



УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ДИПЛОМСКИ – МАСТЕР РАД

Кандидат: Бранислав Кордић

Број индекса: Е2 10/2012

Тема рада: Паралелна обрада видео сигнала у више канала заснована
на архитектури са више језгара

Ментор рада: проф. др Мирослав Поповић

Нови Сад, Јул 2013.



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Дипломски – мастер рад
Аутор, АУ :	Бранислав Кордић
Ментор, МН :	др Мирослав Поповић, ред. проф.
Наслов рада, НР :	Паралелна обрада видео сигнала у више канала заснована на архитектури са више језгара
Језик публикације, ЈП :	Српски / латиница
Језик извода, ЈИ :	Српски
Земља публиковања, ЗП :	Република Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2013
Издавач, ИЗ :	Ауторски репринт
Место и адреса, МА :	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, ФО : (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/47/10/4/11/3/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	дигитална обрада сигнала, наменски системи, архитектура са више језгара, паралелна обрада
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	У овом раду дато је једно решење система за паралелну обраду видео сигнала у више канала које је засновано на ДСП архитектури са више језгара и употреби паралелних програмских блокова. Посебна пажња посвећена је анализи скалабилности реализованог система и оствареног убрзање са становишта повећања броја језгара која учествују у обради. Такође, с обзиром на специфичну архитектуру меморије, анализиран је и утицај пинг-понг механизма. Добијени резултати показују да је употреба ДСП са више језгара у сврху имплементације и постизања бољих перформанси циљног наменског система оправдана.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: др Растислав Струхарик, доц.
	Члан: др Илија Башичевић, доц.
	Члан, ментор: др Мирослав Поповић, ред. проф.
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	Monographic publication		
Type of record, TR :	Textual printed material		
Contents code, CC :	Master Thesis		
Author, AU :	Branislav Kordić		
Mentor, MN :	prof. Miroslav Popović, PhD		
Title, TI :	Parallel Processing of Multichannel Video Based on Multicore Architecture		
Language of text, LT :	Serbian		
Language of abstract, LA :	Serbian		
Country of publication, CP :	Republic of Serbia		
Locality of publication, LP :	Vojvodina		
Publication year, PY :	2013		
Publisher, PB :	Author's reprint		
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/47/10/4/11/3/0		
Scientific field, SF :	Electrical Engineering		
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems		
Subject/Key words, S/KW :	digital signal processing, embedded systems, multicore architecture, parallel processing		
UC			
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia		
Note, N :			
Abstract, AB :	This paper presents an implementation of parallel processing system for multichannel video based on a DSP multicore architecture using different building blocks. Through the implementation of system we analyzed system scalability and processing speedup in terms of increasing the number of cores that participate in the parallel processing. Also, due to the specific memory architecture, the influence of a ping-pong mechanism has been analyzed. Based on these results, the use of multicore architecture in order to implement parallel processing and achieve significantly better performances of target system is justified.		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	prof. Rastislav Struharik, PhD	
	Member:	prof. Ilija Bašičević, PhD	Menthor's sign
	Member, Mentor:	prof. Miroslav Popović, PhD	

Zahvalnost

Najiskrenije se zahvaljujem svom mentoru prof. Miroslavu Popoviću na razumevanju, savetovanju i stručnoj pomoći tokom izrade diplomskog *master* rada. Takođe se zahvaljujem svom kolegi Vladimiru Marinkoviću na pomoći i podršci koju mi je pružio u toku izrade rada.

SADRŽAJ

1. Uvod.....	1
2. Teorijske osnove.....	3
2.1 Texas Instruments TMS320C6678 DSP.....	3
2.2 Advantech Lightning PCIe ploča	5
2.3 SYS/BIOS RTOS	6
2.3.1 Podrška za konkurentno izvršavanje niti.....	7
2.3.2 Podrška za upravljanje memorijom.....	7
3. Koncept rešenja	8
3.1 Arhitektura PQM sistema	8
3.2 Upravljački podsistem	9
3.2.1 Upravljački program	10
3.2.2 Poštansko sanduče.....	10
3.2.3 Memorijski rukovalac	11
3.2.4 PCIE rukovalac	11
3.2.5 Raspoređivač.....	12
3.3 Podsistem za paralelnu obradu digitalnog signala	14
3.3.1 SYS/BIOS.....	14
3.3.2 Poštansko sanduče.....	15
3.3.3 Program jezgra.....	15
3.3.4 Algoritam obrade	16
3.3.4.1 Algoritam za otkrivanje greške crne slike (BSD algoritam)	16
3.4 Funkcionisanje PQM sistema.....	18
3.4.1 Funkcionisanje PQM sistema bez <i>ping-pong</i> mehanizma	18

3.4.2	Osnove <i>ping-pong</i> mehanizma	19
3.4.3	Funkcionisanje PQM sistema sa <i>ping-pong</i> mehanizmom	21
3.4.4	Primer razmene podataka (međusobne komunikacije)	22
4.	Programsko rešenje.....	24
4.1	Modul MailBox.....	25
4.1.1	Funkcija mailBoxCreate	25
4.1.2	Funkcija mailBoxDelete.....	26
4.1.3	Funkcija mailBoxQuery	26
4.1.4	Funkcija mailBoxRead	26
4.1.5	Funkcija mailBoxWrite	27
4.2	Modul PCIeDriver	27
4.2.1	Funkcija pcieDriverOpen	27
4.2.2	Funkcija pcieDriverClose	27
4.2.3	Funkcija pcieDriverDspWrite.....	28
4.2.4	Funkcija pcieDriverDspRead.....	28
4.2.5	Funkcija pcieDriverDmaWriteStart	28
4.2.6	Funkcija pcieDriverDmaReadStart	29
4.2.7	Funkcija pcieDriverDmaCheck	29
4.3	Modul MemoryDriver	29
4.3.1	Funkcija MemoryDriverOpen	29
4.3.2	Funkcija MemoryDriverClose	30
4.3.3	Funkcija MemoryDriverAllocate.....	30
4.3.4	Funkcija MemoryDriverFree	30
4.4	Modul TaskScheduler.....	31
4.4.1	Funkcija startTaskScheduler.....	31
4.4.2	Struktura TaskSchedulerConfiguration.....	31
4.4.2.1	Polje dspChipNum	31
4.4.2.2	Polje coreNumPerDspChip	31
4.4.2.3	Polje schedulingAlgorithm.....	32
4.4.2.4	Polje taskSchedulerThreadPriority	32
4.4.2.5	Polje coreControllerThreadConfigs	32
4.5	Modul CoreController	32
4.5.1	Funkcija coreControllerFunction.....	33
4.5.2	Funkcija initCoreController.....	33
4.5.3	Funkcija preprocessImageFrame	33

4.5.4	Funkcija sendDataToDspThroughDma.....	33
4.5.5	Funkcija getDataThroughDma	34
4.6	Modul CoreApplication.....	34
4.6.1	Funkcija coreApplicationProcessing.....	34
4.6.2	Funkcija MemoryDriverFree	34
4.7	Modul ProcessingAlgorithm.....	35
4.7.1	Funkcija blackScreenReserveMemory.....	35
4.7.2	Funkcija blackScreenProcessingInit	35
4.7.3	Funkcija blackScreenIteration	35
4.7.4	Funkcija blackScreenGetMax.....	36
5.	Testiranje.....	37
6.	Rezultati	39
6.1	Opis merenja	39
6.2	Dobijeni rezultati merenja	40
6.2.1	Rezultati vremena obrade	40
6.2.2	Rezultati čekanja jezgara na nove podatke.....	42
6.2.3	Rezultati ostvarenog ubrzanja.....	43
7.	Zaključak.....	45
8.	Literatura	47

SPISAK SLIKA

Slika 2.1 – Najvažnije komponente fizičke arhitekture TI C6678 DSP	4
Slika 2.2 – Izgled Advantech Lightning ploče sa označenim najvažnijim komponentama ..	5
Slika 3.1 – Arhitektura PQM sistema	9
Slika 3.2 – Struktura poštanskog sandučeta	10
Slika 3.3 – Preslikavanje zadataka obrade na odgovarajuća jezgra DSP	12
Slika 3.4 – Proces raspoređivanja zadataka i prihvatanja rezultata obrade	13
Slika 3.5 – Struktura modula programa jezgra sa jednim i više algoritma obrade.....	16
Slika 3.6 – Blok dijagram BSD algoritma	17
Slika 3.7 – Ponašanje PQM sistema bez implementiranog <i>ping-pong</i> mehanizma	19
Slika 3.8 – Primer <i>ping-pong</i> prenosa u memorijski distribuiranom sistemu.....	21
Slika 3.9 – Ponašanje PQM sistema sa implemntiranim <i>ping-pong</i> mehanizmom.....	23
Slika 6.1 – Grafik rezultata vremena paralelne obrade.....	41
Slika 6.2 – Grafik rezultata srednjeg vremena čekanja jezgara na novi okvir	43
Slika 6.3 – Grafik rezultata ostvarenog ubrzanja	44

SPISAK TABELA

Tabela 4.1 – Spisak modula programskog rešenja	25
Tabela 6.1 – Rezultati vremena paralelne obrade.....	40
Tabela 6.2 – Rezultati srednjeg vremena čekanja jezgara na novi okvir.....	42
Tabela 6.3 – Rezultati ostvarenog ubrzanja	43

SKRAĆENICE

PQM	- <i>Picture Quality Meter</i> , Sistem za procenu kvaliteta slike
DSP	- <i>Digital Signal Processor</i> , Procesor za digitalnu obradu signala
PP	- <i>Ping-Pong Mechanism</i> , Ping-Pong mehanizam
BBT	- <i>Black Box Testing</i> , Testiranje zasnovano na modelu crne kutije
FPGA	- <i>Field Programming Gate Array</i> , Programabilne sekvencijalne mreže
BSD	- <i>Black Screen Detection</i> , Algoritam za otkrivanje crne slike
DMA	- <i>Direct Memory Access</i> , Mehanizam direktnog pristupa memoriji
API	- <i>Application Interface</i> , Programska sprega
GCC	- <i>GNU C Compiler</i> , GNU C prevodilac
OS	- <i>Operating System</i> , Operativni sistem
RTOS	- <i>Real Time Operating System</i> , Operativni sistem za rad u realnom vremenu
STB	- <i>Set-Top Box</i> , Digitalni prijemnik
TV	- <i>Television</i> , Televizija
PCIe	- <i>PCI Express</i> , Brza magistrala za povezivanje periferija
HD	- <i>High Definition Picture</i> , Slika visoke rezolucije
FFT	- <i>Fast Fourier Transform</i> , Brza Furijeova transformacija
IFFT	- <i>Inverse Fast Fourier Transform</i> , Inverzna brza Furijeova transformacija
DCT	- <i>Discrete Cosine Transform</i> , Diskretna kosinusna transformacija
PC	- <i>Personal Computer</i> , Računar

1. Uvod

U ovom radu predstavljeno je jedno od rešenja sistema za paralelnu obradu video signala u više kanala zasnovanog na fizičkoj arhitekturi sa više jezgara. Sistem za paralelnu obradu video signala predstavlja sistem za procenu kvaliteta video signala (eng. *Picture Quality Meter, PQM*) koji je razvijen na institutu RT-RK u Novom Sadu. PQM sistem je deo BBT (eng. *Black Box Testing, BBT*) sistema koji se koristi za automatsko funkcionalno testiranje digitalnih TV prijemnika (eng. *Set-Top Box, STB*) i digitalnih TV uređaja [1], koji je takođe razvijen na institutu RT-RK.

PQM sistem se sastoji od upravljačkog podsistema i podsistema za paralelnu obradu digitalnog signala. Osnovni zadatak upravljačkog podsistema je da omogući upravljanje podsistemom za paralelnu obradu u cilju ostvarivanja paralelne obrade više video kanala. Podsistem za paralelnu obradu čine procesori za digitalnu obradu signala (eng. *Digital Signal Processor, DSP*) sa više jezgara. Podsistem za paralelnu obradu služi za implemenaciju različitih algoritama koji omogućuju procenu kvaliteta video signala. U predstavljenom rešenju PQM sistema na jezgrima podsistema za paralelnu obradu implementiran je algoritam za otkrivanje crne slike (eng. *Black Screen Detection, BSD*) koja se u digitalnim STB i TV uređajima najčešće pojavljuje kao posledica neispravnosti uređaja ili neispravnosti nekog drugog dela u sistemu koji je zadužen za obradu i prenos digitalnog video signala.

Kao posledica distribuirane memorijske arhitekture koju poseduje PQM sistem, u ovom radu analiziran je i uticaj *ping-pong* (PP) mehanizma na ostvareno ubrzanje rada sistema (eng. *speed up*) u kontekstu povećanja broja jezgara koja učestvuju u paralelnoj obradi.

U drugom poglavlju opisana je fizička arhitektura DSP sa više jezgara i fizička arhitektura PCIe ploče (eng. *PCI Express board, PCIe*) koja predstavlja fizičku okosnicu podsistema za

paralelnu obradu video signala. Pored DSP i PCIE ploče, opisan je i SYS/BIOS operativni sistem koji čini osnovu programske podrške dostupne na jezgrima DSP.

U trećem poglavlju dat je koncept rešenja PQM sistema. Detaljno je opisana arhitektura i ponašanje svih gradivnih blokova PQM sistema. Takođe, analiziran je i PP mehanizam.

U četvrtom poglavlju dat je pregled svih programskih modula rešenja PQM sistema zajedno sa opisom najvažnijih funkcija aplikativne sprege (eng. *Application Interface, API*).

U petom poglavlju opisan je način provere ispravnosti rada i verifikacije implementiranog programskog rešenja.

U poslednjem poglavlju predstavljeni su rezultati koji se odnose na vremena obrade referentne količine podataka, vremena koja jezgra provedu u čekanju na nove podatke i ostvareno ubrzanje u zavisnosti od broja jezgara koji učestvuju u obradi. Svi rezultati su analizirani u slučaju kada u PQM sistemu nije, odnosno kada jeste, implementiran PP mehanizam.

2. Teorijske osnove

U ovom poglavlju analizirani su neke od programskih i fizičkih komponenti koje se koriste u implementaciji PQM sistema. Od fizičkih komponenti PQM sistema opisana je arhitektura Texas Instruments TMS320C6678 DSP i arhitektura Advantech DSPC-8681 PCIe ploče. Zbog složenosti pomenutih fizičkih arhitektura u nastavku su ukratko opisane komponente koji su od interesa za PQM sistem. Od programskih komponenti opisan je operativni sistem (eng. *Operating System*, OS) SYS/BIOS. Kao i u slučaju opisa fizičkih arhitektura, opisani su samo delovi SYS/BIOS koji su od važnosti za shvatanje rada PQM sistema.

2.1 Texas Instruments TMS320C6678 DSP

Osnovu obrade digitalnog signala u PQM sistemu čini Texas Instruments (TI) TMS320C6678 (C6678) DSP. C6678 je novi DSP visokih performansi koji se zasniva na novoj TI KeyStone arhitekturi sa više jezgara. On omogućava podršku rada u aritmetici sa fiksnim i pokretnim zarezom. U C6678 DSP pored osnovnog skupa periferija ugrađeno je 8 C66 jezgara (eng. *C66 corepac*). Svako od jezgara radi na frekvenciji od 1.25 GHz. TI C6678 DSP omogućuje kumulativno 10 GHz za obradu, uz mogućnost podešavanja radne frekvencije svakog jezgra u cilju postizanja efikasnije potrošnje energije (eng. *power-efficient control*). C6678 DSP je u potpunosti usklađen (kompatibilan) sa svim DSP C6000 serije [2].

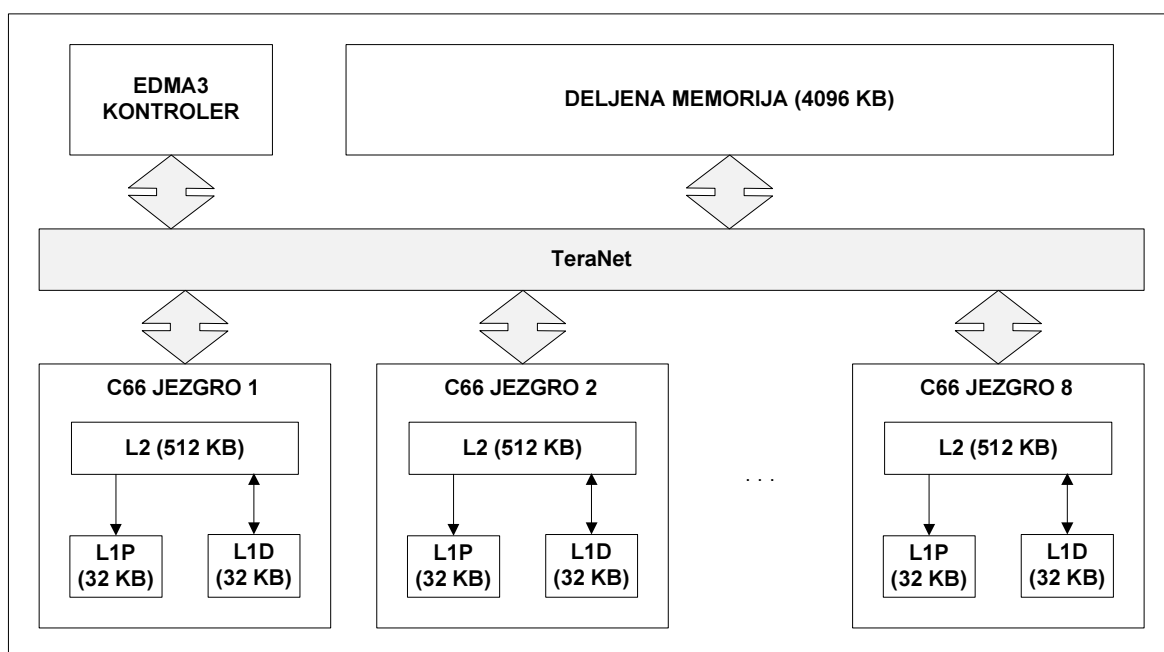
TI KeyStone arhitektura predstavlja nove fizičke programabilne komponente u svrhu lakšeg upravljanja jezgrima, memorijom, periferijama i podsistemima za ubrzanje izvršenja (eng. *accelerators*). Takođe uvodi i nove namenske podsisteme koji obezbeđuju efikasniji način komunikacije između jezgara i dostupnih periferija. Osnovu ove arhitekture čine fizičke komponente kao što su Multicore Navigator, TeraNet i kontroler za deljenje memorije. Multicore

Navigator služi za upravljanje i rukovanje podacima između različitih komponenti DSP. TeraNet je nova magistrala koja omogućava neblokirajuću razmenu podataka visoke propusnosti. Kontroler za deljenje memorije obezbeđuje istovremeni pristup više različitih delova arhitekture deljenoj memoriji u svrhu čitanja ili pisanja, uz mogućnost fizičke zaštite pristupa određenim memorijskim adresama.

TI C6678 DSP omogućava zavidnu količinu memorije koja je dostupna unutar samog DSP (L2, L1P i L1D). Svako C66 jezgro poseduje 32 KB L1P i L1D keš (eng. *cache*) memorije. Pored toga, svako C66 jezgro poseduje i 512 KB L2 memorije koja može da radi u keš ili RAM režimu. TI C6678 DSP pored pomenutih memorija poseduje i 4 MB posebne deljene RAM memorije (eng. *multicore shared memory*) koja je dostupna svim jezgrima jednog DSP. Deljena memorija može da radi u RAM ili L3 režimu. Ukoliko bi se L2 memorija svih jezgara koristila kao RAM memorija, onda TI C6678 DSP ima ukupno 8 MB unutrašnje memorije koja može da se koristi za smeštanje podataka obrade [3].

Jedina periferija kojom neposredno rukuje PQM sistem je kontroler direktnog pristupa memoriji (eng. *Direct Memory Access, DMA*). Periferija koja na TI C6678 DSP predstavlja DMA kontroler je od strane TI nazvana Enhanced Direct Memory Access 3 (EDMA3). EDMA3 kontroler služi za efikasan prenos podataka iz različitih memorijskih nivoa. [4].

Najvažnije komponente fizičke arhitekture C6678 DSP prikazani su na slici 2.1. Više podataka o TI C6678 DSP, memorijskoj strukturi, podržanim periferijama, kao i o EDMA3 kontroleru dostupno je u [2], [3] i [4].

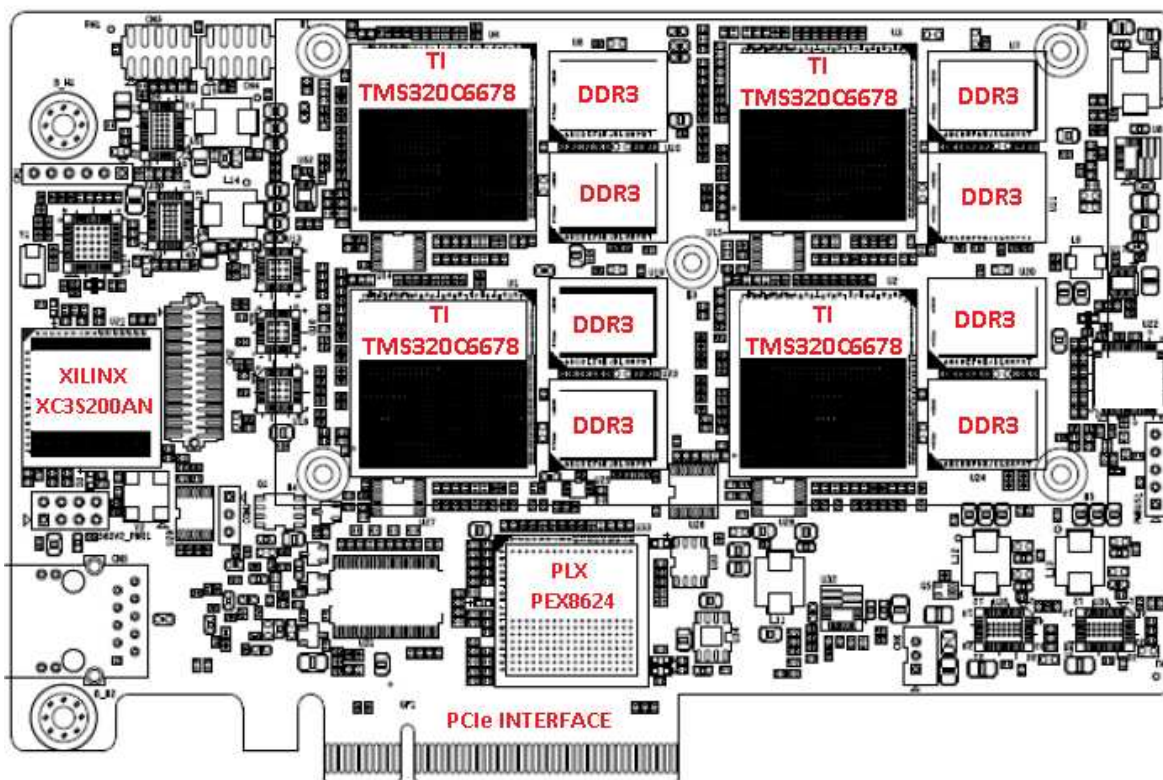


Slika 2.1 – Najvažnije komponente fizičke arhitekture TI C6678 DSP

2.2 Advantech Lightning PCIe ploča

Advantech Lightning (DSPC-8681E) PCIe ploča predstavlja okosnicu podsistema za paralelnu obradu digitalnog signala. Pored procesora za digitalnu obradu signala, Advantech Lightning ploča poseduje veliki broj periferija i sprega (eng. *interface*) koje su dostupne korisniku, pa je iz tog razloga u ovom poglavlju dat samo kratak pregled ploče i njenih najvažnijih fizičkih komponenti. Na slici 6.2 prikazan je izgled Advantech Lightning (DSPC-8681E) PCIe ploče sa označenim najvažnijim fizičkim komponentama. Najvažnije fizičke komponente koje se koriste u PQM sistema su:

- Texas Instruments TMS320C6678 DSP
- PCIe sprega
- DDR3 memorija
- PLX PEX8624 PCIe usmerač (eng. *switch*)
- Xilinx XC3S200AN FPGA



Slika 2.2 – Izgled Advantech Lightning ploče sa označenim najvažnijim komponentama

Advantech Lightning ploča se sa ostatkom dela sistema (PC računarem) povezuje preko magistrale za brzo povezivanje periferija (eng. *PCI Express, PCIe*). Takođe, Advantech Lightning PCIe ploča poseduje 4 Texas Instruments TMS320C6678 DSP koja su u potpunosti

dostupna za obradu. Texas Instruments TMS320C6678 DSP je detaljnije opisan u prethodnom odeljku ovog poglavlja. Svaki od C6678 DSP je podržan sa 1 GB DDR3 spoljašnje memorije koja može da se koristi za smeštanje i razmenu podataka između jezgara [5]. C6678 DSP su povezani PLX PEX8624 PCIe usmerivačem preko kojeg je omogućena međusobna komunikacija. Xilinx XC3S200AN FPGA obezbeđuje upravljanje svim kontrolnim signalima ploče. Više podataka o Advantech Lightning PCIe ploče je dostupno u [5] i [6].

2.3 SYS/BIOS RTOS

Sa stanovišta programske podrške koja je dostupna na jezgrima DSP, u ovom odeljku ukratko je opisana struktura i najvažniji delovi SYS/BIOS OS. Više podataka o strukturi, funkcionisanju i načinu modifikovanja SYS/BIOS OS dostupno je u [7].

SYS/BIOS je OS namenjen za rad u realnom vremenu (eng. *Real Time Operating System, RTOS*) koji je razvijen od strane Texas Instruments kompanije i koristi se u različitim tipovima namenskih uređaja (mikrokontroleri, DSP, ARM, itd.). SYS/BIOS je namenjen za korišćenje u namenskim sistemima koji treba da omoguće konkurentno izvršavanje više različitih procesa u realnom vremenu (eng. *multitasking*). Ono što je odlika SYS/BIOS RTOS je njegova modularnost i veličina. Veličina SYS/BIOS RTOS, u zavisnosti od odabranih modula koje uključuje, iznosi svega nekoliko KB.

Strukturu SYS/BIOS operativnog sistema čini veliki broj funkcionalnih modula koji se u zavisnosti od načina implementacije sistema mogu, a i ne moraju, koristiti. Naime, svaki od modula omogućava odgovarajuće programske primitive koje se mogu koristiti po potrebi. Recimo, ukoliko neki sistem ima potrebu za konkurentnim izvršavanjem više algoritama obrade na jednoj arhitekturi, tada on može uključiti modul koji SYS/BIOS OS omogućava upotrebu sistemskih programskih niti (eng. *OS threads*). U tom slučaju, funkcionalnost i programske primitive odabranog modula unutar SYS/BIOS OS su dostupne preko odgovarajuće aplikativne sprege. Modularnost SYS/BIOS OS je veoma korisna jer se njome može postići dobar odnos između veličine OS, koja je u namenskim sistemima od izuzetnog značaja, i potrebne funkcionalnosti koju programska podrška treba da obezbedi.

Najvažniji delovi programske podrške koji su dostupni u SYS/BIOS OS su:

- podrška za konkurentno izvršavanje niti (eng. *multitasking*)
- podrška za upravljanje memorijom (eng. *memory management unit*)

2.3.1 Podrška za konkurentno izvršavanje niti

SYS/BIOS OS omogućuje korišćenje nekoliko tipova niti:

- nit fizičkog prekida (eng. *Hardware Interrupt, Hwi*)
- nit programskog prekida (eng. *Software Interrupt, Swi*)
- nit zadatka obrade (eng. *Task*)
- nit čekanja (eng. *Idle*)

Nit fizičkog prekida predstavlja apstrakciju prekidne rutine (eng. *Interrupt Service Routine, ISR*), i ovaj tip niti u SYS/BIOS OS ima najveći prioritet izvršenja. Telo funkcije niti fizičkog prekida sa stanovišta vremena obrade treba da bude što kraće, tako da su neke funkcije pojedinih modula unutar ovog tipa niti zabranjene (npr. čekanje semafora). Nit fizičkog prekida se koristi isključivo za funkcije obrade prekida.

Niti programskog prekida (eng. *Software Interrupt, Swi*) predstavljaju novi tip niti koji u OS opšte namene nije toliko zastupljen. Programska nit prekida ima slične osobine kao nit fizičkog prekida, s tim da je prioritet izvršavanja programskih niti niži. Naime, ideja programske niti jeste da se obezbedi programska rutina koja se izvršava na višem nivou prioriteta nego standardne niti obrade zadataka (eng. *Task*) kako bi se mogla koristiti za delove programske obrade koji su kritični (npr. signalizacija).

Nit zadatka obrade po svojoj strukturi odgovara programskim nitima procesa koji postoje u OS opšte namene. Niti zadataka obrade se najčešće koriste za implementaciju određenih algoritama koji se sinhronizuju upotrebom niti programskog prekida. Prioritet niti zadataka obrade je moguće podešavati tako da određene niti obrade imaju različite prioritete, ali ti prioriteti su manji od prioriteta programskih i fizičkih niti prekida.

Nit čekanja (eng. *Idle*) predstavlja poseban tip niti u sistemu koja ima najniži prioritet izvršavanja. Ona se izvršava u situaciji kada ni jedna druga nit u sistemu nije aktivna.

2.3.2 Podrška za upravljanje memorijom

Podrška za upravljanje memorijom, pored podrške konkurentnog izvršavanja niti, predstavlja jednu od najvažnijih funkcionalnosti koja je dostupna upotrebom SYS/BIOS OS. Modul za upravljanje memorijom obezbeđuje alate za lako i jednostavno podešavanje memorijske mape, kao i alate koji omogućuju statičko i dinamičko zauzimanje određenih memorijskih zona kojima se može upravljati u toku izvršavanja programa.

3. Koncept rešenja

U ovom poglavlju opisana je arhitektura i ponašanje PQM sistema bez i sa implementiranim *ping-pong* mehanizmom. Takođe, detaljno je opisana funkcionalnost i namena svih gradivnih blokova rešenja.

3.1 Arhitektura PQM sistema

PQM je složeni sistem koji je namenjen za rad u realnom vremenu i koristi se u svrhu određivanja kvaliteta i pronalaženja artefakata digitalnog video signala koji mogu nastati kao posledica kompresije ili nekog drugog oblika digitalne obrade. PQM sistem se sastoji od dva fizički i funkcionalno odvojena podsistema koji su u međusobnom sadejstvu, a to su:

- upravljački podsistem (eng. *host*)
- podsistem za paralelnu obradu digitalnog signala (eng. *DSP processing board*)

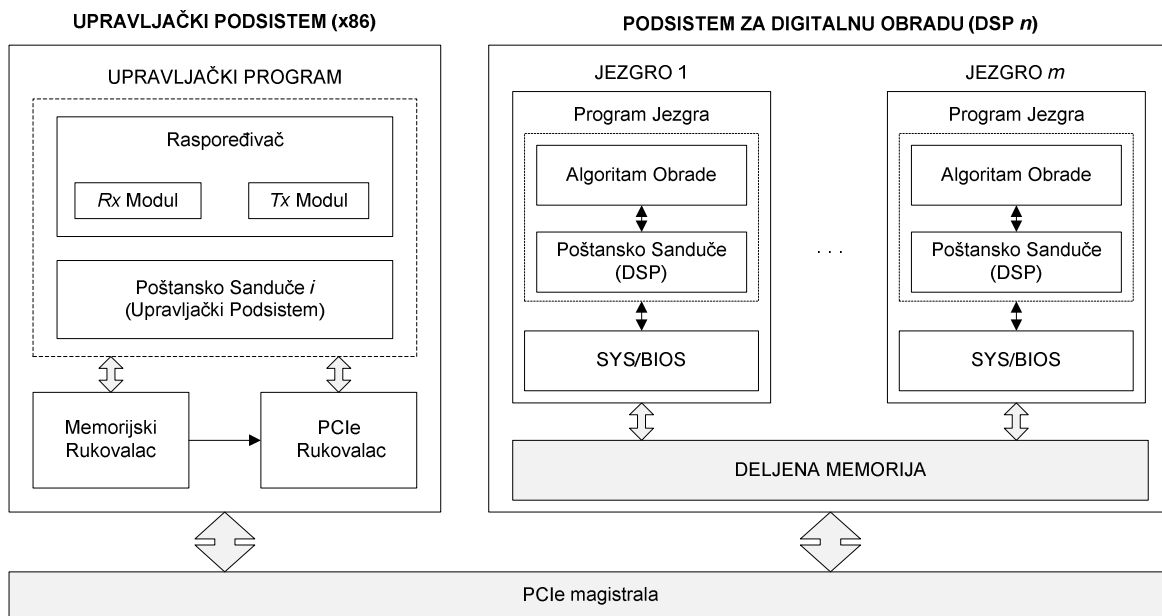
Fizičku okosnicu upravljačkog podsistema čini x86 arhitektura. Upravljački podsistem ima zadatak nadzora i upravljanja radom podsistema za paralelnu obradu digitalnog signala na kom se nalaze algoritmi obrade.

Za razliku od upravljačkog podsistema, fizičku okosnicu podsistema za paralelnu obradu čine jedan, ili više, DSP sa više jezgara. Podsistem za paralelnu obradu digitalnog signala implementira određene algoritme obrade na osnovu čijih rezultata se pouzdano može utvrditi da li se određeni artefakt pojavio, ili je kvalitet signala, odnosno video materijala, na zadovoljavajućem nivou.

Dva dela PQM sistema su fizički povezani brzom magistralom za povezivanje periferija (eng. *Peripheral Component Interconnect Express, PCIe*) preko koje je omogućena međusobna

komunikacija kao i prenos podataka za obradu. U konkretnom slučaju PCIe magistrala predstavlja optimalan izbor sprege za povezivanje jer ona pruža visoku propusnost podataka u realnom vremenu koja je u slučaju obrade video signala neophodna.

Na slici 3.1 predstavljena je arhitektura PQM sistema. U nastavku ovog poglavlja predstavljena je struktura upravljačkog programa, struktura podsistema za paralelnu obradu digitalnog signala, kao i opis ponašanja PQM sistema bez i sa podržanim *ping-pong* mehanizmom.



Slika 3.1 – Arhitektura PQM sistema

3.2 Upravljački podsistem

Kao što je ranije pomenuto, fizičku okosnicu upravljačkog podsistema čini x86 arhitektura. Osnovna programska podrška je dostupna u vidu Linux operativnog sistema (eng. *Operating System, OS*). Upravljački podsistem PQM sistema čine sledeći moduli:

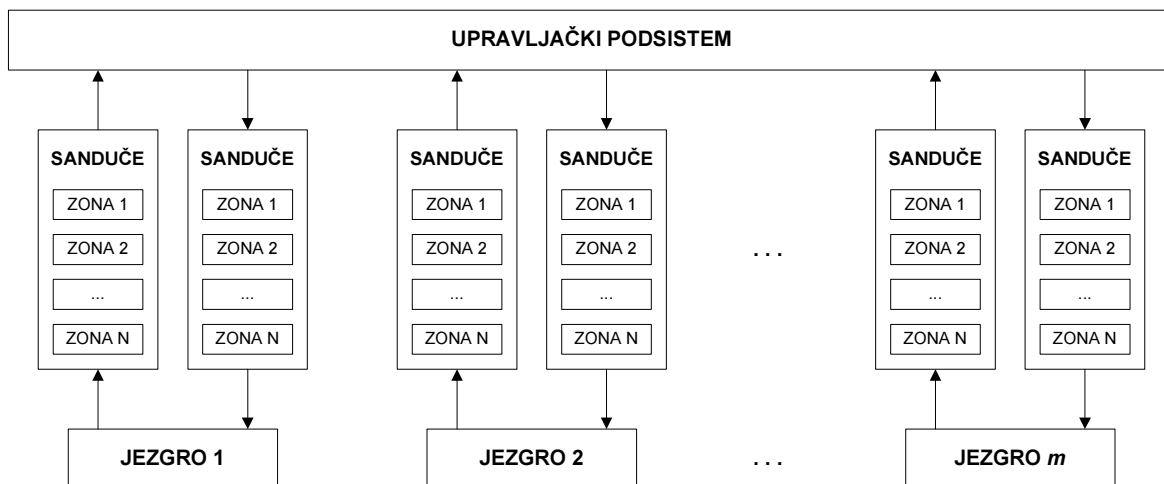
- Upravljački program (eng. *Core Controller*)
- Poštansko sanduče (eng. *Mail Box*)
- Memorijski rukovalac (eng. *Memory Driver*)
- PCIe rukovalac (eng. *PCI Express Driver*)
- Raspoređivač (eng. *Scheduler*)

3.2.1 Upravljački program

Upravljački program PQM sistema (eng. *Core Controller*) je implementiran na računaru opšte namene i njegov osnovni zadatak je upravljanje podsistemom za paralelnu obradu. Pod upravljanjem se smatra uspostavljanje i održavanje komunikacije sa podsistemom za paralelnu obradu na kome su implementirani algoritmi za obradu digitalnog video signala, obezbeđivanje učitavanja i prenos podataka za obradu do konkretnog procesora, odnosno do konkretnog jezgra, kao i prihvatanje rezultata obrade. Podaci koji se šalju na obradu predstavljaju okvire (eng. *frame*) koji čine video materijal čiji kvalitet se treba proceniti. Pored pomenutih funkcija program treba da omogući unos početnih parametara i da izvrši početno podešavanje PQM sistema.

3.2.2 Poštansko sanduče

Poštansko sanduče (eng. *Mail Box*) je jedan od modula koji omogućava komunikaciju između upravljačkog programa koji se izvršava na računaru i algoritama za obradu slike koji su implementirani na DSP. Upotrebom mehanizma poštanskih sandučića omogućen je jednostavan način razmene poruka između više procesnih elemenata sistema (jezgara) koji učestvuju u paralelnoj obradi. Poštanski sandučići služe za razmenu kratkih poruka, kao što su informacije o adresi memorije na kojoj se nalaze podaci spremni za obradu, njihova veličina, ili za slanje kontrolnih poruka. Na slici 3.2 prikazana je struktura poštanskog sandučeta.



Slika 3.2 – Struktura poštanskog sandučeta

Kao što se može videti sa slike 3.2, slanje poruka kroz poštansko sanduče je jednosmerno, pa za svako jezgro postoje dva poštanska sandučeta koja omogućuju dvosmernu

komunikaciju. Jedno poštansko sanduče se koristi za slanje poruka od upravljačkog programa ka određenom jezgru dok se drugo poštansko sanduče koristi za slanje poruka od određenog jezgra ka upravljačkom programu. Za svako poštansko sanduče moguće je podesiti proizvoljnu veličinu memorije koja je podeljena u više zona (eng. *slot*). Memorijske zone omogućuju podsistemima da sačuvaju proizvoljan broj novopristiglih poruka koje su poslate od druge strane. Ova mogućnost je neophodna u situacijama kada jezgro zbog obavljanja prioritetnijeg zadatka nije u mogućnosti da trenutno obradi novopristiglu poruku.

3.2.3 Memorijski rukovalac

Za prenos podataka sa računara na DSP i obrnuto koristi se mehanizam sa direktnim pristupom memoriji (eng. *Direct Memory Access, DMA*). Iako su podržani drugi mehanizmi prenosa kao što su neposredno kopiranje podataka preko PCIe magistrale upotrebom C *memcpy* funkcije i neposredno preslikavanje memorijskih lokacija PC računara na memorijske lokacije DSP, DMA mehanizam kao najbrži je odabran za konkretno rešenje.

Da bi DMA prenos podataka mogao funkcionisati potrebno je obezbediti da memorijski prostor iz kojeg se čitaju podaci, odnosno memorijski prostor u koji se podaci upisuju, bude neprekidan. Operativni sistem pod kojim se implementira upravljački program (Linux) ima ograničenje u pogledu memorijskog prostora koji može biti zauzet u bloku. Maksimalni neprekidni blok memorije koji može biti zauzet iznosi 4 MB, što je nedovoljno za podatke koji se prenose na obradu. Tipično, veličina slike visoke rezolucije (eng. *High Definition, HD*) u RGB prostoru boja koja se šalje na obradu iznosi približno 6 MB.

Ukoliko se ne obezbedi odgovarajući blok neprekidne memorije koji je potreban, DMA prenos ne bi bio moguć. Zadatak memorijskog rukovaoca (eng. *Memory Driver*) je da omogući dodelu neprekidnog memorijskog prostora proizvoljne veličine. Memorijski rukovalac obezbeđuje mehanizme za zauzimanje, rukovanje i oslobađanje neprekidnog memorijskog bloka proizvoljne veličine. Ukoliko se ne bi koristio memorijski rukovalac, maksimalna veličina neprekidnog memorijskog bloka bila bi ograničena na 4 MB.

3.2.4 PCIe rukovalac

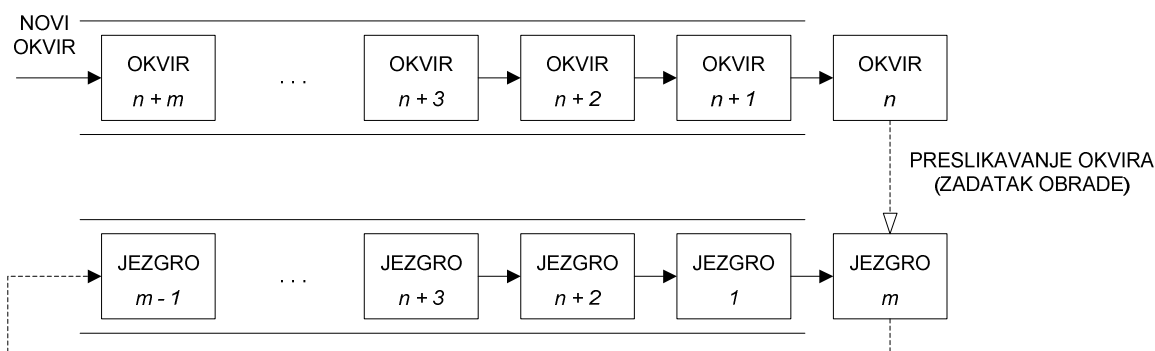
PCIe rukovalac (eng. *PCI Express Driver*) je modul koji kao i poštansko sanduče omogućuje međusobnu komunikaciju upravljačkog podsistema i podsistema za paralelnu obradu digitalnog signala. Za razliku od poštanskog sandučeta koje omogućava razmenu kratkih kontrolnih poruka, PCIe rukovalac treba da obezbedi prenos svih podataka za obradu iz radne memorije upravljačkog podsistema do deljene memorije DSP kojoj mogu pristupiti sva jezgra

konkretnog procesora. Naime, podaci koji se šalju preko poštanskog sandučeta na kraju moraju proći kroz PCIe rukovalac jer on u suštini predstavlja apstrakciju brze magistrale za povezivanje periferija koja povezuje dva podsistema u jedinstveni PQM sistem. Osnovni zadatak ovog modula je da omogući prenos podataka između dva podsistema na najnižem fizičkom nivou. Takođe, PCIe rukovalac treba da omogući upravljanje određenim periferijama koje su dostupne na DSP, a koje su od značaja za ciljni sistem. U slučaju PQM sistema, PCIe rukovalac omogućuje upravljanje DMA kontrolerom u svrhu ostvarivanja DMA prenosa podataka.

3.2.5 Raspoređivač

Raspoređivač (eng. *Scheduler*) predstavlja složen modul unutar upravljačkog programa čija je osnovna namena da omogući raspoređivanje zadataka obrade (eng. *task*) na odgovarajuća jezgra DSP. Kao što je ranije rečeno, na svakom jezgri DSP implementiran je po jedan algoritam obrade koji omogućava analizu video okvira na osnovu čega se utvrđuje kvalitet video signala. Sa stanovišta algoritama, obrada jednog video okvira predstavlja elementarni zadatak obrade koji im može biti dodeljen.

Raspoređivač zadataka je složen modul zato što se sastoji od dva međusobno odvojena podmodula koja imaju različitu funkcionalnost. To je modul za slanje video okvira na obradu (Tx modul) i modul za prihvatanje rezultata obrade (Rx modul). Raspoređivač za planiranje i raspodelu zadataka koristi algoritam redom u krug (eng. *Round Robin, RR*) [8]. U nastavku ovog odeljka na slikama 3.3 i 3.4 prikazan je proces preslikavanja zadataka obrade na pojedinačna jezgra, proces raspoređivanja zadataka, kao i prihvatanje rezultata obrade.



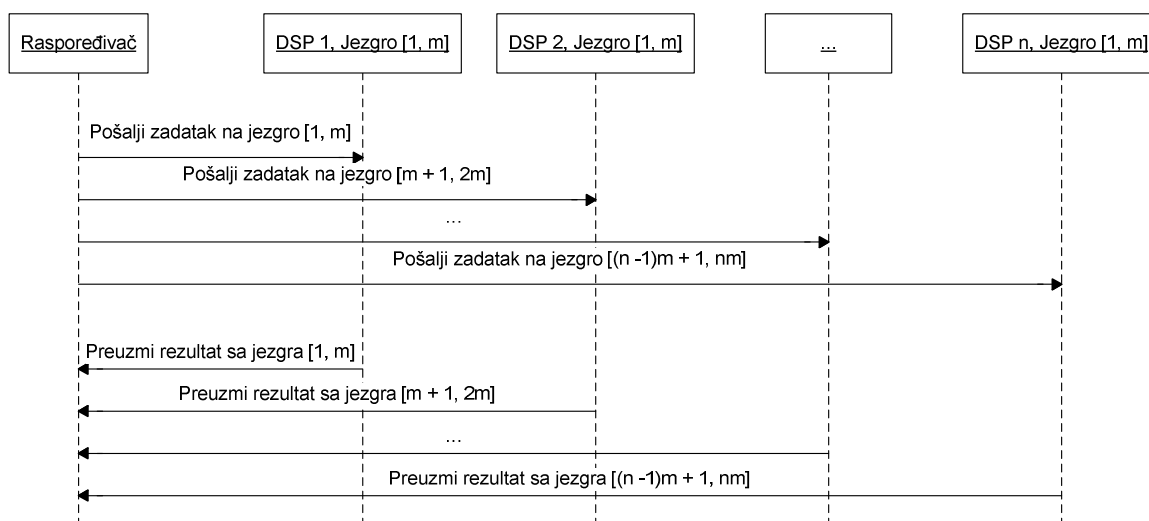
Slika 3.3 – Preslikavanje zadataka obrade na odgovarajuća jezgra DSP

U procesu raspoređivanja zadataka za obradu, prvi zadatak se raspoređuje na unapred definisano jezgro DSP nakon čega se svi naredni zadaci raspoređuju sekvencijalno na ostala jezgra. Sva jezgra koja učestvuju u obradi su numerisana, i sa stanovišta raspoređivača

predstavljaju niz jedinica obrade na koje se preslikavaju zadaci koji treba da budu obrađeni. Konkretno, početno jezgro na koje se raspoređuje prvi zadatak obrade predstavlja prvo jezgro unutar prvog DSP koji je dostupan u podsistemu za paralelnu obradu (jezgro 0, DSP 0), nakon čega se proces raspoređivanja odvija po RR algoritmu planiranja zadataka. Po raspoređivanju poslednjeg zadatka na poslednje jezgro u nizu, raspoređivač naredni posao dodeljuje početnom jezgrou sa kojim je i počeo raspoređivanje.

Sa obzirom da raspoređivač sva jezgra u podsistemu za paralelnu obradu digitalnog signala logički posmatra kao niz procesnih elemenata koji su dostupni za paralelnu obradu, parametri koji su potrebni za podešavanje raspoređivača jesu broj jezgara koja se nalaze na DSP (m) i broj DSP koji se nalaze na PCIE ploči (n). Na osnovu ova dva parametra svako od jezgara dobija jedinstveni broj (eng. *node ID*) koji se koristi u procesu izvršavanja.

Proces prihvatanja rezultata obrade prati proces raspoređivanja zadataka obrade, odnosno rezultati obrade se očekuju u redosledu u kome su zadaci raspoređeni. Prvo se očekuje da početno jezgro u nizu vrati rezultat obrade (jezgro 0, DSP 0), nakon čega se čeka rezultat narednog jezgra. Kada se preuzmu rezultati sa svih jezgara (jezgro m , DSP n), proces raspoređivanja se ponavlja istim redosledom, kao što RR algoritam definiše.



Slika 3.4 – Proces raspoređivanja zadataka i prihvatanja rezultata obrade

Upotreba raspoređivača sa RR algoritmom za raspoređivanje zadataka u konkretnom slučaju nije optimalno ali jeste zadovoljavajuće rešenje. Potencijalni problem se ogleda u načinu prihvatanja rezultata obrade. Naime, trenutnim izborom RR algoritma za raspoređivanje i planiranje zadataka, potencijalno je moguća situacija da je neko jezgro završilo svoj posao i da je spremno za predaju rezultata, a da trenutno upravljački program očekuje rezultat od nekog

drugog jezgra koje se nalazi prethodno u nizu, a koje nije završilo svoju obradu. Ovakva pojava ne smanjuje performanse sistema ali postoji mogućnost da ih delimično ograniči.

3.3 Podsystem za paralelnu obradu digitalnog signala

Podsystem za paralelnu obradu digitalnog signala čine jedan ili više DSP sa više jezgara na kojima postoji odgovarajuća programska podrška koja omogućuje primenu različitih programskih gradivnih blokova u cilju realizacije paralelne obrade.

Sa stanovišta upravljačkog podsystema, skup DSP sa više jezgara u PQM sistemu predstavlja koprocesor koji služi za realizaciju procesorski veoma zahtevnih algoritama. Svako jezgro u podsystemu za paralelnu obradu se posmatra kao jedan procesni element koji učestvuje u paralelnoj obradi. Ovakav način posmatranja jezgara omogućava jednostavniju podelu ukupnog posla na manji broj zadataka koji se mogu izvršavati paralelno. Kao što je prikazano na slici 3.1, pretpostavlja se da podsystem za paralelnu obradu na neki način podržava deljenu memoriju kojoj neposredno mogu pristupati sva jezgra. Zbog specifičnosti svake arhitekture, u ovom odeljku, broj jezgara, veličina keš, interne i deljene memorije nisu razmatrani. Takođe, iako na slici 3.1 nije prikazano, podrazumeva se da je arhitekturom podržan DMA kontroler ili neki drugi mehanizam pristupa deljenoj memoriji koji je dostupan za korišćenje upravljačkom podsystemu. Sve informacije o konkretnoj fizičkoj arhitekturi koja je korišćena za implementaciju PQM sistema (DSP i PCIE ploča) su dostupne u drugom poglavlju.

Obrada na arhitekturi sa više jezgara podeljena je u nekoliko modula:

- Poštansko sanduče
- SYS/BIOS
- Program jezgra (eng. *Core Application*)
- Algoritam obrade (eng. *Processing Algorithm*)

3.3.1 SYS/BIOS

Na svakom jezgru, sistemska programska podrška je dostupna u vidu SYS/BIOS OS. SYS/BIOS je OS namenjen za rad u realnom vremenu (RTOS) i koristi se u različitim namenskim sistemima. SYS/BIOS RTOS omogućava različit skup sistemskih funkcija koje omogućava i OS opšte namene kao što je Linux OS, koji je osnova sistemske programske podrške upravljačkog podsystema.

Sa stanovišta realizacije ostalih modula podsystema za paralelnu obradu, najvažnija podrška koju omogućava SYS/BIOS RTOS je podrška za konkurentno izvršavanje sistemskih

нити (eng. *threads*). Upotrebom niti na jednom jezgru DSP moguće je realizovati više različitih algoritama obrade čime se postiže bolje iskorišćenje jezgara. SYS/BIOS RTOS je detaljnije opisan u odgovarajućem odeljaku poglavlja Teorijske osnove.

Pored obezbeđivanja osnovnih programskih gradivnih blokova kojima se realizuje obrada na jezgrima DSP, SYS/BIOS RTOS omogućuje i funkcionalnost koja je u upravljačkom podsistemu dodeljena PCIe rukovaocu. Isto kao i PCIe rukovalac, omogućava pristup i upravljanje svim periferijama koje su dostupne na određenom DSP.

3.3.2 Poštansko sanduče

Modul poštanskog sandučeta koji se koristi u podsistemu za paralelnu obradu digitalnog signala se funkcionalno ne razlikuje od modula poštanskog sandučeta koje se koristi u upravljačkom podsistemu. Naime, njihova struktura, zadatak i namena je suštinski ista. Oba modula se koriste za razmenu kratkih poruka. Više podataka o načinu upotrebe i strukturi poštanskog sandučeta je dostupno u odeljku 3.2.1 ovog poglavlja.

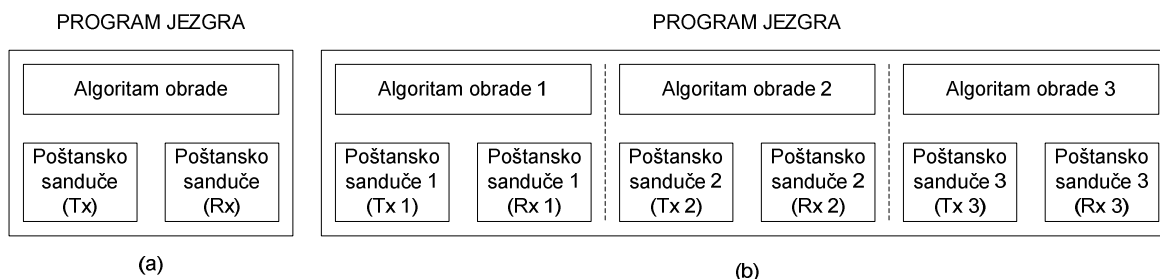
3.3.3 Program jezgra

Program jezgra (eng. *Core Application*) predstavlja složen modul koji obuhvata sledeće module:

- Algoritam obrade
- Poštansko sanduče

Program jezgra je modul koji objedinjuje sve algoritme obrade koji su implementirani na jednom jezgru, kao i njihove poštanske sandučiće koji služe za komunikaciju sa upravljačkim programom na drugoj strani. Postojanje ovog modula je neophodno jer raspoređivač zadataka upravljačkog programa svako jezgro posmatra kao jedinstveni program kome šalje podatke za obradu. Međutim, na jezgru nije isključeno da postoji implementirano više različitih algoritama obrade koji se izvršavaju nad istim ili različitim podacima. Da li će se na jednom jezgru implementirati samo jedan ili više algoritama obrade zavisi od prirode ciljnog sistema. Na primer, (i) ukoliko ciljni sistem ne mora da garantuje rad u realnom vremenu, na jezgru može da bude implementirano proizvoljno mnogo algoritama obrade. Sa druge strane, (ii) ako ciljni sistem treba da garantuje rad u realnom vremenu, u zavisnosti od složenosti obrade i procesne snage jezgra, na svakom jezgru se implementira znatno manje algoritama. Poštanski sandučići su potrebni kako bi se mogla ostvariti međusobna komunikacija i razmena poruka.

Na slici 3.5 prikazana je struktura modula programa jezgra u sistemu sa implementiranim jednim i više algoritama obrade. Na slici pod (a) prikazan je jedan algoritam obrade sa njemu odgovarajućim poštanskim sandučićima za slanje (Rx) i primanje (Tx) poruka. Pored, na slici pod (b) su prikazana tri algoritma obrade, takođe sa svojim odgovarajućim poštanskim sandučićima. Uopšteno, ukoliko su pretpostavljeni algoritmi iste složenosti, prvi slučaj može odgovarati sistemu koji treba da garantuje rad u realnom vremenu, dok drugi slučaj može odgovarati sistemu čiji vremenski rokovi trajanja, odnosno završetka obrade, nisu kritični.



Slika 3.5 – Struktura modula programa jezgra sa jednim i više algoritma obrade

3.3.4 Algoritam obrade

Modul algoritma obrade (eng. *Processing Algorithm*) predstavlja proizvoljni algoritam koji je implementiran na nekom od jezgara DSP i koji se izvršava na ulaznim podacima. Algoritam obrade je izdvojen kao poseban modul jer on predstavlja posebnu logičku i funkcionalnu celinu podсистema za paralelnu obradu. U prethodnim odeljcima ovog poglavlja predstavljena je prednost izdvajanja algoritama obrade kao posebne celine, a to je prvenstveno mogućnost efikasnijeg iskorišćenja potencijala jezgara DSP. Takođe, posmatranje algoritama obrade kao poseban modul omogućuje lakšu, bolju i efikasniju organizaciju obrade.

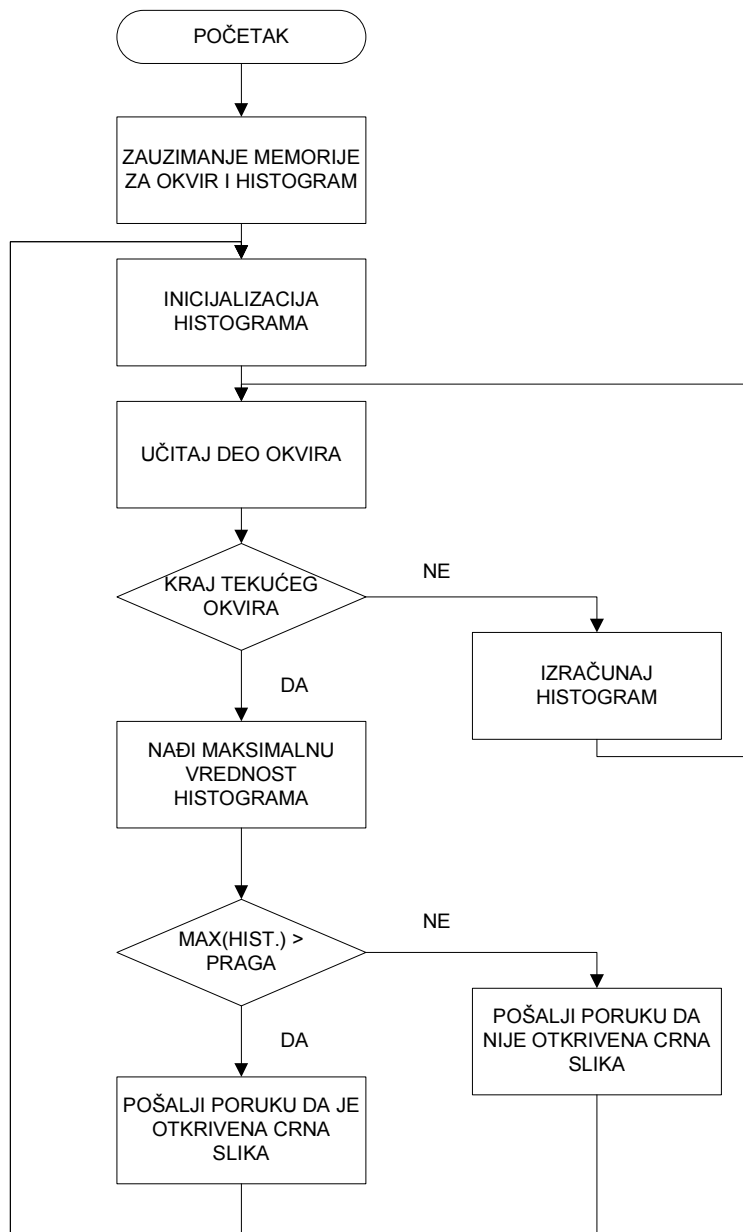
Uopšteno gledano, sa stanovišta obrade, na jezgrima DSP mogu biti implementirani različiti tipovi algoritama, međutim, ukoliko se pogleda fizička arhitektura podсистema za paralelnu obradu digitalnog signala koju čine DSP, intuitivno se može zaključiti da najbolje performanse sa stanovišta brzine izvršavanja mogu ostvariti algoritmi za obradu digitalnog audio i video signala (npr. FFT, IFFT, DCT, filtri, itd.). U nastavku ovog odeljka predstavljen je algoritam koji je korišćen u PQM sistemu za obradu video signala.

3.3.4.1 Algoritam za otkrivanje crne slike (BSD algoritam)

U konkretnom rešenju PQM sistema na jezgrima DSP implementira se algoritam koji omogućava otkrivanje crne slike (eng. *Black Screen Detection, BSD*). Blok dijagram BSD

algoritma je prikazan na slici 3.6. BSD algoritam omogućava otkrivanje specifične crne slike koja se pojavljuje u nizu slika koje čine video materijal. Pojava crne slike koju otkriva algoritam je specifična jer se ona razlikuje od crne slike koja prirodno čini sastavni deo video materijala. Pojava crne slike može nastati kao posledica prouzrokovana od strane više faktora:

- procesa digitalne obrade signala
- neispravnosti uređaja za slanje signala (transpondera)
- neispravnosti uređaja za dekodovanje video signala (STB)



Slika 3.6 – Blok dijagram BSD algoritma

3.4 Ponašanje PQM sistema

Ukoliko se malo detaljnije analizira memorijska arhitektura u prethodno opisanom PQM sistemu može se primetiti da ne postoji jedinstvena, centralizovana, memorija koja bi istovremeno mogla biti deljena između svih delova sistema. Drugačije rečeno, opisani PQM sistem poseduju distribuiranu memorijsku arhitekturu.

Iako na slici 3.1 nije prikazana radna memorija upravljačkog podsistema, ona se podrazumeva. Memorijsku mapu PQM sistema čine dva tipa memorije:

1. radna memorija upravljačkog podsistema
2. deljena memorija koja se nalazi na DSP

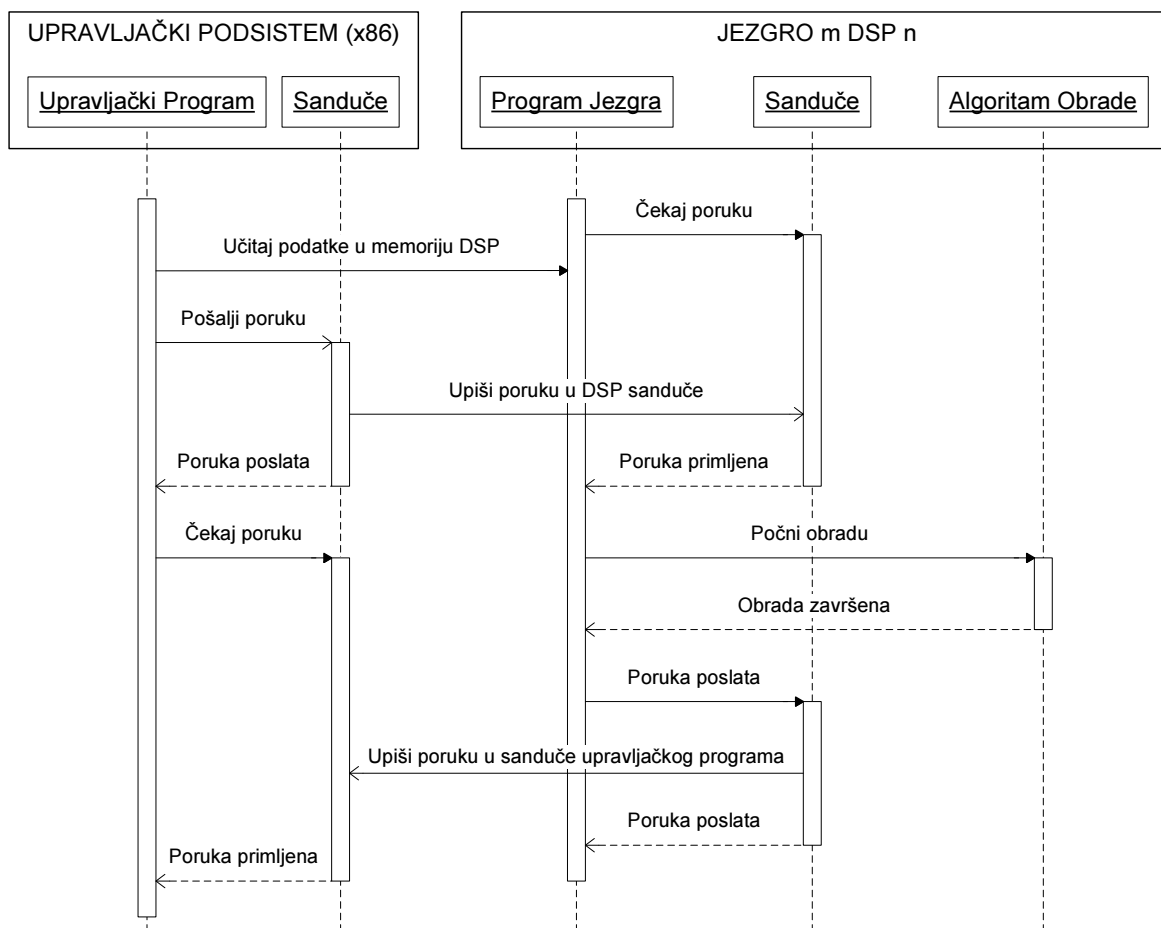
Sa stanovišta podataka, radna memorija upravljačkog podsistema služi za čuvanje i smeštanje podataka koji treba da budu poslani na obradu, ili podataka koji su obrađeni. Ovaj tip memorije se takođe koristi za smeštanje podataka koje je potrebno predprocesirati kako bi podaci bili spremni za slanje na stranu DSP.

Deljena memorija predstavlja memoriju koja je dostupna na svakom DSP i koja je namenjena za korišćenje od strane svih jezgara koji se nalaze na konkretnom DSP. Deljena memorija se uglavnom koristi za smeštanje trenutno obrađivanih podataka i podataka koji čekaju na obradu. Jedini izuzetak kad se deljena memorija koristi za nešto drugo jeste prilikom inicijalizacije sistema kada se u njoj nalaze različiti tipovi struktura podataka sa početnim parametrima inicijalizacije, kako za sam DSP tako i za PQM sistem.

3.4.1 Ponašanje PQM sistema bez *ping-pong* mehanizma

Uzimajući u obzir distribuiranu memorijsku arhitekturu sistema i njegovu funkcionalnost, na slici 3.7 dat je dijagram sekvenci koji opisuje ponašanje (eng. *behavior*) PQM sistema bez implementiranog *ping-pong* mehanizma. Na strani upravljačkog podsistema prikazani su upravljački program i poštansko sanduče upravljačkog programa, dok je na strani jezgra prikazan program jezgra, poštansko sanduče DSP i algoritam obrade. Početak procesa obrade na jezgru započinje tako što program jezgra daje znak poštanskom sandučetu DSP da čeka poruku od upravljačkog programa kako bi se mogao pokrenuti algoritam obrade. Istovremeno, na drugoj strani, upravljački program iz radne memorije upravljačkog podsistema prebacuje podatke koje je potrebno obraditi u deljenu memoriju DSP. Kada je prenos podataka završen, upravljački program šalje poruku programu jezgra na drugoj strani da su podaci za obradu spremni. Poruka sadrži sve informacije koje od interesa za proces obrade (adresa ulaznih podataka, veličina, itd.). Nakon poslane poruke, upravljački program čeka odgovor kojim se označava da je obrada na

drugoj strani završena, i sa stanovišta upravljačkog programa proces se onda ponavlja. Sa druge strane, program jezgra, nakon primljene poruke, i u skladu sa njenim sadržajem pokreće odgovarajući algoritam obrade. Po završetku obrade, slanjem poruke, upravljačkom programu se signalizira da je obrada završena, nakon čega se proces ponavlja. Opet, u skladu sa obradom, poruka sadrži sve informacije od interesa.



Slika 3.7 – Ponašanje PQM sistema bez implementiranog *ping-pong* mehanizma

3.4.2 Osnove *ping-pong* mehanizma

Sušтина funkcionisanja *ping-pong* (PP) mehanizma je slikovito sadržana u samom nazivu mehanizma. PP mehanizam se odnosi na način rukovanja, odnosno prenosa podataka, i služi kao mehanizam za prevazilaženje problema koje sa sobom donosi distribuirana memorijska arhitektura.

U sistemima koji poseduju distribuiranu memorijsku arhitekturu, kao što je PQM sistem, postoji problem koji se odnosi na rukovanje i pristup podacima bilo u svrhu čitanja ili pisanja.

Problem se ogleda u tome što svaki od delova sistema koji želi da pristupi određenim podacima kod sebe u lokalnoj radnoj memoriji mora imati kopiju tih podataka, tj. svaki put kad se želi pristupiti novim podacima koji se ne nalaze u lokalnoj radnoj memoriji potrebno je sačekati da se oni učitaju i postanu dostupni. Ovo u nekim sistemima opšte namene ne predstavlja problem jer performanse i vremenski rokovi za završetak određenih operacija nisu od ključne važnosti. Za razliku od sistema opšte namene, kod namenskih sistema koji treba da omoguće rad u realnom vremenu ovaj nedostatak predstavlja ozbiljno ograničenje.

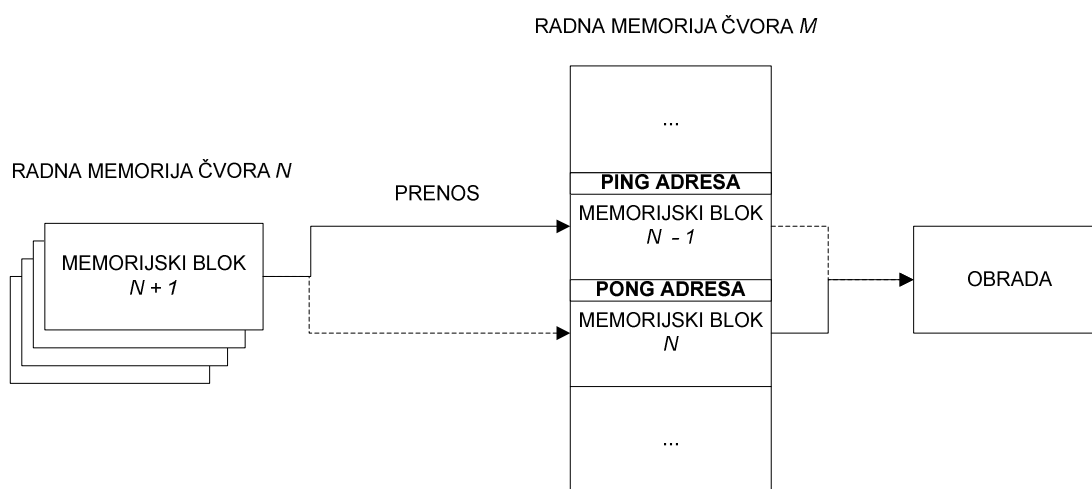
Kao primer posmatrajmo distribuirani sistem u kojem postoje dva čvora koji su u takvom odnosu da je jedan čvor glavni (eng. *master*) a drugi podređeni (eng. *slave*). Pretpostavimo da se podređeni čvor koristi isključivo kao deo za obradu podataka koji se na ulazu sistema pojavljuju periodično (istom frekvencijom). Ako se prethodno pomenuti problem distribuirane memorije analizira sa stanovišta pretpostavljenog sistema, može se primetiti da glavni čvor nakon poslate grupe podataka na podređeni čvor biva upošljen na svojim lokalnim zadacima sve dok ne dobije zahtev da je podređeni čvor završio obradu, odnosno da je podređeni čvor spreman za obradu nove grupe podataka. Glavni čvor po primanju poruke da je obrada gotova počinje sa slanjem nove grupe podataka. Prethodno opisano ponašanje se odnosi na glavni čvor. Ukoliko posmatramo podređeni čvor, njegova radna procedura se svodi na (i) čekanju nove grupe podataka koju treba da obradi, (ii) konkretnu obradu i (iii) komunikaciju sa glavnim čvorom. Sa stanovišta performansi celog sistema kritični deo je vreme koje podređeni čvor provede na čekanju da stignu novi podaci. Naime, vreme obrade i komunikacije zavisi od implementiranog algoritma obrade i odabranog mehanizma komunikacije, što nije slučaj sa vremenom koje čvor provede u čekanju. Vreme provedeno u čekanju je jednako vremenu koje je potrebno da glavni čvor prebaci podatke iz svoje radne memorije u radnu memoriju podređenog čvora, koje opet zavisi od mehanizma prenosa podataka kao i same količine podataka koja se prenosi. U zavisnosti od odabira mehanizma prenosa podataka, za veliku količinu podataka (čak i reda nekoliko desetina MB), vreme prenosa može da bude značajno.

PP mehanizam pruža jednostavan način za prevazilaženje ovog problema na taj način što se u toku obrade jedne grupe podataka, nezavisno od tekuće obrade, prenosi druga grupa podataka koja je sledeća u redu za obradu. Primenom mehanizma učitavanja podataka, PP mehanizam u memorijski distribuiranom sistemu obezbeđuje prividnu centralizovanu memorijsku arhitekturu. Ovo je moguće zato što PP mehanizam u lokalnoj memoriji svakog distribuiranog čvora pravi bazen podataka (eng. *data pool*) u kojem se u svakom trenutku, bez dodatnog čekanja, nalaze novi podaci koji su spremni za obradu. U idealnom slučaju, ukoliko je obrada trenutna, vreme čekanja između dve obrade se svodi na vreme prenosa podataka od

jednog čvora do drugog, što je upotrebom nekog od mehanizama prenosa kao što je DMA prenos, veoma malo.

Zbog jednostavnosti, na slici 3.8 prikazan je primer PP mehanizma između dva distribuirana čvora. Primer pokazuje način prenosa bloka podataka za obradu iz radne memorije jednog čvora u radnu memoriju drugog čvora na kojem se oni obrađuju.

Ovo rešenje nije idealno jer zahteva dodatni memorijski prostor za podatke koji se učitavaju. U današnjim namenskim sistemima koji služe kao podrška za obradu podataka veličina memorije ne predstavlja kritičan resurs, tako da implementacija PP mehanizma ne utiče negativno na ostatak sistema.



Slika 3.8 – Primer *ping-pong* prenosa u memorijski distribuiranom sistemu

3.4.3 Ponašanje PQM sistema sa *ping-pong* mehanizmom

U prethodno opisanom ponašanju PQM sistema bez implementiranog *ping-pong* mehanizma (odjeljak 3.4.1) ne može se uticati na brzinu obrade ili na brzinu prenosa podataka jer ona veoma zavisi od konkretne fizičke arhitekture, ali se može uticati na vreme koje svako jezgro provede u čekanju na nove podatke. Naime, prilikom prenosa podataka iz radne memorije upravljačkog podsistema u deljenu memoriju DSP, koja se dešava između svake dve uzastopne obrade, svako jezgro provodi sve vreme čekajući poruku od upravljačkog programa, tj. da se završi prenos tekućih podataka za obradu. Ovo je posledica distribuirane memorijske arhitekture, jer da u sistemu postoji zajednička centralizovana memorija, pomenuto vreme čekanja ne bi postojalo. Kao što je i ranije pomenuto, jedan od načina za rešenje ovog problema je upotreba *ping-pong* (PP) mehanizma.

Primena PP mehanizma ne zahteva implementaciju novih programskih modula već se ogleda u načinu prenosa podataka. Osnova PP mehanizma je da omogući dvostruko učitavanje podataka obrade u deljenu memoriji DSP čime se izbegava čekanje na završetak prenosa podataka. Čekanje na završetak prenosa podataka se izbegava na taj način što u deljenoj memoriji uvek postoje podaci koji su spremni za obradu. Praktično, paralelno sa tekućom obradom u memoriji DSP se učitavaju sledeći podaci koje je potrebno obraditi. Na Slici 3.9 prikazano je ponašanje sistema sa implementiranim PP mehanizmom. Ponašanje na jezgrima podsistema za digitalnu obradu podataka je isto kao u prethodnom slučaju. Razlika je u načinu ponašanja upravljačkog podsistema.

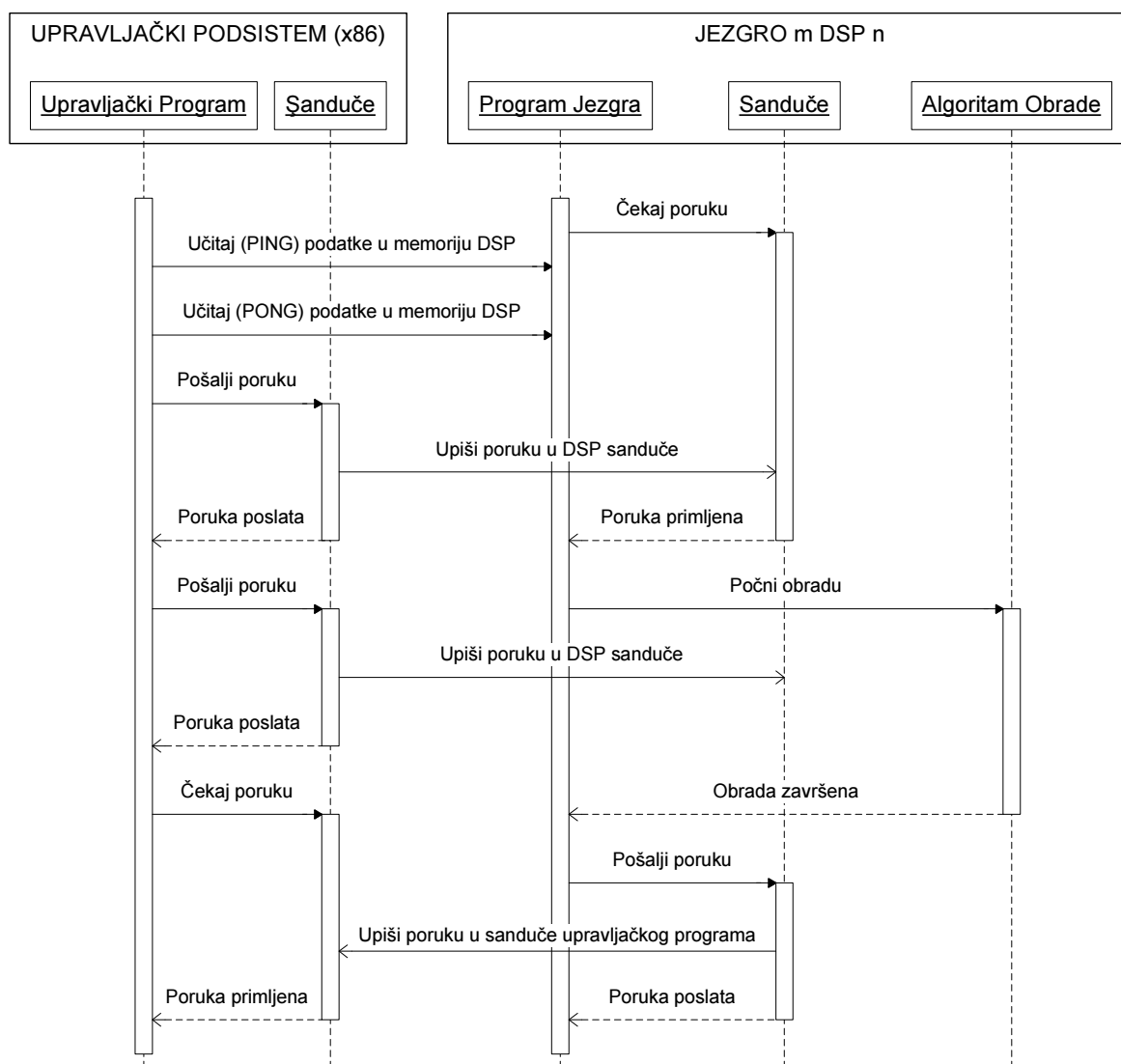
Može se videti da za razliku od prethodnog slučaja, upravljački program na početku izvršavanja u memoriju DSP prebacuje dve grupe podataka, označenih sa PING i PONG. Nakon prenosa podataka upravljački program šalje dve poruke drugoj strani. Na ovaj način, programu jezgra je omogućeno da odmah nakon završetka obrade PING počne sa obradom PONG podataka, bez čekanja na završetak prenosa novih podataka obrade. Sa druge strane, upravljački program kad primi poruku da je obrada tekuće grupe podataka završena istog trenutno počinje sa prenosom novih podataka. Na ovaj način izbegnuto je čekanje programa jezgra na nove podatke čime je omogućeno da algoritmi obrade stalno budu aktivni.

3.4.4 Primer razmene podataka (međusobne komunikacije)

U ovom delu u vidu pseudokoda opisan je primer komunikacije između upravljačkog podsistema i podsistema za paralelnu obradu digitalnog signala. Funkcije koje su pozivane na strani upravljačkog podsistema su označene sa prefiksom “x86”, dok su funkcije koje su pozivane na strani jezgara označene sa prefiksom “DSP CORE”.

Pseudo kod:

1. DSP CORE: *čekaj poruku od upravljačkog programa*
2. x86: *učitaj podatke u deljenu memoriju DSP*
3. x86: *pošalji poruku konkretnom jezgru*
4. x86: *čekaj poruku od odgovarajućeg jezgra*
5. DSP CORE: *pročitaj novu poruku*
6. DSP CORE: *počni obradu podataka*
7. DSP CORE: *obrada podataka završena*
8. DSP CORE: *pošalji poruku upravljačkom programu, vrati se na 1*
9. x86: *pročitaj novu poruku, vrati se na 2*

Slika 3.9 – Ponašanje PQM sistema sa implementiranim *ping-pong* mehanizmom

4. Programsko rešenje

Upravljački program, kao deo upravljačkog podsistema, je implementiran pod Linux OS korišćenjem programskog jezika C. U skladu sa OS, programsko rešenje je prevedeno korišćenjem GNU C prevodioca (eng. *GNU C Compiler, GCC*) za x86 arhitekturu. Na isti način, implementirani su i programski moduli koji čine podsistem za paralelnu obradu digitalnog signala. U ovom slučaju korišćen je GCC prevodilac za odgovarajuću Texas Instruments (TI) C66 DSP arhitekturu koja je korišćena kao fizička osnova podsistema za digitalnu obradu signala. Pomenuta TI C66 DSP arhitektura je detaljnije opisana u odgovarajućem odeljku poglavlja Teorijske osnove.

U tabeli 1 je priložen spisak svih programskih modula korišćenih pri implementaciji programskog rešenja. Uz svaki programski modul dat je kratak opis njegove funkcionalnosti. Kako su neki od modula programskog rešenja priloženi od strane proizvođača procesora za digitalnu obradu signala (Texas Instruments), u nastavku poglavlja opisan je samo najvažniji deo programske sprege (eng. *Application Interface, API*) modula koji omogućuju rad PQM sistema. Više informacija o programskom modulu SYS/BIOS je takođe dostupno u drugom poglavlju.

Naziv modula	Datoteke modula	Kratak opis modula
MailBox	– mailBox.c – mailBox.h	Modul koji predstavlja strukturu poštanskog sandučeta koje se koristi za razmenu poruka.
PCIEDriver	– pcieDriver.c – pcieDriver.h	Modul PCIe rukovaoca koji omogućava komunikaciju podsistema na fizičkom nivou.

MemoryDriver	– memoryDirver.c – memoryDirver.h	Modul memorijskog rukovaoca koji omogućava prevazilaženje problema maksimalne veličine memorijskog bloka na Linux OS.
TaskScheduler	– scheduler.c – scheduler.h	Modul raspoređivača zadataka na jezgra DSP koji implementira redom u krug (RR) algoritam planiranja.
CoreController	– coreController.c – coreController.h	Modul upravljačkog programa koji upravlja podsistemom za paralelnu obradu digitalnog signala.
CoreApplication	– coreApplication.c – coreApplication.h	Modul programa jezgra koji enkapsulira sve algoritme obrade i komunikacione strukture u jedinstveni program koji se izvršava na jezgru DSP.
ProcessingAlgorithm	– processingAlgorithm.c – processingAlgorithm.h	Modul algoritma obrade koji se izvršava nad ulaznim podacima.
SYS/BIOS	–	Operativni sistem za rad u realnom vremenu.

Tabela 4.1 – Spisak modula programskog rešenja

4.1 Modul MailBox

Najvažnije funkcije API *MailBox* modula:

- int mailBoxCreate (void *, int , unsigned short, mailBoxConfig *)
- int mailBoxDelete (void *)
- int mailboxQuery (void *)
- int mailBoxRead (void *, char *, unsigned int *, unsigned int *)
- int mailBoxWrite (void *, char *, unsigned int *, unsigned int *)

4.1.1 Funkcija mailBoxCreate

Opis: Pravi i inicijalizuje strukturu poštanskog sandučeta.

Prototip: int mailBoxCreate (void *mailBoxHandle, int nodeId, unsigned short direction, mailBoxConfig *config)

Ulazni argumenti:

1. mailBoxHandle – pokazivač na napravljeno poštansko sanduče
2. nodeId – oznaka jezgra za čiju komunikaciju se pravi poštansko sanduče
3. direction – smer sandučeta (od ili ka jezgru)
4. config – pokazivač na strukturu sa parametrima inicijalizacije

Povratna vrednost: kod greške

4.1.2 Funkcija mailBoxDelete

Opis: Briše poštansko sanduče iz sistema i oslobađa zauzetu memoriju.

Prototip: int mailBoxDelete (void *mailBoxHandle)

Ulazni argumenti:

1. mailBoxHandle – pokazivač na napravljeno poštansko sanduče

Povratna vrednost: kod greške

4.1.3 Funkcija mailBoxQuery

Opis: Proverava da li se u sandučetu nalazi nepročitana poruka.

Prototip: int mailboxQuery (void *mailBoxHandle)

Ulazni argumenti:

1. mailBoxHandle – pokazivač na poštansko sanduče

Povratna vrednost: kod greške

4.1.4 Funkcija mailBoxRead

Opis: Čita poruku iz poštanskog sandučeta.

Prototip: int mailBoxRead (void *mailBoxHandle, char *buf, unsigned int *size, unsigned int *transId)

Ulazni argumenti:

1. mailBoxHandle – pokazivač na poštansko sanduče
2. buf – pokazivač na memoriju u koju se upisuje sadržaj primljene poruke
3. size – veličina poruke
4. transId – oznaka poruke

Povratna vrednost: kod greške

4.1.5 Funkcija mailBoxWrite

Opis: Čita poruku iz poštanskog sandučeta.

Prototip: int mailBoxWrite (void *mailBoxHandle, char *buf, unsigned int *size, unsigned int *transId)

Ulazni argumenti:

1. mailBoxHandle – pokazivač na poštansko sanduče
2. buf – pokazivač na memoriju u kojoj se nalazi sadržaj poruke za slanje
3. size – veličina poruke
4. transId – oznaka poruke

Povratna vrednost: kod greške

4.2 Modul PCIEDriver

Najvažnije funkcije API *PCIEDriver* modula:

- int pcieDriverOpen (pcieDriverOpenConfig *)
- int pcieDriverClose (void)
- int pcieDriverDspWrite (int, unsigned int, char *, int)
- int pcieDriverDspRead (int, unsigned int, char *, int)
- int pcieDriverDmaWriteStart (int, unsigned int, cmemHostFrameDesc *, int, int *)
- int pcieDriverDmaReadStart (int, unsigned int, cmemHostFrameDesc *, int, int *)
- int pcieDriverDmaCheck (int, int)

4.2.1 Funkcija pcieDriverOpen

Opis: Otvara i inicijalizuje modul PCIe rukovaoca u skladu sa prosledenim parametrima.

Prototip: int pcieDriverOpen (pcieDriverOpenConfig *config)

Ulazni argumenti:

1. config – inicijalizacioni parametri

Povratna vrednost: kod greške

4.2.2 Funkcija pcieDriverClose

Opis: Zatvara modul PCIe rukovaoca i oslobađa zauzetu memoriju.

Prototip: int pcieDriverClose (void)

Ulazni argumenti: nema

Povratna vrednost: kod greške

4.2.3 Funkcija **pcieDriverDspWrite**

Opis: Funkcija piše podatke u memoriju DSP upotrebom C *memcpy* funkcije.

Prototip: `int pcieDriverDspWrite (int dspId, unsigned int addr, char *buf, int size)`

Ulazni argumenti:

1. `dspId` – oznaka DSP u čiju memoriju se upisuju podaci
2. `addr` – adresa na koju se prenose podaci
3. `buf` – pokazivač na lokalne podatke koji se kopiraju u memoriju DSP
4. `size` – veličina podataka

Povratna vrednost: kod greške

4.2.4 Funkcija **pcieDriverDspRead**

Opis: Funkcija čita podatke iz memorije DSP upotrebom C *memcpy* funkcije.

Prototip: `int pcieDriverDspRead (int dspId, unsigned int addr, char *buf, int size)`

Ulazni argumenti:

1. `dspId` – oznaka DSP iz čije memoriju se čitaju podaci
2. `addr` – adresa sa koje se čitaju podaci
3. `buf` – pokazivač na lokalnu memoriju u koju se kopiraju podaci iz memorije DSP
4. `size` – veličina podataka

Povratna vrednost: kod greške

4.2.5 Funkcija **pcieDriverDmaWriteStart**

Opis: Funkcija pokreće prenos podataka u memoriju DSP upotrebom DMA prenosa.

Prototip: `int pcieDriverDmaWriteStart (int dspId, unsigned int addr, cmemHostFrameDesc *frameDesc, int flag, int *transId)`

Ulazni argumenti:

1. `dspId` – oznaka DSP u čiju memoriju se upisuju podaci
2. `addr` – adresa na koju se prenose podaci
3. `frameDesc` – struktura koja sadrži podatke o memoriji koja je zauzeta od strane memorijskog rukovaoca, a iz koje se prenose podaci u memoriju DSP
4. `transId` – oznaka prenosa

Povratna vrednost: kod greške

4.2.6 Funkcija **pcieDriverDmaReadStart**

Opis: Funkcija pokreće prenos podataka iz memorije DSP upotrebom DMA prenosa.

Prototip: `int pcieDriverDmaReadStart (int dspId, unsigned int addr, cmemHostFrameDesc *frameDesc, int flag, int *transId)`

Ulazni argumenti:

1. `dspId` – oznaka DSP iz čije memorije se čitaju podaci
2. `addr` – adresa sa koje se čitaju podaci
3. `frameDesc` – struktura koja sadrži podatke o memoriji koja je zauzeta od strane memorijskog rukovaoca, a u koju se prenose podaci iz memorije DSP
4. `transId` – oznaka prenosa

Povratna vrednost: kod greške

4.2.7 Funkcija **pcieDriverDmaCheck**

Opis: Proverava da li je DMA prenos završen.

Prototip: `int pcieDriverDmaCheck (int dspId, int transId)`

Ulazni argumenti:

1. `dspId` – oznaka DSP u kojem se odvija DMA prenos
2. `transId` – oznaka prenosa

Povratna vrednost: kod greške

4.3 Modul **MemoryDriver**

Najvažnije funkcije API *MemoryDriver* modula:

- `int MemoryDriverOpen (void)`
- `int MemoryDriverClose (void)`
- `int MemoryDriverAllocate (unsigned int, unsigned int, short, cmemHostBufDesc *)`
- `int MemoryDriverFree (unsigned int, unsigned int, short, cmemHostBufDesc *)`

4.3.1 Funkcija **MemoryDriverOpen**

Opis: Otvara modul memorijskog rukovaoca.

Prototip: `int MemoryDriverOpen (void)`

Ulazni argumenti: nema

Povratna vrednost: kod greške

4.3.2 Funkcija MemoryDriverClose

Opis: Zatvara modul memorijskog rukovaoca.

Prototip: int MemoryDriverClose (void)

Ulazni argumenti: nema

Povratna vrednost: kod greške

4.3.3 Funkcija MemoryDriverAllocate

Opis: Zauzima neprekidni (kontinualni) blok memorije unutar upravljačkog podsistema.

Prototip: int MemoryDriverAllocate (unsigned int numOfBuf, unsigned int sizeOfBuf, short type, cmemHostBufDesc *bufDesc)

Ulazni argumenti:

1. numOfBuf – broj memorijskih blokova
2. sizeOfBuf – veličina memorijskog bloka
3. type – tip memorije (statička ili dinamička)
4. bufDesc – struktura u koju se upisuju podaci o zauzetoj memoriji

Povratna vrednost: kod greške

4.3.4 Funkcija MemoryDriverFree

Opis: Oslobađa prethodno zauzeti neprekidni (kontinualni) blok memorije.

Prototip: int MemoryDriverFree (unsigned int numOfBuf, unsigned int sizeOfBuf, short type, cmemHostBufDesc *bufDesc)

Ulazni argumenti:

1. numOfBuf – broj memorijskih blokova
2. sizeOfBuf – veličina memorijskog bloka
3. type – tip memorije (statička ili dinamička)
4. bufDesc – struktura u kojoj se nalaze podaci o zauzetoj memoriji

Povratna vrednost: kod greške

4.4 Modul TaskScheduler

Modul *TaskScheduler* je implementiran upotrebom sistemskih niti Linux OS. Modul *TaskScheduler* je implementiran na ovaj način kao posledica potrebe da bude uvek u pripravnosti i spreman da uposli jezgro koje je završilo sa obradom.

Najvažnije funkcije API *TaskScheduler* modula:

- `int startTaskScheduler (void *)`

4.4.1 Funkcija startTaskScheduler

Opis: Implementira funkciju raspoređivanja zadataka.

Prototip: `int startTaskScheduler (void *args)`

Ulazni argumenti:

1. `args` – inicijalizaciona struktura sa ulaznim parametrima

Povratna vrednost: kod greške

4.4.2 Struktura TaskSchedulerConfiguration

Struktura *TaskSchedulerConfiguration* sadrži sve ulazne parametre koji se koriste za podešavanje i rad raspoređivača zadataka.

Polja strukture:

- `unsigned char dspChipNum`
- `unsigned char coreNumPerDspChip`
- `SchedulingAlgorithm schedulingAlgorithm`
- `int taskSchedulerThreadPriority`
- `void **coreControllerThreadConfigs`

4.4.2.1 Polje dspChipNum

Polje *dspChipNum* sadrži broj koji označava koliko DSP učestvuje u paralelnoj obradi. U trenutnom rešenju PQM sistema, podsistem za paralelnu obradu čine 4 DSP sa po 8 jezgara.

4.4.2.2 Polje coreNumPerDspChip

Broj jezgara koji su dostupni na DSP je predstavljen *coreNumPerDspChip* poljem. Za ispravan rad raspoređivača zadataka, tj. PQM sistema, podsistem za digitalnu obradu podataka mora biti sačinjen isključivo od DSP koji imaju simetričan i jednak broj jezgara.

4.4.2.3 Polje `schedulingAlgorithm`

Polje *schedulingAlgorithm* predstavlja polje koje određuje koji algoritam raspoređivanja zadataka je u upotrebi. Trenutno je podržan samo RR algoritam raspoređivanja.

4.4.2.4 Polje `taskSchedulerThreadPriority`

Kako je funkcionalnost raspoređivača zadataka kao i upravljačkih niti jezgara implementirana kao posebana nit na Linux OS, potrebno je definisati odgovarajući prioritet za svaku nit. Konkretno, *taskSchedulerThreadPriority* polje definiše prioritet niti programskog raspoređivača. Obično je prioritet izvršavanja niti raspoređivača veći od prioriteta izvršavanja niti koje upravljaju algoritmima obrade.

4.4.2.5 Polje `coreControllerThreadConfigs`

Polje *coreControllerThreadConfigs* predstavlja pokazivač na niz struktura u kojima se nalaze informacije o upravljačkim nitima jezgara koje upravljaju algoritmima obrade implementiranim na jezgrima DSP.

4.5 Modul `CoreController`

Modul *CoreController* predstavlja upravljački program koji je jedan od najvažnijih modula upravljačkog podsistema i služi za upravljanje algoritmima obrade koji se nalaze na jezgrima DSP. Funkcionalnost ovog programskog modula je implementirana upotrebom sistemskih niti koje su obezbeđene od strane Linux OS. Naime, funkcionalnost upravljanja jezgrima, sa stanovišta upravljačkog programa, je ista za sva jezgra. Ono što se razlikuje u zavisnosti od konkretnog jezgra jesu poštanski sandučići i memorijski prostor koji je dodeljen od strane rukovaoca memorijom. Zbog ove činjenice da je funkcionalnost ista a da se samo lokalni parametri razlikuju, uz potrebu da se upravlja sa više jezgara istovremeno, za realizaciju su odabrane niti Linux OS.

Najvažnije funkcije API *CoreController* modula:

- `int coreControllerFunction (void *args)`
- `int initCoreController (unsigned int)`
- `int preprocessImageFrame (void *buff, unsigned int size)`
- `int sendDataToDspThroughDma (int, unsigned int)`
- `int getDataThroughDma (int , unsigned int)`

4.5.1 Funkcija **coreControllerFunction**

Opis: Funkcija niti koja služi za upravljanje algoritmima obrade na jezgrima.

Prototip funkcije: `int coreControllerFunction (void *args)`

Ulazni argumenti:

1. `args` – pokazivač na ulazni parametar (oznaka jezgra kojim upravlja)

Povratna vrednost: kod greške

4.5.2 Funkcija **initCoreController**

Opis: Omogućuje inicijalizaciju lokalnih struktura u zavisnosti od oznake jezgra.

Prototip funkcije: `int initCoreController (unsigned int coreId)`

Ulazni argumenti:

1. `coreId` –oznaka jezgra za koje treba da inicijalizuje lokalne strukture

Povratna vrednost: kod greške

4.5.3 Funkcija **preprocessImageFrame**

Opis: Funkcija obavlja predprocesiranje okvira slike, tj. izdvaja samo Y komponentu.

Prototip funkcije: `int preprocessImageFrame (void *buff, unsigned int size)`

Ulazni argumenti:

1. `buff` – memorija u kojoj se nalaze podaci za predprocesiranje
2. `size` – veličina memorije

Povratna vrednost: kod greške

4.5.4 Funkcija **sendDataToDspThroughDma**

Opis: Upotrebom DMA mehanizma prenosi podatke za obradu iz memorije računara u deljenu memoriju DSP.

Prototip funkcije: `int sendDataToDspThroughDma (int nodeId, unsigned int transId)`

Ulazni argumenti:

1. `nodeId` – oznaka jezgra koja identifikuje rezultat obrade
2. `transId` – oznaka transakcije

Povratna vrednost: kod greške

4.5.5 Funkcija `getDataThroughDma`

Opis: Na osnovu podataka dobijenih preko poštanskog sandučeta upotrebom DMA mehanizma omogućuje preuzimanja rezultata obrade.

Prototip funkcije: `int getDataThroughDma (int nodeId, unsigned int transId)`

Ulazni argumenti:

1. `nodeId` – oznaka jezgra koja identifikuje rezultat obrade
2. `transId` – oznaka transakcije

Povratna vrednost: kod greške

4.6 Modul `CoreApplication`

Modul *CoreApplication* je modul koji čini deo podsistema za paralelnu obradu digitalnog signala koji obuhvata sve algoritme obrade i njihove poštanske sandučice u jedan jedinstveni program koji se izvršava na jezgru DSP. Modul *CoreApplication*, odnosno modul programa jezgra je detaljnije opisan u trećem poglavlju (3.3.3). Sa stanovišta programske implementacije, *CoreApplication* je implementiran upotrebom specifične vrste sistemskih niti koji su dostupni od strane SYS/BIOS OS – *SYS/BIOS Task*. Ova pomenuta programska struktura odgovara strukturi sistemskih niti koja postoji na Linux OS.

Najvažnije funkcije API *CoreApplication* modula:

- `Void coreApplicationProcessing (UArg, UArg)`
- `int initCoreApplication (unsigned int)`

4.6.1 Funkcija `coreApplicationProcessing`

Opis: Funkcija u kojoj je implementirana obrada na konkretnom jezgru.

Prototip: `Void coreApplicationProcessing (UArg a0, UArg a1)`

Ulazni argumenti:

1. `a0` – prvi argument funkcije (za buduću upotrebu)
2. `a1` – drugi argument funkcije (za buduću upotrebu)

Povratna vrednost: nema

4.6.2 Funkcija `MemoryDriverFree`

Opis: Inicijalizuje lokalne strukture i otvara poštanske sandučice za komunikaciju sa upravljačkim podsistemom.

Prototip: `int initCoreApplication (unsigned int coreId)`

Ulazni argumenti:

1. `coreId` – oznaka jezgra

Povratna vrednost: kod greške

4.7 Modul *ProcessingAlgorithm*

U *ProcessingAlgorithm* modulu je opisan API algoritma za detekciju crne slike jer je to konkretni algoritam obrade koji se koristi u trenutnoj verziji PQM sistema. API *Black Screen Detection* algoritma:

- `int *blackScreenReserveMemory (void)`
- `void blackScreenProcessingInit (unsigned int)`
- `blackScreenIteration (char *, char *, unsigned int *, unsigned short, unsigned short)`
- `blackScreenGetMax (unsigned int)`

4.7.1 Funkcija *blackScreenReserveMemory*

Opis: Na osnovu unapred definisanih podataka dinamički zauzima memoriju koja se koristi za izračunavanje histograma.

Prototip funkcije: `unsigned int *blackScreenReserveMemory (void)`

Ulazni argumenti: nema

Povratna vrednost: pokazivač na zauzetu memoriju

4.7.2 Funkcija *blackScreenProcessingInit*

Opis: Inicijalizuje histogram na početne vrednosti.

Prototip funkcije: `void blackScreenProcessingInit (unsigned int * histogram)`

Ulazni argumenti:

1. `histogram` – pokazivač na memoriju u kojoj se nalazi histogram

Povratna vrednost: nema

4.7.3 Funkcija *blackScreenIteration*

Opis: Implementira funkciju izračunavanja histograma, proveru maksimalne vrednosti histograma kao i popunjavanje poruke poštanskog sandučeta preko koje se šalju podaci o rezultatu obrade.

Prototip funkcije: void blackScreenIteration (char *inBuf, char * msg, unsigned int * histogram, unsigned short frameHeight, unsigned short frameWidth)

Ulazni argumenti:

1. inBuf – pokazivač na podatke za koje treba izračunati histogram
2. msg – poruka u koju se upisuje rezultat obrade
3. histogram – pokazivač na histogram
4. frameHeight – visina trenutno obrađivanog okvira (slike)
5. frameWidth – širina trenutno obrađivanog okvira (slike)

Povratna vrednost: nema

4.7.4 Funkcija blackScreenGetMax

Opis: Na osnovu histograma izračunava maksimalnu vrednost boje koja se nalazi u obrađenom okviru.

Prototip funkcije: unsigned int blackScreenGetMax (unsigned int * histogram)

Ulazni argumenti:

1. histogram – pokazivač na histogram

Povratna vrednost: maksimalna vrednost histograma

5. Testiranje

Provera funkcionalnosti opisane verzije PQM sistema nije temeljno testirana i verifikovana upotrebom nekih od alata za funkcionalno testiranje kao što je na primer CppUnit ili JUnit. Iako nisu korišćeni neki od pomenutih alata za testiranje, funkcionalnost rada PQM sistema je proverena analizom i poređenjem dobijenih rezultata obrade svakog jezgra sa referentnim rezultatima obrade.

Kao što je u programskom rešenju pomenuto, svi moduli programskog rešenja su implementirani upotrebom programskog jezika C, što omogućava da se moduli koji nisu na određeni način vezani za specifični deo programske podrške (SYS/BIOS), mogu implementirati na sistemima opšte namene (PC). Imajući u vidu ovu činjenicu, u svrhu testiranja, modul algoritma obrade koji služi za otkrivanje crne slike (BSD algoritam) je implementiran na računaru opšte namene kao zaseban program, a dobijeni rezultati obrade ovog programa se koriste kao referentni u daljem procesu testiranja. Ono što treba napomenuti je to da se testiranje funkcionalnosti implementiranih algoritama ne testira u realnom vremenu. Na određenom jezgru pojedinačno se testira svaki algoritam obrade, a nakon njihove provere prelazi se na testiranje algoritama narednog jezgra. Cilj ovog tipa testiranja jeste proveriti da li algoritam obrade, tj. BSD algoritam, funkcionalno dobro radi a ne da li sistem ispunjava vremenski zahtevana ograničenja.

Postupak testiranja obuhvata sledeće korake:

1. Odabir testnog video materijala
2. Obrada odabranog testnog video materijala BSD algoritmom koji je implementiran na računaru opšte namene
3. Smeštanje rezultata obrade u posebnu referentnu datoteku

4. Obrada odabranog testnog video materijala BSD algoritmom koji je implementiran na konkretnom jezgri DSP
5. Smeštanje rezultata obrade u posebnu izlaznu datoteku
6. Poređenje dobijene izlazne datoteke rezultata sa referentnom datotekom rezultata
7. Ukoliko se sadržaji datoteka u potpunosti poklapaju BSD algoritam, odnosno PQM sistem ispravno funkcioniše, u suprotnom sistem ne funkcioniše dobro
8. Postupak od koraka 4 do koraka 7 ponoviti za svako jezgro koje čini podsistem za paralelnu obradu video signala

Prethodno opisani postupak testiranja nije najpouzdaniji način za proveru rada nekog sistema, međutim ukoliko se u obzir uzme činjenica da sistema za procenu kvaliteta video signala predstavlja složen sistem kojeg čine više različitih funkcionalnih modula koji su u međusobnom sadejstvu, kao i činjenicu da trenutna verzija sistema nije konačna, ovaj način provere se može smatrati zadovoljavajućim. Najveći nedostatak ovog postupka testiranja je to što se proverava samo funkcionalnost algoritama obrade a ne i funkcionalnost drugih modula.

6. Rezultati

U ovom poglavlju predstavljeni su dobijeni rezultati merenja koja su rađena na implementiranom PQM sistemu u slučajevima bez i sa implementiranim *ping-pong* (PP) mehanizmom. Rezultati merenja se odnose na praćenje vremena obrade, prosečnog vremena čekanja jezgra na nove podatke i ostvarenog ubrzanja (eng. *speedup*) kada u procesu paralelne obrade učestvuje različiti broj jezgara. Pored konkretno dobijenih rezultata, u ovom poglavlju je opisan proces i način merenja.

6.1 Opis merenja

Podsistem za paralelnu obradu digitalnog signala trenutne verzije PQM sistema, koja je korišćena u procesu merenja, čine ukupno 32 jezgra na kojima je implementiran BSD algoritam. Sva jezgra su grupisana u 4 posebna DSP koja se nalaze na PCIe ploči. Svako od 8 jezgara koje se nalazi na DSP radi na frekvenciji od 1.25 GHz. Takođe, svaki od DSP poseduje 1 GB deljene memorije koja je dostupna svim jezgrima konkretnog DSP. Više podataka o DSP i PCIe ploči je dostupno u drugom poglavlju.

Uzimajući u obzir broj jezgara dostupnih za obradu i zahteve da PQM sistem omogući rad u realnom vremenu, za referentnu količinu podataka koja je korišćena za merenje odabran je HD video materijal (1080p@60fps) u trajanju od 32 sekunde.

Kako podsistem za paralelnu obradu poseduje relativno veliki broj jezgara koji su dostupni za obradu, u cilju što boljeg praćenja performansi sistema, merenja su rađena za sledeći broj jezgara: 1, 2, 4, 8, 16 i 32 jezgra. Dobijeni rezultati obrade kada se u sistemu koristi samo jedno jezgro za obradu predstavljaju rezultate serijske obrade. Rezultati serijske obrade predstavljaju početne rezultate koji su korišćeni za praćenje vremena izvršavanja kao i ostvarenog ubrzanja.

Rezultati obrade koji su dobijeni kada u sistemu učestvuje više jezgara (2, 4, 8, 16 i 32) predstavljaju rezultate paralelne obrade. Naravno, sa stanovišta PQM sistema i PP mehanizma postoje dva tipa rezultata. Jedan par rezultata merenja (serijski i paralelni) je dobijen kada u PQM sistemu postoji implementiran PP mehanizam, dok su drugi rezultati merenja dobijeni kada PP mehanizam nije implementiran.

Kao rezultat merenja predstavljeno je i srednje vreme čekanja jezgara na nove podatke. U PQM sistemu srednje vreme čekanja predstavlja srednju vrednost vremena koja su jezgra provela u čekanju na novi okvir slike koji treba da bude obrađen BSD algoritmom. Ovo vreme se može posmatrati i kao vreme koje jezgro provede u nekorisnom (eng. *idle*) stanju. Kao i u slučaju prethodnih rezultata, i za srednje vreme čekanja jezgara postoje dva tipa merenja.

6.2 Dobijeni rezultati merenja

U ovom poglavlju predstavljani su konkretno dobijeni rezultati merenja koji se odnose na:

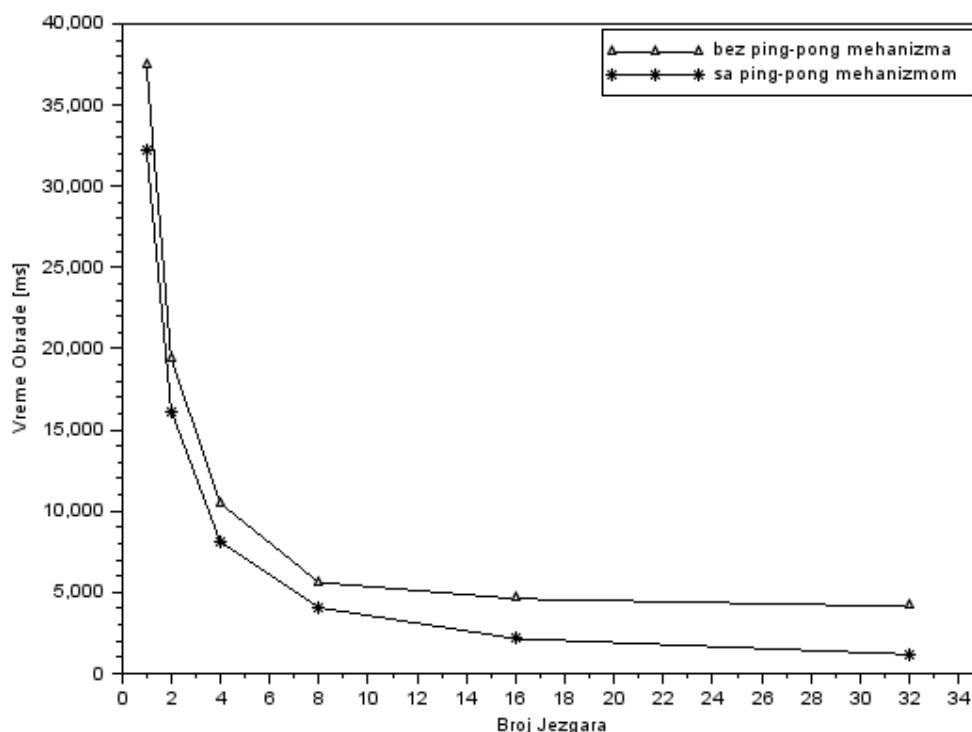
- vreme obrade
- prosečno vreme čekanja jezgra na podatke
- ostvareno ubrzanje

6.2.1 Rezultati vremena obrade

Sa T_s i T_s' označena su vremena serijske obrade kad u PQM sistemu nije, odnosno kad u PQM sistemu jeste implementiran PP mehanizam, respektivno. Kao što je prethodno pomenuto, rezultati serijske obrade se odnose na rezultate dobijene kada se u sistemu za paralelnu obradu koristi samo jedno jezgro. Na isti način kao u slučaju označavanja serijske obrade, sa T_p i T_p' označena su vremena paralelne obrade. Na slici 6.1 predstavljen je grafik rezultata paralelne obrade referentnog video materijala, dok su konkretne izmerene vrednosti date u tabeli 6.1. Vremena serijske obrade, tj. T_s i T_s' imaju sledeće vrednosti respektivno: 37393 i 32288 ms.

Broj jezgara	2	4	8	16	32
T_p [ms]	19375	10490	5613	4606	4130
T_p' [ms]	16156	8097	4057	2150	1182

Tabela 6.1 – Rezultati vremena paralelne obrade



Slika 6.1 – Grafik rezultata vremena paralelne obrade

Na osnovu priloženih rezultata može se primetiti značajan uticaj PP mehanizma na ostvarena vremena obrade. Upoređivanjem dobijenih rezultata vremena paralelne obrade kada u sistemu nije implementiran PP mehanizam sa dobijenim rezultatima paralelne obrade kada on jeste implementiran, vidi se da je primena PP mehanizma znatno uticala na dobijanje osetno boljih rezultata vremena obrade i to u svim slučajevima paralelne obrade. Ovo je neposredna posledica smanjenja vremena čekanja jezgra na nove podatke za obradu koje se javlja u slučaju kada PP mehanizam nije implementiran. Primena PP mehanizma daje dobre rezultate i u slučaju serijske obrade, što se može potvrditi upoređivanjem vremena T_s (37393) i T_s' (32288).

Sa stanovišta ostvarenih rezultata vremena obrade uticaj povećanja broja jezgara koja učestvuju u paralelnoj obradi je očigledan, ali ipak je potrebno posebno posmatrati rezultate u kontekstu da li PP mehanizam jeste ili nije implementiran. Naime, na osnovu rezultata bez implementiranog PP mehanizma može se videti da sa povećanjem broja jezgara vreme obrade stalno opada, ali vrednosti za koje se smanjuje znatno varira. Kada u sistemu postoje 2, 4 i 8 jezgara vremena obrade se skoro linearno smanjuju. U slučajevima kada u paralelnoj obradi učestvuju 16 i 32 jezgra vreme obrade se i dalje smanjuje ali ne linearno, tj. dolazi do zasićenja. Slično kao i u prethodnom slučaju, ovo zasićenje je posledica vremena čekanja koja jezgara provedu čekajući na nove podatke. Za razliku od rezultata kada u sistemu nije implementiran PP mehanizam, rezultati dobijeni sa implementiranim PP mehanizmom pokazuju da se vreme

obrade referentnog video materijala, sa stalnim povećanjem broja jezgara koja učestvuju u paralelnoj obradi, skoro linearno smanjuje. Za obe verzije sistema u obzir treba uzeti postojanje viška vremena (eng. *overhead*) koje se utroši na međusobnu komunikaciju.

6.2.2 Rezultati čekanja jezgara na nove podatke

U ovom odeljku predstavljeni su rezultati srednjeg vremena čekanja jezgara na nove podatke koji treba da se obrade. Ovi rezultati za cilj imaju da pokažu neposredan uticaj PP mehanizma na vremena obrade na taj način što se vreme čekanja jezgara na nove podatke znatno smanjuje, a u nekim slučajevima se u potpunosti izbegava.

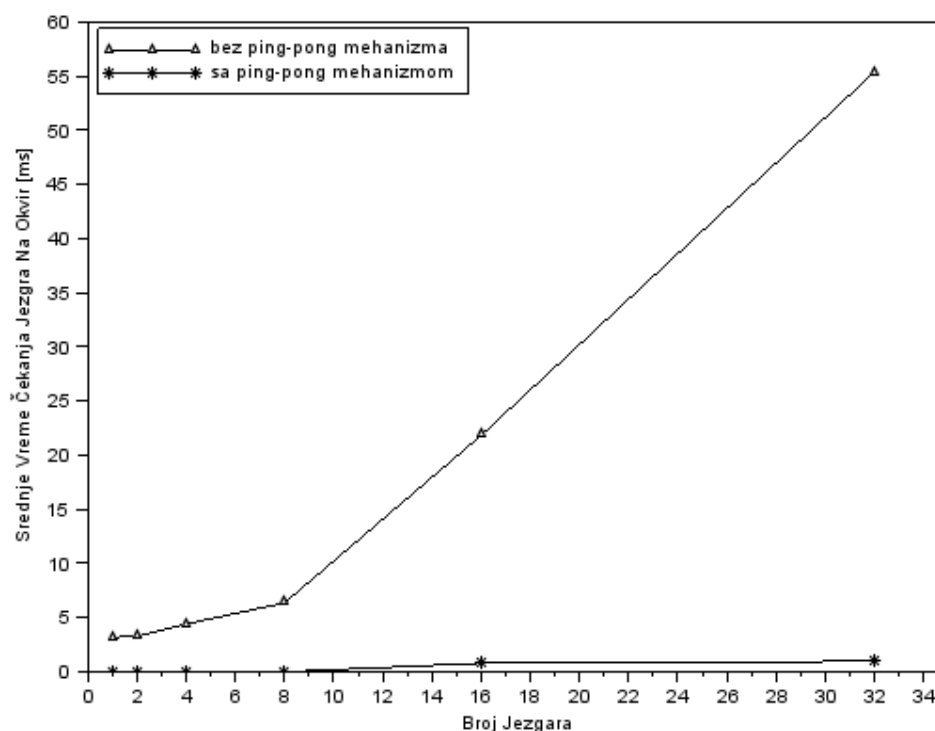
Srednje vreme čekanja jezgra na novi okvir u PQM sistemu bez i sa implementiranim PP mehanizmom su označena sa T_w i T_w' , respektivno. Na slici 6.2 predstavljen je grafik rezultata srednjeg vremena čekanja jezgara, dok su konkretne izmerene vrednosti date u tabeli 6.2.

Broj jezgara	1	2	3	4	5	6
T_w	3.19	3.29	4.42	6.36	21.84	55.39
T_w'	0	0	0	0	0.8	0.98

Tabela 6.2 – Rezultati srednjeg vremena čekanja jezgara na novi okvir

Poređenjem dobijenih rezultata srednjeg vremena čekanja još jednom je pokazan značajan uticaj PP mehanizma na performanse PQM sistema. U sistemu bez i sa implementiranim PP mehanizmom može se primetiti trend porasta vremena čekanja na novi video okvir sa porastom broja jezgara koja učestvuju u paralelnoj obradi. U međusobnom poređenju rezultata porast vremena čekanja jezgara u PQM sistemu sa implementiranim PP mehanizmom je zanemarljivo.

Sa aspekta složenosti PQM sistema, na vreme čekanja jezgara, pored vremena koje je potrebno da se određena količina podataka prenese, utiče način implementacije međusobne komunikacije kao i to da li u sistemu postoje zajednički deljeni resursi. U PQM sistemu, kao što je opisano u programskom rešenju, upravljački podsistem je implementiran na Linux OS upotrebom sistemskih niti. Kao posledica činjenice da u OS postoje prioritetnije sistemske niti koje je potrebno opslužiti, može se pojaviti određeno vreme kašnjenja dok nit upravljačkog podsistema ne postane ponovo aktivna i ne bude u mogućnosti da uposli konkretno jezgro koje je završilo sa obradom. Ovo ograničenje treba uzeti u obzir pogotovo kada u sistemu postoji veliki broj jezgara, jer je upravljački podsistem implementiran tako da svako jezgro ima svoju posebnu upravljačku nit.



Slika 6.2 – Grafik rezultata srednjeg vremena čekanja jezgara na novi okvir

6.2.3 Rezultati ostvarenog ubrzanja

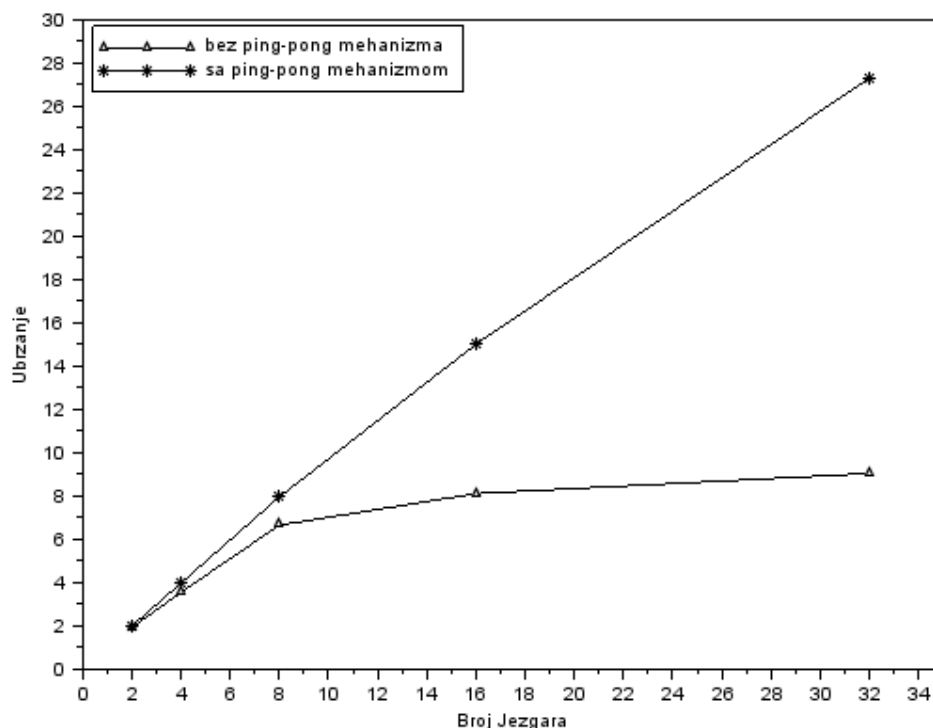
U ovom poglavlju predstavljeni su rezultati ostvarenog ubrzanja obrade referentnog video materijala sa stanovišta povećanja broja jezgara koji učestvuju u paralelnoj obradi. Ostvareno ubrzanje je označeno sa S i S' , gde S' predstavlja ostvareno ubrzanje obrade u sistemu sa implementiranim PP mehanizmom. Na slici 6.3 predstavljen je grafik rezultata ostvarenog ubrzanja, dok su konkretno dobijene vrednosti ubrzanja date u tabeli 6.3.

Broj jezgara	2	4	8	16	32
S	1.93	3.56	6.67	8.12	9.05
S'	2.0	3.99	7.96	15.02	27.32

Tabela 6.3 – Rezultati ostvarenog ubrzanja

Zaključak koji se može izvesti za ostvareno ubrzanje je veoma sličan zaključku koji se odnosi na rezultate vremena obrade koji su pretstavljani u odeljku 6.2.1 ovog poglavlja. Na osnovu rezultata vidi se da ostvareno ubrzanje sa povećanjem broja jezgara raste, sa tim da korak povećanja zavisi i od broja jezgara kao i od toga da li je u sistemu implementiran PP mehanizam.

Ako se posmatra sistem bez PP mehanizma, vidi se da ostvareno ubrzanje raste ali ono ne prati broj jezgara koji učestvuju u paralelnoj obradi, kao što bi to bilo očekivano. Iz ovoga se mogu izuzeti slučajevi za 2 i 4 jezgra gde je ostvareno zadovoljavajuće ubrzanje. U idealnom slučaju definiše se pojam linearnog ubrzanja (eng. *linear speedup*) po kojem bi sistem koji poseduje p jedinica za paralelnu obradu (procesora ili jezgara) trebao da ostvaruje ubrzanje p [9]. U sistemu bez implementiranog PP mehanizma za 32 jezgra ostvareno je ubrzanje od približno 9 puta, što je veoma skromno. Sa druge strane, ostvareno ubrzanje PQM sistema sa implementiranim PP mehanizmom je znatno bolje. Ukoliko se analizira trend rasta ostvarenog ubrzanja, može se primetiti da je ono u slučajevima za 2, 4 i 8 jezgara skoro linearno, što predstavlja skoro idealno ubrzanje. Međutim, u slučajevima za 16 i 32 jezgra situacija je nešto drugačija. Posledica ostvarivanja manjeg ubrzanja je prouzrokovano time što upravljačke niti koje su deo upravljačkog podsistema nisu u mogućnosti da trenutno uposle jezgra koja su završila obradu. Ovo je naročito izraženo u slučaju kada u sistemu postoje 32 jezgra za paralelnu obradu. Bez obzira na ovo ograničenje, primena PP mehanizma je u slučaju 2, 4, 8 i 16 jezgara omogućila postizanje približno linearnog ubrzanja, što predstavlja veoma dobre rezultate.



Slika 6.3 – Grafik rezultata ostvarenog ubrzanja

7. Zaključak

U ovom radu opisano je jedno od rešenja sistema za paralelnu obradu video signala u više kanala zasnovanog na fizičkoj arhitekturi sa više jezgara. Predstavljeni sistem je specifičan po tome što paralelnu obradu zasniva na DSP sa više jezgara koji su zadnjih nekoliko godina postali veoma zastupljeni u svim tipovima namenskih sistema. Na konkretno implementiranom sistemu analizirani su problemi i predstavljena su rešenja za uočene probleme i ograničenja koja se javljaju u sistemima koji svoj rad zasnivaju na DSP sa više jezgara.

Kao konkretni sistem za paralelnu obradu signala implementiran je PQM sistem. PQM sistem je razvijen na institutu RT-RK i predstavlja deo BBT sistema za automatsko funkcionalno testiranje digitalnih STB i TV uređaja, koji je takođe razvijen na institutu RT-RK. PQM sistem, upotrebom različitih algoritama za obradu slike, omogućuje analizu i procenu kvaliteta video signala sa stanovišta postojanja određenih artefakata. U opisanoj verziji PQM sistema, na svim jezgrima za paralelnu obradu korišćen je BSD algoritam. BSD algoritam otkrivanjem specifične slike u video signalu prijavljuje grešku u radu uređaja. Kao fizička osnova paralelne obrade korišćena je Advantech Lightning PCIe kartica koja poseduje 4 Texas Instruments C6678 DSP. Svaki C6678 DSP sadrži 8 jezgara visokih performansi, što znači da je za paralelnu obradu dostupno najviše 32 jezgra.

U cilju implementacije PQM sistema razvijena su dva podsistema, upravljački podsistem i podsistem za paralelnu obradu digitalnog signala. Svaki od podsistema sačinjen je od velikog broja programskih modula koji su posebno razvijeni kako bi se prilagodili zahtevima PQM sistema, kao i modula koji su dostavljeni od strane Texas Instruments kompanije čiji se DSP koristi. Upravljački podsistem služi za komunikaciju i upravljanje algoritmima obrade koji se izvršavaju na jezgrima DSP u cilju analize više video kanala. Podsistem za paralelnu obradu

omogućuje implementaciju različitih algoritama obrade na osnovu kojih PQM sistem može da vrši procenu kvaliteta.

Kao posledica distribuirane memorijske arhitekture PQM sistema, implementiran je *ping-pong* mehanizam pomoću koga se prevazilaze problemi i ograničenja koja su prouzrokovana specifičnom memorijskom arhitekturom.

Takođe, kao jedna od mera sistema paralelne obrade, analizirano je i ostvareno ubrzanje rada PQM sistema sa stanovišta povećanja broja jezgara koji učestvuju u paralelnoj obradi. Na osnovu ostvarenog ubrzanja može se zaključiti da *ping-pong* mehanizam predstavlja dobro rešenje za prevazilaženje ograničenja u sistemima sa distribuiranom memorijskom arhitekturom, kao i da je primena DSP sa više jezgara u sistemima za paralelnu obradu opravdana.

Predstavljeno rešenje jednog sistema za paralelnu obradu signala zasnovanog na DSP arhitekturi sa više jezgara, odnosno PQM sistema, može se upotrebiti kao dobra polazna osnova za pravljenje novog sistema koji treba da služi sličnoj svrsi. Naime, PQM sistem se odlikuje visokim stepenom modularnosti čime se na jednostavan način, i uz malo promena, može postići odgovarajuća funkcionalnost. Takođe, kao što se može vidite na osnovu rezultata ostvarenog ubrzanja, PQM sistem sa stanovišta povećanja broja jezgara poseduje dobru skalabilnost. Na ovaj način jednostavnim dodavanjem novih DSP (procesnih elemenata) u sistem paralelne obrade moguće je ostvariti željeno ubrzanje obrade.

Trenutna verzija PQM sistema veoma malo koristi periferije koje su dostupne od strane DSP, što predstavlja potencijalno ograničenje. Ovo se pre svega odnosi na iskorišćenje DMA kontrolera. DMA kontroler se trenutno koristi samo za prenos podataka između memorije upravljačkog podsistema (PC) u memoriju PCIe ploče, odnosno DSP, dok se kopiranje memorije unutar DSP ostvaruje upotrebom C *memcpy* funkcije. Trajanje prenosa podataka upotrebom C *memcpy* funkcije, u poređenju sa trajanjem prenosa iste količine podataka upotrebom DMA prenosa, traje znatno duže, što negativno utiče na performanse sistema. Takođe, u trenutnoj verziji PQM sistema za procenu kvaliteta video signala dostupan je samo jedan (BSD) algoritam za obradu slike. Rešavanje prethodno pomenutih ograničenja, zajedno sa implementiranjem novih algoritama obrade, predstavlja pravce daljeg razvoja opisanog PQM sistema.

Prethodno predstavljeni rezultati su u kraćem obliku dati u radu na međunarodnoj konferenciji ECBS-EERC 2013 [10].

8. Literatura

- [1] Dušica Marijan, Vladimir Zlokolica, Nikola Teslić, Vukota Peković, Tarkan Teckan, „Automatic Functional TV Set Failure Detection System”, IEEE Transactions on Consumer Electronics, Vol. 56, No. 1, February 2010
- [2] Texas Instruments, „TMS320C6678 Multicore Fixed and Floating Point Digital Signal Porcessor “, Literature Number: SPRS691D, April 2013
- [3] Texas Instruments, „TMS320C66xx CorePac “, Literature Number: SPRUGW0B, July 2011
- [4] Texas Instruments, „KeyStone Architecture Communications Infrastructure Digital Signal Processor, Enhanced Direct Memory Access (EDMA3) Controller “, Literature Number: SPRUGS5A, December 2011
- [5] Advantech, „Lightning (DSPC-8681E) User Guide “, Revision 0.7
- [6] Advantech, „Lightning DSPC-8681E Quad-TMS320C6678 DSP PCI-E HL Card H/W Manual“, Version 0.3
- [7] Texas Instruments, „TI SYS/BIOS v6.34 Real-time Operating System“, Literature Number: SPRUEX3L, Semptember 2012
- [8] Vladimir Kovačević, Miroslav Popović: „Sistemska programska podrška u realnom vremenu“, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2002
- [9] <http://en.wikipedia.org/wiki/Speedup>
- [10] Branislav Kordić, Vladimir Marinković, Miroslav Popović, Vukota Peković, „Parallel Processing of Multichannel Video Based on Multicore Architecture”, ECBS-EERC, Budapest, Hungary, August 29 - 30, 2013