VISOKA TEHNIČKA ŠKOLA

STRUKOVNIH STUDIJA

SUBOTICA



**ALU sa 4 bita**

**Bez namenskih kola**

projekat

iz predmeta: Arhitektura računara

mentor: Dr. Tibor Sakal kandidat :David Katrinka

profesor strukovnih studija br.indeksa : 26123035 studijski program: Informatika

**Sadržaj**

[UVOD 2](#_Toc187581146)

[**1.** **PROJEKTNI ZADATAK** 3](#_Toc187581147)

[1.1 Svrha projekta i ograničenja 3](#_Toc187581148)

[1.2 Način realizacije 3](#_Toc187581149)

[**2.** **TEORIJSKE OSNOVE** 4](#_Toc187581150)

[2.1 Aritmetički deo 4](#_Toc187581151)

[*2.1.1 Modul za sabiranje* 4](#_Toc187581152)

[*2.1.2 Modul za oduzimanje* 5](#_Toc187581153)

[*2.1.3 Modul za inkrement* 6](#_Toc187581154)

[*2.1.4 Modul za dekrement* 7](#_Toc187581155)

[*2.1.5 Modul za drugi komplement* 8](#_Toc187581156)

[2.2 Logički deo 9](#_Toc187581157)

[*2.2.1 NOT modul* 9](#_Toc187581158)

[*2.2.2 AND modul* 10](#_Toc187581159)

[*2.2.3 OR modul* 11](#_Toc187581160)

[*2.2.4 NAND modul* 12](#_Toc187581161)

[*2.2.5 NOR modul* 12](#_Toc187581162)

[*2.2.6 XOR modul* 13](#_Toc187581163)

[2.3 Komparator 14](#_Toc187581164)

[2.4 Dodatni moduli 15](#_Toc187581165)

[*2.4.1 Overflow modul* 15](#_Toc187581166)

[*2.4.2 Buffer modul* 16](#_Toc187581167)

[*2.4.2 Dekoder sa negativnom logikom* 17](#_Toc187581168)

[**3. PRIKAZ SIMULACIJE** 19](#_Toc187581169)

[**4. PRILOG** 23](#_Toc187581170)

[**5. LITERATURA** 33](#_Toc187581171)

# UVOD

Ovim projektom prikazujemo aritmetičko logičku jedinicu od 4 bita, implementiranu bez korišćenja namenskih kola.

Naša aritmetičko logička jedinica se sastoji od logičkog dela (AND, OR, XOR, NOT, NAND i NOR) aritmetičkog dela (sabiranje, oduzimanje, negacija, inkrement i dekrement) i od komparatora (> ,<, =, >= ,<= ,!=). Pošto imamo ALU od 4 bita znači da su nam ulazi i izlazi četvorobitni brojevi. Osim brojeva sa kojima baratamo, imamo ulaze za izbor operacije i za kontrolu komparatora. Na izlazu pored rezultata od 4 bita, imamo izlaz za Carry/Borrow i Overflow bitove kao i za komparator. Limitirani smo time što ne možemo da koristimo namenska kola (kao što su 74HC283 ili 74HC85), ali možemo da ih zamenimo koristeći module napravljene sa osnovnim logičkim kolima.

Ovaj dokument služi kao dodatak projektu napravljenom u Proteus, programu za digitalnu simulaciju.

# **PROJEKTNI ZADATAK**

Izabrana tema: 36-E6

Naziv teme: ALU sa 4 bita

Dodatni zahtev: Bez namenskih kola

## 1.1 Svrha projekta i ograničenja

Svrha projekta je prikaz rada aritmetičko logičke jedinice pomoću Proteus-ove simulacije bez namenskih kola. Svrha dodatnog zahteva je da pokažemo da je moguće napraviti svako namensko kolo pomoću osnovnih kola. Glavno ograničenje je 4 bita, imamo samo brojeve od 0 do 15 tj. Od -8 do 7 ako koristimo brojeve sa predznakom. Simulacija mora biti proverena za sve kombinacije od dva četvorobitna broja.

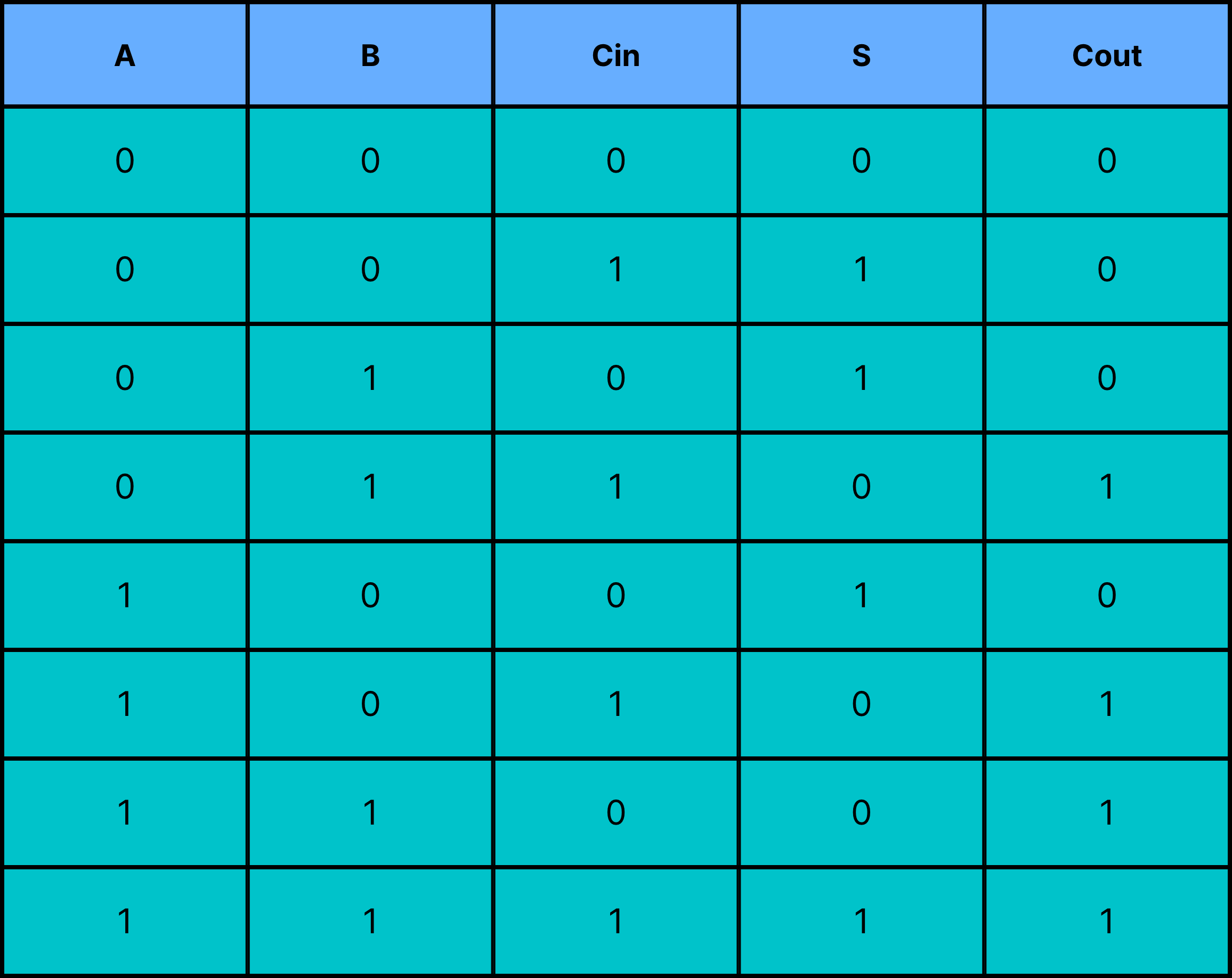
## 1.2 Način realizacije

Projekat je realizovan kroz implementaciju najosnovnijih modula (npr. pun sabirač) pomoću osnovnih logičkih kapija, pa implementacijom složenijih modula koristeći nove module(npr. četvorobitni sabirač od četiri puna sabirača). Nakon implementacije svaki modul je posebno testiran van ALU i kasnije testiran u njoj. Finalni modul će biti prikazan šemama i njegova funkcija će biti prikazana simulacijom i tablicama istinitosti. Funkcije naše ALU su: AND, OR, XOR, NOT, NAND, NOR, sabiranje, oduzimanje, inkrement, dekrement i upoređivanje dva četvorobitna broja.

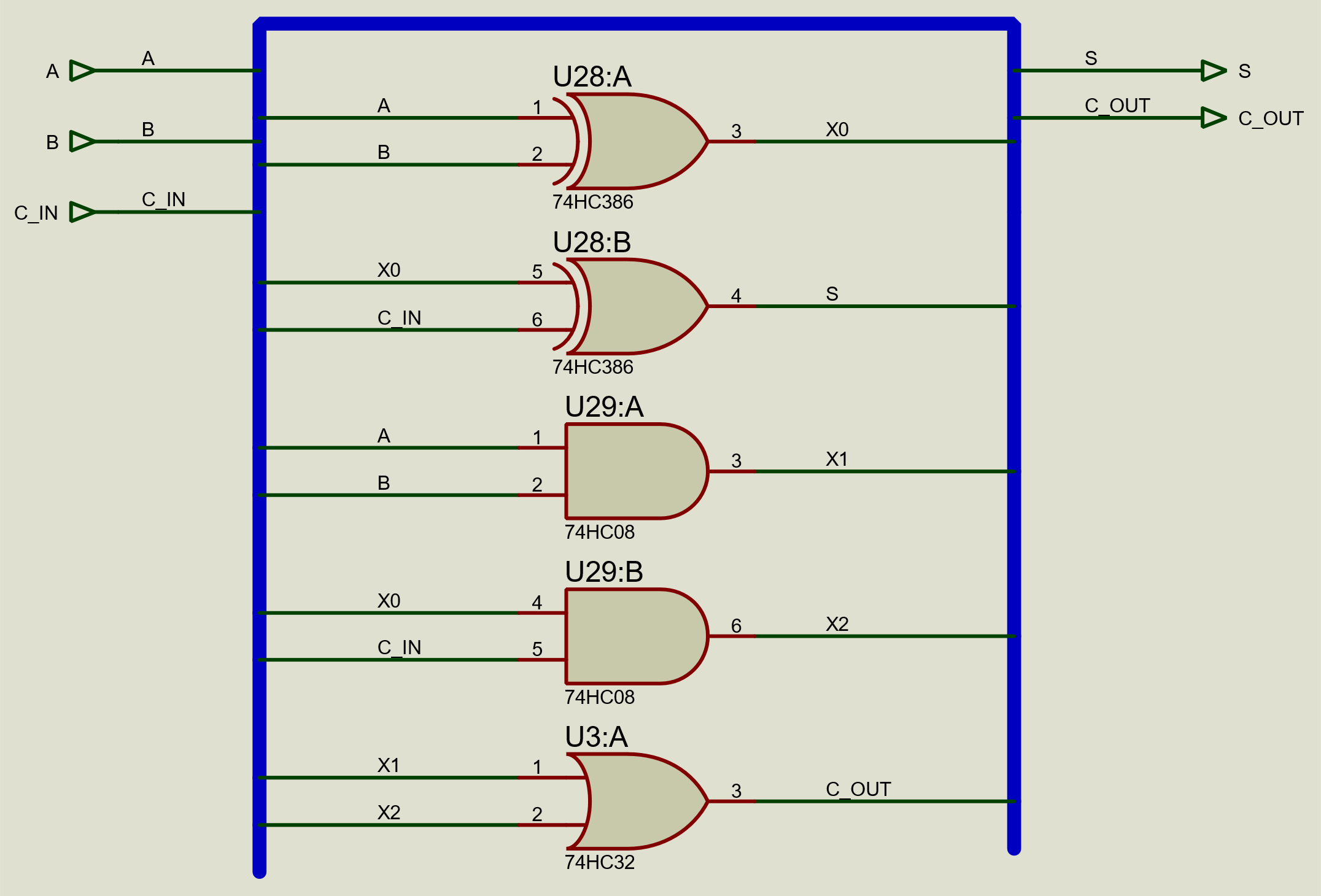
# **TEORIJSKE OSNOVE**

## 2.1 Aritmetički deo

### *2.1.1 Modul za sabiranje*

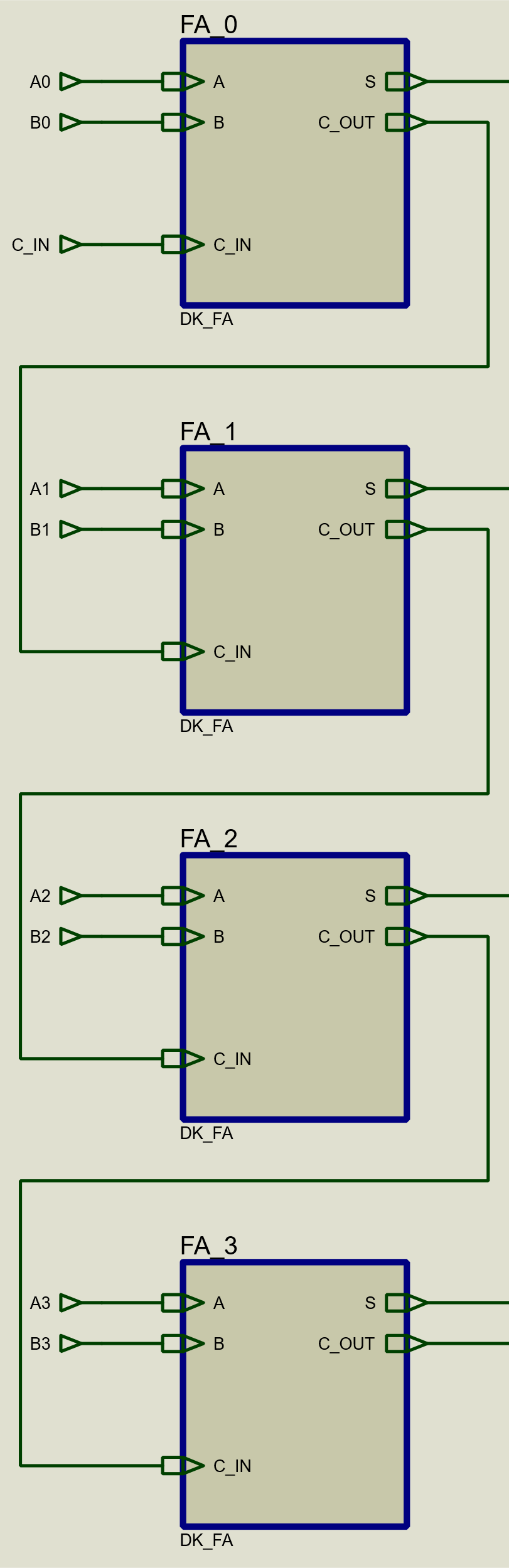


*Tabela 1. – tablica istinitosti za 1-bitni pun sabirač*

**

*Slika 1.- šema 1-bitnog punog sabirača*

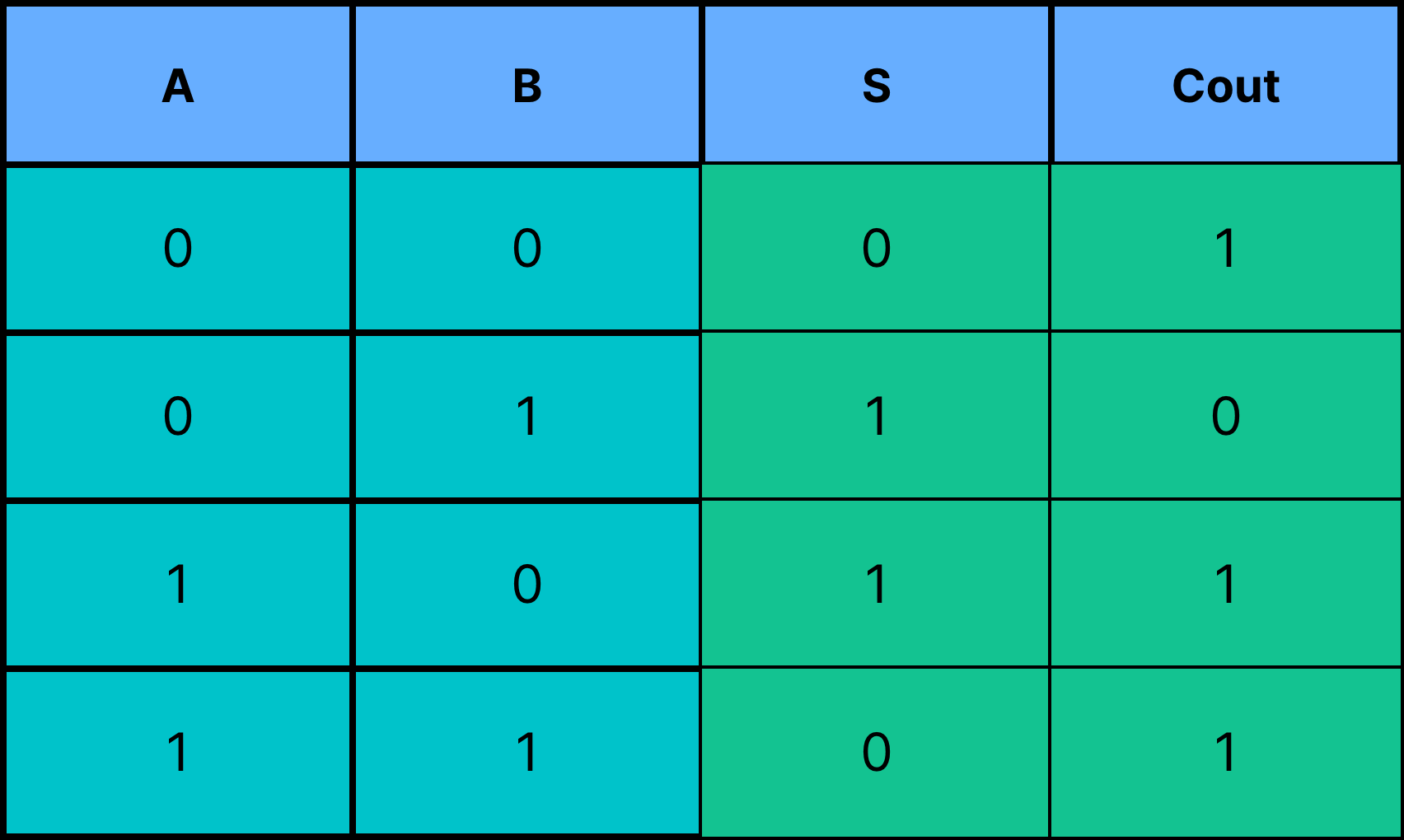
Uz pomoć dva XOR, dva AND i jednog OR kola sastavljamo pun sabirač sa Carry in i Carry out, koji sabira dva broja od jednog bita.



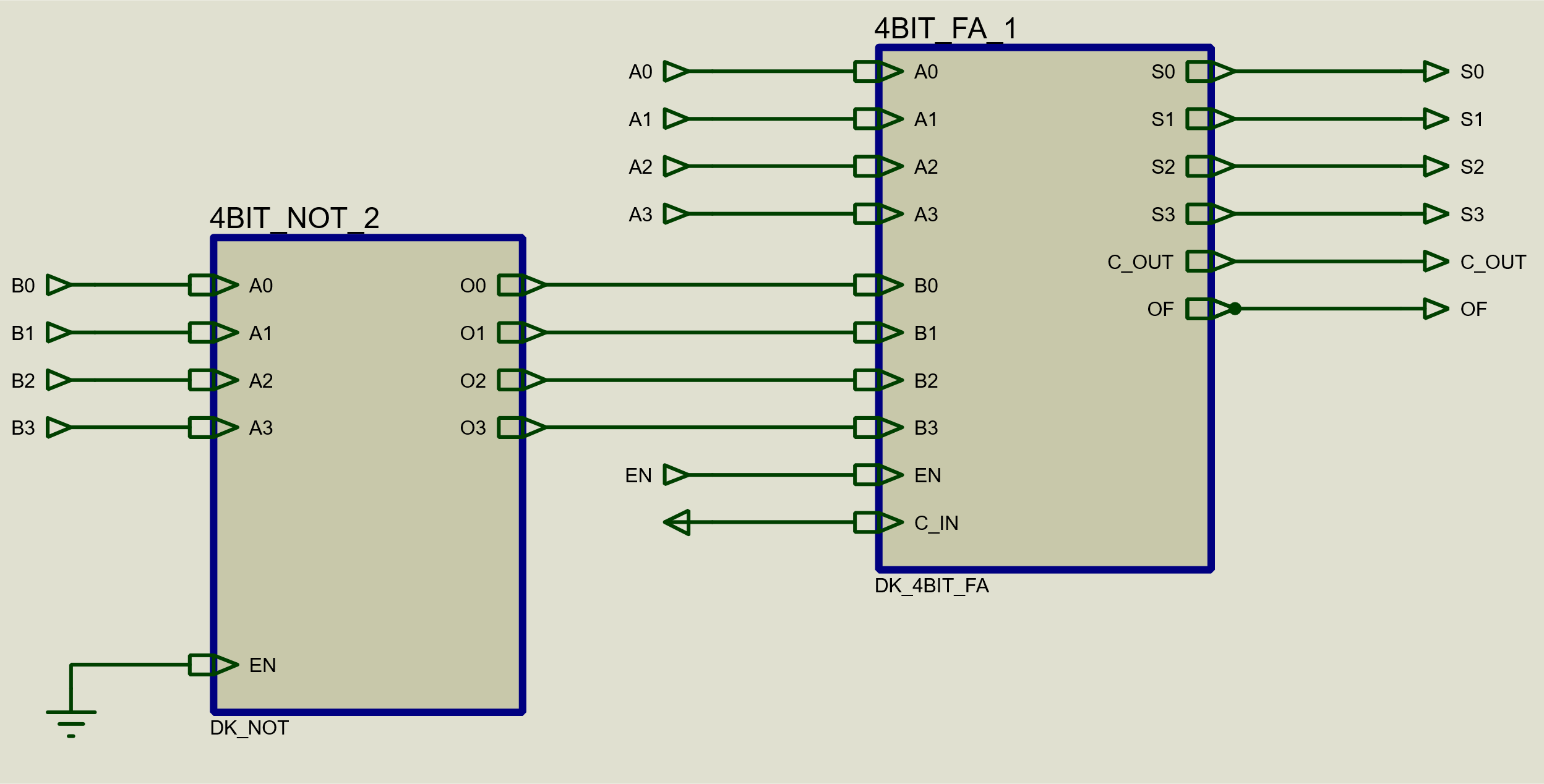
*Slika 2.- šema 4-bitnog sabirača*

Lančanjem 1-bitnih sabirača (iz slike 1.) tako što Carry out svakog prethodnog stavljamo u Carry in svakog sledećeg. Prvi Carry in je 0, poslednji Carry out je Carry out celog sabirača od 4 bita.

### *2.1.2 Modul za oduzimanje*



*Tabela 2. – tablica istinitosti za 1-bitni oduzimač*

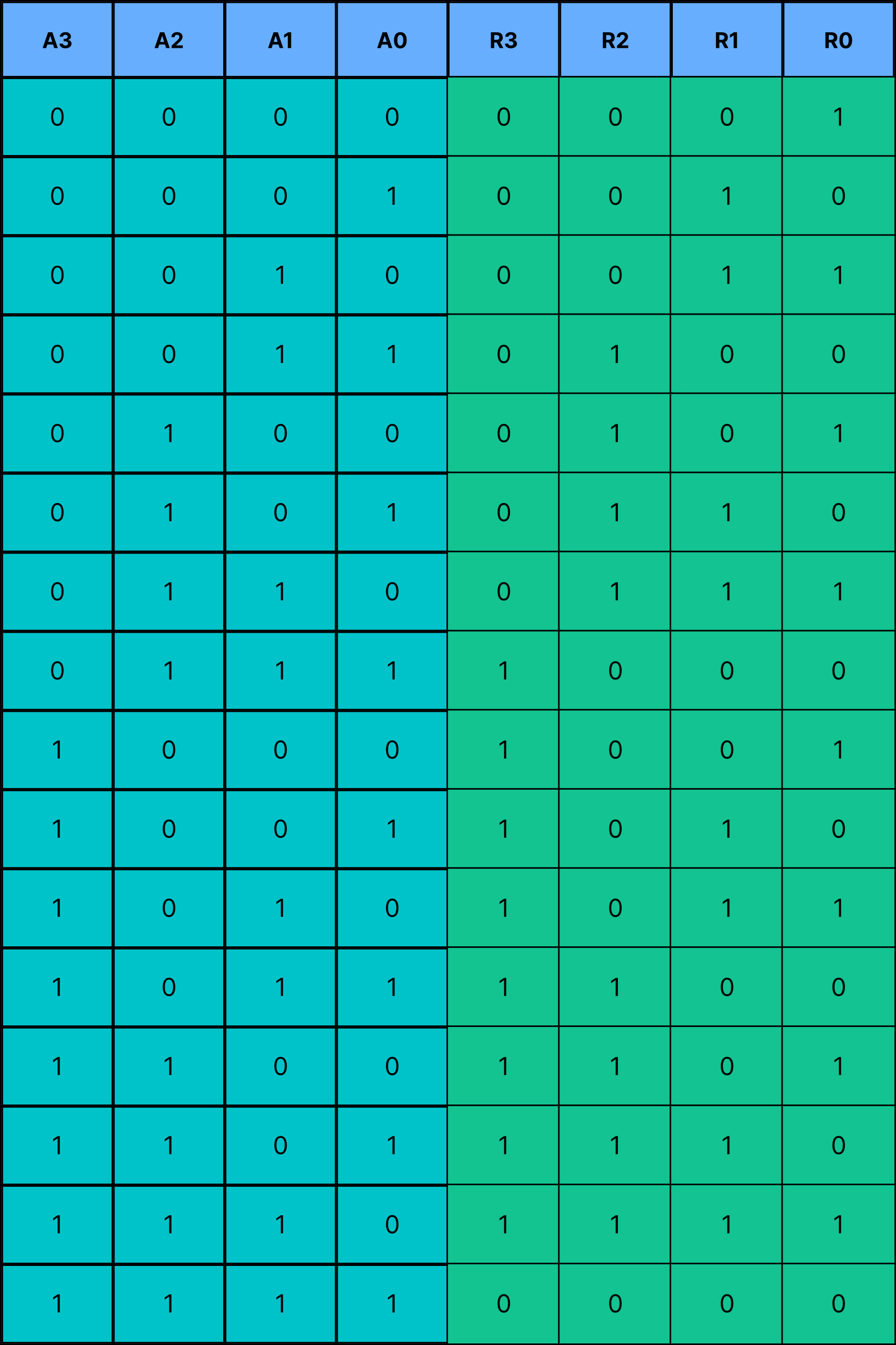


*Slika 3. – Šema 4-bitnog oduzimača*

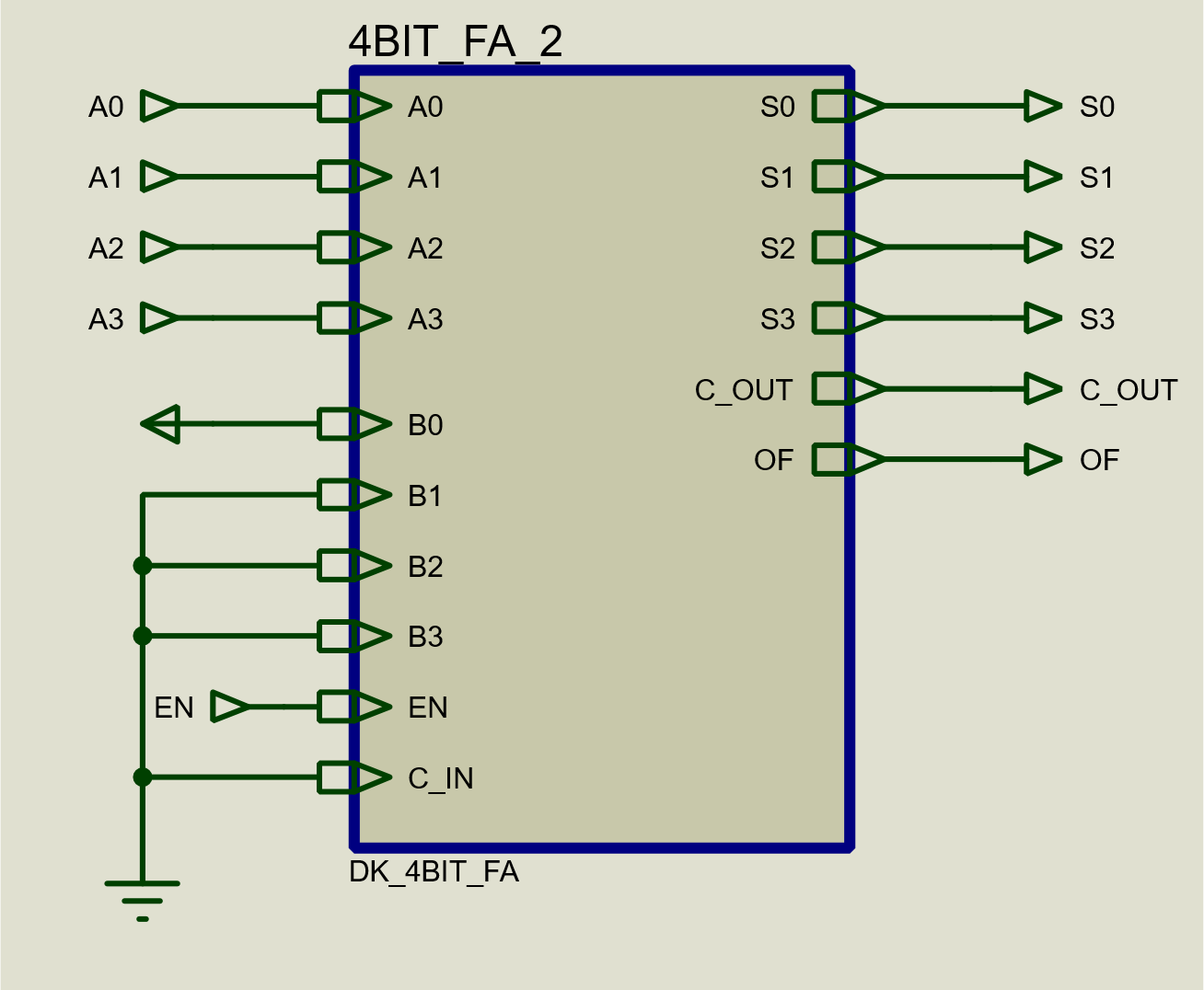
Oduzimač smo napravili koristeći 4-bitni sabirač (2.1.1) i 4-bitni NOT (2.2.1), tako da umesto A – B imamo A + (-B). Broj A ubacujemo u sabirač nepromenjenog, a broj B ubacujemo u NOT, pa dobijemo B u prvom komplementu. Pošto nam treba drugi komplement, umesto da inkrementiramo B, postavljamo Cin na Power tj. na 1 i sa tim dobijamo -B (broj B u drugom komplementu).

U tabeli 2 imamo prikaz 1 bitnog modula za oduzimanje jer bi tablica istinitosti za 4 bitni oduzimač bila prevelika za ovaj rad.

### *2.1.3 Modul za inkrement*



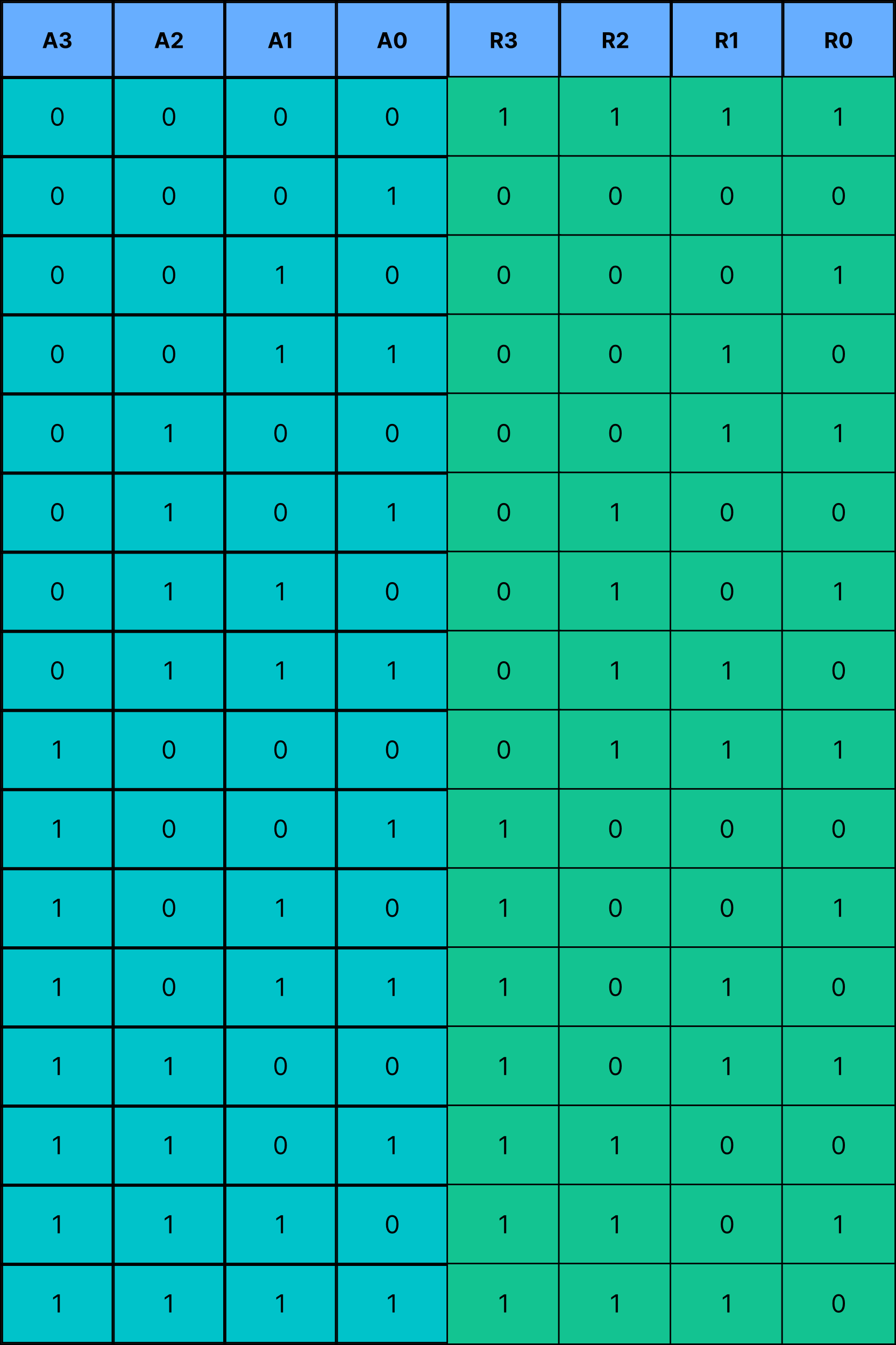
*Tabela 3. – tablica istinitosti za modul za inkrement*



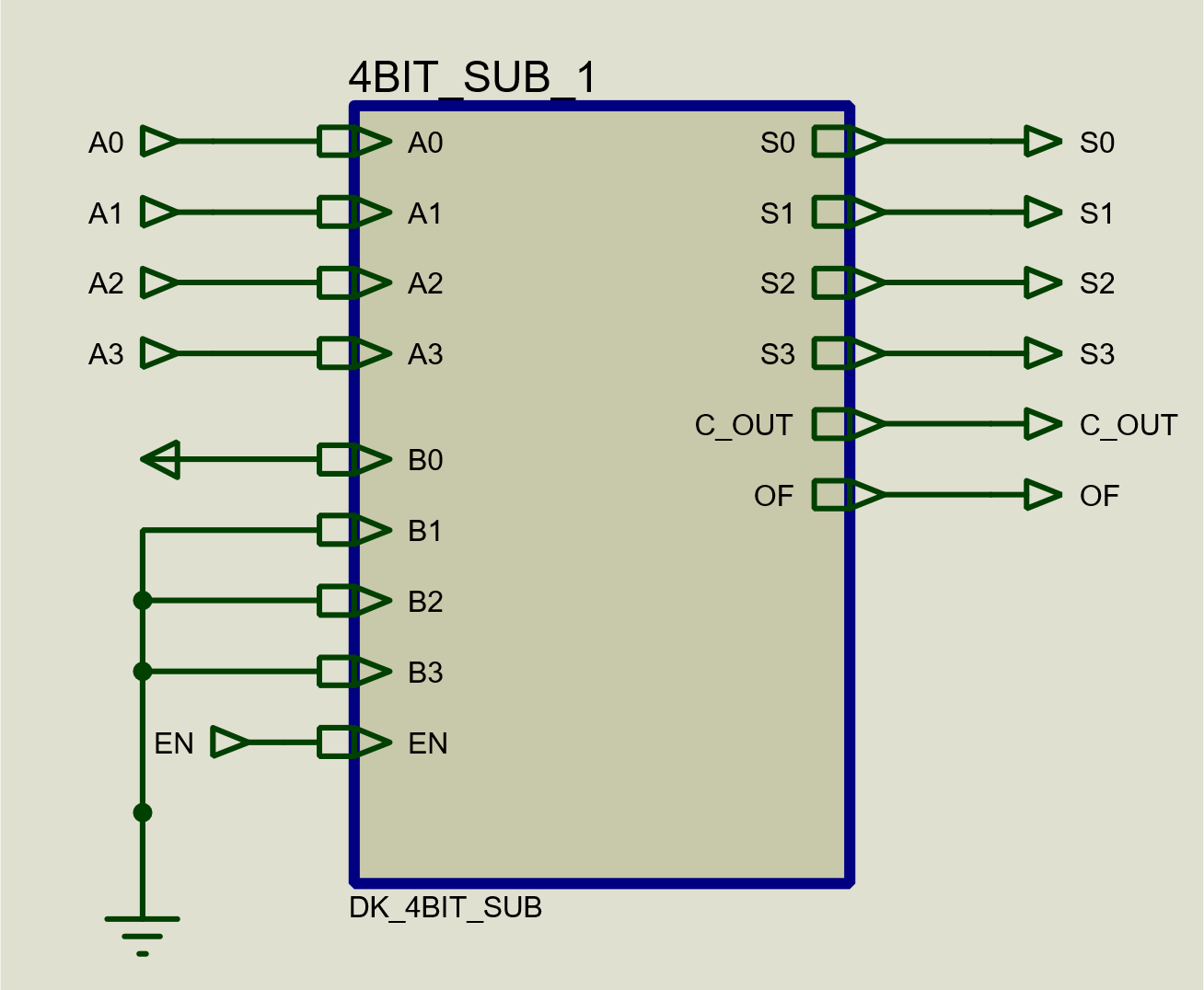
*Slika 4. – šema 4.bitnog modula za inkrement*

Modul za inkrement smo dobili koristeći jedan 4-bitni sabirač (2.1.1). Broj koji želimo da inkrementiramo smo stavili na ulaz A. Na ulazu B stavili smo LSB na Power (1), a ostale bitove na Ground (0). Tako da u sabiraču imamo A + 0001.

### *2.1.4 Modul za dekrement*



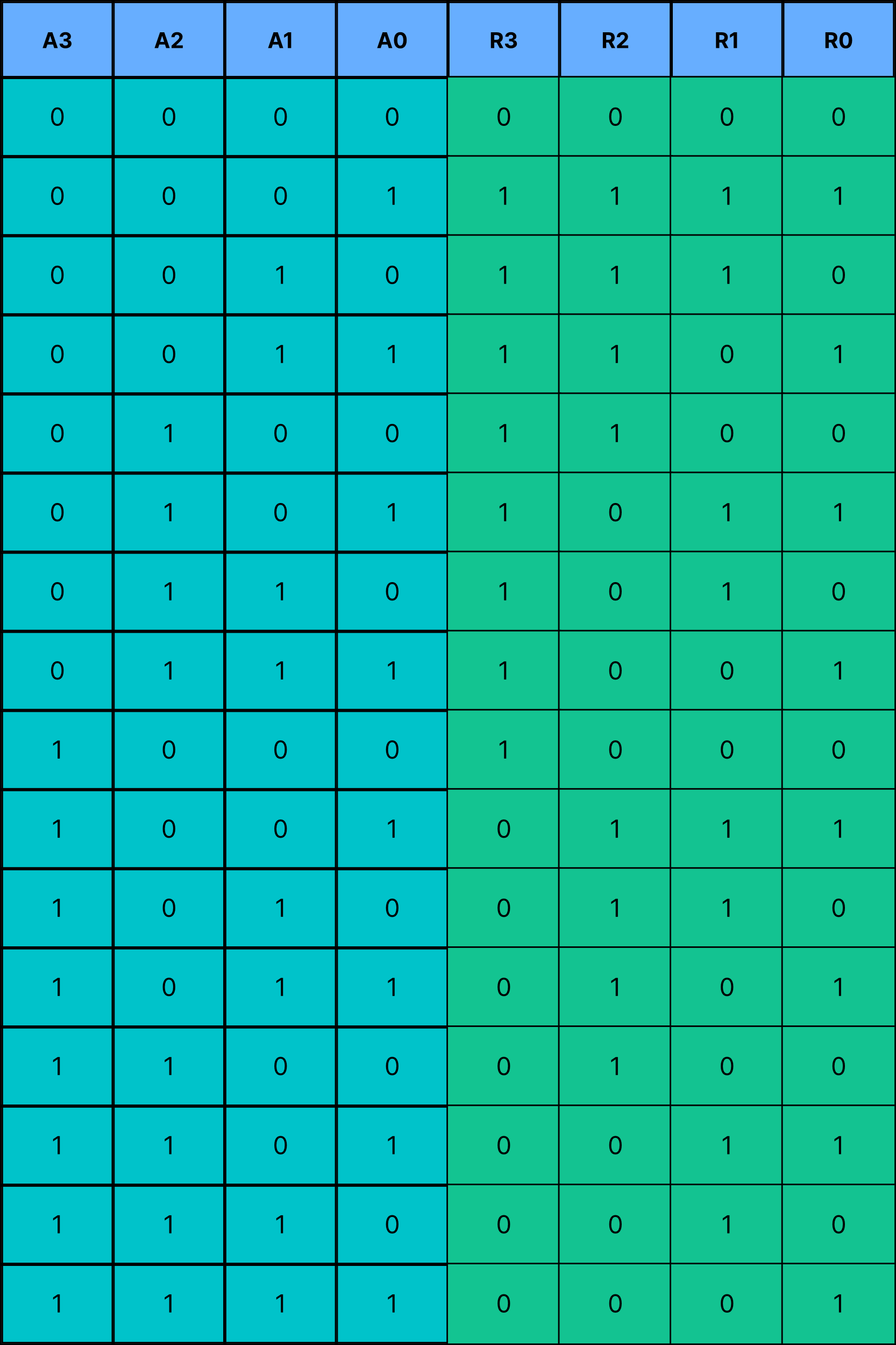
*Tabela 4. – tablica istinitosti za modul za dekrement*



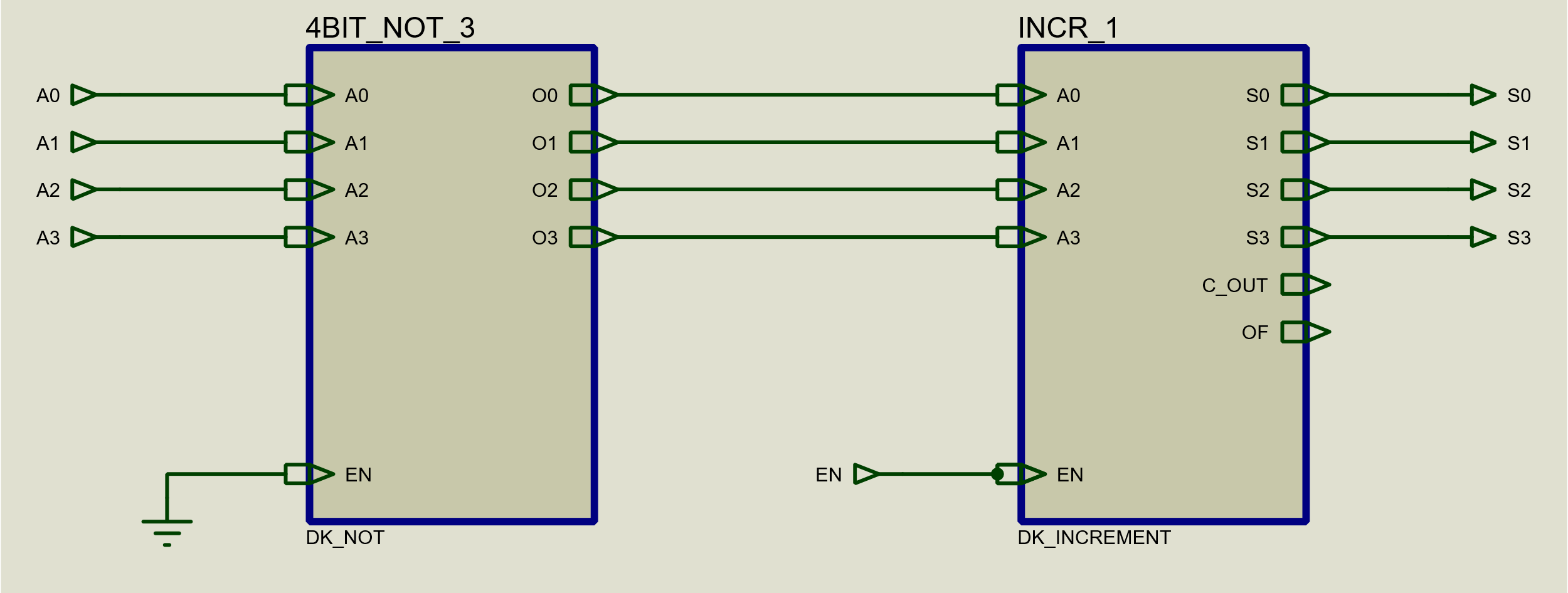
*Slika 5. – šema 4.bitnog modula za dekrement*

Modul za dekrement smo dobili koristeći jedan 4-bitni oduzimač (2.1.2). Broj koji želimo da dekrementiramo smo stavili na ulaz A. Na ulazu B stavili smo LSB na Power (1), a ostale bitove na Ground (0). Tako da u oduzimaču imamo A – 0001.

### *2.1.5 Modul za drugi komplement*



*Tabela 5. – tablica istinitosti za modul za drugi komplement*

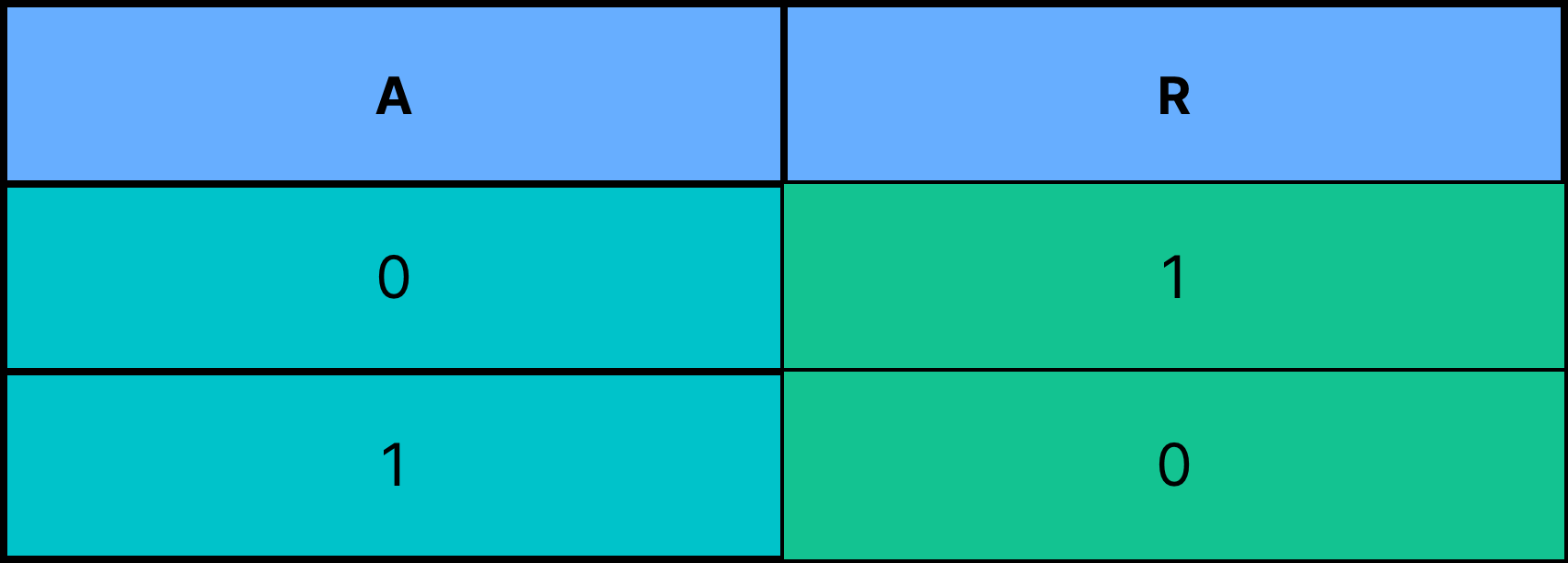


*Slika 6. – šema 4.bitnog modula za drugi komplement*

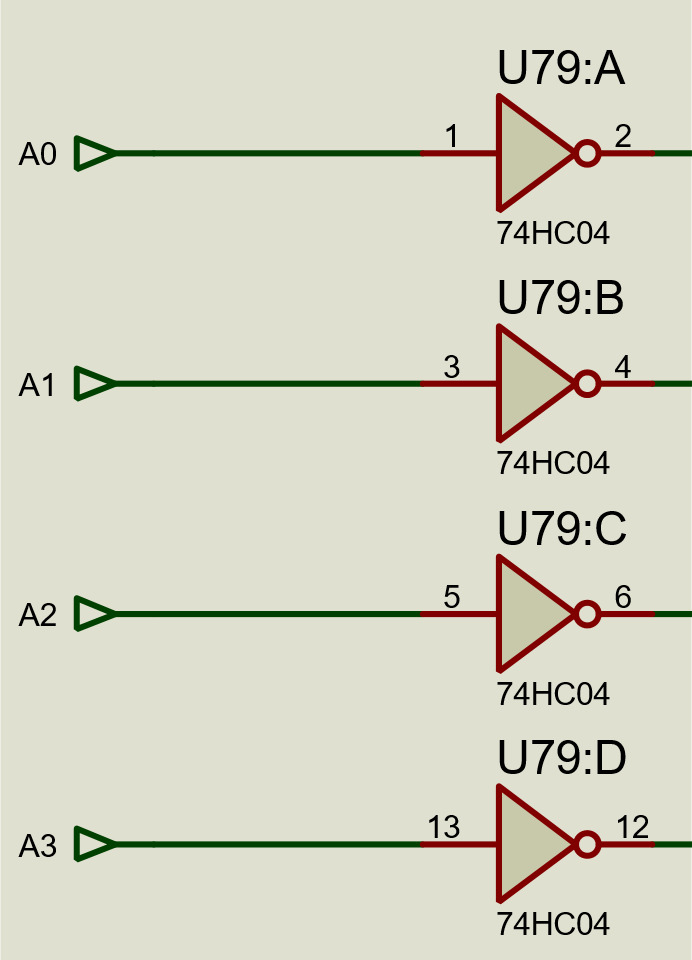
Modul za drugi komplement smo dobili, prvo ubacivanjem našeg ulaza (A) u NOT modul (2.2.1) posle čega izlaz iz NOT modula (prvi komplement) ulazi u modul za inkrement (2.1.3) i postaje drugi komplement tj. na izlazu je negiran ulazni broj (-A).

## 2.2 Logički deo

### *2.2.1 NOT modul*



*Tabela 6. – tablica istinitosti za NOT logičku funkciju*



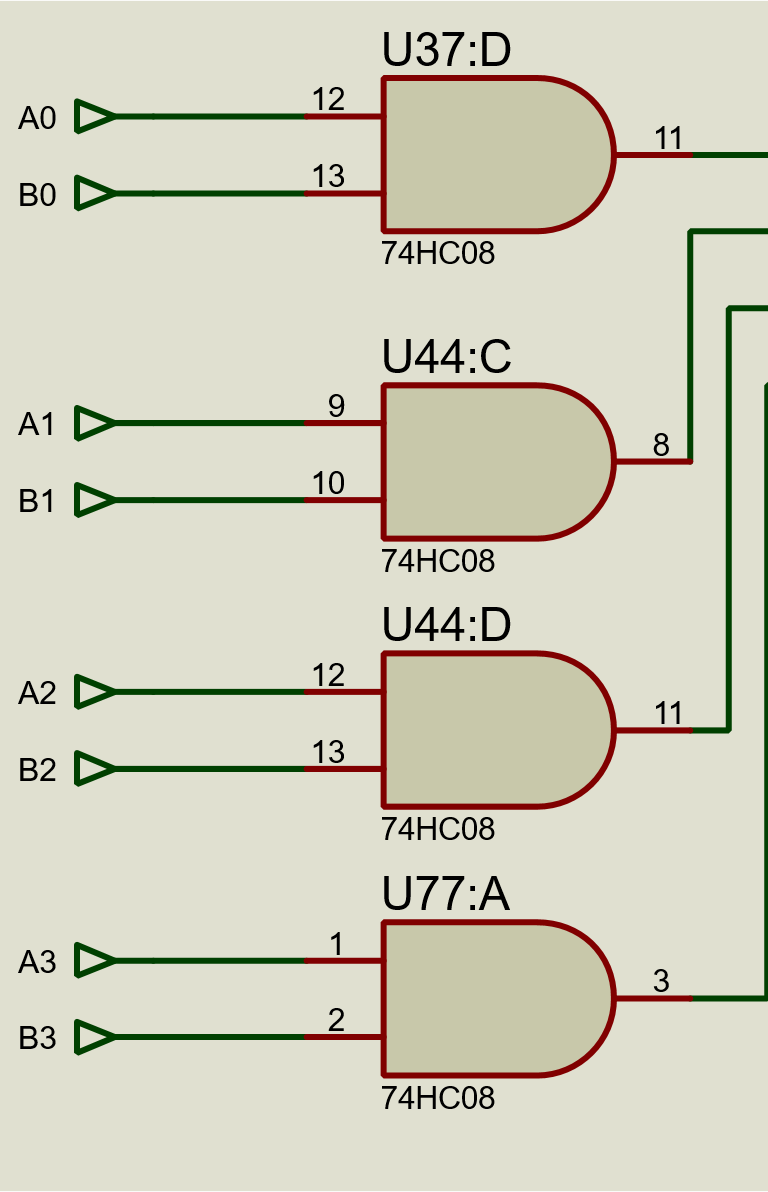
*Slika 7. – šema 4-bitnog NOT modula*

Sa 4 NOT logičke kapije svaki bit ulaza negiramo posebno.

### *2.2.2 AND modul*



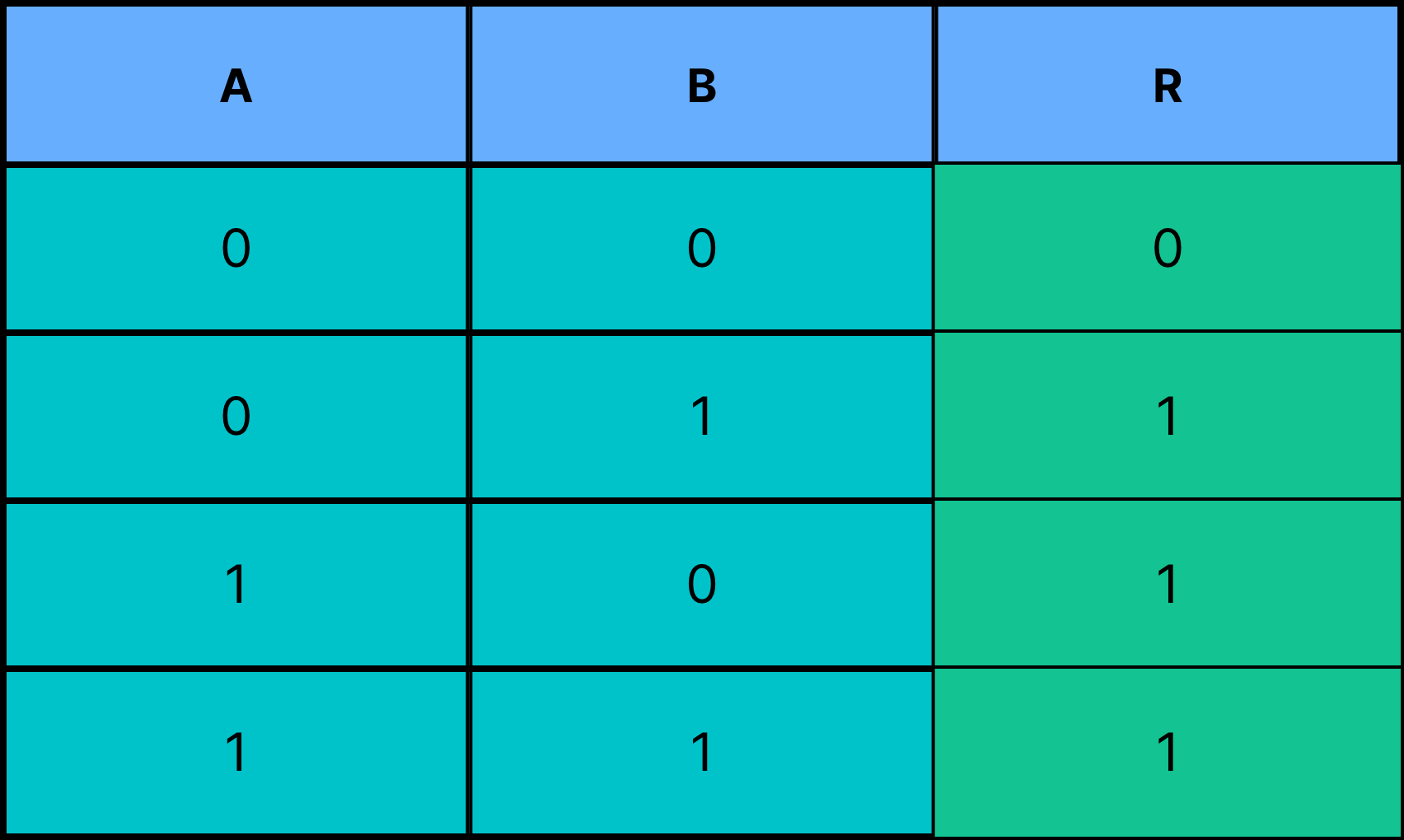
*Tabela 6. – tablica istinitosti za AND logičku funkciju*



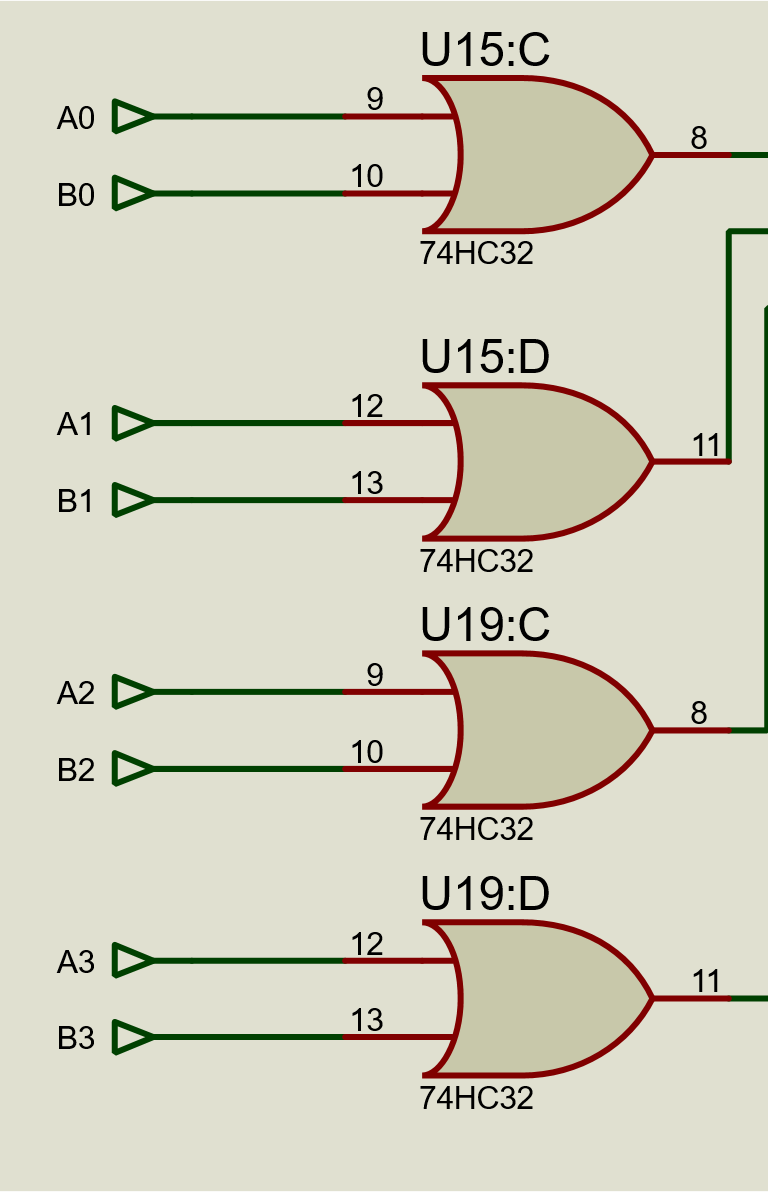
*Slika 7. – šema 4-bitnog AND modula*

Sa 4 AND logičke kapije, radimo AND svakog bita broja A sa svakim bitom broja B (A0 AND B0, A1 AND B1…).

### *2.2.3 OR modul*



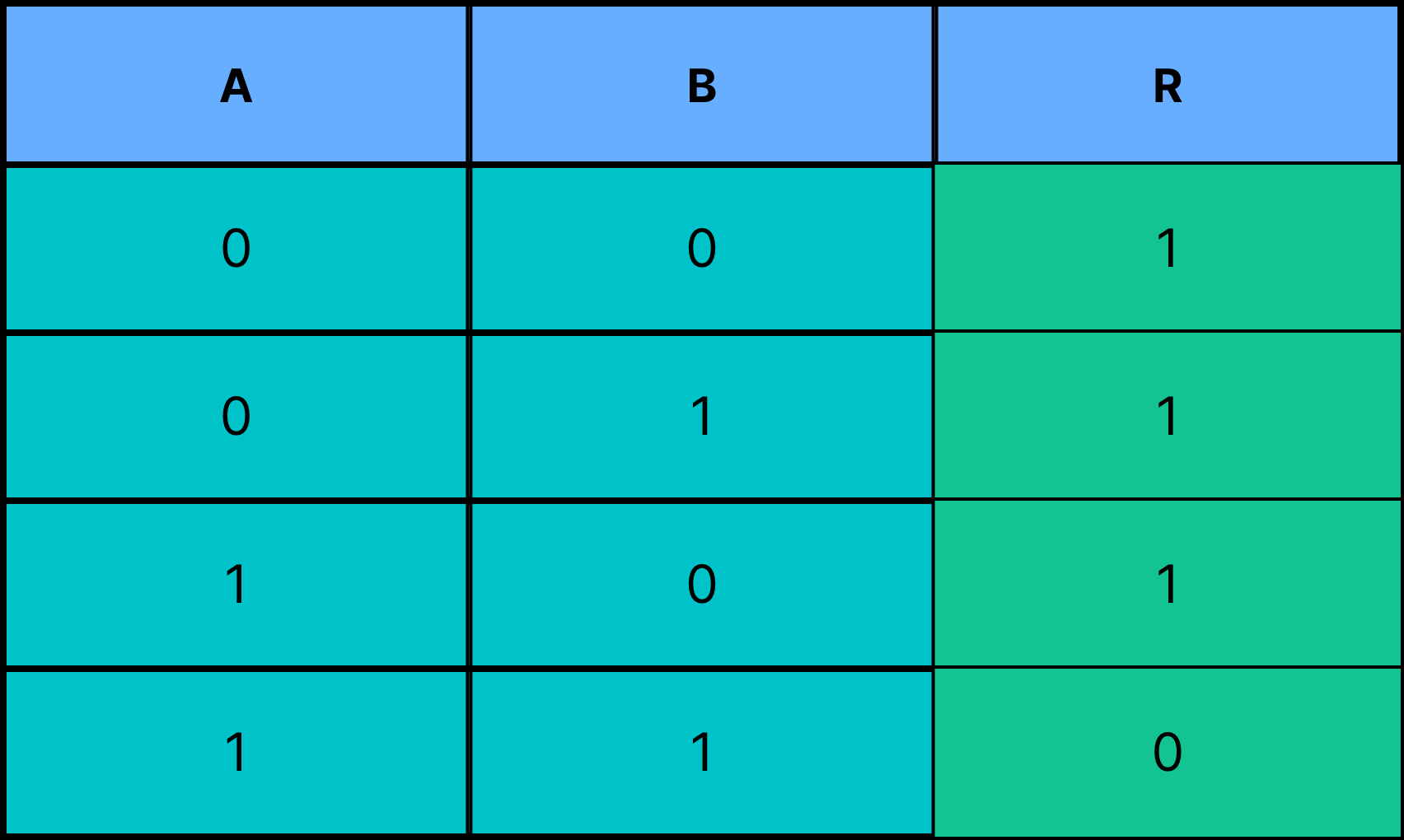
*Tabela 7. – tablica istinitosti za OR logičku funkciju*



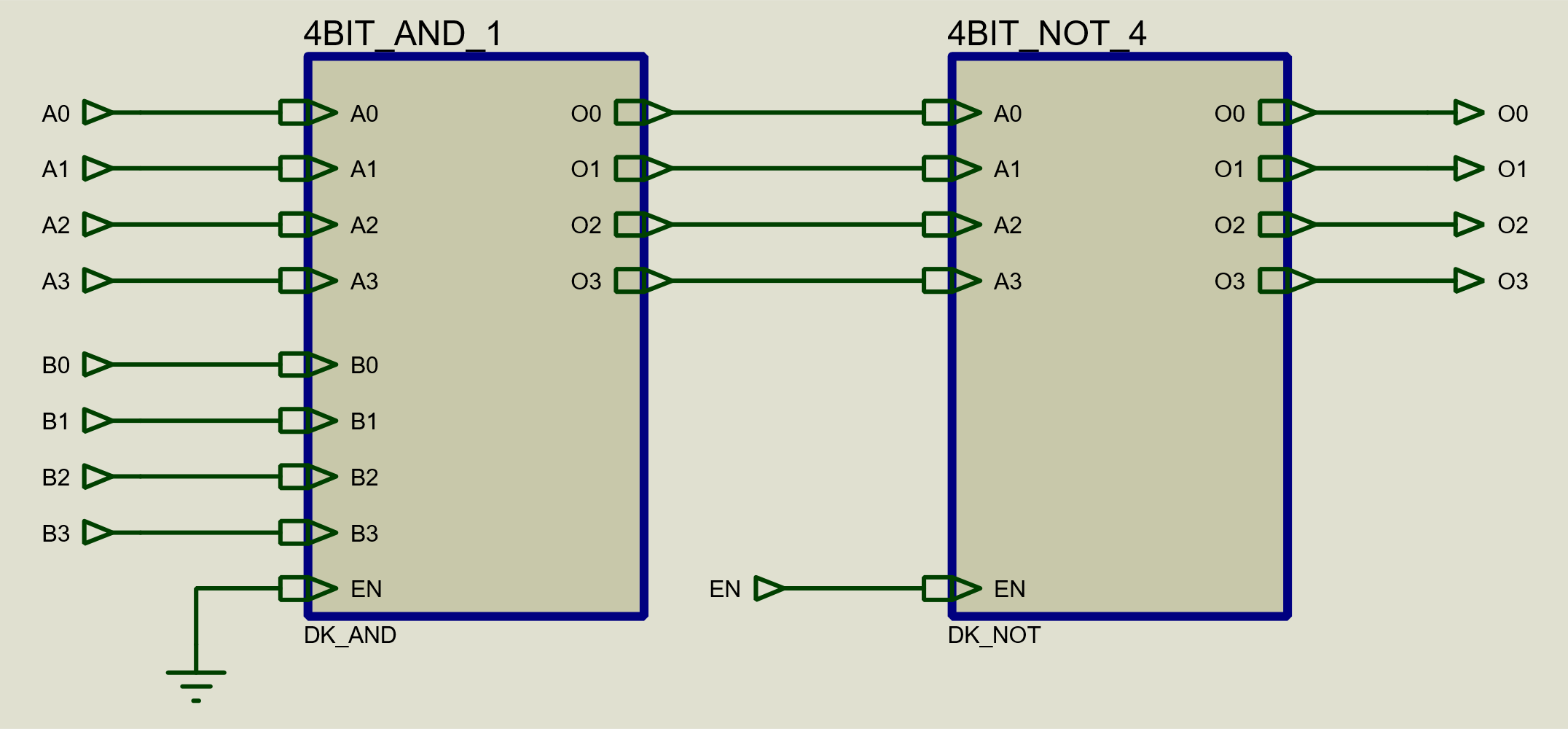
*Slika 8. – šema 4-bitnog OR modula*

Sa 4 OR logičke kapije, radimo OR svakog bita broja A sa svakim bitom broja B (A0 OR B0, A1 OR B1…).

### *2.2.4 NAND modul*



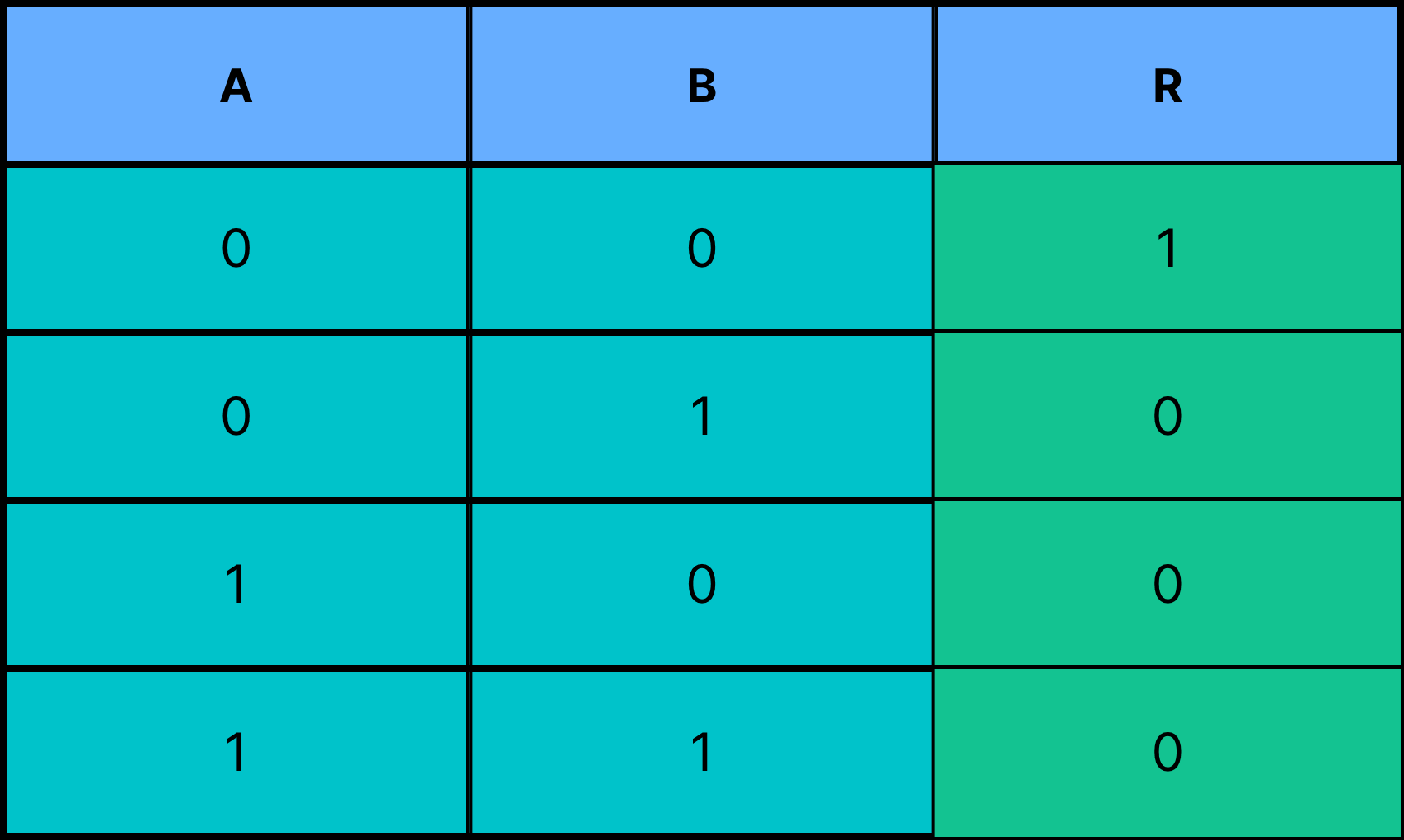
*Tabela 8. – tablica istinitosti za NAND logičku funkciju*



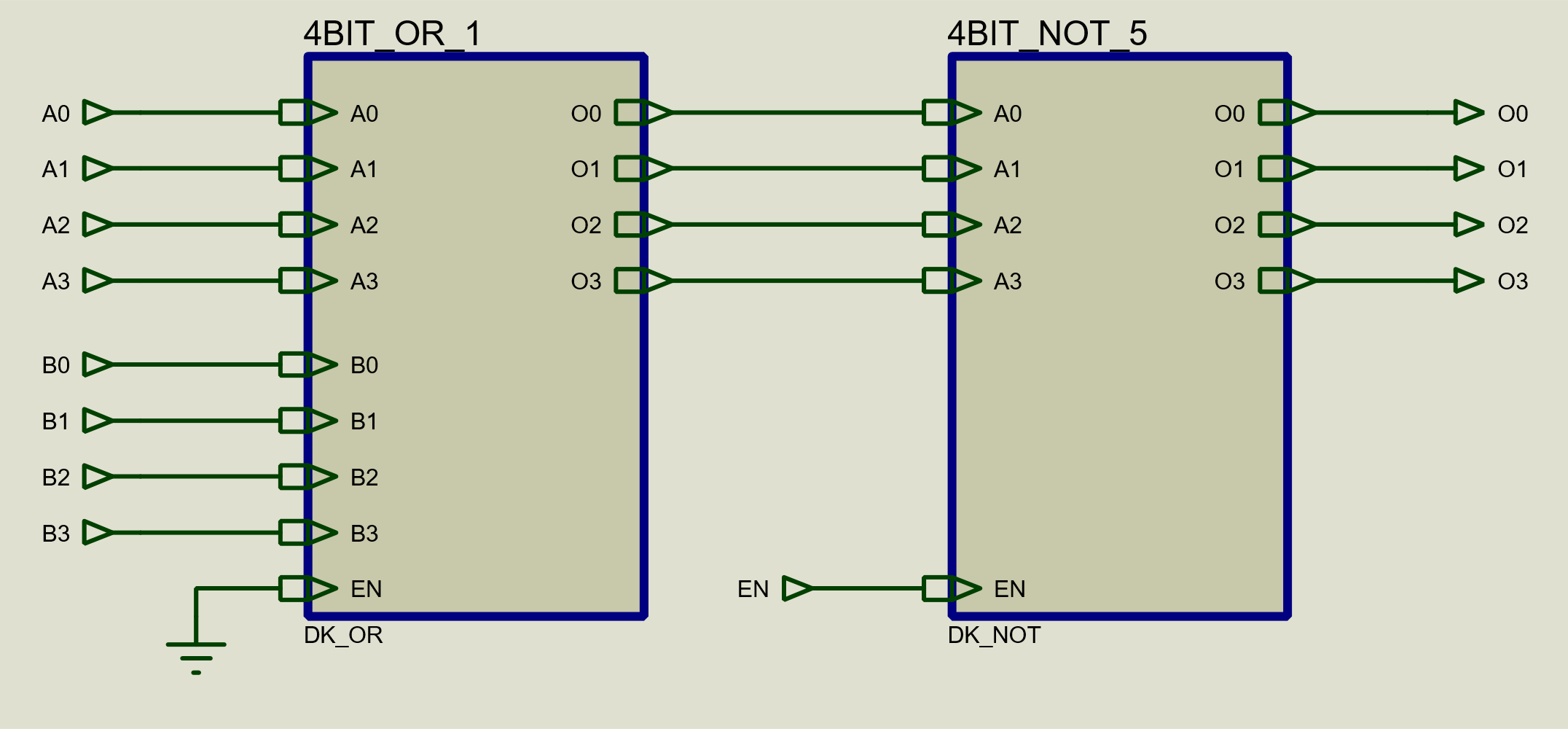
*Slika 9. – šema 4-bitnog NAND modula*

Koristili smo AND (2.2.2) i NOT (2.2.1) module kako bismo dobili 4-bitni NAND modul (A0 NAND B0, A1 NAND B1…).

### *2.2.5 NOR modul*



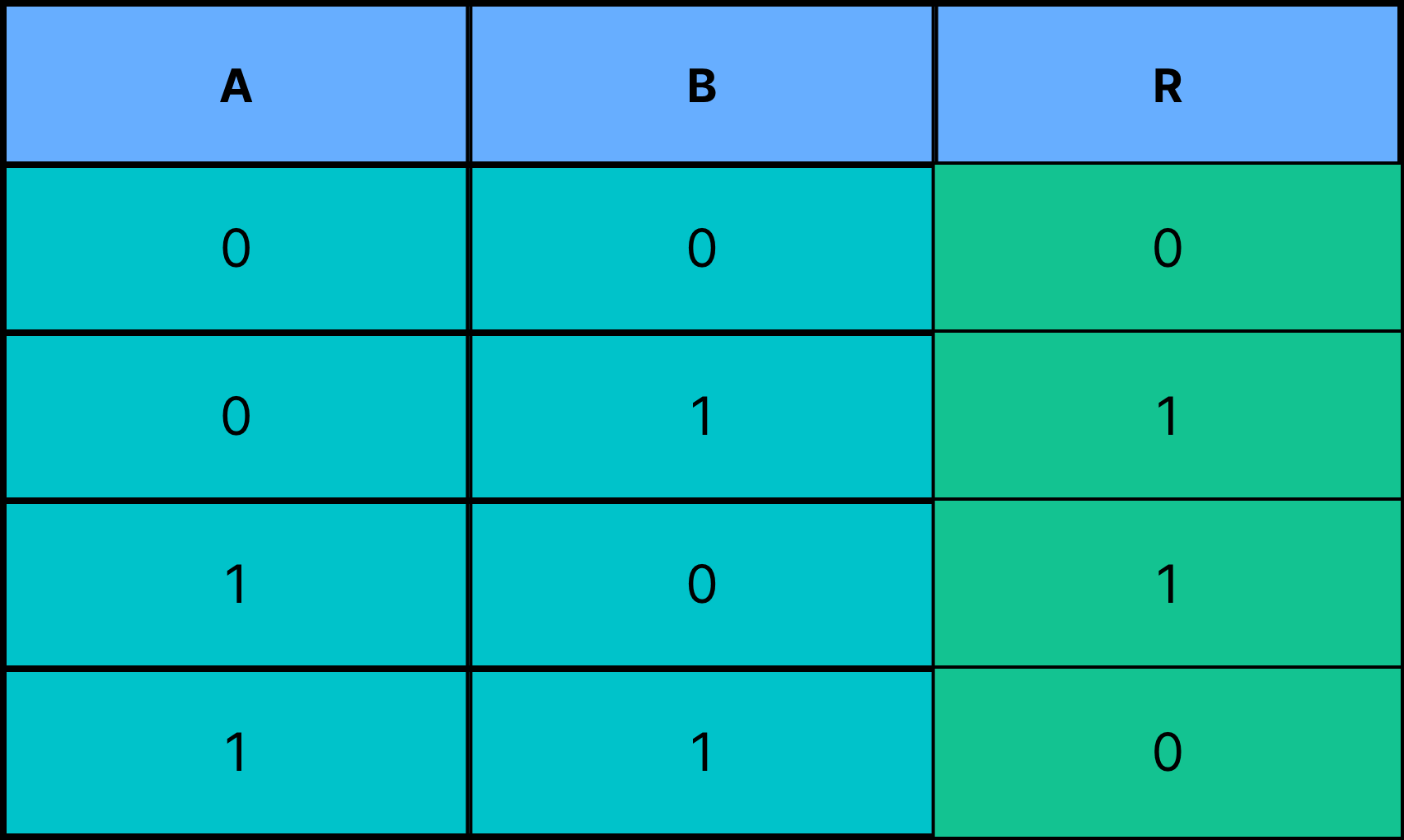
*Tabela 9. – tablica istinitosti za NOR logičku funkciju*



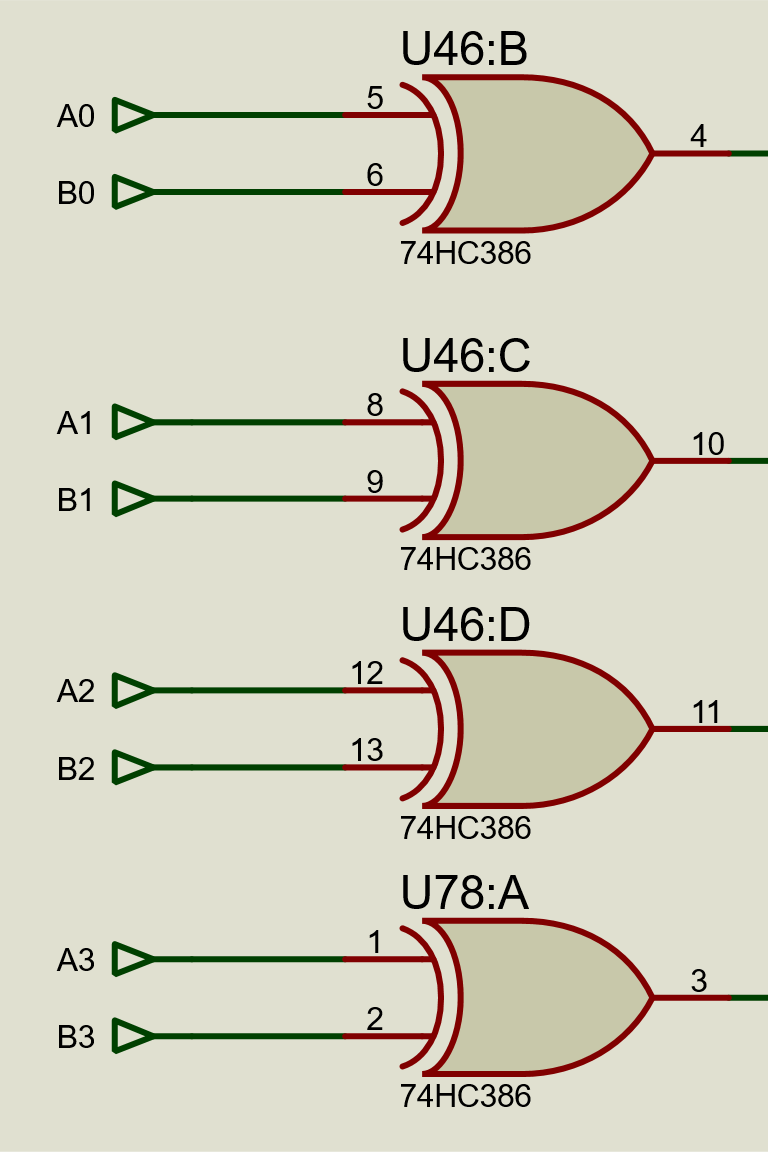
*Slika 10. – šema 4-bitnog NOR modula*

Koristili smo OR (2.2.3) i NOT (2.2.1) module kako bismo dobili 4-bitni NOR modul (A0 NOR B0, A1 NOR B1…).

### *2.2.6 XOR modul*



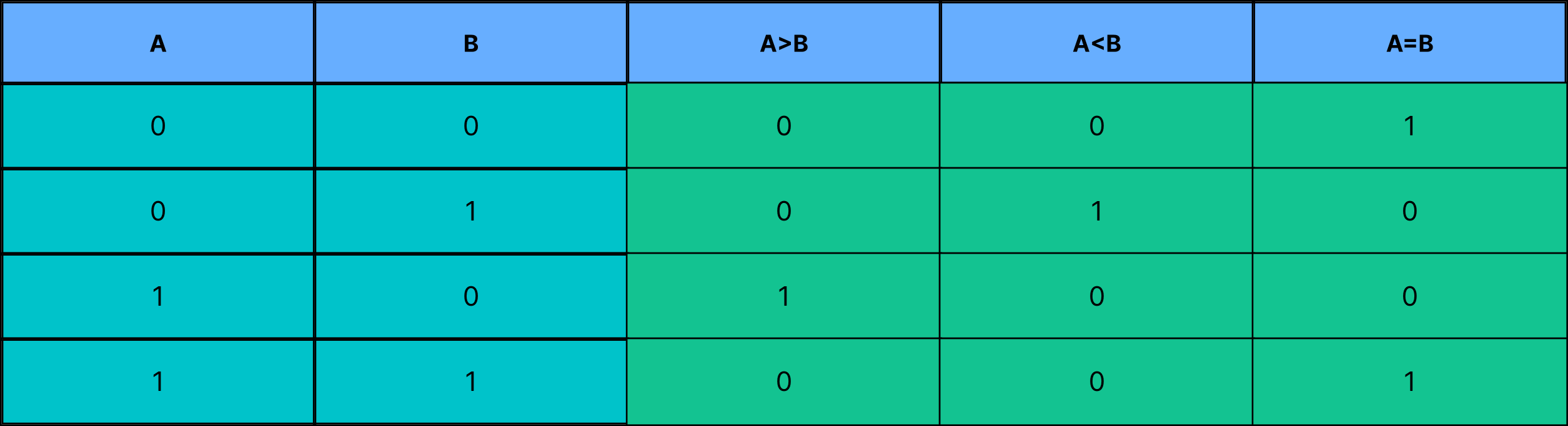
*Tabela 10. – tablica istinitosti za XOR logičku funkciju*



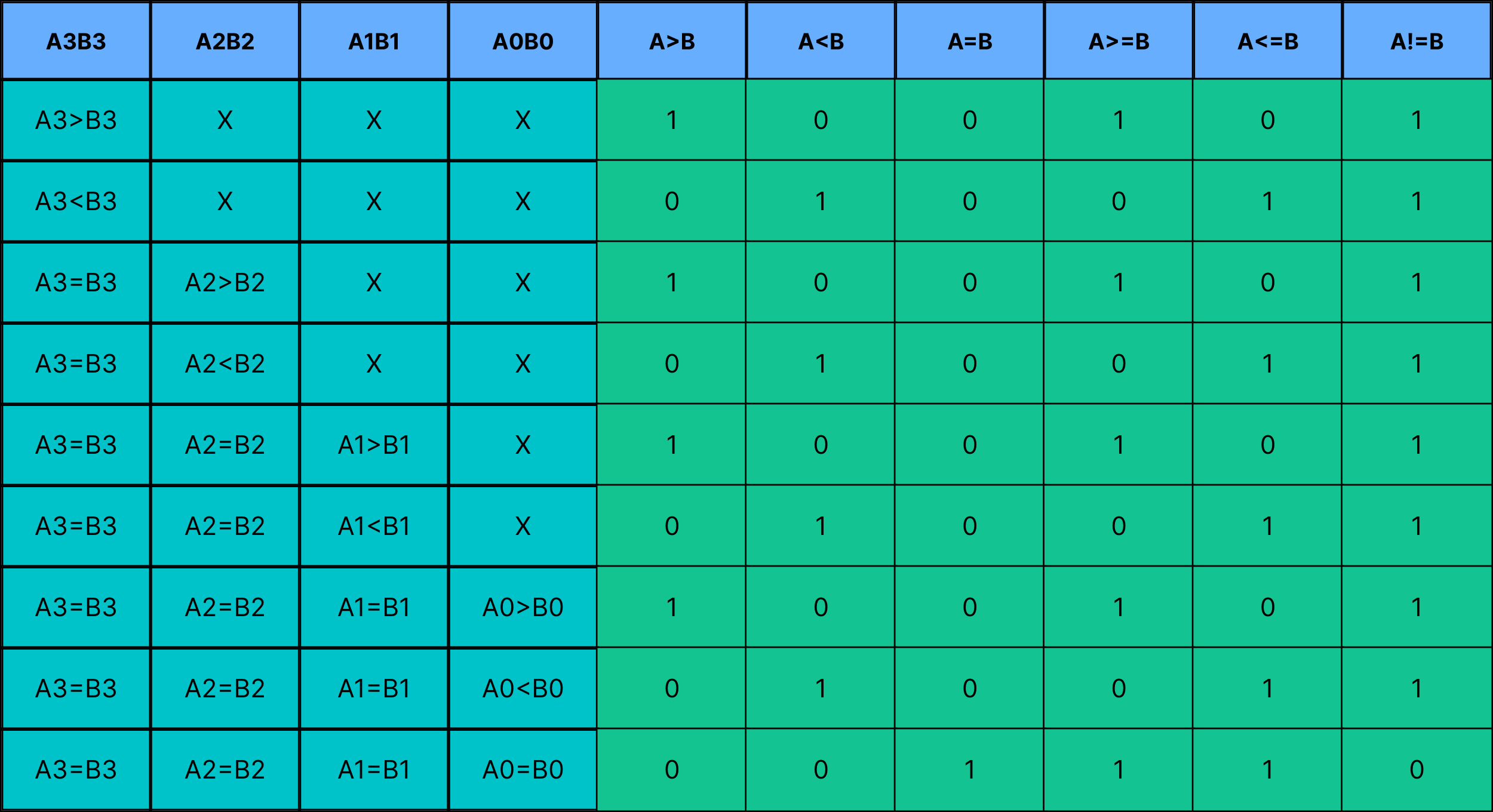
*Slika 11. – šema 4-bitnog XOR modula*

Sa 4 XOR logičke kapije, radimo XOR svakog bita broja A sa svakim bitom broja B (A0 XOR B0, A1 XOR B1…).

## 2.3 Komparator

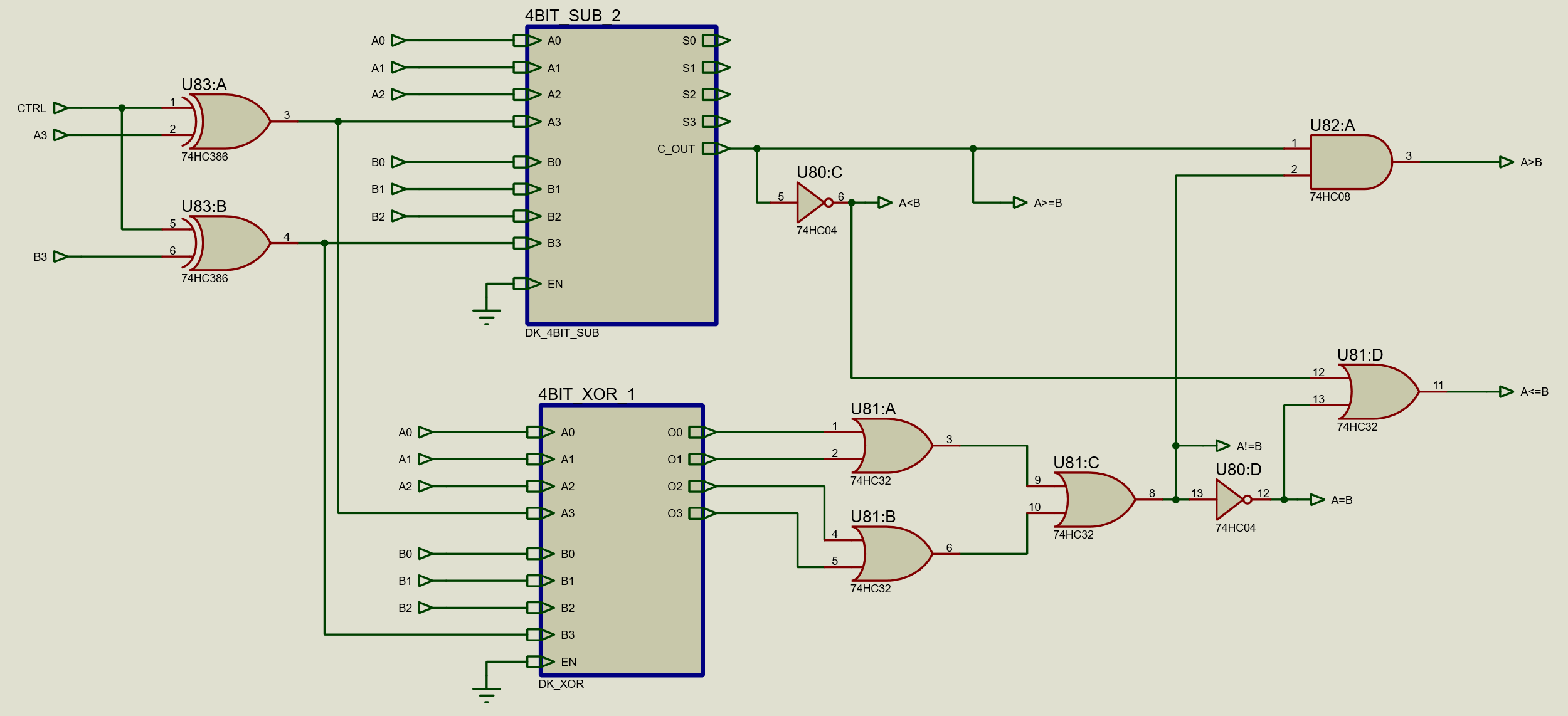


*Tabela 11. – tablica istinitosti za 1-bitni komparator*



*Tabela 12. – tablica istinitosti za 4-bitni komparator*

Tabela 11. je za 1-bitni komparator i koristimo je da pojednostavimo tabelu za 4-bitni komparator (tabela 12.). Tabela za 4-bitni koristi rezultate komparacije svakog bita posebno.



*Slika 12. – šema 4-bitnog komparatora*

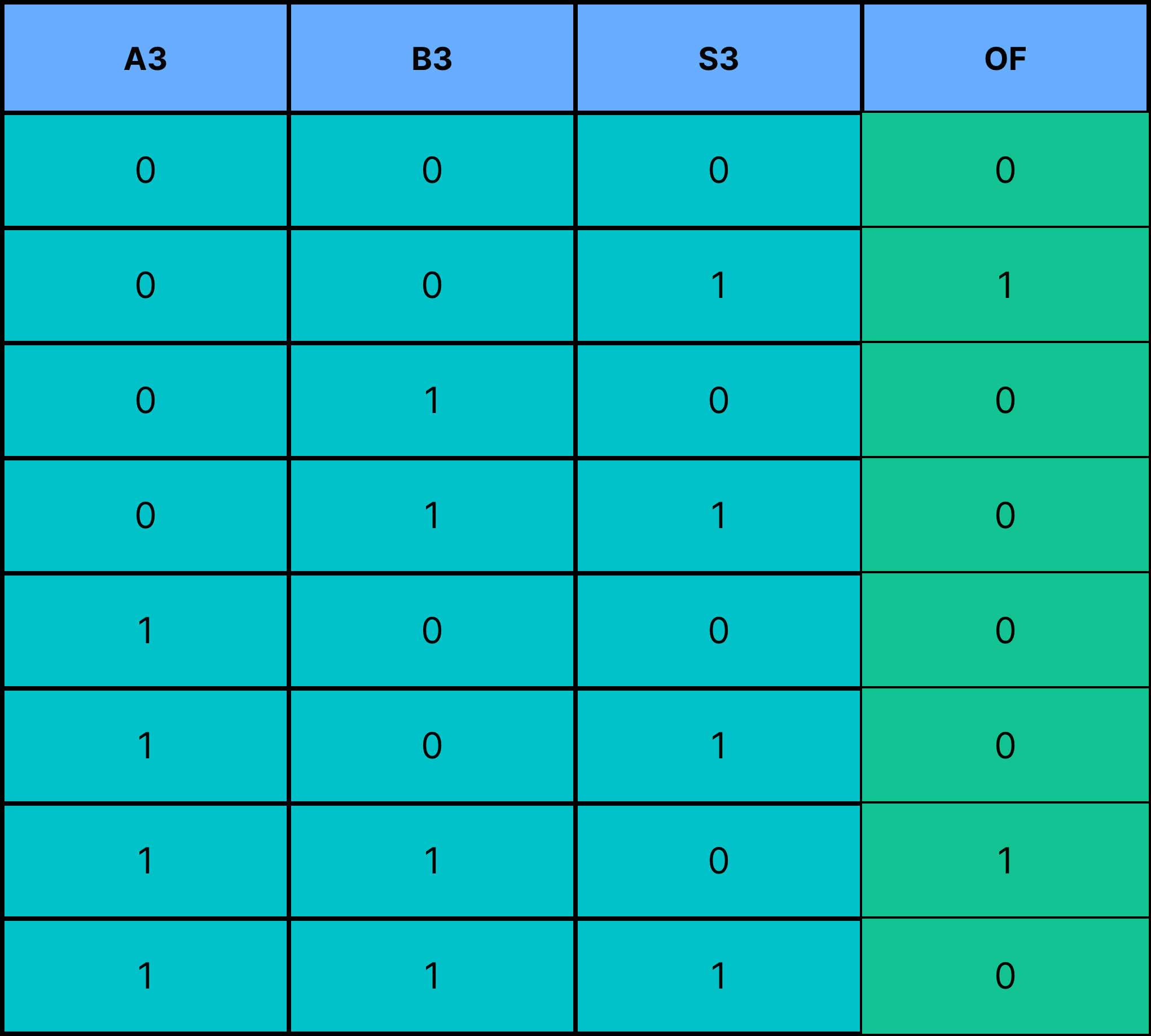
Komparator smo napravili koristeći modul za oduzimanje (2.1.2), XOR modul (2.2.6), XOR, OR, NOT i AND logička kola.

Koristili smo oduzimač jer znamo da, kada je Borrow out (c\_out na slici 12.) 1, prvi broj mora biti veći ili jednak sa drugim. Kada je Borrow out 0 prvi broj mora biti manji od drugog. Znači Borrow out vežemo direktno sa izlazom A >= B i negaciju (NOT) Borrow out izlaza sa A < B. Sa XOR modulom i OR kolima proveravamo da li je A = B. Ako je bar jedan izlaz XOR modula 1 znamo da A != B i negacijom toga (NOT) dobijamo izlaz A = B. Za A > B ubacujemo A >= B i A != B u AND kolo. Za A <= B ubacujemo A < B i A = B u OR kolo.

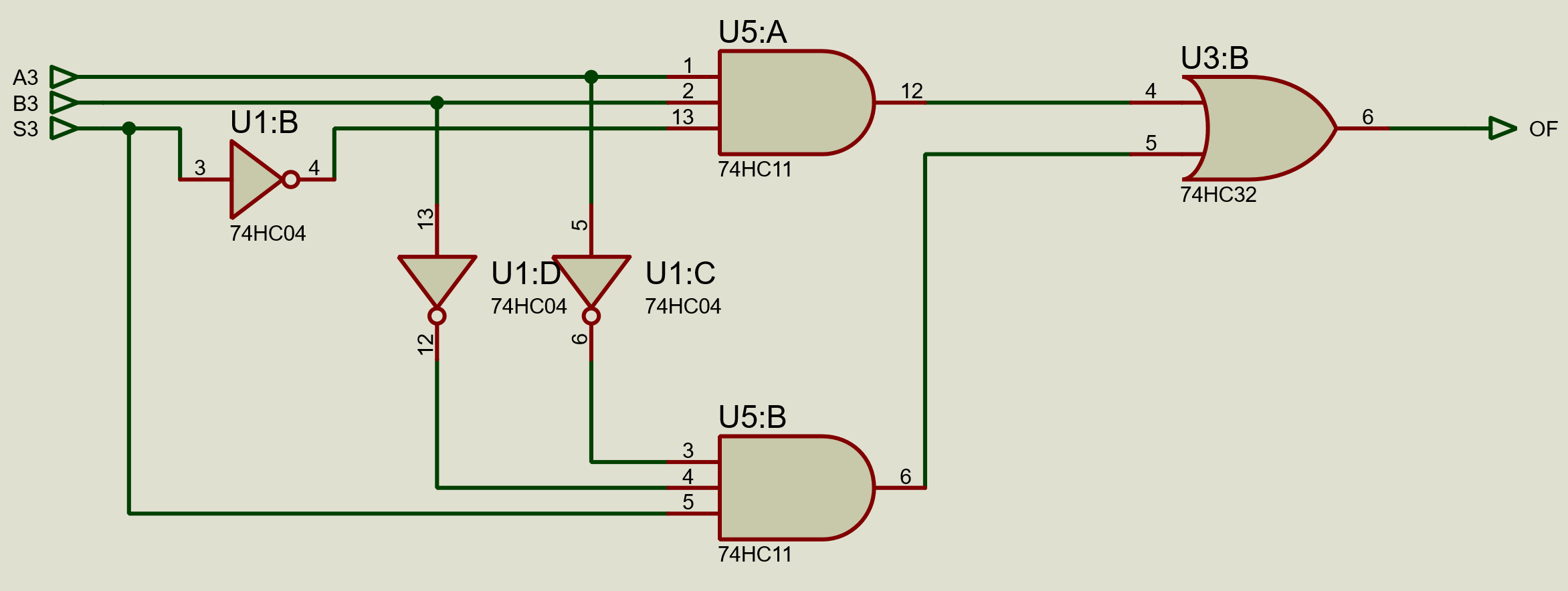
Na kraju za komparaciju Z brojeva koristimo dodatni input-CTRL i XOR kola. Kada je CTRL 0 kompariramo N brojeve i ulazi ostaju nepromenjeni, a kada je 1 kompariramo Y brojeve tako što MSB (bitove koji određuju predznak broja) negiramo pomoću XOR kola.

## 2.4 Dodatni moduli

### *2.4.1 Overflow modul*



*Tabela 13. – tablica istinitosti za overflow bit*

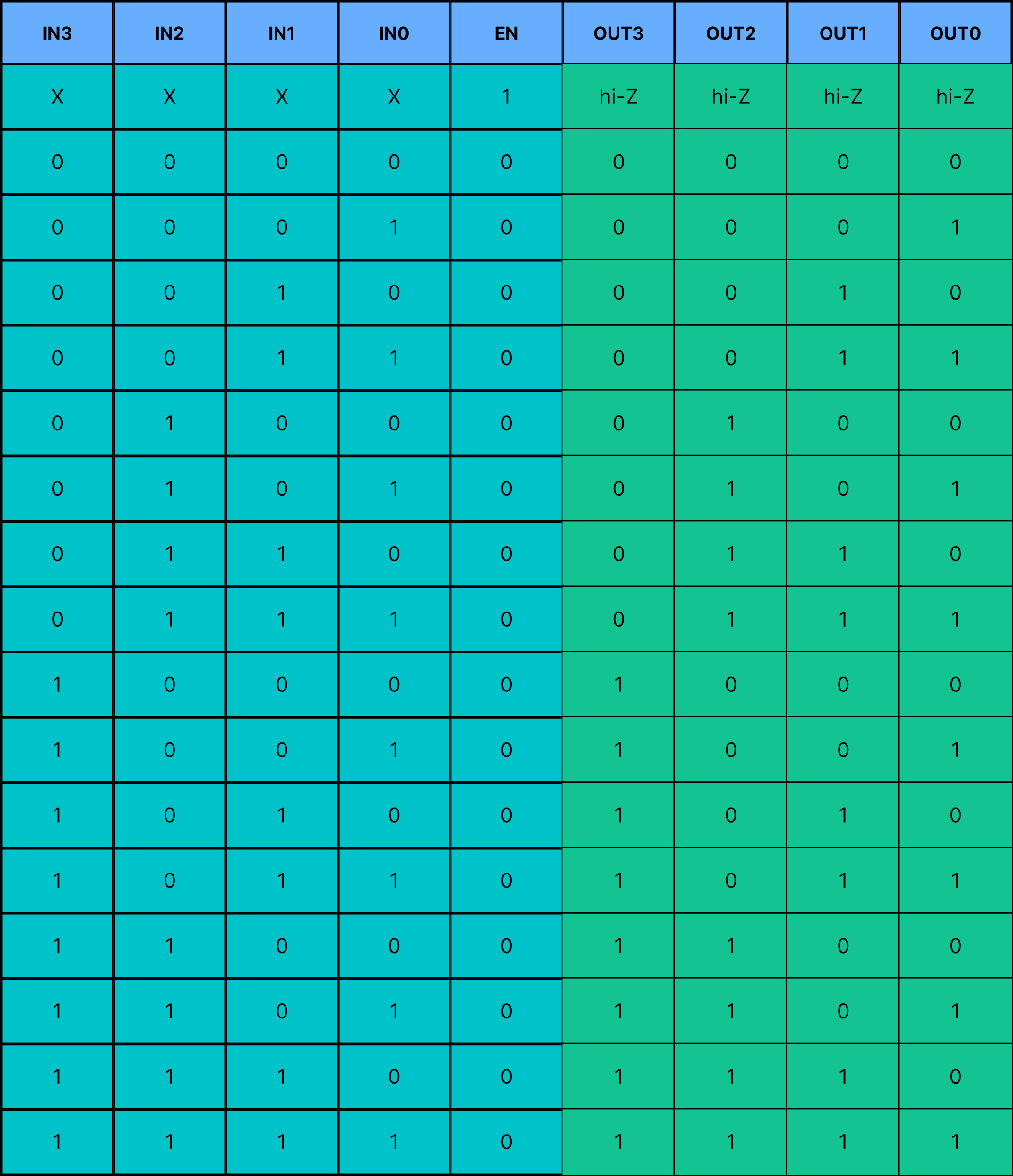


*Slika 13. – šema overflow modula*

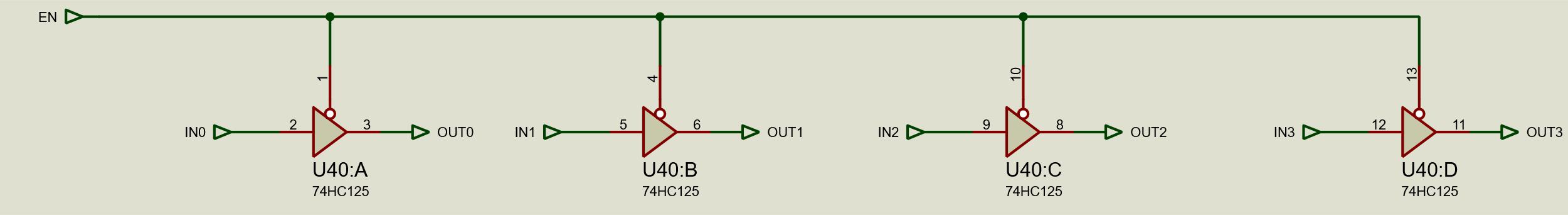
Uz NOT, AND i OR kola proveravamo da li je izlaz kod predznačnog sabiranja i oduzimanja validan. Overflow bit je 1 ako je predznak (MSB) ulaza isti, ali se predznak izlaza ne poklapa sa njihovim. Ako je overflow 1 znamo da je rezultat izvan opsega od 4 bita.

Overflow bit ima stanje visoke impedanse kada se ne bavimo aritmetičkim funkcijama ili za drugi komplement.

### *2.4.2 Buffer modul*



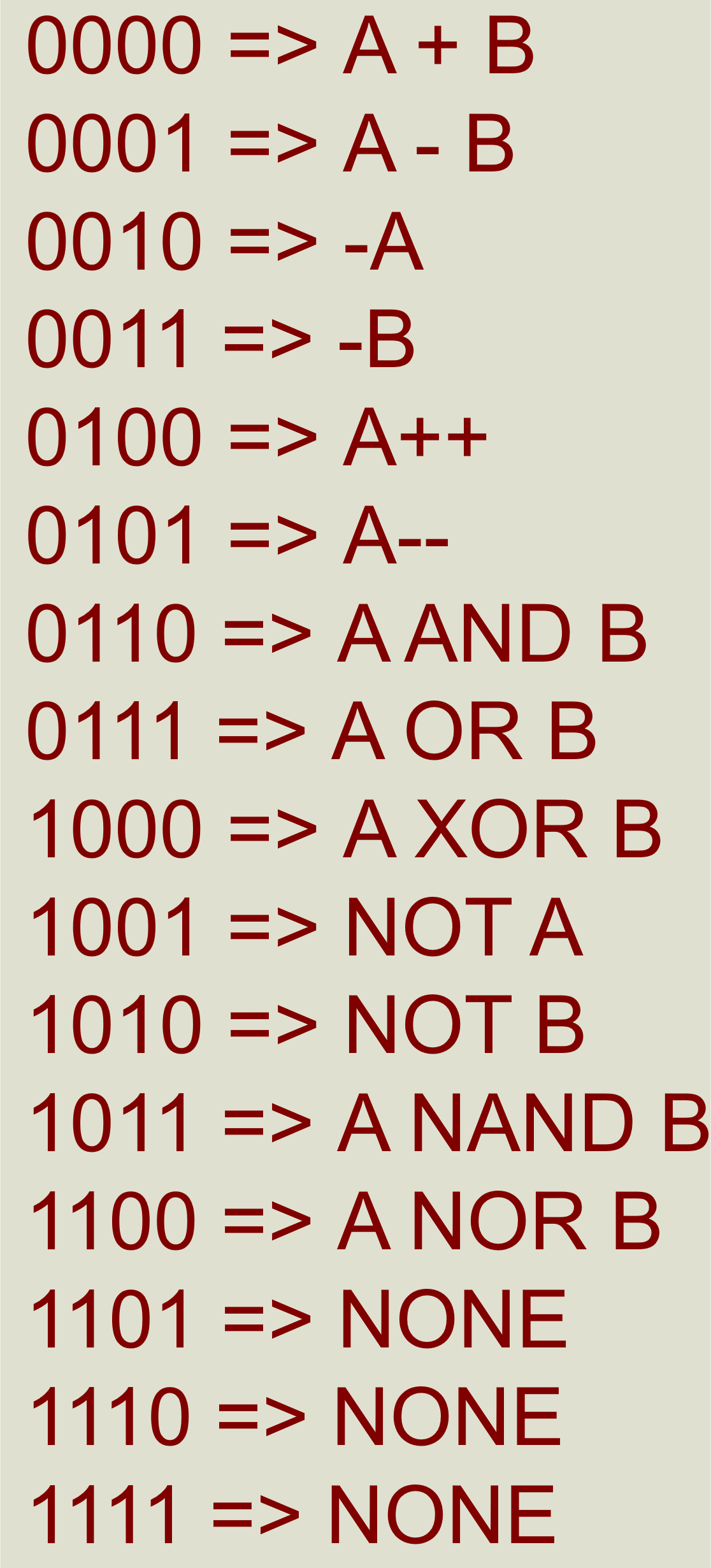
*Tabela 14. – tablica istinitosti za buffer modul*



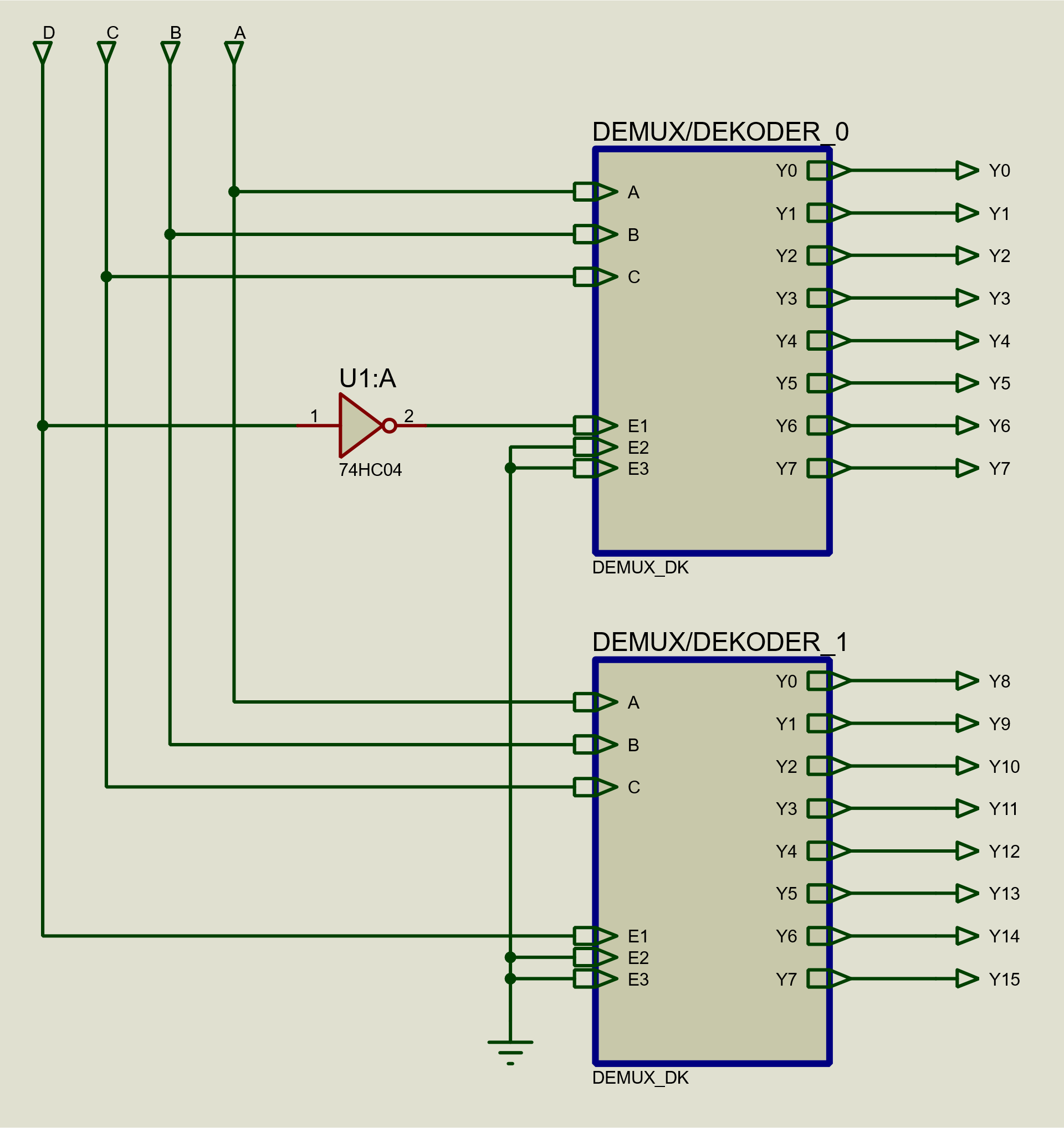
*Slika 14. – šema buffer modula*

Pošto imamo mnogo funkcija koristimo buffer modul za gašenje određenih izlaza koji nam ne trebaju. Pošto ima negativnu logiku on omogućava izlaze kada je EN = 0, a gasi ih kada je 1. Ovaj modul koristimo posle svakog drugog modula (osim komparatora) i u EN stavljamo izlaz dekodera sa negativnom logikom, tako možemo da biramo koju funkciju želimo na izlazu.

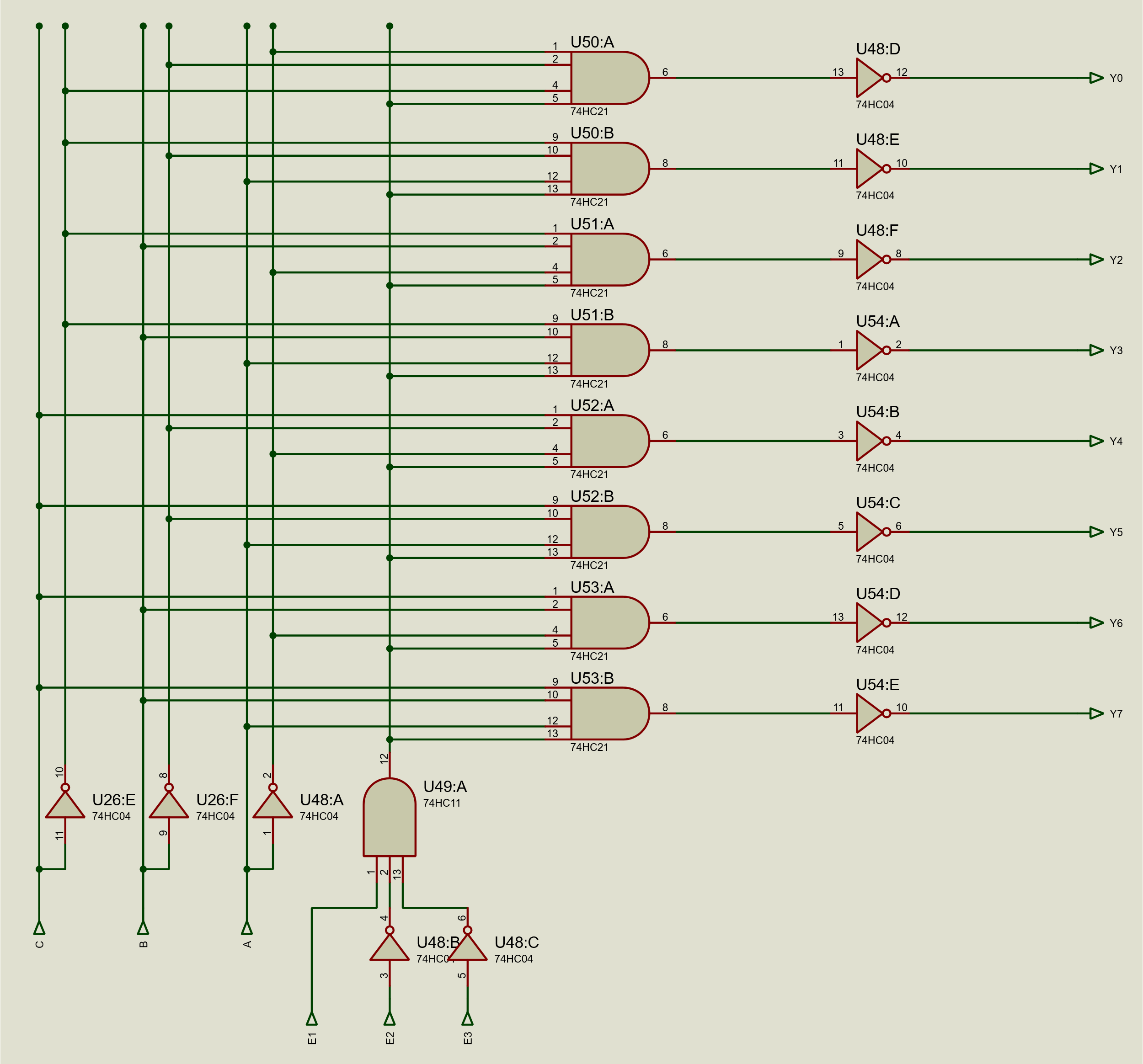
### *2.4.2 Dekoder sa negativnom logikom*



*Slika 15. – izbor funkcije kod 4 na 16 dekodera sa negativnom logikom*

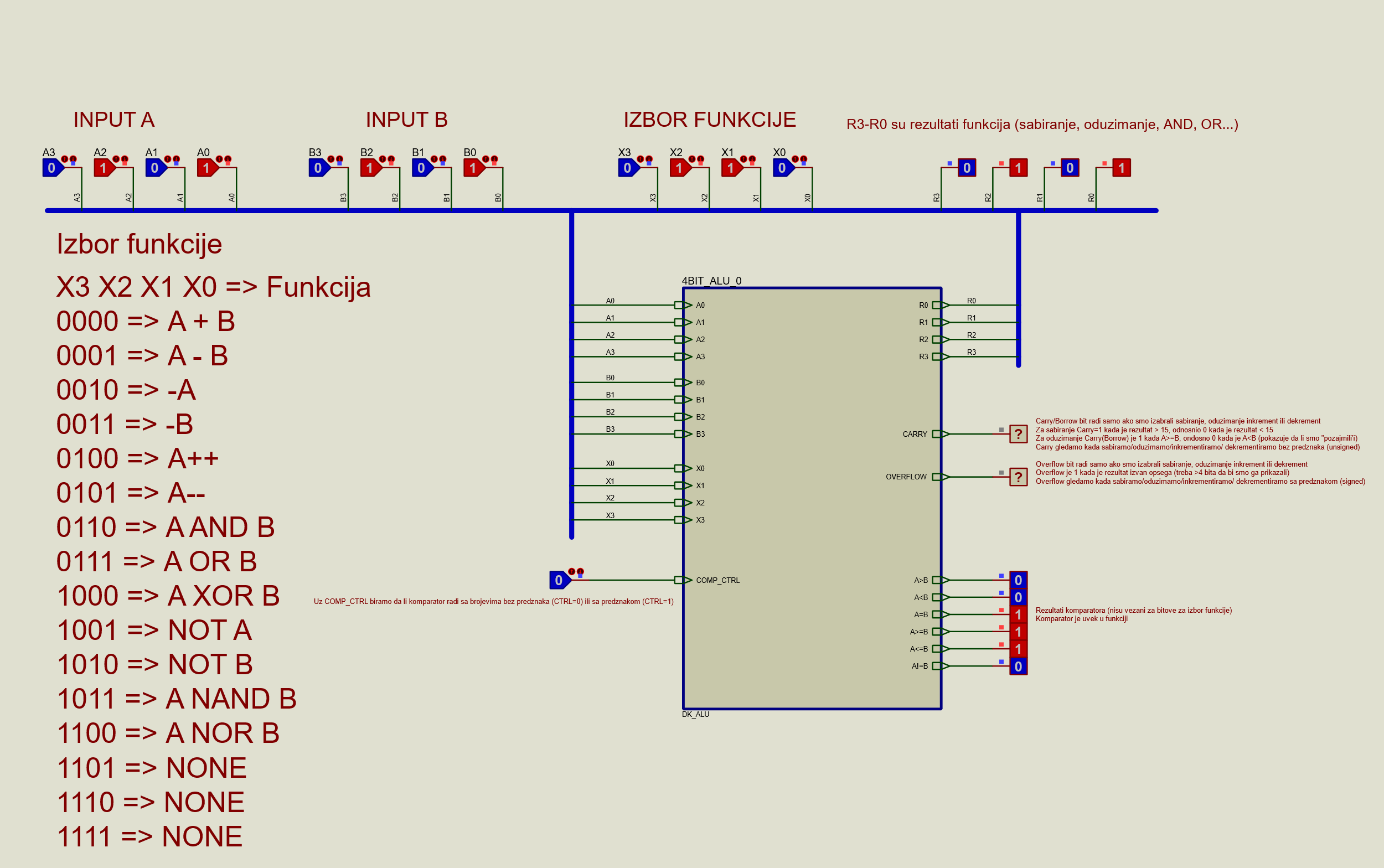


*Slika 16. – šema 4 na 16 dekodera sa negativnom logikom*



*Slika 17. – šema 3 na 8 dekodera sa negativnom logikom*

# **3. PRIKAZ SIMULACIJE**



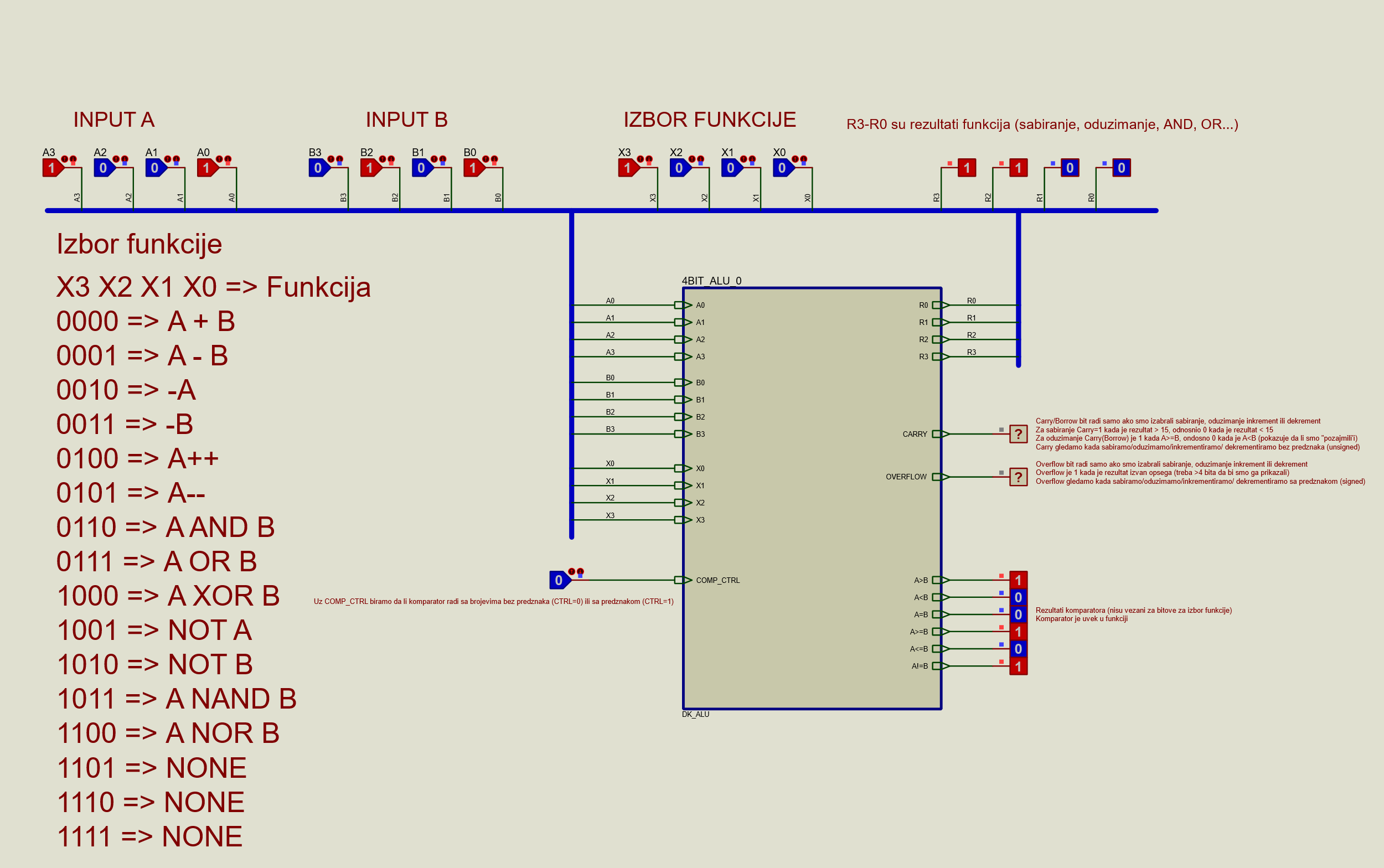
*Slika 18. – A AND B*



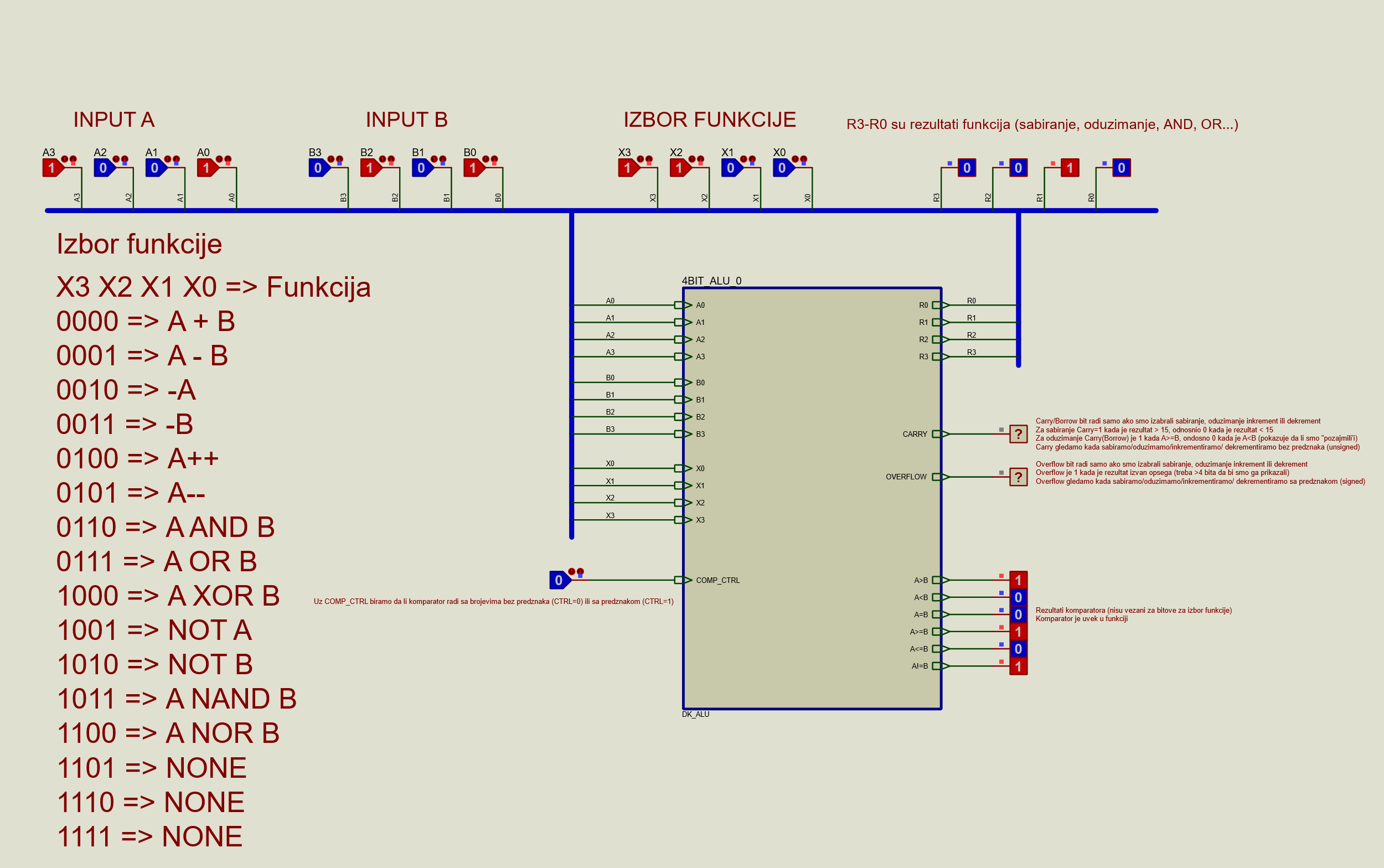
*Slika 19. – A++*



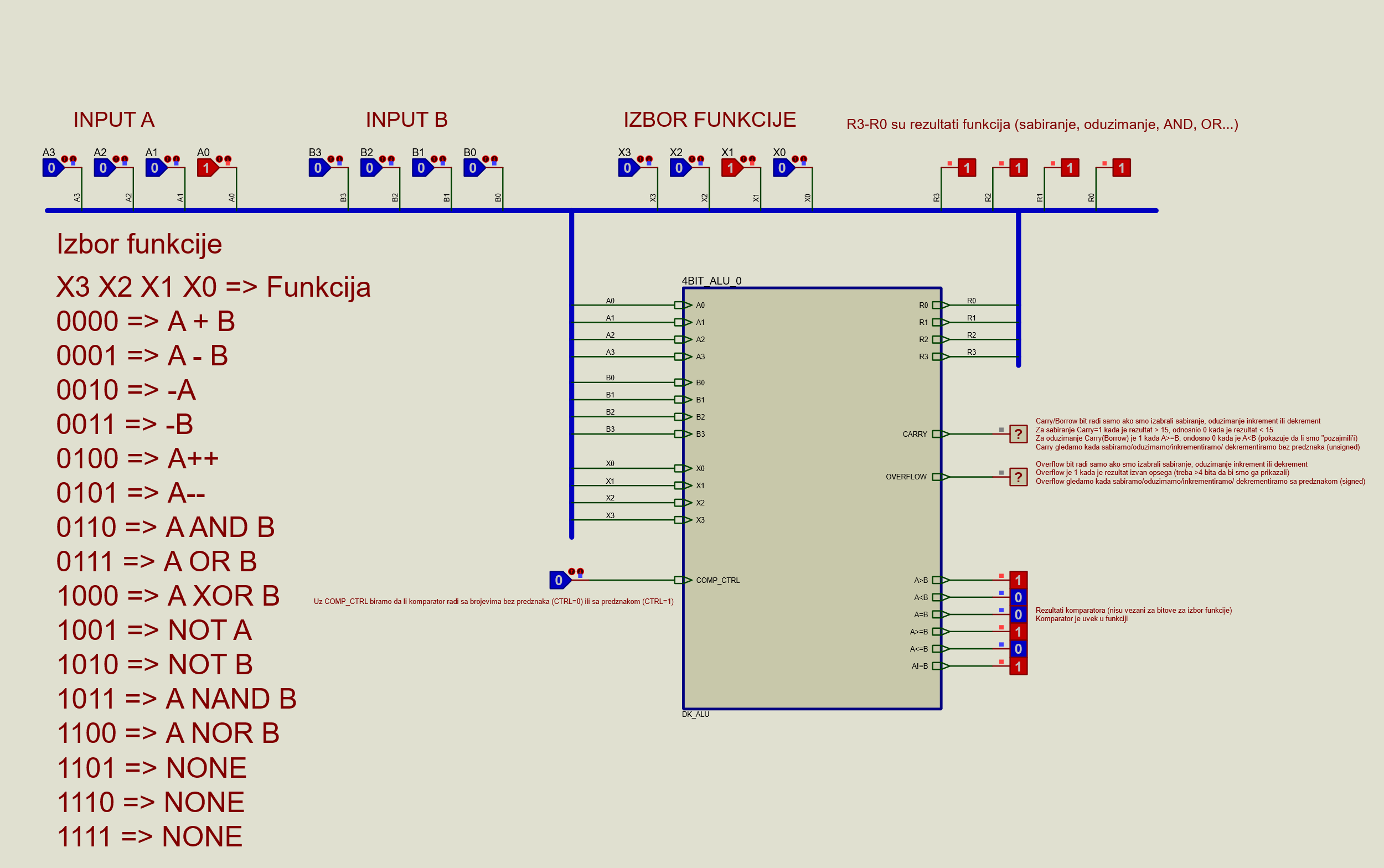
*Slika 20. – A + B*



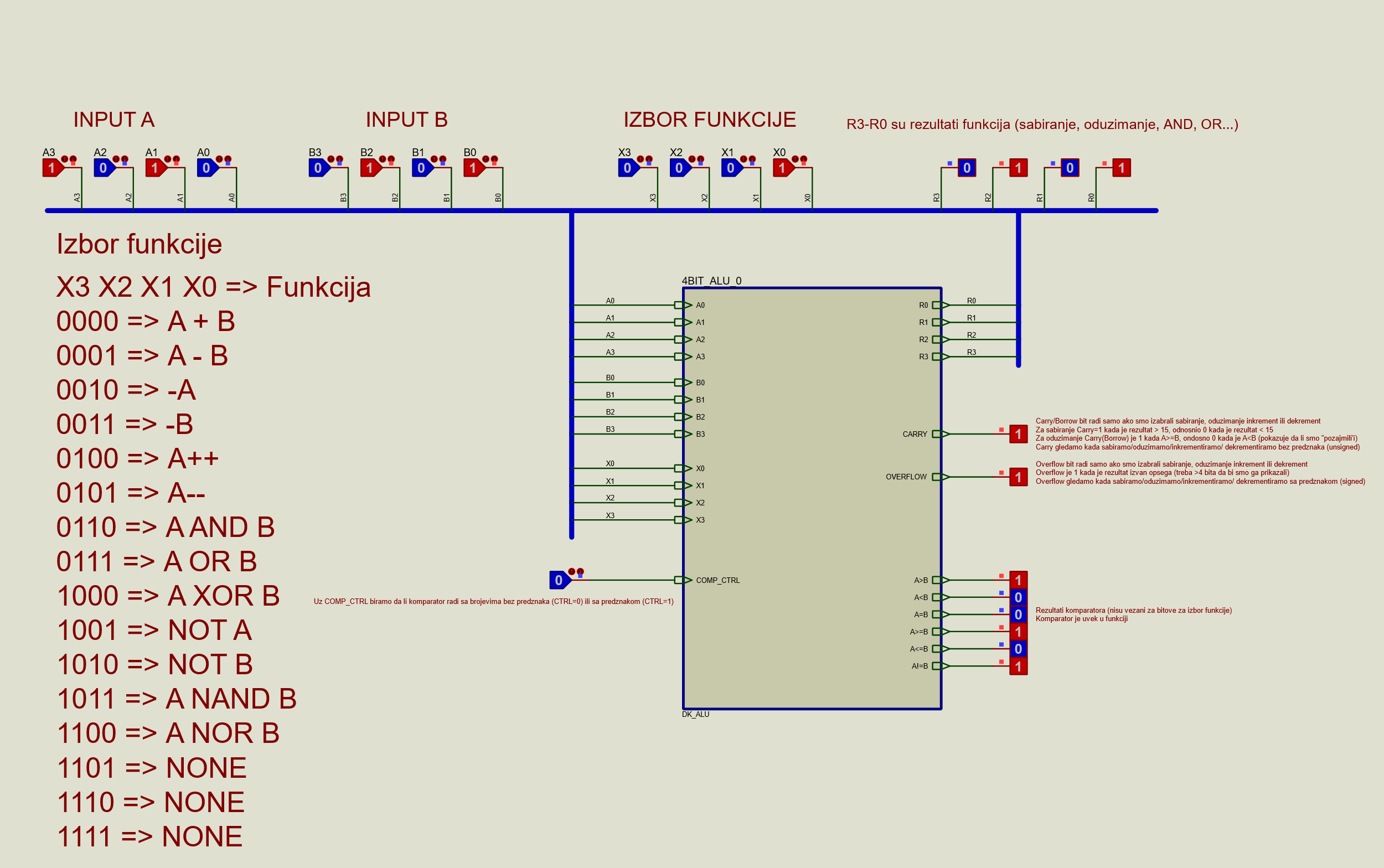
*Slika 21. – A XOR B*



*Slika 22. – A NOR B*

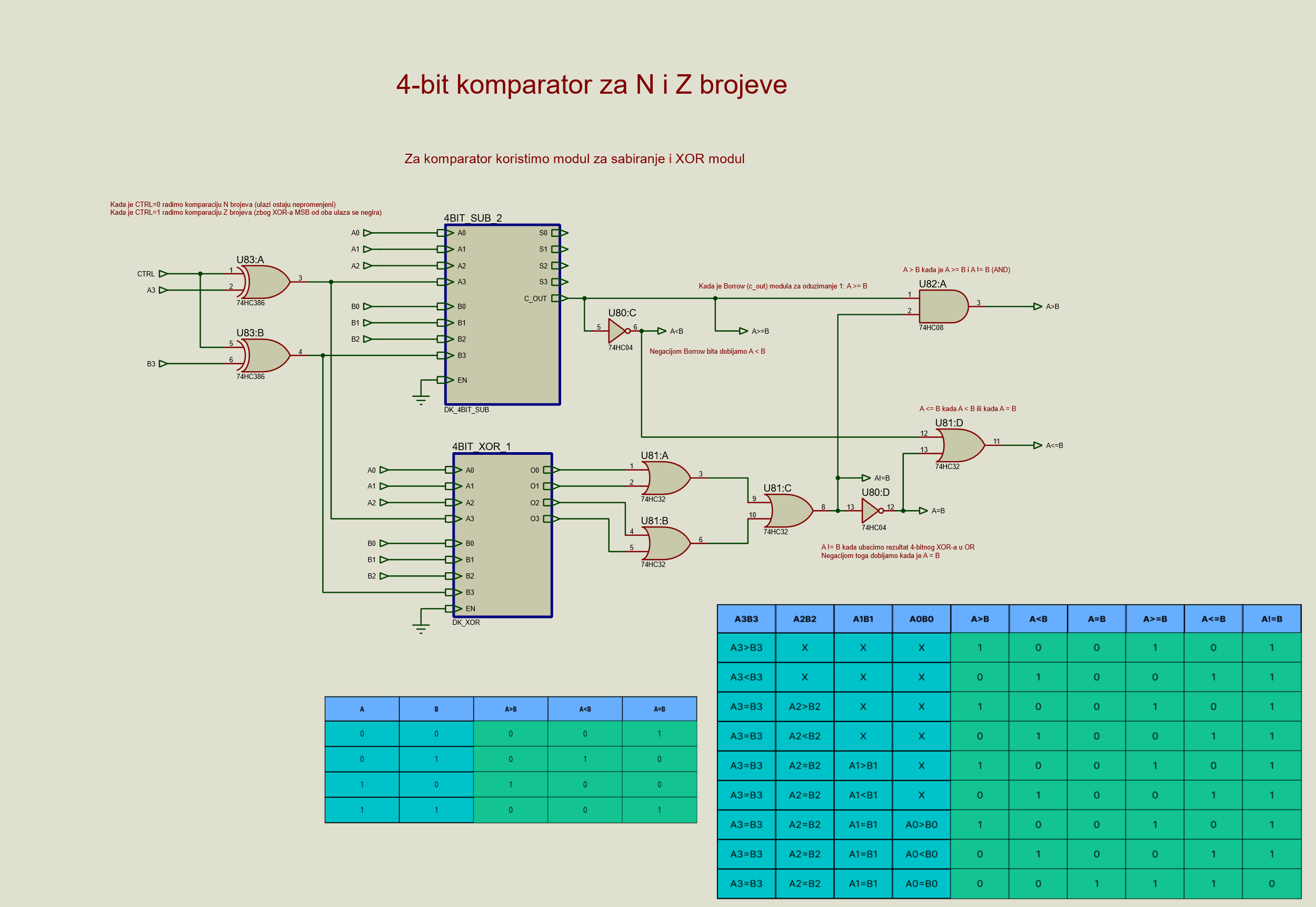
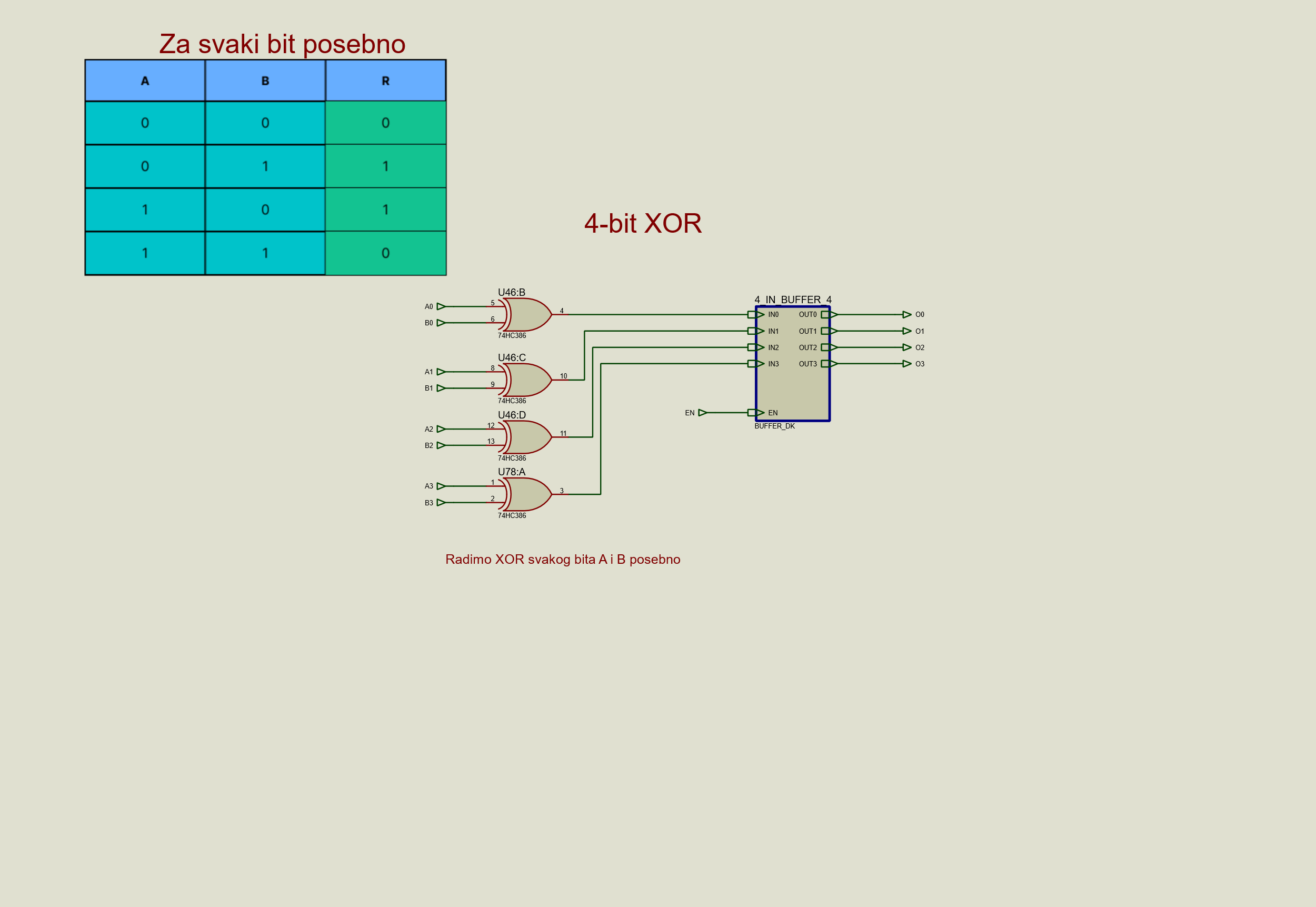
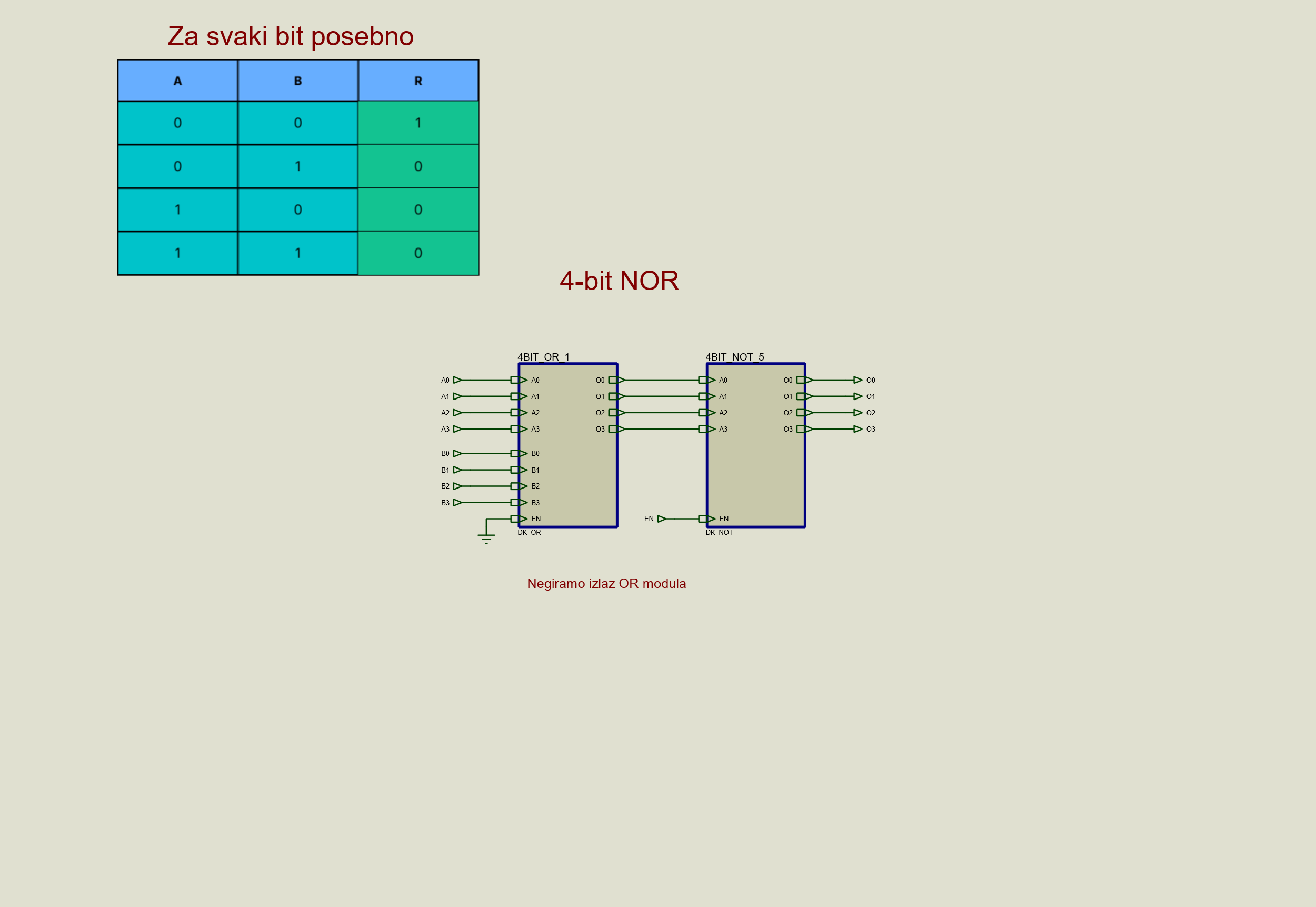
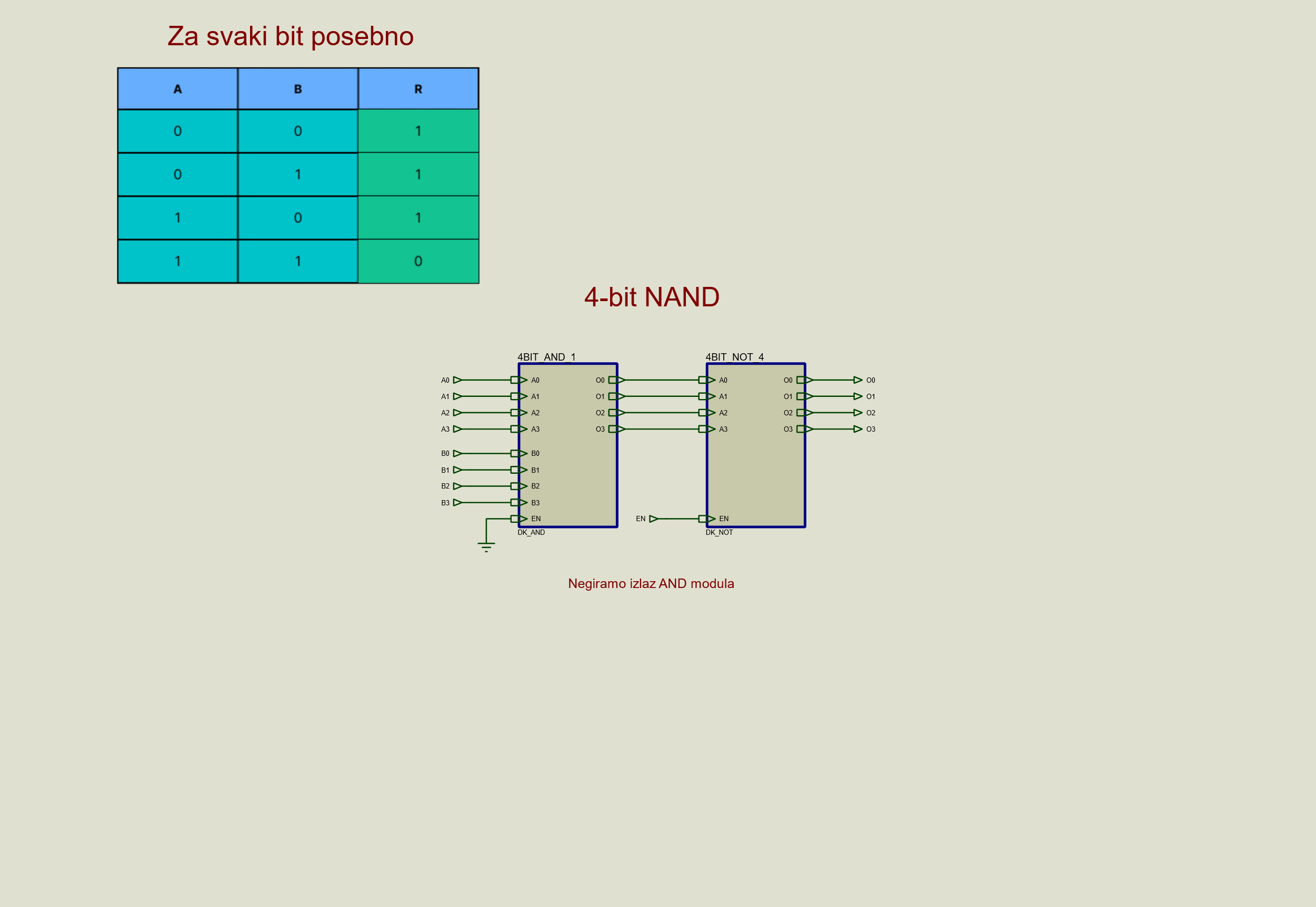
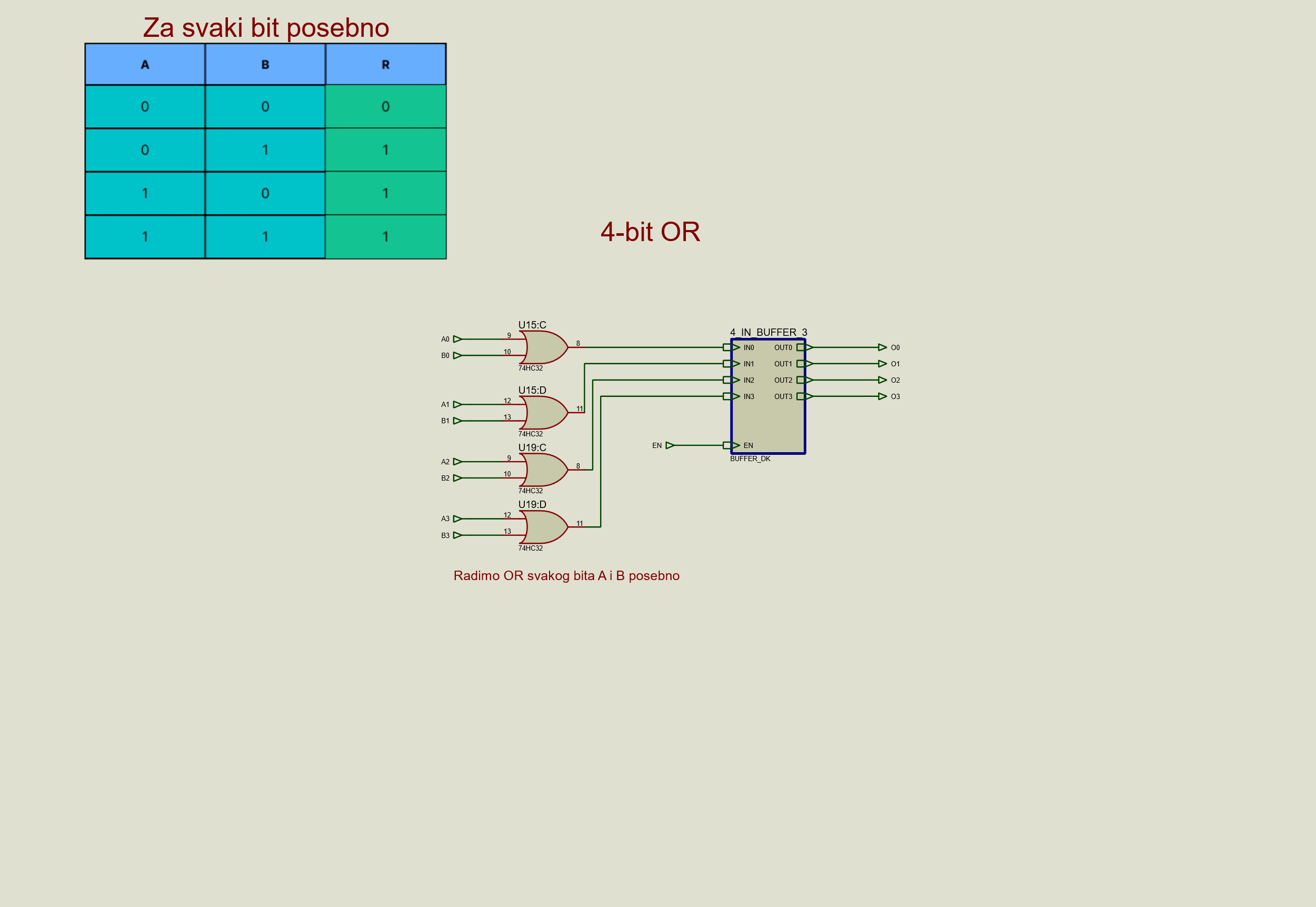
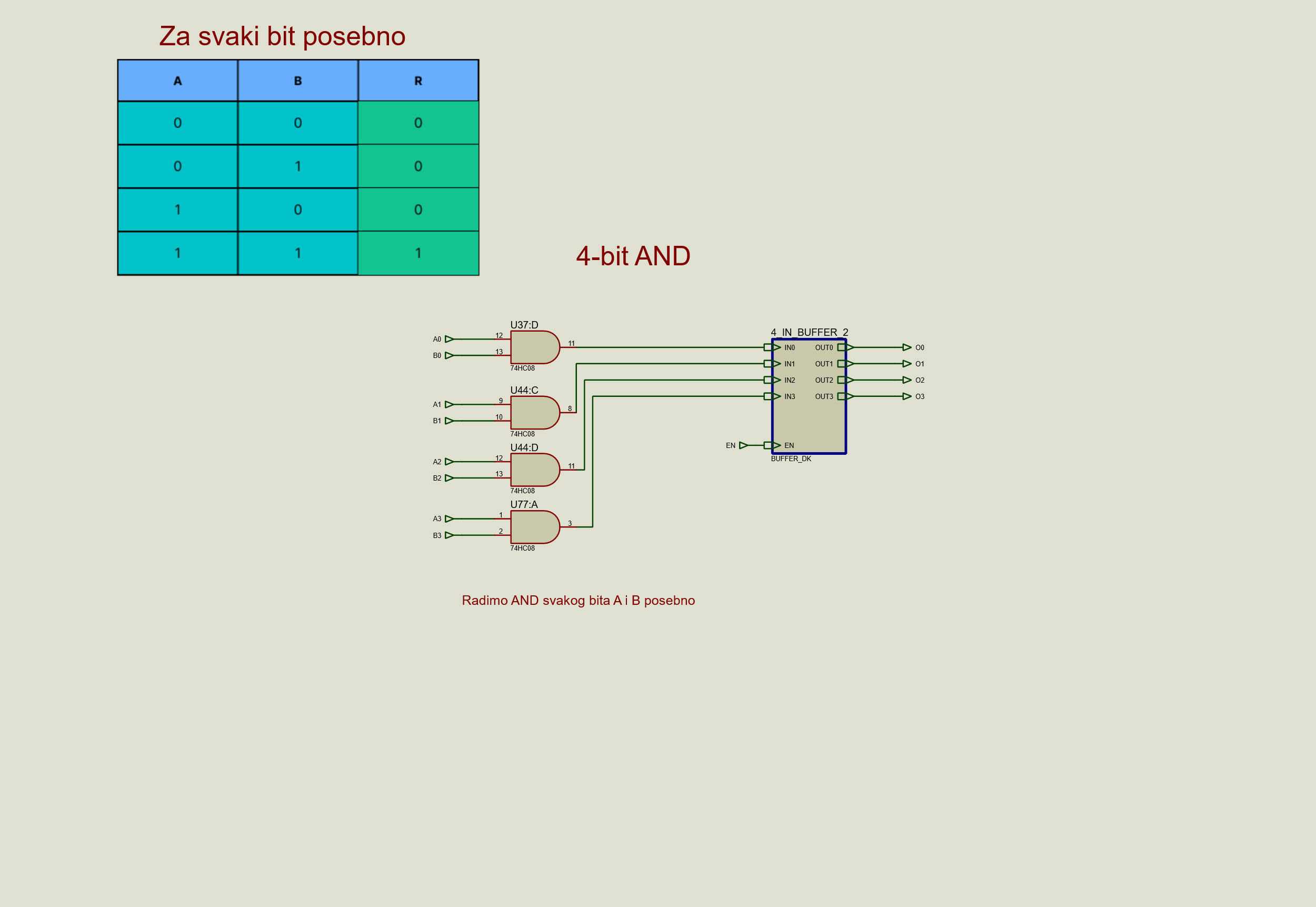
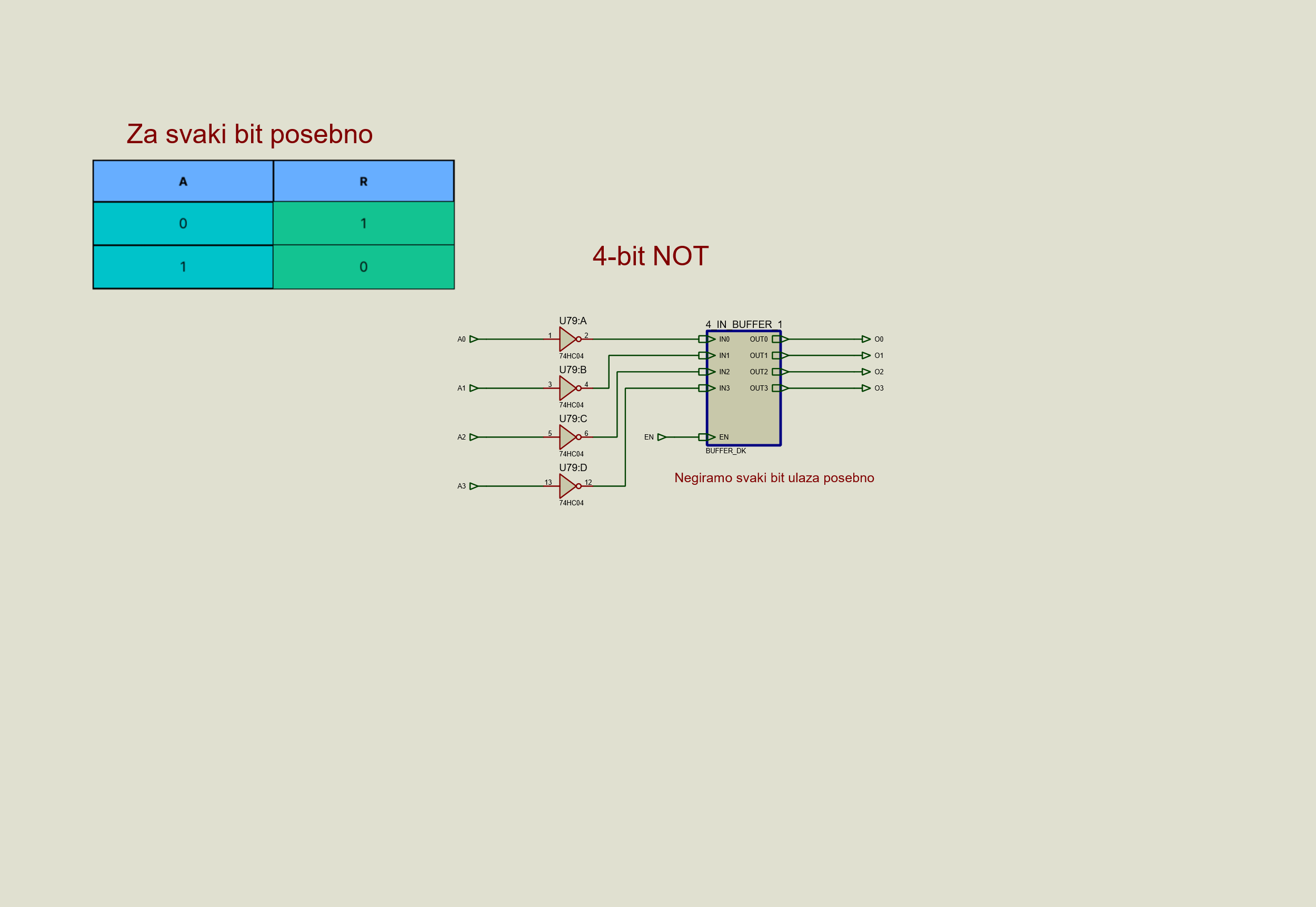
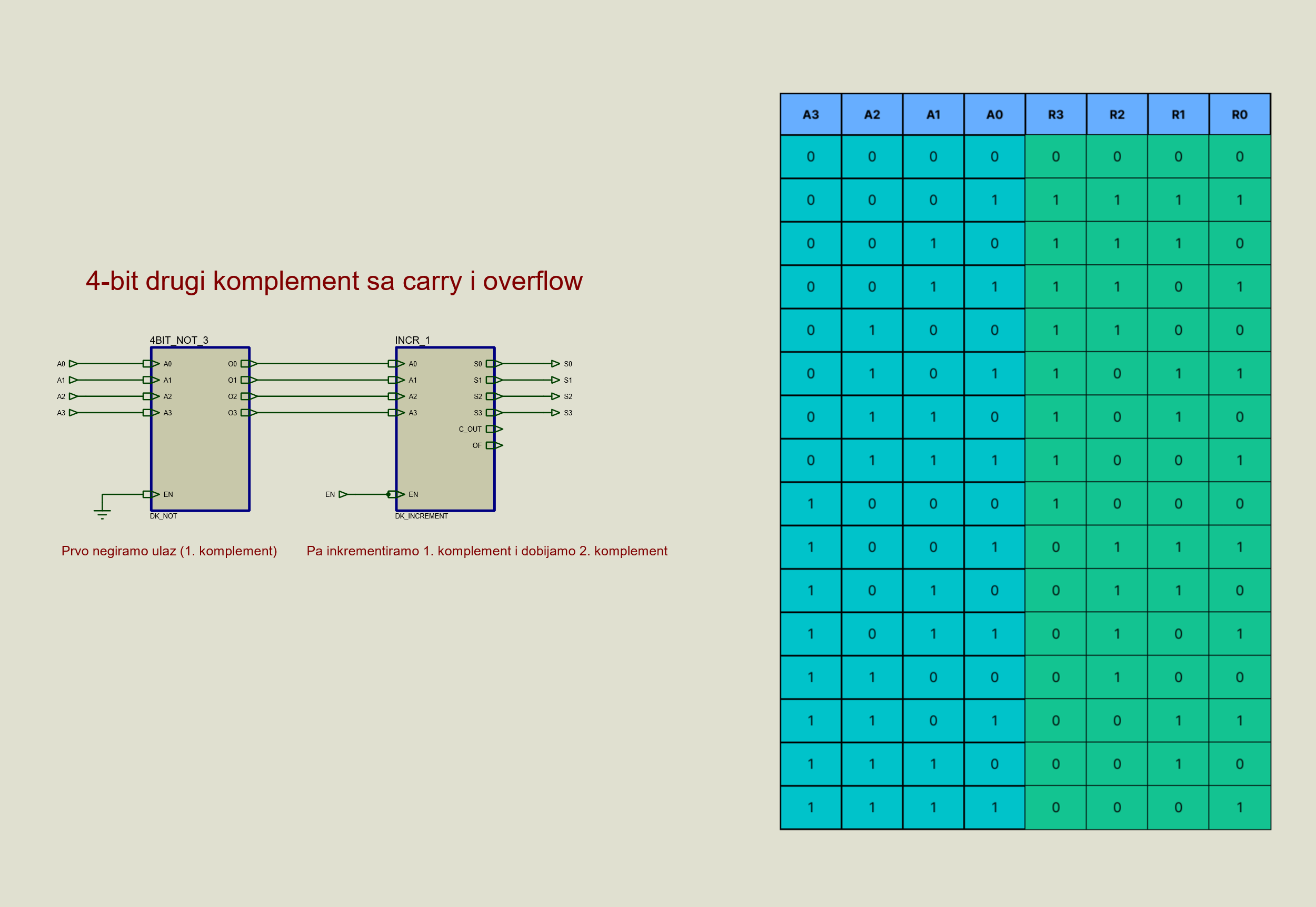
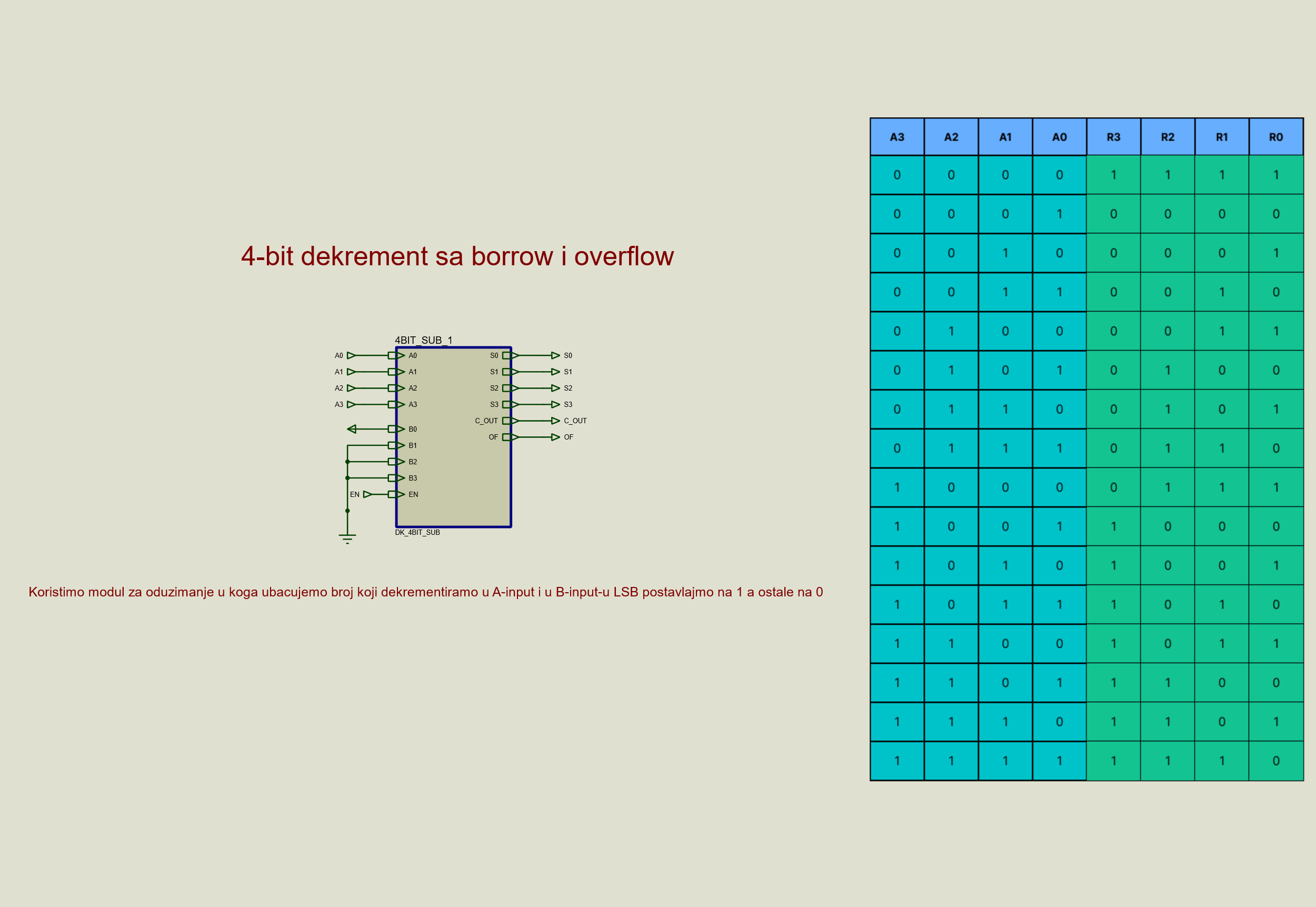
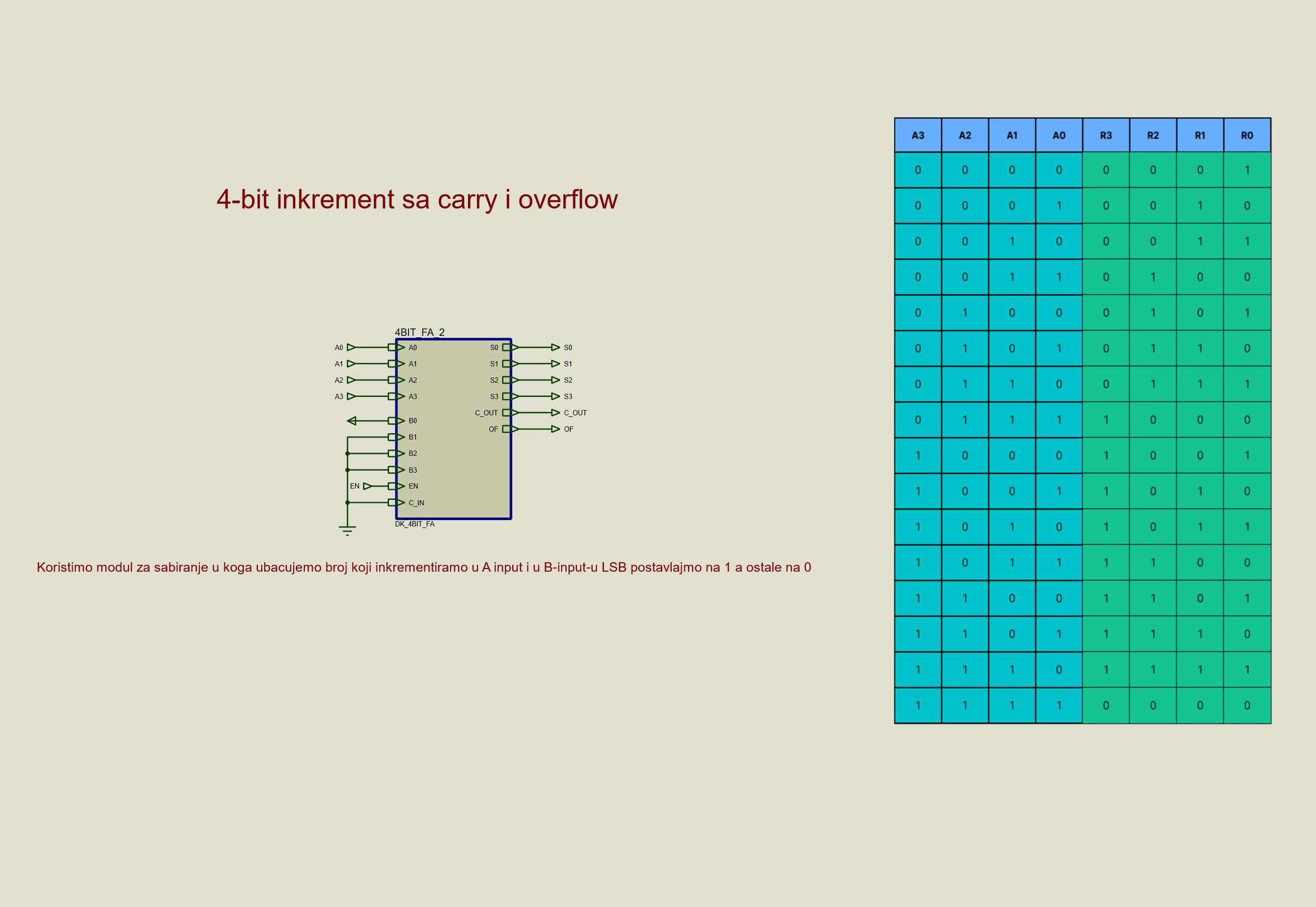
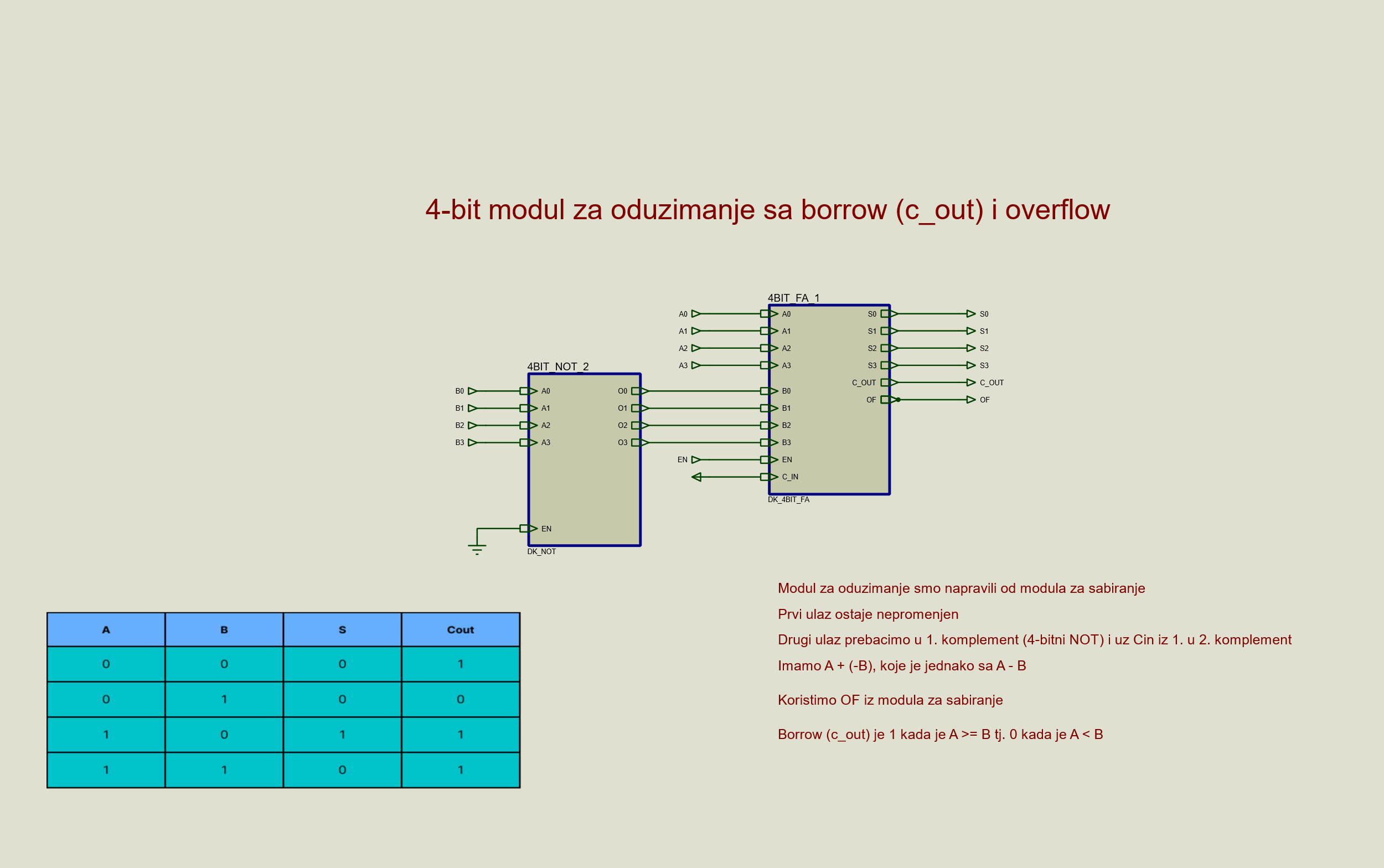
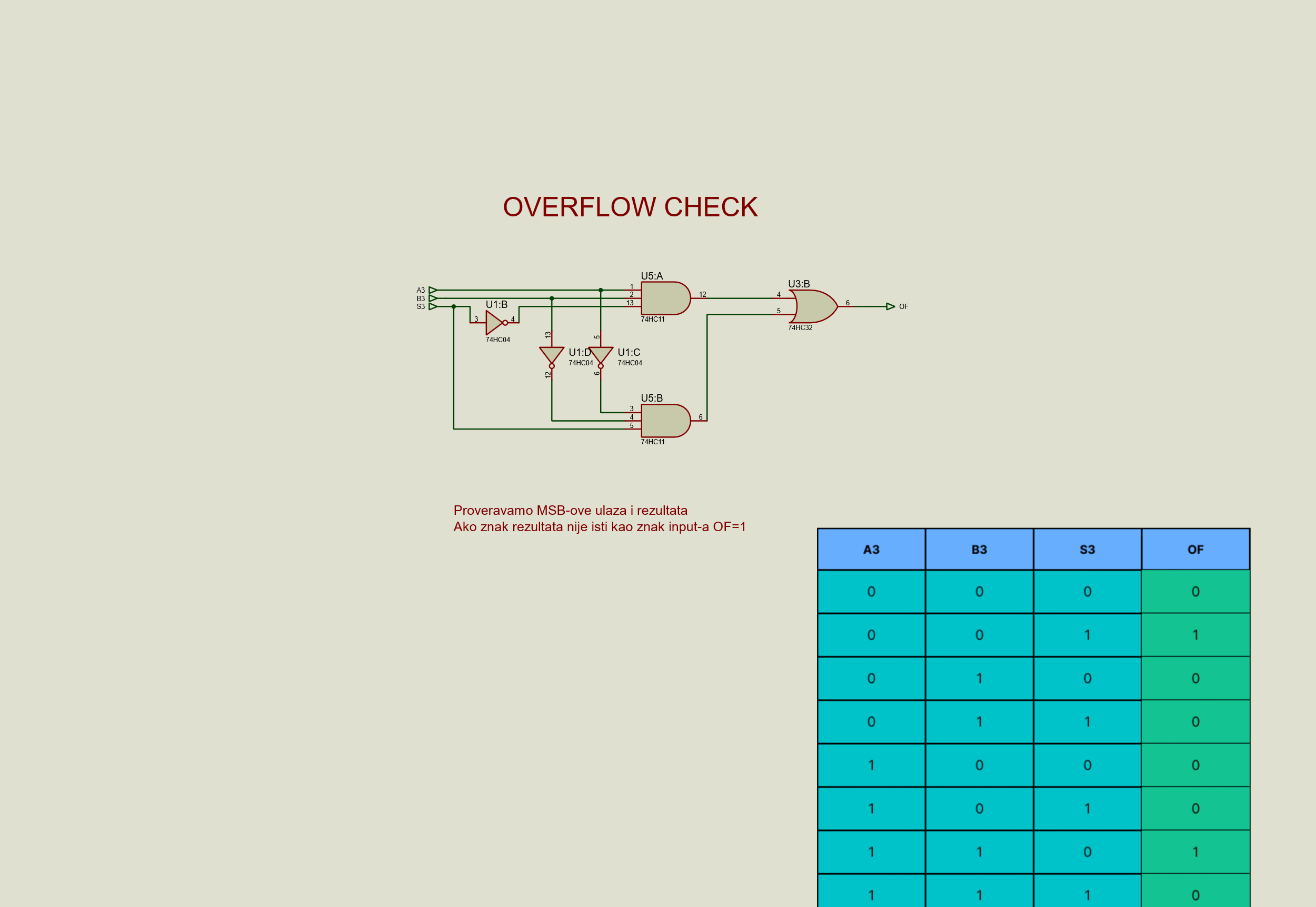
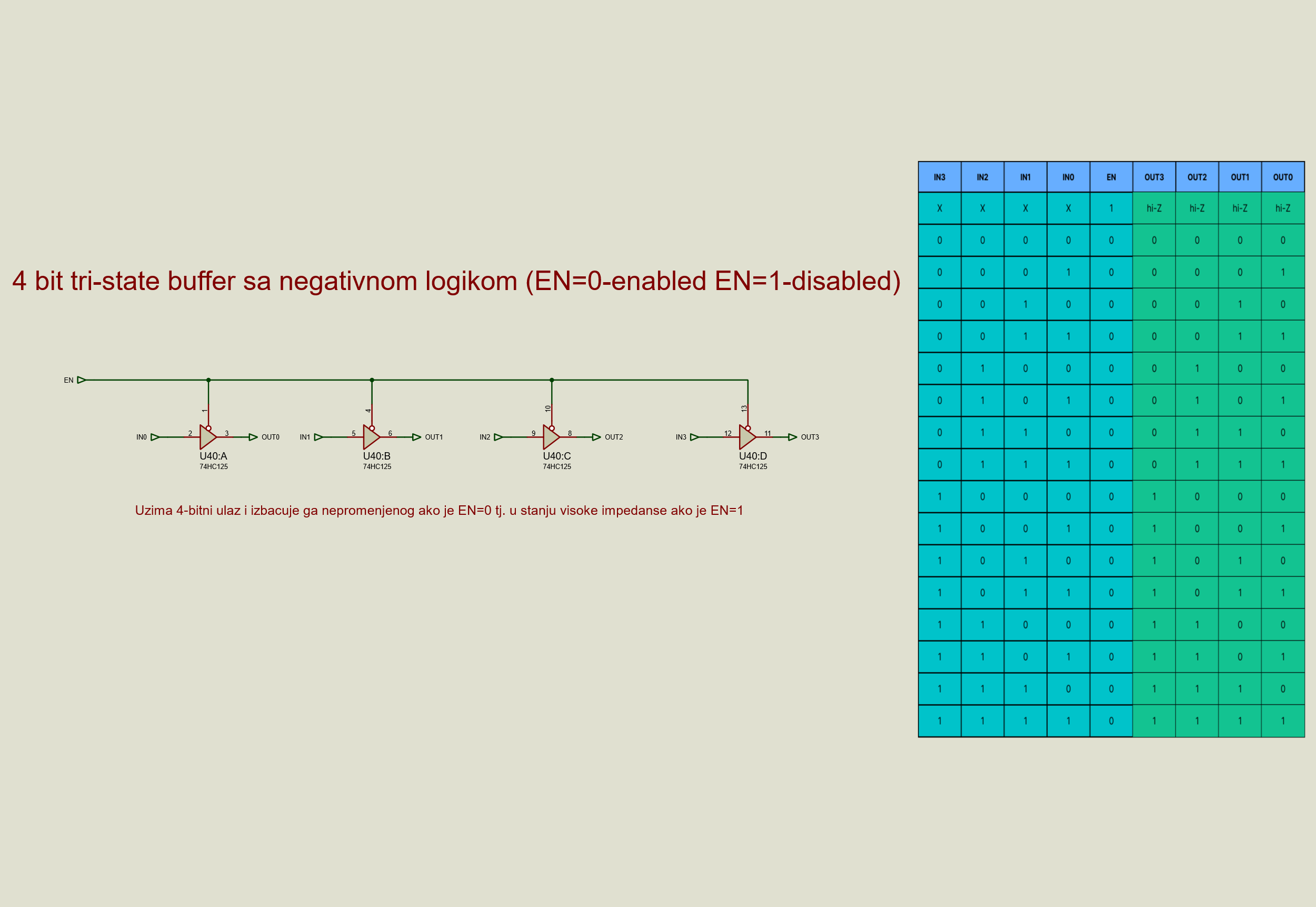
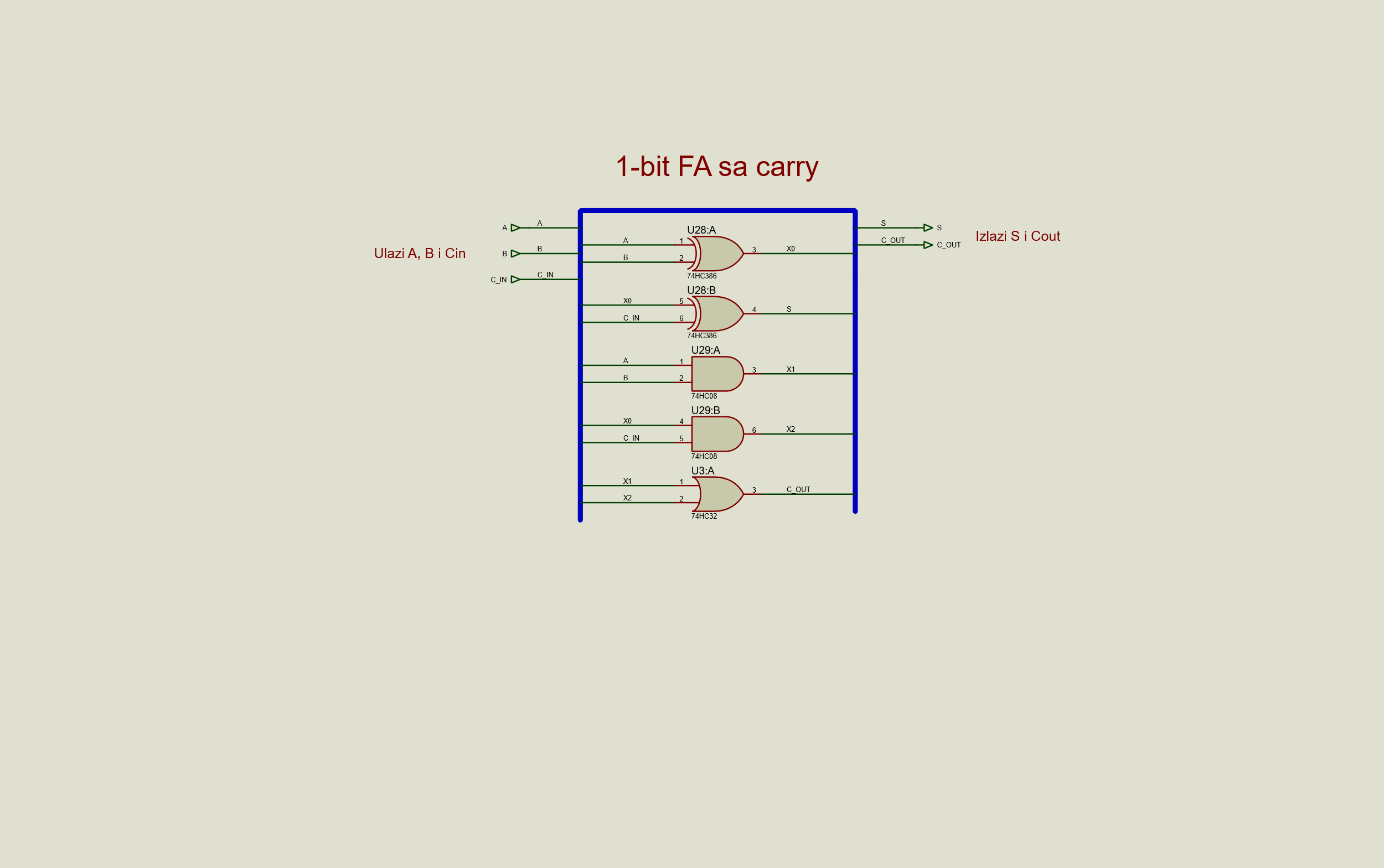
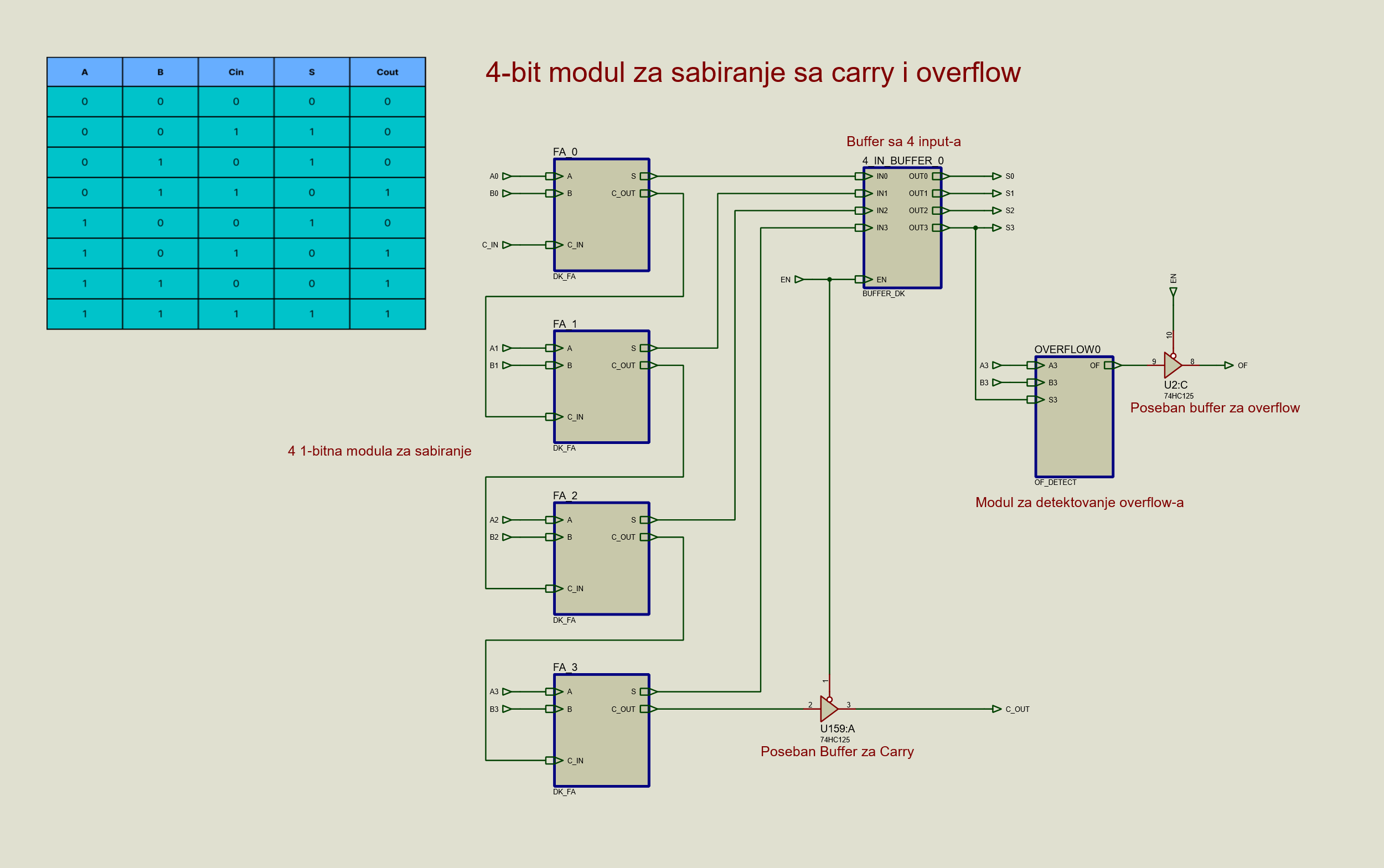
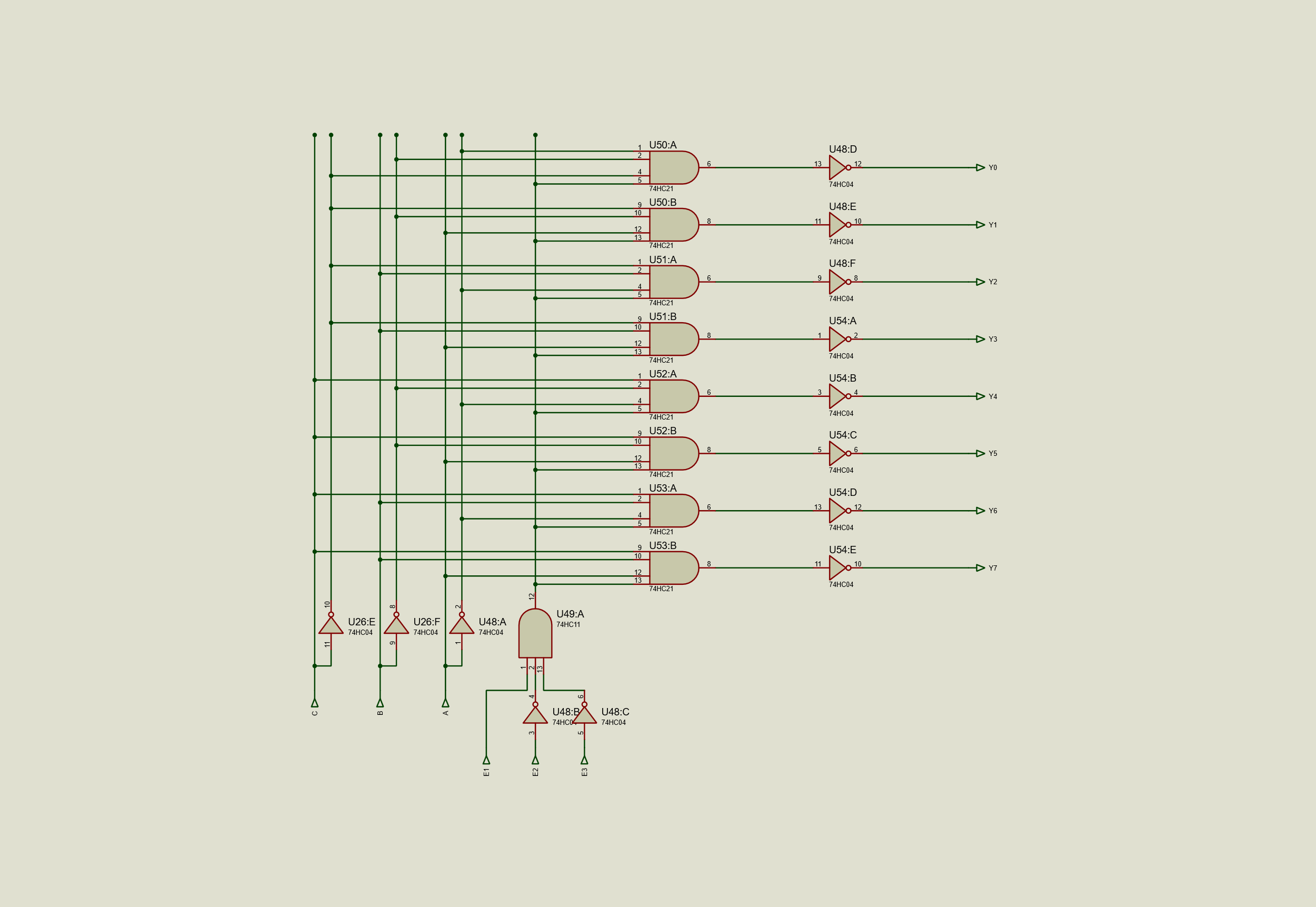
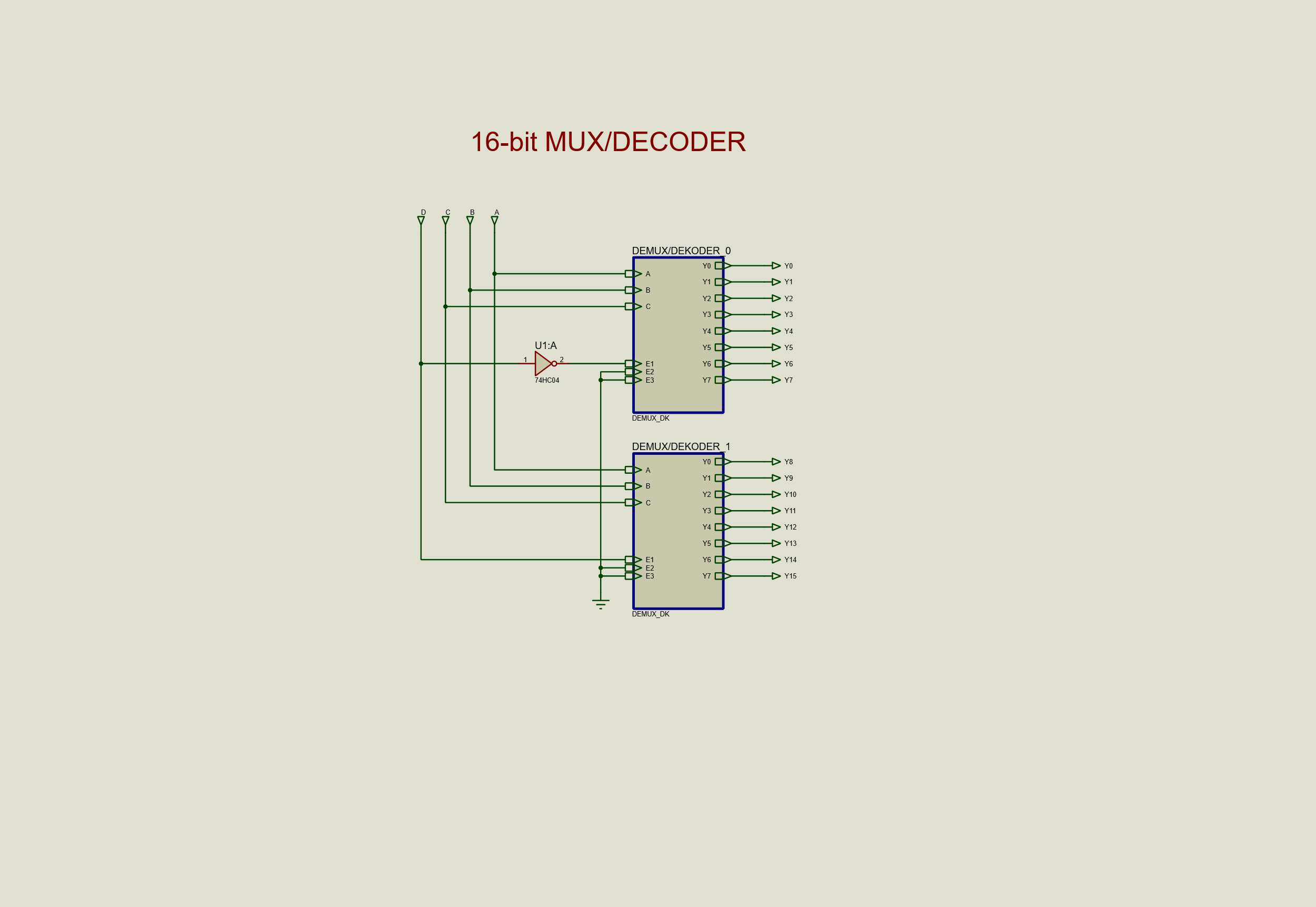
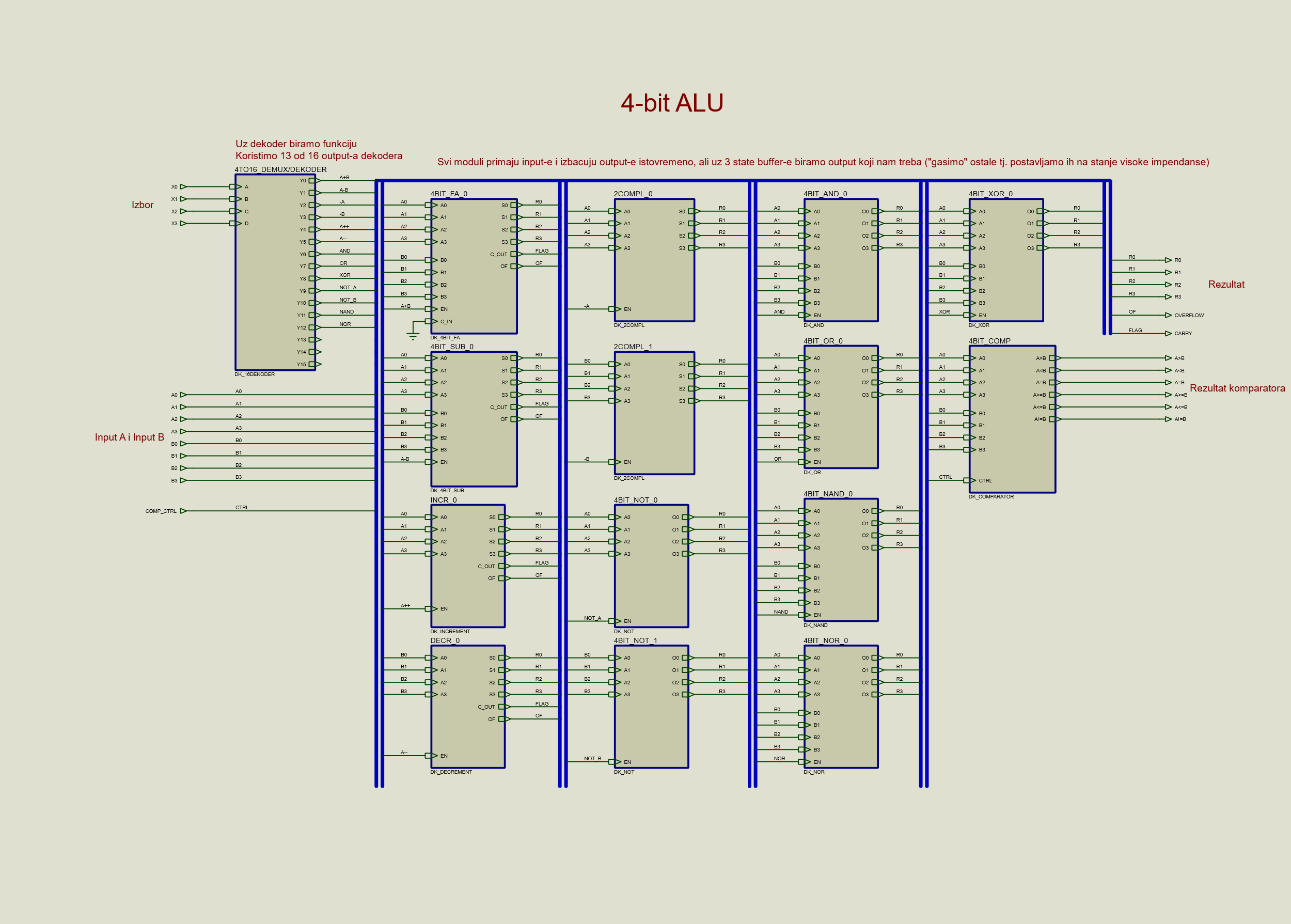
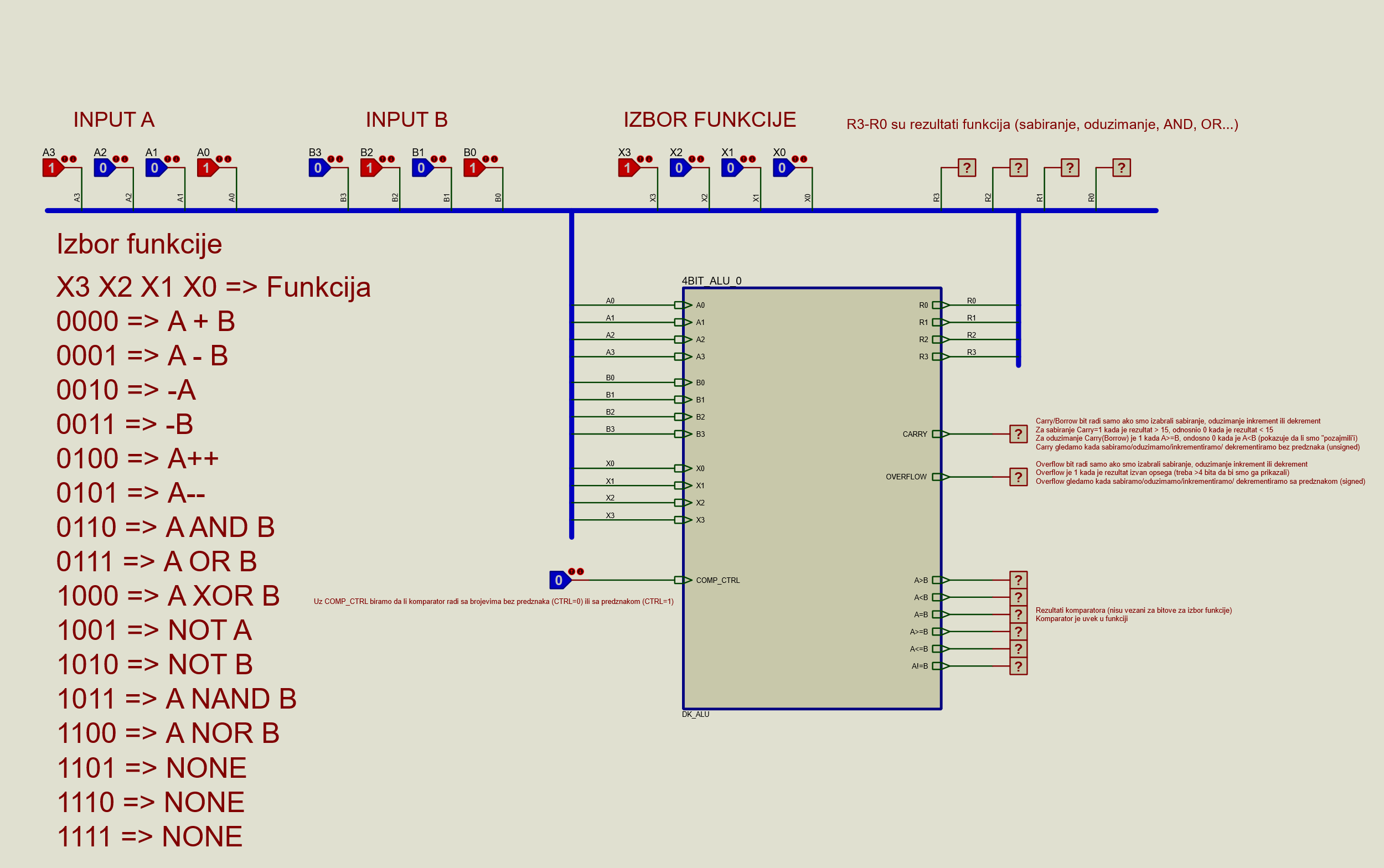


*Slika 23. – -A*



*Slika 24. – A - B*

# **4. PRILOG**



# **5. LITERATURA**

[W1] Tablica istinitosti za komparator - https://technobyte.org/2-bit-4-bit-comparator/