

**REALIZACIJA KONAČNIH MAŠINA STANJA U PROGRAMABILNIM
LOGIČKIM KOLIMA**

FINITE STATE MACHINE DESIGN WITH PROGRAMMABLE LOGIC

AUTOR:

DUŠAN KRČUM

III razred XIV beogradske gimnazije
Regionalni centar za talente Beograd II

MENTOR:

Docent, Dr. **LAZAR SARANOVAC**, dipl. inž.
Univerzitet u Beogradu, Elektrotehnički fakultet
Katedra za elektroniku

Rezime

Teorija konačnih mašina stanja (*finite state machines*, FSM) se koristi u realizaciji digitalnih sistema kako u realizaciji hardvera tako i u realizaciji softvera. Mnogi delovi računarskih sistema su realizovani kao konačne mašine stanja, na primer mikrosekvencer u procesoru. Današnja tehnologija i načini realizacije dozvoljavaju da se projektuju i koriste kompletni računarski sistemi u jednom čipu (*system on chip*, SoC). To se po pravilu radi u programabilnim logičkim kolima (*field programmable gate array*, FPGA) pri čemu se realizacija konačnih mašina stanja mora prilagoditi strukturama programabilnih komponenti. Da bi se ovakav proces dosledno sproveo na današnjem nivou tehnologije izvršeno je detaljno istraživanje i predloženo pogodno rešenje na primeru binarnih množača kao sastavnih delova aritmetičko logičkih jedinica.

Ključne reči: konačne mašine stanja, FSM, programabilne komponente, FPGA, binarni množač, Butov algoritam

Summary

Finite state machines theory is one of most useful technique in realization hardware and software. Many computer systems are realized by finite state machine concepts, like micro sequencer in CPU. Present technology is capable to design complete computer system in one chip, system on chip, SoC. One of most useful component in prototyping and design is field programmable gate array, FPGA. Realization of finite state machines in that case must be compatible with programmable component structure. Detail research within present technology is made in order to provide suitable solutions for binary multiplier, necessary in arithmetic logic units.

Key words: finite state machines, FSM, programmable components, FPGA, binary multiplier, Booth's algorithm

I. Uvod

Stanje tehnologije je uvek određivalo mogućnosti realizacije nakupljenih znanja. Mnoge teoretske postavke nisu mogle biti implementirane dok stepen razvoja tehnologije to nije omogućio. Tipičan primer je i u elektronici. Osnove moderne digitalne elektronike je postavio *George Boole* još 1854 godine, ali ne sa idejom primene u elektronici, koja tada nije ni postojala, nego sa idejom da se napravi formalizacija matematičke logike. Sa razvojem tehnologije i pojavom elektronike kao posebne oblasti 1938 godine je *Claude Shannon* uočio i primenio elemente Bulove matematičke logike na relejna, digitalna, kola i tada je nastala digitalna elektronika. Slična situacija je i danas. Stalna je trka između mogućnosti tehnologije i želje „čoveka“ za što boljim kvalitetom „života“.

Računarski sistemi su danas praktično sastavni deo svakodnevnog života. U okviru složenijih funkcija koje izvršavaju, obavljaju i aritmetičke operacije. Poseban značaj ima množenje koje se obavlja u posebnim delovima aritmetičko logičke jedinice, označenim kao množači. Množači vrše množenje brojeva prevedenih u binarni brojni sistem. Problem koji se pojavljuje kod množenja brojeva u binarnom brojnom sistemu proističe iz činjenice da brojevi mogu biti smatrani samo pozitivnim ili mogu biti i negativni pri čemu se za predstavu negativnih brojeva koristi drugi komplement [1]. Znači operacija množenja, množać, mora znati sa kojom predstavom brojeva radi i kakvi su činioči. Prema načinu na koji vrše operacije sa negativnim brojevima postoji nekoliko vrsta množača. Najjednostavniji način, kod kojeg se ne vodi računa o znaku činilaca, je pomoću *Butovog algoritma* (A.D. Booth) [2,3].

Da bi se povećala efikasnost projektovanja, implementacije, realizacije raznih digitalnih sistema današnja tehnologija nudi raznoliki izbor programabilnih komponenti. Njih je moguće lako programirati i privesti potrebnoj nameni. Moguće ih je takođe reprogramirati, za neku drugu namenu ili zbog popravljivanja grešaka. Sam proces projektovanja i implementacije je danas dosta automatizovan korišćenjem raznih programski alata [4], kao što su *VHDL (Very-High-Speed-Integrated-Circuit Hardware Description Language)* ili *Verilog*. Međutim zbog želje da se tehnologija iskoristi do maksimuma i danas je potrebno potrošiti izvesno vreme na prilagođenje postojećoj tehnologiji. Takva istraživanja su uvek aktuelna kao što se vidi iz radova [5,6,7,8].

U radu će biti prezentirano istraživanje trenutnog stanja tehnologije i mogućnost implementacije Butovog algoritma množenja u programabilnim komponentama.

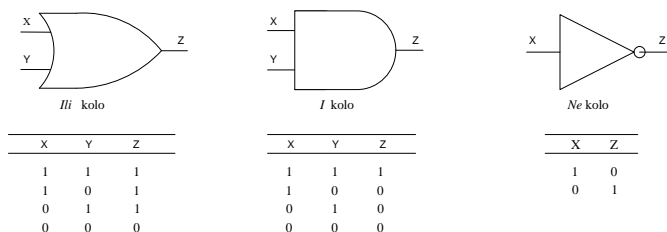
II. Stanje nauke i tehnologije

A. Kombinatorna i sekvencijalna kola

U oblasti digitalne elektronike najčešća, osnovna, podela elektronskih kola je na:

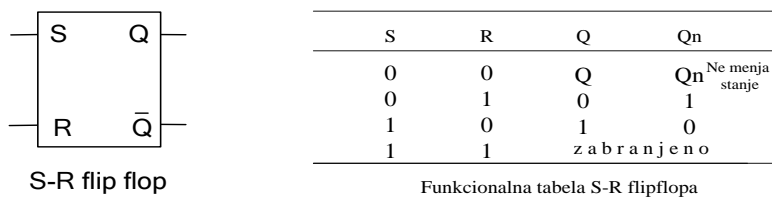
- kombinaciona i
- sekvencijalna.

Izlazni signali kod kombinacionih kola zavise samo od trenutne kombinacije logičkih stanja ulaznih signala i po tome su i dobili ime. Zavisnost ulaznih i izlaznih signala je moguće opisati odgovarajućim logičkim funkcijama. Osnovna kombinaciona kola koja ostvaruju osnovne logičke funkcije poznata su i pod engleskim imenom „gate“, pa se i kod nas koristi termin gejt, bez prevoda. Razlikujemo tri osnovna tipa ovih elemenata, **OR**, **AND** i **NOT** koja ostvaruju funkcije **ILI**, **I** i **NE** Bulove algebre, kao što je prikazano na slici 1.



Slika 1. Osnovna logička kola.
Figure 1. Basic logic circuits.

Kod sekvencijalnih kola izlazni signali zavise od kombinacije trenutnih stanja ulaznih signala ali i od redosleda, sekvence¹, pojave prošlih ulaznih signala. Ova kola „pamte“ prošle vrednosti ulaznih signala, odnosno njihovi izlazi zavise i od istorije događaja na ulazu. Osnovni memorijski element sekvencijalnih kola je *leč*, *flip-flop*, *registar*. U literaturi ne postoji jedinstvena definicija ali se kod nas najviše odomaćio izraz flip-flop za uopšteno bistabilno kolo, kolo sa dva stanja [4]. Flip-flop je element realizovan od, na poseban način, povezanih osnovnih logičkih kola, koja obezbeđuju dva različita, jasno definisana, stanja kola. Prelazak iz jednog u drugo stanje se vrši delovanjem spoljne pobude. Ako spoljna pobuda ne postoji kolo ostaje u jednom od stabilnih stanja koja se po prirodi stvari zovu „zapamćena logička 1“ i „zapamćena logička 0“, kao što je prikazano na slici 2.



Slika 2. SR flip flop
Figure 2. SR flip flop

¹ Engleski sequence-redosled; bez prevoda sekvenca, sekvencijalna

Konstrukcije složenijih flip-floпова su takve da je njihov rad uslovljen, pored ulaznih pobudnih signala, i posebnim taktim signalom. U tom slučaju rad flip-flopa je takav da se stanja, odnosno izlazni signali, dobijaju kao sintezu ulaznih signala samo u trenucima određenim taktim signalom. Takti signal može da traje onoliko dugo koliko je potrebno (10ns, 10 godina...). Kada takti signal nije aktivan uređaj „pamti“ prošlo stanje.

B. Konačne mašine stanja

Trenutne vrednosti memorisanih stanja na pojedinačnom memorijskom elementu su binarne veličine, odnosno postoje samo dva stanja. Kolo koje sadrži n memorijskih elemenata će u tom slučaju moći da memoriše samo 2^n različitih situacija. Kako je n uvek konačan broj takva sekvencijalna kola se zovu i **Konačne mašine stanja** (*finite state machine* ili **FSM**) [4].

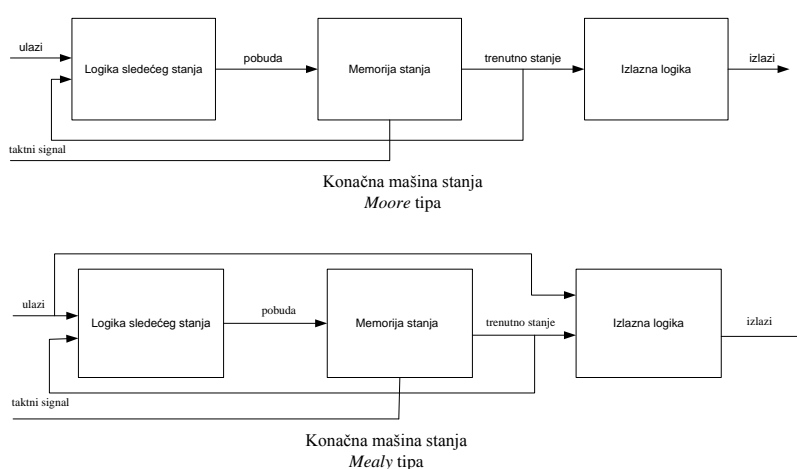
FSM se sastoje od tri osnovne celine:

1. kombinacione mreže koja određuje sledeće stanje (*next-state logic NSL*)
2. memorijskih elemenata (*state memory SM*)
3. kombinacione mreže koja određuje izlazne signale (*output logic OL*)

NSL su kombinaciona logička kola, koja određuju novo stanje FSM u zavisnosti od trenutnog stanja memorijskih elemenata i trenutnih ulaznih signala.

SM je najvažniji deo FSM. To je skup n flip-floпова koji čuvaju informaciju o trenutnom stanju FSM. Kao što je rečeno FSM u tom slučaju može da ima 2^n različitih stanja.

OL je kombinaciona mreža koja daje izlazne signale u zavisnosti od trenutnog stanja memorijskih elemenata. Ako na zavisnost izlaznih signala utiču samo stanja memorijskih elemenata onda se govori o *Moore* FSM, a ako na zavisnost izlaznih signala utiču i trenutna stanja ulaznih signala onda se govori *Mealy* FSM, kao što je prikazano na slici 3 [4].



Slika 3. Osnovne konfiguracije mašina stanja.
Figure 3. Basic configurations for state machines.

Postoji više različitih vrsta flip-floпова koji karakterišu SM kao njeni sastavni delovi. Ukoliko svi flip-floпови u jednoj FSM koriste isti taktни signal onda takvu mašinu zovemo sinhronom FSM [2].

FSM se sastoji iz nekoliko celina koje se najčešće realizuju u programabilnim logičkim kolima. Programabilna logička kola su složene strukture, tako da njihovo povezivanje zahteva, kako poznavanje njihove teorije, tako i poštovanja nekih pravila izrade same FSM.

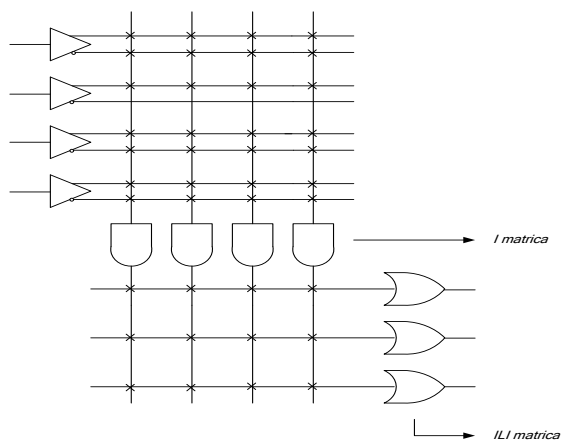
C. Programabilna logička kola

Na tržištu elektronskih komponenti, sa današnjim stepenom tehnološkog razvoja, kao i po unutrašnjoj strukturi programabilne komponente se mogu podeliti u nekoliko grupa [2,3,4]:

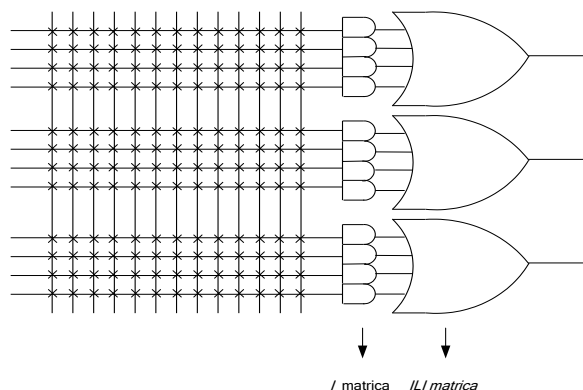
1. *programmable logic array* (PLA)
2. *programmable array logic* (PAL)
3. *programmable logic device* (PLD)
4. *complex PLD* (CPLD)
5. *field programmable gate array* (FPGA)

Sva programabilna logička kola se sastoje od osnovnih elektronskih kola, gejtova, dok se njihova programabilnost ogleda u programljivoj strukturi povezivanja pojedinih ulaznih i unutrašnjih signala. Željena ili nova funkcija dobija se *programiranjem* povezivanja signala, bez ikakvog fizičkog kontakta (zamene ili otklanjanja gejtova), pa su po tome i dobila ime. Kako se svaka logička funkcija može dobiti kombinacijom AND i OR logičkih funkcija, povezivanjem ovakvih kola, u osnovi velikog broja programabilnih komponenti nalaze se programljive AND matrice ili programljive OR matrice ili i jedne i druge. Pod matricom se podrazumeva mogućnost da se više ulaznih signala programiranjem udruži u AND ili OR logičku funkciju.

PLA je jedno od osnovnih programabilnih kola. Sastoji se iz dva dela, kao što je prikazano na slici 4. Prvi deo čini skup AND gejtova. Ulazni signali ovog dela komponente ujedno predstavljaju ulaze u programabilno kolo. Izlazni signali AND matrice su u stvari ulazi signali za drugi deo PLA komponente odnosno OR matrice. Izlazi iz OR gejtova predstavljaju izlaze iz PLA komponente. Jedan isti izlaz iz AND matrice može da bude ulaz za više OR gejtova. Za dobijanje korektnih izlaznih signala programira se i AND i OR matrica. Broj AND i OR gejtova teoretski nije ograničen, a kod konkretne komponente zavisi od toga šta je proizvođač napravio.



Slika 4. PLA programabilna komponenta.
Figure 4. PLA programmable component.



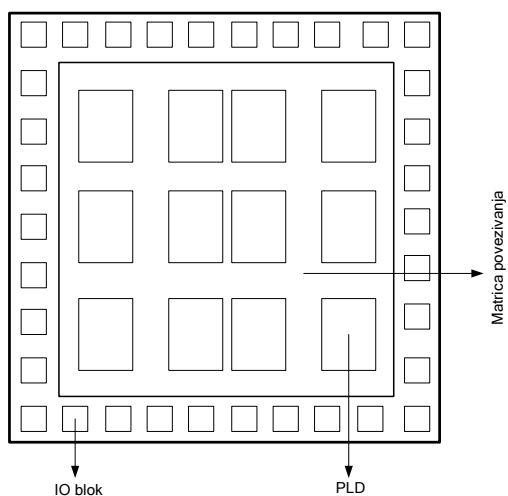
Slika 5. PAL programabilna komponenta.
Figure 5. PAL programmable component.

Kod **PAL** komponente fiksirana je OR matrica, odnosno određeni skup AND gejtova je direktno priključen na OR matricu, kao što je prikazano na slici 5. Time se otklanja mogućnost da jedan isti izlaz iz nekog AND gejta koristi više OR gejtova. U zavisnosti od broja ulaza, broja AND gejtova i izlaza, postoji više vrsta ovih kola.

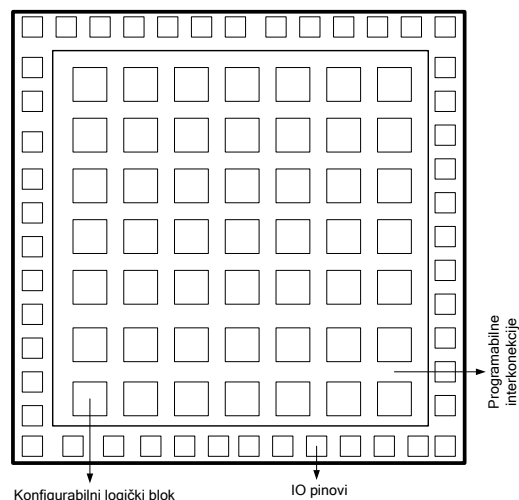
Dodavanjem novih funkcija, sa razvojem tehnologije, kao što je na primer ubacivanje flip-floпова u PAL komponente na izlazu OR matrice, proizvođači su počeli da za ovakve komponente koriste generičko ime *programmable logic device-PLD*. Jasno je da izlazi iz PLD-a mogu postati ulazi u neko novo programabilno kolo. Uobičajen naziv za fizičke ulaze i izlaze na jeste *I/O pins*, ili ulazno izlazni pinovi. Kod složenijih PLD komponenti izlazi iz programabilnih matrica nisu direktno povezani na pinove nego idu preko dodatnih kola koja u tom slučaju čine *I/O block (IOB)*. Ova dodatna kola uglavnom obezbeđuju dobre električne karakteristike signala. Flip-floповi unutar komponenti uglavnom koriste zajednički takt, pa se ovakvi PLD zovu i sinhroni PLD. Ulazi u ove flip-floповe su zapravo izlazi iz OR matrice, što znači da izlazi iz PLD-a više nisu izlazi iz OR matrice nego izlazi iz flip-floпова. Flip-floповi i njihove osobine da pamte prošlo stanje koriste se u FSM. Znači jedan takav PLD ima sve elemente neophodne za realizaciju jedne sinhronne FSM.

Sa razvojem računarskih sistema, razvijala se i potreba za što boljim, većim i efikasnijim programabilnim komponentama. Tako se došlo na ideju o pravljenju CPLD i FPGA [2,3,4]. CPLD predstavlja skup od više međusobno povezanih PLD-ova (koje neki proizvođači zovu i *functional block-FB*) i IOB-a, smeštenih na jedan čip, kao što je prikazano na slici 6. Svi FB su spojeni preko programabilne matrice povezivanja, *switch matrix*. Ovo je veoma važan deo CPLD jer se njime ostvaruje komunikacija između elemenata koji nemaju direktnu fizičku vezu. Najčešća situacija u CPLD komponenti je da su IOB povezani direktno na neke FB a da se potrebne funkcije ostvaruju programiranjem samo unutar FB kao i

njihovim međusobnim povezivanjem, pri čemu su mogućnosti međusobnog povezivanja unapred predefinisane.



Slika 6. CPLD programabilna komponenta.
Figure 6. CPLD programmable component.



Slika 7. FPGA programabilna komponenta.
Figure 7. FPGA programmable component.

FPGA se sastoji od mnoštva programabilnih logičkih blokova (*configurable logic block*-CLB) i IOB-a. CLB su raspoređeni kao polja na šahovskoj tabli, pa se stoga veličina FPGA čipa često izražava preko broja „redova“ i „kolona“. CLB je u funkciji koju vrši u FPGA kolu sličan onoj koju vrši PLD u CPLD, ali se njihove strukture bitno razlikuju. Logičke funkcije se unutar CLB-a ostvaruju ne preko AND i OR matrica nego preko *look up table* dela. Preko matrice povezivanja moguće je povezati bilo koji CLB sa bilo kojim CLB-om ili IOB-om. Kako je ova matrica povezivanja formirana u vidu redova i kolona fizičkih provodnika, mesta na kojima se seku ti provodnici zovu se *programmable switch elements*. Oni su kao skretnice koje usmeravaju signal između CLB.

D. Realizacija konačnih mašina stanja

Nije moguće svaki sistem realizovati konačnom mašinom stanja [2]. Osnovna osobina konačne mašine stanja je konačna dužina memorije. Svaki zahtev koji prevazilazi taj uslov nije moguće realizovati, kao što je na primer, realizovati uređaj koji će prebrojati sve proste brojeve u beskonačnoj sekvenci brojeva.

Ukoliko je određeni zahtevani digitalni sistem uopšte moguće realizovati pomoću mašine(a) stanja neophodno je poznavanje nekoliko osnovnih koraka u izradi same FSM. Realizacija mašina stanja je formalni postupak, proizišao iz teorijskih osnova, i on garantuje ispravno funkcionisanje mašine sa minimalnim brojem elemenata. Kao što se videlo resursi u PLD komponentama su limitirani i mora se o njima voditi računa.

Na samom početku najvažnije je uvideti suštinu problema, prepoznati pod kojim uslovima se izvršava neka radnja, sa kakvom periodičnošću, razlikovati ulazne i izlazne signale. Postavka problema može biti zadata ili deskriptivno, običnim jezikom ili grafičkom predstavom zavisnosti ulaznih i izlaznih signala povezanih sa vremenom izvršavanja, vremenskim dijagramima.

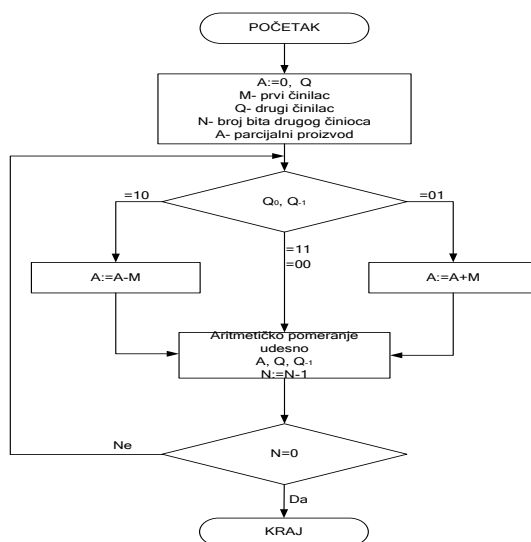
Šablon za realizaciju FSM sastoji se iz nekoliko koraka:

1. *Formiranje „tablice stanja“*. Prilikom formiranja tablice stanja njeni elementi se unose onakvi kakvi su dati u postavci problema. U tablicu se upisuju: signali ulaza, taktni signal kao i signal *inicijalnog stanja*. Inicijalno stanje daje početne vrednosti ulaznih signala. Ovo stanje je važno za početak rada FSM. Iz deskriptivne forme problema zaključuje se u kakvim su odnosima dati i traženi elementi. Radi preglednosti tablice svako moguće stanje proisteklo iz zapisa problema u tablicu se upisuje simboličkim oznakama (kombinacija slova i cifara).
2. Realizacija konačnih mašina stanja podrazumeva njihov korektan rad sa minimalnim brojem elemenata. To znači da sva stanja zapisana u tablicu nisu neophodna za ispravan rad FSM. Postupak svođenja broja stanja FSM- na minimalan broj naziva se *minimizacija*. Minimizaciju omogućavaju tzv. ekvivalentna stanja. To su ona stanja čije se trenutne vrednosti razlikuju ali koja uzrokuju iste izlaze iz FSM i koja za sve kombinacije ulaza daju ista sledeća stanja. Dva ili više ekvivalentnih stanja moguće je zameniti jednim.
3. *State assignment*. Optimalan broj stanja proistekao iz minimizacije upisan je u tablicu simboličkim oznakama. Da bi logička kola unutar FSM mogla registrovati ova stanja, potrebno ih je kodirati. Kodirati stanja znači svakom stanju dodeliti kombinaciju binarnih cifara (0 i 1) i time omogućiti logičkim kolima da prepoznaju ova stanja kao logičke 0 i logičke 1. Ovaj postupak dodele kombinacije binarnih cifara stanjima u tablici naziva se *state assignment*.
4. Formiranje „*transition/output table*“. Nakon dodele stanja sledi zamenjivanje kodiranih stanja u tablicu stanja (zamenjuju se ulazni /dati/ elementi) i sledeći zakonitosti po kojima izlazi zavise od ulaza, formiramo privremenu tablicu izlaza. Ova tablica jasno i konkretno pokazuje kombinaciju izlaznih signala u zavisnosti od konkretne kombinacije ulaznih signala.
5. Važan korak u izradi konačne mašine stanje jeste i odabir vrste flip-flopa za memorijske elemente. Danas se najviše koriste D-flipflopovi.

III. Implementacije konačne mašine stanja za Butov algoritam množenja

U okviru složenih funkcija koje se u digitalnim sistemima implemetiraju najčešće su aritmetičke operacije. Poseban značaj ima množenje jer ovu operaciju u binarnoj, digitalnoj, elektronici nije lako realizovati da se brzo izvršava. Množači, kao specijalizovani delovi su zaduženi da vrše množenje brojeva prevedenih u binarni brojni sistem, i kod njih se često pravi kompromis između složenosti i brzine izvršavanja. Najbrži su paralelni množači ali i najsloženiji odnosno i najskuplji. Često je sastavni deo FPGA kola poseban množač ali su takva kola skupa. U situaciji kada se hoće doći do jeftinog rešenja, a nisu preterani zahtevi za brzinom, može da se napravi kompromis i izabere jeftinija FPGA komponenta bez hardverskog množača, a da se množenje realizuje preko sekvencijalnih, akumulacionih, algoritamskih množača, korišćenjem mašina stanja.

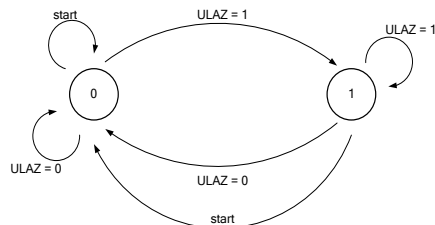
Binarni brojevi mogu biti smatrani samo pozitivnim ali mogu biti i negativni pri čemu se za predstavu negativnih brojeva koristi drugi komplement [1]. Da bi se ispravno uradila operacija množenja, množač, mora znati sa kojom predstavom brojeva radi. Najjednostavniji način, kod kojeg se ne vodi računa o znaku činilaca, je pomoću *Butovog algoritma* (A.D. Booth) [2,3]. Butov algoritam može biti implementiran i kao paralelno i kao akumulaciono množenje. U slučaju kada brzina nije bitna, a potrebno je jeftino rešenje, Butov algoritam predstavljen na slici 8 je osnova po kojoj se izrađuje hardverska struktura u vidu konačne mašine stanja.



Slika 8. Butov algoritam množenja.
Figure 8. Booth's multiplying algorithm.

Množenje pomoću Butovog algoritma se zasniva na posmatranju i pamćenju susednih parova bita drugog činilaca, počevši s desna. Ukoliko su ta dva bita 01, prvi činilac se dodaje na parcijalni proizvod činilaca. Ako su ta dva bita 10, prvi činilac se oduzima od parcijalnog

proizvoda. Posle svake od ovih operacija sabiranja i oduzimanja vrši se aritmetičko pomeranje, tj. poslednji bit sa desne strane se uklanja, i posmatraju se predposlednji i njemu susedni s leva. U slučaju da su biti 11 ili 00 nema operacije, odmah se vrši aritmetičko pomeranje. Ovo se ponavlja dok se ne obrade svi parovi susednih bita drugog činioca. Znači problem realizacije konačne mašine stanja je relativno jednostavan kao što je predstavljeno dijagramom stanja na slici 9. Akcija koju treba izvesti prikazana je na slici 10.

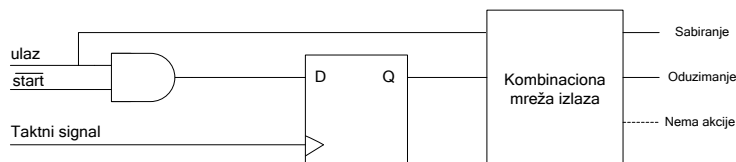


Slika 9. Dijagram stanja mašine.
Figure 9. Machine's state diagram.

Ulaz	Stanje	Izlaz
0	0	Nema akcije
0	1	Sabiranje
1	0	Oduzimanje
1	1	Nema akcije

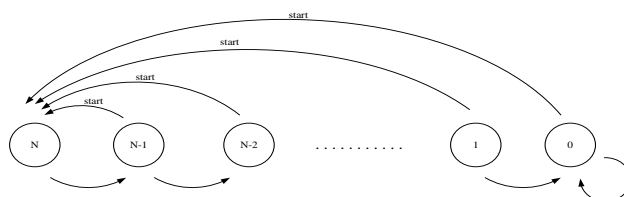
Slika 10. Tablica stanja/izlaza.
Figure 10. Transition/output table.

Unutrašnjost Butovog algoritama se realizuje kao jednostavna FSM sa jednim memorijskim elementom, koji pamti poslednji bit s desne strane drugog činioca. Trenutno stanje mašine predstavlja vrednost zapamćenog bita. Ulazni signal u mašinu je vrednost susednog bita. Ovaj signal zajedno sa signalom upamćene vrednosti bita daje jednu od 4 kombinacije izlaza koje uslovljavaju sledeće operacije množača. Znači ova FSM je *Mealy* tipa. Pored ovoga, ulazni signal mašine biva zapamćen u memoriji i vrednost ulaznog bita postaje novo stanje mašine sve do dolaska novog bita na ulaz mašine.



Slika 11. Realizacija unutrašnje Butove mašine.
Figure 11. Realization of first Booth's machine.

Spoljašost Butovog algoritma, koja vodi računa o broju prolaza, ciklusa, se realizuje kao nova, spoljašnja mašina stanja. Ona u osnovi predstavlja brojač stanja čije je realizacija automatizovana u svim HDL programskim jezicima. Spoljašnja Butova mašina svojim radom određuje kraj rada cele mašine. Kada se iscrpi broj posmatranih bitova ($N=0$), mašina prekida rad do dolaska novog ulaznog signala.



Slika 12. Dijagram stanje spoljašnje Butove mašine.
Figure 12. State diagram of second Booth's machine.

IV. Zaključak

U radu je prezentirano istraživanje trenutnog stanja tehnologije i mogućnost jednostavne implementacije Butovog algoritma množenja u programabilnim komponentama. Akumulacioni Butov algoritam je izabran kao kompromis između brzine rada množača i njegove složenosti. Realizovani delovi množača su male kompleksnosti i kao što se vidi lako ih je implementirati u standardne FPGA komponente. Stvarna implementacija i merenje performansi nije urađeno jer bi to zahtevalo od autora da se upozna i sa odgovorajućim programskim alatima.

Jedan deo istraživanja je sproveden u cilju da se izabere odgovarajuća, pogodna, arhitektura množača. Zbog obimnosti materije u radu nisu prikazane sve analizirane arhitekture. Osnovi rezultati te analize su već poznati i mogu se svesti na sledeće činjenice:

- paralelni množači su brzi ali im je realizacija zahtevna dok su
- akumulacioni množači sporiji ali im je realizacija jednostavnija.

Na osnovu prikazane metodologije i izbora arhitekture množača prilagođene jednostavnim FPGA komponentama autor smatra da bi se i u realnim uslovima implementirano rešenje pokazalo kao dobar odnos složenosti i brzine rada.

Literatura

- [1] D. Živković, M. Popović, "Impulsna i digitalna elektronika", Nauka, Beograd, 1997.
- [2] D.A. Hodges, H.G. Jackson, R.A. Saleh, "Analysis and Design of Digital Integrated Circuit" McGraw-Hill, New York, 2004.
- [3] J.M.Rabaey, A. Chandrakasan, B. Nikolic, "Digital integrated circuits", Prentice Hall International, Inc., New Jersey, 2003.
- [4] John F. Wakerly, "Digital Design – Principles and Practices", Prentice Hall International, Inc., New Jersey, 2001.
- [5] H. Kubátová, T. Hrdý, M. Prokeš, "Problems with the Encoding of the FSM with the Relation to its Implementation by FPGA" *ECI2000 Electronic Computers and Informatics*, International Scientific Conference, Herl'any, pp. 183–188, 2000.
- [6] H. Kubátová, "Implementation of the FSM into FPGA" *Proceedings of the International Workshop on Discrete-Event System Design DESDes' 01*, Oficyna Wydawnicza Politechnika Zielona Góra, Przytok, Poland, pp. 141–146, 2001.
- [7] H. Kubátová, "How to obtain better implementation of the FSM in FPGA" *Proceedings of the 5th IEEE Workshop DDECS 2002*, Brno, Czech Republic, pp. 332–335, 2002.
- [8] Senhadji-Navarro R., Garcia-Vargas I., Jimenez-Moreno G., Civit-Ballcells A., "ROM-based FSM implementation using input multiplexing in FPGA devices" *Electronics Letters* Volume 40, Issue 20, pp 1249 – 1251, 30 Sept. 2004.