LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

UNIX veida mikrokodola operētājsistēma FPGA procesoram

MAĢISTRA DARBS

Autors: **Ansis Liepkalns**
Stud. apl. al06011
Darba vadītājs: Dr.comp.sc. Leo Seļāvo

RĪGA 2012

ANOTĀCIJA

Risinājumos, kuros izmanto programmējamo loģisko mezglu masīvu (FPGA) procesorus, programmatūras pieejamība ir svarīga, lai samazinātu galaprodukta iegūšanai nepieciešamo laiku. Plašu programmatūras atbalstu ir ieguvušas UNIX veida operētājsistēmas. To kombinācija ar FPGA procesoriem spēj nodrošināt vēlamo izstrādes ātrumu. Lai apmierinātu kvalitātes prasības, tiek piedāvāts izmantot mikrokodola operētājsistēmu. Darbā tiek apskatīta sistēmas mikroshēmā izveide darbībai ar „Minix 3“ mikrokodola operētājsistēmu.

ABSTRACT

Title: **UNIX-like microkernel operating system on FPGA processor**.

Software support is important for Field-Programmable Gate Array (FPGA) soft processor applications for reduced product time-to-market. UNIX-like operating systems have gained wide software availability. Using in combination with FPGA processors, the demanded development speed can be achieved. To assure the quality demands, use of microkernel operating system is suggested. This paper covers design of system-on-chip for operation with "Minix 3" microkernel operating system.

AUTOREFERĀTS

Maģistra darba izstrādes gaitā autors ir

- apskatījis iegulto sistēmu izstrādes problemātiku un aprakstījis nepieciešamās priekšzināšanas pārkonfigurējamās aparatūras un mikrokodolu operētājsistēmu jomā,
- izanalizējis mikrokodola *Minix 3* operētājsistēmas uzbūvi,
- projektējis un īstenojis sistēmu mikroshēmā, kas ir paredzēta darbībai ar mikrokodola operētājsistēmu,
- analizējis izveidotās sistēmas veiktspējas rādītājus,
- analizējis *Minix 3* pārnešanas iespējas uz izveidoto sistēmu, kā arī veicis pirmos pārnešanai nepieciešamos soļos.

Galvenais autora ieguldījums šajā darbā ir mikrokodola operētājsistēmai pielāgotas sistēmas mikroshēmā projektējums, īstenošana un darbības analīze.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS 4
IEVADS 6
1. MOTIVĀCIJA 7
2. ESOŠIE RISINĀJUMI 8
2.1. Operētājsistēmas 8
2.1.1. Linux 8
2.1.2. µClinux 8
2.1.3. QNX Neutrino 8
2.1.4. L4 8
2.2. Risinājumu problemātika 8
3. IESKATS APARATŪRĀ 10
3.1. Procesori iegultās sistēmās 10
3.2. Pārkonfigurējamā aparatūra 10
3.2.1. FPGA tehnoloģija 10
3.2.2. FPGA programmēšana 11
3.2.3. FPGA procesori 11
4. IESKATS PROGRAMMATŪRĀ 13
4.1. Operētājsistēmas loma iegultajās sistēmās 13
4.2. UNIX veida operētājsistēmas 13
4.3. Mikrokodola operētājsistēmas 13
5. PIEDĀVĀTAIS RISINĀJUMS 15
5.1. Minix 3 darbībai ar LatticeMico32 procesoru 15
5.2. Avnet Spartan-6 LX9 Microboard 15
5.2.1. Avnet aparatūra 15
5.2.2. Xilinx Spartan-6 LX9 16
5.3. LatticeMico32 procesors 16
5.4. Minix 3 operētājsistēma 17
5.4.1. Vēsture 17
5.4.2. Versija 17
5.4.3. Struktūra 17
5.4.4. Kodols 18
5.4.5. Serveri 18
5.4.6. Dziņi 19
6. SOC IZVEIDE 20
6.1. SoC projektējums 20
6.2. LatticeMico32 pielāgošana Spartan-6 20
6.2.1. Speciālie moduļi 20
6.2.2. Atmiņas kontrolleris 22

6.3. LatticeMico32 lietotāja un kodola režīmi . . . 24
6.3.1. Nepieciešamība . . . 24
6.3.2. LatticeMico32 priviliģētās instrukcijas . . . 25
6.4. LatticeMico32 MPU . . . 25
6.5. Ethernet saskarne . . . 26
6.5.1. Ethernet saskarnes pielietojums . . . 26
6.5.2. Media Independent Interface . . . 26
6.5.3. Izveidotais protokols . . . 27
6.6. FPGA resursu izmantošana . . . 27
6.7. SoC testi . . . 28
6.7.1. Nepieciešamība un metodika . . . 28
6.7.2. Atmiņas pārbaude . . . 28
6.7.3. GPIO testa programma . . . 29
6.7.4. Testi dažādām SoC konfigurācijām . . . 31
6.7.5. Konteksta pārslēgšana . . . 32
7. MINIX 3 PĀRNEŠANA . . . 34
7.1. Programmatūras izstrādes rīki un metodes . . . 34
7.2. Kompilācija . . . 34
7.3. Arhitektūras atkarība . . . 35
7.4. Konteksta pārslēgšana . . . 35
7.5. Pozīcijas neatkarīgs kods . . . 36
7.6. Piekļuve perifērijas iekārtām . . . 38
REZULTĀTI . . . 39
SECINĀJUMI . . . 40
DARBS NĀKOTNĒ . . . 41
LITERATŪRAS SARAKSTS . . . 42
PIELIKUMS . . . 44
1. pielikums: „Galvenais modulis“ . . . 44
2. pielikums: „Atmiņas modulis“ . . . 48
3. pielikums: „Atmiņas kontrollera modulis“ . . . 52
4. pielikums: „Ethernet programmēšanas un atkļūdošanas modulis“ . . . 56
5. pielikums: „Atmiņas aizsardzības modulis“ . . . 64
6. pielikums: „Atkļūdošanas rīks datoram“ . . . 67

Attēlu saraksts

3.1. Vispārināta FPGA struktūra . . . 10
4.1. Mikrokodola un monolītisko sistēmu salīdzinājums . . . 14
5.1. Avnet Spartan-6 LX9 Microboard izstrādes rīks . . . 15
6.1. SoC projektējums . . . 21
6.2. GPIO testa programmas oscilogrāfa uzņēmumi . . . 29

a bez kešatmiņas 29
b ar kešatmiņu 29
6.3. GPIO testa programmas *ChipScope* uzņēmumi 30
a bez kešatmiņas 30
b ar kešatmiņu 30
6.4. Konteksta pārslēgšanas testa programmas oscilogrāfa uzņēmumi 33
a bez kešatmiņas 33
b ar kešatmiņu 33
7.1. Pozīcijas neatkarīga koda izmantošanas piemērs 37

Tabulu saraksts

6.1. Atmiņas saskarne ar procesoru 23
6.2. MIG piedāvātā saskarne ar atmiņu 24
6.3. Atmiņas izdalījums procesam „vfs“ 26
6.4. SoC FPGA izmantotie resursi 28
6.5. Konfigurāciju veiktspējas salīdzinājumi 32

APZĪMĒJUMU SARAKSTS

ACK *Amsterdam Compiler Kit*, kompilācijas rīku kolekcija. 17

CLB *Configurable Logic Block*, konfigurējams loģikas bloks. 10, 16

COTS *Commercial off-the-shelf*, komerciāla, viegli integrējama trešās puses programmatūra. 6, 7, 13

FDPIC *Function Descriptor Position Independent Code*, pozīcijas neatkarīgs kods, kas izmanto funkciju deskriptorus. 37, 41

FIFO *First In, First Out*, rindošanas disciplīna „pirmais iekšā, pirmais ārā". 20, 22

FPGA *Field-Programmable Gate Array*, Programmējamais loģisko mezglu masīvs. 8–12, 15, 16, 20, 27, 39, 40

FPU *Floating-Point Unit*, peldošā komata aritmētikas bloks. 12, 35

GCC *GNU Compiler Collection*, „GNU" kompilācijas rīku kolekcija. 17, 34, 36

GOT *Global Offset Table*, globālā nobīžu tabula. 36, 37

GPIO *General Purpose Input/Output*, vispārēja pielietojuma ievade / izvade. 20, 28, 29, 32

JTAG *Joint Test Action Group*, iekārtas testēšanas ports, izmanto arī iekārtas atkļūdošanai un programmēšanai. 16

LLVM *Low Level Virtual Machine*, kompilācijas rīku kolekcija. 17, 34

LUT *Lookup Table*, pārlūktabula. 10

MCB *Memory Controller Block*, *Xilinx Spartan-6* atmiņas kontrollera elements. 22

MIG *Memory Interface Generator*, *Xilinx* atmiņas saskarnes ģenerators. 22

MII *Media Independent Interface*, datu nesēja neatkarīga saskarne. 26

MMU *Memory Management Unit*, atmiņas pārvaldības bloks. 12, 15, 25, 35

MPU *Memory Protection Unit*, atmiņas aizsardzības bloks. 8, 15, 25, 26, 35, 37

NOMMU *No Memory Management Unit*, apzīmē sistēmas, kas neizmanto atmiņas pārvaldības bloku. 8, 41

PIC *Position Independent Code*, pozīcijas neatkarīgs kods. 36

PLL *Phase-Locked Loop*, „fāzes cilpa" automātiskai frekvences pieskaņošanai. 16, 20

POSIX *Portable Operating System Interface*, *Institute of Electrical and Electronics Engineers* (*IEEE*) standartu saime, kas norāda programmēšanas saskarsnes programmatūras un operētājsistēmu saderībai. 13, 17

QEMU *Quick EMUlator*, procesora emulators. 34

RAM *Random Access Memory*, brīvpiekļuves atmiņa. 16, 19

RISC *Reduced Instruction Set Computing*, sašaurinātās instrukcijkopas dators. 11, 16

SIMD *Single instruction, Multiple data*, vienas instrukcijplūsmas un daudzu datplūsmu arhitektūra. 12

SoC *System-on-Chip*, sistēma mikroshēmā. 10, 15, 20, 27, 41

TCB *Trusted Computing Base*, uzticamā skaitļošanas sistēmas bāze. 14

UART *Universal Asynchronous Receiver-Transmitter*, universālais asinhronais raiduztvērējs. 16, 20

UNIX Pārnesama, vairāklietotāju un vairākuzdevumu operētājsistēma, ko 70. gadu sākumā izstrādāja firmas *AT&T* pētniecības centrā *Bell Laboratories*. 6, 8, 9, 13

USB *Universal Serial Bus*, universālā seriālā kopne. 16, 20

IEVADS

Augstu stabilitāti nodrošinošas operētājsistēmas, atrod savu vietu iegulto sistēmu sfērā, kur programmatūras avārijas var būt kritiskas pašas sistēmas darbībai un tam, ko šī sistēma nodrošina.

Nestabilitāti izraisošs kods var būt gan pašu rakstīts, gan arī trešās puses COTS veida piedāvājums. Vispārēja programmatūras kļūdu izskaušana var neatmaksāties gadījumos, kad stabilitātes nodrošināšanai pietiek ar programmu savstarpēju izolāciju, sadalot sistēmu loģiskās daļās. Šādu iespēju piedāvā mikrokodola operētājsistēmas, kas nodrošina labu izolāciju arī pašas operētājsistēmas daļām. Ar elastību, ko sniedz pārkonfigurējamā aparatūra, tiek pavērstas jaunas iespējas šādas operētājsistēmas nodrošināšanai.

Izmantojot atvērtā koda operētājsistēmu un atvērtā koda procesoru, izstrādātājs nav ierobežots attiecībā uz savas sistēmas pilnveidošanu jebkurā jomā — izstrādātājs var mainīt procesora saskarņu skaitu un veidu, pievienot jaunas saskarnes vai pat instrukcijas un blakusprocesorus. Šādas iespējas izstrādātājus atbalsta nestandarta izstrādājumu izgatavošanā un vienkāršo jau izstrādātu sistēmu attīstību, samazinot nepārkonfigurējamās aparatūras modificēšanas nepieciešamību.

Šobrīd izmantotās iegulto sistēmu operētājsistēmas ir vai nu komerciālas (*QNX*) vai atsevišķos gadījumos nav pietiekami modulāras (*Linux*). Elastīgam aparatūras un programmatūras risinājumam ir izvēlēts „LatticeMico32“ procesors un pēdējos gados progresējusī „Minix 3“ UNIX veida mikrokodola operētājsistēma, kura tiek attīstīta pētniecības tēmas ietvaros, kas ir 2008. gadā ieguvusi 2,5 miljonu eiro finansējumu no Eiropas Zinātnes padomes (*European Research Council*) [22].

Darba pirmā nodaļa izklāsta motivāciju šī darba veikšanai. Otrajā nodaļā tiek apskatīti izmantotie operētājsistēmu risinājumi un uzskaitīta to problemātika. Trešajā nodaļā aprakstītas priekšzināšanas aparatūras jomā, bet ceturtajā nodaļā ir sniegts ieskats programmatūras jomā. Piektajā nodaļā ir aprakstīts piedāvātais problēmas risinājums. Risinājuma veikšana ir aprakstīta sestajā (sistēmas mikroshēmā izveide) un septītajā nodaļā (apsvērumi „Minix 3“ pārnešanai).

1. MOTIVĀCIJA

Iegultās sistēmas uzlabo dzīves kvalitāti, piedāvājot arvien sarežģītākas un augstāku funkcionalitāti nodrošinošas sistēmas: mājas automatizācija, viedie telefona aparāti, bezvadu maršrutētāji ir tikai daļa no iegultajām sistēmām, ko cilvēki izmanto ikdienā. Netiešā veidā tiek izmantoti, piemēram, sakaru tīkli, kurus veido apjomīgs datu pārraides iekārtu skaits, kur katra no iekārtām var būt noteicoša gala lietotāja izmantoto servisu kvalitātē [8].

No darba pieredzes iegulto sistēmu izstrādes sfērā, autors secina, ka mūsdienīgu iekārtu izstrādē a) COTS veida programmatūra var būtiski paātrināt izstrādes gaitu, bet tās stabilitāte var būt apšaubāma un izolācija no pārējām sistēmas daļām uzlabo kopējo sistēmas stabilitāti; b) bezmaksas atvērtā koda risinājumi, kas nodrošina labu to programmatūras komponenšu izolāciju nav tūlītēji izmantojami iegultajām sistēmām; c) plaši izmantotās „Linux“ operētājsistēmas pielietošana ir sarežģīta „copyleft“ veida licences dēļ; d) iekārtas nepārkonfigurējamā elektriskā plānojuma izmaiņas var dažkārt izrādīties dārgas, lai pievienotu jaunas iespējas vai arī tikai nodrošinātu sadarbības iespējas ar mikroshēmām, kuru konfigurāciju pilnībā nodrošina trešo pušu izstrādātāji un kuru konfigurācijas izmaiņas var ietekmēt saskarņu darbību.

„Copyleft“ veida licenču ievērošana sagādā problēmas daudziem iegulto sistēmu ražotājiem, kas nevēlas publiskot izmantoto atvērto kodu [12]. Izvairoties no šāda veida licencēm un izmantojot elastīgu, viegli pārnesamu atvērtā koda pārkonfigurējamās aparatūras un stabilitāti nodrošinošu, modulāru programmatūras risinājumu, izstrādātājiem būtu iespēja nodrošināt un attīstīt iegulto iekārtu ar konkurētspējīgu izstrādes ātrumu un kvalitāti.

2. ESOŠIE RISINĀJUMI

2.1. Operētājsistēmas

2.1.1. Linux

Linux ir 1991. gadā izveidota monolītiska atvērtā koda operētājsistēma, kas tiek plaši izmantota iegulto sistēmu izstrādē. Tās galvenie plusi ir plašais aparatūras atbalsts un programmatūras pieejamība.

Linux piedāvā ierobežotu aizsardzību no kļūdainiem dziņiem, kas tiek dēvēta par „kernel oops“: ja dziņa izpildes laikā notiek izņēmums (cēlonis var būt, piemēram, vēršanās 0. atmiņas adreses jeb *NULL pointer*), šī dziņa darbība tiek pārtraukta, bet sistēma ir spējīga turpināt darbību [8].

Linux ir saderīgs ar galveno FPGA ražotāju izstrādātajiem procesoriem (*NIOS II*, *MicroBlaze*, *LatticeMico32*).

2.1.2. µClinux

µClinux ir *Linux* modifikācija, kas nodrošina šīs sistēmas darbību NOMMU sistēmās. Šī *Linux* versija tiek attīstīta arī tādā virzienā, lai tā būtu resursu prasību ziņā minimāla, tādā veidā gūstot vēl plašāku aparatūras saderību [17].

2.1.3. QNX Neutrino

QNX Neutrino ir *QNX Software Systems* izstrādāta komerciāla reāllaika mikrokodola operētājsistēma [11]. Tā ir dubulti licencēta — tās kods ir pieejams akadēmiskam pielietojumam bez maksas, bet komerciāliem risinājumiem tās izmantošanas tiesības tiek pārdotas [25].

2.1.4. L4

„L4“ mikrokodols ir gan ātrdarbīgs, gan arī plaši izmantots [23]. Kodols nepiedāvā UNIX veida saskarni, bet to var izmantot, lai darbinātu, piemēram, „Linux“ lietotāja režīmā, tādā veidā iegūstot vienu drošu mikrokodolu („L4“) un vienu vai vairākas lietotāja režīma operētājsistēmas / citas programmas [10].

2.2. Risinājumu problemātika

Linux operētājsistēmai pilnīgu avāriju var izraisīt kļūda dzinī vai jebkurā no tās komponentēm, kas nav lietotāja programmas. *µClinux* gadījumā daudzkārt ir sliktāk — tajā arī lietotāja režīma programmas var izraisīt sistēmas avāriju, jo izmanto vienu adrešu telpu. Tādi procesori kā *Analog Devices* „Blackfin“ izmanto MPU, lai šo situāciju uzlabotu [3].

Cits *Linux* operētājsistēmas trūkums no iegulto sistēmu izstrādātāju skata punkta var

būt tā „copyleft“ veida licence „GNU General Public Licence v2.0“. Nepieciešamība publicēt izmainītās sistēmas daļas, kas licencētas ar šo licenci, kas var būt par cēloni šo operētājsistēmu neizmantot vai izmantot nelikumīgi [12].

Neutrino mīnuss ir tā licence, kas ir negatīvi ietekmējošais faktors attiecībā uz tā pieejamību darbībai ar pārkonfigurējamiem procesoriem. Darba tapšanas brīdī šī operētājsistēma nav paredzēta darbībai ar galveno FPGA ražotāju izstrādātajiem *Microblaze* un *Nios II* procesoriem.

L4 kodols arī ir licencēts ar „copyleft“ veida licenci. Tas var tikt izmantots kā zemākais UNIX veida sistēmas līmenis vai uzraudzītājs, bet tādā gadījumā tam komplektā ar virssistēmu nāk arī tās problēmas.

3. IESKATS APARATŪRĀ

3.1. Procesori iegultās sistēmās

Mūsdienīgas iegultās sistēmas satur vismaz vienu vispārēja pielietojuma procesoru, kas paredzēts sistēmas datu apstrādes un kontroles programmatūras darbībai. Šādam procesoram ir jāsadarbojas ar tā perifērijām — citām mikroshēmām. Sadarbību nodrošina datu kopnes, kurās izmanto standartizētus saziņas protokolus.

Tipiski integrēto mikroshēmu procesori piedāvā fiksēta skaita un veida datu kopnes; arī to arhitektūra un instrukciju kopa ir neelastīga attiecībā uz izmaiņām — projektējot sistēmu ar šādu procesoru, izstrādātājam ir jāparedz ne tikai sadarbības perifērijas, bet arī programmatūra, kas tiks izmantota.

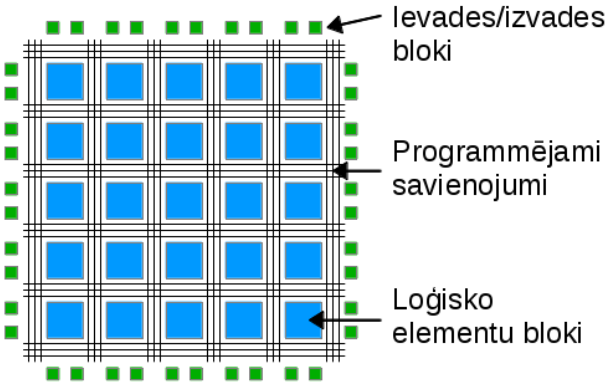
Pārkonfigurējamas aparatūras izmantošanas izvēle gūst popularitāti, mazinot kļūdainas izvēles risku un padarot sistēmas attīstības elastību lielāku. FPGA mikroshēmas ir viens no pārkonfigurējamās aparatūras veidiem, kas ļauj izveidot aparatūras (arī procesora) konfigurāciju sistēmas darbības laikā (*field-programmable*).

FPGA mikroshēmas var veikt vairāk kā vienkārša procesora darbību: tās var nodrošināt arī, piemēram, digitālu signālu apstrādi, iekļaut tipiskus un arī specifiskus kontrollerus. Tādā veidā iespējams izveidot pilnvērtīgu sistēmu vienā mikroshēmā jeb SoC.

3.2. Pārkonfigurējamā aparatūra

3.2.1. FPGA tehnoloģija

FPGA jeb *Field-Programmable Gate Array*, kas ir viena no pārkonfigurējamās aparatūras veidiem, ir programmējams loģisko mezglu masīvs. Tā priekšrocība ir liels loģisko elementu blīvums un neierobežotas pārkonfigurēšanas reizes. Vispārināta FPGA struktūra parādīta attēlā 3.1. [6].



3.1. att. **Vispārināta FPGA struktūra**

Lai radītu FPGA, tiek izmantotas dažādas metodes un to detalizētāka uzbūve var būt

krietni atšķirīga starp dažādiem ražotājiem. Tā piemēram, loģisko elementu bloks jeb CLB var saturēt dažādus apakšelementus. Galvenokārt tiek izmantoti LUT elementi, kas spēj uzglabāt nelielu informācijas daudzumu (16 – 64 biti). Tomēr modernas FPGA mikroshēmas satur arī dažādus sarežģītus elementus: reizinātājus, iebūvētos kontrollerus, zibatmiņas un arī gatavus (nepārkonfigurējamus) procesorus [6].

3.2.2. FPGA programmēšana

Pārkonfigurējamu aparatūru var aprakstīt dažādās valodās: gan aprakstot zema līmeņa loģisko elementu konfigurāciju, gan arī mazāk precīzi — translējot no augsta līmeņa valodām (piemēram, *C* vai *C++*) [9].

„VHDL“ un „Verilog“ ir tipiskās valodas, ko izmanto FPGA projektos [9]. Ar šo valodu palīdzību var aprakstīt saistību starp signāliem laikā. Tā, piemēram, sekojošā veidā var aprakstīt multipleksoru ar 2 viena bita ieejām valodā Verilog:

```
1 module mux(d1, d2, select, q);
2   input d1;
3   input d2;
4   input select;
5   output q;
6
7   assign q = select ? d2 : d1;
8 endmodule
```

No tāda koda ar FPGA ražotāja rīku palīdzību var sintezēt aparatūras konfigurācijas failu formātā, kas ir ielādējams FPGA mikroshēmā.

Atkļūdošanai un darbības korektuma pārbaudei var izmantot simulāciju. Programmai, kas nodrošina simulāciju ieejā tiek dots kods (un, iespējams, papildu faili) un tests, bet izejā var iegūt laika diagrammas. Otrs atkļūdošanas rīks ir iekšējais loģikas analizators, kas spējīgs ierakstīt un attēlot laika diagrammas no reālas aparatūras darbības. *Xilinx ChipScope* ir viens no rīkiem, kas tiek piedāvāts šādām vajadzībām [4].

3.2.3. FPGA procesori

FPGA nozarei tiek attīstīti procesori, kas ir optimizēti darbībai FPGA. To ņemot vērā, no mūsdienīgās procesoru saimes „x86“ ir pieejami vien vienkāršākie to FPGA atveidojumi, atstājot lielu tās programmatūras apjomu nesaderīgu.

Lielākie ražotāji, *Xilinx* un *Altera*, attīsta slēgtā koda procesorus — „MicroBlaze“ un „Nios II“. Lattice Semiconductors piedāvā atvērtā koda procesoru „LatticeMico32“, kura izmantošana ir atļauta bez atsaucēm uz „Lattice“ un bez koda publikācijas (atsauces nepieciešamas tikai koda izplatīšanas gadījumā). Kopīgais šiem procesoriem ir RISC un Hārvarda arhitektūra [26].

Eksistē arī atvērtā koda procesori („OpenFire“, „aeMB“, „MB-Lite“, „SecretBlaze“), kas atveido „Microblaze“, piedāvājot ierobežotu bināru saderību un ļauj izmantot tos pašus kompilā-

cijas un atkļūdošanas rīkus, kas izstrādāti priekš „MicroBlaze“ [2, 18]. Šie procesori atsevišķās jomās pārspēj oriģinālo „MicroBlaze“, tomēr attīstības tempa ziņā netiek līdzi *Xilinx*.

„LEON3“ un „LEON4“ ir atvērtā koda procesori, kas nodrošina „SPARC V8“ saderību un 32 vai 64 bitu instrukciju kopas arhitektūru, un tiek piedāvāti ar „copyleft“ vai komerciāla veida licenci [1, 26]. „OpenRISC“ atvērtā koda procesors līdzīgi ar „LEON“ procesoriem ir iespējām bagāts (SIMD instrukcijas, FPU, MMU) [26]. Tomēr tiem nepieciešamie FPGA resursi ir salīdzinoši augsti [19].

4. IESKATS PROGRAMMATŪRĀ

4.1. Operētājsistēmas loma iegultajās sistēmās

Mazos iegulto sistēmu projektos ir iespējams iztikt bez operētājsistēmas, ļaujot programmatūrai darboties tiešā veidā ar aparatūru. Pieaugot sistēmas sarežģītībai, aparatūras pārvaldīšanu un multipleksēšanu ir izdevīgi nodrošināt ar operētājsistēmas palīdzību. Vienveidota dziņu saskarne, bibliotēka un iebūvētās iespējas, saīsina programmatūras izstrādes laiku un piedāvā saderību ar jau pieejamo programmatūru.

4.2. UNIX veida operētājsistēmas

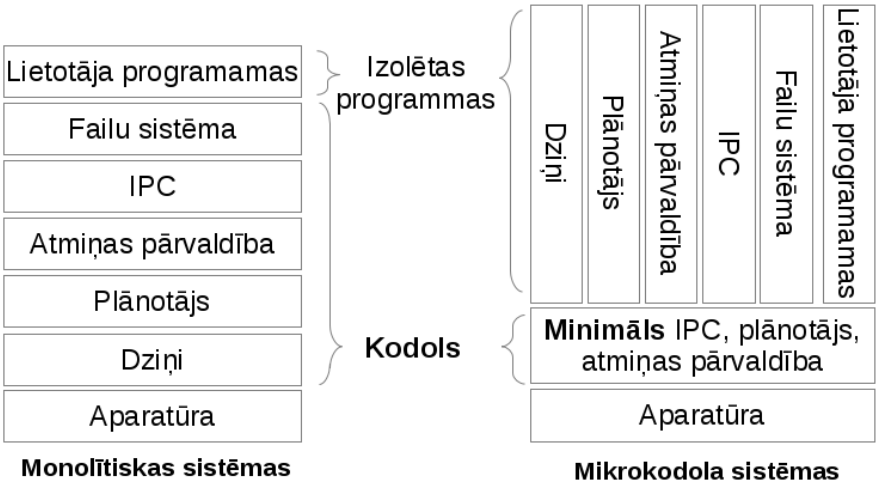
UNIX operētājsistēma ir pazīstama jau no 70. gadu sākuma. Tās koncepti un jēdzieni tiek izmantoti arī mūsdienīgās operētājsistēmās un ir standartizēti *Single UNIX Specification* standartā, kura pēdējās versijas sakrīt ar POSIX (*Portable Operating System Interface*) standartu. Lai operētājsistēmu sauktu par „UNIX“, tai ir jāatbilst „Single UNIX Specification“ standartam. Sertifikācija ne vienmēr ir finansiāli izdevīga, bet atbilstība var būt nodrošināta arī bez tās, paliekot pie nosaukuma „UNIX veida“ (*UNIX-like*) [15].

Tipiski COTS veida risinājumi tiek piedāvā sistēmām ar POSIX saskarni. Liela daļa bezmaksas atvērtā koda programmatūras ir pieejama *UNIX* veida sistēmām. Pieejamā programmatūra ir par iemeslu izstrādātājiem izvēlēties ar to saderīgu operētājsistēmu, lai saīsinātu sistēmas izstrādes laiku.

4.3. Mikrokodola operētājsistēmas

Operētājsistēmu organizācijas ir dažādas. Viengabalainu jeb monolītisku sistēmu komponentes ir līdzvērtīgas; piemēram, printera dzinis un procesu pārvaldības komponentes ir līdzvērtīgas attiecībā uz to kvalitātes sekām. Printera dziņa avārija var ietekmēt visas sistēmas darbību. Ja pieņem, ka procesu pārvaldības komponentes kļūdas ir izskaustas, veicot apjomīgu testēšanu, tad mazākas nozīmes daļas var būt palikušas bez tik augstas kvalitātes nodrošinājuma.

Turpretim mikrokodola operētājsistēmas ir sadalītas un tās kodola koda apjoms tiek veidots pēc iespējas minimāls, atstājot pārējās sistēmas komponentes, tai skaitā dziņus ārpus kodola, kur tās ir izolētas savstarpēji un no kodola. Vizuāli attēlota mikrokodolu un monolītisko kodolu sistēmu uzbūve ir 4.1. attēlā.



4.1. att. **Mikrokodola un monolītisko sistēmu salīdzinājums**

Samazinot kodola koda apjomu, tiek iegūta augsta modularitāte un vienkāršota sistēmas uzturēšana. Maksa par šādu modularitāti ir veiktspēja, kas tiek zaudēta moduļu savstarpējas komunikācijas un pārslēgšanās dēļ. Maza apjoma kritiskais kods var veidot arī pārskatāmu uzticamo sistēmas bāzi jeb *Trusted Computing Base* (TCB), vienkāršojot sistēmas drošuma īpašību auditēšanu un palielinot TCB kvalitāti.

5. PIEDĀVĀTAIS RISINĀJUMS

5.1. Minix 3 darbībai ar LatticeMico32 procesoru

Tiek piedāvāts izveidot sistēmu mikroshēmā (SoC), kuras aparatūru nodrošina *LatticeMico32* procesors *Xilinx Spartan-6* FPGA mikroshēmā, bet programmatūru — „Minix 3“ atvērtā koda mikrokodola *UNIX* veida operētājsistēma. Neviena no šīs sistēmas komponentēm nesatur „copyleft“ veida licencētas sastāvdaļas, kas brīvi ļauj izmantot to un tās modificētos variantus komerciālos risinājumos neuzstādot prasību koda publikācijai.

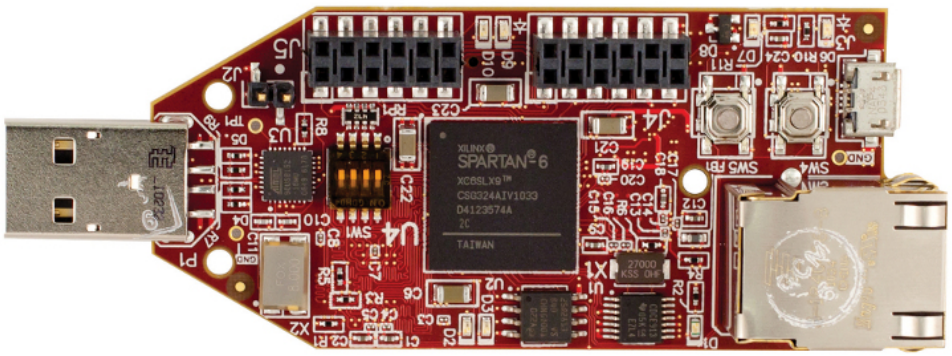
Ņemot vērā, ka *LatticeMico32*, lai gan 32 bitu, ir neliels un vienkāršs procesors, kuram nav MMU. Lai mikrokodola operētājsistēma attiecīgi spētu darboties, tai ir nepieciešama atmiņas aizsardzība. Tāpēc tiek piedāvāts izveidot atmiņas aizsardzības bloku (MPU).

Lai šādu sistēmu izveidotu ir nepieciešami pielāgojumi *LatticeMico32* procesoram, jāpielāgo „Spartan-6“ atmiņas kontrolleris, jāveic fundamentālas izmaiņas „Minix 3“ operētājsistēmai.

5.2. Avnet Spartan-6 LX9 Microboard

5.2.1. Avnet aparatūra

Avnet Spartan-6 LX9 Microboard ir *Avnet, Inc* izveidots izstrādes rīks. Tā izvēle pamatojama ar relatīvi nelielo cenu (89 ASV dolāri darba tapšanas laikā) un tajā pieejamo *Spartan-6* FPGA mikroshēmu. Izstrādes platforma redzama 5.1. attēlā.



5.1. att. **Avnet Spartan-6 LX9 Microboard izstrādes rīks**

Izstrādes rīks sastāv no:

- *Xilinx Spartan-6 LX9*;

- 64 megabaitu *Low Power Double Data Rate Static Random Access Memory* (*LPDDR SDRAM* atmiņas;
- 128 megabitu zibatmiņas;
- *Fast Ethernet* fiziskā līmeņa (*PHY*);
- programmējama takts devēja;
- UART saskarnes caur USB;
- JTAG saskarnes caur USB;
- 4 vispārēja pielietojuma slēdžiem un 4 gaismas diodēm [5].

5.2.2. Xilinx Spartan-6 LX9

Viena no mūsdienīgajām FPGA mikroshēmu sērijām ir lielākā FPGA ražotāja pēc tirgus daļas, *Xilinx*, radītā „Spartan-6“. Šīs sērijas 45 nanometru tehnoloģijas FPGA izceļas ar zemo enerģijas patēriņu, kas ir konkurētspējīgs ar nepārkonfigurējamiem mikroprocesoriem.

Spartan-6 CLB elements sastāv no divām šķēlēm (*slices*). Katra šķēle satur četrus 6 ieeju *LUT*, astoņus trigerus un citus elementus, kas ir atkarīgi no šķēles veida; ir 3 šķēļu veidi: 25% ir tādas, kas var būt arī atmiņa („Distributed RAM“ jeb izdalītais RAM) vai bīdes reģistri, kā arī piedāvā aritmētisko darbību pārnešanas iespēju, vēl 25% ir tādas pašas bez bīdes reģistriem un atlikušie 50% ir kā iepriekšējie bez aritmētiskās pārnešanas un platiem multipleksoriem [27].

Ir arī šādi speciāli elementi: „DCM“ jeb „Digital Clock Manager“, kas kombinācijā ar PLL var tikt izmantots takts frekvenču sintēzei, iekšējā atmiņa „Block RAM“, kas „LX9“ piedāvā 576 kilobitu glabātuvi, gMCB, „DSP48A1“, kas piedāvā reizinātājus un akumulatorus [27].

Izvēlētais „LX9“ no šīs sērijas ir viens no mazākajiem *Spartan-6* FPGA pēc loģikas elementu skaita — *Xilinx* piedāvā no „LX4“ līdz „LX150“, kur skaitlis nosaukumā apzīmē loģisko elementu skaitu tūkstošos.

5.3. LatticeMico32 procesors

LatticeMico32 ir *Lattice Semiconductor Corporation* izstrādāts atvērtā koda procesors. Šī procesora kodu iespējams iegūt ar programmu „Lattice Diamond“, kura piedāvā izveidot procesora konfigurāciju, izmantojot grafisko interfeisu [13].

Procesors nodrošina RISC instrukciju kopu un Hārvarda arhitektūru atmiņas pieejai. Tam ir 6 stāvokļu konveijers un 32 vispārēja pielietojuma reģistri. Atsevišķi iespējams konfigurācijā norādīt reizināšanas, dalīšanas, bitu pārbīdes iespējas. Procesoram iespējams pievienot arī *write-through* veida kešatmiņu (ar 2 vai 1 kanālu asociativitāti). Iespēju konfigurācija atspoguļojas aparatūras izmantotajos resursos [14].

LatticeMico32 piedāvā pārtraukumu apstrādi (32 ārēji pārtraukumi) un *Wishbone* kopni datu apmaiņai ar perifērijas ierīcēm, kas ir kartētas galvenajā atmiņā. *Wishbone* kopne ir atvēr-

tā datu kopnes specifikācija [24], kuru piedāvā OpenCores kopiena (*http://www.opencores.org*). Šai kopnei ir pieejams gatavu perifērijas iekārtu klāsts.

5.4. Minix 3 operētājsistēma

5.4.1. Vēsture

Minix ir profesora Endrjū Tanenbauma (Andrew S. Tannenbaum) 1987. gadā radīta operētājsistēma, kas sākotnēji paredzēta pedagoģiskiem nolūkiem. No nelielas operētājsistēmas, kuras 1. versija paredzēta darbībai ar *Intel* 8088, 8086 un 80286 (vēlāk arī *SPARC*, *Motorola 68000*), tās 3. versija ir izaugusi par modernu operētājsistēmu, kas darbojas uz jaunākajiem 32-bitu „x86“ saimes procesoriem un izmanto simetrisko multiapstrādi. Lai gan jaunākais 3. versijas *Minix* ir zaudējis ne-„x86“ arhitektūras procesoru saderību, to ir plānots pielāgot darbībai ar visuresošajiem „ARM“ procesoriem [22].

Minix 3, salīdzinot ar tā pirmajām versijām, ir ieguvis fundamentālas izmaiņas dažādās sfērās. Autoraprāt būtiskākās ir:

- virtuālā atmiņa, izmantojot lapošanu (*paging*) ir nomainījusi segmentācijas atmiņas modeli;
- vairāku pavedienu virtuālā failu sistēma ir ieviesusi gan failu sistēmu abstrakciju, gan veiktspējas uzlabojumus;
- „NetBSD“ operētājsistēmas *C* bibliotēkas izmantošana piedāvā modernu POSIX saderīgu saskarni;
- operētājsistēma ir kompilējama ar 3 kompilatoriem — ACK, GCC un LLVM kombinācijā ar „Clang“;

Pētniecības tēmai, kuras ietvaros tiek attīstīts *Minix 3*, 2008. gadā Eiropas Zinātnes padome (*European Research Council*) piešķīra 2,5 miljonus eiro [22].

5.4.2. Versija

Darbā tiek apskatīta tā tapšanas laikā jaunākā operētājsistēmas versija „3.2.0“, kura ir identificējama ar pēdējās izmaiņas identifikatoru „9e56468d6cd29e516cfc7ee24d29c5a6b79fdf25“ publiskajā repozitorijā (*git.minix3.org*).

5.4.3. Struktūra

Minix 3 mikrokodols tiek papildināts ar lietotāja līmeņa daļām, kas nodrošina dažādas operētājsistēmas pamatfunkcijas un tiek sauktas par „serveriem“. Serveri, izmantojot *IPC*, komunicē savā starpā un veic kodola izsaukumus. Dziņi un serveri tiek veidoti tā, lai tie būtu atkārtoti palaižami kļūdainas darbības gadījumā. Papildus iespēja ir „live update“, kas nodrošina dziņu un serveru versijas darbības laikā ar nelieliem darbības pārtraukumiem.

5.4.4. Kodols

Kodols ir organizēts tā, ka pēc palaišanas brīža nav ne viena kodola pavediena, kuram būtu nepieciešams plānotājam (*scheduler*) atvēlēt laiku. Arī plānotājs ir realizēts servera veidā, bet kodolā ir tikai tā zemākā līmeņa daļas. Kodolam eksistē „pseido-pavedieni“:

- „asyncm“ nodrošina asinhrono ziņojumu piegādi;
- „idle“ atbild par sistēmas tukšgaitas stāvokli. Paredzēts enerģijas taupošo režīmu ieslēgšanai un izslēgšanai;
- „clock“ nodrošina taimera uzdevumus un komunicē ar taimera aparatūru;
- „system“ nodrošina sistēmas izsaukumu apstrādi;
- „kernel“ nodrošina izņēmumu un pārtraukumu apstrādi.

„Pseido-pavedieni“ nav īsti pavedieni, jo tiem nav atsevišķu steku, bet galvenokārt kalpo kā identifikatori.

5.4.5. Serveri

Serveri ir sadalāmi pēc to nozīmes — eksistē tādi, kas ir minimālās darbības nepieciešamais komplekts un tiek ielādēti atmiņā jau no sistēmas palaišanas programmas (*boot loader*) un ir tādi, kuru nozīme ir tikai papildinoša.

Minimālai darbībai nepieciešamie serveri:

- „rs“ jeb „Reincarnation Server“ ir pirmais serveris, ko palaiž kodols: tas nodrošina procesu palaišanu sistēmas sākumā un atjaunošanu avārijas gadījumā.
- „pm“ jeb „Process Manager“ nodrošina procesu pārvaldību;
- „ds“ jeb „Data Store“ nodrošina nelielu datu glabātuvi, kura var tikt izmantota serveru un dziņu stāvokļa saglabāšanai;
- „sched“ jeb „Scheduler“ nodrošina procesu plānošanu;
- „vfs“ jeb „Virtual File System“ nodrošina virtuālo failu sistēmu;
- „mfs“ jeb „Minix File System“ nodrošina *Minix* failu sistēmu;
- „vm“ jeb „Virtual Memory“ nodrošina virtuālās atmiņas pārvaldību;
- „pfs’ jeb „Pipe File System“ nodrošina programmkanālu (*pipe*) failu sistēmu un *UNIX* ligzdas (*sockets*);
- „init“ ir pārējo procesu sakne — tas nodrošina sistēmas palaišanos, startējot citus serverus, dziņus un lietotāja programmas.

5.4.6. Dziņi

Minix 3 dziņiem tiek piedāvāta vienveidota saskarne; „block“ un „character“ dziņu veidi ir pieejami. Sekojoši dziņi ir nepieciešami minimālai darbībai:

- „memory“ nodrošina tādas atmiņas iekārtas kā „/dev/null“ un RAM „diskus“;
- „log“ centralizē sistēmas diagnostikas paziņojumus;
- „tty“ ir termināla dzinis.

6. SOC IZVEIDE

6.1. SoC projektējums

Loģisko komponenšu izkārtojums un saistības redzamas 6.1. attēlā (ar bultām attēloti signālu virzieni). SoC ārējās iekārtas, kas ir ārpus FPGA ir GPIO slēdži un gaismas diodes, pāreja no UART uz USB, operatīvā atmiņa un *Ethernet* fiziskā līmeņa mikroshēma. FPGA konfigurācijā ietilpst perifērijas iekārtas (GPIO kontrolleris, UART kontrolleris, taimeris), atmiņas modulis ar *Ethernet* atkļūdošanas saskarni un *LatticeMico32* procesors.

Projektējot *SoC*, tika pievienoti sekojoši jauni moduļi, kurus *Lattice* rīki nepiedāvāja:

- sistēmas galvenais modulis, kas satur visus apakšmoduļus, tai skaitā *LatticeMico32* procesoru (6.1. attēlā nosaukts „FPGA“, atbilst 1. pielikumam);
- atmiņas modulis (6.1. attēlā — „Memory module“, atbilst 2. pielikumam);
- atmiņas kontrolleris (6.1. attēlā — „Memory Controller“, atbilst 3. pielikumam);
- *Ethernet* programmēšanas un atkļūdošanas modulis (6.1. attēlā — „Ethernet Debug Interface“, atbilst 4. pielikumam);
- atmiņas aizsardzības modulis (6.1. attēlā — „Memory Protection Unit“, atbilst 5. pielikumam);

  LatticeMico32 procesors ir pieslēgts FPGA iekšējam PLL, kas dod 50MHz frekvenci. Tādā veidā frekvence ir konfigurējama, mainot konstanti kodā.

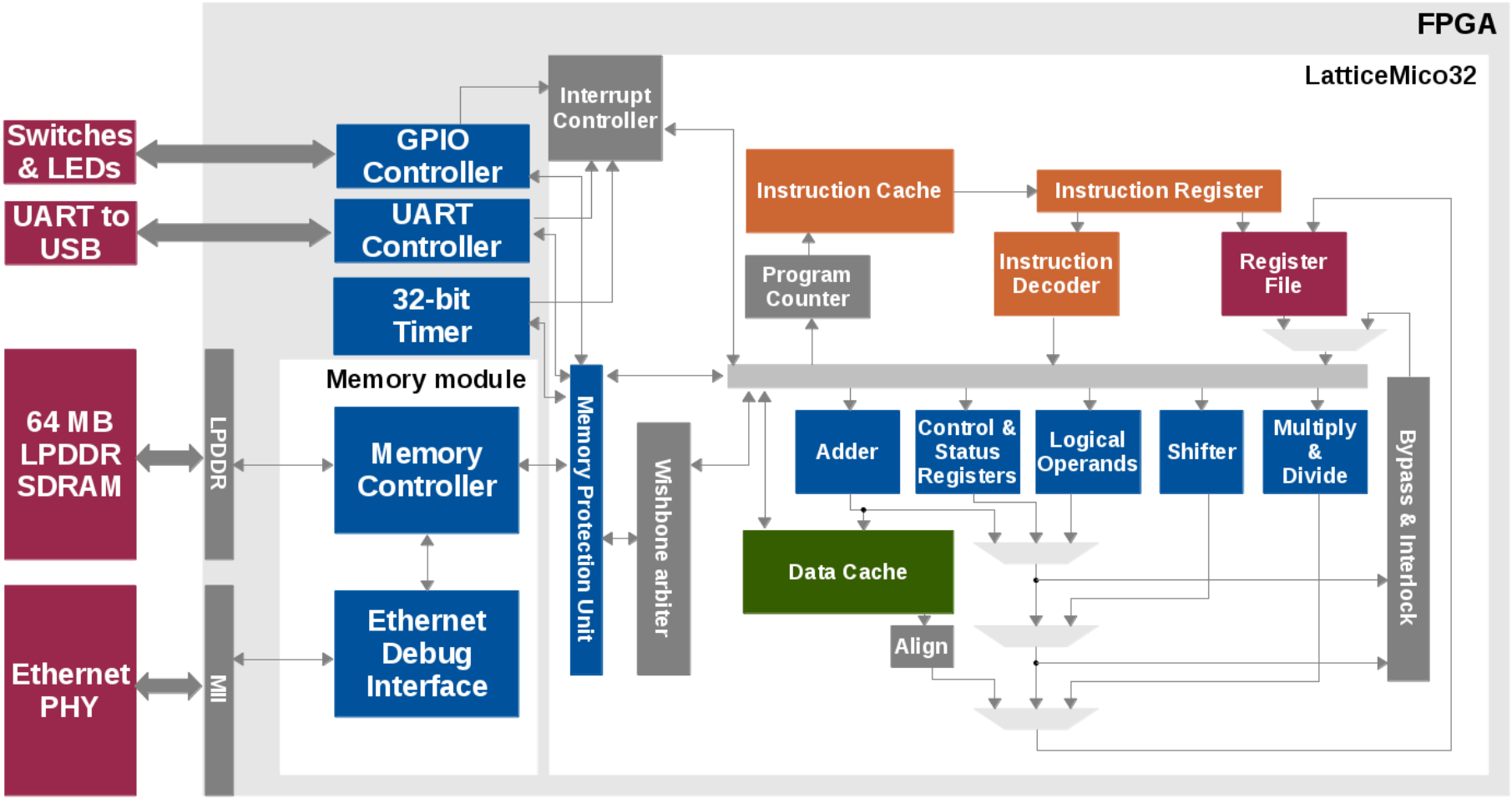
6.2. LatticeMico32 pielāgošana Spartan-6

6.2.1. Speciālie moduļi

Speciālie moduļi, kurus sauc arī par „black boxes“ ir tādi moduļi, kuru kods netiek atklāts izstrādātājam un daudzkārt ir FPGA ražotāja specifisks. Tādā veidā, piemēram, tipiski tiek piedāvāta iekšējā atmiņa jeb *Block RAM* vai vienkāršs rindu elements, vai taimeris.

Lai gan šādu moduļu izmantošana vienkāršo izstrādi un spēj nodrošināt vislabāko efektivitāti ar konkrētā ražotāja FPGA aparatūru, tā var sagādāt problēmas pārnešanā uz cita ražotāja aparatūru, kur atsevišķi moduļi var nebūt pieejami.

LatticeMico32 gadījumā vienīgais „black box“ veida modulis, kas tiek izmantots procesoram, ir iebūvētā atmiņa, lai nodrošinātu kešatmiņas. *Lattice* un *Xilinx Block RAM* moduļu saskarnes ir atšķirīgas. Problēmu var atrisināt, izmantojot *Verilog RAM* atmiņas konstrukciju, kuram *Xilinx* sintēzes rīks piemeklē attiecīgus *Block RAM* resursus. Citi „black box“ veida moduļi attiecās uz izvēles iespējām procesora perifērijas iekārtās (3 stāvokļu porti, FIFO rindas).



6.1. att. SoC projektējums

6.2.2. Atmiņas kontrolleris

6.2.2.1. Atmiņas kontrollera prasības

Atmiņas kontrollerim ir jānodrošina atmiņas mikroshēmas darbība un jāpiedāvā datu kopne, ko var izmantot procesors. *LatticeMico32* un *Microboard* gadījumā tā ir *Wishbone* kopne un *LPDDR SDRAM* veida atmiņa.

6.2.2.2. LPDDR SDRAM atmiņa

Sinhronā dinamiskā atmiņa tiek plaši izmantota gan datoriem, gan iegultajām sistēmām. Veidojot sistēmu ar šāda veida atmiņu, ir jāņem vērā tās latentums — rakstīšanas un lasīšanas operācijas tiek liktas rindā, bet izpildītas tiek ar aizkavi. Lai kopējo aizkavi mazinātu, var izmantot *burst* veida operācijas — noteikta skaita secīgu atmiņas adrešu lasīšanu vai rakstīšanu.

Pēc 6.3. attēla redzamās atmiņas kopnes darbības analīzes var redzēt, ka atmiņas pieprasījums tiek īstenots 11 taktīs, kas ir 0.22 μs. Ar šādu latentumu *burst* veida operācijas nodrošina secīgo darbību aptuveni desmitkārtēju paātrinājumu.

6.2.2.3. MCB elements

Ņemot vērā, ka atmiņas mikroshēmas kopnes darbība ir iespējama ar 200MHz frekvenci un ka tā ir signālu izplatīšanās laika ziņā ierobežota, *Spartan-6* tiek piedāvāti divi iebūvētie atmiņas kontrollera elementi jeb MCB.

MCB nodrošina nepieciešamo komunikāciju ar atmiņas mikroshēmu, ieskaitot atmiņas atsvaidzināšanas (*auto refresh*) komandas. MCB nav paredzēts tā izstrādātāja izmantošanai tiešā veidā, bet ar komplektā piedāvāto rīku „Memory Interface Generator“ (MIG), kas vienkāršo MCB izmantošanu.

Ar MIG var ģenerēt „Verilog“ vai „VHDL“ kodu, kas piedāvā dažāda platuma un skaita „lietotāja“ saskarnes un nodrošina atbilstošu iebūvētā atmiņas kontrollera darbību. MIG vienkāršo dažādu atmiņas mikroshēmu izmantošanu, jo piedāvā tām vienveidotu saskarni. Piedāvātās saskarne ir 3 FIFO rindas: operāciju rinda, nolasīto datu rinda, ierakstāmo datu rinda.

6.2.2.4. Pāreja uz Wishbone kopni

Tabulā 6.1. redzama procesora atmiņas saskarne, bet tabulā 6.2. ir parādīta MIG piedāvātā saskarne. Būtiska abu saskarņu nesaderība ir *burst* atmiņas rakstīšanas / lasīšanas garuma trūkums procesora piedāvātajā saskarnē. Lai gan šīs saskarnes specifikācijā ir atzīmēta iespēja norādīt *burst* garumu, *LatticeMico32* to nedara, bet signalizē *burst* beigas.

Tas ir iemesls, kādēļ šajā risinājumā *burst* veida pieprasījumi tiek apstrādāti kā parastie viena vārda pieprasījumi, zaudējot dažas mikrosekundes secīgu atmiņas darbību veikšanā. Šīs problēmas atrisināšanai ir divi risinājumi. Pašsaprotamākais, bet iespējams sarežģītākais, būtu risinājums, veicot izmaiņas procesorā, kas liktu tam *Wishbone* kopnē norādīt *burst* operāciju

garumus. Otrs risinājums ir iespējā pieņemt, ka katra atmiņas darbība ir *burst* veida un gadījumā, ja tā izrādās īsāka, norādīt atmiņas kontrollerim, ka pārējās darbības ir jāignorē (6.2. tabulā norādītais „c3_p0_wr_mask“ piedāvā maskēt rakstīšanas operācijas, bet liekas lasīšanas operācijas nevar būt kaitīgas).

6.1. tabula

Atmiņas saskarne ar procesoru

Kopne / signāls	**Virziens**	**Nozīme**
DDR_ADR_I[31:0]	Ieeja	Atmiņas adrese
DDR_DAT_I[31:0]	Ieeja	Ierakstāmais datu vārds
DDR_WE_I	Ieeja	Rakstīšana
DDR_CYC_I	Ieeja	Derīgs kopnes cikls
DDR_STB_I	Ieeja	Derīgs datu pārraides cikls
DDR_SEL_I[3:0]	Ieeja	Derīgo bitu maska (1 bits atbilst vienam baitam no ierakstāmā datu vārda)
DDR_CTI_I[2:0]	Ieeja	Cikla tipa identifikators
DDR_BTE_I[1:0]	Ieeja	*Burst* tipa paplašinājums
DDR_LOCK_I	Ieeja	Nepārtraucams kopnes cikls
DDR_DAT_O[31:0]	Izeja	Nolasītais datu vārds
DDR_ACK_O	Izeja	Apstiprinājums
DDR_ERR_O	Izeja	Kļūda
DDR_RTY_O	Izeja	Atkārtošanas pieprasījums

6.2. tabula

MIG piedāvātā saskarne ar atmiņu

Kopne / signāls	Virziens	Nozīme
c3_p0_cmd_clk	Izeja	Komandu rindas takts
c3_p0_cmd_en	Izeja	Komanda ir derīga
c3_p0_cmd_instr[2:0]	Izeja	Komanda
c3_p0_cmd_bl[5:0]	Izeja	*burst* garums
c3_p0_cmd_byte_addr[29:0]	Izeja	Atmiņas adrese
c3_p0_cmd_empty	Ieeja	Komandu rinda tukša
c3_p0_cmd_full	Ieeja	Komandu rinda Pilna
c3_p0_wr_clk	Izeja	Rakstīšanas rindas takts
c3_p0_wr_en	Izeja	Rakstīšana
c3_p0_wr_mask[3:0]	Izeja	Rakstāmā vārda derīgie baiti
c3_p0_wr_data[31:0]	Izeja	Rakstāmie vārdi
c3_p0_wr_full	Izeja	Rakstīšanas rinda pilna
c3_p0_wr_empty	Ieeja	Rakstīšanas rinda tukša
c3_p0_wr_count[6:0]	Ieeja	Rakstīšanas rindas skaits
c3_p0_wr_underrun	Ieeja	Mēģinājums ierakstīt vairāk par iedotajiem rindas elementiem
c3_p0_wr_error	Ieeja	Rakstīšanas rindas kļūda
c3_p0_rd_clk	Izeja	Lasīšanas rindas takts
c3_p0_rd_en	Izeja	Lasīšana
c3_p0_rd_data[31:0]	Ieeja	Nolasītie dati
c3_p0_rd_full	Ieeja	Lasīšanas rinda pilna
c3_p0_rd_empty	Ieeja	Lasīšanas rinda tukša
c3_p0_rd_count[6:0]	Ieeja	Lasīšanas rindas elementu skaits
c3_p0_rd_overflow	Ieeja	Lasīšanas rinda pārpildīta
c3_p0_rd_error	Ieeja	Lasīšanas rindas kļūda

6.3. LatticeMico32 lietotāja un kodola režīmi

6.3.1. Nepieciešamība

Lai nodrošinātu, ka lietotāja programmas, serveri un dziņi nevar negatīvi ietekmēt kodola un citu programmu darbību, ir nepieciešams ieviest režīmus ar dažādām privilēģijām.

„x86“ saimes arhitektūrām ir 4 līmeņu aizsardzība (*protection ring*). Sadalīt aizsardzību vairākos līmeņos var izrādīties izdevīgi, tomēr sākotnējs risinājums nedrīkst uzstādīt prasību pēc smalkas līmeņu detalizācijas, vienkāršai pārnešanai uz citām arhitektūrām, kā arī atstājot iespēju paplašinājumiem vēlāk.

Tiek piedāvāts ieviest divus režīmus — lietotāja un kodola. Lai procesors pārslēgtos uz kodola režīmu, ir jānotiek izņēmumam (vai sistēmas izsaukumam). Pārslēgšanās atpakaļ uz

lietotāja režīmu notiek ar „eret“ instrukciju.

6.3.2. LatticeMico32 priviliģētās instrukcijas

LatticeMico32 ir identificējamas šādas instrukcijas, kas var novest pie negatīva efekta kļūdainas programmas darbības rezultātā:

- Instrukcija „wcsr“, kas ļauj ierakstīt kontroles reģistros.
- Instrukcija „eret“, kas domāta priekš atgriešanās no pārtraukuma un pārraksta „Interrupt Enable“ reģistru.

Modifikācijas rezultātā lietotāja režīmā šīs instrukcijas tiek interpretētas kā „nop“ (*no operation*), kas nedara neko.

6.4. LatticeMico32 MPU

Procesoram ir nepieciešams MPU, lai no lietotāja procesiem (arī serveriem un dziņiem) aizsargātu a) citus lietotāja procesus, b) kodolu un c) atmiņā kartētās ievades / izvades ierīces.

Neatļautas lasīšanas operācijas nevar izraisīt sistēmas nestabilitāti, tāpēc aizsardzība tiek veikta rakstīšanai ar izņēmumu attiecībā uz ievades / izvades atmiņas apgabalu, kuru jāaizsargā arī no lasīšanas, jo atsevišķas ierīces reaģē arī uz lasīšanu (piemēram, *read and clear* veida reģistri). Atsevišķos pielietojumos ir svarīga arī privātuma aizsardzība (piemēram, aizsargātas paroles citā procesa adrešu telpā). Tam ir iespējams ieviest papildus karodziņu, kas norādītu, ka konkrēto apgabalu lasīt nedrīkst. Pieeja, ka atmiņas lasīšanu būtu īpaši jāaizliedz, vienkāršo informācijas apmaiņu starp procesiem.

Piedāvātais MPU neatšķiras no citām *LatticeMico32* perifērijas ierīcēm ar to, ka tas ir konfigurējams, izmantojot atmiņas adrešu telpu. Kā sekas ir tas, ka MPU vienmēr ir pieejams izmaiņām kodola režīmā un, ja kodols to ir attiecīgi nokonfigurējis, tad arī lietotāja režīmā.

Lai MPU sevi attaisnotu, tas nedrīkst būt aparatūras resursu prasīgs kā MMU (*page tables, Translation Lookaside Buffer*). MPU iekļauj atmiņas apgabalu adreses, kuriem tekošajam procesam ir ļauta piekļuve. Atmiņas apgabalu raksturo 32 bitu sākuma adrese un apgabala izmērs. Neatļautas piekļuves gadījumā MPU izraisa datu kopnes izņēmumu.

Ir redzams, ka piedāvātā MPU atmiņas aizsardzības detalizācija ir tieši proporcionāla MPU aparatūras resursiem. *Minix 3* (un citas operētājsistēmas) atmiņu organizē tā, ka tekošā pavediena steks aug tā datu apgabala virzienā. Aizsardzības nolūkos tiek atstāts neizmantots atmiņas fragments starp abiem apgabaliem gadījumam, kad stekam paredzētais atmiņas apgabals tiek pārkāpts. Tādā gadījumā nostrādā atmiņas aizsardzība — kodols tiek informēts par nekorektu darbību, process tiek izbeigts. Ir svarīgi, lai piedāvātais MPU nezaudētu šo īpašību.

Ņemot vērā, ka vienam procesam var būt vairāki pavedieni, ir nepieciešams atmiņas aizsardzību organizēt individuāli pavedieniem, nevis procesiem, lai pietiktu ar nelielu MPU atļauto atmiņas apgabalu skaitu. Tabulā 6.3. redzama „Minix 3“ izdalītā virtuālā atmiņa „vfs“ procesam „x86“ sistēmā, kurā skaidri redzami neizdalītie atmiņas fragmenti aizsardzībai.

6.3. tabula

Atmiņas izdalījums procesam „vfs“

Atmiņas apgabals	Izmērs	Nozīme
00000000-0003c000	240 kB	Programmas teksts
0003c000-00122000	920 kB	Datu apgabals
7fffe000-80000000	8 kB	1. pavediena steka apgabals
80001000-80003000	8 kB	2. pavediena steka apgabals
80004000-80006000	8 kB	3. pavediena steka apgabals
80007000-80009000	8 kB	4. pavediena steka apgabals
8000b000-8000c000	4 kB	5. pavediena steka apgabals
8000e000-8000f000	4 kB	6. pavediena steka apgabals
80011000-80012000	4 kB	7. pavediena steka apgabals
80014000-80015000	4 kB	8. pavediena steka apgabals
80017000-80018000	4 kB	9. pavediena steka apgabals
8001a000-8001b000	4 kB	10. pavediena steka apgabals
8001d000-8001e000	4 kB	11. pavediena steka apgabals

Lai gan apgabalu skaits ir sintezēšanas laikā konfigurējams, sākotnēji tiek piedāvāts izmantot 4 adrešu apgabalus, kas būtu pietiekami priekš 1) teksta, 2) datu, 3) steka, 4) ievades / izvades apgabaliem. Tam būtu nepieciešami vien $2 * 4 * 4 = 32$ baiti no aparatūras resursiem.

Šāda pieeja nepieļauj pieaugoša izmēra atmiņas apgabalus. Redzams, ka, palielinot MPU aizsargāto apgabalu skaitu, ir iespējams piedāvāt pavedienam papildus atmiņu pēc nepieciešamības.

6.5. Ethernet saskarne

6.5.1. Ethernet saskarnes pielietojums

Tipiski iegultās sistēmas programmas glabā zibatmiņā. Tomēr izstrādes vajadzībām zibatmiņa ir neparocīga ar savu nelielo darbības ātrumu. Tika izveidots *Ethernet* modulis, kas izmanto *Ethernet* saskarni programmēšanai un atkļūdošanai, piedāvājot ar to veikt operatīvās atmiņas lasīšanas un rakstīšanas operācijas.

6.5.2. Media Independent Interface

Ethernet modulis izmanto MII saskarni, kas ir pieejama *Ethernet* fiziskā līmeņa mikroshēmai (*PHY*), lai nodrošinātu 100 Mbit/s savienojumu. Lai gan tipiski izstrādes rīki piedāvā *Fast Ethernet* fizisko līmeni, MII izmantošana vienkāršo pārnešanu uz citu platformu.

6.5.3. Izveidotais protokols

Tika izveidots protokols ar sekojošām iespējām:

- procesora *reset* funkcija;
- procesora apstādināšana, darbības turpināšana;
- vārda (4 baitu) lasīšana / rakstīšana;
- 8 vārdu lasīšana / rakstīšana.

Izmantojot vairāku vārdu lasīšanu (pēdējā iespēja), iespējams nolasīt no atmiņas vai ierakstīt atmiņā failus ar šim nolūkam izveidoto programmu priekš datora (6. pielikums).

Saziņai ar datoru tiek izmantotas *User Datagram Protocol* jeb *UDP* apraides (*broadcast*) paketes.

Internet Protocol (*IP*) galvene netiek mainīta un tās kontrolsumma ir ierakstīta kodā kā konstante; *UDP* galvenes kontrolsumma netiek izmantota (tās vērtība ir 0), tādā veidā atstājot *Ethernet* kontrolsummu kā vienīgo, kas ir jāaprēķina FPGA. Ņemot vērā, ka *Ethernet* kontrolsumma ir paketes pēdējie baiti, tā tiek rēķināta paketes sūtīšanas brīdī.

Pakešu datu struktūra ir reprezentējama ar sekojošu *C* struktūru:

```
1  struct dbg_pkt {
2    uint8_t command;            /* write/read/run/stop etc */
3    uint8_t length;             /* length of data */
4    uint8_t padding[2];
5
6    uint32_t address;           /* memory address */
7    uint32_t data[8];           /* payload */
8  };
```

6.6. FPGA resursu izmantošana

Tika apskatītas dažādas SoC konfigurācijas, kuru resursu izmantošanas novērtējums apkopots tabulā 6.4..

Ņemot vērā, ka normālā darbībā atkļūdošanas modulis netiek izmantots, pirmā tabulas 6.4. rinda atspoguļo minimālās SoC prasības. Pārējās rindas iekļauj atkļūdošanas moduli un pieaugošā secībā kādu no aritmētiskajām darbībām. Visas konfigurācijas ir bez kešatmiņām. Kešatmiņu izmērs ir konfigurējams un galvenokārt atspoguļojas uz iebūvētās „Block RAM“ atmiņas izmantojumu.

6.4. tabula

SoC FPGA izmantotie resursi

Konfigurācija	Elementi	Izmantoti	Pieejami	Daļa
Bez atkļūdošanas moduļa, minimāls LatticeMico32	Slice Registers	2 410	11 440	21%
	Slice LUTs	2 913	5 720	50%
	DSP48A1s	0	16	0%
Minimāls LatticeMico32	Slice Registers	4 085	11 440	35%
	Slice LUTs	5 003	5 720	87%
	DSP48A1s	0	16	0%
LatticeMico32 ar *sign extension*	Slice Registers	4 086	11 440	35%
	Slice LUTs	4 983	5 720	87%
	DSP48A1s	0	16	0%
LatticeMico32 ar *sign extension*, dalītāju	Slice Registers	4 225	11 440	36%
	Slice LUTs	5 159	5 720	90%
	DSP48A1s	0	16	0%
LatticeMico32 ar *sign extension*, dalītāju, bīdītāju	Slice Registers	4 258	11 440	37%
	Slice LUTs	5 280	5 720	92%
	DSP48A1s	0	16	0%
LatticeMico32 ar *sign extension*, dalītāju, bīdītāju, reizinātāju	Slice Registers	4 326	11 440	37%
	Slice LUTs	5 153	5 720	90%
	DSP48A1s	3	16	18%

6.7. SoC testi

6.7.1. Nepieciešamība un metodika

Lai pārliecinātos par pareizu sistēmas darbību un noskaidrotu veiktspējas rādītājus, tika uzrakstītas nelielas *C* un *assembler* programmas. Laika mērīšana ir veikta ar oscilogrāfa palīdzību.

6.7.2. Atmiņas pārbaude

Sistēmas atmiņas darbības pārbaude tika veikta, izmantojot iespēju rakstīt un lasīt no atmiņas ar *Ethernet* saskarni. Ņemot vērā, ka šī saskarne tiek līdzvērtīgi multipleksēta ar procesoru, šāda veida pārbaudei jāapstiprina korektu darbību arī ar procesoru.

Tipisks atmiņas pārbaudes raksts tika izmantots pārbaudei: 0x5555, 0xAAAA, 0x0000, 0xFFFF. 64 MB fails ar šādu saturu tika ierakstīts un nolasīts no iekārtas, salīdzinot saturu.

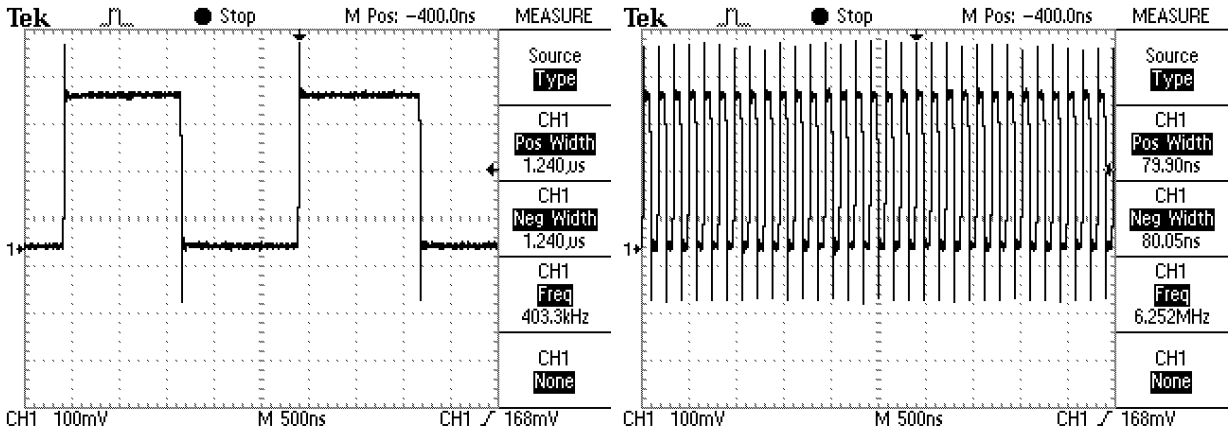
6.7.3. GPIO testa programma

Vienkārša GPIO stāvokļa maiņa ir primitīvs, bet efektīvs tests, lai noteiktu, ka procesors ir darboties spējīgs. Ar to var noteikt darbības ātrumu, kā arī pārliecināties par instrukciju kešatmiņas darbību. Ņemot vērā, ka programma ir vienkārša, var viegli pārbaudīt, vai kopne tiek izmantota pareizi, izmantojot *ChipScope* rīku.

Programmas kods:

```
1  start:
2      mvhi r1, 0x8000      /* Address of the GPIO device register */
3      mvhi r2, 0xFFFF      /* GPIO ON */
4
5  gpio_loop:
6      xnori r2, r2, 0      /* Invert */
7      sw (r1+0), r2        /* Store */
8      bi gpio_loop         /* Branch */
```

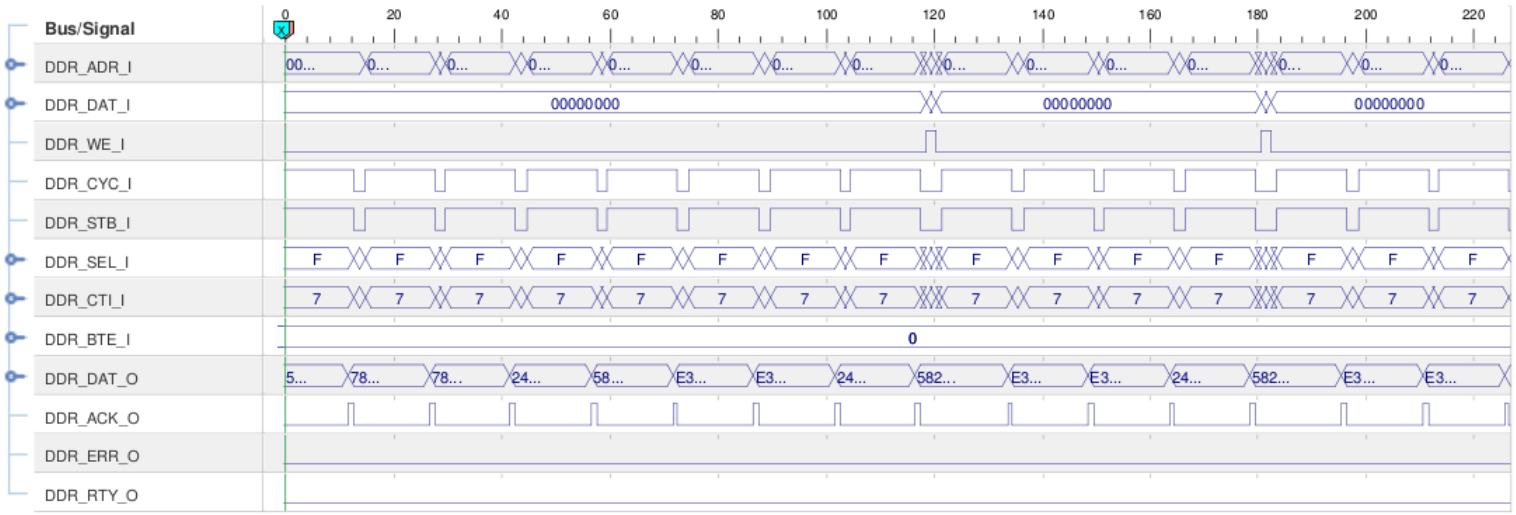
Attēlā 6.2. redzama programmas darbība konfigurācijā bez kešatmiņas (a) un konfigurācijā ar kešatmiņu (b). Tiek konstatēts, ka GPIO stāvokļu maiņas frekvence bez kešatmiņas ir 403kHz, bet ar kešatmiņu — 6,25MHz; starpība skaidri parāda kešatmiņas darbību.



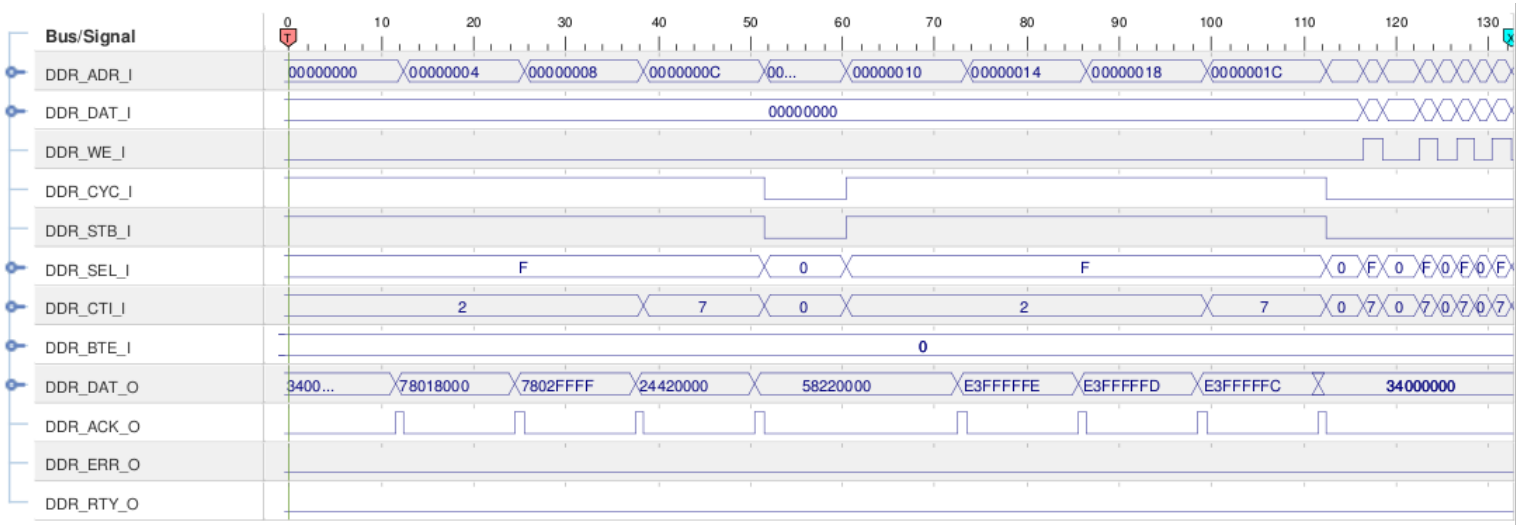
(a) (b)

6.2. att. **GPIO testa programmas oscilogrāfa uzņēmumi**:
(a) bez kešatmiņas, (b) ar kešatmiņu

Attēlā 6.3. redzams *ChipScope* loģikas analizatora uzņēmums datu kopnei šīs programmas palaišanas brīdī. Gadījumā (a) bez kešatmiņas redzams, ka programma tiek lasīta visu laiku, bet gadījumā (b) ar kešatmiņu visa programma (un 3 instrukcijas vairāk) tiek ielasīta kešatmiņā ar divām *burst* veida lasīšanas operācijām, un tālāk darbība norit tikai no kešatmiņas, piedāvājot maksimālo procesora ātrumu.



(a)



(b)

6.3. att. **GPIO testa programmas *ChipScope* uzņēmumi:**
(a) bez kešatmiņas, (b) ar kešatmiņu

6.7.4. Testi dažādām SoC konfigurācijām

Tika izveidota sekojoša *C* programma, lai pārbaudītu programmu ātrdarbību dažādās *LatticeMico32* konfigurācijās.

```
1  int main(int argc, char **argv)
2  {
3      register int  gpio_state = 1;
4      register int  i, j;
5      int       a;
6      unsigned u;
7      char      b;
8
9      for (j = 0; j < 0xFFFFFFFF; j++)
10     {
11         a = -77;
12
13         gpio_state = !gpio_state;
14         gpio_set(gpio_state);
15
16         for (i = 0; i < 20; i++)
17         {
18             b = a;                    /* Sign extension */
19             a *= a;                   /* Multiplication */
20             u = a / 13;               /* Division */
21             a = u << 10;              /* Barrel shift */
22         }
23     }
24     return a;
25 }
```

Kompilators konfigurācijām, kur iztrūkst aparatūras, izmanto programmatūras risinājumu — bibliotēku, kas gan palielina izpildāmā faila izmēru, gan arī samazina veiktspēju. Iegūtie rezultāti apkopoti tabulā 6.5.

6.5. tabula

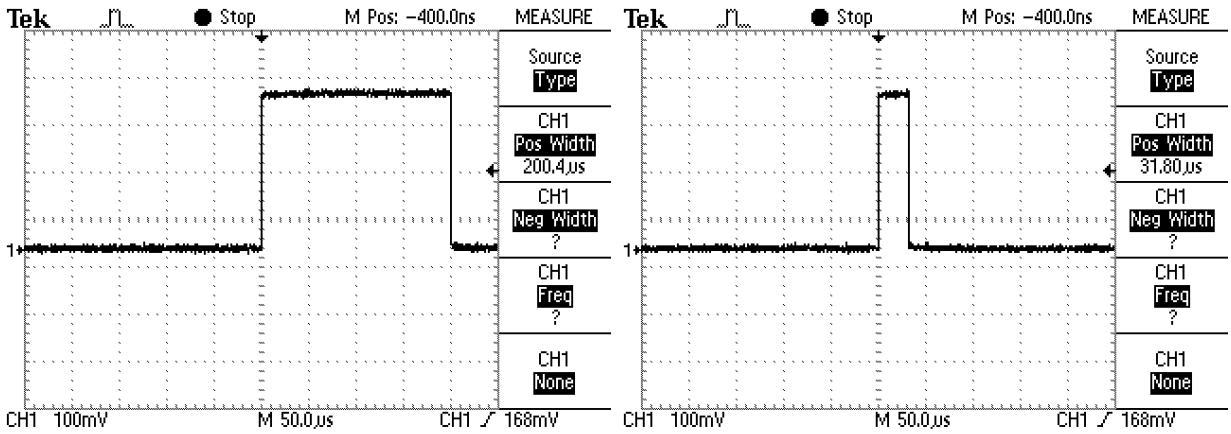
Konfigurāciju veiktspējas salīdzinājumi

Konfigurācija	Cikla laiks (ms)	Programmas izmērs (baiti)
Minimāls LatticeMico32	2,274	5976
LatticeMico32 ar *sign extension*	2,274	5976
LatticeMico32 ar *sign extension*, dalītāju	2,110	5424
LatticeMico32 ar *sign extension*, dalītāju, bīdītāju	1,934	5264
LatticeMico32 ar *sign extension*, dalītāju, bīdītāju, reizinātāju	0,104	5216

6.7.5. Konteksta pārslēgšana

Ātra konteksta maiņa ir būtisks mikrokodola operētājsistēmas veiktspējas ietekmējošais faktors. Tika izveidota testa programma, kas izmanto 50Hz taimerpārtraukumus, lai tajos pārslēgtu kontekstu no viena pavediena konteksta uz pārtraukumu apstrādes pavediena kontekstu, tad atpakaļ uz cita pavediena kontekstu. Pārtraukuma ieejas punktā tika GPIO izvada stāvoklis nomainīts uz loģisko „1“, bet atgriežoties no pārtraukuma — uz loģisko „0“.

Attēlā 6.4. redzams oscilogrāfa uzņēmums divu konteksta pārslēgšanas (1. no viena pavediena uz izņēmuma apstrādi, 2. atpakaļ uz citu pavedienu) ilgumam abiem gadījumiem — 201 μs bez kešatmiņas (a) un 32 μs ar kešatmiņu (b).



(a) (b)

6.4. att. **Konteksta pārslēgšanas testa programmas oscilogrāfa uzņēmumi:** (a) bez kešatmiņas, (b) ar kešatmiņu

7. MINIX 3 PĀRNEŠANA

7.1. Programmatūras izstrādes rīki un metodes

GCC ir daudzām arhitektūrām pielāgots kompilācijas rīku komplekts, kura 3. versiju *Lattice Semiconductor* ir pielāgojuši arī savai *LatticeMico32* arhitektūrai. Operētājsistēmas Real-Time Executive for Multiprocessor System izstrādātāji ir attīstījuši GCC versiju 4.5.3, kuru autors atzīst par labu esam *LatticeMico32* koda ģenerēšanā [20].

Tika apsvērts izmantot lietot LLVM un „Clang“ rīkus, kuru pielāgošanu *LatticeMico32* ir veikta „Milkymist“ projekta vajadzībām [21]. Tomēr autors šo pielāgoto versiju neatzīst par pietiekami nobriedušu šai arhitektūrai, jo atsevišķu programmu kompilācija beidzas ar kompilatora avāriju.

Veiksmīga *LatticeMico32* programmu darbība tika iegūta ar QEMU procesora emulatoru. Tas atļauj izmantot arī „gdb“ atkļūdošanas rīku [7].

7.2. Kompilācija

Izstrādājot iegulto sistēmu, tās kompilācija tipiski tiek veikta ar datoru. Ja datora procesora arhitektūra nesakrīt ar iegultās sistēmas izmantoto procesora arhitektūru, ir nepieciešams kompilators, kas prot kompilēt kodu šim procesoram. Šāda veida kompilāciju sauc par „cross-compilation“.

Ar *Minix 3* jaunāko kompilācijas sistēmu (*build system*) kompilēšana tiek veikta tikai pašā *Minix 3* sistēmā. Lai gan tas nav būtisks sarežģījums, izstrādājot ar „x86“ saimes arhitektūras datoru (piemēram, darbinot *Minix 3* virtuālajā mašīnā), iegulto sistēmu izstrādē *cross-compilation* tiek veikta vienmēr.

Minix 3 piedāvātā kompilācijas sistēma izmanto „BSD“ veida „Makefile“ failus priekš „bmake“ programmas. Tāpēc, lai kompilāciju varētu veikt, izmantojot „Linux“ šie faili tika pārrakstīti „GNU Make“ sintaksē, ļaujot sistēmas daļu kompilēt atsevišķi. No šiem failiem var izveidot kaskādi, kas spēj kompilēt sistēmu vienā piegājienā.

Sekmīgu kompilāciju izdevās nodrošināt šādām sistēmas daļām:

- kodolam;
- serverim „sched“;
- taimeru sistēmas bibliotēkai „libtimers“;
- izpildīšanas sistēmas bibliotēkai „libexec“;
- serveru un kodola sistēmas bibliotēkām „libminc“, „libminlib“, „libsys“.

7.3. Arhitektūras atkarība

Par arhitektūras atkarīgām operētājsistēmas daļām, pirmkārt, ir atzīstamas visas tās, kas ir rakstītas mašīnkodā. Otrkārt, tās ir tādas, kas ir atsevišķi izdalītas, jo par tādām tiek atzītas un nosauktas izstrādes brīdī. Treškārt, tās ir tādas, kas parādās netiešā veidā: tās ir kopējā kodā, bet par to arhitektūras atkarību ir jāizsecina izstrādātājam, kas veic operētājsistēmas pārnešanu.

Minix 3 gan bibliotēkām, gan kodolam mapē „arch/i386“ ir atsevišķi izdalītas *assembler* un *C* valodās rakstītās „x86“ saimes arhitektūru atkarīgās daļas. Atsevišķi neizdalīts kods ir peldošā komata aritmētikas bloka jeb FPU saistīts kods un virtuālās atmiņas pārvaldības kods.

7.4. Konteksta pārslēgšana

Ņemot vērā mikrokodola operētājsistēmu nepieciešamību pēc biežas konteksta pārslēgšanās, tās pārslēgšanās laiks ir noteicošais kopējā sistēmas ātrdarbībā. *LatticeMico32* ir 32 vispārēja pielietojuma reģistri, kurus konteksta pārslēgšanas brīdī ir jāsaglabā un jāatjauno. Procesors atmiņā var saglabāt vienu reģistru ar vienu instrukciju:

```
1     sw      (sp+ 0*4), r1
2     sw      (sp+ 1*4), r2
3     sw      (sp+ 2*4), r3
```

⋮

```
31    sw      (sp+30*4), r31
```

Tāda saglabāšana prasa 2 darbības ar atmiņu katram reģistram — instrukcijas nolasīšanai un reģistra vērtības saglabāšanai.

Procesora koda pieejamība ļauj veikt pielāgojumus situācijas uzlabošanai. Sekojošas optimizācijas ir iespējamas un ir pētīšanas vērts temats pats par sevi:

- Vienkāršākā optimizācija būtu „x86“ saimes arhitektūru instrukciju „pusha“, „popa“ līdzīgas ieviešana — viena instrukcija, kas saglabā visus reģistrus.
- Pārslēdzoties uz kodola režīmu var saglabāt tikai tos reģistrus, kas tiks izmantoti kodola kodā. Kodola koda izmantoto reģistru skaitu var ierobežot, meklējot optimālu variantu ātrdarbībai.
- Var ieviest reģistru kopijas — reģistru bankas, lai pārslēgšanās brīdī reģistrus būtu jāglabā atmiņā tikai tad, kad banku jau ir aizņēmis cits pavediens [16]. Palielinot banku skaitu, var iegūt pat vienas instrukcijas konteksta saglabāšanu.

Papildus reģistriem, ir jānomaina MPU konteksts. Ņemot vērā, ka MPU ir perifērijas ierīce, kuras saturs neglabājas galvenajā atmiņā, tā nav drauds ātrdarbības zudumam.

7.5. Pozīcijas neatkarīgs kods

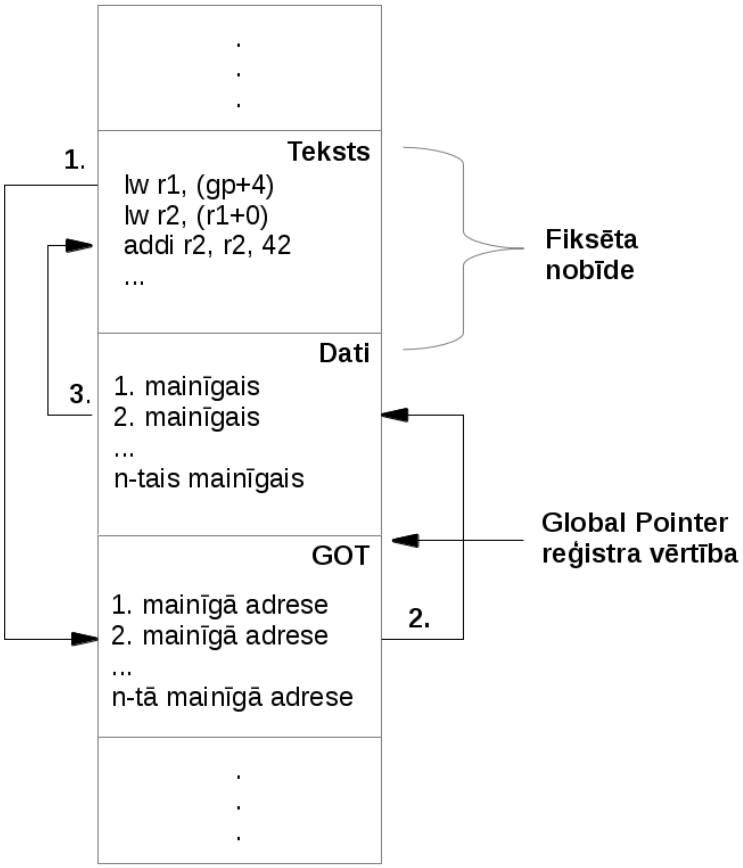
Lai dažādi procesi varētu sadalīt atmiņas telpu, sistēmā, kurā nav MMU, ir vairākas metodes. Vienkāršākā metode ir kompilācijas laikā norādītas programmas darbības adreses. Gadījumiem, kad ir jālieto vairāki programmas eksemplāri un jādala nelieli atmiņas resursi, šī metode nav derīga. Tā netiek plaši izmantota arī tādēļ, ka apgrūtina izstrādi, atstājot atmiņas pārvaldīšanu programmētāja ziņā.

Otra metode ir pozīcijas neatkarīga koda jeb *Position-Independent Code* (PIC) izmantošana. PIC ir tāds kods, kas ir spējīgs darboties dažādās atmiņas adresēs. Parasti tas nozīmē, ka kodā tiek izmantota relatīvā nevis absolūtā adresēšana. Lai iegūtu *PIC* kodu, ir nepieciešams kompilators, kas tādu prot ģenerēt un aparatūras atbalsts. GCC karodziņš „-fpic“ nodrošina šāda koda ģenerēšanu. *LatticeMico32* zarošanās instrukcija var strādāt ar relatīvām adresēm, izmantojot arī negatīvas adreses.

Papildus relatīvai adresēšanai tiek izmantota globālā nobīžu tabula jeb GOT. Programmas tekstā tiek izmantotas relatīvas (bet fiksētas) atsauces uz šo tabulu, kas atrodas programmas datu sekcijā, kad ir nepieciešams vērsties pie datiem. Pēc GOT tiek atrastas šo datu īstā atrašanās vieta. Ņemot vērā, ka GOT tabulā tiek rakstītas absolūtās adreses, programmas ielādētājam šo tabulu ir jāizlabo (tipiskā gadījumā tā ir konstantes pieskaitīšanas vai atņemšana katram tabulas ierakstam).

Relatīvo adresēšanu asistē *LatticeMico32* „Global Pointer“ reģistrs, kas ir viens no 32 vispārēja pielietojuma reģistriem, kuru var izmantot šādam nolūkam. Šo reģistru izmanto visām instrukcijām, kurām nepieciešams vērsties pie programmas datiem. Šajā reģistrā programmas darbības laikā tiek glabāta GOT adrese.

Attēlā 7.1. ir parādīta PIC izmantošana programmai, kas no datiem paņem globālo mainīgā vērtību un tai pieskaita 42. Tas notiek šādā secībā:



7.1. att. **Pozīcijas neatkarīga koda izmantošanas piemērs**

1. tiek ielādēta mainīgā absolūtā adrese;
2. pēc adreses tiek ielādēta mainīgā vērtība;
3. vērtība tiek izmantota saskaitīšanai.

Tomēr šādam kodam ir savi ierobežojumi: tikai daļa no 32 bitu adresēm ir iekodējamas 32 bitu instrukcijās, kā arī fakts, ka kompilators pieņem, ka datu segments ir fiksētā attālumā no teksta segmenta un iekļauj „Global Pointer“ relatīvus funkciju izsaukumus. Tāpēc, lai darbinātu vairākas programmas, kas var izmantot vienu un to pašu teksta segmentu, bet savu datu segmentu, ir nepieciešams glabāt arī absolūtās funkciju adreses. FDPIC formāts šādu iespēju nodrošina, piedāvājot glabāt funkciju deskriptorus pirms GOT, kuriem papildus tiek norādīts arī funkcijā izmantojamā „Global Pointer“ vērtība, tādā veidā ļaujot izmantot citu datu segmentu.

Minix 3 pielāgošanas vajadzībām, serveriem, kas tiek darbināti sistēmas sākumā pirms „init“ servera, var izmantot kompilācijas laikā norādītas programmas darbības adreses. Visām citām programmām, kas tiek sāktas no „init“ ir jāizmanto aprakstītais FDPIC formāts.

7.6. Piekļuve perifērijas iekārtām

Piekļuvei perifērijas iekārtām „x86“ saimes arhitektūras izmanto ne tikai galvenajā atmiņā kartētas iekārtas, bet tai ir arī papildus ievades / izvades adrešu telpa, kurai var piekļūt izmantojot speciālas priviliģētās instrukcijas. Ņemot vērā, ka *LatticeMico32* šādas adrešu telpas nav, tā var tikt radīta mākslīgi saderības vajadzībām. Tomēr visātrākais risinājums ir tieša piekļuve: arī nepriviliģēti *Minix 3* dziņi var izmantot atmiņā kartētās iekārtas ar attiecīgu MPU konfigurāciju.

REZULTĀTI

Resursu ziņā nelielam FPGA tika uztaisīta sistēma mikroshēmā, kas iekļauj 32 bitu arhitektūras procesoru un kuras minimālā konfigurācija atstāj aptuveni pusi no FPGA resursiem brīvus.

Pēc 6.2. (a) un (b) attēliem var spriest, ka atmiņas operāciju darbību ātrumu ciklos var palielināt līdz 15,5 reizēm, izmantojot kešatmiņu. Spriežot pēc tabulās 6.4. un 6.5. apkopotās informācijas, aritmētisko operāciju ātrdarbību var palielināt līdz pat 22 reizēm uz FPGA resursu rēķina, izmantojot FPGA iebūvētos reizinātājus.

Tika apskatīta *Minix 3* mikrokodola operētājsistēma, un ar to saistītās nepieciešamības attiecībā uz aparatūru tika īstenotas. Tika novērtēts mikrokodola operētājsistēmām kritisks parametrs — konteksta pārslēgšanas laiks, kas labākajā gadījumā ir 32 μs.

Darbā tika apzinātas *Minix 3* arhitektūras atkarīgās daļas un izveidoti „Makefile“ faili, kas nepieciešami *Minix 3* kompilācijai priekš *LatticeMico32* procesora. Tika iegūta daļēja kodola un tā bibliotēkas darbība, ļaujot sistēmai veikt statusa un atkļūdošanas izdrukas, kā arī palaist serveri.

SECINĀJUMI

Darbā tika parādīts, ka atvērtā koda procesors *LatticeMico32* ir pielāgojams un elastīgs, pārnesot to uz konkurējoša ražotāja FPGA aparatūru, pievienojot tam iepriekš nebijušas iespējas un apskatot dažādas tā konfigurācijas.

Lai gan pielāgojama aparatūra programmatūras pārnešanu vienkāršo, vēl svarīgāka ir programmatūras struktūra attiecībā uz pārnešanas iespējām. Jaunākās *Minix 3* versijas dažādu arhitektūru saderības trūkums liecina, ka pārnešana uz tām prasa netriviālu operētājsistēmas restrukturizāciju.

DARBS NĀKOTNĒ

Darbs ir jāturpina *Minix 3* pārnešanā. Ir jāpielāgo visi sistēmas izsaukumi, kas ir nepieciešami programmu darbībai, kā arī jāpielāgo minimālo serveru un dziņu komplektu darbībai ar izveidoto SoC. Sistēmas bibliotēkām ir jāpārraksta viss *assembler* valodā rakstītais kods. Ir nepieciešamas fundamentālas izmaiņas *Minix 3* atmiņas pārvaldībā, lai tās izmantotu aprakstītās FDPIC veida programmas un minimizētu tādus NOMMU sistēmu draudus kā atmiņas fragmentāciju.

LITERATŪRAS SARAKSTS

[1] **Aeroflex Gaisler AB**. Licensing FAQ. Tiešsaiste. Pieejams (21.05.2012): http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=177&Itemid=102.

[2] **Aeste Works**. AEMB Core. Tiešsaiste. Pieejams (21.05.2012): http://www.aeste.my/aemb.

[3] **Analog Devices**. Blackfin Processor Architecture Overview. Tiešsaiste. Pieejams (21.05.2012): http://www.analog.com/en/content/blackfin_architecture/fca.html.

[4] **Arshak, K. and Jafer, E. and Ibala, C**. Testing FPGA based digital system using XILINX ChipScope logic analyzer. In *Electronics Technology, 2006. ISSE'06. 29th International Spring Seminar on*, pages 355–360. IEEE, 2006.

[5] **Avnet, Inc**. Avnet Product Brief: Xilinx® Spartan®-6 FPGA LX9 MicroBoard. Tiešsaiste. Pieejams (21.05.2012): http://www.em.avnet.com/en-us/design/drc/Documents/Xilinx/xlx-s6-lx9-microboard-pb102611.pdf.

[6] **Brown, S. and Rose, J**. FPGA and CPLD architectures: A tutorial. *Design & Test of Computers, IEEE*, 13(2):42–57, 1996.

[7] **Gatliff, B**. Embedding with GNU: the GDB remote serial protocol. *Embedded Systems Programming*, 12:108–113, 1999.

[8] **Gu, W. and Kalbarczyk, Z. and Iyer, K. and Yang, Z. and others**. Characterization of linux kernel behavior under errors. In *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, pages 459–468. IEEE, 2003.

[9] **Habibi, A. and Tahar, S**. A survey on system-on-a-chip design languages. In *System-on-Chip for Real-Time Applications, 2003. Proceedings. The 3rd IEEE International Workshop on*, pages 212–215. IEEE, 2003.

[10] **Herder, J. and Bos, H. and Gras, B. and Homburg, P. and Tanenbaum, A**. Reorganizing UNIX for reliability. *Advances in Computer Systems Architecture*, pages 81–94, 2006.

[11] **Hildebrand, D**. An architectural overview of QNX. In *Proceedings of the USENIX Workshop of Micro-Kernels and Other Kernel Architectures*. 1992.

[12] **Jaeger, T**. Enforcement of the GNU GPL in Germany and Europe. *JIPITEC: Journal of Intellectual Property, Information Technology and E-Commerce Law*, 1, 2010.

[13] **Lattice Semiconductors**. Lattice Diamond Design Software. Tiešsaiste. Pieejams (21.05.2012): http://www.latticesemi.com/products/designsoftware/diamond/index.cfm.

[14] **Lattice Semiconductors**. LatticeMico32 Processor Reference Manual. Tiešsaiste. Pieejams (21.05.2012): http://www.latticesemi.com/documents/doc20890x45.pdf.

[15] **Lhotka, L.** History of Unix. *Sborník příspěvků*, page 23.

[16] **Liedtke, J.** Lazy Context Switching Algorithms for Sparc-like Processors. *Arbeitspapiereder GMDNo*, 776, 1993.

[17] **Lu, Z. and Zhang, X. and Sun, C.** An Embedded System with uClinux based on FPGA. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 2, pages 691–694. IEEE, 2008.

[18] **Lyonel Barthe.** SecretBlaze. Tiešsaiste. Pieejams (21.05.2012): `http://janela.lirmm.fr/~barthe/index.php/page/secretblaze.html`.

[19] **Mattsson, D. and Christensson, M.** Evaluation of synthesizable CPU cores. *Master's thesis, Department of Computer Engineering, Chalmers University of Technology*, 2004.

[20] **Milkymist project.** Building the RTEMS toolset on Debian. Tiešsaiste. Pieejams (21.05.2012): `http://milkymist.org/wiki/index.php?title=Building_the_RTEMS_toolset_on_Debian`.

[21] **Milkymist project.** Milkymist Public Repositories. Tiešsaiste. Pieejams (21.05.2012): `https://github.com/milkymist`.

[22] **Minix Community.** Minix 3 News. Tiešsaiste. Pieejams (21.05.2012): `http://www.minix3.org/news/`.

[23] **Open Kernel Labs.** Company Facts. Tiešsaiste. Pieejams (21.05.2012): `http://www.ok-labs.com/_assets/download_library/OK_Labs_Company_Datasheet.pdf`.

[24] **Opencores, S.** Wishbone system-on-chip (soc) interconnection architecture for portable ip cores. Technical report, Technical report, Opencores, 2002.

[25] **QNX Software Systems.** QNX Licensing. Tiešsaiste. Pieejams (21.05.2012): `http://www.qnx.com/legal/licensing/`.

[26] **Tong, J.G. and Anderson, I.D.L. and Khalid, M.A.S.** Soft-core processors for embedded systems. In *Microelectronics, 2006. ICM'06. International Conference on*, pages 170–173. IEEE, 2006.

[27] **Xilinx, Inc.** Spartan-6 Family Overview. Tiešsaiste. Pieejams (21.05.2012): `http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`.

PIELIKUMS

1. pielikums: „Galvenais modulis“

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:        Ansis Liepkalns
5  //
6  // Create Date:     19:09:46 04/02/2012
7  // Design Name:
8  // Module Name:     top
9  // Project Name:    Minix 3 on LM32
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module top(
22             input          CLOCK_Y3,
23             input          USER_RESET,
24             input [3:0]    GPIO_DIP,
25             output [3:0]   GPIO_LED,
26             output [7:0]   PMOD1,
27
28             input          USB_RS232_RXD,
29             output         USB_RS232_TXD,
30
31             input          ETH_COL,
32             input          ETH_CRS,
33             output         ETH_MDC,
34             output         ETH_MDIO,
35             input          ETH_RX_CLK,
36             input          ETH_RX_DV,
37             input [4:0]    ETH_RX_D,
38             input          ETH_TX_CLK,
39             output         ETH_TX_EN,
40             output [3:0]   ETH_TX_D,
41             output         ETH_RESET_n,
42
43             inout [15:0]   LPDDR_DQ,
44             output [12:0]  LPDDR_A,
45             output [1:0]   LPDDR_BA,
46             output         LPDDR_RAS_n,
47             output         LPDDR_CAS_n,
48             output         LPDDR_WE_n,
49             output         LPDDR_CKE,
50             output         LPDDR_CK_P,
51             output         LPDDR_CK_N,
52             inout          LPDDR_LDQS,
53             inout          LPDDR_UDQS,
54             inout          LPDDR_UDM,
55             inout          LPDDR_LDM,
56             inout          LPDDR_RZQ
57     );
```

```
58
59
60      wire [31:0]             DDR_ADR_I;
61      wire [32-1:0]           DDR_DAT_I;
62      wire                    DDR_WE_I;
63      wire                    DDR_CYC_I;
64      wire                    DDR_STB_I;
65      wire [32/8-1:0]         DDR_SEL_I;
66      wire [2:0]              DDR_CTI_I;
67      wire [1:0]              DDR_BTE_I;
68      wire                    DDR_LOCK_I;
69      wire [32-1:0]           DDR_DAT_O;
70      wire                    DDR_ACK_O;
71      wire                    DDR_ERR_O;
72      wire                    DDR_RTY_O;
73
74
75      wire                    sys_clk;
76      wire                    do_cpu_reset;
77      wire                    dbg_cpu_reset;
78
79      mem mem_i (
80                  .MEM_CLK_I      (CLOCK_Y3),
81                  .SYS_CLK_O      (sys_clk),
82                  .RST_I          (0),
83                  .CPU_RESET      (dbg_cpu_reset),
84
85                  // Ethernet MII
86                  .ETH_COL        (ETH_COL),
87                  .ETH_CRS        (ETH_CRS),
88                  .ETH_MDC        (ETH_MDC),
89                  .ETH_MDIO       (ETH_MDIO),
90                  .ETH_RX_CLK     (ETH_RX_CLK),
91                  .ETH_RX_DV      (ETH_RX_DV),
92                  .ETH_RX_D       (ETH_RX_D),
93                  .ETH_TX_CLK     (ETH_TX_CLK),
94                  .ETH_TX_EN      (ETH_TX_EN),
95                  .ETH_TX_D       (ETH_TX_D),
96                  .ETH_RESET_n    (ETH_RESET_n),
97
98                  // LPDDR interface
99                  .LPDDR_DQ       (LPDDR_DQ),
100                 .LPDDR_A        (LPDDR_A),
101                 .LPDDR_BA       (LPDDR_BA),
102                 .LPDDR_RAS_n    (LPDDR_RAS_n),
103                 .LPDDR_CAS_n    (LPDDR_CAS_n),
104                 .LPDDR_WE_n     (LPDDR_WE_n),
105                 .LPDDR_CKE      (LPDDR_CKE),
106                 .LPDDR_CK_P     (LPDDR_CK_P),
107                 .LPDDR_CK_N     (LPDDR_CK_N),
108                 .LPDDR_LDQS     (LPDDR_LDQS),
109                 .LPDDR_UDQS     (LPDDR_UDQS),
110                 .LPDDR_UDM      (LPDDR_UDM),
111                 .LPDDR_LDM      (LPDDR_LDM),
112                 .LPDDR_RZQ      (LPDDR_RZQ),
113
114                 // Wishbone interface
115                 .DDR_ADR_I      (DDR_ADR_I),
116                 .DDR_DAT_I      (DDR_DAT_I),
117                 .DDR_WE_I       (DDR_WE_I),
118                 .DDR_CYC_I      (DDR_CYC_I),
119                 .DDR_STB_I      (DDR_STB_I),
```

```
120                 .DDR_SEL_I        (DDR_SEL_I),
121                 .DDR_CTI_I        (DDR_CTI_I),
122                 .DDR_BTE_I        (DDR_BTE_I),
123                 .DDR_LOCK_I       (DDR_LOCK_I),
124                 .DDR_DAT_O        (DDR_DAT_O),
125                 .DDR_ACK_O        (DDR_ACK_O),
126                 .DDR_ERR_O        (DDR_ERR_O),
127                 .DDR_RTY_O        (DDR_RTY_O)
128                 );
129
130     reg [3:0]             do_cpu_reset_cntr = 'hF;
131     assign do_cpu_reset = do_cpu_reset_cntr > 0 ? 1 : 0;
132
133     always @(negedge sys_clk)
134       begin
135          if (USER_RESET || dbg_cpu_reset)
136            begin
137               do_cpu_reset_cntr <= 'hF;
138            end
139          else
140            begin
141               if (do_cpu_reset_cntr > 0)
142                 do_cpu_reset_cntr <= do_cpu_reset_cntr - 1;
143            end
144       end
145
146     // Debug outputs
147     wire exception_enter;
148     wire exception_leave;
149     wire privileged_mode;
150     assign PMOD1[4] = exception_enter | exception_leave;
151     assign PMOD1[5] = exception_enter;
152     assign PMOD1[6] = exception_leave;
153     assign PMOD1[7] = privileged_mode;
154
155     mico32 mico32_u (
156                      .clk_i(sys_clk),
157                      .reset_n(!do_cpu_reset),
158                      .privileged_mode(privileged_mode),
159                      .exception_enter(exception_enter),
160                      .exception_leave(exception_leave)
161                      , .gpioPIO_BOTH_IN(GPIO_DIP) // [4-1:0]
162                      , .gpioPIO_BOTH_OUT({PMOD1[3:0], GPIO_LED[4-1:0]}) // [4-1:0]
163                      , .uartSIN(USB_RS232_RXD) //
164                      , .uartSOUT(USB_RS232_TXD), //
165
166                      .RAM_ADR_I        (DDR_ADR_I),
167                      .RAM_DAT_I        (DDR_DAT_I),
168                      .RAM_WE_I         (DDR_WE_I),
169                      .RAM_CYC_I        (DDR_CYC_I),
170                      .RAM_STB_I        (DDR_STB_I),
171                      .RAM_SEL_I        (DDR_SEL_I),
172                      .RAM_CTI_I        (DDR_CTI_I),
173                      .RAM_BTE_I        (DDR_BTE_I),
174                      .RAM_LOCK_I       (DDR_LOCK_I),
175                      .RAM_DAT_O        (DDR_DAT_O),
176                      .RAM_ACK_O        (DDR_ACK_O),
177                      .RAM_ERR_O        (DDR_ERR_O),
178                      .RAM_RTY_O        (DDR_RTY_O)
179                      );
180
181
```

```
182
183 endmodule
```

2. pielikums: „Atmiņas modulis“

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:        Ansis Liepkalns
5  //
6  // Create Date:     07:19:49 03/29/2012
7  // Design Name:
8  // Module Name:     mem
9  // Project Name:    Minix 3 on LM32
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module mem(
22             input           MEM_CLK_I,
23             output          SYS_CLK_O,
24             input           RST_I,
25             output          CPU_RESET,
26
27             input           ETH_COL,
28             input           ETH_CRS,
29             output          ETH_MDC,
30             output          ETH_MDIO,
31             input           ETH_RX_CLK,
32             input           ETH_RX_DV,
33             input [4:0]     ETH_RX_D,
34             input           ETH_TX_CLK,
35             output          ETH_TX_EN,
36             output [3:0]    ETH_TX_D,
37             output          ETH_RESET_n,
38
39             inout [15:0]    LPDDR_DQ,
40             output [12:0]   LPDDR_A,
41             output [1:0]    LPDDR_BA,
42             output          LPDDR_RAS_n,
43             output          LPDDR_CAS_n,
44             output          LPDDR_WE_n,
45             output          LPDDR_CKE,
46             output          LPDDR_CK_P,
47             output          LPDDR_CK_N,
48             inout           LPDDR_LDQS,
49             inout           LPDDR_UDQS,
50             inout           LPDDR_UDM,
51             inout           LPDDR_LDM,
52             inout           LPDDR_RZQ,
53
54
55             input [31:0]    DDR_ADR_I,
56             input [32-1:0]  DDR_DAT_I,
57             input           DDR_WE_I,
58             input           DDR_CYC_I,
59             input           DDR_STB_I,
```

```
60              input [32/8-1:0] DDR_SEL_I,
61              input [2:0]      DDR_CTI_I,
62              input [1:0]      DDR_BTE_I,
63              input            DDR_LOCK_I,
64
65              output [32-1:0]  DDR_DAT_O,
66              output           DDR_ACK_O,
67              output reg       DDR_ERR_O,
68              output reg       DDR_RTY_O
69              );
70
71      wire [35:0]              ctrl;
72  `ifdef CHIPSCOPE_DEBUG
73      icon icon (
74                  .CONTROL0(ctrl) // INOUT BUS [35:0]
75                  );
76
77      ila ila (
78                .CONTROL(ctrl),
79                .CLK(SYS_CLK_O),
80                .DATA({DDR_ADR_I[31:0], DDR_DAT_I[31:0], DDR_WE_I,
81                       DDR_CYC_I, DDR_STB_I, DDR_SEL_I[3:0],
82                       DDR_CTI_I[2:0], DDR_BTE_I[1:0],
83                       DDR_DAT_O[31:0], DDR_ACK_O, DDR_ERR_O, DDR_RTY_O}),
84                .TRIG0(DDR_CYC_I)
85                );
86  `endif
87      wire                     sys_clk;
88      wire                     memc_clk;
89
90      assign SYS_CLK_O = sys_clk;
91
92      wire                     cpu_dis_req;
93      reg                      cpu_running;
94
95      wire                     dbg_stb;
96      wire                     dbg_ack;
97
98      wire                     mem_stb;
99      wire                     mem_ack;
100
101     wire [29:0]              mem_addr;
102     wire [3:0]               mem_sel;
103     wire                     mem_wr_en;
104     wire [31:0]              mem_data_wr;
105     wire [31:0]              mem_data_rd;
106
107     wire [29:0]              dbg_addr;
108     wire                     dbg_wr_en;
109     wire [31:0]              dbg_data_wr;
110     wire [31:0]              dbg_data_rd;
111
112     reg [29:0]               cpu_addr;
113     reg                      cpu_wr_en;
114     reg [31:0]               cpu_data_wr;
115
116     wire [31:0]              cpu_data_rd;
117     reg                      cpu_stb;
118     wire                     cpu_ack;
119
120
121     assign mem_addr    = cpu_running ? cpu_addr    : dbg_addr;
```

```
122     assign mem_sel     = cpu_running ? ~DDR_SEL_I  : 'h0;
123     assign mem_wr_en   = cpu_running ? cpu_wr_en   : dbg_wr_en;
124     assign mem_data_wr = cpu_running ? cpu_data_wr : dbg_data_wr;
125     assign cpu_data_rd = mem_data_rd;
126     assign dbg_data_rd = mem_data_rd;
127
128     assign mem_stb     = cpu_running ? cpu_stb : dbg_stb;
129     assign dbg_ack     = mem_ack & !cpu_running;
130     assign cpu_ack     = mem_ack &  cpu_running;
131
132
133     assign DDR_DAT_O = cpu_data_rd;
134     assign DDR_ACK_O = cpu_ack & DDR_CYC_I & DDR_STB_I;
135
136
137     always @(*)
138       begin
139          if (CPU_RESET)
140            begin
141               cpu_addr = 0;
142               cpu_data_wr = 0;
143               cpu_wr_en = 0;
144               cpu_stb = 0;
145            end
146          else if (DDR_CYC_I && DDR_STB_I)
147            begin
148               cpu_addr = DDR_ADR_I;
149               cpu_data_wr = DDR_DAT_I;
150               cpu_wr_en = DDR_WE_I;
151               cpu_stb = 1;
152            end
153          else
154            begin
155               cpu_stb = 0;
156            end
157       end
158
159     always @(posedge sys_clk)
160       begin
161          if (cpu_running)
162            begin
163               if ((cpu_dis_req && !DDR_CYC_I && !DDR_STB_I) ||
164                   CPU_RESET)
165                 cpu_running <= 0;
166            end
167          else if (!cpu_dis_req)
168            cpu_running <= 1;
169       end
170
171     memctrl memory (
172                     .sys_clk     (sys_clk),
173                     .memc_clk    (MEM_CLK_I),
174                     .rst         (RST_I),
175                     .mem_ready   (mem_ready),
176
177                     .lpddr_dq    (LPDDR_DQ),
178                     .lpddr_a     (LPDDR_A),
179                     .lpddr_ba    (LPDDR_BA),
180                     .lpddr_ras_n (LPDDR_RAS_n),
181                     .lpddr_cas_n (LPDDR_CAS_n),
182                     .lpddr_we_n  (LPDDR_WE_n),
183                     .lpddr_cke   (LPDDR_CKE),
```

```
184                     .lpddr_ck_p   (LPDDR_CK_P),
185                     .lpddr_ck_n   (LPDDR_CK_N),
186                     .lpddr_ldqs   (LPDDR_LDQS),
187                     .lpddr_udqs   (LPDDR_UDQS),
188                     .lpddr_udm    (LPDDR_UDM),
189                     .lpddr_ldm    (LPDDR_LDM),
190                     .lpddr_rzq    (LPDDR_RZQ),
191
192                     .stb          (mem_stb),
193                     .ack          (mem_ack),
194
195                     .mem_addr     (mem_addr),
196                     .mem_sel      (mem_sel),
197                     .mem_wr_en    (mem_wr_en),
198                     .mem_data_wr  (mem_data_wr),
199                     .mem_data_rd  (mem_data_rd)
200                     );
201
202     dbg debug (
203                 .sys_clk      (sys_clk),
204                 .rst          (RST_I),
205
206                 .e_col        (ETH_COL),
207                 .e_crs        (ETH_CRS),
208                 .e_mdc        (ETH_MDC),
209                 .e_mdio       (ETH_MDIO),
210                 .e_rx_clk     (ETH_RX_CLK),
211                 .e_rx_dv      (ETH_RX_DV),
212                 .e_rxd        (ETH_RX_D),
213                 .e_tx_clk     (ETH_TX_CLK),
214                 .e_tx_en      (ETH_TX_EN),
215                 .e_txd        (ETH_TX_D),
216                 .e_reset_n    (ETH_RESET_n),
217
218                 .cpu_dis_req  (cpu_dis_req),
219                 .cpu_running  (cpu_running),
220                 .cpu_reset    (CPU_RESET),
221                 .dbg_stb      (dbg_stb),
222                 .dbg_ack      (dbg_ack),
223
224                 .mem_addr     (dbg_addr),
225                 .mem_wr_en    (dbg_wr_en),
226                 .mem_data_wr  (dbg_data_wr),
227                 .mem_data_rd  (dbg_data_rd)
228                 );
229 endmodule
```

3. pielikums: „Atmiņas kontrollera modulis“

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:        Ansis Liepkalns
5  //
6  // Create Date:     08:13:25 03/29/2012
7  // Design Name:
8  // Module Name:     memctrl
9  // Project Name:    Minix 3 on LM32
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module memctrl(
22                 output              sys_clk,
23                 input               memc_clk,
24                 input               rst,
25                 output              mem_ready,
26
27                 inout [15:0]        lpddr_dq,
28                 output [12:0]       lpddr_a,
29                 output [1:0]        lpddr_ba,
30                 output              lpddr_ras_n,
31                 output              lpddr_cas_n,
32                 output              lpddr_we_n,
33                 output              lpddr_cke,
34                 output              lpddr_ck_p,
35                 output              lpddr_ck_n,
36                 inout               lpddr_ldqs,
37                 inout               lpddr_udqs,
38                 inout               lpddr_udm,
39                 inout               lpddr_ldm,
40                 inout               lpddr_rzq,
41
42                 input               stb,
43                 output reg          ack,
44
45                 input [29:0]        mem_addr,
46                 input [3:0]         mem_sel,
47                 input               mem_wr_en,
48                 input [31:0]        mem_data_wr,
49                 output reg [31:0]   mem_data_rd
50     );
51
52
53 `define MEM_WR     4'b0000
54 `define MEM_RD     4'b0001
55 `define MEM_WR_PR  4'b0010
56 `define MEM_RD_PR  4'b0011
57
58
59 `define MEM_STATE_IDLE 0
```

```
60  `define MEM_STATE_WR_1 1
61  `define MEM_STATE_WR_2 2
62  `define MEM_STATE_RD_1 3
63  `define MEM_STATE_RD_2 4
64
65     reg [3:0]                       mem_state = `MEM_STATE_IDLE;
66
67     reg [2:0]                       c3_p0_cmd_instr;
68     reg [5:0]                       c3_p0_cmd_bl;
69     reg [29:0]                      c3_p0_cmd_byte_addr;
70     reg [3:0]                       c3_p0_wr_mask;
71     reg [31:0]                      c3_p0_wr_data;
72     wire [6:0]                      c3_p0_wr_count;
73     wire [31:0]                     c3_p0_rd_data;
74     wire [6:0]                      c3_p0_rd_count;
75     reg                             c3_p0_rd_en;
76     wire                            c3_p0_rd_empty;
77     reg                             c3_p0_wr_en;
78     reg                             c3_p0_cmd_en;
79     wire                            c3_clk0;
80
81     wire                            calib_done;
82     assign mem_ready = calib_done;
83
84     always @(negedge sys_clk)
85       begin
86          if (calib_done) // Do not do anything until calibration is done
87            begin
88               if (!rst &&
89                   stb &&          // Data valid strobe
90                   mem_state == `MEM_STATE_IDLE)
91                 begin
92                    ack <= 0;
93
94                    c3_p0_cmd_byte_addr <= mem_addr;    // Set address
95
96                    if(mem_wr_en)
97                      begin
98                         c3_p0_wr_data <= mem_data_wr; // Set FIFO data
99                         c3_p0_cmd_instr <= `MEM_WR;   // Set Write instruction
100                        mem_state <= `MEM_STATE_WR_1; // Next state
101                        c3_p0_wr_en <= 1;             // Set FIFO write enable
102                        c3_p0_rd_en <= 0;             // Set FIFO read disable
103                        c3_p0_wr_mask <= mem_sel;     // Set write mask
104                     end
105                   else
106                     begin
107                        c3_p0_cmd_instr <= `MEM_RD;   // Set Read instruction
108                        mem_state <= `MEM_STATE_RD_1; // Next state
109                        c3_p0_cmd_en <= 1;            // Set command enable
110                        c3_p0_wr_en <= 0;             // Set FIFO write disable
111                     end
112                end
113
114             if (rst)
115               begin
116                  mem_state <= `MEM_STATE_IDLE;
117               end
118             else
119               begin
120                  case (mem_state)
121                    `MEM_STATE_WR_1:
```

```
122                         begin
123                             c3_p0_wr_en <= 0;                 // Set FIFO write disable
124                             mem_state <= `MEM_STATE_WR_2;     // Next state
125                             c3_p0_cmd_en <= 1;                // Set command enable
126                         end
127
128                       `MEM_STATE_WR_2:
129                         begin
130                             c3_p0_cmd_en <= 0;              // Set command disable
131                             if (c3_p0_wr_empty)
132                               begin
133                                   mem_state <= `MEM_STATE_IDLE; // Next state
134                                   mem_data_rd <= mem_data_wr;   // Store data
135                                   ack <= 1;
136                               end
137                         end
138
139                       `MEM_STATE_RD_1:
140                         begin
141                             c3_p0_cmd_en <= 0;              // Set command disable
142                             mem_state <= `MEM_STATE_RD_2;
143                             c3_p0_rd_en <= 1;               // Set read enable
144                         end
145
146                       `MEM_STATE_RD_2:
147                         begin
148                             if (!c3_p0_rd_empty)
149                               begin
150                                   mem_data_rd <= c3_p0_rd_data;// Store data
151                                   mem_state <= `MEM_STATE_IDLE;
152                                   ack <= 1;
153                               end
154                         end
155
156                       default:
157                         begin
158                             ack <= 0;
159                         end
160                     endcase
161                 end
162           end
163       end
164
165
166
167       memc # (
168                 .C3_P0_MASK_SIZE(4),
169                 .C3_P0_DATA_PORT_SIZE(32),
170                 .C3_P1_MASK_SIZE(4),
171                 .C3_P1_DATA_PORT_SIZE(32),
172                 .DEBUG_EN(0),
173                 .C3_MEMCLK_PERIOD(10000),
174                 .C3_CALIB_SOFT_IP("TRUE"),
175                 .C3_SIMULATION("FALSE"),
176                 .C3_RST_ACT_LOW(0),
177                 .C3_INPUT_CLK_TYPE("SINGLE_ENDED"),
178                 .C3_MEM_ADDR_ORDER("BANK_ROW_COLUMN"),
179                 .C3_NUM_DQ_PINS(16),
180                 .C3_MEM_ADDR_WIDTH(13),
181                 .C3_MEM_BANKADDR_WIDTH(2)
182                 )
183       u_memc (
```

```
184             .c3_sys_clk             (memc_clk),
185             .c3_sys_rst_i           (rst),
186
187             .mcb3_dram_dq           (lpddr_dq),
188             .mcb3_dram_a            (lpddr_a),
189             .mcb3_dram_ba           (lpddr_ba),
190             .mcb3_dram_ras_n        (lpddr_ras_n),
191             .mcb3_dram_cas_n        (lpddr_cas_n),
192             .mcb3_dram_we_n         (lpddr_we_n),
193             .mcb3_dram_cke          (lpddr_cke),
194             .mcb3_dram_ck           (lpddr_ck_p),
195             .mcb3_dram_ck_n         (lpddr_ck_n),
196             .mcb3_dram_dqs          (lpddr_ldqs),
197             .mcb3_dram_udqs         (lpddr_udqs),     // for X16 parts
198             .mcb3_dram_udm          (lpddr_udm),      // for X16 parts
199             .mcb3_dram_dm           (lpddr_ldm),
200
201             .c3_clk0                (sys_clk),
202             .c3_rst0                (c3_rst0),
203
204
205             .c3_calib_done          (calib_done),
206
207             .mcb3_rzq               (lpddr_rzq),
208             .c3_p0_cmd_clk          (sys_clk),
209
210             .c3_p0_cmd_en           (c3_p0_cmd_en),
211             .c3_p0_cmd_instr        (c3_p0_cmd_instr),
212             .c3_p0_cmd_bl           (c3_p0_cmd_bl),
213             .c3_p0_cmd_byte_addr    (c3_p0_cmd_byte_addr),
214             .c3_p0_cmd_empty        (c3_p0_cmd_empty),
215             .c3_p0_cmd_full         (c3_p0_cmd_full),
216
217             .c3_p0_wr_clk           (sys_clk),
218             .c3_p0_wr_en            (c3_p0_wr_en),
219             .c3_p0_wr_mask          (c3_p0_wr_mask),
220             .c3_p0_wr_data          (c3_p0_wr_data),
221             .c3_p0_wr_full          (c3_p0_wr_full),
222             .c3_p0_wr_empty         (c3_p0_wr_empty),
223             .c3_p0_wr_count         (c3_p0_wr_count),
224             .c3_p0_wr_underrun      (c3_p0_wr_underrun),
225             .c3_p0_wr_error         (c3_p0_wr_error),
226
227             .c3_p0_rd_clk           (sys_clk),
228             .c3_p0_rd_en            (c3_p0_rd_en),
229             .c3_p0_rd_data          (c3_p0_rd_data),
230             .c3_p0_rd_full          (c3_p0_rd_full),
231             .c3_p0_rd_empty         (c3_p0_rd_empty),
232             .c3_p0_rd_count         (c3_p0_rd_count),
233             .c3_p0_rd_overflow      (c3_p0_rd_overflow),
234             .c3_p0_rd_error         (c3_p0_rd_error)
235             );
236
237 endmodule
```

4. pielikums: „Ethernet programmēšanas un atkļūdošanas modulis“

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:        Ansis Liepkalns
5  //
6  // Create Date:     07:22:55 03/29/2012
7  // Design Name:
8  // Module Name:     dbg
9  // Project Name:    Minix 3 on LM32
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module dbg(
22             input              sys_clk,
23             input              rst,
24
25             input              e_col,
26             input              e_crs,
27             output reg         e_mdc = 0,
28             output reg         e_mdio = 0,
29             input              e_rx_clk,
30             input              e_rx_dv,
31             input [4:0]        e_rxd,
32             input              e_tx_clk,
33             output reg         e_tx_en,
34             output reg [3:0]   e_txd,
35             output reg         e_reset_n = 1,
36
37             output             cpu_dis_req,
38             input              cpu_running,
39             output reg         cpu_reset,
40             output reg         dbg_stb,
41             input              dbg_ack,
42
43             output reg [29:0]  mem_addr,
44             output reg         mem_wr_en,
45             output reg [31:0]  mem_data_wr,
46             input [31:0]       mem_data_rd
47             );
48
49 `define DATA_WORDS (8)
50 `define FRAME_BYTES (8 + 14 + 20 + 8 + `DATA_WORDS*4 + 8 + 4)
51
52    // ---------------------------------------------------------------------------
53    // Debug interface data
54 `define CMD_POS  (50 + 0)
55 `define LEN_POS  (50 + 1)
56 `define ADDR_POS (50 + 4)
57 `define DATA_POS (50 + 8)
58
59    wire [7:0]                 rx_cmd;
```

```
60      wire [7:0]                rx_len;
61      wire [31:0]               rx_addr;
62
63      assign rx_cmd  = rx_frame[`CMD_POS];
64      assign rx_len  = rx_frame[`LEN_POS];
65      assign rx_addr = {rx_frame[`ADDR_POS + 0], rx_frame[`ADDR_POS + 1],
66                        rx_frame[`ADDR_POS + 2], rx_frame[`ADDR_POS + 3]};
67
68      reg [3:0]                 data_state;
69      reg [7:0]                 dataw_cntr = 1;
70      reg                       rst_dataw_cntr;
71
72      reg                       init_dbg = 1;
73      assign cpu_dis_req = init_dbg;
74
75      // Data processing states
76  `define DATA_IDLE    0
77  `define DATA_INIT1   1
78  `define DATA_INIT2   2
79  `define DATA_INIT3   3
80  `define DATA_CLEAR   4
81  `define DATA_READ    5
82  `define DATA_WRITE   6
83
84
85
86      // ------------------------------------------------------------------
87      // Receiver data
88      reg [15:0]                rx_nibbles = 'hFFFF;
89      reg                       reset_rx_nibbles;
90      reg [3:0]                 prev_rx_nibble;
91      reg                       prev_e_rx_dv;
92      reg [7:0]                 rx_frame [`FRAME_BYTES-1:0];
93      reg                       rx_frame_rdy;
94      reg [7:0]                 rx_frame_rdy_cntr;
95
96      // ------------------------------------------------------------------
97      // Transmitter data
98      reg [15:0]                tx_nibbles = 'hFFFF;
99      reg [7:0]                 tx_frame [`FRAME_BYTES-1:0];
100     initial $readmemh("frame_init.txt", tx_frame);
101     wire                      do_tx;
102     reg [7:0]                 do_tx_cntr;
103     reg [15:0]                gap_nibbles;
104
105     reg [7:0]                 crc_byte;
106     reg [31:0]                tx_crc;
107     wire [7:0]                d;
108     wire [31:0]               c;
109     assign d = crc_byte;
110     assign c = tx_crc;
111
112     wire [35:0]               ctrl;
113
114     assign do_tx = do_tx_cntr > 0 ? 1 : 0;
115
116     // ------------------------------------------------------------------
117     // Positive edge of system clock:
118     //   - read / write data
119     always @(posedge sys_clk)
120       begin
121         if (data_state == `DATA_IDLE)
```

```
122             begin
123                 if (do_tx_cntr > 0)     // Ready to transmit
124                   do_tx_cntr <= do_tx_cntr - 1;
125                 else if (rx_frame_rdy) // A new frame has been received
126                   data_state <= `DATA_INIT1;
127             end
128           else if (data_state == `DATA_INIT1)
129             begin
130                 // Stop the CPU
131                 if (rx_cmd == "s" || rx_cmd == "w" || rx_cmd == "r" ||
132                     rx_cmd == "b")
133                   init_dbg <= 1;
134
135                 // Reset data word counter
136                 dataw_cntr <= 0;
137
138                 // Copy the command and length
139                 tx_frame[`CMD_POS] <= rx_cmd;
140                 tx_frame[`LEN_POS] <= rx_len;
141                 data_state <= `DATA_INIT2;
142             end
143           else if (data_state == `DATA_INIT2)
144             begin
145                 // Copy the address
146                 {tx_frame[`ADDR_POS + 0], tx_frame[`ADDR_POS + 1],
147                  tx_frame[`ADDR_POS + 2], tx_frame[`ADDR_POS + 3]}
148                   <= rx_addr;
149                 data_state <= `DATA_INIT3;
150             end
151           else if (data_state == `DATA_INIT3)
152             begin
153                 case (rx_cmd)
154                   "w":
155                     begin
156                         data_state <= `DATA_WRITE;
157                         // Initiate the first write
158                         mem_addr <= rx_addr;
159                         mem_wr_en <= 1;
160                         mem_data_wr <=
161                         {rx_frame[`DATA_POS + 0],
162                          rx_frame[`DATA_POS + 1],
163                          rx_frame[`DATA_POS + 2],
164                          rx_frame[`DATA_POS + 3]};
165                     end
166                   "r":
167                     begin
168                         data_state <= `DATA_READ;
169
170                         // Initiate the first read
171                         mem_addr <= rx_addr;
172                         mem_wr_en <= 0;
173                     end
174                   default:
175                     begin
176                         mem_wr_en <= 0;
177                         data_state <= `DATA_CLEAR;
178                     end
179                 endcase
180
181                 dbg_stb <= 1;
182             end
183           else if (data_state != `DATA_IDLE)
```

```
184             begin
185                 // reset?
186                 if (!cpu_running && rx_cmd == "b")
187                   cpu_reset <= 1;
188                 if (!cpu_running && dbg_stb)
189                   begin
190                     if (dbg_ack)
191                       begin
192                         dbg_stb <= 0;
193                         dataw_cntr <= dataw_cntr + 1;
194                       end
195                   end
196                 else
197                   begin
198                     if (dataw_cntr < `DATA_WORDS)
199                       dbg_stb <= 1;
200
201                     case (data_state)
202                       `DATA_READ:
203                         begin
204                           {tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 0],
205                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 1],
206                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 2],
207                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 3]}
208                             <= mem_data_rd;
209
210                           if (dataw_cntr < rx_len)
211                             begin
212                               mem_addr <= rx_addr + (dataw_cntr<<2);
213                             end
214                           else
215                             begin
216                               mem_addr <= 0;
217                             end
218                         end
219                       `DATA_WRITE:
220                         begin
221                           {tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 0],
222                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 1],
223                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 2],
224                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 3]}
225                             <= mem_data_rd;
226
227                           if (dataw_cntr < rx_len)
228                             begin
229                               mem_data_wr
230                                 <= {rx_frame[`DATA_POS + (dataw_cntr<<2) + 0],
231                                     rx_frame[`DATA_POS + (dataw_cntr<<2) + 1],
232                                     rx_frame[`DATA_POS + (dataw_cntr<<2) + 2],
233                                     rx_frame[`DATA_POS + (dataw_cntr<<2) + 3]};
234                               mem_addr <= rx_addr + (dataw_cntr<<2) + 1;
235                             end
236                           else
237                             begin
238                               mem_addr <= 0;
239                               mem_wr_en <= 0;
240                             end
241                         end
242                       default:
243                         begin
244                           {tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 0],
245                            tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 1],
```

```
246                         tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 2],
247                         tx_frame[`DATA_POS + ((dataw_cntr-1)<<2) + 3]}
248                          <= 0;
249                     end
250                 endcase
251
252
253                 if (dataw_cntr == `DATA_WORDS)
254                   begin
255                     cpu_reset <= 0;
256
257                     // CPU can be run only after the reply packet
258                     // has been sent
259                     if (rx_cmd == "c")
260                       init_dbg <= 0;
261
262                     do_tx_cntr <= 16;
263                     data_state <= `DATA_IDLE;
264                     dbg_stb <= 0;
265                   end
266               end
267           end
268         else if (data_state == `DATA_IDLE)
269           begin
270             // Read while idling, not write!
271             mem_wr_en <= 0;
272             dbg_stb <= 0;
273           end
274       end
275
276
277       // ------------------------------------------------------------------------
278     // Negative edge of rx clock:
279     //   - rx nibble counter
280     //   - frame ready detection
281     always @(negedge e_rx_clk)
282       begin
283         if (rx_frame_rdy_cntr == 0)
284           rx_frame_rdy <= 0;
285         else
286           begin
287             rx_frame_rdy <= 1;
288             rx_frame_rdy_cntr <= rx_frame_rdy_cntr - 1;
289           end
290
291         if (reset_rx_nibbles)
292           begin
293             rx_nibbles <= 1;    // The first (0) nibble triggered the
294                                 // reset, begin by second (1)
295           end
296         else if (e_rx_dv)       // Receiving
297           begin
298             if (rx_nibbles == (`FRAME_BYTES) * 2 - 1)
299               begin
300                 // The whole frame has been received
301
302                 if (rx_frame[8+0] == 'hFF && rx_frame[8+1] == 'hFF &&
303                     rx_frame[8+2] == 'hFF && rx_frame[8+3] == 'hFF &&
304                     rx_frame[8+4] == 'hFF && rx_frame[8+5] == 'hFF
305                     &&
306                     {rx_frame[8+36+0], rx_frame[8+36+1]} == 'd9876)
307                   begin
```

```
308                         if (! (rx_cmd == "c" && !init_dbg)) // We are
309                                                             // not
310                                                             // letting
311                                                             // to resume
312                                                             // the CPU
313                                                             // twice
314                           rx_frame_rdy_cntr <= 16;//64;
315                       end
316
317                     rx_nibbles <= rx_nibbles + 1;
318                   end
319                 else if (rx_nibbles < (`FRAME_BYTES) * 2)
320                   begin
321                     rx_nibbles <= rx_nibbles + 1;
322                   end
323             end
324       end
325
326
327     // ---------------------------------------------------------------------
328     // Positive edge of rx clock:
329     //   - data valid detection
330     //   - store rx frame when data is valid
331     always @(posedge e_rx_clk)
332       begin
333         // This is the first nibble, initiate a receive
334         if (!prev_e_rx_dv && e_rx_dv)
335           begin
336             prev_e_rx_dv <= 1;
337             reset_rx_nibbles <= 1;
338
339             prev_rx_nibble <= e_rxd[3:0];
340           end
341         // Not receiving
342         else if (prev_e_rx_dv && !e_rx_dv)
343           begin
344             prev_e_rx_dv <= 0;
345           end
346         // Receiving
347         else if (e_rx_dv)
348           begin
349             reset_rx_nibbles <= 0;
350
351             if (rx_nibbles>>1 < `FRAME_BYTES)
352               begin
353                 if (rx_nibbles[0]) // Odd nibble
354                   rx_frame[rx_nibbles>>1] <= {e_rxd[3:0], prev_rx_nibble[3:0]};
355                 else               // Even nibble
356                   prev_rx_nibble <= e_rxd[3:0];
357               end
358           end
359       end
360
361     // ---------------------------------------------------------------------
362     // Positive edge of tx clock:
363     //   - gap and tx nibble counter
364     always @(posedge e_tx_clk)
365       begin
366         if (do_tx)
367           begin
368             gap_nibbles <= 'd160;
369           end
```

```
370         else if (gap_nibbles == 1)
371           begin
372              gap_nibbles <= 0;
373              tx_nibbles  <= 0;
374           end
375         else if (gap_nibbles > 0)
376           begin
377              gap_nibbles <= gap_nibbles - 1;
378           end
379         else if (e_tx_en)
380           tx_nibbles <= tx_nibbles + 1;
381       end
382
383     // ------------------------------------------------------------------
384     // Negative edge of tx clock:
385     //   - tx frame output to txd
386     //   - FCS calculation
387     always @(negedge e_tx_clk)
388       begin
389          if (tx_nibbles < (`FRAME_BYTES) * 2)
390            begin
391               e_tx_en <= 1;
392               if (!tx_nibbles[0]) // Even nibble
393                 begin
394                    if (tx_nibbles == 0)
395                      tx_crc <= 32'hFFFFFFFF;
396
397                    crc_byte <= tx_frame[tx_nibbles>>1];
398
399                    case (tx_nibbles)
400                      (`FRAME_BYTES-4) * 2 + 0: e_txd[3:0] <= tx_crc[27:24] ^ 'hF;
401                      (`FRAME_BYTES-4) * 2 + 2: e_txd[3:0] <= tx_crc[19:16] ^ 'hF;
402                      (`FRAME_BYTES-4) * 2 + 4: e_txd[3:0] <= tx_crc[11: 8] ^ 'hF;
403                      (`FRAME_BYTES-4) * 2 + 6: e_txd[3:0] <= tx_crc[ 4: 0] ^ 'hF;
404
405                      default: e_txd[3:0] <= tx_frame[tx_nibbles>>1][3:0];
406                    endcase
407
408
409                 end
410               else                // Odd nibble
411                 begin
412                    if (tx_nibbles >= 8 * 2 &&
413                        tx_nibbles < (`FRAME_BYTES-4) * 2)
414                      tx_crc <=
415                               // generated using easics tool from:
416                               //    http://www.easics.be/webtools/crctool
417                               {d[0]^d[5]^d[6]^c[23]^c[30]^c[29]^c[24],
418                                d[4]^d[5]^c[22]^c[29]^c[28],
419                                d[1]^d[3]^d[4]^d[7]^c[21]^c[31]^c[28]^c[27]^c[25],
420                                d[0]^d[2]^d[3]^d[6]^c[20]^c[30]^c[27]^c[26]^c[24],
421                                d[1]^d[2]^d[5]^c[19]^c[29]^c[26]^c[25],
422                                d[0]^d[1]^d[4]^c[18]^c[28]^c[25]^c[24],
423                                d[0]^d[3]^c[17]^c[27]^c[24],
424                                d[2]^c[16]^c[26],
425                                d[2]^d[3]^d[7]^c[15]^c[31]^c[27]^c[26],
426                                d[1]^d[2]^d[6]^c[14]^c[30]^c[26]^c[25],
427                                d[0]^d[1]^d[5]^c[13]^c[29]^c[25]^c[24],
428                                d[0]^d[4]^c[12]^c[28]^c[24],
429                                d[3]^c[11]^c[27],
430                                d[2]^c[10]^c[26],
431                                d[7]^c[9]^c[31],
```

```
                                d[1]^d[6]^d[7]^c[8]^c[31]^c[30]^c[25],
                                d[3]^d[4]^d[6]^d[7]^c[7]^c[31]^c[30]^c[28]^c[27],
                                d[2]^d[3]^d[5]^d[6]^c[6]^c[30]^c[29]^c[27]^c[26],
                                d[2]^d[4]^d[5]^d[7]^c[5]^c[31]^c[29]^c[28]^c[26],
                                d[3]^d[4]^d[6]^d[7]^c[4]^c[31]^c[30]^c[28]^c[27],
                                d[1]^d[2]^d[3]^d[5]^d[6]^d[7]^c[3]^c[31]^c[30]^c[29]^c[27]^c[26]^c[25],
                                d[0]^d[1]^d[2]^d[4]^d[5]^d[6]^c[2]^c[30]^c[29]^c[28]^c[26]^c[25]^c[24],
                                d[0]^d[1]^d[3]^d[4]^d[5]^c[1]^c[29]^c[28]^c[27]^c[25]^c[24],
                                d[0]^d[2]^d[3]^d[4]^c[0]^c[28]^c[27]^c[26]^c[24],
                                d[1]^d[7]^c[31]^c[25],
                                d[0]^d[1]^d[6]^d[7]^c[31]^c[30]^c[25]^c[24],
                                d[0]^d[1]^d[5]^d[6]^d[7]^c[31]^c[30]^c[29]^c[25]^c[24],
                                d[0]^d[4]^d[5]^d[6]^c[30]^c[29]^c[28]^c[24],
                                d[1]^d[3]^d[4]^d[5]^d[7]^c[31]^c[29]^c[28]^c[27]^c[25],
                                d[0]^d[1]^d[2]^d[3]^d[4]^d[6]^d[7]^c[31]^c[30]^c[28]^c[27]^c[26]
                                  ^c[25]^c[24],
                                d[0]^d[1]^d[2]^d[3]^d[5]^d[6]^c[30]^c[29]^c[27]^c[26]^c[25]^c[24],
                                d[0]^d[2]^d[4]^d[5]^d[7]^c[31]^c[29]^c[28]^c[26]^c[24]
                                };
                    case (tx_nibbles)
                      (`FRAME_BYTES-4) * 2 + 1: e_txd[3:0] <= tx_crc[31:28] ^ 'hF;
                      (`FRAME_BYTES-4) * 2 + 3: e_txd[3:0] <= tx_crc[23:20] ^ 'hF;
                      (`FRAME_BYTES-4) * 2 + 5: e_txd[3:0] <= tx_crc[15:12] ^ 'hF;
                      (`FRAME_BYTES-4) * 2 + 7: e_txd[3:0] <= tx_crc[ 7: 4] ^ 'hF;
                      default: e_txd[3:0] <= tx_frame[tx_nibbles>>1][7:4];
                    endcase
                end

            end
          else
            begin
               e_tx_en <= 0;
            end
      end

endmodule
```

5. pielikums: „Atmiņas aizsardzības modulis“

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:        Ansis Liepkalns
5  //
6  // Create Date:     17:23:43 03/29/2012
7  // Design Name:
8  // Module Name:     lm32_mpu
9  // Project Name:    Minix 3 on LM32
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module lm32_mpu(
22                 input           clk,
23                 input           rst,
24                 output          privileged_mode,
25
26                 input           exception_enter,
27                 input           exception_leave,
28
29                 // Subset of Wishbone interface from CPU
30                 input [31:0]    M_ADR_I,
31                 input [32-1:0]  M_DAT_I,
32                 input           M_WE_I,
33
34                 input           M_CYC_I,
35                 input           M_STB_I,
36                 input [32/8-1:0] M_SEL_I,
37                 input [2:0]     M_CTI_I,
38                 input [1:0]     M_BTE_I,
39                 input           M_LOCK_I,
40
41                 output          M_ACK_O,
42                 output          M_ERR_O,
43
44                 // Subset of Wishbone interface to peripherals
45                 output reg      S_WE_I,
46
47                 output reg      S_CYC_I,
48                 output reg      S_STB_I,
49
50                 input           S_ACK_O,
51                 input           S_ERR_O
52             );
53
54 `define MPU_ADR       32'hF0000000
55 `define IO_ADR        32'h70000000
56
57 `define SEGMENT_COUNT 4
58    reg [31:0]              segment_addr [`SEGMENT_COUNT-1:0];
59    reg [31:0]              segment_len  [`SEGMENT_COUNT-1:0];
```

```
60
61      reg             protect, tprotect;
62      reg             mpu_ack;
63      reg             user_mode;
64
65      assign privileged_mode = ~user_mode;
66
67      integer         i;
68
69      assign M_ACK_O = S_ACK_O | (mpu_ack & M_CYC_I & M_STB_I);
70      assign M_ERR_O = S_ERR_O | (protect & M_CYC_I & M_STB_I);
71
72      always @(*)
73        begin
74           if (rst)
75                user_mode = 0;
76           else
77             begin
78                if (exception_enter)
79                  user_mode = 0;
80                else if (exception_leave)
81                  user_mode = 1;
82             end
83        end
84
85
86      always @(*)
87        begin
88           tprotect = 0;
89           if (user_mode)  // Protect only when non-privileged
90             begin
91                for (i = 0; i < `SEGMENT_COUNT; i = i + 1)
92                  begin
93                     if (M_ADR_I < segment_addr[i] ||
94                         M_ADR_I > segment_addr[i] + segment_len[i])
95                       tprotect = 1;
96                  end
97             end
98
99           protect = tprotect;
100
101          if (user_mode && protect)
102            begin
103               S_WE_I = 0;
104               if (M_ADR_I >= `IO_ADR) // Protect I/O space from reading as well
105                 begin
106                    S_CYC_I = 0;
107                    S_STB_I = 0;
108                 end
109               else
110                 begin
111                    S_CYC_I = M_CYC_I;
112                    S_STB_I = M_STB_I;
113                 end
114          end // if (user_mode && protect)
115          else
116            begin
117               S_WE_I = M_WE_I;
118               S_CYC_I = M_CYC_I;
119               S_STB_I = M_STB_I;
120          end // else: !if(user_mode && protect)
121       end
```

```
122
123     always @(posedge clk)
124       begin
125          if (M_CYC_I && M_STB_I)
126            begin
127               for (i = 0; i < `SEGMENT_COUNT; i = i + 1)
128                 begin
129                    if (M_ADR_I == `MPU_ADR + i*4)
130                      begin
131                         mpu_ack <= 1;
132                         segment_addr[i] <= M_DAT_I;
133                      end
134                    if (M_ADR_I == `MPU_ADR + 'h100+ i*4)
135                      begin
136                         mpu_ack <= 1;
137                         segment_len[i] <= M_DAT_I;
138                      end
139                 end
140            end // if (M_CYC_I && M_STB_I)
141          else
142            begin
143               mpu_ack <= 0;
144            end // else: !if(M_CYC_I && M_STB_I)
145       end
146  endmodule
```

6. pielikums: „Atkļūdošanas rīks datoram“

```
1  #include <arpa/inet.h>
2  #include <netinet/in.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <sys/types.h>
7  #include <sys/socket.h>
8  #include <unistd.h>
9  #include <errno.h>
10 #include <net/if.h>
11
12 #define MAX_WORDS 8
13 #define BUFLEN (8 + MAX_WORDS*4)
14 #define PORT   9876
15
16 #define DEST_IP "192.168.7.255"
17 #define MY_IP   "192.168.7.1"
18
19 char buf[BUFLEN];               // Buffer for packet payload
20
21 // header is as follows:
22 //   1 byte for command
23 //   1 byte for length
24 //   2 bytes reserved
25 //   4 bytes for address
26
27 struct sockaddr_in si_other;
28 struct sockaddr_in si_me;
29 struct sockaddr_in si_from;
30
31 int s,                          // Socket
32     slen;                         // Address lenght
33
34 int print_help_exit()
35 {
36     fprintf(stderr,
37             "Allowed params:\n"
38             "\ts ---------------------- - stop\n"
39             "\tc ---------------------- - continue\n"
40             "\tb ---------------------- - restart CPU\n"
41             "\tr   <addr> ------------ - read word\n"
42             "\tw   <addr> <data> ------ - write word\n"
43             "\tf r <addr> <length> --- - read binary to stdout\n"
44             "\tf w <addr> ------------ - write binary from stdin\n"
45         );
46     exit(1);
47 }
48
49
50
51 void do_write(char *str_addr, char *str_data)
52 {
53     memset(buf, 0, BUFLEN);
54     buf[0] = 'w';
55     buf[1] = 1;                   // 1 word
56
57     unsigned long addr = strtoul(str_addr, NULL, 16);
58     unsigned long data = strtoul(str_data, NULL, 16);
59
```

```
60      // The target is big endian
61
62      buf[4] = (addr >> 24) & 0xFF;
63      buf[5] = (addr >> 16) & 0xFF;
64      buf[6] = (addr >>  8) & 0xFF;
65      buf[7] = (addr >>  0) & 0xFF;
66
67      buf[8]  = (data >> 24) & 0xFF;
68      buf[9]  = (data >> 16) & 0xFF;
69      buf[10] = (data >>  8) & 0xFF;
70      buf[11] = (data >>  0) & 0xFF;
71
72      if (sendto(s, buf, BUFLEN, 0,
73                 (const struct sockaddr *)&si_other, slen)==-1) {
74          fprintf(stderr, "sendto() failed\n");
75          exit(1);
76      }
77
78      // Read just to keep in sync
79      int i;
80      for (i = 0; i < 2; i++) {
81          if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)&si_from, &slen)==-1) {
82              fprintf(stderr, "recvfrom() failed\n");
83              exit(1);
84          }
85      }
86  }
87
88  void do_read(char *str_addr)
89  {
90      memset(buf, 0, BUFLEN);
91      buf[0] = 'r';
92      buf[1] = 1;                     // 1 word
93
94      unsigned long addr = strtoul(str_addr, NULL, 16);
95      buf[4] = (addr >> 24) & 0xFF;
96      buf[5] = (addr >> 16) & 0xFF;
97      buf[6] = (addr >>  8) & 0xFF;
98      buf[7] = (addr >>  0) & 0xFF;
99
100     if (sendto(s, buf, BUFLEN, 0,
101                (const struct sockaddr *)&si_other, slen)==-1) {
102         fprintf(stderr, "sendto() failed\n");
103         exit(1);
104     }
105
106     memset(buf, 0, BUFLEN);
107     int i;
108
109     // Receive. Broadcast packets are sent to everyone including
110     // ourselves. Hence two receives.
111     for (i = 0; i < 2; i++) {
112         if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)&si_from, &slen)==-1) {
113             fprintf(stderr, "recvfrom() failed\n");
114             exit(1);
115         }
116
117         if (! strcmp(inet_ntoa(si_from.sin_addr), MY_IP))
118             continue;
119
120         printf("%c : %02x%02x%02x%02x : %02x%02x%02x%02x\n",
121                buf[0]&0xFF,
```

```
122                 buf[4]&0xFF, buf[5]&0xFF, buf[6 ]&0xFF, buf[7 ]&0xFF,
123                 buf[8]&0xFF, buf[9]&0xFF, buf[10]&0xFF, buf[11]&0xFF);
124     }
125 }
126
127 void do_default (char cmd)
128 {
129     // Default action is just sending the command character and skipping
130     // the answer
131
132     memset(buf, 0, BUFLEN);
133
134     buf[0] = cmd;
135
136     if (sendto(s, buf, BUFLEN, 0,
137                 (const struct sockaddr *)&si_other, slen)==-1) {
138         fprintf(stderr, "sendto() failed\n");
139         exit(1);
140     }
141
142     // Read just to keep in sync
143     int i;
144     for (i = 0; i < 2; i++) {
145         if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)&si_from, &slen)==-1) {
146             fprintf(stderr, "recvfrom() failed\n");
147             exit(1);
148         }
149     }
150 }
151
152 void do_read_file(char *str_addr, char *str_len)
153 {
154     unsigned long addr = strtoul(str_addr, NULL, 16);
155     int len = atoi(str_len) / 4 + ((atoi(str_len) % 4) ? 1 : 0);
156
157     int i, j;
158     while (len > 0) {
159         memset(buf, 0, BUFLEN);
160         buf[0] = 'r';
161         if (len >= MAX_WORDS)
162             buf[1] = MAX_WORDS;
163         else
164             buf[1] = len % MAX_WORDS;
165
166         buf[4] = (addr >> 24) & 0xFF;
167         buf[5] = (addr >> 16) & 0xFF;
168         buf[6] = (addr >>  8) & 0xFF;
169         buf[7] = (addr >>  0) & 0xFF;
170
171
172         if (sendto(s, buf, BUFLEN, 0,
173                     (const struct sockaddr *)&si_other, slen)==-1) {
174             fprintf(stderr, "sendto() failed\n");
175             exit(1);
176         }
177
178         memset(buf, 0, BUFLEN);
179         for (i = 0; i < 2; i++) {
180             if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)&si_from, &slen)==-1) {
181                 fprintf(stderr, "recvfrom() failed\n");
182                 exit(1);
183             }
```

```
184
185             if (! strcmp(inet_ntoa(si_from.sin_addr), MY_IP))
186                 continue;
187
188             fwrite(&buf[8], 1, buf[1] * 4, stdout);
189         }
190
191         addr += MAX_WORDS*4;
192         len -= MAX_WORDS;
193     }
194     fflush(stdout);
195 }
196
197 void do_write_file(char *str_addr)
198 {
199     unsigned long addr = strtoul(str_addr, NULL, 16);
200     int bytes = 0;
201
202     int i;
203     int c;
204     memset(buf, 0, BUFLEN);
205     while ((c = getchar()) != EOF)
206     {
207         buf[8 + bytes] = c & 0xFF;
208         bytes++;
209
210         if (bytes == MAX_WORDS*4)
211         {
212             buf[0] = 'w';
213             buf[1] = MAX_WORDS;
214             buf[4] = (addr >> 24) & 0xFF;
215             buf[5] = (addr >> 16) & 0xFF;
216             buf[6] = (addr >>  8) & 0xFF;
217             buf[7] = (addr >>  0) & 0xFF;
218
219
220             if (sendto(s, buf, BUFLEN, 0,
221                        (const struct sockaddr *)&si_other, slen)==-1) {
222                 fprintf(stderr, "sendto() failed\n");
223                 exit(1);
224             }
225
226             // Read just to keep in sync
227             for (i = 0; i < 2; i++) {
228                 if (recvfrom(s, buf, BUFLEN, 0,
229                              (struct sockaddr *)&si_from, &slen)==-1) {
230                     fprintf(stderr, "recvfrom() failed\n");
231                     exit(1);
232                 }
233             }
234
235             addr += MAX_WORDS*4;
236             bytes = 0;
237             memset(buf, 0, BUFLEN);
238         }
239     }
240
241     if (bytes > 0)
242     {
243         buf[0] = 'w';
244         buf[1] = (bytes / 4) + ((bytes % 4) ? 1 : 0);
245
```

```
246         buf[4] = (addr >> 24) & 0xFF;
247         buf[5] = (addr >> 16) & 0xFF;
248         buf[6] = (addr >>  8) & 0xFF;
249         buf[7] = (addr >>  0) & 0xFF;
250 
251 
252         if (sendto(s, buf, BUFLEN, 0,
253                 (const struct sockaddr *)&si_other, slen)==-1) {
254             fprintf(stderr, "sendto() failed\n");
255             exit(1);
256         }
257 
258         // Read just to keep in sync
259         for (i = 0; i < 2; i++) {
260             if (recvfrom(s, buf, BUFLEN, 0,
261                         (struct sockaddr *)&si_from, &slen)==-1) {
262                 fprintf(stderr, "recvfrom() failed\n");
263                 exit(1);
264             }
265         }
266     }
267 }
268 
269 
270 int main(int argc, char *argv[])
271 {
272     int i;
273 
274     // Some (weak) parameter checks
275     if (argc < 2)
276         print_help_exit();
277 
278     if (argv[1][0] == 'r' && argc != 3) // read
279         print_help_exit();
280 
281     if (argv[1][0] == 'w' && argc != 4) // write
282         print_help_exit();
283 
284     if (argv[1][0] == 'f') {
285         if (argc < 3)
286             print_help_exit();
287 
288         if (argv[2][0] == 'r' && argc != 5) // file read
289             print_help_exit();
290 
291         if (argv[2][0] == 'w' && argc != 4) // file write
292             print_help_exit();
293     }
294 
295     // Set up the socket
296     slen = sizeof(si_other);
297 
298     if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1) {
299         fprintf(stderr, "socket() failed\n");
300         exit(1);
301     }
302 
303     memset((char *) &si_other, 0, sizeof(si_other));
304     si_other.sin_family = AF_INET;
305     si_other.sin_port = htons(PORT);
306     if (inet_aton(DEST_IP, &si_other.sin_addr)==0) {
307         fprintf(stderr, "inet_aton() failed\n");
```

```
308             exit(1);
309         }
310 
311 
312         int opt_on = 1;
313         setsockopt(s, SOL_SOCKET, SO_BROADCAST, &opt_on, sizeof(opt_on));
314         setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &opt_on, sizeof(opt_on));
315 
316         memset((char *) &si_me, 0, sizeof(si_me));
317         si_me.sin_family = AF_INET;
318         si_me.sin_port = htons(PORT);
319 
320 
321         si_me.sin_addr.s_addr = htonl(INADDR_ANY);
322 
323         if (bind(s, (const struct sockaddr *)&si_me, sizeof(si_me))==-1)
324         {
325             fprintf(stderr, "bind() failed\n");
326             exit(1);
327         }
328 
329         // Dispatch the command
330 
331         switch (argv[1][0]) {
332         case 'r':
333             do_read(argv[2]);
334             break;
335         case 'w':
336             do_write(argv[2], argv[3]);
337             break;
338         case 'f':
339             if (argv[2][0] == 'r')
340                 do_read_file  (argv[3], argv[4]);
341 
342             if (argv[2][0] == 'w')
343                 do_write_file (argv[3]);
344 
345             break;
346         default:
347             do_default(argv[1][0]);
348         }
349 
350         close(s);
351 
352         return 0;
353     }
```

Maģistra darbs: **UNIX veida mikrokodola operētājsistēma FPGA procesoram**

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____

(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: _____

(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.

(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____

(Metodiķes paraksts)

Recenzents: Dr.sc.comp. Jānis Martinsons

Darbs aizstāvēts datorzinātņu maģistra gala pārbaudījuma komisijas sēdē

_____ prot.nr._____, vērtējums _____

(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____

(Sekretāra paraksts)