Západočeská univerzita v Plzni
Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Řízení komunikačních rozhraní v obvodu FPGA

Diplomová práce

Autor práce: Bc. Jakub Valenta
Vedoucí práce: Ing. Petr Burian

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub VALENTA**
Osobní číslo:	**E10N0138P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Řízení komunikačních rozhraní v obvodu FPGA**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Seznamte se s vývojovým kitem Altera DE2-70 a s rozhraním Avalon.
2. Vytvořte moduly (v jazyku VHDL) pro obsluhu komunikačních rozhraní (USB, PS/2, Ethernet) s možností ovládání těchto modulů procesorem NIOS II.
3. Navrhněte a vytvořte modul schopný komunikace po rozhraní Bluetooth.
4. Vytvořte ukázkovou aplikaci, která bude demonstrovat funkci všech vytvořených modulů.
5. Postup návrhu podrobně popište. Vypracujte stručný manuál pro použití vytvořených modulů.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

PINKER, J.; POUPA, M.: Číslicové systémy a jazyk VHDL, 2006.

Vedoucí diplomové práce: **Ing. Petr Burian**
Regionální inovační centrum elektrotechniky

Konzultant diplomové práce: **Ing. Petr Burian**
Regionální inovační centrum elektrotechniky

Datum zadání diplomové práce: **17. října 2011**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Anotace

Tato diplomová práce se zabývá návrhem a realizací VHDL modulů pro řízení komunikačních rozhraní v obvodech FPGA. Podstatnou část tvoří podrobný popis vytvořených modulů pro rozhraní PS/2 klávesnice a rozhraní Ethernet. U každého modulu se nachází jeho specifikace, možná obsluha procesorem Nios II a způsob nastavení. Pro demonstraci a ověření funkčnosti modulů je vytvořena ukázková aplikace. Součástí práce jsou přiložené zdrojové kódy.

Klíčová slova

Řízení, komunikační rozhraní, Ethernet, PS/2 klávesnice, VHDL, FPGA, Nios II

Abstract

This diploma thesis deals with design and realization of VHDL modules for communication interface control in FPGA. The main part is consists of description of modules for PS/2 keyboard interface and Ethernet interface. Each part contains specifications, interfacing with Nios II processor and configuration. For demonstration of module function there is an example application. Source code are included.

Keywords

Control, Communication interface, Ethernet, PS/2 keyboard, VHDL, FPGA, Nios II

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni, dne .....

Jakub Valenta

Poděkování

Rád bych tímto poděkoval vedoucímu diplomové práce, Ing. Petrovi Burianovi a dále také Ing. Jakubovi Vláškovi, za pomoc, kterou mi věnovali formou konzultací, diskusí i cenných připomínek, jež přispěly ke splnění úkolu této práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ASCII	American Standard Code for Information Interchange
BSP	Board Support Package
FIFO	First In First Out
FPGA	Field Programmable Gate Array
IP	Internet Protocol
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LSB	Least Significant Bit
MAC	Media Access Control
MM	Memory Mapped
PC	Personal Computer
PHY	PHYsical layer
PS/2	Personal System/2
RAM	Random Access Memory
SOPC	System on a Programmable Chip Builder
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

1 Úvod

Programovatelná hradlová pole FPGA jsou číslicové integrované obvody, které díky své programovatelnosti a snadnému návrhu nacházejí uplatnění v široké škále aplikací. Součástí FPGA jsou rychlé a konfigurovatelné bloky pamětí, hardwarové násobičky pro podporu aritmetických operací, fázové závěsy a velké množství IO standardů.

Pro výukové účely jsou ve škole používány vývojové desky Altera DE2-70, které obsahují množství periférií. Příkladem mohou být zvukové vstupy a výstupy, VGA, Ethernet, USB, sériový port RS-232 a mnoho dalších. Součástí vývojové desky je i softwarový procesor Nios II, který umožňuje pohodlné řízení těchto periférií. Nevýhodou však zůstává fakt, že pro některé rozhraní není procesor natolik rychlý, aby využil jejich plný výkon.

Cílem je tedy zrychlení, kterého bude dosaženo přesunem obsluhy do hradlového pole mimo pomalý soft-core procesor Nios II. Tato diplomová práce se zabývá návrhem modulů pro komunikační rozhraní v jazyce VHDL, to zahrnuje rozhraní pro PS/2 klávesnici, Ethernet, USB a Bluetooth. V případě rozhraní Bluetooth, které není na vývojové desce dostupné, se použije modul připojený přes sériové komunikační rozhraní RS-232. Součástí zadání je také vytvoření ukázkové aplikace, na které bude demonstrována funkčnost a využití jednotlivých modulů. K dosažení těchto cílů je zapotřebí se detailně seznámit s vývojovou deskou Altera DE2-70 a rozhraním Avalon. Informace budou čerpány téměř výhradně z firemních materiálů společnosti Altera a katalogových listů.

Diplomová práce je rozdělena do několika částí. První část se věnuje použitým vývojovým prostředkům, protože podrobné informace jsou dostupné v katalogových listech a uvedené literatuře, je tato část zaměřena především na problematiku s platformou Linux a věcmi s ní související. V následujících částech jsou detailně popsány navržené komponenty, u rozhraní Ethernet a USB je otestována datová propustnost a provedeno srovnání s dostupnými řešeními. Další část je pro ověření správné funkčnosti věnována demonstraci vytvořených modulů. Závěrečná část obsahuje návod k inicializaci a k nastavení těchto modulů.

2 Linux a vývojové prostředí společnosti Altera

Společnost Altera poskytuje své vývojové prostředí a nástroje i pro operační systém Linux. Většina nástrojů funguje korektně, problém však může nastat např. při použití USB JTAG nebo při simulaci systému Qsys v simulátoru Modelsim.

Pro tuto práci byl využíván následující software:

- ModelSim-Altera Starter Edition
- Quartus II Web Edition

Oba tyto softwarové balíky jsou dodávány ve formě shellového skriptu, který obsahuje samotný instalační program zabalený ve formátu *tar*. Po spuštění se archiv rozbalí a následuje instalace podobná verzi pro MS Windows. Protože je Quartus II psán v jazyce Java, je nutné mít pro jeho běh také Java Virtual Machine (JVM).

2.1 Nastavení rozhraní JTAG

Abychom mohli nahrát konfiguraci do FPGA, musíme mít správně nastavený JTAG. Problém připojení JTAGu většinou řeší správné nastavení práv pro USB. To se provede přidáním řádku uvedeného ve výpisu 1 do souboru /etc/fstab.

Výpis 1 Nastavení přístupových práv pro JTAG

```
usbfs /proc/bus/usb usbfs devmode=0666 0 0
```

Alternativou je vytvoření nového pravidla pro udev (správa souborů zařízení). Pravidlo zapíšeme např. do souboru /etc/udev/rules.d/51-usbblaster.rules a vložíme obsah výpisu 2.

Výpis 2 Vytvoření pravidla pro udev

```
# USB-Blaster
SUBSYSTEMS=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6001", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6002", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6003", MODE="0666"
```

Pravidla udev můžeme znovunačíst bez nutnosti restartu následujícím příkazem:

```
udevadm control --reload-rule
```

V obou případech nastavujeme práva 0666, což znamená čtení a zápis pro všechny (vlastník, skupina, ostatní). Ve druhém případě aplikujeme pravidla pouze na konkrétní typ zařízení. Podle uvedeného postupu máme tedy připravený JTAG přes USB.

2.2 Ovládání Nios II pomocí příkazové řádky

Vývojové prostředí a nástroje Nios II jsou kromě grafického rozhraní dostupné i ve formě příkazové řádky, což pro někoho může být pohodlnější. Nesporná výhoda této alternativy spočívá v tom, že je k dispozici více informací a výpisů. V případě nějakého problému jsme tedy schopni závadu lépe lokalizovat a identifikovat. Z vlastní zkušenosti mohu říci, že chybové výpisy např. v případě nerozpoznání hardware v nástroji *Nios II Software Build Tools for Eclipse* jsou více než strohé a je velmi obtížné daný problém vyřešit.

Příkazový řádek Nios II je standardně dostupný po instalaci Quartus II, skript pro spuštění se nachází v

```
{INSTALL_DIR}/altera/11.0/nios2eds/nios2_command_shell.sh
```

Po spuštění dostaneme k dispozici nové příkazy pro ladění, nahrávání a obsluhu zařízení. Připojené zařízení ověříme příkazem

```
$ jtagconfig
```

výpis by měl vypadat zhruba následovně:

```
1) USB-Blaster [2-1.2]
   020B60DD EP2C70
```

Kód EP2C70 je název zařízení, pokud ve výpisu tento název není, jedná se pravděpodoně o chybné spuštění *jtagd* ze strany Quartus II. Řešením je ukončení procesu *jtagd* a jeho následné spuštění. Aby bylo možné proces ukončit, je třeba zavřít *Nios II SBT for Eclipse*.

```
$ killall jtagd
$ jtagd
```

Použití příkazů pro konfiguraci FPGA a nahrání programu pro Nios II je uvedeno ve výpisu 3. První příkaz *nios2-configure-sof* zajišťuje stažení souboru .sof a následnou konfiguraci

FPGA, tento soubor vznikne po kompilaci projektu v Quartus II. Parametr, tedy název souboru není nutný, pokud se ale v adresáři vyskytuje více .sof souborů, je třeba jej specifikovat. Druhý příkaz *nios2-download* slouží pro nahrání programu .elf do procesoru Nios II, tento soubor vznikne po kompilaci projektu v *Nios II Software Build Tools for Eclipse*. Parametr -r resetuje cílové zařízení a parametr -g automaticky spustí procesor, u tohoto příkazu je nutné zadat cestu k .elf souboru. Poslední volitelný příkaz *nios2-terminal* umožňuje sledovat výpisy vytvořené v kódu programu pro Nios II.

U všech příkazů je dostupná také nápověda s detailním popisem všech možností daného příkazu, která se vyvolá parametrem –help. Kompletní dokumentace příkazů, které lze použít je dostupná v katalogovém listu [3].

Výpis 3 Konfigurace FPGA a nahrání programu pro Nios II

```
$ nios2-configure-sof ethernet.sof
$ nios2-download -r -g software/ethernet/ethernet.elf
$ nios2-terminal
```

3 Rozhraní Avalon

Rozhraní Avalon je paralelní sběrnice, určená pro snadný návrh systémů, propojování komponent a perfiférií. Každá komponenta je připojena pomocí jednoho ze standardizovaných rozhraní, v této práci jsou použity následující typy:

- Avalon-MM Master
- Avalon-MM Slave
- Avalon Conduit

Rozhraní Avalon-MM neboli *Memory Mapped* je založené na čtení a zápisu z určité adresy a slouží pro komunikaci *master - slave*. Avalon Conduit je určen pro propojení signálů do nejvyšší struktury Qsys, ze které mohou být připojeny na piny FPGA. Každá komponenta může obsahovat několik těchto rozhraní, např. jeden *slave* port pro konfiguraci a zároveň *master* port pro řízení jiné komponenty.

Rozhraní Avalon specifikuje velké množství signálů, ne všechny jsou však povinné. V tabulce 1 najdeme seznam a popis vybraných signálů, které jsou použity v této práci. Důležitou informací je to, že veškeré signály jsou synchronní. Kompletní informace lze získat např. z katalogového listu [1] nebo z literatury [9].

Tabulka 1: Popis signálů rozhraní Avalon

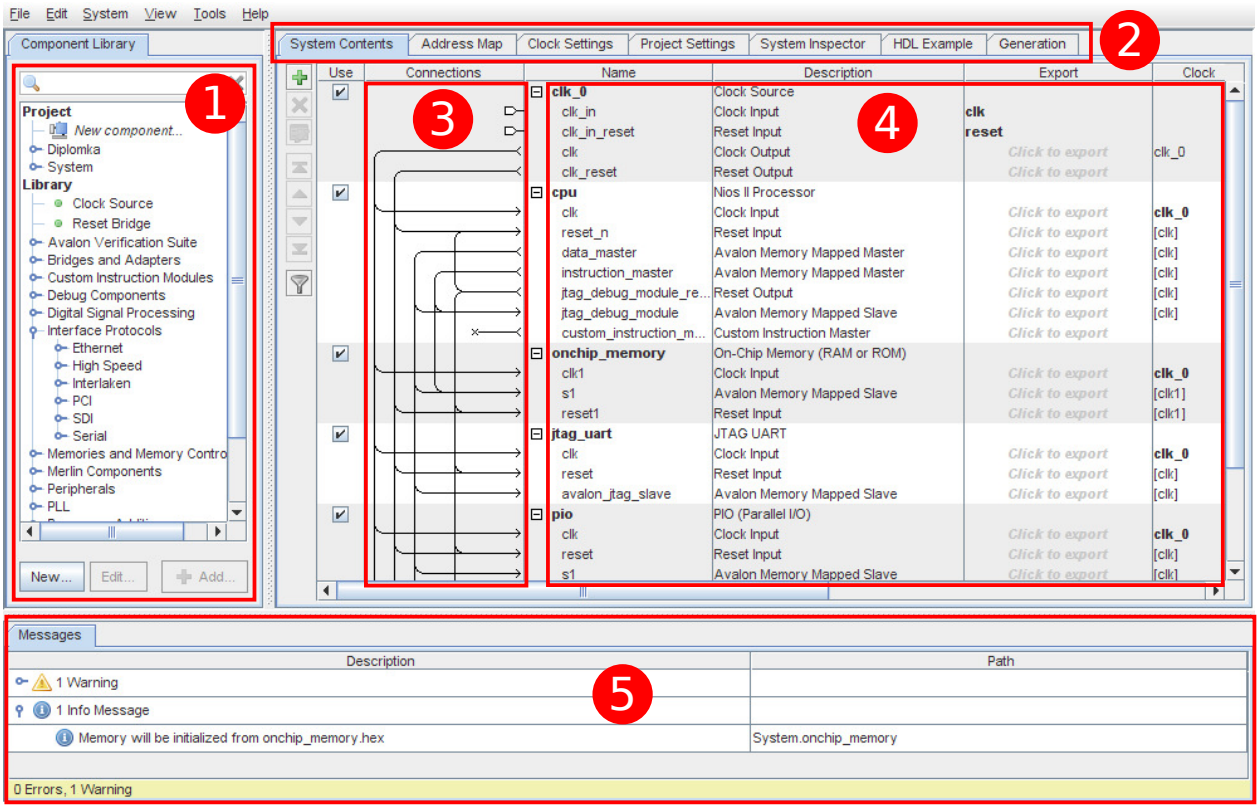
Název	**Šířka**	**Popis**
clk	1	vstupní hodinový signál
address	1-32	adresový prostor *slave* komponenty
read	1	požadavek o čtení z komponenty
readdata	1-128	data přečtená z komponenty
write	1	požadavek o zápis do komponenty
writedata	1-128	data určená pro zápis do komponenty
waitrequest	1	žádost *slave* komponenty o pozdržení přenosu
irq	1	žádost *slave* komponenty o přerušení

Šířka signálu je uváděna v bitech.

Spolu s rozhraním Avalon úzce souvisí nástroj Qsys (na obr. 1), který je součástí vývojového prostředí Quartus II. S tímto nástrojem lze jednoduše tvořit rozsáhlé systémy a sub-

systémy, které se skládají z jednotlivých komponent, procesoru Nios II, pamětí a periférií. Qsys navazuje na svého předchůdce - SOPC Builder, s nímž je zpětně kompatibilní a umožňuje jednoduchý import staršího návrhu. Oba vývojové nástroje mají podobné uživatelské rozhraní, od svého předchůdce se však Qsys liší zejména v následujících bodech [17]:

- vyšší výkon díky architektuře propojení založené na NoC (network-on-chip)
- ladění systému v reálném čase
- systém je možné poskládat ze subsystémů
- rozhraní periférií jsou vytvořeny podle průmyslových standardů



Obrázek 1: Nástroj Qsys

Komponenty je možné vybírat ze seznamu (1) a následně přidat do navrhovaného systému (4). V okně navrhnovaného systému (4) jsou pak dostupné informace o rozrhaní a paměťovém rozsahu.

Kromě připravených komponent je možné vytvořit i svojí komponentu, pomocí dialogového okna se vybere náš zdrojový kód a následně se toplevel entitě přiřadí odpovídající rozhraní a Avalon signály. Tím je vytvořena nová komponeta, která může být přidána do systému.

V části (3) propojujeme jednotlivé komponenty, různé hodinové signály a reset. Není problém např. připojit více komponent typu *master* k jedné komponentě typu *slave*, v tomto případě Qsys vytvoří arbitráž. Ve výpisu (5) jsou zobrazeny varovné hlášky, chyby a oznámení. Přepínač záložek (2) slouží pro zobrazení adresové mapy, nastavení projektu a vygenerování systému. Hotový a zkompilovaný Qsys systém lze ve vývojovém prostředí vkládat a připojovat např. na piny FPGA.

4 Rozhraní klávesnice PS/2

4.1 Protokol PS/2

Personal System/2 neboli zkráceně PS/2 je rozhraní typu *master - slave* a v dnešní době je skoro výhradně používáno pro připojení klávesnice a myši. Protokol umožňuje obousměrnou komunikaci - v případě klávesnice se scan kódy odesílají do řadiče, ve druhém směru nastavujeme např. LED. U protokolu PS/2 není *master* tvořen řadičem v PC, ale samotným koncovým zařízením. Rozhraní obsahuje kromě napájecích vodičů pouze dva datové signály:

- CLK - hodinový signál, který je generován *master* zařízením a to pouze pokud chceme přenášet data. Frekvence signálu je okolo 20 kHz a v klidovém stavu setrvává v log. 1. Pokud chce *slave* vyslat data, může signál stáhnout do log. 0 a tím oznámit druhému zařízení, aby přerušil vysílání dat.
- DATA - datový signál je ovládán *master* i *slave* zařízením a v případě potřeby může být také oběma stažen do log. 0.

4.1.1 Scan kódy

Každá klávesa má přiřazený svůj scan kód, který je odeslán při jejím stisknutí či uvolnění. Není tedy odesílána hodnota přímo v ASCII, na tu je potřeba daný scan kód následně převést. Kromě vlastního kódu, tzv. *make* kódu, je navíc odesílán i *break* kód, oznamující uvolnění klávesy. Pokud má například *make* kód klávesy „A“ hodnotu 0x1C, potom její *break* kód bude 0xF0, 0x1C. Při delším stisku klávesy se odesílá sekvence *make* kódů, po jejím uvolnění opět následuje ukončovací *break* kód.

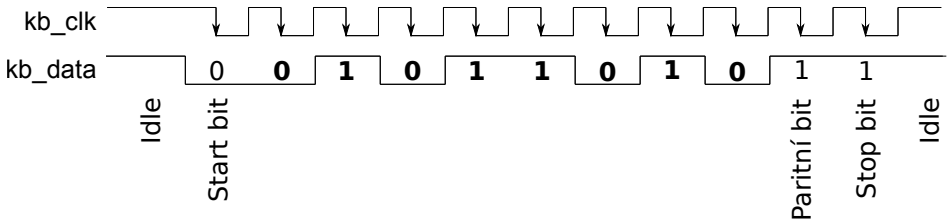
Některé speciální klávesy jsou složeny ještě z kódu 0xE0, jedná se např. o šipky, pravý Alt, Ctrl, Delete. Jejich *make* kód pak může vypadat jako sekvence znaků 0xE0, 0x71 a pro *break* kód 0xE0, 0xF0, 0x71 (klávesa Delete).

4.1.2 Přenos dat z koncového zařízení

Jak již bylo řečeno výše, protokol umožňuje obousměrnou komunikaci. Jednodušší variantou je přenos z koncového zařízení do řadiče, který jen naslouchá a na sestupnou hranu vzorkuje

data. Vysílaný datový rámec o délce 11 bitů je znázorněn na obr. 2, v tomto případě se přenáší scan kód 0x5A (klávesa Enter), složení jednotlivých bitů a jejich význam je popsán následovně:

- 1 start bit, vždy ve stavu log. 0
- 8 datových bitů, jako první se vysílá LSB
- 1 paritní bit (lichá parita)
- 1 stop bit, vždy ve stavu log. 1



Obrázek 2: Entita Keyb - časový průběh čtení PS/2

4.1.3 Přenos dat do koncového zařízení

Přenos dat do koncového zařízení je složitější než předchozí varianta. Lze tak odesílat do klávesnice různé příkazy sloužící např. pro žádost o znovuzaslání posledního bajtu, změnu sady scan kódu, rozsvěcení LED nebo nastavení doby po které se budou klávesy opakovat. Kompletní soupis příkazů je k dispozici v literatuře [7].

V této práci není přenos ve směru z řadiče do koncového zařízení použitý, zmíněn je pouze pro úplnost.

4.2 PS/2 komponenta

Komponenta pro čtení dat z koncového zařízení obsahuje rozhraní pro PS/2 a Avalon-MM Slave. Princip je takový, že po úspěšném přijmutí dat je zkontrolována parita a pokud jsou data v pořádku, tak se vyvolá žádost o přerušení signálem *avs_irq*. Následně jsou data vyčtena Avalon-MM Master komponentou, v tomto případě procesorem Nios II.

Z důvodu více hodinových domén je nutné řešit uvnitř komponenty resynchronizaci potvrzovacích signálů mezi Avalon-MM Slave a PS/2 rozhraním. Nestihne-li Nios II zpracovat přerušení dříve než přijde další bajt, je tato informace oznámena signálem *error_flag* Pokud nastane případ, kdy je přenos dat z nějakého důvodu přerušen (např. odpojením klávesnice), vyvolá watchdog časovač reset.

Reset komponenty může být vyvolán dvěma způsoby, prvním z nich je externě tlačítkem (signál *reset_n*) nebo vnitřně pomocí Watchdog časovače, který hlídá maximální čas pro přijmutí scan kódu. Pokud je přenos v některé části přerušen, čítač přeteče a komponenta je resetována.

4.2.1 Entita Keyb

PS/2 komponenta obsahuje pouze tuto jednu entitu, proto jsou jejich popisy shodné.

Obrázek 3: Entita Keyb

Tabulka 2: Signály VHDL komponenty pro PS/2

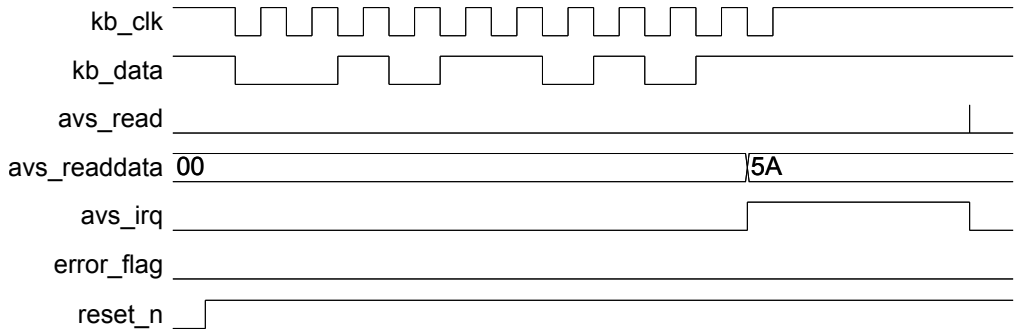
Název	**Směr**	**Typ**	**Šířka**	**Popis**
kb_clk	in	std_logic	1	hodinový signál PS/2
kb_data	in	std_logic	1	sériová data z PS/2
avs_clk	in	std_logic	1	hodinový signál Avalon
reset_n	in	std_logic	1	asynchronní reset
avs_read	in	std_logic	1	čtecí signál
avs_readdata	in	std_logic_vector	8	čtená data
avs_irq	out	std_logic	1	žádost o přerušení
error_flag	out	std_logic	1	oznámení o chybě

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Na obr. 4 je znázorněn časový průběh čtení přijatého scan kódu 0x5A. Z důvodu několikanásobně vyšší frekvence není hodinový signál *avs_clk* zobrazen. Význam jednotlivých bitů u signálů *kb_clk* a *kb_data* je detailně popsán v kapitole 4.1.2.

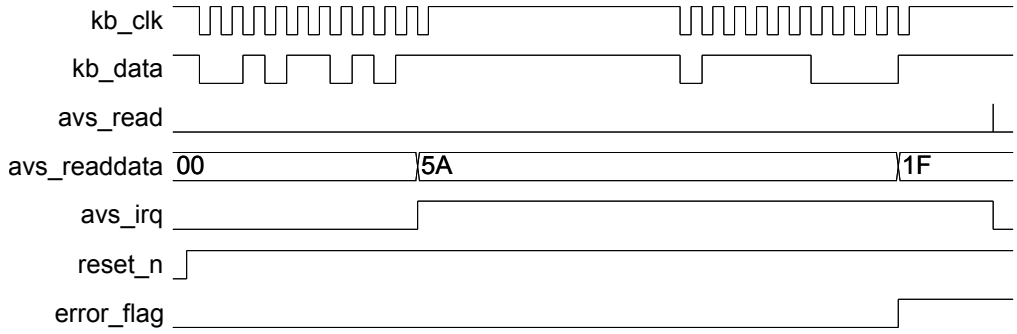
Po resetu jsou všechny signály ve své výchozí hodnotě a entita čeká na příjem dat z PS/2. S hodinovým signálem *kb_clk* jsou sériově bit po bitu přijímána data *kb_data*, nejprve LSB. Přijmutím stop bitu je zkontrolována i parita, pokud je vše v pořádku, vystaví se data na sběrnici *avs_readdata* a signálem *avs_irq* se vyvolá žádost o přerušení. Tímto je Nios II informován o přijatém znaku.

V obsluze přerušení vyčte Nios II signálem *avs_read* data *avs_readdata*, tím přechází přerušení *avs_irq* do neaktivního stavu.



Obrázek 4: Entita Keyb - časový průběh čtení PS/2 a Avalon

Na obr. 5 je simulace možné chyby, která vznikne v případě, kdy Nios II nestihne přečíst data dříve, než přijdou následující data. Levá část obrázku znázorňuje příjem scan kódu 0x5A, po dokončení informuje signál *avs_irq* o připravených datech. Nyní nastává situace, kdy se začne přijímat další scan kód 0x1F, aniž by byl ten předchozí přečten. Nios II nestihl data přečíst a přichází tak o jeden znak, to je signalizováno chybovým signálem *error_flag*.



Obrázek 5: Entita Keyb - časový průběh čtení PS/2 a Avalon s chybou

4.2.2 Stavový automat v Nios II

Stavový automat v Nios II slouží pro filtraci příchozích scan kódů. V kapitole 4.1.1 bylo podrobněji psáno o tzv. *make* a *break* kódech. Cílem tohto stavového automatu je odfiltrovat sekvenci znaků v *break* kódu a propustit jen *make* kód, včetně případu, kdy držíme klávesu trvale stisknutou.

Pro lepší představu o tom, jaké sekvence scan kódů mohou nastat, slouží tab. 3. V tabulce je uveden skutečný průběh sekvence, kterou posílá PS/2 klávesnice při stisknutí a uvolnění klávesy - *break* kód je zde oddělen čárkou. Stavový automat tedy musí rozpoznat typ scan kódu a podle toho odfiltrovat nepotřebná data. Zdrojový kód se nachází v souboru *ps2.c* umístěném v Quartus projektu, viz 9.1.

Tabulka 3: Příklad scan kódů

	Skutečný průběh	**Požadovaný průběh**
Jeden stisk (klávesa Enter)	5A, F0 5A	5A
Dlouhý stisk (klávesa Enter)	5A 5A 5A, F0 5A	5A 5A 5A
Jeden stisk (klávesa Delete)	E0 71, E0 F0 71	E0 71
Dlouhý stisk (klávesa Delete)	E0 71 E0 71 E0 71, E0 F0 71	E0 71 E0 71 E0 71

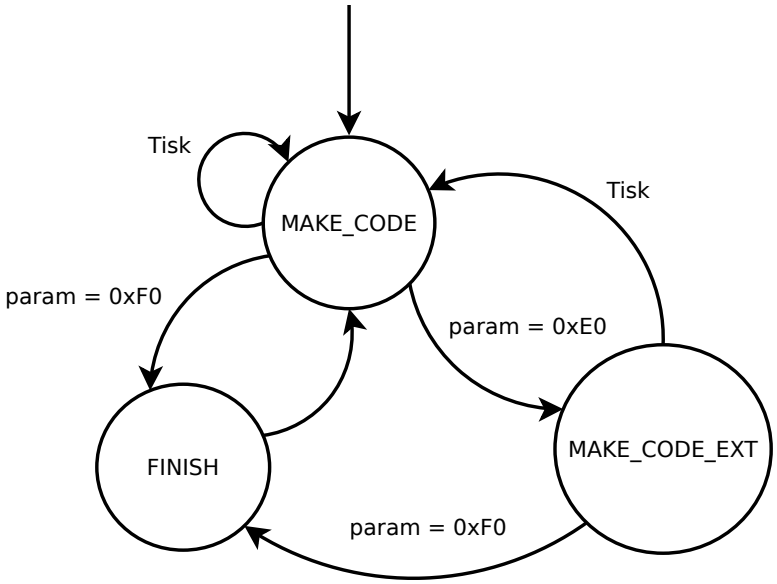
Hodnoty uvedené v tabulce jsou hexadecimální.

Chování stavového automatu v Nios II je zachyceno na obr. 6. Akce „Tisk“ u hran přechodů znamená v tomto případě vypsání scan kódu do konzole, stejně tak to ale může být např. odeslání po rozhraní UART. Výchozím stavem pro oba typy scan kódu je MAKE_CODE, v proměnné *param* je předáván aktuálně přečtený kód.

Nejprve popíšeme situaci jednoduššího scan kódu, např. 5A F0 5A. V momentě kdy přijde tento kód, je vyhodnoceno zda sekvence začíná E0 (druhý typ scan kódu) nebo F0 (začátek *break* kódu). Nejedná se ani o jeden případ a tak je provedena akce Tisk, ta se opakuje do té doby, dokud nepřijde začátek *break* kódu F0 (tím je ošetřen i dlouhý stisk). Příchodem kódu F0 setrvá stavový automat ještě jeden přechod ve stavu MAKE_CODE (bez Tisku) a následně přechází do stavu FINISH. Těmito kroky je odfiltrován *break* kód F0 5A.

Nyní situace s druhým typem scan kódu, např. E0 71, E0 F0 71. Stavový automat vyhodnotí první část sekvence E0 a uloží jí do proměnné *buffer*. V dalším kroku přechází automat do stavu MAKE_CODE_EXT a kontroluje, zda je další kód F0 nebo jiný (v tomto případě

71). Protože se nejedná o začínající *break* kód, provede se akce Tisk proměnné *buffer* i aktuálního kódu, tedy E0 71. Tento postup se opakuje do té doby, dokud není rozpoznán *break* kód začínající na E0 F0, poté následuje stav FINISH jako u předchozího příkladu. Protože v této variantě obsahuje *make* i *break* kód na začátku E0, je nutné posuzovat sekvenci se zpožděním jednoho kroku, tzn. počkat zda přijde F0 nebo jiný kód.



Obrázek 6: Stavový automat PS/2 v Nios II

4.2.3 Seznam kláves

Komponenta umí zpracovat libovolný scan kód, protože se jedná jen o posloupnost přijatých dat. Vytvořený filtr v Nios II je schopen zpracovat všechny klávesy kromě dvou speciálních, jedná se o PrintScreen a Pause. Tyto dvě klávesy mají atypické složení scan kódu, klávesa Pause navíc neobsahuje *break* kód.

5 Ethernet

5.1 Existující dostupná řešení

S vývojovým kitem DE2-70 je dodávána řada ukázkových aplikací [15], které demonstrují funkčnost jednotlivých komponent. Jednou z nich je i obsluha ethernetového čipu DM9000A, realizována pomocí softwarového procesoru Nios II. Ukázkový kód obsahuje funkce zajišťující

- inicializaci čipu
- nastavení fyzické vrstvy
- odeslání a příjem

Pro odeslání dat je v ukázce připraven ethernetový rámec se statickými daty, který je adresován pro všechna zařízení v síti.

Funkce jsou řešeny způsobem ne zcela odpovídajícím údajům v katalogovém listu. Konkrétně mezi zápisové a čtecí signály jsou vkládána zpoždění v řádech tisíců násobků základní periody. Spoléhá se tak na to, že během této doby bude mít ethernetový čip dostatek času na všechny operace a není třeba číst či generovat potvrzovací signály. Výsledkem je ale velmi pomalý přenos.

5.2 Důvody pro vytvoření vlastní komponenty

Jak bylo řečeno výše, celá ukázková aplikace je řešena pomocí Nios II. To vede k tomu, že procesor je tak stále zbytečně vytížený a v případě jiných operací je přenos zpomalován.

Cílem vlastní komponenty je přenést veškerou činnost spojenou s odesíláním dat mimo procesor. Výsledkem by tedy měla být VHDL komponenta, která bude zpracovávat příchozí data a posílat je v rámcích do ethernetového řadiče bez účasti procesoru.

5.3 Ethernetový řadič DM9000A

Abychom byli schopni správně pochopit činnost jednotlivých entit, zejména té, která obsluhuje přímo výstupní signály, je třeba uvést některé základní informace o tomto řadiči.

DM9000A je plně integrovaný ethernetový řadič s rozhraním pro řídící procesor. Řadič disponuje 10/100M PHY rozhraním a je navržen jako nízkopříkonový s podporou napájecího napětí od 3.3 do 5 V. Pro přístup do interní paměti můžeme použít 8 bit nebo 16bit rozhraní. Pro příjem a odesílání je zde paměť SRAM fungující jako fronta o celkové velikosti 16 kB, ta je rozdělena na 13 kB pro příjem (RX SRAM) a zbylé 3 kB pro odesílání (TX SRAM). Podrobnější informace jsou uvedeny v katalogovém listu [11].

5.3.1 Ovládání řadiče

Komunikace s řadičem DM9000A je vždy synchronní. Řadič lze ovládat signály uvedenými v tabulce 4.

Tabulka 4: Signály ethernetového řadiče DM9000A

Název	**Směr**	**Šířka**	**Popis**
IOR#	in	1	čtecí signál
IOW#	in	1	zápisový signál
CS#	in	1	výběr čipu
CMD	in	1	adresový signál (0 = index, 1 = data)
INT	out	1	signál přerušení
SD0~7	in/out	8	datová sběrnice, bity 0-7
SD8~15	in/out	8	datová sběrnice, bity 8-15 (pouze v 16bit režimu)

Znak # v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

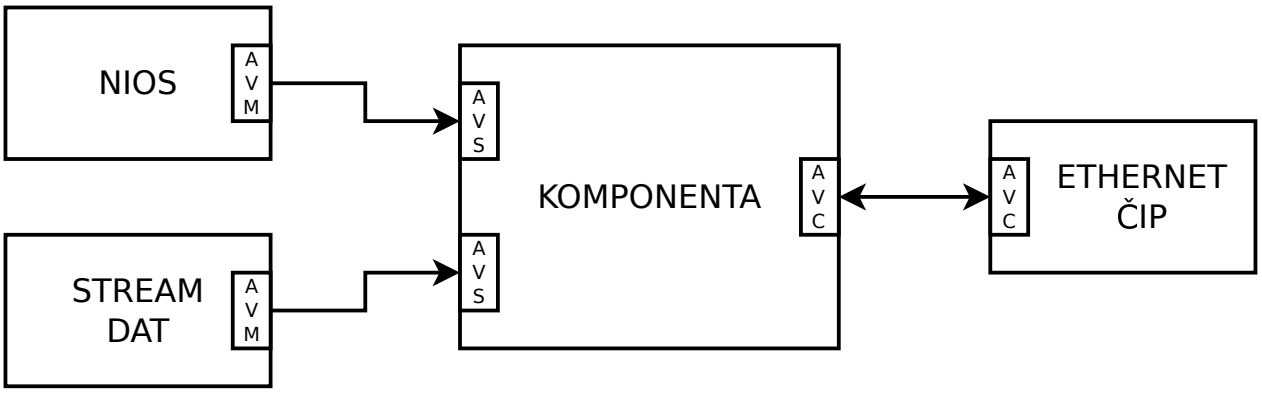
5.3.2 Zápis a čtení registru

Pro komunikaci s procesorem je řadič vybaven dvěma registry - indexovým a datovým. Jak zápis, tak i čtení tedy probíhá ve dvou krocích. Při zápisu do konkrétního registru je tedy nutné nejprve zapsat index do indexového registru a následně data do datového registru. V případě čtení se zapíše index registru a poté se přečtou data na sběrnici.

Pro výběr mezi indexovým a datovým registrem slouží signál CMD. Podrobné časové průběhy signálů jsou znázorněny např. v kapitole 5.4.5 nebo v katalogovém listu výrobce [11].

5.4 Ethernet komponenta

Komponenta je umístěna mezi ethernetovým řadičem a Nios II, resp. zdrojem datového streamu. Cílem bylo přesunout veškeré zpracování dat mimo procesor, inicializace řadiče a nahrání potřebných konfiguračních dat do RAM zůstává v režii Nios II. Vzhledem k tomu, že inicializace řadiče i zápis dat do RAM nastává pouze jednou, a to po resetu, není procesor dále vytěžován. Blokové schéma komponenty je na obr. 7.



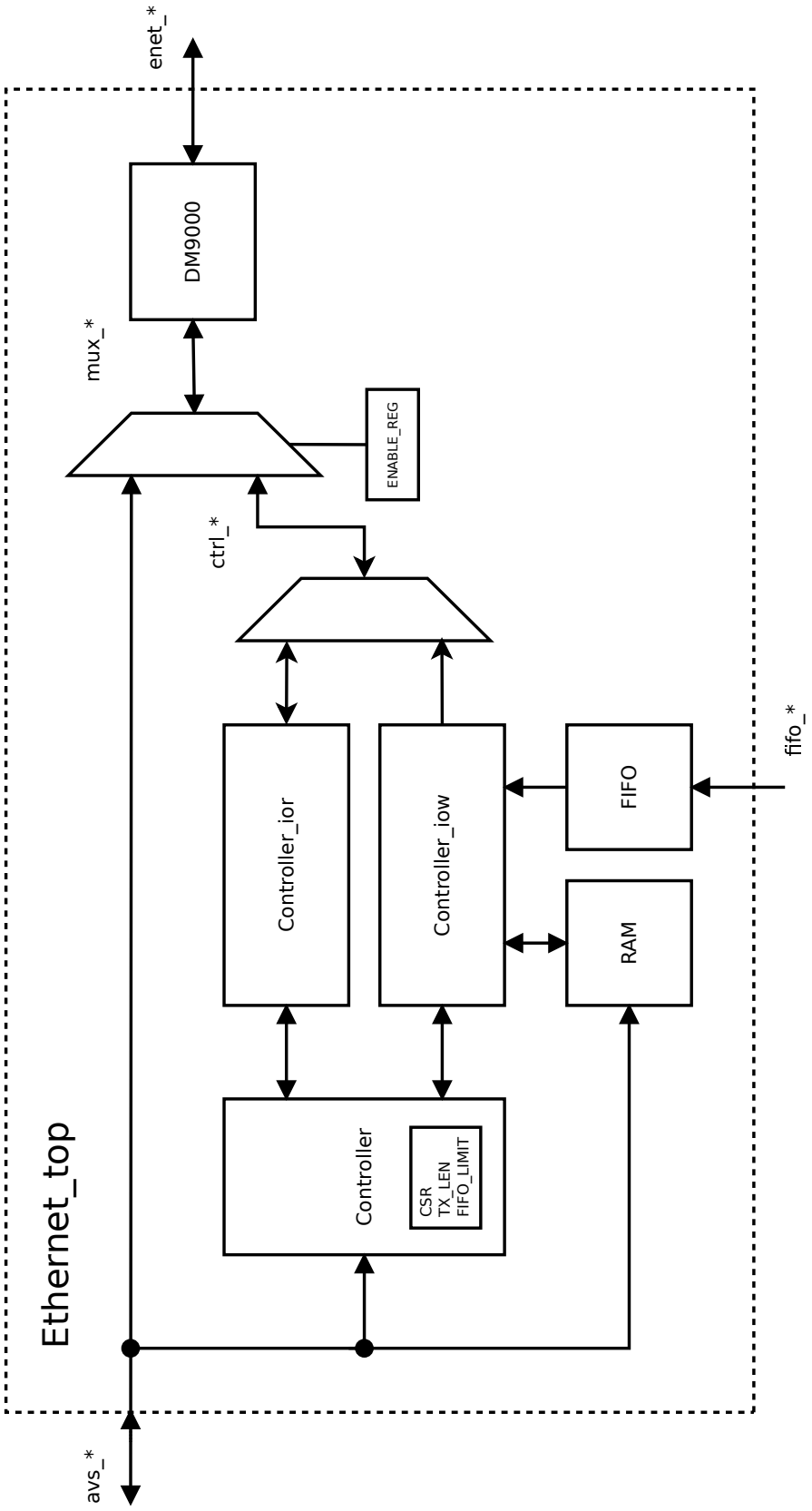
AVM = Avalon-MM Master
AVS = Avalon-MM Slave
AVC = Avalon Conduit

Obrázek 7: Blokové schéma

Komponenta může v závislosti na stavu řídícího registru pracovat v několika režimech:

- komponenta je zastavena
- přímý přístup Nios II k ethernetovému řadiči
- zápis dat z Nios II do RAM
- zpracování streamu dat a vysílání ethernetových rámců

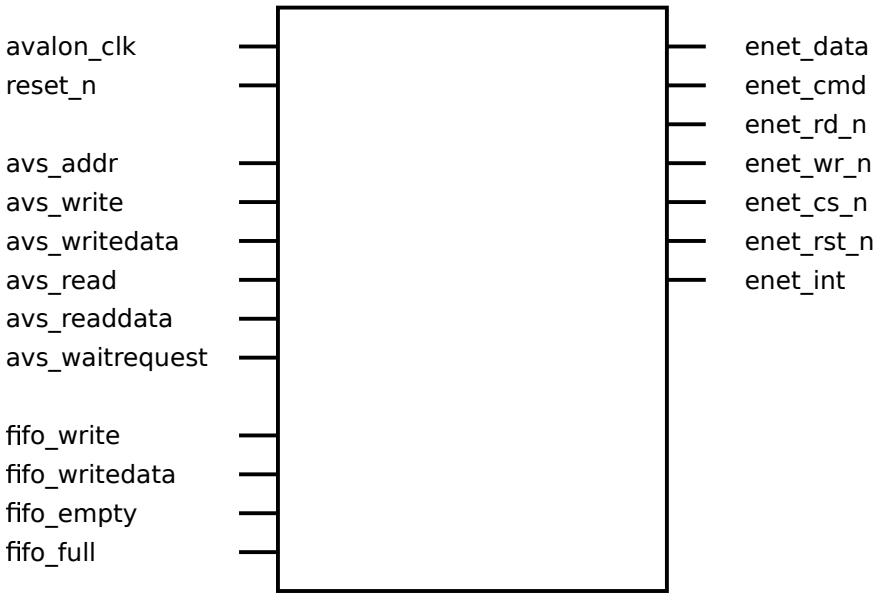
Samotná komponenta je tvořena hlavní entitou Ethernet_top, ta v sobě zahrnuje a vhodně propojuje další jednotlivé entity. Její součástí je i asynchronní multiplexer, ovládaný řídícím registrem. Tímto způsobem je vždy vybrána skupina odpovídajích signálů, které jsou následně přivedeny do ethernetového řadiče.



Obrázek 8: Detailní blokové schéma

5.4.1 Entita Ethernet_top

Entita Ethernet_top obsahuje několik rozhraní, jedná se o Avalon-MM Slave (prefix „avs_“), který je využíván pro přímý přístup Nios II k ethernetovému řadiči, konfiguraci komponenty a nahrání potřebných dat do RAM. Dalším vstupem je paměť FIFO (prefix „fifo_“), která je určena pro datový stream. Kromě zápisových a datových signálů jsou zde vyvedeny i stavové signály FIFO, říkající zdroji dat, zda je paměť prázdná či plná. Pro připojení k ethernetovému řadiči je použito rozhraní Avalon Conduit, které je vyvedeno přímo na piny (prefix „enet_“).



Obrázek 9: Entita Ethernet_top

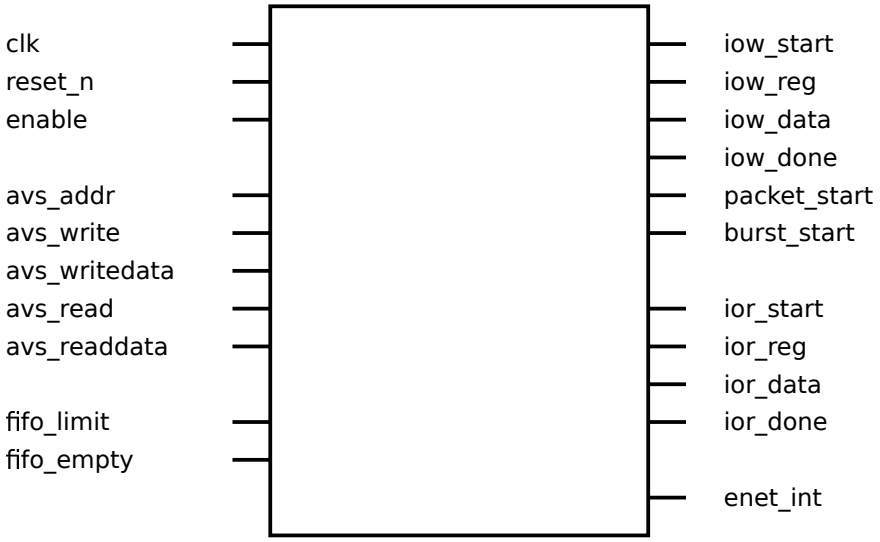
Tabulka 5: Popis signálů entity Ethernet_top

Název	**Směr**	**Typ**	**Šířka**	**Popis**
avalon_clk	in	std_logic	1	hodinový signál
reset_n	in	std_logic	1	asynchronní reset
avs_addr	in	std_logic_vector	5	adresový signál
avs_write	in	std_logic	1	zápisový signál
avs_writedata	in	std_logic_vector	32	data určená pro zápis
avs_read	in	std_logic	1	čtecí signál
avs_readdata	out	std_logic_vector	32	čtená data
avs_waitrequest	out	std_logic	1	signál pro pozdržení přenosu
fifo_write	in	std_logic	1	zápisový signál do paměti FIFO
fifo_writedata	in	std_logic_vector	32	data určená pro zápis do FIFO
fifo_empty	out	std_logic	1	signál oznamující prázdné FIFO
fifo_full	out	std_logic	1	signál oznamující plné FIFO
enet_data	inout	std_logic_vector	16	vstupní/výstupní data
enet_cmd	out	std_logic	1	adresový signál
enet_rd_n	out	std_logic	1	čtecí signál
enet_wr_n	out	std_logic	1	zápisový signál
enet_cs_n	out	std_logic	1	výběr řadiče
enet_rst_n	out	std_logic	1	asynchronní reset
enet_int	in	std_logic	1	přerušení ze strany řadiče

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

5.4.2 Entita Controller

Úkolem této entity je ovládání samotného ethernetového řadiče. To je realizováno stavovým automatem. Zahájení činosti automatu je podmíněno spouštěcím bitem *fsm_enable*, který je nastaven pomocí Nios II, a dále pak požadavkem o zahájení přenosu ze strany paměti FIFO. Tento požadavek je vyvolán v momentě, kdy je v paměti alespoň jedno slovo. Následně jsou v jednotlivých krocích postupně čteny hodnoty registrů, zejména se jedná o Network Status Register [11, str. 15] a Interrupt Status Register [11, str. 27]. V okamžiku odesílání dat je zápis přepnut do režimu „burst“, statická data jsou čtena z paměti RAM a po nich následují data z paměti FIFO. Slova z FIFO se čtou až do maximálního počtu, který je definován hodnotou v registru *CONTROLLER_FIFO_LIMIT*.



Obrázek 10: Entita Controller

Tabulka 6: Popis signálů entity Controller

Název	**Směr**	**Typ**	**Šířka**	**Popis**
clk	in	std_logic	1	hodinový signál
reset_n	in	std_logic	1	asynchronní reset
enable	in	std_logic_vector	2	povolení činnosti
avs_addr	in	std_logic_vector	5	adresový signál
avs_write	in	std_logic	1	zápisový signál
avs_writedata	in	std_logic_vector	32	data určená pro zápis
avs_read	in	std_logic	1	čtecí signál
avs_readdata	out	std_logic_vector	32	čtená data
fifo_limit	out	std_logic_vector	16	informace o max. počtu čtení z FIFO
fifo_empty	in	std_logic	1	informace o prázdném FIFO
iow_start	out	std_logic	1	oznámení o zahájení zápisového cyklu
iow_reg	out	std_logic_vector	16	hodnota indexu registru
iow_data	out	std_logic_vector	16	data určená pro zápis do registru
iow_done	in	std_logic	1	oznámení o konci zápisového cyklu
packet_start	out	std_logic	1	oznámení o zahájení odesílání rámce
burst_start	out	std_logic	1	oznámení o zahájení dávkového zápisu
ior_start	out	std_logic	1	oznámení o zahájení čtecího cyklu
ior_reg	out	std_logic_vector	16	hodnota indexu registru
ior_data	in	std_logic_vector	16	čtená data
ior_done	in	std_logic	1	oznámení o konci čtecího cyklu
enet_int	in	std_logic	1	přerušení ze strany řadiče

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Stavový automat

Výchozím stavem je START, ve kterém automat setrvává po resetu nebo v případě, že je činnost automatu zastavena. Do následujícího stavu SET_IMR se přechází v momentě kdy jsou splněny obě podmínky:

- automat je spuštěn pomocí bitu *fsm_enable*
- ve FIFO je alespoň jedno slovo

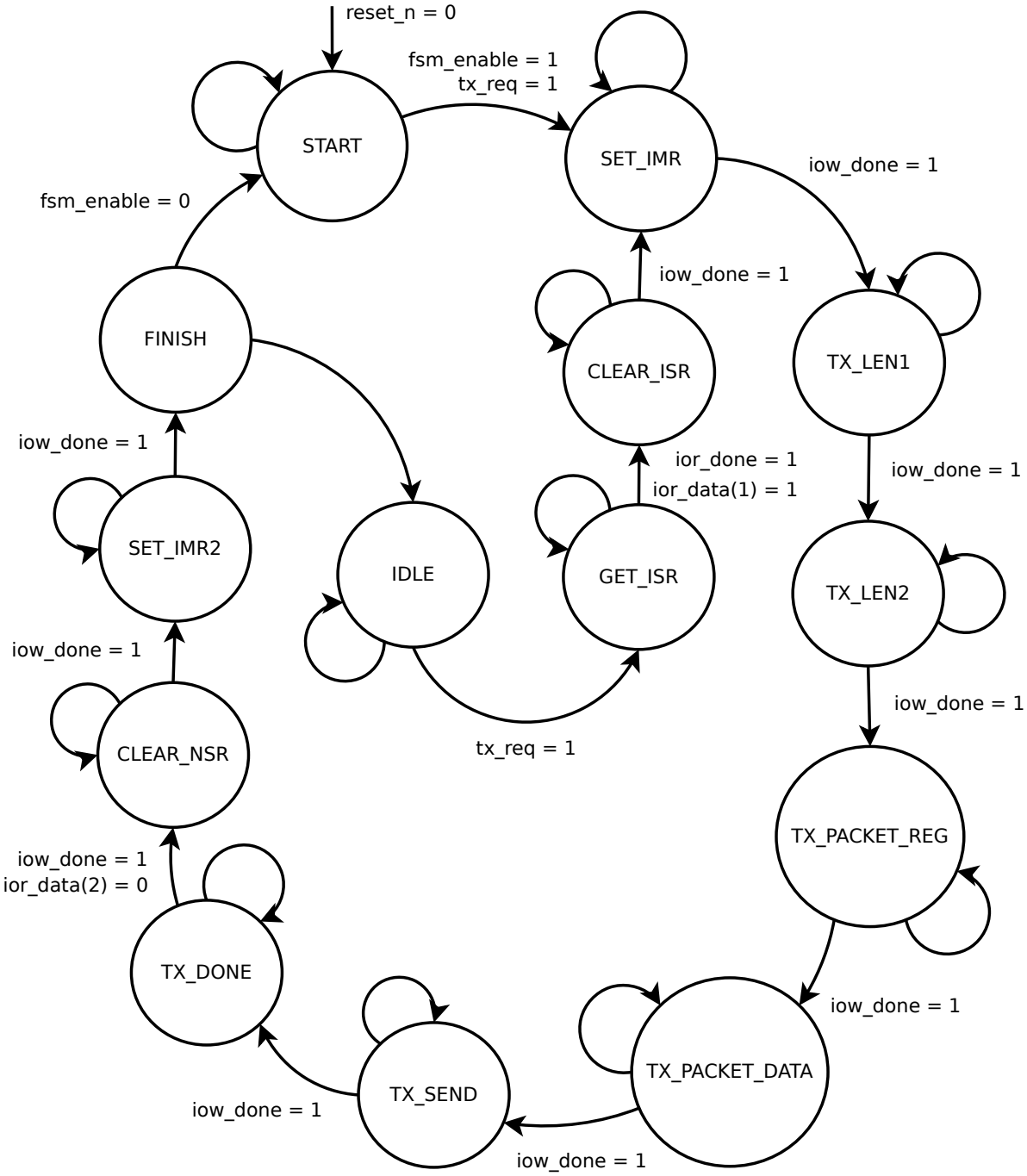
Stav SET_IMR nastavuje masku přerušení neboli Interrupt Mask Register [11, str. 27]. V následujících dvou stavech TX_LEN1 a TX_LEN2 se do řadiče zapíše celková délka paketu[1], nejprve horní a poté dolní bajt. Nyní se stavem TX_PACKET_REG zapíše index registru, který oznamuje „burst“ režim. V tomto momentě přechází automat do TX_PACKET_DATA a řadič tak zapisuje data do SRAM.

Po ukončení dávkového zápisu se rámec stavem TX_SEND odešle. Protože tato akce nějakou dobu trvá, čeká ve stavu TX_DONE na potvrzení o dokončeném odeslání, to je oznámeno bitem v registru NSR. Následujícím stavem CLEAR_NSR se registr NSR vynuluje. Stav SET_IMR2 znovunastaví masku přerušení. Stavem FINISH je celý cyklus ukončen a stavový automat tak přechází do IDLE s čekáním na další data. Pokud byl ale mezitím automat vypnut, vrací se do stavu START.

Další odesílání již vychází ze stavu IDLE, kde se vyčkává dokud není dostatek dat ve FIFO. V okamžiku kdy jsou data připravena se stavem GET_ISR ověří, zda byl předchozí rámec odeslán. Pokud ano, vynuluje se stavem CLEAR_ISR odpovídající registr a stavový automat přejde do stavu SET_IMR. Následující kroky jsou stejné a byly popsány výše.

Všechny přechody tohoto stavového automatu jsou podmíněném dokončením zápisu či čtení jeho vnořených stavových automatů. Tato informace je oznámena signálem *iow_done* nebo *ior_done*. Výjimku tvoří pouze stavy START, IDLE a FINISH.

[1]) V katalogovém listu výrobce řadiče se uvádí „TX Packet Length“, do této délky se však započítávají i MAC adresy, tudíž se nejedná přímo o paket, ale ani o úplný rámec.

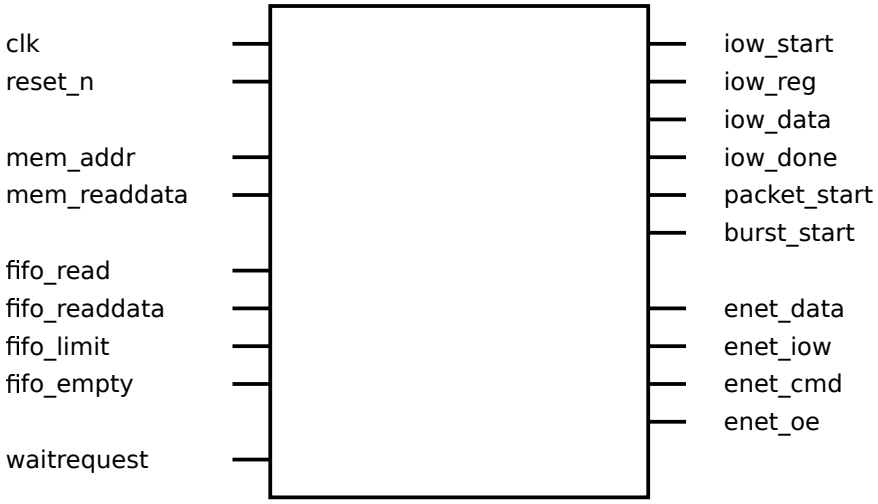


Obrázek 11: Stavový automat Controller

5.4.3 Entita Controller_iow

V entitě Controller_iow se rozhoduje, zda budou mít zapisovaná data význam registru nebo samotných dat. Dále pak umožňuje režim „burst“, při kterém jsou data souvisle zapisována do SRAM v ethernetovém řadiči. Součástí entity je i proces, který čte data z RAM a následně

i z FIFO. Počet slov v RAM je předem daný a neměnný, jedná se o statické data. Množství slov z FIFO je rovněž definované a odpovídá délce datové části UDP datagramu. Protože délka slov v RAM i FIFO je 32 bitů a šířka výstupní datové sběrnice je poloviční, musí se data číst pouze každý druhý cyklus.



Obrázek 12: Entita Controller_iow

Tabulka 7: Popis signálů entity Controller_iow

Název	**Směr**	**Typ**	**Šířka**	**Popis**
clk	in	std_logic	1	hodinový signál
reset_n	in	std_logic	1	asynchronní reset
mem_addr	out	std_logic_vector	5	adresový signál pro RAM
mem_readdata	in	std_logic_vector	32	čtená data z RAM
fifo_read	out	std_logic	1	čtecí signál pro FIFO
fifo_readdata	in	std_logic_vector	32	čtená data z FIFO
fifo_limit	in	std_logic_vector	16	informace o max. počtu čtení z FIFO
fifo_empty	in	std_logic	1	informace o prázdném FIFO
waitrequest	in	std_logic	1	signál pro pozdržení přenosu
iow_start	in	std_logic	1	oznámení o zahájení zápisového cyklu
iow_reg	in	std_logic_vector	16	hodnota indexu registru
iow_data	in	std_logic_vector	16	data určená pro zápis do registru
iow_done	out	std_logic	1	oznámení o konci zápisového cyklu
packet_start	in	std_logic	1	oznámení o zahájení odesílání rámce
burst_start	in	std_logic	1	oznámení o zahájení dávkového zápisu
enet_data	out	std_logic_vector	16	výstupní data
enet_iow	out	std_logic	1	zápisový signál
enet_cmd	out	std_logic	1	adresový signál
enet_oe	out	std_logic	1	povolení výstupu

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

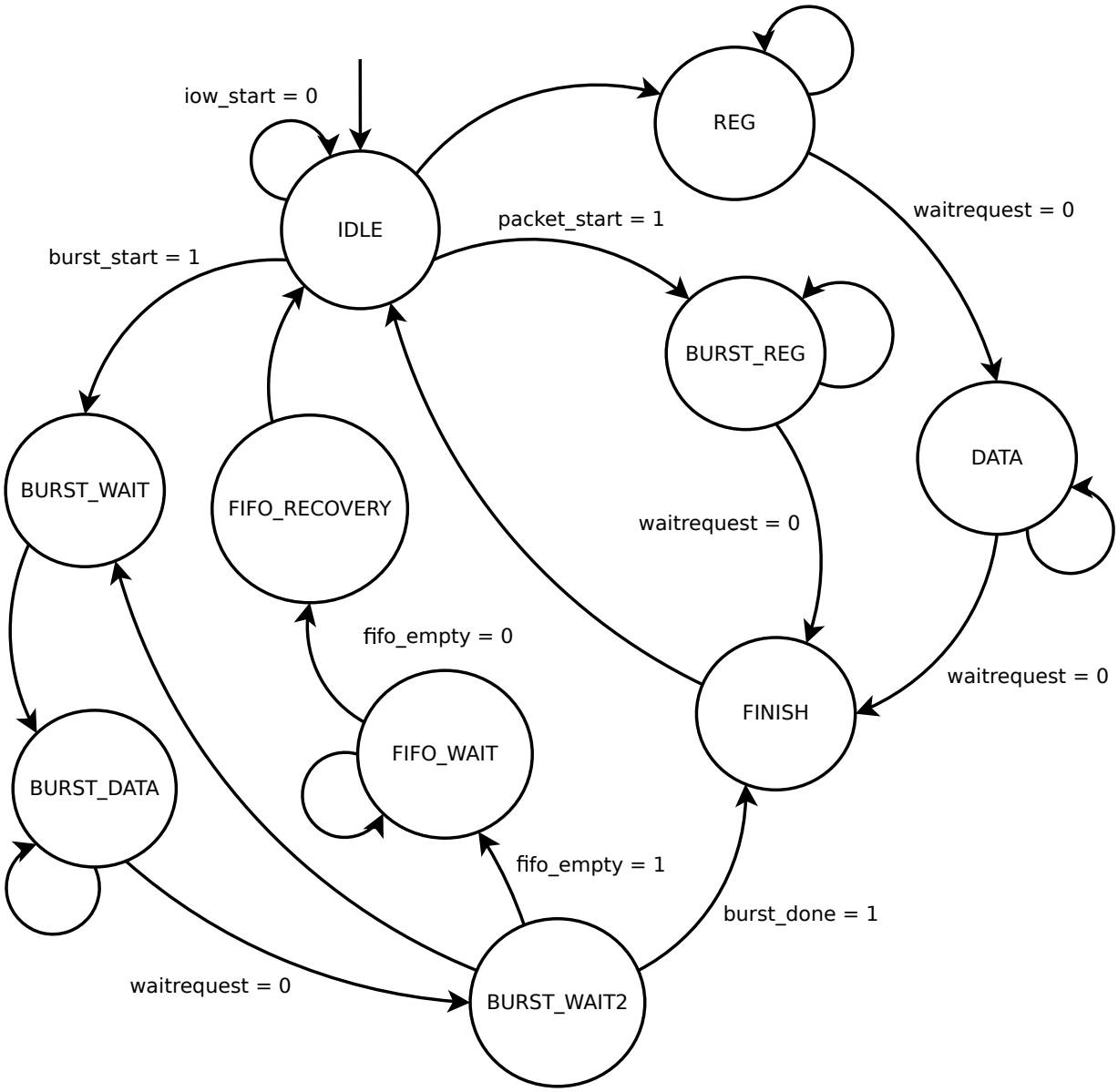
Stavový automat

Stavový automat setrvává standardně ve stavu IDLE a vyčkává na signál *iow_start*. Pokud je zápis povolen, přechází automat buď do režimu dávkového zápisu „burst“ nebo do klasického zápisu. V případě klasického zápisu se nejprve zapíší data do indexového registru (stav REG) a v dalším taktu do datového registru (stav DATA). Každý krok stavového automatu může být pozdržen signálem *waitrequest*. Po dokončení je stavem FINISH nastaven potvrzovací signál *iow_done*, který oznámí nadřazené entitě konec zápisu.

Zahájení dávkového zápisu je oznámeno signálem *packet_start*, po kterém je nejprve vybrán vhodný registr (stav BURST_REG, poté FINISH). V dalším kroku začíná se signálem *burst_start* (aktivní po celou dobu) samotný datový zápis, během kterého se kvůli dodržení latence střídají stavy BURST_WAIT a BURST_DATA. Stav BURST_WAIT2 slouží

pro kontrolu příznaků a podle nich buď dávkový zápis pokračuje, je ukončen nebo se vyčkává na další data z FIFO. V momentě, kdy čítač dokončí čtení z RAM a FIFO, je oznámen konec dávky signálem *burst_done*. Celý zápis je ukončem opět stavem FINISH s potrvzovacím signálem *iow_done*.

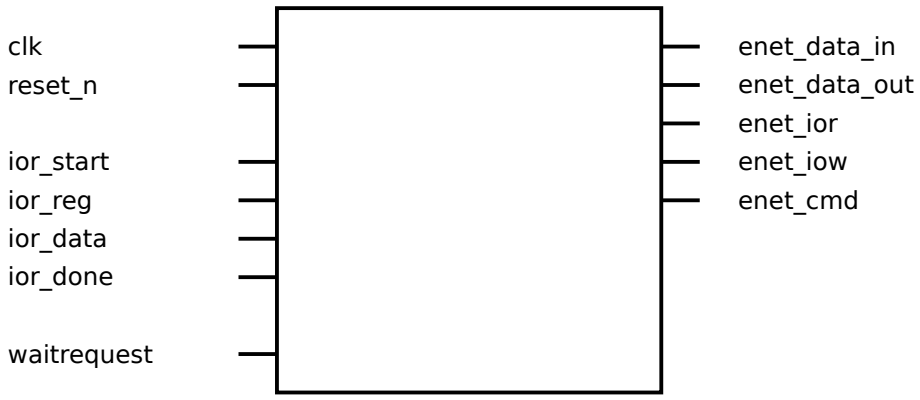
Nastaneli situace, kdy dojdou data ve FIFO, pozastaví se čítač a s ním i čtení z paměti. Stavový automat tak přechází do stavu FIFO_WAIT, ve kterém čeká do doby, než budou v paměti dostupná další data. Poté se stavem FIFO_RECOVERY provede jeden čtecí cyklus pro načtení aktuálních dat a dávkový zápis může pokračovat.



Obrázek 13: Stavový automat Controller_iow

5.4.4 Entita Controller_ior

Tato entita obsahuje poměrně jednoduchý stavový automat, který zajišťuje čtení z ethernetového řadiče. Kromě čtecích jsou zde i zápisové signály. Ve čtecím cyklu je nejprve nutné zapsat index registru a až následně přečíst jeho obsah.



Obrázek 14: Entita Controller_ior

Tabulka 8: Popis signálů entity Controller_ior

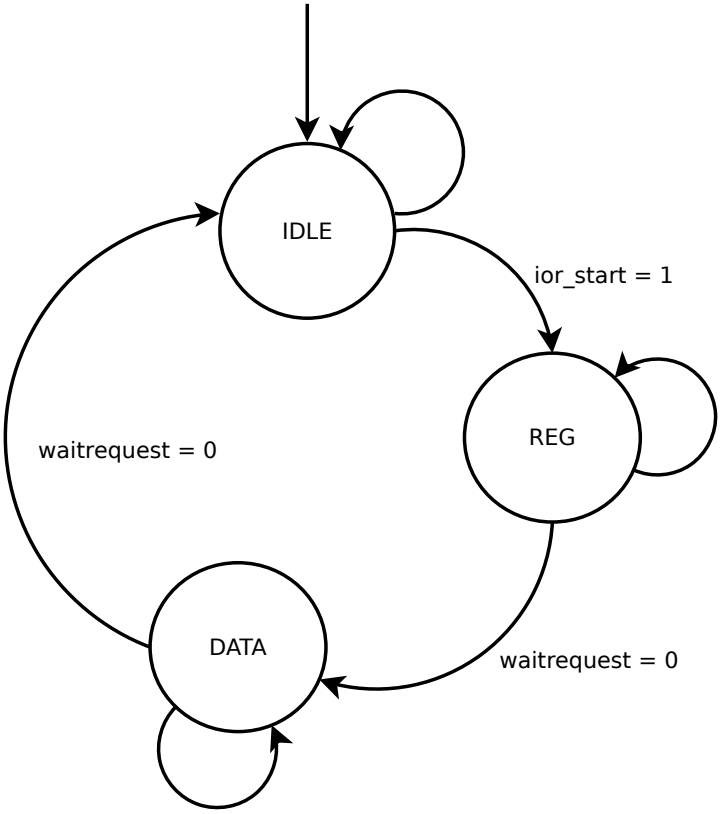
Název	**Směr**	**Typ**	**Šířka**	**Popis**
clk	in	std_logic	1	hodinový signál
reset_n	in	std_logic	1	asynchronní reset
ior_start	in	std_logic	1	oznámení o zahájení čtecího cyklu
ior_reg	in	std_logic_vector	16	hodnota indexu registru
ior_data	out	std_logic_vector	16	čtená data
ior_done	out	std_logic	1	oznámení o dokončení čtecího cyklu
waitrequest	in	std_logic	1	signál pro pozdržení přenosu
enet_data_in	in	std_logic_vector	16	vstupní data
enet_data_out	out	std_logic_vector	16	výstupní data
enet_ior	out	std_logic	1	čtecí signál
enet_iow	out	std_logic	1	zápisový signál
enet_cmd	out	std_logic	1	adresový signál

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Stavový automat

Stavový automat setrvává standardně ve stavu IDLE a vyčkává na signál *ior_start*. Ve stavu REG se do řadiče zapíše index registru, následně se stavem DATA přečte samotný obsah

registru. Přechod mezi stavy může být pozdržen signálem *waitrequest*.

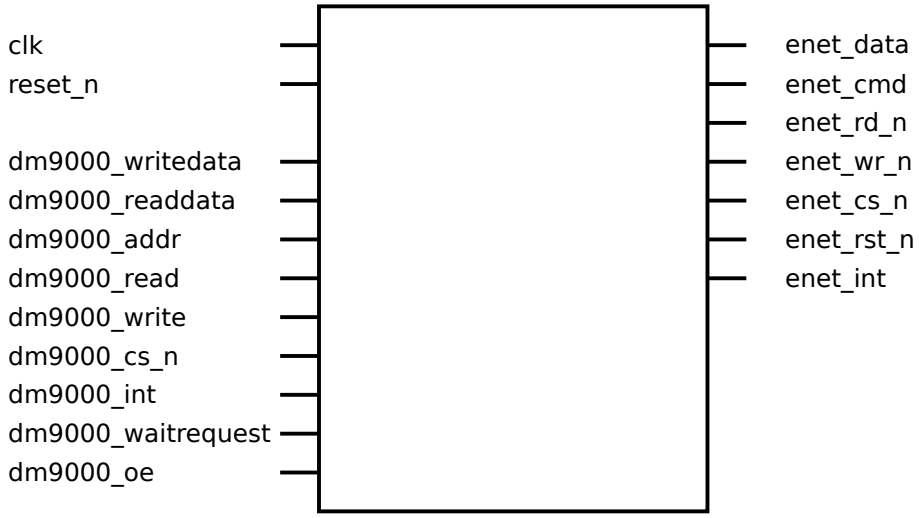


Obrázek 15: Stavový automat Controller_ior

5.4.5 Entita DM9000

Úkolem entity DM9000 je vhodná úprava signálů jdoucích na výstupní piny a následně do ethernetového řadiče. Součástí entity je stavový automat s registrem na výstupu, který ovládá IO FPGA.

Ethernetový řadič vzorkuje data na náběžnou hranu zápisového nebo čtecího signálu. Není tedy možné, aby byly tyto signály stále aktivní, docházelo by tím k zápisu pouze posledního vzorku dat. Tento problém odstraňuje právě entita DM9000, která tvoří rozhraní mezi signály vhodnými pro Avalon a signály pro řadič.



Obrázek 16: Entita DM9000

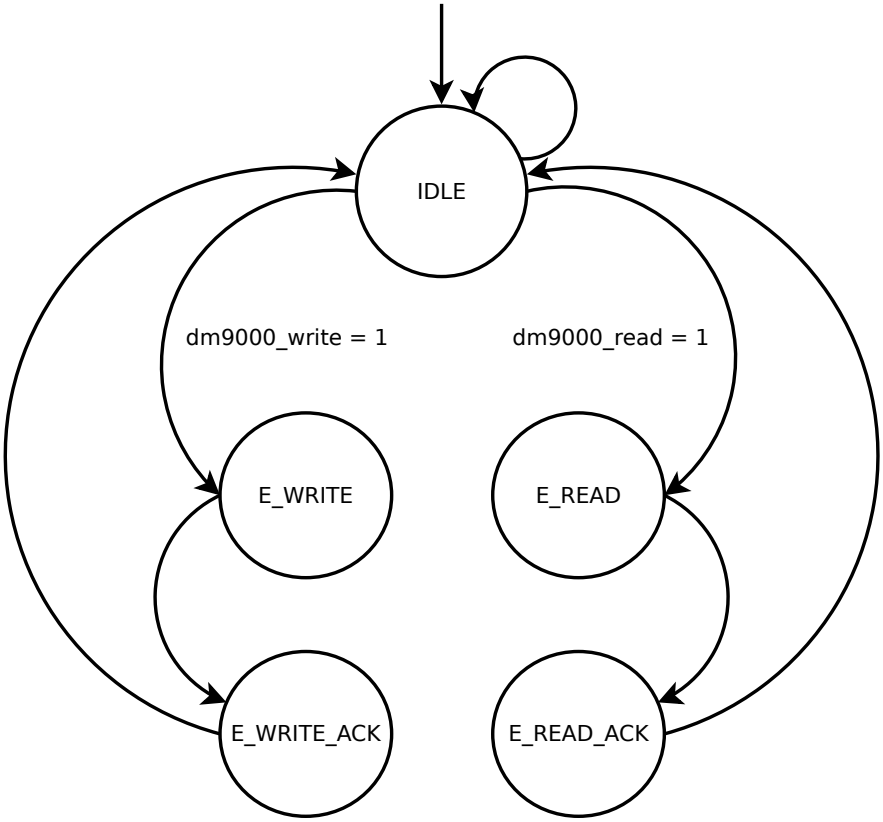
Tabulka 9: Popis signálů entity DM9000

Název	**Směr**	**Typ**	**Šířka**	**Popis**
clk	in	std_logic	1	hodinový signál
reset_n	in	std_logic	1	asynchronní reset
dm9000_writedata	in	std_logic_vector	16	data určená pro zápis
dm9000_readdata	out	std_logic_vector	16	čtená data
dm9000_addr	in	std_logic	1	adresový signál
dm9000_read	in	std_logic	1	čtecí signál
dm9000_write	in	std_logic	1	zápisový signál
dm9000_cs_n	in	std_logic	1	výběr řadiče
dm9000_int	out	std_logic	1	přerušení ze strany řadiče
dm9000_waitrequest	out	std_logic	1	signál pro pozdržení přenosu
dm9000_oe	in	std_logic	1	povolení výstupu
enet_data	inout	std_logic_vector	16	vstupní/výstupní data
enet_cmd	out	std_logic	1	adresový signál
enet_rd_n	out	std_logic	1	čtecí signál
enet_wr_n	out	std_logic	1	zápisový signál
enet_cs_n	out	std_logic	1	výběr řadiče
enet_rst_n	out	std_logic	1	asynchronní reset
enet_int	in	std_logic	1	přerušení ze strany řadiče

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Stavový automat

Stavový automat setrvává standardně ve stavu IDLE. S aktivním signálem *dm9000_write* je zahájen zápisový cyklus. Stav E_WRITE drží zápisový signál aktivní a zároveň vystavuje data na výstupu. Následuje E_WRITE_ACK, který změní stav zápisového signálu - tím se vytvoří náběžná hrana a data se zapíší, zároveň jsou platná data vystavena ještě po dobu jednoho taktu. Během zápisu je pozdržen přenos signálem *dm9000_waitrequest*, který povolí další data až stavem E_WRITE_ACK. Průběh čtecího cyklu je analogický zápisovému.



Obrázek 17: Stavový automat DM9000

Časový průběh zápisu do registru

Na obrázku 18 je znázorněn časový průběh zápisu do registru, jedná se tedy o dva zápisové cykly entity DM9000. V tomto případě chceme zapsat do registru s indexem 0x0001 hodnotu 0x00FF. Předpokládejme, že řadič je vybrán a signál *reset_n* je neaktivní. V následujících bodech jsou popsány jednotlivé hodinové takty:

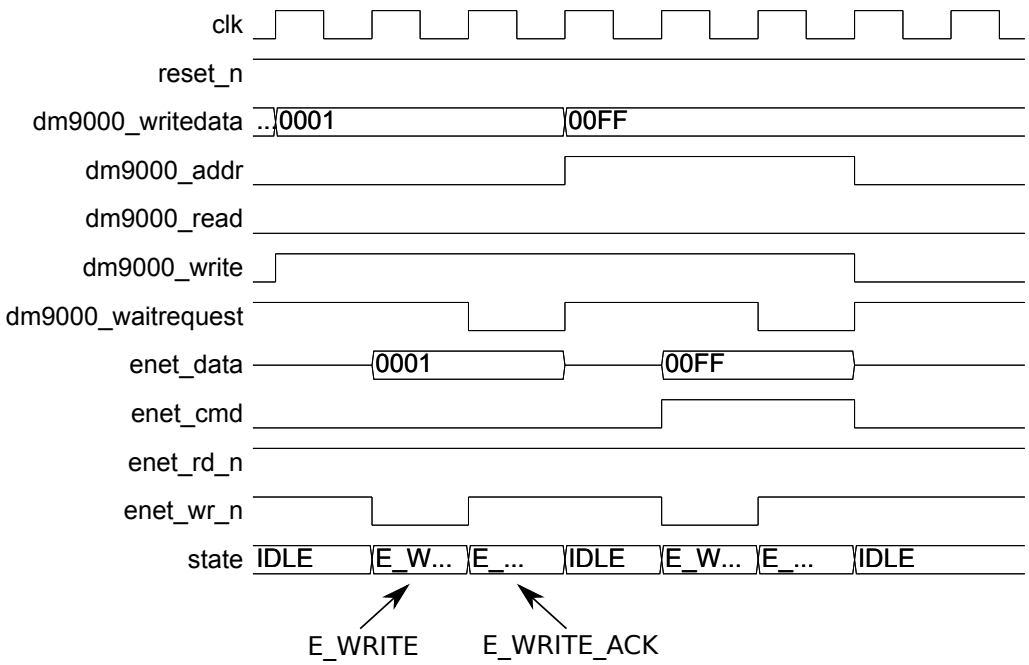
1. S trvale aktivním signálem *dm9000_write* jsou přivedena i data *dm9000_writedata*, adresový vodič *dm9000_addr* je v 0, tzn. budeme zapisovat do indexového registru. Stavový automat se zatím stále nachází ve stavu IDLE. Signál *dm9000_waitrequest* pozdržuje přísun dalších dat z nadřazené entity.

2. S následující náběžnou hranou přechází stavový automat do stavu E_WRITE. V tomto okamžiku jsou na výstup entity přivedena pomocí sběrnice *enet_data* data 0x0001, adresový signál *enet_cmd* a zápisový signál *enet_wr_n* ve svém aktivním stavu 0.

3. Nyní se stav mění na E_WRITE_ACK. Spolu s ním přejde do neaktivního stavu i zápisový signál *enet_wr_n*, tím vznikne náběžná hrana, která do řadiče zapíše. Data jsou kvůli časovým požadavkům ještě tento takt vystavena. Singnál pro pozdržení přenosu *dm9000_waitrequest* je nyní neaktivní a říká tak nadřazené entitě, že může poslat další data.

4. Stavový automat přechází opět do stavu IDLE, následují další data určená pro datový registr. Postup je identický s předchozími kroky s tím rozdílem, že adresový signál *dm9000_addr* resp. *enet_cmd* má hodnotu 1.

Z časového průběhu signálů je patrné, že do ethernetového řadiče lze s využitím signálu *dm9000_waitrequest* zapisovat podstatně rychleji než při vkládání zpoždění 20 us mezi jednotlivé zápisové cykly. Časové požadavky řadiče jsou dodrženy při periodě hodinového signálu *clk* 20 ns.



Obrázek 18: Entita DM9000 - časový průběh zápisu do registru

Časový průběh čtení z registru

Čtení z registru ethernetového řadiče se opět skládá ze dvou cyklů. V prvním kroku jde o klasický zápis do indexového registru, ten je shodný jako v předchozím případě a je zbytečné jej znovu popisovat. Takt po dokočení zápisu jsou na sběrnici vystavena odpovídající data. Čtecí část má tedy následující průběh:

1. Zápis do indexového registru byl dokončen, stavový automat se nachází v IDLE a signál *dm9000_waitrequest* pro pozdržení přenosu je aktivní.

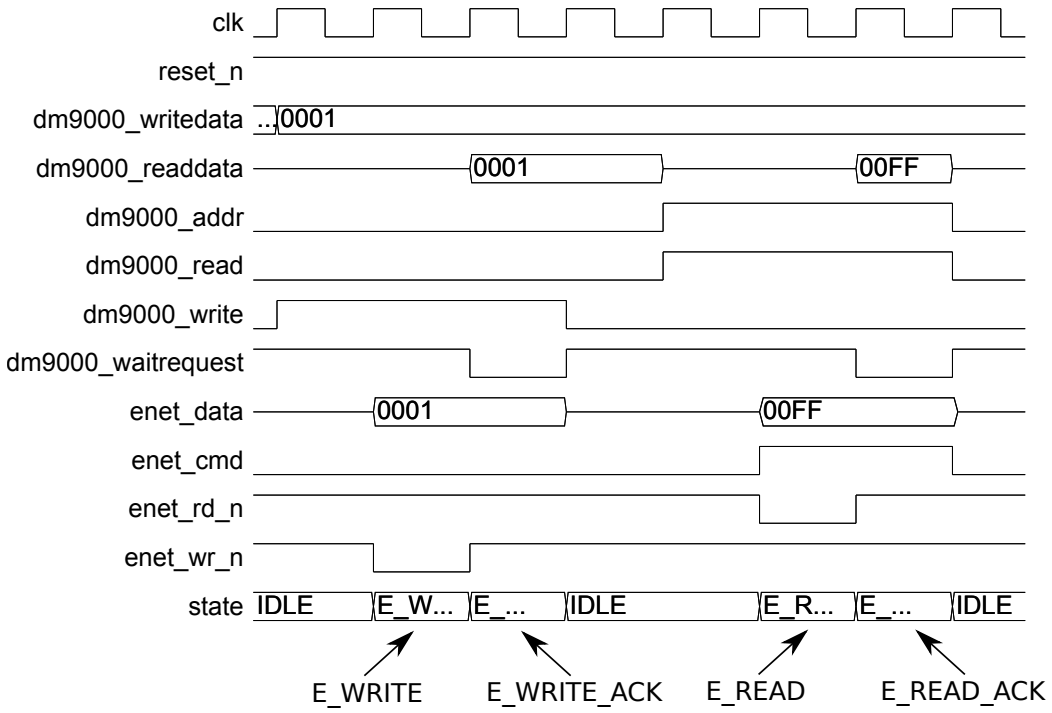
2. S další náběžnou hranou hodinového signálu se stane aktivní čtecí signál *dm9000_read* a adresový signál *dm9000_addr* nyní ukazuje na datový registr.

3. V tomto taktu se signály dostávají z entity na výstup, stavový automat je ve stavu E_READ a čtecí signál *enet_rd_n* je ve svém aktivním stavu. Ethernetový řadič současně vystavuje na sběrnici *enet_data* data uložená v adresovaném registru.

4. Nyní se stav mění na E_READ_ACK. Spolu s ním přejde do neaktivního stavu i čtecí signál *enet_rd_n*, tím vznikne náběžná hrana, která data přečte. Čtená data se

v tomto okamžiku přenáší pomocí *dm9000_readdata* na vstup entity. Přestože jsou data *dm9000_readdata* zpožděnou kopií všeho, co se na sběrnici objeví, je nutné pro správnou funkci řadiče zachovat požadované průběhy čtecích signálů. Singnál pro pozdržení přenosu *dm9000_waitrequest* je nyní neaktivní a říká tak nadřazené entitě, že může číst další data.

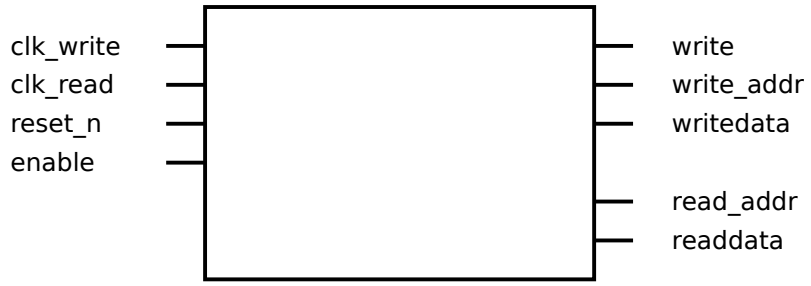
5. Stavový automat přechází opět do stavu IDLE.



Obrázek 19: Entita DM9000 - časový průběh čtení z registru

5.4.6 Entita RAM

Entita tvoří dvoubránovou RAM, do které jsou po resetu z Nios II nahrány potřebná data. Na jeden port jsou přivedeny signály z Avalon-MM Slave, zápis zde probíhá pouze pokud je aktivní signál *enable*. Na druhém portu je připojena entita Controller_iow, která z RAM data čte.



Obrázek 20: Entita RAM

Tabulka 10: Popis signálů entity RAM

Název	**Směr**	**Typ**	**Šířka**	**Popis**
clk_write	in	std_logic	1	hodinový signál pro zápis
clk_read	in	std_logic	1	hodinový signál pro čtení
reset_n	in	std_logic	1	asynchronní reset
enable	in	std_logic_vector	2	povolení činnosti
write	in	std_logic	1	zápisový signál
write_addr	in	std_logic_vector	5	adresový signál pro zápis
writedata	in	std_logic_vector	32	data určená pro zápis
read_addr	in	std_logic_vector	5	adresový signál pro čtení
readdata	out	std_logic_vector	32	čtená data

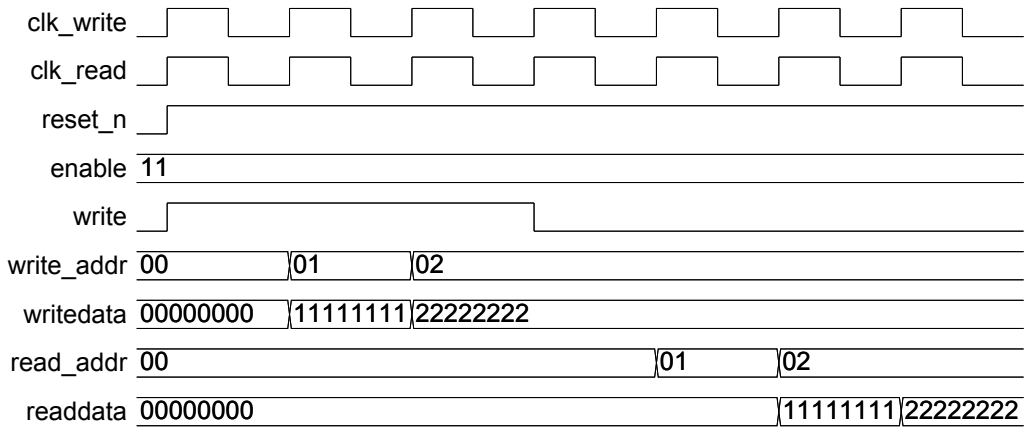
Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Časový průběh zápisu a čtení

Paměť je dvoubránová, na obě brány je však přiveden stejný hodinový signál, proto jsou signály *clk_write* a *clk_read* identické. Pro zápis do paměti je nutné, aby měl povolovací signál *enable* odpovídající hodnotu, v tomto případě 11b.

Pokud je zápisový signál *write* aktivní, jsou s každou náběžnou hranou signálu *clk_write* zapisována data ze sběrnice *writedata* do paměti, adresa je dána sběrnicí *write_addr*. Na obr. 21 vidíme postupný zápis dat 0x00000000, 0x11111111, 0x22222222 na odpovídající adresu 0x00, 0x01 a 0x02.

V případě čtení je situace o něco jednodušší, na adresovou sběrnici *read_addr* stačí přivést vhodnou adresu a o takt déle jsou data dostupná na datové sběrnici *readdata*. Není tedy potřeba žádného čtecího signálu.



Obrázek 21: Entita RAM - časový průběh zápisu a čtení

5.4.7 Entita FIFO

Jedná se vyrovnávací paměť FIFO vygenerovanou pomocí nástroje *MegaWizard Plug-In Manager* ve vývojového prostředí Quartus II. Využíván je pouze jeden hodinový signál pro zápis i čtení, o stavu paměti informují signály *empty*, *full* a *usedw*. Šířka paměti je 32 bitů.

Obrázek 22: Entita FIFO

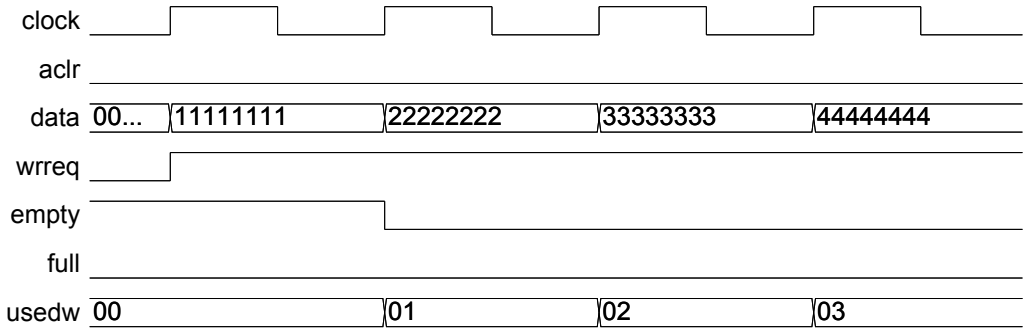
Tabulka 11: Popis signálů entity FIFO

Název	**Směr**	**Typ**	**Šířka**	**Popis**
aclr	in	std_logic	1	asynchronní reset
clock	in	std_logic	1	hodinový signál
data	in	std_logic_vector	32	vstupní data
rdreq	in	std_logic	1	čtecí signál
wrreq	in	std_logic	1	zápisový signál
empty	out	std_logic	1	informace o stavu, kdy je paměť prádzná
full	out	std_logic	1	informace o stavu, kdy je paměť plná
q	out	std_logic_vector	32	výstupní data
usedw	out	std_logic_vector	5	informace o aktuálním počtu slov v paměti

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.

Časový průběh zápisu

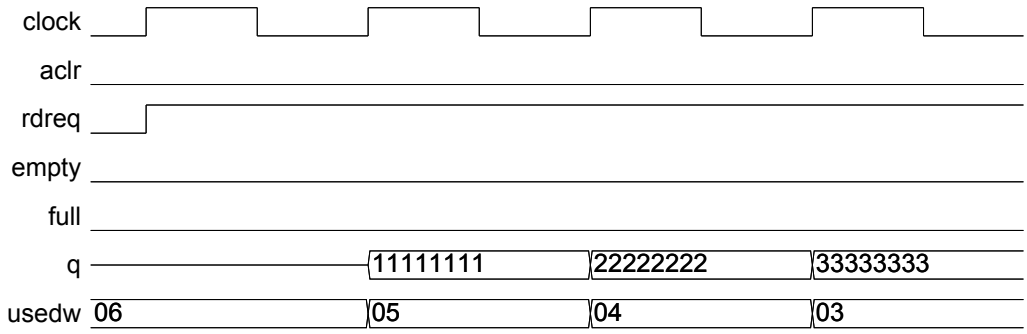
Zápis do FIFO začíná s náběžnou hranou hodinového signálu *clock* a aktivním zápisovým signálem *wrreq*. V tento okamžik se *data* zapisují, signál *empty* zatím značí prázdnou paměť a počet slov *usedw* je také 0. V následujícím taktu přestává být signál *empty* aktivní a počet slov *usedw* se s každým zápisem inkrementuje. Časový průběh zápisu je znázorněn na obr. 23.



Obrázek 23: Entita FIFO - časový průběh zápisu

Časový průběh čtení

Pro čtení z paměti slouží signál *rdreq*, pokud je aktivní, jsou následující takt data vystaveny na výstupu *q*. S každým čtením se dekrementuje počet slov *usedw*. Časový průběh zápisu je znázorněn na obr. 24.



Obrázek 24: Entita FIFO - časový průběh čtení

5.5 Inicializace pomocí Nios II

Aplikace v Nios II slouží k inicializaci ethernetového řadiče a konfiguraci komponenty, v tabulce 12 jsou popsány její dva adresáře s několika důležitými soubory. Část souborů s funkcemi určenými pro ethernetový řadič byla převzata z ukázkové aplikace [15]. Obsluhu a konfiguraci komponenty včetně definování IP a UDP hlavičky bylo nutné vytvořit.

Tabulka 12: Adresářová struktura Nios II

Adresář	**Soubor**	**Popis**
ethernet	DM9000A.h	hlavičkový soubor pro řadič DM9000A
	DM9000A.c	zdrojový soubor pro řadič DM9000A
	ethernet.h	hlavičkový soubor pro komponentu Ethernet
	ethernet.c	zdrojový soubor pro komponentu Ethernet
ethernet_bsp	system.h	hlavičkový soubor s konfigurací Qsys

Adresář *ethernet* obsahuje vlastní projekt se soubory určenými pro konfiguraci ethernetového řadiče a komponenty. Druhý adresář *ethernet_bsp* je systémová knihovna, kterou generuje Qsys. Zkratka *bsp* v názvu znamená *Board Support Package*. Nachází se zde důležitý hlavičkový soubor *system.h*, ve kterém jsou definována makra pro použití v Nios II aplikaci. Patří mezi ně např. typ komponenty, bázová adresa, číslo přerušení, šířka sběrnice atd. Tuto systémovou knihovnu je nutné vždy při změně Qsys systému znovu vygenerovat a zkompilovat.

5.5.1 Inicializace komponenty

Součástí navržené komponenty je registr *ENABLE_REG* (adresa 0x7C), který slouží pro adresování jednotlivých entit a přepínání multiplexeru určeného pro výběr výstupu. Popis možných stavů je znázorněn v tabulce 13. Další řídící registry jsou obsaženy v entitě Controller, zde se spouští stavový automat, nastavuje se délka paketu a počet čtených slov z FIFO. Aby bylo možné k těmto registrům přistupovat, musí být v registru *ENABLE_REG* hodnota *ENABLE_CONTROLLER*, tím je povolen zápis do entity.

V tabulce 14 jsou popsány registry entity Controller. Řídící registr *CONTROLLER_CSR* obsahuje spuštěcí bit pro stavový automat *fsm_enable*, pro jeho aktivaci je nutné zapsat hodnotu 0x01. V registru *CONTROLLER_TX_LEN* je uložena celková délka paketu, ta se pak během zápisu do registrů ethernetového řadiče dělí na horní a dolní slovo. Počet čtení z paměti FIFO pro jeden rámec je omezen hodnotou zapsanou v registru *CONTROLLER_FIFO_LIMIT*, tato hodnota je vypočítána z délky datové části *payload* a dále závisí na šířce FIFO. Pokud máme např. datovou část *payload* 32 bajtů, bude při šířce FIFO 4 bajty nutné vykonat 8 čtecích cyklů, tato hodnota je tedy zapsána do registru.

Tabulka 13: Popis hodnot registru ENABLE_REG

Hodnota	**Název**	**Popis**
0x00	ENABLE_STOP	komponenta je zastavena
0x01	ENABLE_CONDUIT	komponenta je v režimu Conduit
0x02	ENABLE_CONTROLLER	komponenta je v režimu Controller
0x03	ENABLE_RAM	řežim zápisu do RAM

Tabulka 14: Popis registrů entity Controller

Název	**Adresa**	**Šířka**	**Popis**
CONTROLLER_CSR	0x00	8	řídící registr pro stavový automat
CONTROLLER_TX_LEN	0x04	16	délka paketu v bajtech
CONTROLLER_FIFO_LIMIT	0x08	16	počet čtení z FIFO

Šířka signálu je uváděna v bitech.

5.5.2 Inicializace ethernetového řadiče

Součástí inicializace ethernetového řadiče DM9000A jsou již připravené funkce z ukázkové aplikace od společnosti Terasic Technologies. Jedinou modifikací je odstranění zpoždění 20 us mezi jednotlivými zápisovými a čtecími cykly. Důvodem je lépe řešené generování řídících signálů v komponentě, zpoždění tedy není třeba.

Hlavičkový soubor *DM9000A.h* obsahuje následující definice maker:

- offset registrů řadiče
- konfigurační hodnoty registrů řadiče
- zdrojové a cílové MAC adresy pro ethernetový rámec
- zdrojové a cílové adresy pro IP paket, výpočet kontrolního součtu
- údaje pro UDP datagram

Pro inicializaci ethernetového řadiče stačí ve funkci *main()* zavolat funkci *DM9000_init()*. Následně proběhne konfigurace registrů, včetně fyzické vrstvy PHY. Seznam použitých funkcí je uveden v tabulce 15.

Tabulka 15: Přehled funkcí pro inicializaci řadiče

Návratový typ	**Deklarace**	**Popis**
void	iow(unsigned int reg, unsigned int data)	zápis do registru
unsigned int	ior(unsigned int reg)	čtení z registru
void	phy_write(unsigned int reg, unsigned int value)	zápis do registru PHY
unsigned int	DM9000_init(void)	inicializace řadiče

5.5.3 Zápis a čtení pomocí Nios II

Před popisem kompletní inicializace je ještě třeba zmínit, jakým způsobem lze pomocí Nios II číst či zapisovat do komponenty. K tomuto účelu slouží funkce *IOWR()* pro zápis a *IORD()* pro čtení, v těchto případech je automaticky rozhodnuto, jaká šířka sběrnice se zvolí. Pokud chceme ručně použít funkci pro konkrétní šířku, použijeme funkci *IOWR_32DIRECT()* resp. *IORD_32DIRECT()* pro 32bit sběrnici. Parametry funkcí obsahují bázovou adresu, offset a v případě zápisu i data.

Zápis hodnoty 0x11223344 na bázovou adresu *COMPONENT_BASE* s offsetem 0x00 a její následné přečtení je znázorněno ve výpisu 4.

Výpis 4 Zápis a čtení pomocí Nios II

```
alt_u32 data;
IOWR_32DIRECT(COMPONENT_BASE, 0x00, 0x11223344); //write to Avalon
data = IORD_32DIRECT(COMPONENT_BASE, 0x00); //reading from Avalon
```

5.5.4 Popis kompletní inicializace

Kompletní incializace se nachází v souboru *ethernet.c*, kde je postupně konfigurován ethernetový řadič a následně komponenta. Součástí je i pole *MyPacket* složené z údajů, které dohromady tvoří ethernetovou hlavičku, IP hlavičku a hlavičku UDP datagramu. Tato data jsou nahrána do paměti RAM v komponentě a jsou k dispozici po celou dobu provozu. Celý proces inicializace lze shrnout do následujících kroků:

1. Přepnutí komponenty do režimu Conduit
2. Inicializace ethernetového řadiče
3. Čekání na informaci o správně zapojeném kabelu
4. Přepnutí komponenty do režimu pro zápis do RAM
5. Nahrání dat z pole *MyPacket* do RAM
6. Přepnutí komponenty do režimu Controller
7. Zapis délky paketu do registru *CONTROLLER_TX_LEN*
8. Zapis velikosti datové části do registru *CONTROLLER_FIFO_LIMIT*
9. Spuštění hlavního stavového automatu zápisem do registru *CONTROLLER_CSR*

5.6 Komponenta Data_generator

Pro účely otestování rychlosti ethernetu byla vytvořena jednoduchá komponenta s rozhraním Avalon-MM Master. Jedná se o čítač, který cyklicky vydává přednastavená data na výstup

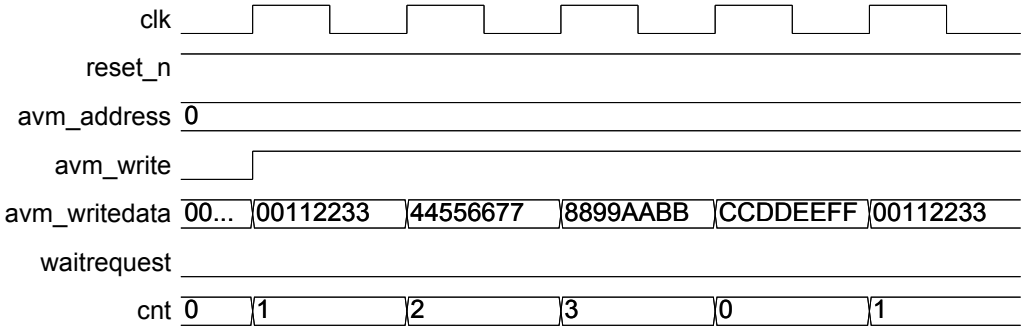
a simuluje tak stream dat pro Ethernet komponentu. Signálem *waitrequest* je možné vysílání dat pozastavit. Detailní popis všech signálů je v tabulce 16, časový průběh signálů pak na obr. 26.

Obrázek 25: Komponenta Data_generator

Tabulka 16: Popis signálů komponenty Data_generator

Název	**Směr**	**Typ**	**Šířka**	**Popis**
reset_n	in	std_logic	1	asynchronní reset
clk	in	std_logic	1	hodinový signál
avm_address	out	std_logic_vector	32	adresový signál
avm_write	out	std_logic	1	zápisový signál
avm_writedata	out	std_logic_vector	32	výstupní data
waitrequest	in	std_logic	1	signál s žádostí o pozdržení přenosu

Sufix _n v názvu značí, že je signál aktivní v 0.
Šířka signálu je uváděna v bitech.



Obrázek 26: Komponenta Data_generator - časový průběh zápisu

5.7 Naměřené výsledky

Při měření datové propustnosti Ethernet komponenty a řadiče byly zvoleny různé délky ethernetových rámců. Měření probíhalo pomocí nástroje *iptraf-ng* pro sledování, zaznamená-

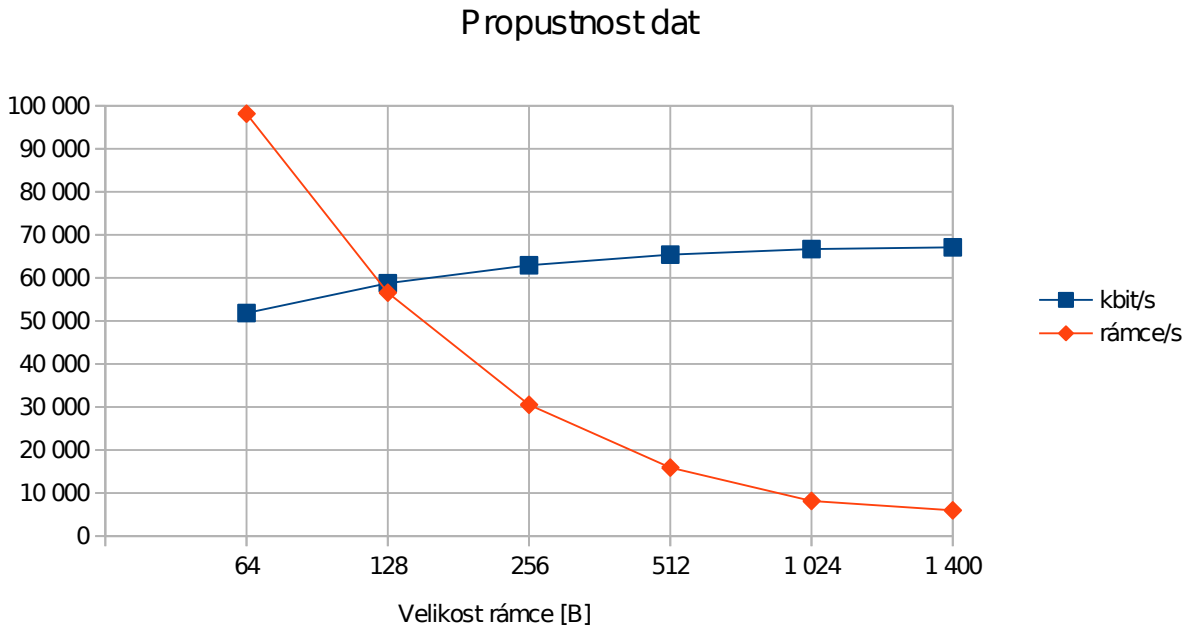
vání a analýzu síťového provozu v reálném čase. Pro stream dat byla využita komponenta Data_generator, která byla stejně jako Ethernet komponenta taktována hodinami o frekvenci 50 MHz. Informace o aktuálním datovém toku a počtu rámců za sekundu byly zaznamenány do tabulky 17.

Tabulka 17: Naměřené hodnoty

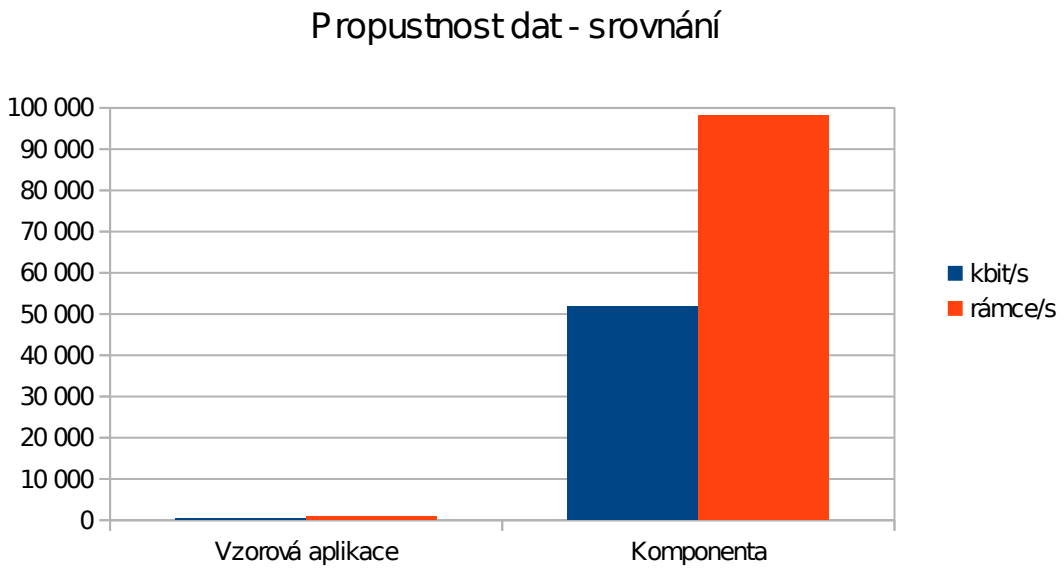
Velikost rámce [B]	64	128	256	512	1 024	1 400
kbit/s	51 830	58 750	62 940	65 400	66 700	67 100
rámce/s	98 150	56 560	30 520	15 920	8 160	5 980

V grafu 27 jsou znázorněny naměrené výsledky. Jedna datová řada reprezentuje datový tok a druhá řada vyjadřuje počet rámců za sekundu v závislosti na velikosti rámce. Z naměřených hodnot je vidět, že s rostoucí velikostí rámce strmě klesá počet odeslaných rámců v jedné sekundě. Datová propustnost je při vyšších velikostech rámce téměř konstantní.

Pro srovnání Ethernet komponenty se vzorovou aplikací od firmy Terasic Technologies [15] byly naměřené hodnoty vyneseny do sloupcového grafu 28. Komponenta i ukázka vysílaly stejně velká data o velikosti 64 bajtů, v případě ukázky bylo odesílání realizováno pomocí procesoru Nios II. Rozdíl v propustnosti dat je znatelný.



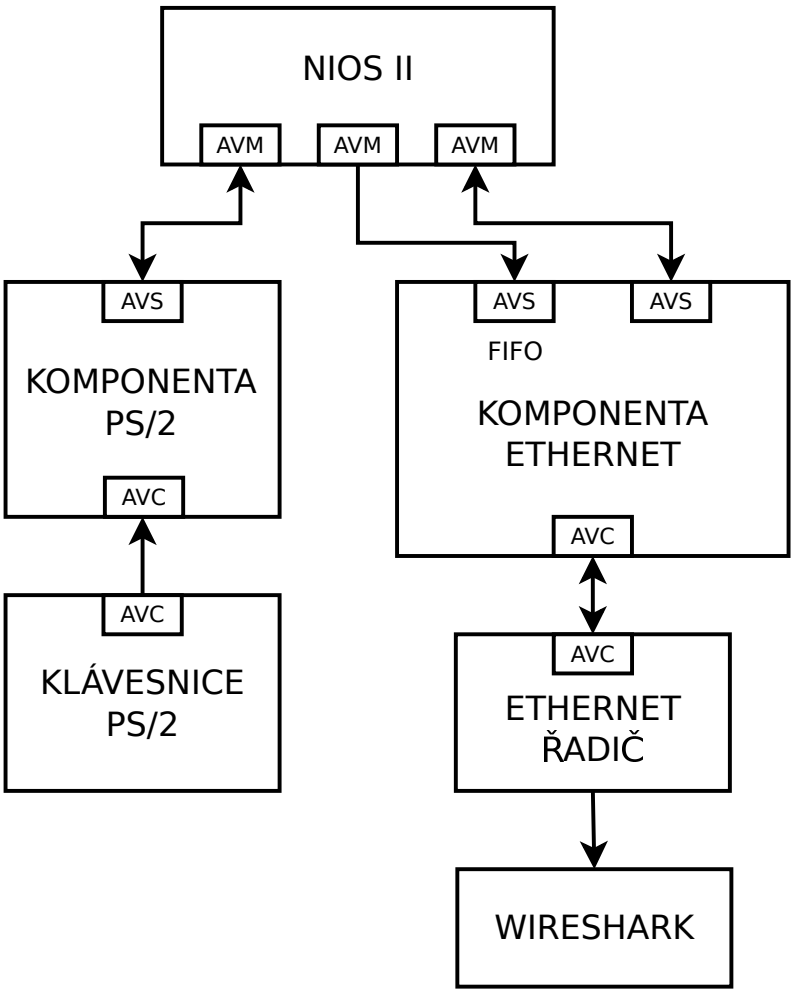
Obrázek 27: Graf naměřených výsledků



Obrázek 28: Srovnání vzorové aplikace a Ethernet komponenty

6 Ukázková aplikace

Ukázková aplikace se skládá z obou komponent, kde rozhraní klávesnice PS/2 tvoří stream dat, která jsou následně odesílána Ethernet komponentou ve formě UDP datagramů. Ty je možné pozorovat např. nástrojem Wireshark pro sledování a analyzování síťové komunikace. Blokové schéma ukázkové aplikace je znázorněno na obr. 29.



Obrázek 29: Blokové schéma ukázkové aplikace

Ukázková aplikace posílá stisk libovolné klávesy na připojené klávesnici s rozhraním PS/2. Data z klávesnice jsou zpracována PS/2 komponentou a po ověření správnosti je vyvoláno přerušení, tím je Nios II informován o přijatém scan kódu. V obsluze přerušení je scan kód přečten a poté filtrován stavovým automatem. Na rozdíl od řešení uvedeného v kapitole 4.2.2 se data netisknou jen do konzole, ale jsou posílána i na adresu FIFO, které tvoří vstupní

frontu v Ethernet komponentě. Délka posílaných dat závisí na typu scan kódu, po filtraci a odstranění *break* kódu tedy dostaneme jeden bajt dat, resp. dva bajty v případě druhého typu kódu.

Pro demonstraci využití ethernetu je použito UDP datagramů. Pomocí maker jsou definovány hlavičky IP a UDP, které se při incializaci nahrají do RAM Ethernet komponenty a jsou tak po dobu provozu trvale k dispozici. Pro správnou funkci je nutné v hlavičkovém souboru *DM9000A.h* nastavit MAC a IP adresu vývojového kitu a cílového zařízení, na kterém budeme síťový provoz sledovat.

Spolu s tiskem scan kódu do konzole se tedy data odesílají i na adresu FIFO v Ethernet komponentě. Protože je šířka FIFO a tím i délka UDP dat 4 bajty, zůstává část slova při odesílání kódů nevyužita. Výhodou však zůstává fakt, že každý stisk klávesy odpovídá právě jednomu UDP datagramu.

Přijatá data lze sledovat nástrojem Wireshark, kde jsou ethernetové rámce vypisovány v chronologickém pořadí. Základní informací je zdrojová a cílová IP adresa, typ protokolu a délka rámce v bajtech. V detailních informacích lze mimo jiné zobrazit data v hexadecimální, popř. binární podobě.

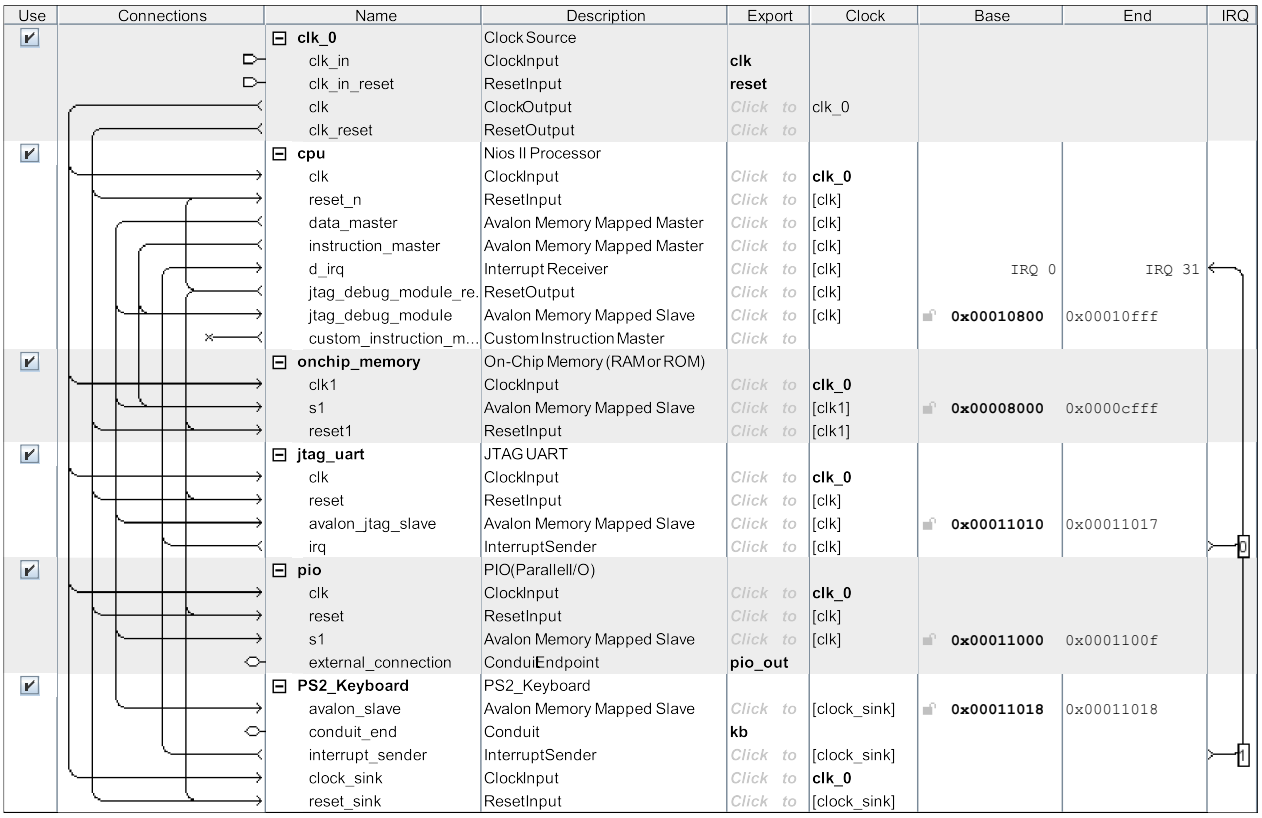
7 Návod na použití vytvořených komponent

Po importu do Qsys vznikne v seznamu komponent nová kategorii Diplomka, ve které se nachází konkrétní importovaná komponenta. Nyní stačí komponentu přidat do námi vytvořeného Qsys systému. Pro správnou spolupráci s Nios II je nutné zachovat vždy původní názvy komponent. Výjimkou je přípona s pořadovým číslem, např. „_0“, která se do názvu automaticky vloží a je třeba jí odstranit. Nedodržení tohoto požadavku by vedlo k tomu, že by se musely přejmenovávat použitá makra ve zdrojových kódech pro Nios II. Dalším požadavkem je zachování exportovaných názvů rozhraní Avalon Conduit. Signál *reset* je u vytvořených Qsys systémů vyveden na přepínač SW[0], proto musí být pro správnou funkci přepnut ve stavu 1.

7.1 PS/2 komponenta

Po importu a přidání komponenty PS2_Keyboard do Qsys systému je nutné připojit všechny potřebné signály. Vstupním portem je *conduit_end*, na který je připojen hodinový a datový signál klávesnice. Na port *clock_sink* přivedeme hodinový signál, na port *reset_sink* signál reset. Port *avalon_slave* je propojen s portem *avalon_master* procesoru Nios II. Dále je zde port *interrupt_sender*, jedná se o signál s žádostí o přerušení a je připojen k procesoru.

Pro vyvedení scan kódu na LED vývojové desky může být použita paralelní brána pio, připojená a řízená v obsluze přerušení procesoru. Tento modul nepotřebuje při zachování názvů komponent žádné speciální nastavení v Nios II. Náhled celého Qsys systému se správně zapojenou komponentou je na obr. 30.



Obrázek 30: Qsys systém s PS/2 komponentou

7.2 Ethernet komponenta

Po importu a přidání komponenty Ethernet_controller do Qsys systému je nutné připojit všechny potřebné signály. Na port *clock_sink* přivedeme hodinový signál 50 MHz, na port *reset_sink* signál reset. Port *avalon_slave* musí být připojen k portu *data_master* procesoru Nios II. Stream dat, který chceme odesílat, připojíme na port *fifo_slave*. Pokud chceme zpracovávat i stavový signál informující o prázdné paměti, použijeme dle potřeby *fifo_end*. Pro vyvedení výstupních signálů na piny ethernetového řadiče slouží port *conduit_end*. Pro ethernetový řadič je důležité mít z fázového závěsu exportovaný hodinový signál o frekvenci 25 MHz. Náhled celého Qsys systému se správně zapojenou komponentou je na obr. 31.

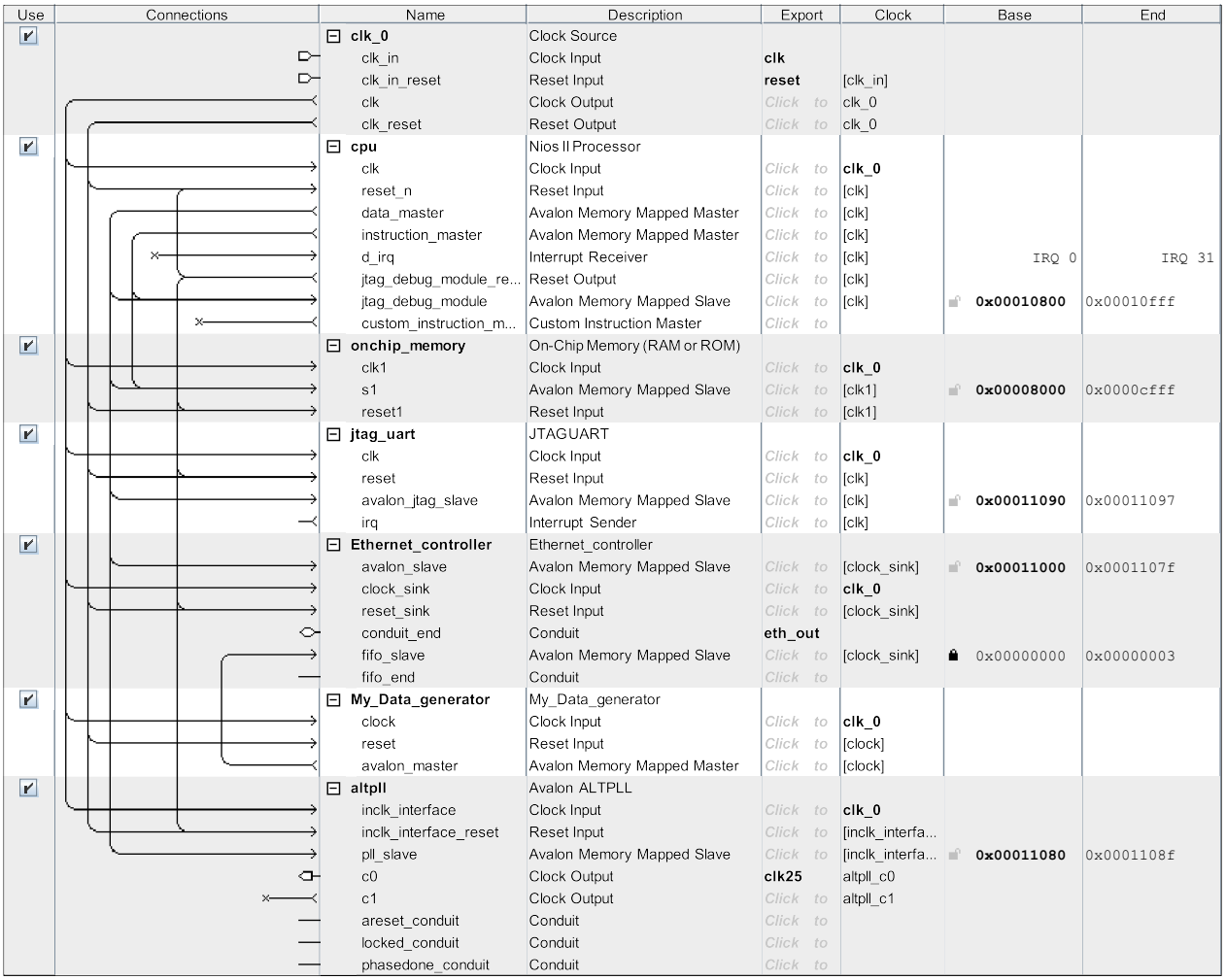
V případě, že chceme použít generátor dat, vložíme ze seznamu do systému komponentu My_Data_generator. Tato komponenta má kromě hodinového signálu a resetu pouze port *avalon_master*, který připojíme na port *fifo_slave* komponenty Ethernet_controller. Protože generátor dat posílá data pouze na adresu 0x00000000, je nutné tuto adresu u portu *fifo_slave*

nastavit a následně zamknout.

Dalším krokem je nastavení maker pro Nios II v souboru *DM9000A.h*, ve kterém jsou definovány zdrojové a cílové adresy, ty se při použití na jiných zařízení liší. Pro správnou funkci je třeba upravit údaje v tabulce 18. Definicí makra *payload* určujeme délku datové části UDP datagramu, typicky se jedná o 32 - 1400 bajtů.

Tabulka 18: Nastavení zdrojových a cílových adres v Nios II

Název makra	**Popis**
Src_IP	zdrojová IP adresa, skládá se ze 4 bajtů
Dest_IP	cílová IP adresa, skládá se ze 4 bajtů
Src_MAC	zdrojová MAC adresa, skládá se z 6 bajtů
Dest_MAC	cílová MAC adresa, skládá se z 6 bajtů



Obrázek 31: Qsys systém s Ethernet komponentou

8 Závěr

Před započetím práce jsem důkladně nastudoval a analyzoval možnosti rozhraní Avalon a jeho možné využití pro účely zadané práce. Část seznámení se týkala také osvojení si práce s vývojovými prostředky, zejména s vývojovým prostředím společnosti Altera. Většina těchto informací byla čerpána z firemních materiálů a cizojazyčné literatury.

Modul pro rozhraní klávesnice PS/2 byl úspěšně navržen. Skládá se z VHDL komponenty a softwarového filtru, který je realizován stavovým automatem v procesoru Nios II, ten umožňuje na základě přijaté sekvence kódů odstranit nadbytečné informace. Komunikace komponenty s procesorem je založena na žádosti o přerušení, které je voláno v případě přijmutí korektních dat. Procesor tak není zbytečně zatěžován neustálým čtením. V samotném VHDL kódu bylo nutné kvůli dvěma časovým doménám řešit resynchronizaci signálů. Součástí kódu je i kontrola správnosti dat, včetně situace, kdy dojde k odpojení klávesnice. V případě odpojení se vyvolá reset, který umožní klávesnici po opětovném připojení znovu používat. Samozřejmostí je signalizace chybového stavu.

Dalším úspěšně navrženým modulem je rozhraní Ethernet. Navržená komponenta nabízí uživateli předpokládané vlastnosti, a těmi jsou snadná inicializace pomocí procesoru a zároveň rychlé zpracování dat uživatelskou logikou, bez zatěžování procesoru. Komponenta obsahuje několik rozhraní, první z nich slouží pro konfiguraci, dalším rozhraním je vyrovnávací paměť FIFO určená pro vstupní data a rozhraní pro připojení k ethernetovému řadiči. Další součástí komponenty je paměť RAM, do které Nios II nahraje při konfiguraci kompletní ethernetovou hlavičku a hlavičky vyšších vrstev. Ta umožňuje díky nízké latenci a 32 bit sběrnici velmi rychlé čtení dat. Během samotného odesílání se vždy střídá čtení z těchto dvou pamětí, nejprve se přečtou statická data z paměti RAM a vzápětí se za ně připojí data z vyrovnávací paměti FIFO. Pro účely otestování datové propustnosti modulu byl vytvořen jednoduchý a rychlý generátor dat. Naměřené výsledky ukazují datovou propustnost téměř 70 000 kbit/s, to je 150krát více než výkon řešení dodávaného společností Terasic Technologies.

Z důvodu časové náročnosti nebyl realizován modul pro USB a Bluetooth. Hlavní příčinou zdržení byla chyba v nástroji Qsys, kdy docházelo k nekorektnímu dekódování adres při použití 16bitového rozhraní Avalon. Chyba se projevovala až přímo v zařízení, což značně

ztížilo její lokalizaci.

Pro demonstraci navržených modulů byla vytvořena ukázková aplikace, která se skládá z modulu rozhraní klávesnice PS/2 a modulu pro rozhraní Ethernet. Z klávesnice PS/2 jsou posílány jednotlivé stisky kláves, ty se zpracují softwarovým filtrem v Nios II a následně jsou v reálném čase odesílány po rozhraní Ethernet ve formě UDP datagramů. Správná funkčnost zpracování a odesílání dat byla ověřena na cílovém zařízení nástrojem Wireshark pro sledování a analyzování síťové komunikace.

Literatura

[1] ALTERA CORPORATION. *Avalon Bus Specification* [online]. Document Version 2.3. 2003 [cit. 2011-10-31]. Dostupné z:
`http://www.altera.com.cn/literature/manual/mnl_avalon_bus.pdf`

[2] ALTERA CORPORATION. *Internal Memory (RAM and ROM)* [online]. 2011 [cit. 2012-02-03]. Dostupné z:
`http://www.altera.com/literature/ug/ug_ram_rom.pdf`

[3] ALTERA CORPORATION. *Nios II Software Developer's Handbook* [online]. 2011 [cit. 2011-11-11]. Dostupné z:
`http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf`

[4] ALTERA CORPORATION. *Qsys System Design Tutorial* [online]. 2011 [cit. 2011-10-31]. Dostupné z:
`http://www.altera.com/literature/tt/tt_qsys_intro.pdf`

[5] ALTERA CORPORATION. *Quartus II Handbook Version 11.0* [online]. 2011 [cit. 2011-10-28]. Dostupné z:
`http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf`

[6] ALTERA CORPORATION. *USB-Blaster Download Cable* [online]. Document Version 2.5. 2009 [cit. 2011-11-11]. Dostupné z:
`http://www.altera.com/literature/ug/ug_usb_blstr.pdf`

[7] BROUWER, Andries. *Keyboard scancodes: Keyboard commands* [online]. v1.2g. 2009 [cit. 2012-04-27]. Dostupné z:
`http://www.win.tue.nl/~aeb/linux/kbd/scancodes-12.html`

[8] BROŽ, Jan. *Příjem streamovaných dat po Ethernetu pomocí FPGA*. Plzeň, 2009. Diplomová práce. Západočeská univerzita, Fakulta elektrotechnická, Katedra aplikované elektroniky a telekomunikací.

[9] BURIAN, Petr. *Vkládání obrazu do TV signálu*. Plzeň, 2007. Diplomová práce. Západočeská univerzita, Fakulta elektrotechnická, Katedra aplikované elektroniky a telekomunikací.

[10] CHAPWESKE, Adam. *The PS/2 Keyboard Interface: Scan Codes: Set 2* [online]. 2004 [cit. 2012-04-24]. Dostupné z:
`http://www.computer-engineering.org/ps2keyboard/scancodes2.html`

[11] DAVICOM SEMICONDUCTOR, Inc. *DM9000A Datasheets: Ethernet Controller with General Processor Interface* [online]. DM9000A-17-DS-F01. 2006 [cit. 2011-12-05]. Dostupné z:
`http://www.davicom.com.tw/userfile/24247/DM9000A-DS-F01-101906.pdf`

[12] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-730-0198-5.

[13] SCOVILLE, Ryan. *TimeQuest User Guide* [online]. Wiki release 1.1. 2010 [cit. 2012-03-23]. Dostupné z:
`http://www.eet.bme.hu/~nagyg/mikroelektronika/TimeQuest_User_Guide.pdf`

[14] TERASIC TECHNOLOGIES. *Altera DE2-70 User Manual* [online]. Version 1.08. 2009 [cit. 2011-12-10]. Dostupné z:
`http://www.terasic.com.tw`

[15] TERASIC TECHNOLOGIES. *Ukázkové aplikace pro Altera DE2-70* [online]. 2011 [cit. 2011-12-05]. Dostupné z:
`http://www.terasic.com/downloads/cd-rom/de2_70/DE2_70_demonstrations_V10.rar`

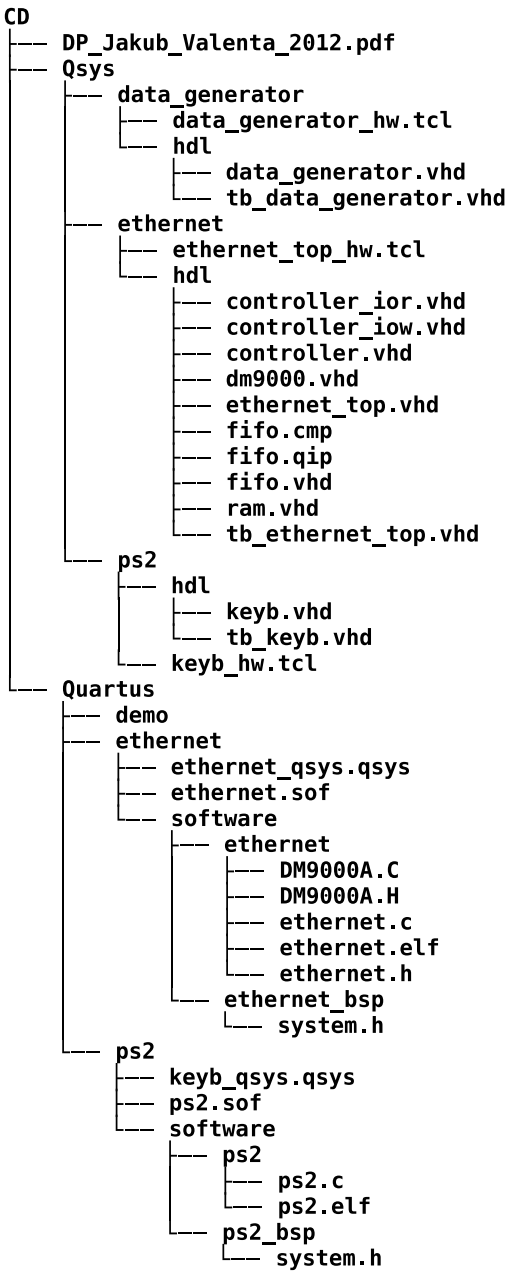
[16] VAŠÍČEK, Zdeněk. *FITkit: Řadič PS/2* [online]. © 2006 - 2012, 20.3.2009 [cit. 2011-11-06]. Dostupné z:
`http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_ps2.html`

[17] YANG, Richard. *FPGA CPLD and ASIC from Altera: 5 Reasons to Switch from SOPC Builder to Qsys* [online]. © 1995-2012 [cit. 2012-01-25]. Dostupné z: `http://www.altera.com/education/webcasts/all/wc-2011-reasons-switch-qsys.html`

9 Přílohy

9.1 Adresářová struktura CD

Na obr. 32 je zobrazena adresářová struktura přiloženého CD. Vypsány jsou pouze nejduležitější adresáře a soubory.



Obrázek 32: Adresářová struktura CD