ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA HDB3 KODERA/DEKODERA

–Diplomski rad–

Kandidat:

Katarina Lazović 290/2009

Mentor:

doc. dr Zoran Čiča

Beograd, Oktobar 2014.

SADRŽAJ

1. UVOD

U osnovi prenosa digitalnih signala u osnovnom opsegu učestanosti je tehnika koja podrazumeva da se signal oblikuje tako da bude što pogodniji za prenos u skladu sa karakteristikama medijuma kojim se prenosi, kao i sa uređajima koji u tom prenosu učestvuju. Pod linijskim kodovima podrazumevaju se signali koji poseduju osobine koje mogu poboljšati celokupne performanse sistema. Šezdesetih godina prošlog veka razvijani su za potrebe digitalnog prenosa kroz telefonske kanale ili za snimanje digitalnih signala na magnetne medijume. U današnje vreme, istraživanja su koncentrisana na razvoj linijskih kodova za prenos signala optičkim vlaknima.

HDB3 (*High Density Bipolar Of Density 3*) kod namenjen je upotrebi u evropskim E1 i E3 digitalnim hijerarhijskim nivoima. HDB3 je bipolaran linijski kod, što znači da se za prenos simbola 1 koriste i pozitivan i negativan naponski nivo. Pri pojavi dve uzastopne jedinice njihovi naponski nivoi alterniraju, čime se potiskuje prenos jednosmerne komponente. Pri prenosu više uzastopnih nula, primenom ovog načina kodiranja, broj nula koje mogu uslediti jedna za drugom se ograničava na tri. Tranzicija naponskog nivoa signala je od ključnog značaja za regeneraciju takta. Potiskivanje DC komponente i mogućnost sinhronizacije takta ujedno čine preduslov koji linijski kodovi koji se koriste za povezivanje centrala moraju ispuniti. U ovom radu su prikazane implementacija i simulacija HDB3 kodera/dekodera, analiza rezultata dobijenih pokretanjem simulacije, sa posebnim osvrtom na ponašanje dekodera u slučaju kada je pri prenosu prisutna greška.

Ostatak rada organizovan je na sledeći način. U drugom poglavlju biće više reči o najkorišćenijim linijskim kodovima i njihovim karakteristikama, dok se treće poglavlje koncentriše na sam HDB3 kod. U trećem poglavlju biće izložen princip rada ovakvog načina kodovanja, kao i detaljno objašnjenje implementacije HDB3 kodera, dekodera bez greške i dekodera sa greškom. Četvrto poglavlje se detaljno koncentriše na samu simulaciju HDB3 kodera/dekodera, proces unosa greške i ponašanje dekodera u slučaju kada postoji greška, kao i na analizu ponašanja dekodera u ovom slučaju. Na kraju je dato peto poglavlje koje se sastoji od zaključka i zapažanja vezanih za obrađenu tematiku u okviru ove teze.

2. LINIJSKI KODOVI

U ovom poglavlju biće više o ulozi linijskih kodova, njihovoj podeli, kao i o tipičnim talasnim oblicima pojedinih vrsta digitalnih signala.

Liniijski kodovi pomažu pri otklanjanju nedostataka ulaznog niza. Pomoću njih je moguće:

- Zadržati binarnu prirodu originalnog signala i pri tome povećati brzinu prenosa
- Za istu bitsku brzinu dobiti digitalni signal sa većim brojem nivoa
- Povećati broj nivoa i sniziti brzinu prenosa

Potrebno je napomenuti i da se pri svim ovim postupcima unosi neka redundantnost.

2.1. TALASNI OBLICI SIGNALA

Elementarni digitalni signal je upravo binarni i on predstavlja najjednostavniji oblik digitalnog signala. Digitalni signali se u opštem slučaju mogu podeliti na sledeće grupe:

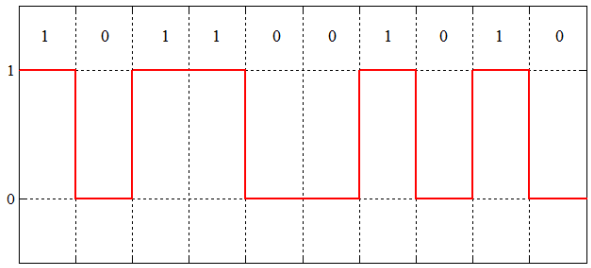
- Binarni signali bez povratka na nulu, NRZ (*Non Return to Zero*)
- Binarni signali sa povratkom na nulu, RZ (*Return to Zero*)
- Fazno kodirani binarni signali, PE (*Phase Coded*)
- Višenivovski binarni signali, ML (*Multilevel*)
- M-arni signali

Pri izboru odgovarajućeg digitalnog signala, osnovni parametri koji se uzimaju u obzir su:

- Jednosmerna komponenta – digitalni signali bez jednosmerne komponente omogućavaju kondenzatorsku, odnosno transformatorsku spregu u sistemu i štite sistem od izobličenja signala u niskofrekvencijskom domenu;
- Sposobnost samosinhronizacije – digitalni signali koji imaju ovu osobinu znatno olakšavaju proces akvizicije i održavanja sinhronizacije u sistemu (tipičan primen Manchester kod);
- Korekcija greške – pojedini postupci formatiranja, poput duobinarnog, omogućavaju detekciju greške u toku prenosa bez dodavanja posebnog bita za detekciju greške;
- Spektralna efikasnost – definiše se kao količnik ekvivalentnog binarnog protoka i minimalnog potrebnog opsega učestanosti za prenos signala. Korišćenjem višenivovskih signala povećava se spektralna efikasnost sistema;
- Otpornost na šum – na ovaj parametar je moguće značajno uticati odabirom pogodnog oblika digitalnog signala;

2.1.1. Unipolarni binarni signal bez povratka na nulu – NRZ

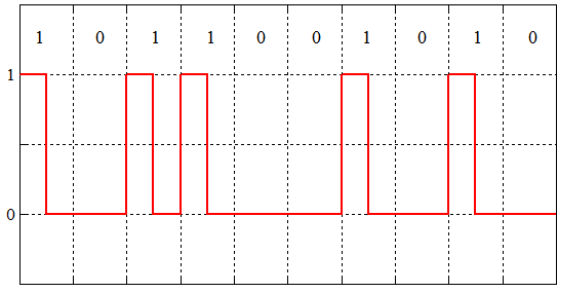
Unipolarni binarni signal bez povratka na nulu – NRZ (*Non Return To Zero*) naziva se još i jednosmerni signal. Odgovaraju mu dva naponska nivoa, 1 i 0. Kada postoji signal na ulazu ima vrednost 1, dok 0 označava da na ulazu nije prisutan signal. Nedostatak predstavlja činjenica da je nemoguće utvrditi granice između bitova ako se u kraćem vremenskom periodu javi niz impulsa istog naponskog nivoa. Ovaj signal sadrži DC komponentu, što predstavlja veliki nedostatak. Na slici 2.1.1. prikazan je oblik proizvoljno izabranog NRZ signala.



Slika 2.1.1.1. NRZ signal

2.1.2. Unipolarni binarni signal sa povratkom na nulu - RZ

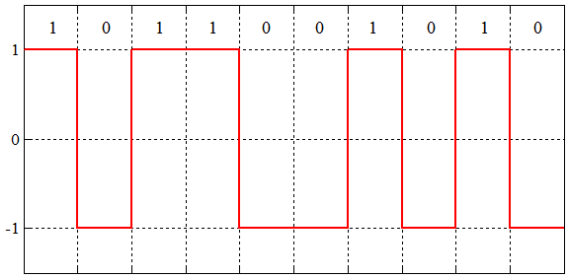
Kod RZ (*Retutn To Zero*) signala nivo između uzastopnih bita se uvek vraća na nulu i signal postoji samo jednu polovinu intervala. Kada se signal ne bi vraćao na nulu, moglo bi doći do gubitka sinhronizacije u prijemniku. Na slici 2.1.2.1. prikazan je talasni oblik RZ signala iz prethodnog odeljka.



Slika 2.1.2.1. RZ signal

2.1.3. Polarni binarni signal – NRZ

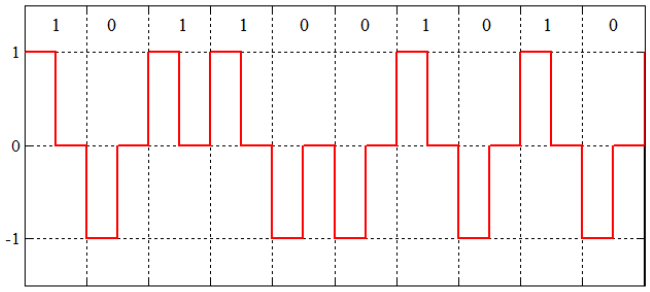
Može imati vrednosti +1 ili -1, gde +1 odgovara logičkoj jedinici a -1 logičkoj nuli. Znatno je otporniji na uticaj šuma u odnosu na unipolarni RZ binarni signal. Na slici 2.1.3.1. je prikazan talasni oblik proizvoljno odabranog polarnog binarnog NRZ signala.



Slika 2.1.3.1. Polarni binarni NRZ signal

2.1.4. Polarni binarni signal – RZ

Za vrednosti logičke nule uzima vrednost -1 a za vrednost logičke jedinice vrednost +1, pri čemu trajanje jednog impulsa iznosi pola takta. Talasni oblik signala prikazan je na slici 2.1.4.1.



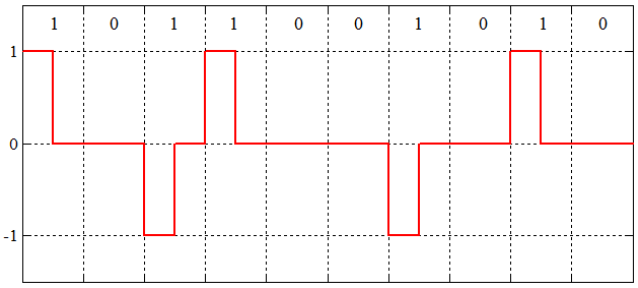
Slika 2.1.4.1. Polarni binarni RZ signal

2.1.5. AMI (Alternating Mark Inversion) kod

Kod ovog koda se binarna poruka predstavlja pomoću tri naponska nivoa, pri čemu impulsi binarne jedinice moraju naizmenično menjati polaritet. U odnosu na pomenute kodove, AMI ima višestruke prednosti:

- Spektar signala je znatno uži nego kod NRZ signala;
- Alterniranjem naponskih nivoa uzastopnih jedinica postiže se da srednja vrednost jednosmerne komponente teži nuli.

Talasni oblik AMI signala iz prethodnih primera prikazan je na slici 2.1.5.1.



Slika 2.1.5.1. Talasni oblik AMI signala

3. HDB3 KOD – IMPLEMENTACIJA

U ovom poglavlju biće više reči o HDB3 kodu, samoj implementaciji HDB3 kodera/dekodera, kao i dekodera sa greškom. Kod je pisan u Matlab R2013a okruženju, koje je pogodno za realizaciju simulatora zbog svoje jednostavnosti i razumljive sintakse. Celokupni program je podeljen na 3 celine – koder, dekoder i funkciju unosa greške. Funkcija za unos greške može biti realizovana za više verovatnoća pojavljivanja grešaka i u zavisnosti od toga moguće je analizirati efikasnost dekodera za slučajeve koji sadrže grešku u prenosu.

3.1. HDB3 KOD

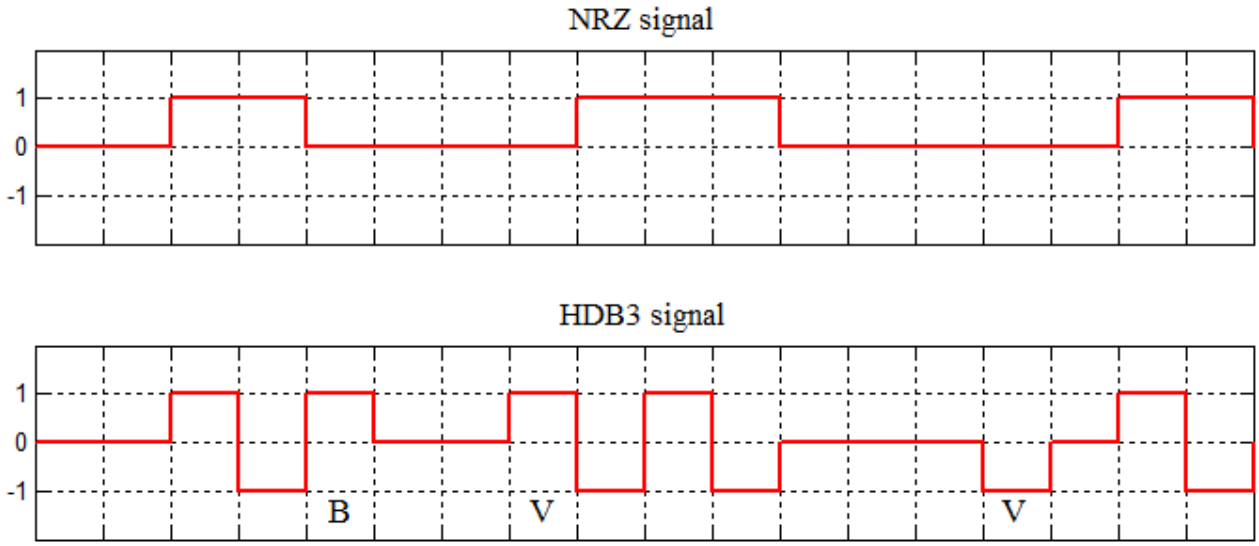
HDB3 kod je unapređena verzija AMI koda. Poseduje sve prednosti AMI koda, kao što su na primer funkcija detekcije grešaka, uzak spektar i odsustvo jednosmerne komponente. Kao što je već navedeno na početku rada, HDB3 predstavlja binarni linijski kod koji suzbijanjem pojave više uzastopnih nula i alterniranjem naponskih nivoa susednih jedinica omogućava prenos bez jednosmerne komponente, kao i regenerisanje takta. Ovakav prenos je moguć uz primenu sledećih pravila kodiranja:

- Kada se u nizu pojave četiri uzastopne nule, poslednja nula u tom nizu se menja sa takozvanim V (*Violation*) bitom koji ima isti nivo kao i poslednja poslata jedinica. Pri tome se ne uzima u obzir znak poslednjeg V bita, već samo znak regularne jedinice
- U slučaju kada je broj regularnih jedinica između V bita koji se utiskuje i prethodno utisnutog V bita paran, prva nula u nizu od četiri nule se menja takozvanim B (*Balancing*) bitom koji ima suprotnu vrednost od poslednje poslate jedinice, tj. računa se kao da je u pitanju regularna jedinica. V bit koji se utiskuje posle B bita ima isti znak kao i B bit, iz razloga što se B bit računa kao regularna jedinica.

Princip HDB3 kodiranja pojasnićemo na primeru kada je data ulazna binarna sekvenca 001100001110000011.

Kodovanje se vrši pod pretpostavkom da je broj jedinica paran, kao i da je sekvenca prikazana od početka (tj. smatramo da ne postoji mogućnost da je neka od nula sa početka niza ustvari četvrta nula u nizu). Kako je broj uzastopnih nula na početku manji od četiri, vrednost prva dva bita ostaje nepromenjena. Slede dve jedinice – za prvu se usvaja pozitivan naponski nivo dok sledeću alterniramo i ona uzima obrnuti znak. Sada se pojavljuju četiri uzastopne nule pa se mora uzeti u obzir broj jedinica koji prethodi. Vraćajući se na pravila kodiranja, može se uvideti da je potrebno umetnuti i B i V bit, na mestu krajnjih nula u nizu od četiri nule, respektivno. Slede tri uzastopne jedinice koje će alternirati znake u odnosu na prethodnu jedinicu (u suštini u odnosu na B bit). Zatim nailazimo na pet uzastopnih nula, ali ovaj put nema potrebe za utiskivanjem B bita jer je broj regularnih jedinica između ove povorke nula i poslednjeg umetnutog V bita neparan. Ipak, neophodno je utisnuti V bit jer je broj nula veći od 3. U ovom slučaju V bit uzima vrednost poslednje poslate jedinice, odnosno -1. Preostala tri bita se koduju u skladu sa prethodno opisanim

postupkom pa se tako nula preslikava u nula, a dve jedinice koje slede alterniraju znak jedna u odnosu na drugu. Opisana dešavanja su prikazana grafički na slici 3.1.1.



Slika 3.1.1. Konverzija u HDB3 kod - primer

3.2. IMPLEMENTACIJA HDB3 KODERA

Cilj kodiranja jeste da signal u NRZ obliku prevedemo u HDB3 predstavu. Zamisao je da se prolazi kroz ulazni NRZ niz bit po bit uz odgovarajuće petlje i u skladu sa uslovima koje kod mora zadovoljavati. Realizacija ovog postupka je izvršena uz pomoć funkcije koja za ulazni NRZ niz kao izlazni parametar vraća ovaj niz prikazan u HDB3 kodu.

Prvi deo funkcije čini definisanje promenljivih:

```
function [OUT_HDB3] = kod(NRZ)
                duzina = length(NRZ);     %duzina ulaznog niza
                HDB3 = zeros(1,duzina);  %izlazni niz se inicijalizuje nulama
                zero_cnt = 0;            %brojac uzastopnih nula
                one_cnt = 0;             %brojac regularnih jedinica izmedju
                                         %dva V bita
                B = 1; V = 1;            %pocetne vrednosti B i V bita
                HDB3_br = 1;             %iterator kroz izlazni niz
```

Inicijalizacija izlaznog niza nulama je neophodna kako bi se u isti taj niz kasnije upisale vrednosti koje se dobijaju nakon primene pravila kodiranja.

Sledi *for* petlja koja prolazi kroz celu sekvencu. Prva stvar koja nas zanima za dalju analizu jeste da li je cifra koja se trenutno obrađuje nula ili jedinica. U slučaju da je reč o jedinici, iterator kroz niz se uvećava za onu vrednost koliko se nula nalazi pre ove jedinice (pri čemu taj broj mora biti manji od 4), tako da nule ostanu u izlaznom nizu. Posle ovog koraka je neophodno resetovati brojač uzastopnih nula. Jedinici se dodeljuje odgovarajuća vrednost B, koja nosi unapred invertovanu vrednost prošle jedinice a brojač jedinica između dva V bita se uvećava za jedan. Preostalo je još postaviti V i B na odgovarajuće vrednosti – V bit dobija znak B bita, dok B dobija suprotan znak. Kod opisanog postupka dat je u nastavku teksta.

```
for NRZ_br = 1 : length(NRZ)
if(NRZ(NRZ_br) == 1)           %provera da li se obradjuje 1 ili 0
 HDB3_br = HDB3_br + zero_cnt;%preskace se <4 nule koje se nalaze
pre 1, tako da u izl nizu ostaju 0
            zero_cnt = 0;           %resetuje se brojac uzastopnih nula
HDB3(HDB3_br) = B; %regularna 1 dobija odg vrednost
            HDB3_br = HDB3_br + 1;
            one_cnt = one_cnt + 1;     %brojac jedinica izmedju 2 V bita ++
            V = B;
            B = -1 * B;            %postavljanje vrednosti V i B bita
```

Slučaj u kome se obrađuje nula ima više mogućnosti jer sa pojavom četiri ili više uzastopnih nula dolazi do utiskivanja V i B bita, zato je neophodno da se ispita da li je broj jedinica koje se pojavljuju između dva V bita paran ili neparan. U slučaju da je paran, utisnuće se V i B bit pa će se u izlazni niz ispisati vrednost B00V, dok se u suprotnom ispisuje 000V. Kod koji je priložen je nastavak na kod sa prethodno naveden kod.

```
else%jeste nula
            zero_cnt = zero_cnt + 1;
if(zero_cnt == 4)      %ako je broj uzastopnih nula jednak 4,
%treba utisnuti V i B bite po potrebi
        zero_cnt = 0;       %resetuje se broj uzastopnih nula, nikad
if mod(one_cnt,2) == 0   %provera da li je broj jedinica izmedju
%izmedju zadnja 2 V bita paran
                    V = B;%jeste paran
HDB3(HDB3_br + 0) = B;
HDB3(HDB3_br + 1) = 0;
HDB3(HDB3_br + 2) = 0;
HDB3(HDB3_br + 3) = V;
                    HDB3_br = HDB3_br + 4;
                    B = -1 * B;
else%neparan broj jedinica, utiskuje se V bit
HDB3(HDB3_br + 0) = 0;
HDB3(HDB3_br + 1) = 0;
HDB3(HDB3_br + 2) = 0;
HDB3(HDB3_br + 3) = V;
                    HDB3_br = HDB3_br + 4;
end
                one_cnt = 0;            %kako se pojavio V bit, resetuje se
                                        %brojac jedinica izmedju 2 V bita
end
end
end
    OUT_HDB3 = HDB3;
```

Funkcija kao izlazni rezultat vraća vrednost signala u HDB3 kodu.

3.3. IMPLEMENTACIJA HDB3 DEKODERA

Pre same detaljne analize dekodovanja sa i bez greške, objasnićemo pomoćnu funkciju koja se koristi u oba dekodera (sa i bez greške) i koja simulira bafer u koji se skladište biti koji će kasnije biti upisani u izlazni niz. Bafer je neophodan jer u najvećem broju sitacija četvorke bita u sebi nose informaciju o V i B bitu.

Funkcija ima jedan ulazni i jedan izlazni parametar. Ulazni parametar *ulaz* služi kao pomoćna promenljiva koja u sebi nosi informaciju o tome da li se upisuje 1 ili 0, dok vrednost 2 ove

promenljive predstavlja specijalni slučaj. Za vrednosti promenljive *ulaz* 0 ili 1, vrednosti bita u baferu se pomeraju za jedno mesto udesno, a vrednost promenljive *ulaz* se preslikava u prvi bit. Vrednost 2 predstavlja specijalni slučaj i označava da sva četiri bita koja se nalaze u baferu treba pretvoriti u nulu. Pojasnimo na primeru -11001. Logično bi bilo da koder počne prevođenje u 11001. Međutim, ovu sekvencu bi bilo ispravno posmatrati kao karakteristični slučaj 1001 (koji se preslikava u 0000) a ispred koga stoji -1. Kako bi se izbeglo pogrešno tumačenje ovakve sekvence bita, neophodno je implementirati ovaj scenario u pomoćnu funkciju.

Promenljiva *pomocna* koristi se za čuvanje informacije o vrednosti onog bita koji je najduže bio u baferu, a ujedno predstavlja i izlazni parametar ove funkcije.

```
function [izl] = insert(ulaz) %pomocna funkcija koja vraca onu vrednost koja se
%najduze nalazila u baferu
pomocna = buffer(4);

if (ulaz ==  2) %specijalan slucaj kada je potrebno upisati 0000
buffer = [0,0,0,0];

else%sve u baferu se pomera za jedno mesto, ono sa poslednjeg
%izbacujemo napolje a na prvo mesto
buffer(4) = buffer(3);
buffer(3) = buffer(2);
buffer(2) = buffer(1);
buffer(1) = ulaz;
end

 [izl] = pomocna;
end
```

Ovu funkciju je potrebno umetnuti u kod nakon inicijalizacije ulaznih podataka, a pre ulaska u *for* petlju.

3.3.1. Dekoder bez greške

Dekoder je realizovan kao funkcija koja ima jedan ulazni i dva izlazna parametra, pod pretpostavkom da na ulazu nema greške, tj. da nije moguće da se na ulazu u dekoder nađe sekvenca koja se kosi sa osnovnim načelima HDB3 kodiranja. Ulazni parametar je HDB3 niz koji je dobijen kao rezultat funkcije kodiranja NRZ sekvence postupkom opisanim u potpoglavlju 3.2.

Prvi korak je definisanje promenljivih koje se koriste unutar funkcije; sledi kod kojim se ovo postiže:

```
function [nrz_novi] = dekoder_bez_greske(HDB3)
duzina = length(HDB3);
         NRZ = zeros(1,duzina);
znak = 0;
         zero_cnt = 0;%trenutni brojac uzastopnih 0
         one_cnt = 0; %brojac uzastopnih jedinica

buffer = zeros(1,4);%sluzi za 'skladistenje' vrednosti 0 ili 1
izlaz = zeros(1,duzina+4); %niz koji ce sadrzati NRZ output
```

Promenljiva *zero_cnt* je trenutni brojač uzastopnih nula; kada se na ulazu pojavi nula uvećava se za jedan, a kada dođe nešto što nije nula (+1 ili -1) resetuje se. Brojač uzastopnih jedinica, *one_cnt* uvećava se za jedan svaki put kada dođe jedinica istog polariteta. Kada je na ulazu

jedinica suprotnog polariteta postavlja se na vrednost 1 jer se sada računa broj jedinica tog polariteta, dok se za pojavu nule ne menja. *Buffer* se koristi za pomoćnu funkciju *insert* i iz njega se šalju vrednosti u promenljivu *izlaz*. Ova promenljiva definiše izlazni NRZ niz, proširen za četiri mesta kako se ne bi uračunale četiri nule koje se koriste za inicijalizaciju promenljive *buffer*.

Sledeći deo koda čini *for* petlja koja prolazi kroz niz i za različite vrednosti ulaznih parametara primenjuje pravila kodiranja i kao rezultat daje izlazne članove NRZ niza. Prva stvar koju ispitujemo jeste vrednost trenutnog bita. Ako je vrednost trenutnog bita +1 ili -1, postoje dva moguća scenarija.

1) Vrednost promenljive *znak* je različita od vrednosti trenutnog bita – jednaka je nuli (predstavlja indikator da je u pitanju prva pročitana jedinica) ili je suprotnog polariteta od prethodne jedinice. Stoga je neophodno dodeliti ovoj promenljivoj vrednost bita koji se trenutno obrađuje, kao i podesiti parametre *one_cnt* i *zero_cnt* na vrednosti 1 i 0, respektivno. Sada je na redu upisivanje u izlazni niz. Funkcijom *insert(1)* postižemo da se na prvo mesto u baferu upiše vrednost 1.
2) Polaritet ove jedinice je isti kao i polaritet prošle – *one_cnt* se uvećava za jedan, dok se *zero_cnt* vraća na nulu. U ovoj situaciji je potrebno primeniti pomoćnu funkciju *insert(2)* koja obuhvata specijalan slučaj kada je potrebno sve u baferu zameniti sa četiri nule kako bi se sekvenca ispravno dekodovala.

Sa druge strane, ako je bit koji se trenutno obrađuje nula, potrebno je uvećati *zero_cnt* za jedan i primeniti funkciju *insert(0).* Time se postiže da se u izlazni niz upisuje onaj bit koji je najduže bio u baferu, ostali se pomeraju za jedno mesto udesno, a na prvo mesto u baferu upisuje se nula.

Za *i >= HDB3_br(i),* „insertujemo“ vrednost 0 četiri puta za redom kako bi se vrednosti koje se nalaze u baferu ispisale u izlazni niz, a bafer se vraća u početno stanje.

Konačno, izlazni NRZ niz se dobija izbacivanjem prva četiri bita iz dobijenog niza *izlaz*. Ove bite je neophodno izbaciti jer su oni „pomoćni“ biti koji su služili za inicijalizaciju bafera. Kod kojim je ovo postignuto nalazi se u nastavku teksta.

```
function [izl] = insert(ulaz) %pomocna funkcija
pomocna = buffer(4);
if (ulaz ==  2)%specijalan slucaj kada je potrebno %%
                                %‘pregaziti’ jedinicu koja se nalazi u baferu;
                              %npr. kada dodje sekvenca -11001 deo posle -1
                               %treba tumaciti kao 0000

buffer = [0,0,0,0]
else%sve u baferu se pomera za jedno mesto, ono
                                %sto je bilo poslednje se upisuje u izlaz a na
                                %prvo mesto upisujemo ulaz

buffer(4) = buffer(3);
buffer(3) = buffer(2);
buffer(2) = buffer(1);
buffer(1) = ulaz;
end
                  [izl] = pomocna;
end

for i = 1 : duzina
if (HDB3(i) ~= 0) %trenutni bit je ne-nula
```

```
if ((znak == 0)) | ((znak  ~=   HDB3(i)))
znak = HDB3(i);
                      zero_cnt = 0;
                      one_cnt = 1;
izlaz(i) = insert(1);

else%polaritet ove jedinice je isti kao prosle
                                  one_cnt = one_cnt+1;
                                  zero_cnt = 0;
izlaz(i) = insert(2); %specijalan slucaj
end
else%trenutno je nula
                      zero_cnt = zero_cnt+1;
izlaz(i) = insert(0);
end
end

         i = length(HDB3)+1;%mora izvan for petlje zbog indeksiranja
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;

         [nrz_novi] = izlaz(5:(length(izlaz)));
end
```

3.3.2. Dekoder sa greškom

Ideja je da se u *HDB3* niz dobijen primenom funkcije *kod(NRZ)* unese greška i da se zatim tako dobijen niz rekonstruiše kako bi se ispitala efikasnost rada dekodera u slučajevima kada prenos nije idealan, odnosno kada je u pitanju realan prenos. Dekoder sa greškom realizovan je na isti način kao i dekoder bez greške uz nekoliko izmena kako bi se pokrili nedozvoljeni slučajevi koji se mogu pojaviti na ulazu. Pre opisa rada samog dekodera, pojasnićemo postupak ubacivanja greške.

Funkcija *unos_greske* pretpostavlja sledeće parametre:

- Ulazne parametre *HDB3* (niz dobijen kao rezultat funkcije *kod(NRZ)* iz poglavlja 3.2.) i *P* (verovatnoću greške)
- Izlazne parametre *OUT_GR* i *p_sent*, gde je prva promenljiva izlazni niz sa ubačenom greškom a druga nosi informaciju o broju različitih bita u izlaznom u odnosu na ulazni HDB3 niz
- Za slučaj greške na 0, verovatnoću 50-50 da je ispravan bit +1, odnosno -1
- Za slučaj greške na +1, verovatnoću greške od 90% da je trebalo da bude prenesena nula, odnosno 10% da je trebalo da bude prenesena -1
- Za slučaj da se greška desila pri prenosu -1, verovatnoću 90% da je trebalo da bude prenesena nula i verovatnoću 10% da je ispravno preneti +1
- Greška se generiše kao slučajna promenljiva sa uniformnom raspodelom i uzima vrednost između 0 i 1

Uzimajući u obzir navedene pretpostavke, kod ima sledeću strukturu:

```
function [OUT_GR,p_sent]=unos_greske(HDB3,P)
duzina = length(HDB3);
         HDB3_gr = zeros(1,duzina);
for i = 1 : length(HDB3)
greska = rand;    %greska ima slucajnu vrednost izmedju 0 i 1

if (greska < P)

if HDB3(i) == 0
                       greska0 = rand;
if greska0 < 0.5
                           HDB3_gr(i) = +1;
else
                           HDB3_gr(i) = -1;

if (HDB3(i) == 1)
                       greska1 = rand;

if greska1 < 0.9
                           HDB3_gr(i) = 0;
else
                           HDB3_gr(i) = -1;
end

if (HDB3(i) == 1)
                       greska11 = rand;
if (greska11 < 0.9)
                          HDB3_gr(i) = 0;
else
                          HDB3_gr(i) = +1;
end
end
end
end
end

else
                  HDB3_gr(i) = HDB3(i);
end
end
         sent_cnt = 0;
for i = 1:duzina
if (HDB3(i) ~= HDB3_gr(i))
                 sent_cnt = sent_cnt + 1;
end
end

         p_sent=sent_cnt;
         OUT_GR=HDB3_gr;
```

Na ovaj način dobijeni niz *HDB3_gr* prosleđuje se kao ulazni parametar funkciji dekodera sa greškom. Za razliku od običnog dekodera, u ovom slučaju postoje tri izlazna parametra, a to su *NRZ_novi*, *p_det*i *greska* pri čemu *NRZ_novi* predstavlja rekonstruisani *HDB3_gr* niz, *p_det* broji

koliko je grešaka detektovano, a *greska* je promenljiva koja ima pozitivan naponski nivo na mestima gde je došlo do greške. Nedozvoljeni slučajevi su:

1) Više od tri uzastopne nule
2) Dve jedinice istog polariteta između kojih nema nula ili postoji samo jedna nula
3) Tri ili više uzastopne jedinice istog naponskog nivoa

Projektovanje dekodera sa greškom započinje se na sličan način kao i u prošlom odeljku:

```
function [p_det, nrz_novi,greska] = dekoder_greska(HDB3_gr)
duzina = length(HDB3_gr);
         ERROR = zeros(1,duzina);
         NRZ = zeros(1,duzina);
znak = 0;
         zero_cnt = 0; %trenutni brojac uzastopnih 0
         one_cnt = 0; %brojac koliko je jedinica doslo uzastopno
buffer = zeros(1,4); %sluzi za 'skladistenje' vrednosti koje se
 %kasnije salju u izlazni niz
izlaz = zeros(1,duzina+4); %niz koji ce sadrzati NRZ output
```

Promenljiva *ERROR* u sebi skladišti informaciju o tome na kom je bitu došlo do nedozvoljenog stanja. Na primer, ako je na ulaz dekodera došla sekvenca 1101, on će pretpostaviti da je „uljez“ druga po redu jedinica i da je tu trebalo da bude nula pa će ovu sekvencu dekodovati kao 0000 a vrednost same promenljive biće 0100. *Zero_cnt* je brojač uzastopnih nula i uvećava se za jedan svaki put kada se detektuje nula, a u suprotnom se resetuje. Može imati vrednost 0, 1, 2 ili 3, dok se za slučaj četiri ili više nula prijavljuje greška. Brojač uzastopnih jedinica, *one_cnt*, se uvećava za jedan svaki put kada se šalje jedinica istog polariteta, postavlja na 1 ako dođe jedinica suprotnog polariteta, a u slučaju pojave nule se ne menja. Funkcije promenljivih *buffer* i *izlaz* su iste kao i u prethodnom odeljku.

Nakon ovog koraka potrebno je da umetnemo pomoćnu funkciju *insert* a zatim sledi obrada ulaznog niza. Deo programskog koda koji se odnosi na dekodovanje bez greške je isti kao i u odeljku 3.3.2. te će u ovom delu teksta biti objašnjeni samo delovi koda koji se odnose na slučajeve u kojima se javlja greška.

U svim situacijama koje obuhvata ovaj deo teksta promenljiva *ERROR* dobija vrednost 1 na trenutnom bitu. Ako to nije slučaj, biće posebno naglašeno.

Krenimo od situacije kada se na ulazu nalazi +1 ili -1. Sledeća stvar koja se ispituje je broj uzastopnih nula – za manje od tri uzastopne nule i pod pretpostavkom da je jedinica koja se trenutno obrađuje pogrešna, menja se nulom, uvećava se brojač nula za jedan i u promenljivu *izlaz* se upisuje ona cifra koja je najduže bila u baferu, ostale cifre se pomeraju za jedno mesto udesno i na prvo mesto u baferu se upisuje nula. U suprotnom, tj. ako već postoje tri nule za redom, potrebno je promeniti polaritet trenutnoj jedinici i tu vrednost upisati kao ispravljenu u *HDB3_gr(i)*. Promenljiva *one_cnt* sada dobija vrednost 1 jer je ovo ujedno i prva jedinica tog polariteta na koju smo naišli, a *zero_cnt* se resetuje. U izlazni niz se upisuje *buffer(4)*, ostale cifre se pomeraju za jedno mesto udesno a na prvo mesto se upisuje 1. Preostalo je obraditi još slučaj kada između dve jedinice istog polariteta postoje manje od dve ili više od tri nule, a zna se da je pre jedinice koja se trenutno obrađuje postojala samo jedna jedinica istog polariteta. Tada je potrebno ispraviti član ulaznog niza u nulu, uvećati brojač nula za jedan i pozvati funkciju *insert(0)*.

Programska realizacija glasi:

```
for i = 1 : duzina
if (HDB3_gr(i) ~= 0) %trenutni znak je ne-nula
if (znak == 0)   %sigurno nije greska jer je ovo prva procitana
                                %jedinica
znak = HDB3_gr(i);
                    zero_cnt = 0;
                    one_cnt = 1;
izlaz(i) = insert(1);
else
if (znak  ~=   HDB3_gr(i)) %polaritet ove jedinice je
                                                %razlicit od prosle, nema greske
znak = HDB3_gr(i);
                         zero_cnt = 0;
                         one_cnt = 1;
izlaz(i) = insert(1);
else%polaritet ove jedinice isti kao prosle

if (one_cnt >=  2)      %vec su se pojavile dve
                                                %jedinice istog znaka
if (zero_cnt<3)
                                HDB3_gr(i) = 0; %pod pretpostavkom da je ova
                                                %jedinica pogresna, menjamo je
                                                %sa nulom
ERROR(i) = 1;
                                zero_cnt = zero_cnt+1;
izlaz(i) = insert(0);

else%vec postoje 3 nule za redom,
                                                %menja se u suprotni polaritet
znak = -1*znak;
                                HDB3_gr(i) = znak;
ERROR(i) = 1;
                                zero_cnt = 0;
                                one_cnt = 1;
izlaz(i) = insert(1);

end

else%bila samo jedna jedinica
%tog polariteta pre ove dve
if (zero_cnt >=  2)   %nema greske, ovo je slucaj
%1001 ili -1000-1

                                one_cnt = one_cnt+1;
                                zero_cnt = 0;
izlaz(i) = insert(2); %za slucaj da je u
                                                %pitanju -100-1 pa sve
                                                %pretvaramo u 0000

else%broj nula izmedju dve iste jedinice
                                                %manji od dva ili veci od 3
ERROR(i) = 1;
                                HDB3_gr(i) = 0;
                                zero_cnt = zero_cnt+1;
izlaz(i) = insert(0);
end
end
end
```

```
end
```

Preostalo je još analizirati slučaj kada se na ulazu nalazi nula. Ako se utvrdi da postoje više od 3 nule potrebno je ispitati vrednost promenljive *znak*. Ako je njena vrednost 0, usvajamo da se pogrešan bit menja sa 1 (iako ne bi bilo pogrešno ni da se menja sa -1 jer dekoder samo utvrđuje gde je greška i nema sposobnost da tačno dekoduje pogrešan bit). Brojač uzastopnih nula se resetuje, a brojač uzastopnih jedinica dobija vrednost 1 i poziva se funkcija *insert(1)*. Za nulu na ulazu i već postojeći niz od dve ili više uzastopnih jedinica, trenutnu nulu konvertujemo u -1, menjamo polaritet promenljivoj *znak* i tu vrednost upisujemo u niz *HDB3_gr(i)* a ostali parametri dobijaju iste vrednosti kao u prethodnom slučaju. Ako se dogodi da je na ulazu nula i postoje dve ili više jedinica, ali one nisu istog znaka, trenutnu nulu konvertujemo u poslednju jedinicu, brojač *one_cnt* uvećavamo za jedan, *zero_cnt* se restuje, a *HDB3_gr(i)* se dodeljuje onu vrednost koju trenutno nosi promenljiva *znak*. Poziva se funkcija *insert(2)* kako bi se sprečilo da niz od četiri nule bude drugačije dekodovan. U nastavku se nalazi programski kod kojim je ovo postignuto (nastavak na funkciju koja se nalazi iznad ovog teksta).

```
else%trenutno je nula

if (zero_cnt < 3)     %broj uzastopnih nula manji od 3, sve ok
                     zero_cnt = zero_cnt+1;
izlaz(i) = insert(0);
else
if (znak ==  0)   %usvajamo da se menja sa +1
znak = 1;
                         one_cnt = 1;
                         zero_cnt = 0;
                         HDB3_gr(i)=1;
ERROR(i) = 1;
izlaz(i) = insert(1);
else
if (one_cnt >= 2) %slucaj kada su dve susedne jedinice
                                            %istog polariteta pa trenutnu nulu
                                            %konvertujemo u -1
                                            %(npr 1001000 pa dodje 0 --> ide u 1)
ERROR(i) = 1;
                             HDB3_gr(i) = -1*znak;
znak = -1*znak;
                             zero_cnt = 0;
                             one_cnt = 1;
izlaz(i) = insert(1);

else%susedne jedinice nisu istog znaka,
                                            %ovu nulu konvertujemo u poslednju
                                            %jedinicu
ERROR(i) = 1;
                             HDB3_gr(i) = znak;
                             one_cnt = one_cnt+1;
                             zero_cnt = 0;
izlaz(i) = insert(2);
end
end
end
end
end
         i = length(HDB3_gr)+1;
```

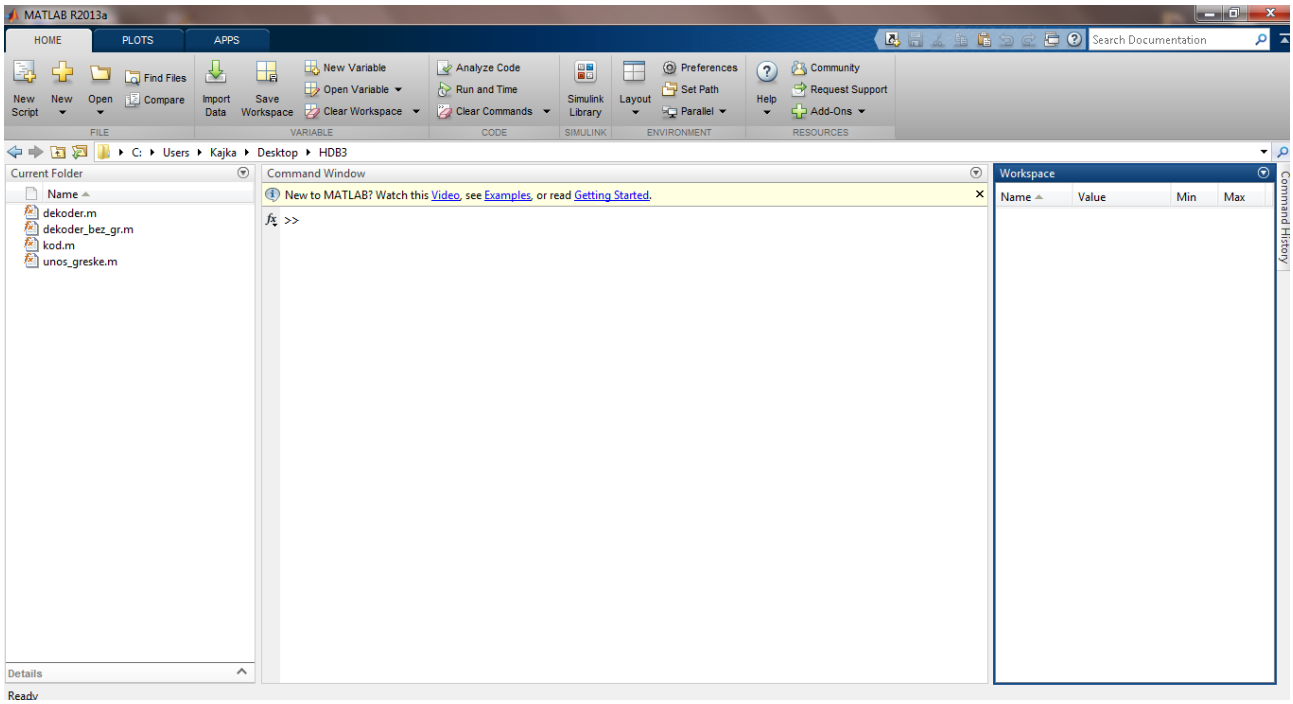
```
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;
izlaz(i) = insert(0);
             i = i+1;
det = 0; %brojac detektovanih gresaka
for i = 1 : length(ERROR)
if ERROR(i) == 1
det = det + 1;
end
end

          [p_det] = det;
          [nrz_novi] = izlaz(5:(length(izlaz)));
          [greska] = ERROR;
end
```

Poslednju *for* petlju koristimo kako bismo prebrojali broj detektovanih grešaka.

4. SIMULACIJA HDB3 KODERA/DEKODERA

U ovom poglavlju biće prikazani i analizirani rezultati koji su dobijeni pokretanjem simulacije navedenih kodova u programskom okruženju MATLAB R2013a, sa posebnom pažnjom na sposobnosti detekcije greške.

Pre početka simulacije neophodno je odabrati odgovarajući direktorijum u kome se nalaze funkcije. Na slici 4.1. prikazan je prozor u MATLABU pre same simulacije. Sa leve strane mogu se videti prethodno definisane funkcije čijim se pozivanjem izvršavaju kodiranje, unos greške odnosno dekodiranje zadate sekvence.



Slika 4.1. Izgled prozora pre početka simulacije

4.1. KODER

U potpoglavlju 3.2. detaljno je objašnjen rad kodera, a u ovom će biti prikazan i njegov rad na nekoliko primera.

Od zadatog ulaznog *NRZ* niza izlazni *HDB3* se dobija izvršavanjem naredbe:

```
HDB3 = kod(NRZ);
```

Uzmimo proizvoljni niz:

NRZ = [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,1,1,1,0,0,0,0,1,0];

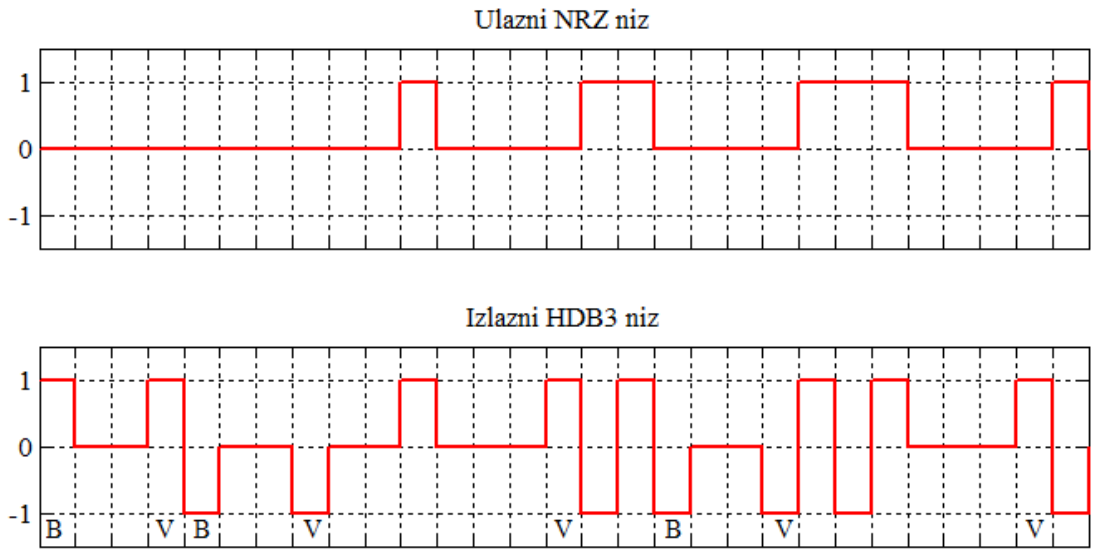
Nakon izvršavanja pozvane funkcije, njegova predstava u HDB3 kodu biće:

HDB3 = [1,0,0,1,-1,0,0,-1,0,0,1,0,0,0,1,-1,1,-1,0,0,-1,1,-1,1,0,0,0,1,-1,0];

Grafički prikaz ulaznog i izlaznog parametra dobijenog pokretanjem simulacije dat je na slici 4.1.1. Obeleženi su i *Balancing* i *Violation* biti. Kao primer je uzet niz sa velikim brojem nula sa ciljem da se ukaže na utiskivanje V i B bita.

Grafici su iscrtani pomoću sledećnih naredbi:

```
figure(1)
subplot(2,1,1); stairs([0 : length(NRZ)-1], NRZ); axis([0 length(NRZ) -1.5
1.5]); title('Ulazni NRZ niz'); grid on;
subplot(2,1,2); stairs([0 : length(HDB3)-1], HDB3); axis([0 length(HDB3) -1.5
1.5]); title('Izlazni HDB3 niz'); grid on;
```

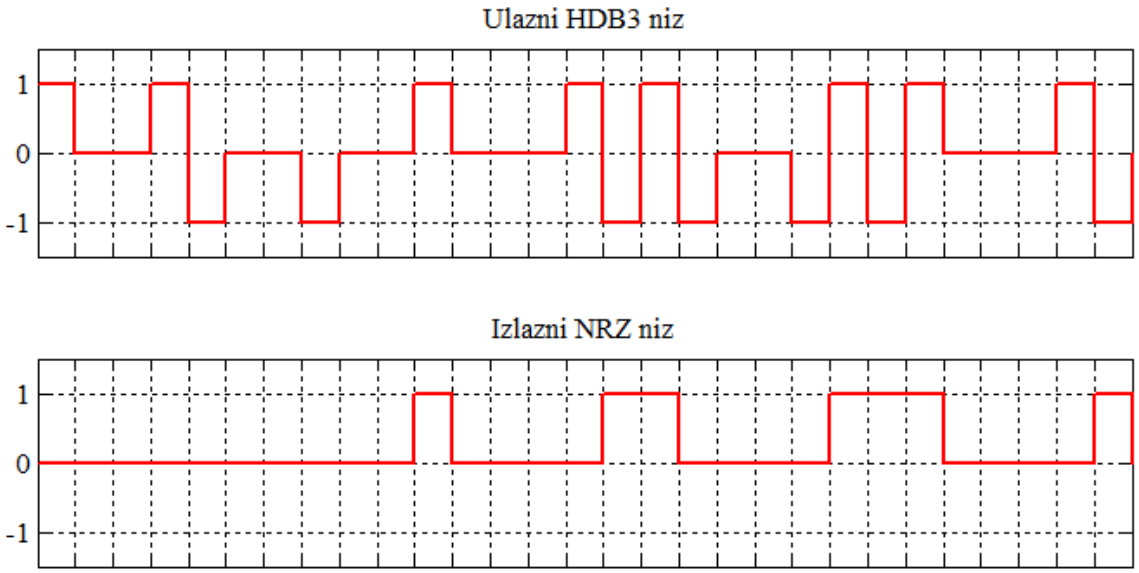


Slika 4.1.1. Ulazni NRZ niz i izlazni HDB3 niz dobijen pozivanjem funkcije *kod(NRZ)*

4.2. DEKODER BEZ GREŠKE

Dekoder bez greške je konstruisan tako da vrši inverznu funkciju onoj demonstriranoj u prethodnom poglavlju. Kao dokaz, primenićemo funkciju *dekoder_bez_gr*na novodobijeni niz *HDB3*. Rezultati su dobijeni unošenjem sledećih naredbi:

```
NRZ1 = decoder_bez_gr(HDB3);
figure(2)
subplot(2,1,1); stairs([0 : length(HDB3)-1], HDB3); axis([0 length(HDB3) -1.5
1.5]); title('Ulazni HDB3 niz'); grid on;
subplot(2,1,2); stairs([0 : length(NRZ1)-1], NRZ); axis([0 length(NRZ1) -1.5
1.5]); title('Izlazni NRZ niz'); grid on;
```



Slika 4.2.1. Rezultat simulacije dekodera bez greške

Kao što je moguće primetiti na slici 4.2.1, niz *NRZ* je uspešno rekonstruisan.

4.3. UNOS GREŠKE

Funkciju *unos_greske* primenićemo za više vrednosti ulazne promenljive *P* koja označava dozvoljenu verovatnoću greške. Parametri proračuna biće upisani u tabelu kako bi se dalja analiza učinila lakšom i preglednijom. Promenljiva *sent_cnt* dobijena je kao rezultat funkcije *unos_greske(HDB3,P)* i označava broj poslatih grešaka za različite verovatnoće pojavljivanja. Da bi se efikasnost dekodera u narednom odeljku što realnije prikazala, uzećemo niz sa velikim brojem bita (u konkretnom primeru dužina niza je 150 bita).

Ulazni niz i kod kojim se dobijaju vrednosti parametra *sent_cnt* dati su u nastavku teksta.

```
NRZ=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,0,0,0
,0,0,0,0,0,0,0,0,1,0,1,0,1,1,1,1,0,1,0,1,0,1,0,1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,1,1
,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1
,1,1,1,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0];
HDB3 = kod(NRZ);
[HDB3_gr1, sent_cnt1] = unos_greske(HDB3, 0.1);
[HDB3_gr2, sent_cnt2] = unos_greske(HDB3, 0.01);
[HDB3_gr3, sent_cnt3] = unos_greske(HDB3, 0.001);
[HDB3_gr4, sent_cnt4] = unos_greske(HDB3, 0.0001);
[HDB3_gr5, sent_cnt5] = unos_greske(HDB3, 0.00001);
```

U tabelu 4.3.1.su upisane vrednosti dobijene izvršavanjem navedenih naredbi.

Tabela 4.3.1. Broj pogrešno poslatih bita za različite verovatnoće pojave greške

P	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
sent_cnt	18	4	1	0	0

4.4. DEKODER SA GREŠKOM

Kako je prenos sa greškom realan slučaj, funkcija koja zavređuje najviše pažnje svakako jeste dekoder sa greškom. Upravo njenom simulacijom možemo da ispitujemo efikasnost primene HDB3 dekodera za prenos sa greškom. Pre nego što isprobamo dekoder na malopređašnjem nizu, osvrnućemo se na karakteristične greške koje se javljaju i ispitati dekoder na kratkim sekvencama koje obuhvataju ove slučajeve.

4.4.1. Nedozvoljene sekvence

i) Četiri ili više nula na ulazu

Neka se na ulazu nalazi sekvenca *HDB3_1* = [0,0,0,0,0,0,0,0,0,0]. Prikazaćemo vrednost promenljive *ERROR* koja beleži na kojim je bitima došlo do greške.

HDB3_1 =0 0 0 0 0 0 0 0 0 0

ERROR =0 0 0 1 0 0 0 1 0 0

Radi preglednosti, ova dva niza biće prikazana jedan ispod drugog. Crvenom bojom označeni su biti na kojima je došlo do greške. Izlazni NRZ niz sada ima vrednost

NRZ_gr = 0 0 0 1 0 0 0 0 0 0

pri čemu su usvojene pretrostavke da se obe nule na kojima je došlo do greške u originalnom nizu menjaju sa jedinicom.

ii) Dve uzastopne jedinice istog polariteta

Ovaj slučaj obuhvata pojavu dve uzastopne jedinice istog polariteta između kojih nema nula ili između kojih postoji samo jedna nula. Kao i u prethodnom primeru, nizove ćemo radi preglednosti napisati jedan ispod drugog.

HDB3_2 = 1 0 0 -1 0 -1 0 0 1 1 0 0

ERROR = 0 0 0 0 0 1 0 1 0 1 0 0

NRZ_gr = 1 0 0 1 0 0 0 0 1 0 0 0

Prva greška je detektovana kada se pojavila jedinica istog polariteta na razmaku od samo jedne nule od poslednje i u tom trenutku je vrednost date -1 prepravljena u 0. Dekoder nastavlja da radi sa izmenjenom vrednošću pa iz tog razloga pretpostavlja sledeći grešku dva bita kasnije, detektujući ovo kao četvrtu nulu u nizu. Ovu nulu menja u jedinicu koja je polariteta kao poslednja primljena jedinica, u ovom slučaju negativnog. Zatim nastavlja dekodovanje sa usvojenim izmenama i konačno detektuje poslednju grešku kada se pojave dve iste jedinice jedna za drugom. Druga jedinica se menja u nulu i do kraja niza ne prijavljuje se više grešaka.

iii) Tri ili više uzastopnih jedinica istog polariteta

Podrazumeva da su se javile tri ili više jedinice istog polariteta, bez obzira na broj nula između njih

HDB3_3 = 1 0 0 1 1 0 1 -1 -1 0 0 -1

ERROR = 0 0 0 0 1 0 1 0 1 0 0 0

NRZ = 0 0 0 0 0 0 0 1 0 0 0 0

U ovom slučaju, prva greška se prijavljuje na trećoj uzastopnoj jedinici istog polariteta (razmak između prve dve je dve nule tako da tu ne postoji greška). Trenutna jedinica iz ulaznog niza se zamenjuje nulom i nastavlja se dekodovanje. Sledeća jedinica ponovo dovodi do greške jer je uprkos ispravljenoj prethodnoj jedinici treća jedinica istog polariteta po redu pa se i ona menja nulom. Slede dve uzastopne jedinice negativnog polariteta, od kojih druga ponovo dovodi do pojave greške i menja se nulom. Ostatak sekvence je ispravan.

4.4.2. Analiza greške

U ovom odeljku ćemo primeniti prethodno objašnjen postupak dekodovanja na početni niz dužine 150 bita za različite verovatnoće greške, kako bismo iz te analize mogli da izvučemo zaključke o sposobnosti HDB3 dekodera da detektuje grešku. Informacija o broju detektovanih bita biće smeštena u tabelu radi preglednosti, a dobija se direktno primenom funkcije *dekoder(HDB3_gri)*, gde *i* uzima vrednost 1-5, u zavisnosti od toga za koju verovatnoću greške se računa ovaj parametar.

Tabela 4.4.2.1. Broj detektovanih grešaka za različite verovatnoće pojavljivanja

P	0.1	0.01	0.001	0.0001	0.00001
det_cnt	12	0	0	0	0

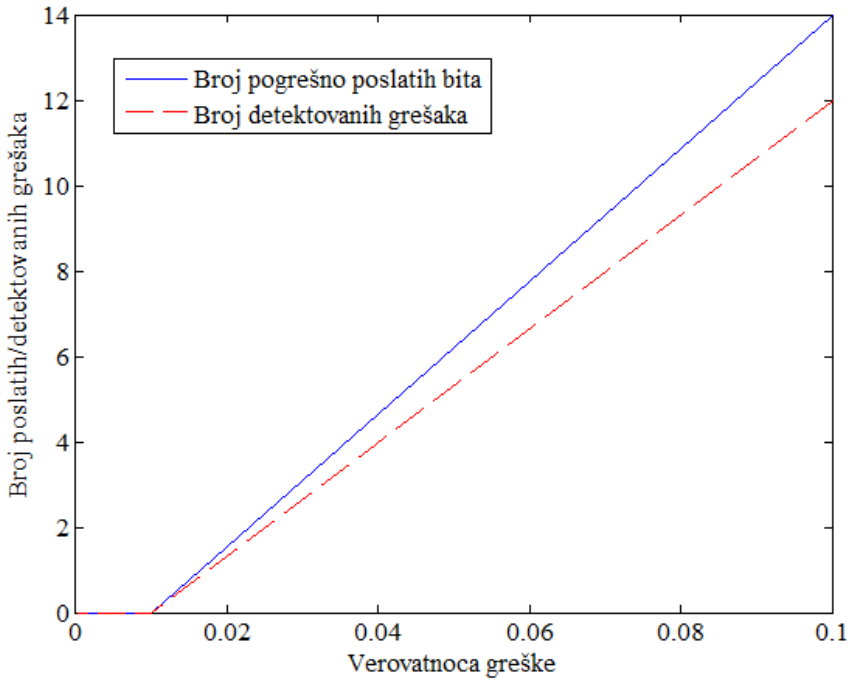
Rezultati funkcije upisani su u tabelu 4.4.2.1. i biće upotrebljeni za grafički prikaz odnosa broja poslatih i broja detektovanih grešaka. Vrednosti promenljive *det_cnt* dobijene su naredbama:

```
[det_cnt1, NRZ1] = dekoder(HDB3_gr1);
[det_cnt2, NRZ2] = dekoder(HDB3_gr2);
[det_cnt3, NRZ3] = dekoder(HDB3_gr3);
[det_cnt4, NRZ4] = dekoder(HDB3_gr4);
[det_cnt5, NRZ5] = dekoder(HDB3_gr5);
```

Grafički prikaz zavisnosti broja poslatih, odnosno broja detektovanih bita u zavisnosti od dozvoljene verovatnoće pojave greške prikazan je na slici 4.4.2.1 a realizuje se sledećim kodom:

```
G = [0.1, 0.01, 0.001, 0.0001, 0.00001];  %niz koji sadrzi razlicite vrednosti
                                          %verovatnoce greske
P = [sent_cnt1, sent_cnt2, sent_cnt3, sent_cnt4, sent_cnt5]; %niz sa brojem
                                          %poslatih greska za razlicite
                                          %verovatnoce greske
D = [det_cnt1, det_cnt2, det_cnt3, det_cnt4, det_cnt5]; %niz sa brojem
                                          %detektovanih greska za razlicite
                                          %verovatnoce greske
```

```
figure(3), plot(G, P, G, D, '--r'),ylabel('Broj poslatih/detektovanih grešaka'),
xlabel('Verovatnoca greške'),legend('Broj pogrešno poslatih bita', 'Broj
detektovanih grešaka')
```



Slika 4.4.2.1. Zavisnost broja pogrešnih ili detektovanih bita sa greškom u zavisnosti od dozvoljene verovatnoće pojavljivanja greške

Sa ovog grafika moguće je zaključiti da sa povećanjem dozvoljene verovatnoće greške opada sposobnost dekodera da detektuje pogrešno poslate bite. Ipak, u realnim sistemima su nizovi mnogo većih dužina a verovatnoća pojave greške je relativno mala tako da su rezultati koje projektovani dekoder daje više nego prihvatljivi.

U tabeli 4.4.2.2. prikazan je broj grešaka u izlaznom *NRZ* nizu u zavisnosti od verovatnoće pojave greške.

Tabela 4.4.2.2. Broj grešaka u izlaznom nizu u zavisnosti od verovatnoće greške

P	0.1	0.01	0.001	0.0001	0.00001
gr_cnt	19	0	0	0	0

Za relativno veliku verovatnoću greške ovaj broj je veliki, ali kako se današnji sistemi projektuju tako da je verovatnoća pojave greške znatno manja, moguće je uvideti da je projektovani dekoder jako efikasan. Rezultati su dobijeni (za svaku od verovatnoća greške) jednostavnom *for* petljom.

```
gr_cnt1=0;
gr_cnt2=0;
gr_cnt3=0;
```

```
gr_cnt4=0;
gr_cnt5=0;
for i = 1 : length(NRZ)
if NRZ(i) ~= NRZ1(i)
        gr_cnt1 = gr_cnt1 + 1;
end
end
for i = 1 : length(NRZ)
if NRZ(i) ~= NRZ2(i)
gr_cnt2 = gr_cnt2 + 1;
end
end
for i = 1 : length(NRZ)
if NRZ(i) ~= NRZ3(i)
        gr_cnt3 = gr_cnt3 + 1;
end
end
for i = 1 : length(NRZ)
if NRZ(i) ~= NRZ4(i)
gr_cnt4 = gr_cnt4 + 1;
end
end
for i = 1 : length(NRZ)
if NRZ(i) ~= NRZ5(i)
gr_cnt5 = gr_cnt5 + 1;
end
end
```

5. ZAKLJUČAK

U ovom radu prikazano je kako funkcionišu projektovani HDB3 koder i dekoder, na koji način se unosi greška i kako se rekonstruiše niz koji se šalje sa greškom. Iako nije moguće kompletno rekonstruisati ulazni NRZ niz, dekoder ima sposobnost detekcije greške, kao i mogućnost da donekle ispravi unetu grešku. Korišćeno okruženje MATLAB R2013a pruža neophodne alate za pregledno pisanje i testiranje koda.

Prednosti HDB3 dekodera sa greškom su svakako mogućnost da detektuje gde se dogodila greška, relativno jednostavna implementacija, kao i računarska efikasnost. Pri analizi rezultata primetili smo da za male verovatnoće pojave greške dekoder postiže jako dobre reultate, te je u skladu sa tim, kao i sa malopre pomenutim prednostima jako zahvalan za primenu.

Sa druge strane, ispitani slučajevi ne obuhvataju ni približan broj poslatih bita koliko ih se šalje u realnim sistemima pa su dobijeni rezultati donekle nepouzdani. Još jedna od mana ovako realizovanog dekodera jeste ta što pri verovatnoći greške koja je veća od 10^{-2} daje lošije rezultate i onemogućava pouzdan prenos.

Postojeća tema se može proširiti analizom drugih linijskih kodova i time utvrditi koje je najbolje koristiti za prenos digitalnih signala. Posebno zanimljivo bi bilo upoređivati BnZS (*Binary Zero Substitution*) kodove kod kojih se veći broj uzastopnih nula menja odgovarajućim kodnim dodatkom, a koji se koriste u Americi sa HDB3 kodom sa stanovišta detekcije i korekcije grešaka, što bi moglo biti obuhvaćeno u nekom od narednih radova.

LITERATURA

[1] G. Lukatela, D. Drajić, G. Petrović, “Digitalne telekomunikacije“, Građevinska knjiga, 1978.

[2] Zoran Čiča, “Komutacioni sistemi”,2013.

[3] M. L. Dukić, “Principi telekomunikacija“, Akademska misao, 2008.

[4] http://www.researchgate.net/publication/224593016_A_New_Design_of_HDB3_Encoder_and_Decoder_Based_on_FPGA

[5] http://en.wikipedia.org/wiki/Line_code