Jedno rešenje ispitivanja sprežne komponente za kontrolisano izvršenje programa na digitalnim signal procesorima

Branislav Rankov, Marko Krnjetin, Miodrag Đukić, Jelena Kovačević

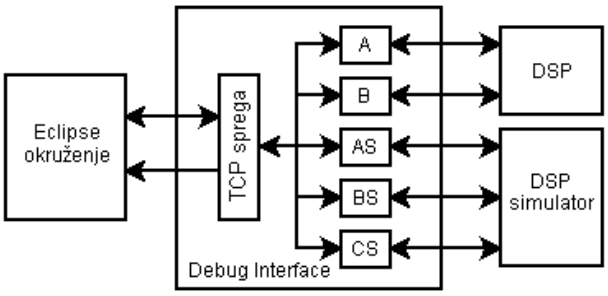
***Sadržaj* — U ovom radu je opisano ispitivanje sprežne komponente jednog rešenja programske podrške za kontrolisano izvršenje programa. Prikazani su bitni aspekti programske podrške kao celine i ispitivane komponente posebno. Opisana je platforma za automatsko ispitivanje i važne grupe ispitnih slučaja.**

***Ključne reči* — DSP, ispitivanje, kontrolisano izvršenje programa, otkrivanje grešaka**

I. Uvod

Programska podrška za kontrolisano izvršenje programa (debugger) omogućava izvršavanje programa korak po korak ili do određene tačke prekida (breakpoint), i time daje mogućnost pronalaženja i ukljanjanja grešaka u programu. Ona predstavlja spregu između korisnika i ciljne platforme na kojoj se program izvršava. Više o ovoj vrsti programske podrške se može pročitati u [1].

Konkretna realizacija programske podrške za kontrolisano izvršenje programa podeljena je na dve osnovne komponente kao što je prikazano na sl. 1. Prva komponenta, nazvana *eclipse okruženje*, se odnosi na spregu sa korisnikom, grafički prikaz, rukovanje izvornim kodom i potrebnim informacijama. Druga komponenta sistema, nazvana *debug interface* (skaćeno DI) obezbeđuje jedinstvenu spregu prema podržanim ciljnim platformama za izvršavanje programa.



Sl. 1. Struktura programske podrške za kontrolisano izvršenje programa

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

Branislav Rankov, Fakultet tehničkih nauka u Novom Sadu, Srbija, (email: branislav.rankov@rt-sp.com)

Marko Krnjetin, RT-SP, Vojvode Šuplikca 26, 2100 Novi Sad, Srbija, (email: marko.krnjetin@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)

Tema ovog rada je ispitivanje funkcionalnosti DI komponente. Cilj ispitivanja je proveravanje da li je DI ispravno realizovan za sve podržane ciljne platforme. Slična ispitivanja se vrše u sklopu GDB projekta (*The GNU Project Debugger*). U slučaju GDB-a korišćeno je *DejaGNU* okruženje za automatsko ispitivanje. Opis ispitivanja u GDB-u je delimično dat u [2], a *DejaGNU* okruženje je opisano u [3].

Debug interface je sprežna komponenta ka ciljnoj platformi. Opis njene realizacije se nalazi u [4]. DI definiše osnovne funkcionalnosti koje su potrebne za kontrolu bilo koje ciljne platforme. Osnovne funkcionalnosti DI su: izvršenje jedne instrukcije sa ulaskom u potprograme (stepinto) i bez ulaska u potprograme (stepover) , pokretanje izvršenja programa (run), postavljanje i uklanjanje tačke prekida (breakpoint), zaustavljanje izvršenja (pause), čitanje i pisanje vrednosti registra ili bloka memorijskih lokacija. Za svaku od podržanih ciljnih platformi posebno se realizuju osnovne funkcije DI-a za kontrolu te platforme, što se naziva realizacija DI-a. U određenom trenutku može biti aktivna samo jedna realizacija DI-a, što znači da DI može kontrolisati samo jednu ciljnu platformu. Ciljna platforma sa kojom radi DI se bira pri njegovom pokretanju putem parametra komandne linije.

Trenutno postoje pet realizacija DI: A, B, AS, BS i CS. Prve dve realizacije, A i B, pružaju podršku za dva tipa DSP-a (skr. digitalni signal procesor). Razlika između procesora tipa A i B je u tome što tip A sadrži dva DSP jezgra, dok procesor tipa B sadrži jedno DSP jezgro. Druge dve komponente, AS i BS se odnose na simulatore ove dve vrste procesora, a komponenta CS predstavlja simulator odvojenog DSP jezgra bez perifernih uređaja koje sadrže procesori. Podrška za procesore sa više jezgara je ostvarena tako što sve funkcije DI imaju parametar koji identifikuje jezgro nad kojim funkcija treba da se izvrši. Ovaj parametar se može ignorisati kod realizacije DI komponenti za kontrolu procesora sa jednim jezgrom.

DI komunicira sa *eclipse* okruženjem mehanizmom udaljenog poziva funkcije (remote procedure call). Ovaj mehanizam je realizovan putem TCP protokola . DI u ovoj komunikaciji predstavlja komponentu posluživanja (server) i pri pokretanju osluškje dva TCP kanala (port) na koje se *eclipse okruženje* povezuje. Jedan kanal služi za slanje zahteva DI-u, kao i prijem odgovora, a drugi kanal služi za prijavljivanje asinhronih događaja od strane DI-a. Poruke koje se razmenjuju putem TCP kanala se razdvajaju znakom nove linije (carriage return) i sastoje se

od nekoliko reči. Prva reč u poruci koja prenosi zahtev je ime zahteva. Komponenta TCP sprege na osnovu imena zahteva identifikuje funkciju koju treba da pozove unutar DI, dok se ostatak poruke prevodi u parametre te funkcije. Poruke koje prenose događaje identifikuju stanje u kome se nalazi DI odnosno ciljna platforma.

TCP je izabran kao metod komunikacije jer predstavlja standardnu spregu koja se lako realizuje u oba korišćena programska jezika, Java na strani Eclipse okruženja i C++ na strani DI. TCP takođe pruža mogućnost proširenja sistema u kom bi se podržalo izvršavanje DI na udaljenom računaru, na koji je povezana razvojna ploča. Pored ovih prednosti TCP olakšava ispitivanje DI-a, jer se DI lako odvaja od *eclipse okruženja* i povezuje na platformu za ispitivanje.

Budući da DI kao ciljnu platformu podržava procesore za digitalnu obradu signala, za ispitivanje DI je potrebno znati osnovne informacije o ovim procesorima. DSP procesori su zasnovani na Harvard arhitekturi, što znači da imaju odvojene magistrale za pristup programskoj memoriji i memoriji podataka. Ova osobina zajedno sa protočnom strukturom procesora omogućava paralelan pristup instrukcijama i podacima. Ovi odvojeni delovi memorije se nazivaju memorijske zone. DI podražava DSP procesore koji pored programske zone i zone podataka imaju i dodatnu zonu podataka. Ova dodatna zona služi za paralelizaciju instrukcija koje imaju dva parametra. Pored RAM memorije DSP procesori imaju integrisanu i ROM memoriju, i ona se nalazi u istom adresnom prostoru kao i RAM memorija, tako da svaka memorijska zona ima ROM i RAM deo. Više informacija o DSP procesorima se može naći u [5].

DI je deo složene programske podrške za kontrolisano izvršenje programa zasnovane na *eclipse* platformi. *Eclipse* je skup projekata koji imaju za cilj stvaranje proširive platforme za razvoj i održavanje programske podrške kroz ceo njen životni ciklus. *Eclipse* projekti između ostalog obuhvataju: integrisano razvojno okruženje za programski jezik *Java*, razvojno okruženje za C i C++ programske jezike, razvojno okruženje za internet aplikacije, razvoj samostalnih aplikacija i drugo. Proširivost *eclipse* platforme se ogleda u tome što je sastavljena od komponenti (plugin) koje su međusobno povezane mehanizmom priključnih tačaka (extension point) i priključaka (extension). Komponente mogu sadržati više priključnih tačaka i priključaka. Dve komponente se povezuju uparivanjem priključne tačke jedne komponente sa priključkom druge komponente. Na priključnu tačku se mogu povezivati različite komponente ukoliko realizuju isti priključak. Ovim mehanizmom se obezbeđuje proširivost i fleksibilnost platforme. Komponente za eclipse kao i ceo eclipse se razvijaju u programskom jeziku Java, dok se komponente međusobno povezuju putem XML datoteka.

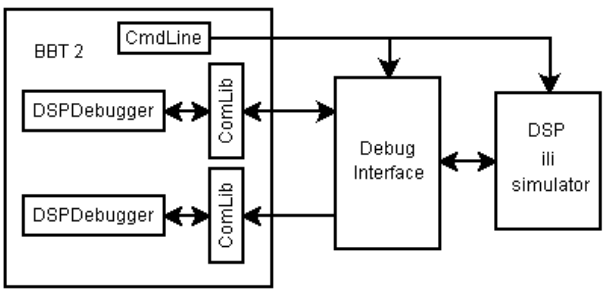
Pri realizaciji *eclipse okruženja* opisanog u ovom radu *eclipse* platforma je korišćena kao proširivo integrisano razvojno okruženje. Zadržano je osnovno radno okruženje *eclipse*-a i dodate su nove komponente kao što su editor za asemblerski jezik podržanih DSP procesora, komponente za čitanje datoteka koje generiše asembler i komponente za povezivanje *eclipse* okruženja sa DI.

II. Okruženje za automatsko ispitivanje

Za automatsko ispitivanje DI korišćena je platforma ATT (skr. Automatic Test Tool) razvijena na odseku za računarsku tehniku i računarske komunikacije fakulteta tehničkih nauka u Novom Sadu. ATT je platforma za automatsko ispitivanje sistema zasnovana na principu crne kutije. Princip crne kutije podrazumeva ispitivanje sistema bez poznavanja njegove unutrašnje strukture. Po ovom principu sistem se ispituje izvršavanjem niza ispitnih slučaja od kojih svaki ispituje jednu funkcionalnost datog sistema. Svaki ispitni slučaj definiše vrednosti ulaza ispitivanog sistema i očekivane rezultate.

Osnovna jedinica funkcionalnosti ATT platforme je uređaj (device). Uređaji predstavljaju spregu prema ispitivanom sistemu. Svaki uređaj podržava jednu klasu ispitivanih sistema i definiše skup operacija koje se nad tom klasom sistema mogu izvršiti posredstvom ovog uređaja. Operacije nad uređajima mogu zadavati ulaze ispitivanom sistemu, preuzimati rezultate i porediti dobijene rezultate sa očekivanim. Ispitni slučaji se definišu kao niz operacija izvršenih nad uređajima. Ispitni slučaj se smatra uspešno izvršenim ukoliko nema neuspešno izvršenih operacija i ako je bar jedna operacija uspešno izvršena. ATT platforma se može proširiti dodavanjem novih uređaja u obliku DLL biblioteke pisane u nekom standardnom programskom jeziku, npr. C++.

Za ispitivanje DI korišćena su tri uređaja: *CmdLine*, *CommLib* i *DSPDebugger*. *CmdLine* je uređaj koji omogućava izvršavanje spoljnih programa putem komandne linije. U ovom slučaju *CmdLine* uređaj je korišćen za pokretanje DI-a i za pozivanje rutine koja spušta program na ciljnu platformu. Drugi korišćen uređaj, *CommLib* omogućava komunikaciju putem TCP protokola. I treći uređaj, DSPDebugger je razvijen posebno za potrebe ispitivanja DI-a. On koristi operacije CommLib-a i dodaje mogućnost slanja i prijema poruka u formatu koji zahteva DI. Struktura ovog okruženja za ispitivanje prikazana je na sl. 2.



Sl. 2. Povezivanje DI sa ATT platformom

Da bi se ispitivanje usmerilo na DI komponentu poželjno je da se ona odvoji od ostatka sistema. Odvajajnje se postiže tako što se putem TCP veze poveže ATT platforma umesto *eclipse* okruženja. Sa druge strane na DI se povezuje jedna po jedna podržana ciljna

platforma da bi se ispitala realizacija DI za svaku platformu posebno.

Na ciljnoj platformi se za vreme ispitivanja izvršava posebna aplikacija koja sadrži programski kod i memorijske blokove potrebne za izvršavanje pojedinih ispitnih slučaja. Ova aplikacija se ponaša isto za sve podržane platforme, što znači da se sve funkcije i svi blokovi memorije nalaze na istim adresama na svim platformama. Ovo je izrazito važno da bi se ispitni slučaji napisani za jednu platformu mogli lako iskoristiti za sve ostale platforme.

III. Opis pojednih ispitnih slučaja

Struktura ispitnih slučaja je organizovana tako da se isti ispitni slučaji uz minimalne prepravke koriste za sve podržane ciljne platforme. Ovo se postiže tako što svi ispitni slučaji za jednu ciljnu platformu imaju iste početne korake, a za drugu platformu se kopiraju ispitni slučaji i menjaju se samo parametri ovih koraka. Prvi korak svakog ispitnog slučaja je pozivanje rutine koja upisuje program ispitne aplikacije na ciljnu platformu. Sledeći korak pokreće DI sa parametrom koji označava tip ciljne platforme. Pri pokretanju DI koristi se MSDOS komanda *start* da bi se moglo nastaviti izvršavanje ispitnog slučaja dok je DI pokrenut. Ova dva koraka koriste *CmdLine* uređaj. Zatim sledi iniciranje *DSPDebugger* uređaja i slanje zahteva za povezivanje na ciljnu platformu. Posle ovih koraka se izvršavaju koraci specifični za pojedina ispitivanja. Svako ispitivanje se završava pozivanjem zahteva za gašenje DI i pozivom MSDOS komande *taskkill* za slučaj da se desila neka greška u izvršavanju i da nije moguće regularno ugasiti DI. Ovakva struktura svakog ispitnog slučaja omogućava da se DI ponovo startuje i program ponovo upisuje na ciljnu platformu na početku svakog ispitnog slučaja, čime se postiže međusobna nezavisnost ispitnih slučaja a za uzvrat se žrtvuje vreme potrebno za upis programa na ciljnu platformu.

Na osnovu toga koja funkcionalnost DI-a se ispituje ispitni slučaji se dele na sledeće grupe:

- ispitivanje upisa i čitanja registara
- ispitivanje upisa i čitanja bloka memorije
- ispitivanje zaustavljanja izvršenja
- ispitivanje izvršenja instrukcije bez ulaska u potprograme
- ispitivanje izvršenja instrukcije sa ulaskom u potprograme
- ispitivanje programski podržane tačke prekida
- ispitivanje postavljanja i uklanjanja tačke prekida
- ispitivanje TCP veze

Ispitivanje čitanja registara se izvodi tako što se postavi tačka prekida na kraju funkcije koja postavlja registre na prethodno definisane vrednosti i sačeka se da se izvršavanje zaustavi. Potom se izda zahtev za čitanje registara i proveri da li su vrednosti ispravno pročitane. Upis vrednosti u registre se izvodi zaustavljanjem procesora na istom mestu kao i kod čitanja, ali se upisuju vrednosti koje se razlikuju od inicijalnih vrednosti. Zatim se čitaju vrednosti registara i upoređuju sa upisanim vrednostima. Ovom metodom je moguće ispitati upis i čitanje samo radnih registara, jer mnogi kontrolni i registri specijalne namene menjaju vrednost izvršenjem svake instrukcije.

Čitanje meorijskog bloka se ispituje tako što se čita blok memorije koji se pri upisu programa inicira prethodno definisanim vrednostima, a zatim se proverava da li je blok ispravno pročitan. Ova procedura se ponavlja za sve memorijske zone koje poseduje DSP. Upis memorijskog bloka se ispituje tako što se prvo upišu prethodno definisani podaci u neiniciran blok memorije, a zatim se taj isti blok čita i porede se dobijene sa početnim vrednostima. Kao i kod čitanja ispitivanje se ponavlja za sve memorijske zone. Specijalni slučaj ovoga ispitivanja je pokušaj upisa u ROM memoriju kada DI treba da vrati poruku o nemogućnosti izvršenja zahteva. Ispitivanje upisa bloka memorije pretpostavlja da je prethodno ispitana funkcija čitanja.

Zahtev za zaustavljanje izvršenja programa se ispituje tako što se pošalje zahtev, a zatim se očekuje dojava događaja prekidanja izvršenja. Posle ovoga se dva puta čita vrednost programskog brojača i proverava da li se njegova vrednost menja da bi se ustanovilo da li je izvršavanje programa zaista zaustavljeno.

Izvršavanje jedne instrukcije sa ulaskom u potprograme (stepinto) se ispituje u dva slučaja. Prvi slučaj je izvršenje intstrukcije koja nije instukcija poziva potprograma, a drugi izvršenje instrukcije poziva potprograma. Oba slučaja se ispituju zajedno tako što se postavi tačka prekida na adresi pre instrukcije poziva potprograma, a zatim se dva puta pošalje zahtev za izvršenje jednog koraka. Posle svakog koraka se ispituju prijavljeni događaji. Posle prvog koraka treba da se dobije događaj pokretanja izvršenja i događaj zaustavljanja izvršenja na adresi instrukcije poziva potprograma. Posle drugog koraka treba da se dobiju događaj početka izvršenja i događaj zaustavljanja izvršenja programa na prvoj adresi pozvanog potprograma.

Izvršavanje jedne instrukcije bez ulaska u potprograme (stepover) se ispituje na isti način kao i izvršavanje instrukcije sa ulaskom u potprograme, s time što se posle drugog koraka očekuje da se dobije događaj zaustavljanja izvršenja programa na adresi za jedan većoj od adrese instrukcije poziva potprograma.

Ispitivanje programski podržane tačke prekida je vezano za pojedine realizacije DI jer mnogi DSP procesori nemaju fizičku podršku za tačke prekida, i zbog toga DI realizuje programski podržane tačke prekida za takve DSP procesore. Programski podržane tačke prekida se realizuju upisivanjem posebne instrukcije u programsku memoriju na adresu na koju treba da se postavi tačka prekida. Ova instrukcija zaustavlja izvršenje programa na procesoru, a DI reaguje na zaustavljanje izvršenja i dojavljuje događaj eclipse okruženju. DI čuva kod instrukcije koja je zamenjena instrukcijom tačke prekida, i kada se izda zahtev za nastavak izvršenja programa, DI izvršava sačuvanu instrukciju naprocesoru i nastavlja izvršenje programa. Kada je potrebno ukloniti tačku prekida DI

vraća originalnu instrukciju na njeno mesto.

Ovakva realizacija tačke prekida zahteva ispitivanje da li se originalna instrukcija zaista izvršava kada se posle zaustavljanja na tački prekida izda zahtev za nastavljanje izvršenja ili zahtev za izvršenje jedne instrukcije. Ovaj slučaj se ispituje tako što se postavi tačka prekida na instrukciju u kodu koja menja vrednost jednog registra i u slučaju ispitivanja zahteva za nastavak izvršenja postavi se još jedna tačka prekida iza prve tačke prekida, a zatim se pošalje zahtev za nastavak izvršenja ili zahtev za izvršenje jedne instrukcije. Kada se primi događaj o zaustavljnju izvršenja, čita se vrednost datog registra i proverava se da li se ta vrednost promenila. Ukoliko se vrednost registra promenila znači da je zamenjena instrukcija zaista izvršena. Ovu grupu ispitnih slučaja nije potrebno izvršavati nad simulatorima , jer se kod simulatora tačke prekida realizuju drugačije.

Funkcije za postavljanje i uklanjanje tačaka prekida se ispituju tako što se posle izdavanja zahteva za postavljanje ispituje da li je upisana instrukcija tačke prekida na odgovarajuću lokaciju u programskoj memoriji i proverava se da li DI šalje poruku da je prekinuto izvršavanje programa. Zatim se šalje zahtev za uklanjanje te prekidne tačke i ispituje se da li je vraćena originalna instrukcija na svoju lokaciju u memoriji. Specijalani slučaji su ispitivanje postavljanja tačke prekida u ROM deo programske memorije, pokušaj postavljanja tačke prekida na memorijsku lokaciju na kojoj već postoji tačka prekida i pokušaj uklanjanja nepostojeće tačke prekida. U ovim slučajevima se očekuje poruka o nemogućnosti izvršenja zahteva. U slučaju dvostrukog postavljanja tačke prekida se proverava da li je moguće ukloniti tačku prekida i da li je posle uklanjanja originalna instrukcija vraćena na svoju memorijsku lokaciju.

Posebna grupa ispitnih slučaja se odnosi na ispitivanje komponente koja omogućije TCP komunikaciju. Ovi ispitni slučaji se odnose na zadavanje niza zahteva koji nisu ispravno formulisani i očekivanje odgovora o nemogućnosti izvršenja.

Ovim je zaokruženo ispitivanje osnovnih funkcionalnosti *debug interface* komponente.

IV. Zaključak

Primenom ispitivanja opisanog u ovom radu pronađen je određen broj do tada nepoznatih grešaka u realizaciji DI. Takođe je utvrđeno je da li prethodno poznate greške pronađene u realizaciji DI za jednu platformu postoje u realizacijama za ostale platforme. Ovo je omogućeno univerzalnošću načina ispitivanja. Univerzalnost ispitivanja se ogleda u tome što se svi ispitni slučaji napisani za jednu ciljnu platformu lako prenose na drugu platformu. Ako bi se DI proširio da podrži novu DSP platformu, većina ispitnih slučaja bi se mogla preuzeti uz pisanje ispitne aplikacije za tu platformu i minimalne izmene ispitnih sličaja. Ispitna aplikacija bi u tom slučaju trebala da sadrži potrebne funkcije i memorijske blokove na adresama koje se koriste u ispitivanjima.

Pored osnovne namene, inicijalnog ispitivanja sistema, ovaj način ispitivanja DI-a se primenjuje i za regresivno ispitivanje, to jest ispitivanje da li izmene unesene tokom daljeg razvoja i održavanja, kvare dosadašnju funkcionalnost DI.

Literatura

[1] V. Kovačević, M. Popović, *Sistemska programska podrška u realnom vremenu*, Fakultet tehničkih nauka, Novi Sad, 2002.

[2] John Gilmore, Stan Shebs, *GBD Internals – A guide to the internals of GNU debugger*, Cygnus Solutions, 2004, ch. 20.

[3] Rob Savoye, *DejaGnu – The GNU testing framework*, Free Software Foundation

[4] Marko Krnjetin, *Jedno rešenje simulatora digitalnog signal procesora za integrisano razvojno okruženje*, diplomski – master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2007.

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

Abstract

This paper describes a way of testing a debugger interface component. Firstly it shows the immportant aspects of the debugger it self and the component at question. Then it explains the automatic testing platform which is used and finally the specifics of the test cases.

ONE SOLUTION FOR AUTOMATIC TESTING OF DEBUGGING INTERFACE FOR DSPs

Branislav Rankov, Marko Krnjetin, Miodrag Đukić, Jelena Kovačević.