Projektovanje mikroprocesorskih sistema visoke pouzdanosti

Osnovni pojmovi

Sistemi visoke pouzdanosti, ili sistemi otporni na otkaze (*fault-tolerant systems*) su oni koji i u vrlo nepovoljnim, pa i ekstremnim uslovima, zahvaljujući pre svega sposobnosti da tolerišu pojedinačne kvarove, nastavljaju da izvršavaju svoje funkcije.

Pouzdanost

Pouzdanost R(t) sistema je vremenska funkcija, definisana kao uslovna verovatnoća da će sistem raditi korektno u intervalu $[t_0, t]$, pretpostavljajući da je sistem radio korektno u trenutku t_0. Drugim rečima, pouzdanost je verovatnoća da će sistem biti ispravan u *celokupnom* posmatranom intervalu, pod pretpostavkom da je bio ispravan na početku tog intervala. **Nepouzdanost Q**(t) sistema je vremenska funkcija, definisana kao uslovna verovatnoća da će sistem raditi *pogrešno* u intervalu $[t_0, t]$, pretpostavljajući da je sistem radio *ispravno* u trenutku t_0. Za nepouzdanost se često koristi termin *verovatnoća otkaza*.

Pouzdanost se najčesce koristi za karakterizaciju sistema kod kojih su čak i trenutni periodi neispravnog rada neprihvatljivi, ili u kojima je nemoguće popraviti sistem. Vremenski intervali pri razmatranju pouzdanosti su ponekad veoma dugi, kao recimo deset godina, a u nekim drugim slučajevima dugi svega nekoliko sati. Uobičajeno je da se pri predstavljanje pouzdanosti koristi notacja 0.9_i koja označava vrednosti koja ima *i* devetki desno od decimalne tačke. Na primer, 0.9999999 se piše kao 0.9_7.

Bitno je razumeti razliku izmedju otpornosti na otkaze i pouzdanosti. Otpornost na otkaze je tehnika koja može poboljšati pouzdanost sistema održavajući sistem operativnim uprkos pojavi hardverskih kvarova i softverskih grešaka. (Na primer, računar koji poseduje rezervni procesor, može biti projektovan tako da nastavi sa radom nakon otkaza glavnog procesora). Međutim, sistem otporan na otkaze ne mora uvek imati visoku pouzdanost. Naime, sistem može biti projektovan da toleriše bilo koji pojedinačni hardverski kvar ili softversku grešku koji se mogu pojaviti, ali bez obzira na to, ako je verovatnoća pojavljivanja ovakvih problema velika, pouzdanost sistema biće mala. Slično, visoko-pouzdani sistem ne mora obavezno biti i otporan na otkaze. Na primer, visoka pouzdanost se može postići ugradnom u sistem kvalitetnih komponenti. Međutim, bez obzira na veoma malu verovatnoću pojave hardverskog kvara, ako nakon pojave kvara sistem ne može da nastavi sa radom, takav sistem ipak nije otporan na otkaze.

Dostupnost

Dostupnost A(t) je vremenska funkcija, definisana kao verovatnoća da sistem radi korektno i da je na raspolaganju da izvrši svoje funkcije u vremenskom trenutku *t*. Dostupnost se razlikuje od pouzdanosti u tome što pouzdanost zavisi od vremenskog *intervala*, dok je dostupnost uzeta u jednom vremenskom *trenutku*. Mnogi sistemi se u toku svog životnog veka više puta kvare i popravljaju. Sistem može biti visoko-dostupan, a da u isto vreme nije pouzdan, sve dok su intervali u kojim je sistem u kvaru kratki. Dostupnost sistema zavisi ne samo od toga koliko često sistem postaje neispravan već takođe i od toga koliko brzo može biti popravljen. Najčesce korišćena mera dostupnosti je očekivani vremenski period u kome je sistem na raspolaganju da korektno izvrši svoje funkcije, tj. količnik vremena u kome je sistem bio ispravan i ukupnog vremena njegovog života. Dostupnost se koristi kao cilj projektovnja kada je osnovna uloga sistema obezbediti njegove usluge sto je češće moguće.

Bezbednost

Bezbednost S(t) je verovatnoća da će sistem ili obavljati svoje funkcije korektno ili će, zbog nastalog kvara, prekinuti svoj rad na način koji ne ometa rad drugih sistema ili ugrožava bezbednost ljudi koji su u kontaktu sa sistemom. Bezbednost predstavlja meru sposobnosti sistema za tzv. *bezbedni-otkaz*. Naime, ako sistem ne radi korektno, želimo da makar on otkaže na bezbedan način. Na primer, poželjno je da u slučaju otkaza mikroprocesorskog sistema, ventil koji kontroliše dotok neke hemikalije u reaktor ostane u zatvorenom položaju. Bezbednost sistema je verovatnoća da se to i desi.

Performabilnost

Performabilnost P(L,t) sistema je vremenska funkcija, definisana kao verovatnoća da će performanse sistema biti na, ili iznad, nekog nivoa *L* u vremenskom trenutku *t*. Performabilnost se razlikuje od pouzdanosti po tome što je pouzdanost mera verovatnoće da se *sve* funkcije izvršavaju korektno, dok je performabilnost verovatnoća da se neki *podskup* funkcija izvršava korektno. Na primer, kod multiprocesorskih sistema, trenutni broj operativnih procesora ukazuje na performabilnost.

Postepeno otkazivanje je važna osobina koja je usko povezana sa performabilnošću. **Postepeno otkazivanje** je sposobnost sistema da automatski snizi svoj nivo performansi da bi kompenzovao hardverski kvar ili softversku grešku.

Opravljivost

Opravljivost je mera lakoće sa kojom sistem može biti popravljen. Drugim rečima opravljivost **M**(t) je verovatnoća da "pokvareni" sistem može biti doveden u operativno stanje unutar određenog vremenskog perioda *t*. Proces *opravljanja* obuhvata lociranje problema, fizičku popravku sistema, i vraćanje sistema u njegovo operativno stanje. Opravljivost je kritična kod svih sistema, ali je posebno važna kad su ljudski životi, oprema ili sredina ugroženi dok se sistem popravlja.

Testabilnost

Test je sredstvo kojim se utvrđuje postojanje i kvalitet određenih karakteristika sistema. **Testabilnost** je *sposobnost* ispitivanja odredjenih karakteristika sistema. Testabilnost je jasno povezana sa opravljivošću jer je važno minimizirati vreme potrebno za identifikaciju i lociranje specifičnih problema.

Uslužnost

Termin **uslužnost** objedinjuje pojmove pouzdanosti, dostupnosti, sigurnosti, opravljivosti, performabilnosti, i testabilnosti. Uslužnost je kvalitet usluge koju određeni sistem obezbeđuje. Pouzdanost, dostupnost, sigurnost, opravljivost, performabilnost i testabilnost su mere koje karakterisu uslužnost sistema.

1.1 Primena sistema visoke pouzdanosti

Postoji više faktora koji su doveli do razvoja i širenja koncepta sistema otpornih na otkaze. Savremeni sistemi postaju sve složeniji, sa sve većim brojem sastavnih komponenti, što neminovno povećava verovatnoću pojave kvarova. Drugi bitan faktor je napredak VLSI tehnologije koja je mnoge tehnike otpornosti na otkaze učinila praktično izvodljivim. Konačno, mnogi sistemi koji su ranije bili mehanički sada su elektronski. Elektronski sistemi

su po pravilu manje pouzdani od mehaničkih, te su zbog toga neophodne dodatne mere za povećanje njihove pouzdanosti.

Postojeće primene sistema otpornih na otkaze mogu se svrstati u četiri primarne oblasti: aplikacije sa dugim vremenom života, aplikacije sa kritičnim izračunavanjima, aplikacije sa odloženim održavanjem, i aplikcije visoke dostupnosti.

1.2.1 Aplikacije sa dugim vremenom života

Primeri **aplikacija sa dugim vremenom života** su svemirske letelice i sateliti bez ljudske posade. Tipični zahtevi koji se postavljaju za ovakve aplikacije su da imaju verovatnoću od 0.95 da će biti operativne posle perioda od deset godina. Radi se o sistemima koji duži vremenski period moraju raditi samostalno, bez mogućnosti direktne intervencije radi popravke eventualnog kvara, te stoga trebaju postajati ugrađeni mehanizmi koji će omogućiti da u slučaju pojave kvara, sistem samostalno peruzme odgovarajuće aktivnosti kako bi se predupredio otkaz.

1.2.2 Aplikacije sa kriticnim izračunavanjima

Najšire primena sistema na otkaze su izračunavanja kritična za ljudsku bezbednost, očuvanje okoline, ili zaštitu opreme. Primeri uključuju sisteme za kontrolu leta aviona, vojne sisteme, i određene tipove industrijskih kontrolera. U **aplikacijama sa kriticnim izračunavanjima** neispravan rad sistema može da dovede do katastrofalnih posledica. Tipičan zahtev koji se postavlja pred ovakve sisteme je da imaju pouzdanost od 0.9_7 na kraju tročasovnog perioda. Glavni cilj u skoro svim aplikacijama sa kritičnim izračunavanjima je sprečiti da elektronika bude *slaba tačka* u sistemu.

1.2.3 Aplikacije sa odloženim održavanjem

Aplikacije sa odloženim održavanjem se najčesce javljaju u slučajevima kada su operacije održavanja (servisiranja) ekstremno skupe, neprikladne, ili teške za izvodjenje. Osnovni cilj je koristiti toleranciju otkaza ne bi li dozvolili odlaganje servisiranja do prikladnijeg i ekonomičnijeg vremena. Osoblje održavanja može obilaziti lokaciju mesečno i vršiti neophodne popravke. Između ovih servisnih obilazaka, sistem koristi toleranciju otkaza za neprekidno izvršavanje svojih zadataka.

1.2.4 Visoko dostupne aplikacije

Primeri sistema od kojih se zahteva visoka dostupnost su bankarski i drugi sistemi zasnovani na *on-line* opsluživanju korisnika**.** Korisnici ovakvih sistema, nakon što su zatražili uslugu očekuju veliku verovatnoću da budu opsluženi.

2.1 Kvar, Greska i Otkaz

Tri osnovna termina u projektovanju sistema otpornih na otkaze su kvar, greška i otkaz. Postoji uzročno posledična veza između kvara, greške i otkaza. Konkretno, kvarovi su uzrok grešaka, a greške su uzrok otkaza.

Kvar je fizički defekt, nepotpunost, ili proboj koji se javlja unutar neke hardverske ili softverske komponente. Kvar je takođe mana, slabost ili ostarelost određene hardverske ili softverske komponente. Primeri kvarova su kratki spojevi između električnih provodnika, prekidi provodnika i sl.. Primer softverskog ″kvara″, je programska petlja u koju kada se uđe, više se nemože izaći.

Greska je manifestacija kvara. Konkretno, greška je odstupanje od tačnosti ili ispravnosti. Na primer, pretpostavimo da na liniji postoji spoj koji rezultuje time da je linija uvek na nivou logičke 1. Fizički spoj je kvar u kolu. Ako se pojavi stanje koje zahteva da se linija prebaci na nivo logicke 0, vrednost na liniji ce biti pogrešna. Drugim rečima, ispravna vrednost na liniji bi bila logička 0, ali postojanje kvara će rezultovati pogrešnom vrednošću na liniji. S druge strane, ako je stanje kola takvo da na konkretnoj liniji treba biti nivo logičke 1, postojanje kvara ne uzrokuje grešku. Dakle, čak iako postoji kvar, greška ne mora da se ispoljava u svim situacijama.

Konačno, ako greška rezultira time da sistem neku svoju funkciju izvršava pogrešno, došlo je do **otkaza** sistema. Suštinski, otkaz je neizvršavanje neke akcije kako bi trebalo, ili kako je očekivano. Otkaz takođe predstavlja i izvršavanje neke funkcije u neodgovarajućem kvalitetu ili obimu.

Slika Sl. 1 ilustruje vezu između kvarova, grešaka i otkaza. Otkazi su uzrokovani greškama, koje su uzrokovane kvarovima.

Sl. 1 Kvar stvara grešku. Greška dovodi do otkaza sistema.

2.2 Uzroci kvarova

Kvarovi mogu biti rezultat raznovrsnih pojava unutar same elektronske komponente, u okruženju komponente ili mogu biti uzrokovani još za vreme projektovanja ili proizvodnje komponente ili sistema.

Mogući **uzroci kvarova** se mogu povezati sa problemima u četiri osnovne oblasti: specifikacija, realizacija, komponente i spoljašnji faktori.

Prvi mogući uzrok kvarova su **greške u specifikaciji.** Ovo uključuje netačne algoritme, arhitekture, ili pogrešne specifikacije u projektovanju hardvera i softvera.

Sledeći mogući uzrok kvarova su **greske u realizaciji.** Realizacija je proces transformacije hardverske i softverske specifikacije u hardver i softver. Realizacija može proizvesti kvarove zahvaljujući lošem projektu, lošem izboru komponenti, lošoj konstrukciji, ili greškama u kreiranju softvera.

Sledeći uzrok kvarova su **defekti komponenti**. Nesavršenosti u proizvodnji, slučajni otkazi uređaja, i istrošenost komponenti su tipični primeri defekta komponenti. Defekt može biti rezultat preloma veza u kolu ili korozije metala.

Poslednji mogući uzrok kvarova su **spoljašnji poremećaji**, kao na primer, radijacija, elektromagnetna interferencija, greške operatera, i ekstremne promene u radnoj sredini. Naime, elektronski sistemi su veoma osetljivi na temperaturske varijacije, elektrostatičke izvore kao sto su munje ili drugi vremenski efekti. Takođe, greške operatera se smatraju za spoljasnje poremećaje jer je, gledano sa fizičke strane samog sistema, operater spoljašnji uticaj. Naime, operater može sistemu zadati pogrešne komande, koje, na kraju, dovode do otkaza sistema.

2.3 Karakteristike kvarova

Priroda kvara opisuje tip kvara. Kvar može biti hardverski ili softverski, takođe on može biti u analognom ili digitalnom kolu, i sl. Greška u programu je primer softverkog kvara.

Trajanje kvara opisuje dužinu vremenskog perioda u kom je kvar aktivan:

- **Stalni** ili **permanentni kvar** koji ostaje neograničeno dugo ukoliko se ne preduzmu korektivne mere.
- **Prolazni ili tranzijentni kvar** se može pojaviti i zatim nestati u veoma kratkom vremenskom periodu.
- **Povremeni kvar** je onaj koji se pojavljuje, nestaje, i zatim se ponovo, regularno pojavljuje.

Rasprostranjenost kvara opisuje da li je uticaj kvara lokalizovan na konkretni hardverski ili softverski modulu, ili on globalno utiče na rad sistema.

2.4 Projektantski prilazi eliminaciji kvarova

Postoje tri osnovne tehnike za poboljšavanje ili zadržavanje nominalnih performansi sistema u okruženju u kome su mogući kvarovi: izbegavanje kvarova, maskiranje kvarova, i tolerancija kvarova.

Izbegavanje kvarova je bilo koja tehnika koja se koristi za sprečavanje pojave kvarova. Izbegavanje kvarova može uključivati takve procese kao što su preispitivanja projekta, zaštita komponenti od dejstva visokog napona, testiranje, i druge metode kontrole kvaliteta. Naime, ako se pregled projekta izvrši korektno mnoge greške nastale u specifikaciji se mogu otkloniti. Takođe, ako se sistem oklopi mogu se sprečiti spoljašnji poremećaji. A isto tako, ako se sistem pažljivo testira, mnogi kvarovi se na vreme mogu otkriti i ukloniti pre nego što započne eksplatacija sistema.

Maskiranje kvarova je bilo koja tehnika koji sprečava da kvar u sistemu proizvede grešku unutar strukture sistema. Primeri maskiranja kvara su: memorije sa mogućnošću korekcije grešaka, ili većinsko izglasavanje u sistemima gde postoje tri istovetna modula, i gde dva, na osnovu većine mogu maskirati treći neispravan modul.

Tolerancija kvara je sposobnost sistema da nastavi sa izvršavanjem svojih zadataka i posle pojave kvara. Osnovni cilj tolerancije kvara je sprečavanje pojave otkaza sistema usled pojave kvara. Tolerancija kvara se može postići mnogim tehnikama. Maskiranje kvara je jedna takva tehnika. Drugi tehnika se sastoji u tome da se kvar detektuje i locira, a da se zatim, sistem rekonfiguriše kako bi se eliminisao uticaj pokvarene komponente ili modula. Uopšteno govoreći, r**ekonfiguracija** je proces eliminisanja pokvarenog entiteta iz sistema i vraćanje sistema u operativno stanje. Rekonfiguracija uključuje sledeće aktivnosti:

1. **Detekcija kvara** je proces otkrivanja kvara koji se pojavio.
2. **Lokalizacija kvara** je proces određivanja gde se kvar pojavio.
3. **Izolacija kvara** je proces izolovanja kvara i sprečavanja da se efekti tog kvara prošire kroz sistem.
4. **Sanacija kvara** je proces koji obezbedjuje da se i u prisustvu kvara, a uz pomoć rekonfiguracije, zadrži ili ponovo uspostavi operativni status sistema.

3. Tehnike projektovanja za postizanje visoke pouzdanosti

3.1 Redundansa

Redundansa predstavlja dodavanje informacija, resursa, ili vremena iznad nivoa koji je potreban za normalan rad sistema. Redundansa može imati više oblika:

1. **Hardverska redundansa** predstavlja dodavanje hardvera, obično za detekciju ili toleranciju kvarova.
2. **Softverska redundansa** predstavlja dodavanje softvera, iznad onog neophodnog za obavljanje zadatih funkcija, a u cilju detekcije i tolerancije kvarova.
3. **Informaciona redundansa** je dodavanje dodatnih informacija iznad zahtevanih za izvrsavanje zadate funkcije.
4. **Vremenska redundansa** koristi dodatno vreme za izvršavanje funkcija sistema tako da detekcija kvara i često i tolerancija kvara mogu biti postignute.

3.2 Hardverska redundansa

Fizičko umnožavanje hardvera je najčešći oblik redundanse koji se danas koristi u cilju postizanja otpornosti na otkaze. Postoje tri osnovna oblika hardverske redundanse: pasivna, aktivna i hibridna.

Pasivne tehnike koriste princip maskiranja kvara da sakriju pojavu kvara i da spreče da kvar uslovi pojavu greske.

Aktivne tehnike postiže toleranciju kvara detektovanjem postojanja kvara i izvođenjem akcija uklanjanja (ili izolovanjaa) pokvarenog hardvera iz sistema. Aktivna hardverska redundansa koristi detekciju kvara, lokaciju kvara, i sanaciju kvara u cilju postizanja tolerancije kvara.

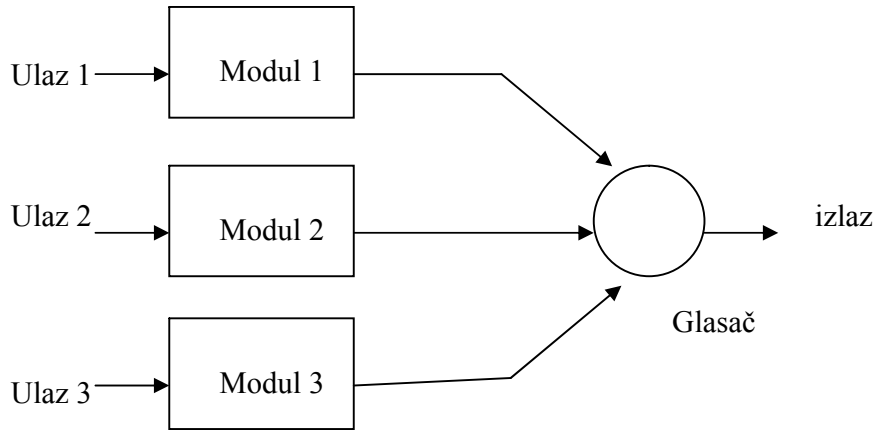
Hibridna tehnika koristi dobre osobine pasivnog i aktivnog prilaza. Maskiranje kvara se koristi u hibridnim sistemima da spreči generisanje pogrešnih rezultata. Onda kada maskiranje nije više u stanju da obezbedi ispravan rad sistema, primenjuje se neka aktivna tehnika kako bi se kvar locirao i sanirao zamenom pokvarenog hardvera rezervnim.

3.2.1 Pasivna hardverska redundansa

Pasivna hardverska redundansa se oslanja na mehanizam glasanja da bi maskirala pojavu kvarova. Većina pasivnih tehnika je zasnovana na principu većinskog glasanja. Kao što je već pomenuto, pasivne tehnike postižu toleranciju kvara bez potrebe detekcije kvara ili rekonfiguracije sistema.

Trostruka modularna redundansa

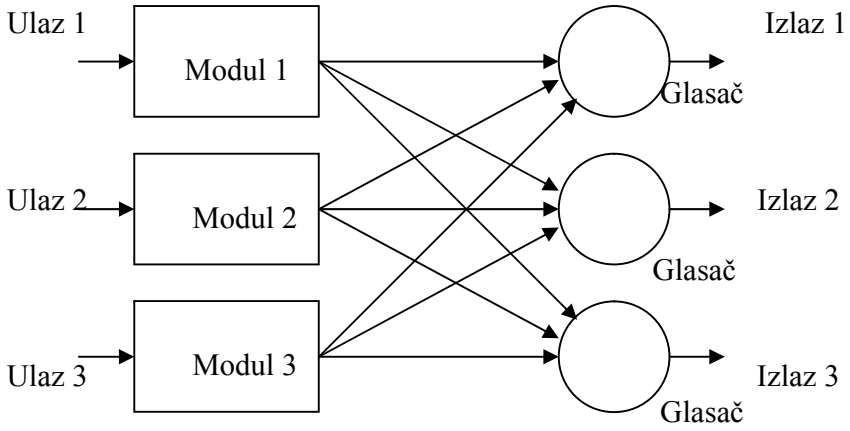
Osnovni princip **trostruke modularne redundanse (TMR),** kao i što se vidi sa Sl. 2, je utrostručavanje hardvera i primena većinskog glasanja pri određivanju izlaza sistema. Ako se jedan modul pokvari, dva preostala ispravna modula pri većinskom glasanju maskiraju rezultate neispravnog modula.



Sl. 2 Model pasivne hardverske redundanse sa tri istovetna modula i jednim glasacem.

Osnovni problem sa TMR-om je glasač; ako glasač otkaže, ceo sistem će otkazati. Drugim rečima, pouzdanost najjednostavnijeg oblika TMR-a nije bolja od pouzdanosti glasača. Bilo koja pojedinačna komponenta unutar sistema čiji otkaz dovodi do otkaza sistema zove se **jedinstvena tačka otkaza.**

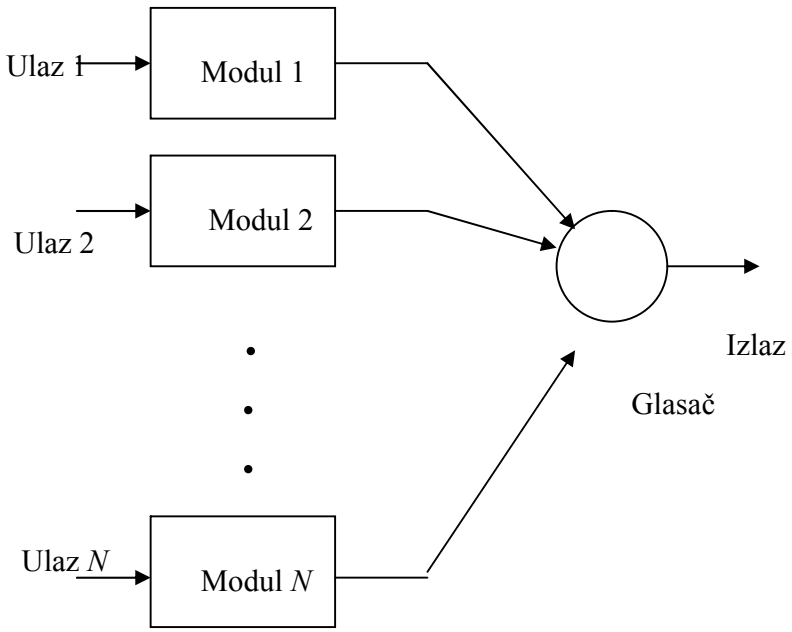
Nekoliko tehnika se može upotrebiti za prevazilaženje efekta otkaza glasača. Jedan prilaz, prikazan na Sl. 3, je utrostručavanje samog glasača i obezbedjivanje tri nezavisna izlaza. (Pretpostavka je da se tri izlaza vode na tri ulaza podistema koji takođe koristi trostruku modularnu redundansu.)



Sl. 3 Model prevazilazenja jedinstvene tacke otkaza koriscenjem tri nezavisna glasaca i tri izlaza.

N-Modularna redundansa

N-Modularna redundansa (NMR) predstavlja generalizaciju TMR-a. NMR je zsnovana na istom principu principe kao i TMR, ali koristi *N* modula. U većini slučajeva, *N* je izabrano kao neki neparan broj tako da se mogu primeniti principi većinskog glasanja. Prednost koriscenja *N* modula umesto samo tri je da se može tolerisati kvar više modula. Na Sl. 4 prikazana je principska sema NMR-a.

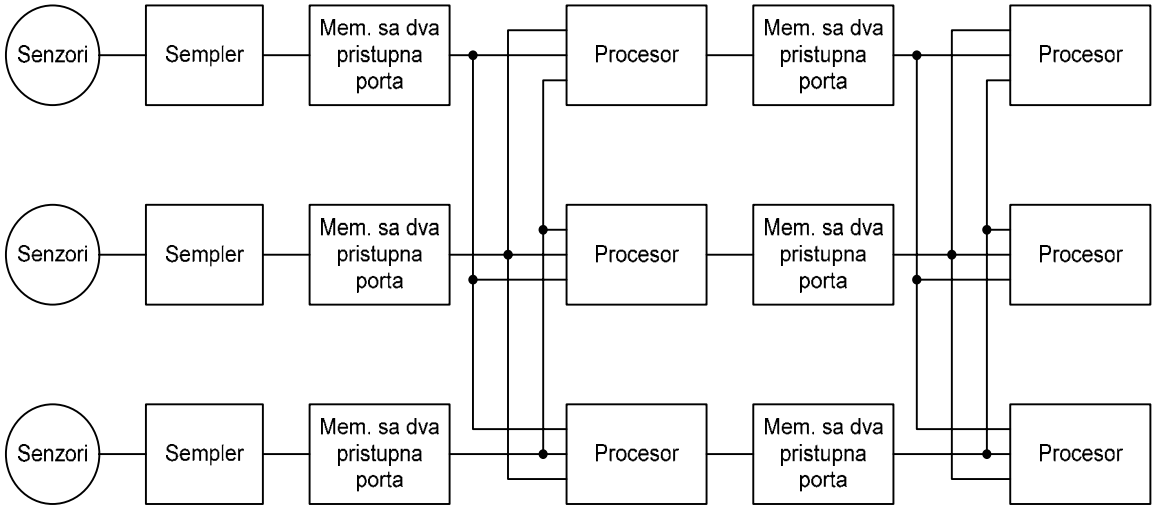


Sl. 4 Model pasivne redundanse koji koristi N istovetnih modula i jedan glasac.

Tehnike glasanja

Sam proces glasanja uključuje nekoliko proceduralnih problema. Prvi je odlučivanje da li će biti iskorišćen hardverski glasač, ili ce proces glasanja biti realizovan u softveru.

Primer mikroprocesorskog sistema koji koristi softversko glasanje prikazan je na Sl. 5.



Sl. 5 TMR sa glasanjem implementiranim u softveru.

Vrednosti sa senzora (koji su takođe triplicirani) se sempluju (odmeravaju) i čuvaju u svakoj od tri memorije sa dva porta. Svaki procesor može čitati iz svake memorije da bi dobio sve vrednosti sa senzora. Kada su vrednosti sa senzora pročitane iz memorije, svaki procesor glasa nezavisno da bi odlučio koji od tri rezultata da koristi u svojim proračunima. Posle završetka izračunavanja, svaki procesor smešta svoje rezultate u drugi set memorija sa dva porta, tako da drugi procesori, koji su deo istog ili nekog drugog sistema, mogu da

koriste ove rezultate. Ovaj princip predstavlja softversku realizaciju TMR-a sa trostrukim glasačem.

Drugi problem sa praktičnom realizacijom glasanja je taj da se, na primer, tri rezultata u TMR sistemu ne slažu u potpunosti, čak i u slučaju kad ne postoji kvar. Senzori koji se koriste u mnogim upravljačkim sistemima su fabrički proizvedeni tako da se njihove vrednosti skoro nikad ne slažu u potpunosti. Takodje, A/D konvertor može generisati vrednosti koje se razlikuju u bitu najmanje težine, čak iako se isti signal propusti kroz isti konvertor više puta. Kad se ove vrednosti koje se malo razlikuju procesiraju, neslaganje može dobiti veće razmere. Konačno, većinski glasač može utvrditi da ne postoje dva rezultata koja se slažu u TMR sistemu, iako sistem radi ispravno.

Jedan prilaz koji ublažava problem neslaganja rezultata je tehnika **selekcije srednje vrednosti.** U osnovi, tehnika selekcije srednje vrednosti, od tri ponuđene vrednosti u TMR sistemu bira onu koja se nalazi između preostale dve.

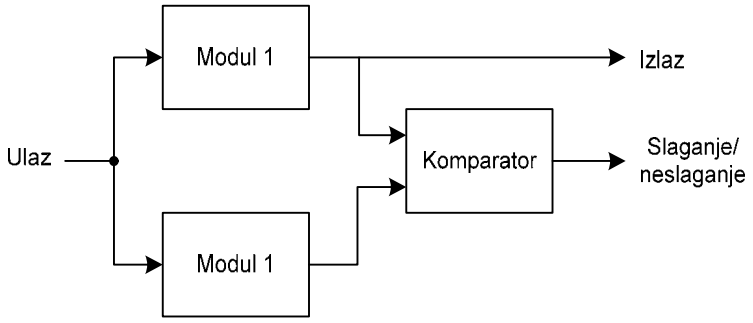
Sledeći pristup koji se obično koristi kad se binarne vrednosti ne slažu u potpunosti je da se ignorišu bitovi najmanje težine. Drugim recima, većinsko glasanje se izvodi sa *k* bitova najveće težine. Broj bitova koji se ignorišu zavisi od aplikacije i funkcija je tačnosti upotrebljenih komponenti.

3.2.2 Aktivna hardverska redundansa

Tehnika aktivne hardverske redundanse nastoji da postigne toleranciju kvara pomoću detekcije kvara, lokacije kvara i sanacije kvara. Aktivna hardverska redundansa ne poseduje osobinu maskiranja kvara i primenjuje se kod sistema koji mogu tolerisati privremene, pogrešne rezultate, pod uslovom se sistem rekonfiguriše i povrati svoj operativni status u zadovoljavajućem vremenu.

Udvostručavanje sa poredjenjem

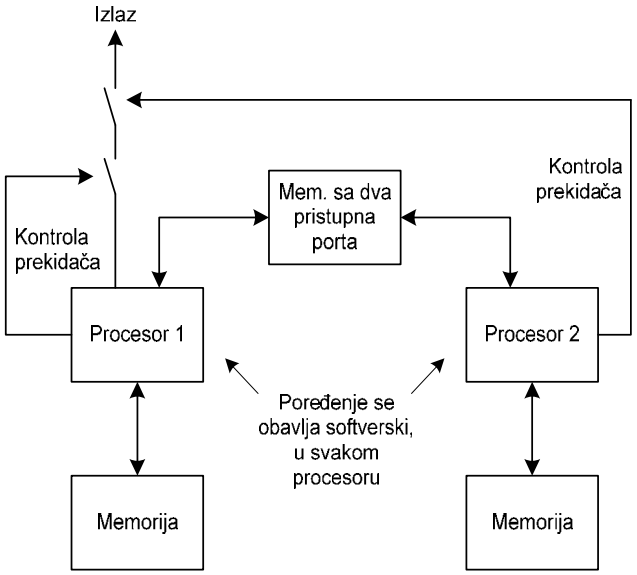
Osnovni princip ***udvostručavanja sa poredjenjem*****,** prikazanog na Sl. 6 je ugraditi dva identična hardverska modula, pustiti ih da u paraleli rade ista izračunavanja, i zatim uporediti njihove rezultate. U slučaju neslaganja, generiše se signal greške. U svom najosnovnijem obliku, princip udvostručavanja može samo detektovati postojanje kvara, a ne i tolerisati ga, jer ne postoji način za određivanje koji je od dva modula u kvaru.



Sl. 6 Model aktivne hardverske redundanse, gde se posle paralelne obrade rezultati uporedjuju.

Opisani metod aktivne redundanse poseduje nekoliko potencijalnih problema. Naime, otkaz ulaznog uređaja ili linija preko kojih ulazni signali moraju biti preneseni do modula će rezultovati time da oba modula proizvedu isti, pogrešan rezultat. Konačno, kvarovi u komparatoru mogu prouzrukovati signal greške iako greška ne postoji, ili, još gore, komparator se može tako pokvariti tako da eventualni kvar u modulima ne bude detektovan.

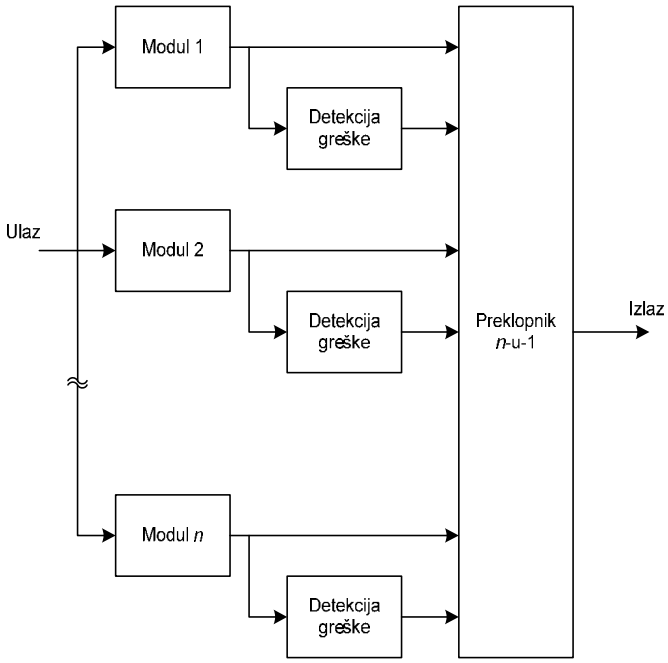
Tehnika koja može biti upotrebljena kod dvo-procesorskih sistema za prevazilaženje upravo spomenutih problema prikazana je na Sl. 7. Neophodna poređenja mogu biti realizovana i softverski. Da bi se generisao izlaz, oba procesora se moraju složiti da se rezultati podudaraju.



Sl. 7 Model aktivne hardverske redundanse sa paralelnom obradom i softverski ostvarenim poredjenjem.

Spremna rezerva

U ovoj formi aktivne hardverske redundanse, prikazane na Sl. 8, jedan modul je operativan, a jedan ili više služe kao rezerva. Ako je kvar detektovan i lociran, pokvareni modul se uklanja iz sistema i zamenjuje se rezervnim.



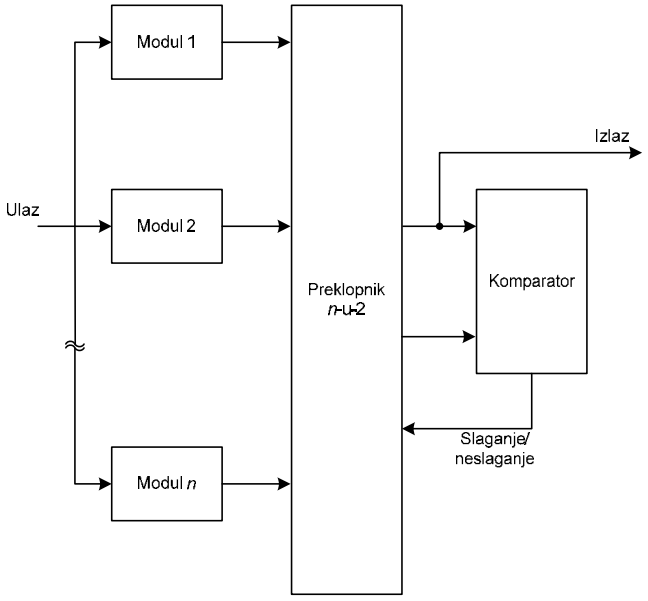
Sl. 8 Model aktivne hardverske redundanse sa jednim aktivnim i *n-1* modula koji služe kao rezerva.

Ova metoda može vratiti sistemu punu operativnu sposobnost posle pojave kvara, ali izaziva trenutno narušavanje performansi dok se izvodi rekonfiguracija. Ako ovo narušavanje rada mora biti minimizirano, primenjuje se ***"vruća spremna rezerva"***. Naime, kod ove tehnike, rezervni moduli rade sinhrono sa aktivnim modulima, i spremni su da preuzmu sistem bilo kada. Nasuprot ovome, imamo ***"hladnu spremnu rezervu"*** kod koje su rezervni moduli neaktivni, sve dok se ne javi potreba da zamene pokvareni modul. Nedostatak *"hladne spremne rezerve"* je vreme koje je potrebno da se modul priključi na napajanje i da se uvede u aktivan rad. Prednost je da rezervni moduli nisu priključeni na napajanje, a time i ne troše energiju, sve dok se ne javi potreba za zamenu pokvarenog modula.

Ključna komponenta opisanog pristupa je način na koji se detektuje greška i locira neispravni modul. Dve tehnike koje se mogu koristiti za ovu namenu biće opisane u nastavku.

Par i rezerva

"Par i rezerva" tehnika koristi prednosti i *"spremne rezerve"* i *udvostručavanja sa poredjenjem*. U osnovi, *"par i rezerva"* prilaz koristi *"spremnu rezervu",* ali tako da uvek dva modula rade u paraleli sve vreme, i njihovi rezultati se upoređuju u cilju detekcije greške. Signal greške iz komparatora je upotrebljen da inicira proces rekonfiguracije koji uklanja pokvarene module i zamenjuje ih rezervnim. Treba napomenuti da se, iako je pokvaren jedan modul, iz sistema odstranjuje i njegov par, prvenstveno zbog nemogućnosti detekcije koji je od njih u kvaru.



Sl. 9 Model tehnike par-i-rezerva, koji kombinuje udvostručavanje sa poređenjem i spremnu rezervu.

Watchdog tajmer

Oblik aktivne hardverske redundanse koji je veoma koristan za detekciju kvarova u sistemu je **watchdog tajmer.** Princip watchdog tajmera je da je izostanak neke akcije indikator kvara. Watchdog tajmer je tajmer koji se mora periodično resetovati. Osnovna je pretpostavka da je sistem ispravan ako je sposoban da periodično izvršava ovu jednostavnu funkciju.

Watchdog tajmer se može primeniti za detekciju kvarova i u hardveru i u softveru sistema.

3.2.3 Hibridna hardverska redundansa

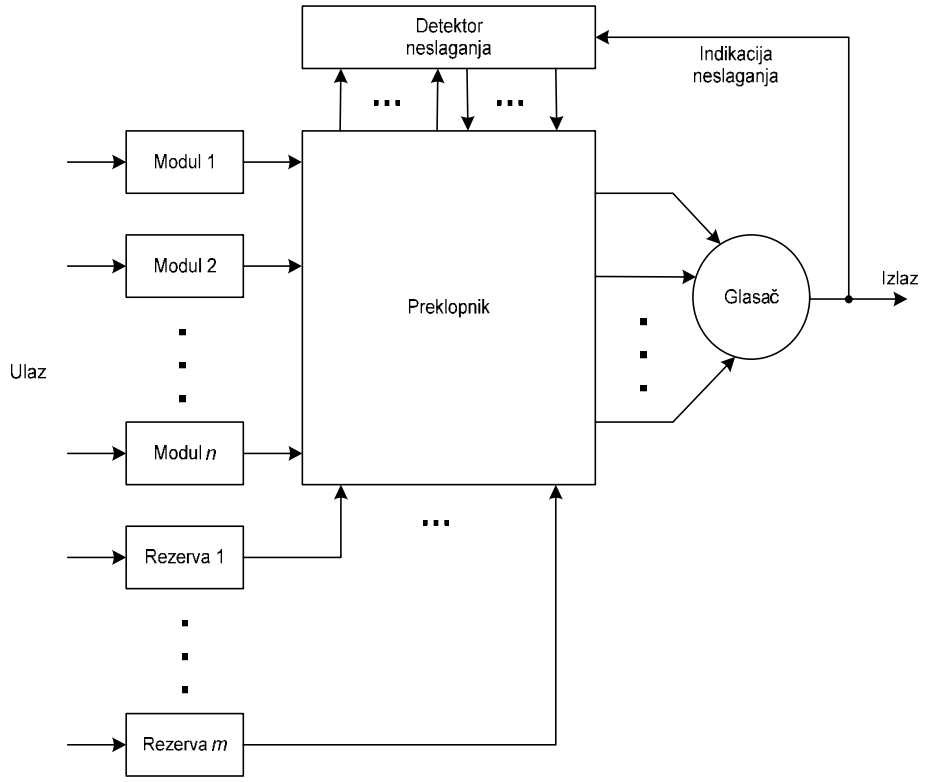
U osnovi, hibridna hardverska redundansa koristi dobre osobine i pasivne i aktivne hardverske redundanse. Maskiranje kvara se koristi da spreči sistem da proizvede pogrešne rezultate, a detekcija, lokacija i sanacija kvara se koriste za rekonfiguraciju sistema u slučaju pojave kvara.

N-Modularna redundansa sa rezervama

Ideja ***N*-Modularne redundanse (NMR) sa rezervama** je obezbediti konfiguraciju čiju osnovu čini *N* modula uključenih u glasanje, ili neku vrstu glasanja. Glasanje obezbeđuje maskiranje pokvarenog modula za vreme dok se pokvareni modul ne zameni rezervnim.

Primer ove tehnike prikazan je na Sl. 10. Sistem ostaje u svojoj osnovnoj NMR konfiguraciji (operativni su Modul 1, Modul 2 do Modul *n*) sve dok glasač ne utvrdi da postoji neispravni modul. Modul koji se ne slaže sa većinom se označava kao pokvaren i uklanja iz NMR konfiguracije (zadatak detektora neslaganja). Rezervni modul se tada uključuje i zauzima mesto pokvarenog. Glasanje se uvek izvodi izmedju aktivnih članova NMR-a, cime se maskiraju kvarovi i obezbedjuje neprekidan ispravan rad sistema.

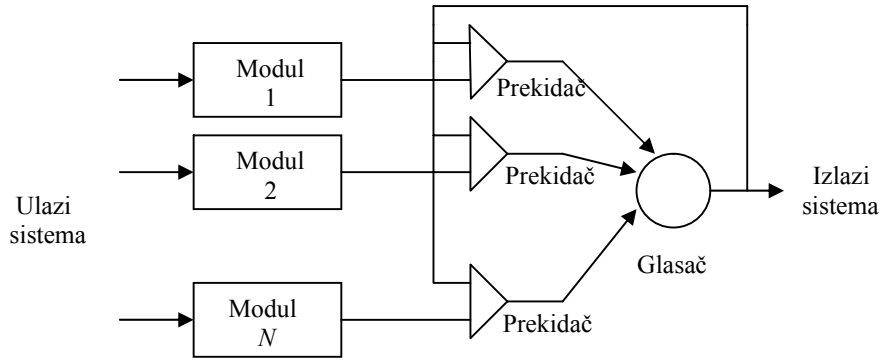
U poređenju sa pasivnom NMR tehnikom, NMR tehnika u kombinaciji sa rezervama zahteva manji broj modula za ostvarivanje istog nivoa pouzdanosti. Na primer, za tolerisanje dva nispravna modula kod bazične NMR tehnike potrebno je 5 modula, dok je kod NMR sa rezervom potrebno 4.



Sl. 10 Model hibridne hardverske redundanse koji pored osnove koju cini NMR ima i rezervne module.

Redundansa sa samo-izbacivanjem

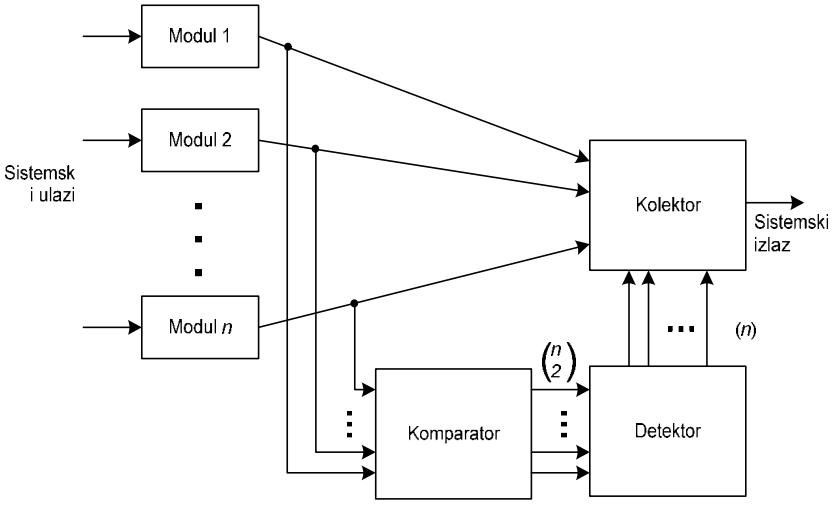
Osnovni princip ovog prilaza je sličan prilazu koji imamo kod *NMR-a sa rezervama*. Razlika je u tome da kod tehnike *samo-izbacivanja* sve jedinice aktivno učestvuju u radu sistema, za razliku od NMR-a gde rezervne jedinice nisu aktivne sve dok se ne pojavi kvar. Svaki od *N* identičnih modula je napravljen sa sposobnošću da ukloni sebe iz sistema u slučaju da se njegov izlaz razlikuje od izglasanog izlaza sistema. Model sistema koji koristi *tehniku samo-izbacivanja* prikazan je na Sl. 11.



Sl. 11 Model hibridne hardverske redundanse koji koristi tehniku samo-izbacivanja neispravnih modula.

Redundansa sa "prosejavanjem" modula

Ova tehnika takođe koristi *N* identičnih modula koji su povezani u sistem korišćenjem specijalnih bloka: komparator, detektor i kolektor. Model sistema koji koristi redundansu sa prosejavanjem modula predstavljen je na Sl. 12.



Sl. 12 Model hibridne hardverske redundanse sa "prosejavanjem" modula.

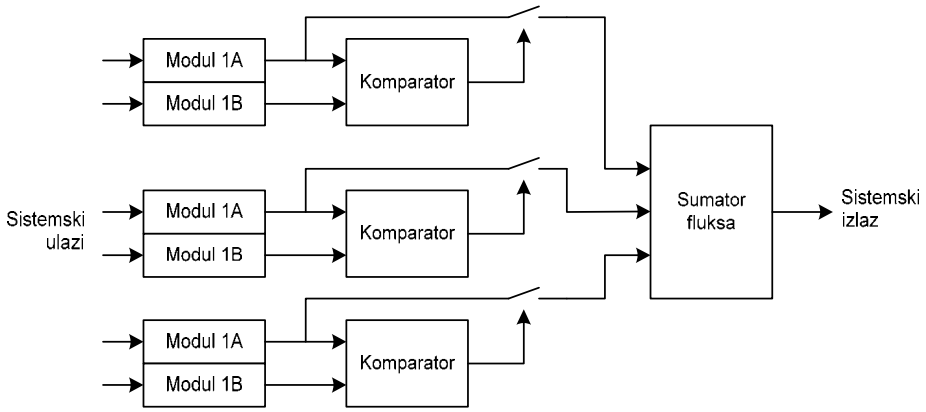
Uloga komparatora je da uporedjuje izlaz svakog modula sa izlazima ostalih modula, i on proizvodi po jedan signal za svako poredjenje koje je izvrsio. (Za *n* modula, ukupan broj poređenja iznosi $\binom{n}{2}$. Signal koji je generisao komparator je 1 ako se dve jedinice koje su poređene ne slažu; i 0 u suprutnom. Uloga detektora je da odredi koja neslaganja je komparator prijavio, i da onesposobi jedinicu koja se ne slaže sa **većinom** preostalih modula. Detektor proizvodi po jedan signal za svaki modul, i vrednost tog signala je 1 ako se dati

modul ne slaže sa većinom preostalih modula, i 0 u suprutnom. Uloga kolektora je da proizvede izlaz sistema, koristeći pritom izlaze pojedinačnih modula i signale iz detektora koji indiciraju koji je od modula pokvaren. Modulu koji je identifikovan kao pokvaren nije dozvoljeno da utiče na izlaz sistema.

Utrostručeno-dvostruka arhitektura

Ova tehnika kombinuje udvostručavanje sa poredjenjem i trostruku modularnu redundansu. Po dva modula rade u tandemu. Komparator pridružen paru modula poredi njihove rezultate i isključuje tandem iz sistema ukoliko ustanovi neslaganje. Upotreba TMR-a dozvoljava maskiranje kvara, i neprekidan, pravilan rad sa jednim neispravnim modulom.

Upotreba *udvostručavanja sa poredjenjem* omogucava detekciju kvarova i uklanjanje neispravnih modula iz sistema glasanja unutar TMR-a. Ovakva arhitektura se obično koristi u upravljačkim aplikacijama u kojima se kao glasač može koristiti proces "sumiranja fluksa". Naime, izlaz se dobija nekom vrstom sumiranja rezultata generisanih od strane pojedinačnih tandema. Za vreme dok su sva tri tandema ispravna, učešće izlaza svakog tandema u izlazu sistema jednako je 1/3 (tj. izlaz je jednak $I=1/3*T_1+1/3*T_2+1/3*T_3$, gde su T_1,T_2 i T_3 izlazi pojedinačnih tandema.) Nakon "ispadanja" jednog od tandema, učešće preostala dva tandema se povećava na 1/2, tj. $I= 1/2*T_1+1/2*T_2$, pod pretpostavkom da je isključen tandem T_3. Na Sl. 13 prikazan je model jednog ovakvog sistema.



Sl. 13 Model hibridne hardverske redundanse koji koristi *utrostruceno-dvostruku* arhitekturu.

3.3 Informaciona redundansa

Informaciona redundansa predstavlja dodavanje redudantnih informacija podacima u cilju omogućavanja detekcije kvara, maskiranja kvara, ili čak tolerancije kvara. Dobri primeri informacione redundanse su kodovi za detekciju, kao i kodovi za korekciju grešaka, koji se formiraju dodavanjem redudantnih informacija rečima, ili prevodjenjem reči u neki novi oblik koji sadrži redudante informacije.

Kod je način predstavljanja informacija, ili podataka, korišćenjem dobro definisanog skupa pravila.

Kodna reč je skup simbola koji su iskorišćeni da predstavi odredjeni deo podatka zasnovan na definisanom kodu. **Binarni kod** je onaj u kome se simboli za formiranje kodnih reči sastoje samo od 0 i 1.

Kodiranje je proces odredjivanja odgovarajuče kodne reči za zadati podatak. Drugim rečima, proces kodiranja uzima originalni podatak i kovertuje ga kao kodnu reč koristeći pravila koda.

Dekodiranje je proces vraćanja originalnog podatka iz kodne reči. Drugim rečima, proces dekodiranja uzima kodnu reč i odredjuje podatak koji ona predstavlja.

Kod za detekciju greške je specificna reprezentacija koja omogućava detekciju grešaka unutar kodne reči.

Kod za korekciju grešaka predstavlja kod koji ima sposobnost da ispravlja greške. Ovakvi kodovi se obično opisuju brojem bitskih grešaka koje mogu ispraviti.

Osnovni pojam pri karakterizaciji kodova za detekciju i kodova za korekciju grešaka je **Hamingovo rastojanje.** Hamingovo rastojanje izmedju bilo koje dve binarne reči je broj bitskih pozicija u kojima se te dve reci razlikuju. Za svaki kod se može definisati **kodno rastojanje** kao minimalno Hamingovo rastojanje izmedju bilo koje dve važeće kodne reči. U suštini, kod može korigovati i do c bitskih grešaka i detektovati i do d dodatnih bitskih grešaka ako i samo ako važi:

$$2c + d + 1 \leq H_d$$

gde je H_d kodno rastojanje.

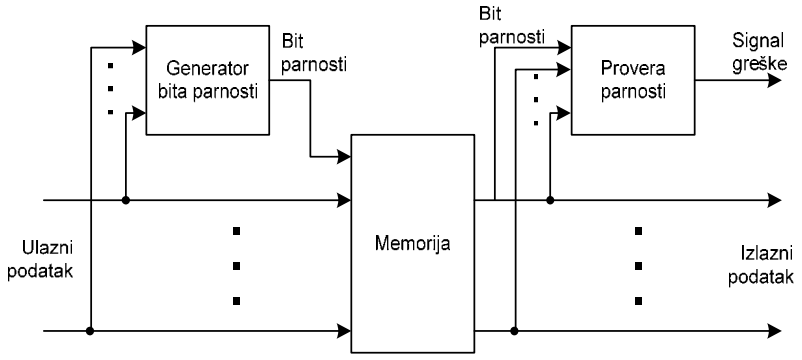
Razdvojiv kod je onaj u kome je originalna informacija pridružena sa novom informacijom u cilju formiranja kodne reči, omogućavajući tako da se proces dekodiranja sastoji od jednostavnog uklanjanja dodatnih informacija i zadržavanja originalnog podatka. Drugim rečima, originalan podatak se dobija iz kodne reci uklanjajući dodatne bitove, koji se zovi kodni ili kontrolni bitovi, i zadržavajući samo one koji predstavljaju originalnu informaciju. **Nerazdvojivi kodovi** ne poseduju mogucnost razdvajanja, i samim tim zahtevaju složenije procedure dekodiranja.

3.3.1 Kodovi parnosti

Najjednostavniji oblik koda je **kod parnosti.** Osnovni princip parnosti je veoma jednostavan, ali su moguće i brojne varijacije osnovne ideje. Jedno-bitni kodovi parnosti zahtevaju dodavanje dodatnog bita binarnoj reči, tako da dobijena kodna rec ima ili paran ili neparan broj jedinica. Ako ovaj dodatni bit rezultira time da je ukupan broj jedinica u kodnoj reči neparan, kod je označen kao *kod neparnosti*; ako je ukupan broj jedinica paran, kod se zove *kod parnosti*. Ako kodna reč sa neparnim brojem jedinica doživi grešku na jednom od svojih bitova, ona postaje parna, a takodje ako je kodna reč bila parna, posle pojave greške postaje neparna. Prema tome, jedno bitna greška može biti detektovana prebrojavanjem jedinica u kodnoj reči.

Najčešća primena kodova parnosti je u memorijama računarskih sistema. Pre nego što se upiše u memoriju, podatak se kodira. Kodiranje se sastoji u dodavanju jednog bita sa ciljem da rezultujuća reč ima paran broj jedinica. Kada se potom podatak čita iz memorije, parnost se mora proveriti da bi se verifikovalo da se nije promenila kao posledica neke greške.

Na Sl. 14 je prikazana organizacija memorije koja koristi jedno-bitni kod parnosti, i kao sto se vidi, ovakvi kodovi zahtevaju dodatni hardver. Naime, sama memorija mora sadržati po jedan dodatni bit za svaku reč, a i ostali hardver mora biti projektovan tako da može generisati i proveravati bit parnosti. Iz ovoga vidimo da informaciona redundansa najčešće zahteva i hardversku redundansu.



Sl. 14 Organizacija memorije sa upotrebom koda-parnosti.

Najveći problem sa jedno-bitnim kodovima parnosti je njihova nesposobnost da garantuju detekciju više-bitnih gresaka. Postoji pet osnovnih načina modifikacije tehnike parnosti koji unapredjuju sposobnost detekcije grešaka:

1. Bit parnosti za svaku reč (reč se sastoji od jednog ili više bajtova)
2. Bit parnosti za svaki bajt
3. Bit parnosti za svaki *čip* (čip je skup od četiri bita)
4. Bit parnosti za više čipova
5. Parnost sa preplitanjem

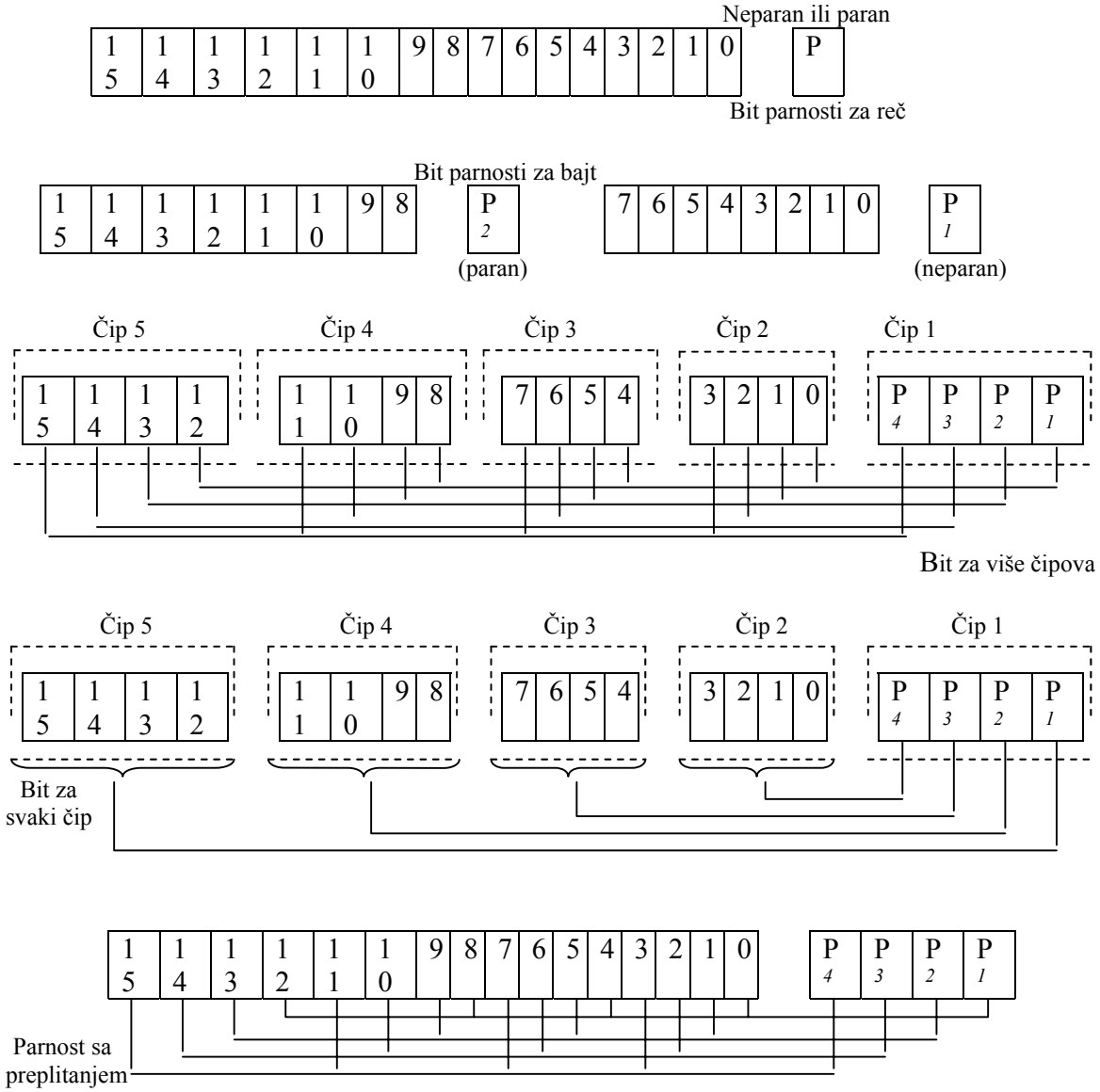
Osnovni princip svakog od ovih prilaza prikazan je na Sl. 15.

Bit parnosti za svaku reč ima osnovnu ideju da se svakoj reči doda po jedan bit parnosti. Medjutim, ako se zbog kvara magistrale ili bafera, svi bitovi neke reči, uključujući i bit parnosti, trajno postave na 1 ili 0, takva greška može proći neprimećena.

U **bit za svaki bajt** tehnici koriste se dva bita parnosti za dva razdvojena dela originalne reči. Tehnika se zove "bit za svaki bajt", ali treba reći da grupe parnosti (skup bitova kojima se pridruzuje jedinstveni bit parnosti naziva se *grupa parnosti*) mogu imati bilo koji broj bitova, a ne samo 8 kao u bajtu. Pri tome, broj informacionih bitova koji su pridruženi svakom bitu parnosti je jednak, a takodje, parnost jedne grupe treba da bude *parna,* a parnost druge grupe treba da je *neparna.* Osnovna prednost ovog prilaza je da ima sposobnost da detektuje stanje kad usled nekog kvara svi bitovi jedne reči imaju vrednost 1, ili vrednost 0. Takodje, ova tehnika obezbedjuje dodatnu zaštitu od više-bitnih grešaka: na primer, dvo-bitna greška će biti detektovana pod uslovom da je jedan bit iz prve (*parne*) grupe parnosti, a drugi bit iz druge (*neparne*) grupe.

Osnovni nedostatak i *bit za svaku reč* i *bit za svaki bajt* tehnika je nedovoljna efikasnost pri detekciji više-bitnih gresaka.

Osnovni princip **bit parnosti za više čipova** prikazan je na Sl. 15 i sadržan je u tome da se po jednom bitu iz svakog čipa pridruži jedinstveni bit parnosti. (Memorije su obično sastavljene od individualnih *čipova,* pri cemu svaki od njih sadrži odredjen broj bitova. Radi jednostavnijeg izlaganja usvojeno je da je broj bitova po čipu 4.) Važno je uočiti da grupe parnosti uključuju po jedan, i samo jedan, bit iz svakog čipa. Nedostatak ovakvog metoda je da se otkaz celog cipa detektuje, ali se pogrešan čip ne može locirati. Naime, otkaz bilo kojeg čipa uzrokuje da sve grupe parnosti imaju grešku, tako da uzrok problema ne može biti identifikovan.



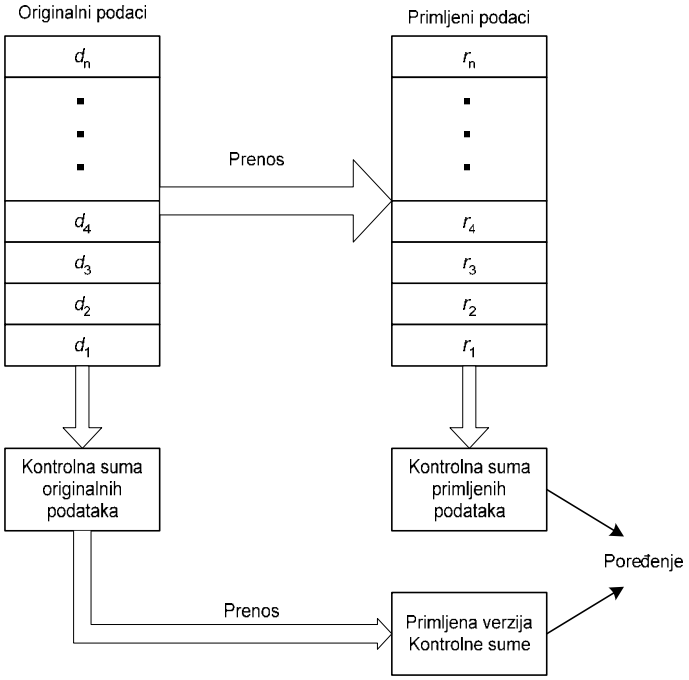
Sl. 15 Kodovi parnosti: bit parnosti za svaku reč; bit parnosti za svaki bajt; bit parnosti za više cipova; bit parnosti za svaki cip; parnost sa preplitanjem.

Procedura koja prevazilazi ovaj problem je **bit parnosti za svaki čip.** Ovde je svakom čipu pridružen jedan bit parnosti. Ako neki bit postane pogrešan, postojanje greške je detektovano, i čip koji sadrži pogrešan bit može biti identifikovan. Osnovni nedostatak ovakvog metoda je osetljivost na otkaz celog čipa. Zato što osnovni kod parnosti može detektovati samo pojedinačne greške, višestruka greška, kao što je recimo otkaz celog čipa moze proći neprimećeno.

Parnost sa preplitanjem je slična formi *bit parnosti za više čipova,* ali sa jednom ključnom razlikom. Naime, kod ove metode grupe parnosti su formirane bez obračanja pažnje na fizičku organizaciju memorije. Bitovi koji čine podatak su podeljeni u jednake grupe, i po jedan bit parnosti je dodeljen svakoj grupi. Bitovi u svakoj grupi su tako raspoređeni da ne postoje dva susedna bita koja pripadaju istoj grupi parnosti.

3.3.2 Kontrolne-sume (Checksums)

Kontrolna-suma predstavlja vrstu razdvojivog koda koji se koristi kada se blokovi podataka (poruke) prenose iz jedne tačke u drugu. **Kontrolna-suma** predstavlja odredjenu količinu informacija koja se na predajnoj strani dodaje bloku podataka radi lakše detekcije grešaka na prijemnoj strani. Četiri osnovna tipa kontrolnih-suma se mogu koristiti: kontrolna suma jednostruke-tačnosti, kontrolna-suma dvostruke-tačnost, kontrolna suma tipa *Honeywell*, i kontrolna-suma ostatka.



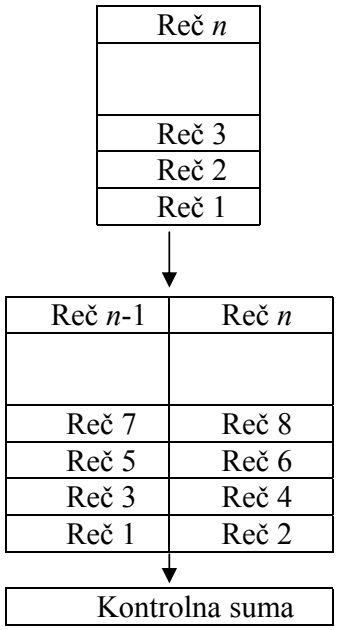
Sl. 16 Prenosa podataka sa upotrebom kontrolne sume.

Osnovni princip kontrolne-sume prikazan je na Sl. 16. Na predajnoj strani, bloku podataka koji se prenosi pridodaje se kontrolna suma. Kada su na odredištu primljeni podaci, kontrolna suma se ponovo generiše. Ova ponovo generisana kontrolna-suma i originalna kontrolna-suma se upoređuju da bi odredili da li je pri prenosu došlo do pojave greske u podacima.

U osnovi kontrolna-suma predstavlja sumu originalnih podataka. Ono što stvara razliku između različitih oblika kontrolnih-suma je način na koji se sumiranje izvodi. **Kontrolna-suma jednostruke tačnosti** je najprostiji oblik kontrolne-sume i formira se izvođenjem binarnog sabiranja nad podatkom koga treba zaštiti, pri čemu se ignoriše eventualno prekoračenje opsega rezultata. Na primer, ako su podaci 8-bitni, sumiranje se obavlja po modulu 256. Ili, u opštem slučaju, za n bitne podatke, sumiranje se obavlja po modulu 2^n, a rezltujuća kontrolna suma je n-bitna. Upravo zanemarivanje prenosa pri sabiranju predstavlja glavni nedostatak ove tehnike jer smanjuje sposobnost detektovanja greške.

Tehnika koja se često koristi za prevazilaženje ograničenja koja ima jednostruka tačnost je **kontrolna-suma dvostruke tačnosti.** Osnovni princip kontrolne-sume dvostruke tačnosti je izračunati $2n$-bitnu kontrolnu-sumu za blok n-bitnih reči koristeći 2^{2n}-modularnu aritmetiku. I u ovom slučaju se javlja prekoračenje opsega, ali to je sada prenos $2n$-bitne sume, za razliku od prekoračenja n-bitne sume u prethodnom slucaju.

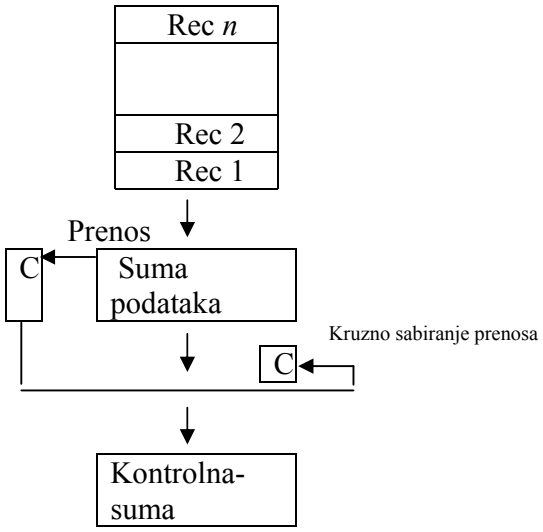
Osnovna ideja **Honeywell kontrolne sume,** prikazane na Sl. 17, je od datih reči formirati kolekciju reči dvostruke dužine. Na primer, ako imamo *k n*–bitnih reči, formiraćemo skup od *k*/2 2*n*-bitnih reči koje se potom sumiraju kako bi se dobila kontrolna suma.



Sl. 17 Honeywell kontrolna-suma.

Osnovna prednost *Honeywell* kontrolne-sume je ta da jedno-bitska greška koja se javlja na istoj bitskoj poziciji u svim rečima utiče na najmanje dve bitske pozicije u kontrolnoj sumi, sto je od posebne važnosti pri detekciji greške nastale usled kvara na magistrali.

Princip **kontrolne-sume ostatka,** prikazane na Sl. 18, je isti kao kod kontrolne-sume jednostruke-tacnosti, s tom razlikom sto se bit prenosa sa bitske pozicije najveće težine ne zanemaruje već se sabira sa kontrolnom-sumom.



Sl. 18 Kontrolna-suma ostatka.

Treba primetiti da kontrolne-sume mogu samo detektovati greške, a ne i locirati ih. Ako se kontrolna-suma generisana u prijemnoj tački razlikuje od kontrolne-sume generisane u tački transmisije, imamo indikaciju greške, ali nemamo dovoljno potrebnih informacija da

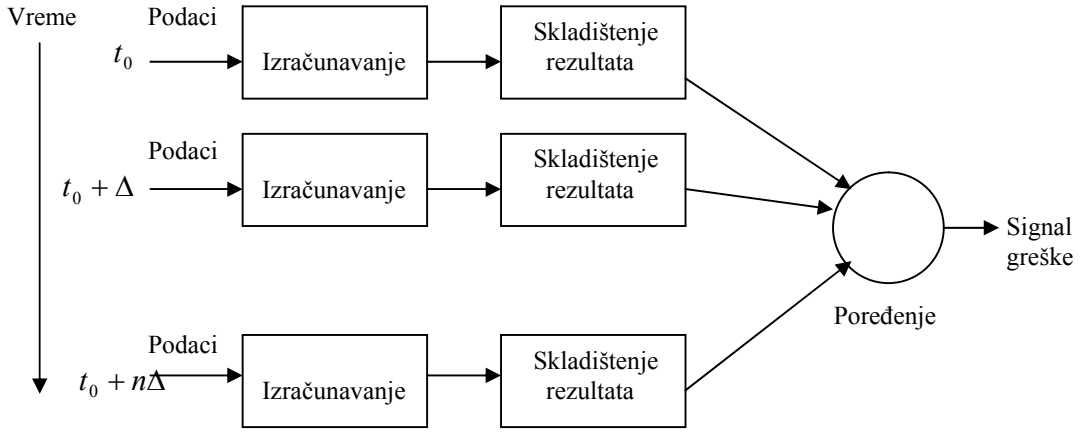
odredimo gde se greška pojavila. U tom slučaju moramo ispraviti celokupni blok podataka (tj. ponoviti prenos bloka podataka) nad kojim je indicirana greška.

3.4 Vremenska redundansa

Metodi vremenske redundanse nastoje da smanje količinu dodatnog hardvera po ceni korišćenja dodatnog vremena. U mnogim aplikacijama vreme je od znatno manje važnosti od hardvera, zato sto je hardver fizički entitet koji značajno utiče na težinu, veličinu, potrošnju snage, i cenu.

3.4.1 Detekcija tranzijentnih kvarova

Osnovni princip vremenske redundanse je u ponavljanju izračunavanja na način koji omogućava detekciju kvarova. Vremenska redundansa u sistemu može funkcionisati na nekoliko načina, ali osnovniji oblik vremenske redundanse prikazan je na Sl. 19.



Sl. 19 Osnovni princip vremenske redundanse.

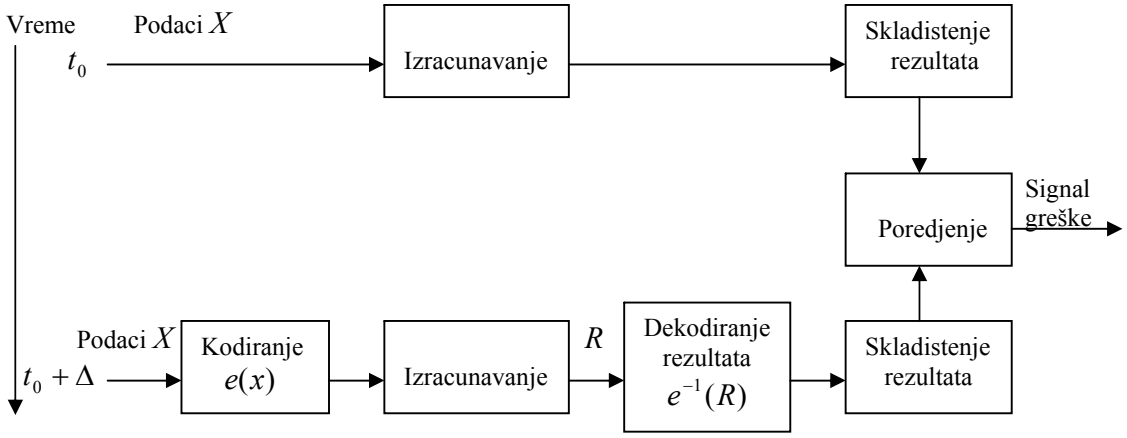
Osnovni princip je izvršavati ista izračunavanja dva ili više puta i upoređivati rezultate u cilju određjivanja eventualnog neslaganja. Ako se detektuje greška, možemo ponoviti izračunavanje da bi videli da li neslaganje ostaje, ili nestaje. Ovakav prilaz je dobar za detekciju grešaka nastalih usled nekog prolaznog, trenutnog (tzv. tranzijentnog) kvara, ali nas on ne može zastititi od grešaka nastalih usled nekog stalnog kvara.

3.4.2 Detekcija permanentnih kvarova

Jedan od najvećih potencijala vremenske redundanse je sposobnost detekcije permanentnih, nepromenljivih kvarova, uz korišćenje minimuma dodatnog hardvera. Značajna su četiri pristupa:

- naizmenična logika,
- ponovno izračunavanje sa pomerenim operandima (*recomputing with shifted operands – RESO*),
- ponovno izračunavanje sa preokrenutim operandima (*recomputing with swapped operands - RESWO*), i
- ponovno izračunavanje koje koristi udvostručavanje sa poređenjem (*recomputing with duplication with comparasion – REDWC*).

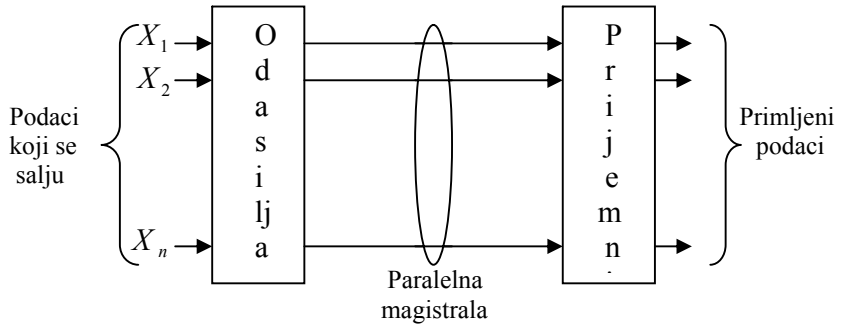
Osnovni princip svih ovih prilaza je isti i prikazan je na Sl. 20, a jedina razlika je u izvođenju drugog izračunavanja. Za vreme prvog izračunavanja ili prenosa podataka, operandi se koriste onakvi kakvi oni stvarno jesu, a rezultati se smeštaju u neki registar. Neposredno pre drugog izračunavanja ili prenosa, operandi se kodiraju koristeći funkciju kodiranja *e*. Nakon sto su izvršene operacije sa kodiranim podacima, rezultati se dekodiraju, i onda porede sa rezultatima dobijenim tokom prvog izračunavanja. Izbor kodirajuće funkcije je urađen tako da omogući detekciju kvarova u hardveru. Metod naizmenične logike kao funkciju kodiranja koristi komplement operatora. RESO kao funkciju kodiranja koristi aritmetičko pomeranje, RESWO koristi funkciju rotiranja za kodiranje operanada, a REDWC je varijacija RESWO-a. U nastavku će biti opisana tehnika naizmenične logike.



Sl. 20 Model vremenske redundanse za detekciju permanentnih kvarova.

Naizmenična logika

Princip naizmenicne logike se primenjuje u prenosu digitalnih podataka preko zičanih medija i u detekciji kvarova u digitalnim kolima. Pretpostavimo da želimo da detektujemo greške u podacima koji se prenose putem paralelne magistrale, kao što je pokazano na Sl. 21, pomoću vremenske redundanse. U trenutku t_0 poslali smo originalni podatak, a u trenutku $t_0 + \delta$ šaljemo komplement podatka. Ako je, recimo, neka linija magistrale permanentno postavljena bilo na 1 ili na 0, dve verzije podataka koje smo primili neće biti komplementi jedna druge, tako da se detektuje kvar. U suštini, ako korišćenjem ovog prilaza šaljemo neku sekvencu informacija, svaka bitska linija, u slučaju da je prenos bez grešaka, će naizmenično menjati svoju vrednost izmedju logicke 1 i logicke 0. Otuda i potiče naziv *naizmenična logika.*



Sl. 21 Prikaz prenosa podataka na daljinu koriscenjem paralelne magistrale.

3.5 Softverska redundansa

U aplikacijama koje koriste računare, mnoge tehnike za detekciju i toleranciju kvarova mogu biti implementirane u softveru. Redudantni softver se može realizovati u više oblika, i nije potrebno replicirati celokupne programe da bi dobili redudantni softver. Softverska redundansa se može realizovati kao nekoliko dodatnih linija naredbi koje se koriste za proveru amplitude signala, ili kao mala programska procedura koja se koristi da periodično testira memoriju učitavajuci i čitajuci sa odredjenih lokacija.

Postoje tri glavne tehnike softverske redundanse: kontrola doslednosti, kontrola sposobnosti, i metoda replikacije softvera.

3.5.1 Kontrola doslednosti

Kontrola doslednosti koristi *a priori* znanje o karakteristikama neke informacije da bi verifikovala ispravnost date informacije. Na primer, u nekim aplikacijama unapred je poznato da digitalna veličina nikad ne bi smela dostići određenu vrednost. Ako signal ipak dostigne tu vrednost, prisutna je neka vrsta greške. Kontrola doslednosti se može realizovati i hardverski, ali se ona uglavnom pojavljuje u okviru softvera. Na primer, procesni sistem može obrađivati i čuvati mnoga senzorska ocitavanja, i svako od senzorskih očitavanja moze biti proveravano da bi se utvrdilo da li leži unutar nekog prihvatljivog opsega vrednosti.

3.5.2 Kontrola sposobnosti

Kontrola sposobnosti se primenjuje da bi potvrdili da sistem poseduje očekivane performanse. Postoji nekoliko formi provere sposobnosti. Prva je prosto testiranje memorije. Procesor jednostavno upisuje odredjene informacije u određene memorijske lokacije, a onda isčitava te memorijske lokacije da bi potvrdio da su podaci uskladišteni i povraćeni ispravno.

Sledeći oblik provere sposobnosti se odnosi na testiranje ALU jedinice. Periodično, procesor izvršava određene instrukcije nad određenim podacima i upoređuje rezultate sa poznatim rezultatima sačuvanim u ROM memoriji. Ova metoda istovremeno kontroliše i ALU i memoriju u kojoj se čuvaju poznati rezultati.

Još jedan oblik kontrole sposobnosti se sastoji od verifikacije da su svi procesori u više-procesorskom sistemu sposobni da međusobno komuniciraju. Test se sastoji u periodičnom prenosu specificnih informacija između procesora.

3.5.3 *N*-verziono programiranje

N-verziono programiranje je razvijeno u cilju omogućavanja detekcije odredjenih projektantskih propusta u softveru. Osnovni princip *N*-verzionog programiranja je projektovati i programirati softverski modul *N* puta i onda uporediti *N* rezultata dobijenih sa ovih softverskih modula. Svaki od *N* modula projektuje i programira posebna grupa programera. Svaka grupa projektuje softver na osnovu istih specifikacija tako da svaki od *N* modula izvršava iste funkcije. Ovo je urađeno sa nadom da *N* nezavisnih projektantskih grupa neće načiniti iste greške.

Postoje dve osnovne teskoće sa ovim prilazom. Prva je da softverski projektanti uglavnom prave slične greske, tako da nije isključeno da dve potpuno nezavisne verzije programa imaju iste greške. Druga je ta da je svih *N* verzija programa razvijeno na osnovu identične specifikacije, tako da je nemoguće detektovati greške u specifikaciji.