Diskretni sistemi

Predavanje XIII

Sadržaj predavanja

- Hardverska implementacija diskretnih sistema
- Hardverska implementacija pomerača
- Hardverska implementacija sabirača

Hardverska implementacija diskretnih sistema

Hardverska implementacija diskretnih sistema I

- Nakon što smo projektovali diskretni sistem sa željeni karakteristikama, odabrali formu pomoću koje ćemo ga realizovati i odredili format brojeva pomoću kojega ćemo reprezentovati vrednosti diskretnih signala u sistemu možemo pristupiti poslednjem koraku, implementaciji diskretnog sistema

- Kao što je ranije već rečeno, implementacija diskretnih sistema može se izvršiti na dva načina:

  - softverski i

  - hardverski

Hardverska implementacija diskretnih sistema II

- U slučaju **softverske implementacije** diskretni sistem se implementira pisanjem odgovarajućeg programa, u nekom od standardnih programskih jezika (asembler, neki od viših programskih jezika kao što su C++, Pascal, itd.), pri čemu se taj program izvršava na:
  - računaru opšte namene (PC),
  - procesoru za digitalnu obradu signala (DSP), ili
  - usko specijalizovanim procesorima (konvolucioni procesori, vektorski procesori).
- Ovo je najjednostavniji način implementacije diskretnog sistea, ali diskretni sistem implementiran na ovaj način imaju ograničenu brzinu rada, koja je posledica sekvencijalne implementacije na nekom od procesora
- U slučaju projektovanja diskretnih sistema sa visokom učestanošću odabiranja, softverska implementacija najverovatnije neće biti moguća

Hardverska implementacija diskretnih sistema III

- Na primer, pretpostavimo da je potrebno projektovati digitalni pojasni filtar koji će služiti za selekciju željenog FM radio kanala unutar prijemnika
- Kako se FM radio kanali nalaze u opsegu od 87,5 MHz do 108,0 MHz, pretpostavimo da potrebna učestanost odabiranja u diskretnom sistemu koji projektujemo iznosi 100 MHz
- Takođe pretpostavimo da je projektovani filtar FIR tipa, reda 50
- Procenimo na kolikoj učestanosti mora raditi neki hipotetički procesor koji bi bio u stanju da filtrira FM radio signal pomoću ovako projektovanog digitalnog filtra
- Kako se pomoću softerske implementacije diskretni sistem implementira sekvencijalno, u svakoj periodi odabiranja potrebno je izvršiti 50 operacija množenja i sabiranja, odnosno takozvanih MAC (*Multiply Accumuate*) operacija

Hardverska implementacija diskretnih sistema IV

- Potreban broj MAC operacija koje hipotetički procesor mora da izvrši u jednoj sekundi onda iznosi 50*100 MHz = 5 GMAC/s
- Neka je hipotetički procesor u stanju da u svakom taktu izvrši jednu MAC operaciju, što je vrlo optimistična pretpostavka
- Pod ovom pretpostavkom naš hipotetički procesor morao bi raditi na minimalno 5 GHz kako bi bio u stanju da izvrši potreban broj računskih operacija u toku jedne sekunde
- Danas na tržištu ne postoji ovakav procesor
- Čak ako bi i postojao, on bi bio u potpunosti zaokupljen opisanim procesom filtriranja i ne bi imao vremena za izvođenje bilo kojih drugih operacija

Hardverska implementacija diskretnih sistema V

- Drugi način implementacije diskretnih sistema podrazumeva projektovanje specijalizovanih digitalnih sistema pomoću kojih će biti realizovan željeni diskretni sistem

- Ovaj drugi način naziva se **hardverska implementacija**

- Generalno hardverska implementacija može se izvršiti na dva načina, projektovanjem VLSI integrisanih kola posebne namene, što predstavlja jako skupu opciju, ili korišćenjem FPGA programibilnih kola, koja predstavljaju ekonomičnu varijantu za male i srednje serije

- Hardverskom implementacijom diskretnih sistema mogu se postići najbolje performanse, ali često na račun veće cene realizacije

- U nekim aplikacijama, kada ne možemo zadovoljiti postavljene kriterijume pomoću softverske implementacije, ovo je jedina preostala mogućnost

Hardverska implementacija diskretnih sistema VI

- Procenimo sada kolika bi bila potrebna učestanost rada digitalnog sistema pomoću kojega je implementiran pojasni filtar za selekciju FM kanala iz prethodnog primera

- Kako ovaj put FIR filtar 50-tog reda možemo implementirati u paraleli, gde se svaka od 50 operacija množenja i sabiranja izvršava istovremeno, potrebna učestanost rada ovaj put je jednaka brzini pristizanja novih odbiraka, koja iznosi 100 MHz

- Projektovanje digitalnih elektronskih kola koja treba da rade na učestanosti od 100 MHz danas ne predstavlja naročito veliki izazov i standardno je izvodljivo korišćenjem komercijalno dostupnih FPGA komponenti, o VLSI kolima da i ne govorimo

Hardverska implementacija diskretnih sistema VII

- Prilikom hardverske implementacije potrebno je projektovati digitalne kombinacione i sekvencijalne mreže pomoću kojih je moguće realizovati osnovne gradivne blokove svakog linearnog, vremenski nepromenljivog diskretnog sistema:
  - pomerače,
  - sabirače, i
  - množače.
- Nakon što su ove mreže projektovane, njihovim povezivanjem, na osnovu blok dijagrama dobijenog kao rezultat sinteze diskretnog sistema, dobija se kompletan digitalni elektronski sistem koji implementira željeni diskretni sistem
- Sam proces projektovanja u praksi se svodi na pisanje odgovarajućih HDL modela i korišćenje odgovarajućih alata za automatsku sinzetu hardvera, kako bi se na kraju stiglo do željene hardverske implementacije

Hardverska implementacija pomerača

Hardverska implementacija pomerača I

- Kao što je ranije već rečeno, uloga pomerača unutar diskretnog sistema je da vremenski pomeri diskretni signal, unapred ili unazad, odnosno u levo ili u desno, za željeni broj mesta

- Kako je fizički nemoguće realizovati pomerače koji pomeraju diskretni signal unapred, prilikom hardverske implementacije uvek se koristi prva verzija pomerača

- Na osnovu gornjeg opisa ponašanja pomerača, u slučaju kada se vrši pomeranje diskretnog signala za jedno mesto unazad što inače najčešći slučaj, jasno je da se on može realizovati pomoću običnog *n*-bitnog registra sa paralelnim upisom i paralelnim čitanjem (PIPO)

Hardverska implementacija pomerača II

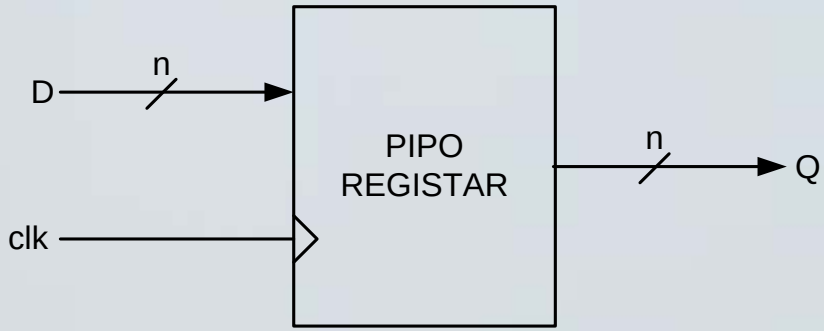
- Širina ulaznog i izlaznog porta *n*, jednaka je broju bitova koji se koriste za reprezentaciju diskretnog signala koji je doveden na ulaz pomerača
- Ovaj broj bitova određen je na takav način da greške usled konačne preciznosti predstavljanja realnih brojeva pomoću mašinskih budu ispod prihvatljive granice
- O ovoj problematici je bilo reči u predavanjima 11 i 12
- Potrebno je napomenuti da učestanost globalnog sinhronizacionog signala mora biti odabrana na takav način da bude jednaka učestanosti odabiranja za koju je projektovan dati diskretni sistem

Hardverska implementacija pomerača III

- Osnovni interfejs *n*-bitnog registra sa paralelnim upisom i čitanjem prikazan je na slici desno

- N bitni registar sa paralelnim upisom i čitanjem ima dva ulazna porta:
  - *D* ulaz za podatke, širine *n* bita,
  - *clk* ulaz za sinhronizaciju rada registra.

- N bitni registar sa paralelnim upisom i čitanjem ima jedan izlazni port:
  - *Q* izlaz za podatke, širine *n* bita



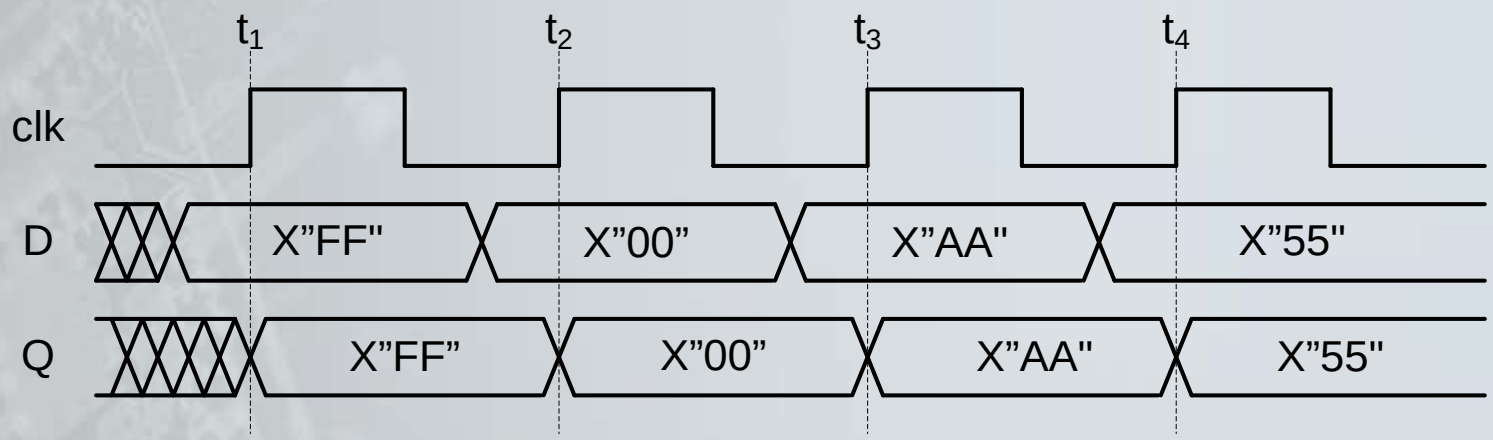
Hardverska implementacija pomerača IV

- Funkcionalnost registra sa paralelnim upisom i čitanjem prikazana je u tabeli desno
- Kao što se može primetiti na osnovu tabele, sve dok se na *clk* ulazu ne pojavi rastuća ivica, registar čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu
- Tek kada se na *clk* ulazu pojavi „sinhronizacioni događaj“ (rastuća ili opadajuća ivica), u registar se upisuje trenutna vrednost koja se nalazi na *D* ulazu
- Bitno je naglasiti da se ovaj upis obavlja paralelno, u svaki od unutrašnjih flip flopova koji čine registar se istovremeno upisuje odgovarajući bit ulaznog porta *D*
- Takođe, podaci koji su trenutno upisani u unutrašnje flip flopove se istovremeno (paralelno) pojavljuju na izlaznom portu *Q*
- Zbog toga se ovaj registar i zove registar sa paralelnim upisom i čitanjem podataka

clk	D	Q(t+1)
0	x	Q(t)
1	x	Q(t)
↓	x	Q(t)
↑	data	data

Hardverska implementacija pomerača V

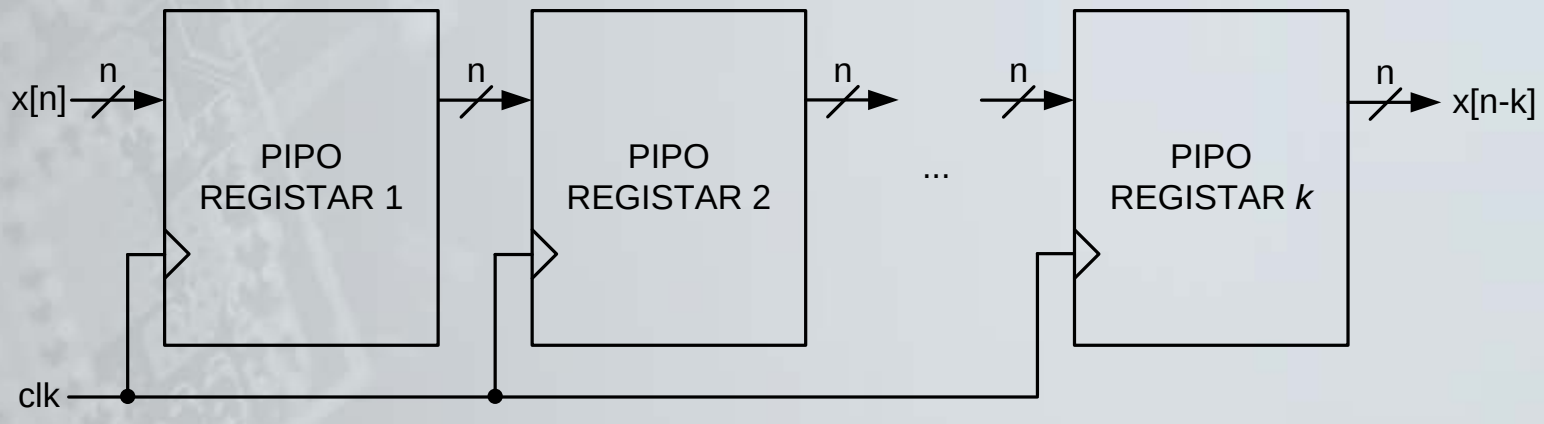
- Na primer, vremenski dijagram rada 8-bitnog registra sa paralelnim upisom i čitanjem prikazan je na slici dole



- VHDL model *n*-bitnog registra sa paralelnim upisom i čitanjem krajnje je jednostavan i prikazan je u nastavku

Hardverska implementacija pomerača VI

- U slučaju kada je potrebno realizovati pomerač koji pomera ulazni diskretni signal za *k* mesta u desno, on se lako projektuje kaskadnim vezivanjem *k* *n*-bitnih PIPO registara, kao što je prikazano na slici dole



Hardverska implementacija sabirača

Hardverska implementacija sabirača I

- Prilikom hardverske implementacije diskretnog sistema uobičajeno je da se za reprezentaciju vrednosti diskretnih signala i realnih konstanti koristi sistem sa fiksnom tačkom, baziran na bipolarnom kodu komplementa dvojke

- Stoga će u nastavku ovog i sledećeg poglavlja biće prikazana digitalna kola koja obavljaju aritmetičke operacija sabiranja i množenja u ovom sistemu reprezentacije brojeva

- Iako je poznato da je sistem za reprezentaciju realnih brojeva baziran na pokretnoj tački superiorniji, on se ređe koristi u praksi, jer rezultuje u znatno složenijim i sporijim digitalnim mrežama za realizaciju operacija sabiranja i množenja, u poređenju sa mrežama za izvođenje ovih aritmetičkih operacija u fiksnom zarezu

- Iz toga razloga digitalne mreže za realizaciju operacija sabiranja i množenja u pokretnom zarezu neće biti razmatrane

Hardverska implementacija sabirača II

- Pored gore navedene podele, aritmetička kola se mogu podeliti i prema tome kako obrađuju podatke prilikom svog rada na sledeće dve velike grupe:
  - **serijska**, odnosno, **bit po bit aritmetička kola** – kod kojih se ulazni operandi dovode na ulaze kola i obrađuju bit po bit,
  - **paralelna aritmetička kola** – kod koji se ulazni operandi dovode na ulaze kola i obrađuju reč po reč.
- Jasno je da je paralelno izvođenje aritmetičkih operacija znatno brže od serijskog, te se ono koristi ukoliko je potrebno implementirati diskretne sisteme visokih performansi
- Međutim, paralelno izvođenje aritmetičkih operacija ima i ozbiljne nedostatke, jer zahteva znatno veći broj hardverskih resursa (konkretno logičkih kapija) prilikom implementacije, što rezultuje u skupljem sistemu sa većom potrošnjom
- Upravo iz ovih razloga serijska realizacija aritmetičkih operacija nekad je od interesa, pogotovo ukoliko se u obzir uzmu ne samo performanse već i potrošnja kao i cena finalne implementacije

Hardverska implementacija sabirača III

- Osnovne gradivne blokove većeg broja različitih vrsta sabirača ali i ostalih tipova aritmetičkih kola predstavljaju:

  - jednobitni **polu-sabirač** (*Half Adder*, HA) i
  - **puni sabirač** (*Full Adder*, FA).

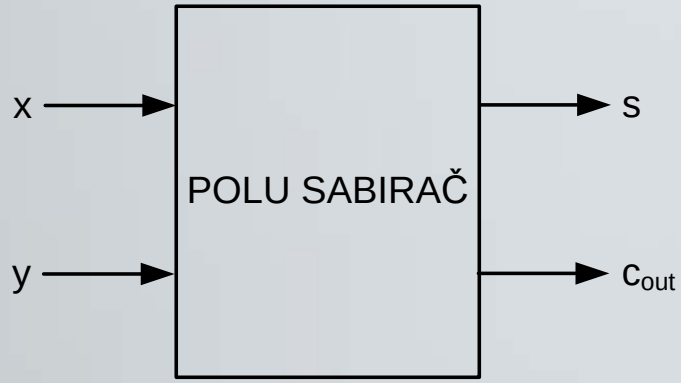
Polu-sabirač

- Polu-sabirač je kombinaciona mreža sa dva ulaza bita koji se sumiraju *x* i *y*, i dva izlaza, sume *s* i izlaznog prenosa c_{out}, prikazana na slici desno.

- Vrednosti izlaznih signala sume i prenosa, *s* i c_{out}, računaju se na osnovu sledećih izraza

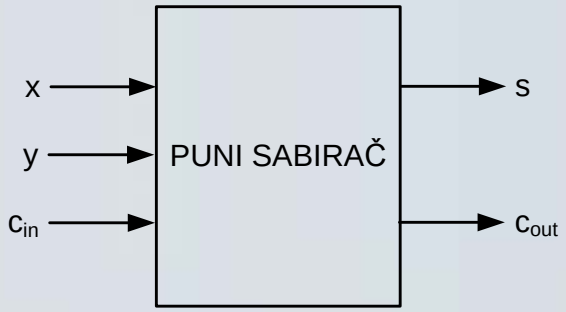
$$s = x \oplus y = x\overline{y} + \overline{x}y$$

$$c_{out} = xy$$



Puni sabirač

- Puni sabirač je kombinaciona mreža sa tri ulaza, dva ulazna bita koji se sumiraju *x* i *y*, i ulaznog prenosa c_{in}, i sa dva izlaza, sume *s* i izlaznog prenosa c_{out}, prikazana na slici desno gore

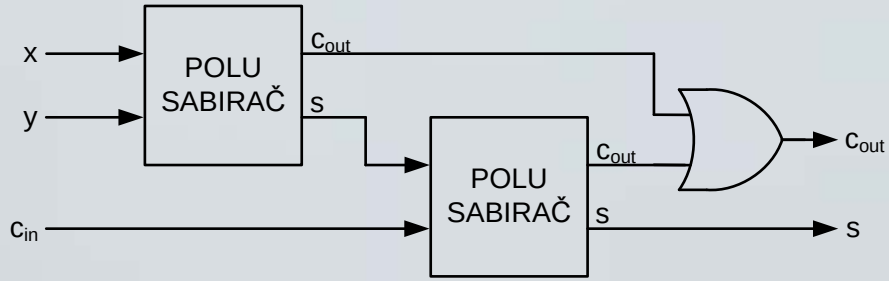


- Vrednosti izlaznih signala sume i izlaznog prenosa, *s* i c_{out}, ovaj put se računaju na osnovu sledećih izraza

$$s = x \oplus y \oplus c_{in} = xyc_{in} + \overline{xy}c_{in} + \bar{x}y\bar{c}_{in} + x\overline{yc}_{in}$$

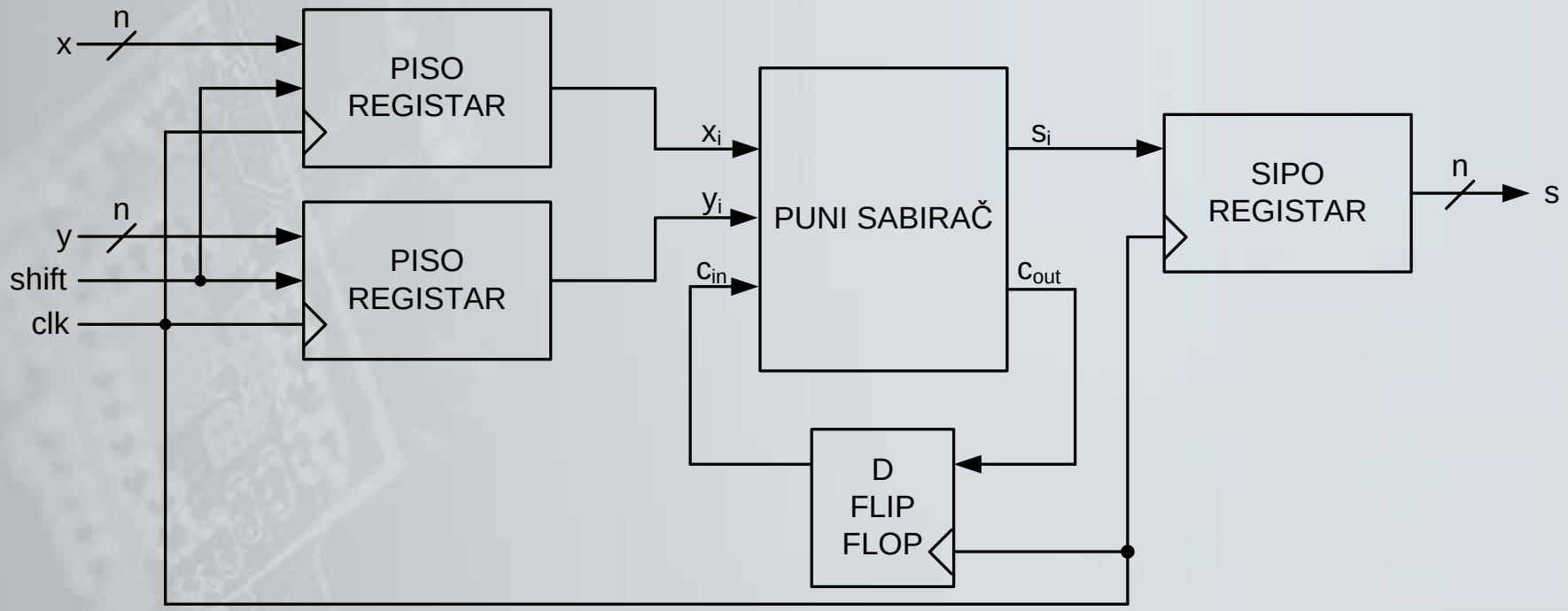
$$c_{out} = xy + xc_{in} + yc_{in}$$

- Puni sabirač može se realizovati pomoću dva polu-sabirača i jednog ILI kola, kao što je prikazano na slici desno dole



Serijski sabirači I

- Kao što je ranije već rečeno, serijski sabirač vrši sabiranje dva *n*-bitna broja serijski, bit po bit. Blok dijagram serijskog sabirača prikazan je na slici dole.



- Serijski sabirač sastoji se iz dva registra sa paralelnim upisom i serijskim čitanjem (PISO), u koje se upisuju vrednosti sabiraka koje je potrebno sabrati, preko ulaznih portova *x* i *y*, na početku ciklusa sabiranja

Serijski sabirači II

- Sam proces sabiranja odvija se serijski, bit po bit, počevši od bita najmanje težine, korišćenjem jednog punog sabirača
- Kako je potrebno pamtiti prenos sa prethodne pozicije i uključiti ga prilikom sumiranja cifara sa tekuće pozicije, neophodan je i jedan flip flop koji služi za memorisanje vrednosti prenosa
- Izračunate cifre sume smeštaju se bit po bit u registar sa serijskim ulazom i paralelnim izlazom (SIPO)
- Sam proces sabiranja traje ukupno *n* taktova
- Nakon završetka sabiranja rezultat se može pročitati u paralelnom obliku na izlaznom portu *s*

Serijski sabirači III

- Za sabiranje dva *n*-bitna broja pomoću serijskog sabirača potrebno *n* taktova
- Ukoliko se serijski sabirač koristi za implementaciju diskretnog sistema koji radi na učestanosti odabiranja f_s, serijski sabirač mora raditi na učestanosti koja je *n* puta veća od učestanosti odabiranja, $f_{SA}=n \cdot f_s$
- Ovo može biti problem ukoliko se serijski sabirači koriste prilikom implementacije diskretnih sistema sa visokom učestanošću odabiranja ili sa velikim dužinama digitalnih reči za predstavljanje vrednosti signala
- Kao što je već bilo reči u uvodnom delu ovog predavanja, hardverska implementacija pruža veliku fleksibinost prilikom implementacije diskretnog sistema

Serijski sabirači IV

- Umesto da se individualne aritmetičke operacije izvode u paraleli, korišćenjem odgovarajućih paralelnih arhitektura, moguće je veći broj artimetičkih operacija izvesti u paralelim korišćenjem paralelnih arhitektura za implementaciju diskretnih sistema

- U ovom slučaju se individualne operacije sabiranja i množenja izvode serijski, bit po bit, ali se sve ove operacije izvode istovremeno, za svaki sabirač i množač koji je prisutan u blok dijagramu diskretnog sistema koji se implementira

- Međutim, serijske arhitekture za izvođenje aritmetičkih operacija zahtevaju znatno manji broj hardverskih resursa za njihovu implementaciju, po pravilu *n* puta manje resursa od odgovarajućih paralelnih arhitektura, gde je sa *n* označen broj bita sa kojima su predstavljene vrednosti operanada koji učestvuju u aritmetičkoj operaciji

Serijski sabirači V

- Pravilnim izborom mesta na kojim će se primeniti paralelni, odnosno serijski način obrade mogu se postići različiti kompromisi u pogledu brzine rada sistema i potrebnih hardverskih resursa za njegovu realizaciju

- Zbog toga se u praksi često koriste i serijske arhitekture za izvođenje aritmetičkih operacija

Paralelni sabirači I

- Paralelni sabirači izvode sabiranje dva operanda tako što istovremeno, u paraleli, izračunavaju vrednosti svih bita sume
- Obzirom da se svih *n* bita rezultata sume izračunava istovremeno, paralelni sabirači su teoretski *n* puta brži od serijskih
- Međutim, sada je potrebno imati *n* puta više hardverskih modula, u poređenju sa serijskim arhitekturama, koji će istovremeno izračunavati vrednosti bitova rezultujuće sume
- To znači da će paralelne arhitekture zahtevati barem *n* puta više hardverskih resursa u poređenju sa serijskim arhitekturama
- U stručnoj literaturi postoji veliki broj različitih arhitektura za paralelno sabiranje brojeva, koje se međusobno razlikuju po brzini sabiranja i potrebnim hardverskim resursima za njihovu implementaciju

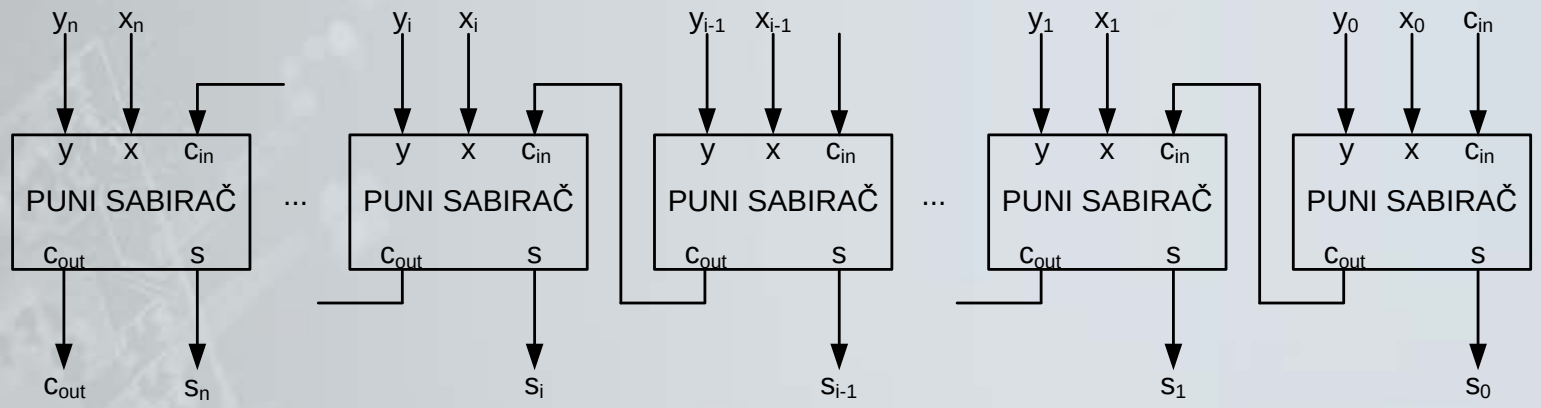
Paralelni sabirači II

- Najpoznatije arhitekture paralelnih sabirača su:
  - ***Ripple-carry*** sabirači,
  - ***Carry-look-ahead*** sabirači,
  - ***Prefix*** sabirači,
  - ***Carry-select*** sabirači,
  - ***Carry-skip*** sabirači,
  - ***Carry-save*** sabirači,
  - ***Conditional-sum*** sabirači,
  - hibridni sabirači.

- Na ovom mestu prikazaćemo način rada dve najjednostavnije vrste paralelnih sabirača, *ripple-carry* i *carry-look-ahead* sabirača

Ripple-carry sabirač I

- Kaskadnim vezivanjem *n* punih sabirača, na način prikazan na slici dole, dobijamo *n*-bitni *ripple-carry* sabirač



- Kao što se sa slike može videti, na *x* i *y* ulaze *i*-tog punog sabirača dovode se *i*-ti bitovi iz sabiraka *x* i *y*, x_i i y_i
- Puni sabirači su međusobno povezani u dugački lanac preko svojih c_{out} i c_{in} portova
- Na c_{in} ulaz *i*-tog punog sabirača dovodi se signal koji predstavlja c_{out} izlaz *(i-1)*-og punog sabirača
- Na ovaj način se obezbeđuje pravilno prostiranje signala prenosa (takozvani „*carry ripple*“ efekat, otuda i naziv za ovu vrstu paralelnog sabirača) sa pozicija niže težine ka pozicijama više težine u procesu sabiranja

Ripple-carry sabirač II

- Kašnjenje prilikom sabiranja dva *n*-bitna binarna broja pomoću *ripple-carry* sabirača može se proceniti na sledeći način

- Obzirom da je reč o kombinacionoj mreži, kašnjenje je jednako najdužem kombinacionom putu kroz *ripple-carry* sabirač sa slike sa prethodnog slajda

- Analizom sistema sa slike sa prethodnog slajda može se zaključiti da je najduža putanja kroz *ripple-carry* sabirač jedna od putanja koje počinju na x_0, y_0 ili c_{in} ulazima koji su povezani na ulaze punog sabirača na poziciji bita najmanje značajnosti (lociranog krajnje desno na slici), prolaze kroz *carry* lanac, i završavaju na s_n ili c_{out} izlazima punog sabirača na poziciji bita najveće značajnosti (lociranog krajnje levo na slici)

Ripple-carry sabirač III

- Uzimajući ovo u obzir, kašnjenje *n*-bitnog *ripple-carry* sabirača ima sledeću vrednost

$$T_{RCA}(n) = \max\{T_{FA}(x, y, c_{in} \rightarrow c_{out})\} + (n-2)T_{FA}(c_{in} \rightarrow c_{out}) + \max\{T_{FA}(c_{in} \rightarrow c_{out}, s)\}$$

- Na osnovu prethodnog izraza možemo videti da kašnjenje *ripple-carry* sabirača raste linearno sa porastom broja bita pomoću kojih su reprezentovane vrednosti sabirača *x* i *y*, *n*!

- Zbog toga su *ripple-carry* sabirači nepoželjni u diskretnim sistemima koji koriste veliki broj bita za reprezentaciju signala ili rade na visokim učestanostima odabiranja.

Ripple-carry sabirač IV

- Obratite pažnju da kašnjenje serijskog sabirača, prikazanog na slajdu 25 takođe raste linearno sa *n*!
- Ovo znači da *ripple-carry* sabirač ima kašnjenje koje je proporcionalno sa serijskim sabiračem!
- Kašnjenje *ripple-carry* sabirača je zapravo manje zbog toga što je je perioda klok signala, neophodnog za rad serijskog sabirača uvek veća od $T_{RCA}(n)/n$, jer kritična putanja unutar serijskog sabirača pored propagacionog kašnjenja punog sabirača uključuje i T_{cq} i T_{setup} vremena na registrima, odnosno flip-flopu za čuvanje prenosa
- Ripple-carry sabirač zapravo predstavlja najneefikasniju arhitekturu paralelnog sabirača, kada se kao kriterijum efikasnosti uzima samo vreme potrebno da se završi operacija sabiranja

Carry-look-ahead sabirač I

- Na osnovu analize trajanja operacije sabiranja realizovane pomoću *ripple-carry* sabirača, jasno je da je glavni faktor koji utiče na trajanje operacije sabiranja propagacija signala prenosa sa pozicije namanje težine kroz čitav sabirač sve do pozicije najveće težine

- U cilju ubrzavanja procesa sabiranja potrebno je na neki način skratiti ovaj dugački propagacioni put

- U opštem slučaju ovo ubrzavanje se može izvesti na dva načina:

  - Skraćivanjem/ubrzavanjem vremena propagacije signala prenosa.

  - Detekcijom završetka propagacije signala prenosa duž lanca i kompletiranjem procesa sabiranja. Na ovaj način se izbegava čekanje na kompletiranje u maksimalnom trajanju od $n \cdot T_{FA}$ sekundi, osim kada je to stvarno neophodno.

Carry-look-ahead sabirač II

- Drugi pristup bi rezultovao u promenljivom trajanju operacije sabiranja, u zavisnosti od vrednosti ulaznih operanada, što nije poželjno prilikom korišćenja ovakvih modula unutar sinhronih sekvencijalnih sistema
- Zbog toga se većina predloženih arhitektura zasniva na prvom pristupu
- Najčešće korišćena arhitektura za ubrzavanje propagacija signala prenosa jeste takozvana „*carry-look-ahead*“ šema
- Paralelni sabiračikoji koriste ovu šemu poznati su pod nazivom *carry-look-ahead* sabirači, CLA
- Glavna ideja ovog pristupa je da se svi potrebni signali prenosa (za svaki od punih sabirača, osim prvog) generišu/izračunaju istovremeno, u paraleli

Carry-look-ahead sabirač III

- Na ovaj način bi se izbegla potreba za propagacijom potrebne vrednosti duž dugačkog lanca punih sabirača
- Ovako nešto je u principu moguće izvesti, jer način generisanja vrednosti svih signala prenosa, kao i način njihove propagacije kroz sabirač, zavisi samo od vrednosti ulaznih operanada *x* i *y*
- Ove vrednosti su na raspolaganju, istovremeno, svakom od punih sabirača
- Stoga svaki puni sabirač ima na raspolaganju sve informacije koje su neophodne za izračunavanje potrebne vrednosti ulaznog signala prenosa, a samim tim i za korektno izračunavanje asociranog bita sume
- Ovakav način bi na žalost, zahtevao jako složene kombinacione mreže sa velikim brojem ulaza, na ulazima za prenos svakog od punog sabirača, te se stoga ne koristi u praksi

Carry-look-ahead sabirač IV

- Jedan od načina da se smanji broj potrebnih ulaza u mreže za generisanje signala za prenos je da se, na osnovu pridruženih cifara ulaznih operanada, proceni da li će novi signali prenosa biti generisani ili propagirani

- Razmotrimo pod kojim uslovima dolazi do generisanja, odnosno propagacije signala prenosa kroz puni sabirač

- U slučaju kada su ulazni signali x_i i y_i jednaki jedan, $x_i = y_i = 1$, na izlazu punog sabirača generisaće se izlazni signal prenosa, c_{i+1}, bez obzira na vrednost ulaznog signala prenosa, c_i

- U slučaju kada važi $x_i y_i = 10$ ili $x_i y_i = 01$, signal prenosa se može samo propagirati

- Na kraju, u slučaju da važi $x_i y_i = 00$ izlazni signal prenosa se ne može ni generisati ni propagirati

Carry-look-ahead sabirač V

- Definišemo dve nove funkcije, funkciju generisanog signala prenosa $G_i = x_i \cdot y_i$, i funkciju propagiranog signala prenosa $P_i = x_i + y_i$, gde su sa $\cdot$ i + označene operacije logičkog I i ILI repsektivno

- Korišćenjem ove dve funkcije, izraz za računanje izlaznog signala prenosa *i*-tog punog sabirača može se napisati na sledeći način

$$c_{i+1} = x_i \cdot y_i + c_i \cdot (x_i + y_i) = G_i + c_i \cdot P_i.$$

- Kako važi

$$c_i = G_{i-1} + c_{i-1} \cdot P_{i-1},$$

- prethodni izraz za računanje izlaznog signala prenosa postaje

$$c_{i+1} = G_i + G_{i-1} \cdot P_i + c_{i-1} \cdot P_{i-1} \cdot P_i.$$

Carry-look-ahead sabirač VI

- Daljim zamenama izraza sa signale c_{i-i}, c_{i-2}, ..., konačno dobijamo

$$c_{i+1} = G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_{i-1} \cdot P_i + c_{i-2} \cdot P_{i-2} \cdot P_{i-1} \cdot P_i = \ldots =$$
$$= G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_{i-1} \cdot P_i + \ldots + c_{i-2} \cdot P_0 \cdot P_1 \cdot \ldots \cdot P_i.$$

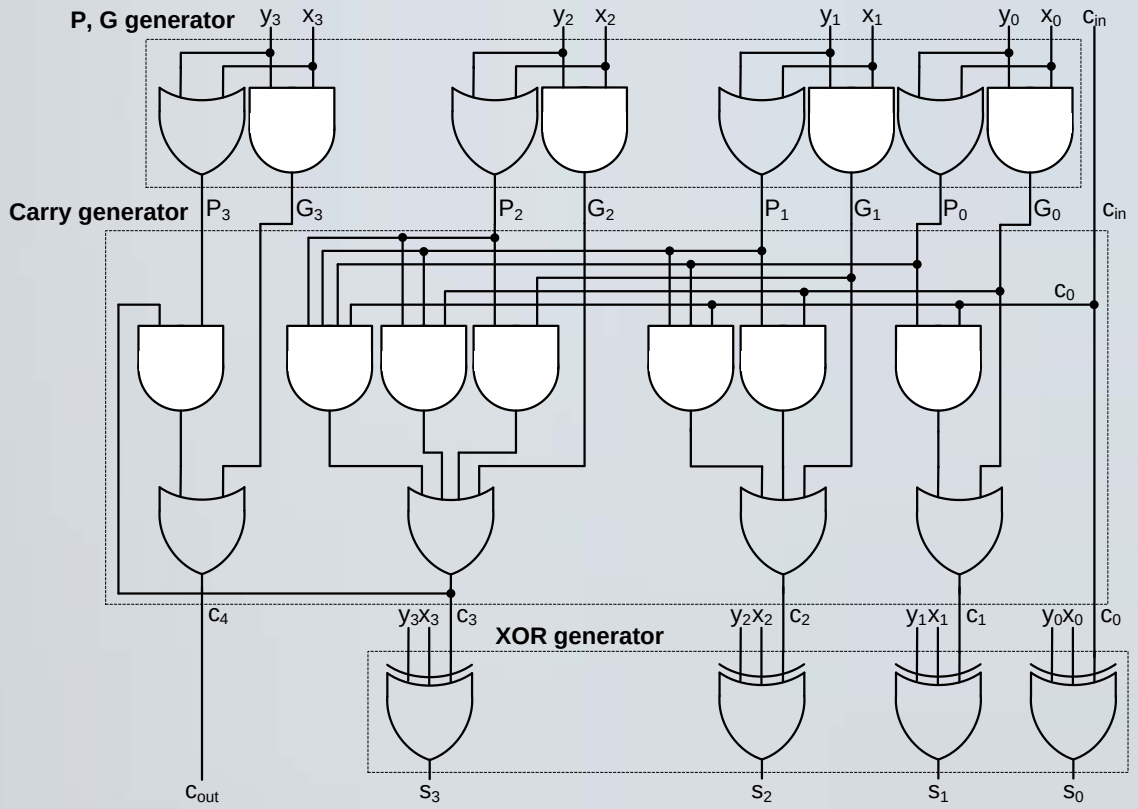
- Na osnovu prethodnog izraza moguće je izračunati, **u paraleli**, vrednosti svih izlaznih signala prenosa samo na osnovu vrednosti cifara operanada x i y, $x_{n-1}x_{n-2}\ldots x_0$, $y_{n-1}y_{n-2}\ldots y_0$ i ulaznog signala prenosa c_0

- Na primer, za 4-bitni CLA sabirač vrednosti izlaznih signala prenosa mogu se izračunati na sledeći način

$$c_1 = G_0 + c_0 \cdot P_0,$$
$$c_2 = G_1 + G_0 \cdot P_1 + c_0 \cdot P_0 \cdot P_1,$$
$$c_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + c_0 \cdot P_0 \cdot P_1 \cdot P_2,$$
$$c_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + c_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3.$$

Carry-look-ahead sabirač VII

- Struktura 4-bitnog CLA sabirača projektovanog na ovaj način prikazana je na slici desno
- Kao što se sa slike može videti, 4-bitni CLA sabirač se sastoji iz tri bloka:
  - Bloka za generisanje signala P_i i G_i,
  - Bloka za generisanje potrebnih unutrašnjih signala prenosa, c_i,
  - Bloka za računanje signala izlazne sume, s.
- Svaki od ovih blokova računa potrebne signale u paraleli.



Carry-look-ahead sabirač VIII

- Procenimo koliko bi bilo kašnjenje *n*-bitnog CLA sabirača koji koristi ovakav način generisanja unutrašnjih signala prenosa
- Generisanje signala G_i i P_i traje T_G sekundi, gde je sa T_G označeno propagaciono kašnjenje kroz standardnu logičku kapiju (logičko I ili ILI kolo)
- Dodatno kašnjenje od $2T_G$ je neophodno da se generišu unutrašnji signali prenosa, c_1, c_2, ..., c_{n-1}, pod pretpostavkom da su Bulove funkcije koje opisuju generisanje ovih signala realizovane pomoću kombinacionih mreža sa dva nivoa
- Na kraju, potrebno je dodatnih $2T_G$ sekundi za generisanje izlaznih bitova sume, s_0, s_1, ..., s_n, ponovu uz pretpostavku da se generišu korišćenjem kombinacionih mreža sa dva nivoa (pogledati izraz za generisanje izlaznog signala sume *s*, punog sabirača)
- Dakle potrebno je ukupno $5T_G$ sekundi, bez obzira na dužinu ulaznih operanada *n*, za izračunavanje sume dva *n*-bitna broja pomoću ovako projektovanog sabirača!

Carry-look-ahead sabirač IX

- Na žalost, za velike vrednosti *n*, na primer *n*=32, broj individualnih logičkih kapija postaje ekstremno velik, a što je još značajnije neophodno je korišćenje logičkih kapija sa jako velikim brojem ulaza (*fan-in*)
- Ovakve kapije je vrlo kompolikovano fizički realizovati, a i njihova propagaciona kašnjenja su znatno veća od propagacionih kašnjenja dvoulaznih kapija, što menja izraz za kašnjenje sabirača
- Stoga je neophodno smanjiti opseg paralelno generisanih unutrašnjih signala prenosa, po cenu povećanja kašnjenja
- U praksi se ovo postiže tako što se sabirač koji se sastoji iz *n* punih sabirača podeli u grupe od po *k* sabirača, a za svaku od ovih grupa se koristi jedan blok za generisanje unutrašnjih *k* signala prenosa

Carry-look-ahead sabirač X

- Ove grupe se zatim povezuju na način na koji je to urađeno u slučaju *ripple-carry* sabirača
- Veličina grupe obično iznosi 4, jer je ovaj broj zajednički faktor velikog broja dužina digitalnih reči koje se koriste u praksi, i zato što fabrikacija logičkih kapija sa do četiri ulaza ne predstavlja veliki tehnološki izazov
- Procenimo kašnjenje CLA sabirača koji bi koristio prikazani način generisanja unutrašnjih signala prenosa, baziran na korišćenju grupnih generatora opisanih malopre
- U ovom slučaju se lanac od *n* punih sabirača deli u grupe od po 4 sabirača, odnosnu na *n/4* grupa

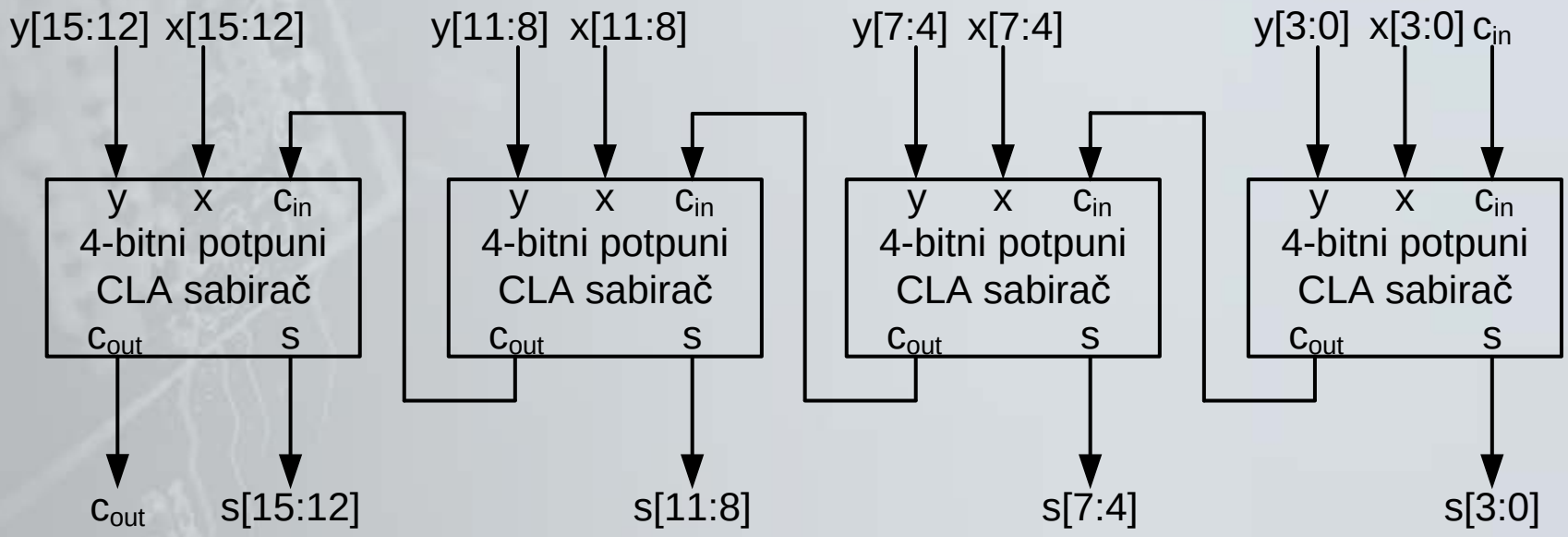
Carry-look-ahead sabirač XI

- Na osnovu izraza sa dna slajda 45 vidimo da je potrebno $2T_G$ sekundi da se generišu svi signali prenosa unutar jedne grupe, nakon što su poznate vrednosti P_i, G_i i c_0
- Kao i ranije, za računanje vrednosti P_i, G_i potrebno je T_G sekundi
- Kako su grupe kaskadno vezane, najgore vreme za propagaciju signala prenosa kroz svih $n/4$ grupa iznosi $(n/4)2T_G$
- Nakon što su se svi signali prenosa stabilizovali, potrebno je još $2T_G$ sekundi za računanje izlaznih bitova sume, s_i
- Ukupno trajanje operacije sabiranja dva n-bitna broja korišćenjem ovog CLA sabirača iznosi $(2n/4+3)T_G = (n/2+3)T_G$ sekundi

Carry-look-ahead sabirač XII

- Ovo je skoro četiri puta manja vrednost od trajanja operacije sabiranja u slučaju korišćenja RCA sabirača, $T_{RCA}=2nT_G$

- Na primer, u slučaju realizacije 16-bitnog sabirača korišćenjem ove arhitekture, čiji blok dijagram je prikazan na slici dole, sabiranje dva broja trajalo bi 11 T_G sekundi



Carry-look-ahead sabirač XIII

- Trajanje operacije sabiranja možemo dodatno skratiti ako primenimo ideju paralelnog generisanja signala prenosa, ali ovaj put nad grupama

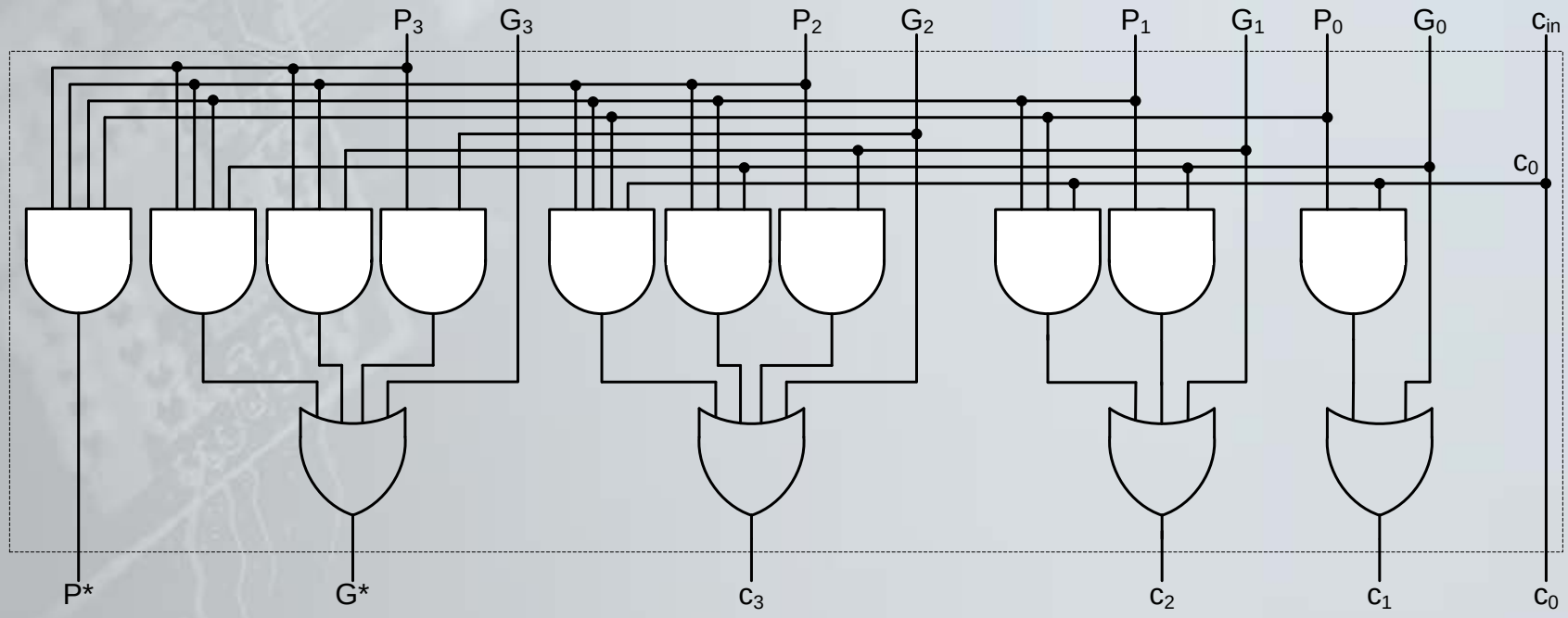
- Definišimo **grupno generisani signal prenosa**, G^*, i **grupno propagirani signal prenosa**, P^*, za grupu veličine 4 puna sabirača sabirača na sledeći način

$$G^* = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3,$$
$$P^* = P_0P_1P_2P_3.$$

- Signal G^* ima vrednost 1 ako se izlazni signal prenosa čitave grupe generiše unutar te grupe, a signal P^* ima vrednost 1 ako se ulazni signal prenosa u grupu propagira unutar grupe do izlaznog signala prenosa te grupe

Carry-look-ahead sabirač XIV

- Kombinaciona mreža koja generiše grupu unutrašnjih signala prenosa, zajedno sa njima asociranim signalima *G** i *P**, poznata je pod nazivom **CLA (*carry-look-ahead*) generator**
- Na primer, izgled 4-bitnog CLA generatora prikazan je na slici dole



Carry-look-ahead sabirač XV

- Ovako generisani signali *G** i *P** iz većeg broja grupa se sada mogu koristiti za paralelno generisanje ulaznih signala prenosa u pojedinačne grupe, na način koji je sličan generisanju individualnih signala prenosa unutar grupe

- Na primer, ukoliko je potrebno projektovati 16-bitni CLA sabirač, koristeći CLA generatore veličine 4, postojaće sledeći grupno generisani i propagirani signali

$$G_0^*, G_1^*, G_2^*, G_3^*, P_0^*, P_1^*, P_2^*, P_3^*$$

- Ovi signali predstavljaju ulaze u dodatni CLA generator, koji se nalazi na sledećem nivou

Carry-look-ahead sabirač XVI

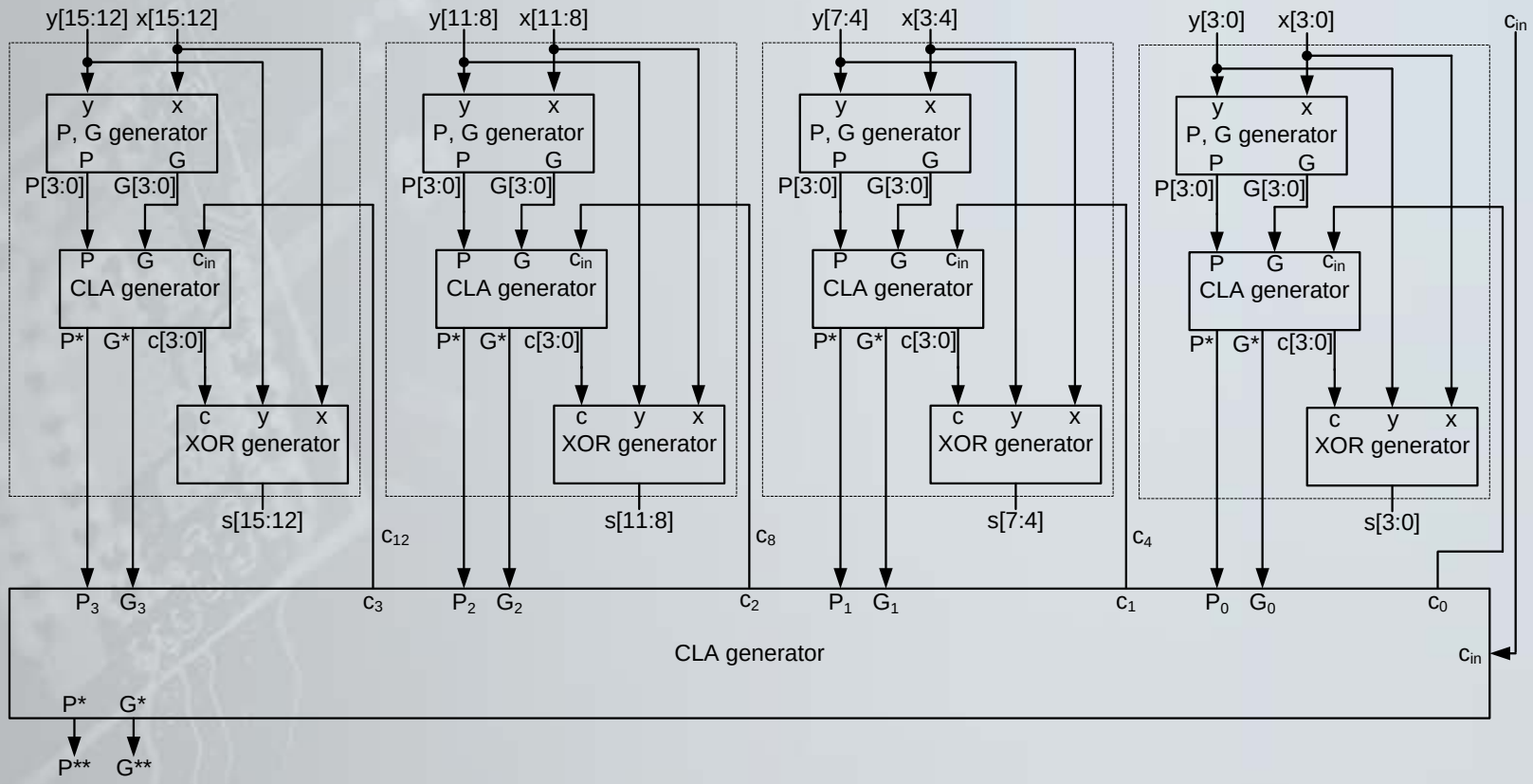
- Izlazi ovog CLA generatora zapravo su signali prenosa na pozicijama 4, 8 i 12, c_4, c_8 i c_{12}, koji se generišu na sledeći način

$$c_4 = G_0^* + c_0 P_0^*,$$
$$c_8 = G_1^* + G_0^* P_1^* + c_0 P_0^* P_1^*,$$
$$c_{12} = G_2^* + G_1^* P_2^* + G_0^* P_1^* P_2^* + c_0 P_0^* P_1^* P_2^*$$

- Vidimo da gornji su gornji izrazi identični sa defincionim izrazima CLA generatora, prikazanim ranije

Carry-look-ahead sabirač XVII

- Struktura 16-bitnog CLA sabirača projektovanog na ovaj način prikazana je na slici dole



Carry-look-ahead sabirač XVIII

- Procenimo trajanje operacije sabiranja dva 16-bitna broja pomoću gornjeg sabirača
- U prvom koraku generišu se G_i i P_i signali. Ova operacija traje T_G sekundi.
- Nakon toga CLA generatori iz prvog nivoa generišu svoje i signale. Ova operacija traje $2T_G$ sekundi.
- U sledećem koraku, CLA generator iz drugog nivoa generiše signale c_4, c_8 i c_{12}. Ova operacija takođe traje $2T_G$ sekundi.
- Na kraju, CLA generatori iz prvog nivoa, na osnovu signala c_4, c_8 i c_{12} generišu unutrašnje signale prenosa i na osnovu njih računaju izlazne bitove sume, s_i. Ove dve operacije traju $2T_G + 2T_G = 4T_G$ sekundi.
- Ukupno, čitav proces sabiranja pomoću ovog CLA sabirača traje $9T_G$ sekundi, umesto $11T_G$ sekundi u slučaju korišćenja CLA sabirača sa kaksadnom vezom između grupa, opisanog ranije

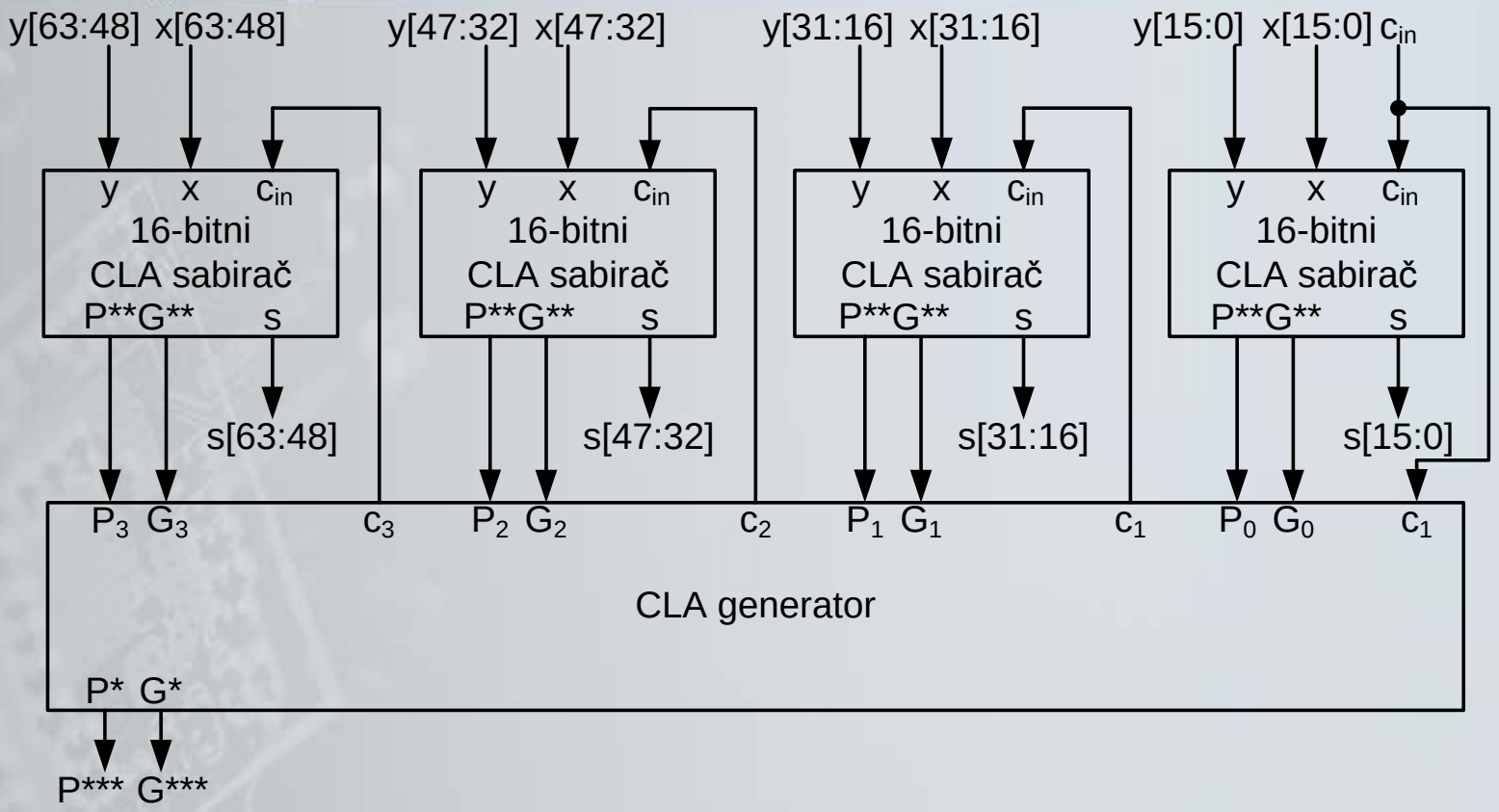
Carry-look-ahead sabirač XIX

- Proces generisanja grupnih signala prenosa se može ponavljati neograničen broj puta, dodavanjem dodatnih nivoa CLA generatora, u slučaju implementacije sabirača veće širine

- Na primer, u slučaju implementacije 64-bitnog sabirača možemo koristiti 4 16-bitna CLA sabirača sa slajda 59 čiji su i signali dovedeni na dodatni CLA generator, lociran u sledećem nivou, koji na osnovu njih generiše signale prenosa c_{16}, c_{32} i c_{48}

- Struktura 64-bitnog CLA sabirača projektovanog na ovaj način prikazana je na sledećem slajdu

Carry-look-ahead sabirač XX



- Kašnjenje ovako projektovanog CLA sabirača, u slučaju sabiranja dva *n*-bitna broja, gde je *n* multipl broja 4, iznosi $T_{CLA4} = 4\log_4 n + 1$
- Svaki nivo CLA generatora unosi dodatno kašnjene od 4 T_G sekundi, plus kašnjenje od T_G sekundi koje je potrebno za generisanje bitskih G_i i P_i signala

Carry-look-ahead sabirač XXI

- Na osnovu poslednjeg izraza možemo zaključiti da kašnjenje CLA sabirača raste logaritamski sa porastom širine ulaznih operanada, *n*
- Ovo je mnogo bolje od RCA sabirača, kod kojega kašnjenje raste linearno sa porastom širine ulaznih operanada, *n*
- Bitno je napomenuti da navedena teorijska kašnjenja različitih arhitektura sabirača važe u slučaju kada se ovi sistemi implementiraju kao ASIC kola. U slučaju FPGA to ne mora uvek da bude slučaj.
- Glavni razlog zbog čega je to tako je da se logičke mreže unutar FPGA kola mapiraju u takozvane *look-up* tabele, male memorije sa tipično 4 do 6 ulaza i jednim izlazom, a ne realizuju se pomoću logičkih kapija kao u slučaju ASIC tehnologije

Carry-look-ahead sabirač XXII

- Način mapiranja logičkih kapija u *look-up* tabele u opštem slučaju je teško predvideti, tako da karakteristike različitih arhitektura sabirača mogu u većoj ili manjoj meri odstupati od izvedenih izraza
- Takođe, u slučaju FPGA tehnologije kašnjenje signala koji povezuju *look-up* tabele je znatno veće od odgovarajućeg kašnjenja signala koji povezuju logičke kapije u ASIC tehnologiji
- Zbog toga će sam način raspoređivanja potrebnih hardverskih komponenti za implementaciju odabranog sabirača imati daleko veći uticaj na krajnje performanse sistema u slučaju FPGA tehnologije nego što je to slučaj ako se koristi ASIC tehnologija
- Pored toga, savremena FPGA komponente po pravilu poseduju odgovarajući broj hardverskih sabirača i množača, organizovanih u takzvane DSP blokove

Carry-look-ahead sabirač XXIII

- Ova aritmetička kola su realizovana kao takozvani *hard IP* blokovi, odnosno nisu implementirana pomoću programabilnih resursa, već imaju fiksnu, optimizovanu organizaciju koja im omogućuje da budu daleko efikasnija u odnosu na ista takva kola koja su implementirana pomoću programabilnih resursa

- Zbog svega navedenog, u slučaju FPGA implementacije treba biti vrlo oprezan prilikom izbora optimalnog načina implementacije aritmetičkih operacija, jer se optimalne arhitekture mogu u znatnoj meri razlikovati od arhitektura koje su optimalne prilikom ASIC implementacije