České vysoké učení technické v Praze
Fakulta elektrotechnická

Diplomová práce

Implementace mikrořadiče AVR využívajícího všechny periferie přípravku DIGILENT Starter kit

Pavel Dědourek

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Elektrotechnika a informatika strukturovaný magisterský

Obor: Informatika a výpočetní technika

leden 2008

Poděkování

Chtěl bych poděkovat svým spolužákům za neocenitelné rady během práce, nejen závěrečné a rodině, která mě podporovala po celou dobu studia.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 7. ledna 2008 .....

Abstract

The goal of this thesis is to implement already existing microcontroller AVR described in VHDL language for FPGA device Spartan 3 on development board "Starter Kit". The utilization of this microcontroller allows the developer an easier design of some more complex devices. The control of this circuit is achieved by some higher programming language. Firstly VHDL description of microcontroller AVR will be documented and sequentially modified for implementation. Modified microcontroller will be extended by peripheries that are offered by "Started kit" board from Digilent. For final implementation will be written test application which use all peripheries.

Abstrakt

Cílem této diplomové práce je implementovat existující řešení mikrořadiče AVR popsaného v jazyce VHDL do FPGA obvod Spartan 3 na vývojové desce Stater kit. Použití mikrořadiče v tomto obvodu umožňuje vývojáři zjednodušit návrh některých složitých zařízení. Řízení tohoto obvodu je možné pomocí vyššího programovacího jazyka (například C). VHDL popis mikrořadiče AVR bude nejprve zdokumentován a následně upraven do podoby vhodné pro implementaci. Takto upravený mikrořadič bude rozšířen o periferie obsažené na vývojové desce Starter kit od firmy Digilent. Pro výslednou implementaci bude napsána testovací aplikace využívající všechny periferie.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V této diplomové práci je popsáno stávající řešení mikrořadiče ATmega 103 od firmy Atmel [6], které bylo zveřejněno na [12]. Tato práce pana Ruslana Lepetenoka [11] byla realizována pro potřeby simulace (například v modelsim [3]). Zdrojový kód bylo nutno pozměnit pro implementaci do reálného hardwaru. Hardware tvoří vývojová deska Starter kit [8] od firmy Digilent. Vlastní mikrořadič je nahrán do obvodu FPGA (viz 2.2.4).
Tento procesor z rodiny AVR, je vývojově podporován společností Atmel.Žádnou překážkou tedy není vyvíjet software pro tento mikrořadič. Program může být tvořen jak v jazyku symbolických adres (assembler), tak i ve vyšším programovacím jazyku (například C).
Poté je možné použít daný procesor k ovládání složitějších hardwareových celků, u kterých by čistě hardwarová realizaci byla příliš složitá. Mikrořadič je připojen k periferiím standartním způsobem a to pomocí datové, adresové a řídící sběrnice. Je taktéž vybaven přerušovacím systémem.
Diplomová práce tvoří dvě ucelené části. První část se týká jádra mikrořadiče, druhá část realizace periferií mikrořadiče. Kapitola o jádru mikrořadiče obsahuje analýzu, popisující situaci mezi procesory dostupnými pro FPGA obvody. Dále je zde rozebráno existující řešení mikořadiče ATmega 103.
Návrh řešení popisuje možnosti, jak by bylo možné realizovat daný mikrořadič přímo do FPGA obvodu.
V implementační části je popis hotového jádra mikrořadiče vhodného pro FPGA.
Poslední část popisuje vyhodnocení mikrořadiče vzhledem k jeho funkčnosti a omezení.
Kapitola, věnující se periferním obvodům, obsahuje opět část analýzy. Zde jsou zhodnoceny možnosti připojení periferií k jádru procesoru a rozbor jednotlivých periferií na vývojové desce (VGA, RS232, PS/2), jejich komunikační protokoly a časování.
V další kapitole je ke každé periferii navrženo přibližné řešení s odůvodněním, proč takto realizovat daný obvod.
Dále následuje testování, kde je popsán program pro vyzkoušení celého přípravku s jeho periferiemi (textový terminál, zkouška grafického režimu).
Poslední kapitola obsahuje zhodnocení funkčnosti celého přípravku včetně jeho periferií.

2 AVR Jádro

2.1 Úvod

Jelikož některé systémy v FPGA jsou příliš složité na návrh, je možné využít mikroprocesory k řízení jednodušších celků a pomocí softwaru v mikroprocesoru se tak vyhnout složité hardwarové realizaci celého obvodu. V následující části jsou zmíněny dostupné softwarové procesory, dodávané jako softcore projekty pro použití v návrhu do FPGA obvodu.

Název	[bit]	typ procesoru	velikost adresového prostoru	LUTs	SLICEs	f_{MAX}
PicoBlaze	8	RISC	-	-	96	88MHz
PacoBlaze	32	RISC	4GB	-	-	-
LatticeMico8	8	RISC	256	250	140	80MHz
LatticeMico32	32	RISC	4GB	2000	-	100MHz
MicroBlaze	32	RISC	4GB	1357	-	100MHz
Nios	16	RISC	-	-	-	-
Nios II	32	RISC	2GB	-	-	-

Tabulka 2.1: Seznam dostupných softwarových procesorů pro FPGA

2.2 Analýza

Existují dvě varianty mikroprocesorů a to hardwarové (přímo realizované na FPGA obvodu), nebo softwarové (nutné nahrát pomocí bit streamu do FPGA). Jednodušší typy FPGA obvodů však neobsahují mikroprocesor přímo na sobě.

2.2.1 Softwarové procesory

Námi dostupný FPGA obvod Spartan 3 [13] neobsahuje žádný z daných procesorů a proto můžeme použít pouze softwarové verze mikrořadičů. V tabulce 2.1 je seznam dostupných softwarových procesorů pro FPGA.

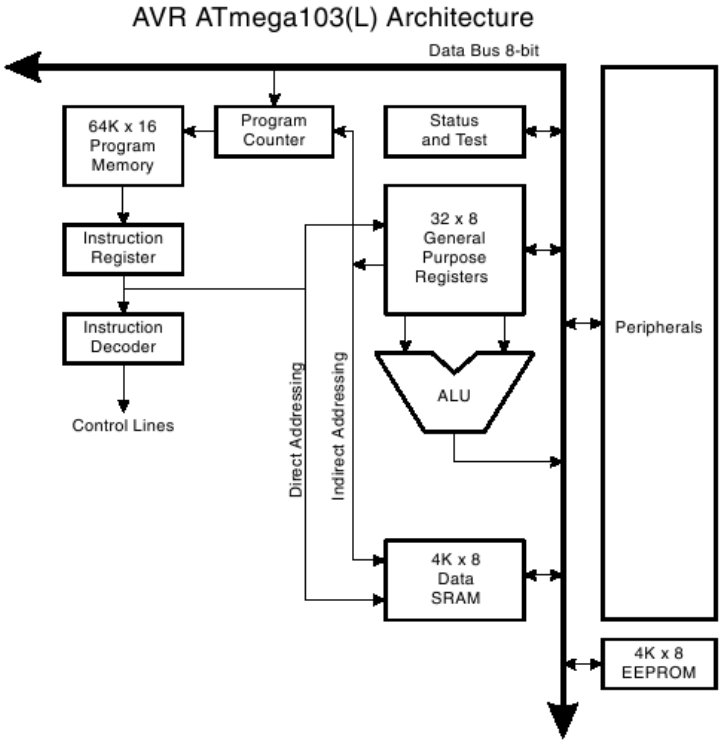
2.2.2 ATmega 103

AVR je označení pro rodinu mikrořadičů firmy Atmel [5]. Jde o 8bitové RISC procesory. Existuje několik modelových řad. Mezi jednu z nich patří ATmega (podskupinu tvoří řada ATtiny). V této práci byl implementován řadič ATmega 103, což je v současnosti již zastaralá řada, ale její kód, vhodný pouze pro simulaci, je dostupný na internetových stránkách [12].
Mikrořadič ATmega103 disponuje těmito vlastnostmi:

- 121 výkonných instrukcí (většinou jednocyklové)
- 32 8bitových registrů (tj. registrové pole GPR - General Purpose Register)
- 128KB paměti programu (16-bitová adresová sběrnice a 2B instrukce)
- 4KB interní RAM paměť
- SPI - sériové periferní rozhraní
- Watchdog
- UART - univerzální asynchronní sériový kanál
- RTC - real time čítač
- 16bitový časovač/čítač
- ADC - analogočíslicový převodník
- více druhů přerušení (vnitřní,vnější)

2.2.2.1 Popis AVR architektury

AVR jádro je založeno na RISC architektuře, kombinující velké množství instrukcí, s využitím 32 registrů v registrovém poli. Všech 32 registrů je přímo napojeno na aritmetickou jednotku (ALU), to umožňuje nezávislý přístup ke dvěma registrům na jedno cyklovou instrukci. Tímto může být tato architektura mnohonásobně rychlejší než CISC mikroprocesory.



Obrázek 2.1: ATmega103 AVR RISC architektura

AVR jádro používá hardwardskou architekturu s oddělenou datovou a instrukční pamětí a oddělenými sběrnicemi pro data a instrukce. K programové paměti je přistupováno pomocí jednoúrovňové pipeliny. V jednom taktu se při provádění instrukce načítá další instrukce v pořadí. To umožňuje dokončení instrukce v každém taktu hodin. AVR instrukce mají slova dlouhá 16bitů (každá adresa v instrukční paměti obsahuje jednu 16bitovou instrukci).

2.2.2.2 Zásobník

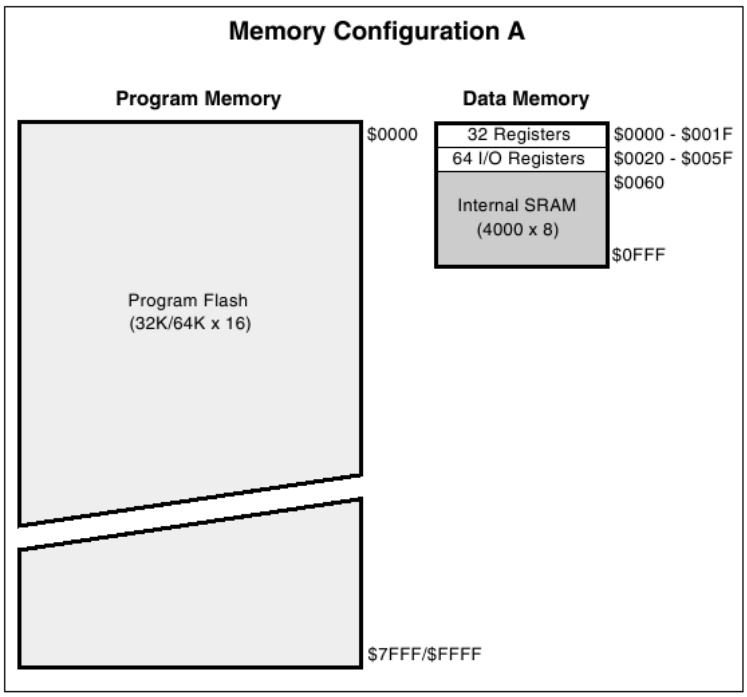
Během vykonávání přerušovacích rutin a při odskoku do programu, se ukládá programový čítač (PC) na zásobník. Zásobník je uložen v datové paměti (organizovaná po 8bitových slovech). Velikost zásobníku je omezena jen volnou datovou pamětí. Před jakýmkoliv použitím zásobníku je nutné provést inicializaci ukazatele (SP) do zásobníku. Ukazatel (Stack Pointer) je reprezentován 16bitovým registrem, ke kterému můžeme přistupovat přes vstupně-výstupní (IO) adresový prostor.

2.2.2.3 Přerušení

Přerušení může být zakázáno či povoleno pomocí bitu v registru SREG (Status Registr). Jednotlivá přerušení mají různé vektory přerušení uložené v tabulce, která je umístěna na začátku instrukční paměti. Přerušení s nižší adresou přerušovacího vektoru má vyšší prioritu.

2.2.2.4 Adresový prostor

AVR architektura disponuje dvěmi pamětmi a to datovou a programovou. Programová paměť je typu SRAM a při normálním provozu procesor může z této paměti pouze načítat instrukce, či načítat data pomocí instrukce LPM (Load Program Memory). Datová paměť je připojena na



Obrázek 2.2: Procesory řady AVR jsou hardwardské koncepce a obsahují tedy paměť programu a paměť dat. V datové paměti jsou navíc namapovány vstupně-výstupní registry a periferní zařízení.

adresovou a datovou sběrnici. V adresovém prostoru je namapováno registrové pole (registry r0-r31), IO registry (registry periférií) a datová paměť.

2.2.3 AVR core, Ruslan Lepetenok

Jádro mikrořadiče ATmega 103 popsané na [12] má již zmíněné vlastnosti popsané výše (viz 2.4.5). Jelikož dané řešení je vhodné pouze pro simulaci, objevují se zde specifické vlastnosti, které v reálné implementaci způsobují problémy.

2.2.3.1 Cpuwait

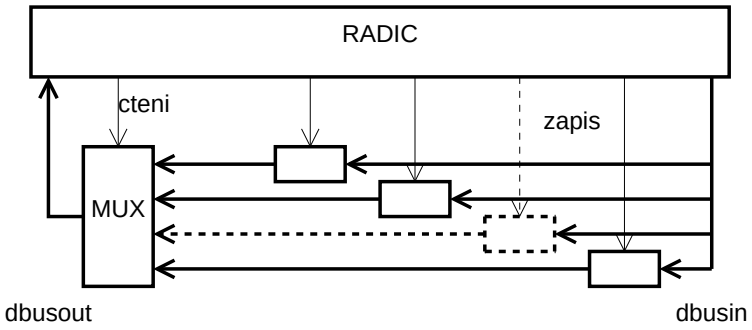
Prvním problémem je nedořešení zastavování procesoru. Signál `cpuwait` byl generován pouze při přístupu do datové paměti. Avšak pro práci s externími periferiemi a pro potřeby zpomalení designu je nutno všechny registry v jádru procesoru pozastavit.

2.2.3.2 Datová sběrnice

Další problém nastává s navrženou architekturou datové sběrnice. Datová sběrnice není jen jedna, ale každý blok má svůj vstup a výstup. Všechny výstupy z bloků jsou centralizovány do hlavního multiplexoru a teprve z něj jsou data posílána do vlastního jádra mikropočítače.

2.2.3.3 Paměti

Všechny paměti (RAM, ROM) jsou v návrhu tvořeny za pomoci VHDL kódu. Není využito externích pamětí a ani block RAM. Je zde paměť programu (128kB), datová paměť (4kB) a registrové pole (32 8b registrů).

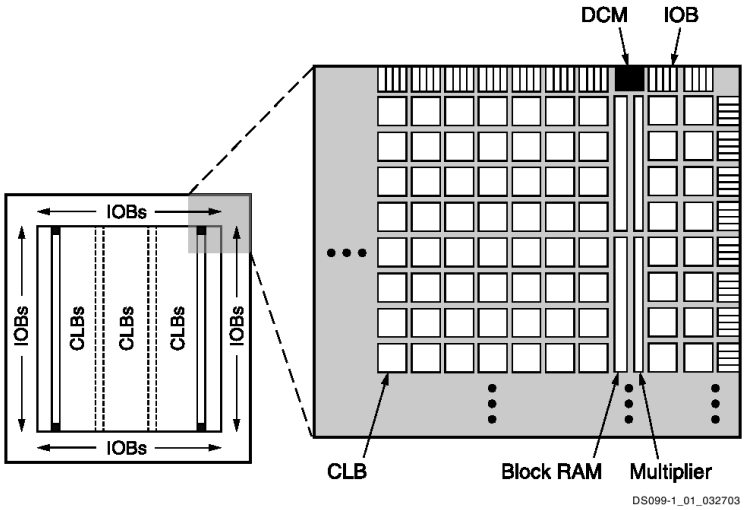


Obrázek 2.3: Datová sběrnice je rozdělena na vstupní a výstupní. Zápis do periferního zařízení je podmíněn adresou a signálem pro zápis. Výstupy z periferií obsahují skutečnou hodnotu registru a teprve až v multiplexoru se vybere daná hodnota pomocí adresy a signálu pro čtení.

2.2.4 FPGA, Spartan 3,

Řada Spartan 3 (viz [13]) navazuje na úspěch předchozí řady: Spartan IIE. Od architektury Spartan IIE se liší v počtu systémových hradel a logických buněk, velikostí RAM, počtu I/O a implementací některých nových bloků do architektury, jako jsou DCM nebo násobičky. FPGA osazené na této desce disponuje 200 tis. systémových hradel.
Hlavní rysy architektury Spartan 3 (XC3S200FT256):

- nová 90nm technologie výroby
- 200 tisíc systémových hradel nebo 4320 logických buněk
- hodinový signál až 326MHz
- 3 separátní napájecí napětí: 1,2V, 2,5V, 3,3V
- až 173 I/O pinů, 622Mb/s přenosová rychlost pinu
- podpora až 17 úrovňových standartů
- Double Data Rate (DDR) podpora
- integrovaná struktura pro konstrukci rychlých sčítaček
- 12 integrovaných násobiček 18 x 18
- JTAG port pro testování
- až 216Kb blokové RAM
- až 30Kb distribuované RAM
- až 4 Digital Clock Manager (DCM) umožňující frekvenční syntézu
- 8 globálních hodinových linek



Obrázek 2.4: Obvod FPGA je složen z vstupně-výstupních bloků (IOB), konfigurovatelných bloků (CLB), pomocné paměti (Block RAM), násobiček (Multiplier) a obvodu pro řízení časování (DCM).

- plná podpora vývojového prostředí Xilinx ISE

Architektura je složena z 5 základních funkčních/logických bloků:

- IOBs (Input/Output Blocks) řídí tok dat mezi I/O pinem a vnitřní logikou.
- CLBs (Configurable Logic Blocks) obsahující LUTs (Look-Up Tables) na principu paměti RAM.
- Block RAM umožňující ukládaní dat ve formátu 18Kb dual-port blocks.
- násobičky umožňující vynásobení dvou 18-bitových čísel.
- DCM (Digital Clock Manager) poskytuje autokalibraci, plně digitální řešení distribuce zpoždění, násobení, dělení a fázový posun hodinového signálu.

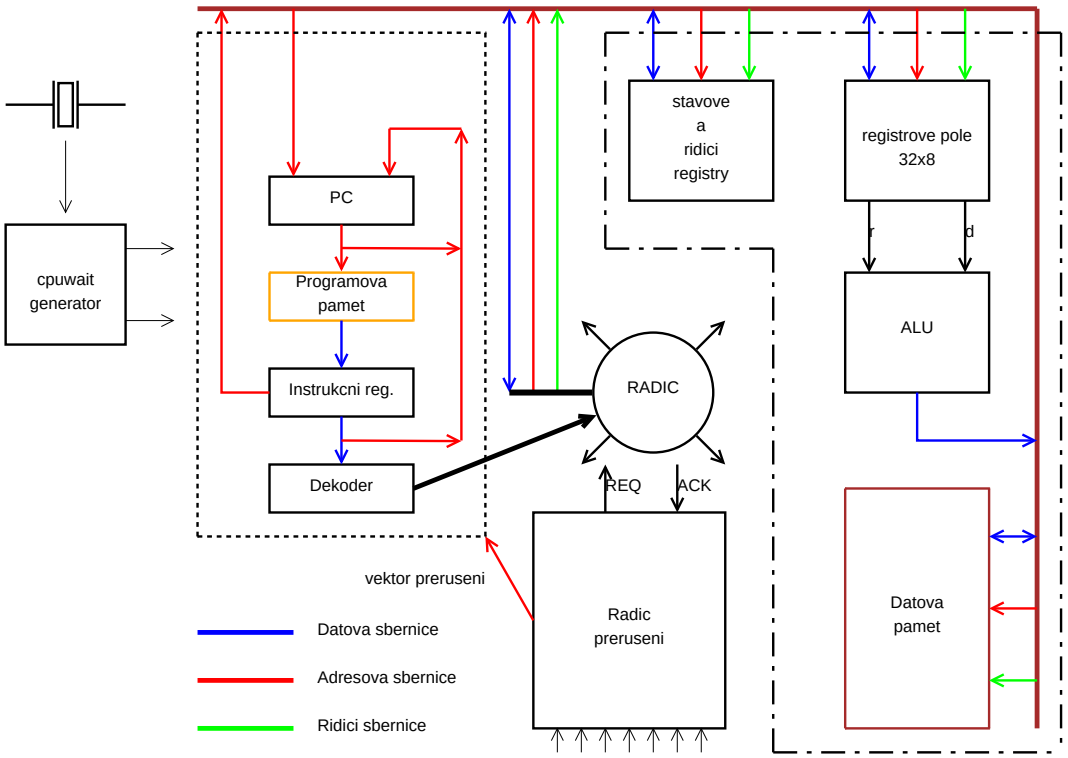
Spartan 3 využívá pro konfiguraci paměťových buněk RAM, tedy při vypnutí napájení je daná konfigurace ztracená. Proto je potřeba zajistit nahrání konfigurace při každém zapnutí FPGA. Konfigurační data jsou automaticky čtena z externího zdroje dat (PROM, JTAG, FPGA, flash paměť) a to buď sériově nebo paralelně.

2.3 Návrh řešení

Procesor ATmega 103 [6], který byl dostupný na internetových stránkách [12], byl vhodný pouze pro simulaci. Pro použití tohoto řešení v FPGA obvodu je nutné provést některé úpravy. Pro snížení zaplněnosti FPGA je nutné maximálně využít bloky, které má obvod FPGA integrované přímo na sobě (paměti). Dále by mělo být možno zastavovat procesor tak, aby mohl počkat na pomalejší periferie. Je potřeba realizovat programovou paměť jako paměť RAM (buď jako vnitřní nebo jako vnější).
Datová sběrnice by měla být realizována jako třístavová sběrnice. V již navržené implementaci byly dvě datové cesty a to vstupní a výstupní. Pro větší zřetelnost a snadné připojování periférií je výhodnější použít všeobecně známý systém třístavové sběrnice, kde se při zápisu vyšle signál write a zapíší se data na datovou sběrnici. V případě čtení se z datové sběrnice přečtou data. Jako řídící jednotka této komunikace je vlastní řadič procesoru (viz 2.3.1.9).

2.3.1 Architektura

AVR mikrořadič je hardwardské koncepce, je tedy nutné oddělit části datovou a programovou. Na obrázku 2.5 je vidět nástin jednotlivých bloků jádra mikroprocesoru. Je zde registrové pole, datová a programová paměť, aritmeticko-logická jednotka, stavové a řídící registry, programový čítač, dekodér instrukcí, řadič přerušení a nakonec řadič procesoru s jednotkou `cpuwait`. Cpuwait bude pozastavovat procesor při čekání na pomalejší periferie.



Obrázek 2.5: AVR jádro je dle hardwardské architektury rozděleno na dvě části. Levá část se týká všeho, co se týká programové paměti (instukce a tvorba hodnoty programového čítače). Datovou část je možné vidět na pravé straně. Je zde registrové pole, jehož hodnoty může použít aritmeticko-logická jednotka (ALU) pro provedení výpočtu. Celý tento systém řídí řadič, do kterého vstupují informace o právě prováděné instrukci. Řadič řídí datové toky v procesoru. Jednotka cpuwait dovede zastavit procesor. V dolní části obrázku je umístěna jednotka přerušení, která komunikuje pouze s řadičem. Tato jednotka se stará o priority přerušení a jeho správné vybrání.

2.3.1.1 Registrové pole

Navržený obvod měl veškerou RAM paměť realizovanou pomocí D registrů a proto bylo zaplnění čipu neúnosně veliké. Větší celky jako registrové pole a datovou paměť je tedy výhodné implementovat v block RAM FPGA obvodu.
Registrové pole tvoří 32 univerzálních registrů. Z toho 26 registrů je možno přímo umístit do block RAM neb není vyžadována jiná funkce těchto registrů než jen prosté ukládání a čtení. Datová šířka paměti je 8bitů a velikost 26B. Proto stačí nejmenší dostupná block RAM paměť.

2.3.1.2 Datová paměť

Datová paměť u obvodu ATmega 103 má velikost 4kB při datové šířce 8b.

2.3.1.3 Programová paměť

Programová paměť uchovává instukce programu. Jelikož mají instrukce 16bitový tvar, tak šířka datové sběrnice je 16bitů a velikost adresové sběrnice (pro 128kB) je tedy taktéž 16bitů. Z pohledu dostupných pamětí se jeví jako nejschůdnější řešení použít externí SRAM paměť pro ukládání programového kódu.

2.3.1.4 Aritmeticko-logická jednotka

Slouží k výpočtu součtu a rozdílu dvou čísel, nastavení příznaků (viz kapitola 2.4.8) a jiným různým logickým operacím. Daný mikrořadič disponuje pouze instrukcemi pro sčítání (odčítání), a proto bude v této jednotce realizována pouze sčítačka. Pro logické operace, jako je posun, je zde posuvný registr.

2.3.1.5 Stavové a řídící registry

Tyto registry slouží k ovládání přerušovacího systému a nastavení či čtení příznaků (SREG), zásobníku (SP) a výběru stránky z instrukční paměti (RAMPZ). Budou vynechány registry pro ovládání spotřeby a rychlosti mikořadiče, jelikož jsou na FPGA obvodu nepoužitelné.

2.3.1.6 Programový čítač

Tato jednotka obsahuje adresu další prováděné instrukce. Je to 16bitový registr (viz velikost programové paměti) a jeho hodnota se určuje podle právě prováděné instrukce (skoky, podprogram, přerušení, adresování instrukční paměti).

2.3.1.7 Dekodér instrukcí

Tento blok obsahuje dekodér instrukcí [7] a výstupem bude signál, určující jaká instukce je na vstupu. Dále bude obsahovat hodnotu adresy (dat), je-li v instrukci přímo zakódována.

2.3.1.8 Řadič přerušení

Obstarává vybrání přerušení s nejvyšší prioritou a umožní tvorbu vektoru přerušení. Dále vydává signál REQ (Request) žádost o přerušení a vlastní řadič mu na ní odpovídá signálem ACK (Acknowledge). Ten umožní informovat periferii, že byla obsluha přerušení dokončena.

2.3.1.9 Řadič procesoru

Vlastní řadič řídí datové cesty podle druhu instrukce. Procesor ATmega 103 [**?**] má jak jednocyklové instrukce, tak i vícecyklové instrukce. Řadič pro vícetaktové instrukce obsahuje proto konečné automaty, které postupně vykonávají danou sekvenci příkazů.

2.3.2 Zastavení procesoru

Pro potřeby zastavení procesoru z následujících důvodů (přizpůsobení rychlosti pomalejších periferií, změna obsahu programové paměti), je zde tato `cpuwait` jednotka. Zastavení znamená přerušit provádění instrukcí, výpočtů a ponechat procesor v aktuálním stavu (nezměnit obsah žádného registru v mikrořadiči). Proto tento účel je výhodné vytvořit signál (`cpuwait`) vedoucí

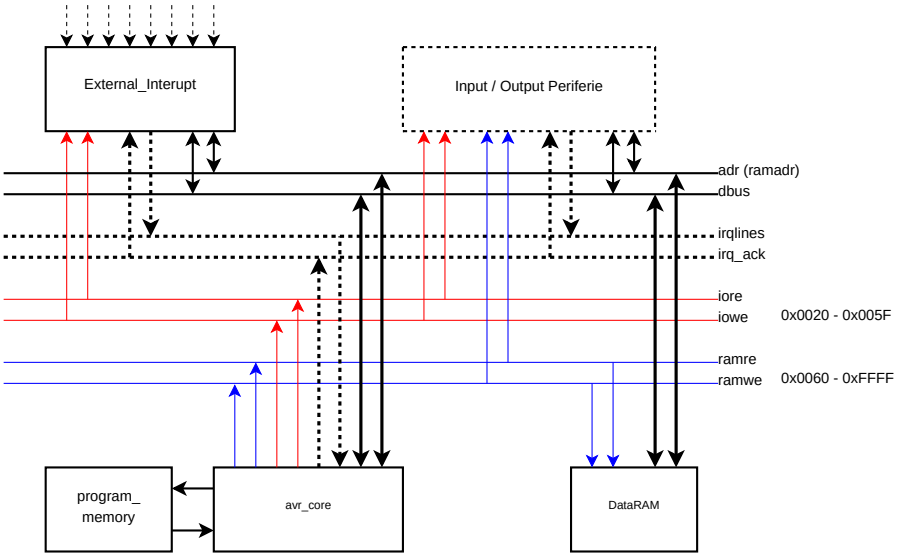
až do top architektury. Tento signál určuje, zda se stav každého registru závislého na datových signálech bude s taktem hodin obnovovat či ne.

2.3.3 Datová sběrnice

Pro snadnější připojení veškerých periferií byl navržen systém třístavové datové sběrnice. Periferie se připojí paralelně k datové, adresové a řídící sběrnici. Pokud se shoduje adresa periferie a je nastaven signál zápisu, otevře se cesta z datové sběrnice k periferii. Při aktivním signálu čtení se provede zápis z registru periferie na sběrnici. V jiných případech se na datové sběrnici objeví stav vysoké impedance (viz realizace datové paměti 2.4.2).

2.4 Implementace

2.4.1 AVR, top modul



Obrázek 2.6: Blokové schéma popisuje způsob připojení jádra mikrořadiče k ostatním perifériím a datové paměti. Datová sběrnice (dbus) je třístavová a přenášejí se po ní data. Směr toku dat ovládá vlastní jádro procesoru pomocí řídících signálů. Pro rozlišení adres pro vstupně-výstupní registry jsou zavedeny řídící signály iowe a iore. Pro ostatní adresy jde o signály ramwe,ramre.

Top modul navrženého designu 2.6 obsahuje vlastní propojení jádra procesoru 2.4.5 s periferiemi, paměťmi a řadičem externího přerušení.
Pro potřeby ladění a zpomalování procesoru je zaveden signál `cpuwait`. Pokud je tento signál v log.1, dochází k zastavení celého procesoru. Procesor tedy vyčkává a nevykonává žádnou činnost. Tohoto signálu je možné využít i v případě, kdyby měl procesor čekat na některé pomalejší periferii z hlediska nejnižší hardwarové úrovně.

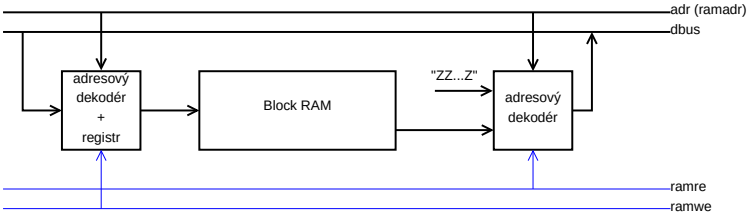
2.4.2 Programová paměť

Programová paměť je realizována v externí paměti SRAM, do které můžeme zapisovat instrukce (Bajt po Bajtu) pomocí sériové linky v programovacím režimu (`prog_enable` = log. 1). V tomto režimu se při každém přijetí signálu `rx_recv` uloží na aktuální adresu Bajt, přijatý na sběrnici `rx_data` a poté se inkrementuje adresa v SRAM paměti o jednu. Nejdřív se ukládá vyšší Bajt

z instrukce a poté nižší. Pro snadné programování by měl být propojen tento blok s blokem periferie `uart` (viz kapitola 3.4.4).

2.4.3 Datová paměť

Datová paměť je vytvořena z block RAM, v našem případě jsou použity dvě s konfigurací 4k x 4bity, tj. 4kB. V tomto modulu je nejprve záchytný registr, který se při zápisu (`ramwe`) do adresového prostoru (0-4096, 0x0000 - 0x0FFF) naplní daty z adresové sběrnice `dbus`. Další důvod použití tohoto registru je, že Block RAM potřebuje jeden takt hodin pro zápis a po tuto dobu záchytný registr uchovává hodnotu, která je potřeba uložit. Blokové schéma je na obrázku 2.7.



Obrázek 2.7: Datová paměť je přímo mapována do adresového prostoru. Velikost paměti je dána velikostí Block RAM. Při zápisu do paměti je nutné zvolit adresu, která je v uvedeném adresovém prostoru a nastaví signál ramwe. Proces čtení je téměř identický, tj. signál ramre určuje výskyt dat na datové sběrnici.

2.4.4 Externí přerušení

Řadič externího přerušení je složen z bloku, kde se nejdříve navzorkuje vstupní signál. Podle nastavení registrů EICR (náběžná, sestupná hrana), EIMSK (maskování zdrojů externího přerušení) se provede daná rutina přerušení.

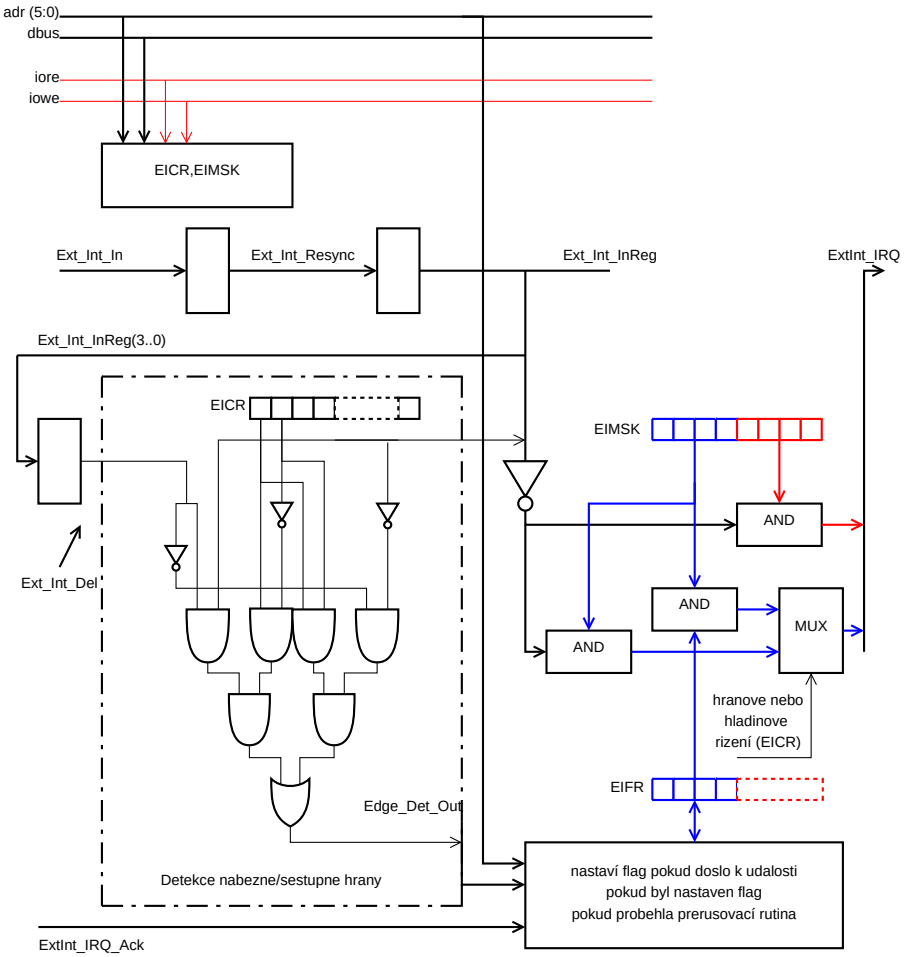
EIMSK - External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
0x39	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 4. bit - INT7 až INT4 (External Interrupt Request 7 - 4 Enable)
  Pokud je daný bit nastaven, je povolena možnost vyvolání přerušení od externího signálu. Událost, při které se generuje přerušení, se nastaví v registru EICR.
- 3. až 0. bit - INT3 až INT0 (External Interrupt Request 3 - 0 Enable)
  Povolení přerušení od externího zdroje. Toto přerušení je generováno při signálu logická nula.

EIFR - External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
0x38	INTF7	INTF6	INTF5	INTF4	-	-	-	-
čtení/zápis	rw	rw	rw	rw				
inic. hodnota	0	0	0	0				



Obrázek 2.8: Tento blok externího přerušení umožňuje pomocí registrů EICR nastavit, na jaké události bude mikrořadič reagovat. Registr EIMSK povoluje (zakazuje) dané přerušení. Ext_Int_in, Ext_Int_InReq - registry, které obsahují navzorkované signály externího přerušení. Ext_Int_Det - registr sloužící pro detekci náběžné (sestupné) hrany.

- 7. až 4. bit - INTF7 až INTF4 (External Interrupt 7 - 4 Flags)
  Indikuje, že bylo zaznamenáno přerušení od změny signálu INT7 až INT4. Pokud je navíc přerušení povoleno IE je v log. 1 (SREG) a daný bit z EIMSK je v log. 1, tak se provede přerušovací rutina a daný flag z EIFR se opět vynuluje. Bit z EIFR můžeme též vynulovat zápisem logické jedničky na jeho místo. Tyto bity jsou vždy nulové, pokud jde o přerušení od události, že daný signál je v logické nule (tj. přerušení od úrovně log. 0).

- 3. až 0. bit - nevyužito

EICR - External Interrupt Control Register

Bit	7	6	5	4	3	2	1	0
0x3A	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. bit - ISCx1, ISCx0 (External Interrupt 7 - 4 Sense Control Bits
  Určuje při jaké události dojde k přerušení od signálů INT7 až INT4.

ISCX1	ISCX0	Popis
0	0	Přerušení generováno při signálu v log. 0
0	1	Nevyužito
1	0	Spádová hrana INTx generuje přerušení
1	1	Náběžná hrana INTx generuje přerušení

Tabulka 2.2: Řízení druhu přerušení

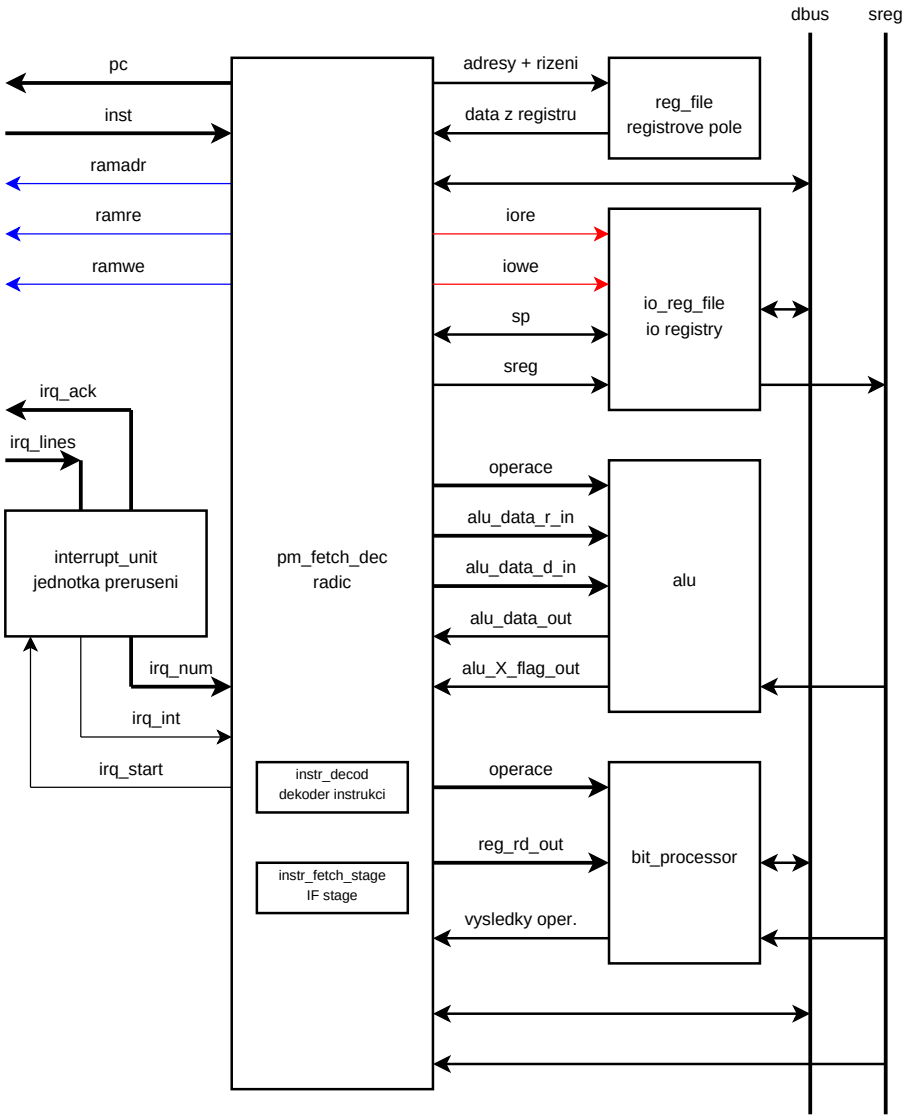
Piny INT7 až INT4 jsou vzorkovány ještě předtím, než se provádí rozpoznávání detekce hrany. Přerušení je generováno s ohledem na délku hodinového pulsu.

2.4.5 AVR jádro

Vlastní jádro procesoru se skládá z řadiče (`pm_fetch_dec`, viz 2.4.6), ke kterému je připojeno registrové pole (`reg_file`, viz 2.4.7), IO registry (`io_reg`, viz 2.4.8), logická jednotka (alu, viz 2.4.9), bitový procesor (`bit_processor`, viz 2.4.10) a jednotka obstarávající přerušení (`interrupt_unit`, viz 2.4.6.4). Jádro procesoru je propojeno s periferiemi pomocí třístavové datové sběrnice, adresové sběrnice a řídících signálů (zápis, čtení). Řídící signály jsou rozděleny podle toho, zda-li jde o IO prostor nebo prostor datový.

2.4.6 Řadič

Vlastní řadič obsahuje jednotku programového čítače 2.4.6.1 pro tvorbu adresy další instrukce. Tato instrukce se vybere z programové paměti a uloží se do registru (`instruction_code_reg`). Na instrukční registr je přímo připojen dekodér instrukcí 2.4.6.3. Po rozdekódování instrukce se nastaví signál (`idc_xxxx`), který indikuje, o kterou instrukci (`xxxx`) šlo.
Signál `nop_insert_st` je aktivní v případě, že se jedná o vícetaktovou instrukci a je tedy potřeba zastavit vykonávání další instukce tím, že se místo ní vloží instrukce nop (no operation).
Jelikož má procesor specifický přístup k vstupně-výstupním operacím (IO) a k datové paměti, jsou zavedeny dva druhy sběrnic. Pro komunikaci s IO zařízeními (adresy 0x0020 až 0x005F) je

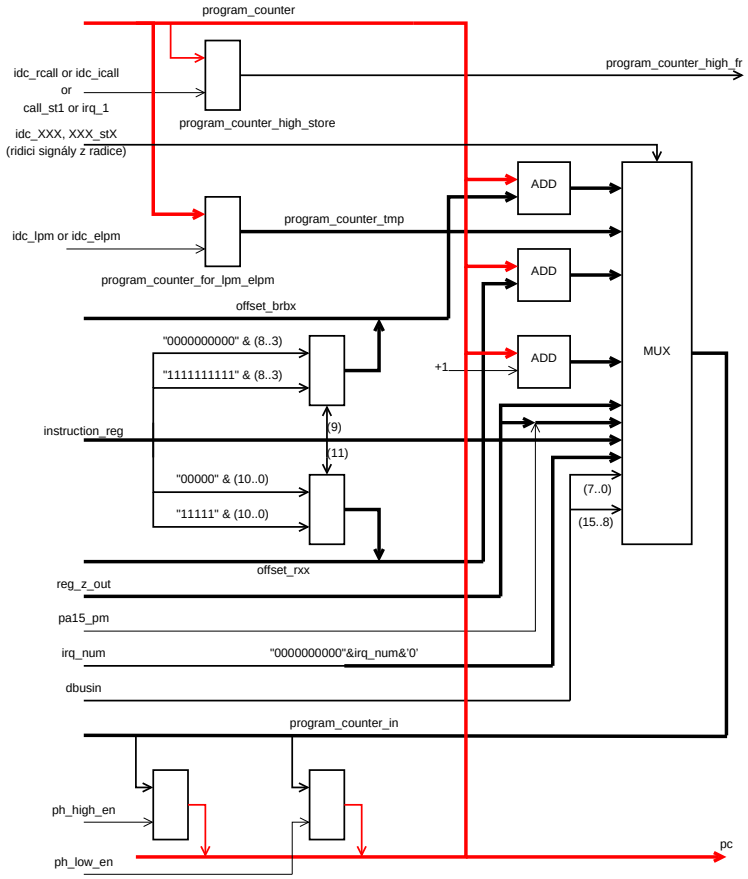


Obrázek 2.9: AVR jádro obsahuje jednotky registrového pole (r0-r31), vstupně-výstupní registry pro řízení vlastního jádra procesoru, aritmetickologickou jednotku a bit procesor, určený k výpočtu a jednotku přerušení, obstarávající výběr přerušení dle priority.

adresová sběrnice (`adr`) a řídící signály (`iore`, `iowe`). Pro adresování prostoru, kde se nachází datová paměť (adresy 0x0060 až 0xFFFF) je adresová sběrnice `ramadr` a řídící (`ramre`, `ramwe`). Jednotlivé datové cesty se řídí pomocí prováděných instrukcí. Pokud je instrukce vícetaktová, je rozložena do více úkonů. To zabezpečují konečné automaty. Signály těchto automatů mají definované názvy následujícího tvaru. Pro první takt je název `nXXXX_st0`, pro další `XXXX_st1`, `XXXX_st2` a `XXXX_st3`.
Pro zastavení mikrořadiče je každý registr doplněn o clock enable signál (`cpuwait`), který obstarává zápis nové hodnoty do registru v případě, je-li signál `cpuwait` v log. 0.

2.4.6.1 Programový čítač

Programový čítač (PC, Program Counter) slouží v procesoru k vytváření adresy další zpracovávané instrukce. Tento čítač je reprezentován 16bitovým registrem (v našem případě dvěmi 8bitovými registry). Převážnou část běhu programu se tento čítač inkrementuje o jedna. V případě skoku v programu, skoku do podprogramu nebo skoku do přerušovací rutiny, se hodnota čítače nastaví přímo na adresu dané instrukce programu, podprogramu nebo na adresu přerušovací rutiny.



Obrázek 2.10: Programový čítač (PC) je 16bitový registr, ale v principu obsahuje dva 8 bitové registry. Na zásobník můžeme uložit pouze 8bitovou hodnotu, proto je nutné mít PC rozdělené do dvou registrů. Ukládání hodnoty PC se tedy provádí ve dvou fázích. Nová hodnota PC se načítá z různých zdrojů a mezi těmito zdroji se přepíná pomocí multiplexoru. Standartní situace je, kdy se k aktuální hodnotě přičte jedna.

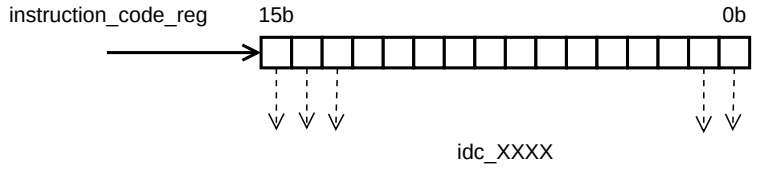
První skupinou instrukcí jsou skoky v programu. Z hlediska bloku `pc_next` je nejdůležitější

rozlišení na skoky relativní a absolutní. Při relativním skoku se vezme aktuální hodnota PC a k ní se přičte relativní část z instrukce. Při absolutním skoku je celá část nové hodnoty PC převzata z instrukce.
Další skupinou je změna programového čítače při skoku do podprogramu, či při vykonávání rutiny přerušení. Při skoku do podprogramu se nejprve musí uložit hodnota PC na zásobník (spodní a poté horní bajt), poté se nastaví PC na novou hodnotu. Při návratu se vybere hodnota ze zásobníku a díky rozdělení PC na dva 8bitové registry se opět poskládá nová hodnota programového čítače.
Při příchodu přerušení (signál `irg_num`) se opět nejdříve uloží hodnota PC na zásobník a poté skočí na adresu (dvojnásobek čísla přerušení) rutiny přerušení. Po návratu z přerušení se opět vybere hodnota PC ze zásobníku.
Další funkcí této jednotky je tvorba adresy pro adresování programové paměti instrukcemi LPM a ELPM, viz. 2.4.8 registr RAMPZ.

2.4.6.2 Cpuwait

Signál `cpuwait` je odvozován od systémových hodin. Většinu času je signál nastaven na hodnotu log. 1. V tomto stavu je jádro procesoru zastaveno. Po načítání určitého množství taktů, které je dáno konstantou *FcpuDivider* (`top_avr_core_sim`), se signál `cpuwait` vynuluje na dobu jednoho taktu systémových hodin. V tomto stavu procesor vykonává instrukci. Rychlost vykonávání instrukcí je dána dělícím poměrem ($\frac{F_{osc}}{2^{1+FcpuDivider}}$), kde F_{osc} je frekvence systémových hodin .

2.4.6.3 Instrukční dekodér



Obrázek 2.11: Instrukční dekodér je tvořen pouze kombinační jednotkou o 16 vstupech a výstupem je rozkódovaná instrukce.

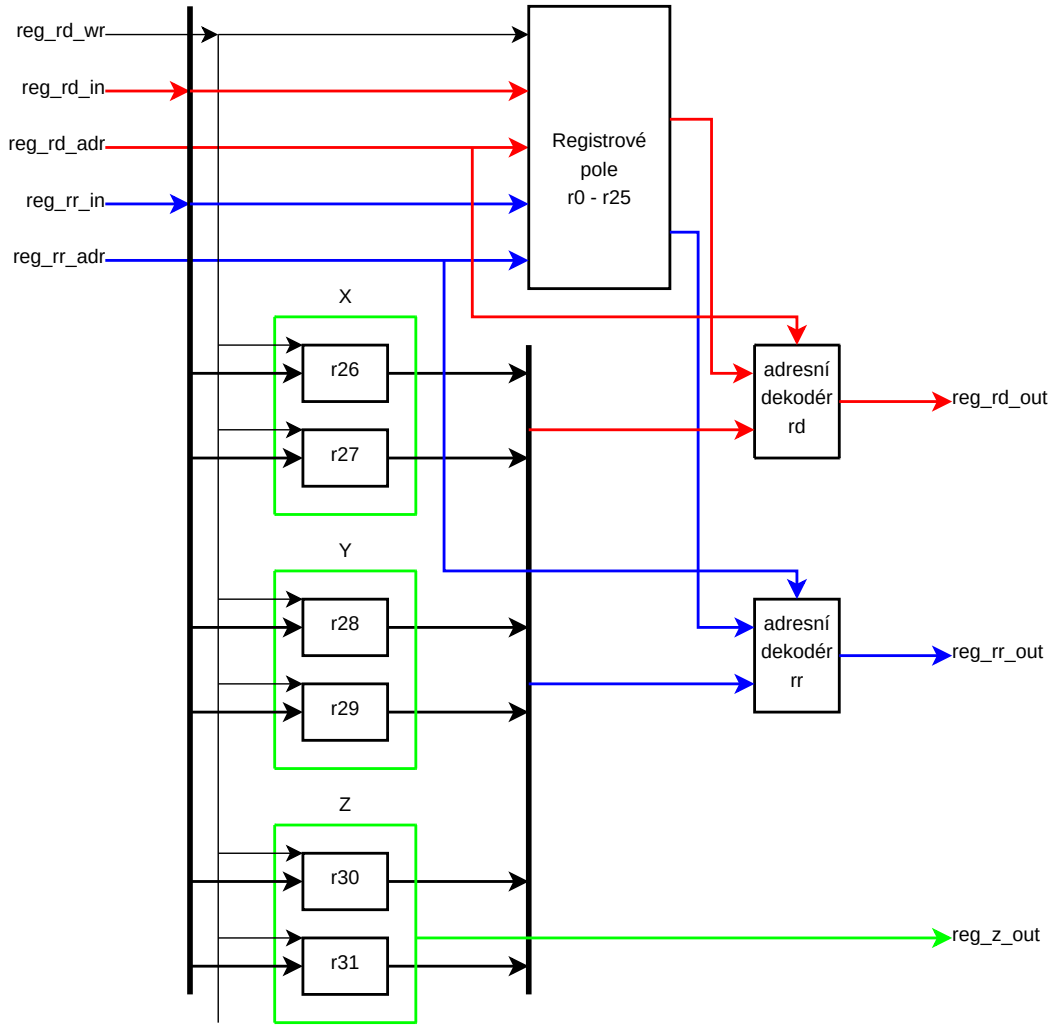
Tento blok je jednoduchá kombinační jednotka, která má dva druhy výstupů. První výstupy jsou signály udávající, o kterou instrukci se jedná (signály typu `idc_xxxx`), druhé výstupy jsou přímé hodnoty konstant (přímá adresa, hodnota).

2.4.6.4 Přerušení

Při požadavku na přerušení (`irq_req`) se nejprve vyhodnotí Interrupt bit ze stavového registru (SREG) a signál `cpu_busy` (ten je nastaven při provádění vícecyklových instrukcí). Pokud je splněna výše uvedená podmínka povolení přerušení, spustí se konečný automat o čtyřech stavech a řadič odpoví signálem `irq_start`, že se spustil přerušovací systém.
V prvním kroku konečného automatu se zastaví načítání PC (vloží se NOP instrukce). Ve druhém kroku se provede zápis hodnoty PC na zásobník (nejdříve nižší Bajt) a nová hodnota PC se nastaví podle vektoru přerušení. Třetí krok ukládá vyšší Bajt ze zálohy programového čítače na zásobník a ukončuje do něj zápis. V dalších krocích se zpracovává rutina přerušení.

2.4.7 Registrové pole

Registrové pole je sada 32 univerzálních registrů 2.12. Prvních 26 registrů jsou ve fyzickém návrhu realizovány pomocí block RAM. Z důvodu přímého přístupu k registrům r26 až r31 jsou tyto registry implementovány jako skutečné registry (D klopné obvody).

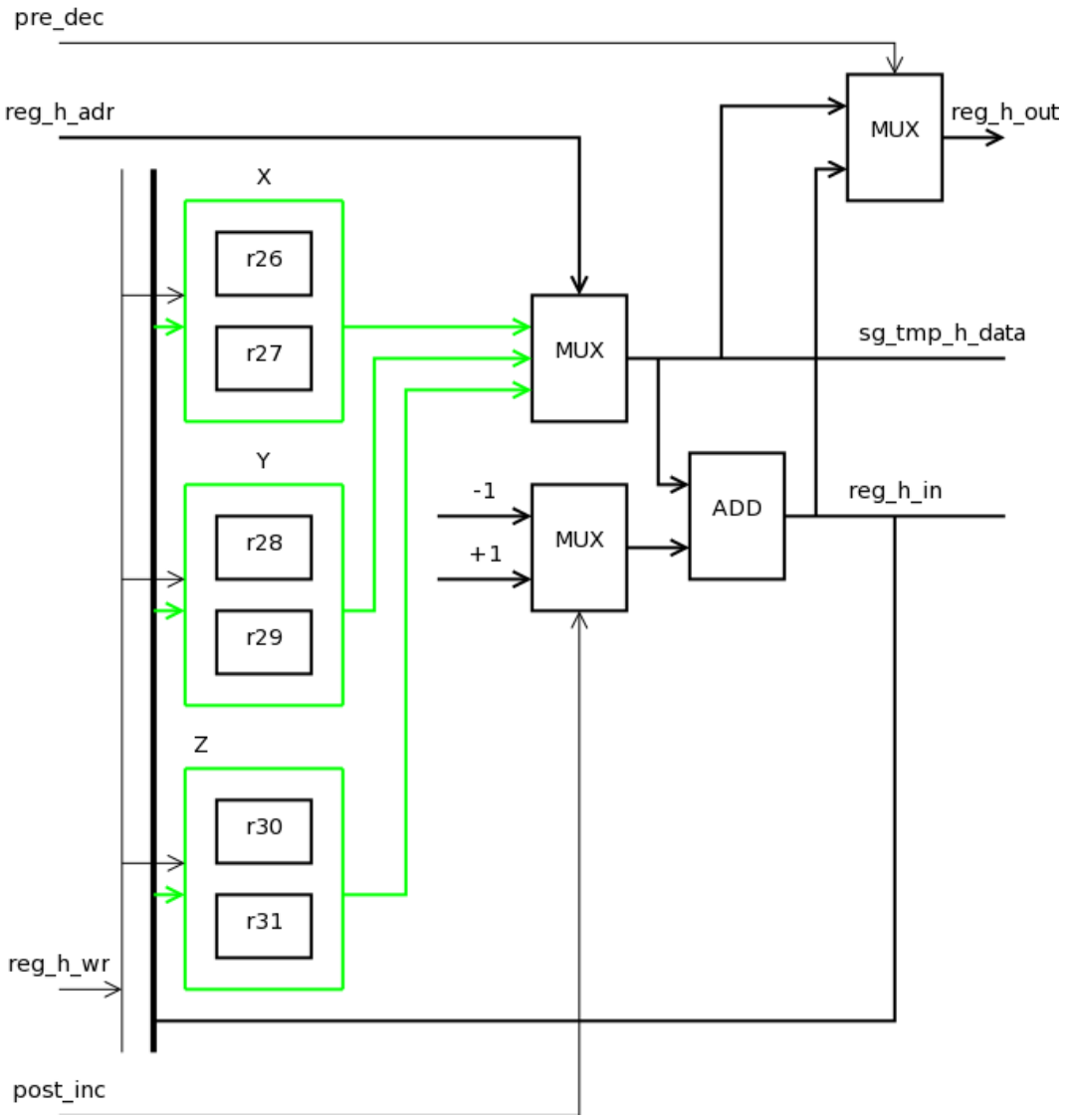


Obrázek 2.12: Registrové pole obsahuje registry r0 až r25, uložené v block RAM a r26 až r31 tvořené 8 bitovými registry, které je možné po dvojicích spárovat.

Následuje tedy šest 8bitových registrů 2.13, resp. tři 16bitové registry X, Y, Z. Tyto registry se používají k adresování do datové paměti. Jen registr Z se používá k přímému adresování instrukční paměti (instrukce LPM a ELPM).
Jelikož procesor je možné zastavovat, je nutno provést post-inkrementaci a pre-dekrementaci jen v případě, je-li cpuwait v log. 0.

2.4.8 IO registry

Tato jednotka 2.14 obsahuje ovládání registru SREG, SPH, SPL a RAMPZ. Registry XDIV (XTAL Divide Control Register), MCUCR (MCU General Control Register) a MCUSR (MCU Status Register) byly vynechány z důvodů nevyužitelnosti a ušetření místa na FPGA.



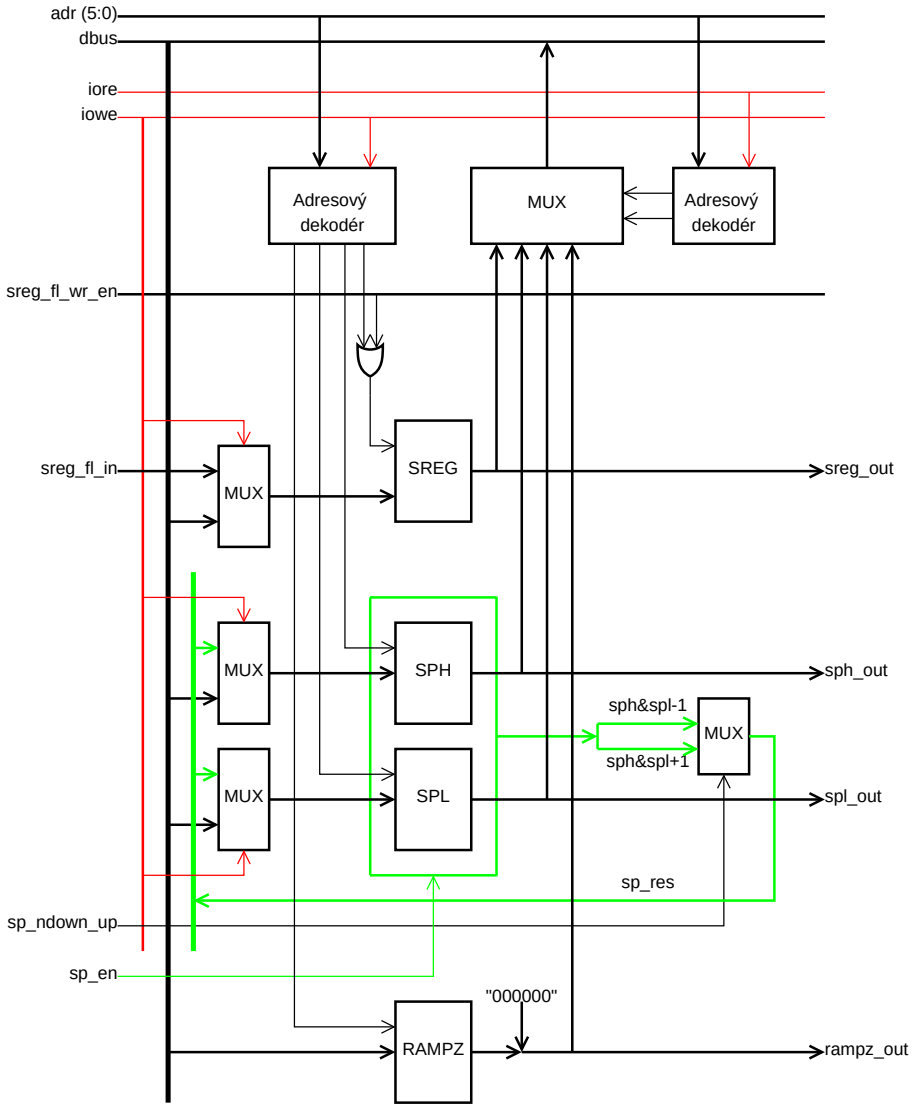
Obrázek 2.13: Post inkrementace a predekrementace registrů X, Y, Z

SREG - Status REGister

Bit	7	6	5	4	3	2	1	0
0x3B	I	T	H	S	V	N	Z	C
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

- 7. - I (Global Interrupt Enable)
  Globální povolení přerušení od jakékoliv periferie či externího přerušení (podmíněno dílčím povolením od daného signálu). Je-li globální povolení v log. 0, je celý přerušovací systém blokován a nemůže být vykonáno přerušení.

- 6. - T (Bit Copy Storage)
  Tento bit používají instrukce BLD (Bit LoaD) a BST (Bit STore). Pomocí těchto instrukcí je možno uložit či nastavit daný bit u jakéhokoliv registru v registrovém poli (viz. bit procesor 2.4.10).



Obrázek 2.14: Tento blok IO registrů obsahuje SREG, což je registr uchovávající aritmeticko-logické příznaky a bit povolení přerušení, ukazatel na vrchol zásobníku (dva 8 bitové registry SPL, SPH) a RAMPZ, který určuje stánku paměti při čtení z paměti programu.

- 5. - H (Half-carry Flag)
  Jedná se o příznak poloviního přenosu, kdy dojde k přenosu mezi 3. a 4. bitem.
- 4. - S (Sign Bit, $S = N \oplus V$)
  Znaménkový bit, je vytvořen jako non-ekvivalence příznaku negace a příznaku přetečení u operací s dvojkovým doplňkem.
- 3. - V (Two's Complement Overflow Flag)
  Příznak přetečení u dvojkového doplňku.
- 2. - N (Negative Flag)
  Jde o MSB ve výsledné hodnotě (viz. logická jendotka **??**).
- 1. - Z (Zero Flag)
  Indikuje, že výsledek aritmetické operace je roven nule.
- 0. - C (Carry Flag)
  Příznak přenosu po aritmetické nebo logické operaci.

SP - Stack Pointer Ukazatel do zásobníku je 16bitový registr, který fyzicky tvoří dva 8bitové registry (High, Low). Umožňuje adresovat 64KB datové paměti. Tento zásobník se používá k uložení programového čítače při skoku do podprogramu, či při provádění přerušení. SP musí být inicializován na adresu vyšší než 0x60 (v adresovém prostoru datové paměti 0x00 až 0x60 jsou namapovány IO periferie). Při provádění instrukce PUSH se na zásobník uloží data a dekrementuje se SP. Pokud se jedná o volání podprogramu, či obsluhu přerušení tak se dekrementuje dvakrát (programový čítač je 16bitů). Při instrukci POP se ze zásobníku přečtou data a SP se automaticky inkrementuje. Je-li potřeba návrat z podprogramu (RET) nebo z obsluhy přerušení (RETI), tak se inkrementuje taktéž nadvakrát.

Bit	7	6	5	4	3	2	1	0
0x3E	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
0x3D	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

RAMPZ - RAM Page Z Select Register

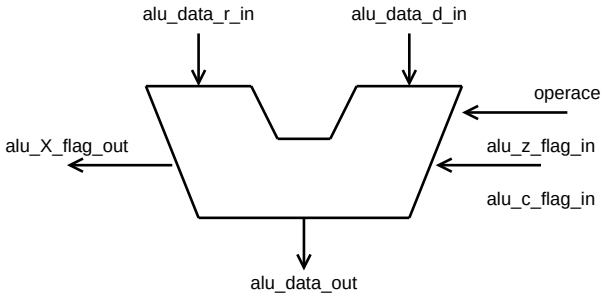
Bit	7	6	5	4	3	2	1	0
0x3F	-	-	-	-	-	-	-	RAMPZ0
čtení/zápis								rw
inic. hodnota	0	0	0	0	0	0	0	0

- 0. bit - RAMPZ0
  Tento bit určuje, na kterou 64KB stránku budeme přistupovat pomocí instrukce ELPM. Bit udává 15bit adresy do programové paměti (viz. 2.4.6.1).

RAMPZ0	Rozsah adresového prostoru přístupného přes ELPM
0	0x0000 - 0x7FFF
1	0x8000 - 0xFFFF

2.4.9 Aritmetickologická jednotka (ALU)

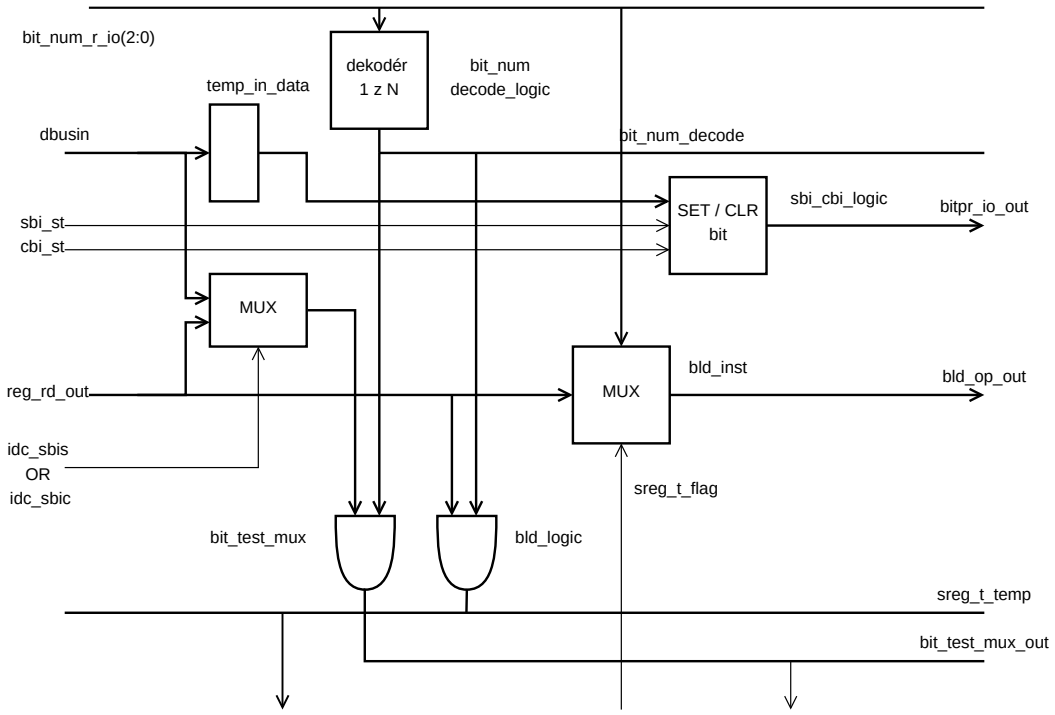
Do tohoto bloku vstupují dva datové kanály z registrového pole (`alu_data_r_in` a `alu_data_d_in`). Vlastní jednotka obsahuje sčítačku (aritmetické operace) a posuvný registr (logické operace) a kombinační logiku pro nastavení příznaků. Další logické operace provádí bitový procesor 2.4.10.



Obrázek 2.15: Aritmetickologická jendotka

2.4.10 Bitový procesor

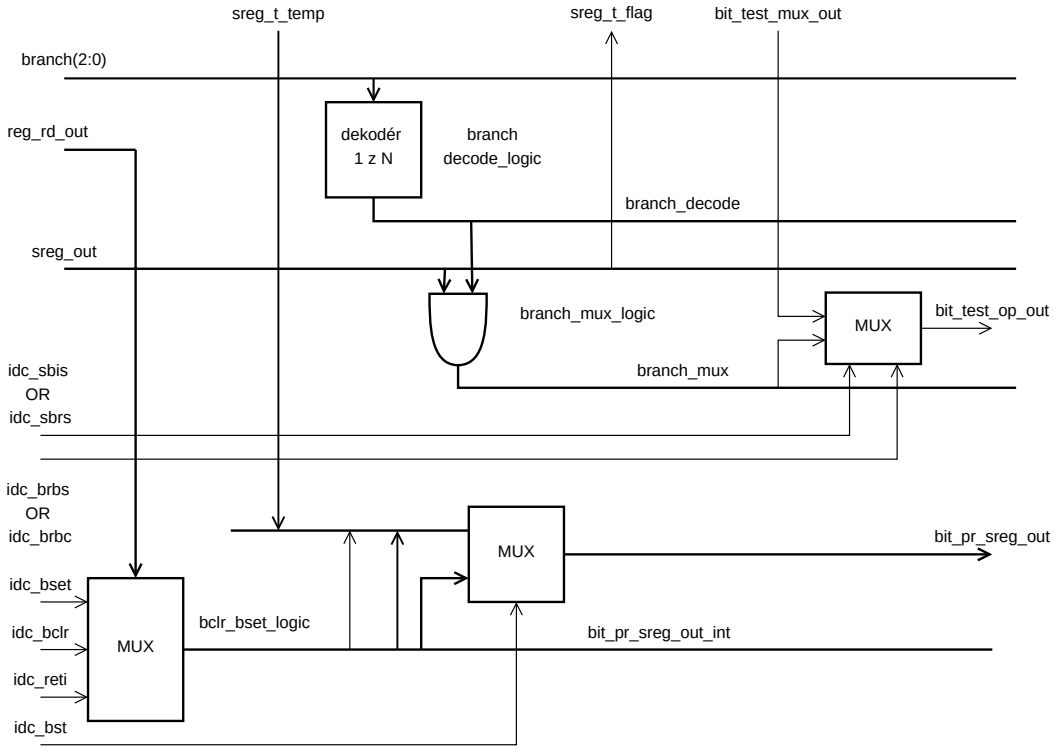
Tato jednotka (2.16, 2.17) je určena pro práci s jednotlivými bity nad registry a práci s nimi.



Obrázek 2.16: Tato část bitového procesoru umožňuje daný bit nastavit či vynulovat.

Vstupními signály jsou:

- bit_num_r_io a branch - určuje se kterým bitem z registru (univerzální, SREG) budeme pracovat.



Obrázek 2.17: Druhá část bitového procesoru slouží k vyhodnocení podmínky u instrukce podmíněného skoku (rozhodování dle daného bitu).

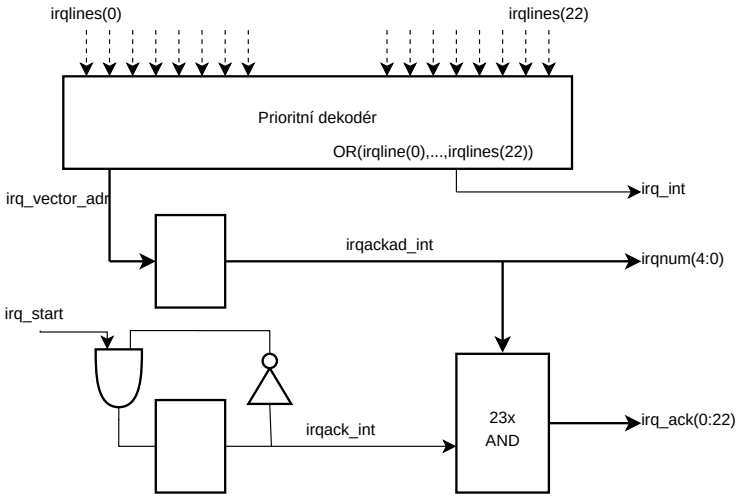
- dbusin - jediný vstup z venku, kdy pomocí instrukce SBI (Set Bit in IO Register) lze nastavit bit v IO registru, či pomocí instrukce CBI (Clear Bit in IO Register) vynulovat daný bit určený signálem `bit_num_r_io`.
- reg_rd_out - slouží jako vstup z registrového pole.
- sreg_out - aktuální hodnota SREG (Status REGister), tj. registru s flagy.

Výstupy jsou:

- bitpr_io_out - SBI, CBI aplikované na IO registry.
- bld_op_out - slouží při instrukci BLD k uložení T flagu z SREG na místo bitu daného signálem `bit_num_r_io`.
- bit_test_op_out - rozhodování skoku za podmínky určené bitem v IO registru (instrukce SBIS) či univerzálním registru (SBRS), nebo v závislosti na daném flagu z SREG (instrukce BRBS).
- bit_pr_sreg_out - slouží k nastavování nové hodnoty SREG (instrukce BSET, BCLR), nastavení I flagu při dokončení přerušovací rutiny (RETI) a ukládání daného bitu (`bit_num_r_io`) do T flagu v SREG.

2.4.11 Jednotka přerušení

Tento blok (2.18) slouží k obsluze přerušení, odskoku na podprogram a odpovědi periferii, že se dané přerušení obsloužilo. Při výskytu více požadavků o přerušení je tato jednotka vybavena prioritním dekodérem. Procesor tedy obslouží jen jedno přerušení.



Obrázek 2.18: Tato jednotka přerušení obstarává výběr zdroje přerušení s nejvyšší prioritou a taktéž odpověď (irq_ack), že daná rutina přerušení byla zpracována.

Při přijetí přerušovacího signálu irqlines se výstup `irq_int` nastaví do log. 1. Dále se na sběrnici irqnum objeví číslo přerušení (pro odvození adresy v tabulce přerušení), výběr daného přerušení je dán prioritním dekodérem. Vyšší prioritu mají přerušení s nižším číslem přerušení (tj. LSB v `irq_lines` má nejvyšší prioritu). Když program obslouží přerušení, vygeneruje se signál `irq_start`. Ten způsobí, že v bloku s logickým součinem se vygeneruje odpověď pro periférii (dané přerušení bylo obslouženo).

2.5 Testování

Pro potřeby testování je nutný zdrojový kód, vytvořený buď v jazyku symbolických adres, či jazku C, nahrát do přípravku. Překlad z ihex formátu do a VHDL (PROM.vhd) umožňuje program hex2vhd. Vzniklý soubor PROM.vhd můžeme přeložit spolu s celým řadičem a nahrát jako bitstream do FPGA obvodu. Existuje i druhá varianta, kdy je možné programem `uploader` (byl vytvořen v rámci této diplomové práce) nahrát `ihex` soubor do externí paměti SRAM. Za pomoci konstanty `simulation` v top modulu můžeme vybrat paměť, odkud bude program načítán (z PROM.vhd, block RAM, externí SRAM).
V této fázi testování bylo využíváno souboru PROM.vhd. Po vytvoření programu a jeho převedení do VHDL mohlo následovat testování v modelsim [3].
Pro otestování byly vytvořeny testovací programy v jazku symbolických adres (různé druhy adresování, skoky, přerušení, práce se zásobníkem) a poté byly odzkoušeny i větší projekty v jazyku C. AVR jádro se jevilo plně funkční.

AVR jádro v konečné verzi má tyto vlastnosti:

- 121 výkonných instrukcí
- 32 8bitových registrů
- 128KB paměti programu (16-bitová adresová sběrnice a 2B instrukce)
- 4KB interní RAM paměť (je zde možnost rozšíření)
- více druhů přerušení (14x vnitřní, 8x vnější)

- maximální možná frekvence na přípravku 12,5MHz (takt jádra)
- zaplněnost FPGA je 1583 LUTs (852 CLB)
- využívá 1x block RAM pro registrové pole a 2x block RAM pro datovou paměť

2.6 Závěr

Výsledkem této části je funkční AVR jádro mikrořadiče ATmega 103. Programování mikrořadiče je možné buď přes sériové rozhraní (viz. druhá část diplomové práce), či pomocí nahrání programu přímo ve VHDL kódu.
Během této práce bylo nejprve nutné se seznámit s danou problematikou návrhu mikrořadiče dostupné na internetových stránkách opencores [12]. Bylo nutné provést zpomalování procesoru pro účely testování v reálném hardwaru. Poté bylo zjištěno, že design zabírá příliš mnoho místa na čipu FPGA obvodu. Proto se muselo přistoupit k optimalizaci navrženého designu. Veškeré paměťové bloky byly přesunuty do block RAM a tím byla snížena prostorová náročnost na polovinu. Datová sběrnice byla předělána na třístavovou a tím se zjednodušilo připojování dalších periférií.
Pro potřeby nahrávání programu do programové paměti byl vytvořen program pod operační systém Linux (`uploader`), který slouží k poslání ihex souboru do SRAM paměti přes sériové rozhraní.
K ověření mikořadiče bylo vytvořeno několik testovacích programů, které jsou přiloženy na CD jako zdrojové soubory.

3 Periferie

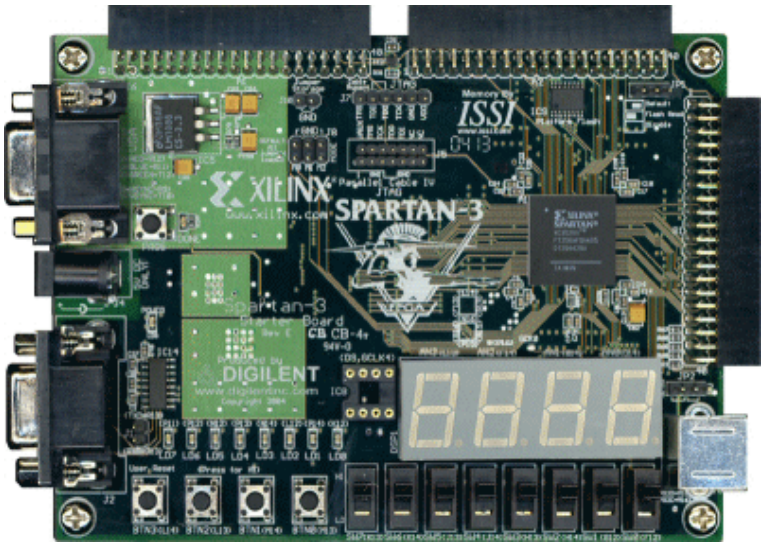
3.1 Úvod

Abychom mohli komunikovat či jinak přistupovat k vnějšímu světu, musíme doplnit mikrořadič o periferie. V této kapitole bude popsána vývojová deska Starter kit [8] a její dostupné periferie.

3.2 Analýza

3.2.1 DIGILENT Starter kit

Vývojová deska [8] od firmy Digilent je postavena na hradlovém poli FPGA Spartan 3 [13]. Obsahuje nejvíce používané periferie pro snadný vývoj aplikací. Pro další připojení periferií je deska vybavena třemi rozšiřujícími konektory.



Obrázek 3.1: DIGILENT Starter kit

Deska je osazena těmito částmi:

- Hradlové pole Spartan 3 obsahuje 200 tis. hradel (XC3S200FT256) - (viz 2.2.4)
- Konfigurační flash paměť XCF02S, 2Mbit sloužící k trvalému uložení konfiguračního souboru (bit stream). Při správné konfiguraci (jumper JP1) je FPGA obvod nakonfigurován ihned po přivedení napájení.
- Dvě rychlé SRAM 256k x 16b, 10ns - (viz 3.2.2)
- Port VGA - (viz 3.4.6)
- Port RS232 - UART 3.4.4
- Port PS/2 - (viz 3.4.5)
- Čtyřmístný, sedmisegmentový displej - (viz 3.4.2)
- 8x LED - (viz 3.4.1)
- 8x posuvný spínač

- 4x tlačítko
- Taktovací oscilátor 50MHz
- Patice pro další oscilátor
- Tři 40 pinové rozšiřující konektory (viz 3.2.8)

3.2.2 SRAM

Deska je osazena dvěmi moduly SRAM paměti. Každý z těchto modulů je v konfiguraci 256k x 16b. Celková možná kapacita přípravku je tedy 1MB. Oba obvody mají sdílené signály povolení zápisu (Write Enable, $\overline{WE}$), povolení čtení/výstup (Output Enable, $\overline{OE}$) a adresové vodiče (šířka adresové sběrnice je 18 bitů). Toto omezení znesnadňuje použití každého z modulů pro jinou aplikaci. Oba moduly mají separovány řídící signály a to pro výběr obvodu (Chip Enable, $\overline{CE}$) a bajtově orientované (byte enable), které slouží k výběru daného bajtu (datová sběrnice má šířku 2B).

3.2.3 LED

Deska poskytuje výstup pro 8 LED, které jsou připojeny přímo k FPGA obvodu přes sériové rezistory. LED svítí, pokud se na výstupu FPGA vyskytne logická 1.

3.2.4 Display - výstup na 7-segmentovky

Vývojový kit je osazen čtyřmi 7-segmentovkami se společnou katodou, která je spínána přes tranzistor. Pokud chceme přistupovat k určitému řádu na display je nutno nastavit daný pin na log. 0. Řády jsou multiplexovány a pro rozsvícení je nutné mít daný vstup v log. 0. Pro zobrazení znaků na všech 4 místech je nutné provozovat display v režimu časového multiplexu, kdy se rychle přepínají řády a pro každý řád se zobrazuje ta hodnota, která má bý na daném místě vidět. Toto se musí dít s dostatečnou frekvencí, aby lidské oko nepocítilo blikání displaye. Podle doporučení se používá frekvence kolem 1,5kHz.

3.2.5 RS-232

Na desce je osazen obvod MAX232, který slouží ke konverzi mezi TTL úrovněmi FPGA obvodu a úrovněmi RS232, které poskytuje toto rozhraní (například počítač). Deska má konektor DB9, ke kterému můžeme připojit prodlužovací kabel a propojit desku s počítačem. Vyvedeny jsou pouze datové vodiče (Rx - přijímaná data, Tx - vysílaná data). Komunikace na sběrnici je klasická sériová komunikace, kde v klidu je signál v log. 1. Při zahájení přenosu se nejprve vyšle start bit (log. 0), od kterého se synchronizuje spojení. Po dobu převrácené doby přenosové rychlosti je signál v log. 0, a poté jsou odesílána data. Následuje stop bit (log. 1). Tato sekvence (start, data a stop bit) se může nekonečněkrát opakovat.
Při vysílání se nejprve vyšle nejnišší bit (LSB) a jako první se taktéž přijme.

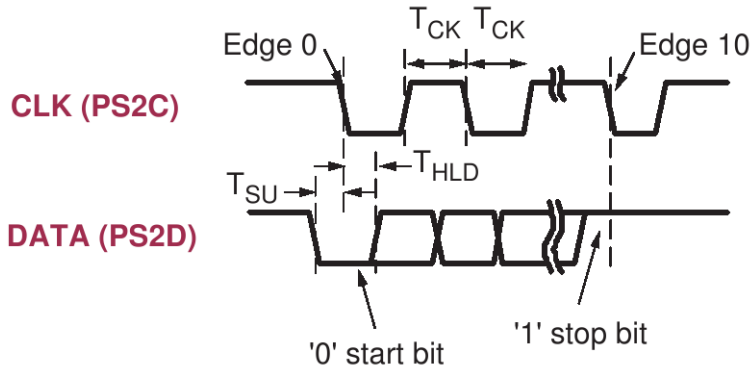
3.2.6 PS/2

Spartan 3 Starter Kit obsahuje konektor pro připojení PS/2 myši a klávesnice přes standartní 6ti pinový mini-DIN konektor. Do FPGA obodu je přiveden vodič od datového a hodinového signálu. Jak myš, tak i klavesnice řídí tuto sběrnici protokolem (PS/2). Ten určuje, že se budou posílat 11bitová slova obsahující start, stop a paritní bit (lichá parita). Pro načítání z klávesnice je potřeba jen jednosměrná komunikace a proto se v dalším textu i v návrhu budu zabývat jen jí.

Symbol	Parametr	Min.	Max.
T_{CK}	šířka pulsu	30 μs	50 μs
T_{SU}	délka náběžné hrany	5 μs	25 μs
T_{HLD}	zpoždění dat za hodinami	5 μs	25 μs

Tabulka 3.1: Časování PS2

3.2.6.1 Časování



Obrázek 3.2: PS/2 - časování

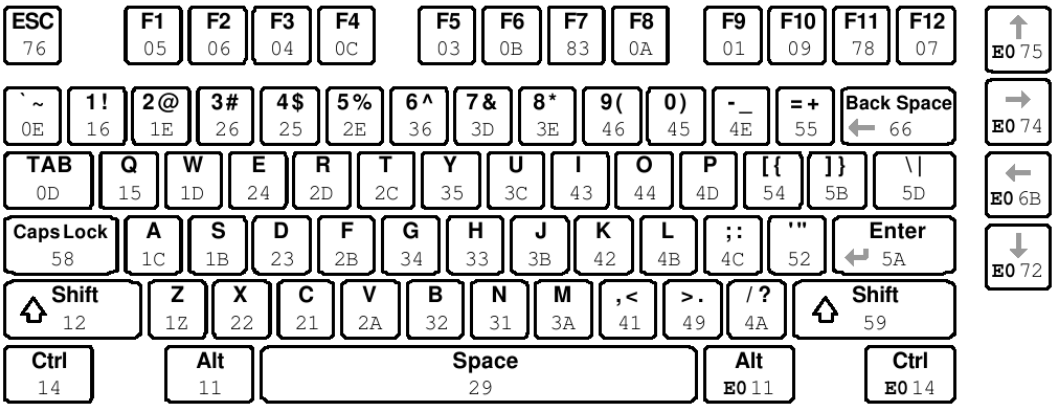
Rozhraní PS/2 nabízí dva signály a to data a hodiny. Oba tyto vodiče jsou v klidu připojeny přes pullup rezistory k napájecímu napětí. V obecném případě může jak host (FPGA), tak připojené zařízení (klávesnice, myš) budit oba tyto signály. V našem případě potřebujeme pouze načítat hodnoty z klávesnice, proto bude stačit, když host bude jen načítat vstupní datový signál a signál hodin. Na obrázku 3.2 je zobrazen komunikační protokol. Připojené zařízení (klávesnice, myš) zapisuje na datový vodič v okamžiku, když je hodinový signál `ps2_clk` v log. 1. Host (FPGA) čte z datového vodiče, pokud je hodinový signál v log. 0.

3.2.6.2 Klávesnice

Klávesnice připojená pomocí PS/2 sběrnice může fungovat jako vysílač i jako přijímač. My ji budeme používat pouze jako vysílač dat a hodinového signálu.
PS/2 klávesnice používají scan kód k popisu, jaká klávesa byla stisknuta či puštěna. Každá klávesa má unikátní scan kód, který je vyslán při stisku klávesy. Na obrázku 3.3 je nejobvyklejší rozložení kláves a jejich scan kódy.
Pokud je klávesa stisknuta a držíme ji, klávesnice opakovaně vysílá scan kód a to přibližně každých 100ms. Jestliže klávesu pustíme, klávesnice pošle "F0" key-up kód a za ním následuje scan kód.
Některé klávesy, patřící do skupiny "extended", posílají "E0" na začátku scan kódu a mohou posílat více než pouze jeden scan kód. Pokud je tato rozšířená klávesa puštěna, klávesnice vyšle sekvenci "E0 F0" s následným scan kódem.

3.2.7 VGA

Deska 3.1 obsahuje VGA rozhraní s výstupem signálů pro vertikální (`vga_vs`) a horizontální (`vga_hs`) synchronizaci a barvami červená (`vga_r`), zelená (`vga_g`), modrá (`vga_b`). Tyto sig-



Obrázek 3.3: PS/2 Rozložení kláves a jejich scan kód

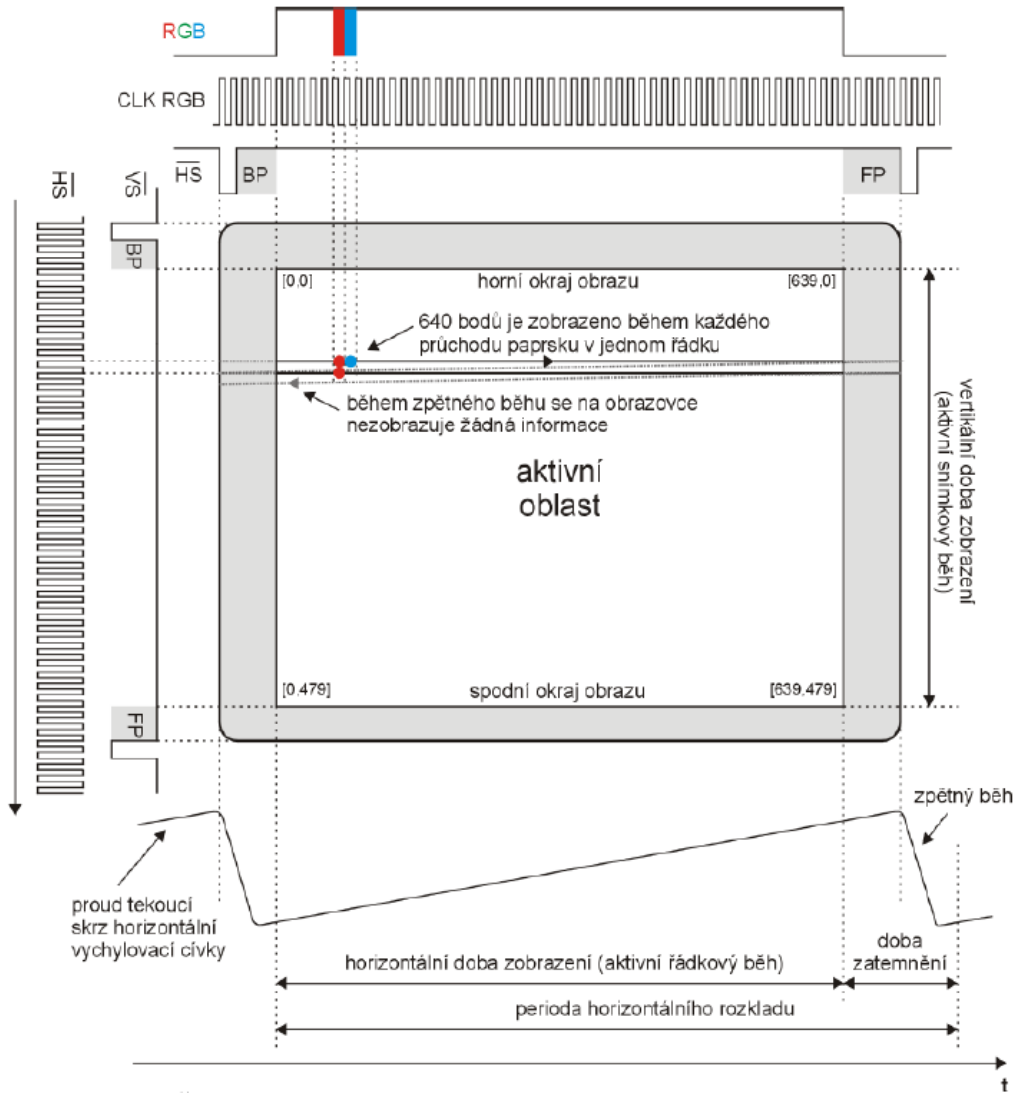
Rozlišení	Velikost video paměti
320x240, mono	76 800b
320x240, color	230 400b
640x480, mono	307 200b
640x480, color	921 600b

Tabulka 3.2: V uvedené tabulce je uvedeno, jak velká paměť je potřebná pro dané rozlišení a barevnost.

nály jsou vyvedeny na standartizovaný konektor DB15, ke kterému můžeme připojit VGA monitor (CRT, LCD).
Pro zobrazování na VGA potřebujeme mít v určité paměti uložen zobrazovaný obraz. Na uvedené vývojové desce je externí paměť SRAM, nebo je možné využít block RAM uvnitř FPGA obvodu.

3.2.7.1 Časování VGA - 640x480, 60Hz

CRT a odvozené VGA monitory používají amplitudovou modulaci k vychýlení elektronového paprsku (z katodového děla). Tento paprsek poté dopadá na stínítko obrazovky, kde se objevují dané informace. LCD monitory jsou založeny na matici jednotlivých pixelů, které přímo zobrazují scénu. Aby byla dodržena zpětná kompatibilita LCD monitorů s CRT, tak oba monitory používají stejný model časování.
Pro řízení zobrazování monitoru jsou dva signály a to pro horizontální a vertikální vychylování. Tyto signály řídí vychylovací cívky u CRT monitorů. Obrázek 3.4 [10] ukazuje, jak se vykreslují jednotlivé pixely na obrazovku. Horizontální synchronizace slouží k řízení průběhu proudu cívkou, která ovlivňuje paprsek v horizontálním směru. Tedy vykresluje jednotlivé řádky, tj. paprsek se pohybuje zleva doprava (dopředný běh, Front Porch) a poté až dorazí na konec obrazovky, tak se paprsek přesune opět do pravé části stínítka (zpětný běh, Back Porch). Pro odstartování zpětného běhu je zaveden synchronizační impuls (`vga_hs`), který musí být generován námi připojeným zařízením. Tento stejný proces se děje i ve vertikálním vychylování, ale s pomalejší frekvencí (v našem případě 60Hz) a pro synchronizaci vertikálního vychylování je zde signál `vga_vs`.
Pokud použijeme standartní rozlišní 640x480 bodů při obnovovací frekvenci 60Hz, musíme dodržet časování uvedené v tabulce 3.3. Obrázek 3.12 vyobrazuje vztahy mezi jednotlivými



Obrázek 3.4: VGA - časování

Parametr	Vertikální synchronizace		horizontální synchronizace	
	délka pulsu	počet taktů	délka pulsu	počet taktů
Délka synchronizačního pulsu (TOTAL)	16,7 ms	521	32 μs	800
Čas zobrazení (ACTIVE)	15,36 ms	480	25,6 μ s	640
Šířka pulsu (SYNC)	64 μs	2	3,84 μs	96
Dopředný běh (FRONT PORCH)	320 μs	10	640 ns	16
Zpětný běh (BACK PORCH)	928 μs	29	1,29 μs	48

Tabulka 3.3: Časování VGA - 640x480, 60Hz

symboly.

3.2.8 Rozšiřující konektory

Deska poskytuje tři rozšiřující konektory, na které je možné připojit další periferie. Jelikož mikrořadič poskytuje vstupně-výstupní bránu (viz 3.4.3), bylo by možné využít tyto konektory právě pro tento účel.

3.3 Návrh řešení

3.3.1 LEDS

Jelikož se jedná pouze o výstupní periferii, bude nejsnažší napojit LEDky na výstup registru, kde vstupem bude datová sběrnice. Zápis do registru bude proveden při aktivním signálu `iowe` a správné adrese.

3.3.2 Display

Realizace displaye by mohla být provedena tak, že zobrazovaná data z registrů připojených ke sběrnicím, budou dále připojeny na vstupy BCD dekodéru. Z tohoto dekodéru povede 7+1 vodičů přímo k jednotce, která bude postupně připojovat tyto vodiče k jednotlivých řádům displaye a to s frekvencí přibližně 1,5kHz. Tato frekvence bude generována ve vnitřní děličce tohoto bloku. Ta bude dělit s poměrem $1 : 2^{15}$, viz:

$$\frac{50 \cdot 10^6}{1{,}5 \cdot 10^3} = 33,333 \doteq 32768 = 2^{15}$$

3.3.3 PORT

Tento blok by měl umět načíst hodnoty z daného pinu. K této realizaci bude stačit pouze registr, jehož vstupem budou jednotlivé piny. Dále by měl blok obsahovat registr určující směr (vstupní, výstupní pin). Tento registr bude pouze ovládat multiplexory, které buď registr odpojí či připojí k danému pinu. A nakonec bude tento blok obsahovat registr s hodnotou určující výstupní hodnotu na pinu.

3.3.4 UART

Jednotka bude rozdělena do dvou částí a to vysílací a přijímací, protože se jedná a fullduplex zařízení.
Přijímací část sériového portu musí obsahovat jednoduchý filtr na vstupním pinu. Ten bude realizován tak, že se načtou vždy postupně tři vzorky vstupního signálu a udělá se z nich

střední hodnota pomocí majoritní funkce. To by mělo zajistit dostatečné filtrování signálu. Této metody používá i běžný UART v ATmega 103.
Vysílací část není ničím zvláštní a proto bude obsahovat je nejnutnější posuvný registr a generátor přenosové rychlosti (tak jako přijímací část).
Obě části budou řídit dva nezávislé konečné automaty.

3.3.5 PS/2

Tato periferie se narozdíl od UART liší jen v tom, že přenosovou rychlost definuje připojené zařízení. Proto nebude nutné dělat generátor frekvence. Jen je nutné časový signálový vodič přivést do filtru a odstranit tak odrazy na vedení. Dále bude potřeba načítat data do posuvného registru s taktem přijímaných hodin. Celý systém bude opět řízen konečným automatem.

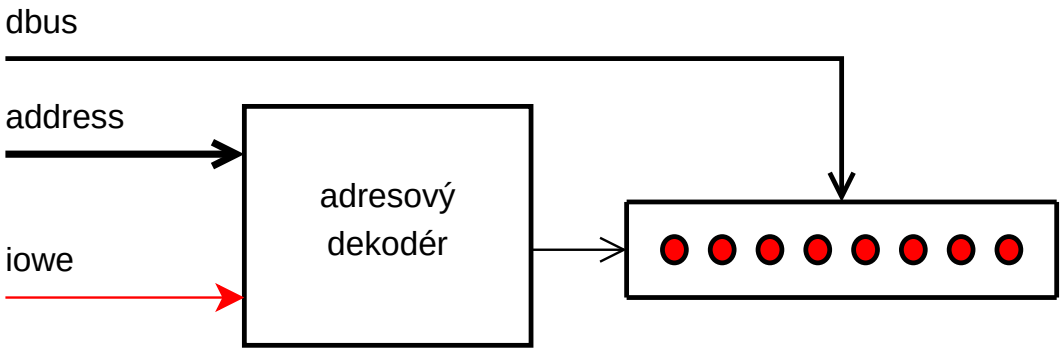
3.3.6 VGA

Blok VGA periferie by měl obsahovat jednotku, která se bude starat o časování VGA signálů. Dále zde bude ovládání a přístup k paměti a vyrovnávací buffer (posuvný registr), který bude dodávat hodnoty pro jednotlivé pixely. Poslední částí bude ovládací jednotka. K ovládání signálů pro načítání z bufferu je však možné použít i jednotku časování. Inspiraci pro tuto periferii lze nalézt na internetových stránkách Columbijské univerzity [4].
Jako nejschůdnější řešení pro video paměť se jeví použití block RAM, jelikož jen k té máme přístup po celou dobu práce mikrořadiče. Pokud bychom chtěli využít externí SRAM paměť, muselo by se vyřešit přepínání mezi video a programovou pamětí. Pokud bychom tedy kvůli vykreslování video paměti měli zastavovat procesor, značně by se tím snížila jeho výpočetní schopnost.

3.4 Implementace

3.4.1 LEDS - výstup na kontrolky

Kontrolky jsou připojeny k datové sběrnici přes registr. Pokud dojde na adrese 0x00 k IO zápisu, tak se zapíše hodnota z datové sběrnice do tohoto registru.



Obrázek 3.5: LEDS - blokové schéma připojení kontrolek

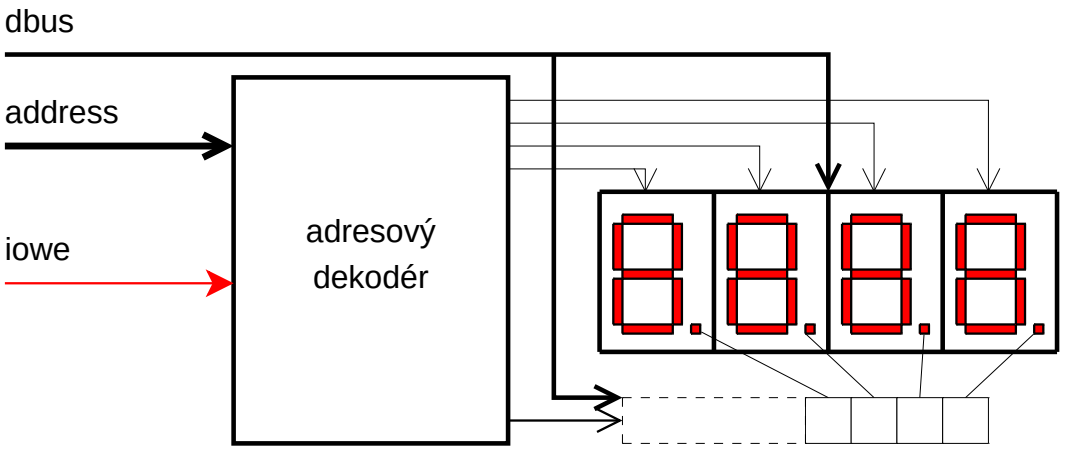
LEDS registr

Bit	7	6	5	4	3	2	1	0
0x00	MSB							LSB
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  Je-li bit nastaven, tak daná LED svítí, jinak je zhasnutá.

3.4.2 Display - výstup na 7-segmentovky

Tento modul obsahuje 2 registry (DILR, DIHR), kde každý z nich uchovává BCD informaci na dvojici 7-segmentovek. Tato informace je předávána dál do BCD dekodéru. Následuje již informace o tom, který segment má svítit. Dále je zde stavový automat, který postupně vybírá daný řád na display a přiřazuje mu hodnoty z výstupů BCD dekodéru.



Obrázek 3.6: Display - blokové schéma připojení display k AVR

DILR - DIsplay Low Register

Bit	7	6	5	4	3	2	1	0
0x04	MSB							LSB
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  8bitová BCD hodnota zobrazovaného čísla na 0. a 1. místě displaye

DIHR - DIsplay High Register

Bit	7	6	5	4	3	2	1	0
0x05	MSB							LSB
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  8bitová BCD hodnota zobrazovaného čísla na 2. a 3. místě displaye

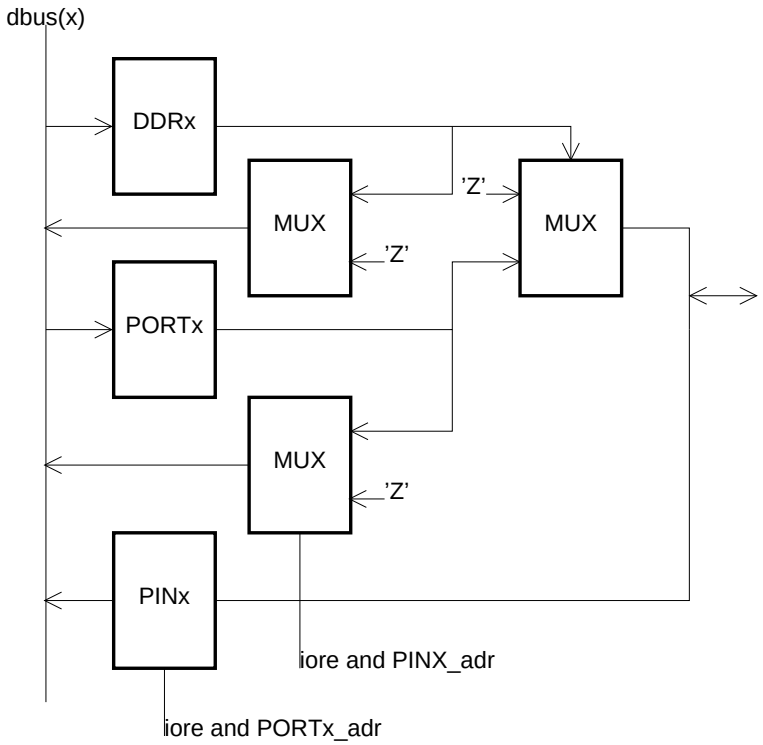
DIPR - DIsplay Point Register

Bit	7	6	5	4	3	2	1	0
0x03	-	-	-	-	DIP3	DIP2	DIP1	DIP0
čtení/zápis					w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- DIP0 - Display Point 0
  Je-li bit vynulován, tak svítí desetinná tečka na 0. místě
- DIP1 - Display Point 1
  Je-li bit vynulován, tak svítí desetinná tečka na 1. místě
- DIP2 - Display Point 2
  Je-li bit vynulován, tak svítí desetinná tečka na 2. místě
- DIP3 - Display Point 3
  Je-li bit vynulován, tak svítí desetinná tečka na 3. místě

3.4.3 PORT - vstupně-výstupní port

Vstupně-výstupní port je realizován třemi registry, kde první určuje směr toku dat (DDR). Druhý určuje hodnotu na pinu (PORT). Třetí registr obsahuje aktuálně načtenou hodnotu daného pinu.



Obrázek 3.7: Vstupně výstupní x-tý port s řízením směru přenosu. Registr DDR řídí, zda-li daný pin bude vstupní, či výstupní. PORTx je registr určující výstupní hodnotu na pinu. PINx registr uchovává v každém bitu hodnotu úrovně na vstupním pinu.

DDRx - Data Direction Register

Bit	7	6	5	4	3	2	1	0
-	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  Je-li bit nastaven, chová se daný pin jako výstupní, pokud je v log. 0, tak se jedná o vstupní pin.

PORTx - Data Register

Bit	7	6	5	4	3	2	1	0
-	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  Určuje logickou úroveň na daném pinu, je-li zvolen jako výstupní.

PINx - Input Pins Address

Bit	7	6	5	4	3	2	1	0
-	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0
čtení/zápis	r	r	r	r	r	r	r	r
inic. hodnota	0	0	0	0	0	0	0	0

- 7. až 0. - bit
  Aktuálně načtená hodnota pinu.

3.4.4 UART - Univerzální asynchronní sériový kanál

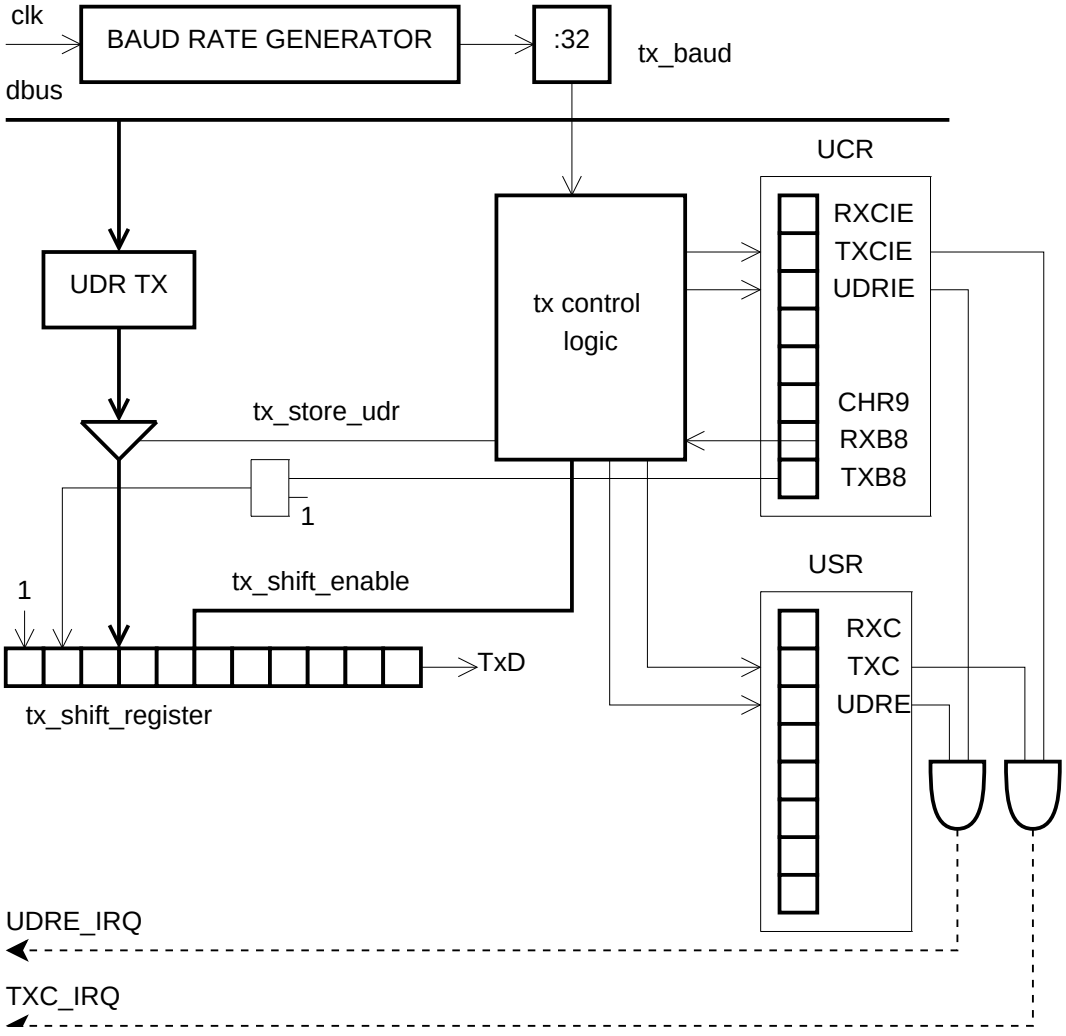
Navrhovaný design sériového kanálu obsahuje dvě nezávislé jednotky (přijímač a vysílač) a je tedy schopen full duplex provozu. Vyznačuje se těmito vlastnostmi:

- Možnost měnit přenosovou rychlost ve velkém rozsahu
- 8 nebo 9 bitový přenos
- Vstupní filtr
- Detekce přepisu vstupního bufferu
- Generuje tři druhy přerušní (vysílání dokončeno, prázdný vysílací registr, příjem dokončen)

3.4.4.1 Vysílač

Blok vysílače 3.8 se skládá z několika částí a to: generátoru přenosové frekvence (Baud Rate Generator), posuvného registru (tx shift registr), registru UDR obsahujícího posílaná data, řídícího (UCR) a stavového (USR) registru a jednotky řízení (konečný automat).
Vysílač je v klidu (nevysílá) do té doby, dokud nejsou uložena nová data v UDR. To se pozná podle bitu UDRE (prázdný datový registr UDR) v registru USR. V okamžiku zápisu do UDR se vynuluje bit UDRE a načtou se data z UDR do 11ti bitového posuvného registru (pomocí

signálu `tx_store_udr`). Poté se začnou z posuvného registru vysouvat (LSB) sériová data ven. Pokud je nastaven bit CHR9 (UCR) vysílá se i 9. bit TXB8 (UCR). Při vysílání stop bitu (log.1) se rozhodneme podle UDRE, zda-li jsou již připravena data pro další vysílání, či zda se má čekat. V poslední fázi se generuje signál TXC, že bylo dokončeno vysílání. Přenosová frekvence se odvíjí od nastavení registru UBRR.
Bit TXEN (pin pro UART/normální pin) byl oproti modelu Atmega103 vynechán, protože u tohoto designu nesdílíme vývody s jinými periferiemi.

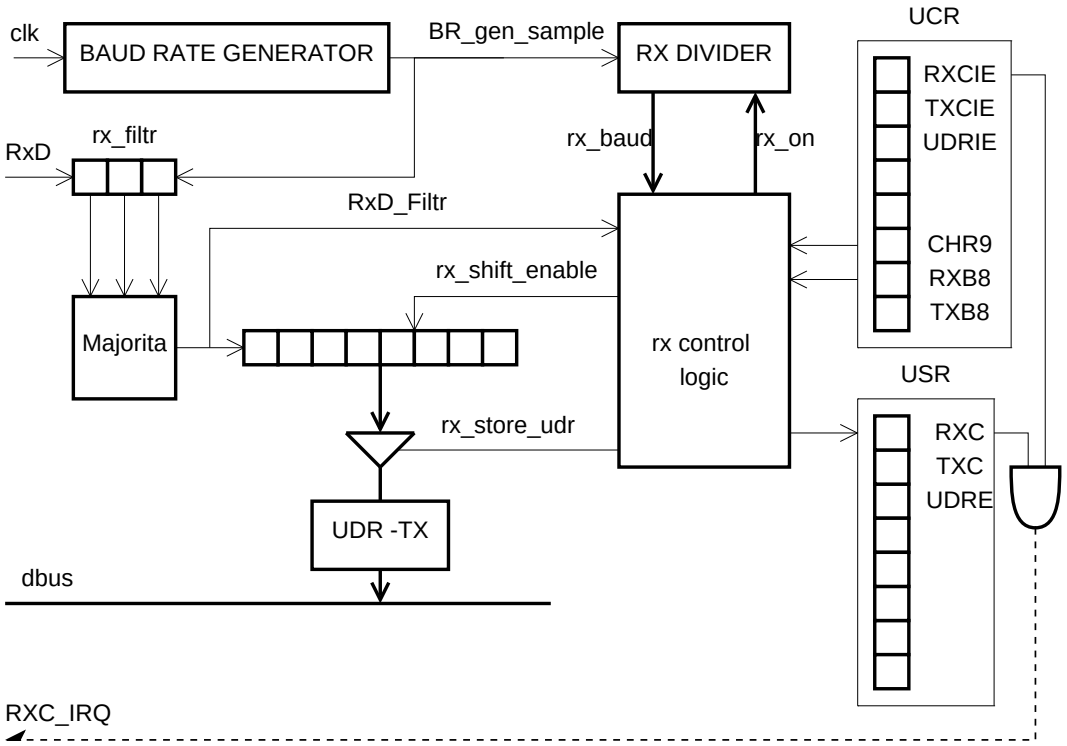


Obrázek 3.8: UART - blokové schéma vysílače

3.4.4.2 Přijímač

Přijímací část 3.9 sériového portu tvoří jako u vysílače blok zvaný Baud Rate Generator, který bude popsán dále. Výstupem tohoto bloku je frekvence 32krát vyšší, než je přenosová frekvence sériového portu. Touto frekvencí je navzorkován signál ze vstupního pinu a je přiveden do jednoduchého filtru, tvořeným 3 bitovým posuvným registrem. Na každý bit tohoto registru je připojena majoritní funkce ze 3 a výstup (`RxD_Filtr`) je teprve potom přiveden do 8 bitového posuvného registru (`rx_shift_reg`).
Celý blok ovládá řídící automat. Nejprve čeká na to, až poklesne úroveň vstupního signálu na log. nulu. V tento okamžik se spustí čítač (`rx_count`) a začne odpočítávat polovinu periody

přenosové rychlosti. Po načítání této hodnoty začíná snímat vzorky signálu (`RxD_Filtr`) a postupně ukládat do posuvného registru (na místo MSB). Pokud je nastaven CHR9 bit, přijímá automat 9 bitů, zatímco standartně 8 bitů. Poté se čeká na stop bit (log. 1). Při příjmu posledního bitu se automat navrátí do počátečního stavu. Při příjmu stop bitu dojde k uložení přijmutých dat do registru UDR pomocí signálu `rx_store_udr` a v tento okamžik se nastaví bit RXC (registr USR). Pokud byl před tímto úkonem již bit nastaven, tak se též nastaví DOR bit (USR).



Obrázek 3.9: UART - blokové schéma přijímače

UDR

Bit	7	6	5	4	3	2	1	0
0x0C	MSB							LSB
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	0	0	0	0	0	0	0	0

UDR registr jsou fyzicky dva registry na jedné IO adrese. Jeden pro čtení (přijímací část) a druhý pro zápis (vysílací část). Při zápisu do UDR začne UART vysílat a při přečtení registru se odstartuje příjem dalšího znaku.

USR - UART Status Register

Bit	7	6	5	4	3	2	1	0
0x0B	RXC	TXC	UDRE	-	DOR	-	-	-
čtení/zápis	r	rw	r	-	r	-	-	-
inic. hodnota	0	0	1	0	0	0	0	0

- 7. bit - RXC (UART Receive Complete)
  Tento bit je nastaven poté, co byl přijmutý bajt přesunut z posuvného registru do datového (UDR). Pokud je navíc nastaven i RXCIE (UCR), tak se vyšle signál přerušení k jádru AVR. Bit RXC je vynulován v momentě čtení UDR.

- 6. bit - TXC (UART Transmit Complete)
  Bit TXC se nastaví na log. 1 když vysílaný Bajt byl právě vyslán a UDR registr je prázdný (tj. UDRE = log. 1). Pokud je nastaven i TXCIE (UCR) tak se hodnota TXC přenáší jako přerušení k jádru procesoru. TXC je nulován buď odpovědí od obsluhy přerušení, nebo zápisem log. 1 na místo bitu TXC.

- 5. bit - UDRE (UART Data Register Empty) UDRE bit je v log. 1, pokud byl Bajt zapsán z UDR do posuvného registru. To indikuje, že je vysílač připraven pro uložení dalšího Bajtu k vysílání. Jestliže je UDRIE (UCR) v log.1, tak je generováno přerušení tak dlouho, dokud je nastaveno UDRE. UDRE je vynulováno, když dojde k zápisu do UDR.

  UDRE je po restartu nastaveno, což indikuje, že je vysílač připraven k vysílání.

- 3. bit - DOR (UART Data Register Empty) Pro detekci přepsání ještě nevyzvednutých dat z UDR slouží tento bit. Je nastaven, pokud ještě nedošlo k přečtení UDR, ale již byl přijat další bajt. Bit je nulován při příjmu dat a zápisu do UDR.

UCR - UART Control Register

Bit	7	6	5	4	3	2	1	0
0x0A	RXCIE	TXCIE	UDRIE	-	-	CHR9	RXB8	TXB8
čtení/zápis	r	rw	r	-	-	w	r	w
inic. hodnota	0	0	0	0	0	0	0	0

- 7. bit - RXCIE (RX Complete Interrupt Enable) Slouží k povolení generování přerušení při dokončení příjmu Bajtu.

- 6. bit - TXCIE (TX Complete Interrupt Enable) Povolení přerušení, které je vyvoláno při dokončení vysílání Bajtu.

- 5. bit - UDRIE (UART Data Register Empty Interrupt Enable) Povolení přerušení pokud je vysílací registr prázdný. Slouží k indikaci, že je možno naplnit UDR dalšími daty.

- 2.bit - CHR9 (9-bit Characters) Pokud je tento bit nastaven, umožňuje přijímač i vysílač 9bitový datový přenos. Pro vysílání je 9. bit reprezentován bitem TXB8 a přijímaný bit se ukládá do RXB8. 9. bit může sloužit jako extra stop bit, parita, či k rozlišení adresy a dat u multiprocesorové komunikace.

- 1.bit - RXB8 (Receive Data Bit 8) Zde je uložen 9. bit přijatého znaku.

- 0.bit - TXB8 (Transmit Data Bit 8) 9. bit vysílaného znaku.

3.4.4.3 Baud Rate Generator

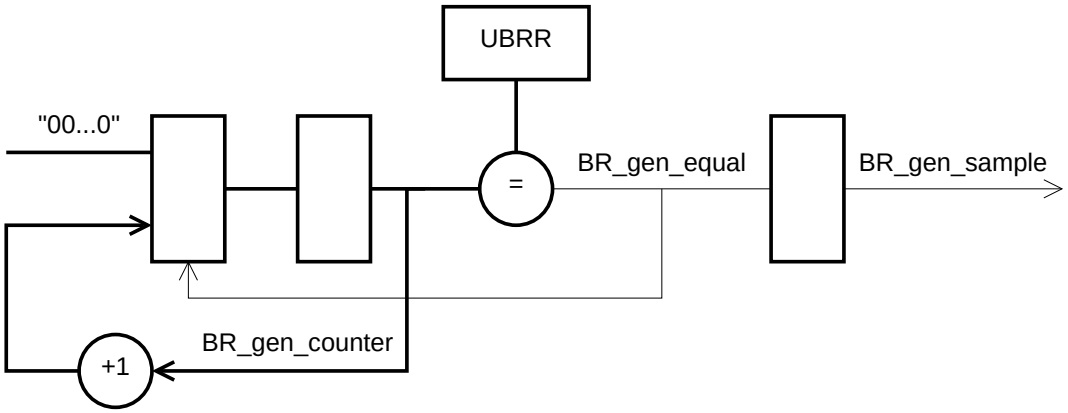
Generátor přenosové rychlosti je tvořen děličkou frekvence, která generuje přenosovou rychlost dle následující rovnice:

$$BAUD = \frac{f_{CLK}}{32(UBRR+1)} \qquad UBRR = \frac{f_{CLK}}{32 \cdot BAUD} - 1$$

Generátor 3.10 je tvořen čítačem, který při každém taktu přičte jedničku. Aktuální hodnota čítače je porovnána s hodnotou v UBRR. Pokud jsou hodnoty shodné (signál `BR_gen_equal`) je čítač vynulován.

$f_{CLK} = 50MHz$		
Přenosová rychlost	UBRR	Error [%]
9600	162	0,15
14400	108	0,45
28800	53	0,47
38400	40	0,76
57600	26	0,47
76800	19	1,73

- BAUD - přenosová rychlost
- f_{CLK} - hodinová frekvence designu
- UBRR - obsah UART Baud Rate Registru (rozsah 0 - 255)



Obrázek 3.10: UART - generátor přenosové rychlosti

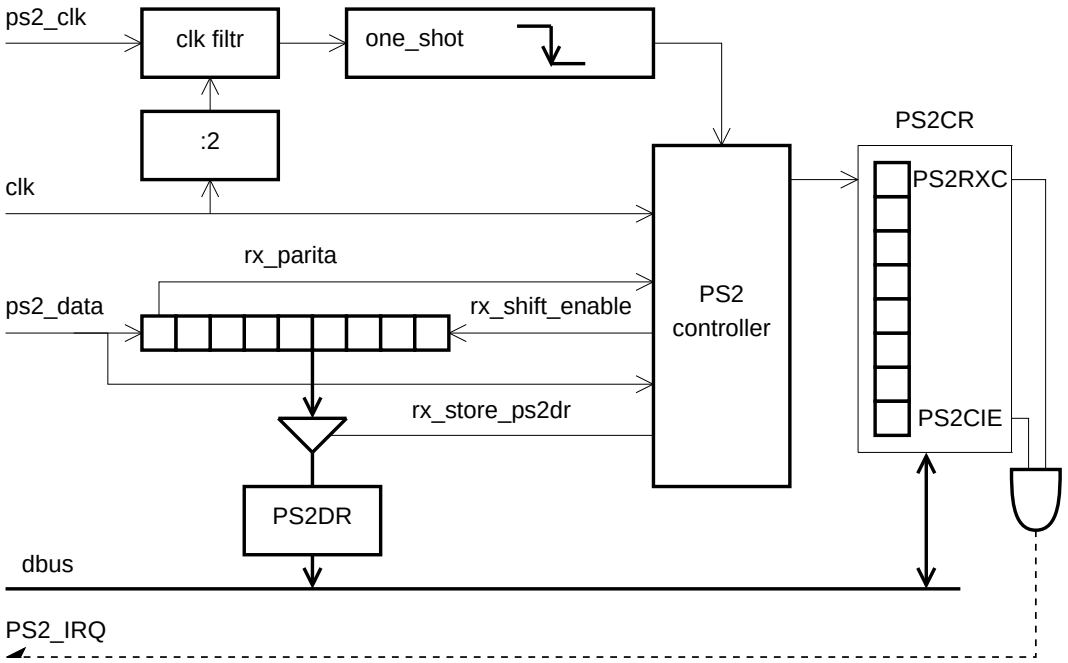
UBRR - UART Baud Rate Register

Bit	7	6	5	4	3	2	1	0
0x09	MSB							LSB
čtení/zápis	rw	rw	rw	rw	rw	rw	rw	rw
inic. hodnota	1	0	1	0	0	0	1	0

UBBR je 8bitový registr, který udává rychlost UART. Viz. Baud Rate Generator

3.4.5 PS/2 - rozhraní klávesnice

Jelikož u PS/2 sběrnice není ošetřeno impedanční přizpůsobení vedení, byl použit na hodinovém vstupu filtr. Tento filtr [9] je složen z posuvného registru a logické funkce AND. Posuvný registr načítá hodnoty hodinového signálu s frekvencí 25MHz a z osmi po sobě jdoucích vzorků vytváří za pomoci funkce and výsledný filtrovaný signál hodin pro budoucí synchronizaci.
V navrženém obvodu 3.11 je za filtrem hodinového signálu umístěn blok (`one shot`), generující impuls při spádové hraně hodin. Příjem začíná při detekci spádové hrany hodin a zároveň musí



Obrázek 3.11: PS/2 - blokové schéma

být datový vodič v log. 1. Poté při každé spádové hraně hodin je načten jeden bit, který je načten do posuvného registru. Poté až se načte i paritní bit, tak se vypočte lichá parita. Pokud je počet načtených jedniček do této doby lichý, je vše v pořádku. Za tohoto předpokladu a další spádové hrany hodin a datového signálu `ps_data` v log. 1 se provede zápis hodnot (signál `rx_store_ps2dr`) z posuvného registru do datového registru PS2DR. Pro indikaci ukočení příjmu je nastaven také bit PS2RXC (v registru PS2CR). Generování přerušení při této události je provedeno, je-li bit PS2CIE v log. 1.

PS2DR - PS/2 Data Register

Bit	7	6	5	4	3	2	1	0
0x01	MSB							LSB
čtení/zápis	r	r	r	r	r	r	r	r
inic. hodnota	0	0	0	0	0	0	0	0

Při ukončení příjmu (viz. registr PS2CR) jsou uložena data do tohoto datového registru.

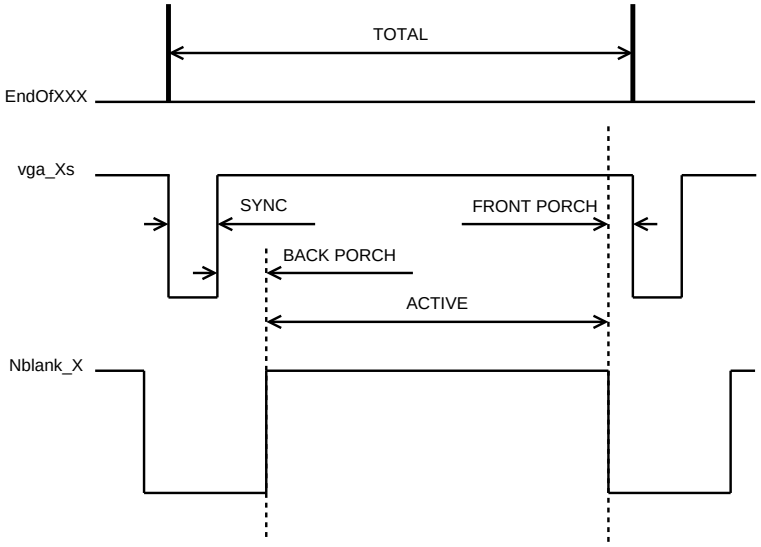
PS2CR - PS/2 Control Registr

Bit	7	6	5	4	3	2	1	0
0x02	PS2RXC	-	-	-	-	-	-	PS2CIE
čtení/zápis	r							w
inic. hodnota	0							0

- 7. bit - PS2RXC (PS/2 Receive Complete)
  Bit je nastaven při dokončení příjmu (přesun dat do datového registru PS2DR).
- 0. bit - PS2CIE (PS/2 Receive Complete Interupt Enable)
  Pokud je bit nastaven na log. 1, tak je po přijetí slova generováno přerušení.

3.4.6 VGA - grafické rozhraní

Design VGA časování byl podle vzoru [4] vytvořen jako dva čítače a to horizontální a vertikální.



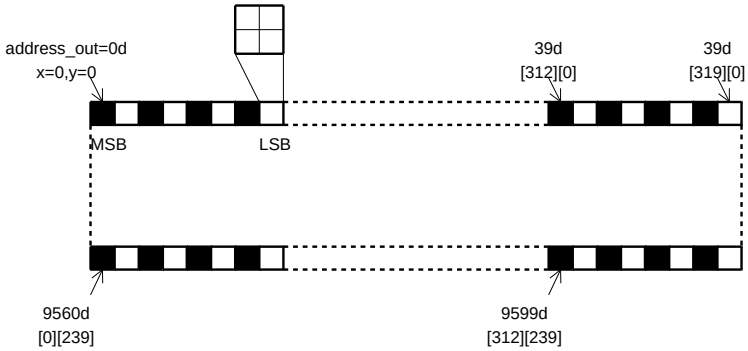
Obrázek 3.12: VGA - řídící signály

Horizontální čítač čítá s frekvencí 25MHz. To je frekvence vykreslování 640 pixelů na jeden řádek. To je dáno délkou synchronizačního pulsu (32 μs) a tedy, čítač musí načítat 800 impulsů na jeden řádek. Tj. $\frac{1}{32 \cdot 10^{-6} \cdot 800} = 25MHz$. Každé načítání 800 impulsů se vygeneruje signál `EndOfLine` a čítač se vynuluje.
Vertikální čítač se inkremetuje při každé nové řádce (impuls EndOfLine), tedy každých 32 μs. Pokud tento čítač načítá hodnoty 521, tak je generován signál `EndOfField`.
Horizontální synchronizační signál je v hodnotě logické nuly v době, kdy začíná návrat paprsku (signál `EndOfLine`) a končí po načítání 96 pulsů (při frekvenci 25MHz).
Obdobně je tomu i při generovní vertikálního synchronizačního signálu. Tj. při návratu paprsku zdola nahoru (zpětná fáze), kterou nám oznamuje signál `EndOfField`, se `vga_vs` vynuluje a za 2 takty vertikálního čítače (64μs) se opět nastaví do logické jedničky.
Pro generování signálu (`Nblank`), který nám oznamuje, že se paprsek nachází ve viditelné části stínítka (640x480pixelů), jsou v této části ještě dva procesy pro generování signálů `Nblank_h` (horizontální doba zobrazení) a `Nblank_v` (vertikální doba zobrazení). Nastavení těchto signálů nastává v době načítání čítačů hodnoty součtu délky synchronizačního pulsu a doby zpětného návratu paprsku. K vynulování dojde po načítání hodnoty součtu délky pulsu, návratu paprsku a délky aktivní doby. Pokud oba signály jsou v log. 1, tak se provádí kreslení na viditelnou část obrazovky. Aktuální pozici elektronového děla můžeme vypočítat z hodnot horizontálního a vertikálního čítače a to podle následujícího vztahu:

$$x = H_{COUNT} - H_{SYNC} - H_{BACKPORCH}$$
$$y = V_{COUNT} - V_{SYNC} - V_{BACKPORCH} + 1$$

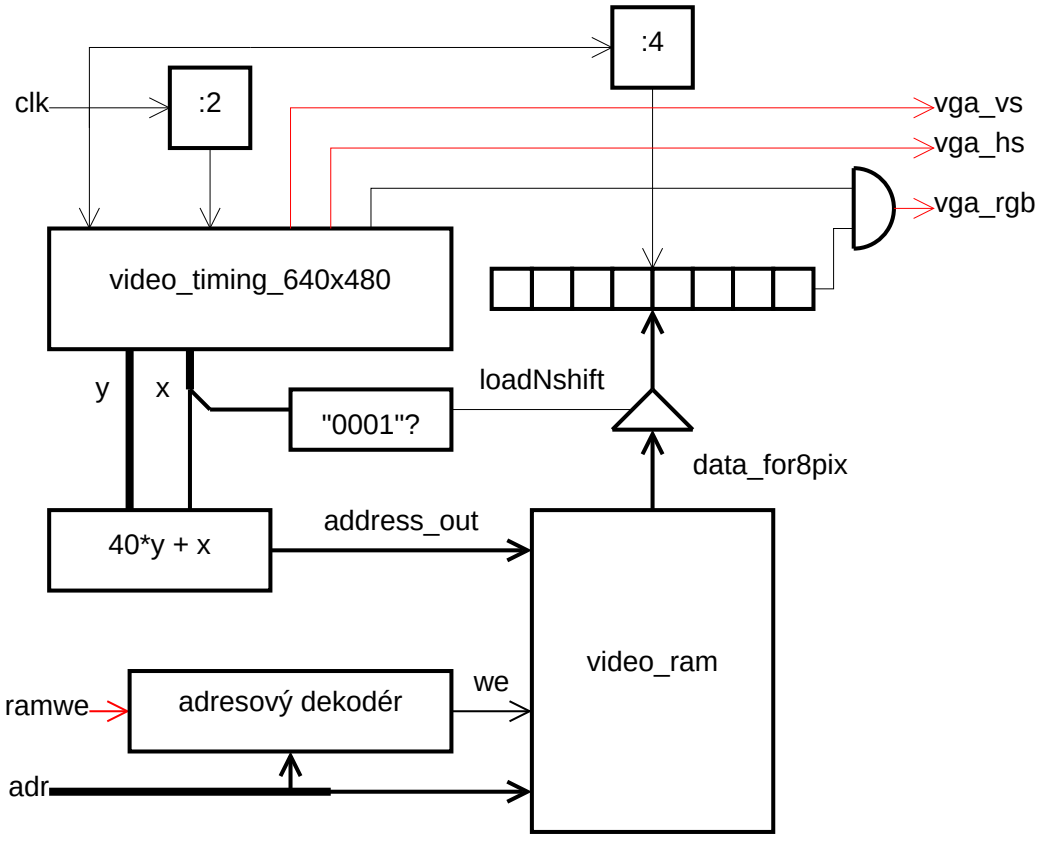
Vývojová deska neumožnila použít externí SRAM paměť pro video RAM z důvodu špatného přístupu k této paměti v době, kdy procesor vykonává instrukce. Vývojová deska má totiž dvě SRAM paměti, které však mají sdílené datové vodiče. Při použití jedné paměti pro uložení instrukcí je tedy nevýhodné druhou paměť používat jako video RAM. Z tohoto důvodu byla vytvořena obrazová paměť v FPGA obvodu pomocí block RAM. To byl limitující faktor pro výběr rozlišení a barevné hloubky.

Periferie byla tedy vyvinuta s rozlišení 320x240 v monochromatickém provedení z důvodu nejmenší paměťové náročnosti (viz tabulka 3.2) na video paměť. VGA periferie se skládá z generátoru signálu VGA, který potřebuje pro svoji činnost frekvenci 25MHz z důvodu vysvětleného dříve v odstavci o časování 3.2.7.1.



Obrázek 3.13: VGA - způsob adresování video paměti

Dále tato periferie obsahuje video paměť. V té je přímo uložen výsledný obrázek, který je poté přímo zobrazován na monitor. Šířka paměti je 8 bitů což znamená 8 pixelů vykreslených vedle sebe na jednom řádku viz. obrázek 3.13. Byla zvolena dvojbranová paměť. Zápis od procesoru je na jedné bráně a čtení nutné k vykreslování na druhé. Tak bylo docíleno neomezeného provozu procesoru i této periferie bez jakéhokoliv zastavování.



Obrázek 3.14: VGA - blokové schéma

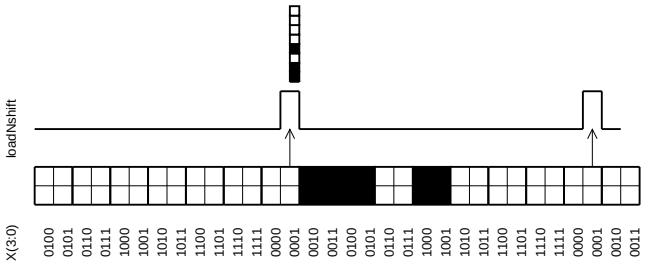
Vykreslování se provádí podle následujícího postupu:

1. Blok `video_timing_640x480` dodává souřadnice x+1 a y (v rozlišení 640x480). Pro další zpracování x,y souřadnic se vydělí hodnota dvěmi. Tj. nejnižší bit (LSB) zahodíme a dostaneme tedy souřadnice v rozlišení 320x240. V dalším bloku se pomocí vzorce $address = (320/8) \cdot y + x = 40 \cdot y + x$ vypočte adresa do obrazové paměti.

2. V dalším taktu se získají data z paměti. Signál pro načítání do posuvného registru se generuje za předpokladu, že hodnota x-ové souřadnice (rozlišení 640x480) po celočíselném dělení 16 je rovna jedné. V této rovnici je ukryta následující myšlenka.

   Data pro vykreslování potřebujeme přesně jeden pixel (takt) před tím, než je vykreslujeme viz. 3.15. Protože každý pixel z video paměti je na obrazovce zobrazován jako 4 pixely (2x2), tak je výhodné pro přednačítání 8 pixelů využít původní hodnotu x-ové souřadnice v rozsahu 640 pixelů na řádek. Pokud tuto hodnotu vydělíme 2, dostáváme 320 bodů na řádce. Po vydělení osmi (šířka paměti) dostanem 40 Bajtu na řádek. Tato hodnota je důležitá pro adresování paměti.

3. Poslední fázi vykreslování tvoří posuvný registr, do kterého je načítán obraz následujících 8 pixelů (rozlišení 320x240). Tento registr vysouvá z nejvyššího bitu (MSB) hodnotu barvy aktuálního pixelu. Posuv nastává s frekvencí signálu `pixel_clk`, tj. pro rozlišení 320x240, to je poloviční frekvence než je u rozlišení 640x480, tedy 12,5MHz.



Obrázek 3.15: Generování signálu loadNshift

Řízení Video paměť je přímo namapována do vnějšího adresového prostoru v rozsahu od 0x4000 do 0x6580. Na obrázku 3.13 je v relativních adresách náhled, jak jsou uloženy jednotlivé pixely.
Přístup k daným 8 pixelům (x-ová souřadnice je zarovnána v paměti po 8 bitech) je možný zápisem následujícího bajtu na adresu $y \cdot \left(\frac{320}{8}\right) + \frac{x}{8} + 4000_{HEX}$:

Bit	7	6	5	4	3	2	1	0
viz. výše	MSB							LSB
čtení/zápis	w	w	w	w	w	w	w	w
inic. hodnota	0	0	0	0	0	0	0	0

4 Demo aplikace

4.1 Textový terminál

`text_terminal`

4.1.1 Úvod

Tato aplikace je určena k připojení k unixovému stroji, či jinému zařízení. Je schopná vypisovat na obrazovku ASCII znaky. Podporuje ANSI/VT100 příkazy. Počet znaků na řádek je 40 a na obrazovku je možno zobrazit až 30 řádků. Terminál nedovede posouvat text (scroolovat).

4.1.2 Implementace

Vlastní program pro vypisování textu obsahuje generátor znaků. Je zde použito fontu 8x8. Tato velikost fontu byla vybrána s ohledem na způsob adresování grafické paměti, kde na jednu adresu připadá 8bitů. Tedy na jeden řádek se naskládá vedle sebe 40 znaků ($\frac{320}{8}$). Každý znak se skládá z 8 mikrořádek. V souboru font.c je uložen font 8x8 v poli. V tomto poli je indexem ASCII hodnota znaku + mikrořádek. Tato matice je uložena v paměti programu (nutné použít funkci `pgm_read_byte()`).
Na obrazovku se vypisují znaky, které přijal mikroprocesor po sériové lince. V rutině přerušení (příjem v UART dokončen) je konečný automat. Ten přijímá postupně znaky a reprezentuje jejich význam.
Existují řídící znaky v rozsahu 0 až 32, které říkají, co má terminál udělat. Tyto znaky se nevykreslují. Speciální znak ESC (0x1B) uvozuje escape příkazy pro posun kurzoru, výmaz obrazovky atd.
Tento terminál podporuje tyto příkazy:

ESC 7 - ulož aktuální souřadnice

ESC 8 - obnov polohu aktuálních souřadnic

ESC [A - posun kurzoru o jednu pozici nahoru

ESC [B - posun kurzoru o jeden řádek dolu

ESC [C - posuň kurzor o jeden sloupec doprava

ESC [D - posuň kurzor zpět o jednu pozici

ESC [x A - posun kurzoru o x pozic nahoru

ESC [x B - posun kurzoru o x řádek dolů

ESC [x C - posuň kurzor o x sloupců doprava

ESC [x D - posuň kurzor o x pozic zpět

ESC [H - posuň kurzor na počátek (vlevo nahoře)

ESC [J - vymaž obrazovku od aktuální pozice dolů

ESC [1 - vymaž obrazovku od aktuální pozice nahoru

ESC [J 2 - vymaž obrazovku a nastav kurzor na počátek (vlevo nahoře)

ESC [K - vymaž řádek od aktuální pozice doleva

ESC [K 1 - vymaž řádek od aktuální pozice doprava

ESC [K 2 - vymaž aktuální řádek

ESC [S - ulož aktuální souřadnice

ESC [U - obnov polohu aktuálních souřadnic

ESC [y ; x H - posuň kurzor na pozici x,y

ESC [y ; x f - posuň kurzor na pozici x,y

Následující příkazy dokáže přijmout, ale nikterak je nereprezentuje.

ESC [m

ESC [x m

ESC [x ; y m

Při stisku klávesy na PS/2 klávesnici se vyšle do mikroprocesoru scankód. V přerušovací rutině (příjem od PS/2 ukončen) je druhý konečný automat, který provádí překlad scankódu na řídící sekvenci, či ASCII znak. Dále se vše posílá pomocí sériového rozhraní ven z mikroprocesoru.
Překlad ze scankódu do ASCII se provádí pomocí tabulky. Pokud je stisknuta klávesa Shift, je nastavena proměnná `shift_down` a význam kláves se změní tak, že se čte z druhé části překladové tabulky.
Rozpoznání puštění klávesy se provádí pomocí kódu 0xF0. Při stisku kurzorové šipky nahoru (dolu, doprava, doleva) se vygeneruje escape sekvence ESC [A (B ,C , D). Klávesa Ctrl mění význam některých kláves a generuje na výstupu řídící znaky v rozsahu 0x00 až 0x32.

4.1.3 Závěr

Uvedený jednoduchý program demonstruje, jak tisknout znaky v grafickém režimu bez použití hardwarové podpory. Dále je zde ukázka zpracovávání přerušení (UART, PS2) a nakonec zpracování scankódu přijatého od klávesnice. Vývojová deska byla připojena k pracovní stanici Linux, na které bylo spuštěno přihlašování ze sériové konzole. Program se choval přesně tak, jak měl. Nevýhoda tohoto textového terminálu je, že řádek může obsahovat pouze 40 znaků (dáno malým rozlišením grafického režimu). V neposlední řadě by mělo být zmíněno, že tento terminál neumí všechny escape sekvence a tedy některé příkazy nejsou správně interpretovány.

4.2 Ukázka grafického režimu

`test_vga_320x240`

4.2.1 Úvod

Uvedený program představuje použití grafického režimu pro zobrazování obrázku, uloženého v paměti jako raw soubor. Obrázek je uložen v programové paměti. Každý bit reprezentuje zobrazovaný pixel.

4.2.2 Imlementace

Program pouze načítá obrázek z programové paměti a kopíruje ho do obrazové paměti. Obrázek byl získán programem GIMP [2] a poté byl uložen (formát: zdrojový kód v C). Tento formát reprezentuje každý pixel jako trojici Bajtu (R,G,B) a proto je nutné provést převod do černobílého modelu. To provádí program `convert_to_BW`. Ten tedy vytvoří z Céčkového souboru LogoCVUT.c zdrojový soubor obsahující obrázek uložený v poli.

4.2.3 Závěr

Výsledkem je logo ČVUT (lev s kružítkem), který se postupně posouvá ve vertikálním směru. Jelikož hardware neposkytuje informaci o zpětném běhu paprsku v monitoru, není možné scénu překreslit v době, kdy není paprsek na stínítku. To se tedy projevuje "blikáním" obrazu.

5 Závěr

Úkol, který byl na začátku práce zadán, byl splněn. Výsledkem je zdokumentované a funkční jádro mikrořadiče implentované v FPGA obvodu Spartan 3. Veškeré periferie, co poskytuje přípravek Digilent Starter kit (PS/2, RS232, SRAM, VGA, tlačítka, přepínače, display) je možné bez omezení využívat. Tyto periferie jsou přímo namapovány do adresového prostoru AVR. Externí SRAM tvoří programovou paměť, ve které jsou uloženy instrukce, popř. konstanty. Jelikož Spartan 3 neobsahuje dostatečné množství block RAM, je grafický režim ochuzen o barevné podání. Rozlišení VGA rozhraní je pro použití 320x240 velice omezující. Pro praktické použití by bylo vhodné použít jiný FPGA obvod nebo jiný přípravek, který má separátně oddělené externí paměti, které by šly využít pro video paměť. Všechny periferie lze k návrhu přidávat či odebírat a je tedy například možné vytvořit mikrořadič s několika sérivými rozhraními. Jsme tedy pouze omezeni schopnostmi FPGA obvodu (počtu základních bloků).
Byly přiloženy dvě aplikace pro demostraci funkčnosti. Nepovedlo se vytvořit procesor lepší, než jiné dostupné (PicoBlaze, MicroBlaze), ale přínosem této práce je procesor, který je šířen pod licencí GNU [1] a software pro tento procesor je možné vytvářet za pomoci jazyka C a překladače gcc. Dále je možné zasahovat do vnitřního zapojení mikrořadiče, pokud by bylo zapotřebí. Při simulaci obvodu může též sloužit jako názorná pomůcka k demonstraci práce mikrořadiče přímo uvnitř v jádru.

6 Literatura

[1] GNU General Public License.
URL http://www.gnu.org/licenses/gpl.html

[2] GNU Image Manipulation Program.
URL http://www.gimp.org/

[3] ModelSim.
URL http://www.model.com/

[4] VGA timing.
URL http://www1.cs.columbia.edu/~sedwards/classes/2005/emsys-summer/

[5] Atmel: .
URL http://www.atmel.com

[6] Atmel: ATmega103.
URL www.atmel.com/atmel/acrobat/doc0945.pdf

[7] Atmel: Instruction Set.
URL www.atmel.com/atmel/acrobat/doc0856.pdf

[8] Digilent: Starter kit.
URL http://www.xilinx.com/bvdocs/userguides/ug130.pdf

[9] Furman Hamblen: *Rapid Prototyping of Digital Systems*. 2002.

[10] David Hradecký: VGA řadič na FPGA.
URL amber.feld.cvut.cz/fpga/teaching/fpga/VGA_FPGA.pdf

[11] Ruslana Lepetenoka: AVR core.
URL http://www.opencores.com/people.cgi/info/lepetenokr

[12] Opencores: AVR core.
URL http://www.opencores.com/projects.cgi/web/avr_core/overview

[13] FEL ČVUT: SPARTAN - 3,Xilinx FPGA Device. 2003.
URL http://noel.feld.cvut.cz/vyu/ap2/SPARTAN-3.pdf

7 Uživatelská příručka

7.1 Hardware

Popis AVR jádra pomocí VHDL kódu je v adresáři `vhdl/avr_core_III/avr_core_3.6`. Všechny periferie, které lze připojit k jádru (soubor `top_avr_core_sim.vhd`) lze nalést v adresáři `vhdl/periferie`. Periferie jsou připojeny k datové, řídící a adresové sběrnici. Napojení na systém obsluhy přerušení se provádí přes sběrnici `sg_core_irqlines`, kde každý signál je pojmenován (soubor `AVRuCPackage.vhd`).
Všechny nadefinované adresy periférií a jejich registrů lze nalézt v souboru `AVRuCPackage.vhd`. Mikrořadič a tedy jeho programovou paměť lze naplnit pomocí sériového portu. Je nutno mít připojenu tuto periferii a musí být nastavena proměnná simulation na 2. Programování probíhá následovně. Nejprve musí být aktivní signál `prog_enable` (přepínač SW0 nahoře). Poté vyresetován přípravek (tlačítko BTN3). Dále je nutné nahrát data pomocí programu `uploader` do paměti. Nakonec vrátit signál `prog_enable` (přepínač SW0 dolu) a znovu resetovat. V tuto chvíli procesor zpracovává nahraný kód.
Další možnost je začlenit kód programu přímo do designu a tady je nutné mít přiložen soubor `PROM.vhd` (generuje program `hex2vhdl`) a proměnná simulation musí být nastavena na 0.
Další proměnnou je `FcpuDivider`, který je dělícím poměrem hlavní frekvence. Určuje, s jakou rychlostí bude pracovat jádro procesoru. Tj. $\frac{f_{CLK}}{2^{FcpuDivider+1}}$

7.2 Software

Software pro mikroprocesor může být vytvořen v assembleru nebo jazyku C. Pro oba způsoby programování byl použit GNU [1] překladač gcc.
Při tvorbě programu v jazyku C je možno použít hlavičkového souboru `iomFPGA.h`, který obsahuje adresy a názvy registrů pro tento mikrořadič.
V adresáři `c/` je možné nalézt některé demonstrační programy. Pro překlad byl použit `makefile`. Tj. při potřebě přeložit soubor se v unixovém prostředí zadá příkaz `make`. Pro nahrání do přípravku `make load`.
Program se nahrává defaultně přes rozhraní `/dev/ttyS0`. Pro správné nastavení tohoto zařízení je nutno zadat před prvním programováním následující příkaz:

```
stty -F /dev/ttyS0 clocal cread -crtscts cs8 -cstopb hup -parenb parodd -brkint
-icrnl ignbrk -igncr ignpar imaxbel -inlcr inpck -istrip -iuclc -ixany
ixoff -ixon bs0 cr0 ff0 nl0 -ocrnl -ofdel -ofill -olcuc -onlcr -onlret
onocr -opost tab0 vt0 -crterase crtkill
-ctlecho -echo -echok -echonl -echoprt
-icanon -iexten -isig -noflsh -tostop -xcase time 5 min 1
```

8 Obsah přiloženého CD

Zdrojové testovací programy

```
c
|-- blinker
|-- hex2vhd
|-- iomFPGA.h
|-- iomFPGA.h~
|-- nightrider
|-- nightrider_s_fukci
|-- string_SRAM
|-- string_in_ROM
|-- test_call
|-- test_counter
|-- test_jmp
|-- test_ps2
|-- test_ps2_interrupt
|-- test_pushpop
|-- test_reg_file
|-- test_uart
|-- test_vga_320x240
|-- text_terminal
‘-- uploader
```

Bloková schémata do programu DIA

```
obrazky_DIA/
```

VHDL soubory, popis hardwaru

```
vhdl/
|-- avr_core_I
|-- avr_core_II
|-- avr_core_III
|   |-- avr_core_3.0
|   |-- avr_core_3.1
|   |-- avr_core_3.2
|   |-- avr_core_3.3
|   |-- avr_core_3.4
|   |-- avr_core_3.5
|   ‘-- avr_core_3.6
‘-- periferie
    |-- avr_Timer_Counter_1.0
    |-- avr_display_1.0
    |-- avr_leds_1.0
    |-- avr_portx
    |-- avr_prog_mem_1.0
    |-- avr_ps2
    |-- avr_uart_1.2
    ‘-- avr_vga_1.0
```

Projekty do WebPack ISE

```
WebPack_projects/
|-- av_core_2.3
|-- avr_core
|-- avr_core_1.0
|-- avr_core_1.1
|-- avr_core_1.2
|-- avr_core_1.4
|-- avr_core_1.5
|-- avr_core_1.6
|-- avr_core_1.7
|-- avr_core_2.0
|-- avr_core_2.0.1
|-- avr_core_2.1.2
|-- avr_core_2.2
|-- avr_core_2.2.1
|-- avr_core_2.4
|-- avr_core_2.5
|-- avr_core_2.5.1
|-- avr_core_2.5.2
|-- avr_core_2.6
|-- avr_core_2.6.1
|-- avr_core_2.6.2
|-- avr_core_2.6.3
|-- avr_core_2.6.4
|-- avr_core_3.0
|-- avr_core_3.1
|-- avr_core_3.1_speed.zip
|-- avr_core_3.2
|-- avr_core_3.3
|-- avr_core_3.3.1
|-- avr_core_3.4
|-- avr_core_3.5
|-- avr_core_3.6
|-- avr_ps2_1.0
|-- avr_rs232
|-- avr_uart_1
|-- avr_uart_1.1
|-- avr_vga_1.0
|-- dcm
|-- text_vga_edif
|-- top.xcf
`-- top_avr_core_sim.ucf
```

Poznámka: Poslední funkční verze je vždy ta nejposlednější.

Instalační balíčky překladače gcc pro Slackware

```
pkg/
`-- tgz
    |-- avarice-2.4-1.i386.tgz
```

```
|-- avr-binutils-2.15tinyos-3.i386.tgz
|-- avr-gcc-3.4.3-1.i386.tgz
|-- avr-insight-6.3-1.i386.tgz
`-- avr-libc-1.2.3-1.i386.tgz
```