**Facial recognition**

*Dokumentacija za predmet: Embeded operativni sistemi*

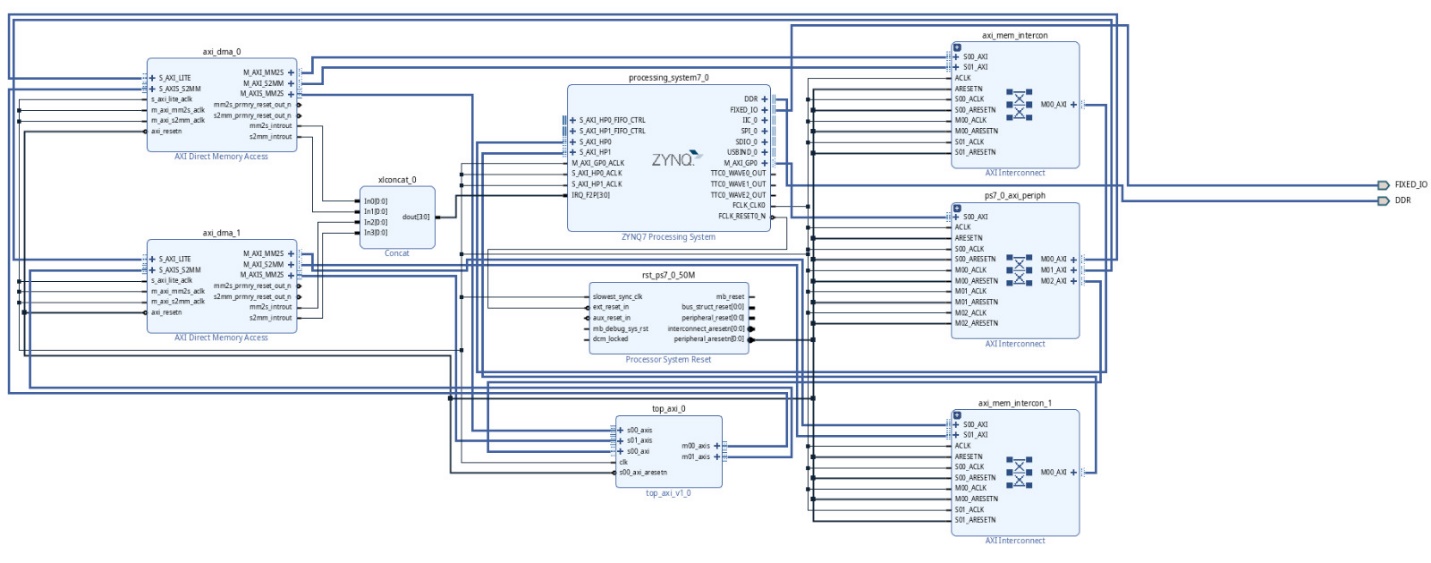
1. Opis algoritma

Ideja projekta je prepoznavanje ljudskog lica na fotografiji. Fotografija se prosleđuje programu, a on iscrtava okvir oko detektovanog lica. Centralni dio algoritma predstavlja filtriranje slike (po x-osi i y-osi) koje podrazumijeva množenje manjih dijelova slike sa filterima. Naime, dimenzije filtera su 3x3, te se svaki komad slike dimenzija 3x3 množi sa njime. Šablon se ponavlja sve dok se ne prođe kroz čitavu sliku pomjerajući se prvo po x-osi za 1 piksel, a nakon obrađenog reda za 1 piksel po y-osi. Ostatak algoritma sprovodi niz transformacija nad slikom čiji je krajnji rezultat skup HOG vektora. Zatim je potrebno pronaći skup HOG vektora na dijelu originalne slike koji je sličan skupu HOG vektora šablona slike. Algoritam ekstrahuje HOG šablona slike kojoj se mijenjaju dimenzije počevši od minimalne dimenzije pa do trećine minimalne dimenzije originalne slike da bi se omogućilo prepoznavanje lica bez obzira na njegovu veličinu na originalnoj slici. Sa različitim veličinama HOG-a šablona slike algoritam prolazi kroz originalnu sliku i pamti one koje su iznad određenog praga (threshold).

*Slika 1.1 Originalna slika Slika 1.2 Šablon slika Slika 1.3 Izlazna slika*

Na predmetu Projektovanje složenih digitalnih sistema algoritam je optimizovan. Konačan izgled sistema prikazan je na *Slici 1*.*1.*



*Slika 1.1 Blok dizajn sistema*

Koristi se jedan GP port za čitanje i upis odgovarajućih registara kao i za konfigurisanje 2 DMA kontrolera. Registri za upis i čitanje su: SAHE1, SAHE2, SAHE3 u kome se nalaze svi potrebni konfiguracioni parametri potrebni za rad algoritma. Takođe, na ovaj način se konfigurišu i odgovarajući registri unutar DMA kontrolera.

1. Razvoj drajvera

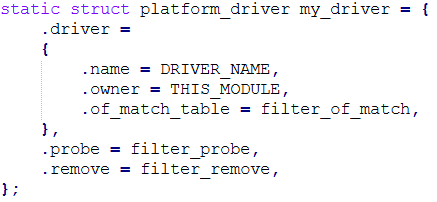
2.2 *Platform\_driver* i *of\_device\_id* struture

Na *Slici 1.1* može se uočiti da sistem treba da pristupa DMA kontrolerima i samom IP jezgru. Drajver se sastoji iz *platform\_driver* strukture i nekoliko funkcija koje će biti opisane u nastavku.

Struktura prikazana na *Slici 2.1* inicijalizuje hardverski sistem. Unutar ove strukture nalazi se još jedna struktura *driver*, kao i pokazivači na *filter\_probe* i *filter\_remove* funkciju (opisane kasnije). Struktura *driver* ima tri polja:

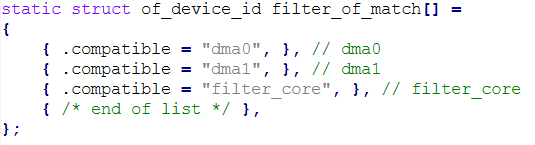
1. *name* – string koji predstavlja ime drajvera. Njemu se prosljeđuje makro DRIVER\_NAME koji ima vrijednost *filter\_core*.
2. *owner* – predstavlja vlasnika drajvera, prosljeđuje mu se makro THIS\_MODULE, što je najčešća praksa.
3. *of\_match\_table* – polje koje pokazuje na niz-strukturu *of\_device\_id*, sa nazivom *filter\_of\_match*.

Na *Slici 2.1* prikazana je struktura *my\_driver*.



*Slika 2.1 Struktura my driver*

Niz-struktura  *оf\_device\_id* prikazana je na *Slici 2.2.* Sadrži *compatible* polja u kojima se navode imena svih uređaja za koje je namjenjen drajver. Imena se moraju poklapati sa imenima uređaja u *device tree* na osnovu čega kernel može utvrditi fizičke adrese uređaja i povezati ih sa drajverom. U kreiranom stablu se nalaze tri modula *“dma0”*, *“dma1”,*  *“filter\_core”* te su njihova imena navedana unutar *compatible* polja.



*Slika 2.2 Struktura of\_device\_id*

2.3 Funkcije *filter\_init* i *filter\_exit*

*Filter\_init*  funkcija sluzi za dinamičko povezivanje modula i kernela. Ona se poziva uz pomoć *insmode* komande i na taj način ubacujemo modul u kernel. Ova funkcija dinamički alocira upravljačke brojeve uz pomoć *alloc\_chrdev\_region.* U sistemu se nalaze 3 modula, alocira se isti broj upravljačkih brojeva. Zatim se kreiraju *node* fajlovi u */dev* folderu za svaki od 3 modula iz *compatible* polja. To se radi uz pomoć dvije funkcije *class\_create* i *device\_create,* povratna vrijednost *class\_create* se koristi kao prvi parameter u funkciji *device\_create*. Ako se pronađe odgovarajuća vrijednost *compatible* polja poziva se *filter\_probe* funkcija.

Uz pomoć *cdev* strukture funkcija *filter\_init*  povezuje *node* fajlove sa funkcionalnostima koje treba da budu izvršene kada se pozove neka od funkcija za rad sa fajlovima. Na primjer, ukoliko se nešto čita iz */dev/filter\_core* structure, kernel će pozvati funkciju na koju pokazuje polje *read* u strukturi *filtre\_operations* (*Slika 2.3*). Za DMA kontrolere potrebno je alocirati kontinualnu memoriju za upotrebu u drajveru. To se radi uz pomoć *dma\_alloc\_coherent.* S obzirom na to da sistem ima 2 DMA kontrolera i da je potrebno čitati iz DRAM-a i upisivati u DRAM alocira se memorija za četiri virtuena bafera. Ako se neka od naredbi ne izvrši uspiješno, sve prethodne se poništavaju.

*Filter\_exit* funkcija ima suprotnu funkcionalnosti. Uz pomoć ove funkcije se modul isključuje iz kernela, briše se *cdev* struktura, svi *node* fajlovi (*device\_destroy* i *class\_destroy*), oslobađa se memorija za virutuelne bafere kao i upravljački brojevi (*unregister\_chrdev\_region*). Poslije *filter\_exit* funkcije poziva se *filter\_remove*.

2.4 Funkcije *filter\_probe* i *filter\_remove*

Funkcija *filter\_probe* služi za inicijalizaciju uređaja i alociranje resursa koji su potrebni za rad sa njim. Prvo se pribavljaju fizičke adrese uređaja odnosno *mem\_start*, *mem\_end* i *base\_addr.* Potrebno je rezervisati opseg adresa da ih kernel ne bi dodijelio nečemu drugome. To se vrši uz pomoć funkcije *request\_mem\_region.* Ukoliko je zauzimanje prošlo uspiješno, fizičke adrese se mogu mapirati na virtuelne kako bi se moglo čitati i upisivati u te memorijske lokacije. To se postiže funkcijom *ioremap*.Ukoliko se vrši inicijalizacija DMA uređaja dodatno se poziva funkcija *platform\_get\_irq* koja vraća broj prekida tog uređaja.

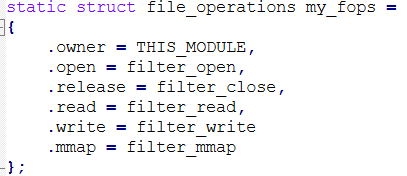
Ukoliko se neki korak nije uspiješno izvršio svi prethodni koraci se poništavaju. U suprotnom, drajver posjeduje virtuelnu adresu uređaja. Preko te adrese može da upisuje odnosno čita iz proizvoljnih lokacija uređaja uz pomoć *iowrite32* i *ioread32* funkcije.

Funkcija *filter\_remove,* ima suprotnu funkcionalnost. Zaustavlja rad uređaja uz oslobađanje svih resursa koje zauzima prethodna.

S obzirom da se ove funkcije pozivaju više puta, uvodi se pomoćna promjenljiva *probe\_counter,* koja vodi računa o redoslijedu inicijalizacije i oslobađanja resursa.

2.5 Funkcije za rad sa fajlovima

U strukturi *file\_operations* nalaze se pokazivači na sve funkcije koje rade sa fajlovima. Od interesa su funkcije za čitanje, upis i memorijsko mapiranje. Struktura *file\_operations* prikazana je na *Slici 2.3*.



*Slika 2.3 Struktura file\_operations*

Funkcija za upis podataka je *filter\_write.* Ona omogućava upis u *filter\_core* modul, inicijalizaciju i konfiguraciju DMA. Očekuje dva ulazna parametra iz bafera: vrijednost i adresu na koju se treba upisati u odogovarajućem formatu (odnosi se na filter\_core), vrijednosti veličine paketa koji se prenosi u bajtovima (odnosi se na DMA). Pomoću upravljačkog broja se određuje u kom medujumu i na koju adresu je potrebno upisati datu vrijednost. Prije konfiguracije DMA kontrolera poziva se *DMA\_init* funkcija (opisana u nastavku).

Za čitanje se koristi *filter\_read* funkcija. Ona takođe čita samo iz *filter\_core* modula, a u zavisnosti od upravljačkog broja zna koji je ciljni modul. Ona korisniku salje vrijednosti svih mogućih lokacija u okviru tog modula.

*Filter\_map* se koristi za mapiranje fizičke memorije DMA bafera u virtuelnu memoriju korisničkog prostora koristeći *mmap* poziv. Dobije se upravljački broj iz inode fajla kako bi se identifikovalo o kom DMA uređaju je riječ. Izračunava se veličina memorije koja treba da bude mapirana. Funkcija razlikuje konfigurisanje DMA za čitanje iz DDR-a i upis u isti. Za upis se koriste *tx\_vir\_buffer*, a za čitanje *rx\_vir\_buffer*. Za mapiranje se koristi *dma\_mmap\_coherent* koja omogućava mapiranje DMA memorije u virutuelni prostor.

2.6 *dma\_init*, *dma\_isr*, *dma\_simple\_write*, *dma\_simple\_read* funkcije

*Dma\_init* funkcija podešava određene bitove unutar registara za DMA. Prvo postavlja reset bit na visoku vrijednost, što će resetovati DMA kontroler. Nakon toga se omogućavaju prekidi na oba kanala.

*Dma0\_isr i dma1\_isr* su prekidne rutina za rukovanje prekidima u oba kontrolera. One se pozivaju kada DMA generiše prekide. Prvo se pročita stanje prekida, a zatim se očisti stanje prekida.

*Dma\_simple\_write* ima za cilj da pokrene DMA transakciju koja prenosi podatke iz memorije. Konfiguriše se MM2S\_DMACR\_reg, zatim se postavi veličina transakcije i početna adresa.

*Dma\_simple\_read* ima za cilj da pokrene DMA transakciju koja prenosi podatke iz perifernog uređaja u memoriju. Nakon konfiguracije registara, pokreće se DMA transakcija.

3. Razvoj aplikacije

Za prethodno razvijeni drajver potrebno je napisati odgovarajuću aplikaciju. Aplikacija treba da liči na sw modul iz virtuelne platforme što je i ostvareno.

Za upis u registre koristi se funkcija *write\_hard*. Argumenti koji se prosljeđuju ovoj funkciji su vrijednost i adresa na koju se upisuje ta vrijednost. To se upisuje u */dev/filter\_core* *fajl,* kako drajver i očekuje. Iz jezgra se čita uz pomoć *read\_hard* *funkcije.* Ova funkcija čita iz svih registara, ali će korisniku biti vraćen onaj registar za koji je data adresa.

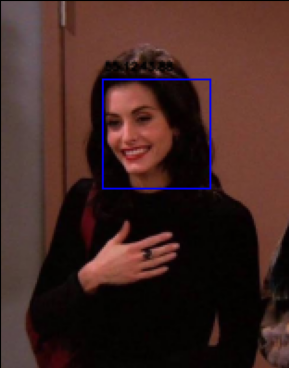
Za konfigurisanje DMA kontrolera koristi se funkcija *config\_dma* kao prvi parametar, ovoj funkciji se prosljeđuje veličina transakcije (u bajtovima) za čitanje iz DRAMA, a kao drugi parametar veličina transakcije (u bajtovima) za upis u DRAM, treći parametar određuje koji kontroler se konfiguriše.

Takođe, poziva se mmap poziv za mapiranje fizičke memorije u virtuelnu memoriju korisničkog prostora.

1. Podizanje sistema i testiranje na uređaju

Za podizanje Linux operativnog sistema na Zybo razvojnoj ploči korišćeno je razvojno okruženje Petalinux. Ovaj alat sadrži sve programe koji su potrebni za podizanje, razvijanje, debagovanje i testiranje Linux embeded platforme. Petalinux generiše četiri glavne komponente koje su potrebne za rad Linux OS-a. Linux kernel (image.ub), stablo uređaja (system.dtb), Bootloader (BOOT.BIN) i Root fajl sistem (rootfs.tar.gz). Ovi fajlovi se u pravilnom formatu prebacuju na SD karticu sa koje se Linux OS podiže svaki put kada se ploča uključi.

Nakon što je sistem podignut pristupilo se testiranju sistema. Sistem je testiran za detektovanje lica na fotografijama koje su korištene kao testni uzorci prilikom simulacije u virtuelnoj platformi. Vrijednosti koje su u DRAM memoriji su identične očekivanim što predstavlja dodatnu verifikaciju rada sistema. S obzirom na to da su vrijednosti filterovanih piksela očekivane, pozivom još nekoliko funkcija dobijaju se rezultati koridnata lica na slici. Ulazna fotografija prikazana je na *Slici 5.1,* dok je izlaz algoritma prikazan na *Slici 5.2.*

*Slika 5.1 Ulazna slika Slika 5.2 Izlazna slika*