Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalářská práce

Procesory pro FPGA obvody

Matouš Filip

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

2. ledna 2014

Poděkování

Rád bych touto formou poděkoval vedoucímu práce Ing. Pavlu Kubalíkovi, Ph.D. za veškerou podporu, rady, nápady a připomínky poskytnuté na konzultacích, jakožto i jeho trpělivost, kterou spolupráce s autorem vyžaduje.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 2. ledna 2014 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2014 Matouš Filip. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Filip, Matouš. *Procesory pro FPGA obvody*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

The thesis focuses on comparing available soft-processors and possibilities of extending them. The goal is to choose one type of processor and create additional peripherals for it. It's also planned to create background for working with the processor in the C language, ideally using already existing compiler. Part of that is also the creation of libraries for interfacing with both new and already existing peripherals and creation of a demo application, demonstrating the functionality of the whole project and the implementation of the project into an FPGA.

Keywords FPGA, soft-processor, picoBlaze, C

Abstrakt

Práce se zabývá porovnáním dostupných soft-processorů a jejich možným rozšiřováním. Cílem je vybrat jeden typ processoru a vhodně ho doplnit o některé periferie. Počítá se také s vytvořením zázemí pro práci se soft processorem v jazyce C, ideálně s využitím již existujícího překladače. Součástí

tohoto je i vytvoření knihoven pro přístup k nově vytvořeným i stávajícím periferiím a demo aplikace, demonstrující funkčnost celého projektu, a následná implementace processoru na FPGA.

Klíčová slova FPGA, soft-procesor, picoBlaze, C

Obsah

Seznam obrázků

Úvod

V současné době je dostupná poměrně široká škála soft-processorů, neboli processorů implementovaných kódem v jazyku pro popis hardwaru. Jednotlivé procesory se mohou značně lišit velikostí, funkcionalitou i dostupností. Prvním úkolem této práce je dostupné procesory prostudovat a jeden, nejlépe volně dostupný a nepříliš komplikovaný, vybrat.
Druhým úkolem je prostudovat již existující zázemí k vybranému procesoru, zejména pak dostupnost překladače pro jazyk C.
Další částí bude realizovat několik vhodných periferií pro tento processor v jazyce VHDL a vytvořit knihovny pro softwarový přístup k těmto periferiím i dalším, které již existují na dostupném FPGA[1].
Posledním krokem je vytvořit demo-aplikaci demonstrující funkčnost celého projektu a jeho implementace do FPGA.

[1]FPGA - field programable gate array, neboli programovatelné hradlové pole, je integrovaný obvod obsahující různě složité programovatelné bloky propojené konfigurovatelnou maticí spojů. Výhodou oproti zákaznickému obvodu je zejména možnost měnit vnitřní propojení a tak implementovat různé obvody

Kapitola 1

Popis problému, specifikace cíle

Soft-processor je vybírán na základě srozumitelnosti jeho implementace, důležité pro budoucí rozšiřování, a velikosti (množství potřebných zdrojů FPGA, spotřeba). Cílovou platformou bude na katedře dostupné FPGA Spartan 3E od firmy Xilinx, tedy je výhodou, pokud je zvolený processor např. optimalizován pro toto zařízení, či již obsahuje nástroje pro připojení k jeho periferiím (tlačítka, LCD).
V budoucnu je plánováno pracovat s tímto procesorem v jazyce C, je tedy nutné opatřit překladač podporující alespoň základní konstrukce jazyka C. Processor je třeba smysluplně rozšířit, v úvahu přichází například obvod pro realisaci matematických funkcí, které samotný processor nerealizuje. Výsledné zařízení musí také mít dostatečně velkou paměť pro program. Veškerý hardware bude realizován v jazyce VHDL.
Důležitým faktorem je také zohlenění možnosti dalších projektů jiných autorů, kteří snad v budoucnu na tuto práci navazovat. Je tedy přikládán důraz na vypracování postupů pro další rozšíření a práci s picoBlaze na softwarové i hardwarové úrovni.

Kapitola 2

Existující řešení

Tato kapitola obsahuje popis dostupných soft-processorů a mikrokontrollerů. Úplný seznam existujících obvodů není možné pořídit, uvádím tedy několik nejvýznamějších z hlediska požadavků uvedených v předchozí kapitole a dostupné podpory.

2.1 Malé processory

Následuje popis procesorů, které se nedostali do užšího výběru.

2.1.1 ADOP

Autorem ADOPu je Václav Koubek. Původně implementován v rámci bakalářské a diplomové práce jako vylepšení staršího DOP (zmíněn níže), odtud jméno Advanced DOP. Od té doby prošel projekt několika revizemi a úpravami. Jádro používá 16bit registry a ALU a pracuje s doplňkovým kódem. Implementuje 5 stupňovou pipeline připomínající architekturu RISC, v mnoha bodech se však liší:

- pořadí $IF \triangleright ID \triangleright MEM \triangleright EX \triangleright WB$ - implementovány istrukce registr-pamět $\Rightarrow$ MEM fáze musí předcházet EX
- složitější načítání istrukcí, podpora instrukcí s proměnlivou délkou
- implementována řada FLAGů figurujících jako vstupní i výstupní operandy v instrukcích $\Rightarrow$ komplikovanější sledování datových závyslostí
- podpora nezarovnaných přístupů do paměti $\Rightarrow$ složítější MEM stupeň

Další vlastnosti:

- mikroprogramovaný řadič 512 x 64bit
- 65536 KB hlavní paměti vně čipu
- zhruba 20 MIPS při 50MHz

Programová podpora zahrnuje nástroje pro nahrání procesoru do FPGA a následné programování a simulátor.

ADOP je pro naše účely zbytečně velký a nemá (v porovnání s rozšířeným Pico Blaze) příliš široké zázemí.

2.1.2 pAVR

Autorem pAVR je Doru Cuturela. Jedná se o projekt Opencores implementující mikrokontrolér Atmel AVR a dosahující až trojnásobného výkonu původního AVR.

- 8 bit implementující architekturu procesorů AVR
- 6 stupňová pipeline
- 28 – 50 MIPS
- vyvýjený na přípravku Spartan 3
- kompatibilní s nástroji pro AVR
- v současnosti obsahuje několik prozatim neopravenych chyb
- Open Source

Ani pAVR však nemá příliš bohaté uživatelské zázemí.

2.1.3 DOP

Mezi tvůrce patří Alois Pluháček, Miloš Bečvář, Jiří Daněček a další. DOP je malý procesor původem ze začátku 90. let navržen pro studijní účely a snadnou implementaci kompilátorů. V rámci bakalářských a diplomovových prací i jiných projektů je rozšiřován dodnes.

- 16 bit procesor

- 65536B paměť
- 8 a 16 bit registry
- 8 nebo 16 bit data v doplňkovém kódu nebo bez znaménka
- 1B, 2B a 3B instrukce
- horizontálně orientovaný mikroprogramovaný řadíč s délkou instrukce 64 b
- původně implementován na XILINX XC4000

K dispozici je simulátor simDOP pro DOS a NetBeans.

2.2 Velké processory

Následuje stručný popis processorů, které byli v ranných fázích zavrhnuty zejména proto, že jsou pro naše potřeby příliš velké, avšak pro úplnost je třeba je zde zmínit.

2.2.1 Micro Blaze

Tvůrcem je společnost Xilinx. Jedná se o komerční verzi PicoBlaze, který bude představen v příští kapitole.

- 32 bit
- implementuje pipeline
- optimalizovaný pro xilinx FPGA
- velké množství konfigurovatelných parametrů
- podporuje řadu typů připojení periferií
- oficiální C kompilátor od Xilinx

2.2.2 SecretBlaze

Autorem je Lyonel Barthe. Jedná se o open source projekt založený na komerčním Micro Blaze a používající jeho instrkční sadu.

- 32 bit, používá instrukční sadu Micro Blaze
- pipeline
- až 1.35 MIPS
- verze pro digilent s3 a s6
- částečně kompatibilní s nástroji pro MicroBlaze
- není dále rozvíjen

2.2.3 LEON (2, 3, 4)

Autory jsou: ESA, Aeroflex Gaisler a další. LEONy jsou rodina mikrokontrolerů architektury SPARC-V8 v několika implementacích od různých autorů.

- 32 bit, založený na SPARC-V8 RISC
- pipeline
- vysoce konfigurovatelné na úrovni VHDL
- open source

2.2.4 JOP

Autorem je Martin Schoeberl. Java optimised processor – hardwarová implementace JVM navžená pro FPGA (zejména Altera a Xilinx) jako PhD práce a cílená na vestavná zařízení.
Mezi klady patří zejména předvidatelné vykonávání java bytecodu.

Kapitola 3

PicoBlaze

Po dlouhém rozhodování byl jako nejpříhodnější zvolen jednoduchý a velmi populární mikrokontroller PicoBlaze společnosti Xilinx. PicoBlaze je malý, avšak efektivní, 8bit RISC mikrokontroller. Existuje v několika verzích optimalizovaných pro různé typy FPGA Spartan a Virtex. Shrnutí základních parametrů:

- sekvenční zpracování instrukcí
- 8bit ALU
- 44 – 66 MIPS
- ryhlá a převídatelná odezva na přerušení
- každá istrukce trvá 2 takty → snadno vypočitatelná doba trvání programu

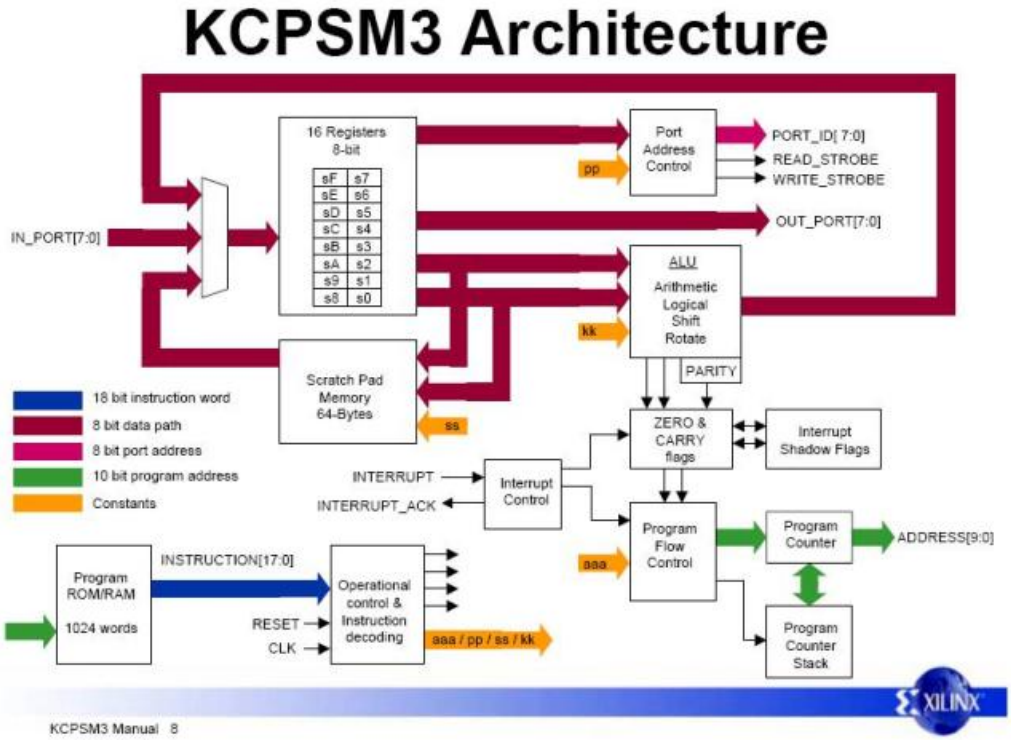
Jelikož budeme pracovat s FPGA Xilinx Spartan 3E, je pro nás nejvýhodnější verze PicoBlazu KCPSM3 ((K)constant coded programmable state machine), která je zaměřená na optimální velikost právě na přípravcích Spartan 3. Základní verze makra KCPSM3 na přípravcích Spartan 3 zabírá:

- 96 sliců - zhruba 5% kapacity nejmenších přípravků serie Spartan 3
- 1 blok RAM 1024x18 pro paměť programu

PicoBlaze je volně dostupný na webu společnosti Xilinx (http://www.xilinx.com/). Navíc KCPSM3 spolu s demonstračním programem a zapojením je přikládán na CD přímo k přípravkům Spartan 3.

3.1 Parametry PicoBlaze

Schéma mikrokontrolleru:



Obrázek 3.1: Schéma KCPSM3

3.1.1 Velikost programu

KCPSM3 disponuje pamětí pro 1024 18bit instrukcí uložených v blok RAM. To se ukazuje jako dostatečné množství pro programy psané rozumně v jazyce symbolických instrukci (pico ASM), avšak budeme-li chtít používat například programovací jazyk C, můžeme zde narazit na určité překážky. V praxi je případná potřeba delšího programu obvykle řešena použitím několika standardních PicoBlazů. Existují však i nestandardní řešení, jako např. přidání dalších bloků RAM, jak bude popsáno v dalších kapitolách.

3.1.2 Registry

KCPSM3 obsahuje 16 universálních 8bit registrů. Tyto registry jsou očíslovány s0-sF, jména jim však lze v picoASM kódu změnit. Všechny registry

jsou rovnocenné, bez speciálních funkcí nebo priorit ani není implementován speciální střadač, neboť tuto funkci může zastávat libovolný registr.

3.1.3 ALU: Aritmeticko-logická jednotka

ALU picoBlazu nabízí řadu operací s jedním, nebo dvěma 8bit operandy. První operandem je vždy obsah registru a do tohoto registru je poté uložen výsledek. Druhým operandem může být buď registr, nebo přímá hodnota. Operace s přímým operandem nijak nezvedají nároky na čas nebo na prostor v paměti programu. Operace sčítání a odčítání mohou využívat příznaku CARRY. ALU však standardně nerealizuje operace pro násobení nebo dělení, což poskytuje námět k možnému rozšíření.

3.1.4 Příznaky

ALU operace standardně nastavují příznaky CARRY a ZERO. Příznak ZERO je nastaven kdykoli je výsledkem ALU operace nula (0x00), CARRY bit je nastaven pokud při operaci ALU došlo k přetečeni a končí v něm bit který z registru vypadne pří instrukcích posunu.

3.1.5 Skoky a podprogramy

PicoBlaze implementuje instrukci podmíněného i nepodmíněného skoku na absolutní adresu - JUMP. Jako rozhodovací kritérium v případě podmíněného skoku mohou být použity příznaky CARRY a ZERO.
Dále je implementována dvojice instrukcí CALL a RETURN pro volání podprogramu a následný návrat. Návratová adresa volání je ukládána na zásobník podporující až 31 vzájemně zanořených volání.

3.1.6 Vstupy a výstupy

KCPSM3 implementuje 256 8bit portů pro vstup a dalších 256 pro výstup. Instrukce INPUT přenese hodnotu na vstupním portu danném druhým operandem do registru zadaného jako první operand. Instrukce OUTPUT analogicky vystaví hodnotu v registru zadaného jako první operand na příslušný výstupní port.

3.1.7 Paměť pro data

Součástí KCPSM3 je tzv. “scratch-pad” paměť o velikostí 64 bytů využívaná zejména pro odkládání nadbytečných hodnot, jsou-li registry plně vytíženy.

Rozsah adres je tedy 0x00 až 0x3f. Pro přístup do paměti se používá dvojice instrukcí STORE a FETCH.

3.2 Příslušenství

PicoBlaze je značně populární a proto nepřekvapí, že existuje množtví oficiálnich i neoficiálních nástrojů pro práci s ním. V této kapitole se zaměřím na ty z nich, které byly použity při tvorbě této práce.

3.2.1 KCPSM3 Assembler

Základní nástroj pro překlad kódu programu pro Windows, je součástí balíčku KCPSM3 od Xilinx. Vstupem je textový soubor formátu psm (PicoASM), výstupem pak mimo jiné i vhdl soubor obsahující kompletní definici bloku RAM paměti, který se dá zakomponovat přímo do projektu s definicí procesoru.
Assembler se spouští z příkazové řádky Windows příkazem:

```
kcpsm3.exe <zdrojovy_soubor>
```

Předpokládá se, že spustitelný soubor kcpsm3.exe i zdrojový soubor jsou umístěny v pracovním adresáři. KCPSM3.exe však není kompatibilní s 64bit systémy, pod kterými je nutno použít například emulátor DoSBOX (http://www.dosbox.com/).

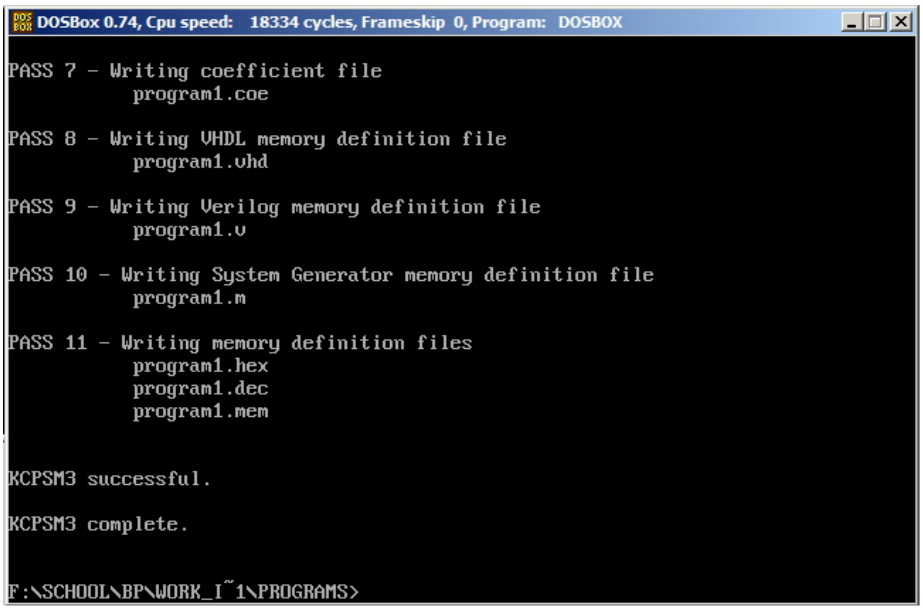
3.2.2 Simulátor pBlazSIM

Nástroj pro Windows, umožnující interpretaci PicoASM kódu a simulaci jader KCPSM 3 a 6.

- podpora breakpointu a základních periferií
- volně dostupný na http://www.mediatronix.org/pages/pBlazSIM

3.2.3 PBCC PicoBlaze C Compiler

Pravděpodobně jediný volně dostupný a fungující překladač z jazyka C. Autorem je Zbyněk Křivka. PBCC je založen na Small Device C Compiler (SDCC) a bez větších obtíží zvládá při nejmenším překlad z C do PicoASM, poté je již možné zbytek překladu obstarat pomocí KCPSM3.exe (viz výše). PBCC implementuje většinu funkcí, které lze od programu pro PicoBlaze požadovat, včetně:



Obrázek 3.2: Assembler v akci

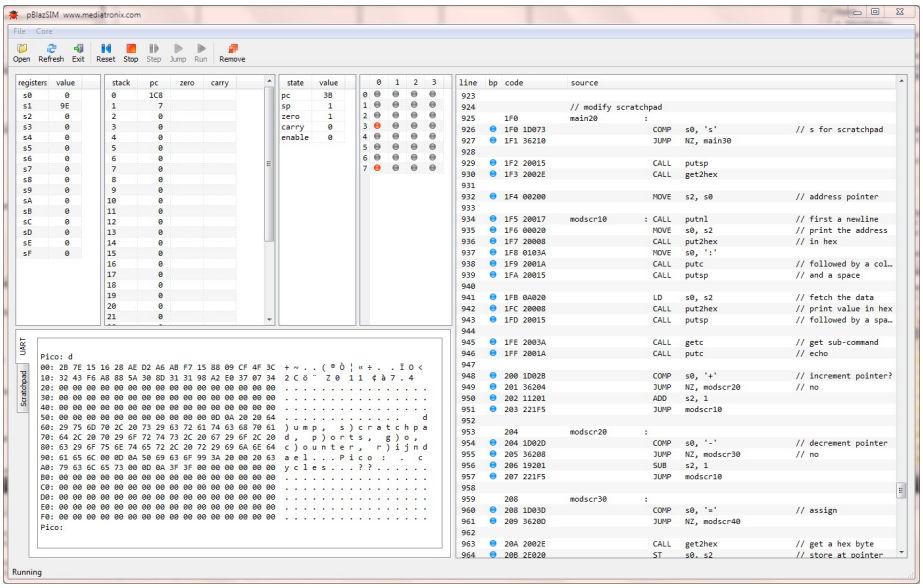
- staticky (ale ne dynamicky) deklarovaných polí
- cyklů
- funkcí
- in-line assembleru

Zdrojové soubory, jakožto i manuál jsou volně ke stažení na <`http://www.fit.vutbr.cz/~meduna/work/doku.php.cs?id=projects%3Avlam%3Apbcc%3Apbcc`>
Doporučuje se použít nejnovější verzi pro cygwin a při instalaci postupovat podle manuálu. Příkaz pro překlad v manuálu se však zdá být nefunkční, lze však použít např.

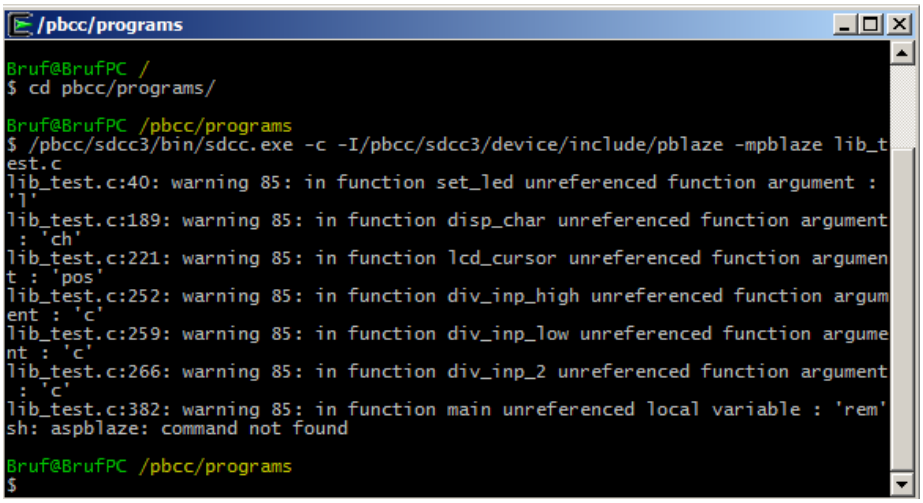
```
/pbcc/sdcc3/bin/sdcc.exe -c -I/pbcc/sdcc3/device/include/pblaze
 -mpblaze <zdroj_soubor>.c
```

Chybové hlášení "sh: aspblaze: command not found" je možno ignorovat, překlad do PicoASM proběhne úspěšně, nenastanou-li jiné chyby.

Poslední verze tohoto překladače byla zveřejněna v roce 2011 a není zdaleka dokonalá. V oblasti in-line assembleru i některých konstrukcí jazyka C mohou nastat problémy, jak bude podrobněji popsáno dále. I tak je však



Obrázek 3.3: Simulátor pBlazSIM



Obrázek 3.4: Překlad pomocí PBCC

více než možné s pomocí tohoto nástroje při opatrném použití dosáhnout dobrých výsledků.

3.2.4 Jiný překladač C

Při rozsáhlejším průzkumu lze nalézt několik dalších projektů, pokoušejících se relizovat překlad z C do PicoASM, avšak většina takových projektů nikdy nebyla dovedena do konce.

Nejslibnější alternativou k předcházejícímu PBCC se zdá být UTIA PicoBlaze C Compiler Toolchain pro Linux s domovskou stránkou <`http://sp.utia.cz/index.php?ids=pblaze-cc`> . Tento překladač se však autorovi přes značné úsilí nepodařilo uvést do funkčního stavu.

3.2.5 Vzorové zapojení a programy

Vzhledem ke značné popularitě PicoBlaze je možné nálézt množství vzorových programů.

Pro tuto práci však byl zdaleka největším přínosem vzorový projekt, který byl součástí balíčku obsahujícího KCPSM3, zejména pak soubory:

- *s3esk_startup.vhd* - ukazuje funkční propojení processoru s blok RAM a také připojení vstupních a výstupních portů
- *s3esk_startup.ucf* - vzorové propojení s periferiemi FPGA
- *control.psm* - zdrojový kód pro demo aplikaci, ukazuje jak je možno používat tlačítka, LED a LCD na FPGA

Tyto soubory byly použity jako základ pro hardware i software vyvýjený v rámci této práce.

3.3 Shrnuti PicoBlaze

Balíček obsahující verzi PicoBlazu KCPSM3, volně ke stažení za webu společnosti Xilinx se stal téměř ideálním základem pro tuto práci. Poskytuje dostatečně srozumitelný základ a množství možností k rozšiřování.

Podařilo se také nalézt dostačující překladač z jazyka C. Jeho použití může sice být místy záludné, avšak možné.

Záporem, který s sebou nese malá velikost PicoBlazu je omezený prostor v paměti programu. Ze záporu se však stala výzva a rozšíření pamě ti programu o další blok RAM se stal následujícím úkolem této práce.

Kapitola 4

Rozšíření paměti pro program

Jak již bylo zmíněno, stávající paměť pro program KCPSM3 poskytující prostor pouze pro 1024 instrukcí je nedostačující a proto bylo jako první tvůrčí úkol stanoveno rozšíření této paměti. Tato kapitola se zabývá procesem vývoje způsobu rozšíření paměti pro program až po finální implementaci.

4.1 Možná řešení

Nepodařilo se nalézt žádné přístupné a fungující řešení problému rozšíření paměti. Avšak Ken Chapman, důchovní tvůrce PicoBlazu na fórech webu Xilinx v roce 2007 nastínil několik možných řešení.

4.1.1 Použití několika samostatných PicoBlazů

Toto řešení by nebylo přímým rozšířením paměti stávajícího processoru, nicméně pravděpodobně by bylo možné docílit podobné funkčnosti.

Řešění vychází z faktu, že picoBlaze je velmi malý a byl zamýšlen pro malé úkoly. V souladu s tím, bychom tedy měli být schopni náš problém rozdělit na několik částí a spustit každou na samostatné instanci PicoBlazu.

Tato metoda však neplní zadání našeho úkolu a není ani dostatečně universální.

4.1.2 Přidání dalších adresních bitů

Originální verze KCPSM3 implementuje adresu pro instrukce o šíři 10 bitů. Ideou tohoto řešení by bylo adresu rozšířit o další bity.

V rámci řešení by bylo nutné upravit samotné jádro procesoru a přidat další adresní vodič, jakožto i rošířit program-counter a další části, které s adresou instrukce pracují. Dále by bylo nutné upravit instrukce pro skok (JUMP) a volání podprogramu (CALL a RETURN) tak, aby s rozšířenou adresou počítaly. Další problémy by pravděpodobně nastaly při samotném překladu programu, neboť existující nástroje počítají s vytvořením programu o maximální délce 1024 instrukcí.

Takové řešení by poskytovalo ideální funkčnost. Z předchozího odstavce je však zřejmé, že realizace dalších adresních vodičů by byla nesmírně komplikovaná a pravděpodobně nad rámec této práce.

4.1.3 Přepínač mezi bloky RAM

Zlatou střední cestou mezi způsoby rozšíření paměti se zdá být implementace malého obvodu, který by byl schopen na základě podnětů z výstupních portů, nebo pouze na základě adresy přepínat mezi několika bloky RAM.

Přepínač by byl realizován malým řadičem, který by ovládal multiplexer přepínající mezi vstupy z různých bloků RAM pro instrukce a svůj výstup napojil na stávající PicoBlaze tak, aby nebylo nutné provádět změny uvnitř samotného processoru. Takové řešení se zdá být proveditelné v relativně kratkém čase.

Tento koncept však s sebou nutně přináší jistá omezení, především:

- Instrukce skoku a volání podprogramu zůstanou nezměněny, počítají však se skokem na absolutní adresu o šíři 10 bitů a budou tedy omezeny na pohyb pouze v rámci jednoho bloku RAM.
- Informaci o tom, ve kterém bloku paměti se program nachází je nutné hlídat programově.
- Problémy s programováním a překladem - překladač z jazyka C a zejména překladač picoASM nepočítají s tvorbou pogramů rozložených na několik block RAM. Je tedy nutné vyvíjet několik samo-

statných programů, které mají možnost skoku z jenoho programu na začátek jiného, avšak bez možnosti např. sdílet proměnné.

I přes tato omezení byl koncept přepínače pro naše potřeby shledán nejvýhodnějším.

4.2 Návrh

Hlavní myšlenkou bylo vložit jednoduchý řídící obvod mezi processor a bloky paměti pro instrukce tak, aby bylo možné mezi bloky přepnout plynule za běhu programu.

4.2.1 Automatické přepínání

Jako první byla zvažována možnost vytvořit takový přepínač, který by k funkci nepotřeboval žádný speciální podnět processoru. Stačí, když přepínač bude naslouchat adresám, které processor do paměti posílá a dorazí-li na adresu poslední, přepínač pomocí multiplexeru plynule přepne na druhý blok. Uvnitř processoru dojde k přetečení program-counteru, avšak jediným důsledkem je přechod zpět na první adresu. Tato adresa se však díky přepínači nachází v druhém bloku RAM a přechodu mezi bloky tak došlo bez jakýchkoli nestandardních postupů uvnitř processoru.

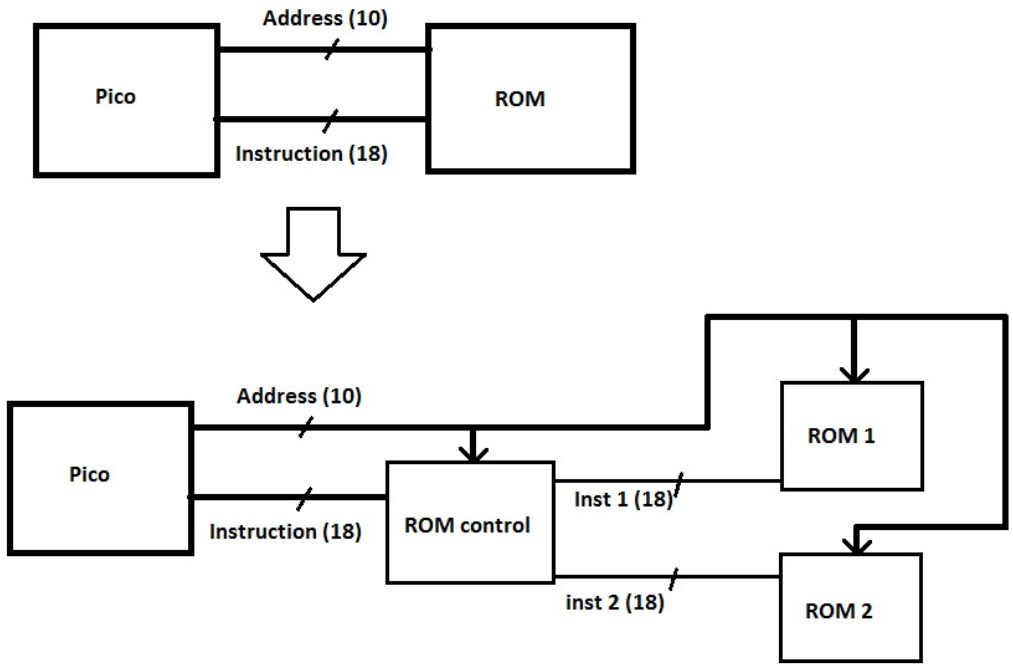
Programově by byla obsluha přechodu mezi bloky velmi jednoduchá. Stačí zajistit, aby

1. poslední adresa paměti zůstala obsazená instrukcí neovlivňující běh programu (picoBlaze přímo neimplementuje instrukci NOP)

2. program začínal na první adrese

Obě tyto podmínky je možné snadno splnit, neboť překladač pro KCPSM3 nabízí možnost explicitně nastavit adresu na kterou chceme instrukci umístit.

Zkušební verze takového obvodu byla realizována, avšak ukázala se nedostatečnou. Hlavním omezením této varianty je nemožnost programově zvolit, na které programové RAM pokračovat. Máme-li rozsáhlejší program, zabírající několik block RAM, stane se z tohoto závažný nedostatek. Možnost proto byla zavrhnuta a nahrazena universálnější.



Obrázek 4.1: Přidání automatického přepínače

4.2.2 Programově ovladatelné přepínání

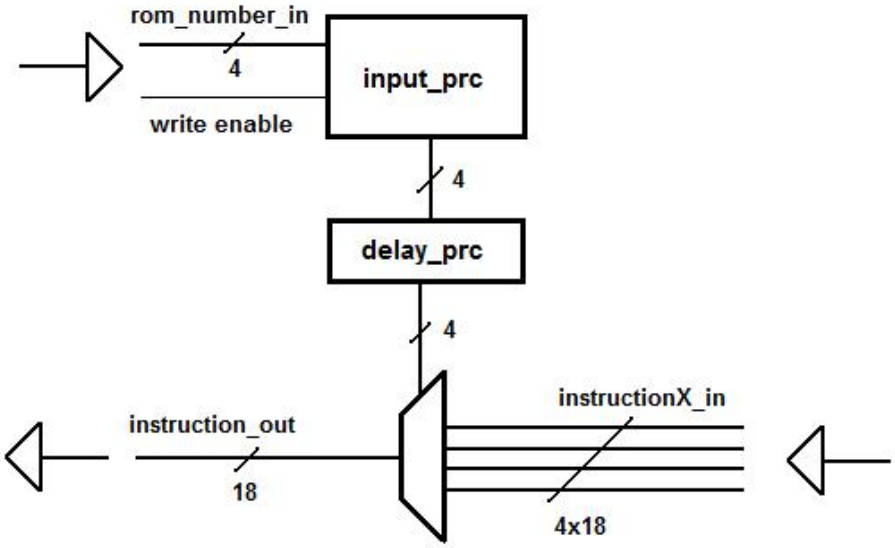
Druhou a lepší možností je vytvoření přepínače, který je možno ovládat programem tak, že přepínač bude připojen na jeden z výstupních portů processoru. Úkolem programu je vystavit na příslušný výstupní port číslo block RAM na kterou má být proveden skok a poté nastavit program counter na začátek programu, ideálně pomocí instrukce JUMP. Úkolem přepínacího obvodu je zapamatovat si čislo RAM na které se má pokračovat a po dostatečné prodlevě pomocí multiplexeru na tuto RAM přepnout.

Ukázalo se, že ideální doba prodlevy je 3 hodinové cykly. PicoBlaze načítá novou instrukcí každé 2 cykly, prodleva 3 cyklů od vystavení čísla nové RAM tedy dává dostatek prostoru k načtení instrukce skoku na adresu 0 a přepne multiplexer včas, aby následující instrukce byla již načtena z nové RAM.

4.3 Realizace

Přepínač byl jako synchronní obvod realizován v jazyce VHDL. Je schopen plynule přepínat mezi až 4 block RAM s možností snadného rozšíření až na 256, což je maximální počet umožňovaný šířkou výstupního portu picoBlaze a pravděpodobně více než budeme schopni vtěsnat na většinu FPGA (Spartan 3E disponuje 20 block RAM).
Vstupy a výstupy:

- 4 x 18 bit instructionX_in - vstupy pro instrukce z 4 různých block RAM
- 1 x 18 bit instruction_out - výstup pro instrukci,která je předána procesoru
- 1 x 4 bit rom_number_in - vstup pro číslo RAM na kterou se bude přepínat
- 1 bit write_enable - vstup write enable, indikující platnost dat rom_number_in, aktivní v logické 1
- vstupy pro synchronní reset a hodinový signál



Obrázek 4.2: Schéma programově ovladatelného přepínače

Funkce přepínače je realizována pomocí 3 procesů popsaných v následujících podkapitolách.

4.3.1 Input_prc

Proces zaopařující načítání vstupu s číslem následující block RAM. Jedná se v podstatě o 4 bit D klopný obvod s resetem a signálem pro povolení čtení. Je-li write_enable aktivní, jsou data z rom_number_in načtena.

4.3.2 Delay_prc

Proces realizující zpoždění mezi daty načítanými z rom_number_in a výstupním multiplexerem. Jak již bylo zmíněno dříve, je nutné 3 hodinové cykly posečkat mezi obdržením čísla následující block RAM a samotným přepnutím. Proces realizuje 2 za sebou spojené D klopné obvody vytvářející zpoždění 2 hodinových cyklů, zbývající 1 cyklus je již způsoben v Input_prc.

4.3.3 Instruction Mux

Proces realizující výstupní multiplexer, který propouští instrukce z block RAM podle aktuální hodnoty v Delay_prc.

4.4 Testování

Testování obvodu přepínače proběhlo jak na úrovni VHDL pomocí programu ModelSIM, tak pomocí testovacích programů implementovaných přímo do přípravku Spartan 3E.

Mezi úvodní problémy patřil zejména fakt, že PicoBlaze žádá o instrukci pouze každé 2 hodinové cykly. Během těchto dvou cyklů tak docházelo k přepnutí přepínače dvakrát a tedy setrvání ve čtení z původního bloku paměti. Problém byl vyřešen přidáním procesu Delay_prc, který realizuje potřebnou prodlevu. Po této úpravě je již přepínání hladké a bezproblémové.

4.5 Shrnutí

V relativně krátkém čase se podařilo realizovat dostačující řešení, které umožňuje rozšířit KCPSM3 o pořebné množství block RAM pro program.

Výhodami řešení jsou především snadná programová obsluha a neporušení originálního procesoru i přez to, že může být objem programu i mno-

honásobně rozšířen.

Nevýhodou je nutnost vytvářet program pro každý blok paměti zvlášť a omezení instrukcí skoku a volání podprogramů pouze na rámec aktuálního bloku paměti. Odstranění těchto nevýhod by však vyžadovalo rozsáhlé a komplikované úpravy jak samotného processoru, tak veškerého vývojového softwaru.

Mezi možnosti dalšího rezšíření patří např. realizace paměti pro odkládání dat mezi přepínaními programových RAM, neboť současné vývojové nástroje nemohou realizovat přenášení proměnných jazyka C z části programu na jedné RAM do části programu na jiné RAM.

Kapitola 5

Dělička

Na podnět vedoucího práce bylo rozhodnuto, že dalším úkolem v rámci rozšíření KCPSM3 bude realizace děličky.

ALU PicoBlazu neimplementuje operace násobení ani dělení a proto může být externí obvod pro dělení cenným rozšířením, například chcemeli na displeji zobrazovat čísla v jiné než hexadecimální soustavě. Dělení je navíc časově poměrně náročné a jeho provádení vně nebude processor blokovat. V případě, že budeme vyžadovat velké množství dělících operací, navíc můžeme připojit děliček několik a provádět operace alespoň částečně paralelně.

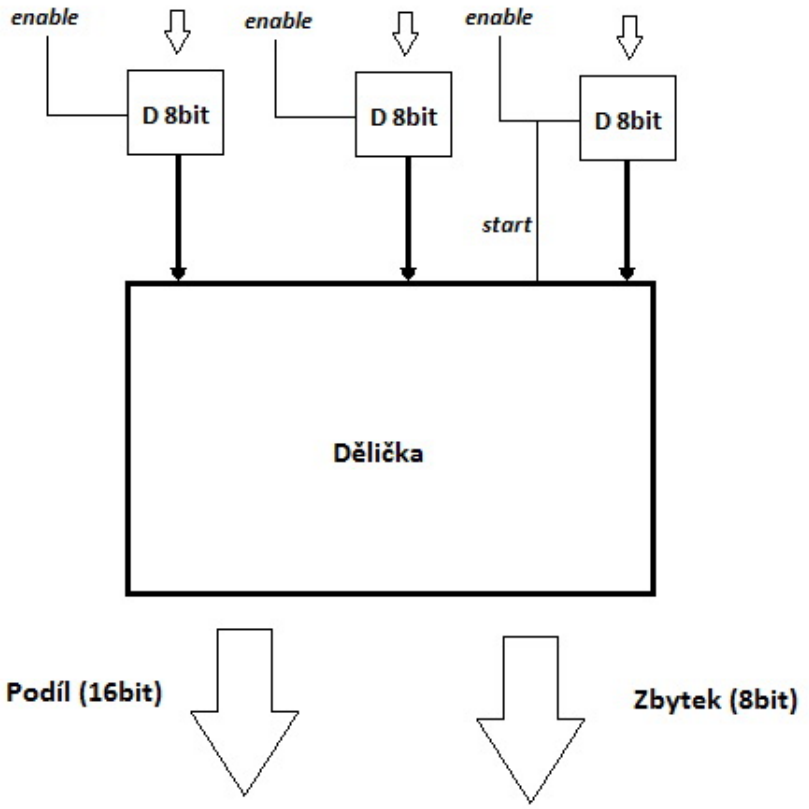
5.1 Návrh

Dělička zpracovává 2 operandy - 16 bitový dělenec a 8 bitový dělitel. Výstupem je pak 16 bitový rozdíl a 8 bitový zbytek. Předpokládá se, že všecha čísla jsou bez znaménka.

Na začátku dělení je dělenec nahrán do spodních 16 bitů registru A a dělitel do registru B. Na konci procesu je podíl uložen ve spodních 16 bitech registru A a zbytek v horních 8 bitech registru A.

Dělení trvá 18 hodinových cyklů, tedy ekvivalent 9 PicoBlaze instrukcí. Samotný rozdíl je dostupný již po 17 cyklech, rozdíl je za ním jeden takt opožděn. Během tohoto taktu probíhá návrat přes nulu, byl-li na konci dělení zbytek záporný.

Signál alpha je aktivní ve všech taktech dělení kromě posledního, kdy probíhá úprava zbytku. Signál beta je aktivní pouze v posledním taktu a signál gamma pouze v prvním.



5.2 Realizace

Dělička byla realizována v jazyce VHDL pomocí 6 samostatných entit, rozdělujících obvod na logické části. Tyto části byly poté propojeny v sedmé entitě a vytvořili kompletní funkční obvod.

Vtupy a výstupy:

- 3 x 8 bit vstup inp_X - vstupy pro nahrání operandů, inp_1 pro vyšší byte dělence, inp_2 pro nižší a inp_3 pro dělitele
- 3 x 1 bit vstup inp_E_X - enable pro každý ze vstupů pro operandy, aktivní v logické 1
- 1 bit výstup ready - logická 1 indikuje, že výpočet je dokončen

- 16 bit výstup result - výsledek dělení, data jsou platná, je-li ready aktivní
- 8 bit výstup residue - zbytek po dělení, data jsou platná, je-li ready aktivní
- hodinový vstup

5.2.1 Entita A flip flop

Realizuje registr A ze schématu. Oproti schématu však také zahrnuje signál reset, pro úvodní nahrání dělence do registru a signál done, který zamezí dalším změnám v obsahu registru po uplynutí potřebného počtu taktů - tedy když již registr obsahuje podíl.

5.2.2 Entita B flip flop

Realizuje kombinaci registru B a bloku XOR ve schématu. Opět je implementován signál reset pro úvodní nahrání dělence. Po úvodním nahrání blok na svůj výstup každý takt vystaví buď hodnotu dělitele, nebo jeho bitovou inverzi podle hodnoty signálu minus.

5.2.3 Entity Shift A a B

Obě realizují bitový posun tak, jak je znázorněno ve schématu, tedy na základě signálu alpha a beta vysílaných řadičem děličky.

5.2.4 Adder

Kombinační obvod realizující 9 bitovou sčítačku se vstupem pro přenos do nejnižšího řádu (ve schématu je do něj zapojen signál minus) a výstupem pro přenos z nejvyššího řádu (ve schématu signál q).

5.2.5 Controller

Realizuje řídící jednotku děličky, která ve schématu patrná není a zodpovídá za správné nastavování řídících signálu alpha, beta, gamma, reset a done.

Základem je čítač, který se každý hodinový cyklus inkrementuje a obsahuje číslo taktu $0 - 18$. Podle hodnoty v čítači pak kombinační logika řadiče nastavuje řídící signály. Stavy podle hodnoty čítače:

- 0 - klidový stav, v tomto stavu se dělička nachází po dokončení výpočtu, z výstupu děličky je možno přečíst podíl a zbytek
- 1 - úvodní stav kdy je aktivní signál reset a do registrů jsou nahrány operandy
- $2 - 17$ - probíhá výpočet
- 18 - podíl je již připraven, pokud je zbytek záporný, probíhá návrat přes nulu

5.2.6 Entita Divide 16 8 top

Top-level Entita propojující jednotlivé části děličky dohromady. Jsou zde také implementovány tři 8 bitové registry pro uložení vstupů, každý opatřený signálem enable. Poté, co je do třetího registru nahrána hodnota (dělitel), je do řadiče děličky automaticky vyslán signál pro zahájení výpočtu.

5.3 Testování a ladění

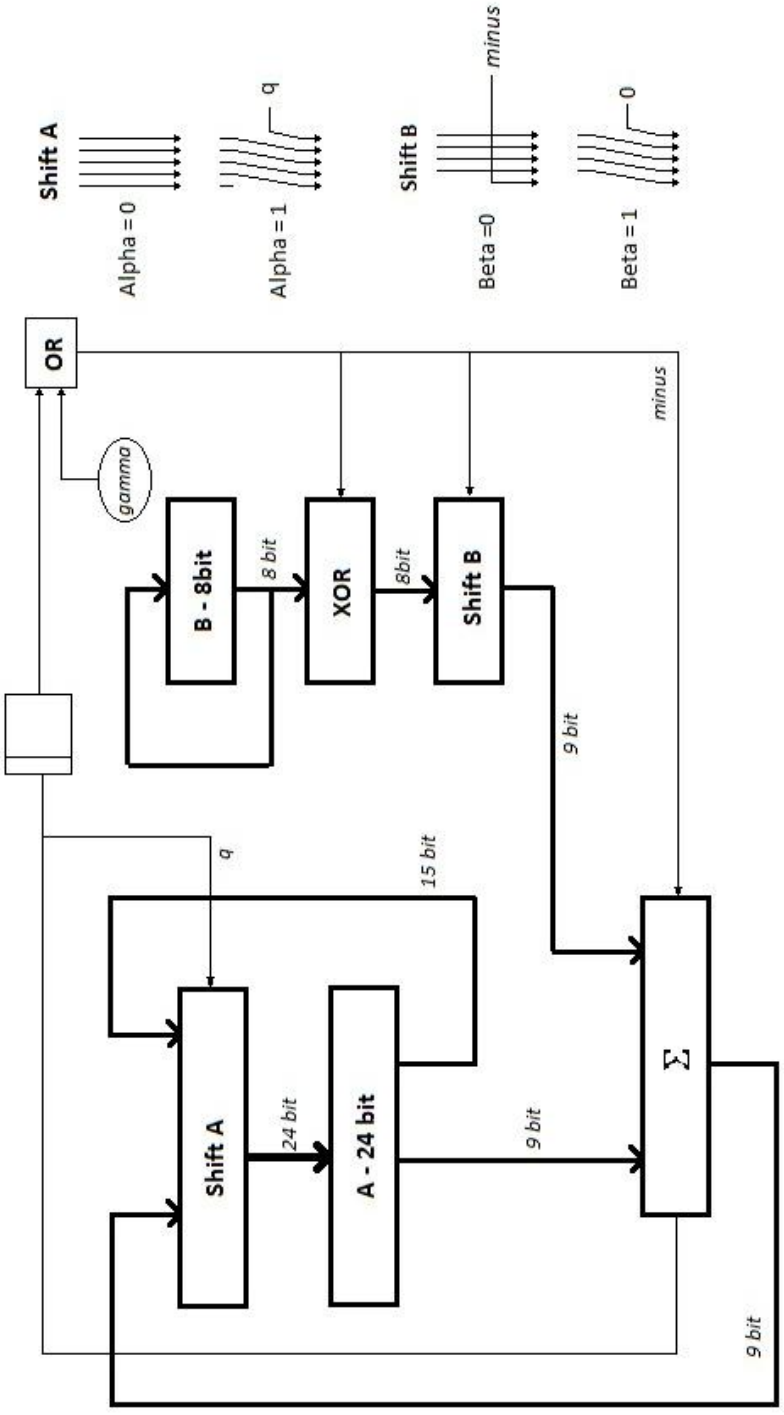
Testování děličky zabralo značné množství času a bylo prováděno jako na úrovni VHDL modelu, tak na samotném přípravku Spartan 3E.

Po dlouhou dobu se nedařilo implementovat správné nastavování řídících signálů a zejména závěrečná úprava hodnoty zbytku se ukázala být oříškem.

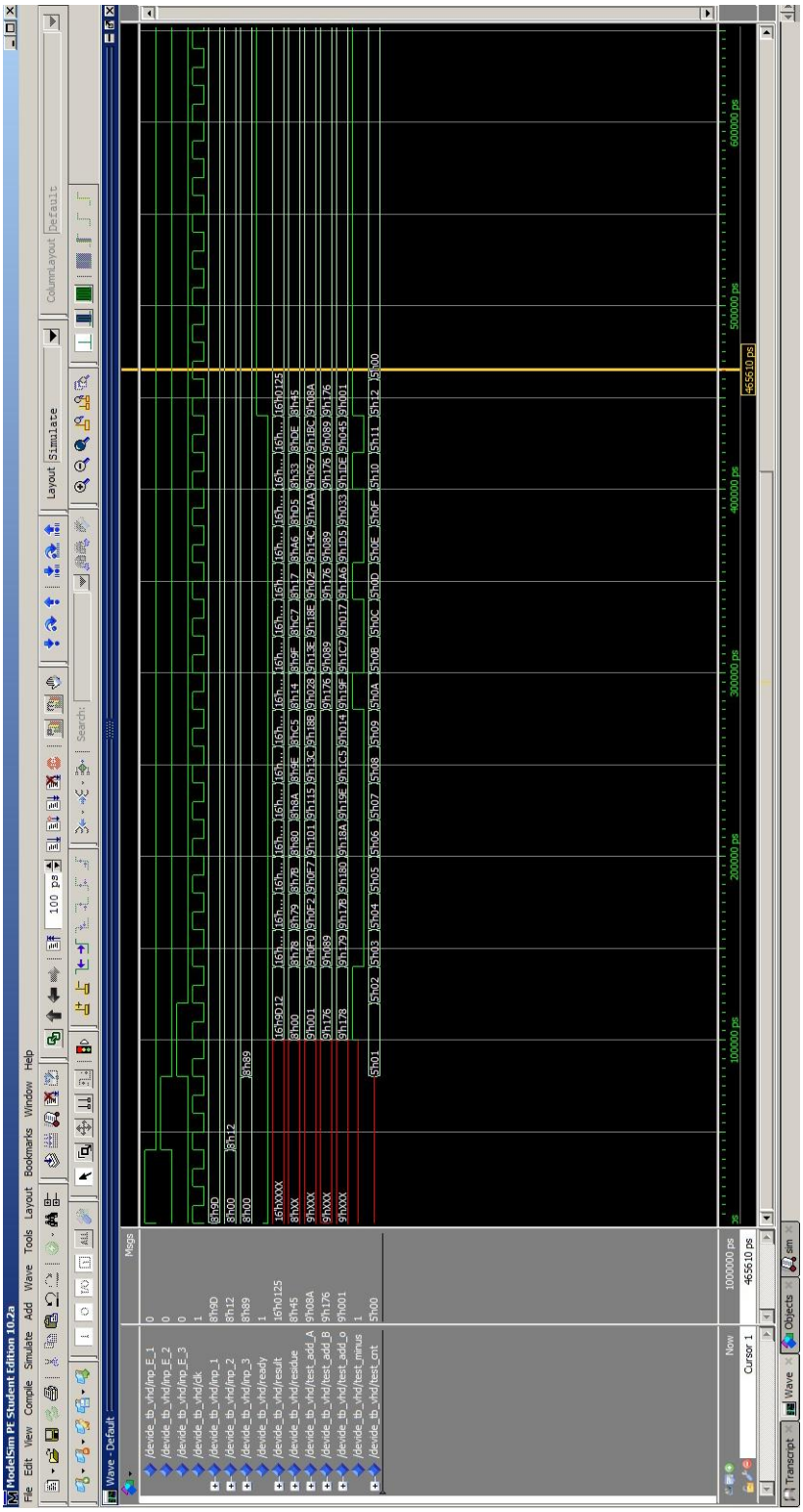
5.4 Shrnutí

Implementace děličky se ukázala být obtížnějším úkolem, než se zprvu zdálo. Po vynaložení velkého množství času a úsilí byl však i tento obvod uveden do funkčního a úspěšně připojen k KCPSM3.

Děličku by bylo v budoucnosti možno rozšířit např. pro použití s 16bit dělitelem a zbytkem.



Obrázek 5.1: Schéma děličky



Obrázek 5.2: testování děličky

Kapitola 6

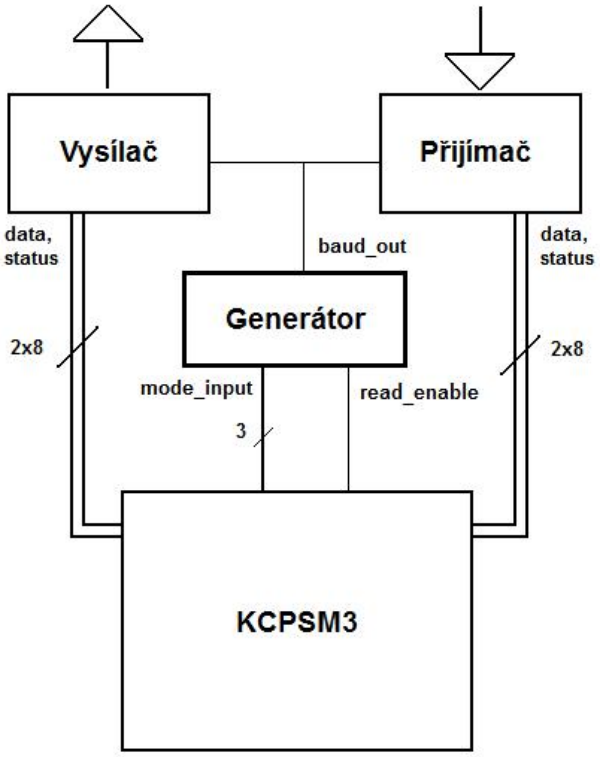
Sériová linka

Jako další rozšíření byla zvolena implementace sériové linky, umožnující rozšířenému picoBlaze komunikovat s rozličnými jinými zařízeními. Tento úkol je značně usnadněn tím, že součástí balíčku KCPSM3, který společnost Xilinx dává k dispozici, jsou již implementace výkonných částí vysílače i přijímače UART, které jsou do značné míry kompatibilní s picoBlaze. Tyto komponenty však sami o sobě nejsou ještě příliš užitečné. Vyžadují správné připojení k picoBlaze a také generátor pulzů, určující baudrate, jakožto i programovou podporu zpřístupňující ovládání všech komponent pomocí jazyka C.

6.1 Návrh

S balíčku KCPSM3 máme již k dispozici komponenty pro realizaci vysílače a příjímače pro seriový přenos 8bit + 1 stop bit bez parity. Baudrate přenosu je však určen vstupním signálem, jenž musí být generován mimo tyto komponenty. Je vyžadován signál, který je aktivní vždy 1 hodinový cyklus a frekvence těchto aktivních pulzů musí být přibližně 16x vyšší než požadovaný baudrate. Vysílací komponenty pracují synchronně a tento signál v nich zastává funkci enable. Úkolem této části tedy je navrhnout programově ovladatelný generátor takového signálu.

Generátor je připojen na jeden z výstupních portů picoBlaze, pomocí něhož je možné zvolit jeden ze 4 předdefinovaných baudratů. Obvod je také snadné upravit tak aby podporoval další volby (teoreticky až 256). Generovaný signál je možné použít jak pro přijímač, tak pro vysílač.



Obrázek 6.1: seriová linka - celkové schema

6.2 Realizace

Hlavní částí je čítač, jenž se každý hodinový cyklus inkrementuje a výsledný stav je porovnáván s konstantou dannou konkrétním nastavením. Je-li konstanty dosaženo, čítač je vynulován a je generován puls trvající jeden hodinový takt. Konstanty jsou předpočítány pro několik základních baudratů za předpokladu, že obvod je připojen na standardní oscilátor Spartanu 3E o frekvenci 50 MHz. Při jiné hodinové frekvenci je nutno tyto konstanty odpovídajícím způsobem změnit, tak aby byly pulzy generovány s frekvencí přibližně 16x vyšší než je požadovaný baudrate.

Vstupy a výstupy:

- 3 bit vstup mode_input - vstup pro volbu baudrate
- 1 bit vstup read_enable - indikuje, že se mají načíst data z mode_input, aktivní v logické 1
- 1 bit výstup baud_out - generovaný signál pro připojení k ostatním komponentám UART

- vstupy pro hodinový signál a synchronní reset

Předdefinované hodnoty baudratu:

mode	limit čítače	baudrate
000	325	9 600 b/s
001	165	19 200 b/s
010	26	115 200 b/s
jiný	0	3 125 000 b/s

Funkce generátoru je realizována ve VHDL pomocí 3 procesů:

6.2.1 input_D

Proces zaopatřující vstupní registr pro mode_input. Nová data jsou načítána, je-li aktivní read_enable. Jedná se v podstatě o 3 bitový D klopný obvod s enable a resetem.

6.2.2 limit_switch

Proces nastavující správnou hodnotu limitu pro nulování čítače, na základě hodnoty ve vstupním registru. Jedná se v podtatě o jednoduchý dekóder.

6.2.3 counter

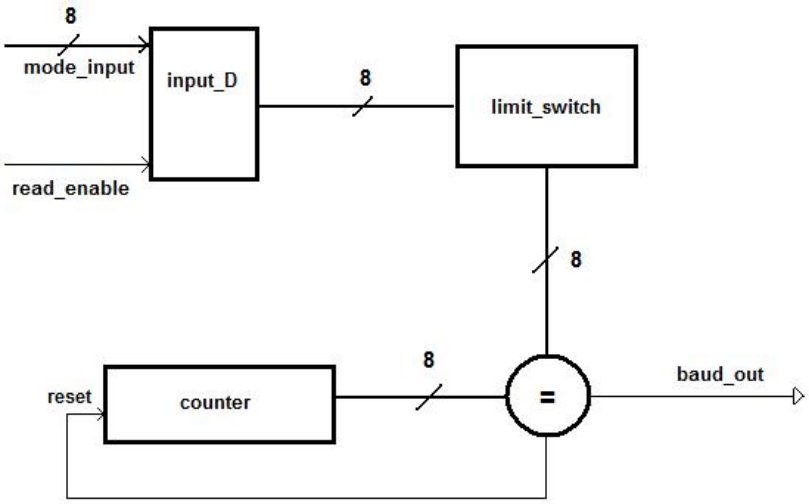
Proces realizující 8 bitový čítač, který se inkrementuje s každým hodinovým cyklem. Dosáhne-li hodnota čítače limitu zvoleného v limit_switch je čítač vynulován pomocí baud_out je vyslán puls.

6.3 Testování

Testování bylo prováděno zejména v praxi na přípravku Spartan 3E a se všemi podporovanými baudraty a to jak mezi dvěma picoBlazy, tak proti PC. Obvod funguje dobře se všemi podporovanými baudraty je-li připojen na standardní hodinový kmitočet 50 MHz.

6.4 Shrnutí

Po pochopení činnosti a nároků dodaných komponent seriové linky se podařilo poměrně hladce implementovat generátor pulzů, umožňující komunikaci



Obrázek 6.2: schema - generátor

na široké škále vysílacích rychlostí. Celou soustavu se podařilo úspěšně zakomponovat do celkového projektu.
Generátor lze dle potřeby snadno rozšířit o další vysílací rychlosti.

Kapitola 7

Knihovna pro práci s KCPSM3

Jedním z hlavních úkolů této práce bylo zpřístupnit PicoBlaze pro programovací jazyk C. Za tímto účelem byla vytvořena knihovna *picolib.c* obsahující definice základních funkcí pro ovládání LCD obrazovky, LED kontrolek, tlačítek, přepínačů i otočného knoflíku přítomných na desce přípravku Spartan 3E a také funkce pro ovládání nově přidaných obvodů dělička, přepínač RAM a sériová linka.

Knihovna byla vytvořena pro zpracování překladačem PBCC a její funkčnost s použitím jiného překladače není zaručena.

7.1 Implementace

Hlavním problémem při implementaci se ukázalo použití C proměnných v in-line picoASM kódu. PBCC by sice měl být schopen použít jednobytovou proměnnou místo jména registru jako operand instrukce, ale tato funkce se přes značné úsilí nepovedla uvést do provozu. Knihovní funkce, které to nutně vyžadují místo toho využívají skutečnosti, že první parametr volané funkce je vždy předáván v registru sB a další parametry po řadě v registrech sC, sD a sE. Vytvoříme-li funkci s více než 4 byty parametrů, jsou nadbytečné paramentry předávány skrze scratchpad pamět. Stejným způsobem je vždy návratová hodnota funkce předávána registrem sB, popřípadě sB a sC, jedná-li se o 16bit datový typ. Opatrným využitím této skutečnosti a omezením velikosti funkcí vyžadujících přímou interakci s registry tak předejdeme kolizím a můžeme úspěšně vytvářet funkční programy.

7.2 Použití

Knihovnu je možné jako celek zakomponovat do programu podle zvyklostí jazyka C direktivou *#include*, avšak takový postup nelze doporučit. Hlavní nevýhodou je velikost výsledného programu, neboť překladač pbcc přeloží do jazyka symbolických instrukcí veškeré funkce v knihovně definované a zabere jimy téměř polovinu kapacity block RAM. Efektivnější způsob spočívá ve vybrání pouze těch funkcí z knihovny, které bude část programu na danné RAM používat a nakopírování jejich definic spolu s definicemi konstant do zdrojového C kódu pro tuto RAM.
Některé funkce knihovny při překladu generují varování s označením *warning 85*:

```
in function XXX unreferenced argument : 'Y'
```

Varování je zapříčiněno přístupem k proměnným pomocí in-line assembleru namísto jazyka C a je považováno za normální chování.

7.3 Funkce pro práci se Spartan 3E

V této sekci je uveden popis funkcí používaných pro interakci s displejem, LED, přepínači, tlačítky a otočným knoflíkem na desce Spartan 3E. Předpokládá se, že periferie jsou přípojeny na vstupní a výstupní porty tak, jak je uvedeno v tabulce na konci kapitoly, jiné zapojení vyžaduje změnu deklarací konstant pro přístup k portům.

Inspirací pro tyto funkce byl vzorový program v PicoASM kódu přiložený k balíčku ubsahující KCSM3.

7.3.1 led btn init

```
void led_btn_init ()
```

Funkce obsahující deklaraci konstant pro přístup k portům používaným LED, přepínači, tlačítky a otočným knoflíkem. Vzhledem k tomu, jak PBCC provádí překlad, není nutné aby byla funkce za běhu programu volána, dostačuje přítomnost její definice ve zdrojovém kódu. Funkce nemá parametry ani návratovou hodnotu.

7.3.2 set led

```
void set_led (volatile char l)
```

Funkce pro rozsvěcování a zhasínání LED kontrolek na Spartan 3E. Parametrem je 8 bitová hodnota, kde každý bit odpovídá stavu jedné LED. Je-li bit nastaven kontrolka svítí. Nejnižší bit odpovídá LED na přípravku nejvíce vpravo, nejvyšší pak nejvíce vlevo. Při přípravě parametru pro tuto funkci je možné využít předdefinovaných konstant pro jednotlivé kontrolky nazvaných LED0 - LED7.

7.3.3 read buttons

```
char read_buttons ()
```

Funkce pro čtení stavu tlačítek a přepínačů. Návratovou hodnotou funkce je 8 bitová hodnota, jejíž každý bit reprezentuje stav jednoho tlačítka nebo přepínače na desce KCPSM3, je-li bit nastaven, znamená to že tlačítko je stisknuto, nebo že přepínač je v pozici zapnuto. Pro odfiltrování informace o jednom konkrétním tlačítku nebo přepínači je možno použít předdefinovaných hodnot $switch0 - switch3$ pro přepínače a BTN_east, BTN_west, BTN_north a BTN_south pro tlačítka.

7.3.4 read rotary

```
char read_rotary ()
```

Funkce pro čtení stavu otočného knoflíku. Návratovou hodnotou je 8 bitová hodnota avšak pouze 3 z těchto bitů nesou informaci.

- $bit0$ - representuje směr posledního otočení tlačítkem. Logická 1 indikuje směr doleva, logická 0 směr doprava.
- $bit1$ - informuje zda je knoflík stlačen. Logická 1 indikuje stlačení.
- $bit2$ - informuje o tom, zda bylo od posledního volání knoflíkem otočeno. Logická 1 indikuje, že data v bit 0 jsou platná. Každé volání této funkce bit 2 vynuluje.

Pro snadnější izolaci jednotlivých bitů lze také použít definované konstanty ROT_left, ROT_event a ROT_press.

7.3.5 lcd init

```
void lcd_init ()
```

Funkce pro inicializaci LCD displeje. V programu je nutné tuto funkci zavolat dříve než začneme s displejem pracovat. Funkce nemá parametry ani návratovou hodnotu.

7.3.6 lcd clear

```
void lcd_clear ()
```

Funkce pro vynulování stavu displeje. Veškerý obsah displeje je vymazán a kurzor je nastaven na první pozici. Funkce nemá parametry ani návratovou hodnotu.
Funkce může být programem volána až poté, co byla alespoň jednou zavolána funkce `lcd_init`.

7.3.7 disp char

```
void disp_char (volatile char c)
```

Funkce pro zobrazení jednoho znaku na displej. Parametrem je 8 bitová hodnota, jejíž protějšek v ASCII je zobrazen na pozici kurzoru. Kurzor je poté posunut o jednu pozici dále na stejném řádku. Displej Spartan 3E je schopen zobrazit malá i velká písmena anglické abecedy, číslovky a většinu běžně používaných znaků, nikoliv však celý rozsah ASCII.
Funkce může být programem volána až poté, co byla alespoň jednou zavolána funkce `lcd_init`.

7.3.8 disp string

```
void disp_string (char * string, volatile char len)
```

Funkce pro zobrazení řetězce znaků na displej. Prvním parametrem je ukazatel na řetězec, druhým je délka řetězce, který chceme zobrazit. Řetězec je zobrazován od aktuální pozice kurzoru a nepřechází na další řádek, po dosažení konce řádku zůstává zbytek znaků nevyužit. Je tedy zbytečné aby byl funkci předáván druhý parametr s hodnotou větší než 16. Funkce nemá návratovou hodnotu.
Funkce může být programem volána až poté, co byla alespoň jednou zavolána funkce `lcd_init`.

Tato funkce požívá funkci `disp_char`, je proto nutné aby definice `disp_char` byla v kódu přítomna.

7.3.9 lcd cursor

```
void lcd_cursor (volatile char pos)
```

Funkce nastaví kurzor na jednu z 32 možných pozic. Kurzorem se rozumí lokace na displeji, kam ostatní funkce této knihovny zapisují znaky. Volání této funkce je jediný způsob přechodu na jiný řádek. Parametrem je 8 bitová hodnota, která by měla být jednou z validních pozic na displeji, jak je zobrazeno v následující tabulce.

Funkce může být programem volána až poté, co byla alespoň jednou zavolána funkce `lcd_init`.

Pozice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Řádek 1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Řádek 2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

7.4 Funkce pro práci s děličkou

V této sekci je uveden popis funkcí používaných pro interakci s děličkou, navrženou a realizovanou v rámci tohoto projektu. Funkce obtarávají nezbytné úkony nahráváním operandů počínaje a vyzvedáváním výsledků konče. Předpokládá se, že dělička je připojena na vstupní a výstupní porty tak, jak je uvedeno v tabulce na konci kapitoly, jiné zapojení vyžaduje změnu deklarací konstant pro přístup k portům.

7.4.1 div init

```
void div_init ()
```

Funkce obsahující deklaraci konstant pro přístup k portům používaným děličkou. Vzhledem k tomu, jak PBCC provádí překlad, není nutné aby byla funkce za běhu programu volána, dostačuje její přítomnost ve zdrojovém kódu. Funkce nemá parametry ani návratovou hodnotu.

7.4.2 div inp

```
void div_inp_high (volatile char c)
void div_inp_low (volatile char c)
void div_inp_2 (volatile char c)
```

Trojice funkcí pro nahrání operandů do děličky. Parametry jsou 8 bitové hodnoty operandů.

Funkce `div_inp_high` a `div_inp_low` slouží k nahrání vyššího, resp. nižšího, bytu dělence. Tyto funkce mohou být volány v libovolném pořadí.

`div_inp_2` slouží k nahrání dělitele. Zavoláním této funkce zároveň děličku zpustíme a je tedy třeba funkci volat až po nahrání dělence.

7.4.3 div out

```
unsigned char div_out_high ()
unsigned char div_out_low ()
unsigned char div_out_remain ()
```

Trojice funkcí pro vyzvedávání výsledků z děličky. Návratovou hodnotou je 1 byte podílu nebo zbytku.
Funkce je možné volat kdykoliv po dokončení výpočtu (9 instrukcí po volání funkce `div_inp_2`) a v libovolném pořadí.

7.4.4 divider input

```
void divider_input (unsigned int a, unsigned char b)
```

Funkce obstarávající celý proces nahrávání operandů do děličky. Prvním parametrem je 16 bitový dělenec, druhým 8 bitový dělitel. Oba operandy by pro správnou funkčnost měly být kladná čísla bez znaménka.

Funkce využívá trojici funkcí `div_inp` a ve většině případů je jejich pohodlnější náhradou. Definice těchto funkcí však musí být součástí kódu.

7.4.5 divide

```
unsigned int divide (unsigned int a, unsigned char b)
```

Funkce obstarávající celý proces dělení včetně nutné prodlevy při čekání na výsledek. Prvním parametrem je 16 bitový dělenec, druhým parametrem je 8 bitový dělitel. Oba operandy by pro správnou funkčnost měly být kladná čísla bez znaménka. Návratovou hodnotou je 16 bitový podíl.

Funkce postupně nahraje do děličky oba operandy, poté počká na výsledek a vyzvedne a vrátí podíl, zbytek, je-li požadován, je třeba vyzvednout zvlášť pomocí funkce `div_out_remain`.

Z hlediska efektivity však tato funkce není ideální, neboť čekání na výsledek by mohlo být při postupném volání předchozích funkcí vyplněno smysluplnou činností.

7.5 Funkce pro obsluhu přepínání bloků paměti

Zde je uveden popis funkce pro práci s přepínačem pamětí pro program. Vzhledem k implementaci přepínače vystačíme s jedinou jednoduchou funkcí. Funkce předpokládá se, že přepínač je připojen na výstupní porty tak, jak je uvedeno v tabulce na konci kapitoly. Jiné připojení je možné, avšak vyžaduje změnu hodnoty konstanty ve funkci `romSwitch`

7.5.1 romSwitch

```
void romSwitch (char c)
```

Funkce realizující skok na začátek programu nacházející se na block RAM danné parametrem. 8 bitový parametr *c* by měl obsahovat číslo požadované RAM. Současná verze dovoluje hodnoty 0x00, 0x01, 0x02, 0x03. Jakákoli jiná hodnota provede skok do RAM 0.

7.6 Funkce pro práci se sériovou linkou

V této sekci jsou popsány funkce pro práci s přijímačem a vysílačem sériové linky a také pro nastavení baudrate pomocí obvodu pro generování pulzů. Předpokládá se, že všechny komponenty sériové linky jsou připojeny na vstupní a výstupní porty picoBlazu tak, jak je uvedeno v tabulkách na konci této kapitoly. Jiné připojení je možné, avšak vyžaduje změnu hodnot konstant ve funkci `uart_init`.

7.6.1 uart init

```
void uart_init ()
```

Funkce obsahující deklarace konstant pro přístup k portům používaným sériovou linkou. Pro správné chování není nutné funkci během programu volat, postačuje, pokud je její definice přítomna ve zdrojovém kódu.

7.6.2 uart putc

```
void uart_putc (char c)
```

Funkce pro odeslání znaku pomocí sériové linky. Parametrem je 8 bitová hodnota která má být odeslána.
Vyžaduje přítomnost definice funkce `uart_init` ve zdrojovém kódu.

7.6.3 uart rx status

```
char uart_rx_status ()
```

Funkce pro přečtení stavového registru přijímače seriové linky. 8 bitová návratová hodnota nese informace o stavu výstupní fronty přijímače.

- *bit*0 - je-li nastaven, idnikuje, že ve výstupní frontě čeká alespoň 1 znak, a data tedy je možno z přijímače číst
- *bit*1 - je-li nastaven, indikuje, že výstupní fronta je plná a přijímač nemůže přijímat další znaky
- *bit*2 - je-li nastaven, indikuje, že ve výstupí frontě čeká alespoň 8 znaků (polovina kapacity) a tedy, že brzy dojde k jejímu zaplňení
- $bity3 - 7$ - nevyužity

Vyžaduje přítomnost definice funkce `uart_init` ve zdrojovém kódu.

7.6.4 uart getc

```
char uart_getc ()
```

Funkce pro vyčtení znaku z výstupní fronty přijímače sériové linky. 8 bitová návratová hodnota je znak který byl právě přečten. Pro spávné fungování je třeba se pomocí funkce `uart rx status` nejprve ujistit, že ve frontě

alespoň jeden znak čeká a že je tedy co číst. V opačném případě je navrácená hodnota neplatná.
Vyžaduje přítomnost definice funkce `uart_init` ve zdrojovém kódu.

7.6.5 uart baud mode

```
void uart_baud_mode (char c)
```

Funkce pro volbu rychlosti vysílání po serivé lince. Po zavolání přepne jak vysílač, tak přijímač na zvolenou rychlost. Parametrem je číslo jedné z předdefinovaných vysílacích rychlostí.

parametr c	baudrate
0x00	9 600 b/s
0x01	19 200 b/s
0x02	115 200 b/s
jiný	3 125 000 b/s

Tuto funkci je třeba zavolat před započetím přenosů.
Vyžaduje přítomnost definice funkce `uart_init` ve zdrojovém kódu.

7.7 Definice vstupních a výstupních portů

V této sekci je uveden seznam vstupních a výstupních portů, na které musí být periferní obvody připojeny, aby funkce uvedené v této kapitole správně pracovaly a který je použit v projektu, jenž je součástí této práce.

Jiné zapojení je možné, avšak vyžaduje zásah do knihovny. Konkrétně je třeba změnit deklarace konstant portů v *init* funkcích.

7.7.1 Vstupní porty

Zde je uveden seznam obvodů a příslušných vstupních portů, s nimiž funkce této kapitoly počítají.

Zařízení	Číslo portu
tlačítka a přepínače	0x00
otočný knoflík	0x01
lcd data	0x02
vyšší byte výstupu děličky	0x03
nižší byte výstupu děličky	0x04
výstup děličky pro zbytek	0x05
datový výstup přijímače sériové linky	0x06
stavový výstup přijímače sériové linky	0x07

7.7.2 Výstupní porty

Zde je uveden seznam obvodů a příslušných výstupních portů, s nimiž funkce této kapitoly počítají.

Zařízení	Číslo portu
vyšší byte 1. operandu děličky	0x10
nižší byte 1. operandu děličky	0x20
2. operand děličky	0x30
LCD data	0x40
datový vstup vysílače sériové linky	0x50
vstup generátoru pulzů sériové linky	0x60
potvrzení přečtení znaku z přijímače sériové linky	0x70
LED kontrolky	0x80
vstup přepínače programových block RAM	0x90

Kapitola 8

Demo aplikace

Součástí této práce bylo i vytvoření aplikace pro vylepšený KCPSM3, která by demonstovala funkčnost všech přidaných komponent a vytvořených funkcí. Aplikace je rozložena na všechny 4 dostupné programové RAM, využívá všech funkcí knihovny picolib i prvků Spartan 3E, se kterými interagují.
Je třeba poznamenat, že ačkoliv se nejedná o příliš složitý program, zabírá téměř celou kapacitu 4 programových RAM a to i přes značnou snahu autora psát úsporně. Samotný tento fakt demonstruje, že picoBlaze je skutečně pico a příliš se nehodí na realizaci komplexních programů.

8.1 RAM 0 - Menu

RAM 0 je pamětí ze které picoBlaze začíná číst po restartu a je zde umístěno jednoduché menu. Volbu v menu měníme pomocí rotace otočného knoflíku a výběr potvrdíme jeho stlačením. Volby jsou:

1. Kalkulačka
2. Vysílač UART
3. Přijímač UART
4. Světelná show

Zvolíme-li světelnou show, následuje rychlá demonstrace funkčnosti svítivých diod přípravku. Zvolíme-li jinou možnost, přejde picoBlaze na jednu z dalších RAM.

8.2 RAM 1 - Kalkulačka

Jednoduchá kalkulačka umožňující sečtení nebo odečtení dvou operandů z rozmezí 0-9999. Výsledek bude správně zobrazen, i pokud je záporný. Operandy jsou voleny postpně pomocí rotace otočným knoflíkem a potvrzeny jeho stiskem. Po zadání druhého operandu je zobrazen výsledek a po dalším stisku, následuje návrat zpět do hlavního menu. Tato část demonstruje především práci s děličkou, bez které bychom nebyli schopni zobrazovat čísla v desítkové soustavě.

8.3 RAM 2 - Vysílač UART

Tato část demonstruje funkčnost vysílače sériové linky. Nejprve je třeba zvolit pomocí otočného knoflíku vysílací rychlost z nabídky a po té můžeme po jednom volit znaky a stiskem knoflíku je odesílat. Odeslané znaky se též vypisují na displej. Do hlavního menu se můžeme kdykoliv vrátit stiskem tlačíka nahoru (north). Vysílač je standardně připojen na DCE (Female) sériový port Spartan 3E.

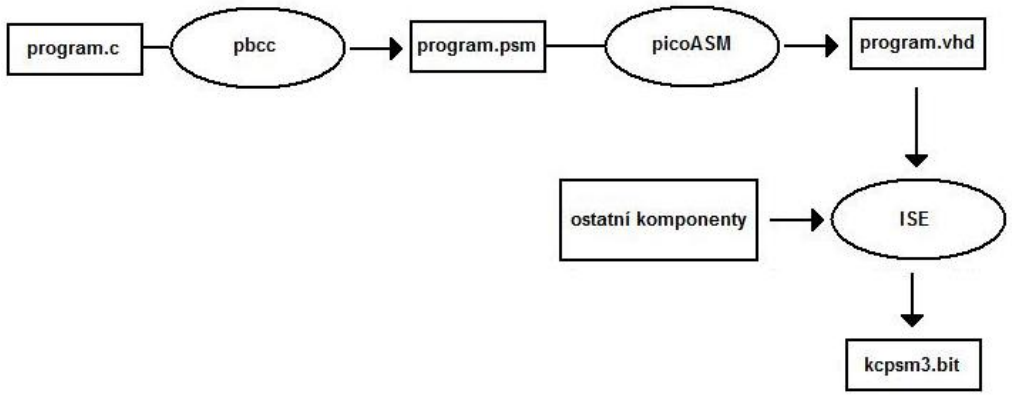
8.4 RAM 3 - Přijímač UART

Tato část obsluhuje přijímač sériové linky. Nejprve otočným knoflíkem zvolíme vysílací rychlost a poté již automaticky začne přenos. Veškeré přijaté znaky jsou vypisovány na displej. Přijímač je standardně připojen na DCE (Female) sériový port Spartan 3E.

Kapitola 9

Programování pro picoBlaze

Tvorba, překlad a ladění programů pro picoBlaze je netriviální úkol a skrývá mnohá úskalí. Účelem této kapitoly je srozumitelně popsat postup, jakým byla vytvářena knihovna a demo aplikace, které jsou součástí této práce, a ušetřit čas každému, kdo se snad bude v snažit na tuto práci navázat. Následující text popisuje celý proces od tvorby zdrojového kódu v C po nahrání bitstreamu do FPGA.
Na tomto místě je také důležité připomenout, že picoBlaze je malý procesor pro malé úkoly. S vylepšeními realizovanými v této práci lze vytvořit poměrně rozsáhlý program, to však neznamená, že picoBlaze je pro provádění velkých komplexních programů ideální platformou.



Obrázek 9.1: Průběh překladu programu

9.1 Zdrojový kód v C

Překladač pbcc podporuje většinu základních konstrukcí jazyka C, jako proměnné včetně staticky deklarovaných polí, podmínky, cykly a funkce a běžné datové typy jako char (8 bit) a int (16 bit). Při psaní kódu je však třeba mít na paměti určité specifické vlastnosti a postupovat opatrně, neboť při překladu některých konstrukcí může dojít ke generování chybného kódu.

9.1.1 Překlad

Jak již bylo zmíněno, překladač pbcc je ideální instalovat do emulátoru cygwin pod libovolným systémem Windows. Zdrojový C soubor umístíme do pracovního adresáře a samotný řeklad provedeme příkazem:

```
/pbcc/sdcc3/bin/sdcc.exe -c -I/pbcc/sdcc3/device/include/pblaze
 -mpblaze <zdrojový soubor.c>
```

Ve vráceném výpisu ignorujeme chybu:

```
sh: aspblaze: command not found
```

a také varování `warning 85` produkované funkcemi knihovny picolib.
Výstupem je pak soubor s příponou .psm, který můžeme dále zpracovat překladačem jazyka symbolických instrukcí (picoASM).

9.1.2 Velikost programu

Jak již bylo zmíněno dříve, počet instrukcí, které může program obsahovat je značně omezen a proto je doporučeno psát kód úsporně.
Nejdůležitějším faktorem pro velikost se ukázal být počet deklarovaných proměnných a četnost volání funkcí. V přeloženém kódu jsou před voláním každé funkce veškeré lokální proměnné odloženy na zásobník (do scratchpad paměti) a po návratu z funkce jsou opět vyjmuty a vráceny do registrů. Jak uložení, tak vyjmutí zabere 2 instrukce za každý byte proměnné. Máme-li tedy např. lokální proměnné ve funkci main o celkovém objemu 10 bytů, pak při každém volání podfunkce spotřebujeme

```
10 * (2 + 2) = 40
```

instrukcí pouze na odkládání proměnných na zásobník. Proto se důrazně doporučuje omezit objem lokálních proměnných na nutné minimum a volat raději jednu komplexnější funkci než několik jednodušších.

9.1.3 Chyby v pbcc

Pbcc je použitelný nástroj, avšak zkušenost úkázala, že obsahuje i několik chyb, které pravděpodobně nebudou opraveny. Proto ladíme-li program, je třeba v neposlední řadě zkontrolovat, zda proběhl překlad z C do picoASM správně. Následuje výčet nejproblematičtějších konstrukcí.

- Datový typ int - pbcc uomžňuje práci i s 16 bitovým typem int, avšak zkušenost ukázala, že je lepší se mu vyhnout, pokud je to možné. Mezi hlaví problémy patří přetypování mezi char a int, která ne vždy dopadnou tak, jak je podle konvencí jazyka C očekáváno. Dalším problémem je porovnávání dvou proměnných typu int mezi sebou pomocí operátorů < a >, jehož výsledek v některých případech není správný.
- Podmíněné přiřazování do proměnné - překlad konstrukcí typu `if - else` obvykle dopadne dobře. Vyjjímka může nastat, přiřazujeme-li v jedné větvi konstrukce do proměnné a v jiné větvi ne. V takovém případě může překladač ztratit přehled o tom, ve kterém registru má dannou proměnnou uloženou a výsledný program se nemusí chovat tak, jak bychom očekávali.

9.2 Kód v psm

Psm nebo-li pico assembler je jazyk symbolických instrukcí používaný picoBlazem. Detailní popis všech instrukcí nalezneme v manuálu KCPSM3, ale kód je srozumitelný i bez manuálu., neboť se do značné míry podobá např.x86 assembly.

Za normálních okolností požadujeme pouze soubour .psm vygenenerovaný pomocí pbcc zpracovat pico-překladačem na výstupní kód VHDL. To provedeme příkazem:

```
kcpsm3.exe <zdrojový soubour bez přípony .psm>
```

Předpokládá se, že jak program KCPSM3.exe, tak zrojový soubor se nalézají v pracovním adresáři a pracujeme v prostředí DOSBox, nebo pod 32-bit systémem Windows. Nejčastější chybou je překročení kapacity paměti pro program, v takovém případě žádný výstup vygenerován není.

Z výstupních souborů je nejdůležitější `<jméno programu>.VHD`, obsahující definici samotného bloku paměti, který zakomponujeme do celkového projektu.

9.3 Komponenta ve VHDL

Soubor .VHD, který jsme vygenerovali v předchozím kroku obsahuje definici celého bloku paměti v jazyce vhdl připravení k použití například v prostředí Xilinx ISE a nevyžaduje již dalších zásahů. Komponentu vložíme do projektu, její port *adress* je třeba připojit k adresnímu výstupu KCPSM3, port *instruction* ke komponentě přepínače RAM a port *clk* k hodinovému signálu.

Kapitola 10

Možná vylepšení

Šíře projektu, jakým je PicoBlaze, je obrovská a možnosti jsou téměř neomezené. Dalšími rozšířeními přímo navazujícími na tuto práci by mohli být:

1. Vytvoření HW a SW podpory pro nějaký sofistikovanější ovládací prvek, např. klávesnici.
2. Vytvoření násobičky. Logický krok po implementaci obvodu pro dělení.
3. Vytvořit plnou podporu pro větší oběm paměti pro program. Takovéto rozšíření by vyžadovalo zásahy přímo do jádra procesoru, včetně úprav některých instrukcí. Náročný úkol.

Závěr

Přes značné obtíže se podařilo realizovat většinu cílů v rozumném měřítku a práce snad bude v budoucnu prospěšná pro další, kteří se na ni pokusí navázat.

Smutnou skutečností je, že ačkoliv byla tvorba této práce vysoce poučná a často i zábavná, dostal se autor s přibližujícím se termínem odevzdání do značné časové tísně, zapříčiněné zejména protahujícím se vývojem děličky. Vlivem tohoto kvalita práce pravděpodbně utrpěla.

Literatura

[1] Novotný, M.: Praktika v návrhu číslicových obvodů. Přednásky bakalářského studia, 2013.

[2] Pluháček, A.: Jednotky počítače. Přednásky bakalářského studia, 2013.

[3] Wikipedia: Field-programmable gate array — Wikipedia, The Free Encyclopedia. 2013, [Online; accessed 1-February-2013]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Field-programmable_gate_array>

[4] Wikipedia: Soft microprocessor — Wikipedia, The Free Encyclopedia. 2013, [Online; accessed 1-February-2013]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Soft_processor>

[5] Xilinx: *KCPSM3_Manual*. Dostupné z WWW: <http://www.xilinx.com/products/intellectual-property/picoblaze.htm>

[6] Xilinx: *UART_Manual*. Dostupné z WWW: <http://www.xilinx.com/products/intellectual-property/picoblaze.htm>

Příloha **A**

Seznam použitých zkratek

FPGA Field programmable gate array

VHDL VHSIC Hardware Description Language

VHSIC Very-high-speed integrated circuits

KCPSM Konstant Coded Programmable State Machine

ASM Assembler - jazyk symbolických instrukcí

RAM Random access memory

ROM Read only memory

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `project` ..... adresář obsahující kompletní projekt, se všemi rozšířeními a demo aplikací pro vývojové prosředí Xilinx ISE, včetně bitstreamu pro nahrání do Spartan 3E
- `src` ..... zdrojové kódy
  - `C` ..... adresář obsahující zdrojové kódy v jazyce C
    - `picolib.c` ..... knihovna funkcí pro práci s picoBlaze
    - `program0-3.c` .. zdrojový kód demo aplikace pro block RAM 0-3
  - `VHDL` ..... zdrojové kódy v jazyce VHDL
    - `original` . adresář s původními zdrojovými kódy od spol. Xilinx
    - `created` .. adresář se zdrojovými kódy vytvořenými v rámci této práce
  - `BP_Filip_Matous_2014.tex` zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář obsahující text práce
  - `BP_Filip_Matous_2014.pdf` ..... text práce ve formátu PDF
  - `BP_Filip_Matous_2014.ps` ..... text práce ve formátu PS