Razvoj upravljačkog programa PCIe sabirnicom na strani PC računala

Marković, Bože Eugen

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAZVOJ UPRAVLJAČKOG PROGRAMA PCIe SABIRNICOM NA STRANI PC RAČUNALA

Diplomski rad

Bože Eugen Marković

Osijek, 2018.

SADRŽAJ

1. UVOD

U automobilskoj industriji trendovi razvoja pokazuju porast u korištenju elektroničkih komponenata. Prvenstveno se radi o primjeni upravljačkih sustava i senzora. Najogledniji primjer su mnogi moderniji automobili sa senzorima za pomoć prilikom parkiranja. Računalni sustavi obrađuju informacije iz okoline koje dobivaju pomoću senzora i na taj način pomažu vozaču prilikom vožnje. Algoritmi koji se implementiraju u te računalne sustave nazivaju se ADAS (engl. (*Advanced driver-assistance systems*) algoritmi. ADAS algoritmi zaduženi su za detekciju i klasifikaciju objekata u okolini i predviđanje mogućih nesreća te na taj način osiguravaju veću razinu sigurnosti u prometu. Razvijeni su algoritmi za kontrolu razmaka kod vožnje u kolonama, nadzor promjene kolničke trake i slično. Prema tome i trendovima razvoja može se u skorijoj budućnosti očekivati visoka razina autonomnosti vozila [1].

Razvojem sve složenijih ADAS algoritama otežano je njihovo testiranje. Testiranje je zaduženo za validaciju odluka algoritama i stavlja naglasak na nedvosmislenost tih odluka. Algoritme je potrebno ispitati slijedno. Drugim riječima, svaka promjena u algoritmu treba se testirati u realnim uvjetima u prometu ili na poligonu predviđenom za ovakva testiranja. Promatrajući cjelokupan sustav, testiranje predstavlja složen proces. Cijena poligona i rizici testiranja algoritma u stvarnim uvjetima u prometu predstavljaju najveće probleme, stoga se pristupa alternativnim rješenjima. Razvijaju se sustavi za ispitivanje kojima bi se sustavi za računalnu percepciju okoline vozila mogli testirati u laboratorijima, a ne u stvarnim uvjetima [1].

Jedan takav uređaj je i AMV (engl. *Automotive Machine Vision*) ADAS *Grabber* uređaj za istovremeno preuzimanje i reprodukciju video tokova. Omogućuje prikupljanje video sadržaja s devet kamera i simultanu reprodukciju prikupljenog video sadržaja. U tipičnim test slučajevima kamere su prostorno raspoređene po vozilu i svaka ima svoju zadaću. AMV *Grabber* podržava prijenos video signala velikim brzinama preko PCI (engl. *Peripheral Component Interconnect*) *Express* sabirnice. Ovim sučeljem omogućeno je i upravljanje uređajem [1].

PCI *Express* sabirnica predstavlja standard komunikacije kojeg karakterizira visoka brzina prijenosa podataka. Jedna od primjena ovog standarda prijenos je višestrukih video tokova za uređaje koji imaju zahtjeve za rad u stvarnom vremenu. Razvijaju se uređaji koji snimaju video s kamera, skladište ga na PC računalo i kasnije predaju snimljeni video ADAS platformi na kojoj se izvršavaju pripadajući algoritmi za obradu video podataka [1]. U okviru ovog rada razvijen je upravljački program (engl. *driver*) za PCI *Express* sabirnicu koji omogućava komunikaciju i prijenos velikih količina video podataka s uređaja za snimanje i verifikaciju na PC računalo i

obratno. Uređaj koji se koristi za snimanje i verifikaciju je AMV *Grabber*. Rješenje je ostvareno u radnom okruženju *Windows* operacijskog sustava, u programskom jeziku C.

Ovaj diplomski rad podijeljen je na pet poglavlja uključujući uvod i zaključak. Drugo poglavlje sadrži osnovne teorijske opise AMV *Grabber*-a, *Windows* operacijskog sustava, *Windows* upravljačkih programa i PCI *Express* sabirnice. Treće poglavlje predstavlja glavne koncepte korištene kod implementacije rješenja, podijeljeno je na inicijalizaciju upravljačkog programa i povezivanje s aplikacijama. U četvrtom poglavlju dani su rezultati testiranja izrađenog programskog rješenja te analiza dobivenih rezultata.

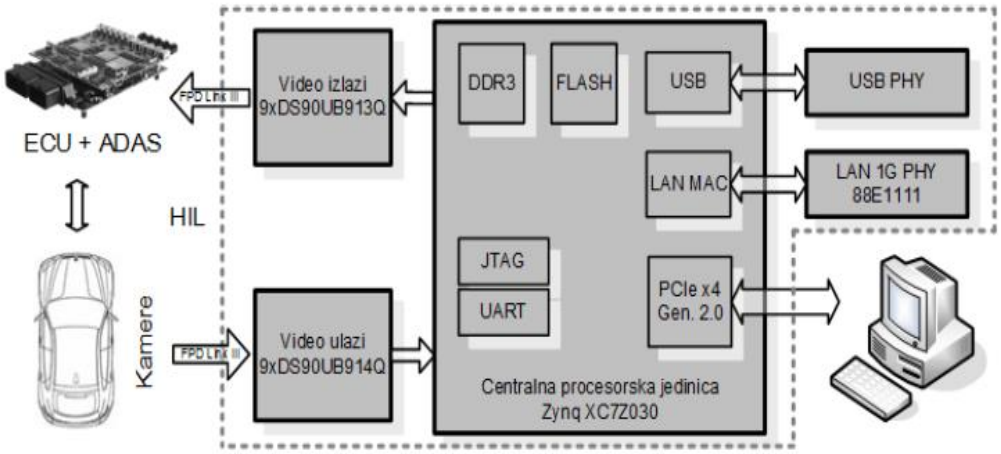
2. TEORIJSKA POZADINA

2.1. AMV *Grabber*

AMV *Grabber* prikazan je na slici 2.1.

Sl. 2.1. AMV *Grabber* [1].

Uređaj je baziran na centralnoj procesorskoj jedinici koja je realizirana u obliku hibridne programabilne mreže *Zynq XC7Z030*. Uz FPGA (engl. *Field-programmable gate array*) dio sadržava i ARM (engl. *Advanced RISC Machine*) Cortex-A9 procesor s dvije jezgre [1]. Arhitektura AMV *Grabber* uređaja s pripadajućim podsustavima prikazana je na slici 2.2.



Sl. 2.2. Arhitektura AMV *Grabbera* [1].

Video ulazi podrazumijevaju 9 kamera koje mogu biti podešene za rad u tri različite rezolucije:

- 1280x1080
- 1280x800
- 1280x720.

U ovom radu koristi se RDACM24B platforma za kameru s OV10640 senzorom prikazana na slici 2.3 [2].

Sl. 2.3. Kamera [2].

2.2. *Windows* operacijski sustav

Windows operacijski sustav razdvaja korisničke aplikacije od samog operacijskog sustava. Programski kod jezgre operacijskog sustava izvodi se zasebno u privilegiranom procesorskom načinu rada – jezgrinom načinu (engl. *kernel mode*). Jezgrin način rada ima pristup osjetljivim sistemskim podacima i samom hardveru. Korisničke aplikacije izvode se u korisničkom načinu rada (engl. *user mode*) s ograničenim mogućnostima bez izravnog pristupa hardveru. Kada se iz korisničke aplikacije poziva neki od servisa operacijskog sustava, procesor izvodi poseban set instrukcija kako bi se nit u kojoj se izvodi korisnička aplikacija prebacila u jezgrin način rada i time se proširuje skup mogućnosti za spomenutu nit. Nakon izvođenja potrebnog *Windows* servisa nit se vraća u korisnički način rada i nastavlja s izvođenjem [3].

HAL (engl. *Hardware Abstraction Layer*) je modul koji omogućuje sučelje prema hardverskim platformama na kojima se *Windows* izvodi. Ovaj modul sakriva njihovu složenost i pridonosi portabilnosti programskog koda. Upravljački programi predstavljaju sučelje između I/O

upravitelja (engl. *Input/Output manager*) operacijskog sustava i HAL-a. Izvode se u jezgrinom načinu rada u jednom od tri različita konteksta:

- Kontekst korisničke niti koja je inicirala I/O zahtjev.
- Kontekst sustavske niti koja se izvodi u jezgrinom načinu rada.
- Rezultat prekida – van konteksta konkretnog procesa ili niti (prekida se nit koja se u tom trenutku izvodila) [3].

Upravljački programi ne upravljaju hardverom direktno, već pozivaju HAL funkcije kako bi se osposobilo sučelje prema hardveru [3].

Za razvoj i testiranje upravljačkih programa koristi se WDK (engl. *Windows Driver Kit*). WDK zahtijeva *Microsoft Visual Studio* SDK (engl. *Software Development Kit*). Kompatibilne verzije WDK-a i *Visual Studia* stvaraju integriranu okolinu za razvoj upravljačkog programa. WDK uključuje dokumentaciju s opsežnim opisima funkcija jezgre operacijskog sustava. Sadržava i potrebne biblioteke koje definiraju interne strukture podataka, konstante i sučelja prema mnogim internim sistemskim funkcijama. Ključne zaglavlja su *ntddk.h, ntifs.h* i *wdm.h*. Ova zaglavlja sadržavaju posebne funkcije, strukture i tipove podataka korištene za razvoj upravljačkih programa [3].

2.3. *Windows* upravljački programi

Prije razvoja upravljačkog programa potrebno je imati određena predznanja u pogledu funkcioniranja *Windows* operacijskog sustava. Potrebna predznanja prije svega podrazumijevaju razumijevanje C programskog jezika, s obzirom da se WDM (engl. *Windows Driver Model*) upravljački programi pišu u C-u.

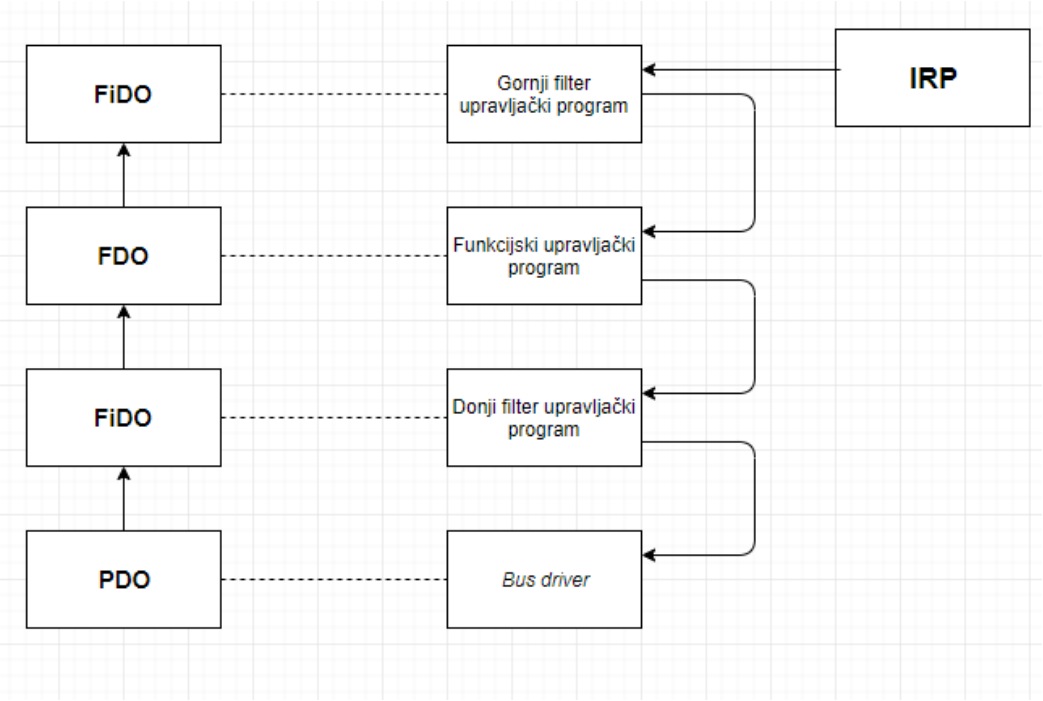
Svrha upravljačkog programa upravljanje je zadanim hardverom ili dijelom hardvera. Upravljački program može se promatrati kao skup funkcija koje operacijski sustav poziva i obavlja različite operacije kako bi hardver funkcionirao na način na koji se to od njega očekuje. Slično kao i jednostavna PC aplikacija s izvršnom datotekom koja ima „*.exe*“ ekstenziju, upravljački program ima izvršnu datoteku s ekstenzijom „*.sys*“. Isto tako, upravljački program određene funkcije može dinamički povezivati iz jezgre operacijskog sustava ili drugih podržanih biblioteka. S druge strane, upravljački program nema glavnu funkciju (ekvivalent *main* funkciji u C aplikaciji), nego ima kolekciju funkcija koje *Windows* poziva po potrebi. U suštini, upravljački program je zadužen za vlastiti hardver, a za sve ostalo je zadužen operacijski sustav, uključujući i odlučivanje o tome

kada će se izvršavati programski kod upravljačkog programa. Primjerice, kada je potrebno obaviti određenu zadaću poput rukovanja prekidom (engl. *interrupt handling*) ili obrade zahtjeva za čitanjem ili pisanjem, operacijski sustav odlučuje koja će se funkcija pozvati iz skupa funkcija upravljačkog programa, odnosno funkcija koje je definirao programer. Nakon što se funkcija izvršila kontrola se ponovno vraća operacijskom sustavu [4].

Upravljački programi imaju pasivnu ulogu i prilikom njihovog učitavanja. Prije učitavanja operacijski sustav detektira hardver, odlučuje koji upravljački program za njega učitati i konfigurira ga kako bi mogao upravljati hardverom od interesa. *Plug-and-play* uređaji sadržavaju elektronički potpis na temelju kojega operacijski sustav ima informaciju o kakvom hardveru se radi. Koncept slojevitosti upravljačkog programa prikazan je na slici 2.4. Lijevi stupac prikazuje stog jezgre operacijskog sustava s *DEVICE_OBJECT* strukturama povezanim prema gore. Sve strukture povezane su s upravljanjem hardvera. Desni stupac prikazuje skup upravljačkih programa koji međudjeluju kako bi hardver koji je njima upravljan ispravno radio. Pored desnog stupca prikazan je tok IRP-a (engl. *Input/Output request packet*) kroz slojevitost upravljačkih programa. IRP je osnovna struktura za komunikaciju prema upravljačkom programu i između dva upravljačka programa [5]. I/O upravitelj zadužen je za generiranje IRP zahtjeva i pozivanje funkcije u upravljačkom programu koja je zadužena za njegovu obradu. Kada je IRP zahtjev riješen, kontrolu ponovo preuzima I/O upravitelj i obavještava inicijatora zahtjeva. Detalji oko načina rada IRP-a dani su u poglavlju 3. U prikazanoj strukturi, svaki WDM upravljački program ima barem dva upravljačka programa:

- Funkcijski
- *Bus driver* [4].

Funkcijski upravljački program podrazumijeva sve funkcionalnosti potrebne za adekvatan rad hardvera poput ulazno/izlaznih operacija i rukovanja prekidima. „Upravljački program sabirnicom“ (engl. *bus driver*) je zadužen za povezivanje između hardvera i računala. Primjerice *bus driver* za PCI sabirnicu je softver koji detektira da je uređaj spojen u utor (engl. *slot*) na računalo i određuje zahtjeve predmetnog uređaja u pogledu povezivanja putem mapirane memorije. Također, to je softver koji je zadužen za isključivanje i uključivanje dotoka struje na konektor uređaja [4].



Sl. 2.4. Slojevitost upravljačkog programa.

Neki uređaji imaju više od dva upravljačka programa ako su pridodani i filter upravljački programi. Često se filter upravljački programi implementiraju kako bi se modificirale određene mogućnosti funkcijskog upravljačkog programa, primjerice za direktno pristupanje PCI *Express* konfiguracijskom prostoru. Svaki od ovih upravljačkih programa povezan je s *DEVICE_OBJECT* strukturom:

- PDO (engl. *physical device object*) – struktura koja predstavlja vezu između spojenog hardvera i računala, koristi ju *bus driver*.
- FDO (engl. *function device object*) – struktura pomoću koje se upravlja mogućnostima uređaja, koristi ju funkcijski upravljački program.
- FiDO (engl. *filter device object*) – struktura u koju se spremaju informacije o hardveru i aktivnostima filtriranja, koristi ju filter upravljački program [4].

2.4. Arhitektura PCI *Express* sabirnice

2.4.1. Usporedba sa starijim sabirnicama

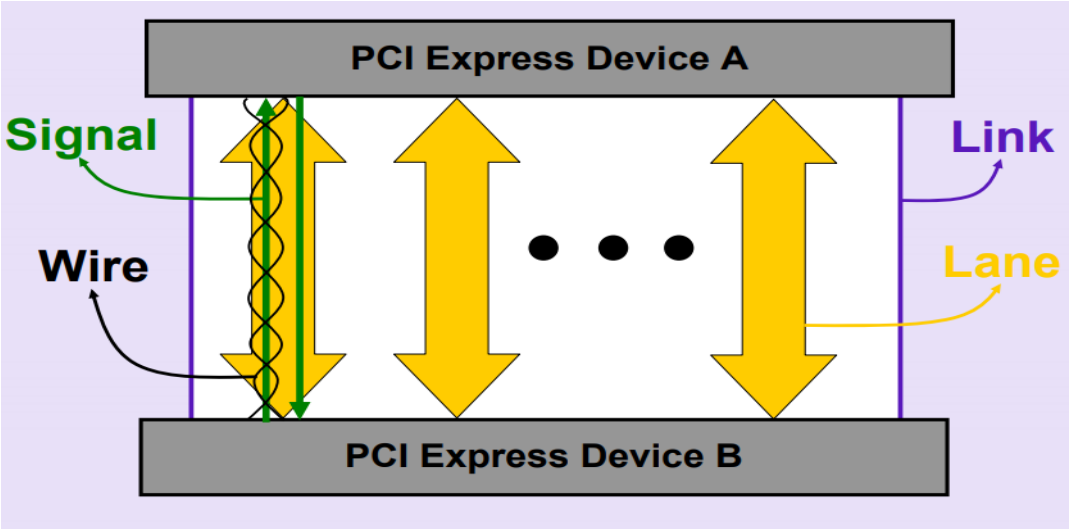
PCI *Express* je standard razvijen za međupovezivanje opće namjene definiran za širok spektar računalnih i komunikacijskih platformi. Predstavlja treću generaciju sabirnica visoke propusnosti za povezivanje vanjskih uređaja. Prva generacija sabirnica uključuje ISA, EISA,

VESA i *Micro-Channel* sabirnice dok druga generacija podrazumijeva AGP, PCI i PCI-X. PCI *Express* arhitektura zadržala je najkorisnije mogućnosti prethodnih generacija arhitektura sabirnica, ali su pridodane i nove značajke omogućene naprecima u razvoju računalne arhitekture. Koristi sličan model čitanja i zapisivanja na tvrdi disk, u memoriju ili u konfiguracijske registre kao i prethodne generacije. Model memorijskog i konfiguracijskog adresnog prostora također je zadržan, što omogućava kompatibilnost sa starijim verzijama softvera pisanim za PCI i PCI-X sustave. Drugim riječima, očuvanjem istog modela adresnog prostora upravljački programi i aplikacije korišteni kod starijih generacija sabirnica će bez modifikacija raditi i za PCI *Express* sustave [6].

2.4.2. Komunikacija PCI *Express* sabirnicom

Kako bi se povećale performanse rada, bilo je potrebno značajno redizajniranje arhitekture sabirnice. PCI i PCI-X su sabirnice koje funkcioniraju paralelno gdje je slučaj da više uređaja dijeli jednu sabirnicu. Prilikom slanja jedan uređaj šalje podatke, a ostali uređaji osluškuju samo one podatke koje namjeravaju primiti (engl. *multi-drop bus*). S druge strane PCI *Express* implementira serijsku direktnu vezu (engl. *point-to-point*) za komunikaciju između dva uređaja. Ako je potrebno povezati više od dva uređaja koristi se prekidač (engl. *switch*). Serijska veza rezultira manjim brojem nožica (engl. *pins*) što dovodi do smanjenja troškova u procesu proizvodnje [6].

Veza se može sastojati od x1, x2 ,x4, x8, x16 ili x32 parova signala u oba smjera. Ovi parovi signala nazivaju se trake (engl. *lanes)* [6]. Struktura sabirnice prikazana je na slici 2.5.



Sl. 2.5. Struktura PCI *Express* sabirnice [7].

Na slici je prikazano povezivanje dva uređaja pomoću PCI *Express* sabirnice. Povezani su pomoću jedne ili više traka. Traka se sastoji od dvaju signala, jedan za primanje, a drugi za slanje podataka. Signal se sastoji od dvije žice kojima se šalju komplementarni signali kako bi se smanjio utjecaj elektromagnetskog šuma [6].

Brzina prijenosa preko sabirnice određena je brojem traka i verzijom sabirnice. Kako se tehnologija razvijala, izdavane su nove verzije, odnosno nove generacije PCI *Express* sabirnice. Primjerice, sabirnica druge generacije za svaku traku ima brzinu prijenosa 5.0 Gbit/s [6].

Kako bi se postigla visoka otpornost na pojavu pogreške, svaki bajt se konvertira u 10-bitni kod na strani transmitera (8b/10b kodiranje). Drugim riječima, svaki informacijski bajt kojega je potrebno prenijeti, prenosi se pomoću 10 bita preko sabirnice. Na ovaj način dobiva se na robusnosti, ali performanse znatno opadaju. Novije generacije PCI *Express* sabirnica imaju manje gubitke performansi zbog korištenja drukčijih, optimiziranih kodiranja (128b/130b). Propusnosti, brzine prijenosa i načini kodiranja za prve četiri generacije PCI *Express* sabirnica navedeni su u tablici 2.1. Bitno je naglasiti kako se sve navedene vrijednosti u tablici odnose na prijenos u jednom smjeru. Prijenosi u oba smjera mogu se odvijati istovremeno prema navedenim brzinama s obzirom da se radi o *dual-simplex* vezi [6].

Tab. 2.1. Propusnosti za različite generacije PCI *Express* sabirnica [8].

PCI *Express* verzija		1.0	2.0	3.0	4.0
Godina		2003.	2007.	2010.	2017.
Linijsko kodiranje		8b/10b	8b/10b	128b/130b	128b/130b
Brzina prijenosa		2.5 GT/s	5.0 GT/s	8.0 GT/s	16.0 GT/s
Propusnost	x1	250 MB/s	500 MB/s	984.6 MB/s	1969 MB/s
	x2	0.5 GB/s	1.0 GB/s	1.97 GB/s	3.94 GB/s
	x4	1.0 GB/s	2.0 GB/s	3.94 GB/s	7.88 GB/s
	x8	2.0 GB/s	4.0 GB/s	7.88 GB/s	15.75 GB/s
	x16	4.0 GB/s	8.0 GB/s	15.8 GB/s	31.5 GB/s

2.4.3. Rukovanje prekidima

Prekid je signal kojeg emitira hardver ili aplikacija prema procesoru kako bi se ukazalo na događaj koji zahtjeva trenutnu pažnju. Kada je procesor obavješten o prekidu, prestaje s izvođenjem koda kojeg je u tom trenutku izvodio i kreće s izvođenjem funkcije za rukovanje

prekidima (engl. *interrupt handling*). Nakon izvođenja ove funkcije procesor se vraća aktivnosti koju je obavljao kada je bio prekinut [3].

Rukovanje prekidima postignuto je pomoću MSI (engl. *Message Signaled Interrupt)* protokola. MSI prekidi koriste isti kanal preko kojeg se prenose informacije poput video podataka (engl. *in-band signaling*). MSI protokol dopušta alociranje više od jednog prekida. Prilikom programiranja uređaja određuje se adresa registra za prekide i vrijednost za svaki od alociranih prekida. Zapisivanjem odgovarajuće vrijednosti na adresu registra za prekide, okida se prekid [6].

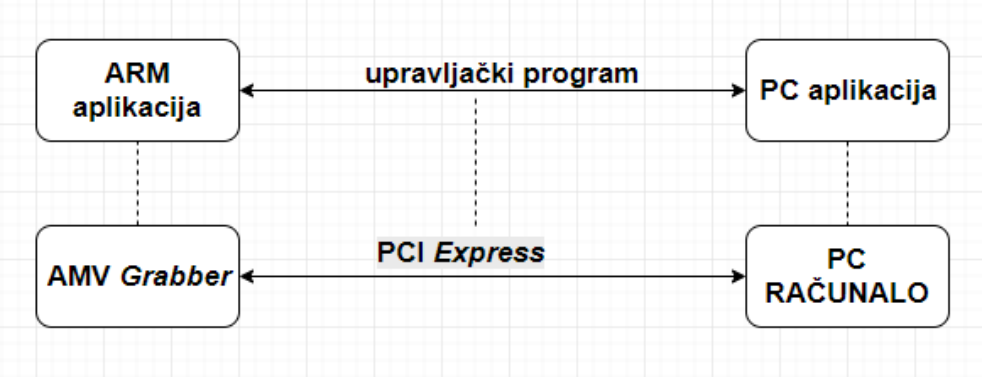
Kada je iz upravljačkog programa potrebno komunicirati prema uređaju, svi potrebni parametri se zapisuju na mapiranu memoriju i inicira se MSI prekid kojega uređaj detektira. Uređaj tada obavlja traženu zadaću, zapisuje potrebne podatke na mapiranu memoriju i inicira prekid prema upravljačkom programu. Kod inicijalizacije upravljačkog programa definirana je povratna (engl. *callback function*) koja se okida na MSI prekid prema upravljačkom programu.

Za prijenos se koristi i DMA (engl. *Direct Memory Access*) princip. DMA omogućava pristup memoriji računala neovisno o centralnom procesoru. Procesor je dugotrajno okupiran operacijama čitanja i pisanja koje su relativno spore. Koristeći DMA, procesor inicira transfer i može se posvetiti drugim operacijama dok je transfer u tijeku. Kada je transfer završen, DMA upravljač (engl. DMA *controller*) inicira prekid prema procesoru i obavještava ga kako je transfer završen [6].

3. RAZVOJ UPRAVLJAČKOG PROGRAMA ZA PCIe sabirnicu

U ovom poglavlju opisana je izrada upravljačkog programa za PCI *Express* sabirnicu za *Windows* operacijski sustav. S obzirom da se radi o *Windows* operacijskom sustavu treba se upoznati s radom *Windows*-a i konceptima na kojima on počiva. S druge strane, bitno je znati tehničke informacije vezane uz hardver za koji se piše upravljački program.

AMV *Grabber* ploča spaja se na računalo preko PCI *Express* sabirnice za koju se razvija upravljački program. Ploča ima ARM procesor koji izvodi aplikaciju zaduženu za komunikaciju s upravljačkim programom. S druge strane PC aplikacija inicira zahtjeve i putem upravljačkog programa koristi funkcionalnosti ploče.



Sl. 3.1. Komunikacija AMV *Grabber* ploče s računalom.

3.1. Inicijalizacija *Windows* upravljačkog programa za PCIe sabirnicu

Glavnu ulogu kod inicijalizacije upravljačkog programa ima PnP (*Plug and Play*) upravitelj operacijskog sustava. Kod spajanja uređaja ispituje se elektronički potpis kako bi se identificirao uređaj. PnP upravitelj koristi identifikator uređaja kako bi locirao njegov hardverski ključ u sistemskom registru (engl. *system registry*). Ako se uređaj prvi puta spaja na računalo njegov hardverski ključ neće biti pronađen. Tada je potrebno specificirati instalacijske upute za uređaj u posebnoj datoteci s ekstenzijom INF. Nakon stvaranja hardverskog ključa PnP upravitelj je svjestan postojanja uređaja i zna koji je upravljački program odgovoran za taj uređaj. Sljedeći put kad se uređaj spoji na računalo, hardverski ključ će biti spremljen i neće se morati ponavljati proces stvaranja hardverskog ključa. Kada je definiran hardverski ključ, PnP upravitelj predaje kontrolu memorijskom upravitelju (engl. *memory manager*) koji poziva *DriverEntry* funkciju.

DriverEntry funkcija svakog upravljačkog programa zadužena je za globalnu inicijalizaciju, odnosno odgovorna je za definiranje pokazivača na funkcije u objektu upravljačkog programa. Ove pokazivače koristi operacijski sustav kako bi znao koju od funkcija pozvati za određenu zadaću. Primjer strukture *DriverEntry* funkcije:

```
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject,
                     PUNICODE_STRING RegistryPath) {
DriverObject->DriverUnload = DriverUnload;
DriverObject->DriverExtension->AddDevice = AddDevice;
DriverObject->MajorFunction[IRP_MJ_PNP] = DispatchPnp;
DriverObject->MajorFunction[IRP_MJ_POWER] = DispatchPower;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = DispatchWmi;
return STATUS_SUCCESS;
}
```

- *DriverUnload* – poziva ju I/O upravitelj neposredno prije brisanja upravljačkog programa iz memorije.
- *MajorFunction* – vektor pokazivača na funkcije definirane u upravljačkom programu, inicijalizira ga I/O upravitelj. Svaki pokazivač određuje koju funkciju pozvati na odgovarajući IRP zahtjev prema upravljačkom programu.
- *DriverStartIo* – definira funkciju koja obavlja ulazno-izlazne zahtjeve, poziva ju I/O upravitelj.
- *DriverExtension->AddDevice* - funkcija koju poziva PnP upravitelj za svaku instancu hardvera za koju je upravljački program odgovoran [4].

Nakon što je *DriverEntry* funkcija završila s izvođenjem i vratila povratnu vrijednost, PnP upravitelj poziva *AddDevice funkciju*. *AddDevice* funkcija u suštini kreira instancu uređaja i povezuje ju na stog uređaja. Koristi dvije funkcije iz biblioteke *wdm.h*: *IoCreateDevice()* i *IoAttachDeviceToDeviceStack()*. Ako je instanca uređaja uspješno kreirana, PnP upravitelj konfigurira hardver. U ovoj fazi inicijalizacije definira se *callback* funkcija za prekide s ploče kao i svi potrebni mehanizmi za mapiranje memorije. Upravljački program dobiva poseban IRP kojim je obavješten o tome koji su mu I/O resursi dodijeljeni. Time je proces inicijalizacije dovršen i hardver je spreman za korištenje [4].

3.2. Povezivanje na AMV *Grabber* i PC aplikaciju

Kako bi se postigla viša razina performansi i širi spektar funkcionalnosti, u programsko rješenje su uz mogućnosti za prijenos video podataka dodane i mogućnosti koje se tiču upravljanja spojenim kamerama. Podrazumijevaju izmjene u postojećem upravljačkom programu, korisničkoj aplikaciji na strani PC računala i ARM aplikaciji na strani ploče. Bitno je reći da je naglasak na cijeloj putanji stavljen na optimiziranost s obzirom da se svi procesi trebaju izvoditi u stvarnom vremenu.

3.2.1. Upravljački program

Upravljački program PCI *Express* sabirnicom nadograđen je u pogledu dodavanja novih naredbi za povezivanje s jedne strane na PC korisničku aplikaciju i s druge strane na ARM aplikaciju na ploči. Za povezivanje PC aplikacije i upravljačkog programa koriste se IOCTL kontrolni kodovi (engl. I/O *Control*) [5]. Kada korisnička aplikacija poziva upravljački program s određenim zahtjevom predaje mu između ostalog i IOCTL kod koji detaljnije određuje zahtjev o kojem se radi. IOCTL kodovi definirani su u posebnom modulu prema sljedećem formatu:

```
#define IOCTL_Device_Function CTL_CODE( DeviceType,
                                        Function,
                                        Method,
                                        Access)
```

- *DeviceType* – tip uređaja, vrijednost mora odgovarati vrijednosti iz strukture *DEVICE_OBJECT*.
- *Function* – govori o kojem se IOCTL kodu radi.
- *Method* – specificira metodu prijenosa podataka između PC aplikacije i upravljačkog programa. Bitno je dobro razumjeti odabranu metodu prijenosa IOCTL kodova jer će se u suprotnom i kod najmanje nepravilnosti prilikom prijenosa pojaviti BSOD (*Blue Screen Of Death*).
- *Access* - razina prava pristupa korisničke aplikacije za čitanje i pisanje prema upravljačkom programu [4].

Kada korisnička aplikacija inicira zahtjev, I/O upravitelj operacijskog sustava pakira zahtjev u IRP sa svim potrebnim varijablama i šalje ga na IRP stog, gdje je dostupan odgovarajućem upravljačkom programu. I/O upravitelj poziva odgovarajuću funkciju definiranu kod inicijalizacije upravljačkog programa, dobiva se IRP, prema IOCTL kontrolnom kodu zna o kojem se zahtjevu radi te na taj način i postupa. Uz kontrolni kod dohvaćaju se i veličine ulaznih i izlaznih spremnika (engl. *buffer*). Ulazni spremnik prenosi podatke od aplikacije do upravljačkog programa, a izlazni obratno.

Nakon što se zahtjev izvrši, funkcija vraća statusnu vrijednost o uspješnosti zahtijevane operacije. U slučaju da zahtjev nije moguće odmah izvršiti, IRP se stavlja na čekanje pomoću funkcije *IoMarkIrpPending()* i potrebno ga je izvršiti u nekoj drugoj funkciji, nakon što je ploča obavila traženo i vratila prekid.

Funkcija koja upravlja zahtjevima iz PC aplikacije ima sljedeći oblik:

```
NTSTATUS DispatchControl(PDEVICE_OBJECT fdo, PIRP Irp) {
PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
NTSTATUS status = STATUS_SUCCESS;
ULONG info = 0;
PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
ULONG cbin = stack->Parameters.DeviceIoControl.InputBufferLength;
ULONG cbout = stack->Parameters.DeviceIoControl.OutputBufferLength;
switch (stack->Parameters.DeviceIoControl.IoControlCode) {
      ...
        default:
            status = STATUS_INVALID_DEVICE_REQUEST;
            break;
      }
return IoCompleteRequest(Irp, IO_NO_INCREMENT);
}
```

DispatchControl() funkcija razvrstava IRP zahtjeve upućene prema upravljačkom programu. Nakon što je dohvatila odgovarajući IRP pomoću *IoGetCurrentIrpStackLocation(),* provjerava o kakvom se zahtjevu radi preko IOCTL kontrolnog koda i poziva odgovarajuću funkciju definiranu u *DriverEntry()* funkciji.

Kada je zahtjev iniciran iz PC aplikacije, potrebno je preko upravljačkog programa dostaviti zahtjev s pripadajućim parametrima do namjenskog uređaja (AMV *Grabber*). Uređaj i

računalo povezani su posredstvom mapirane memorije. Kada ploča inicira prekid prema upravljačkom programu poziva se pripadajuća *callback* funkcija. Struktura takve jednostavne funkcije:

```
BOOLEAN OnInterrupt(PKINTERRUPT InterruptObject, PDEVICE_EXENSION pdx) {
if (<ne radi se o mom uređaju>)
      return FALSE;
IoRequestDpc(pdx->DeviceObject, NULL, pdx);
return TRUE;
}
```

Oba argumenta primljena u *callback* funkciji kreirana su prilikom inicijalizacije. Prvi predstavlja objekt prekida i obično se ne koristi, a drugi je pokazivač na ekstenziju uređaja sa svim pomoćnim varijablama i zastavicama poput informacija o adresama mapirane memorije, brojačima (engl. *timer*), adresama međuspremnika i informacijama koje se tiču DMA prijenosa. *Callback* funkcija poziva DPC (engl. *Deferred Procedure Call*) funkciju u kojoj se zapravo obavlja obrada prekida. Svaka funkcija u upravljačkom programu izvodi se s predodređenom IRQL (engl. *Interrupt request level*) razinom prioriteta. MSI *callback* funkcija izvodi se s visokom razinom prioriteta što zabranjuje ostalim funkcijama upravljačkog programa da ju prekinu u izvođenju. Ovakva funkcija ne smije se izvoditi dugo vremena pa su uvedena određena ograničenja. Glavno ograničenje je nemogućnost izvršavanja/zatvaranja IRP zahtjeva pa se to obavlja u DPC funkciji koja ima nižu IRQL razinu prioriteta. Nakon što se IRP zahtjev zatvori, *Windows* I/O upravitelj obavještava aplikaciju. U slučaju da je u aplikaciji potrebna povratna informacija o statusu obavljene operacije implementira se mogućnost vraćanja vrijednosti o uspješnosti [4].

3.2.2. ARM aplikacija

ARM aplikacija izvodi se na *Zynq* procesoru koji se nalazi na AMV Grabber ploči kako je opisanu u potpoglavlju 2.1. Temelji se na FPGA hardverskom dizajnu napravljenom pomoću *Vivado* garniture. ARM aplikacija izvodi se u kontekstu *FreeRTOS* (engl. *Real Time Operating System*) operacijskog sustava. *FreeRTOS* omogućava korištenje niti, *timer*-a i semafora. Aplikacija je podijeljena na više podsustava među kojima su PCI *Express*, video i I2C (engl. *Inter-Integrated Circuit*) podsustavi. Prilikom pokretanja aplikacije inicijaliziraju se I2C i PCI *Express* podsustav.

Prilikom inicijalizacije PCI *Express* podsustava definiraju se potrebne *callback* funkcije i red (engl. *queue*) za spremanje prekida dobivenih s računala koji je realiziran u obliku statičkog polja. Potrebne *callback* funkcije podrazumijevaju funkciju koja se okida na prekid s računala, funkciju koja se okida na prekid iniciran sa strane hardvera ploče i funkcije za početak i završetak snimanja i slanja video tokova na zahtjev s PC računala. Kreira se nit koja obrađuje sve zadaće koje se tiču PCI *Express* podsustava. Izvedena je na način da čeka određeni zahtjev, postupa sukladno tome zahtjevu i ako je potrebno komunicira s ostalim modulima ARM aplikacije. Kada PC upravljački program zapiše zahtjev sa svim potrebnim parametrima na PCI *Express* memoriju i inicira prekid, okida se *callback* funkcija *pc_interrupt_handler()* koja je zadužena za obradu svih prekida sa strane računala. Funkcija zapisuje vrstu zahtjeva s potrebnim parametrima u vektor *request_queue[]*. Ovaj proces koristi princip semafora jer spomenuti niz koriste i drugi podsustavi ARM aplikacije, a zahtjevi se uvijek zapisuju na prvo slobodno mjesto u nizu. PCI *Express* nit osluškuje niz i kada je novi zahtjev zapisan u njega, nit poziva potrebne module za obradu dobivenog zahtjeva i nakon što je zahtjev obrađen vraća prekid s povratnim vrijednostima o uspješnosti prema računalu.

Inicijalizacijom I2C podsustava omogućeno je sučelje prema registrima potrebnima za korištenje kamera. Video podsustav inicijalizira se kada ARM aplikacija dobije zahtjev za početak snimanja i slanja videa. Zauzima se potrebna memorija ovisno o broju kamera koje se namjerava koristiti, postavljenoj rezoluciji kamere, broju okvira u sekundi (engl. FPS – *Frames per second*) i broju bajtova po elementu slike (engl. *pixel*). Zauzima se dvostruka količina memorije s obzirom da se koriste dvostruki spremnici. Kreira se video nit koja na snimljenih 10 okvira predaje kontrolu PCI *Express* modulu kako bi se video dostavio na računalo.

3.2.3. PC aplikacija

PC aplikacija predstavlja jednostavno sučelje prema korisniku. Olakšava korištenje razvijenih funkcionalnosti. Glavna svrha aplikacije zapisivanje je video sadržaja na tvrdi disk i komunikacija s upravljačkim programom kako bi se iz grafičkog sučelja aplikacije upravljalo mogućnostima na AMV *Grabberu.* Najbitnija funkcija za povezivanje s upravljačkim programom je *DeviceIoControl()*[5]. Koristi se prilikom slanja svakog zahtjeva. Funkcija prima kontrolni IOCTL preko kojeg određuje upravljački program za kojeg je zahtjev namijenjen. Na poziv

DeviceIoControl() funkcije, I/O upravitelj pakira predane argumente u IRP i šalje ga odgovarajućem upravljačkom programu. Prototip *DeviceIoControl()* funkcije:

```
BOOL WINAPI DeviceIoControl(
  HANDLE       hDevice,
  DWORD        dwIoControlCode,
  LPVOID       lpInBuffer,
  DWORD        nInBufferSize,
  LPVOID       lpOutBuffer,
  DWORD        nOutBufferSize,
  LPDWORD      lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

- *hDevice* – instanca uređaja
- *dwIoControlCode* – kontrolni IOCTL kod
- *lpInBuffer* – spremnik za slanje podataka prema upravljačkom programu
- *nInBufferSize* – veličina spremnika
- *lpOutBuffer* – spremnik za primanje podataka iz upravljačkog programa
- *nOutBufferSize* – veličina spremnika
- *lpBytesReturned* – broj bajtova koje upravljački program vraća
- *lpOverlapped* – pokazivač na *overlapped* strukturu podataka koja rukuje asinkronom komunikacijom s upravljačkim programom [4].

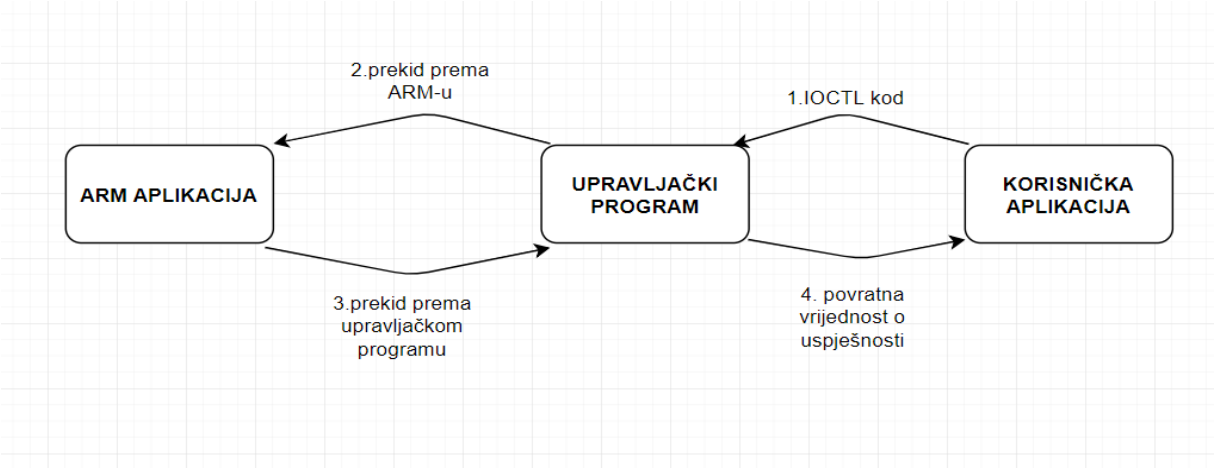
3.2.4. Primjer korištenja

Svi zahtjevi koriste putanju: PC aplikacija – upravljački program – ARM aplikacija – upravljački program – PC aplikacija. Zahtjevi su:

- Inicijalizacija ulaza (engl. *port*) za kameru
- Inicijalizacija kamere sa svim potrebnim instrukcijama
- Obavijest o zadnjoj instrukciji i provjera uspješnosti cjelokupnog procesa inicijalizacije kamere
- Postavljanje video parametara poput broja okvira u sekundi i rezolucije okvira
- Početak video toka (engl. *start streaming*)

- Zaustavljanje video toka (engl. *stop streaming*)
- Zapisivanje paketa video podataka na tvrdi disk.

Ilustracija putanje koju prolazi svaki od ovih zahtjeva prikazana je na slici 3.2.



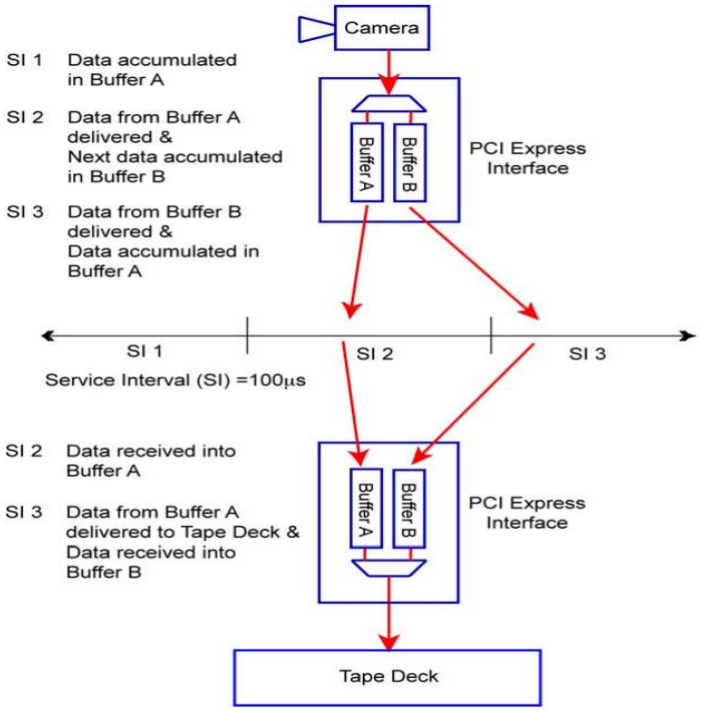
Sl. 3.2. Dijagram toka korištenja upravljačkog programa.

Kod inicijalizacije *port*-a potrebno je uključiti i konfigurirati sklopove za deserijalizaciju (engl. *deserializers*). Sklopovi za deserijalizaciju primaju video signal s kamera koji je serijski zapakiran i paraleliziraju ga kako bi bio podešen za 12-bitnu sabirnicu kojom dolazi do procesora. Svaka kamera ima svoj sklop za deserijalizaciju.

Za inicijalizaciju kamere putanjom prikazanom na slici 3.2. prolazi se onoliko puta koliko ima instrukcija za konfiguraciju kamere (~1000). Instrukcije su zapravo skup od oko 1000 parova adresa registara i vrijednosti koje treba zapisati na te adrese kako bi kamera bila konfigurirana na željeni način. Nakon svakog prolaska putanje PC aplikacija ima povratnu informaciju o uspješnosti tražene operacije. Ako jedna od instrukcija nije uspješno izvršena, odnosno jedna od poslanih vrijednosti nije uspješno zapisana u registre kamere, prestaje se s procesom konfiguracije kamere. Ovakav princip korišten je i kod provjere povezanosti kamere s pločom. Ako kamera nije spojena, za očekivati je da se prva instrukcija iz skupa instrukcija neće uspjeti zapisati u registar.

Postoji i drugi princip konfiguracije kamera. Moguće je instrukcije za konfiguraciju kamere spremiti na uređaju. U tom slučaju postojala bi zadana (engl. *default*) konfiguracija i trebalo bi implementirati mehanizme za izmjenu u slučaju potrebe za drugačijom konfiguracijom. Zbog toga je odabran drukčiji pristup kojim je konfiguracijska datoteka smještena na PC računalu i kod potrebe konfiguracije određene kamere šalje se instrukcija po instrukcija.

Potonji pristup omogućava lakše testiranje i veći broj konfiguracijskih datoteka. Korisnik je u mogućnosti konfigurirati kameru na željeni način, u pogledu rezolucije i broja okvira u sekundi. Zbog dozvoljavanja različitih rezolucija videa potrebno je voditi računa i o alociranoj memoriji. Primjerice, video koji se snima u rezoluciji 1280x1080 zahtijevat će više alocirane memorije od videa koji se snima u rezoluciji 1280x720. Prema tome, treba voditi računa o memoriji na AMV *Grabberu* i u PC aplikaciji gdje se video podaci zapisuju iz RAM memorije na tvrdi disk. ARM aplikacija koristi dvostruke spremnike (engl. *double buffering*). Koncept dvostrukih spremnika detaljnije je prikazan na slici 3.3.



Sl. 3.3. Koncept dvostrukih spremnika [6].

Kamera snima video i sprema ga u prvi spremnik. Nakon što je 10 okvira videa spremljeno u prvi spremnik, kamera nastavlja snimati, ali video podatke sprema u drugi spremnik, a podaci iz prvog spremnika šalju se dalje u putanji prema PC-u. Kada je drugi spremnik pun, kamera ponovno počinje zapisivanje u prvi spremnik i tako se započinje novi ciklus. Kod ovakvog koncepta potrebno je imati mehanizme sinkronizacije kako bi se pravovremeno signalizirao početak i kraj spremanja u određeni spremnik.

Kada se otvori grafičko sučelje, korisnik prvo odabire konfiguracijsku datoteku za kamere. Postoje tri različite konfiguracijske datoteke za tri postojeće rezolucije. Slijedi odabiranje jednog od devet *port*-ova na koji je spojena željena kamera. Kada je odabran *port*, kreće proces

konfiguriranja kamere. Prvo se šalje naredba za konfiguraciju i uključivanje sklopova za deserijalizaciju što podrazumijeva dovođenje napajanja na željeni *port*. Zatim kreće slanje instrukcija za konfiguraciju kamere. Nakon što je zadnja instrukcija poslana, šalje se obavijest da su sve instrukcije poslane. Ako su sve instrukcije uspješno zapisane na pripadajuću adresu, mijenja se vrijednost statusa konfiguriranosti kamere i kamera je spremna za snimanje. Opisani proces ponavlja se onoliko puta koliko je kamera potrebno koristiti. Kada su sve željene kamere konfigurirane šalju se video parametri potrebni za alociranje video spremnika. Video parametri podrazumijevaju rezoluciju kamere, broj bajtova po elementu slike i broj okvira u sekundi. Tada se šalje zahtjev za početak slanja video toka prilikom kojeg se inicijalizira video podsustav u ARM aplikaciji. Prije samog slanja videa, ARM aplikacija preuzima S/G (engl. *Scatter/Gather*) listu od upravljačkog programa. S/G lista je zapravo lista adresa na koje je potrebno zapisati video. Kada ima potrebne adrese, ARM aplikacija može početi sa slanjem videa.

4. TESTIRANJE

Za testiranje funkcionalnosti u programskom kodu upravljačkog programa korišten je *DebugView* alat. *DebugView* je sučelje za ispis formatiranih poruka iz funkcija upravljačkog programa koje se izvode u jezgrinom načinu rada [4].

Testiranje kašnjenja prilikom konfiguracije jedne kamere izvodilo se na različitim *port*-ovima s konfiguracijskim datotekama za različite rezolucije. Rezolucija ne utječe na kašnjenje prilikom konfiguracije kamere s obzirom da je isti broj instrukcija za svaku od tri podržane rezolucije. Kamera je konfigurirana tako da snima 30 okvira u sekundi. U svakom testu ispravno je inicijalizirana ciljana kamera. Rezultati mjerenja kašnjenja kod konfiguracije jedne kamere pomoću razvijenog upravljačkog programa dani su u tablici 4.1.

Tab. 4.1. Testiranje kašnjenja kod konfiguracije kamere.

Broj testa	Broj instrukcija	Kašnjenje (ms)
1	1176	1177
2	1176	1177
3	1176	1181
4	1176	1176
5	1176	1177
6	1176	1178
7	1176	1185
8	1176	1178
9	1176	1178
10	1176	1178
11	1176	1179
12	1176	1186
13	1176	1177
14	1176	1177
15	1176	1182
16	1176	1177
17	1176	1189
18	1176	1184
19	1176	1177

20	1176	1177

Tab. 4.2. Statističke vrijednosti mjerenja.

Srednja vrijednost kašnjenja (ms)	Maksimalno kašnjenje (ms)	Minimalno kašnjenje (ms)	Standardna devijacija
1179.5	1189	1176	3.7205

Prosjek kašnjenja za ovih 20 testiranja je 1179.5 milisekundi. Slanje jedne po jedne instrukcije kod konfiguriranja kamere ne utječe znatno na performanse jer je za jedan prolazak putanje PC aplikacija – upravljački program – ARM aplikacija – upravljački program – PC aplikacija potrebna približno jedna milisekunda. Prema tome proces konfiguracije jedne kamere traje u prosjeku 1.1795 sekundi. Niska vrijednost standardne devijacije ukazuje na mala odstupanja.

U tablicama 4.3., 4.4., 4.5. i 4.6. dana su testiranja zapisivanja video sadržaja za slučajeve kada su spojene 1, 2, 3 ili 4 kamere. Testirano je zapisivanje videa na tvrdi disk za različite vremenske periode. Za svaki od ovih slučajeva zabilježena je veličina videa zapisanog na tvrdi disk u kilobajtima, broj primljenih okvira, broj izgubljenih okvira, brzina zapisivanja na tvrdi disk i omjer izgubljenih i poslanih okvira.

Tab. 4.3. Testiranje zapisivanja s jedne kamere.

Vrijeme snimanja (sekunde)	Veličina videa (KB)	Broj primljenih okvira	Broj izgubljenih okvira	Brzina zapisivanja (MB/s)	Omjer izgubljenih i poslanih okvira
30	2.322.000	860	40	75.59	4.44%
60	4.455.000	1650	150	72.51	8.33%
120	8.478.000	3140	460	68.99	14.65%

Tab. 4.4. Testiranje zapisivanja s dvije kamere.

Vrijeme snimanja (sekunde)	Veličina videa (KB)	Broj primljenih okvira	Broj izgubljenih okvira	Brzina zapisivanja (MB/s)	Omjer izgubljenih i poslanih okvira
30	4.266.000	1580	220	138.87	12.22%
60	7.776.000	2880	720	126.57	20%
120	15.174.000	5620	1580	123.47	21.94%

Tab. 4.5. Testiranje zapisivanja s tri kamere.

Vrijeme snimanja (sekunde)	Veličina videa (KB)	Broj primljenih okvira	Broj izgubljenih okvira	Brzina zapisivanja (MB/s)	Omjer izgubljenih i poslanih okvira
30	4.698.000	1740	1110	152.93	41.11%
60	9.072.000	3360	2130	147.66	39.44%
120	16.767.000	6210	4590	136.45	42.5%

Tab. 4.6. Testiranje zapisivanja s četiri kamere.

Vrijeme snimanja (sekunde)	Veličina videa (KB)	Broj primljenih okvira	Broj izgubljenih okvira	Brzina zapisivanja (MB/s)	Omjer izgubljenih i poslanih okvira
30	4.860.000	1800	1800	158.20	50%
60	9.504.000	3520	3680	154.67	51.11%
120	18.576.000	6880	7520	151.17	52.22%

Testiranje se radilo za prijenos videa s jedne i više kamera. Tablice prikazuju koliko je video sadržaja zapisano na tvrdi disk. Javlja se problem odbacivanja okvira ako tvrdi disk ne stigne zapisati kompletan video. Zapisuje se 10 okvira odjednom za svaku kameru. Kod testiranja se koristila maksimalna rezolucija videa 1280x1080 s 30 okvira u sekundi. Korišten je Toshiba HDWD120 tvrdi disk za kojega je maksimalna teorijska brzina zapisivanja 194 MB/s [9]. Maksimalna dobivena brzina prijenosa je 158.20 MB/s. Iz rezultata se može zaključiti da se

pokušavanjem prijenosa većih količina video podataka gubi određeni broj okvira ovisno o trajanju prijenosa videa i broju kamera s kojih se video želi prenijeti. Velika količina paketa se gubi kada se brzina prijenosa približi maksimalnoj brzini zapisivanja na tvrdi disk. Video se zapisuje u sekvencama po 10 okvira pa su broj zapisanih i broj izgubljenih okvira uvijek višekratnici broja 10.

Testiranjem je otkriveno kako je usko grlo (engl. *bottleneck*) cijele putanje od kamere do PC aplikacije proces zapisivanja na tvrdi disk. Brzina zapisivanja na tvrdi disk znatno je sporija od ostalih procesa u prijenosu duž cijele putanje. Ovisi o korištenju predmemorije (engl. *cache*) što znači da bi se performanse smanjile kod dugotrajnijih prijenosa. Korištenjem SSD-a (engl. *solid-state-drive*) teorijski bi se moglo prenijeti sadržaj s 5 kamera, a korištenjem PCI *Express* SSD-a i sadržaj sa svih 9 kamera.

5. ZAKLJUČAK

Trendovi u automobilskoj industriji pokazuju napretke u razvoju autonomnosti vozila. Sve je više algoritama koji pomažu vozaču u vožnji i osiguravaju sve veću autonomnost vozila. Testiranje ovih algoritama u stvarnim uvjetima izuzetno je nepraktično zbog visoke cijene i rizika u prometu. Stoga se razvijaju sustavi koji omogućuju testiranje algoritama u laboratorijima. Jedan od takvih sustava je AMV *Grabber* koji omogućuje prijenos velikih količina video podataka putem PCI *Express* sabirnice. Na njega se spajaju kamere koje snimaju video i šalju ga preko sabirnice na PC računalo. U okviru ovog rada izrađen je upravljački program za PCI *Express* sabirnicu za *Windows* operacijski sustav. Upravljački program inicijalizira sve potrebne mehanizme za povezivanje s korištenim uređajem. Funkcionalnostima upravljačkog programa upravlja se iz korisničke PC aplikacije kako bi se sakrila kompleksnost upravljačkog programa i korisniku približile mogućnosti koje podržava. Glavne mogućnosti su konfiguriranje kamera na željenu rezoluciju i prijenos video sadržaja s više kamera. Prilikom testiranja uočeno je kako postoje određena ograničenja vezana uz kašnjenje prilikom konfiguracije kamere i prijenos video sadržaja. Kašnjenje kod konfiguracije jedne kamere iznosi oko 1.2 sekunde. Usko grlo cijelog procesa snimanja, slanja i spremanja video podataka je zapisivanje na tvrdi disk gdje je maksimalna postignuta brzina zapisivanja 158 MB/s. Performanse bi se mogle povećati korištenjem SSD-a ili PCI *Express* SSD-a koji imaju znatno veću brzinu zapisivanja.

LITERATURA

[1] M. Krbanjević, I. Rešetar, V. Škobić, „*A platform for Hardware In the Loop Testing of Multicamera Automotive Machine Vision Systems*“, Etran, Kladovo, srpanj, 2017.

[2] „*RDACM24B Automotive Camera Platform Technical Datasheet“*, IMI D&D, 2015.

[3] M. Russinovich, D. A. Solomon, A. Ionescu, „*Windows Internals Sixth Edition Part 1*“, Microsoft Press, Washington, 2012.

[4] W. Oney, „*Programming the Microdoft Windows Driver Model – Second Edition*“, Microsoft Press, Washington, 2003.

[5] E. Perla, M. Oldani, „*A Guide to Kernel Exploatation: Attacking the Core*“, Elsevier, Burlington, 2010.

[6] R. Budrok, D. Anderson, T. Shanley, „*PCI Express System Architecture*“, MindShare, Colorado Springs, 2008.

[7] R. Budrok, „*PCI Express Basics*“, MindShare, Santa Carla, 2007.

[8] C. Castellanos, „*PCI-SIG Press Release*“, Beaverton, studeni, 2011.

[9] Toshiba Electronic Devices & Storage Corporation, dostupno na: https://toshiba.semicon-storage.com/us/product/storage-products.html [21.9.2018.]

SAŽETAK

Cilj ovog diplomskog rada bio je razviti funkcionalan upravljački program za PCI *Express* sabirnicu sa strane *Windows* operacijskog sustava. Za razvoj upravljačkog programa potrebno je znati tehničke informacije o hardveru kao i koncepte koje koristi operacijski sustav. Potrebno je omogućiti inicijalizaciju upravljačkog programa i funkcionalnosti za povezivanje s uređajem i s korisničkom aplikacijom. Upravljački program razvijao se za uređaj AMV *Grabber*. Za uspješno korištenje funkcionalnosti se prilikom inicijalizacije upravljačkog programa definiraju mehanizmi od kojih je najbitniji rukovanje prekidima. Funkcionalnosti podrazumijevaju prijenos video sadržaja i proces konfiguracije kamera koje su spojene na uređaj. Proces konfiguriranja kamera minimalno utječe na performanse, za konfiguriranje jedne kamere potrebno je manje od 1.2 sekunde. Usko grlo cijele putanje prijenosa video podataka je proces zapisivanja na tvrdi disk koji je znatno sporiji od ostalih procesa.

Ključne riječi: PCI *Express, Windows* upravljački program, AMV *Grabber*.

ABSTRACT

The aim of this graduate thesis was to develop a functional driver for the PCI Express bus for Windows operating system. For driver development, it is essential to know the technical information about the hardware as well as the concepts used by the operating system. It is necessary to enable the initialization of the driver and the functionalities in order to connect to the device and to the user application. The driver was developed for AMV Grabber. During the initialization of the driver, for successful use of the functionalities, mechanisms are defined, most importantly interrupt handling. Functionalities include video content transfer and process of configuring cameras that are connected to the device. The camera configuration process has a minimal impact on performance, to configure one camera it takes less than 1.2 seconds. The bottleneck of the entire video data transmission path is the process of writing on the hard drive that is considerably slower than the other processes.

Keywords: PCI Express, Windows driver, AMV Grabber.

ŽIVOTOPIS

Bože Eugen Marković rođen je 20. lipnja 1994. u Osijeku. U Osijeku je završio osnovnu školu te je nakon toga upisao Prirodoslovno-matematičku gimnaziju. Završetkom gimnazije upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, smjer računarstvo. Nakon završenog preddiplomskog studija upisuje i diplomski studij na istom fakultetu, smjer programsko inženjerstvo. Dvije godine dobitnik je stipendije Instituta RT-RK u Osijeku. Od stranih jezika služi se engleskim i njemačkim.