VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE FILTRU SÍŤOVÉHO PROVOZU V FPGA PROCESOREM MICROBLAZE

IMPLEMENTATION OF THE NETWORK TRAFFIC FILTER BY MICROBLAZE IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN VIKTORIN
AUTHOR

VEDOUCÍ PRÁCE Ing. JAN KAŠTIL
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá možnostmi hardwarové akcelerace softwarového filtru síťového provozu běžícímu na procesoru MicroBlaze v FPGA čipu Spartan-3E. Akcelerovanou aplikací je standardní firewall Linuxového jádra názývaný iptables, konkrétně jeho rozšíření L7-filter. L7-filter vyhledává regulární výrazy v provozu procházejícím systémem, a tím umožňuje rozpoznávat různé protokoly aplikační vrstvy. Pro jeho významný vliv na výkonnost aplikace byl nahrazen hardwarovou jednotkou řízenou z vlastního modulu Linuxového jádra. Profilací a měřením propustnosti bylo zjištěno, že je takto možné zvýšit propustnost systému i více než dvojnásobně.

Abstract

The thesis explores the area of hardware acceleration of a software network traffic filter running inside processor MicroBlaze in the FPGA Spartan-3E. The accelerated application is widely used firewall from the Linux Kernel called iptables, more precisely its extension L7-filter. L7-filter performs lookups inside network traffic using regular expressions. Because of its significant influence on the application performance, it has been exchanged with a hardware unit controlled from the Linux Kernel. The performance has been increased more than twice.

Klíčová slova

FPGA, MicroBlaze, iptables, síťové filtry, regulární výrazy, NFA.

Keywords

FPGA, MicroBlaze, iptables, network filters, regular expressions, NFA.

Citace

Jan Viktorin: Implementace filtru síťového provozu
v FPGA procesorem Microblaze, bakalářská práce, Brno, FIT VUT v Brně, 2011

Implementace filtru síťového provozu v FPGA procesorem Microblaze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Kaštila

.....
Jan Viktorin
18. května 2011

Poděkování

Na tomto místě bych chtěl poděkovat za velkou ochotu a věcnou zpětnou vazbu při konzultacích svému vedoucímu Ing. Janu Kaštilovi.

© Jan Viktorin, 2011.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **4**

2 Filtrování v softwaru **6**

2.1 Filtrování paketů 6

2.2 Cíl práce 7

2.3 OS Linux a křížová kompilace 8

2.4 Netfilter/iptables 10

3 Hardwarová platforma **14**

3.1 Procesor MicroBlaze 14

3.2 Sběrnice PLB 4.6 16

3.3 Fast Simplex Link 16

3.4 Periferie 18

4 Vývoj aplikací pro MicroBlaze **22**

4.1 Microprocessor Hardware Specification (MHS) 22

4.2 Microprocesor Software Specification (MSS) 24

4.3 Další součásti PSF 24

4.4 Tvorba vlastní periferie pomocí PSF 25

4.5 Projekt v EDK a jeho součásti 26

4.6 Tvorba aplikací v EDK 27

4.7 Možnosti hardwarové akcelerace na MicroBlaze 27

5 Návrh a implementace aplikace **30**

5.1 Popis zpracování paketu 30

5.2 Distribuce OS Linux 31

5.3 Implementace hardwarové platformy 31

5.4 Měření implementovaného systému 32

5.5 Vyhodnocení dat získaných profilací 34

6 Návrh a implementace akcelerátoru **36**

6.1 Návrh rozhraní 36

6.2 Návrh struktury 36

6.3 Implementace akcelerátoru 38

6.4 Měření propustnosti s akcelerátorem 40

6.5 Využité zdroje v FPGA 41

7 Závěr **44**

Literatura **45**

Seznam příloh **48**

1 Úvod

Nacházíme se v době, kdy se již Internet stal základním informačním médiem, na které spoléhají malé i velké společnosti, nadnárodní koncerny, ale i domácnosti. Uživatelé chtějí mít stále rychlejší přístup k informacím a k multimediálnímu obsahu, a to s minimálními náklady na provoz. Velmi rozšířenými zařízeními v této souvislosti jsou malé domácí směrovače, které uživatelům umožňují jednak vlastní připojení k Internetu, ale v neposlední řadě také zabezpečení domácí sítě, případně další služby jako např. IP telefonii nebo digitální televizi.

Současné domácí směrovače a firewally disponují nedostatečným výkonem pro další rozšiřování směrem k rychlostem 100–1000 Mbit/s nebo i vyšším při zachování nebo zlepšování kvality svých služeb. Filtrování provozu je typicky implementováno v procesoru bez hardwarové akcelerace. Vývoj aplikačně specifických ASIC čipů je neúměrně drahý.

Vhodným kandidátem pro řešení těchto problémů se jeví čipy FPGA, které nabízejí nízkou spotřebu cílových zařízení, výkon blížící se ASIC čipům, ale přitom levnější vývoj i pozdější možnosti opravy chyb nebo dokonce rozšiřování funkčnosti už hotového výrobku.

V této práci je naznačen jeden z možných způsobů vývoje těchto vestavěných zařízení za pomoci soft procesoru MicroBlaze podpořeného hardwarovou akcelerací na FPGA čipu Spartan-3E. V kapitole *2 Filtrování v softwaru* se zabývám obecnými charakteristikami síťových filtrů a jedním z nejrozšířenějších softwarových řešení – aplikací *iptables*. V kapitole *3 Hardwarová platforma* popisuji součásti hardwarové platformy pro FPGA vč. procesoru MicroBlaze, na kterou navazuje část *4 Vývoj aplikací pro MicroBlaze* o vývojovém prostředí *Embedded Development Kit* a jeho možnostech při vývoji vestavěných systémů. Poslední dvě kapitoly *5 Návrh a implementace aplikace* a *6 Návrh a implementace akcelerátoru* popisují návrh a implementaci síťového filtru, jeho měření a také provedenou hardwarovou akceleraci.

2 Filtrování v softwaru

2.1 Filtrování paketů

Pro ujasnění hlavního cíle bakalářské práce je nutné definovat, co je myšleno pojmy *filtrování paketů* anebo *paketový filtr*[1].

Podle knihy Building Firewalls lze pojem *paketový filtr* vymezit takto (volně přeloženo):

> *Akce, kterou provádí zařízení pro selektivní řízení toku dat do a ze sítě. Paketové filtry propouštějí nebo blokují pakety, které dále obvykle směrují z jedné sítě do jiné (typicky z Internetu do vnitřní sítě a naopak). Pro tento účel lze nastavit množinu pravidel určujících, které typy paketů budou propuštěny a které zablokovány.* [5, s. 103]

V této knize autoři poznamenávají, že pojem *paketový filtr* může v jiné literatuře označovat zařízení pro výběr paketů ze sítě pro **analýzu**. [5, s. 103]

2.1.1 Základní možnosti filtrování

Filtrování podle hlaviček paketů. Každý paket přenášený po počítačové síti s sebou nese metadata v podobě jedné nebo více hlaviček. Informace z nich získané mohou sloužit právě k filtraci paketů. Pro filtraci se nejčastěji vyžívají tyto [5, s. 105]:

- zdrojová IP adresa,
- cílová IP adresa,
- označení protokolu L4 vrstvy,
- zdrojový port TCP nebo UDP,
- cílový port TCP nebo UDP,
- typ ICMP zprávy,
- velikost paketu.

Filtrování podle rozhraní. Kromě filtrování podle metadat, které nesou pakety samotné, lze využívat informaci o příchozím a odchozím rozhraní zařízení. Jedním z využití je např. kontrola adres používaných ve vnitřních a vnějších sítích. Pakety přicházející z vnější sítě nesoucí vnitřní zdrojovou adresu lze považovat za podvržené, nebo upozorňují na nesprávné nastavení sítě. [5, s. 174]

2.1.2 Stavové filtrování

Paketový filtr může sledovat stav přenosů probíhajících v síti. V takovém případě bývá označován za *stavový filtr*. Filtrování může být např. podmíněno na základě dříve zpracovaných paketů, typicky požadavků. Propuštěny jsou potom pouze odpovědi, na které existuje záznam o provedení požadavku. Stavovost paketového filtru bývá spojována se stavovostí protokolu *TCP*, které omezuje některé typy útoků. [5, s. 169]

[1] tyto pojmy jsou dále chápány téměř jako synonyma lišící se jen mluvnickou kategorií

Protože si stavový filtr musí udržovat informace o stavu jednotlivých spojení, zvyšuje se tím jeho zatížení, což otevírá možnosti pro provádění některých útoků typu *odmítnutí služby* (*denial of service*). Zařízení se při takovém útoku může restartovat (např. kvůli nedostatku paměti), a tím dojde k zahození paketů, které měly být propuštěny. [5, s. 170]

2.1.3 Filtrování podle obsahu paketů

Paketový filtr může pracovat také s daty v paketech. To umožňuje provádět filtraci na základě podrobnějších informací (např. podle URL adres, o které uživatel žádá). Dále může kontrolovat, zda jsou data správně naformátována, ověřovat, zda souhlasí parametry uvedené v hlavičkách (např. že délka paketu odpovídá faktické délce dat). Tím lze zabránit některým útokům typu *odmítnutí služby*. [5, s. 105]

Filtrování podle obsahu paketů může být prováděno formou rozpoznávání aplikačního protokolu. Např. lze vytvořit filtrovací pravidlo pro službu *DNS*, kdy pakety přicházející na port služby *DNS* budou propuštěny pouze v případě, že data v nich obsažená odpovídají očekávané struktuře takových paketů. Tento způsob nemusí být vždy spolehlivý, avšak jedná se o použitelnou metodu. Aby bylo možné provádět takovou analýzu, paketový filtr musí rozumět aplikačnímu protokolu, což je obecně možné jen pro nejpoužívanější protokoly, jako je *HTTP*. Kromě toho jsou v tomto případě na tento typ filtrů kladeny výrazně vyšší nároky na výkon, protože musí zpracovávat daleko větší množství dat než při filtrování podle hlaviček. [5, s. 170–171]

2.1.4 Vybrané vlastnosti paketového filtru

Podle knihy Building Firewalls by měl paketový filtr dovolovat zadávání pravidel pracujících s jakoukoliv informací získanou z hlavičky paketu nebo se vztahující k paketu (např. rozhraní). Jedná se typicky o položky vyjmenované v části *Filtrování podle hlaviček paketů* na straně 6, dále o IP volby (*IP options*) nebo TCP příznaky, případně o informace poskytované jinými filtrovanými protokoly. [5, s. 194]

Důležitou vlastností paketového filtru je také dodržování pořadí zadaných pravidel při jejich aplikaci. To umožňuje předvídat chování systému a lépe jej ladit. Některé paketové filtry nabízejí možnost přeskládat pravidla pro vyšší výkonnost, což může mít negativní vliv na zabezpečení např. v případě, že implementace „přeskládávacího“ algoritmu obsahuje chybu. [5, s. 194–198]

Pravidly by mělo být možné filtrovat nezávisle příchozí a odchozí pakety, případně pakety přicházející z konkrétních rozhraní. To může zvýšit zejm. flexibilitu a výkonnost systému. [5, s. 198–201]

Paketový filtr by měl být schopen provádět logování propuštěných i zahozených paketů. Tato vlastnost dovoluje zjišťovat, zda se někdo nepokouší narušit bezpečnost chráněného subjektu, dále ladit konfiguraci fitru samotného nebo odhalovat probíhající útoky. [5, s. 201–202]

2.2 Cíl práce

Na základě obecných znalostí o paketových filtrech lze nyní definovat cíl práce. Tím je poskytnutí aplikace, která pracuje na vrstvě L2 nebo L3 síťového modelu `ISO OSI`, kde provádí směrování paketů dále do sítě, ale z hlediska filtrování může zasahovat až do vrstvy aplikační.

V implementované aplikaci bude dále třeba identifikovat ty části kódu, které významně ovlivňují rychlost zpracování síťového provozu, tedy které omezují jeho celkovou propustnost. K takovému kódu bude následně navržena alternativa, která potenciálně celkovou propustnost zařízení

zvýší. Alternativní implementace bude vytvořena v logice použitého FPGA čipu s využitím akceleračních možností procesoru MicroBlaze.

2.3 OS Linux a křížová kompilace

V době psaní této práce je k dispozici několik možností, jak získat a přeložit jádro operačního systému (dále jen OS) Linux pro procesor MicroBlaze. Je třeba zvolit, z jakého zdroje získat samotné jádro, jakou verzi použít, dále jaký použít překladač. V neposlední řadě je třeba si uvědomit, že překlad aplikace pro embedded zařízení typicky znamená použít tzv. křížové kompilace, která vyžaduje mít k dispozici kromě překladače též knihovny, assembler a linker[2]. [2]

2.3.1 GNU toolchain

Nejrozšířenějším *toolchainem* pro výše zmíněné účely je GNU toolchain, který je výhodný zejm. z hlediska své dostupnosti pod GNU GPL licencí. I firma Xilinx nabízí záplatované varianty GNU toolchainu pro použití s procesorem MicroBlaze. Tento systém se skládá z několika částí:

gcc
: soubor překladačů několika programovacích jazyků (zejm. C a C++),

binutils
: soubor assemblerů a linkerů,

glibc
: implementace standardní knihovny `libc`,

kernel headers
: hlavičkové soubory použitého Linuxového jádra

Práce s celým GNU toolchainem je popsána v mnoha publikacích na internetu, např. [26], i v tištěné literatuře, např. [4], [3]. Hlavními problémy při portování na novou platformu je implementace *assembleru*, úpravy *gcc* a také překlad knihovny *glibc*. Knihovna *glibc* bývá v embedded prostředí nahrazována alternativní implementací, např. μClibc, a to zejm. kvůli své velikosti a složitosti.

2.3.2 Kompilace Linuxového jádra

Konfigrace jádra vyžaduje nastavit verzi procesoru a některé parametry použité při syntéze procesoru. OS také potřebuje vědět, jakým způsobem má přistupovat k periferiím. K tomuto účelu lze využít *Device Tree*. Jedná se o datovou strukturu, která obsahuje metadata týkající se periferií jako název, verze, adresa, kam je periferie namapována v paměťovém prostoru, atd.

Novější verze Linuxového jádra obsahují mezi přibalenými nástroji pro překlad také program `dtc`. Ten slouží k překladu souboru `<nazev>.dts` do podoby `<nazev>.dtb`. Soubor `<nazev>.dts`

[2] bývá souhrnně označováno jako *toolchain*

je textový, určený pro editaci člověkem nebo vygenerování nějakým nástrojem. Soubor `<nazev>.dtb` je binární, je vložen do výsledného obrazu nahrávaného do paměti procesoru při kompilaci jádra. Pokud je do adresářové struktury Linuxového jádra správně vložen potřebný soubor `<nazev>.dts`, volání `dtc` je provedeno automaticky.

Pro úspěšné přeložení Linuxového jádra pro MicroBlaze je třeba sestavit `Device Tree` podle specifik konkrétního *designu*. Tuto činnost lze automatizovat. Např. mezi nástroji firmy Xilinx[3] lze nalézt soubory

- `device_tree_v0_00_x.mld` a
- `device_tree_v0_00_x.tcl`,

které definují v rámci prostředí EDK operační systém. Ten lze nastavit v příslušném souboru MSS (popsáno v části *4.2 Microprocesor Software Specification (MSS)*). Potom při spuštění programu `libgen` za účelem vygenerování *softwarové platformy* se v adresáři

`microblaze_#/libexec/device_tree_v0_00_x/`[4]

vytvoří soubor `xilinx.dts`.

2.3.3 Zdroje součástí toolchainu

Na Internetu lze nalézt zejm. dvě místa, která poskytují podporu pro vývoj na platformě MicroBlaze spojenou s OS Linux.

Petalogix
Firma Petalogix vyvíjí vlastní distribuci postavenou na Linuxovém jádře, která se nazývá `petalinux`. Základy distribuce tvoří původně distribuce μCLinux a Linuxové jádro verze 2.4, pro kterou vytvořil John Williams, zakladatel této firmy, port na MicroBlaze. Firma má veřejný GIT repozitář na adrese

`http://developer.petalogix.com`,

kde poskytuje Linuxové jádro 2.6, záplatované *binutils*, *gcc* a *glibc*.

Xilinx
Firma Xilinx aktivně podporuje rozvoj Open Source Linuxu pro procesory MicroBlaze. Nabízí své záplatované jádro a nástroje v GIT repozitáři

`http://git.xilinx.com/`.

Firma provozuje webovou stránku

`http://xilinx.wikidot.com/`

s dokumentací k Linuxu na platformě MicroBlaze a PowerPC.

[3] `http://git.xilinx.com/device-tree.git`
[4] kde `#` označuje číslo procesoru MicroBlaze v designu – obvykle je 1

S oběma firmami spolupracuje Michal Šimek, který spravuje oficiální Linuxové jádro pro MicroBlaze. Výsledky jejich spolupráce lze nalézt přímo v GIT repozitářích *vanilla* jádra Linuxu na `kernel.org`.

MicroBlaze a gcc. V uvedených zdrojích lze nalézt několik verzí programu *gcc*, a to ve zdrojovém i předkompilovaném tvaru. Pro účely této práce byly využity verze *gcc* umístěné v repozitářích

- `http://git.xilinx.com/xldk/microblaze_v1.0.git` a
- `http://git.xilinx.com/mb_gnu.git`.

Program *gcc* oficiálně nepodporuje procesor MicroBlaze, proto jej nelze použít. Na stránce projektu *gcc* se lze dočíst, že MicroBlaze bude podporován od verze 4.6. [24]

2.3.4 Zkušenosti s toolchainem

Překlad jádra pro účely této práce byl bezproblémový. Jádro lze přeložit použitím *gcc* od firmy Xilinx a předpřipravené konfigurace jádra pro MicroBlaze s upravením parametrů procesoru.

Žádným z dostupných kompilátorů se mi nepodařilo přeložit knihovnu `glibc`. Tuto knihovnu lze nalézt už přeloženou mezi soubory v repozitáři

`http://git.xilinx.com/xldk/microblaze_v1.0.git`.

2.4 Netfilter/iptables

Součástí jádra operačního systému Linux je framework *netfilter*, který umožňuje nějakým způsobem předzpracovat pakety procházející daným počítačovým systémem. Skládá se ze tří hlavních částí [27]:

- ze záchytných bodů (`hooks`),
- z mechanismu pro zpracování paketů v záchytných bodech a
- z fronty paketů směrem do *userspace* (`ip_queue`).

2.4.1 Záchytné body (`hooks`)

Každý paket, který prochází systémem, prochází přes záchytné body, kde jej *netfilter* umožňuje ovlivnit. Na kterýkoliv záchytný bod lze připojit některý modul jádra OS, např. `ip_tables` nebo `ip6_tables`. Každý modul, který chce ovlivňovat procházející pakety, registruje pomocí funkce `xt_hook_link()` na některé záchytné body obsluhu, která je v daný okamžik zavolána s procházejícím paketem a informacemi o vstupním a výstupním síťovém zařízení.

Obslužná funkce určí, co se s paketem má stát. *Netfilter* definuje množinu návratových kódů, na které umí zareagovat. Jsou to [1, s. 4]:

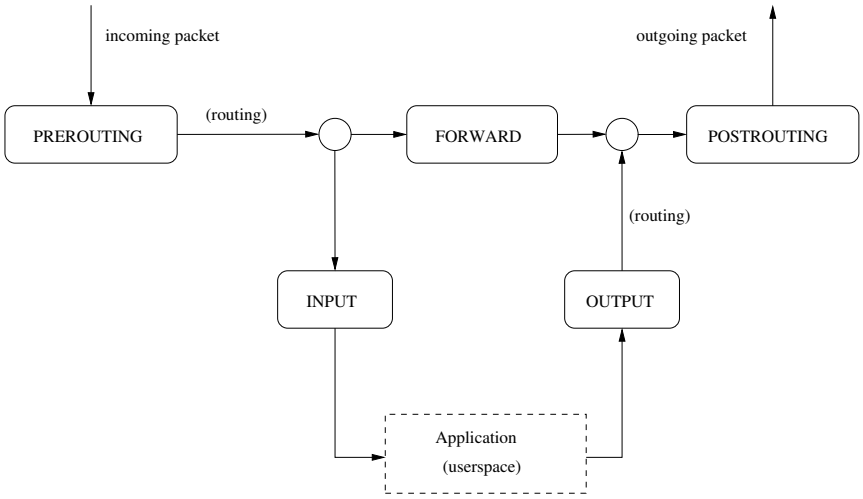
`NF_DROP` – Paket není dále zpracováván a je zahozen.
`NF_ACCEPT` – Paket pokračuje dále (např. k další obsluze).
`NF_STOLEN` – Paket zůstává v režii obsluhy, není dále zpracováván (jeho zpracování mohlo být takto odloženo).

NF_QUEUE – Paket není dále zpracováván a je odeslán do fronty, odkud jej mohou číst uživatelské aplikace.
NF_REPEAT – Způsobí opětovné zavolání obsluh.
NF_STOP – Podobný význam jako NF_ACCEPT, zamezuje opětovnému průchodu paketu.

Netfilter používá pět vestavěných záchytných bodů, které mohou být využity rozšiřujícími moduly [1, s. 4–5]:

- PREROUTING,
- INPUT,
- OUTPUT,
- FORWARD,
- POSTROUTING.

Schéma *2.1 Volání záchytných bodů* znázorňuje, za jakých okolností jsou jednotlivé záchytné body volány. Každý paket je při příchodu ze sítě zpracován v PREROUTING. Potom je předán do směrovače Linuxu (routing). Pokud směrovač rozhodne, že je paket určen k doručení na lokální stanici, zpracování bude pokračovat v INPUT a potenciálně v uživatelské aplikaci. V případě, že má být paket směrován jinam do sítě[5], bude zpracování pokračovat ve FORWARD. Pakety vytvořené na lokální stanici jsou nejprve zpracovávány v OUTPUT. Každý paket, který je odesílán ven ze stanice, je nakonec prověřen v POSTROUTING. [1, s. 5]



Obr. 2.1: Volání záchytných bodů [1, s. 5]

2.4.2 Filtrování pomocí iptables

Program *iptables* umožňuje komunikovat s modulem jádra Linuxu ip_tables, který umožňuje zpracovávat IPv4 provoz. Pro IPv6 je k dispozici analogický program *ip6tables* a v jádře modul ip6_tables. Oba zmíněné moduly jsou postaveny nad frameworkem *netfilter*. Umožňují definovat pravidla pro IP pakety a také akce, které mají být s paketem vykonány v případě, že odpovídá pravidlu.

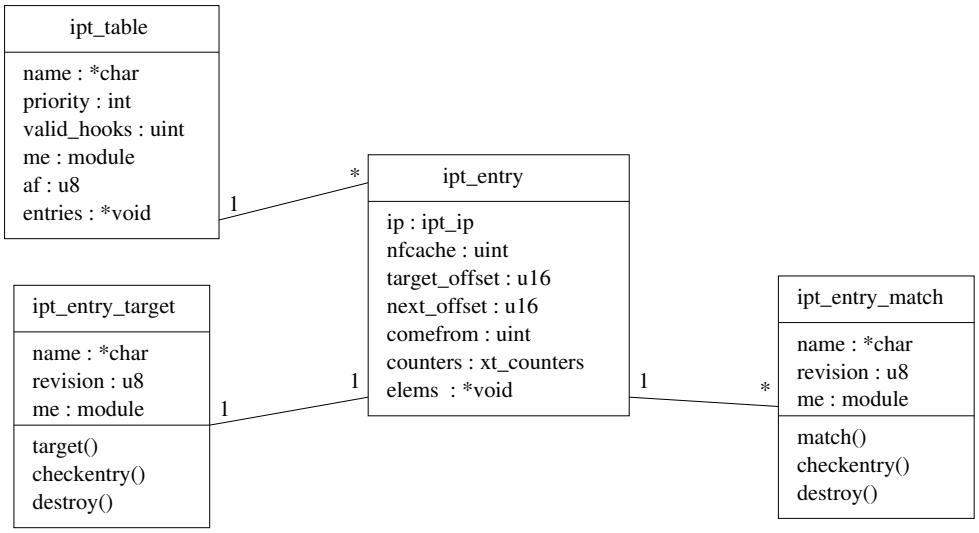
[5] v OS musí být povoleno směrování

Typické použití *iptables* vypadá např. takto:

```
$  iptables -A INPUT  -s 192.168.0.0/24 -j DROP
$  iptables -A OUTPUT -d 10.10.0.0/16   -j DROP        (1)
```

V uvedeném výpisu (1) jsou definována dvě pravidla.

- První je spuštěno v případě, že příchozí paket je ze sítě `192.168.0.0/24`. Tento paket je zahozen akcí `DROP`.
- Druhé je spuštěno v případě, že odchozí paket míří do sítě `10.10.0.0/16`. Tento paket je taktéž zahozen akcí `DROP`. Přesněji, uživatelský program obdrží chybový kód říkající, že odeslání paketu nebylo dovoleno.



Obr. 2.2: Datové struktury v `ip_tables`

Zpracování v `ip_tables` je prováděno pomocí datových struktur `ipt_table`. Ty se skládají ze záznamů `ipt_entry`. Každý záznam je složen z množiny pravidel `ipt_entry_match` a z akce `ipt_entry_target`, která se nad paketem provede v případě, že některé pravidlo uspěje. Provázanost těchto datových struktur je znázorněna v diagramu *2.2 Datové struktury v ip_tables* (pro přehlednost je schéma zjednodušeno). Tabulek v `ip_tables` existuje standardně několik, každá je určena pro řešení jiné skupiny problémů. Implicitní a nejpoužívanější je tabulka `filter`, která umožňuje provádět filtraci paketů (použita v příkladu 1). Dále jsou k dispozici tabulky `nat`, `mangle` a `raw`. Je možné definovat vlastní tabulky přidáním modulu do jádra OS.

Každá tabulka je připojená na některé záchytné body. Tabulka `filter` umožňuje používat záchytné body `INPUT`, `OUTPUT` a `FORWARD`. Při průchodu paketu systémem a zavolání modulu `ip_tables` v některém záchytném bodě jsou postupně procházeny připojené tabulky a v nich jednotlivé záznamy. Nejdříve je paket porovnán s položkou `ipt_entry.ip`, kde jsou uloženy informace o zdrojové a cílové adrese, dále o zdrojovém a cílovém síťovém zařízení a o protokolu vyšší vrstvy. Pokud paket této struktuře odpovídá, jsou postupně procházena jednotlivá pravidla (pokud existují). Pokud žádné z nich paket neodmítne, je provedena definovaná akce.

Poznámka
Průchod pravidly znamená volání funkce `ipt_entry_match.match()` pro každé pravidlo. Provedení akce potom analogicky odpovídá zavolání funkce `ipt_entry_target.target()`.

Informace o dostupných funkcích v použitém jádře lze vyčíst ze souborového systému `/proc`.

- `$ cat /proc/net/ip_tables_names` – vypíše seznam dostupných tabulek
- `$ cat /proc/net/ip_tables_matches` – vypíše seznam dostupných typů pravidel
- `$ cat /proc/net/ip_tables_targets` – vypíše seznam dostupných typů cílů

2.4.3 L7-filter pro iptables

Jedním z možných rozšíření systému *iptables* je modul L7-filter. Je distribuován ve formě záplat pro Linuxové jádro a pro aplikaci *iptables*. Umožňuje filtrovat pakety na základě jejich obsahu až do nejvyšší vrstvy `ISO OSI` pomocí regulárních výrazů. Z pohledu uživatele přibývá do rozhraní programu *iptables* nový typ pravidla s názvem `layer7`. Vyhledávané vzory musí být uloženy v souborech pojmenovaných podle protokolu, který je jimi vyhledáván. Příkaz

```
$ iptables -A INPUT -m layer7 --l7proto http --l7dir /etc/l7-protocols -j DROP
```

vytvoří pravidlo pro zahazování paketů přicházejících do systému a odpovídajících vzoru uloženém v souboru `/etc/l7-protocols/http.pat`. Regulární výrazy se zapisují podle syntaxe PCRE.

Z pohledu implementace přibyla do jádra Linuxu instance struktury `ipt_match` definující funkci, která provádí prohledání paketu, a také knihovna `regexp`. Rozšíření spolupracuje s modulem `nf_conntrack` zajišťující stavovost *iptables*, pokouší se prohledávat na úrovni toků a nikoliv pouze jednotlivých paketů.

3 Hardwarová platforma

3.1 Procesor MicroBlaze

Mikroprocesor MicroBlaze patří mezi procesory, které bývají označovány jako *soft*. Nejedná se o fyzický procesor, ale o soubor syntetizovatelných (předkompilovaných) zdrojových kódů v jazyce VHDL. Procesor je díky tomu konfigurovatelný, a je tak možné ovlivnit zejm. jeho výkonnost a velikost na čipu podle požadavků konkrétní aplikace. [14, s. 14]

Z hlediska architektury jde o 32bitový procesor typu RISC s *Harvardskou architekturou*. Lze volit, zda se procesor chová jako *Big Endian* nebo *Little Endian*. Jádro používá *zřetězené zpracování instrukcí* (pipelining), kde každý stupeň obvykle trvá jeden takt. Pokud některá instrukce vyžaduje v některém stupni více taktů, dojde k pozastavení procesoru (stall). [14, s. 14, 52, 54]

3.1.1 Popis registrů

MicroBlaze definuje 32 obecných registrů a skupinu speciálních registrů [14, s. 29–51]:

PC
Program Counter – adresa prováděné instrukce.

MSR
Machine Status Register – registr příznaků.

EAR, ***ESR***, ***BTR***, ***FSR***, ***EDR***
Registry s podrobnostmi o výjimkách.

TLBLO, ***TLBHI***, ***TLBX***, ***TLBSX***, ***ZPR***, ***PID***
Registry pro podporu MMU.

PVR0–PVR11
Processor Version Register – skupina registrů, která obsahuje informace o konfiguraci procesorového jádra, např. jeho verzi, jestli používá hardwarovou děličku, jednotku pro výpočty v plovoucí řádové čárce, zda byla syntetizována podpora pro MMU, apod.

Některé registry pro obecné použití (pojmenovány `R#`, kde `#` je číslo) mohou nabýt zvláštního významu. Registry `R14`–`R17` slouží mechanismu výjimek a přerušení k ukládání návratových adres. Zvláštní význam má také registr `R0`, který má vždy hodnotu 0 a nikdy nemění svůj obsah (vše do něj zapsané je zahozeno). Ostatní registry `R1`–`R13` a `R18`–`R31` lze používat libovolně. [14, s. 28]

3.1.2 Instrukční sada

MicroBlaze podporuje dva druhy instrukcí [14, s. 18]:

Type A
Instrukce obsahuje až dva zdrojové registrové operandy a jeden registr jako cílový operand.

Type B
Instrukce obsahuje jeden zdrojový registrový operand, 16bitový přímý operand (ten lze rozšířit na 32 bitů pomocí předcházející instrukce `imm`) a jeden cílový registrový operand.

Podle účelu lze instrukce rozdělit na aritmetické, logické, instrukce skoku, instrukce load/store a speciální instrukce. Některé instrukce jsou privilegované a lze je tedy provádět pouze v privilegovaném režimu (používané operačními systémy). Pro přechod do privilegovaného režimu lze použít instrukce `BRALID` nebo `PRKI`. Pro opuštění tohoto režimu existuje instrukce `RTED`. [14, s. 55]

3.1.3 Paměťová architektura

Jak bylo zmíněno, procesor používá *Hardvardskou architekturu*. Paměť je rozdělena na datovou a instrukční, každou adresovatelnou 32 bity. Procesor má tedy dostupný prostor až 4 GB jak pro program, tak pro data. Procesor umožňuje přístup do paměti po 8, 16 a 32 bitech, datové přístupy musí být zarovnané na hranici slova (32 bitů) nebo půlslova. [14, s. 54]

Pro oba adresové prostory lze nakonfigurovat cache pro přednačítání (*prefetch*) instrukcí/dat. Cache jsou realizovány pomocí XCL sběrnice a mají nastavitelnou velikost. [14, s. 54]

Adresový prostor může být tvořen několika paměťmi. Pro jejich připojení lze použít sběrnic LMB (pro připojení BlockRAM paměti přímo na čipu), PLB, XCL anebo AXI. Přístup přes LMB a úspěšný přístup do cache (*cache read hit*) po XCL trvají jeden nebo dva takty v závislosti na konfiguraci jádra MicroBlaze. [14, s. 54]

Na nejnižších adresách, tj. `0x0000--0x0024` jsou umístěny přerušovací vektory. Procesor definuje následující adresy [14, s. 66]:

- Reset,
- User Vector (vyvolané instrukcí `BRALID R#, 0x8`, viz *3.1.2 Instrukční sada*),
- Interrupt (přerušení od periferie),
- Break: Non-maskable hardware,
- Break: Hardware,
- Break: Software (vyvolané instrukcemi `BRK` nebo `BRKI`) a
- Hardware exception (výjimka, oznamuje, že došlo k chybě).

3.1.4 Parametrizace jádra

Díky tomu, že MicroBlaze je *soft* procesor, je možné jej parametrizovat a díky tomu snáze dosahovat požadovaných výkonnostních a prostorových požadavků aplikace. Následuje stručný přehled parametrů dostupných přímo v prostředí EDK:

- Urychlení operací posuvu: *Barrel shifter*.
- Urychlení celočíselného násobení: *Integer Multiplier*.
- Urychlení celočíselného dělení: *Integer Divider*.
- Zahrnutí instrukcí pro porovnávání obsahu registrů po bytech: *Pattern Comparator*.

- Zahrnutí instrukcí pro výpočty v plovoucí řádové čárce: *FPU*.
- Cache pro optimalizaci instrukcí skoku.
- Volitelná instrukční a datová cache, nastavení jejích velikostí.
- Použití jednotky MMU.
- Počet FSL linek a zahrnutí instrukcí pro komunikaci po FSL.
- Povolení registrů `PVR`.

3.1.5 Periferie a rozšiřitelnost

Procesor je dále rozšiřitelný pomocí sběrnic. Verze `8.00` disponuje sběrnicemi PLB a Fast Simplex Link (FSL), které jsou určeny pro připojení periferií a pro hardwarovou akceleraci výpočtů. Nově je v této verzi podporována sběrnice AXI. [14, s. 15–16]

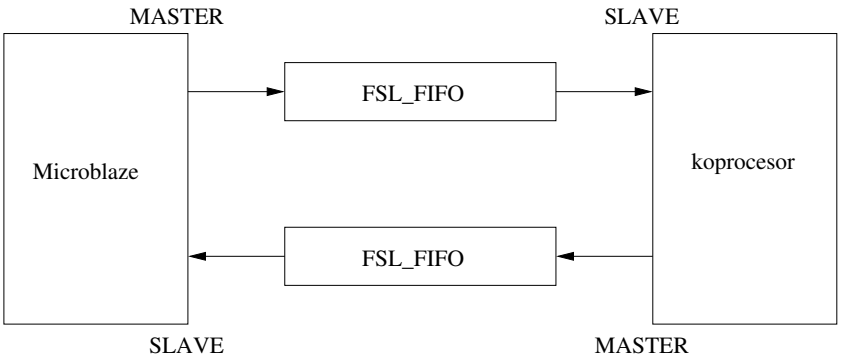
3.2 Sběrnice PLB 4.6

Připojení periferií k MicroBlaze se liší v průběhu vývoje tohoto procesoru. V dřívějších implementacích se používala sběrnice OPB, nyní je k dispozici sběrnice PLB 4.6 (dále jen PLB) a na čipech Spartan 6 a Virtex 6 je podporována sběrnice AXI, která by měla nahradit stávající PLB.

Sběrnice PLB je sdílenou sběrnicí obsahující *arbitr*, ke kterému lze připojit několik zařízení typu *master* a několik zařízení typu *slave*. Sběrnice podporuje též schéma *point-to-point*. Xilinx doporučuje používat PLB zjednodušeným způsobem za účelem snížení množství spotřebovaných zdrojů na čipu a snížení složitosti jednotlivých komponent. Např. pro potenciálně pomalejší periferie jako UART definuje datovou šířku 32 bitů a pro rychlejší periferie šířky až 64 a 128 bitů. Adresová sběrnice je 32bitová. [17, s. 5, 7]

3.3 Fast Simplex Link

Rozhraní *Fast Simplex Link* (dále jen FSL) umožňuje rozšiřovat procesor MicroBlaze pomocí dalších komponent v FPGA čipu, které bývají označovány jako *koprocesory* nebo *akcelerátory*. Instrukční sada procesoru obsahuje instrukce, pomocí kterých lze s těmito akceleračními jednotkami komunikovat. [7, s. 1; 14, s. 83]



Obr. 3.1: Schéma připojení akcelerátoru přes FSL

Jedná se o propojení *point-to-point* mezi dvěma komponentami. Skládá se ze dvou částí – *master* a *slave*, které jsou propojeny FSL frontou. Rozhraní *master* umožňuje zapsat data do fronty. Druhá strana – *slave* – tato data z fronty vybírá. [7, s. 6] Každé zařízení typicky implementuje jedno rozhraní *master* a jedno rozhraní *slave*, aby byl umožněn obousměrný přenos. Procesor nabízí až 16 FSL rozhraní (každé obsahuje *master* i *slave*). Ukázka připojení akcelerátoru k procesoru MicroBlaze je znázorněna ve schématu *3.1 Schéma připojení akcelerátoru přes FSL*.

Rozhraní nabízí kontrolní bit, který umožňuje rozlišit dva druhy dat. Např. lze takto odlišit datová a kontrolní slova nebo určit začátek a konec datového toku. [7, s. 1] Datová šířka přenášeného slova je obecně volitelná, procesor MicroBlaze běžně používá 32 bitů.

3.3.1 Popis instrukcí pro komunikaci přes FSL

Procesor MicroBlaze disponuje skupinou instrukcí `put` a `get`, které dovolují komunikovat po FSL. K oběma instrukcím existují modifikace, které mění jejich chování vzhledem k vykonávanému programu. Instrukce lze provádět v blokujícím i neblokujícím režimu a skupinu FSL instrukcí lze provádět atomicky nebo přerušitelně. [14, s. 170–173, 199–202]

Zápis do FSL. Instrukce `put` zapisuje data z pracovního registru procesoru do FSL fronty. Typické použití:

```
put R0, FSL0
```

zapíše data z registru `R0` do FSL číslo 0. Cílové FSL rozhraní je zakódováno přímo v instrukci. To může být nevýhodné, proto je k dispozici též dynamická verze instrukce `putd`. Typické použití:

```
putd R0, R1
```

zapíše data z registru `R0` do FSL daného registrem `R1`.

Čtení z FSL. Instrukce `get` čte data z FSL fronty do pracovního registru procesoru. Analogicky k instrukci `put` existují dvě varianty:

```
get  R0, FSL0
getd R0, R1
```

pro vyčtení dat z FSL číslo 0, resp. z FSL daného registrem `R1` do registru `R0`.

Práce s kontrolním bitem. FSL obsahuje v rozhraní kontrolní bit (jak bylo zmíněno v části *3.3 Fast Simplex Link*). Tento bit lze nastavovat modifikací instrukce `put`:

```
cput R0, FSL0 .
```

Instrukce `get` očekává, že příchozí data budou označena kontrolním bitem s hodnotou 0. V případě modifikované instrukce `cget` je očekávána hodnota bitu 1. Pokud není tato podmínka splněna (v obou případech), je nastaven bit `FSL_Error` v registru `MSR`. [14, s. 171, 173, 200, 202]

3.3.2 Popis signálů FSL

Typický akcelerátor definuje v rozhraní signály pro stranu *master* a pro *slave*. Obě poloviny mají svůj hodinový signál, což umožňuje provozovat FSL v asynchronním módu. V módu synchronním je použit hlavní hodinový signál (CLK). Souhrn signálů je uveden v tabulce *3.1: Přehled signálů rozhraní FSL*. [7, s. 1–3]

Pro připojení akcelerátoru k FSL frontě je třeba propojit signály akcelerátoru `FSL_M_*` na signály fronty `FSL_M_*`, `FSL_S_*` analogicky na `FSL_S_*`. [7, s. 1–3]

Akcelerátor vyčítá data ze vstupní fronty řídicím signálem `FSL_S_Read` v případě, že fronta nastaví bit `FSL_S_Exists` – signalizuje, že ve frontě jsou data. Obdobně akcelerátor zapisuje do výstupní fronty řídicím signálem `FSL_M_Write` v případě, že fronta nemá nastaven bit `FSL_M_Full` – fronta není plná. [7, s. 1–3]

Název	**Směr**	**Šířka**
Obecné:		
CLK	Input	1
RST	Input	1
Master:		
FSL_M_Clk	Input	1
FSL_M_Rst	Input	1
FSL_M_Full	Input	1
FSL_M_Write	Output	1
FSL_M_Data	Output	32
FSL_M_Control	Output	1
Slave:		
FSL_S_Clk	Input	1
FSL_S_Rst	Input	1
FSL_S_Exists	Input	1
FSL_S_Read	Output	1
FSL_S_Data	Input	1
FSL_S_Control	Input	1

Tab. 3.1:: Přehled signálů rozhraní FSL [7]

3.4 Periferie

K procesoru MicroBlaze lze přes již zmíněné sběrnice PLB, OPB, AXI (ale i další) připojovat perfierie. Následující část shrnuje periferie použité v této práci, způsob jejich připojení a případně další problematiku.

3.4.1 Multi-Port Memory Controller (`mpmc`)

Základní součástí téměř každého systému postaveného na procesoru MicroBlaze je řadič paměti `mpmc`. Verze `v6.02.a` může být až osmiportová, kde pro každý jeden port lze zvolit, po jaké sběrnici bude komunikovat. Na výběr jsou PLB, XCL (umožňuje připojit 2 XCL sběrnice na jeden port `mpmc`), LocalLink, NPI, ale i další, které nemají přímou souvislost s procesorem MicroBlaze. Podporovaná rozhraní označuje Xilinx souhrnně jako *Personality Interface Modules* (PIM). `mpmc` umožňuje pomocí PIM adresovat maximálně 2 GB paměti. [15, s. 54–55, 131]

Arbitrace. Transakce na jednom portu jsou obsluhovány v pořadí, v jakém byly iniciovány. Řadič umožňuje nastavit různé algoritmy arbitrace v rámci několika portů. Pořadí vykonání

transakcí při přístupu z několika portů není definováno, může docházet k situacím, kdy při souběžných operacích *čtení* a *zápis* (každá z jiného portu) bude vyčtena původní hodnota místo hodnoty zapisované. Dostupné algoritmy arbitrace [15, s. 61–62]:

Fixed
Každý port má pevně danou prioritu, Přednost má port 0, dále port 1, ...

Round Robin
Všechny porty mají stejnou základní prioritu. Průběžně při každé arbitraci se mění tzv. relativní priorita každého portu.

Custom
Umožňuje nastavit pro každý port parametry arbitrace.

Poznámka
FPGA Spartan 6 obsahuje `mpmc` jako *hard block* – není syntetizován, ale je přímo součástí čipu. [15, s. 122]

Komunikace po LocalLink sběrnici. LocalLink je používán pro provádění *DMA* přenosů. Skládá se ze dvou kanálů, jeden pro příjem a druhý pro odesílání dat. Oba kanály jsou vzájemně nezávislé. Každý kanál má k dispozici přerušovací signál. Data jsou přenášena po *paketech*. Každý paket může obsahovat data z různých částí paměti, což lze využít pro zvýšení výkonnosti. [15, s. 136–137]

Název	Směr	Šířka
InitDone	Input	1
Addr	Output	32
AddrReq	Output	1
AddrAck	Input	1
RNW	Output	1
Size	Output	4
RdModWr	Output	1
WrFIFO_Data	Output	32/64
WrFIFO_BE	Output	4/8
WrFIFO_Push	Output	1
WrFIFO_Empty	Input	1
WrFIFO_AlmostFull	Input	1
WrFIFO_Flush	Output	1
RdFIFO_Data	Input	32/64
RdFIFO_Pop	Output	1
RdFIFO_RdWdAddr	Input	4
RdFIFO_Empty	Input	1
RdFIFO_Flush	Output	1
RdFIFO_Latency	Input	2

Tab. 3.2:: Přehled signálů rozhraní NPI z hlediska periferie [15, s. 30–33]

Rozhraní NPI. Dalším rozhraním, kterým lze s pamětí komunikovat, je NPI. Jedná se o nejuniverzálnější rozhraní pro přístup do paměti a Xilinx jej uvádí jako možnost pro přidání podpory dalších protokolů. Umožňuje používat datové šířky 32 a 64 bitů. [15, s. 184]

Ve skutečnosti (alespoň nahlédnutím do zdrojových kódů) jsou i ostatní protokoly PIM realizovány jako adaptéry pro protokol NPI. Stručný přehled signálů NPI uvádím v tabulce *3.2: Přehled signálů rozhraní NPI z hlediska periferie* [15, s. 30–33]. Časové diagramy lze nalézt v dokumentu [15] na stranách 189–201.

Rozhraní NPI nabízí několik režimů čtení z paměti. Při spuštění požadavku řídicím bitem `NPI_AddrReq` je režim specifikován vektorem `NPI_Size`. Jedná se o [15, s. 30]:

- přenos jednoho slova,
- přenos čtyř slov seřazených pro cache,
- přenos osmi slov seřazených pro cache,
- *burst* přenos 16 slov,
- *burst* přenos 32 slov a
- *burst* přenos 64 slov.

NPI transakce musí používat adresu zarovnanou na velikost celé této transakce. [15, s. 66, 186]

3.4.2 XPS LL TEMAC (`xps_ll_temac`)

Jedná se o *Ethernet Media Access Controller*, tedy o jednotku zajišťující připojení k *Ethernetové* počítačové síti. Podporuje rychlosti: 10 Mb/s, 100 Mb/s i 1000 Mb/s. Periferie je obsažena jako *hard block* v FPGA čipech Virtex 6, Virtex 5 a Virtex 4, pro ostatní podporované čipy je k dispozici *soft* varianta. [9, s. 2]

> **Poznámka**
> *Soft* varianta vyžaduje zakoupení licence, jinak pracuje ve zkušebním režimu. V takovém případě pro svoji funkčnost vyžaduje, aby bylo po několika hodinách zařízení resetováno. [9, s. 2]

Poskytuje konfigurovatelnou vstupní i výstupní frontu pro *Ethernetové* rámce, umožňuje pouze *Full-Duplex* režim, volitelnou podporu pro *jambo frames* a některá další nastavení, která umožňují odlehčit práci softwaru. [9, s. 1]

`xps_ll_temac` lze připojit sběrnicí PLB k procesoru MicroBlaze pro přístup k registrům periferie. Pro přenos *Ethernetových* rámců je k dispozici rozhraní LocalLink a vestavěná podpora *DMA*. Typicky se jednotka připojuje k řadiči `mpmc`. [9, s. 3]

> **Poznámka**
> K dispozici je také jednotka XPS LL FIFO, která umožňuje provádět paměťově mapovaný přístup k LocalLink sběrnici, čímž lze komunikovat s `xps_ll_temac` bez použití *DMA*.

Podrobnější informace lze získat v dokumentech [18], [19], [20] a [11] v závislosti na použitém čipu FPGA. [9, s. 3]

3.4.3 Ostatní běžně používané periferie

EDK obsahuje velké množství předpřipravených periferií. V následujícím textu jsou stručně popsány ty nejobecnější a nejčastěji používané.

MicroBlaze Debug Module (`mdm`)
Ladění aplikací v procesoru MicroBlaze je umožněno modulem `mdm`. Ten zajišťuje komunikaci přes rozhraní JTAG s ladicím softwarem (typicky na počítači typu PC) a s několika (verze `v2.00.a` podporuje až osm) procesory MicroBlaze verze 7 a vyšší. Jako ladicí software lze využít program XMD od firmy Xilinx. Modul lze propojit se systémem ChipScope. [12, s. 1]

XPS Interrupt Controller (`xps_intc`)
Tato jednotka spravuje příchozí přerušení od jiných periferií a oznamuje tyto události procesoru. Procesor poté může z registrů (po sběrnici PLB) `xps_intc` rozpoznat, o jaké přerušení se jedná, zareagovat a explicitně potvrdit, že bylo přerušení přijato. [8, s. 1]

Počet přerušujících signálů lze nastavit, maximálně je podporováno 32 samostatných přerušení. Jednotky `xps_intc` lze zapojovat do kaskád a zvýšit tak počet přerušujících signálů. Priorita jednotlivých přerušení je dána pozicí v přerušovacím vektoru. K dispozici je registr pro maskování jednotlivých přerušení. Lze také nastavovat způsob detekce přerušení u jednotlivých linek, tedy zda má být přerušení detekováno na vzestupnou nebo sestupnou hranu signálu, popř. jeho aktivní úroveň. [8, s. 1]

XPS Timer/Counter (`xps_timer`)
Důležitou součástí každého procesorového systému je časovač. Pro procesor MicroBlaze existuje periferie `xps_timer` připojitelná přes sběrnici PLB. Čítače mohou být volitelně dva (vyžadováno Linuxovým jádrem), oba o stejné šířce až 32 bitů. Pro každý lze definovat počáteční hodnotu pro *generování událostí* (*event generation*), *záchyt hrany* (*event capture*). Jednotka generuje jeden přerušovací signál pro oba čítače dohromady. Rozlišit je lze vyčtením *kontrolního slova*. `xps_timer` také umožňuje provádět *pulsně šířkovou modulaci* (*PWM*) při použití obou čítačů. [10, s. 1–2]

XPS General Purpose Input/Output (`xps_gpio`)
Periferie `xps_gpio` poskytuje univerzální (*general purpose*) vstupně výstupní porty přístupné z PLB sběrnice. Porty mohou být seskupeny do jednoho nebo dvou kanálů. Každý port může být nastaven jako vstupní nebo výstupní. Každý kanál lze nakonfigurovat pro generování přerušení při změně hladiny na vstupu. Modul je možné používat např. pro ovládání LCD displeje, detekci stisknutého tlačítka apod. [21, s. 1–2]

Komunikace po RS-232
V EDK jsou k dispozici dvě periferie pro komunikaci po *RS-232*. Jedná se o `xps_uartlite` a o `xps_uart16550`. Obě komunikují po PLB sběrnici. První uvedená je nastavitelná před syntézou, později nelze její vlastnosti měnit. Díky tomu je možné ušetřit zdroje na čipu. Druhá jednotka je kompatibilní s *UART* standardy 16450 a 16550, lze tedy konfigurovat parametry přenosu za běhu. [23, s. 1–2; 22, s. 1–2]

4 Vývoj aplikací pro MicroBlaze

Vývojové prostředí EDK disponuje generickým systémem pro tvorbu aplikací postavených okolo MicroBlaze, ale lze jej uchopit i obecněji – jako nadstavbu nad nástroji nižší úrovně. Těmi jsou programy provádějící syntézu zdrojových kódů v některém HDL (Hardware Description Language) jazyce. Každý EDK projekt sestává z množství konfiguračních souborů, které popisují jednotlivé součásti aplikace deklarativně i procedurálně. [16]

Deklarativní složky mají za úkol určit, z čeho se aplikace skládá a jak jsou jednotlivé komponenty propojeny (z hlediska hardwaru se jedná o sběrnice, z hlediska softwaru o knihovny a ovladače periferií). Setkáme se zde se soubory MPD, MLD, MDD, BBD, XBD, ale i dalšími. [16]

Procedurální složky, typicky sestávající z TCL skriptů, potom umožňují vygenerovat pro konkrétní aplikaci správné hlavičkové soubory nebo VHDL soubory realizující instanciaci periferií a procesorů propojených sběrnicemi. Výsledná podoba systému (s procesorem MicroBlaze) je automaticky generována vývojovými nástroji zejm. podle specifikací MHS a MSS (deklarativní popis). [16]

Firma Xilinx označuje tyto konfigurační soubory souhrnně jako *Platform Specification Format* (PSF). [16, s. 15]

Pro překlad EDK projektu se používají zejm. programy `platgen` a `libgen`, kdy první zpracovává hardwarovou část aplikace (generuje VHDL soubory, provádí syntézu apod.) a druhý část softwarovou. Grafické prostředí potom zajišťuje editaci některých projektových souborů a správné volání těchto programů.

4.1 Microprocessor Hardware Specification (MHS)

Popisuje systém z hlediska hardwaru. Tato specifikace je k dispozici ve formě souboru s příponou `.mhs`. Obsahuje [16, s. 23]:

- architekturu sběrnic,
- popis a konfiguraci periferií,
- nastavení procesoru (nebo procesorů),
- popis adresového prostoru.

Ve výpisu *4.1 Zkrácená ukázka MHS souboru* na straně 23 jsou instanciovány tři hardwarové komponenty. Jedná se o procesor MicroBlaze (2), sběrnici PLB (3) a periferii zajišťující seriovou komunikaci (4). Každá instance je pojmenovaná. Z výpisu lze vyčíst, že procesor `microblaze_0` používá PLB sběrnici `mb_plb`. K této sběrnici je připojena periferie `RS232_DCE` a je přístupná na adresách `0x84000000`–`0x8400ffff`.

Kromě připojení na sběrnici jsou zde uvedeny také některé parametry. Jedná se o rychlost seriové komunikace `C_BAUDRATE` (115200 Baud) u `RS232_DCE` a o `C_USE_BARREL` u procesoru `microblaze_0`. Tento parametr nařídí nástroji, který provádí syntézu, aby v ALU procesoru použil *barrel shifter* pro instrukce bitového posuvu.

```
...
BEGIN microblaze                                                    (2)
 PARAMETER INSTANCE = microblaze_0
 PARAMETER C_USE_BARREL = 1
 BUS_INTERFACE DPLB = mb_plb
 ...
END
BEGIN plb_v46                                                       (3)
 PARAMETER INSTANCE = mb_plb
 ...
END
BEGIN xps_uartlite                                                  (4)
 PARAMETER INSTANCE = RS232_DCE
 PARAMETER C_BAUDRATE = 115200
 PARAMETER HW_VER = 1.01.a
 ...
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
END
...
```

Výpis 4.1: Zkrácená ukázka MHS souboru

```
...
BEGIN OS                                                            (5)
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 3.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_DCE
 PARAMETER STDOUT = RS232_DCE
END
BEGIN PROCESSOR                                                     (6)
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.13.a
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
END
...
BEGIN DRIVER                                                        (7)
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = RS232_DCE
END
```

Výpis 4.2: Zkrácená ukázka MSS souboru

4.2 Microprocesor Software Specification (MSS)

Popisuje systém z pohledu softwaru. Reprezentuje jej soubor s příponou `.mss`. Nalezneme zde většinou [16, s. 97]:

- popis potřebných knihoven,
- nastavení ovladačů a jejich propojení s periferiemi,
- specifikaci operačního systému a jeho nastavení,
- nastavení přerušení,
- parametry překladače.

Ve výpisu *4.2 Zkrácená ukázka MSS souboru* nalezneme popis použitého OS, určení architektury, pro kterou bude prováděn překlad, a deklaraci ovladače seriového portu. Typ OS `standalone` (5) v prostředí EDK označuje aplikace bez OS. Toto nastavení instruuje překladový systém o zahrnutí `libc` knihovny a rozšiřujících knihoven od firmy Xilinx.

Nastavení v místě (6) popisuje, pro jakou architekturu bude prováděn překlad. Označení `microblaze_0` odkazuje do souboru MHS. Všechen software, který bude překládán pro tento procesor, použije zde uvedený překladač.

Deklarace v místě (7) popisuje, jaký použít ovladač pro komunikaci s periferií `RS232_DCE` (která byla uvedena v MHS souboru).

4.3 Další součásti PSF

Soubory MHS a MSS slouží pro popis celkového systému. Ten je složen z menších součástí, kdy každá má svůj definiční soubor. Pro popis jednotlivých rozšiřujících periferií, softwarových knihoven a ovladačů a vývojových desek slouží soubory MPD, MLD, MDD a XBD.

Microprocesor Peripheral Definition (MPD)
Definuje rozhraní periferií, tzn. dostupné porty, sběrnice. Deklaruje parametry, kterými lze periferii nastavit. Dále může obsahovat metadata, jako např. pro jaké FPGA je periferie určena, její verzi, ale také klauzule ověřující, zda je periferie vhodně připojena ke zbytku aplikace. [16, s. 45]

Každý soubor MPD popisuje jednu periferii. Periferie musí být pojmenována podle konvencí definovaných firmou Xilinx. Sestává z názvu periferie a její verze. Podle tohoto údaje jsou periferie odkazovány ze souboru MHS.

V ukázce *4.1 Zkrácená ukázka MHS souboru* byla použita periferie `xps_uartlite`. Tuto periferii lze nalézt v adresářové struktuře instalace EDK. Ve verzi EDK 12.4 je k dispozici v:

`$EDK/hw/XilinxProcessorIPLib/pcores/xps_uartlite_v1_01_a` .

Proměnná `$EDK` označuje adresář s instalací vývojového prostředí EDK. Na tomto místě nalezneme i MPD soubor k této komponentě.

Microprocesor Library Definition (MLD)
Obsahuje direktivy pro přizpůsobení knihoven, pro generování BSP (Board Support Package) a pro nastavení operačních systémů. K souboru MLD se váže TCL skript, který přizpůsobuje danou knihovnu nebo OS podle nastavených parametrů. [16, s. 107–108]

Jako příklad lze uvést generátor souborů *DTS* pro překlad Linuxového jádra. Více o této problematice viz část *2.3.2 Kompilace Linuxového jádra*. Podobně ke komerčním distribucím Linuxu lze v prostředí EDK najít předpřipravené generátory BSP.

Microprocesor Driver Definition (MDD)
Popisuje, jak lze nastavit konkrétní softwarový ovladač periferie. Se souborem dále souvisí i TCL skript, který přizpůsobuje ovladač podle parametrů nastavených v souboru MSS při jeho instanciaci. Skript obvykle generuje hlavičkové soubory, soubory se zdrojovými kódy a provádí kontrolu platnosti nastavení. [16, s. 121–122]

Xilinx Board Description (XBD)
Specifikace XBD umožňuje popsat vývojovou desku. Obsahuje popis jednotlivých fyzických součástek a jejich propojení s FPGA. Ze specifikace XBD lze vygenerovat UCF soubor (popisuje, jak implementovat logický systém v konkrétním fyzickém čipu, např. rozmístění pinů [6, s. 34–35]) pro správné namapování VHDL entit (resp. architektur) na piny FPGA při syntéze. [16, s. 131]

```
test_v1_00_a/
  + data/
    + test_v2_1_0.mpd
    + test_v2_1_0.pao
   (+ test_v2_1_0.bbd)
   (+ test_v2_1_0.tcl)
  + doc/
    +  dokumentace periferie
  + hdl/
    + verilog/
      +  soubory jazyka Verilog
    + vhdl/
      +  soubory jazyka VHDL
  + netlist/
    + NGC soubory
    + EDIF soubory
```

Výpis 4.3: Základní adresářová struktura periferie

4.4 Tvorba vlastní periferie pomocí PSF

Pro rozšíření funkcionality procesoru MicroBlaze o novou periferii je třeba vytvořit kromě vlastní hardwarové jednotky další soubory s metadaty (inspirovat se lze zejm. z hotových jednotek dostupných v EDK). Základem je soubor MPD.

Překladový systém potřebuje mít k dispozici seznam zdrojových souborů a pořadí, v jakém je má překládat. K tomuto účelu slouží soubor PAO (Peripheral Analyze Order). [16, s. 91]

Při použití komponent ve formě netlistů (vygenerovaných např. nástrojem `coregen`) ve formátech EDIF nebo NGC je třeba vytvořit soubor BBD (Black Box Definition) s jejich seznamem a v souboru MPD nastavit

```
OPTION STYLE = MIX .
```

BBD soubor je vyhledávací tabulka, která umožňuje, aby jednotlivé netlist soubory mohly být zahrnuty na základě hodnoty některého parametru periferie (nastaveného v souboru MHS). Takto lze např. použít pro různé FPGA čipy vhodné komponenty. [16, s. 95–96]

Jak zmiňuji v úvodu kapitoly *4 Vývoj aplikací pro MicroBlaze*, součástí PSF je také procedurální složka v podobě TCL skriptů. V TCL souboru lze pro každou periferii definovat funkce, které jsou volány v určitých fázích překladu. V souboru MHS lze specifikovat názvy těchto funkcí a navázat je na překladový systém. Např. pro generování VHDL souboru TCL funkcí pojmenovanou `generate_hdl` při syntéze stačí v MPD souboru specifikovat [16, s. 50–54]

```
OPTION ELABORATE_PROC = generate_vhdl .
```

Pro myšlenou periferii `test` verze `1.00.a` bude základní adresářová struktura vypadat přibližně jako ve výpisu *4.3 Základní adresářová struktura periferie* na straně 25.

4.5 Projekt v EDK a jeho součásti

Každý projekt v prostředí EDK sestává zejm. z následujících souborů:

system.xmp
Soubor popisující projekt; určuje cesty k souborům MSS a MHS, které mají být použity pro překlad; ukládá metadata k softwarovým projektům v prostředí.

system.mhs
Soubor popsaný v části *4.1 Microprocessor Hardware Specification (MHS)*.

system.mss
Soubor popsaný v části *4.2 Microprocesor Software Specification (MSS)*.

system.bsb
Binární soubor grafického průvodce *Base System Builder*.

pcores/
Adresář, kam je vhodné ukládat vlastní hardwarové jednotky. EDK toto místo automaticky prohledává a periferie zde nalezené nabídne pro vložení do projektu.

etc/
Do tohoto adresáře ukládá prostředí pomocné soubory, např. skripty pro spouštění překladových nástrojů.

data/
Adresář, který typicky obsahuje vygenerovaný UCF soubor.

Poznámka
Pro automatizaci překladu dávkovým způsobem je velmi vhodné zachovávat též soubory `system.make` a `system_incl.make`, které generuje prostředí EDK vždy při otevření projektu.

Tyto součásti definují vývojovou platformu nad procesorem MicroBlaze. Pomocí nástrojů `libgen` a `platgen` je nyní možné vygenerovat vše potřebné pro vývoj aplikací, např. hlavičkové soubory nebo VHDL soubory.

4.6 Tvorba aplikací v EDK

Posledně zmíněné soubory tvoří v prostředí EDK základnu pro tvorbu aplikací pro MicroBlaze. Je možné vytvářet uživatelské definice pro vlastní knihovny a hardwarové jednotky připojitelné k MicroBlaze.

Aplikace vyvíjené pro platformu MicroBlaze lze rozdělit do několika kategorií:

standalone
Aplikace napsaná „na zelené louce“ na míru požadavkům a hardwarovému vybavení. Může těžit z možnosti využít procesor na maximum.

xilkernel
Jedná se o vrstvu, která aplikaci poskytuje podmnožinu služeb rozhraní *POSIX*. Můžeme ji považovat za odlehčené jádro operačního systému. Je vyvíjena firmou Xilinx a nabízena jako součást vývojového prostředí EDK.

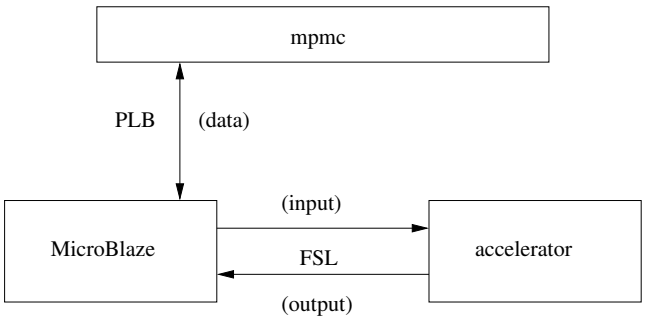
s operačním systémem
Jako základ aplikace můžeme použít některý operační systém upravený pro běh v embedded prostředí. Jako příklad lze uvést operační systémy Linux nebo FreeRTOS. Exituje velké množství operačních systémů pro vestavěné aplikace, ale bývají zpravidla poskytovány komerčně.

Tyto kategorie je třeba zohlednit v MSS. Standalone a xilkernel aplikace jsou podporovány vývojovým prostředím EDK, obsahuje pro ně potřebné definiční soubory. EDK takto dále podporuje některé komerční distribuce Linuxu.

4.7 Možnosti hardwarové akcelerace na MicroBlaze

V částech *3.1 Procesor MicroBlaze*, *3.3 Fast Simplex Link* a *3.4 Periferie* je popsána nejpodstanější část hardwarové platformy okolo procesoru MicroBlaze a také to, že procesor je navržen s možností urychlovat některé algoritmy prováděné v softwaru vhodným převedením do podoby hardwarových obvodů. K tomuto účelu může nepochybně sloužit sběrnice FSL.

V následujícím textu se pokusím nastínit obecné případy, kdy je vhodná část softwarového kódu nahrazena hardwarovou komponentou. Za zásadní považuji analyzovat zvolený algoritmus z hlediska vstupů a výstupů. To je rozhodující zejm. z toho důvodu, že pro efektivní zpracování síťového provozu je třeba zpracovat co největší množství dat v omezeném čase.



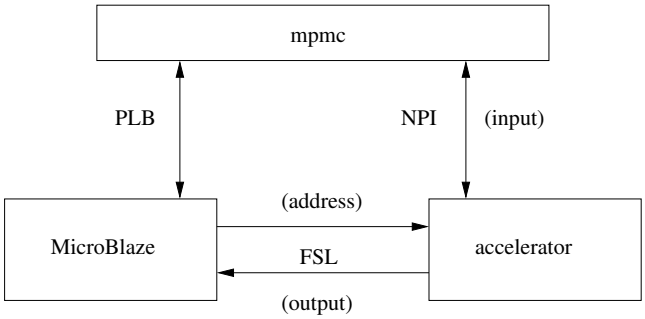
Obr. 4.1: Schéma připojení akcelerátoru při malém množství vstupů i výstupů

4.7.1 Urychlení algoritmů s malým množstvím vstupů i výstupů

Pokud *kód* vyžaduje pouze omezené množství vstupů a výstupů, je vhodnou variantou použití FSL pro jejich přenos mezi procesorem a akcelerátorem (analogie: předávání parametrů hodnotou). Lze dosáhnout nízké režie tohoto přenosu. Klasifikace paketů podle `IPv4` adres a podle portů je vhodným příkladem. O této možnosti se také zmiňuje dokument [14] na straně 83. Ilustrace této situace je ukázána na obrázku *4.1 Schéma připojení akcelerátoru při malém množství vstupů i výstupů* na straně 27.

4.7.2 Urychlení algoritmů s velkým množstvím vstupů a malým množstvím výstupů

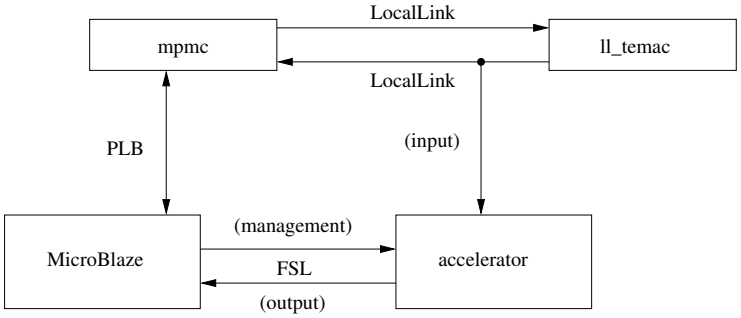
V případě, že je třeba, aby algoritmus pracoval s větším úsekem paměti, začne se projevovat režie FSL, protože je pro každé datové slovo třeba přistoupit do paměti. Kromě toho je posílání dat pouze a jenom v režii procesoru a akcelerátor nemá možnost požádat o konkretní úsek dat nebo dokonce některý úsek přeskočit. Přesněji, bylo by třeba implementovat komunikační protokol po FSL, čímž by se zvýšila režie procesoru.



Obr. 4.2: Schéma připojení akcelerátoru při velkém množství vstupů

Pro tento případ existuje efektivnější řešení. Akcelerátor může být připojen k paměti přímo vhodnou sběrnicí. Jak bylo uvedeno v části *3.4.1 Multi-Port Memory Controller (mpmc)*, řadič paměti dokáže obsloužit až 8 nezávislých spojení. V takovém případě stačí přes FSL předávat akcelerátoru adresy (analogie: předávání parametrů odkazem).

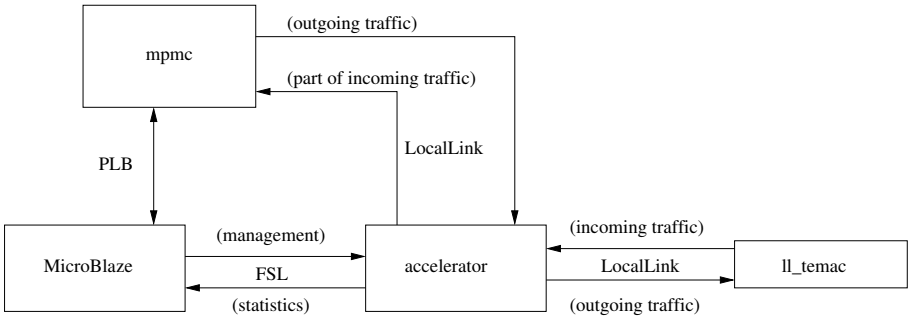
Příkladem algoritmu, který může využít tento princip, je *pattern match*. Tímto způsobem je také možné urychlit extrakci potřebných údajů pro filtrování z paketu.



Obr. 4.3: Schéma připojení akcelerátoru k `xps_ll_temac`

Řešení specifické pro síťové přenosy. Při použití komponenty `xps_ll_temac` lze využít toho, že jsou data posílána po rozhraní LocalLink formou *DMA*. Přicházející pakety mohou být takto předány akcelerátoru přímo a softwarová aplikace si pouze vyzvedne výsledek, viz schéma *4.3 Schéma připojení akcelerátoru k xps_ll_temac* na straně 28. Toto řešení však může být vzhledem k nutnosti dekódovat *DMA* přenosy technicky náročnější. Je nutné zajistit, aby přicházející pakety do akcelerátoru byly správně spárovány s pakety, které zpracovává software, a v případě, kdy jedna strana paket zahodí, musí druhou o této události vhodně informovat. Kromě toho není tímto způsobem zpracováván provoz odchozí.

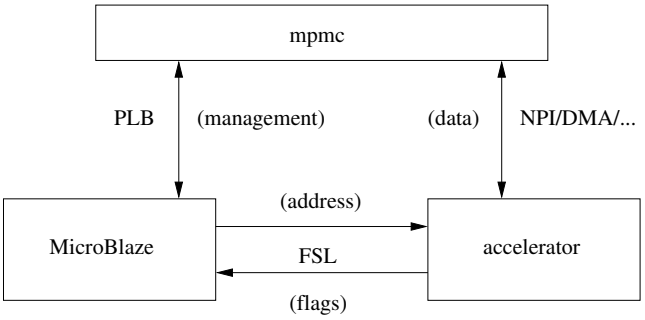
Podobně je možné přesunout značnou část filtrování do hardwaru a do procesoru propouštět jen omezené množství provozu, jak ukazuje schéma *4.4 Schéma zapojení akcelerátoru do cesty veškerému provozu*. V tomto případě jednotka pracuje více méně autonomně, je konfigurována přes FSL a stejnou cestou poskytuje statistiky provozu.



Obr. 4.4: Schéma zapojení akcelerátoru do cesty veškerému provozu

4.7.3 Urychlení algoritmů s velkým množstvím vstupů i výstupů

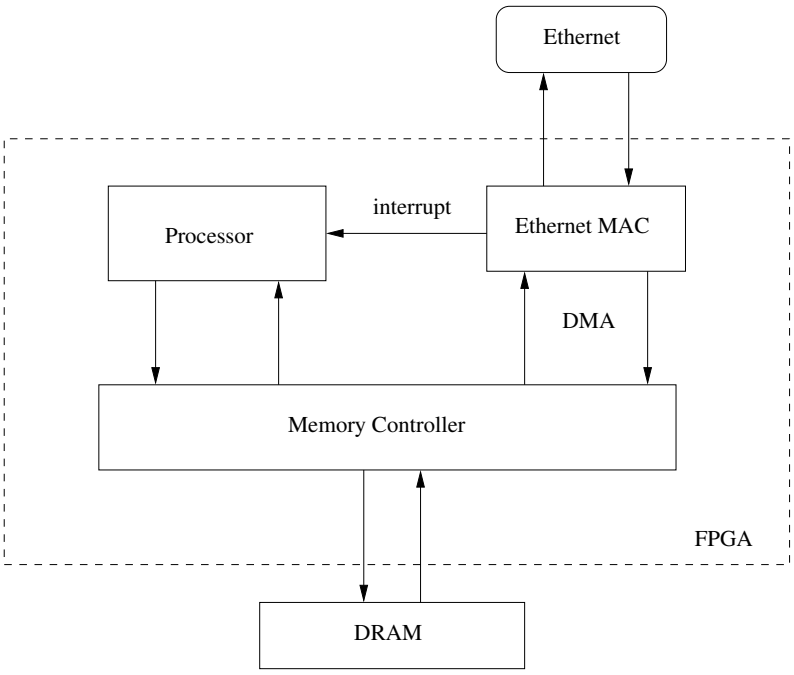
Akcelerátor může z paměti číst a může do ní též zapisovat. Pokud je nutné provést např. transformaci vstupních dat na jiná data, lze vyhradit v paměti prostor (např. v softwarové části aplikace, voláním typu `malloc`), do kterého jednotka umístí výstup. Využít lze nejspíše sběrnice NPI nebo LocalLink (*DMA*). Tento typ akcelerátoru lze podobně jako předchozí variantu označit také jako autonomně pracující.



Obr. 4.5: Schéma zapojení akcelerátoru, který ukládá výsledky zpět do paměti

5 Návrh a implementace aplikace

Obrázek *5.1 Návrh struktury hardwarové platformy* zobrazuje schéma hardwarové platformy aplikace pro filtrování provozu. Příchozí pakety jsou ukládány do paměti pomocí DMA přenosů. Procesor je o příchozím paketu informován přerušením. Pakety jsou zpracovány procesorem, který rozhodne, zda mají být přeposlány dalšímu zařízení v síti, anebo zahozeny.



Obr. 5.1: Návrh struktury hardwarové platformy

5.1 Popis zpracování paketu

S každým přijatým paketem je provedeno několik operací, které ovlivní výkonnost aplikace. Propustnost budou typicky snižovat ty operace, které zpracovávají obsah paketu, což znamená vysoký počet přístupů do paměti. Průchod paketu systémem lze shrnout do několika bodů:

1. Příjem paketu.
2. Kontrola délky.
3. Kontrola verze IP protokolu.
4. Ověření kontrolního součtu.
5. Směrování paketu.
6. Aplikace filtrovacích pravidel.

    a. Kontrola metadat spojených s paketem.
    b. Hledání vzorů v obsahu paketu.

7. Doručení paketu podle výsledku směrování.

Lze předpokládat, že nejpomalejšími budou operace *4*, kde je třeba provést kontrolu celého paketu, dále *5*, kde dochází ke spuštění algoritmu *Longest Prefix Match* pro určení další trasy paketu v síti, a nakonec *6* v závislosti na složitosti nastavených pravidel.

5.2 Distribuce OS Linux

Použití Linuxu má několik výhod oproti vytvoření aplikace na „zelené louce“. Jádro operačního systému obsahuje:

- hotový TCP/IP stack podporující velké množství běžně používaných protokolů jako `TCP`, `UDP`, `ICMP`, `IPv4`, stále více se rozšiřující `IPv6` a mnoho dalších,
- mechanismus pro zpracovávání síťového provozu *netfilter*,
- podporu víceúlohového zpracování (*multitasking*)
- a také aplikační rozhraní[6].

V oblasti bezpečnosti, resp. bezpečnosti sítí, kam aplikace *filtr síťového provozu* nepochybně patří, jsou kladeny vysoké nároky na bezchybnost a spolehlivost softwaru. Je proto výhodnější použít už hotové a prověřené programy, případně programy, které jsou průběžně záplatovány.

Implementace aplikace bez OS má proti zmíněným možnostem výhodu v potenciálně vyšší výkonnosti, protože nutně neobsahuje změny kontextu aplikace-jádro a může být lépe optimalizovaná vzhledem k architektuře procesoru.

OS Linux je typicky šířen v podobě distribucí, které v sobě zahrnují jednak jádro OS, ale zejm. uživatelské programy, které dovolují celou aplikaci spravovat, případně samy provádějí potřebnou funkčnost. V rámci této práce byla vytvořena distribuce pokrývající konfiguraci frameworku *netfilter*, profilování a měření propustnosti aplikace. Distribuce je dále rozšiřitelná i pro jiné aplikace na procesoru MicroBlaze.

5.2.1 Zpracování paketu v OS Linux

Jednotlivé fáze zpracování lze nalézt v jádru OS Linux v podobě několika funkcí, které budou sledovány při profilování. Příjem paketu provádí funkce `net_rx_action` volaná z `do_softirq`, která obsluhuje hardwarová přerušení.

Kontrola paketu probíhá ve funkcí `ip_rcv`, odkud je také zavolána funkce `do_csum`, která počítá kontrolní součet IP hlavičky. Kromě toho dochází ke směrování paketu, a tedy určení jeho další trasy.

Další zpracování znamená přeposlání paketu na jinou stanici v síti, nebo doručení paketu lokálně, které už je v roli vyšších vrstev. Na úrovni transportní vrstvy jsou počítány kontrolní součty segmentů, které se mohou při profilování projevit režií funkce `do_csum`.

5.3 Implementace hardwarové platformy

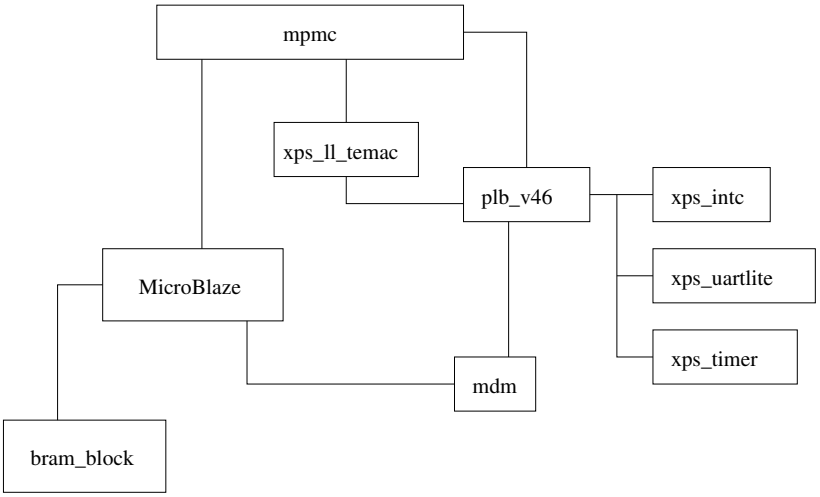
Aplikace byla realizována na vývojové desce `Spartan-3E SP3E16`, která je osazená FPGA čipem `xc3e1600e` Spartan 3E, pamětí DDR SDRAM `MT46V32M16` o velikosti 512 Mbit, čipem `SMSC`

[6] API, které je využíváno uživatelskými programy (typicky skrze knihovnu *libc*)

`LAN83C185 10/100 Ethernet PHY` a konektorem RJ-45 pro přístup k sítím typu Ethernet, konektorem RS-232, USB konektorem pro nahrávání konfigurace FPGA a pro inicializaci obsahu paměti a dalšími komponentami, které byly popsány v části *3.4 Periferie* nebo nebyly využity. Popis uvedené desky lze nalézt v dokumentu [13].

5.3.1 Popis vytvořeného designu

Konfigurační řetězec FPGA (dále zjednodušeně `design`) sestává ze součástí uvedených ve schématu *5.2 Implementovaný design*. Každá z použitých jednotek je stručně popsána v sekci *3.4 Periferie*. Podrobnější informace lze nalézt v přiloženém souboru `system.mhs`.



Obr. 5.2: Implementovaný design

Komunikace s designem. S designem lze komunikovat pomocí seriového portu typu `DCE`. Konkrétní zapojení konektoru DB9 lze nalézt v dokumentu [13] na straně 60. Další parametry pro komunikaci:

- Rychlost: 115200 Bd,
- Parita: žádná,
- Počet stop bitů: 1,
- Počet datových bitů: 8.

5.4 Měření implementovaného systému

Pro potvrzení hypotéz z části *5.1 Popis zpracování paketu* je třeba provést měření. Bude provedeno měření propustnosti a také profilování jádra pro nalezení nejvytíženějších funkcí.

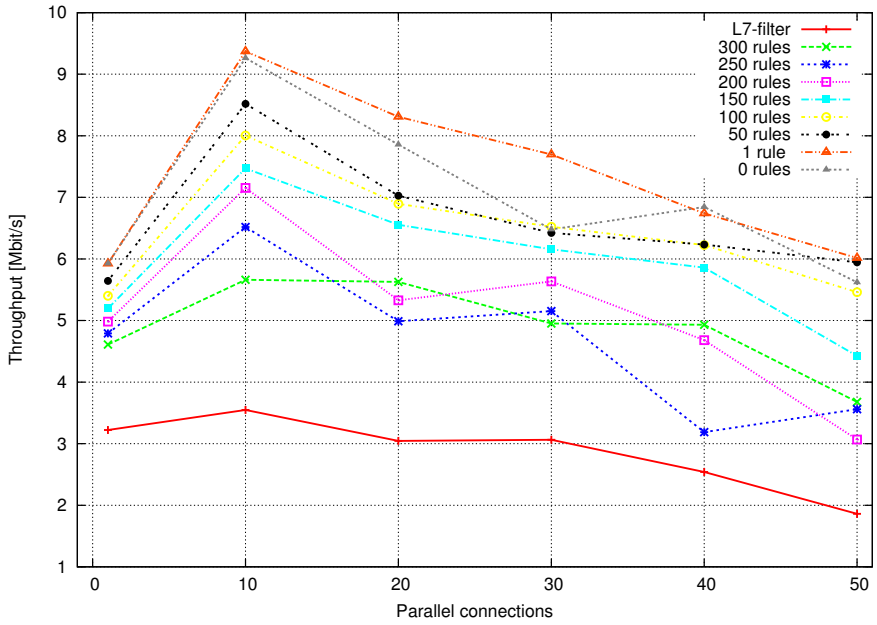
5.4.1 Použité nástroje

Jádro OS Linux dovoluje provádět profilaci pomocí vhodného nastavení při překladu. Povolením profilace a zadáním parametru `profile=2` při bootování se v běžícím systému zpřístupní

profilovací paměť `/proc/profile`, kterou lze číst, anebo nulovat. Pro interpretaci dat uložených v tomto bufferu je používán program `readprofile`.
Měření propustnosti. Program `iperf` měří propustnost mezi dvěma počítačovými systémy. Na jedné straně je spuštěn *server* a na druhé *klient*, kde lze nastavit parametry měření, jako délku měření, počet paralelně běžících přenosů, který protokol transportní vrstvy použít atp. Klient po spuštění odesílá data na server. Na závěr obě strany vypíší dosaženou průměrnou přenosovou rychlost a množství přenesených dat.

5.4.2 Měření propustnosti modulu `ip_tables`

Modul `ip_tables` bude při měření postupně plněn pravidly tak, aby každé z nich bylo aplikováno na každý přijatý paket, aniž by jej systém zahodil. Měření bude prováděno nejdříve pomocí jednoho vlákna, ale později také pomocí několika paralelně běžících vláken provádějících přenosy. Každé měření bude provedeno desetkrát a výsledná propustnost bude aritmetickým průměrem všech měření.



Obr. 5.3: Výsledky měření propustnosti

V grafu *5.3 Výsledky měření propustnosti* jsou znázorněny naměřené hodnoty. Lze pozorovat závislost propustnosti (Throughput) na množství pravidel, která s jejich rostoucím počtem klesá. Obdobně je tomu při vyšším počtu paralelních spojení (Parallel connections). Zajímavostí je, že `ip_tables` s nastaveným jedním pravidlem dosahuje mírně vyšší propustnosti než filtr bez pravidel. To může být způsobeno jinou, efektivnější cestou paketu uvnitř jádra OS Linux.
Nejvyšší rychlost. Křivka znázorňující měření při použití 1 pravidla ukazuje nejvyšší změřenou propustnost zařízení.
Nejméně výkonné nastavení. Kromě experimentů se zvyšováním počtu pravidel je v grafu znázorněna propustnost při vyhledávání 28 regulárních výrazů (tzn. 28 různých pravidel, kdy každé vyhledává jeden regulární výraz, tím lze rozlišit až 28 různých protokolů aplikační vrstvy)

pomocí L7-filteru. Propustnost tohoto nastavení je výrazně nižší než v ostatních případech. Dosahuje pod hranici 4 Mbit/s.

5.5 Vyhodnocení dat získaných profilací

Během měření propustnosti byla prováděna profilace systému. V tabulce *5.1: Nejvytíženější funkce* uvádím nejvytíženější operace vč. průměrného zatížení spočítaného ze všech dosud provedených měření. Uvedeny jsou všechny funkce, které se průměrně podílely na zatížení procesoru alepoň jedním procentem.

5.5.1 Úzká hrdla aplikace

Dalším podrobnějším průzkumem lze zjistit, že vysoké zatížení funkce `strcmp` a `match` pochází z měření L7-filteru. Lze předpokládat, že funkci `strcmp` používá knihovna pro vyhledávání regulárních výrazů. Funkce `match` je vstupní funkcí (ve smyslu kapitoly *2.4 Netfilter/iptables*) rozšíření L7-filter, o kterém pojednávám v části *2.4.3 L7-filter pro iptables*.

Zatížení (%)	**Funkce**
24.7415	`strcmp`
13.2651	`ipt_do_table`
13.0253	`default_idle`
12.3662	`match`
8.98979	`cpu_idle`
8.7483	`__copy_tofrom_user`
7.50775	`do_csum`
3.16048	`tcp_packet`
3.12566	`a_block_aligned`
1.82	`memset`
1.64364	`nf_conntrack_in`
1.60344	`tcp_collapse`
1.40887	`nf_iterate`
1.21052	`kfree`
1.17452	`____nf_conntrack_find`
1.12083	`total_acct_packets`
1.09283	`tcp_rcv_established`
1.04719	`tcp_event_data_recv`
1.02639	`ip_rcv`

Tab. 5.1:: Nejvytíženější funkce

Funkce `ipt_do_table` je vstupním voláním modulu `ip_tables`, v této funkci jsou dále procházena a aplikována jednotlivá pravidla. Jde o klíčovou funkci filtrovací části aplikace. Většina dalších intenzivních volání je optimalizována překladačem (zejm. pomocí *inlining*), proto je nelze zaznamenat podrobněji.

Další funkce, která je výrazně vytížena, je `do_csum`, kde probíhají kontrolní součty paketů.

Ve funkcích `cpu_idle` a `default_idle` stráví procesor dohromady průměrně přes 20 procent času. Tato volání jsou procesorem prováděna v době, kdy nemá co zpracovávat. Proč je tato doba tak výrazná, nebylo v této práci zjišťováno.

Další skupinou funkcí, které vytvářejí režii při zpracování provozu, jsou:

- `tcp_collapse`,
- `tcp_rcv_established`,
- `tcp_event_data_recv` a
- `tcp_packet`.

Jedná se o část implementace transportního protokolu TCP. Tyto a ostatní dále nepopsané funkce mají jen poměrně nízký vliv na celkový výkon aplikace.
Režie programu *iperf*. Nevýhodou použitého způsobu měření je, že aplikace *iperf* pracuje díky aktivnímu MMU ve svém paměťovém prostoru. Veškerý provoz, který tento program měří, je proto kopírován z paměti jádra OS do paměti programu a obráceně. Režie tohoto kopírování se projevuje vysokým využitím funkce `__copy_tofrom_user`, ale také `a_block_aligned`, která zarovnává paměť pro rychlejší kopírování. V době psaní této práce mi nebyl znám žádný program, který by dokázal tuto nevýhodu odstranit.

5.5.2 Shrnutí výsledků profilace

Profilace potvrdila hypotézy uvedené v sekci *5.1 Popis zpracování paketu*. Operace, které nejvíce limitují propustnost aplikace, jsou:

- vyhledávání regulárních výrazů v paketech (`strcmp`, `match`),
- průchod pravidly *iptables* (`ipt_do_table`),
- výpočty kontrolních součtů (`do_csum`).

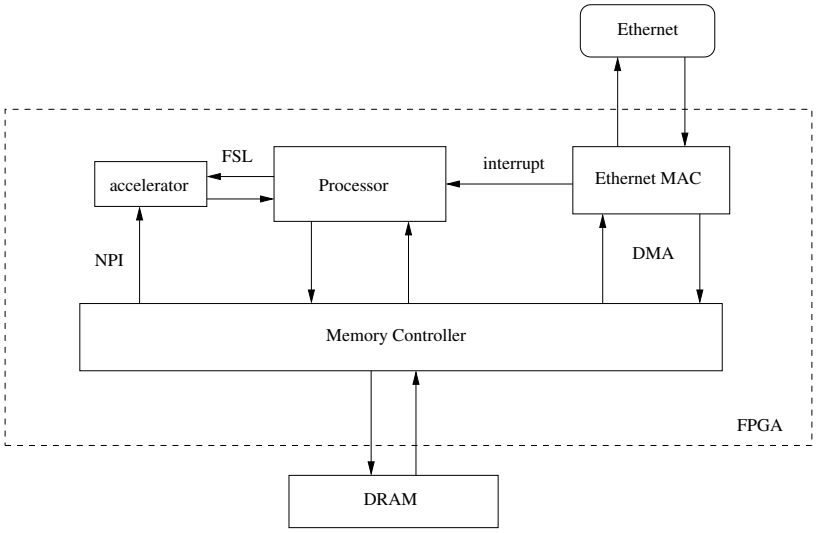
6 Návrh a implementace akcelerátoru

Předchozí kapitola pojednává o úzkých místech aplikace. Jednoznačně nejvýraznějším zpomalením je aktivace L7-filteru při pokusu o rozpoznávání různých protokolů aplikační vrstvy. V následujícím textu bude navržena a implementována hardwarová jednotka pro nahrazení softwarového L7-filteru.

6.1 Návrh rozhraní

Podstatným rysem operace rozpoznávání protokolu je zpracovávání velkého množství dat, ale pouze omezené množství dat reprezentuje výsledek (stačí např. bitové pole o délce větší nebo rovné počtu implementovaných regulárních výrazů).

Podle sekce *4.7.2 Urychlení algoritmů s velkým množstvím vstupů a malým množstvím výstupů* je vhodné implementovat akcelerátor za pomoci sběrnice FSL pro řízení jednotky a vyčítání výsledků a dále s přímým připojením akcelerátoru k hlavní paměti pomocí NPI. Hardwarová platforma obsahující tuto akcelerační jednotku je znázorněna ve schématu *6.1 Návrh hardwarové platformy s akcelerátorem*.



Obr. 6.1: Návrh hardwarové platformy s akcelerátorem

6.2 Návrh struktury

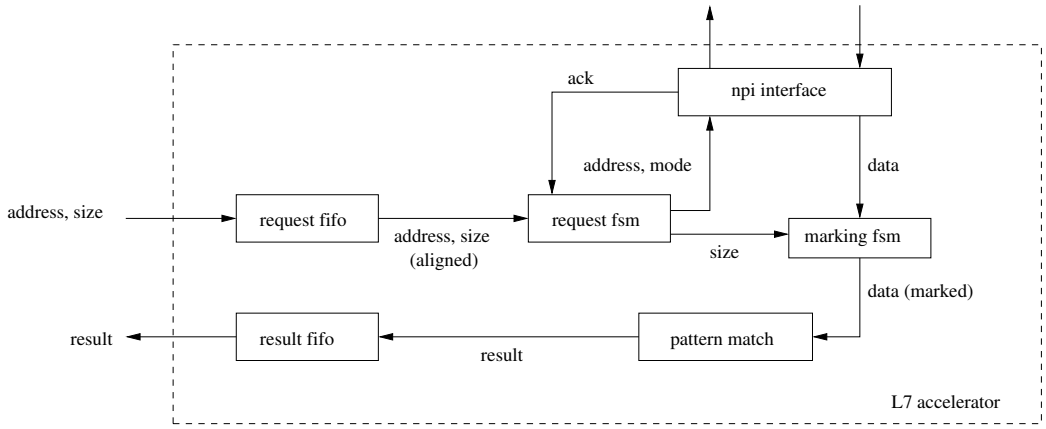
Akcelerátor bude provádět především následující akce:

- načítání adresy a délky paketu z procesoru,
- posílání požadavků do paměti,
- příjem dat z paměti,
- vyhledávání vzorů v proudu dat,

- vrácení výsledku do procesoru.

6.2.1 Popis struktury

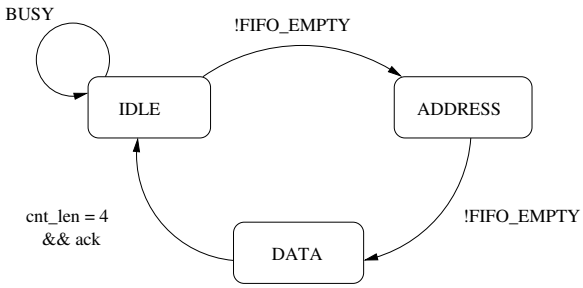
Pro zmíněné činnosti jsem navrhl systém, který lze rozdělit na dvě části: generátor požadavků, příjemce dat. Generátor požadavků je řízen hlavním automatem – **request fsm**. Příjemce dat řeší čtení příchozích dat z paměti a jejich zpracování jednotkou *pattern match*. Struktura jednotky je zobrazena na obrázku *6.2 Návrh hlavních částí akcelerátoru*.



Obr. 6.2: Návrh hlavních částí akcelerátoru

request fifo
Každý požadavek přicházející z procesoru je vložen do této fronty. V případě, že akcelerátor neprovádí zpracovávání dat, vybírá odsud nejstarší požadavek. Tím je v prvním slově adresa a v dalším délka. Pro snazší implementaci v kombinaci s procesorem MicroBlaze lze uvažovat šířku adresy 32 bitů a šířku délky dat menší nebo rovnu 32 bitům. Fronta odděluje hodinové domény akcelerátoru a procesoru. Akcelerátor tak může běžet na vyšší frekvenci.

request fsm
Schéma automatu je zachyceno na obrázku *6.3 Stavy automatu* **request fsm**. Ve stavu *IDLE* čeká na příchozí požadavky. Pokud je první část požadavku – adresa – přijata, přechází do stavu *ADDRESS* a čeká na druhou část – délku paketu. Oba údaje jsou uloženy do čítačů.



Obr. 6.3: Stavy automatu **request fsm**

Ve stavu *DATA* provádí automat opakované požadavky do paměti tak dlouho, dokud nepřečte celý požadovaný úsek. Na každý požadavek musí dostat od paměti potvrzení – `ack`, které způsobí inkrementaci čítače adresy a dekrementaci zbývajícího počtu bytů (hodnoty v čítačích byly zarovnány na šířku slova). Při přijetí potvrzení poslední žádosti přechází automat zpět do stavu *IDLE*. Protože paměť může mít zpoždění mezi potvrzením požadavku a dodáním dat, čeká automat na vynulování příznaku `BUSY` od jednotky přijímající data.

Automat má možnost rozhodnout o typu paměťového dotazu (viz část *3.4.1 Multi-Port Memory Controller (mpmc)*). Pokud je požadavek dostatečně dlouhý a aktuální adresa je vhodně zarovnaná, lze provést požadavek typu *burst*.

npi interface
Jednotka zajišťuje doplňující úpravy požadavků, např. odečtení bázové adresy (příchozí adresa ukazuje do adresového prostoru procesoru, ve kterém může být samotná paměť téměř libovolně umístěna). Důležitým úkolem této jednotky je vypořádání se s pevným zpožděním řadiče paměti. Příjem slova z fronty řadiče může být zpožděn až o 2 takty, což může v ostatních jednotkách vést ke zbytečně komplikované logice a potenciálnímu zanesení logických chyb.

marking fsm
Automat na základě délky paketu umí rozlišit první a poslední datové slovo. V příchozích datech proto provádí jejich značení. Jednotka pro vyhledávání regulárních výrazů tak může rozpoznat počátek a konec paketu.

pattern match
Nejdůležitější část akcelerátoru je samotná jednotka pro vyhledávání výrazů. Ze vstupu čte postupně jednotlivá slova paketu, která může v jednom taktu zpracovat (zde je jedno z potenciálních zrychlení L7-filteru). Výstupem jednotky je slovo, jehož jednotlivým bitům odpovídají implementované regulární výrazy. Pokud je příslušný bit nastaven, výraz byl v paketu detekován. V opačném případě výraz detekován nebyl.

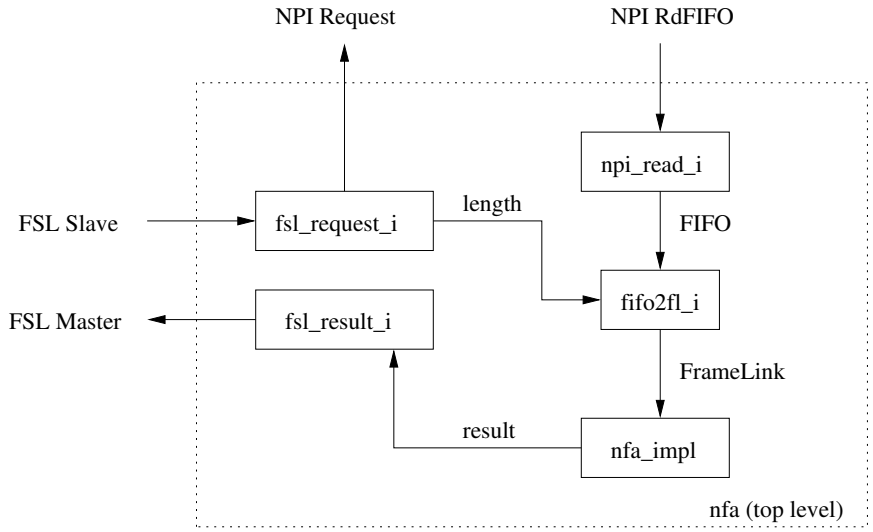
result fifo
Poslední akcí akcelerátoru je vrácení výsledku zpět do procesoru. Budeme-li uvažovat maximální počet regulárních výrazů roven 32, je možné vrátit výsledek jediným zápisem do výstupní fronty. V případě podpory více výrazů bude nutné implementovat protokol pro detekci posledního slova nebo umožnit procesoru vyčíst počet implementovaných výrazů a vracet smluvený počet slov, ze kterých si procesor vymaskuje výsledek.

6.3 Implementace akcelerátoru

Na základě předchozího návrhu byla v jazyce VHDL implementována jednotka nazvaná `nfa`. Vstupem jednotky je adresa *payloadu* paketu a jeho délka. Výstupem je bitový vektor určující, který z implementovaných regulárních výrazů byl v paketu nalezen.

Ze schématu *6.4 Akcelerátor nfa* na straně 38 je patrné, že se implementace liší v některých detailech. Logika **npi interface** byla rozdělena z části do `fsl_request_i` a z části do `npi_read_i`. Princip zpracování však zůstává stejný. Ostatní jednotky ve schématu již odpovídají původnímu návrhu.

Označení počátku a konce dat je prováděno pomocí protokolu FrameLink (jedná se o podmnožinu protokolu LocalLink, který se používá v rámci projektu Liberouter [25]).



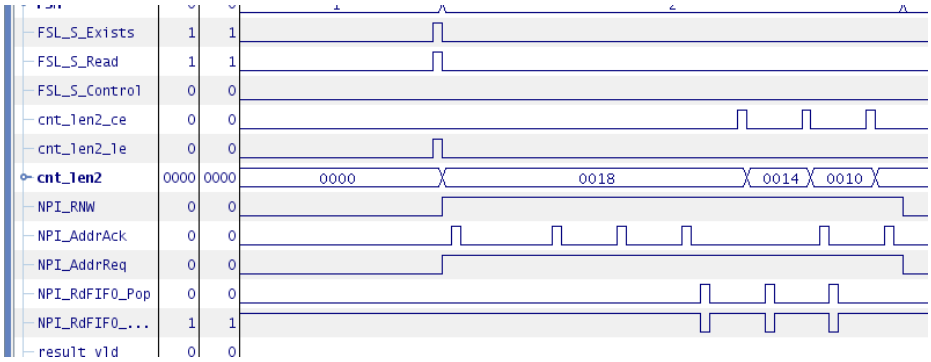
Obr. 6.4: Akcelerátor `nfa`

6.3.1 Implementace jednotky pattern match

V `nfa` je použit automat pro vyhledávání regulárních výrazů, který byl vytvořen v rámci projektu Liberouter, viz [25]. Před syntézou je nutné vygenerovat jeho přechodovou tabulku. Implementované regulární výrazy jsou tedy pouze statické a pro jejich změnu je třeba provést novou syntézu celého systému.

Počet implementovaných regulárních výrazů není teoreticky omezen, avšak z důvodů uvedených v části *result fifo* umožňuje současná verze `nfa` použít maximálně 32 samostatných výrazů. Jednotku lze rozšířit pro podporu více regulárních výrazů, a to úpravou výstupního modulu `fsl_result` z podoby kombinačního obvodu do jednoduchého konečného automatu. Výsledky zpracování pak bude nutné získávat několika čteními z FSL.

6.3.2 Implementace rozhraní NPI



Obr. 6.5: Průběh provádění požadavků do paměti

Akcelerátor `nfa` pracuje na frekvenci `mpmc`, která je typicky vyšší než frekvence procesoru MicroBlaze (v použitém designu dvojnásobně). Při optimálním čtení dat z paměti (bez prodlev) `nfa`

zpracuje jedno slovo za takt, kdy šířka slova může být 32 nebo 64 bitů (testována byla pouze 32bitová varianta). Pomocí sondy ChipScope umístěné v designu z důvodu ladění byl pořízen záznam testovacího přenosu (viz *6.5 Průběh provádění požadavků do paměti* na straně 39), ze kterého je patrné, že při provádění požadavků dochází k mnohataktovým prodlevám (vzdálenost jednotlivých náběžných hran signálu `NPI_AddrAck`).

Podle návrhu je možné optimalizovat přístupy do paměti do *burstů*. Tato část nebyla implementována, avšak v jednotce je i nadále prostor pro přidání této funkčnosti.

6.3.3 Komunikace s akcelerátorem

Jednotku lze uvést do provozu dvěma zápisy do příslušné FSL linky. Nechť je v registru `R0` adresa dat, která se budou zpracovávat, a v registru `R1` délka těchto dat, potom

```
put R0, FSL0
put R1, FSL0
```

spustí vyhledávání. Jakmile je paket zpracován, načteme výsledek např. do registru `R2`

```
get R2, FSL0 .
```

6.3.4 Propojení akcelerátoru s OS Linux

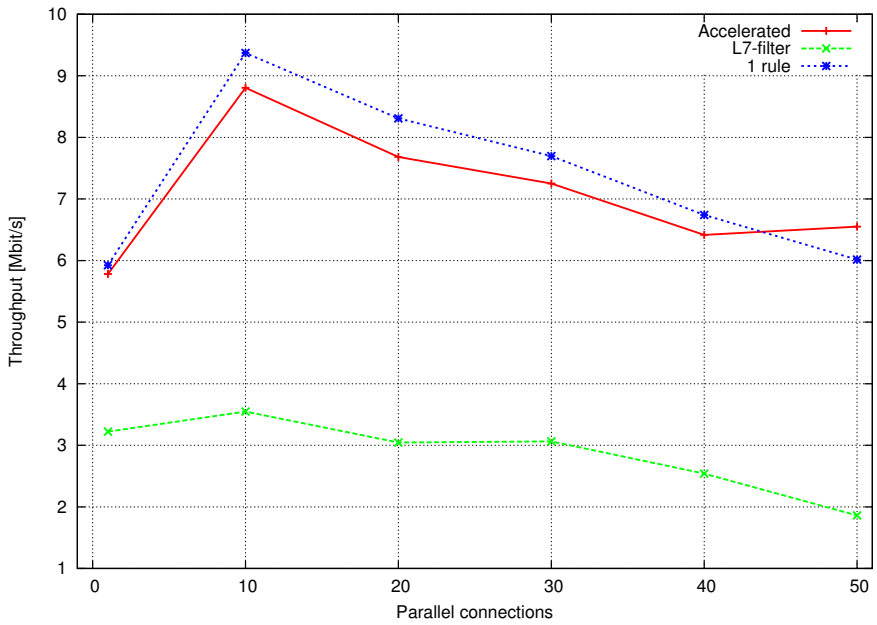
Aby bylo možné implementovanou jednotku použít v OS Linux, naprogramoval jsem modul systému `ip_tables` nazvaný `pm`. Tento modul umí komunikovat s hardwarovou jednotkou podle principu z předcházejícího textu. Modul lze nastavit programem *iptables* takto:

```
iptables -A INPUT -m pm --mask <BIT-MASK> .
```

Parametr `<BIT-MASK>` určuje, které regulární výrazy musí být nalezeny, aby bylo pravidlo považováno za úspěšné. Toto jediné pravidlo způsobí vyhledání všech výrazů. Výsledek vyhledání je možné uložit, aby další pravidla hledající některý z těchto výrazů nemusela prohledávat celý paket znovu (v rámci této práce nebylo implementováno). Každá další detekce protokolu nad stejným paketem by tedy obnášela pouze bitovou operaci.

6.4 Měření propustnosti s akcelerátorem

V grafu *6.6 Propustnost s akcelerátorem, porovnání* na straně 40 jsou vyneseny výsledky měření propustnosti designu s přidaným akcelerátorem. Pro porovnání jsou v grafu uvedena také předchozí měření s 1 pravidlem a výsledky L7-filteru. K měření jsem použil stejné nástroje, jaké jsou popsány v části *5.4.1 Použité nástroje*. Zrychlení oproti softwarovému řešení je evidentní a v některých konfiguracích i více než dvojnásobné. Filtr s akcelerací se výkonově blíží nejvyšší naměřené rychlosti.



Obr. 6.6: Propustnost s akcelerátorem, porovnání

6.5 Využité zdroje v FPGA

Veličina	**Počet**	**Počet** (%)
RAMB16s	31	86
Slices	12802	86

Tab. 6.1: Celý systém po `place & route`

Veličina	**Počet**	**Počet** (%)
BRAMs	4	11

Tab. 6.2: BRAM paměť procesoru

Veličina	**Počet**	**Počet** (%)
Slices	11351	76
Slice Flip Flops	11695	39
BRAMs	31	86
Frequency	24.702 ns	40.483 MHz

Tab. 6.3: Celý systém po `xst`

Veličina	**Počet**	**Počet** (%)
Slices	2397	16
Slice Flip Flops	2970	10
BRAMs	7	19
Frequency	13.185 ns	75.844 MHz

Tab. 6.4: DDR SDRAM řadič

Veličina	**Počet**	**Počet** (%)
Slices	4322	29
Slice Flip Flops	3854	13
BRAMs	6	16
Frequency	20.363 ns	49.109 MHz

Tab. 6.5: Procesor MicroBlaze

Veličina	**Počet**	**Počet** (%)
Slices	53	0
Slice Flip Flops	66	0
BRAMs	1	2
Frequency	5.650 ns	176.991 MHz

Tab. 6.6: FSL v20 FIFO

Veličina	**Počet**	**Počet** (%)
Slices	255	1
Slice Flip Flops	156	0
Frequency	6.586 ns	151.837 MHz

Tab. 6.7: MicroBlaze PLB

Veličina	**Počet**	**Počet** (%)
Slices	95	0
Slice Flip Flops	126	0
Frequency	11.412 ns	87.627 MHz

Tab. 6.8: MDM

Veličina	**Počet**	**Počet** (%)
Slices	106	0
Slice Flip Flops	142	0
Frequency	6.666 ns	150.015 MHz

Tab. 6.9: XPS UARTLITE (RS-232)

Veličina	**Počet**	**Počet** (%)
Slices	2000	13
Slice Flip Flops	2388	8
Frequency	9.525 ns	104.987 MHz

Tab. 6.10: XPS LL TEMAC (soft)

Veličina	**Počet**	**Počet** (%)
Slices	121	0
Slice Flip Flops	174	0
Frequency	5.810 ns	172.117 MHz

Tab. 6.11: XPS INTC

Veličina	**Počet**	**Počet** (%)
Slices	325	2
Slice Flip Flops	360	1
Frequency	7.795 ns	128.287 MHz

Tab. 6.12: XPS Timer

Veličina	**Počet**	**Počet** (%)
Slices	1096	7
Slice Flip Flops	883	2
BRAMs	8	22
Frequency	8.568 ns	116.715 MHz

Tab. 6.13:: Akcelerátor `nfa`

7 Závěr

Výsledkem této práce je funkční aplikace pro filtrování síťového provozu implementovaná na FPGA čipu Spartan-3E za pomoci operačního systému Linux a filtrovacího systému *iptables*. Měření propustnosti aplikace a její profilování umožnilo identifikovat části, které omezují její celkovou výkonnost. Jedná se o rutiny provádějící:

- aplikaci jednotlivých pravidel filtru (v závislosti na jejich množství),
- výpočty kontrolních součtů paketů,
- vyhledávání regulárních výrazů v aplikační vrstvě.

Poslední zmíněná možnost způsobovala nejvýraznější zpomalení systému, proto byla nahrazena akcelerační jednotkou v FPGA a novým modulem operačního systému, který s touto jednotkou komunikuje – předává jí pakety ke zpracování.

Z dalšího měření vyplynulo, že přesunutí detekce aplikačních protokolů ze softwaru do logiky FPGA umožnilo zvýšit propustnost systému přibližně dvojnásobně a při některých situacích i trojnásobně. Akcelerátor navíc používá neoptimální přístup do hlavní paměti, což má negativní vliv na jeho propustnost. Jednotku lze potenciálně rozšířit pro zpracovávání až dvojnásobného množství dat za takt zdvojnásobením šířky jeho datové sběrnice.

Další zvýšení propustnosti je možné vhodnou akcelerací hlavní funkce modulu `ip_tables`, kde dochází k lineárnímu průchodu pravidly, což zejm. při větším množství pravidel vede k nízkému výkonu. Akcelerátor by musel implementovat nalezení odpovídajících pravidel v logice FPGA jiným způsobem, např. vhodně uzpůsobenou hashovací funkcí. Pokud by takto zachoval stávající princip filtrování, bylo by možné využívat `ip_tables` vč. všech existujících rozšíření.

Literatura

[1] HAEGARTY, Tim. *Using iptables and the Netfilter Framework. Linux+.* 2007-02, Vol. 1, No. 3, s. 30–40. Dostupný také z WWW: <http://lpmagazine.org/magazine/1373-gentoo-linux-2007-0-guide>. ISSN 1896-8309.

[2] LOVE, Robert. *Linux Kernel Development.* 3rd edition. Upper Saddle River, NJ: Addison-Wesley, c2010. 440 s. ISBN 978-0-672-32946-3.

[3] MASTERS, Jon; BLUM, Richard. *Professional Linux Programming.* Indianapolis (Indiana): Wiley Publishing, c2007. 466 s. ISBN 978-0-471-7761.

[4] RODRIGUEZ, Claudia Salzberg; FISCHER, Gordon; SMOLSKI, Steven. *The Linux Kernel Primer: A Top-Down Approach for x86 and PowerPC Architectures.* Prentice Hall PTR, c2006. 648 s. ISBN 0-13-118163-7.

[5] ZWICKY, Elizabeth D.; COOPER, Simon; CHAPMAN, D. Brent. *Building Internet Firewalls.* Second Edition. O'Reilly Media, 2000. 896 s. ISBN 978-1-56592-871-8.

[6] *Constraints Guide* [online]. UG625 (v12.3). Xilinx, 2010-09-21 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/cgd.pdf>.

[7] *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)* [online]. DS449. Xilinx, 2010-04-19 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf>.

[8] *LogiCORE IP XPS Interrupt Controller (v2.01a)* [online]. DS572. Xilinx, 2010-04-19 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf>.

[9] *LogiCORE IP XPS LL TEMAC (v2.03a)* [online]. DS537. Xilinx, 2010-12-14 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_temac.pdf>.

[10] *LogiCORE IP XPS Timer/Counter (v1.02a)* [online]. DS573. Xilinx, 2010-04-19 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_timer.pdf>.

[11] *LogiCORE IP Tri-Mode Ethernet MAC v4.5 User Guide* [online]. UG138. Xilinx, 2011-03-01 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/tri_mode_eth_mac_ug138.pdf>.

[12] *MicroBlaze Debug Module (MDM) (v2.00.a)* [online]. DS641. Xilinx, 2010-6-23 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/mdm.pdf>.

[13] *MicroBlaze Development Kit Spartan-3E 1600E Edition User Guide* [online]. UG257 (v1.1). Xilinx, 2007-12-05 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/boards_and_kits/ug257.pdf>.

[14] *MicroBlaze Processor Reference Guide* [online]. UG081 (v11.4). Xilinx, 2010-11-15 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf>.

[15] *Multi-Port Memory Controller (MPMC) (v6.02.a)* [online]. DS643. Xilinx, 2010-09-21 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf>.

[16] *Platform Specification Format Reference Manual* [online]. UG642. Xilinx, 2010-09-21 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/psf_rm.pdf>.

[17] *PLBV46 Interface Simplifications* [online]. SP026 (v1.2). Xilinx, 2008-12-05 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/edk10_sp026.pdf>.

[18] *Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC User Guide* [online]. UG074. Xilinx, 2010-02-22 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/user_guides/ug074.pdf>.

[19] *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide* [online]. UG194. Xilinx, 2011-02-14 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/user_guides/ug194.pdf>.

[20] *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide* [online]. UG368. Xilinx, 2011-03-01 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/user_guides/ug368.pdf>.

[21] *XPS General Purpose Input/Output (GPIO) (v2.00.a)* [online]. DS269. Xilinx, 2010-04-19 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_gpio.pdf>.

[22] *XPS 16550 UART (v3.00a)* [online]. DS577. Xilinx, 2010-08-2010 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_uart-16550.pdf>.

[23] *XPS UART Lite (v1.01a)* [online]. DS571. Xilinx, 2010-04-19 [cit. 2011-05-17]. Dostupné z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/xps_uart-lite.pdf>.

[24] *GCC, the GNU Compiler Collection: GCC 4.6 Release Series* [online]. March 2011, 2011-05-11 [cit. 2011-05-17]. Changes, New Features, and Fixes.
Dostupné z WWW: <http://gcc.gnu.org/gcc-4.6/changes.html>.

[25] *Liberouter: Programmable hardware* [online]. 2002, 2010-09-27 [cit. 2011-05-17]. Dostupné z WWW: <http://www.liberouter.org>.

[26] *Linux from Scratch* [online]. c1998 [cit. 2011-05-17]. Dostupné z WWW: <http://www.linux-fromscratch.org/>.

[27] RUSSELL, Rusty; WELTE, Harald. *The netfilter.org project* [online]. rev. 521. 2002, 2002-07-02 [cit. 2011-05-17]. Linux netfilter Hacking HOWTO. Dostupné z WWW: <http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html>.

Seznam příloh

1. Obsah DVD

Příloha 1
Obsah DVD

`/mbsl`
: MicroBlaze Simple Linux distribution, vývojová verze určená k profilování. Oficiální verze je k dispozici ve veřejném GIT repozitáři na adrese `http://www.stud.fit.vutbr.cz/~xvikto03/gitweb/mbsl.git`.

`/linux-design`
: EDK projekt se zdrojovými kódy použitého designu.

`/text`
: Text práce ve zdrojové podobě.

`/text/refs`
: Některé dokumenty uvedené v seznamu literatury.

`/results`
: Výsledky měření a profilování na MicroBlaze.

`/pcores`
: Zdrojové kódy implementovaného akcelerátoru vč. grafických schémat.