Projekt

**Efekt pucnja**

Studenti:

Krešimir Tomić i Tomislav Gelešić

Sadržaj

[1. Uvod 1](#_Toc41990057)

[1.1. Zadatak 1](#_Toc41990058)

[2. Sinteza, projekcija i simulacija zvuka pucnja – MATLAB 2](#_Toc41990059)

[2.1. Sinteza zvuka 2](#_Toc41990060)

[2.1.1. Bijeli šum 2](#_Toc41990061)

[2.1.2. Dizajn filtera i filtriranje 3](#_Toc41990062)

[2.1.3. ADSR omotnica 4](#_Toc41990063)

[2.1.4. Projekcija zvuka i MATLAB grafovi 6](#_Toc41990064)

[3. C6713 DSK 8](#_Toc41990065)

[3.1. TMS320C6713 8](#_Toc41990066)

[3.2. Konfiguracijski prekidači 9](#_Toc41990067)

[4. Implementacija na razvojni sustav 10](#_Toc41990068)

[4.1. Code Composer Studio 10](#_Toc41990069)

[4.2. Programski kôd 13](#_Toc41990070)

[4.2.1. Biblioteke 13](#_Toc41990071)

[4.2.2. Konstante 14](#_Toc41990072)

[4.2.3. Varijable i tipovi podataka 14](#_Toc41990073)

[4.2.4. Funkcije 17](#_Toc41990074)

[4.2.5. Glavni program 21](#_Toc41990075)

[5. Izrada makete pištolja 24](#_Toc41990076)

[5.1. Ideja 24](#_Toc41990077)

[5.2. Komponente i njihove uloge 24](#_Toc41990078)

[6. Prilog 28](#_Toc41990079)

[30](#_Toc41990080)

[31](#_Toc41990081)

[7. Literatura 32](#_Toc41990082)

# Uvod

Ovim seminarskim radom detaljnije je pojašnjen projekt u kojem je sintetiziran zvuk pucnja pištolja uz dodatne efekte. Prvotno je razvijeni algoritam simuliran u MATLAB okruženju te je potom isti implementiran na C6713 razvojnom sustavu. Svaki korak razvoja i implementacije je popraćen komentarima, slikama i programskim kôdom. Sintetiziran algoritam zvuka je najprije testiran u MATLAB razvojnom okruženju, a nakon toga su algoritam za zvuk, korišteni filtri i funkcionalnost sklopki isprogramirani u programskom jeziku C kako bi upravljali razvojnim sustavom C6713 DSK i svim komponentama projekta. Programski kôd omogućuje preko sklopki korisniku odabir efekata rafalne i pojedinačne paljbe, odabir udaljenosti izvora zvuka od korisnika te omogućuje komunikaciju s perifernim uređajem – zvučnikom, na kojem se čuje zvuk pucnja. Zbog nedostatka funkcionalnog 3D printera, za maketu oružja iskorišten je stariji model konzole (u obliku pištolja) za arkadne video igre. U plastičnu konzolu dodane su sklopke (okidač i ostale koje omogućuju odabir udaljenost i tip paljbe). Konzola i zvučnici su žično povezani preko C6713 DSK sustava.

## Zadatak

U MATLAB okruženju potrebno je razviti i testirati algoritam koji sintetizira zvuk pucnja oružja. Nakon testiranja u MATLAB okruženju isti efekt implementirati na C6713 DSK razvojnom sustavu. Omogućiti rafalnu i pojedinačnu paljbu te kontrolu udaljenosti. Dizajnirati 3D oružje koje će omogućiti žičnu kontrolu rada. Isti izraditi na 3D pisaču. U seminaru potrebno je detaljno opisati razvijeni MATLAB kôd, korake izrade projekta, komentirati pojedine dijelove kôda, te opisati način implementacije/testiranja kôda (vidjeti primjere predložaka LV-a). Također, potrebno je napraviti (max. 10 min) prezentaciju s kojom će studenti prezentirati svoj projekt prilikom ocjenjivanja.

# Sinteza, projekcija i simulacija zvuka pucnja – MATLAB

„MATLAB je programski jezik visoke razine i interaktivna je okolina za numeričko računanje, matrično računanje, vizualizaciju podataka i programiranje. Naziv je nastao kao kratica od engleskih riječi '*MATrix LABoratory'*.”

„Pomoću MATLAB-a mogu se analizirati podaci, izraditi algoritmi te stvoriti modeli i aplikacije. Jezik, alati i matematičke funkcije omogućuju brži rad nego s tablicama ili tradicionalnim programskim jezicima, kao što su C/C++ ili Java. MATLAB se može koristiti za niz aplikacija, uključujući obradu signala i komunikacija, obradu sustava kontrole, ispitivanja i mjerenja, . . . ” [1] [2]

## Sinteza zvuka

### Bijeli šum

Za potrebe sinteze zvuka prvotno su generirani nasumični brojevi u rasponu [-1,1] koji predstavljaju bijeli šum te je odabrana frekvencija uzorkovanja od 24kHZ. Frekvencija uzorkovanja nam govori kojom će se brzinom puštati uzorci zvuka na zvučnik. Vrijednosti nasumičnih brojeva spremljene su u vektor (sound\_values) koji predstavlja zapis šuma. Vidi kôd 1 ispod. Varijabla *fire\_lenght* predstavlja duljinu trajanja reprodukcije samo vrijednosti zvuka koje predstavljaju pucanj, dok *sound*\_length predstavlja duljinu zvuka pucnja zajedno s „tišinom“ nakon istog. Kada bi sintetiziran zvuk sadržavao isključivo vrijednosti koje ljudsko uho čuje (pucanj), rafalna paljba ne bi zvučala vjerodostojno jer bi se samo nastavljao zvuk na zvuk bez „tišine“ između pucnjeva.

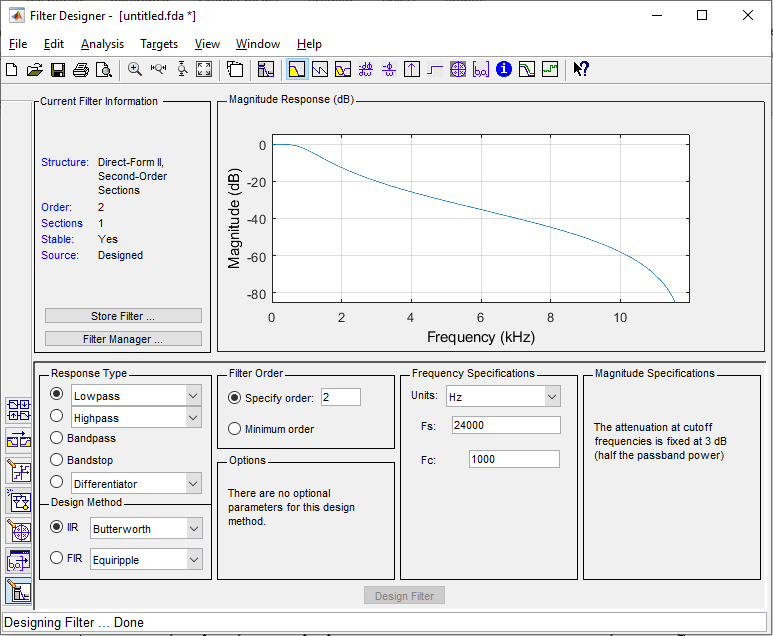
|  |  |
| --- | --- |
| Linija | Kôd |
| 1: | Fs = 24000; % fs = 24 kHz |
| 2: | sound\_length = 1; % t = 1 s |
| 3: | number\_of\_sound\_values = sound\_length \* Fs; |
| 4: | sound\_values = 2 .\* rand(number\_of\_sound\_values, 1) - 1; |
| 5: | fire\_length = sound\_length – 0.6; |

Kôd 1. Bijeli šum

### Dizajn filtera i filtriranje

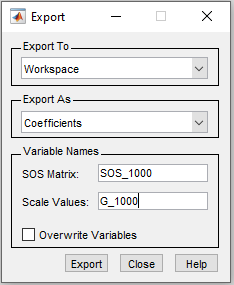
Nakon generiranja nasumičnih vrijednosti iste je bilo potrebno filtrirati. Za potrebe filtriranja dizajnirani su nisko-propusni filtri beskonačnog impulsnog odziva (IIR). Za dizajn filtra korišten je „*Filter Designer*“ alat u MATLAB-u te je dizajnirano 5 različitih filtera koji služe za simulaciju zvuka pucnja s različitih udaljenosti. Dizajnirani filtri razlikuju se samo u iznosu granične frekvencije (eng. cut-off frequency).

Kao što je već ranije rečeno potrebno je dizajnirati nisko-propusni filtar što znači da filtar propušta sve frekvencije niže od granične frekvencije (Fc), a više frekvencije guši. Nadalje, odabran je IIR Butterworth filtar drugog reda koji je specifičan po tome da maksimalno „pegla“ frekvencijski odziv u pojasu frekvencija koje filtar propušta (eng. passband). Na posljetku u „*Frequency Specifications*“ potprozoru odabrana je frekvencija uzorkovanja (Fs) te granična frekvencija (Fc) koja će se razlikovati od filtra do filtra. Vrijednosti korištenih graničnih frekvencija u ovom projektu su : 400Hz, 550Hz, 700Hz, 850Hz i 1000Hz. Postavke filtera su određene testiranjem uz referenciranje na BBC-ev „Gunfire Effects Generator“ [3]. Za postavke filtra vidi sliku 1.



Slika . Postavke filtra u MATLAB alatu - Filter Designer

Nakon što je filtar dizajniran potrebno ga je izvesti (eng. export) u obliku koeficijenata u radno okruženje (eng. workspace) klikom na '*File*' te zatim '*Export*'. Također, potrebno je dati specifična imena iz kojih će se moći iščitati o kojoj je graničnoj frekvenciji riječ (npr. SOS\_1000, G\_1000). Vidi sliku 2 ispod.



Slika . Izvoz parametara filtra

Zatim je bilo potrebno generirati prijenosnu funkciju uz pomoć *sos2tf* funkcije čija se povratna vrijednost sprema u dva vektora koji predstavljaju brojnik i nazivnik prijenosne funkcije. Kao argumenti *sos2tf* funkcije predaju se prethodno izvezeni koeficijenti ovisno o filtru koji želimo primijeniti (npr. SOS\_1000 i G\_1000). Ukoliko se koeficijenti ne nalaze u radnom okruženju (eng. workspace) potrebno ih je učitati naredbom *load*. Generirani vektori predaju se funkciji *filter* uz 24000 nasumično generiranih vrijednosti, a povratna vrijednost funkcije pohranjuje se u varijablu *sound\_values\_fc1000* koja predstavlja podatke filtriranog šuma filtrom s graničnom frekvencijom od 1000Hz. Vidi kôd 2 ispod.

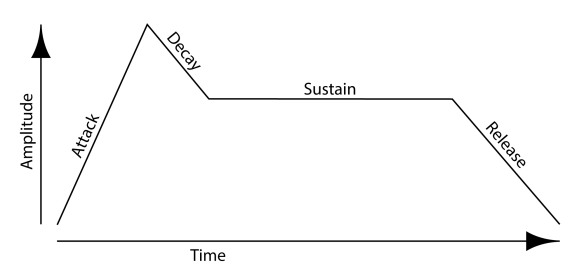
|  |  |
| --- | --- |
| Linija | Kôd |
| 6: | % load('SOS\_1000.mat'); |
| 7: | % load('G\_1000.mat'); |
| 8: | [b\_coeff, a\_coeff] = sos2tf(SOS\_1000,G\_1000); |
| 9: | sound\_values\_fc1000 = filter(b\_coeff,a\_coeff,sound\_values); |

Kôd 2. Prijenosna funkcija filtra

### ADSR omotnica

Kako bi dobiveni filtrirani šum oblikovali u zvuk pucnja bilo je potrebno proučiti kako treba izgledati odziv zvuka pucnja. Zvuk pucnja ima specifičan odziv, to jest nagli porast amplitude zvuka te postupno smanjenje. Navedeni odziv dobiven je korištenjem ADSR omotnice nad filtriranim šumom.

ADSR je skraćenica od „Attack“, „Decay“, „Sustain“ i „Release“ dijelova omotnice. Omotnica se može shvatiti kao predložak koji služi za obrezivanje željenog signala. Vidi graf 1 ispod.



Graf . ADSR omotnica

Željeni rast i pad dobili su se međusobnim množenjem redaka ADSR omotnice i šuma, a ADSR omotnica je generirana pomoću *linespace* funkcije. Za početak potrebno je odrediti koliko će zasebno vremenski trajati „Attack“, „Decay“, „Sustain“ i „Release“ dijelovi i na kojim amplitudama počinju te završavaju. Testiranjem je utvrđeno da „Attack“ i „ Decay“ dijelovi trebaju imati zasebno po 480 od 24000 vrijednosti zvuka što je 2% cjelokupnog trajanja zvuka, „Sustain“ dio ima 0 vrijednosti te se on ne uračunava u omotnicu, a „Release“ dio ima 8640 vrijednosti zvuka što je 36% cjelokupnog trajanja pucnja. Granične amplitude omotnice korištene su kao i u BBC-evom primjeru ovog efekta („Attack“: 0 - 1, „Decay“: 1 - 0.3 i „Release“: 0.3 - 0) [3].

Funkcija *linespace* ravnomjerno raspoređuje vrijednosti u danim granicama te naposljetku dobiveni vektori linearno rastu ili padaju. Nakon što su određeni vektori za pojedine dijelove omotnice vektori se nadovezuju jedan na drugoga tako da tvore jedan *ADR* vektor (bez „Sustain“ dijela). Omotnica u tom trenutku ima 9600 vrijednosti što je 40% ukupnog signala koji trebamo generirati. Zbog nemogućnosti množenja vektora različitih veličina potrebno je povećati broj vrijednosti u omotnici tako što će se preostalih 60% signala ispuniti nulama koje će predstavljati tišinu nakon pucnja. Na posljetku se šum i omotnica množe redak s retkom te je time završeno oblikovanje pucnja. Vidi kôd 3 ispod.

|  |  |
| --- | --- |
| Linija | Kôd |
| 10: | % ADSR envelope |
| 11: | A = linspace(0, 1, int32(0.05\*Fs\*fire\_length)); % 480 values |
| 12: | D = linspace(1, 0.3, int32(0.05\*Fs\*fire\_length)); %480 values |
| 13: | % S = not used |
| 14: | R = linspace(0.3, 0, int32(0.9\*Fs\*fire\_length)); % 8640 values |
| 15: | ADSR = [A, D, R]; |
| 16: | envelopeADSR = zeros(24000,1); |
| 17: | envelopeADSR(1:length(ADSR)) = ADSR; |
| 18: | gunfire\_sound = sound\_values\_fc1000 .\* envelopeADSR; |

Kôd 3. ADSR omotnica

### Projekcija zvuka i MATLAB grafovi

Funkcijom *sound* moguće je reproducirati zvuk pucnja na zvučnike, a kao argumenti se predaju vrijednosti koje su ranije oblikovane te frekvencija uzorkovanja. Funkcija plot grafički prikazuje signal zvuka. Vidi kôd 4 ispod.

|  |  |
| --- | --- |
| Linija | Kôd |
| 19: | % play sound |
| 20: | sound(gunfire\_sound, Fs); |
| 21: | plot(gunfire\_sound); |

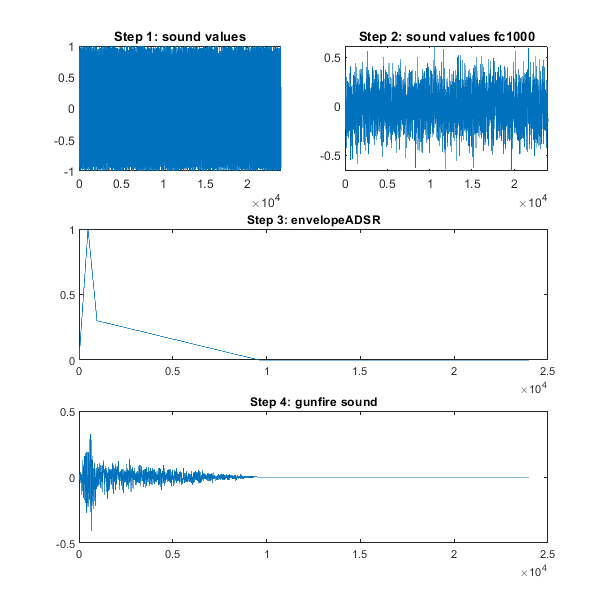
Kôd 4. Projekcija zvuka i grafa

Efekt udaljenosti može se testirati promjenama SOS i G koeficijenata, a rafalna paljba na način da se funkcija *sound* poziva u for petlji sa određenim vremenskim kašnjenjima. Vidi kôd 5 ispod.

|  |  |
| --- | --- |
| Linija | Kôd |
| 19: | % play rafal sound |
| 20: | for i=0:1:20 |
| 21: | pause(0.1) |
| 22: | if bitget(i,2) |
| 23: | sound(out, Fs); |
| 24: | end |
| 25: | end |

Kôd 5. Projekcija zvuka i grafa

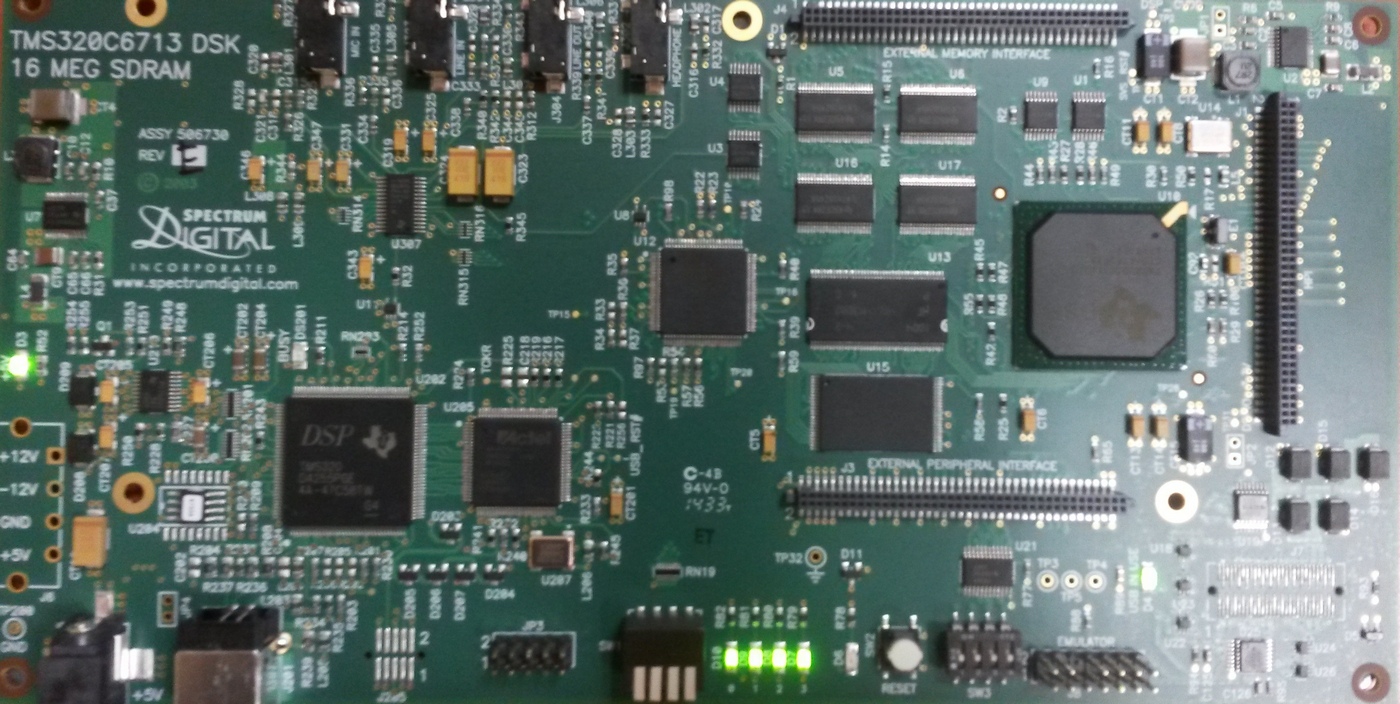
Na slici 3 prikazan je izgled pojedinih faza oblikovanja od šuma (*sound\_values*) do rezultantnog zvuka pucnja (*gunfire\_sound*). Osobito se može primijetiti utjecaj omotnice na filtrirani šum te kako ona oblikuje signal.



Slika . Grafički prikaz razvoja zvuka pucnja

# C6713 DSK

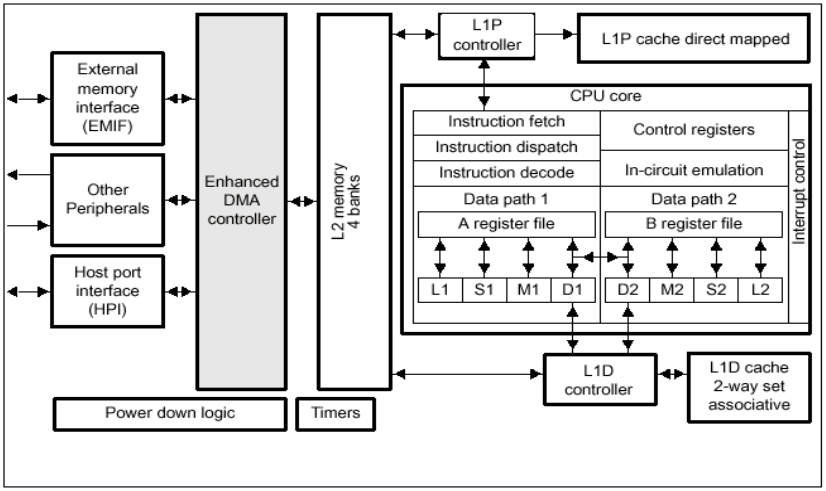
Razvojni sustav i komponente (skraćenog naziva C6713 DSK) korišteni u projektu su rezultat suradnje dviju tvrtki *Texas Instruments* i *Spectrum Digital*. Svrha sustava je obrada različitih signala (npr. obrada audio signala, telekomunikacijskih signala, itd.). Razvojni sustav sadrži i SDRAM modul od 16 MB, 512k *Flash* memoriju i JTAG *debug* sučelje te je kompatibilan s razvojnim okruženjem (*IDE*) *Code Composer Studio*.



Slika . C6713 DSK platforma

## TMS320C6713

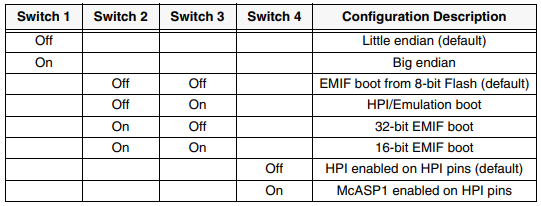
TMS320C6713 je 32-bitni procesor s pomičnim zarezom baziran na VLIW arhitekturi. Sadrži 8 funkcionalnih jedinica koje se mogu koristiti u paraleli. Procesor može raditi sa signalom takta od 120, 150 ili 167 MHz te na maksimalnoj frekvenciji omogućava performanse do 1GFLOPS. TMS320C6713 sadrži 32 registra opće namjene koji mogu biti korišteni od strane svih funkcionalnih jedinica. Uz registre sadrži 1MB On-Chip SRAM memorije i 52 MB vanjske memorije ( SDRAM, SRAM, EEPROM...) te sva ta memorija može biti korištena od strane procesora. Memoriji se može pristupiti od strane 4-kanalnog DMA upravljača.



Slika . VLIW arhitektura

## Konfiguracijski prekidači

C6713 DSK sadrži 4 konfiguracijska prekidača koja omogućuju kontrolu operacijskog stanja DSP-a. Prvi konfiguracijski prekidač kontrolira poredak bajtova (eng. endianness), dok drugi i treći prekidač konfiguriraju način pokretanja koji će se koristiti kada DSP počne s izvođenjem. Konfiguracijski prekidač četiri kontrolira „on-chip“ multipleksiranje HPI i McASP signala te time omogućuje korištenje dodatnih GPO signala. Zadana konfiguracija je da su svi prekidači isključeni što korespondira sa EMIF načinom pokretanja u „little endian“ načinu rada sa HPI signalima na HPI priključku za proširenja.



Slika . Konfiguracijska tablica

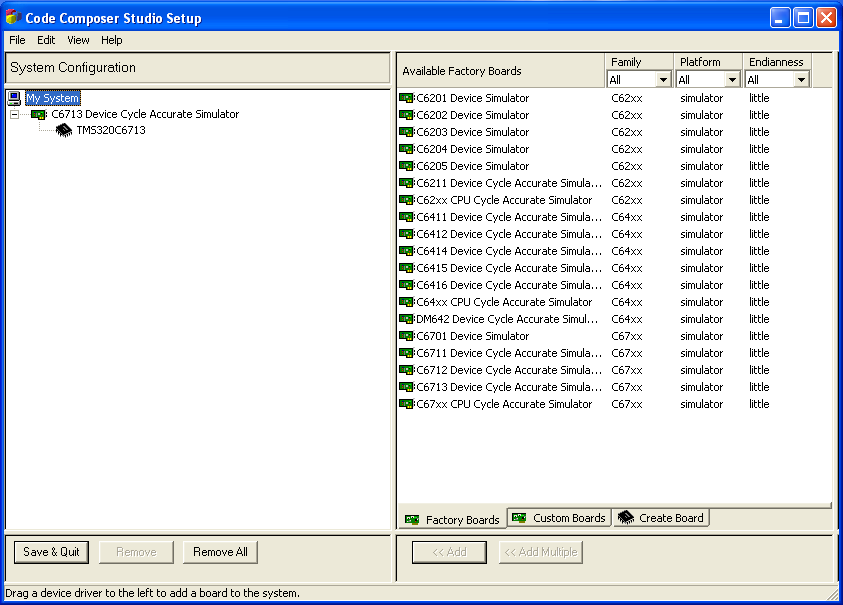
# Implementacija na razvojni sustav

## Code Composer Studio

Code Composer Studio je besplatno integrirano razvojno okruženje (IDE) koje služi razvoju aplikacija za procesore koji su razvijeni od strane tvrtke Texas Instruments. U ovom projektu se isključivo koristilo navedeno okruženje za razvoj programskog koda. Vrlo važno je ispravno konfigurirati razvojno okruženje te je taj postupak detaljno opisan u ovom poglavlju.

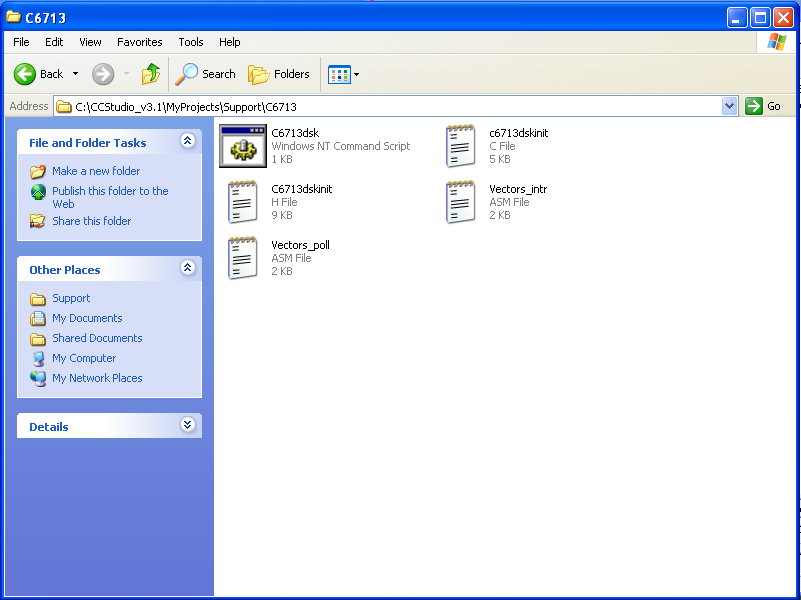
Razvojni sustav TMS320C6713 DSK potrebno je priključiti na napajanje pomoću adaptera te isto tako USB kabelom na jedan od slobodnih USB portova na računalu. Zatim je potrebno provjeriti je li razvojni sustav ispravno priključen. Provjerava se nakon što sustav izvrši Boot program, a to možemo prepoznati ako svijetle 4 LE diode na razvojnom sustavu. Provjera se vrši pomoću „6713 DSK Diagnostics Utility“ alata koji provodi određene testove i shodno tome vraća povratnu informaciju je li sustav ispravno ili neispravno spojen. Nakon što je sustav ispravno spojen zatvoriti „6713 DSK Diagnostics Utility“ alat.

Sljedeći korak je pokretanje „CCS v3.1 Setup“ te dodavanje razvojnog sustava C6713 koji je ponuđen u potprozoru „*Available Factory Boards*“. Nakon dodavanja sustav bi se trebao pojaviti u potprozoru „*System Configuration*“. Ako je sustav već bio dodan može se zanemariti ovaj korak. Vidi sliku 7.



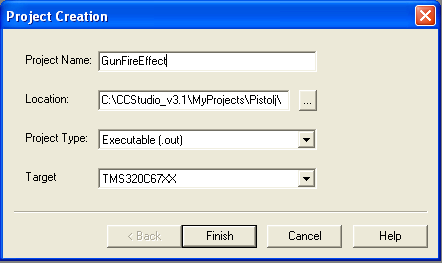
Slika . CCS Setup

Nadalje, izrađena je mapa „*Support*“ na putanji gdje je instaliran CCStudio, u ovom slučaju to je na putanji *C:\CCStudio\MyProjects*. U novonastalu mapu „*Support*“ potrebno je otpakirati „*Support datoteke C6713 za CCS\_V3.1*“ koje se nalaze na Loomen stranici kolegija Algoritmi i arhitektura DSP-a [4]. Vidi sliku 8.



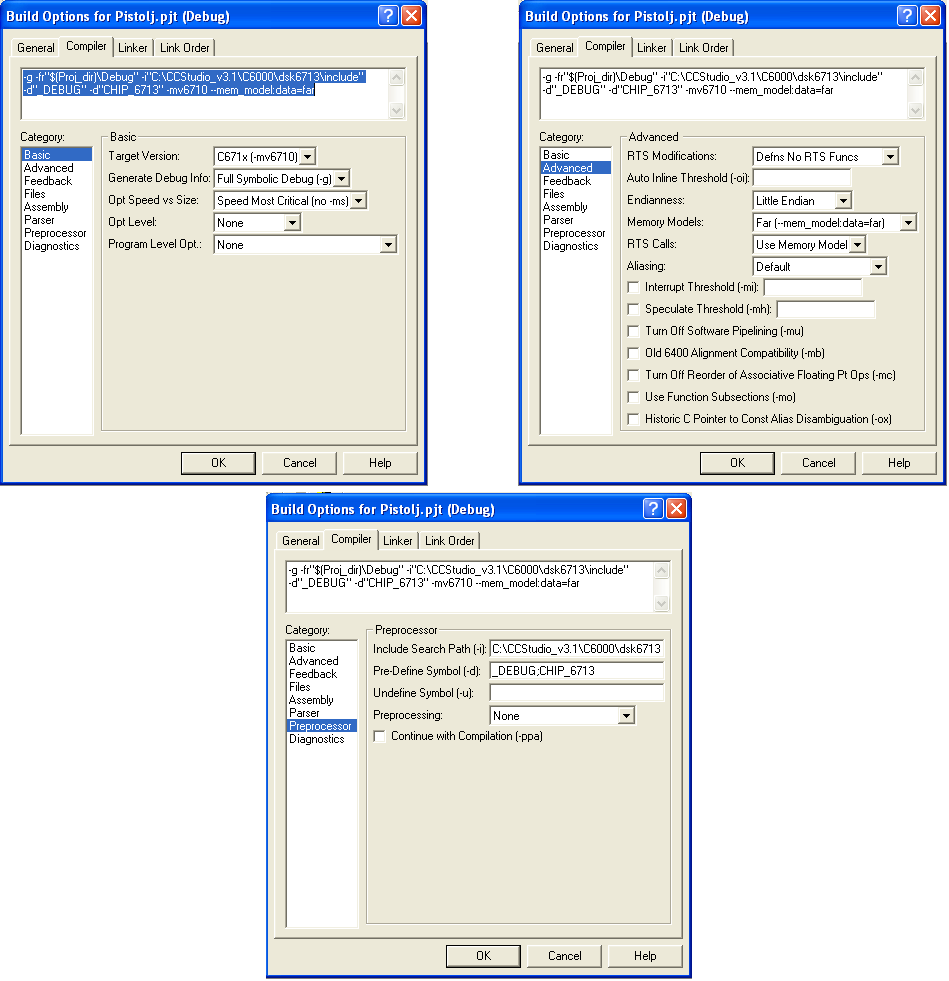
Slika . Support datoteke

Sljedeći korak je kreiranje projekta. Potrebno je kliknuti dvostrukim klikom na „CCStudio 3.1“, zatim u padajućem izborniku kliknuti na „New“ te odabrati postavke projekta kao na slici. Vidi sliku 9. U projekt je potrebno dodati *Support* datoteke koje postavljaju prekidne vektore (Vectors\_poll.asm) i povezuju pojedine dijelove memorije (C6713dsk.cmd) te biblioteke razvojnog sustava (rts6700.lib, dsk6713bsl.lib i csl6713.lib). Potrebne datoteke su dodane na način da se klikne na padajući izbornik „*Project*“ te je odabrano „*Add Files to Project*“(sve datoteke će biti vidljive ako se u „*Files of Type* odabere „*All Files*“).

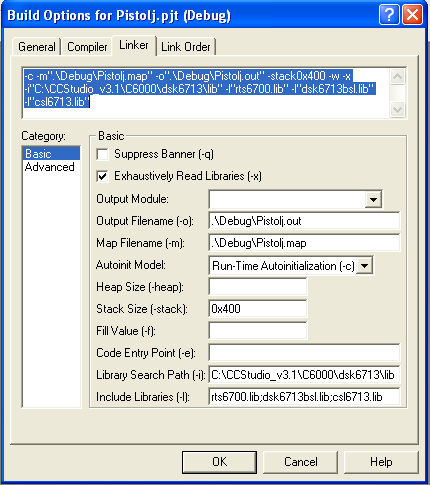


Slika . Postavke projekta

Zatim je potrebno kliknuti na „*Build Options*“ u *Project* padajućem izborniku. U tabu „*Compiler*“ pod kategorijom „*Basic*“ , „*Advanced*“ i „*Preprocessor*“ namjestiti postavke kao na slici. Vidi sliku 10. Zatim, u tabu „*Linker*“ pod kategorijom „*Basic*“ također namjestiti postavke kao na slici. Vidi sliku 11.



Slika 10. "Compiler" postavke



Slika 11. "Linker" postavke

## Programski kôd

U ovom poglavlju će se objasniti svi dijelovi koda te njihove funkcije. Kod će se razlomiti na dijelove zbog preglednosti i mogućnosti paralelnog čitanja seminara i programiranja.

Napomena: U prikazu kôda nekad će biti praznih linija, zbog lakše snalažljivosti u prilogu na kraju seminara te kako bi ovaj rascjepkani kod bio identičan kodu i prilogu.

### Biblioteke

Navedeno je 8 biblioteka i sve su prijeko potrebne za izvedbu projekta osim „*dsk6713\_dip.h*“. Ova biblioteka je korištena za mogućnost testiranja pucnjeva i rafalne paljbe bez makete pištolja to jest uz pomoć DIP prekidača. U daljnjem objašnjenju koda navest će se gdje bi se mogla iskoristiti „dsk6713\_dip.h“ biblioteka. Važno je napomenuti da zadnje 4 biblioteke omogućuju promjene stanja određenih registara kako bi se omogućili dodatni GPIO pinovi. Više o tome kasnije u seminaru. Vidi kôd 6 ispod.

|  |  |
| --- | --- |
| Linija | Kod |
| 1: | #include "dsk6713\_aic23.h"//codec podrška |
| 2: | #include <dsk6713\_dip.h> //dip prekidači–tesitranje pucnja |
| 3: | #include <stdlib.h> //potrebno za srand() |
| 4: | #include <time.h> //potrebno za time() |
| 5: | #include <dsk6713.h> //GPIO |
| 6: | #include <csl\_gpio.h> //GPIO |
| 7: | #include <csl\_gpiohal.h> //GPIO |
| 8: | #include <csl\_irq.h> //GPIO |

Kôd 6. Uključivanje biblioteka

### Konstante

Definirane su dvije konstante. Konstanta *GAIN* služi za pojačanje krajnjih vrijednosti signala kako bi se pojačao zvuk na zvučnicima. Konstanta *DSK6713\_AIC23\_INPUT\_LINE* sadrži heksadecimalni zapis koji omogućuje korištenje zvučnika. Vidi kôd 7 ispod.

|  |  |
| --- | --- |
| Linija | Kod |
| 9: |  |
| 10: | #define GAIN 10 |
| 11: | #define DSK6713\_AIC23\_INPUT\_LINE 0x0011 |

Kôd 7. Definiranje konstanti

### Varijable i tipovi podataka

* Deklarirana je globalna varijabla „*inputsource*“ koja određuje tip ulaza na razvojnoj ploči. Vidi liniju kôda 14.
* Deklarirana je globalna varijabla „*fs*“ koja određuje frekvenciju uzorkovanja od 24kHz. Vidi liniju kôda 15.
* Globalno su deklarirana dva polja:
  + - „*xInArray*“ služi za pamćenje zadnja 3 slučajno generirana broja
    - „*yOutArray*“ služi za pamćenje zadnje dvije filtrirane vrijednosti

Vidi liniju kôda 16.

* Deklarirana je globalna varijabla „*yOut*“ u koju se spremaju filtrirane vrijednosti. Vidi liniju kôda 17.
* Deklarirana je globalna varijabla „*randomNumber*“ u koju se sprema nasumično generirani broj u rasponu -1 do 1. Vidi liniju kôda 18.
* Deklarirana je globalna varijabla „*yOutShort*“ u koju se spremaju filtrirane vrijednosti, ali u *short* tipu podatka. Vidi liniju kôda 19.
* Deklarirana je globalna varijabla „*delay*“ koja predstavlja brojač potreban za određeno čekanje programa, a koristi se u *for* petlji. Vidi liniju kôda 20.
* Deklarirane su globalne varijable „*nA*“, „*nD*“ i „*nR*“ koje predstavljaju duljinu pojedinog dijela omotnice. Vidi linije kôda 21-23.
* Deklarirane su globalne varijable „*aADR*“, „*dADR*“ i „*rADR*“ koje predstavljaju početne amplitude za pojedini dio ADSR omotnice. Vidi linije kôda 24-26.
* Globalno je deklarirano 2D-polje „*filterCoefficientsMatrix*“ koje sadrži koeficijente pojedinih filtera ovisno o graničnoj frekvenciji. Vidi linije kôda 28-34.
* Globalno je deklarirano polje „*gainArray*“ koje sadrži koeficijente pojačanja za pojedini filter ovisno o graničnoj frekvenciji. Vidi linije kôda 35-36.
* Globalno su deklarirane varijable:
  + - „*previous\_state*“ služi za pamćenje prošlog stanja okidača
    - „*current\_state*“ služi za pamćenje trenutnog stanja okidača
    - „*flagRisingEdge*“ uz pomoć prethodne dvije varijable služi za detekciju rastućeg brida

Vidi linije kôda 37-39.

* Globalno je deklarirana unija „*AIC\_data*“ koja sadrži podatak koji treba pustiti na zvučnike te polje koje pomoću kojega se zvuk pušta na desni i lijevi zvučnik. Vidi linije kôda 41-44.
* Deklarirana je globalna varijabla „*gpio\_handle*“ koja upravlja GPIO pinovima. Vidi liniju kôda 46.
* Linije kôda 50-58 konfiguriraju registre kako bi se omogućilo upravljanje GPIO pinovima.
  + - *gpgc* - GPIO Global Control Register
    - *gpen* - GPIO Enable Register
    - *gdir* - GPIO Direction register value
    - *dpval* - GPIO Value register value
    - *dphm* - GPIO High Mask register value
    - *dplm* - GPIO Low Mask register value
    - *gppol* - GPIO Interrupt Polarity register value

Vidi linije kôda 48-56. Vidi dokumentaciju [5].

* Deklarirane su globalne varijable „*i*“ i „*j*“ koje služe kao brojači za *for* petlje. Vidi liniju kôda 58.

|  |  |
| --- | --- |
| Linija | Kod |
| 12: | //VARIABLES |
| 13: |  |
| 14: | Uint32 inputsource=DSK6713\_AIC23\_INPUT\_LINE; |
| 15: | Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; |
| 16: | float xInArray[3],yOutArray[2]; |
| 17: | float yOut; |
| 18: | float randomNumber; |
| 19: | short yOutShort; |
| 20: | int delay; |
| 21: | short nA=480; |
| 22: | short nD=480; |
| 23: | short nR=8640; |
| 24: | float aADR=0.0; |
| 25: | float dADR=1.0; |
| 26: | float rADR=0.3; |
| 27: |  |
| 28: | float filterCoefficientsMatrix[5][2] = { |
| 29: | {-0.6905989232, 1.6329931619}, //1000 |
| 30: | {-0.7300190220, 1.6873607674}, //850 |
| 31: | {-0.7716989200, 1.7420316578}, //700 |
| 32: | {-0.8157642007, 1.7969735593}, //550 |
| 33: | {-0.8623486260, 1.8521464854}, //400 |
| 34: | }; |
| 35: | float gainArray[5] = {69.43749902, 93.76848728, |
| 36: | 134.8287542, 212.8719253, 392.0745796}; |
| 37: | char previous\_state = 1; |
| 38: | char current\_state= 1; |
| 39: | char flagRisingEdge=0; |
| 40: |  |
| 41: | union AIC\_data{ |
| 42: | Uint32 data; |
| 43: | short channel[2]; |
| 44: | }aicSample; |
| 45: |  |
| 46: | GPIO\_Handle gpio\_handle; |
| 47: |  |
| 48: | GPIO\_Config gpio\_config = { |
| 49: | 0x00000000, // gpgc |
| 50: | 0x0000FF00, // gpen |
| 51: | 0x00000000, // gdir |
| 52: | 0x00000000, // gpval |
| 53: | 0x00000000, // gphm |
| 54: | 0x00000000, // gplm |
| 55: | 0x00000000 // gppol |
| 56: | }; |
| 57: |  |
| 58: | short i=0,j=0; //"for" counter |

Kôd 8. Varijable i tipovi podataka

### Funkcije

* *void addNewXin(float)*
  + - funkcija prima slučajno generirani decimalni broj u rasponu [-1,1] te vrši zamjenu elemenata uz pomoć dvije privremene varijable koje pamte zadnji i predzadnji slučajno generirani broj, a na kraj polja dodaje novi
    - funkcija ne vraća ništa
    - namjena funkcije je da polje „*xInArray*“ uvijek sadrži posljednja 3 slučajno generirana broja

Vidi kôd 9.

|  |  |
| --- | --- |
| Linija | Kod |
| 60: | void addNewXin(float randomNumber){ |
| 61: | float temp1=xInArray[1]; |
| 62: | float temp2=xInArray[2]; |
| 63: | xInArray[0]=temp1; |
| 64: | xInArray[1]=temp2; |
| 65: | xInArray[2]=randomNumber; |
| 66: | //return xInArray; |
| 67: | } |

Kôd 9. Funkcije - addNewXin

* *void addNewYout(float)*
  + - funkcija prima posljednju filtriranu vrijednost te je stavlja na kraj polja, a prethodnu vrijednost pomiče na prvo mjesto u polju
    - funkcija ne vraća ništa
    - namjena funkcije je da polje „yOutArray“ uvijek sadrži posljednje dvije filtrirane vrijednosti

Vidi kôd 10.

|  |  |
| --- | --- |
| Linija | Kod |
| 70: | void addNewYout(float yOut){ |
| 71: | float temp1=yOutArray[1]; |
| 72: | yOutArray[0]=temp1; |
| 73: | yOutArray[1]=yOut; |
| 74: | //return yOutArray; |
| 75: | } |

Kôd 10. Funkcije - addNewYout

* *void initXinYout()*
  + - *funkcija ne prima ništa*
    - *funkcija ne vraća ništa*
    - *namjena funkcije je da popuni polja koja služe za filtriranje nulama jer se za svaki novi pucanj polja moraju resetirati kako bi filtriranje bilo uspješno*

Vidi kôd 11.

|  |  |
| --- | --- |
| Linija | Kod |
| 77: | void initXinYout(){ |
| 78: | for(i=0;i<3;i++){ |
| 79: | xInArray[i]=0.0; |
| 80: | } |
| 81: | for(i=0;i<2;i++){ |
| 82: | yOutArray[i]=0.0; |
| 83: | } |
| 84: | } |

Kôd 11. Funkcije - initXinYout

* *void singleShot(char)*
  + - funkcija prima znak koji određuje koji će se filtar primijeniti, znak može biti : 0,1,2,3 ili 4
    - funkcija ne vraća ništa
    - namjena funkcije je filtrirati vrijednosti, primijeniti omotnicu te reproducirati dobiveni zvuk na zvučnike
* Linija 89 ⤳ generiranje slučajnog broja u rasponu [-1,1]
* Linija 90 ⤳ dijeljenje generiranog broja s pojačanjem specifičnim za određeni filtar [6]
* Linija 91 ⤳ dodavanje novog broja u polje koje pamti zadnja 3 slučajno generirana
* Linija 93 ⤳ filtriranje po formuli uz koeficijente za određeni filtar [6]
* Linija 95 ⤳ dodavanje nove filtrirane vrijednosti u polje koje pamti zadnje dvije takve vrijednosti
* Linije 97-100 ⤳ ukoliko je *i* (vrijednost brojača u *for* petlji) manji od *nA* znači da smo u „Attack“ dijelu omotnice te se množi filtrirana vrijednost s vrijednošću omotnice koja se zatim povećava za određenu konstantu koja je izračunata kako bi držala linearan rast za „Attack“ dio omotnice
* Linije 101-104 ⤳ ukoliko je *i* (vrijednost brojača u *for* petlji) veći od *nA* i manji od *nA+nD* znači da smo u „Decay“ dijelu omotnice te se množi filtrirana vrijednost s vrijednošću omotnice koja se zatim povećava za određenu konstantu koja je izračunata kako bi držala linearan pad za „Decay“ dio omotnice
* Linije 105-108 ⤳ ukoliko je *i* (vrijednost brojača u *for* petlji) veći od *nA+nD* i manji od *nA+nD+nR* znači da smo u „Release“ dijelu omotnice te se množi filtrirana vrijednost s vrijednošću omotnice koja se zatim povećava za određenu konstantu koja je izračunata kako bi držala linearan pad za „Release“ dio omotnice
* Linije 109-110 ⤳ ukoliko je *i* (vrijednost brojača u *for* petlji) veći od *nA+nD+nR* izašli smo iz područja na koji djeluje omotnica te se izlazne vrijednosti postavljaju na 0.
* Linija 112 ⤳ izlazna filtrirana vrijednost se množi sa 32768 kako bi se skalirao *float* tip podatka te kako bi se kasnije mogao prebaciti u *short* iz razloga što se na zvučnike može reproducirati samo *short* tip podatka
* Linija 113 ⤳ pojačanje izlaznog signala (pojačanje glasnoće)
* Linije 115-122 ⤳ pretvorba *float* tipa podatka u *short*
* Linije 124-127 ⤳ reprodukcija *short* vrijednosti na lijevi i desni zvučnik

Vidi kôd 12.

|  |  |
| --- | --- |
| Linija | Kod |
| 87: | void singleShot(char coeff){ |
| 88: |  |
| 89: | randomNumber=-1+2\*((float)rand())/RAND\_MAX; |
| 90: | randomNumber=randomNumber/gainArray[coeff]; |
| 91: | addNewXin(randomNumber,xInArray); |
| 92: |  |
| 93: | yOut= xInArray[0] + 2 \* xInArray[1] + xInArray[2] + |
|  | filterCoefficientsMatrix[coeff][0]\*yOutArray[0] + |
|  | filterCoefficientsMatrix[coeff][1]\*yOutArray[1]; |
| 94: |  |
| 95: | addNewYout(yOut,yOutArray); |
| 96: |  |
| 97: | if(i<nA){ |
| 98: | yOut=yOut\*aADR; |
| 99: | aADR=aADR+0.002087682672223382; |
| 100: | } |
| 101: | else if(i>=nA && i<(nA+nD)){ |
| 102: | yOut=yOut\*dADR; |
| 103: | dADR=dADR-0.001461377870564; |
| 104: | } |
| 105: | else if(i>=(nA+nD) && i<(nA+nD+nR)){ |
| 106: | yOut=yOut\*rADR; |
| 107: | rADR=rADR-0.000034722222222; |
| 108: | } |
| 109: | else |
| 110: | yOut=0.0; |
| 111: |  |
| 112: | yOut=yOut\*32768.0f; |
| 113: | yOut=GAIN\*yOut; |
| 114: |  |
| 115: | if(yOut>32767.0f) |
| 116: | yOutShort=32767; |
| 117: | else if(yOut>=0.0f) |
| 118: | yOutShort=(short)yOut; |
| 119: | else if(yOut<=-32768.0f) |
| 120: | yOutShort=-32768; |
| 121: | else |
| 122: | yOutShort=(short)yOut; |
| 123: |  |
| 124: | aicSample.channel[0]=yOutShort; |
| 125: | aicSample.channel[1]=yOutShort; |
| 126: |  |
| 127: | output\_sample(aicSample.data); |
| 128: | } |
| 129: |  |
|  |  |

Kôd 12. Funkcije - singleShot

### Glavni program

U glavnom programu se ispituju stanja sklopki te se u odnosu na njih primjenjuje određeni filtar. Također, glavni program sadrži inicijalizacijske naredbe kao i određena kašnjenja koja pospješuju reprodukciju zvuka.

* Funkcija *comm\_poll()* izvršava prozivanje (eng. polling) registara McBSP (Multichannel Buffered Serial Port). McBSP omogućuje izravno sučelje s DSP-ovima, kodecima i drugim uređajima. Primarna upotreba McBSP-a je u svrhu audio sučelja. Sastoji se od podatkovnog puta (eng. Dana Path) i upravljačkog puta (eng. Control Path) te uz primarnu audio upotrebu može se programirati za podršku drugih serijskih formata, ali imajući na umu da nije namijenjen korištenju kao sučelje za potrebe velikih brzina. Vidi liniju kôda 131.
* Funkcija *DSK7613\_DIP\_init()* inicijalizira i omogućava korištenje DIP prekidača na samom razvojnom sustavu. Kao što je već rečeno DIP prekidači su korišteni u svrhu testiranja. Vidi liniju kôda 132.
* Naredba *initXinYout()* poziva funkciju koja vrijednosti polja postavlja na 0. Vidi liniju kôda 133. Vidi *4.2.4 Funkcije*.
* Naredba *srand(time(0))* se koristi kao bi se svaki puta generirao novi nasumični broj tako što se svaki puta pomoću funkcije *time(0)* koristi drugo sjeme (eng. seed). Vidi liniju kôda 134.
* Funkcija *DSK6713\_init()* inicijalizira biblioteku za podršku razvojnom sustavu (eng. Board Support Library) koja služi za kontrolu i konfiguraciju svih uređaja na ploči. Vidi liniju kôda 135.
* Funkcija *GPIO\_open()* „otvara“ GPIO uređaj tako što prima kanal koji treba otvoriti (*GPIO\_DEV0)* te zastavicu za otvaranje (*GPIO\_OPEN\_RESET).* Povratna vrijednost funkcije je jedinstvena ručka (eng. handle) uređaja koja se koristi u kasnijim GPIO API pozivima. Vidi liniju kôda 136. Vidi [5].
* Funkcija *GPIO\_config()* postavlja globalni GPIO kontrolni registar koristeći konfiguracijsku strukturu. Vidi liniju kôda 137. Vidi [5]. Vidi konfiguracijsku strukturu u *4.2.4. Varijable i tipovi podataka.*
* Petlja *while(1)* je beskonačna petlja i služi za neprestano provjeravanje je li okidač pritisnut, to jest služi za mogućnost paljbe dok god je razvojni sustav uključen. Vidi liniju kôda 138.
* Linije 139-144 služe za provjeravanje je li okidač na pištolju pritisnut ili nije. U ovisnosti o uvjetima pritiska trenutačna stanja zastavica se postavljaju na određene vrijednosti. Petlja koja se nalazi unutar drugog *if* uvjeta služi za kratko čekanje izvođenja programa u svrhu sprječavanja naglog stiskanja i puštanja prekidača i nekontroliranu promjena zastavica. Vidi linije kôda 139-144.
* Linije 135-148 služe za provjeravanje je li došlo do rastućeg brida prilikom pritiska okidača iz razloga što postoji mogućnost da korisnik drži okidač te bi se pucanj izvodio više puta što je uredu ako je riječ o rafalnoj paljbi, ali nije uredu ako je riječ o pojedinačnim pucnjevima. Vidi linije kôda 135-148.

|  |  |
| --- | --- |
| Linija | Kod |
| 130: | void main(){ |
| 131: | comm\_poll(); |
| 132: | DSK6713\_DIP\_init(); |
| 133: | initXinYout(); |
| 134: | srand(time(0)); |
| 135: | DSK6713\_init(); |
| 136: | gpio\_handle = GPIO\_open(GPIO\_DEV0,GPIO\_OPEN\_RESET ); |
| 137: | GPIO\_config(gpio\_handle,&gpio\_config); |
| 138: | while(1){ |
| 139: | if(GPIO\_read(gpio\_handle,GPIO\_PIN8)==256) //ukljucen |
| 140: | current\_state = 0; //ukljucen |
| 141: | if(GPIO\_read(gpio\_handle,GPIO\_PIN8)==0){ |
| 142: | current\_state = 1; //iskljucen |
| 143: | for(delay=0; delay<5000; delay++){} |
| 144: | } |
| 145: | if(previous\_state == 1 && current\_state == 0 ){ |
| 146: | flagRisingEdge = 1; |
| 147: | for(delay=0; delay<5000; delay++){} |
| 148: | } |
| 149: |  |

Kôd 13. Glavni program(1)

* Linije 150-180 ispituju sve prekidače te pozivaju funkciju *singleShot()* za reprodukciju zvuka. Ukoliko je detektiran rastući brid, to jest ukoliko je pritisnut okidač izvest će se rafalna paljba ili pojedinačni pucanj. Vrsta paljbe ovisi o tome što će se pročitati s pina 13. Dok god je pritisnut okidač i rafalna paljba omogućena izvršavati će se linije kôda 153-169. Tada će se u jednoj *for* petlji ispitivati koji je filtar odabran te će s njegov indeks slati kao parametar u funkciju *singleShot().* Brojač koji pripada petlji se koristi za detekciju dijela ADSR omotnice koji se treba primijeniti, no u slučaju rafalne paljbe taj brojač ide samo do 3600 (omotnica ima 9600 vrijednosti) te se time postiže preklapanje pucnjeva kao efekt rafalne paljbe. Nakon što je prekidač pušten izvesti će se jedan cjeloviti pucanj koji ima efekt postupnog utišavanja pucnja. Ukoliko korisnik želi pojedinačni pucanj i pin 13 to očitava, *while* petlja će se preskočiti te će se izvršiti samo cjeloviti pucanj. Na kraju pucnja potrebno je obnoviti polja te postaviti zastavice na određene vrijednosti.

|  |  |
| --- | --- |
| Linija | Kod |
| 150: | if(flagRisingEdge){ |
| 151: |  |
| 152: | while((GPIO\_read(gpio\_handle,GPIO\_PIN13) == 0) && |
|  | (GPIO\_read(gpio\_handle,GPIO\_PIN8)==256)){ |
| 153: | for(i=0;i<3600;i++){ |
| 154: | // singleShot parameter is distance |
| 155: | for filter coeff |
| 156: | if(GPIO\_read(gpio\_handle,GPIO\_PIN10)==0) |
| 157: | singleShot(1); |
| 158: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN9)==0) |
| 159: | singleShot(2); |
| 160: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN12)==0) |
| 161: | singleShot(3); |
| 162: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN11)==0) |
| 163: | singleShot(4); |
| 164: | else |
| 165: | singleShot(0); |
| 166: | } |
| 167: | initXinYout(); |
| 168: | flagRisingEdge = 0; |
| 169: | } |
| 170: |  |
| 171: | for(i=0;i<10000;i++){ |
| 172: | if(GPIO\_read(gpio\_handle,GPIO\_PIN10)==0) |
| 173: | singleShot(1); |
| 174: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN9)==0) |
| 175: | singleShot(2); |
| 176: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN12)==0) |
| 177: | singleShot(3); |
| 178: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN11)==0) |
| 179: | singleShot(4); |
| 180: | else |
| 181: | singleShot(0); |
| 182: | } |
| 183: |  |
| 184: | initXinYout(); |
| 185: | flagRisingEdge = 0; |
| 186: | } |
| 187: | previous\_state = current\_state; |
| 188: | } |
| 189: | } |

Kôd 14. Glavni program(2)

* Pinovi:
  + - pin8 - okidač
    - pin9 - odabir filtra sa graničnom frekvencijom od 700Hz
    - pin10 - odabir filtra sa graničnom frekvencijom od 850Hz
    - pin11 - odabir filtra sa graničnom frekvencijom od 400Hz
    - pin12 - odabir filtra sa graničnom frekvencijom od 550Hz
    - pin 13 - rafalna paljba

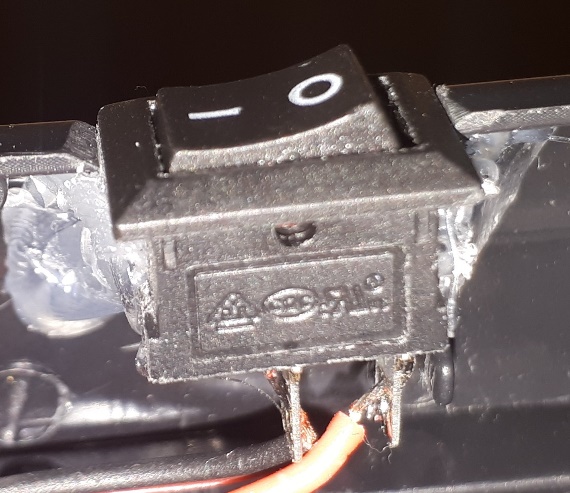
# Izrada makete pištolja

## Ideja

Prvotna ideja je bila da se maketa pištolja modelira te isprinta 3D printerom, ali zbog neispravnosti 3D printera korištena je maketa pištolja igraće konzole PlayStation 1. Maketa je u sebi sadržavala raznu elektroniku, no ništa osim mikro prekidača nije bilo potrebno za ovaj projekt pa je cijela maketa ispražnjena i sačuvan je samo mikro prekidač koji služi kao okidač paljbe. Zadatak projekta je podrazumijevao da pištolj ima prekidač koji će određivati način rada pištolja, to jest omogućiti ili onemogućiti rafalnu paljbu. Zatim, pištolj je morao imati i mogućnost odabira prigušenosti zvuka, to jest promjenu jačine zvuka za određene udaljenosti. Na posljetku sve komponente je trebalo elegantno spojiti kako bi izgled pištolja bio što bolji te je korišten LAN konektor.

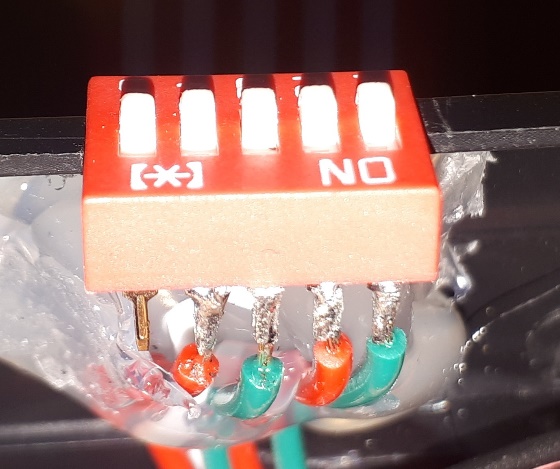
## Komponente i njihove uloge

* On/0ff prekidač ⤳ ima dva pina s kojim se pali i gasi rafalna paljba. Jedan pin je potrebno spojiti kao ulaz na GPIO pin, a drugi pin je potrebno uzemljiti (GND). Vidi sliku 12.



Slika 12. On/Off prekidač

* DIP prekidač ⤳ upravlja odabirom filtra te je potrebno spojiti cijeli red pinova na GPIO pinove razvojne ploče, a drugi red pinova uzemljiti (GND). Korišteni DIP prekidač ima 5 malih prekidača za 5 različitih udaljenosti, to jest 10 pinova sveukupno. Jedan pin nije iskorišten zbog manjka slobodnih pinova na razvojnoj ploči. Vidi sliku 13.



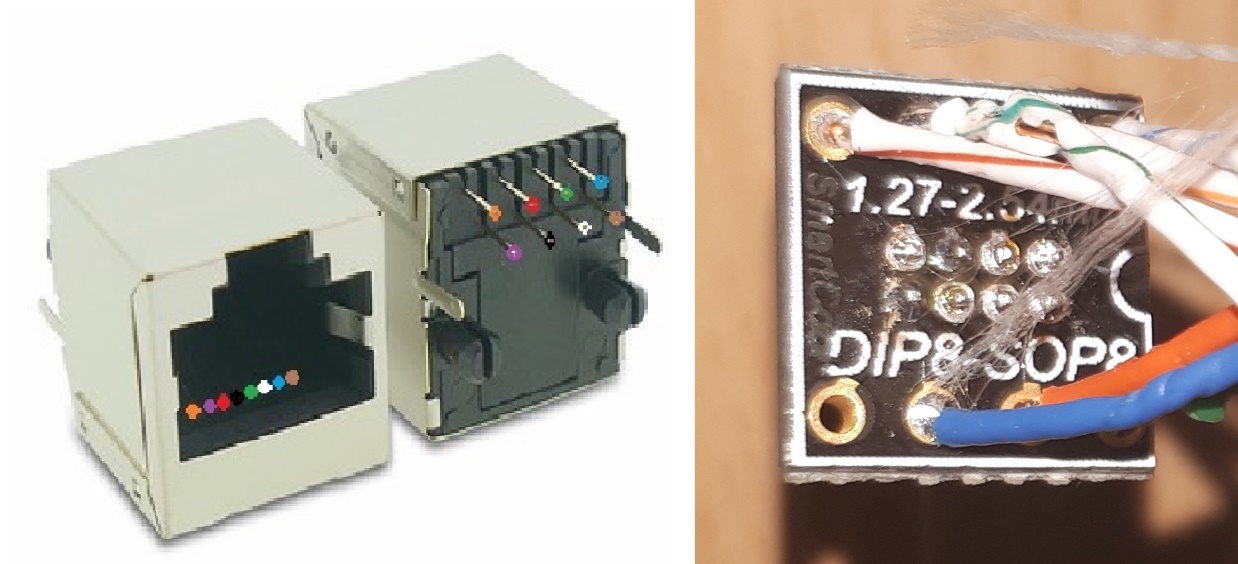
Slika 13. DIP prekidač

* Mikro prekidač ⤳ čija je zadaća dati signal za paljbu ima 3 pina, no potrebno je iskoristiti samo 2. Jedan pin je potrebno spojiti kao GPIO ulaz na razvojnu ploču, a drugi pin je potrebno uzemljiti (GND). Kod ovog prekidača potrebno je znati koji će se pin spojiti na ulaz, a koji na GND. Na slici 14 možete vidjeti kako su dva pina kratko spojena kada je prekidač u početnoj poziciji. U ovisnosti o tome odabirete hoće li pištolj okidati na logičku „0“ ili „1“. U ovom projektu je odabrano da se puca na logičku jedinicu jer su odabrane dvije žice koje su kratko spojene na početnoj poziciji prekidača te do trenutka pucnja prekidač daje razvojnoj ploči signal s logičkom nulom. Vidi sliku 14.

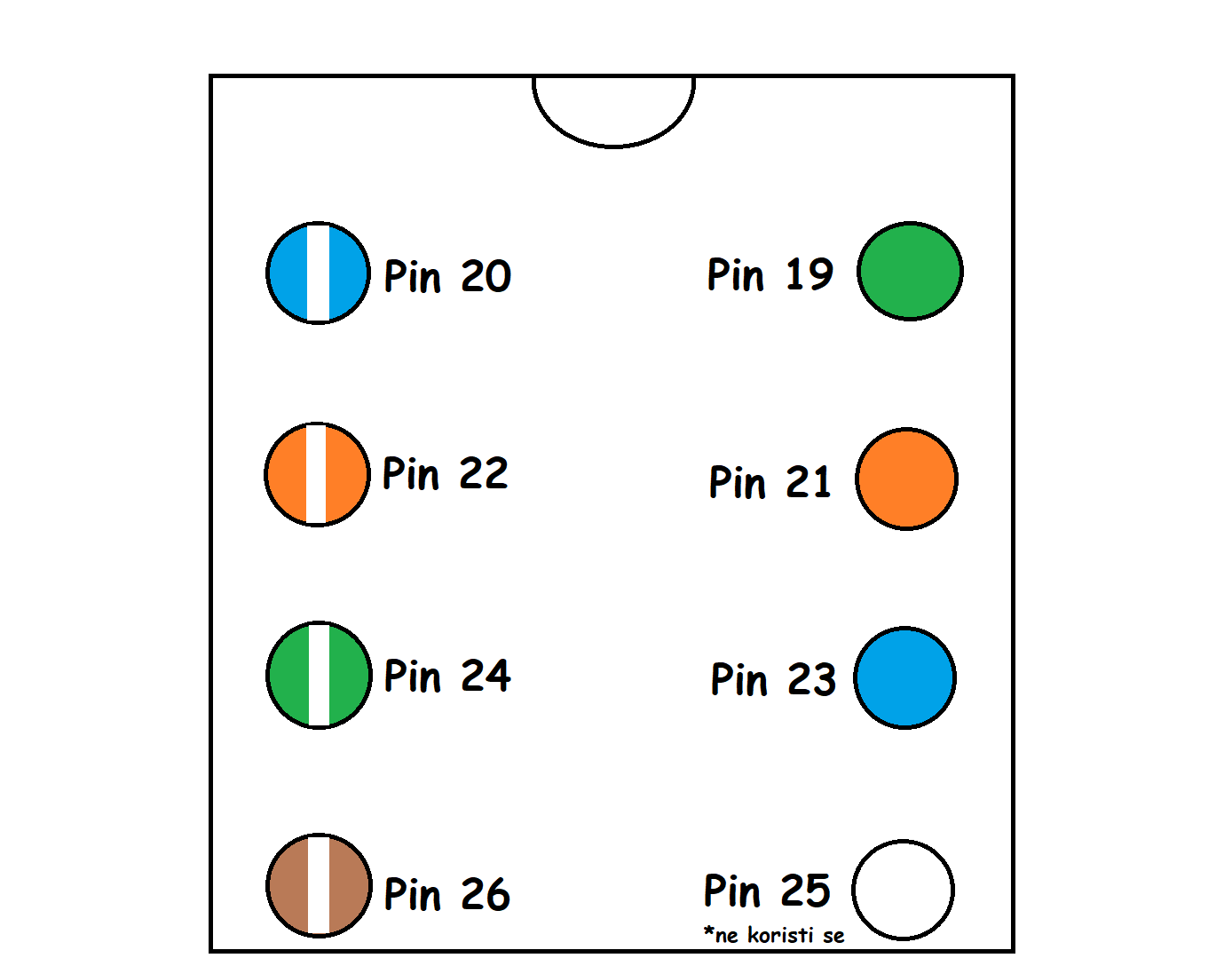


Slika 14. Mikro prekidač

* LAN konektor i kabel ⤳ ima 8 pinova i od iznimne je važnosti zapamtiti na koje pinove se spajaju pojedine komponente kako bi se znalo koja žica dovodi pojedini signal. Kako bi se olakšalo priključivanje žica LAN kabla na razvojni sustav korišten je adapter DIP8 - SOP8 koji se spaja na J1 konektor razvojne ploče. Na slici 15 može se vidjeti raspored žica po bojama i pripadajući pinovi na razvojnoj ploči. Vidi sliku 15 i 16.



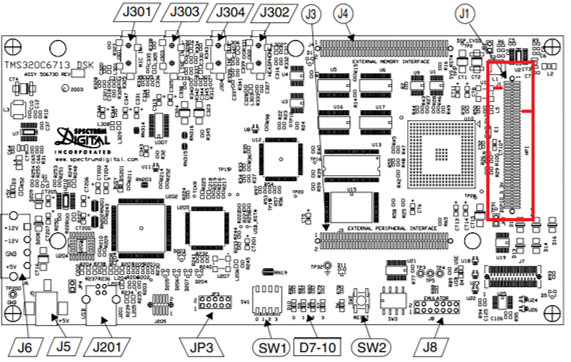
Slika 15. LAN konektor i DIP8-SOP8 adapter



Slika 16. Raspored pinova



Slika 17. Maketa pištolja



Slika 18. Konektor J1 na razvojnom sustavu

# Prilog

|  |  |
| --- | --- |
| Linija | Kod |
| 1: | #include "dsk6713\_aic23.h"//codec podrška |
| 2: | #include <dsk6713\_dip.h> //dip prekidači–tesitranje pucnja |
| 3: | #include <stdlib.h> //potrebno za srand() |
| 4: | #include <time.h> //potrebno za time() |
| 5: | #include <dsk6713.h> //GPIO |
| 6: | #include <csl\_gpio.h> //GPIO |
| 7: | #include <csl\_gpiohal.h> //GPIO |
| 8: | #include <csl\_irq.h> //GPIO |
| 9: |  |
| 10: | #define GAIN 10 |
| 11: | #define DSK6713\_AIC23\_INPUT\_LINE 0x0011 |
| 12: | //VARIABLES |
| 13: |  |
| 14: | Uint32 inputsource=DSK6713\_AIC23\_INPUT\_LINE; |
| 15: | Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; |
| 16: | float xInArray[3],yOutArray[2]; |
| 17: | float yOut; |
| 18: | float randomNumber; |
| 19: | short yOutShort; |
| 20: | int delay; |
| 21: | short nA=480; |
| 22: | short nD=480; |
| 23: | short nR=8640; |
| 24: | float aADR=0.0; |
| 25: | float dADR=1.0; |
| 26: | float rADR=0.3; |
| 27: |  |
| 28: | float filterCoefficientsMatrix[5][2] = { |
| 29: | {-0.6905989232, 1.6329931619}, //1000 |
| 30: | {-0.7300190220, 1.6873607674}, //850 |
| 31: | {-0.7716989200, 1.7420316578}, //700 |
| 32: | {-0.8157642007, 1.7969735593}, //550 |
| 33: | {-0.8623486260, 1.8521464854}, //400 |
| 34: | }; |
| 35: | float gainArray[5] = {69.43749902, 93.76848728, |
| 36: | 134.8287542, 212.8719253, 392.0745796}; |
| 37: | char previous\_state = 1; |
| 38: | char current\_state= 1; |
| 39: | char flagRisingEdge=0; |
| 40: |  |
| 41: | union AIC\_data{ |
| 42: | Uint32 data; |
| 43: | short channel[2]; |
| 44: | }aicSample; |
| 45: |  |
| 46: | GPIO\_Handle gpio\_handle; |
| 47: |  |
| 48: | GPIO\_Config gpio\_config = { |
| 49: | 0x00000000, // gpgc |
| 50: | 0x0000FF00, // gpen |
| 51: | 0x00000000, // gdir |
| 52: | 0x00000000, // gpval |
| 53: | 0x00000000, // gphm |
| 54: | 0x00000000, // gplm |
| 55: | 0x00000000 // gppol |
| 56: | }; |
| 57: |  |
| 58: | short i=0,j=0; //"for" counter |

|  |  |
| --- | --- |
| Linija | Kod |
| 46: | GPIO\_Handle gpio\_handle; |
| 47: |  |
| 48: | GPIO\_Config gpio\_config = { |
| 49: | 0x00000000, // gpgc |
| 50: | 0x0000FF00, // gpen |
| 51: | 0x00000000, // gdir |
| 52: | 0x00000000, // gpval |
| 53: | 0x00000000, // gphm |
| 54: | 0x00000000, // gplm |
| 55: | 0x00000000 // gppol |
| 56: | }; |
| 57: |  |
| 58: | short i=0,j=0; //"for" counter |
| 59 |  |
| 60: | void addNewXin(float randomNumber){ |
| 61: | float temp1=xInArray[1]; |
| 62: | float temp2=xInArray[2]; |
| 63: | xInArray[0]=temp1; |
| 64: | xInArray[1]=temp2; |
| 65: | xInArray[2]=randomNumber; |
| 66: | //return xInArray; |
| 67: | } |
| 68: |  |
| 69: |  |
| 70: | void addNewYout(float yOut){ |
| 71: | float temp1=yOutArray[1]; |
| 72: | yOutArray[0]=temp1; |
| 73: | yOutArray[1]=yOut; |
| 74: | //return yOutArray; |
| 75: | } |
| 76: |  |
| 77: | void initXinYout(){ |
| 78: | for(i=0;i<3;i++){ |
| 79: | xInArray[i]=0.0; |
| 80: | } |
| 81: | for(i=0;i<2;i++){ |
| 82: | yOutArray[i]=0.0; |
| 83: | } |
| 84: | } |
| 85: |  |
| 86: |  |
| 87: | void singleShot(char coeff){ |
| 88: |  |
| 89: | randomNumber=-1+2\*((float)rand())/RAND\_MAX; |
| 90: | randomNumber=randomNumber/gainArray[coeff]; |
| 91: | addNewXin(randomNumber,xInArray); |

# 

|  |  |
| --- | --- |
| Linija | Kod |
| 93: | yOut= xInArray[0] + 2 \* xInArray[1] + xInArray[2] + |
|  | filterCoefficientsMatrix[coeff][0]\*yOutArray[0] + |
|  | filterCoefficientsMatrix[coeff][1]\*yOutArray[1]; |
| 95: | addNewYout(yOut,yOutArray); |
| 96: |  |
| 97: | if(i<nA){ |
| 98: | yOut=yOut\*aADR; |
| 99: | aADR=aADR+0.002087682672223382; |
| 100: | } |
| 101: | else if(i>=nA && i<(nA+nD)){ |
| 102: | yOut=yOut\*dADR; |
| 103: | dADR=dADR-0.001461377870564; |
| 104: | } |
| 105: | else if(i>=(nA+nD) && i<(nA+nD+nR)){ |
| 106: | yOut=yOut\*rADR; |
| 107: | rADR=rADR-0.000034722222222; |
| 108: | } |
| 109: | else |
| 110: | yOut=0.0; |
| 111: |  |
| 112: | yOut=yOut\*32768.0f; |
| 113: | yOut=GAIN\*yOut; |
| 114: |  |
| 115: | if(yOut>32767.0f) |
| 116: | yOutShort=32767; |
| 117: | else if(yOut>=0.0f) |
| 118: | yOutShort=(short)yOut; |
| 119: | else if(yOut<=-32768.0f) |
| 120: | yOutShort=-32768; |
| 121: | else |
| 122: | yOutShort=(short)yOut; |
| 123: |  |
| 124: | aicSample.channel[0]=yOutShort; |
| 125: | aicSample.channel[1]=yOutShort; |
| 126: |  |
| 127: | output\_sample(aicSample.data); |
| 128: | } |
| 129: |  |
| 130: | void main(){ |
| 131: | comm\_poll(); |
| 132: | DSK6713\_DIP\_init(); |
| 133: | initXinYout(); |
| 134: | srand(time(0)); |
| 135: | DSK6713\_init(); |
| 136: | gpio\_handle = GPIO\_open(GPIO\_DEV0,GPIO\_OPEN\_RESET ); |
| 137: | GPIO\_config(gpio\_handle,&gpio\_config); |

# 

|  |  |
| --- | --- |
| Linija | Kod |
| 138: | while(1){ |
| 139: | if(GPIO\_read(gpio\_handle,GPIO\_PIN8)==256) //ukljucen |
| 140: | current\_state = 0; //ukljucen |
| 141: | if(GPIO\_read(gpio\_handle,GPIO\_PIN8)==0){ |
| 142: | current\_state = 1; //iskljucen |
| 143: | for(delay=0; delay<5000; delay++){} |
| 144: | } |
| 145: | if(previous\_state == 1 && current\_state == 0 ){ |
| 146: | flagRisingEdge = 1; |
| 147: | for(delay=0; delay<5000; delay++){} |
| 148: | } |
| 149: |  |
| 150: | if(flagRisingEdge){ |
| 151: |  |
| 152: | while((GPIO\_read(gpio\_handle,GPIO\_PIN13) == 0) && |
|  | (GPIO\_read(gpio\_handle,GPIO\_PIN8)==256)){ |
| 153: | for(i=0;i<3600;i++){ |
| 154: | // singleShot parameter is distance |
| 155: | for filter coeff |
| 156: | if(GPIO\_read(gpio\_handle,GPIO\_PIN10)==0) |
| 157: | singleShot(1); |
| 158: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN9)==0) |
| 159: | singleShot(2); |
| 160: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN12)==0) |
| 161: | singleShot(3); |
| 162: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN11)==0) |
| 163: | singleShot(4); |
| 164: | else |
| 165: | singleShot(0); |
| 166: | } |
| 167: | initXinYout(); |
| 168: | flagRisingEdge = 0; |
| 169: | } |
| 170: |  |
| 171: | for(i=0;i<10000;i++){ |
| 172: | if(GPIO\_read(gpio\_handle,GPIO\_PIN10)==0) |
| 173: | singleShot(1); |
| 174: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN9)==0) |
| 175: | singleShot(2); |
| 176: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN12)==0) |
| 177: | singleShot(3); |
| 178: | else if(GPIO\_read(gpio\_handle,GPIO\_PIN11)==0) |
| 179: | singleShot(4); |
| 180: | else |
| 181: | singleShot(0); |
| 182: | } |

|  |  |
| --- | --- |
| Linija | Kod |
| 184: | initXinYout(); |
| 185: | flagRisingEdge = 0; |
| 186: | } |
| 187: | previous\_state = current\_state; |
| 188: | } |
| 189: | } |

Kôd 15. Cjeloviti kod

# Literatura

|  |  |
| --- | --- |
| [1] | [https://en.wikipedia.org/wiki/MATLAB](https://www.mathworks.com/products/matlab.html) |
| [2] | <https://www.mathworks.com/products/matlab.html> |
| [3] | <https://webaudio.prototyping.bbc.co.uk/gunfire/> |
| [4] | <https://loomen.carnet.hr/course/view.php?id=5571> |
| [5] | <https://loomen.carnet.hr/pluginfile.php/450432/mod_resource/content/1/Chip%20Support%20Library%20spru401j.pdf> |
| [6] | <https://www-users.cs.york.ac.uk/~fisher/mkfilter/trad.html> |
| [7] | <https://e2e.ti.com/support/processors/f/791/t/46560?C6713-GPIO-Help> |
| [8] | <https://loomen.carnet.hr/pluginfile.php/450430/mod_resource/content/1/tms320c6713_CPU%20Instruction%20Set.pdf> |
| [9] | <https://loomen.carnet.hr/pluginfile.php/450431/mod_resource/content/1/TMS320C6000%20programmers%20guide%20spru198k.pdf> |
| [10] | <https://loomen.carnet.hr/pluginfile.php/450433/mod_resource/content/1/BSL%20board%20support%20lib%20spru432a.pdf> |