Jedno rešenje dinamičkog prilagođavanja integrisanog razvojnog okruženja na različite ciljne DSP platforme

Bojan Ćulafić, Branislav Rankov, Miodrag Đukić, Jelena Kovačević

***Sadržaj* — U radu je prikazano jedno rešenje prilagođenja razvojnog okruženja dinamičkom rekonfigurisanju u zavisnosti od izabrane platforme. Razvojno okruženje se zasniva na *Eclipse* platformi. U radu je dat kratak prikaz razvojnog okruženja, kao i prikaz potrebnih prilagođenja koja su izvedena da bi se postigao željeni cilj zadatka. U kratkim crtama je objašnjen sistem za ispitivanje i rezultati koji su dobijeni ispitivanjem.**

***Ključne reči* — DSP, dinamičko ponašanje razvojnog okruženja, XML.**

I. UVOD

DINAMIKA razvoja DSP procesora nameće zahtev za brzim razvojem alata u vidu razvojnih okruženja pomoću kojih će se za njih obezbediti razvoj programske podrške. Prilikom razvoja integrisanih razvojnih okruženja za DSP procesore teži se da razvojno okruženje podrži što širi spektar klasa ovih procesora. Podaci o podržanim DSP procesorima ili ciljnim platformama su obično upisani direktno u izvorni kod razvojnog okruženja.

Cilj ovog rada je prilagođenje izvornog koda razvojnog okruženja tako da se ono dinamički rekonfiguriše u zavisnosti od dobijenih podataka o izabranoj platformi za koju se razvija programska podrška. Realizacijom rada je omogućeno da se podaci o podržanim platformama dobijaju dinamički i da se koriste u celom razvojnom okruženju sa jednog centralnog mesta. Ovakav pristup razvoju razvojnog okruženja omogućuje rad sa različitim ciljnim platformama bez menjanja izvornog koda samog okruženja.

II. PREGLED ARHITEKTURE DSP PROCESORA

DSP procesori predstavljaju specijalizovani oblik procesora namenjenih prvenstveno obradi digitalnih signala. Njihova upotreba se poslednjih godina drastično povećala, a samim tim su postali i ključne komponente u raznim korisničkim, komunikacijskim, medicinskim i industrijskim uređajima. Razlog ovome je manja cena u odnosu na klasične procesore i veća energetska efikasnost [1].

Arhitektura DSP procesora omogućuje veću propusnu moć u odnosu na klasične procesore. Njihova arhitekrura je obično zasnovana na Harvard arhitekturi. Harvard arhitektura rešava problem „uskog grla“ prilikom učitavanja podataka iz memorije odvajanjem memorije podataka i programske memorije i dodavanjem posebnih magistrala za obe memorije [2]. Arhitektura ovih procesora obično sadrži i razna poboljšanja na fizičkom nivou, kao što su:

- Brze množačke jedinice,
- Više množačkih jedinica,
- Efikasan pristup memoriji,
- Rad sa brojevima u nepokretnom zarezu,
- Specijalizovan skup instrukcija,
- Programske petlje,
- Direktan pristup memoriji, itd.

Veći zahtevi za obradom zvuka i video slike u visokoj rezoluciji, kao i nagli razvoj telekomunikacionih uređaja, pre svega mobilnog Interneta, usmerio je razvoj DSP procesora ka povećanju broja procesorskih jezgara, registara i memorije koja je sadržana u samom procesoru u cilju povećanja propusne moći [3].

III. OPIS RAZVOJNOG OKRUŽENJA

Razvojno okruženje se zasniva na *Eclipse* platformi i namenjeno je prvenstveno za razvoj programske podrške na jednoj klasi DSP procesora, uz mogućnost proširenja i za druge ciljne platforme. *Eclipse* platforma pruža mogućnost lakog proširivanja postojeće funkcionalnosti putem komponenti (*plugin*). Komponente su međusobno povezane putem priključnih tačaka (*extension point)* i priključaka (*extension*). Komponente mogu da sadrže više priključnih tačaka i priključaka. Uparivanjem priključaka je moguće povezati dve komponente [5]. Na ovaj način je obezbeđena laka proširivost i fleksibilnost razvojne platforme.

Razvojni sistem se sastoji od dve odvojene celine povezane putem TCP sprege, kao što je to prikazano na

Ovaj rad je delimično finansiran od Ministarstva za nauku Republike Srbije, projekat 12004,2008.

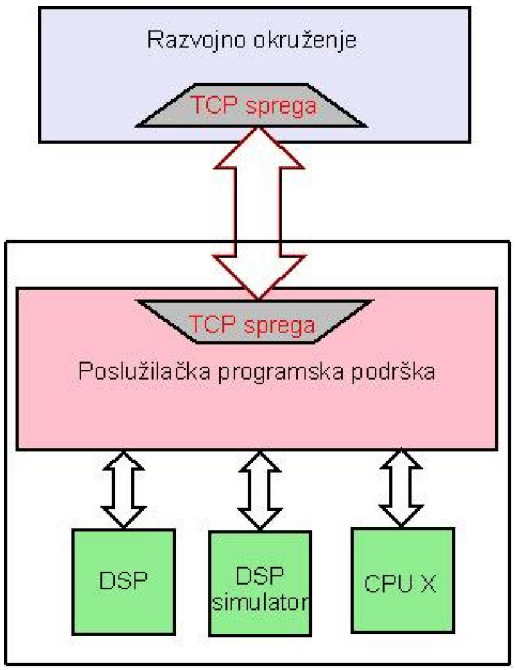
Bojan Ćulafić, Fakultet tehničkih nauka u Novom Sadu, Srbija, (email: bojan.culafic@rt-sp.com)

Branislav Rankov, RT-SP, Vojvode Šupljikca 26, Novi Sad, Srbija, (email: branislav.rankov@rt-sp.com)

Miodrag Đukić, Fakultet tehničkih nauka u Novom Sadu, Srbija, (email: miodrag.djukic@rt-sp.com)

Jelena Kovačević, Fakultet tehničkih nauka u Novom Sadu, Srbija, (email: jelena.kovacevic@rt-sp.com)

Sl. 1.



Sl. 1. Blok šema celina razvojnog sistema. Blok pod nazivom *CPU X* predstavlja jedno moguće proširenje poslužilačke programske podrške.

Razvojno okruženje dobija sve potrebne podatke koji su neophodni za kontrolisano izvršenje realizovane programske podrške putem TCP sprege.

Poslužilačka programska podrška za pružanje podataka o podržanim platformama (u daljem tekstu poslužilac) predstavlja spregu između ciljnih platformi (razvojnih ploča, simulatora) i razvojnog okruženja. U poslužiocu su sadržani svi podaci o platformama za koje je moguće razvijati programsku podršku. Preko poslužioca je moguće dobiti podatke o trenutno podržanim platformama, o trenutno aktivnoj platformi, kao i podatke o vrednostima upisanim u procesorske registre i memorijske lokacije aktivne platforme. Poslužioc je razvijen u programskom jeziku C++. Podaci o podržanim simulatorima se nalaze u posebnoj DLL (*Dynamic-link library*) biblioteci koja se dinamički učitava kada se pojavi potreba za simulatorom. Na ovaj način je obezbeđen jednostavan mehanizam za dodavanje novih simulatora u vidu DLL biblioteka koji ne moraju biti vezani samo za DSP procesore.

Razvojno okruženje i poslužioc su pokrenuti na istom računaru.

IV. Opis realizacije

Realizacija se sastoji iz više koraka:

- Definisanje sadržaja XML (*Extensible Markup Language*) datoteka za opis ciljnih platformi;
- Prilagođenje izvornog koda razvojnog okruženja;
- Prilagođenje izvornog koda poslužioca.

Pored toga je realizovan i skup klasa za opis platformi koje su implementirane u izvorni kod razvojnog okruženja.

A. Definisanje XML datoteka

Za potrebe zadatka definisane su datoteke koje sadrže podatke o podržanim platformama sa kojima se može raditi u razvojnom okruženju. Bilo je potrebno obezbediti dobru čitljivost i laku proširivost tih datoteka. Za opis podataka u datotekama izabran je XML jezik zbog svoje proširivosti i tekstualnog oblika koji je lako čitljiv. XML je proširivi jezik za označavanje (*markup*) tekstualnih dokumenata [6].

Definisane su dve XML datoteke koje sadrže podatke o platformama

Na Sl. 2. je prikazan oblik XML datoteke koja sadrži podatke o podržanim platformama u razvojnom okruženju.

U ovoj datoteci se definišu:

- Sve podržane platforme,
- Nazivi tih platformi,
- Jezgra svake platforme,
- Ime za svako navedeno jezgro.

```
<targetinfo_list>
 <target>
 <target_name>ime_uređaja</target_name>
  <cores>
   <core>
    <core_name>ime_jezgra </core_name>
   </core>
   <core>
    <core_name>ime_jezgra </core_name>
   </core>
  </cores>
 </target>
</targetinfo_list>
```

Sl. 2. Oblik XML datoteke koja opisuje podržane platforme u razvojnom okruženju

Sa Sl. 2. se vidi da je početak i kraj datoteke označen XML oznakom (*XML tag*) <*targetinfo_list*> između kojih se definišu podržane platforme. Svaki novi opis platforme počinje oznakom <*target*>. Između ovih oznaka se definišu osnovni podaci o platformi. Naziv platforme se definiše između oznaka <*target_name*>, a između oznaka <*cores*> se definišu jezgra platforme. Svaka platforma može imati proizvoljan broj jezgara koja se definišu između oznaka <*core*>. Jedini podatak o jezgru podržane platforme je njegovo ime, koje se upisuje između oznaka <*core_name*>.

Na Sl. 3. je prikazan oblik XML datoteke koja sadrži detaljniji opis platforme.

Podaci koji se definišu o ovoj datoteci su:

- Ime uređaja na koji se odnosi datoteka,
- Definicija svakog jezgra platforme,
- Ime definisanog jezgra,
- Memorijski prostor svakog jezgra,
- Memorijski segmenti unutar memorijskog prostora,
- Podaci o svakom memorijskom segmentu,
- Registri sadržani u jezgru,
- Grupe registara,

- Nazivi navedenih grupa,
- Registri u okviru navedene grupe,
- Ime svakog registra,
- Tip navedenih registara.

```
<target_info>
 <target_name>ime</target_name>
 <core_info>
  <core_name>ime_jezgra</core_name>
  <memory_map>
   <memory_segment>
    <zone>mem_zona</zone>
    <kind></kind>
    <start_addr>početna_adresa</start_addr>
    <end_addr>krajnja_adresa</end_addr>
   </memory_segment>
  <memory_map>
  <registers>
   <group>
   <group_name>ime_grupe</group_name>
    <register>
    <reg_name>ime_registra</reg_name>
    <reg_type>tip_registra</reg_type>
   </register>
   </group>
  </registers>
 </core_info>
</target_info>
```

Sl. 3. Oblik XML datoteke koja sadrži šire podatke o platformi

Datoteka počinje i završava se oznakom <*target_info*> između kojih se nalaze detaljni podaci o platformi. U ovoj datoteci je moguće definisati memorijske segmente za svako jezgro, kao i njegove registre. Definisanje memorijskog segmenta je predviđeno između oznaka <*memory_segment*>. Bliži opis memorijskog segmenta, kao što je navođenje zone memorije kojoj pripada, tipa memorije, početne i krajnje adrese memorijskog segmenta, kao i navođenje da li je moguće upisivati sadržaj u memorijski segment se obavlja između oznaka <*zone*>, <*kind*>, <*start_addr*>, <*end_addr*> i <*writeable*>. Svako jezgro može imati više ovakvih memorijskih segmenata koji su definisani između oznaka <*memory_map*>. Na ovaj način je omogućen opis strukture memorije raznih ciljnih platformi.

Pored memorijskih segmenata svako jezgro se opisuje i skupom registara. Registri se definišu između oznaka <*registers*> i podeljeni su u grupe. Svaka grupa ima svoj naziv koji se stavlja između oznaka <*group_name*>. Unutar svake grupe se definišu registri. Svaki registar je opisan svojim imenom i tipom između oznaka <*reg_name*> i <*reg_type*>.

Koristeći oblik XML datoteke prikazane na Sl. 3 moguće je opisati sva jezgra ciljne platforme sa memorijskim prostorom i registrima koje određena jezgra sadrže i na taj način omogućiti rad sa tom platformom u okviru razvojnog okruženja.

B. Prilagođenje razvojnog okruženja i poslužilačke programske podrške

Cilj prilagođenja razvojnog okruženja je da se podaci o ciljnim platformama dobijaju putem TCP sprege od poslužioca i da se pomoću dobijenih podataka dinamički rekonfiguriše razvojno okruženje za svaku ciljnu platformu posebno.

Prilikom pokretanja kontrolisanog izvršenja programske podrške, razvojno okruženje prima znakovni niz uređen po XML specifikaciji od poslužioca putem TCP sprege koji sadrži podatke o izabranoj platformi. On se u razvojnom okruženju parsira pomoću realizovanog parsera XML datoteka, a podaci dobijeni parsiranjem se učitavaju u realizovani skup klasa za opis platforme. Implementacija klasa za opis platforme je izvedena tako da im se pristupa statički iz izvornog koda razvojnog okruženja, odnosno u svakom trenutku postoji samo jedna instanca klase koja daje podatke o platformama. Na ovaj način su podaci o platformama centralizovani unutar izvornog koda razvojnog okruženja čime je otklonjena potreba za višestrukim postojanjem ovih podataka.

Podaci o registrima i memorijskim segmentima se učitavaju samo u instancu klase koja opisuje izabranu platformu. Po zaustavljanju kontrolisanog izvršenja programske podrške ovi podaci se brišu iz instance klase.

U okviru razvojnog okruženja su sadržani preglednik registara i preglednik memorije. Ovi preglednici se rekonfigurišu prilikom pokretanja kontrolisanog izvršenja programske podrške na osnovu učitanih podataka o izabranoj platformi.

Preglednik registara, putem realizovane klase, učitava podatke o registrima iz instance klase koja opisuje trenutno izabranu platformu i upisuje ih u stabla (*tree*) koja se potom prikazuju u pregleniku. Preglednik može prikazati bilo koji skup registara opisan u dobijenim podacima. Po prekidu kontrolisanog izvršenja programske podrške, podaci o registrima se brišu, pa samim tim i preglenik registara ne prikazuje registre.

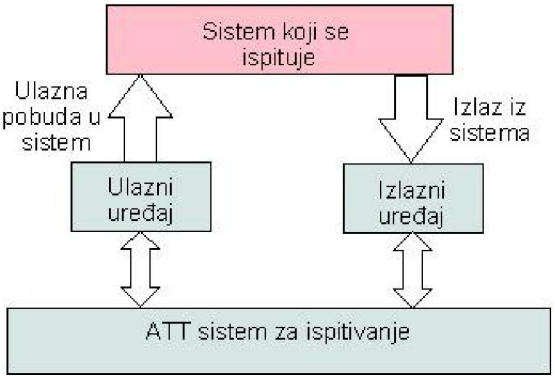
Preglednik memorije omogućuje pregled sadržaja memorijskih segmenata koji su sadržani u nekoj platformi. Podaci o memorijskim segmentima koji mogu da se prikažu u pregledniku memorije se čitaju iz instance klase koja opisuje trenutno aktivnu platformu. Prekidom kontrolisanog izvršenja programske podrške za izabranu platformu se gubi mogućnost pregleda memorijskih segmenta usled nedostatka podataka o njima.

Prilagođenje izvornog koda poslužioca se odnosi na realizaciju funkcije koja obezbeđuje podatke ciljnoj platformi. Ova funkcija treba da se realizuje u svim trenutno podržanim platformama u okviru poslužioca. Zadatak funkcije je učitavanje svih podataka o izabranoj platformi i smeštanje tih podataka u znakovni niz uređen po XML specifikaciji. Dobijeni znakovni niz se šalje razvojnom okruženju prilikom pokretanja kontrolisanog izvršenja programske podrške za izabranu platformu.

V. ISPITIVANJE

Realizacija rešenja je ispitana pomoću programske podrške ATT (*Automatic Test Tool*) koja je razvijena na odseku za računarsku tehniku i računarske komunikacije Fakulteta Tehničkih Nauka u Novom Sadu. ATT platforma je razvijena na principu crne kutije što znači da

je moguće ispitivati razne sisteme bez poznavanja njihove unutrašnje strukture. Platforma za ispitivanje se sastoji od uređaja koji predstavljaju osnovnu jedinicu funkcionalnosti sistema, a ujedno su i sprega sistema za ispitivanje sa sistemom koji se ispituje. Uređaji su realizovani u obliku DLL biblioteka, što omogućuje lako proširenje sistema za ispitivanje novim uređajima. Na Sl. 4. je prikazan princip ispitivanja pomoću ATT sistema za ispitivanje.



Sl. 4. Princip ispitivanja pomoću ATT sistema za ispitivanje.

Rešenje je ispitano pomoću ATT sistema za ispitivanje i programske podrške *diff.exe*. Programska podrška *diff.exe* služi za poređenje sadržaja dve datoteke. Pošto je funkcija realizovanih klasa isključivo opis uređaja i pružanje podataka o njima, ispitivanje je obavljeno upravo u ovim sferama, učitavanju i pružanju učitanih podataka.

Provedene su dve grupe ispitnih slučajeva od kojih se prvi odnosi na podatke koji se učitavaju iz XML datoteke koja sadrži osnovne podatke o podržanim uređajima, a drugi skup ispitnih slučajeva se odnosi na šire podatke o uređaju, koji se takođe dobijaju u XML obliku. Za potrebe ispitivanja realizovana je konzolna programska podrška koja omogućuje učitavanje podataka u realizovane klase, a kao izlaz daje XML oblikovanu datoteku dobijenu iščitavanjem podataka iz tih klasa. Pored toga ona i upoređuje ulazne i izlazne datoteke pomoću *diff.exe* programske podrške. Prilikom pokretanja ove programske podrške potrebno joj je proslediti putanju do ulazne XML datoteke, putanju do izlazne XML datoteke i putanju do datoteke u koju će se upisivati rezultati poređenja (*log*).

Obe grupe ispitnih slučajeva se sastoje od po dva skupa ispitnih slučajeva od kojih rezultat prvih treba da bude pozitivan, a drugih negativan. Za obe grupe ispitnih slučajeva su napravljeni ispitni koraci u ATT sistemu za ispitivanje, koristeći uređaj *CmdLine*. Putem uređaja *CmdLine* moguće je pozivati proizvoljne konzolne programske podrške iz ATT sistema. Ispitni slučajevi se pokreću u ATT sistemu, a zatim se vrši provera dobijenih rezultata koji se nalaze u datoteci sa rezultatima. Za svako uspešno poređenje dve datoteke, programska podrška upisuje *true* u datoteku sa rezultatima, a za neuspešno poređenje *false*.

Rezultati ispitivanja su očekivani. Ulazne XML datoteke koje su pravilno oblikovane se učitavaju u klase i na izlazu iz programske podrške će biti izlazna XML datoteku identična ulaznoj. Nepravilno oblikovane ulazne XML datoteke prouzrokuju generisanje izuzetka od strane korišćenog parsera XML dokumenata, podaci o uređajim nisu učitani u klase, što dovodi do ne postojanja podataka u izlaznoj XML datoteci, a samim tim i negativnog rezultata ispitnog slučaja.

VI. ZAKLJUČAK

Prednost realizovanog rešenja u odnosu na slična je ta što se na ovaj način dobilo dinamičko rekonfigurisanje razvojnog okruženja u odnosu na podatke koje razvojno okruženje dobija o izabranoj platformi. Ovo znači da je moguće izmeniti postojeću ili dodati potpuno novu platformu u okviru poslužioca i ona će biti automatski podržana i u razvojnom okruženju bez potrebe za menjanjem samog izvornog koda razvojnog okruženja. Dobijenim rešenjem je omogućen rad razvojnog okruženja i sa višejezgarnim sistemima

Realizacijom ovog zadatka je značajno ubrzan proces dodavanja novih platformi koje su podržane razvojnim okruženjem i smanjena je mogućnost greške u podacima koje opisuju platformu, jer su ovi podaci centralizovani.

Izborom XML jezika za opis podataka o ciljnim platformama, ostavljena je mogućnost lakog dodavanja novih platformi koje ne moraju biti vezane samo za DSP arhitekturu, što može predstavljati mogući smer daljeg razvoja ovog rešenja.

LITERATURA

[1] Jennifer Eyre , Jeff Bier, *The Evolution of DSP Processors, IEEE Signal Processing Magazine,* 2000.

[2] V. Kovačević, M. Popović, M. Temerinac, N Teslić, *Arhitekture i algoritmi digitalnih signal procesora I*, Fakultet tehničkih nauka, Novi Sad, 2005.

[3] Lina J. Karam, Ismail AlKamal, Alan Gatherer,Gene A. Frantz, David V. Anderson, Brian L. Evans, *Trends in multi-core DSP platforms, IEEE Signal Processing Magazine,* 2009.

[4] Branislav Rankov, *Jedan pristup proširenju sistemskih programskih alata na osnovu standarda DWARF 2*, diplomski – master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2009.

[5] Wayne Beaton, Jeff McAffer, *Eclipse Rich Client Platform*, Eclipse Foundation, Inc, 2006

[6] James Duncan Davidson, *Java API for XML Parsing,* Sun Microsystems, Inc, 2000

ABSTRACT

This paper presents one solution of changing Eclipse based IDE to achieve dynamic behavior dependent on chosen platform. It gives basic information about developed IDE and changes made to fulfill demands of dynamic behavior. Last part of this paper explains testing system which is used and testing results.

ONE IMPLEMENTATION OF IDE WITH CAPABILITY OF DYNAMIC ADAPTATION TO DIFFERENT TARGET DSP PLATFORMS

Bojan Ćulafić, Branislav Rankov, Miodrag Đukić, Jelena Kovačević