VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

REFERENČNÍ PŘÍKLADY PRO ŘÍDÍCÍ SYSTÉM EUS FS

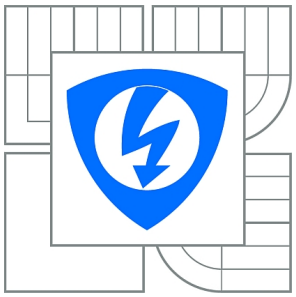
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PAVEL JÍŠA
AUTHOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

REFERENČNÍ PŘÍKLADY PRO ŘÍDÍCÍ SYSTÉM EUS FS

EUS FS REFERENCE DESIGN

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

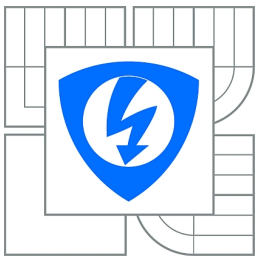
AUTOR PRÁCE — PAVEL JÍŠA

AUTHOR

VEDOUCÍ PRÁCE — Ing. SOBĚSLAV VALACH

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Pavel Jíša	***ID:***	106507
Ročník:	3	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Referenční příklady pro řídící systém EUS FS

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s řídícím systémem EUS FS a navrhněte několik referenčních příkladu pro hradlové FPGA a Linuxový procesor.

DOPORUČENÁ LITERATURA:

www.xilinx.com (řady spartan 3E)
www.axis.com (procesor Etrax FS a OS Linux)
www.ti.com (procesor MSP430)

Termín zadání:	8.2.2010	***Termín odevzdání:***	31.5.2010

Vedoucí práce: Ing. Soběslav Valach

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato bakalářská práce se zabývá popisem zařízení EUS FS a v další části vytvoření referenčních příkladů na hradlové pole FPGA a další příklady na kooperaci hradlového pole a linuxového procesoru, dále je popsán program BUS pro práci se sběrnicí a byla provedena jeho modifikace.

KLÍČOVÁ SLOVA

EUS FS, FPGA, VHDL, čítače, Spartan3e, Etrax FS, Picoblaze, spolupráce procesoru s FPGA,knihovna BUS_SPACE, program Bus

ABSTRACT

This bechelor thesis, in the first part, is focused on a description of the EUS FS device and in a second part there is created reference excercise on gate array FPGA and other examples on cooperation of a gate array and linux processor. Next step describes the BUS programme which is designed for a work with a bus. There has been made a modification of this Bus programme.

KEYWORDS

EUS FS, FPGA, VHDL, counters, Spartan3e, Etrax FS, Picoblaze, cooperation processor with FPGA, library BUS_SPACE, program Bus

JÍŠA, Pavel *Referenční příklady pro řídící systém EUS FS*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2010. 72 s. Vedoucí práce byl Ing. Soběslav Valach,

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Referenční příklady pro řídící systém EUS FS“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V první části této práce se budu zabývat seznámení s řídícím systémem EUS FS - popisem jeho vlastností, chování a funkcí nejdůležitějších jeho obvodů.

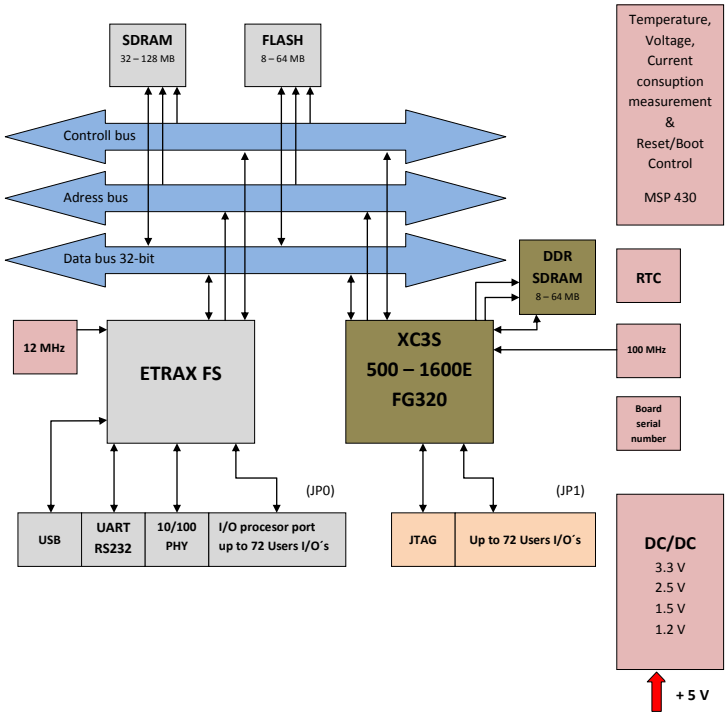
V další části této práce bych se rád zaměřil hlavně na tvorbu referenčních příkladů na tento systém - spíše tedy na hradlové pole a také na jeho jednoduché propojení s linuxovým procesorem a také vytvořit nějaký jednoduchý příklad na linuxový procesor. Referenční příklady bych rád koncipoval jako seznámení člověka který programování hradlového pole nikdy nedělal, postupně od nejzákladnějších věcí až po složitější. Budu se snažit použít jednoduché a lehce pochopitelné postupy aby čtenář pochopil základní prvky pro práci na hradlovém poli a principy použité komunikace s procesorem a princip práce s ním. Příklad bych rád koncipoval tak, aby pomocí něho byla možná komunikace s FPGA.

1 POPIS MODULU EUS FS, STARTER KITU A JEJICH PERIFERIÍ

V této kapitole se budu zabývat samotným popisem řídícího systému zejména ze strany hardwaru. Dále se budu zabývat možností připojení periferií pomocí starter kitu - hlavně jeho obsah a rozhraní.

1.1 Řídící systém EUS FS

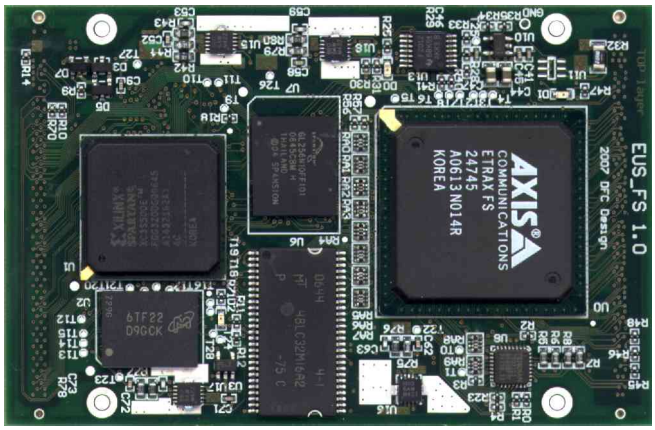
Jedná se o ”otevřenou”(licence GNU GPL) jednoprocesorovou desku, která je ovládána pomocí operačního systému Linux. Tento systém je vytvořen pro průmyslové řízení a pro aplikace na sběr. Je vybaven 32-bit RISC procesorem Axis ETRAX FS pracující na frekvenci 200 MHz, hradlovým polem Xilinx Spartan 3E - programovatelné hradlové pole (do kterého můžeme libovolně pomocí programovacího jazyka VHDL či VERILOG vytvořit vlastní strukturu) a podpůrnou elektroniku více v [1].



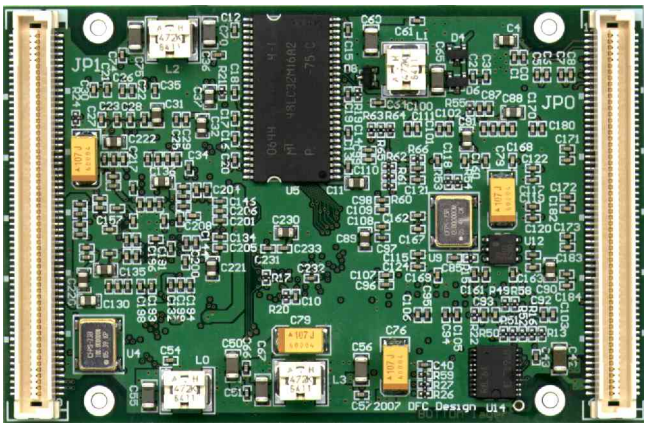
Obrázek 1.1: Blokové schéma systému EUS FS [1]

Přehled hardwaru použitém v systému:

- 200 MHz RISC procesor Axis ETRAX FS s nahraným jádrem operačního systému Linux, tento procesor obsahuje spoustu možných rozhraní a funkcí viz dále
- Programovatelné logické pole typu FPGA Spartan XC3S500E (může se lišit dle konfigurace)
- Sdílená operační paměť SDRAM o velikosti 128 MB běžící na frekvenci 100 MHz(může se lišit dle konfigurace)
- Sdílená uživatelská paměť typu NOR flash o velikosti 64 MB (může se lišit dle konfigurace)
- DDR RAM paměť určená pro hradlové pole běžící přibližně na frekvenci 100 MHz
- Mikrokontrolér MSP430 sloužící k managmentu desky - hlavně se stará o napájení(jeho chování lze upravit modifikací programu)
- Obvod obsahující unikátní seriové číslo každé desky
- Hodiny reálného času
- 120-ti pinová patice sloužící pro připojení periferií k hradlovému poli
- 120-ti pinová patice kde jsou vyvedeny všechny vstupně - výstupní obvody (I/O) procesoru
- Enthernetové rozhraní
- Interní komunikační serivá sběrnice Komunikační seriová sběrnice (I_2C)
- více informací v [1]



Obrázek 1.2: Fotografie reálného systému EUS FS z vrchní strany [1]

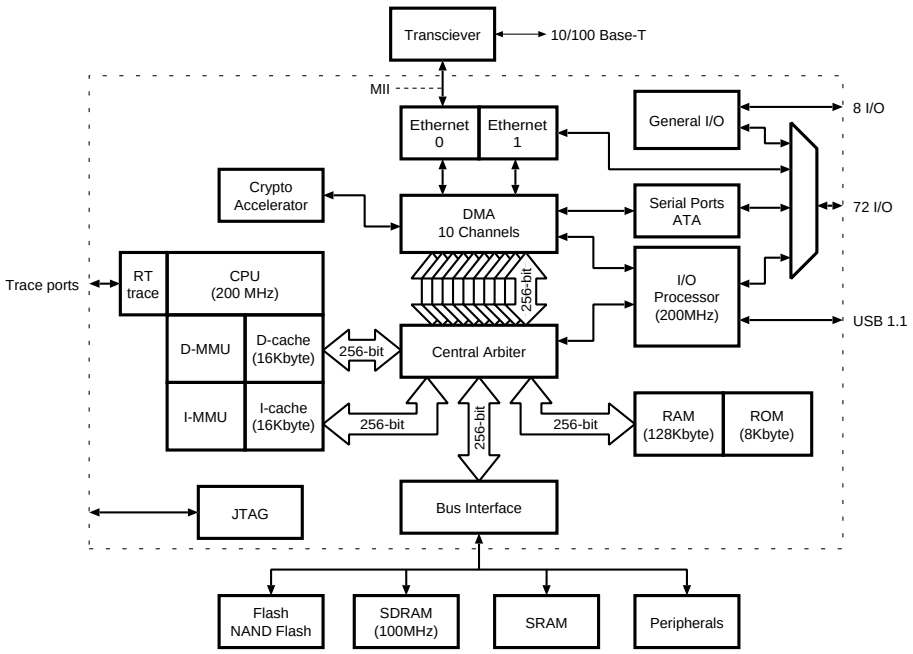


Obrázek 1.3: Fotografie reálného systému EUS FS ze spodní strany [1]

1.2 Procesor Axis ETRAX FS

Jedná se, jak již bylo řečeno, o procesor tipu RISC běžící na frekvenci 200 MHz. Tento procesor má délku datové a adresové sběrnice 32-bit, instrukce však mají délku 16-bit a jsou optimalizovány pro kompaktní kód. Obsahuje hardwarovou podporu šifrování. Procesor je konstruován pro velké možnosti připojení periferií pomocí I/O obvodů. Základní vlastnosti procesoru:

- 32-bitová RISC architektura běžící na frekvenci 200 MHz
- Obsahuje MMU pro reálnou podporu opreačního systému Linux
- Podporuje paměti:
  - SRAM
  - SDRAM
  - EPROM
  - EEPROM
  - NOR/NAND Flash PROM
- Má integrovaný I/O procesor pro:
  - Ethernet
  - Synchronní seriový port
  - ATA rozhranní
  - EEPROM
- Je k němu dostupný Linux s jádrem verze 2.6 a volně dostupnými vývojovými nástroji



Obrázek 1.4: Funkční blokový diagram [3]

1.2.1 Botování procesoru

Tento procesor obsahuje více možností bootování. V této aplikaci je použito bootování z:

- NOR Flash
- Sítě RX kanálu
- Sítě RX/TX kanálu
- Sériového rozhraní

Nejvíce se zde používá bootování z přiložené NOR Flash paměti.

1.2.2 Architektura procesoru

Jak již vyplývá z funkčního blokového diagramu tak tento procesor obsahuje zařízení zařízení pro zprávu paměti tak i zařízení pro přístup do paměti bez použití Central procesor unit - hlavní výpočetní jednotka v procesoru (CPU). Obsahuje také obvody pro hardwarovou akceleraci šifrování, RAM a ROM paměťovou jednotku.

32-bit RISC CPU

Procesor je tipu RISC běžící na frekvenci 200 MHz. Má délku datové a adresové sběrnice 32-bit, instrukce však mají délku 16-bit a jsou optimalizovány pro kompaktní kód. Instrukce jsou před načítány a CPU dynamicky předvídá jejich další větvení. Obsahuje 256 bitovou systémovou sběrnici. Dále obsahuje možnost až 5-ti stupňového větvení instrukcí z tohoto plyne že je schopen v jednom cyklu provést násobení. Chráněný přístup do paměti je realizován pomocí Uživatelského i jaderného režimu.

DMA Direct memory access - Přímý přístup k paměti

Jedná se o obvody díky kterým můžou periferní obvody přistupovat k interní paměti bez využití CPU. Tvoří ho 10 kanálů, každý tvoří 64 bytový registr typu FIFO pro nízkou odezvu a vysoký výkon přenosu dat do a z interní a externí paměti. Šířka pásma pro jeden kanál je 400 MB/s.Podporuje vícenásobné virtuální kanály s rychlým přepínáním kanálů.

MMU - Memory management unit - jednotka řízení paměti

Oddělené instrukce a data jednotky řízení paměti předvídá až 4GB z virtuálního jednotného adresového prostoru pro každý uživatelský proces. Ochrana adresového prostoru. Podporuje nulové kopírování sdílených paměťových schemat a připojuje dva 64-vstupové Translation lookaside buffers - překládající posraní vyrovnávací paměť.

Cache memory - vyrovnávací paměť

Multi procesorová cache paměť povoluje instrukční a datovou cache paměť. Každá cache paměť je velká 16KB a 2 způsoby nastavení asociativní a má 256 bitové rozhraní do systémové sběrnice pro vysoký vnitřní BANDWITH. Sledovací cache paměť se coherentním mechanismem (MESI protokol).

Clock generator

Vnitřní 200MHz operační frekvence, generovaná PLL z externího 12MHz oscilátoru.

Interup control - Řízení přerušení

Vektorizované přerušení: Vnitřní(I/O porty, síťové rozhraní, Direct memory access - Přímý přístup k paměti (DMA), a od časovače), vnější přerušení(IRQ, NMI) ZJISTIT.

Central arbiter - centrální rozhodovací jednotka

Centrální rozhodovací jednotka zajišťuje neblokovaný přístup do vnitřní paměti skrz externí paměťový přístup. Maximální rychlost přenosu je 1.6.GB/s.

Internal RAM - Vnitřní RAM paměť

Tento procesor obsahuje 128KB paměti připojené na vnitřní sběrnici o šířce 256 bit. Doba cyklu této paměti je 20ns.

On-chip Debug

V obvodu pro ladění obsahující Rozhraní sloužící k nahrávání a ladění aplikací (JTAG) rozhraní ([11]) nepotřebuje funkční softwarový plán. Hardwarové nahlížecí, zastavovací body a jednotný krok. Trasování a nahlížení se provádí v reálném čase vyhrazeným vysokorychlostním I/O rozhraním.

Crypto accelerator - Šifrovací akcelerátor

Konfigurovatelný, je schopný hardwarově akcelerovat šifrování typu DES, 3DES, AES, MD5, SHA-1, a počítat IP součet pro zašifrování dat. Datová propustnost akcelerátoru je 2*100Mbit/s. Je stejně efektivní pro sériovou nebo paralelní konfiguraci pro rostoucí stromový algoritmus.

1.2.3 Vstupně výstupní obvody

Procesor obsahuje mimo jiné ještě Dále obsahuje podprocesor starající se o I/Orozhraní a také obsahuje velké možnosti variant I/O obvodů.

General purpose port - Hlavní port

Jedná se o 80 configurobatelných vstupně výstupních pinů multiplexovaných I/O funkcemi. 8 pinů může být nakonfigurováno jako vstupní pro přerušení.

Dual ethernet controller's

Dvojitá 10/100Mbit/s ethernet MAC adresa kompatibilní se standardem IEEE 802.3 a Fast Ethernet standardem.

4 asynchnonní Sériové porty

Plný buffering a kontrola parity. Jednotné míchání signálu z každého portu. Podporuje oslovení, řízené přerušení a operace kontrolované DMA. Nezávislé RX a TX operace. Podporuje baudovu [1] rychlost od 56.25 baud do 12.5 Mbaud.

[1] je modulační rychlost (také znaková rychlost nebo anglicky baud rate) udávající počet změn stavu přenosového média za jednu sekundu. Pro některé typy modulací může platit, že 1 baud = 1 bit/s. (Baud (Bd))

2 synchronní sériové porty

I2S master a slave mode s vnitřním a vnějšími časováním. Universální SPI rozhraní zvládající běh až na 16.67 MHz, některé módy až 50 MHz. Částečná podpora I_2C. Kompatibilní se spoustou podobných synchronních seriových protokolů. Dále podporuje mód IEC 60958. Obsahuje až 50Mhz vnitřní hodinový generátor. Až 100 Mbit/s přenos a 300 Mbit/s pro příjem v chip-to-chip módu.

ATA rozhranní

Jedná se o jeden ATA řadič. Podporuje PIO mode 4 (8 MB/s), Víceslovný DMA mode 2 (16 MB/s) a UltraDMA mode 2 (33MB/s). Je možno nakonfigurovat až 4 ATA porty pro připojení az 8 IDE zařízení.

I/O Processor

Jeden hlavní a dva podřízené 32-bitové procesory běžící na frekvenci 200 MHz. Má přístup na vnitřní sytémovou sběrnici a DMA. Obsahuje vlastní - lokální paměť. Funkční bloky dostupné v hlavním CPU jsou generátor hodinového signálu, časovače, logický spoštěč, hardwarově akcelerovaný kontrolní součet - CRC. Schopný implementace minimálně 2 I/O protokolů současně v mikro kódu. Obsahuje 72 100 MHz I/O pinů. Může být programován pro podporu paralelních a seiových portů, PC-Card/CardBus/PCI, USB2.0FS/HS host a zařízení, SCSI-2/SCSI-3, ATA PIO mód 4 (8MB/s)/Multiword DMA mode 2 (16MB/s)/Ultra DMA mód 2 a 5 (33 a 100MB/s), a vlastní rozhraní.

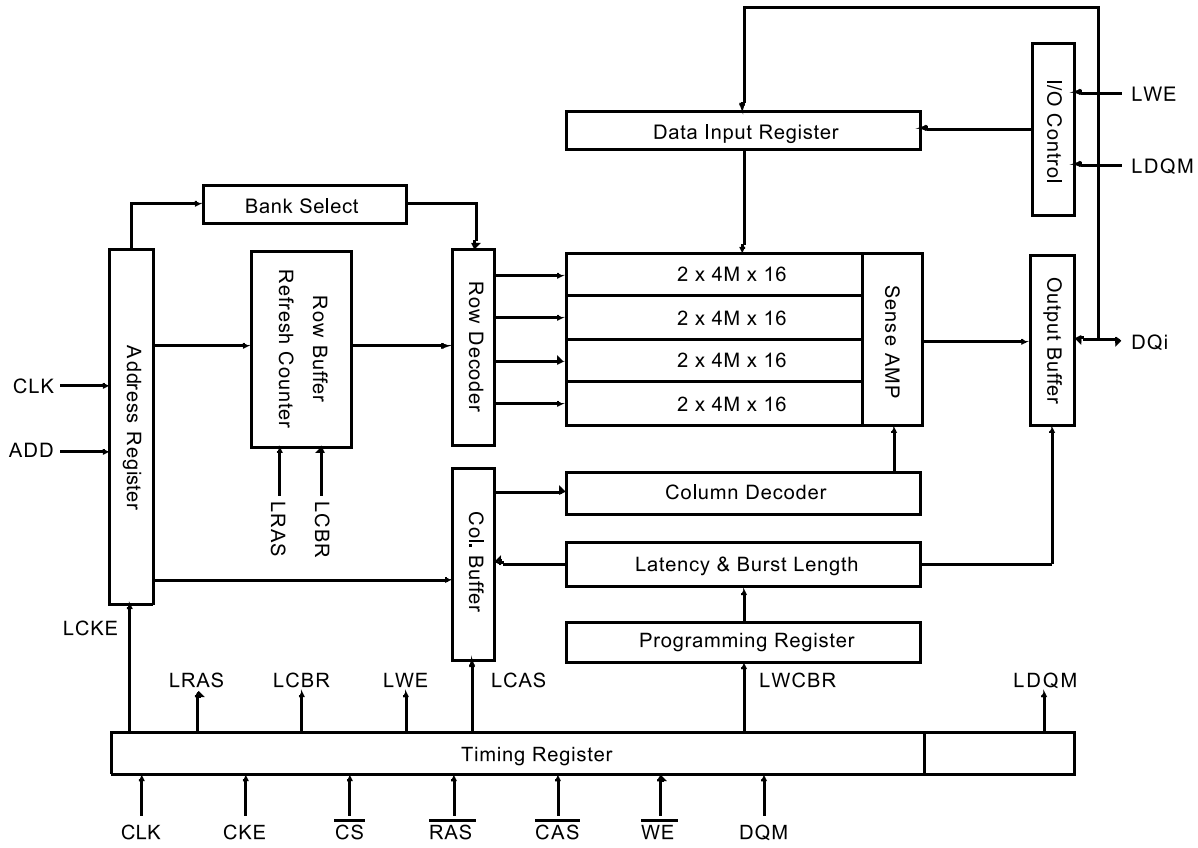
Více o procesoru ETRAX, jeho možnostech a popis jednotlivých vlastností najdeme v [3].

1.3 Paměti

Toto zařízení obsahuje několik různých pamětí.Jak operační paměť typu SDRAM tak i paměť pro uložení dat. Obsahuje i operační paměť typu DDR která je vyhrazena speciálně pro hradlové pole. Přístup k paměti je vlastně rešen pomocí režimu master-slave kde jako master je vždy zařízení vyžadující informaci.

1.3.1 Paměť SDRAM

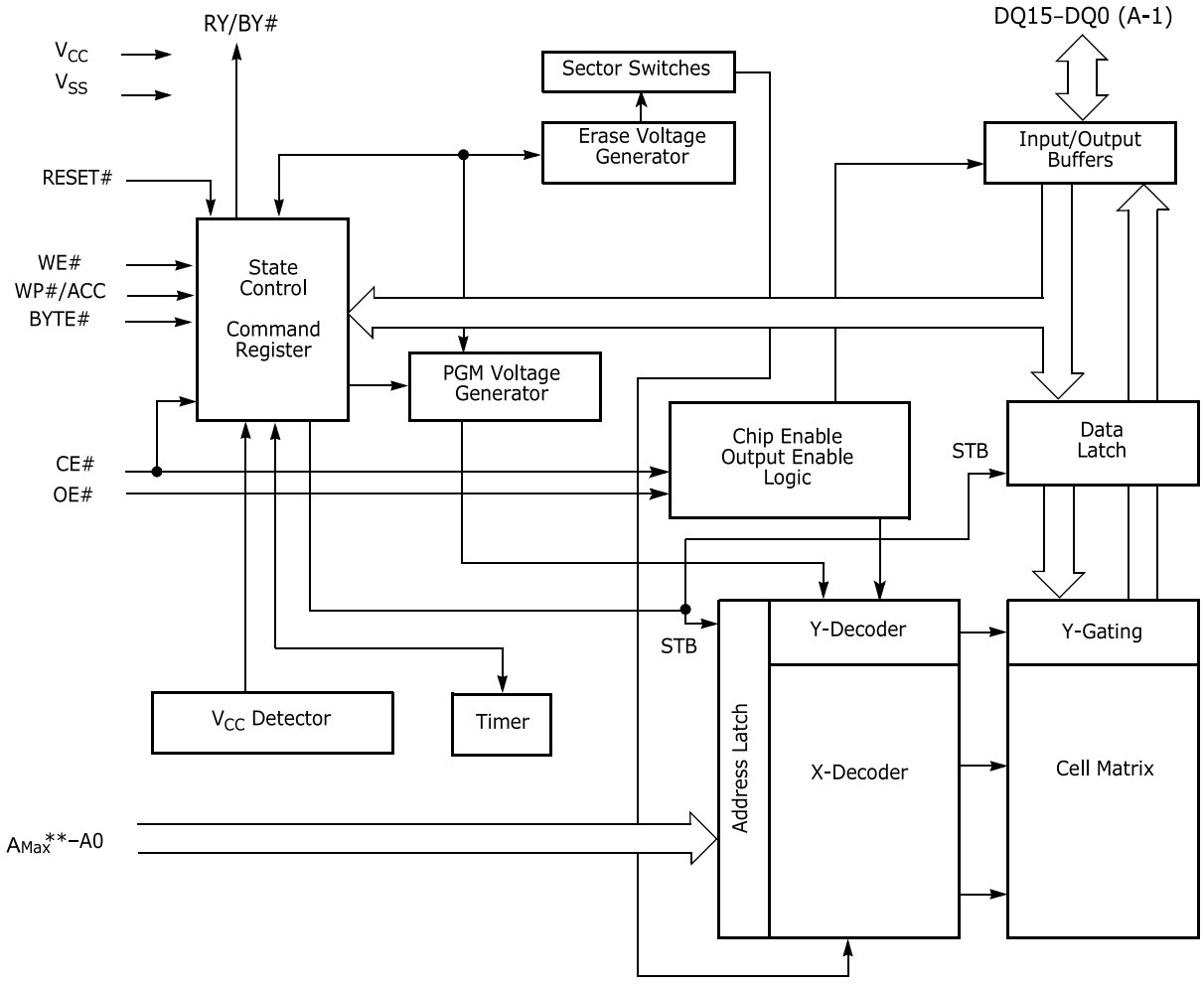
Je to sdílená operační paměť (pro procesor i hradlové pole) typu SDRAM. Toto zařízení může dle konfigurace obsahovat 32 až 128 MB paměti. jedná se o synchronní paměť která potřebuje k udržení uložených dat napájení. Na desce je tvořena pomocí 2 integrovaných obvodů. Jednotlivé obvody jsou voleny pomocí pinu CS (chip select). Každý integrovaný obvod obsahuje 4 banky v kterých používá 16 bitová slova. Je připojena na datovou a adresovou sběrnici která je sdílená jak pro procesor tak i pro hradlové pole. Je napájen pomocí napětí 3.3V. Princip paměti je velmi přehledně vidět na blokovém schématu. Více informací o zapojení v této aplikaci v [2] a podrobnější informace o samotné paměti [5].



Obrázek 1.5: Blokový diagram paměti SDRAM [5]

1.3.2 Paměť FLASH

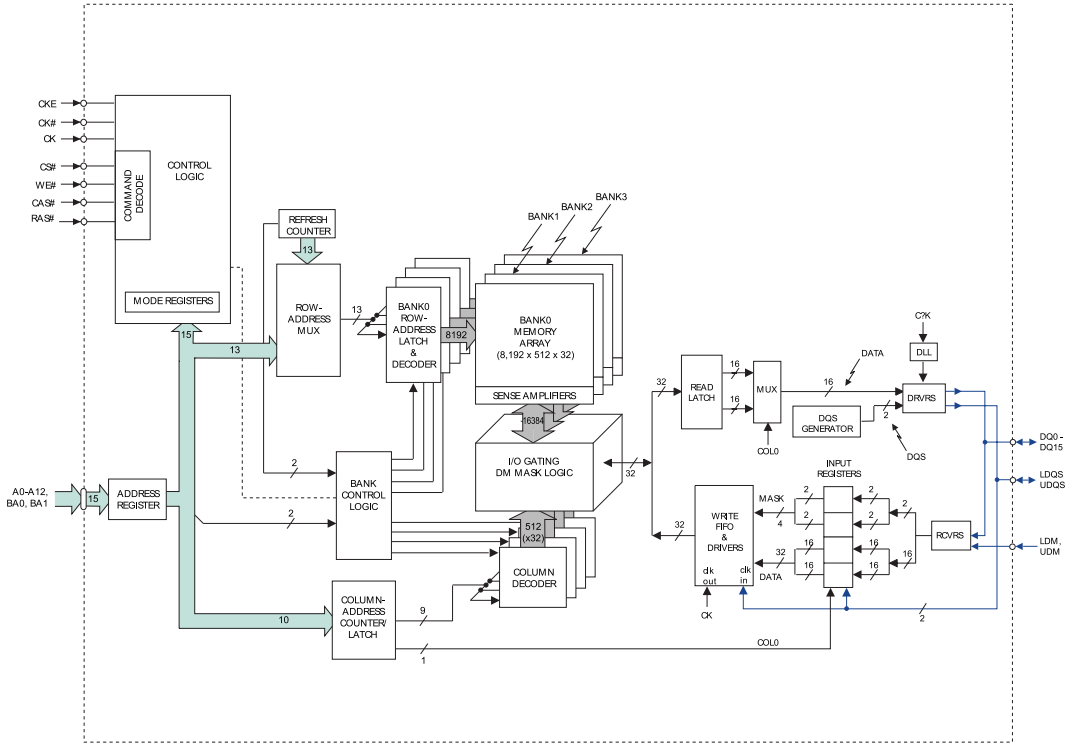
Je to taktéž sdílená paměť typu FLASH sestavená z klopných obvodů typu NOR. Tato paměť slouží k uložení programu jak pro procesor tak pro hradlové pole. Jedná se samozřejmě o paměť která udrží informaci i bez napájení. Obvod je napájen pomocí napájecího napětí 3.3V. Data tady jsou uložena v sektorech kde v každém sektoru je uloženo datové slovo, každé datové slovo má délku dle použité paměti má sběrnici širokou buď 8 nebo 16 bit. Princip funkce je také jasně vidět z blokového diagramu na blokovém schematu. Více informací o zapojení v této aplikaci v [2] a podrobnější informace o samotné paměti [6].



Obrázek 1.6: Blokový diagram paměti FLASH [6]

1.3.3 Paměť DDR SDRAM

Jedná se o rychlou paměť o velokosti 64 MB vyhrazenou pouze pro hradlové pole. Jedná jek už je z názvu patrné o paměť která potřebuje k uložení dat napájení. Paměť typu DDR SDRAM je oproti paměti typu SDRAM 2x rychlejší, protože reaguje jak na náběžnou hranu tak na sestupnou hranu časovacího signálu. Přístupová doba je 2ns. Obvod je napájen napájecím napětím 2.5V. Data jsou v této paměti uloženy, podobně jako v paměti SDRAM, ve 4 bankách kde do každé banky se vejde 8 mega 16-ti bitových slov. Princip je vcelku přehledně vidět na blokovém schématu. Více informací o zapojení v této aplikaci v [2] a podrobnější informace o samotné paměti [4].



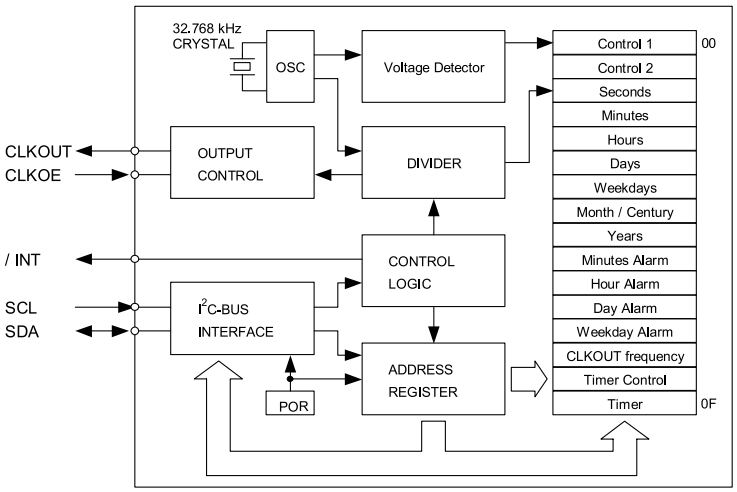
Obrázek 1.7: Blokový diagram paměti DDR SDRAM [4]

1.3.4 Hodiny reálného času, Mikrokontrolér MSP430, Napájení, Vstupně/Výstupní konektor

Zde jsou uvedeny souhrnně ostatní obvody. Zařízení také obsahuje obvod díky kterému má unikátní sériové číslo, které je nezměnitelné. Hradlové pole si nechávám na závěr a popíšu ho podrobně protože jsem na něm také vytvářel referenční příklady.

1.3.5 Hodiny reálného času

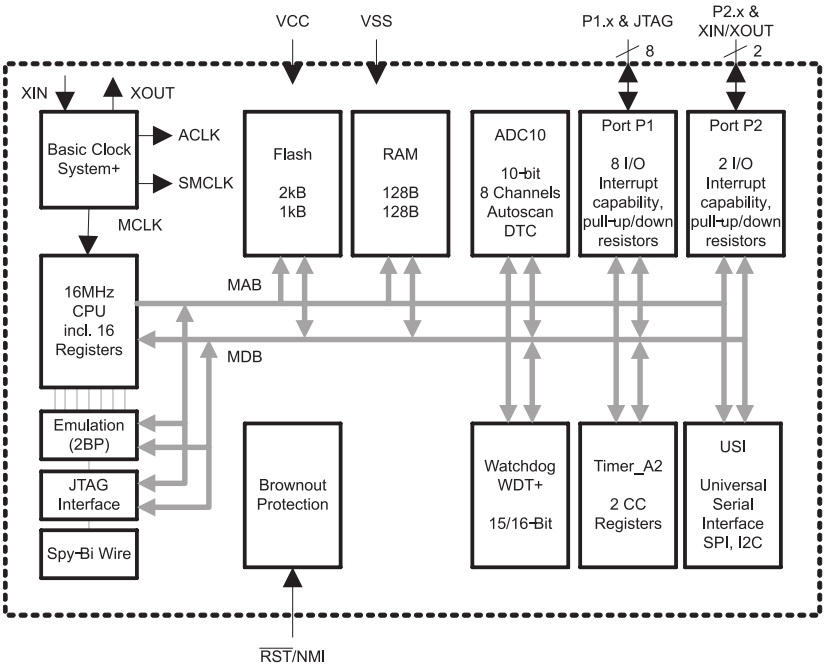
Jedná se o integrovaný obvod ve kterém je realizována funkce pro reálné hodiny (sekundy, minuty, hodiny, dny,..). Obvod je připojený přes sběrnici I_2C, která propojuje hlavní komponenty v tomto zařízení. Dále obsahuje funkce kalendáře, alarmu. Podrobněji je to vidět na blokovém schématu. Více informací v [7].



Obrázek 1.8: Blokové schema Hodin reálného času [7]

1.3.6 Mikrokontrolér MSP430

Tento procesor má v této desce hlavně funkci monitorování a řízení napájení a obsahuje i funkci Watchdogu. Jedná se o signálový procesor s nízkou spotřebou. Je připojený na napájecí napětí 3.3V jdoucí ale z obvodů mimo tuto desku a je schopen i běhu na baterii. Procesor je připojený na sběrnici I_2C, která, která jak již bylo řečeno, propojuje hlavní komponenty v tomto zařízení. Obsahuje 2KB Flash paměti pro uložení programu a 128B RAM paměti. Periferie je možno připojit přes 8-mi pinový I/O konektor. Dále je možné provádět debug programu pomocí JTAG rozhraní ([11]). Funkce je přehledně vidět na blokovém schématu. Více informací v [8].



Obrázek 1.9: Blokové schema Hodin reálného času [8]

1.3.7 Napájení

Jako hlavní napájecí napětí je zde 5V. Odběr tohoto zařízení je 700mA. Toto zařízení obsahuje několik úrovní napájecího napětí (3.3V, 2.5V, 1.5V, 1.2V a samozřejmě hlavní napájecí napětí 5V) kde některé okruhy lze dle zvoleného napájecího režimu vypnout. Zařízení obsahuje, jak již bylo řečeno, signálový procesor který se stará o řízení a monitorování napájení. Více informací lze vyčíst ze schématu [2].

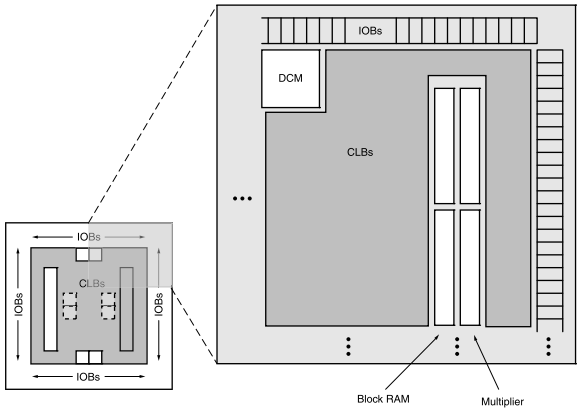
1.3.8 Vstupně/Výstupní konektory

Toto zařízení má všechny vstupně/vystupní obvody vyvedeny na dva 120-ti pinové konektory. Jeden z konektorů je vyhrazen převážně pro vstupně výstupní obvody hradlového pole. Na druhý jsou vyvedeny vlastně všechny rozhraní vystupující i vstupující ze nebo do zařízení. Je zde přivedeno i napájení. Více informací lze vyčíst ze schématu [2].

1.4 Hradlové pole FPGA Xilinx

Hradlové pole je vlastně zařízení, kde je možné pomocí softwarového programu, vytvořeného pomocí speciálního programovacího jazyka, vytvořit hardwarovou strukturu. V této aplikaci je použito hradlové pole FPGA od firmy Xilinx řady Spatran3E. Struktura FPGA , jak je možné vidět na Obr. 1.10, se skládá z těchto základních bloků:

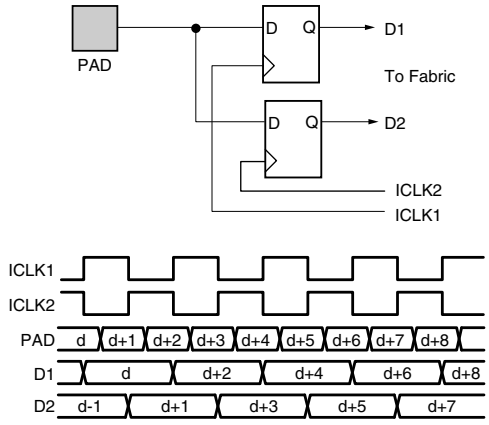
- Input/Output blok (IOBs)
- Configurable Logic Block (CLB) a tzv. Slice
- Blok RAM - úložný prostor o velikosti 516 kbit (pro model XC3S1200E)
- Multiplikátory(Multiplikators) - slouží k provádění binárního součinu dvou 18-ti bitových čísel.
- Digital clock manager (DCM)



Obrázek 1.10: Architektura FPGA [10]

1.4.1 Input/Output blok (IOBs)

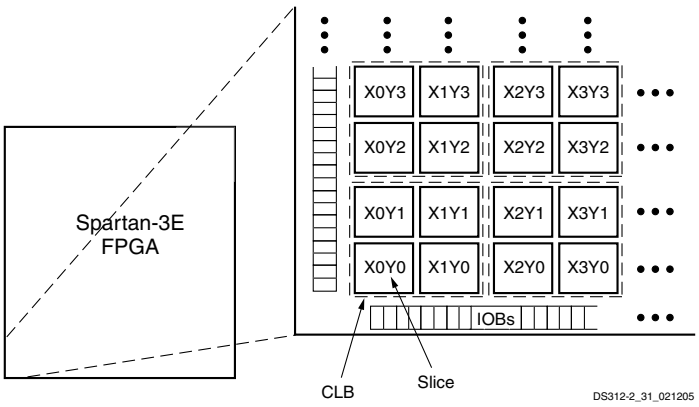
Tvoří to rozhraní mezi řídící logikou a výstupními piny. Jsou buď jednosměrně nebo obousměrně propustné. Obsahuje určité konfigurační možnosti. Je možné nastavit reakci na hranu při příchodu vstupního signálu a také jestli bude na výstup použit pull up či pull down rezistor. Obsahuje také klopný obvod typu D který lze použít buď jako Latch nebo jako paměťová buňka o velikosti 1b. IOB jsou organizovány po společných blocích (bankách), zde použitý typ Spartan 3E obsahuje 4 banky. Každý IOB obsahuje také ochranu svého vstupu proti výboji statické elektřiny. Speciální konfigurací je možné také dosáhnout DDR(double data rate) módu který nám zajitří reakci nejen na jednu ale na obě hrany vstupního signálu. K jeho použití je potřeba dvou klopných obvodů typu D a speciálně upravený multiplexor.



Obrázek 1.11: Blokové schema zapojení funkce DDR (obě metody) [10]

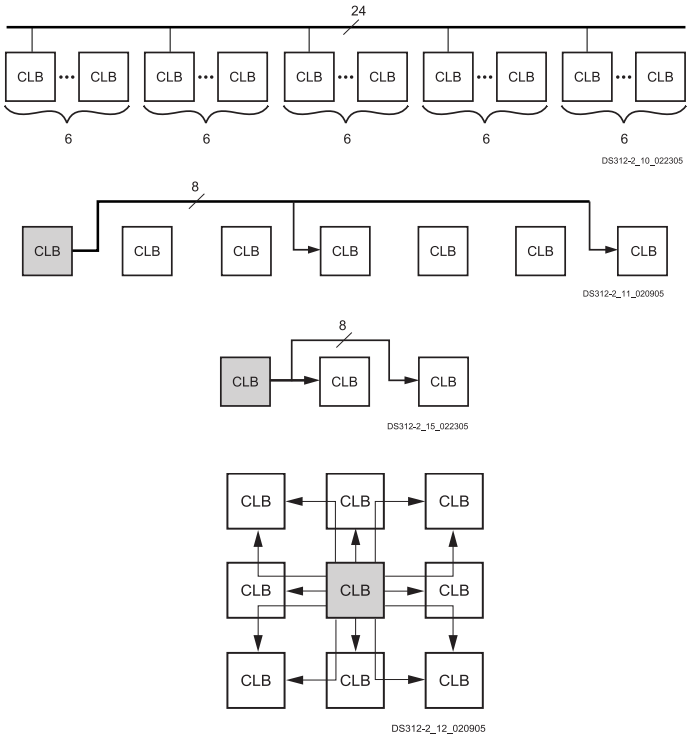
1.4.2 Configurable Logic Block (CLB) a tzv. Slice

CLB je hlavní logický prvek který nabízí implementaci jak sekvenčních tak kombinačních obvodů. Každý CLB obsahuje 4 tzv. slice a každý slices obsahuje 2 Look-Up Tabulky (LUT) jsou v ní vytvořeny logické a dva oddělené ukládací buňky, které mohou být použity jako flip-flop nebo latch. LUT může být použita jako paměť o velikosti 16x1 (RAM16) nebo jako 16-ti bitový posuvný registr (SLR16), přídavné multiplexory a aritmetické funkce. Volba a mapování slice v CLB se provádí automaticky při překladu.



Obrázek 1.12: Blokové schema zapojení Slice v CLB [10]

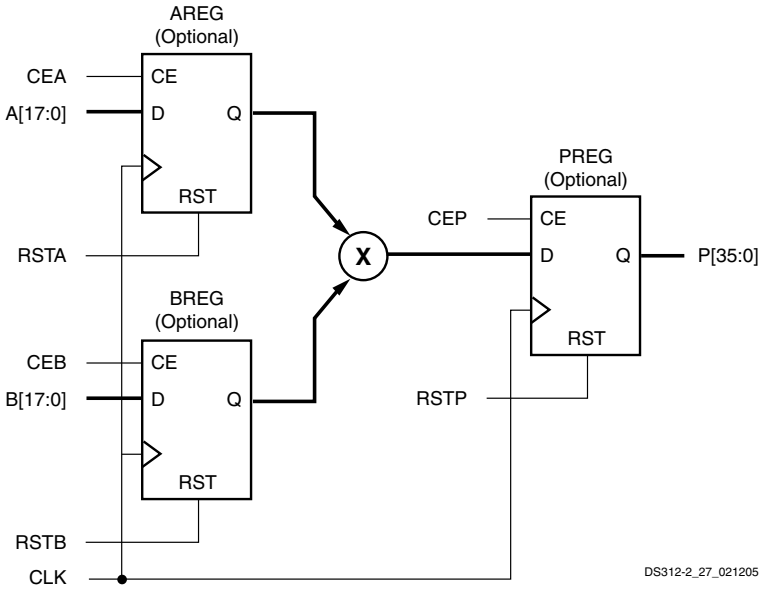
Dále obsahuje propojovací sítě, které zajišťují propojení mezi CLB navzájem, funkčními bloky a také vstupně výstupními obvody. Propojovací sítě se dělí na Long Line, Hex Line, Double Line a Direct Line jak je vidět na obrázku. Speciálním případem propojovací sítě jsou rozvody hodin. Je jich omezený počet a mají některé specifické vlastnosti (nízká kapacita, velmi malé rozdíly ve zpoždění signálů v různých částech chipu).



Obrázek 1.13: Způsoby propojení CLB (od vrchu Direct Line, Long, Hex a Double Line) [10]

1.4.3 Multiplikátory

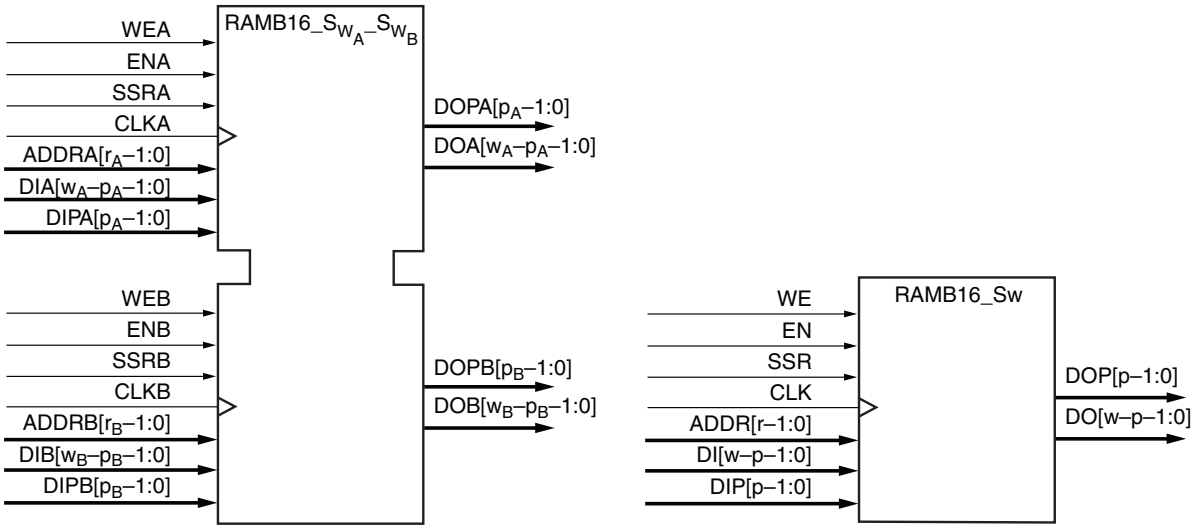
Jedná se o se o obvod který vykonává funkci násobičky. Tento obvod je tvořen jako násobička 2x18 bit.



Obrázek 1.14: Princip multiplikátoru [10]

1.4.4 Block RAM

Spartan 3E obsahuje obsahuje dle typu 4 až 36 oddělených bloků RAM. Tyto bloky jsou organizované jako dvou portové konfigurovatelné 18 Kbit velké bloky. Každý blok obsahuje 2 identitcké porty A a B s nezávislým přístupem. Blok RAM paměť můžeme využívat buď jako 2 portovou nebo jako jednoportovou. Každý z bloků obsahuje mimo obvyklých vstupů a výstupů oddělený vstup/výstup pro paritní data. Více informací o BlockRAM se dá najít v literatuře [22].

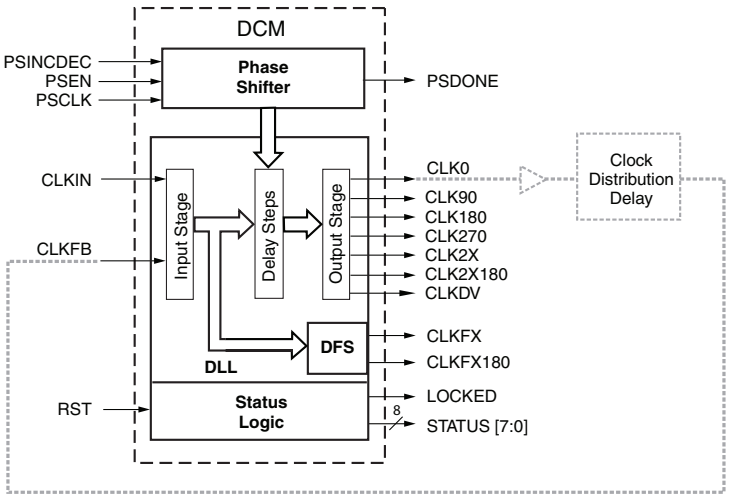


Obrázek 1.15: Dvouportová je jednoportová Block RAM [10]

1.4.5 Digital clock manager (DCM)

Jedná se o obvod starající se o kompletní kontrolu nad časováním, fází a zkreslením signálu. Zkládá se hlavně z těchto stavebních celků:

- Delay/Locked Loop (DLL)
- Digital Frequency Syntetizer (DFS)
- Phase Shifter (PS)
- Staus Logic



Obrázek 1.16: Blokové schema DCM [10]

Delay/Locked Loop (DLL)

Jedná se o obvody pro synchonizaci hodinového signálu. Umí za běhu pomocí nastavení zpoždění v cestě hodinového signálu dosáhnout stravu, kdy hodinový signál odrazí na vstupy klopných obvodů přesně o periodu později než na vstup FPGA. Obvod využívá 2 vstupní piny CLKIN na vstup hodinového signálu a CLKFB (feed-back clock) - jedná se o pin kde je přiveden zpětnovazebně signál od CLKIN. DLL se používá především k těmto funkcím:

- vygenerovat hodinový signál o dvojnásobné frekvenci
- vydělit frekvenci jedním z koeficientů (viz datasheet k fpga str. 50, tab. 29)

Digital Frequency Syntetizer (DFS)

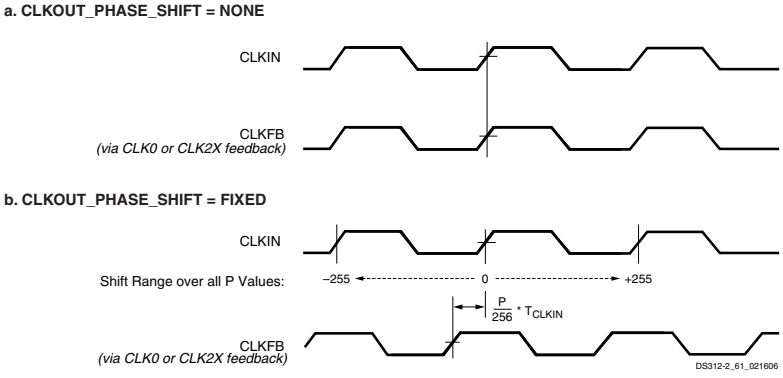
Jedná se obvod zajišťují generování signálů odvozených od vstupního signálu CLKIN. Generování se provádí nastavením násobiče (CLKFX_MULTIPLY) a děliče (CLKFX_DIVIDE). Obě hodnoty musí být typu integer v rozsahu (2-32). Lze použít jak s DLL tak i bez použití DLL.

$$f_{CLKFX} = f_{CLKIN}\frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE} \tag{1.1}$$

Phase Shifter (PS)

Jedná se o obvod který zvládne upravit vstupní časovací signál tak, že posune fázi vstupního signálu. Obvod zvládá fázové posunutí hodinového signálu o 90,180 a 270°. Ovládá též posunutí o předem danou velikost z rozsahu hodnot typu integer -255 až 255 - proměnná se jmenuje PHASESHIFT. Výpočet posunutí se počítá ze vzorce:

$$t_{PS} = (\frac{PHASESHIFT}{256}) * T_{CLKIN} \tag{1.2}$$



Obrázek 1.17: Ukázka funkce Phase Sifteru [10]

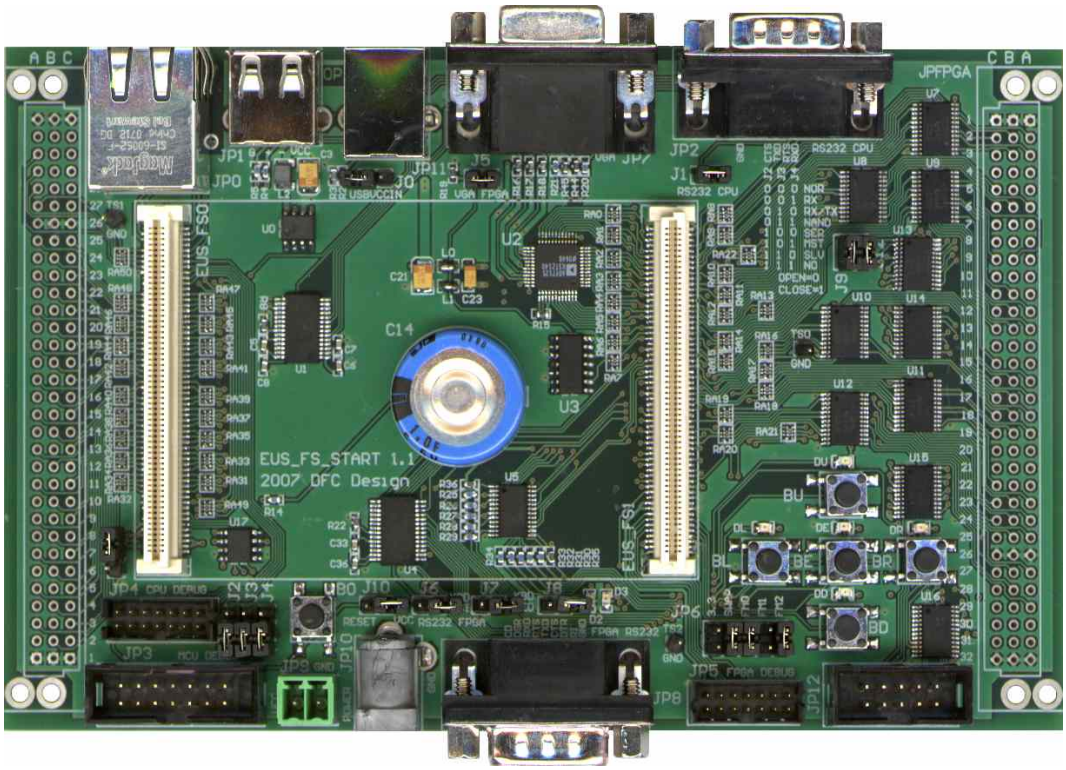
Status Logic

Jak už z názvu plyne jedná se o obvod který zajišťuje indikaci stavů DCM. Umožňuje i její reset. Indikace je realizována pomocí předem definovaných signálů. Více informací najdeme v datasheetu k hradlovému poly.

Informace z této kapitoly a další se dozvíme v [10], [15], .

1.5 Starter Kit

Jedná se o vývojovou desku k systému EUS FS. Tato deska obsahuje rozhraní pro připojení k řídícímu systému.



Obrázek 1.18: Foto Starter kitu z vrchní strany [1]

Obsahuje rozhraní:

- 2x 96 pinový I/O konektor
- 1x 10/100Mb Ethernet konektor
- 2x RS232 konektor (1x pro procesor, 1x pro FPGA)
- 2x JTAG rozhraní (1x pro procesor, 1x pro FPGA)
- 1x JTAG rozhraní pro MCU430
- 1x VGA analogový video konektor

- 1x USB host konektor
- 8kB FRAM I_2C paměť
- klávesnice s 5 tlačítky a 5 LED diod a RESET tlačítko
- 1F vysokokapacitní kondenzátor pro zálohu RTC
- Jedno napájecí napětí 5V

Více informací je na stránkách výrobce ([1]).

2 PICOBLAZE

Jedná se o jednoduchý softcore procesor [1]. Je vytvořen firmou XILINX, kde po registraci je možné ho zdarma stáhnout. Tento procesor 8 bitový procesor s redukovaným instrukční sadou - RISC a otimalizováno pro hradlová pole řady Spartan 3 s podporou hradlových polí Virtex 5, Spartan 6 a Virtex 6. Jedná se v podstatě o jednoduchý procesor. Tato konstrukce obsazuje pouze 96 CLB bloků což je dle typu pole od 0,3% až do 12,5% kapacity pole. V implementaci dodávané výrobcem umožňuje uložit do paměti až 1024 instrukcí, které jsou automaticky nahrávané do FPGA. Pamět je tvořena programově jako paměť typu ROM která je vytvořena v BlokRAM hradlového pole. Procesor dosahuje rychosti 44 az 100 MIPS.

Picoblaze se dodává v balíčku jako programový soubor buď v jazyce *VHDL* nebo *Verilog*, kde v balíčku se jedná o soubor s názvem *kcpsm3*. V tomto balíčku se mimo jiné dodává také ROM paměť, která se nazývá *ROM_form*. V tomto souboru jsou uloženy instrukce. Obsah paměti se vygeneruje z napsaného programu v assembleru buď za pomoci nástroje dodávaného přímo v tomto balíčku nebo pomocí programu pBlazIDE. Dále jsou v tomto balíčku ještě další makra potřebné pro UART komunikaci ale tímto se nebudu v této práci zabývat.

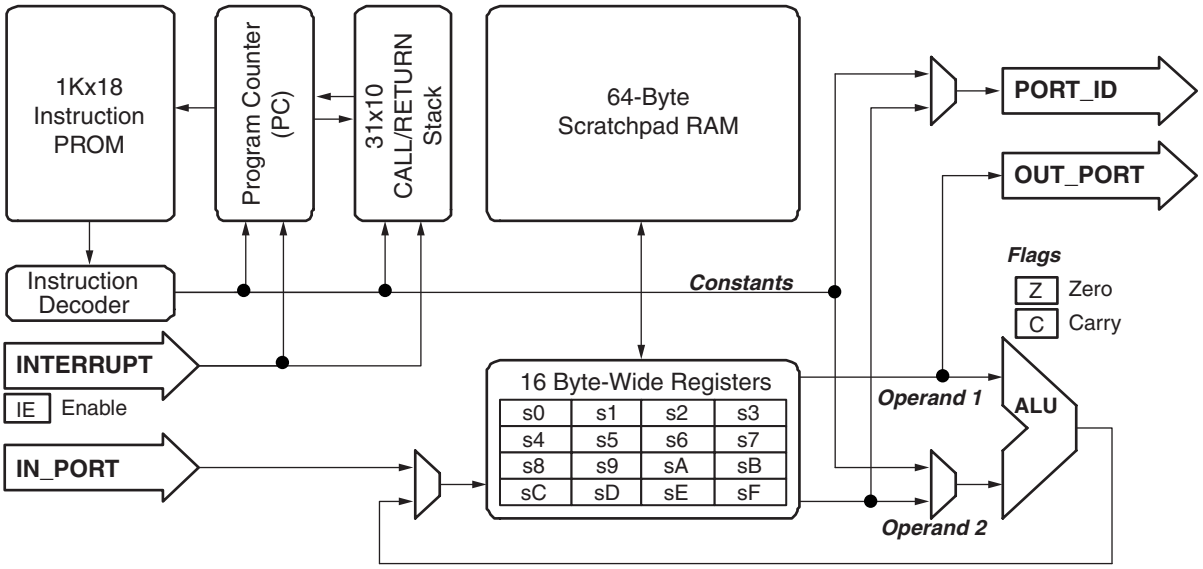
2.1 Vlastnosti procesoru

Přehled základních vlastností procesoru:

- 16. bitové datové registry
- paměť až na 1024 instrukcí
- ALU s příznakem CARRY a ZERO
- 64 byte interní zápisníkové RAM paměti
- až 256 vstupů a 256 výstupů
- zásobníková paměť o velikosti až 31 položek
- 2 takty CLK signálu na instrukci
- rychlá odpověď na přerušení (maximálně 5 taktů CLK signálu cyklů)
- optimalizováno pro Spartan 3 architekturu
- podpora instrukční sady Assembleru

Zde uvedené vlastnosti jsou pro zakladní balíček a pokud použijeme propojení dodávaných komponent podle návodu výrobce. Jejich modifikací můžeme dosáhnout jiných vlastností.

[1] je to typ procesoru který se nahraje do hradlového pole jako program, tj. není vytvořen na chipu



Obrázek 2.1: Blokové schéma Picoblaze

Hlavní registry Picoblaze obsahuje 16 registrů, kde každý má 16 bitů. Nazývají se s0 až sF. Je možné je přejmenovat pro lepší přehlednost programu. Žádný z těchto registrů není rezervovaný pro speciální úlohy a nemají priority. Procesor neobsahuje žádný accumulator, výsledek operací se ukládá do specifického registru.

Instrukční paměť Jedná se o paměť typu ROM až na 1024 instrukcí. Je to paměť vytvořená na FPGA pomocí BlokRAM. Každá instrukce pro Picoblaze je dlouhá 18 bitů. Program je vytvořen a nahrán do paměti ještě před kompilací designu, je tedy do pole nahrán společně s celým designem.

Aritmeticko-logická jednotka (ALU) Jednotka podporuje základní aritmetické operace pro sčítání a odčítání, logické operace AND, OR, XOR. Dále obsahuje funkce pro porovnávání výsledků operací a pro operaci testování bitů. Podporu je i operace pro rotaci a posun.

Příznaky Obsahuje příznaky pro operace ALU - ZERO a CARRY. Příznak ZERRO indikuje pokud je výsledek poslední operace nulový. Příznak CARRY značí různé podmínky v závislosti na poslední provedené instrukci. Příznakem INTERRUPT_ENABLE se povoluje přerušení.

Zápisníková paměť Picoblaze obsahuje 64 Byte zápisníkové RAM paměti. Je přímo i nepřímo adresovatelná. Obsah paměti se ukládá do určeného registru pomocí instrukcí *STORE* a *FETCH*. Paměť je možno přímo adresovat přímo adresou jako konstanta nebo nepřímo uložením adresy do jiného registru. Instrukce *STORE* slouží pro zapsaní obsahu některého z 16 registrů do nějaké pozice z 64 pozic v RAM. Instrukce *FETCH* slouží naopak pro čtení. Více v

Vstupně/výstupní port Vstupní a výstupní port je u Picoblaze 8 bitový s tím že *PORT_ID* nám slouží k adresaci jak vstupů tak i výstupů. Díky tomu že je *PORT_ID* 8 bitový tak může adresovat až 256 výstupních(výstupních) portů nebo kombinací. Při čtení nám procesor čte z *IN_PORT* a zapisuje na *OUT_PORT*. Během vstupně/výstupních operací se využívají registry s0 až sF do kterých se zapisuje nebo se z nich čte.

Programový čítač Obsahuje 10 bitový programový čítač - tj je možné do něho uložit maximálně 1024 instrukcí. S každou proběhlou instrukcí se inkrementuje. Není přímo ovlivnitelná programem, ale je částečně ovlivněna instrukcemi pro volání podprogramu, skoku a v případech skoku a přerušení. Po přetečení se čítač vynuluje.

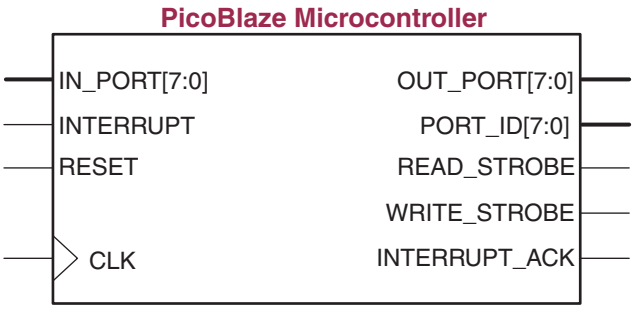
Program řízení toku Program umožňuje podmíněné a nepodmíněné řízení toku. Zajišťují se instrukcí skoku, instrukcemi volání podprogramu, návratu a vyvoláním přerušení. Více v

CALL/RETURN Stack Obsahuje až 31 úrovní podprogramu při přivedení přerušení, kde poslední úroveň je rezervována pro uložení povolení přerušení. Je implementován jako oddělený cyklický buffer. Při přetečení bufferu se přepíše nejstarší hodnota.

Přerušení Picoblaze obsahuje mimo jiné i vstup pro přerušení které se vyvolává asynchronně ze svého vstupu. Procesor odpoví na přerušení maximálně za 5 taktů CLK signálu.

Reset Pokud přijde reset tak se procesor nastaví do počátečního stavu. Po dobu resetu (tj. po dobu co je v 1) tak procesor stojí.

2.2 Popis vstupně/výstupních signálů



Obrázek 2.2: Vstupně/výstupní signály na Picoblaze

2.2.1 Vstupní signály

IN_PORT Jedná se 8. bitový datový vstup kde data jsou snímány s příchozí náběžnou hranou časovacího signálu.

INTERRUPT Tímto signálem určujeme přerušení přicházející na procesor. Přerušení na procesor přijde pokud je povolené příznakem *INTERRUPT_ENABLE* a je tento signál vystaven minimálně 2 takty CLK signálu. Jinak není přerušení přijato.

RESET Když je tento signál v log. 1 s náběžnou hranou CLK signálu tak se vyresetuje procesor do počátečních podmínek. Je automatiky generován při nahrání do FPGA.

CLK Jedná se o časovací vstup tím pádem i jako synchronizační signál kde se řídí náběžnou hranou signálu.

PORT_ID Jedná se o 8. bitový adresový výstup sloužící jako adresa pro *IN_PORT* a *OUT_PORT*.

OUT_PORT Jedná se o 8. bitový výstupní port kde data vystaví pro 2 takty CLK signálu kde se zároveň nastaví na log. 1 i signál *WRITE_STROBE*, kde tento signál je v log 1 po jeden takt CLK signálu. a nastaví se příslušná adresa na *PORT_ID*.

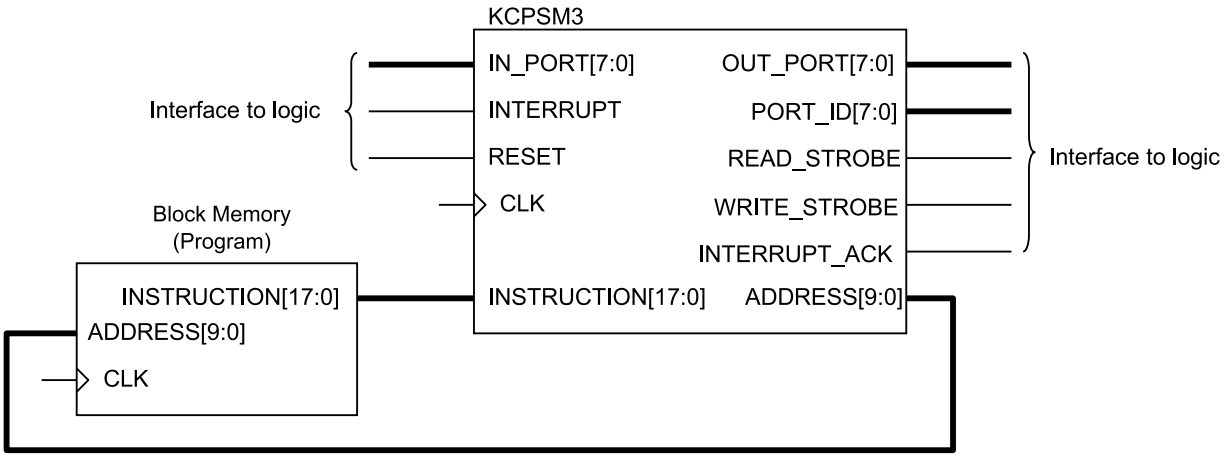
READ_STROBE Jedná se o signál který indikuje že jsou data čtena z *IN_PORT*u je nastaven na log.1 po dobu trvání log 1 jednoho taktu CLK signálu.

WRITE_STROBE Jedná se o signál který indikuje že jsou data zapisována na *OUT_PORT* je nastaven na log.1 po dobu trvání log 1 jednoho taktu CLK signálu.

INTERRUPT_ACK Tento signál nám ukazuje v průběhu druhého CLK cyklu že nám přerušení na procesor přišlo a je aktivní.

2.3 Základní připojení Picoblaze do designu

Základní propojení si ukážeme později v praktickém příkladě. V této kapitole ale ukážu co je nezbytně nutné udělat abychom mohli vůbec samotný Picoblaze zprovoznit. Nejedná se o nijak složité řešení, jak je vidět na obrázku.

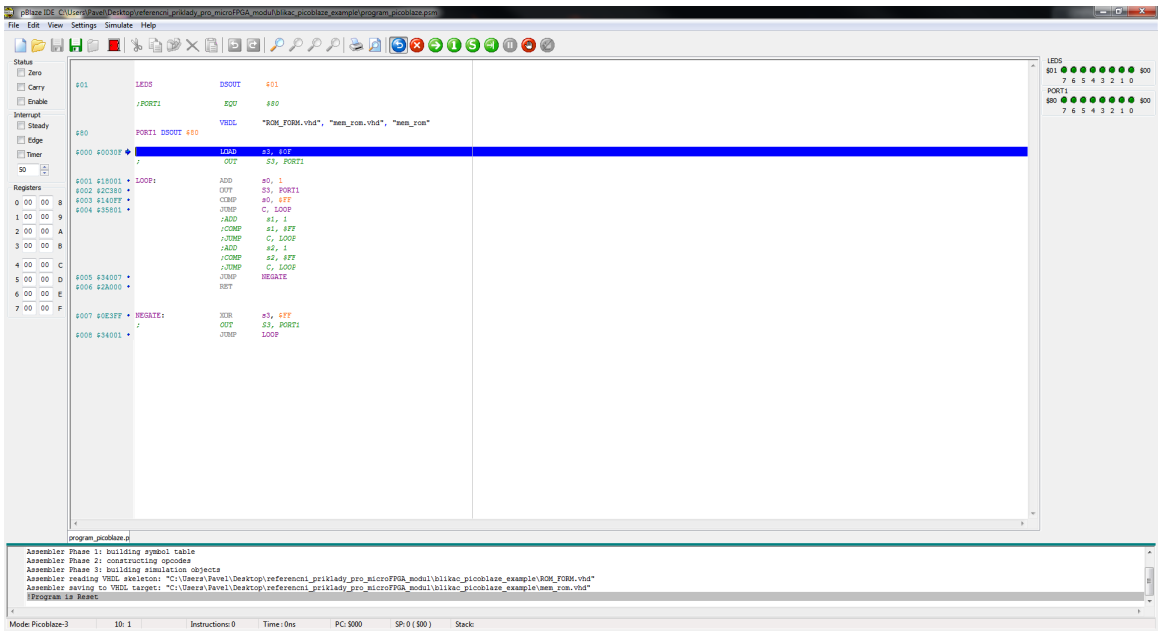


Obrázek 2.3: Základní zapojení Picoblaze s pamětí [18]

Ke správné funkci potřebujeme tedy pouze připojit paměť s instrukcemi a signál časovací signál CLK. Více informací nalezneme v [17]

2.4 Programování procesoru

Programování procesoru je řešeno pomocí symbolického jazyka Assembler. Výrobce v balíčku dodává překladač a nástroj pro nahrání programu do přiloženého souboru kde je za pomocí BlockRAM vytvořena struktura do které se data automatiky vloží. Konkrétní postup a způsob práce s ním najdeme v balíčku s Picoblazem viz. [18], [20]Mě osobně se ale pro práci s Picoblazem nejlépe osvědčil program *pBlazIDE*.



Obrázek 2.4: Ilustrační obrázek obrazovky v pBlazIDE

Jedná se o program zdarma dostupný na stránkách výrobce [19]. Je to kompletní vývojový nástroj kde nám nabízí nástroje pro samotné naprogramování, simulaci a následné vytvoření a nahrání programu do dodaného vzoru paměti. Přesný postup bude ukázáno v příkladě blikače ve čtvrté kapitole.

3 MOŽNOSTI PŘIPOJENÍ SYSTÉMU EUSFS

V této krátké kapitole bych rád čtenáře seznámil s možností připojení zařízení k PC, společné komunikaci a se základní prací s Linuxem (budou zde uvedeny pouze příklady potřebné pro pozdější využití v příkladech později v této práci pokud čtenář potřebuje i jiné potřebné věci, dají se bez problémů dohledat na internetu).

V této kapitole se z hlediska připojení budeme soustředit především na rozhraní později potřebné k práci s příklady tj. sériové rozhraní RS232 a Ethernet.

3.1 Sériové rozhraní

Toto rozhraní umožňuje sériovou komunikaci mezi zařízeními. Toto je rozhraní pro přenos informací vytvořené původně pro komunikaci dvou zařízení. Pro větší odolnost proti rušení je informace po propojovacích vodičích přenášena větším napětím, než je standardních 5 V. Přenos informací probíhá asynchronně, pomocí pevně nastavené přenosové rychlosti a synchronizace sestupnou hranou startovacího impulzu. Pro v9ce informací se můžeme podívat do literatury

Toto bylo jen několik málo slov o tomto rozhraní a nyní si ve zkratce ukážeme jak spolu budou tyto systémy komunikovat. K propojení u počítačů s tímto rozhraním provedeme pouze propojení příslušným kabelem. U počítačů neobsahujících toto rozhraní musíme sehnat převodník nejčastěji z rozhraní usb. Když máme propojeno tak musíme zvolit program v kterém budeme s tímto procesorem na kterém je Linux komunikovat. Ke komunikaci můžeme zvolit z velké škály programů např. Hyperterminal, Tera Term, a jiné. Po volbě programu už můžeme přistoupit k samotné komunikaci, zde po vybrání patřičného portu ještě musíme zvolit správné nastavení. Pro naše zařízení nastavíme **rychlost: 115 200 Bd, datové slovo 8bit, Bez parity, 1 STOP bit, žádné řízení toku (Flow control)**. Po té už máme správně nastavenou komunikaci. Teď už zbývá jen zapnout zařízení či, v případě že už zařízení běží, stisknout enter aby se nám objevila hlavička konsole. Více o sériovém rozhraní na [16]

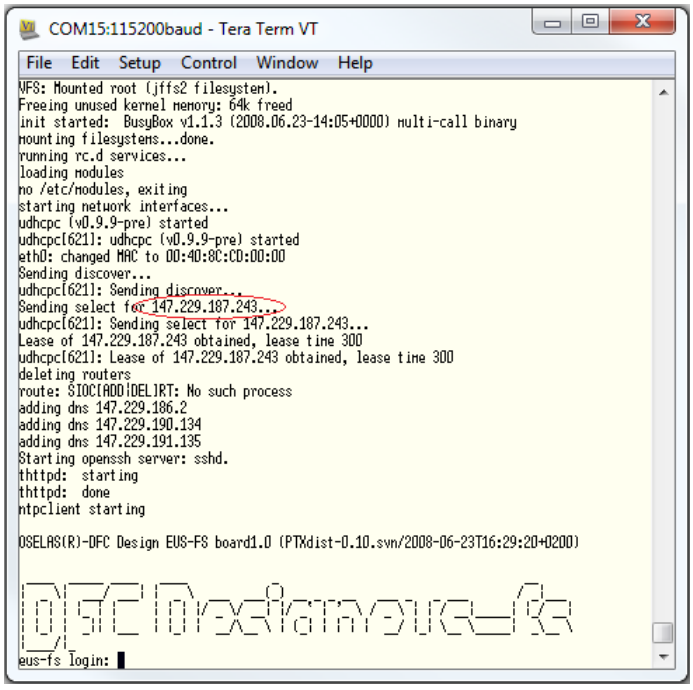
3.2 Rozhraní Ethernet

Jedná se o velmi rozšířené rozhraní kde je možné propojit velké množství zařízení kde každý v této síti má svoji unikátní adresu. Toto je i jeho nesporná výhoda.

Pro naše potřeby budeme uvažovat připojení do sítě, kde nám adresu přidělí DHCP server (pro opačný případ najdeme nastavení v [1]). Zařízení připojíme nejprve přes sériovou linku (viz. výše) a poté je třeba zjistit jakou máme přidělenou ip adresu. Adresu zjistíme po přihlášení jako root zadáním *ifconfig* do konzole. Ve vypsaném nás zajímá

hodnota u *inet addr* tato hodnota odpovídá IP adrese daného zařízení kterou budeme dále používat.

Tuto adresu lze zjistit také při nabíhání systému pokud ho máme připojen přes sériové rozhraní tak nám ji tam vypíše viz obr.



Obrázek 3.1: Ukázka ip adresy po naběhnutí zařízení

Dále budeme ethernet využívat hlavně kvůli přenosu souborů na zařízení k čemuž je dle mého názoru nejlepší použít program WinScp.

4 ÚVOD DO JAZYKA VHDL A VYTVOŘENÉ REFERENČNÍ PŘÍKLADY

Jedná se o jazyk hojně používaný k návrhu hardwarových struktur v hradlových polích typu CPLD a FPGA. Je definováno mezinárodním standardem IEEE 1076. Umožňuje návrh jak logických tak i sekvenčních struktur. Jeho největší výhodou je jeho univerzálnost. Konečná implementace navržené struktury. Konečná implementace navržené struktury je závislá až na kompilaci VHDL kódu. Z tohoto vyplývá, že pomocí tohoto jazyka lze provádět návrhy pro hradlová pole různých typů a výrobců vše je závislé až na použitém kompilátoru VHDL kódu, který je dostupný na stránce výrobce obvodu. Návrh strukturu je možné (vhodné) rozdělit na samostatné bloky které jsou následně pospojovány ve finální modul. Tento princip je vlastně analogický s tvorbou elektrických obvodů. V této aplikaci máme použity obvody firmy XILINX. Z tohoto důvodu je použit vývojový nástroj také od tohoto výrobce - ISE WebPack ([12]). Jedná se o komplexní vývojový nástroj který obsahuje všechny potřebné komponenty pro vytváření, syntézu, simulaci a nahrání projektu. Více v [13]. Hlavní části tohoto programu jsou:

- Editory - jedná se součásti které slouží k vytvoření dané implementace. Obsahuje jak možnost grafického návrhu pomocí editoru schémat elektrických obvodů tak i možnost návrhu pomocí textového popisu v jazyce VHDL či Verilog. Pomocí textového popisu se navrhuje i simulační rozhraní.
- Syntetizér - jedná se o součást která se stará o překlad implementace vytvořené v editoru do jazykuv kterém to počítač dále zpracovává. Stará se též o částečnou optimalizaci návrhu. Převede také výsledek z editoru na kombinační a sekvenční obvodu použité v hradlovém poli.
- Implementace designu - jedná se o součást která vytváří strukturu už rovnou pro to určité pole.
- Simulátor - je to šikovný nástroj pro návrh obvodu kde si můžeme ještě před samotným nahráním ověřit funkčnost pomocí časových průběhů které jsou výsledkem simulace.
- Generátor programovacího souboru - jedná se o binární soubor s konfigurací který se rovnou nahrává na hradlové pole
- Komunikační program s hradlovým polem - jedná se o program Impact přes který se i nahrává vygenerovaná konfigurace

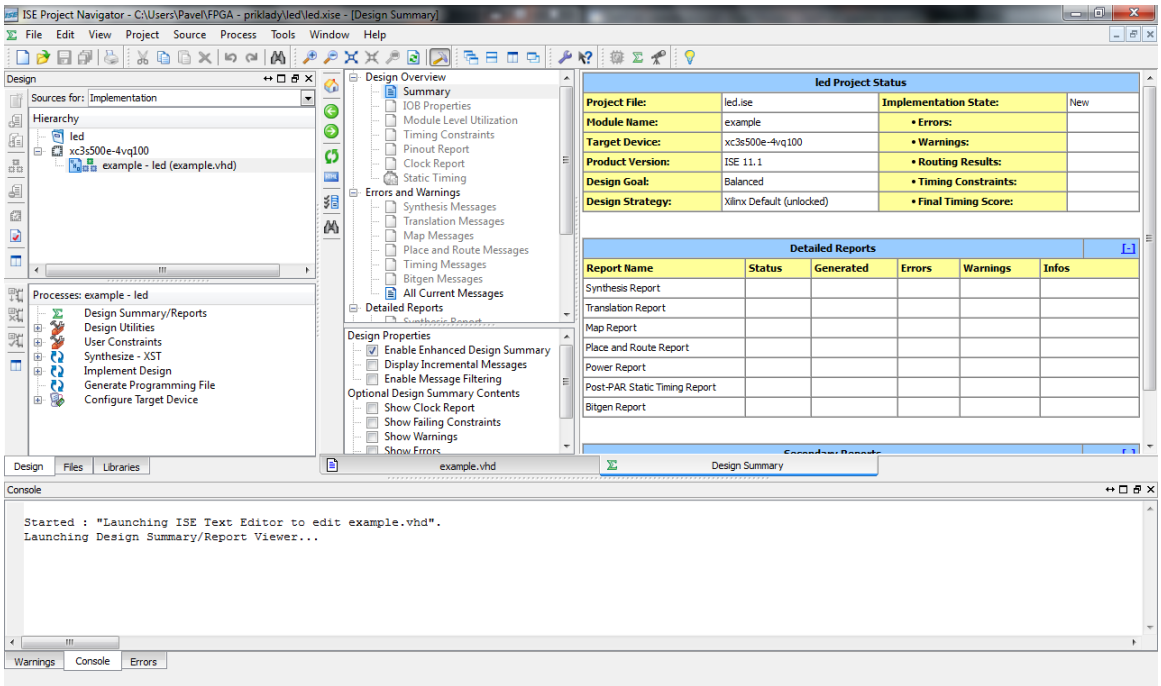
V této kapitole bude polopatě vysvětlena základní práce v jazyce VHDL, vytvoření základních struktur a pochopení jejich implementace.

4.1 Založení projektu a tvorba prvního programu

Celou dobu budeme pracovat ve vývojovém prostředí ISE WebPack 11 od firmy Xilinx ([12]).

4.1.1 Založení projektu

Po spuštění programu založíme nový projekt *File - New Projekt*. Zde si zvolíme jméno a umístění složky s projektem. V dalším kroku zvolíme použité Field Programmable Gate Arrays – na místě programovatelné logické obvody (FPGA) a zvolíme si jazyk psaní (v našem případě VHDL). V následujícím kroku zvolíme *New Source* zde zvolíme *VHDL Module* a napíšeme název a dáme next dokončíme oba průvodce. Zobrazí se nám něco takového:



Obrázek 4.1: Základní obrazovka po založení projektu

4.1.2 První program

Zde i popíšeme první program pro rozsvícení led diody a podíváme se co se nám vlastně ukáže po otevření vytvořeného souboru. Tvorba projektu se skládá z jednotlivých konstrukcí které se pak v hlavním popropojují a naváží na vstupu a výstupy. V každé konstrukci je nutné aby byla entita ve které jsou definovány všechny vstupy a výstupu z/do této konstrukce a architektura kde je popsáno jak se daná konstrukce chová. Pro začátek bych chtěl říci, jestli nebude čtenáři něco z tohoto jazyka jasné dá se to najít v [13] nebo [14] nebo jinde na internetu.

```
-- toto jsou použité knihovny anlogie #include v C
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Zde je deklaracni cast entity-definují se vstupy
-- a výstupy tohoto souboru
entity example is

end example;


architecture led of example is -- Zde je popis architektury
                               -- konstrukce
-- Zde se deklaruji vnitrni signaly a komponenty
begin


end led;
```

Zde abychom dosáhli rozsvícení diody tak musíme do entity přidat řádek

```
Port ( X : out  STD_LOGIC_VECTOR (27 downto 0));
```

což nám říká že to bude výstupní port s názvem LED_out a bude typu STD_LOGIC což znamená že bude nabývat puze 0 nebo 1. Dále musíme přidat do architektury (za begin)

```
LED_out <= '1';
```

z čehož je jasně vidět že tímto říkáme přiřaď do LED_out logickou 1. Všimněme si že v jazyce VHDL se nepoužívá znak = pro přiřazení signálů a práci s nimi.

4.1.3 Implementation Constraint File (UCF)

Založíme ho tak že klikneme pravym tlačítkem myši do okna Hierarchy a zvolíme *New Source*, v dalším kroku si vybereme *Implementation Constraint File* a napíšeme jméno a dokončíme průvodce. Tímto krokem jsme vytvořili soubor, kde navážeme jednotlivé proměnné z hlavního souboru na fyzické piny zařízení a případně tam můžeme doplnit některé volby. Tento soubor bude pro náš případ vypadat takto:

```
NET "LED_out" LOC = "A11";
```

Toto nám říká že výstup `LED_out` bude navázán a pin `A11`.

4.1.4 Vytvoření a nahrání designu do FPGA

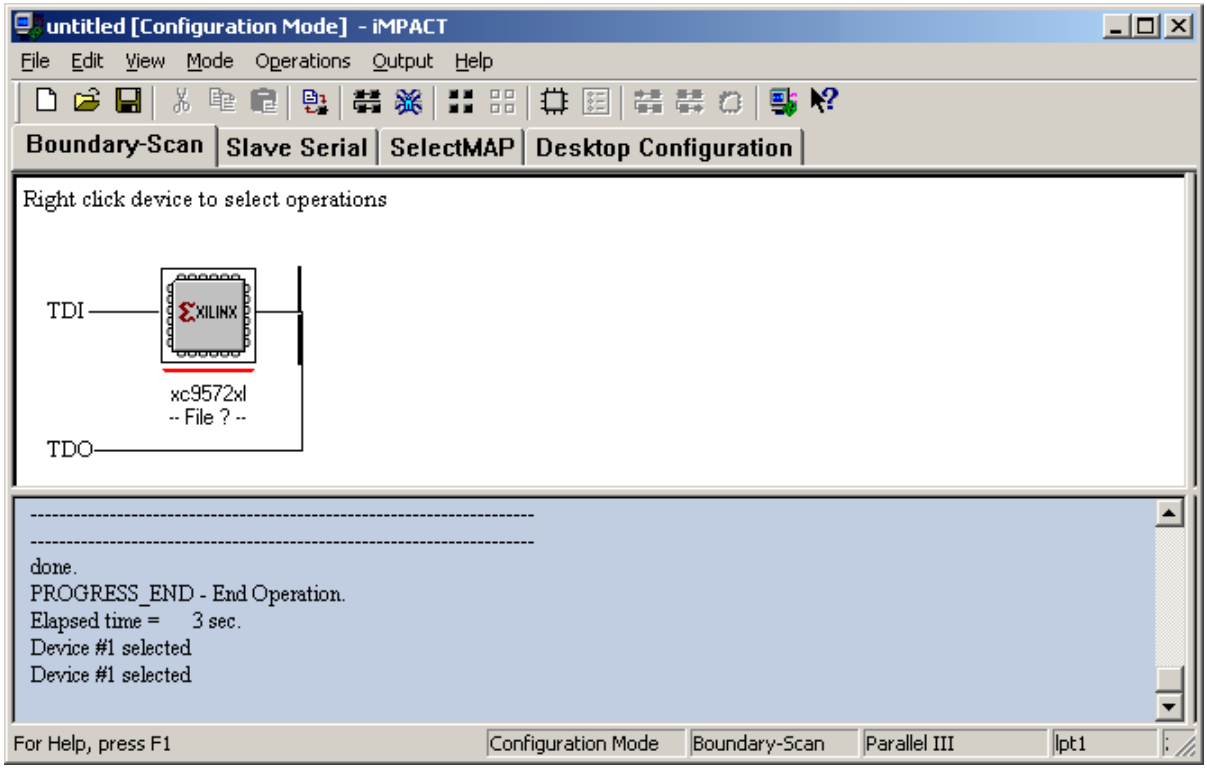
Existují dvě možnosti jak nahrát program do hradlového pole. Jednak můžeme nahrát program přímo do FPGA pomocí JTAG programátoru nebo pomocí vygenerovaného *mcs* souboru který nahrajeme do linuxového procesoru, kde poté tento program nahrajeme do FPGA.

Nahrání designu přímo do FPGA pomocí programátoru

Nejdříve musíme vytvořit programovací soubor (*.bit). Ten vytvoříme tak, že v okně Hierarchy zvolíme nejvyšší programovací soubor jen na něj jednou klikneme aby se jen zabarvil a dáme *Generate Programming File* po tomto kroku se nám ve složce s projektem objeví soubor s binární soubor s projektem.

Dále musíme nahrát vytvořenou konfiguraci do FPGA k tomuto účelu slouží program který si nainstalujeme spolu s ISE a jmenuje se iMPACT. Pro naprogramování musíme FPGA připojit pomocí JTAG rozhraní přes programátor k počítači a samozřejmě připojit napájení. Pokud je vše správně připojeno svítí na programátoru zelená dioda. Poté spustíme iMPACT a provedeme Boundary scan. Po tomto kroku by se nám mělo zobrazit použité zařízení (viz obr 4.2). Klikneme na něj 2x a zvolíme soubor s projektem a nahrajeme ho. Klikneme pravím tlačítkem myši na zařízení a dáme *Program* - po tomto kroku by se měla konfigurace nahrát do FPGA a u našeho modelového projektu rozsvítit LED dioda.

Jako programátor je možné použít (a je použit i v našem případě) např. XILINX Platform cable usb ke kterému se nám do Windows nahrají ovladače už při instalaci ISE WebPack.



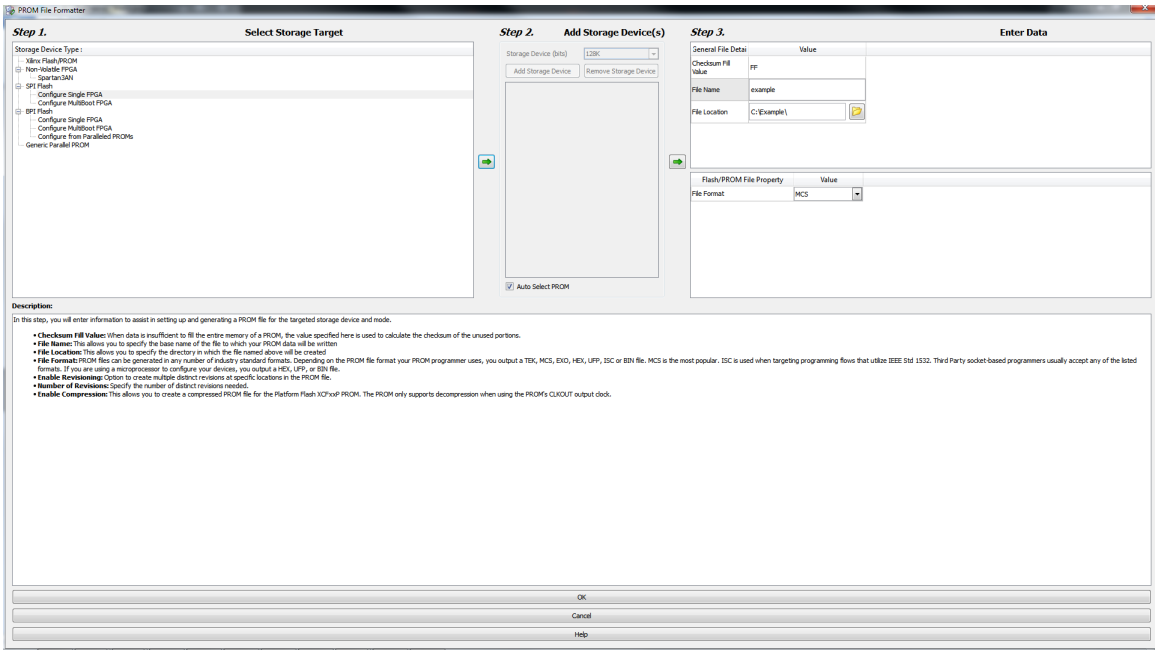
Obrázek 4.2: Obrazovka v programu iMPACT po připojení zařízení (jen pro ilustraci)

Nahrání designu do FPGA za pomoci procesoru

Zde budeme vycházet z předchozí části kde si opět vygenerujeme programovací soubor (*.bit). V dalším kroku musíme vytvořit samotný *(.mcs)* soubor. Ten vytvoříme tam že si buď v iMPACT založíme nový projekt a zde poté zvolíme *Prepare a PROM File* nebo u již založeného projektu spustíme *PROM File Formatter* v levém okně. V dalším kroku nastavíme vše dle následujícího obrázku, kde v druhém kroku zaškrtneme *Auto Select PROM*. [1] Ve třetím kroku zvolíme cestu kam soubor uložit a jeho název a také typ výstupního souboru (mcs). Po té si vybereme vygenerovaný soubor (.bit) a vygenerujeme patřičný mcs - *Generate File* v levém okně (viz obr. 4.3).

Teď musíme nahrát vygenerovaný PROM File do procesoru a z něho do hradlového pole. Připojení zařízení a práce s ním je popsáno v předchozí kapitole. Kompletní příklad je také přiložený na CD.

[1] Ve verzi iMPACT 11.1 toto nefunguje korektně a proto budeme muset později tuto velikost upravit v nabídce na objektu SPI PROM - Edit PROM - Modify PROM kde nastavíme patřičnou, kde paměť která je potřeba je uvedena v okně s obsazením paměti dole červeným písmem



Obrázek 4.3: Obrazovka v programu iMPACT pro generování mcs souborů

4.2 Synchronní blikání LED diody

Jako další příklad si uděláme synchronní blikání diody s frekvencí cca 1 Hz. Zde si ukážeme jak to udělat abychom to měli zároveň s časovacím signálem a také jeho využití. Budeme vycházet z postupů z předchozí kapitoly i ze zdrojového kódu ukázaného na začátku kapitoly kde si ukážeme akorát co se kam bude doplňovat. Celý kód příkladu bude vidět v kompletním příkladu na přiloženém CD.

4.2.1 Teoretická úvaha řešení

My máme použit oscilátor s frekvencí 100 MHz a potřebujeme vytvořit takovou děličku abychom na výstupu dosáhli frekvenci přibližně 1 Hz. Toho dosáhneme tak že si uděláme vektor hodnot a tam nám vlastně vznikne dělička hodnot pro nulté místo je to dělička 2 a první je to 4 a takto analogicky až po 2^{27}.A my na výstup použijeme pouze 27 prvek vektoru což je frekvence přibližně 0.37 Hz což znamená že výstup změní stav každých 0.74 sekundy. Pro ilustraci je to ukázáno na pravdivostní tabulce se třemi prvky, kde kdybychom použili prvek a_2 tak dostaneme děličku 8:

Tabulka 4.1: Příklad pravdivostní tabulky pro dělič s vektorem o délce 3

a_2	a_1	a_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Pro synchronní blikání použijeme jednoduchou věc a to, že budeme řídit pomocí náběžné hrany signálu k čemuž nám stačí jednoduchý řádek kódu který je vidět v další podkapitole.

Zdrojový kód k blikání LED diody

Jak již jsem napsal dříve budeme vycházet z kapitoly První program a konkrétně z úvodního - zatím nedoplněného kódu. Jako první musíme samozřejmě doplnit vstupy a výstupy do entity. Zde budeme používat výstup ale jako vektor kde ale, jak bude ukázáno později v ucf, budeme používat pouze nejvyšší bit. Takže do entity doplníme:

```
    Port ( SYSCLK : in  STD_LOGIC;
           X : out  STD_LOGIC_VECTOR (27 downto 0));
```

Zde je jasně vidět že používáme jako vstup používáme časovací signál s názvem SYSCLK a dále tam máme bitový vektor X o délce 27 prvků ale v UCF souboru je namapován na fyzický pin pouze 27 bit.

Dále vytvoříme samotnou architekturu. Zde si vytvoříme pomocný signál který do ní vložíme (ještě před begin) a vynulujeme ho. A v samotné architektuře vložíme podmínku aby reagoval pouze na náběžnou hranu signálu.

```
architecture Behavioral of blikani_example is

signal temp: std_logic_vector(27 downto 0)
                                        := (others=>'0');

begin

        process(SYSCLK)
        begin
                if SYSCLK'event and SYSCLK = '1' then
                        temp <= temp + 1;
                end if;
        end process;
        X <= temp;
end Behavioral;
```

Zde jsme použili nový prvek - process jedná, když to přirovnám k jazyku C, tak o nekonečný while cyklus. Takže v FPGA může být paralelně velké množství na sobě nezávislých procesů. Jako parametr do závorky zde dáváme proměnnou na kterou je některé část z procesu citlivá. Dále tu máme podmínku (if) který nám, spolu s podmínkou, zajišťuje reakci na náběžnou hranu signálu a dále je už jen inkrementace signálu temp.

UCF soubor tedy upravíme na:

```
#CLK
NET "SYSCLK"      LOC = "C9";

#LED
NET "X<27>"         LOC = "A11";
```

Jak je vidět musíme ještě doplnit do UCF souboru časovací signál SYSCLK připojený na pin C9. Dále ještě X27 značí že je 27 prvek vektoru sběrnice připojený na pin A11.

Výsledný zdrojový kód tohoto příkladu je na přiloženém CD.

4.3 Binární čítač

Jedná se o obyčejný čítač který nám v binární podobě přičítá po jedničce a je řešen stejným způsobem jak blikání LED diody v předchozím případě. Kód je tedy stejný jako u blikání LED diody ale s tím rozdílem že v UCF souboru jsem přidali další 4 LED diody

k výstupní sběrnici. To provedeme analogicky jako v předchozím případě. Kompletní příklad je také přiložený na CD.

4.4 Johnsonův čítač

Jedná se o kruhový čítač který posouvá postupně s každým příchozím signálem se nám překlopí a na výstup se nám posune. Vše je přehledně vidět na pravdivostní tabulce.

Tabulka 4.2: Pravdivostní tabulka Johnsonova čítače

posun	a_2	a_1	a_0
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	1	0
6	0	0	1
7	1	0	0
8	0	1	0

4.4.1 Teoretická úvaha řešení

Nejdříve si musíme vytvořit děličku kmitočtu abychom mohli na diodách vůbec lidským okem zpozorovat změnu stavu. To uděláme obdobným způsobem jako v případě blikání led diody ale jako výstup bude použita binární proměnná a bude sloužit jako povolení posunu. Dále jsem zde realizoval resetovací tlačítko to je řešeno pomocí podmínky, kde po jeho zmáčknutí se nám čítač vrátí do počátečního stavu.

Samotný posun je řešen pomocí ”přehození”prvního bitu vektoru na první místo.

Zdrojový kód pro Johsonův čítač

Budeme postupovat podobně jako v předchozím případě tj. vysvětlíme si jak dílčí části doplnit do základní šablony programu. Nejdříve začněme entitou. Zde budeme mít dva vstupní signály a jeden výstupní vektor. Signál SYSCLK jak je nám již známo je požit jako časovací, dále přidáme RST máme vstup pro reset od vnějšího tlačítka a vektor X o délce 5 je výstupní vektor na LED diody.

Dále zde máme signálové vektory, kde máme 2 pomocné, jeden dlouhý 27 bitů pro proces k děličce, další o délce 5 bitů k samotnému čítači a poslední signál - enable kterým povoluje posun čítače.

Dále je tu kód pro proces kde je realizována dělička je až na jednu drobnost stejná jako v případě blikače algoritmus je lehce pochopitelný proto ho nebudu dále rozebírat:

```
delic:process(SYSCLK)
        begin
        if SYSCLK'event and SYSCLK = '1' then
                tmp_delic <= tmp_delic +1;
                if tmp_delic(7) = '1' then
                        EN <= '1';
                        tmp_delic <= (others => '0');
                else
                        EN <= '0';
                end if;
        end if;
        end process;
```

Program pokračuje procesem pro samotný čítač.Zde také není v podstatě co popisovat, jen na čtvrtém řádku od konce je vidět to co jsem popisoval v teoretickém řešení že je vždy, pokud přijde povolení, poslední bit vektoru přesunut na první úroveň.

```
jon_cit:process(SYSCLK,RST)
        begin
                if SYSCLK'event and SYSCLK = '1' then
                        if RST = '1' then
                          tmp_cit <= "10000";
                        elsif EN = '1' then
                          tmp_cit <= tmp_cit(0)
                                  & tmp_cit(4 downto 1);
                        end if;
                end if;
        end process;
```

Do UCF souboru jsme přidali i ostatní diody (X) a vstup tlačítka takže jeho nynější podoba vypadá takto:

```
#CLK
NET "SYSCLK"        LOC = "C9";

#BUTTON
NET "RST"           LOC = "G10";
#LED
NET "X<4>"          LOC = "A11";
NET "X<3>"          LOC = "B13";
NET "X<2>"          LOC = "A13";
NET "X<1>"          LOC = "B14";
NET "X<0>"          LOC = "A14";
```

Kompletní příklad je také přiložený na CD.

4.5 BCD čítač

Binary Coded Decimal (zkráceně BCD, dvojkově reprezentované dekadické číslo) je způsob kódování celých čísel s využitím pouze desítkových číslic (0-9).

Vzhledem k tomu, že existuje šestnáct různých kombinací čtyř bitů, a desítkových číslic je jen deset, je šest kombinací nevyužito. V porovnání s hexadecimální soustavou, kde je pro každé čtyři bity využíváno všech šestnáct hodnot.

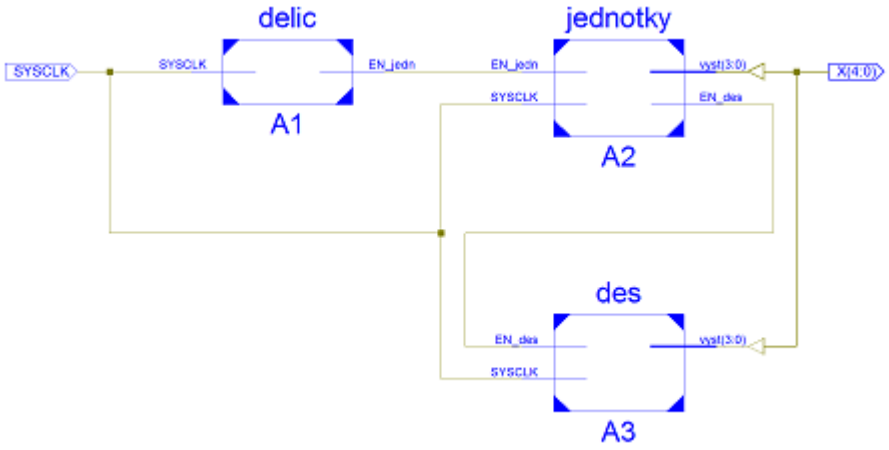
V našem případě budeme realizovat pouze BCD do 19, protože můžeme využít pouze 5 LED diod.

4.5.1 Teoretická úvaha řešení

V našem případě budeme používat 2 binární čítače jeden pro desítky a jeden pro jednotky s tím že jak již bylo řečeno v úvodu nebudou čítat až do přetečení ale musí být omezeny na určité hodnotě a pak opět vynulovány. Kvůli správnému zobrazení zde bude použita i předdělič. Vše je přehledně vidět na obrázku:

Zdrojový kód pro BCD čítač

Jak je již vidět z blokového schématu, je čítač tvořen čítačem desítek a jednotek a děličem. Rozebereme si jednotlivé části postupně začněme děličkou. Tu vytvoříme analogicky z předchozího příkladu.



Obrázek 4.4: Blokové zapojení BCD čítače

Jak je vidět dělička je principielně stejná jako v předchozím případě s tím že jejím výstupem je povolení čítání jednotek. Proto se s ní nebudu dále zabývat. A půjdeme na čítač jednotek:

```
entity jednotky is
    Port ( SYSCLK : in std_logic;
                        EN_jedn : in std_logic;
                        EN_des : out std_logic;
           vyst : out std_logic_vector(3 downto 0));
end jednotky;

architecture jednotky of jednotky is

signal jedn : std_logic_vector(3 downto 0)
                            := (others => '0');

begin
process(SYSCLK, EN_jedn)
begin

        if SYSCLK'event and SYSCLK = '1' then
                if EN_jedn = '1' then
                        jedn <= jedn + 1;
                        if jedn = "1001" then
          --podminka pro maximalni
```

```
                                    jedn <= "0000" --vynulovani
                                    EN_des <= '1'  -- povoleni
                                                   --citani desitek
                            else
                                    EN_des <= '0';
                            end if;

                    end if;
            end if;
end process;

vyst <= jedn;

end jednotky;
```

Zde, jak je vidět, se jedná v podstatě o binární čítač s omezením na hodnotě 1001 což je dekadicky 9. Čítač přičte jedničku při každém příchodu povolení čítání. Jakmile dosáhne maximální hodnoty tak se vynuluje a pošle povolení čítání desítek. Dále budeme pokračovat čítačem desítek.

Čítač desítek je prakticky stejný s tím rozdílem že zde čítáme pouze do jedné a na výstupu máme pouze hodnotu desítek, protože už nemusíme nic dále povolovat.

Hlavní - zasřešující soubor, slouží k provázání těchto dílčích souborů. V této entitě už definujeme přímo proměnné navázané na UCF soubor. Zde nejdříve musíme v architektuře popsat jednotlivé bloky které budeme provazovat, což je vlastně obsah entit jednotlivých bloků. Pro náš případ to vypadá takto:

```
component delic
          Port ( SYSCLK : in std_logic;
                                    EN_jedn : out std_logic);
end component;

component jednotky
    Port ( SYSCLK : in std_logic;
                                EN_jedn : in std_logic;
                                EN_des : out std_logic;
           vyst : out std_logic_vector(3 downto 0));
end component;

component des
```

```
    Port ( SYSCLK : in std_logic;
                                EN_des : in std_logic;
           vyst : out std_logic_vector(3 downto 0));
end component;
```

V dalším kroku si musíme vytvořit pomocné signály kterými budeme ty jednotlivé bloky propojovat:

```
signal jednot,desitky : std_logic;
signal tmp_vyst : std_logic_vector(7 downto 0);
```

A teď už nám zbývá samotné propojení které je zde vidět (pro ilustraci co s čím se propojuje jsem to přidal do komentářů s tím že je to i alternativní způsob zápisu portmap ve VHDL):

```
A1 : delic --pojemnovani bloku : nazev souboru
                    --s prislusnou casti
  port map(
                SYSCLK => SYSCLK,
                EN_jedn => jednot);

A2 : jednotky
        port map (
                SYSCLK => SYSCLK,
                jednot => EN_jedn,
                EN_des => desitky,
                vyst => tmp_vyst(3 downto 0));

A3 : des
        port map (
                SYSCLK => SYSCLK,
                desitky => EN_des,
                vyst => tmp_vyst(4));
```

Zde již máme hotový kompletní BCD čítač teď už jen zbývá poslat proměnnou tmp_vyst na výstup tj. přiřadit k proměnné X

UCF soubor je stejný jako v předchozím případě jen zde nemáme použit reset takže nevyužíváme vstup od tlačítka.

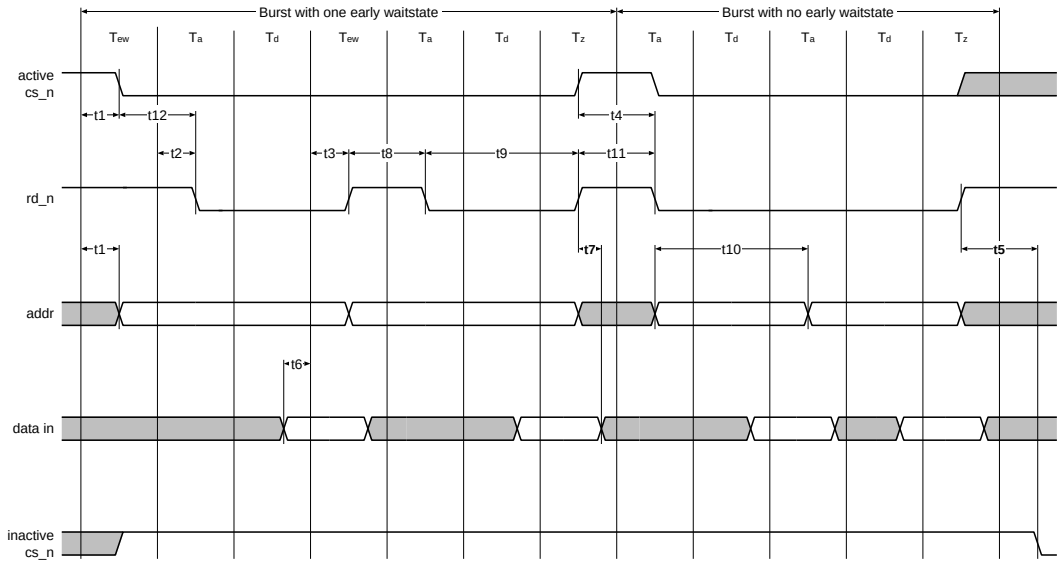
Kompletní příklad je také přiložený na CD.

4.6 Komunikace procesor - FPGA

V této části se budu zabývat už komunikací mezi FPGA a procesorem. V FPGA toto budeme implementovat jako 8 registrů s kterými budeme pracovat. Toto řešení bude tvořeno bez časovacího signálu.

4.6.1 Teoretická úvaha řešení

V hradlovém poli toto bude řešeno vytvořením 8 registrů a napojením na paměťovou sběrnici procesoru. V připojeném designu pro FPGA musíme také dodržet časování sběrnice pro čtení a pro zápis. To je vidět na následujících obrázcích:



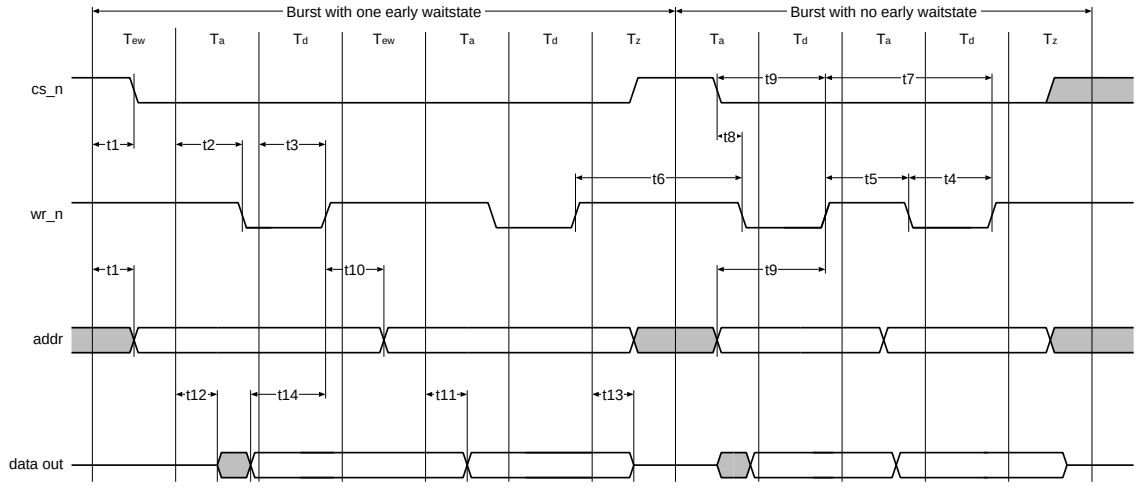
Obrázek 4.5: Časování sběrnice pro čtení procesorem na sběrnici [10]

Dále ještě musíme dodržet podmínku, že pokud zapisujeme z FPGA na sběrnici (probíhá čtení z procesoru) ale smíme mít data na sběrnici vystavena pouze po dobu aktivního signálu pro čtení jinak tam musíme poslat impedanci Z. Další věc co musíme ošetřit je že musíme napevno nastavit WAIT signál procesoru na log1. Z procesoru budeme se sběrnicí pracovat pomocí programu bus.

Programová část pro FPGA

Zde jak jsme si popsali dříve musíme dodržet časování sběrnice. Nejprve se podíváme na entitu. Zde budeme mít signály:

- WAITT - jedná se o výstupní signál namapovaný na wait signál procesoru, který musíme ošetřit



Obrázek 4.6: Časování sběrnice pro zápis procesorem na sběrnici [10]

- WR, RD jedná se o vstupní signály které nám značí povolení čtení a zápisu od procesoru
- ENABLE jedná se o vstupní signál chip select (volba chipu)
- ADRESS jedná se o vstupní 3 bitovou adresovou sběrnici
- DATA jedná se o 32. bitová datová sběrnice.

Toto máme v entitě. V V těchto řídících signálech je použita záporná logika (řídí se signálem log. úrovní 0). Jak vytvořit entitu již víme z předešlých příkladů. Dále si vytvoříme signály R1 až R8 které nám značí jednotlivé registry. Budou dlouhé 32 bitů a hned v definici je vynulujeme.

V architektuře vytvoříme blok pro čtení z procesoru (zápis na sběrnici).

```
DATA <=
   R1 when (ENABLE ='0' and RD ='0' and ADRESS = "000") else
   R2 when (ENABLE ='0' and RD ='0' and ADRESS = "001") else
   R3 when (ENABLE ='0' and RD ='0' and ADRESS = "010") else
   R4 when (ENABLE ='0' and RD ='0' and ADRESS = "011") else
   R5 when (ENABLE ='0' and RD ='0' and ADRESS = "100") else
   R6 when (ENABLE ='0' and RD ='0' and ADRESS = "101") else
   R7 when (ENABLE ='0' and RD ='0' and ADRESS = "110") else
   R8 when (ENABLE ='0' and RD ='0' and ADRESS = "111") else
   (others => 'Z');
```

Zde vidíme blok pro čtení kde se v podstatě jedná o podmíněné přiřazení signálů na datový výstup, kde podmínky jsou jasně vidět z závorkách. Protože se jedná o řízení zápornou logikou proto máme signály RD a ENABLE rovny nule a v ADRESS máme

určenou adresu toho určitého registru. V závěru je vidět že pokud nemáme zvolen ani jeden registr tak se nám na datovou sběrnici vloží impedance.

Dále si ukážeme blok pro zápis z procesoru (čtení v FPGA):

```
if ENABLE = '0' and WR = '0'
        case ADRESS is
          when "000" => R1 <= DATA;
          when "001" => R2 <= DATA;
          when "010" => R3 <= DATA;
          when "011" => R4 <= DATA;
          when "100" => R5 <= DATA;
          when "101" => R6 <= DATA;
          when "110" => R7 <= DATA;
          when others => R8 <= DATA;
      end case;
       end if;
```

Jedná se o příkaz switch který je zapouzdřen spolu jedním příkazem if kde, jak je vidět, máme zase obdobně povolení od řídících signálů. Zde je vše jasně vidět tj. pokud nám přijde povolení a adresa tak se nám do příslušného registru uloží data.

Teď už nám zbývá jen ošetřit signál WAITT který pevně nastavíme na log. 1.

Ještě musíme vytvořit UCF soubor. Do něho musíme vložit všechny signály. Adresové signály jsou použity *A2, A3, A4*, kde A1 zůstane nevyužitý. Jednotlivé piny na které je to připojeno najdeme ve schématu,které se nachází na přiloženém CD nebo na [2]. Datovou sběrnici připojíme na datové vodiče. Jako řídící signály jsou použity: WAIT (P18), RD (C18), WR0(D17), CSP0 (ENABLE - V3), kde k ENABLE a WR ještě musíme doplnit řádek:

NET "ENABLE"CLOCK_DEDICATED_ROUTE = FALSE;.

Celý příklad je vidět na přiloženém CD.

Programová část pro FPGA

Zde si ukážeme jak ověřit správnou funkci pomocí příkazu bus. Budeme předpokládat, že máme procesor propojený s počítačem přes sériovou linku nebo ethernet viz. předchozí kapitola.

Pro čtení použijeme příkaz zavolaný pouze s adresou.

bus -a 0x900000XX

Za *XX* doplníme adresu viz následující tabulka.

Zápis provedeme příkazem bus zavolaným s adresou a příslušnými daty.

bus -a 0x900000XX -w 0xYYYYYYYY

Tabulka 4.3: Tabulka adres registrů

Binární adresa FPGA	Zadaná adresa - bus
000	00
001	04
010	08
011	0C
100	10
101	14
110	18
110	1C

Adresu doplníme viz. čtení a za *YYYYYYYY* doplníme příslušná data.
Kompletní zdrojové soubory jsou přiloženy na CD.

4.7 Příklad použití Picoblaze

V tomto příkladu si ukážeme vytvoření jednoduchého designu s picoblaze a s možností přehrání programu v picoblaze. Budeme vycházet z balíčku, který je k dispozici na stránkách výrobce a z modulů v něm obsažených.

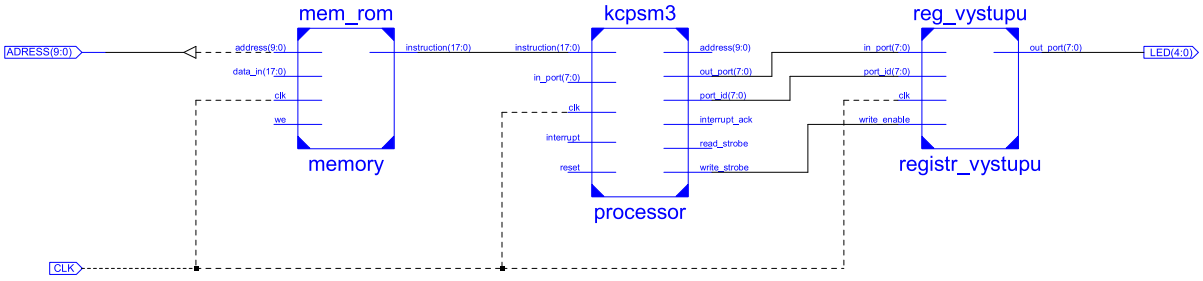
4.7.1 Teoretická úvaha řešení pro desing FPGA

Zde musíme v první části vytvořit do kterého připojíme design procesoru picoblaze, připravenou paměť a musíme vytvořit a připojit nějaký registr výstupů. Paměť musíme upravit protože dodávaná paměť je typu ROM a je postavená na BlokRAM hradlového pole a my ji musíme jednoduchou úpravou změnit na RAM. Dále musíme vytvořit registr výstupu který nám z portů picoblaze přenese výstup na LED. Ještě poté takto upravený design doplníme o změnu adresy a poslání nového programu přímo do paměti. Toto bude lepší pochopeno když se to ukáže prakticky dále.

4.7.2 Základní design do FPGA

Na obrázku 4.7 vidět základní blokové schéma designu ještě bez možnosti přehrání programu.

Z tohoto blokového schématu nás zajímají pouze bloky pro paměť (který musíme upravit) a blok Registr_výstupů. Blok processor zatím nebudeme řešit protože ho pouze tak připojíme do designu.



Obrázek 4.7: Základní zapojení Picoblaze

Nejdříve si popíšeme jak upravíme paměť, to bude potřeba později kde si ukážeme možnost přehrání program a k tomu musí být samozřejmě paměť uzpůsobena. Jako paměť vezmeme blok *ROM_form* umístěný v balíčku s picobalze. Zde nejprve do entity doplníme vstupní sběrnici dat o velikosti 18 bit a vstupní signál pro povolení zápisu (write enable). Dále v na konci souboru ještě musíme tyto signály napojit na paměť. Vstupní datovou sběrnici zapojíme obdobně jako signál *instruction*, tj. na vstup *DI* připojíme nejnižších 16 bitů a na vstup DIP připojíme zbylé 2 bity. Na signál povolení zápisu připojíme na *WE*. Výsledné zapojení by mělo vypadat takto:

```
port map( DI => data_in(15 downto 0),
          DIP => data_in(17 downto 16),
          EN => '1',
          WE => we,
          SSR => '0',
          CLK => clk,
          ADDR => address,
          DO => instruction(15 downto 0),
          DOP => instruction(17 downto 16));
```

V dalším kroku musíme vytvořit registr výstupů to je jednoduchý synchronní podmíněný výstup, kde jako vstup bude signál *write_strobe*, 8. bitová sběrnice *port_id* a *in_port*. Výstup z tohoto bloku budeme mít *out_port*, který později v hlavním souboru [2]. Samotné tělo registru je ukázáno níže zde už není co popisovat je to z toho jasné.

[2] hlavním souborem je myšlen soubor kde jsou všechny ostatní bloky propojeny

```
process(write_enable, clk, port_id)
begin
        if(clk'event and clk = '1') then
         if((write_enable = '1') and
              (port_id = "10000000")) then
                        out_port <= in_port;
            end if;
        end if;
end process;
```

Teď už jen zbývá propojit jednotlivé bloky to uděláme podle obr. 4.7 a způsob propojení je stejný jako v příkladě pro BCD čítač, kde je podrobně popsáno propojování bloků.

Do UCF vložíme akorát signál časovací signál CLK a LED diody na výstup. Z tohoto plyne že můžeme použít tento soubor z některého z předchozích příkladů.

Program ještě nekompilujeme, ještě musíme vytvořit program pro samotný Picoblaze.

4.7.3 Vytvoření programu pro Picoblaze

K tvorbě programu budeme využívat program pBlazIDE ([19]). Vytvoříme jednoduchý příklad - blikač. V podstatě se jedná o 3 cyklické čítače vzájemně do sebe vnořené.

```
PORT1       EQU         $80

VHDL        "ROM_FORM.vhd", "mem_rom.vhd", "mem_rom"
MEM         "data1.mem"

            LOAD        s3, $0F

LOOP:       ADD         s0, 1
            OUT         S3, PORT1
            COMP        s0, $FF
            JUMP        C, LOOP
            ADD         s1, 1
            COMP        s1, $FF
            JUMP        C, LOOP
            ADD         s2, 1
            COMP        s2, $FF
            JUMP        C, LOOP
            JUMP      NEGATE
```

```
                RET

NEGATE:         XOR         s3 ,  $FF
                JUMP        LOOP
```

Zde je vidět kompletní program jednoduchého čítače. Jedná se o jednoduchý program na kterém není asi moc co vysvětlovat. Jediné na co bych chtěl upozornit je na druhém a třetím řádku. Na druhém řádku příkladu máme příkazem *VHDL* vytvoříme a nahrajeme strukturu do paměti. Zde je definováno na prvním místě definováno vzorový soubor do kterého se data vloží, na druhém místě je jak se má jmenovat výstupní soubor a poslední je název entity ve vygenerovaném souboru s pamětí. V našem příkladě si jako zdrojovou paměť můžeme dát tu námi upravenou. Program se vygeneruje zpětnou šipkou v horním panelu.

Na dalším řádku nám příkaz *MEM* slouží k vygenerování textového souboru kde je vygenerován obsah instrukční paměti procesoru a je uložen v 32 bitových slovech (ale v každém slově jsou instrukce uloženy jen v nejnižších 18 bitech). Toto využijeme později.

Teď už můžeme kompletní design nahrát zkompilovat a nahrát o FPGA.

Kompletní zdrojové soubory jsou přiloženy na CD.

4.8 Přehrání programu v Picoblaze

Budeme vycházet z předchozího příkladu který doplníme o možnost přehrání programu z procesoru, kde ale v tomto případě budeme pro společnou komunikaci využívat časovací signál. Tady nás zajímá pouze zápis na sběrnici (z hlediska procesoru) tj. musíme zase dodržet časování. Dále máme ještě podmínku že data musíme přečíst v jednom strojovém cyklu.

Zde je vidět časování procesoru na sběrnici, z něho nás ale budou zajímat jen některé signály. Zajímá nás časovací signál *sdclk*, adresa *addr*, chip select signál *csd*, povolení zápisu *sdwe* a data *data_out*. Dále musíme zajistit aby se nám příchozí adresa nezpozdila při přepínání ani o jeden tak.

Dále ještě musíme održet časování pro zápis do Blok RAM. To vidíme na následujícím obrázku.

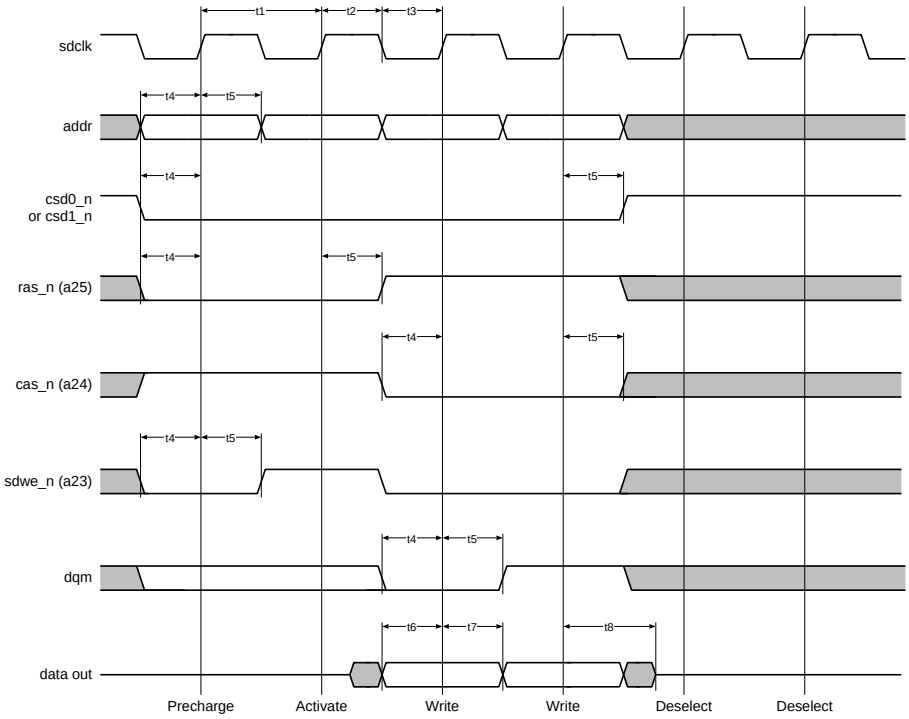
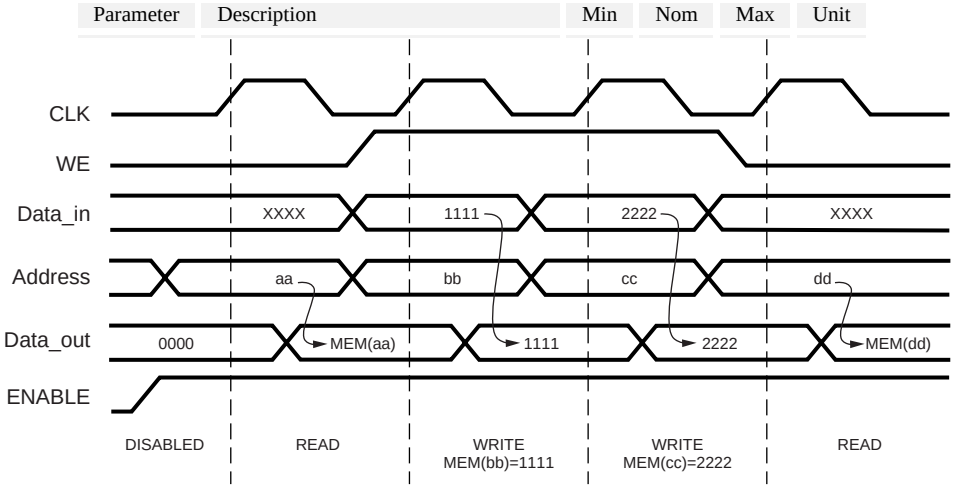


Figure 4.28: *SDRAM write timing*

Obrázek 4.8: Časování procesoru na sběrnici [10]



Obrázek 4.9: Časování z8pisu do Blok RAM [22]

4.8.1 Programové řešení pro FPGA

Zde je velmi jednoduché řešení:

```
write_mem_process : process(CLK, ENABLE, WE)

begin
```

```
        if  CLK = '0' and CLK'EVENT then
         if ENABLE = '0' and WE = '0' then
                data_in_signal <= DATA(17 downto 0);

                if temp = '0' then
                        we_signal <= '1';
                        reset_signal <= '1';
                        temp <= '1';
                else
                        we_signal <= '0';
                        reset_signal <= '0';
                end if;
         else
                temp <= '0';
                we_signal <= '0';
                reset_signal <= '0';
          end if
        end if;
end process;
```

Tento kód nám slouží k nahrání programu. Nejdříve máme obvyklé podmínky pro synchronizaci a pro příchod povolení, takže pokud nám přijde povolení zápisu a přijde sestupná hrana časovacího signálu (viz. časování BlokRAM) tak se nám do signálu *data_in_signal* zapíšou data ze sběrnice. Dále tu máme blokovací podmínku pomocí signálu *temp*, která nám zajistí aby se data zapsala pouze v jednom strojovém cyklu (aby to bylo více je zbytečné). V tomto cyklu se nám vygeneruje na dobu zápisu reset procesoru a povolí se nám zápis do paměti. Pokud tam nemáme data nebo není tento strojový cykl ta se nám nic neděje. Jak jsme si již určitě všili tak toto celé je schováno v procesu.

Dále ještě nám zbývá vyřešit adresa tu vyřešíme jednoduše do architektury dáme podmínku že pokud máme zapsat data (tj *temp* je v log 1) tak přivedeme do paměti adresu od z venkovní sběrnice jinak tam bude adresa z Picoblazu.

UCF soubor použijeme stejný jako v příkladě registrů (4.6.1) s tím že tam doplníme LED diody (ty také můžeme vzít z předchozích projektů) a doplníme časovací signál tvořený procesorem. Jedná se o signál *SDRCLKF* přivedený na pin B9.

4.8.2 Způsob přehrání programu z linuxového procesoru

Zde budeme, obdobně jako v příkladu registrů, uvažovat že systém je připojený k sériové lince, také budeme používat program Bus. Nejprve musíme do procesoru nahrát vytvořený

soubor s obsahem paměti procesoru (např. pomocí WinScp). poté už nám nic nebrání v nahrání paměti. Nejprve musíme změnit aktuální používanou složku příkazem *cd* a poté můžeme použít program Bus. Uvedu zde příklad použití.

cd /home/fpga/

bus -a 0x90000000 -c 20 -f data.mem

Na tomto příkladu je vidět nejprve, již zmíněná, změna pracovního adresáře a posléze i nahrání programu. Parametrem *-a* nastavíme počáteční adresu, parametrem *-c* nastavíme kolik datových slov se má nahrát ze souboru do paměti.

Kompletní zdrojové soubory jsou přiloženy na CD.

5 POUŽITÝ SOFTWARE

V této kapitole v první části několika slovy popíšeme operační systém použitý v tomto zařízení. V další části si představíme knihovnu Bus_space, která je použita pro práci se sběrnicí. Dále si řekneme základní vlastnosti programu.

5.1 Použitý operační systém

Tento řídící systém pohání Linux, toto řešení se nazývá system on the chip (systém na chipu). Jedná se o jednoduchou a flexibilní konfiguraci hardwarového nastavení a softwarových aplikací. Tento systém je založe na linuxovém jádru 2.6 s plnou podporou MMU, standartních glibc knihoven, sdílených knihoven a dalších aplikací. Linuxové jádro obsahuje ovladače zařízení pro procesor jako je Ethernet, sériové a paralelní porty, univerzální I/O, USB, IDE, pro komunikaci s FPGA a s dalšími zařízeními.

5.2 Program BUS

Jedná se o program který umožňuje různou formou zapisovat či číst z paměťové sběrnice. Tento program jsem získal od vedoucího práce a upravil o možnost zápisu programu v souboru pro Picoblaze vygenerovaný programem pBlazIDE, což využiji později.

5.2.1 Bus_space

Jedná se o nejdůležitější knihovnu tohoto programu proto si ji popíšeme samostatně.

Knihovnu Bus_space připojíme k programu jako *machine/bus.h*. Slouží, nezávisle na ovladači zařízení,k přístup na paměťovou sběrnici a oblast registrů. Více společných zařízení je použito na více architekturách, ale přístupy jsou pro každé zařízení jiné, protože každá architektura je určitým způsobem omezená. Pro příklad zařízení které je mapované na jeden systémový I/O prostor může být mapované v paměťovém prostoru druhého systému. Na třetím systému mohou být limity dané architekturou, mohou změnit přístup k registrům (např. vytvářením nelineárního prostoru). V některých případech, může jeden ovladač potřebovat přístup do stejného typu zařízení mnoha způsoby v jednom ze systémů nebo architektur. Cílem funkce BUS SPACE je povolit jednomu ovladači ze zdrojového souboru manipulovat a nastavovat zařízení u rozdílných systémových architektur a povolit jednomu ovladači objektového souboru manipulovat a nastavovat na zařízení vícenásobných typů jedné architektury.

Adresový prostor je popsán tagy, které mohou být jedině v kódu závislém na zařízení. Na daném zařízení může být několik rozdílných typů adresových prostorů (např. paměťový

prostor I/O prostor), a tak mohou poskytovat více rozdílných tagů. Jednotlivé sběrnice nebo zařízení na stroji, mohou používat více než jeden tag. Architektury mohou mít několik různých tagů, které představují stejný typ prostoru, například proto, že z více různých hostů sběrnice na rozhraní chipsetu.

Rozsah adresového prostoru je popsán adresou sběrnice a její velikostí. Adresa sběrnice popisuje začátek adresového prostoru. Sběrnice které nejsou byte adresovatelné můžou vyžadovat použití rozsahu adresového prostoru odpovídajícím způsobem sladěnou adresou a správnou velikostí.

Přístup do oblastí adresového prostoru je usnadněn použitím adresových ovladačů, které jsou obvykle vytvořeny namapováním specifického rozsahu v adresovém prostoru. Ovladače mohou být vytvořeny alokováním a namapováním rozsahu adresového prostoru, aktuální pozice, která je vybrána při provádění v mezích určených volajícím alokační funkce.

Všechny funkce přistupující do adresového prostoru odpovídají jednomu prostoru tagových argumentů, alespoň jeden argument ovladače a alespoň jeden argumentu offsetu (velikosti sběrnice). Tag adresového prostoru specifikuje prostor, každý ovladač definuje oblast v prostoru a každý offset specifikuje offset uvnitř oblasti v aktuálním umístění k přístupu. Offset se udává v bytech, ačkoliv sběrnice může uložit zarovnání konstant. Offset se používá k přístupu k datům vzhledem k ovladačům k ovladačům musí být takové, aby byl přístup na namapovaný region ovladačem popsán. Přístup k datům regionu vyvolá chybu.

Zde jsou popsány jen některé informace. Více je napsáno a popsáno v [21]

5.2.2 Popis programu BUS

Jak již bylo řečeno jedná se o program, který umožňuje práci se I/O sběrnicí procesoru. je založen na, již výše popsané, knihovně Bus_space, která nám umožňuje práci s adresovým prostorem.

Tento program má tyto možnosti (parametry):

- Zadání adresy
- Nastavení šířky přístupu (1, 2, 4)
- Nastavení množství přenesených datových slov[1]
- Kopírování dat z textového hex souboru do paměti
- Kopírování dat ze souboru Motorola SREC do paměti
- Zákaz inkrementace adresy u vícenásobných operací
- Práce se stránkami registrů
- Zápis hodnoty

[1] Datovým slovem je myšleno v tomto případě jedno přenesení 32 bit dat

Program v podstatě v závislosti za zadaných parametrech volá funkce knihovny bus_space. Nejprve jsou při procházení jednotlivých parametrů zkontrolovány jestli jsou parametry v pořád, dále jsou takto získané informace použijeme pro volání funkcí knihovny. Při pokusu práce se sběrnicí musíme nejprve namapovat daný prostor a až po té můžeme provádět zápis či čtení do tohoto prostoru.

Já jsem v rámci své práce upravil tento program pro možnost nahrání programu pro Picoblaze vygenerovaného programem pBlazIDE. Vygenerovaný soubor je ve formátu kde máme nejdříve adresu a poté 16 32. bitových slov. Program tuto adresu odfiltruje protože data jdou postupně a počáteční adresu určujeme programem jsou pro nás zbytečná. Data poté pošle na sběrnici. V základu je nastavena šířka sběrnice na 32 bit což můžeme ponechat. Použitý program je také přiložený na CD.

6 ZÁVĚR

V této práci jsem se zabýval tvorbou referenčních příkladů na systém EUS FS se zaměřením hlavně na hradlové pole a komunikaci s linuxovým procesorem. Pro linuxový procesor jsem provedl modifikaci programu pro práci se sběrnicí.

V první části jsem se seznámil s tímto systémem a jeho vlastnostmi a funkcemi. Dále jsem popsal soft core procesor Picoblaze, použitý v příkladech, - jeho základní vlastnosti, funkce a ve zkratce i něco málo o práci s ním. Dále jsem v krátké kapitole popsal možnost připojení linuxového procesoru přes sériovou linku a přes ethernet k počítači. V další části jsem vytvořil základní prvky - čítače na kterých je ukázáno základní postupy práce v jazyce VHDL, dále jsem se zaměřil nejprve na ukázání komunikace procesoru s FPGA bez synchronizačního signálu (na příkladu registrů) a dále jsem se zaměřil na vytvoření příkladu se soft core procesorem a posléze i na přehrání programu v tomto soft core procesoru (kde je řešeno časování pomocí signálu z procesoru). Postupy jsem se snažil volit tak aby to bylo snadno pochopitelné a srozumitelné hlavně pro čtenáře který nemá s prací na hradlovém poli, v jazyce VHDL ani s prací s linuxovým procesorem žádné zkušenosti. Touto kapitolou jsem chtěl aby snadno pochopil základní způsob práce v jazyce VHDL a také se naučil používat základní prvky práce s tímto systémem. V poslední kapitole je popsána knihovna BUS_SPACE, která je hlavním prvkem pro práci se sběrnicí v následně popsaném a mnou modifikovaném programu BUS. Tento program jsem modifikoval o možnost nahrání programu ze souboru vygenerovaného programem pBlazIDE.

REFERENCE

[1] *EUS FS - Alice II - Embeddable Single Board Computer* [online]. English Dostupné z WWW: <http://dspfpga.com/index.php?option=com_content&task=view&id=23&Itemid=42?=en>.

[2] *EUS FS - Alice II - Embeddable Single Board Computer* [online]. English Dostupné z WWW: <http://dspfpga.com/images/stories/produkty/aliceII/EUS_FS 1.1 schematic.pdf>.

[3] Axis *ETRAX FS Designer's Reference* [online]. 2007, poslední aktualizace 6. 2. 2007 [cit. 29. 12. 2009]. Dostupné z URL: <http://www.axis.com/files/manuals/etrax_fs_des_ref-070821.pdf>.

[4] Micron *Double Data Rate (DDR) SDRAM datasheet* Micron, Inc. [online]. 2005, poslední aktualizace 15. 3. 2005 [cit. 20. 1. 2010]. Dostupné z URL: <http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf>.

[5] Samsung *DDP 512Mbit SDRAM 8M x 16bit x 4 Banks* Samsung, Inc. [online]. 2002, poslední aktualizace July 2002 [cit. 20. 1. 2010]. Dostupné z URL: <http://digchip.com/datasheets/parts/datasheet/409/K4S511632D-pdf.php>.

[6] Spansion *S29GLxxxM MirrorBit Flash Family datasheet* Spansion, Inc. [online]. 2004, poslední aktualizace 18.10.2004 [cit. 20. 1. 2010]. Dostupné z URL: <http://pdf1.alldatasheet.co.kr/datasheet-pdf/view/164981/SPANSION/S29GL256M.html>.

[7] Epson *Real time clock module datasheet* Epson, Inc. [online]. 2003, poslední aktualizace 18.10.2003 [cit. 20. 1. 2010]. Dostupné z URL: <ftp://ftp.phytec.de/pub/Datasheets/Epson_RTC/rtc8564.pdf>.

[8] Texas Instruments *MSP430 - MIXED SIGNAL MICROCONTROLLERS* Texas Instruments, Inc. [online]. 1998, poslední aktualizace September 2004 [cit. 18. 5. 2010]. Dostupné z URL: <http://pdf1.alldatasheet.net/datasheet-pdf/view/27250/TI/MSP430.html>.

[9] Teridian Semiconductor *78Q2123/78Q2133 MicroPHY^{TM} 10/100BASE-TX Transeiver* Teridian Semiconductor, Inc. [online]. 2006, poslední aktualizace Januar 2006 [cit. 23. 1. 2010]. Dostupné z URL: <http://www.scantec.de/Hi-Q-News/2005/Teridian-Microphy/Teridian-78Q2123-33-Datenblatt.pdf>.

[10] XILINX *Spartan-3E FPGA Family Data Sheet* XILINX Inc. [online]. 2009, poslední aktualizace 26.8.2009 [cit. 23. 1. 2010]. Dostupné z URL: <http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf>.

[11] WIKIPEDIA. *Joint Test Action Group* Wikimedia Foundation, Inc. [online]. 2004, poslední aktualizace 3. 12. 2009 [cit. 8. 1. 2010]. Dostupné z URL: <http://en.wikipedia.org/wiki/Joint_Test_Action_Group>.

[12] XILINX. *ISE WebPack s aplikací ISE Simulator (ISim)*[Počítačový program] Ver. 11.1, San Jose, USA, 2009 [cit. 24. 1. 2010]. Dostupné z URL: <https://xilinx.entitlenow.com/cocoon/across/general/home.html?tab=DownloadSoftware&product=&licenseType=&legacyVersion=> přes aplikaci Web Install Client Ver. 11.1., 89 MB. Freeware po registraci uživatele. Přehled podporovaných systémů je uveden na této URL : <http://www.xilinx.com/ise/ossupport/index.htm>. Velikost instalace může dosáhnout i 10 GB.

[13] XILINX. *XST User Guide* Xilinx, Inc. [online]. 2004, poslední aktualizace 16. 9. 2009 [cit. 8. 1. 2010]. Dostupné z URL: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf>.

[14] *Popis jazyku VHDL* [online]. [cit. 23.1.2010]. Dostupné z URL: <http://www.hepin.cz/storage/1245608020_sb_vhdl.pdf>.

[15] Soběslav Valach. *Moderní trendy v programovatelné logice, aplikace v automatizační a měřicí technice* Soběslav Valach. [online]. 2007, poslední aktualizace 29.3.2007 [cit. 23. 1. 2010]. Dostupné z URL: <http://www.roznovskastredni.cz/dwnl/pel2007/10/Valach.pdf>.

[16] Vít Olmr. *HW server představuje - Sériová linka RS-232* Vít Olmr. [online]. 2007, poslední aktualizace 12.12.2005 [cit. 22. 5. 2010]. Dostupné z URL: <http://hw.cz/rs-232>.

[17] XILINX. *PicoBlaze 8-bit Embedded Microcontroller User Guide* Xilinx,Inc. [online]. 2004, poslední aktualizace 28.1.2010 [cit. 23. 5. 2010]. Dostupné z URL: <http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf>.

[18] XILINX. *PicoBlaze for Extended Spartan-3A Family, Virtex-4, Virtex-II, and Virtex-II Pro FPGAs*[Referenční návrh] Ver. 8a, San Jose, USA, 2005 [cit. 23. 5. 2010]. Dostupné z URL: <http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm>.

[19] MEDIATRONIX. *pBlazIDE*[Program] Ver. 3.74 Beta, Hoofddorp, Holandsko, 2006 [cit. 24. 5. 2010]. Dostupné z URL: <http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm>.

[20] Doc. Ing. Jaromír Kolouch, CSc. *Implementace procesorů v obvodech FPGA* Doc. Ing. Jaromír Kolouch, CSc. [online]. 2008, poslední aktualizace 24.6.12.2008 [cit. 24. 5. 2010]. Dostupné z URL: <http://www.urel.feec.vutbr.cz/web_pages/projekty/clanky/Kolouch_FPGA.pdf>.

[21] Chris Demetriou. *bus_space manual page* Chris Demetriou. [online]. 2008, poslední aktualizace 1.3.2008 [cit. 25. 5. 2010]. Dostupné z URL: <http://www.daemon-systems.org/man/bus_space_map.9.html>.

[22] XILINX. *Using Block RAM in Spartan-3 GenerationFPGAs* Xilinx, Inc. [online]. 2005, poslední aktualizace 1. 3. 2005 [cit. 26. 5. 2010]. Dostupné z URL: <http://www.xilinx.com/support/documentation/application_notes/xapp463.pdf>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

I/O vstupně - výstupní obvody

UART RS-232 Komunikační seriové rozhraní

JTAG Rozhraní sloužící k nahrávání a ladění aplikací

I_2C Komunikační seriová sběrnice

MMU Memory Management Unit - jednotka řízení paměti

CPU Central procesor unit - hlavní výpočetní jednotka v procesoru

RISC Typ konstrukce procesoru s redukovanou instrukční sadou

DMA Direct memory access - Přímý přístup k paměti

MMU Memory management unit - jednotka řízení paměti

Baud (Bd) je modulační rychlost (také znaková rychlost nebo anglicky baud rate) udávající počet změn stavu přenosového média za jednu sekundu. Pro některé typy modulací může platit, že 1 baud = 1 bit/s.

UCF Implementation coinstant file - soubor pro namapování proměnných na fyzické piny

BCD Binary Coded Decimal - dvojkově reprezentované dekadické číslo

LED Světlo vyzařující dioda

DDR Double data rate

CPLD Komplexní programovatelná hradlová pole

FPGA Field Programmable Gate Arrays – na místě programovatelné logické obvody

DLL Delay Locked loop

DHCP Dynamic Host Configuration Protocol - protokol pro dynamické přidělování parametrů ethetnetové sítě

BCD Binary Coded Decimal - dvojkově reprezentované dekadické číslo

ROM Read Only Memory - pamět umožňující pouze čtení

RAM Random Access Memory - pamět s libovolným přístupem

MIPS Milion Instruction Per Second - milion instrukcí za sekundu

ALU Arithmetic Logic Unit - aritmeticko logická jednotka

CLB configurable Logic Blocks - konfigurovatelný logický blok

SEZNAM PŘÍLOH

A POROVNÁNÍ INSTRUKCÍ A DIREKTIV ASSEMBLERU KCPSM3 A PBLAZIDE

A.1 Instrukce

Tabulka A.1: Přehled rozdílů instrukcí assembleru mezi KCPSM3 a pBlazIDE [17]

Instrukce KPCSM3	Instrukce pBlazIDE
RETURN	RET
RETURN C	RET C
RETURN NC	RET NC
RETURN Z	RET Z
RETURN NZ	RET NZ
RETURNI ENABLE	RETI ENABLE
RETURNI DISABLE	RETI DISABLE
ADDCY	ADDC
SUBCY	SUBC
INPUT sX, (sY)	INPUT sX, sY
INPUT sX, kk	INPUT sX, kk
OUTPUT sX, (sY)	OUTPUT sX, sY
OUTPUT sX, kk	OUTPUT sX, kk
ENABLE INTERRUPT	EINT
DISABLE INTERRUPT	DINT
COMPARE	COMP
FETCH sX, (sY)	FETCH sX, sY
STORE sX, (sY)	STORE sX, sY

A.2 Direktivy

Tabulka A.2: Přehled rozdílů direktiv assembleru KCPSM3 a pBlazIDE [17]

Funkce	Direktiva KPCSM3	Direktiva pBlazIDE
Poloha instrukce	**ADDRESS** 3FF	**ADDRESS** $3FF
Přejmenování registru	**NAMEREG** s5, myregname	myregname **EQU** s5
Deklarace konstant	**CONSTANT** myconstant, 80	myconstant **EQU** $80
Pojmenování ROM souboru s pamětí	Pojmenováno podle zdrojového souboru assembleru	**VHDL** „template.vhd“, „target.vhd“, ”jmeno_entity“