SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 301

Direktna digitalna sinteza sinusa visoke točnosti na procesoru OMAP porodice

Marko Brezović

Zagreb, lipanj 2011.

Svima koji su mi pomogli savjetima i objašnjenjima prilikom pisanja ovoga rada, zahvaljujem.

Posebna zahvala prof. dr. sc. Davoru Petrinoviću na mentorstvu!

Sadržaj

1. Uvod

Direktna digitalna sinteza je metoda kojom je moguće realizirati digitalne signale različitih valnih oblika. Postoji nekoliko osnovnih tehnika za direktnu digitalnu sintezu:

- kutna dekompozicija
- osnovne metode kutne rotacije
- kompresija sinusne amplitude
- polinomne aproksimacije
- kombinacija faze u PSAC-u (eng. Phase to Sinus Amplitude Conversion)

U radu se govori o direktnoj digitalnoj sintezi sinusnog signala visoke točnosti korištenjem po dijelovima glatkog kubičnog polinoma gdje se kao ulaz zadaje početna faza, a na izlazu dobiva jedan uzorak sinusoide.

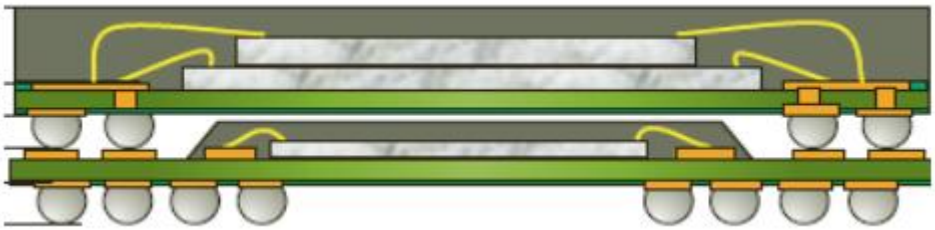
Sustav na kojem će se izvršavati direktna digitalna sinteza je *Texsas Instruments Beagleboard*. *Beagleboard* je izrađen tako da bi bio vrlo jeftin i podržan je od otvorene zajednice (eng. *Open Community*) pa postoje besplatni alati za razvoj programa. Upravo to ga čini popularnim u raznim krugovima npr. sveučilišnim.

Na Beagleboardu je OMAP3530 procesor koji između ostalog ima glavni procesor ARM Cortex A8 i koprocesor TM320C64x+.

Direktna digitalna sinteza sinusa će se izvoditi na oba procesora.

2. Beagleboard

Korištena razvojna pločica za izradu diplomskog rada je Beagleboard[1]. Beagleboard je cijenom vrlo pristupačna i s vrlo moćnim procesorom kojem nije potrebno hlađenje, razvio ju je *Texas Instruments*. Bazirana je na *Texas Instruments* OMAP3530 procesoru koji će biti detaljnije opisan u slijedećem poglavlju. OMAP3530 koristi POP (eng. *Package-on-Package*) memoriju. Memorija je MCP (eng. *Multi Chip Package*) koja sadrži i mobilni DDR SDRAM (eng. *Double Data Rate Synchronous Dynamic Random Access Memory*) i NAND memoriju. Na OMAP3530 se nalazi 256 MB NAND i 256 MB MDDR SDRAM memorije te je na slici 1. prikazan princip POP memorije.



Slika 1. POP memorija

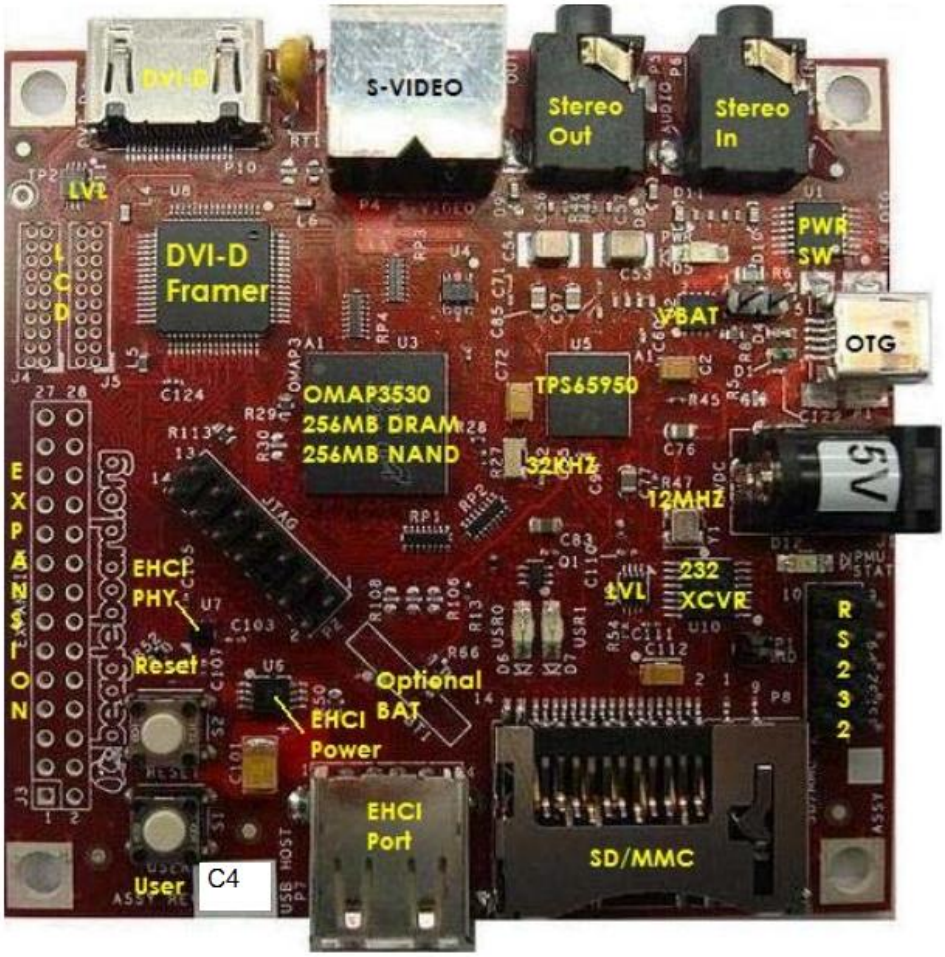
Glavne karakteristike *Beagleborada* su:

- OMAP3530 procesor
- 256 MB MDDR SDRAM (166 MHz) i 256 MB NAND memorije
- utor za SD/MMC kartice
- USB priključak (s podrškom za USB čvorište)
- video izlaz S-video i digitalni DIV-D
- stereo izlaz i stereo ulaz
- JTAG i RS232 konektori
- ...

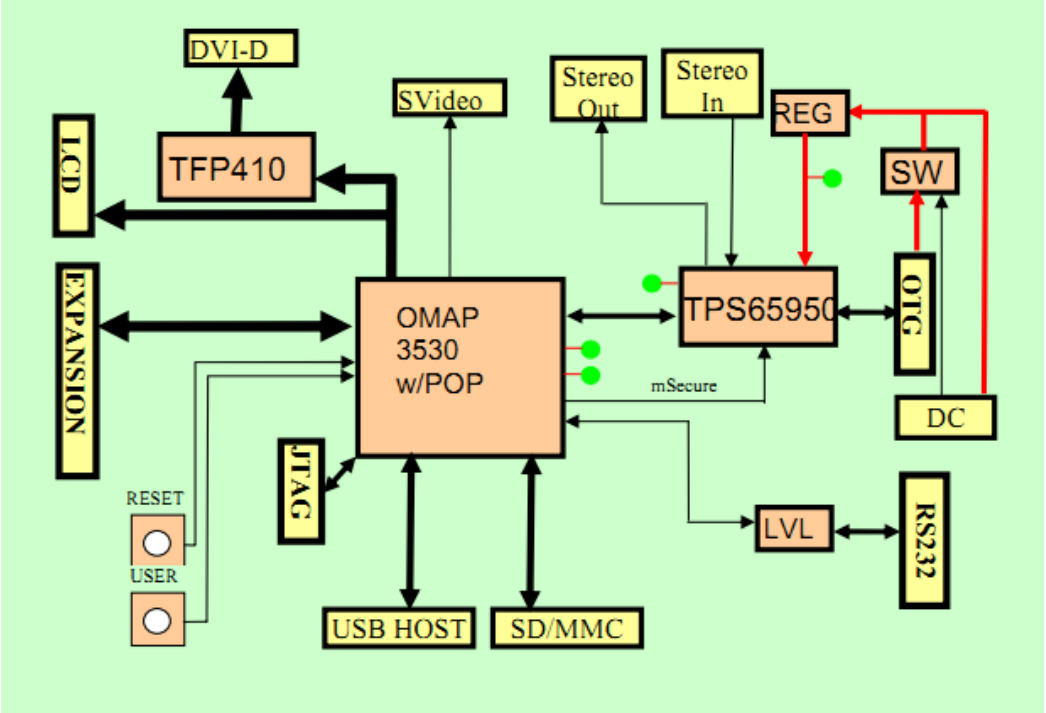
Nabrojene karakteristike se neće detaljno objašnjavati već samo one bitne za izradu ovog diplomskog rada. Na slici 2. je prikazan izgled Beagleboarda s oznakama pojedinih konektora i čipova. Bitni sustavi za izradu ovog diplomskog

[1] Tehnička dokumentacija Beagleboarda [1]

rada su OMAP3530, TPS65950 i stereo izlaz. OMAP zato što je je to glavni procesor i na njemu će biti pokretani programi, TPS čip jer se u njemu nalazi audio kodek koji pretvara digitalni u analogni signal koji se prosljeđuje na stereo izlaz na koji se spoje zvučnici. Ti sustavi će biti dodatno pojašnjeni u narednim poglavljima. Slika 3. prikazuje blok dijagram *Beagleboarda* gdje se vidi međusobna povezanost cijelog sustava.



Slika 2. Beagleboard



Slika 3. Blok dijagram Beagleboarda

3. OMAP3530

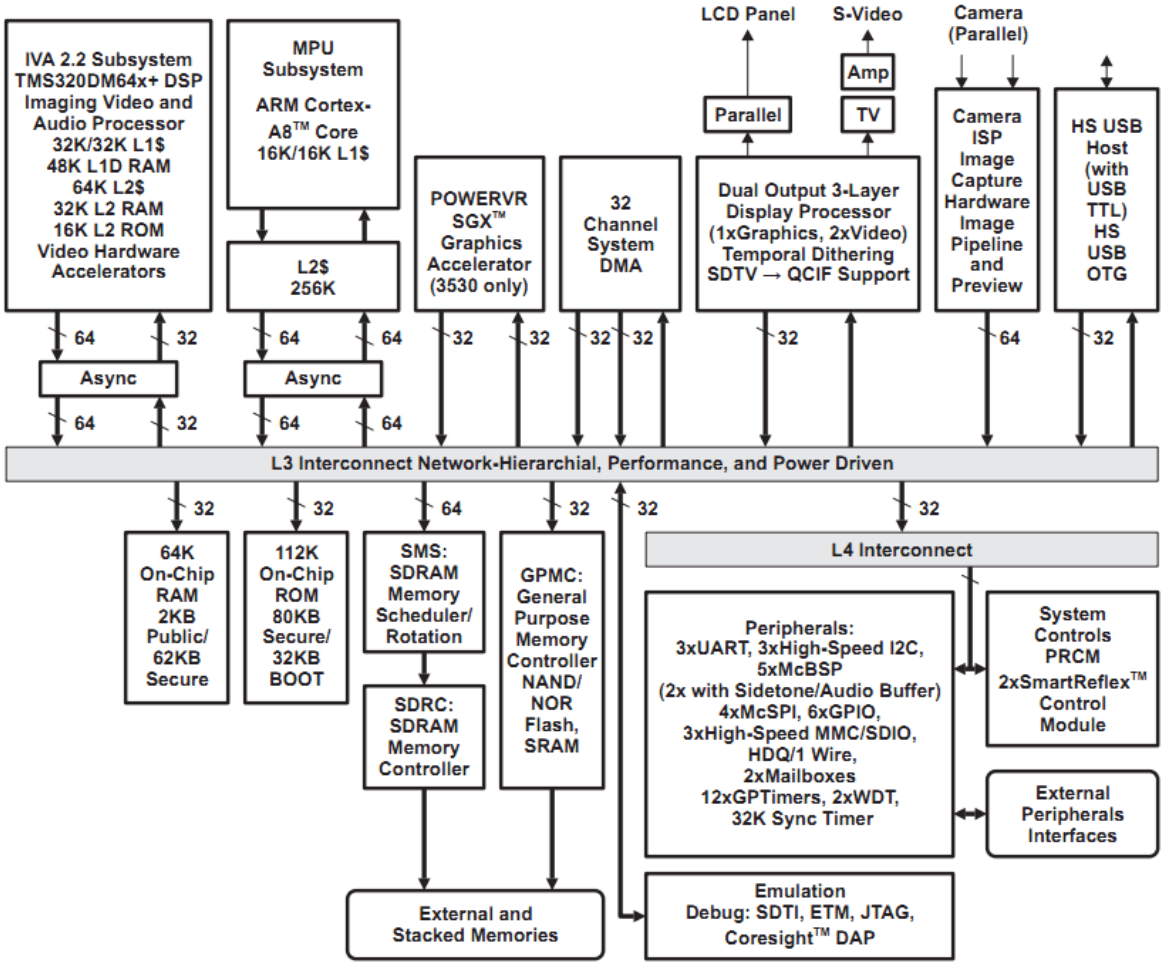
Kao što je već rečeno srce *Beagleboarda* je OMAP3530[2] procesor. Procesor nije x86 arhitekture već je unutar OMAP arhitekture integrirano više podsistema. Integrirani podsustavi su:

- MPU (eng. *Microprocessor unit*)
- IVA2.2 (eng. *Image and Video Accelerator*)
- memorija na čipu
- sučelje za vanjsku memoriju
- DMA (eng. *Direct Memory Access*) kontroler
- *PowerVR* SGX grafički procesor
- multimedijalni akcelerator
- iscrpno upravljanje napajanjem
- vanjske jedinice

Na slici 4. je prikazan blok dijagram OMAP3530 procesora gdje su vidljivi navedeni podsistemi i njihova međusobna povezanost. Može se vidjeti da je glavna jezgra MPU podsustava ARM cortex A8 procesor, a IVA2.2 podsustava *Texas Instruments* TMS320DMC64x+ VLIW (eng. *Very Long Instruction Word*) DSP (eng. *Digital Signal Processing*) procesor. Kratki opis oba procesora se nalazi u narednom tekstu. Komunikacija između gotovo svih modula odvija se preko L3 sabirnice osim vanjskih jedinica koja još ide preko L4 sabirinice, a komunikacija unutar procesora (ARM i DSP) se odvija preko L1 i L2 što znači da postoje četiri razine komunikacije unutar OMAP procesora.

U nastavku su ukratko objašnjeni ARM i DSP procesor.

[2] Tehnička dokumentacija procesora OMAP3530 [3]



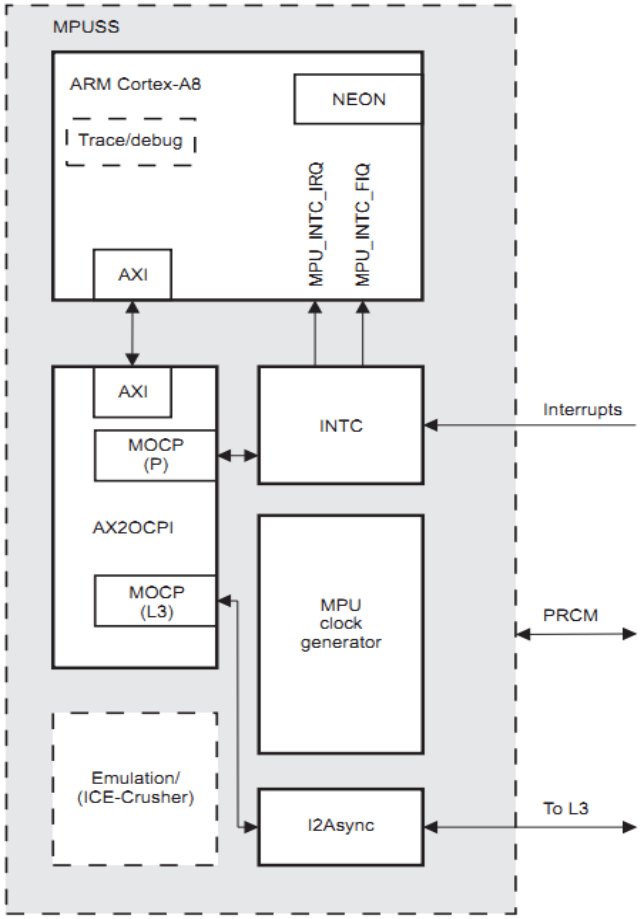
Slika 4. Blok dijagram OMAP3530 procesora

3.1. MPU - ARM Cortex A8

MPU podsustav upravlja prometom između ARM jezge, L3 sabirnice i prekidnog kontrolera. Na slici 5. je prikazan blok dijagram MPU podsustava. Unutar MPU podsustava su integrirani:

- ARM cortex A8 jezgra
- prekidni kontroler sa 96 sinkronih prekidnih linija osjetljivih na razinu
- AXI2OPC most između ARM AXI (eng. *Advanced Extensible Interface*) sabirnice, L3 glavne OCP (eng. *Open Core Protocol*) sabirnice i glavne OPC sabirnice prekidnog kontrolera

- MPU generator takta – generira takt, načine napajanja te neaktivne i aktivne potvrdne signale
- I2Async most – sučelje asinkronog mosta omogućuje sučelje za asinkroni OCP prema OCP
- module za ispravljanje, traženje grešaka i emulator funkcija



Slika 5. MPU blok dijagram

ARM Cortex A8

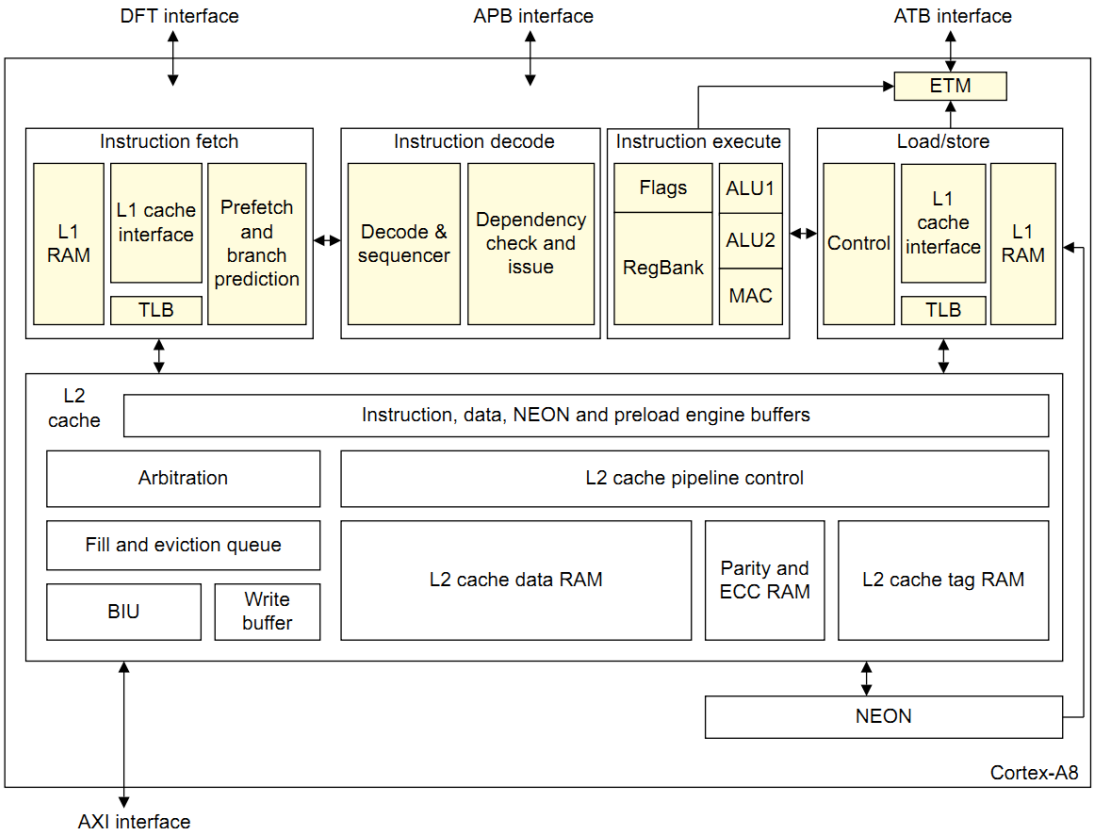
ARM Cortex A8[3] je RISC (eng. *Reduced Instruction Set Computer*) arhitekture s tipičnim RISC obilježjima: velik skup registara, dohvati/spremi arhitektura, operacije nad podacima u registrima (ne direktno u memoriji), jednostavno adresiranje. ARM procesor daje dobar odnos između visokih performansi, malog kôda, male potrošnje i malog fizičkog zauzeća.

[3] Tehnička dokumentacija ARM Cortex A8 procesora [4]

ARM Cortex A8 na *Beaglebordu* (revizija C4) radi na 720 MHz i ima 16 KB instrukcijske i podatkovne L1 priručne memorije, 256 KB L2 priručne memorije. Procesor ima ukupno 40 32 bitna registara, od toga je 32 registra opće namjene i sedam statusnih registra. Svi registri nisu uvijek dostupni već ovisi u kojem načinu rada se procesor nalazi. 16 registara i jedan ili dva statusna registra su uvijek dostupna. Registri od R0 do R13 su opće namjene, a registar R14 se koristi kao povezni registar, dok se R15 koristi kao programsko brojilo.

ARM Cortex A8 podržava standardni ARM skup v7-A instrukcija te su uz standardne ARM instrukcije uključeni *Thumb-2*, *Jazell* i *Neon* tehnologija.

Slika 6. pokazuje blok dijagram ARM Cortex A8 procesora gdje su vidljive glavne komponente procesora kao što su dohvat instrukcije (eng. *Instruction fatch*), dekodiranje instrukcije (eng. *Instruction decode*), izvršavanje instrukcije (eng, *Instruction execute*), dohvaćanje/spremanje (eng. *Load/Store*), L2 priručna memorija, *Neon* i ETM (eng. *Embedded Trace Macrocell*).



Slika 6. ARM Cortex A8 blok dijagram

3.2. IVA2.2 - TI TMS320C64x+ DSP

IVA2.2 podsustav isto kao i MPU unutar svojeg podsustava integrira dosta drugih modula:

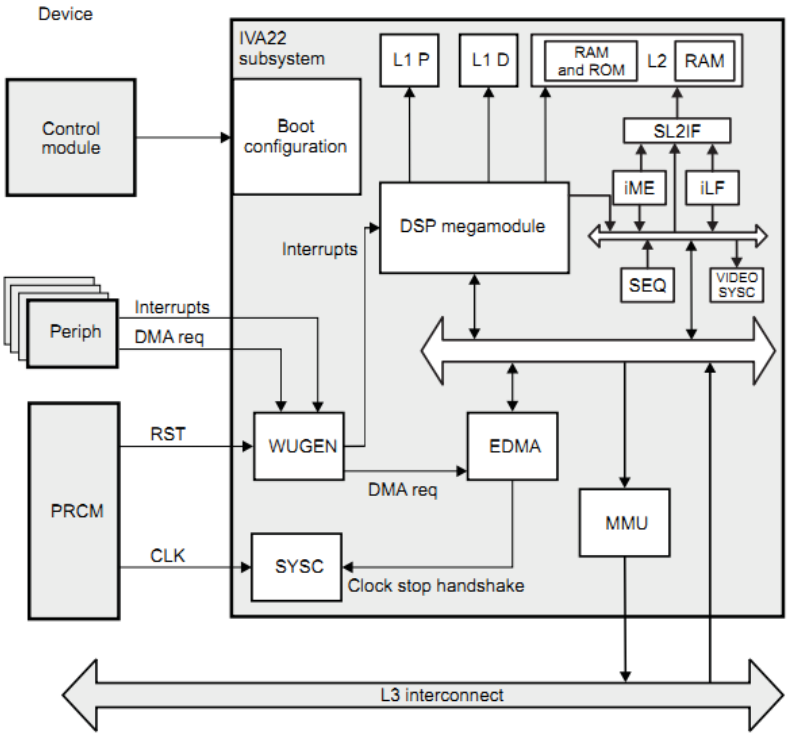
- TI DSP visokih performansi TMS320DMC64x+ integriran u megamodul s uključenom L1 i L2 priručnom memorijom i memorijskim kontrolerom
- L1 RAM i L2 RAM i ROM
- sklopovski ubrzivač videa
- pridruženi EDMA (eng. *Enhance Data Memory Access*) mehanizam za prihvaćanje/slanje podataka iz/prema memoriji i vanjskim periferijama
- MMU (eng. *Memory Managment Unit*) za pristup L3 sabirnici
- te pridruženi moduli SYSC i WUGEN zaduženi za upravljanje napajanjem, generiranjem takta i poveznicom prema PRCM (eng. *Power, Reset and Clock Manager*) modulu

IVA2.2 podsustav je prikazan na slici 7. a DSP megamodul je prikazan na slici 8.

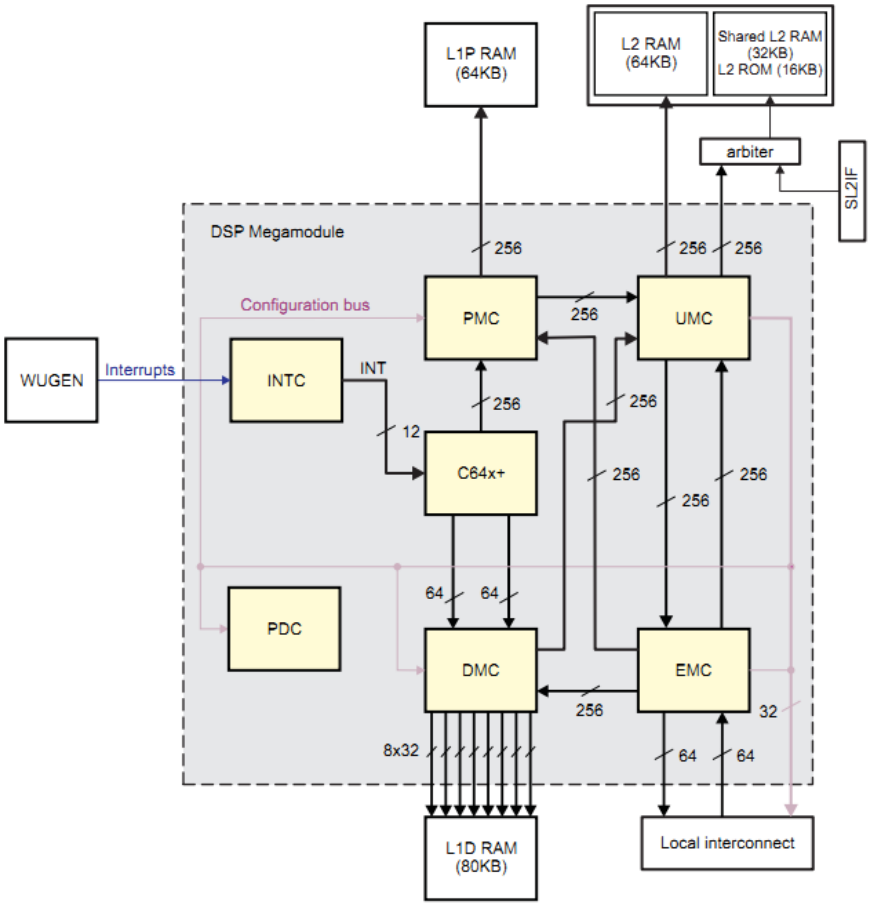
TI DSP C64x+

DSP[4] jezgra unutar *Beagleboarda* radi na 520 MHz. To je 32 bitni cjelobrojni medijski procesor te, kao što je već navedeno, koristi VLIW arhitekturu. Procesor ima mogućnost izvođenja osam instrukcija po ciklusu tj. ima osam funkcijskih jedinica (.L1, .L2, .S1, .S2, .M1, .M2, .D1 i .D2). Svaki par funkcijskih jedinica ima svoj skup instrukcija koji se može izvoditi na određenoj funkcijskoj jedinici. Tih osam funkcijskih jedinica je podijeljeno po četiri jedinice na svaki skup registra opće namjene A i B. Oba ta skupa registra imaju po 32 32 bitna registra i međusobno su povezana vezama 1X i 2X. Također ima po dva puta LD1 i LD2 za dohvat iz memorije, ST1 i ST2 za spremanje u memoriju i dva adresna puta DA1 i DA2.

[4] Tehnička dokumentacija za C64x+ DSP procesor

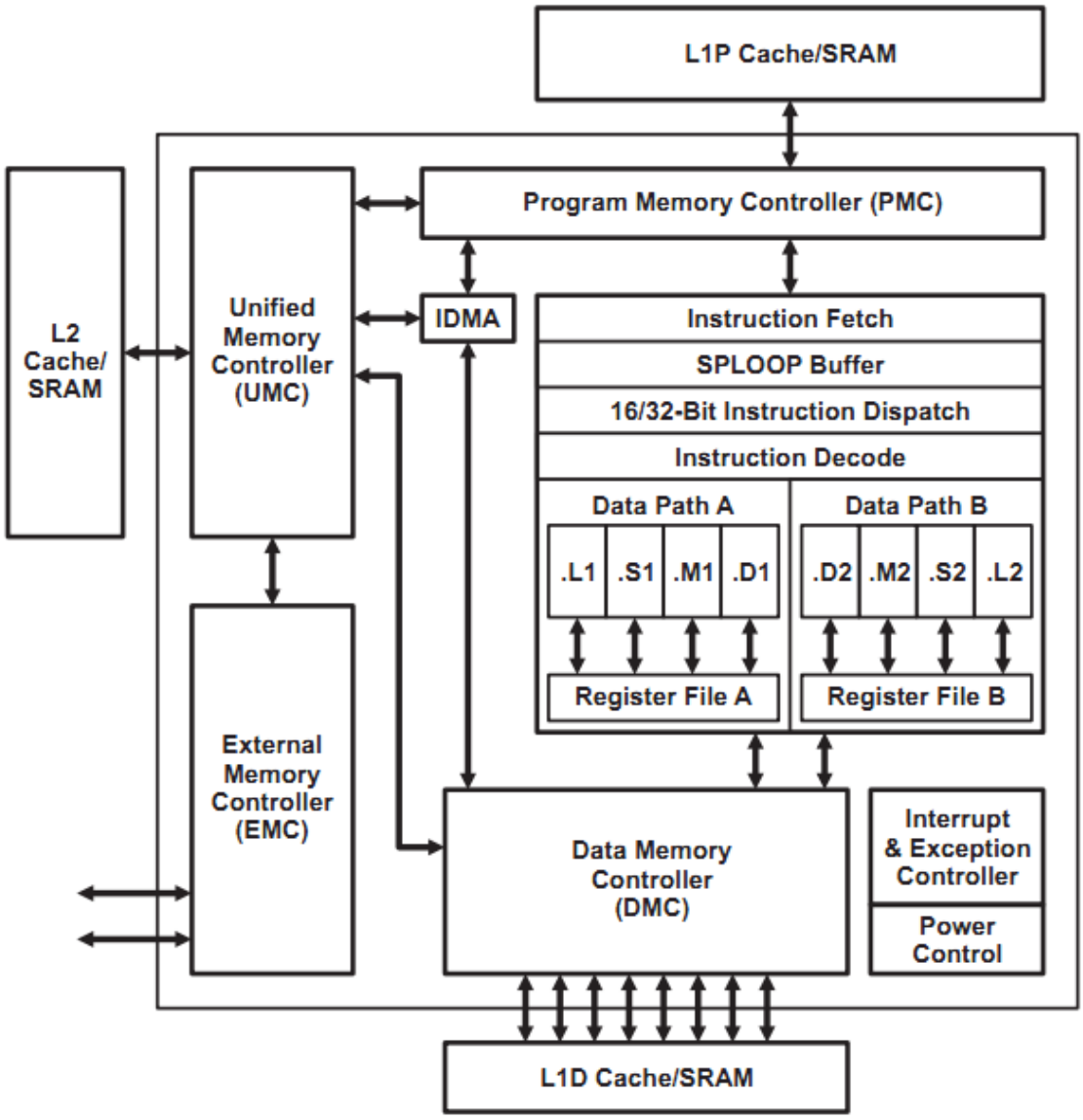


Slika 7. IVA2.2 blok dijagram



Slika 8. DSP megamodul blok dijagram

Na slici 9. je prikazan blok dijagram C64x+ DSP procesora gdje su vidljive i ostale komponente procesora kao što su priručna memorija, kontroler prekida i iznimaka, podatkovni i programski memorijski kontroler...



Slika 9. DSP C64x+ procesora blok dijagram

4. TI TPS65950 – Audio Codec

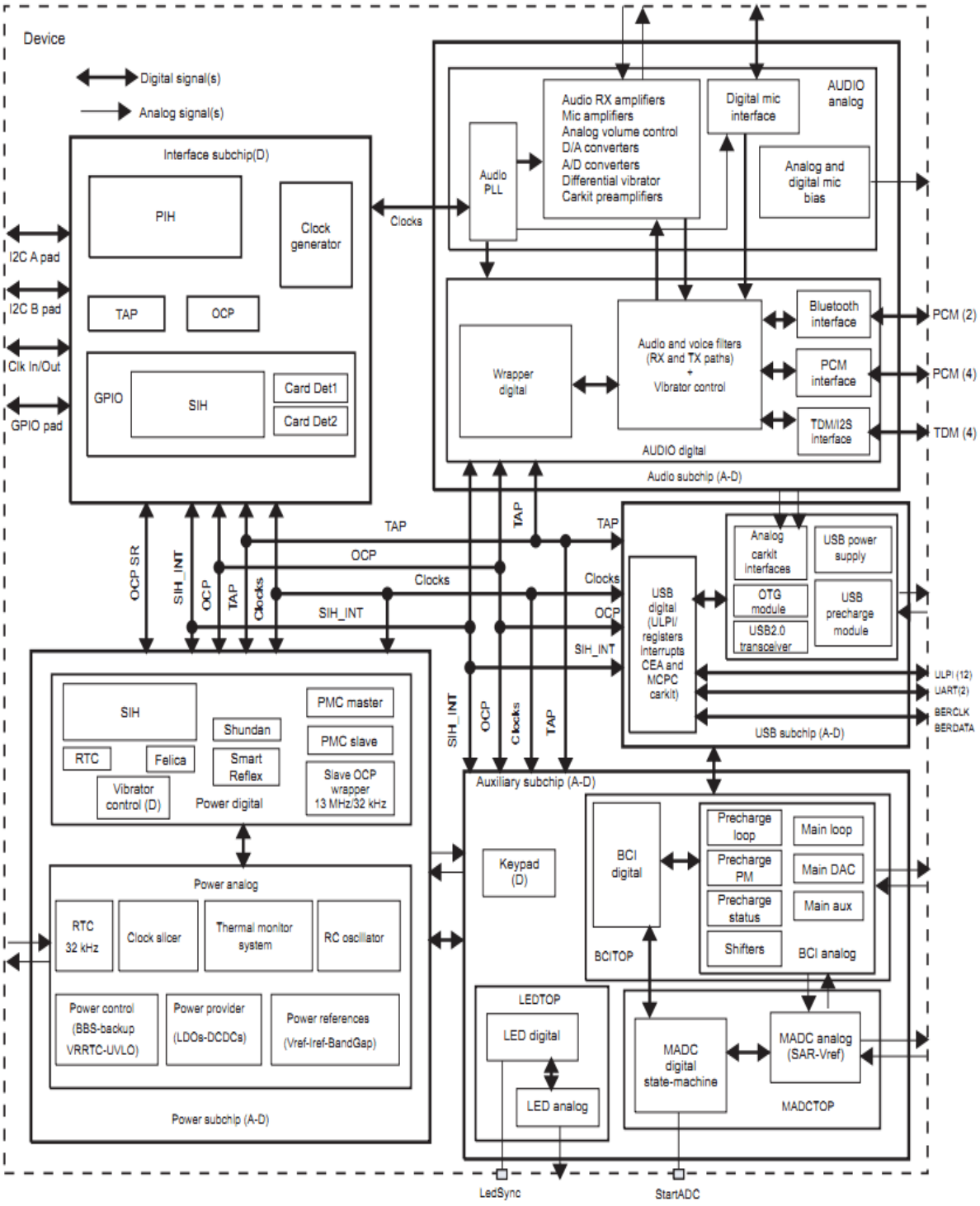
Nakon opisanih ARM i DSP jezgri slijedeći važan sklop je TI TPS65950[5]. On je važan iz razloga što se u njemu nalazi audio kodek koji je povezan sa stereo izlazom/ulazom. Da bi procesori izvršavali program za direktnu digitalnu sintezu dobiveni digitalni podatak bi trebali pretvarati u analogni te onda prosljeđivali na stereo izlaz.

TPS65950 je visoko integrirani upravljač napajanja i audio koder/dekoder koji podržava napajanje i vanjske jedinice OMAP procesora. Unutar sklopa su sadržani upravljač napajanja, audio kodek, HSUSB (eng. *High Speed Universal Serial Bus*), odašiljač, AC/USB punjač, upravljač LED diodama, analogno-digitalni pretvarač, RTC (eng. *Real-Time Clock*) i ugrađena kontrola napajanjem. Na slici 10. je prikazan blok dijagram sklopa TPS65950. Naravno ni ovdje se ne razmatra cijeli sklop već samo audio kodek s obzirom da je isti bitan za izradu ovog diplomskog rada.

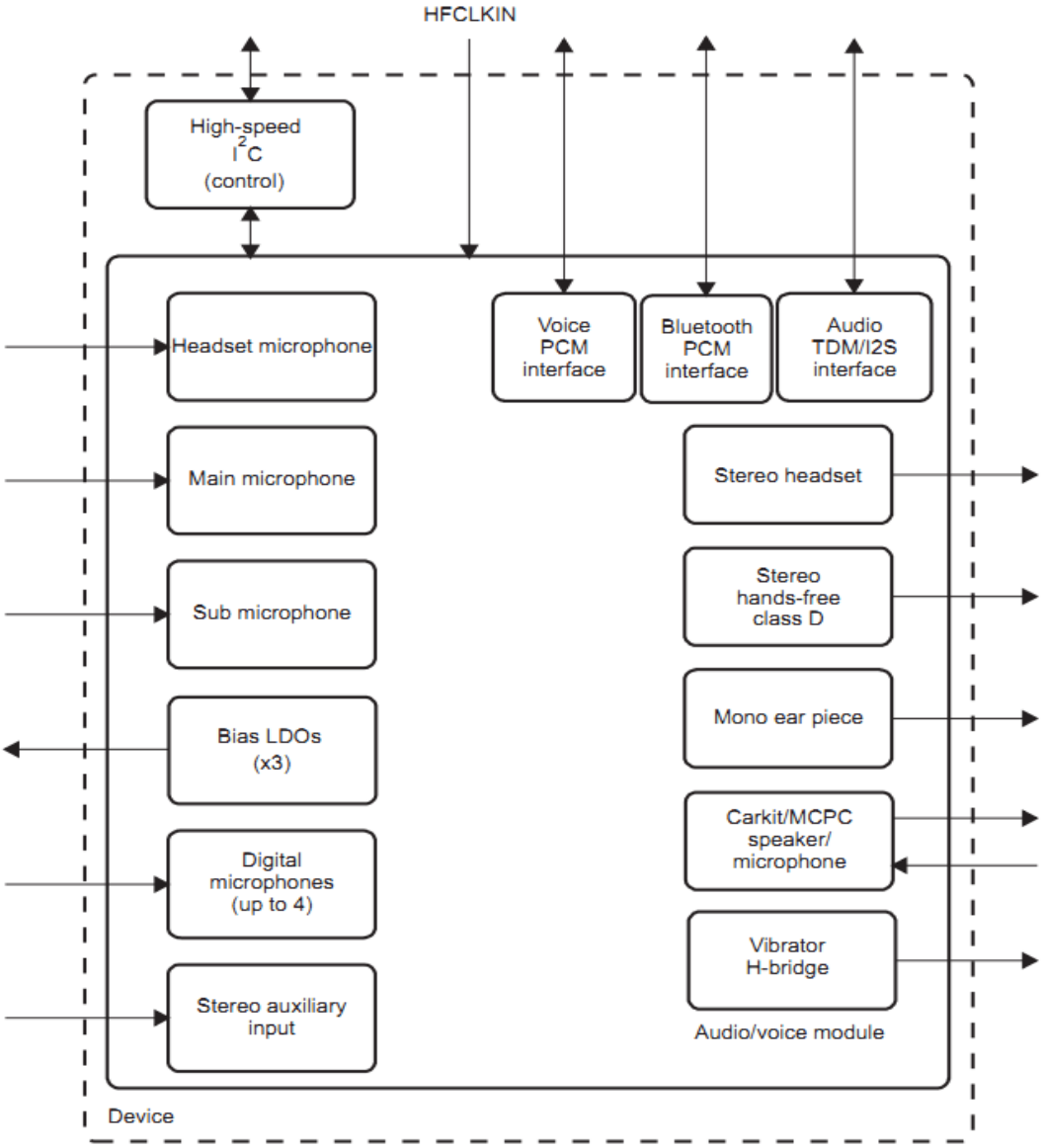
Audio kodek koji je u sklopu uključuje pet digitalno-analognih pretvarača i dva analogno-digitalna pretvarača za posluživanje više kanalnog glasovnog i stereo kanala (16 bitni) koji podržava sve standardne audio brzine uzorkovanja (od 8 kHz do 48 kHz) kroz nekoliko I2S/TDM (eng. *Inter-integrated circuit Sound/Time Division Multiplexing*) formata sučelja. Put audio izlaza uključuje pojačala za stereo naglavni set, dva integrirana pojačala klase D za pružanje diferencijalnog stereo izlaza, pretpojačala za izlazne linije i pojačala za slušalice. Put audio ulaza uključuje tri diferencijalna ulaza za mikrofon, stereo ulaz i sučelje za digitalni mikrofon. Na slici 11. je prikazan blok dijagram samo od audio modula.

Audio kodek tj. njegovi registri se podešavaju preko I2C (eng. *Inter-Integrated circuit*) sabirnice, a komunikacija između OMAP procesora i audio kodeka se odvija preko McBSP (eng. *Multi-Channel Buffered Serial Port*) sabirnice koristeći I2S format što je prikazano na slici 12. McBSP kada se koristi za prijenos audio podataka u međuspremnik se može spremiti do ukupno 5 kB podataka.

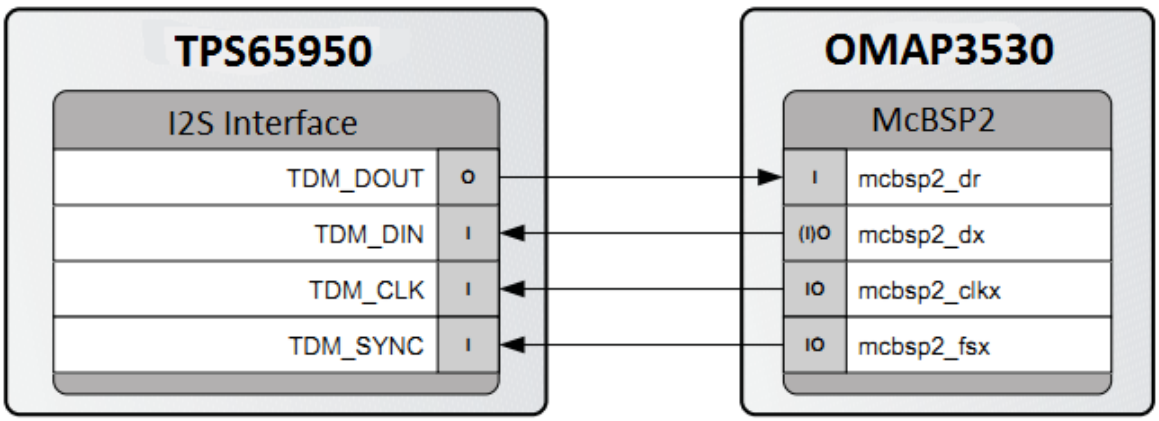
[5] Dokumentacija o TPS65950 sklopu [9] i [10]



Slika 10. TPS65950 blok dijagram



Slika 11. Blok dijagram audio modula



Slika 12. Veza između TPS65950 i OMAP3530

5. Direktna digitalna sinteza sinusa visoke točnosti

U ovom poglavlju biti će opisan kratki teoretski uvod o direktnoj digitalnoj sintezi[6]. Isto tako u nastavku je opisan programski model direktne digitalne sinteze sinusa na osnovu kojega su izrađena oba kôda i za ARM i DSP procesor.

5.1. Kratki teoretski uvod

$y_s(\phi) = \sin(\phi)$ je idealna sinusna funkcija za zadanu fazu ϕ te može biti aproksimirana po dijelovima polinomnom funkcijom $y_r(\phi)$ sa N uniformnih dijelova unutar sva četiri kvadranta od $y_s(\phi)$. Za kubični polinom segmenti između uzoraka faze $n \cdot 2\pi/N$ i $(n+1) \cdot 2\pi/N$ se evaluiraju prema Hornerovom pravilu:

$$w_n(\Delta x) = \big((d[n] \cdot \Delta x + c[n]) \cdot \Delta x + b[n]\big) \cdot \Delta x + a[n] \tag{1}$$

Argument Δx kubičnog polinoma je jednak nuli u desnoj točki segmenta, a jedinici u lijevoj točki $0 \leq \Delta x \leq 1$. Kontinuiranost faze aproksimirane sinusne funkcije se dobiva spajanjem pojedinačnih kubičnih segmenata:

$$y_r(\phi) = \left\{ w_n(\Delta x) \middle| \Delta x = N\frac{\phi}{2\pi} - n, \phi \in \left[n\frac{2\pi}{N}, (n+1)\frac{2\pi}{N} \right), \forall n \in \{0, N-1\} \right\} \tag{2}$$

Koeficijenti će biti zapisani u vektorkoj notaciji:

$$cub[n] = [d[n]\ c[n]\ b[n]\ a[n]]^T, \quad \phi\epsilon[0, 2\pi] \tag{3}$$

Koeficijenti $d[n], c[n], b[n], a[n]$ su odabrani tako da se postigne minimalna maksimalna pogreška aproksimacije:

$$e(\phi) = y_s(\phi) - y_r(\phi), \quad \phi \in [0, 2\pi] \tag{4}$$

[6] Za više detalja pogledati literaturu [11]

Postizanje maksimalne glatkoće po dijelovima polinoma se ovdje postiže korištenjem B-spline interpolacije. Koeficijenti $cub[n]$ se mogu izračunati filtriranjem diskretnog sinusnog signala $y_d[n] = y_s(n \cdot 2\pi/N)$ sa nekauzalnim sustavom opisanim transfer matricom:

$$T(z) = \frac{1}{z^{-1}+4+z}\begin{bmatrix} -z^{-1}+3-3z+z^2 \\ 3z^{-1}-6+3z \\ -3z^{-1}+3z \\ z^{-1}+4+z \end{bmatrix} \tag{5}$$

$$Cub(z) = [D(z)\ C(z)\ B(z)\ A(z)]^T = T(z) \cdot Y_d(z) \tag{6}$$

gdje su $Cub(z)$ i $Y_d(z)$ z-transformacije od $cub[n]$ i $y_d[n]$. Obzirom na linearnost od $T(z)$ koeficijenti $cub[n]$ su također diskretne sinusoide čije su magnitude i faza određene frekvencijskim odzivom $T(z)$ na frekvenciji $\omega_0 = 2\pi/N$ za $z = e^{j\omega_0}$. To nas dovodi do jednostavnih formulacija spektralnih domena od koeficijenata:

$$cub[n] = \begin{bmatrix} d[n] \\ c[n] \\ b[n] \\ a[n] \end{bmatrix} = \begin{bmatrix} D_m\cos(n \cdot \omega_0 + \theta_d) \\ C_m\cos(n \cdot \omega_0 + \theta_c) \\ B_m\cos(n \cdot \omega_0 + \theta_b) \\ A_m\cos(n \cdot \omega_0 + \theta_a) \end{bmatrix} \tag{7}$$

$$\left|T(e^{j\omega_0})\right| = \begin{bmatrix} D_m \\ C_m \\ B_m \\ A_m \end{bmatrix} = \frac{1}{2+cos\omega_0}\begin{bmatrix} 3\sin\left(\frac{\omega_0}{2}\right) - \sin(3\frac{\omega_0}{2}) \\ 3(1-cos\omega_0) \\ 3sin\omega_0 \\ 2+cos\omega_0 \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} \theta_d \\ \theta_c \\ \theta_b \\ \theta_a \end{bmatrix} = \angle T\left(e^{j\omega_0}\right) - \frac{\pi}{2} = \begin{bmatrix} \omega_0/2-\pi \\ \pi/2 \\ 0 \\ -\pi/2 \end{bmatrix} \tag{9}$$

Opisani kubični B-spline interpolator konstruira kontinuirani fazni signal $y_s(\phi)$ iz uzoraka $y_d(\phi)$ oponašajući idealni $sinc$ interpolator.

Rekonstruirani signal može se dobiti iz:

$$y_r(\phi) = G \cdot \sum_{i=-\infty}^{\infty} \frac{\sin(\phi(1-iN))}{(1-iN)^4} \tag{10}$$

gdje je G odziv spektra kardinalnog spline-a na normaliziranoj frekvenciji ω_0:

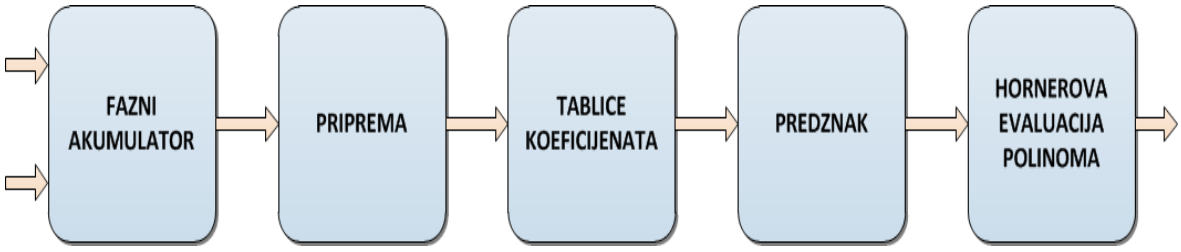
$$G = H\left(j\frac{\omega_0}{T}\right) = \frac{\sin(\omega_0/2)^4}{(\omega_0/2)^4} \cdot \frac{3}{2+cos\omega_0} \text{ za } \omega_0 = \frac{2\pi}{N} \quad (11)$$

Rekonstruirani signal $y_r(\phi)$ sadrži skaliranu verziju željenog signala $G \cdot y_s(\phi)$ pokvarenog za beskonačan broj neželjenih harmonika čija magnituda nestaje približno s faktorom $(jN)^{-4}$ (analitičkim proračunom se dobije da se pogreška može aprokimirati s približno $1.83N^{-4}$). Dobiveni signal $y_r(\phi)$ ima kontinuirane derivacije prvog i drugog reda na svim spojnim točkama.

5.2. Programski model

Sama izvedba direktne digitalne sinteze je više prilagođena za izvedbu u FPGA (eng. *Field Programmable Gate Array*) *Spartan3*. Upravo zbog te prilagodbe za sklopovlje *Spartana3* se koristi manji broj bitova što se je moralo ograničiti maskama ili posmacima u C jeziku dok npr. u VHDL-u to nebi bilo potrebno. Detaljni pregled C kôda za određeni procesor je dan u poglavljima 8. i 9. dok je ovdje opisan općeniti koncept.

Blok dijagram slijednog programskog modela direktne digitalne sinteze sinusa je prikazan na slici 13. gdje je cijeli program podijeljen u pet glavnih faza. Pojedine faze modela se detaljinije objašnjavaju u nastavku teksta.



Slika 13. Blok dijagram programskog modela

Fazni akumulator je prvi u modelu te mu se za ulaz zadaje početna faza i korak, a kao izlaz daje zbroj „stare faze" i koraka tj. faza = faza + korak.

Kada bi izlaz faznog akumulatora grafički prikazali on bi izgledao kao pilasti napon. Veličina faznog akumulatora je 28 bita što znači da se u C-u mora koristiti `int` tip, ali on je standardno 32 bita pa ga se mora dodatno ograničiti npr. maskom koja ima 28 jedinica i unarnim operatorom `and`.

Priprema je drugi model u nizu koji od 28 bitne faze, što je ujedno i ulaz u ovaj model, izvlači pojedine bitove i skupine bitova. Dva najviša bita služe za određivanje predznaka koeficijenata, slijedećih osam bita je adresa s koje se očitavaju koeficijenti, a najnižih 18 bita je pozicija unutar uzoraka koju smo označili s Δx. Δx je još potrebno pretvoriti u cjelobrojni prikaz komplementiranjem najvišeg bita.

Tablice koeficijenata je slijedeći model koji predstavlja čitanje koeficijenata A, B, C i D koji su pohranjeni bez predznaka u memoriju. Koficijenti A, B, C, D su slijedećih širina: 31, 23, 14 i 3 bita. Širine su smanjene tako da troše što manje memorije, a da se pritom ne povećava greška sveukupnog modela. Dodatnog utjecaja može imati kod sustava koji imaju manje memorije na raspolaganju.

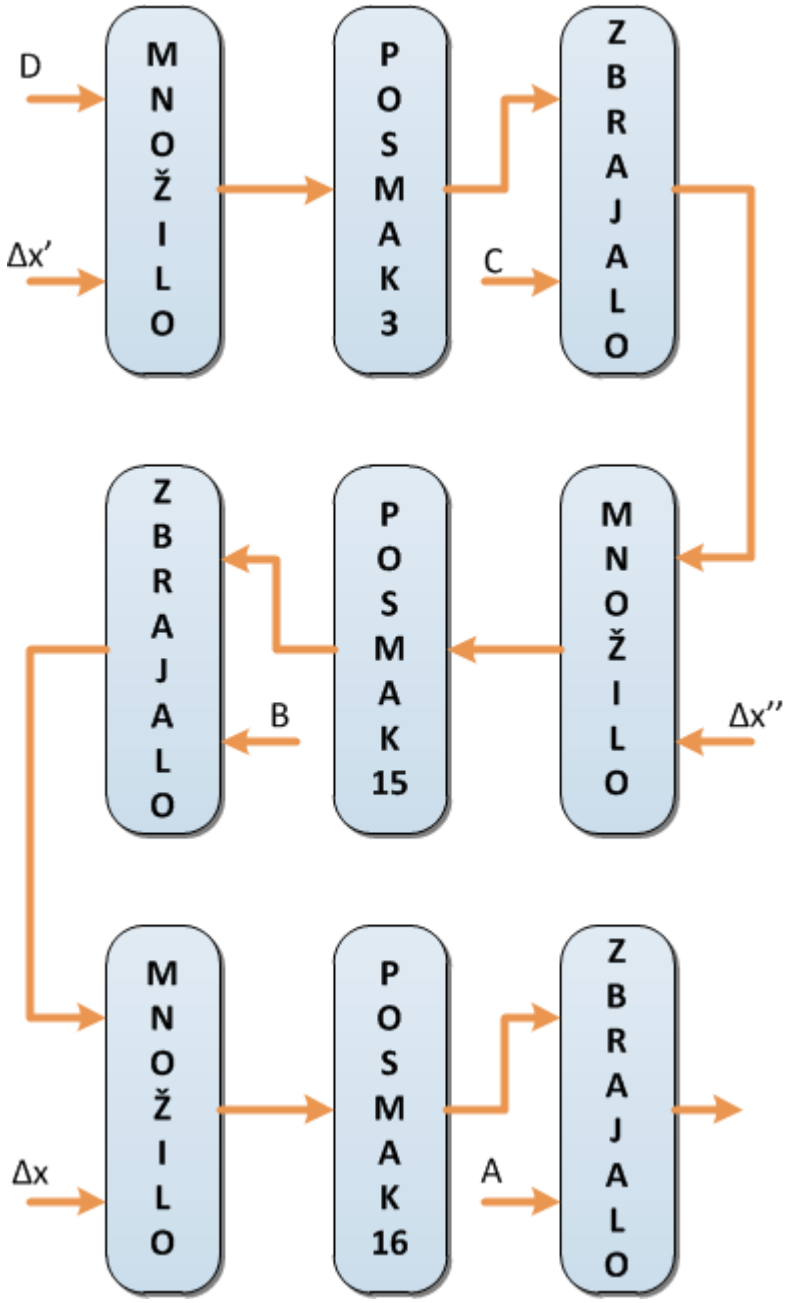
Predznak pretvara pročitane koeficijente iz prethodnog modela na osnovu dva najviša bita faznog akumulatora, koje smo odvojili u drugom modulu, u brojeve s predznakom. Tablica 1. predstavlja odnose najviših bitova i predznaka pojedinih koeficijenata.

Tablica 1. Odnosi koeficijeneta

	MSB - NSB			
	00	**01**	**10**	**11**
A	pozitivan	pozitivan	negativan	negativan
B	pozitivan	negativan	negativan	pozitivan
C	negativan	negativan	pozitivan	pozitivan
D	negativan	pozitivan	pozitivan	negativan

Hornerova evaluacija polinoma je zadnji modul koji računa polinom prema formuli (1). Detaljna slika *Hornerove* evaluacije polinoma je prikazana na slici 14. Kao što se na slici vidi, da bi napravili evaluaciju polinoma potrebna su tri množenja, posmaka i zbrajanja. Oznake Δx, $\Delta x'$ i $\Delta x''$ predstavljaju „isti" broj samo je razlika u količini korištenih najznačajnijih bitova koja je redom svih 18 bita, najviših 17 bita i najviših 6 bita. Prvo je množenje koeficijenta D i $\Delta x'$, frakcionalno zapisano 1.3x1.5, što daje rezultat množenja 1.8 koji se zaokružuje na 1.4. Zaokruživanje se radi tako da produkt množenja posmaknemo za tri mjesta što je prikazano drugim blokom na slici 14. te najniži bit dodamo na mjesto više. To zaokruživanje se radi zajedno sa zbrajanjem s koeficijentom C 1.14, stoga je dobiveni rezultat frakcionalno 1.14. Rezultat se prosljeđuje do drugog množila pa se gore opisani postupak ponavlja s razlikama u širinama korištenih bitova i posmacima. Znači imamo sumu s prvog zbrajala 1.14 i $\Delta x''$ koji je 1.16, posamk za 15 bita te zaokruživanje i zbrajanje i koeficijentom B 1.23, pa je izlaz iz drugog zbrajala 1.23 koji se prosljeđuje na treće množilo. Također i ovdje ponavljamo postupak. Suma s drugog zbrajala 1.23 i Δx 1.17 (cijeli Δx), posamk za 16 bita te zaokruživanje i zbrajanje s koeficijentom A 1.31 nakon čega dobivamo i konačan izlaz tj. prvi rezultat.

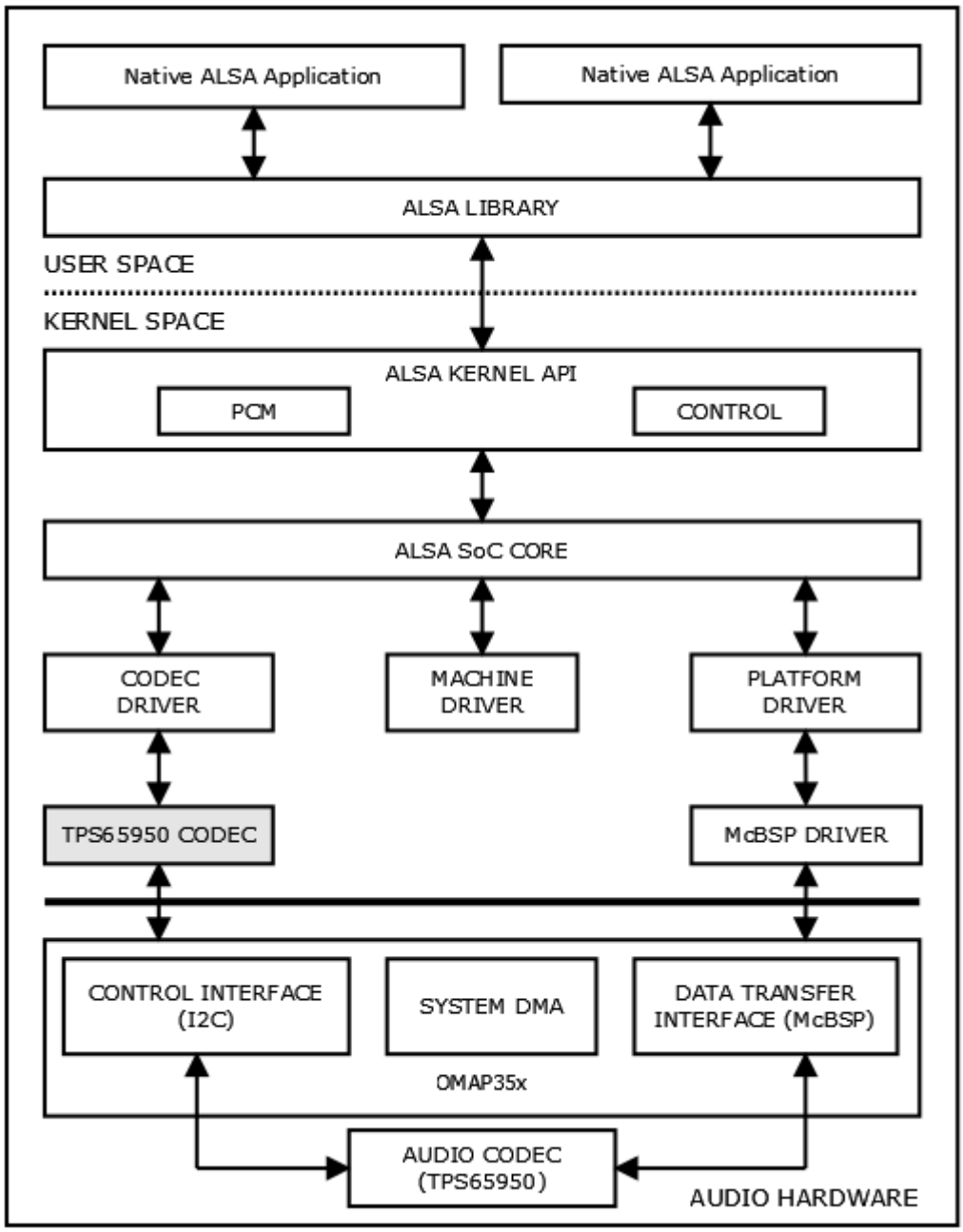
Može se primjetiti da kod trećeg množila gdje množimo 1.23x1.17 je rezultat 1.40 što ne stane u 32 bit npr. `int` tip podatka. Stoga je potrebno ili koristiti tip podatka koji je širi od 32 bita ili napraviti prošireno množenje u dva registra širine 32 bita. U ovome radu je korišten širi tip podatka te time dobivamo egzaktan rezultat kao i u *Matlabu*, pogledati poglavlje 8.



Slika 14. Hornerova evaluacija blok dijagram

6. ALSA (eng. *Advanced Linux Sound Architecture*)

Kao što je već rečeno audio sustav se nalazi unutar integriranog sklopa TPS65950. Za upravljanje i komunikaciju između sklopovlja i jezgre se koristi ALSA SoC (eng. *System on Chip*), a za komunikaciju između jezgre i korisničkih aplikacija ALSA biblioteka što je prikazano na slici 15. ALSA SoC je prilagođena za ugradbene sustave.



Slika 15. ALSA na OMAP sustavu

Svojstva ALSA SoC sloja su:

- Neovisnost kôda – mogućnost korištenja i na drugim platformama.
- Lagano podešenje sučelja između audio sustava i procesora putem I2S/PCM (eng. *Pulse Code Modulation*).
- Dinamičko upravljanje napajanja audio sustava (DAPM) – automatski postavlja audio sklop u minimalno korištenje napajanja
- smanjenje pucketanja i klikanja

Da bi navedena svojstva bila ostvarena ALSA SoC praktički dijeli ugradbene audio sustve u tri komponente:

- Kodek upravljač – neovisan je o platformi i sadrži audio kontrole, audio sučelje, DAPM definicije i ulazno/izlazne funkcije
- Platform upravljač – sadrži audio DMA i upravljač audio sučelja (npr. I2S, PCM) za određenu platformu
- Sklopovski upravljač – rukovodi specifičnim sklopovskim kontrolama i audio događajima

6.1. ALSA biblioteka

Nakon kratkog uvoda o ALSA-i u ovome potpoglavlju će biti objašnjene pojedine funkcije potrebne da bi se audio sklop inicijalizirao kako bi mogao na izlazu reproducirati dobivene uzorke od DDSS-a.

Koristiti će se PCM sučelje ALSA-e, ali iako se PCM odnosi na eng. *Pulse Code Modulation* u ovom slučaju se poima za osnovno digitalno audio procesuiranje uzoraka generiranih u kontinuiranim vremenskim periodima.

U nastavku će se spominjati uzorci, što podrazumijeva jednu digitalnu vrijednost dok okvir podrazumijeva skup od više uzoraka. ALSA koristi kružni međuspremnik za spremanje podataka koji se šalju na izlaz audio sklopa.

```
int snd_pcm_open(snd_pcm_t **   pcmp,
```

```
const char *        name,
snd_pcm_stream_t    stream,
int                 mode  );
```

Funkcija naravno otvara PCM i vraća nulu ako je uspješna, inače negativnu vrijednost. U `pcmp` se vraća upravljač PCM-a, `name` je ASCII identifikator PCM upravljača, a korišteni su `plughw:0,0`, tada ne treba voditi puno brige oko sklopovlja i `hw:0,0` ako se koristi treba znati točno što sklopovlje podržava. Prvi broj u nazivu predstavlja broj zvučne kartice, a drugi broj uređaja. `stream` predstavlja željeni tok podataka, slušanje ili snimanje i `mode` koje predstavlja mod otvaranja koji može biti postavljen u nula (standardni mod), ne blokirajući mod (`SND_PCM_NONBLOCK`) što znači da će se nakon pisanja ili čitanja odmah vratiti PCM upravljaču i asinkroni mod (`SND_PCM_ASYNC`) koji vraća signal kada je jedan period u potpunosti završen na zvučnoj kartici.

```
int snd_pcm_close ( snd_pcm_t * pcm);
```

Funkcija zatvara otvoreni upravljač PCM-a te vraća nulu za uspješno zatvaranje, a inače negativnu vrijedost.

```
int snd_pcm_hw_params_any ( snd_pcm_t *   pcm,
                snd_pcm_hw_params_t *    params );
```

Funkcija puni `params` sa punim konfiguracijskim prostorom PCM upravljača, a vraća negativan broj za grešku.

```
int snd_pcm_hw_params_set_rate_resample(snd_pcm_t *pcm,
                    snd_pcm_hw_params_t *      params,
                    unsigned int                  val );
```

Funkcija ograničava konfiguracijski prostor da sadrži samo sklopovski podržane frekvencije uzorkovanja. Vraća nulu ako je uspješna, inače negativnu vrijednost. Parametar `val`, ako je jedan tada omogućuje ponovno uzorkovanje, a ako je nula tada ga isključuje.

```
int snd_pcm_hw_params_set_access ( snd_pcm_t * pcm,
                    snd_pcm_hw_params_t *    params,
                    snd_pcm_access_t         access );
```

Navedena funkcija ograničava konfiguracijski prostor da ima samo jedan način pristupa, a vraća negativnu vrijednost za neuspjeh, nulu za uspjeh. Parametar `access` može poprimiti dvije vrijednosti `SND_PCM_ACCESS_RW_INTERLEAVED` ili `SND_PCM_ACCESS_RW_NONINTERLEAVED`. Taj parametar označuje kako su podaci spremljeni u međuspremniku. Za `INTERLEAVED` to znači da su u međuspremniku uzorci spremljeni naizmjenično, jedan uzorak za jedan kanal pa jedan za drugi i tako naizmjenično za dva kanala. Kod `NONINTERLEAVED` su svi uzorci jednog kanala prvo pa zatim svi uzorci drugog kanala spremljeni u međuspremnik.

```
int snd_pcm_hw_params_set_format ( snd_pcm_t * pcm,
                    snd_pcm_hw_params_t *     params,
                    snd_pcm_format_t          format );
```

Funkcija ograničava konfiguracijski prostor na samo jedan format zapisa uzorka u međuspremnik, a vraća za uspješno obavljanje nulu, inače negativnu vrijednost ako će konfiguracijski prostor postati prazan. Od strane biblioteke je podržan vrlo velik broj formata širine 8, 16, 24, 32 bita, uzorci bez predznaka i sa predznakom, cjelobrojni i s pomičnim zarezom... Najveći mogući format za audio sklop na *Beagleboardu* je `SND_PCM_FORMAT_S16_LE` jer je digitalno-analogni pretvornik 16 bitan i zapis podataka najniži bajt na najnižu adresu u memoriji, a najviši na veću adresu (eng. *Little Endian*).

```
int snd_pcm_hw_params_set_channels ( snd_pcm_t * pcm,
                    snd_pcm_hw_params_t *      params,
                    unsigned int               val );
```

Funkcija ograničava konfiguracijski prostor da sadrži samo jedan broj kanala te vraća nulu za uspješno obavljanje, a negativnu vrijednost ako će konfiguracijski prostor postati prazan. U parametar `val` se postavlja željeni broj kanala.

```
int snd_pcm_hw_params_set_rate_near ( snd_pcm_t *  pcm,
                          snd_pcm_hw_params_t *   params,
                          unsigned int *           val,
                          int *                    dir );
```

Funkcija ograničava konfiguracijski prostor na najbližu moguću brzinu uzorkovanja koja je sklopovski podržana. Vraća nulu za uspjeh, a negativnu vrijednost ako je konfiguracijski prostor prazan. U parametar `val` se postavlja željena brzina uzorkovanja, a nakon izvršavanja funkcije tu se nalazi postavljena vrijednost, a u parametar `dir` se postavlja vrijednost -1, 0 ili 1, ovisno o odnosu (manja, jednaka ili veća) tražene i dobivene brzine uzorkovanja.

```
int snd_pcm_hw_params_set_buffer_time_near
                         ( snd_pcm_t *              pcm,
                           snd_pcm_hw_params_t * params,
                           unsigned int *           val,
                           int *                    dir );
```

Funkcija ograničava konfiguracijski prostor kako bi vrijeme međuspremnika bilo što bliže vremenu na sklopu. Vraća nulu, ako je uspješna, a negativnu vrijednost ako je konfiguracijski prostor prazan. Parametar `val` postavlja željeno vrijeme, a nakon izvršavanja funkcije tu se nalazi postavljena vrijednost. U parametar `dir` se postavlja vrijednost -1, 0 ili 1 ovisno o odnosu (manja, jednaka ili veća) traženog i dobivenog vremena.

```
int snd_pcm_hw_params_get_buffer_size
               ( const snd_pcm_hw_params_t *   params,
                 snd_pcm_uframes_t *           val );
```

Funkcija dohvaća veličinu međuspremnika u broju okvira iz konfiguracijskog prostora. Vraća nulu ako je uspješna, a negativnu vrijednost ukoliko vrijednost nije jedinstvena. U parametru `val` se sprema veličina međuspremnika u broju okvira.

```
int snd_pcm_hw_params_set_period_time_near
                    ( snd_pcm_t *              pcm,
                      snd_pcm_hw_params_t *    params,
                      unsigned int *           val,
```

```
                    int *                   dir );
```

Funkcija postavlja vrijeme perioda što bliže traženom vremenu definiranom u parametru `val` u konfiguracijski prostor. Nakon izvršavanja funkcije u `val` se nalazi postavljena vrijednost, a u parametru `dir` se postavlja vrijednost -1, 0 ili 1 ovisno o odnosu (manje, jednako ili veće) traženog i dobivenog vremena perioda. Funkcija vraća nulu ako je uspješna, a negativnu vrijednost ako je konfiguracijski prostor prazan.

```
int snd_pcm_hw_params_get_period_size
                ( const snd_pcm_hw_params_t *   params,
                  snd_pcm_uframes_t *           val,
                  int *                         dir );
```

Funkcija dohvaća približnu veličinu perioda u broju okvira koju sprema u `val` parametar. U parametru `dir` se sprema vrijednost -1, 0 ili 1 ovisno o odnosu (manje, jednako ili veće) stvarne i približne veličine perioda. Funkcija vraća nulu ako je uspješna, a negativnu vrijednost ako ne postoji samo jedna vrijednost.

```
int snd_pcm_hw_params ( snd_pcm_t *             pcm,
                        snd_pcm_hw_params_t *   params );
```

Funkcija postavlja postavljene sklopovske parametre kroz konfiguracijski prostor i priprema PCM za korištenje. Sklopovski parametri se ne mogu mijenjati kada se pušta tok podataka. Funkcija vraća nulu ako je uspješna, a inače negativnu vrijednost.

```
int snd_pcm_sw_params_current ( snd_pcm_t *     pcm,
                      snd_pcm_sw_params_t *     params );
```

Funkcija postavlja trenutnu programsku konfiguraciju za PCM u parametar `parms` i vraća nulu ako je uspješna, inače negativnu vrijednost.

```
int snd_pcm_sw_params_set_start_threshold
                    ( snd_pcm_t *               pcm,
                      snd_pcm_sw_params_t *     params,
                      snd_pcm_uframes_t         val );
```

Funkcija postavlja graničnu vrijednost unutar programskog konfiguracijskog prostora. PCM se automatski starta kada su dostupni okviri veći ili jednaki od postavljene vrijednosti u parametru `val`. Funkcija vraća nulu ako je uspješna, inače negativnu vrijednost.

```
int snd_pcm_sw_params_set_avail_min
                    ( snd_pcm_t *              pcm,
                      snd_pcm_sw_params_t *    params,
                      snd_pcm_uframes_t        val );
```

Funkcija postavlja minimalan broj okvira koji je potreban da bi se PCM smatrao spremnim tj. minimalan broj okvira koji se prenosi. Taj broj se nalazi u parametru `val`. Funkcija vraća nulu ako je uspješna, inače negativnu vrijednost.

```
int snd_pcm_sw_params ( snd_pcm_t *            pcm,
                        snd_pcm_sw_params_t *  params );
```

Funkcija postavlja programsku konfiguraciju PCM-a definiranu s parametrom `params`. Programski parametri se mogu mijenjeti u bilo koje vrijeme za razliku od sklopovskih. Funkcija vraća nulu ako je uspješna, inače negativnu vrijednost.

```
snd_pcm_sframes_t snd_pcm_writei
                        ( snd_pcm_t *          pcm,
                          const void *         buffer,
                          snd_pcm_uframes_t    size );
```

Piše `INTERLEAVED` okvire u PCM. Parametar buffer sadrži okvire koji će se zapisivati, a parametar size sadrži broj okvira koji se trebaju zapisati. Funkcija vraća stvaran broj zapisanih okvira, a u slučaju neuspješnog pisanja negativnu vrijednost greške.

7. Korišteni alati

Razvoj softvera za *Beagleboard* je otvoreno podržan od zajednice te postoje besplatni alati i kompajleri. Preporučeni kompajler za ARM je od *CodeSourcery* pod imenom `arm-none-linux-gnueabi`, a za DSP je preporučen C6000 *Code Generator Tools* od *Texas Instruments*.

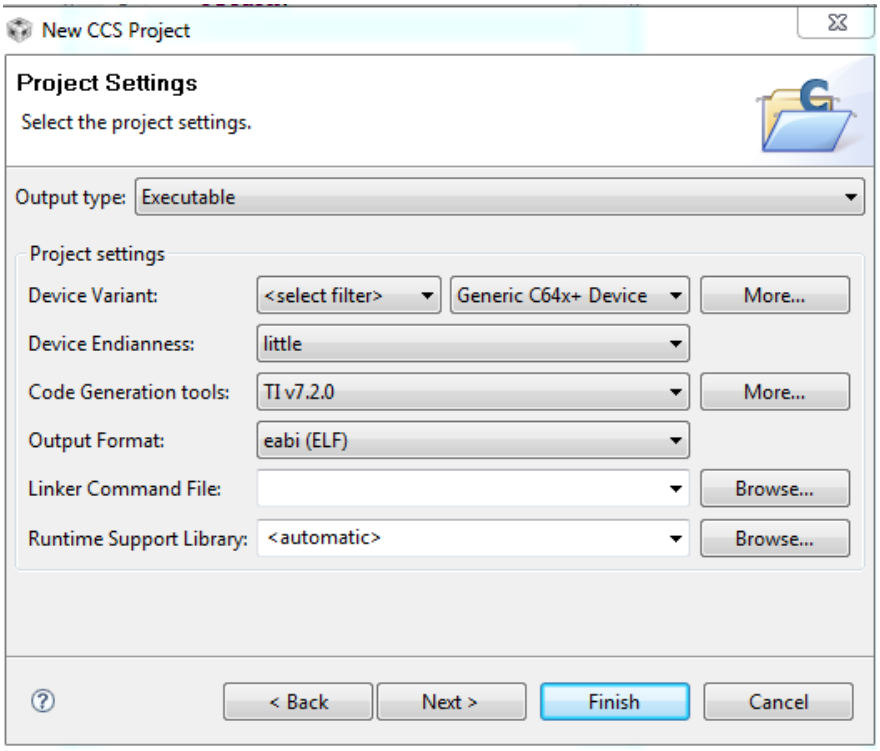
Razvojni alat koji unutar sebe već u sebe ima integrirane kompajlere za oba procesora je *Texas Instruments Code Composer Studio* (CCS). U verziji 4.1 CCS koristi C6000 kompajler verziju 7.2 za DSP procesor, a za ARM koristi TMS470 kompajler verziju 4.6.4. Kompajler TMS470 se nije pokazao najboljim što je pokazano na primjeru u potpoglavlju 7.2. te je preporuka da se koristi kompajler od *CodeSourcery*. U verziji 5.1 CCS koja je izašla u drugoj polovici svibnja 2011. godine, koja je bazirana na *Eclipse* 3.6 verziji, pa je unutar ove verzije CCS moguće koristiti i druge kompajlere (npr. *CodeSourcerya* za ARM procesore), ali i ostale dodatke koji su dostupni za *Eclipse*.

U izradi diplomskog rada je korišten CCS verzija 4.2.3 na windows platformi za pisanje i otklanjanje pogrešaka unutar funkcije koja će obavljati direktnu digitalnu sintezu sinusa. Završno stvaranje kompletnog programa je rađeno na linux platformi za ARM sa *CodeSourcery* `arm-none-linux-gnueabi` verzija 2011.03-41 i C6000 kompajler verzija 7.2.3.

7.1. Code Composer Studio – stvaranje novog projekta

Nakon pokretanja CCS-a potrebno je napraviti novi projekt `File -> New -> CCS Project`. Nakon toga otvara se novi prozor u koji upisujemo ime projekta i postavljamo lokaciju gdje se nalazi te kilknemo `next`. U slijedećem dijelu postavljamo tip projekta ARM ili C6000, a ostale postavke ostavimo i kliknemo `next`. U ovome prozoru ostavljamo sve predefinirano i samo kliknemo `next`. U sljedećem prozor postavljamo tip izlaza, vrstu uređaja,

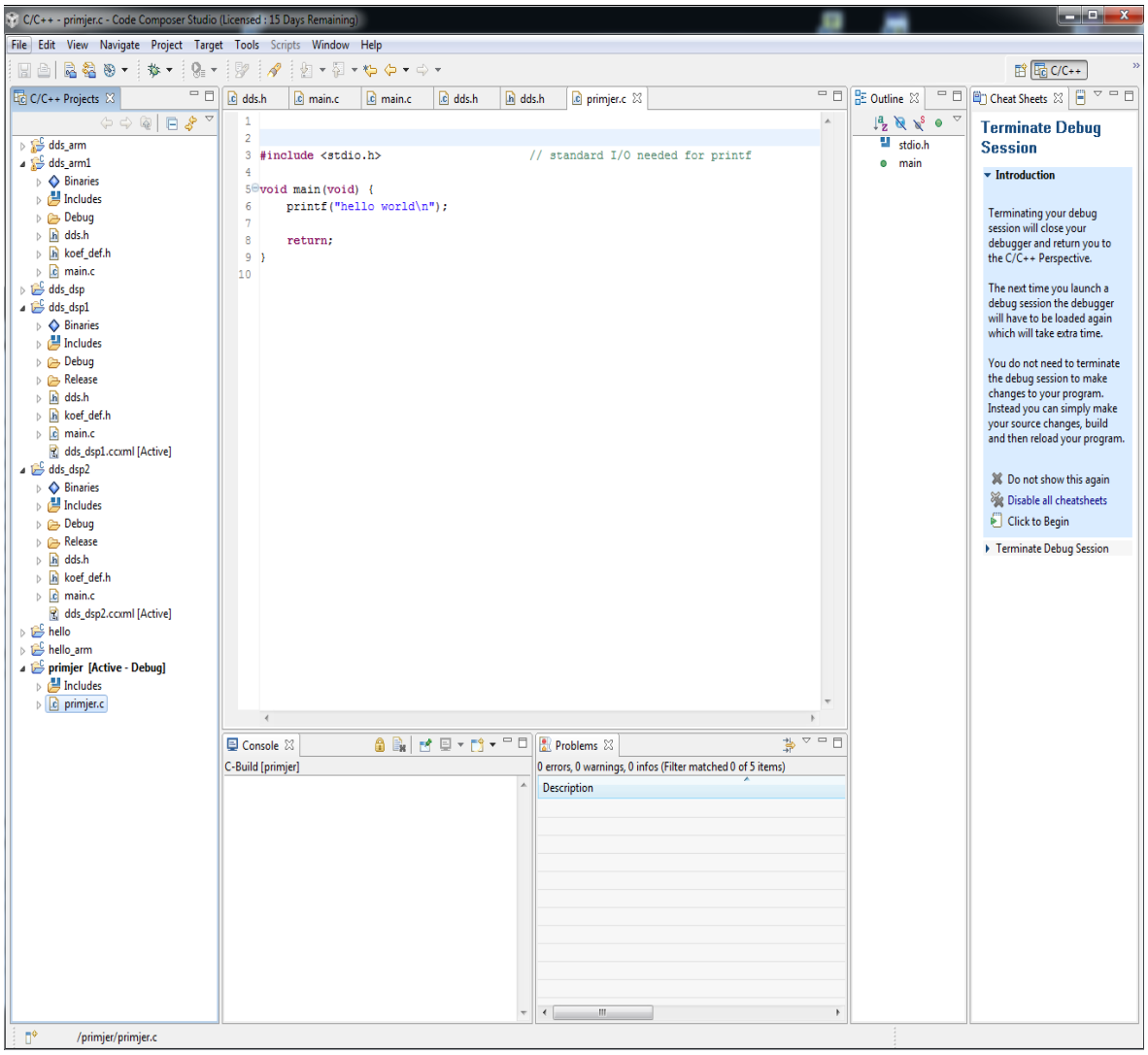
način pohrane u memoriju, kompajler, format izlaza, povezivač i podržane biblioteke. Na slici 15. su pikazane postavljene opcije koje se odnose na DSP procesor. Povezivač se može ostaviti prazan što znači da će CCS automatski uzeti predefinirani povezivač.



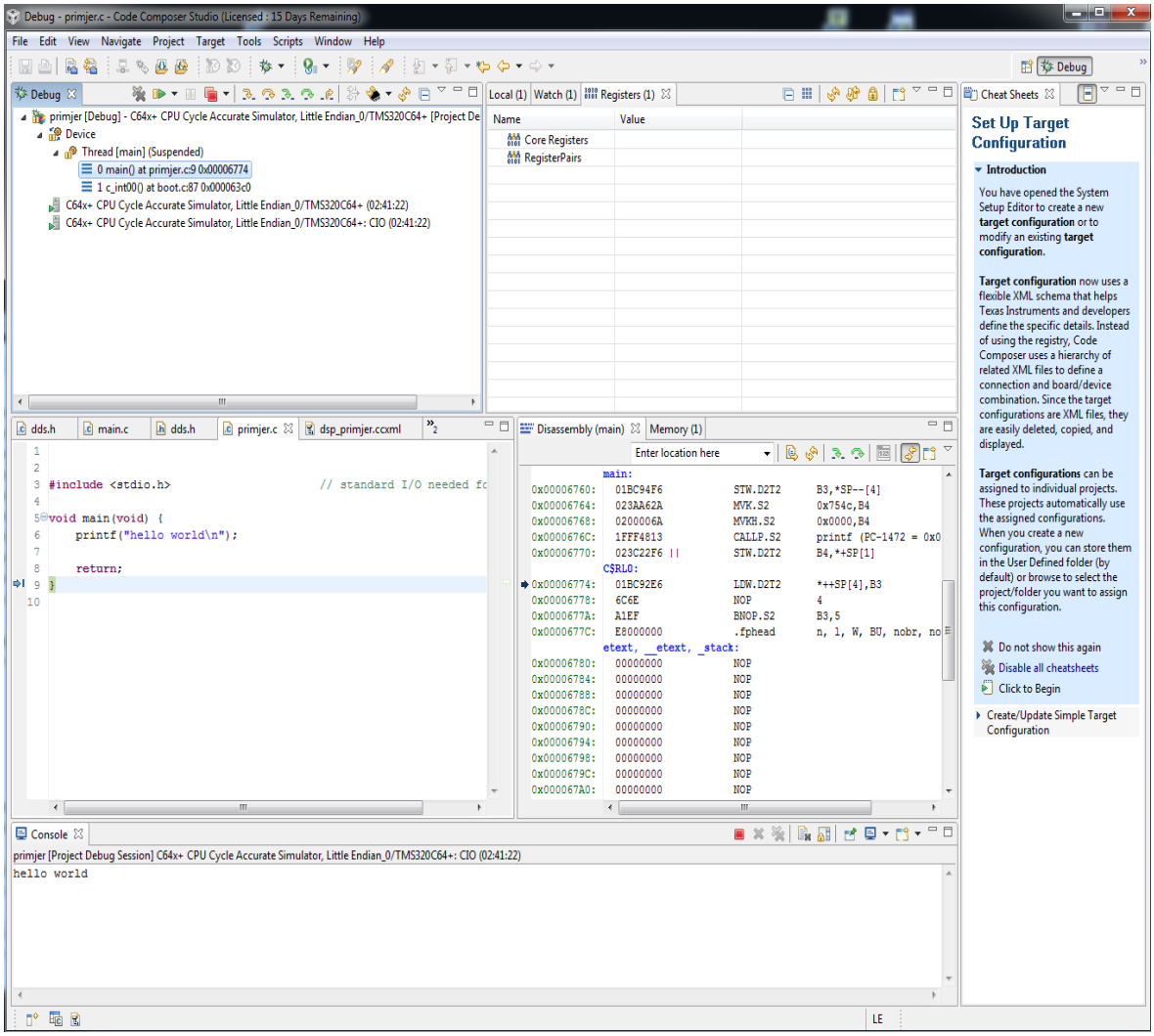
Slika 15. CCS postavke novog projekta

Nakon što smo kliknuli `next` u novom prozoru izaberemo Empty Project te kliknemo `finish`. Time smo stvorili novi projekt koji je prazan. Za dodavanje izvora kôda potrebno je `File -> New -> Source File` i u novome prozoru upisati ime s ekstenzijom (.c ili .cpp). Time imamo prvi projekt kojeg možemo kompajlirati, a da bi ga mogli testirati moramo još dodati u projekt konfiguracijsku datoteku `File -> New -> Target Configuration File` te u novo otvorenom prozoru izabrati potrebnu datoteku za DSP `C64x+ CPU Cycle Accurate Simulator. Little Endian`, a za `ARM ARM7TDMI CPU Cycle Accurate Simulator, Little Endian` i kliknuti `save`. Nakon što je konfiguracijska datoteka postavljena možemo simulirati program. Na slici 16. je prikazan radni prozor CCS-a, a na slici 17. je prikazan simulacijski prozor. Kao što je vidljivo na slici

moguće je pregledavanjeregistara, lokalnih varijabli, memorije, prikaz asemblerskog kôda te ostale standardne opcije.



Slika 16. CCS prikaz radnog prozora



Slika 17. CCS prikaz simulacijskog prozora

7.2. Code Composer Studio – greška u kompajleru

Greška koja se javlja prilikom izvršavanja funkcije direktne digitalne sinteze sinusa je:

```
TMS470R1x_0: Can't Single Step Target Program: Error number -1260 Error address 245ff
```

Ta greška se javlja prilikom izvršavanja ove C linije kôda:

```
dx = (*fa & MASK_DX) - NEG_DX;
```

Adresa za koju javlja da je greška je zapravo jednaka faznom broju. To je vidljivo u odsječku asemblerskog kôda:

```
LDR             R12, [R1]
LDRB            R12, [R12]
```

U registru `R1` u danom trenutku se nalazi vrijednost faznog broja, a kao što se vidi iz asemblerskih naredbi pokušava se pročitati vrijednost s adrese koja je pohranjena u `R1`.

Prilikom pokretanja `hello world` primjera se također javlja greška prilikom izvršavanja `printf` naredbe tj. ne javi se greška već program ostane zaglavljen unutar sljedećih asemblerskih naredbi:

```
...
LDR             R12, $C$CON1
LDR             R12, [R12]
BLX             0x00C02484
...
BX              R14
```

Nakon izvršavanja BLX naredbe program skače na BX naredbu koja vraća program na prvu LDR naredbu i tako u krug.

Upravo zbog ovih problema *Texas Instruments* kompajler za ARM nije korišten već kompajler od *CodeSourcery*.

7.3. CodeSourcery kompajler

CodeSourcery kompajler `arm-none-linux-gnueabi` je korišten unutar linuxa kroz komandni redak za kompajliranje ARM programa. Primjer takvog poziva kompajlera s uključenom ALSA bibliotekom i optimizacijom je sljedeći:

```
arm-none-linux-gnueabi-gcc –O2 -o ddss_arm main.c –I
/staza_do/alsa-lib/user/include/ -L  /staza_do/alsa-
lib/user/lib/ -l asound
```

`arm-none-linux-gnueabi-gcc` je poziv arm gcc kompajlera, `-O2` je stupanj optimizacije, `-o ime` označava gdje će se smjestiti izlaz kompajlera te pod kojim imenom, ime.c označava datoteku koja se kompajlira, `-I staza_do_zaglavlja` gdje su sadržana dodatna zaglavlja, `-L staza_do_biblioteke` dodaje dodatni put gdje će se tražiti dodatne biblioteke i posljednja opcija je `-l ime_biblioteke` koje se dodaje u listu datoteka koje se povezuju s bibliotekama. Ime biblioteke piše se bez prefiksa `lib` što znači da `asound` ne postoji na disku već `libasound`.

Dodavanjem opcije `-S` možemo dobiti na izlazu asemblerski kôd za zadanu ulaznu datoteku.

Postoji više stupnjeva optimizacije osim gore spomenute `-O2`, a to su slijedeće:

- `-O0` ovo je predefiniran stupanj optimizacije ukoliko se eksplicitno ne navede. Ovdje se ustvari ne radi o optimizaciji već se samo smanjuje vrijeme kompajliranja i stvara ispravljačka procedura radi lakšeg ispravljanja i analiziranja kôda.
- `-O1` ili `-O` za ovaj stupanj optimizacije kompajler pokušava smanjiti veličinu kôda i vrijeme izvršavanja bez uključivanja optimizacija koje bi značajno povećale vrijeme kompajliranja.
- `-O2` optimizira još više. Uključuje skoro sve optimizacije koje ne uključuju veličinu – brzina zamjenske optimizacije tj. one koje uključivanjem narušavaju veličinu kôda, a povećavaju brzinu i obrnuto. U odnosu na `-O1` ova optimizacija povećava vrijeme kompajliranja i poboljšava performanse generiranog kôda.
- `-O3` naravno optimizira još više od `-O2`, ali u smjeru poboljšanja brzine kôda uz vrlo vjerojatno povećanje veličine kôda.
- `-Os` je optimizacija za veličinu kôda. Uključuje sve optimizacije kao i `-O2`, ali ne one koje uglavnom povećavaju veličinu kôda.

7.4. C6000 kompajler

Kompajler C6000 je korišten kroz CCS kao što je već rečeno. Kompajliranje se odvija još jednostavnije nego za ARM obzirom da imamo grafičko sučelje koje klikom na `build project` poziva potrebni kompajler.

Kompajler ima slično kao i ARM nekoliko razina optimizacija. Optimizacije su od `-O0` do `-O3`. Možemo reći da je optimizacija s najnižim brojem najslabija, a s većim brojem jača s time da se ovdje odnos optimizacija brzina – veličina dodatno regulira opcijama `-mf` i `-ms`. Opcija `-mf` se odnosi na brzinu izvršavanja kôda te ima šest razina od 0 (omogućuje optimizaciju veličine kôda uz veliku mogućnost narušavanja brzine kôda) do 5 (omogućuje optimizaciju brzine kôda uz veliku mogućnost narušavanja veličine kôda), a `-ms` se odnosi na veličinu kôda te ima četiri razine od 0 (loša optimizacija veličine kôda) do 3 (najbolja optimizacija veličine kôda). U ovom slučaju opcija `-ms` nije uopće korištena, a opcija `-mf` je postavljena na petu razinu.

8. Direktna digitalna sinteza sinusa - C kôd

Direktna digitalna sinteza sinusa napisana u programsku jeziku C je prikazana u nastavku teksta. Prikazani kôd se može izvršavati i na ARM i DSP procesoru.

8.1. C-kôd - ARM

```
int dds (unsigned int *fa, unsigned int korak) {
      int dx, adr, msb_fa, nsb_fa;
      int as, bs, pr2, sum2, rez;
      short cs, ds, pr1, sum1;
      long long pr3;

      dx = ((*fa) & MASK_DX) - NEG_DX;
      msb_fa = ((*fa) & MASK_MSB) >> SHF_MSB;
      nsb_fa = ((*fa) & MASK_NSB) >> SHF_NSB;
      adr = ((*fa) & MASK_ADR1) >> SHF_ADR;
      adr = (!nsb_fa) * adr + (nsb_fa) * ((~adr) & MASK_8);

      *fa = ((*fa) + korak) & MASK_FA;

      ds = (msb_fa ^ nsb_fa) * du[adr] +
           !(msb_fa ^ nsb_fa) * (-du[adr]);
      cs = msb_fa * cu[adr] + !(msb_fa) * (-cu[adr]);
      bs = !(msb_fa ^ nsb_fa) * bu[adr] +
           (msb_fa ^ nsb_fa) * (-bu[adr]);
      as = !(msb_fa) * au[adr] + msb_fa * (-au[adr]);

      pr1 = (ds * (dx >> SHF_DX1));
      pr1 = pr1 >> SHF_PR1;
      sum1 = cs + (pr1 >> 1) + (pr1 & 0x1);
      pr2 = (sum1 * (dx >> SHF_DX2));
      pr2 = pr2 >> SHF_PR2;
      sum2 = bs + (pr2 >> 1) + (pr2 & 0x1);
      pr3 = (long long) sum2 * (long long) dx;
      pr3 = pr3 >> SHF_PR3;
      rez = as + (pr3 >> 1) + (pr3 & 0x1);

      return rez;
}
```

Programski kôd je napisan u obliku funkcije koja se poziva s parametrima `*fa` koja je početna faza i parametrom `korak` s kojim će se početna faza uvećavati, a vraća jednu vrijednost uzorka sinusoide. Kôd je podijeljen u četiri cjeline (osjenčane crvenom, zelenom, plavom i žutom bojom) koje odgovaraju opisanom programskom modelu u poglavlju 5.2. s razlikom što je tamo čitanje tablica koeficijenata odvojeno od dodjeljivanja predznaka koeficijentima.

U kôdu osjenčanom crvenom bojom se iz varijable `*fa` izlučuju `dx`, `msb_fa`, `nsb_fa` i `adr`. `dx` je širok 28 bita, a donjih 16 bita predstavlja `dx` bez predznaka koji se oduzimanjem od `NEG_DX` pretvara u predznačni broj širine 18 bita. `msb_fa` je najviši 28 bit od `*fa`. S `MASK_MSB` izbrišemo donjih 27 bita te dobiveni bit posmaknemo za `SHF_MSB` mjesta u desno da dođe na odgovarajuće mjesto. Isto vrijedi i za `nsb_fa` samo što se ovdje radi o bitu niže u `*fa`. `adr` je širok osam bita i nalazi se od 19 do 26 bita u `*fa`. S `MASK_ADR1` brišemo nepotrebne bitove i posmičemo `adr` na potrebno mjesto. Nakon toga bitove `adr` u ovisnosti o `nsb_fa`, ako je jedan onda radimo prvi komplement bitova varijable `adr`, ali moramo dodatno izbrisati ostale bitove maskom MASK_8 jer `adr` je tipa `int` pa da izbacimo gornjih 24 bita koji će komplementiranjem postati jedan, a ako je `nsb_fa` nula tada `adr` ne mijenjamo.

Kôd osjenčan zelenom bojom predstavlja fazni akumulator. Obzirom da je `*fa` tipa `int`, a treba biti širok 28 bita dodatno je ograničen maskom `MASK_FA`. Fazni akumulator je pokazivač tako da bi ponovnim pozivom funkcije vrijednost ostala sačuvana.

U kôdu osjenčanim plavom bojom se odvija čitanje tablica koeficijenata i dodjeljivanje predznaka u ovisnosti o `msb_fa` i `nsb_fa` kao što je prikazano u poglavlju 5.2. u tablici 1. dodjeljivanje predznaka nije izvedeno `if-else` naredbama da se izbjegne korištenje uvjetnih naredbi.

U zadnjem osjenčanom dijelu žutom bojom se odvija Hornerova evaluacija polinoma kao što je objašnjeno u poglavlju 5.2 i prikazano slikom 14. `SHF_DX1`, `SHF_PR1`, `SHF_DX2` i `SHF_PR3` služe za odbacivanje viškova bitova.

U nastavku teksta se nalaze sve korištene predefinirane vrijednosti:

```
#define NEG_DX          131072
#define MASK_DX         0x3FFFF
#define MASK_ADR1       0x3FC0000
#define MASK_8          0xFF
#define MASK_MSB        0x8000000
#define MASK_NSB        0x4000000
#define MASK_FA         0xFFFFFFF

#define SHF_MSB         27
#define SHF_NSB         26
#define SHF_ADR         18
#define SHF_DX1         12
#define SHF_DX2         1
#define SHF_PR1         3
#define SHF_PR2         15
#define SHF_PR3         16
```

8.1.1. Asemblerski kôd ARM

```
dds:
        .fnstart
.LFB18:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        ldr     ip, [r0, #0]
        stmfd   sp!, {r4, r5, r6, r7, r8, r9, sl, fp}
        .save {r4, r5, r6, r7, r8, r9, sl, fp}
        and     r3, ip, #66846720
        mov     r3, r3, lsr #18
        and     r2, ip, #67108864
        mvn     r9, r3
        mov     r2, r2, lsr #26
        and     r9, r9, #255
        mul     r9, r2, r9
        eor     r4, r2, #1
        mla     r9, r3, r4, r9
```

```
    ldr   r5, .L2
    mov   r6, r9, asl #1
    and   r3, ip, #134217728
    mov   r3, r3, lsr #27
    ldrh  r7, [r5, r6]
    eor   fp, r2, r3
    mov   r8, r7, asl #16
    mul   r7, fp, r7
    cmp   r3, r2
    mov   r8, r8, asr #16
    movne sl, #0
    moveq sl, #1
    bic   r4, ip, #-16777216
    add   r6, r5, r6
    rsb   r8, r8, #0
    mla   r8, sl, r8, r7
    bic   r4, r4, #16515072
    add   r6, r6, #512
    sub   r4, r4, #131072
    ldrh  r6, [r6, #0]
    mov   r7, r4, asr #12
    mul   r8, r7, r8
    mov   r7, r6, asl #16
    mul   r6, r3, r6
    mov   r7, r7, asr #16
    eor   sl, r3, #1
    rsb   r7, r7, #0
    mla   r6, sl, r7, r6
    mov   r8, r8, asl #16
    mov   r8, r8, asr #19
    mov   r8, r8, asl #16
    add   r6, r6, r8, asr #17
    mov   r8, r8, asl #15
    add   r6, r6, r8, lsr #31
    mov   r6, r6, asl #16
    mov   r7, r4, asr #1
    mov   r6, r6, asr #16
    mul   r6, r7, r6
    add   ip, r1, ip
    add   r9, r5, r9, asl #2
    bic   ip, ip, #-268435456
    str   ip, [r0, #0]
    ldr   r0, [r9, #1024]
    cmp   r3, r2
    rsbne r2, fp, #0
    rsbeq r2, fp, #1
```

```
        mov   r1, r6, asr #16
        mla   r2, r0, r2, r1
        mov   r6, r6, asl #16
        add   r6, r2, r6, lsr #31
        smull r0, r1, r6, r4
        ldr   r2, [r9, #2048]
        mov   r4, r0, lsr #16
        orr   r4, r4, r1, asl #16
        mov   r5, r1, asr #16
        movs  r7, r5, asr #1
        mov   r6, r4, rrx
        and   r4, r4, #1
        add   r6, r4, r6
        rsb   r3, r3, sl
        mla   r0, r3, r2, r6
        ldmfd sp!, {r4, r5, r6, r7, r8, r9, sl, fp}
        bx    lr
```

Prikazani asemblerski kôd odnosi se samo na funkciju koja obavlja direktnu digitalnu sintezu. Asemblerski kôd dobiven je pokretanjem kompajlera sa opcijom `-S` uz korištenu optimizaciju `-O3`.

8.2. Razlika u C kôdu za DSP

Razlika u odnosu na gore dani kôd je u tome što se u deklaraciji parametra `*fa` dodaje ključna riječ `restrict`. `restrict` se može koristiti uz pokazivače, reference ili polja. Ona kaže kompajleru da se objektu na koji pokazuje pokazivač može pristupiti samo s tim pokazivačem. Bilo koje narušavanje tog uvijeta čini program nedefiniranim, ali pomaže kompajleru u optimizaciji određenih sekcija kôda jer se preklapanje informacija može lakše detektirati.

```
int dds (unsigned int * restrict fa, unsigned int korak);
```

Druga razlika je kod zadnjeg množenja u Hornerovoj evaluaciji polinoma koje daje rezultat širine 41 bit. Za množenje se koristi unutarnja (eng. *Intrinsics*) naredba kompajlera.

```
pr3 = (_mpy32ll (sum2, dx));
```

Naredba `_mpy32ll` radi množenje dva 32 bitna broja i vraća 64 bitni rezultat. Unutarnje naredbe kompajlera počinju sa „_" te se direktno preslikavaju u instrukcije procesora i time povećavaju brzinu optimizacije.

8.2.1. Asemblerski kôd DSP

```
            NOP             1
            ADD     .L1     A5,A4,A5          ; |15|
            LDH     .D1T1   *+A10[A5],A4      ; |27|
            NOP             3
            XOR     .L1     A3,A6,A16         ; |27|

            NEG     .S1     A4,A3             ; |27|
||          CMPEQ   .L1     A6,A3,A4          ; |27|
||          MPY     .M1     A16,A4,A7         ; |27|

            MPY     .M1     A4,A3,A17         ; |27|
||          LDH     .D1T1   *+A13[A5],A31     ; |27|

            NOP             1

            ADD     .L1     A17,A7,A7         ; |27|
||          SHR     .S1     A9,12,A8          ; |27|

            MPY     .M1     A8,A7,A7          ; |27|
            NOP             1

            NEG     .L1     A31,A8            ; |27|
||          EXT     .S1     A7,16,16,A7       ; |27|

            EXT     .S1     A7,13,16,A3       ; |27|
||          MPY     .M1     A31,A6,A17        ; |27|
||          CMPEQ   .L2X    A6,0,B4           ; |27|

            MPY     .M2X    B4,A8,B5          ; |27|
||          EXT     .S1     A7,13,17,A7       ; |27|
||          AND     .L1     1,A3,A3           ; |27|

            PACK2   .L1     A3,A7,A3          ; |27|
            PACK2   .L2X    B5,A17,B5         ; |27|
```

```
           ADD2    .L2X    A3,B5,B5          ; |27|
           DOTP2   .M2     B5,B10,B6         ; |27|
           NOP             2
           SHR     .S2X    A9,1,B31          ; |27|

           EXT     .S2     B6,16,16,B6       ; |27|
||         LDW     .D1T1   *+A11[A5],A8      ; |34|
||         MVKL    .S1     au,A30

           MPYLI   .M2     B6,B31,B7:B6      ; |27|
||         MVKH    .S1     au,A30

           LDW     .D1T1   *+A30[A5],A7      ; |34|
           NOP             2

           SHR     .S2     B6,15,B5          ; |27|
||         MPYLI   .M1     A4,A8,A5:A4       ; |34|

           SHR     .S2     B5,1,B5           ; |34|
||         AND     .L2     1,B5,B6           ; |34|
||         MPY32   .M1     A16,A8,A3         ; |34|

           MPYLI   .M2X    B4,A7,B5:B4       ; |34|
||         ADD     .L2     B6,B5,B6          ; |34|

           NOP             2
           ADD     .L2X    A4,B6,B30         ; |34|

           SUB     .L2X    B30,A3,B5         ; |34|
||         EXT     .S1     A6,16,16,A29      ; |34|

           MPY32   .M2X    B5,A9,B7:B6       ; |34|
||         MPYLI   .M1     A29,A7,A5:A4      ; |34|

           NOP             3
           SUB     .L1X    B4,A4,A3          ; |34|
           SHL     .S1X    B7,0x10,A5        ; |34|

           SHL     .S1X    B7,0xf,A4         ; |34|
||         SHRU    .S2     B6,0x10,B5        ; |34|

           CALL    .S1     printf            ; |34|
||         SHRU    .S2     B6,0x11,B4        ; |34|

           OR      .L1X    A5,B5,A4          ; |34|
||         OR      .L2X    A4,B4,B4          ; |34|
```

```
           AND     .L1     1,A4,A3          ; |34|
||         ADDU    .L2X    B4,A3,B5:B4      ; |34|

           NOP             1

           ADDU    .L2X    A3,B4,B5:B4      ; |34|
||         STW     .D2T2   B8,*+SP(4)       ; |34|

           ADDKPC  .S2     $C$RL0,B3,0      ; |34|
||         STW     .D2T2   B4,*+SP(8)       ; |34|
```

Dobiveni asemblerski kôd je nastao uz opcije optimizacije `-O3` i `-mf 5`, a za te optimizacije kôd ima najbolje performanse kao što je vidljivo po tablici 2. Asemblerski kôd nije cijeli već samo odsječak unutar kojega se izvršava direktna digitalna sinteza sinusa. Moguće je da aseblerski kôd nije potpun tj. da nedostaje neka instrukcija koja se zbog optimizacije i paralelizma procesora nalazi više ili niže od danog odsječka kôda, ali ono što je sigurno je da se unutar ovoga kôda dobiva uzorak sinusoide koji se nalazi u registru `B4`.

Dvije paralelne linije ispred asemblerskih naredbi označavaju da se te naredbe izvode paralelno, zajedno sa prvom naredbom koja se nalazi iznad naredbe/i sa paralelnim linijama. Unutar danog odsječka pet skupina od po tri naredbe se izvode zajedno u paraleli što znači da se te instrukcije izvedu unutar jednog takta procesora. Da se cijeli odsječak izvrši potrebno je ukupno 48 takta procesora.

Nakon opisnika asemblerske naredbe slijedi oznaka koja označuje na kojim funkcijskim jedinicama se izvršava naredba, a nakon nje slijede podaci nad kojima se izvršava zadana operacija naredbom.

Instrukcije označene crvenom bojom izvršavaju završno zbrajanje međuprodukta `pr3` i koeficijenta `A` te zaokruživanje (zadnja instrukcija u žuto osjenčanom dijelu koda u cijelini 8.1).

8.3. Rezultati

Oba kôda i za ARM i za DSP se u potpunosti podudaraju sa referentnim kôdom u *Matlabu*. Stoga su ostale nepromijenjene pogreške u odnosu na idealnu sinusoidu.

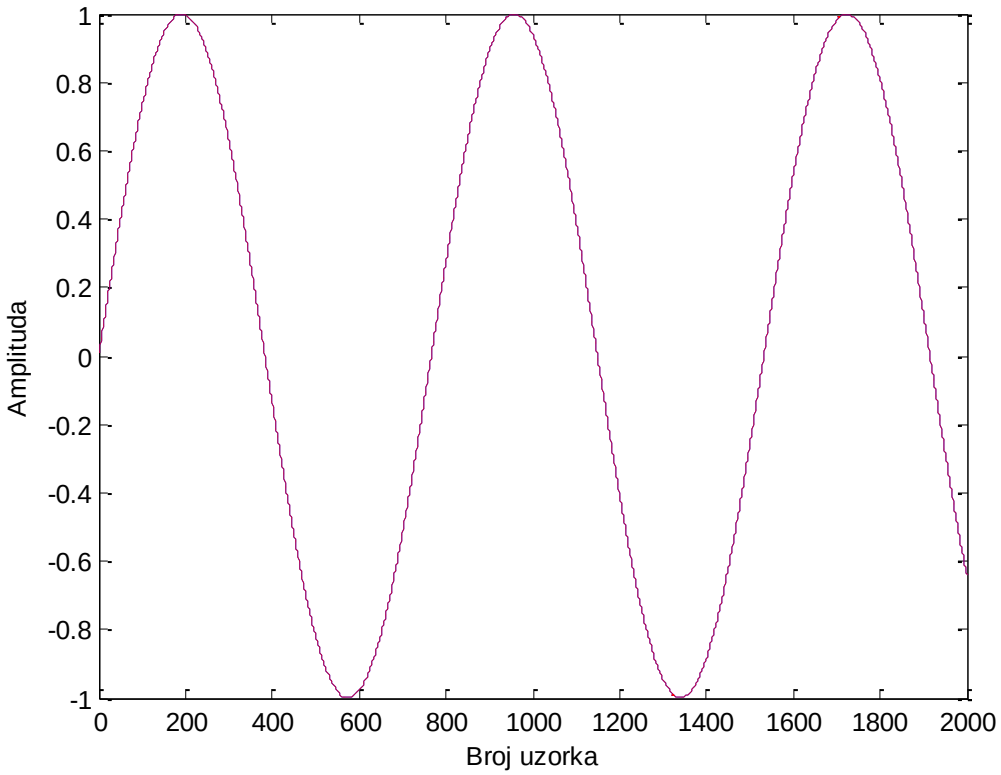
Maksimalna apsolutna pogreška u najmanje značajnom bitu iznosi:

0.76092422

Standardna devijacija pogreške u najmanje značajnom bitu iznosi:

0.13533466

Na slici 18. se može vidjeti i grafički prikaz idealne (prikazana crvenom isprekidanom linijom) i dobivene (prikazana plavom bojom) sinusoide za 2000 uzoraka te se i vizualno podudaraju. Početna faza i korak za prikazanu sinusoidu je 350451 kao i za gore izračunate pogreške.



Slika 18. Prikaz dobivene i idealne sinusoide

8.3.1. Usporedba razina optimizacija na DSP procesoru

Kao što je već bilo rečeno u poglavlju 7.4. DSP kompajler ima više razina optimizacija. U tablici 2. su prikazani odnosi u broju taktova za različite stupnjeve optimizacije. Programi se u strukturi ne razlikuju već je jedino razlika u tipu korištenih varijabli obzirom da DSP ima specijalizirane instrukcije za izvođenje nad manjim širinama podataka pa može još bolje optimizirati kôd. U prvoj koloni se nalaze podaci za program koji koristi reducirane veličine varijabili gdje god je moguće, a u drugoj koloni se nalaze podaci za program koji uglavnom koristi za sve varijable `int` tip podatka.

Opcije koje su još korištene uz optimizacijske opcije, a usmjerene su poboljšanju performansi su:

1. `-mf 5` –> maksimalno povećaj performanse kôda
2. `-ms 0` –> veličina kôda je nebitna
3. `-symdebug:none` -> isključuje ispravljanje i potiskuje sve simbole ispravljanja koji se pojavljuju u izlaznoj datoteci

Tablica 2. Broj potrebnih taktova DSP-a

	-O0	-O1	-O2	-O3
DDSS1	6 723	6 608	6 209	1 276
DDSS2	6 623	6 908	6 309	1 154

Podaci u tablici 2 su dobiveni korištenjem *Code Composer Studio* i simuliranjem kôda u ispravljačkom načinu rada sa spomenutim opcijama. Broj taktova se računao od samog početka programa pa sve do poziva `fopen` funkcije (tu je ujedno kraj petlje), a funkcija koja obavlja direktnu dikitalnu sintezu sinusa se je pozivala 100 puta unutar petlje.

Rezultati prvog retka su uglavnom bolji od rezultata drugog retka, a povećavanjem stupnja optimizacije rezultati su bolji u odnosu na manji stupanj optimizacije. Neočekivani rezultat je u drugom retku za optimizacije `-O0` i `-O1`

gdje se je optimizacija `-O0` pokazala bolja. Drugi neočekivani rezultat se pojavljuje kod optimizacije `-O3` gdje se je pokazalo da je bolja ona verzija koja koristi sve varijable tipa `int`. Mogući razlog tome je „nespretno" mjerenje pa je moguće da su u broj taktova uletjele neke druge instrukcije koje se nisu brojale u drugoj verziji.

9. Pokretanje programa na Beagleboardu

Za pokretanje izrađenog programa koji inicijalizira audio kodek i izvršava direktnu digitalnu sintezu sinusa te njezine uzorke šalje na izlaz je izvršavano pod linux operacijskim sustavom Angstrom verzije jezgre 2.6.32. Angstrom linux je prilagođena verzija linuxa za ugradbene sustave. Sustav se pokreće sa memorijske kartice koja se mora posebno pripremiti da bi se operacijski sustav mogao pokrenuti.

9.1. Priprema SD kartice za pokretanje OS Angstrom

Kartica se priprema pod linux operacijskim sustavom. Preporuka je da se koristi kartica minimalne veličine od 2 GB da ima dovoljno prostora za sve potrebne datoteke.

Na SD kartici će se nalaziti dvije particije od kojih će jedna služiti za pokretanje sustava, a druga će biti sistemska particija gdje će se nalaziti operacijski sustav i korisničke datoteke.

Nakon što smo umetnuli karticu u čitač kartica, karticu je potrebno formatirati na točno određeni način da bi se sustav mogao bez problema pokrenuti. U nastavku je primjer za 1 GB karticu.

9.1.1. Formatiranje SD kartice sa „fdisk“ alatom u „Expert mode“

1. Najprije obrišemo particijsku tablicu na slijedeći način:

```
============================================================
$ sudo fdisk /dev/sdb

Command (m for help): o
Building a new DOS disklabel. Changes will remain in
memory only,
until you decide to write them. After that, of course,
the previous
content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be
corrected by w(rite)
============================================================
```

2. Ispisujemo informacije o kartici:

```
============================================================
Command (m for help): p

Disk /dev/sdb1: 1023 MB, 1023809024 bytes
32 heads, 63 sectors/track, 991 cylinders
Units = cylinders of 2016 * 512 = 1032192 bytes
Disk identifier: 0x6a115c34

     Device Boot      Start         End      Blocks   Id
System
============================================================
```

3. Prelazimo u „Expert mode“:

```
============================================================
Command (m for help): x
============================================================
```

4. Sada postavljamo geometriju na 255 glava, 63 sektora i računamo broj potrebnih cilindara za SD karticu:

```
============================================================
Expert command (m for help): h
Number of heads (1-256, default 4): 255

Expert command (m for help): s
Number of sectors (1-63, default 63): 63
Warning: setting sector offset for DOS compatiblity
============================================================
```

Prije sljedećeg koraka prvo moramo izračunati broj cilindara prema formuli:

$$C = B/255/63/512 \tag{12}$$

gdje je B broj bajtova kartice, a C broj cilindara.

5. Kada dobijemo broj prema (12) (zaokružimo ga prema dolje):

```
============================================================
Expert command (m for help): c
Number of cylinders (1-1048576, default 991): 12
============================================================
```

6. Sada stvaramo prvu particiju u FAT32 formatu za podizanje sustava i prebacivanje datoteka kroz operacijski sustav windows:

```
============================================================
Expert command (m for help): r
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-245, default 1): (press Enter)
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-245,
default 245): +50
```

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))

Command (m for help): a
Partition number (1-4): 1
============================================================
```

7. Stvaranje druge particije za linux:

```
============================================================
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (52-245, default 52): (press Enter)
Using default value 52
Last cylinder or +size or +sizeM or +sizeK (52-245,
default 245):(press Enter)
Using default value 245
============================================================
```

8. Ispis novih particija i spremanje novih zapisa:

```
============================================================
Command (m for help): p

Disk /dev/sdc: 2021 MB, 2021654528 bytes
255 heads, 63 sectors/track, 245 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

  Device Boot      Start         End      Blocks   Id
System
/dev/sdc1   *           1          51      409626    c
W95 FAT32 (LBA)
/dev/sdc2              52         245     1558305   83
Linux
```

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error
16: Device or resource busy. The kernel still uses the
old table. The new table will be used at the next reboot.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for
additional information.
Syncing disks.
=========================================================
```

9. Nakon što smo stvorili particije, sada ih moramo formatirati: (NAPOMENA: Ukoliko particije ne postoje, potrebno je izvaditi i ponovo staviti karticu u čitač i tada će linux prepoznati nove particije)

```
=========================================================
$ sudo mkfs.msdos -F 32 /dev/sdc1 -n LABEL
mkfs.msdos 2.11 (12 Mar 2005)

$ sudo mkfs.ext3 /dev/sdc2
mke2fs 1.40-WIP (14-Nov-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
195072 inodes, 389576 blocks
19478 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=402653184
12 block groups
32768 blocks per group, 32768 fragments per group
16256 inodes per group
Superblock backups stored on blocks:
       32768, 98304, 163840, 229376, 294912
```

```
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting
information:
============================================================
```

Nakon što smo formatirali kartice smo gotovi. Sada možemo prebaciti potrebne datoteke na karticu.

9.1.2. Prebacivanje potrebnih datoteka na karticu

Nakon što smo stvorili dvije particije možemo prebaciti potrebne datoteke. Na prvu particiju prebacujemo slijedeće datoteke:

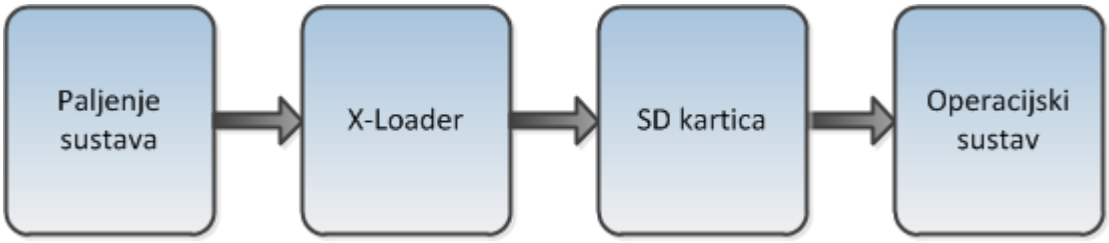
```
$ cp MLO /mnt/sda1
$ cp u-boot.bin /mnt/sda1
$ cp uImage /mnt/sda1
```

Vrlo je važno da se prvo zapiše `MLO` jer se on mora nalaziti u prvom sektoru SD kartice. `MLO` je ustvari `X-Loader` koji će pokrenuti `u-boot`. `U-boot` je osnovna okolina koja omogućuje pokretanje operacijskog sustava, ažuriranje *flash* memorije i druge stvari. `uImage` je jezgra linuxa. Na slici 19. je prikazano kojim se redosljedom pokreće sustav, u ovom slučaju na *Beagleboardu*. Na mjestu SD kartice na slici 19. može biti NAND memorija, USB ili UART.

Nakon što smo prebacili potrebne datoteke na prvu particiju možemo na drugu particiju prebaciti sistemske datoteke:

```
$ sudo tar -xjv -C /media/Angstrom -f
/path/to/angstrom_backup.tar.bz2
$ sync
```

Prebacivanje sistemskih datoteka može vremenski potrajati obzirom da se radi o većoj količini datoteka. Naredbu `sync` je potrebno pokrenuti da bi se osigurali da će se sve datoteke iz priručne memorije pravilno zapisati na SD karticu.



Slika 19. Redosljed pokretanja sustava na *Beagleboardu*

9.2. Pokretanje Beagleboarda

Na *Beagleboard* nisu spajane dodatne periferije tipa tastature, miša ili monitora već minimalan broj periferija da bi se *Beagleboard* mogao pokrenuti i prikazati izlaze. Tako su korišteni serijski DB9-F na DB9-F ukriženi kabel (eng. *null modem*) koji se spaja na dodatni kabel IDC10 na DB9 kabel jer se na *Beagleboardu* nalazi IDC10 konektor. Povezivanje serijskim kabelom se koristi za komunikaciju između *Beagleboarda* i računala.

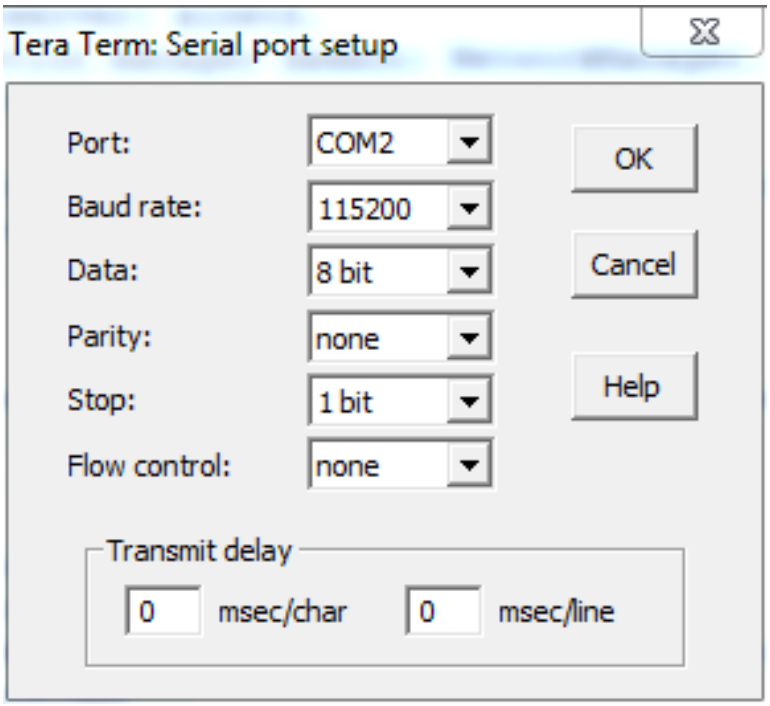
Beagleboard koristi brzinu prijenosa od 115200 bitova po sekundi, a šalje se osam bitova podatka i jedan stop bit uz isključenu kontrolu toka i onemogućenu provjeru pariteta. Na slici 22. su prikazane postavke serijske komunikacije za korišteni program *Tera Term*.

Za napajnje *Beagleboarda* se koristi standardni USB-A muški na mini USB-B muški kabel Y tipa koji se spaja na 5 pinski USB OTG mini-AB ženski konektor. Kabel je Y tipa zato što u nekim slučajevima kada je *Beagleboardu* potrebno više struje za rad, samo jedan USB port na računalu na koje je spojeno ne može dati dovoljno struje.

Slika 20. Serijski ukriženi kabel

Slika 21. IDC10 na DB9 kabel

Zadnje što je korišteno su slušalice koje su spojene na 3.5 mm konektor na *Beagleboardu* kako bi mogli čuti što šaljemo na audio izlaz.



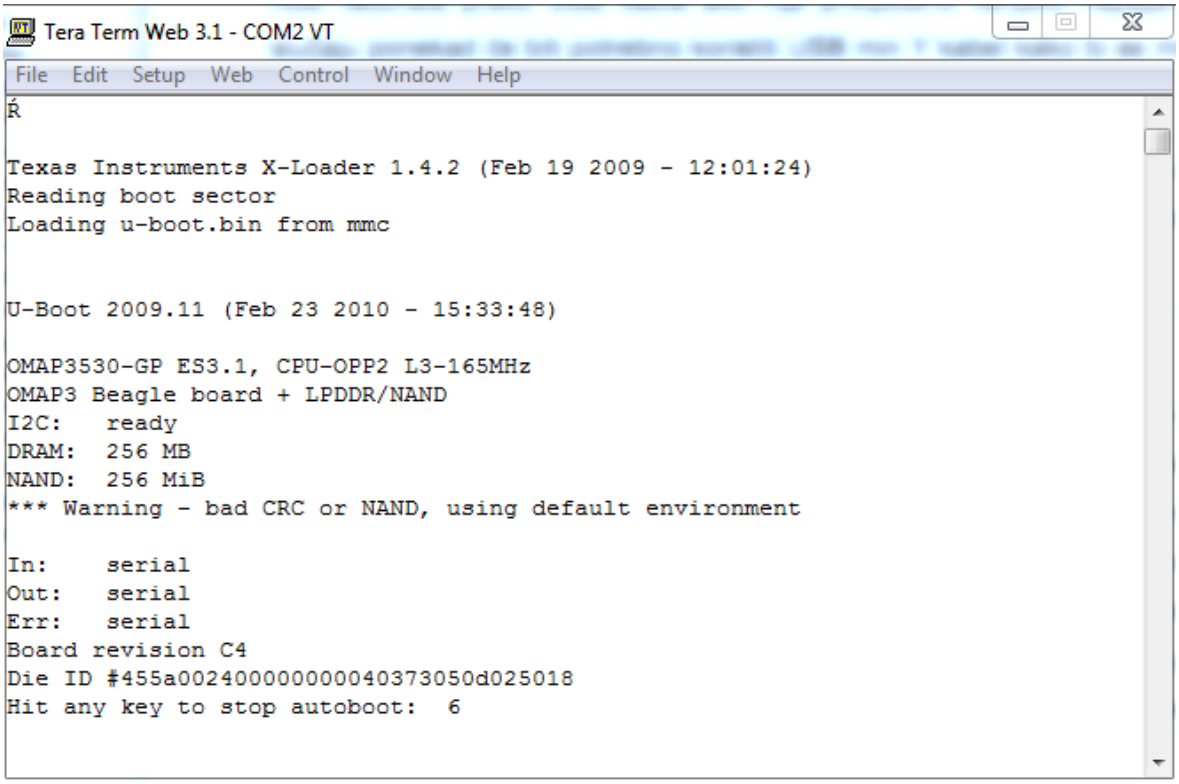
Slika 22. Postavke serijske komunikacije

Slika 23. USB mini Y kabel

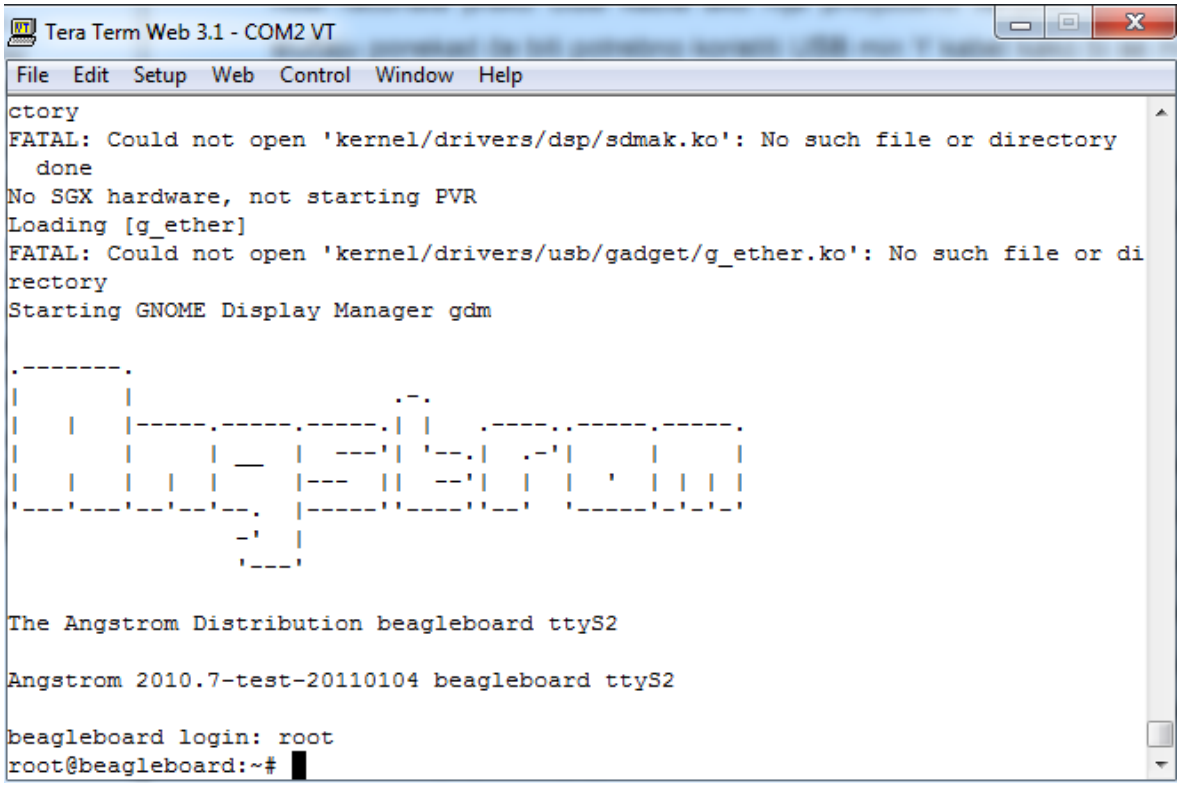
Nakon što je sve priključeno i postavljeno, paljenjem *Beagleboarda* se dobiva ispis kao što je prikazano na slici 24, nakon čega se automatski pokreće *Angstrom* operacijski sustav. Na sustav se prijavljuje upisivanjem `root` kao što je prikazano na slici 25.

Nakon što je sustav pokrenut potrebno se je samo pozicionirati u direktorij gdje se nalazi program te ga pokrenuti. U primjeru niže se program nalazi na prvoj particiji u direktoriju `dds_test`. Ukoliko sustav javi da ne može otvoriti audio sklop vrlo vjerojatno treba „ubiti" `pulseaudio` proces.

```
root@beagleboard:~# cd /media/mmcblk0p1/dds_test/
root@beagleboard:/media/mmcblk0p1/dds_test# ls
dds_test
root@beagleboard:/media/mmcblk0p1/dds_test# ./dds_test
Greska kod otvaranja: Device or resource busy
root@beagleboard:/media/mmcblk0p1/dds_test# killall -9
pulseaudio
root@beagleboard:/media/mmcblk0p1/dds_test# ./dds_test
```



Slika 24. Prvi ispis nakon paljenja Beagleboarda



Slika 25. Izlaz nakon pokretanja operacijskog sustava

10. Zaključak

Program za direktnu digitalnu sintezu sinusa je izrađen za oba procesora u C programskom jeziku. U oba slučaja program daje egzaktne rezultate kao i referentni kod u *Matlabu* s koji je uspoređen. Performanse programa su zadovoljavajuće pa stoga program nije pisan i u asembleru. Dodatno izvlačenje maksimalnih performansi programa je moguće jedino direktnim pisanjem asemblerskog koda.

Program za DSP procesor se u trenutku pisanja ovoga teksta nije uspio pokrenuti na *Beagleboardu* već se je samo testirao na simulatoru unutar programskog paketa *Code Composer Studio*.

Program za ARM procesor je uspješno pokrenut na *Beagleboardu*, ali pod linux okruženjem. Kao rezultat pokretanja, na slušalicama ili zvučnicima se na jednom kanalu čuje sinusni ton (gornjih 16 bita), a na drugome šum (donjih 16 bita).
Iako se prvotno planiralo napisati samostalni program koji bi se izvršavao na *Beagleboardu*, to nije napravljeno zbog nedostatka gotovih upravljača dodatnim sustavima koji bi bili potrebni za „oživljavanje" *Beagleboarda*, a njihovo pisanje bi bilo vremenski prezahtjevno i prelazilo bi obujam samog diplomskog zadatka.

Beagleboard sustav kao razvojni sustav je vrlo dobar te omogućuje jeftin početak svima koji žele raditi s ugradbenim računalnim sustavima, a kako je podržan od otvorene zajednice dostupno je puno informacija i alata na internetu koji su naravno besplatni.

Literatura

[1] BBSRM_latest, 15. prosinca 2009.
Beagleboard system reference manual Rev C4
http://beagleboard.org/static/BBSRM_latest.pdf
(7. ožujka 2011.)

[2] Beagle_C4B_Schematic, 15. prosinca 2009.
http://beagle.s3.amazonaws.com/design/Beagle_C4B_Schematic.pdf
(1. travnja 2011.)

[3] Texas Instruments, spruf98r, travanj 2010.
OMAP35x Application Processor Technical Reference Manual
http://www.ti.com/litv/pdf/spruf98r
(13. travnja 2011.)

[4] ARM, DDI0344K_cortex_a8_r3p2_trm, 15. svibnja 2009.
Cortex –A8 Technical Reference Manual
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf
(13. travnja 2011.)

[5] Texas Instruments, spru732j, srpanj 2010.
TMS320C64x/C64+ DSP CPU and Instruction Set Reference Guide
http://focus.ti.com/lit/ug/spru732j/spru732j.pdf
(11. travnja 2011.)

[6] Texas Instruments, spru198j, travanj 2010.
TMS320C6000 Programmer's Guide
http://focus.ti.com/lit/ug/spru198j/spru198j.pdf
(14. travnja 2011.)

[7] Texas Instruments, spru187s, 28. siječnja 2011.
TMS320C6000 Optimizing Compiler v 7.2 User's Guide
http://focus.ti.com.cn/cn/lit/ug/spru187s/spru187s.pdf
(27. travnja 2011.)

[8] Texas Instruments, spru186u, 28. siječanj 2011.
TMS320C6000 Assembly Language Tools v 7.2 User's Guide
http://focus.ti.com.cn/cn/lit/ug/spru186u/spru186u.pdf
(27. travnja 2011.)

[9] Texas Instruments, swcu050g, travanj 2008.
TPS65950 OMAP Power Management and System Companion Device Silicon Revision 1.2 Version G Technical Reference Manual
http://focus.ti.com/general/docs/lit/getliterature.tsp?literatureNumber=swcu050g&fileType=pdf
(16. svibnja 2011.)

[10] Texas Instruments, swcs032e, kolovoz 2008.
TPS65950 Integrated Power Management/Audio Codec Silicon Revision 1.2 Data Manual
http://focus.ti.com/lit/ds/swcs032e/swcs032e.pdf
(16. svibnja 2011.)

[11] Davor Petrinović, Member IEEE and Marko Brezović: „*Direct Digital Frequency Synthesis Using B-Spline Piecewise-Polynomial Model*", unpublished paper

[12] Prof. dr. sc. Davor Petrinovic, Matlab referentni kôd, 29. listopada 2010.

[13] Danijel Babić, diplomski rad br. 305, lipanj 2011.
„*Razvoj korisničkih aplikacija za beagleboard pločicu pod linux operacijskim sustavom*"

[14] BeagleBoardBeaginners
http://elinux.org/BeagleBoardBeginners
(31. svibnja 2011.)

[15] BeagleBoard
http://elinux.org/BeagleBoard
(25. ožujka 2011.)

[16] ALSA Home Page
http://www.alsa-project.org/main/index.php/Main_page
(1. lipnja 2011.)

Dodatak A

Naslov:

Direktna digitalna sinteza sinusa visoke točnosti na procesoru OMAP porodice

Sažetak:

Cilj rada je napisati program koji će izvršavati direktnu digitalnu sintezu sinusa iz zadane faze te ga pokrenuti na *Beagleboardu*. Za simulaciju i kompajliranje programa za DSP procesor je korišten *Code Composer Studio*, a za kompajliranje programa za ARM procesor *CodeSourcery*. Optimizacije C koda su dobivene korištenjem optimizacijskih mogućnosti kompajlera. Dobiveni izlazi programa za direktnu digitalnu sintezu sinusa su u potpunosti jednaki onima u referentnom *Matlab* kodu. Dobiveni izlaz je širine 32 bita.

Izvršavanje programa na *Beagleboardu* se radi pod *Angstrom* (linux) operacijskim sustavom. Za komunikaciju između procesora i audio kodeka se koristi ALSA biblioteka. Audio kodek je potrebno pravilno konfigurirati da bi se podaci mogli slati na audio izlaz i samim time čuti na zvučnicima. Pokretanjem programa na jedan kanal se šalje gornjih 16 bita sinusoide što se reprezentira kao ton u zvučniku, a na drugi kanal se šalje donjih 16 bita sinusoide što se reprezentira kao šum.

Ključne riječi:

Direktna digitalna sinteza, Hornerovo pravilo, Beagleboard, Texas Instruments, OMAP3530, ARM, Cortex A8, DSP, TM320C64x+, TPS65950, audio kodek, code compose studio, ALSA, Angstrom, linux, CodeSourcery

Dodatak B

Title:

High accuracy direct digital synthesis using OMAP processors family

Summary:

The goal of this paper is to write a program which will be executing direct digital sinus synthesis from the apointed faze, and start it on *Beagleboard*. For simulation and building of the program for DSP processor, *Code Composer Studio* has been used, and *CodeSourcery* for building a program for ARM prcessor. Optimizations of C kode have been given by using optimization possibilitiess of compieler. Achieved exits of the program for direct digital sinus synthesis are exactly the same as those in the referent *Matlab* code. Achieved exit is 32 bit wide.
Executing a program on *Beagleboard* is done under a strong *Angstrom* (linux) operation sistem. ALSA library has been used for comunication between processor and audio codec. Audio codec needs to be properly configured so that data can be sent to audio exit and therefore be heard on the speaker. When starting a program, uper 16 bits of the sinusoid is sent in the first channel, which is represented as a tone in the speaker, and on the other channel is sent the lower 16 bits of sinusoid which is represented as a noise.

Keywords:

Direct digital synthesis, Hoorners rule, Beagleboard, Texas Instruments, OMAP3530, ARM, Cortex A8, DSP, TM320C64x+, TPS65950, audio codec, Code Composer Studio, ALSA, Angstrom, linux, CodeSourcery

Dodatak C

Glavni kôd za ARM procesor.

Datoteka `main.c`

```
#include <stdio.h>
#include <sys/time.h>
#include <alsa/asoundlib.h>
#include "koef_def.h"
#include "dds.h"

int dds (unsigned int *fa, unsigned int korak);

static char *device = "hw:0,0";
static snd_pcm_format_t format = SND_PCM_FORMAT_S16;
static unsigned int rate = 44100;
static unsigned int channels = 2;
static unsigned int buffer_t = 500000;
static unsigned int period_t = 100000;
static int resample = 0;

static snd_pcm_sframes_t buffer_s;
static snd_pcm_sframes_t period_s;

static void generiraj_sin(const snd_pcm_channel_area_t *areas,
             int count, unsigned int *_fa, unsigned int korak) {

      unsigned int fa = *_fa;
      unsigned char *samples[channels];
      int steps[channels];
      unsigned int chn;
      int format_bits = snd_pcm_format_width(format);
      int bps = format_bits / 8;

      for (chn = 0; chn < channels; chn++) {
            if ((areas[chn].first % 8) != 0) {
                  printf("areas[%i].first == %i, prekidam
                  izvodjenje...\n", chn, areas[chn].first);
                  exit(EXIT_FAILURE);
            }
            samples[chn] = (((unsigned char *)areas[chn].addr) +
                        (areas[chn].first / 8));
```

```
            if ((areas[chn].step % 16) != 0) {
                  printf("areas[%i].step == %i, prekidam
                  izvodjenje...\n", chn, areas[chn].step);
                  exit(EXIT_FAILURE);
            }
            steps[chn] = areas[chn].step / 8;
      }
      while (count-- > 0) {
            int res, i;
            res = dds(&fa, korak);
            for (chn = 0; chn < channels; chn++) {
                  for (i = 0; i < bps; i++) {
                        if (chn % 2)
                              *(samples[chn] + i) =
                              (res >>  ((i+2) * 8)) & 0xff;
                        else
                              *(samples[chn] + i) =
                              (res >>  i * 8) & 0xff;
                  }
                  samples[chn] += steps[chn];
            }
      }
      *_fa = fa;
}
static int set_hwparams(snd_pcm_t *handle,
                              snd_pcm_hw_params_t *params,
                              snd_pcm_access_t access) {

      unsigned int rrate;
      snd_pcm_uframes_t size;
      int err, dir;

      err = snd_pcm_hw_params_any(handle, params);
      if (err < 0) {
            printf("Kriva konfiguracija za sviranje; nema
            dostupne konfiguracije: %s\n", snd_strerror(err));
            return err;
      }
      err = snd_pcm_hw_params_set_rate_resample(handle, params,
      resample);
      if (err < 0) {
            printf("Postavljanje ponovnog uzorkovanja: %s\n",
            snd_strerror(err));
            return err;
      }
```

```
err = snd_pcm_hw_params_set_access(handle, params,
access);
if (err < 0) {
      printf("Tip pristupa nije dostupan: %s\n",
      snd_strerror(err));
      return err;
}
err = snd_pcm_hw_params_set_format(handle, params,
format);
if (err < 0) {
      printf("Format uzorka nije dostupan: %s\n",
      snd_strerror(err));
      return err;
}
err = snd_pcm_hw_params_set_channels(handle, params,
channels);
if (err < 0) {
      printf("Broj kanala (%i) nije dostupan: %s\n",
      channels, snd_strerror(err));
      return err;
}
rrate = rate;
err = snd_pcm_hw_params_set_rate_near(handle, params,
&rrate, 0);
if (err < 0) {
      printf("Brzina uzorkovanja %iHz nije dostupna: %s\n",
      rate, snd_strerror(err));
      return err;
}
if (rrate != rate) {
      printf("Brzina uzorkovanja se ne podudara (zatrazeno
      %iHz, dobiveno %iHz)\n", rate, err);
      return -EINVAL;
}
err = snd_pcm_hw_params_set_buffer_time_near(handle,
params, &buffer_t, &dir);
if (err < 0) {
      printf("Neuspjesno postavljanje vremena spremnika od
      %i: %s\n", buffer_t, snd_strerror(err));
      return err;
}
err = snd_pcm_hw_params_get_buffer_size(params, &size);
if (err < 0) {
      printf("Neuspjesno dohvacanje velicine spremnika:
      %s\n", snd_strerror(err));
      return err;
}
buffer_s = size;
```

```
      err = snd_pcm_hw_params_set_period_time_near(handle,
      params, &period_t, &dir);
      if (err < 0) {
            printf("Neuspjesno postavljanje vremena perioda od
            %i: %s\n", period_t, snd_strerror(err));
            return err;
      }
      err = snd_pcm_hw_params_get_period_size(params, &size,
      &dir);
      if (err < 0) {
            printf("Neuspjesno dohvacanje velicine perioda:
            %s\n", snd_strerror(err));
            return err;
      }
      period_s = size;
      err = snd_pcm_hw_params(handle, params);
      if (err < 0) {
            printf("Neuspjesno postavljanje parametara u sklop:
            %s\n", snd_strerror(err));
            return err;
      }
      return 0;
}

static int set_swparams(snd_pcm_t *handle,
                        snd_pcm_sw_params_t *swparams) {

      int err;

      err = snd_pcm_sw_params_current(handle, swparams);
      if (err < 0) {
            printf("Ne mogu dohvatiti trenutne sw postavke:
            %s\n", snd_strerror(err));
            return err;
      }

      err = snd_pcm_sw_params_set_start_threshold(handle,
      swparams, (buffer_s / period_s) * period_s);
      if (err < 0) {
            printf("Ne mogu postaviti prag za spremnik: %s\n",
            snd_strerror(err));
            return err;
      }
      err = snd_pcm_sw_params_set_avail_min(handle, swparams,
      period_s);
```

```
	if (err < 0) {
		printf("Ne mogu postaviti avail_min: %s\n",
		snd_strerror(err));
		return err;
	}
	err = snd_pcm_sw_params(handle, swparams);
	if (err < 0) {
		printf("Ne mogu postaviti sw parametre: %s\n",
		snd_strerror(err));
		return err;
	}
	return 0;
}

static int xrun_recovery(snd_pcm_t *handle, int err) {

	if (err == -EPIPE) {
		err = snd_pcm_prepare(handle);
		if (err < 0)
			printf("Ne mogu se oporaviti od \"underrun\"
			greske: %s\n", snd_strerror(err));
		return 0;
	} else if (err == -ESTRPIPE) {
		while ((err = snd_pcm_resume(handle)) == -EAGAIN)
			sleep(1);
		if (err < 0) {
			err = snd_pcm_prepare(handle);
			if (err < 0)
				printf("Ne mogu se oporaviti od
				\"suspend\" greske: %s\n",
				snd_strerror(err));
		}
		return 0;
	}
	return err;
}

static int salji_audio(snd_pcm_t *handle,
			signed short *samples,
			snd_pcm_channel_area_t *areas) {

	unsigned int fa = 96843953;
	unsigned int korak = 96843953;
	signed short *ptr;
	int err, cptr;
	while (1) {
		generiraj_sin(areas, period_s, &fa, korak);
```

```
            ptr = samples;
            cptr = period_s;
            while (cptr > 0) {
                  err = snd_pcm_writei(handle, ptr, cptr);
                  if (err == -EAGAIN) {
                        printf("Saljem ponovo...\n");
                        continue;
                  }
                  if (err < 0) {
                        if (xrun_recovery(handle, err) < 0) {
                              printf("Greska kod pisanja: %s\n",
                              snd_strerror(err));
                              exit(EXIT_FAILURE);
                        }
                                    printf("Preskocit cu jedan
                                    period... err = %s\n",
                                    snd_strerror(err));
                        break;
                  }
                  ptr += err * channels;
                  cptr -= err;
            }
      }
}

int main (void) {

      int i, err;
      unsigned int fa, korak;
      snd_pcm_t *handle;
      snd_pcm_sframes_t frames;

      snd_pcm_hw_params_t *hwparams;
      snd_pcm_sw_params_t *swparams;
      signed short *samples;
      unsigned int chn;

      snd_pcm_channel_area_t *areas;
      snd_pcm_hw_params_alloca(&hwparams);
      snd_pcm_sw_params_alloca(&swparams);

      printf("Uredjaj is %s\n", device);
      printf("Parametri: %iHz, %s, %i kanala\n", rate,
      snd_pcm_format_name(format), channels);
```

```
if ((err = snd_pcm_open(&handle, device,
      SND_PCM_STREAM_PLAYBACK, 0)) < 0) {
      printf("Greska kod otvaranja: %s\n",
      snd_strerror(err));
      return 0;
}

if ((err = set_hwparams(handle, hwparams,
      SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
      printf("Postavljanje hw parametara: %s\n",
      snd_strerror(err));
      exit(EXIT_FAILURE);
}

if ((err = set_swparams(handle, swparams)) < 0) {
      printf("Postavljanje sw parametara: %s\n",
      snd_strerror(err));
      exit(EXIT_FAILURE);
}

samples = malloc((period_s * channels *
snd_pcm_format_physical_width(format)) / 8);
if (samples == NULL) {
      printf("Nema dovoljno memorije!\n");
      exit(EXIT_FAILURE);
}

areas = calloc(channels, sizeof(snd_pcm_channel_area_t));
if (areas == NULL) {
      printf("Nema dovoljno memorije!\n");
      exit(EXIT_FAILURE);
}
for (chn = 0; chn < channels; chn++) {
      areas[chn].addr = samples;
      areas[chn].first = chn *
            snd_pcm_format_physical_width(format);
      areas[chn].step = channels *
            snd_pcm_format_physical_width(format);
}

printf("Period size = %d\n", period_s);
printf("Buffer size = %d\n", buffer_s);

err = salji_audio(handle, samples, areas);
if (err < 0)
      printf("Neuspjesan prijenos: %s\n",
      snd_strerror(err));
```

```
        free(areas);
        free(samples);
        snd_pcm_close(handle);

        return 0;
}
```