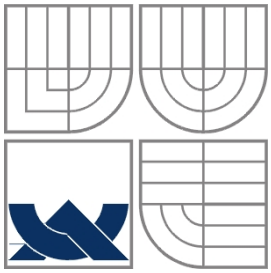
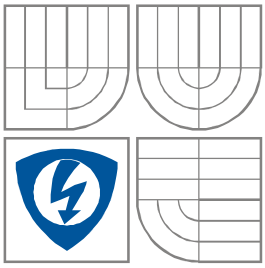


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

IP generátor mikroprocesorového systému

Microprocessor system IP core generator

DIPLOMOVÁ PRÁCE
MASTER'S PROJECT

AUTOR PRÁCE — Bc. Rostislav Kerber
AUTHOR

VEDOUCÍ PRÁCE — Ing. Michal Kubíček, Ph.D.
SUPERVISOR

BRNO, 2011

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Rostislav Kerber	***ID:***	98117
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

IP generátor mikroprocesorového systému

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s návrhovým systémem ISE, jazykem VHDL a mikroprocesorem PicoBlaze. Navrhněte blokové schéma systému s mikroprocesorem PicoBlaze s periferiemi. Vytvořte popis nejběžněji používaných periferií procesoru (řadič přerušení, řadič LCD displeje, UART,...)

Ve vhodném vývojovém prostředí na PC vytvořte program pro generování IP jádra procesoru PicoBlaze s periferiemi.

Ověřte funkci navrženého systému procesoru a generátoru IP jádra na vhodné aplikaci. Vytvořte zadání laboratorní úlohy, kde bude využit generátor IP jádra procesoru PicoBlaze.

DOPORUČENÁ LITERATURA:

[1] Xilinx, Inc., 2100 Logic Drive San Jose, CA 95124-3400, USA. PicoBlaze 8-bit Embedded Microcontroller User Guide, [online], 2004 [cit. 18. 12. 2009]. Dostupné na www: http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm

Termín zadání:	7.2.2011	***Termín odevzdání:***	20.5.2011

Vedoucí práce: Ing. Michal Kubíček, Ph.D.

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práce se týká programovacího jazyku VHDL, návrhového systému ISE Webpack a mikroprocesoru PicoBlaze. Popisuje základy jazyka VHDL a jeho použití. Dále je v práci popsán způsob práce s programem ISE Webpack. V práci jsou popsány nejběžnější periferie a je zde popsán také Picoblaze procesor s jeho parametry a realizace. Na konec je zde popsán IP generátor pro generaci komplexního FPGA návrhu s procesorem Picoblaze.

KLÍČOVÁ SLOVA

VHDL, Xilinx, ISE Webpack, procesor PicoBlaze

ABSTRACT

This master's thesis deal's with VHDL programming language, ISE Webpack design system and PicoBlaze microprocessor. The thesis describes essentials of VHDL programming language and its application. A simple introduction to ISE Webpack design environment is given. The thesis describes common peripherals and the PicoBlaze processor is described too, including its parameters and implementation aspects. Finally the thesis describes IP generator for generating complex FPGA design including Picoblaze processor.

KEYWORDS

VHDL, Xilinx, ISE Webpack, PicoBlaze processor

Kerber,R. IP generátor mikroprocesorového systému. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2011. 56 s., 0 s. příloh. Diplomová práce. Vedoucí práce: Ing. Michal Kubíček, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma IP generátor mikroprocesorového systému jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Michal Kubíček, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

Obsah

Seznam obrázků

1. Úvod

V rámci návrhu IP generátoru je požadavek na možnost snadné generace hotových popisů procesoru PicoBlaze s navolenými periferiemi. V některém programovacím nástroji vyšších programovacích jazyků např. Borland C++ bude vytvořen generátor s interaktivním menu pro volbu požadovaných periferií a jejich nastavení. Program bude vycházet z již předpřipravených polotovarů zdrojových souborů a v rámci svého programu je bude podle předvoleb spojovat do jednoho komplexního návrhu, který bude splňovat všechny požadavky uživatele. Bude nutné zajistit správnou fúzi jednotlivých zdrojových souborů, aby se dosáhlo syntaktické i sémantické správnosti výsledného souboru.

2. VHDL

Jazyk VHDL vznikl na základě jazyků HDL (Hardware Description Language). Jeho vznik je provázen spoluprací několika firem a jeho finální podobu a i jeho další nadstavby má na svědomí organizace IEEE, která z něho vytvořila otevřený standart. Díky tomu tento jazyk není zatížen autorským právem a to umožňuje jeho velké rozšíření. Primárně tento jazyk vznikl pro modelování a simulaci velkých číslicových systémů, u kterých je syntéza příliš komplikovaná, či nemožná[1]. Tento fakt ovšem neznamená, že jazyk VHDL nelze použit pro číslicové obvody, pro které syntéza smysl má, či je přímo žádaná.

Základní vlastnosti VHDL:

- Použití VHDL není zatíženo autorským právem. To činí jeho používání mnohem méně finančně náročným.

- Není nutné volit cílový obvod hned na začátku návrhu. Ten lze zvolit až v době, kdy jsou známy všechny detaily jeho použití. Kód zapsaný podle pravidel HDL je možno použít jak v obvodech PLD, tak i FPGA.

- Kód zapsaný ve VHDL je přenositelný, tedy kód zapsaný ve VHDL (s nepříliš složitým či neobvyklým způsobem zápisu) je možno použít v simulátorech a syntetizátorech různých výrobců a ověřený funkční kód lze použít znovu v jiném projektu.

- Tak jako vyšší programovací jazyky i ve VHDL lze vytvářet neefektivní bloky a případná efektivnost vytvořeného kódu je závislá na kvalitě návrhových prostředků a schopnostech konstruktéra[1].

Jazyk VHDL umožňuje tři typy sestavování modelu:

- Zdola nahoru – Postupným vytvářením dílčích bloků a jejich skládáním do celků se vytváří finální model.

- Shora dolů – Na základě funkce požadovaného systému se vyčleňují jednotlivé bloky a jejich vzájemné propojení. Tento postup pokračuje dokud se nedosáhne takových bloků, které jsou dostatečně jednoduché pro popsání například behaviorálním stylem.

- Plochý typ – Na rozdíl od předchozích dvou typů se u tohoto typu modelu finální model nedělí na dílčí bloky, ale vytváří se jako jeden kompaktní celek[1].

Každý schopný konstruktér by měl mít tendenci psát zdrojové kódy co nejpřehledněji tak, aby se mohl snadno orientovat v kódu. To umožňuje hierarchičnost prvních dvou typů sestavování modelu s tím, že v praxi se častěji využívá typu „Shora dolů“[1].

2.1 Struktura modelu

Základní model v jazyku VHDL se skládá ze dvou částí: - Deklarace entity
- Popis Architektury

Deklarace entity popisuje vstupy a výstupy z popisovaného bloku. Pokud se jedná o vrcholovou jednotku v hierarchickém typu, pak deklarace entity popisuje fyzické vstupy a výstupy zvoleného cílového obvodu[1]. V popisu architektury se definuje funkce popisovaného modelu. Pro snazší orientaci se ve zdrojovém kódu následující řádky opatřily pořadovým číslem. Ty však při skutečné práci v návrhovém systému být mohou jen ve formě komentáře. V následujícím zdrojovém kódu je popsán jednoduchý čítač, který na základě přivedeného hodinového signálu po dosažení hodnoty 1111 binárně (15 dekadicky) zapíše logickou jedničku na výstupní port, což může být povolení operace pro jinou architekturu. Pokud bychom vzali v úvahu základní hodinový signál 50MHz, pak by výsledná frekvence byla 3,3MHz. Jde tedy o jednoduchý způsob, jak nastavit např. přenosovou rychlost periferie typu UART.

```
--text 1.
LIBRARY ieee;                                                    --1.
USE ieee.std_logic_1164.ALL;                                     --2.
USE IEEE.STD_LOGIC_ARITH.ALL;                                    --3.
USE IEEE.STD_LOGIC_UNSIGNED.ALL;                                 --4.
-----------------------------------------------------------------
ENTITY citac IS                                                  --5.
      PORT( clk:        IN std_logic;                            --6.
            clk_EN:     OUT std_logic);                          --7.
END citac;                                                       --8.
-----------------------------------------------------------------
ARCHITECTURE Behavioral OF citac IS                              --9.

SIGNAL Cit:  STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');    --10.

BEGIN                                                            --11.
-----------------------------------------------------------------
Arch_citac: PROCESS (clk) BEGIN                                  --12.
      IF clk'EVENT AND clk = '1' THEN                            --13.
            IF Cit =  "1111" THEN                                --14.
                  Cit(3 downto 1) <= (OTHERS => '0');            --15.
                  Cit(0) <= '1';                                 --16.
                  clk_EN <= '1';                                 --17.
            ELSE                                                 --18.
                  clk_EN <= '0';                                 --19.
                  Cit <= Cit + '1';                              --20.
            END IF;                                              --21.
      END IF;                                                    --22.
END PROCESS Arch_citac;                                          --23.
-----------------------------------------------------------------
END Behavioral;                                                  --24.
```

2.1.1 Deklarace entity

Deklarace entity začíná na řádku 5 a končí na řádku 8 v ukázce. V deklaraci entity se vstupní a výstupní brány uvádějí v těle příkazu PORT. Takováto deklarace obsahuje celkem tři části. První část je jméno brány, které poté figuruje při každém jejím použití

v popisu architektury. Druhá část je mód, ve kterém brána pracuje a třetí část je její datový typ.

Druhy módů:

- `IN` - vstup
- `OUT` - výstup
- `BUFFER` - výstup se zpětnou vazbou
- `INOUT` - obousměrný vývod.

Mód OUT nelze použít v těle příkazu. Jinak řečeno ho nelze použít pro přiřazení hodnoty jakékoliv proměnné. Tento problém řeší mód BUFFER, avšak kvůli možným problémům při kompilaci u některých systémů se doporučuje kombinace módu OUT a vnitřního signálu deklarovaného v popisu architektury. Viz. kapitola níže. Poslední možností je použití módu INOUT, avšak je zde problém vzniku zbytečné komplikace při simulaci[1]. Z praktického hlediska se tedy často volí právě kombinace módu OUT a vnitřního signálu jemu přiřazeného.

Datových typů je v jazyku VHDL celá řada. Mezi nejpoužívanější patří datové typy předdefinované v základních standardech IEEE. Tyto standarty jsou popsány v knihovně `ieee.std_logic_1164.ALL`. Jejich hlavním představitelem jsou typy `STD_LOGIC` a jejich složené verze. Tyto spolu s vlastními typy, definovanými konstruktérem, jsou nejčastěji používánými typy ve VHDL[1].

2.1.2 Popis architektury

Pro lepší orientaci ve zdrojových kódech v jazyku VHDL lze způsoby, kterými se architektury popisují rozdělit do tří skupin.

- **Behaviorální popis** - Obvykle je tento styl spojován s použitím vyhrazeného slova `PROCESS`. V textu 1. je tedy tento styl charakterizován řádky 12. a 23. Jedna se o jakési ohraničení navrhovaného bloku. Začíná jeho názvem před dvojtečkou. Poté následuje klíčové slovo `PROCESS` a za ním v závorce seznam vnějších signálů, na které popisovaný proces reaguje. Vlastní tělo procesu je pak ohraničeno slovy `BEGIN` a `END PROCESS` s příslušným názvem procesu.

- **Popis toku dat** - Jedná se o zjednodušenou variantu behaviorálního popisu, kde absenci klíčových slov `BEGIN` a `END PROCESS` zastupují použité souběžné příkazy[1].

- **Strukturální popis** - Při tomto stylu se vkládají tzv. komponenty do kódového útvaru nazývaného netlist[1]. Tento styl se nejčastěji používá při popisování propojení bloků nižší úrovně v hierarchickém uspořádání. Konstruktér si vytvoří vlastní knihovnu, která obsahuje popisované komponenty. Použitím takovéto knihovny pak konstruktérovi dovolí použít tyto vlastní komponenty pro práci se signály. Strukturální styl se většinou používá v situacích, kdy behaviorální popis po syntéze vedl k jiné struktuře než bylo požadováno.

2.2 Datové objekty

Datovým objektem se rozumí objekt, který využívá architektura pro svou funkci. Jsou to předměty, které potřebují VHDL příkazy pro své vykonání.

Druhy datových objektů:

- **Konstanty** - Mají předem určenou neměnnou hodnotu, která se uvádí při jejich deklaraci. Konstant se využívá při častém použití neměnného čísla v konstruktech pro jejich zpřehlednění. Navíc lze takto využitou konstantu snadno měnit. Praktické využití je pak například při definování přenosové rychlosti sériové linky UART.

```
CONSTANT Rychl: integer := 5208;
```

- **Signály** - Signály v konstrukci zpravidla reprezentují skutečné elektrické signály v obvodu. Jejich deklarace se dělí na deklaraci jako `PORT` (viz řádek 6 a 7 v textu 1.), které přísluší zvolený mód a nebo na deklaraci typu `SIGNAL` (viz řádek 10 v textu 1.). Stejně jako konstantám i signálům může být přiřazená počáteční hodnota. Avšak toto přidělení má význam pouze pro simulaci. Při syntéze se ignoruje[1].

- **Proměnné** – Nejsou skutečné elektrické signály. Používají se jen jako pomocné objekty pro texty určené k simulaci. Při psaní zdrojových textů pro syntézu se jejich používání nedoporučuje kvůli možným problémům s jejich zpracováním návrhovými systémy. Zpravidla se používají jen tam, kde to doporučuje dokumentace k použitému návrhovému systému[1].

```
VARIABLE SwC: std_logic_vector(4 DOWNTO 0);
```

- **Soubory** – Použití souborů ve VHDL má stejný význam jako jejich použití v kterémkoliv jiném programovacím jazyku. Avšak ve VHDL se tento datový typ využívá pouze při simulaci pro uchování získaných dat. Při syntéze se opět ignorují.

2.2.1 Typy dat

Jazyk VHDL si vynucuje jasné uvedené konkrétního typu dat u deklarace Datových typů. Hlavním důvodem je zabezpečení korektního použití. Například nemožnost použít datové pole typu `STD_LOGIC_VECTOR` (lze uložit pouze 0 a 1) k uložení textu.

Nejdůležitější datové typy:

- Skalární
  - Výčtové
  - Celočíselné
  - S pohyblivou čárkou
  - Fyzické

- Složené
  - Pole
  - Záznam

Výčtový typ má stavy, kterých může nabývat, definovány v závorce v deklaraci typu. V tomto výčtu záleží na pořadí. Položka nejvíce vlevo se pokládá za nejmenší či první a každá následující pak za větší či na řadě. Standart IEEE definuje dva důležité výčtové typy `BIT` a `BOOLEAN`[1]. Často se lze setkat s vlastními výčtovými typy definovanými konstruktérem. Ty se zpravidla používají u stavových automatů.

Celočíselný typ je charakterizován klíčovým slovem INTEGER a systémové nástroje pracující s tímto typem musí umět pracovat s čísly od $-(2^{31}-1)$ do $(2^{31}-1)$[1]. Pro syntetizaci se však tento rozsah musí vhodně upravit podle požadavků.

Pole se využívá pro uchování více dat stejného typu. V poli se data řadí do řetězce. Naplnění takového pole se provádí přiřazovacím příkazem, přičemž vlastní hodnoty se zadávají buď jako celek v uvozovkách "00011011" a nebo po jedné hodnotě v apostrofech '0'.

Důležitou odvozeninou od standardních typů je tzv. subtyp. Často se používá v kombinaci s poli. Jde vlastně o mezení základního rozsahu použitého typu. Jeho syntaxi a použití je vidět na řádku 10 v textu 1. Deklarované pole bez subtypu má neomezenou velikost. Při praktickém využití konstruktér omezí toto pole na velikost dostačující pro jeho použití. Zmiňovaný subtyp je v textu 1 reprezentován výrazem `(3 DOWNTO 0)`. Tím se neomezená velikost pole omezila na velikost 4 bitů.

2.3 Příkazy ve VHDL

Příkazy ve VHDL se dělí na souběžné a sekvenční. Toto rozdělení ve skutečnosti týká toho, kde v popisu se tyto příkazy mohou nacházet[1].

Souběžné příkazy jsou příkazy, které se nemohou naházet v procesech, definici funkcí a procedur[1]. U souběžných příkazu při kompilaci nezáleží na jejich pořadí. Patří sem:

- Podmíněné přiřazovací příkazy `(WHEN-ELSE)`
- Výběrové přiřazovací příkazy `(WITH-SELECT-THEN)`
- Procesy jako celky
- Vložené komponenty
- Příkaz `GENERATE[1]`

Sekvenční příkazy mohou být naopak jen v procesech, definici funkcí a procedur. U sekvenčních příkazů při kompilaci na jejich pořadí naopak záleží.
Patří sem:

- Příkazy určené pro proměnné
- Příkazy `(IF-THEN-ELSE)`
- Příkazy `(CASE-WHEN)`
- Smyčky `(LOOP, NEXT, EXIT)[1]`

2.3.1 Operátory

Výrazy (příkazy) jsou identifikátory, které se spojují pomocí operátorů. Pomocí operátorů se konkretizuje požadavek na použitý příkaz.

- **Logické operátory** - `AND`, `OR`, `NAND`, `NOR`, `XOR`, `XNOR` a `NOT`. Tyto operátory mají klasický význam, který je znám z číslicové techniky.
- **Relační operátory** - `=`, `/=`, `<`, `<=`, `>`, `>=` Relační operátory se používají buď v přiřazovacích příkazech nebo v příkazech pro porovnávání.
- **Operátory posuvu a rotace** – `SLL`, `SRL`, `SLA`, `SRA` , `ROL`, `ROR`
- **Aritmetické operátory** - `+`, `-`, `&`. První dva jsou klasické sčítání a odečítání. Poslední operand je operandem sjednocení.

2.4 Postup pro vytvoření konstrukce ve VHDL

Funkce, která je obvykle v praxi požadována je příliš složitá na to, aby konstruktér zpracovával jako celek. Je proto nutné ji rozdělit na dílčí bloky takové úrovně, kdy je konstruktér schopen dostatečně přesného popisu. Tento způsob práce je popsán v kapitole 1.1.2 Popis architektury. Obvykle se nejnižší struktura popisuje behaviorálním style a její propojení ve vyšší struktuře popisuje stylem strukturálním.

Obvyklý postup návrhu:

- Popis
- Syntéza
- Implementace
- Simulace

Popis lze vytvářet dvěma možnými způsoby. Tím základním je popis v některém z textových editorů. Jako editor schopný vytvořit popis použitelný pro syntézu lze využít kterýkoliv editor, který nepoužívá formátovacích znaků. Návrhové systémy určené pro VHDL obvykle obsahují pokročilé editory, které jsou schopny zpřehlednit práci konstruktérovi pomocí barevného zvýraznění klíčových slov, či pomocí odsazování textu. Propracované návrhové systémy obsahují ještě navíc i druhý způsob vytvoření popisu pomocí grafického návrhu. Nejčastěji se jedná o kreslení stavových automatu, pravdivostních tabulek, či kreslení schémat[1]. Tyto návrhové systémy se pak i starají o správný převod těchto grafických popisů do standardního VHDL popisu.

Syntéza je převod popisu (ať už textového či grafického) do tvaru, se kterým si poradí implementační nástroje. Součástí Syntézy obvykle bývá optimalizace, která upravuje syntézu pro konkrétní použitou technologii, a někdy i kompilace jako mezistupeň mezi syntézou a implementací[1].

Implementace vytváří strukturu, která odpovídá strukturám pro daný cílový obvod. Z implementace vychází soubory, kde jsou finální parametry pro naprogramování cílového obvodu.

Simulace slouží konstruktérovi pro kontrolu a ověření navrhnutého modelu Zde se porovnává výsledná funkce modelu s požadavky.

3. ISE WebPack

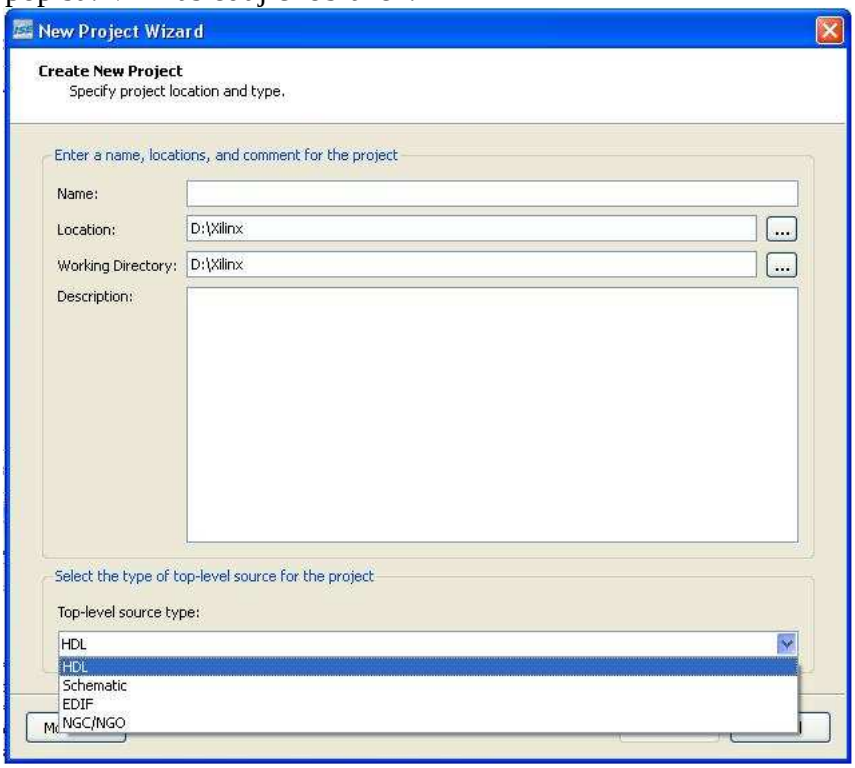
ISE WebPack je vývojové prostředí od firmy Xilinx, který umožňuje efektivně a pohodlně modelovat a simulovat programovatelné logické obvody od této firmy. ISE WebPack je multiplatformní, volně šiřitelný a lze pomocí něho pracovat s obvody CPLD a FPGA.

3.1 Realizace konstruktu

Zde bude popsán základní postup při realizaci konstruktů v jazyku VHDL. Program ISE webpack je samozřejmě velice sofistikovaný a obsahuje mnohem více nástrojů než zde bude popsáno. Pro základní ucelení práce s tímto programem však bude následující část dostačující.

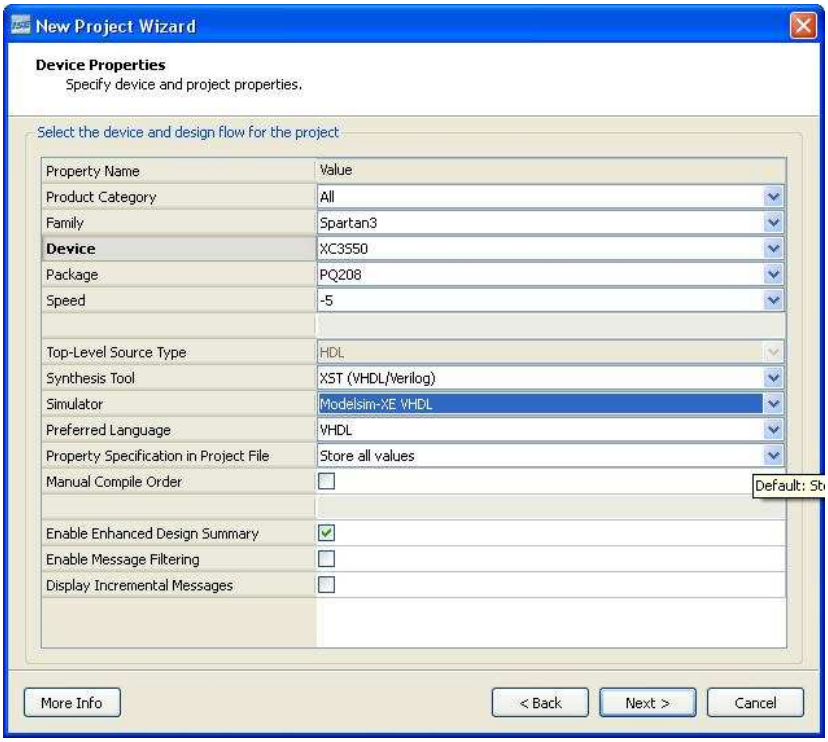
3.1.1 Popis

Po spuštění programu ISE webpack otevřeme wizard pro vytvoření nového projektu *file->new project.* Otevře se úvodní okno pro zadání názvu projektu. Kvůli možným problémům s hlášením chyb při kontrole syntaxe se doporučuje v názvu neužívat speciálních znaků a na prvním místě se vyhnout číslicím. Poté je ještě potřeba zadat druh popisu. Viz následující obrázek.



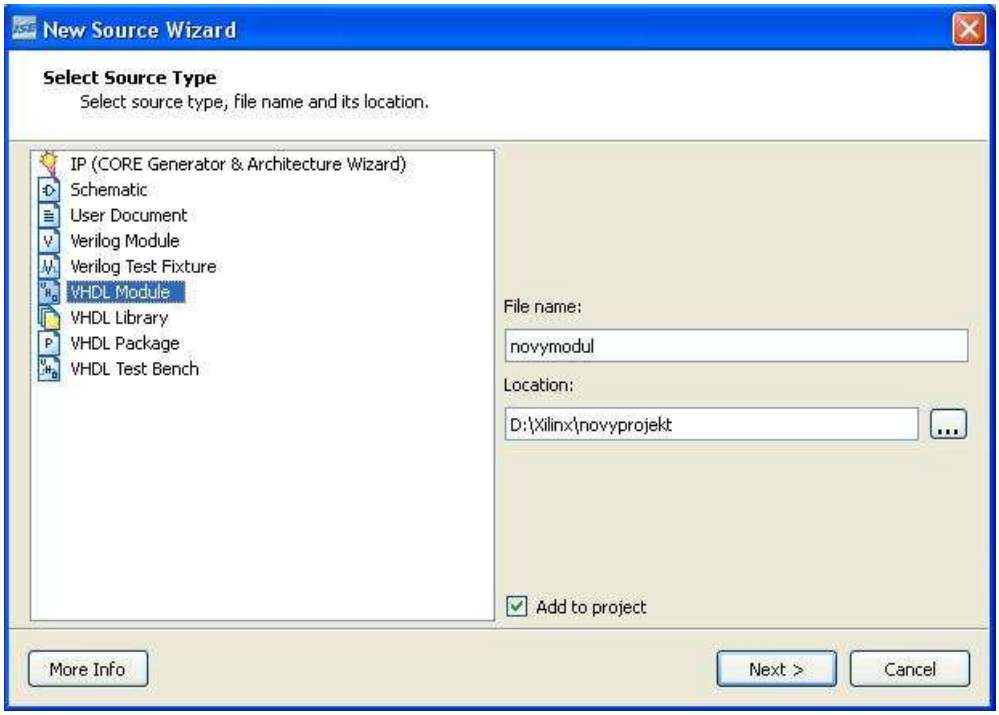
Obr. 3.1. Wizard pro vytvoření nového projektu[2]

Po stisknutí tlačítka next se wizard přesune do okna pro výběr cílového obvodu a simulačního prostředí. U typu cílového obvodu se volí i pouzdro a jeho rychlost. Ze simulačních prostředí je na výběr celá řada. Mimo základního simulačního prostředí ISim je k dispozici ještě Modelsim, NC a VCS s dělením na použití jazyků VHDL, Verilog nebo obojí.



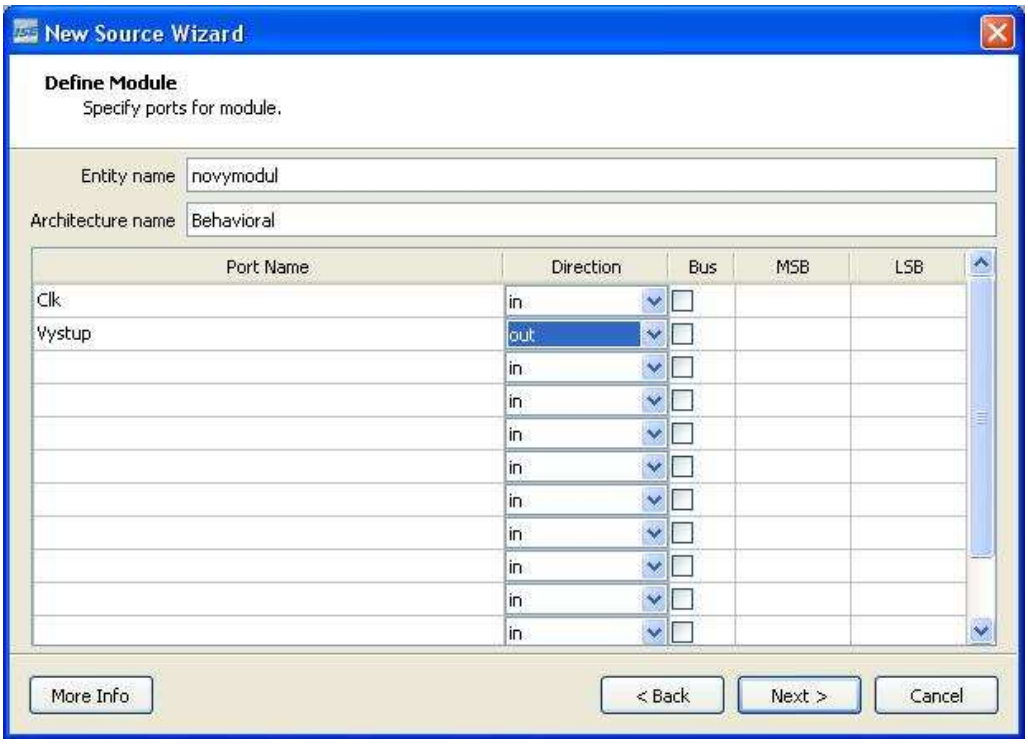
Obr. 3.2 Wizard pro volbu cílového obvodu [2]

Následují dvě rekapitulační obrazovky pro kontrolu zvolených parametrů. Po jejich odsouhlasení se otevře nový prázdný projekt, který je připravený pro tvorbu popisů. Nový zdrojový soubor se přidá přes menu *edit->new source*. Otevře se wizard pro vytvoření nového zdrojového souboru. V levé části je seznam typů zdrojových souborů. V pravé části se zadává jméno souboru a jeho umístění.



Obr. 3.3 Wizard pro vytvoření nového zdrojového souboru [2]

Po potvrzení voleb se zobrazí okno pro zadávání vstupních a výstupních portů. Toto okno není nutné bezpodmínečně vyplnit. Deklaraci portů je možné provést až ve vytvořeném zdrojovém souboru.

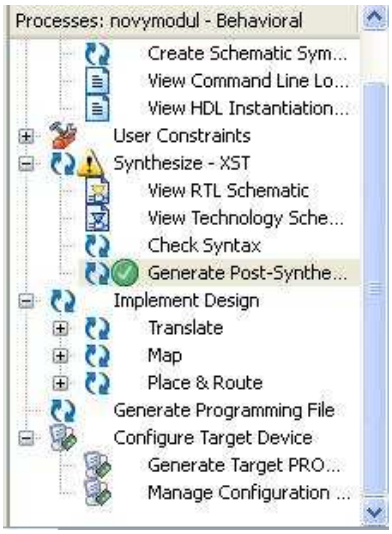


Obr. 3.4 Wizard pro deklaraci portů [2]

Po tomto okně již následuje jen poslední rekapitulační okno po jehož odsouhlasením se vytvoří nový zdrojový soubor připravený k vytvoření popisu.

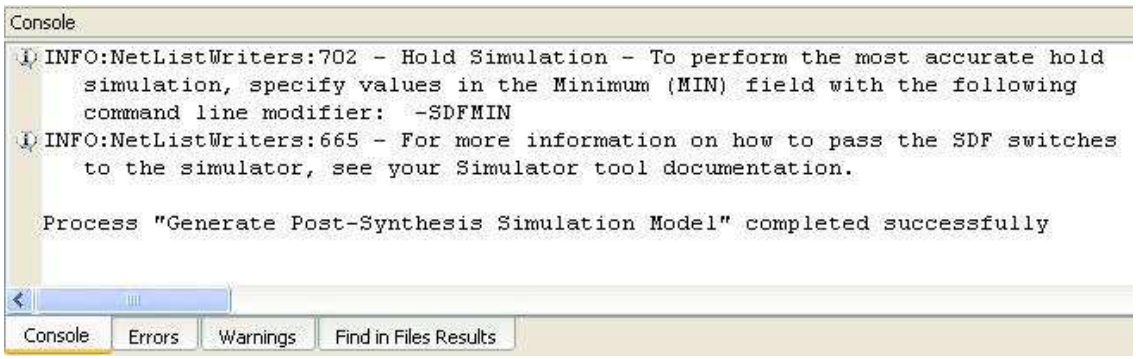
3.1.2 Syntéza

Při syntéze vznikají soubory netlist, které jsou vstupními daty pro implementaci.[3]



Obr. 3.5 Generace simulačního modelu po syntéze[2]

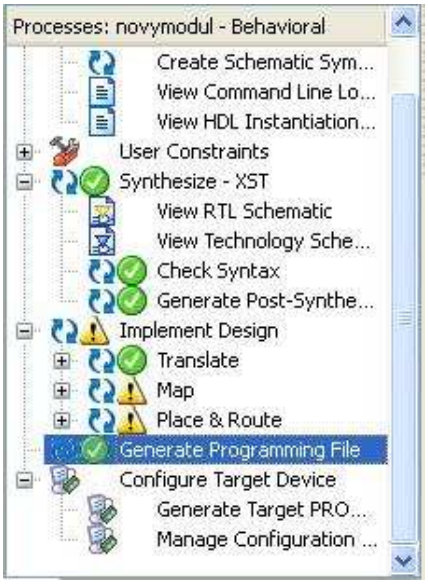
Při syntéze se také kontroluje syntaxe zdrojového kódu a také je to krok nutný pro simulaci modelu po syntéze. Výsledek simulace se zobrazuje v konzolovém okně, kde se vypíší i případné varování či chyby.



Obr. 3.6 Resume po provedení syntézy[2]

3.1.3 Implementace

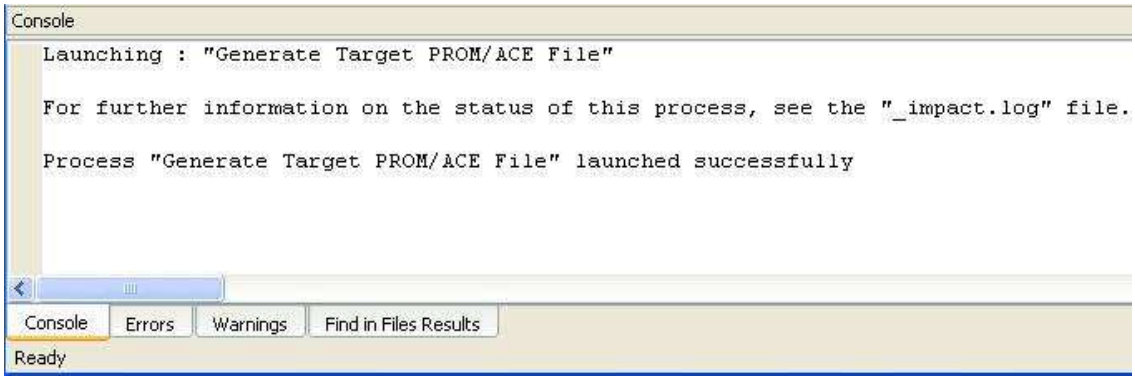
Při implementaci se slučují veškeré soubory vytvořené při syntéze a kontrolu je se kompletní uspořádání. I zde je možné si prohlednout jednotlivé rekapitulace, kde lze nalézt případná varování či chyby. V tomto kroku se také provádí nejoptimálnější návrh konfigurace obvodu FPGA tak, aby bylo dosaženo co největší efektivnosti[3].



Obr. 3.7 Implementace a generování konfiguračního souboru [2]

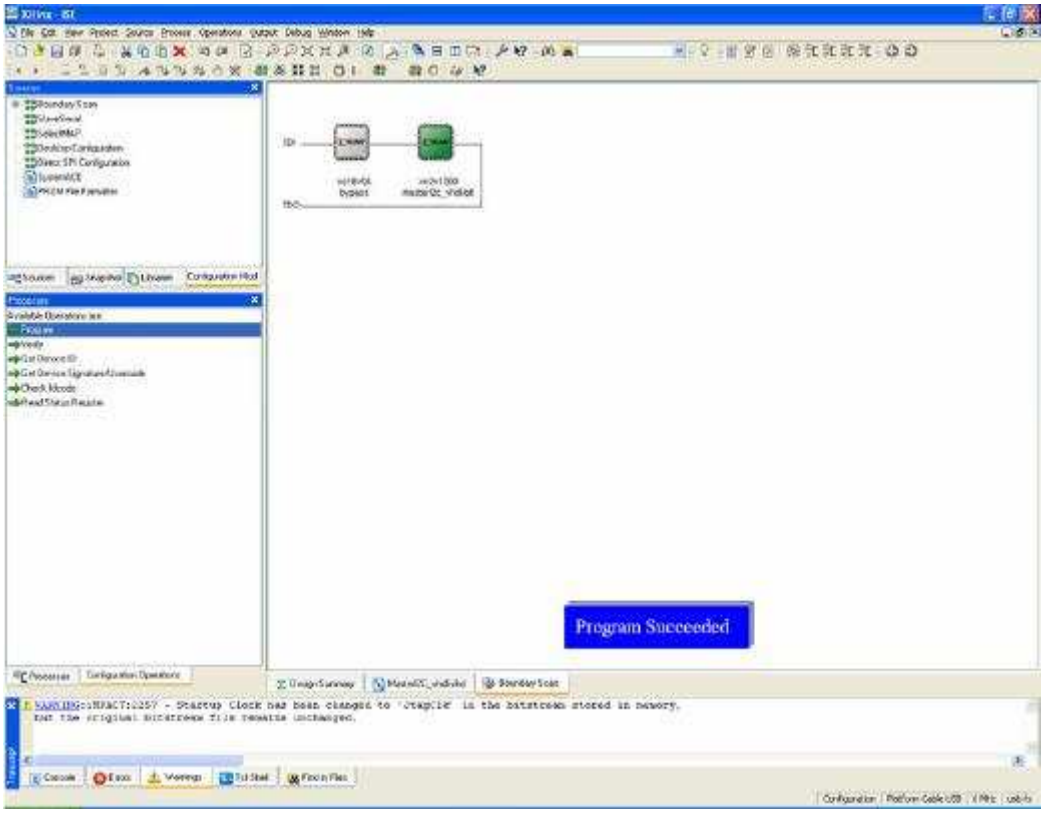
3.1.4 Konfigurace

Po spuštění procesu generování konfiguračního souboru se vytvoří soubor popisující nastavení všech programovatelných buněk v obvodu FPGA (tzv. bitstream)



Obr. 3.8 Vygenerování konfiguračního souboru [2]

S tímto souborem lze již konfigurovat cílový obvod pomocí programu iMPACT. Program je nutné nakonfigurovat na použitý cílový obvod a popřípadě i paměť ze které se bude po zapnutí obvodu nahrávat konfigurační soubor do FPGA.



Obr. 3.9 Konfigurování cílového obvodu [2]

4. Implementace procesorů do obvodů FPGA

Vývoj a stále větší dostupnost obvodů FPGA vedl k použití procesorů jako implementovaných struktur přímo do obvodů FPGA. Dnešní FPGA obvody jsou natolik rozvinuté, že implementovaný procesor v obvodu FPGA již nepotřebuje žádné další přídavné periferie ke své činnosti[5]. Funkce, kterou takový procesor zastává lze samozřejmě zastat standardními strukturami popsanými jazyky VHDL, či Verilog. Procesor je vhodný pro řízení úloh, které mají složitější sekvenční charakter. Často se tyto procesory používají právě kvůli jejich universálnosti. Jejich činnost lze totiž snadno popsat programem s takřka libovolnou funkcí. Hlavním omezením použitelnosti procesorů je jejich výkonnost. Ta se udává v jednotkách MIPS (Million Instructions Per Second) a udává počet instrukcí vykonaných za 1 vteřinu[5]. Tento parametr je pak přímo spjatý s tzv. reakcí procesoru na vnější podnět. Tzn. Počet instrukcí, které procesor musí provést k vykonání určitého algoritmu spjaté spolu s dobu trvání jednoho cyklu. Stejné úvahy lze provádět i s klasickým použitím procesorů ve formě fyzické součástky a klasických číslicových obvodů realizovanými obvodovými součástkami[5]. Jediný rozdíl je v tom, že veškerá navrhnutá konstrukce je implementována v čipu jediné integrované součástky, tedy obvodu FPGA. Rozhodnutí o tom, který způsob je lepší, záleží na konkrétní situaci a požadavcích na procesor. Mimo procesorového jádra PicoBlaze, které patří mezi jednodušší jádra, jsou ještě hojně používána jádra MicroBlaze a Nios. První dva jsou od firmy Xilinx a třetí je od firmy Altera.

Výhody použití:

- Možnost konfigurace procesoru a jeho periferií podle konkrétních požadavků.
- Možnost paralelního provozu několika procesorů implementovaných do jednoho obvodu FPGA
- Nižší riziko nedostupnosti součástek při volbě dostatečně „mladého“ FPGA obvodu. Dostupnost procesorového jádra jako součástky není nutno řešit díky jejímu softwarovému provedení.
- Mnohem nižší počet součástek na plošném spoji. Výhoda ve spolehlivosti, jednoduchosti a ceně je zřejmá.
- Mimo velmi kvalitních komerčních procesorů a jejich vývojových prostředků je možnost využití i jejich volně dostupných variant.
- V dnešní době již poměrně rozšířená uživatelská základna pro používání procesorů v obvodech FPGA.
- Při upgrade výrobku není nutné zasahovat do hardware konstrukce. Stačí upgrade konfiguračního souboru.

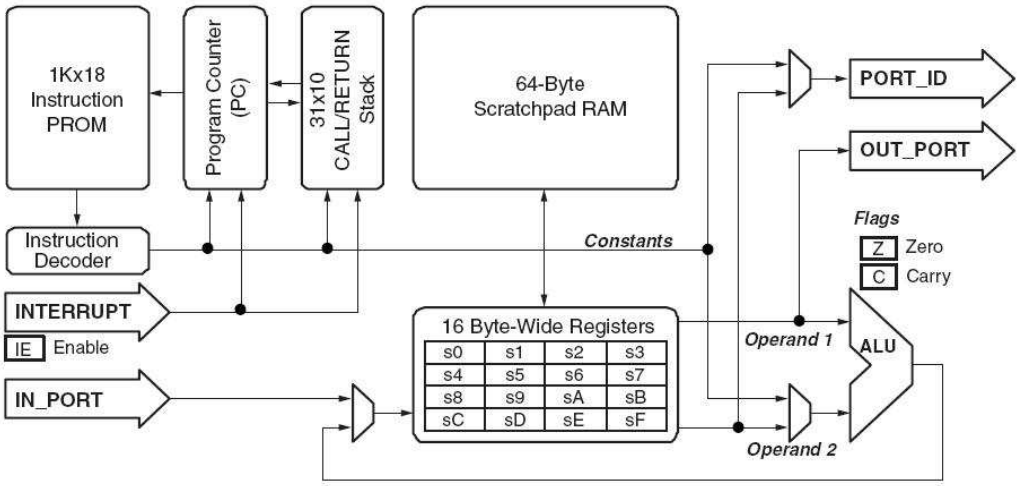
4.1 Procesorové jádro PicoBlaze

Procesor PicoBlaze vyvinul pracovník firmy Xilinx Ken Chapman. PicoBlaze je jednoduchý osmibitový RISC procesor s Harvardskou architekturou. Tedy s oddělenou pamětí programu a dat. Tento procesor je ze skupiny volně dostupných procesorů. Všechny potřebné vývojové prostředky jsou tedy dostupné zdarma na internetu. PicoBlaze je procesor navržený pro čipy Spartan-3, Virtex II, Virtex II Pro a Virtex-4[7]. Největší předností jádra PicoBlaze je jeho velikost. V obvodu Spartan-3 totiž zabírá jen 96 řezů, což je cca 5% z modelu XC3S200 a méně než 0,3% z modelu XC3S5000[7]. Při použití nejmenšího modelu z rodiny Spartan-3 XC3S50 se teoreticky vejde až osm jader PicoBlaze[5]. Jeho rychlost je v závislosti na použitém obvodu od

43 do 100 MIPS s pracovní frekvencí od 88 do 200 MHz[7]. Procesoru PicoBlaze je několik verzí. Zde se bude mluvit o verzi KCPSM3[6]. Zdrojová jednotka pro tento procesor se jmenuje kcpsm3.vhd. V označení KCPSM písmena PSM značí Programmable State Machina, ale pro písmena KC zdroje uvádějí dva významy. Buď se jedná o (K)constant Coded nebo po svém tvůrci Ken Chapman.

4.1.1 Architektura procesoru

Na následujícím obrázku je blokové schéma procesoru PicoBlaze ve verzi pro obvody typu Spartan-3 a Virtex-II.



Obr. 4.1 Blokové schéma procesoru PicoBlaze

Základní parametry:

- 1Kx18 Instrukční paměť. Tedy paměť programu pro 1024 18-ti bitových instrukcí
- Programový čítač
- Zásobník pro ukládání návratových adres pro až 31 vnořených podprogramů
- Uživatelská paměť o velikosti 64 bajtů
- 16 osmibitových registrů (s0 až sF)
- Jeden zdroj přerušení
- Aritmeticko – logickou jednotku ALU s příznaky nuly a přenosu
- 256 osmibitových vstupů
- 256 osmibitových výstupů
- Reset

Pokud by nestačil pouze jeden zdroj přerušení, je možné si další přerušení realizovat v obvodu FPGA.

4.1.2 Instrukční sada

Procesor PicoBlaze je typu RISC tzn., že má redukovanou instrukční sadu. Instrukční sada procesoru PicoBlaze má tedy celkem 57 instrukcí. Na následujícím obrázku je výčet instrukční sady procesoru PicoBlaze.

'X' and 'Y' refer to the definition of the storage registers 's' in the range 0 to F.
'kk' represents a constant value in the range 00 to FF.
'aaa' represents an address in the range 000 to 3FF.
'pp' represents a port address in the range 00 to FF.
'ss' represents an internal storage address in the range 00 to 3F.

Program Control Group

```
JUMP aaa
JUMP Z,aaa
JUMP NZ,aaa
JUMP C,aaa
JUMP NC,aaa

CALL aaa
CALL Z,aaa
CALL NZ,aaa
CALL C,aaa
CALL NC,aaa

RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC
```

Note that call and return supports up to a stack depth of 31.

Arithmetic Group

```
ADD sX,kk
ADDCY sX,kk
SUB sX,kk
SUBCY sX,kk
COMPARE sX,kk

ADD sX,sY
ADDCY sX,sY
SUB sX,sY
SUBCY sX,sY
COMPARE sX,sY
```

Interrupt Group

```
RETURNI ENABLE
RETURNI DISABLE

ENABLE INTERRUPT
DISABLE INTERRUPT
```

Logical Group

```
LOAD sX,kk
AND sX,kk
OR sX,kk
XOR sX,kk
TEST sX,kk

LOAD sX,sY
AND sX,sY
OR sX,sY
XOR sX,sY
TEST sX,sY
```

Storage Group

```
STORE sX,ss
STORE sX,(sY)
FETCH sX,ss
FETCH sX,(sY)
```

Shift and Rotate Group

```
SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX

SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX
```

Input/Output Group

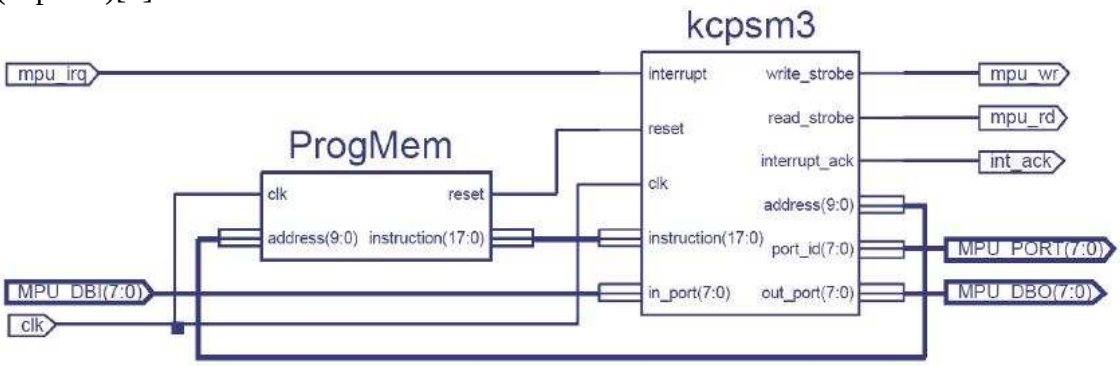
```
INPUT sX,pp
INPUT sX,(sY)
OUTPUT sX,pp
OUTPUT sX,(sY)
```

Obr. 4.2 Instrukční sada procesoru PicoBlaze [9]

Kompletní význam a způsob použití instrukcí lze nalézt v manuálu KCPSM3 [8].

4.1.3 Implementace procesoru

Při vytváření procesorového jádra jsou nutné dvě složky. Prvně je nutné sestavit program pro procesor. Z něho se vytvoří kompilací komponenta paměti programu. Druhou částí důležitou pro vytvoření procesorového jádra je komponenta procesoru (kcpsm3)[5].



Obr. 4.3 Vytvoření komponenty procesoru PicoBlaze[5]

Po vytvoření procesorového jádra se v ISE Webpack vytvoří model s propojením procesoru se všemi jeho periferiemi. Nejčastěji se takto děje buď intuitivně grafickým schématem a nebo strukturálním stylem, pokud je rozsah popisu příliš komplexní. Takto navržený popis se poté může implementovat do obvodu FPGA.

Pro práci s procesorem PicoBlaze je mimo ISE Webpack nutný ještě nástroj pro kompilaci programu (vytvoření paměti programu procesoru). Těchto nástrojů je vytvořeno také několik. Mimo původních programů vytvořených firmou Xilinx je velmi populární pBlazIDE od firmy Mediatronix. Tento program není zatížený autorským právem a je tedy možné jej získat zdarma z webových stránek firmy Mediatronix[10].

5. Periferie pro Picoblaze

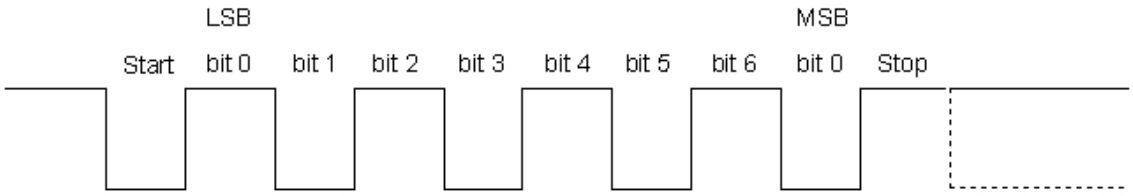
Zde budou uvedeny periferie, které budou pro tento projekt použity. Jejich obecný popis i konkrétní použití v tomto projektu. Všechny periferie budou psány stylem „zdola nahoru“ (viz. str. 8).

5.1 Sériová komunikace po RS-232

Sériová komunikace, někdy se nazývá i sériovou linkou, představuje komunikaci mezi dvěma zařízeními (např. mezi dvěma počítači) po jedné lince, kde komunikace probíhá posíláním dat bit za bitem. Při sériové komunikaci je možné využít buď synchronního nebo asynchronního přenosu dat (USART - Universal Synchronous/Asynchronous Receiver and Transmitter). U synchronního systému je nutné spolu s datovým vodičem vést i vodič synchronizační, který informuje přijímací stranu o okamžiku vysílání dat. U asynchronního systému se tato synchronizace provádí vysíláním předem definovaných dat, kterým se cílová stanice synchronizuje.

5.1.1 UART

UART představuje asynchronní sériovou komunikaci mezi dvěma stanicemi. Datový vodič, kterým se přenášejí jednotlivé bity dat, je v klidovém stavu na hodnotě logické jedničky. Přijímací strana hlídá příchod logické nuly, která představuje synchronizační bit, podle kterého se přijímací strana synchronizuje. Za tímto synchronizačním bitem následují data bit po bitu obvykle v bloku 8 bitů. Toto množství lze však v případě potřeby volit i jinak (např. 7 nebo 9 bitů). Přenos dat se provádí od nejnižšího LSB bitu až po nejvyšší MSB bit. Pokud je pro zabezpečení přenosu použita parita, je tento paritní bit zařazen za posledním datovým bitem. Nakonec následuje jeden nebo dva tzv. stop bity, které ukončují datový rámec za nimž se datová linka uvádí opět do klidového stavu.

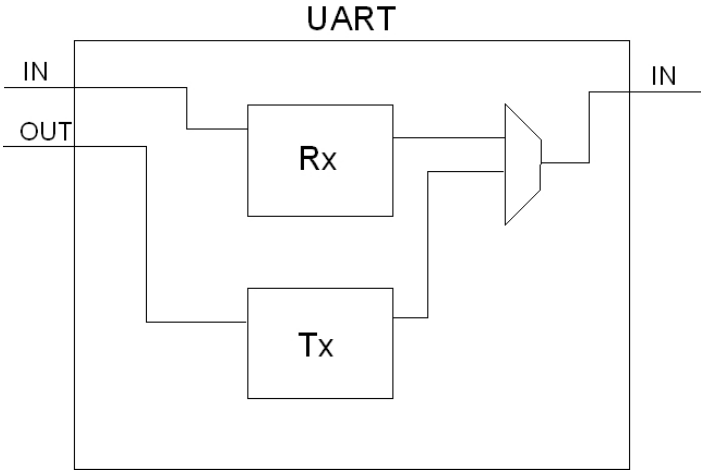


Obr. 5.1 Schéma datového rámce pro UART

Na obrázku výše je uvedeno schématické znázornění datového rámce, kterým jsou data po sériové lince přenášena. Tento rámec nemusí striktně vypadat vždy takto, ale je nutné zajistit, aby se zvoleným schématem pracovala jak vysílací, tak i přijímací stanice. Dále je možné si zvolit i různé modulační rychlosti (např. 4800Bd, 9600Bd aj.), v našem případě, kdy se 1Bd rovná 1 bit/s, můžeme prohlásit, že modulační rychlost v jednotkách Bd (Baud) (počet změn stavu přenosové linky za 1 sekundu) se rovná přenosové rychlosti v jednotkách bit/s. Avšak i v tomto případě je nutné, aby s námi zvolenou přenosovou rychlostí pracovala jak vysílací, tak i přijímací stanice. Na tomto údaji totiž závisí vyhodnocování přijímaných bitů. Pokud by přijímací strana očekávala data o rychlosti 4800Bd a data by přicházela rychlostí 9600Bd, tak by přijímací strana stále očekávala další příchozí data dávno po tom, co již skutečný přenos dávno skončil.

5.1.2 VHDL popis

Celý UART systém je rozdělen na dva dílčí bloky. Jeden představuje přijímací blok Rx a druhý naopak vysílací blok Tx. Tyto dva bloky jsou pak doplněny o VHDL kód, který zajišťuje jejich řízení a propojení s I/O piny celého bloku UART.



Obr. 5.2 Strukturní schéma UART bloku

Nejdříve je nutné vytvořit z hodinového signálu o frekvenci 50 MHz hodinový signál, který odpovídá zvolené přenosové rychlosti. V tomto případě byla zvolena přenosová rychlost 9600kHz.

```
--text 2.
clk_gen_proces: PROCESS(clk) BEGIN
      --proces pro generaci hodinoveho signalu 9600kHz
      IF clk'EVENT AND clk= '1' THEN
            IF clk_gen = per_bin(20 DOWNTO 1) THEN
                  clk_gen(20 downto 2) <= (OTHERS => '0');
                  clk_gen(1) <= '1';
                  clk_EN <= '1';
            ELSE
                  clk_EN <= '0';
                  clk_gen <= clk_gen + 1;
            END IF;
      END IF;
END PROCESS clk_gen_proces;
```

Konstanta `per_bin` je číslo, kterým je nutné základní frekvenci 50 MHz vydělit, aby bylo možné vytvořit frekvenci 9600kHz. V podmínce IF se s každým hodinovým taktem inkrementuje proměnná `clk_gen` a ve chvíli, kdy dosáhne hodnoty rovné konstantě `per_bin`, se generuje v proměnné `clk_EN` logická jednička, která představuje hodinový signál o frekvenci 9600kHz. Tento systém generace hodinového signálu je shodný pro oba bloky Rx i Tx.

Vlastní jádro programu je pro blok Rx, tak i pro blok Tx v zásadě stejné. Je tvořeno stavovým automatem, který po signalizaci začátku příjmu (resp. vysílání) dat, s každým hodinovým taktem převádí sériová data na paralelní (resp. paralelní na sériová).

```
--text 3.
state_machine: PROCESS (current_state) BEGIN
    CASE current_state IS
      WHEN startb => next_state <= bit0;
      WHEN bit0   => next_state <= bit1;
      WHEN bit1   => next_state <= bit2;
      WHEN bit2   => next_state <= bit3;
      WHEN bit3   => next_state <= bit4;
      WHEN bit4   => next_state <= bit5;
      WHEN bit5   => next_state <= bit6;
      WHEN bit6   => next_state <= bit7;
      WHEN bit7   => next_state <= stopb;
      WHEN stopb  => next_state <= idle;
      WHEN OTHERS => next_state <= idle;
    END CASE;
 END PROCESS state_machine;
```

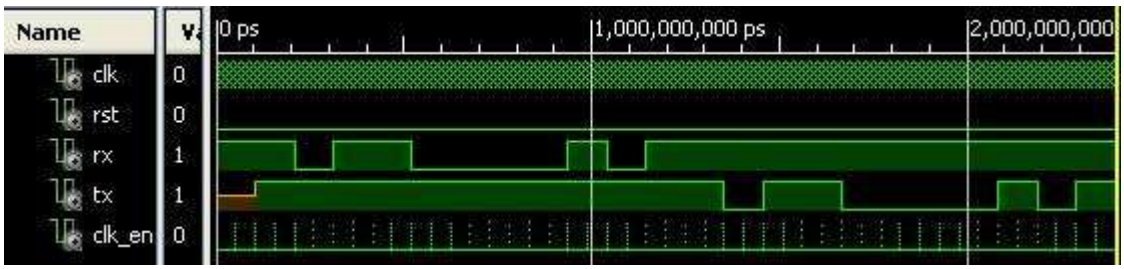
Signály `current_state` a `next_state` jsou speciální typy signálů, které nabývají uživatelem předem definované stavy. V tomto případě stavy odpovídající zvolenému schématu datového rámce pro UART.

U bloku Rx se v každém stavu stavového automatu načte aktuální hodnota na vstupním pinu Rx a uloží se do vnitřního bufferu, který se po ukončení příjmu odesílá ke zpracování procesoru Picoblaze.

U bloku Tx se v každém stavu stavového automatu odešle příslušná hodnota z vnitřního bufferu na výstupní pin Tx.

5.1.3 Simulace a implementace

Na následujícím obrázku je vidět, že data, která se přijmou na vstupu Rx jsou vzata procesorem Picoblaze a následně odeslána na výstup Tx. Je zde také zobrazen hodinový signál `clk_en` představující přenosovou rychlost 9600KHz (resp. 9600 bit/s).



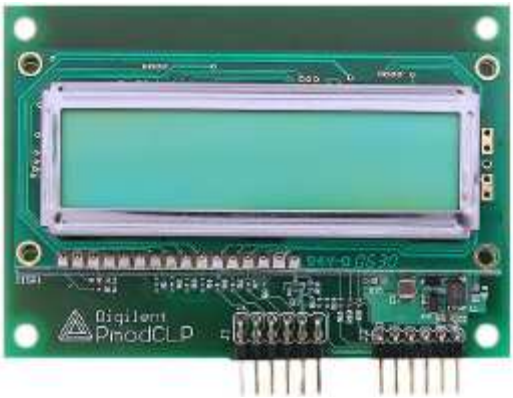
Obr. 5.3 Simulace přenosu dat po sériové lince

Funkčnost navrhnutého modelu dokládá i printscreen programu Terminal, kterým se sestavuje komunikace mezi počítačem a vývojovým kitem.

V tomto návrhu procesor Picoblaze přijatá data nemění a jen je obratem odesílá zpět na sériovou linku. Případný požadavek na jiný typ zpracování od procesoru Picoblaze (např. převod z malých písmen na velká) je velmi snadný a je otázkou pár dalších řádků ve zdrojovém kódu.

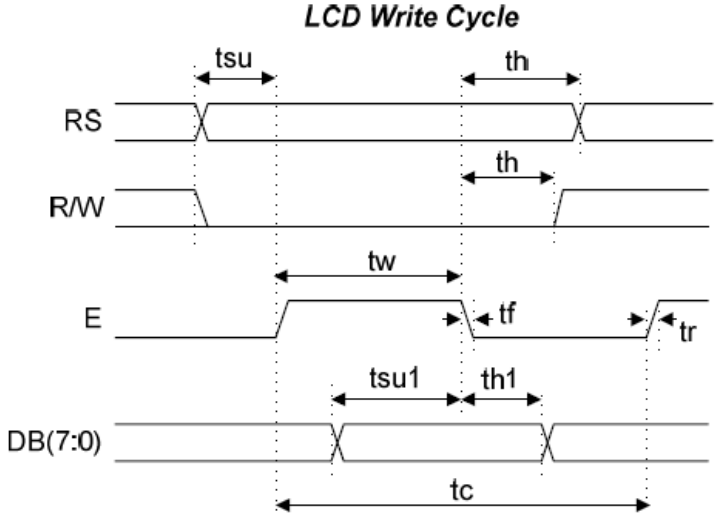
5.2 LCD displej

V tomto projektu je použit dvouřídkový, jednobarevný LCD display, který bude zobrazovat data ze sériové linky.



Obr. 5.4 Obrázek LCD display[14]

LCD display má tři řídící a jeden obousměrný osmibitový datový port. LCD pracuje s ASCII tabulkou, kterou lze v případě potřeby upravit.



Obr. 5.5 Časové schéma jednoho zápisového cyklu pro komunikaci s LCD[14]

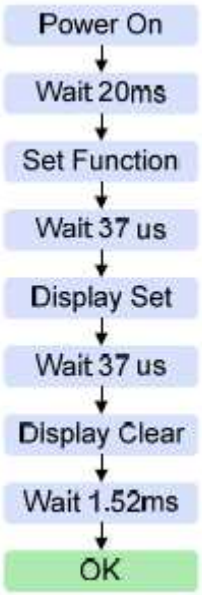
Signálem RS se LCD signalizuje, zda má očekávat na datovém portu příkaz nebo data. Signálem R/W se oznamuje LCD zdali se budou dat číst nebo se budovou data na LCD zapisovat. V tomto projektu se pracuje pouze se zápisem na LCD. Posledním řídícím signálem je signál E (enable), kterým se povoluje spuštění operace na LCD. Pro správnou funkci je nutné, aby RS a R/W byli nastaveny alespoň 40ns před nástupnou hranou enable signálu a alespoň 10ns po sestupné hraně enable signálu. Dále je nutné, aby vysoká úroveň enable signálu pro správnou funkci LCD trvala alespoň 220 ns[14].

Signál DB[7:0] je osmi bitový datový port jenž musí být nastaven alespoň 40ns před sestupnou hranou enable signálu. Procesor Picoblaze pracuje s osmi bitovou datovou sběrnicí a znaky ASCII tabulky jsou též osmi bytová čísla. Jedna komunikační sekvence tedy odpovídá jednomu poslanému znaku. Dle [14] každá komunikační

sekvence (resp. vysoká úroveň enable signálu) musí být oddělena mezerou alespoň 500 ns. Při práci na projektu však bylo experimentálně zjištěno, že tato mezera musí být minimálně 16μs.

Signál DB[7:0] se také používá pro nastavování LCD v tzv. příkazovém modu (command mode). V příkazovém modu lze LCD nastavovat, tak aby zobrazování znaků splňovalo požadované parametry. Zapisování příkazů v command modu se docílí tím, že na port RS LCD displeje se při komunikační sekvenci přivede logická nula. Jednak se v tomto modu nastavuje zapnutí a vypnutí displeje, mód zobrazování kurzoru, posunování textu po displeji a posunování kurzoru po řádku při zobrazování více znaků.

Při každém restartu je nutné nejdříve provést startovní sekvenci, kterou se LCD uvádí do provozního režimu.



Obr. 5.6 Startovní sekvence LCD[14]

Startovní sekvence obsahuje tři komunikační sekvence, které jsou odděleny předepsanými časovými mezerami. Sekvence Set Function nastavuje displej do módu 8bit datového přenosu, počet zobrazených řádků a font písma použitý pro zobrazená data. Display Set nastavuje mód posunu kurzoru doleva nebo doprava a posun posun celého zobrazeného textu doleva či doprava. Poslední sekvence Display Clar vyčistí display od případných pozůstatků od minulé komunikace a přesune kurzor na začátek LCD[14].

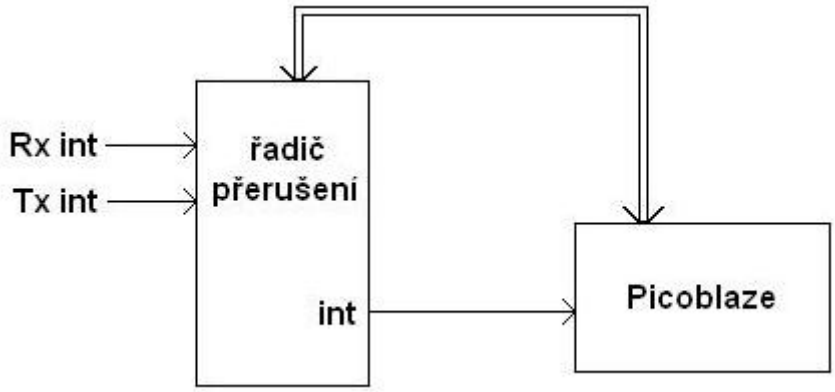
Celá programová část pro ovládání periferie LCD je soustředěna pouze v programu pro procesor. Ve VHDL části se provádí pouze fyzické propojení obvodu FPGA s LCD. Při použití LCD v tomto projektu se pracovalo s LCD připojeným na rozšiřující konektor A_2 Spartan-3 Starter Kitu. Pokud bude připojen na jiný konektor, je třeba změnit definici připojení signálů na výstupy obvodu FPGA v UCF souboru.

5.3 Řadič přerušení

Procesor Picoblaze má jeden zdroj externího přerušení (viz. kap. 4.1). Pokud tedy potřebujeme vytvořit návrh, ve kterém bude figurovat více druhů externích přerušení, je nutné vytvořit řadič přerušení, který se postará o jejich přepínání na interrupt vstup procesoru Picoblaze.

5.3.1 VHDL popis

V základní konfiguraci systému existují dva zdroje přerušení. První je signál, který oznamuje nově příchozí data v Rx bloku periferie UART, která čekají na zpracování a druhým signálem je potvrzení dokončení vysílací sekvence vysílací částí periferie UART. Ve VHDL bloku řadiče pak tyto dva signály generují signál přerušení běhu programu procesoru (viz obr. 5.8). Přerušení je citlivé na úroveň (level sensitive). To znamená, že na *interrupt* vstupu procesoru Picoblaze je vysoká úroveň tak dlouho, dokud procesor toto přerušení neobslouží.



Obr. 5.7 Schéma řadiče přerušení

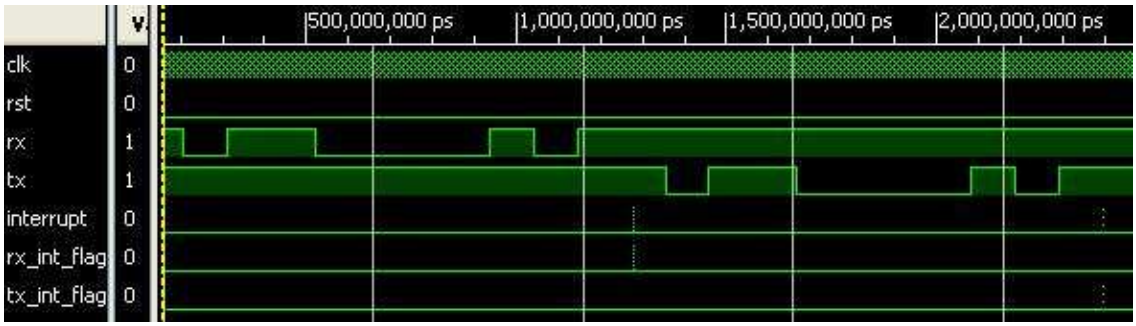
V řadiči přerušení se také zapisuje do *status registru* o jaké přerušení se jednalo. Procesor zaznamená na vstupním *interrupt* portu (viz. kap. 4.1) signál o příchodu přerušení. Dokončí právě probíhající instrukci a začne s obsluhou přerušení. V rámci této obsluhy procesor zaznamená na svém vstupu informaci o původu přerušení a na základě této informace spustí příslušný podprogram.

```
--text 4.
Int_mux: PROCESS(clk) BEGIN
      IF clk'EVENT AND clk = '1' THEN
      ------------------------------------------------------------
         IF rx_int_flag = '1' OR tx_int_flag = '1' THEN

                        IF tx_int_flag = '0' THEN
                              Status(0) <= '1';
                        END IF;

                        IF rx_int_flag = '0' THEN
                              Status(1) <= '1';
                        END IF;

                        Pico_int  <= '1';
         END IF;

         IF Address = Int_port_ID AND read_strobe = '1' THEN
                   Pico_int  <= '0';
                   Status(0) <= '0';
                   Status(1) <= '0';
         END IF;
     ------------------------------------------------------------
    END IF;
END PROCESS Int_mux;
```

Výše uvedený zdrojový kód je hlavním tělem bloku řadiče přerušení. Z každým hodinovým taktem se kontroluje stav signalizace externího přerušení. Pokud je některý z nich aktivní, generuje se na vstupní port procesoru logická 1 (Pico_int). Zároveň se do Status registru uloží původ přerušení.

5.3.2 Simulace a implementace

Na obrázku 5.9 lze vidět, že krátce se po úspěšném přijetí znaku z periferie UART spustí interrupt rutina procesoru picoblaze a po stejnou událost lze pozorovat i po úspěšném odvysílání dat zpět na UART. Signály rx_int_flag a tx_int_flag ukazují konkrétní zdroj přerušení dle umístění jejich pulsů.



Obr. 5.8 Simulace řadiče přerušení

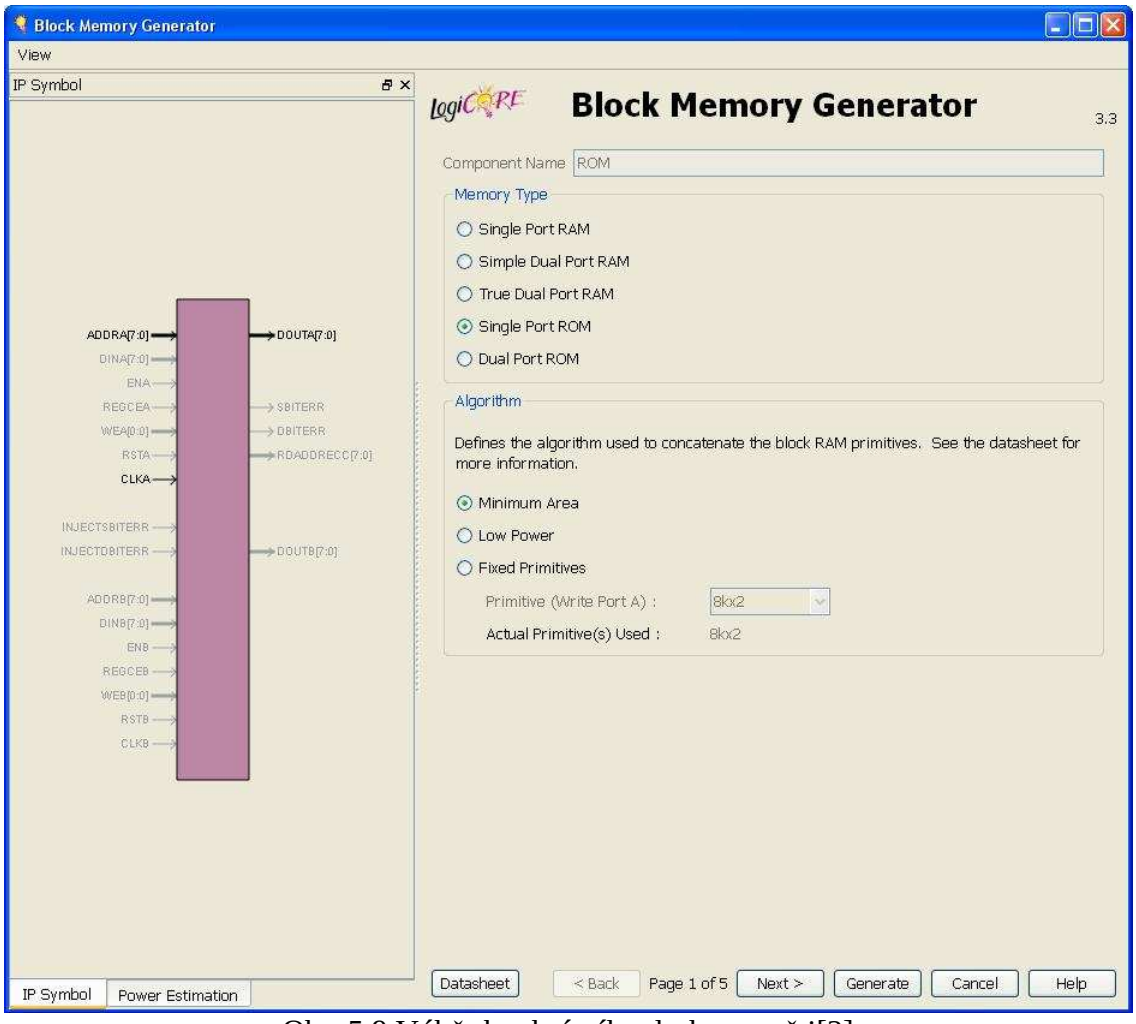
Jak již bylo zmíněno výše, je tato komponenta návrhu v podstatě nečinná a proto při její implementaci není na vývojovém kytu vidět žádný její vliv. Při vytváření VHDL popisu této periferie byly do obsluhy přerušení vkládány umělé příkazy, které dovolily při simulaci ověřit funkčnost návrhu. Tyto však byly pro finální verze zdrojových kódů v IP generátoru vypuštěny.

5.4 ROM paměť

Program ISE WebPack obsahuje velké množství IP jader od komunikačních, přes matematické až po bloky zpracovávající video a obraz. Tyto IP jádra představují hotové bloky určené ke vložení do projektu. Pro tento projekt byl pro názornost zvolen blok paměti ROM. Z této paměti je možné procesorem číst data a ty odesílat pomocí rozhraní UART do PC. Data, která jsou v paměti ROM uložena jsou při implementaci načtena ze souborů typu COE, ve kterých jsou všechny znaky uloženy ve formě ASCII kódů. Data jsou v paměti ROM uložena v adresním registru a každy znak má přiřazenou svoji vlastní hexadecimální adresu. Uživatel odešle po seriové lince příkaz značící čtení z paměti ROM z hexadecimální adresy uvedené v příkazu. Procesor Picoblaze se poté postará o doručení příkazu na vstup ROM a v následujícím taktu o vyzvednutí dat z odpovídající adresy z výstupu ROM, které se odešlou zpět uživateli.

5.4.1 VHDL popis

Programátor musí udělat dva kroky. Prvním je vygenerovat IP jádro paměti ROM a připojit k němu soubor s daty pro uložení (viz. obr. 5.10).



Obr. 5.9 Výběr konkrétního druhu paměti[2]

Druhým krokem je připojení komponenty ROM do navrhovaného systému. To se provádí stejným způsobem jako připojení jakéhokoliv jiného VHDL bloku v tomto projektu.

Nejprve je nutné zahrnout do vrcholového VHDL souboru komponentu ROM se všemi jejími vstupními a výstupními porty.

```
--text 5.
COMPONENT rom_top
    GENERIC(Rom_port_ID: STD_LOGIC_VECTOR(7 DOWNTO 0):= x"01"
            );
    Port (clka:   IN std_logic;
          addra:  IN std_logic_VECTOR(7 downto 0);
          douta:  OUT std_logic_VECTOR(7 downto 0));
END COMPONENT;
```

Klíčovým slovem **GENERIC** se komponentě přiřazují konstanty, které mají být viditelné i pro ostatní komponenty. V tomto návrhu se tyto konstanty používají pro multiplexování přístupů všech periferií k vstupnímu a výstupnímu portu procesoru Picoblaze. Klíčovým slovem **Port** se deklarují všechny vstupní a výstupní porty dané komponenty. Druhá část připojení komponenty do návrhu je její propojení s ostatními komponentami [3](viz text 6.).

```
--text 6.
inst_rom_top : rom_top
      GENERIC MAP(
                  Rom_port_ID     => ROM_port_ID
                  );
      PORT MAP(
                  clka            => clk,
                  addra           => Rom_addr,
                  douta           => Rom_data
                  );
```

Klíčová slova **GENERIC MAP** uvozují programovou část, kde se propojují konstanty jednotlivých komponent s konstantami deklarovanými jako vnitřní konstanty ve vrcholovém souboru. Klíčová slova port map analogicky uvozují propojení vstupních a výstupních portů dané komponenty se vstupními a výstupními porty ostatních komponent či s vnitřními signály vrcholového souboru.

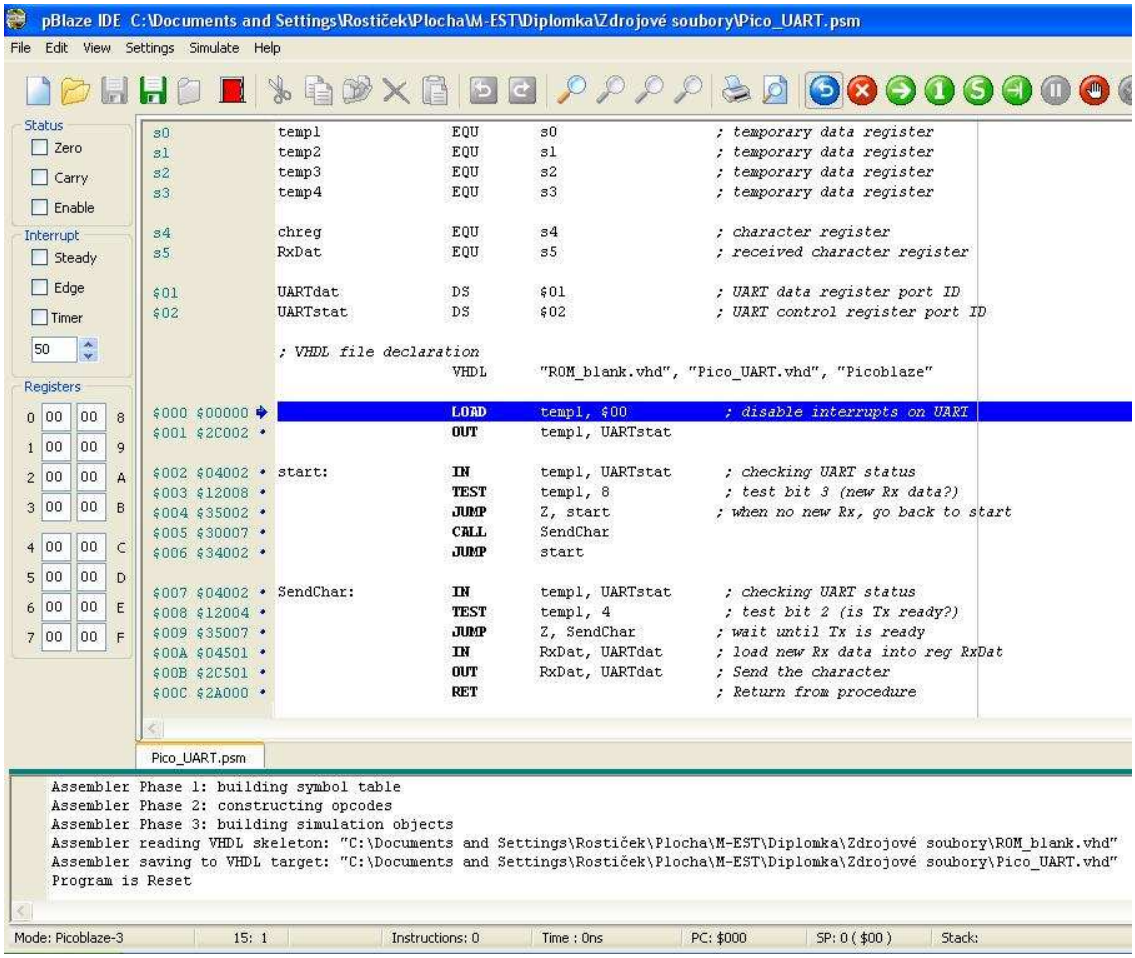
5.4.2 Implementace

V kap. 5.4 bylo popsáno, jak je v tomto projektu periferie ROM provozována. Je tedy zcela jasné, že periferie ROM v tomto projektu ke svému chodu bezpodmínečně potřebuje přítomnost periferie UART.

Po implementaci celého návrhu do obvodu FPGA se uživatel spojí přes program Terminal (nebo některým jiným programem pro komunikaci přes sériovou linku) s vývojovým kitem. Posláním znaku odpovídající klávese Esc, se procesor Picoblaze přesune do programové části čtení z periferie ROM. Uživatel poté odešle znak po znaku příkazovou sekvenci pro čtení z ROM na adrese uvedené v těle příkazu. Pro přechod zpět do režimu sériové komunikace je nutné stisknout opět klávesu Esc.

6. Program pro procesor

Program pro procesor byl psán a simulován v programu pBlazIDE.



Obr. 6.1 Ilustrační obrázek programu pBlazIDE

Pro přehlednější práci s registry s0 až sF jim bude pomocí klíčového slova `EQU` dočasně přiděleno jméno, které uživatelsky zjednoduší orientaci při jejich používání. Stejně tak je všem konstantám, které budou použity, přiřazeno pomocí klíčového slova `DS` jméno, které koresponduje s jeho použitím.

6.1 Programová část pro UART

Blok UART obsahuje VHDL kód, který se stará o signalizaci jeho vnitřního stavu procesoru. Konkrétně to tedy znamená, že UART blok nastavuje status registr podle toho, zda chce oznámit procesoru příchod nově přijatých dat ke zpracování nebo zda je vysílací jednotka Tx připravena k novému vysílaní dat.

```
--text 7.
start:              IN        temp1, UARTstat
                    TEST      temp1, 8
                    JUMP      Z, start
                    CALL      SendChar
                    JUMP      start
```

Hlavní smyčka je tvořena několika příkazy, které se díky příkazu **JUMP**, kterým program skočí zpět na začátek této rutiny, neustále opakují dokud nedojde k očekávané události. Příkazem **IN** se provede načtení stavu výše zmíněného status registru. Následuje příkaz **TEST**, který provede kontrolu nastavení třetího bitu. V případě, že je na této pozici logická nula, je nastaven příznak *zero*. Třetí příkaz je podmíněným skokem na začátek rutiny řízeným právě nastaveným příznakem *zero*. Tedy tato sekvence třech příkazů se bude opakovat tak dlouho, dokud na vstupu procesoru nebude nastavena binární hodnota čísla 8, která odpovídá signalizaci nově příchozích dat ke zpracování. Pokud tedy procesor obdrží pokyn ke zpracování nově přijatých dat, výše zmíněný podmíněný skok se neprovede a program pokračuje příkazem **CALL**, který volá programovou část starající se o zpracování dat na jeho vstupu. Zároveň se do zásobníku uloží návratová adresa pro návrat s volané programové části.

```
--text 6.
SendChar:           IN        temp1, UARTstat
                    TEST      temp1, 4
                    JUMP      Z, SendChar
                    IN        RxDat, UARTdat
                    OUT       RxDat, UARTdat
                    RET
```

Opět se příkazem **IN** načte stav status registru a příkazem **TEST** provede jeho kontrola. Tentokrát se kontroluje nastavení druhého bitu, který na hodnotě logické jedničky oznamuje, že poslední vysílání dat je již dokončené a blok Tx je tedy připraven pro opětovné vysílání. Jakmile příchozí informace o připravenosti vysílací jednotky dovolí procesoru pokračovat v programu, načte procesor pomocí příkazu IN z datového vstupu osmibitový blok dat, který přijal po sériové lince. V dalším kroku tyto data procesor odešle na svůj datový výstup pomocí příkazu **OUT**. Zároveň na výstupních pinech write_strobe a port_id signalizuje bloku UART, že se jedná o data k odeslání. Poslední příkaz **RET** volá z paměti zásobníku návratovou adresu a vrací se na pozici v programu odpovídající této hodnotě. Tím se celý cyklus uzavírá a procesor je připraven na opětovný příjem dat.

6.2 Programová část pro LCD

V bodě 5.2 bylo zmíněno, že pro správnou funkci je nutné dodržet určité časové rozestupy mezi jednotlivými komunikačními okny. Jelikož Picoblaze neustále provádí zadané operace a nelze mu nařídit, aby přestal pracovat, je nutné tyto časové rozestupy vytvořit. Lze je vytvořit velmi snadno voláním programových části, ve kterých se provádí určitý počet operací, které procesor zaměstnají právě na potřebnou dobu, kterou potřebuje LCD pro svoji práci.

```
--text 7.
delay_16us:             LOAD      temp1, 200
                        LOAD      temp2, 2
wait_16us:              SUB       temp1, 1
                        JUMP      NZ, wait_16us
                        SUB       temp2, 1
                        JUMP      NZ, wait_16us
                        RET
```

Příkazem LOAD se do registru `temp1` uloží konstanta 200 a do registru `temp2` uloží konstanta 2 . Příkaz **`SUB`** od registru `temp1` odečte jedničku. Pokud se tímto příkazem sníží hodnota registru `temp1` na nulu, je nastaven příznak zero. Následuje podmínění skok, který kontroluje příznak non zero, tedy negovaný zero příznak. Skočí na návěští wait_1us pokud je NZ nastaven na hodnotu logické jedničky a NZ je nastaven na hodnotu logické jedničky dokud se hodnota registru temp1 nesníží na nulu. Jelikož hodinový takt Spartanu je 50MHz, trvá jeden příkaz 40 ns. Provedení operace SUB 200-krát za sebou tedy procesoru trvá cca 8 μs. Druhý příkaz **`SUB`** spjatý s registrem `temp2` zajišťuje, že se předchozí sekvence odečítání bude opakovat tolikrát kolikrát je hodnota registru `temp2`. Tímto jsme dosáhli časové prodlevy alespoň 16 μs a stejným principem lze dosáhnout jakéhokoliv časového zpoždění.

```
--text 8.
delay_40us:         LOAD        s1, 28
wait_40us:          CALL        delay_1us
                    SUB         s1, 01
                    JUMP        NZ, wait_40us
                    RET
```

Při použití periferie LCD se po restartu a dokončení startovní sekvence automaticky vypíše sada znaků

„Dobrý den!
Poslední znak: „.

Tuto funkci v programu pro procesor zajišťuje programová rutina LCD_message zde uvedená ve zkrácené verzi (viz. text 9).

```
--text 9.
LCD_message:            LOAD    LCDbuff, $44        ;char "D"
                        CALL    CharToLCD
                        CALL    delay_16us
                        LOAD    LCDbuff, $6F        ; char "o"
                        CALL    CharToLCD
                        CALL    delay_16us
                        LOAD    LCDbuff, $62        ; char "b"
                        CALL    CharToLCD
```

```
        CALL    delay_16us
        .
        .
        .
        LOAD    LCDbuff, $7A       ; char "z"
        CALL    CharToLCD
        CALL    delay_16us
        LOAD    LCDbuff, $6E       ; char "n"
        CALL    CharToLCD
        CALL    delay_16us
        LOAD    LCDbuff, $61       ; char "a"
        CALL    CharToLCD
        CALL    delay_16us
        LOAD    LCDbuff, $6B       ; char "k"
        CALL    CharToLCD
        CALL    delay_16us
        LOAD    LCDbuff, $3A       ; char ":"
        CALL    CharToLCD
        CALL    delay_16us
        RET
```

Poté se program dostane do hlavní smyčky (viz text 4.),ve které se čeká na příchod nového znaku z periferie UART. Po příchodu nového znaku z periferie UART se provede programová rutina pro opětovné odeslání znaku na UART (viz text 5.) a za ní následuje rutina pro odeslání znaku na LCD.

```
--text 10.
CharToLCD:      OUT     LCDbuff, LCDdat   ;send char to LCD driver
                LOAD    LCDbuff, $01      ;set RS and R/W control
                OUT     LCDbuff, LCDctrl
                CALL    LCD_pulse_E       ;send data to LCD
                RET
```

Nejprve se odešlou data na datový vstup LCD a vzápětí se nastavý kontrolní signály RS a R/W. Komunikace s LCD se zahájí voláním rutiny `LCD_pulse_E`.

```
--text 11.
LCD_pulse_E:    XOR     LCDbuff, LCD_E    ; set enable
                OUT     LCDbuff, LCDctrl
                CALL    delay_220ns       ; delay for enable pulse
                XOR     LCDbuff, LCD_E    ; unset enable
                OUT     LCDbuff, LCDctrl  ; LCD working launch
                                          ;(falling edge)
                RET
```

Aby byl na LCD zobrazen vždy jen poslední přijatý znak, posune se, po každém datovém přenosu na LCD, kurzor zpět o jedno místo. Viz následující zdrojový text.

```
--text 12.
cursor:         LOAD    LCDbuff, $10            ; set LCD mod
                OUT     LCDbuff, LCDdat
                LOAD    LCDbuff, $00            ; set RS, R/W
                OUT     LCDbuff, LCDctrl
                CALL    LCD_pulse_E
                RET
```

Funkce je naprosto totožná jako v případě rutiny `CharToLCD:`. Jediný rozdíl je v tom, že RS signál je nastaven tak, aby signalizoval příchod command dat.

6.3 Programová část pro řadič přerušení

Celá ovládací část periferie je obsažena ve VHDL bloku přerušení. V paměti programu se pouze zjišťuje z jakého zdroje bylo přerušení vyvoláno a podle toho spouští příslušná obsluha.

```
--text 12.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Interrupt routine
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Rx_interrupt:           JUMP      end_interrupt

Tx_interrupt:           JUMP      end_interrupt

Interrupt:              IN        temp1, Int
                        COMP      temp1, 1
                        JUMP      Z, Rx_interrupt
                        COMP      temp1, 2
                        JUMP      Z, Tx_interrupt
end_interrupt:          RETI      ENABLE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Interrupt jump instruction
                        ORG       $3FF                        ;
interrupt
                        JUMP    Interrupt
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

V programu pro procesor je po příchodu *interrupt* signálu automaticky proveden skok na poslední pozici v paměti programu, kde je uložena instrukce skoku na obslužný program přerušení. V této obsluze se nejprve načte příkazem **`IN`** *Status registr* řadiče přerušení a poté se příkazem **`COMP`** zjišťuje z jakého zdroje bylo přerušené vyvoláno. Na základě příkazu **`COMP`** se nastavuje příznak *zero*, podle kterého se řídí podmíněné skoky do příslušných podprogramů obsluhy přerušení. Rutina končí příkazem **`RETI`**, kterým se program vrací zpět na místo v programu, kde došlo k přerušení. Parametrem `ENABLE` se povoluje spuštění dalšího přerušení.

6.4 Programová část pro ROM

V kap. 5.4.2 bylo vysvětleno jak moc je paměť ROM v této práci propojena s komponentou UART. Tento fakt se nejvíce podepsal právě na provázanosti programových částí obou komponent v programu pro procesor Picoblaze. Zásadní částí je kód díky nemuž se provádí přepínání mezi módy UART komunikace a čtení z paměti ROM.

```
--text 12.
start:                  IN          temp1, UARTstat
                        TEST        temp1, 8
                        JUMP        Z, start

                        IN          RxDat, UARTdat
                        CALL        ComData

                        COMP        Escreg, 1
                        JUMP        Z, Command

                        CALL        SendChar

                        JUMP        start
```

Oproti zdrojovému kódu pro samostatný UART (viz text. 5) zde přibylo několik příkazů. Prvním z nich je volání programové rutiny, která zjišťuje, jestli je právě přijatý znak znakem odpovídajícím klávese Esc (viz následující zdrojový kód).

```
--text 12.
ComData:                COMP        RxDat, 27
                        CALL        Z, Switch
                        RET
```

Pokud byl přijat znak Esc, nastaví se příznak zero. Na základě příznaku zero se pak provede podmíněné volání funkce Switch, ve které se provádí přepnutí z řežimu UART na režim ROM a obráceně.

```
--text 12.
Switch:                 XOR     Escreg, 1
                        COMP    Escreg, 1
                        CALL    Z, mes1
                        COMP    Escreg, 0
                        CALL    Z, mes2
                        RET
```

Příkaz **`XOR`** provádí logickou operaci XOR registru `Escreg` ,podle kterého se řídí přechod z režimu UART do režimu ROM. Výsledek logická operace XOR je roven 0 pokud jsou oba operandy stejné (log. 0 nebo 1) a roven 1 pokud jsou různé. Díky této operaci tedy se mění hodnota registru `Escreg` z nuly na jedničku a z jedničky na nulu pokaždé, když se nově příchozí znak vyhodnotí jako stisk klávesy Esc. Další čtyři příkazy pak jen zjišťují do jakého režimu se program přesouvá, aby mohl program uživatele o změně režimu informovat textovou zprávou(viz obr. 5.11). Program se příkazem **`RET`** vrátí zpět z volaných podprogramů do hlavní smyčky, kde následuje podmíněný skok do programové částí `Command` pro komunikaci s pamětí ROM.

```
--text 12.
Command:            IN      temp1, UARTstat
                    TEST    temp1, 8
                    JUMP    Z, Command

                    IN      RxDat, UARTdat
                    CALL    ComData

                    COMP    Escreg, 0
                    JUMP    Z, start

                    COMP    RxDat, 114
                    JUMP    Z, Read

                    COMP    rom2,2
                    JUMP    Z, Adress2
Adress2zpet:
                    COMP    rom2,3
                    JUMP    Z, SendCharRom
SendCharRomzpet:
                    COMP    rom2,1
                    JUMP    Z, Adress1
Adress1zpet:
                    JUMP    Command
```

Základ tvoří opět čekání na nově příchozí znak po sériové lince. Nově příchozí znak se opět prověří na klávesu *Esc* voláním podprogramu `ComData`. Jestliže příchozí znak není *Esc*, testuje program začátek příkazu pro čtení z paměti ROM. Příkaz pro čtení má tvar „rxx“, kde písmeno r představuje identifikaci příkazu pro čtení a písmeno x představuje hexadecimální číslici v rozsahu 0 až f. Pokud program neidentifikuje znak r, vrátí se do čekací smyčky pro detekci nového znaku.

Po identifikaci znaku r program příjme další dva znaky, o kterých předpokládá, že jsou číslicemi hexadecimální adresy v paměti ROM. Program postupně přijaté ASCII znaky převede na jím odpovídající číslice a spojí v jedno osmibitové hexadecimální číslo(viz text 13).

```
--text 13.
Adress1:            LOAD    rom1, RxDat
                    SUB     rom1, $30
                    SL0     rom1
                    SL0     rom1
                    SL0     rom1
                    SL0     rom1
                    ADD     rom2,1
                    JUMP    Adress1zpet

Adress2:            SUB     RxDat, $30
                    XOR     rom1, RxDat
                    ADD     rom2,1
                    OUT     rom1, ROM
                    JUMP    Adress2zpet
```

Takto získané číslo odešle na vstup paměti ROM a v dalším cyklu si z jejího datového výstupu odebere data uložená na příslušné adrese. S těmito daty program počká až je UART blok připraven vysílat a odešle data zpět uživateli.

7. IP generátor

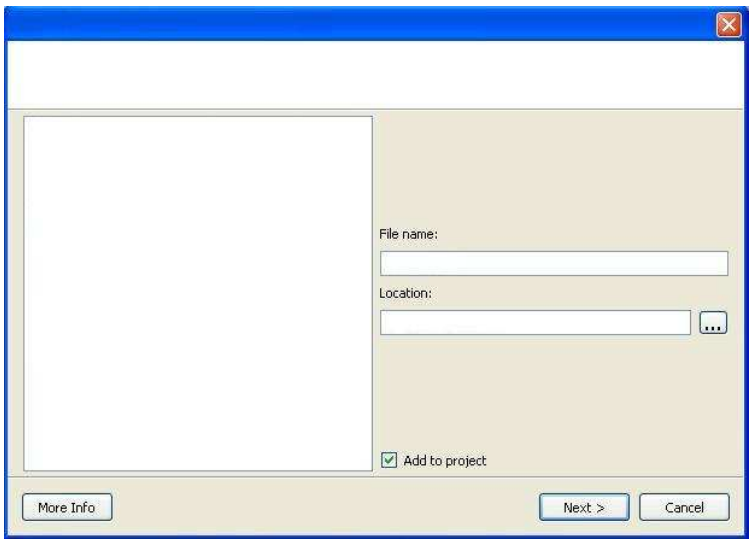
Smysl IP generátoru je v tom, aby bylo používání procesoru Picoblaze a jeho periferií co nejvíce intuitivní a aby byl takový návrh použitelný i pro uživatele bez rozsáhlých znalostí problematiky programování FPGA čipů. Výhoda použití tohoto IP generátoru je v tom, že se uživatel pouze rozhodne, které periferie se mají použít a vytvoří se kompletní popis se všemi nutnými zdrojovými soubory. S takto vygenerovaným návrhem bude možné přeskočit popisovou část návrhu obvodů FPGA a přejít rovnou k Implementační.

7.1 Návrhové prostředky

Pro IP generátor bylo zvoleno jazykové prostředí C/C++. V programovacím softwaru Borland C++ builder lze vytvářet programy na bázi objektového programování dvěma způsoby. Prvním klasičtějším způsobem je psaní zdrojového kódu standardními textovými příkazy, tak jak je to zvykem u většiny programovacích jazyků. Následující zdrojový kód představuje ukázku tohoto klasického způsobu psaní zdrojového kódu.

```
--text 14.
int main( void)
{
clrscr();                    // vymazani obrazovky
cprintf("Hello world\r\n");   // vypis na obrazovku
return 0;
}
```

Druhá možnost je vytvoření interaktivního grafického návrhu. Tento grafický design, povědomý ze starších operačních systémů Windows, je uživatelsky velmi přívětivý a přitom stejně účinný jako předchozí metoda.



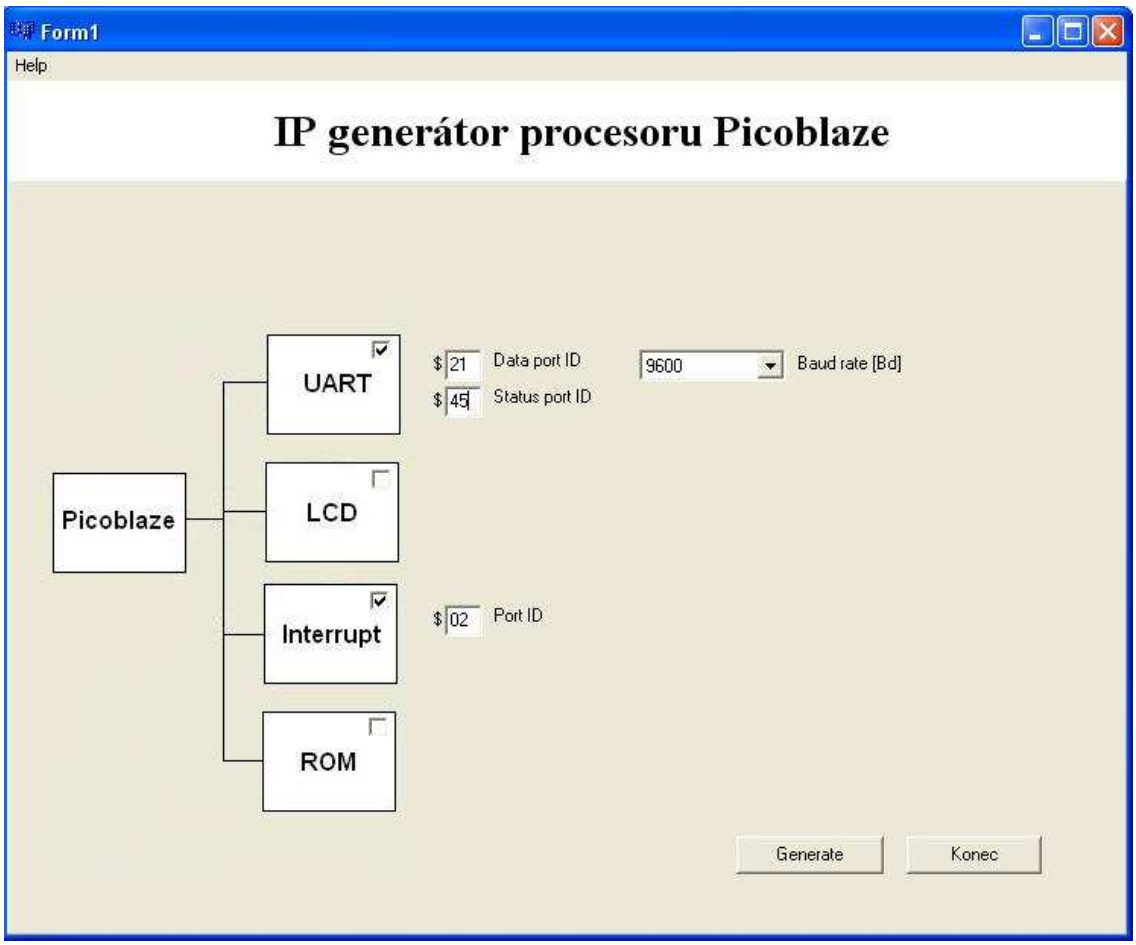
Obr. 7.1 Ilustrační obrázek grafického návrhu programu Borland C++

Editor vestavěný do tohoto programu umožňuje vložit celé řady použitelných struktur, jako jsou tlačítka, textová pole, rolovací lišty, zaškrtávací políčka a mnohé jiné. Se všemi těmito strukturami lze pracovat jako se samostatnými bloky, kterým

můžeme libovolně přiřazovat parametry a hlavně umožňují vložení zdrojového kódu, který bude vykonávat uživatelem požadovanou funkci.

7.2 Návrh IP generátoru

Jak už bylo popsáno výše, je IP generátor navrhnut v grafickém prostředí programu Borland Builder C++. Schéma procesoru a jeho periferií je řešeno ve formě propojených bloků, u kterých jsou v případě jeho volby zobrazeny okna pro nastavení jejich parametrů.



Obr. 7.2 Náhled grafického vzhledu IP generátoru pro procesor PicoBlaze

Jelikož procesor má jen jeden vstupní a jeden výstupní datový port, je nutné všechny jeho periferie, které jsou na něj připojeny, multiplexovat. Aby měl procesor přehled o tom, se kterou periferií zrovna komunikuje a jestli se jedná o datovou komunikaci nebo o řídící komunikaci, přiřazuje se každé takové komunikaci adresa, která je vždy v celém systému jedinečná. Tyto adresy jsou jedním z volitelných parametrů, které může uživatel měnit dle libosti. Je však nutné dodržet pravidla zápisu těchto adres a také jejich výše zmíněnou jedinečnost. Některé periferie mají ještě svůj vlastní specifický parametr. Jedním z nich je třeba přenosová rychlost u UARTu. U tohoto parametru může uživatel volit jednu ze sedmi nabízených rychlostí.

Poté co uživatel provede volbu požadovaných periferií a jejich parametrů, stiskne tlačítko Generate. Program spustí generování zdrojových souborů, přičemž se jako rozhodovací prvky berou volby ze zaškrtávacích polí a hodnoty zadané

v parametrech jednotlivých periferií. Příklad na obrázku 7.2 ukazuje, že uživatel vybral dvě periferie pro procesor a to řadiče UART a Interrupt. IP generátor tedy pomocí podmínek typu IF zjistí volbu v zaškrtávacím poli a na základě této volby zahrne do generace zdrojové kódy UART a Interrupt bloků.

```
if( CheckBox1->Checked)
```
[15]

Dále zasáhne do programu procesoru a vhodně umístěnými komentáři odstraní části programu, které se týkají nepoužitých periferií. Tento zásah se provádí v části **else** daného **if** a samozřejmě se musí odladit tak, aby nedošlo k ovlivnění částí programu, které jsou nutné pro správnou funkci procesoru i jeho použitých periferií. Posledním úkonem IP generátoru je upravení adresních konstant a ostatních volitelných parametrů, tak aby odpovídaly požadavkům.

Program bude mít předpřipravený zdrojové soubory všech periferií i program pro procesor. Pro případ zásahu do souboru kvůli komentaci či změně některého z parametrů se použije několika příkazů. IP generátor má všechny kompletní zdrojové kódy připravené v adresáři system a aby byl program použitelný vícekrát než jednou, je nutné, aby tento adresář zůstal nedotčený. Proto se na základě textu 15. překopírují do pracovního adresáře projekt všechny zdrojové kódy, se kterými se bude pracovat.

```
CopyFile("system\\Spartan.txt", "projekt\\Spartan.txt",0);
```
[15]

Nejprve se musí daný soubor otevřít, aby mohl program pracovat s daty uloženými v něm.

```
FILE *fw(char *filename, char *mode);
```
[12]

Tímto příkazem se otevře soubor jehož jméno je uloženo v řetězci filename. Tato funkce může pracovat v několika módech. Mód, ve kterém se daný soubor otevře se určuje řetězcem mode. V našem případě to bude mod "r+". Tento mód otevře zadaný soubor aniž by smazal jeho obsah a umožní zapisovaní do něj. Pokud zapisujeme data na určité místo, tak na toto místo ukazuje tzv. file-pointer. Pokud chceme zapisovat na určité místo je nutné použít funkci, která na toto místo přesune file-pointer.

```
int fseek (FILE *file, long offset, int mode);
```
[12]

Tato funkce přesune file-pointer souboru, jehož jméno je uloženo v řetězci file, na pozici vzdálenou o hodnotu uvedenou v parametru offset, od místa uvedeném v parametru mode.[12]

Parametr mode může obsahovat několik konstant odpovídající:

- seek_set Začátek souboru
- seek_cur Aktuální pozice v souboru
- seek_end Konec souboru

Po nastavení file-pointeru je již možné zapisovat požadovaná data do souboru.

```
int fprintf(FILE *file, char *format, c);
```
[12]

Tato funkce zapíše do souboru znak, který je uložen v proměnné c ve tvaru definovaném v parametru format. Do této proměnné se budou ukládat parametry zadané uživatelem před stiskem tlačítka generate. V praxi to tedy bude vypadat tak, že do proměnné c se uloží adresa datového portu UART a příkaz `fprintf` ji zapíše na příslušné místo ve zdrojovém kódu pro UART. Na stejném principu se budou vkládat komentové znaky ";"a "--" do programu pro procesor a všech použitých VHDL kódů, tak aby zůstaly aktivní jen ty části programu, které jsou požadovány.

`DeleteFile(char *filename);`[13]

Nakonec se provede příkazem `DeleteFile` odstranění všech souborů, které nebudou ve finálním návrhu použité a všech souborů, které v adresáři zbyly od předchozího generování. Ve zvoleném adresáři tedy zůstanou jen zdrojové soubory, které jsou potřeba pro kompilaci návrhu požadovaného uživatelem.

8. Laboratorní úloha

Cíle měření

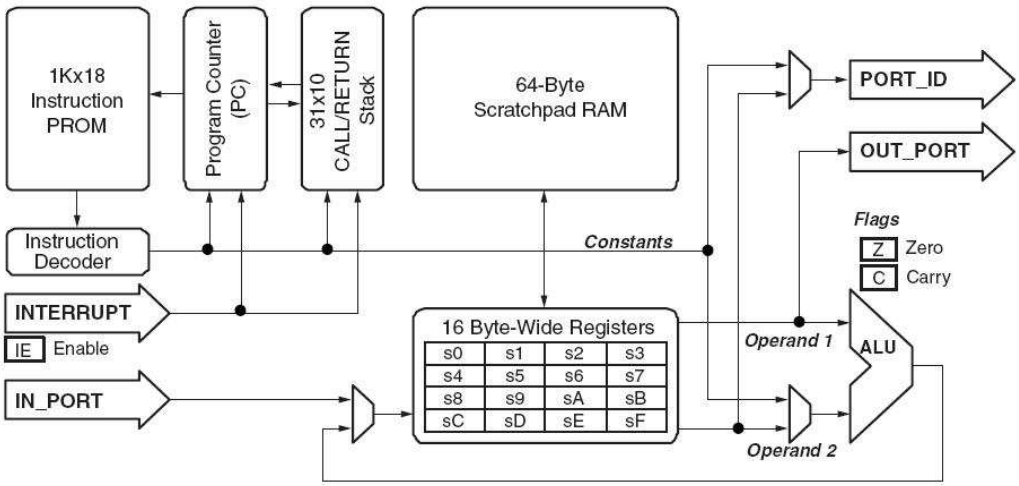
V této laboratorní úloze se studenti seznámí s jedním z nejjednodušších RISC procesorů Picoblaze, s jeho základními vlastnostmi, využitím a způsobem použití s nejběžněji používanými periferiemi v obvodech FPGA.

Zadání

1. Seznamte se se základními vlastnostmi a parametry procesoru Picoblaze, nejběžněji používanými periferiemi a způsobem práce s nimi.
2. V programu IP generator Picoblaze vytvořte návrh, ve kterém budou použity všechny dostupné periferie pro procesor Picoblaze.
3. Pomocí programů ISE WebPack a pBlazIDE implementujte vygenerovaný návrh do obvodu FPGA a ověřte jeho funkčnost.
4. Znovu vytvořte návrh v programu IP generátor Picoblaze s libovolným výběrem periferií a ověřte jeho funkčnost.
5. Zhodnoťte míru využitelnosti softwarových procesorů ve VHDL návrzích.

Úvod

Procesor PicoBlaze vyvinul pracovník firmy Xilinx Ken Chapman. PicoBlaze je jednoduchý osmibitový RISC procesor s Harvardskou architekturou. Tedy s oddělenou pamětí programu a dat. Tento procesor je ze skupiny volně dostupných procesorů. Všechny potřebné vývojové prostředky jsou tedy dostupné zdarma na internetu. PicoBlaze je procesor navržený pro čipy Spartan-3, Virtex II, Virtex II Pro a Virtex-4[7]. Největší předností jádra PicoBlaze je jeho velikost. V obvodu Spartan-3 totiž zabírá jen 96 řezů (slice), což je cca 5% z řezů dostupných v obvodu XC3S200 a méně než 0,3% z obvodu XC3S5000[7].



Obr. 8.1 Blokové schéma procesoru PicoBlaze

Základní parametry:

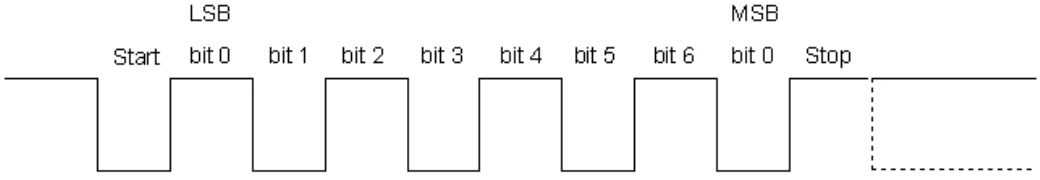
- 1Kx18 Instrukční paměť. Tedy paměť programu pro 1024 18-ti bitových instrukcí
- Programový čítač
- Zásobník pro ukládání návratových adres pro až 31 vnořených podprogramů
- Uživatelská paměť o velikosti 64 bajtů
- 16 osmibitových registrů (s0 až sF)
- Jeden zdroj přerušení
- Aritmeticko – logickou jednotku ALU s příznaky nuly a přenosu
- 256 osmibitových vstupů
- 256 osmibitových výstupů
- Reset

Procesor PicoBlaze je typu RISC tzn., že má redukovanou instrukční sadu. Instrukční sada procesoru PicoBlaze má tedy celkem 57 instrukcí a jejich kompletní význam a způsob použití lze nalézt v manuálu KCPSM3 [8].

V tomto protokolu se bude pracovat s nejběžněji používanými periferiemi při programování FPGA čipů.

- **UART**

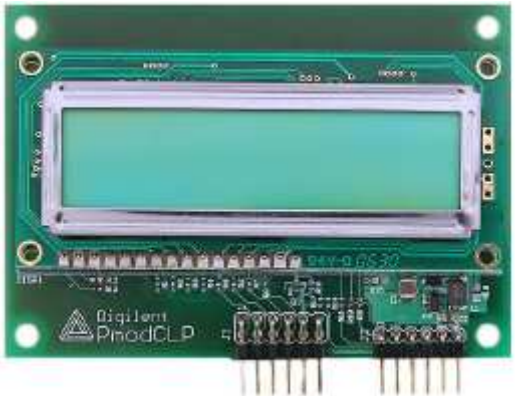
  Jedním z možných spojení dvojce stanic je UART, který realizuje asynchronní sériovou komunikaci mezi dvěma stanicemi. Datový vodič, kterým se přenášejí jednotlivé bity dat, je v klidovém stavu na hodnotě logické jedničky. Přijímací strana hlídá příchod logické nuly, která představuje synchronizační bit, podle kterého se přijímací strana synchronizuje. Za tímto synchronizačním bitem následují data bit po bitu obvykle v bloku 8 bitů. Toto množství lze však v případě potřeby volit i jinak (např. 7 nebo 9 bitů). Přenos dat se provádí od nejnižšího LSB bitu až po nejvyšší MSB bit. Pokud je pro zabezpečení přenosu použita parita, je tento paritní bit zařazen za posledním datovým bitem. Nakonec následuje jeden nebo dva tzv. stop bity, které ukončují datový rámec za nimž se datová linka uvádí opět do klidového stavu.



Obr. 8.2 Schéma datového rámce pro UART

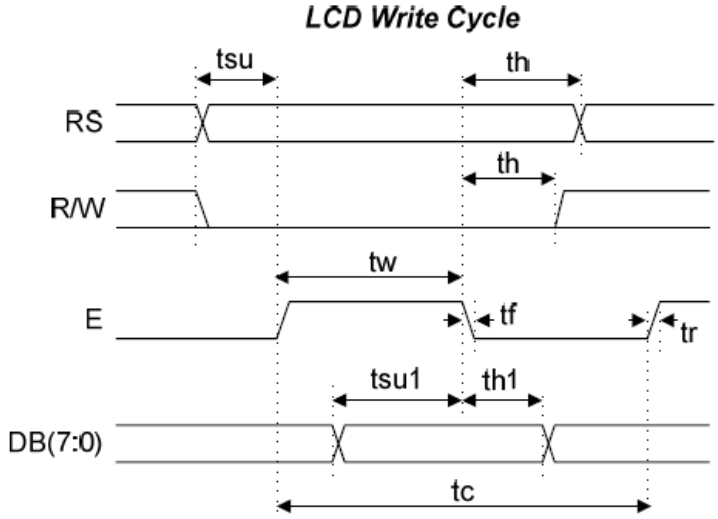
- **LCD**

  V tomto projektu je použit dvou řádkový, jednobarevný LCD display, který bude zobrazovat data ze sériové linky.



Obr. 8.3 Obrázek LCD display[14]

LCD display má tři řídící a jeden obousměrný osmibitový datový port. LCD pracuje s ASCII tabulkou, kterou lze v případě potřeby upravit.

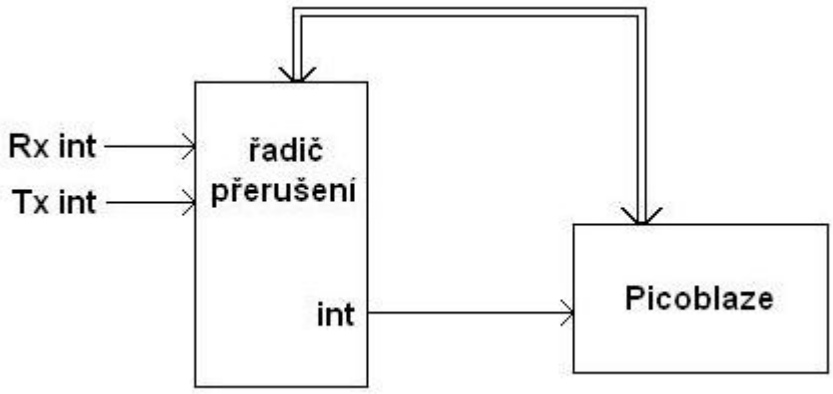


Obr. 8.4 Časové schéma jednoho čtecího cyklu pro komunikaci s LCD[14]

Signálem RS se LCD oznamuje, zda má očekávat na datovém portu příkaz nebo data. Signálem R/W se oznamuje LCD zda se budou data číst nebo se budovou data na LCD zapisovat. V tomto projektu se pracuje pouze se zápisem na LCD. Posledním řídícím signálem je signál E (enable), kterým se povoluje spuštění operace na LCD.

- **Interrupt**

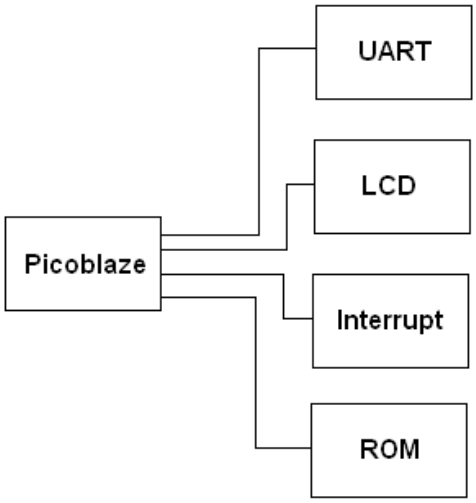
  Procesor Picoblaze má jeden zdroj externího přerušení (viz. kap. 4.1). Pokud tedy potřebujeme vytvořit návrh, ve kterém bude figurovat více druhů externích přerušení, je nutné vytvořit řadič přerušení, který se postará o jejich multiplexování na interrupt vstup procesoru Picoblaze.



Obr. 8.5 Schéma řadiče přerušení

Jako externí zdroje přerušení jsou použity dva signály. První je signál, který oznamuje nově příchozí data v Rx bloku periferie UART, která čekají na zpracování a druhým signálem je potvrzení dokončení vysílací sekvence Tx Blokem periferie UART. Ve VHDL bloku řadiče pak tyto dva signály generují signál přerušení běhu programu procesoru (viz obr. 8.5).
V řadiči přerušení se také zapisuje do status registru o jaké přerušení se jednalo. Procesor zaznamená na vstupním interrupt portu signál o příchodu přerušení. Dokončí právě probíhající instrukci a začne s obsluhou přerušení. V rámci této obsluhy si procesor nechá na vstupní port zaslat informace o původu přerušení a na základě této informace spustí příslušnou rutinu.

- **ROM**
  Blok paměti ROM představuje datové úložiště pro čtení dat do ní externě uložených. Data jsou uložena v souborech typu COE a každý znak zabírá jednu adresu paměti ROM. Pro čtení z paměti je nutné na vstup ROM přivést hexadecimální adresu, ze které je požadováno čtení dat a ROM odešle v dalším cyklu na výstup data odpovídající přijaté adrese.

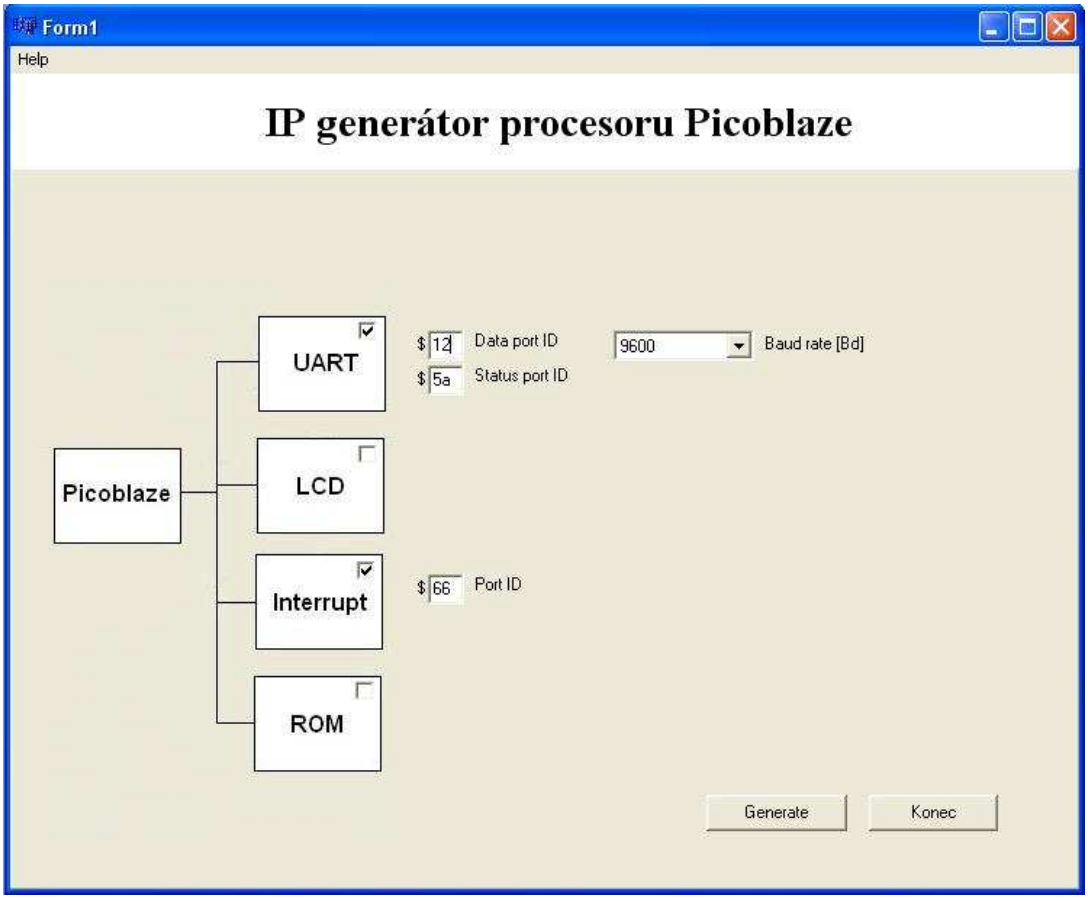


Obr. 8.6 Schéma návrhu procesoru Picoblaze s periferiemi

Takový systém lze samozřejmě vytvořit i bez použití procesorů. Absence procesoru však znamená, že je nutné veškerou logiku návrhu pospat pomocí VHDL (jako stavové automaty, čítače, atd.). Takový návrh je sice naprosto stejně spolehlivý jako s procesorem, ale obvykle je pro jeho implementaci potřeba více hardwarových prostředků obvodu FPGA. Případná modifikace takového návrhu je navíc často velmi obtížná a zdlouhavá. To je také největší přednost a hlavní důvod používání procesorů jako SoC (System on Chip) integrovaných do FPGA.

Postup

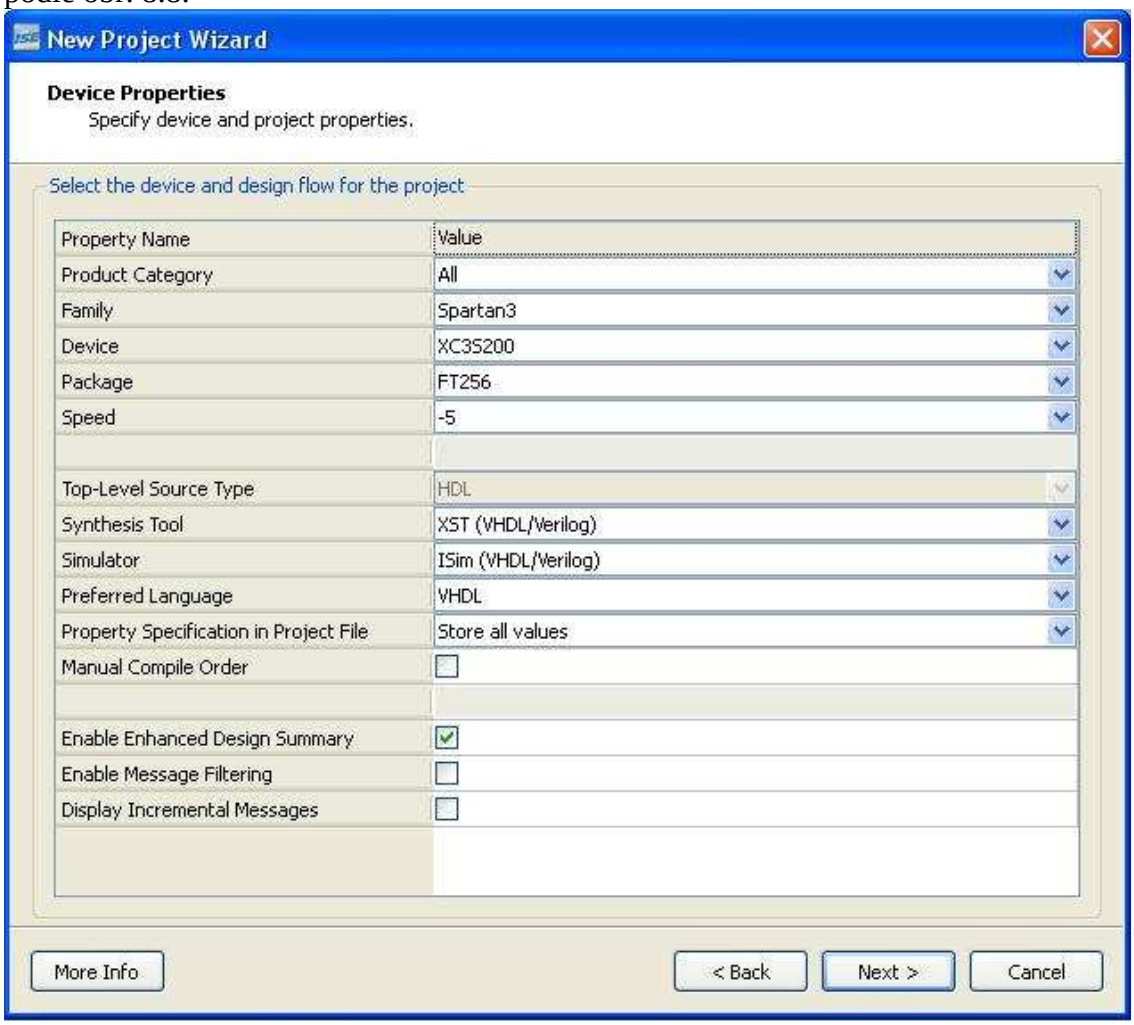
2) Spusťte program IP generátor Picoblaze. Ve schématu vyberte všechny dostupné periferie zaškrtnutím příslušného bločku. Po zaškrtnutí se objeví pole, do kterých je nutné vepsat parametry dané periferie. U všech periferiií je společným parametrem, který volí uživatel, adresa používaná pro multiplexování komunikace s procesorem Picoblaze. Pravidla pro adresy periferií jsou uvedena v nápovědě programu IP generátor Picoblaze. Po vyplnění všech polí stiskněte tlačítko Generace (viz obr. 8.7).



Obr. 8.7 Screen programu IP generátor Picoblaze

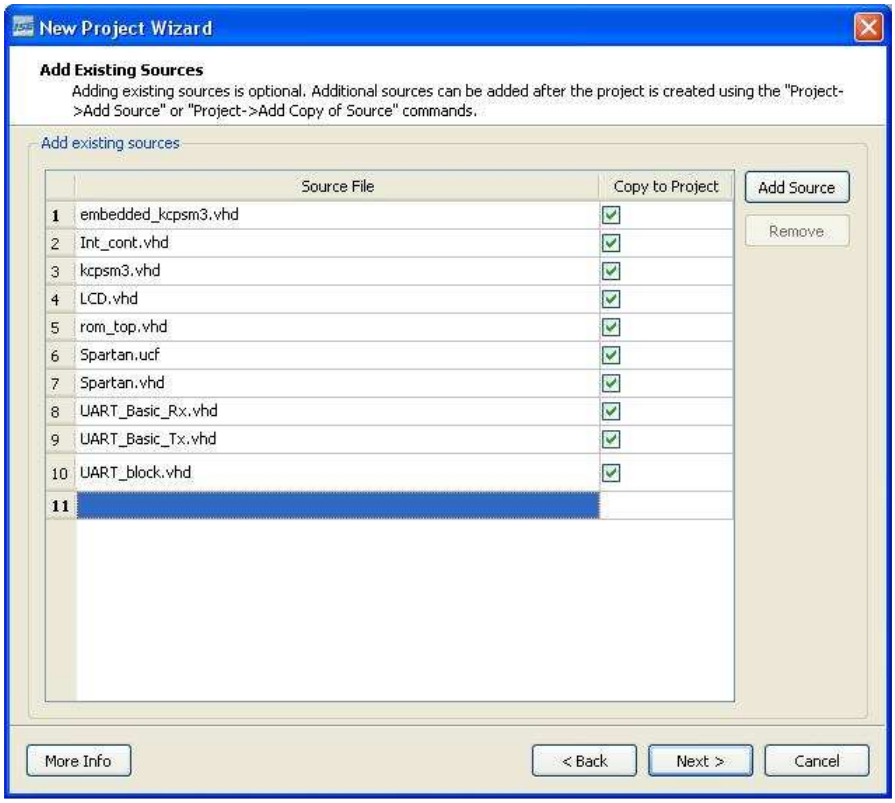
3) V adresáři projekt jsou nyní připraveny všechny zdrojové kódy nutné pro implementaci návrhu do FPGA. Spusťte program pBlazIDE. V menu File→Open najděte soubor v adresáři projekt s názvem Pico.psm, který představuje program pro procesor Picoblaze. Klávesou Assemble & Simulate (F12) zkompilujte program do VHDL kódu.

Spusťtě program ISE WebPack. Založte nový projekt File→New project. V prvním okně projekt nazvěte a vyberte umístění. Jako „Top-level source“ type vyberte „HDL“ a pokračujte „next“. Následující okno výberu cílového obvodu vyplňte podle obr. 8.8.



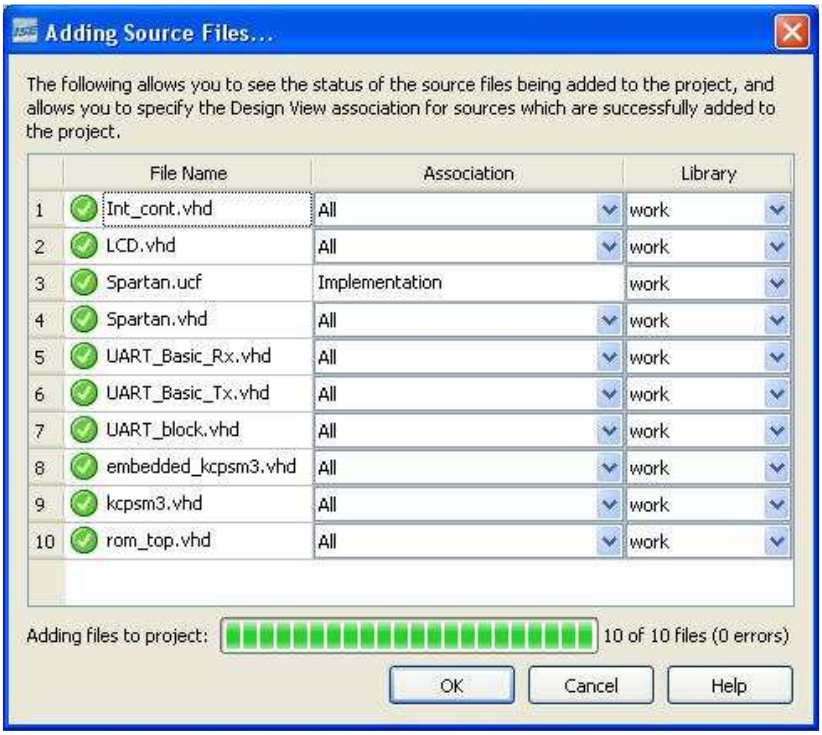
Obr. 8.8 Výběr cílového obvodu

Obrazovku „Create new source“ přeskočte stiskem „next“. Nyní stiskněte tlačítko „add source“ a v adresáři projekt programu IP generátor Picoblaze vyberte všechny zobrazené soubory kromě souboru „ROM_blank.vhd“. Potvrzením se všechny vybrané soubory zobrazí v přehledu (viz obr 8.9).



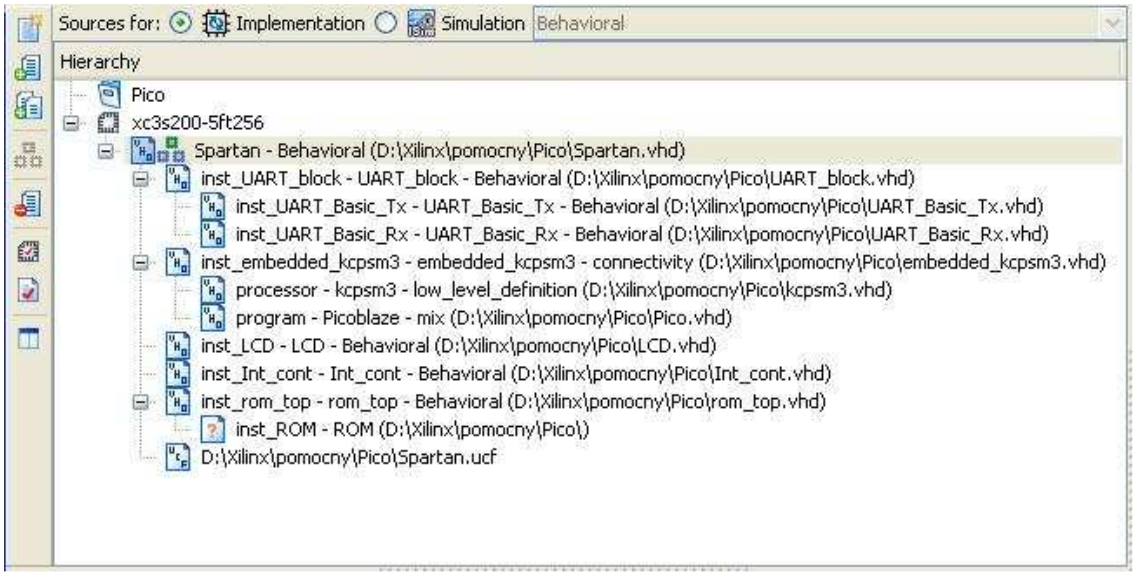
Obr. 8.9 Přidávání zdrojových souborů do projektu

Po stisku dokončení tvorby projektu (next→finish) program ISE překopíruje vybrané zdrojové kódy do složky nového projektu a zobrazí jejích přehled (viz obr. 8.10).



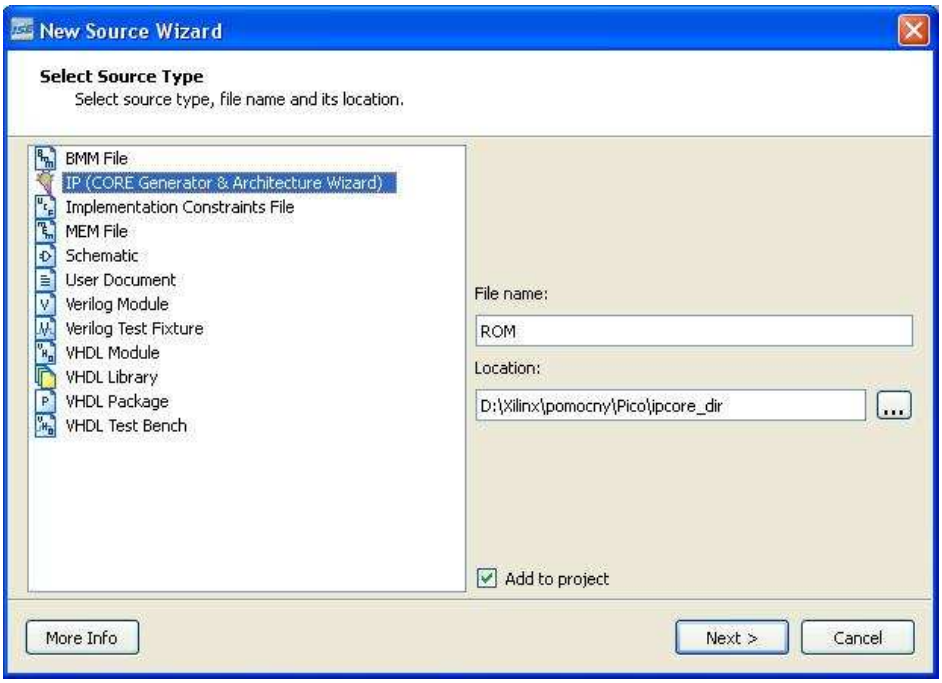
Obr. 8.10 Přehled přidaných zdrojových kódů do projektu

V okně Hierarchy jsou nyní vidět všechny přidané zdrojové soubory. Pokud jste při generaci návrhu nezahrnuly periferii ROM, přejděte rovnou k implementaci. Po rozbalení komponenty rom_tom lze vidět, že komponenta ROM periferie ještě není kompletní (viz obr. 8.11).



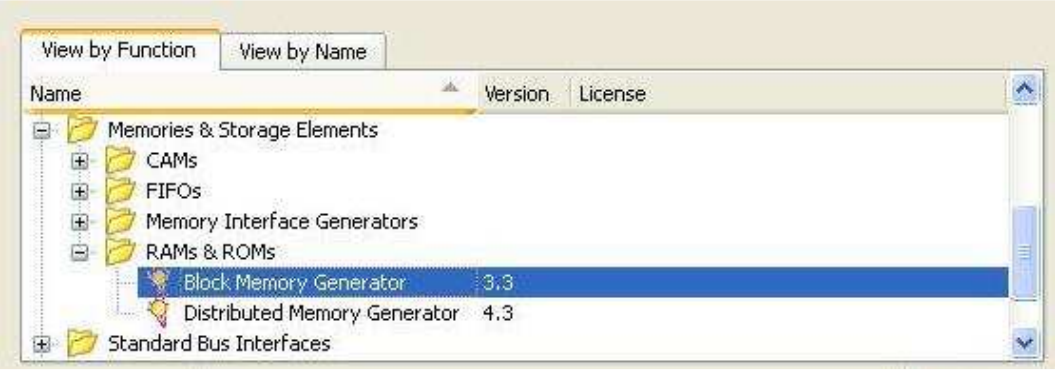
Obr. 8.11 Přehled Hierarchického uspořádání zdrojových kódů v projektu

Zbývá přidat komponentu ROM. Stiskněte pravé tlačítko myši a ze seznamu vyberte „New source“. V nabídce „source type“ vyberte „IP(CORE generátor & architecture wizard“ a jako název zvolte „ROM“ (viz. obr 8.12).



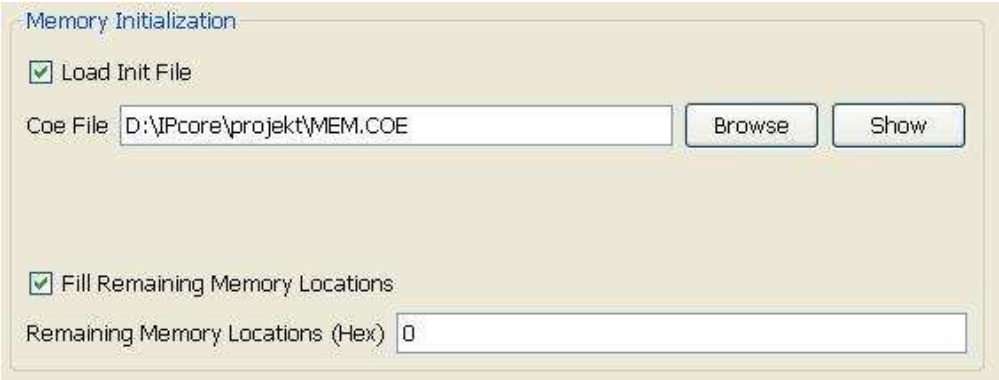
Obr. 8.12 Vložení komponenty IP jádra

V záložce „view by function“ vyberte „Memories&Storage elements→RAMs & ROMs→Block Memory Generátor“ (viz obr 8.13).



Obr. 8.13 Výběr komponenty ROM

Po dokončení výběru komponenty se zobrazí okno pro nastavení parametrů paměti ROM. Na první stránce vyberte v okně „Memory type“ „Single port ROM“. Na Další straně v okně „Memory Size“ nastavte parametr „Read Width“ na hodnotu 8 a „Read Depth“ na hodnotu 256. Na třetí straně v okně „Memory Initialization“ zaškrtněne volbu „Load Init File“ a přidejte soubor MEM.COE umístěný v adresáři projekt IP generátoru Picoblaze (viz obr. 8.14). Dokončete generování ROM stiskem tlačítka „Generate“.



Obr. 8.14 Připojení souboru s daty k paměti ROM

Implementace – Nyní je projekt kompletní a je možné jej implementovat do cílového obvodu. Označte v okně Hierarchy vrcholovou komponentu Spartan a v okně „Process Spartan – Behavioral“ dvojklikem spusťte „Configure Target Device“. Po otevření programu ISE Impact stiskněte ikonu “File→Open Project“ a v adresáři projekt programu IP generátor Picoblaze nalezněte soubor „Spartan.ipf“. Po načtení souboru dvojklikem otevřete komponentu „xc3s200“ a v adresáři Vašeho ISE projektu nalezněte soubor „Spartan.bit“. Pravým tlačítkem na tuto komponentu a stiskem „Program“ implementujete celý projekt do obvodu FPGA.

4) Opakujte body 2) a 3) tentokrát s vlastním výběrem periferií. Vzhledem k podstatě návrhu je se doporučuje zahrnutí periferie UART do návrhu. I přesto, že její absence není pro implementaci kritická, je případné ověření funkčnosti ostatních periferií problematické.

Použité přístroje

- PC
- Spartan-3 Starter Kit

9. Závěr

V této diplomové práci byla nastíněna základní problematika programovacího jazyku VHDL. V další kapitole byl probrán programovací nástroj ISE Webpack, jeho vlastnosti a způsob práce v něm. Také zde byla nastíněna oblast používání mikroprocesorů při implementaci do obvodů FPGA. U jednoho z nich, mikroprocesoru PicoBlaze, byly popsány parametry a jeden ze způsobů realizace. Podle zadání zde byly rozebrány některé z nejpoužívanějších periferií, způsob práce s nimi, jejich VHDL popis a simulace. Dále zde byla uvedena realizace ovládání těchto periferií pomocí procesoru Picoblaze. Hlavním výstupem diplomové práce je IP generátor, který umožní automaticky vytvářet jednoduchý systém s procesorem Picoblaze pro obvody FPGA. IP generátor je navržen jako grafické intuitivní rozhraní, se kterým může pracovat i uživatel, který není zběhlý v problematice VHDL a mikroprocesorů pro FPGA. Nic méně je nutné, aby obsluha zvládala základní ovládání programu ISE a pBlazIDE, neboť IP generátor není navržen pro kompilaci a implementaci zdrojových kódů do čipu FPGA. V poslední části práce je počítačová úloha zaměřená na použití IP generátoru v laboratorní výuce předmětu Programovatelné logické obvody (MPLD), ve kterém bude studentům ukázána základní metodika použití softwarových procesorů v obvodech FPGA.

10. Zkratky

IEEE	Institute of Electrical and Electronics Engineers
HDL	Hardware Description Language
VHDL	Vhsic Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
PLD	Programmable logic Device
FPGA	Field Programmable Gate Array
ISE	Integrated Software Environment
MIPS	Milion Instructions Per Sekond
RISC	Reduced Instruction Set Computer
KCPSM	(Konstant coded/Ken Chapman) Programmable State Machine
pBlazIDE	picoBlaze Integrated Development Environment
UART	Universal Asynchronous Receiver and Transmitter
LCD	Liquid Crystal Display
AD/DA	Analog to Digital/ Digital to Analog
LSB	Least Significant Bit
MSB	Most Significant Bit
SoC	System on Chip

11. Literatura

[1] KOLOUCH, J. Programovatelné logické obvody. Elektronické texty a přednášek a počítačových cvičení. Brno:FEKT VUT v Brně, 2007

[2] Xilinx. ISE WebPACK. [online]. 2008 – [cit. 3. května 2010]. Dostupné na WWW: http://www.xilinx.com/ise/logic_design_prod/webpack.htm

[3] Nápověda programu ISE WebPACK, spustitelná z programu. Xilinx

[4] PicoBlaze, J. Svozil, L. Kafka, J. Kadlec. [online]. – [cit. 3. května 2010]. Dostupné na WWW: www.vlam.cz/files/lectures/pb_lecture1.pdf

[5] Kolouch, J. Implementace procesorů v obvodech FPGA. [cit. 3. května 2010]. Dostupné na WWW: Www.urel.feec.vutbr.cz/web_pages/projekty/clanky/Kolouch_FPGA.pdf

[6] Martínek, T. Picoblaze, MicroBlaze, PowerPC. [cit. 3. května 2010] Dostupné na WWW: http://www.fit.vutbr.cz/study/courses/NCS/public/prednasky/ppt/PicoMicroPowerPC.ppt

[7] Xilinx. PicoBlaze™ 8-bit Microcontroller Reference Design for FPGAs and CPLDs. [cit. 3. května 2010] Dostupné na WWW: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/picoblaze_productbrief.pdf

[8] PicoBlaze 8-bit Embedded Microcontroller User Guide. [cit. 3. května 2010] Dostupné na WWW: http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

[9] Chapman, K. KCPSM3 8-bit Micro Controller for Spartan-3, Virtex II and Virtex II Pro. [cit. 3. května 2010]. Dostupné na WWW: http://www.eng.auburn.edu/~strouce/class/elec4200/KCPSM3_Manual.pdf

[10] Mediatronix. pBlazIDE. [online]. – [cit. 3. května 2010]. Dostupné na WWW: http://www.mediatronix.com/pBlazeIDE.htm

[11] Xilinx, Initial Design for the Spartan-3E FPGA Starter Kit Board [cit. 31. prosince 2010] Dostupné na WWW: http://www.xilinx.com/products/boards/s3estarter/files/s3esk_startup.pdf

[12] Builder, Učíme se C (20. díl) - Práce se soubory II.Board [cit. 2. ledna 2011] Dostupné na WWW: http://www.builder.cz/art/cpp/cpp_file2.html

[13] Linuxsoft, C/C++ (24) – Soubory [cit. 2. ledna 2011] Dostupné na WWW: http://www.linuxsoft.cz/article.php?id_article=899

[14] Digilent Inc., Parallel LCD Display Module [cit. 12. května 2011] Dostupné na WWW:
http://www.digilentinc.com/Data/Products/PMOD-CLP/PmodCLP_rm_RevA.pdf

[15] Nápověda programu C++ Builder 6, spustitelná z programu. Borland