MODEL VIRTUELNOG KOMUNIKACIONOG KANALA IZMEĐU TEST APLIKACIJE I SIMULATORA HARDVERA H.264 VIDEO KODEKA

Vladimir Ćirić [1], Vesna Smiljković [2], Milan Pavlović [3], Nebojša Miletić [2] i Ivan Milentijević [1]
[1] Elektronski fakultet, Univerzitet u Nišu, Srbija *{vciric, milentijevic}@elfak.ni.ac.yu*
[2] Accordia Group, Niš, Srbija *{vesna.smiljkovic, nebojsa.miletic}@accordia.co.yu*
[3] Troxo, Niš, Srbija, *milan.pavlovic@troxo.com*

Nagrađeni rad mladog istraživača

Sadržaj – *U ovom radu predložen je model virtuelnog komunikacionog kanala između softvera za generisanje test sekvenci i simulatora hardvera, čime se izbegava potreba za projektovanjem dodatnog sprežnog komunikacionog hardvera, a kompletan problem testiranja sa realnim test sekvencama svodi na problem simulacije na simulatoru. Za implementaciju virtuelnog kanala biće korišćen mehanizam pipe-ova operativnog sistema preko kojih se informacije predaju simulatoru, a takođe i dobijaju rezultati sa simulatora. Primena modela biće ilustrovana na primeru testiranja akceleratora debloking filtra H.264 video kodeka. Model virtuelnog kanala biće prikazan i objašnjen pomoću UML dijagrama sekvenci. Karakteristični parametri implementacije modela će biti prikazani.*

1. UVOD

Posmatrajući razvoj procesa projektovanja elektronskih kola u prošlosti, može se jasno videti da se trend kretao od nižih ka višim nivoima apstrakcije. Na ovaj način omogućeno je da veličina i složenost kola i sistema budu u stalnom porastu, a da obrađivanje detalja na nižim nivoima bude ostavljeno poluautomatskim kompjuterskim alatima za projektovanje. Ovaj trend je posebno vidljiv poslednjih godina zahvaljujući novim tehnologijama proizvodnje koje omogućavaju integraciju celog sistema na jednom čipu (system-on-chip, tj. SOCs). Mogućnost izrade složenog SOCs je prokrčio put novim nivoima integracije, a SOC tehnologija je imala za posledicu mnoge izazove u oblasti projektovanja [1].

Jedna od stvari koje ova tehnologija omogućava je da se projektanti mogu više fokusirati na definiciju ponašanja sistema koje najbolje odgovara potrebama korisnika, pritom analizirajući različite arhitekture. Takodje, ova tehnologija promenila je i način na koji se izvršavaju aktivnosti validacije i testiranja, što dovodi do pomeranja ovih aktivnosti na nivo sistema. Nažalost, trenutno stanje je da alati za projektovanje koji su na nivou sistema još ne podržavaju testiranje i validaciju, mada se neka istraživanja već uveliko time bave [1].

Za testiranje hardvera za obradu digitalnih signala na nivou sistema koriste se razvojna okruženja, a proces testiranja se oslanja na spregu između softvera za generisanje test sekvenci i simulatora hardvera. Ovakvi hibridi softvera i hardvera za testiranje zahtevaju dodatno vreme za projektovanje, a često su i nefleksibilni.

H.264 je standard za kodiranje digitalnih video signala, takođe poznat i kao „MPEG-4 Part 10“, ili MPEG-4 AVC, čija je prva finalna verzija usvojena u maju 2003 [2]. H.264/AVC projekat je imao za cilj kreiranje standarda koji daje zadovoljavajući kvalitet video sadržaja za veoma male brzine protoka (eng. *bit-rate*), čak i dvostruko manje u odnosu na poznati „MPEG-4 Part 2“ standard. Dodatni zahtev koji je postavljen pred projektante je zahtev za jednostavnom primenom standarda na što širi spektar aplikacija, uključujući HDTV, s jedne strane i mobilne uređaje kao drugu krajnost [2],[4].

H.264 koderi i dekoderi, zavisno od primene, mogu se implementirati kao softverski ili hardverski sistemi u celosti, ili kao softverski sistemi kod kojih su određene, vremenski zahtevne funkcije implementirane u vidu hardverskih akceleratora. Za testiranje ovakvih sistema obično se koristi referentni softver sa test video sekvencama, koji je razvijen i priložen uz standard [5]. Akcelerator se obično implementira na nekom sistemu za brzo prototipovanje sa FPGA čipom [3],[4],[6].

Za spregu sistema za prototipovanje sa referentnim softverom neophodno je za hardverski akcelerator projektovati dodatni sprežni hardver, najčešće u vidu DMA kontrolera, što povećava vreme projektovanja, a samim tim i utiče na cenu finalnog proizvoda. Cilj ovog rada je projektovanje modela virtuelnog komunikacionog kanala između referentnog softvera i simulatora hardvera, čime se izbegava potreba za projektovanjem dodatnog sprežnog komunikacionog hardvera, a kompletan problem testiranja sa realnim video sekvencama svodi na problem simulacije na simulatoru [3]. Za implementaciju virtuelnog kanala biće korišćen mehanizam *pipe*-ova operativnog sistema preko kojih se informacije predaju simulatoru. Model virtuelnog kanala biće prikazan i objašnjen pomoću UML dijagrama sekvenci. Primena modela biće ilustrovana na primeru testiranja akceleratora debloking filtra H.264 video kodeka. Karakteristični parametri implementacije modela će biti prikazani.

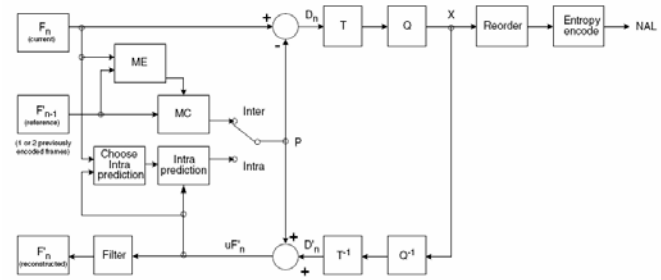
Rad čine pet poglavlja. U 2. poglavlju dat je princip rada H.264 standarda za kodiranje video signala, kao i kratak osvrt na klasično testiranje; u 3. poglavlju detaljno je prikazan model simulatora i način implementacije virtuelnog kanala; u 4. poglavlju je dat primer simulacije akceleratora H.264 debloking filtra, dok je u 5. poglavlju dat zaključak.

2. H.264 MODEL I KLASIČNI PRISTUP TESTIRANJU HARDVERA

U cilju pojednostavljenja objašnjenja uloge i pozicije virtuelnog komunikacionog kanala, u ovom poglavlju dat je

kratak pregled H.264/AVC video kodeka i klasičnog pristupa testiranju hardvera za video kompresiju.

Funkcionalni blok dijagram H.264 kodera prikazan je na slici 1. H.264 koder [2],[4] deli sliku u grupe tačaka kvadratnog oblika, sa 16x16 tačaka u svakoj grupi, tzv makroblok. Svaki makroblok se dalje deli u blokove od 4x4 tačke. Funkcionalni blok, označen sa T, transformiše blokove tačaka u blokove koeficijenata, koji se dalje kodiraju (slika 1). Transformacija T je u većini slučajeva diskretna kosinusna transformacija (DCT), koju prati kvantizacija koeficijenata (blok Q, slika 1) [2],[4].



Slika 1. *Funkcionalni dijagram H.264 enkodera*

Prilikom testiranja hardverski implementiranih blokova kodera prikazanog na slici 1 često se koriste platforme sa FPGA čipovima, povezane sa host računarom preko PCI ili USB interfejsa. Host računar vrši kompresiju test video sekvence izvršavajući funkcije referentnog softvera [5] koji predstavlja dodatak standardu čija je namena testiranje usklađenosti sa standardom.

Klasični pristup testiranju hardvera sa realnim video sekvencama, čiji je model prikazan na slici 2 ogleda se u presretanju poziva funkcije referentnog softvera [5] koja je implementirana hardverski i prosleđivanju zahteva FPGA platformi [4],[6]. Kanal kojim se prosleđuje zahtev od referentnog softvera do prototipa hardvera na FPGA platformi prikazan je na slici 2.

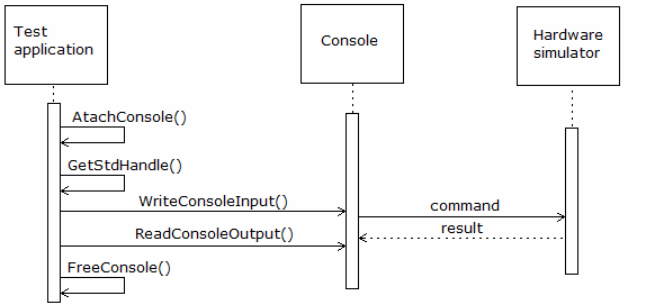
U cilju testiranja funkcije H.264 video kodeka koja je implementirana hardverski, funkcija iz referentnog softvera (slika 2, [5]) se modifikuje tako da se ulazni parametri funkcije prosleđuju hardveru, a rezultat obrade dobijen od hardvera vraća funkciji. Ovim se postiže testiranje hardverskih akceleratora u realnom okruženju. Sprežne komponente između softvera koji generiše realne test sekvence i hardvera prototipovanog na FPGA platformi [3] prikazane su na slici 2.

Sistem za testiranje sa slike 2 zahteva dodatno vreme za projektovanje sprežnog kanala između softvera i akceleratora, a samim tim i povećava cenu testiranja hardvera.

3. IMPLEMENTACIJA VIRTUELNOG KOMUNIKACIONOG KANALA

Za testiranje hardvera dostupna su mnoga okruženja [3]. Xilinx ISE, kao ciljno okruženje u ovom radu, za simulaciju hardvera koristi ISim simulator [7]. ISim predstavlja *frontend* VHDL opisa simulatora u vidu konzolne aplikacije, koja omogućava postavljanje vrednosti ulaznih signala, pokretanje simulacije na određeno vreme i tekstualni prikaz rezultata simulacije [7]. Xilinx ISE koristi ISim simulator na način transparentan korisniku, tako što test vektore koje je korisnik zadao, prosleđuje ISim-u, a rezultate simulacije prikazuje u vidu talasnih oblika.

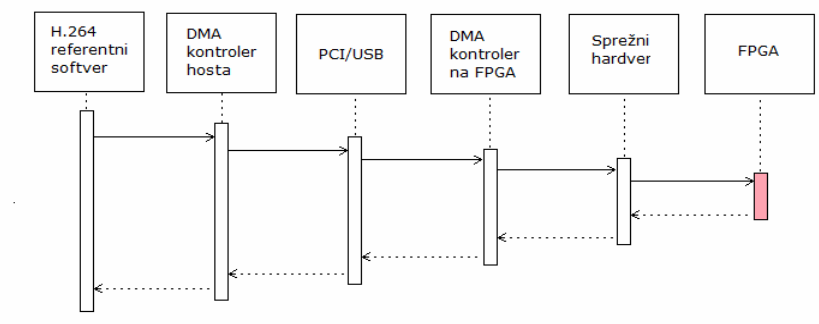
„Zadavanje“ test vektora od strane korisnika je vremenski zahtevan posao, a u slučaju testiranja video kodera može se reći i neizvodljiv. U ovom radu predloženo je rešenje kojim se test vektori, za razliku od načina prikazanog na slici 2, prosleđuju simulatoru hardvera (slika 3). Za implementaciju virtuelnog kanala koristi se mehanizam *pipe*-ova operativnog sistema preko kojih se informacije predaju simulatoru, a takođe i dobijaju rezultati sa simulatora.



Slika 3. Model virtuelog komunikacionog modela

Komunikacija između test aplikacije i simulatora hardvera, u cilju prosleđivanja realnih test vektora, odvija se preko Windows API funkcija [8]: *AttachConsole*, *FreeConsole*, *GetStdHandle*, *ReadConsoleOutput* i *WriteConsoleInput*, kao što je to prikazano modelom sa slike 3.

AttachConsole pridružuje konzolni objekat pokrenutom procesu, i on služi kao ulazno-izlazni kanal za komunikaciju sa procesom ISim simulatora, zadavanje parametara i čitanje rezultata. *FreeConsole* oslobađa konzolni objekat od procesa. Sve sekvence naredbi kojima se komunicira sa konzolom



Slika 2. Model klasičnog pristupa

oivičene su parovima *AttachConsole* i *FreeConsole*. Na taj način je omogućeno nezavisno simultano funkcionisanje više simulatora, bez konflikata sa konzolnim objektima ISim simulatora. *GetStdHandle* vraća handle prema standardnom ulazu, izlazu, ili greški. *ReadConsoleOutput* čita trenutno stanje sa konzole simulacije u bafer. Taj bafer se koristi za parsiranje izlaza i smeštanje rezultata simulacije u strukture koje odgovaraju izlaznim signalima. *WriteConsoleInput* upisuje niz događaja na standardni ulaz konzole i time joj diktira komandu za izvršenje. Korišćeni događaji su događaji prouzrokovani od strane tastature (*keystroke*-ovi, [8]) i to u parovima *keyUp* i *keyDown*.

Predloženim rešenjem (slika 3) izbegnuta je potreba za projektovanjem dodatnog sprežnog komunikacionog hardvera, a kompletan problem testiranja sa realnim video sekvencama sveden je na problem simulacije na simulatoru. Cena smanjenja vremena projektovanja dodatnog sprežnog hardvera plaćena je produžetkom vremena simulacije. Problem predloženog rešenja koji dovodi do degradacije performansi je nepostojanje mogućnosti da konzola signalizira dostupnost rezultata (*result* poruka, slika 3), tako da je očitavanje potrebno izvoditi periodično.

4. PRIMER TESTIRANJA H.264 DEBLOKING FILTRA

Predloženo rešenje implementirano je za testiranje akceleratora debloking filtra (slika 1) H.264 sistema za kodiranje video zapisa. Model u koji je ugrađen virtuelni kanal sa slike 3 prikazan je na slici 4.

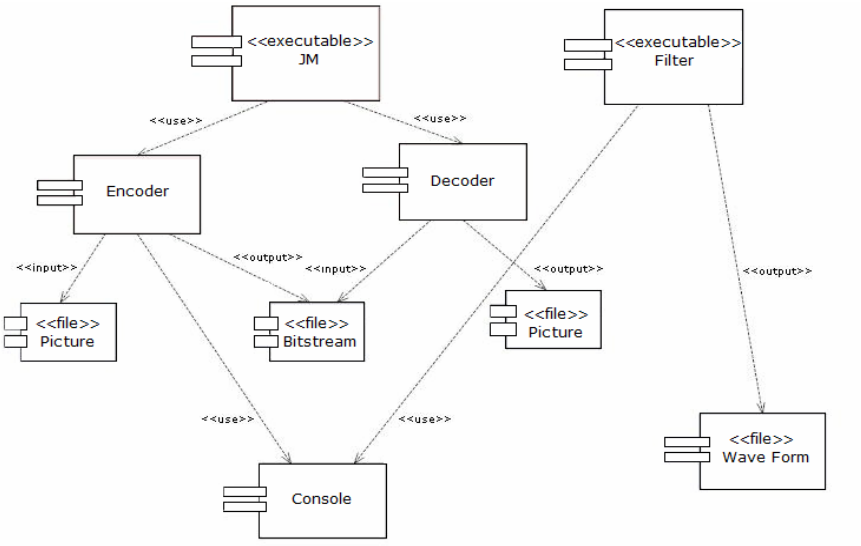
Simulacioni model prikazan na slici 4 čine dve celine. Jednu celinu predstavlja ISim simulator koji simulira hardverski akcelerator debloking filtra, kome se pristupa preko konzole ISim simulatora (slika 3), a drugu celinu čini referentni softver (JM, [5]). *Console*, *Filter* i *WaveForm* (slika 4) čine celinu koja predstavlja simulator, dok deo vezan za JM čine *JM*, *Encoder*, *Decoder*, *Picture* i *Bitstream* (slika 4). Na slici 4 prikazana je komunikacija između opisanih celina.

JM komponenta (slika 4) je dostupna uz standard i služi za testiranje novih rešenja, tj. radi proveru da li je novo rešenje u skladu sa standardom [5]. Ovaj standard ne definiše eksplicitno par koder-dekoder, već samo sintaksu video *bitstream*-a i metode za dekodiranje ovog *bitstream*-a (slika 4, [5]).

JM komponenta modela sa slike 4 sastoji se iz dva odvojena projekta [5]: koder i dekoder. Filtriranje je proces koji se izvršava i u koderu i u dekoderu, ali za testiranje je dovoljno koristiti akcelerator filtra u koderu. Uloga akceleratora je da poveća efikasnost video kompresije [2].

Kodiranje video zapisa se vrši tako što se obradjuju *frame*-ovi, tj. pojedinačne slike. Nakon završetka celog procesa kodiranja, dobija se *bitstream* koji se dalje može prenositi ili čuvati, a da bi se izvršila rekonstrukcija originalnog video zapisa, potrebno je koristiti dekoder, kome se na ulaz dovodi *bitstream*, a kao izlaz dobija rekonstruisani *frame* video zapisa (slika 4, [2], [5]). Filtar (slika 1 i slika 4) je opisan u VHDL-u, sintetizovan i implementiran na Xilinx VirtexE 2000 bg560 čipu, a izabrana arhitektura filtra je pogodna za implementaciju u mobilnim uređajima [4]. Pristup eksternom filtru iz JM komponente izvršen je preko konzole, na način predložen na slici 3. Konzolu dele roditeljski proces, tj. proces JM [5] i dete proces [8], tj. proces ISim simulatora. U zajedničku konzolu modifikovana funkcija JM-a vrši postavljanje signala na ulazne portove filtra (slika 3, 4), upis komandi za izvršavanje filtra i upis komandi za dobijanje vrednosti signala na izlaznim portovima filtra.

Za implementaciju komunikacionog kanala (slika 3) JM je modifikovan u skladu sa slikom 4, dodavanjem određenog broja funkcija komponenti JM (*FilterInit*, *InitializeConsole*, *CreateChildProces*, *FilterConfig*, *FilterRun*, *FilterExit*). Implementirane funkcije pozivaju sistemske funkcije za rad sa konzolom, zatim funkcije za inicijalizaciju filtra, kao i funkcije za obradu i prosledjivanje ulaznih parametara u filtar



Slika 4. *Dijagram simulacionog modela* deblocking *filtra*

i za čitanje i obradu rezultata dobijenih iz filtra.

FilterInit() je funkcija za inicijalizaciju filtra. Ona se poziva samo jednom i to na početku glavne funkcije JM komponente. U njoj se:

- poziva funkcija za inicijalizaciju konzole *InitializeConsole()* u kojoj se postavljaju hendleri za pristup konzoli, podešava veličina prozora konzole,
- poziva funkcija za kreiranje child procesa *CreateChildProces()* gde se kreira proces ISim simulatora *Filter.exe,*
- inicijalizuje filtar (dovode RESET i ENABLE signali).

Svaki put kada je potrebno izvršiti filtriranje nad granicom izmedju blokova [2], odnosno proslediti realne test vektore ISim simulatoru, potrebno je pozvati funkcije *FilterConfig(int mode)* i *FilterRun(int inputs[])*.

FilterConfig(int mode) je funkcija koja služi za konfigurisanje filtra. Kao ulazni parametar prosledjuje se mod u kome će filtar raditi, tj. jačina granice [2].

FilterRun(int inputs[]) je funkcija kojoj se kao ulazni parametar unosi niz piksela koje treba filtrirati. Vrednosti ovih piksela prosleđuju se virtuelnim komunikacionim kanalom na ulaz za podatke komponente filtra (slika 4), a zatim se istim kanalom iz komponente filtra čitaju rezultati filtriranja, odnosno vrednosti filtriranih piksela (slike 3, 4).

Kada se celokupan postupak kodiranja izvrši, na samom kraju glavnog programa JM komponente, poziva se funkcija *FilterExit()* kojom se uništava proces dete (ISim simulator).

Predloženim rešenjem problem testiranja sa realnim video sekvencama sveden je na problem simulacije na simulatoru. Što se performansi simulacije tiče, filtriranje 1 frejma u rezoluciji 28x20 kod klasičnog pristupa (slika 2) vrši se u realnom vremenu. Kod manuelnog kodiranja test vektora u Xilinx okruženju za ovaj obim problema potrebno je pripremiti i kodirati 500 test vektora, što može trajati veoma dugo i podložno je greškama. Simulacija istog obima na predloženom modelu traje približno 10 min, što je u našem slučaju prihvatljivo, imajući u vidu vreme potrebno za projektovanje DMA kontrolera sa slike 2 pri svakoj značajnijoj modifikaciji testiranog hardvera. Konfiguracija na kojoj se softver izvršava nije od presudnog značaja za vreme simulacije jer je kašnjenje određeno samim modelom (*result* poruka, poglavlje 3, Slika 3).

5. ZAKLJUČAK

U ovom radu predložen je model virtuelnog komunikacionog kanala između softvera za generisanje test sekvenci i simulatora hardvera, čime je izbegnuta potreba za projektovanjem dodatnog sprežnog komunikacionog hardvera, a kompletan problem testiranja FPGA prototipa uvođenjem realnih test sekvenci sveden je na problem simulacije na simulatoru. Za implementaciju virtuelnog kanala korišćen je mehanizam *pipe*-ova operativnog sistema preko kojih se informacije predaju simulatoru, a takođe i dobijaju rezultati sa simulatora. Primena modela ilustrovana je na primeru testiranja akceleratora debloking filtra H.264 sistema za kodiranje video signala. Model virtuelnog kanala prikazan je i objašnjen pomoću UML dijagrama sekvenci. Za razliku od klasičnog testiranja, za implementaciju virtuelnog kanala nije potreban dodatni hardver. Cena smanjenja vremena projektovanja dodatnog sprežnog hardvera plaćena je produžetkom vremena simulacije.

LITERATURA

[1] Matteo Sonza Reorda, Zebo Peng, Massimo Violante, „*System-level Test and Validation of Hardware/software Systems*“, Springer, 2005.

[2] I. Richardson, “H.264 and MPEG-4 Video Compression - Video Coding for Next Generation Multimedia”, *John Wiley & Sons, In*., New York, 2003.

[3] R. C. Cofer, Benjamin F. Harding, „*Rapid System Prototyping with FPGAs*“, Elsevier, 2005.

[4] V. Ćirić, I. Milentijević, "Area-Time Tradeoffs in H.264/AVC Deblocking Filter Design for Mobile Devices", *Proceedings of IEEE Conference on Signal Processing and its Applications*, Sharjah, United Arab Emirates, February 2007, 1-4244-0779-6/07/ IEEE.

[5] JVT software JM10.2, January 2006.

[6] V. Ćirić, J. Kolokotronis, I. Milentijević: “Partial Error-Tolerance for Bit-Plane FIR Filter Architecture”, *International Journal of Electronics and Communication*, Esevier Science, accepted 2008.

[7] Nicholas M. Karayanakis, "*Advanced System Modelling and Simulation with Block Diagram Languages*", CRC Press, 1995.

[8] David Harms, David Gerhard, "*Clarion Tips and Techniques*", CoveComm Inc., 2003.

Abstract – In this paper we propose a model of virtual communication channel between a test application and a hardware simulator. The model is proposed in order to avoid design of additional communication hardware between test application and FPGA hardware prototype, and moves a complete test problem into domain of simulation. For implementation of the virtual channel we use a pipes provided by an operating system as an API. Pipes are used to pass test vectors to hardware simulator and vice versa. Design example that illustrates functionality of proposed model will be given using H.264 advanced video coding system. The model will be given as UML sequence diagram, and will be described in detail. Implementation results related to H.264 advanced video coding system will be given.

VIRTUAL COMMUNICATION CHANNEL MODEL FOR TEST APPLICATION AND H.264 VIDEO CODEC HARDWARE SIMULATOR

Vladimir Ćirić, Vesna Smiljković, Milan Pavlović, Nebojša Miletić and Ivan Milentijević