SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 444

PROCESORI ZASNOVANI NA FPGA SKLOPOVIMA

Dario Zurovec

Zagreb, lipanj 2012.

Zahvala

Zahvaljujem se mentoru doc. dr. sc. Tomislavu Pribaniću na ukazanom povjerenju i slobodi koju mi je dao tijekom pisanja ovog rada.

Zahvaljujem se mentorici dipl. ing. Maji Vlah iz društva Končar – Institut za elektrotehniku na zaista velikom strpljenju,mnogobrojnim savjetima te velikoj količini vremena, živaca i pomoći koje mi je maksimalno posvetila uz svoj posao.

Zahvaljujem se svima onima koji su mi davali podršku kroz sve godine mog studiranja i bez kojih realizacija ovog rada ne bi bila moguća – roditeljima na poticanju na rad kad sam imao najmanje volje, mnogobrojnim prijateljima na zbijanju šala i podizanju morala dok sam padao na ispitima, te pogotovo bratu na financijskoj podršci i šogorici na kolačima koji su mi služili kao izvrstan doping u mnogim neprospavanim noćima posvećenim fakultetu i učenju.

Popis slika

Popis tablica

SADRŽAJ

Uvod

Procesor (u stvari kratak oblik za riječ mikroprocesor, često nazvan CPU ili centralna procesorska jedinica) predstavlja središnji sastavni dio računalnog sustava i vitalnu komponentu koja je odgovorna za sve što sustav radi. Također, određuje i koliko će cijeli sustav koštati: što je procesor noviji i moćniji, sustav je skuplji.

Unutar ovog rada bit će opisana procesorska rješenja implementabilna u FPGA sklopovima, njihova svojstva te analiza AVR jezgre s poboljšanom protočnom strukturom - pAVR. Ovaj rad napravljen je u suradnji Končar institutom za elektrotehniku – odjelom za Ugradbene računalne sustave.

U prvom poglavlju će biti detaljnije opisani FPGA sustavi, zbog čega se koriste, opća arhitektura te standardni pojmovi.

U drugom će biti detaljniji opis ugradbenih procesora u FPGA, objašnjenje osnovnih pojmova poput RISC, CISC arhitekture, licenci, IP jezgri, razlika između soft i hard jezgri, detaljniji opis AVR procesora te navedene i opisane određene inačice, prednosti i nedostataci te specifikacije pAVR procesora kao najprikladnije rješenje.

Treće poglavlje će sadržavati detaljan opis svih shema, neposrednu implementaciju i testiranje pAVR jezgre, pisane u VHDL-u, testirane u Xilinx webISE alatu, opis i analizu performansi.

Unutar četvrtog poglavlja bit će navedene standardne optimizacijske tehnike vezane uz softver i hardver, uz opise izvedbi memorija te njihovih optimizacija. Rad završava zaključkom, popisom literature te sažetkom na hrvatskom i engleskom jeziku.

1. FPGA sklopovi

1.1 Što je to FPGA?

FPGA (engl. ***F**ield **p**rogrammable **g**ate **a**rray*) je integrirani sklop dizajniran da bi se konfigurirao od strane kupca ili dizajnera.

Sadrži velik broj programirljivih logičkih vratiju i velik broj povezanih dijelova, čime se omogućava implementacija složenih digitalnih krugova.

FPGA konfiguracija definira se programskim jezikom za hardverski opis HDL (engl. *hardware description language*). Taj programski jezik sličan je onome koji se koristi za aplikacije specifičnih integriranih sklopova ASIC (engl. *Application specific integrated circuit*).

Da bi definirao ponašanje FPGA korisnik koristi jezik za opis hardvera – HDL ili shematski dizajn. Pomoću HDL-a moguć je lakši rad s velikim strukturama jer ih je moguće samo navesti brojčano, nema potrebe da se svaki komad crta rukom. Dok kod shematskog dizajna prednost je u tome što možemo slagati sheme i tako dobiti lakšu vizualizaciju dizajna. Jednom kada se napravi dizajn i provjera valjanosti proces je kompletan, binarna datoteka generirana. Datoteke se zatim prenose na FPGA preko različitih protokola, također se datoteke mogu prenijeti na vanjsku memoriju uređaja poput EEPROM.

Najčešći HDL programski jezici su VHDL i Verilog, pošto su ti programski jezici prilično složeni njihova složenost se pokušava smanjiti uvođenjem alternativnih jezika osnosno funkcija. Kako bi se pojednostavio dizajn složenih sustava u FPGA-ovima, postoje biblioteke predefiniranih složenih funkcija i sklopova koji su testirani i optimizirani za ubrzanje procesa dizajna. Ti krugovi koji su unaprijed definirani obično se nazivaju IP jezgre i mogu se dobiti od proizvođača FPGA.

Standardne primjene su u logici za povezivanje (eng. *glue logic*), brzoj izradi prototipova (eng. *fast prototyping*), proizvodnji u malim i srednjim serijama, aplikacijama koje traže rekonfiguriranje sklopovlja u toku eksploatacije, paralelizaciji algoritama, brzoj obradi velike količine informacija (multimedije itd.).
Cijene se kreću od nekoliko dolara do nekoliko stotina dolara po komadu, a proizvođači su Xilinx, Altera, Actel, Lattice, QuickLogic i Atmel.

1.2 Arhitektura FPGA

Iako svaki proizvođač FPGA koristi vlastitu specifičnu FGPA arhitekturu, sve se u osnovi baziraju na slijedećoj arhitekturi - logički blokovi, raspoređeni u dvodimenziono polje; U/I blokovi, raspoređeni po obodu čipa i programabilna sprežna mreža smještena u kanalima između logičkih blokova (detaljnija analiza dana je kroz naredna poglavlja).

Logički dijelovi služe za realizaciju logičkih funkcija, mreža omogućava povezivanje logičkih dijelova u cilju kreiranja složenijih funkcija, dok se putem U/I blokova ostvaruje povezanost internih resursa s pinovima čipa. Dodatno, FPGA može sadržavati i različite specijalizirane logičke resurse, kao što su ALU jedince, RAM memorija, dekoderi i dr.

Za veće brzine, neke FPGA arhitekture koriste duže usmjeravanje linije koje premošćuje višestruka logika blokova.

Prednost FPGA je u tome što posjeduje sposobnost nadogradnje funkcionalnosti nakon što se jednom isprogramira, tj. možemo raditi izmjene programskog koda ako želimo u bilo kojem trenutku nakon upotrebe FPGA čipa. Prednosti FPGA tehnologije također uključuje kraće vrijeme na tržištu, te niže inženjerske troškove prilikom izrade FPGA.

1.3 FPGA implementacijske tehnologije

Konfiguracija FPGA se može implementirati pomoću različitih tehnologija, a većina FPGA-ova koristi SRAM (eng. *Static RAM*). SRAM bazirani FPGA pohranjuje konfiguracijske podatke logičkih ćelija u statičnu memoriju (organiziranu kao polje bistabila). S obzirom da je SRAM izbrisiva memorija i ne može čuvati konfiguraciju bez izvora napajanja, takav FPGA se mora isprogramirati (konfigurirati) na početku korištenja. Dva su načina programiranja, master mode, kada FPGA čita konfiguracijske podatke iz vanjskog izvora (npr. vanjske flash memorije) i slave mode, kada se FPGA konfigurira preko vanjskog *master* uređaja, poput procesora. Uobičajeni način ostvarivanja drugog načina se ostvaruje pomoću konfiguracijskog sučelja kao što je JTAG sučelje. SRAM bazirani FPGA uključuju velik broj chip-ova Xilinx-ove Virtex i Spartan familije i Altera-inih Stratix i Cyclone.

Postoji i SRAM bazirani FPGA koji je se od prethodno navedenog razlikuje po tome što sadrži unutarnje flash blokove, čime se eliminira potreba za vanjskom izbrisivom memorijom. Primjeri takve konfiguracije su Xilinx Spartan-3AN familija te LatticeXP familija proizvođača Lattice Semiconductors.

Flash bazirani FPGA ne treba miješati s prethodnim tipom. SRAM bazirani FPGA s unutarnjom flash memorijom koristi flash samo tijekom pokretanja da bi pohranio podatke u SRAM konfiguracijske ćelije. Flash bazirani FPGA koristi flash kao primrani izvor za pohranu konfiguracije i ne zahtjeva SRAM. Prednost te tehnologije je u tome što troši manje energije. Primjeri flash baziranih FPGA familija su Igloo i ProASIC3 proizvođača Actel.

Antifuse-bazirani FPGA se mogu programirati samo jednom. Antifuse je uređaj koji ne provodi struju „inicijalno“ nego se mora „zapaliti“. Baš zbog tog svojstva se ne mogu reprogramirati s obzirom da ne postoji način da se vrati u inicijalno stanje. Antifuse bazirane familije uključuju Axcelerator proizvođača Actel.

2. Ugrađivanje procesora unutar FPGA

Ugrađivanje procesora unutar FPGA ima mnoge prednosti, pogotovo vezanih uz specifične periferije koje se onda lako povezuju na FPGA, poput memorijskih kontrolera. FPGA ugradbeni procesori koriste opće-namjensku FPGA logiku za izgradnju interne memorije, sabirnica, internih periferija i vanjskih perifernih kontrolera (uključujući vanjske memorijske kontrolere).

"Meki" procesori (engl. *soft processors*) su također građeni po opće-namjenskom modelu FPGA. Kako se sve više dijelova dodaje procesoru (poput sabirnica, memorije, memorijskih kontrolera, periferija, periferijskih kontrolera), sustav postaje sve moćniji i korisniji. Međutim, ti dodaci smanjuju performanse i povećavaju cijenu ugradbenog sustava, usput crpeći resurse FPGA. Isto tako, veće vanjske memorijske banke se mogu povezati na FPGA i procesor im pristupa koristeći memorijski kontroler. Nažalost, latencija može imati značajan i negativan utjecaj na performanse.

FPGA proizvođači često objavljuju nova mjerila, standarde i namjene vezane za performanse ugradbenih procesora. Kao i sve druge tvrtke, namjera im je konstruirati i koristiti FPGA ugradbeni procesor koji se najbolje ponaša za neku određenu namjenu. To znači da takav jedan sustav ima malo periferija i pogoni se isključivo koristeći internu memoriju. Dizajner takvog sustava mora biti vrlo dobro upoznat kako će određene periferije i memorijske arhitekture utjecati na performanse. Međutim, ne postoji gotova formula ili graf za usporedbu performansi i cijene za različite memorije i skup periferija.

U daljnjim poglavljima će biti objašnjeni pojmovi poput RISC i CISC procesora, VLIW arhitekture, licenci te će biti prikazano nekoliko primjera koji ispituju učinke različitih ugradbenih procesora, tj. njihovih memorija i periferija te usporedba različitih sustava i tehnika za optimiranje performansi i cijene FPGA ugrađenog procesora.

2.1 Tipovi procesora

2.1.1 RISC

Koncept je razvio John Cocke u IBM istraživačkom laboratoriju tijekom 1974. Njegov argument je bio da računalo koristiti samo 20% instrukcija, što je činilo ostalih 80% suvišnima. Procesor baziran na tom konceptu bi koristio nekoliko instrukcija, što bi pak smanjilo broj tranzistora i pojeftinilo proizvodnju. Smanjujući broj tranzistora i instrukcija na samo one koje se najčešće koriste, računalo bi više radnji obavilo u kraćem vremenu. Termin RISC (skraćenica od eng. ***R**educed **I**nstrucion **S**et **C**omputer*) je kasnije izmislio David Patterson, profesor na Berkley-u.

RISC koncept je korišten za pojednostavljenje dizajna IBM PC/XT i kasnije se koristio u IBM RISC System/6000 i SUN Microsystem SPARC mikroprocesorima. Potonji CPU je doveo do osnivanja tvrtke MIPS Technologies (tehnologije), čija je direktna posljedica razvoj M.I.P.S. RISC mikroprocesora (***M**icroprocessor without **I**nterlocked **P**ipe **S**tages* - mikroprocesor bez isprepletenih dijelova protočne strukture). Mnogi MIPS arhitekti su odigrali značajne uloge u stvaranju Motorole 68000, korišteni u prvim Amiga računalima (MIPS tehnologije je kasnije kupila tvrtka Silicon Graphics). MIPS procesor se nastavio razvijati, postajući popularan izbor na tržištu ugradbenih i „low-end“ sustava.

2.1.2 CISC

CISC (skraćenica od eng. ***C**omplex **I**nstruction **S**et **C**omputer*) je retroaktivna definicija uvedena da bi se definirala razlika dizajna od RISC mikroprocesora.

Za razliku od RISC-a, CISC chip-ovi imaju velik broj različitih i složenih (eng. *complex*) instrukcija. Argument za nastavak korištenja te tehnologije je taj da bi dizajneri chip-ova mogli olakšati život programeru smanjujući broj instrukcija potrebnih za programiranje CPU-a. Zbog visoke cijene memorije i spremnika podataka CISC mikroprocesori su smatrani superiornima zbog zahtjeva za malim, brzim kodom. U doba smanjenja cijena memorije, HDD-ova itd., veličina koda je postalo pitanje bez veće težine (npr. MS Windows). Međutim, CISC bazirani

sustavi još uvijek pokrivaju većinu tržišta vis a vis desktop računala. Većina tih sustava je bazirano na x86 arhitekturi i varijacijama.

Detaljnija usporedba između CISC i RISC je dana u sljedećoj tablici:

CISC - veći skup instrukcija s mnogim načinima adresiranja	RISC - manji skup instrukcija s manje načina adresiranja
Koristi zasebne jedinice za mikro-programiranje s kontrolnom memorijom za implementaciju složenih instrukcija	Ima fiksno ožičenu programiranu jedinicu bez kontrolne memorije i zasebni hardware za implementaciju svake instrukcije
Jednostavno dizajniranje kompajlera	Složeno dizajniranje kompajlera
Kalkulacije su spore i precizne	Kalkulacije su brze i precizne
Dekodiranje instrukcija je složeno	Dekodiranje instrukcija je jednostavno
Vrijeme izvođenja je veliko	Vrijeme izvođenja je malo
Često zahtjeva pristup vanjskoj memoriji za obavljanje kalkulacija	S obzirom da koristi fiksno ožičeni model, često nema potrebe za korištenje vanjske memorije
Protočna struktura ne funkcionira zbog složenosti instrukcija	Protočna struktura ne predstavlja problem, dapače, opcija pipeline-a ubrzava procesor(e)
Često odugovlačenje izvođenja zbog problema nepostojanja protočne strukture	Instrukcije nisu složene, mogućnost pipeline-a, tako da se brzo izvode
Povećanje koda ne predstavlja problem za CISC procesore	Povećanje koda može predstavljati problem za neke RISC procesore
Troši više memorije	Troši manje memorije
Koriste se u Low end aplikacijama kao što su sigurnosni sustavi, automatizacija itd.	Koriste se u High end aplikacijama poput obrade video zapisa, telekomunikacijama, obradi slike itd.

2.1.3 VLIW

Pojam koji označava vrlo dugu instrukcijsku riječ (engl. ***v****ery* ***l****ong* ***i****nstruction* ***w****ord* – VLIW) i odnosi se na arhitekturu procesora dizajniranu u cilju iskorištavanja prednosti paralelizma instrukcija (engl. ***i****nstruction* ***l****evel* ***p****arallelism* – ILP). Dok konvencionalni procesori uglavnom podržavaju programe koji specificiraju instrukcije koje se izvode jedna za drugom, VLIW procesor dopušta programu da eksplicitno odredi instrukcije koje će se izvršavati u isto vrijeme, tj. paralelno. Ovaj tip arhitekture procesora je namijenjen da bi omogućio bolje performanse bez "standardne" kompleksnosti prisutnih kod ostalih pristupa.

Tradicionalni pristupi za poboljšavanje performansi uključuju razbijanje instrukcija u pod-korake tako da se instrukcije mogu izvoditi djelomično u isto vrijeme (protočna struktura), raspoređivanje individualnih instrukcija kako bi se izvodile neovisno u različitim dijelovima procesora (superskalabilne arhitekture) ili izvršavanje instrukcija u redoslijedu koji nije definiran programom (eng. *out-of-order execution*).

Takvi pristupi uključuju povećanu složenost sklopovlja (veća cijena, veći krugovi, veća potrošnja) zato što procesor mora sve naredbe izvršiti interno da bi pristupi funkcionirali.

VLIW pristup radi suprotno, ovisi o programima koji sami donose sve odluke koje se tiču instrukcija koje se izvršavaju simultano i kako se eventualno kršenje naredbi rješava. U praksi to znači da je prevodioc kompliciraniji, ali je hardware jednostavniji nego kod ostalih slučajeva.

Kao što je to slučaj sa svim arhitekturnim pristupima, koncept je jednostavan onoliko koliko kôd to omogućuje. Arhitektura koja je dizajnirana za korištenje u obradi signala može imati velik broj posebnih, tj. namjenskih instrukcija kako bi se olakšale komplicirane operacije poput brze Fourier-ove transformacije (FFT). Međutim, takve optimizacije su beskorisne ukoliko prevodioci nisu dovoljno dobri da prepoznaju značajni dio izvornog (eng. *source* – izraz često korišten u daljnjem tekstu) koda i generiraju ciljani kod koji bi iskoristio sve prednosti procesora. Stoga, programeri moraju biti sposobni napisati ili optimirati algoritme na način koji će omogućiti lakši rad prevodiocu.

2.2 Licence – GPL, LGPL, MIT, BSD

GPL predstavlja akronim za GNU opću javnu licencu (eng. *GNU GPL – general public license*, GNU GPL ili jednostavno GPL) i najčešće je korištena slobodna licenca za softver, izvorno napisana od strane Richarda Stallmana za GNU Projekt.

GPL je prva „copyleft“ licenca za opću namjenu, što znači da se izvedeni radovi pod tom licencom mogu dalje distribuirati pod uvjetima koje ta licenca propisuje. Prema tom načelu, GPL garantira prava slobodnog softvera primatelju računalnog programa i koristi se copyleft kako bi se osigurala „sloboda“ tog softvera, čak i prilikom mijenjanja samog softvera, za razliku od „dopustivih“ sofverskih licenci (npr. BSD licenca).

Tekst GPL licence ne spada pod GPL licencu, tj. korisnička prava licence onemogućuju modifikaciju licence. Kopiranje i distribucija licence je dopuštena s obzirom da GPL zahtjeva od primatelja da dobije „kopiju ove licence zajedno s programom“ (eng. *a copy of this License along with the Program*). Prema GPL FAQ-u (eng. *frequently asked questions* – najčešće postavljena pitanja), svatko može napraviti novu licencu koristeći modificiranu verziju GPL licence dokle god koristi drukčije ime za licencu, ne spominje „GNU“ i uklanja preambulu iako se preambula može koristiti u modificiranoj licenci ako se dobije odobrenje od Free Software Foundation (FSF) , tj. fondacije za besplatni softver.

GNU Lesser General Public License ili kraće LGPL je besplatna softverska licenca objavljena od strane FSF-a. Dizajnirana je kao kompromis između jake „copyleft“ GNU GPL-e ili GPL i „permissive“ licenci (kao što su BSD licence i MIT licence). LGPL stavlja copyleft restrikcije na program kojim se upravlja, ali se ne odnosi na restrikcije prema drugom softveru koji se povezuju s programom.

LGPL se primarno koristi za softverske biblioteke, iako se također koristi kroz neke samostalne aplikacije, poput Mozille i OpenOffice.org-a.

MIT licenca potječe s Massachussetts Institute of Technology – Instituta tehnologije iz Masačusetsa i predstavlja „permissive“ besplatnu licencu za softver, što znači da dopušta ponovno korištenje unutar proprietary softvera (proprietary – softver nad kojim postoji vlasništvo).

GPL je kompatibilna, što znači da GPL dopušta kombinaciju i redistribuciju sa softverom koji koristi MIT licencu. Softverski paketi koji koriste jednu od verzija

MIT Licence uključuju Expat, PuTTy, biblioteke klasa Mono razvojne platforme, Ruby on Rails, CakePHP, Lua (od verzije 5.0 nadalje) i X Window System, za koji je licenca originalno pisana.

BSD licenca spada u familiju permissive besplatnih softverskih licenci. Originalna licenca je korištena za **B**erkeley **S**oftware **D**istribution, operacijski sustav sličan Unix-u.

Dvije varijacije licence New/Modified BSD License i Simplified BSD License/FreeBSD licence su verificirane kao GPL kompatibilne od strane FSF-a te su prihvaćene kao open source licence od strane Open Source Inicijative, dok originalna licenca nije prihvaćena kao open source licenca te ju (iako se originalna verzija smatra kao slobodna softverska licenca od strane FSF-a) FSF ne smatra kao kompatibilnu s GPL s obzirom na klauzulu o oglašavanju.

S obzirom da je permissive slobodna softverska licenca, ima minimalna ograničenja oko toga kako se softver može redistribuirati, za razliku od copyleft licenci.

2.3 Hard naspram Soft jezgri

Postoje dvije vrste CPU jezgri za FPGA: “čvrste” (eng. *hard*) i “meke” (eng. *soft*). Hard jezgra je dio integriranog kruga, dok se mekana implementira koristeći FPGA logičke ćelije. Primjeri FPGA chip-ova s ugrađenim hard CPU jezgrama su Xilinx Virtex-4 FX i Virtex-5 FXT s PowerPC jezgrama, ARM922T unutar Altera Excalibur familije i PowerPC 405 unutar Xilinx Virtex-II Pro i Virtex-4 familije, Atmel FPSLIC s AVR jezgrom. Altera je koristila Excalibur familiju s ugrađenom hard ARMv4 jezgrom, al sada koristi Altera-in Nios-II soft CPU jezgru i Cortex-M1 ARMv6-baziranu CPU jezgru.

Tzv. “meki” procesor je izgrađen pomoću FPGA opće namjenske logike. Uobičajeno je opisan pomoću prethodno navedenih jezika za opis sklopovlja (HDL) ili pomoću „netliste“. Za razliku od *hard* procesora, meki procesori moraju biti sintetizirani i moraju odgovarati FPGA proizvodu.

U oba slučaja, lokalna (često korišten naziv - unutarnja) memorija, sabirnice, interne periferije, periferijski kontroleri i memorijski kontroleri moraju biti izgrađeni pomoću FPGA opće namjenske logike.

Da bi se olakšao dizajn FPGA ugradbenog procesora, proizvođači nude opsežne biblioteke u formi intelektualnog vlasništa (eng. *intelectual property* - IP) u obliku periferija i memorijskih kontrolera. Proizvođači uključuju taj IP, tj. biblioteke u alate koji dolaze u paketu s ugradbenim procesorom. Da bi se naglasila raznovrsnost i fleksibilnost koju nude, dana je djelomična lista IP-eva uključena kod ugradbenih procesora:

- periferije i periferijski kontroleri - ulazno izlazne jedinice opće namjene, UART, Timer, Debugger, SPI, DMA kontroler, Ethernet (sučelje prema vanjskom MAC/PHY chip-u).
- memorijski kontroleri - SRAM, Flash, SDRAM, DDR SDRAM (isključivo Xilinx), CompactFlash

Prednosti ugradbenih procesora u FPGA su prilagodba, umanjen utjecaj zastarjevanja komponenata, smanjenje broja komponenata i cijene, ubrzanje sklopovlja (eng. *hardware acceleration*). Svaka od navedenih prednosti bit će objašnjena u daljnjem tekstu.

2.3.1 Prilagodba

Dizajner ugradbenog procesorskog sustava za FPGA ima kompletnu fleksibilnost u izboru kombinacija periferija i kontrolera. U stvari, čak može i stvoriti nove, jedinstvene periferije koje se mogu direktno povezati na procesorsku sabirnicu. Ako dizajner ima nestandardne zahtjeve za skup periferija, nailazi na plodno tlo s ugradbenim procesorskim sustavom za FPGA. Primjerice, dizajneru nije baš lako naći "*off-the-shelf*" procesor s 10 UART-a, no u FPGA sustavu, takvu konfiguraciju je lako postići.

2.3.2 Zastarjevanje komponenata

Neke tvrtke, posebice one koje imaju ugovore s vojskom, dobivaju strože dizajnerske zahtjeve vezane uz životni vijek komponeneta s ciljem da osiguraju puno duže trajanje komponenata nego što je to slučaj sa standardnim elektroničkim proizvodima. Iako to nije lako izvedivo, FPGA soft procesori su odlično rješenje, s obzirom da se izvorni HDL model može prilagoditi toj primjeni. Vlasništvo nad HDL kodom može udovoljiti zahtjevima za garancijom dužeg životnog vijeka proizvoda.

2.3.3 Smanjenje broja komponenata i cijene

Uzevši u obzir svestranost FPGA, ranije proizvedeni sustavi koji zahtjevaju više komponenti se mogu nadomjestiti s jednim FPGA uređajem. Dakako, to je slučaj kad je potreban pomoćni U/I chip ili pomoćni procesor (u nastavku korišten naziv ko-procesor) uz *off-the-shelf* procesor. Smanjujući broj komponenata u dizajnu, tvrtka može smanjiti veličinu same ploče i manje trošiti na rezervne dijelove, a pritom uštedjeti vrijeme za razvoj i novac.

2.3.4 Ubrzanje sklopovlja

Vjerojatno najbitniji razlog za izbor ugradbenog procesora u FPGA je mogućnost kompromisa između razvoja hardvera i softvera kako bi se povećala učinkovitost i performanse.

Ako je algoritam identificiran kao softversko usko grlo, naš prilagođeni ko-procesor može biti prilagođen u FPGA uređaju da izvodi taj algoritam. Naš ko-

procesor može biti priključen na glavni FPGA procesor kroz zasebne kanale s niskom latencijom i kroz prilagođene instrukcije se može definirati suradnja s glavnim procesorom.

S modernim alatima za dizajn FPGA, izbjegavanje softverskih uska grla sa softvera na hardver je puno lakše s obzirom da se softver pisan u npr. C programskom jeziku može lako prilagoditi za hardver, s manjim izmjenama i dopunama izvornog koda.

2.3.5 Nedostaci

Za razliku od off-the-shelf procesora, hardverska platforma za FPGA ugradbeni procesor mora biti dizajnirana. Zbog integracije hardverske i softverske platforme, alati za dizajn su puno kompleksniji, što zahtjeva više znanja od samog dizajnera.

S obzirom da je softverski dizajn za FPGA ugradbene procesore relativno nov u odnosu na standardne procesore, alati za njihov razvoj su još relativno nerazvijeni. Najveći napredak u tom području su napravili Altera i Xilinx.

Cijena uređaja je isto jedan aspekt koji treba uzeti u obzir. Ako standardni off-the-shelf procesor može obaviti posao, taj će procesor biti jeftiniji nego FPGA s ekvivalentnim procesorskim dizajnom.

Međutim, ako je “veći” FPGA već u sistemu, iskorištavanje inače nekorištenih vratiju ili hard procesora u FPGA u suštini čini cijenu ugrađenog procesora neprimijetnom.

2.4 Ugradbene Soft jezgre prikladne za FPGA

Tablica 1. Meke jezgre za FPGA

CPU jezgra	Arhitektura	Bit-ova	Licenca	Dubina protočne strukture -a	Ciklusa po Instrukciji -1	MMU-2	MUL-3	FPU-4	broj LE-ova-5	Komentar
S1 jezgra	SPARC-v9	64	Open-source (GPL)	6	1	+	+	+	37000-60000	Jednojegrena verzija UltraSPARC T1
LEON3	SPARC-v8	32	Open-source (GPL)	7	1	+	+	+	3500	
LEON2	SPARC-v8	32	Open-source (LGPL)	5	1	+	+	prošireno	5000	Neodržavano u "usluzi" LEON3
OpenRISC 1200	OpenRISC 1000	32	Open-source (LGPL)	5	1	+	+	-	6000	
MicroBlaze	MicroBlaze	32	Proprietary	3, 5	1	opcionalno	opcionalno	opcionalno	1324	Ograničeno na Xilinx uređaje
aeMB	MicroBlaze	32	Open-source (LGPL)	3	1	-	opcionalno	-	2536	Open-source klonovi MicroBlaze-a
OpenFire	MicroBlaze	32	Open-source (MIT)	3	1	-	opcionalno	-	1928	Open-source klonovi MicroBlaze-a
Nios II/f	Nios II	32	Proprietary	6	1	+	+	opcionalno	1800	Ograničeno na Altera-ine uređaje
Nios II/s	Nios II	32	Proprietary	5	1	-	+	opcionalno	1170	Ograničeno na Altera-ine uređaje
Nios II/e	Nios II	32	Proprietary	ne	6	-	-	opcionalno	390	Ograničeno na Altera-ine uređaje

LatticeMico32	LatticeMico32	32	Open-source	6	1	-	opcionalno	-	1984	Nije ograničeno na Lattice uređaje (može biti upotrijebljeno drugdje)
Cortex-M1	ARMv6	32	Proprietary	3	1	-	+	-	2600	
DSPuva16	DSPuva16	16	Open-source	ne	4	-	+	-	510	
PicoBlaze	PicoBlaze	8	Proprietary, zero-cost	ne	2	-	-	-	192	Ograničeno na Xilinx uređaje
PacoBlaze	PicoBlaze	8	Open-source (BSD)	ne	2	-	-	-	204	Open-source klon PicoBlaze-a
LatticeMico8	LatticeMico8	8	Open-source	ne	2	-	-	-	200	Nije ograniče-no na Lattice uređaje (može biti upotrijebljeno i drugdje)

1 - navedena vrijednost može biti valjana za većinu instrukcija. Npr. množenje često traje više ciklusa nego obična ALU instrukcija.

2 - Memory Management Unit - jedinica za upravljanje memorijom

3 - Hardversko množilo

4 - jedinica za pomičnu točku

5 - Područje se mjeri u Logičkim elementima (LE) koje se sastoji od 4-ulazna LUT-a i bistabila. Estimirano područje služi samo za referencu.

2.4.1 S1 jezgra, LEON2 i LEON3, Picoblaze, LatticeMico8

S1 jezgra je open-source implementacija SPARCv9 arhitekture. 21. ožujka 2006. Sun Microsystems, tvrtka koja je razvila SPARC arhitekturu, pustila je na tržište UltraSPARC T1 mikroprocesor pod GNU General Public License (GPL).

UltraSPARC T1 je 8-bitna implementacija SPARCv9 64-bitne arhitekture, dizajnirana kako bi radila u uvjetima kad je dovod energije iz okoline kritičan.

UltraSPARC T1 sama po sebi je prevelika da bi se implementirala na FPGA. S1 jezgra je "slabija" verzija UltraSPARC T1 koja sadrži samo jednu CPU jezgru i dodatni kontroler za Wishbone sabirnicu. Čak i takva krnja verzija zauzima 37000-60000 Virtex-5 LUT-ova (ovisno o verziji) i prevelika je za većinu SoC dizajna.

GPL licenca osigurava slobodno korištenje jezgre, ali se traži odricanje izvornog koda cijelog dizajna kod distribuiranja uređaja baziranih na S1 jezgri.
LEON3 je open-source implementacija SPARCv8 32-bitne arhitekture dizajnirane od strane Aeroflex Gaisler AB. Dio je GRLIB open-source IP core library dostupan pod GPL uvjetima.

LEON3 je potpuno funkcionalan procesor s protočnom strukturom SPARCv8 i sadrži opcionalnu jedinicu za operacije s brojevima s pomičnom točkom. GRLIB library sadrži mnogo korisnih IP blokova poput sabirničkih i RAM kontrolera. Koristi otprilike 3500 Virtex-4 LUT-ova i stoga može biti lako korišten u aplikacijama između srednje i visoke "gustoće" (eng. *mid-to-high densitiy applications*).

GPL licenca dopušta slobodno korištenje LEON3 CPU jezgre, ali samo u open-source hardware projektima. Kako bi se odstranila mogućnost za distribuciju source-a za cijeli dizajn, potrebna je posebna komercijalna dozvola od Aeroflex Gaisler.

Postojala je starija LEON2 jezgra licencirana pod blažom LGPL licencom koja je dopuštala korištenje jezgre u vlasničkim (eng. *proprietary*) projektima (iako su neki uvjeti morali biti ispunjeni.)

LEON2 je open-source CPU jezgra koja je implementacija 32-bitne SPARCv8 arhitekture, napisana od strane Jiri Gaisler-a. Ova verzija je uklonjena sa službenog site-a, Aeroflex Gaisler, u korist nove verzije, LEON3. Nova verzija

ima mnoga poboljšanja, ali neki ljudi i dalje preferiraju staru zbog liberalnije licence LGPL, koji dopušta slobodnu komercijalnu upotrebu (pod određenim uvjetima). Za LEON3, licenca je stroža (GPL) i dopušta korištenje samo u open-source projektima (komercijalna licenca se i dalje može dobiti od Aeroflex Gaisler za korištenje LEON3 u projektima zatvorenog source-a eng. closed-source projects).

PicoBlaze je proprietary (ali besplatna) 8-bitna RISC arhitektura i CPU jezgra razvijena od strane tvrtke Xilinx. Iako je besplatna, jezgra je vezana za Xilinx arhitekturu. Postoji i open-source klon imenom PacoBlaze, CPU jezgra neovisna o uređaju i kompatibilna s originalnim PicoBlaze-om.

PicoBlaze je dizajniran kako bi radio s FPGA-ovima niske gustoće (eng. *low-density* FPGAs) i kako bi zauzimao 100 Spartan/Virtex slojeva (eng. *slices*).

LatticeMico8 je open-source 8-bitna RISC arhitektura i CPU jezgra razvijena od strane tvrtke Lattice Semiconductor i napisaa u HDL-u neovisnom o uređaju/platformi. (tj. nije vezano za Lattice FPGA sustave). Usporediva je s PicoBlaze-om.

2.4.2 DSPuva16, OpenRISC, LaticeMico32, Nios II, Microblaze i klonovi

DSPuva16 je mala 16-bitna open-source CPU jezgra. Zanimljiva značajka je što je vrlo mala (usporediva s PicoBlaze-om), a opet sadrži namjensko hardversko množilo pa ju je moguće koristiti u FPGA-ovima male gustoće (eng. low-density FPGAs).

Negativna strana je ta da CPU podržava malu programsku memoriju i nema podatkovne memorije (osim unutarnjih registara). Ovaj procesor se dakle može koristiti za implementaciju jednostavnih algoritama koji uključuju mnogo operacija množenja. Jedna moguća takva operacija je automatska kontrola.

DSPuva16 jezgru je napisao Santiago de Pablo sa sveučilišta u Valladolidu, software je napisao/la Carmen Cascon.

OpenRISC 1200 je 32-bitna open-source jezgra koja implementira *OpenRISC 1000* RISC arhitekturu (naziv jezgre - OpenRISC 1200, a naziv arhitekture - OpenRISC 1000) razvijena od strane OpenCores.org zajednice.

OpenRISC arhitektura koristi 5-razinsku protočnu strukturu i vlastiti skup instrukcija, a koristi GNU toolchain i preneseni („portani“) Linux OS na OpenRISC arhitekturu.

OpenRISC 1200 je dostupan pod uvjetima LGPL licence, tako da je moguće koristiti u vlasničkim (proprietary) projektima.

LatticeMico32 je open-source 32-bitna RISC arhitektura i CPU jezgra razvijena od strane tvrtke Lattice Semiconductor. Treba napomenuti da za razliku od Xilinx-a i Altera-e, Lattice kreira prijenosan (portabilan) dizajn tako da se može koristiti na drugim FPGA-ovima ili čak na ASIC-ovima. Usporedive je kvalitete s MicroBlaze-om i Nios II/s, ne uključuje jedinicu za operacije s pomičnom točkom. GNU toolchain i Linux OS se mogu portati na LatticeMico32 arhitekturu.

Nios II je proprietary 32-bitna RISC arhitektura i procesorska jezgra razvijena od strane tvrtke Altera za upotrebu u njihovim FPGA sustavima. Nasljednik je starijeg 16-bitnog Nios ugradbenog procesora. Nios II soft jezgra dolazi u 3 verzije:

1. Nios II/f –

   optimirana za maksimalne performanse, 6-stupanjska protočna struktura, 1 instrukcija po ciklusu i uključuje razdvojeni instrukcijski i podatkovnu priručnu memoriju (eng. cache, u nastavku često korišten termin), MMU i MPU (ranije opisani), hardversko množilo, operacije dijeljenja i pomaka.

2. Nios II/s –

   dizajniran kao kompenzirani model, 5-stupanjska protočna struktura, 1 instrukcija po ciklusu i instrukcijski cache (nema podatkovni cache), hardversko množilo, operacije dijeljenja i pomaka, bez MMU i MPU jedinica.

3. Nios II/e –

   dizajniran kako bi zauzimao što je manje moguće prostora, ne koristi protočnu strukturu, izvršava jednu instrukciju u 6 ciklusa i nema dodatnih aritmetičkih blokova. Broj logičkih elemenata u Nios II jezgri je u rangu MicroBlaze-a, osim što Nios II/e koristi znatno manje logičkih elemenata.

Nios II može biti korišten jedino na Altera-inim FPGA-ovima ili na Altera HardCopy strukturiranim ASIC-ovima. GNU toolchain i Linux OS se mogu portati na Nios II.

MicroBlaze je proprietary 32-bitna RISC arhitektura i soft CPU jezgra dizajnirana od strane tvrtke Xilinx za upotrebu u njihovim FPGA sustavima. Postoje i open-source klonovi MicroBlaze-a, imenom aeMB i OpenFire, koji su neovisni o uređaju i kompatibilni su s originalom.

S obzirom da je razvijen posebno za FPGA, MicroBlaze dizajn zauzima manje prostora nego LEON ili OpenRISC bazirani dizajni, što je osobito pogodno za FPGA sustave srednje gustoće (eng. mid-densitiy FPGAs).

Valja napomenuti da originalni MicroBlaze implementira značajke koje nisu podržane od strane open-source klonova, poput pomične točke ili 5-stupanjske protočne strukture (open-source klonovi implementiraju samo 3-stupanjsku protočnu strukturu). Postoji mogućnost i implementiranja uClinux OS-a na MicroBlaze arhitekturu.

2.4.3 Cortex M1

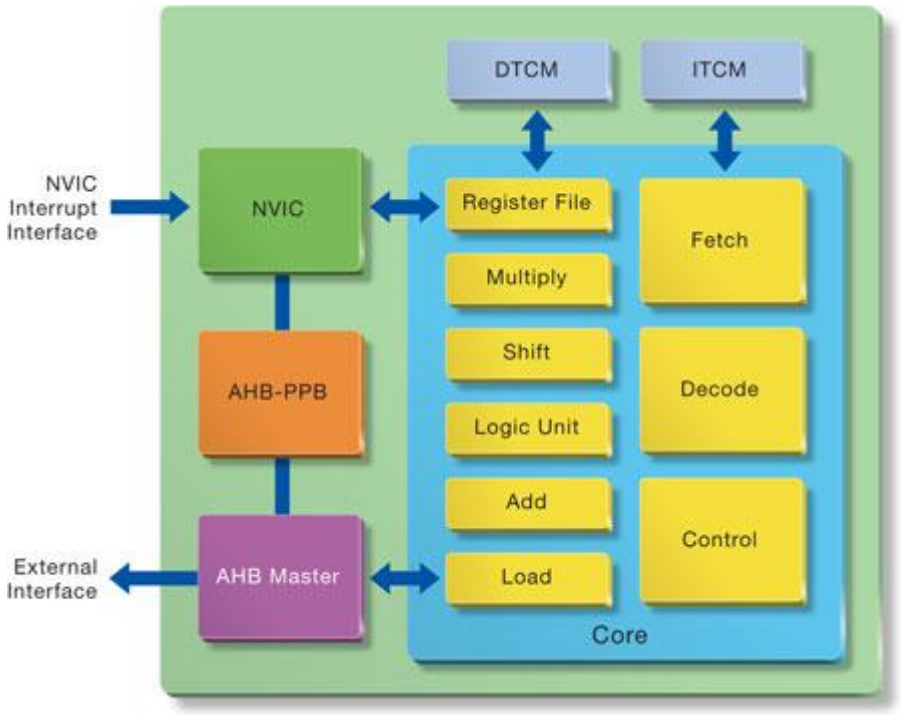
Cortex-M1 je implementacija proprietary 32-bitne ARMv6 arhitekture dizajnirane za FPGA. Korištenje Cortex-M1 zahtjeva licencu ARM Limited. Razvijen od strane ARM-a u suradnji s Actel-om, 32-bitni ARM Cortex-M1 je prvi ARM procesor dizajniran za FPGA implementacije. ARM (eng. *Advanced RISC Machine*) je popularna RISC arhitektura posebno pogodna za aplikacije koje zahtjevaju manju potrošnju energije.

Uzevši u obzir balans između veličine i brzine, besplatan Cortex-M1 procesor radi na 60 MHz i može se implementirati na 4.353 pločica (eng. *tiles*). Uravnotežena 3-stupanjska protočna struktura, s podržanim Tumb 2 skupom instrukcija tako da se postojeći Tumb kod može koristiti bez promjena. Podesivi Cortex M1 procesor se povezuje na najbržu sabirnicu (eng. *Advanced High Performance Bus* – AHB), omogućavajući dizajnerima da izgrade svoj podsustav i lako dodaju periferije. Osim SmartDesgin-a, CoreConsole i SoftConsole od Actel-a i RealView-a alata od ARM-a, drugi dobavljači također nude alate od prevodioca i debugger-a do RTOS rješenja.

Postoje Actel-ovi FPGA chip-ovi bazirani na flash-u koji se dobivaju s Cortex-M1 licencom (nije potrebna druga licenca). Cortex-M1 se tkđ. može koristiti sa Xilinx-ovim i Altera-inim FPGA sustavima. Mnogi se OS-ovi mogu portati na ARM, uključujući Linux i Windows CE.

2.4.3.1 Specifikacije Cortex-M1

Dizajniran specifično za FPGA implementaciju, ARMv6-M instrukcijski skup, izvršava cjelokupni Thumb kod - može izvoditi ARM7 i ARM9 Thumb podrutine, 3 stupanjska 32-bitna protočna struktura, razdvojena memorija i AHB-Lite sučelje, konfigurabilan prekidni kontroler s prekidnim vektorima (eng. *nested vectored interrupt controller*, u nastavku korištena skraćenica NVIC) opcionalno brzo ili malo množilo, kompatibilno s Cortex-M3, korisnički-programirano u FPGA. Sve Cortex-M1 U/I jedinice i signali su dostupni korisniku, optimiran za Actel-ove flash bazirane M1 uređaje, dostupno bez plaćanja licence, s ugrađenim real-time debug i JTAG sučeljem te podrškom velikog broja razvojnih alata.



Slika 1. Shema Cortex - M1 jezgre

2.4.3.2 M1 uređaji

Cortex-M1 je dostupan za upotrebu u M1 uređajima flash familije. Uređaji su bazirani na Actel-ovim non-volatile flash (neizbrisivi flash) IGLOO (M1AGL), ProASIC3 (M1A3P/M1A3PE), ProASIC3L (M1A3PL) i Fusion (M1AFS) uređajima s karakteristikama navedenim u sljedećoj tablici:

Tablica 2. Popis Cortex-M1 uređaja

M1 uređaji	**250**	**400**	**600**	**1000**	**E1500**	**E3000**
M1 IGLOO	+		+	+		+
M1 ProASIC3	+	+	+	+	+	+
M1 ProAsic3L			+	+		+
M1 Fusion	+		+		+	

Napomena: sve v2.x verzije ARM Cortex M1 na Actel M1 uređajima imaju jednu user-selectable konfiguracijsku opciju - sa ili bez debug-a. Osim toga, prekonfigurirani su 0K ITCM, 0K DTCM, malim množilom, little-endian, bez OS proširenja i s jednim prekidom (interrupt-om).

2.4.3.2.1 IGLOO (M1AGL)

M1 IGLOO uređaji su reprogramirljivi, sa “svom opremom” - full-featured flash FPGA dizajniran kako bi zadovoljio zahtjeve za potrošnjom snage vezane uz današnju prijenosnu elektroniku. Uz Flash Freeze tehnologiju i radni napon od 1.2 V/ 1.5 V, ti uređaji nude industriji najmanju potrošnju.

2.4.3.2.2 ProASIC3 (M1A3P/M1A3PE)

M1 ProASIC3 uređaji, bazirani na trećoj generaciji Actel flash FPGA-ova, pružaju „low-power“ , „live-at-power-up“ , „single-chip“ rješenja. Reprogramabilni su i nude brze time-to-market beneficije na ASIC razini jediničnog troška. Te značajke omogućavaju inženjerima kreiranje sustava s visokim performansama, aplikacija visoke gustoće (eng. *high-density system applications*) s Cortex-M1 koristeći postojeće FPGA dizajne i alate.

2.4.3.2.3 ProASIC3L (M1A3PL)

ProASIC3L uređaji, bazirani na Actel ProASIC3 FPGA sustavima, nude nisku potršnju, visoke performanse i nisku cijenu s „mixed voltage core“ podrškom. Ugradnjom Flash Freeze tehnologije, može se postići trenutna promjena (unutar 1 us) iz aktivnog u statično stanje, s dodatkom od 40% dinamične štednje energije s operacijama izvedevim na 1.2 V.

2.4.3.2.4 Fusion (M1AFS)

Actel Fusion je prvi mixed signal FPGA. Fusion integrira 12-bitni A/D pretvornik, 40 analognih U/I jedinica, do 8 Mbit-a flash memorije i FPGA „fabric“ u jednom uređaju.

2.4.3.3 Intelektualno vlasništvo

Važan skup funkcionalnih blokova potrebnih za korištenje Cortex-M1 u aplikacijama su podsistemske periferije. Sljedeće IP (eng*. intelectual property* – intelektualno vlasništvo) jezgre su dostupne u Libero IDE katalogu i SmartDesign-u te se mogu koristiti s Cortex-M1:

- CoreGPIO
- CorePCIF
- CoreI2C
- Core10/100
- Core429
- CoreAHB2APB
- CoreAI
- CoreCFI
- CoreFMEE
- CoreFROM
- CoreInterrupt
- CoreMemCtrl
- CorePWM
- CoreTimer
- CoreUARTapb
- CoreWatchdog

Actel tkđ. nudi softver za ove jezgre, s fokusom na funkcionalnosti jezgre, a ne na interne operacije. FirmWare katalog uključuje sve moguće drivere s instaliranim LiberoIDE ili je dostupan kao samostalan.

2.4.3.4 Dostupni alati

Libero Integrated Design Environment (IDE) je Actelov sveobuhvatan softvere za dizajn i razvoj FPGA-ova.

SmartDesign je alat za kreiranje grafičkih blokova i za dizajn kompletnog FPGA i SoC, uključujući procesore, „DirectCores“, jezgre iz standardne biblioteke, korisničke IP i prilagođeni, vlastiti HDL. Automatski kreira HDL spreman za sintezu. Radi u sklopu Libero IDE.

SoftConsole je programsko okruženje za razvoj Actel procesora. Uključuje GNU C/C++ prevodioc, GDB debugger i simulator.

CoreConsole je Actel-ova platforma za intelektualno vlasništvo (eng. *intellectual property deployment platform* – IDP). Uključuje grafičko sučelje i „block stitcher“ u cilju pojednostavljenja kod IP jezgri za ugradbene aplikacije u FPGA-ovima. Radi samostalno ili s Libero IDE.

2.4.3.5 Hardverski dizajn

Actel FlashPro3 programator omogućuje programiranje Cortex-M1 IGLOO, ProASIC3, ProASIC3L i Fusion uređaja, kao i standardne verzije tih uređaja - podržanih s unutar sistemskim (eng. in-system) programiranjem (ISP). Konfiguracijski podatci se isporučuju kroz standardno JTAG sučelje mikroprocesora, Silicon Sculptor 3 ili FlashPro3.

M1-omogućen IGLOO Starter Kit je kompletna platforma za razvoj sustava s ARM Cortex-M1 u M1 omogućenim IGLOO FPGA sustavima. Uključuje ploču (eng. *board*, termin često korišten u daljnjem tekstu) s M1AGL600 uređajem, Actel Libero IDE Gold, SmartDesign, SoftConsole, CoreConsole i on-board FlashPro3 programator. Omogućuje razvoj FPGA aplikacije male snage (low power) s ARM Cortex-M1.

M1-omogućen ProASIC3L Development Kit je kompletno okružje za razvoj, čak i prototipova. Uključuje board s M1A3P1000L uređajem, Actel Libero IDE Gold, SmartDesign, SoftConsole, CoreConsole i on-board FlashPro3 programator. Omogućuje razvoj aplikacija s ARM Cortex-M1 u Actel non-volatile M1 omogućenim ProASIC3L uređajima, podržava ISP, serijalizaciju uređaja (eng. device serialization) i FlashLock on-chip sigurnost sustava.

2.5 AVR

AVR predstavlja modificiranu Harvardsku arhitekturu (program i podaci su pohranjeni u zasebnim dijelovima memorije i pojavljuju se u zasebnim adresnim prostorima, ali imaju mogućnost čitanja podataka iz programske memorije koristeći posebne instrukcije) 8 bitnog RISC single chip mikrokontrolera razvijenu od strane tvrtke Atmel 1996. i jedna je od prvih mikrokontrolerskih familija koja je koristila on-chip flash memoriju za pohranu programa.

2.5.1 Osnovne familije

AVR se generalno svrstava u pet širokih grupa:

- tinyAVR - ATiny serija, 0.5-8 kB programske memorije, 6-8 pin-ova pakiranje i ograničeni skup periferija
- megaAVR - ATmega serija, s proširenim skupom instrukcija (eng. multiply instructions i instrukcije za rukovanje većim programskim memorijama) te opsežnim skupom periferija
- XMEGA - ATxmega serija, 16-384 kB programske memorije, 44-64-100 pin-ova pakiranje (A4, A3, A1), s dodatcima za poboljšanje performansi (poput DMA, "Event System" i podrška za kriptografiju) te opsežan set periferija s DAC-om (detaljno objašnjeno u 3. poglavlju)

- AVR za specifične aplikacije:
  - megaAVR s posebnim dodacima koji se ne nalaze kod ostalih "članova" AVR familije poput LCD kontrolera, USB kontrolera, naprednog PWM, CAN itd.
  - FPSLIC™ (AVR s FPGA) - FPGA s 5K do 40K vratiju, SRAM memorijom za programski kod AVR-a (za razliku od svih drugih AVR-ova). AVR jezgra može raditi na taktu do 50 MHz.

- 32-bitni AVR-ovi:
  AVR32 arhitektura koja uključuje SIMD i DSP instrukcije, zajedno s drugim dodacima za obradu audia i videa. Ta familija je namijenjena konkuriranju

ARM-ovim procesorima. Instrukcijski skup je sličan drugim RISC jezgrama, ali nije kompatibilan s originalnom AVR ili bilo kojom ARM jezgrom.

2.5.2 Specifikacije

Trenutne AVR familije sadrže sljedeće specifikacije:

- Multifunkcionalnost, dvosmjerne opće namjenske U/I portove s podesivim, ugrađenim pull-up otpornicima.
- Više unutarnjih oscilatora, uključujući RC oscilator bez vanjskih dijelova
- Unutarnju, samo-programirljivu instrukcijsku flash memoriju do 256 kB (384 kB na XMega)
- "In-system" programirljivost koristeći serijsko/paralelno nisko-naponsko sučelje ili JTAG
- Opcionalni dio za „boot" kod s nezavisnim bitovima za zaključavanje u svrhu zaštite
- On-chip debugging (OCD) podrška kroz JTAG ili debugWIRE na većini uređaja
- JTAG signali (TMS, TDI, TDO i TCK) su multipleksirani na GPIO. Ti pinovi mogu biti podešeni da funkcioniraju kao JTAG ili GPIO ovisno o tome kako je podešen "osiguravački" bit (eng. fuse bit), koji može biti programiran putem ISP ili HVSP. Po defaultu, AVR-ovi s JTAG dolaze s omogućenim JTAG sučeljem.
- debugWIRE koristi /RESET pin kao dvosmjerni komunikacijski kanal da bi pristupio on-chip debug krugu. Prisutno na uređajima s manje pin-ova, s obzirom da traži samo 1 pin
- Unutar podatkovni EEPROM je do 4 kB
- Unutarnji SRAM je do 16 kB (32 kB na XMega)
- Vanjska 64 kB little endian podatkovni prostor na određenim modelima, uključujući Mega8515 i Mega162.
- Vanjski podatkovni prostor je "obložen" s unutarnjim podatkovnim prostorom, na način da se puni 64 kB adresni prostor ne pojavljuje na vanjskoj sabirnici. Pristupom prema npr. adresi 0100_{16} ćemo pristupiti unutarnjem RAM-u, ne vanjskoj sabirnici.

- U određenim modelima XMega serije, vanjski podatkovni prostor je poboljšan u svrhu podrške SRAM-a i SDRAM-a. Načini za adresiranje podataka su proširenі da bi se omogućilo direktno adresiranje do 16 MB podatkovne memorije.
- AVR-ovi općenito ne podržavaju izvršavanje koda iz vanjske memorije. Neki ASSP-ovi koji koriste AVR jezgru podržavaju vanjsku programsku memoriju.
- Sadrže 8-bitna i 16-bitna brojila
- PWM izlaz (Neki uređaji imaju pojačanu PWM periferiju koja uključuje "dead-time" generator)
- Dio za snimanje ulaza (eng*. input capture*)
- Analogni komparator
- 10 ili 12-bitni A/D pretvarači, s multipleksiranjem do 16 kanala
- 12-bitni DA pretvarači.
- Niz serijskih sučelja, uključujući I^2C kompatibilno dvožično sučelje (eng. Two-Wire Interface - TWI), sinkrone/asinkrone serijske periferije (UART/USART) korištene s RS-232, RS-485 itd.
- Serial Peripheral Interface Bus (SPI)
- Univerzalno serijsko sučelje (USI) za dvo ili trožični sinkroni prijenos podataka
- Detekcija propada napajanja (eng. *Brownout detection*)
- Watchdog brojač (WDT)
- Višestruki načini za uštedu snage (eng. *power-saving sleep modes*)
- Modeli kontrolera za kontrolu rasvjete i pokreta(nja) (eng. lighting and motor control)
- Podrška za CAN, USB.
- Pravilni full-speed hardware (12 Mbit/s) i hub kontroler s ugrađenim AVR-om. Tkđ. su besplatne i dostupne nisko-brzinske (eng. *low-speed*) 1.5 Mbit/s (HID) softverske emulacije za „bitband-ing".
- Podrška za ethernet, LCD
- Nisko naponski uređaji koji rade na 1.8 V (do 0.7 V za dijelove s ugrađenim DC-DC pretvaračima)
- Podržavaju picoPower uređaje

- Sadrže DMA kontrolere i “event system” perifernu komunikaciju
- Podrška za brzu kriptografiju AES i DES algoritma

2.5.3 Osnovna arhitektura

Flash, EEPROM i SRAM su integrirani na jedan chip, uklanjajući potrebu za vanjskom memorijom u većini aplikacija. Neki uređaji imaju opciju za paralelnu vanjsku sabirnicu da bi omogućili dodavanje podatkovne memorije ili uređaja koji su memorijski mapirani (eng. memory-mapped devices). Skoro svi uređaji (osim najmanjeg TinyAVR chip-a) imaju serijsko sučelje, koje se može upotrijebiti za povezivanje većih serijskih EEPROM ili flash chip-ova.

2.5.3.1 Programska memorija

Programske upute, tj. instrukcije su pohranjene u neizbrisivu (eng. non-volatile) flash memoriju. Iako su MCU jedinice 8 bitne, svaka instrukcija zauzima jednu ili dvije 16-bitne riječi. Veličina programske memorije je obično naznačena pri imenovanju samog uređaja (npr. ATmega64x znači 64 kB flash memorije dok ATmega32x znači da ima 32 kB).

Ne postoji odredba za „off-chip“ programsku memoriju; sav kod koji se izvršava od strane AVR jezgre mora se nalaziti u flash memoriji na chip-u. Međutim, to ograničenje se ne odnosti na AT94 FPSLIC AVR/FPGA chip-ove.

- Unutarnja podatkovna memorija - adresni prostor podataka se sastoji od register file-a (RF, registarska datoteka, objašnjeno u daljnjem tekstu i 3. poglavlju), U/I registara i SRAM-a.

- Unutarnji registri - AVR-ovi imaju 32 jedno-bajtna registra i klasificirani su kao 8-bitni RISC “uređaji”. U većini inačica AVR arhitekture, radni registri su mapirani u prve 32 memorijske lokacije (0000-001F$_{16}$) nakon kojih slijedi 64 U/I registara (0020-005F$_{16}$). Stvarni SRAM počinje nakon svih tih sekcija (od adrese 0060$_{16}$). (Uzeti u obzir da prostor za U/I registre može biti veći na nekim uređajima; u tom slučaju memorijski mapirani U/I registri će zauzimati dio lokacija SRAM-a).

Iako postoje zasebne adresne sheme i optimirane instrukcije, za pristup RF-u i U/I registara, sve se dalje može adresirati i manipulirati kao da se nalazi u SRAM-u.

U XMEGA varijanti, radni RF nije mapiran u podatkovni adresni prostor; posljedica je ta da nije moguće tretirati ijedan XMEGA radni registar kao da je SRAM. Umjesto toga, U/I registri su mapirani u podatkovni adresni prostor na samom početku. Dodatno, količina podatkovnog adresnog prostora posvećenog U/I registrima je znatno povećana na 4096 bajtova (0000-0FFFF$_{16}$). Međutim, instrukcije za brzo manipuliranje U/I registrima može samo dohvatiti prvih 64 U/I registarske lokacije (prve 32 lokacije su za bitovne, eng. *bitwise* instrukcije). Nakon U/I registara, XMEGA serija izdvaja 4096 bajtova iz podatkovnog adresnog

prostora koji se može opcionalno iskoristiti za mapiranje interne EEPROM memorije u podatkovni adresni prostor od adrese 1000-1FFF$_{16}$. Stvarni SRAM je smješten nakon tih lokacija, počevši od 2000$_{16}$.

2.5.3.2 Izvršavanje programa

Atmel-ovi AVR-ovi imaju dvostupanjsku, jedno razinsku protočnu strukturu. To znači da se sljedeća instrukcija dohvaća dok se trenutna izvršava. Većina instrukcija traje 1 ili 2 ciklusa, čineći AVR relativno brzim među 8-bitnim mikrokontrolerima.

AVR procesori su dizajnirani s ciljem za učinkovito izvršavanje C koda. Takt rada se kreće između 0-20 MHz, s time da neki uređaji dosežu i 32 MHz. Operacije s niskom potršnjom energije obično zahtjevaju manju frekvenciju takta.

S obzirom da su sve operacije na registrima od R0-R31 jednociklusne, AVR može postići 1 MIPS po Hz, odnosno procesor s taktom od 8 MHz može dostići 8 MIPS-a. Load i store naredbe traju 2 ciklusa, grananje traje 2 ciklusa.

2.5.4 8 i 32-bitni mikrokontroleri tvrtke Atmel

U nastavku teksta bit će dan pregled trenutnih familija uređaja:

Tablica 3. Specifikacije UC3 uređaja

Familija uređaja	Sažetak prednosti	Aplikativnost	Podržane tehnologije	Ključne značajke
32-bitni AVR UC3	najefikasniji 32-bitni mikrokontroler	opća namjena	picoPower Sleepwalking FlashVault DMA Event System Samo programirljivost	16-512 KB flash 48-144 pin-ova do 66 MHz 1.5 MIPS/MHz

- ✓ 32-bitni AVR UC3 podržava DSP operacije s fiksnom točkom, sadrži dvopristupni SRAM, višeslojnu podatkovnu sabirnicu, DMA kontroler za periferije, periferni Event system i podršku za inteligentne periferije. Određeni UC3 uređaji uključuju integriranu jedinicu za operacije s pomičnom točkom (eng. *floating point unit* - FPU).

Tablica 4. 8/16 bitni XMEGA

Familija uređaja	Sažetak prednosti	Aplikativnost	Podržane tehnologije	Ključne značajke
8/16 bitni AVR XMEGA	ekstremna 8-bitna performansa	opća namjena	picoPower Sleepwalking DMA Event System EEPROM Samo programirljivost	16-512 KB flash 44-100 pin-ova do 32 MHz 1.0 MIPS/MHz

- ✓ Podrška za AES i DES kriptiranje, sadrži brze analogne module, fleksibilne Timere/Brojila, višestruke komunikacijski module, upravljanje napajanjem, DMA kontroler. Svi AVR XMEGA su „code-compatible“, tj. kompatibilni po kodu.

Tablica 5. 8 bitni megaAVR

Familija uređaja	Sažetak prednosti	Aplikativnost	Podržane tehnologije	Ključne značajke
8 bitni megaAVR	više periferija	opća namjena rasvjeta LCD	QTouch picoPower SleepWalking Event System EEPROM Samo programirljivost	4-256 KB flash 28-100 pin-ova do 20 MHz 1.0 MIPS/MHz

- ✓ Idealan za aplikacije s velikom količinom koda, performanse do 20 MIPS. PicoPower tehnologija smanjuje potrošnju. Svi megaAVR-ovi nude samo-programirljivost za brže, sigurnije, jeftinije in-circuit nadogradnje. Podržava mogućnost nadogradnje flash-a tijekom izvršavanja aplikacije.

Tablica 6. 8 bitni tinyAVR

Familija uređaja	Sažetak prednosti	Aplikativnost	Podržane tehnologije	Ključne značajke
8 bitni tinyAVR	malen i moćan	opća namjena ograničen prostor na ploči	QTouch EEPROM Operacije na 0.7 V	0.5-8 KB flash 6-32 pin-ova do 20 MHz 1.0 MIPS/MHz

- ✓ Atmelovi tinyAVR uređaji su optimizirani za aplikacije koje zahtjevaju performanse, efikasnu potrošnju i jednostavnost korištenja u malom pakiranju. Svi su bazirani na istoj arhitekturi i kompatibilni su s drugim AVR uređajima. Integrirani AD pretvornik, EEPROM i detektor za propad napajanja, flash, „on-chip debug“ , jedini uređaj koji radi na 0.7 V.

Tablica 7. Upravljanje baterijama

Familija uređaja	Sažetak prednosti	Aplikativnost	Podržane tehnologije	Ključne značajke
Upravljanje radom baterija - battery management - Li-Ion		mjerenje razine plina balansiranje ćelija ovjera zaštita protiv kratkog spoja i zagrijavanja	1-4 ćelije High Side N-MOS Brojilo Kulona Temperaturni senzor Samo programirljivost	operacije od 1.8- 25 V 8-40 KB flash 28-48 pin-ova do 8 MHz 1.0 MIPS/MHz

- ✓ Atmelovi AVR nude single-chip proizvode za upravljanje Li-Ion baterijama. Dizajnirane za povećanje životnog vijeka i energije po ciklusu, sadrže AD pretvornike prilagođene za mjerenje količine goriva unutar baterija (eng. fuel gauging) i praćenje razina napona. Atmelove baterije imaju podršku za mehanizme autentifikacije da bi se odbile moguće nesigurne kopije.

2.6 pAVR

U ovom potpoglavlju, uz karakteristike i specifikacije, dane su specifikacije 8 bitnog kontrolera kompatibilnog s Atmelovom AVR arhitekturom u svrhu uvoda u sljedeće poglavlje.

U trećem poglavlju bit će sprovedena detaljna analiza VHDL implementacije, testnih kodova te opis dijelova procesora.

pAVR (skraćenica od *pipelined* AVR - AVR s protočnom strukturom) nije specifičan kontroler AVR familije nego maksimalno specificiran AVR. Dovoljno je podesiv da može simulirati većinu kontrolera AVR familije. pAVR je triput brži nego originalna jezgra ako je izgrađen istom tehnologijom. Spada u vrstu FPGA klona koji omogućava performanse koje su bolje od originalne jezgre, poput primjerice dublje protočne strukture.

2.6.1 Specifikacije pAVR-a

- protočna struktura sa 6 razina (originalna AVR jezgra ima 2)
- 1 clock/instrukciji za većinu instrukcija
- estimirana frekvencija takta : ~50 MHz i 0.5 um; uz pretpostavku da Atmelova jezgra radi na 15 MHz i 0.5 um. Otprilike tri puta veće performanse nego kod originalne Atmelove jezgre.
- estimirani MIPS na 50 MHz: 28 MIPS (tipično), 50 MIPS (najviše). Također, otprilike tri puta veće performanse nego kod originalne Atmelove jezgre. Na 15 MHz, Atmelova jezgra ima tipično 10 MIPS-a i 15 MIPS najviše.
- CPI (perioda po instrukciji - clocks per instruction): 1.7 clock/instrukciji (tipično), 1 clock/instrukciji najviše. Približno 0.75 puta (tipično), jedamput (najviše) „veće” performanse nego kod originalne Atmelove jezgre.
- do 32 izvora prekida. Svaki prekid ima programabilni prioritet i adrese na koje se skače u tom slučaju
- izričito parametriziran dizajn koji dopušta fleksibilnost

3. pAVR – izvedba, sheme, kod, analiza

Unutar ovog poglavlja bit će dana detaljna analiza 8-bitnog pAVR kontrolera koristeći VHDL (eng. *Very High speed integrated circuits Hardware Description Language*), stavljene i opisane sve relevantne sheme, protumačena protočna struktura te protumačeni rezultati testnih kodova.

Kao što je navedeno ranije, pAVR nije specifičan kontroler AVR familije nego je AVR s maksimalnim poboljšanjima koji je dovoljno konfigurabilan da simulira većinu AVR familije.

Cilj je dobiti AVR procesor koji će biti moćan koliko je god moguće (u terminima MIPS-a).

Source kodovi su modularizirani. Pisani su s ciljem poštivanja smislenih konvencija (dijeljenje procesa, imenovanje signala itd.) tako da pAVR ima prenosivi dizajn.

Naglasak je na samoj jezgri, dok je periferijama dana sekundarna važnost. Timer i 8 bitna U/I jedinica su implementirane za demonstracijske svrhe.

Kako je cilj ostvariti dublji pipeline radi veće brzine i boljih performansi, treba naglasiti da je prosječni period po instrukciji (eng. *clock per instruciton* – CPI) ipak veći nego kod AVR, zbog skokova, grananja, poziva i povrata. Te naredbe zahtjevaju da se protočna struktura povremeno bar djelomično isprazni (eng. flush – ispust, detaljnije objašnjeno u daljnjem tekstu) tako da se poneki period izgubi dok se puni protočna struktura.

Kodovi su preuzeti sa stranice doru.info, vrlo malo su modificirani, a autor cjelokupnog dizajna je Stefan Kristiansson. Okvirno radno vrijeme koje se preporuča za razvoj ovog procesora je 6 mjeseci do godinu dana po čovjeku.

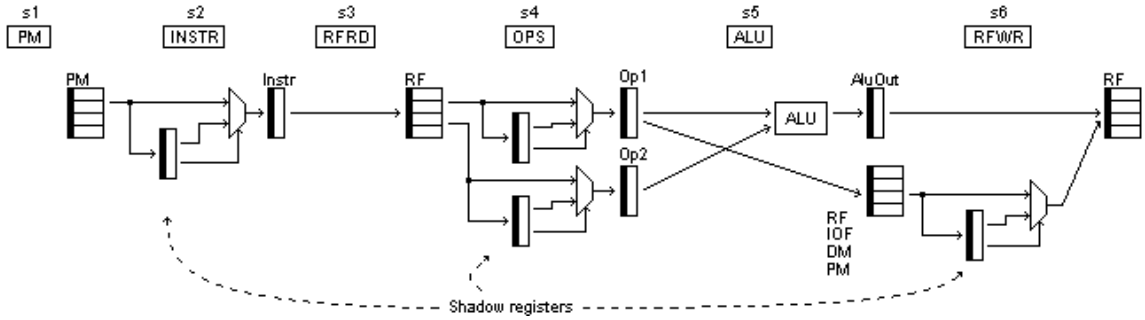
3.1 Implementacija

3.1.1 Protočna struktura

pAVR sadrži protočnu strukturu sa šest razina (u nastavku poglavlja će biti redovito upotrebljavani izrazi poput „dijelovi“ i „faze“):

1. Čitanje programske memorije (eng. *program memory*) – PM
2. Prosljeđivanje podatka iz izlaza programske memorije u instrukcijski registar – INSTR
3. Dekodiranje instrukcije i čitanje registarske datoteke (eng. *register file*) – RFRD
4. Čitanje izlaza registarske datoteke – OPS
5. Izvršavanje ili pristup jedinstvenoj memoriji – ALU
6. Pisanje u registarsku datoteku – RFWR

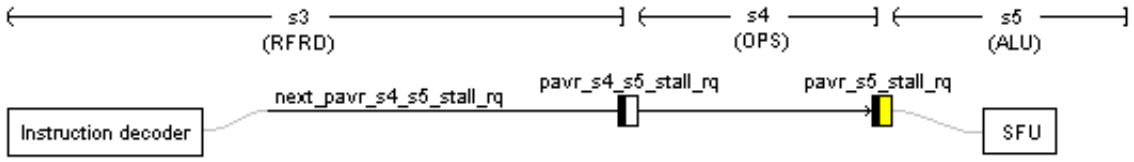
U nastavku će se upotrebljavati skraćenice za svaku od ovih razina - s1 (što predstavlja eng. *stage* – faza, dio), s2, s3, s31, s41 i sl.



Slika 2. Protočna struktura pAVR-a

Svaki dio protočne strukture je nezavisni automat stanja.

U osnovi, svaki dio protočne strukture prima vrijednost iz prethodnog stupnja. Jedino „terminalni“ registri drže podatke koji se koriste, dok se prethodni koriste samo za sinhronizaciju. Npr. prijenos hardverskog resursa kroz dijelove strukture 3, 4 do dijela 5.



Slika 3. Prijenos hardverskog resursa

Iznimke kod ovog „normalnog“ protoka su zadržavanje/odugovlačenje (eng. *stall*) i ispuštanje (eng. *flush*) akcije, koje se mogu nezavisno zadržati ili resetirati na nulu svaki dio. Ostale iznimke su kada koristimo nekoliko registara u takvom lancu, a ne samo terminalne.

Osim glavnih dijelova protočne strukture (dijelovi od 1-6), postoje dijelovi koji služe samo za određene instrukcije (poput operacija nad 16 bitnim brojevima, skokovi, povrati): s61, s51, s52, s53 i s54. Dok su aktivni ti dijelovi, glavni dijelovi su u stall fazi.

Dijelovi s1, s2 su zajednički za sve instrukcije. Prenose instrukcije iz programske memorije (eng. *Program Memory* – PM, u nastavku teksta često korištena skraćenica) u instrukcijski registar (faze dohvata instrukcije). Tijekom s3 dijela, instrukcija koja se tek pročita iz PM se dekodira, tj. dijelovima s4, s5, s6, s61, s51, s52, s53, s54 je „naloženo“ što trebaju raditi, preko dodijeljenih registara. U datom trenutku, dio protočne strukture može napraviti sljedeće:

- Izvršavati se normalno

  Registri u toj fazi se pune s:
  - Vrijednostima iz prethodne faze, ako se ta faza razlikuje od s1 ili s2 ili s3
  - Određenim vrijednostima ako je ta faza s1 ili s2 (te vrijednosti određuje upravitelj memorijom – eng. *Program Memory manager*, tkđ. pojam koji će se često upotrebljavati u nastavku teksta)
  - Vrijednostima iz instrukcijskog dekodera, ako se nalazimo u fazi s3
- flush

  Svi registri u toj fazi se resetiraju na nulu
- stall

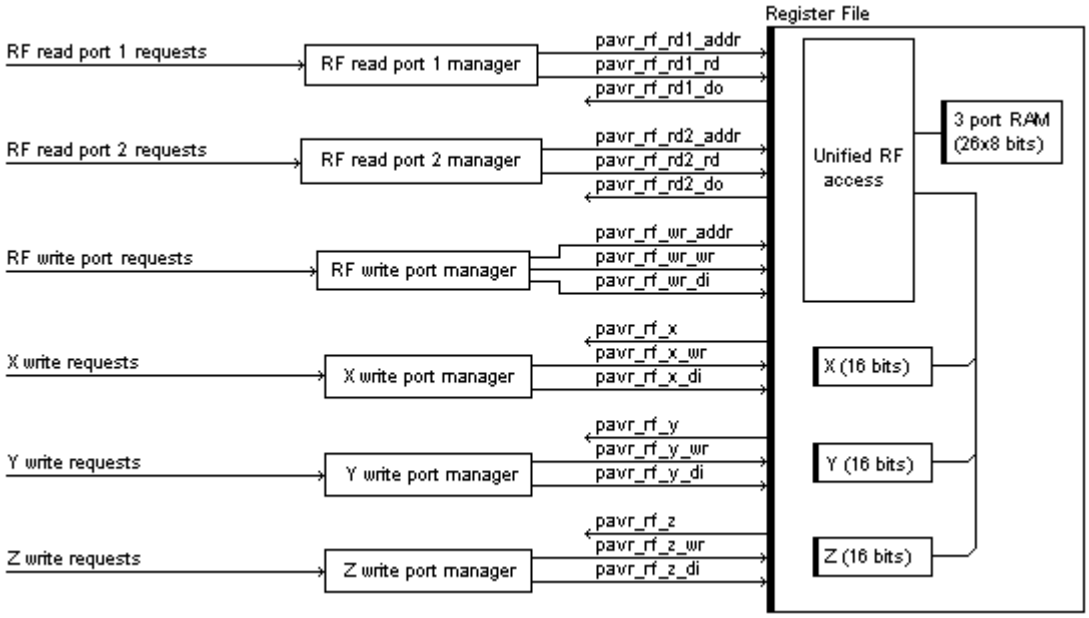
  Svi registri u toj fazi ostaju nepromijenjeni

3.1.2 Hardverski resursi

3.1.2.1 Podatkovni registar – register file – RF

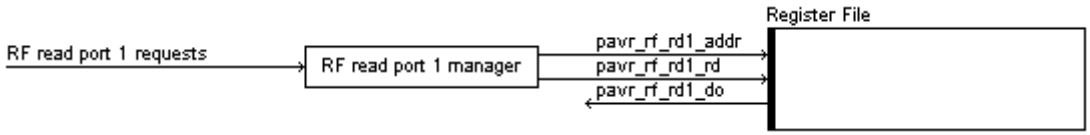
RF je memorija s 3 porta, od toga su 2 porta za čitanje i 1 za pisanje. Sadrži 32 lokacije, 8 bita svaka.

Omogućeni su odvojeni portovi za čitanje i pisanje za gornje tri 16 bitne riječi. Gornje tri 16 bitne riječi su pokazivački registri X (na adrese bajta 27:26), Y (29:28) i Z (31:30). RF je smješten na početak Unificiranog memorijskog prostora (eng. *Unified Memory space* – UM).



Slika 4. RF – shema

3.1.2.1.1 RF – povezivost porta 1 za čitanje



Slika 5. RF - smjer zahtjeva – port 1

Zahtjevi prema RF portu 1 za čitanje:

- Pavr_s3_rfrd_rq

  Većina ALU instrukcija treba čitati operand iz RF porta 1 za čitanje u istom periodu u kojem je instrukcija dekodirana.

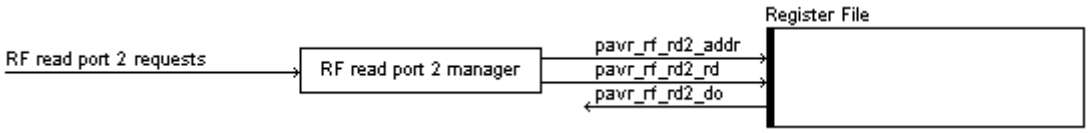
- Pavr_s5_dacu_rfrd1_rq

  To je tzv. „pogrešni“ zahtjev za RF port 1 za čitanje. Kako bi se zadržala kompatibilnost s AVR arhitekturom, operacije za učitavanje i pohranu (eng. *load* i *store*, često korišteni izrazi u daljnjem tekstu) moraju „operirati“ u UM-u, što uključuje RF, IOF (eng. input output file – ulazno izlazna (UI) datoteka) i DM (eng. Data memory – DM – podatkovna memorija). Stoga, moguće je za load da prenese primjerice podatak iz RF-a u RF, prije nego iz podatkovne memorije DM-a u RF (ovisno o uključenim adresama).

  Upravitelj jedinice za izračun adresa podataka (eng. Data Address Calculation Unit – DACU) donosi odluku koji će se fizički uređaj koristiti i smješta posljedične pozive prikladnom upravitelju (eng. manager) hardverskih resursa.

  Ista situacija je s „pogrešnim“ RF pisanjem u store operaciji. Store operacija čita i RF i piše u RF, IOF ili DM.

3.1.2.1.2 RF – povezivost porta 2 za čitanje



Slika 6. RF - smjer zahtjeva – port 2

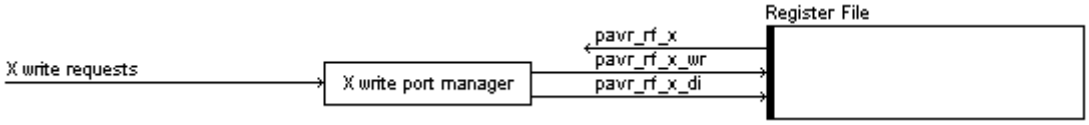
- Pavr_s3_rfrd2_rq
  Korišteno od strane instrukcija koje imaju 2 operanda (većina ALU instrukcija, premještanja itd.)

3.1.2.1.3 RF – port za pisanje

Zahtjevi za RF port za pisanje:

- Pavr_s6_aluoutlo8_rfwr_rq
  Zahtjev za pisanje nižih 8 bitova rezultata ALU operacije u RF
- Pavr_s61_aluouthi8_rfwr_rq
  Zahtjev za pisanje viših 8 bitova rezultata ALU operacije u RF
- Pavr_s6_iof_rfwr_rq
  Zahtjev za pisanje IOF podatkovnog izlaza u RF
  Potreban od strane IN, BLD.
- Pavr_s6_dacu_rfwr_rq
  Zahtjev za pisanje podatkovnog izlaza iz Unified Memory (DACU podatkovni izlaz) u RF
- Pavr_s6_pm_rfwr_rq
  Zahtjev za pisanje podatkovnog izlaza i PM-a u RF
  Potreban od strane LPM, ELPM.
- Pavr_s5_dacu_rfwr_rq
  Zahtjev za pisanje izlaza iz RF-a u RF.
  Potreban od strane store i PUSH naredbi.

3.1.2.1.4 Povezivost porta X



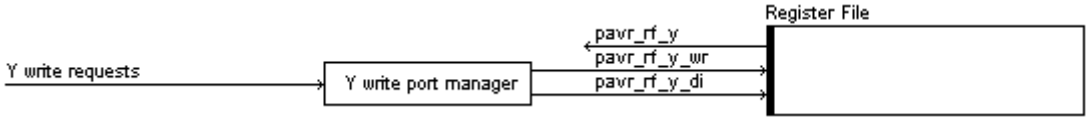
Slika 7. Smjer zahtjeva – port X

Ovo je port za čitanje i pisanje. Sadržaj registra X je stalno dostupan za čitanje, pod imenom pavr_rf_x. Port X za pisanje se sastoji od ulaznih podataka (pavr_rf_x_di) i „write strobe“ (pavr_rf_x_wr).

Zahtjevi za X port za pisanje:

- Pavr_s5_ldstincrampx_xwr_rq
  Povećava X. Ako kontroler ima više od 64 KB memorije, tada povećava RAMPX:X (24 bita) radije nego X (16 bita).
  Potreban od strane load i store naredbi s postinkrementiranjem (povećanjem poslije izvršavanja).
- Pavr_s5_ldstdecrampx_xwr_rq
  Smanjuje X. Ako kontroler ima više od 64 KB memorije, tada smanjuje RAMPX:X radije nego X.
  Potreban od strane load i store naredbi s predekrementiranjem (smanjenjem prije izvršavanja).

3.1.2.1.5 Povezivost porta Y



Slika 8. Smjer zahtjeva – port Y

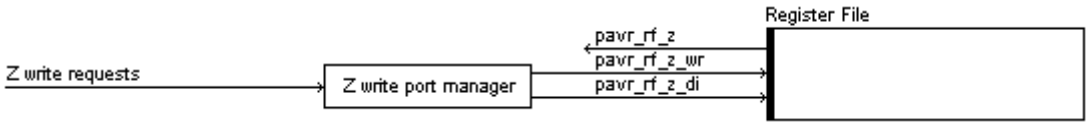
Port za čitanje i pisanje.

Zahtjevi za Y port za pisanje:

- Pavr_s5_ldstdecrampy_ywr_rq
  Povećava Y ili RAMPY:Y.
  Potreban od strane load i store naredbi s post-inkrementiranjem.

- Pavr_s5_ldstdecrampy_ywr_rq
  Smanjuje Y ili RAMPY:Y.
  Potreban od strane load i store naredbi s predekrementiranjem.

3.1.2.1.6 Povezivost porta Z



Slika 9. Smjer zahtjeva – port Z

Port za čitanje i pisanje.
Zahtjevi za Z port za pisanje:

- Pavr_s5_ldstincrampz_zwr_rq
  Povećava Z ili RAMPZ:Z.
  Potreban od strane load i store naredbi s post-inkrementiranjem.
- Pavr_s5_ldstdecrampz_zwr_rq
  Smanjuje Z ili RAMPZ:Z.
  Potreban od strane load i store naredbi s predekrementiranjem.
- Pavr_s5_lpminc_zwr_rq
  Povećava Z.
  Potreban od strane LPM naredbi s post-inkrementiranjem.
- Pavr_s5_elpmincrampz_zwr_rq
  Povećava RAMPZ:Z.
  Potreban od strane ELPM naredbi s post-inkrementiranjem.

3.1.2.2 Jedinica za premošćivanje – Bypass Unit – BPU

BPU je privremeni spremnik nalik na FIFO memoriju (eng. *first in first out* – prvi koji ulazi, prvi i izlazi), koji čuva podatke koji trebaju biti zapisani u RF. Ako instrukcija računa vrijednost koja se mora zapisati u RF (npr. ALU instrukcija) prvo se piše u BPU i onda (ili u isto vrijeme) piše u RF.

Ako naredne instrukcije trebaju operand iz RF-a, na istoj adresi gdje je trebao biti zapisan prethodni rezultat u RF, te će instrukcije čitati operand iz BPU-a prije nego iz RF-a. Na taj način se izbjegava hazard „čitanja prije pisanja“.

Specifične situacije gdje je BPU potreban:

- Tijekom čitanja operanada RF-a
  Čitanje operanada RF-a se vrši kroz BPU
- Tijekom čitanja pokazivačkih registara
  Čitanje pokazivačkih registara se vrši preko BPU-a

Algoritam korištenja BPU-a:

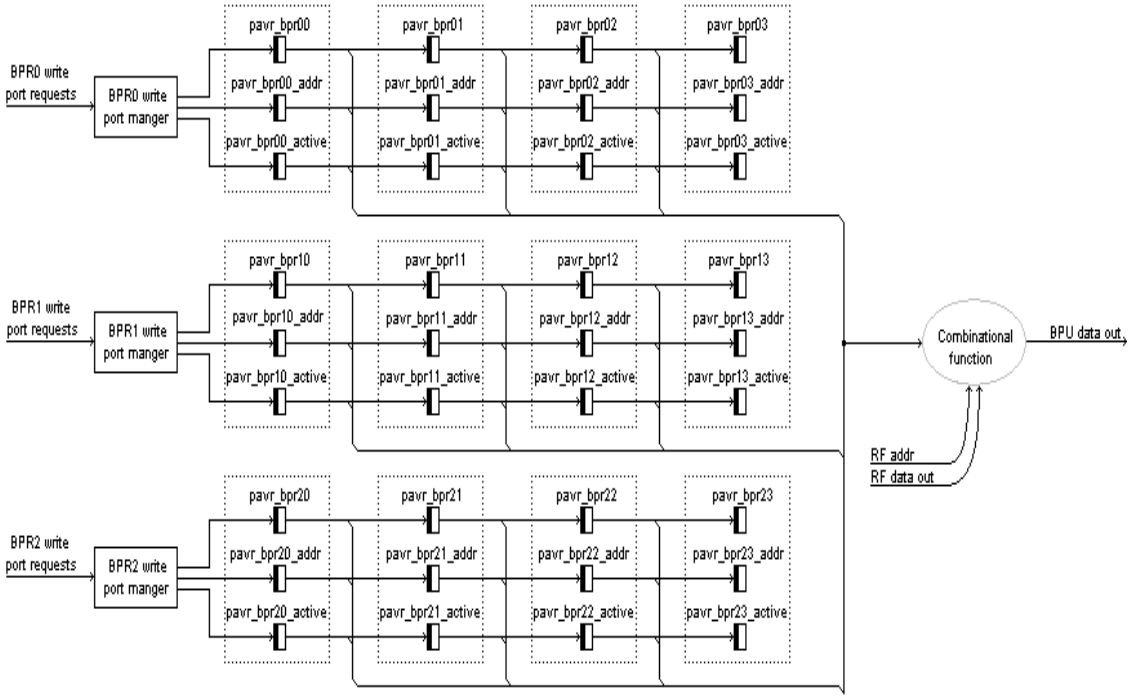
- Instrukcija koja želi pisati rezultat u RF, prvo piše u BPU u 3 podatkovna polja:
  - Sami rezultat
  - Adresu rezultata u RF
  - Zastavicu koja označava da taj BPU ulaz sadrži valjani podatak (tzv. „aktivna“ zastavica)
- Sljedeće instrukcije koje trebaju operand iz RF-a, čitaju ga kroz dodijeljenu funkciju (kombinacijska logika) koja radi sljedeće:
  - Provjerava sve BPU ulaze i gleda koji su aktivni (koji drže značajne podatke)
  - Uspoređuje adrese operanada s adresama u svim aktivnim BPU ulazima
  - Ako se samo jedna adresa poklapa, dohvaća podatak iz tog BPU ulaza radije nego podatak iz RF-a
  - Ako se više adresa poklapa, dohvaća podatak iz „najnovijeg“ BPU ulaza. Iako je moguće da se dogode dva simultana BPU ulaza, takva situacija se ne bi smjela nikad dogoditi; označavala bi dizajnersku grešku (bug). Takva situacija će javiti pogrešku tijekom simulacije.
  - Ako se nijedna adresa ne poklapa, dohvaća podatak iz RF-a (kao da BPU ne postoji).

Maksimalno kašnjenje između pisanje i čitanja iz RF-a je 4 perioda takta. Stoga, BPU FIFO-nalik struktura ima dubinu 4. S druge strane, BPU mora biti u

mogućnosti pisati 3 jednobajtna operanda u isto vrijeme (mora imati 3 porta za pisanje). Većina BPU instrukcija su store naredbe s pre(post) de(in)krementiranjem. Jednobajtni podatak i dvobajtni pokazivački registar moraju biti zapisani u BPU i u RF. Tri bajta su simultano zapisana u BPU lance (eng. chains, u nastavku korišten izraz) ili BPU registre (BPU chains 0, 1, 2; ili BPU registri 0, 1, 2; ili BPR0, BPR1, BPR2).

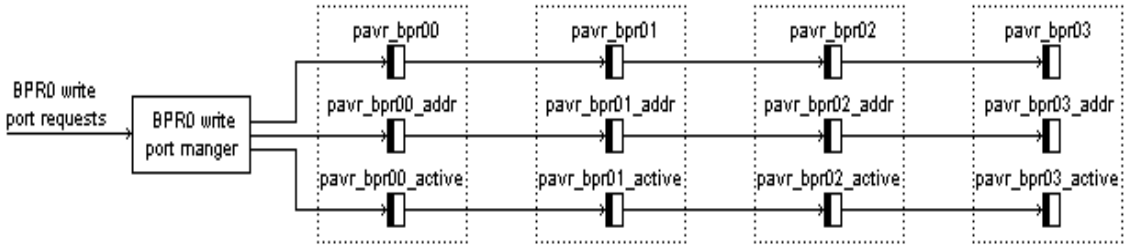
BPU ima 3 x 4 ulaza, svaki se sastoji od:

- 8 bitnog podatkovnog polja
- 5 bitnog adresnog polja
- Zastavice koja označava ulaz kao aktivan ili neaktivan



Slika 10. BPU - shema

3.1.2.2.1 Bypass chain 0

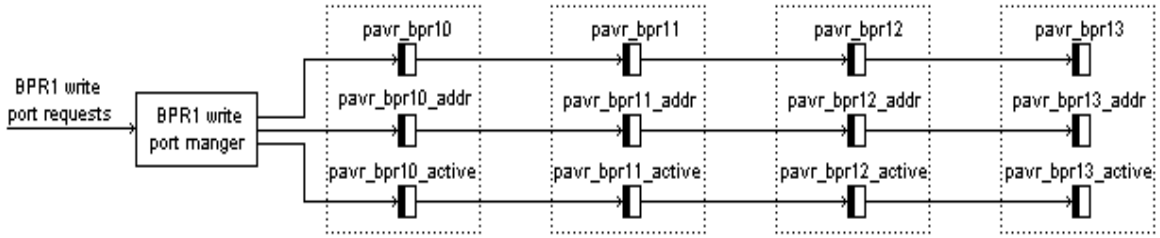


Slika 11. BPU lanac/registar 0

Zahtjevi za BPR0 port za pisanje:

- Pavr_s5_alu_bpr0wr_rq
  Potreban od strane ALU instrukcija
- Pavr_s6_iof_bpr0wr_rq
  Potreban od strane instrukcija koje čitaju UI datoteku – IOF (IN, BLD)
- Pavr_s6_daculd_bpr0wr_rq
  Potreban od strane load instrukcija
- Pavr_s5_dacust_bpr0wr_rq
  Potreban od strane store instrukcija.
- Pavr_s5_pmdo_bpr0wr_rq
  Potreban od strane LPM instrukcija.

3.1.2.2.2 Bypass chain 1



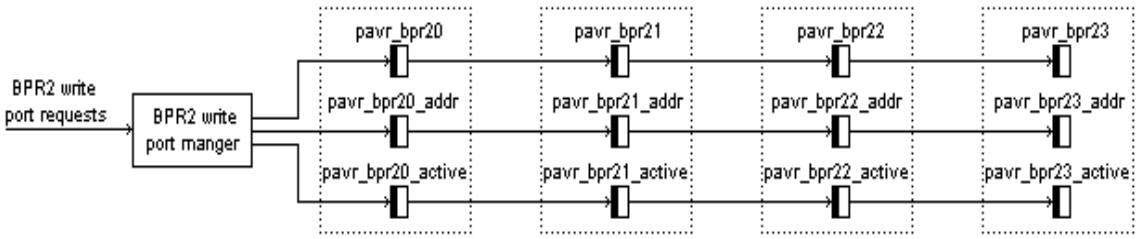
Slika 12. BPU lanac/registar 1

Zahtjevi za BPR1 portom za pisanje:

- Pavr_s5_alu_bpr1wr_rq
  Potreban od strane ALU instrukcija sa 16 bitnim rezultatom (ADIW, SBIW, MUL, MULS, MULSU, FMUL, FMULS, FMULSU, MOVW)

- Pavr_s5_dacux_bpr12wr_rq
  Potreban od strane load i store instrkcija s pre(post) in(de)krementiranjem
  Niži bajt pokazivača X će biti zapisan u BPR1
- Pavr_s5_dacuy_bpr12wr_rq
  Potreban od strane load i store instrukcija s pre(post) in(de)krementiranjem
  Niži bajt pokazivača Y će biti zapisan u BPR1
- Pavr_s5_dacuz_bpr12wr_rq
  Potreban od strane load i store instrukcija s pre(post) in(de)krementiranjem
  Niži bajt pokazivača Z će biti zapisan u BPR1

3.1.2.2.3 Bypass chain 2



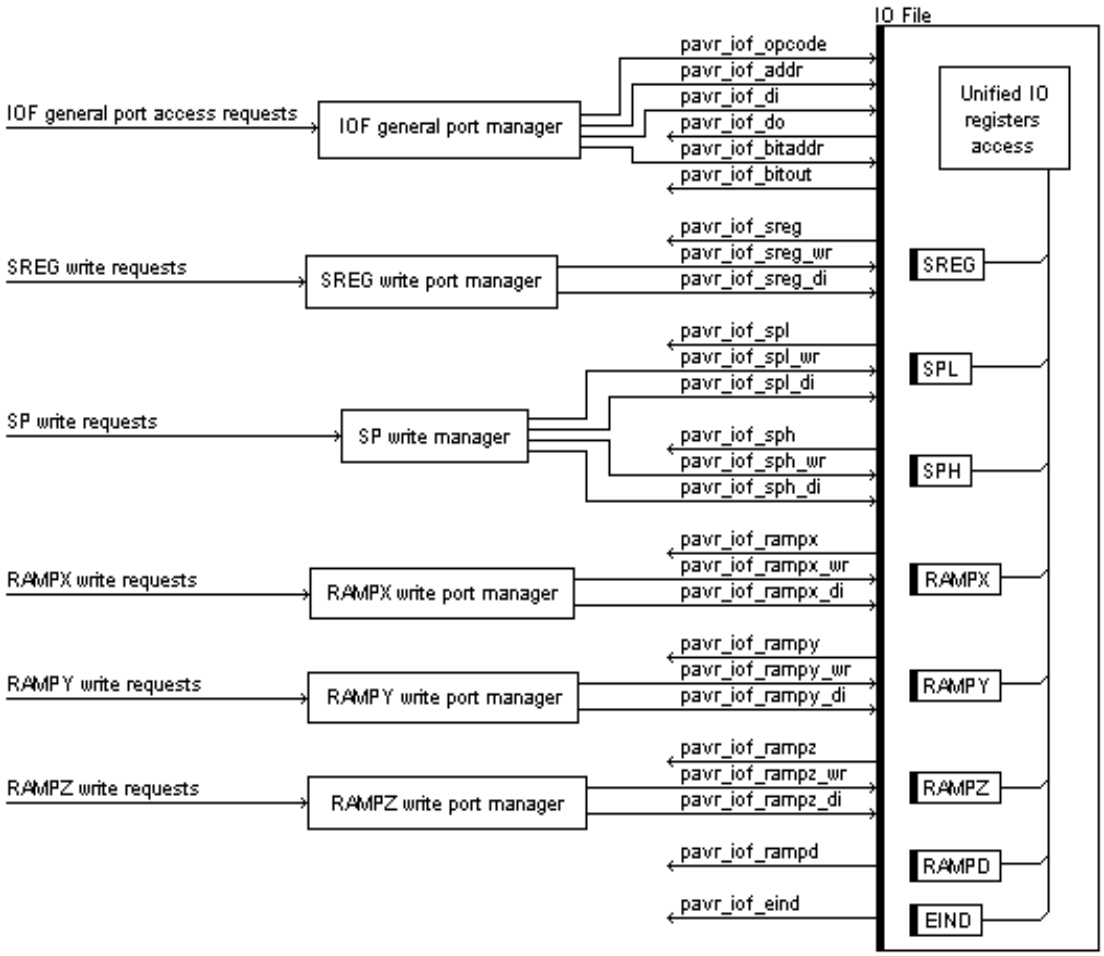
Slika 13. BPU lanac/registar 2

Zahtjevi za BPR2 portom za pisanje:

- Pavr_s5_dacux_bpr12wr_rq
  Potreban od strane load i store instrukcija s pre(post) in(de)krementiranjem.
  Viši bajt pokazivača X će biti zapisan u BPR2
- Pavr_s5_dacuy_bpr12wr_rq
  Potreban od strane load i store instrukcija s pre(post) in(de)krementiranjem
  Viši bajt pokazivača Y će biti zapisan u BPR2
- Pavr_s5_dacuz_bpr12wr_rq
  Potreban od strane load i store instrukcija s pre(post) in(de)krementiranjem
  Viši bajt pokazivača Z će biti zapisan u BPR2

3.1.2.3 UI datoteka – IO File – IOF

IOF se sastoji od skupa diskretnih registara, grupiranih u entitet nalik memoriji. Sadrži bajtno-orijentirani port za čitanje/pisanje te odvojene portove za čitanje i pisanje za svaki registar u IOF.

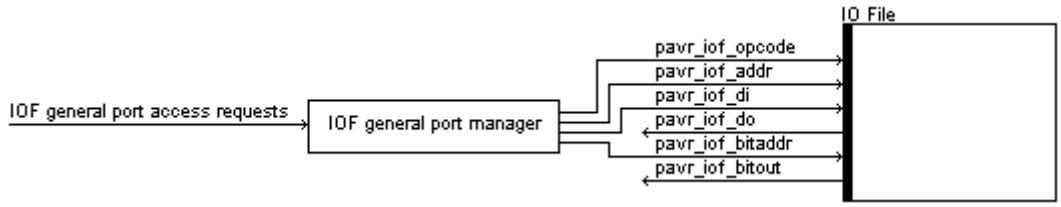


Slika 14. IOF - shema

Svakom IOF registru je dodijeljena jedinstvena adresa u UI prostoru. Ta adresa je definirana u file-u gdje su definirane sve konstante (pavr_constants.vhd).

UI prostor je smješten u unificiranoj memoriji (eng. *Unified Memory* – UM) iznad RF-a, tj. počinje s adresom 32. UI adresni prostor je u rangu od 0...63, tj. od adresa u Unified memoriji 32...95. Nedefinirani UI registri će pročitati nedefiniranu vrijednost.

3.1.2.3.1 UI port – IO port



Slika 15. UI port - shema

UI port čita bajtove UI registara i šalje ih na izlaz te piše bajtove s ulaza u UI registre. Također, može i obrađivati bitove: učitava bitove (iz T zastavice u SREG-u na izlaz), sprema bitove (iz ulaza u T bit u SREG-u), postavlja UI bitove, briše UI bitove.

Instrukcija se mora navesti kako bi se odredila jedna od akcija koje port radi.

Implementirani su sljedeće instrukcije za UI port:

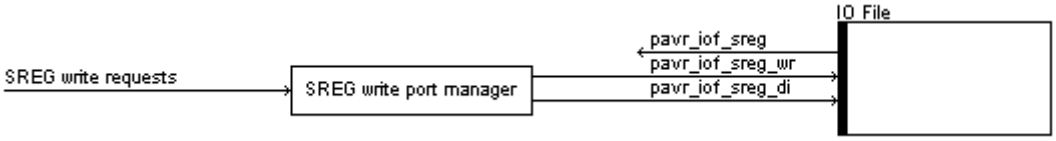
- Bajt za čitanje – potreban od instrukcija – IN, SBIC, SBIS
- Bajt za pisanje – write byte – OUT
- Bit za brisanje – clear bit – CBI
- Bit za postavljanje – set bit – SBI
- Bit za učitavanje – load bit – BLD
- Bit za spremanje – store bit – BST

Zahtjevi za ovim portom:

- Pavr_s5_iof_rq
  Potreban od strane instrukcija koje manipuliraju IOF-om u fazi s5: CBI, SBI, SBIC, SBIS, BSET, BCLR, IN, OUT, BLD, BST
- Pavr_s6_iof_rq
  Potreban od strane instrukcija koje manipuliraju IOF-om u fazi s6: CBI, SBI, BSET, BCLR
- Pavr_s5_dacu_rq

Potreban od strane instrukcija load i store koje su dekodirane od strane DACU jedinice pri pristupu IOF-u

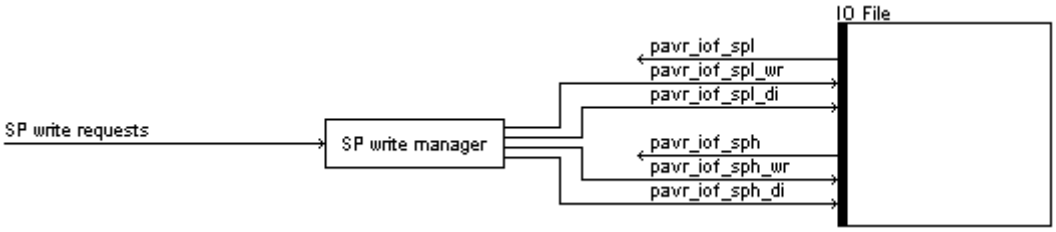
3.1.2.3.2 SREG port



Zahtjevi prema SREG portu:

- Pavr_s5_alu_sregwr_rq
  Signalizira kako instrukcija koja koristi ALU želi osvježiti aritmetičke zastavice. Zastavice I (opći prekid omogućen, SREG(7)) i T (eng. *transfer bit* – bit za prijenos, SREG(6)) ostaju nepromijenjeni.
- Pavr_s5_setiflag_sregwr_rq
  Postavlja I zastavicu. Samo RETI instrukcija to zahtjeva.
- Pavr_s5_clriflag_sregwr_rq
  Briše I zastavicu. Nijedna instrukcija eksplicitno to ne zahtjeva. Potrebno je samo kada je prekid odobren (tijekom posljedičnog implicitnog poziva (CALL) objašnjenog kasnije)

3.1.2.3.3 Pokazivač stoga – stack pointer – SP port



Slika 16. Stog - shema

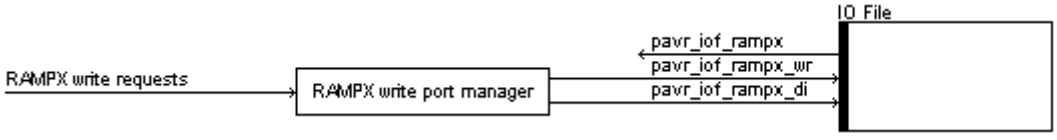
Pokazivač na stog, širine 16 bitova, sastavljen je od dva 8 bitna registra, SPL i SPH. Stog se može smjestiti bilo gdje u prostoru Unified memorije, tj. bilo gdje u RF, IOF ili DM-u. Može čak započeti u primjerice RF-u i nastaviti se u IOF-u. Međutim, smještati pokazivač na stog u IOF će izazvati pogrešku prilikom

programiranja, s obzirom da IOF registri imaju dodijeljene funkcije. Nasumične vrijednosti sa stoga upisane u IOF mogu rezultirati primjerice, nepredvidljivim okidanjem bilo kojeg prekida i općenito nepredvidljivim ponašanjem kontrolera.

Zahtjevi prema SP portu:

- Pavr_s5_inc_spwr_rq
  Povećava SP (SPH i SPL) za 1. Potreban od strane POP naredbe.
- Pavr_s5_dec_spwr_rq
  Povećava SP za 1. Potreban od strane PUSH naredbe.
- Pavr_s5_inc_calldec_rq
  Smanjuje SP za 1. Potreban od strane RCALL, ICALL, EICALL, CALL naredbi. Implicitnog poziva prekida – CALL.
- Pavr_s51_calldec_spwr_rq
  Smanjuje SP za 1. Potreban od strane RCALL, ICALL, EICALL, CALL naredbi. Implicitnog poziva prekida – CALL.
- Pavr_s52_calldec_spwr_rq
  Smanjuje SP za 1. Potreban od strane RCALL, ICALL, EICALL, CALL naredbi. Implicitnog poziva prekida – CALL.
- Pavr_s5_retinc_spwr_rq
  Povećaj SP za 2. Potreban od strane RET, RETI naredbi.
- Pavr_s51_retinc_spwr_rq
  Povećaj SP za 1. Potreban od strane RET, RETI naredbi.

3.1.2.3.4 RAMPX port



Slika 17. RAMPX port - shema

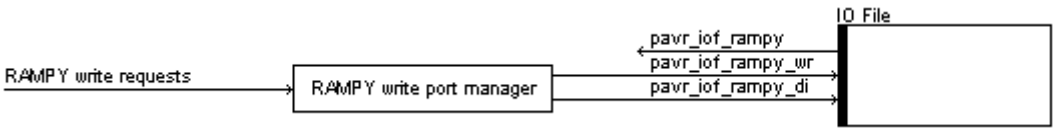
Zahtjevi prema RAMPX portu:

- Pavr_s5_ldstincrampx_xwr_rq

Potreban od strane load i store naredbi s postinkrementiranjem. RAMPX modificirati ako kontroler ima više od 64 KB DM-a.

- Pavr_s5_ldstdecrampx_xwr_rq
  Potreban od strane load i store naredbi s predekrementiranjem. RAMPX modificirati ako kontroler ima više od 64 KB DM-a.

3.1.2.3.5 RAMPY port

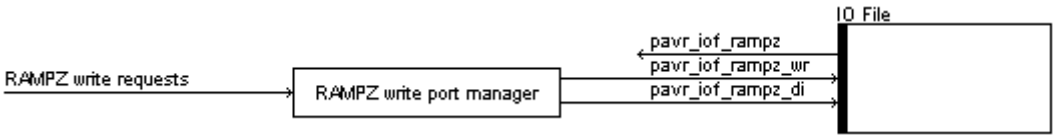


Slika 18. RAMPY port - shema

Zahtjevi prema RAMPY portu:

- Pavr_s5_ldstincrampy_xwr_rq
  Potreban od strane load i store naredbi s postinkrementiranjem. RAMPY modificirati ako kontroler ima više od 64 KB DM-a.
- Pavr_s5_ldstdecrampy_xwr_rq
  Potreban od strane load i store naredbi s predekrementiranjem. RAMPY modificirati ako kontroler ima više od 64 KB DM-a.

3.1.2.3.6 RAMPZ port



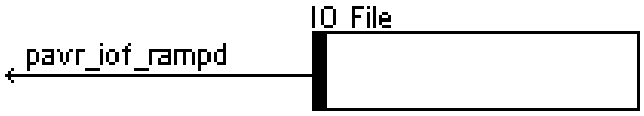
Slika 19. RAMPZ port - shema

Zahtjevi prema RAMPZ portu:

- Pavr_s5_ldstincrampz_xwr_rq
  Potreban od strane load i store naredbi s postinkrementiranjem. RAMPZ modificirati ako kontroler ima više od 64 KB DM-a.

- Pavr_s5_ldstdecrampz_xwr_rq
  Potreban od strane load i store naredbi s predekrementiranjem. RAMPZ modificirati ako kontroler ima više od 64 KB DM-a.

3.1.2.3.7 RAMPD port

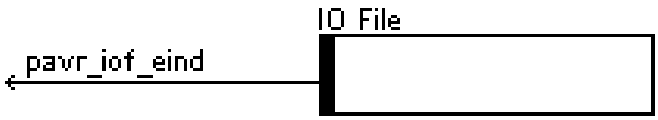


Slika 20. RAMPD port - shema

RAMPD je jednostavni port samo za čitanje. Koristi se u kontrolerima s više od 64 KB podatkovne memorije, kako bi se pristupilo cijelom prostoru podatkovne memorije. Korišten je od strane instrukcija LDS (eng. ***L****oa****D*** *direct from data* ***S****pace* – učitaj izravno iz podatkovnog prostora) i STS (eng. STore direct to data Space – pohrani izravno u podatkovni prostor). Kako bi se dobile željene adrese u prostoru podatkovne memorije, te instrukcije nadograđuju RAMPD sa 16 bitnom konstantom iz instrukcijske riječi – RAMPD:k16.

U kontrolerima s manje od 64 KB podatkovne memorije, ovaj se registar ne koristi. U RAMPD registar se može pisati jedino preko IOF porta za čitanje i pisanje. Nijedna instrukcija eksplicitno ne zahtjeva pisanje u RAMPD registar.

3.1.2.3.8 EIND port



Slika 21. EIND port - shema

EIND je jednostavni port samo za čitanje. Koristi se u kontrolerima s više od 64 KB podatkovne memorije, kako bi se pristupilo cijelom prostoru podatkovne memorije. EIND je korišten od strane instrukcija EICALL (eng. Extended Indirect CALL – prošireni neizravni poziv) i EIJMP (eng. Extended Indirect JuMP –

proširenи neizravni/indirektni skok). Kako bi se dobile željene adrese u prostoru podatkovne memorije, te instrukcije nadovezuju EIND sa Z registrom – EIND:Z.

3.1.2.3.9 Periferije – Port A, vanjski prekid 0, Timer 0

Periferijama je dana sekundarna važnost, no UI port, vanjski prekid i 8 bitni timer su implementirani kao bi se testirao prekidni sustav. Periferije su dizajnirane kako bi funkcionirale odvojeno i kako bi se lako nadograđivale.

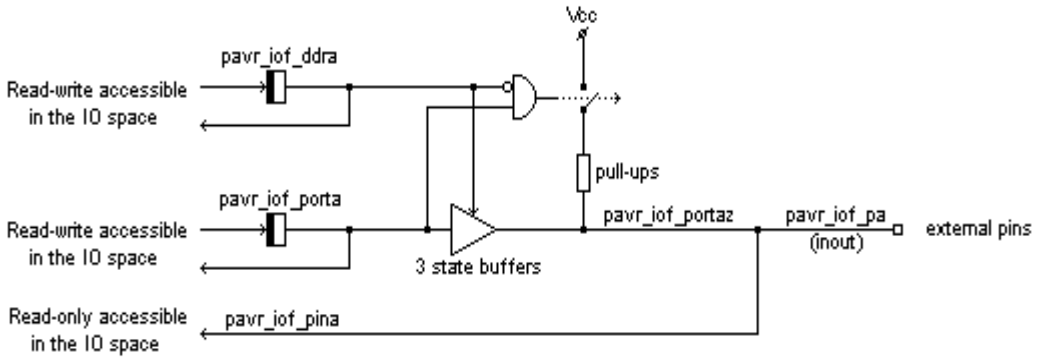
Port A sadrži 8 dvosmjernih UI linija opće namjene. Linije 0 i 1 sadrže također i alternativne funkcije:

- Linija 0 se može koristiti kao ulaz za vanjski prekid 0
- Linija 1 se može koristiti kao ulaz za timer 0

Portom A se upravlja kroz 3 IOF lokacije: PORTA, DDRA i PINA. DDRA postavlja smjer svakog priključka. DDRA(i)=0 znači da je linija i ulaz, DDRA(i)=1 znači da je linija i izlaz. Kada se upisuje vrijednost na port, ta vrijednost ide u PORTA. Ako DDRA postavi odgovarajuće linije kao izlaze, sadržaj PORTA će biti dostupan na izlaznim priključcima. Međutim, ako DDRA postavi linije kao ulaze (DDRA(i)=0), onda:

- Ako je PORTA(i)=0, linija i je „čisti“ ulaz (visoka impedancija – eng. High Z)
- Ako je PORTA(i)=1, linija i je ulaz slabe visoke razine

PINA čita fizičke vrijednosti vanjskih linija, i to prije nego PORTA.



Slika 22. Port A - shema

Vanjski prekid 0 je fizički mapiran na liniji 0 (bit 0) porta A. Vezana prekidna zastavica se nalazi u IOF registru GIFRP (eng. *General Interrupt Flags Register* – registar za opće prekidne zastavice).

GIFR

	INTF0						

Slika 23. GIFR registar

Vanjski prekid 0 je omogućen/onemogućen postavljajući/brišući bit 6 u GIMSK (eng. *General Interrupt Mask* – opća maska za prekide) registru.

GIMSK

	INT0						

Slika 24. GIMSK registar

Ako je omogućen, može okinuti prekid na tranziciji s visoke na nisku razinu i obratno te na nisku razinu ulaza prekida 0. To ponašnje je definirano s 2 bita u MCUCR (eng. *microcontroller control* – kontrola mikrokontrolera) registru:

MCUCR

						ISC01	ISC00

ISC00	ISC01	Behavior
0	0	The low level of INT0 generates an interrupt request.
0	1	Undefined.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Timer 0 je IOF registar pod imenom TCNT0. Ponašanje mu je definirano preko skupa IOF registara:

- TIFR (eng. *Timer Interrupt Flag register* – registar sa zastavicom prekida za timer) sadrži Timer 0 prekidnu zastavicu:

  TIFR

						TOV0	

  Slika 25. TIFR registar

- TIMSK (eng. *Timer Interrupt Mask* – maska za prekida za timer) sadrži zastavicu koja omogućava/onemogućava Timer 0 prekid:

  TIMSK

						TOIE0	

  Slika 26. TIMSK registar

- TCCR0 (eng. *Timer 0 Control Register* – kontrolni registar) definira izvor skaliranja za Timer 0. Kada je odabran vanjski ulazni priključak, izvor takta za Timer 0 će biti linija 0 porta A:

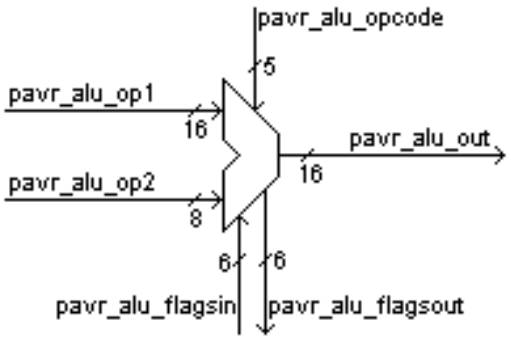
TCCR0

					CS02	CS01	CS00

CS02	CS01	CS00	Prescaler
0	0	0	Timer 0 is stopped.
0	0	1	Clk
0	1	0	Clk/8
1	0	0	Clk/64
1	0	0	Clk/256
1	0	1	Clk/1024
1	1	0	External pin T0, falling edge
1	1	1	External pin T0, rising edge

Slika 27. TCCR0 registar

3.1.2.4 Aritmetičko logička jedinica – arithmetical & logical unit – ALU



Slika 28. ALU - shema

ALU jedinica je 100% kombinacijski dio. Prihvaća 2 operanda:

- 16 bitni operand – kroz BPU
- 8 bitni operand – kroz BPU

ALU izlaz je širine 16 bitova.

ALU instrukcije:

- NOP
- OP1 – prenosi operand 1 izravno na ALU izlaz
  OP2 – prenosi operand 2 izravno na nižih 8 bita ALU izlaza
- ADD8
  ADC8 – zbraja s prijenos nižih 8 bita operanda 1 s operandom 2

SUB8

SBC8

- AND8, EOR8, OR8
- INC8, DEC8
- COM8, NEG8, SWAP8
- LSR8, ASR8. ROR8
- ADD16 – prošireno zbrajanje s predznakom bez prijenosa operanda 1 s operandom 2 na 16 bitova

  SUB16
- MUL8, MULS8, MULSU8, FMUL8, FMULS8, FMULSU8

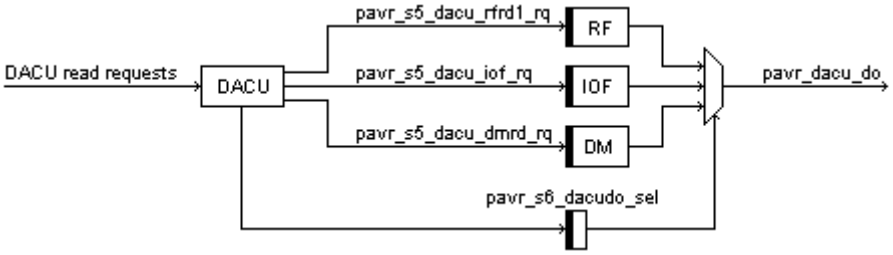
ALU zastavice:

- H – half carry – poluprijenos
- S – sign – predznak
- V – dvojni komplement
- N – negative
- Z – zero – nula
- C – carry – prijenos

3.1.2.5 DACU

Data Address Calculation Unit – jedinica za računanje adresa podataka obrađuje pristupe čitanja i pisanja nad spojenim RF, IOF i DM prostorom, tj. preko prostora Unified memorije (UM).

Load i store naredbe rade u UM prostoru. Koriste DACU kako bi translatirale adrese iz UM-a u RF, IOF ili DM adrese. DACU zaprima zahtjeve za čitanje i pisanje u UM prostor, translatira UM adrese u RF, IOF ili DM adrese i transparentno smješta zahtjeve za čitanje ili pisanje specifičnog hardverskog resursa (RF, IOF ili DM) koji odgovaraju danoj UM adresi.



Slika 29. DACU – shema čitanja

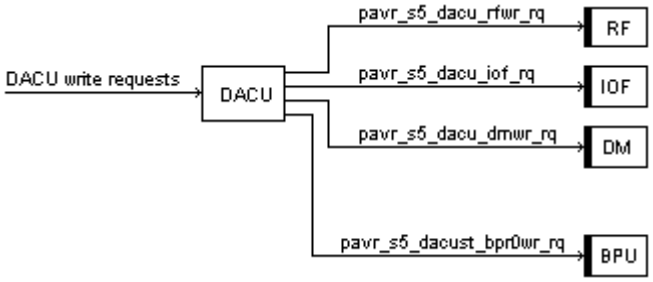
DACU zahtjevi za čitanje:

- Pavr_s5_x_dacurd_rq
  Potreban od strane load naredbi kojima adresu daje X pokazivački registar
- Pavr_s5_y_dacurd_rq
  Potreban od strane load naredbi kojima adresu daje Y pokazivački registar
- Pavr_s5_z_dacurd_rq
  Potreban od strane load naredbi kojima adresu daje Z pokazivački registar
- Pavr_s5_sp_dacurd_rq
  Potreban od strane POP (naredba za uzimanje sa stoga) naredbe.
- Pavr_s5_k16_dacurd_rq
  Potreban od strane LDS naredbe.
  Ako kontroler ima više od 64 KB DM-a, UM adresa je sastavljena od RAMPD s nadovezanom 16 bitnom konstantom.
- Pavr_s5_pchi8_dacurd_rq
  Viših 8 bitova PC-a se učitavaju sa stoga. Potreban od strane RET i RETI instrukcija.
- Pavr_s51_pcmid8_dacurd_rq
  Srednjih 8 bitova PC-a se učitavaju sa stoga. Potreban od strane RET i RETI instrukcija.
- Pavr_s52_pclo8_dacurd_rq
  Nižih 8 bitova PC-a se učitavaju sa stoga. Potreban od strane RET i RETI instrukcija.

Kao odgovor na zahtjeve za čitanje, DACU „upućuje“ zahtjeve RF-u, IOF-u ili DM-u:

- pavr_s5_dacu_rfrd1_rq
- pavr_s5_dacu_iof_rq

- pavr_s5_dacu_dmrd_rq



Slika 30. DACU – shema pisanja

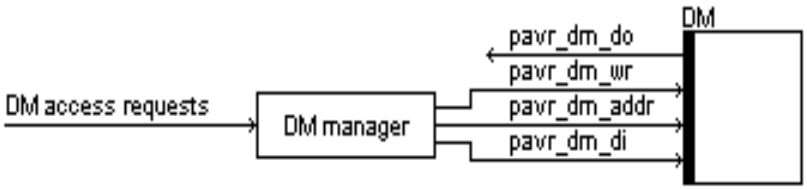
DACU zahtjevi za pisanje:

- Pavr_s5_x_dacuwr_rq
  Potreban od strane store naredbi kojima adresu daje X pokazivački registar
- Pavr_s5_y_dacuwr_rq
  Potreban od strane store naredbi kojima adresu daje Y pokazivački registar
- Pavr_s5_z_dacuwr_rq
  Potreban od strane store naredbi kojima adresu daje Z pokazivački registar
- Pavr_s5_sp_dacuwr_rq
  Potreban od strane PUSH (naredba za stavljanje na stog) naredbe.
- Pavr_s5_k16_dacuwr_rq
  Potreban od strane STS naredbe.
  Ako kontroler ima više od 64 KB DM-a, UM adresa je sastavljena od RAMPD s nadovezanom 16 bitnom konstantom.
- Pavr_s52_pchi8_dacuwr_rq
  Viših 8 bitova PC-a se pohranjuju na stog. Potreban od strane CALL instrukcija (CALL, RCALL, ICALL, EICALL, CALL implicitnog prekida).
- Pavr_s51_pcmid8_dacuwr_rq
  Srednjih 8 bitova PC-a se pohranjuju na stog. Potreban od strane CALL instrukcija.
- Pavr_s5_pclo8_dacuwr_rq
  Nižih 8 bitova PC-a se pohranjuju na stog. Potreban od strane CALL instrukcija.

Kao odgovor na zahtjeve za pisanje, DACU „upućuje“ zahtjeve RF-u, IOF-u ili DM-u te BPU:

- pavr_s5_dacu_rfwr_rq
- pavr_s5_dacu_iof_rq
- pavr_s5_dacu_dmwr_rq
- pavr_s5_dacust_bpr0wr_rq

3.1.2.6 Podatkovna memorija – data memory – DM



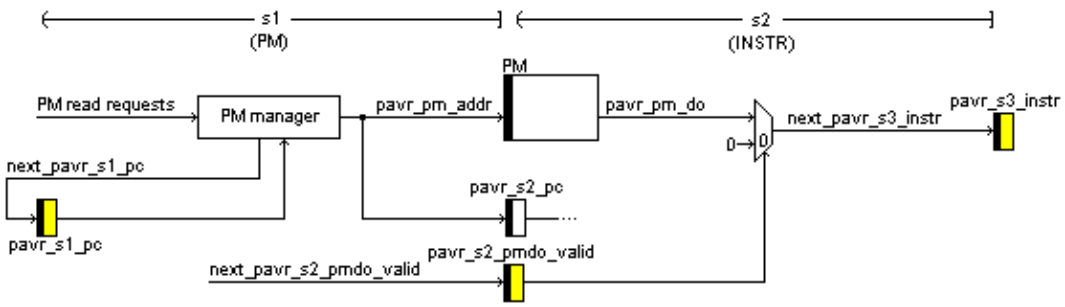
Slika 31. DM – shema

Podatkovna memorija ili DM je RAM s jednim portom koji omogućava pristup DM-u za čitanje i pisanje. Organizirana je po bajtovima, a veličina joj je namještena preko konstante definirane u datoteci pavr-costants.vhd.

Zahtjevi za pristup DM-u dolaze samo iz DACU:

- pavr_s5_dacu_dmrd_rq
- pavr_s5_dacu_dmwr_rq

3.1.2.7 Programska memorija – program memory – PM



Slika 32. PM – shema

Programska memorija (PM) je RAM s jednim portom koji omogućava samo čitanje (eng. *read-only*). Organizirana je po 16 bitnim riječima, a veličina joj je namještena preko konstante definirane u datoteci pavr-costants.vhd.

Osim kontroliranja PM-a PM manager (upravitelj) također kontrolira i programsko brojilo (eng. program counter). Neki PM pristupi trebaju modificirati PC, dok neki ne – load naredbe iz PM-a (LPM i ELPM instrukcije). Ostali zahtjevi odgovaraju instrukcijama koje trebaju modificirati tijek instrukcija (skokovi, grananja, pozivi i povrati).

U određenom momentu, protočna struktura može proslijediti više od jedne instrukcije. Do 6 instrukcija može biti istovremeno obrađeno. Iz toga je očigledno da svaka od tih instrukcija ima vlastitu adresu u PM-u. Na to se postavlja pitanje: kako je definirano programsko brojilo (eng. *program counter* - PC), ako se istovremeno izvršavaju dvije ili više instrukcija? Na koje adrese pokazuje PC?

Jednostavno, ne postoji jedinstveni PC u datom trenutku. PC se sastoji od skupa registara. Svaka instrukcija u protočnoj strukturi ima vezani PC koji ju „slijedi“ dok prolazi kroz protočnu strukturu. Implementacija je dalje razjašnjena u poglavlju vezanom za detaljniji opis protočne strukture i vezu s hardverskim resursima.

Primjerice, kada se izračuna adresa za relativni skok, uzima se u obzir „vlastiti“ PC prije nego adresa instrukcije dohvaćene tog momenta iz PM-a. Instrukcije koje mijenjaju redoslijed programskog tijeka (skokovi, grananja, preskoci, pozivi i povrati) moraju biti u mogućnosti manipuliranja PC-ovima vezanima s fazama s1, s2 i s3. Međutim, to nije napravljeno direktno nego preko Program Memory managera. PM manager centralizira sve zahtjeve za promjene tijeka izvršavanja instrukcija (zahtjevi za skok, grananje itd.) i brine o organizaciji PC-a.

Zahtjevi prema PM-u:

- pavr_s5_lpm_pm_rq
  Potreban od strane LPM naredbe.
  Ovaj zahtjev ne mijenja tijek izvršavanja instrukcija.
- pavr_s5_elpm_pm_rq
  Potreban od strane ELPM naredbe.
  Ovaj zahtjev ne mijenja tijek izvršavanja instrukcija.

- pavr_s4_z_pm_rq
  Potreban od strane ICALL i IJMP naredbe.
- pavr_s4_zeind_pm_rq
  Potreban od strane EICALL i EIJMP naredbe.
- pavr_s4_k22abs_pm_rq
  Potreban od strane CALL i JMP naredbe.
  Kako bi se dobila adresa skoka, na 16 bitnu instrukcijsku konstantu se nadovezuje 6 bitna prethodno pročitana konstanta iz instrukcije
- pavr_s4_k12rel_pm_rq
  Potreban od strane RCALL i RJMP naredbe.
  Primijetiti kako je pavr_s4_pc registar u protočnoj strukturi koji drži adresu instrukcije iz PM koja se izvršava u fazi s4. Zbog relativnog skoka koji se dogodi u s4, pavr_s4_pc je potrebniji nego trenutni PC (pavr_pc)
- pavr_s6_branch_pm_rq
  Potreban od strane naredbi za grananje – BRBC i BRBS.
- pavr_s6_skip_pm_rq
  Potreban od strane određenih naredbi za skok – CPSE, SBRC i SBRS.
- pavr_s61_skip_pm_rq
  Potreban od strane određenih naredbi za skok – SBIC i SBIS.
- pavr_s4_k22int_pm_rq
  Potreban od strane naredbe za implicitni prekid – CALL.
- pavr_s54_ret_pm_rq
  Potreban od strane naredbi RET i RETI

3.1.3 Upravljanje hardverskim resursima

Dijelovi protočne strukture mogu zatražiti pristup hardverskim resursima. Pristup hardverskom izvoru se vrši putem dodijeljenih upravitelja (eng. *managera*) po principu jedan upravitelj po hardverskom resursu, tj. jedan VHDL proces po upravitelju.

Glavni hardverski resursi:

- Registarska datoteka (eng. *Register File* –RF)
- Jedinica za premošćivanje (eng. Bypass Unit – BPU)
  - Bypass Registar 0 – Bypass lanac – chain 0 – BPR0
  - Bypass Registar 1 – Bypass lanac – chain 1 – BPR1
  - Bypass Registar 2 – Bypass lanac – chain 2 – BPR2
- U/I datoteka – IO (eng. *input output* file IOF)
- Statusni registar – SREG
- Pokazivač na stog (eng. *Stack pointer* – SP)
- Aritmetičko logička jedinica (eng. *Arithmetical and logical unit* – ALU)
- Jedinica za kontrolu pristupa podatcima (eng. Data Access Control Unit – DACU)
- Programska memorija (eng. *program memory* – PM)
- Jedinica za zadržavanje i ispuštanje (eng. *Stall* i *Flush unit* – SFU)

Samo jedan takav zahtjev može biti zaprimljen od datog izvora u vremenu. Ako postoji više zahtjeva prema resursu, upravitelj pristupa će javiti grešku tijekom simulacije – indikacija dizajnerskog bug-a. Protočna struktura je konstruirana tako da se može pristupiti svakom resursu tijekom fiksne faze protočne strukture:

- RF se normalno čita u s3 i u njega se piše tijekom s6
- IOF se normalno čita/piše u s5
- DM se normalno čita/piše u s5
- DACU se normalno čita/piše u s5
- PM se normalno čita u s1

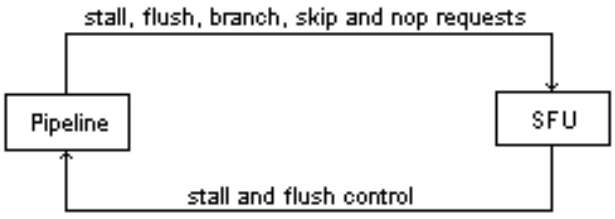
Međutim, iznimke se mogu dogoditi. Npr. LPM instrukcije trebaju čitati PM u fazi s5. Također, load/store naredbe moraju biti u mogućnosti čitati/pisati RF u fazi s5. Iznimke se obrađuju na razini upravitelja hardverskih resursa.

3.1.3.1 Jedinica za zadržavanje i ispuštanje – Stall i Flush unit – SFU

Zbog prethodno navedenih iznimki, različiti dijelovi protočne strukture se mogu natjecati za dani hardverski resurs. Kako bi se riješili eventualni konflikti oko hardverskog resursa, potreban je mehanizam. SFU implementira takvu funkciju, upravljajući zahtjevima za hardverske resurse. SFU zadržava neke instrukcije (neke faze protočne strukture), dok drugima omogućava izvršavanje.
Upravljanje zadržavanjem (eng. stall handling) se vrši kroz dva skupa signala:

- SFU zahtjevi (SFU ulazi)
  - Stall zahtjevi
  - Flush zahtjevi
  - Zahtjevi za grananjem (eng. *branch*)
  - Zahtjevi za preskok (eng. *skip*)
- SFU kontrolni signali (SFU izlazi)
  - Stall kontrola
  - Flush kontrola

Postoji po jedan par stall-flush kontrolnih signala za svaki dio protočne strukture s1, s2, s3, s4, s5, s6.



Slika 33. STALL i FLUSH "dijagram"

Svaka instrukcija ima ugrađeno stall ponašanje, koje se dekodira preko instrukcijskog dekodera. Različite instrukcije u protočnoj strukturi, u različitim fazama izvršavanja, pristupaju SFU jedinici na identičan način kao što pristupaju bilo kojem hardverskom resursu, preko SFU zahtjeva za pristup.

SFU prioritizira stall/flush/branch/skip/nop (zadnje navedeno eng. no operation – nop, nema operacije) zahtjeve i odgađa nove instrukcije dok stare

instrukcije ne oslobde hardverski resurs (SFU hardverski izvor uključen). Proces odgode se vrši putem stall-flush kontrola.

Vrijedi pravilo: kada se dogodi konflikt oko resursa, starija instrukcija ima prednost.

Neke instrukcije zahtjevaju ubacivanje nop prije instrukcije, kako bi se oslobodio hardverski resurs koji se koristio od strane ranije dospjelih instrukcija. Npr. load instrukcije moraju „ukrasti“ port 1 RF-u od ranijih instrukcija.

NOP-ovi (eng. *no operation*) se ubacuju na način da se odugovlače jedni dijelovi protočne strukture i izbacuju drugi ili pak isti dijelovi. Druge instrukcije trebaju nop poslije instrukcije, tako da bi se prethodna instrukcija završila i oslobodili se hardverski resursi. Npr. store instrukcija mora čekati jedan period takta, dok prethodna instrukcija oslobodi port za pisanje RF-a.

Dvije situacije se razlikuju s gledišta kontrolne strukture. U drugoj situaciji, instrukcija samu sebe zadržava i ispušta, što uzrokuje dodatne probleme koji se rješavaju uvđenjem nop automata stanja u fazi s4, čija je jedina uloga ubacivanje barem jedne nop poslije bilo koje instrukcije.

3.1.3.1.1 Zahtjevi za SFU

- Stall zahtjevi

  SFU odugovlači sve ranije faze. Međutim, ako se samo odugovlači, trenutna instrukcija se razvlači u 2 instance. Jedna od njih se mora uništiti (ispustiti). Ranija instanca se uništava (prethodna faza se ispušta).
  Stoga, nop se uvodi u protočnoj strukturi prije instrukcije.
  Ako više od jedne faze zahtjevaju stall u isto vrijeme, stariji zahtjev ima prioritet (raniji će biti odugovlačen zajedno s ostalima). Samo poslije toga, ranijem zahtjevu će se odobriti stall status preko stall i flush kontrolnih signala.
  - Pavr_s3_stall_rq
  - Pavr_s5_stall_rq
  - Pavr_s6_stall_rq
- flush zahtjevi

  SFU jednostavno ispušta tu fazu.

Više od jednog ispuštanja se može prihvatiti u isto vrijeme, bez natjecanja. Međutim, događa se da svi flush zahtjevi zahtjevaju ispuštanje istog dijela protočne strukture, s2.

Flush zahtjevi:

- Pavr_s3_flush_s2_rq
- Pavr_s4_flush_s2_rq
- Pavr_s4_ret_flush_s2_rq
- Pavr_s5_ret_flush_s2_rq
- Pavr_s51_ret_flush_s2_rq
- Pavr_s52_ret_flush_s2_rq
- Pavr_s53_ret_flush_s2_rq
- Pavr_s54_ret_flush_s2_rq
- Pavr_s55_ret_flush_s2_rq

- branch zahtjevi

  SFU tretira skokove kao grananje koje ima relativne adrese skoka jednakima 0, 1 ili 2, ovisno o uvjetu skoka i duljini sljedeće instrukcije (16/32 bita)

  Zahtjevi za preskok (eng. skip requests):

  - Pavr_s6_skip_rq
  - Pavr_s61_skip_rq

- Nop zahtjevi

  SFU odugovlači sve ranije pristigle instrukcije. Trenutna instrukcija se razbija na dvije instance. Starija instanca se uništava (ista faza koja je predala zahtjev za nop fazu se ispušta).

  Stoga, nop je uvodi u protočnu strukturu poslije instrukcije.

  Kako bi se to ostvarilo, manji automat stanja je potreban izvan protočne strukture, zato što bi u protivnom taj dio beskonačno odugovlačio samog sebe.

  Nop zahtjevi:

  - Pavr_s4_nop_rq

3.1.3.1.2 SFU kontrolni signali

Svaki glavni dio protočne strukture (s1 – s6) ima 2 vrste kontrolnih signala generiranih od strane SFU-a:

- Stall kontrola
  Svi registri u ovoj fazi ostaju nepromijenjeni. Svi zahtjevi prema hardverskim resursima (poput RF, IOF, BPU, DACU, SREG itd.) su resetirani na 0.
- flush kontrola
  Svi registri u ovoj fazi su resetirani na 0, skoro na nop stanje. Također, svi zahtjevi prema hardverskim resursima su resetirani.

Svaki dio protočne strukture ima dodijeljenu zastavicu koja određuje da li ta faza ima pravo pristupa hardverskom resursu. Te zastavice se također mogu upravljati preko SFU-a.
Zastavice za omogućavanje hardverskih resursa:

- pavr_s1_hwrq_en
- pavr_s2_hwrq_en
- pavr_s3_hwrq_en
- pavr_s4_hwrq_en
- pavr_s5_hwrq_en
- pavr_s6_hwrq_en

3.1.3.1.3 Rad u sjeni – Shadowing

Pogledajmo sljedeću situaciju: load instrukcija čita podatkovnu memoriju tijekom s5 faze. Pretpostavimo da sljedeći period, starija instrukcija odugovlači s6, tijekom koje bi izlaz podatkovne memorije trebao biti upisan u RF. Poslije još jednog perioda, stall se miče skupa sa s6 zahtjevom za pisanje u RF, ali se izlaz iz podatkovne memorije promijenio tijekom odugovlačenja. Kada se makne odugovlačenje, u RF se upisao sačuvani podatak.

3.1.3.1.4 Shadow protokol

Ako fazi protočne strukture nije dopušteno davati zahtjeve za hardverskim resursima, tada se označava svaki entitet u toj fazi kao da se „zasjenjuje“ izlaz, i piše se u vezani „shadow“ registar odgovarajući izlaz. U protivnom se označava kao „unshadowed“.

Dokle god je svaki entitet označen kao „shadowed“, bit će pročitan (od strane bilo kojeg entiteta kojem je to potrebno) od strane vezanog shadow registra, prije nego direktno iz izlaza podataka.

Kako bi se omogućio „shadowing“ tijekom više uzastopnih odugovlačenja, shadowing entiteta se radi jedino ako još nisu „zasjenjeni“.

U osnovi, uvjet da se izlaz entiteta stavi pod shadow stanje je da su hardverski resursi onemogućeni tijekom te faze. Međutim, postoje iznimke. Npr. LPM skup instrukcija „krade“ pristup programskoj memoriji odugovlačeći instrukciju koja bi se normalno u tom trenutku dohvatila. Odugovlačeći, zahtjevi za hardverskim resursom postaju onemogućeni u toj fazi. Svejedno, LPM skup mora biti u mogućnosti direktnog pristupa izlaza programske memorije. PM ne smije biti shadowed iako tijekom faze s2 (u kojoj se PM normalno pristupa) su svi zahtjevi za hardverom onemogućeni po defaultu. Srećom, postoji samo nekoliko takvih iznimki („rupe“ u shadow protokolu). Ukupno gledajući, shadow protokol je dobra ideja, s obzirom da dopušta prirodno i automatsko upravljanje grupom registara pozicioniranih u osjetljivim područjima.

3.1.4 Protočna struktura i hardverski resursi

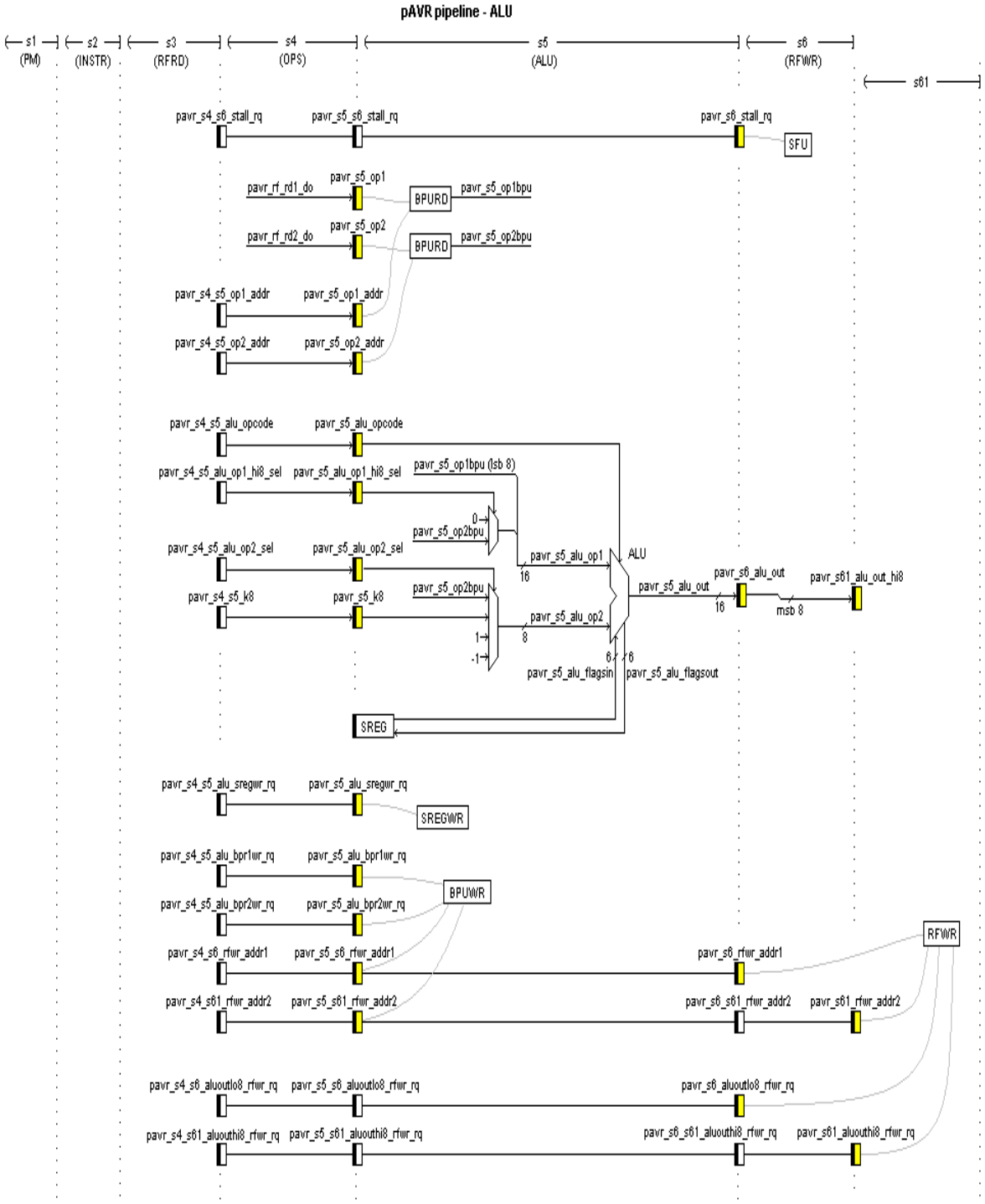
3.1.4.1 ALU

ALU je resurs koji je pod punom kontrolom faze s5. Postoje 2 ulazna operanda. Prvi se uzima s RF porta 1 za čitanje, veličine 8 bita ili se uzima iz RF-a porta 1 (nižih 8 bita) i RF porta 2 za čitanje ako se radi o 16 bitnom operandu. Drugi operand se uzima iz RF-a porta 2 za čitanje ili direktno iz instrukcije; uvijek je širine 8 bita. Oba operanda se šalju u ALU kroz BPU.

Sve instrukcije koje zahtjevaju ALU upisuju svoj rezultat u BPU. Detalji oko hardverskih resursa ALU jedinice su opisani ranije. Instrukcije koje koriste registre vezane uz protočnu strukturu:

- ADD, ADC, ADIW
- SUB, SUBI, SBC, SBCI, SBIW
- INC, DEC
- AND, ANDI
- OR, ORI, EOR
- COM, NEG, CP, CPC, CPI, SWAP
- LSR, ROR, ASR
- MUL, MULS, MULSU
- FMUL, FMULS, FMULSU
- MOV, MOVW

Registri iz protočne strukture vezani uz ALU su prikazani na sljedećoj slici.



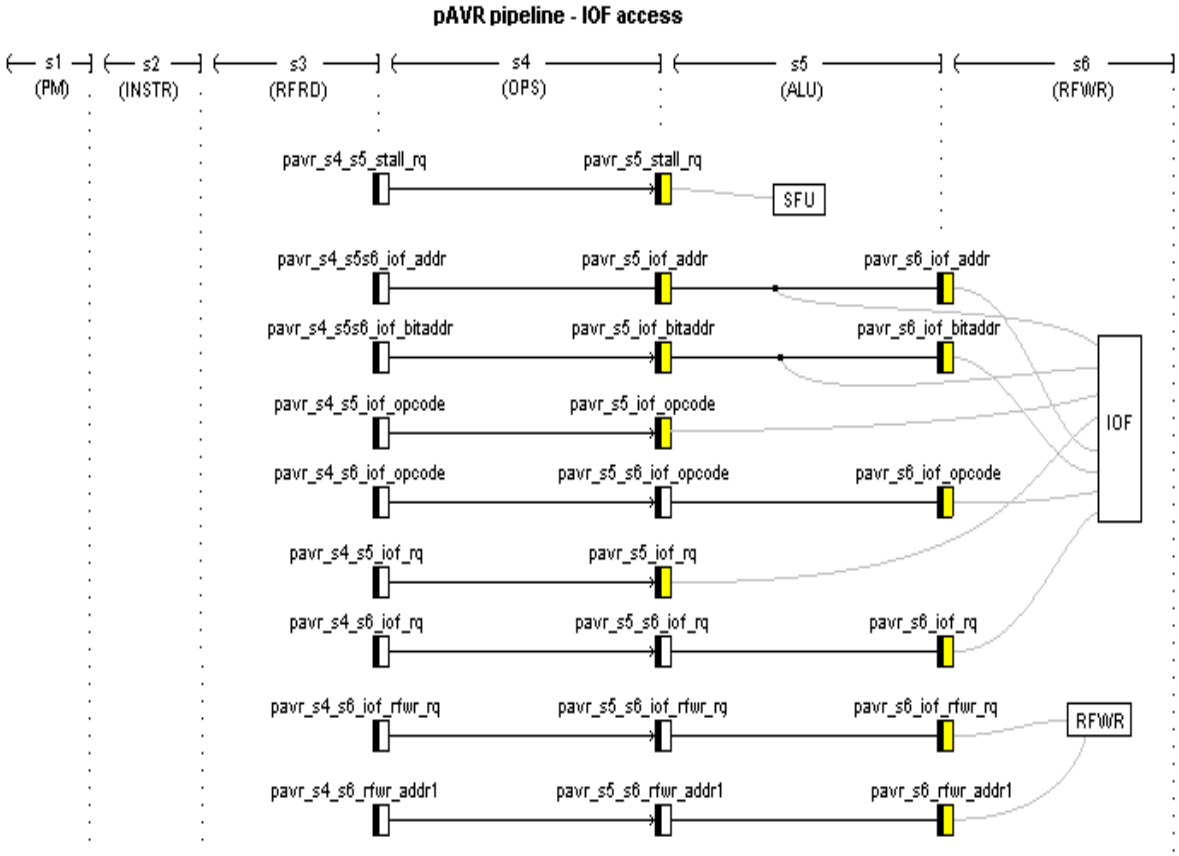
Slika 34. Protočna struktura – komunikacija s ALU

3.1.4.2 IOF pristup

IOF-u se pristupa tijekom faza s5 i s6. Osim bajtno orijentiranih operacija čitanja i pisanja, moguća je i obrada bitova. Sljedeći podaci se daju IOF-u, za svaki dio protočne strukture u kojima se traži pristup IOF-u:

- Adresa bajta
- Adresa bita
- Ulazni podatak, tj. bajt

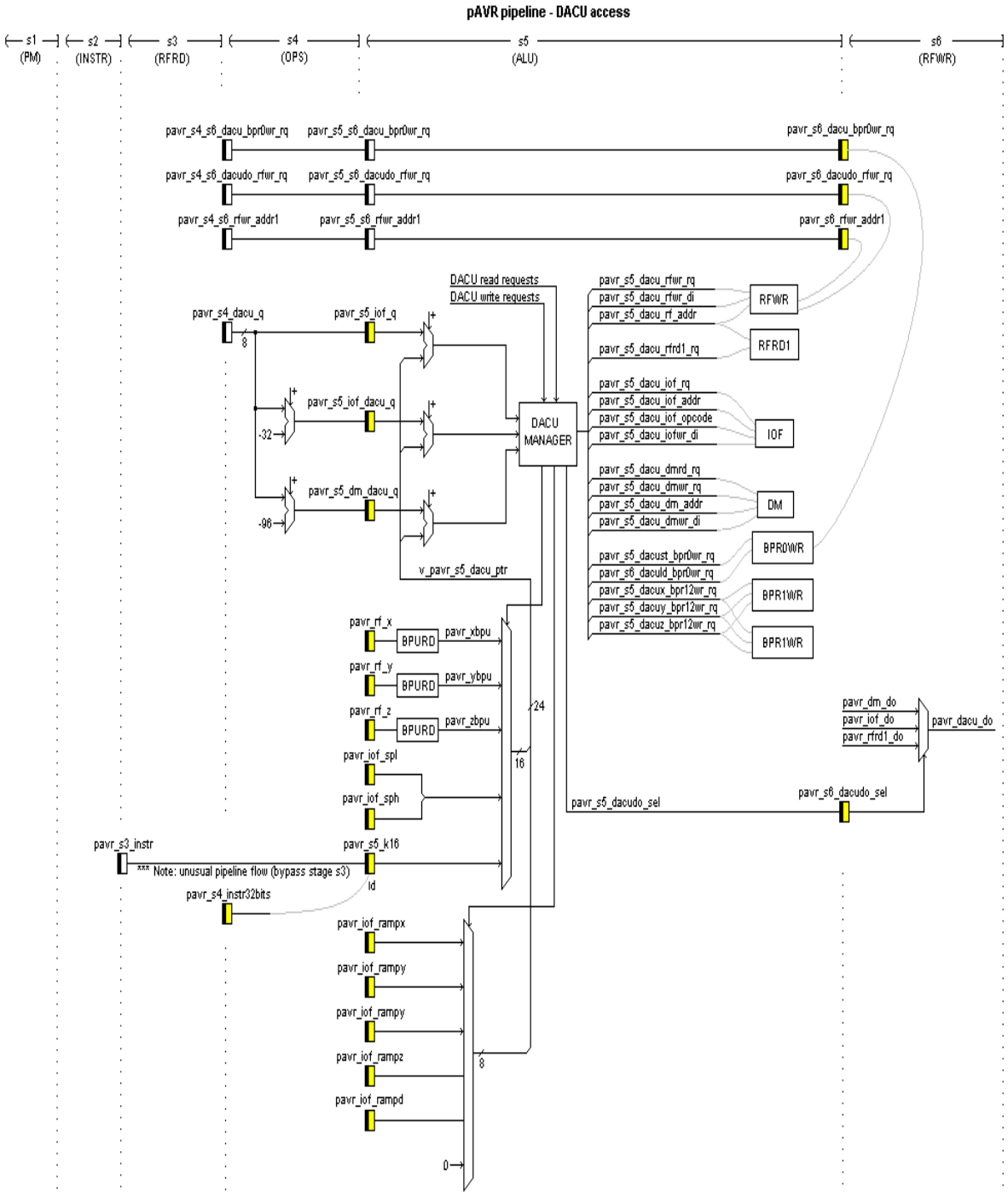
Glavni registri koji implementiraju IOF pristupne instrukcije su prikazani na sljedećoj slici:



Slika 35. Protočna struktura – IOF pristup

3.1.4.3 DACU pristup

DACU upravlja Unified memorijom, mapirajući adrese UM-a u RF, IOF ili DM adrese. Također smješta zahtjeve za pristup prema RF-u, IOF-u ili DM-u, kao odgovor na UM zahtjeve za pristup.



Slika 36. Protočna struktura – DACU pristup

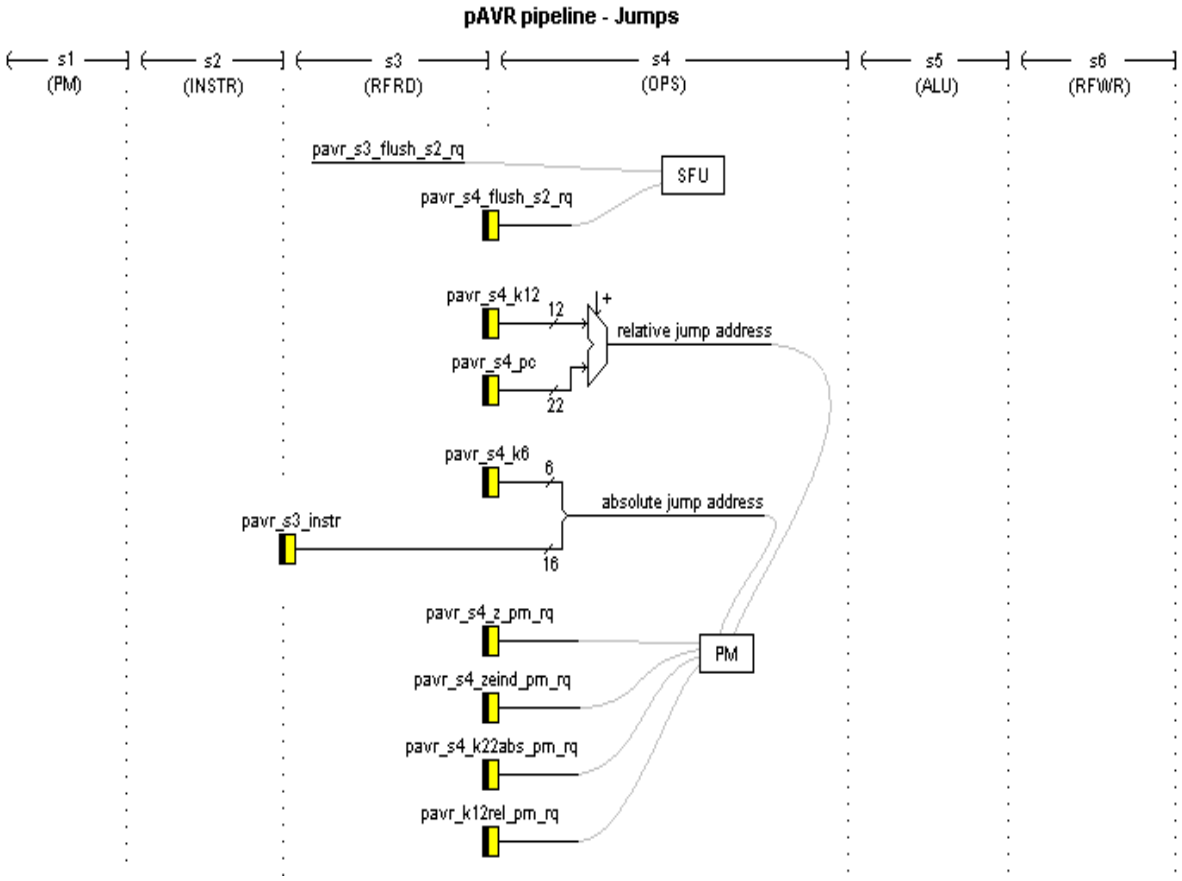
3.1.4.4 Skokovi – Jumps

Postoje 4 instrukcije skoka:

- RJMP (eng*. relative jump* – relativni skok)
  Adresa skoka se dobiva dodajući trenutnom programskom brojilu 12 bitni predznačni odmak (eng. offset) dobiven od strane instrukcijske riječi
- IJMP (eng. *indirect jump* – indirektni skok)
  Adresa na koju se skače se čita iz Z pokazivačkog registra.
  Odredište na koje se skače se nalazi u nižih 64K-riječi PM-a.
- EIJMP (eng. *extended indirect jump* – prošireni indirektni skok)
  Adresa na koju se skače se čita iz EIND:Z (višljih 6 bitova EIND registra u IOF-u i nižih 16 bitova iz Z pokazivača u RF-u). Ovaj skok pristupa cjelom 22 bitnom adresnom prostoru PM-a.
- JMP (eng. *long jump* – „dugačak“ skok)
  Adresa na koju se skače se čita iz dvije uzastopne instrukcijske riječi
  Ovaj skok pristupa cjelom 22 bitnom adresnom prostoru PM-a.

Kada se skok detektira u protočnoj strukturi, sljedeće dvije instrukcije (koje je već „beskorisno“ dohvatila PM) se ispuštaju (eng. *flushed*). Tada se traži dopuštenje od PM managera za pristup PM-u i modificiranju instrukcijskog tijeka (mijenjanje programskog brojila – PC-a). Nakon toga, osim ako se ne ispusti ili odugovlači od strane starije instrukcije, instrukcija skoka će namjestiti protočnu strukturu za dohvat s nove adrese u PM-u.

RJMP i JMP traju 3 perioda, dok IJMP i EIJMP traju 4 perioda takta.

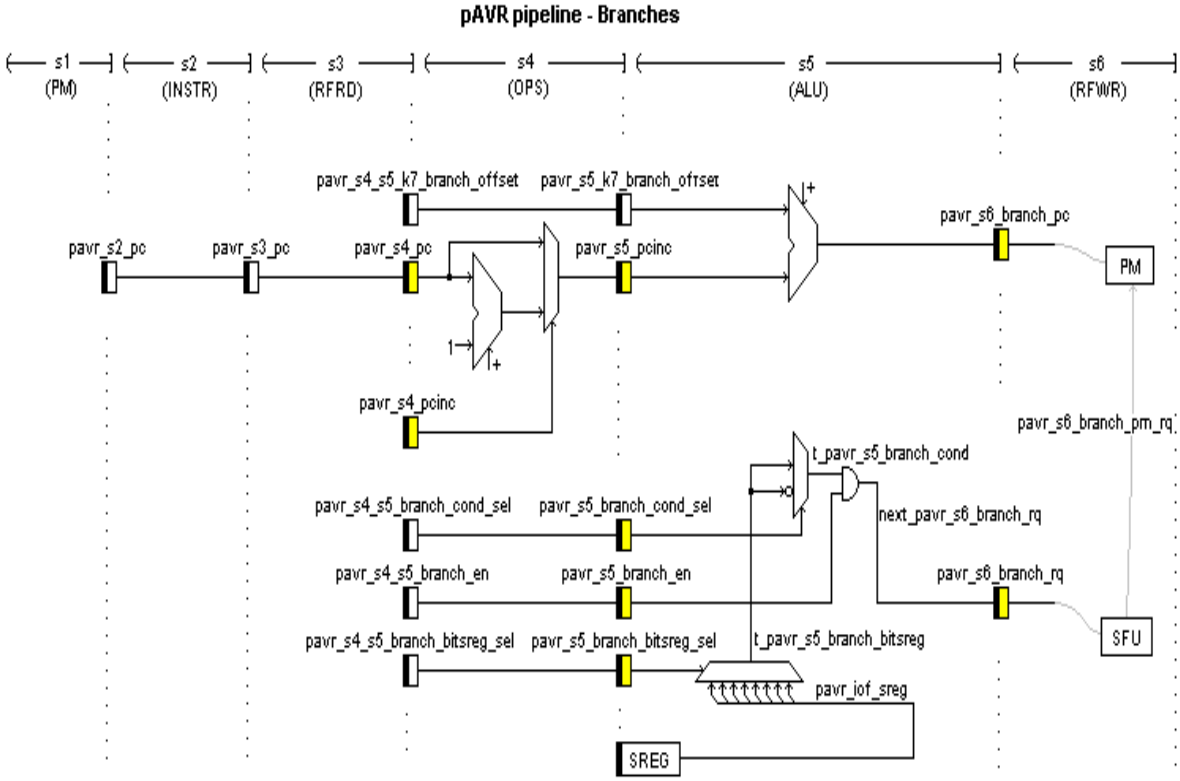


Slika 37. Protočna struktura – skokovi

3.1.4.5 Grananja

Uvjeti za grananje kod 7 bitnog relativnog skoka se čitaju u SR-u. Ako uvjet za skok nije zadovoljen, nikakva daljnja akcija se ne sprovodi. Međutim, ako se uvjet za skok pokaže kao istinit, tada se prethodne faze ispuštaju i od SFU-a se traži grananje. SFU potom traži od PM managera dopuštenje za pristup PM-u kako bi modificirao programski tijek. Grananje se nalazi u fazi s6.

Grananja koja se ne dogode, tj. čiji uvjet nije istinit traju 2 perioda, dok grananja čiji je uvjet istinit traju 4 perioda takta.

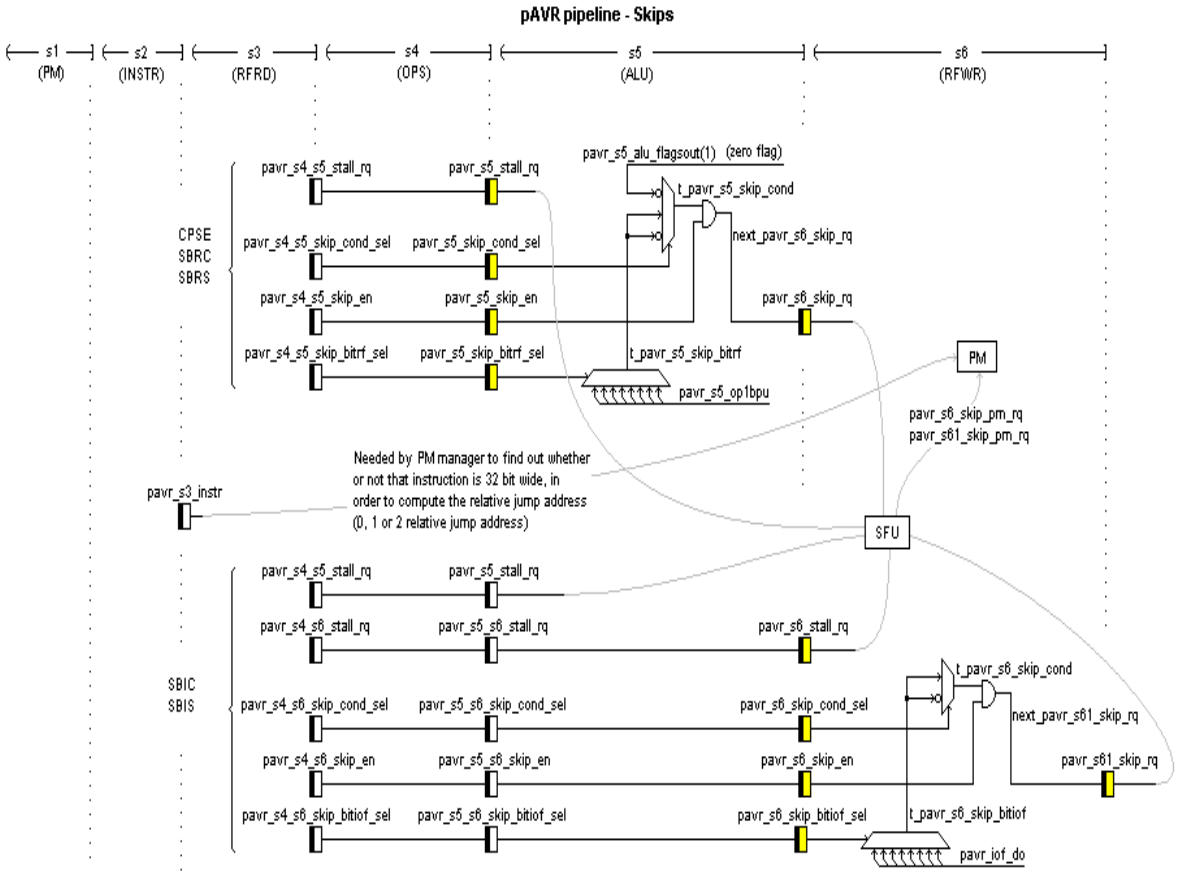


Slika 38. Protočna struktura – grananja

3.1.4.6 Preskoci – Skips

Preskoci su implementirani kao grananja s relativnim adresama 0, 1 ili 2, ovisno o uvjetu i veličini sljedeće naredbe (16 ili 32 bita). Postoje dvije vrste: prva koja šalje zahtjev u fazi s6 (isto kao i grananje) i druga koja šalje zahtjev u fazi s61. Prva uključuje instrukcije CPSE (eng. Compare registers and skip if equal – usporedi registre i preskoči ako su jednaki), SBRC i SBRS (preskoči ako je je bit u registru obrisan/postavljen). Druga uključuje SBIC, SBIS (prekosči ako je bit u UI regisrtu obrisan/postavljen).

CPSE, SBRC i SBRS traju 2 perioda ako se ne dogode, 4 ako se dogode. SBIC i SBIS traju 3 perioda ako se ne dogode, 5 ako se dogode.



Slika 39. Protočna struktura – preskoci

3.1.4.7 Pozivi

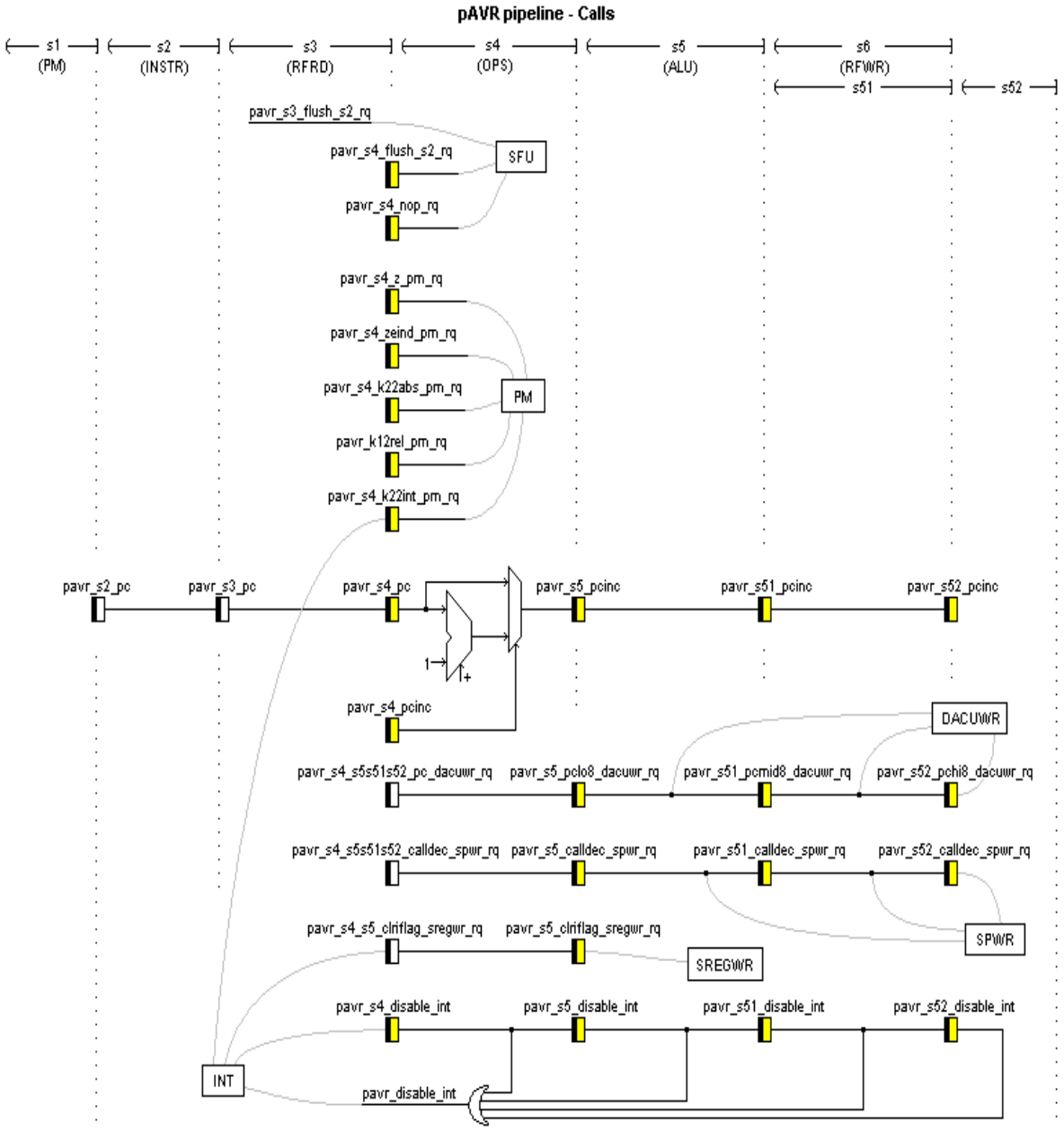
Postoje 4 instrukcije za poziv, analogno instrukcijama skoka:

- RCALL (eng. relative call – relativni poziv)
  Adresa poziva se dobiva dodajući trenutnom programskom brojilu 12 bitni predznačni odmak (eng. offset) dobiven od strane instrukcijske riječi
- ICALL (eng. indirect call – indirektni poziv)
  Adresa za koju se poziva se čita iz Z pokazivačkog registra.
  Odredište na koje se ide se nalazi u nižih 64K-riječi PM-a.
- EICALL (eng. extended indirect call – prošireni indirektni poziv)
  Adresa za koju se poziva se čita iz EIND:Z (višljih 6 bitova EIND registra u IOF-u i nižih 16 bitova iz Z pokazivača u RF-u). Ovaj poziv pristupa cjelom 22 bitnom adresnom prostoru PM-a.
- CALL (eng. far call – „daleki“ poziv)
  Adresa za koju se poziva se čita iz dvije uzastopne instrukcijske riječi
  Ovaj poziv pristupa cjelom 22 bitnom adresnom prostoru PM-a.

Osim navedenog, postoji još jedna vrsta poziva, autmatski ubačena u protočnu strukturu kada se obrađuje prekid. U dodatku regularnim pozivima, implicitni prekid može isto pobrisati prekidnu zastavicu (zastavica I u SR-u). Na taj način, ugniježđeni prekidi su inicijalno onesposobljeni. Međutim, mogu se eksplicitno omogućiti, no ne preporuča se iako je implementirano radi AVR kompatibilnosti.

Nakon što prekid generira implicitni poziv, daljnji prekidi su onemogućeni 4 perioda. Na taj način, barem jedna instrukcija će se izvršiti u pozvanoj podrutini. Tek nakon toga, drugi prekid može promijeniti instrukcijski tijek.

Svi pozivi traju 4 perioda.



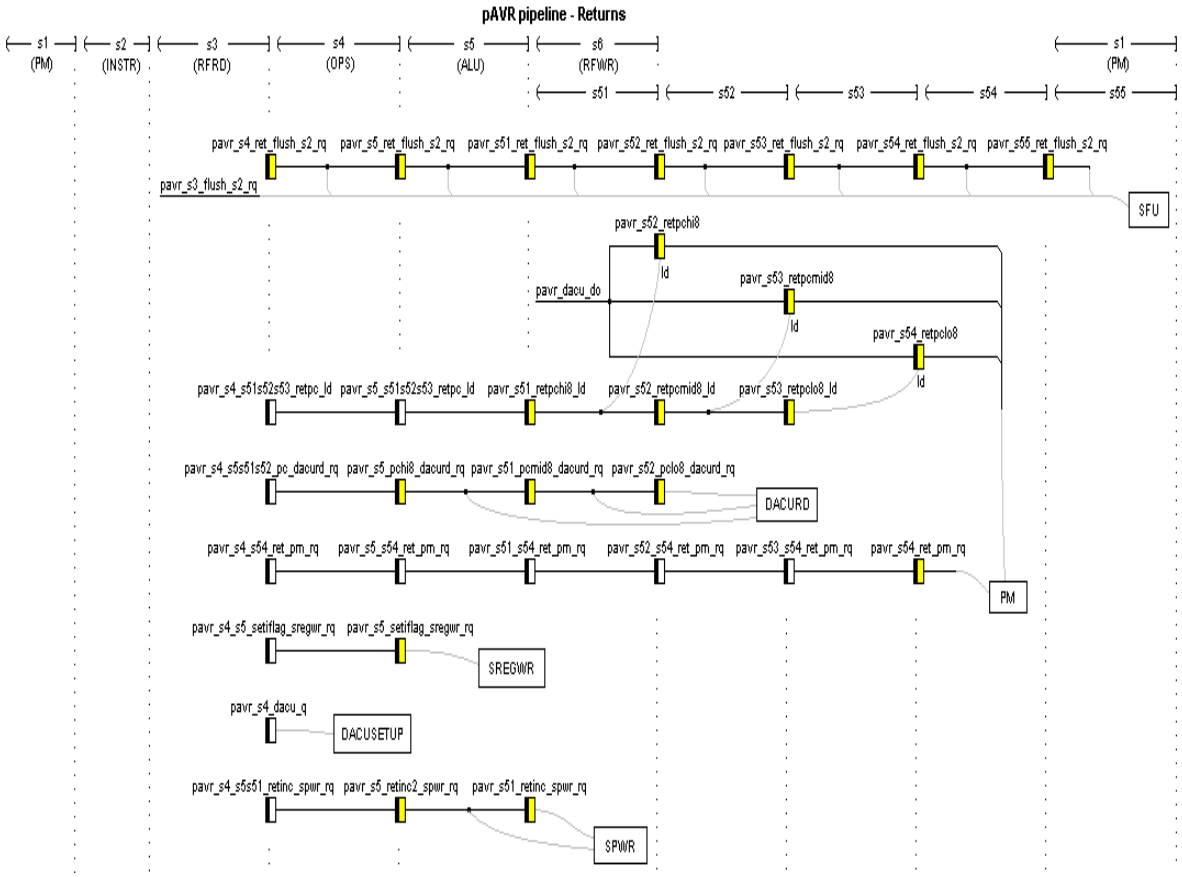
Slika 40. Protočna struktura – pozivi

3.1.4.8 Povrati

Postoje dvije vrste povrata:

- RET – povratak iz podrutine. Programsko brojilo se puni s povratnom adresom (22 bita širine) pročitanom sa stoga i pokazivač na stog se povećava za 3.
- RETI – isto kao i RET s dodatkom da postavlja prekidnu zastavicu (zastavica I u SR-u).

Povrati su najsporije instrukcije u pAVR-u. Traju 9 perioda. Prva 2 perioda se troše zbog čekanja da prethodne instrukcije pišu u Unified memoriju. Sljedećih 5 perioda, programsko brojilo se čita iz Unified memorije. U konačnici, još 2 perioda se troše dok se ciljana instrukcija prenese u instrukcijski registar.

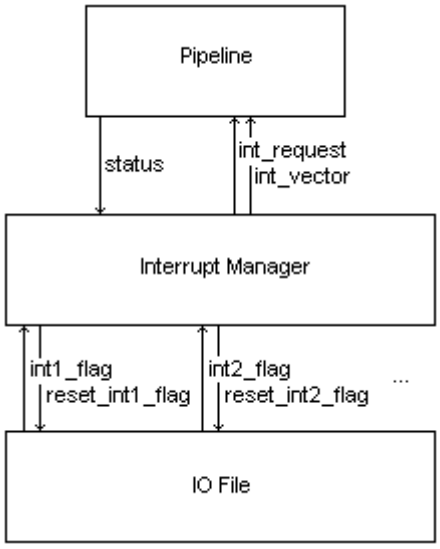


Slika 41. Protočna struktura – povrati

3.1.4.9 Prekidi

Prekidni sustav može na silu smjestiti pozive u fazu s3, kao rezultat specifične UI aktivnosti.

Jezgra prekidnog sustava je modul za prekidnog upravitelja (eng. Interrupt manager module). Daje prioritete izvorima prekida, provjerava da li su prekidi omogućeni i ako je protočna struktura spremna obrađuje prekide, šalje zahtjeve za prekid protočnoj strukturi, skupa s vezanim prekidnim vektorom i kontrolnim signalima za protočnu strukturu.



Slika 42. Dijagram prekidnog sustava

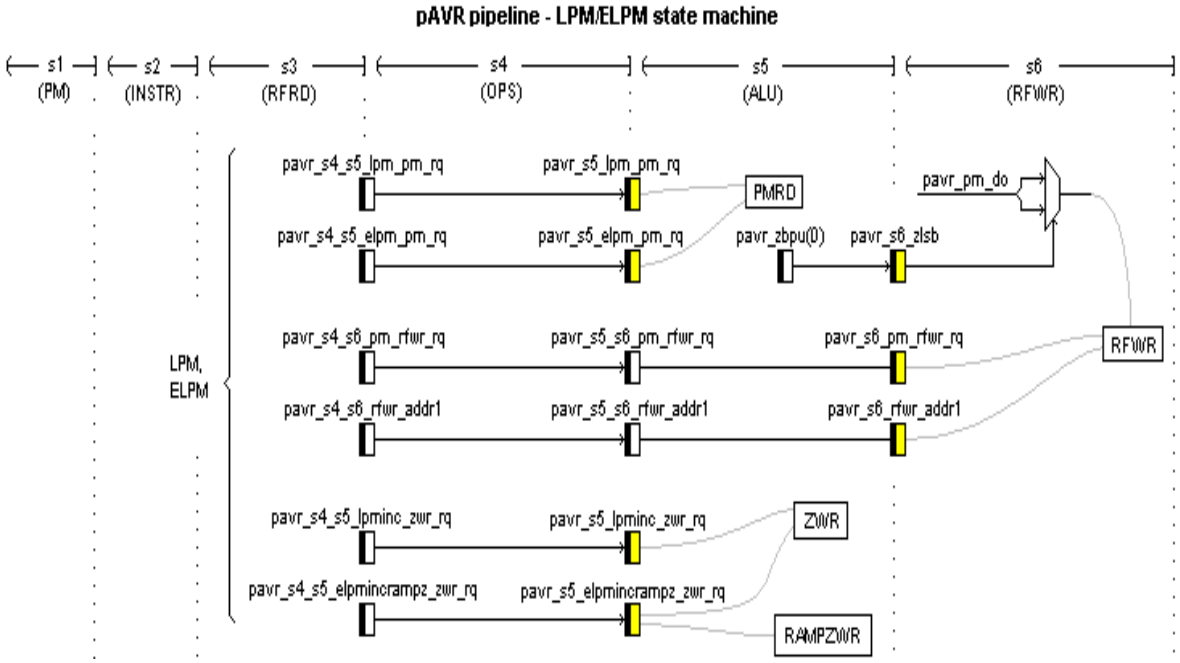
Protočna struktura prima zahtjeve za prekid prisiljavajući instrukcijski dekoder da dekodira sve instrukcije poziva, s apsolutnom adresom skoka danom od strane Interrupt managera. Sljedeće dvije instrukcije, koji su već uzaludno dohvaćene, se ispuštaju.

Prekidni vektori su parametrizirani i mogu se smjestiti bilo gdje u PM. Svaki prekid ima parametrizirani prioritet. U trenutnoj implementaciji, 32 izvora prekida su predviđena, 2 su implementirana: ranije spomenuti vanjski prekid 0 i timer 0 prekid.

Zbog razloga što Interrupt manager dijeli mnogo resursa s IOF-om, nisu napravljeni kao zasebni entiteti, nego je manager ugrađen u IOF. Latencija prekida je 5 perioda (1 period treba Interrupt manageru i 4 perioda trebaju za implicitni poziv).

3.1.4.10 Ostalo

Priključak LPM/ELPM automata stanja u protočnu strukturu:



Slika 43. LPM/ELPM automat stanja

3.2 Izvorni kodovi

Kodovi se sastoje od sljedećih datoteka:

- Std_util.vhd
  - Rutine za konverziju često korištenih tipova
  - Osnovne aritmetičke funkcije
  - Funkcije za predznačno proširenje i proširenje nulama
  - Funkcije za usporedbu vektora
- pavr_util.vhd
  - pristupne funkcije za BPU
  - funkcija za upravljanje prekidima
- pavr_constants.vhd
  - često korištene konstante
- pavr_data_mem.vhd
- pavr_alu.vhd
- pavr_register_file.vhd
- pavr_io_file.vhd
- pavr_control.vhd
  - pAVR protočna struktura (pAVR jezgra)

Testni kodovi:

- test_pavr_alu.vhd
  Testira ALU.
- test_pavr_util.vhd
  Testira „pogodnosti“ (eng. *utilities*) definirane u std_util.vhd.
- test_pavr_data_mem.vhd
  Testira DM.
- test_pavr_register_file.vhd
  Testira RF.
- test_pavr_io_file.vhd
  Testira IOF.
- test_pavr_constants.vhd

Definira potrebne konstante glavnom testu entiteta.

- test_pavr_util.vhd
  Definira potrebne „utilities“ glavnom testu entiteta.
- test_pavr_pm.vhd
  Definira PM potrebnu glavnom testu entiteta.
- test_pavr.vhd
  Definira glavni test entiteta. Testira pAVR kao cjelinu.

3.2.1 Konvencije pri pisanju VHDL kod(ov)a

Terminologija se odnosi na protok podataka. Primjerice, pavr_s4_s6_rfwr_addr1 se dodjeljuje u s3 (od strane instrukcijskog dekodera), pomiče se u pavr_s5_s6_rfwr_addr1, koji se u konačnici pomiče u pavr_s6_rfwr_addr1 (terminalni registar). Samo taj nosi informaciju koji stvarno koristi manager hardverskih resursa. Određeno ovaj navedeni signalizira zahtjev za pristup RF manageru za pisanje.

Razbijanje procesa:

- Zahtjev da se hardverski izvori upravljaju preko dodijeljenih procesa, jedan VHDL proces po hardverskom resursu
- Glavni asinkroni proces (instrukcijski dekoder) računa vrijednosti koje inicijaliziraju protočnu strukturu u s3
- Glavni asinkroni proces dodjeljuje nove vrijednosti registrima u protočnoj strukturi.

3.3 Testiranje pAVR modula

Svaki pAVR modul je zasebno testiran. Zasebni izvršeni testovi su navedeni ispod, grupirani pod entitetima pod testom:

- **utilities** definirani u std_util.vhd
  Pridruženi testni dokument je test_std_util.vhd
  Definicije koje se nalaze u std_util.vhd:
  - rutine za konverziju tipova često korištenih u drugih source dokumentima
  - osnovne aritmetičke funkcije
  - funkcije za proširenje s predznakom i nulama (oboje testirano u test_std_util.vhd).
  - funkcija za usporedbu vektora (testirano u test_std_util.vhd)

- **aritmetičko logička jedinica – ALU**
  Vezani testovi su definirani u test_pavr_alu.vhd. Sastoje se od provjere ALU izlaza i zastavica za sve ALU instrukcije, jedne po jedne, za sljedeće situacije:
  - carry in = 0
  - carry in = 1
  - zbrajanja generiraju preljev
  - oduzimanja generiraju preljev

  Ukupno je 26 ALU instrukcija za provjeru svake situacije.

- **Register File**
  Vezani testovi su definirani u test_pavr_register_file.vhd:
  - Čitanje svih portova, jedan po jedan
    - Čitaj port 1 (RFRD 1)
    - Čitaj port 2 (RFRD 2)
    - Piši na port (RFWR)
    - Piši na pointer registar X (RFXWR)

  - Piši na pointer registar Y (RFYWR)
  - Piši na pointer registar Z (RFZWR)
- Kombinirano RFRD1, RFRD2, RFWR
  Trebaju raditi simultano.
- Kombinirano RFXWR, RFYWR, RFZWR
  Trebaju raditi simultano
- Kombinirano RFRD1, RFRD2, RFWR, RFXWR, RFYWR, RFZWR
  tj. svim RF portovima se pristupa simultano.
  Treba uzeti u obzir da su pokazivački registri dostupni za pisanje, osim preko vlastitih portova, i preko RF write port-a. Pisanje u njih preko

- **U/I File**

  Vezani testovi su definirani u test_pavr_io_file.vhd:
  - testiranje IOF generalnog write/read/bit processing porta, te svih opcodes
    - wrbyte
    - rdbyte
    - clrbit
    - setbit
    - ldbit
  - testiranje IOF port A.
    Port A –> pin-level U/I veza s „vanjskim svijetom“
    Testiranje – čitanje i pisanje s Port A, provjera da li Port A pinovi poprimaju točne logičke vrijednosti (visoko, nisko, visoko i niska impedancija, eng. high, low, high Z ili weak high)
  - testiranje Timera 0:
    - testiranje Timer 0 preskalera
    - testiranje Timer 0 preljev
    - testiranje Timer 0 prekid
  - testiranje Vanjskog Prekida 0 – External Interrupt 0
    Testiranje svake moguće konfiguracije (aktivaciju na nisku razinu, rastući ili padajući brid), tj. da li se aktiviraju zastavice External Interrupt 0-le.

- **Podatkovna memorija – Data Memory**

Vezani testovi su definirani u test_pavr_dm.vhd i sastoje se od potvrda uspješnosti read-write operacija izvedenih nad podatkovnom memorijom

3.4 Testiranje pAVR entiteta

pAVR je u cjelini testiran na način da se izgrađivao entitet koji je obuhvaćao procesor, programsku memoriju i nekoliko multipleksora. Ti multipleksori omogućavaju kontrolu programskoj memoriji kako bi mogla testirati entitet ili sam procesor (promatranje kako izvršava instrukcije iz programske memorije).

Binarna datoteka koja se izvršava od strane pAVR-a tokom testa je automatski učitana u programsku memoriju koristeći ANSI C Utility, TagScan. Testni entitet ima više oznaka raširenih preko source koda u obliku komentara. TagScan utility čita binarnu datoteku koja će se učitati, skenira testnu datoteku i ubacuje VHDL linije u propisno označena mjesta. Te linije pune programsku memoriju koristeći vlastiti port za pisanje. Takav način inicijaliziranja programske memorije je općenitiji nego korištenje U/I VHDL funkcija.

TagScan utility se koristi i za druge svrhe kao primjerice ubacivanje zaglavlja u sve source datoteke te se uvelike koristi kao opći pretprocesor.

Testiranje pAVRa podrazumijeva dizajniranje i izvršavanje ekstremnih naredbi. Sljedeći testovi su izvršeni od strane autora kodova:

- **Prekidi - Interrupts**
  Testiranje interrupt handlinga. Testirani su svi prekidi i vezane periferije (Port A, Timer 0 i External Interrupt 0). Rezultati: tbd.

- **Opći test**
  Kod pisan u strojnom jeziku, testira svaki pAVR instrukciju, jednu po jednu, npr. :
  - Paralelna kašnjenja (eng. *concurrent stalls*)
  - Kašnjenja kombinirana s 32 bitnim instrukcijama
  - Kašnjenja kombinirana s instrukcijama koje mijenjaju tijek instrukcija
  - kontrolne dijelove (Program Memory Manager i Stall i Flush jedinica)

- dijelove s potencijalnim podatkovnim hazardom (Bypass jedinica)
  Rezultati: Verifikacija se sastoji od provjere svake instrukcije, svakog međurezultata i svakog međustanja.

Tablica 8. Rezultati općeg testa

Assembler	Taktova	Broj instrukcija	CPI
Avrasm32, Atmel	667	361	1.85

- **„Sieve“**
  Eratostenovo sito (algoritam za dobivanje svih prostih brojeva manjih od unaprijed izabranoga prirodnog broja), nalazi prvih 100 brojeva :

Tablica 9. Rezultati za Eratostenovo sito

Prevodioc	Taktova	Broj instrukcija	CPI
Avr-gcc, O0	12170	8851	1.37
Avr-gcc, O3	11946	8824	1.35

- **Valni oblici**
  Simuliranje valnih oblika. Pisano u ANSI C-u. Korišteni brojevi s pomičnom točkom (pAVR – 200 clocks za operacije nad brojevima s pomičnom točkom). Pet iteracija izvršeno. Veće vrijednosti čine simulaciju predugom. Provjera rezultata je napravljena konverzijom 25 float brojeva u skalirano polje 25 char vrijednosti, kopirajući („ručno“) iz podatkovne memorije, konstruirajući 3D sliku rezultata i uspoređujući je s referentnom 3D slikom. Rezultati:

Tablica 10. Rezultati za valne oblike

Prevodioc	Taktova	Broj instrukcija	CPI
Avr-gcc	209.175	122.236	1.71

Testovi i sinteze izvršeni s Xilinx web ISE 13.2 paketu i usporedba s ranije navedenim 8-bitnim procesorima:

- **Sinteza za Virtex 5**
  Usporedba s UltraSPARC T1 procesorom koji zauzima između 37000-60000 LUT-ova, sinteza izvršena za Virtex 5 XC5VLX20T FF323 brzine -1:

Tablica 11. Rezultati sinteze za Virtex 5

Prednosti u logici	Korišteno	Omogućeno	Postotak iskorištenosti
Broj registara po sloju	977	12480	7%
Broj LUT-ova po sloju	2920	12480	23%
Broj skroz iskorištenih LUT-FF parova	813	3084	26%
Broj vezanih UI blokova	52	172	30%
Broj blok RAM-ova	1	26	3%
Broj BUFG/BUFGCTRL-ova	1	32	3%

- **Sinteza za Virtex 4**
  Usporedba s LEON3 procesorom koji zauzima oko 3500 LUT-ova, sinteza izvršena za Virtex 4 XC4VFX12 SF363 brzine -10:

Tablica 12. Rezultati sinteze za Virtex 4

Prednosti u logici	Korišteno	Omogućeno	Postotak iskorištenosti
Broj registara po sloju	2245	5472	41%
Broj LUT-ova po sloju	991	10944	9%
Broj skroz iskorištenih LUT-FF parova	4193	10944	38%

Broj vezanih UI blokova	52	240	21%
Broj blok RAM-ova	1	36	5%
Broj BUFG/BUFGCTRL-ova	1	32	3%

- **Sinteza za Spartan**

  Usporedba s PicoBLAZE procesorom koji zauzima oko 100 LUT-ova, sinteza izvršena za Spartan6 XC6SLX75T FGG676 brzine -3:

Tablica 13. Rezultati sinteze za Spartan

Prednosti u logici	Korišteno	Omogućeno	Postotak iskorištenosti
Broj registara po sloju	978	93296	1%
Broj LUT-ova po sloju	3003	46648	6%
Broj skroz iskorištenih LUT-FF parova	825	3156	26%
Broj vezanih UI blokova	52	348	14%
Broj blok RAM-ova	2	172	1%
Broj BUFG/BUFGCTRL-ova	1	16	6%

4. Optimizacijske tehnike

4.1 Proizvodni standardi

Industrijski standardi propisani za FPGA ugradbene procesore su Dhrystone MIPs (DMIPs – eng. ***M****icroprocessor* ***W****ithout* ***I****ntelocked* ***P****ipe* ***S****tages* - mikroprocesor bez isprepletenih dijelova protočne strukture).

Ostvareni DMIPS su bazirani na nekoliko faktora koji maksimiziraju standardne rezultate. Neki od faktora:

- Optimalna razina prevođenja
- Najbrža moguća familija uređaja (osim ako nije drugačije naglašeno)
- Mala latencija memorije, tipično izvedeno na chip-u
- Optimizacija značajnijih parametara procesora

Tablica 14. Rezultati za Altera-u

Procesor	**Tip procesora**	**Korištena familija**	**Postignuta brzina (MHz)**	**Postignut DMIPs**
ARM922T	Hard	Excalibur	200	210
NIOS	Soft	Stratix - II	180	Nije poznato
Nios II	Soft	Stratix - II	Nije poznato	200
Nios II	Soft	Cyclone - II	Nije poznato	100

Tablica 15. Rezultati za Xilinx

Procesor	Tip procesora	Korištena familija	Postignuta brzina (MHz)	Postignut DMIPs
PowerPCTM 405	Hard	Virtex - 4	450	680
MicroBlaze	Soft	Virtex - II Pro	150	123
MicroBlaze	Soft	Spartan - 3	85	65

4.2 Tehnike za poboljšanje performansi

Dizajneri ugradbenih sustava su ponekad frustrirani kad su performanse njihovog FPGA procesora otprilike polovične ili čak manje nego što očekuju. U nekim slučajevima, dizajner nije u mogućnosti udvostručiti rezultate za koje proizvođač tvrdi da su mogući.

Razlog može biti da dizajner nije upoznat sa svim tehnikama dostupnima za optimiranje performansi FPGA procesora. Proizvođači su također toga svjesni pa stoga u potpunosti koriste sve mogućnosti dodatne opreme prilikom vrednovanja svog proizvoda i plasmana na tržište.

Dizajneri koji su upoznati sa standardnim postupcima optimizacije mikroprocesora moraju naučiti koje se tehnike optimiranja softvera primjenjuju na FPGA procesore, dapače, čak i posavladati poneke nove.

Dizajnerski “krajolik” je zasigurno kompliciraniji s FPGA ugradbenim procesorom. Sve ostvarene prednosti ipak vuku neke mane. Naime, složeniji dizajn je neodoljiv mnogim dizajnerima, čak i onim iskusnijim, bilo da se radi o klasičnom ugradbenom ili FPGA dizajneru. Proizvođači i njihovi partneri ulažu značajn napor u educiranje, trening i pružanje podrške dizajnerima koji se služe s njihovom tehnologijom.

Kao uvod u ovu vrstu dizajna, bit će istaknuto par tehnika. Posebne reference su bazirane na istraživanju Xilinx-ovih FPGA i pripadajućih alata, iako Altera-ini FPGA-ovi imaju slične značajke.

4.2.1 Optimizacijske tehnike specifične za FPGA

Budući da dizajner zapravo stvara i izgrađuje hardverski sustav ugradbenog procesora u FPGA, mnogo se može napraviti u svrhu poboljšavanja performansi hardvera. Osim toga, s FPGA ugradbenim procesorom uz dodatne FPGA hardverske izvore, dizajner može razmisliti o vlastitom koprocesoru kojim bi posebno ciljao na glavni algoritam jezgre.

4.2.1.1 Optimizacija i smanjivanje logike

Samo povezivati one periferije i sabirnice koje se koriste. Primjeri:
Ako dizajn ne pohranjuje i izvršava ijednu instrukciju koristeći vanjsku memoriju, ne povezivati instrukcijsku stranu na sabirnicu za periferije. Povezujući instrukcijsku i podatkovnu stranu procesora na jednu sabirnicu stvara se multi-master sustav, koji pak zahtjeva “suca” (engl. *arbiter* - termin često korišten u nastavku). Optimalne performanse sabirnice se postižu kada je jedan master nad sabirnicom.

Logika za debugiranje zahtjeva resurse u FPGA i može predstavljati usko grlo s hardverske strane. Kad je dizajn kompletno debugiran, logika za debugiranje se može ukloniti iz sustava, potencijalno povećavajući performanse sustava. Npr. ako maknemo MicroBlaze Debug Module (MDM) s FSL akceleracijskim kanalom, uštedili smo 950 logičkih ćelija.

Korištenje OPB External Memory Controller - kontrolera za vanjsku memoriju (EMC), za povezivanje SRAM-a i Flash memorije. Stvara 32 bitnu adresnu sabirnicu čak i ako 32 bita nisu potrebna za adresiranje memorije. Korištenje „bustrimming“ periferije, koja miče sve nekorištene adresne bitove. Kada se koristi memorijski kontroler, bustrimmer bi se uvijek trebao koristiti za uklanjanje nekorištenih adresa. To oslobađa puteve i priključke koji bi inače bili iskorišteni. Npr. Xilinx Base System Builder (BSB) to radi automatski.

4.2.1.2 Površinsko i vremensko ograničavanje

Alati za pozicioniranje i povezivanje (eng. *place & route*) izvršavaju svoju ulogu puno bolje bolje kad se dizajneru daju smjernice o tome što je najvažnije. Kod tih alata, preferira se da dizajner može odrediti željenu frekvenciju takta, lokacije priključaka i lokacije logičkih elemenata. Pružanjem tih detalja, alati su u mogućnosti napraviti pametniji kompromis tijekom implementacije dizajna.

Neke periferije zahtjevaju dodatna ograničenja da bi se osigurao pravilan rad. Npr., DDR SDRAM kontroler i 10/100 Ethernet MAC zahtjevaju dodatna ograničenja za garanciju da će alati kreairati točnu i optimalnu logiku. Dizajner mora čitati datasheet za svaku periferiju i slijediti preporučene smjernice za dizajn.

4.2.1.3 Ubrzanje hardvera

Hardver s posebnom namjenom nadmašuje softver. Dizajner koji ozbiljno želi povećati performanse mora ozbiljno uzeti u obzir mogućnost ubrzanja procesora s posebnim hardverom. Iako ova tehnika troši resurse FPGA, poboljšanja performansi su izvanredna.

Dodatna poboljšanja se mogu dobiti korištenjem hardverskog dijelila i „barrel shiftera“ nego za softversko izvođenje tih funkcija. Omogućavanje tih procesorskih mogućnosti troši se više logike, ali se poboljšavaju performanse. U jednom primjeru, omogućavanje hardverskog dijelila i barrel-shifter-a dodaje 414 LC-a, ali su performanse poboljšane za 18,1% (DMIP mjerilo).

Hardverska logika napravljena po mjeri može biti dizajnirana u cilju rasterećivanja FPGA ugradbenog procesora. Kada je identificirano softversko usko grlo, dizajner može izabrati “pretvaranje” tog algoritma u hardver koji mu je potreban. Softverske instrukcije tada mogu biti definirane za rad hardverskog ko-procesora.

Npr. Virtex-4 predstavlja **Auxiliary Processing Unit** (APU) - Pomoćnu Procesorsku Jedinicu za PowerPC. APU pruža izravnu vezu iz PowerPC-ja prema ko-procesorskom hardveru. Primjerice, u MicroBlaze-u, sučelje s niskom latencijom se zove Fast Simplex Link (FSL) sabirnica. FSL sabirnica sadrži više kanala dodijeljenih, jednosmjernih, 32-bitnih sučelja. Budući da su FSL kanali

dodijeljeni, nije potreban arbiter ili posebno upravljanje sabirnicom što omogućava brzo sučelje prema procesoru.

4.2.1.3.1 Rješavanje softverskog uskog grla

Pretvaranje softverskog uskog grla u hardver se može činiti kao vrlo težak zadatak. Tradicionalno, softveraš identificira usko grlo, poslije kojeg se algoritam prebacuje na FPGA dizajnera koji piše VHDL ili Verilog kod u cilju stvaranja hardverskog ko-procesora. Srećom, proces je uvelike pojednostavljen alatima koji su u stanju generirati FPGA hardware iz C koda.

Jedan takav je CoDeveloper tvrtke Impulse Accelerated Technologies. Taj alat omogućuje jednom dizajneru koji je upoznat sa C jezikom da prenese (u nastavku često korišten izraz - porta) softversko usko grlo u vlastiti komad ko-procesorskog FPGA hardvera koristeći CoDevelopers Impulse C biblioteke.

Primjeri algoritama koji bi mogli biti usmjereni na hardverski ko-procesor:

- Inverzna Diskretna Kosinusna transformacija, korištena u JPEG 200
- Brza Fourierova transformacija (FFT)
- MP3 dekodiranje
- Triple-DES i AES enkripcija
- Manipuliranje matricama

Bilo koja operacija, bilo algoritamska, matematička ili paralelna je dobar kandidat za hardversku ko-procesorsku implementaciju. Trošenje FPGA logike je cijena za bolje performanse. Prednosti mogu biti ogromne, poboljšavanje performansi 10 do 100 puta.

4.2.2 Optimizacijske tehnike nespecifične za FPGA

Cilj ovog potpoglavlja je naglasiti kako se mnoge standardne optimizacijske tehnike mogu primijeniti čak i na FPGA i uz to pružiti odlične beneficije.

4.2.2.1 Manipulacija koda

Dostupne su mnoge optimizacije koje utječu na aplikacijski kod. Neke se tehnike upotrebljavaju kad je kod već napisan. Ostale tehnike utječu na to kako se prevodioc nosi s kodom.

4.2.2.2 Razine optimiranja

Optimiranje prevodioca je omogućeno u Xilinx Platform Studio-u (XPS) baziranom na GCC-u. Trenutna verzija MicroBlaze i PowerPC GCC-baziranih prevodioca u EDK 6.3 je 2.95.3-4. Ti prevodioci imaju nekoliko razina optimizacije, uključujući: Razinu 0, 1, 2 i 3 i tako optimizaciju veličine koda. Detaljnije objašnjenje:

- Razina 0: Nema optimizacije
- Razina 1: Prva razina optimizacije. Izvodi skok i “pop” optimizacije.
- Razina 2: Druga razina optimizacije - ova razina aktivira gotovo sve optimizacije koje ne uključuju kompromis između brzine i prostora, tako da se broj izvršenih naredbi ne bi trebao povećavati. Prevodioc ne izvodi „loop unrolling“, „function in-lining“ ili strogu optimizaciju „aliasinga“. Ovo je standardna optimizacijska razina koja se koristi prilikom implementacije programa.
- Razina 3: Najviša optimizacijska razina. Ova razina dodaje opcije poput povećanja veličine koda. U nekim slučajevima ova optimizacijska razina stvara kod koji je manje učinkovit nego kod razine 2 i kao takav bi trebao biti korišten s posebnom pažnjom.

Veličina: Veličina optimizacije - cilj je proizvesti najmanju moguću veličinu koda.

4.2.2.3 Korištenje optimiranih instrukcija proizvođača

Xilinx nudi nekoliko prilagođenih funkcija za Xilinx ugradbene procesore. Jedan primjer je xil_printf. Funkcija je gotovo identična printf funkciji, ali sa sljedećim iznimkama: podrška za realne brojeve je maknuta; nije „reentrant“ funkcija (funkcija koja može biti pozvana od strane više procesa, a da pri tome uvijek ispravno funkcionira); i nema više longlong tipa (64-bita). Za te razlike, xil_printf funkcija je 2953 bajta, što je drastično manje nego printf, koja zauzima 51788 bajtova.

4.2.2.4 Optimiranje u strojnom jeziku (assembleru)

Optimiranje u strojnom jeziku, assembleru, uključujući „in-line assembly“ je podržano od strane GCC-a. Kao i kod svakog mikroprocesora, assembler je vrlo koristan za potpuno optimiziranje kritičnih sekcija. Iako, neki prevodioci ne optimiziraju ostali C kod u datoteci ako je in-line assembly korišten u toj datoteci. Također je problematičan prijenos koda, a na taj problem ne nailazimo kod C-a.

4.2.2.5 Ostale optimizacije

Druge optimizacije vezane uz kod koje bi trebali uzeti u obzir prilikom optimiziranja ugradbenog procesora FPGA uključuju:

- smještaj referenci
- profiliranje koda
- pažljiva definicija varijabli (Xilinx pruža definicije osnovnih tipova)

Strateška upotreba malih dijelova podataka, s pristupima koji su dvostruko brži, razumno korištenje poziva funkcija da se smanji stavljanje i uzimanje sa stoga, duljina petlji (posebno tamo gdje je uključena priručna memorija, tj. cache).

4.3 Memorije

Arhitekture FPGA sklopova podržavaju distribuirane i blok RAM memorije dok se ROM memorija izvodi kao prozivna tablica, implementirane korištenjem LUT-ova (eng. look up table) ili blok RAM-a kojem je kod inicijalizacije FPGA sklopa dodijeljen sadržaj.

Distribuirani RAM se koristi kad nam treba „manji“ kapacitet dok se blok RAM koristi kad nam treba „veći“ kapacitet. Blok RAM memorije su izvedene kao fizički nezavisne jedinice u integriranom krugu. Mogu se konfigurirati da budu jednopristupne ili dvopristupne, s time da ako im se dodijeli sadržaj kod incijalizacije, mogu se koristiti kao ROM memorije kako je navedeno ranije u tekstu.

4.3.1 Korištenje memorije

Mnogi procesori osiguravaju pristup brzoj, lokalnoj memoriji, kao i sučelje prema sporijoj, sekundardnoj memoriji. Isto vrijedi i za FPGA ugradbene procesore. Način na koji je ta memorija korištena ima značajn utjecaj na performanse. Kao i kod drugih procesora, korištenje memorije kod FPGA ugradbenog procesora se može manipulirati s „linker script-om“.

4.3.1.1 Izvedba samo s lokalnom memorijom

Najbrža moguća memorijska opcija je da se sve stavi u lokalnu memoriju. Kao što je već prije navedeno, lokalne memorije su izvedene u obliku BlokRAM-a (BRAM). Ugradbeni procesori pristupaju BRAM-u u jednom sabirničkom ciklusu.

S obzirom da procesor i sabirnice rade na istoj frekvenciji (npr. u MicroBlaze-u), instrukcije pohranjene u BRAM se izvršavaju u punoj frekvenciji procesora.

Primjerice, u MicroBlaze sustavu, BRAM je u osnovi jednak po performansama Razini 1 (L1 cache). PowerPC može raditi na frekvencijama većim nego što su frekvencije sabirnice i ima ugrađeni L1 cache. Stoga BRAM u PowerPC sustavu ima jednake performanse Razini 2 (L2 cache).

Ako program kompletno stane u lokalnu memoriju, tada postižemo optimalne memorijske performanse. Međutim, mnogi programi trebaju veće kapacitete.

4.3.1.2 Izvedba samo s vanjskom memorijom

Memorijski kontroleri za interakciju s različitim vanjskim memorijskim uređajima su povezani na procesorsku sabirnicu za periferije.

Tri podržana tipa izbrisive memorije su SRAM, jedno-podatkovni SDRAM, i double-data-rata (DDR) SDRAM. SRAM kontroler je najmanji i najjednostavniji unutar FPGA, ali je i najskuplji od navedena tri memorijska tipa. DDR kontroler je najveći i najsloženiji unutar FPGA, ali zahtjeva manje priključaka (pin-ova) i najjeftiniji je gledano po megabajtu.

Kao dodatak, vezano uz vrijeme pristupa memoriji, prisutna je latencija na sabirnici za periferije. Npr. OPB (On-chip sabirnica za periferije) SDRAM kontroler zahtjeva četiri do šest ciklusa latencije za pisanje i osam do deset ciklusa latencije za čitanje, ovisno o taktu. Najgora moguća izvedba programa se može postići na način da ga se smjesti u vanjsku memoriju. Kako je optimizacija brzine izvođenja cilj, cijeli program bi trebao rijetko, ako ne i nikad, biti usmjeren isključivo na vanjsku memoriju.

4.3.1.3 Priručna vanjska memorija

Omogućavanje priručne memorije daje gotovo uvijek bolje performanse. Arhitektura priručne memorije varira od modela do modela, s blagom tendencijom da se direktno ugradi u silicij hard procesora. Instrukcijski i podatkovni cache kontroleri spadaju pod birane parametre. Kad su ti kontroleri uključeni, priručna memorija (eng. cache, termin često korišten u nastavku teksta) se “stvara” iz BRAM-a. Stoga, omogućavanje cache-a troši BRAM koji bi se inače mogao koristiti za lokalnu memoriju. Uspoređujući s lokalnom memorijom, cache troši više BRAM-a nego lokalna memorija za istu količinu spremljenih podataka zato što arhitektura cache-a zahtjeva pohranu oznaka adresnih linija. Osim toga, omogućivši cache troši se i dio logike za opću namjenu da bi se stvorili cache kontroleri.

Npr. , test sa Spartan-3 omogućuje 8 KB podatkovnog cache-a i određuje 32 MB vanjske memorije koja se pretvara u priručnu. Cache zahtjeva 12 oznaka za adresne bitove. Takva konfiguracija troši 124 logičke ćelije i 6 BRAM-a. Samo 4 BRAM-a su potrebna u Spartan-3 da bi imali 8 KB lokalne memorije. U ovom slučaju, cache je 50% skuplji u terminima iskorištavanja BRAM-a nego lokalna memorija. Dva dodatna BRAM-a se koriste da bi se spremile oznake adresnih bitova.

Ako je 1 MB vanjske memorije pretvoreno u priručnu uz 8KB podatkovnog cache-a, tada se broj oznaka adresnih bitova može smanjiti na 7. Takva konfiguracija traži samo 5 BRAM-ova, a ne 6 (4 BRAM-a za cache, 1 BRAM za oznake). To je još uvijek 25% više nego da su BRAM-ovi korišteni kao lokalna memorija.

Osim toga, frekvencija rada sustava se može smanjiti kad je omogućen cache. U jednom primjeru, sustav bez ikakvog cache-a je sposoban raditi na 75 MHz; sustav s cache-om može raditi samo na 60 MHz. Omogućavajući cache kontroler dodaje se dodatna logika i kompleksnost dizajnu, smanjujući postignutu frekvenciju sustava prilikom pozicioniranja i povezivanja (“place and route”) FPGA. Stoga, u dodatku trošenja FPGA BRAM-a koji bi se mogli koristiti u svrhu povećanja lokalne memorije, implementacija cache-a može također uzrokovati smanjivanje cjelokupne frekvencije sustava.

Uzevši sve te parametre, omogućavajući cache, posebno s instrukcijskim cache-om, mogu se poboljšati performanse, čak i kad sustav mora raditi na nižoj frekvenciji. Kao što će biti dalje prikazano u DMIP poglavlju, sustav koji radi na 60 MHz s omogućenim instrukcijskim cache-om ima 150% bolju izvedbu nego sustav od 75 MHz bez instrukcijskog cache-a (oba sustava pohranjuju cijeli program u vanjsku memoriju). Kada su omogućeni instrukcijski i podatkovni cache, sustav od 60 MHz nadmašuje sustav od 75 MHZ za 308%.

Ovaj primjer nije najpraktičniji s obzirom da će cjelokupni DMIP program stati u cache. Realni test je koristiti aplikaciju koja je veća od cache-a. Druga mjera opreza se tiče aplikacija koje često premašuju veličinu cache-a. Višestruki promašaji u čitanju cache-a smanjuju performanse, ponekad čineći vanjsku “cache-iranu” memoriju lošiju nego vanjsku memoriju bez cache-a.

Uzevši u obzir sve mjere opreza, omogućavanje cache-a se isplati ako se uspostavi da poboljšava performanse aplikacije.

4.3.1.4 Razbijanje i raspoređivanje koda unutar memorija

Memorijska arhitektura koja pruža najbolje performanse je ona s lokalnom memorijom. Međutim, takva arhitektura nije uvijek praktična budući da mnogi programi premašuju kapacitet lokalne memorije. S druge strane, izvođenje isključivo iz vanjske memorije može imati i do 8 puta slabije performanse s obzirom na latencije sabirnice periferija. Cache-iranje vanjske memorije definitivno poboljšava rezultate, ali postoje i alternativne metode koje mogu pružiti optimalne rezultate.

Jedna od metoda je razbijanje programskog koda, maksimiziranje frekvencije sustava i veličine unutarnje memorije. Kritični podaci, instrukcije i stog su smješteni u unutarnju memoriju. Podatkovni cache nije korišten, čime se omogućuje veća unutarnja memorijska banka. Ako lokalna memorija nije dovoljno velika za pohranu svih instrukcija, dizajner bi trebao razmisliti o omogućavanju instrukcijskog cache-a za veći doseg adresiranja u vanjskoj memoriji koji se koristi za instrukcije.

Ako se ne troši BRAM u podatkovnom cache-u, unutarnja memorija se može povećati i shodno tome imati više prostora. Instrukcijski cache za instrukcije dodijeljene vanjskoj memoriji može biti vrlo učinkovit. Testiranje ili profiliranje pokazuje kojim dijelovima koda je najteže pristupiti; dodjeljivanje tih dijelova lokalnoj memoriji daje veće poboljšanje performansi nego cache-iranje.

Express Logics Thread-Metric test suite je primjer kako malo razbijanje malog koda u unutarnjoj memoriji može imati značajno poboljšanje performansi. Jedna funkcija u **Thread-Metric Basic Processing** testu je identificirana kao vremenski kritična. Podatkovni dijelovi funkcija (koji čine 19% ukupne veličine koda) su allocirani u unutarnju memoriju i instrukcijski cache je omogućen. 60 MHz, cache i razbijen programski sustav postižu performanse koje su 560% bolje nego ne korištenje cache, ne “razbijenog” sustava od 75 MHz koji koristi samo vanjsku memoriju.

Međutim, sustav od 75 MHz pokazuje veće poboljšanje s razbijanjem koda. Ako su vremenski kritični podaci funkcija dodijeljeni unutarnjoj memoriji sustava od 75 MHz, dobivamo poboljšanje od 710%, čak i bez instrukcijskog cache-a za ostatak koda dodijeljenog vanjskoj memoriji.

U ovom jednom slučaju, optimalna memorijska konfiguracija je ona koja maksimizira unutarnju memoriju i frekvenciju sustava bez cache-a. U ostalim sustavima gdje kritični kod nije tako lako identificirati, cache-irani sustav može imati bolje performanse. Na dizajnerima je da iskušaju obje metode i odluče koja im je najbolja.

5. Zaključak

Ugradbeni procesori za FPGA nude mnoge mogućnosti s obzirom na samu preprogramirljivost FPGA sustava, pogotovo ako se razvija i/ili dorađuje već postojeća soft jezgra. Mogućnosti za optimiranje ima uvijek, pogotovo kada je riječ o softverskoj optimizaciji meke jezgre, od racionalnog korištenja hardverskih resursa, ponajprije memorije, pa do razbijanja i skraćivanja programskog koda. To najbolje pokazuje primjer pAVR jezgre gdje se s dobrom i temeljitom softverskom optimizacijom može u pojedinim sustavima puno uštediti te dobiti na brzini.

Glavni problem predstavlja vrijeme razvoja koje prosječno iznosi 6 mjeseci do godinu dana po čovjeku pa je s obzirom na definiranu ulogu tih procesora kao i specificiranu namjenu FPGA te potrebnu razinu znanja i vremena jednog dizajnera (bilo da je riječ o znanju jezika za opis sklopovlja ili o znanju dostupnih alata koje se koriste) preporuka kupiti gotovi „off-the-shelf“ hard procesor, s ugrađenim željenim svojstvima.

6. Literatura

[1] Vučić M. , Molnar G.; Alati za razvoj digitalnih sustava – Materijali za predavanja I, II, III; FER; 2010.

[2] Fletcher B. H. ; FPGA Embedded Processors: Revealing True System Performance; Memec, San Diego, California; 2005.

[3] S interneta, http://www.1-core.com/resources/ , Third-party open-source projects,

[4] S interneta, http://www.actel.com/products/ip/ ,

[5] S interneta, http://doru.info/projects/hdl/pavr/group__pavr__pavr.html

[6] S interneta, http://support.atmel.no/knowledgebase/avrstudiohelp/mergedProjects/AVRASM/AVRASM.htm

[7] S interneta, http://www.opencores.org/project,pavr

Sažetak

FPGA (engl. Field programmable gate array) je integrirani sklop dizajniran da bi se konfigurirao od strane kupca ili dizajnera. Sadrži velik broj programirljivih logičkih vratiju i velik broj povezanih dijelova, čime se omogućava implementacija složenih digitalnih krugova.

Ugrađivanje procesora unutar FPGA ima mnoge prednosti, pogotovo vezanih uz specifične periferije koje se onda lako povezuju na FPGA, poput memorijskih kontrolera.

Postoje dvije vrste CPU jezgri za FPGA: “čvrste” (eng. hard) i “meke” (eng. soft). Hard jezgra je dio integriranog kruga, dok se mekana implementira koristeći FPGA logičke ćelije. Primjer meke jezgre je pAVR.

PAVR (skraćenica od pipelined AVR - AVR s protočnom strukturom) nije specifičan kontroler AVR familije nego maksimalno specificiran AVR. Dovoljno je podesiv da može simulirati većinu kontrolera AVR familije. PAVR je triput brži nego originalna jezgra ako je izgrađen istom tehnologijom. Spada u vrstu FPGA klona koji omogućava performanse koje su bolje od originalne jezgre, poput primjerice dublje protočne strukture.

Industrijski standardi propisani za FPGA ugradbene procesore su Dhrystone MIPs bazirani na nekoliko faktora koji maksimiziraju standardne rezultate (poput male latencije memorije, optimalne razine prevođenja itd.).

Razvoj meke jezgre iziskuje značajnu količinu vremena i znanja, ali može dati izvrsne rezultate, kao što je vidljivo iz primjera pAVR jezgre.

Ključne riječi: FPGA, ugradbeni procesor, MIPS, pAVR

Summary

FPGA (Field programmable gate array) is an integrated circuit designed to be configured by the customer or designer. It contains a large number os programmable logic gates and a large number of connected components, which allows implementation of complex digital circuits.

Embedding of the processor inside the FPGA has many advantages, especially related to specific peripherals that are then easily connected to the FPGA, such as memory controllers.

There are two types of CPU core for FPGA, hard and soft. Hard core is part of an integrated circuit, while soft is implemented using FPGA logic cells. An example of a soft core is pAVR.

PAVR (short for pipelined AVR) is not specific AVR controller but maximally specefied AVR. It's adjusted enough to simulate most AVR controller family. PAVR is three times faster than the original core if it's built with the same technology. It belongs to the type of FPGA clone that enables performance that is better than the original core, sucah as deeper pipeline.

Industry standards prescribed for FPGA embedded processors are Dhrystone MIPs based on several factors that maximize the standard results (such as low memory latency, the optimal level od compiling etc.).

Development of soft cores requires a significant amount of time and knowledge but can give excellent results, as is evident from the examples' pAVR core.

Keywords: FPGA, embedded processor, MIPS, pAVR