Diskretni sistemi

Predavanje XIV

Sadržaj predavanja

- Hardverska implementacija množača
- Hardverska implementacija LVN DS

Hardverska implementacija množača

Hardverska implementacija množača

- Kao i u slučaju implementacije operacije sabiranja, i u slučaju implementacije operacije množenja na raspolaganju nam stoje dva pristupa

- Operaciju množenja možemo implementirati:
  - serijski, gde se tokom većeg broja ciklusa izvodi množenje dva broja, ili
  - paralelno, gde se u jednom ciklusu, u paraleli, vrši množenje dva broja.

Serijski množači

- Kod serijskih množača množenje dva operanda se odvija tokom većeg ili manjeg broja ciklusa
- Kod većine serijskih algoritama za množenje, samo množenje se realizuje računanjem parcijalnih proizvoda i njihovim sumiranjem
- Serijski algoritmi su više ili manje efikasni u zavisnosti od toga da li prilikom računanja ovih parcijalnih proizvoda vode računa da li je računanje tekućeg parcijalnog proizvoda zaista neophodno ili nije
- U praksi je predložen veliki broj različitih algoritama za serijsko (sekvencijalno) množenje, od kojih ćemo na ovom mestu prikazati samo najjednostavnije

Množač baziran na sukcesivnom sabiranju I

- Najjednostavniji algoritam za sekvencijalno izvođenje operacije množenja dva broja predstavlja množač baziran na sukcesivnom sabiranju (*Repetitive Addition Multiplier*)
- Osnovna ideja ovog množača je da se proizvod dva broja izračuna pomoću sukcesivnog sabiranja jednog od činioca, pri čemu je broj potrebnih sabiranja određen vrednošću drugog činioca
- Algoritam je opisan pomoću pseudo-koda prikazanog desno
- Ulazi u algoritam su brojevi *a_in* i *b_in*, čiji proizvod je potrebno izračunati, a izlaz predstavlja vrednost promenljive *r_out*

```
if (a_in=0 or b_in=0) then
      r = 0;

   else
   {
      a = a_in;
      n = b_in;
      r = 0;
      while (n != 0 )
      {
         r = r + a;
         n = n - 1;
      }
   }
   r_out = r;
```

Množač baziran na sukcesivnom sabiranju II

- Osnovni nedostatak prethodnog algoritma je što je trajanje operacije množenja dva broja izuzetno dugačko i što samo trajanje množenja zavisi od vrednosti brojeva koje se množe

- U najgorem slučaju množenje dva *n*-bitna broja pomoću ovog algoritma može da traje 2^n ciklusa, pri čemu se u svakom ciklusu izvršava po jedna operacija sabiranja čije trajanje zavisi od odabrane arhitekture za implementaciju sabirača

- Na primer, ukoliko se za realizaciju operacije sabiranja koristi *ripple-carry* sabirač, onda je trajanje sabiranja u svakom ciklusu proporcionalno sa dužinom digitalne reči pomoću koje je reprezentovan rezultat množenja, a koja iznosi 2*n* bita

- Ukupno trajanje operacije množenja dva *n*-bitna broja u ovom slučaju iznosilo bi

$$T_{RAM} = 2n \cdot 2^n$$

Množač baziran na sukcesivnom sabiranju III

- U slučaju korišćenja *carry-look-ahead* sabirača ukupno trajanje operacije množenja dva *n*-bitna broja iznosilo bi

$$T_{RAM} = (4\log_4(2n)+1)\cdot 2^n$$

- Ono što se može primetiti iz prethodnih izraza je da trajanje operacije množenja korišćenjem množača baziranog na sukcesivnom sabiranju raste eksponencijano sa porastom dužine digitalnih reči pomoću kojih su predstavljene vrednosti ulaznih operanada, *n*!

- Takođe, ukoliko je potrebno množiti označene brojeve, onda je potrebno modifikovati gornji algoritam da vodi računa o znaku ulaznih operanada, kako bi se izračunao pravilan rezultat

- Ove modifikacije će dodatno produžiti trajanje operacije množenja

Množač baziran na pomeranju i sabiranju I

- Značajno poboljšanje u brzini izvođenja operacije množenja može se ostvariti korišćenjem sledećeg algoritma, koji je baziran na pomeranju i sabiranju (*Add-and-Shift Multiplier*)

- Množenje dva 4-bitna broja pomoću pomenutog algoritma prikazano je na slici desno

- Množenje dva 4-bitna broja pomoću „*Add-and-Shift*“ algoritma uključuje:

  1. Pomnoži cifre Operanda 2 (b_3, b_2, b_1, b_0) sa Operandom 1 (A), jednu po jednu, da bi se dobili sledeći proizvodi: b_3*A, b_2*A, b_1*A, b_0*A. Proizvod b_i*A se računa na sledeći način:

     $$b_i*A = (a_3 \cdot b_i, a_2 \cdot b_i, a_1 \cdot b_i, a_0 \cdot b_i)$$

  2. Pomeri proizvod b_i*A za i pozicija u levo

  3. Saberi pomerene članove b_i*A kako bi dobio konačni rezultat

*					a_3	a_2	a_1	a_0	Operand 1
					b_3	b_2	b_1	b_0	Operand 2
					a_3b_0	a_2b_0	a_1b_0	a_0b_0	
				a_3b_1	a_2b_1	a_1b_1	a_0b_1		
			a_3b_2	a_2b_2	a_1b_2	a_0b_2			
+		a_3b_3	a_2b_3	a_1b_3	a_0b_3				
	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	Proizvod

Množač baziran na pomeranju i sabiranju II

- Opisani „*Add-and-Shift*“ algoritam lako se može prevesti u sekvencijalni algoritam
- Možemo procesirati jednu cifru Operanda 2 (b_i) u jednom trenutku i ponoviti postupak za sve cifre Operanda 2 (*B*)
- U svakoj iteraciji, izračunaćemo po jedan proizvod b_i*A, pomeriti ga za *i* mesta u levo, i zatim ga dodati na tekuću vrednost proizvoda
- Obzirom da je b_i zapravo binarna cifra, može imati samo dve vrednosti, 0 ili 1
- Umesto da računamo proizvod b_i*A možemo iskoristiti naredbu grananja da proverimo vrednost cifre b_i i ukoliko je ona jednaka 1 dodati pomerenu vrednost Operanda 1 (*A*) na tekuću vrednost proizvoda
- Pretpostavimo da su vrednosti dva broja koja je potrebno pomnožiti zadate pomoću *n*-bitnih binarnih brojeva, *a_in* i *b_in*
- Tada se „*Add-and-Shift*“ algoritam može opisati pseudo-kodom prikazanim desno. Kao i u slučaju „*Repetitive Addition*“ algoritma, krajnji rezultat množenja smešten je u promenljivoj *r_out*.

```
i = 0;
p = 0;
while (i != n)
{
    if (b_in(i) = 1) then
    {
        p = p + (a_in << i);
    }
    i = i + 1;
}
r_out = p;
```

Množač baziran na pomeranju i sabiranju III

- Vreme potrebno za izvršavanje „*Add-and-Shift*“ algoritma jednako je broju bita koji se koriste za reprezentaciju vrednosti ulaznih operanada, *n*, i što je takođe važno, nezavisno je od konkretnih vrednosti ulaznih operanada

- Kao i u slučaju množača baziranog na sukcesivnom sabiranju, u svakoj iteraciji potrebno je odgovarajuće vreme da se izvrši sabiranje pomerenog ulaznog operanda na tekuću vrednost proizvoda

- Ovo vreme opet zavisi od odabrane arhitekture za realizaciju operacije sabiranja

- Ukoliko se za realizaciju operacije sabiranja koristi *ripple-carry* sabirač onda je trajanje sabiranja u svakom ciklusu proporcionalno sa dužinom digitalne reči pomoću koje je reprezentovan rezultat množenja, a koja iznosi *2n*

- Ukupno trajanje operacije množenja dva *n*-bitna broja u ovom slučaju iznosilo bi

$$T_{SAM} = 2n \cdot n = 2n^2$$

Množač baziran na pomeranju i sabiranju IV

- U slučaju korišćenja *carry-look-ahead* sabirača ukupno trajanje operacije množenja dva *n*-bitna broja iznosilo bi

$$T_{SAM} = (4\log_4(2n)+1)\cdot n$$

- Na osnovu prethodnih izraza može se primetiti da trajanje operacije množenja korišćenjem „*Add-and-Shift*“ algoritma u najgorem slučaju raste kvandratno sa porastom širinom ulaznih operanada *n*, što je znatno bolje od „*Repetitive Addition*“ algoritma, kod kojega trajanje operacije množenja raste eksponencijalno sa *n!*

Množač baziran na pomeranju i sabiranju V

- Prikazani „*Add-and-Shift*“ algoritam u stanju je da korektno množi samo neoznačene brojeve
- Ukoliko je potrebno množiti označene brojeve, pogotovo ako su oni predstavljeni pomoću prvog ili drugog komplementa, gornji algoritam se mora modifikovati kako bi se generisao korektni rezultat
- Ove modifikacije ne menjaju vremensku kompleksnost algoritma, ali ipak dovode do usporenja koje može biti od značaja prilikom konkretne hardverske implementacije
- Iz ovog razloga u praksi se često koristi sledeći algoritam, koji na efikasniji način procesira označene brojeve, što rezultuje boljim performansama u odnosu na „*Add-and-Shift*“ algoritam
- Reč je o **Butovom** algoritmu (*Booth’s algorithm*) za množenje dva označena broja, koji je prikazan u nastavku

Butov algoritam za množenje

- Butov algoritam množi dva označena broja, *a_in* i *b_in*, i u tačno *n* koraka, gde je sa *n* označen broj bita koji se koriste za reprezentaciju vrednosti činioca, i generiše rezultat množenja, smešten u promenljivu *r_out*

- Upoređivanjem Butovog algoritma sa „*Add-and-Shift*“ algoritmom, može se zaključiti da oni imaju istu vremensku kompleksnost, koja u najgorem slučaju raste kvadratno sa porastom širine ulaznih operanada *n*

```
A = a_in&”000000000”;
S = (-a_in)&”000000000”;
P = “00000000”&b_in&’0’;
for (i=0; i<8; i++)
{
    if (P(1:0) = “01”) then
        temp = P+A;
    else if (P(1:0) = “10”) then
        temp = P+S;
    else if (P(1:0) = “00”) then
        temp = P;
    else
        temp = P;
    P = temp >> 1;
    r_out = P(16:1);
}
```

Paralelni množači I

- Operacija množenja sastoji se iz dve osnovne operacije:
  - računanja parcijalnih proizvoda, i
  - njihovog sumiranja.
- Imajući ovo u vidu, postoje dva generalna načina za ubrzanje izvršavanja operacije množenja:
  - smanjenjem broja potrebnih parcijalnih proizvoda koje je potrebno izračunati,
  - ubrzanjem procesa sumiranja izračunatih parcijalnih proizvoda

Paralelni množači II

- Kada se posmatraju paralelni množači, oni se mogu podeliti u dve velike grupe:
  - **množače tipa stabla** (*Tree Multiplier*), kod kojih se svi potrebni parcijalni proizvodi računaju u paraleli, a zatim se korišćenjem brzog više-operandnog sabirača oni sumiraju kako bi se izračunao krajnji rezultat,
  - **mrežne množače** (*Array Multiplier*), koji su sastavljeni od dvodimenzionalne mreže identičnih ćelija pomoću kojih se istovremeno i generišu i akumuliraju novi parcijalni proizvodi. Ova grupa paralelnih množača odlikuje se skraćenim vremenom množenja, koje je postignuto uvećanjem hardverske složenosti.
- Kako se prilikom hardverske implementacije uglavnom ne koriste više-operandni sabirači, u nastavku će biti razmatrani samo mrežni množači

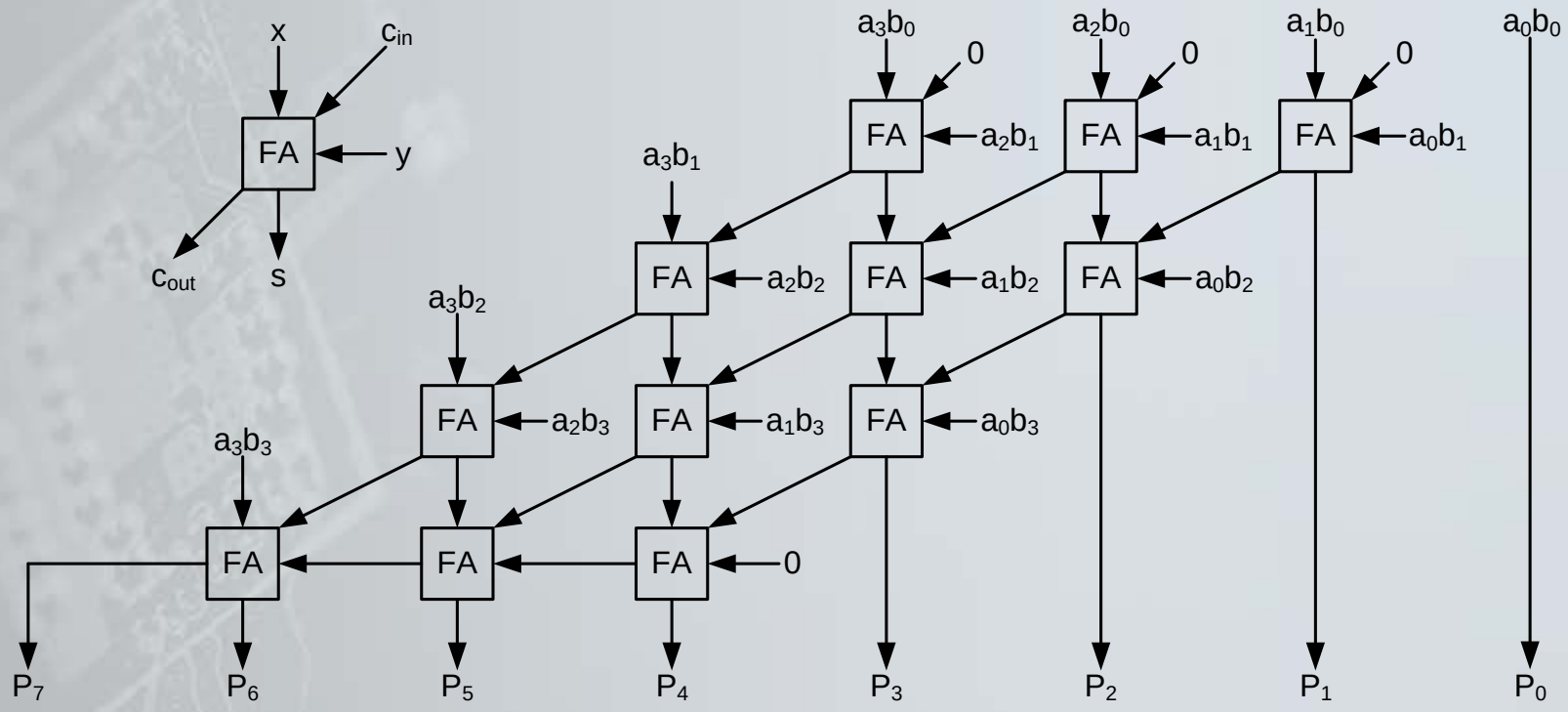
Paralelni množači III

- Način rada mrežnog množača biće ilustrovan na primeru mrežnog množača koji množi dva 4-bitna broja
- Množenje dva 4-bitna broja sastoji se iz generisanja 16 1-bitnih parcijalnih proizvoda $a_i \cdot b_j$, prikazanih na slici desno
- Na osnovu slike desno jasno je da se svih biti proizvoda mogu izračunati u paraleli, sabiranjem parcijalnih sabiraka koji se nalaze u odgovarajućim kolonama, uz odgovarajuće sumiranje bita prenosa generisanog u susednoj koloni
- Za ovu operaciju mogu se iskoristiti standardni puni sabirači

*					a_3	a_2	a_1	a_0	Operand 1
					b_3	b_2	b_1	b_0	Operand 2
					a_3b_0	a_2b_0	a_1b_0	a_0b_0	
				a_3b_1	a_2b_1	a_1b_1	a_0b_1		
			a_3b_2	a_2b_2	a_1b_2	a_0b_2			
+		a_3b_3	a_2b_3	a_1b_3	a_0b_3				
	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	Proizvod

Paralelni množači IV

- Izgled 4-bitnog mrežnog množača, koji koristi ovaj princip generisanja proizvoda, prikazan je na slici dole



Paralelni množači V

- Mrežni množač sa prethodnog slajda u stanju je da korektno množi samo neoznačene binarne brojeve
- U prve tri vrste nema horizontalnog prostiranja signala prenosa
- Horizontalno propagiranje signala prenosa dozvoljeno je samo u poslednjoj vrsti množača
- Na slici sa prethodnog slajda, poslednja vrsta množača zapravo predstavlja jedan *ripple-carry* sabirač, koji se može zameniti nekim bržim sabiračem (na primer *carry-look-ahead* sabiračem) kako bi se poboljšale performanse množača

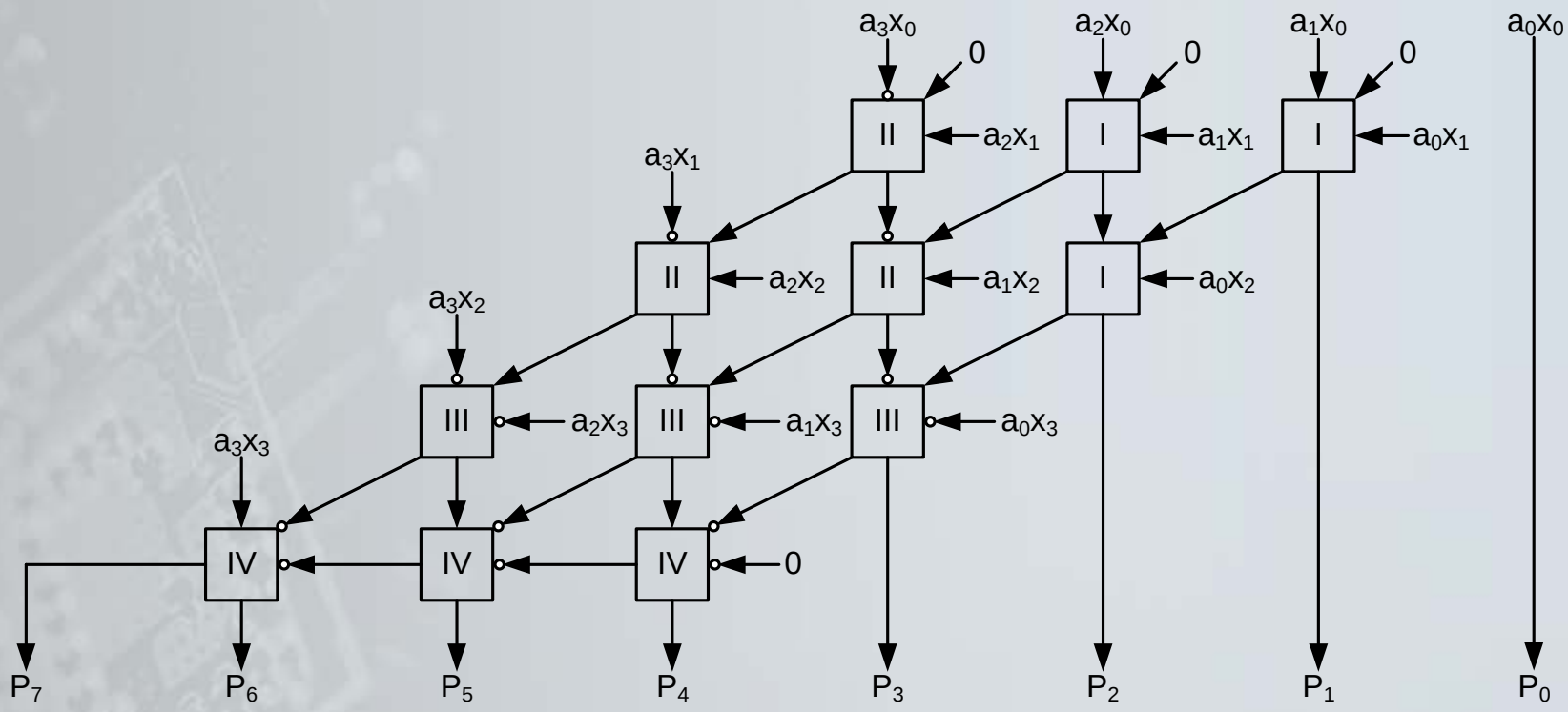
Paralelni množači VI

- Ukoliko je potrebno izvršiti množenje označenih brojeva, predstavljenih pomoću drugog komplementa, mrežni množač sa slajda 18 potrebno je modifikovati na sledeći način

- Pošto bitovi najveće težine, a_3 i b_3, zapravo predstavljaju indikaciju o znaku operanda, kada bitovi a_3 i b_3 imaju vrednost jednaku 1, parcijalni proizvodi koji sadrže ove bitove, na primer a_3b_0 ili a_0b_3, zapravo imaju negativnu težinu, i treba ih oduzeti umesto sabrati prilikom generisanja konačnog rezultata

- Jedan od načina za korektno baratanje sa ovim parcijalnim poizvodima prikazan je na sledećem slajdu

Paralelni množači VII



- Parcijalni proizvodi koji u sumu ulaze sa negativnim znakom označeni su na slici sa kružićem ispred strelice

- Ove parcijalne proizvode potrebno je umesto sabiranja oduzeti

Paralelni množači VIII

- Procenimo trajanje operacije množenja dva *n*-bitna broja pomoću opisanog mrežnog množača
- Analizom blok dijagrama sa slajda 21 može se zaključiti da kritična putanja kroz množač:
  - počinje na ulazima u ćeliju tipa I lociranu u gornjem desnom uglu,
  - prolazi kroz dijagonalni lanac koje formiraju ćelije tipa I i ćelija tipa III,
  - prolazi kroz poslednji horizontalni red, sastavljen od ćelija tipa IV, i
  - završava na izlazima ćelije tipa IV locirane u donjem levom uglu.

Paralelni množači IX

- Kako sve ćelije implementiraju Bulove funkcije koje se mogu realizovati pomoću kombinacionih mreža sa dva nivoa, propagaciono kašnjenje kroz bilo koju ćeliju iznosi $2T_G$ sekundi

- U opštem slučaju, broj ćelija u ovom lancu iznosi *n-2*, gde je sa *n* označena širina ulaznih operanada, tako da ukupno vreme množenja dva n-bitna broja pomoću mrežnog sabirača sa slajda 21 iznosi

$$T_{AM} = 2T_G(n-2)$$

- Na osnovu prethodnog izraza vidimo da vreme množenja dva *n*-bitna broja pomoću mrežnog množača raste linearno sa širinom ulaznih operanada, *n*!

- Ovo je znatno kraće od vremena množenja dva *n*-bitna broja pomoću „*Add-and-Shift*“ množača, koje raste kvadratno sa *n*, ili pomoću „*Repetitive Addition*“ množača, koje raste eksponencijalno sa *n*

- Naravno, ovo ubrzanje u izvođenju operacije množenja „plaćeno“ je povećanom hardverskom složenošću mrežnog množača

Množači sa spregnutim akumulatorom I

- Posmatrajmo izraz za izračunavanje tekuće vrednosti izlaznog signala LVN DS, koji je bio korišćen u predavanju 10, prilikom sinteze IIR LVN DS

$$y(n) = \frac{1}{a_0}\left[\sum_{k=0}^{M} b_k u(n-k) + \sum_{k=1}^{N} (-a_k) y(n-k)\right]$$

- Na osnovu prethodne jednačine možemo primetiti da se trenutna vrednost izlaznog signala LVN DS dobija računanjem dve konačne sume

- U svakoj od suma sumiraju se proizvodi koeficijenata operatorskih polinoma i odbiraka ulaznog, odnosno izlaznog signala

- Ukoliko se u prethodnoj jednačini izvrši normalizacija svih koeficijenata sa članom a_0, onda su to i jedine operacije koje je potrebno izvršiti

Množači sa spregnutim akumulatorom II

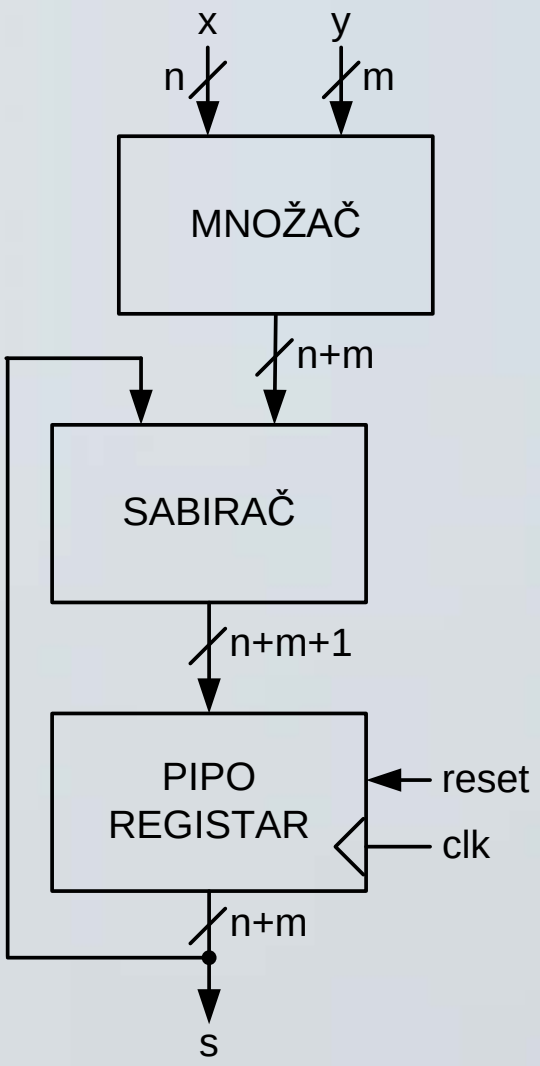
- Iz prethodne analize može videti da je centralna operacija prilikom izračunavanja odziva LVN DS, operacija sumiranja sekvence proizvoda dva broja

- U svakoj od iteracija, na tekuću vrednost sume dodaje se sledeći sabirak, koji se dobija računanjem proizvoda odgovarajućeg koeficijenta operatorskih polinoma i odbirka ulaznog, odnosno izlaznog signala

- Kako ova kompozitna operacija, računanja proizvoda praćenog akumuliranjem, zauzima centralno mesto prilikom računanja tekuće vrednosti izlaznog signala LVN DS, ima i poseban naziv, MAC operacija (*Multiply-Accumulate*)

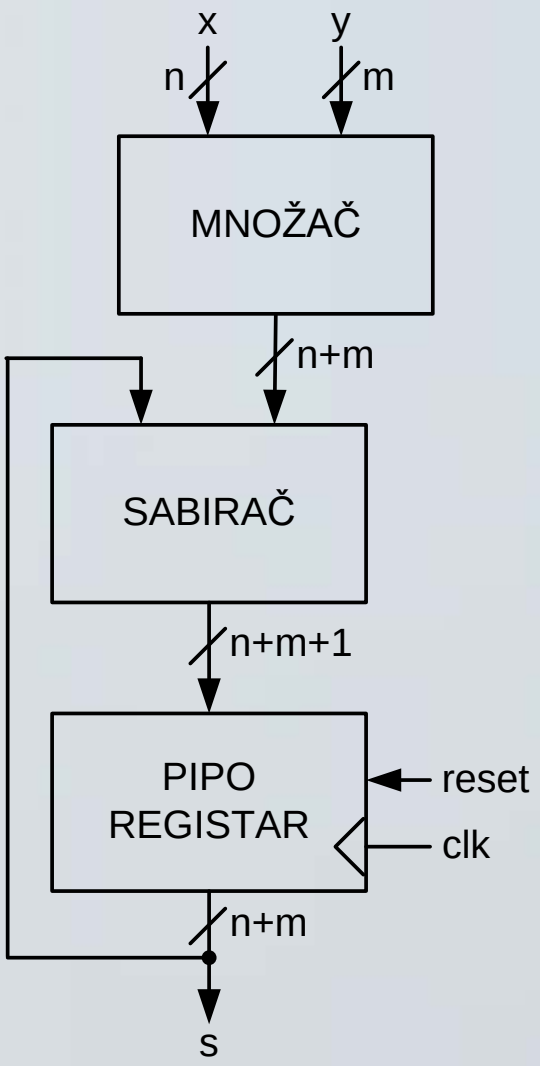
Množači sa spregnutim akumulatorom III

- Hardverski modul, pomoću kojega je moguće realizovati MAC operaciju prikazan je na slici desno
- Kao što se sa slike može videti MAC modul ima dva ulaza, *x* i *y*, koji u opštem slučaju ne moraju biti iste širine, i jedan izlaz, *s*, čija širina se najčešće bira tako da bude jednaka zbiru širina ulaznih operanada
- Pored toga, MAC modulu potrebno je dovesti i odgovarajući sinhronizacioni signal, preko *clk* ulaza, kao i reset signal, preko *reset* ulaza, koji su neophodni za pravilan rad PIPO registra
- Sam MAC modul sastoji se iz jednog množača, koji množi trenutne vrednosti na ulazima u MAC modul, *x* i *y*, i jednog sabirača, koji dodaje vrednost izračunatog proizvoda na tekuću vrednost sume, koja se čuva u PIPO registru



Množači sa spregnutim akumulatorom IV

- Izlaz PIPO registra predstavlja izlaz čitavog sistema, a njegova vrednost odgovara trenutnoj vrednosti akumulirane sume
- Obratite pažnju da se prilikom sumiranja dva (*n*+*m*)-bitna broja, na izlazu sabirača generiše (*n*+*m*+*1*)-bitni rezultat
- Kako je PIPO registar (*n*+*m*)-bitni, mora se vršiti skraćivanje (*n*+*m*+*1*)-bitnog izlaznog signala sabirača na (*n*+*m*)-bitni ulazni signal PIPO registra
- Ovo skraćivanje može se izvesti odsecanjem ili zaokruživanjem, na način koji je diskutovan u predavanju 11, i nije eksplicitno naznačeno na slici desno, ali je neophodno
- U protivnom bi došlo do rasta širine akumulirane sume (*bit growth*) za po jedan bit u svakom ciklusu



Množači sa spregnutim akumulatorom V

- Sam način implementacije MAC modula zavisi od načina na koji su implementirani množač i sabirač
- Obzirom na veliki broj različitih arhitektura za implementaciju sabirača i množača, od kojih su neke diskutovane u ovom predavanju, postoji i veliki broj različitih implementacija MAC modula sa različitim odnosima performanse/veličina
- Ukoliko se koriste serijske arhitekture za implementaciju operacija sabiranja i množenja, dobija se MAC modul koji zahteva minimalne hardverske resurse, ali istovremeno ima i najlošije performanse u pogledu brzine izračunavanja akumulirane sume
- Sa druge strane, ukoliko se za implementaciju operacija sabiranja i množenja koriste paralelne arhitekture, dobijamo MAC modul maksimalnih performansi, ali sa maksimalnim zahtevima u pogledu potrebnih hardverskih resursa za njegovu implementaciju

Množači sa spregnutim akumulatorom VI

- Savremena FPGA kola na sebi već poseduju posebne hardverske blokove pomoću kojih je moguće realizovati MAC modul, bez potrebe za korišćenjem programabilne logike
- Potrebni blokovi za realizaciju MAC modula nalaze se unutar takozvanih DSP blokova
- Prilikom ovakve implementacije nije potrebno razvijati HDL modele sabirača, množača i registara, već je samo potrebno na pravilan način konfigurisati odgovarajući DSP blok
- Često je ovo vrlo jednostavan proces, koji je kompajler za automatsku sintezu hardvera u stanju potpuno samostalno da izvrši, samo na osnovu analize priloženog HDL modela sistema u kome se koriste MAC operacije

Hardverska implementacija LVN DS

Hardverska implementacija LVN DS I

- Prilikom hardverske implementacije LVN DS, u opštem slučaju je potrebno projektovati digitalni sistem koji će biti u stanju da izračuna sledeći izraz

$$y(n) = \frac{1}{a_0}\left[\sum_{k=0}^{M} b_k u(n-k) + \sum_{k=1}^{N} (-a_k) y(n-k)\right]$$

- Prilikom izračunavanja gornjeg izraza potrebno je imati na raspolaganju *M* prethodnih odbiraka ulaznog signala *u(n)*, kao i prethodnih *N* odbiraka izlaznog signala *y(n)*

- Takođe, potrebno je uskladištiti i ukupno *N+M* koeficijenata prenosne funkcije pomoću kojih je definisan LVN DS koji se implementira

Hardverska implementacija LVN DS II

- Da bi se izračunala naredna vrednost izlaznog signala *y(n)*, potrebno je izračunati vrednosti dve sume koje se sastoje iz konačnog broja proizvoda odbiraka ulaznog, odnosno izlaznog signala, sa odgovarajućim koeficijentima prenosne funkcije sistema

- Pored toga, potrebno je i ažurirati memorije u kojima su smešteni odbirci ulaznog, odnosno izlaznog signala, koji su neophodni za izračunavanje naredne vrednosti izlaznog signala

- Sam proces računanja dve gornje sume može se realizovati na razne načine

- Sume se mogu izračunavati serijski, pri čemu se u svakom taktu na tekuću vrednost sume dodaje vrednost narednog proizvoda iz sume

Hardverska implementacija LVN DS III

- Ukoliko se i sume izračunavaju sekvencijalno, jedna za drugom, potrebno je ukupno *N+M+2* taktova da bi se izračunala vrednost narednog odbirka izlaznog signala *y(n)*
- Označimo ovu arhitekturu sa ***A1***
- Međutim, ove sume se mogu izračunavati i u paraleli, pa je u ovom slučaju potrebno *max(N+1, M+1)* taktova za izračunavanje naredne vrednosti izlaznog signala, *y(n)*
- Označimo ovu arhitekturu sa ***A2***

Hardverska implementacija LVN DS IV

- Koristeći ideju paralelizacije, moguće je u istom taktu izračunati veći broj članova iz svake sume u paraleli, i takođe ih sumirati u paraleli

- Opisani postupak poznat je u stručnoj literaturi pod nazivom „**razmotavanje petlje**“ (*loop unrolling*)

- Broj proizvoda iz sume koji se izračunavaju istovremeno određuje „**faktor razmotavanja petlje**“ (*loop unrolling factor*)

- Na primer, ukoliko se primeni tehnika razmotavanja petlje sa faktorom dva, i ako se obe sume izračunavaju u paraleli, potrebno vreme za izračunavanje naredne vrednosti izlaznog signala *y(n)* iznosi

$$\max\left(\left\lceil\frac{N+1}{2}\right\rceil,\left\lceil\frac{M+1}{2}\right\rceil\right)$$

- Označimo ovu arhitekturu sa ***A3***

Hardverska implementacija LVN DS V

- Primenjujući koncept razmotavanja petlje do krajnjih granica, moguće je svaku od dve sume izračunati samo u jednom taktu
- U ovom slučaju vreme potrebno za izračunavanje naredne vrednosti izlaznog signala *y(n)* iznosilo bi samo 1 takt!
- Označimo ovu arhitekturu sa ***A4***
- Opisane hardverske arhitekture odlikuju se raličitim brzinama izračunavanja vrednosti izlaznog signala, ali se one takođe razlikuju i po broju potrebnih hardverskih resursa neophodnih za njihovu realizaciju

Hardverska implementacija LVN DS VI

- Na primer:
  - arhitektura ***A1*** zahteva po jedan sabirač i jedan množač,
  - arhitektura ***A2*** dva sabirača i množača,
  - arhitektura ***A3*** četiri sabirača i množača, dok
  - arhitektura ***A4*** zahteva *N+M* sabirača i *N+M+2* množača.
- Kao što možemo primetiti, ubrzavanje postupka izračunavanja naredne vrednosti izlaznog signala *y(n)* praćeno je i povećanjem broja neophodnih hardverskih resursa
- Ovo je tipičan slučaj koji se uvek sreće prilikom hardverske implementacije algoritama
- Ukoliko želimo bržu implementaciju, moramo biti spremni da je „platimo“ povećanjem potrebnih hardverskih resursa, i obrnuto

Hardverska implementacija LVN DS VII

- Sledeća tabela prikazuje brzinu izračunavanja vrednosti izlaznog signala, kao i potrebne hardverske resurse (u terminima potrebnog broja sabirača i množača), za sve razmatrane arhitekture

Arhitektura	Potrebno vreme za izračunavanje izlaznog signala	Potreban broj sabirača	Potreban broj množača
A1	$N+M+2$	1	1
A2	$\max(N+1, M+1)$	2	2
A3	$\max\left(\left\lceil\frac{N+1}{2}\right\rceil,\left\lceil\frac{M+1}{2}\right\rceil\right)$	4	4
A4	1	$N+M$	$N+M+2$

Hardverska implementacija LVN DS VIII

- Na osnovu prethodne analize može se zaključiti da nam prilikom hardverske implementacije na raspolaganju stoji veliki broj različitih hardverskih arhitektura, koje nude različite odnose performanse/veličina

- Ovo zapravo i predstavlja glavnu prednost hardverske implementacije u odnosu na softversku, jer nam pruža mogućnost izbora optimalne arhitekture za implementaciju LVN DS u zavinosti od zahteva krajnje aplikacije LNV DS

- U slučaju softverske implementacije mogućnost izbora je ograničena, jer je arhitektura procesorskog sistema na kojem će se izvršavati softverska implementacija fiksna, i po pravilu ne pruža veliku mogućnost izbora

Hardverska implementacija LVN DS IX

- U opštem slučaju, prilikom hardverske implementacije LVN DS se mogu implementirati na dva načina:

  - korišćenjem serijskih arhitektura, kod kojih postoji veći ili manji stepen paralelizma prilikom izračunavanja konačnih suma (u zavisnosti od toga da li se koristi *loop unrolling* tehnika i sa kojim faktorom), prisutnih u izrazu za izračunavanje naredne vrednosti izlaznog signala *y(n)*,

  - korišćenjem paralelnih arhitektura, kod kojih se svi članovi iz suma izračunavaju istovremeno.

Serijska implementacija LVN DS I

- Kod serijske implementacije LVN DS, sume iz izraza za računanje naredne vrednosti izlaznog signala *y(n)* računaju se serijski (sekvencijalno), tako što se u toku jednog perioda taktnog signala izračuna vrednost jednog ili više sabiraka (u zavisnosti od toga da li je primenjena *loop unrolling* tehnika)

- Računanje svakog od sabiraka zahteva izvođenje operacije množenja vrednosti odgovarajućeg koeficijenta prenosne karakteristike LVN DS i vrednosti odgovarajućeg odbirka ulaznog, odnosno izlaznog signala

- Iz prethodne analize jasno je da se u svakom periodu taktnog signala zapravo računaju jedan ili više proizvoda i vrši jedno ili više sumiranja

- Ove operacije se mogu jednostavno realizovati korišćenjem jednog ili više MAC modula, koji su opisani ranije

Serijska implementacija LVN DS II

- Pored toga, potrebno je na neki način sačuvati vrednosti koeficijenata prenosne karakteristike, kao i vrednosti prethodnih *M* odbiraka ulaznog signala *u(n)* i *N* vrednosti izlaznog signala *y(n)*

- Vrednosti koeficijenata prenosne karakteristike se ne menjaju tokom vremena, jer razmatramo samo LVN DS, tako da se oni mogu sačuvati u običnim memorijskim bankama

- Međutim, vrednosti odbiraka ulaznog, odnosno izlaznog signala se menjaju sa svakim taktom, jer sistem mora da sačuva *M*, odnosno *N* prethodnih vrednosti ulaznog, odnosno izlaznog signala

- Kako se pomera diskretno vreme, tako se menja *M*, odnosno *N* odbiraka ulaznog i izlaznog signala koje je potrebno imati na raspolaganju

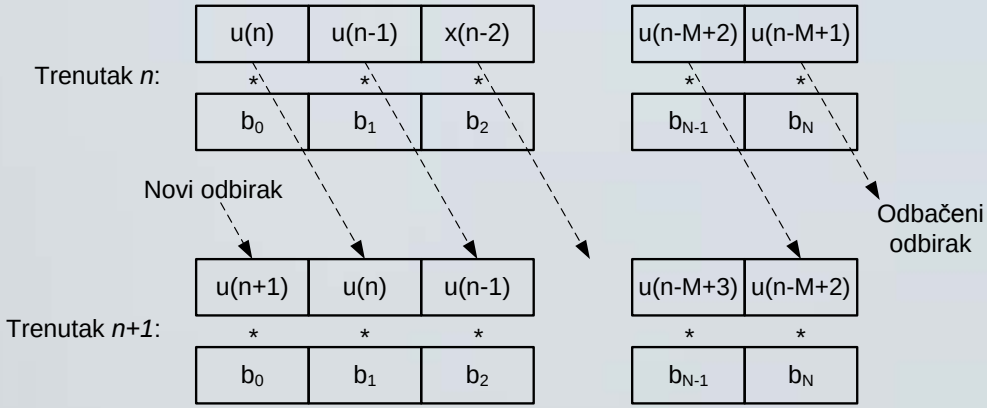
Serijska implementacija LVN DS III

- I ovi odbirci se čuvaju u odgovarajućim memorijama ali su, zbog efikasnijeg pristupa odbircima, ove memorije organizovane u takozvani **cirkularni bafer**

- Slika desno ilustruje način osvežavanja bafera koji služi za smeštanje poslednjih *M* odbiraka ulaznog signala *u(n)*

- U trenutku diskretnog vremena *n*, sadržaj bafera prikazan je gornjom vrstom na slici, i sadrži tekuću i prethodnih *M-1* vrednosti ulaznog signala *u(n)*

- Ove vrednosti odbiraka se množe sa odgovarajućim koeficijentima b_i, koji prikazani u drugoj vrsti od gore na slici



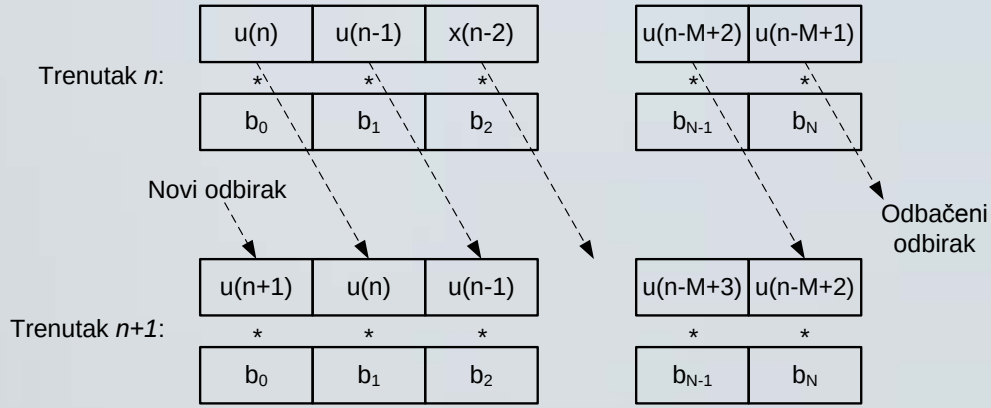
Serijska implementacija LVN DS IV

- Nakon svakog takta, koji odgovara uvećanju diskretnog vremena za 1, potrebno je ažurirati sadržaj signalnog bafera na sledeći način

- Najstariji odbirak, *u(n-M+1)*, ispada iz bafera, preostali odbirci (*u(n)* do *u(n-M+2)*) se pomeraju za jedno mesto u desno, a na prvo mesto u bafer se upisuje novi odbirak ulaznog signala, *u(n+1)*

- Zatim se ovih novih M odbiraka ulaznog signala množi sa istim vrednostima prenosne funkcije, b_i

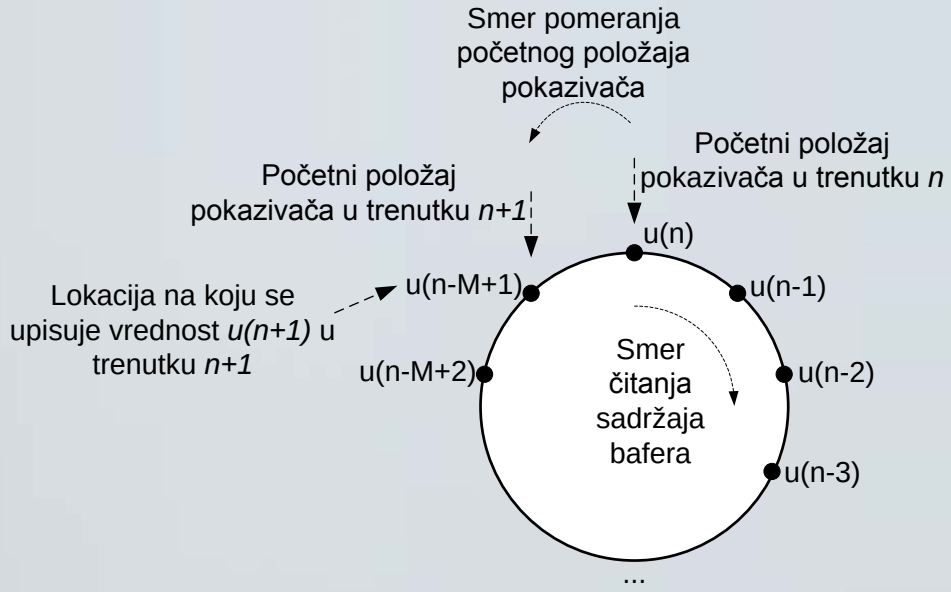
- Obratite pažnju da nije potrebno menjati redosled, niti vrednosti, koeficijenata b_i, tako da mogu biti smešteni u običnu memoriju



Serijska implementacija LVN DS V

- Kako bi se izbegla potreba za premeštanjem *M-1* odbiraka ulaznog signala za jedno mesto u desno u svakom taktu, uobičajeno je da se signalni bafer realizuje u vidu cirkularnog bafera, prikazanog na slici desno

- Umesto pomeranja odbiraka ulaznog signala za jedno mesto u desno i držanja početne adrese bafera fiksnom, kao što je bio slučaj na slici sa prethodnog slajda, u cirkularnom baferu odbirci se ne pomeraju, već se početna adresa pomera za jedno mesto u smeru suprotnom od kretanja kazaljke na satu



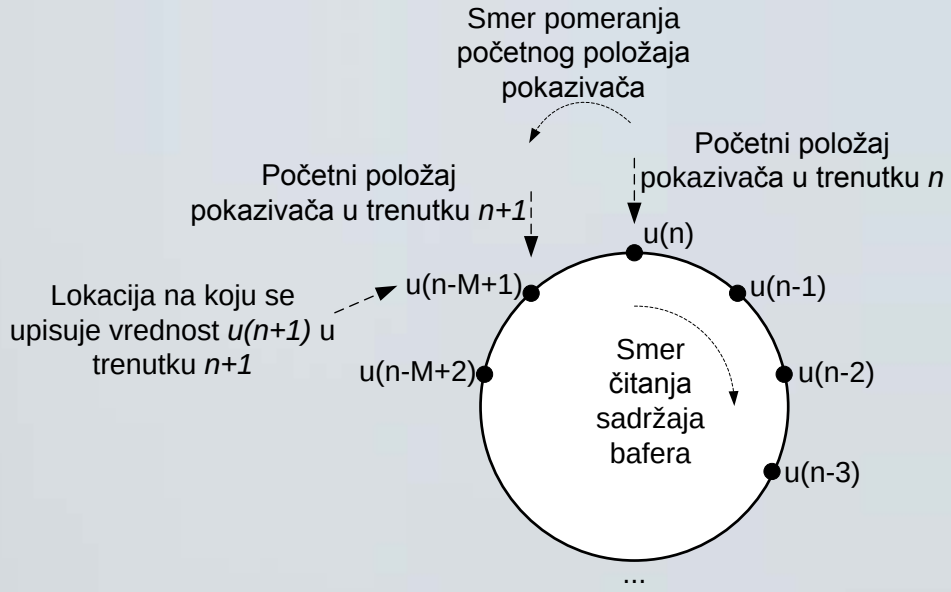
Serijska implementacija LVN DS VI

- U trenutku diskretnog vremena *n*, početna adresa pokazuje na lokaciju u kojoj je smešten odbirak *u(n)*, a odbirci se čitaju iz bafera u smeru kazaljke na satu, tako da se na izlazu bafera dobija sledeća sekvenca odbiraka ulaznog signala

  u(n), u(n-1), ..., u(n-M+2), u(n-M+1).

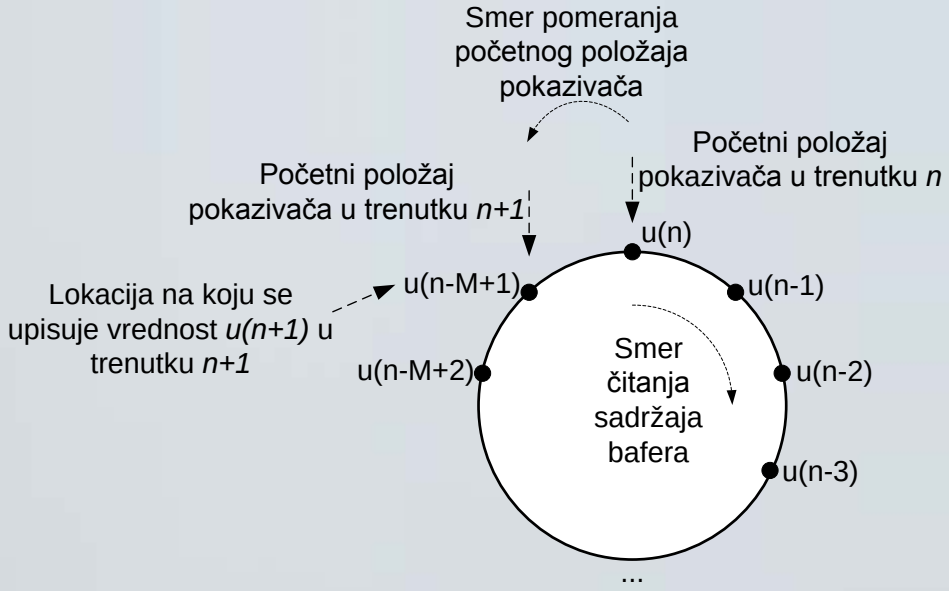
- Vidimo da ova sekvenca upravo odgovara potrebnoj sekvenci odbiraka ulaznog signala koju je potrebno pomnožiti sa koeficijentima prenosne funkcije sistema b_i, kao što je prikazano na gornjem delu slike sa slajda 60

- U trenutku dikretnog vremena *n+1*, potrebno je smestiti vrednost sledećeg odbirka ulaznog signala, *u(n+1)*, u bafer



Serijska implementacija LVN DS VII

- Ova vrednost treba da bude smeštena unutar cirkularnog bafera na jednoj poziciji u levo od pozicije na kojoj je smeštena vrednost ulaznog signala *u(n)*
- Ovo se lako postiže, pomerajući adresni pokazivač za jedno mesto u levo
- Nakon upisa vrednosti *u(n+1)*, koja će biti upisana na lokaciju u kojoj je prethodno bila smeštena vrednost ulaznog signala *u(n-M+1)*, adresni pokazivač i dalje je pozicioniran tako da pokazuje na memorijsku lokaciju u kojoj se sada nalazi vrednost ulaznog signala *u(n+1)*



Serijska implementacija LVN DS VIII

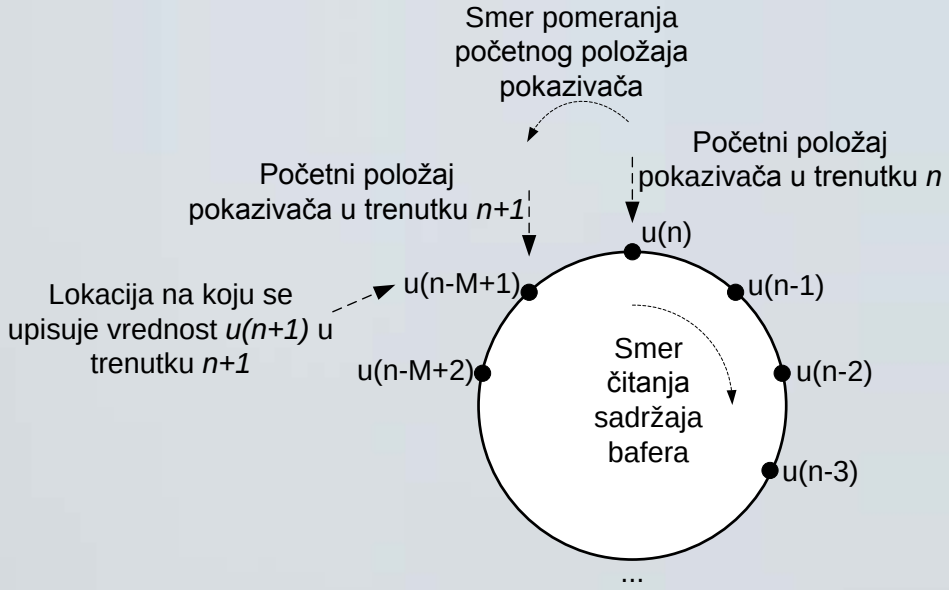
- Obilaženjem cikularnog bafera u smeru kazaljke na satu, počevši od lokacije na kojoj je smeštena vrednost *u(n+1)*, tokom narednih *M* taktova na izlazu cirkularnog bafera pojavljuje se sekvenca

  u(n+1), u(n), u(n-1), ..., u(n-M+2), u(n-M+1)

- Ovo je upravo sekvenca koja je potrebna za računanje sume u trenutku diskretnog vremena *n+1*, kao što se može videti na donjem delu slike sa slajda 60

- Opisani postupak korišćenja cirkularnog bafera prikazuje njegovu najveću prednost

- Svako ažuriranje cirkularnog bafera zahteva samo ažuriranje početne pozicije adresnog pokazivača, bez fizičkog premeštanja odbiraka koji se nalaze smešteni unutar bafera



Serijska implementacija LVN DS IX

- Nakon što smo opisali princip rada cirkularnog bafera, koji će biti iskorišćen za smeštanje i ažuriranje poslednjih *M* odbiraka ulaznog, odnosno *N* odbiraka izlaznog signala, možemo pristupiti opisu strukture hardverskog sistema za serijsku implementaciju LVN DS

- Radi jednostavnosti, u nastavku će biti opisana samo jedna od mogućih struktura digitalnog sistema koji se može iskoristiti za hardversku implementaciju FIR LVN DS

- U slučaju FIR LVN DS, opšta jednačina koja opisuje način na koji se računa sledeći odbirak izlaznog signala *y(n)* svodi se na sledeći oblik

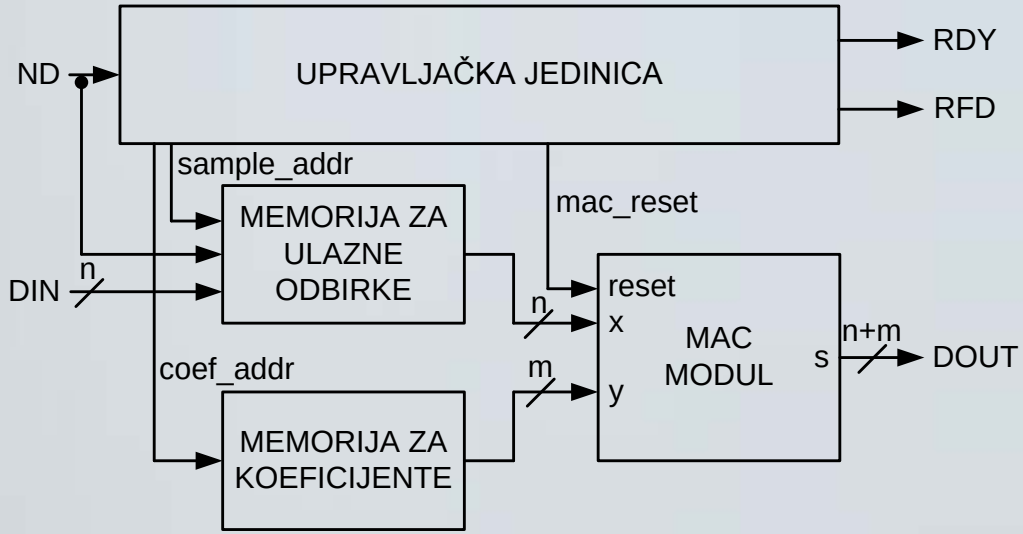
$$y(n) = \sum_{k=0}^{M} b_k u(n-k)$$

Serijska implementacija LVN DS X

- U prethodnoj jednačini takođe smo pretpostavili da je vrednost koeficijenta a_0 jednaka 1
- U slučaju kada je vrednost koeficijenta a_0 različita od 1, potrebno je izvršiti normalizaciju koeficijenata b_i, deleći svaki sa a_0
- Vidimo da se u slučaju FIR LVN DS, izraz za računanje vrednosti narednog odbirka izlaznog signala *y(n)* svodi na računanje samo jedne sume
- Najjednostavniji hardverski sistem za serijsku implementaciju FIR LVN DS u svakom taktu računa i akumulira samo jednu vrednost sabirka iz konačne sume, odnosno ne koristi *loop unroling* tehniku
- Struktura ovakvog sistema prikazana je na sledećoj slici

Serijska implementacija LVN DS XI

- Sistem sa slike desno sastoji se iz jednog MAC modula, koji u svakom taktu računa vrednost narednog sabirka iz sume i akumulira ga na tekuću vrednost sume
- Koeficijenti prenosne funkcije b_i nalaze se smešteni unutar memorije za koeficijente
- Poslednjih *M* odbiraka ulaznog signala smešteni su unutar memorije za ulazne odbirke, koja je organizovana u formi cirkularnog bafera
- Čitavim sistemom upravlja upravljačka jedinica, koja je zadužena za sledeće funkcije



Serijska implementacija LVN DS XII

1. Pravilno generisanje adresa za pristup podacima unutar memorije za koeficijente (*coef_addr* magistrala sa prethodne slike)

2. Pravilno generisanje adresa za pristup podacima unutar memorije za ulazne odbirke (*sample_addr* magistrala sa prethodne slike)

3. Upisivanje naredne vrednosti ulaznog signala u pravilnim trenucima, na pravo mesto, unutar cirkularnog bafera za ulazne odbirke

   Naredna vrednost ulaznog signala prosleđuje se sistemu preko *DIN* ulaznog porta, a trenutak kada je ona validna i kada ju je potrebno upisati u cirkularni bafer signalizira se pomoću *ND* ulaznog porta

   Nakon upisa naredne vrednost ulaznog signala, sistem automatski započinje proces računanja naredne vrednosti izlaznog signala

Serijska implementacija LVN DS XIII

4. Resetovanje MAC modula svaki put kada započinje ciklus računanja naredne vrednosti izlaznog signala (pomoću *mac_reset* signala sa slike sa slajda 50)

5. Generisanje odgovarajućih statusnih signala:

   - *RDY* signala – koji predstavlja indikaciju da je sistem izračunao narednu vrednost izlaznog signala i da se ona nalazi na *DOUT* izlazu

   - *RFD* signala – koji predstavlja indikaciju da je sistem spreman da prihvati vrednost narednog odbirka ulaznog signala i započne računanje naredne vrednosti izlaznog signala

Paralelna implementacija LVN DS I

- U slučaju paralelne implementacija LVN DS zapravo se postojeće dve petlje u izrazu za računanje naredne vrednosti izlaznog signala *y(n)* maksimalno raspliću, tako da se u svakom taktu računaju vrednosti svih sabiraka koji figurišu unutar suma i vrši njihovo sumiranje

- Ovakav sistem može se realizovati na različite načine, od kojih su najčešće korišćeni diskutovani u predavanju 10, posvećenom sintezi LVN DS

- Prilikom hardverske implementacije, odabrana forma za realizaciju LVN DS se prevodi u odgovarajući digitalni sistem

- Prvi korak je da se izaberu širine digitalnih reči koje će biti korišćene za reprezentaciju signala unutar LVN DS koji se implementira

- O ovome je detaljno bilo reči u predavanjima 11 i 12

Paralelna implementacija LVN DS II

- Nakon što je izabran format za reprezentaciju svih signala unutar LVN DS, potrebno je izabrati hardverske arhitekture koje će biti korišćene za implementaciju tri osnovna elementa LVN DS, sabirača, množača i pomerača
- Obratite pažnju da je čak i prilikom paralelne implementacije, i dalje moguće koristiti serijske arhitekture za implementaciju operacija sabiranja i množenja, diskutovane ranije u ovom predavanju
- Na ovaj način će se svaka od operacija sabiranja ili množenja izvoditi serijski, tokom više taktova, ali će se sve operacije sabiranja i množenja koje postoje unutar odbrane forme za realizaciju LVN DS i dalje izvoditi u paraleli
- Kao što vidimo, i prilikom paralelne hardverske implementacije LVN DS, i dalje imamo mogućnost da biramo između različitih paralelnih arhitektura koje se međusobno razlikuju po brzini izračunavanja vrednosti izlaznog signala iz sistema i količini potrebnih hardverskih resursa za njihovu implementaciju

Paralelna implementacija LVN DS III

- Kao ilustraciju hardverske implementacije LVN DS korišćenjem paralelne arhitekture, biće predstavljen digitalni sistem koji je u stanju da implementira FIR LVN DS čija relacija ulaz-izlaz ima dobro poznati oblik

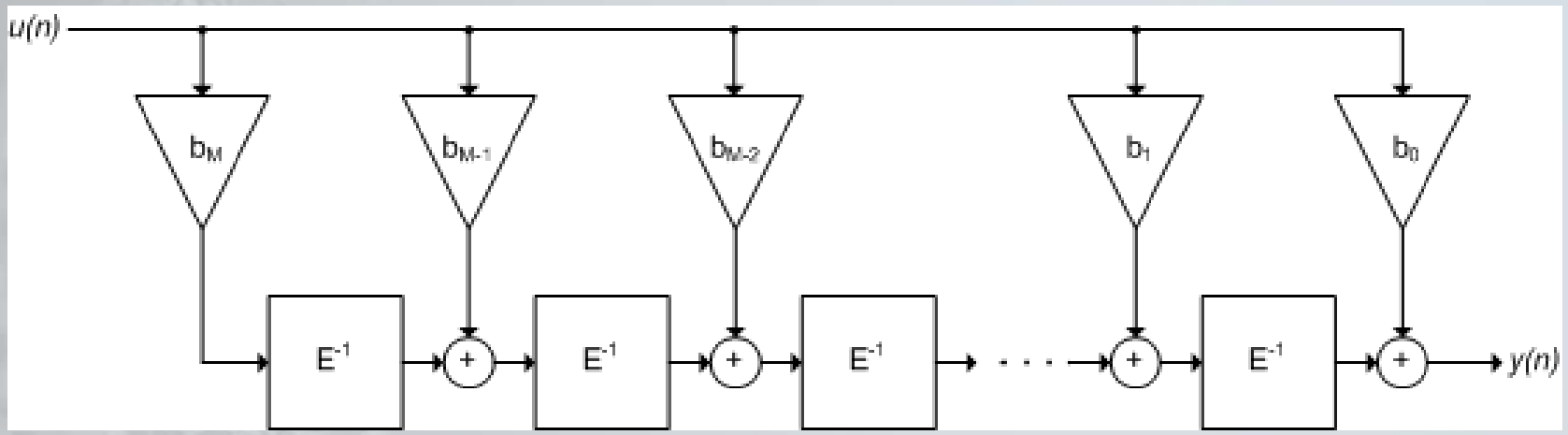
$$y(n) = \sum_{k=0}^{M} b_k u(n-k)$$

- Kao što je rečeno u predavanju 10, FIR LVN DS može se realizovati korišćenjem većeg broja različitih struktura, od kojih su najčešće korišćene direktna forma, transponovana direktna forma i kaskadna forma

- Na ovom mestu projektovaćemo digitalni sistem koji je baziran na transponovanoj direktnoj formi realizacije FIR LVN DS

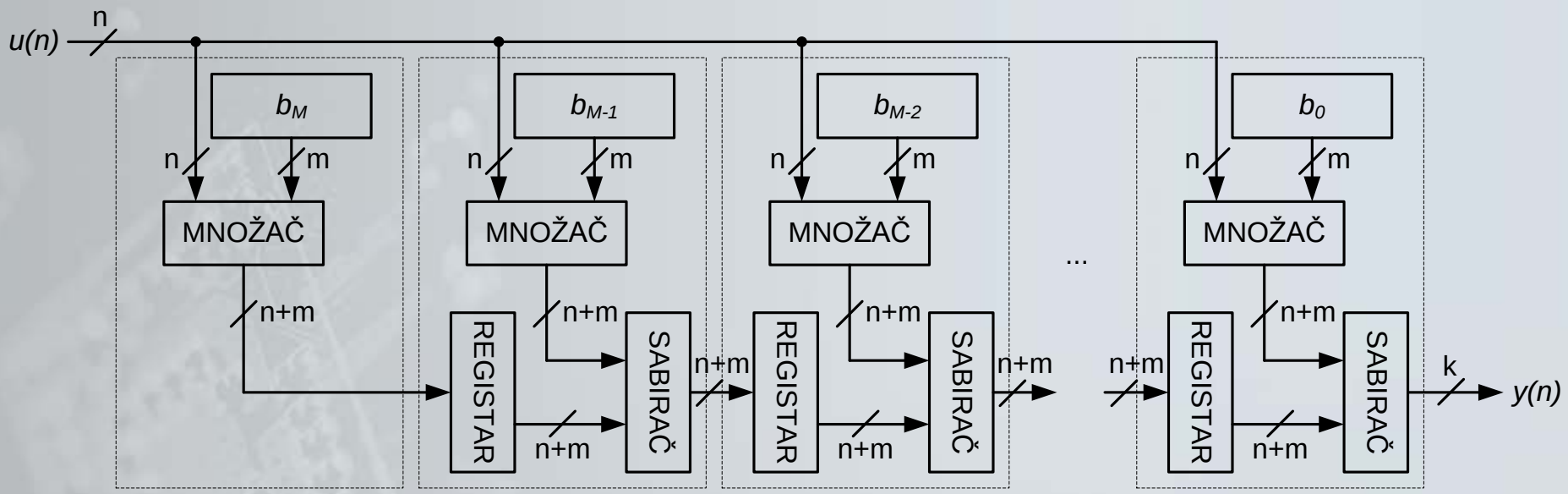
Paralelna implementacija LVN DS IV

- Kao što je u predavanju 10 detaljno objašnjeno, FIR LVN DS *M*-tog reda, realizovan pomoću transponovane kaskadne forme ima strukturu prikazanu na slici dole



- Zamenom svakog sabirača, množača i pomerača sa gornje slike odabranim hardverskim arhitekturama za realizaciju ovih operacija, dobijamo blok dijagram digitalnog sistema koji realizuje posmatrani FIR LVN DS, prikazan na sledećoj slici

Paralelna implementacija LVN DS V

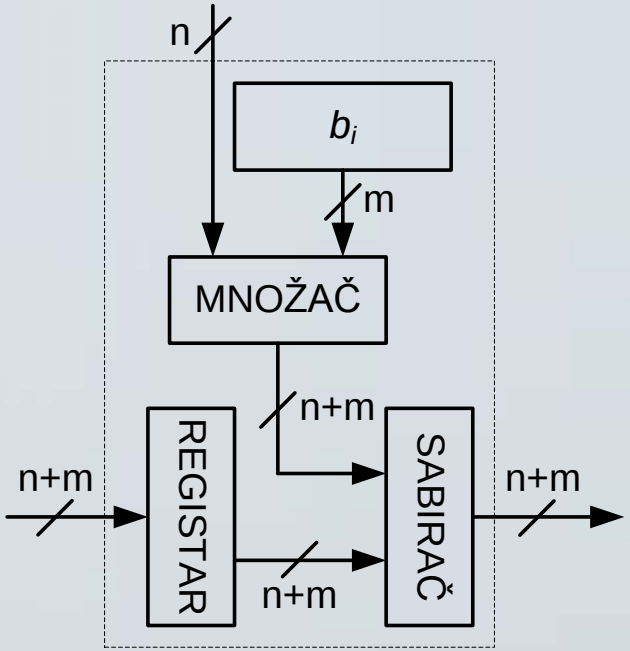


- Kao što se sa gornje slike može videti, struktura digitalnog sistema za implementaciju LVN DS, baziranog na korišćenju transponovane direktne forme, je izrazito regularna

- Čitav sistem sastoji se iz *M+1* blokova, od kojih su svi, osim prvog, identični

Paralelna implementacija LVN DS VI

- Struktura identičnih blokova prikazana je na slici desno
- Svaki od blokova sastoji se iz jednog množača, jednog sabirača i dva registra
- Jedan registar služi za čuvanje vrednosti koeficijenta prenosne funkcije sistema, b_i, koji je asociran posmatranom bloku, dok drugi registar služi za čuvanje međurezultata sumiranja proizvoda
- U opštem slučaju širine ulaza u ova dva registra, kao i širine ulaza u množač i sabirač mogu biti različite, kao što je i prikazano na slici desno
- Kao i ranije, da ne bi došlo do porasta broja bitova koji se koriste za reprezentaciju vrednosti izlaznog signala sabirača, potrebno je izvršiti njegovo skraćivanje, koristeći postupak odsecanja ili zaokruživanja



Paralelna implementacija LVN DS VII

- Po pravilu, za reprezentaciju unutrašnjih signala digitalnog sistema koji implementira odgovarajući diskretni sistem koristi se veći broj bita nego za reprezentaciju vrednosti izlaznog signala čitavog sistema, *y(n)*
- Ovo je jasno naznačeno na slici čitavog sistema, gde je sa *k* označen broj bitova koji se koristi za reprezentaciju vrednosti izlaznog signala *y(n)*
- Postojeća modularnost omogućava da se na vrlo koncizan način napiše odgovarajući VHDL model digitalnog sistema koji zapravo implementira LVN DS, sintetizovan korišćenjem transponovane direktne forme