Izrada PC aplikacije za emulaciju komunikacionog podsistema integrisanog kola

Milan Z. Bjelica, *Fakultet tehničkih nauka, Novi Sad*
Milan Savić, Velimir Vujanović, Miodrag Temerinac, *MicronasNIT, Novi Sad*

***Sadržaj* — Cilj rada je da prikaže postupak izrade i konačno rešenje PC aplikacije za emulaciju komunikacionog podsistema integrisanog kola. Opisani su razlozi za kreiranje takve aplikacije, kao i primeri u kojima primena ovakve aplikacije može biti korisna. U okviru opisa razvoja aplikacije, biće predstavljeni osnovni koncepti izrade interpretera i primene jezika za emulaciju komunikacionih scenarija.**

Ključne reči — emulator, interpreter, TV, komunikacija, testiranje.

I. UVOD

UBRZANJE razvoja digitalnih sistema u oblasti potrošačke elektronike jedan je od prioriteta u industriji ovih sistema. Dekomponovanje problema i pristup projektovanju odozdo prema gore (*bottom-up*) dovode do paralelizacije razvoja. Nesumnjivo najveći problem predstavlja paralelno projektovanje fizičke arhitekture i razvoj programske podrške. Ostvarenje cilja, da se programska podrška što brže završi po pojavi prvih primeraka razvojnih ploča odredišne fizičke arhitekture, nameće zahtev da se ona razvija pre nego što razvojna platforma bude na raspolaganju.

Emulacija razvojne platforme predstavlja jedan od preduslova za rani razvoj programske podrške. Postoji nekoliko različitih mogućnosti za emulaciju.

Prva mogućnost je da se delovi programske podrške razvijaju na razvojnim platformama koje su dostupne, ukoliko poseduju sličnu arhitekturu. U toku razvoja na ovim platformama, izostavlja se izrada onih delova za koje se zna da strogo zavise od fizičke arhitekture. Ovde se prevashodno misli na razlike sprežnih podsistema.

Druga mogućnost je da se ciljna platforma u potpunosti emulira korišćenjem *FPGA (Field-Programmable Gate Array)*, pod uslovom da je projektovanje fizičke arhitekture završeno i da je odgovarajuća konfiguraciona datoteka dostupna. Na ovaj način ostvaruje se maksimalno ubrzanje razvoja programske podrške, ako se u obzir uzme vreme potrebno da se realna platforma fabrički izradi i verifikuje. Međutim, uređaji opšte namene koji omogućavaju ovakvu emulaciju (npr. *ChipIt* kompanije *Pro Design Electronics*) nisu jeftini.

Ukoliko je u pitanju emulacija integrisanog kola koje je potrebno povezati sa postojećom fizičkom arhitekturom korišćenjem neke od standardnih sprega, emulacija komunikacionog podsistema tog integrisanog kola može se obezbediti i za *PC* sa operativnim sistemom *Windows* ili *Linux*. Na ovaj način, postojećoj fizičkoj arhitekturi je omogućeno da se ponaša kao da je neophodno integrisano kolo već izrađeno i prisutno u sistemu.

Emulacija dela čipa razvojem *PC* aplikacije za tu namenu, osim što predupređuje postojanje fizičke razvojne platforme, odnosno integrisanog kola, takođe omogućuje veoma brzo ispitivanje platforme sa kojom se povezuje. Npr. definisanjem različitih scenarija komunikacije, moguće je ispitati ponašanje fizičke razvojne platforme u zavisnosti od odgovora na zahtev, i sl. Dalje, emulirana platforma može biti i fizički dostupna, pa da i tada ima smisla koristiti *PC* emulaciju. Neki slučajevi u kojima je *PC* emulacija dobro rešenje su: (1) postojanje nedovoljnog broja razvojnih ploča, (2) složenost podešavanja kompletnog razvojnog okruženja, (3) potreba za poznavanjem samo onih detalja koji su neophodni za razvoj, a za koje je *PC* emulacija sasvim dovoljna i (4) ispitivanje graničnih slučajeva koji se teško rekonstruišu u realnoj primeni.

II. MOTIVI ZA RAZVOJ

U oblasti potrošačke elektronike za digitalnu televiziju, proširenja fizičke arhitekture dešavaju se često. Pri pojavi novih video i audio formata kompresije (u poslednje vreme *H.264* i *AC3*), neophodno je obezbediti dodatna integrisana kola koja će omogućiti dekodovanje i prikaz slike i zvuka. Arhitektura se često proširuje i različitim integrisanim kolima za obradu video ili audio signala, kako bi se poboljšao kvalitet (npr. kompenzacija pokreta, interpolacija, izoštravanje i sl). Osnovni upravljački procesor, koji predstavlja srce fizičke arhitekture sistema koji se proširuje, komunicira sa dodatim blokovima, izdajući komande i ispitujući trenutno stanje.

Ukoliko integrisano kolo nije fizički dostupno, osnovni procesor može da komunicira sa *PC* emulatorom. *PC* aplikacija emulira ponašanje komunikacionog podsistema *TV* čipa, a rezultate operacija koje obavlja i izmenu stanja

Ovaj rad je delimično finansiran od Ministarstva za nauku Republike Srbije, projekat 11005, od 2008. god.

Milan Z. Bjelica, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija; (telefon: 381-21-4801112; faks: 381-21-450721; e-mail: milan.bjelica@rt-rk.com).

Milan Savić, MicronasNIT, Institut za informacione tehnologije, Fruškogorska 11, 21000 Novi Sad, Srbija; (e-mail: milan.savic@micronas.com).

Velimir Vujanović, MicronasNIT, Institut za informacione tehnologije, Fruškogorska 11, 21000 Novi Sad, Srbija; (e-mail: velimir.vujanovic@micronas.com).

Miodrag Temerinac, MicronasNIT, Institut za informacione tehnologije, Fruškogorska 11, 21000 Novi Sad, Srbija; (e-mail: miodrag.temerinac@micronas.com).

vrši u zavisnosti od zadatog scenarija. Jednostavnim učitavanjem i izvršavanjem scenarija na *PC* strani, moguće je detaljno ispitati ponašanje fizičke arhitekture osnovne platforme.

U nastavku rada, predstavljen je koncept rešenja i neki detalji izrade *PC* aplikacije za emulaciju komunikacionog podsistema *TV* bloka za dekodovanje i prikaz video i audio formata kompresije *H.264* i *AC3*. Povezivanjem sa *PC* emulatorom, umesto sa realnim integrisanim kolom, omogućen je razvoj većeg dela programske podrške za ciljnu platformu, sa operativnim sistemom *Linux*. Takođe, s obzirom da je reč o komunikacionom podsistemu, uloge se mogu zameniti tako da *PC* emulira osnovni sistem.

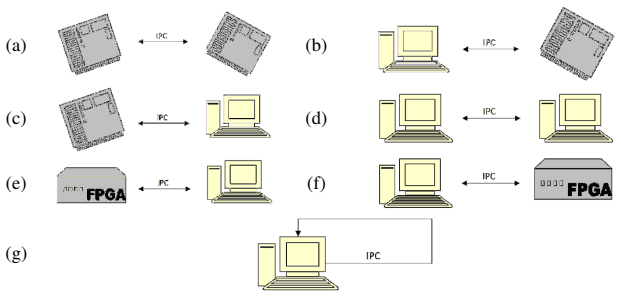
III. KONCEPT REŠENJA

Platforma koja se proširuje (osnovna platforma) i integrisano kolo za proširenje (koprocesor) povezuju se serijskom *UART (Universal Asynchronous Receiver/Transmitter)* spregom. Ukoliko se *Rx* i *Tx* linije izvedu sa pinova osnovne platforme i povežu sa serijskim adapterom, moguće je izvršiti povezivanje osnovne platforme sa *PC*-jem, umesto sa koprocesorom.

Osnovna platforma i koprocesor komuniciraju posredstvom protokola za pouzdanu komunikaciju na transportnom nivou (*InterProcessor Communication Protocol - IPC*). *IPC* omogućava povezivanje rukovanjem, gde su obe strane u komunikaciji jednake (*peer-to-peer*). On omogućava otkrivanje greške u prenosu, tako što uvodi mehanizam potvrđivanja poslatih datagrama i kontrolnu sumu za proveru integriteta podataka. Protokol ne uvodi mehanizme retransmisije u slučaju greške u prenosu, već samo obaveštava korisnika da prenos nije uspeo, pa se retransmisija mora obavljati na višem nivou.

Konkretni formati upravljačko-kontrolnih poruka, zajedno sa mehanizmima za isporuku rezultata zahtevane operacije, izrađeni su u okviru protokola sesionog nivoa.

Da bi se emulacija uspešno obavila, minimalan skup zahteva koje *PC* aplikacija mora da obezbedi su izrada potrebnih komunikacionih protokola. Oslanjajući se na serijsku spregu, ova aplikacija omogućava emulaciju bilo osnovne platforme, bilo koprocesora. Teoretski, mogućnosti primene aplikacije *PC* emulatora su višestruke, što je prikazano na Slici 1.



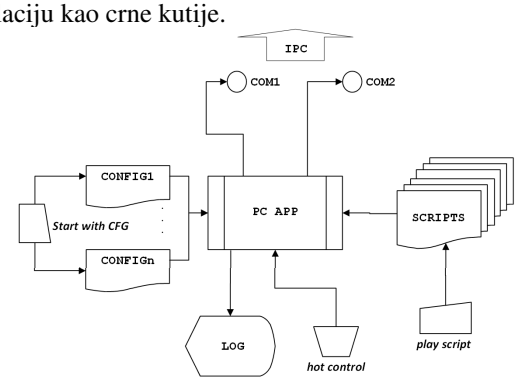
Slika 1 – *Načini primene PC aplikacije za emulaciju* – (a) realno okruženje (osnovna platforma je povezana sa koprocesorom) – (b) Osnovnu platformu emulira *PC* – (c) Koprocesor emulira *PC* – (d) I osnovnu platforma i koprocesor emulira po jedan *PC* – (e) Osnovnu platformu emulira *FPGA,* a koprocesor *PC* – (f) Osnovnu platformu emulira *PC*, a koprocesor *FPGA* – (g) Osnovnu platformu i koprocesor emulira isti *PC*, sa po jednom instancom aplikacije.

PC emulator aplikacija obezbeđuje sledeće:

- Mogućnost povezivanja serijskom spregom na proizvoljnom komunikacionom prolazu, sa proizvoljnom konfiguracijom (*HW flow control, baudrate, stop bit* itd);
- Mogućnost konfigurisanja parametara serijske veze u *XML (Extensible Markup Language)* datoteci, kao i učitavanja različite *XML* konfiguracione datoteke u zavisnosti od parametra komandne linije;
- Mogućnost pokretanja i zaustavljanja *IPC* veze;
- Mogućnost interpretiranja *XML-like* skriptova koji opisuju ponašanje komunikacionog podsistema, uz definisanje pravila jezika za skriptovanje;
- Mogućnost izvršavanja *XML* skriptova po pozivu, unošenjem naziva skripta, kao i njihovo prekidanje u bilo kom trenutku;
- Mogućnost uticanja na tok izvršavanja jednog skripta „u letu“, objavljivanjem posebnog režima rada.
- Mogućnost pokretanja više instanci *PC* aplikacije na različitim komunikacionim prolazima.

PC aplikacija izrađena je u okruženju *Microsoft Visual Studio 2005*, kao *MFC* aplikacija, s obzirom na jednostavnost *GUI* i malu veličinu izvršne datoteke.

Na slici 2 je prikazan koncept *PC* aplikacije za emulaciju kao crne kutije.



Slika 2 – *Koncept PC aplikacije za emulaciju*

IV. PROGRAMSKO REŠENJE

Upravljanje serijskom spregom obezbeđeno je klasom *CSerial*. Ova klasa omogućuje jednostavno otvaranje komunikacionog *COM* prolaza, podešavanje parametara veze, kao i čitanje sa prolaza i pisanje na njega. Klasa omogućava podešavanje ponašanja funkcije čitanja, u smislu blokirajućeg i neblokirajućeg pristupa podacima, odnosno uz upotrebu vremenske kontrole. Klasa takođe obezbeđuje kompletan mehanizam upravljanja greškama.

Podešavanje serijske sprege se obavlja automatski, učitavanjem odgovarajuće konfiguracije iz *XML* datoteke ***config.xml***. Unutar ove datoteke mogu se naći sva podešavanja koja su potrebna za korišćenje aplikacije. Pri pokretanju aplikacije može se navesti neka druga konfiguraciona datoteka umesto podrazumevane, tako što će se aplikacija pokrenuti sa imenom konfiguracione datoteke kao jedinim parametrom:

mdxh_emul drugi_config.xml

Na ovaj način moguće je napraviti više različitih *Windows* prečica, čijim će se aktiviranjem *PC* aplikacija

za emulaciju pokrenuti sa različitom konfiguracijom, što izuzetno olakšava rad.

Izgled konfiguracione datoteke prikazan je u okviru:

```
<?xml version="1.0" standalone="no"?>
<SERIAL>
   <PORT>COM1</PORT>
   <BAUDRATE>115200</BAUDRATE>
   <DATAB>8</DATAB>
   <STOPB></STOPB>
   <PARITY>EVEN</PARITY>
   <HSHAKE>HW</HSHAKE>
</SERIAL>
```

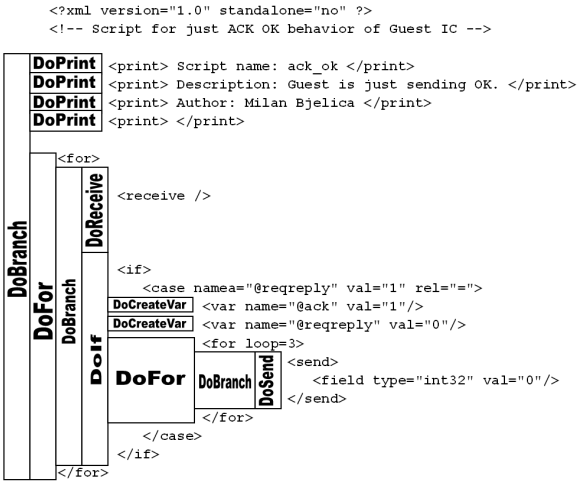
Unutar projekta integrisan je *XML* sintaksni analizator otvorenog koda (*TinyXML*) kao kompaktno rešenje u obliku klasa za jednostavnu sintaksnu analizu osnovnih *XML* datoteka. Korišćenjem *XML* analizatora *PC* aplikacija pristupa konfiguracionim opcijama konfiguracione datoteke. Analiza se obavlja po samom učitavanju datoteke, kada analizator formira *XML* stablo, kome se pristupa korišćenjem odgovarajućih metoda za dobavljanje sledećeg čvora istog nivoa, odnosno čvora potomka (*child*) ili matičnog (*parent*) čvora (nivoi ispod i iznad). Postojanje analizatora nametnulo je rešenje za skripting jezik, koji je *XML-like*. U suštini, skript predstavlja *XML* datoteku, a skript jezik je tako orjentisan, da se uklapa u pravila *XML* jezika.

Skript jezik omogućava pisanje izvršivog koda koji opisuje ponašanje komunikacionog podsistema, na način sličan programskim jezicima višeg nivoa. Jezik podržava osnovne osobine kao što su deklarisanje promenljivih različitih tipova, dodela vrednosti, selekcija i petlje. Dodatne osobine jezika su mogućnost jednostavnog slanja i primanja *IPC* zahteva, i jednostavni log ispisi. Skript jezik je optimizovan za konkretan problem, a to je opis ponašanja komunikacionog podsistema.

Sintaksni analizator interpretira naredbe jezika onako kako se skript čita, redom. U slučaju sintaksne greške *XML* tipa, skript neće moći da bude izvršen. U slučaju greške u jeziku, kao npr. navođenja nepostojeće naredbe, manjka parametara i sl, interpreter će najčešće koristiti podrazumevane vrednosti za propuštene parametre, odnosno preskočiće naredbe koje neće moći da budu izvršene, uz upozorenje na log ekranu. Postoje slučajevi grešaka u kojima će skript prestati da se izvršava. Deklarisane promenljive interpreter postavlja u bazen promenljivih u internoj memoriji, u kojoj inicijalno ima mesta za 100 promenljivih. Bazen se sastoji od niza struktura koji predstavljaju promenljivu, sa baferom nedefinisanog tipa. Ovaj bafer se kasnije razrešava tipom koji je promenljivoj dodeljen. Struktura koja opisuje promenljivu prikazana je u okviru:

```
typedef struct _var_t
{
  char name[20];
  unsigned char size;
  unsigned char type;
  unsigned char buf[255];
} var_t;
```

Sintaksna analiza jezika izvršava se rekurzivno. Osnovnu nit koja izvršava skript naziva se *ScriptExecutor*. Funkcija *ExecuteScript* pokreće interpretiranje skripta sa prosleđenim nazivom. Ova funkcija inicira memorijsku sliku u kojoj se skript izvršava, te poziva funkciju *DoBranch* radi izvršavanja grane skripta na najvišem hijerarhijskom nivou (što predstavlja samo telo skripta). U zavisnosti od sintaksnih elemenata, prozivaju se dalje interpreterske funkcije *DoFor, DoIf, DoCreateVar, DoSend, DoReceive* i sl. Ugnježđeni kod se dalje interpretira rekurzivnim pozivom funkcije *DoBranch.* Ilustracija jednog mogućeg skripta sa rekurzivnim izvršavanjem prikazana je na Slici 3.



Slika 3 – *Ilustracija interpretiranja jednog skripta*

Sprega ka korisniku je grafički jednostavno koncipirana, kao jedan dijalog okvir sa najvećim delom prostora rezervisanim za log ekran, koji služi za prikaz rezultata izvršavanja skriptova, odnosno stanja veze između platformi. Takođe, prisutna su i dugmad za pokretanje i zaustavljanje veze, okvir za unošenje naziva skripta te dugmad za njegovo pokretanje i zaustavljanje, kao i okvir za unošenje naziva režima rada trenutno izvršavanog skripta i dugme za objavljivanje tog režima.

V. XML JEZIK ZA PISANJE SKRIPTOVA

Jezik omogućava pisanje skriptova koji opisuju ponašanje komunikacionog *IPC* podsistema koprocesora, odnosno osnovne platforme. Jezik omogućava:

1. Deklarisanje promenljivih proizvoljnog tipa i dodelu vrednosti;
2. Deklarisanje uređenih bafera (struktura) uz ograničene mogućnosti upotrebe;
3. Petlje;
4. Selekcije;
5. Slanje *IPC* upravljačkog zahteva;
6. Prijem *IPC* upravljačkog zahteva uz beskonačno čekanje, ili čekanje na istek unapred određenog vremena;
7. Proizvoljan ispis na log ekran, uz mogućnost ispisa vrednosti promenljive;
8. Odloženo izvršavanje komandi (*sleep*);
9. Vezu između PC aplikacije i skripta preko zajedničke sistemske promenljive.

Za pisanje skripta koristi se *XML* sintaksa. Jedan deo posebnih pravila jezika za emulaciju naveden je u nastavku.

A. Promenljive

Promenljiva se deklariše kao u okviru na sledećoj strani (uglasta zagrada označava opcioni deo):

```
<var name="ime_promenljive"
     type="tip_promenljive"
    [val="vrednost_promenljive"]/>
```

Tipovi promeljivih mogu biti celobrojni neoznačeni (*UINT32, UINT16, UINT8*) ili celobrojni označeni (*INT32, INT16, INT8*), niz karaktera (*STRING*), odnosno proizvoljni niz bajtova (*BUFFER*). Promenljiva tipa *BUFFER* može se inicirati ugnježdavanjem promenljivih drugih tipova unutar konstrukta *var*. Promena vrednosti i tipa može se obaviti ponavljanjem deklaracije.

Sistemske promenljive počinju karakterom „@“ i prisutne su bez iniciranja, te se koriste za posebne namene. To su, npr. promenljiva @*dev_id* koja sadrži vrednost ID-a uređaja sa kojeg je stigao ili kome je poslat poslednji komunikacioni upravljački zahtev, @*size* koja čuva informaciju o veličini poslednjeg komunikacionog zahteva, ili, @*prog_mode*, koja menja vrednost u zavisnosti od akcije iz aplikacije, čime omogućava menjanje toka izvršavanja skripta u bilo kom trenutku.

B. Petlje

Ukoliko se želi da se neki deo skript koda ponavlja beskonačno, određen broj puta ili dok ne bude ispunjen neki uslov, koristi se naredba *for*.

Da bi se izvršio deo koda beskonačan broj puta, ili određen broj puta, koristimo:

```
<for [loop="5"]>
   <!-- naredbe -->
</for>
```

Ukoliko se *loop* atribut koji određuje broj ponavljanja petlje ne navede, petlja se izvršava beskonačno. Ukoliko je potrebno petlju izvršavati do nekog trenutka, određenog uslovom, petlja se može prekinuti korišćenjem naredbe:

```
<break/>
```

nakon koje će se izvršavanje najbliže *for* naredbe u hijerarhiji odmah obustaviti.

C. Selekcija

Ukoliko se želi izvršavanje različitih delova koda u zavisnosti od postavljenog uslova, koristi se naredba *if*. Naredba ima neki od sledećih formata:

```
<if>
    <case namea="ime_prom_1" [nameb="ime_prom_2"]
          val="vrednost" rel="relacija">
      <!-- kod za jedan od uslova -->
    </case>
    <default>
      <!-- kod ako nijedan uslov ne vazi -->
    </default>
</if>
```

Atribut *rel* može biti <, > ili =.

D. Prijem i slanje IPC upravljačkog zahteva

Da bi se obavio prijem IPC upravljačkog zahteva koji stiže putem IPC protokola i serijske sprege sa druge strane, koristi se naredba *receive*, na neki od sledećih načina:

```
<receive />
```

ili

```
<receive [timeout="vreme_u_ms"]>
   <var name="ime_promenljive_1"
       [type="tip_promenljive"]/>
   <var name="ime_promenljive_n"
       [type="tip_promenljive"]/>
   <!-- još polja ili promenljivih -->
</receive>
```

Dakle, *receive* naredba može da ima ili da nema ugnježđeni blok. Sama naredba aktivira čekanje nove *IPC* upravljačke poruke. Ukoliko je atribut *timeout* definisan, sa izvršavanjem skripta se nastavlja u nekom od sledeća dva slučaja:

- vreme za čekanje je isteklo, a *IPC* poruka nije pristigla za to vreme;
- *IPC* poruka je pristigla pre isteka vremena.

Ukoliko *timeout* atribut nije naveden, čekanje na IPC poruku je beskonačno. Po prijemu IPC poruke, sistemske promenljive dobijaju vrednosti odgovarajućih polja upravljačkog zahteva, a ugnježđeni *var* konstrukti (re)deklarišu promenljive i dodeljuju im vrednosti iz polja za podatke *IPC* zahteva.

Slično, obavlja se slanje zahteva korišćenjem konstrukta *send*, uz prethodnu dodelu željenih vrednosti sistemskim promenljivama.

VI. ZAKLJUČAK

Izrađena *PC* aplikacija omogućava višestruko jeftiniji, brži i fleksibilniji razvoj programske podrške za višeprocesorske sisteme između kojih treba ostvariti komunikaciju. Uopštavanje primene koje je omogućeno različitim scenarijima povezivanja i mogućnošću definisanja različitog ponašanja zahvaljujući jezičkom interpreteru, omogućava primenu ove aplikacije nad klasom srodnih problema u oblasti ugrađenih elektronskih sistema.

LITERATURA

[1] Milan Bjelica, “Jedno rešenje PC aplikacije za emulaciju komunikacionog podsistema TV arhitekture u svrhu ubrzanja razvoja programske podrške,” MicronasNIT, interna dokumentacija 2008.

[2] Milan Savić, Nikola Smiljković, Siniša Stanojlović “AVC-Accelerator-SW-Concept,” MicronasNIT, interna dokumentacija 2007.

[3] MSDN korisnička dokumentacija, Microsoft, 2006.

[4] Brian Benz, John Durant, *XML Programming Bible*. John Wiley & Sons Canada, 2003.

ABSTRACT

This paper presents an idea, concepts and implementation of a PC application for integrated circuit’s communication behaviour emulation, to be connected with another embedded system via serial interface. Application has a script language interpreter, to allow creating different communication scenarios for testing purposes. This approach supports rapid software development for embedded systems, when certain parts of hardware are yet unavailable, thus allowing simultaneous development of both hardware and software blocks.

REALIZATION OF A PC APPLICATION FOR COMMUNICATION SUBSYSTEM OF AN INTEGRATED CIRCUIT EMULATION

Milan Bjelica, Milan Savić,
Velimir Vujanović, Miodrag Temerinac