SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1935

Nove paradigme akvizicije podataka hibridnim sustavom

Antonija Marinović

Zagreb, veljača 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Ovaj rad je nastao u sklopu suradnje Zavoda za elektroničke sustave i obradu informacija (ZESOI) s Fakulteta elektrotehnike i računarstva te Ericsson Nikole Tesle d.d. na projektu „Poboljšanje karakteristika rada LTE radio pristupnih uređaja“ (ILTERA, engl. Improvements for LTE Radio Access Equipment).

Zahvaljujem se svom mentoru dr.sc. Damiru Seršiću, asistentu mag.ing. Tinu Vlašiću i mag.ing. Hrvoju Butini na stručnom vođenju, pomoći i strpljenju pri izradi ovog diplomskog rada. Iskreno hvala, onima koji su me podupirali sve ove godine, mojoj obitelji, dečku i prijateljima. Na kraju se želim zahvaliti svojim kolegama zbog kojih mi studentski dani ostaju u lijepom sjećanju.

SADRŽAJ

1. Uvod

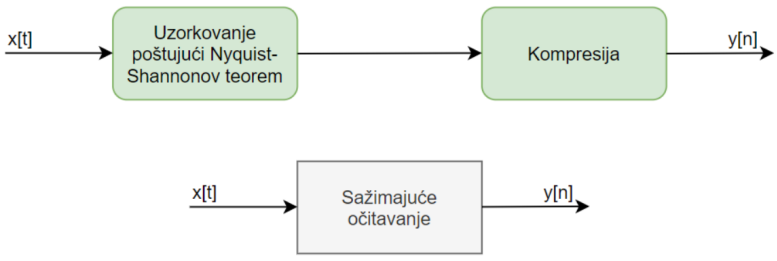
Komunikacijske tehnologije i sustavi za obradu slike, zvuka i videa nalaze se pod sve većim pritiskom jer moraju zadovoljiti rastuće zahtjeve današnjice među kojima su pohrana velikog broja podataka, brže uzimanje uzoraka i manja potrošnja energije. Temelj današnjih digitalnih sustava prikupljanja podataka je Nyquist-Shannonov teorem koji nalaže da frekvencija uzorkovanja signala bude dvostruko veća od maksimalne frekvencije prisutne u signalu ako se želi izbjeći gubitak podataka prilikom digitalizacije signala ili slike. Sklopovska ograničenja postojećih senzorskih sustava u kombinaciji s Nyquistovom frekvencijom stvorili su zid koji je trebalo prijeći nekim novim pristupom očitavanja signala. Tako se pojavila relativno mlada teorija sažimajućeg očitavanja koja na polju matematike i inženjerstva donosi nove paradigme akvizicije podataka i otvara put za razvoj velikog broja novih aplikacija. Princip teorije sažimajućeg očitavanja (engl. *Compressive Sensing*) u potpunosti zanemaruje zahtjeve Nyquist-Shannonovog teorem te predlaže akviziciju i rekonstrukciju signala iz puno manjeg broja uzoraka za signale koji su rijetki u nekoj bazi.

Cilj ovog rada je realizacija teorije sažimajućeg očitavanja u sustav koji bi radio sa stvarnim analognim signalima. U okviru diplomskog rada realiziran je hibridni analogno-digitalni sustav koji se sastoji od razvijenog akvizicijskog sklopa spojenog na razvojnu pločicu ZedBoard preko FMC konektora. Na početku rada ukratko je objašnjena teorijska pozadina koja stoji iza principa sažimajućeg očitavanja. U radu je opisana implementacija akvizicijskog sustava u razvojnoj okolini *Vivado Design Suite*. Pritom se u najvećem dijelu diplomskog rada opisuje implementacija paralelne i serijske komunikacije pomoću koje se odvija prijenos podataka između programabilne FPGA logike i procesorskog sustava na ZedBoard-u i obrnuto. Također, u radu je opisano podizanje operacijskog sustava Linux na Zynq-7000 procesorskom sustavu, konkretno PetaLinux inačica namijenjena za ciljanu platformu ZedBoard. Time je omogućeno izvođenje realizirane aplikacije na PetaLinux-u kojom se upravlja radom cijelog sustava.

2. Princip sažimajućeg očitavanja

Tradicionalni pristupi uzorkovanja signala poštuju Nyquist-Shannonov teorem koji postavlja zahtjev na frekvenciju uzorkovanja. Frekvencija uzorkovanja mora biti dvostruko veća od najveće frekvencije u signalu ukoliko se signal želi jednoznačno rekonstruirati. Uzorkovanjem vremenski kontinuiranog signala uz zadovoljen uvjet o frekvenciji ponekad nastaje previše uzoraka u odnosu na informacijski sadržaj samog signala. Osim spomenutog zahtjeva, tradicionalni pristup akvizicije rijetkih signala podrazumijeva kompresiju signala računanjem potrebnih koeficijenata transformacije za sve uzorke. Zbog ograničenja veličine memorije zadržavaju se samo dovoljno veliki koeficijenti, dok se manji koeficijenti odbacuju.

S druge strane metoda sažimajućeg očitavanja stavlja naglasak na svojstvo rijetkosti signala kojeg tradicionalni principi akvizicije signala zanemaruju. Rijetkim signalom može se proglasiti signal kojem je cijela informacija sadržana u najznačajnijim komponentama signala u odnosu na cijelu duljinu signala, dok su ostali uzorci jednaki nuli. Na svojstvo rijetkosti nadovezuje se svojstvo kompresibilnosti odnosno svojstvo da signal u transformacijskoj bazi sadrži nekoliko značajnih koeficijenata koji nose glavninu energije signala, dok se ostali koeficijenti zanemaruju. Nije nužno da signali imaju svojstvo rijetkosti ili kompresibilnosti u originalnoj domeni. Oslanjajući se na spomenuta svojstva, metoda sažimajućeg očitavanja omogućava uzorkovanje signala frekvencijom puno nižom od Nyquistove granice. Također, korištenjem raznih algoritama moguće je rekonstruirati originalni signal iz samo nekoliko uzoraka [10].



Slika 2.1: Usporedba tradicionalnog pristupa uzorkovanja i sažimajućeg očitavanja

2.1. Akvizicijski i rekonstrukcijski model

Matematički akvizicijski sustav sažimajućeg očitavanja može biti zapisan matričnom jednadžbom:

$$y = \varphi x \tag{2.1}$$

gdje $x \in \mathbb{R}^n$ ili $\mathbb{C}^n$ označava ulazni signal duljine n, $\varphi \in \mathbb{R}^{m \times n}$ predstavlja mjernu matricu dimenzija $m \times n$, a mjerni vektor dimenzija m predstavljen je kao $y \in \mathbb{R}^n$ ili $\mathbb{C}^n$. Mjerni rezultati nastaju kao produkt množenja ulaznog signala i mjerne matrice. Dobiveni mjerni rezultati imaju dimenzije puno manje od ulaznog signala ($m \ll n$) te su proporcionalni rijetkosti ulaznog signala.

Mjerni vektor y i rekonstrukcijska matrica $\Theta \in \mathbb{R}^{m \times n}$ ili $\mathbb{C}^{m \times n}$ ulazi su rekonstrukcijskog algoritma. Relacije koje vrijede su:

$$\Theta = \varphi\psi \tag{2.2}$$

$$y = \varphi x = \varphi\psi s = \Theta s \tag{2.3}$$

gdje je ψ matrica vektora u novoj bazi, a $s \in \mathbb{R}^n$ ili $\mathbb{C}^n$ predstavlja signal x u bazi ψ. Signal x može se rekonstruirati rješavanjem jednadžbe koja slijedi iz relacije 2.1

$$x = y\varphi^{-1} \tag{2.4}$$

čineći pritom nedeterminirani sustav linearnih jednadžbi koje imaju beskonačno mnogo rješenja. Cilj rekonstrukcije signala je pronaći maksimalno rijetko rješenje i upravo zbog tog se kao jedno od rješenja nameće korištenje ℓ_1 norme. Norma ℓ_2 se odbacuje jer se njome ne dobiva rijetko rješenje, dok se ℓ_0 norma odbacuje jer je presložena za računanje. Optimizacijski problem primjenom minimizacije ℓ_1 norme zapisuje se kao:

$$\hat{s} = arg\min_s \|s\|_1 \quad \text{uz uvjet } \Theta s = y. \tag{2.5}$$

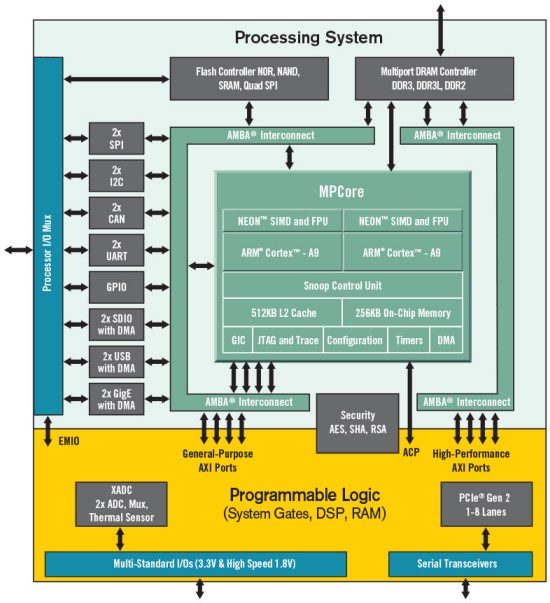
Prednost ℓ_1 norme je što se za njeno računanje mogu koristiti efikasni optimizacijski algoritmi [10].

3. Zynq-7000 SoC arhitektura i alati

Ugradbeni računalni sustavi svakim danom postaju sve složeniji. Tipične komponente kao što su senzori, procesori posebne namjene kombiniraju se zajedno s procesorima opće namjene pritom tvoreći složenije sustave na čipu (engl. *SoC – System on Chip*). Sustavi na čipu često kombiniraju i nadilaze tradicionalne granice između sklopovske potpore (engl. *hardware*) i programske potpore (engl. *software*) što od inženjera zahtijeva znanje mnogih koncepata i načela za realiziranje efikasnih rješenja. Dizajn modernog sustava na čipu svojom snažnom i modularnom arhitekturom omogućuje podizanje i izvođenje operacijskog sustava. Kako bi ostvarili izvođenje željene aplikacije na realiziranom sustavu podignut je PetaLinux koji predstavlja komercijalnu ugradbenu inačicu Linux distribucije kojeg je Xilinx razvio za svoje silicijske sustave.

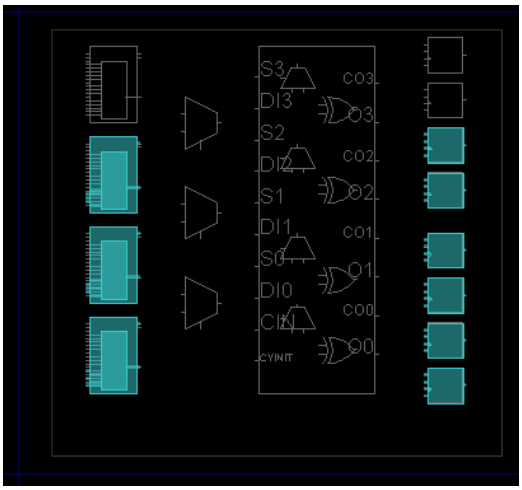
3.1. ZedBoard

ZedBoard razvojni sustav koristi Xilinx Zynq-7000 SoC arhitekturu čija je blok shema prikazana na slici 3.1. Čip se može podijeliti u dva dijela – procesorski i programabilni. Programabilni dio čini programabilna logika PL (engl. *Programmable Logic*) s 85 000 programabilno logičkih blokova, dok se procesorski dio sastoji od dvojezgrenog sustava ARM Cortex – A9 PS (engl. *Processing System*). Ovaj razvojni sustav sadrži potrebna sučelja i funkcije za ostvarivanje velikog broja višenamjenskih aplikacija. Velike mogućnosti proširenja funkcionalnosti ZedBoard platformu čini idealnom za razvoj prototipnih ugradbenih i konceptualnih sustavnih dizajna namijenjen ne samo iskusnim dizajnerima već i početnicima. Neke od karakteristika koje odlikuju Zynq-7000 SoC arhitekturu su: procesorski sustav podržan s 512 MB dinamičke DDR memorije, vanjski izvori takta osiguravaju procesorskom sustavu rad na 33.33 MHz, dok programabilni dio radi na 100 MHz. Podržan je i 10/100/1000 Gigabitni Ethernet protokol kao i USB-JTAG programiranje.



Slika 3.1: Blok shema Zynq-7000 [1]

Programabilna logika sastoji se od memorijskih RAM blokova (engl. *Block Random Access Memory, BRAM*), konfigurabilnih logičkih blokova (engl. *Configure Logic Block, CLB*), programabilnih ulazno-izlaznih blokova (engl. *Input Output Block, IOB*), množila opće namjene (engl. *General Purpose Multiplier*), kontrolera takta i DSP (engl. *Digital Signal Processing*) odsječaka za efikasnu obradu signala. Za povezivanje odvojenih cjelina procesora, periferija i programabilne logike koristi se AMBA-AXI sabirnički sustav (engl. *Advanced Micro-controller Bus Architecture - Advanced Extensible Interface*) [4], [5]. Na slici 3.2 prikazan je primjer korištenog odsječka nekog implementiranog dizajna u razvojnom okruženju *Vivado Design Suite*.



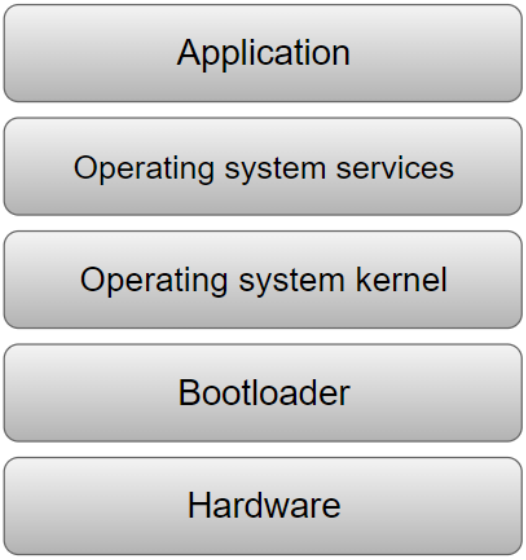
Slika 3.2: Odsječak implementiranog dizajna

3.2. Alati za razvoj digitalnih sustava

Povećanjem složenosti digitalnog sklopovlja pojavila se potreba za formalnim opisivanjem digitalnih elektroničkih sustava. Tako su razvijeni računalom potpomognuti alati koji na temelju opisanog digitalnog sustava omogućuju programiranje programabilnih blokova u skladu sa zadanim opisom. Za formalno opisivanje i modeliranje digitalnog sklopovlja razvijeno je mnogo jezika za opis sklopovlja HDL (engl. *Hardware Description Language*) među kojima se po učestalosti upotrebe izdvajaju VHDL, Verilog i SystemVerilog. VHDL jezik koji se u ovom diplomskom radu koristi za opis sklopovlja dolazi od skraćenice *VHSIC Hardware Description Language* u kojoj se prvi dio kratice odnosi na *Very High Speed Integrated Circuit*. Tvrtka Xilinx za svoje razvojne sustave osigurava razvojne alate za prilagodbu sklopovlja i za stvaranje programske podrške. Za Zynq-7000 generaciju koriste se razvojna okruženja *ISE Design Suite - WebPack* s paketom XPS (engl. *Xilinx Platform Studio*) te *Vivado Design Suite*. Osim mogućnosti prilagodbe sklopovlja oba alata nude mogućnost razvoja programske podrške pomoću alata Xilinx SDK (engl. *Software Development Kit*) koji ne zahtijeva dodatnu instalaciju već dolazi unutar istog paketa [9].

3.3. Linux na ZedBoard-u

Operacijski sustav Linux osigurava podršku za različite uređaje i konfiguracije te nudi raznovrsne mogućnosti za ugradbene računalne sustave. S obzirom na to da je izvorni kod javan svatko može prilagoditi jezgru prema potrebama ciljanog sustava. Kao što je prikazano na slici 3.3 cijeli Linux sustav može se podijeliti na 5 komponenata: sklopovlje, univerzalni programski podizač (engl. *Bootloader*), jezgru operacijskog sustava (engl. *kernel*), usluge operacijskog sustava (engl. *Operating system services*) i korisničke aplikacije (engl. *User Application*). Zbog kompleksnosti cijelog operacijskog sustava u nastavku će ukratko biti objašnjene komponente Linux operacijskog sustava. Univerzalni programski podizač *bootloader* u RAM memoriju (engl. *Random Access Memory*) učitava jezgru iz trajne memorije prilikom uključivanja računala. Jezgra operacijskog sustava je posrednik između sklopovlja i programa. Jezgra upravlja programskim zahtjevima prevodeći instrukcije za procesor (engl. *Central Processing Unit*). *Kernel* operacijskog sustava čini pet modula: modul upravljanja memorijom (engl. *Memory Managment*), modul rasporeda procesa (engl. *Process Schedulling Managment*), modul za komunikaciju između procesa (engl. *Inter-Process Communication*), modul mrežnog sučelja (engl. *Network Interface*) te modul virtualnog organi-



Slika 3.3: Linux osnovne komponente

ziranja podataka (engl. *Virtual File System*). Usluge operacijskog sustava opslužuju programe korisničkog sučelja. Na vrhu hijerarhije Linux operacijskog sustava stoje korisničke aplikacije [7].

Kao što je već spomenuto komercijalna ugradbena inačica Linux distribucije naziva se PetaLinux koja omogućuje izvođenje Linux operacijskog sustava na ZedBoard-u. Za razvoj Linux ugradbene aplikacije na računalu potrebno je imati podignut Linux operacijski sustav. Zbog zahtjeva za kompatibilnošću verzija s razvojnim okruženjem Vivado 2018.3 koji je korišten za digitalni dizajn sklopovlja sa službenih stranica Xilinx tvrtke bilo je potrebno preuzeti verziju PetaLinux 2018.3[1]. PetaLinux zahtijeva instaliranje određenog broja standardnih razvojnih alata i biblioteka. Najbrži put za dohvat potrebnih alata i biblioteka je sljedeća naredba:

```
sudo apt-get install -y gcc git make net-tools libncurses5-dev tftpd
zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget diffstat
chrpath socat xterm autoconf libtool tar unzip texinfo zlib1g-dev
gcc-multilib build-essential libsdl1.2-dev libglib2.0-dev zlib1g:
    i386
```

Potrebno je postaviti *bash* ljusku (engl. *shell*). Osnovna zadaća ljuske je interpretacija naredbi koje je upisao korisnik odnosno koristi se kao sučelje između korisnika i računala. Na većini Linux distribucija već je zadana predefinirana ljuska *dash*. Za postavljanje potrebne *bash* ljuske u terminal je potrebno upisati sljedeću naredbu:

[1]https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2018-3.html

```
sudo dpkg-reconfigure dash
```

Postoje dva pristupa podizanja PetaLinux distribucije: stvaranje potpuno novog projekta ili korištenje referentnog dizajna navedenog u paketu podrške za ciljanu platformu BSP (engl. *Board Support Package*). Podizanje programa koji se izvodi na Petalinux-a stvaranjem novog projekta može se podijeliti na tri bloka: dizajn sklopovlja, razvoj programske podrške te razvoj aplikacije. Prilikom izrade ovog diplomskog rada s podizanjem operacijskog sustava PetaLinux krenulo se od referentnog dizajna pod nazivom ZED BSP koji je dohvaćen sa službenih stranica tvrtke Xilinx.

3.3.1. Podizanje PetaLinux distribucije

BSP-ovi uključuju sve potrebne konfiguracijske datoteke, unaprijed ugrađene i testirane slike sklopovlja i programske potpore. BSP je ključan za konfiguriranje *kernela* odnosno opisuje sklopovlje i značajke koje to sklopovlje podržava. Stvaranje novog projekta iz BSP-a je najjednostavniji način podizanja PetaLinux-a jer nije potrebno prolaziti opsežan proces generiranja BOOT.BIN slike koja je potrebna za podizanje operacijskog sustava na SoC-u. BOOT.BIN slika čini izravnu vezu operacijskog sustava i ciljanog sklopovlja. Za podizanje operacijskog sustava koristi se SD kartica veličine 8 GB koja se stavlja u odgovarajući fizički priključak ZedBoard razvojne pločice. BOOT.BIN slika jedna je od datoteka Linux distribucije koja se prebacuje na SD karticu. U nastavku će biti opisani koraci podizanja PetaLinux distribucije na ZedBoard. Prvo je potrebno postaviti radno okruženje:

```
$ source <path-to-installed-PetaLinux>/settings.sh
```

Zatim je potrebno kreirati novi PetaLinux projekt:

```
$ petalinux-create --type project --template <PLATFORM> --name <
   PROJECT_NAME>
```

Parametar PLATFORM podržava platforme zynqMP (za UltraScale + MPSoC), zynq (za Zynq) i microblaze (za Microblaze), a umjesto parametra PROJECT_NAME upisuje se proizvoljno ime projekta. Potrebno je pozicionirati se unutar stvorenog projekta.

```
$ cd <plnx-proj-root>
```

PetaLinux alat za izgradnju sustava gradi cijeli ugradbeni Linux sustav ili određenu komponentu Linux sustava. Specifičnosti ove naredbe mogu se diktirati putem *petalinux-build* i *petalinux-c naredbe*.

```
petalinux-build -c bootloader -x distclean
```

Potrebno je izmijeniti konfiguracijski izbornik za PetaLinux projekt. Nakon pokretanja sljedeće naredbe otvara se još jedan prozor u kojem se nude razne mogućnosti konfiguriranja sustava. Kako je za podizanje PetaLinux operacijskog sustava korišten BSP nije bilo potrebno ništa mijenjati jer su postavke u konfiguracijskom izborniku inicijalno ispravno postavljene.

```
petalinux-config -c kernel
```

Sljedeći korak je generiranje DTB datoteke stabla (engl. *Device Tree Blob*), *bootloader*, U-Boot, Linux *kernel* i slike korijenskog datotečnog sutava. U konačnici generira se spomenuta BOOT.BIN slika. Potrebno je pokrenuti sljedeće dvije naredbe ovim redoslijedom:

```
petalinux-build
petalinux-package --force --boot --fsbl images/linux/zynq_fsbl.elf
    --u-boot images/linux/u-boot.elf
```

Nakon uspješnog generiranja datoteka potrebno je prebaciti datoteke BOOT.BIN i image.ub datoteke na SD karticu sljedećim naredbama.

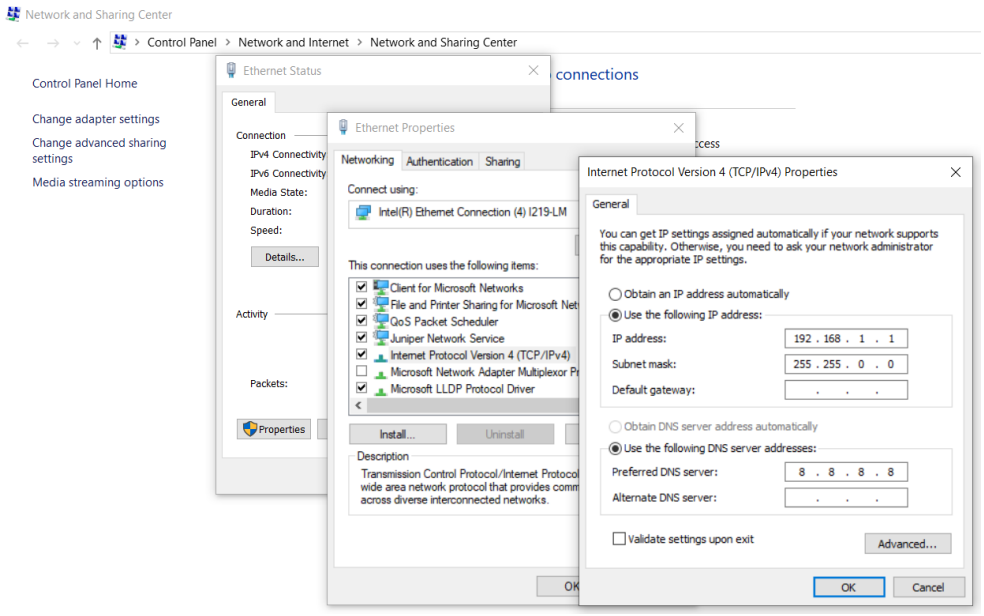
```
$ cp images/linux/BOOT.BIN /media/BOOT/
$ cp images/linux/image.ub /media/BOOT/
```

Za početnika koji se prvi put susreće s podizanjem PetaLinux distribucije preporučuje se proučavanje odgovarajućih službenih Xilinx dokumenata u kojima je detaljno opisana svaka naredba kao i cijela struktura PetaLinux-a [7], [15], [16].

3.4. Pokretanje aplikacije

Razvoj programske podrške namijenjene Xilinx ugradbenim sustavima odvija se unutar alata Xilinx SDK. U ovom radu aplikacije su pisane u C++. Nakon što je aplikacija gotova i ispravno napisana, pokretanjem naredbe *Build project* SDK alat generira izvršnu datoteku s ekstenzijom *elf* koja sadrži izvršnu sliku CPU koda. *Elf* datoteku potrebno je prebaciti na SD karticu kako bi se omogućilo njeno izvođenje. Prvi korak je povezivanje osobnog računala i ZedBoard-a preko Ethernet kabela SSH protokolom (engl. *Secure Shell*). SSH je mrežni protokol koji korisnicima omogućuje uspostavu sigurnog komunikacijskog kanala između dva računala preko nesigurnog medija kao što je Internet. Potrebno je postaviti statičke IP adrese (engl. *Internet Protocol*) osobnog računala i razvojnog sustava ZedBoard. Na osobnom računalu koji podržava Windows 10 operacijski sustav treba pronaći *Internet Protocol Version 4* postavke i postaviti IP

adresu kao što je to prikazano na slici 3.4. U ovom slučaju postavljena IP adresa je 192.168.1.1.



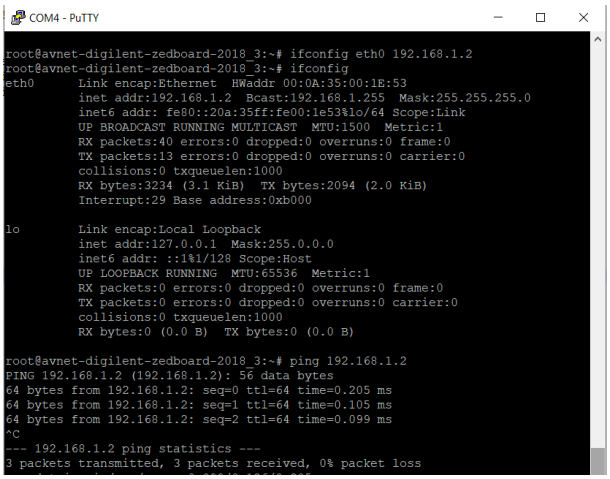
Slika 3.4: Postavljanje statičke IP adrese osobnog računala

Postavljanje statičke adrese razvojnog sustava Zedboard vrši se preko Putty ili nekog drugog odgovarajućeg terminala. IP adresa postavlja se unutar lokalne mreže

```
ifconfig eth0 192.168.1.2
```

Za provjeru veze te vidljivosti računala i ZedBoard razvojnog sustava međusobno koristi se naredba

```
ping 192.168.1.2.
```

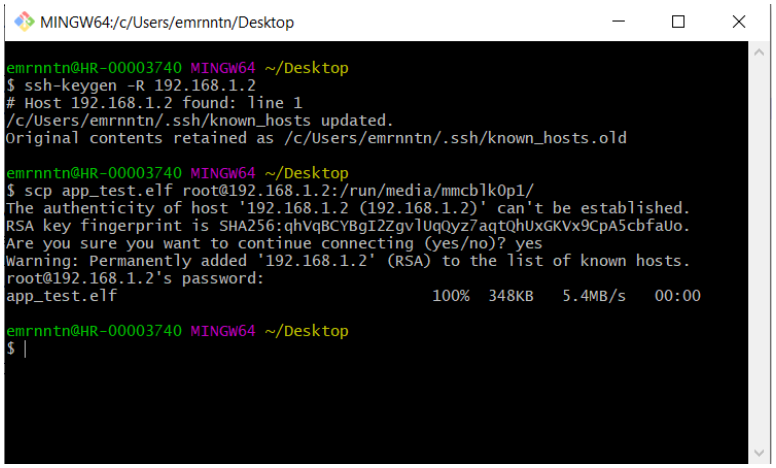


Slika 3.5: Postavljanje statičke IP adrese ZedBoard-a

Usporedbom poslanih i primljenih podataka i postotkom izgubljenih podataka donosi se zaključak o uspješnosti veze.

Sljedeći korak je pozicioniranje u *Git Bash*, aplikaciji za okruženje zasnovano na Windows operacijskom sustavu, unutar direktorija u kojem se nalazi izvršna *elf* datoteka. SSH protokol koristi kriptografiju javnog ključa za provjeru autentičnosti domaćina i korisnika. Ključevi za provjeru autentičnosti, poznati pod imenom SSH ključevi, stvoreni su pomoću *keygen* programa.

```
ssh-keygen -R 192.168.1.2
```



Slika 3.6: Prebacivanje *elf* datoteke

Scp je uslužni program naredbenog retka koji omogućava sigurno kopiranje mapa i datoteka između dvije lokacije. Sljedećom naredbom *elf* datoteka prebacuje se na SD karticu.

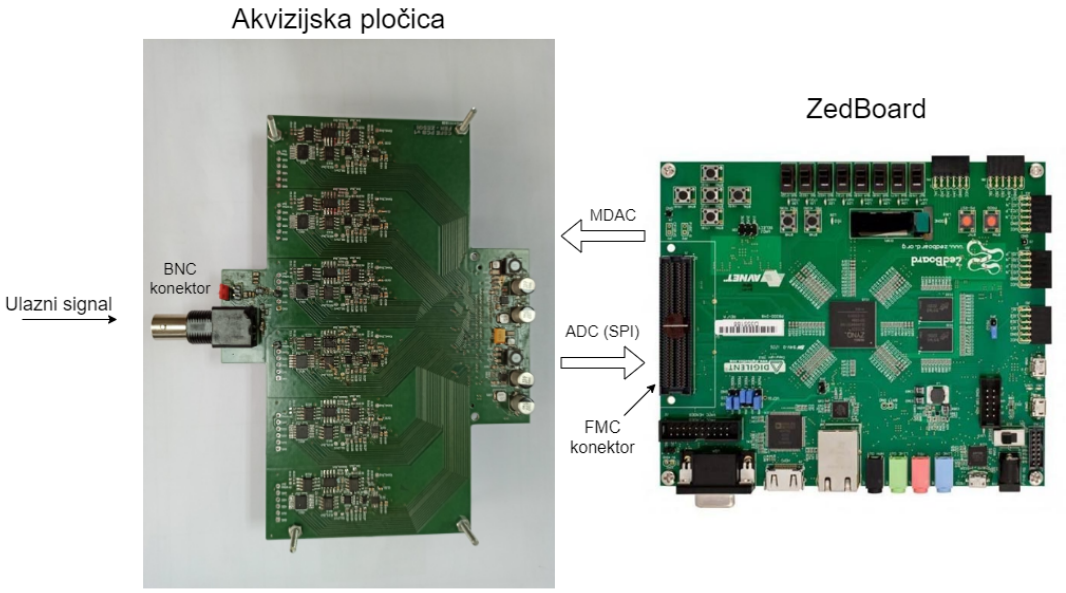
```
scp app_test.elf root@192.168.1.2:/run/media/mmcblk0p1/
```

Umetanjem SD kartice u odgovarajući utor na ZedBoardu i naredbom

```
.\app_test.elf
```

u *Putty* terminalu pokreće se željena aplikacija.

4. Sustav sažimajućeg očitavanja



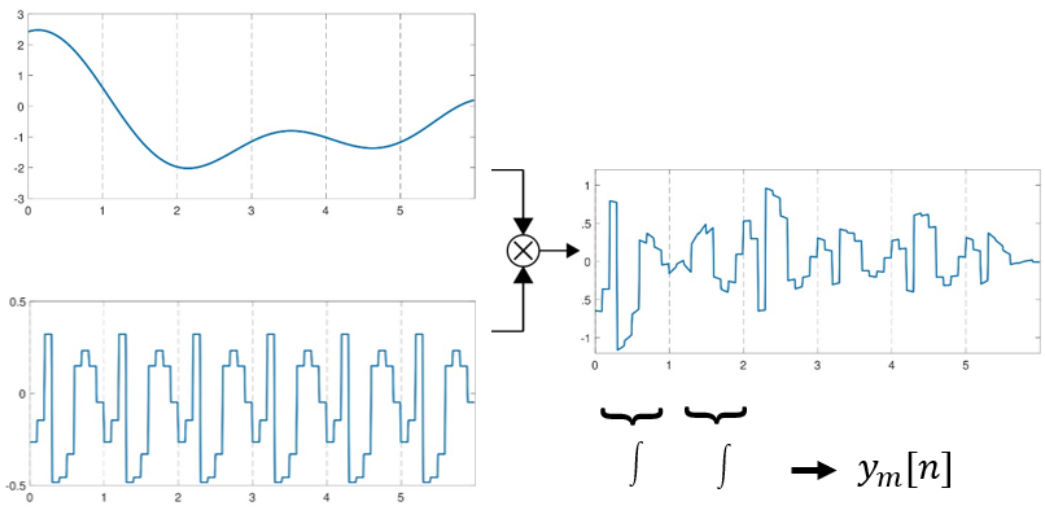
Slika 4.1: Sklopovlje akvizicijskog sustava

Nakon istraživanja teorije sažimajućeg očitavanja bilo je potrebno razviti sustav kojim bi se dokazalo da je moguće realizirati teoriju sažimajućeg očitavanja. Realizirani sustav dijeli se na dvije cjeline: ZedBoard i akvizicijsku pločicu. Kao što je prikazano na slici 4.1 akvizicijska pločica se preko FMC (engl. *FPGA Mezzanine Card*) konektora spaja na ZedBoard. Ulazni signal dovodi se na akvizicijsku pločicu preko BNC konektora. Akvizicijski sustav sastoji se od *front-end* dijela i *back-end* dijela koji je realiziran na Zynq-7000 SoC. *Front-end* dio sastoji se od šest paralelnih kanala na koje se dovodi ulazni signal. Sustav je projektiran u razvojnoj okolini *Altium Designer* u sklopu diplomskog seminara [12]. Osnovna funkcija akvizicijskog sklopa je množenje ulaznog signala s retkom mjerne matrice koja se generira iz procesorskog dijela razvojnog sustava ZedBoard. U nastavku će ukratko biti opisana akvizicijska pločica kao i tok signala kroz jedan kanal. Rad svake komponente detaljnije je opisan u poglavlju *Ispitivanje i rad kanala* u [6]. U ovom diplomskom radu naglasak je na *back-end* di-

jelu odnosno na implementiranom digitalnom dizajnu na Zynq-7000 koji je opisan u sljedećem poglavlju.

4.1. Akvizicijska pločica

Jedna od osnovnih funkcija akvizicijskog sklopa je množenje ulaznog signala s retkom mjerne matrice. Cijela ideja je prikazana na slici 4.2 gdje se ulazni signal množi s periodičnim valnim oblikom perioda T koji u ovom radu iznosi 1 μs. Svakih 100 ns valni oblik poprima neku vrijednost u rasponu [-1, 1] te se taj slijed od deset razina ponavlja svake 1 μs. Valni oblici s kojim se množi ulazni signal razlikuju se za svaki kanal.



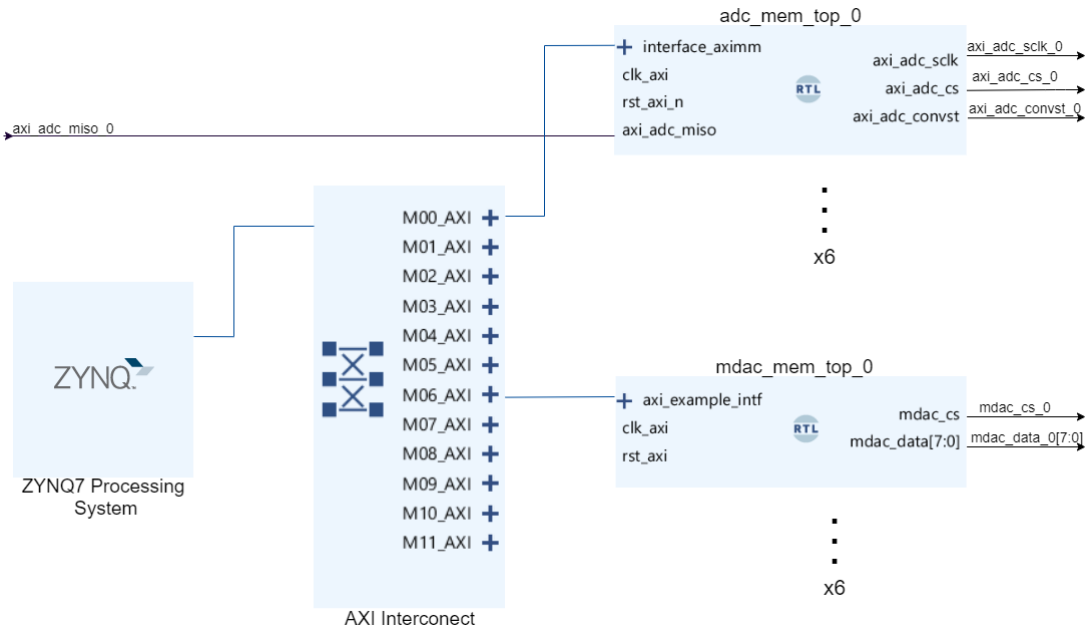
Slika 4.2: Množenje ulaznog signala s periodičnim valnim oblikom [11]

Svaki od šest kanala koji se nalazi na akvizicijskoj pločici sastoji se od digitalno-analognog pretvornika, integratora, sklopa za prilagodbu signala i analogno-digitalnog pretvornika. Zbog istog načina rada svih šest kanala u nastavku će biti opisan put signala kroz samo jedan kanal. Prvo se ulazni signal dovodi na pretpojačalo AD8067 kojim se podešavaju potrebne razine za rad digitalno-analognog pretvornika. Digitalno-analogni pretvornik AD5424 nalazi se na početku kanala i njegova uloga je množenje signala s 8-bitnom vrijednosti koje predstavljaju spomenuti redak mjerne matrice. Nakon množenja signal se prosljeđuju na integrator. Na izlazu iz integratora svaku 1 μs dobiva se jedan uzorak mjernog vektora. Zatim se nad signalom izvršava prilagodba naponske razine kako bi se iskoristio raspon napona pune skale analogno-digitalnog

pretvornika. U radu je korišten 12-bitni sukcesivno aproksimativni analogno-digitalni pretvornik AD7091R [2] koji koristi serijsko sučelje odnosno SPI (engl. *Serial Peripheral Interface*) za komunikaciju s razvijenim FPGA sklopom koji radi prihvat podataka. Za ispravni rad digitalno-analognog i analogno-digitalnog pretvornika u kanalu bilo je potrebno implementirati sklopovsku podršku u programabilnoj logici ZedBoard razvojnog sustava koja će biti opisana kasnije. Osim tog, trebalo je ostvariti paralelnu komunikaciju kojom se omogućava slanje mjerne matrice iz procesorskog sustava razvojnog sustava ZedBoard i serijsku komunikaciju za slanje mjernih vektora s akvizicijskog sklopa prema procesorskom sustavu ZedBoard.

5. FPGA digitalni dizajn

Kao i kod uhodavanja kanala ispitivanje i razvoj digitalnog dizajna pojedinih komponenti u FPGA tehnologiji odvijao se postupno. Prvo je napisan i ispitan potreban digitalni dizajn za svaku komponentu pojedinačno kao i za svaki kanal posebno. Bilo je potrebno generirati upravljačke signale za digitalno-analogne pretvornike kao i za analogno-digitalne pretvornike. Slika 5.1 prikazuje pojednostavljenu blokovsku shemu realiziranog akvizicijskog sustava. Iz procesorskog sustava ZedBoard šalje se mjerna matrica koju čini set od 6$\times$ 10 izlaznih 8-bitnih brojeva. Podatke je trebalo pohraniti u dvopristupnu memoriju na FPGA strani čija je realizacija opisana u sljedećem potpoglavlju. FPGA razvijeni sklop treba posluživati digitalno-analogne pretvornike 8-bitnim podacima. Na izlazu iz akvizicijskog sklopa svake 1 μs generira se šest 12-bitnih podataka koji predstavljaju mjerni vektor. Za taj set podataka bilo je potrebno ostvariti memoriju u koju će se podaci privremeno pohranjivati i kojima je moguće neometano pristupati iz PS-a.



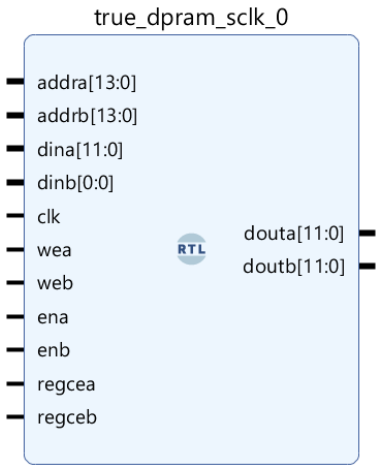
Slika 5.1: Pojednostavljena shema digitalnog dizajna

Adc_mem_top_0 i *mdac_mem_top_0* prikazani na slici 5.1 predstavljaju pojednostavljene blokove unutar kojih su ostvarene tražene funkcionalnosti. Kako bi FPGA razvijeni sklop mogao komunicirati s procesorskim sustavom na ZedBoard-u bilo je potrebno ostvariti odgovarajući komunikacijski protokol. Na slici 5.1 vidljivo je da je za Zynq-7000 sklopovlje komunikacija između PS i PL-a ostvarena preko AXI protokola.

5.1. Dvopristupna RAM memorija

Korištenjem blok RAM-ova mogu se pohraniti velike količine podataka. Svaki Zynq-7000 SoC uređaj ima između 60 i 456 namjenskih dvopristupnih blok RAM-ova, a svaki konfigurabilni blok RAM može pohraniti do 36 Kbit-a podataka. Inačica 7z020 nudi 140 blok RAM-ova na raspolaganju. Svaki blok RAM ima dva potpuno nezavisna porta koji ne dijele ništa osim pohranjenih podataka. Ovisno o zahtjevima za veličinu memorije ulazi se mogu konfigurirati kao 32K×1, 16K×2, 8K×4, 4K×9, 2K×18, 1K×36, 512×72. Svaki blok RAM može biti podijeljen na dva potpuno neovisna blok RAM-a koji mogu biti konfigurirani u rasponu od 16K×1 do 512×36. U ovom diplomskom radu implementirane memorije za operacije čitanja i pisanja koriste zajednički takt.

Budući da u razvojnoj okolini *Vivado Design Suite* ne postoji IP (engl. *Intellectual Property*) blok koji opisuje dvopristupnu memoriju bilo je potrebno napisati vlastitu digitalnu komponentu dvopristupne memorije. Realizirana je dvopristupna memorija koja za oba porta koristi isti brid takta. Podaci se mogu pisati i čitati s oba porta. Postoje tri pristupa pisanja u memoriju koja određuju koji podatak se postavlja na izlaznu podatkovnu sabirnicu. U *write-first* pristupu prilikom upisa novog podatka u memorijsku lokaciju na izlaznu podatkovnu sabirnicu postavlja se nova vrijednost memorijske lokacije, dok se u *read-first* pristupu umjesto nove vrijednosti postavlja stara vrijednost memorijske lokacije. *No-change* pristup prilikom upisa novog podatka u memorijsku lokaciju ne mijenja stanje izlazne podatkovne sabirnice. U okviru realizirane dvopristupne memorije ostvaren je *read-first* način rada [17]. Osim zajedničkog takta svaki od portova A i B imaju po pet ulaza i jedan izlaz. Na slici 5.2 dan je primjer dvopristupne memorije čija je širina adresnih sabirnica 14 bitova, a širina podatkovnih sabirnica 12 bitova. Ulazi porta A su *addra, dina, wea, ena, regcea*. *Addra* predstavlja adresnu sabirnicu koja služi za adresiranje memorijske lokacije čiji se podatak želi pročitati ili na koju se lokaciju želi upisati novi podatak. Širina sabirnice se računa kao logaritam po bazi 2 kojem je ulazni argument željena dubina RAM-a. *Dina* predstavlja podatkovnu



Slika 5.2: Dvopristupna memorija sa zajedničkim taktom

sabirnicu preko koje se upisuje podatak u adresiranu memorijsku lokaciju dok *douta* predstavlja podatkovnu sabirnicu s koje se čita sadržaj adresirane memorijske lokacije. *Ena* omogućuje rad komponente, *wea* omogućuje pisanje na port A dok se *regcea* u ovom slučaju ne koristi.

Priloženi isječak koda predstavlja proces u kojem se ovisno o ulazu *wea* piše u memoriju odnosno čita iz memorije. Ako je *wea* postavljena u '1' podatak s podatkovne sabirnice *dina* upisuje se u memoriju. U suprotnom podatak se čita sa željene memorijske lokacije i sprema u registar *ram_data_a*. Isječak koda odnosi se na port A, dok za port B postoji isti isječak s odgovarajućim izmjenama koji se odvija paralelno s danim procesom.

```
process(clk)
  begin
    if rising_edge(clk) then
      if(ena = '1') then
        if(wea = '1') then
          true_dp_ram(to_integer(unsigned(addra))) := dina;
        else
          ram_data_a <= true_dp_ram(to_integer(unsigned(addra)));
        end if;
      end if;
    end if;
  end process;
```

U ovom diplomskom radu napisana je komponenta pod nazivom *true_dpram_sclk* s generičkim entitetom čime je ostvarena potrebna dvopristupna memorija. Generički entiteti služe pisanju familija istovrsnih sklopova s različitim svojstvima prilagođavanjem

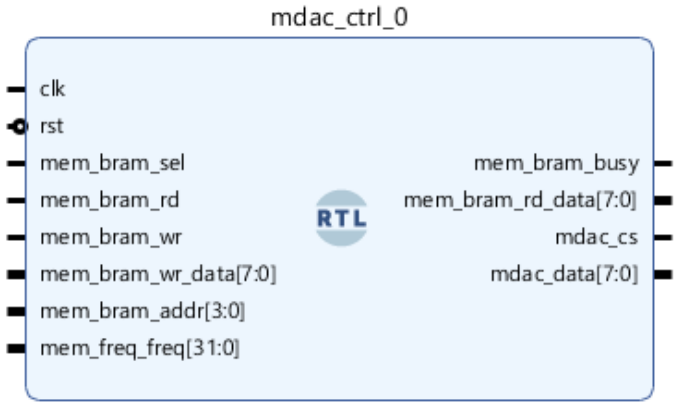
samo generičkih konstanti. Prilikom izrade diplomskog rada za svaki kanal trebalo je instancirati dvije komponente dvopristupne memorije. Jedna od njih predstavlja memoriju od šesnaest 8-bitnih podataka od kojih se prvih deset 8-bitnih podataka izravno prosljeđuju na izlaze *mdac_ctrl* logičkog bloka. Druga komponenta dvopristupne memorije zadužena je za pohranu podataka s ADP-ova. Svaka memorije velika je 16384 memorijskih lokacija u koju se spremaju 12 bitni podaci s ADP-a. Ova memorija je instancirana unutar *transfer_ctrl* VHDL blok koji se nalazi unutar *adc_mem_top* bloka.

5.2. *Mdac_ctrl* VHDL blok

Zadatak *mdac_ctrl* logičkog bloka je cirkularno čitanje 8-bitnih podataka iz implementirane memorije koja je ostvarena na PL strani i generiranje signala *mdac_cs* kojim se ostvaruje funkcionalnost DAP-a, odnosno množenje. Ovaj VHDL blok implementiran je za jedan kanal akvizicijskog sklopa i primjenjiv je na ostalih pet kanala. Unutar *mdac_ctrl* bloka instancirana je komponenta dvopristupne memorije pod nazivom *true_dpram_sclk*. Podaci o širini i dubini memorije zapisani su unutar paketa *utils_pkg_mdac* koji sadrži konstante, pomoćne varijable i potrebne funkcije. Naveden je primjer zadavanja širine i dubine memorije unutar paketa.

```
constant C_RAM_WIDTH : integer := 8;
constant C_RAM_DEPTH : integer := 16;
```

Ulazi i izlazi bloka prikazani su na slici 5.3.



Slika 5.3: Entitet komponente *mdac_ctrl*

Memorija se puni iz PS-a preko AXI sučelja stoga je potrebno realizirati ulaze i izlaze komponente *mdac_ctrl* koji će se prilikom povezivanja logičkih sklopova povezati izravno na podređeni AXI sklop koji je objašnjen u potpoglavlju 5.3. Sljedeći isječak

koda prikazuje instanciranje dvopristupne memorije i navodi na što se spajaju ulazi i izlazi memorije. S linijom koda *wea=> mem_bram_wr* omogućeno je pisanje na port A, dok je na portu B omogućeno samo čitanje. Budući da se na port A upisuju podaci iz procesorskog sustava ZedBoard, adresna *addra* i podatkovna *dina* sabirnica izravno se spajaju na AXI podređeni sklop.

```
  generic map (
    C_RAM_WIDTH       => work.utils_pkg_mdac.C_RAM_WIDTH,
    C_RAM_DEPTH       => work.utils_pkg_mdac.C_RAM_DEPTH,
    C_RAM_PERFORMANCE => work.utils_pkg_mdac.C_RAM_PERFORMANCE,
    C_INIT_FILE       => work.utils_pkg_mdac.C_INIT_FILE
  ) port map (
    clk    => clk,
    addra  => mem_bram_addr,
    addrb  => cnt_addr_gen,
    dina   => mem_bram_wr_data,
    dinb   => (others => '0'),
    wea    => mem_bram_wr,
    web    => '0',
    ena    => '1',
    enb    => '1',
    regcea => '1',
    regceb => '1',
    douta  => mem_bram_rd_data,
    doutb  => mdac_data_ss);
```

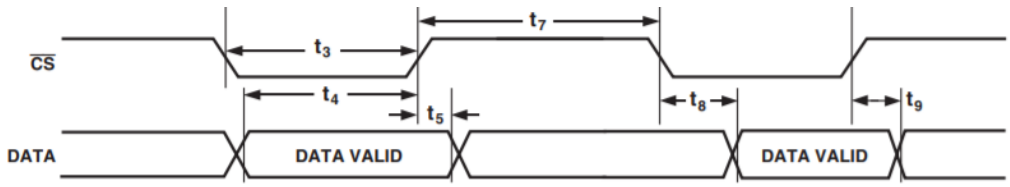
Preko sabirnice *mem_freq_freq* pristupa se AXI registru koji sadrži informaciju o frekvenciji rada digitalno-analognog pretvornika odnosno o brzini promjene bitova na njegovim ulazima. Podatak se postavlja prilikom inicijalizacije sustava u aplikaciji koja se izvodi na PS-u, dok ga logički sklop pohranjuje u registar *cnt_period_compare*. Brojač se povećava na svaki rastući brid takta sustava *clk* sve dok ne postane jednak registru *cnt_period_compare* prilikom čega se zastavica *cnt_period_ovf* postavlja u '1', a brojač resetira.

```
process(clk)
 begin
 if rising_edge(clk) then
    if rst = '1' then
      cnt_period <= (others => '0');
      cnt_period_compare <= (others => '0');
      cnt_period_ovf <= '0';
    else
     cnt_period_compare <= mem_freq_freq;
```

```
        if cnt_period = cnt_period_compare then
          cnt_period <= (others => '0');
          cnt_period_ovf <= '1';
        else
          cnt_period <= std_logic_vector(unsigned(cnt_period) + 1);
          cnt_period_ovf <= '0';
        end if;
    end if;
  end if;
end process;
```

Istovremeno s opisanim procesom odvija se proces generiranja adrese koja je izravno spojena na adresnu sabirnicu porta B. S obzirom na to da se radi o memoriji koja sadrži deset podataka koji se ciklički prosljeđuju na pinove DAP-a generiranje adrese odnosi se na povećavanje brojača *cnt_addr_gen* uz ispunjen uvjet da je zastavica *cnt_period_ovf*='1'. Kad je pročitan zadnji podatak iz memorije, proces kreće ispočetka.

```
process(clk)
begin
  if rising_edge(clk) then
       if rst = '1' then
          cnt_addr_gen <= (others => '0');
       else
          if cnt_period_ovf = '1' then
             if (cnt_addr_gen = "1001") then
                 cnt_addr_gen <= (others => '0');
             else
                  cnt_addr_gen <= std_logic_vector(unsigned(
                      cnt_addr_gen) + 1);
             end if;
          end if;
        end if;
      end if;
end process;
```

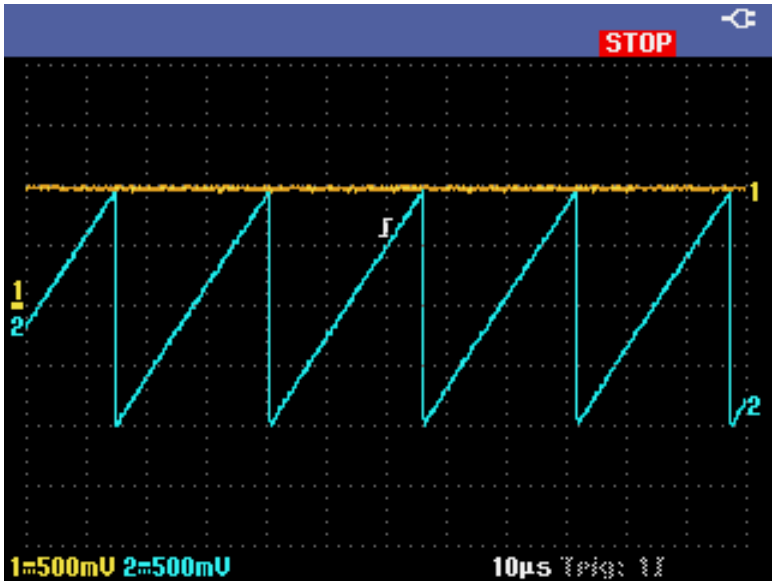


Slika 5.4: Vremenski dijagram DAP-a [3]

U radu je bilo potrebno implementirati upravljački signal *mdac_cs* perioda 100 ns koji je prvih 50 ns u stanju '1', a zatim se spušta u '0'. U dokumentu tehničke specifikacije DAP-a zadana su minimalna vremena trajanja pojedinih stanja. Minimalno trajanje konstante *t3* sa slike 5.4 iznosi 10 ns, dok minimalno vrijeme trajanja konstante *t7* iznosi 9 ns. Generiranje signala vrši se uz pomoć brojača *count_cs* koji se povećava na svaki rastući brid takta sustava. *Count_cs* se povećava dok ne dođe do 10, zatim se resetira i proces kreće ispočetka.

```
process (clk) is
begin
 if rising_edge(clk) then
     if rst = '1' then
        count_cs <= 0;
     else
         if count_cs = 10 then
            count_cs <= 1;
         else
            count_cs <= count_cs + 1;
                  if count_cs > 0 and count_cs < 6 then
                        mdac_cs_s <= '1';
                  else
                        mdac_cs_s <= '0';
                  end if;
         end if;
    end if;
 end if;
end process;
```

Da bi se ispitala ispravnost dizajnirane komponente *mdac_ctrl* izvršeno je testiranje na stvarnom sklopovlju. Na slici 5.5 ulazni signal označen žutom bojom množi se sa svih 256 razina počevši od najveće negativne vrijednosti -1 do najveće pozitivne +1. Široki raspon razina ostvaren je realizacijom *for* petlje unutar privremeno izmijenjenog *mdac_ctrl* bloka. Signali su snimljeni digitalnim osciloskopom Tektronix THS3024. Izlazni signal prikazan plavom bojom je pilastog oblika pa se može zaključiti i potvrditi linearnost digitalno-analognog pretvornika.



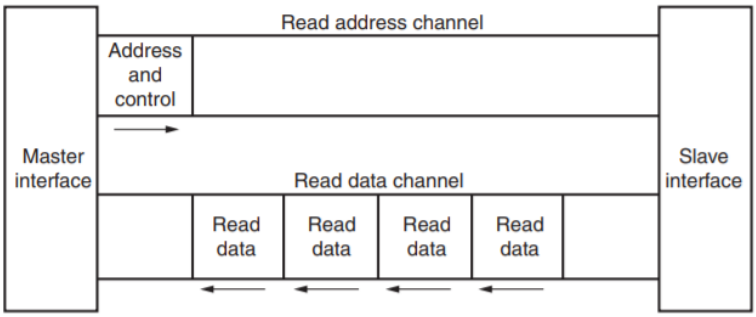
Slika 5.5: Ulazni signal (žuto) i izlazni signal (plavo) DAP-a

5.3. *Mdac_mem_axi_slv* VHDL blok

Pomoću AXI protokola omogućena je komunikacija između programabilne logike PL koja je u ovom slučaju podređeni sklop i procesorskog sustava na ZedBoard-u koji predstavlja nadređeni sklop. Unutar jednog kanala realizirana su dva podređena sklopa: *mdac_mem_axi_slv* koji je zadužen za slanje mjerne matrice prema digitalno-analognom pretvorniku i *adc_mem_axi_slv* preko kojeg se odvija prijenos podataka iz realizirane dvopristupne memorije u procesorski sustav. U ovom poglavlju naglasak je stavljen na prijenos mjerne matrice iz procesorskog sustava ZedBoard na akvizicijsku pločicu odnosno na razvoj *mdac_mem_axi_slv* VHDL blok. Kako je razvoj AXI sučelja zahtjevan proces, a nije u samom fokusu ovog diplomskog rada za razvoj AXI podređenog sklopa korišten je kostur gotovog AXI sklopa koji je prilagođen potrebama ovog rada.

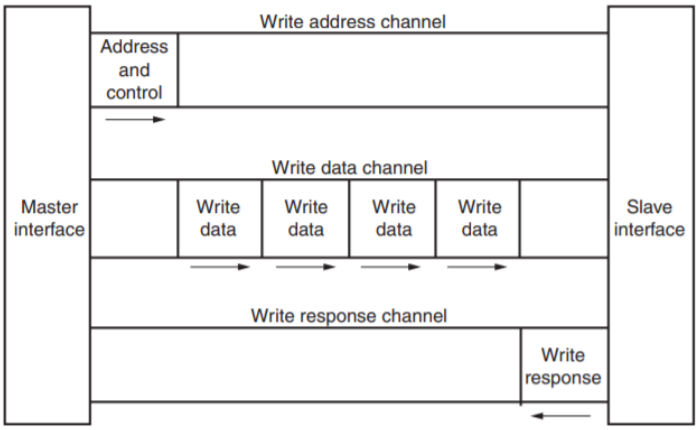
AXI protokol pripada ARM-AMBA obitelji mikroprocesorskih sabirnica i osigurava odvojene podatkovne i adresne veze za čitanje i pisanje što omogućuje istodobni, dvosmjerni prijenos podataka. Tri su tipa AXI4 sučelja u uporabi: AXI-stream sučelje te AXI4 i AXI4-Lite koji predstavljaju AXI memorijski mapirana sučelja. U ovom radu korišten je AXI4-Lite. Transakcije između blokova odvijaju se preko pet AXI kanala od kojih se dva odnose na čitanje, a tri na pisanje. Kao što je prikazano na slici 5.6 signale za čitanje čine dva kanala: adresni kanal za čitanje (engl. *Read Address Channel*) i podatkovni kanal za čitanje (engl. *Read Data Channel*). Nadređeni sklop (engl. *Master*) adresni kanal za čitanje koristi za slanje adrese i upravljačke informacije

podređenom sklopu (engl. *Slave*). Podatkovnim kanalom za čitanje prenose se podaci iz podređenog u nadređeni sklop s adrese koja je postavljena u adresni kanal. Kod



Slika 5.6: Kanali za čitanje [13]

pisanja je situacija slična samo što postoji dodatan kanal za potvrdu upisa. Slično kao kod adresnog kanala za čitanje nadređeni sklop adresni kanal za pisanje (engl. *Write Address Channel*) koristi za slanje upravljačke informacije i adrese u podređeni sklop. Podatkovnim kanalom za pisanje (engl. *Write Address Channel*) prenose se podaci iz nadređenog u podređeni sklop na adresu koja je postavljena u adresni kanal. Kanal za potvrdu upisa (engl. *Write Response Channel*) omogućuje podređenom sklopu da potvrdi nadređenom sklopu uspješnost primitka informacije ili da javi ako je došlo do neke pogreške prilikom pisanja.



Slika 5.7: Kanali za pisanje [13]

Upravljanje zahtjevima za pisanje i čitanje vrši se signalima za uspostavu prijenosa odnosno transakcije između AXI4 uređaja. Ti se signali nazivaju *handshake* signali i potrebni su za ispravan rad svih pet AXI4 kanala. Primalac koji može biti i nadređeni i podređeni sklop koristi signal (engl. *ready*) da bi označio da je spreman za primitak

adrese ili podatka. Pošiljatelj koji također može biti nadređeni i podređeni sklop koristi signal (engl. *valid*) da bi obavijestio primaoca da je podatak ili adresa na podatkovnom ili adresnom kanalu valjana i da primalac može obaviti prihvat adrese ili podatka. Kako bi se ostvario prijenos adrese ili podatka oba signala *ready* i *valid* moraju istovremeno biti postavljeni. Za ispravan rad potrebno je poštovati pravila i odnos između ta dva signala [14], [13].

AXI prospojni blok (engl. *AXI Interconnect*) je osnovna komponenta u AXI sustavu koja omogućuje spajanje više nadređenih sklopova i podređenih sklopova. Umjesto definiranja funkcioniranja načina rada sustava s više podređenih i nadređenih sklopova, AXI standard čvrsto definira samo prospojni blok za spajanje podređenih i nadređenih sklopova. Upravo se u tome očituje fleksibilnost AXI protokola. U slučaju kada u dizajnu postoji jedan nadređeni sklop i više podređenih sklopova prospojni blok treba ispravno protumačiti adresu podređenog uređaja kako bi ispravno usmjerio zahtijevane radnje. Tad se može koristiti adresni dekoder i upravo tako pristupalo se problemu. Na osnovu adrese na adresnoj sabirnici, AXI4 prospoj direktno šalje informacije prema ciljanom podređenom sklopu. Prilikom projektiranja digitalnog dizajna u razvojnom okruženju *Vivado Design Suite* svakom podređenom sklopa mora se odrediti ili automatski dodijeliti vrijednost bazne adrese na koju će sklop biti fizički mapiran unutar adresnog prostora nadređenog sklopa. Kasnije se prilikom pisanja aplikacije za PS ciljanom podređenom sklopu pristupa upravo preko dodijeljene adrese. Adrese su poravnate na 4 bajta budući da se koristi 32-bitna adresna sabirnica. Može se uočiti da sklop ne koristi zadnja dva bita adrese da bi znao u koje registre ili memoriju pisati ili iz kojih registara ili memorije čitati.

Za omogućavanje komunikacije između PL-a i PS-a AXI sučelje koristi signale *clk_axi*, *rst_axi* kao i 19 signala koji su potrebni za implementaciju AXI4-Lite sučelja. *Vivado* razvojno okruženje spomenutih 19 signala sabire u jedan port jer prepoznaje da implementirani logički blok komunicira preko AXI sučelja. Osim signala koji vode prema procesoru za ispravan rad potrebno je dodati niz signala koji služe kao sučelje prema memoriji i registar koji služi za određivanje frekvencije rada već opisane *mdac_ctrl* komponente. Sljedeći isječak koda ilustrira način rada adresnog dekodera za pisanje u standardne registre koji su vidljivi u memorijskom prostoru PL-a i PS-a. Transakcija pisanja odnosi se na prijenos podataka iz nadređenog sklopa u podređeni sklop, odnosno iz PS-a u PL. U registru *awaddr_reg_s* pohranjena je adresa na koju se upisuje podatak. Na adresi koja ima *offset* 0x00000050 u odnosu na baznu adresu na koju je sklop mapiran, u Linux aplikaciji koja se izvršava na procesorskom sustavu zadaje se podatak koji sadrži informaciju o frekvenciji rada DAP-a. Na toj adresi ma-

piran je registar *mem_freq_freq*. Adresni dekoder zanemaruje dva najmanje značajna bita adrese 0x00000050 stoga se registar *mem_freq_freq* dekodira preko memorijske lokacije 0x20 (binarno "10100").

U registar *wstrb_reg_s* pohranjuje se 4-bitni podatak iz AXI *wstrb* signala koji je izravno spojen na ulaz sklopa. Svaki bit registra *wstrb_reg_s* koji je postavljen u '1' označava koji bajt podatka je potrebno uzeti u obzir. Budući da je vrijednost signala *wstrb_reg_s* = "1111" ta kombinacija bitova označava da sva 4 bajta podatka predstavljaju korisnu informaciju.

```
case awaddr_reg_s(6 downto 2) is
   when "10100" =>
       wr_reg_type_v := std_wr;
       if wstrb_reg_s(3) = '1' then
       mem_freq_freq_int(31 downto 24) <= wdata_reg_s(31 downto 24);
       end if;
       if wstrb_reg_s(2) = '1' then
       mem_freq_freq_int(23 downto 16) <= wdata_reg_s(23 downto 16);
       end if;
       if wstrb_reg_s(1) = '1' then
       mem_freq_freq_int(15 downto 8) <= wdata_reg_s(15 downto 8);
       end if;
       if wstrb_reg_s(0) = '1' then
       mem_freq_freq_int(7 downto 0) <= wdata_reg_s(7 downto 0);
       end if;
     when others => null;
end case;
```

Sljedeći isječak koda prikazuje način rada adresnog dekodera za pisanje u memoriju koju izravno koristi komponenta *mdac_ctrl*. U memoriju se zapisuje deset 8-bitnih podataka koji čine jedan redak mjerne matrice. Registar *awwadr_reg_s* sadrži adresu AXI signala na koju se podatak upisuje. Na početku koda provjerava se podudaranje vrijednosti adresnog registra s jednom od šesnaest adresa na koje je mapirana memorija. Unutar *mdac_ctrl* VHDL bloka određeno je da se vrijednosti s prvih deset memorijskih lokacija prosljeđuju na pinove DAP-ova. Iako je veličina potrebne memorije 10×8, instancirana je komponenta dvopristupne memorije veličine 16×8 jer se prilikom instanciranja memorije unutar *mdac_ctrl* bloka koristi funkcija *clogb2*. Funkcija se nalazi u paketu *utils_pkg_mdac* i kao ulazni argument prima dubinu memorije koja mora biti potencija broja 2. Ovisno o dubini memorije funkcija računa širinu adresne sabirnice kao logaritam dubine memorije. Podatak iz registra *wdata_reg_s* prosljeđuje se na izlazni port *mem_bram_wr_data* koji je izravno spojen na podatkovnu sabirnicu

mdac_ctrl VHDL bloka.

```
if (unsigned(awaddr_reg_s(6 downto 2)) >= "00000") and  (unsigned(
    awaddr_reg_s(6 downto 2)) <= "01111") then
       wr_reg_type_v  := mem_wr
       mem_busy_v     := mem_bram_busy;
       mem_bram_addr <= awaddr_reg_s(5 downto 2);
       if (mem_cycle_s >= 0) and (mem_cycle_s <= 3) then
          mem_bram_sel <= '1';
       end if;
       if mem_cycle_s = 3 and retry_mem_s = '0' then
           mem_bram_wr       <= '1';
           mem_bram_wr_data  <= wdata_reg_s(7 downto 0);
       end if;
end if;
```

U danom isječku koda prikazano je čitanje podatka iz registra *mdac_freq_freq* kojim se upravlja radom *mdac_ctrl* komponente. Registar koji sadrži adresu s koje se čita podatak naziva se *araddr_reg_s*, a pročitani podatak šalje se procesoru preko 32-bitne podatkovne sabirnice *rdata*. Ukoliko se želi pristupiti nemapiranoj memorijskoj lokaciji na sabirnicu *rdata* se sprema informacija o nedopuštenoj operaciji.

```
case araddr_reg_s(6 downto 2) is
   when "10100" =>
         rd_reg_type_v := std_rd;
         rdata(31 downto 0)  <= mem_freq_freq_int;
    when others =>
         rdata <= x"DEADDEAD";
end case;
```

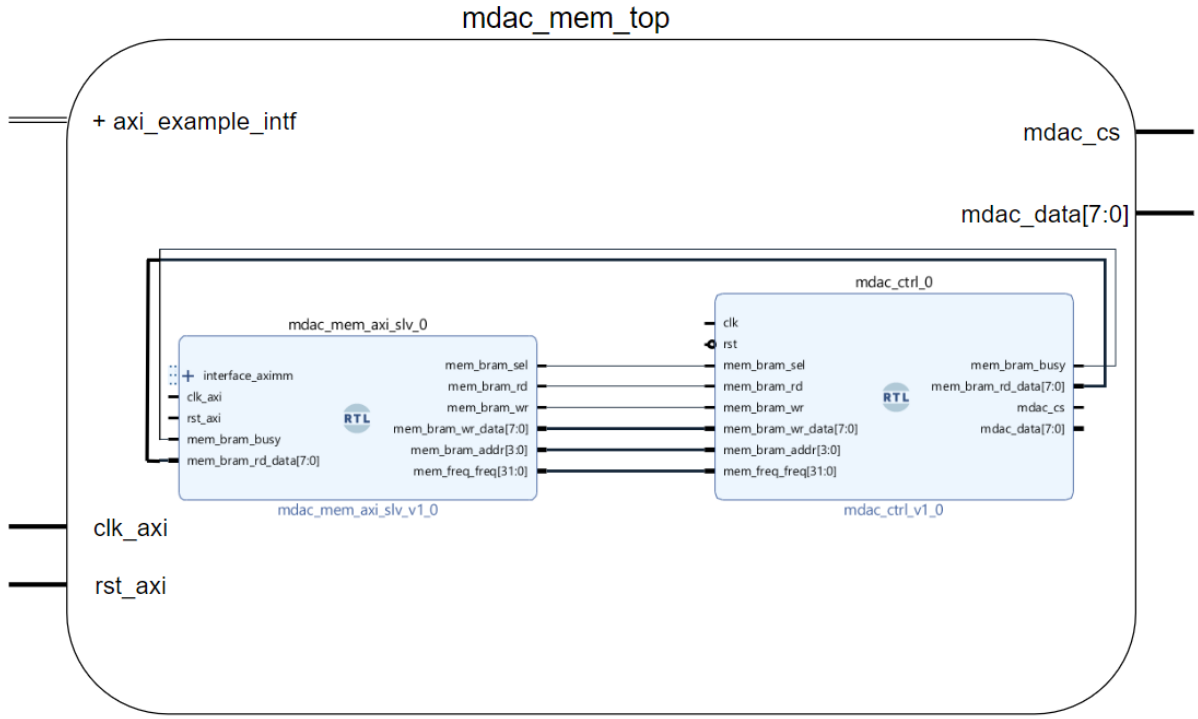
U sljedećem isječku koda prikazan je proces čitanja podataka iz realizirane dvopristupne memorije. Iz memorije se čita deset 8-bitnih podataka koji predstavljaju redak mjerne matrice. *Arradr_reg_s* predstavlja registar koji sadrži adresu s koje se čita željeni podatak. Vrijednost registra izravno se prosljeđuje na AXI sabirnicu *araddr* koja predstavlja ulaz komponente *mdac_mem_axi_slv*. Kao i kod pisanja u memoriju provjerava se je li adresa s koje se želi čitati unutar memorijskog raspona adresa. Pročitani podatak se preko 8-bitnog registra *mem_rd_data_s* prosljeđuje na podatkovnu sabirnicu za čitanje *rdata*.

```
if (unsigned(araddr_reg_s(6 downto 2)) >= "00000") and
     (unsigned(araddr_reg_s(6 downto 2)) <= "01111") then
    rd_reg_type_v  := mem_rd;
    mem_busy_v     := mem_bram_busy;
    mem_rd_data_s(7 downto 0) <= mem_bram_rd_data;
```

```
    mem_bram_addr <= araddr_reg_s(5 downto 2);
    if (mem_cycle_s >= 0) and (mem_cycle_s <= 3) then
        mem_bram_sel <= '1';
    end if;
    if mem_cycle_s = 0 then
        mem_bram_rd <= '1';
    end if;
end if;
```

5.4. *Mdac_mem_top* VHDL blok

Radi jednostavnosti i preglednosti dizajna, obrađeni blokovi koji se odnose na način rada digitalno-analognog pretvornika povezani su u jedan modul. U grafičkom sučelju *Vivado Design Suite* modul se prikazuje kao viši hijerarhijski blok u odnosu na opisane komponente. U ovom radu logički blokovi povezuju se u modul pod nazivom *mdac_mem_top*. *Mdac_mem_top* VHDL modul povezuje dva bloka: *mdac_mem_axi_slv* i *mdac_ctrl*. Unutar logičkog bloka *mdac_ctrl* koristi se komponenta *true_dpram_sclk* kojom je ostvarena dvopristupna memorija.



Slika 5.8: Povezivanje komponenti unutar *mdac_mem_top* modula

Ulazi i izlazi modula *mdac_mem_top* prikazani su na slici 5.8 kao i način povezivanja komponenti *mdac_mem_axi_slv* i *mdac_ctrl* međusobno. Kad bi samo promatrali spomenute ulaze i izlaze, ne bi mogli otkriti ništa više o načinu rada samog bloka pa bi se komponenta *mdac_mem_top* mogla promatrati kao svojevrsna crna kutija. Unutar modula *mdac_mem_top* ulazi i izlazi svih komponenti koje se nalaze i spajaju unutar modula moraju biti ispravno spojeni. Spajanje komponenti unutar istog modula vrši se pomoću internih signala. Na primjer izlazni port *mem_bram_wr* komponente *mdac_mem_axi_slv* preko kojeg se omogućava pisanje u memoriju potrebno je spojiti na ulaz komponente *mdac_ctrl* naziva *mem_bram_wr*. To je izvedeno pomoću dodatnog signala *mem_bram_wr* koji ne treba nužno imati isto ime kao ulaz ili izlaz na koji se spaja. Signal se pridružuje spomenutom ulazu i izlazu čime su ta dva priključka uspješno spojena. U danom isječku koda moguće je uočiti da je ulaz komponente *mdac_mem_axi_slv* koji omogućava pisanje u memoriju *mem_bram_rd* postavljen kao *'open'* čime se označava da priključak nije spojen. Kao što je objašnjeno u poglavlju 5.1 iz dvopristupne memorije se čita onda kad se ne izvršava operacija pisanja. Memorija je realizirana tako da je čitanje omogućeno kada je *wea* = '0' i nije potrebno koristiti nikakve dodatne zastavice ili registre.

```
mdac_mem_axi_slv_inst : mdac_mem_axi_slv
 port map (
  clk_axi  => clk_axi,
  rst_axi  => rst_axi,
  awaddr   => awaddr,
  awprot   => awprot,
  awvalid  => awvalid,
  awready  => awready,
  wdata    => wdata,
  wstrb    => wstrb,
  wvalid   => wvalid,
  wready   => wready,
  bready   => bready,
  bresp    => bresp,
  bvalid   => bvalid,
  araddr   => araddr,
  arprot   => arprot,
  arvalid  => arvalid,
  arready  => arready,
  rready   => rready,
  rdata    => rdata,
  rresp    => rresp,
```

```
  rvalid   => rvalid,
  mem_bram_busy      => mem_bram_busy,
  mem_bram_sel       => mem_bram_sel,
  mem_bram_rd        => open,
  mem_bram_wr        => mem_bram_wr,
  mem_bram_rd_data   => mem_bram_rd_data,
  mem_bram_wr_data   => mem_bram_wr_data,
  mem_bram_addr      => mem_bram_addr,
  mem_freq_freq      => mem_freq_freq);
mdac_ctrl_inst : mdac_ctrl
 port map (
  clk => clk_axi,
  rst => rst_axi,
  mem_bram_busy           => mem_bram_busy,
  mem_bram_sel            => mem_bram_sel,
  mem_bram_rd             => '0',
  mem_bram_wr             => mem_bram_wr,
  mem_bram_rd_data        => mem_bram_rd_data,
  mem_bram_wr_data        => mem_bram_wr_data,
  mem_bram_addr           => mem_bram_addr,
  mem_freq_freq           => mem_freq_freq,
  mdac_cs                 => mdac_cs,
  mdac_data               => mdac_data );
```

U akvizicijskom sustavu postoji šest kanala, stoga je bilo potrebno upogoniti šest *mdac_ mem_top* komponenti. Adresa na koju se mapira podređeni sklop unutar memorijskog prostora nadređenog sklopa može biti proizvoljna ili automatski dodijeljena. Pritom se treba poštovati dozvoljeni raspon adresa. U tablici 5.1 dan je popis podređenih sklopova, početnih adresa na koje se mapiraju podređeni sklopovi i raspon memorije koju zauzima svaki sklop. Ovisno o rasponu memorije razvojno okruženje *Vivado Design Suite* izračuna četvrti parametar tablice, a to je najviša adresa podređenog sklopa. U slučaju *mdac_mem_top* za svaki kanal je odabran raspon od 4 KB što je i više nego dovoljno s obzirom na to da memorija za svaki DAP sadrži šesnaest 8-bitnih podataka.

Tablica 5.1: Mapiranje podređenih sklopova

Slave	Offset Address	Range	High Address
mdac_mem_top_0	0x5800_0000	4K	0x5800_FFFF
mdac_mem_top_1	0x6000_0000	4K	0x6000_FFFF
mdac_mem_top_2	0x7800_0000	4K	0x7800_FFFF
mdac_mem_top_3	0x4C00_0000	4K	0x4C00_FFFF
mdac_mem_top_4	0x5400_0000	4K	0x5400_FFFF
mdac_mem_top_5	0x5C00_0000	4K	0x5C00_FFFF

Programiranje priključaka definirano je u posebno kreiranoj UCF datoteci (engl. *User Constraints File*) u kojoj se svakom ulazu i izlazu sustava dodjeljuje fizički priključak na razvojnom sustavu ZedBoard. Promatrat ćemo primjer upravljačkog signala *mdac_cs* kojem je dodijeljen odgovarajući slobodni pin imena J17. Ovisno o pinu koji je dodijeljen potrebno je odrediti standard napajanja za taj pin. Kako taj pin pripada banci priključaka broj 34 potrebno je koristiti standard napajanja 2.5 V kojeg zahtjeva ta banka priključaka. Korištenje UCF datoteka omogućava jednostavno dodjeljivanje fizičkih priključaka svim pinovima sustava.

```
set_property PACKAGE_PIN J17     [get_ports {mdac_cs_0}]
set_property IOSTANDARD LVCMOS25 [get_ports {mdac_cs_0}]
```

5.5. *Adc_mem_top* VHDL blok

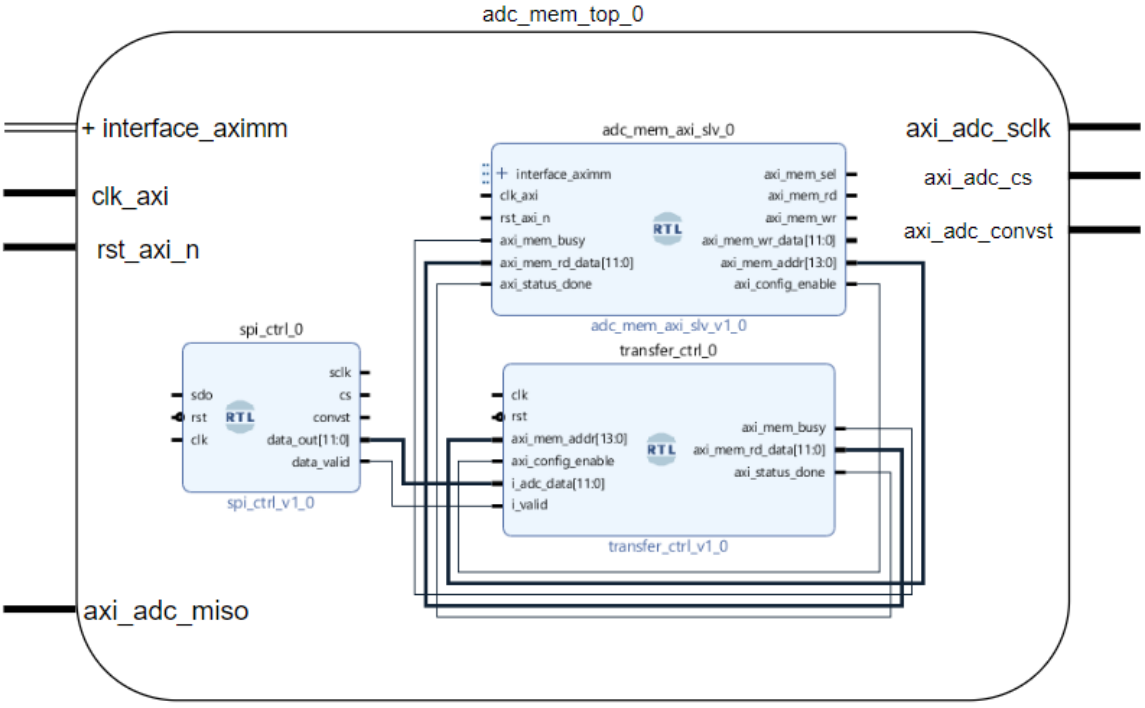
Da bi se postigla funkcionalnost cijelog akvizicijskog sustava za svaki kanal su razvijena dva FPGA podređena sklopa koja komuniciraju s procesorom. U prethodnim potpoglavljima pratio se razvoj *mdac_mem_top* logičkog sklopa koji poslužuje DAP s retkom mjerne matrice kojim se ulazni signal množi. Osim razvijenog *mdac_mem_top* modula bilo je potrebno razviti sklop kojim se ostvaruje akvizicija i pohrana mjerenih podataka u procesorski sustav na ZedBoard-u. Detaljniji opis ovog dijela akvizicijskog sustava dan je u radu *Akvizicijski sustav iznad Nyquistove granice*[1]. Za praćenje rezultata i funkcioniranje sustava u cjelini u ovom potpoglavlju ukratko će biti opisan rad *adc_mem_top* logičke komponente. U tablici 5.2 dan je popis šest *adc_mem_top* podređenih sklopova, po jedan za svaki kanal, pomoću kojih se odvija prijenos podataka iz memorije u procesorski sustav na ZedBoard. Osim imena podređenih sklopova

[1] I.Marinović, "Akvizicijski sustav iznad Nyquistove granice", Diplomski rad, Zagreb, 2020.

Tablica 5.2: Mapiranje podređenih sklopova

Slave	Offset Address	Range	High Address
adc_mem_top_0	0x4000_0000	128K	0x4001_FFFF
adc_mem_top_1	0x5000_0000	128K	0x5001_FFFF
adc_mem_top_2	0x7000_0000	128K	0x7001_FFFF
adc_mem_top_3	0x4800_0000	128K	0x4801_FFFF
adc_mem_top_4	0x6800_0000	128K	0x6801_FFFF
adc_mem_top_5	0x4400_0000	128K	0x4401_FFFF

u tablici su navedene početne adrese na koje su podređeni sklopovi mapirani, raspon memorije koji zauzimaju kao i najviša memorijska lokacija podređenog sklopa. Za razliku od 4 KB memorije koju zauzima svaki *mdac_mem_top* modul prilikom realizacije *adc_mem _top* podređenog sklopa svakom je bilo potrebno osigurati 128 KB memorije. Za pojedini kanal implementiran je po jedan *adc_mem_top* podređeni sklop. *Adc_mem_top* modul je hijerarhijski viši blok koji povezuje tri bloka: *spi_ctrl*, *transfer_ctrl* i *adc_mem_axi_slv*.



Slika 5.9: Povezivanje komponenti unutar *adc_mem_top* modula

Dvije glavne zadaće ovog sklopa su generiranje upravljačkih signala za ispravan rad analogno-digitalnog pretvornika i pohranjivanje 12-bitnih podataka s izlaza ADP-a u realiziranu dvopristupnu memoriju koja se nalazi na PL strani. Slika 5.9 prikazuje na koji način i preko kojih sabirnica se povezuju blokovi unutar jednog *adc_mem_top* VHDL modula. Inače, unutar *Vivado Design Suite* projekta mogu se vidjeti samo ulazi i izlazi *adc_mem_top* bloka dok su ulazi i izlazi ostalih komponenti skriveni unutar samog bloka. Unutar *utils_pkg_adc* paketa definirana je veličina i širina dvopristupne memorije. Funkcija *clogb2* iz istog paketa na temelju ulaznog argumenta dubine memorije izračunava širinu adresne sabirnice koja logaritamski ovisi o toj dubini. Za pohranu 12-bitnih podataka koje *spi_ctrl* komponenta daje na svom izlazu implementirana je dvopristupna memorija s 16384 memorijskih lokacija za svaki kanal.

```
constant C_RAM_WIDTH : integer := 12;
constant C_RAM_DEPTH : integer := 16384;
```

Sklopovska podrška za analogno-digitalni pretvornik započinje logičkim blokom *spi_ctrl*. Radi se o sklopu za prihvat podataka s ADP-a koji prima niz bitova preko SPI sučelja i na svom izlazu akvizira 12-bitne podatke. Kako svako SPI sučelje mora imati barem jedan *master* i jedan *slave* sklop, u ovoj komunikaciji *spi_ctrl* predstavljen je u ulozi *master* sklopa, a ADP je zapravo *slave* kojim se upravlja. Na izlazu komponente *spi_ctrl* generiraju se upravljački signali *sclk*, *convst*, *cs* koji se preko FMC konektora izravno dovode do fizičkih pinova ADP-a na akvizicijskoj pločici. Pomoću njih se upravlja radom ADP-a i određuje frekvencija uzimanja uzoraka koja iznosi 1 MHz. Tako je zadovoljena početna ideja da se ulazni signal množi svakih 100 ns i da se nakon deset perioda odnosno 1 μs uzimaju uzorci s integratora koji se prosljeđuju procesoru na daljnju rekonstrukciju.

Nakon što je razvijen sklop za prihvat podataka s ADP-a bilo je potrebno ostvariti sklop koji kontrolira upis podataka u realiziranu dvopristupnu memorija i signalizira procesoru da može započeti s prijenosom podataka. Radi se o *transfer_ctrl* logičkom bloku koji unutar sebe instancira već opisanu *true_dpram_sclk* dvopristupnu memoriju s drugačijim parametrima nego u slučaju *mdac_ctrl* logičkog bloka. Kao što je već istaknuto realizirana memorija može pohraniti 16384 12-bitnih podataka, no ako je potrebno veličinu memorije vrlo je jednostavno izmijeniti zbog korištenja generičkih konstanti u dizajnu. Logička komponenta *transfer_ctrl* povezuje ostale komponente *adc_mem_top* modula. Glavni zadatak ove komponente je pohrana 12-bitnih podataka proslijeđenih iz sklopa *spi_ctrl* koji se spremaju u realiziranu dvopristupnu memoriju. S očitavanjem i upisom u memoriju započinje se kad procesorski sustav na ZedBoard-u postavi signal *adc_config_enable* u visoko stanje čime dojavljuje da je sve spremno

za akviziciju. Komponenta *transfer_ctrl* ima ulogu dojaviti sklopu *adc_mem_axi_slv* kad je memorija puna kako bi sklop *adc_mem_axi_slv* mogao preko AXI sučelja obavijestiti procesor da može krenuti s prijenosom podataka. U ovom slučaju signalizacija se odvija preko signala *axi_status_done* koji se postavlja u '1' ako je memorija puna. Upisivanje ADP uzoraka u memoriju ostvareno je automatom s konačnim brojem stanja. Postoje četiri stanja u kojem se *tranfer_ctrl* može nalaziti a to su: *init*, *get_sample*, *write_mem* i *done*. U stanje *init* logički sklop *transfer_ctrl* ulazi kad procesorski sustav na ZedBoard-u postavi signal *adc_config_enable* u nisku razinu što znači da nije još spreman za akviziciju. Postavljanjem signala *adc_config_enable* u '1' ulazi se u stanje *get_sample*. Iz stanja *get_sample* logički sklop *tranfer_ctrl* može ići u dva stanja. U stanje *mem_write* prelazi ako želi pisati u memoriju, a u stanje *done* prelazi ako je gotov s pisanjem u memoriju. Kada je sklop u stanju *mem_write* potrebno je stalno provjeravati zastavicu koja dolazi s logičkog sklopa *spi_ctrl* i koja označava je li podatak koji se želi upisati valjan. Kada logički sklop uđe u stanje *done* potrebno je postaviti signal *axi_status_done* u visoku razinu čime zapravo procesorski sustav na ZedBoard-u zna da je memorija puna i da može krenuti s prijenosom podataka iz napunjene dvopristupne memorije u PS putem AXI sučelja.

S obzirom na to da se sva komunikacija između PL i PS dijela odvija preko AXI4-Lite sučelja trebalo je razviti novi AXI podređeni sklop kojim će se omogućiti prijenos AD uzoraka iz realizirane dvopristupne memorije u ARM procesor na ZedBoard-u. *Adc_mem_axi_slv* podređen sklop usmjeren je na komunikaciju prema *tranfer_ctrl* sklopu. Baš kao i kod *mdac_mem_axi_slv* podređenog sklopa na sličan način ostvareno je adresno dekodiranje standardnih registara i memorije u transakcijama pisanja i čitanja opisano u potpoglavlju 5.3. Na slici 5.9 je između ostalog prikazana komponenta *adc_mem_axi_slv* sa svojim ulazima i izlazima. Osim AXI sabirnice *interface_aximm* komponenta *adc_mem_axi_slv* ima portove koji služe kao sučelje prema memoriji u *transfer_ctrl* logičkom bloku iz kojeg se prenose 12-bitni podaci i portove za signale *axi_status_done* i *axi_config_enable* preko kojih se ostvaruje kontrolna logika.

Za svaki kanal su razvijena po dva FPGA podređena sklopa koja komuniciraju s procesorom. Prvi modul je *mdac_mem_top* koji poslužuje DAP s retkom mjerne matrice kojim se ulazni signal množi, a drugi modul odnosi na *mdac_mem_top* kojim se ostvaruje akvizicija i pohrana mjerenih podataka u procesorski sustav na ZedBoard-u. Ukupno je implementirano dvanaest podređenih sklopova koji za svoj rad koriste AXI protokol. Točnije šest sklopova *adc_mem_top* koji ostvaruju komunikaciju prema analogno-digitalnim pretvornicima i šest sklopova *mdac_mem_top* koji poslužuju DAP-ove.

6. Sinkronizacijski sklop

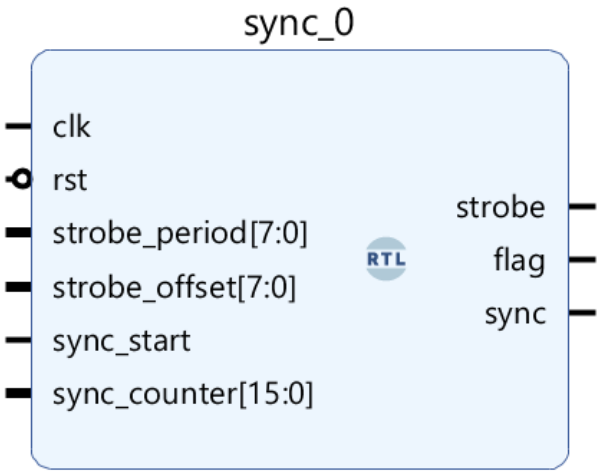
Signal koji se dovodi na ulaz akvizicijskog sklopa prvo se prosljeđuje na pretpojačalo nakon čega se množi periodičnim valnim oblikom perioda T. Frekvencija kojom digitalno-analogni pretvornik množi ulazni signal s 8-bitnom vrijednosti deset puta je veća od frekvencije uzorkovanja analogno-digitalnog pretvornika. Za vrijeme perioda uzorkovanja analogno-digitalnog pretvornika koji iznosi 1 μs signal se množi s deset 8-bitnih vrijednosti. Slijed od deset različitih vrijednosti koji se periodično ponavlja svake 1 μs predstavlja jedan redak mjerne matrice.

Pretvorba signala u digitalnu reprezentaciju započinje na padajući brid signala *convst* pa se signal *convst* kojeg generira logički blok *spi_ctrl* u definiranim trenutcima spušta u nisku razinu. Sama ideja akvizicijskog sustava postavlja zahtjev za sinkronizaciju ADP i DAP elektroničkih komponenti. Na početku izrade ovog diplomskog rada nije bilo moguće uzeti u obzir sinkronizaciju ADP-a i DAP-a budući da su komponente promatrane i uhodavane samostalno, a tek kasnije je omogućen istovremeni zajednički rad. U okviru ovog rada trebalo je razviti sklop za sinkronizaciju DAP-a i ADP-a koji svakih 10 ciklusa takta frekvencije 100 MHz obavještava DAP da može krenuti s množenjem ulaznog signala odnosno koji svakih 100 ciklusa takta frekvencije 100 MHz signalizira ADP-u da može krenuti s očitavanjem. Zbog tog je razvijen univerzalni sinkronizacijski sklop koji se instancira unutar *mdac_mem_top* i *adc_mem_top* modula.

6.1. Realizacija sinkronizacijskog sklopa

Sklopovlje koje upravlja radom ADP-a i sklopovlje koje upravlja radom DAP-a mora krenuti s radom u istom trenutku. U obzir je bilo potrebno uzeti moguće digitalno kašnjenje ADP-a. U tehničkoj dokumentaciji ADP-a AD7091R [2] trebalo je obratiti pažnju na aperturno kašnjenje koje iznosi 5 ns. Aperturno kašnjenje označava vremenski odmak od trenutka kad je ADP trebao i kad je zapravo započeo konverziju. Sinkronizacijskim sklopom bilo je potrebno ostvariti mogućnost unosa kašnjenja u rad

ADP-a i DAP-a. Cijeli akvizicijski sustav radi na 100 MHz, stoga kašnjenje koje se unosi mora biti u kvantovima od najmanje 10 ns. Sinkronizacija je ostvarena pomoću signala *strobe* čiji puls signalizira ADP-u i DAP-u da krene s konverzijom odnosno s množenjem. Osim *strobe* signala trebalo je dodati i ostale signale koji sudjeluju u sinkronizacijskom sklopu. Na slici 6.1 dan je prikaz entiteta sinkronizacijskog sklopa.



Slika 6.1: Entitet sinkronizacijskog sklopa

Sinkronizacijski sklop kao jedan od ulaza ima *sync_start* signal na kojem sklop očekuje puls trajanja jednog ciklusa takta. Ako se na *sync_start* ulazu pojavi puls, sva se logika u sinkronizacijskom sklopu mora resetirati i zaustaviti. Također, izlaz *sync* mora otići u visoku razinu u trajanju *sync_counter* ciklusa takta. *Sync* signal izravno je doveden na logičke blokove *mdac_ctrl* i *spi_ctrl* i služi da bi se ADP i DAP komponente postavile u stanje pripravnosti za početak konverzije odnosno množenja. *Strobe* signal moguće zakasniti za *strobe_offset* ciklusa takta kojim se uklanjanju moguća kašnjenja. Prije generiranja *strobe* signala potrebno je pričekati vremenski period koji je zapravo suma vremenskih odsječaka *sync_counter* i *strobe_offset* ciklusa takta. Za brojanje određenog perioda definiran je 16-bitni brojač *counter*. Brojač se resetira kada se na ulazu *sync_start* pojavi puls, a inače se povećava.

Dani isječak koda zaslužan je za generiranje *sync* signala koji osigurava da sva logika kreće s radom u istom trenutku. Promjena razine *sync* signala mijenja stanje *flag* zastavice o kojoj ovisi generiranje *strobe* signala. Izlaz *sync* je postavljen u '1' sve dok je brojač *counter* manji od zadanog *sync_counter* ciklusa takta. Kada brojač *counter* postane jednak sumi trajanja vrijednosti *sync_counter* i *strobe_offset* zastavica *flag* se postavlja u '1'.

```
process(clk) is
```

```
 begin
     if rising_edge(clk) then
          if rst = '1' then
                sync_s <= '0';
                flag_s  <= '0';
        else
             if counter < unsigned(sync_counter) then
                   sync_s <= '1';
                   flag_s <= '0';
             elsif counter = sum then
                   sync_s <= '0';
                   flag_s <= '1';
             else
                   sync_s <= '0';
            end if;
         end if;
    end if;
end process;
```

Generiranje *strobe* signala ovisi o stanju zastavice *flag*. Sklop generira puls trajanja jednog ciklusa takta na izlaz *strobe* svakih *strobe_period* ciklusa takta. Za praćenje *strobe_period* uveden je novi 16-bitni brojač *strobe_count*. Ako je zastavica *flag* u visokoj razini provjerava se stanje brojača *strobe_count*. Kad brojač *strobe_count* postane jednak trajanju *strobe_period-1* ciklusa takta *strobe* signal se postavlja u '1', a brojač *strobe_count* resetira na 0. Ako nije ispunjen zadani uvjet brojač *strobe_count* se povećava, dok se *strobe* signal postavlja u '0'.

```
process(clk) is
  begin
     if rising_edge(clk) then
        if rst = '1' then
             strobe_count <= (others=>'0');
             strobe_s <= '0';
         else
            if flag_s = '1' then
              if(strobe_count = unsigned(strobe_period)-1) then

                   strobe_count <= (others=>'0');
                   strobe_s <= '1';
              else
                   strobe_count <= strobe_count +1;
                   strobe_s <='0';
              end if;
```

```
            else
               strobe_s <='0';
               strobe_count <= (others=>'0');
             end if;
         end if;
  end if;
end process;
```

6.2. Ispitna okolina

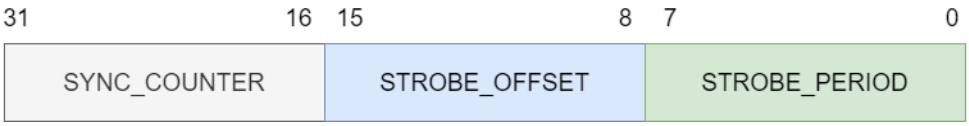
Nakon što je sinkronizacijski sklop dizajniran u VHDL jeziku bilo je potrebno provjeriti njegovu funkcionalnost. Ispitno okruženje (engl. *test bench*) predstavlja apstraktni model okoline te se sastoji od generiranja pobudnih signala, spajanja sustava koji se testira, provjere ispravnosti. Entitet ispitnog okruženja nema nikakvih ulaznih ni izlaznih priključaka jer predstavlja okolinu koja se ne povezuje na druge komponente. Simulacija je izvršena unutar razvojnog okruženja *Vivado Design Suite*. Na početku koda ispitne okoline nalazi se entitet komponente koja se promatra. U danom isječku koda prikazan je način generiranje odgovarajućih pobudnih signala potrebnih za ispitivanje ispravnosti opisa sinkronizacijskog sklopa. Generiran je takt perioda 10 ns. Sinkronizacijski sklop generira puls trajanja jednog ciklusa takta na izlaz *strobe* svakih *strobe_period* ciklusa takta odnosno svakih 30 ns jer je u *strobe_period* pohranjena vrijednost 3. *Strobe* signal je zakašnjen za *strobe_offset* ciklusa takta i to vrijeme u simulaciji iznosi 20 ns. *Sync_start* postavljen je u visoku razinu u razdoblju od 55 ns do 65 ns i 605 ns do 615 ns, dok je *rst* aktivan u vremenskom periodu od 310 ns do 320 ns.

```
clk <= not clk after 5 ns;
strobe_period <= "00000011";
strobe_offset <= "00000010";
sync_counter <= "0000000000000100";
sync_start <= '0', '1' after 55 ns, '0' after 65 ns, '1' after 605
   ns, '0' after 615 ns;
rst <= '0', '1' after 310 ns, '0' after 320 ns;
```

Na zaslonu 6.3 prikazana je simulacija u trajanju od 1 μs. Moguće je uočiti da nakon što se *sync* signal u 105. ns opet spusti u '0' do generiranja *strobe* signala prođe točno 50 ns odnosno vrijeme od 5 ciklusa takta koje se dobije zbrajanjem *sync_counter* i *sync_offset*.

6.3. Nadogradnja sustava sinkronizacijskim sklopom

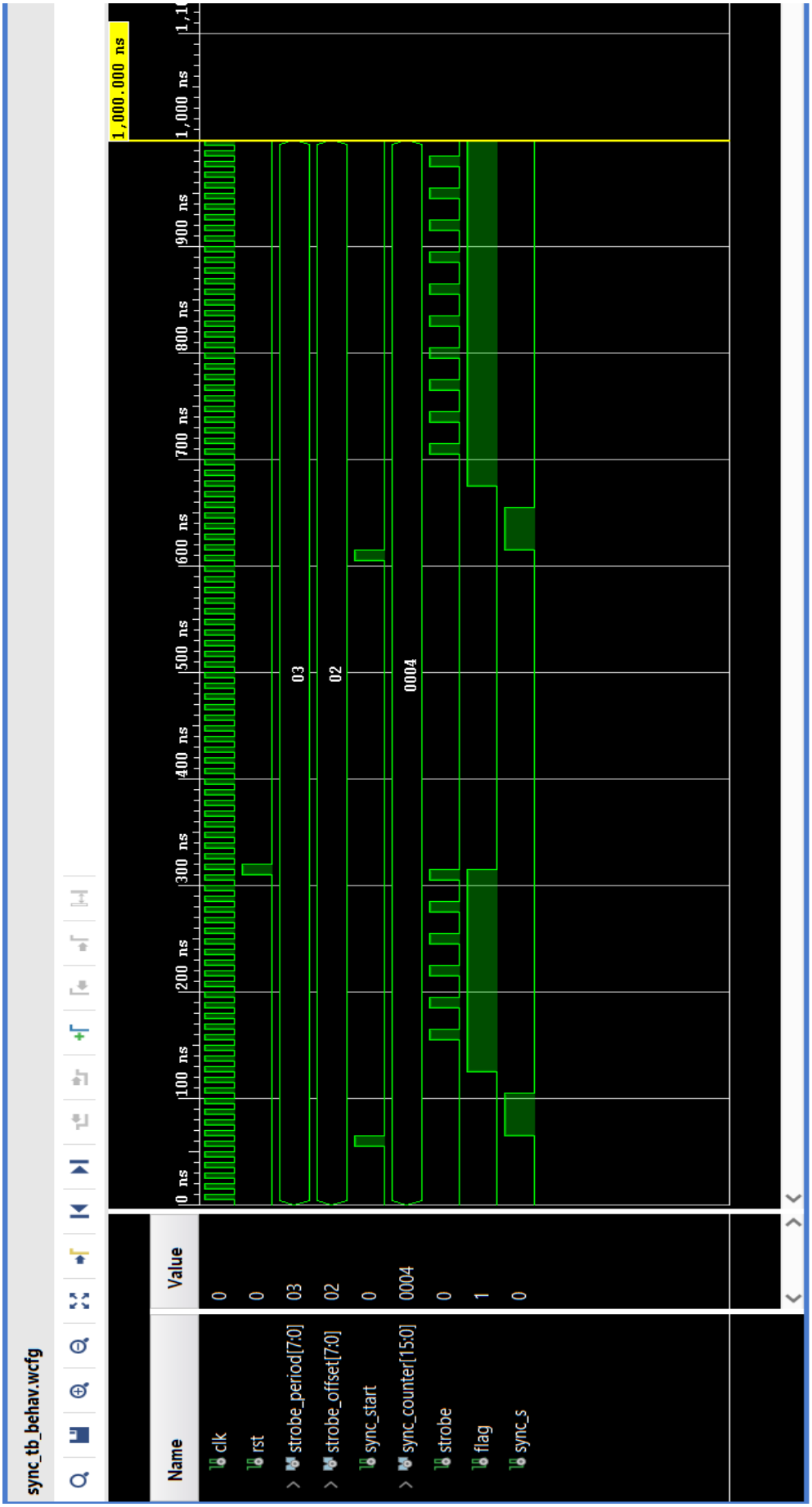
Nakon što je u simulaciji *Vivado* razvojnog okruženja ispitano ponašanje sinkronizacijskog sklopa, sklop je bilo potrebno uklopiti u razvijeni akvizicijski sustav. Sinkronizacijski sklop instancira se unutar logičkih blokova *mdac_mem_top* i *adc_mem_top*. Na ulaze te dvije komponente izravno iz sinkronizacijskog sklopa dovode se dva signala: *strobe* signal i *sync* signal. Kako bi sinkronizacijski sklop mogao komunicirati sa procesorskim sustavom na ZedBoard-u realiziran je još jedan AXI4-Lite podređeni sklop *sync_ctrl_axi_slv*. Unutar *sync_ctrl_axi_slv* podređenog sklopa definira se konfiguracijski *SYNC* registar kojim se upravlja radom ostalih 12 podređenih sklopa. Postavljanjem LSB bita tog registra u '1' započinje se sinkronizacija cijelog sustava. Također, za svaki ADP i DAP definira se SYNC_CTRL konfiguracijski registar. Izgled



Slika 6.2: Izgled konfiguracijskog registra SYNC_CTRL

tog registra prikazan je na slici 6.2. Na gornjih 16 bitova upisuje se iznos *sync_counter* kojim se osigurava dovoljan broj ciklusa takta kako bi se sve komponente uspjele sinkronizirati. ADP-ovi i DAP-ovi tijekom trajanja *sync_counter* ciklusa takta odlaze u stanje pripravnosti i čekaju puls *strobe* signala kako bi mogli započeti s konverzijom odnosno množenjem. Na bitove [15, 8] upisuje se iznos trajanja *strobe_offset* kojim se kompenziraju moguća kašnjenja akvizicijskog sklopa, dok se u donjih 8 bitova upisuje iznos trajanja *strobe_period*. O iznosu *strobe_period* direktno ovisi generiranje *strobe* signala pa se taj period razlikuje za ADP-ove i DAP-ove. Dan je isječak koda C++ aplikacije u kojem je prikazano konfiguriranje SYNC_CTRL registra za jedan ADP i jedan DAP. U ovom slučaju vrijeme, koje je zadano da se akvizicijski sustav sinkronizira iznosi 80 ns što se može iščitati iz gornjih 16 bitova registra SYNC_CTRL u koje je upisan podatak koji predstavlja 8 ciklusa takta. Pomoću *strobe_offset* signala ne kompenzira se nikakvo kašnjenje jer je postavljen u 0. U donjih 8 bitova konfiguracijskog registra SYNC_CTRL za DAP postavljena je vrijednost 10, dok je za ADP postavljena vrijednost 100. Time je omogućeno generiranje *strobe* signala kojim se izravno utječe na način rada odnosno na frekvencije rada ADP-a i DAP-a.

```
*sync_ctrl_ADC = 0x0008000A;
*sync_ctrl_DAC = 0x00080064;
```



Slika 6.3: Rezultati simulacije

7. Testiranje i rezultati

U ovom diplomskom radu *Vivado* razvojno okruženje korišteno je za sintezu i analizu digitalnog dizajna, dok se za razvoj C++ aplikacije koja se izvodi na realiziranom sustavu koristi Xilinx SDK (engl. *Software Development Kit*). Kako bi ostvarili izvođenje željene aplikacije na realiziranom sustavu podignut je PetaLinux operacijski sustav, a proces podizanja PetaLinux-a opisan je u potpoglavlju 3.3.1.

7.1. Testna aplikacija u C++

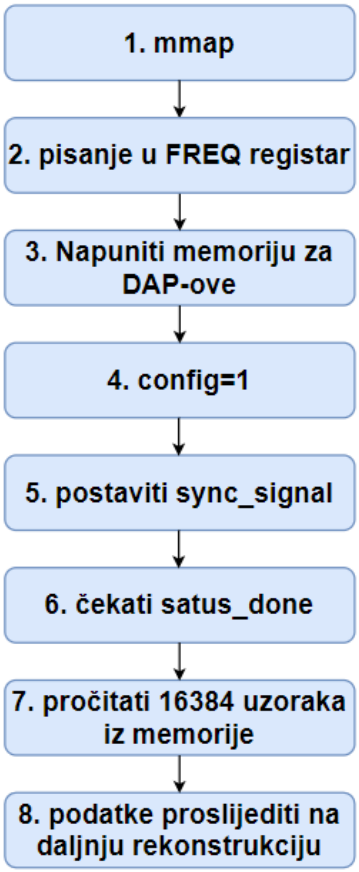
Razvijena aplikacija omogućava komunikaciju između programabilne logike i procesorskog sustava na ZedBoard-u. Na slici 7.1 prikazan je slijed izvođenja aplikacije. Komunikacija između PL-a i PS-a izvedena je AXI4-Lite sučeljem i odvija se putem registara čija je širina 32 bita. Kako procesor ne može izravno pristupati fizičkim adresama registara u svrhu pisanja ili čitanja bilo je potrebno napraviti mapiranje u virtualni adresni prostor. Mapiranje fizičkih adresa registara u virtualni adresni prostor obavlja se na početku aplikacije pozivom funkcije *mmap*. Ova funkcija omogućuje mapiranje dvanaest podređenih sklopova navedenih u tablici 5.1 i tablici 5.2. Zatim se pristupa AXI registru koji sadrži informaciju o frekvenciji rada digitalno-analognog pretvornika. U danom isječku koda u taj registar se zapisuje podatak 10 koji pomnožen s periodom rada takta sustava od 10 ns predstavlja učestalost promjene bitova na ulazima DAP-ova.

```
#define FREQ_REG_OFFSET     0x00000050
*((unsigned *)(ptr +page_offset +(unsigned)(FREQ_REG_OFFSET)))= 10u;
```

Za svaki digitalno-analogni pretvornik na odgovarajuće registre koji su mapirani unutar adresnog prostora procesora potrebno je upisati deset 8-bitnih podataka. Vrijednost podataka se zadaje u aplikaciji i putem AXI sučelja ti se podaci upisuju u realiziranu dvopristupnu memoriju koja se nalazi na PL strani. Dan je primjera zapisivanja 8-bitnog podatka 0xFF u jedan član retka mjerne matrice.

```
#define TEST_REG1            0x00000004
*((unsigned *)(ptr + page_offset + (unsigned)(TEST_REG1))) = 0
    x000000FF;
```

S očitavanjem i upisivanjem podataka u memoriju započinje se kad procesorski sustav na ZedBoard-u postavi signal *adc_config_enable* u visoko stanje čime dojavljuje da je sve spremno za akviziciju. U aplikaciji je to ostvareno postavljanjem pokazivača *config* u '1'. Zatim je potrebno podignuti *sync* signal u '1' kako bi svi ADP-ovi i DAP-ovi krenuli raditi paralelno. Da bi se krenulo s prijenosom očitanih podataka iz PL-a u PS potrebno je pričekati da se registar *axi_status_done* postavi u '1' čime je signalizirano da je memorija puna. U aplikaciji se provjeravanje stanja registra *axi_status_done* vrši preko pokazivača *status_done*. Kako bi uspješno prenijeli podatke iz PL dijela realizirana je *for* petlja raspona [0, 16383] u kojoj se prilikom prolaza svake petlje dohvati po jedan podatak s ADP-a.

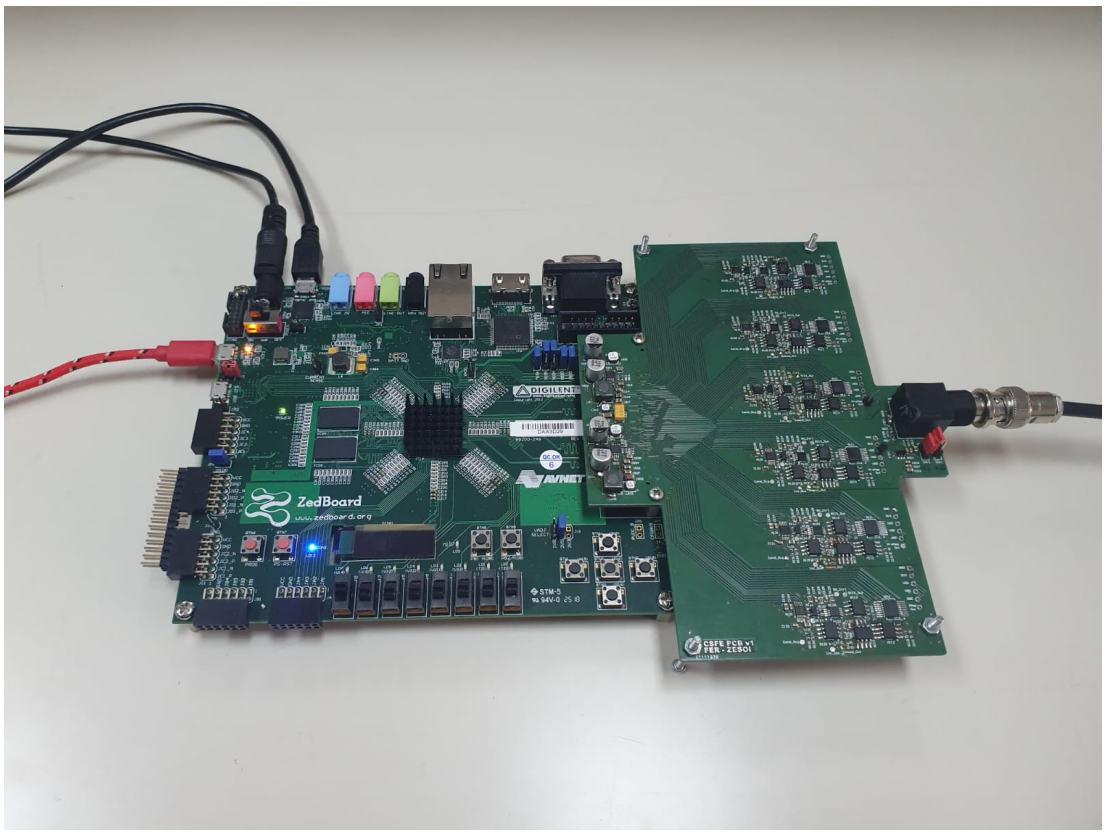


Slika 7.1: Slijed izvođenja aplikacije

Očitani podaci šalju se na daljnju rekonstrukciju. Algoritam za rekonstrukciju signala implementiran je u okviru diplomskog rada *Sustav sažimajućeg očitavanja za akviziciju 1D signala* [8]. Rekonstrukcijski algoritam je napisan u programskom jeziku C++ i implementiran na procesorskom dijelu ZedBoard razvojnog sustava.

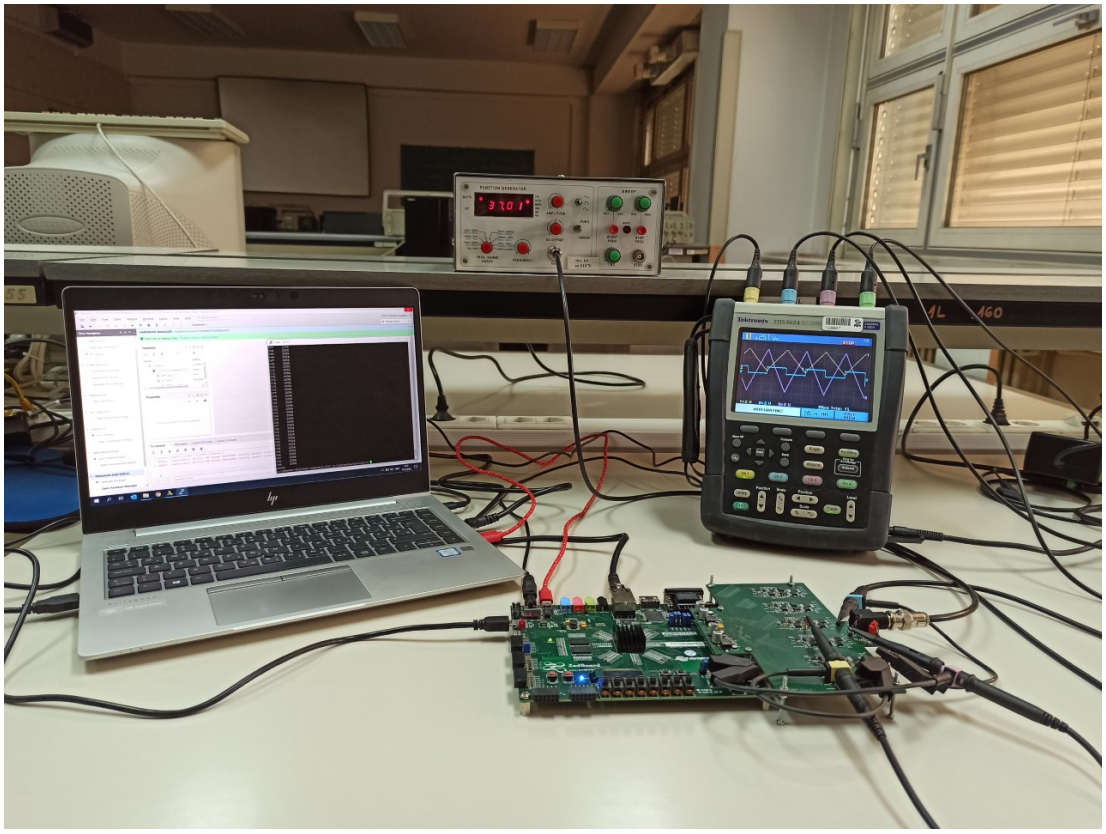
7.2. Uhodavanje sustava

Na slici 7.2 prikazan je akvizicijski sustav koji se sastoji od ZedBoard razvojnog sustava i akvizicijske pločice. S desne strane moguće je uočiti BNC konektor preko kojeg se dovodi ulazni signal kao i šest paralelnih kanala zaduženih za akviziciju signala.



Slika 7.2: Akvizicijski sustav

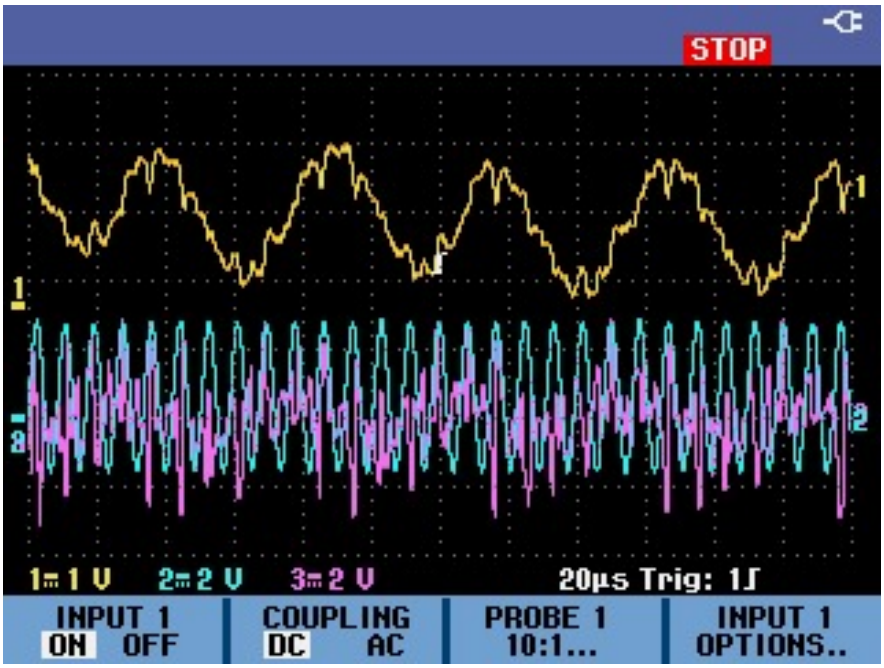
Slika 7.3 prikazuje izgled radne okoline prilikom ispitivanja razvijenog sustava. Sustav za ispitivanje ispravnosti rada čine razvijeni akvizicijski sustav, digitalni četverokanalni osciloskop, laptop na kojem se ispisuju podaci u *Putty* terminalu i funkcijski generator AC signala.



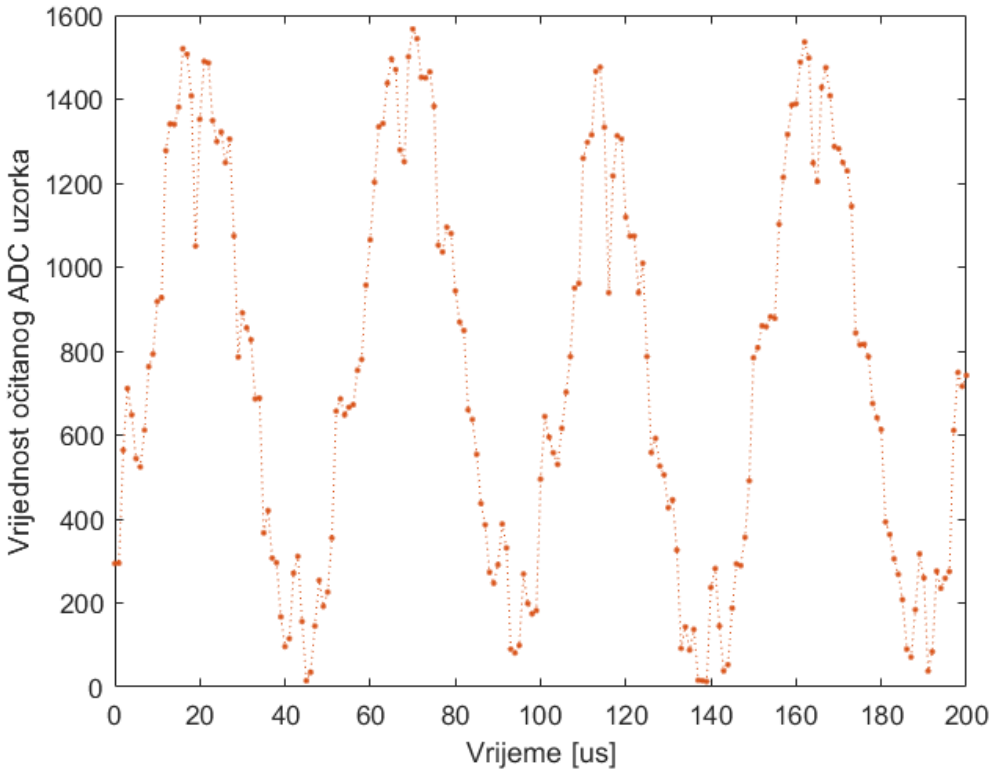
Slika 7.3: Radna okolina

7.3. Rezultati

Kako bi se ispitao rad cijelog kanala dovedeni su različiti ulazni signali koji se množe s nasumičnim vrijednostima mjerne matrice, a te vrijednosti se zadaju u aplikaciji. Na slici 7.4 signal predstavljen plavom bojom označava ulazni signal sinusnog valnog oblika. Ulazni signal se množi vrijednostima mjerne matrice. Kao produkt tog množenja dobiva se signal na izlazu DAP-a koji je predstavljen ljubičastom bojom. Taj se signal slabije vidi jer se nastojalo prikazati više perioda izlaznog signala koji se očitava. Žutom bojom prikazan je signal koji predstavlja ulaz u ADP. Unutar C++ aplikacije omogućeno je spremanje ADP uzoraka u datoteku. Podaci se u datoteku spremaju u parovima gdje prvi podatak predstavlja redni broj uzorka, a drugi vrijednost ADP uzorka. Kako bi se podaci iz datoteke mogli grafički prikazati napisana je skripta unutar programskog okruženja Matlab. Primjer grafičkog prikaza dan je na slici 7.5 gdje su ADP uzorci prikazani pomoću linearne interpolacije.

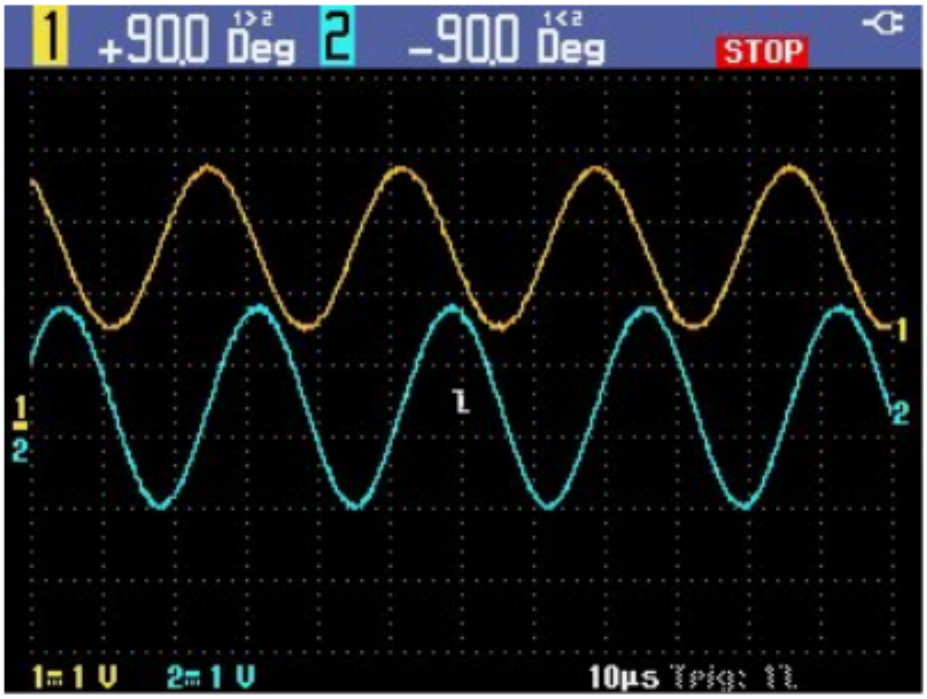


Slika 7.4: Ulaz u ADP-žuta boja, ulazni signal-plava boja, izlaz iz DAP-a-ljubičasta boja

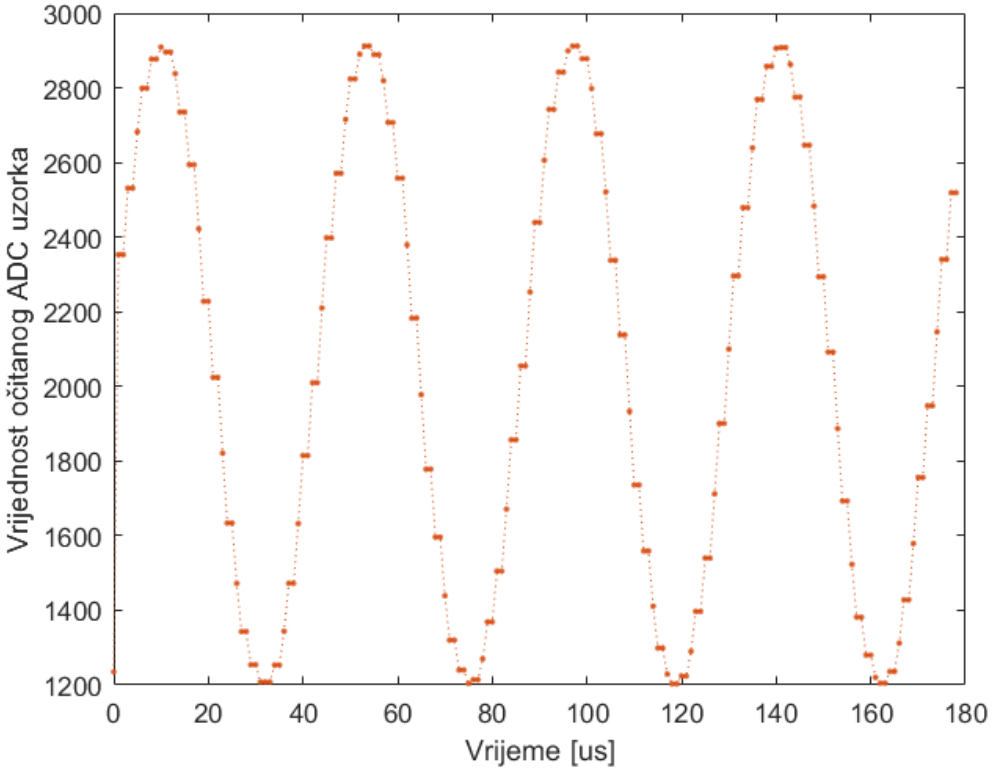


Slika 7.5: Grafički prikaz vrijednosti ADP uzoraka

U sljedećem primjeru ulazni signal sinusnog valnog oblika množi se vrijednostima "0xFF" koje predstavljaju član retka mjerne matrice. Definirana kombinacija bitova na izlazu DAP-a daje izlazni signal koji je jednak ulaznom signalom. Na slici 7.6 plavom bojom prikazan je izlazni signal DAP-a koji je jednak ulaznom signalu sinusnog valnog oblika. Žutom bojom prikazan je signal koji predstavlja izlaz iz integratora odnosno ulaz u ADP. Potrebno je uočiti da se radi o istom valnom obliku koji za ulaznim signalom kasni za 90°. Kao i u prethodnom primjeru podaci se spremaju u datoteku i pokretanjem napisane skripte u programskom okruženju Matlab pojavljuje se grafički prikaz 180 vrijednosti ADP uzoraka koji su interpolirani i prikazani na slici 7.7.



Slika 7.6: Izlaz DAP-a-plava boja, ulaz u ADP-žuta boja



Slika 7.7: Grafički prikaz vrijednosti ADP uzoraka

8. Zaključak

Teorija sažimajućeg očitavanja tvrdi da je rekonstrukcija signala moguća iz puno manjeg broja uzoraka u odnosu na standardne metode koje podliježu Nyquist-Shannon teoremu. Kako bi dokazali ovu teoriju u praksi, u okviru diplomskog rada realiziran je hibridni sustav sastavljen od razvijenog akvizicijskog sklopa koji se preko FMC konektora spaja na razvojnu ploču ZedBoard. Akvizicijski sustav sastoji se od *front-end* dijela implementiranog na akvizicijskoj pločici koja se sastoji od šest kanala i *back-end* dijela koji je realiziran na Zynq-7000 sustavu na čipu. Unutar razvojne okoline *Vivado Design Suite* dizajniran je opisani sustav koji implementira serijsko-paralelni međusklop. Njime se omogućava paralelna komunikaciju za slanje mjerne matrice iz procesorskog sustava ZedBoard i serijska komunikacija za slanje mjernog vektora s akvizicijskog sklopa prema procesorskom sustavu razvojnog sustava ZedBoard. Podaci se putem AXI sučelja šalju na ciljano računalo i pritom je osigurana kompletnost svih podataka.

Razvijen je sinkronizacijski sklop kojim se ostvaruje sinkronizacija ADP-ova i DAP-ova. Aplikacija koja je razvijena u alatu Xilinx SDK pokreće se na podignutom operacijskom sustavu PetaLinux te omogućava komunikaciju između programabilne logike i procesorskog sustava na ZedBoard-u kao i kontrolu slijeda izvođenja programa. U posljednjem poglavlju dan je prikaz rezultata u Matlab okruženju. Jedno od mogućih poboljšanja ovog sustava odnosi se na potrebu za pohranjivanjem podataka na PL strani. U izvedenoj implementaciji prvo se određeni broj mjerenja sprema u realiziranu dvopristupnu memoriju na PL-u te se putem AXI sučelja ti podaci šalju na PS. Moguća izmjena i nadogradnja odnosi se na izravni prijenos podataka s analogno-digitalnog pretvornika na procesorski sustav koji je zadužen za rekonstrukciju signala, bez potrebe za spremanjem u memoriju.

. . . . . .

LITERATURA

[1] SoCs with Hardware and Software programmability. 2020. URL https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[2] *1 MSPS, Ultralow Power,12-Bit ADC in 10-Lead LFCSP and MSOP*. Analog Devices, 2016. URL http://www.farnell.com/datasheets/2250666.pdf?_ga=2.11038452.1965143953.1581966023-780011795.1571923239.

[3] *8-/10-/12-Bit, High Bandwidth, Multiplying DACs with Parallel Interface*. Analog Devices, 2017. URL http://www.farnell.com/datasheets/52538.pdf.

[4] *ZedBoard (Zynq™ Evaluation and Development) Hardware User's Guide*. Avnet, 2014. URL http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[5] *ZedBoard Getting Started Guide*. Avnet, 2017. URL http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf.

[6] I.Marinović. Akvizicijski sustav iznad nyquistove granice.

[7] HU Jie. Research transplanting method of embedded linux kernel based on arm platform. 2010. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5573883&tag=1.

[8] K.Sever. Sustav sažimajućeg očitavanja za akviziciju 1D signala. 2019.

[9] Petar Matković. Razvoj demonstracijskih aplikacija na zynq-7000 soc arhitekturi pod operacijskim sustavom Linux. 2015. URL https://bib.irb.hr/datoteka/779615.Final_0036456409_48.pdf.

[10] Meenu Rani, S. B Dhok, i R. B Deshmukh. A systematic review of compressive sensing: Concepts, implementations and applications. 2018. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8260873&tag=1.

[11] Tin Vlašić, Damir Seršić, et al. *A system for compressive sensing of analog signals*. Fakultet elektrotehnike i računarstva, 2019.

[12] V.Papa. Akvizicijski sklop sustava sažimajućeg očitavanja.

[13] *AXI Reference Guide*. Xilinx, 2011. URL https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[14] *Vivado Design Suite - Vivado AXI Reference*. Xilinx, 2017. URL https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.

[15] *PetaLinux Tools Documentation - Reference Guide*. Xilinx, 2018. URL https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1144-petalinux-tools-reference-guide.pdf.

[16] *PetaLinux Tools Documentation - PetaLinux Command Line Reference*. Xilinx, 2018. URL https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1157-petalinux-tools-command-line-guide.pdf.

[17] *Zynq-7000 SoC- Technical Reference Manual*. Xilinx, 2018. URL https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

Nove paradigme akvizicije podataka hibridnim sustavom

Sažetak

U ovom diplomskom radu osmišljen je i realiziran sustav za akviziciju 1D podataka korištenjem ZedBoard Zynq-7000 ARM/FPGA SoC razvojnog sustava. Za implementaciju akvizicijskog sustava kojim se omogućava pohrana podataka i paralelno-serijska komunikacija između programabilne logike i procesorskog sustava na ZedBoard-u korišteno je razvojno okruženje *Vivado Design Suite*. Sklop osigurava pohranu i brzi prijenos podataka sa šesterokanalnog SPI sučelja i učitavanje na ugradbeni računalni sustav bez gubitka podataka. Kako bi se omogućilo izvođenje željene C++ aplikacije na razvojnom sustavu ZedBoard podignut je operacijski sustav Linux. U programskom okruženju Matlab prikazani su rezultati mjerenja, točnije vrijednosti ADP uzoraka za različite kombinacije ulaznog signala i mjerne matrice s kojom se ulazni signal množi.

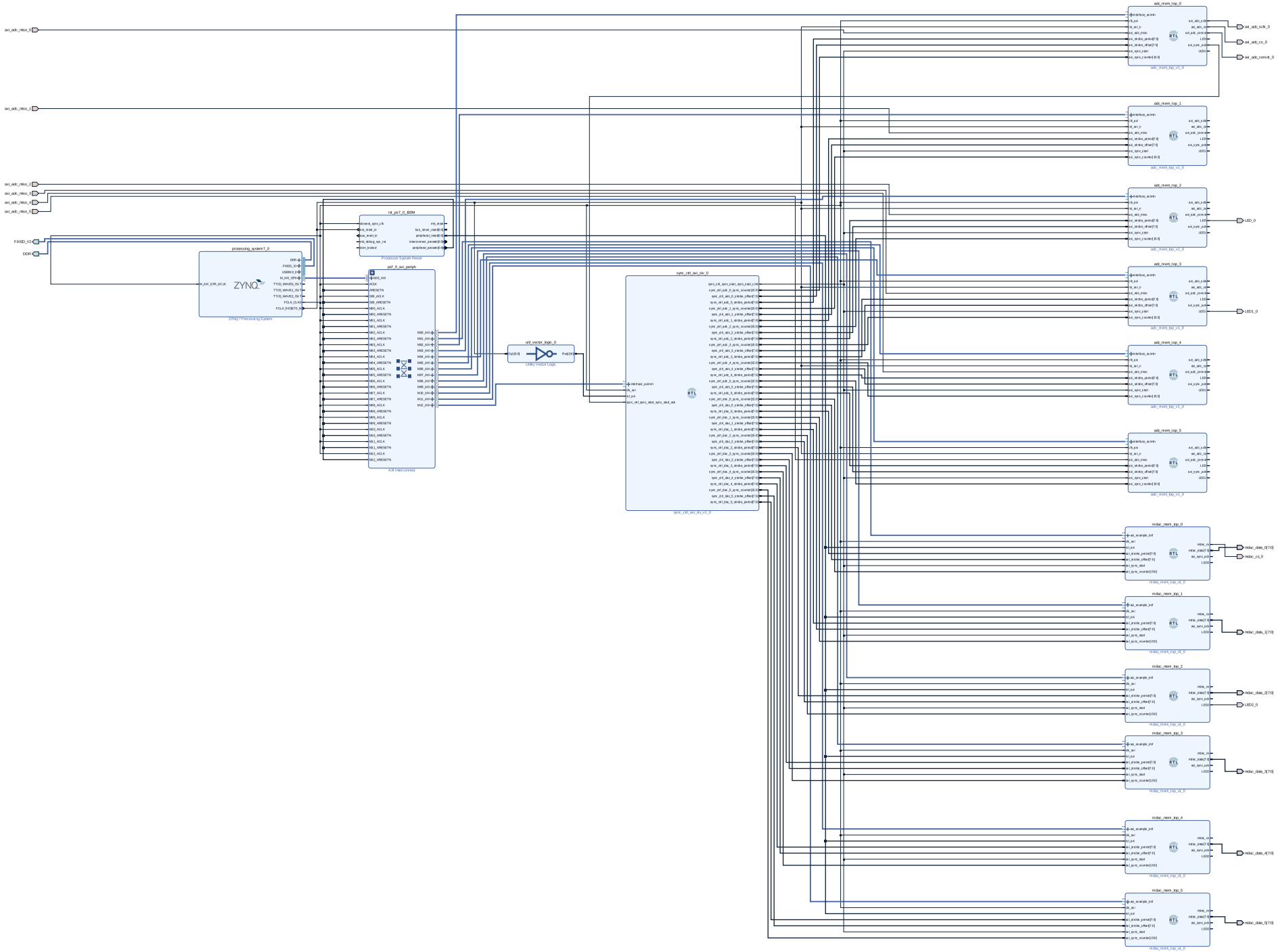
Ključne riječi: sažimajuće očitavanje, ZedBoard, AXI, FPGA, analogno-digitalni pretvornik, digitalno-analogni pretvornik, dvopristupna memorija, VHDL, PetaLinux

New paradigms for data acquisition using a hybrid system

Abstract

In this master thesis, a system for acquisition 1D data was designed and implemented using the ZedBoard Zynq-7000 ARM / FPGA SoC development system. The Vivado Design Suite development environment is used to implement an acquisition system that enables data storage and parallel-serial communication between programmable logic and the processor system on ZedBoard. The implemented system provides storage and fast data transfer from a six-channel SPI interface and loading on an embedded system without loss of data. A Linux operating system was built on ZedBoard development system. Developed PetaLinux application forwards acquired data to reconstruction algorithm. In the Matlab software environment, the measurement results are displayed. The system was tested on various input signals and different combinations of the measurement matrix.

Keywords: compressive sensing, ZedBoard, AXI, FPGA, analog-to-digital converter, digital-to-analog converter, true dual port RAM, VHDL, PetaLinux


axi_adc_rtso_0
axi_adc_rtso_1
axi_adc_rtso_2
axi_adc_rtso_3
axi_adc_rtso_4
axi_adc_rtso_5
FIXED_IO
DDR
processing_system7_0
ZYNQ7 Processing System
rst_ps7_0_100M
Processor System Reset
ps7_0_axi_periph
AXI Interconnect
util_vector_logic_0
Utility Vector Logic
sync_ctrl_axi_slv_0
sync_ctrl_axi_slv_v1_0
adc_rtxrrt_top_0
adc_rtxrrt_top_v1_0
adc_rtxrrt_top_1
adc_rtxrrt_top_2
adc_rtxrrt_top_3
adc_rtxrrt_top_4
adc_rtxrrt_top_5
rtxdac_rtxrrt_top_0
rtxdac_rtxrrt_top_v1_0
rtxdac_rtxrrt_top_1
rtxdac_rtxrrt_top_2
rtxdac_rtxrrt_top_3
rtxdac_rtxrrt_top_4
rtxdac_rtxrrt_top_5
axi_adc_sclk_0
axi_adc_cs_0
axi_adc_convst_0
LED_0
LED1_0
LED2_0
rtxdac_data_0[7:0]
rtxdac_data_1[7:0]
rtxdac_data_2[7:0]
rtxdac_data_3[7:0]
rtxdac_data_4[7:0]
rtxdac_data_5[7:0]