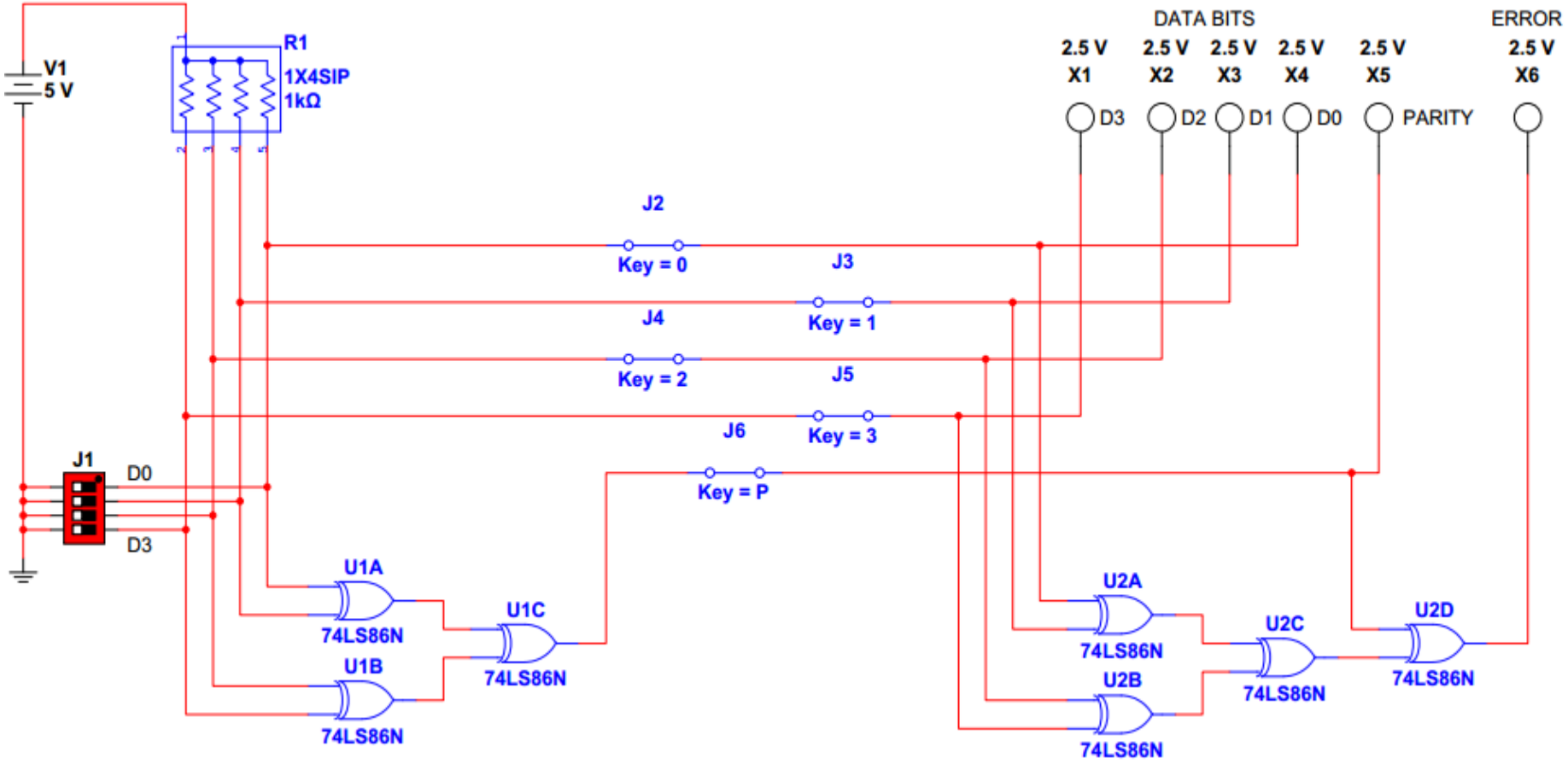
ADÓ – paritásgeneráló:
- Az adott jel biteket kiegészítjük a paritás bittel => a kiegészített kódszóban az 1-esek száma páros.

VEVŐ – paritásvizsgáló:
- A kiegészített kódszó - paritásellenőrzés



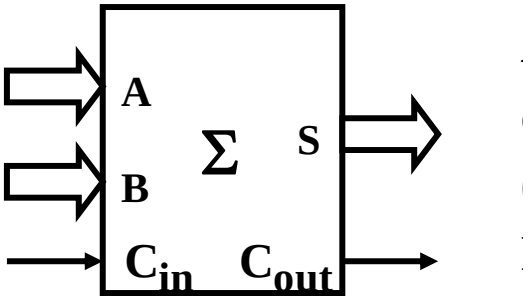


Adatátviteli rendszer paritásellenőrzéssel



Összeadok

- Az összeadó áramkör (adder)
  - bemenetek
    - A és B érkező számok
    - az előző helyérték átvitel (Cin-carry)
  - kimenetek
    - az összeg (S)
    - átvitel (Cout)



- Fél összeadók (half adder)
- Teljes összeadók (full adder)

Működési mód tekintetében:

- SOROS ÖSSZEADÓK
- PÁRHUZAMOS ÖSSZEADÓK

Az operandusok kódolását tekintve:

- BINÁRIS ÖSSZEADÓK
- BCD ÖSSZEADÓK

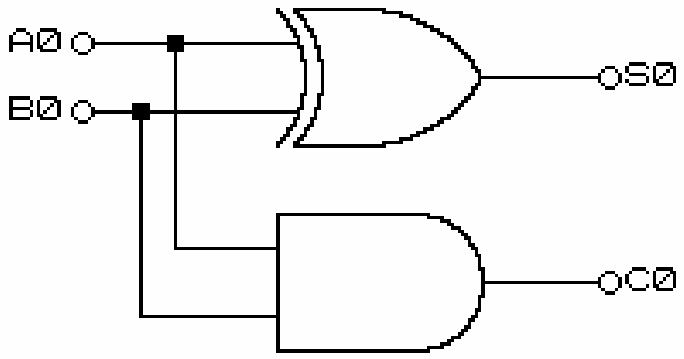
Fél összeadok

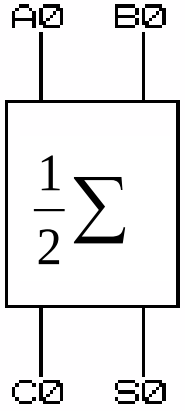
Nem veszik figyelembe az előző helyérték átvitelét
Csak a legkisebb helyértéken használható

A_0	B_0	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S_0 = \overline{A}_0 \bullet B_0 + A_0 \bullet \overline{B}_0 = A \oplus B$$

$$C_0 = A_0 \bullet B_0$$





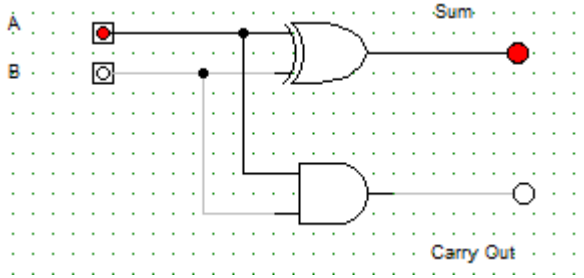
Fél összeadok

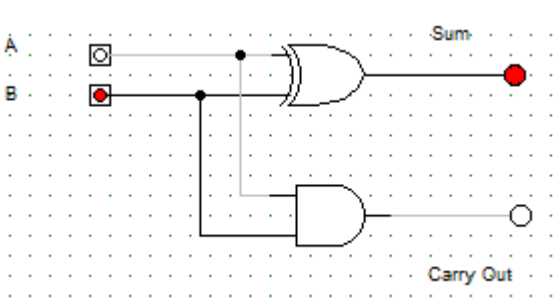
Nem veszik figyelembe az előző helyérték átvitelét
Csak a legkisebb helyértéken használható

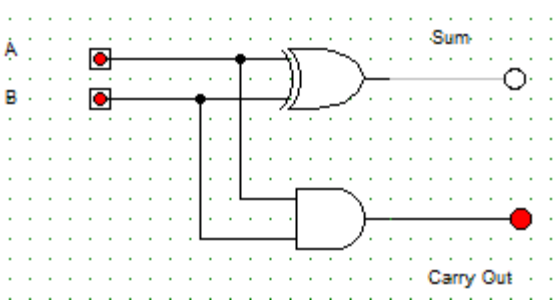
A_0	B_0	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S_0 = \overline{A}_0 \bullet B_0 + A_0 \bullet \overline{B}_0 = A \oplus B$$

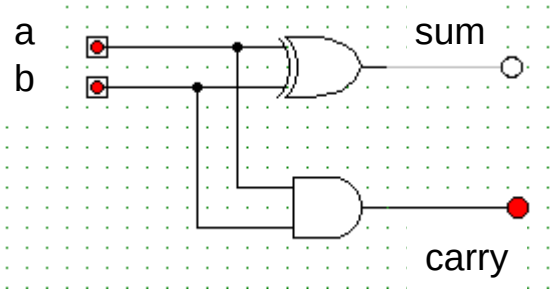
$$C_0 = A_0 \bullet B_0$$



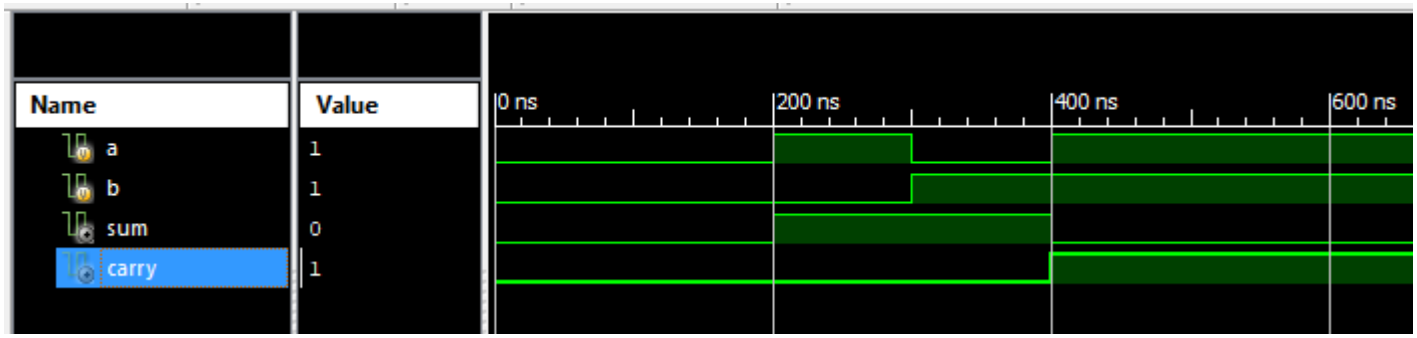




1-bites fél összeadó Verilog strukturális modellje




```
module half_add (output sum, carry, input a, b);
        xor (sum, a, b);     // exclusive OR
        and (carry, a ,b);   // AND
endmodule
```



Teljes összeadok

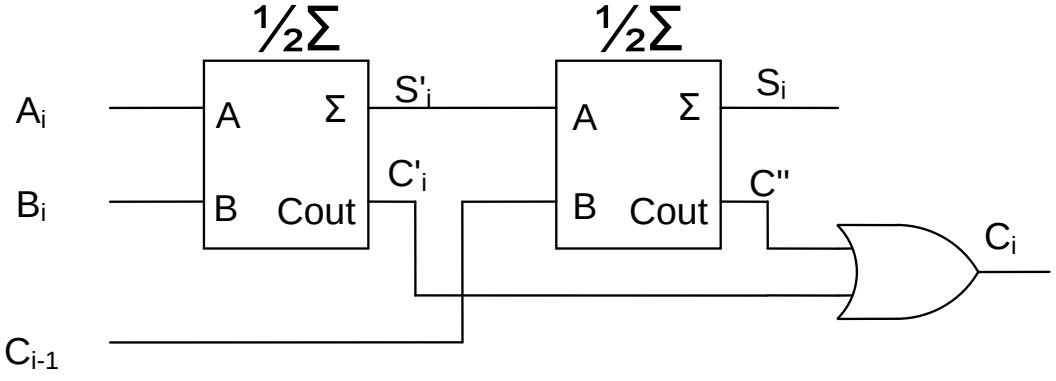
Figyelembe veszik az előző helyérték átvitelét

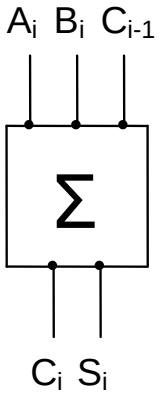
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

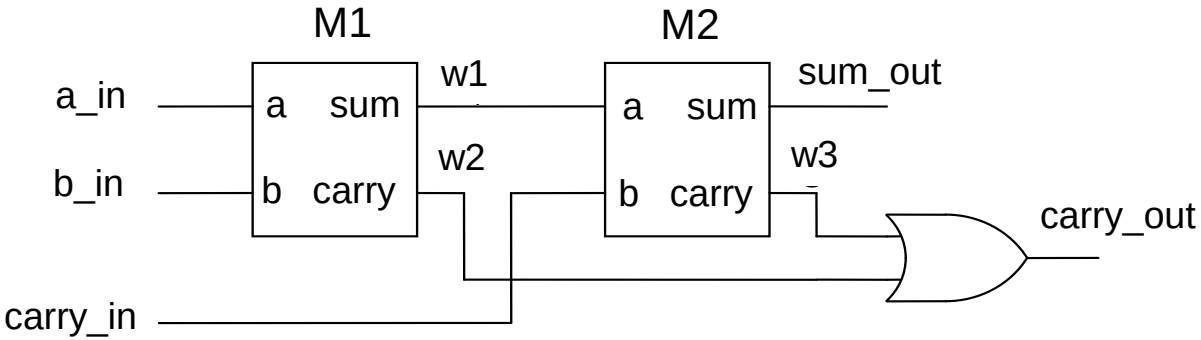
$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

Teljes összeadó két félösszeadóból:





1-bites teljes összeadó Verilog strukturális modellje



Névszerinti összekötés

```
module full_add (output sum_out, carry_out, input a_in, b_in, carry_in );

wire w1, w2, w3;

        half_add M1 (.a(a_in), .sum(w1), .b(b_in), .carry(w2));
        half_add M2 (.sum(sum_out), .a(w1), .carry(w3), .b(carry_in));

        or (carry_out, w2, w3);
endmodule
```

Teljes összeadok

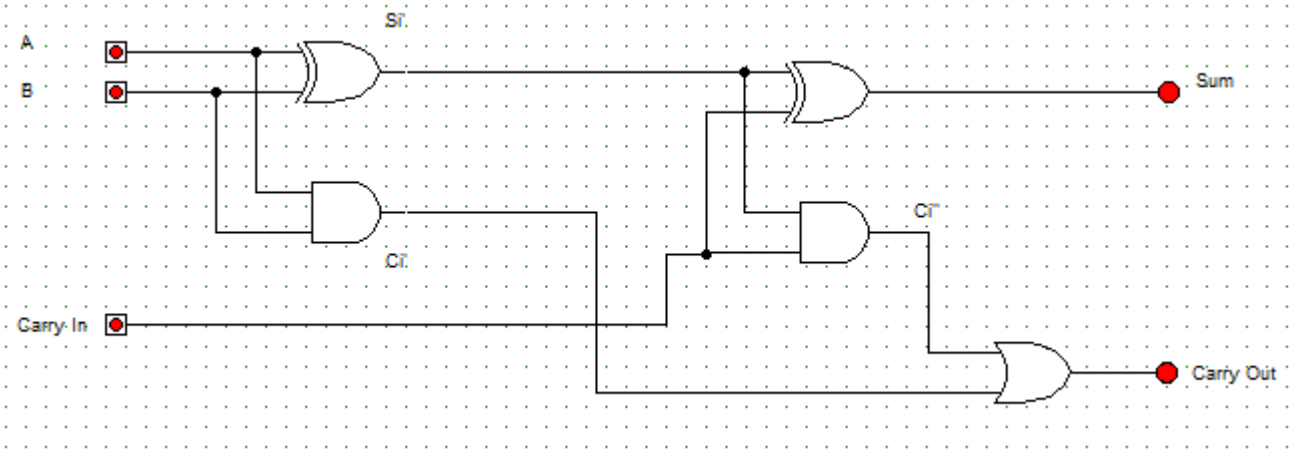
Figyelembe veszik az előző helyérték átvitelét

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

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

Teljes összeadó két félösszeadókból:



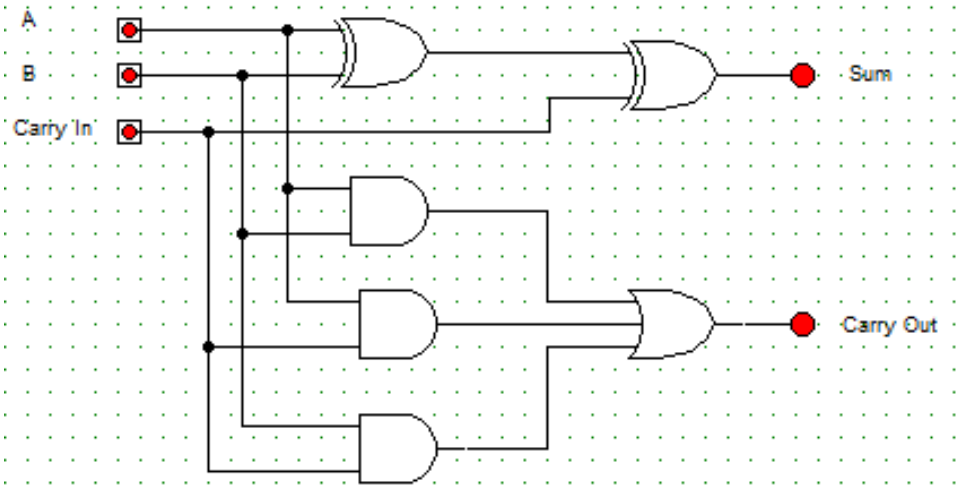
Teljes összeadok

Figyelembe veszik az előző helyérték átvitelét

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

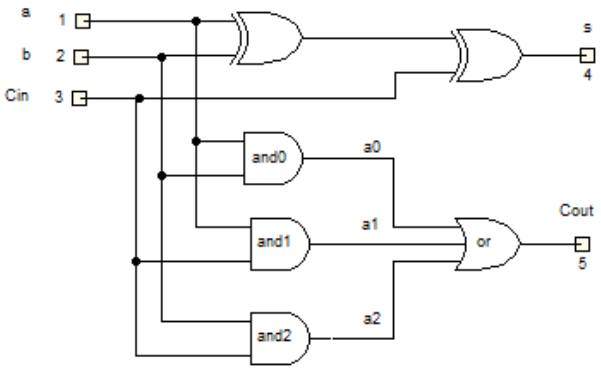
$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

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



Lab6_4: 1-bites teljes összeadó leírása

```
// A verzió
module add1_full (input a, b, cin, output cout, s);
xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
wire a0, a1, a2;
and2_m and0(.i0(a), .i1(b), .o(a0));
and2_m and1(.i0(a), .i1(cin), .o(a1));
and2_m and2(.i0(b), .i1(cin), .o(a2));
or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```



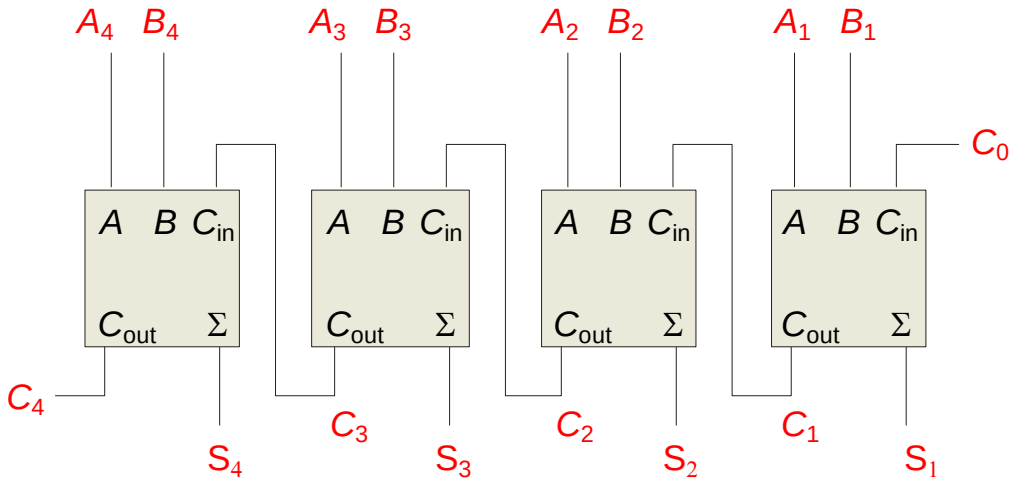

```
// B verzió
module add1_full (input a, b, cin, output cout, s);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

```
// C verzió
module add1_full (input a, b, cin, output cout, s);
assign {cout, s} = a + b + cin;
endmodule
```

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

4 bites soros átvitelű összeadó

- Funkcionális kódrészlet

```
// 5_5a 4 bites soros átvitelű összeadó
module add4 (input [3:0] a, b, output [4:0] s);
wire [3:0] c;
add1_full add0(.a(a[0]), .b(b[0]), .cin(1'b0),    .cout(c[0]), .s(s[0]));
add1_full add1(.a(a[1]), .b(b[1]), .cin(c[0]), .cout(c[1]), .s(s[1]));
add1_full add2(.a(a[2]), .b(b[2]), .cin(c[1]), .cout(c[2]), .s(s[2]));
add1_full add3(.a(a[3]), .b(b[3]), .cin(c[2]), .cout(s[4]),    .s(s[3]));
endmodule
```

4 bites összeadó viselkedési leírás

```
//  5_5b
module add4 (input [3:0] a, b, output [4:0] s);
assign s = a + b;
endmodule
```

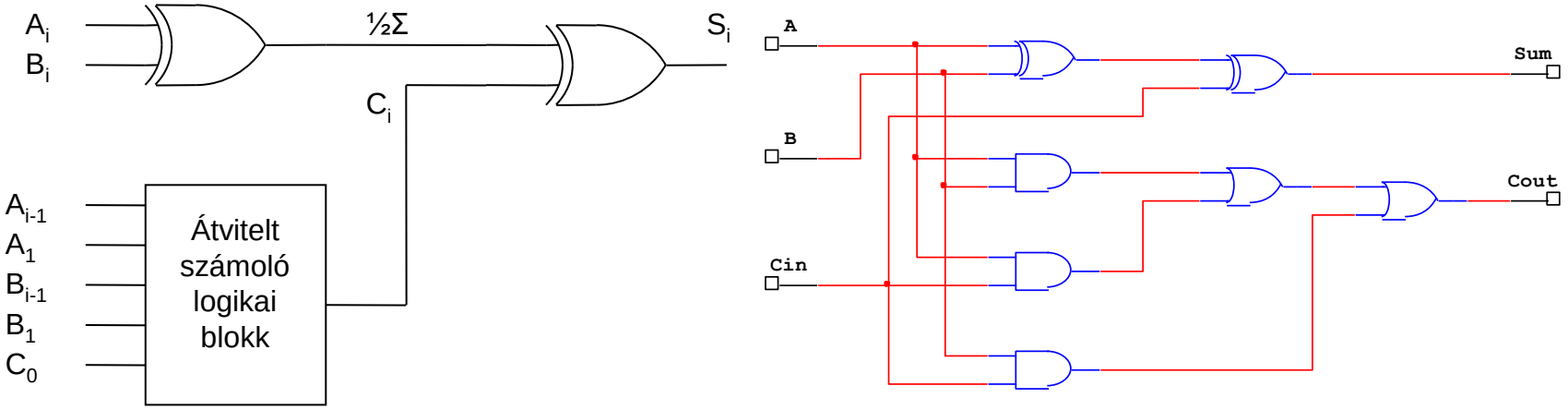
```
module add4 (input [3:0] a, b, input cin, output cout, output [3:0] sum);
assign {cout, sum} = a + b + cin;
endmodule
```

Több bites összeadók II

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

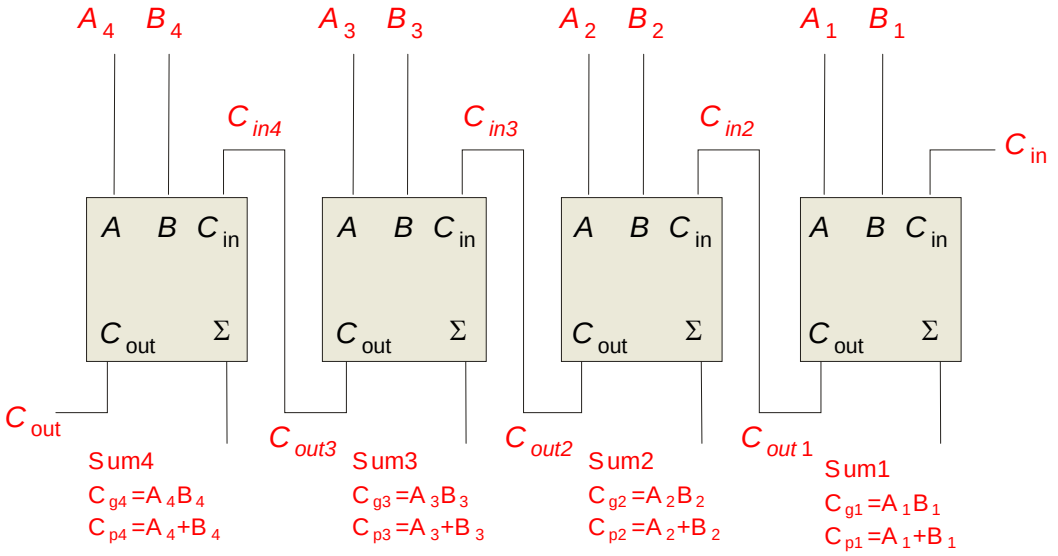
$$C_i = A_iB_i + (A_i + B_i)C_{i-1} = (A_iB_i + A_iC_{i-1}) + B_iC_{i-1}$$

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

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

$C_{in2}=C_{out1}$

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

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

$C_{in3}=C_{out2}$

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

$C_{in4}=C_{out3}$

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