Enostavna dvobitna ALE

Gregor Jeraj, Urša Levičnik, Zahir Mujanović, Sašo Skube

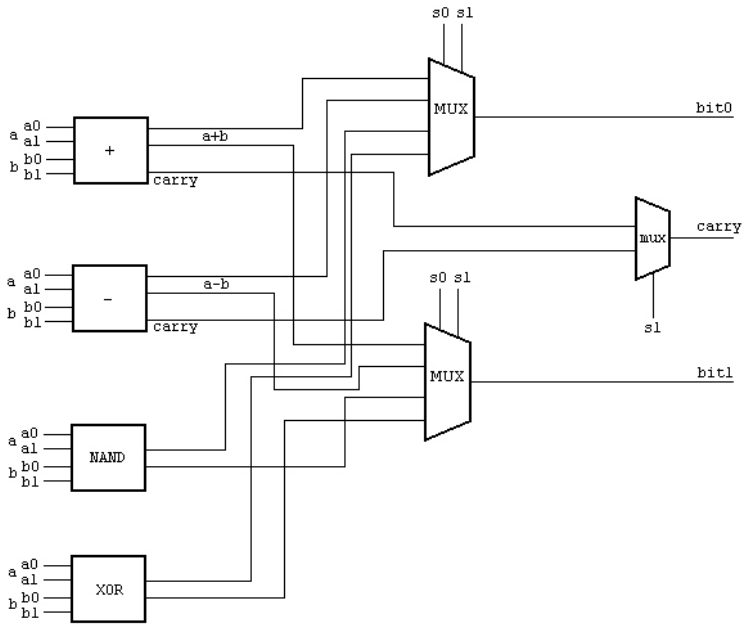
27. november 2008

1 Uvod

Za seminarsko nalogo smo v QCA Designer-ju realizirali enostavno dvobitno aritmetično logično enoto. Upoštevati smo morali, da zna ta ALE seštevati in odštevati dve dvo-bitni števili, zna realizirati poljubno dvo-vhodno dvojiško funkcijo, ima overflow zastavico in kontrolni vhod.

2 Realizacija

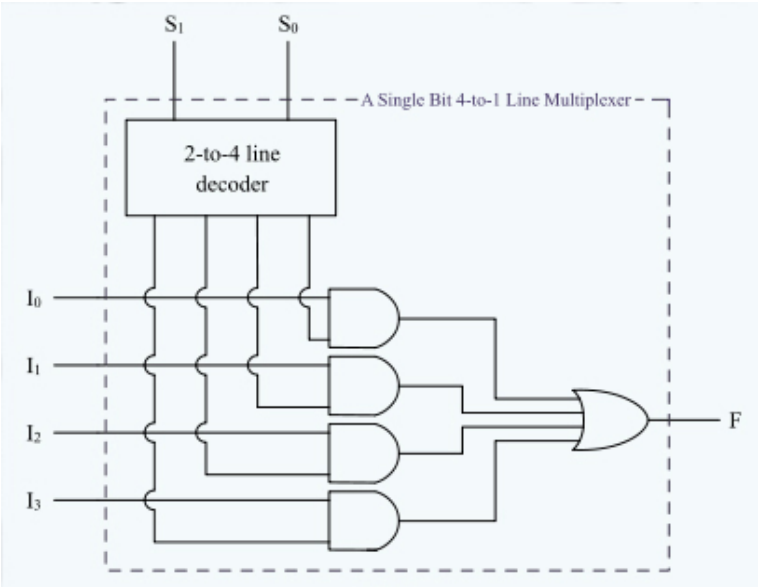
ALE enoto smo realizirali na način kot kaže slika 9. Posebej smo realizirali določene komponente, ki se pojavljajo v vezju, nato pa smo jih združili v enotno vezje.



Slika 1: Aritmetično logična enota

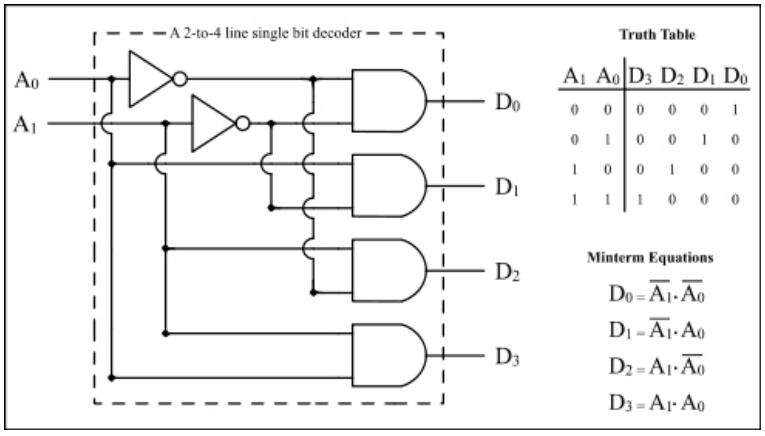
2.1 Multiplekser

Multiplekser (dalje MUX) je enostaven element, ki glede na kontrolna vhoda izbere enega izmed vhodnih signalov. Prikazan je na sliki 2



Slika 2: Multiplekser

Kot kaže slika je treba v multiplekser vgraditi tudi dekoder. Realizirali smo ga kot kaže slika 3:



Slika 3: Dekoder

MUX smo potrebovali za izbiro funkcij, ki jih opravlja ALE. Za izbiro med štirimi vhodi imamo na voljo dva kontrolna vhoda $s0$ in $s1$. Ker imamo 2-bitni seštevalnik in odštevalnik, moramo uporabiti 2 MUX-a, da izberemo 2 bita rezultata teh dveh operacij, ter še en MUX, ki izbere carry bit. Tretji MUX je kar 2/1, saj logični funkciji carry bita ne potrebujeta in ju zvežemo le čez dva

4/2 MUX-a, čez katera smo pripeljali tudi seštevalnik in odštevalnik.

Kaj ALE dela izbiramo s select bitoma s0 in s1 in sicer tako:

s0 s1	funkcija
0 0	seštevanje
0 1	odštevanje
1 0	NAND
1 1	XOR

2.2 Seštevalnik in odštevalnik

Seštevalnik in odštevalnik sta sicer ista stvar, razlika pa je v pridobivanju carry-ja: *notAandB* za odštevalnik, *AandB* za seštevalnik. Če imamo A1 A0 in B1 B0 kot vhode A in B, želimo pa dobiti A-B, se izvede naslednje:

```
R0 = A0 - B0; C0 = carry te operacije
R0 = A0 XOR B0; C0 = (NOT A0) AND B0

R1v = A1 - B1; C1 = carry te operacije
R1v = A1 XOR B1; C1 = (NOT A0) AND B0

R1 = R1v - C0; Cv = carry te operacije
R1 = R1v XOR C0; Cv = (NOT R1v) AND C0

C = koncni carry
C = C1 OR Cv

Izhodi so torej:
C, R1, R0
```

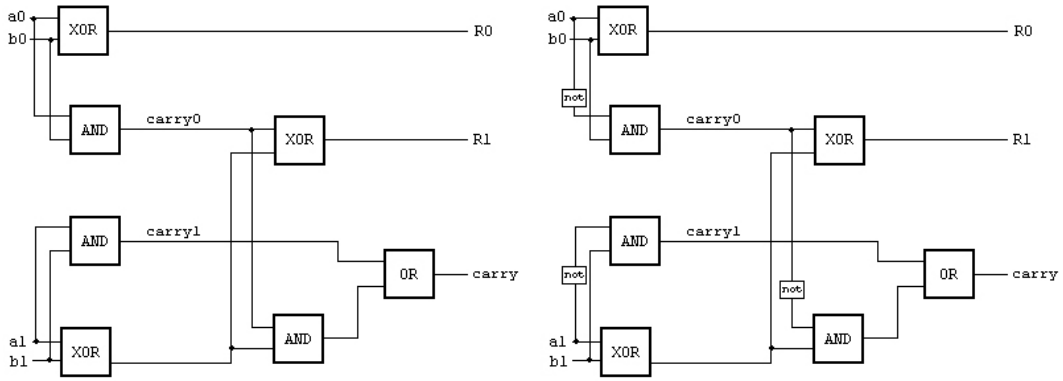
Kot primer:

```
10 - 11 (koncni rezultat je 11 s carryjem)

Vhodi: A1 = 1, A0 = 0, B1 = 1, B0 = 1;

R0 = 0 XOR 1 = 1; C0 = (NOT 0) AND 1 = 1
R1 = 1 XOR 1 = 0; C1 = (NOT 1) AND 1 = 0
R1 = 0 XOR 1 = 1; Cv = (NOT 0) AND 1 = 1

C = 0 OR 1 = 1

Izhodi so torej:
C=1, R1=1, R0=1

Kar je pravilno - R = 11, carry = 1.
```

Tole je pravilnostna tabela seštevalnika in odštevalnika:

seštevalnik				odštevalnik			
a	b	add	carry	a	b	sub	carry
00	00	00	0	00	00	00	0
00	01	01	0	00	01	11	1
00	10	10	0	00	10	10	1
00	11	11	0	00	11	01	1
01	00	01	0	01	00	01	0
01	01	10	0	01	01	00	0
01	10	11	0	01	10	11	1
01	11	00	1	01	11	10	1
10	00	10	0	10	00	10	0
10	01	11	0	10	01	01	0
10	10	00	1	10	10	00	0
10	11	01	1	10	11	11	1
11	00	11	0	11	00	11	0
11	01	00	1	11	01	10	0
11	10	01	1	11	10	01	0
11	11	10	1	11	11	00	0

V QCADesigner-ju smo seštevalnik in odštevalnik realizirali po logičnih shemah na sliki 4



Slika 4: Logično vezje seštevalnika in odštevalnika

Zaradi lažje realizacije smo obe aritmetični funkciji naredili ločeno, nato pa prek MUX-ov izbirali katero smo uporabili. Prav tako smo morali to narediti s carry bitom. Če se pri seštevanju ali odstevanju pojavi overflow, potem naredimo intuitivno: če overflow je, potem je carry bit 1, drugače je 0. Kdaj se bit postavi vidimo tudi iz zgornje pravilnostne tabele seševalnika in odštevalnika.

2.3 Logični funkciji

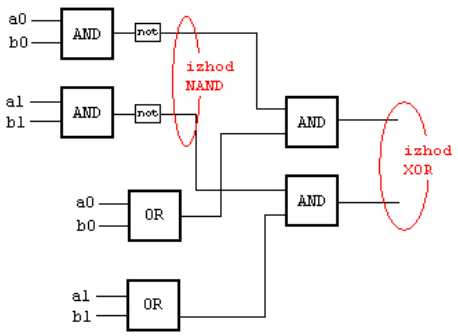
V vezju smo realizirali še logični funkciji XOR in NAND, da smo izkoristili prostor.
Pravilnostna tabela funkcije NAND:

a b	a nor b
0 0	1
0 1	1
1 0	1
1 1	0

Pravilnostna tabela funkcije XOR:

a b	a xor b
0 0	0
0 1	1
1 0	1
1 1	0

Naredili smo tako:



Slika 5: Logična shema realizacije funkcij NAND in XOR

3 Komentar

Kot se vidi iz našega dela smo pri izdelavi vezja izhajali kar iz CMOS logike. Najprej smo si narisali logično vezje ALE enote, nato pa realizirali vsak element posebej, ter to na koncu združili, kar nam je povzročalo nemalo težav, saj samo okolje v katerem smo delali ni ravno uproabniku prijazno.

Problem, pri tem, da realiziramo vezje v taki obliki, kot ga že poznamo, pa vendar pod popolnoma drugačnimi pogoji, nastane z novo tehnologijo s katero smo se spoznali pri tej seminarski nalogi. Ker gre tukaj za majhne energije in delce, ki hitro reagirajo na kakršenkoli dodaten dražljaj iz okolja je potrebno biti pazljiv, da se sosednji elementi (linije) ne motijo med seboj, torej morajo biti tudi fizično dovolj narazen. Tako smo prišli to problema dolgih linij in posledično zakasnitev. Linijo je potrebno razbiti na urine cikle, saj drugače vezje ne deluje več pravilno. Posledica razbijanja vezja na različne urine cikle je zakasnitev vezja, ki je pri nas 15 urinih ciklov. Tukaj lahko povemo še, da tudi ura ni več

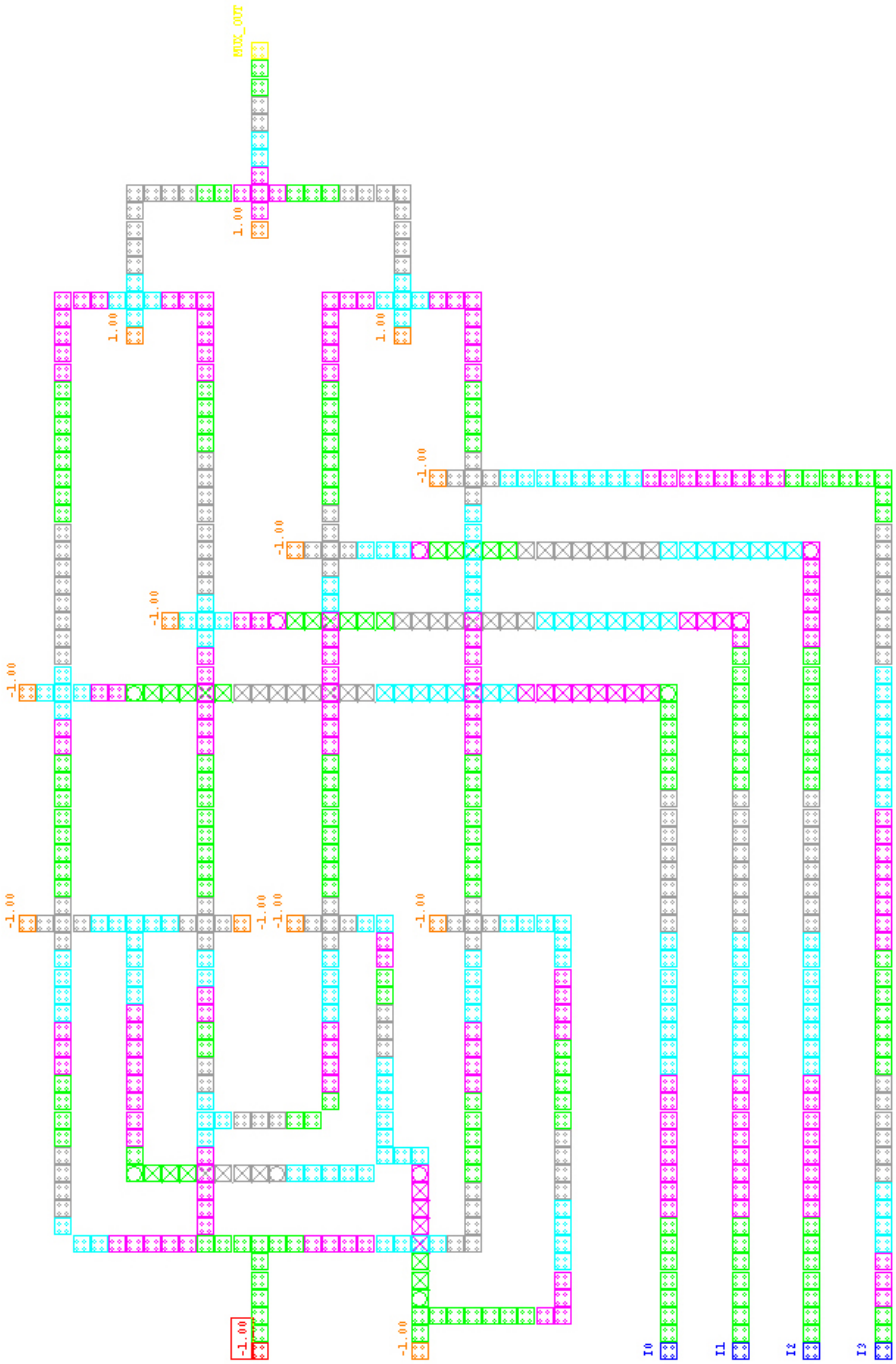
taka ura, kot jo poznamo iz CMOS vezij, ampak služi kot močnostno ojačanje. To potrebujemo zaradi neidealnosti kvantnih struktur (izgube v daljših linijah). Zato moramo nadomestiti to izgubljeno energijo z električnim poljem. Ura ima 4 faze: preklop, zadrževanje, sproščanje, sproščenost. Ločimo jih s pregradami. Pri našem vezju smo imeli maksimalno 8 celic na eno uro.

Problem ni samo v linijah, temveč tudi v večplastnem križanju, kjer si je prav tako potrebno vzeti dovolj maneverskega prostora, da vezje deluje, kar pa je zopet potrata celic.

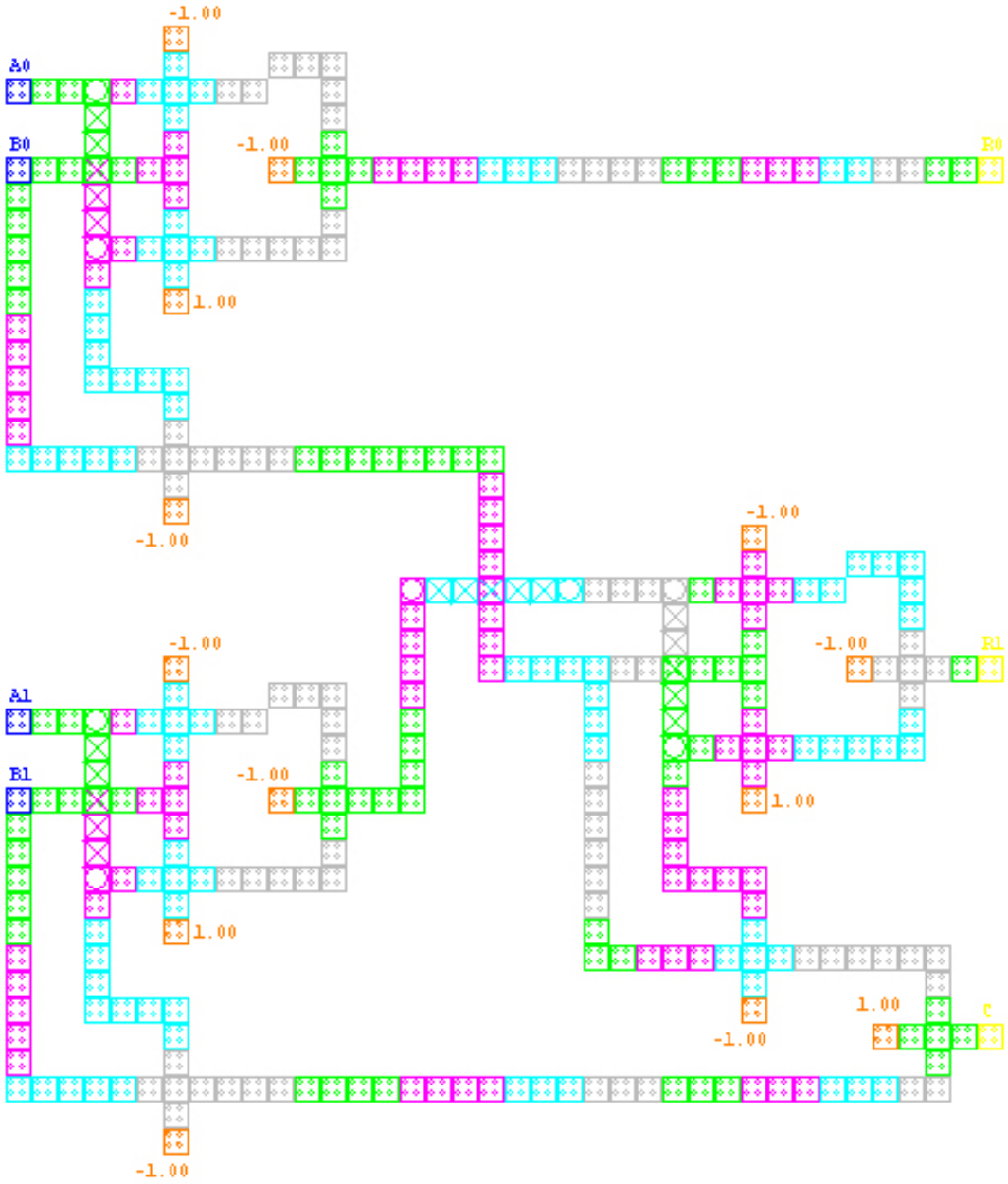
Ča bi se lotili izdelave tako zahtevnega vezja na popolnoma drugačen pristop, kot ga poznamo, mislimo, da ne bi dokončali seminarske naloge v zglednem času, zato smo z našimi rezultati zadovoljni. Vseeno pa smo po izdelavi seminarske naloge iz področja kvantnih celularnih avtomatov zelo radovedni, kaj nam bo prinesla prihodnost.

4 Realizacije v QCADesigner-ju

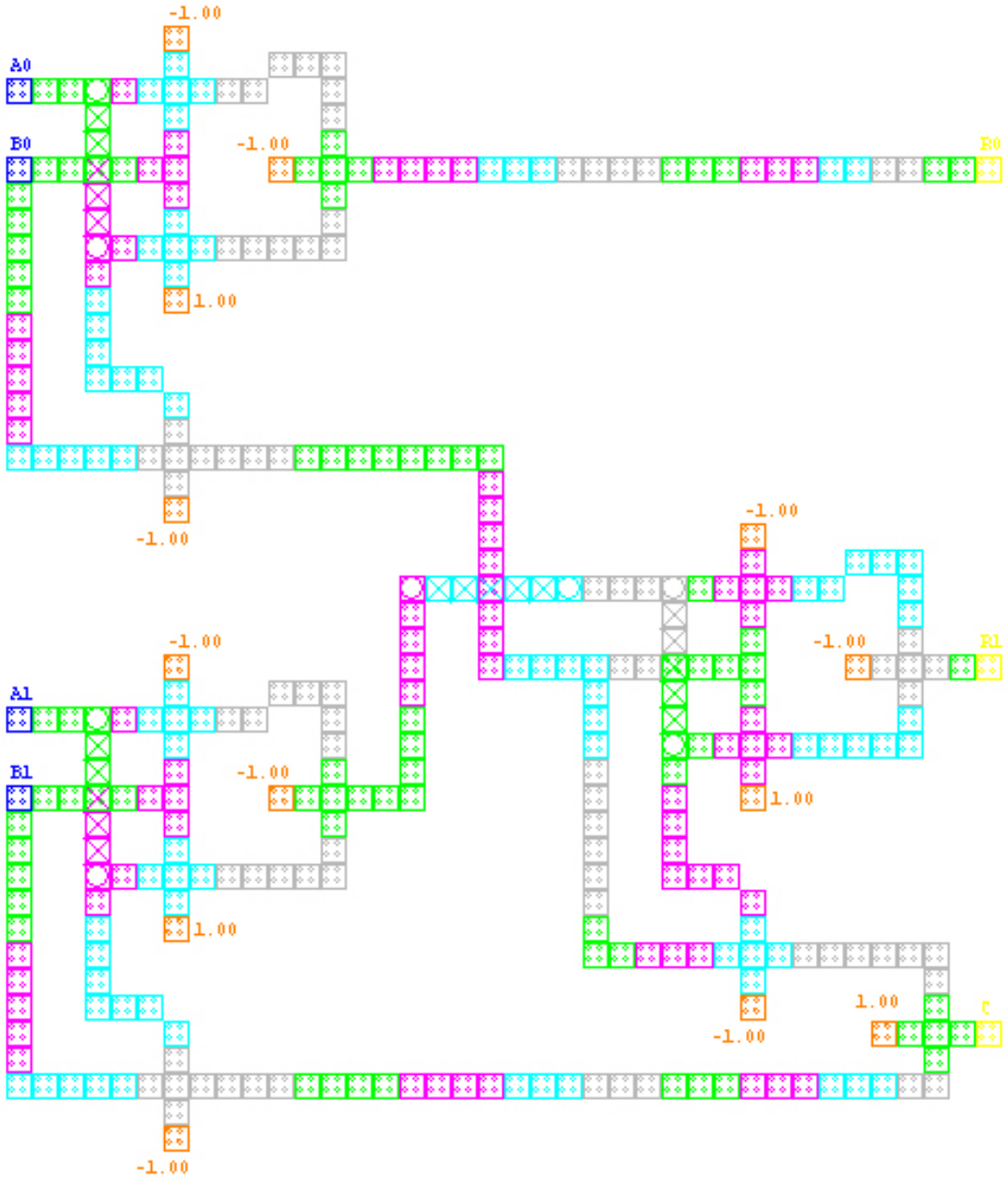
Tukaj so podane realizacije vezja v QCADesigner-ju.



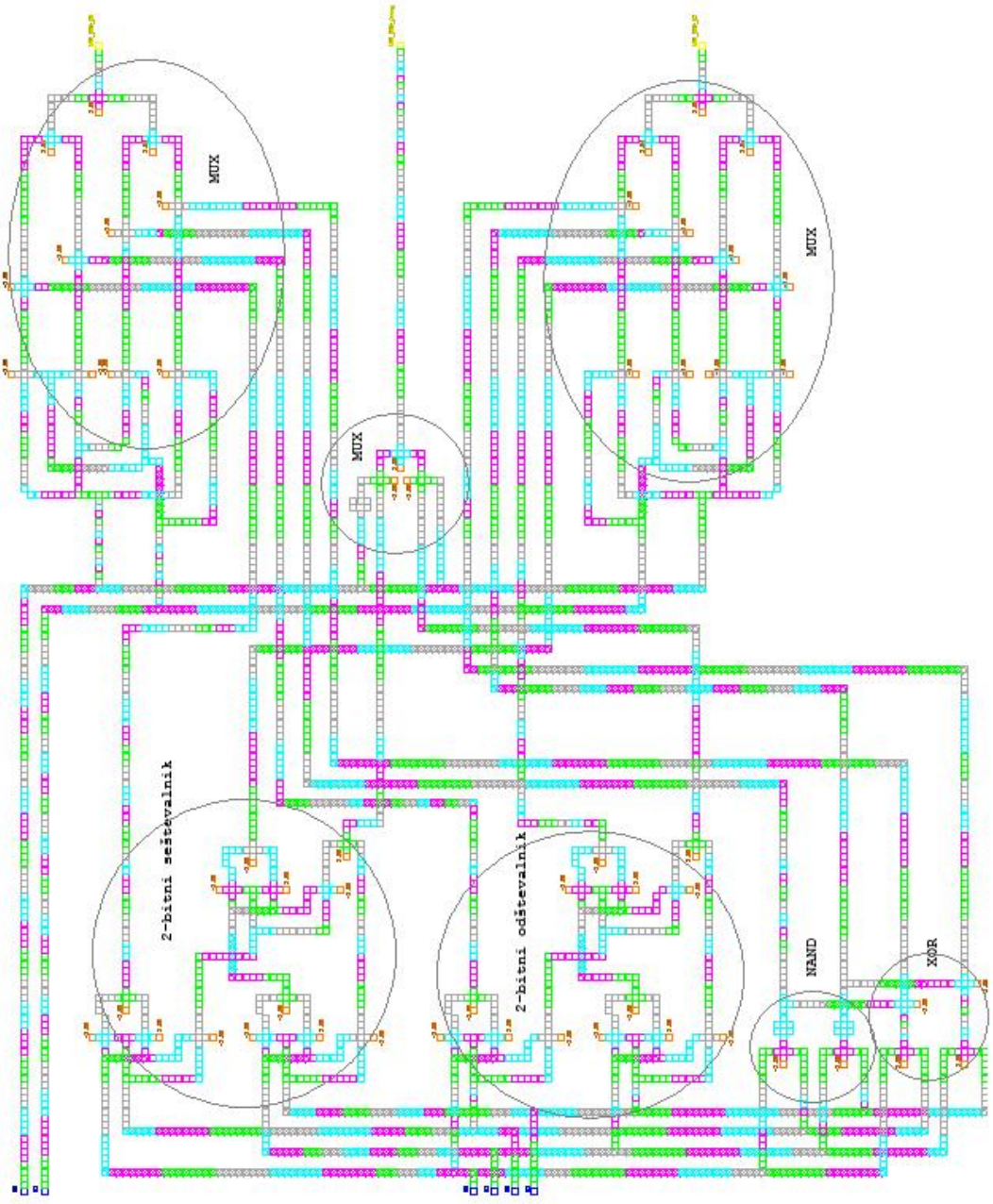
Slika 6: Multiplekser



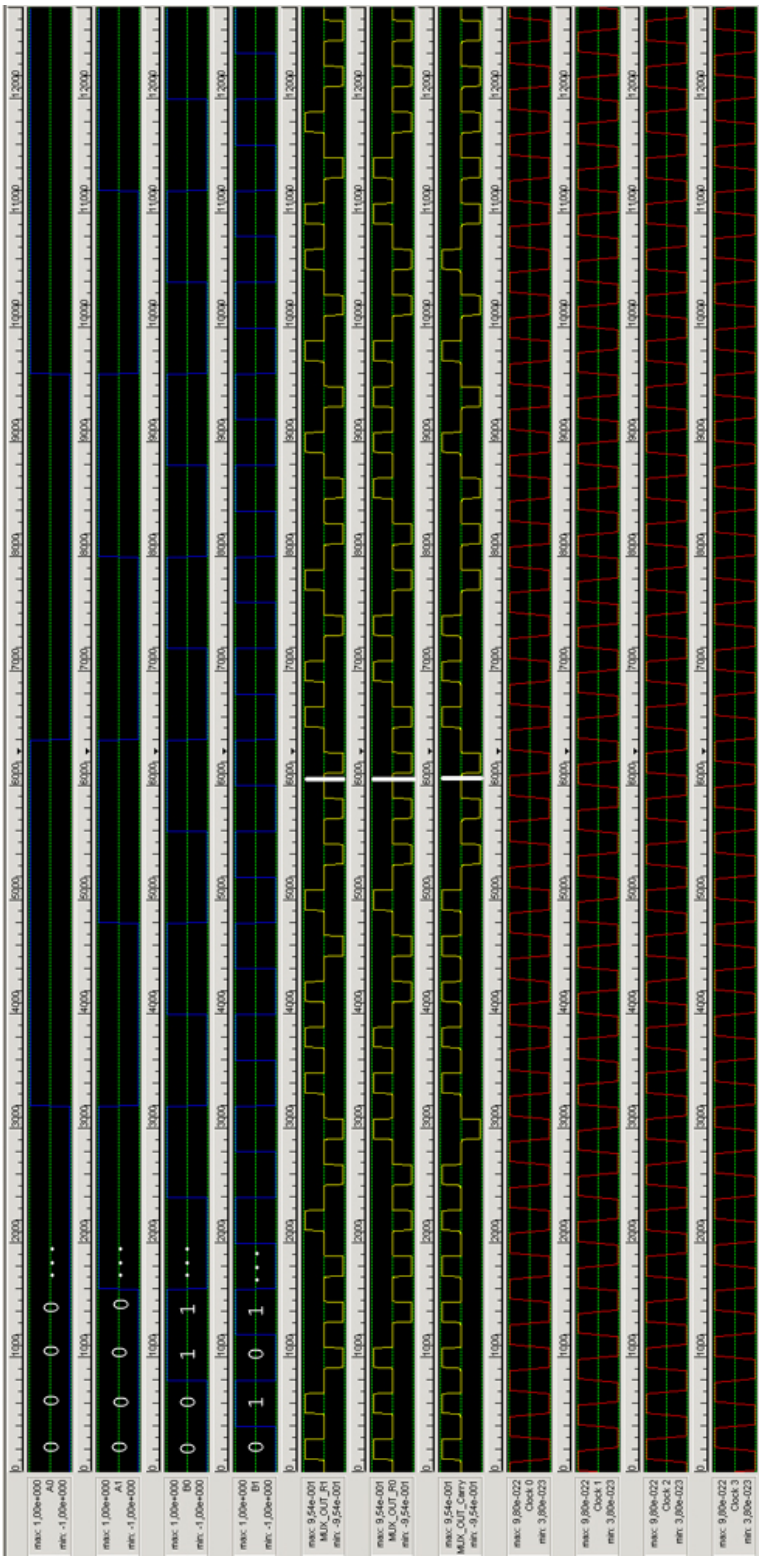
Slika 7: Seštevalnik



Slika 8: Odštevalnik



Slika 9: Celotno vezje



Slika 10: Primer rezultatov simulacije za odštevanje dveh števil. Ker ima vezje zakasnitve se gledajo vrednosti izhodov šele po 15 zakasnitvah (za označenim - bela črta)