Univerzitet u Beogradu
Elektrotehnički fakultet

Nikola Cvetković, 3094/2020

Ispitivanje hardverskih performansi procesora TSM320C5545

Projekat - **13M041DPS**

Beograd, Mart 2021.

Sadržaj

Glava 1

Uvod

Cilj projekta jeste prenos postojećih primera sa vežbi predmeta, sa stare na novu platformu. Dodatno, proverena je ispravnost rada svih postojećih primera.

Platforma koja je korišćena na predmetu jeste TMDX5505EZDSP za koju je, u trenutnku izrade projekta, podrška proizvođača bila sve slabija, uključujući ažuriranje dokumentacije i razvoj biblioteka/drajvera za platformu.

Osim sigurnosti po pitanju održive podrške, prelazak na novu, BOOST5545ULP platformu, proizvedenu 2016. godine, pruža mnogo više mogućnosti predmetnom nastavniku pri odabiru tema projektnih zadataka na predmetu, a korisniku platforme više mogućnosti za interagovanje sa hardverom i predstavljanje rezultata.

Glavna razlika između stare i nove platforme jeste u procesoru za digitalnu obradu signala. Stari procesor TSM320C5505 zamenjen je za noviji iz iste familije - TSM320C5545. Međutim, bitne razlike između ova dva procesora sa programerskog stanovišta nema - imaju iste memorijske kapacitete, jednak broj DMA kontrolera, skoro isti broj ulaza za analogno-digitalnu konverziju.

U ovom dokumentu će biti dat pregled karakteristika i interfejsnih mogućnosti pločice, kao i opšte uputstvo za uspešno kreiranje i kompajliranje projekta u razvojnom okruženju.

Glava 2

Pregled karakteristika platforme BOOST5545ULP

Glavne karakteristike korišćene platforme date su u njenom korisničkom uputstvu. Za BOOST5545ULP nije napisan standardni *datasheet* već je dokumentovan u vidu dva teksta:

- TMS320C5545 BoosterPack Hardware User's Guide
- C5545 BoosterPack Software User's Guide

Iz prvog dokumenta se mogu izvući informacije o hardverskim mogućnostima pločice. Vidi se da je prisutan pomenuti procesor, TSM320C5545, ali i mikrokontroler opšte namene, CC2650, koji sadrži ARM Cortex-M3 procesor. Ovaj mikrokontroler sadrži bežični primopredajnik, koji se u primeru sa SD kartice BOOST5545ULP pločice koristi za ostvarenje*Bluetooth* komunikacije.

Ova platforma, kao i stara, sadrži audio kodek, s tim što on više nije AIC3204, već AIC3206, a postoji i čitač SD kartice, sa koje je moguće čitanje i upis podataka, pa i podizanje sistema.

Osim jednog ulaznog i jednog izlaznog audio priključka, od interfejsa postoje tri programabilna tastera opšte namene, tri programabilne LE diode, OLED displej, dva Micro USB konektora, JTAG konektor (samo za CC2650), LaunchPad konektor za povezivanje sa istoimenim pločicama, kao i jedan mikrofon.

Sve ovo pruža korisniku velik broj mogućnosti za testiranje i razvoj različitih oblasti i algoritama digitalne obrade signala, a procesorka moć komponenata omogućava implementaciju više različitih interfejsa odjednom.

Tako se u primeru sa SD kartice u istom trenutku koristi složena obrada govora, čitanje zvučnog signala sa kartice i njegovo prosleđivanje na audio izlaz, prikaz trenutnog stanja kôda na OLED displeju, kao i komunikacija preko *Bluetooth*-a.

Drugi gorepomenuti dokument predstavlja opis rada primera sa SD kartice. On takođe objašnjava korisniku kako da podesi radno okruženje za uspešnu analizu i kompajliranje ovog programa, kao i samo podešavanje kratkospajača na platformi koje će dovesti do uspešnog učitavanja programa sa kartice.

Dodatni dokumenti koji korisniku mogu biti od značaja jesu:

- TMS320C55x Chip Support Library API Reference Guide
- TMS320C5545/35/34/33/32 Ultra-Low Power DSP Technical Reference Manual

Važno je napomenuti da se procesor C5545 u odnosu na korišćeni C5505 suštinski ne razlikuje. Glavna učinjena promena je u broju regulatora koji smanjuju radni napon većeg broja modula čipa, a pored toga je sâm čip za oko 30% manji. Ovo dovodi do značajnog smanjenja u potrošnji.

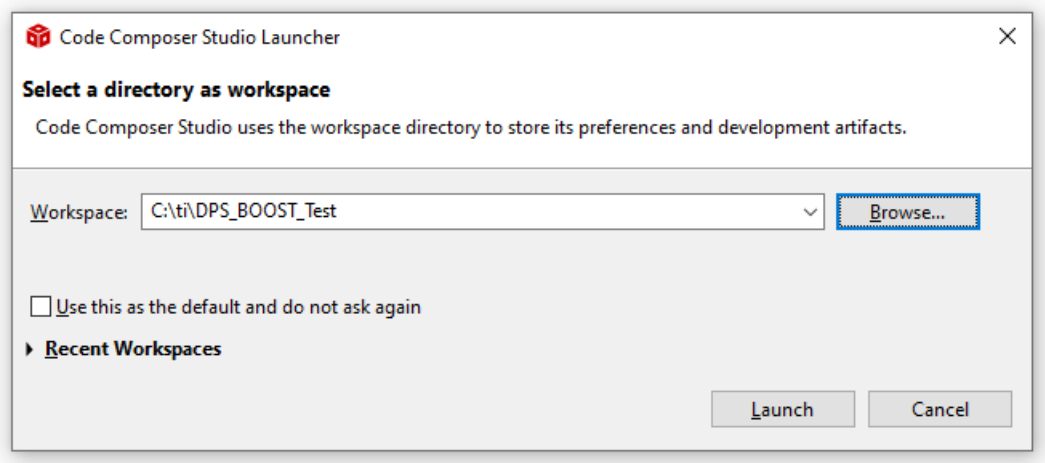
Gotovo svi navedeni dokumenti se mogu naći na sajtu proizvođača, na stranici koja se odnosi na BOOST5545ULP platformu, a koja je data u glavi 1.

Glava 3

Uputstvo za podešavanje okruženja

Integrisano razvojno okruženje (IDE) koje je korišćeno jeste *Code Composer Studio v10.2.0.00009*. To je besplatno okruženje dostupno na sajtu proizvođača platforme, a podržavaju ga svi rasprostranjeni operativni sistemi.

Pri pokretanju programa, korisniku će biti ponuđen izbor radnog prostora na računaru (*workspace*), koji je sledećeg izgleda:



Slika 3.1: Prikaz prozora za izbor radnog prostora

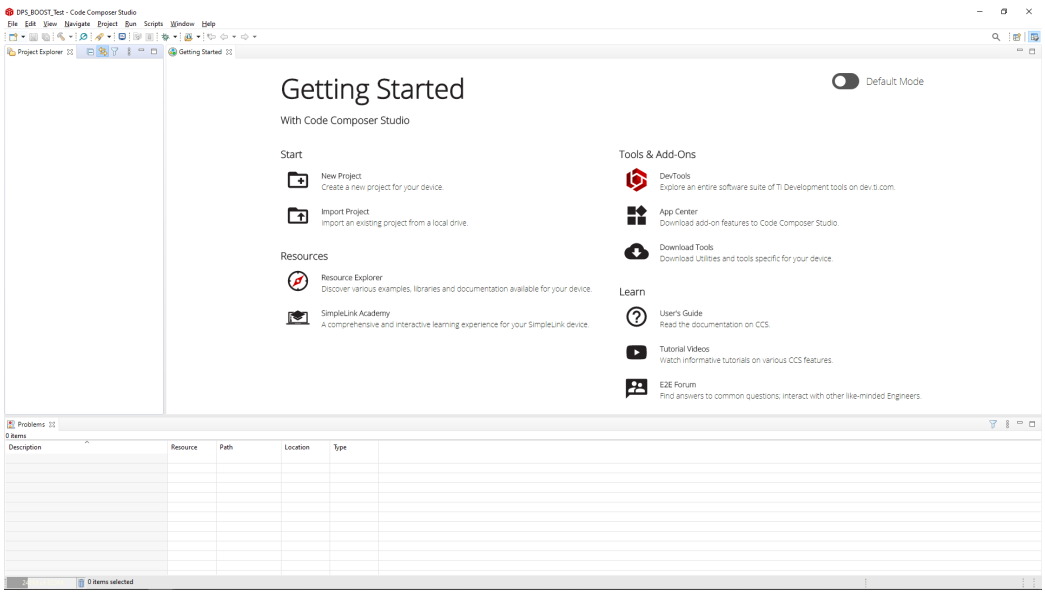
Pri izboru putanje i direktorijuma u kom će se raditi, bitno je voditi računa o nazivu i lokaciji. Preporuka je da se folder u okviru kog će se raditi nalazi na putanji `C:\ti\`, a da sâm folder ne sadrži razmake i specijalne karaktere, kao što je prikazano na prethodnoj slici.

Nakon pritiska na dugme `Launch`, pokreće se radno okruženje, a ukoliko je prvi put kreirano, u njemu neće biti prikazano ništa osim uvodnog prozora - `Getting Started`.

Pritiskom na `View -> Project Explorer` otvara se prozor u okviru kog se vrši pregled i rad nad dostupnim projektima. Pri prvom pokretanju, ovaj prozor nije fiksiran za prozor tekstualnog editora, pa ga je potrebno prevući u stranu editora na kojoj se želi fiksirati - obično je to leva strana okruženja.

Nakon pomenutih izmena u prikazu, okruženje bi trebalo da izgleda kao na slici 3.2, a `Project Explorer` prozor bi trebalo da bude prazan, kako nijedan projekat do sad nije uvezen u radni prostor (*workspace*).

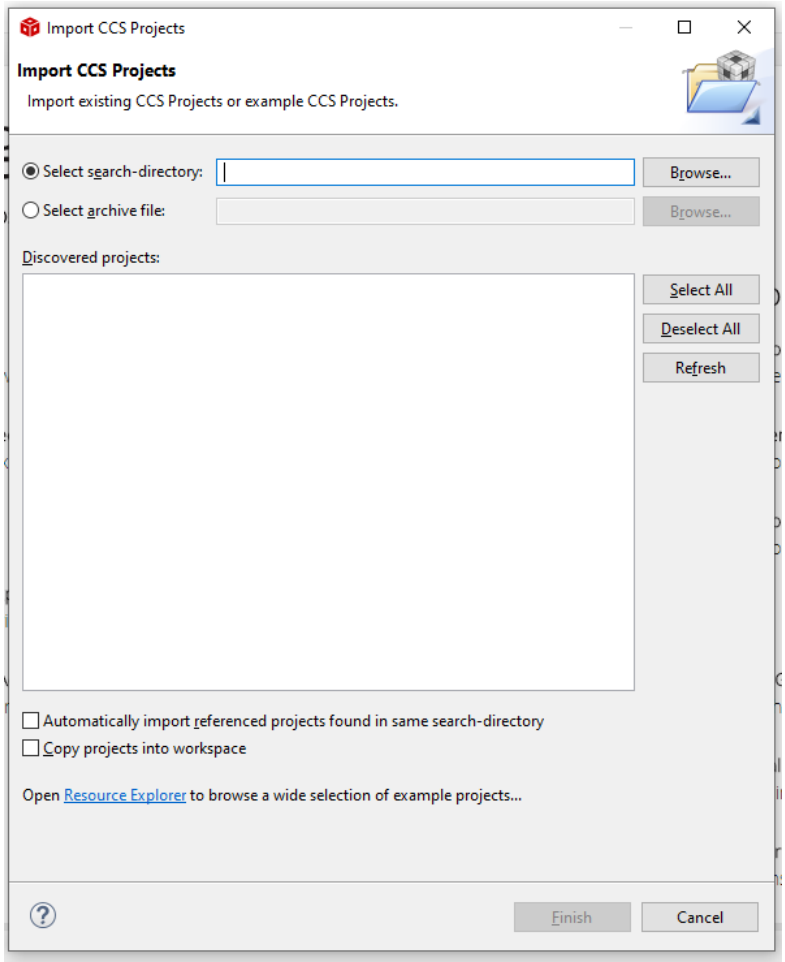
Prikaz ovog prozora može da se razlikuje u zavisnosti od verzije okruženja, ali bi sve funkcionalnosti opisane u ovom dokumentu trebalo da postoje.



Slika 3.2: Prikaz okruženja nakon fiksiranja Project Explorer prozora

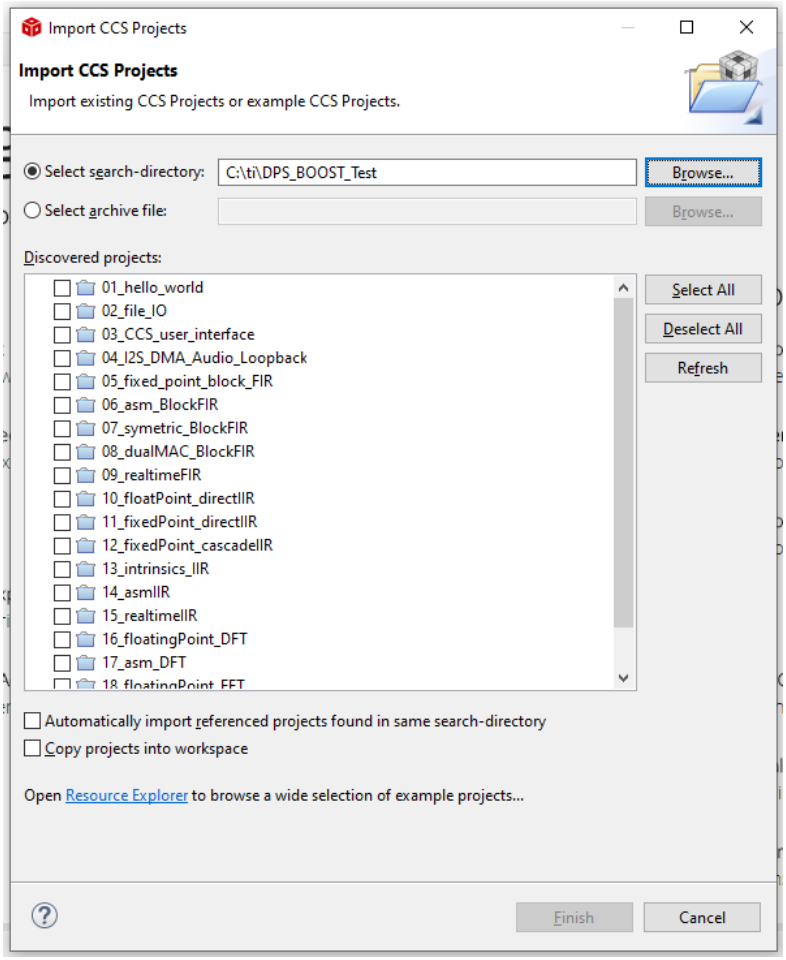
3.1 Uvoz postojećih projekata

Projekti se uvoze (*import*) u radni prostor odlaskom na `Project -> Import CCS Projects...` nakon čega se otvara sledeći prozor:



Slika 3.3: Prozor za uvoz projekata

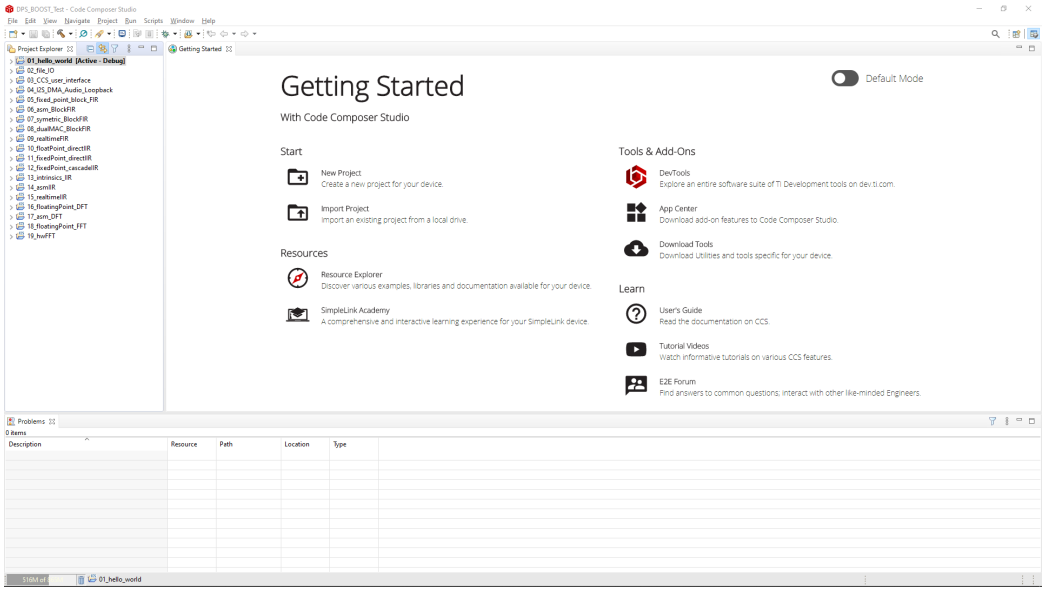
Ovde je moguće odabrati projekte koji se ne nalaze u odabranom *workspace*-u, ali se mora voditi računa da će u tom slučaju svaka izmena odabranog projekta biti sačuvana na lokaciji na kojoj se projekat nalazi.Kako bi se ovo izbeglo, moguće je štiklirati opciju `Copy projects into workspace` na dnu prozora sa slike 3.3, kako bi se samo lokalna kopija u radnom prostoru menjala, a projekat na originalnoj lokaciji ostao netaknut. Konkretno, najlakši pristup je preuzeti i otpakovati projekte sa sajta u odabrani *workspace* (ovde je to `C:\ti\DPS_BOOST_Test`). Nakon toga je moguće pritiskom dugmeta `Browse...` odabrati folder u kom se nalaze projekti koji će potom biti prikazani u prozoru, što je prikazano na slici 3.4



Slika 3.4: Prozor za uvoz projekata - odabran *workspace*

Ovde je korisno napomenuti da se prilikom kopiranja projekata u odabrani *workspace* njegov sadržaj promenio, iako korisnik do sada nije radio ništa u okviru foldera radnog prostora. Folderi dodati u folder radnog prostora predstavljaju neophodan sadržaj koji je *Code Composer Studio* kreirao radi praćenja promena nad radnim prostorom i ne treba ih menjati/brisati.

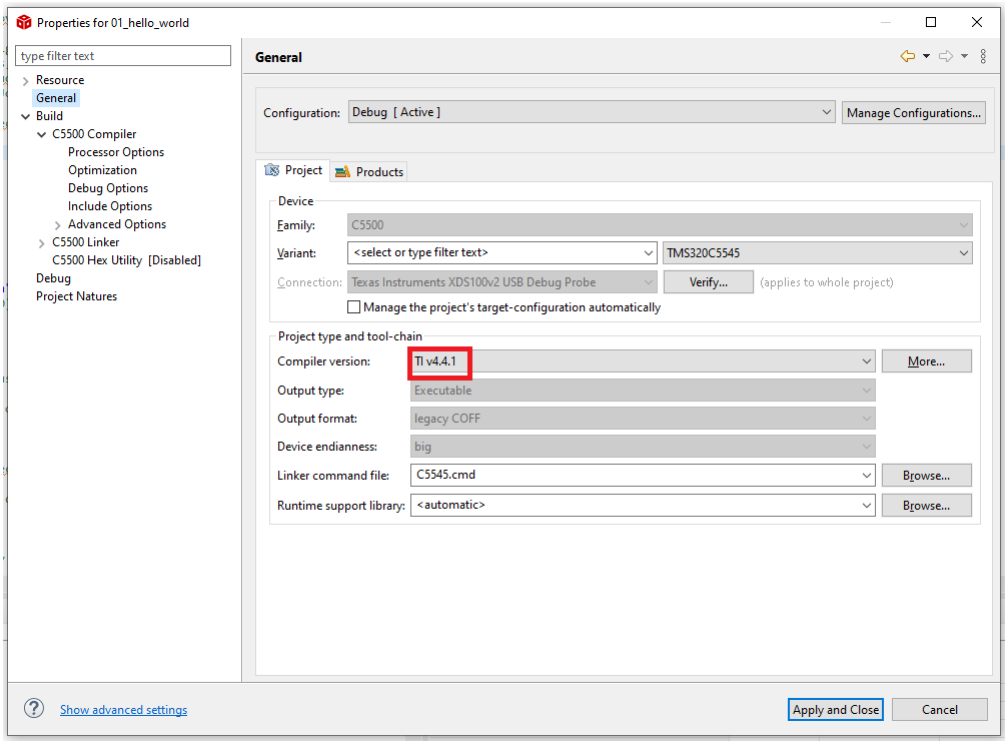
Nakon što je odabran folder iz koga će se uvesti projekti, pritisnuti na dugme `Select All`, a potom na `Finish`, nakon čega se prozor za uvoz projekata zatvara i projekti su prikazani u okviru `Project Explorer` prozora, kao na slici 3.5 na sledećoj strani:



Slika 3.5: Glavni prozor - uvezeni projekti

Ukoliko se to nije automatski obavilo, pritisnuti dva puta na projekat `01_hello_world`, a zatim na fajl `main.c`, nakon čega bi trebalo da se ovaj `C` fajl otvori u editoru.

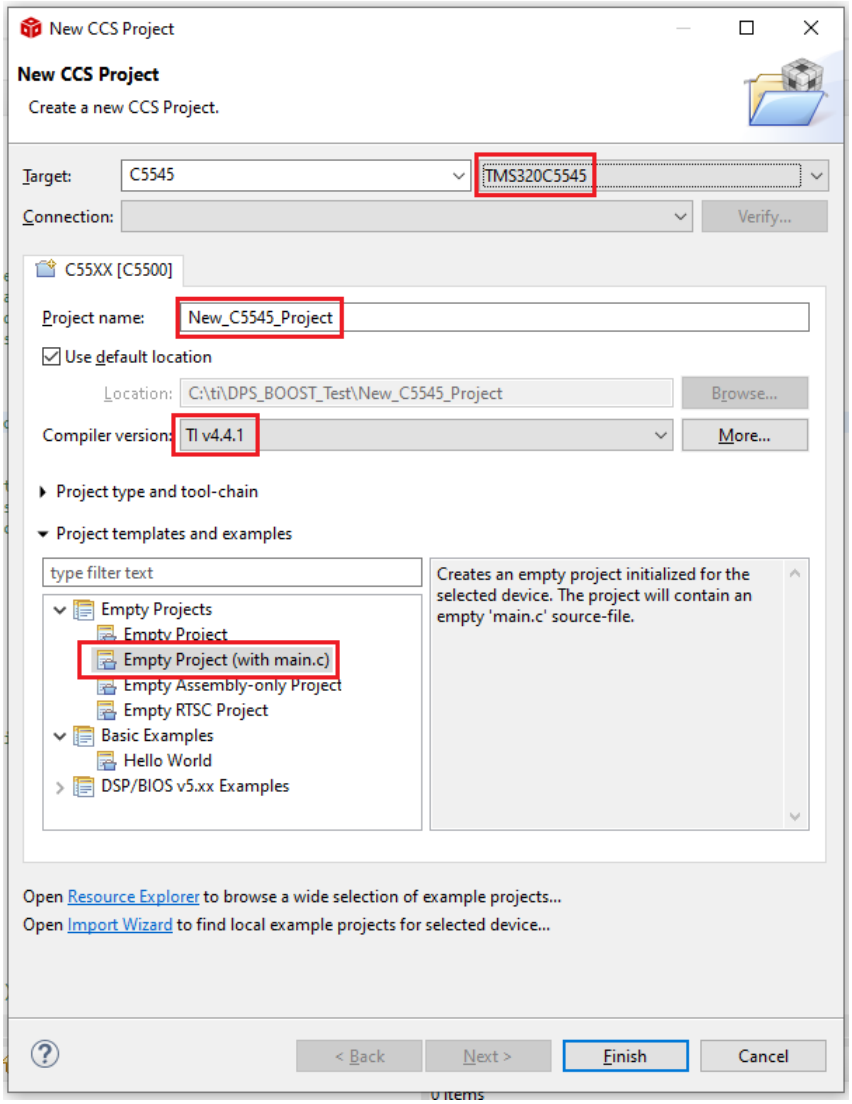
Kada bi se probalo kompajliranje ovog programa, ono bilo bi neuspešno. Razlog za to je što nije preuzet i instaliran kompajler za ovaj procesor, a može se preuzeti sa ovog linka, u okviru *C55xx Code Generation Tools Download*. Nakon uspešne instalacije u prozoru `Problems` na dnu okruženja bi trebalo da je nestalo upozorenje koje govori o nekompatibilnosti kompajlera i projekta. Ukoliko je verzija instaliranog kompajlera veća od verzije sa kojom je projekat kompajliran (v4.4.1), treba otići na `Project -> Properties`, i pod `General - Compiler Version` izabrati `TI v4.4.1`, kao što je prikazano:



Slika 3.6: Odabir kompajlera projekta

3.2 Kreiranje novog projekta

Ukoliko se želi kreirati novi projekat za C5545 procesor, to se može lako uraditi odlaskom na `Project -> New CCS Project...` nakon čega se otvara sledeći prozor u okviru kog treba odabrati za koji se procesor pravi projekat, koji je tip kompajlera i projekta, a treba uneti i naziv projekta (preporuka je da naziv ne sadrži razmake):

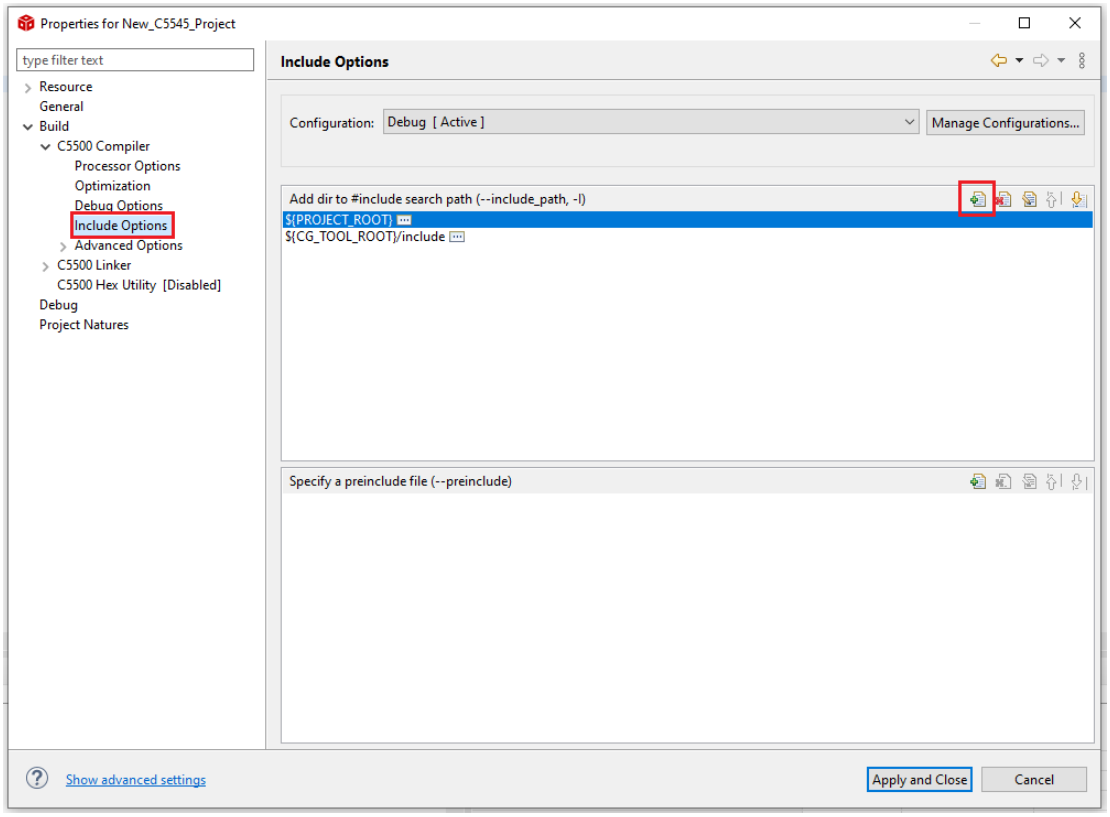


Slika 3.7: Podešavanje pri kreiranju novog projekta

Na ovaj način će se kreirati dodatni projekat u `Project Explorer` prozoru koji bi trebalo da sadrži module `Debug`, `Includes`, `C5545.cmd` i `main.c`. Kreirani glavni `C` fajl se može izbrisati ako se želi drugačija struktura projekta.

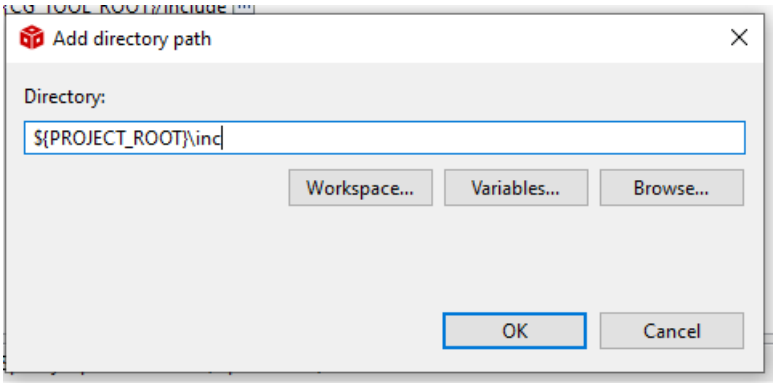
Često se organizacija projekta vodi u dva odvojena foldera u okviru foldera projekta - `src` i `inc`, tako da su u prvom sadržani implementacioni fajlovi (oni sa ekstenzijom `.c` ili `.asm`), dok su u drugom biblioteke (*header* - `.h` fajlovi).

Upravo ovakva organizacija postoji u primerima sa vežbi, pa se u cilju uspešnog uvezivanja fajlova (*linking*) mora specificirati putanja sa koje će kompajler čitati `.h` fajlove. Ovo se postiže tako što se u `Project -> Properties` doda/izmeni podrazumevana putanja fajlova, odnosno u opciji `Build -> C5500 Compiler -> Include Options` pritisne `Add...` dugme u prvom prozoru:



Slika 3.8: Dodavanje putanje biblioteka

U ponuđen prostor uneti `${PROJECT_ROOT}\inc`:



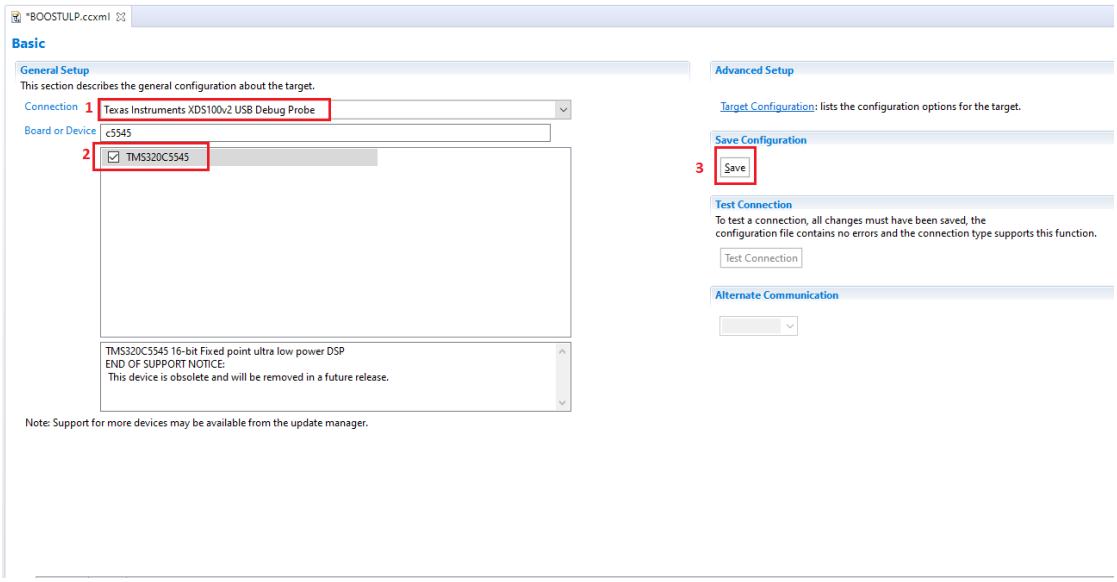
Slika 3.9: Dodavanje putanje biblioteka - nastavak

Nakon ovoga je moguće kreirati folder `inc` u folderu projekta putem fajl sistema operativnog sistema, ili putem razvojnog okruženja, tako što se desnim klikom na željeni projekat ode na `New -> Folder`, a potom odabere projekat u koji se želi dodati folder i specificira njegovo ime. Na isti način je moguće kreirati nove fajlove, s tim što je poželjno `.c` i `.h` fajlove dodavati opcijama `New -> Source File` i `New -> Header File`, respektivno, dok se asemblerski fajlovi mogu dodati opcijom `New -> File` i potom uneti naziv sa ekstenzijom `.asm`.

Na opisani moguće je dodati i upravljati `src` folderom i tako kreirati pregledan i funkcionalno izdeljen projekat. Neophodno je prethodno ukloniti `main.c` fajl ukoliko se želi ovakav način organizacije.

Po kompajliranju projekta neophodno je upisati izvršne fajlove u memoriju platforme, za šta je potrebno dodati fajl za konfiguraciju platforme (*target configuration file*). Najlakše ga je dodati tako što se desnim klikom na željeni projekat u

okviru `Project Explorer` prozora odabere `New -> Target Configuration File`, a potom unese željeno ime fajla. Ovo može biti proizvoljno, a preporuka je da ime asocira na ime platforme na koju će program biti spušten. Neophodno je da ekstenzija imena bude `.ccxml`. Nakon ovoga se otvara sledeći prozor, u okviru kog treba odabrati tip veze sa pločicom i sâm uređaj:



Slika 3.10: Konfigurisanje *target configuration* fajla

Sada je pritiskom na `Run -> Debug` moguće debagovati program, koji će se automatski upisati u memoriju platforme.

3.3 Primer sa SD kartice

Uvodni, `out-of-the-box` primer sa SD kartice, predstavlja složen projekat koji uvezuje veći broj biblioteka i modula, baziran je na operativnom sistemu u realnom vremenu i koristi gotovo sve periferije platforme kako bi demonstrirao sve njene hardverske mogućnosti.

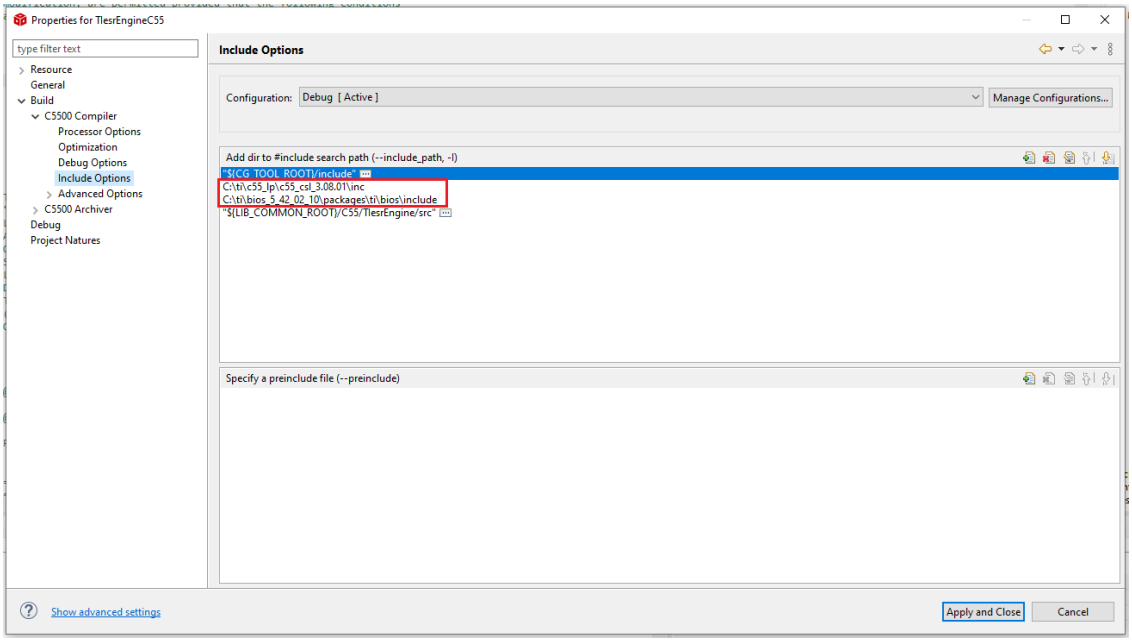
Primer je vrlo lako pokrenuti, ubacivanjem SD kartice i konfigurisanjem nekoliko kratkospajača na pločici, kao što je opisano u ovom kratkom video klipu. Nakon pokretanja se kroz meni programa jednostavno prolazi, a instrukcije su prikazane na OLED displeju.

Analizirati ili modifikovati program ovog primera je, međutim, nešto zahtevnije od postavke nekoliko kratkospajača.

Njegov kôd je dat na stanici platforme, pod sekcijom *Order & start development*. Preuzet `.zip` fajl treba otpakovati i pokrenuti aplikaciju koja se u njemu nalazi. Ona će u odabrani folder (preporuka da to bude `C:\ti\`) instalirati sve fajlove neophodne za analizu primera sa SD kartice uključujući firmver mikrokontrolera koji implementira *Bluetooth* komunikaciju, kôd Android aplikacije, ali i najbitnije, firmver C5545 procesora koji vrši svu obradu signala.

Za ispravno kompajliranje primera neophodno je preuzeti i DSP-BIOS operativni sistem verzije 5.42.1.09, ali i biblioteke za podršku rada procesora za digitalnu obradu signala, C55x CSL. Dalje treba pratiti uputstva iz sekcije *3.2.3 Build Setup* dokumenta C5545 BoosterPack Software User's Guide

Dodatna instrukcija koja nije navedena u softverskom uputstvu jeste da treba dodati dve putanje C55xx kompajleru kako bi se program pravilno uvezao. Ono što treba uraditi jeste da se u okviru postavki projekta `TIesrEngineC55 Project -> Properties -> Build -> C5500 Compiler -> Include Options` dodaju lokacije: `${C55XX_CSL_ROOT}/inc` i `${BIOS_INCLUDE_DIR}`. Kako će korisnik najverovatnije instalirati verziju *chip support library* i *DSP-BIOS* različitu od one na kojoj je primer napravljen, bolje je specificirati tačnu putanju do traženih biblioteka, kao što je to prikazano na sledećoj slici:



Slika 3.11: Konfigurisanje TIesrEngineC55 projekta

Ove lokacije se, naravno, mogu razlikovati u zavisnosti od toga gde je korisnik odlučio da instalira operativni sistem čipa i biblioteke za njegovu podršku.

Posle ove izmene treba se vratiti uputstvu i dovršiti kompajliranje po opisanom redosledu i zahtevima, nakon čega se program može spustiti na procesor i pratiti njegovo izvršavanje u debageru.

Glava 4

Opšte preporuke i napomene

Razvojno okruženje za procesor C5545 vrši nešto drugačiju podelu memorije u okviru svog komandnog fajla za linker (*linker command file*) u odnosu na C5505. Stoga je za pojedine projekte bilo neophodno povećati memoriju u koju se smeštaju određene sekcije.

Iz komandnog fajla projekta `19_hwFFT` se vidi da je za sekciju `.stack` rezervisana memorija sa dvostrukim pristupom `DARAM0` i `DARAM1`. Kako je poželjno da stek nije izdeljen po memoriji, odnosno da bude kontinualno raspoređen, onda je za specificiranje memorijskih lokacija u koje treba smestiti stek korišćen operator `>`, kao što se može videti na slici:

```
64
65 SECTIONS
66 {
67     vectors (NOLOAD)  >  VECS  /* If MPNMC = 1, remove the NOLOAD directive */
68     .cinit            >  DARAM0
69     /* Arbitrary assignment of memory segments to .text section.   */
70     /* Can be expanded or reduced observing limitations of SPRAA46 */
71     .text             >> SARAM0|SARAM1|SARAM2|SARAM3|SARAM4
72     .stack            >  DARAM0|DARAM1
73     .sysstack         >  DARAM0|DARAM1
74     .sysmem           >  DARAM3|DARAM4
75     .data             >  DARAM4
76     .cio              >> DARAM0|DARAM2
77     .bss              >  DARAM5
78     .const            >  DARAM0
79     data_br_buf       >  DARAM2
80     scratch_buf       >  DARAM2
81 }
```

Slika 4.1: Komandni fajl za linker - memorijske sekcije

Sekcija ulazno-izlaznih funckija (`C` *input-output - cio functions*), kao što je `printf` funkcija, je smeštena u memorije `DARAM0` i `DARAM2`, ali je operator dodele `> >`, što znači da linker sme ovu sekciju prekinuti i eventualno popuniti nekom drugom sekcijom, na primer `.stack` sekcijom.

Upravo ovaj primer koristi hardversku akceleraciju pri računanju brze Furijeove transformacije (FFT). Ova akceleracija se postiže softverski kontrolisanim koprocesorom koji računa Furijeovu transformaciju u broju tačaka koje predstavljaju stepena broja dva, koristeći *radix 2 Decimation in time (DIT) FFT* algoritam, a moguće je izračunati FFT od 8 pa sve do 1024 tačke. Koprocesor se koristi pozivanjem određenih funkcija koje su upisane u ROM C5545 procesora, a kojoj mogu pristupati i procesor i koprocesor. Funkcije imaju fiksnu adresu u memoriji i date su u sekciji *2.5.5 Project Configuration for Calling Functions from ROM* ovog korisničkog uputstva, datog na početku glave 2.

Kada se adrese funkcija dodaju u komandni fajl, njegov izgled je sledeći:

```
64
65 SECTIONS
66 {
67     vectors (NOLOAD)  >  VECS  /* If MPNMC = 1, remove the NOLOAD directive */
68     .cinit            >  DARAM0
69     /* Arbitrary assignment of memory segments to .text section.   */
70     /* Can be expanded or reduced observing limitations of SPRAA46 */
71     .text             >> SARAM0|SARAM1|SARAM2|SARAM3|SARAM4
72     .stack            >  DARAM0|DARAM1
73     .sysstack         >  DARAM0|DARAM1
74     .sysmem           >  DARAM3|DARAM4
75     .data             >  DARAM4
76     .cio              >> DARAM0|DARAM2
77     .bss              >  DARAM5
78     .const            >  DARAM0
79     data_br_buf       >  DARAM2
80     scratch_buf       >  DARAM2
81 }
82
83 /*** Add the following code to the linker command file to call HWAFFT Routines from ROM ***/
84 /* HWAFFT Routines ROM Addresses */
85 _hwafft_br = 0x00fefe9c;
86 _hwafft_8pts = 0x00fefeb0;
87 _hwafft_16pts = 0x00feff9f;
88 _hwafft_32pts = 0x00ff00f5;
89 _hwafft_64pts = 0x00ff03fe;
90 _hwafft_128pts = 0x00ff0593;
91 _hwafft_256pts = 0x00ff07a4;
92 _hwafft_512pts = 0x00ff09a2;
93 _hwafft_1024pts = 0x00ff0c1c;
94
```

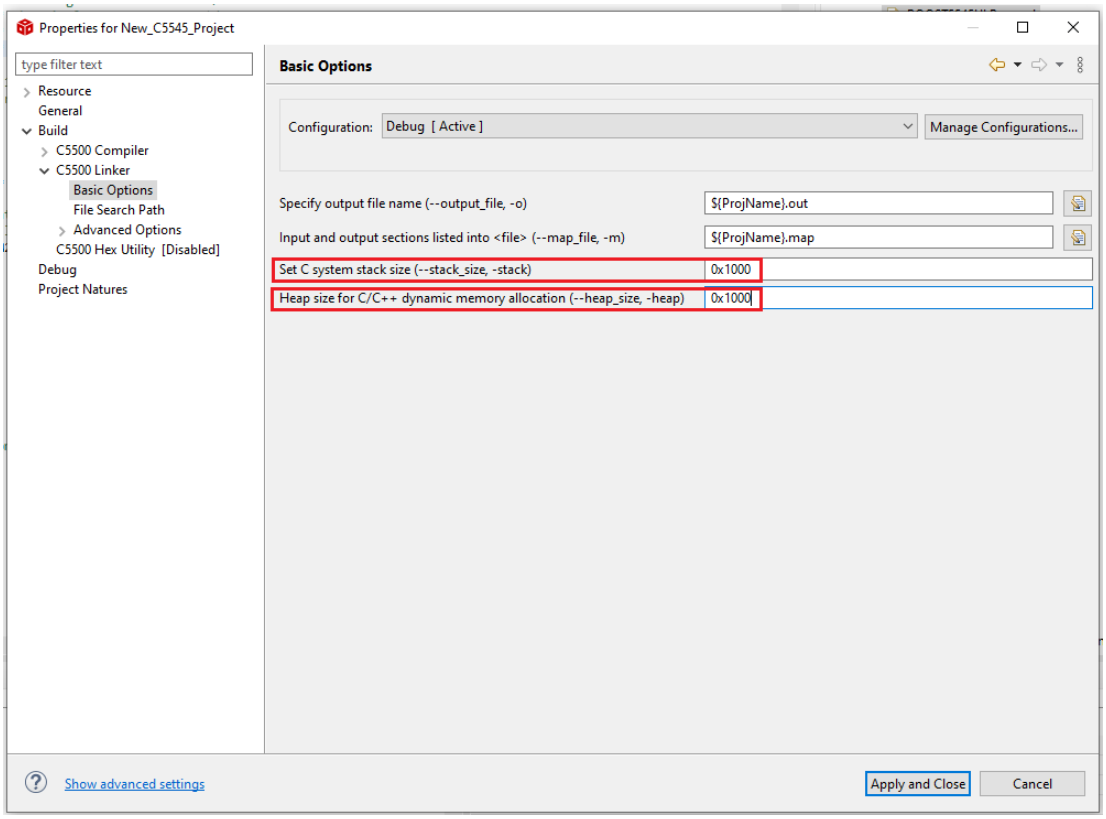
Slika 4.2: Komandni fajl za linker - funkcije akceleratora

Na početku dodate sekcije je i funkcija *bit reverse* koja je implementirana kako bi se brzo obavila neophodna inverzija ulaznih podataka potrebna za pravilno računanje transformacije.

Ukoliko je neophodno definisati sopstvenu sekciju u kôdu, kao što je to urađeno u opisanom primeru, onda treba te sekcije dodati u komandni fajl. Zato se u okviru dela kôda `SECTIONS` može uočiti korisnički definisana sekcija `data_br_buf`, kao i `scratch_buf`, koje je potrebno dodefinisati i smestiti u željeni deo memorije. Treba napomenuti da će kompajler samo dati upozorenje korisniku da sekcije nisu eksplicitno definisane i uspešno će kompajlirati program, a sekcije smestiti na slobodne lokacije.

Dodatna napomena vezana za `cio` funkcije jeste količina steka koju one koriste. Naime, iako često korišćena u takozvanim *printf debugging* metodama, lako se implementira na sistemu i pruža razumljiv interfejs, funkcija `printf` i njoj slične ulazno-izlazne funkcije zauzimaju veoma veliku količinu memorije i procesorskog vremena, stoga je poželjno izbegavati ih. Podrazumevana veličina steka prilikom kreiranja programa je 512 bajtova (0x200), a memorije za dinamičku alokaciju (*heap*) 1024 bajta (0x400). To je nedovoljno prostora za korišćenje ovakvih funkcija, pa je neophodno povećati stek i *heap* na 0x1000 odnosno na po 4096 bajtova. Mmoguće je dodeliti i manje memorije ovim sekcijama kako bi rad funkcija bio ispravan, ali za većinu primera procesor ima dovoljno memorije da bi se za stek i *heap* koristila ova količina memorije.

Veličina steka i *heap*-a se može podesiti u okviru podešavanja projekta, odabirom `Project -> Properties -> Build -> C5500 Linker -> Basic Options`, kako je prikazano:



Slika 4.3: Podešavanje veličine steka i *heap*-a

Vrlo je važno dobro podesiti veličine ovih sekcija, s obzirom na to da će se program uspešno kompajlirati, a kompajler neće korisnika upozoriti na nedostatak steka ili *heap*-a, pošto je ograničenje ovih sekcija obično poželjna funkcionalnost u *embedded* programima. Nedovoljna veličina za pozivanje ulazno-izlaznih funkcija `C` programa se može primetiti prilikom debagovanja, kada se program zamrzava na ovim funkcijama ili preko njih samo prelazi, ne izvršivši ih. Tada treba povećati ove sekcije na opisan način.

Glava 5

Dobijeni rezultati

Primeri sa vežbi su modifikovani primenjujući opisane metode kako bi mogli pravilno da se kompajliraju na novoj platformi. Svi modifikovani primeri se nalaze na ovoj lokaciji, na kojoj se nalaze i biblioteke - *Chip Support Library*. Rezultati koji se dobijaju za brzine izračunavanja bitnih DSP funkcija i operacija pri izvršenju primera sa vežbi su u skladu sa pruženom dokumentacijom, a ne razlikuju se previše u odnosu na to kada su bili pokretani na procesoru C5505.

FIR filtar

U nekoliko primera se vrši filtriranje ulaznog signala dužine 80 odbiraka FIR filtrom dužine 48 odbiraka. U zavisnosti od implementacije filtriranja, različito vreme je procesoru potrebno da ga izvrši, pa tako sledeća tabela ilustruje dobijene rezultate:

Tabela 5.1: Brzina filtriranja FIR filtrom.

Primer	Opis	Broj taktova [clk]
`05_fixed_point_block_FIR`	FIR filtar napisan u `C` programskom jeziku	147 093
`06_asm_BlockFIR`	FIR filtar napisan u asembleru	4 214
`07_symetric_BlockFIR`	Simetričan FIR filtar napisan u asemleru	2 311
`08_dualMAC_BlockFIR`	FIR filtar napisan u asembleru, korišćenjem dualne MAC jedinice	2 184

Iz navedene tabele se može zaključiti da se značajna ušteda dobija implementacijom funkcije za filtriranje u asembleru (skoro 35 puta brži kôd), dok se brzina povećava još dva puta ukoliko se koristi simetrično filtriranje ili korišćenje dualne MAC jedinice.

IIR filtar

Filtriranje IIR filtrom se koristi u pet primera sa vežbi. Kako je implementacija direktne-I forme različita od direktne-II gde se filtrira veći broj odbiraka signala

(blok podataka), ne može se meriti vreme izvršavanja funkcije za filtriranje. Korisnija informacija je vreme potrebno da se izvrši kompletno filtriranje signala sa sve propratnim operacijama (programske petlje, priprema ulaza i slično). Tako se dobija sledeća tabela:

Tabela 5.2: Brzina filtriranja IIR filtrom.

Primer	Opis	Broj taktova [clk]
`10_floatPoint_directIIR`	IIR filtar napisan u `C` programskom jeziku u direknoj-I formi, bez ograničenja dužine reči	36 032 257
`11_fixedPoint_directIIR`	IIR filtar napisan u `C` programskom jeziku u direknoj-I formi, sa ograničenom dužinom reči - Q15 format	5 607 987
`12_fixedPoint_cascadeIIR`	IIR filtar napisan u `C` programskom jeziku u direknoj-II formi, sa ograničenom dužinom reči - Q15 format	8 729 962
`13_intrinsics_IIR`	IIR filtar napisan u `C` programskom jeziku u direknoj-II formi, korišćenjem *intrinsic* funkcija za MAC jedinicu	5 119 098
`14_asmIIR`	IIR filtar napisan u asembleru u direknoj-I formi	1 318 049

Vreme potrebno za izračunavanje samih funkcija je mnogo manje - kod direktne-I forme je reda hiljade taktova (za primere 10 i 11 je to 4247 i 1088, redom), kod direktne-II reda stotine hiljada taktova (190 123 za primer 12 i 90 884 za primer 13), dok je u asemblerskoj implementaciji direktne-II forme (primer 14) ono smanjeno na nekoliko hiljada taktova - 4863.

Računanje DFT

Prilikom direktnog izračunavanja diskretne Furjeove transformacije dobijena je značajna ušteda u vremenu prelaskom iz `C` programskog jezika u asembler, kao što je prikazano:

Tabela 5.3: Brzina izračunavanja DFT.

Primer	Opis	Broj taktova [clk]
`16_floatingPoint_DFT`	Direktni DFT algoritam napisan u `C` programskom jeziku	276 943 881
`17_asm_DFT`	Direktni DFT algoritam napisan u asembleru	1 198 104

Kao što se vidi, ukoliko je neophodno koristiti direktno izračunavanje DFT, vrlo je korisno funkciju za tu operaciju napisati u asembleru, jer se konkretno, za DFT u 128 tačaka, dobija 230 puta brži kôd. Korisno je pogledati da li se i koja ušteda dobija izračunavanjem amplitude dobijenog spektra signala, kako je ona najčešće potrebna informacija u analizi spektra signala. Sledeća tabela ilustruje dobijene rezultate pri računanju amplitude spektra signala u 128 tačaka:

Tabela 5.4: Brzina izračunavanja amplitude spektra.

Primer	Opis	Broj taktova [clk]
`16_floatingPoint_DFT`	Računanje amplitude dobijenog spektra signala u `C` programskom jeziku	80 497
`17_asm_DFT`	Računanje amplitude dobijenog spektra signala u asembleru	216

Amplitudska karakteristika u 128 tačaka se izračunava oko 370 puta brže.

Računanje FFT

Brza Furijeova transformacija (*Fast Fourier Transform - FFT*) predstavlja algoritam za izračunavanje diskretne Furijeove transformacije koji donosi značajnu uštedu u korišćenom procesorskom vremenu. Ovaj često korišćen algortiam je kod proizvođača procesora za obradu signala izazvao potrebu za implementacijom raznih načina hardverskog ubrzanja njegovog izračunavanja. Tako je u procesoru C5545 kao i u prethodnom C5505 implementiran hardverski akcelerator za izračunavanje FFT algoritma u vidu koprocesora koji ima pristup memoriji deljen sa glavnim procesorom. Hardverski akcelerator izračunava FFT u broju tačaka koji je stepen broja 2 (*radix-2 DIT* algoritam), a mogući stepen je od 3 do 10 (o 8 do 1024 tačke). Tabela 5.5 prikazuje broj taktova za izračunavanje DFT putem FFT algoritma, sa i bez korišćenja akceleratora:

Tabela 5.5: Brzina izračunavanja DFT FFT algoritmom.

Primer	Opis	Broj taktova [clk]
`18_floatingPoint_FFT`	FFT algoritam napisan u `C` programskom jeziku bez korišćenja akceleratora	1 218 993
`19_hwFFT`	FFT algoritam napisan u `C` programskom jeziku koristeći akcelerator	627

Iz priloženog se mogu videti dve stvari. Prvo, FFT radix-2 algoritam je vrlo efikasan algoritam koji u `C` implementaciji pruža bolje rezultate nego direktno izračunavanje DFT u asembleru. Drugo, hardverski akcelerator unosi neverovatno ubzanje

sistema. Spektar signala u 128 tačaka FFT algoritmom se dobija oko 1900 puta brže koristeći akcelerator, dok je ubrzanje dobijeno korišćenjem akceleratora u odnosu na *brute-force* direktnu implementaciju u jeziku `C` oko 440 000 puta.

Ipak, kako bi odbirci signala bili poređani u ispravnom redosledu nakon poziva funkcija za FFT, potrebno je implementirati funkciju koja vrši određenu preraspodelu odbiraka ulaznog signala - *bit reverse* funkciju. Korisno je pogledati koliko se ubrzanje dobija korišćenjem akceleratora prilikom poziva ove funkcije, kako je ona takođe zapisana u ROM procesora:

Tabela 5.6: Brzina izračunavanja *bit reverse* funkcije.

Primer	Opis	Broj taktova [clk]
`18_floatingPoint_FFT`	Funkcija napisana u `C` programskom jeziku bez korišćenja akceleratora	8 561
`19_hwFFT`	Korišćenje funkcije akceleratora sadržane u ROM-u procesora	279

U primerima predstavljenim u ovoj sekciji filtrira se signal od 1664 odbirka, pa se funkcija za izračunavanje DFT u 128 tačaka poziva 13 puta. U primeru `18_floatingPoint_FFT` se ova operacija izvrši za oko 20 miliona taktova (uključuje pripremu ulaznih signala, *bit reverse* funkciju i samo računanje DFT). U primeru `19_hwFFT` se odlazi korak dalje, pa se za signal od 1664 odbirka računa DFT i inverzna DFT, sve u 120 hiljada taktova. Kako je takt procesora u svim primerima 100MHz, potrebna je 1ms da bi se ovakvom signalu izračunao spektar i potom signal vratio u vremenski domen radi daljih izračunavanja. Ako se na to doda vreme potrebno da se signal isfiltrira FIR filtrom od 48 odbiraka korišćenjem dualne MAC jedinice, dolazi se do zaključka da se signal u 1664 tačke može efikasno filtrirati za manje od 2ms.

Glava 6

Zaključak

Rezultati dobijeni pokretanjem primera sa vežbi ilustruju povod za korišćenje digtalnih procesora signala. Procesor C5545 je vrlo moćan predstavnik ove familije procesora, a platforma BOOST5545ULP pruža mnoštvo mogućnosti korisniku/studentu da testira svoje aplikacije koje imaju potrebu za nešto zahtevnijom obradom signala. Iz navedenog je njena upotreba na kursevima čiji je predmet izučavanja digitalna obrada signala veoma poželjna.

Pored ovoga, platforma poseduje mikrokontroler sa Cortex-M3 procesorom, pa je zbog toga potencijalno interesantna za korišćenje i na drugim kursevima na kojima se proučavaju integrisani računarski sistemi ili sistemi u realnom vremenu (na kontroleru se može instalirati na primer FreeRTOS, dok na procesoru C5545 može postojati DSP-BIOS, drugi tip operativnog sistema u realnom vremenu).

Kako C5545 procesor poseduje koprocesor koji ima ulogu hardverskog akceleratora, on može biti interesantan primer implementacije ovakvog modula čipa na kursevima na kojima se proučavaju digitalni VLSI sistemi.

Korišćenje biblioteka za podršku rada ovog procesora nije pokriveno ovim dokumentom, ali kako se u primeru sa SD kartice platforme te biblioteke koriste, njihova implementacija u pojekte i aplikacije ne treba da predstavlja nikakav poblem.