UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
Odsek za elektrotehniku i računarstvo
Institut za računarstvo i automatiku
Katedra za računarsku tehniku i računarske komunikacije

Jedno rešenje programske podrške sistema za snimanje i reprodukciju digitalnog audio/video signala zasnovanog na TMS320DM642 DSP-u

Diplomski rad iz predmeta
Logičko projektovanje računarskih sistema

Mentor:
doc. dr. Nikola Teslić

Kandidat:
Jaroslav Hlavač, E7628

Novi Sad, Februar 2006.

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj: **RBR**	
Identifikacioni broj: **IBR**	
Tip dokumentacije: **TD**	Monografski rad
Tip zapisa: **TZ**	Štampa
Vrsta rada: **VR**	Diplomski rad
Autor: **A**	Jaroslav Hlavač
Mentor/Komentor: **MN**	doc. dr. Nikola Teslić
Naslov rada: **NR**	Jedno rešenje programske podrške sistema za snimanje i reprodukciju digitalnog audio/video signala zasnovanog na TMS320DM642 DSP-u
Jezik publikacije: **JP**	Srpski
Jezik izvoda: **JI**	Srpski
Zemlja publikovanja: **ZP**	Srbija i Crna Gora
Uže geografsko područje: **UGP**	Vojvodina
Godina: **GO**	2006

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Izdavač: **IZ**	Fakultet Tehničkih Nauka
Mesto i adresa: **MA**	21000 Novi Sad, Trg Dositeja Obradovića 6
Fizički opis rada: **FO**	Diplomski rad je realizovan u vidu programske podrške
Naučna oblast: **NO**	Elektrotehnika
Naučna disciplina: **ND**	Računarska tehnika
Predmetna odrednica/ključne reči **PO/UDK**	Računarstvo, televizija, logičko projektovanje
Čuva se: **ČU**	U biblioteci Fakulteta Tehničkih Nauka
Važna napomena: **VN**	
Izvod: **IA**	Razvijena je programska podrška za snimanje i reprodukciju digitalnog audio/video signala na sistemu koji se sastoji od PC računara sa WindowsXP OS i DM642 EVM PCI kartice.
Datum prihvatanja teme od strane NN veća: **DP**	
Datum odbrane: **DO**	06.03.2006
Članovi komisije: **KO** **Predsednik:** **Član:** **Član:**	 prof. dr. Miroslav Popović prof. dr. Branislav Atlagić doc. dr. Nikola Teslić

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES
KEY WORDS DOCUMENTATION

Accession number: **ANO**	
Identification number: **INO**	
Document type: **DT**	Monographic
Type of record: **TR**	Printed
Content code: **CC**	Graduate thesis
Author: **A**	Jaroslav Hlavač
Mentor: **MN**	Nikola Teslić, Ph D
Title: **TI**	One solution of software on system for receiving and transmiting digital audio/video signals based on the TMS320DM642 DSP
Language of text: **LT**	Serbian
Language of abstract: **LA**	Serbian
Country of publication: **CP**	Serbia and Montenegro
Locality of publication: **LP**	Vojvodina
Publication year: **PY**	2006

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES
KEY WORDS DOCUMENTATION

Publisher: **PU**	Faculty of Technical Sciences
Publication place: **PP**	21000 Novi Sad, Trg Dositeja Obradovića 6
Physical description: **PD**	
Scientific field: **SF**	Electrical Engineering
Scientific discipline: **SD**	Computer engineering
Subject/Key words: **SKW** **UC**	Computer engineering, television, digital audio and video communications
Holding data: **HD**	The Library of the Faculty of Technical Sciences
Note: **N**	
Abstract: **A**	Software for receiving and transmiting digital audio/video signals on the system that consist of PC with WindowsXP OS and DM642 EVM PCI kard.
Accepted by the Scientific Board on: **ASB**	
Defended on: **DE**	06.03.2006
Thesis defend board: **DB** **President:** **Member:** **Member:**	 Miroslav Popović, Ph D Branislav Atlagić, Ph D Nikola Teslić, Ph D

SKRAĆENICE

ADC - *Analog to Digital Converter*, Analogno/digitalni konvertor
AIC23B - *Stereo Codec for Input and Output Audio Signals*, Stereo I/O audio kodek
ALU - *Arithmetic-Logical Unit,* Aritmetičko-Logička Jedinica (ALJ)
API - *Application Programming Interface,* Aplikativna programska sprega
CCS - *Code Composer Studio*
CODEC - *Coder/Decoder*, Koder/Dekoder – KODEK
CPU - *Central Processing Unit,* Centralna procesna jedinica (CPJ)
CSL - *Chip Support Library*
DAC - *Digital to Analog Converter*, Digitalno/analogni konvertor
DDK - *Device Driver Kit,* Programski alat za pravljenje rukovaoca
DIT - *Digital Audio Interface,* Digitalna audio sprega
DLL - *Dynamic Link Library,* Biblioteka koja se dinamički povezuje
DM642 - *Digital Media DSP,* DSP za digitalnu audio/video obradu
DM642D - *DM642 EVM Driver,* Rukovalac za DM642 EVM
DMA - *Direct Memory Access,* Direktan pristup memoriji
DPC - *Deferred Procedure Call,* Procedura sa odloženim pozivom
DRAM - *Dynamic RAM,* Dinamička RAM memorija
DSP - *Digital Signal Processor*, Procesor za digitalnu obradu signala
DVS - *Digital Video Sistem,* Video format zapisa
EAV - *End of Active Video,* Kraj aktivnog video signala
EDMA - *Enhanced DMA,* Povećana DMA
EMIF - *External Memory Interface*, Sprega sa eksternom memorijom
EPROM - *Erasable Programmable ROM,* Izbrisiva programabilna ROM memorija
EVM - *Evaluation Module,* Platforma za evaluaciju, razvoj i testiranje
FIFO - *First In First Out*, Red čekanja (bafer) gde prvi element koji je ušao mora prvi i izaći
FPGA - *Field Programmable Gate Array,* Lokalno programabilna sekvencijalna mreža
HAL - *Hardware Abstraction Layer*
HDTV - *High Definition Television*, Televizija visoke rezolucije
HWI - *Hardware Interrupt*
I/O - *Input/Output,* Ulaz/Izlaz (U/I)
IBM - *International Business Machines,* Međunarodne poslovne mašine
IDE - *Integrated Development Enviroment,* Integrisano razvojno okruženje
I^2C - *Inter-Integrated Circuit,* Magistrala za kontrolu periferija
I^2S - *Inter-IC Sound, or Integrated Interchip Sound,* Sprega za povezivanje digitalnih audio uređaja
IRP - *I/O Request Packet,* Paket za U/I zahtev
IRQ - *Interrupt Request,* Zahtev za prekid
ISR - *Interrupt Service Routine,* Rutina za obradu prekida
ISRAM - *Internal SRAM*, Interna statička RAM memorija (unutar DSP)
ITU - *The International Telecommunications Union*
JTAG - *Joint Test Action Group*
LED - *Light Emitting Diode,* Svetleća dioda
McASP - *Multichannel Audio Serial Port*, Višekanalni serijski audio priključak
MFC - *Microsoft Foundation Classes,* Osnovne Microsoft-ove klase

MIPS - *Million Instructions Per Second,* Milion instrukcija u sekundi
MS-DOS - *Microsoft Disk Operating System,* Operativni sistem na masovnoj memoriji zasnovan na znakovnim naredbama (konzola)
NTSC - *National Television System(s) Committee,* Nacionalna Komisija za Televizijske Sisteme
OSD - *On Screen Display,* Prikaz na ekran
PAL - *Phase Alternation Line,* Linija izmenjive faze
PC - *Personal Computer,* Personalni računar
PCI - *Periferal Component Interconect*, Periferijska magistrala
PCM - *Pulse Code Modulation,* Modulacija kodiranja impulsa
RAM - *Random Access Memory*, Memorija sa slučajnim pristupom
RCA - *Radio Corporation of America*
RGB - *Red-Green-Blue output*, Crveni-Zeleni-Plavi signal na izlazu
ROM - *Read-Only Memory,* Memorija samo za čitanje
SAA7105 - *Video Encoder*, Video koder
SAV - *Start of Active Video,* Početak aktivnog video signala
SDRAM - *Synchronous DRAM*, Sinhrona dinamička RAM memorija
SRAM - *Static RAM,* Statička RAM memorija
SWI - *Software Interrupt*
TDM - *Time-Division Multiplexed,* Vremenski raspodeljene
TI - *Texas Instruments Inc.,* Firma Texas Instruments
TVP5150A - *Video Decoder*, Video dekoder
UART - *Universal Asynchronous Receiver/Transmitter,* Univerzalni asinhroni primopredajnik
VP - *Video Port,* Video priključak
WAV - *WAVE Form Audio Format,* Audio format zapisa
WDM - *Windows Driver Model,* Windows specifikacija za pisanje rukovalaca

SADRŽAJ

1. Uvod

Sa porastom upotrebe računara raste potreba i korišćenje digitalne obrade signala. Digitalna obrada signala (DSP) je nauka koja se bavi signalima u digitalnoj reprezentaciji i metodima njihove obrade. Algoritmi koji se koriste kod digitalne obrade signala se često implementiraju na specijalizovanim mikroprocesorima koji se zovu "procesori za digitalnu obradu signala" (takođe DSP). Kod njih se obrada signala izvršava u realnom vremenu.

Da bi se analogni signal mogao obraditi na računaru mora biti digitalizovan pomoću analogno/digitalnog pretvarača (ADC). Ovo pretvaranje se još zove odabiranje (*sampling*) i radi se u dva koraka. Prvi se zove „diskretizacija" gde se izvorni signal pretvara u seriju diskretnih vrednosti. Drugi se zove „kvantizacija" gde se svaki odabirak (*sample*) pretvara u neki broj. Signal se pretvara natrag u analogni pomoću digitalno/analognog pretvarača (DAC).

Među glavne kategorije digitalne obrade signala spadaju obrada digitalnih audio signala i obrada digitalnih video signala.

Prvi PC računari su pristupali periferijskim uređajima direktno a periferije su morale da se konfigurišu ručno pomoću prekidača. Kasnije je MS-DOS operativni sistem objedinio šemu upravljanja periferijama, tako što je uveo datoteku CONFIG.SYS preko koje je operativni sistem mogao učitati sve potrebne rukovaoce. Sa pojavom paralelne obrade zadataka i višekorisničkog sistema, Microsoft je napravio operativni sistem zasnovan na virtuelnim mašinama. Svaka aplikacija se izvršavala na vlastitoj virtuelnoj mašini ali je svaka aplikacija pokušavala da pristupi uređajima direktno što je prouzrokovalo mnoge konflikte. Da bi se omogućilo brojnim aplikacijama da dele zajedničke uređaje Microsoft je smislio koncept virtuelnih rukovaoca uređajima (*virtual device driver*). Ovi rukovaoci su bili poznati pod imenom VxD rukovaoci zbog njihovih imena po šablonu VxD.386 gde je x označavao tip uređaja. Ipak, svaki ovakav rukovalac se oslanjao na realni MS-DOS rukovalac uređajem. Najzad je Microsoft predstavio novu osnovu za projektovanje rukovaoca i nazvao je WDM (*Windows Driver Model*). Windows 98 i Windows Me su projektovani sa ovakvim rukovaocima. Ideja je bila da WDM bude isti na svim platformama i dovoljno bi bilo napisati samo jedan rukovalac za sve. Windows 2000 i Windows XP su takođe zasnovani na ovom modelu projektovanja rukovalaca. Ovi rukovaoci imaju ektenziju (.sys) isto kao prvi rukovaoci uređajima.

Zadatak ovog projekta je realizacija programske podrške sistema za snimanje i reprodukciju digitalnog audio i video podatka u realnom vremenu preko I^2S/BT.656 sprege zasnovanog na *Texas Instruments* TMS320DM642 DSP procesoru. Sistem se sastoji od PC računara (sa Windows XP operativnim sistemom) i TMS320DM642 EVM (*Evaluation Module*) PCI kartice na kojoj se nalazi navedeni procesor. DM642 EVM je kartica namenjena za razvoj, evaluaciju i testiranje aplikacija za *Texas Instruments* C6xx familiju procesora za digitalnu obradu signala.

Programska podrška treba da se sastoji iz tri komponente: PC aplikacije, rukovaoca TMS320DM642 EVM PCI karticom po WDM specifikaciji i programske podrške za TMS320DM642 DSP.

TMS320DM642 EVM ploča će dalje u tekstu biti označena kraće kao DM642 EVM ili samo EVM. Slično će procesor TMS320DM642 DSP biti označen kao DM642 ili samo DSP.

PC aplikacija treba da obezbedi da se audio podaci snimaju ili reprodukuju iz datoteke po WAV formatu, a video podaci snimaju ili reprodukuju iz datoteke DVS formata. Rukovalac DM642 EVM PCI karticom treba da obezbedi pristup resursima kartice od strane PC aplikacije kao i prenos podataka. Programska podrška na DM642 DSP treba da obezbedi prenos podatka između I^2S/BT.656 sprege i radne memorije PC računara posredstvom PCI magistrale.

Sistem za snimanje i reprodukciju digitalnog audio/video signala se sastoji od tri projekta. To su PC aplikacija, rukovalac DM642 EVM PCI karticom i programska podrška za DM642. PC aplikacija i rukovalac EVM PCI karticom su deo operativnog sistema Windows XP a treći projekat je programska podrška za DSP na EVM kartici.

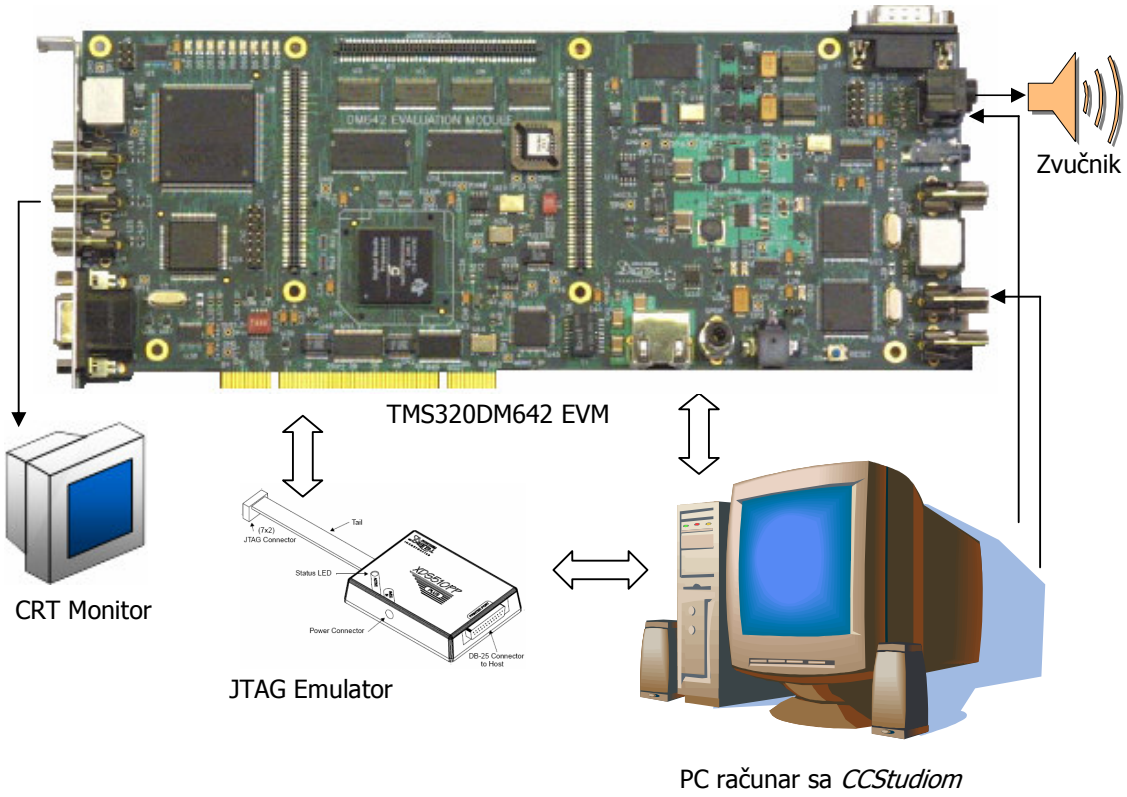
Rukovalac i PC aplikacija su realizovani korišćenjem *Microsoft Visual C++ 6.0™ IDE* razvojnog okruženja. Kao razvojno okruženje za programsku podršku DSP, za potrebe prevođenja i simulacije korišćen je *Texas Instruments Code Composer Studio 2.21™ IDE* koji podržava TMS320DM642 DSP procesor za koji je ovaj projekat napravljen.

2. Okruženje za razvoj i testiranje

Za realizaciju i testiranje zadatog problema u ovom projektu korišćene su sledeće komponente:

- platforma TMS320DM642 EVM za razvoj i testiranje aplikacija zasnovanih na TMS320DM642 DSP procesoru
- PC računar sa *TI Code Composer Studio 2.21*™ razvojnim okruženjem
- *Spectrum Digital* XDS510PP PLUS JTAG Emulator
- prikazna jedinica sa ulazom za kompozitni video signal
- zvučnici

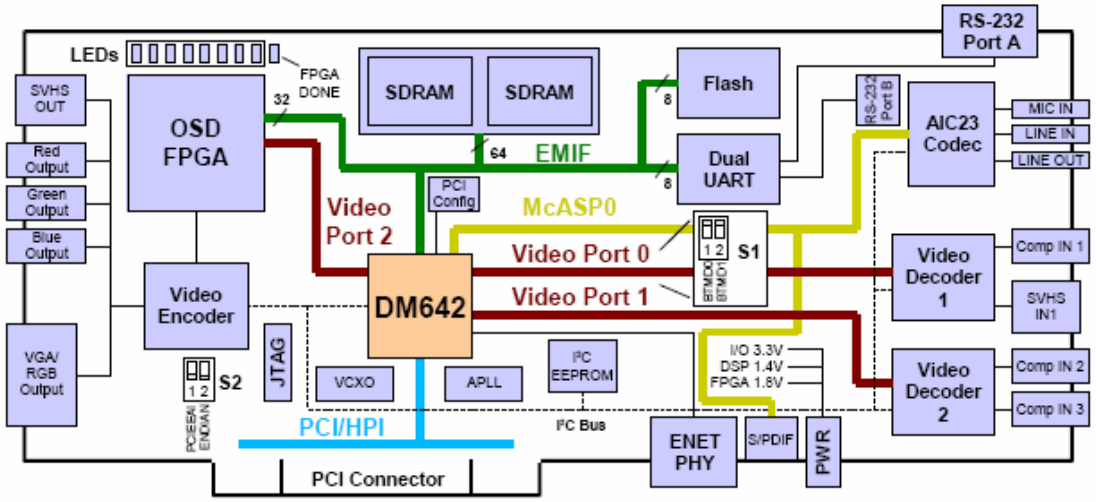
Na slici 2.1 dat je blok dijagram okruženja za razvoj i testiranje.



Slika 2.1: Blok dijagram okruženja za razvoj i testiranje

2.1 Arhitektura TMS320DM642 EVM platforme[2]

DM642 EVM je samostalna razvojna platforma koja omogućava korisnicima evaluaciju, razvoj i testiranje aplikacija za TI C64xx DSP familiju procesora. Zasnovana je na DM642 procesoru za digitalnu obradu signala i veličine je 210x115mm. Može da se koristi kao samostalna platforma sa spoljašnjim napajanjem ili kao PCI (*Periferal Component Interconect*) kartica koja omogućava evaluaciju određenih karakteristika DM642 da bi se odredilo da li procesor zadovoljava zahteve aplikacije. EVM takođe služi i kao referenca za kreiranje ciljne platforme za DM642 DSP.



Slika 2.2: DM642 EVM blok dijagram

Osnovne karakteristike ove platforme su:

- Texas Instruments TMS320DM642 DSP sa radnim taktom od 720 MHz
- Standardni PCI konektor
- 3 video priključka (*port*) sa 2 video dekodera i 1 video koderom
- 32 MB sinhrone DRAM memorije
- OSD posredstvom FPGA
- 4 MB Flash memorije
- AIC23B stereo audio kodek
- Mrežni sprežni sistem
- Programsko konfigurisanje ploče preko registara unutar FPGA
- Mogućnost konfiguracije opcija inicijalnog punjenja (*boot*)
- JTAG emulacija preko sprege sa eksternim emulatorom
- 8 korisničkih LED dioda
- Mogućnost eksternog napajanja od +5V
- Konektori za proširenje dodatnim karticama
- Dvostruki UART sa RS-232 rukovaocima

2.1.1 Funkcionalni pregled

DSP na DM642 EVM ploči je spregnut sa ostalim periferijama na ploči posredstvom 64-bitne EMIF sprege ili kroz jedan od tri 8/16-bitna video priključka.

Dva video dekodera i jedan video koder su standard na EVM ploči. Video koderi i dekoderi na ploči su spregnuti sa video priključcima i konektorima za proširenje. Priključci 0 i 1 su sprega sa TI TVP5416 i TVP5150A video dekoderima a priključak 2 posredstvom FPGA je sprega sa Phillips SAA7105 video koderom. Video priključak 0 i video priključak 1 se koriste za snimanje a video priključak 2 se koristi za reprodukciju digitalnih video signala. Brzina snimanja digitalnog video signala je do 80 MHz a brzina prikazivanja je do 110 MHz. Kao ulaz kod snimanja i kao izlaz kod prikazivanja podržavaju YCbCr 4:2:2 format digitalnog video signala 8-bitne rezolucije po ITU-R BT.656 preporuci. Na standardnoj EVM konfiguraciji su video priključak 0 i video priključak 1 programirani tako da mogu da se podele da bi omogućili implementiranje McASP funkcija kao i spregu sa AIC23B stereo audio kodekom. Video dekoderi i video koder se programiraju preko I^2C magistrale na ploči.

Video dekoderi su spojeni sa video izvorima kroz RCA i S-Video utičnicu i podržavaju sve glavne kompozitne video standarde (NTSC, PAL, SECAM). Ulaz bi trebalo da je kompozitni video signal.

Video koder može na izlazu dati RGB, HD video, NTSC/PAL kompozitni video ili S-Video signal takođe preko RCA i S-Video utičnica. Funkcije OSD su implementirane u eksternom FPGA koji se nalazi između izlaznog video priključka i video kodera.

AIC23B stereo audio kodek (TLV320AIC23B) na ploči omogućava DSP-u da reprodukuje i snima analogne audio signale. Podržava uobičajene frekvencije odabiranja kao 8kHz, 48kHz do 96 kHz. Komunikacija sa kodekom ide preko dva serijska kanala. I^2C magistrala je jednosmerni kontrolni kanal koji se koristi kao sprega za kontrolu i podešavanje kodeka. McASP je dvosmerni kanal koji služi za prenos podataka. Analogna sprega je realizovana kroz tri 3.5mm audio utičnice koje odgovaraju ulazu za mikrofon, ulaznom signalu i izlaznom signalu. Kodek može da selektuje između mikrofon ulaza i ulaznog signala kao aktivnih ulaza. Analogni izlaz je doveden na izlazni konektor.

Lokalno programabilna sekvencijalna mreža ili FPGA služi za implementaciju logike koja će povezivati komponente na ploči. FPGA takođe ima programsku korisničku spregu zasnovanu na registrima koja omogućava korisniku da konfiguriše ploču pomoću čitanja iz registara i upisa u njih.

EVM sadrži i 8 LED dioda koje mogu da se koriste za povratnu spregu sa korisnikom. Pristup diodama je pomoću čitanja i upisa u FPGA registre.

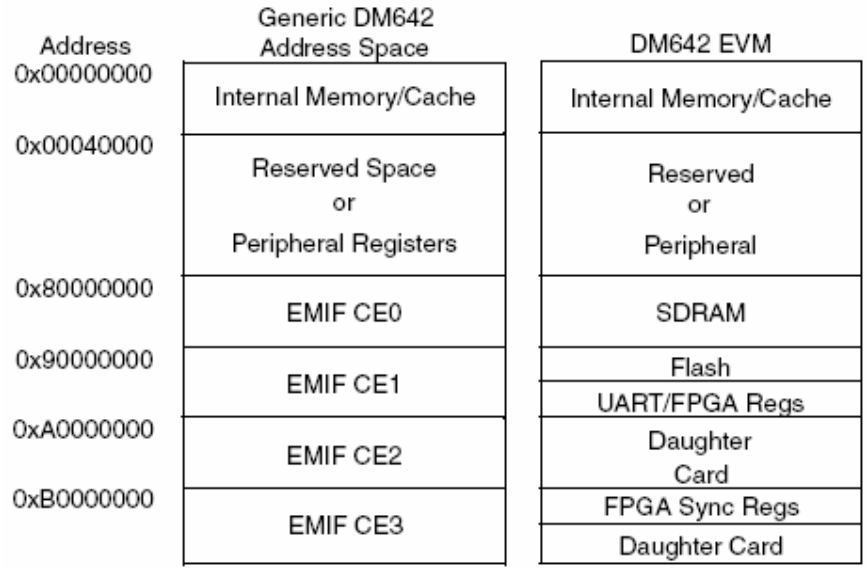
Ploča ima i eksterno napajanje od 5V koje se koristi kod samostalnih aplikacija dok u drugom slučaju, kad je ploča priključena na PCI magistralu, napaja se preko PCI magistrale. Na ploči su i regulatori za prebacivanje napona koji obezbeđuju napon od 1.4V za DSP i 3.3V za I/O potrebe. Ploča je u resetu sve dok su ove potrebe u okviru operativnih specifikacija. Takođe postoji i LDO regulator koji obezbeđuje +1.8V napon jezgra FPGA i +3.3V napajanje za koder i dekodere.

2.1.2 Memorijska mapa

Familija C64xx DSP procesora ima 32-bitni adresni prostor. Program i podaci mogu biti smešteni bilo gde u okviru unifikovanog adresnog prostora.

Memorijska mapa na slici prikazuje adresni prostor DM642 procesora na levoj strani a specifične detalje o svakom regionu i kako se koristi na desnoj strani. Podrazumeva se da se interna memorija uvek nalazi na početku adresnog prostora. Delovi interne SRAM memorije mogu imati ulogu L2 skrivene memorije.

EMIF ima 64-bitnu magistralu sa maksimalnom brzinom od 133 MHz. Podržava operacije čitanja i upisa podataka sa sinhronim i asinhronim pristupom širine 8-, 16-, 32- i 64-bita. Ima 4 signala za selekciju (*chip enable – CE*) memorijskog prostora kojem se pristupa (CE0 – CE3). SDRAM se nalazi u CE0 regionu dok Flash, UART i FPGA su mapirani u CE1 region. Eventualne dodatne kartice koriste CE2 i CE3 region. CE3 region je takođe konfigurisan za sinhrone operacije za funkcije OSD i druge sinhrone registre implementirane unutar eksternog FPGA.

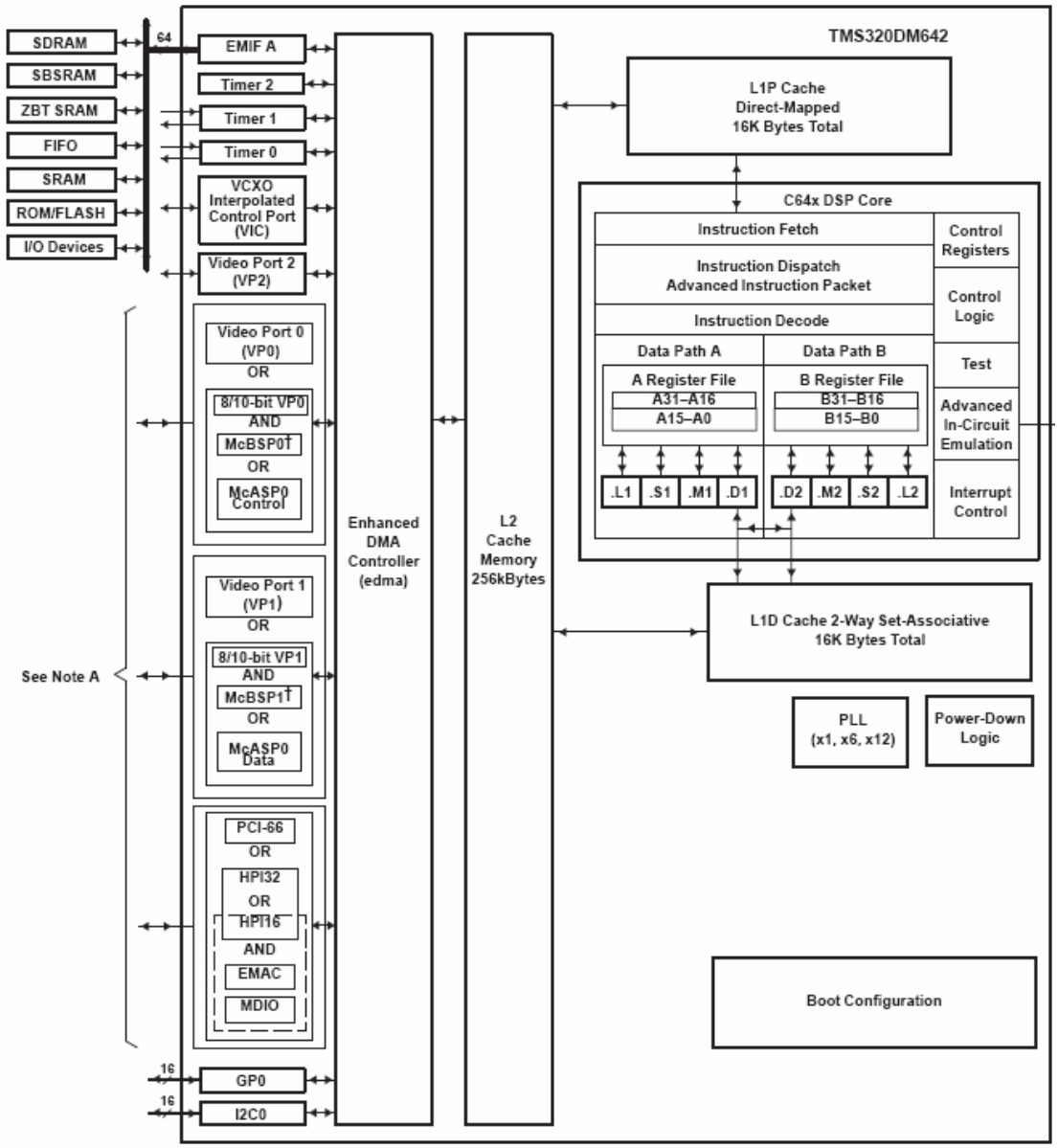


Slika 2.3: Memorijska mapa DM642 EVM

2.2 Arhitektura TMS320DM642 DSP[3]

Procesori serije TMS320C64x™ (uključujući TMS320DM642) su procesori najviših performansi iz TMS320C6000™ DSP serije. TMS320DM642 (DM642) je zasnovan na drugoj generaciji procesora visokog nivoa performansi sa naprednom VelociTI™ arhitekturom sa veoma dugačkom instrukcijskom reči (*Very Long Instruction Word* – VLIW) koja nosi ime VelociTI.2™. Razvila ju je firma Texas Instruments (TI) a namenjena je upravo za obradu digitalnih audio-video signala.

Na slici 2.4 je prikazan blok dijagram TMS320DM642 DSP.



Slika 2.4: Funkcionalni blok dijagram TMS320DM642 DSP

Sa taktom od 600 MHz daje nivo performansi od 4800 miliona instrukcija u sekundi (MIPS). Procesori C64x™ familije imaju 64 registra opšte namene širine 32 bita i paralelizam od 8 nezavisnih funkcionalnih jedinica na osnovu VelociTI.2™ proširenja VelociTI™ arhitekture. Ova proširenja uključuju nove instrukcije koje uvećavaju performanse upravo kod digitalne video obrade.

DM642 ima arhitekturu sa dva nivoa skrivene memorije. Prvi nivo (L1) čine direktno mapirana skrivena memorija za instrukcije (L1P) i dvostruko asocijativna memorija za podatke (L1D), svaka je kapaciteta 16KB. Drugi nivo je (L2) memorija veličine 256KB, koja može biti i interna SRAM memorija u zavisnosti od postavke DM642. Periferije koje se koriste su:

Višekanalni serijski audio priključak (McASP - *Multichannel Audio Serial Port*) podržava jednu zonu za prijem i jednu za prenos, zajedno 8 serijskih linija podataka. Serijski priključak podržava vremensko multipleksiranje na svakoj liniji od 2 do 32 vremenska intervala. DM642 ima dovoljnu propusnu moć da prenosi preko svih 8 linija stereo audio signal frekvencije odabiranja do 192 kHz. Tok podataka iz svake zone se može primati i prenositi simultano u I^2S formatu. Znači McASP može simultano prenositi signale S/PDIF, IEC60958, AES-3, CP-430 formata.

DM642 ima tri periferije za video priključke (VP0, VP1 i VP2) koje mogu da se konfigurišu. Oni obezbeđuju spregu sa video dekoderom i video koderom. Mogu da rade u režimu za snimanje i režimu za reprodukciju. Podržavaju različite rezolucije i video standarde (CCIR601, ITU–BT.656, BT.1120, SMPTE 125M, 260M, 274M, and 296M). Svaki priključak ima bafer veličine 5120 bajta.

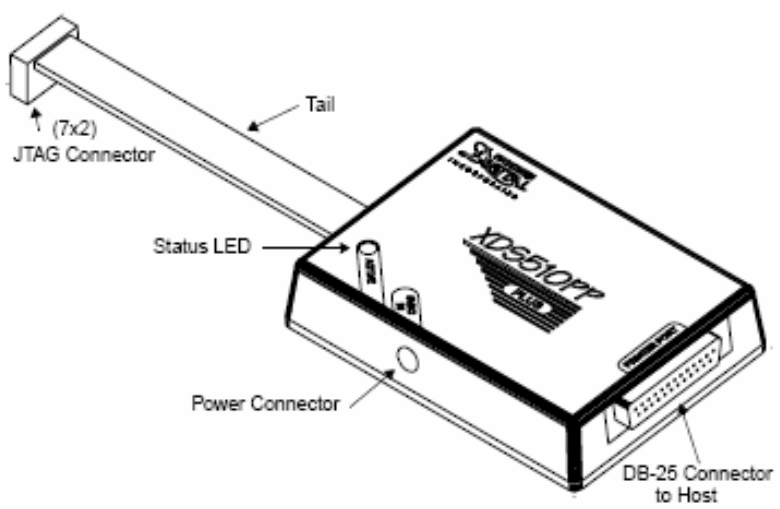
I^2C priključak na TMS320DM642 omogućava kontrolu periferija koje su u sprezi sa procesorom. Osnovne karakteristike su:

- Procesor za obradu digitalnih audio i video signala visokih performansi (TMS320DM642 DSP)
  - 2-, 1.67-ns je vreme trajanja jednog instrukcijskog ciklusa
  - 500-, 600-MHz frekvencija takta
  - Osam 32-bitnih instrukcija u jednom ciklusu
  - 4000, 4800 MIPS (miliona instrukcija u sekundi)
  - Programski potpuno kompatibilan sa C64x™ serijom procesora
- VelociTI.2™ ekstenzije za VelociTI™ napredno TMS320C64x™ jezgro DSP koje radi sa veoma dugačkom instrukcijskom reči
  - Osam nezavisnih funkcionalnih jedinica sa VelociTI.2™ ekstenzijama:
    - 6 ALJ koje podržavaju pojedinačnu 32-bitnu, dvostruku 16-bitnu ili četvorostruku 8-bitnu aritmetiku u jednom taktu
    - 2 množača koji podržavaju četiri 16x16-bitna množenja (32-bitni rezultat) u jednom taktu ili osam 8x8-bitnih množenja (16-bitni rezultat) u jednom taktu
  - 64 registra opšte namene širine 32 bita
  - Pakovanje instrukcija koje smanjuje veličinu programskog koda
  - Sve instrukcije su uslovne
- Karakteristike instrukcijskog skupa
  - Bajt adresabilne (8-, 16-, 32-, 64-bitni podaci)
  - Zaštita od 8-bitnog prekoračenja
  - Operacije nad pojedinačnim bitovima
  - Normalizacija, saturacija, brojanje bitova
  - VelociTI.2™ povećana ortogonalnost

- L1/L2 memorijska arhitektura
  - 128 Kbita (16KB) L1P skrivena memorija za instrukcije sa direktnim pristupom
  - 128 Kbita (16KB) L1D dvosmerna asocijativna skrivena memorija za podatke
  - 2 Mbita (256KB) L2 unifikovano mapirana RAM memorija/skrivena memorija
- 64-bitnu spregu sa eksternom memorijom (EMIF)
  - Spregu ka asinhronim memorijama (SRAM i EPROM) i sinhronim memorijama (SDRAM, SBSRAM, ZBT SRAM i FIFO)
  - 1024MB adresabilni eksterni memorijski prostor
- EDMA (direktni pristup memoriji) kontroler (64 nezavisna kanala)
- 10/100 Mb/s Ethernet MAC
  - IEEE 802.3 standard
  - 8 nezavisnih kanala za prenos i 8 nezavisnih kanala za prijem
- Menadžer ulaznih i izlaznih podataka
- 3 konfigurabilna video priključka
  - Sprega za video dekodere i video koder
  - Podržavaju više rezolucija i video standarda
  - Podržavaju RAW video I/O
  - Sprega za prenos toka podataka
- Priključak sa umetnutom VCXO kontrolom
  - Podržava audio/video sinhronizaciju
- 32-bitna/66-MHz, 3,3-V PCI sprega saglasna sa PCI specifikacijom 2.2
- Višekanalni audio serijski priključak (McASP)
  - Osam serijskih linija podataka
  - Podržava mnogo I^2S formata i sličnih formata
  - Integrisani sprežni sistem za digitalni audio
- I^2C magistrala
- Dva višekanalna baferovana serijska priključka (McBSP)
- Tri 32-bitna tajmera opšte namene
- 16 ulazno/izlaznih linija opšte namene
- Fleksibilni PLL generator takta
- IEEE-1149.1 (JTAG) standard

2.3 Spectrum Digital XDS510PP PLUS JTAG Emulator[4]

XDS510PP PLUS JTAG Emulator je projektovan za upotrebu sa procesorima za digitalnu obradu signala i mikroprocesorima koji imaju radne naponske nivoe od +3.3V do +5V. Navedeni emulator može biti napajan preko PD priključka na JTAG konektoru ili preko spoljašnjeg izvora napajanja koji se dobija uz njega. XDS510PP PLUS je napravljen tako da bude kompatibilan sa Texas Instruments XDS510 emulatorom i sa Texas Instruments razvojnim okruženjem. Standardizovan je 1990 godine kao IEEE 1149.1 standard. Na slici 2.5 prikazan je JTAG emulator

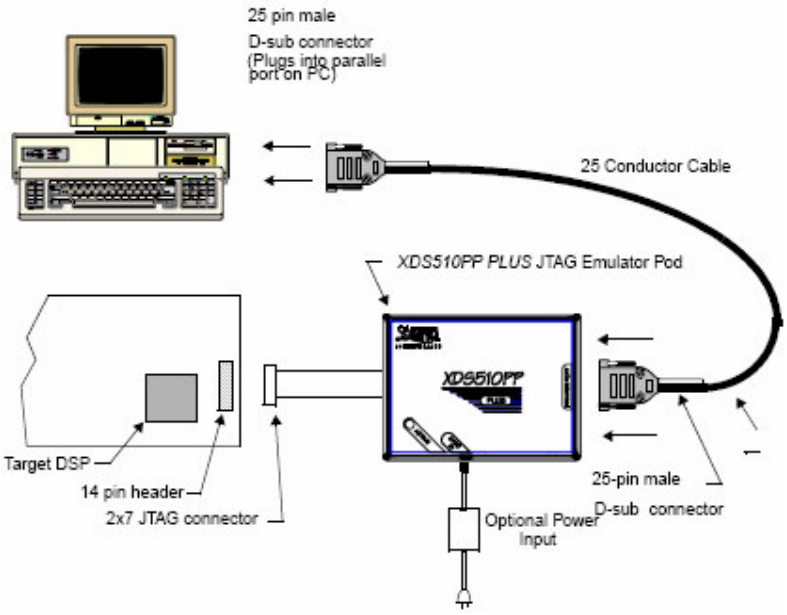


Slika 2.5: XDS510PP PLUS JTAG Emulator

Dimenzije XDS510PP PLUS JTAG emulatora su 8.255 x 11.1252 x 24.13 cm. Njegove glavne karakteristike su:

- Podržava Texas Instruments DSP procesore i mikroprocesore sa JTAG spregom (IEEE 1149.1)
- Kompatibilan je sa Texas Instruments XDS510 emulatorom
- Podržava standardnu paralelnu komunikacionu spregu sa PC računarom (SPP8, EPP i ECP) bez dodatnih adaptera
- Podržava JTAG sprege sa naponima u rasponu od 3.3V do 5V
- Napaja se sa platforme ili preko spoljnog napajanja
- Kompatibilan je sa *Texas Instruments Code Composer Studio* razvojnim okruženjem

Na slici 2.6 prikazano je povezivanje XDS510PP PLUS emulatora sa PC računarom i ciljnom DSP platformom.



Slika 2.6: Povezivanje XDS510PP PLUS

Sa ciljnom platformom se povezuje preko 14-pinskog konektora po standardu IEEE 1149.1 (JTAG) čiji je raspored linija signala prikazan na slici 2.7 i opisan u tabeli 2.8

TMS	1	2	TRST-
TDI	3	4	GND
PD	5	6	no pin (key)
TDO	7	8	GND
TCK-RET	9	10	GND
TCK	11	12	GND
EMU0	13	14	EMU1

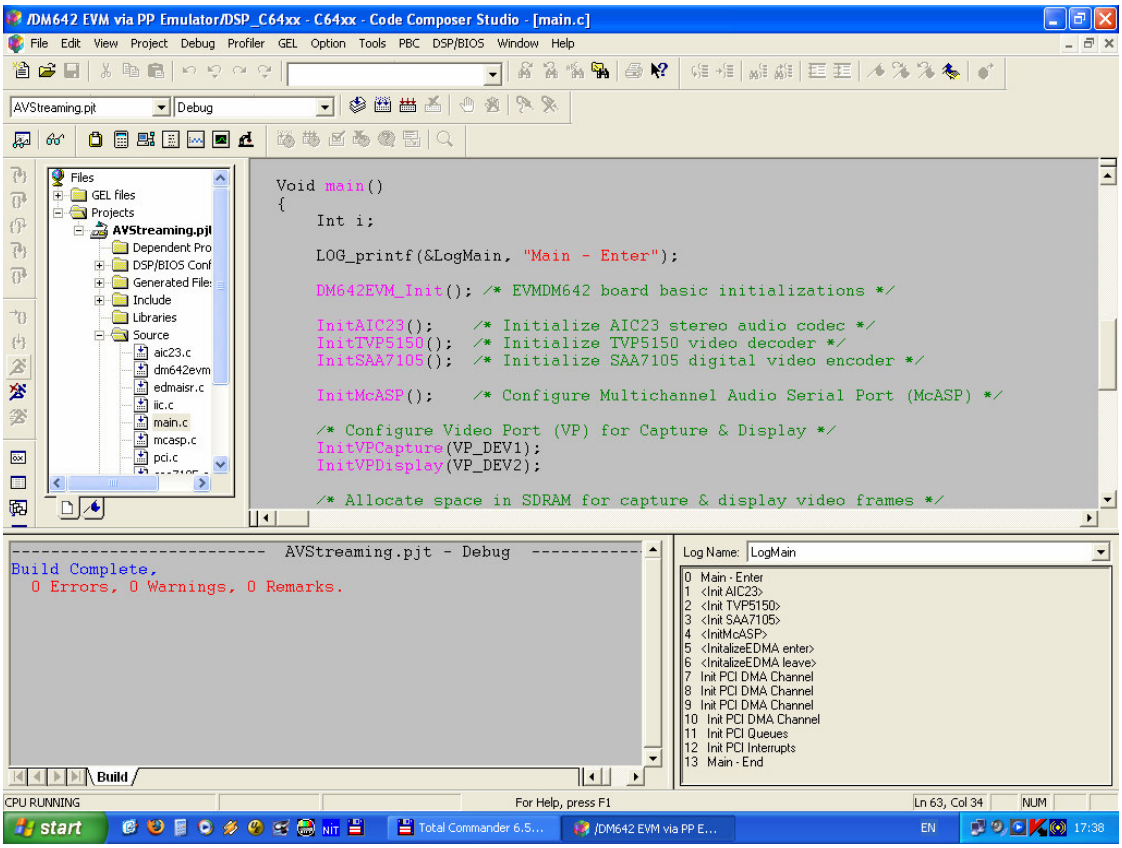
Slika 2.7: Raspored signala na 14-pinskom konektoru

Signal	Description	Emulator State	Target State
TMS	JTAG test mode select.	Output	Input
TDI	JTAG test data input.	Output	Input
TDO	JTAG test data output.	Input	Output
TCK	JTAG test clock. TCK is a 12-MHz clock source from the emulation pod. This signal can be used to drive the system test clock.	Output	Input
TRST-	JTAG test reset.	Output	Input
EMUO	Emulation pin 0.	I/O	I/O
EMU1	Emulation pin 1.	I/O	I/O
PD	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to the target processor's I/O pins Vcc.	Input	Output
TCK_RET	JTAG test clock return. Test clock input to the emulator. May be a buffered or unbuffered version of TCK.	Input	Output

Slika 2.8: Tabela opisa signala za 14-pinski konektor

2.4 Integrisano razvojno okruženje TI Code Composer Studio 2.21™[5]

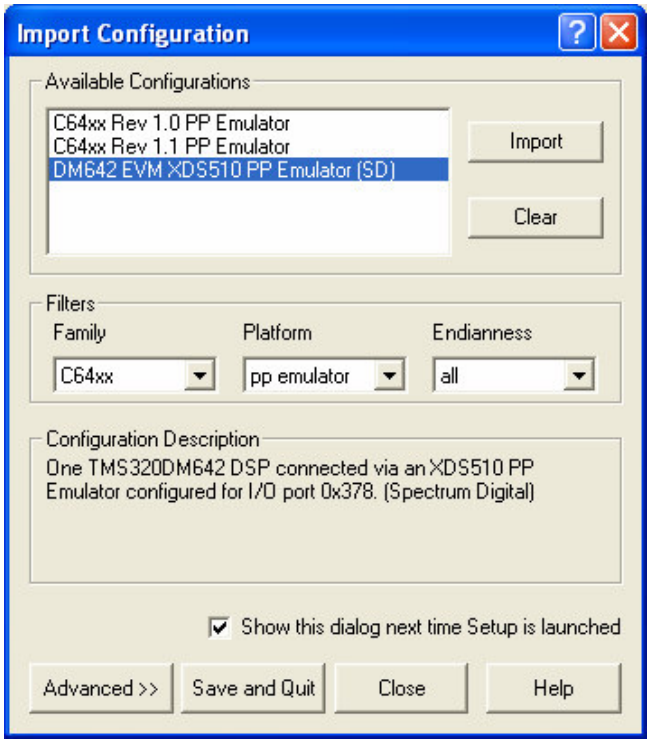
Za razvoj, analizu i testiranje programske podrške za DM642 u okviru ovog projekta korišćen je programski paket *Texas Instruments Code Composer Studio 2.21*™ IDE (*Integrated Development Enviroment*) namenjen Texas Instruments DSP procesorima iz familije C6000 (dalje u tekstu CCStudio). U CCStudiu je moguće realizovati sve faze razvoja programske podrške, pisanje izvornog koda (asembler ili C/C++), prevođenje, povezivanje, analiziranje i testiranje u realnom vremenu. EVMDM642 kartica je predviđena za rad sa CCStudio razvojnim okruženjem. CCStudio komunicira sa pločom preko eksternog JTAG emulatora.



Slika 2.9: Radni ekran CCStudia

2.4.1 Konfigurisanje CCStudia

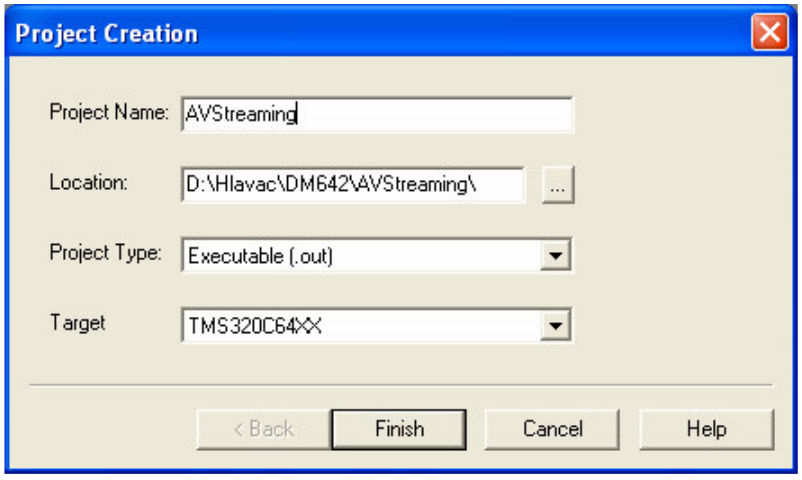
Pre korišćenja je potrebno izvršiti konfigurisanje CCStudia programom CCStudio Setup. Za ovaj projekat odgovarajuća konfiguracija je DM642 EVM XDS510 PP Emulator (SD). Prozor za konfiguraciju je prikazan na slici 2.10



Slika 2.10: Prozor za izbor konfiguracije

2.4.2 Formiranje projekta U CCStudiu

Prvi korak u razvoju programske podrške je kreiranje novog projekta. Za kreiranje novog projekta pojavljuje se na ekranu dijalog koji je prikazan na slici 2.11

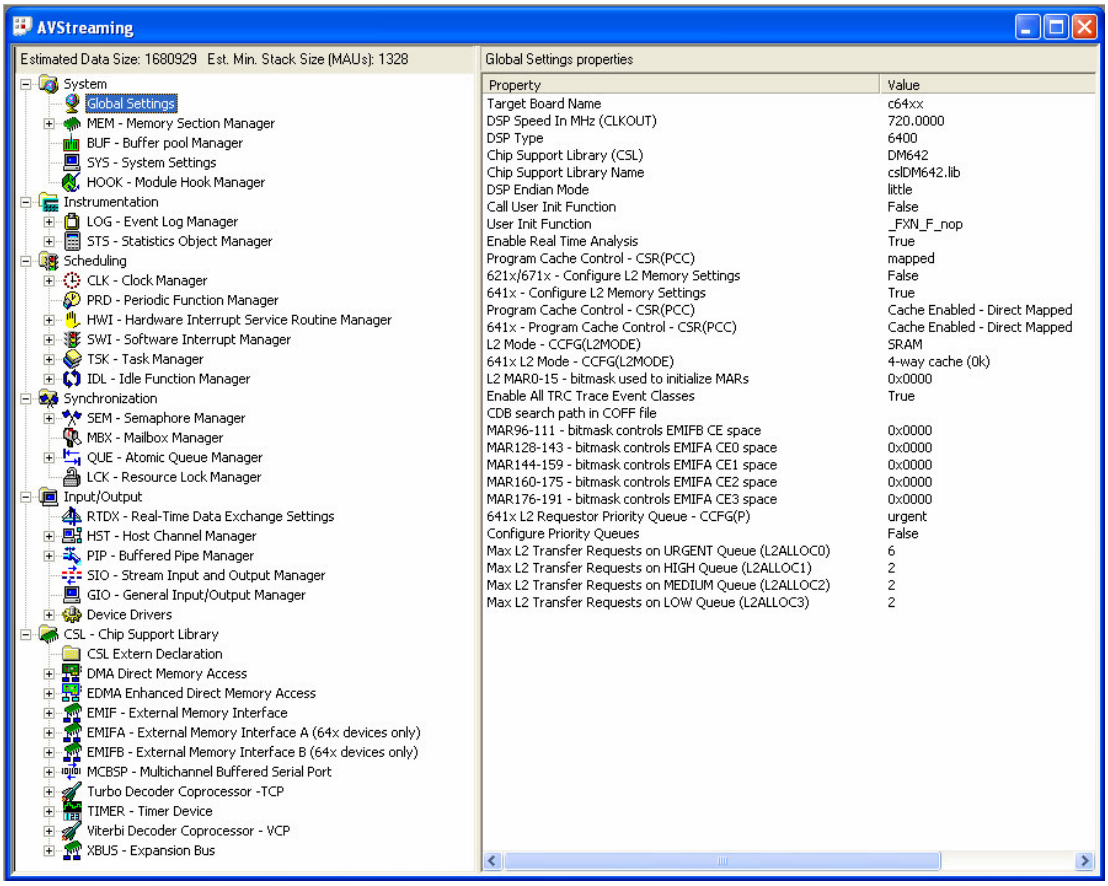


Slika 2.11: Kreiranje novog projekta u CCStudiu

2.4.3 DSP/BIOS

DSP/BIOS je osnova operativnog sistema za DSP. Namenjen je za rad u realnom vremenu, kreiranju, raspodeli i pristupu resursima, raspodeli procesa, za komunikaciju i sinhronizaciju između procesa, za merenje i analizu u realnom vremenu. Objekti koji se koriste u programu se mogu kreirati u DSP/BIOS-u i u izvornom kodu programa. Razlika

je u tome, što su objekti kreirani u DSP/BIOS statički i povezuju se sa izvršnim kodom a objekti kreirani u izvršnom kodu su dinamički. Način za kreiranje i korišćenje DSP/BIOS je preko konfiguracione datoteke (.CDB). Na slici 2.12 je prozor otvorene jedne takve konfiguracione datoteke



Slika 2.12: Izgled konfiguracije DSP/BIOS

Alat za konfiguraciju DSP/BIOS-a je podeljen na sledeće grupe modula:

- **System** – ovde su moduli neophodni za ispravno funkcionisanje celog sistema
- **Instrumentation** – ovi moduli su namenjeni za praćenje izvršavanja programa, vođenje dnevnika i merenje brzine delova koda
- **Scheduling** – moduli za vremensko planiranje izvršavanja određenih procedura i funkcija
- **Synchronization** – moduli za sinhronizaciju delova programa i ulančavanje zahteva
- **Input/Output** – moduli za pravljenje objekata namenjenih za kontinualni prenos podataka u realnom vremenu
- **CSL** (**Chip Support Library**)[8] – moduli za upravljanje raznim delovima DSP procesora

Posle podešavanja parametara svih DSP/BIOS objekata potrebno je snimiti datoteku sa DSP/BIOS konfiguraciom. Prilikom snimanja ove datoteke, alat za konfiguraciju DSP/BIOS-a automatski generiše i datoteke sa izvornim kodom u C-u i

asembleru u kojima su definisani i inicijalizovani svi DSP/BIOS objekti. Takođe se automatski generiše i komandna datoteka povezioca (datoteka sa ekstenzijom .cmd) koja je neophodna za povezivanje programa u slučaju da se koristi tekstualni povezilac. Datoteke sa ekstenzijama .cmd i .cdb je potrebno uključiti u projekat.

2.4.4 Prevođenje i povezivanje programa

Zadnji korak u razvoju programske podrške za DSP je njeno prevođenje i povezivanje. Pre prevođenja i povezivanja je potrebno podesiti parametre prevodioca i povezioca. Ako se posle prevođenja projekta promeni neka od datoteka uključenih u projekat, dovoljno je prevesti samo tu datoteku i povezati je sa programom. Ako nema grešaka pri prevođenju, prevodilac u *Debug* direktorijum unutar direktorijuma projekta kreira datoteku sa ekstenzijom .out u kojoj se nalazi relokatibilni izvršni kod prevedenog programa. Ovakav izvršni kod se prebaci u memoriju DSP procesora i spreman je za pokretanje.

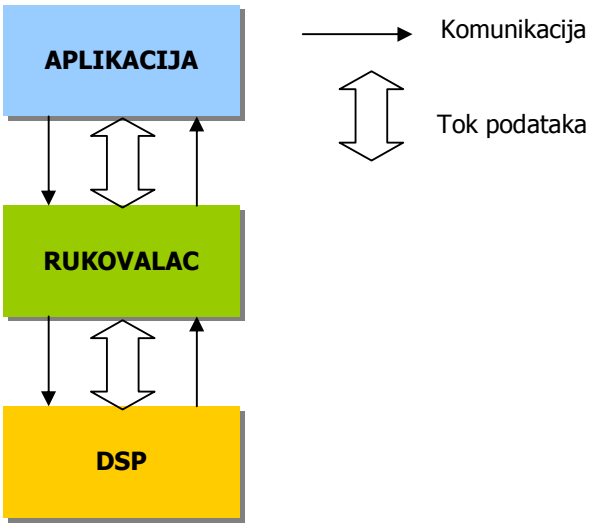
3. Sistem za snimanje i reprodukciju digitalnog audio/video signala

Sistem za snimanje i reprodukciju digitalnog audio/video podatka se zasniva na dvosmernom toku audio/video podataka. Prenos audio/video podataka od I^2S/BT.656 sprege na TMS320DM642 EVM PCI kartici prema radnoj memoriji PC računara se zove „snimanje“ a prenos audio/video podataka u suprotnom smeru, od radne memorije PC računara prema I^2S/BT.656 spregi se zove „reprodukcija“. I^2S je audio sprega a BT.656 je video sprega.

Programska podrška sistema za snimanje i reprodukciju digitalnog audio/video podatka u realnom vremenu se sastoji od tri komponente koje su međusobno povezane. To su:

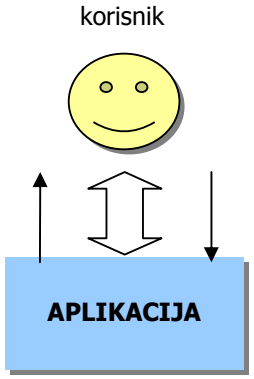
- PC aplikacija
- Rukovalac DM642 EVM karticom
- Programska podrška za DM642 DSP.

Između ova tri programa postoji dvosmerna komunikacija i dvosmerni tok podataka i to na način kao što je prikazano na slici 3.1:



Slika 3.1: Komunikacija i tok podataka u sistemu

Korisnička sprega je deo PC aplikacije. Korisnik upravlja sistemom za snimanje i reprodukciju kroz PC aplikaciju. To je prikazano na slici 3.2



Slika 3.2: Sprega korisnika sa sistemom

Snimanje i reprodukcija digitalnog audio i video signala radi u full-duplex režimu. Snimanje i reprodukcija su međusobno nezavisni, audio i video su međusobno nezavisni tako da postoje četiri potpuno nezavisna toka podataka koji se mogu simultano izvršavati u istom trenutku vremena.

3.1 PC aplikacija

Kako se u PC aplikaciji nalazi sprega sistema sa korisnikom, moglo bi se reći sa korisničke tačke gledišta da PC aplikacija treba da omogući ono što se očekuje od samog sistema a to je snimanje i reprodukcija audio signala kao i snimanje i reprodukcija video signala. Za reprodukciju digitalnih audio/video signala kao izvor se koriste ulazne datoteke a kod snimanja digitalnih audio/video signala se podaci smeštaju u izlazne datoteke. Za audio se koristi WAV format datoteka a za video se koristi DVS format datoteka.

3.2 Rukovalac DM642 EVM karticom

Rukovalac ima centralnu ulogu u sistemu za snimanje i reprodukciju audio/video signala. Treba da omogući prebacivanje izvršnog koda programa u memoriju DSP, da omogući komunikaciju između PC aplikacije i DSP-a, transfer podataka između njih kao i transfer podataka preko PCI magistrale. Treba da prepozna da li se radi o audio ili video transferu i koji smer prenosa podataka je u pitanju.

Rukovalac dobija zahteve direktno od PC aplikacije, tako da su rukovalac i aplikacija usko povezani. Na drugoj strani, rukovalac i DSP mogu da komuniciraju na dva načina. Prvi je preko hardverskih prekida između rukovaoca kao dela operativnog sistema i kartice koja je priključena na PCI magistralu. Druga mogućnost je memorijski prozor od 4MB kroz koji rukovalac može direktno da pristupi memoriji DSP-a. Rukovalac treba da ima mogućnost da prenese DSP-u informaciju o tome da li se u zahtevu od strane aplikacije radi o audio ili video transferu i takođe o kojem smeru transfera se radi. Rukovalac treba da obezbedi i prostor u operativnoj memoriji računara u koji će se smeštati podaci preneti preko PCI magistrale, na putu ka aplikaciji, ili od aplikacije preko PCI magistrale ka DSP.

3.3 Programska podrška za DSP

Programska podrška za DSP treba da omogući: konfigurisanje audio kodeka, video kodera i video dekodera, audio i video priključaka, konfiguraciju magistrala za prenos podataka od priključaka do interne (ISRAM) ili eksterne (SDRAM) memorije. Do koje memorije se podaci prenose zavisi prvenstveno od veličine tih podataka. U ovom slučaju za audio je dovoljna interna memorija, u koju se smešta i izvršni kod programa a za video se mora koristiti eksterna memorija na platformi.

Pošto se radi o toku podataka, sva tri navedena programa treba da obezbede sinhronizaciju, konzistentnost podataka i ispravan redosled primljenih i prenetih podataka. Treba da prepoznaju korisničko zaustavljanje i ponovno pokretanje transfera, početak i kraj prenosa podataka, prirodni završetak prenosa podataka.

Krajnji ulazi i izlazi na platformi su analogni audio i video signali, gde A/D (analogno/digitalnu) i D/A (digitalno/analognu) konverziju rade audio koder, video koder i video dekoder koji se nalaze na samoj platformi. Jedino je potrebno od strane DSP da se kodek konfiguriše po I^2S standardu za prenos audio signala a video koder i video dekoder konfigurišu po BT.656 preporuci za prenos video signala.

4. Realizacija programske podrške sistema za snimanje i reprodukciju digitalnog audio/video signala

Programska podrška sistema za snimanje i reprodukciju digitalnog audio/video signala se sastoji iz tri komponente: PC aplikacije, rukovaoca TMS320DM642 EVM PCI karticom po WDM specifikaciji i programske podrške za TMS320DM642 DSP.

Programska podrška je napravljena kao tri nezavisna projekta. Projekat za PC aplikaciju (TEST) i projekat za rukovalac (DM642D) su napravljeni u *Microsoft Visual C++ 6.0*™ razvojnom okruženju. Rukovalac je u skladu sa WDM (*Windows Driver Model)* specifikacijom za rukovaoce. Projekat programske podrške za DSP (AVStreaming) je napravljen u *Texas Instruments Code Composer Studio 2.0*™ *IDE* (CCStudio) koji kao osnovu koristi programski jezik C. Svaki od ova tri programa se nalazi u datoteci sa odgovarajućom ekstenzijom:

- PC aplikacija (Test.exe)
- Rukovalac EVM karticom (dm642d.sys)
- Programska podrška za DSP (AVStreaming.out)

Pošto platforma ima mogućnost da sama upravlja PCI DMA transferom (*PCI Bus Master*) prenos preko PCI magistrale je realizovan kao DMA (*Direct Memory Access*) transfer gde je EVM vodeći (*Master*) a rukovalac je prateći (*Slave*). Maksimalna veličina pojedinačnog DMA PCI transfera je 64 KB.

Pošto postoje dva tipa podataka (audio i video), zajedno će biti četiri tipa transfera:

- Snimanje audio signala
- Snimanje video signala
- Reprodukcija audio signala
- Reprodukcija video signala

4.1 Format signala

Format audio i video signala je unapred određen po traženim standardima u zadatku. Audio datoteke su u WAV (.wav) formatu a video datoteke u DVS (.lum, .bmy, .rmy) formatu. Formati se korisnički ne mogu menjati.

4.1.1 Audio format

I^2S je serijski sprežni sistem koji povezuje digitalne audio uređaje. Koristi se za prenos 2 kanala (stereo) PCM digitalnih podataka. Digitalni audio format koji se koristi je WAV audio format. To je *Microsoft* i *IBM* standard za smeštanje digitalnog audio podatka na PC računarima. Oslanja se na PCM format audio zapisa koji je nekompresovan (bez gubitka podataka). WAV PCM format čuva sve audio odabirke (*samples*) i koristi se za maksimalni audio kvalitet. Takođe je WAV format relativno lak za korišćenje i obradu.

Format digitalnog audio podatka u ovom primeru je stereo (dva kanala – levi i desni) sa brojem odabiraka u sekundi (*SampleRate*) od 48000 Hz (48KHz). Broj bita u odabirku je 16 (16-bitna rezolucija) a veličina odabirka je 32 bita (4 bajta) znači potrebna su 2 bajta za smeštanje jednog odabirka. Funkcija koja predstavlja količinu bajtova snimljenog digitalnog audio podatka u zavisnosti od vremena (B/s) će biti:

$$f_A(t) = 2 \times 2 \times 48.000\, t = 192.000\, t \quad \left[\frac{B}{s}\right]$$

To je 187,5KB u sekundi.

Primer:
Za 1 minut snimanja digitalnog audio signala datoteka će biti veličine

$$192.000 \times 60 = 11.520.000 \quad [B]$$

To je skoro 11MB podataka.

Pošto se audio zapisuje u (*.wav*) format datoteke, zaglavlje (*header*) je veličine 44 bajta. Veličina zaglavlja se dodaje kao konstanta jer ne zavisi od količine podataka.

4.1.2 Video format

Slika se sastoji od konačnog broja elemenata (piksela). Svaki piksel ima svoju sjajnost i boju. Kada se broj piksela po jedinici površine, gustina slike, poveća iznad određene granice ljudsko oko prestaje da ih vidi kao odvojene elemente nego ih vidi kao zajedničku celinu. Broj piksela u slici je određen brojem po horizontali i vertikali odnosno horizontalnom i vertikalnom rezolucijom slike. Svaki piksel kolor slike je potrebno opisati sa tri vrednosti da bi se informacija o sjajnosti i boji prenela do gledaoca. Za crno-belu sliku (jednobojnu, monohromatsku) je dovoljna samo informacija o sjajnosti slike.

Format digitalnog video podatka je po preporuci ITU-R BT.656 (8-bitni) koji definiše paralelnu i serijsku spregu za primopredaju 4:2:2 YCbCr (YUV) digitalnog videa između studijske opreme i profesionalnih video aplikacija. Aktivna rezolucija je 720x576 piksela (PAL 625/50 video sistem). Broj 625 je broj linija jedne slike od kojih je 576 aktivnih linija. Broj 50 je brzina osvežavanja slike sa 50 prepletenih poluslika u sekundi (ili 25 celih slika u sekundi).

PAL je analogni televizijski sistem koji je glavni evropski sistem za emitovanje televizijskih signala. Zbog povratne kompatibilnosti sa crno-belom televizijom PAL koristi luminantno-hrominentni (*luminance-chrominance*) kodni sistem. Luminanca je preuzela mesto originalnog monohromatskog signala i nosi informaciju o sjajnosti a hrominanca nosi informaciju o boji. Ovo omogućava starim crno-belim televizorima da prikažu na ekranu PAL signal jednostavno ignorišući hrominansu. Broj 720 kod rezolucije je broj

piksela u jednoj liniji. Za jedan piksel su potrebna dva bajta memorije, jedan zbog luminanse (Y-komponente) a drugi zbog hrominanse (Cb- i Cr-komponente zajedno). Y komponenta je dvostruko veća od Cb i Cr komponente, koje su iste veličine (oznaka 4:2:2). Znači količina memorije potrebna za smeštanje jedne slike video signala je:

$$720 \times 576 \times 2 = 829.440 \quad [B]$$

To je 810 KB podataka.

Pošto imamo 25 takvih slika u sekundi, količina memorije u bajtima potrebna za smeštanje digitalnog video signala u sekundi je (B/s):

$$f_V(t) = 25 \times 829.440t = 20.736.000t \quad \left[\frac{B}{s}\right]$$

To je skoro 20MB podataka u sekundi.

Primer:
Za 1 minut snimanja digitalnog video signala datoteka će biti veličine

$$20.736.000 \times 60 = 1.244.160.000 \quad [B]$$

To je preko 1GB podataka u minuti.

Kako se video zapisuje u DVS formatu zapisa, podaci će biti smešteni u 3 zasebne datoteke zbog postojanja 3 komponente video signala (Y, U i V komponenta). Oznaka svake od datoteka je sledeća: za Y-komponentu (*.lum*), za U-komponentu (*.bmy*) i za V-komponentu (*.rmy*). Kod DVS formata veličina zaglavlja je 120 bajta. Dodaje se u svaku datoteku. Veličina datoteka je određena veličinom komponenti, znači za jednu sekundu snimanja video signala (829.440 B) biće:

Y - komponenta = 414720 B (*VideoCaptureY.lum*)
U - komponenta = 207360 B (*VideoCaptureU.bmy*)
V - komponenta = 207360 B (*VideoCaptureV.rmy*)

Kada se količina video podataka podeli sa količinom audio podataka u sekundi dobijamo odnos

$$\frac{f_V(t)}{f_A(t)} = 108 \qquad f_V(t) \quad >> \quad f_A(t)$$

Zbog ovoga možemo reći da je količina video podataka daleko veća od količine audio podataka. Svi problemi vezani za izvršavanje u realnom vremenu i oni koji se odnose na sam prenos podataka u sistemu se mogu analizirati i rešavati samo sa video podacima jer je sam video tok podataka zanemarljivo manji od celokupnog toka podataka (prenos video i audio podataka zajedno).

Svi primeri i objašnjenja u vezi sa tokom podataka i implementacijom su dati na osnovu video podataka, konkretno na snimanju video signala.

5. PC aplikacija

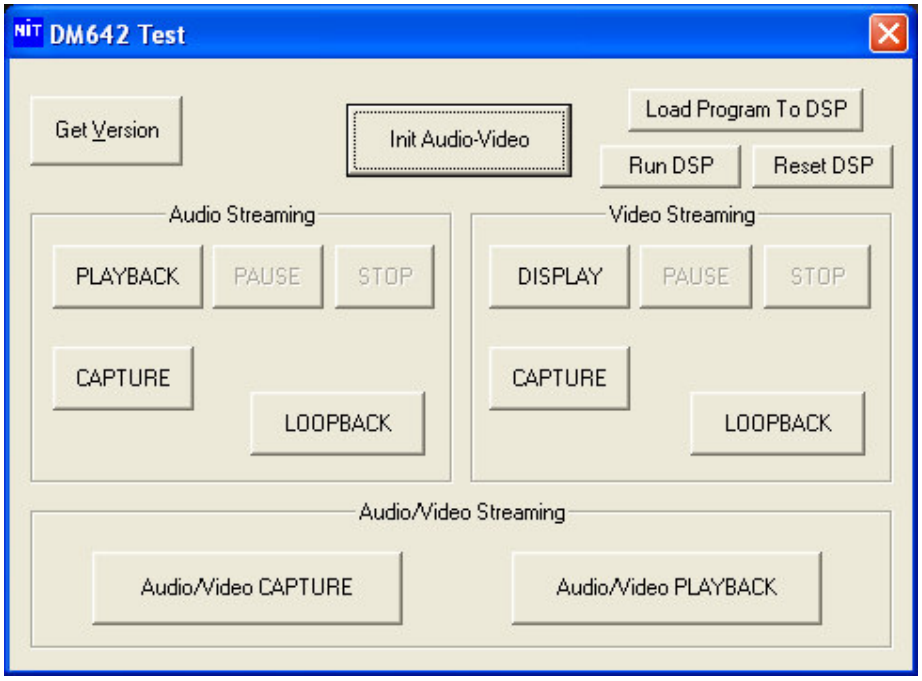
PC aplikacija se sastoji iz:

- sprege programa sa korisnikom
- bloka za transfer podataka između korisnika i rukovaoca
- bloka za prebacivanje izvršnog koda u memoriju DSP-a

Ulaz i izlaz aplikacije su datoteke koje se nalaze na masovnoj memoriji računara. Aplikacija je realizovana u vidu dijaloga na osnovu MFC klasa. Kreira se izvršna datoteka sa ekstenzijom .exe (*Test.exe*). Kod pokretanja aplikacije se poziva funkcija `OnInitDialog` (*TestDlg.cpp*) koja otvara prozor DM642 Test i alocira memoriju potrebnu za transfer audio i video podataka, koja se oslobađa zatvaranjem aplikacije. U okviru ove funkcije se poziva `GetDeviceViaInterface` (*TestDlg.cpp*) koja inicijalizuje pokazivač na rukovaoca DM642 EVM karticom `hDM642Device` (*TestDlg.h*). Ovaj pokazivač na rukovalac je glavni parametar kod svih funkcija komunikacije i prenosa podataka aplikacije sa rukovaocem.

5.1 Korisnička sprega

Izgled sprege programa sa korisnikom dat je na slici 5.1.



Slika 5.1: Izgled prozora korisničke sprege

Sprega programa sa korisnikom je realizovana pomoću dijaloga koji ima inicijalizacioni blok, blok za audio tok podataka, blok za video tok podataka i blok za zajednički audio/video tok podataka. Inicijalizacija obuhvata alociranje struktura podataka u rukovaocu za DMA prenos i prebacivanje izvršnog koda u memoriju DSP-a. Blokovi za tok podataka sadrže pokretanje i zaustavljanje transfera audio i/ili video podataka.

❑ Inicijalizacija

- **Get Version.** Poziva funkciju `OnGetVersion` (*TestDlg.cpp*) preko koje dobijamo informaciju o trenutnoj verziji rukovaoca za DM642 EVM ploču.
- **Init Audio-Video.** Poziva funkciju `OnInitAudioVideo` (*TestDlg.cpp*). Priprema se program za snimanje i reprodukciju tako što se inicijalizuje struktura podataka za komunikaciju između rukovaoca i DSP programa.
- **Load Program To DSP.** Poziva funkciju `OnLoadProgram` (*TestDlg.cpp*). Zahtev rukovaocu da učita izvršni kod u internu memoriju DSP (ISRAM).
- **Run DSP.** Poziva funkciju `OnRunDsp` (*TestDlg.cpp*) koja pokreće izvršavanje koda u DSP-u.
- **Reset DSP.** Poziva funkciju `OnResetDsp` (*TestDlg.cpp*) za zaustavljanje i resetovanje DSP-a.

❑ Audio Streaming

- **PLAYBACK.** Poziva funkciju `OnPlaybackStart` (*TestDlg.cpp*) za pokretanje reprodukcije audio signala sa datoteke na masovnoj memoriji.
- **PAUSE.** Poziva funkciju `OnPlaybackPause` (*TestDlg.cpp*). Pauziranje reprodukcije audio signala sa mogućnošću ponovnog pokretanja sa tačke gde je zaustavljena.
- **STOP.** Poziva funkciju `OnPlaybackStop` (*TestDlg.cpp*). Zaustavljanje reprodukcije audio signala i vraćanje na početak.
- **CAPTURE (STOP).** Poziva funkciju `OnCaptureAudio` (*TestDlg.cpp*). Pokretanje snimanja audio signala u datoteku na masovnoj memoriji. Po pokretanju se dugme menja na STOP koje zaustavlja snimanje. Po završetku snimanja ponovo postaje CAPTURE.
- **LOOPBACK (STOP).** Poziva funkciju `OnLoopbackAudio` (*TestDlg.cpp*). Pokretanje spojenog audio toka podataka u povratnu petlju. Po pokretanju se dugme menja na STOP koje zaustavlja izvršavanje. Po završetku snimanja ponovo se menja na LOOPBACK.

❑ Video Streaming

- **DISPLAY.** Poziva funkciju `OnDisplayStart` (*TestDlg.cpp*) za pokretanje reprodukcije video signala sa datoteka na masovnoj memoriji.
- **PAUSE.** Poziva funkciju `OnDisplayPause` (*TestDlg.cpp*). Pauziranje reprodukcije video signala sa mogućnošću ponovnog pokretanja sa tačke gde je zaustavljena.
- **STOP.** Poziva funkciju `OnDisplayStop` (*TestDlg.cpp*). Zaustavljanje reprodukcije video signala sa vraćanjem na početak.
- **CAPTURE (STOP).** Poziva funkciju `OnCaptureVideo` (*TestDlg.cpp*). Pokretanje snimanja video signala u datoteke na masovnoj memoriji. Po pokretanju se dugme menja na STOP koje zaustavlja snimanje. Po završetku snimanja ponovo se menja na CAPTURE**.**

- **LOOPBACK (STOP).** Poziva funkciju `OnLoopbackVideo` (*TestDlg.cpp*). Pokretanje spojenog video toka podataka u povratnu petlju. Po pokretanju se dugme menja na STOP koje zaustavlja izvršavanje. Po završetku snimanja ponovo postaje LOOPBACK.

❑ **Audio/Video Streaming**

- **Audio/Video CAPTURE (STOP).** Poziva funkciju `OnAudioVideoCapture` (*TestDlg.cpp*). Pokretanje istovremenog snimanja audio i video signala u datoteke na masovnoj memoriji. Po pokretanju se dugme menja na STOP koje zaustavlja snimanje. Po završetku snimanja ponovo se menja na Audio/Video CAPTURE**.**
- **Audio/Video PLAYBACK (STOP).** Poziva `OnAudioVideoPlayback` (*TestDlg.cpp*) funkciju. Pokretanje istovremene reprodukcije audio i video signala sa datoteka na masovnoj memoriji. Po pokretanju se dugme menja na STOP koje zaustavlja reprodukciju. Po završetku reprodukcije ponovo postaje Audio/Video PLAYBACK**.**

5.2 Prebacivanje izvršnog koda u memoriju DSP-a

Pritiskom na dugme **Load Program To DSP** u inicijalizacionom bloku korisničke sprege poziva se funkcija `OnLoadProgram`. Unutar ove funkcije se poziva funkcija `DeviceIoControl` sa zahtevom rukovaocu da učita izvršni kod DSP programa u internu memoriju DM642 DSP-a. Izvršavanje programa se pokreće pritiskom na dugme **Run DSP** a zaustavlja i resetuje pritiskom na dugme **Reset DSP.** Pokretanje i zaustavljanje je takođe preko funkcije `DeviceIoControl`.

5.3 Transfer podataka

Za potrebe transfera prilikom pokretanja aplikacije se alociraju 4 statička bafera u operativnoj memoriji (memorija se oslobađa tek kad se aplikacija zatvara) za svaki tip transfera. To su:

- **`pAudioCaptureBuffer`** (za snimanje audio podataka) – 61.440 B
- **`pAudioPlaybackBuffer`** (za reprodukciju audio podataka) – 61.440 B
- **`pVideoCaptureBuffer`** (za snimanje video podataka) – 829.440 B
- **`pVideoDisplayBuffer`** (za reprodukciju video podataka) – 829.440 B

Veličina ovih bafera je ista kao veličina bafera u rukovaocu. Video baferi su veličine jedne video slike po 4:2:2 YUV formatu a veličina audio bafera je 8 x 7680B, gde je 7680 broj bajtova za audio podatke koji se odabiru za vreme trajanja jedne slike u video prenosu.

Transfer je realizovan pomoću programskih niti (*thread*). Za svaki tip transfera kreira se posebna nit koja omogućava transfer podataka između aplikacije i rukovaoca i transfer podataka između aplikacije i masovne memorije. Za komunikaciju aplikacije i rukovaoca i prenos podataka između njih koriste se sledeće funkcije:

- **`DeviceIoControl`** – komunikacija aplikacije sa rukovaocem i eventualni prenos manjih količina podataka
- **`ReadFile`** (čitanje) – prenos podataka KA aplikaciji OD rukovaoca
- **`WriteFile`** (pisanje) – prenos podataka OD aplikacije KA rukovaocu

Kod sve tri funkcije je prvi parametar `hDM642Device`, pokazivač na rukovalac preko kojeg aplikacija komunicira sa rukovaocem. Ove funkcije mogu biti blokirajuće i neblokirajuće. Koriste se kao neblokirajuće da bi se omogućio paralelizam svih transfera podataka. Funkcijama `ReadFile` i `WriteFile` se takođe vrši prenos podataka između aplikacije i masovne memorije na analogni način kao sa rukovaocem. Znači kod snimanja signala (*Capture*) imamo čitanje podataka (`ReadFile`) od rukovaoca ka aplikaciji a zatim pisanje podataka (`WriteFile`) od aplikacije na masovnu memoriju. Analogno kod reprodukcije signala (*Playback, Display*) imamo čitanje podataka (`ReadFile`) od masovne memorije ka aplikaciji a zatim pisanje podataka (`WriteFile`) od aplikacije ka rukovaocu.

Pre početka bilo kakvog transfera potrebno je inicijalizovati sistem. Prvo je potrebno učitati izvršni kod u memoriju DSP-a pritiskom na dugme **Load Program To DSP**. Zatim se pokrene izvršavanje DSP programa dugmetom **Run DSP**. Sledeći korak je inicijalizovanje sistema za prenos podataka pritiskom na dugme **Init Audio-Video**. Posle ovog koraka sistem je spreman za prenos podataka.

Realizacija prenosa podataka zasnovana na programskim nitima je objašnjena na primeru snimanja video podataka (*CaptureVideo*). Snimanje video podataka obavlja funkcija `OnCaptureVideo` (*TestDlg.cpp*).

```
void CTestDlg::OnCaptureVideo()
{
   CString sButtonTitleText;
   m_ctlCaptVideo.GetWindowText(sButtonTitleText);
   BOOL stopedVideoCapture;

 1
   if (!isVideoInit)
      {
      <Error>
      return;
       }

 2
   if (sButtonTitleText == "CAPTURE")
      {
       if (CreateVideoCaptureFiles())
          {
          isCaptVideoThreadRunning = TRUE;
          tCaptVideoThread = AfxBeginThread(CaptVideoThreadFunc, ...);

          if (tCaptVideoThread != NULL)
             m_ctlCaptVideo.SetWindowText("STOP");
          else
             {
             CloseVideoCaptureFiles();
             <Error>
             }
          }
```

```
      else
      <Error>
      }
 3
   else
      {
      isCaptVideoThreadRunning = FALSE;
      WaitForSingleObject(tCaptVideoThread, INFINITE);

      DeviceIoControl(hDM642Device, IOCTL_CAPTURE_VIDEO_STOP, ...);
      stopedVideoCapture = GetOverlappedResult(hDM642Device, ...);

      CloseVideoCaptureFiles();

      if (stopedVideoCapture)
         {
         ...
         m_ctlCaptVideo.SetWindowText("CAPTURE");
         }
      else
         <Error>
      }
}
```

1. Proverava se da li je spreman transfer video podataka. Ako nije funkcija završava sa greškom.

2. Provera da li je snimanje aktivno. Ako nije, kreiraju se datoteke za snimanje video podataka funkcijom `CreateVideoCaptureFiles` (*TestDlg.cpp*). Ako su datoteke uspešno kreirane poziva se funkcija `AfxBeginThread` za kreiranje programske niti `tCaptVideoThread` (*TestDlg.h*). Ako nit nije kreirana, zatvaraju se datoteke funkcijom `CloseVideoCaptureFiles` (*TestDlg.cpp*) i prikazuje se poruka greške na ekranu.

3. Ako je snimanje aktivno, zadaje se zahtev zaustavljanja niti funkcijom `WaitForSingleObject` i naredbom `DeviceIoControl` se javlja rukovaocu da zaustavi video tok podataka. Sledi zatvaranje video datoteka i provera da li je rukovaoc zaustavio prenos podataka. Ako rukovaoc nije zaustavio snimanje, prikazuje se na ekranu poruka greške.

Pozivom funkcije `AfxBeginThread` se kreira nit `tCaptVideoThread` i poziva se funkcija `CaptVideoThreadFunc` (*TestDlg.cpp*) koja opisuje funkcionisanje date niti. Ova funkcija predstavlja „telo niti“

```
UINT CaptVideoThreadFunc(...)
{
   DWORD framecount;
   DWORD length = VIDEO_READ_LENGTH;

 1
   VideoCaptOverlapFileY.Offset = DVS_HEADER_SIZE;
   VideoCaptOverlapFileU.Offset = DVS_HEADER_SIZE;
   VideoCaptOverlapFileV.Offset = DVS_HEADER_SIZE;
```

```
   ReadFile(hDM642Device, pVideoCaptureBuffer, length, ...);
   DeviceIoControl(hDM642Device, IOCTL_CAPTURE_VIDEO_START, ...);
   GetOverlappedResult(hDM642Device, ...);

2
   while (isCaptVideoThreadRunning)
     {
     ReadFile(hDM642Device, pVideoCaptureBuffer, length, ...);
     GetOverlappedResult(hDM642Device, ...);

     WriteFile(hVideoCaptFileY, pVideoCaptureBuffer, ...);
     WriteFile(hVideoCaptFileU, pVideoCaptureBuffer, ...);
     WriteFile(hVideoCaptFileV, pVideoCaptureBuffer, ...);

     VideoCaptOverlapFileY.Offset += length/2;
     VideoCaptOverlapFileU.Offset += length/4;
     VideoCaptOverlapFileV.Offset += length/4;
     }

3
   framecount = (GetFileSize(hVideoCaptFileY, ...) - DVS_HEADER_SIZE) /
                FRAMESIZE;

   WriteDVSHeaderOf(hVideoCaptFileY, framecount, ...);
   WriteDVSHeaderOf(hVideoCaptFileU, framecount, ...);
   WriteDVSHeaderOf(hVideoCaptFileV, framecount, ...);

   return(0);
}
```

1. Pomeraju se pokazivači u video datotekama za veličinu zaglavlja DVS fomata. Rukovaocu se pozivom funkcije `ReadFile` šalje zahtev za čitanjem podataka (slike u video prenosu) koji se smeštaju u `pVideoCaptureBuffer` (*TestDlg.h*) (memorijska lokacija za snimanje video slike). Sledi zahtev pokretanja video transfera rukovaocu pomoću funkcije `DeviceIoControl`. Prvi zahtev za čitanjem je pre pokretanja transfera i zbog toga je neblokirajući (asinhroni). Ova veza dva poziva postavlja jedan zahtev za čitanjem u red čekanja kod rukovaoca pre početka transfera. Ovim se realizuje sinhronizacija između prijema i prenosa podataka u sistemu kojom upravlja rukovaoc.

2. Svi sledeći zahtevi za čitanjem (`ReadFile`) se nalaze u petlji koja se izvršava sve dok korisnik ne zatraži zaustavljanje snimanja video podataka. U petlji su zahtevi za čitanjem blokirajući (`GetOverlappedResult`) zbog toga što je sada potrebno da zahtev bude realizovan pre upisivanja u datoteke. Znači mora se sačekati da bude preneta cela video slika i onda se funkcijama `WriteFile` upisuje u video datoteke na spoljnoj memoriji. Na kraju se pomeraju pokazivači u datotekama za veličinu upisanih podataka u svakoj datoteci.

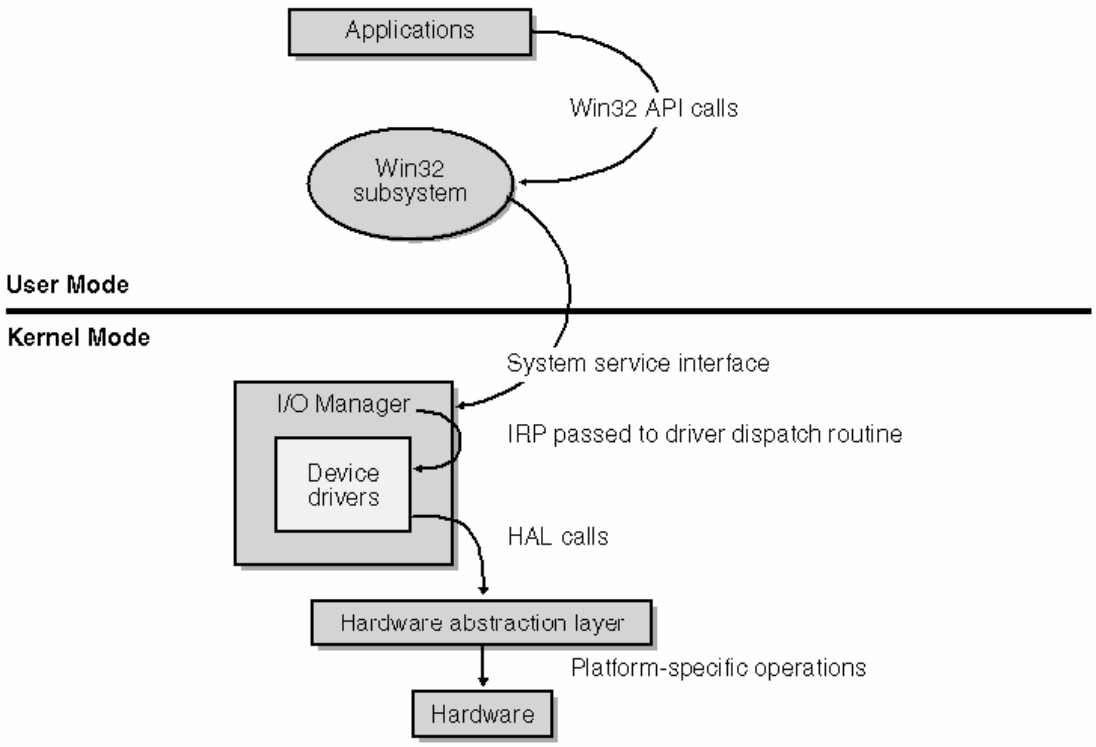
3. Prilikom zaustavljanja snimanja video podataka, prekida se izvršavanje u petlji. Izračunava se broj upisanih video slika `framecount` (*TestDlg.cpp*) na osnovu veličine datoteke. Funkcijama `WriteDVSHeaderOf` (*TestDlg.cpp*) se upisuju zaglavlja u sve tri datoteke za snimanje video podataka, nit završava sa radom i prestaje da postoji.

Kod ponovnog pokretanja snimanja, nit se ponovo kreira. Ostali tipovi transfera su realizovani analogno video prenosu, sa odgovarajućim smerovima prenosa podataka.

Postoje još i dodatne niti za spajanje audio i/ili video toka podataka u povratnu petlju (*loopback*) bez transfera između masovne memorije. Na ovaj način je napravljen prijem podataka sa ulaza EVM ploče do aplikacije i prenos nazad do izlaza na EVM ploči. Ovako se signal sa ulaza EVM ploče dobija odmah na izlazu EVM ploče. Kod *loopback* audio signala posao radi jedna nit dok kod *looback* video signala su to dve niti od kojih prva prima podatke od rukovaoca i prosleđuje drugoj niti koja te podatke prenosi natrag rukovaocu.

6. Rukovalac DM642 EVM karticom[1]

Posmatrano sa tačke gledišta projektovanja rukovalaca, sistem zasnovan na Windows 98 ili Windows XP se sastoji od operativnog sistema i kolekcije rukovalaca za bilo koji uređaj u sistemu. WDM (*Windows Driver Model*) obezbeđuje okvir za rukovaoce koji rade pod operativnim sistemom Windows 98 i Windows XP. Slika 6.1 prikazuje pojednostavljeni funkcionalni dijagram operativnog sistema XP, gde su naglašene osobine bitne za projektovanje rukovalaca



Slika 6.1: Arhitektura operativnog sistema Windows XP

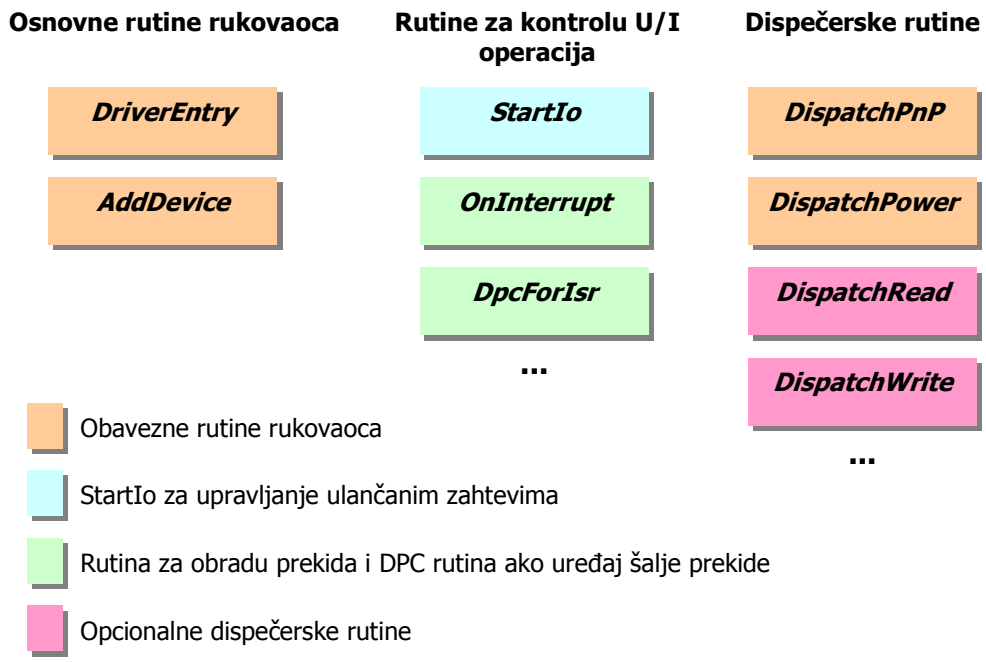
Programska podrška se izvršava ili u korisničkoj zoni rada (*user mode*) koja je ograničena samo na odobrene aktivnosti, ili u jezgru operativnog sistema (*kernel mode*) gde su dozvoljene i privilegovane operacije. Kada je u aplikaciji (*user-mode*) potrebno, napr. da se pročitaju neki podaci iz uređaja, pozove se ReadFile procedura u okviru aplikativne programske sprege (API – *application programming interface*). Podsistem, KERNEL32.DLL, poziva *user-mode* funkciju NtReadFile u okviru sistemske DLL biblioteke, NTDLL.DLL. Ova NtReadFile funkcija preko platformski zavisnog sistemskog servisa poziva *kernel-mode* rutinu istog imena NtReadFile. Rutina NtReadFile je deo sistemske komponente koja se obično zove I/O Manager (U/I Rukovalac). Naziv I/O

Manager ne predstavlja nikakav poseban modul u operativnom sistemu, nego je to skup sistemskih servisa koji okružuju rukovaoca.

Na osnovu U/I zahteva od aplikacije prema rukovaocu, rutine tipa NtReadFile kreiraju IRP paket (*Interrupt Request Packet*), koji se šalje odgovarajućem rukovaocu uređajem. IRP je struktura podataka u operativnom sistemu namenjena komunikaciji sa rukovaocima za obradu U/I zahteva. Rukovaoci se izvršavaju u *kernel-mode* prostoru, koriste HAL (*Hardware Abstraction Layer*) rutine za pristupanje uređaju. HAL rutine za pristup uređaju koriste metode koje su specifične za svaku računarsku konfiguraciju.

6.1 Opis rukovaoca DM642 EVM

Rukovalac zauzima centralno mesto u sistemu za snimanje i reprodukciju digitalnog audio i video signala u realnom vremenu zasnovanog na EVMDM642 platfomi. On rukuje resursima EVM kartice u operativnom sistemu. Na taj način rukovalac upravlja EVM karticom i ima funkciju koordinatora svih tokova podataka. Napravljen je u skladu sa WDM specifikacijom i sastoji se od sledećih rutina:



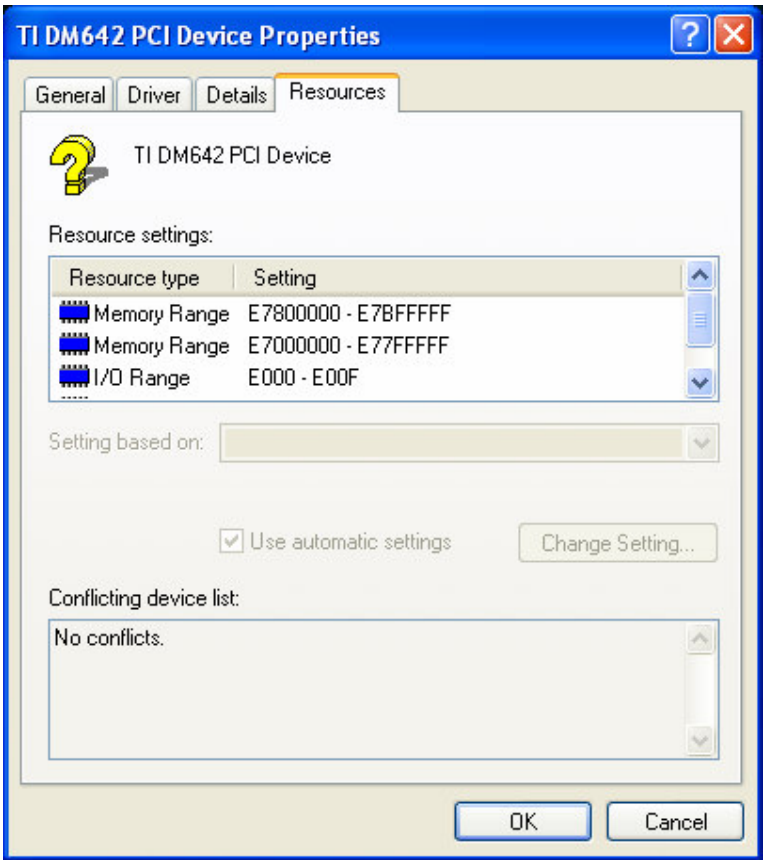
Slika 6.1: Pogled na rukovaoca kao skup rutina

Rukovalac je datoteka tipa .sys koja poseduje zbirku rutina koje sistem poziva kad pristupa uređaju. Rukovalac se učitava prilikom startovanja operativnog sistema ili kad se kartica priključi na računar. Operativni sistem tada poziva `DriverEntry` (*DriverEntry.cpp*) rutinu koja inicijalizuje rukovalac (`DRIVER_OBJECT`), inicijalizuje pokazivač na `AddDevice` (*DriverEntry.cpp*) rutinu i tabelu pokazivača na Dispečerske rutine. Rukovalac za priključivanje periferija (*Plug and Play Manager – PnP Manager*) poziva `AddDevice` rutinu koja inicijalizuje rutine za kontrolu U/I operacija, redove čekanja za U/I zahteve (ukupno 4 zbog potpunog paralelizma svih prenosa podataka) i spregu za komunikaciju sa uređajem. Pošto jedan rukovalac može upravljati sa više uređaja istog tipa, glavna odlika `AddDevice` rutine je da će je PnP Manager pozvati za svaki

takav uređaj u sistemu (`DEVICE_OBJECT`). Unutar `AddDevice` rutine se još poziva i rukovalac napajanjem (*Power Manager*) da inicijalizuje rukovaoca uređajem.

Struktura `DRIVER_OBJECT` predstavlja rukovaoca u operativnom sistemu, struktura `DEVICE_OBJECT` predstavlja svaki uređaj koji je priključen na rukovalac. Struktura `DEVICE_EXTENSION` (*Driver.h*) sadrži podatke koji su jedinstveni za svaku instancu `DEVICE_OBJECT` strukture od uređaja priključenih rukovaocu.

Kada `AddDevice` rutina završi inicijalizaciju, sve je spremno za PnP Managera da dodeli svim rukovaocima odgovarajuće U/I resurse. Na zahtev PnP Managera poziva se funkcija `StartDevice` (*ReadWrite.cpp*) u kojoj rukovaoc identifikuje i inicijalizuje resurse kartice. Resursi su prikazani na slici 6.2

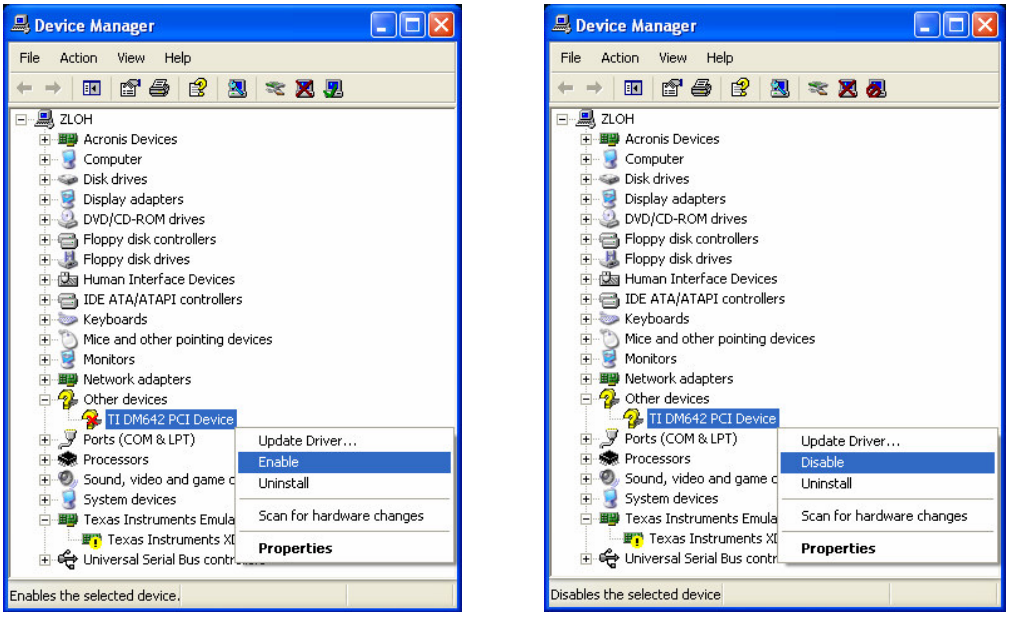


Slika 6.2: Resursi DM642 EVM kartice

Rukovaoc inicijalizuje dva memorijska resursa (memorijske zone u memoriji PC računara preko kojih se pristupa memoriji DSP procesora kroz PCI spregu) za komunikaciju i prenos manjih količina podataka sa DM642 DSP na EVM ploči. Prvi je veličine 4 MB a drugi 8 MB. Veličina U/I resursa je 16 bajta preko kojih rukovaoc pristupa registrima DSP procesora (zovu se PCI U/I registri). Na kraju je vektor za prekide na osnovu kojeg rukovaoc zna da je prekid došao od DM642 EVM platforme. `StartDevice` funkcija alocira i objekte za PCI DMA prenos podataka između rukovaoca i EVM kartice. To su statički baferi u operativnoj memoriji (*CommonBuffer*) koji su zauzeti sve dok je rukovalac aktivan u operativnoj memoriji računara. Ovi baferi se još zovu PCI DMA baferi. *CommonBuffer* je kontinualni prostor u fizičkoj memoriji. Ovaj bafer se koristi za kontinualni DMA prenos podataka preko PCI magistrale. Kako je EVM

kartica u stanju da inicira i kontroliše prenos podataka preko PCI magistrale, u rukovaocu nema posebne rutine za DMA transfer pa je dovoljno alocirati *CommonBuffer* strukture. Svaki *CommonBuffer* ima virtuelnu i fizičku adresu preko kojih rukovaoc i EVM kartica direktno i nezavisno pristupaju baferu. Rukovaoc preko virtuelne a EVM preko fizičke adrese.

Kada I/O Manager pozove funkciju `StopDevice` (*ReadWrite.cpp*) svi resursi se oslobađaju i prekidi onemogućavaju. Funkcije `StartDevice` i `StopDevice` je moguće pozvati i korisnički otvaranjem prozora *DeviceManager* u Windows-u. Desnim klikom na odgovarajuću ikonu rukovaoca pojavljuje se padajući meni. Izbor opcije *Enable* poziva funkciju `StartDevice` a opcija *Disable* poziva `StopDevice` funkciju kao što je prikazano na slici 6.3

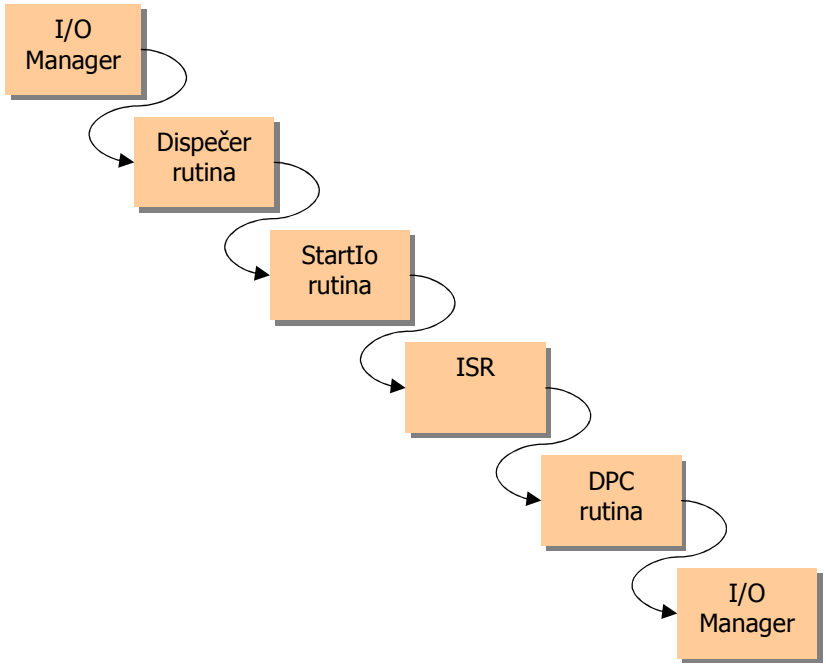


Slika 6.3: *DeviceManager* sa opcijama *Enable* i *Disable*

Kada se funkcija `StartDevice` završi uspešno, rukovalac je spreman za rad. Na osnovu U/I zahteva I/O manageru kreira se IRP paket, koji se šalje odgovarajućem rukovaocu uređajem. Pomoću IRP-a WDM rukovaoci komuniciraju sa rukovaocima uređajem preko dispečerskih rutina. PnP Manager preko `DispatchPnP` rutine (*PlugPlay.cpp*) a Power Manager preko `DispatchPower` rutine (*Power.cpp*). Ulogu raspoređivača procesa (*dispatcher*) igra I/O Manager. Shodno tome i aplikacija svoje U/I zahteve šalje I/O Manageru koji se pretvaraju u IRP i prosleđuju rukovaocu na obradu preko dispečerskih rutina. To su `DispatchRead` (zahtev za čitanjem podataka) i `DispatchWrite` (zahtev za upisom podataka) rutine. `DispatchRead` (*ReadWrite.cpp*) i `DispatchWrite` (*ReadWrite.cpp*) rutine se u rukovaocu dalje obrađuju rutinama za obradu U/I operacija.

Kada se u I/O Manager-u na osnovu zahteva kreira IRP, poziva se odgovarajuća dispečerska rutina. Za potrebe prenosa podataka u svakom rukovaocu postoji red čekanja za ulančavanje U/I zahteva (IRP paketa). Kako u realnom vremenu IRP dolaze nasumično, uvode se redovi zbog sinhronizacije i očuvanja redosleda zahteva. Dispečerska rutina ulančava IRP u red čekanja. Ako uređaj nije zauzet, poziva se `StartIo` rutina koja ima

ulogu da započne obradu sledećeg zahteva u redu. Standardni model za obradu IRP-a na osnovu zahteva aplikacije za prenosom podataka je prikazan na slici 6.4



Slika 6.4: Standardni model obrade IRP

Kada uređaj završi za prenosom podataka, generiše signal prekida. Prekid obrađuje rutina za obradu prekida – ISR (*Interrupt Service Routine*) koja se obično zove `OnInterrupt`. Pošto ISR rutina treba da je što kraća, većinu posla oko obrade prekida koji se ne može ili ne sme obaviti unutar ISR obavlja DPC (*deferred procedure call*) rutina koju poziva ISR (`DpcForIsr`). DPC rutina završava obradu IRP koji se prosleđuje natrag I/O Manager-u. Sa dolaskom novog zahteva ceo proces se ponavlja.

6.1.1 Komunikacija i prenos podataka sa aplikacijom

Funkcija koju aplikacija koristi za komunikaciju i kontrolu rukovaocem je funkcija `DeviceIoControl`. Na poziv ove funkcije, I/O Manager kreira IRP i poziva dispečersku funkciju `DispatchControl`. Preko `DeviceIoControl` funkcije aplikacija zadaje razne komande rukovaocu kao što su inicijalizacija, pokretanje i zaustavljanje transfera, prikaz informacija o rukovaocu, slanje signala prekida ka EVM kartici itd. Svaka od ovih naredbi ima definisani kod (`CTL_CODE`) povezan sa `DispatchControl` funkcijom u okviru koje se izvršava. Definicije ovih kodova se nalaze u zajedničkoj datoteci sa imenom *Ioctls.h* a njihov opis u datoteci *Control.cpp*. Dat je kratak opis ovih kodova:

- **`IOCTL_GET_VERSION`** – informacije o rukovaocu, oznaka verzije itd.
- **`IOCTL_INIT_AUDIO_VIDEO`** – inicijalizuje se struktura `DMA_CONTROLER` (*DMAControl.h*) pozivom funkcije `AllocateDMAChannels` (*DMAControl.cpp*). Inicijalizuju se `AUDIO` strukture za prenos audio podataka (*Audio.h*) funkcijom `InitAudio` (*Audio.cpp*). Inicijalizuju se DMA baferi za smeštanje audio podataka funkcijom `InitAudioDMABuffer` *(Audio.cpp*). Isto tako se inicijalizuju `VIDEO`

strukture za prenos video podataka (*Video.h*) funkcijom `InitVideo` (*Video.cpp*) i DMA baferi za smeštanje video podataka funkcijom `InitVideoDMABuffer` (*Video.cpp*).

- **IOCTL_CLOSE_AUDIO_VIDEO** – oslobađa se struktura `DMA_CONTROLER` pozivom funkcije `ReleaseDMAChannels` (*DMAControl.cpp*).
- **IOCTL_CAPTURE_AUDIO_START** – poziva funkciju `StartAudio` (*Audio.cpp*) koja pokreće snimanje audio signala.
- **IOCTL_CAPTURE_AUDIO_STOP** – poziva funkciju `StopAudio` (*Audio.cpp*) koja zaustavlja snimanje audio signala.
- **IOCTL_PLAYBACK_AUDIO_START** – poziva funkciju `StartAudio` koja pokreće reprodukciju audio signala.
- **IOCTL_PLAYBACK_AUDIO_STOP** – poziva funkciju `StopAudio` koja zaustavlja reprodukciju audio signala.
- **IOCTL_CAPTURE_VIDEO_START** – poziva funkciju `StartVideo` (*Video.cpp*) koja pokreće snimanje video signala.
- **IOCTL_CAPTURE_VIDEO_STOP** – poziva funkciju `StopVideo` (*Video.cpp*) koja zaustavlja snimanje video signala.
- **IOCTL_DISPLAY_VIDEO_START** – poziva funkciju `StartVideo` koja pokreće reprodukciju video signala.
- **IOCTL_DISPLAY_VIDEO_STOP** – poziva funkciju `StopVideo` koja zaustavlja reprodukciju video signala.
- **IOCTL_HOST_TO_DSP** – generiše se signal prekida prema DM642 EVM kartici. Ovim prekidom se pokreću i zaustavljaju svi tipovi transfera za vreme izvršavanja DSP programa. Isto tako se pokreće i izvršavanje DSP programa ako je DSP u stanju reseta.
- **IOCTL_LOAD_PROGRAM** – poziva se funkcija `CoffLoader` (*Coff.cpp*) koja učitava izvršni kod DSP programa u memoriju DSP-a.
- **IOCTL_RESET_DSP** – zaustavlja se izvršavanje DSP programa i DSP prelazi u stanje reseta.

Funkcije za prenos podataka između aplikacije i rukovaoca su:

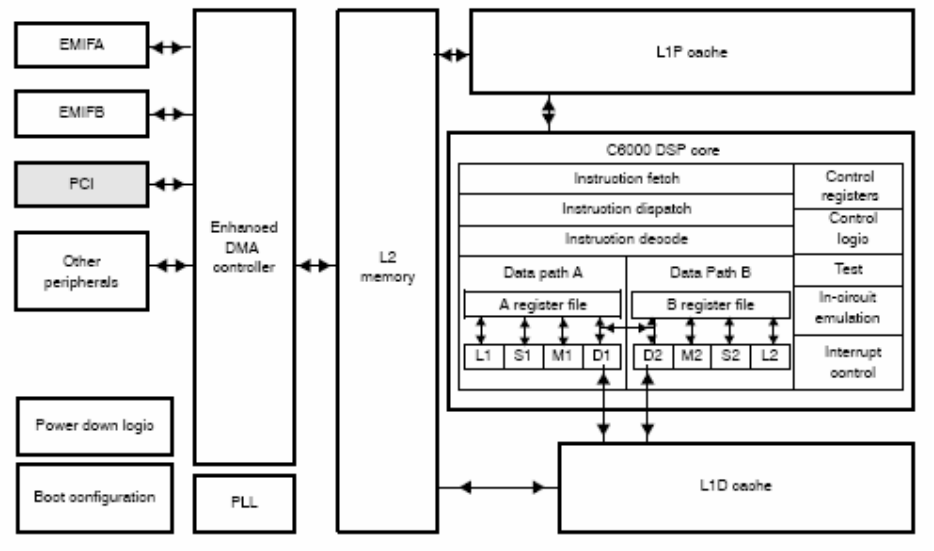
- **ReadFile** (čitanje) – je zahtev za čitanjem podataka. To je transfer podataka od rukovaoca prema aplikaciji
- **WriteFile** (pisanje) – je zahtev za pisanjem podataka. To je transfer podataka od aplikacije prema rukovaocu

Od nekoliko načina za prenos podataka između aplikacije i rukovaoca, ovo rešenje koristi metod sa kopiranjem podataka između bafera u aplikaciji i bafera u kontekstu jezgra operativnog sistema – sistemski bafer (`DO_BUFFERED_IO`).

I/O Manager kreira sistemski bafer jednake veličine kao što je korisnički bafer u aplikaciji. Rukovaoc pristupa sistemskom baferu a I/O Manager obavlja kopiranje podataka između sistemskog i korisničkog bafera.

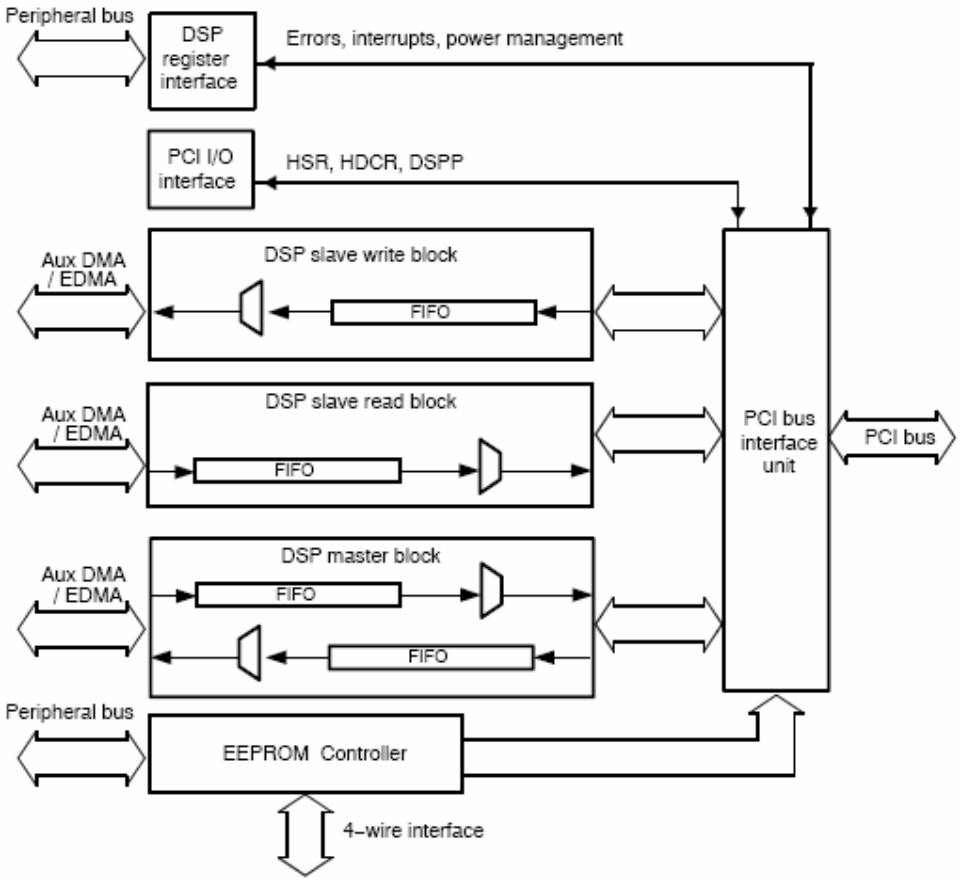
6.1.2 Komunikacija i prenos podataka sa DM642 DSP[6]

EVM DM642 je PCI kartica što znači da komunikacija i prenos podataka između matičnog PC računara i kartice ide preko PCI magistrale. Na slici 6.5 je blok šema TMS320C64x DSP familije procesora



Slika 6.5: Blok šema TMS320C64x DSP

Prenos podataka između PCI sprege i jezgra DSP (C6000 CPJ) obavlja EDMA (*Enhanced DMA*) kontroler. EDMA raspolaže sa 64 nezavisna kanala za prenos podataka. Na slici 6.6 je blok šema PCI sprege



Slika 6.6: Blok šema PCI sprege

Prenos podataka u realnom vremenu između PCI magistrale i EDMA kanala ide preko 4 FIFO bafera koji povećavaju efikasnost transfera. Svaki FIFO bafer je veličine 16 reči (reč je veličine 32 bita) odnosno 64 bajta. Dva FIFO bafera se koriste kada PCI transferom upravlja matični PC računar (*DSP Slave*) a druga dva kada PCI transferom upravlja DSP procesor (*DSP Master*) koji se koristi u ovom rešenju. Smer prenosa podataka od DSP-a do rukovaoca se zove DSP pisanje (*DSP master writes*) a smer od rukovaoca do DSP-a se zove DSP čitanje (*DSP master reads*). Maksimalna veličina pojedinačnog PCI transfera je 64KB.

Prilikom učitavanja (`StartDevice`), rukovalac alocira 4 PCI DMA bafera u operativnoj memoriji za PCI transfer koja su zauzeta sve dok je rukovalac aktivan u operativnom sistemu. Ovi baferi su podeljeni na jednake delove (blokove) koji se prenose kružno preko PCI magistrale. Jedan blok odgovara jednom elementu PCI transfera. Ovako podeljeni baferi se još zovu kružni baferi (*Ring Buffer*). Kružni baferi se koriste da bi se lakše obezbedila sinhronizacija transfera bez gubitka podataka. Za audio transfer veličina kružnih bafera je 61.440 bajtova (60KB). Podeljeni su na 8 blokova veličine 7.680 bajtova. Za video transfer veličina kružnih bafera je 829.440 bajtova (810KB). Podeljeni su na 16 blokova veličine 51.840 bajtova zbog maksimalnog ograničenja jednog PCI prenosa od 64KB.

Komunikacija koja ide PCI magistralom se odvija preko registara DSP procesora koji se zovu PCI registri. PCI sprega omogućava vidljivost cele memorije DSP-a preko tri bazna registra (Baza 0, 1, 2):

- **Baza 0**: 4M-bajtni memorijski prozor cele memorije DSP-a preko registra DSPP (DSP page register)
- **Baza 1**: 8M-bajtna mapa za memorijski mapirane registre DSP procesora
- **Baza 2**: 16-bajtni U/I prostor sa DSP U/I registrima koji se koriste od strane PC računara

Ova tri bazna registra pripadaju grupi PCI konfiguracijskih registara. Inicijalizuju se prilikom startovanja rukovaoca (slika 6.2). DM642 ima tri tipa PCI registara:

- PCI konfiguracijski registri – dostupni samo matičnom računaru. Sadrže standardne informacije o konfiguraciji. Ovde pripadaju i gore spomenuti bazni registri.

- PCI U/I registri – takođe dostupni samo matičnom računaru preko baznog registra broj 2 (Baza 2). To su sledeći registri:

  - **HSR** (Host status register) – preko polja ovog registra rukovalac proverava da li je DSP generisao prekid, omogućava prekide i blokira prekide
  - **HDCR** (Host-to-DSP control register) – preko ovog registra rukovalac generiše prekid prema DSP-u i resetuje DSP
  - **DSPP** (DSP page register) – rukovalac preko memorijskog prozora od 4MB "vidi" memoriju DSP procesora gde je početna adresa u DSPP registru

- Memorijski mapirani PCI registri – dostupni DSP-u ali i matičnom računaru preko baznog registra broj 1 (Baza 1):

  - **RSTSRC** (DSP reset source/status register) – DSP generiše prekid prema rukovaocu a rukovalac resetuje bit DSP prekida da bi se omogućio sledeći prekid.

- **PCIIS** (PCI interrupt source register) – DSP proverava izvor prekida, završetak PCI prenosa, upravlja PCI transferom.
- **PCIIEN** (PCI interrupt enable register) – DSP omogućava generisanje prekida koje kontroliše proverom PCIIS registra.
- **DSPMA** (DSP master address register) – sadrži odredišnu adresu podataka kada DSP čita preko PCI magistrale a izvornu adresu kada DSP piše preko PCI magistrale.
- **PCIMA** (PCI master address register) – sadrži izvornu adresu kada DSP čita preko PCI magistrale a odredišnu adresu kada DSP piše preko PCI magistrale.
- **PCIMC** (PCI master control register) – sadrži 3 bita za pokretanje čitanja ili pisanja preko PCI magistrale i 16 bita za određivanje koliko bajtova se prenosi PCI magistralom (maksimum 64K-1 bajtova (65535B) po jednom prenosu).

Komunikacija i prenos manjih količina podataka koji ide PCI magistralom se odvija preko ovih registara na dva načina:

Prvi način je pristupom rukovaoca memoriji DSP procesora preko DSPP registra. Ovako rukovalac može direktno da čita i upisuje podatke u memoriju DSP-a. Na ovaj način je realizovano prebacivanje izvršnog koda DSP programa u njegovu memoriju. Isto tako je direktnim pristupom memoriji realizovana kontrola i upravljanje tokovima podataka preko PCI magistrale u celom sistemu za snimanje i reprodukciju digitalnog audio i video signala. Napravljena je struktura `PCI_DMA_Controler` (*dma_pci.h*), koja predstavlja logičko rešenje realnog DMA kontrolera. Struktura `PCI_DMA_Controler` i *dma_pci.h* datoteka su deo programske podrške za DSP. Kreira je DSP program na samom početku izvršavanja i nalazi se u ISRAM memoriji DSP-a. Kako je rukovalac u stanju da direktno pristupa memoriji DSP-a, funkcija `AllocateDMAChannels` alocira strukturu `DMA_CONTROLER` u operativnoj memoriji u koju se iskopiraju sve adrese iz `PCI_DMA_Controler` strukture. Sada DSP može pristupati direktno `PCI_DMA_Controler` strukturi u ISRAM memoriji a rukovalac preko `DMA_CONTROLER` strukture adresa. `PCI_DMA_Controler` se sastoji od kanala `PCI_DMA_Channel` (*dma_pci.h*) a kanali od deskriptora `PCI_DMA_Descriptor` (*dma_pci.h*). Analogno se `DMA_CONTROLER` sastoji od kanala `DMA_CHANNEL` (*DMAControl.h*) a kanali od deskriptora `DMA_DESCRIPTOR` (*DMAControl.h*). Svaki kanal opisuje jedan tip transfera zajedno sa njegovim PCI DMA baferom (ukupno 4 kanala) a svaki deskriptor nekog kanala opisuje jedan blok u odgovarajućem PCI DMA baferu (audio kanali imaju 8 deskriptora a video kanali 16).

Drugi način je generisanjem prekida pomoću ovih registara. Prekidi se obrađuju u ISR rutini u rukovaocu i u ISR rutini u DSP programu. Prekid od rukovaoca ka EVM kartici (HDCR registar) pokreće i zaustavlja izvršavanje DSP programa a ako je DSP program pokrenut onda se ovim prekidom pokreće i zaustavlja prenos podataka preko PCI magistrale. Ovi prekidi se obrađuju u ISR rutini u okviru programske podrške za DSP. Prekidi od EVM kartice prema rukovaocu (RSTSRC registar) se obrađuju u ISR rutini u rukovaocu i služe za komunikaciju i sinhronizaciju prenosa podataka između DSP-a i rukovaoca. Pošto DM642 EVM upravlja PCI transferom, ovim prekidima javlja rukovaocu da je završio prenos jednog bloka podataka preko PCI magistrale.

6.2 Prebacivanje izvršnog koda u memoriju DSP

Prvi korak inicijalizacije sistema za snimanje i reprodukciju je zahtev **Load Program To DSP** u aplikaciji koji poziva funkciju `OnLoadProgram` za prebacivanje izvršnog koda u memoriju DSP-a. U funkciji `OnLoadProgram` se preko `DeviceIoControl` funkcije sa kodom `IOCTL_LOAD_PROGRAM` poziva funkcija u rukovaocu `CoffLoader` koja učitava izvršni kod u memoriju DSP-a. Zaglavlje ovog formata datoteka je definisano u datoteci *CoffDef.h*.

```
NTSTATUS CoffLoader(PDEVICE_EXTENSION pdx, PUSHORT CoffFilename)
{
   HANDLE hCoffFile;
   ULONG filesize;
   PUCHAR buffer = NULL;
   BOOLEAN Ok = FALSE;
   NTSTATUS status;
   OBJECT_ATTRIBUTES ObjectAttributes;
   FILE_STANDARD_INFORMATION FileInformation;
   UNICODE_STRING Filename;
1
   RtlInitUnicodeString(&Filename, CoffFilename);
   InitializeObjectAttributes(&ObjectAttributes, &Filename,...);

2
   if (CoffFilename)
      {
      status = ZwCreateFile(&hCoffFile, &ObjectAttributes, ...);

      if (!NT_SUCCESS(status))
         KdPrint(<Error>);
      else
         {
         status = ZwQueryInformationFile(hCoffFile,
                   FileInformation, ...);
         filesize = (ULONG) FileInformation.EndOfFile.LowPart;

         if ( (!NT_SUCCESS(status)) || (filesize <= 0) )
            KdPrint(<Error>);
3
         else
            {
            buffer = (PUCHAR) ExAllocatePool(NonPagedPool, filesize);

            if (buffer)
               {
               status = ZwReadFile(hCoffFile, buffer, filesize, ...);

               if (!NT_SUCCESS(status))
                  KdPrint(<Error>);
               else
                  Ok = TRUE;
               }
           else
               KdPrint(<Error>);
           }
```

```
4
        status = ZwClose(hCoffFile);

        if (!NT_SUCCESS(status))
            {
            KdPrint(<Error>);
            Ok = FALSE;
            }
        }
    }
5
  if (Ok)
     Ok = (CoffScan(pdx, buffer) == LOADER_SUCCESS);

  if (buffer)
     ExFreePool((PULONG) buffer);

  if (Ok)
     status = STATUS_SUCCESS;

  return status;
}
```

1. Obavezna inicijalizacija za otvaranje Coff datoteke.

2. Ako je ime Coff datoteke validno, otvara se datoteka funkcijom `ZwCreateFile`, kojoj će se dalje pristupati preko pokazivača `hCoffFile`.

3. Ako je datoteka uspešno otvorena, alocira se memorija u koju se funkcijom `ZwReadFile` upisuje sadržaj datoteke.

4. Zatvara se datoteka funkcijom `ZwClose`.

5. Ako su sve dosadašnje operacije bile uspešne, poziva se funkcija `CoffScan` (*Coff.cpp*). `CoffScan` ispisuje zaglavlje Coff datoteke i proverava sve njegove delove, dužinu, oznake, adrese itd. Ako je sve u redu poziva funkciju `MemcopyTo` (*Coff.cpp*) koja deo po deo iz alocirane memorije u rukovaocu upisuje sadržaj Coff datoteke u memoriju DSP procesora. Na kraju se oslobađa alocirana memorija i funkcija `CoffLoader` vraća vrednost u zavisnosti od uspeha svih dosadašnjih operacija.

```
VOID MemcopyTo(PDEVICE_EXTENSION pdx, PVOID dst, PVOID src, int count)
{
   ULONG page, offset;
   ULONG old_page;
   int size, remaining;

 1
   old_page = READ_PORT_ULONG((PULONG) DSPP);

 2
   do {
       page = (ULONG) dst >> 22;
       offset = (ULONG) dst & (DM642_PAGE_SIZE - 1);
```

```
      WRITE_PORT_ULONG((PULONG) DSPP, page);

      remaining = DM642_PAGE_SIZE - offset;
      size = (count > remaining) ? remaining : count;
 3
      RtlCopyMemory(pdx->membase0 + offset, src, size);

      dst = (PULONG) dst + size;
      src = (PULONG) src + size;
      count -= size;

   } while (count != 0);

 4
   WRITE_PORT_ULONG((PULONG) DSPP, old_page);
}
```

1. Sačuvati trenutnu vrednost DSPP registra koja se na kraju obnavlja pomoću funkcije `READ_PORT_ULONG`.

2. Ulazak u petlju koja izračunava koji deo koje stranice treba da se upiše u memoriju DSP-a. Postavljanje nove vrednosti DSPP registra pomoću funkcije `WRITE_PORT_ULONG`. Iz petlje se izlazi kad je zahtevani broj bajtova upisan u memoriju.

3. Obnavljanje sačuvane vrednosti DSPP registra i izlazak iz funkcije.

6.3 Transfer podataka

Transfer podataka u rukovaocu je objašnjen na primeru snimanja video podataka (*CaptureVideo*). Na samom početku se prebacuje izvršni kod DSP programa u njegovu memoriju. Onda se pokreće izvršavanje DSP programa pritiskom na dugme **Run DSP** u aplikaciji. Poziva se `DeviceIoControl` sa kodom `IOCTL_HOST_TO_DSP` koja pokreće izvršavanje. Sada rukovalac i DSP program čekaju na zahtev za prenos podataka od aplikacije.

Prenos podataka u rukovaocu počinje sa pozivom funkcije `DeviceIoControl` sa kodom `IOCTL_CAPTURE_VIDEO_START` unutar funkcije `CaptVideoThreadFunc`. Ovaj poziv pokreće funkciju `StartVideo` koja javlja DSP programu da pokrene kanal za snimanje video podataka tako što upisuje u `PCI_DMA_Controler` strukturu znak za pokretanje odgovarajućeg kanala i šalje signal prekida ka DSP-u. Poziv funkcije `ReadFile` koji se takođe nalazi u funkciji `CaptVideoThreadFunc` je zahtev I/O Manager-u da kreira IRP za dobijeni zahtev za čitanjem podataka i pozove dispečersku rutinu `DispatchRead` u rukovaocu:

```
NTSTATUS DispatchRead(PDEVICE_OBJECT fdo, PIRP Irp)
{
   PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

 1
   ULONG length =
           IoGetCurrentIrpStackLocation(Irp)->Parameters.Read.Length;
```

```
2
   IoMarkIrpPending(Irp);

3
   if (length == AUDIO_CAPTURE_BUFSIZE)
      StartPacket(&pdx->dqAudioRead, fdo, Irp, OnCancelRead);
   else
      StartPacket(&pdx->dqVideoRead, fdo, Irp, OnCancelRead);

   return STATUS_PENDING;
}
```

1. Na osnovu IRP paketa se određuje kolika je zahtevana količina bajtova od aplikacije. Pomoću ove vrednosti je rukovalac u stanju da odredi koje je čitanje podataka u pitanju (audio ili video) pošto za oba tipa se koristi ista `DispatchRead` rutina.

2. Aplikaciji se vraća signal da je započeta obrada pristiglog IRP paketa da bi mogla da nastavi dalje sa radom (neblokirajući zahtev).

3. Ovde se određuje o kojem tipu transfera se radi i na osnovu toga se poziva odgovarajuća funkcija `StartPacket` (*DevQueue.cpp*). Pošto je reč o video prenosu, proverava se da li je uređaj zauzet obradom video zahteva. Ako jeste, dobijeni IRP se uključuje u red čekanja na obradu `dqVideoRead` (*DevQueue.h*) a ako nije poziva se funkcija `StartIoVideoRead` (*ReadWrite.cpp*):

```
VOID StartIoVideoRead(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
{

1
   PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
   PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
   NTSTATUS status = IoAcquireRemoveLock(&pdx->RemoveLock, Irp);

2
   if (!NT_SUCCESS(status))
       {
       KdPrint(<Error>);
       CompleteRequest(Irp, status, ...);
       return;
       }

3
   pdx->vncopy = 0;
   pdx->numxferVideoRead = 0;
   pdx->nbytesVideoRead = stack->Parameters.Read.Length;
   pdx->xferVideoRead = pdx->nbytesVideoRead;

   pdx->vaddrVideoRead = Irp->AssociatedIrp.SystemBuffer;

4
   CopyFromVideoCaptureBuffer(pdx, pdx->xferVideoRead);
}
```

1. Inicijalizuje pokazivač na `DEVICE_EXTENSION` strukturu koja sadrži sve informacije o jednom uređaju. Uzima se iz reda tekući IRP paket i zaključava se rukovalac sve dok traje obrada preuzetog IRP (`IoAcquireRemoveLock`). Zaključavanje je zbog konzistentnosti operacija u operativnom sistemu. Ako bi kojim slučajem rukovalac bio uklonjen za vreme obrade nekog IRP paketa, IRP zahtev i aplikacija bi ostali u nedefinisanom stanju.

2. Ako zbog nečega rukovalac ne može biti zaključan, funkcija `StartIoVideoRead` završava obradu IRP sa porukom greške i izlazi.

3. Ako je zaključavanje uspelo, inicijalizuju se promenljive za upravljanje prenosom podataka. Preuzima se pokazivač na sistemski bafer koji je kreirao I/O Manager za potrebe prenosa.

4. Poziva se funkcija `CopyFromVideoCaptureBuffer` (*Video.cpp*) koja kopira blokove, koji su do ovog momenta preneti preko PCI magistrale u bafer za snimanje video podataka `VideoCaptureBuffer` (*Driver.h*), u sistemski bafer. Funkcija ažurira broj prenetih i broj preostalih blokova u zahtevu i završava sa radom.

Transfer je završen tek kad se prenese kompletni broj zahtevanih bajtova u sistemski bafer. Za svaki preneti blok u `VideoCaptureBuffer`, DSP šalje signal prekida ka rukovaocu. Operativni sistem na pojavu signala prekida poziva odgovarajuću ISR rutinu (`OnInterrupt`) koja je povezana sa tim prekidom:

```
BOOLEAN OnInterrupt(PKINTERRUPT InterruptObject, PDEVICE_EXTENSION pdx)
{
   PIRP Irp = NULL;
   BOOLEAN isAudioRead = FALSE;
   BOOLEAN isAudioWrite = FALSE;
   BOOLEAN isVideoRead = FALSE;
   BOOLEAN isVideoWrite = FALSE;

 1
   ULONG hsr = READ_PORT_ULONG((PULONG) HSR);

   if (!(hsr & HSR_INTERRUPT_PENDING))
      {
      KdPrint(<False>);
      return FALSE;
      }

   ULONG rstsrc = READ_REGISTER_ULONG((PULONG) RSTSRC);
   WRITE_REGISTER_ULONG((PULONG) RSTSRC, rstsrc |=
                        RSTSRC_INTERRUPT_RESET);
 2
   if (!(pdx->DMACtrlAllocated))
      return TRUE;

 3
   ULONG irq_source = READ_REGISTER_ULONG((PULONG)
           (pdx->membase0 + pdx->DmaControler.irq_source_addr));

   if (irq_source & VIDEO_CAPTURE_INT)
      {
```

```
      WRITE_REGISTER_ULONG((PULONG)
           (pdx->membase0 + pdx->DmaControler.irq_source_addr),
                irq_source &= ~VIDEO_CAPTURE_INT);
      isVideoRead = TRUE;
      VideoCaptureInterrupt(pdx, &pdx->VideoCapture);
      }

   if (irq_source & VIDEO_DISPLAY_INT)
      ...
   if (irq_source & AUDIO_CAPTURE_INT)
      ...
   if (irq_source & AUDIO_PLAYBACK_INT)
      ...
```

4

```
   if ( (!pdx->busyAudioRead) && (!pdx->busyAudioWrite) &&
        (!pdx->busyVideoRead) && (!pdx->busyVideoWrite) )
      return TRUE;
```

5

```
   if (isVideoRead)
      {
      Irp = GetCurrentIrp(&pdx->dqVideoRead);

      if (Irp != NULL)
         {
         NTSTATUS status;

         if (Irp->Cancel)
            status = STATUS_CANCELLED;
         else
            status = AreRequestsBeingAborted(&pdx->dqVideoRead);

         if (!NT_SUCCESS(status))
            {
            pdx->busyVideoRead = FALSE;
            Irp->IoStatus.Status = status;
            Irp->IoStatus.Information = 0;
            }
         else
            {
            Irp->IoStatus.Status = STATUS_SUCCESS;
            Irp->IoStatus.Information = pdx->numxferVideoRead;
            }

         KeInsertQueueDpc(&pdx->VideoReadDpc, NULL, pdx);
         }
      }
```

6

```
   if (isVideoWrite)
      ...
   if (isAudioRead)
      ...
   if (isAudioWrite)
      ...

   return TRUE;
}
```

1. Provera da li je prekid stigao od uređaja koji pripada rukovaocu. Ako nije, izlazi se iz rutine. Ako jeste, briše se indikator prekida u registru da bi se omogućio dolazak sledećeg.

2. Ako je prekid došao u trenutku kada struktura `DMA_CONTROLER` još nije inicijalizovana, znači da rukovalac nije u stanju da obavlja kontrolu i prenos podataka.

3. Pročita se indikator izvora prekida pomoću `DMA_CONTROLER` (za koji kanal je vezan prekid) i proverava se tip prenosa. Ako je to kanal za čitanje video podataka, briše se odgovarajući indikator, da bi se omogućio dolazak novog. Postavlja se bulova promenljiva za čitanje video podataka i poziva se funkcija `VideoCaptureInterrupt` (*Video.cpp*), koja pomoću `DMA_CONTROLER` strukture označi DSP-u da je stigao prekid i ažurira pokazivač primljenih blokova. Na osnovu primljenih blokova funkcija `CopyFromVideoCaptureBuffer` kopira blokove u sistemski bafer. Ostali tipovi prenosa se proveravaju analogno opisanom čitanju video podataka.

4. Ako je došao prekid od DSP a rukovalac trenutno nije zauzet obradom ni jednog zahteva, odmah se izlazi iz ISR rutine.

5. Ulazi se u blok za čitanje video podataka. Uzima se prvi u redu IRP i ako nije prazan nastavlja se sa njegovom obradom. Ako je od aplikacije zatraženo poništenje zahteva, postavlja se indikator za poništenje zahteva. U zavisnosti da li je redovna obrada zahteva ili zatraženo poništenje, postavljaju se oznake u IRP paket i poziva se procedura sa odloženim pozivom `DpcForVideoRead` (*ReadWrite.cpp*) da završi obradu IRP paketa.

6. Po potrebi se ulazi u blokove za obradu svih tipova transfera koja se odvija analogno opisanom čitanju video podataka.

Sledi opis procedure koja radi detaljniju obradu IRP paketa ili ako je obrada završena, kompletira IRP zahtev i otključava rukovaoca. Procedura `DpcForVideoRead`:

```
VOID DpcForVideoRead(PKDPC Dpc, ..., PDEVICE_EXTENSION pdx)
{
   NTSTATUS status;
   ULONG numxfer;
1
   PIRP Irp = GetCurrentIrp(&pdx->dqVideoRead);

   if (Irp == NULL)
      return;
2
   status = Irp->IoStatus.Status;
   pdx->numxferVideoRead = pdx->vncopy;
   pdx->nbytesVideoRead = pdx->xferVideoRead;

3
   if (pdx->nbytesVideoRead && NT_SUCCESS(status))
      {
      CopyFromVideoCaptureBuffer(pdx, pdx->xferVideoRead);
```

```
4
      if (pdx->xferVideoRead == 0)
          {
          pdx->busyVideoRead = FALSE;
          pdx->numxferVideoRead += pdx->nbytesVideoRead;
          numxfer = pdx->numxferVideoRead;
          Irp->IoStatus.Status = STATUS_SUCCESS;
          Irp->IoStatus.Information = numxfer;

          StartNextPacket(&pdx->dqVideoRead, fdo);

          CompleteRequest(Irp, STATUS_SUCCESS, numxfer);
          IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
          }
      }
5
   else
       {
       pdx->busyVideoRead = FALSE;
       numxfer = pdx->numxferVideoRead;

       StartNextPacket(&pdx->dqVideoRead, fdo);

       CompleteRequest(Irp, status, numxfer);
       IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
       }
}
```

1. Preuzima se tekući IRP sa reda zahteva. Ako je red prazan, izlazi se iz procedure.

2. Ako je IRP validan, ažurira se broj preostalih bajtova i broj prenetih bajtova za dalju obradu zahteva.

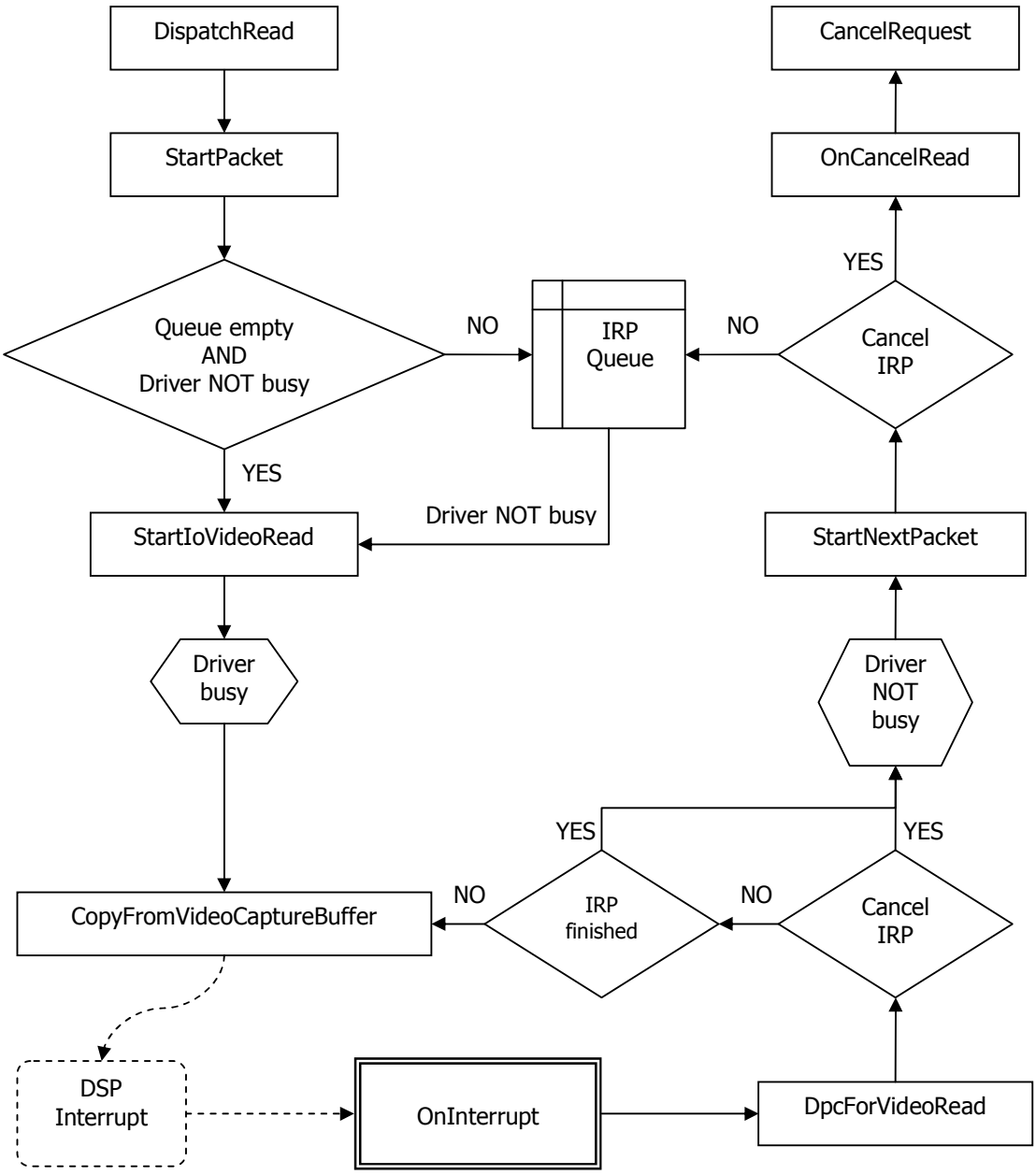
3. Ako je broj preostalih bajtova veći od nule, poziva se ponovo funkcija `CopyFromVideoCaptureBuffer` da iskopira nove blokove podataka koje je DSP preneo preko PCI magistrale.

4. Ako je funkcija `CopyFromVideoCaptureBuffer` prenela sve preostale blokove, ponovo se ažurira broj prenetih bajtova, IRP se označava kao završen i poziva se funkcija `StartNextPacket` (*DevQueue.cpp*) za obradu sledećeg IRP paketa u redu zahteva. Sledi kompletiranje zahteva funkcijom `CompleteRequest` (*DriverEntry.cpp*) i otključavanje rukovaoca pozivom `IoReleaseRemoveLock`. Ako nisu preneti svi blokovi, izlazi se iz procedure i čeka se novi prekid (znak od DSP da je preneo nove blokove preko PCI magistrale).

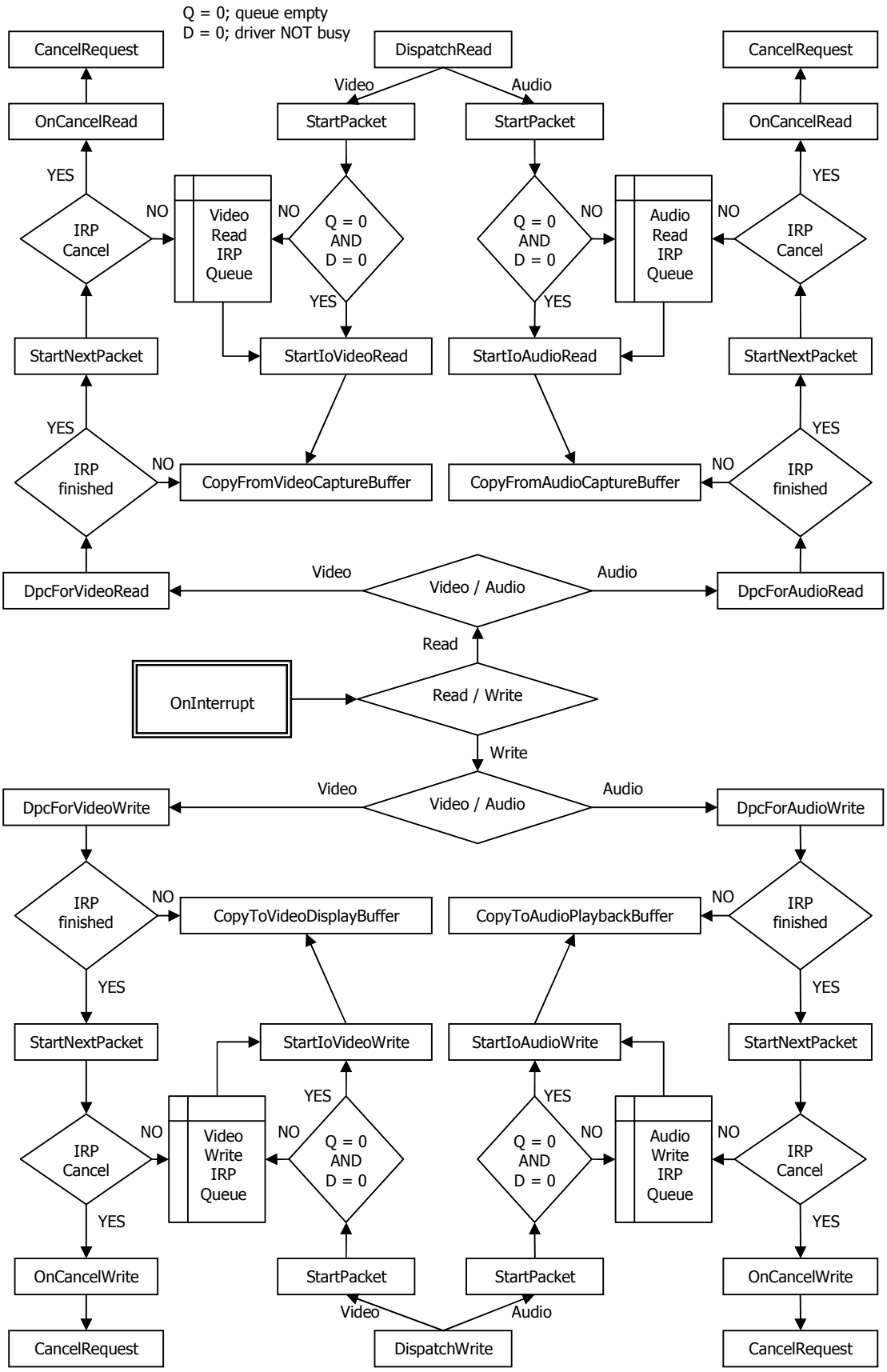
5. Ako je kod 3. broj preostalih bajtova nula ili je zatraženo poništenje zahteva, prosleđuje se broj prenetih bajtova i pokreće se obrada sledećeg IRP-a. Tekući IRP se kompletira, ali kao poništen i otključava se rukovalac.

Za ostale tipove transfera, mehanizam poziva rutina i prenosa podataka izgleda analogno *Video Capture* transferu.

Vidi se da su pozivi rutina StartIo, ISR i DPC usko povezani obradom IRP paketa. Ova zavisnost se može grafički prikazati na sledeći način:



Slika 6.7: Blok šema obrade zahteva za čitanjem video podataka



Slika 6.8: Blok šema obrade svih tipova transfera podataka

7. Programska podrška za DM642

Prvi korak inicijalizacije sistema za snimanje i reprodukciju digitalnog audio i video signala je prebacivanje izvršnog koda DSP programa u internu memoriju DSP-a. Uloga programske podrške za DSP (DSP programa) je prenos podataka od I^2S/BT.656 sprege na DM642 EVM kartici do memorije (ISRAM, SDRAM) i od memorije preko PCI magistrale do operativne memorije matičnog PC računara (PCI DMA baferi u rukovaocu). DSP program inicijalizuje EVMDM642 karticu, inicijalizuje periferije na EVM kartici preko kojih se odvijaju tokovi podataka, alocira memoriju za smeštanje audio (ISRAM) i video (SDRAM) podataka.

Programska podrška za DSP je napravljena u razvojnom okruženju CCStudio (poglavlje 6).

PCI_DMA_Controler struktura i dinamički objekti se kreiraju u programskom kodu DSP-a. Objekti koji se koriste za prenos podataka i sinhronizaciju su statički objekti napravljeni u DSP/BIOS konfiguraciji (*AVStreaming.cdb*).

7.1 Komunikacija sa rukovaocem

Za komunikaciju sa rukovaocem se kreira `PCI_DMA_Controler` (*dma_pci.h*) struktura u internoj memoriji DSP-a koja se sastoji od 4 kanala (2 audio kanala i 2 video kanala). Audio kanal ima 8 deskriptora a video kanal ima 16 deskriptora. Svaki deskriptor se sastoji od tri polja. Polje `busAddress` koje sadrži fizičku adresu određenog bloka kružnog bafera u rukovaocu za prenos podataka. Ove adrese upisuje rukovalac prilikom inicijalizacije `DMA_CONTROLER` strukture i nepromenljive su sve dok su alocirani PCI DMA kružni baferi. Polje `dspAddress` sadrži adresu bafera unutar memorije DSP za prenos podataka. Ove adrese se upisuju dinamički u toku izvršavanja DSP programa zbog ping-pong baferovanja. Treće polje, `cntInfo`, sadrži broj bajtova za prenos preko PCI magistrale i dodatne informacije o bloku koji se prenosi.

7.2 DSP/BIOS konfiguracija[7]

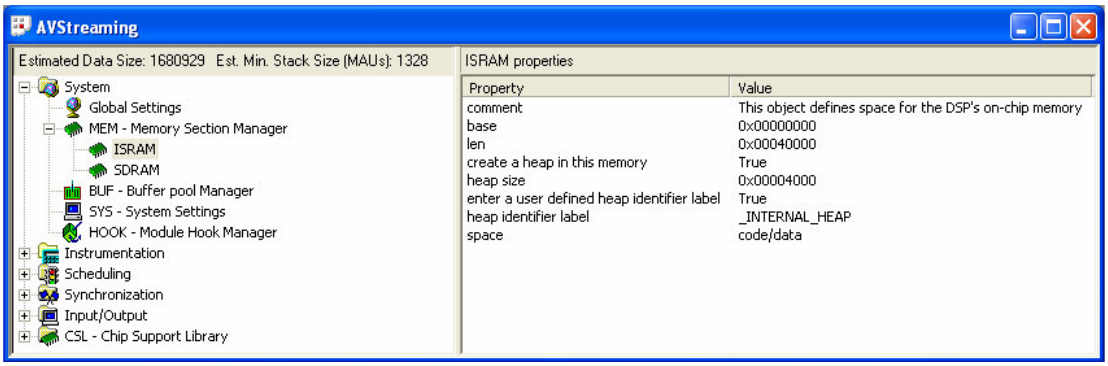
Za strukture podataka, raspodelu resursa, komunikaciju i sinhronizaciju tokova podataka u realnom vremenu korišćeni su sledeći moduli DSP/BIOS:

- **MEM** (*Memory Section Manager*) – Modul za alokaciju memorijskih segmenata
- **LOG** (*Event Log Manager*) – Modul za vođenje dnevnika izvršavanja programa
- **HWI** (*Hardware Interrupt Service Routine Manager*) – Modul koji upravlja rutinama za obradu hardverskih prekida

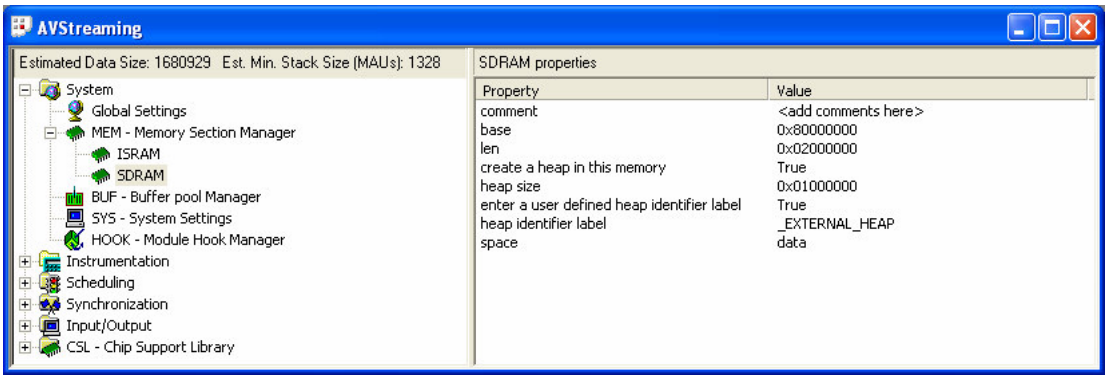
- **SWI** (*Software Interrupt Manager*) – Modul koji upravlja funkcijama za obradu softverskih prekida
- **TSK** (*Task Manager*) – Modul koji upravlja programskim zadacima (*task*) koji su realizovani u zasebnim programskim nitima (*thread*)
- **SEM** (*Semaphore Manager*) – Modul za sinhronizaciju tokova podataka
- **QUE** (*Atomic Queue Manager*) – Modul za kreiranje redova čekanja
- **PIP** (*Buffered Pipe Manager*) – Modul za sinhronizaciju i prenos podataka u realnom vremenu pomoću programskih cevi (*Pipe*) koje upravljaju baferima u memoriji i sadrže internu sinronizaciju
- **EDMA** (*Enhanced Direct Memory Access*) – Poboljšani DMA kontroler sa 64 nezavisna kanala

7.2.1 Modul MEM

Modul koji upravlja alokacijom memorije. Potrebno ga je konfigurisati da bi povezilac programa (*linker*) znao u koji segment memorije treba smestiti koji deo programske podrške. U okviru DSP programa je iskorišćena interna (ISRAM) memorija i eksterna (SDRAM) memorija. ISRAM je veličine 256KB a SDRAM 32MB. Postoje dva segmenta memorije za kreiranje dinamičkih promenljivih tzv. HEAP. U ISRAM-u je alocirana veličina segmenta 16KB (`_INTERNAL_HEAP`) a u SDRAM-u je alocirano 16MB (`_EXTERNAL_HEAP`). U ISRAM memoriji se nalaze: izvršni kod programa, instance korišćenih DSP/BIOS objekata i baferi programskih cevi za prenos audio podataka. U SDRAM memoriji se zbog veličine alociraju baferi programskih cevi za prenos video podataka.



Slika 7.1: ISRAM segment MEM modula



Slika 7.2: SDRAM segment MEM modula

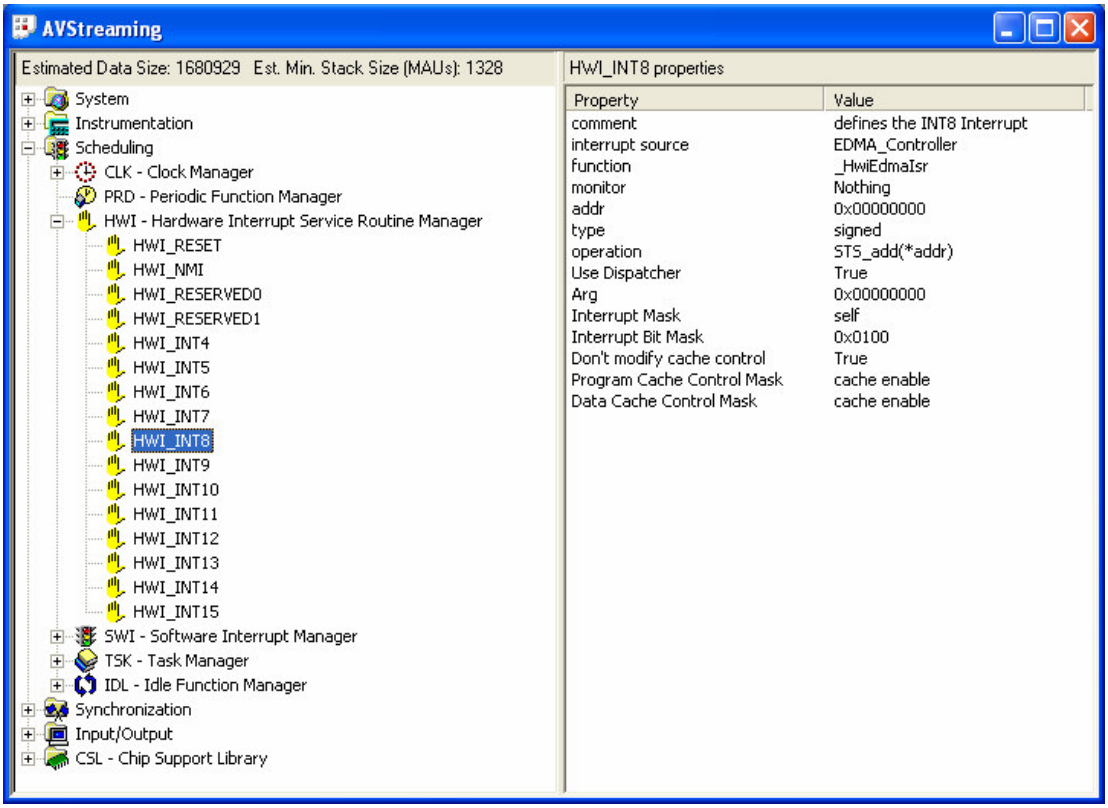
7.2.2 Modul LOG

Modul za vođenje dnevnika izvršavanja programa. Pomoću njega se može pratiti i izvršavanje programa u realnom vremenu u prozoru CCStudia. Ime kreiranog objekta je `LogMain`.

7.2.3 Modul HWI

Ovaj modul upravlja rutinama za obradu hardverskih prekida. U okviru programske podrške koriste se prekidi:

- **HWI_INT8** – prekid generiše EDMA kontroler koji signalizira da je završen transfer posredstvom EDMA kontrolera (audio i/ili video podataka). Ove prekide obrađuje rutina `_HwiEdmaIsr` (*pci.c*).
- **HWI_INT10** – prekid generiše VP2 (Video priključak 2) kao signal da je završen prenos jedne slike video signala iz FIFO bafera za prikaz video signala ka BT.656 spregi (reprodukcija video signala). Prekide obrađuje rutina `_HwiVPDispIsr` (*vportdis.c*).
- **HWI_INT11** – prekid generiše VP1 (Video priključak 1) kao signal da je završen prenos jedne slike video signala od BT.656 sprege prema FIFO baferima (snimanje video signala). Prekide obrađuje `_HwiVPCaptIsr` (*vportcap.c*) rutina.
- **HWI_INT13** – prekidi koje generišu rukovalac i PCI transfer a obradu vrši rutina `_HwiPCIIsr` (*pci.c*).

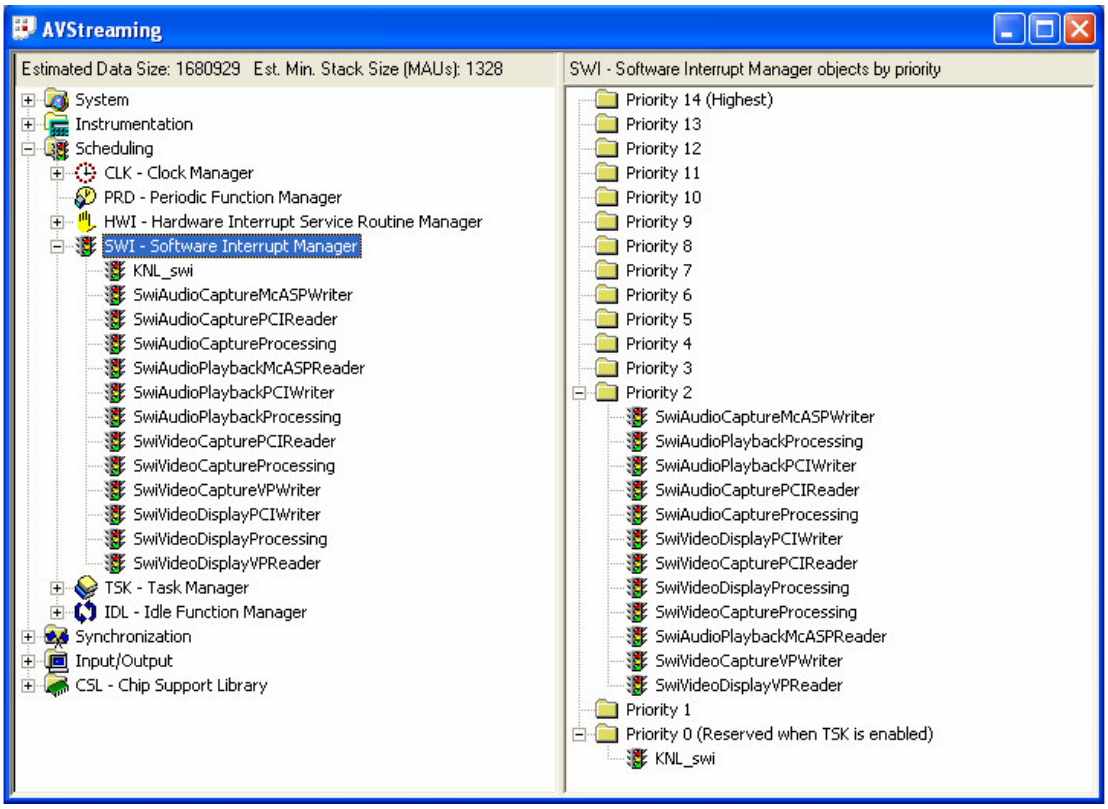


Slika 7.3: Modul HWI

Pošto su ove rutine napisane u programskom jeziku C, za njihovo pozivanje se mora koristiti sistemski raspoređivač procesa (*dispatcher*).

7.2.4 Modul SWI

Modul SWI upravlja softverskim prekidima. U okviru programske podrške je korišćeno 12 objekata ovog tipa. Po 3 objekta za svaki tip prenosa podataka. Za svaki SWI objekat se vezuje njegova funkcija za obradu prekida. Napravljene su tri funkcije za transfer zbog veće robusnosti sistema i bolje sinhronizacije. Preko ove tri funkcije prenos podataka ide redno (kad prva završi obradu, poziva drugu i prosleđuje rezultat a druga poziva treću i prosleđuje rezultat). Slika 7.4 prikazuje SWI modul sa kreiranim objektima koji imaju prioritet 2.

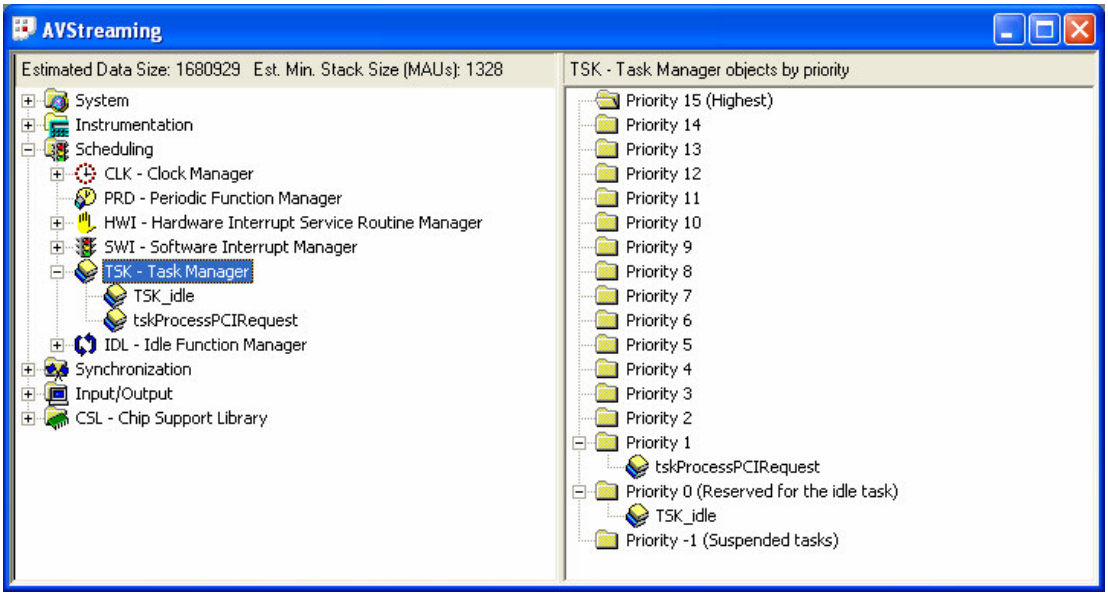


Slika 7.4: Modul SWI

7.2.5 Modul TSK

Modul TSK služi za upravljanje odvojenim zadacima u programu. Svaki zadatak (*task*) se izvršava u okviru posebne programske niti (*thread*).

Kako su i rutine za obradu softverskih prekida posebni procesi, razlika između njih je u tome što su SWI rutine obavezno neblokirajuće a TSK može biti po potrebi blokiran i čekati na neki događaj.

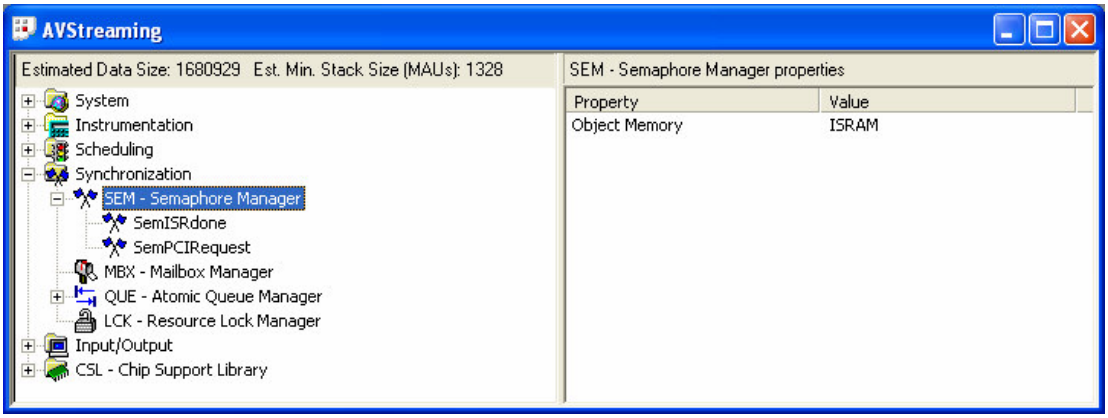


Slika 7.5: Modul TSK

U ovom rešenju se koristi jedan TSK objekat, `tskProcessPCIRequest`. Za svaki TSK se vezuje funkcija (telo) koja opisuje njegovo ponašanje. Funkcija `tskProcessPCIRequestFunc` (*pci.c*) obavlja u beskonačnoj petlji obradu zahteva prenosa podataka iz reda `PCIReqQue`. Podešava parametre transfera, izvornu i odredišnu adresu i pokreće transfer. Na ulazu u petlju čeka na dva događaja. Prvi je završetak aktivnog transfera (semafor `SemISRdone`) a drugi je postavljanje novog zahteva u red čekanja `PCIReqQue` (semafor `SemPCIRequest`). `tskProcessPCIRequest` ima prioritet 1 a SWI objekti prioritet 2, znači da svaka obrada softverskog prekida ima prednost nad PCI transferom. Svaki SWI javlja pomoću `SemPCIRequest` semafora da je postavljen novi zahtev na red čekanja.

7.2.6 Modul SEM

Jedan od modula za sinhronizaciju je SEM modul. Semafori se koriste za sinhronizaciju TSK objekata.

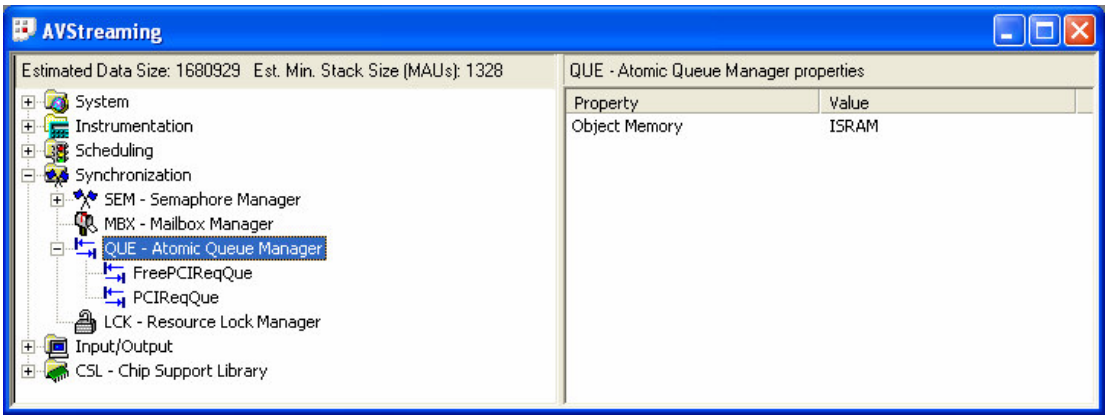


Slika 7.6: Modul SEM

U okviru programske podrške se za sinhronizaciju tokova podataka koriste SWI rutine, TSK proces i semafori. SWI objekti su neblokirajući ali mogu da signalizuju događaj a TSK proces čeka na događaj da bi nastavio sa obradom. Semaforom `SemISRdone`, ISR rutina `HwiPCIIsr` signalizira TSK objektu završetak prenosa podataka preko PCI magistrale. Tada `tskProcessPCIRequest` može da pokrene novi transfer.

7.2.7 Modul QUE

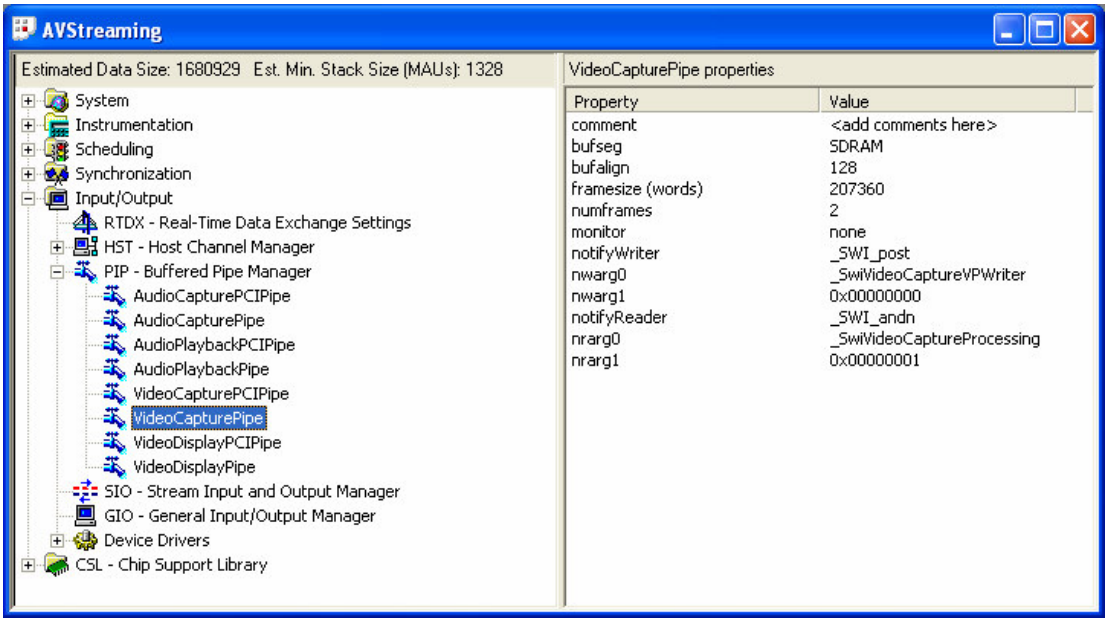
Modul QUE upravlja redovima za čekanje koji podržavaju atomske operacije (pristup redu je zaštićen od prekida). U okviru programske podrške se koriste dva reda za sve zahteve prenosa podataka preko PCI magistrale. Red za aktivne zahteve je `PCIReqQue` a za slobodne zahteve služi red `FreePCIReqQue`.



Slika 7.7: Modul QUE

7.2.8 Modul PIP

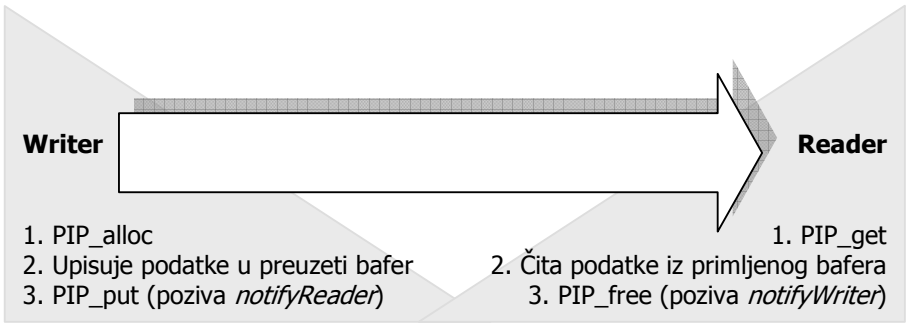
Za transfer podataka u realnom vremenu se koristi PIP modul (programske cevi).



Slika 7.8: Modul PIP

Tok podataka u okviru programske podrške je napravljen kombinacijom programskih cevi i SWI objekata.

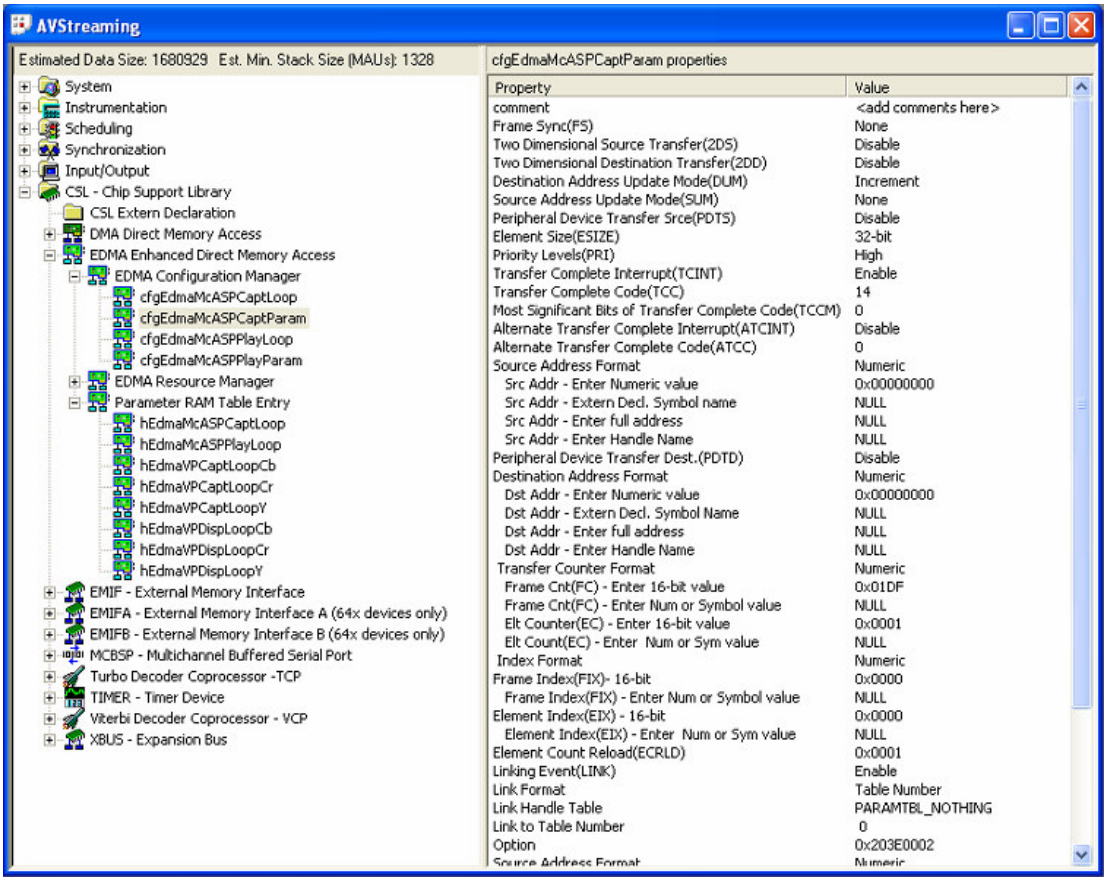
Programska cev (*Pipe*) sadrži dva bafera za podatke koji se koriste po principu ping-pong baferovanja (*Ping-Pong Buffering*). Programskim cevima se prenose podaci između PCI sprege i audio/video priključaka na EVM kartici. Sadrže pokazivače na dve SWI funkcije koje upravljaju programskim cevima. To su *notifyWriter* i *notifyReader* funkcije. Funkcija *notifyWriter* podešava programsku cev za upisivanje podataka. Funkcija *notifyReader* se poziva kada se završi upisivanje i podešava bafer za čitanje podataka i postavlja zahtev u `PCIReqQue`. Svaki tip transfera je realizovan sa dve programske cevi koje su spregnute jednom zajedničkom SWI funkcijom. Znači, *notifyWriter* priprema prvu programsku cev za upis podataka. Pokreće se *notifyReader* koja kopira podatke iz prve programske cevi u drugu programsku cev. Ona je u isto vreme i *notifyWriter* funkcija za drugu programsku cev. Na kraju se poziva *notifyReader* koja priprema drugu programsku cev za čitanje podataka.



Slika 7.9: Funkcionisanje programske cevi (*Pipe*)

7.2.9 Modul EDMA[9]

C64x EDMA može da obezbedi preko 2 GB/s propusne moći. EDMA raspolaže sa 64 kanala koji se aktiviraju nezavisnim događajima. Na raspolaganju stoje ukupno 85 skupova parametara u parametarskoj RAM memoriji (PaRAM) za povezivanje (*linking*) i ulančavanje (*chaining*). Povezivanje omogućava da EDMA automatski obavi niz prenosa podataka na jedan spoljni događaj. Ulančavanje omogućava da jedan EDMA kanal aktivira drugi po završetku prenosa podataka. Povezivanje i ulančavanje omogućavaju kontinualne DMA operacije koje se automatski inicijalizuju i aktiviraju. Samo početna postavka parametara je potrebna od strane programske podrške. Sa ovim osobinama moguće je izvesti kružne bafere, ping-pong bafere i prenos kompleksnih struktura podataka (višedimenzionalnih). Kombinacijom jednodimenzionalnog (1D) i dvodimenzionalnog (2D) prenosa se mogu prenositi slike koje EDMA automatski prepliće ili raspliće što je u ovom projektu i iskorišćeno.



Slika 7.10: Modul EDMA

U okviru programske podrške se koriste 8 EDMA kanala za prenos podataka između memorije DSP i periferija na EVM ploči. Po 1 kanal za prenos audio podataka i po 3 kanala za prenos video podataka. Svaka video komponenta se prenosi zasebnim kanalom (Y, Cb i Cr). Svaki video kanal je podešen za prenos jedne komponente za jednu polusliku (*field 1, field 2*). Za kontinualni prenos je iskorišćena mogućnost povezivanja EDMA kanala i automatsko preplitanje i rasplitanje, tako da se na jednom kraju EDMA prenosa nalaze vremenski raspodeljeni video podaci u progresivnom režimu po BT.656 formatu a sa druge strane se nalazi kompletna prepletena slika (*frame*).

7.3 Konfiguracija EVMDM642 modula

Pokretanjem izvršavanja DSP programa se poziva glavna (*Main*) funkcija. Ona inicijalizuje EVM karticu, inicijalizuje audio kodek, video koder i video dekoder, McASP priključak za prenos audio podataka, dva video priključka za prenos video podataka, alocira memoriju za privremeno smeštanje video slika, kreira PCI_DMA_Controler strukturu i redove za zahteve prenosa podataka preko PCI magistrale. Na početku izvršavanja se inicijalizuju i DSP/BIOS objekti.

7.3.1 I²C sprega[10]

I²C magistrala na DM642 EVM ploči je idealna kao sprega za kontrolu registara eksternih modula na ploči. Kod DM642 EVM se I²C magistrala koristi za konfigurisanje video kodera, video dekodera i stereo audio kodeka. Format magistrale je prikazan na slici 7.11

Start	Slave Address	W	ACK	Sub Address	ACK-S	Data	ACK-S	Stop

Write Sequence

Start	Slave Address	W	Sub Address	ACK-S	
Start	Slave Address	R	Data	ACK-S	STOP

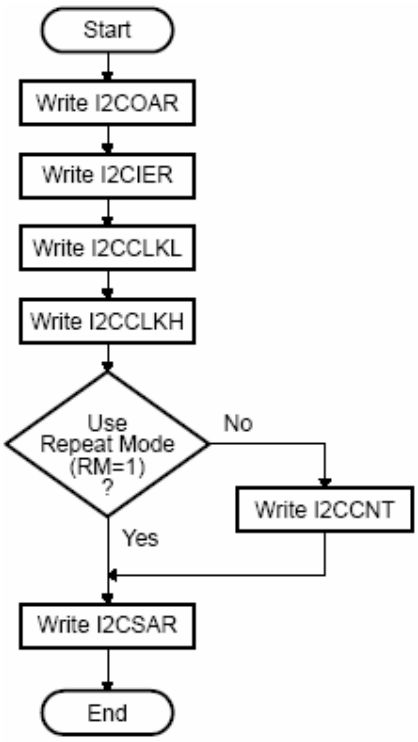
Read Sequence

Slika 7.11: Format I²C magistrale

I²C adrese periferija na kartici su date u tabeli (slika 7.12)

Device	Address	R/W	Function
TVP5146	0xBA	R/W	Capture 1 Decoder
TVP5150A	0xB8	R/W	Capture 2 Decoder
SAA7105	0x88	R/W	Encoder
TLV320AIC23B	0x1A	R/W	CODEC
24WC256	0x50	R/W	I²C EEPROM

Slika 7.12: Memorijska mapa I²C



Slika 7.13: Procedura za podešavanje I²C modula

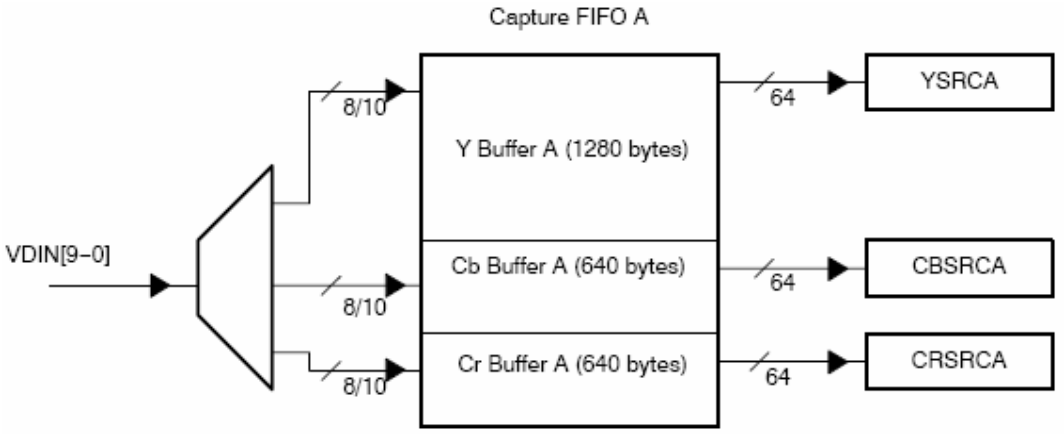
Unutar *Main* funkcije se pozivaju tri funkcije koje vrše inicijalizaciju video dekodera, video kodera i stereo audio kodeka na EVM kartici preko I^2C magistrale:

- Funkcija `InitAIC23` (*aic23.c*) podešava AIC23 stereo audio kodek. Analogni ulaz/izlaz je stereo audio signal. Digitalni izlaz/ulaz je stereo audio signal sa frekvencijom odabiranja od 48000 Hz, 16-bitne rezolucije sa veličinom odabirka (*sample*) od 32 bita.
- Funkcija `InitTVP5150` (*tvp5150.c*) podešava TVP5150 video dekoder. Analogni ulaz je kompozitni prepleteni (*interlaced*) video signal sa aktivnom rezolucijom 720x576 (PAL 625/50 video sistem). Na izlazu daje digitalni video signal YCbCr 4:2:2 formata 8-bitne rezolucije po ITU-R BT.656 preporuci.
- Funkcija `InitSAA7105` (*saa7105.c*) podešava SAA7105 video koder. Video koder je podešen analogno video dekoderu.

7.3.2 Video priključci[11]

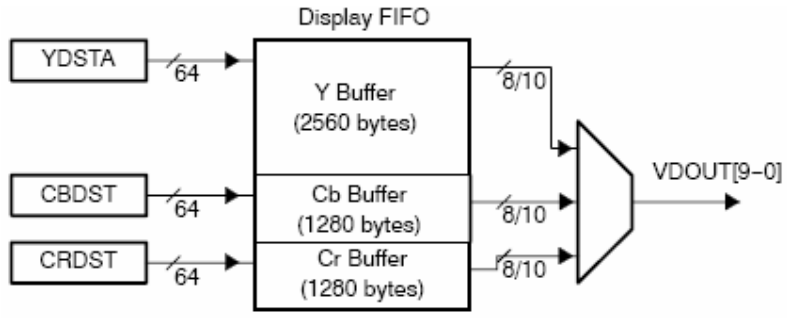
Video priključci su sprega DM642 sa video dekoderima i video koderom na EVM ploči. U programskoj podršci se koristi video priključak 1 za snimanje video signala, koji je sprega sa TVP5150A video dekoderom i video priključak 2 za reprodukciju video signala koji je sprega sa SAA7105 video koderom preko FPGA mreže.

Video priključak 1 je podešen tako da na ulazu dobija digitalni video signal YCbCr 4:2:2 formata rezolucije 8-bita po ITU-R BT.656 preporuci. Za BT.656 mod, FIFO video priključka podeljen je na dva kanala (A i B) od kojih se koristi samo A u okviru programske podrške. Za svaki kanal, FIFO je podeljen na Y, Cb i Cr bafere sa posebnim adresama za čitanje podataka. Y bafer (YSRCA) je veličine 1280 bajtova, Cb (CBSRCA) i Cr (CRSRCA) baferi su veličine 640 bajtova. Konfiguracija i veličina FIFO bafera je prikazana na slici 7.14



Slika 7.14: Konfiguracija FIFO za BT.656 mod snimanja video signala

Video priključak 2 je podešen tako da na izlazu daje kontinualni digitalni video signal YCbCr 4:2:2 formata sa 8-bitnom rezolucijom po ITU-R BT.656 preporuci. Jedini izlaz je preko kanala A, gde je FIFO takođe podeljen na posebne bafere za Y, Cb i Cr komponente sa adresama za upis podataka. Y bafer (YDSTA) je veličine 2560 bajtova a Cb (CBDST) i Cr (CRDST) baferi su veličine 1280 bajtova što se vidi na slici 7.15



Slika 7.15: Konfiguracija FIFO za BT.656 mod prikazivanje video signala

Video priključci su predviđeni za rad u sprezi sa EDMA transferima za prenos podataka između FIFO bafera video priključka i eksterne (SDRAM) ili interne (ISRAM) memorije. EDMA događaj se generiše automatski kada se FIFO dovoljno napuni i/ili isprazni do unapred zadatog praga (*threshold*). Jedino je potrebno podesiti parametre EDMA kanala za ispravno funkcionisanje ovog prenosa. Sledeći primer podešavanja parametara je za kanal Y-komponente snimanja video signala. Ova funkcija se poziva za svaki EDMA kanal. Analogno rešenje je za reprodukciju.

```
Uns ConfigVPCaptureEdmaChannel(EDMA_Handle *hEdmaCha, Uint32 eventId,
                               Uint32 srcAddr, Uint32 dstAddr,
                               Uint32 frameCount,Uint32 elementCount,
                               EDMA_Handle *hEdmaParamTbl,
                               EDMA_Handle *hEdmaLoop)
{
   Int32 tcc = 0;
   Int i;
   EDMA_Config  cfgEdma;

 1
   *hEdmaCha = EDMA_open(eventId, EDMA_OPEN_RESET);
   if (*hEdmaCha == EDMA_HINV)
      return 0;

   if ((tcc = EDMA_intAlloc(-1)) == -1)
      return 0;

 2
   cfgEdma.opt = EDMA_OPT_RMK(
                 EDMA_OPT_PRI_HIGH,
                 EDMA_OPT_ESIZE_32BIT,
                 EDMA_OPT_2DS_NO,
                 EDMA_OPT_SUM_NONE,
                 EDMA_OPT_2DD_YES,
                 EDMA_OPT_DUM_INC,
                 EDMA_OPT_TCINT_NO,
                 EDMA_OPT_TCC_OF(tcc & 0xF),
                 EDMA_OPT_TCCM_OF(((tcc & 0x30) >> 4)),
                 EDMA_OPT_ATCINT_NO,
                 EDMA_OPT_ATCC_OF(0),
                 EDMA_OPT_PDTS_DISABLE,
                 EDMA_OPT_PDTD_DISABLE,
                 EDMA_OPT_LINK_YES,
                 EDMA_OPT_FS_NO);
```

```
3
  cfgEdma.src = EDMA_SRC_RMK(srcAddr);
  cfgEdma.dst = EDMA_DST_RMK(dstAddr);
  cfgEdma.cnt = EDMA_CNT_RMK(EDMA_CNT_FRMCNT_OF(frameCount - 1),
                EDMA_CNT_ELECNT_OF(elementCount));
  cfgEdma.idx = EDMA_IDX_RMK(EDMA_IDX_FRMIDX_OF(elementCount * 8),
                EDMA_IDX_ELEIDX_OF(0));
  cfgEdma.rld = EDMA_RLD_RMK(EDMA_RLD_ELERLD_OF(0), *hEdmaLoop);

  EDMA_config(*hEdmaCha, &cfgEdma);
  EDMA_config(*hEdmaLoop, &cfgEdma);

4
  EDMA_allocTableEx(4, hEdmaParamTbl);

  for (i = 2; i < 4; i++)
      {
      EDMA_config(hEdmaParamTbl[i], &cfgEdma);
      EDMA_FSETH(hEdmaParamTbl[i], OPT, TCINT, EDMA_OPT_TCINT_YES);
      }

  for (i = 0; i < 2; i++)
      {
      cfgEdma.rld = EDMA_RLD_RMK(0, hEdmaParamTbl[i + 2]);
      EDMA_config(hEdmaParamTbl[i], &cfgEdma);
      EDMA_FSETH(hEdmaParamTbl[i], OPT, TCINT, EDMA_OPT_TCINT_NO);
      }
5
  ...
  EDMA_intHook(tcc, HwiEdmaIsr);

  return 1;
}
```

1. Otvara i resetuje kanal `hEdmaCha` sa datim brojem `eventId`. Takođe alocira `tcc` broj signala završetka transfera.

2. Postavljaju se parametri u EDMA konfiguraciji: visoki prioritet, 32-bitna veličina elementa transfera, 1D transfer na ulazu bez ažuriranja izvorne adrese, 2D transfer na izlazu sa ažuriranjem odredišne adrese, bez generisanja prekida, broj signala završetka transfera i mogućnost povezivanja.

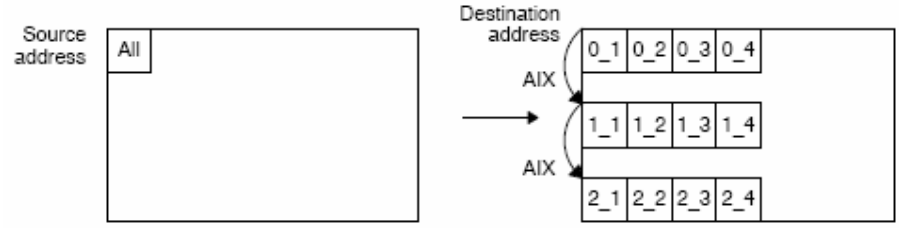
3. Postavljaju se ostali parametri: `srcAddr` je izvorna fiksirana FIFO adresa transfera (u ovom slučaju YSRCA bafer), `dstAddr` je odredišna adresa koja se ažurira (bafer u memoriji), `frameCount` je broj prenosa po jednom sinhronizacionom signalu ili broj linija jedne poluslike (288), `elementCount` je broj elemenata u jednom `frameCount` prenosu ili broj bajtova za jednu liniju slike video signala ali u 32-bitnom formatu (720 / 4 = 180). Ova konfiguracija se dodeljuje `hEdmaCha` EDMA kanalu i `hEdmaLoop` pokazivaču na prazni prenos EDMA kanala (zbog sinhronizacije).

4. Alociraju se 4 skupa parametara `hEdmaParamTbl` u EDMA parametarskom RAM-u. Gornja dva se podešavaju tako da generišu signal završetka transfera prema DSP a donja dva se povezuju sa gornja dva i isključuje se generisanje signala

završetka transfera. Ovako se generiše prekid tek po završenom prenosu druge poluslike (završetak prenosa jedne cele slike).

5. Povezuje se signal završetka EDMA transfera (prekid) sa rutinom za obradu EDMA prekida `HwiEdmaIsr`.

Konfiguracija EDMA parametara u primeru je 1D-do-2D transfer, gde je izvorna adresa fiksirana (SUM = 00) a odredišna adresa se ažurira (DUM = 01). Broj `frameCount` je 3 a `elementCount` je 4, kako je prikazano na slici 7.16

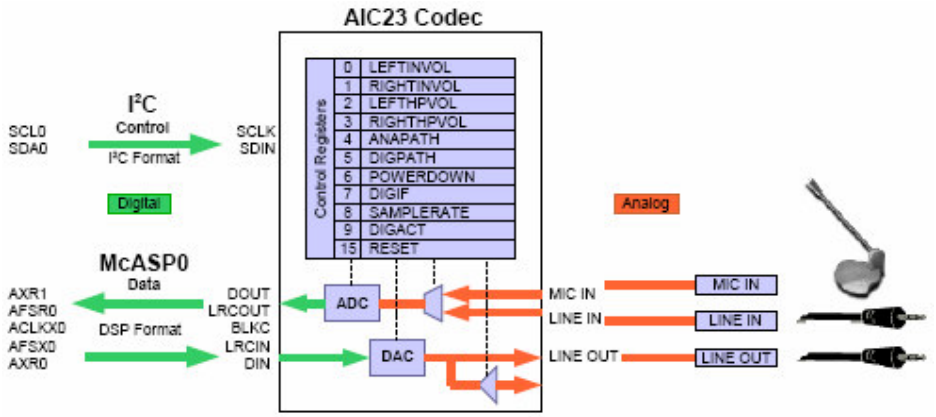


Slika 7.16: Transfer tipa 1D-do-2D sa SUM = 00 i DUM = 01

Za ostale komponente snimanja video signala je ista konfiguracija EDMA kanala. Za prenos komponeti video signala kod reprodukcije je analogno 2D-do-1D transfer sa SUM = 01 i DUM = 00.

7.3.3 McASP sprega[12]

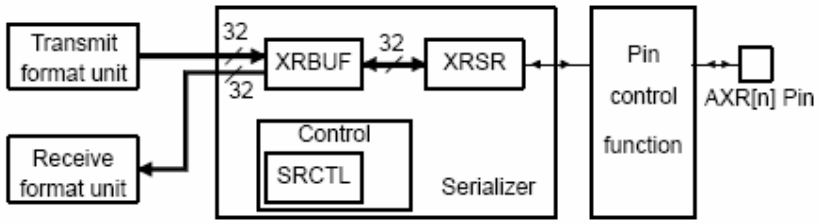
Na standardnoj EVM konfiguraciji su video priključak 0 i video priključak 1 programirani tako da mogu da se podele da bi omogućili implementiranje McASP funkcija kao i spregu sa AIC23 stereo audio kodekom. Višekanalni serijski audio priključak (McASP) podržava I2S protokol prenosa podataka na spoljnoj strani sa AIC23 kodekom. Podešen je u okviru programske podrške za rad u TDM (*time-division multiplexed*) režimu sa 2 kanala (stereo) koji je saglasan I2S protokolu (poglavlje 10). Komunikacija sa AIC23 audio kodekom ide preko dva kanala, kao što je prikazano na slici 7.17



Slika 7.17: DM642 EVM sprega sa audio kodekom

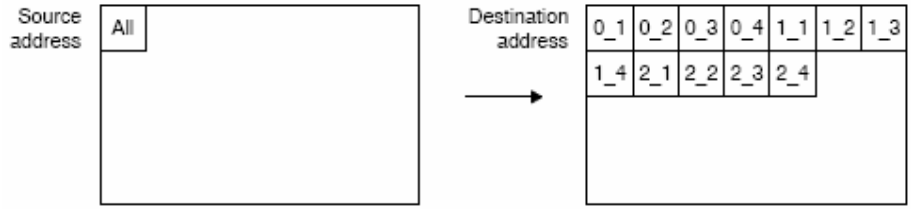
Sa unutrašnje strane, McASP se takođe koristi u sprezi sa EDMA transferima za prenos podataka između bafera McASP sprege i ekterne ili interne (DSP) memorije. Baferi

u McASP sprezi se zovu serijalizeri (*serializers*) koji serijski pomeraju podatke unutra ili napolje iz McASP sprege. Sastoje se iz pomeračkog registra (XRSR) i bafera (XRBUF) čija adresa je fiksirana, kao kod FIFO bafera u video priključku.



Slika 7.18: Izgled jednog McASP serijalizera

Konfiguracija EDMA parametara za snimanje audio signala je 1D-do-1D transfer gde `frameCount` je 3 a `elementCount` je 4, kao na slici 7.19



Slika 7.19: Transfer tipa 1D-do-1D sa SUM = 00 i DUM = 01

Izvorna adresa je fiksirana a odredišna adresa se ažurira. Za reprodukciju audio signala je podešavanje analogno snimanju, 1D-do-1D transfer sa SUM = 01 i DUM = 00.

7.4 Transfer podataka

Broj bajtova potrebnih za smeštanje video podataka je veličine jedne cele slike video signala (829.440 B). Veličina memorije za audio podatke je određena tako da bi se vremenski poklopila sa trajanjem jedne slike video podataka. Ovo poklapanje je zbog sinhronizacije dve različite količine podataka i zbog brzine odabiranja audio i video podataka.

$$f_V(t) = 25 \times 829.440\,t \quad \left[\frac{B}{s}\right] \quad \Rightarrow \quad 829.440 \quad \left[\frac{B}{40\,ms}\right]$$

$$f_A(t) = 192.000\,t \quad \left[\frac{B}{s}\right] \quad \Rightarrow \quad \frac{192.000}{25} = 7.680 \quad \left[\frac{B}{40\,ms}\right]$$

Veličina audio bafera je tako 7.680 B.

Ove vrednosti određuju maksimalne veličine bafera programskih cevi koje se zadaju u DSP/BIOS konfiguraciji. Kako je veličina ISRAM memorije 256KB, baferi za

audio podatke se nalaze u ISRAM memoriji a baferi za video podatke se nalaze u eksternoj memoriji EVM kartice (SDRAM) koji je veličine 32MB.

Sada je EVM kartica spremna za prenos podataka i DSP čeka na zahteve od rukovaoca.

7.4.1 Realizacija mehanizma za transfer podataka

Pozivom funkcije `DeviceIoControl` sa `IOCTL_CAPTURE_VIDEO_START` kodom, u rukovaocu se poziva funkcija `StartVideo` koja javlja DSP programu da pokrene kanal za snimanje video podataka tako što upisuje u `PCI_DMA_Controler` strukturu znak za pokretanje kanala i šalje signal prekida ka DSP-u. Ovaj prekid obrađuje rutina za obradu PCI prekida `HwiPCIIsr`:

```
Void HwiPCIIsr()
{
   Uint32 mask, i;
 1
   if (PCI_intTest(PCI_EVT_MASTEROK))
      {
      PCI_intClear(PCI_EVT_MASTEROK);
      ...
      }

   else if (PCI_intTest(PCI_EVT_PCIMASTER))
      PCI_intClear(PCI_EVT_PCIMASTER);

   else if (PCI_intTest(PCI_EVT_PCITARGET))
      PCI_intClear(PCI_EVT_PCITARGET);

 2
   else if (PCI_intTest(PCI_EVT_HOSTSW))
      {
      PCI_intClear(PCI_EVT_HOSTSW);

      for (i = 0; i < 4; i++)
          {
          mask = 1 << i;
          if (dmaCtrl.dmaChaCtrl & mask)
             {
             dmaCtrl.dmaChaCtrl &= ~mask;
 3
             if (dmaCtrl.dmaChannels[i].chaStatInfo == DMA_CHA_STOPPED ||
              dmaCtrl.dmaChannels[i].chaStatInfo == DMA_CHA_STOP_PENDING)
                {
                dmaCtrl.dmaChannels[i].chaStatInfo = DMA_CHA_RUN_PENDING;
                dmaCtrl.dmaChannels[i].Start();
                }
             else if (dmaCtrl.dmaChannels[i].chaStatInfo ==
                    DMA_CHA_RUNNING)
                {
                dmaCtrl.dmaChannels[i].chaStatInfo = DMA_CHA_STOPPED;
                dmaCtrl.dmaChannels[i].Stop();
                }
             }
          }
      }
}
```

1. Proverava se koji je tip prekida u pitanju.

2. Događaj `PCI_EVT_HOSTSW` je prekid od rukovaoca ka DSP-u. Odmah se briše indikator prekida, da bi se omogućio dolazak novog. Ulazi se u petlju koja ima 4 koraka, za proveru svakog kanala.

3. Kada se dođe do kanala za snimanje video podataka, sledi provera stanja kanala. Ako je kanal zaustavljen, dolazak prekida ga pokreće i poziva funkciju `Start` (`StartVideoCaptDmaChan` (*pci.c*)) za početak prenosa a ako je u ovom momentu već pokrenut, dolazak prekida ga zaustavlja i poziva funkciju za terminiranje prenosa `Stop` (`StopVideoCaptDmaChan` (*pci.c*)).

Funkcija `StartVideoCaptDmaChan` poziva dve funkcije. Pomoću softverskog prekida `SWI_post` poziva *notifyWriter* funkciju `SwiVideoCaptureVPWriterFunc` (*vportcap.c*) u programskoj cevi za snimanje video podataka koji se zove `VideoCapturePipe`. Druga funkcija je `StartVPCapture` (*vportcap.c*) koja resetuje EDMA kanale za snimanje video podataka, postavlja sve potrebne registre i pokreće video priključak 1 za snimanje digitalnih video signala. `SwiVideoCaptureVPWriterFunc` funkcija:

```
Void SwiVideoCaptureVPWriterFunc()
{
   PIP_Handle pipe = &VideoCapturePipe;
   Uint32 bufferY, bufferCb, bufferCr;
   Uint32 bufferY2, bufferCb2, bufferCr2;
   static Uint32 bufferYPrev;
   static Uint32 bufferCbPrev;
   static Uint32 bufferCrPrev;
   static Int index = 0;
   Uns size;

 1
   if (dmaCtrl.dmaChannels[DMA_VIDEO_CAPTURE_CHA].chaStatInfo ==
      DMA_CHA_STOPPED
      || dmaCtrl.dmaChannels[DMA_VIDEO_CAPTURE_CHA].chaStatInfo ==
      DMA_CHA_STOP_PENDING)
        return;
 2
   if (PIP_getWriterNumFrames(pipe) > 0)
      {
      PIP_alloc(pipe);

      bufferY = (Uint32) PIP_getWriterAddr(pipe);
      size = PIP_getWriterSize(pipe);

      bufferCb = bufferY + 4 * size / 2;
      bufferCr = bufferCb + size;
      bufferY2 = bufferY + 4 * VCA_Y_EDMA_ELECNT;
      bufferCb2 = bufferCb + 4 * VCA_Y_EDMA_ELECNT / 2;
      bufferCr2 = bufferCr + 4 * VCA_Y_EDMA_ELECNT / 2;

 3
      EDMA_RSETH(hEdmaVPCaptTblY[index], DST, bufferY);
      EDMA_RSETH(hEdmaVPCaptTblCb[index], DST, bufferCb);
      EDMA_RSETH(hEdmaVPCaptTblCr[index], DST, bufferCr);
```

```
EDMA_RSETH(hEdmaVPCaptTblY[index + 2], DST, bufferY2);
EDMA_RSETH(hEdmaVPCaptTblCb[index + 2], DST, bufferCb2);
EDMA_RSETH(hEdmaVPCaptTblCr[index + 2], DST, bufferCr2);
```

4

```
EDMA_link(hEdmaVPCaptTblY[index], hEdmaVPCaptTblY[index + 2]);
EDMA_link(hEdmaVPCaptTblCb[index], hEdmaVPCaptTblCb[index + 2]);
EDMA_link(hEdmaVPCaptTblCr[index], hEdmaVPCaptTblCr[index + 2]);

EDMA_link(hEdmaVPCaptTblY[index + 2], hEdmaVPCaptLoopY);
EDMA_link(hEdmaVPCaptTblCb[index + 2], hEdmaVPCaptLoopCb);
EDMA_link(hEdmaVPCaptTblCr[index + 2], hEdmaVPCaptLoopCr);

EDMA_disableChannel(hEdmaChaVPCaptureY);
EDMA_disableChannel(hEdmaChaVPCaptureCb);
EDMA_disableChannel(hEdmaChaVPCaptureCr);
```

5

```
if (startingCaptureVPORT)
   {
   if (startingFirstCaptCha)
      {
      EDMA_link(hEdmaChaVPCaptureY, hEdmaVPCaptTblY[index]);
      EDMA_link(hEdmaChaVPCaptureCb, hEdmaVPCaptTblCb[index]);
      EDMA_link(hEdmaChaVPCaptureCr, hEdmaVPCaptTblCr[index]);
      startingFirstCaptCha = FALSE;
      }
   else
      {
      EDMA_link(hEdmaVPCaptPrevY, hEdmaVPCaptTblY[index]);
      EDMA_link(hEdmaVPCaptPrevCb, hEdmaVPCaptTblCb[index]);
      EDMA_link(hEdmaVPCaptPrevCr, hEdmaVPCaptTblCr[index]);
      startingCaptureVPORT = FALSE;
      startingFirstCaptCha = TRUE;
      }
   }
```

6

```
else
   {
   if (EDMA_RGETH(hEdmaChaVPCaptureY, DST) == bufferYPrev)
      EDMA_link(hEdmaVPCaptPrevY, hEdmaVPCaptTblY[index]);
   else
      EDMA_link(hEdmaChaVPCaptureY, hEdmaVPCaptTblY[index]);

   if (EDMA_RGETH(hEdmaChaVPCaptureCb, DST) == bufferCbPrev)
      EDMA_link(hEdmaVPCaptPrevCb, hEdmaVPCaptTblCb[index]);
   else
      EDMA_link(hEdmaChaVPCaptureCb, hEdmaVPCaptTblCb[index]);

   if (EDMA_RGETH(hEdmaChaVPCaptureCr, DST) == bufferCrPrev)
      EDMA_link(hEdmaVPCaptPrevCr, hEdmaVPCaptTblCr[index]);
   else
      EDMA_link(hEdmaChaVPCaptureCr, hEdmaVPCaptTblCr[index]);
   }
```

7

```
bufferYPrev = bufferY;
bufferCbPrev = bufferCb;
bufferCrPrev = bufferCr;
```

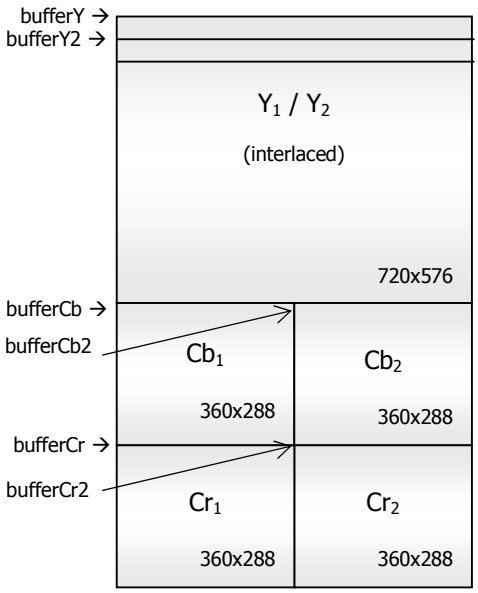
```
      hEdmaVPCaptPrevY = hEdmaVPCaptTblY[index + 2];
      hEdmaVPCaptPrevCb = hEdmaVPCaptTblCb[index + 2];
      hEdmaVPCaptPrevCr = hEdmaVPCaptTblCr[index + 2];

      EDMA_enableChannel(hEdmaChaVPCaptureY);
      EDMA_enableChannel(hEdmaChaVPCaptureCb);
      EDMA_enableChannel(hEdmaChaVPCaptureCr);

      index ^= 1;
      PIP_put(pipe);
      }
```

8

```
  if (dmaCtrl.dmaChannels[DMA_VIDEO_CAPTURE_CHA].chaStatInfo ==
      DMA_CHA_RUN_PENDING)
dmaCtrl.dmaChannels[DMA_VIDEO_CAPTURE_CHA].chaStatInfo = DMA_CHA_RUNNING;
}
```

1. Provera da li je kanal aktiviran ili je zatraženo prekidanje transfera. Ako jeste, kanal se zaustavlja i izlazi se iz funkcije.

2. Provera da li programska cev ima slobodnih bafera za upis podataka. Zauzima se slobodni bafer (`PIP_alloc`) i utvrđuje se veličina bafera koja je jednaka veličini jedne slike. EDMA je nameštena tako da odmah prepliće poluslike koje kopira iz FIFO bafera video priključka i upisuje u bafer programske cevi u SDRAM memoriji. Prva poluslika (*field 1*) se upisuje na adrese `bufferY`, `bufferCb` i `bufferCr` a druga (*field 2*) se upisuje na adrese `bufferY2`, `bufferCb2`, `bufferCr2`. Y, Cb i Cr komponente slike u 4:2:2 formatu se nalaze u memoriji kao što je prikazano na slici 7.20. Ovaj tip skladištenja se naziva planarni (*planar*).



Slika 7.20: Izgled slike u baferu programske cevi

3. Tri EDMA kanala prenose video podatke. Po jedan kanal za svaku komponentu YCbCr formata. Za svaki EDMA kanal postoji tabela sa 4 skupa parametara (2 para) u parametarskoj RAM memoriji. Prvi par se podešava za prenos slike u prvi bafer

programske cevi a drugi par za prenos u drugi bafer programske cevi (*ping-pong buffering*). EDMA se pomoću jednog skupa parametara podešava za prenos jedne poluslike a onda se na osnovu ponovnog punjenja parametara (*reload*) podešava pomoću drugog skupa za prenos druge poluslike u isti bafer. Sledeći prolaz funkcije koristi drugi par parametara za prenos u drugi bafer.

4. Povezivanje (`EDMA_link`) EDMA kanala omogućava automatski niz prenosa podataka. Kanal se povezuje tako da odmah po završetku prenosa prve poluslike počne prenos druge poluslike. Drugo povezivanje je sa prenosa druge poluslike na prazan hod zbog sinhronizacije. Kanali se privremeno blokiraju (`EDMA_disableChannel`) dok ne prođe provera i povezivanje.

5. Provera da li je ovo poziv funkcije na samom početku prenosa podataka. Ako jeste, povezuje se svaki EDMA kanal na prenos komponente prve poluslike u prvi bafer. Druga poluslika je već povezana na prenos prve. U sledećem pozivu funkcije se ulazi u drugi deo ovog uslova, gde se povezuje prenos prve poluslike za sledeću sliku (drugi bafer) na završetak prenosa druge poluslike prethodne slike. Ponovo je druga poluslika (ali sada nove slike) povezana na prenos prve.

6. Svaki sledeći put pozivanja funkcije se ulazi u ovaj deo uslova gde se proverava stanje svakog EDMA kanala i u zavisnosti od toga vrši se povezivanje. Ako kanal u ovom momentu prenosi prvu polusliku prethodne slike, povezuju se parametri prenosa nove slike tek na završetak prenosa druge poluslike. Ako se prenosi druga poluslika, povezuje se EDMA kanal na prenos nove slike.

7. Pokazivači na tekući bafer postaju pokazivači na prethodni zbog sledećeg prolaza i otpuštaju se EDMA kanali (`EDMA_enableChannel`) da produže sa prenosom podataka. Ažurira se indeks i `PIP_put` javlja EDMA kontroleru da može početi prenos podataka i u ovaj bafer.

8. Ako je ovo bio prvi prolaz kroz funkciju, postavlja se stanje kanala na aktivan.

U ovom momentu EDMA prenosi video podatke, prvu polusliku pa odmah drugu polusliku. Po završetku prenosa komponenata svake druge poluslike (prenos kompletne slike), svaki EDMA kanal generiše prekid (Tcc – *transfer complete code*) koji obrađuje rutina `HwiEdmaIsr` (*edmaisr.c*) za obradu EDMA prekida.

```
Void HwiEdmaIsr(...)
{

 1

   if (EDMA_intTest(EdmaVPCaptYTcc))
      {
      EDMA_intClear(EdmaVPCaptYTcc);
      SWI_andn(&SwiVideoCaptureProcessing, 4);
      }

   if (EDMA_intTest(EdmaVPCaptCbTcc))
      EDMA_intClear(EdmaVPCaptCbTcc);

   if (EDMA_intTest(EdmaVPCaptCrTcc))
      EDMA_intClear(EdmaVPCaptCrTcc);
```

```
2
   if (EDMA_intTest(EdmaVPDispYTcc))
       ...
   if (EDMA_intTest(EdmaVPDispCbTcc))
       ...
   if (EDMA_intTest(EdmaVPDispCrTcc))
       ...
   if (EDMA_intTest(EdmaMcASPCaptTcc))
       ...
   if (EDMA_intTest(EdmaMcASPPlayTcc))
       ...
}
```

1. Provera da li je prekid od kanala za snimanje video signala. Obriše se indikator prekida da bi se omogućio dolazak novog. SWI prekidom se javlja funkciji `SwiVideoCaptureProcessingFunc` (*vportcap.c*) da je EDMA prenela celu sliku u `VideoCapturePipe` bafer. Proveravaju se sva tri kanala (Y, Cb i Cr).

2. Analogno se proveravaju svi ostali tipovi transfera.

Ako su zadovoljeni svi uslovi, funkcija `SwiVideoCaptureProcessingFunc` kopira sliku iz `VideoCapturePipe` bafera u `VideoCapturePCIPipe` bafer. Oslobađa prvi bafer za upis nove slike i javlja programskoj cevi za PCI transfer (`VideoCapturePCIPipe`) da je bafer napunjen i poziva njenu *notifyReader* funkciju `SwiVideoCapturePCIReaderFunc` (*pci.c*).

```
Void SwiVideoCapturePCIReaderFunc()
{
   PIP_Handle pipe = &VideoCapturePCIPipe;
   PCI_DMA_Descriptor* dmaDesc;
   PCI_Request* pciReq;
   Uint32 *buffer;
   Uns size;
   Int i;

 1
   if (PIP_getReaderNumFrames(pipe) > 0)
      {
      PIP_get(pipe);

      buffer = PIP_getReaderAddr(pipe);
      size = PIP_getWriterSize(pipe);
 2
      for (i = 0; i < NUM_VIDEO_DESC; i++)
          {
          dmaDesc = GetCurrDmaDesc(DMA_VIDEO_CAPTURE_CHA);
          pciReq = QUE_get(&FreePCIReqQue);
          dmaDesc->dspAddress = (Uint32)buffer + i *
                                    DMA_VIDEO_BYTE_COUNT;
          dmaDesc->cntInfo |= (DMA_VIDEO_BYTE_COUNT << 16);
          pciReq->dmaChannelId = DMA_VIDEO_CAPTURE_CHA;
          pciReq->desc = dmaDesc;

          QUE_put(&PCIReqQue, pciReq);
          SEM_post(&SemPCIRequest);
          }
```

```
3
      PIP_free(pipe);
      }
}
```

1. Provera da li je bafer napunjen podacima. Ako nije, nema šta da se obrađuje. Ako jeste, zauzima se bafer, izvorna adresa i utvrđuje veličina bafera.

2. Ulazi se u petlju koja deli video bafer na 16 blokova (broj deskriptora). Svaki blok je jedan zahtev za PCI transferom koji se ulančava u red čekanja. Uzima se pokazivač na tekući deskriptor kanala za snimanje video signala i slobodni zahtev iz reda (`QUE_get`). Podešava se izvorna adresa PCI transfera i broj bajtova za prenos. Zahtev se označi kao snimanje video podataka i povezuje se sa deskriptorom. Postavlja se u red čekanja (`QUE_put`) i signalizira se semafor (`SEM_post`) da je novi blok spreman za prenos.

3. Na kraju obrade celog bafera, oslobađa se za upis nove slike. Ovo je ujedno i signal `SwiVideoCaptureProcessing` funkciji da je bafer ponovo prazan.

Na signal semafora `SemPCIRequest` se budi `tskProcessPCIRequestFunc` (*pci.c*), ako je završen prethodni prenos bloka preko PCI magistrale.

```
Void tskProcessPCIRequestFunc()
{
   QUE_Handle que = &PCIReqQue;
   PCI_ConfigXfr config;
   Uns xfrMode;
   PCI_DMA_Descriptor* desc;
   static Uns timeout = 0;

1
   while (tskProcessPCIReqRunning)
      {
      SEM_pend(&SemISRdone, timeout);
      SEM_pend(&SemPCIRequest, SYS_FOREVER);

2
      if (!QUE_empty(que))
         {
         timeout = SYS_FOREVER;
         currPCIRequest = QUE_get(que);
         desc = (currPCIRequest->desc);

         switch (currPCIRequest->dmaChannelId)
            {
            case DMA_AUDIO_PLAYBACK_CHA: xfrMode = PCI_READ_PREF;
                                         break;
            case DMA_AUDIO_CAPTURE_CHA:  xfrMode = PCI_WRITE;
                                         break;
            case DMA_VIDEO_DISPLAY_CHA:  xfrMode = PCI_READ_PREF;
                                         break;
            case DMA_VIDEO_CAPTURE_CHA:  xfrMode = PCI_WRITE;
                                         break;
            }
```

```
         config.dspma = (Uns) desc->dspAddress;
         config.pcima = (Uns) desc->busAddress;
         config.pcimc = GET_BYTE_COUNT(desc);

         PCI_xfrConfig(&config);
         PCI_xfrStart(xfrMode);
         }
 3
      else
         timeout = 0;
      }
}
```

1. Na ulazu beskonačne petlje se uvek čekaju dva signala semafora (`SEM_pend`). Završetak prenosa bloka preko PCI magistrale, `SemISRdone` i postavljanje novog zahteva u red čekanja `SemPCIRequest`.

2. Ako red nije prazan, ulazi se u obradu prvog zahteva u redu. Na osnovu indikatora kanala određuje se mod transfera (čitanje ili pisanje). Obrađuje se zahtev (određuje izvorna i odredišna adresa i broj bajtova za prenos). Upisuju se parametri za PCI transfer i sa naredbom `PCI_xfrStart` se pokreće transfer.

3. Ako je red prazan, poništava se čekanje na završetak transfera, jer nikakav nije ni pokrenut.

Kad se završi prenos bloka preko PCI magistrale, generiše se PCI prekid `PCI_EVT_MASTEROK`. Za ovaj prekid se takođe poziva rutina `HwiPCIIsr`, kao za prekid od rukovaoca, ali se ulazi u drugi deo koda.

```
Void HwiPCIIsr()
{
   Uns dmaChannelId;

 1
   if (PCI_intTest(PCI_EVT_MASTEROK))
      {
      PCI_intClear(PCI_EVT_MASTEROK);

      if (currPCIRequest != NULL)
         {
         oldPCIRequest = currPCIRequest;
         currPCIRequest = NULL;

         dmaChannelId = oldPCIRequest->dmaChannelId;
 2
         switch (dmaChannelId)
            {
            case DMA_VIDEO_CAPTURE_CHA:
                  numVideoCaptured++;
                  if (numVideoCaptured == NUM_VIDEO_DESC)
                     {
                     LOG_printf(&LogMain, ...);
                     numVideoCaptured = 0;
                     }
                  break;
```

```
            case DMA_AUDIO_PLAYBACK_CHA:
                 ...
                 break;

            case DMA_AUDIO_CAPTURE_CHA:
                 ...
                 break;

            case DMA_VIDEO_DISPLAY_CHA:
                 ...
                 break;
            }
3
        if (oldPCIRequest->desc->cntInfo & DMA_DESC_MASK_INT_EN)
          {
          dmaCtrl.dmaChaIrqSrc |= (1 << dmaChannelId);
          PCI_dspIntReqSet();
          }

        QUE_put(&FreePCIReqQue, oldPCIRequest);
        SEM_post(&SemISRdone);
        }
      }
4
  else if (PCI_intTest(PCI_EVT_PCIMASTER))
     PCI_intClear(PCI_EVT_PCIMASTER);

  else if (PCI_intTest(PCI_EVT_PCITARGET))
     PCI_intClear(PCI_EVT_PCITARGET);

  else if (PCI_intTest(PCI_EVT_HOSTSW))
     {
     PCI_intClear(PCI_EVT_HOSTSW);
     ...
     }
}
```

1. Provera tipa prekida i da li je tekući deskriptor različit od nule. Ako nije, znači da je već obrađen i završen.

2. Na osnovu oznake kanala se radi određena obrada za svaki transfer. Za snimanje video podataka broje se preneti blokovi da bi se znalo kada je preneta cela slika.

3. Ako je potrebno, šalje se prekid prema rukovaocu (`PCI_dspIntReqSet`) sa oznakom kanala, da je transfer bloka završen. Ovaj prekid obrađuje ISR rutina u rukovaocu (`OnInterrupt`) koja je opisana u poglavlju o rukovaocu.

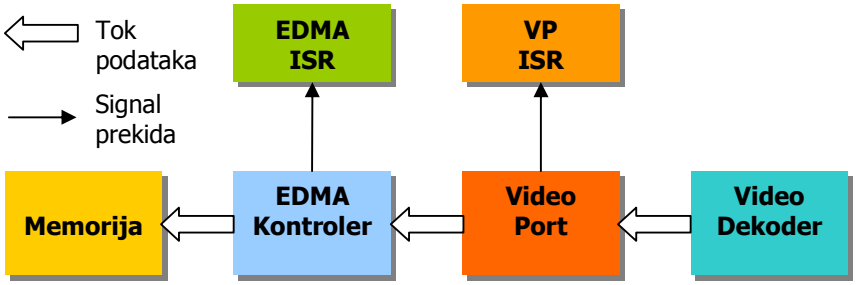
4. Provera ostalih prekida i njihova obrada, koja sada nije od značaja.

Na ovaj način se vrši prenos svih podataka u DSP. Svaki tip transfera ima svoje programske cevi, svoje EDMA kanale, svoje priključke, svoje SWI prekide i njihove funkcije. Sve te obrade se izvršavaju paralelno i međusobno nezavisno. Svi zahtevi se ulančavaju u isti red čekanja koji se obrađuje u zajedničkoj funkciji za sve transfere, `tskProcessPCIRequestFunc`. Zajednička je i ISR rutina `HwiPCIIsr`, koja za svaki preneti blok šalje signal prekida prema rukovaocu.

7.4.2 Moduli DM642 EVM kartice u prenosu podataka

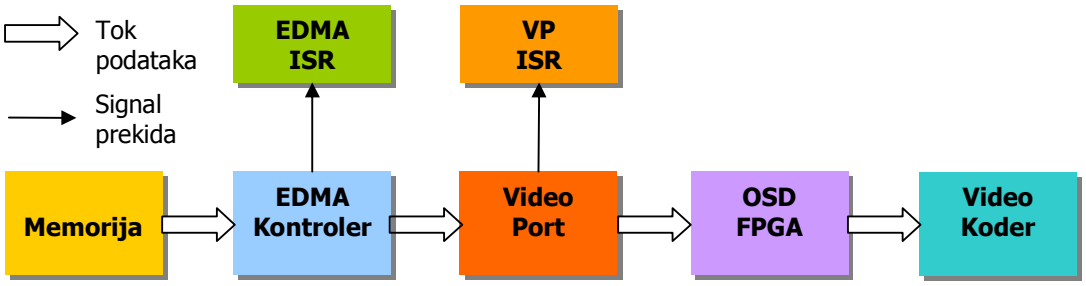
Transferi podataka kojima upravlja programska podrška za DSP se mogu prikazati na sledeći način:

Snimanje video signala (*VideoCapture*) je opisano u primeru. Video dekoder prima analogni video signal spolja, A/D konverzijom pretvara u digitalni koji video priključak 1 prenosi u FIFO bafere za snimanje video podataka. Dalji prenos vrši programska podrška



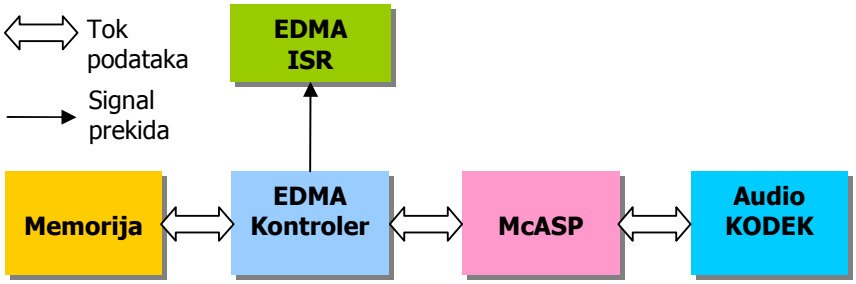
Slika 7.21: Blok dijagram snimanja video signala

Reprodukcija video signala (*VideoDisplay*) je skoro ista snimanju sa obrnutim smerom prenosa podataka. Razlika je u tome što između video priključka 2 i video kodera je FPGA mreža sa OSD mogućnostima. U okviru programske podrške se ova mreža ne koristi pa signali samo prolaze kroz nju do kodera. Video koder pretvara D/A konverzijom digitalne video signale u analogne i šalje napolje



Slika 7.22: Blok dijagram reprodukcije video signala

Snimanje i reprodukcija audio signala (*AudioCapture*, *AudioPlayback*) koriste zajedničke module zbog audio kodeka koji pravi A/D i D/A konverziju audio signala. Umesto video priključka signali idu preko McASP sprege



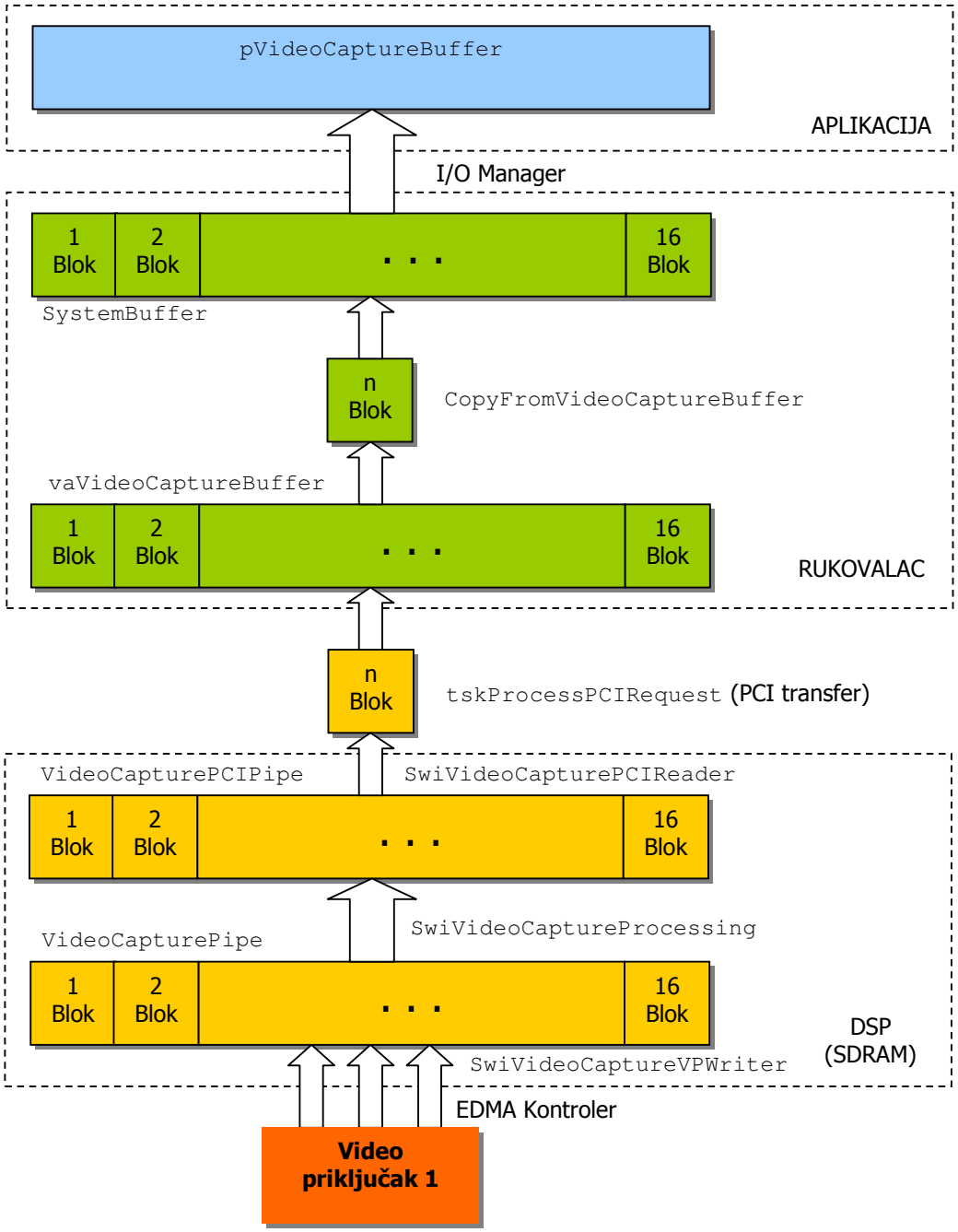
Slika 7.23: Blok dijagram snimanja i reprodukcije audio signala

8. Prenos podataka u sistemu za snimanje i reprodukciju digitalnog audio/video signala

Aplikacija alocira četiri bafera, rukovalac takođe četiri, koji su svi u istoj operativnoj memoriji računara. Programska podrška za DSP takođe alocira bafere ali u internoj (ISRAM) memoriji DSP za audio podatke i eksternoj (SDRAM) memoriji na EVM kartici za video podatke. U realnom vremenu između aplikacije i rukovaoca prenosi se ceo bafer podataka odjednom a između rukovaoca i DSP programa, preko PCI magistrale, prenosi se po jedan blok podataka. Razlog za to je maksimalna veličina pojedinačnog PCI transfera od 64KB a moguće je zbog različitih brzina prenosa. Zbog toga su u rukovaocu formirani kružni PCI DMA baferi. Baferi u PC aplikaciji i u rukovalacu su iste veličine. Baferi u DSP programu su različiti i zbog toga je drugačiji prenos video podataka od prenosa audio podataka.

8.1 Prenos video podataka

Prenos video podataka u celom sistemu možemo prikazati na primeru snimanja video podataka:

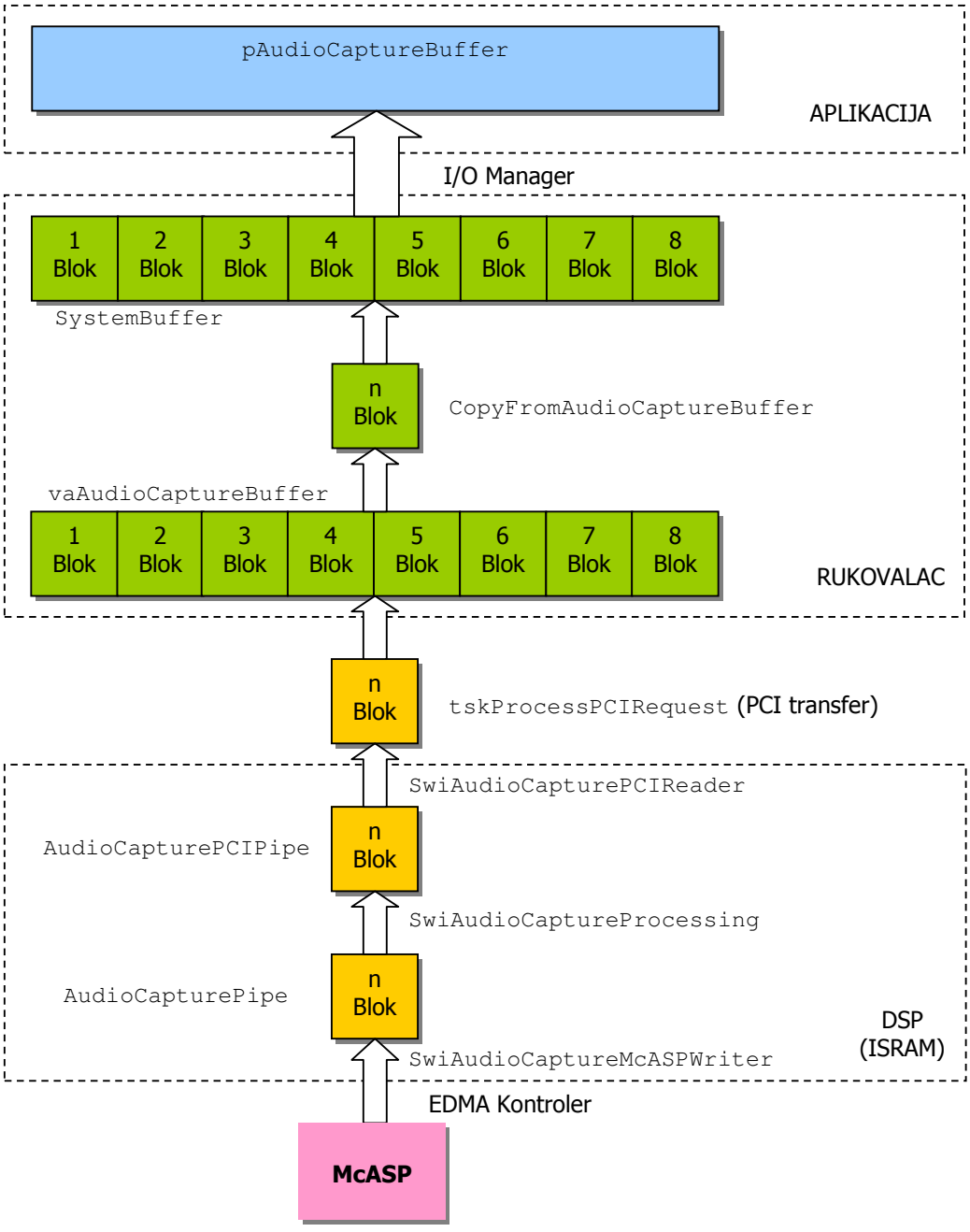


Slika 8.1: Blok dijagram snimanja video podataka

Reprodukcija video podataka je analogna snimanju sa obrnutim smerovima prenosa podataka.

8.2 Prenos audio podataka

Prenos audio podataka u celom sistemu možemo prikazati na primeru snimanja audio podataka:



Slika 8.2: Blok dijagram snimanja audio podataka

Reprodukcija audio podataka je analogna snimanju sa obrnutim smerovima prenosa podataka.

9. Testiranje

Sve tri komponente programske podrške su testirane odvojeno i kao zajednički sistem. Ceo sistem je testiran na sledeći način:

Snimanje digitalnog video signala (*Video Capture*). Za test je spojen analogni video izlaz grafičke kartice računara na analogni video ulaz na EVM kartici. Prenet je trenutni izgled ekrana. U TVP5150 video dekoderu je A/D konverzijom analogni signal pretvoren u digitalni video signal YUV422 formata i prenet sistemom za snimanje do aplikacije. Upisan je u DVS formatu kao tri posebne datoteke (Y, U i V komponente). Dobijena slika je odgovarajućim programom prikazana na ekranu monitora i proverena.

Reprodukcija digitalnog video signala (*Video Display*). Za test je kao ulaz u aplikaciju iskorišćena slika u YUV422 formatu. Preneta je sistemom za reprodukciju do SAA7105 video kodera na EVM kartici. D/A konverzijom je digitalni video signal pretvoren u analogni na izlazu kartice. Na analogni izlaz EVM je priključen TV prijemnik na kojem je prikazana slika iz datoteka DVS formata.

Prenos video signala povratnom petljom (*Loopback Video*). Izgled ekrana računara je prenošen sa izlaza grafičke kartice na analogni ulaz EVM kartice. U TVP5150 dekoderu je A/D konverzijom pretvaran u digitalni video signal YUV422 formata i prenošen sistemom za snimanje u bafer PC aplikacije u operativnoj memoriji. Iz bafera je prenošen natrag sistemom za reprodukciju do SAA7105 video kodera, pretvaran D/A konverzijom u analogni na izlaz EVM kartice. Analogni izlaz EVM kartice je spojen sa TV prijemnikom na kojem je prikazan izgled ekrana računara.

Snimanje digitalnog audio signala (*Audio Capture*). Za test je spojen analogni audio izlaz računara na analogni ulaz na DM642 EVM kartici. Puštena je audio reprodukcija na računaru i signal na izlazu je usmeren na analogni audio ulaz kartice. U AIC23 audio kodeku je A/D konverzijom pretvoren na digitalni audio signal, prenet sistemom za snimanje i na kraju upisan u datoteku WAV formata na masovnoj memoriji. Dobijena datoteka je preslušana pomoću istog programa za reprodukciju zvuka na računaru.

Reprodukcija digitalnog audio signala (*Audio Playback).* Za test je na ulazu u aplikaciju pročitana datoteka u WAV formatu. Preneta je sistemom za reprodukciju do AIC23 audio kodeka na EVM kartici. D/A konverzijom je digitalni audio signal pretvoren u analogni izlaz na kartici. Na analogni izlaz na ploči su priključeni zvučnici preko kojih je preslušana reprodukcija audio podatka iz datoteke.

Prenos audio signala povratnom petljom (*Loopback Audio*). Na računaru je puštena reprodukcija digitalnog audio signala. Analogni izlaz računara je spojen sa analognim ulazom na kartici. U A/D konvertoru AIC23 audio kodeka je pretvoren na digitalni signal i prenošen sistemom za snimanje do bafera Test aplikacije u operativnoj memoriji. Iz bafera je prenošen natrag sistemom za reprodukciju do AIC23 audio kodeka, gde je D/A konverzijom pretvoren na analogni izlaz. Na priključenim zvučnicima se čula reprodukcija audio signala puštena na računaru.

Treba napomenuti da zbog veličine video podataka, snimanje u realnom vremenu na masovnu memoriju računara (*hard disk*) nije moguće zbog toga što video (kako smo ranije videli) podaci stižu brzinom od skoro 20MB u sekundi a upisivanje na spoljnu memoriju konfiguracije ima brzinu najviše 17MB u sekundi (brzina je testirana za konfiguraciju na kojoj je razvijana i testirana programska podrška sistema za snimanje i reprodukciju digitalnog audio i video signala). Dolazi do gubljenja podataka u smislu da operativni sistem ne stiže da upiše sve video slike koje dobije. Analogno je slučaj i sa reprodukcijom video signala sa masovne memorije zbog nedovoljne brzine čitanja. Zbog ovoga je uveden prenos audio i video signala povratnom petljom (*loopback*), gde je izostavljen upis i čitanje sa masovne memorije da bi se proverila ispravnost toka podataka. *Loopback* radi ispravno u realnom vremenu.

10. Zaključak

U ovom projektu realizovana je programska podrška sistema za snimanje i reprodukciju digitalnog audio i video signala zasnovanog na TMS320DM642 DSP. Kao rezultat za video dobijeno je 25 slika u sekundi rezolucije 720 x 576 (snimljen video je u DVS formatu). Rezultat za audio je stereo signal, 16-bitne rezolucije frekvencije 48kHz (stereo audio je u WAV formatu).

Dato rešenje programske podrške sistema je funkcionalno i zadovoljava postavljene zahteve. Omogućava potpunu nezavisnost oba tipa transfera u oba smera:

- snimanje digitalnog video signala
- snimanje digitalnog audio signala
- reprodukcija digitalnog video signala
- reprodukcija digitalnog audio signala

Platforma TMS320DM642 podržava potpuni paralelizam audio i video prenosa. Na drugoj strani, rukovalac ovog tipa nema tu mogućnost. Kako osnovni funkcionalni rukovalac (kakav je implementiran u ovom rešenju) podržava samo jedan tip transfera, rešenje je napravljeno koristeći specifičnosti audio i video podatka. Jedna od specifičnosti je veličina audio i video podataka u jedinici vremena i baferi su različitih veličina. Kad rukovalac dobije zahtev za čitanjem/pisanjem podataka od PC aplikacije, na osnovu veličine zahtevanih podataka može da prepozna da li se radi o audio ili video podacima. Rukovalac koji podržava više tipova transfera je višefunkcionalni rukovalac (*Multifunction Driver*). Međutim implementacija ovakvog tipa rukovaoca izlazi iz okvira ovog rada. Takav rukovalac ima različitu *Plug and Play* implementaciju. Višefunkcionalnost se realizuje tako što glavni (viši) rukovalac sam kreira pod-rukovalac (niži) i selektuje zahteve od aplikacije i po potrebi prosleđuje nižem rukovalacu. U ovom slučaju jedan bi služio za audio prenos a drugi za video prenos podataka.

Takođe su u realizaciji programske podrške ovog sistema formati digitalnog audio i video signala unapred definisani bez mogućnosti menjanja od strane korisnika. Za funkcionisanje ovakvog sistema to nije od značaja a može se bez većih teškoća implementirati.

Ovakvo rešenje je osnova za dalju obradu digitalnog audio i video signala koja se na jednostavan način može dodati postojećem rešenju.

Iako dato rešenje zadovoljava tražene zahteve, uvek ima mesta za dodatne mogućnosti i poboljšanja. Na primer:

- može se staviti korisniku izbor formata digitalnog audio i video signala
- može se staviti korisniku izbor ulaznih i izlaznih datoteka
- može se staviti izbor željene količine prenosa podataka
- ...

11. Dodatak

11.1 I²S sprega[13]

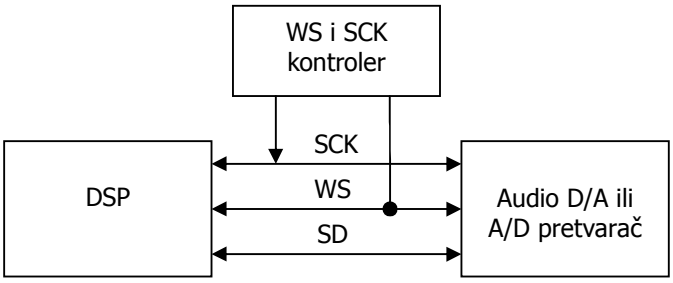
I²S (*Inter-IC Sound or Integrated Interchip Sound)* je električni standard firme *Philips* za serijski sprežni sistem koji povezuje digitalne audio uređaje. Koristi se za prenos 2 kanala (stereo) PCM digitalnih podataka. Primer: prenos informacija između CD transporta i D/A pretvarača u CD čitaču. I²S magistrala ima odvojen signal takta i signale podataka što značajno smanjuje pojavu distorzije u DAC pretvaraču. Magistrala se sastoji od tri linije:

- neprekidni serijski signal takta (SCK – continuous serial clock)
- signal selekcije reči (WS – word select)
- serijski podaci (SD – serial data)

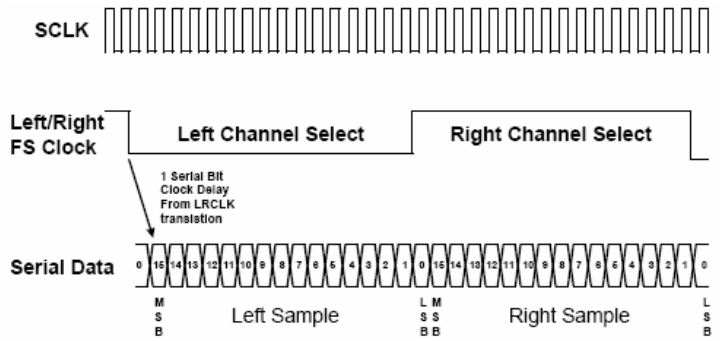
Uređaj koji generiše SCK i WS signal je vodeći. Svaki odabirak se šalje po principu MSB prvi a LSB zadnji. Serijski podaci izlaze iz vodećeg uređaja na opadajuću ivicu SCK takta a odabiraju se u audio DAC pretvaraču na rastuću ivicu SCK takta. Signal WS određuje koji kanal se prenosi:

- WS = 0; levi kanal
- WS = 1; desni kanal

Ova sprega se može uporediti sa sinhronim serijskim portom koji radi u TDM (*time-division multiplexed*) režimu sa 2 aktivna kanala. Na slici 11.1 je blok dijagram I²S sprege



Slika 11.1: Blok dijagram I²S sprege



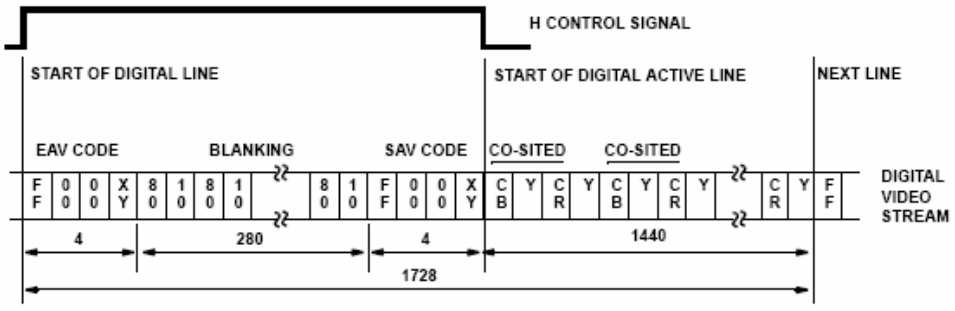
Slika 11.2: I^2S vremenski dijagram za 16-bitne stereo PCM audio podatke

11.2 ITU-R BT.656[14]

ITU-R BT.656 preporuka definiše paralelnu i serijsku spregu za primopredaju digitalnog video signala saglasnog sa 4:2:2 formatom. Rezolucija aktivnog video signala je 720 x 576 piksela za 625/50 video sisteme (PAL) ili 720 x 486 piksela za 525/60 video sisteme (NTSC). BT.656 paralelna sprega koristi 8-bitne ili 10-bitne vremenski prepletene YcbCr podatke i signal takta frekvencije 27 MHz. Umesto konvencionalnih video signala za vremenske reference (HSYNC, VSYNC i BLANK) BT.656 koristi jedinstvene vremenske kodove ugrađene u video sekvencu. Ovo smanjuje broj potrebnih provodnika i priključaka u integrisanim kolima za video BT.656 spregu. Pomoćne digitalne informacije (zvuk, titlovi i teletekst) se mogu prenositi tokom intervala kada nema aktivnog video signala (*blanking intervals).* Ovo eliminiše potrebu za posebnom spregom za zvuk i dodatnim kontrolnim signalima.

11.2.1 YCbCr video sekvenca za 625/50 video sistem

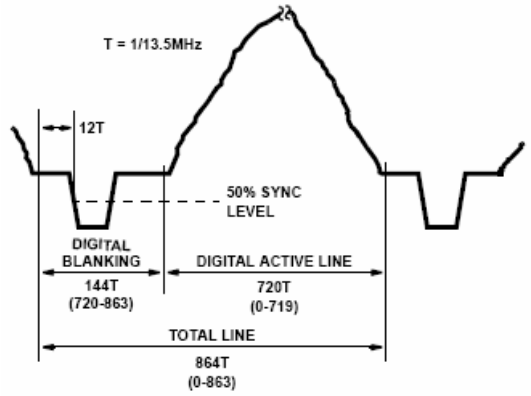
Sekvenca 4:2:2 YcbCr se preuređuje u 8-bitnu ili 10-bitnu sekvencu $Cb_0Y_0Cr_0Y_1Cb_2Y_2Cr_2$… Slika 11.3 ilustruje format za 625/50 video sisteme koji koriste 8-bitne YcbCr podatke.



Slika 11.3: BT.656 8-bitni format podataka paralelne sprege za 625/50 video sisteme

Posle svakog SAV koda, sekvenca aktivnih video podataka uvek počinje sa Cb vrednošću. Svaka linija video signala se odabira sa učestanošću od 13.5 MHz generišući pri tome 720 aktivnih 24-bitnih odabiraka u formatu 4:4:4 YCbCr kao na slici 11.4. Te vrednosti se konvertuju u 16-bitni 4:2:2 YCbCr format što rezultuje sa 720 aktivnih Y

vrednosti po svakoj liniji i 360 aktivnih vrednosti od svake Cb i Cr komponente po liniji. Y podaci i CbCr podaci se prepliću a učestanost takta odabirka se udvostručuje sa 13.5 MHz na 27 MHz.



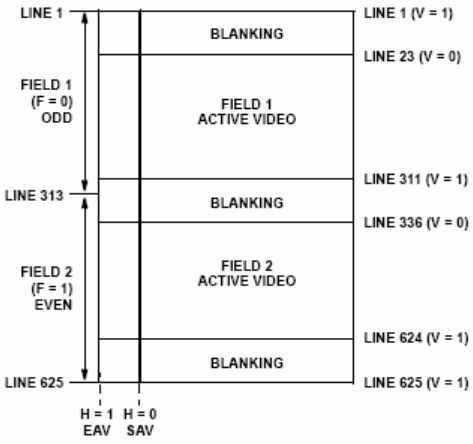
Slika 11.4: BT.656 horizontalne vremenske reference za 625/50 video sisteme

11.2.2 SAV i EAV vremenski kodovi za 625/50

SAV (*start of active video*) i EAV (*end of active video*) kodovi su ugrađeni unutar YCbCr video sekvence. Oni eliminišu potrebu za signalima HSYNC, VSYNC I BLANK koje se obično upotrebljavaju u video sistemima. Status video signala (zajedno sa SAV i EAV sekvencama) se definiše sa bitovima:

- F = 0 za polusliku 1; F = 1 za polusliku 2
- V = 1 tokom povratka mlaza u gornji levi ugao ekrana (*vertical blanking*)
- H = 0 kada je SAV, H = 1 kada je EAV
- P3 – P0 = bitovi za zaštitu
  P3 = V ⊕ H
  P2 = F ⊕ H
  P1 = F ⊕ V
  P0 = F ⊕ V ⊕ H

gde ⊕ predstavlja ekskluzivno ILI funkciju. Bitovi za zaštitu omogućavaju detekciju i korekciju jednobitnih grešaka i detekciju nekih višebitnih grešaka na strani prijemnika.



LINE NUMBER	F	V	H (EAV)	H (SAV)
1-22	0	1	1	0
23-310	0	0	1	0
311-312	0	1	1	0
313-335	1	1	1	0
336-623	1	0	1	0
624-625	1	1	1	0

Slika 11.5: Tipični BT.656 vertikalni intervali za 625/50 video sisteme

12. Literatura

[1] Walter Oney: *Programming the Microsoft Windows Driver Model*, Microsoft Programming Series, Microsoft Press, 1999

[2] *TMS320DM642 Evaluation Module With TVP Video Decoders Technical Reference,* Spectrum Digital, Inc., Stafford, 507345-0001, 2004

[3] *TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor*, Texas Instruments, Inc., SPRS200B, 2002 – 2003

[4] *XDS510PP PLUS Parallel Port JTAG Emulator Installation Guide,* Spectrum Digital, Inc., Stafford, 504955-0001, 2002

[5] *Code Composer Studio v3.0 Getting Started Guide,* Texas Instruments, Inc., SPRU509E, 2004

[6] *TMS320C6000 DSP Peripheral Component Interconnect (PCI) Reference Guide*, Texas Instruments, Inc., SPRU581A, 2003

[7] *TMS320C6000 DSP/BIOS Application Programming Interface (API)*, *Reference Guide,* Texas Instruments, Inc., SPRU403G, 2004

[8] *TMS320C6000 Chip Support Library API Reference Guide,* Texas Instruments, Inc., 2004

[9] *TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide,* Texas Instruments, Inc., SPRU234, 2003

[10] *TMS320C6000 DSP Inter-Integrated Circuit (I2C) Module Reference Guide,* Texas Instruments, SPRU175A, 2002

[11] *TMS320C64x DSP Video Port/VCXO Interpolated Control (VIC) Port Reference Guide*, Texas Instruments, Inc., SPRU629D, 2005

[12] *TMS320C6000 DSP Multichannel Audio Serial Port (McASP) Reference Guide,* Texas Instruments, Inc., SPRU041B, 2003

[13] *I^2S bus specification,* Philips Semiconductors, SN00119, 1986

[14] *BT.656 Video Interface for ICs,* Intersil, AN9728.2, 2002

[15] *A DSP/BIOS PCI Device Driver for TMS320C64xx DSPs*, Texas Instruments, Inc., SPRA845A, 2003

[16] *TLV320AIC23 Stereo Audio CODEC, 8- to 96-kHz, With Integrated Headphone Amplifier,* Texas Instruments, Inc., SLWS106C, 2001

[17] HPA Digital Audio Video: *TVP5150APBS Ultralow Power NTSC/PAL/SECAM Video Decoder With Robust Sync Detector,*
Texas Instruments, Inc., SLES087, 2003
[18] *SAA7104E; SAA7105E Digital video encoder,*
Philips, 2004
[19] *TMS320C6000 Peripherals Reference Guide,*
Texas Instruments, Inc., SPRU190D, 2001
[20] *The TMS320DM642 Video Port Mini-Driver,*
Texas Instruments, Inc., SPRA918A, 2003
[21] *TMS320C64x DSP Two-Level Internal Memory Reference Guide,*
Texas Instruments, Inc., SPRU610, 2002
[22] Kyle Castille: *TMS320C6000 EMIF-to-External SDRAM Interface,*
Texas Instruments, Inc., SPRA433B, 2001
[23] *A DSP/BIOS EDMA McASP Device Driver for TMS320C6x1x DSPs,*
Texas Instruments, Inc., SPRA870A, 2003
[24] *A DSP/BIOS AIC23 Codec Device Driver for the TMS320DM642 EVM,*
Texas Instruments, Inc., SPRA922, 2003
[25] *TMS320DM642 EVM OSD FPGA User's Guide,*
Texas Instruments, Inc., SPRU295, 2003
[26] *Audio Demonstration on the DM642 EVM,*
Texas Instruments, Inc., SPRA930, 2003
[27] *MPEG-2 LOOPBACK FOR DM642 EVM,*
Texas Instruments, Inc., 2003
[28] *Audio Loopback / Echo Example,*
Texas Instruments, Inc.,