**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**ELEKTROTEHNIČKI FAKULTET**

**Sveučilišni studij**

# ARITMETIČKO LOGIČKA JEDINICA

# ( ALU )

**Završni rad**

**Davor Bogdanović**

**Osijek, rujan 2010.**

1. Uvod ---- 1
2. Aritmetičke operacije ---- 2
2.1 Operacija zbrajanja ---- 2
2.1.1 Paralelno zbrajanje binarnih brojeva ---- 4
2.1.2 Serijsko zbrajanje binarnih brojeva ---- 5
2.1.3 Realizacija operacije zbrajanja pomoću kombinacijskih sklopova ---- 6
2.2 Prikaz binarnih brojeva sa predznakom ---- 8
2.3 Komplementi brojeva ---- 9
2.3.1 B-ti komplement ---- 9
2.3.2 (B - 1) – i komplement ---- 10
2.4 Operacija oduzimanja ---- 11
2.4.1 Oduzimanje s (*B – 1)* – tim komplementom ---- 12
2.4.2 Oduzimanje s *B* – tim komplementom ---- 14
2.5 Množenje ---- 15
2.5.1 Booth – ov algoritam za množenje ---- 20
2.6 Dijeljenje ---- 24
2.6.1 Restoring i non – restoring algoritam za dijeljenje ---- 25
3. Logičke operacije ---- 28
3.1 Logički sklop NE ---- 29
3.2 Logički sklop I ---- 29
3.3 Logički sklop ILI ---- 31
4. Aritmetičko – logička jedinica (ALU) ---- 34
4.1 Carry – look ahead ---- 37
4.2 Usporedba carry ripple i carry – look ahead metode ---- 42
4.3 Simulacija ---- 43

4.4 Realizacija ----------45
5. Zaključak ----------48
6. Literatura----------49
6.1 Popis korištenih knjiga:----------49
6.2 Popis korištenih web stranica: ----------49
7. Sažetak ----------50
7.1 Aritmetičko logička jedinica ( ALU ) ----------50
7.2 Arithmetic logic unit (ALU) ----------50
8. Životopis----------51
9. Prilozi ----------52
9.1 P. 9.1 Specifikacije 74181 sklopa----------52
9.2 P.9.2. Izvještaj o vremenu propagacije realiziranog sklopa ----------54
9.3 P.9.3. VHDL kod aritmetičko logičke jedinice ----------56

# 1. Uvod

Osnovni dio računala i općenito svakog mikroprocesora je aritmetičko - logička jedinica (engl. Arithmetic - logic unit). Aritmetičko - logička jedinica je sastavni dio računala od samog začetka računalne tehnologije, te se nalazi u samom Von Neumann-ovom modelu računala. Koristi se za sve logičke operacije, samo za osnovne aritmetičke operacije, kao posmačni sklop, itd. Sve aritmetičke i logičke operacije su opisane, tablično objašnjene i napisane pomoću VHDL koda. Procesor je odgovoran za korištenje informacija koje se nalaze u programskoj memoriji ( naredbe ) za kontroliranje određenih procesa, te ulazno-izlaznih uređaja. Većina naredbi funkcionira na podatkovnoj memoriji. Za funkcioniranje na razini podatkovne memorije potrebna je aritmetičko-logička jedinica. Sastoji se od *n*-bitnih ulaza A i B nad kojima se obavljaju aritmetičke ili logičke operacije, CarryIn ulaza koji predstavlja prijenosni bit, selektorski *s*-bitni ulaz kojim određujemo određeni tip operacije koju obavlja ALU, izlazni n-bitni signal koji predstavlja rezultat operacije obavljene nad ulaznim operandima, te CarryOut jednobitnog signala koji označava prijenosni bit. Prilikom realizacije ALU korištena je integrirana 4-bitna 74181 ALU. Na realiziranom modelu su isprobane sve aritmetičke i logičke operacije, te popratno zabilježene. Izvedena je i simulacija aritmetičko-logičke jedinice, te uspoređena sa realiziranim modelom 4 – bitne 74181 aritmetičko - logičke jedinice. Simulacija je izvedena pomoću Xilinx ISE 9.2i programa. Prilikom simulacije sama ALU je opisana pomoću VHDL koda.

## 2. Aritmetičke operacije

Aritmetika aritmetičko - logičke jedinice obuhvaća osnovne matematičke operacije na razini bita kao što su zbrajanje, oduzimanje, množenje i dijeljenje. Aritmetičke operacije sa binarnim brojevima slijede ista pravila i postupke aritmetike u dekadskom brojevnom sustavu. Budući da je za utvrđivanje predznaka rezultata potrebno usporediti predznake obadvaju operanda, što predstavlja dodatne operacije. Zato formati 'prvog komplementa' i 'drugog komplementa' predstavljaju prikladne načine prikaza i izvođenja osnovnih operacija. Računalni podatci su predstavljeni binarnim sustavom, odnosno brojevnim sustavom čija baza se sastoji od dva stanja koja se prikazuju sa dva logička stanja '0' i '1'. Ovisno o karakteristikama hardverskog dijela ograničeni smo na određen broj stanja (bitova) kojim se zbog beskrajne količine brojeva mogu prikazati samo određeni rasponi brojeva. Za prikaz pozitivnih i negativnih brojeva našlo se najjednostavnije rješenje, tako da pozitivne brojeve označava najznačajniji bit (prvi bit sa lijeve strane) ako se nalazi u stanju '0'. Sukladno tomu se određuju i negativni brojevi tako da se najznačajniji bit nalazi u stanju '1'. Sljedeći primjer prikazuje označavanje 4-bitnih pozitivnih i negativnih brojeva :

$$0101_2 = 5_{10}$$

$$1101_2 = -5_{10}$$

### 2.1 Operacija zbrajanja

Na određenu naredbu, ALU će zbrojiti dva operanda, odnosno podatka. Zbrajanje se vrši u binarnoj formi, pri tome se pribrajanje može izvršiti sa ili bez prijenosa iz prethodne operacije. U slučaju da želimo zbrojiti više operanada, učinit ćemo to na taj način da sumi dva operanda pribrojimo uzastopce nove operande. Zbrajanje u binarnom sustavu slično je zbrajanju u dekadskom, ali se prijenos u lijevo vrši kada zbroj dosegne 2. To npr. znači, da je zbroj 1+1 jednak 0 i prijenos jedinice u lijevu stranu. Pravila zbrajanja u binarnom sustavu:

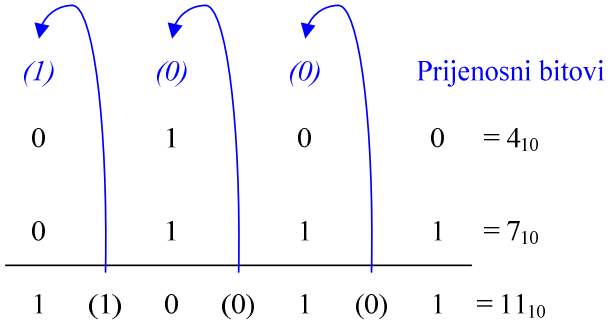
0+0=0

0+1=1

1+1=0 prijenos 1

1+1+1=1 prijenos 1

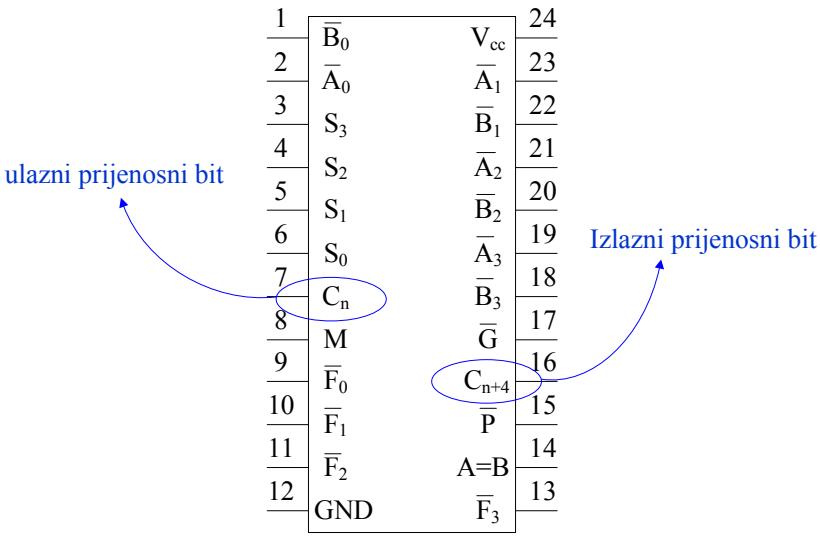
Primjer zbrajanja 2 4 - bitna broja, gdje je prvi broj 4 (u binarnom sustavu „0100“) i drugi broj 7 (u binarnom sustavu „0111“)




Slika 2.1. Primjer zbrajanja

Ograničenost hardvera na prikaz konačnog broja bitova dovodi do problema prikaza rješenja sume dvaju operanada u slučaju kada je rješenje veće od broja koje hardver može prikazati. Za rješenje tog problema uvodi se dodatni hardverski dio koji detektira i prenosi signal pod nazivom carry-out ili carry-in ovisno o smjeru ulaska u sklop.

Prikaz carry-in i carry-out signala na 74181 aritmetičko-logičkoj jedinici nalazi se na slici 2.2.

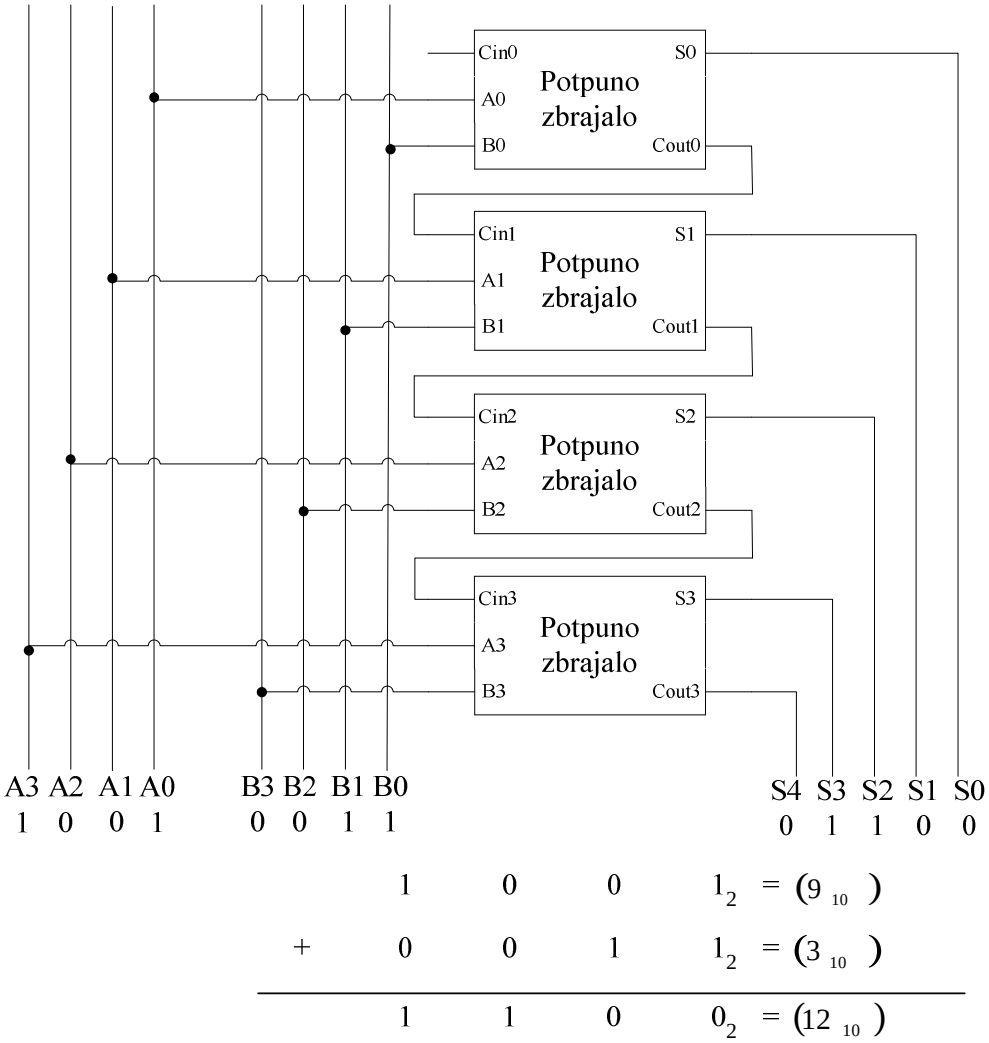



Slika 2.2. Prijenosni ulazni i izlazni pinovi

Carry signal predstavlja dodatni bit koji se dodaje sa lijeve strane izlaznog podatka (rješenja zbrajanja), npr. ako imamo 4-bitni sustav carry signal postaje peti bit, odnosno postaje najvažniji bit ili prvi bit sa lijeve strane. Do pojave preljevanja bitova (eng. Overflow) dolazi u slučaju kada se zbrajaju dva broja istog predznaka. Overflow se neće pojaviti u slučaju kada se zbrajaju dva broja različitog predznaka $x - y = x + (-y)$.

### 2.1.1 **Paralelno zbrajanje binarnih brojeva**

Ako su dva binarna broja istovremeno (paralelno) prisutna u digitalnom sustavu, mogu se zbrojiti kao na slici 2.3. Ulaz čine četverobitni brojevi *a* i *b*.

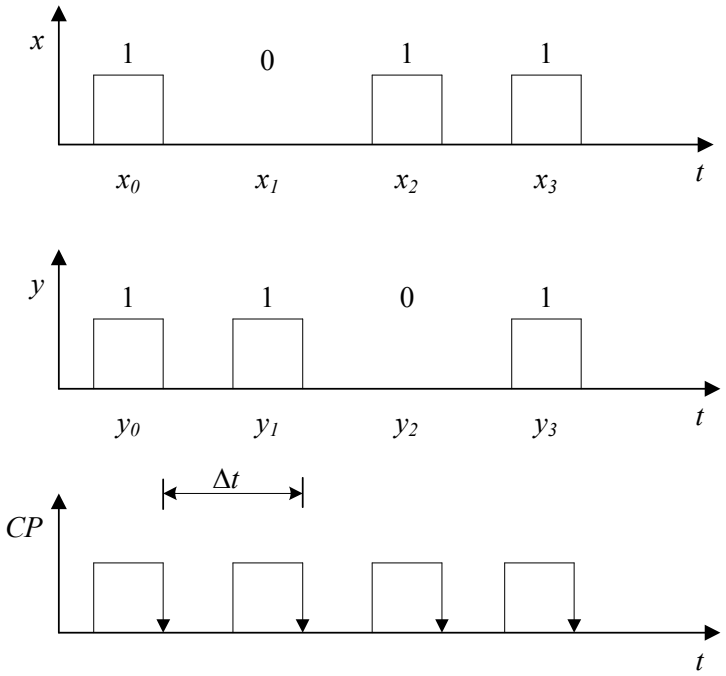



Slika 2.3. Paralelno zbrajanje binarnih brojeva

Najmanje značajni bitovi dovode se na ulaz poluzbrajala koje proizvodi sumu $S_0$ i prijenos $C_0$ na slijedeće brojno mjesto. Prijenos $C_0$ s prvog brojnog mjesta zbraja se sa druge dvije znamenke $a_1$ i $b_1$. Njihova suma je $S_1$, a prijenos $C_1$ zbraja se sa slijedeće dvije znamenke $a_2$ i $b_2$. Njihova suma je $S_2$, a prijenos $C_2$ se pribraja sa preostale dvije znamenke $a_3$ i $b_3$. Konačni rezultat je suma S = $S_0S_1S_2S_3$ i prijenos *P*.

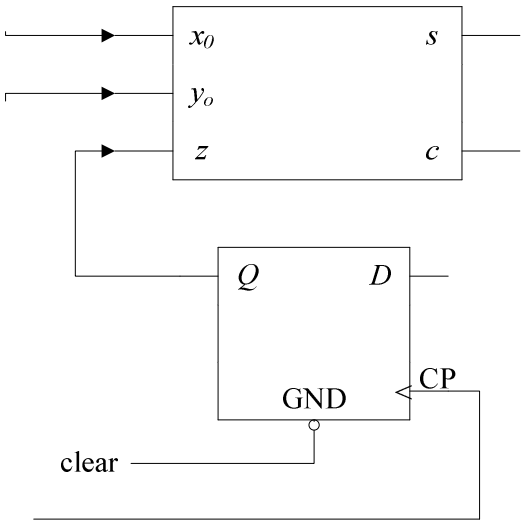
### 2.1.2 **Serijsko zbrajanje binarnih brojeva**

Ako su dva binarna broja predstavljena serijski, onda se zbrajanje može provesti sukcesivnim postupkom tako da se najprije zbroje prve dvije znamenke $x_0$ i $y_0$, a prijenos *C* zadrži dok ne stignu slijedeće dvije znamenke, a onda se zajedno s njima zbroji. Prijenos iz te operacije treba zadržati do slijedeće dvije znamenke itd. Sklop koji radi po takvom serijskom algoritmu prikazan je na slici 2.4. Prijenos *C* treba zadržati kroz vrijeme Δ*t*.




Slika 2.4. Kašnjenje između signala

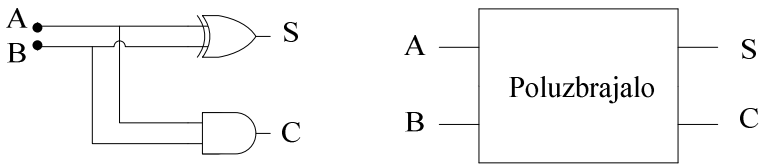
Sinkronizacijski impulsi osiguravaju da ulazni brojevi budu međusobno usklađeni pa je i kašnjenje najbolje tako izvesti da bude jednako kašnjenju između dva ulazna bita. To se postiže D-bistabilom koji će kroz vrijeme Δ*t* zadržavati informaciju koja je neposredno prije toga bila na ulazu, a to je iznos prijenosa *C*.




Slika 2.5. Serijsko zbrajanje binarnih brojeva

### 2.1.3 **Realizacija operacije zbrajanja pomoću kombinacijskih sklopova**

Za zbrajanje dva broja koristi se potpuno zbrajalo, koje se sastoji od dva poluzbrajala. Poluzbrajalo sadrži dva osnovna logička sklopa, sklop I (eng. AND) i sklop isključivo ILI (eng. XOR).

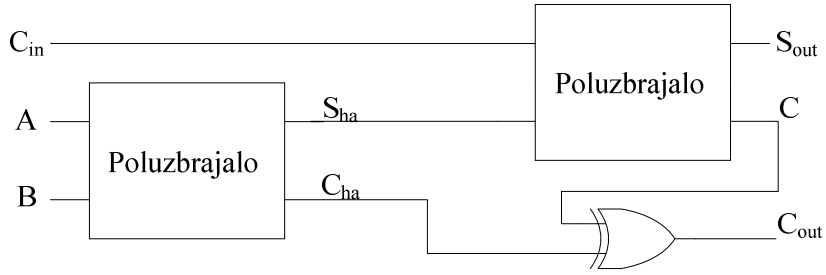



Slika 2.6. Poluzbrajalo prikazano preko osnovnih logičkih sklopova, te prikaz pomoću blok sheme

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Tablica 2.1. Tablica stanja (eng. Truth table).

Spajanjem dvaju poluzbrajala u jednu cjelinu, na taj način da izlazni pin iz prvog poluzbrajala koji označava sumu jednobitnih brojeva A i B se spoji sa ulaznim pinom drugoga poluzbrajala, te potom prijenosni bit koji se označava kao carry-in ($C_{in}$) spaja sa preostalim ulaznim pinom drugoga poluzbrajala. Da bi se ostvarilo potpuno zbrajalo potreban je i carry-out pin koji je označen kao $C_{out}$, a nastaje dodavanjem osnovnog logičkog sklopa ILI (eng. OR) na čije se ulazne pinove spajaju prijenosni bitovi obaju poluzbrajala i pritome izlazni pin postaje carry-out pin. Spajanjem dva pulozbrajala na prethodno opisan način realizira se potpuno zbrajalo, u ovome slučaju jednobitno potpuno zbrajalo koje je prikazano na slijedećoj slici




Slika 2.7. Potpuno zbrajalo ( eng. Full adder)

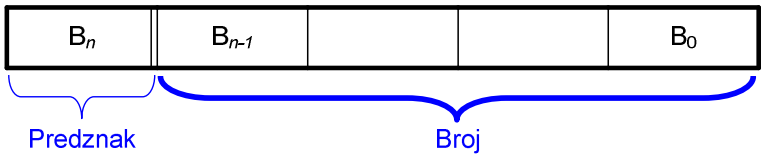
| A | B | $C_{ha}$ | $S_{ha}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| $C_{in}$ | $C_{ha}$ | $S_{ha}$ | $C_{out}$ | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | x | x |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | x | x |

Tablica 2.2. Tablica stanja potpunog zbrajala

## 2.2 Prikaz binarnih brojeva sa predznakom

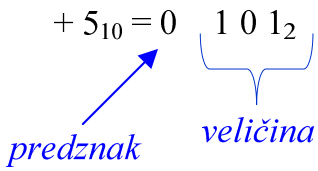
U prethodnom dijelu zbog operacije zbrajanja razmatrani su samo pozitivni brojevi, gdje se sa binarnim kodom prikazivala samo veličina. Zbog slijedeće razmatrane operacije potrebno je imati i mogućnost za prikaz predznaka broja, odnosno za prikaz cijelih brojeva. Mogućnost prikaza predznaka je ostvarena na taj način da je prvi bit, odnosno najljeviji bit rezerviran za određivanje predznaka. Najprikladnije je upotrijebiti „0“ za oznaku predznaka (+),a „1“ za oznaku predznaka (-). Takav način kodiranja, te prikazivanja brojeva je moguć jedino ako je poznata i stalna duljina binarne riječi (broja).




Slika 2.8. Prikaz broja sa predznakom

Pozitivni brojevi uvijek se pišu tako da se broj koji slijedi iza bita za predznak prikazuje neposredno kao veličina u binarnom sustavu.

Primjer prikaza broja 5 sa binarnom riječi od 4 bita:




Slika 2.9. Prikaz pozitivnog broja

Negativni brojevi prikazuju se na tri načina:

**1)** predznakom i veličinom
**2)** predznakom i 2. komplementom
**3)** predznakom i 1. komplementom

Prvi način je isti kao i kod prikaza pozitivnih brojeva. Dok je za aritmetičku operaciju oduzimanja pogodnije broj prikazati u obliku komplementa.

## 2.3 Komplementi brojeva

Komplementi brojeva definirani su općenito za bilo koji brojevni sustav. Definiraju se dva komplementa, jedan s obzirom na modul sustava, a drugi s obzirom na modul umanjen za jedinicu. Komplement se definira s obzirom na modul *m* u kojem je broj izražen, što znači s obzirom na broj cjelobrojnih brojnih mjesta *n* jer je modul $m = B^n$.

### 2.3.1 **B-ti komplement**

Komplement broja *N* s obzirom na bazu sustava *B* i modul $m = B^n$ je:

$$\overline{N_B} = B^n - N = m - N \qquad (2 - 1)$$

U dekadskom sustavu taj se komplement zove 10. komplement.

Primjer 10. komplementa :

za $n = 2$ : $(23)_{10} = 10^2 - 23 = 77$

za $n = 3$ : $(23)_{10} = 10^3 - 23 = 977$

Definicija vrijedi i za decimalne brojeve, pa je za $n = 2$ :

$(\overline{13{,}45})_{10} = 10^2 - 13{,}45 = 86{,}55$

U binarnom sustavu taj se komplement zove 2. komplement.

Primjer 2. komplementa :

za $n = 2$ : $(0101)_2 = 2^2 - 0101 = 10000 - 0101 = 1011$

Također vrijedi da je komplement komplementa nekog broja opet taj isti broj.

2.3.2 **(B - 1) – i komplement**

Za broj u modulu $m = 2^n$ i sa *d* decimalnih mjesta bit će (*B - 1*) –i komplement broja *N* :

$$\overline{N} = B^n - N - B^{-d} = \overline{N_B} - B^{-d} \qquad (2 - 2)$$

U dekadskom sustavu taj se komplement zove 9. komplement.

Ako se uzmu isti primjeri, vrijedi :

za *n = 2 :* $(23)_{10} = 10^2 - 23 - 10^0 = 76$

za *n = 3* : $(23)_{10} = 10^3 - 23\ 10^0 - 10^0 = 976$

za *n = 2* : $(13{,}45)_{10} = 10^2 - 13{,}45 - 10^{-2} = 86{,}54$

Vidljivo je da se uvijek najniža znamenka umanjuje za jedan. 9. komplement vrlo se jednostavno može dobiti bez računanja tako da se svaka znamenka odbije od 9, odnosno od baze umanjene za 1, pa je prema tome i dobio ime.

U binarnom sustavu se (*B - 1*)- *i* komplement zove 1. komplement.

Primjer 1. komplementa :

za *n = 4* : $(0101)_2 = 10000 - 0101 - 1 = 1011$

1. komplement se vrlo jednostavno dobije i bez računanja tako da se komplementira svaka znamenka. Ako se posebno ništa ne specificira, onda se pod nazivom komplement podrazumjeva (*B - 1*) – i komplement. *B* – ti komplement se obično najjednostavnije tvori tako da se najprije odredi (*B - 1*) – i komplement i onda mu se doda jedinica na najmanje značajno mjesto. Određivanje drugog komplementa se može provesti na jednostavniji i prikladniji način, koji umjesto prethodnog načina ima samo jedan korak za izvedbu.

Algoritam :

a) početi sa najmanje značajnom znamenkom i, ako je ona nula, ostaviti je nepromijenjenom kao i sve ostale nule koje slijede.
b) prvu jedinicu koja dolazi nakon niza nula također ostaviti nepromijenjenom.
c) komplementirati svaki slijedeći bit.

## 2.4 Operacija oduzimanja

Dva binarna broja se mogu neposredno oduzeti u skladu s pravilima oduzimanja u binarnom sustavu :

Minuend : 0 1 1 0

Suptrahend : - 0 - 0 - 1 - 1

Razlika : 0 1 0 1

Jedinica posuđena sa višeg stupnja : 1

Prve tri znamenke kao rezultat daju jedan bit, dok kod zadnje znamenke ( kada je umanjitelj veći od umanjenika) ne može se izvršiti oduzimanje ako se ne posudi znamenka iz višeg stupnja. S višeg stupnja uzima se jedna jedinica koja na mjestu oduzimanja vrijedi duplo. Ako se od posuđene jedinice oduzme 1 ostatak je 1.

Drugi način oduzimanja je oduzimanje pomoću komplementa, odnosno oduzimanje pomoću komplementa je moguće izvesti u bilo kojem brojevnom sustavu sa bazom *B*. Kod reprezentacije broja u modulu $m=B^n$, broj $B^n$ je prikazan sa 0. Pa prema tome najveći broj koji se može prikazati sa *n* brojnih mjesta bit će :

$$W=B^n-1 \qquad (2-3)$$

U dekadskom je sustavu W = 9, a u binarnom W = 1

Dodavanje modula $B^n$ neće promjeniti broj prikazan u tom istom modulu, odnosno ako se npr. brojilu u kojem je taj broj zapisan doda još $B^n$ impulsa, brojilo će napraviti puni ciklus i opet doći u isto stanje, pa vrijedi :

$$A + B^n = A \qquad (2-4)$$

Postoje dva algoritma, jedan sa *B* – tim i drugi sa *(B - 1)* – im komplementom.

### 2.4.1 **Oduzimanje s (*B – 1)* – tim komplementom**

Postoje dva moguća slučaja kod oduzimanja s komplementima. Prvi slučaj kada je minuend veći od suptrahenda, i drugi kada je minuend manji od suptrahenda.

Prvi slučaj (M > S) :

Razlika M – S neće se promijeniti ako joj se doda $W + 1 = B^n$ :

$$M - S = M - S + B^n = M - S + W + 1 = M + (W - S) + 1 = M + \overline{S} + 1 \qquad (2 - 5)$$

Ako se minuendu doda komplement suptrahenda, slijedi :

$$M + \overline{S} = M + W - S = W + (M - S) \qquad (2 - 6)$$

Rezultat je broj koji je jednak ili veći od $B^n$. Budući da je taj zbroj veći od modula pojavit će se 1 na većem brojnom mjestu. Doći će do pojave preljeva bita.

Drugi slučaj (M < S) :

Razlika između $M - S$ neće se promijeniti ako joj se doda $W - W = 0$ :

$$M - S = M - S + W - W = M + (W - S) - W = -[W - (M + \overline{S})] = -\overline{(M + \overline{S})} \qquad (2 - 7)$$

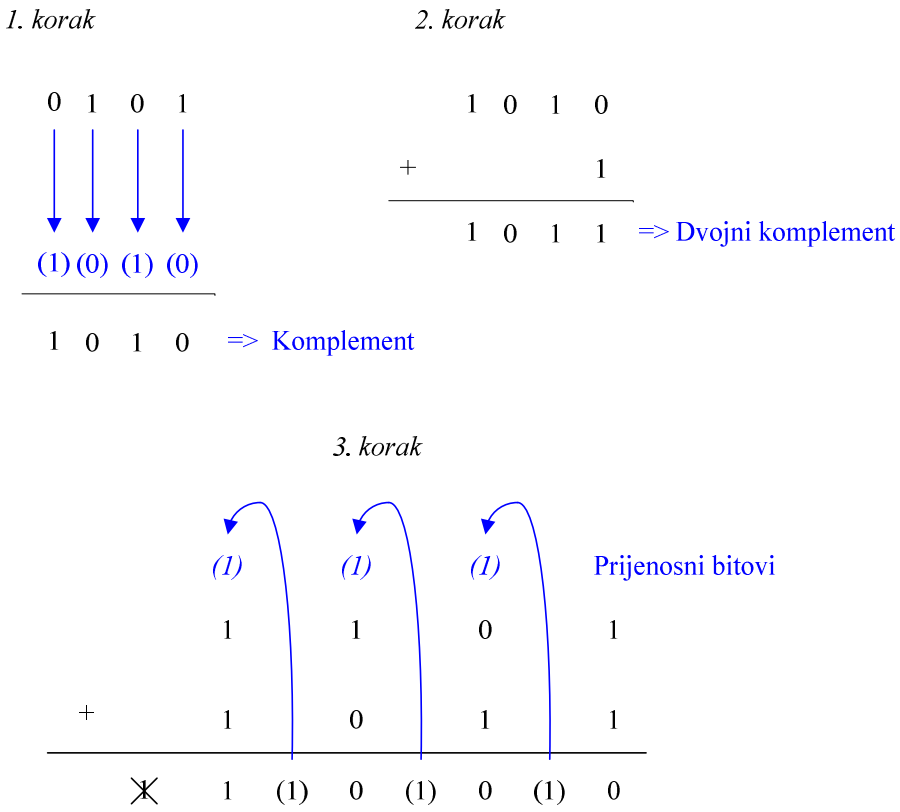
U ovom slučaju nema preljeva pri zbrajanju $M + S$, jer je rezultat negativan, pa je $M + S = W + (M - S)$ manje od $W$. Oba se slučaja mogu sumirati u cjelovit algoritam odbijanja :

a) pribrojiti minuendu komplement suptrahenda
b) ako se pojavi preljev, dodati 1
c) ako nema preljeva, komplementirati još jednom i promjeniti predznak

Primjena prethodno opisanog algoritma na modulu sa četiri znamenke. Primjer oduzimanja dva binarna broja čija je razlika pozitivan broj:

$1101_2 = 13_{10}$

$0101_2 = 5_{10}$




***Provjera:***

$$13_{10} = 1101_2$$

$$- \quad 5_{10} = 0101_2$$

$$8_{10} = 1000_2$$

Prilikom oduzimanja dva binarna broja čiji rezultat je pozitivan binarni broj, dolazi do pojave jedinice na najvažnijem binarnom mjestu (eng. Moost significant bit - MSB) u rezultatu. Ta binarna jedinica se zanemaruje kao što je prikazano u prethodnom primjeru.

### 2.4.2 **Oduzimanje s *B* – tim komplementom**

Postoje dva ista slučaja kao i kod oduzimanja sa *(B - 1)* komplementom za koja vrijedi :

$$\text{Prvi slučaj } (M > S): \quad M - S = M + \overline{S}_B \qquad (2\text{-}8)$$

$$\text{Drugi slučaj } (M < S): \quad M - S = -\,\overline{(M + \overline{S}_B)}_B \qquad (2\text{-}9)$$

U prvom zbroju pojavljuje se preljev, dok ga u drugom nema. Oba slučaja se mogu sumirati u jedan algoritam :

a) pribrojiti minuendu *B*-ti komplement suptrahenda
b) ako se pojavi preljev, to je rezultat
c) ako nema preljeva treba $M + S_B$ još jednom komplementirati s obzirom na bazu i promjeniti predznak

Primjena prethodno opisanog algoritma na modulu sa četiri znamenke. Primjer oduzimanja dva binarna broja čija je razlika pozitivan broj:

$1101_2 = 13_{10}$

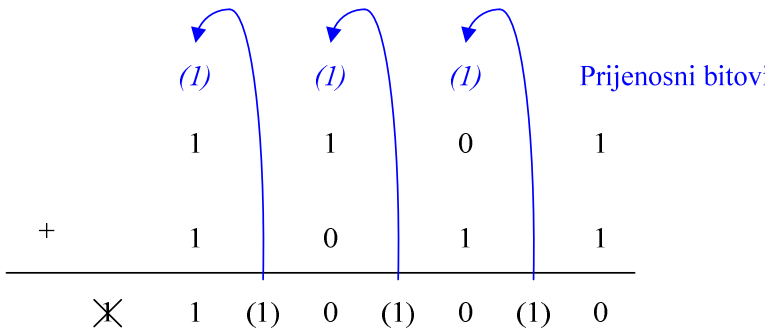
$0101_2 = 5_{10}$

*1. korak*

$\overline{S} = 4^2 - 0101 = 10000 - 0101 = 1011$

B – ti komplement

*2. korak*




***Provjera:***

$$13_{10} = 1101_2$$

$$- \quad 5_{10} = 0101_2$$

$$8_{10} = 1000_2$$

## 2.5 Množenje

Za izvođenje operacije množenja potrebno je poznavati tablicu množenja.

$0 \cdot 0 = 0$

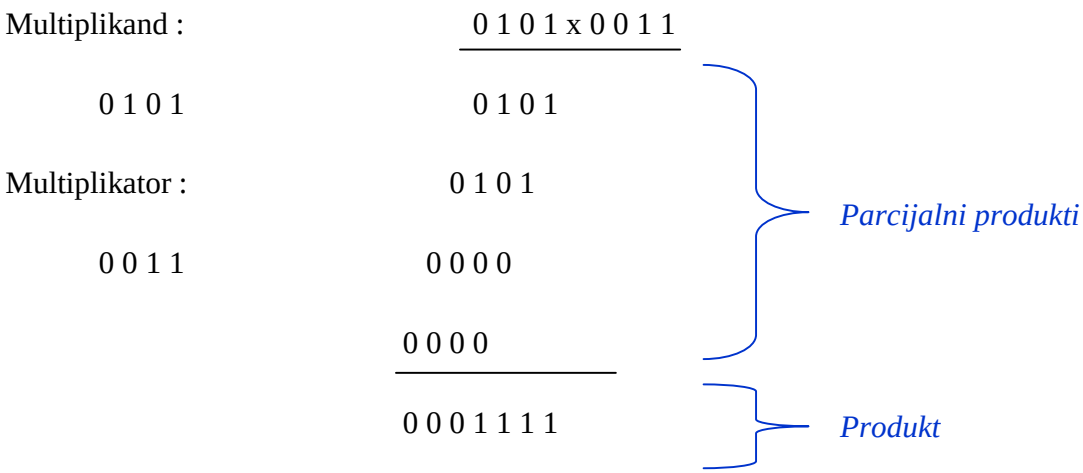
$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

Tablica 2.3. Tablica množenja

Prema stanjima iz tablice preijenosa nema, a množenje se može realizirati običnom logičkom I – funkcijom. Realizacija množenja u digitalnom sustavu počinje od prikazanog primjera.

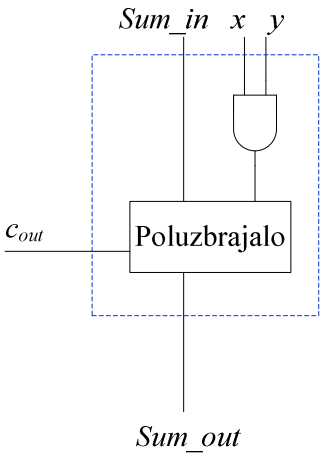



Množenje u svim sustavima sastoji se od dobivanja parcijalnih produkata i njihovom sumiranju uz posmak. Parcijalni produkti se dobiju množenjem multiplikanda sa pojedinim bitom multiplikatora počevši sa bitom na najmanje važnom binarnom mjestu. Broj parcijalnih produkata jednak je broju bitova multiplikatora. Sumiranjem parcijalnih produkata uz uvijet da je svaki parcijalni produkt pomaknut za jedan bit u lijevo dolazi se do produkta, te sa prethodno opisanom metodom množenja dolazi se do prikazanog algoritma.

| | | | | $a_3$ | $a_2$ | $a_1$ | $a_0$ | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | X |
| | | | | $b_3$ | $b_2$ | $b_1$ | $b_0$ | |
| parcijalni produkt 0 | | | | $a_3b_0$ | $a_2b_0$ | $a_1b_0$ | $a_0b_0$ | |
| parcijalni produkt 1 | | | $a_3b_1$ | $a_2b_1$ | $a_1b_1$ | $a_0b_1$ | | |
| parcijalni produkt 2 | | $a_3b_2$ | $a_2b_2$ | $a_1b_2$ | $a_0b_2$ | | | |
| parcijalni produkt 3 | $a_3b_3$ | $a_2b_3$ | $a_1b_3$ | $a_0b_3$ | | | | |
| | $y_0 = a_3b_3$ | $y_0 = a_3b_2 + a_2b_3$ | $y_0 = a_3b_1 + a_2b_2 + a_1b_3$ | $y_0 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3$ | $y_0 = a_2b_0 + a_1b_1 + a_0b_2$ | $y_0 = a_1b_0 + a_0b_1$ | $y_0 = a_0b_0$ | |

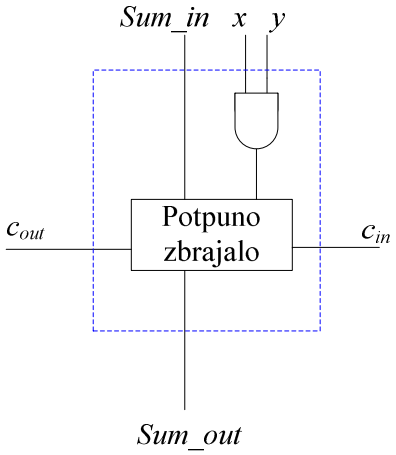
Slika 2.10. Algoritam za množenje

Kombinacijskim sklopovima može se napraviti sklop za izvođenje algoritma. Dodavanjem I – sklopa poluzbrajalu dobiva se jedan dio sklopa za množenje (poluzbrajalo množitelja), pod skraćenim nazivom PM modul.



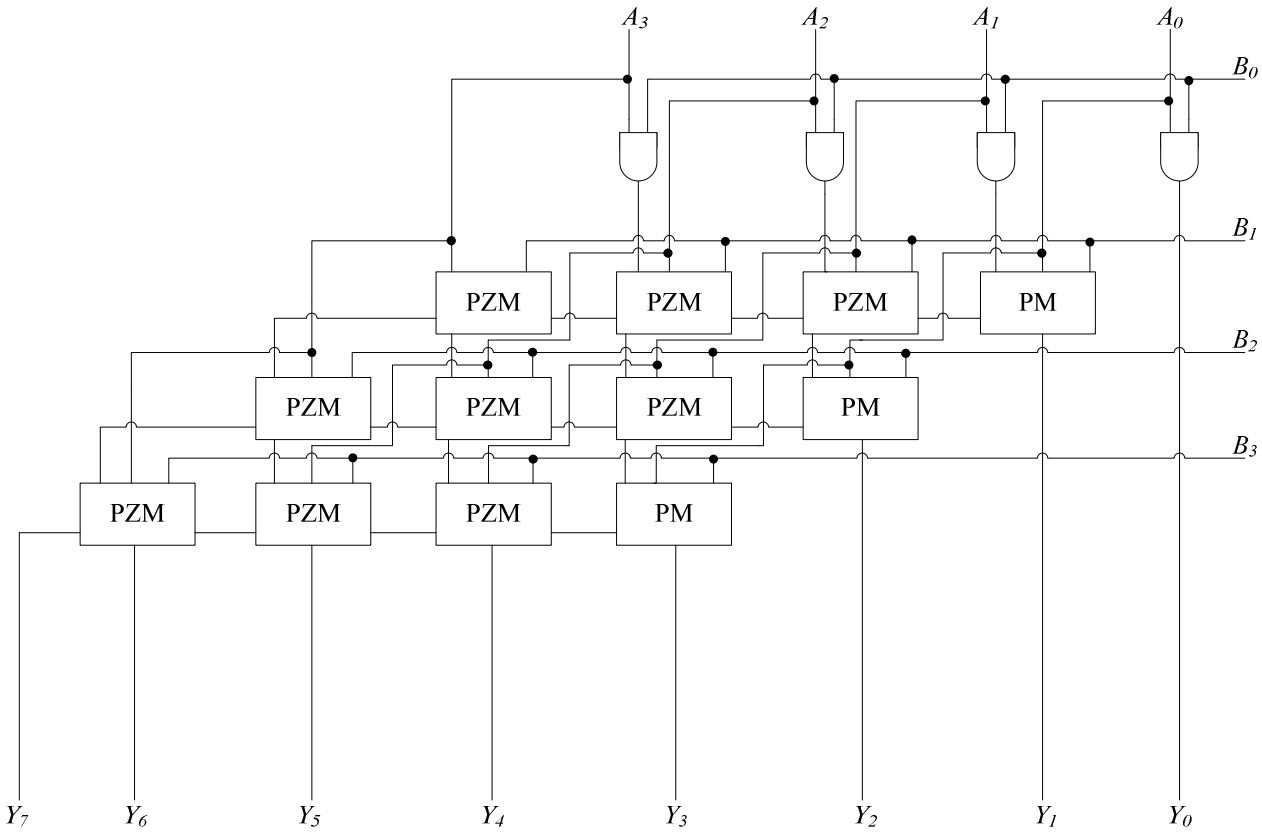

Slika 2.11. Poluzbrajalo u PM modulu

Osim I – sklopa za realizaciju sklopa za množenje potreban je i prilagođeni sklop za potpuno zbrajanje (potpuno zbrajalo množitelja), pod skraćenim nazivom PZM modul.




Slika 2.12. Potpuno zbrajalo u PZM modulu

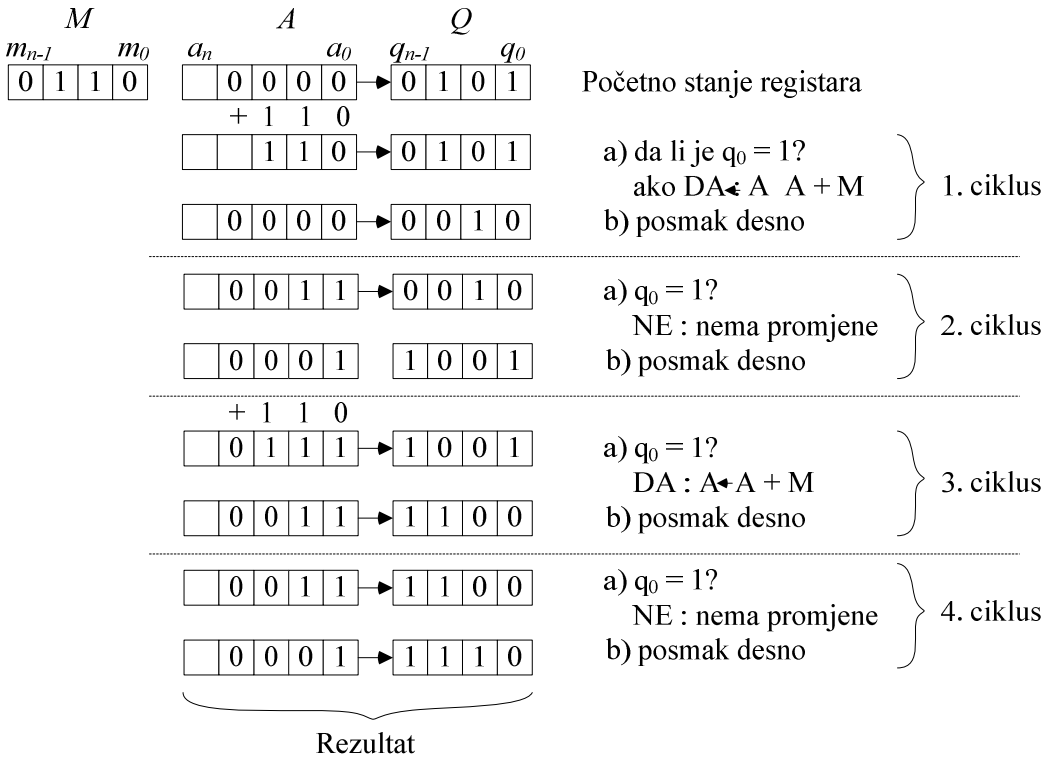
Sastavljanje PM modula, PZM modula i I – sklopa rezultira blok shemom množila.




Slika 2.13. Blok shema množila

Prvi parcijalni produkt rezultat je množenja bita $A_0$ sa 4-bitnim vektorom $B_0 - B_3$. Bit najmanje težine direktno se zapisuje u rezultat kao $Y_0$. Dok je svaki sljedeći bit, osim prvog bita ($Y_7$) koji je prijenos prethodne sumacije, rezultat sume bitova parcijalnih produkata.

Sklop množenja realiziran kombinacijskim sklopovima riješava problem, no sklop je u većini slučajeva vrlo velik i služi samo jednoj svrsi pa je većinu vremena neiskorišten. Zato se takav sklop realizira pomoću tri registra, sklopa za zbrajanje i sklopa za kontrolu čitave operacije. U registar M upiše se multiplikand, u registar Q multiplikator i u posljednjem registru A (akumulator) akumulira se rezultat. Registri M i Q su jednaki svaki po *n* bitova, dok je registar A veći za jedan bit od registara M i Q zbog mogućeg preljeva.




Slika 2.14. Sekvencijalni način množenja

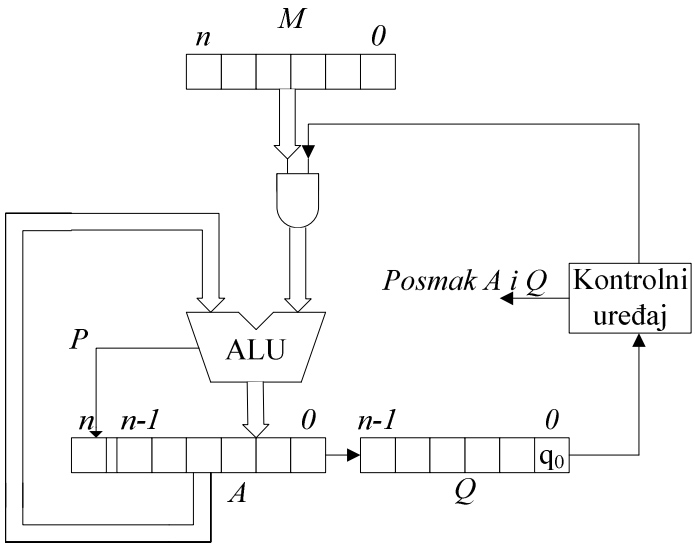
Izvedba množenja :

1. Ispituje se da li je najmanje značajni bit multiplikatora jednak 1, tj. da li je $q_0$ = 1. Ako je $q_0$ = 1, treba pomnožiti multiplikand s 1 i pribrojiti ga sadržaju akumulatora. Budući da množenje sa 1 ne mjenja sadržaj multiplikanda, operacija se svodi na :

$$A = A + M \qquad (2 - 10)$$

2. Pomiću se sadržaji registra A i Q za jedan korak udesno. Ta dva registra trebaju biti posmačni registri, međusobno povezani.
3. Ponavlja se 1. i 2. korak *n* puta, odnosno broj ponavljanja jednak je broju bitova Q registra.

Konačni se rezultat nalazi u registrima Q i A, koji zajedno predstavljaju 8 bitni rezultat operacije množenja.




Slika 2.15. *n* – bitno množilo

Organizacija digitalnog sustava za sekvencijalno množenje dvaju binarnih brojeva na opisan način prikazan je na slici sl…

### 2.5.1 **Booth – ov algoritam za množenje**

Booth - ov algoritam uključuje uzastopno zbrajanje jedne ili druge unaprijed određene vrijednosti A ili S produktu P, te nakon toga se vrši pomak produkta udesno. Označavajući multiplikator i multiplikand slovima *m* i *r*, a broj bitova unutar multiplikatora i multiplikanda *x* i *y*, postupak množenja se provodi u nekoliko slijedećih koraka :

1. Određivanje vrijednosti A i B, te početnu vrijednost P. Sve vrijednosti trebaju imati veličinu binarnog broja x + y + 1.
    1. A : Popuniti najvažnije bitove sa vrijednosti *m*, te preostale bitove sa binarnom nulom.
    2. S : Popuniti najvažnije bitove sa drugim komplementom vrijednosti (- *m*)
    3. P : Popuniti najvažnije bitove x sa binarnom nulom, y bitove sa vrijednosti *r* i ostatak binarnih mjesta popuniti sa binarnom nulom.
2. Utvrđivanje zadnja dva bita produkta
    1. Ako su zadnja dva bita “01”, produktu se pridodaje vrijednost A zanemarujući preljev.
    2. Ako su zadnja dva bita “10”, produktu se pridodaje vrijednost B zanemarujući preljev.

3. Ako su zadnja dva bita “00”, produktu ostaje ne promijenjen.
4. Ako su zadnja dva bita “11”, produktu ostaje ne promijenjen.

3. Aritmetički pomak udesno za jedan bit vrijednosti utvrđene u drugom koraku, te izjednačavanje produkta sa novom vrijednosti.
4. Ponavljanje drugog i trećeg koraka dok se ne ponove *y* puta.
5. Odbacivanje bita na najmanje važnom binarnom mjestu produkta.

Primjer množenja broja 5 i -3 prema prethodno napisanim koracima:

m = 5, r = - 3, x = 4 i y = 4

- A = 0101 0000 0
- B = 1011 0000 0
- P = 0000 1101 0
- Postupak provjere i dodavanja se provodi četiri puta :
    - P = 0000 1101 0. Zadnja dva bita u produktu su “10”
        - P = 0000 1101 0. P = P + B
        - P = 1011 1101 0.
        - Aritmetički pomak udesno
    - P = 1101 1110 1. Zadnja dva bita u produktu su “01”
        - P = 1101 1110 1. P = P + A
        - P = 0010 1110 1
        - Aritmetički pomak udesno
    - P = 0001 0111 0. Zadnja dva bita u produktu su “10”
        - P = 0001 0111 0. P = P + B
        - P = 1100 0111 0
        - Aritmetički pomak udesno
    - P = 1110 0011 1. Zadnja dva bita u produktu su “11”
        - P = 1111 0001 1
        - Aritmetički pomak udesno
- Zanemarujući zadnju binarnu znamenku dolazimo do rješenja operacije, P = 1111 0001. Gdje je binarni zapis produkta ekvivalentan dekadskom broju – 15, koji odgovara rješenju operacije.

Prethodno opisana metoda množenja ima nedostatak prilikom množenja sa maksimalnim negativnim brojem, gdje je kod četiri binarnog sustava maksimalni negativan broj – 8. Ispravak ovog nedostatka ostvaruje se dodavanjem jednog binarnog mjesta sa lijeve strane vrijednosti A. Množenje maksimalnog negativnog broja - 8 sa brojem 3 pomoću ispravljene metode.

m = - 8, r = 3, x = 4, y = 4

- A = 1 1000 0000 0
- B = 0 1000 0000 0
- P = 0 0000 0011 0

- Postupak provjere i dodavanja se provodi četiri puta :
  - P =0 0000 0011 0. Zadnja dva bita u produktu su “10”
    - P = 0 0000 0011 0. P = P + B
    - P = 0 1000 0011 0.
    - Aritmetički pomak udesno
  - P = 0 0100 0001 1. Zadnja dva bita u produktu su “11”
    - P = 0 0100 0001 1
    - Aritmetički pomak udesno
  - P = 0 0010 0000 1. Zadnja dva bita u produktu su “01”
    - P = 0001 0111 0. P = P + A
    - P = 1 1010 0000 1
    - Aritmetički pomak udesno
  - P = 1 1101 0000 0. Zadnja dva bita u produktu su “00”
    - P = 1 1101 0000 0
    - Aritmetički pomak udesno
- Rješenje množenja nakon zanemarivanja prve i zadnje binarne znamenke je “1110 1000”. Pretvoreno u dekadski sustav dobiva se rezultat – 24.

Uzevši za multiplikator pozitivan binarni broj, na primjer “0111” do rješenja, odnosno do produkta se dolazi preko slijedeće jednadžbe:

$$m \times \text{“0 1 1 1”} = m \times (2^2 + 2^1 + 2^0) = m \times 7 \qquad \textit{(2 - 11)}$$

Gdje je u jednadžbi multiplicand predstavljen slovom *m*. Zapisujući multiplikator na drugačiji način operacija se svodi na dva koraka, na čemu se zapravo zasniva ideja Booth-ovog algoritma.

$$\mathrm{m} \times \text{“}1\ 0\ 0\ 0 - 0\ 0\ 0\ 1\text{”} = \mathrm{m} \times (2^3 - 2^0) = \mathrm{m} \times 7 \qquad \textit{(2 - 12)}$$

Odnosno, bilo koji binarni niz jedinica može se prikazati kao razlika dva binarna broja.

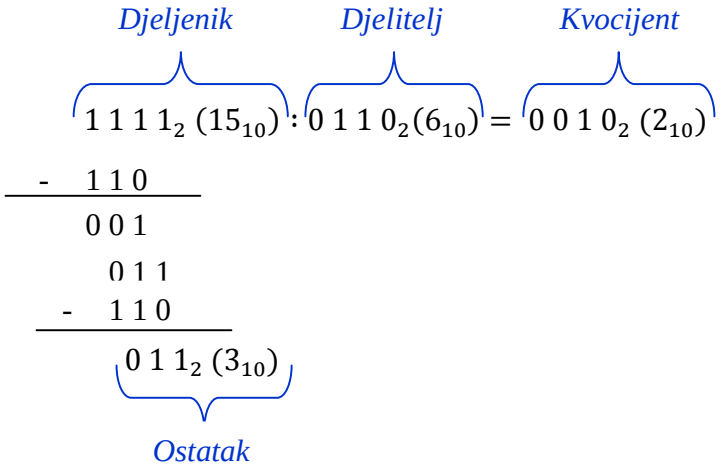
$$(\ldots 0\ \overbrace{1\ldots1}^{n}\ 0\ \ldots)_2 \equiv (\ldots 1\ \overbrace{0\ldots0}^{n}\ 0\ \ldots)_2 - (\ldots 0\ \overbrace{0\ldots1}^{n}\ 0\ \ldots)_2 \qquad \textit{(2 - 13)}$$

Na osnovu ove ideje množenje se može zamijeniti osnovnim operacijama, zbrajanjem multiplikatora produktu, aritmetičkom pomaku udesno parcijalnog produkta i na kraju oduzimanje multiplikatora od produkta. Booth-ov algoritam se može primjeniti na *n* – bitne sustave, a prilikom izvedbe algoritam koristi manji broj operacije zbrajanja i oduzimanja od uobičajenog načina množenja.

## 2.6 Dijeljenje

Recipročna operacija množenja je dijeljenje, rijeđe korištena operacija od ostalih i nudi mogućnost pojavljivanja matematički nedopuštenih operacija poput dijeljenja sa nulom. Sama operacija se u binarnom sustavu provodi na isti način kao u dekadskom.

Primjer dijeljenja u binarnom sustavu :




Slika 2.16. Dijeljenje dva 4- bitna broja

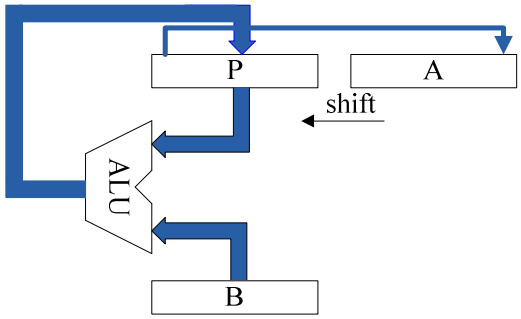
Zbog ograničenja binarnog sustava na samo dvije znamenke dijeljenje je jednostavnije nego u dekadskom sustavu.

Osnovni algoritam dijeljenja pokušava izračunati koliko velik broj se može oduzeti od djeljenika, povećavajući kvocijent svakim pokušajem za 1 krenuvši od nule. U primjeru dijeljenja gleda se koliko puta djeljitelj ide u dio djeljenika, zanemarujući sve nule sa lijeve strane do pojave prve jedinice. Prvih *n* – bitova djeljenika oduzima se sa djeliteljom, ako je broj od *n* – bitova djeljenika veći ili jednak djelitelju rezultat je 1, u suprotnom 0. Nakon svake operacije oduzimanja spušta se slijedeća znamenka djeljenika, gdje se sa spuštanjem zadnje znamenke i provođenjem prethodne operacije može pojaviti broj manji od djeljitelja, koji predstavlja ostatak dijeljenja.

Dijeljenje se može implementirati na sličan način kao i množenje. Zbog mogućnosti pojave ostatka, a i inače, dijeljenje je općenito kompliciranije od množenja i u mnogima, osobito manjim računalima nema posebnog sklopa za dijeljenje, već se ono obavlja pomoću programa.

### 2.6.1 **Restoring i non – restoring algoritam za dijeljenje**

Navedeni algoritmi spadaju u klasu sporih algoritama za izvedbu dijeljenja, a razlika između njih je da jedan vrši operacije samo nad pozitivnim brojevima dok drugi vrši operacije i nad pozitivnim i nad negativnim brojevima bez ikakve korekcije. Za izvedbu restoring algoritma potrebna su tri registra i jedna ALU jedinica (prikazano na slici 2.19), gdje se u jedan registar (registar A) stavlja vrijednost djeljenika, u drugi registar (registar B) se stavlja vrijednost dijelitelja, te u zadnji registar (registar P) nulta vrijednost koji na kraju sadržava ostatak dijeljenja.




Slika 2.17. Hardverska blok shema za izvedbu restoring algoritma

Sprovedba algoritma svodi se na četiri koraka :

a) Pomak vrijednosti registarskog para P i A za jedan bit ulijevo
b) Oduzimanje B od P, te vraćanje rezultata u P registar
c) Ako je rezultat negativan, postavi bit na najnižem mjestu registra A u nulu, a obratno u binarnu jedinicu
d) Ako je rezultat negativan vrati prijašnju vrijednost registra P dodavanjem vrijednosti iz registra B

Primjer dijeljenja dva 4 – bitna broja, $1110_2$ ($14_{10}$) sa $0011_2$ ($3_{10}$), koristeći restoring algoritam prikazan je na slici 2.18.

| P | A | |
|---|---|---|
| 0 0 0 0 0 | 1 1 1 0 | |
| 0 0 0 0 1 | 1 1 0 | 1. korak (a) : posmak |
| - 0 0 0 1 1 | | 1. korak (b) : oduzimanje |
| - 0 0 0 1 0 | 1 1 0 0 | 1. korak (c) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| 0 0 0 0 1 | 1 1 0 0 | 1. korak (d) : vračanje vrijednosti |
| 0 0 0 1 1 | 1 0 0 | 2. korak (a) : posmak |
| - 0 0 0 1 1 | | 2. korak (b) : oduzimanje |
| 0 0 0 0 0 | 1 0 0 1 | 2. korak (c) : rezultat je pozitivan, postavi zadnji bit registra A u jedinicu |
| 0 0 0 0 1 | 0 0 1 | 3. korak (a) : posmak |
| - 0 0 0 1 1 | | 3. korak (b) : oduzimanje |
| - 0 0 0 1 0 | 0 0 1 0 | 3. korak (c) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| 0 0 0 0 1 | 0 0 1 0 | 3. korak (d) : vračanje vrijednosti |
| 0 0 0 1 0 | 0 1 0 | 4. korak (a) : posmak |
| - 0 0 0 1 1 | | 4. korak (b) : oduzimanje |
| - 0 0 0 0 1 | 0 1 0 0 | 4. korak (c) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| 0 0 0 1 0 | 0 1 0 0 | 4. korak (d) : vračanje vrijednosti |

Slika 2.18 Primjer dijeljenja sa restoring algoritmom

Rezultat dijeljenja, odnosno kvocijent se nalazi u registru A, a ostatak unutar registra P.

Non – restoring algoritam za razliku od restoring algoritma jedino preskače zadnji korak, zapravo ne ispravlja negativnu vrijednost. Prema tome se sastoji samo od tri koraka :

- Ako je P negativan
  - 1.a) Pomak registarskog para P i A za jedan bit ulijevo
  - 2.a) Zbrajanje registara B i P
- Ako je P pozitivan
  - 1.b) Pomak registarskog para P i A za jedan bit ulijevo
  - 2.b) Oduzimanje registra B od registra P
- Ako je P negativan postavi najmanje važan bit registra A u nulu, u suprotnome postavi u binarnu jedinicu

Primjer dijeljenja dva 4 – bitna broja, $1110_2$ ($14_{10}$) sa $0011_2$ ($3_{10}$), koristeći non - restoring algoritam nalazi se na slici 2.19.

| P | A | |
|---|---|---|
| 0 0 0 0 0 | 1 1 1 0 | |
| 0 0 0 0 1 | 1 1 0 | 1. korak (1.b) : posmak |
| +0 0 0 1 1 | | 1. korak (2.b) : oduzimanje B registra |
| 1 1 1 1 0 | 1 1 0 0 | 1. korak (3) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| 1 1 1 0 1 | 1 0 0 | 2. korak (1.a) : posmak |
| +0 0 0 1 1 | | 2. korak (2.a) : dodavanje registra B |
| 0 0 0 0 0 | 1 0 0 1 | 2. korak (3) : rezultat je pozitivan, postavi zadnji bit registra A u jedinicu |
| 0 0 0 0 1 | 0 0 1 | 3. korak (1.b) : posmak |
| +1 1 1 0 1 | | 3. korak (2.b) : oduzimanje B registra |
| 1 1 1 1 0 | 0 0 1 0 | 3. korak (3) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| 1 1 1 0 0 | 0 1 0 | 4. korak (1.a) : posmak |
| +0 0 0 1 1 | | 4. korak (2.a) : dodavanje registra B |
| 1 1 1 1 1 | 0 1 0 0 | 4. korak (3) : rezultat je negativan, postavi zadnji bit registra A u nulu |
| +0 0 0 1 1 | | |
| 0 0 0 1 0 | | |

Slika 2.19. Primjer dijeljenja sa non - restoring algoritmom

Pojavom negativnog broja u posljednjem koraku vraća se vrijednost registra B u registar P koji pri završetku operacije sadržava ostatak dijeljnja, dok se kvocijent nalazi u registru A.

# 3. Logičke operacije

Sama sposobnost digitalnih sklopova za obavljanje računskih operacija i obradu podatka zasniva se na njihovoj sposobnosti obavljanja jednostavnih logičkih operacija. Jednostavna odnosno osnovna operacija je provjera da li je neka tvrdnja točna, odnosno istinita ili ispravna, ili pak netočna, odnosno neistinita. Od jednostavnih operacija može se različitim povezivanjem stvoriti složena logička operacija. U logičkoj se algebri ne razmatra konkretan sadržaj pojedine tvrdnje, već se promatra jedino da li je ta tvrdnja točna ili netočna. Ako čitavu tvrdnju označimo nekim slovom A ili brojem 1, onda 1 predstavlja točnu tvrdnju, pa preostala netočna tvrdnja se može predstaviti sa slovom B ili brojem 0. A i B su takozvane logičke varijable koje su međusobno nezavisne, a kao njihov rezultat označava se zavisna varijabla $f$ . Operacije nad binarnom logikom definirane i objašnjene su pomoću Booleove algebre, pomoću koje se dolazi do osnovnih logičkih sklopova :

- NE
- I
- ILI

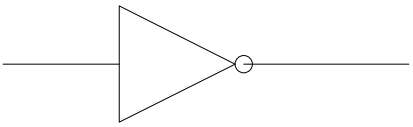
## 3.1 Logički sklop NE

Inverter ili logički sklop NE predstavlja logičku negaciju. Na izlazu iz invertera nalazi se napon koji predstavlja suprotnu binarnu vrijednost od ulazne vrijednosti. Zbog svoje jednostavnosti prilikom izrade postoje razne verzije i korištene tehnike izrade samog sklopa. Sastoji se od jednog ulaza i jednog izlaza, gdje je prema Boolean – ovoj algebri izlaz zavisan o ulazu prema slijedećem izrazu :

$$f = \overline{A} \qquad (3 - 1)$$

Simbol logičkog sklopa i tablica stanja NE sklopa nalazi se na slici 3.1.

a)



b)

| *f* | *A* |
|---|---|
| *0* | *1* |
| *1* | *0* |

Slika 3.1. Simbol NE sklopa pod a) i tablica točnosti pod b)

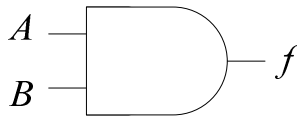
## 3.2 Logički sklop I

Logička funkcija $f$ koja na točno određen način povezuje logičke varijable A i B je točna (tj. u stanju 1) samo ako su jedna i druga varijabla točne (u stanju 1), a naziva se **I – funkcija.** Logička funkcija I dobiva se onda ako se na varijable A i B primjeni Booleov operator I. U matematičkoj logici taj se operator označava sa ∧. U digitalnoj elektronici i računarstvu općenito mnogo je češće u upotrebi točka kao uobičajeni znak za množenje jer se ta funkcija zove još i logički produkt. Upotrebljava se još i znak &, pogotovo u shemama.

$$f = A \wedge B = A \,\&\, B = A \cdot B = A\,B \qquad (3 - 2)$$

Logičke funkcije se još prikazuju i grafički, pa se u tu svrhu za osnovne funkcije upotrebljavaju standardni simboli. Simbol s karakterističnim oblikom odabranim da predstavlja samo ovu logičku funkciju prikazan je na slici 3.2.

a)




b)

| *A* | *B* | *f* |
|---|---|---|
| *0* | *0* | *0* |
| *0* | *1* | *0* |
| *1* | *0* | *0* |
| *1* | *1* | *1* |

Slika 3.2. Simbol I sklopa pod a) i tablica točnosti pod b)

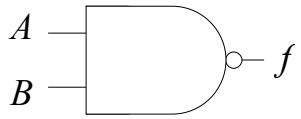
Tablica točnosti prikazuje sve moguće kombinacije logičkih varijabli i njima pripadnih vrijednost logičke funkcije *f*. Ta se tablica zato zove tablica logičkih kombinacija ili skraćeno tablica kombinacija.

Sklop izveden iz logičkog I – sklopa koji se dobiva kombinacijom sa NE sklopom, a ubraja se u osnovne logičke sklopove je NI sklop. Logička funkcija *f* NI sklopa zavisna o logičkim varijablama A i B dobiva se spajanjem funkcija I i NE sklopa, odnosno komplementiranjem funkcije I sklopa. Mogući načini prikaza funkcije NI sklopa :

$$f = \overline{A \wedge B} = \overline{A \,\&\, B} = \overline{A \cdot B} = \overline{A\,B} \qquad (3\text{-}3)$$

Svaka logička funkcija ima odgovarajući grafički simbol koji označava točno određenu funkciju, a simbol NI sklopa uz tablicu prikazan je na slici 3.3.

a)




b)

| *A* | *B* | *f* |
|---|---|---|
| *0* | *0* | *1* |
| *0* | *1* | *1* |
| *1* | *0* | *1* |
| *1* | *1* | *0* |

Slika 3.3. Grafički simbol (a) i tablica točnosti (b) NI sklopa

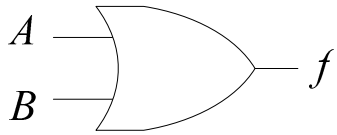
## 3.3 Logički sklop ILI

Složena funkcija *f* je točna ako je jedna ili druga varijabla točna. Može se iskazati na način da je funkcija *f* u stanju 1 ako su varijable A ili B jednake 1, ili ako su obje varijable jednake 1. Tako opisana funkcija se naziva **uključivo ILI,** jer uključuje i stanje kad su obje varijable jednake 1. Uobičajeno se takva funkcija naziva skraćeno ILI – funkcija. Booleov operator ILI označava se sa ∨ ili +. Znak ∨ uobičajen je u matematičkoj logici, dok je znak + u smislu logičke sume više u upotrebi u digitalnoj elektronici.

Logička funkcija ILI može se pisati na slijedeće načine :

$$f = A \vee B = A + B \qquad (3 - 4)$$

Odgovarajući grafički simbol za ILI funkciju prikazan je na slici 3.4.




Slika 3.4. Grafički simbol ILI sklopa

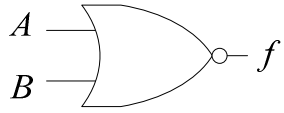
Tablica točnosti sa svim kombinacijama varijabli A i B uz odgovarajuća rješenja funkcije prikazana je na slici 3.5.

| *A* | *B* | *f* |
|---|---|---|
| *0* | *0* | *0* |
| *0* | *1* | *1* |
| *1* | *0* | *1* |
| *1* | *1* | *1* |

Tablica 3.1. Tablica točnosti ILI sklopa

Kombinacijom sklopova sa ILI logičkim sklopom, za razliku od I sklopa, dobivaju se tri različita sklopa koja se ubrajaju u osnovne logičke sklopove. Preostala tri sklopa su NILI, isključivo NILI i isključivo ILI.

Prvi navedeni sklop se dobiva komplementiranjem ILI sklopa, odnosno dodavanjem sklopu ILI sklop NE. Njegova tablica stanja sa svim mogućim stanjima i odgovarajućim rezultatima funkcije uz specifični grafički simbol nalazi se na slici 3.5.

a) 


b)

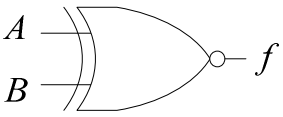
| A | B | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Slika 3.5. Grafički simbol (a) i tablica točnosti (b)

Logička funkcija koja pod uvjetom da su ulazne varijable jednake, odnosno varijable A i B su ili 1 ili 0, daje točan rezultat zove se **isključivo NILI** funkcija. Funkcija isključivo NILI sklopa može se zapisati na slijedeći način :

$$f = AB + \overline{A}\overline{B} \qquad (3 - 5)$$

Odgovarajuća tablica točnosti i grafički simbol isključivo NILI funkcije prikazane su na slici 3.6.

a) 


b)

| A | B | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Slici 3.6. Grafički simbol (a) i tablica točnosti (b)

Zadnja navedena logička funkcija koja daje točan rezultat samo kad su varijable A i B različite jedna od druge zove se **isključivo ILI**. Funkcija isključivo ILI sklopa može se zapisati na slijedeći način :

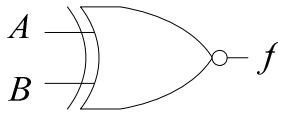
$$f = A\overline{B} + \overline{A}B = A \oplus B \qquad (3 - 6)$$

Odgovarajuća tablica točnosti i grafički simbol isključivo ILI funkcije prikazane su na slici 3.7.

a)

| *A* | *B* | *f* |
|---|---|---|
| *0* | *0* | *1* |
| *0* | *1* | *0* |
| *1* | *0* | *0* |
| *1* | *1* | *1* |

b)



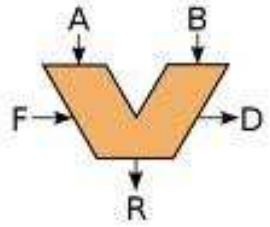

Slika 3.7. Tablica točnosti (a) i grafički simbol (b)

# 4. Aritmetičko – logička jedinica (ALU)

ALU je višefunkcijski *n* – bitni kombinacijski sklop. Temeljni je element centralno procesorske jedinice (CPU) za računalo, pa čak i najjednostavniji mikroprocesori sadrže ALU za namjene kao što je održavanje vremena.

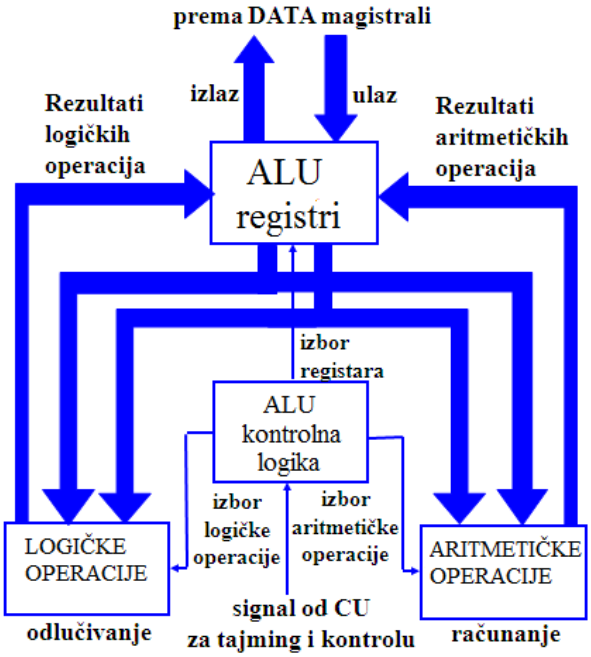
Matematičar John von Neumann predložio ALU koncept u 1945, kada je pisao izvješće o temelje za novo računalo pod nazivom EDVAC. Istraživanje ALUS ostaje važan dio računalne znanosti, spadaju pod aritmetičku i logičku strukturu u ACM Computing Classification System.

Osim prethodno opisanih funkcija ALU služi kao sklop za posmak, te se nalazi iza svake izvršene naredbe CPU – a. Standardizirani karakteristični simbol ALU *n* – bitne jedinice nalazi se na slici 4.1.




Slika 4.1 *n* – bitni ALU simbol

Gdje *n* – bitne ulaze, odnosno operande u ALU označavaju slova A i B. Kontrolni bitovi pomoću kojih se odabire određena ALU funkcija predstavljeni su sa F slovom, dok su prijenos i rezultat odabrane operacije predstavljeni sa slovima D i R. Ustroj unutar svake ALU, sa odgovarajućim registrima prikazan je blok shemom na slici 4.2.




Slika 4.2. Unutarnji ustroj ALU

Korištena i simulirana pomoću Xilinx programskog paketa je paralelna, 4 – bitna 74181 aritmetičko – logička jedinica. 74181 je prva kompletna ALU, koja je izrađena pomoću tranzistorsko – tranzistorske logike (TTL), implementirana u jedan čip. Nalazi se unutar jednog PDIP 24 – pinskog kučišta, sa 16 mogućih aritmetičkih i logičkih funkcija. Korištena je kao aritmetičko – logička jezgra CPU- a i u mnogim povjesno važnijim miniračunalima, te ostalim uređajima. Predstavlja evolucijski korak između CPU- a 1960- tih, koji su konstruirani od konkretnih logičkih sklopova, i današnjih mikroprocesora i CPU- a koji se nalaze unutar jednog čipa.

74181 je kontrolirana pomoču 4- bitnog vektora za odabir funkcije i jednobitnog ulaza M za odabir aritmetičkih ili logičkih operacija. Kada je na jednobitni ulaz M dovedeno stanje binarne jedinice svi unutarnji bitovi prijnosa (eng. carries) su onemogućeni i obavljaju se logičke operacije nad ulaznim bitovima. Obratno ako se na ulaz M dovede binarna nula omogućeni su bitovi prijenosa i obavlja se točno određena aritmetička funkcija odabrana preko ulaznih pinova $S_0$ - $S_3$. Sadržava potpuni unutarnji carry look ahead i pruža mogući carry ripple prijenos između uređaja preko $C_{n+4}$ pina, ili carry look ahead prijenos preko signala propagacije ($\overline{P}$) i signala carry generatora ($\overline{G}$). Na signale $\overline{P}$ i $\overline{G}$ ne utječe ulazni prijenosni bit (eng. carry in). Za sklopove kojima nije bitna brzina izvođenja operacije koristi se ripple carry mod, spajajući signal $C_{n+4}$ sa signalom $C_{in}$ slijedećega

sklopa. Sklopovi sa velikom brzinom izvođenja operacija koriste 74181 sklop u spoju sa 74181 carry look ahead sklopom. Na svake četiri 181 aritmetičko – logičke jedinice koristi se jedan sklop 182, koji u spoju pružaju mogućnost izvođenja operacija velikom brzinom nad *n* – bitnim brojevima.

Komparatoriski izlaz A = B prelazi u stanje binarne jedinice pod uvijetom da se na svim pinovima vektorskog 4 – bitnog izlaza ($F_0 - F_3$) pojavi logička jedinica, te se koristi kao signal za ispitivanje jednakosti dva ulazna broja pri odabranoj funkciji oduzimanja. A = B signal u kombinaciji sa signalom $C_{n+4}$ daje mogućnost provjere (usporedbe) dva broja (operanda).

Tablica operacija (tablica 4.1) prikazana sa lijeve strane pokazuje aritmetičke operacije kad je na ulazni pin $C_{in}$ dovedena logička jedinica, te tablica 4.2 prikazuje operacije kada je signal prijenosnog bita jednak logičkoj nuli.

| MODE SELECT INPUT | | | | ACTIVE INPUTS AND OUTPUTS | |
|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | LOGIC (M = H) | ARITHMETIC (M = L;Cn = H) |
| L | L | L | L | $\overline{A}$ | A |
| L | L | L | H | $\overline{A + B}$ | A + B |
| L | L | H | L | $\overline{A}B$ | A + $\overline{B}$ |
| L | L | H | H | Logical 0 | minus 1 |
| L | H | L | L | $\overline{AB}$ | A plus A$\overline{B}$ |
| L | H | L | H | $\overline{B}$ | (A + B) plus A$\overline{B}$ |
| L | H | H | L | A ⊕ B | A minus B minus 1 |
| L | H | H | H | A$\overline{B}$ | AB minus 1 |
| H | L | L | L | $\overline{A}$ + B | A plus AB |
| H | L | L | H | $\overline{A \oplus B}$ | A plus B |
| H | L | H | L | B | (A + $\overline{B}$) plus AB |
| H | L | H | H | AB | AB minus 1 |
| H | H | L | L | Logical 1 | A plus A[(1)] |
| H | H | L | H | A + $\overline{B}$ | (A + B) plus A |
| H | H | H | L | A + B | (A + $\overline{B}$) plus A |
| H | H | H | H | A | A minus 1 |

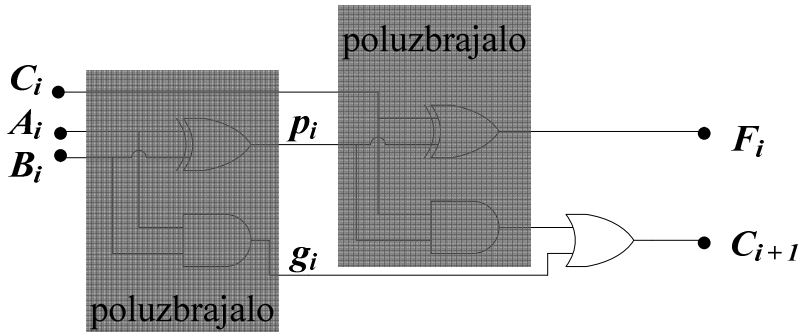
Tablica 4.1. Tablica operacija sa Cn = H

| MODE SELECT INPUT | | | | ACTIVE INPUTS AND OUTPUTS | |
|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | LOGIC (M = H) | ARITHMETIC (M = L;Cn = L) |
| L | L | L | L | $\overline{A}$ | A minus 1 |
| L | L | L | H | $\overline{AB}$ | A B minus 1 |
| L | L | H | L | $\overline{A}$ + B | A $\overline{B}$ minus 1 |
| L | L | H | H | Logical 1 | minus 1 |
| L | H | L | L | $\overline{A + B}$ | A plus (A + $\overline{B}$) |
| L | H | L | H | $\overline{B}$ | AB plus (A + $\overline{B}$) |
| L | H | H | L | $\overline{A \oplus B}$ | A minus B minus 1 |
| L | H | H | H | A + $\overline{B}$ | A + $\overline{B}$ |
| H | L | L | L | $\overline{A}$ B | A plus (A + B) |
| H | L | L | H | $\overline{A \oplus B}$ | A plus B |
| H | L | H | L | B | (A + B)A$\overline{B}$ |
| H | L | H | H | A + B | A + B |
| H | H | L | L | Logical 0 | A plus A[(1)] |
| H | H | L | H | A $\overline{B}$ | A B plus A |
| H | H | H | L | A B | A $\overline{B}$ plus A |
| H | H | H | H | A | A |

Tablica 4.2. Tablica operacija sa Cn = L

## 4.1 Carry – look ahead

Razvojem digitalnih sklopova osmišljavali su se razni algoritmi i načini spajanja za postizanje što veće brzine izvođenja raznih operacija. Najčešće korištena operacija je operacija zbrajanja. Prethodno prikazana i objašnjena operacija nad dva bita izvodi se pomoću potpunog zbrajala. Potpuno zbrajalo je osmišljeno na principu carry ripple metode zbrajanja, koju je nemoguće koristiti sa sklopovima velike brzine. Radi povećanja brzine osmišljena je carry – look ahead metoda zbrajanja, koja je idejno jednostavna, ali sklopovski kompliciranija metoda. Carry – look ahead koristi se signalima generiranja ( eng. *Generate*), širenja (eng. *Propagate*) i prijenosa (eng. *Carry*).




Slika 4.3. Carry – look ahead potpuno zbrajalo

Signal $g_i$ koji predstavlja signal generiranja se nalazu u stanju binarne jedinice ako se zbrajanjem dva jednobitna broja $A_i$ i $B_i$ pojavljuje prijenosni bit, neovisno o stanju ulaznog prijenosnog signala $C_i$. Na primjer, zbrajanjem dva broja u degadskom sustavu (npr. 53 + 65), gdje se prilikom zbrajanja desetica pojavljuje jedna stotica bez obzira na brojeve koji se nalaze na mjestu jedinica. Pojavom stotice, odnosno preljevom znamenaka generira se signal $g_i$. U binarnom sustavu signal $g_i$ se generira jedino u slučaju kada su obje ulazne znamenke jednake binarnoj jedinici. Prethodna tvrdnja može se prikazati matematičkom funkcijom I – logičkog sklopa.

$$g_i = A \wedge B = A \,\&\, B = A \cdot B = A\,B \qquad (4\text{ - }1)$$

Signal širenja ($p_i$) prelazi u stanje binarne jedinice ako se zbrajanjem dva jednobitna broja A i B pojavi prijenosni bit neovisno o ulaznom signalu $C_i$. Odnosno signal širenja se nalazi u stanju binarne jedinice ako i samo ako je barem na jednom od ulaznih jednobitnih brojeva A i B stanje binarne jedinice. Pisanjem funkcije koja opisuje zadane uvjete sa dvije nezavisne varijable dobiva se funkcija ILI – logičkog sklopa.

$$p_i = A \vee B = A + B \qquad (4 - 2)$$

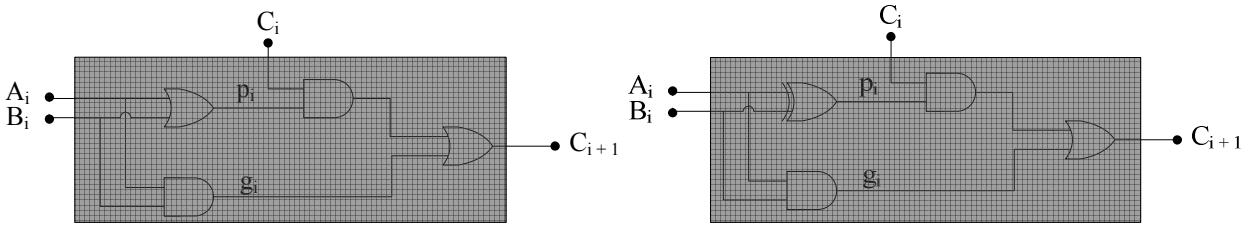
Uzme li se u obzir samo druga definicija, da se signal $p_i$ nalazi u stanju binarne jedinice kada je ili na A ili na B ulazu binarna jedinica, moguće je koristiti i isključivo ILI sklop umjesto ILI sklopa. Isključivo ILI sklop je sporiji od ILI skopa (za čiju implementaciju je potrebno manje tranzistora od isključivo ILI sklopa), no pogodniji je za korištenje kod višebitnih carry – look ahead jedinica. Matematička funkcija definicije odgovara funkciji isključivo ILI sklopa.

$$p_i = A\overline{B} + \overline{A}B = A \oplus B \qquad (4 - 3)$$

Uz objašnjene korištene signale $p_i$ i $g_i$ carry – look ahead jedinice, dolazi se uz pomoć Booleanove algebre do formule koja obuhvaća cjelokupnu jedinicu.

$$C_{i+1} = G_i + (P_i \cdot C_i) \qquad (4 - 4)$$

Prema formuli za carry – look ahead jedinicu dolazi se do modificiranog sklopa potpunog zbrajala pod nazivom djelomično potpuno zbrajalo.




a) b)

Slika 4.4. Djelomično zbrajalo a) sa ILI – logičkim sklopom i

b) sa isključivo ILI – logičkim sklopom

| $A_i$ | $B_i$ | $C_i$ | $p_i$ | $G_i$ | $C_{i+1}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

a)

| $A_i$ | $B_i$ | $C_i$ | $p_i$ | $g_i$ | $C_{i+1}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |

b)

Tablica 4.3. Tablica stanja djelomično potpunog zbrajala a) sa ILI – logičkim sklopom i b) sa isključivo ILI – logičkim sklopom

Razlika između djelomičnog potpunog zbrajala sa ILI sklopom i djelomičnog potpunog zbrajala sa isključivo ILI sklopom, koja je vidljiva iz tablice stanja, je samo u zadnja dva stanja $p_i$ signala. Izlazni signal, neovisno koje je djelomično potpuno zbrajalo odabrano, ostaje ne promijenjen.

Za svaka dva bita koja se zbrajaju u binarnom redosljedu carry – look ahead sklop odredit će hoće li ta dva bita postaviti signale $p_i$ i $g_i$ u binarnu jedinicu. Takav princip rada omogućava određivanje prijenosnog bita prije sume dva bita. Na taj način nema čekanja prijenosnog bita, ili potrebnog vremena da se prijenosni bit prenese od prvog potpunog zbrajala to zadnjega. 4 – bitna carry – look ahead jedinica sastoji se od 4 modificirana potpuna zbrajala pod nazivom djelomično potpuno zbrajalo (DPZ). Svako zbrajalo potrebno je pravilno međusobno spojiti prema formulama do kojih se dolazi primjenom osnovne formule za carry – loog ahead jedinice. Označavajući prvo djelomično zbrajalo kao prvo, i numerirajući redosljedom svako slijedeće potrebno je definirati signale $C_1$, $C_2$, $C_3$ i $C_4$.

$$C_1 = g_0 + p_0 \cdot C_0 \qquad (4 - 5)$$

$$C_2 = g_1 + p_1 \cdot C_1 \qquad (4 - 6)$$

$$C_3 = g_2 + p_2 \cdot C_2 \qquad (4 - 7)$$

$$C_4 = g_3 + p_3 \cdot C_3 \qquad (4 - 8)$$

Uvrštavajući jednu formulu u drugu dobiva se konačni prijenosni signal :

$$C_1 = g_0 + p_0 \cdot C_0 \qquad (4 - 9)$$

$$C_2 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot C_0 \qquad (4 - 10)$$

$$C_3 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot C_0 \qquad (4 - 11)$$

$$C_4 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot C_0 \qquad (4 - 12)$$

Određujući ukupni signal širenja i generiranja :

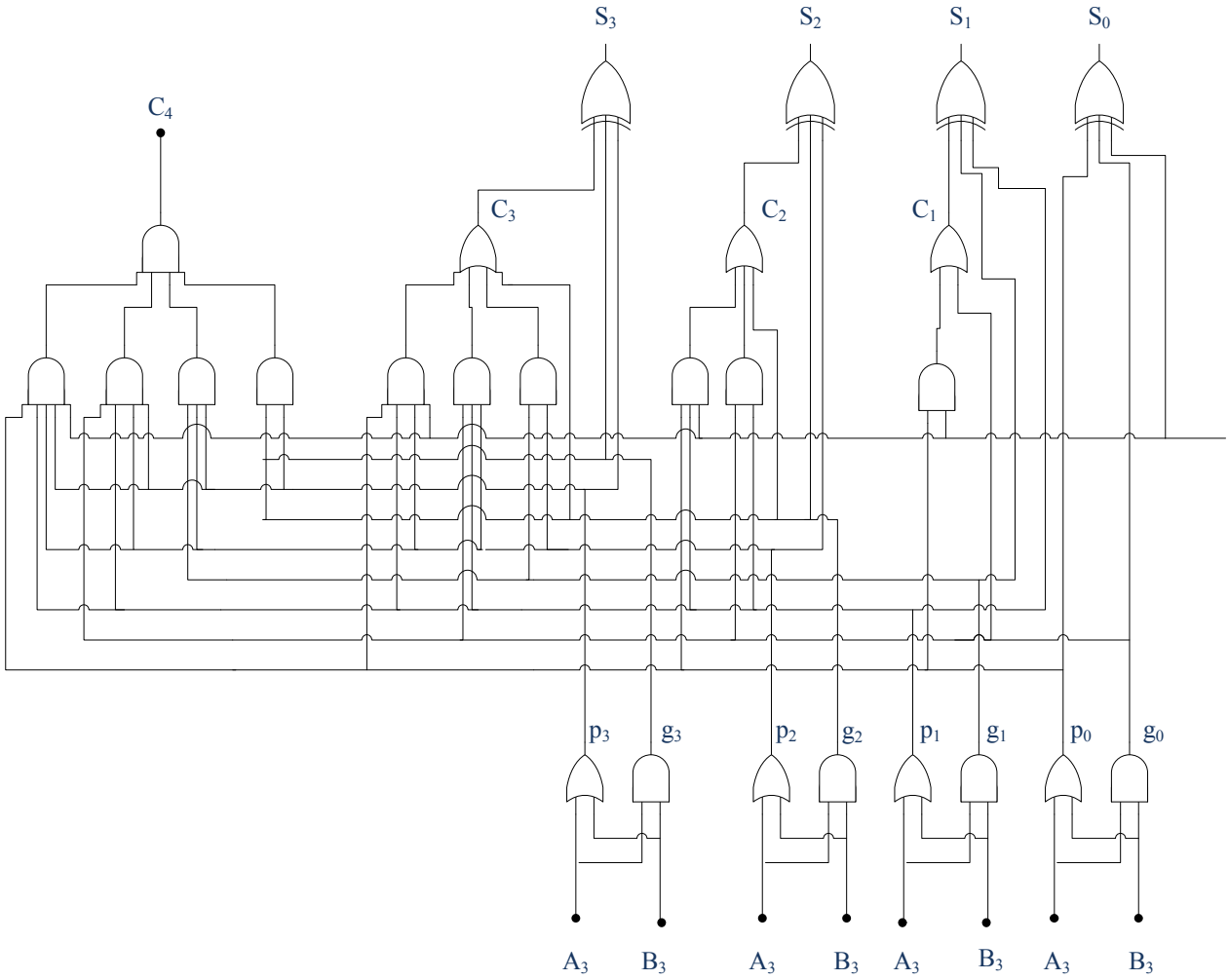
$$P_u = p_3 \cdot p_2 \cdot p_1 \cdot p_0 \qquad (4 - 13)$$

$$G_u = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 \qquad (4 - 14)$$

Uvrštavanjem $G_u$ i $P_u$ u osnovnu formulu dolazi se do formule za određivanje krajnjeg prijenosnog bita.

$$C_{i+1} = G_u + (P_u \cdot C_o) \qquad (4 - 15)$$

Prethodno dobivena formula popraćena je odgovarajućim sklopom na slici 4.5.




Slika 4.5. 4 – bitno carry – look ahead zbrajalo

## 4.2 Usporedba carry ripple i carry – look ahead metode

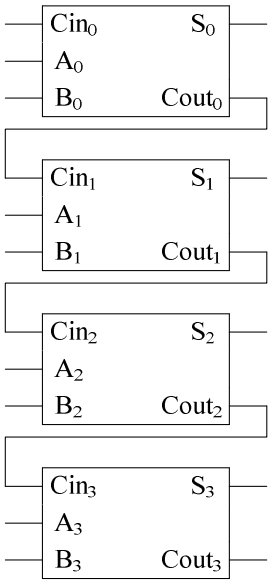
Kašnjenje bilo kojeg sklopa određuje se zbrajanjem sklopova na najdužem putu kroz koje prolazi signal. Prema istome principu određivanja kašnjenja 4 – bitna carry – look ahead jedinica ima kašnjenje od jednog sklopa (1T) za signale $p_i$ i $g_i$, te kašnjenja od 3T i 4T za signale $C_i$ i $S_i$. Najviše vremena je potrebno za signal $S_i$, pa sama 4 – bitna carry – look ahead jedinica ima kašnjenje od četiri sklopa.

Formula za proračun kašnjenja *n* – bitnog carry – look ahead zbrajala sastavljenog od 4 – bitnih carry – look head zbrajala :

$$T_{cla} = 2 + 2 \cdot ( n / 4 ) \qquad (4 - 16)$$

Prema formuli kašnjenje 16 – bitnog zbrajala iznosi 10T.

Carry ripple zbrajala kojima je zbog kaskadnog spoja poteban broj potpunih zbrajala jednakih broju bitova binarnog broja imaju puno značajnije kašnjenje.




Slika 4.6. 4 - bitno carry ripple zbrajalo

Za 4 – bitno carry ripple zbrajalo potrebna su četiri potpuna zbrajala od kojih svako zasebno zbrajalo ima kašnjenje od 2 sklopa (2T), pa maksimalno kašnjenje iznosi 8T. Prema toj tvrdnji lako se dolazi do formule za računanje maksimalnog kašnjenja za carry ripple metodu.

$$T_{cra} = n \cdot 2 \qquad (4 - 17)$$

Prema napisanoj formuli za 16 – bitno zbrajalo dobiva se maksimalno kašnjenje od 32T.

Usporedbom 16 – bitnog carry ripple i carry look ahead moda dolazimo jasne razlike u brzinama izvedbe.

$$T_{cla16} = 2 + 2 \cdot ( n / 4 ) = 10 \qquad (4 - 18)$$

$$T_{cra16} = n \cdot 2 = 32 \qquad (4 - 19)$$

## 4.3 Simulacija

Korišten je, prethodno spomenut, softverski paket Xilinx ISE 9.2i za izvedbu simulacije. Sami kombinacijski sklop (ALU), odnosno svaka njegova mogučnost opisana je pomoću VHDL koda, a kompletan ispis koda se nalazi u prilogu P.9.3. Prilikom izvođenja simulacije dovedeni su nasumični signali na pojedine ulaze, te su popratno zabilježeni izlazni signali. Na slici 4.7. uočljivo je da su prvo izvedene i popraćene logičke operacije, a na sljedećoj slici 4.8. sve aritmetičke operacije. Na slikama 4.7. i 4.8. prikazani su zajedno i ulazni i izlazni signali, pod ulazne signale spadaju *M, Cin, a, b* i *SEL* signali, a pod izlazne preostali (*Cout, equ, Pu, Gu* i *Y*) signali. Jednobitni signali prikazani su preko svoja jedina dva stanja ‘0’ i ‘1’, dok su četverobitni signali *a, b, SEL* i *Y* prikazani u heksadecimalnom sustavu.

4.7. Simulacija logičkih operacija

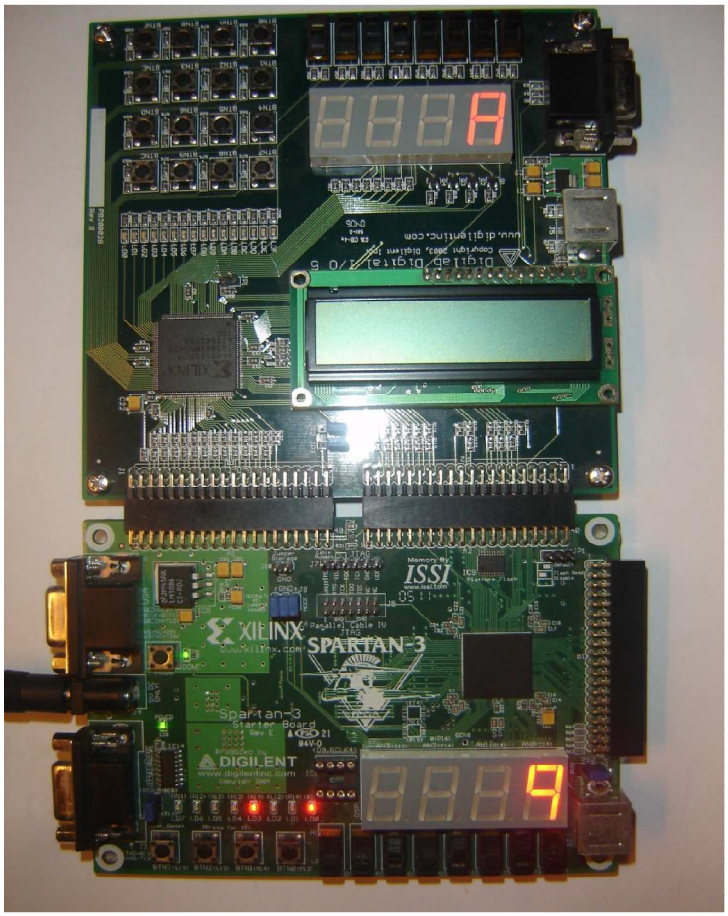



4.8. Simulacija aritmetičkih operacija

## 4.4 Realizacija

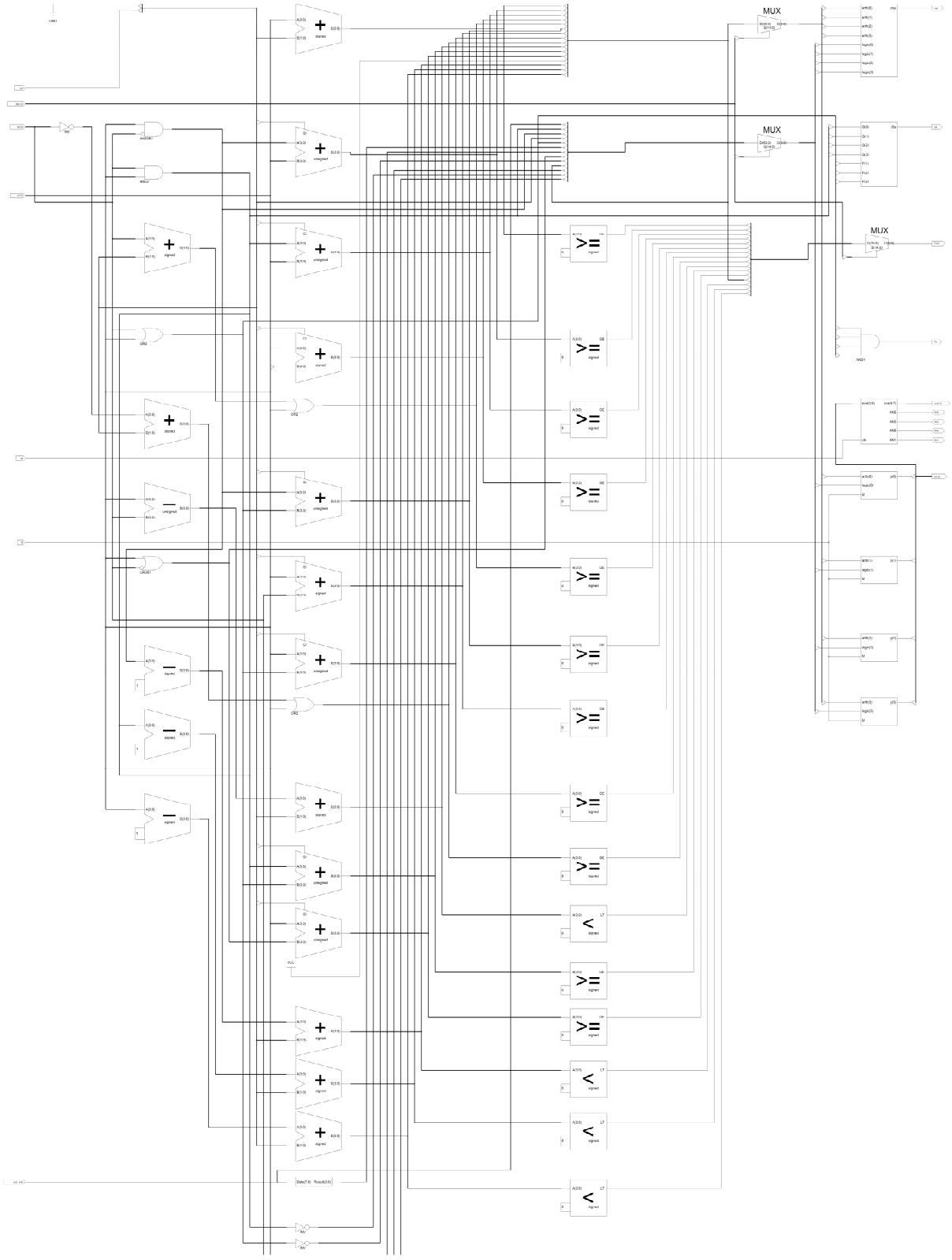
Izvršenom simulacijom i provjerom svih operacija, te utvrđivanjem njihove ispravnosti i funkcioniranja, napisani VHDL kod projveren je i pomoću Spartan 3 razvojnog susava. Osim pisanja koda Xilinx paket je korišten za programiranje koda u FPGA čip, ostvarivanje logičke sheme sklopa, te izračunavanje kašnjenja sklopa. Posebno prilikom programiranja razvojnog sustava napisan je dodatni VHDL kod pomoću kojeg se na sedam segmentnom led displeju prikazuju aritmetičke operacije, radi jednostavnije provjere aritmetičkih operacija.

Na slici 4.9. nalazi se Spartan 3 razvojni sustav na kojem je na sedam segmentnom pokazniku vidljiva znamenka 9, koja je rezultat operacije zbrajanja a i b porta na kojima se u binarnom sustavu nalaze brojevi 4 i 5. Prekidači na dolnjoj strani slike (ispod pokaznika) definiraju ulaze a i b, dok prekidači na gornjoj strani slike označavaju selektivni ulazi i ulaz *M* za odabir aritmetičkih ili logičkih funkcija.




4.9. Spartan 3 razvojni sustav

Izvještaj o vremenu propagacije sa svaki pojedini dio sklopa, te za cijeli sklop nalazi se u prilogu. Radi mogućnosti usporedbe vremena tvorničkog sklopa 74181 sa simuliranim skopom u prilogu se nalaze specifikacije 74181 sklopa. Na slici 4.10. nalazi se logička shema cijelokupnog sklopa.




4.10. Logička shema sklopa

# 5. Zaključak

Jedan od osnovnih dijelova svake matematičke i logičke operacije, pa tako i samim time nezaobilazan dio svakog procesora je aritmetičko logička jedinica. Sastavljena od raznih kombinacija osnovnih digitalnih sklopova pomoću koje je moguče realizirati veliki spektar raznih operacija. Prilikom realizacije jedan od glavnih elemenata je bila minimalizacija samog skopa, rješavanje tog problema ostvarilo se korištenjem istih komponenti na razne načine za ostvarenje određene operacije. Opisivanjem osnovnih logičkih i aritmetičkih operacija, te njihovim spajanjem i kombinacijom dolazi se do ostvarivanja složenijih operacija. Vrlo veliku pomoć pri opisivanju i realizaciji pružio je Xilinx 9.2i softverski paket. Pomoću kojega je kompletno simulirana, realizirana i provjerena aritmetičko logička jedinica. Pomoću istoimenog softvera provjerena su maksimalna i minimalna vremena potrebna za ostvarenje pojedine operacije, te se samim time ustvrđuje maksimalna frekvencija rada sklopa. Ostvarenjem same jedinice vidljivo je da se pomoću algebre jedan komplicirani sklop može uvelike pojednostaviti i samim time smanjiti.

# 6. Literatura

## 6.1 Popis korištenih knjiga:


- U. Peruško, Digitalna elektronika, školska knjiga Zagreb, 1991
- Volnei A. Pedroni, Circuit design with VHDL, 2004
- D. A. Patterson, J. L. Hennesy, Computer organization and design, 2005


## 6.2 Popis korištenih web stranica:


- S. Ribarić, Aritmetičko – logička jedinica, http://www.zemris.fer.hr/predmeti/aior/Predavanja/PDF/08aritmeticka-jedinica.pdf
- S. Winchenbach, M. Driss, 8-Bit Arithmetic Logic Unit, University of Maine 1998, http://www.eece.maine.edu/vlsi/ALU/ALU/ALU.pdf
- http://en.wikipedia.org/wiki/Arithmetic_logic_unit
- Yu-Ting Pai, Yu-Kumg Chen, The Fastest Carry Lookahead Adder, Department of Electronic Engineering Huafan University
- http://en.wikipedia.org/wiki/Division_(digital)
- D. J. Sorin, Booth's Algorithm
- A.P.Godse,D.A.Godse, Computer organization, Technical Publications Pune, 2010
- http://en.wikipedia.org/wiki/Adder_(electronics)
- D. Pham, S. Galal, Division Algorithms and Hardware Implementations

# 7. Sažetak

## 7.1 Aritmetičko logička jedinica ( ALU )

Izradom završnog rada, obrađene su i objašnjene osnovne aritmetičke i logičke operacije. Prikazani su njihovi odgovarajući logički simboli i pripadne matematičke jednadžbe za pojedini sklop. Osim osnovnih matematičkih operacija, pojašnjene su i malo kompliciranije matematičke operacije (množenje i dijeljenje) unutar kojih su dodatno opisani algoritmi (Boothov algoritam, non-restoring algoritam i restoring algoritam). Aritmetičko logička jedinica napisana je pomoću VHDL koda, te je unutar istoga softverskog paketa simulirana. Pomoću Spartan 3 razvojnog sustava i Xilinx softvera napisani VHDL kod je preko paralelnog porta prebačen u razvojni sustav, te je na taj način sama aritmetičko logička jedinica realizirana. Maksimalno vrijeme i logička shema sklopa ostvarena je unutar softvera.

**Ključne riječi :** aritmetičke operacije, logičke operacije, algoritmi, ALU, simulacija, realizacija, maksimalno vrijeme sklopa

## 7.2 Arithmetic logic unit (ALU)

Creating the final work, elaborated and explained are the basic arithmetic and logic operations. Demonstrated are their proper logical symbols and the corresponding mathematical equations for each circuit. In addition to basic mathematical operations, are described and a little more complicated mathematical operations (multiplication, division) within which are further described algorithms (Boothov algorithm, non-restoring algorithm and restoring algorithm). Arithmetic logic unit is written using the VHDL code, and within the same software package is simulated. With the Spartan 3 development system and Xilinx software written VHDL code is transferred via the parallel port in the developmental system, and is thus itself arithmetic logic unit realized. Maximum time and logic circuit scheme is realized within the software.

**Keywords:** arithmetic operations, logical operations, algorithms, ALU, simulation, realization, maximum circuit time

## 8. Životopis

Davor Bogdanović rođen 4.travnja 1988.godine u Slavonskom Brodu gdje 2002. godine završava osnovnu školu „Vladimir Nazor“. U Slavonskom Brodu iste godine upisuje srednju Tehničku školu, te se opredjeljuje za smjer telekomunikacije. Nakon završene srednje škole 2006. godine upisuje se na Elektrotehnički fakultet u Osijeku. Na drugoj godini opredjeljuje se za smjer komunikacije te na taj način nastavlja svoje srednjoškolsko opredijeljenje.

_____

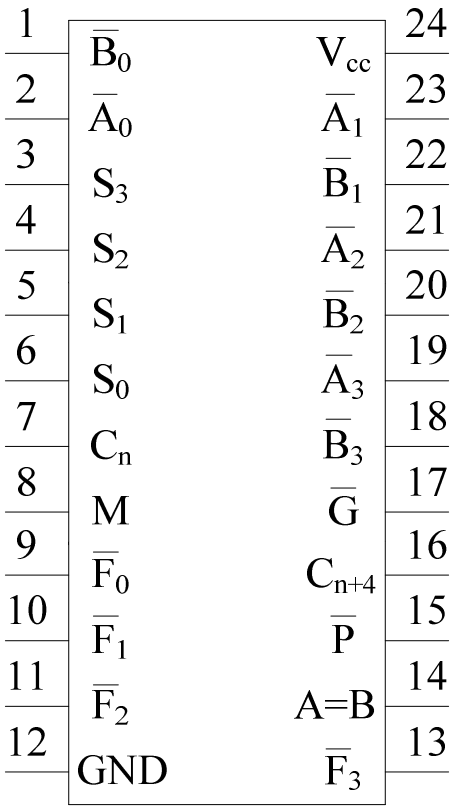
DAVOR BOGDANOVIĆ

# 9. Prilozi

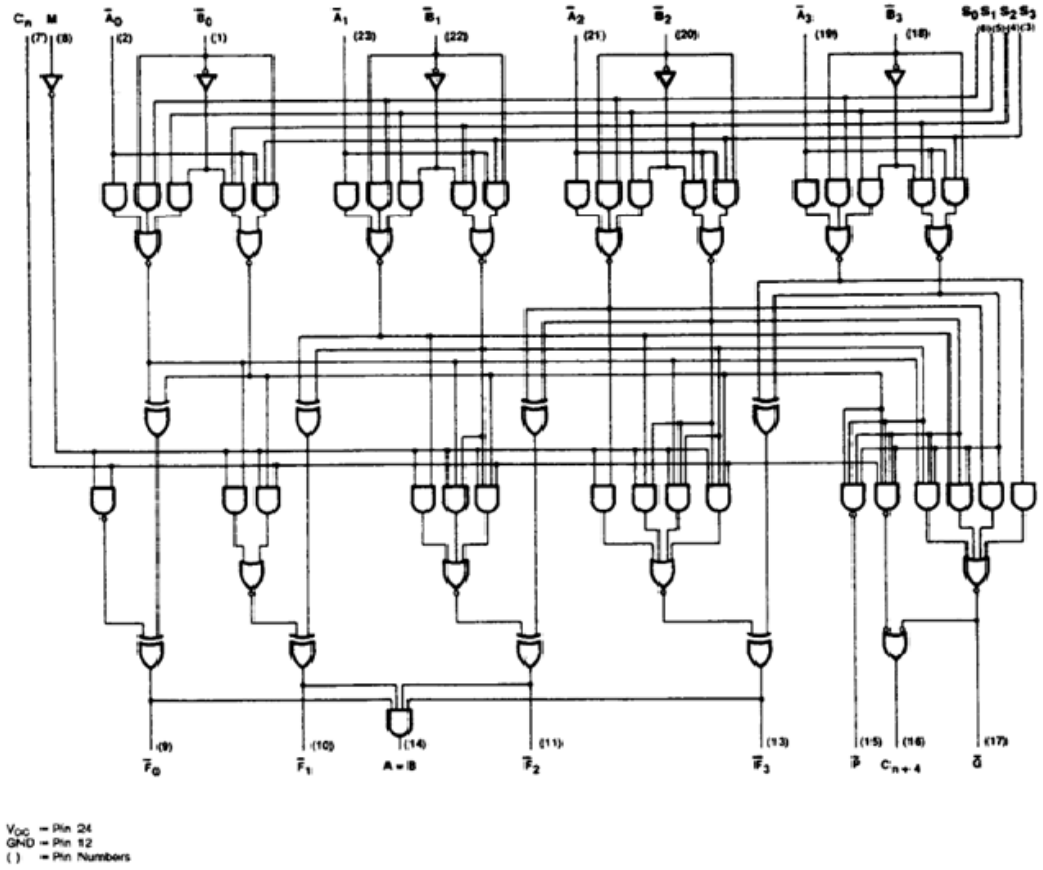
## 9.1 P. 9.1 Specifikacije 74181 sklopa

| TYPE | TYPICAL PROPAGATION DELAY | TYPICAL SUPPLY CURRENT (TOTAL) | FREQUENCY |
|---|---|---|---|
| 74181 | 22 ns | 91 mA | $45.45 * 10^6\ Hz$ |

Popis pinova na 74181 sklopu:



Logička shema sklopa :




$V_{CC}$ = Pin 24
GND = Pin 12
( ) = Pin Numbers

Električne karakteristike sklopa :

| PARAMETER | | TEST CONDITIONS |
|---|---|---|
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $C_n$ to $C_{n+4}$ | M = 0V, Sum or Diff Mode see Waveform 2 and Tables I & II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $C_n$ to $\overline{F}$ outputs | M = 0V, Sum or Diff Mode see Waveform 2 and Tables I & II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $\overline{G}$ output | $M = S_1 = S_2 = 0V$, $S_0 = S_3 = 4.5V$ Sum Mode, see Waveform 2 and Table I |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $\overline{G}$ output | $M = S_0 = S_3 = 0V$, $S_1 = S_2 = 4.5V$ Diff Mode, see Waveform 3 and Table II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $\overline{P}$ output | $M = S_1 = S_2 = 0V$, $S_0 = S_3 = 4.5V$ Sum Mode, see Waveform 2 and Table I |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $\overline{P}$ output | $M = S_0 = S_3 = 0V$, $S_1 = S_2 = 4.5V$ Diff Mode, see Waveform 3 and Table II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}_i$ or $\overline{B}_i$ inputs to $\overline{F}_i$ outputs | $M = S_1 = S_2 = 0V$, $S_0 = S_3 = 4.5V$ Sum Mode, see Waveform 2 and Table I |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}_i$ or $\overline{B}_i$ inputs to $\overline{F}_i$ outputs | $M = S_0 = S_3 = 0V$, $S_1 = S_2 = 4.5V$ Diff Mode, see Waveform 3 and Table II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}_i$ or $\overline{B}_i$ inputs to $\overline{F}_i$ outputs | M = 4.5V, Logic Mode see Waveform 2 and Table III |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $C_{n+4}$ output | $M = 0V$, $S_0 = S_3 = 4.5V$, $S_1 = S_2 = 0V$ Sum Mode, see Waveform 1 and Table I |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to $C_{n+4}$ outputs | $M = 0V$, $S_0 = S_3 = 0V$, $S_1 = S_2 = 4.5V$ Diff Mode, see Waveform 4 and Table II |
| $t_{PLH}$ $t_{PHL}$ | Propagation delay $\overline{A}$ or $\overline{B}$ inputs to A = B output | $M = S_0 = S_3 = 0V$, $S_1 = S_2 = 4.5V$ Diff Mode, see Waveform 3 and Table II |

## 9.2 P.9.2. Izvještaj o vremenu propagacije realiziranog sklopa

Popis korištenih sklopova :

Adders/Subtractors : 26

4-bit adder : 19

4-bit adder carry in : 2

4-bit subtractor : 5

Comparators : 15

4-bit comparator greatequal : 11

4-bit comparator less : 4

Multiplexers : 3

1-bit 16-to-1 multiplexer : 1

4-bit 16-to-1 multiplexer : 2

Xors : 1

4-bit xor2 : 1

Selected Device : 3s200ft256-5

Number of Slices: 85 out of 1920 4%

Number of 4 input LUTs: 163 out of 3840 4%

Number of IOs: 22

Number of bonded IOBs: 22 out of 173 12%

Vrijeme kašnjenja :

Delay: 16.354ns (Levels of Logic = 11)

Maksimalna frekvencija rada :

$$f = \frac{1}{t} = \frac{1}{16.354 * 10^{-6}} = 61.147 * 10^{6}\ Hz$$

## 9.3 P.9.3. VHDL kod aritmetičko logičke jedinice

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity ALU_VHDL is
   port(  M,Cin: in std_logic;
      Cout,equ,Pu,Gu: out std_logic;
      a: in std_logic_vector(3 downto 0);
      b: in std_logic_vector(3 downto 0);
      sel: in std_logic_vector(3 downto 0);
      y: out std_logic_vector(3 downto 0));
end ALU_VHDL;
architecture Behavioral of ALU_VHDL is
     signal logic, arth, shift, P, G: std_logic_vector(3 downto 0);
     signal Couts: std_logic;
     begin
          shift(0)<=a(3);
     A1: FOR i IN 1 TO 3 GENERATE
          shift(i) <= a(i-1);
     end generate;
     A2: FOR i IN 0 TO 3 GENERATE
          P(i) <= a(i)or b(i);
     end generate;
     A3: FOR i IN 0 TO 3 GENERATE
          G(i) <= a(i)and b(i);
     end generate;
               with sel select
               logic<= not a when "0000",
                        not (a and b) when "0001",
```

```
                not a or b when "0010",
                "1111" when "0011",
                not (a or b) when "0100",
                b when "0101",
                not (a xnor b) when "0110",
                a or (not b)when "0111",
                (not a) and b when "1000",
                a xor b when "1001",
                b when "1010",
                a or b when "1011",
                "0000" when "1100",
                a and (not b) when "1101",
                a and b when "1110",
                a when others;
PROCESS (sel,a,b,Cin)
     begin
          case sel is
               when "0000" =>
                    if (a-1+Cin<0)then
                         arth<= a-1+Cin;
                         Couts<= '1';
                    else
                         arth<= a-1+Cin;
                         Couts<= '0';
                    end if;
               when "0001" =>
                    if ((a and b)-1 + Cin<0)then
                         arth<= (a and b)-1+Cin;
                         Couts<='1';
                    else
                         arth<= (a and b)-1+Cin;
                         Couts<='0';
```

```
                    end if;
               when "0010" =>
                    if((a and (not b))-1+Cin<0)then
                         arth<= (a and (not b))-1+Cin;
                         Couts<='1';
                    else
                         arth<= (a and (not b))-1+Cin;
                         Couts<='0';
                    end if;
               when "0011" =>
                         arth<="1111";
                         Couts<='1';
               when "0100" =>
                    if ((a+(a or (not b))+Cin)>=16)then
                         arth<= a+(a or ( not b))+Cin;
                         Couts<='1';
                    else
                         arth<=a + (a or (not b))+Cin;
                         Couts<='0';
                    end if;
               when "0101" =>
                    if   (((a   and   b)   +   (a   or
b)+Cin)>=16)then
                         arth<= (a and b) + (a or b)+Cin;
                         Couts<='1';
                    else
                         arth<= (a and b) + (a or b)+Cin;
                         Couts<='0';
                    end if;
               when "0110" =>
                    if(a-b-1+Cin<0)then
                         arth<= a-b-1+Cin;
```

```
                    Couts<= '1';
               else
                    arth<=a-b-1+Cin;
                    Couts<= '0';
               end if;
          when "0111" =>
               if ((a or (not b)+Cin)>=16)then
                    arth<= a or (not b)+Cin;
                    Couts<='1';
               else
                    arth<= a or (not b)+Cin;
                    Couts<='0';
               end if;
          when "1000" =>
               if ((a+(a or b)+Cin)>=16)then
                    arth<=a+(a or b)+Cin;
                    Couts<='1';
               else
               arth<=a+(a or b)+Cin;
                    Couts<='0';
               end if;
          when "1001" =>
               if ((a+b+Cin)>=16)then
                    arth<=a+b+Cin;
                    Couts<='1';
               else
                    arth<=a+b+Cin;
                    Couts<='0';
               end if;
          when "1010" =>
               if   (((a   and   (not   b))+(a   or
b)+Cin)>=16)then
```

```
                    arth<=(a  and  (not  b))+(a  or
b)+Cin;
                    Couts<='1';
               else
                    arth<=(a  and  (not  b))+(a  or
b)+Cin;
                    Couts<='0';
               end if;
          when "1011" =>
               if ((a or b + Cin)>=16)then
                    arth<=a or b + Cin;
                    Couts<='1';
               else
                    arth<=a or b + Cin;
                    Couts<='0';
               end if;
          when "1100" =>
               if ((a+shift+Cin)>=16)then
                    arth<= a+shift+Cin;
                    Couts<='1';
               else
                    arth<= a+shift+Cin;
                    Couts<='0';
               end if;
          when "1101" =>
               if (((a and b)+a+Cin)>=16)then
                    arth<=(a and b)+a+Cin;
                    Couts<='1';
               else
                    arth<=(a and b)+a+Cin;
                    Couts<='0';
               end if;
```

```
                    when "1110" =>
                         if (((a and (not b))+a+Cin)>=16)then
                              arth<=(a and (not b))+a+Cin;
                              Couts<='1';
                         else
                              arth<=(a and (not b))+a+Cin;
                              Couts<='0';
                         end if;
                    when others =>
                         if (a+Cin>=16)then
                              arth<=a+Cin;
                              Couts<='1';
                         else
                              arth<=a+Cin;
                              Couts<='0';
                         end if;
               end case;
          end process;
     Pu <= P(0) and P(1) and P(2) and P(3);
     Gu <= G(3) or (P(3)and G(2)) or (P(3)and P(2) and G(1)) or
(P(3) and P(2) and P(1) and G(0));
     with M select
          y<= logic when '1',
                arth when others;
     with M select
          Cout<= '0' when '1',
               Couts when others;
     equ<='1' when (arth="1111" or logic="1111") else
            '0';
end Behavioral;
```