Elektrotehnički fakultet
Univerzitet u Beogradu

HARDVERSKI MODUL ZA OPTIMIZACIJU POTROŠNJE DISTRIBUIRANOG NAMENSKOG SISTEMA

Projekat: **Razvoj i modelovanje energetski efikasnih, adaptibilnih, višeprocesorskih i višesenzorskih elektronskih sistema male snage**

Oznaka projekta: **TR32043**
Rukovodilac projekta: **Goran Dimić**

Vrsta dokumenta: **Tehnička dokumentacija projekta**
Stepen poverljivosti: **poverljivo – interno**

Odgovorno lice:

Dragomir El Mezeni
Elektrotehnički fakultet Univerziteta u Beogradu
Tel:
E-mail: elmezeni@el.etf.rs

Realizator:

Dragomir El Mezeni, Ivan Popović, Strahinja Janković, Lazar Saranovac

1 KRATAK OPIS TEHNIČKOG REŠENJA

Primena od (dd.mm.gggg): 30.11.2014. godine
Godina* : 2014
Odgovorno lice : Dragomir El Mezeni

Opis: Modul za optimizaciju potrošnje namenskog distribuiranog sistema omogućava praćenje opterećenja procesora tokom rada i komuniciranje statističkih parametara opterećenja ostalim procesorima u sistemu čime se obezbeđuje podrška za implementaciju algoritama za optimizaciju potrošnje i performansi sistema. Komunikacioni interfejs omogućava fleksibilno povezivanje optimizacionih modula u jedinstvenu heterogenu mrežu i razmenjivanje statistika sa minimalnim opterećivanjem procesora. Kontrola merenja opterećenja (definisanje granica programskih poslova od interesa) se obavlja iz aplikativnog sloja što obezbeđuje maksimalnu fleksibilnost i promenu granica programskog posla od interesa tokom rada sistema. Arhitektura sistema omogućava hijerarhijsku optimizaciju gde se sve statistike procesora koji pripadaju određenoj optimizacionoj grupi centralizovano obrađuju i donose se odluke visokog nivoa kako treba menjati opterećenje svakog od procesora, a procesori sami na osnovu dostupnih mehanizama određuju način kako postići zadate ciljeve. Arhitektura optimizacionog modula je identična za sve procesore a master/slave konfiguracija se obavlja iz aplikacionog sloja što omogućava jednostavnu rekonfiguraciju sistema tokom rada. Prikazani optimizacioni modul je implementiran na Cyclone II FPGA platformi sa Nios II procesorom i OS-II operativnim sistemom.

Karakteristike rešenja:

Tip rešenja:	Modul za optimizaciju potrošnje
Platforma:	Višeprocesorski sistem
Operativni sistem:	Sa ili bez operativnog sistema u realnom vremenu

Hardverska zahtevnost: FPGA platforma sa Avalon procesorskom magistralom

Mogućnosti rešenja: Tehničko rešenje obezbeđuje:

a) Podrška za hijerahijsku optimizaciju višeprocesorkog sistema
b) Praćenje statistika opterećenja svakog procesora i komunikaciju kao ostalim procesorima sa minimalnim učešćem procesora
c) Rekonfiguracija granica programskog posla od interesa iz aplikativnog sloja
d) Rekonfiguracija master/slave optimizacionih modula iz aplikativnog sloja

Realizatori : Dragomir El Mezeni, Ivan Popović, Lazar Saranovac, Strahinja Janković

Korisnici : Laboratorija za Integrisane računarske sisteme na Elektrotehničkom fakultetu u Beogradu
Laboratorija za telekomunikacije
Institut Mihajlo Pupin

Podtip rešenja : Prototip (M85)

SADRŽAJ:

2 STANJE U OBLASTI

Povećanje performansi procesora iz generacije u generaciju je do skoro realizovano povećanjem radne učestanosti. Međutim smanjenjem veličine tranzistora i povećanjem radne učestanosti gustina disipacije se značajno povećava i nemogućnost efikasnog odvođenja toplote ograničava dalje povećanje radne učestanosti. Upravo iz ovog razloga u poslednje vreme su u prvi plan izbili višeprocesorski sistemi kao odgovor na pomenuta tehnološka ograničenja. Pored toga ekspanzija mobilnih baterijski napajanih sistema nametnula je potrebu za efikasnim načinima kontrole i optimizacije potrošnje kako bi se vreme autonomnog rada što više produžilo. Distribuirane mreže pametnih senzora, umrežavanje raznorodnih uređaja u jedinstveni informacioni sistem (*Internet of Things*) predstavljaju primere novih sistema koji se pojavljuju u poslednje vreme.

Većina algoritama za optimizaciju potrošnje višeprocesorskih sistema zasniva se na statičkoj analizi rada sistema i optimizaciju tako da u najgorem slučaju budu ispunjena vremenska ograničenja kritičnih poslova. U sistemima u kojim vreme izvršavanja i trenuci započinjanja programskih proslova variraju dosta u toku vremena statička optimizacija sistema u odnosu na najgori slučaj dovodi do suboptimalne potrošnje tokom većine vremena (kada opterećenje sistema nije maksimalno). Ovo je posebno izraženo u distribuiranim senzorskim mrežama gde opterećenje procesora zavisi od merenih fizičkih veličina. U ovakvim sistemima statička analiza daje prilično loše rezultate i javlja se potreba za dinamičkom optimizacijom potrošnje koja se prilagođava trenutnoj opterećenosti sistema.

Postoji dosta radova u kojima se predlaže arhitektura modula za globalnu optimizaciju potrošnje. Statistički parametri opterećenja i potrošnje se periodično šalju modulu za optimizaciju potrošnje koji na osnovu njih određuje naredno stanje sistema (učestanosti rada i napon napajanja procesora u sistemu). Kako bi se obezbedila fleksibilnost uvode se dva nivoa hijerarhije. Naime globalni kontroler određuje kako bi svaki od procesora u sistemu trebao da promeni svoju potrošnju i opterećenje dok lokalni kontroler koji se nalazi u svakom od procesora određuje način na koji će se postići zadati globalni cilj u zavisnosti od dostupnih mehanizama na datom procesoru. Zajedničko za sve postojeće metode je da se za merenje opterećenosti procesora koriste ugrađeni brojači koje nije moguće programski kontrolisati iz aplikativnog sloja što donekle ograničava fleksibilnost pri definisanju granica poslova od interesa za koje merimo opterećenje procesora. Takođe većina optimizacionih modula predstavlja softverska rešenja koja dodatno opterećuju procesor. Kako sa složenijim sistemima proces optimizacije potrošnje postaje sve složeniji javlja se potreba za što efikasnijom implementacijom optimizacionih modula. U literaturi je predložena arhitektura potpuno hardverskog modula za optimizaciju potrošnje koji ima superiorne karakteristike po pitanju performansi i potrošnje. Problem kod potpuno hardverskog rešenja je nedostatak fleksibilnosti u slučaju potrebe dodavanja novih elemenata u sistem ili redefinisanja samog algoritma optimizacije.

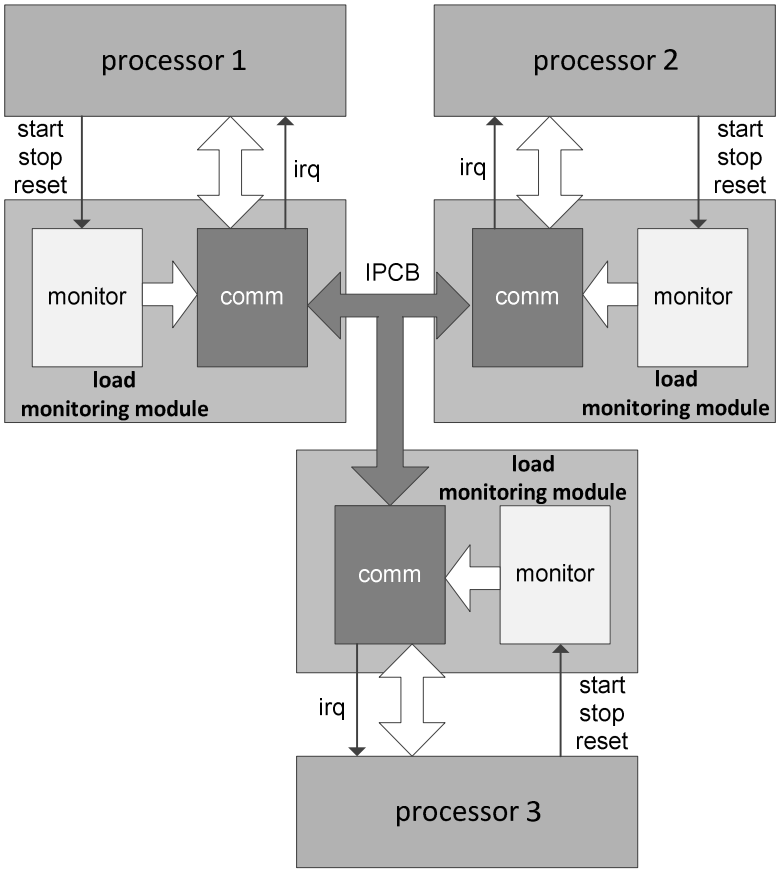
Kako bi se obezbedila maksimalna fleksibilnost i skalabilnost sistema uz minimalno opterećenje procesora predložena je hardverska arhitektura modula za optimizaciju potrošnje. Predloženi modul omogućava fleksibilno definisanje granica posla od interesa iz aplikativnog sloja što ostavlja mogućnost redefinisanja granica posla od interesa dinamički u toku rada sistema. Komunikacija optimizacionih modula omogućena je korišćenjem komunikacionog interfejsa koji omogućava jednostavno povezivanje različitih modula u jedinstvenu komunikacionu mrežu a koji je predstavljen u tehničkom rešenju „Interfejs za međuprocesorku komunikaciju na heterogenoj višeprocesorkoj platformi“ autora Ivana Popovića, Dragomira El Mezenija, Lazara Saranovca, Željka Ilića i Strahinje Jankovića. Prikazano rešenje omogućava realizaciju proizvoljne heterogene višeprocesorske platforme gde je topologija veza realizovana u vidu deljene magistrale čime se postiže maksimalna fleksibilnost i skalabilnost sistema. Hardverska realizacija pomenutog komunikacionog interfejsa je predstavljena u posebnom tehničkom rešenju „Hardverski modul za

medjuprocesorsku komunikaciju na heterogenoj viseprocesorskoj platformi“ autora Ivana Popovića, Dragomira El Mezenija, Lazara Saranovca i Strahinje Jankovića. Modifikovani komunikacioni modul omogućava razmenu statističkih parametara opterećenja procesora uz minimalnu interakciju samih procesora.

3 DETALJNO TEHNIČKO REŠENJE

3.1 Arhitektura modula za optimizaciju potrošnje

Arhitektura sistema sa omogućenom dinamičkom optimizacijom potrošnje podrazumeva da svaki procesor u sistemu koji učestvuje u optimizaciji poseduje po jedan pridružen modul za optimizaciju potrošnje. Svrha ovog modula je da prati statističke parametre opterećenja procesora i razmenjuje ove informacije sa ostalim optimizacionim modulima u sistemu. Svi optimizacioni moduli su povezani na jedinstvenu međuprocesorsku magistralu (IPCB) kao što je prikazano na Slici 3.1. Svaki optimizacioni modul se sastoji iz dve komponente: modula za praćenje opterećenja procesora (monitor modul) i modula odgovornog za komunikaciju parametara opterećenja ka drugim optimizacionom modulima (komunikacioni modul).



Slika 3.1. Sistem sa omogućenom dinamičkom optimizacijom potrošnje pomoću predloženih optimizacionih modula

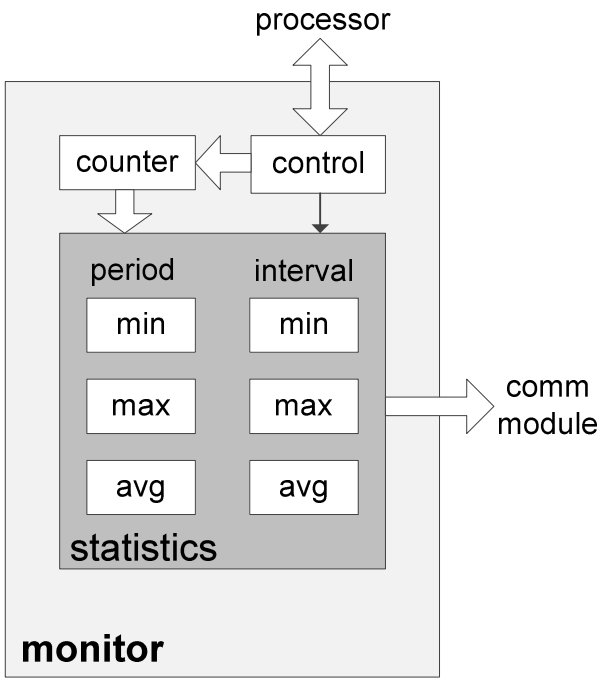
Modul za praćenje opterećenja procesora prati vreme izvršavanja i period poslova od interesa i osnove statističke parametre (maksimum, minimum, srednju vrednost) skladišti u internu banku

registara. Početak i kraj posla od interesa se zadaje iz aplikativnog sloja pozivom odgovarajućih start i stop rutina.

Komunikacioni modul obezbeđuje vezu sa ostalim modulima u sistemu. Komunikacioni moduli implementiraju poseban protokol za razmenu statističkih parametara opterećenja tako da je potrebna minimalna interakcija procesora. Kako bi se maksimalno iskoristila komunikaciona infrastruktura pored razmene statističkih parametara omogućena je i razmena proizvoljnih poruka između procesora. Obaveštavanje procesora o pristizanju nove poruke realizovano je putem prekida.

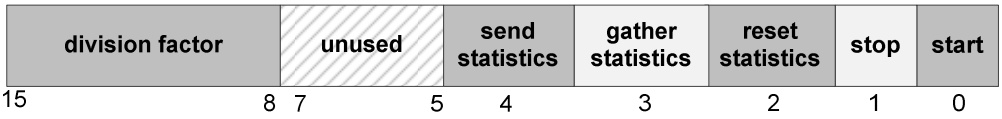
3.1.1 Modul za praćenje opterećenja procesora

Unutrašnja struktura modula za praćenje opterećenja procesora prikazana je na Slici 3.2.



Slika 3.2. Struktura modula za praćenje opterećenja procesora

Opterećenje procesora se najčešće predstavlja kao odnos vremena provedenog u izvršavanju određenog programskog posla i perioda izvršavanja tog programskog posla. Kako ova vremena mogu značajno varirati tokom vremena nije ih dovoljno opisati jednim brojem. Kako bi se što bolje uhvatila priroda varijacija vremena i perioda izvršavanja programskih poslova odabrano je da se prati minimalno, maksimalno i prosečno vreme i period izvršavanja. Merenje vremena se obavlja korišćenjem *free-running* brojača sa programabilnim deliteljem učestanosti. Na taj način je omogućen kompromis između preciznosti i opsega vrednosti vremenskih intervala koje je moguće meriti. U prikazanoj realizaciji delitelj učestanosti je jedinstven za merenje perioda i trajanja posla od interesa. U slučaju da je potrebno podržati programske poslove koji se retko pojavljuju (dugačak period) ali jako brzo izvršavaju (kratak interval izvršavanja) mogu se uvesti zasebni brojači i delitelji učestanosti za period i interval izvršavanja. Ovo sa druge strane komplikuje razmenu statistika pošto se sada pored parametara opterećenja moraju slati i delitelji za interval i period kako bi bilo moguće izračunati opterećenje procesora. U prikazanoj implementaciji korišćen je jedan jedinstveni delitelj učestanosti.



Slika 3.3. Kontrolni registar modula za praćenje opterećenja procesora

Opterećenje procesora se najčešće predstavlja kao odnos vremena provedenog u izvršavanju određenog programskog posla i perioda izvršavanja tog programskog posla. Kako ova vremena

Tabela 3.1 Struktura kontrolnog registra modula za praćenje opterećenja procesora.

BIT	NAZIV	OPIS
0	START	Označava početak izvršavanja posla od interesa.
1	STOP	Označava kraj izvršavanja posla od interesa.
2	RESET STATISTICS	Resetuje sve statističke parametre.
3	GATHER STATISTICS	Inicira slanje poruke sa zahtevom za statističkim parametrima od svih optimizacionih modula u sistemu.
4	SEND STATISTICS	Omogućava slanje statističkih parametara opterećenja po prijemu poruke sa zahtevom za slanje.
7-5	UNUSED	
15-8	DIVISION FACTOR	Delitelj takta brojača za merenje vremenskih intervala posla od interesa.

Modul za merenje opterećenja procesora se kontroliše postavljanjem odgovarajućih bita u kontrolnom registru koji se nalazi na adresi 8 i čija je interna struktura prikazana na Slici 3.3. Pozivom *start* rutine iz aplikativnog sloja postavlja se bit start u kontrolnom registru čime se označava početak izvršavanja posla od interesa i trenutno stanje brojača se čuva u posebnom registru. Vremenski interval između dva poziva *start* rutine predstavlja period izvršavanja programskog posla od interesa. Svakim pozivom *start* rutine dobija se nova trenutna vrednost perioda izvršavanja i na osnovu nje se osvežavaju vrednosti u registrima koji sadrže minimalnu, maksimalnu i srednju vrednost perioda izvršavanja programskog posla od interesa. Kraj izvršavanja posla od interesa se označava pozivom *stop* rutine iz aplikativnog sloja kojom se postavlja *stop* bit u kontrolnom registru monitor modula. Vremenski interval između poziva *start* i *stop* rutine (razlika zapamćenih stanja brojača) predstavlja vreme izvršavanja posla od interesa. Pri svakom pozivu *stop* rutine dobija se trenutna vrednost vremena izvršavanja i na osnovu nje se osvežavaju vrednosti u registrima koji sadrže minimalnu, maksimalnu i srednju vrednost vremena izvršavanja programskog posla od interesa. Srednje vrednosti se izračunavaju aproksimativno korišćenjem IIR filtra prvog reda što značajno pojednostavljuje hardversku implementaciju.

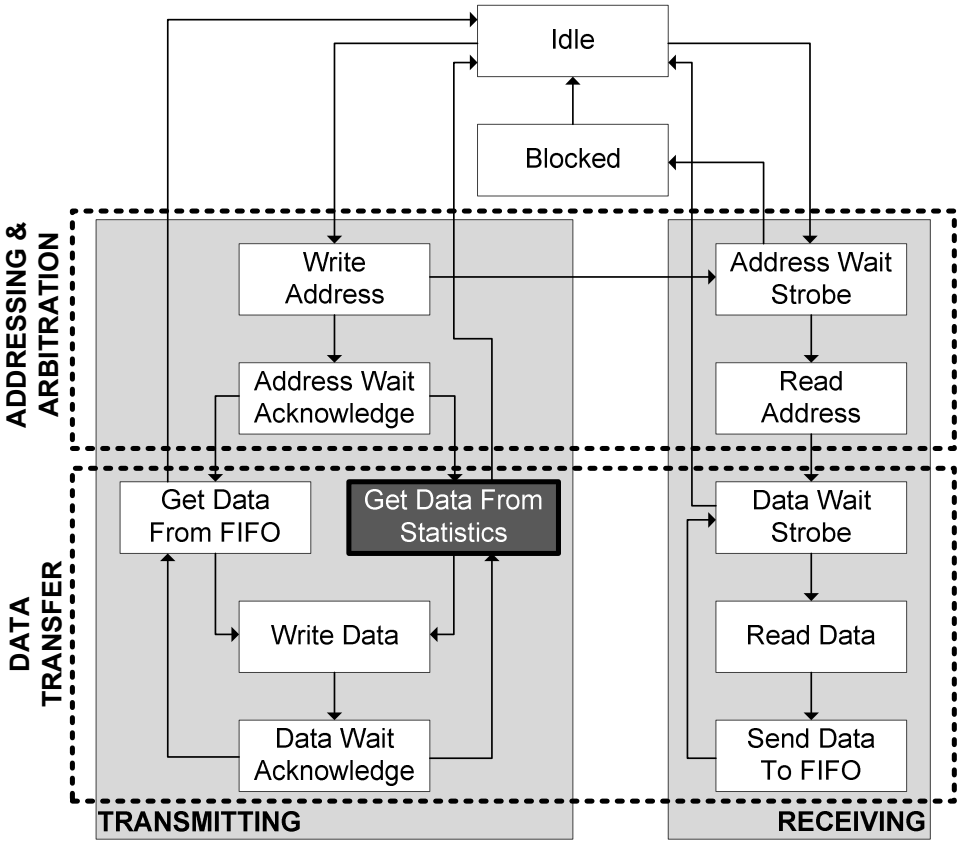
Svaki put kada se promeni neki od parametara koji suštinski menja prirodu vremenskih intervala (promena granica posla od interesa ili promena delitelja učestanosti) potrebno je resetovati sve statističke parametre postavljanjem *reset* bita u kontrolnom registru. Resetovanje statistika je takođe obavezno pri promeni parametara sistema (učestanost, način raspoređivanja poslova i sl.).

Arhitektura sistema sa omogućenom optimizacijom podrazumeva postojanje *master* modula koji prikuplja statističke parametre od *slave* modula izvršava optimizacioni algoritam čiji izlaz predstavlja novo ciljano stanje svakog od *slave* modula. Kako je arhitektura optimizacionog modula identična za sve procesore u sistemu definisanje *slave* i *master* modula se obavlja iz aplikativnog sloja i može se menjati tokom rada sistema. Svaki procesor koji je odgovoran za optimizaciju određene grupe procesora postavljanjem *gather statistics* bita zahteva statistike od svih slave modula u sistemu. Na osnovu adrese pristiglih poruka *master* procesor filtrira samo one poruke koje pripadaju njegovoj optimizacionoj grupi i pokreće optimizacioni algoritam. Svi procesori koji imaju

ulogu *slave* procesora u nekoj optimizacionoj grupi prostavljanjem *send statistics* bita omogućavaju slanje statistika na zahtev nekog od mastera u sistemu. Na ovaj način je obezbeđeno da isti procesor bude *master* u jednoj a *slave* u drugoj optimizacionoj grupi čime se omogućava slojevita optimizacija sistema. Kako se uloge *master* i *slave* procesora jednostavno određuju iz aplikativnog sloja postavljanjem odgovarajućih bita u kontrolnom registru moguće je njihovo redefinisanje tokom rada sistema.

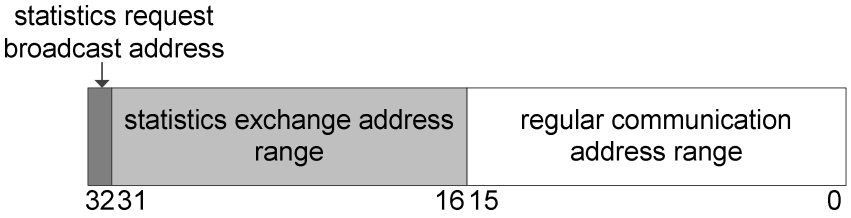
3.1.2 Komunikacioni modul

Svi optimizacioni moduli u sistemu su povezani na jedinstvenu međuprocesorsku magistralu. Protokol međuprocesorske komunikacije omogućava multi master mod rada sa ugrađenom arbitracijom i prioritetizacijom poruka. Komunikaciona magistrala i interfejs omogućavaju jednostavno kreiranje heterogenog sistema uz mogućnost jednostavnog dodavanja novih komponenti u sistem. Komunikacioni protokol i njegova hardverska implementacija je predstavljen u tehničkom rešenju „Hardverski modul za međuprocesorsku komunikaciju na heterogenoj višeprocesorskoj platformi“ autora Ivana Popovića, Dragomira El Mezenija, Lazara Saranovca i Strahinje Jankovića je uz odgovarajuće modifikacije korišćen u okviru realizovanog optimizacionog modula. Iako sam koncept komunikacionog modula podržava proizvoljnu širinu komunikacione magistrale mi smo se ograničili na 32 bita za magistralu za podatke i 3 kontrolna signala. Implementiran je samo jedan nivo adresiranja i arbitracije tako da je na raspolaganju ukupno 33 različitih adresa. Kako bi se obezbedila automatska razmena statistika dodato je novo stanje u mašinu stanja komunikacionog modula (*Get Data From Statistics*) što je prikazano na Slici 3.4. Ovo stanje je aktivno samo kada je u toku razmena statistika i omogućava slanje podataka direktno iz registara sa statistikama monitor modula.



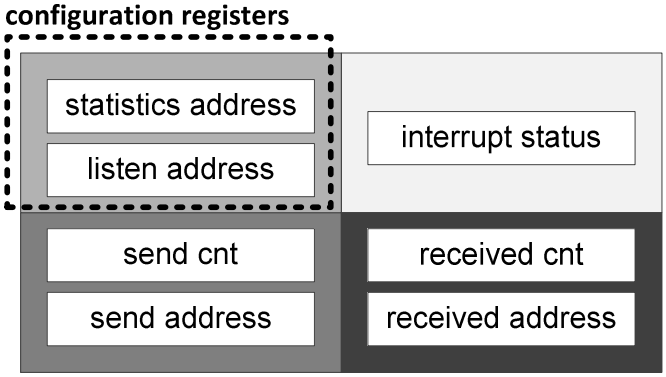
Slika 3.4. Mašina stanja komunikacionog modula sa dodatnim stanjem za razmenu statistika

Dostupne adrese su podeljene u 3 opsega kao što je prikazano na Slici 3.5. Regularna komunikacija (razmena poruka) se obavlja korišćenjem nižih 16 adresa (adrese najvišeg prioriteta). Ovim se garantuje da brza razmena poruka između procesora dok se razmena statistika obavlja kao komunikacija nižeg prioriteta odnosno kao pozadinski proces. Najviša adresa predstavlja *boradcast* adresu koju šalje *master* modul označavajući početak novog optimizacionog ciklusa i zahtevajući statističke parametre od *slave* modula.



Slika 3.5. Opseg adresa komunikacionog modula

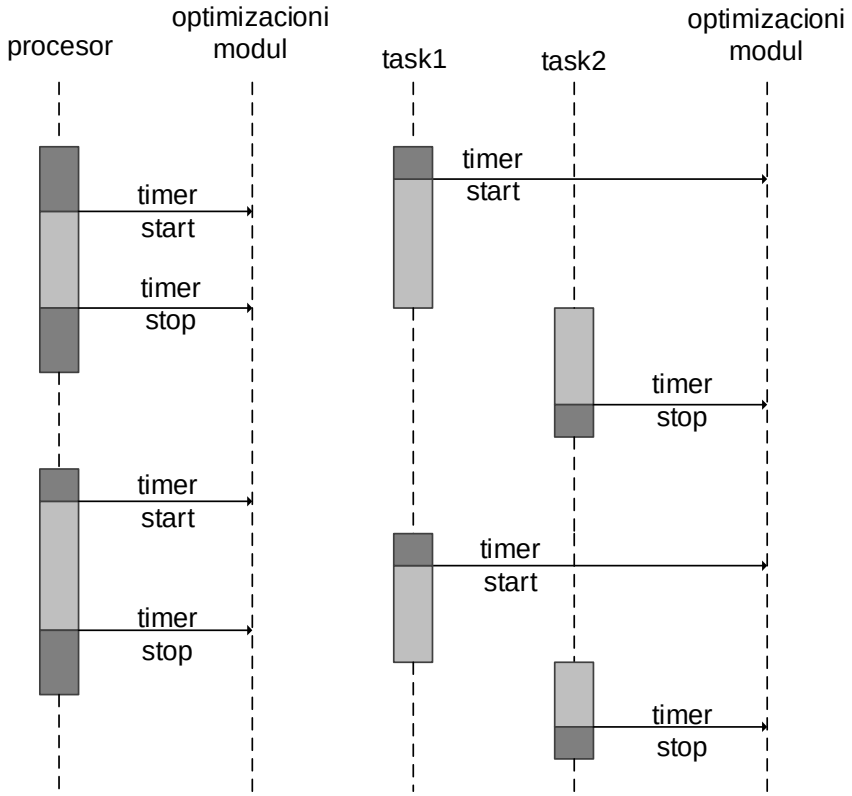
Komunikacioni modul poseduje 7 registara kojima se može pristupiti iz aplikativnog sloja prikazanih na Slici 3.6. Svaki komunikacioni modul ima jedinstvenu adresu za slanje statistika konfigurisanu u *statistics address* registru. Na osnovu ove adrese *master* procesor identifikuje od kog *slave* procesora stiži statistike i koristi istu adresu za slanje novog ciljanog stanja. Takođe svaki procesor u okviru *listen address* registra navodi sa kojih adresa prima poruke pri čemu svaki bit u ovom registru odgovara jednoj adresi (ako je bit N postavljen na 1 znači da komunikacioni modul osluškuje poruke sa N-te adrese u suprotnom ignoriše poruke sa te adrese). Najviša adresa 32 predstavja *broadcast* adresu sa zahtevom za slanje statistika i nju podrazumevano osluškuju svi *slave* optimizacioni moduli. Komunikacioni modul poseduje samo jednu prekidnu liniju kojom je povezan sa procesorom. Koji prekid se trenutno desio se identifikuje očitavanjem *interrupt status* registra. Postoje 3 vrste prekida koji dolaze od komunikacionog modula: pristigla nova poruka, poruka uspešno poslata i greška pri prenosu. Po prispeću nove poruke aktivira se prekid sa odgovarajućim sadržajem statusnog registra. Broj pristiglih podataka se nalazi u *received cnt* registru dok je adresa po kojoj je pristigla poruka upisana u *received address* registar. Sama poruka se nalazi u FIFO baferu i procesor je dužan da u prekidnoj rutini pročita sadržaj pristigle poruke. Po pristizanju poruke sa *broadcast* adresom ne aktivira se prekid. U slučaju da je *send statistics* bit setovan znači da optimizacioni modul predstavlja *slave* modul u nekoj optimizacionoj grupi pa je samim tim dužan da pošalje statistike. U tom slučaju se po prijemu poruke sa zahtevom za slanje statistika automatski inicira proces slanja bez nepotrebnog prekidanja procesora.



Slika 3.6. Registri komunikacionog modula

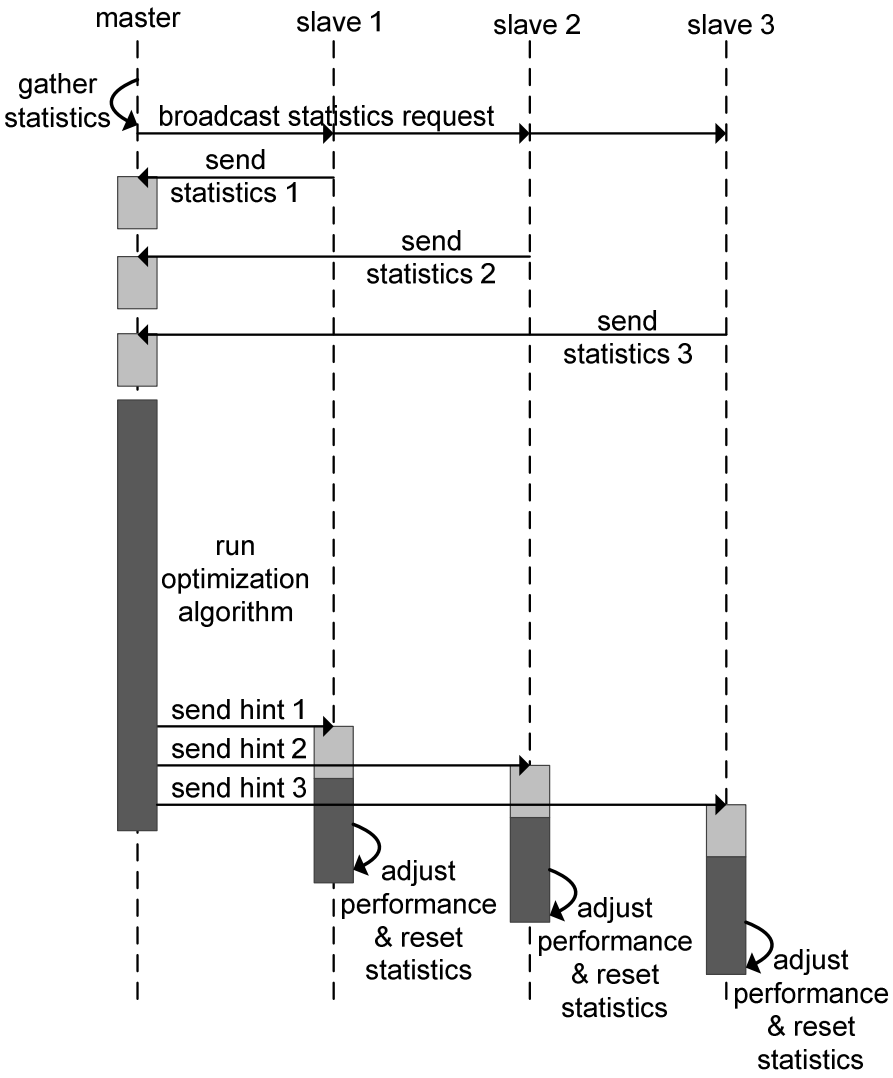
3.2 Protokol optimizacije višeprocesorskog sistema

Optimizacija višeprocesorskog sistema je omogućena korišćenjem niza funkcija dostupnih na aplikativnom sloju. Svaki procesor koji učestvuje u optimizaciji pri inicijalizaciji sistema treba da pozove *enable_statistics* funkciju sa argumentom adrese za razmenu statistika kojom se inicijalizuje *statistics_address* registar i setuje *send_statistics* bit u kontrolnom registru monitor modula. Ovom funkcijom se takođe automatski postavlja odgovarajući bit u *listen address* registru čime se omogućava prijem poruka od master modula. Merenje vremena trajanja i perioda izvršavanja posla od interesa se kontroliše pozivom *timer_start* i *timer_stop* funkcija na početku i na kraju izvršavanja posla od interesa kao što je prikazano na Slici 3.7. Posao od interesa se može sastojati iz više različitih taskova na Slici 3.7. je prikazan i slučaj kada izvršavanje posla od interesa započinje u jednom a završava se u drugom tasku. Mesta pozivanja ovih funkcija se može menjati i u toku rada sistema čime je obezbeđena maksimalna fleksibilnost.



Slika 3.7. Određivanje granica posla od interesa

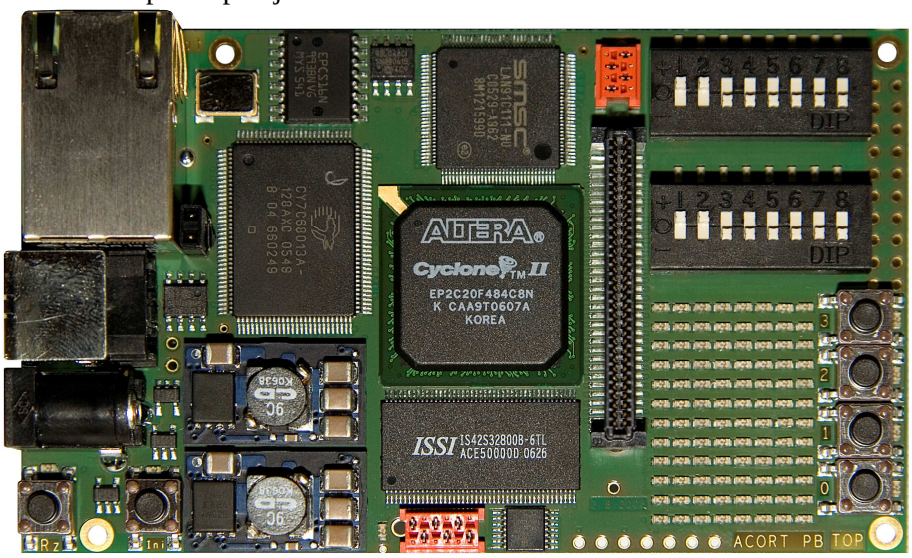
Optimizacioni protokol je prikazan na Slici 3.8. Optimizacioni ciklus jedne optimizacione grupe započinje tako što master modul pozivom *gather_statistics* funkcije inicira slanje *broadcast* poruke kojom se zahtevaju statistički parametri opterećenja od *slave* procesora. Po prijemu ove poruke svi *slave* moduli šalju poruke sa parametrima opterećenja procesora. *Master* modul po prijemu parametara opterećenja od svih *slave* modula u sistemu izvršava optimizacioni algoritam kojim se određuju novi ciljani parametri sistema. Novi parametri se šalju svakom od *slave* procesora. Po prijemu novih parametara svaki od procesora podešava svoje stanje shodno dostupnim mehanizmima optimizacije potrošnje i resetuje sve statističke parametre sistema.



Slika 3.8. Optimizacioni protokol

4 VERIFIKACIJA

Rešenje je verifikovano na razvojnoj platformi FPGA4U, prikazanoj na slici 4.1. koja na sebi ima Cyclone II FPGA čip kompanije Altera.

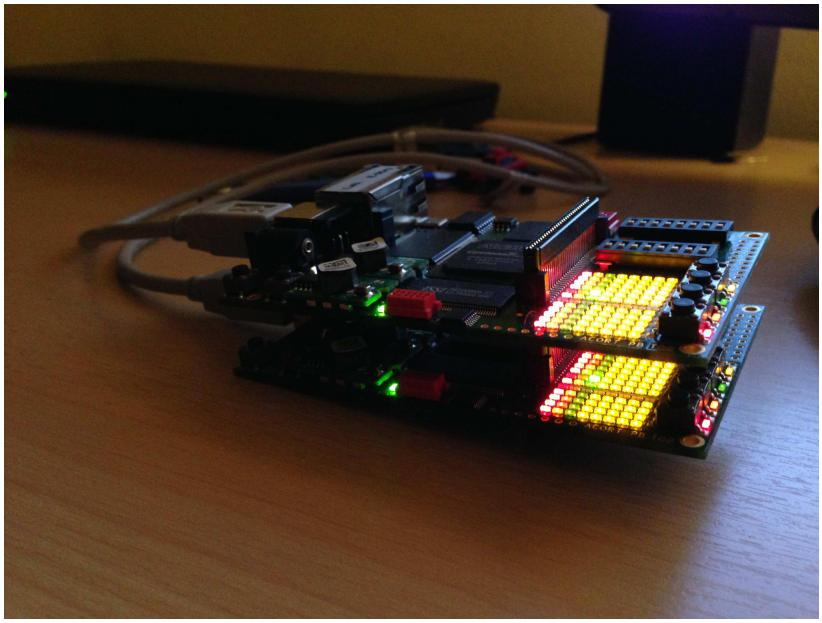


Slika 4.1. Razvojna platforma FPGA4U

Realizovan je kompletan procesorski sistem u FPGA čipu sa Nios II procesorom, PLL modulom za skaliranje osnovnog takta, FIFO baferima, UART modulom za komunikaciju sa računarom i tajmerom koji je korišćen kao sistemski takt operativni sistem. Instrukcije i podaci su smešteni u eksterni SDRAM modul koji se taktuje da 50MHz. U programskom paketu Nios II Software Build Tools razvijen je softver baziran na µC/OS-II operativnom sistemu. Isti hardver je spušten na dve razvojne platforme sa razlikom polažaja mapiranja linija komunikacione magistrale. Na jednoj platformi linije magistrale su mapirane na gornji ekspanzioni konektor dok su na drugoj mapirane na odgovarajuće pinove donjeg ekspanzionog konektora tako da se spajanjem dve pločice na način prikaza na slici 4.2. dobijaju dva procesorska sistema povezana međusobno komunikacionom magistralom. Svi izlazi komunikacione magistrale su trostatički i na njima su uključeni *weak-pullup* otpornici.

Verifikovano je:

1) Razmena statistika i dinamička rekonfiguracija
   - Oba sistema su bila konfigurisana tako da rade na učestanosti od 50 MHz. Procesori su bili konfigurisani tako da oba procesora mogu da primaju i šalju statistike. U toku rada u različitim vremenskim intervalima su zahtevali slanje parametara. Taskovi su bili podešeni tako da je jedan procesor bio značajno više opterećen od drugog što se i oslikalo u poslatim statističkim parametrima. Testiran je i prijem poruka po adresi za razmenu statistika od mastera ka slave procesoru.



Slika 4.2. Povezivanje razvojnih platformi u multiprocesorski sistem

5 ZAKLJUČAK I DALJI RAZVOJ

Predloženi optimizacioni modul predstavlja podršku za efikasnu implementaciju optimizacionih algoritama za kontrolu performansi i potrošnje distribuiranog namenskog sistema. Modul omogućava maksimalnu fleksibilnost pri definisanju granica posla od interesa za koji se meri opterećenje s obzirom da se sva kontrola obavlja iz aplikativnog sloja. Komunikacioni interfejs omogućava efikasnu razmenu statistika bez mnogo interakcije procesora. Optimizacioni protokol podrazumeva podelu procesora u optimizacione grupe u okviru koji postoji *master* procesor zadužen za prikupljanje statisitka i izvršavanje optimizacionog algoritma. Izmeštanjem optimizacionog algoritma u softver omogućava se njegova jednostavna promena.

Nastavak rada podrazumeva razvoj optimizacionih algoritama i testiranje njihove efikasnosti u optimizaciji realinih distribuiranih sistema korišćenjem predložene hardverske arhitekture optimizacionog modula.