VEŽBA 8 – Realizacija tipične audio aplikacije na DSP platformi

8.1 Uvod

Algoritmi za audio kompresiju su razvijeni kao odgovor na zahtev za prenosom i skladištenjem visokokvalitetnog, širokopojasnog audio signala. Njihov osnovni cilj je predstavljanje digitalnog audio signala sa što manjim brojem bita, uz što veće očuvanje kvaliteta što podrazumeva transparentnost rekonstruisanog signala u odnosu na original. Kodovanje audio signala predstavlja tipičnu aplikaciju koja se realizuje na DSP procesorima. Stereo MP3 koder, sa prosečnih 128 kbit/s, je devedestih godina prošlog veka doneo revoluciju u oblasti kompresije i prenosa audio signala. Prvi prenosivi uređaj za reprodukciju MP3 muzičkih datoteka je razvio Institut Fraunhofer, naravno na digitalnom signal procesoru kao osnovi razvijenog sistema. Audio koderi nove generacije zahtevaju i do 24Mbit/s, kao i velik broj audio kanala, pa se realizuju skoro isključivo na DSP platformama.

U prethodnim vežbama su opisani elementi razvoja aplikacija za DSP platforme: prilagođenje referentog koda aritmetici DSP-a, primena C kompajlera, metode optimizacije, profilisanje koda, provera ispravnosti rešenja, kao i integracija blokova obrade u radno okruženje ciljne platoforme. Cilj ove vežbe jeste primena stečenog znanja na implementaciju tipičnog softverskog modula dekoderskog nivoa. Obradiće se jednostavan entropijski dekoder zasnovan na Rajs-Golombovom algoritmu, koji se najčešće koristi u sprezi sa drugim algoritmima kodovanja.

Vežba sadrži opis Rajs-Golombovog algoritma i opis formata bitskog toka koji služi za skladištenje ili prenos kodovanog signala. Nakon toga, sledi opis implementacije Rajs-Golombovog dekoderskog modula za platformu CS48x. Zadaci za samostalnu izradu obuhvataju implementaciju delova dekoderskog modula koji nedostaju.

Tokom izrade ovog zadatka prikazaće se tok implementacije dekoderskog modula u okviru radnog okruženja procesora CS48x, i način implemenentacije osnovih zadataka ovog modula koji obuhvataju:

- Preuzimanje komprimovanog bitskog toka iz ulaznog FIFO memorijskog niza
- Čitanje i parsiranje podataka iz FIFO memorijskog niza
- Konfiguracija radnog okruženja na osnovu primljenih podataka
- Primena algoritma za dekodovanje nad pročitanim podacima
- Smeštanje odbiraka dekodovanog signala u sistemski ulazno izlazni niz

8.2 Golomb-Rajsov algoritam kompresije

Golombov algoritam predstavlja oblik entropijskog kodovanja koji je razvio Solomon Golomb. Pogodan je za upotrebu nad skupovima podataka čije vrednosti prate geometrijsku raspodelu, odnosno raspodelu kod koje je verovatnoća pojavljivanja malih vrednosti veća od verovatnoće pojavljivanja velikih. Rajsov algoritam predstavlja specijalan slučaj Golombovog algoritma koji je nezavisno izveo Robert F. Rajs. Rajsov algoritam podrazumeva da je vrednost konfiguracionog parametra Golombovog algoritma uvek stepen broja 2. Zbog toga je ovaj algoritam pogodan za upotrebu na procesorima za digitalnu obradu signala, jer se time značajno pojednostavljuje sama implementacija (operacija deljenje se zamenjuje operacijom pomeranje bita udesno).

Kodovanje upotrebom Golomb-Rajsovog algoritma vrši se na sledeći način. Ukoliko se kompresuju odbirci koji pripadaju skupu brojeva A, koji sadrži i pozitivne i negativne brojeve, prvi korak u kodovanju predstavlja preslikavanje svih odbiraka tog skupa u skup nenegativnih brojeva:

$$\forall x \in A \Rightarrow x' = \begin{cases} 2x, x \geq 0 \\ -2x + 1, x < 0 \end{cases}$$

Ovako dobijeni brojevi se dalje, uz pomoć pomenutog konfiguracionog parametra ***M***, dele na unarni i binarni deo. Unarni deo ***q*** predstavlja rezultat celobrojnog deljenja reči *x'* sa *M*, dok binarni deo ***r*** predstavlja ostatak pri tom deljenju:

$$q = floor\left(\frac{x'}{M}\right)$$

$$r = x' - q \cdot M$$

Na osnovu dobijenih rezultata za unarni i binarni deo, kodna reč se sastavlja prema sledećem pravilu:

- Unarni deo se koduje tako što se u binarnoj predstavi brojeva predstavi kao niz od *q* jedinica koje se završavaju sa nulom, koja označava kraj sekvence.
- Nakon unarnog dela, upisuje se *N* bita binarnog dela gde je:

$$N = \log_2 M$$

U tabeli 8.1 dat je primer kodovanja brojeva 0-15 upotrebom Rajsovog algoritma za vrednost konfiguracionog parametra *M=4.*

Tabela 8.1 – Rezultat kodovanja upotrebom Rajsovog algoritma za M=4

Vrednost	Unarni deo	Binarni deo	Kodovana reč
0	0	0	0 00
1	0	1	0 01
2	0	2	0 10
3	0	3	0 11
4	1	0	1 0 00
5	1	1	1 0 01
6	1	2	1 0 10
7	1	3	1 0 11
8	2	0	11 0 00
9	2	1	11 0 01
10	2	2	11 0 10
11	2	3	11 0 11
12	3	0	111 0 00
13	3	1	111 0 01
14	3	2	111 0 10
15	3	3	111 0 11

Dekodovanje, odnosno, proces dobijanja originalne reči iz kodovanog bitskog toka može se podeliti u nekoliko koraka: čitanje unarnog dela, čitanje binarnog dela, združivanje unarnog i binarnog dela u jednu reč (*x'*) i određivanje znaka originalne reči.

Unarna reč se dobija čitanjem i brojanjem bita koji imaju vrednost jedan, sve dok se ne naiđe na prvi bit koji ima vrednost nula. Broj pročitanih jedinica ekvivalentan je unarnom delu *q*. Da bi se ispravno pročitao binarni deo kodne reci *r,* mora se znati vrednost konfiguracionog parametra *M* koji je korišćen prilikom kodovanja, na osnovu kojeg se dobija broj bita kojima je predstavljen binarni deo. Čitanje binarnog dela vrši se uzimanjem 2^M bita i posmatranjem pročitane vrednosti kao neoznačenog celog broja. Nakon što su pročitani, binarni i unarni deo se sastavljaju u jednu reč po sledećoj formuli:

$$x' = qM + r$$

Poslednji korak u dobijanju originalne reči x predstavlja rekonstrukcija znaka:

$$x = \begin{cases} \frac{x'}{2}, ako\ je\ x'\ paran\ broj \\ -\frac{x'}{2}, ako\ je\ x'\ neparan\ broj \end{cases}$$

8.3 Format bitskog zapisa signala

Pojednostavljeno, jedinica za kodovanje signala kompresuje audio signal i zapisuje kodovane podatke u odgovarajućem formatu. Pored kompresovanih podataka, bitski zapis sadrži i dodatne informacije kao što su: kofiguracioni parametri bitskog zapisa, kofiguracioni parametri kodera, dodatne informacije o sadržaju, broj kanala, frekvencija odabiranja itd. Format bitskog toka je strogo definisan odgovarajućim standardom.

Slika 8.1 – Format zapisa Rajsovog kodera

U slučaju Rajsovog kodera, bitski zapis je organizovan je kao niz sekvenci. Format ovog zapisa ilustrovan je na slici 8.1. Svaka sekvenca započinje sinhronizacionom reči, koja ujedno označava početak sekvence. Sinhronizaciona reč sastoji se iz dva dela: glavne sinhronizacione reči (0x52542d52) i njenog proširenja (0x4b). Ovakva organizacija svodi verovatnoću pojavljivanja lažne sinhronizacione reči na minimum.

Nakon sinhronizacione reči sledi zaglavlje sekvence, koje sadrži informacije o sadržaju same sekvence. Zaglavlje sekvence veličine je 11 bita. Polja koja se nalaze u zaglavlju su:

- Broj podsekvenci, čija je vrednost u opsegu [1, 32], umanjen za 1.
- Broj odbiraka signala u svakoj od podsekvenci koji može imati vrednost 2^4, 2^5… 2^{10}. Broj odbiraka predstavljen je brojem $k = \log_2(n)$, gde je n stvaran broj odbiraka.
- Polje koje određuje rezoluciju odbiraka kodovanog signala (16 ili 24 bita).

Tabela 8.2 sadrži strukturu zaglavlja sekvence. Informacije sadržane u zaglavlju sekvence zajedničke su za sve podsekvence.

Nakon zaglavlja sekvence, slede podsekvence. Svaka podsekvenca započinje zaglavljem podsekvence. Zaglavlje podsekvence sadrži samo jedno polje veličine 5 bita, koje predstavlja konfiguracioni parametar *M,* korišćen prilikom primene Rajsovog algoritma kompresije. Nakon zaglavlja, slede komprimovani podaci.

Tabela 8.2 – Struktura zaglavlja sekvence

Polje zaglavlja sekvence	Veličina polja (bit)	Značenje polja
Broj podsekvenci	5	$1 \leq (nPSK+1) \leq 32$
Broj odbiraka u podsekvenci	4	$nOdb = 2^k$ k=4, 5, 6, …, 10
Rezolucija odbiraka	2	1 => 16 bita; 2 => 24 bita

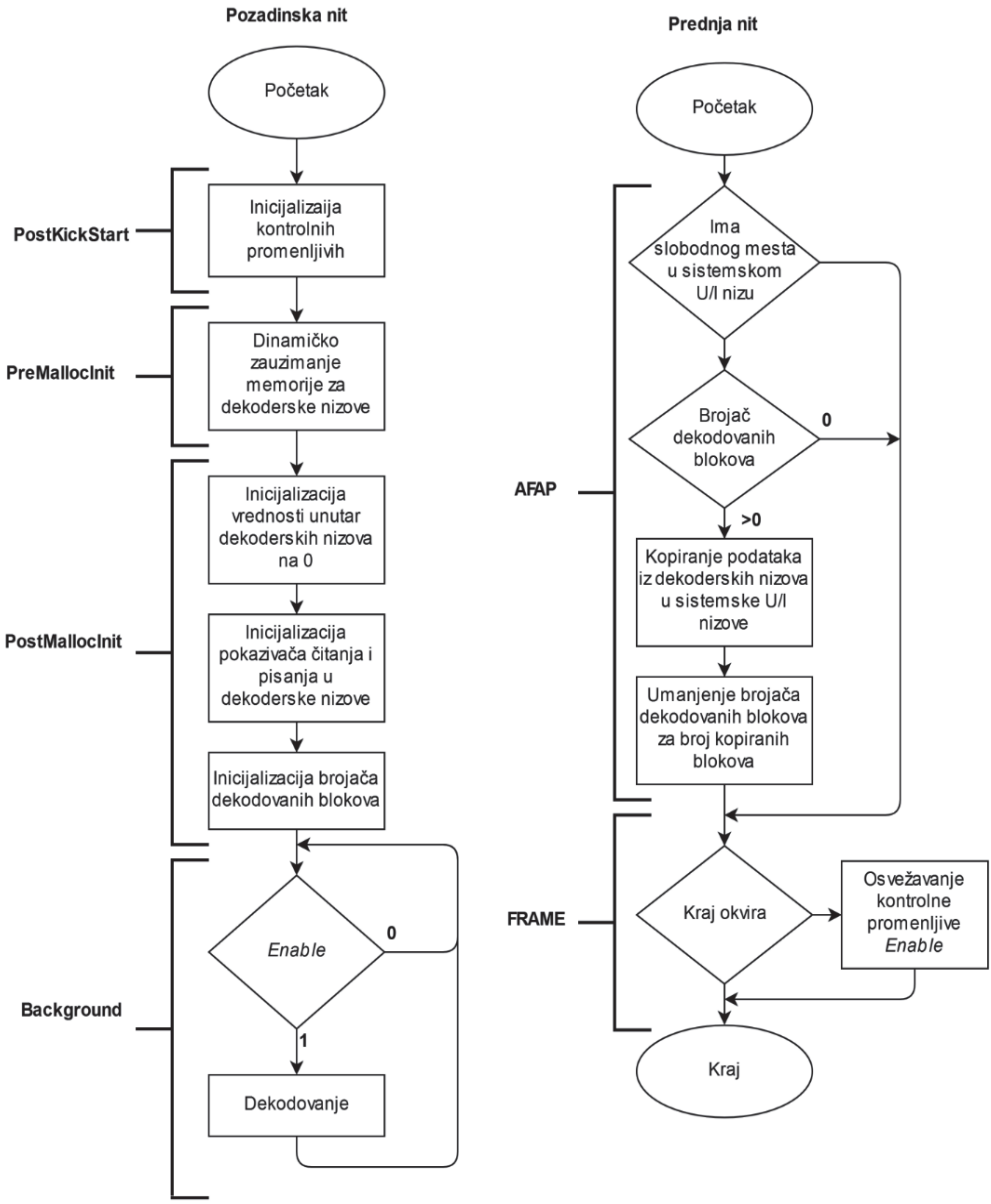
8.4 Implementacija Rajsovog dekodera na platformi CS48x

Kao što je objašnjeno u prethodnom poglavlju programska podrška dekoderskog nivoa u okviru radnog okruženja kod procesora CS48x ima zadatak da vrši čitanje bitskog toka iz ulaznog FIFO memorijskog niza, da dekododuje bitski tok i da dobijene PCM odbirke upiše u sistemske U/I nizove. U ovom poglavlju prikazan je primer implementacije Rajsovog dekodera i njegova sprega sa radnim okvirom.

8.4.1 Sprega dekodera sa radnim okvirom

Graf toka izvršenja opisanog funckija koje predstavljaju spregu dekoderskog modula sa radnim okruženjem prikazan je na slici 8.2. Dekoderski modul sadrži samo jedan kontrolni parametar vidljiv spolja, koji služi za omogućavanje ili onemogućavanje dekodovanja, tako da se MCV tabela modula sastoji iz jedne promenljive `X_VX_riceENABLE`. U slučaju kada je vrednost promenljive 1, dekodovanje je omogućeno.

U slučaju Rajsovog dekodera nije potrebna inicijalizacija pre pokretanja dekoderskog modula, zbog čega *PreKickstart* rutina ne sadrži obradu. Takođe, ne postoji zadatak koji se ponavlja periodično sa određenim vremenskim intervalom, tako da *Timer* rutina nije potrebna. Isti slučaj je i sa *Block* rutinom, s obzirom da se sva obrada nad dekodovanim blokovima obavlja na višim nivoima programske podrške.



Slika 8.2 – Graf toka izvršenja Rajsovog dekodera

Unutar *PostKickstart* rutine se vrši postavljanje indikatora o zauzeću memorije na 0. Pored toga izvršava se učitavanje vrednosti kontrola MCV tabele u lokalnu kopiju (inicijalizacija lokalne MCV tabele).

```
X_S_ricePostKickstart

      a0 = 0
      ymem[X_VY_riceMallocDone] = a0
      a0 = ymem[X_VY_riceHOST_ENABLE]
      xmem[X_VX_riceENABLE] = a0
```

PreMalloc rutina sadrži poziv funkcije za inicijalizaciju memorije koja će biti korišćena za smeštanje dekodovanih odbiraka. Funkcija za inicijalizaciju definisana je sa:

- `X_S_FrmSyncMalloc<mem>_I,`

gde *<mem>* predstavlja kvalifikator memorijske zone (X, Y ili L). Funkcija kao parametre očekuje adresu pokazivača na memoriju koja će biti zauzeta u registru i0 i veličinu memorije u registru x0.

```
X_S_ricePreMallocInit

      uhalfword(x0) = (PCM_DBLBUFF_SIZE)
      i0 = (X_VX_ricePCMbuff_base_Xptr)
      call X_S_FrmSyncMallocX_I
  ret
```

S obzirom da je privremeno smeštanje dekodovanih podataka realizovano koristeći *Ping Pong* princip skladištenja podataka, neophodno je za svaki kanal zauzeti memoriju veličine koja odgovara dvostrukoj dužini kanala. U slučaju jednostavnog Rajsovog dekodera, podrazumevan je samo jedan kanal. Veličina kanala definisana je sa:

- `PCM_BUFF_CH_SIZE .equ 1024`

Iz ovoga sledi da je potrebno zauzeti memorijski niz veličine 2048 reči. Konstanta *PCM_DBLBUFF_SIZE* predstavlja upravo ovu vrednost.

PostMalloc rutina inicijalizuje zauzetu memorije na 0. Potom, poziva se pomoćna rutina za postavljanje pokazivača na memorijske nizove za smeštanje dekodovanih odbiraka za kanale koji se koriste. Nakon toga, brojač dekodovanih blokova podataka se postavlja na 0. Na kraju, potrebno je indikator zauzeća memorije (*X_VY_riceMallocDone*) postaviti na 1, kako bi ostatak sistema znao da je zauzeće uspešno izvršeno.

```
X_S_ricePostMallocInit

      i6 = (PCM_DBLBUFF_SIZE)
      i7 = xmem[X_VX_ricePCMbuff_base_Xptr]
      xmem[X_VX_ricePCMbuff_Wr_Xptr] = i7
      xmem[X_VX_ricePCMbuff_Rd_Xptr] = i7

      a0 = 0
      do(i6),>loop
%loop xmem[i7] = a0; i7+=1
      ymem[X_VY_ricePCMbrickCounter] = a0

      call I_S_riceResetDecoderBufferPtrs

      uhalfword(a0) = (1)
      ymem[X_VY_riceMallocDone] = a0
  ret
```

Ova realizacija dekoderskog modula pretpostavlja da se vrednost kontrolnih promenljivih može menjati isključivo na granicama okvira. Iz tog razloga, ažuriranje vrednosti lokalne MCV tabele vrši se prilikom poziva *Frame* rutine. U slučaju prelaska iz stanja kada je onemogućena obrada u stanje kada je obrada omogućena, neophodno je izvršiti ponovnu inicijalizaciju memorije.

```
X_S_riceFrame
      a0 = ymem[X_VY_riceHOST_ENABLE]      # a0 = nova vrednost
      b0 = xmem[X_VX_riceENABLE]      # b0 = prethodna vrednost

      i0 = (X_BX_riceMCV)
      i4 = (X_BY_riceMCV)
      do(HOST_COEFS_COUNT), >
            y0 = ymem[i4]; i4 += 1
%           xmem[i0] = y0; i0 += 1

      # maskiranje kontrolne promenljive
      uhalfword(a1) = (1)
      a0 = a0 & a1                  # a0 = novo stanje
      b0 = b0 & a1                  # b0 = prethodno stanje
      a0-b0

      if (a>=0)  jmp>riceFrame_ret
            call I_S_riceSoftReInit
%riceFrame_ret
  ret
```

Sam proces dekodovanja vrši se u okviru *Background* rutine. Sama rutina započinje podešavanjem kontrolnih promenljivih radnog okvira (opisanih u poglavlju 7). Lokalne kopije vrednosti ovih promenljivih nalaze se u strukturi *I_BX_riceFrameData.* Kopiranje vrednosti ove strukture u odgovarajuće promenljive, vrši se pozivom funkcije *X_S_CopyReinitFrameDataToOS_i* kojoj se kao parametar prosleđuje pokazivač na strukturu u registru *i7*. Nakon toga, sledi čekanje na završetak zauzeća memorije od strane radnog okvira. Ukoliko je memorija uspešno zauzeta i kontrolna promenljiva koja određuje da li je omogućeno dekodovanje postavljena na 1, poziva se funkcija za dekodovanje *X_S_riceModuleRiceDec*.

```
X_S_riceBackground
      a0 = ymem[X_VY_riceMallocDone]
      a0&a0
      if (a!=0) jmp >ready_to_decode
            uhalfword(a0) =(I_BX_riceFrameData)
            ymem[X_VY_riceDecFrameDataPtr] = a0
            i7 = ymem[X_VY_riceDecFrameDataPtr]
            call X_S_CopyReinitFrameDataToOS_i

            # Čekanje na završetak zauzimanja memorije.
%Wait_For_Malloc_Done
                  a0 = ymem[X_VY_riceMallocDone]
                  a0&a0
            if (a==0) jmp <Wait_For_Malloc_Done
            #---------------------------------

%ready_to_decode
      # Zauzimanje memorije završena, započinje dekodovanje
      x0 = xmem[X_VX_riceENABLE]
      bittst lo(x0),(MCV_CONTROL_ENABLE)
      if(z==0) jmp>riceBackground_ret
            call X_S_riceModuleRiceDec

%riceBackground_ret
  ret
```

S obzirom da *Block* i *Timer* rutine ne koriste, *Background* rutinu prekidaju *Frame* i *AFAP* rutine. *AFAP* rutina koristi se za proveravanje da li postoje dekodovani skupovi odbiraka, i ukoliko postoje, kopiraju se u sistemske ulazno/izlazne memorijske nizove. Unutar ove funkcije prvo se vrši provera da li u okviru sistemskih U/I memorijskih nizova postoji dovoljno slobodnog mesta za smeštanje novog bloka odbiraka koristeći kontrolnu promenljivu radnog okvira *X_VY_IO_Free*.

Ukoliko ne postoji, funkcija se završava. Nakon toga proverava se da li je broj dekodovanih blokova (*ricePCMbrickCounter*) različit od 0. Potrebno je voditi računa da prvo kopiranje dekodovanih odbiraka može započeti tek kada se napune dekoderski memorijski nizovi (*X_VY_ricePCMbrickCounter* treba da se izjednači sa *riceMaxPCM_BRICK_DIPSTICK*). Ovo se radi samo jednom, na početku dekodovanja, kako bi se smanjio rizik od kašnjenja dekodovanih odbiraka tokom dalje obrade. Sledeća akcija jeste podešavanje pokazivača na lokalne kopije vrednosti kontrolnih promenljivih radnog okvira. Ove vrednosti se postavljaju samo u slučaju prvog bloka podataka u okviru, u suprotnom vrednost ovog pokazivača potrebno je postaviti na 0. Ažuriranje vrednosti kontrolnih promenljivih je neophodno kako bi se omogućilo podešavanje radnog okvira u skladu sa informacijama koje nosi zaglavlje okvira (najčešće su to frekvencija odabiranja, broj kanala, itd.). Samo kopiranje odbiraka iz dekoderskih nizova u ulazno/izlazne memorijske nizove vrši se pozivom pomoćne funkcije *X_S_CopyBrickToIO_i* koja kao parametar očekuje pokazivač na niz pokazivača na dekoderske nizove. Nakon toga, poziva se pomoćna funkcija za ažuriranje pokazivača na dekoderske nizove u skladu sa *Ping Pong* šemom skladištenja PCM odbiraka objašnjenom u prethodnom poglavlju.. Pokazivači se uvećavaju za veličinu bloka, a u slučaju dolaska do kraja memorijskog niza, resetuju se na početak. Poslednji korak je umanjenje brojača dekodovanih blokova za 1.

```
X_S_riceAFAP
      a0 = ymem[X_VY_riceMallocDone]
      a0&a0
      if (a==0) jmp >riceAFAP_ret

      # Provera da li ima slobodnog mesta u okviru IO nizova
      a0 = ymem[X_VY_IO_Free]
      uhalfword(a1) = (2*BRICK_SIZE)
      a0-a1
      if (a <= 0) jmp >riceAFAP_ret

      # Provera da li postoje dekodovani blokovi odbiraka
      a0 = ymem[X_VY_ricePCMbrickCounter]
      a0&a0
      if (a==0) jmp >riceAFAP_ret

      # Popunjavanje svih nizova na početku dekodovanja
      b0 = ymem[X_VY_ricePingPongGateOpen]
      b0&b0
      if (b!=0) jmp>SkipStartup
            uhalfword(b0) = (riceMaxPCM_BRICK_DIPSTICK)
            a0-b0
```

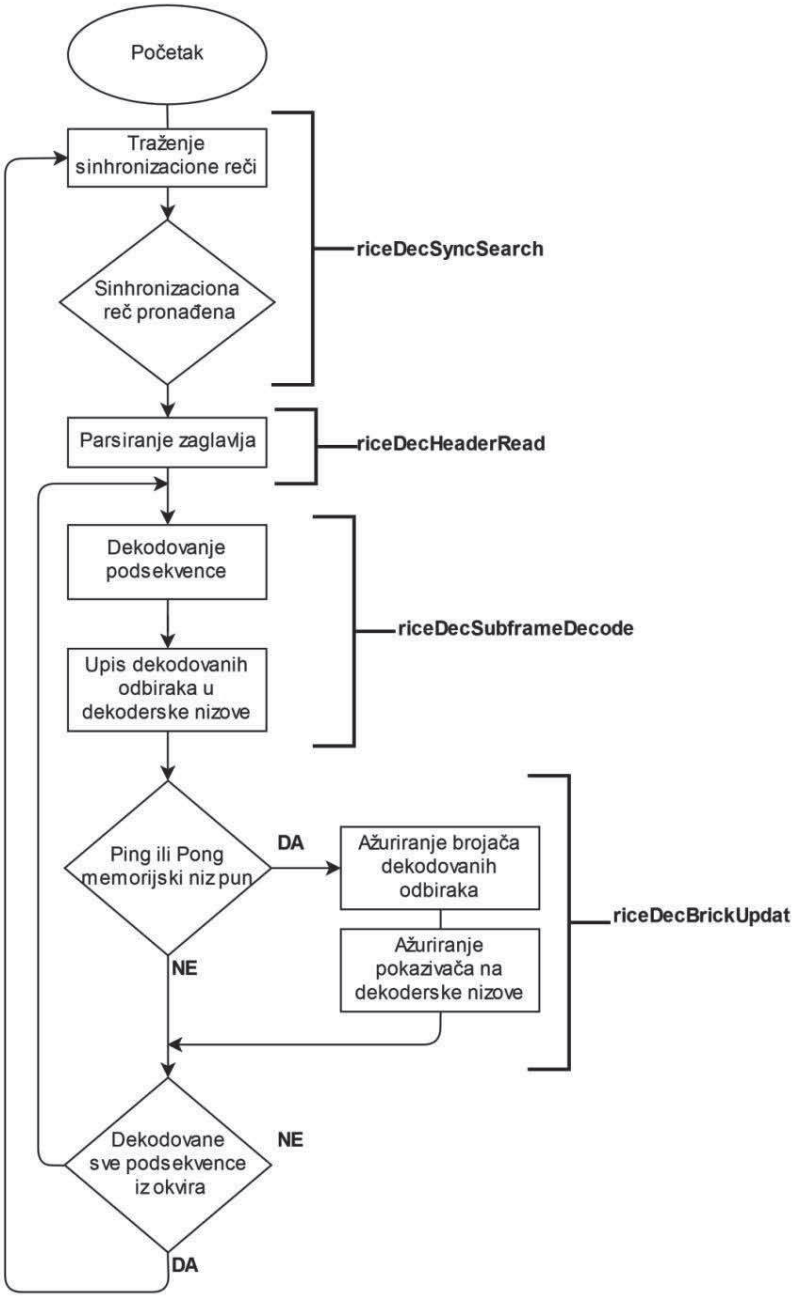
```
            if (a<0) jmp>riceAFAP_ret
                  ymem[X_VY_ricePingPongGateOpen] = a0
%SkipStartup

            # U slučaju granice okvira ažuriraju se i vrednosti
            # kontrolnih promenljivih radnog okvira
            a0 = ymem[X_VY_ricePCMbrickCounter]
            uhalfword(b0) = (ricePCM_FRM_BOUNDARY)
            a0&b0; b0 = a0l
            if (a!=0) jmp>not_the_first_brick
                  uhalfword(b0) =(I_BX_riceFrameData)
%not_the_first_brick
            ymem[X_VY_riceDecFrameDataPtr] = b0

            #Kopiranje odbiraka iz dekoderskih nizova u IO nizove
            i7 = (X_BY_riceDecoderPtrs)
            call X_S_CopyBrickToIO_i

            # Ažuriranje pokazivača
            call I_S_riceUpdateDecoderPCMPtrs

            # Ažuriranje brojača dekodovanih blokova
            a0 = ymem[X_VY_ricePCMbrickCounter]
            uhalfword(b0) = (0x0001)
            a0 = a0 - b0
            ymem[X_VY_ricePCMbrickCounter] = a0

%riceAFAP_ret
  ret
```

8.4.2 Parsiranje bitskog toka i primena Rajsovog algoritma za dekodovanje

Graf toka izvršenja operacije dekodovanja dat je na slici 8.3.

Kao što je napomenuto u okviru opisa *Background* rutine, dekodovanje bitskog toka započinje pozivom funkcije *X_S_riceModuleRiceDec.* Za parsiranje bitskog toka koristi se *BitRipper* biblioteka opisana u prethodnom poglavlju. Dekodovanje započinje traženjem sinhronizacione reči. Kada je sinhronizaciona reč uspešno prnađena sledi parsiranje zaglavlja sekcije. Ukoliko prilikom parsiranja zaglavlja nije došlo do greške pristupa se dekodovanju sadržaja okvira bitskog toka.

Telo funkcije *X_S_riceModuleRiceDec* prikazano je na stranici 199.



Slika 8.3 – Graf toka izvršenja funkcija dekodovanju

```
X_S_riceModuleRiceDec
      # Traženje sinhronizacione reči
      call I_S_riceDecSyncSearch

      # Parsiranje zaglavlja bitskog toka
      call I_S_riceDecHeaderRead

      # Dekodovanje bitskog toka
      uhalfword(b0) = (1)
      a0 = ymem[X_VY_riceDecErrorFlag]
      a0 - b0
      if (a==0) jmp>riceDecMain_ret
            call I_S_riceDecSubframeDecode
%riceDecMain_ret:
      ret
```

Traženje sinhronizacione reči realizovano je u okviru funkcije *I_S_riceDecSyncSearch*. Funkcija je blokirajuća, odnosno traženje reči traje sve dok sinhronizaciona reč ne bude uspešno pronađena. Na početku funkcije se u registre *a3* i *b3* smesti očekivana vrednost sinhronizacione reči i njenog produžetka (*X_CY_riceDecSyncWord* i *X_CY_riceDecSyncWordExt*). Neophodno je pre traženja izvršiti poravnanje bitskog toka pozivom funkcije *X_S_BR_AlignToDword_i*. Nakon toga vrši se čitanje 32 bita iz FIFO memorije i provera jednakosti sa sinhronizacionom reči. Za čitanje bita koristi se funkcija *X_S_BR_Extract_i*.

```
I_S_riceDecSyncSearch:
      a3  =  0
      b3  =  0
      a3h = ymem[X_CY_riceDecSyncWord]
      b3h = ymem[X_CY_riceDecSyncWordExt]

      call X_S_BR_AlignToDword_i
%do_loop
      uhalfword(a0) = (32)
      call X_S_BR_Extract_i
      b0 - a3
      if (b!=0) jmp <do_loop

      uhalfword(a0) = (8)
      call X_S_BR_Peek_i
      b0 - b3
      if (b!=0) jmp <do_loop
           call X_S_BR_Extract_i
      ret
```

Ukoliko pročitana vrednost odgovara sinhronizacionoj reči, vrši se provera narednih 8 bita bez pomeranja pokazivača čitanja (koristeći funkciju *X_S_BR_Peek_i*). Ukoliko pomenutih 8 bita sadrži odgovarajući nastavak sinhronizacione reči, vrši se čitanje tih 8 bita iz bitskog toka i funkcija se završava, u suprotnom ponavlja se korak pretrage. Obe funkcije za čitanje bita kao argument primaju broj bita u registru *a0*, dok je povratna vrednost funkcije smeštena je u *b0*.

```
I_S_riceDecSubframeDecode:
       uhalfword(b3) = (1)
       i2 = ymem[X_VY_riceDecSubframeCount]
       i3 = ymem[X_VY_riceDecSamplePerSubframeCount]
       do(i2),>SubframeLoop
            i1 = xmem[X_VX_ricePCMbuff_Wr_Xptr]
            # Čitanje zaglavlja podsekvece (N = logM)
            uhalfword(a0)=(5)
            call X_S_BR_Extract_i
            i4 = b0
            do(i3),>SampleLoop
            # Brojanje jedinica do prve 0, a3 se koristi kao brojač
                  uhalfword(a3) = (0)
                  uhalfword(a0)=(5)
%UnaryPartLoop    call X_S_BR_Extract_i
                  b0 & b0
                  if (b==0) jmp >EndUnaryPart
                        a3 = a3 + b3
                        jmp <UnaryPartLoop
%EndUnaryPart
                  # Čitanje binarnog dela:
                  a0 = i4            # i4 = N
                  a0 = a0 + b3
                  call X_S_BR_Extract_i

                  b2 = b0
                  b2 = b2 & b3 # Određivanje znaka
                  b0 = b0 >> 1
                  b0l = 0
                  a0 = i4      # a0 = N
                  a0 & a0      # preskoči množenje ako je N nula
                  if (a==0) jmp>Skip_shift
                        do(i4),>ShiftL  # Logički pomeraj u levo za
N
%ShiftL                       a3 = a3 << 1     #
%Skip_shift
                   a3 = a3 | b0    # Sabiranje unarnog i binarnog
dela
```

```
                    b2 & b2
                    if (b==0) jmp >Keep_sign
                          a3 = -a3
%Keep_sign
                     # Upisivanje odbirka u dekoderski niz
%SampleLoop          xmem[i1] = a3; i1+=1
                     nop

             xmem[X_VX_ricePCMbuff_Wr_Xptr] = i1
             call I_S_riceDecBrickUpdate
%SubframeLoop
      ret
```

Nakon uspešnog pronalaska sinhronizacione reči pristupa se parsiranju zaglavlja sekvence podataka. Vrednosti pročitane iz zaglavlja upisuju se u odgovarajuće kontrolne promenljive, koje će biti korišćene u daljem dekodovanju. Promenljive koje je potrebno popuniti su:

- *X_VY_riceDecSubframeCount* – broj podsekvenci u sekvenci podataka
- *X_VY_riceDecSamplePerSubframeCount* – broj odbiraka u svakoj podsekvenci
- *X_VY_riceDecBitResolution* – rezolucija odbiraka (1 ako je 16 bita, 2 ako je 24 bita)

Pored ove tri promenljive, postoji i promenljiva koja služi za signalizaciju greške ostatku dekoderskog modula *X_VY_riceDecErrorFlag.* Ovoj promenljivoj neophodno je dodeliti vrednost različitu od 0, u slučaju da je došlo do greške prilikom parsiranja zaglavlja.

Parsiranje započinje čitanjem prvog polja zaglavlja, dužine 5 bita. Pročitana vrednost se uvećava za 1 i smešta u promenljivu *X_VY_riceDecSubframeCount.* Potom se vrši čitanje naredna 4 bita iz bitskog toka. Potrebno je proveriti da li se pročitana vrednost nalazi u dozvoljenom opsegu [4, 10]. Ukoliko je taj uslov zadovoljen, na osnovu pročitane vrednosti izračunava se broj odbiraka u svakoj podsekvenci zato što se broj 1 pomeri u levo za *N* mesta, gde je *N* pročitana vrednost. Pomeranje za *N* mesta se može realizovati koristeći hardversku petlju, koja će izvršiti pomeraj za jedno mesto *N* puta. Izračunata vrednost smešta se u promenljivu *X_VY_riceDecSamplePerSubframeCount.* Nakon toga sledi čitanje polja dužine 2 bita i vrši se provera da li je pročitana vrednost validna (1 ili 2). U slučaju da jeste, upisuje se u promenljivu *X_VY_riceDecBitResolution.*

Ukoliko je funkcija za parsiranje zaglavlja uspešno izvršena, odnosno nije došlo do greške prilikom njenog izvršenja, poziva se funkcija koja sadrži implementaciju algoritma za dekodovanje bitskog toka *I_S_riceDecSubframeDecode*.

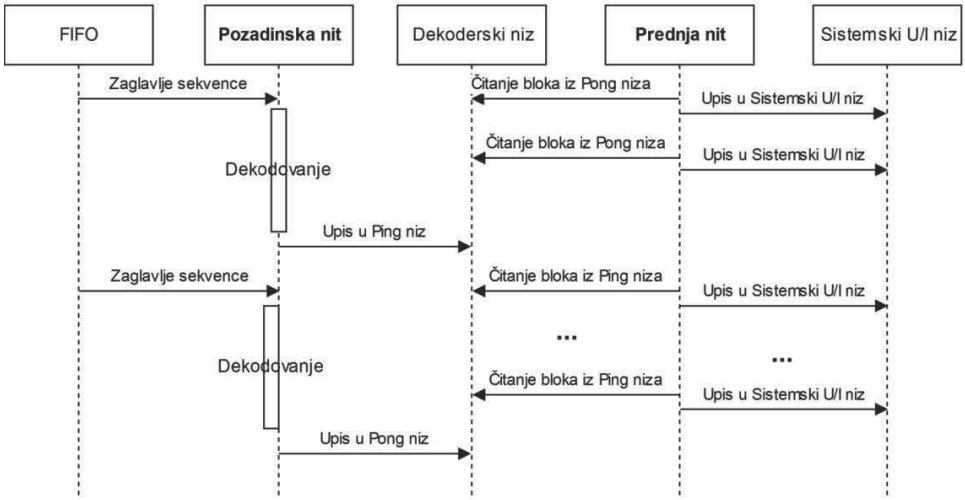
Telo funkcije sadrži hardversku petlju koja se ponavlja onoliko puta koliko postoji podsekvenci u aktivnoj sekvenci podataka (*X_VY_riceDecSubframeCount*). Za svaku podsekvencu čita se 5-bitno zaglavlje podsekvence koje sadrži broj bita sa kojima je predstavljen binarni deo kodovane reči (2^M). Nakon toga sledi hardverska petlja u tajanju od *n* iteracija, gde je *n* broj odbiraka u sekvenci (*X_VY_riceDecSamplePerSubframeCount*). Za svaki odbirak određuje se broj uzastopnih jedinica tako što se podaci iz bitskog toka čitaju bit po bit i uvećava se brojač dok se ne naiđe na 0. Potom se čita binarni deo koristeći pročitani podatak o njegovoj veličini. Sledi rekonstrukcija stvarne vrednosti odbirka po opisanom algoritmu. Određuje se znak odbirka uzimanjem vrednosti najnižeg bita binarnog dela (operacija logičkog i sa maskom 0x0001), i vrednost binarnog dela se deli sa 2 (operacija logičkog pomeraja udesno). Unarni deo se množi sa faktorom M korišćenim prilikom kodovanja. Vrši se sabiranje unarnog i binarnog dela nakon čega se primenjuje sačuvani znak. Dobijeni odbirak se upisuje u dekoderski memorijski niz.

```
I_S_riceDecBrickUpdate:
      a0 = ymem[X_VY_riceDecBrickCounter]
      b0 = ymem[X_VY_riceDecSamplePerSubframeCount]
      do(4),>
%           b0 = b0 >> 1
      a0 = a0 + b0
      ymem[X_VY_riceDecBrickCounter] = a0
      uhalfword(b0) = (riceMaxPingPong_BRICK_COUNT)
      a0 - b0
      if (a<0) jmp>endif
            call I_S_riceDecPCMUpdate
            a0 = 0
            ymem[X_VY_riceDecBrickCounter] = a0
%endif
      ret

I_S_riceDecPCMUpdate:
      intdis
      a0 = ymem[X_VY_ricePCMbrickCounter]   # globalni brojač
      b0 = ymem[X_VY_riceDecBrickCounter]   # lokalni brojač
      a0 = a0 + b0
      ymem[X_VY_ricePCMbrickCounter] = a0
      inten
```

```
%wait       a0 = ymem[X_VY_ricePCMbrickCounter]
            b0 = ymem[X_VY_riceDecBrickCounter]
           a0-b0
      if (a > 0) jmp <wait

      a0 = xmem[X_VX_ricePCMbuff_base_Xptr]
      b0 = xmem[X_VX_ricePCMbuff_Wr_Xptr]
      uhalfword(a1) = (PCM_BUFF_CH_SIZE)
      a1 = a1 + a0
      uhalfword(a2) =(1)
      b0 = b0 - a2
      a1 - b0;  b0 = a1
      if (a>0) jmp>endif
             b0 = a0
%endif
      xmem[X_VX_ricePCMbuff_Wr_Xptr] = b0
      ret
```

Nakon uspešnog dekodovanja podsekvence poziva se funkcija koja služi za ažuriranje trenutnog stanja memorije *I_S_riceDecBrickUpdate.* Zadatak ove funkcije je da ažurira vrednost lokalnog brojača dekodovanih blokova odbiraka. Brojač dekodovanih blokova se uvećava za broj dekodovanih odbiraka podeljen sa veličinom bloka. Kada se na osnovu ovog brojača detektuje ispunjenost Ping odnosno Pong memorijskog niza, vrednost brojača se dodaje na trenutnu vrednost globalne promenljive *X_VY_ricePCMbrickCounter.* Kao što je već objašnjeno, ova promenljiva se koristi u *AFAP* rutini za proveru broja blokova spremnih za kopiranje u sistemske ulazno/izlazne memorijske nizove. Pored postavljanja ove vrednosti, nakon ispunjenja Ping ili Pong memorijskog niza vrši se i smena pokazivača pisanja na drugi niz (sa Ping na Pong ili obrnuto). Pre daljeg dekodovanja neophodno je sačekati da novi niz na koji se pokazuje bude prazan (svi prethodni blokovi prekopirani u u/i nizove).

Na slici 8.4 prikazan je graf izvršenih akcija u toku dekodovanja koji ilustruje na koji način rutine prednje i pozadinske niti vrše čitanje i upis podataka u memoriju. Prikazani grad ilustruje situaciju u toku dekodovanja nakon izvršene inicijalizacije i podrazumeva da je kontrolna promenljiva *enable* postavljena tako da je dekodovanje omogućeno.



Slika 8.4 – Graf izvršenih akcija u toku dekodovanju

8.5 Zadaci za samostalnu izradu

8.5.1 Zadatak 1: Implementacija funkcije za parsiranje zaglavlja sekvence u okviru Rajsovog dekodera

Dat je opis realizacije Rajsovog dekodera kao modula u okviru radnog okruženja namenjenog sistemima baziranim na procesoru CS48x. Opisana je sprega dekoderskog modula sa radnim okruženjem, kao i sama implementacija algoritma dekodovanja. U okviru priložene implementacije nedostaje funkcija za parsiranje zaglavlja sekvence `I_S_riceDecHeaderRead`. Cilj ovog zadatka jeste realizacija te funkcije.

8.5.1.1 Postavka zadatka:

1. Uvući sve projekte iz direktorijuma *Zadatak1* u razvojno okruženje.
2. Otvoriti projekat riceDecoder modula i izvršiti analizu izvornog koda.
    a. riceDataVars.a – sadrži globalne promenljive korišćene od strane više izvornih datoteka
    b. riceOsInterface.a – sadrži strukture koje predstavljaju spregu sa radnim okruženjem
    c. riceModuleMain.a – sadrži implementaciju rutina koje čine spregu sa radnim okruženjem
    d. riceModuleRiceDec.a – sadrži implementaciju algoritma dekodovanja
3. Unutar riceModuleRiceDec.a datoteke implementirati funkciju za parsiranje zaglavlja sekvence. Opis same funkcije, njene svrhe i zadatka koji ona obavlja dat je u tekstu.
4. Za čitanje bitskog toka koristiti funkcije iz *BitRipper* biblioteke.
5. Pokrenuti projekat na simulatoru.
6. Kao ulaznu datoteku koristiti *Rice_LossLessEnc.raw.*
7. Uporediti dobijenu izlaznu datoteku sa referentnom datitekom Rice_LossLessEnc_ref.pcm.

8.5.2 Zadatak 2: Uvezivanje dekoderskog modula i modula naknadne obrade

Cilj ovog zadatka jeste povezivanje realizovanog Rajsovog dekodera i modula za dodavanje višestrukog eho efekta realizovanog u prethodnoj vežbi u jednu aplikaciju.

8.5.2.1 Postavka zadatka:

1. Uvući projekat *multitapEcho_framework* iz prethodne vežbe u razvojno okruženje i prevesti.

2. Otvoriti projekat *simulator_app* u okviru CLIDE razvojnog okruženja.
3. Postaviti modul *OS* na *a0* nivo programske podrške, modul *riceDecoder* na nivo *a1* i modul *multitapEcho_framework* na nivo *a2.*
4. Odabrati za ulaznu datoteku *Rice_LossLessEnc.raw.*
5. Pokrenuti izvršavanje projekta. **Ne menjati** kontrole u toku izvršenja.
6. Uporediti izlaznu datoteku sa referentnom.
7. Izvršiti procenu utroška resursa čitave aplikacije.