UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Borut Rifelj

POVEZAVA CELIČNEGA AVTOMATA S PROCESORJEM OPENRISC

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00462/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **BORUT RIFELJ**

Naslov: **POVEZAVA CELIČNEGA AVTOMATA S PROCESORJEM OPENRISC**

CONNECTION OF CELLULAR AUTOMATON WITH THE OPENRISC PROCESSOR

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Spoznajte se z odprtokodnim procesorjem OpenRISC 1200 se ga naučite uporabljati na nivoju simulacije. Povežite ga s HDL modelom pomnilnika SDRAM preko ustreznega pomnilniškega krmilnika. Implementirajte v Verilogu dodaten periferni modul celični avtomat, ki komunicira s procesorjem OpenRISC preko vodila Wishbone. S simulacijo preverite delovanje celotnega sistema.

Mentor:

prof. dr. Branko Šter

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Borut Rifelj

POVEZAVA CELIČNEGA AVTOMATA S PROCESORJEM OPENRISC

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: izr. prof. dr. Branko Šter

Ljubljana, 2009

Besedilo je oblikovano z urejevalnikom besedil Microsoft Word.

Original izdana tema diplomskega dela

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a BORUT RIFELJ,

z vpisno številko 63040384,

sem avtor/-ica diplomskega dela z naslovom:

Povezava celičnega avtomata s procesorjem OpenRISC 1200

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  izr. prof. dr. Branko Šter

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Prof. dr. Branku Šteru, moji Ani, družini in prijateljem – za vso pomoč in prijazne besede ter preprosto zato, ker mi brez vas ne bi uspelo.

Kazalo

Seznam uporabljenih kratic in simbolov

CISC	Complex Instruction Set Computer – računalnik s širokim naborom ukazov
CPU	Central Processing Unit – centralna procesna enota
CS	Chip Select – Izbira pomnilnika
CSC	Chip Select Configuration register – register za konfiguracijo posameznega pomnilnika
DSP	Digital Signal Processing – digitalno procesiranje signalov
DWB	Data WishBone – podatkovni vmesnik na WISHBONE
GPR	General Purpose Register – splošno-namenski registri
HDL	Hardware Description Language – jezik za opis strojne opreme
IWB	Instruction WishBone – ukazni vmesnik na WISHBONE
LMR	Load Mode Register – naloži nastavitveni register
LRU	Least Recently Used – najmanj uporabljeni (podatki)
LSU	Load/Store Unit – enota za nalaganje in shranjevanje
MIPS	Million instructions per second – milijon operacij v sekundi
MMU	Memory Management Unit – enota za upravljanje s pomnilnikom
MSB	Most Significant Bit – najpomembnejši bit
NOP	NO Operation – brez operacije
PIC	Programmable Interrupt Controller – programirljivi prekinitveni krmilnik
PICMR	PIC Mask Register – maskirni register programirljivega prekinitvenega krmilnika
PICPR	PIC Priority Register– prioritetni register programirljivega prekinitvenega krmilnika
PICSR	PIC Status Register – statusni register programirljivega prekinitvenega krmilnika
POC	Power-On Configuration – konfiguracija ob zagonu
RISC	Reduced Instruction Set Computer – računalnik s skrčenim naborom ukazov
RMW	Read Modify Write – *beri-spremeni-zapiši*
RTL	Register Transfer Level – registerski nivo v hierarhičnem načrtovanju
SDRAM	Synchronous Dynamic Random Access Memory - sinhroni dinamični bralno-pisalni pomnilnik
TLB	Translation Lookaside Buffer – preslikovalni predpomnilnik
TMS	TiMing Select register – register za definiranje časov pomnilnika

Povzetek

V diplomski nalogi je opisana implementacija sistema, ki povezuje mikroprocesor OpenRISC 1200 in celični avtomat. Sistem je napisan v HDL (ang. hardware description language – jezik za opis strojne opreme) jeziku Verilog. Na začetku dela je opisana zgradba vseh modulov, ki jih vsebuje sistem, sledi opis delovanja sistema ter rezultati.

Glavni del sistema je skalarni mikroprocesor OpenRISC 1200, ki pri delovanju uporablja pomnilnik in celični avtomat. V pomnilniku so shranjeni ukazi, ki jih mikroprocesor bere. Celični avtomat je realiziran kot samostojen periferni modul. Mikroprocesor z ustreznim ukazom in ustreznim naslovom pošlje podatke celičnemu avtomatu, ki podatke obdela. Po končani obdelavi so podatki pripravljeni, da jih mikroprocesor prebere.

Implementacija temelji na vodilu WISHBONE, ki povezuje vse glavne elemente sistema. Ker sistem vsebuje dva gospodarja in dva sužnja je izbrana topologija vodila WISHBONE souporaba vodila. Izvedba vodila WISHBONE vsebuje logiko za izbiro gospodarja, ki temelji na signalu CYC, in logiko za izbiro sužnja, ki temelji na različnem naslovnem prostoru.

Ključne besede:

- mikroprocesor OpenRISC 1200
- pomnilniški krmilnik
- SDRAM
- celični avtomat
- vodilo WISHBONE

Abstract

The bachelor's thesis describes implementation of a system that connects the microprocessor OpenRISC 1200 and cellular automata. The system is written in the Verilog HDL language. At the beginning of the work the structure of the modules contained in the system is described. It is followed by a description of the system and the results.

The main part of the system is scalar microprocessor OpenRISC 1200, which uses the memory and cellular automata. In memory commands are stored, which are read by the microprocessor. The cellular automation is implemented as a separate peripheral module. The microprocessor, using an appropriate command and the corresponding address information, sends data to cellular automaton, which processes the data. After completing the processing, the data is ready to be read by the microprocessor.

The implementation is based on the WISHBONE bus, which connects all the main elements of the system. Since the system includes two masters and two slaves, the selected topology is the shared bus. The implementation includes a master-select logic for selecting the master, based on the CYC signal, and a slave-select logic for selection of a slave, based on a separate address space.

Key words:

- microprocessor OpenRISC 1200
- memory controller
- SDRAM
- cellular automata
- WISHBONE bus

1. Uvod

Mikroprocesorji so danes vgrajeni že skoraj v vsako elektronsko napravo, kar omogoča dejstvo, da je mikroprocesor zgrajen v enem integriranem vezju. Glavna naloga mikroprocesorja je izvajanje operacij. Hitrost izvajanja operacij mikroprocesorja se meri v milijonih operacij na sekundo, kar je tesno povezano s frekvenco mikroprocesorja. Frekvenca, s katero deluje mikroprocesor, se po navadi giblje od nekaj kHz pa tja do nekaj GHz. Podatek, ki je tudi zelo pomemben pri vsakem mikroprocesorju, je širina podatkovnega vodila, ki je običajno 64b ali 32b.

Vsak mikroprocesor ima svoj nabor ukazov. Zaporedje ukazov, ki jih naložimo v glavni pomnilnik, imenujemo program. Mikroprocesor po vrsti jemlje ukaze iz glavnega pomnilnika, jih interpretira in izvede.

V grobem mikroprocesorje delimo glede na število ukazov, ki jih zna mikroprocesor interpretirati in izvesti. Tako mikroprocesorje delimo na RISC (ang. reduced instruction set computer – računalnik s skrčenim naborom ukazov) in CISC (ang. complex instruction set computer – računalnik s širokim naborom ukazov) mikroprocesorje. Najbolj znani med CISC mikroprocesorji so mikroprocesorji proizvajalcev AMD in Intel.

Druga skupina mikroprocesorjev pa se imenuje RISC. Mikroprocesorji RISC imajo manjši nabor ukazov. Poleg tega se mikroprocesorji RISC razlikujejo od mikroprocesorjev CISC v tem, da se vsi ukazi izvršijo v eni urini periodi, razen ukazov tipa LOAD in STORE, in vsi ukazi morajo biti enako dolgi. Enako dolgi ukazi se kot prednost pokažejo predvsem v optimalni zapolnitvi cevovodov. V skupino mikroprocesorjev RISC spada tudi OpenRISC 1200.

Če želimo praktično uporabo celičnega avtomata, je potrebo celični avtomat povezati z nekim modulom, ki ga bo uporabljal. V nadaljevanju je prikazana implementacija, ki povezuje celični avtomat in mikroprocesor OpenRISC 1200. Mikroprocesor je torej glavni del sistema, ki bere ukaze iz pomnilnika in zahteva izvajanje celičnega avtomata.

Uporaba celičnega avtomata je praktično neomejena. Primeri uporabe celičnega avtomata:

- uporaba celičnega avtomata za klasifikacijo
- trojiško procesiranje na osnovi celičnih avtomatov
- uporaba celičnega avtomata za preučevanje organizacije sistemov
- modeliranje bioloških in kemičnih sistemov
- zanimiva je uporaba celičnega avtomata za simulacijo obnašanja požara v naravi

2. Predstavitev uporabljenih komponent

2.1. Verilog

Verilog je jezik HDL (ang. hardware description language – jezik za opis strojne opreme), ki se uporablja za opisovanje digitalnih sistemov, na primer mikroprocesorjev, pomnilnikov, krmilnikov itd. Tako lahko z uporabo HDL-a opišemo katerokoli digitalno vezje na kateremkoli nivoju. Z njim lahko opišemo enostavno pomnilniško celico, kot tudi zapletene sisteme, ki lahko vsebujejo tudi več kot milijon logičnih vrat.

Verilog omogoča načrtovanje digitalnih vezij na vedenjskem nivoju (visokonivojski opis), nivoju RTL (ang. register transfer language – registrski nivo), na nivoju logičnih vrat in na nivoju stikal. Verilog omogoča dva načina načrtovanja digitalnih vezij:

- metodologija *od spodaj navzgor* in
- metodologija *od zgoraj navzdol*.

Primer: implementacija pomnilniške celice D na nivoju RTL.

```
module d_ff(d,clk,q,q_bar);
      input d;
      input clk;
      output q;
      output q_bar;

      always @ (posedge clk) begin
            q <= d;
            q_bar <= ~d;
      end
endmodule
```

2.1.1. Metodologija *od spodaj navzgor*

Tradicionalna metoda načrtovanja digitalnih elektronskih vezij je *od spodaj navzgor*. Načrtovanje se začne na nivoju logičnih vrat. Z njihovo uporabo se sestavi digitalno vezje. Z naraščanjem kompleksnosti digitalnih vezij postane takšno načrtovanje skoraj nemogoče. Nove sisteme, ki vsebujejo na milijone tranzistorjev, je nemogoče zasnovati po tradicionalni metodologiji od spodaj navzgor. Zato je bilo potrebno to metodo zamenjati z novimi strukturnimi in hierarhičnimi načrtovanimi metodami.

2.1.2. Metodologija *od zgoraj navzdol*

Prava izbira za načrtovanje kompleksnih digitalnih vezij je torej metodologija *od zgoraj navzdol*. Metodologija *od zgoraj navzdol* omogoča testiranje v zgodnji fazi razvoja, enostavno uporabo različnih tehnologij, strukturiran način načrtovanja in še veliko drugih prednosti. Ker je čista metodologija *od zgoraj navzdol* težka za uporabo, se večina načrtovalcev odloči za kombinacijo obeh metodologij.

2.2. OpenRISC 1200

OpenRISC 1200 je 32-bitni skalarni procesor tipa RISC s harvardsko arhitekturo, ki spada v družino OpenRISC 1000. Vsebuje petstopenjski cevovod, podporo za virtualni pomnilnik in osnovne DSP (ang. digital signal processing – digitalno procesiranje signalov) zmogljivosti. Za predpomnilnik uporablja neposredno preslikan 8kB podatkovni predpomnilnik in neposredno preslikan 8kB ukazni predpomnilnik.

OpenRISC 1200 vsebuje enoto za razhroščevanje (ang. debug), časovnik, programirljivi prekinitveni krmilnik in podporo za upravljanje porabe. Če je procesor zgrajen v 18-mikronski tehnologiji, zmore 2,1 MIPS (ang. million instruction per second – milijon operacij v sekundi) pri frekvenci 300MHz. Namenjen je za uporabo v vgrajenih, prenosnih in omrežnih sistemih.

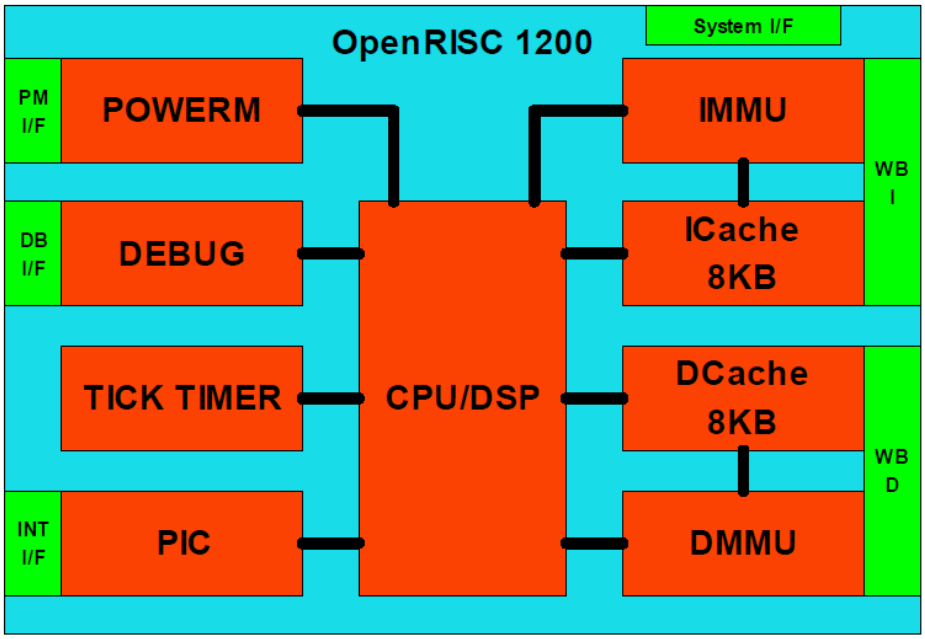
Nekatere lastnosti mikroprocesorja OpenRISC 1200:

- vse bistvene lastnosti jedra (IP core – intellectual property core) so uporabniško nastavljive,
- visoka zmogljivost,
- visoko zmogljivi predpomnilnik in MMU (ang. memory management unit – enota za upravljanje s pomnilnikom) ter
- WISHBONE-kompatibilni vmesnik.

2.2.1. Arhitektura

OpenRISC 1200 je sestavljen iz več elementov, ki so prikazani na Sliki 1:

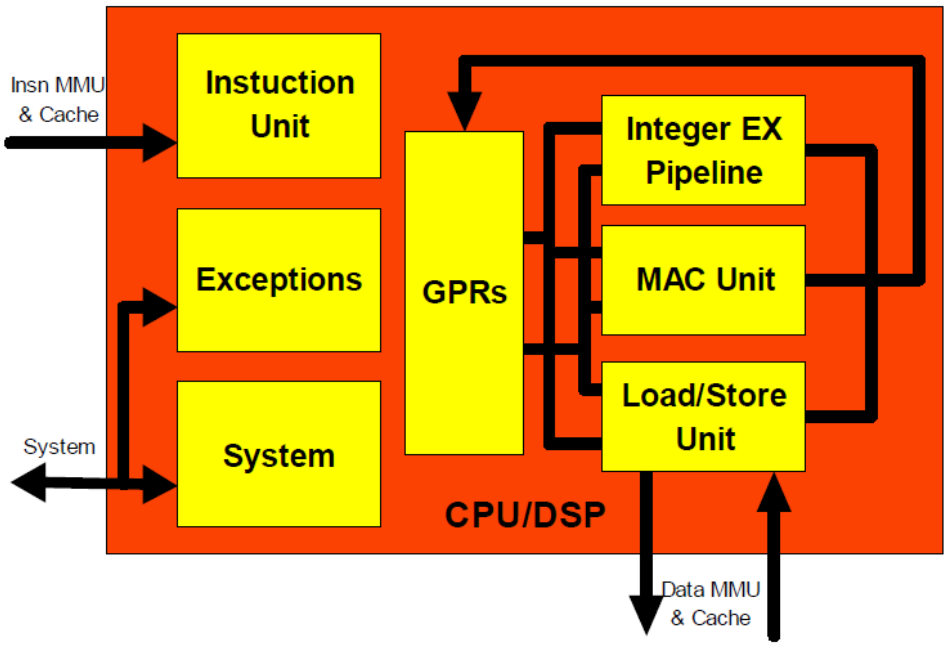
- centralni modul CPU/DSP,
- podatkovni predpomnilnik,
- ukazni predpomnilnik,
- enoto za upravljanje podatkovnega pomnilnika – DMMU (ang. data memory management unit),
- enoto za upravljanje ukaznega pomnilnika – IMMU (ang instruction memory management unit),
- enota za upravljanje porabe,
- časovni modul,
- modul za razhroščevanje in razvojni vmesnik,
- prekinitveni vmesnik in
- podatkovni in ukazni vmesnik na WISHBONE.



Slika 1. Arhitektura mikroprocesorja OpenRISC 1200

2.2.1.1. CPU/DSP

Glavni del procesorja je enota CPU/DSP. Blokovna shema centralne procesne enote je prikazana na Sliki 2. Centralna procesna enota je zasnovana tako, da lahko izvaja 32-bitne operacije. Operacij v plavajoči vejici in vektorskih operacij centralna procesna enota ne podpira.



Slika 2. Blokovna shema CPU/DSP

Ukazna enota implementira osnovni ukazni cevovod, ki dobavlja ukaze iz pomnilniškega podsistema in jih dostavlja izvršilni enoti. Hkrati vzdržuje zgodovino stanja, da zagotovi natančen model izjem in pravilno zaporedje izvajanja ukazov.

OpenRISC 1200 vsebuje 32 splošno-namenskih 32-bitnih registrov, ki so implementirani v modulu GPR (ang. general purpose registers – splošno namenski registri).

Enota za nalaganje in shranjevanje podatkov se imenuje LSU (ang. load/store unit – enota za nalaganje in shranjevanje). LSU prenaša vse podatke med splošno-namenskimi registri in notranjim vodilom centralne procesne enote. LSU je implementirana kot samostojna izvršilna enota, zato lahko zakasnitev v pomnilniškem podsistemu vpliva na glavni cevovod samo v primeru podatkovne odvisnosti. Ko LSU dobi ukaz *naloži* ali *shrani*, enota preveri, ali so prisotni vsi operandi.

Glavni izvršilni cevovod je implementiran za izvrševanje 32-bitnih operacij. Tipi operacij, ki jih lahko izvrši, so:

- aritmetične operacije,
- primerjalne operacije,
- logične operacije in
- pomikalne operacije.

CPU/DSP je povezan z zunanjim vodilom preko ukaznega in podatkovnega vmesnika. S signali, ki niso del ukaznega ali podatkovnega vodila, je CPU/DSP povezan preko sistemske enote. V sistemski enoti so implementirani tudi vsi registri za posebne namene.

Izjeme se generirajo, ko so izpolnjeni pogoji za določeno izjemo. Viri izjem, ki se lahko zgodijo v OpenRISC 1200, so:

- zunanja zahteva za prekinitev,
- določene napake pri dostopu do pomnilnika,
- notranje napake, kot je poizkus izvršitve nepoznanega ukaza,
- sistemski klici in
- notranja izjema, kot je zaustavitvena izjema (ang. breakpoint exception).

Za obravnavanje vseh tipov izjem je mehanizem enak. Ko se zgodi izjema, se nadzor prenese na krmilnik izjem. Vse izjeme se izvajajo v nadzorniškem načinu delovanja mikroprocesorja.

2.2.1.2. Predpomnilnik

Privzeta velikost predpomnilnika je 8KB. Predpomnilnik uporablja neposredno preslikavo naslovov, kar omogoča hiter dostop do podatkov. Velikost predpomnilnika je lahko poleg privzetih 8KB še 1KB, 2KB in 4KB.

OpenRISC 1200 je zgrajen po harvardski arhitekturi, kar pomeni, da ima ločen predpomnilnik za podatke in ukaze. Predpomnilnik je realiziran po algoritmu LRU (ang. least recently used – zamenjava podatkov, ki so najmanj uporabljeni). Predpomnilnik se lahko izklopi ali zaobide s pisanjem v registre za posebne namene.

Ob zgrešitvi se predpomnilnik napolni po načinu *najprej kritična beseda*, kar pomeni, da se iz glavnega pomnilnika najprej prenese ukaz ali podatek, ki je trenutno najbolj potreben. Prvi podatek ali ukaz se hkrati zapiše v predpomnilnik in istočasno prenese v ustrezno enoto, kar minimizira zakasnitve zaradi pisanja in branja iz predpomnilnika.

Podatkovni predpomnilnik dostavlja podatke v splošno-namenske registre na osnovi 32-bitne povezave z LSU. V LSU je implementirana vsa potrebna logika za izračunavanje ustreznih naslovov, za prenos podatkov iz in v podatkovni predpomnilnik in zagotavlja pravilen potek ukazov *naloži* in *shrani*. V pomnilnik lahko pišemo 8b, 16b ali 32b.

Ukazni predpomnilnik dostavlja ukaze izvršilni enoti preko 32-bitnega vmesnika na enoto za zajemanje ukazov. V enoti za zajemanje ukazov je implementirano računanje naslovov za različne ukaze.

2.2.1.3. MMU

OpenRISC 1200 ima implementirano upravljanje navideznega pomnilnika, ki omogoča zaščito dostopa in pretvarjanje naslovov.

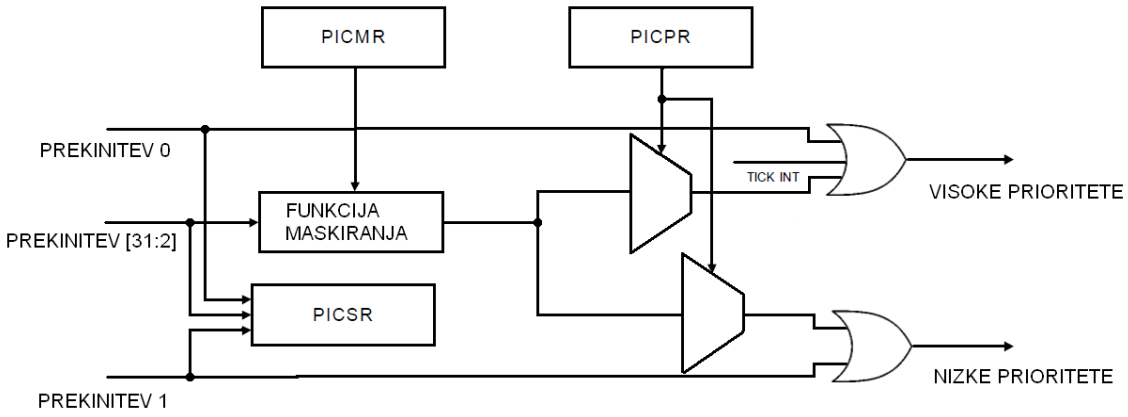
Ker gre za harvardsko arhitekturo procesorja, je podatkovna MMU ločena od ukazne. Velikost strani je 8kB. MMU zagotavlja tudi celovito zaščito strani.

2.2.1.4. Programirljivi prekinitveni krmilnik

Programirljivi prekinitveni krmilnik (PIC – ang. programmable interrupt controller) prejema zunanje zahteve in jih pošilja v CPU, kot visoko ali nizko prioritetne.

Prekinitveni krmilnik ima tri registre za posebne namene in 32 vhodov za prekinitve. Prekinitvena vhoda 0 in 1 sta vedno aktivna, prekinitveni vhod 0 je vedno povezan na visoko-prioritetne izjeme, vhod 1 pa je vedno povezan na nizko-prioritetne izjeme. Ostali prioritetni vhodi so maskirani ali povezani na visoko-prioritetne ali nizko-prioritetne izjeme.

Prekinitvene vhode se maskira s pisanjem v register PICMR (ang. PIC mask register – maskirni register programirljivega prekinitvenega krmilnika). Določanje prioritete prekinitev (visoka ali nizka prioriteta) se izvede s programiranjem registra PICPR (ang. PIC priority register – prioritetni register programirljivega prekinitvenega krmilnika). Register PICSR (ang. PIC status register – statusni register programirljivega prekinitvenega krmilnika) se uporablja za določanje statusa vsake prekinitve. Na Sliki 3 je prikazana blokovna shema programirljivega prekinitvenega krmilnika in vsi registri za posebne namene, ki jih krmilnik uporablja.



Slika 3. Blokovna shema prekinitvenega krmilnika

2.2.1.5. Časovnik

OpenRISC 1200 vsebuje tudi časovnik, ki ga krmili zunanji urin signal. Uporablja ga operacijski sistem, ki teče na mikroprocesorju, za natančno merjenje časa in razvrščanje sistemskih opravil.

Omejitve, ki veljajo za časovnik:

- maksimalna dolžina merjenega časa je 2^{32} urinih period,
- maksimalna časovna perioda med dvema prekinitvama je 2^{28} urinih period in
- maskirana prekinitev časovnika.

Mikroprocesor ima implementiran vmesnik za upravljanje porabe, ki vsebuje različne signale za zniževanje frekvence delovanja mikroprocesorja.

2.2.1.6. Upravljanje porabe

Zaradi optimizacije porabe OpenRISC 1200 omogoča delovanje v načinu nizke porabe. V tem načinu se lahko procesor uporabi za dinamično aktiviranje in deaktiviranje določenih notranjih modulov.

OpenRISC 1200 ima tri glavne načine delovanja za zmanjšanje porabe:

- slow mode (procesor deluje pri nižji frekvenci)
- doze mode (izvajanje programske kode v jedru se odloži)
- sleep mode (vse notranje enote procesorja so onemogočene)

2.2.1.7. Vmesnik na WISHBONE

Dva vmesnika na WISHBONE povezujeta OpenRISC 1200 in zunanje naprave, in sicer 32-bitni vmesnik za ukaze in 32-bitni vmesnik za podatke.

2.3. Vodilo WISHBONE

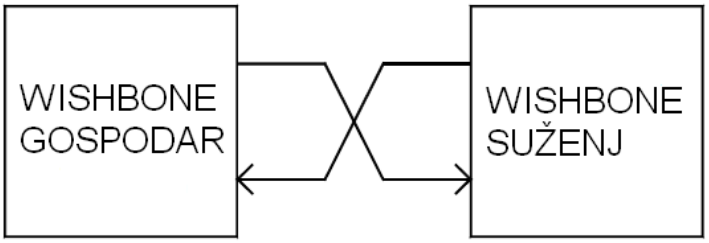
Vodilo WISHBONE je odprtokodna rešitev za povezovanje različnih digitalnih vezij. Namen tega vodila je povezovanje različnih modulov (IP cores) med seboj znotraj enega čipa. Veliko odprtokodnih rešitev uporablja vodilo WISHBONE.

Vodilo uporablja arhitekturo *gospodar/suženj*, kar pomeni, da ima gospodar nadzor nad vodilom, suženj pa se temu podreja. Za povezovanje med gospodarjem in sužnjem obstajajo različni načini. Najenostavnejši način povezovanja je *točka na točko* (ang. point-to-point). Bolj napredne oblike povezovanja so povezava s tokom podatkov (ang. data flow), souporaba vodila (ang. shared bus) in križna povezava (ang. crossbar switch).

2.3.1. Topologije vodila WISHBONE

2.3.1.1. Povezava *točka na točko*

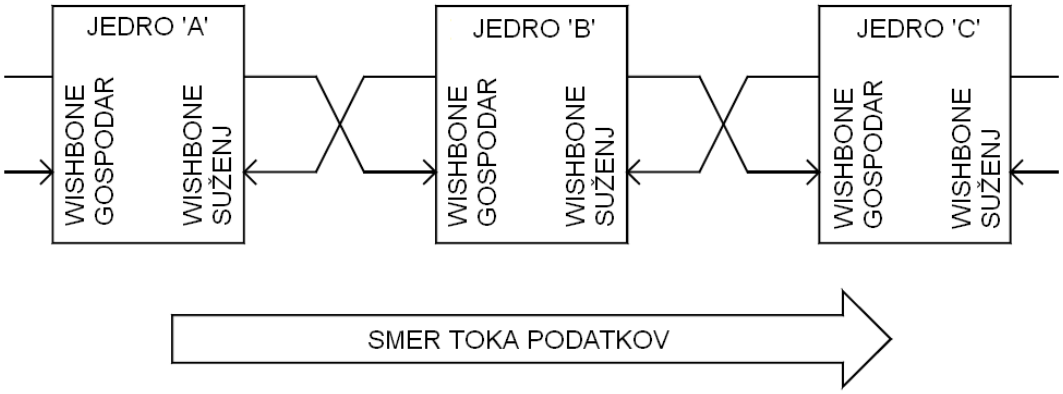
Povezava *točka na točko* (ang. point-to-point) je najenostavnejša oblika povezovanja, ki jo prikazuje Slika 4. Vsebuje enega gospodarja in enega sužnja, ki komunicirata s pomočjo WISHBONE vodila. Primer povezave točka na točko je povezava med mikroprocesorjem in vhodno/izhodnim vmesnikom.



Slika 4. Topologija *točka na točko*

2.3.1.2. Povezava s tokom podatkov

Povezava s *tokom podatkov* (ang. data flow) je uporabna takrat, kadar se podatki obravnavajo v zaporednem vrstnem redu. Povezavo s *tokom podatkov* prikazuje Slika 5. V tem primeru vsebuje vsak modul v sistemu vmesnik gospodarja in sužnja. Podatki se pretakajo od jedra do jedra, temu rečemo tudi cevovod. Arhitektura s tokom podatkov izkorišča paralelizem, kar pospeši izvajalni čas.

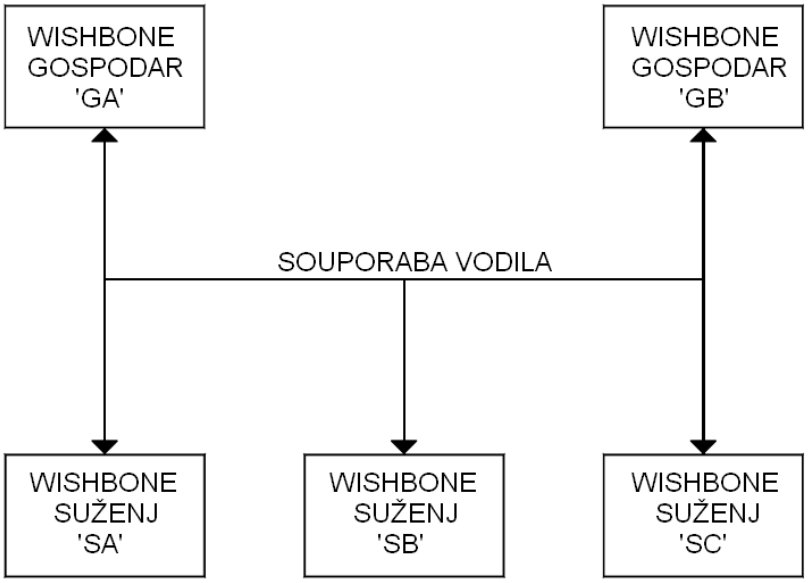


Slika 5. Topologija povezave s tokom podatkov

2.3.1.3. Souporaba vodila

Souporaba vodila (ang. shared bus) je uporabna pri sistemih, ki vsebujejo dva ali več gospodarjev in enega ali več sužnjev. Souporabo vodila prikazuje Slika 6. Pri omenjeni topologiji gospodar vodila začne cikel za določenega sužnja, suženj nato odgovarja gospodarju en ali več ciklov. Povezovalna logika določi, kateri gospodar trenutno zaseda vodilo.

Glavna prednost souporabe vodila je, da vsebuje manj logičnih vrat in manj usmerjevalnih virov kot drugi sistemi. Glavna slabost omenjene arhitekture je čakanje gospodarja na dostop do vodila.

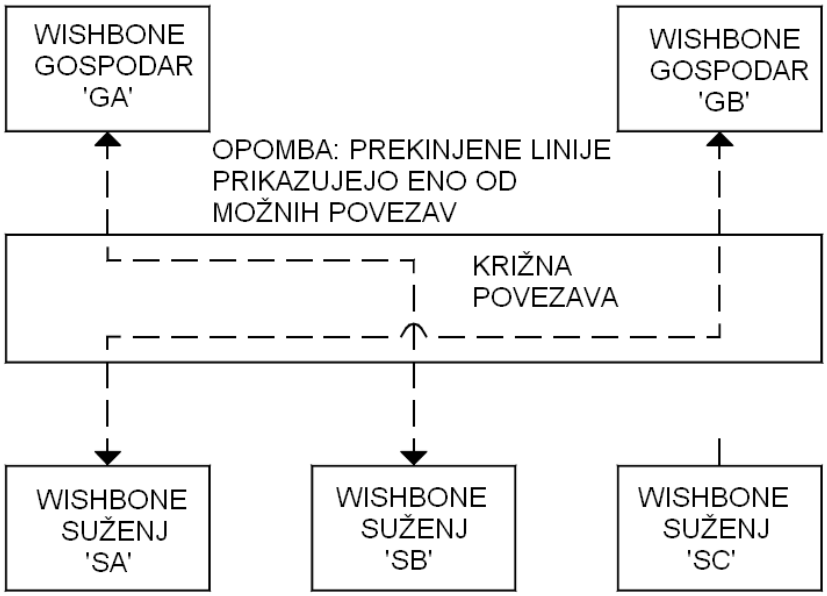


Slika 6. Topologija souporabe vodila

2.3.1.4. Križna povezava

Topologija križne povezave (ang. crossbar switch) se uporablja za povezovanje dveh ali več gospodarjev za hkratni dostop do dveh ali več sužnjev. Topologijo križne povezave prikazuje Slika 7. Povezovalna logika določi kdaj ima gospodar dostop do določenega sužnja. V nasprotju s souporabo vodila, križna povezava omogoča hkratno komunikacijo več gospodarjev z več sužnji pod pogojem, da dva gospodarja ne dostopata do istega sužnja v istem trenutku.

Pri metodi križne povezave vsak gospodar zaprosi za svoj kanal na povezavi. Ko je povezava ustvarjena, se začne prenos podatkov po privatni povezavi med gospodarjem in sužnjem.



Slika 7. Topologija križne povezave

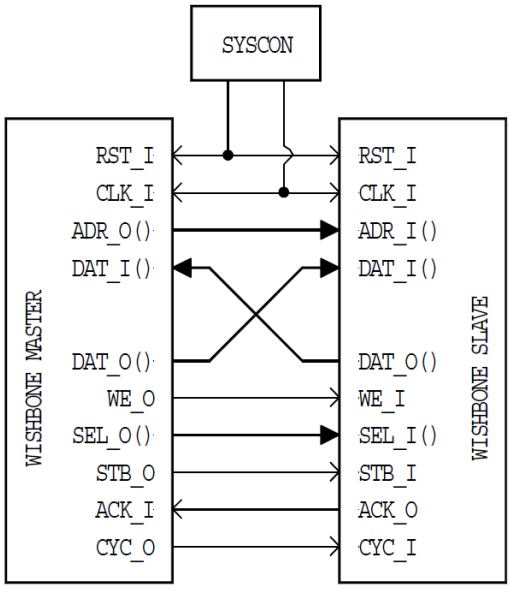
2.3.2. Nabor signalov, ki jih uporablja vodilo WISHBONE

Vodilo WISHBONE uporablja množico signalov za povezovanje gospodarjev in sužnjev. Na Sliki 8 je prikazana osnovna povezava gospodarja in sužnja.

Množica signalov, ki jih uporablja vodilo WISHBONE:

- CLK – urin signal, ki je skupen za gospodarje in sužnje na vodilu,
- RST – signal reset (brisanje), po katerem vse naprave na vodilu preidejo v začetno stanje,
- DAT – podatkovno vodilo, po katerem se prenašajo podatki,
- ACK – signal, s katerim suženj sporoči gospodarju, da so podatki na podatkovnem vodilu pripravljeni,
- ADR – vodilo, s katerim gospodar sporoči sužnju (npr. CPU pomnilniku), na katerem naslovu se nahajajo podatki,
- CYC – signal, s katerim sporoči gospodar vodila veljaven cikel na vodilu (lahko več operacij v enem ciklu),

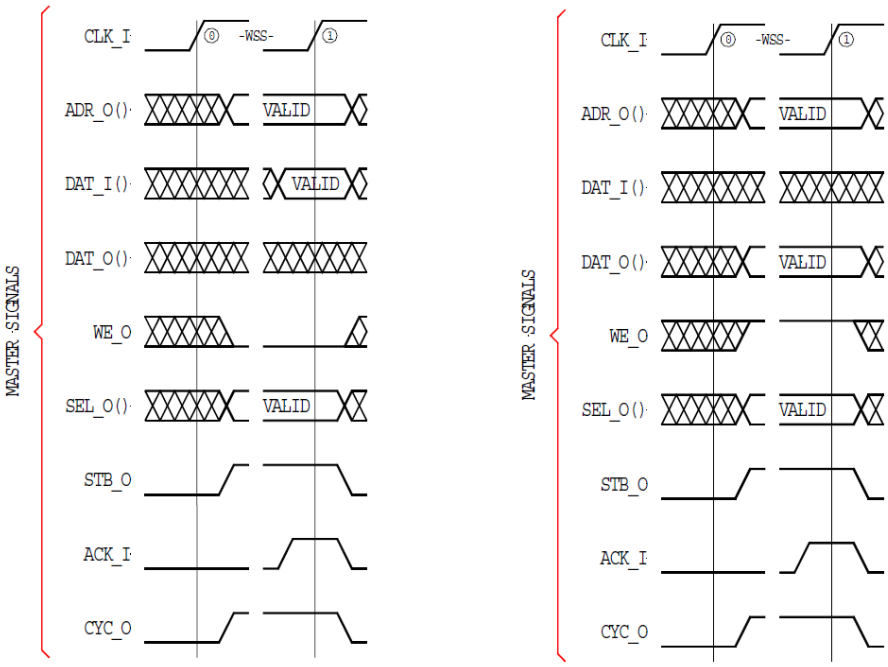
- ERR – signal za sporočanje napak pri prenosu ali prekinitve prenosa,
- LOCK – signal, s katerim sporoči gospodar vodila, da se trenutni cikel ne sme prekiniti (drugi gospodarji morajo počakati),
- RTY – signal, s katerim dobi gospodar vodila informacijo, da suženj ne more ali ni pripravljen sprejeti informacije,
- SEL – signal, s katerim gospodar vodila sporoči, kateri bajti na podatkovnem vodilu so veljavni,
- STB – signal, s katerim gospodar sporoči veljaven prenos enega podatka v ciklu,
- TGA – signal, s katerim gospodar vodila sporoči dodatne informacije v zvezi s naslovom,
- TGD – signal uporabljata gospodar in suženj in vsebuje dodatne informacije o podatkovni liniji,
- TGC – signal, ki vsebuje dodatne informacije o krmilnih signalih in
- WE – signal, s katerim gospodar vodila sporoči sužnju, ali gre za sprejem ali pošiljanje podatkov.



Slika 8. Standardna povezava med gospodarjem vodila in sužnjem

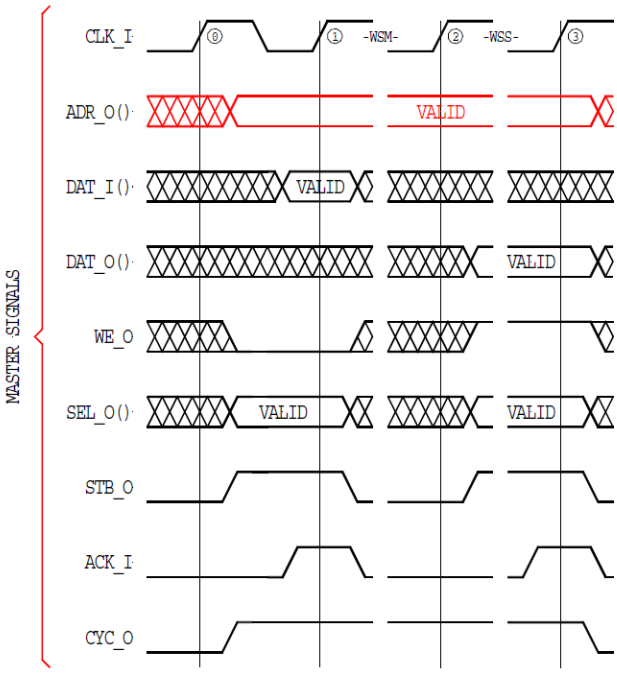
2.3.3. Delovanje

Gospodar in suženj sta povezana z množico signalov, ki omogočajo komunikacijo med njima. Tem signalom sta dodana še signala CLK in RST, ki definirata urine cikle in signal reset. Najosnovnejši operaciji na vodilu so branje in pisanje v pomnilnik. Prikazuje ju Slika 9.



Slika 9. Enojni cikel branja in enojni cikel pisanja

Omogočen je tudi cikel RMW (Slika 10), ki se uporablja za nedeljive semafor operacije. Med prvo polovico cikla se zgodi enojno branje podatkov. Med drugo polovico cikla pa se podatki zapišejo v pomnilnik. Signala ADR in CYC sta veljavna skozi obe polovici cikla.

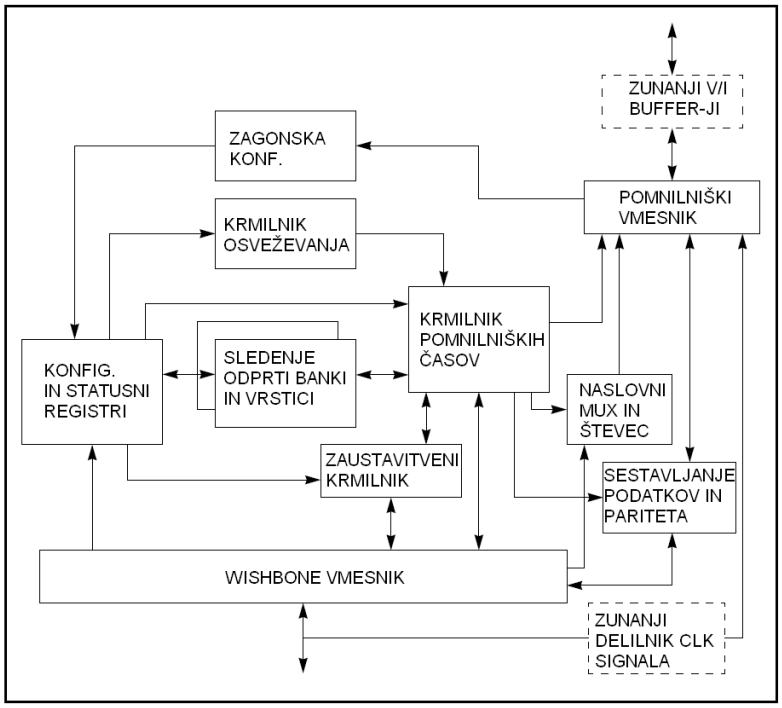


Slika 10. Cikel RMW

2.4. Pomnilniški krmilnik

2.4.1. Arhitektura

Arhitekturi pomnilniškega krmilnika, ki je prikazana na Sliki 11, je potrebno dodati še zunanji delilnik urinega signala (pomnilniški vmesnik uporablja polovično frekvenco urinega signala vodila WISHBONE) in zunanje vhodno-izhodne buffer-je.



Slika 11. Arhitektura pomnilniškega krmilnika

2.4.1.1. Vmesnik na WISHBONE

Pomnilniški krmilnik vsebuje za komunikacijo z drugimi napravami vmesnik na vodilo WISHBONE. Uporabljena različica podpira le 32-bitno širino vodila. Vmesnik opravi enostavno dekodiranje WISHBONE signalov, ki odloči med pisanjem in branjem, ali gre za prvi prenos podatkov v ciklu in ali gre za stanje čakanja.

2.4.1.2. POC

Register POC (ang. power-on configuration – konfiguracija ob zagonu) prebere pomnilniške podatkovne linije med ciklom reset. Prebrana vrednost določi začetno konfiguracijo POC. Tabela 1 prikazuje pomen nekaterih bitov v registru POC.

POC[3:2]	POC[1:0]	Opis
		Podatkovna širina
--	00	8 bit
--	01	16 bit
--	10	32 bit
--	11	Rezervirano
		Tip naprave
00	--	Onemogočeno
01	--	SSRAM
10	--	Asinhrona naprava
11	--	Sinhrona naprava

Tabela 1. Konfiguracija POC

2.4.1.3. Krmilnik za osveževanje

Krmilnik za osveževanje je pomemben za osveževanje pomnilnikov DRAM (ang. dynamic random access memory – dinamični bralno-pisalni pomnilnik). Vsi pomnilniki tipa SDRAM (ang. synchronic DRAM – sinhroni dinamični bralno-pisalni pomnilnik) so osveženi istočasno. Če uporabimo pomnilnike z različnimi dolžinami intervalov za osveževanje, je uporabljen najkrajši interval za vse pomnilnike.

2.4.1.4. Naslovni multipleksor in števec

Naslovni multipleksor je zadolžen za generiranje ustreznega naslova. Generiranje naslova je razdeljeno v tri bloke: generiranje naslova za pomnilniki tipa SRAM, asinhrone pomnilnike in pomnilnike tipa SDRAM. Glede na POC konfiguracijo sistema se zbere ustrezen blok za generiranje naslova.

2.4.1.5. Blok za prenos podatkov

Blok za prenos podatkov je odgovoren za prenos podatkov med pomnilnikom in vodilom WISHBONE. Podatki, ki so namenjeni iz vodila WISHBONE v pomnilnik, se enostavno zapahnejo ob primernem času. Za preverjanje prenosa se uporablja paritetni generator. Podatki, ki so namenjeni iz pomnilnika na vodilo WISBONE, gredo najprej skozi podatkovno pakiranje. Podatkovno pakiranje sestavi 32-bitne podatke iz morebitnih 8- ali 16-bitnih.

2.4.1.6. Pomnilniški časovni krmilnik

Časovni krmilnik se uporablja za določevanje časov v pomnilniškem krmilniku in za nadzor. Generira primerne cikle za dostopanje do različnih pomnilnikov, kot je definirano v nastavitvenih registrih. Časovni krmilnik tudi generira ustrezne signale za nadzor nad ostalimi bloki.

2.4.1.7. Pomnilniški vmesnik

Pomnilniški vmesnik nudi enostavno sinhronizacijo za vhodno-izhodne signale. Vsi izhodi in vhodi so sinhronizirani na prednjo fronto urinega signala.

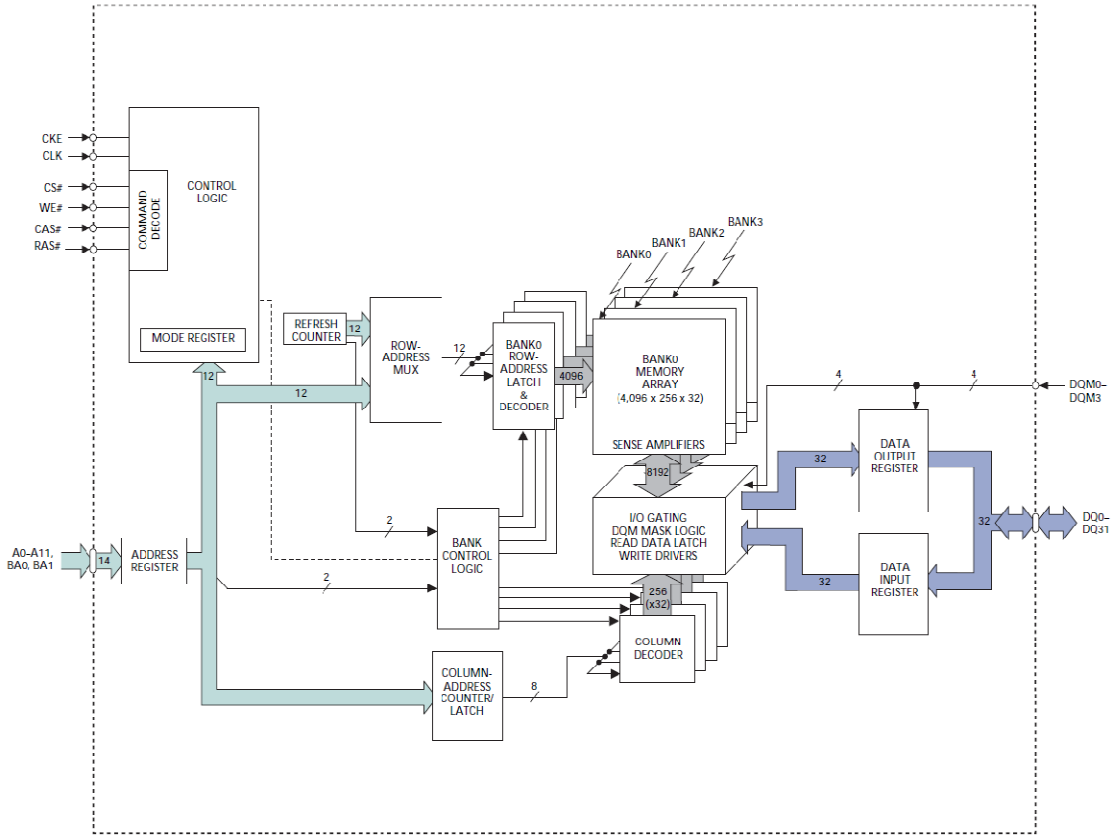
2.4.1.8. Nastavitveni in statusni registri

Vsi registri, ki jih uporablja pomnilniški krmilnik, so vsebovani v enem bloku. Uporablja se tudi za generiranje zahteve po posodobitvi nastavitvenega registra znotraj pomnilnika in za dekodiranje signala CS (ang. chip select – izbira pomnilnika).

2.5. Glavni pomnilnik

Slika 12 prikazuje glavni pomnilnik SDRAM. Kapaciteta uporabljenega pomnilnika je 128Mb. V sistemu je uporabljen model pomnilnika MT48LC4M32B2 [5].

Branje in pisanje po pomnilniku se začne na neki lokaciji in se nadaljuje za določeno število zaporednih lokacij (ang. burst). Dostop do pomnilnika se začne z ukazom ACTIVE, ki povzroči izbiro določene banke in vrstice znotraj banke, in se nadaljuje z branjem ali pisanjem v pomnilnik (ukaza tipa LOAD in STORE).



Slika 12. Blokovni diagram glavnega pomnilnika (SDRAM)

Pred normalnim delovanjem je potrebno SDRAM prednastaviti. Prednastavitev mora potekati v točno določenih korakih in v okviru predvidenih časovnih konstant, drugače se lahko SDRAM postavi v nedefinirano stanje.

Nastavitveni register

Nastavitveni register se uporablja za nastavitve različnih načinov delovanja pomnilnika. Programiranje registra je implementirano s posebnim ukazom in s pomočjo naslovnega vodila, iz katerega se prebere vrednost nastavitvenega registra.

2.6. Celični avtomat

Celični avtomati so diskretni dinamični sistemi, katerih obnašanje je popolnoma določeno z lokalnimi relacijami. Prostor, ki ga celični avtomati uporabljajo, je predstavljen z mrežo. Čas teče v diskretnih korakih in v vsakem koraku centralna celica posodobi svoje stanje glede na stanja bližnjih sosedov.

Definicija celičnega avtomata na dvodimenzionalni mreži (D=2) [1]:

- $I \times I$ določa prostor celičnega avtomata, $I \subset \mathbb{N}$
- sosednost določa funkcija $g(\alpha)$:

$$g(\alpha) = \{\alpha + \delta_1, \alpha + \delta_2, \dots, \alpha + \delta_n\};\ \alpha \in I \times I, \delta_i \in I \times I \tag{1}$$

  opazovana celica je ena iz sosednosti; običajno ima $\delta_1 = (0,0)$

- celica celičnega avtomata je končni avtomat (KA), podan z (V, v_0, f)

  V – množica stanj
  v_0 – začetno stanje
  f – tabela prehajanja stanj (operator prehajanja stanj)

- stanja celic v sosednosti (skupaj s stanjem opazovane celice α) določajo sosednostno stanje (v času *t*):

$$h^t(\alpha) = \left(v^t(\alpha + \delta_1), v^t(\alpha + \delta_2), \dots, v^t(\alpha + \delta_n)\right) \tag{2}$$

- stanje celic v času $t + 1$:

$$v^{t+1}(\alpha) = f\left(h^t(\alpha)\right) \tag{3}$$

- funkcija prehajanja stanj f, ki jo imenujemo tudi pravilo celičnega avtomata, je podana s tabelo vseh možnih parov oblike

$$\left(h^t(\alpha), v^{t+1}(\alpha)\right) \tag{4}$$

- konfiguracija je dovoljena kombinacija vseh stanj celic v celičnem avtomatu

- globalna prenosna funkcija F

$$F(c)(\alpha) = f\big(h(\alpha)\big), \text{ za vse } \alpha \in I \times I \quad (5)$$

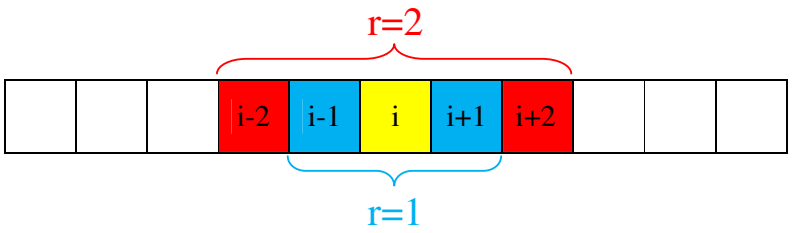
$c \in C$ je množica vseh konfiguracij

F določa sekvenco konfiguracij $c_0, c_1, \ldots, c_t, \ldots$ oziroma $c_{t+1} = F(c_t)$

Celični avtomat je torej prostorska mreža N celic. Vsaka celica je v vsakem trenutku t v enem od k stanj. Za posodobitev svojega stanja vsaka od N celic sledi določenemu pravilu. Za določitev stanja neke celice v naslednjem trenutku, je potrebno poznavanje trenutnega stanja celice in stanja določenih sosednjih celic. Vsaka celica ima svoje pravilo za naslednje stanje, ki temelji na vrednosti celic v njeni okolici. Ko so uveljavljena pravila za vse celice v mreži, nastane nova generacija celic.

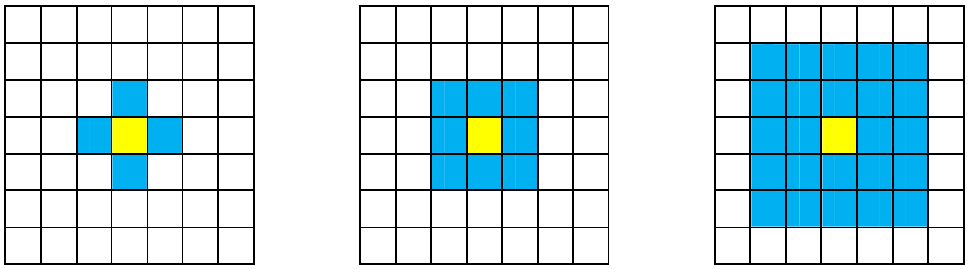
2.6.1. Soseščina

V enodimenzionalnem celičnem avtomatu je soseščina določene celice sestavljena iz same celice in r sosedov na vsaki strani. Slika 13 prikazuje dve različni soseščini v enodimenzionalnem celičnem avtomatu.



Slika 13. Dve različni soseščini v enodimenzionalnem celičnem avtomatu

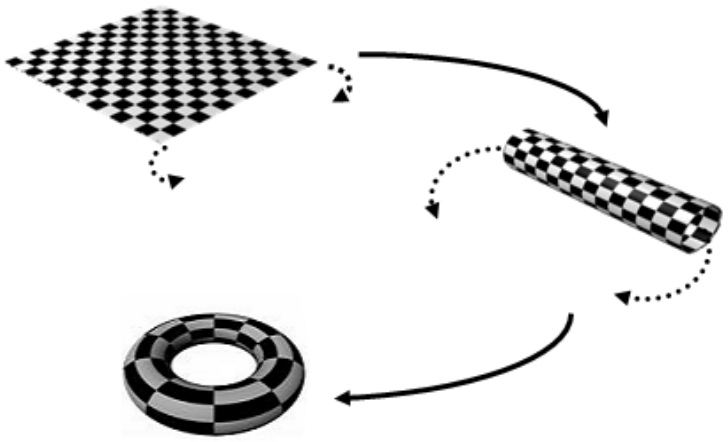
V dvodimenzionalnem celičnem avtomatu se stanje celice prav tako računa s sosednjimi celicami (na Sliki 14 obarvano modro). Obstaja več različnih soseščin za dvodimenzionalne celične avtomate. Von Neumannova soseščina uporablja za pridobitev stanja celice štiri sosednje celice, poimenovane po straneh neba. Moorova soseščina uporablja vseh osem sosednjih celic za pridobitev stanja centralne celice. Obstaja pa tudi razširjena Moorova soseščina.



Slika 14. Različne soseščine v dvodimenzionalnem celičnem avtomatu

2.6.2. Mejne celice

Pri celičnih avtomatih se pojavlja vprašanje, kaj storiti v primerih robnih celic. Ponujata se nam dve možnosti. Prva možnost je, da za vse robne celice definiramo nova pravila z manj sosedi. Druga možnost je, da manjkajoče sosednje celice nadomestimo s pripadajočimi celicami na drugi strani. Z upoštevanjem drugega pravila se nam pri enodimenzionalnem celičnem avtomatu oblikuje krog, v dvodimenzionalnem pa svitek oz. torus. Slika 15 prikazuje postopek zvijanja dvodimenzionalnega polja v svitek, da imajo vse celice 8 sosednih celic.



Slika 15. Zvijanje dvodimenzionalne mreže celičnega avtomata v svitek oz. torus

2.6.3. Simulacija

Način simulacije celičnega avtomata si lahko predstavljamo s pomočjo neskončnega lista mrežastega papirja in množico pravil za določitev naslednje generacije celic. Vsak kvadratek na papirju predstavlja eno celico in lahko zavzame dve stanji. Če upoštevamo za okolico npr. Moorovo soseščino dobimo osem sosednjih celic. Z upoštevanjem, da v pravilo za generacijo centralne celice v naslednjem trenutku spada tudi centralna celica sama, dobimo 512 (2^9) različnih stanj sosednosti.

Za simulacijo enodimenzionalnega celičnega avtomata pri r=1 uporabimo za določanje pravila centralno celico in njenega levega in desnega soseda. Tako nam pravilo generira 8 (2^3) različnih stanj sosednosti. Pravila običajno poimenujemo po desetiški vrednosti, ki jo predstavlja pravilo.

Primer pravila 232 za določanje vrednosti celice v enodimenzionalnem avtomatu prikazuje Tabela 2:

Celice			
i-1	i	i+1	i(t+1)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabela 2. Pravilnostna tabela pravila 232

Binarni niz 11101000 predstavlja število 232, ki ga izračunamo z izrazom
$128\times1+64\times1+32\times1+16\times0+8\times1+4\times0+2\times0+1\times0 = 232$.

2.6.4. Razvrstitev celičnih avtomatov

Stephan Wolfram je definiral štiri enostavne razrede [2], v katere se lahko uvrsti celične avtomate glede na njihovo obnašanje.

1. **Razred**: v ta razred uvrščamo sisteme, pri katerih se iz kateregakoli začetnega vzorca razvije stabilno stanje. Vsi naključni vzorci v začetnem stanju izginejo. Primer je pravilo 232 v enodimenzionalnem celičnem avtomatu.
2. **Razred**: v ta razred spadajo sistemi, pri katerih se iz naključnega začetnega stanja razvije periodično ponavljajoče se stanje. Nekateri naključni vzorci začetnega stanja izginejo, nekateri pa lahko ostanejo. Lokalne spremembe v začetnem vzorcu ostanejo lokalne.
3. **Razred**: v tem razredu so sistemi, pri katerih se iz naključnega začetnega stanja razvijejo naključni in aperiodični vzorci. Vsako stabilno stanje, ki se pojavi na začetku, hitro izgine, medtem ko se lokalne spremembe v začetnem vzorcu porazdelijo globalno. Primer aplikacije v tem razredu je generator naključnih števil.
4. **Razred**: ta razred vsebuje sisteme, pri katerih naključno začetno stanje po končnem številu korakov včasih izumre, možen pa je tudi nastanek nekaterih stabilnih periodičnih vzorcev. Za doseganje omenjenih rezultatov je potrebno veliko število korakov.

Kot pri vsaki klasifikaciji v razrede obstajajo primeri, ki jih lahko z upoštevanjem različnih definicij uvrstimo v več razredov. Tudi pri celičnih avtomatih obstajajo določena pravila, pri katerih celični avtomat vsebuje lastnosti, ki spadajo v več razredov.

3. Implementacija

3.1. Povezava vseh komponent v celoten sistem

Celoten sistem vsebuje:

- mikroprocesor OpenRISC 1200,
- modul SYS_CON za generiranje urinega in reset signala,
- vodilo WISHBONE,
- pomnilniški krmilnik (ang. memory controller),
- modul MC_BUF, v katerem so implementirani pomnilniški buffer-ji,
- modul RAM_LOGIC za povezavo pomnilniškega krmilnika in SDRAM-a,
- pomnilnik SDRAM in
- celični avtomat.

Glavna enota celotnega sistema je skalarni mikroprocesor OpenRISC 1200. Njegova naloga je, da ukaze, ki jih prebere iz glavnega pomnilnika (SDRAM), interpretira in izvršuje. Mikroprocesor komunicira z ostalimi komponentami v sistemu preko dveh WISHBONE vmesnikov. Vmesnik IWB (ang. instruction wishbone – ukazni WISHBONE vmesnik na mikroprocesorju OpenRISC 1200) je namenjen za zajemanje ukazov iz glavnega pomnilnika, DWB (ang. data wishbone – podatkovni WISHBONE vmesnik na mikroprocesorju OpenRISC 1200) pa za branje in pisanje podatkov iz in v pomnilnik.

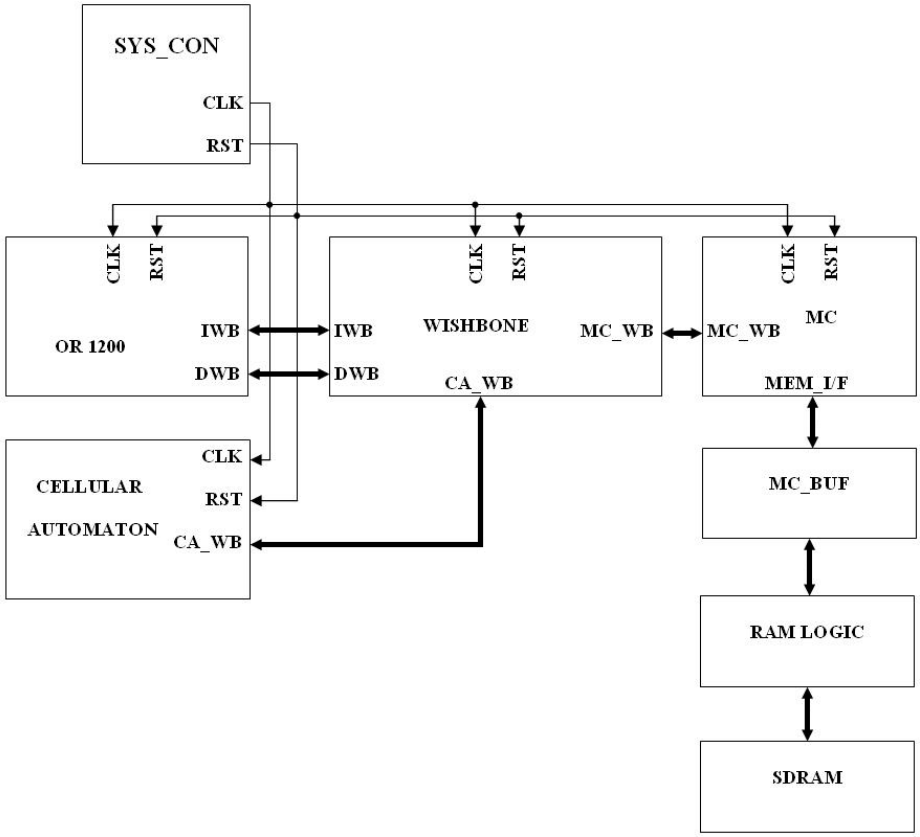
Enota, ki skrbi za interpretacijo signalov in prenos podatkov med OpenRISC 1200 in glavnim pomnilnikom, se imenuje pomnilniški krmilnik (ang. memory controller). Pomnilniški krmilnik je potreben tudi zaradi različnih hitrosti delovanja mikroprocesorja in glavnega pomnilnika. Pomnilniški krmilnik mora upoštevati vse časovne omejitve, ki jih narekuje glavni pomnilnik.

Sistem vsebuje tudi modul *celični avtomat*, ki paralelno izvaja korake celičnega avtomata. Mikroprocesor z ukazom STORE in ustreznim naslovom pošlje začetno stanje, število korakov in pravilo celičnemu avtomatu. Ko mikroprocesor potrebuje rezultat celičnega avtomata, z ukazom LOAD od njega zahteva rezultat.

Vsi trije moduli (OpenRISC 1200, pomnilniški krmilnik in celični avtomat) so povezani z vodilom WISHBONE, z uporabo topologije *souporaba vodila*.

V glavnem pomnilniku so shranjeni ukazi in podatki, ki jih potrebuje mikroprocesor. Mikroprocesor začne najprej brati ukaze iz lokacije v glavnem pomnilniku, kamor kaže reset vektor. Ko mikroprocesor prebere ukaz tipa LOAD ali STORE, prebere ali shrani podatke na lokacijo v glavnem pomnilniku, kamor kaže pripadajoči naslov.

V sistemu se pojavi tudi modul SYS_CON, ki generira urin in reset signal. Slika 16 prikazuje blokovno shemo celotnega sistema.



Slika 16. Blokovna shema celotnega sistema

3.1.1. Implementacija vodila WISHBONE

Glavna naloga vodila WISHBONE je povezava mikroprocesorja OpenRISC 1200 z ostalimi komponentami. Razlogi za izbiro topologije souporabe vodila so:

- dva gospodarja vodila,
- dva sužnja na vodilu in
- enostavna implementacija.

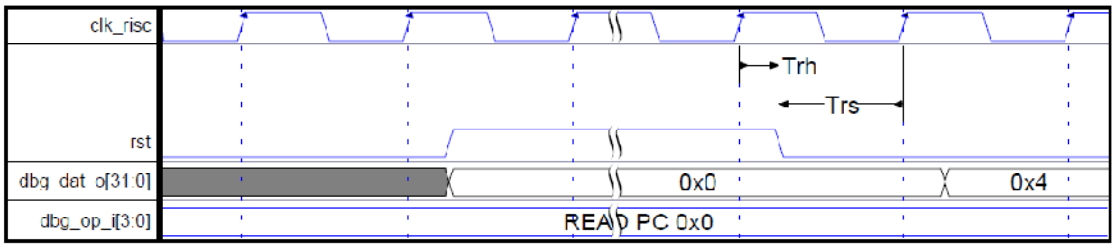
Glavna težava pri implementaciji vodila WISHBONE je logika MASTER_SELECT in logika SLAVE_SELECT. Za razumevanje delovanja omenjene logike, moramo najprej razumeti delovanje vseh komponent v sistemu.

3.1.1.1. Delovanje vmesnika na WISHBONE v mikroprocesorju OpenRISC 1200

Mikroprocesor OpenRISC 1200 je zgrajen po principu harvardske arhitekture, zato vsebuje dva WISHBONE vmesnika. Vmesnik IWB je namenjen zajemanju ukazov iz pomnilnika, vmesnik DWB pa je namenjen prenosu podatkov.

Mikroprocesor OpenRISC 1200 začne delovanje z reset sekvenco, ki jo prikazuje Slika 17. Za delovanje mikroprocesorja OpenRISC 1200 sta ključna dva signala, ki ju generira modul SYS_CON. Glavni signal, ki ga mikroprocesor potrebuje za delovanje, je urin signal. Z

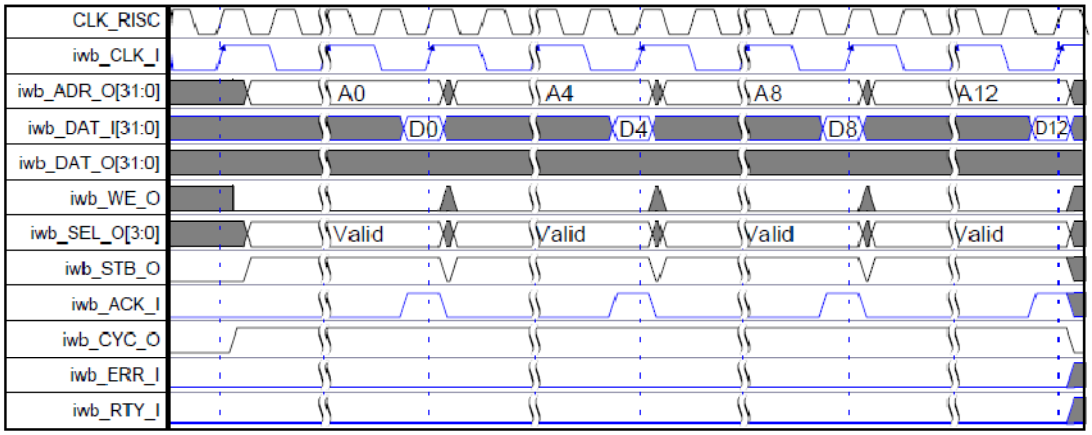
urinim signalom je določena frekvenca delovanja mikroprocesorja. S frekvenco je določena tudi urina perioda. Vse spremembe v mikroprocesorju so sinhronizirane na prednjo fronto urinega signala. Signal reset, ki ga generira modul SYS_CON povzroči reset vseh pomnilniških celic v mikroprocesorju (procesor preide v začetno stanje). Po signalu reset se resetira programski števec, ki nato kaže na reset vektor 0x100. Na tej lokaciji se nahaja prvi ukaz, ki ga mikroprocesor prebere.



Slika 17. Postopek brisanja (reset) pri OpenRISC 1200

Po končanem brisanju, postavi OpenRISC 1200 na IWB vmesnik veljavni naslov in vse ostale potrebne signale. Mikroprocesor začne cikel s signaloma CYC in STB v visokem stanju. Cikel na vmesniku WISHBONE traja toliko časa, dokler je signal CYC v visokem stanju.

Na Sliki 18 je prikazan uspešen prenos ukazov na ukazno vodilo WISHBONE mikroprocesorja OpenRISC 1200.



Slika 18. Uspešen prenos bloka štirih ukazov

3.1.1.2. Delovanje logike MASTER_SELECT

Ker topologija souporabe vodila dovoljuje uporabo enega kanala za prenos podatkov v danem trenutku, mikroprocesor ne more hkrati brati ukaza in pisati podatke. Iz tega razloga je v vodilu WISHBONE implementirana logika MASTER_SELECT, ki odloča, kateri vmesnik WISHBONE na mikroprocesorju ima dostop do vodila.

Logika MASTER_SELECT (Tabela 3) vsebuje 2-bitni register MASTER_SELECT. Logika MASTER_SELECT vpisuje vrednosti v omenjeni register in s tem dovoljuje dostop vmesniku DWB ali IWB dostop do vodila.

MASTER_SELECT[1:0]	Opis
00	Onemogočen
01	Izbran IWB
10	Izbran DWB
11	N/A

Tabela 3. Pomen bitov registra MASTER_SELECT

Logika MASTER_SELECT vpisuje vrednosti v register MASTER_SELECT skladno s prioritetami, ki jih ima določene, in glede na stanje signala CYC, vmesnika DWB in vmesnika IWB. Ob predpostavki, da mikroprocesor bolj pogosto bere ukaze kot piše ali bere podatke v pomnilnik, ima vmesnik DWB višjo prioriteto kot vmesnik IWB. Če imata oba vmesnika spuščen signal CYC, potem je vodilo prosto in ga zasede tisti vmesnik, ki postavi signal CYC na visoko stanje. V primeru, da imata oba vmesnika signal CYC postavljena na visoko stanje, pravico do vodila dobi vmesnik DWB.

Logika MASTER_SELECT je implementirana na naslednji način:

```
////////////////////////
//master select logic //
////////////////////////
always @ (posedge clk_i) begin
     if(dwb_cyc_i) begin
          master_select <= 2'b10;
     end
     else if(iwb_cyc_i) begin
          master_slect <= 2'b01;
     end
     else begin
          master_select <= 2'b00;
     end
end
```

3.1.1.3. SLAVE_SELECT logika

Ker sistem vsebuje dva sužnja, je potrebno dekodiranje, kateremu sužnju je namenjena informacija na vodilu. To dekodiranje opravi logika SLAVE_SELECT, ki vsebuje 2-bitni register SLAVE_SELECT. Tabela 4 prikazuje pomen bitov v registru SLAVE_SELECT. Pisanje v ta register izvede logika SLAVE_SELECT, v odvisnosti od naslovljenega sužnja.

SLAVE_SELECT[1:0]	Opis
00	Onemogočen
01	Izbran pomnilniški krmilnik
10	Izbran celični avtomat
11	N/A

Tabela 4. Pomen bitov registra SLAVE_SELECT

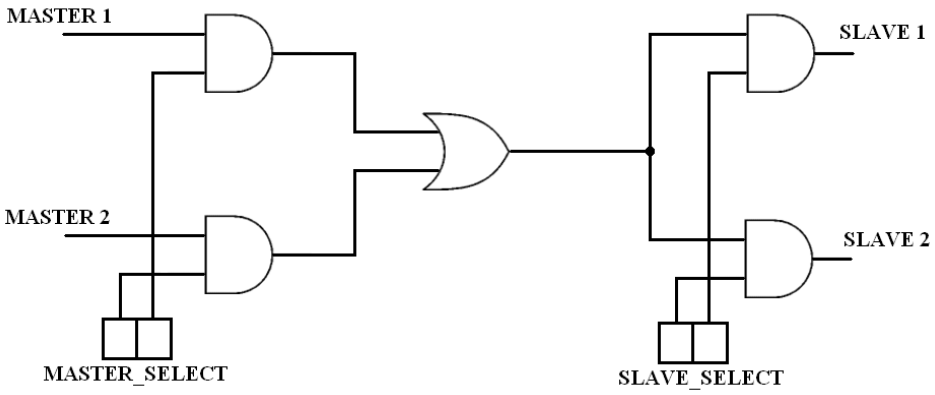
Logika SLAVE_SELECT uporablja enostavno dekodiranje naslovnega vodila, da ugotovi kateremu sužnju so namenjeni podatki. Če je bit MSB (ang. most significant bit – najpomembnejši bit) naslovnega vodila postavljen na "1", potem so podatki namenjeni celičnemu avtomatu, v nasprotnem primeru pa pomnilniškemu krmilniku.

Logika SLAVE_SELECT je implementirana na naslednji način:

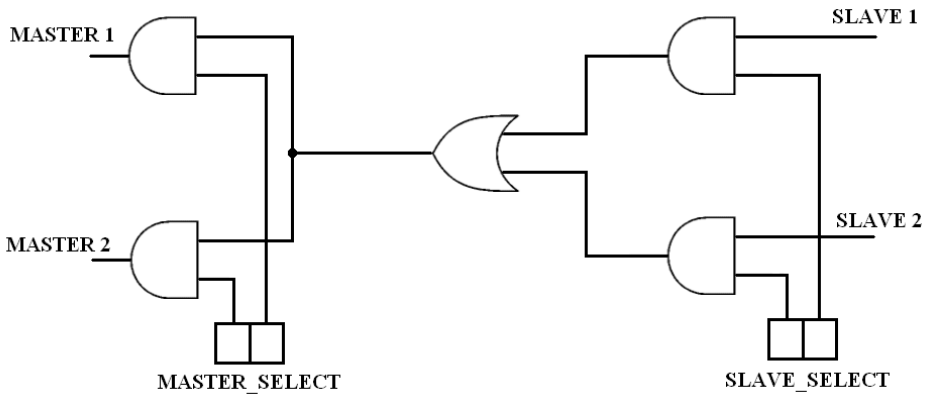
```
//////////////////////
//slave select logic//
//////////////////////
always @ (posedge clk_i) begin
     if(~addr[31])
          slave_select = 2'b01;
     else if (addr[31])
          slave_select = 2'b10;
     else
          slave_select = 1'b00;
end
```

3.1.1.4. Povezovanje vmesnikov v WISHBONE modulu

Povezovalna logika v modulu WISHBONE je realizirana s pomočjo logičnih vrat. Vsi signali uporabljajo isto logiko povezovanja. Povezovalna logika omogoča, da ima dostop do vodila v nekem trenutku en gospodar in en suženj. Slika 19 prikazuje povezovalno logiko signalov, ki potujejo od gospodarja proti sužnju, Slika 20 pa signale, ki potujejo od sužnja proti gospodarju.



Slika 19. Povezovalna logika signalov, ki gredo od gospodarja k sužnju



Slika 20. Povezovalna logika signalov, ki gredo od sužnja k gospodarju

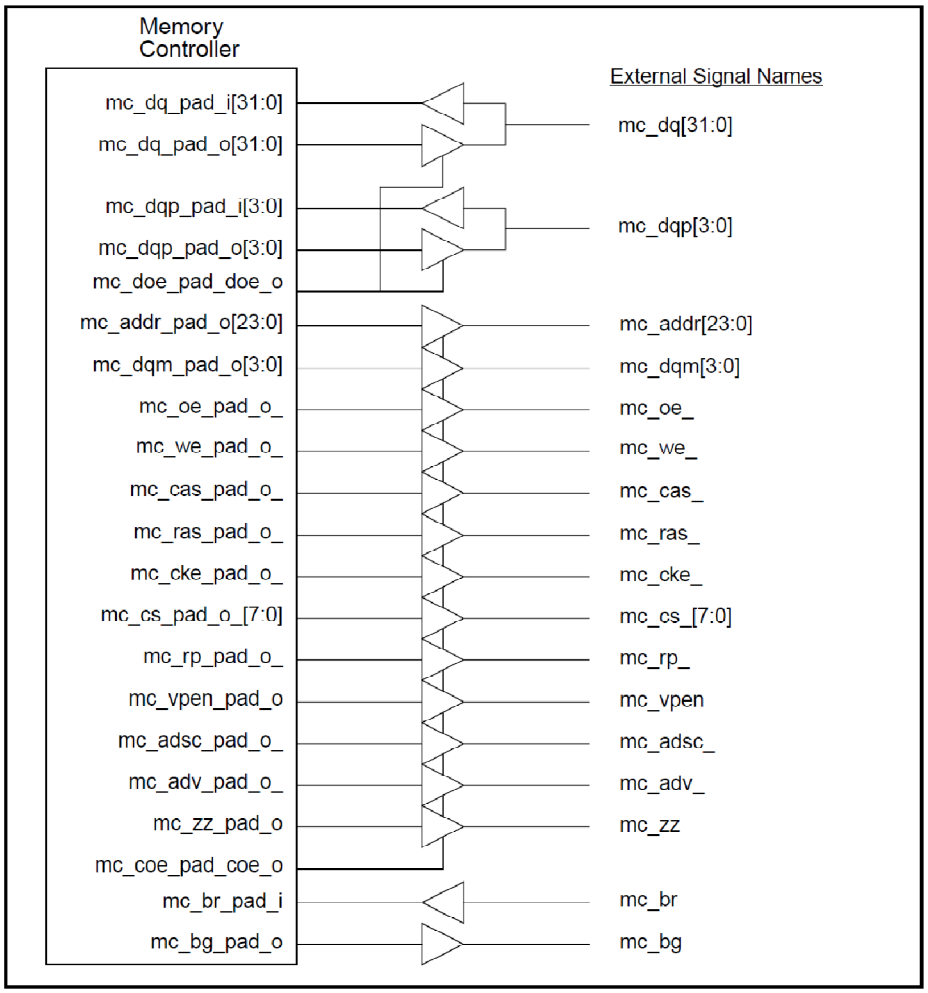
3.1.2. Povezava pomnilniškega krmilnika z glavnim pomnilnikom

Pomnilniški krmilnik (ang. memory controller) ima za komunikacijo z zunanjimi komponentami dva vmesnika. Vmesnik WISHBONE za komunikacijo z vodilom WISHBONE in pomnilniški vmesnik, za komunikacijo s pomnilnikom.

Vmesnik na WISHBONE pomnilniškega krmilnika je preprosto povezan na vodilo WISHBONE kot suženj, ki mu je dodeljen nek naslovni prostor. Ker v našem sistemu logika SLAVE_SELECT izbira aktivnega sužnja glede na MSB bit v naslovnem vodilu, je naslovni prostor pomnilniškega krmilnika od 0x0000_0000 do 0x7FFF_FFFF. Zaradi pravilnega pretvarjanja naslovov v pomnilniškem krmilniku, mora mikroprocesor generirati šestnajstiške naslove, ki so večkratniki števila 4.

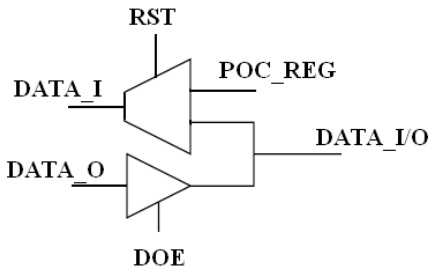
3.1.2.1. MC_BUF modul

Zaradi pravilnega delovanja je potrebno na vsako linijo v pomnilniškem vmesniku priključiti tristopenjski vmesnik (ang. tri-state buffer). Modul MC_BUF opravlja tri funkcije. Zagotavlja pravilno povezavo med pomnilniškim krmilnikom in pomnilnikom, zagotavlja ustrezno programiranje registra POC in ustrezno delitev urinega signala (SDRAM in pomnilniški vmesnik pomnilniškega krmilnika potrebujeta polovično frekvenco urinega signala ostalega sistema).



Slika 21. Vhodno izhodni tristopenjski vmesniki

Kot je prikazano na Sliki 21, je krmiljenje kontrolnih linij izvedeno s signalom COE – Control Output Enable, krmiljenje podatkovnih linij pa s signalom DOE – Data Output Enable. Pomnilniški vmesnik vsebuje eno vodilo za branje podatkov in eno vodilo za pisanje podatkov. Ker SDRAM vsebuje eno vodilo za branje in pisanje podatkov, je potrebna logika, ki povezuje podatkovna vodila pomnilniškega vmesnika in podatkovno vodilo glavnega pomnilnika (Slika 22). Ustrezno krmiljenje logike zagotavlja signal DOE.



Slika 22. Logika povezave podatkovne linije in programiranja registra POC

V modulu MC_BUF je implementirano tudi programiranje registra POC. Programiranje se izvede med reset ciklom z vnaprej nastavljeno vrednostjo. Pomen bitov 3-0 v registru POC kaže Tabela 1. Ostale vrednosti so visoko impedančno stanje Z.

Povezava podatkovnega vodila in programiranje registra POC, je implementirano na naslednji način:

```
//data logic
assign mc_data_pad_o = rst_i ? poc_reg : mc_dq_io;
assign mc_dq_io = mc_doe_pad_doe_i ? mc_data_pad_i : 32'hzzzz_zzzz;

.
.
.

initial begin
     clk = 1'b0;
     poc_reg = 32'hzzzz_zzzE;
end
```

3.1.2.2. Modul RAM_LOGIC

Pomnilniški vmesnik vsebuje veliko signalov, zaradi raznolikosti pomnilnikov, s katerimi lahko komunicira. Za komunikacijo pomnilniškega krmilnika in pomnilnika SDRAM so potrebni le naslednji signali:

- DQ – vhodno/izhodno podatkovno vodilo,
- ADDR – naslovno vodilo,
- BA – naslov banke v pomnilniku,
- CLK – urin signal za SDRAM,
- CKE – omogočanje urinega signala,
- CS_N –pove, kdaj je izbran pomnilnik SDRAM (cship select),
- RAS_N – dostop do vrstice v pomnilniku,
- CAS_N – dostop do stolpca v pomnilniku,
- WE_N –pove, kdaj gre za branje in kdaj za pisanje iz oz. v pomnilnik in
- DQM_N – maskiranje bajtov na podatkovni liniji.

Ker pomnilniški vmesnik ne vsebuje signala BA, ampak je prisoten v naslovnem vodilu, modul RAM_LOGIC poskrbi za ustrezno pretvorbo. Širina naslovnega vodila pri podatkovnem vmesniku je 24 bitov, pri pomnilniku SDRAM pa 11 bitov. Modul RAM_LOGIC poskrbi za pravilno preslikavo naslovnega vodila in pravilno vrednost signala BA.

V naslovno vodilo pomnilnika SDRAM se preslika spodnjih 11 bitov, BA signal pa dobi vrednosti naslovnih bitov 13 in 14.

3.1.3. Delovanje pomnilnika SDRAM

Za razumevanje komunikacije med pomnilniškim krmilnikom in pomnilnikom SDRAM, je najprej potrebno razumevanje delovanja pomnilnika SDRAM. Pomnilnik SDRAM ima svoje ukaze, ki jih interpretira in nato izvršuje. Pomnilnik ukaze dekodira glede na stanje signalov CS_N, RAS_N, CAS_N in WE_N. Tabela 5 prikazuje ukaze in pripadajočo kombinacijo signalov.

Ukaz/signal	CS_N	RAS_N	CAS_N	WE_N
Active enable	0	0	1	1
Auto refresh enable	0	0	0	1
Burst termination	0	1	1	0
Mode register enable	0	0	0	0
Precharge enable	0	0	1	0
Read enable	0	1	0	1
Write enable	0	1	0	0

Tabela 5. Dekodiranje ukazov s pomočjo signalov CS_N, RAS_N, CAS_N in WE_N

Ker je v sistemu uporabljen model realnega pomnilnika, je potrebno upoštevati tudi vse čase in zakasnitve, ki jih ima pomnilnik SDRAM. Za pravilno delovanje je potrebna konfiguracija pomnilnika.

Pomnilnika je razdeljena v 4 banke. Vsaka banka ima 2^{19} (524288) 32-bitnih lokacij za shranjevanje podatkov. Dostop do določene lokacije je implementiran tako, da enostavno sestavimo naslov vrstice in stolpca.

3.1.4. Implementacija enodimenzionalnega celičnega avtomata

Celični avtomat je implementiran v datoteki *ca_top.v*. Vsebuje vmesnik na WISHBONE za komunikacijo z ostalimi moduli. Začetno stanje dobi preko vhodnega podatkovnega vmesnika CA_DATA_I, nad katerim računa naslednja stanja s pomočjo podanega pravila. Končno stanje po *M* korakih (ciklih) pošlje preko izhodnega podatkovnega vmesnika CA_DATA_O. Število korakov (maksimalno 255) in 8-bitno pravilo dobi preko naslovnega vodila. Pomen bitov v naslovnem vodilu prikazuje Tabela 6.

Biti	Opis
31	Dostop do celičnega avtomata
30:28	Rezervirano
27:20	Število korakov
19:12	Rezervirano
11:4	Pravilo celičnega avtomata
3:0	Rezervirano

Tabela 6. Pomen bitov v naslovnem vodilu

Ko pride zahteva (signal CYC), se shrani vrednost na vhodnem podatkovnem vodilu (začetno stanje) v podatkovni register (PR). Število korakov in vektor za pravilo se določita glede na stanje ustreznih bitov na naslovnem vodilu. Ustrezni biti za prenos števila korakov in pravila celičnega avtomata so označeni v Tabeli 6.

Za izračun naslednjega stanja podatkovnega registra, porabi celični avtomat eno urino periodo. Za izračun *n* naslednjih stanj pa porabi *n* urinih period. V primeru, da zunanji modul zahteva rezultat prej kot je možno izračunati končno stanje, mora zunanji modul počakati, da celični avtomat izračuna končno stanje.

Za izračun naslednjega stanja so uporabljeni trije 32-bitni vektorji. Vektor *levo* predstavlja vse leve sosede, vektor *center* predstavlja centralne celice in vektor *desno*, desne sosede. Slika 23 prikazuje določevanje vrednost teh treh registrov.

Levo	PR[0]	PR[31]	…	PR[i-1]	PR[i]	PR[i+1]	…	PR[2]	PR[1]
Center	PR[31]	PR[30]	…	PR[i-1]	PR[i]	PR[i+1]	…	PR[1]	PR[0]
Desno	PR[30]	PR[29]	…	PR[i-1]	PR[i]	PR[i+1]	…	PR[0]	PR[31]

Slika 23. Vrednost registrov *Levo*, *Center* in *Desno*

Definicija vektorjev *levo*, *center* in *desno* ter izračun naslednjega stanje podatkovnega registra, je s pomočjo vektorja, ki vsebuje pravilo implementirano na naslednji način:

```
l = {din[0], din[31:1]};
c = din;
r = {din[30:0], din[31]};

din  = (rule[0] & ~l & ~c & ~r)
     | (rule[1] & ~l & ~c &  r)
     | (rule[2] & ~l &  c & ~r)
     | (rule[3] & ~l &  c &  r)
     | (rule[4] &  l & ~c & ~r)
     | (rule[5] &  l & ~c &  r)
     | (rule[6] &  l &  c & ~r)
     | (rule[7] &  l &  c &  r);
```

3.2. Konfiguracija modulov

3.2.1. Mikroprocesor OpenRISC 1200

Pred uporabo ali simulacijo mikroprocesorja je potrebna konfiguracija. Implementacija mikroprocesorja že vsebuje privzeto konfiguracijo. Uporabnik lahko spremeni določene prednastavljene vrednosti glede na potrebe sistema, vendar morajo biti novonastavljene vrednosti v okviru predvidenih vrednosti. Tabla 7 prikazuje privzete vrednosti parametrov in rang vrednosti, ki jih določen pramater lahko zavzame. Privzeto konfiguracijo mikroprocesorja OpenRISC 1200 kaže Tabela 7.

Variable name	Range	Default	Description	Opis
EADDR_WIDTH	32	32	Effective address width	Efektivna naslovna širina
VADDR_WIDTH	32	32	Virtual address width	Virtualna naslovna širina
PADDR_WIDTH	24-36	32	Physical address width	Fizična naslovna širina
DATA_WIDTH	32	32	Data width / operation width	Širina podatkovnega vodila
DC_IMP	0-1	1	Data cache implementation	Implementacija podatkovnega predpomnilnika
DC_SETS	512	512	Data cache number of sets	Število setov podatkovnega predpomnilnika
DC_WAYS	1	1	Data cache number of ways	Število načinov implementacije podatkovnega predpomnilnika
DC_LINE	16	16	Data cache line size	Dolžina vrstice v podatkovnem predpomnilniku
IC_IMPL	0-1	1	Instruction cache implementation	Implementacija ukaznega predpomnilnika
IC_SETS	512	512	Instruction cache number of sets	Število setov ukaznega predpomnilnika
IC_WAYS	1	1	Instruction cache number of ways	Število načinov implementacije ukaznega predpomnilnika
IC_LINE	16	16	Instruction cache line size	Dolžina vrstice v ukaznem predpomnilniku
DMMU_IMPL	0-1	1	Data MMU implementation	Implementacija podatkovne MMU
DTLB_SETS	64	64	Data TLB number of sets	Število setov podatkovnega TLB
DTLB_WAYS	1	1	Data TLB number of ways	Število načinov implementacije podatkovnega TLB
IMMU_IMPL	0-1	1	Instruction MMU implementation	Implementacija ukazne MMU
ITLB_SETS	64	64	Instruction TLB number of sets	Število setov ukaznega TLB
ITLB_WAYS	1	1	Instruction TLB number of ways	Število načinov implementacije ukaznega TLB
PIC_INTS	2-32	30	Number of interrupt inputs	Število prekinitvenih vhodov

Tabela 7. Konfiguracija mikroprocesorja

Potrebno je opraviti tudi konfiguracijo delilnika urinega signala. Ker mikroprocesor uporablja ločena urina signala za jedro mikroprocesorja in vmesnika na WISHBONE, je potrebno nastaviti pravilno delilno razmerje med njima. V našem primeru sta oba signala enaka, zato je delilno razmerje 1. Možne so tudi druga delilna razmerja, ki jih prikazuje Tabela 8.

Vhod CLMODE_I	Opis
00	WB_CLK = RISC_CLK
01	WB_CLK = RISC_CLK/2
10	N/A
11	WB_CLK = RISC_CLK/4

Tabela 8. Možne vhodne kombinacije vhoda CLMODE_I

3.2.2. Pomnilniški krmilnik

Pomnilniški vmesnik vsebuje več nastavitvenih registrov. Pomnilniški krmilnik podpira priklop do 8 pomnilnikov, različnih tipov. Pomnilnik izbira naslovljene pomnilnike s pomočjo signala CS_N. Ker so pomnilniki lahko različnih tipov, imajo tudi različne lastnosti, zato vsebuje pomnilniški krmilnik 8 registrov TMS (ang. timing select register – register za definiranje časov pomnilnika) in 8 registrov CSC (ang. chip select configuration register – register za konfiguracijo posameznega pomnilnika).

Na začetku je potrebno določiti tip privzetega pomnilnika in njegove nastavitve vpisati v registra TMS in CSC. V Tabeli 9 je prikazan pomen bitov v registru TMS. Ker je v sistemu uporabljen pomnilnik SDRAM, je vrednost registra TMS 0xFFFF_8220, ki se jo nastavi v datoteki *mc_defines.v*.

Bit	Opis
31:28	Rezervirano
27:24	Trfc – čas za osveževanje [periode]
23:20	Trp – čas za prednabijanje [periode]
19:17	Trcd – čas od ukaza ACTIVE do ukaza READ/WRITE [periode]
16:15	Twr – čas za končanje pisanja [periode]
14:10	Rezervirano
9	Število WRITE dostopov v enem ciklu 0 – programirljivo število dostopov 1 – en dostop v enem ciklu
8:7	Način delovanja 0 – normalno delovanje 1-3 – rezervirano
6:4	Število urinih period za dostop do stolpca 0-1 – rezervirano 2 – dva cikla 3 – tri cikle 4-7 – rezervirano
3	Burst type 0 – sekvenčno 1 – način *interleaved*
2:0	Število dostopov v enem ciklu 0 – 1 1 – 2 3 – 4 4 – 8 4-6 – rezervirano 7 – cela stran

Tabela 9. Pomen bitov v registru TMS za pomnilnik SDRAM

Za konfiguracijo pomnilnika SDRAM je potrebno določiti vrednosti registru CSC. S pisanjem v register CSC se omogoči priklop pomnilnika na pomnilniški krmilnik (register CSC0 je vedno omogočen). S pisanjem se določi tip pomnilnika, širino podatkovnega vodila, kapaciteto pomnilnika itd. Register CSC0 dobi vrednosti iz nastavitvenega registra POC.

Tabela 10 prikazuje pomen bitov v registru CSC. Register CSC0 v našem primeru dobi vrednost 0x0000_0021.

Bit	Opis
31:24	Rezervirano
23:16	Naslov posameznega pomnilnika S tem poljem in naslovno masko se naredi operacija AND in se primerja z naslovnim vodilom (bit 28-21).
15:12	Rezervirano
11	Omogočanje paritete 0 – generator paritete in preverjanje paritete onemogočeno 1 – generator paritete in preverjanje paritete omogočeno
10	0 – zapiranje vrstice po *branju* ali *pisanju* 1 – vrstica vedno odprta
9	Izbiranje naslova banke 0 – naslov banke iz naslova stolpca 1 – naslov banke iz naslova vrstice
8	Zaščita pisanja (za zaščito določenega območja) 0 – zaščita pisanja omogočena 1 – zaščita pisanja onemogočena
7:6	Velikost pomnilnika 0 – 64Mb 1 – 128Mb 2 – 256Mb 3 – rezervirano
5:4	Širina podatkovnega vodila 0 – 8-bit 1 – 16-bit 2 – 32-bit 3 – rezervirano
3:1	Tip pomnilnika 0 – SDRAM 1 – SSRAM 2 – asinhrona naprava 3 – sinhrona naprava 4-7 – rezervirano
0	Omogočanje CS (ang. chip select) 0 – onemogočen 1 – omogočen

Tabela 10. Pomen bitov v registru CSC

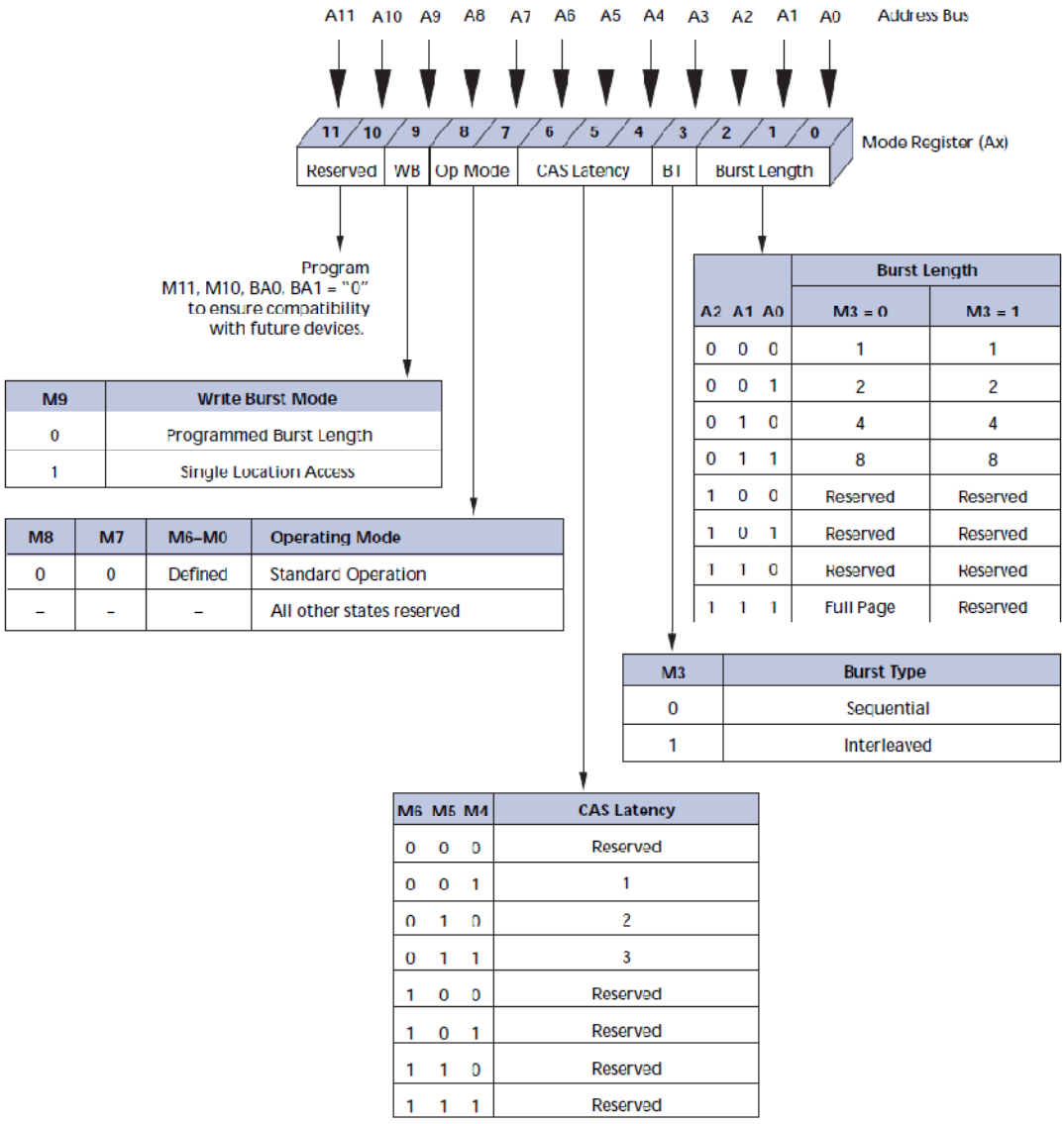
3.2.3. Pomnilnik SDRAM

Za pravilno delovanje pomnilnika je na začetku potrebna inicializacija. Inicializacija je sestavljena iz določenih zaporednih ukazov. Med ukazi so potrebne določene časovne konstante (število urinih period).

Postopek inicializacije:

1. Dovod stabilnega urinega signala.
2. Ko je urin signal stabilen, postavimo signal CKE v visoko stanje.
3. Za spremembo signala CKE, izvedemo vsaj en ukaz NOP (ang. no operation – brez operacije).
4. Naslednji korak izvedemo z ukazom PRECHARGE ALL.
5. Počakamo 20 urinih period. Med tem časom je dovoljen ukaz NOP.
6. Ukaz AUTO REFRESH (avtomatska osvežitev).
7. Počakamo 70 urinih period, med katerimi je dovoljen le ukaz NOP.
8. Ukaz AUTO REFRESH.
9. Čakanje 70 urinih period, med katerimi je dovoljen le ukaz NOP.
10. Pomnilnik je pripravljen za programiranje nastavitvenega registra. Ker ima nastavitveni register na začetku nedefinirano stanje, je potrebno nastaviti ustrezne vrednosti, da dosežemo želeno delovanje pomnilnika. Z ukazom LMR (ang. load mode register – naloži nastavitveni register), ki ga dobimo z stanjem signalov CS_N = 0, RAS_N = 0, CAS_N = 0 in WE_N = 0, naložimo želeno vrednost, ki jo definiramo v naslovnem vodilu.
11. Čakanje 5 urinih period, med katerimi je dovoljen le ukaz NOP.

Logika za programiranje nastavitvenega registra v času trajanja ukaza LMR, nastavi vrednosti nastavitvenega registra glede na stanje naslovnih linij. Slika 24 prikazuje pomen bitov v nastavitvenem registru pomnilnika SDRAM.



Slika 24. Pomen bitov v nastavitvenem registru

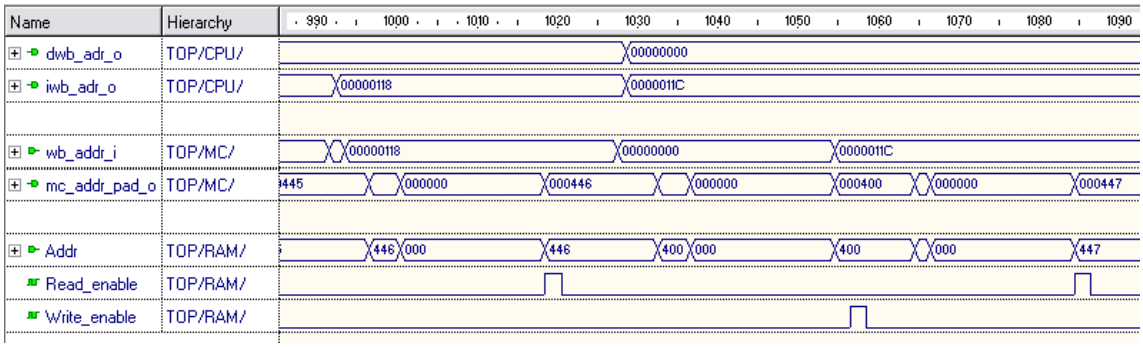
4. Delovanje sistema

4.1. Branje in pisanje podatkov v in iz pomnilnika

4.1.1. Pretvarjanje naslovov

Mikroprocesor OpenRISC 1200 začne z branjem podatkov iz pomnilnika na lokaciji 0x100. Programski števec povečuje naslove po 0x4, zato je naslednje stanje na naslovnem vodilu 0x104. Pomnilniški krmilnik nato naslov, ki ga dobi na vmesniku WISHBONE deli z 0x4. Tako se naslov 0x100 pretvori v naslov 0x40, na katerem je shranjen prvi ukaz, ki ga prebere procesor. Če ukaz na lokaciji 0x40 ni skok, potem se programski števec mikroprocesorja poveča za 0x4 in tako naslednji naslov kaže na lokacijo 0x41 v pomnilniku. Slika 25 prikazuje pretvarjanje naslovov med mikroprocesorjem in pomnilnikom. Naslovi na pomnilniku so veljavni le ob signalu READ ENABLE in signalu WRITE ENABLE. Za vse naslove, ki jih generira programski števec, je branje izvedeno na vmesniku IWB. Za naslove, ki jih generira ukaz *preberi,* se branje izvede na vmesniku DWB.

V primeru, da mikroprocesor prebere iz pomnilnika ukaz *shrani,* se za shranjevanje uporabi vmesnik DWB. Za pomnilniški krmilnik ni pomembno, ali ga naslavlja mikroprocesor z vmesnika IWB ali vmesnika DWB. Pomnilniški krmilnik obravnava vse naslove enako.



Slika 25. Prikaz pretvarjanja naslovov v sistemu

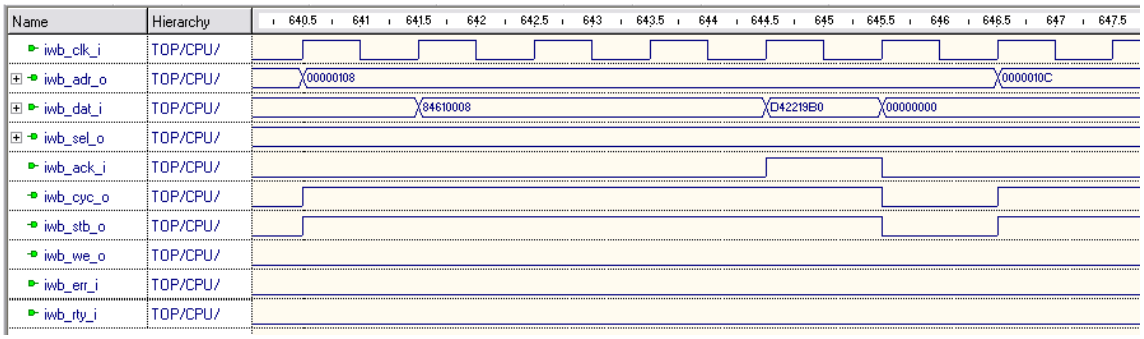
4.1.2. Dostop do pomnilnika

Mikroprocesor začne dostop do pomnilnika z signalom CYC. Če gre za cikel branja ukazov, potem postavi signal IWB_CYC na visoko stanje, v primeru branja ali pisanja podatkov iz ali v pomnilnik postavi signal DWB_CYC na visoko stanje.

S signalom WE OpenRISC 1200 sporoči, ali gre za branje ali pisanje v pomnilnik. Če je WE v nizkem stanju, gre za ukaze tipa LOAD, oziroma, če je v visokem stanju, za ukaze tipa STORE. S signalom SEL mikroprocesor sporoči, kateri bajti na podatkovnem vodilu so veljavni (če zavzame signal SEL vrednost 0xF, potem so na podatkovnem vodilu veljavni vsi biti).

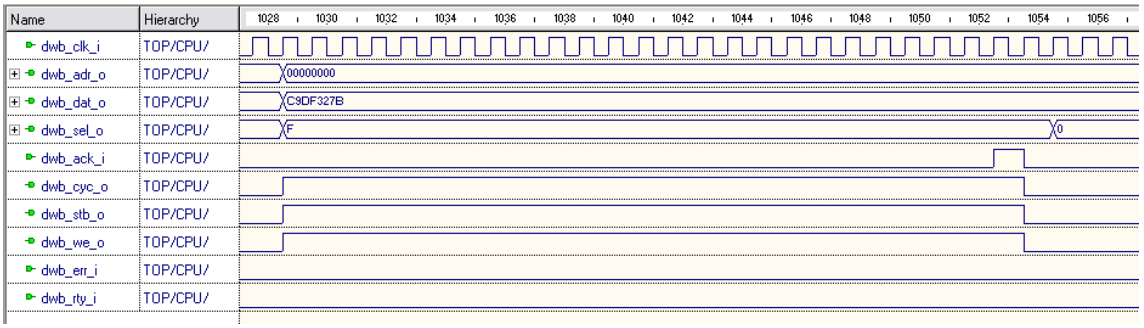
Ko so na mikroprocesorju veljavni signali ADR, WE, STB, CYC in SEL, mikroprocesor čaka na podatke na vhodnem podatkovnem vodilu DATA_I. Ko mikroprocesor dobi signal ACK v visokem stanju, ve, da so podatki na podatkovnem vodilu veljavni in da jih lahko uporabi.

Mikroprocesor lahko namesto signala ACK dobi signala ERR ali RTY, ki kažeta, da s prenosom ni bilo vse v redu in je potrebno cikel ponoviti. Na Sliki 26 je prikazan veljaven prenos podatkov na vmesniku IWB. Mikroprocesor je podal zahtevo za podatke iz lokacije 0x108, v resnici pa so bili zapisani na lokaciji 0x42.



Slika 26. Uspešen cikel branja ukaza

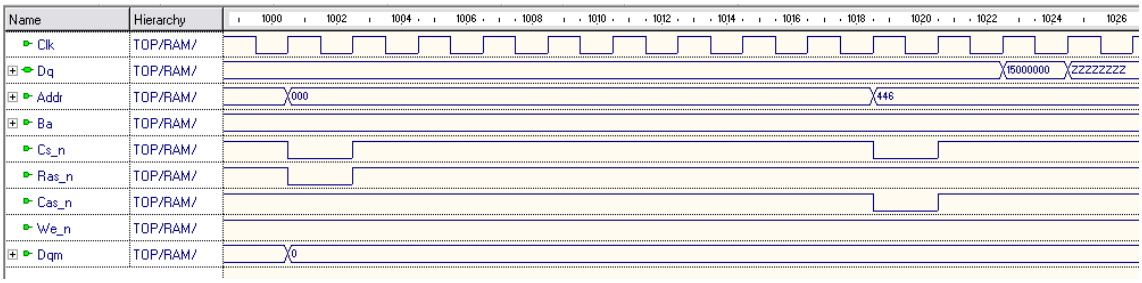
Za branje podatkov iz pomnilnika so potrebni isti signali, le da se uporabi vmesnik na DWB. Od branja ukazov ali podatkov iz pomnilnika se nekoliko razlikuje pisanje podatkov v pomnilnik (ukazi tipa STORE). Glavna razlika je v signalu WE, ki mora biti postavljen visoko, saj signal WE določa, ali gre za branje ali pisanje. Za pisanje podatkov mikroprocesor uporablja DATA_O (izhodno podatkovno vodilo). Pri branju podatkov so podatki na podatkovnem vodilu veljavni samo ob dvignjenem signalu ACK, pri pisanju pa so podatki na izhodnem podatkovnem vodilu veljavni cel cikel. Slika 27 prikazuje uspešen zapis podatkov na navidezno lokacijo 0x0, ob pretvorbi na fizični naslov pa se podatki zapišejo na lokacijo 0x0.



Slika 27. Uspešen zapis podatkov

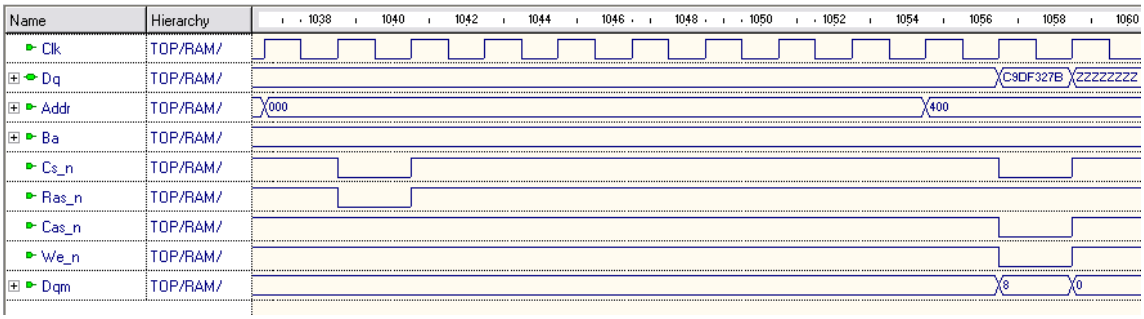
Pomnilniški vmesnik signale iz vodila WISHBONE pretvori v signale, ki jih potrebuje pomnilnik SDRAM za delovanje. Pred vsakim dostopom do pomnilnika se izvede ukaz ACTIVE ENABLE, ki povzroči, da se aktivira določena banka in odpre določena vrstica. S signali CS_N, RAS_N, CAS_N in WE_N, se dekodirajo operacije, signal DQM_N definira maskiranje bajte na podatkovnem vodilu, DQ pa služi kot vodilo za prenos podatkov v obe smeri.

Na Sliki 28 je najprej prikazan ukaz ACTIVE ENABLE, ki mu sledi ukaz *branje* podatkov iz pomnilnika. Podatki se na podatkovnem vodilu DQ ne pojavijo v istem trenutku kot ukaz branja, ampak z zakasnitvijo dveh urinih period. Branje podatkov se izvede na naslovu 0x46.



Slika 28. Branje podatkov iz pomnilnika

Pri pisanju v pomnilnik (Slika 29) so podatki veljavni v istem trenutku, ko nastopi ukaz, zato se ukaz za pisanje zaključi v eni urini periodi. Podatki se v tem ciklu zapišejo na naslov 0x0.

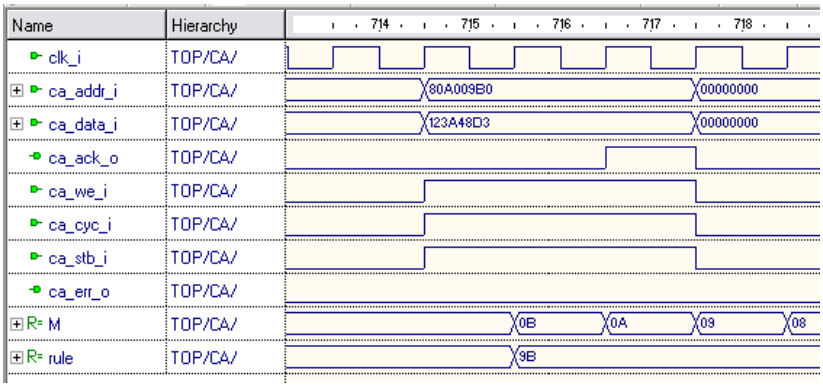


Slika 29. Pisanje podatkov v pomnilnik

4.1.3. Dostop do celičnega avtomata

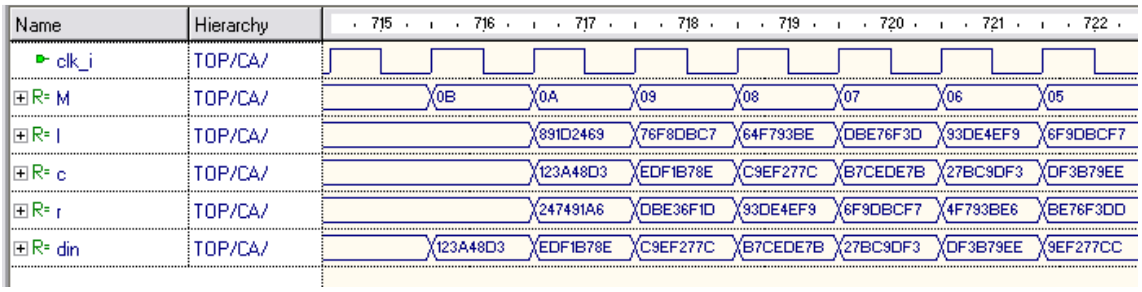
Mikroprocesor dostopa do celičnega avtomata z ukazoma tipa LOAD in STORE preko vmesnika DWB. Vmesnik DWB je primeren, ker lahko na enostaven način pridemo do poljubnega naslova. Ker je izbira sužnja implementirana glede na vrednost naslovnega vodila, je zelo pomembna prej omenjena lastnost. Mikroprocesor OpenRISC 1200 ima dostop do celičnega avtomata v primeru, da je MSB na naslovnem vodilu postavljen na visoko stanje. V tem primeru logika SLAVE_SELECT v implementaciji vodila WISHBONE kot aktivnega sužnja aktivira celični avtomat.

Ko celični avtomat dobi signal CYC, aktivira logiko za prevzem začetnega stanja iz podatkovnega vmesnika v podatkovni register in glede na stanje naslovne linije ustrezno nastavi število korakov *M* in vektor, v katerem je zapisano pravilo. Na Sliki 30 je prikazano branje podatkov iz podatkovnega vodila ter nastavitev števila korakov *M* in vektorja, ki vsebuje pravilo celičnega avtomata.



Slika 30. Branje podatkov iz vmesnika na WISHBONE

Stanje podatkovnega registra in vektorjev *levo*, *center* in *desno* po *M* korakih prikazuje Slika 31.



Slika 31. Stanja podatkovnega registra in vektorjev *levo*, *center* in *desno*

Na Sliki 31 je prikazan podatkovni register celičnega avtomata, ki za posodabljanje stanja uporablja pravilo 155. Pravilo 155 prikazuje Tabela 11.

Celice			
i-1	i	i+1	i(t+1)
0	0	0	1 (LSB vektorja *rule*)
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1 (MSB vektorja *rule*)

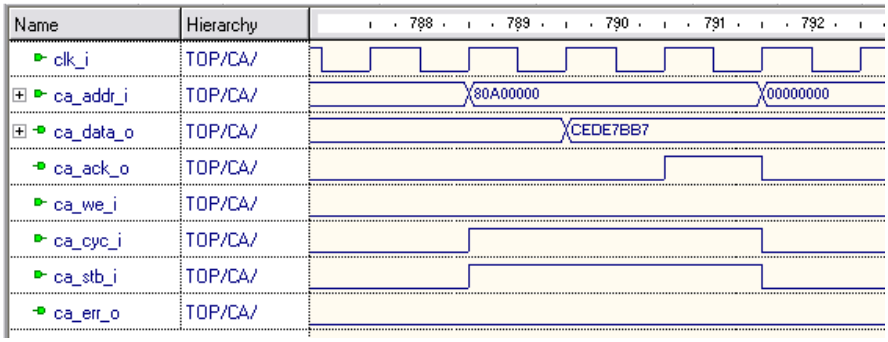
Tabela 11. Pravilnostna tabela pravila 155

Vrednosti podatkovnega registra din po nekaj začetnih korakih prikazuje Tabela 12

Korak	Binarna vrednost registra *din*	Šestnajstiška vrednost registra *din*
0	0001_0010_0011_1010_0100_1000_1101_0011	123A_48D3
1	1110_1101_1111_0001_1011_0111_1000_1110	EDF1_B78E
2	1100_1001_1110_1111_0010_0111_0111_1100	C9EF_277C
3	1011_0111_1100_1110_1101_1110_0111_1011	B7CE_DE7B
4	0010_0111_1011_1100_1001_1101_1111_0011	27BC_9DF3
5	1101_1111_0011_1011_0111_1001_1110_1110	DF3B_79EE
6	1001_1110_1111_0010_0111_0111_1100_1100	9EF2_77CC

Tabela 12. Vrednost registra *din* po nekaj korakih

Celični avtomat končno stanje doseže po *M* korakih. Ko se to zgodi, je v podatkovnem registru končno stanje, ki je pripravljeno in čaka na ukaz branja podatkov iz celičnega avtomata. Glede na stanje signala CA_WE, celični avtomat prepozna, ali gre za ukaz nalaganja podatkov ali gre za ukaz branja končnega stanja podatkov. Če celični avtomat prepozna cikel kot branje podatkov, prikazano na Sliki 32, potem postavi na izhodno podatkovno vodilo končno stanje in signal CA_ACK postavi na visoko stanje. Pri branju podatkov iz celičnega avtomata je pomemben samo bit MSB v naslovnem vodilu.



Slika 32. Branje končnega stanja iz celičnega avtomata

5. Test sistema in rezultati delovanja

5.1. Program in podatki v pomnilniku

Po koncu reset cikla mikroprocesor OpenRISC 1200 zahteva podatke na lokaciji 0x100, kar ustreza lokaciji 0x40 v pomnilniku. Na tej lokaciji se začne program, ki je zapisan v šestnajstiški obliki. Mikroprocesor po vrstnem redu bere ukaze iz pomnilnika, jih interpretira ter izvaja. Podatki so shranjeni od naslova 0x0 naprej.

Večina ukazov za mikroprocesor OpenRISC 1200 je sestavljena iz 6-bitne operacijske kode, dveh 5-bitnih naslovov splošno-namenskih registrov in 16-bitnega odmika. V dodatku A so prikazani vsi ukazi, ki so implementirani v mikroprocesorju OpenRISC 1200. Odmik se običajno prišteje vsebini enega izmed dveh registrov, katerih naslovi so navedeni v operandih ukaza.

V modelu SDRAM-a so pomnilniške lokacije predstavljene z množico 32-bitnih vektorjev. Pomnilniški prostor je razdeljen na štiri banke in vsaka banka vsebuje 2^{19} (524288) pomnilniških lokacij. Ob času 0 model SDRAM-a napolni vse štiri banke 32-bitnih vektorjev. Vrednosti vektorjev prebere iz štirih tekstovnih datotek, v katerih je zapisano začetno stanje vektorjev (če so datoteke prazne, vektorji zavzamejo nedefinirano stanje).

Za potrebe simulacije je v *banki0* zapisan program in podatki, ki jih potrebuje program za pravilno delovanje.

5.1.1. Prikaz pomnilniške *banke0* pred in po zagonu programa

Začetno stanje *banke0,* (z "`//`" je označen komentar in se ob nalaganju v vektorje ne upošteva, z "`@40`" je označen skok na pomnilniško lokacijo 0x40):

```
00000000 //koncno stanje CA
80A00000 //odmik + stevilo korakov
123a48d3 //zacetno stanje CA

@40
84410004 //opcode 0x21;    EA <- I+r1[31:0]; r2[31:0] <- (EA)[31:0]
84610008 //opcode 0x21;    EA <- I+r1[31:0]; r3[31:0] <- (EA)[31:0]
//pravilo za CA 0x9B
D42219B0 //opcode 0x35;    EA <- I+r2[31:0]; (EA)[31:0] <- r3[31:0]
//pravilo za CA 0xAA
//D4221AA0 //opcode 0x35;  EA <- I+r2[31:0]; (EA)[31:0] <- r3[31:0]
84620000 //opcode 0x21;    EA <- I+r2[31:0]; r3[31:0] <- (EA)[31:0]
D4011800 //opcode 0x35;    EA <-I +r1[31:0]; (EA)[31:0] <- r3[31:0]
15000000 //NOP
```

Končno stanje v *banki0* se razlikuje od začetnega samo v prvi vrstici, kjer je namesto `00000000` zapisano končno stanje celičnega avtomata, ki je `CEDE7BB7`.

Sistem omogoča tudi opisno spremljanje dogajanja v sistemu. Če v datoteki celičnega avtomata in pomnilnika SDRAM definiramo `debug = 1'b1`, potem lahko opisno spremljamo dogajanje.

Opisno spremljanje dogajanja branja in pisanja v pomnilnik, pošiljanje in sprejemanje podatkov v in iz celičnega avtomata ter spremljanje računanja stanj v celičnem avtomatu, je prikazano v spodaj. Prikazan je tudi čas, v katerem se zgodi določena aktivnost v sistemu.

Izpis iz vmesnika:

- konfiguracija pomnilnika,

```
# KERNEL: test_bench.TOP.RAM at time  13 ns PRE   : Bank = ALL
# KERNEL: test_bench.TOP.RAM at time  55 ns AREF  : Auto Refresh
# KERNEL: test_bench.TOP.RAM at time  197 ns AREF : Auto Refresh
# KERNEL: test_bench.TOP.RAM at time  339 ns LMR  : Load Mode
Register
# KERNEL:              CAS Latency      = 2
# KERNEL:              Burst Length     = 1
# KERNEL:              Burst Type       = Sequential
# KERNEL:              Write Burst Mode = Single Location Access
```

- branje ukazov iz pomnilnika,

```
# KERNEL: test_bench.TOP.RAM at time      511 ns ACT  : Bank = 0
Row =    0
# KERNEL: test_bench.TOP.RAM at time      531 ns NOTE : Start
Internal Auto Precharge for Bank 0
# KERNEL: test_bench.TOP.RAM at time      531 ns READ : Bank = 0
Row =    0, Col =  64, Data = 84410004, Dqm = 0000
```

- branje podatkov iz pomnilnika,

```
# KERNEL: test_bench.TOP.RAM at time      547 ns ACT  : Bank = 0
Row =    0
# KERNEL: test_bench.TOP.RAM at time      567 ns NOTE : Start
Internal Auto Precharge for Bank 0
# KERNEL: test_bench.TOP.RAM at time      567 ns READ : Bank = 0
Row =    0, Col =   1, Data = 80a00000, Dqm = 0000
```

- pošiljanje podatkov v celični avtomat,

```
# KERNEL: test_bench.TOP.CA at time       716 ns WRITE : M =  10
rule = 155
```

- računanje stanj celičnega avtomata,

```
# KERNEL: test_bench.TOP.CA at time       717 ns NOTE : M =   9
l = 0x891d2469 c = 0x123a48d3 r = 0x247491a6 data = 0xedf1b78e
# KERNEL: test_bench.TOP.CA at time       718 ns NOTE : M =   8
l = 0x76f8dbc7 c = 0xedf1b78e r = 0xdbe36f1d data = 0xc9ef277c
# KERNEL: test_bench.TOP.CA at time       719 ns NOTE : M =   7
l = 0x64f793be c = 0xc9ef277c r = 0x93de4ef9 data = 0xb7cede7b
# KERNEL: test_bench.TOP.CA at time       720 ns NOTE : M =   6
l = 0xdbe76f3d c = 0xb7cede7b r = 0x6f9dbcf7 data = 0x27bc9df3
# KERNEL: test_bench.TOP.CA at time       721 ns NOTE : M =   5
l = 0x93de4ef9 c = 0x27bc9df3 r = 0x4f793be6 data = 0xdf3b79ee
```

```
# KERNEL: test_bench.TOP.CA at time       722 ns NOTE : M =   4
l = 0x6f9dbcf7 c = 0xdf3b79ee r = 0xbe76f3dd data = 0x9ef277cc
# KERNEL: test_bench.TOP.CA at time       723 ns NOTE : M =   3
l = 0x4f793be6 c = 0x9ef277cc r = 0x3de4ef99 data = 0x7cede7bb
# KERNEL: test_bench.TOP.CA at time       724 ns NOTE : M =   2
l = 0xbe76f3dd c = 0x7cede7bb r = 0xf9dbcf76 data = 0x7bc9df32
# KERNEL: test_bench.TOP.CA at time       725 ns NOTE : M =   1
l = 0x3de4ef99 c = 0x7bc9df32 r = 0xf793be64 data = 0xf3b79eed
# KERNEL: test_bench.TOP.CA at time       726 ns NOTE : M =   0
l = 0xf9dbcf76 c = 0xf3b79eed r = 0xe76f3ddb data = 0xef277cc9
```

- sprejemanje končnega stanja iz celičnega avtomata in

```
# KERNEL: test_bench.TOP.CA at time       790 ns READ : data =
0xef277cc9
```

- pisanje končnega stanja v pomnilnik.

```
# KERNEL: test_bench.TOP.RAM at time      875 ns ACT  : Bank = 0
Row =    0
# KERNEL: test_bench.TOP.RAM at time      893 ns WRITE: Bank = 0
Row =    0, Col =   0, Data = ef277cc9, Dqm = 1000
```

6. Sklepne ugotovitve

Cilj diplomske naloge je implementacija paralelnega izvajanja celičnega avtomata na mikroprocesorju OpenRISC 1200. Celični avtomat je implementiran na strojnem nivoju, saj stanja celic računa digitalna logika. Test sistema je opravljen z nekaj ukazi mikroprocesorja OpenRISC 1200. Ker celični avtomat uporabi za računanje naslednjega stanja le eno urino periodo, je izvajanje avtomata zelo hitro. Za komunikacijo mikroprocesorja in celičnega avtomata je uporabljen najlažji način, in sicer z ukazi, implementiranimi na mikroprocesorju, tipa LOAD in STORE.

Celoten sistem je napisan v jeziku Verilog. Odločitev o izbiri je bila preprosta, saj so vse komponente napisane v omenjenem jeziku. Vsi moduli so napisani na RTL nivoju v jeziku Verilog. Pri izdelavi sem imel kar nekaj težav, ki sem jih s pomočjo mentorja uspešno rešil. Mikroprocesor OpenRISC 1200 je namreč zelo kompleksno digitalno vezje. Najtežji del je bil vzpostavljanje komunikacije z mikroprocesorjem in pomnilnikom. Ko je bil ta problem rešen, nadaljevanje ni bilo več tako težko.

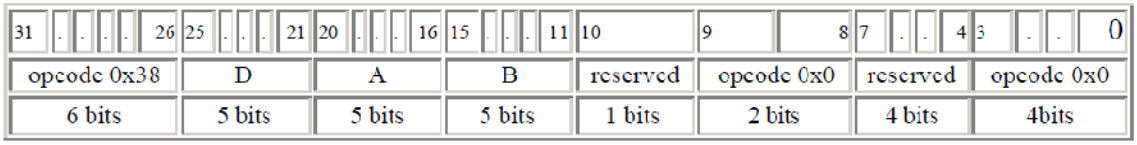
Sistem je delujoč do te mere, da mikroprocesor bere ukaze iz pomnilnika, jih interpretira in izvaja. Za dostop do celičnega avtomata je uporabljeno dekodiranje bita MSB v naslovnem vodilu. Sistem je možno tudi nadgraditi. Naslednja stopnja nadgraditve sistema bi bila implementacija pomožnega pomnilnika (FLASH) in vhodno-izhodne enote. V tem primeru bi bil sistem veliko bolj fleksibilen.

Med izdelavo diplomske naloge sem se naučil veliko novega. Najbolj mi bo koristilo znanje HDL jezika Verilog. Zelo me zanima delovanje mikroprocesorjev in njihova implementacija, zato sem za diplomsko nalogo izbral temo s tega področja. Načrtovanje tovrstnih sistemov me je navdušilo v tej meri, da bom zagotovo poskusil sistem nagraditi.

Dodatek A

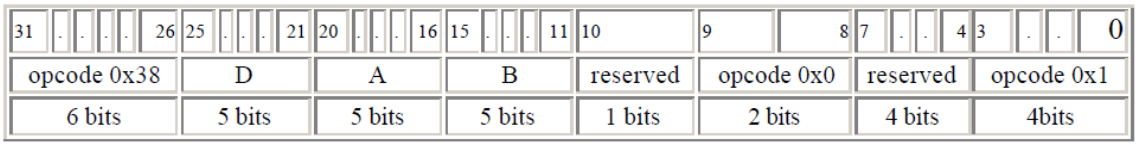
Ukazi, ki so implementirani na mikroprocesorju OpenRISC 1200.

Add signed – l.add



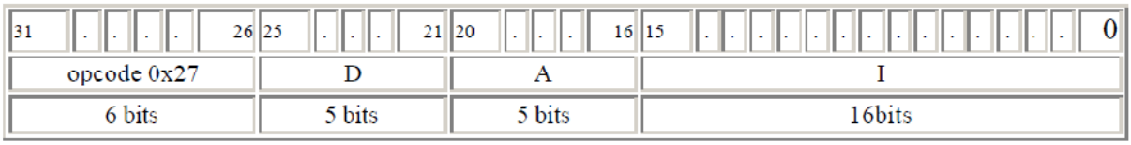
31 26	25 21	20 16	15 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x0
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

Add signed and carry – l.addc



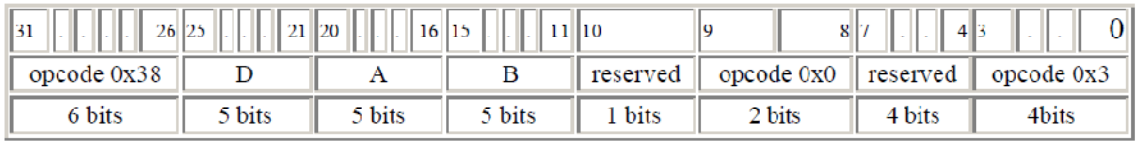
31 26	25 21	20 16	15 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x1
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

Add immediate signed – l.addi



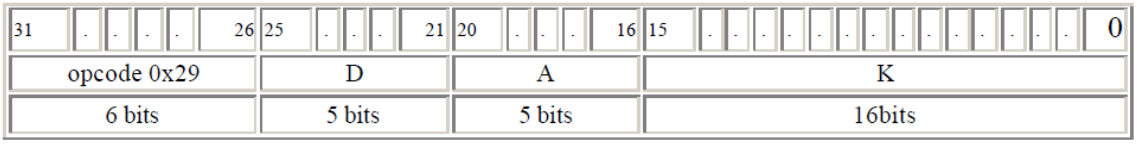
31 26	25 21	20 16	15 0
opcode 0x27	D	A	I
6 bits	5 bits	5 bits	16bits

And – l.and



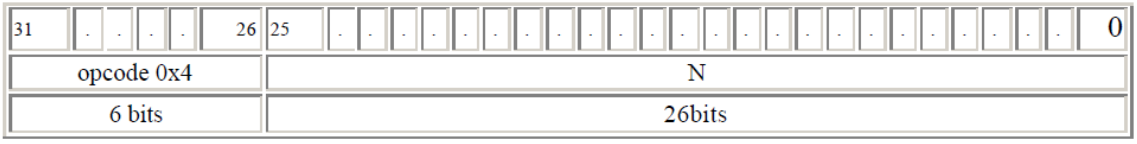
31 26	25 21	20 16	15 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x3
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

And with immediate half word – l.andi



31 26	25 21	20 16	15 0
opcode 0x29	D	A	K
6 bits	5 bits	5 bits	16bits

Branch if flag – l.bf



31 26	25 0
opcode 0x4	N
6 bits	26bits

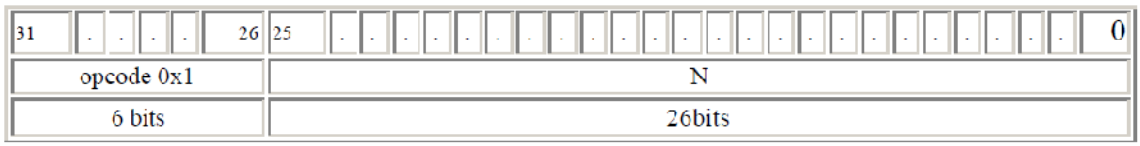
Branch if no flag –l.bnf

31 26	25 0
opcode 0x3	N
6 bits	26bits

Jump – l.j

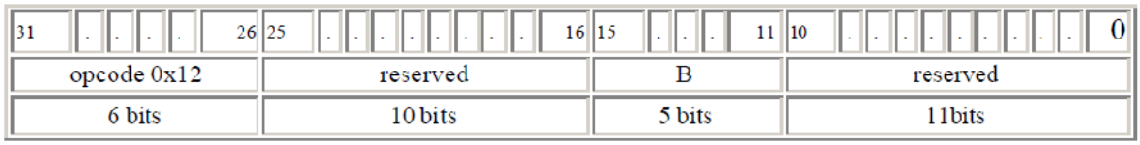
31 26	25 0
opcode 0x0	N
6 bits	26bits

Jump and link – l.jal



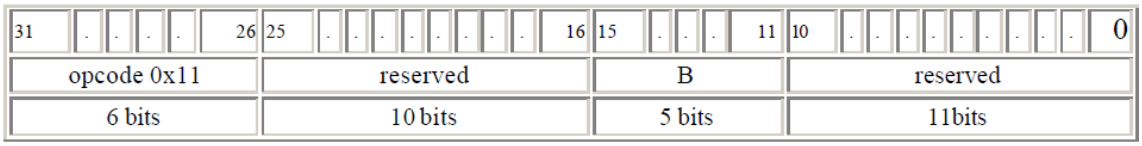
31 26	25 0
opcode 0x1	N
6 bits	26bits

Jump and link register – l.jalr



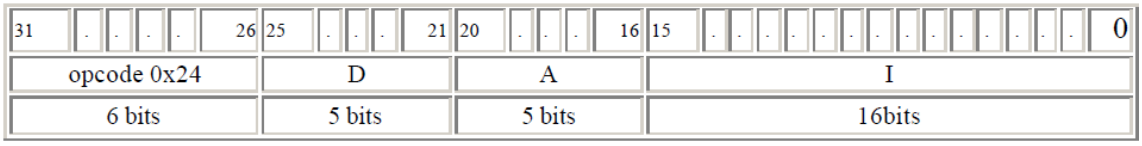
31 26	25 16	15 . . . 11	10 0
opcode 0x12	reserved	B	reserved
6 bits	10 bits	5 bits	11bits

Jump register – l.jr



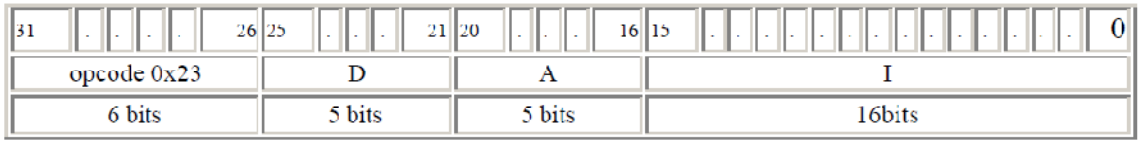
31 26	25 16	15 . . . 11	10 0
opcode 0x11	reserved	B	reserved
6 bits	10 bits	5 bits	11bits

Load byte and extend with signed – l.lbs



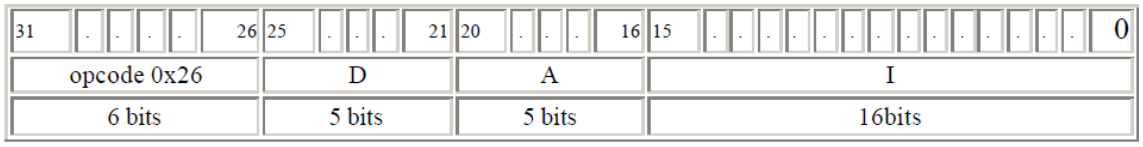
31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x24	D	A	I
6 bits	5 bits	5 bits	16bits

Load byte and extend with zero – l.lbz



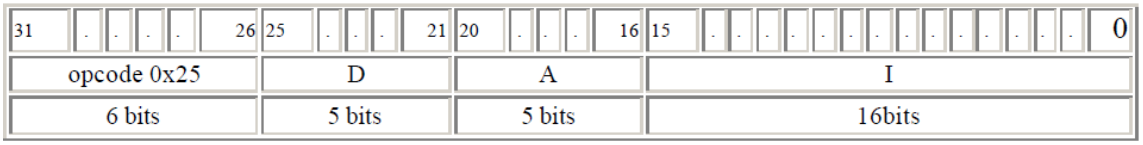
31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x23	D	A	I
6 bits	5 bits	5 bits	16bits

Load half word and extend with sign – l.lhs



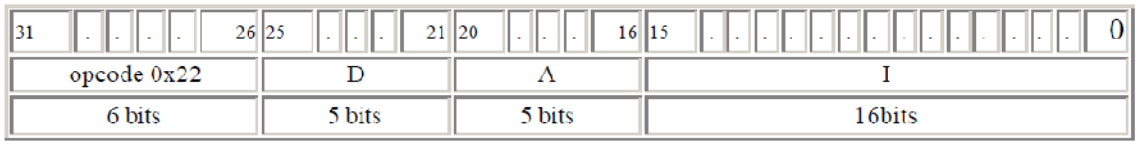
31 26	25 21	20 16	15 0
opcode 0x26	D	A	I
6 bits	5 bits	5 bits	16bits

Load half word and extend with zero – l.lhz



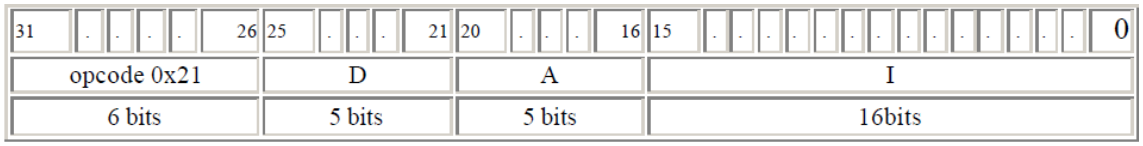
31 26	25 21	20 16	15 0
opcode 0x25	D	A	I
6 bits	5 bits	5 bits	16bits

Load single word extend with sign – l.lws



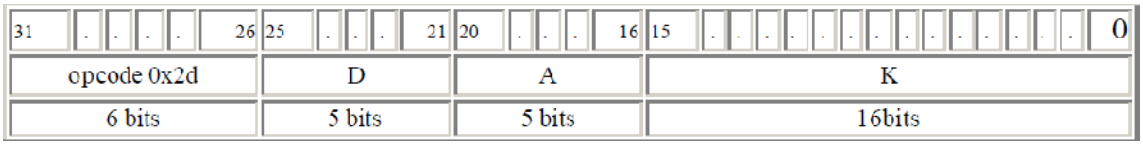
31 26	25 21	20 16	15 0
opcode 0x22	D	A	I
6 bits	5 bits	5 bits	16bits

Load single word extend with zero – l.lwz



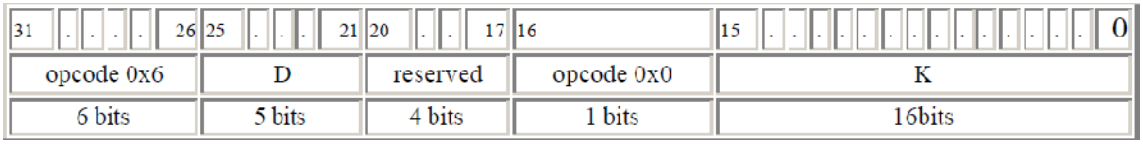
31 26	25 21	20 16	15 0
opcode 0x21	D	A	I
6 bits	5 bits	5 bits	16bits

Move from special-purpose register – l.mfspr



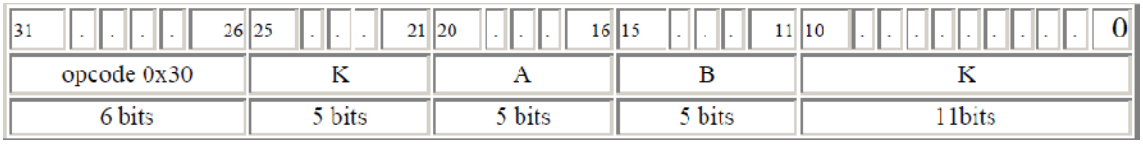
31 26	25 21	20 16	15 0
opcode 0x2d	D	A	K
6 bits	5 bits	5 bits	16bits

Move immediate high – l.movhi



31 26	25 21	20 . . . 17	16	15 0
opcode 0x6	D	reserved	opcode 0x0	K
6 bits	5 bits	4 bits	1 bits	16bits

Move to special-purpose register – l.mtspr



31 26	25 21	20 16	15 . . . 11	10 0
opcode 0x30	K	A	B	K
6 bits	5 bits	5 bits	5 bits	11bits

Multiply signed – l.mul

31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0x6
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

Multiply immediate signed – l.muli

31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x2c	D	A	I
6 bits	5 bits	5 bits	16bits

Multiply unsigned – l.mulu

31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0xb
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

No operation – l.nop

31 24	23 16	15 0
opcode 0x15	reserved	K
8 bits	8 bits	16bits

Or – l.or

31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x4
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

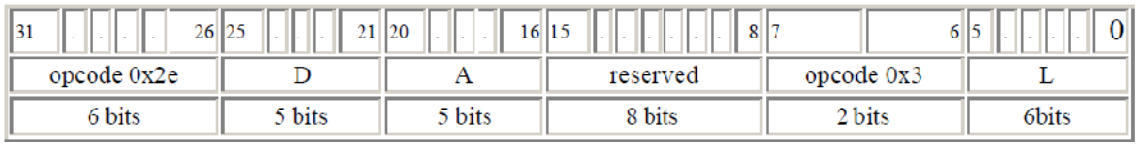
Or with immediate half word – l.ori

31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x2a	D	A	K
6 bits	5 bits	5 bits	16bits

Return from exception – l.rfe

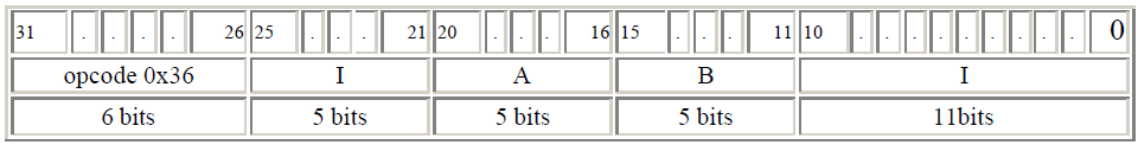
31 26	25 0
opcode 0x9	reserved
6 bits	26bits

Rotate right with immediate – l.rori



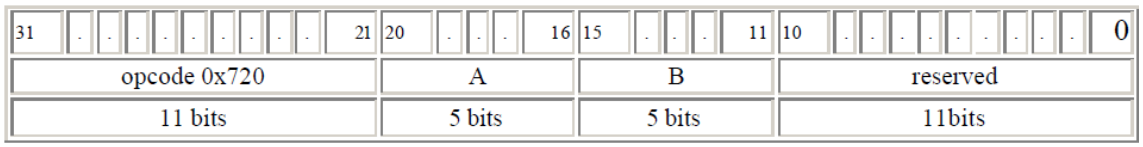
31 26	25 . . . 21	20 . . . 16	15 8	7 6	5 0
opcode 0x2e	D	A	reserved	opcode 0x3	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

Store byte – l.sb



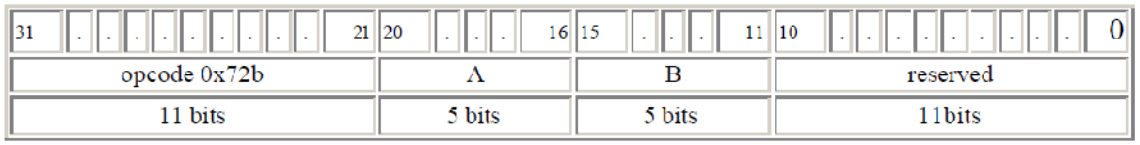
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10 0
opcode 0x36	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

Set flag if equal – l.sfeq



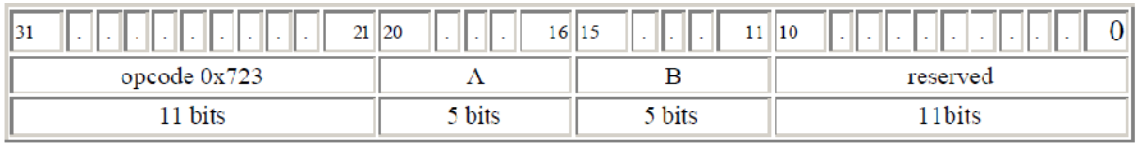
31 21	20 . . . 16	15 . . . 11	10 0
opcode 0x720	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if greater or equal than signed – l.sfges



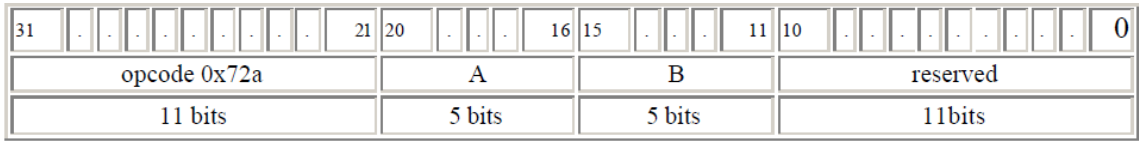
31 21	20 . . . 16	15 . . . 11	10 0
opcode 0x72b	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if greater or equal than unsigned – l.sfgeu



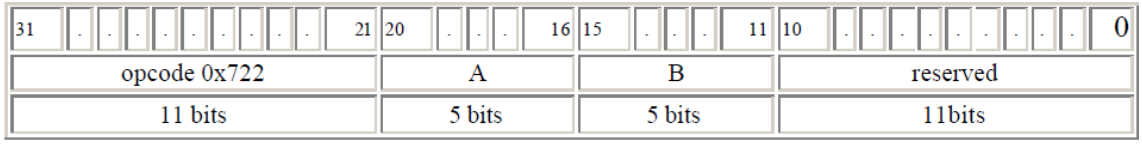
31 21	20 . . . 16	15 . . . 11	10 0
opcode 0x723	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if greater than signed – l.sfgts



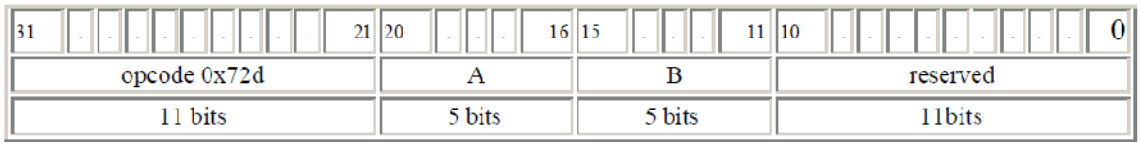
31 21	20 . . . 16	15 . . . 11	10 0
opcode 0x72a	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag id greater than unsigned – l.sfgtu



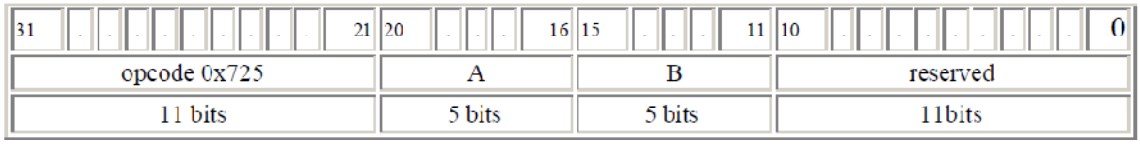
31 21	20 . . . 16	15 . . . 11	10 0
opcode 0x722	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if less or equal than signed – l.sfles



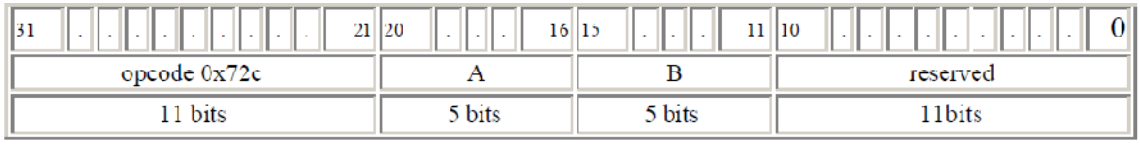
31 21	20 16	15 11	10 0
opcode 0x72d	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if less or equal than signed – l.sfleu



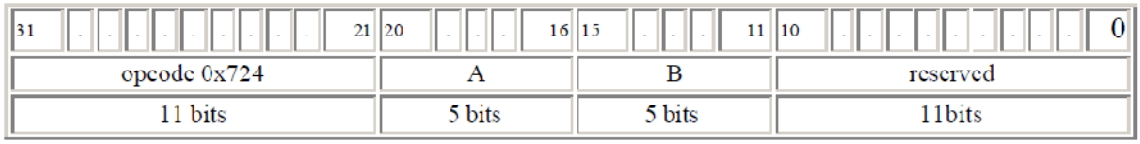
31 21	20 16	15 11	10 0
opcode 0x725	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if less than signed – l.sflts



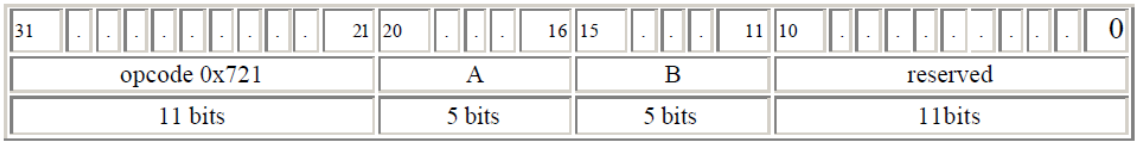
31 21	20 16	15 11	10 0
opcode 0x72c	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if less than unsigned – l.sfltu



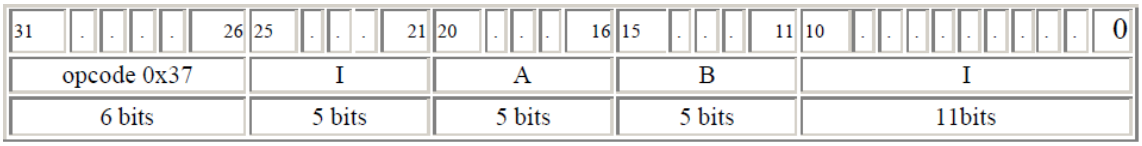
31 21	20 16	15 11	10 0
opcode 0x724	A	B	reserved
11 bits	5 bits	5 bits	11bits

Set flag if not equal – l.sfne



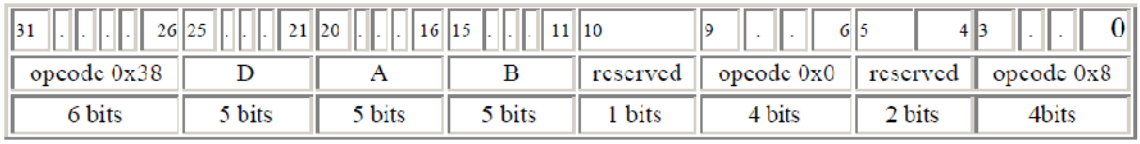
31 21	20 16	15 11	10 0
opcode 0x721	A	B	reserved
11 bits	5 bits	5 bits	11bits

Store half word – l.sh



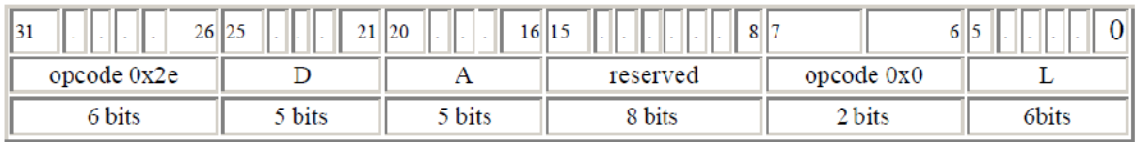
31 26	25 . . . 21	20 16	15 11	10 0
opcode 0x37	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

Shift left logical – l.sll



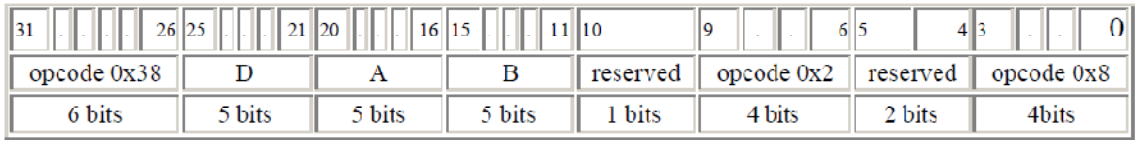
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 . . 6	5 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

Shift left logical with immediate – l.slli



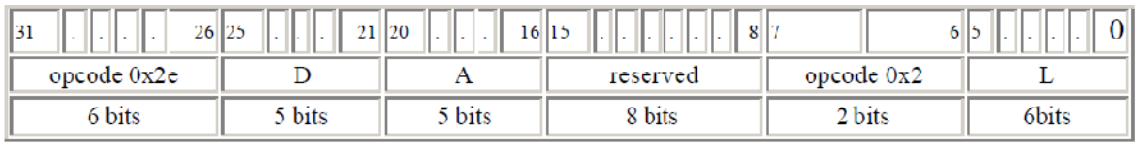
31 26	25 . . . 21	20 . . . 16	15 8	7 6	5 0
opcode 0x2e	D	A	reserved	opcode 0x0	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

Shift right arithmetic – l.sra



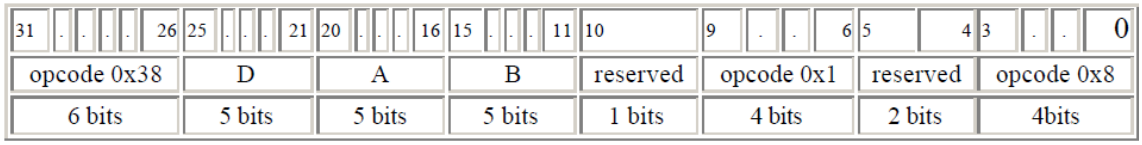
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 . . 6	5 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x2	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

Shift right arithmetic with immediate – l.srai



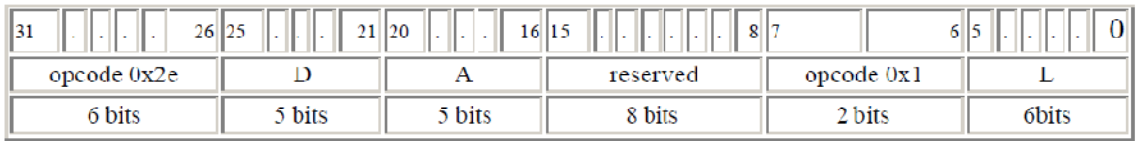
31 26	25 . . . 21	20 . . . 16	15 8	7 6	5 0
opcode 0x2e	D	A	reserved	opcode 0x2	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

Shift right logical – l.srl



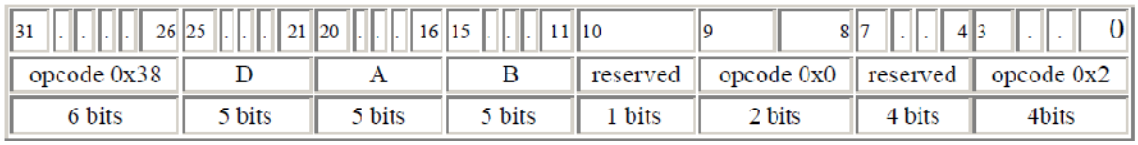
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 . . 6	5 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x1	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

Shift right logical with immediate – l.srli



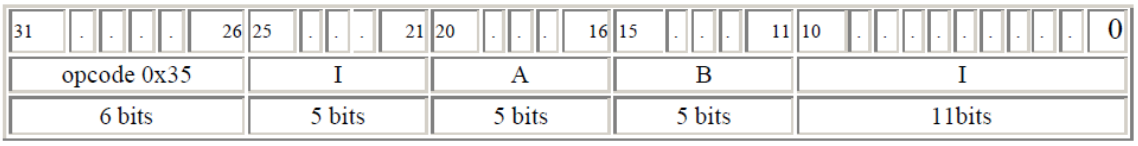
31 26	25 . . . 21	20 . . . 16	15 8	7 6	5 0
opcode 0x2e	D	A	reserved	opcode 0x1	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

Subtract signed – l.sub



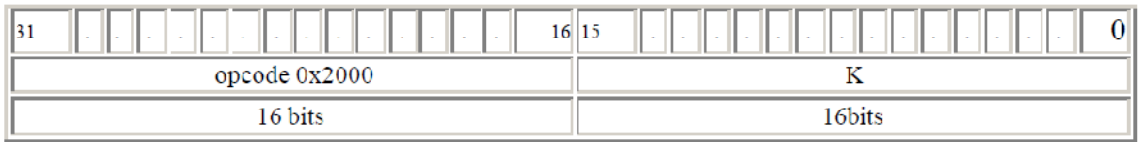
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x2
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

Store single word – l.sw



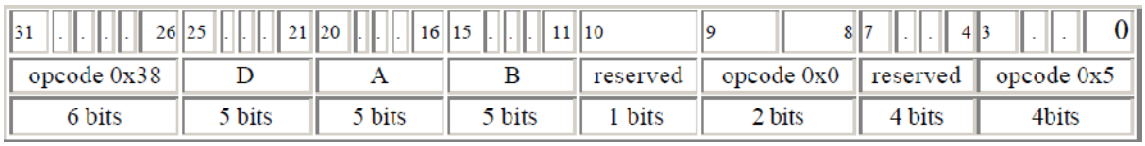
31 26	25 . . . 21	20 . . . 16	15 . . . 11	10 0
opcode 0x35	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

System call – l.sys



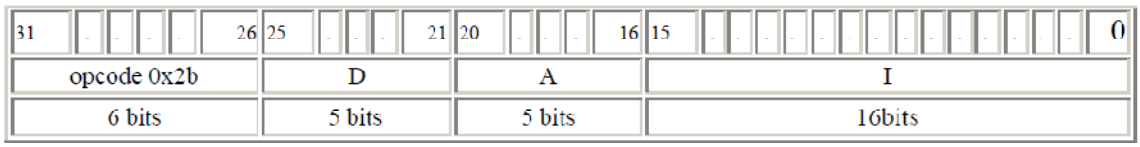
31 16	15 0
opcode 0x2000	K
16 bits	16bits

Exclusive or – l.xor



31 . . . 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x5
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

Exclusive or with immediate half word – l.xori



31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x2b	D	A	I
6 bits	5 bits	5 bits	16bits

Kazalo slik

Kazalo tabel

Literatura

[1] *Sipper, Moshe, "Evolution of parallel cellular machines" v zbirki Lecture notes in computer science, Berlin, 1997, str. 4-6.*

[2] *S. Wolfram, Universilitiy and completing in cellular automata, Physica D,1984, str. 1-35.*

[3] *(2001) OpenRISC 1200 IP core specification, Revision: 0.7. Dostopno na: http://www.opencores.org/svnget,or1k?file=/trunk/docs/openrisc1200_spec.pdf*

[4] *(2002) Memory controller IP core, Revision 1.7. Dostopno na: http://www.opencores.org/download,mem_ctrl,mem_ctrl_latest.tar.gz*

[5] *(2001) Synchronous DRAM MT48LC4M32B2. Dostopno na: http://download.micron.com/pdf/datasheets/dram/sdram/128MbSDRAMx32.pdf*

[6] *(2001) WISHBONE System-On-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision: B.1. Dostopno na: http://users.ece.utexas.edu/~chung/vlsi1/lab3/lab3b/wbspec_b1.pdf*