Sem vložte zadání Vaší práce.

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

Bakalářská práce

Malé procesory AVR pro FPGA obvody

Lukáš Haken

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

14. května 2014

Poděkování

Chtěl bych poděkovat Ing. Pavlu Kubalíkovi, Ph.D. za odborné vedení, cenné rady a vstřícnost při konzultacích a vypracování bakalářské práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či spracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2014 .....

Odkaz na tuto práci

Haken, Lukáš. *Malé procesory AVR pro FPGA obvody.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstrakt

Tato bakalářská práce se zabývá implementací již řešeného malého AVR procesoru, popsaného VHDL kódem, do FPGA obvodu Spartan 3E. Pro interakci s okolím jsou k procesoru přidány vybrané periferie vývojové desky Starter Kit. Práce popisuje a srovnává implementované jádro s hotovými soft procesory a podrobně rozebírá implementaci procesoru i přidaných periferií. Výsledkem implementační částí je plně funkční procesor, jehož chování lze popsat ve vyšším programovacím jazyce C.

Klíčová slova soft procesory, FPGA, Spartan 3E-Starter Kit, AVR procesory, VHDL

Abstract

This bachelor work deals with implementation of already discussed little AVR processor, described by VHDL code, to FPGA circuit Spartan 3E. For interaction with surroundings there are added chosen peripheries of evolutionary board Starter Kit. The work describes and compares implemented core with completed soft procesores and analyses implementation of the processor and added peripheries in detail. The result of implementating part is fully functional processor, whose behaviour can be described in higher programming language C.

Keywords soft processors, FPGA, Spartan 3E-Starter Kit, AVR processors, VHDL

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době je využívání procesorů samozřejmost. Lidé si bez nich už nedokáží představit každodenní život. Procesory nás obklopují všude, už ráno, kdy si v mikrovlnné troubě ohřejeme mléko, používáme procesor a mnohdy o tom ani nevíme. Procesory se ale nenachází pouze v mikrovlnných troubách nebo počítačích, ale i v automobilech, letadlech, pračkách, moderních televizích, chytrých telefonech a někdy dokonce i celé domy, jejich vytápění, roztahování žaluzií a další činnosti, řídí procesor.

Doba pokročila a dnes již se neprogramuje pouze chování procesoru, ale je možno naprogramovat i kompletní procesor. Naprogramovaných procesorů je velké množství. Nepředpokládám, že by implementovaný procesor nějak vybočoval z davu již hotových řešení, ale doufám že alespoň mně, ale snad i Vám, poslouží k pochopení „vnitřností“ procesoru. Procesory jsou velmi složité logické obvody a jejich pochopení vidím jako hlavní přínos této práce. Doufám, že nezůstane jen u tohoto sobeckého přínosu a i vy si z mé práce něco odnesete.

Hlavní cíl bakalářské práce je, do vývojového kitu Spartan-3E Starter Kit firmy Xilinx, implementovat malý AVR procesor, který bude komunikovat s vybranými periferiemi. Dílčím cílem je seznámení s implementovanými řešení, poučení z chyb a využití výhod hotových řešení.

Kapitola 1

Rešerše

Jak se rozmáhají obvody FPGA, tak i roste počet softwarových procesorů pro ně. Firmy vyrábějící FPGA obvody, jako jsou například firmy Xilinx a Altera, většinou dávají k dispozici optimalizovaná jádra pro svoje výrobky. To jsou speciální procesory, které se samostatně jako zákaznický integrovaný obvod nevyrábějí. Další softwarové jádra vznikají povětšinou jako open source projekty hardwarových nadšenců.

1.1 PicoBlaze

PicoBlaze [5], pro rodinu FPGA Spartan 3E taktéž nazývaný anglickou zkratkou KCPSM3((K)constand Coded Programmable State Machine), je velmi malý a jednoduchý RISC procesor s Harwardskou architekturou. Toto softwarové jádro vytvořil vývojář firmy Xilinx Ken Chapman v říjnu roku 2003. Tento procesor můžeme použít jako klasické procesory ke zpracování dat, ale zejména je určen, jak napovídá anglická zkratka, pro komplexní, časově nekritické, stavové automaty.

Vlastnosti KCPSM3:

- Zdrojové kódy a kompletní dokumentace dostupné zcela zdarma ze stránek výrobce.
- Pamět programu zabírá přesně jeden blok block RAM. Jak jsem psal výše, tak velikost jedné block RAM u FPGA Spartan 3E je 18Kb a všechny instrukce mají konstantní velikost instrukčního slova 18b, z toho vyplývá, že toto jádro může mít až 1000 instrukcí. Adresová sběrnice má velikost 10b.
- Programování assemblerem.
- 16 8bitových registrů, které jsou označeny písmenem „s“ a číslem v šestnáctkové soustavě, s0 až sF, ale v assembleru mohou být přejmenovány.

Žádný registr není rezervován pro speciální úlohu. Všechny registry mohou být použity jako zdroj či cíl pro instrukce.

- ALU provádí všechny základní aritmetické, logické a porovnávací operace a nastavuje dva příznaky. Příznak ZERO nastaví, když jsou všechny bity v cílovém registru nulové a příznak CARRY nastaví, pokud při aritmetických operacích dojde k přetečení.
- Interní datová paměť o velikost 64B adresovaná buď přímo nebo nepřímo adresou s registru.
- Maskovatelné přerušení, které můžeme programově povolit.
- Všechny instrukce trvají dva hodinové cykly. Výkonnost procesoru se pohybuje od 43 do 66 MIPS, záleží přímo na konkrétním FPGA kam je procesor nahrán.

1.2 MicroBlaze

MicroBlaze [6] je 32bitový vestavěný soft procesor s redukovanou instrukční sadou(RISC), optimalizovaný pro implementaci v programovatelných logických obvodech firmy Xilinx. Jádro MicroBlaze je podstatně větší než předchozí popsaný procesor tentýž firmy a existuje k němu spousta doplňkových modulů.

Vlastnosti MicroBlaze:

- Není zdarma, pro možnost využití tohoto jádra musí být zakoupen vývojový kit EDK od firmy Xilinx, který v sobě obsahuje všechny potřebné věci pro vývoj aplikací s MicroBlaze.
- Součástí EDK je vývojový software pro programování procesoru vyšším programovacím jazykem C.
- Jelikož MicroBlaze má Harwardskou architekturu, tak má oddělený datový a instrukční adresovatelný prostor. Šířka adresové sběrnice je pro oba prostory stejná 32b, tudíž oba adresní prostory mohou být velké až 4GB.
- Pro ukládání dat, je možnost výběru mezi little nebo big endian.
- Datová paměť umožňuje tři přístupy do paměti. Word má velikost 4B, halfword 2B a Byte.
- MicroBlaze obsahuje také datovou i instrukční přímo mapovanou cache, která má velikost buď čtyři nebo osm slov.
- Každá instrukce má 32bitů a jsou definovány dva typy instrukcí. Typ A má dva zdrojové registry a jeden registr jako cílový operand. Typ B má také jeden cílový registr a dále jeden registr zdrojový a 16b konstantu.

- 32 32bitových uživatelských registrů označovaných r0 až r31. R0 má vždy hodnotu 0. R1 až r13 a r18 až r31 jsou volně k dispozici programátorovy. R14 se používá pro uložení návratové adresy z přerušení. R15 je volně k dispozici uživateli, ale doporučuje se pro uložení návratové adresy z uživatelských vektorů. R16 slouží pro uložení návratové adresy z přestávky (break). Pokud je MicroBlaze nakonfigurován k podpoře Hardwarové výjimky, pak se do registru r17 uloží adresa instrukce, která následuje za instrukcí způsobující hardwarovou výjimku.

- Až osmnáct 32bitových speciálních registrů. Například PC, MSR neboli status registr a další.

- Pokud při překladu zapneme prostorovou optimalizaci, pak má MicroBlaze tří stupňovou pipeline. Pokud je prostorová optimalizace vypnuta, pak i pro větší výkon, má pěti stupňovou pipeline.

1.3 Nios II

Nios II [7] je srovnatelný procesor s MicroBlaze. Nios II je 32bitový soft procesor s harwardskou architekturou a redukovanou instrukční sadou, implementovaný pro FPGA obvody firmy Altera. Vlastnosti NIOS II:

- 32bitová datová a i adresová sběrnice.

- 32 32bitových plně uživatelsky přístupných registrů.

- 32 zdrojů přerušení, plus externí řadič přerušení pro další zdroje přerušení.

- Pohodlné grafické vývojové prostředí Nios II SBT, podobné vývojovému prostředí Eclipse, pro programování softwarového jádra vyššími programovacími jazyky C nebo C++.

- Existují tři verze jádra Nios II.

  - Nios II/f (fast) - Tato konfigurace je optimalizovaná pro maximální výkon. Obsahuje šesti stupňovou pipeline a oddělenou datovou a instrukční cache. Hardwarové násobení a posuv trvající jeden takt.
  - Nios II/s (standart) - Vyvážení výkonu a zabírané plochy. Tato konfigurace má pěti stupňovou pipeline a pouze instrukční cache.
  - Nios II/e (economy) - Tato verze je vytvořená pro největší úsporu vnitřní logiky FPGA. Zabírá výrazně nejméně místa z těchto 3 verzí a jako jediná je k dispozici zdarma.

Zde jsou popsaná pouze 3 soft procesory od dvou největších výrobců FPGA. S rozmachem FPGA vzniká i spoustu jader implementovaných v některém hardware popisujícím programovacím jazyku. Krom firem zabývajících výrobou FPGA, vzniká spousty volně dostupných jader, vytvořených hardwarovými nadšenci, na portálu opencores [8].

Kapitola 2

Analýza

2.1 FPGA

FPGA je anglická zkratka (Field Programmable Gate Array), do češtiny přeložená jako programovatelné hradlové pole. FPGA jsou speciální integrované obvody složené z programovatelných bloků a konfigurovatelné matici spojů, jíž mohou být tyto bloky různě propojeny. Programovatelnost, oprava chyby nebo možnost změny chování, to je hlavní čím se odlišují od zákaznických integrovaných obvodů.

FPGA obvody se v dnešní době velice rozmáhají [9]. Může za to jednak snižující cena a zvyšování výkonu, ale také jejich uplatnění. Tyto programovatelné obvody se s výhodou používají u aplikací kde se nejedná o velkou sérii, k prototypování složitějších návrhů, kdy navrhovaný zákaznický integrovaný obvod můžeme vyzkoušet přímo na hardwaru, ještě před finančně velice náročnou výrobou integrovaného obvodu, kde nás každá chyba vyjde velmi draho.

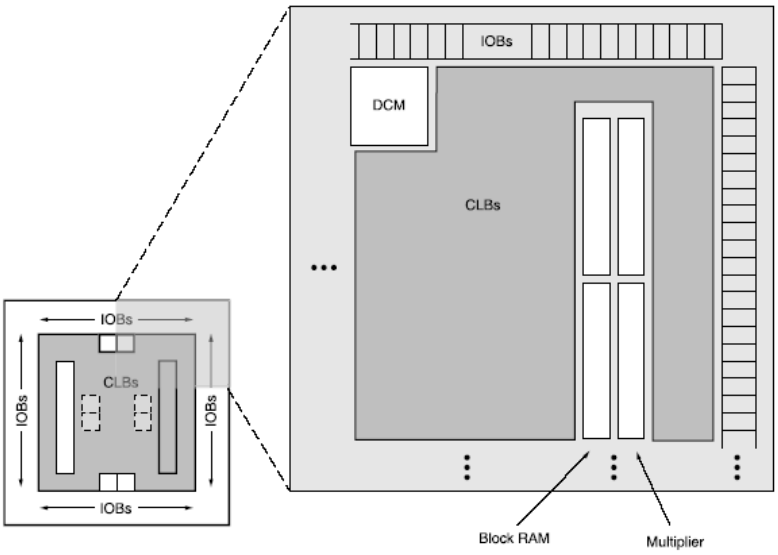
2.1.1 Spartan 3E FPGA Family

Rodina FPGA Spartan 3E staví na předchozí úspěšné řadě Spartan 3 [2], kterou rozšiřuje o větší množství I/O bloků a výrazně snížila cenu za jeden logický blok. Z předchozí řady zůstala 90nm technologie. Díky nízké ceně jsou tyto FPGA obvody vhodné pro širokou škálu zákaznických elektronických obvodů.

Zařízení	hradla	CLBs	Distri. RAM	BlockRAM	DCMs	IOBs
XC3S100E	100K	240	15Kb	72Kb	2	108
XC3S250E	250K	612	38Kb	216Kb	4	172
XC3S500E	500K	1164	73Kb	360Kb	4	232
XC3S1200E	1200K	2168	136Kb	504Kb	8	304
XC3S1600E	1600K	3688	231Kb	648Kb	8	376

Tabulka 2.1: Shrnutí rodiny FPGA Spartan 3E [2]

2.1.1.1 Architektura Spartan 3E



Obrázek 2.1: Znázornění základní architektury FPGA obvodu [2]

Architektura rodiny Spartan 3E se skládá z pěti základních programovatelných funkčních bloků:

Konfigurovatelné logické bloky (CLBs) Základem každého CLB jsou dva LUT4 bloky (Look Up Table). LUT4 je v své podstatě 16bitová paměť, která dokáže realizovat všechny logické funkce o maximálně čtyřech vstupních proměnných. Za každým LUTem je jeden registr, který umožňuje implementovat sekvenční logiku. Dále CLB obsahuje multiplexer, přes který se dají oba LUTy spojit a vytvořit tak logickou funkci o pěti vstupních proměnných.

Vstupně výstupní programovatelné bloky (IOBs) Vstupně výstupní programovatelné bloky jsou po celém obvodu FPGA a slouží ke kontrole

toku dat mezi interní logikou a I/O piny. Každý IOB podporuje obousměrný tok dat.

Blokové RAM(Block RAM) Na rozdíl od distribuované RAM, která je tvořena přímo z CLB, je block RAM samostatná buňka, nezabírající logiku FPGA. V rodině FPGA Spartan 3E má jedna block RAM 18Kb pro uložení dat a dual port, což znamená, že se může číst i zapisovat zároveň.

Násobičky(Multiplier) Vstupem do násobičky jsou 2 18bitová čísla. Násobičky jsou opět jako block RAM dedikované bloky, které nezabírají vlastní logiku FPGA.

Generátor hodin(DCM) Generátor hodin umožňuje distribuovat hodinový signál do celého obvodu a tím zajistí synchronní návrh. Umožňuje digitálně dělit, násobit a různě fázově posouvat hodinový signál.

Složitá logika, především různé řadiče, se v FPGA implementují pomocí stavových automatů. Při stále rostoucí složitosti jsou automaty čím dál hůře implementovatelné a zabírají velkou spoustu logiky v FPGA a tak je někdy výhodné přidat k FPGA obvodu procesor, který nám většinou výrazně usnadní implementaci.

2.1.2 Hard vs. soft procesory

S řešením složitých aplikací nám mohou pomoci procesory. Jsou dva typy procesorů, které se pojí s FPGA obvody:

Hard procesory jsou přímo integrovaná jádra spojená s programovatelným hradlovým polem. Výhoda těchto procesorů je hlavně v rychlosti. Naopak nevýhoda tohoto řešení je v nemožnosti obvod překonfigurovat přímo podle požadavků.

Soft procesory jsou procesory implementované přímo do konfigurovatelných logických bloků, tudíž zabírají místo v logice FPGA, ale s tímto se pojí i výhoda tohoto řešení a tou je možnost procesor nakonfigurovat přímo podle specifik aplikace. Soft jádra jsou povětšinou pomalejší než hard procesory.

2.2 Výběr procesoru

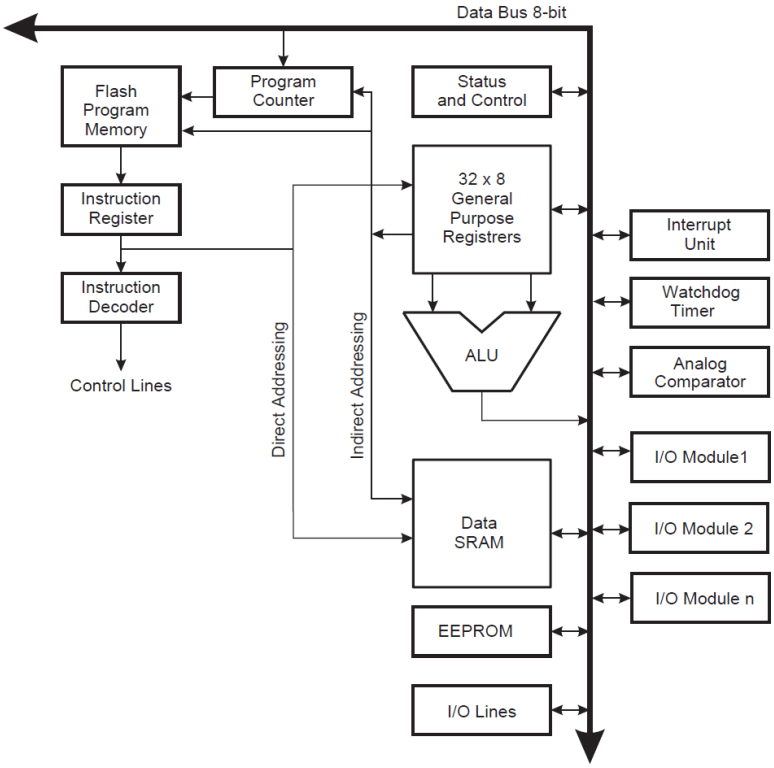
V dnešní době je již vytvořeno spousta soft jader pro FPGA obvody. Přímo pro náš FPGA Spartan 3E jsou optimalizovaná jádra MicroBlaze a PicoBlaze popsaná v rešerši. Tyto procesory se nevyrábí jako samostatné integrované obvody a tudíž se mimo obvody FPGA nevyskytují. To byl jeden z důvodů

proč bylo vybráno AVR jádro. Proti výběru procesoru MicroBlaze byla jeho rozsáhlost a složitost, naopak jádro PicoBlaze nezabírá mnoho logiky FPGA, ale proti němu bylo nemožnost programování ve vyšším programovacím jazyce.

Po prostudování současných řešení jsem zvolil projekt AVRtinyX61core Andrease Hilvarssona [1]. Toto jádro je plně kompatibilní s procesory ATtiny261/461/861 vyrobené firmou Atmel a má tedy plnou podporu velkého množství vývojových nástrojů od firmy Atmel, které jsou poskytovány zdarma.

2.3 AVR ATtiny261/461/861

Nadpis této sekce označuje tři 8bitové mikrokontroléry, vyrobené firmou Atmel[10]. Procesory ATtiny261, ATtiny461 a ATtiny861 [3] jsou RISC architektury a za každý hodinový takt je provedena jedna instrukce. Při maximální hodinové frekvenci jádra 20MHz se blíží výkonu 20 miliónů instrukcí za sekundu. Procesory jsou úplně totožné, liší se pouze ve velikostech pamětí.



Obrázek 2.2: Blokové schéma AVR architektury [3]

Pro maximalizaci výkonu a paralelismu AVR používá Harwardskou architekturu, která odděluje data od instrukcí. Jádro při vykonávání programu využívá rozdělenou instrukční a datovou paměť. Zatímco se provádí aktuální

instrukce, je následující instrukce načtena z paměti programu. Dvoustupňový pipeling umožňuje každý hodinový takt výsledek jedné operace. Dalším typickým rysem AVR architektury je registrové pole, které je složeno z 32 registrů. Přímo na registrové pole je připojená aritmetickologická jednotka. Jádro ATtiny261/461/861 dále obsahuje 64 adres vstupně výstupního prostoru, kde se nacházejí registry pro komunikaci s periferiemi a další speciální funkční registry, jako například status registr.

2.3.1 ALU - Aritmetickologická jednotka

Aritmetickologická jednotka je pro dosažení co nejvyššího výkonu přímo připojená ke všem 32 pracovním registrům. Většina instrukcí má dva zdrojové operandy, což jsou dva registry nebo jeden registr a 1 konstanta. Výsledek je uložen zpět do jednoho z registrů. Jedná se tedy výhradně o operace typu registr - registr. Operace, které ALU provádí se dělí do tří hlavních kategorií, na aritmetické, logické a bitové operace.

2.3.2 SREG - Status registr

Status registr se nachází na poslední adrese v IO prostoru a ukládá informace o výsledku právě provedené instrukce. Tyto informace jsou využity v následně vykonávaných instrukcích. Při skoku do přerušovací rutiny se status registr hardwarově neukládá, ani se neobnovuje při návratu z přerušení, toto se musí zajistit softwarově a většinou se o to postará překladač.

Status Register - SREG

Bit	7	6	5	4	3	2	1	0
0x3F (0x5F)	I	T	H	S	V	N	Z	C
čtení/zápis	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	0	0	0	0	0	0	0	0

- **C** - Carry Flag
  Indikuje přenos při aritmetických nebo logických operacích.

- **Z** - Zero Flag
  Příznak Z značí nulový výsledek aritmetických nebo logických operací.

- **N** - Negative Flag
  N je nastaven pokud je výsledek aritmetických nebo logických operací záporný.

- **V** - Two's Complement Overflow Flag
  Příznak V indikuje přetečení dvojkového doplňku.

- **S** - Sing Bit
  Tento příznak je výsledkem exkluzivní disjunkce (XOR) mezi příznakem N a příznakem V, určuje tedy znaménko výsledku.

- **H** - Half Carry Flag
  Indikuje poloviční přenos, tedy přenos mezi třetím a čtvrtým bitem. Používá se v BCD aritmetice.

- **T** - Bit Copy Storage
  T bit je používán jako zdrojový nebo cílový operand pro instrukce BLD a BST.

- **I** - Global Interrupt Enable
  Povolení přerušení. Pokud je tento bit 0, jsou veškerá přerušení zakázána. Je-li bit v logické 1 a dojde k přerušení, je tento bit automaticky vynulován a nahozen až instrukcí RETI.

2.3.3 Registrové pole

Registrové pole tvoří 32 8bitových registrů. K přístupu k těmto univerzálním registrům stačí jeden hodinový cyklus. Pole registrů je mapováno do spodních 32 adres datového adresního prostoru a tudíž pro práci s jednotlivými registry je možno použít instrukce, které pracují s celým datovým prostorem.

7 … 0	Addr.	
R0	0x00	
R1	0x01	
R2	0x02	
…		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
…		
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

Obrázek 2.3: Universální pracovní registry [3]

Posledních šest, podle obrázku 2.3 registry R26 až R31, jsou po dvojici uspořádány do třech 16bitových registrů, sloužících k nepřímému adresování paměti dat. Programátor též může nastavit, aby se adresy uložené v těchto

registrech predekrementovaly nebo postinkrementovaly. K registrům Y a Z může být přičtena 6 bitová konstanta.

2.3.4 Zásobník

Pro uložení dočasných dat, jako jsou lokální proměnné a návratové adresy z podprogramů nebo přerušení, se používá zásobník. Ukazatel na vrchol zásobníku je uchován ve dvou 8bitových registrech, které se nacházejí v IO prostoru. Jako zásobník je použita vnitřní SRAM. Velikost zásobníku je omezena velikostí SRAM. Na SRAM jsou taktéž uložena uživatelská data a překladač musí hlídat, aby nedošlo k přepsání těchto dat. Ukazatel na vrchol zásobníku musí být na začátku samotného programu nastaven na nejvyšší adresu datové paměti SRAM. Ukazatel na vrchol zásobníku je pokaždé dekrementován, když instrukcí PUSH uložíme data a inkrementován, když data instrukcí POP vyzvedneme. Při uložení nebo vyzvednutí návratové adresy se změní ukazatel o 2.

Stack Pointer Low Register - SPL

Bit	7	6	5	4	3	2	1	0
0x3D (0x5D)	MSB							LSB
čtení/zápis	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA

Stack Pointer High Register - SPH

Bit	7	6	5	4	3	2	1	0
0x3E (0x5E)	MSB							LSB
čtení/zápis	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA	SRAMHA

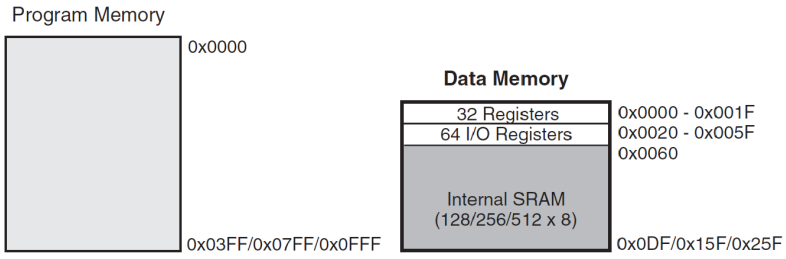
- Inicializační hodnota SRAMHA (SRAM Highest Address) označuje nejvyšší hodnotu datové paměti SRAM.

2.3.5 Přerušení

Nastavením příznaku I ve stavovém registru může dojít k přerušení. Vyvolaný vektor přerušení určuje druh přerušení a adresu v paměti programu, kde je vytvořen odkaz na příslušnou obsluhu přerušení. Nejnižší adresy programové paměti jsou využity právě pro přerušení. Čím nižší adresa, tím vyšší je priorita přerušení. Adresy vektorů přerušení jsou umístěné hned za sebou, tudíž na příslušné adrese najdeme pouze jednu instrukci RJMP, která nás odkáže na příslušné návěští, od kterého začíná obsluha přerušení.

2.3.6 Adresní prostor

AVR architektura obsahuje dva různé adresní prostory. Programový a datový paměťový prostor.



Obrázek 2.4: AVR adresní prostory [3]

Většina instrukcí jsou 16bitové, tudíž je instrukční paměť organizována N × 16. Kde N nabývá hodnoty 1K pro ATtiny261, 2K pro ATtiny461 a 4K pro ATtiny861, v závislosti na velikosti paměti se také mění velikost programového čítače (PC) na 10,11 nebo 12bitů.

Datový adresní prostor je složen ze tří hlavních částí. Prvních 32 adres zabírají univerzální registry, od hexadecimální adresy 20 najdeme 64 adres pro speciální funkční registry a datová paměť SRAM začíná na adrese 60 hexadecimálně. Velikost SRAM paměti je opět závislá na konkrétním typu, procesor ATtiny261 má velikost SRAM 128B, ATtiny461 256B a 512B ATtiny861.

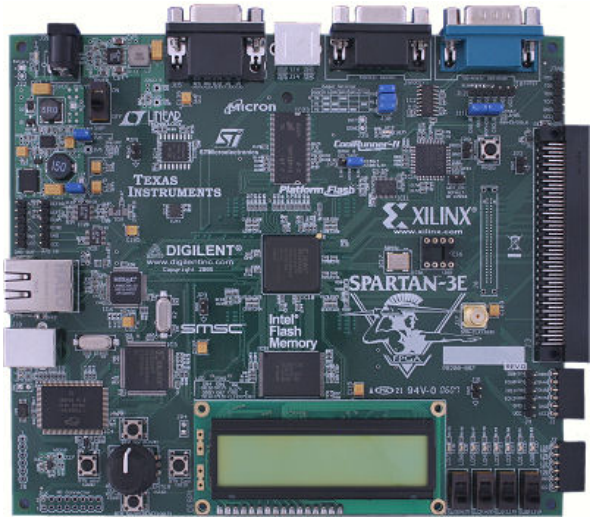
2.4 AVRtinyx61core, Andreas Hilvarsson [1]

Zvolené jádro AVRtinyx61core, popsané na stránkách opencores, [8] je plně kompatibilní s výše popsanými jádry ATtiny261/461/861, až na následujících několik rozdílů. Jádro Andrease Helvarssona neobsahuje pipeling. I díky tomu není k dispozici výsledek instrukce každý takt. Nejkratší instrukce je dostupná za tři hodinové takty a nejdelší instrukce LD je provedena za sedm hodinových taktů. Rozdělení instrukcí do více taktů umožňuje výrazně zvýšit frekvenci procesoru. Integrovaný obvod ATtiny861 má maximální frekvenci 20MHz. Toto jádro při zapnuté optimalizaci na rychlost dosahuje maximální frekvence až 5× vyšší. Jádro nemá implementováno přerušení a z datového adresního prostoru má pouze pole registrů. Toto jádro taktéž neobsahuje jednu společnou datovou a adresní sběrnici, nýbrž má datové a adresové sběrnice rozděleny na vstupní a výstupní a do jednotlivých bloků jsou přivedeny data po samostatných vodičích.

2.5 Spartan-3E Starter Kit

Aby procesor mohl komunikovat s okolím, například aby zobrazoval výsledky nebo reagoval na vnější podněty, musíme k němu přidat periferie.

Deska firmy Xilinx Spartan 3E Starter Kit [4] má všechny potřebné periferie pro komplexní vývoj vestavných aplikací. V této bakalářské práci není využito všech dostupných periferií. Deska obsahuje:



Obrázek 2.5: Spartan-3E Starter Kit [4]

- hradlové pole Xilinx XC3S500E Spartan 3E
  - 232 uživatelských I/O pinů
  - 500 tisíc hradel
  - 20 block RAM
  - 20 integrovaných násobiček
- 64MB DDR SDRAM
- 16MB paralelní NOR Flash(Intel StrataFLash)
- 16Mb SPI sériová flash
- 2řádkový, 16znakový LCD display
- PS/2 port pro myš nebo klávesnici
- VGA displej port

- 10/100 Ethernet
- 2 9pinové RS232 porty
- USB, slouží hlavně pro konfiguraci FPGA a debugování
- 50MHz oscilátor
- hodinový vstup SMA
- 4 výstupy, SPI DA(Digital na Analog) převodník
- 2 vstupy, SPI AD převodník
- rotační dekodér s tlačítkem
- 4 přepínače
- 4 tlačítka
- 8 LED
- 8pinový výstup na oscilátor

Po dohodě s vedoucím práce jsem se rozhodl implementovat následující periferie.

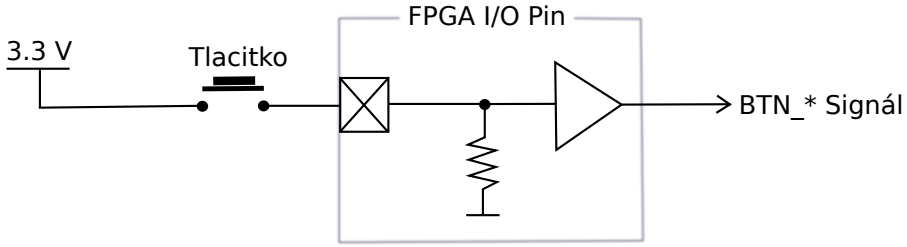
2.6 LED

Spartan-3E Starter Kit má osm samostatných LED, které jsou na desce umístěny vpravo dole nad čtyřmi přepínači a označují se LD0 až LD7. Každá LED má katodu připojenou na zem a anodu k výstupu FPGA, přes rezistor 390Ω. Dioda se tedy rozsvítí při logické 1 na výstupu FPGA.

2.7 Přepínače a tlačítka

Deska obsahuje čtyři přepínače umístěné v pravém dolním rohu pod LED a jsou označeny SW0 až SW3. Přepínače mají dvě polohy. Přepínačem v horní poloze je na vstup FPGA přivedeno napětí 3,3V a tedy logická 1, druhá poloha je tudíž pokud je přepínač dole a pak je vstup FPGA přiveden na zem a tudíž logická 0.

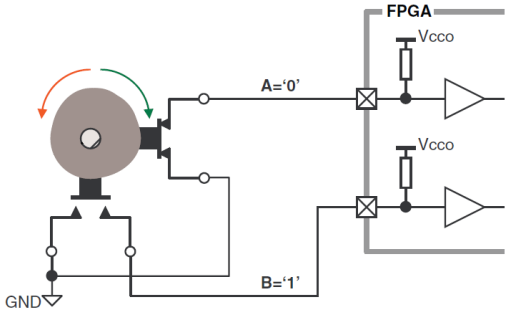
Vývojový kit je také osazen čtyřmi tlačítky, která se nacházejí v levém dolním rohu, okolo otočného dekodéru a označují se BTN_NORTH, BTN_EAST, BTN_SOUTH a BTN_WEST. Pokud stlačíme tlačítko je na I/O pin FPGA přivedeno napětí 3.3V, jinak je I/O pin FPGA přes pull-down rezistor uzemněn.



Obrázek 2.6: Zapojení tlačítka na desce Spartan-3E Starter Kit

2.8 Rotační enkodér

Nalevo od displeje, přímo mezi tlačítky se nachází rotační enkodér, hovorově taktéž občas nazývaný jako „točítko“. Při stisknutí slouží rotační enkodér jako tlačítko.



Obrázek 2.7: Zapojení rotačního enkodéru

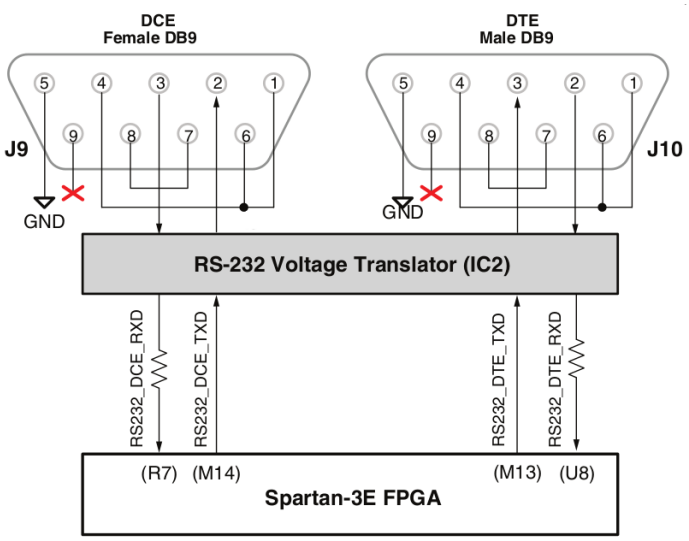
Princip rotační enkodéru je ve dvou tlačítkách, které mají jeden vývod připojený na společnou zem [11]. Výstupy obou tlačítek, tedy signál A a B jsou proti sobě fázově posunuty. Při náběžné hraně jednoho ze signálů a následného porovnání obou signálů zjistíme směr otočení. „točítka“.

2.9 Sériový port RS-232

Nahoře v pravém rohu na desce najdeme dva sériové porty. Konektor DB9 DCE a DTE konektor. Já používám pouze DCE port, který obvykle je, dnes už spíše musím říct, že byl, dostupný na osobních počítačích a pro propojení se používá normální sériový kabel.

Sériová linka je běžnou součástí většiny vestavných procesorů. Je to standartní rozhraní, které se používá i pro meziprocesorovou komunikaci. Umož-

ňuje jednoduchou sériovou komunikaci v 8 nebo 9bitovém asynchronním režimu [12].



Obrázek 2.8: Sériový port RS-232

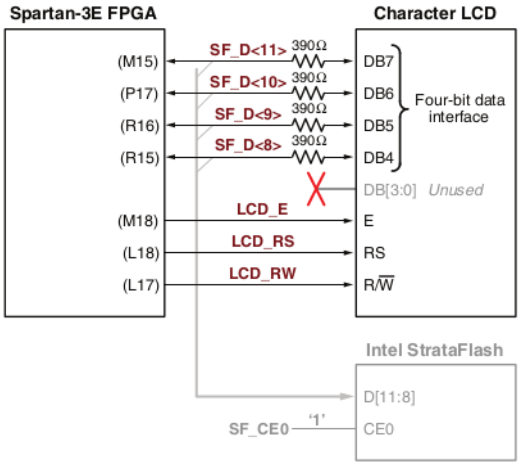
Jelikož RS232 používá jiné napěťové úrovně, než logika FPGA, je na desce také převodník MAX232. FPGA používá výstupní napěťové úrovně LVTTL nebo LVCMOS, obvod MAX232 tyto úrovně při odesílání upraví na napěťovou úroveň RS232 a u příjmu naopak. Při nečinnosti je na sériové lince logická 1, přenos začíná start bitem, což je vysílačem odeslaná logická 0 po dobu 1 bitu. Start bitem se také synchronizují přijímač a vysílač. Data končí stop bitem, nebo-li odvysíláním logické 1.

2.10 Znakový LCD displej

K zobrazení dat slouží dvouřádkový znakový LCD displej, který se nachází uprostřed spodního okraje desky Spartan-3E Starter Kit. Na jeden řádek se vejde maximálně 16 znaků. Datové rozhraní je 8bitové, ale FPGA posílá data jen po čtyřech vodičích, protože chce zachovat kompatibilitu s ostatními deskami společnosti Xilinx a také minimalizovat celkový počet pinů. Tento displej má jednu velkou nevýhodu a to je jeho rychlost. Perioda procesoru je 20ns, oproti tomu například příkaz na vymazání displeje trvá 1.64ms.

2.10.1 Řadič LCD

Tento znakový LCD displej obsahuje interní grafický řadič Sitronix ST7066U. Tento kontrolér tvoří tři interní paměti, každá z nich se specifickým účelem.



Obrázek 2.9: Připojení pinů FPGA k zobrazovači [4]

Název signálu	FPGA pin	Funkce
SF_D<11>	M15	datový bit DB7
SF_D<10>	P17	datový bit DB6
SF_D<9>	R16	datový bit DB5
SF_D<8>	R15	datový bit DB4
LCD_E	M18	povolení čtení/zápis 0: zakázáno 1: operace čtení/zápis povoleny
LCD_RS	L18	příkaz/data 0: Příkaz 1: data pro čtení nebo zápis
LCD_RW	L17	čtení/zápis 0: zápis 1: čtení

Tabulka 2.2: Rozhraní LCD displeje a význam jednotlivých bitů [4]

Nejprve displej inicializujeme než začneme pracovat s pamětmi.

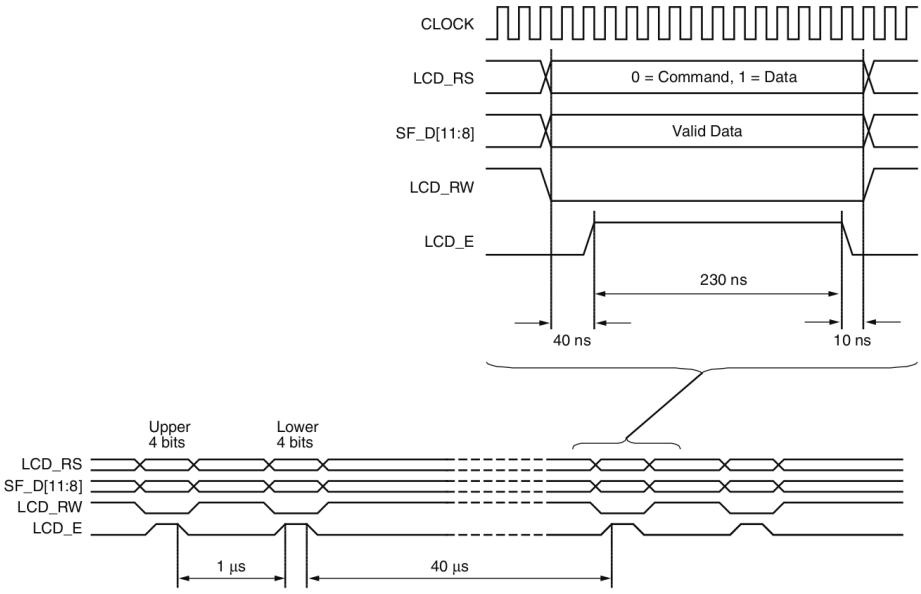
DD RAM Datová paměť ukládá kódy znaků, které se zobrazí na displeji. Uložené kódy znaků jsou vlastně odkazy do paměti CG ROM. Displej umožní zobrazit až 32 znaků, tudíž v každém řádku 16. První řádek je číslován od 0 do 15, hexadecimálně do F, ale číslování druhého řádku nenavazuje, nýbrž začíná hexadecimální adresou 40 a končí 4F.

CG ROM Kódy znaků uložené v DD RAM odkazují do této znakové paměti, kde pod každým kódem je adresa, na které je definovaný příslušný znak. Kódy znaku jsou kompatibilní s ASCII tabulkou.

CG RAM Prvních osm adres z CG ROM jsou vynechány právě pro paměť CG RAM. V CG RAM si můžeme definovat svých osm znaků, které jsou následně přístupné v CG ROM od adresy 0 do adresy 7. Jeden znak je složen ze 40pixelů rozdělených do osmi řádků a pěti sloupců.

2.10.2 Časování LCD

Jelikož datové rozhraní desky Starter Kit používá pouze 4bitové datové rozhraní, ale LCD používá 8 bitů dat, musíme data posílat postupně. Vyšší 4 bity budou odeslány jako první a následně po 1μs odešleme spodní 4 bity, na obrázku 2.10 můžete vidět přesné časování výměny dat pro tento konkrétní typ LCD. Mezi odesíláním jednotlivých příkazů musí být mezera podle typu příkazu.



Obrázek 2.10: Časovaní znakového LCD [4]

Kapitola 3

Návrh řešení

3.1 Jádro

Nejprve bylo zapotřebí se důkladně seznámit s vybraným jádrem a řádně jej otestovat. Hlavním požadavkem na jádro, byla plná funkčnost přeložených programů z vyšších jazyků, hlavně z jazyku C. Jádro se při testech jevilo plně funkční, takže přímo v jádru bylo potřeba dodělat pouze obsluhu přerušení.

3.1.1 Přerušení

Jediný zásah přímo do kódu jádra je implementování obsluhy přerušení. Celá logika jádra je v konečném automatu, takže nejschůdnější variantou pro přerušení je přidat jeden stav, ve kterém se provede obsluha přerušení.

3.2 Programová paměť

Programová paměť je generovaná z hex souboru za pomoci makra hex2vhdl. Zde nastal první problém. Toto makro generovalo špatné adresy. Program se pak choval v celku nahodile. Zde muselo dojít k úpravě souboru hex2vhdl.vhd. Po úpravě paměť poskytovala validní data. Velikost paměti je omezena velikostí programového čítače v samotném jádře.

3.3 IO prostor

IO prostor bude implementovaný jako jednotlivé speciálně funkční registry. Pravě čtená data vybere multiplexer ze všech těchto registrů podle adresy. Zápis bude proveden demultiplexerem.

3.4 Datová paměť

Datová paměť, pro uspoření samotné logiky na FPGA, bude implementována pomocí block RAM.

3.5 LED

LEDky jsou pouze výstupní periferie, bude pro ně vytvořen jeden registr v I/O prostoru. Jednotlivé diody budou připojeny přímo na příslušné bity registru.

3.6 Tlačítka a přepínače

Tyto periferie jsou na rozdíl od LED pouze vstupní. Deska je osazená čtyřmi tlačítky a stejným počtem přepínačů. Proto tyto periferie budou spojeny do jednoho 8bitového registru. Spodní 4 bity jsou určeny pro tlačítka a na horní 4 bity jsou přivedeny výstupy tlačítek.

Jelikož jsou tyto vstupy asynchronní, budou ošetřeny proti metastabilitě. Bude použita synchronizační buňka, složená ze dvou D klopných obvodů. Dva D klopné obvody metastabilitu izolují a další logice, v tomto případě registru, předají vyčištěný signál.

3.7 Rotační enkodér

Při otočení rotačním enkodérem bude vyvolané přerušení a výsledek, tedy směr otočení, bude uložen v registru. Nejprve je ale nutné z vstupních signálů odfiltrovat zákmity, které by signalizovali falešnou otočku.

3.8 UART

Sériová linka se skládá ze dvou základních částí, vysílače a přijímače. Každá z částí bude samostatný blok, což umožní plně duplexní přenos. U obou bloků bude možno programově nastavit rychlost přenosu a velikost odesílaných dat, 8 nebo 9bitový přenos.

Přijímací část bude obsahovat vstupní filtr. Vstupní filtr bude 3 bitový posuvný registr, který se při vzorkovací periodě $\frac{T_{clk}}{32}$ logicky posune o jedna doprava a na nejnižší místo dá vzorek vstupního, synchronizační buňkou již ošetřeného, signálu. Z těchto 3 bitů se pak majoritní funkcí vybere vstup pro další zpracování. Po kompletním přijetí, pokud bude programově povoleno, nastane přerušení.

Ve vysílací části nejsou žádné zvláštnosti. Obsahuje dva druhy přerušení. První přerušení bude vyvoláno, pokud je prázdný datový registr a druhé přerušení bude vyvoláno, když budou odeslána kompletně všechna data, to znamená, že v datovém registru nebudou další data k odeslání.

3.9 Znakové LCD

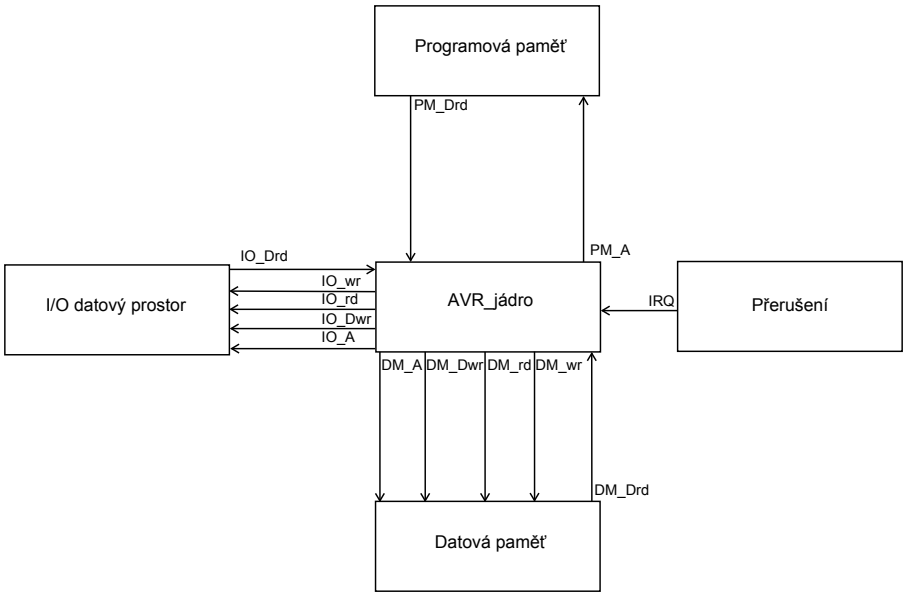
U LCD byli dvě možnosti řešení. První možnost displej kompletně obsluhovat softwarově, pomocí vytvořeného procesoru. Nad touto možností jsem kvůli úspoře hardwaru nejdříve uvažoval. Ale následně jsem tuto variantu vyloučil. Jednak jsem netrpěl nedostatkem místa, ale za druhé a to byl hlavní důvod, kvůli chybějícímu časovači. Na pomalý displej by se muselo čekat aktivním čekáním.

Zvolená řešení je obsluhovat LCD hardwarově, tedy pomocí logiky v FPGA. Tato možnost spočívá v návrhu konečného automatu a čítače. Stavový automat zajistí jak inicializaci, tak následně i kompletní obsluhu displeje. Za pomoci čítače zajistíme potřebné časování.

Kapitola 4

Řešení

4.1 ATtinyx61, top level



Obrázek 4.1: Propojení jádra s IO prostorem, datovou pamětí, programovou pamětí a přerušením

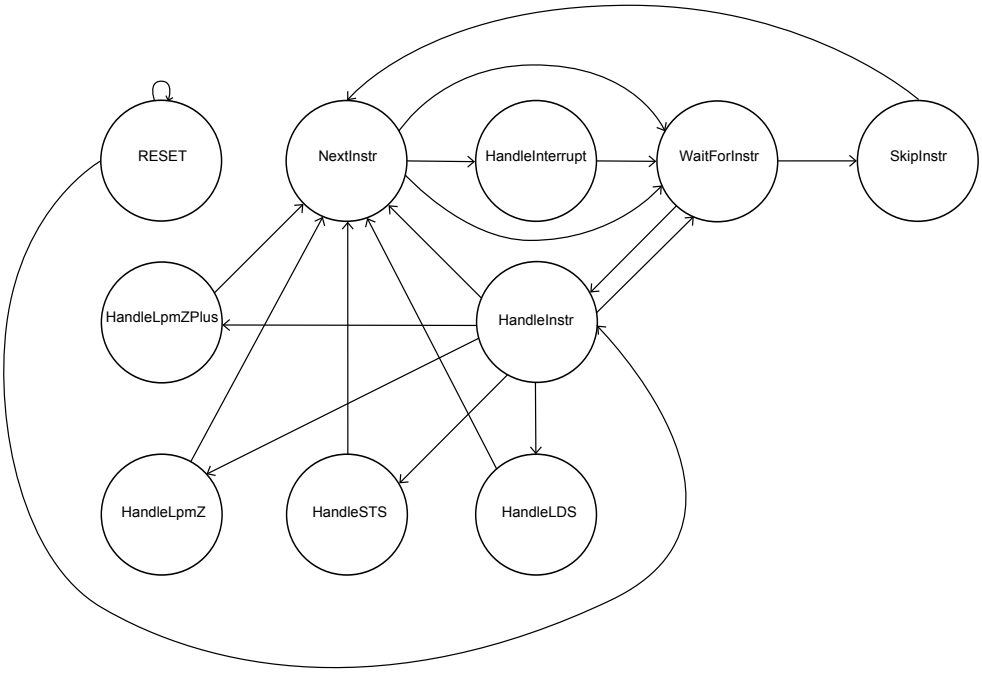
O propojení jádra s periferiemi, datovou pamětí, programovou pamětí a řadičem přerušením se stará top modul. Programová paměť je procesorem adresovaná pomocí 16 bitové adresy PM_A a procesoru pošle taktéž 16 bitová data po výstupní sběrnici PM_Drd. Datová paměť je adresována adresní sběrnicí DM_A. Při čtení se nastaví řídící signál DM_rd a data procesoru odešle po

výstupní sběrnici DM_Drd. Pro zápis je nastaven příznak DM_wr a datová paměť přijme data vyslaná procesorem po DM_Dwr. Komunikace mezi IO prostorem a jádrem probíhá obdobně jako u datové paměti.

4.2 ATtinyx61, jádro

Samotná logika jádra je implementována pomocí jednoho automatu 4.2. Dále jádro obsahuje 32 univerzálních registrů, stavový registr, ukazatel na vrchol zásobníku a samozřejmě programový čítač.

4.2.1 Řídící automat



Obrázek 4.2: Přechodová funkce automatu, který řídí celé jádro

Téměř veškerá složitost jádra je řešena automatem 4.2.

V úvodním stavu Reset se setrvá 16 hodinových taktů a vynuluje se všech 32 registrů. Následně se přejde do stavu HandleInstr, kde se vykoná první instrukce.

Ve stavu NextInstr se do programového čítače uloží následující instrukce. Pokud nastalo přerušení a zároveň je nastaven bit I v stavovém registru, přejde se do stavu HandleInterrupt, jinak se přejde do stavu WaitForInstr.

HandleInterrupt je stav obsluhující přerušení, podrobněji je popsán dále v sekci přerušení. Z tohoto stavu se vždy přechází do stavu WaitForInstr.

WaitForInstr čeká jeden hodinový takt na vystavení dat od programové paměti. Pokud je instrukce 32 bitová přejde se do stavu SkipInstr, jinak se pokračuje obsluhou instrukce.

Stav SkipInstr přeskočí právě prováděnou instrukci a vrátí se do stavu NextInstr, případně jedná-li se o instrukci LDS nebo STS ještě zvětší programový čítač o 1.

Ze všech nejsložitější je stav HandleInstr. V tomto stavu dojde nejprve k dekódování instrukce. Dekódování je pro všechny instrukce stejné, instrukce se liší v samotné výpočetní fázi. Všechny aritmetické a logické operace se provádějí ve třech hodinových taktech. V prvním taktu se vyčká na data z registru, samotný výpočet a uložení výsledku je provedeno ve druhém hodinovém taktu a v posledním taktu se aktualizuje stavový registr. Při práci se stavovým registrem, ať už nastavovaní nebo mazání některého z příznaků nebo podmíněné skoky závislé na SREG, je doba provedení nejkratší a to jeden hodinový takt. Jeden hodinový takt také trvá nepodmíněný skok. Volání podprogramu je provedeno ve třech taktech. V prvních dvou se uloží programový čítač na zásobník a následně je aktualizován programový čítač. U návratu z podprogramu se v prvním hodinovém taktu na adresní sběrnici datové paměti vystaví hodnota ukazatele na zásobník a ukazatel se následně inkrementuje. Další takt proběhne úplně stejný proces. Ve 3.taktu se čeká na data z datové paměti a v posledním čtvrtém taktu se aktualizuje programový čítač. Nejdéle se vykonává instrukce LD. Instrukce LD načítá nepřímo adresovaná data do registru. V prvních třech taktech načte data, ve čvrtém taktu je uloží do příslušného registru a následně inkrementuje X,Y nebo Z. Při predekrementaci se provede jako první aktualizace jednoho z adresních registrů a následně jsou načtená data a uložen výsledek. Ve všech instrukcích, ve kterých dochází k nastavení hodnoty programového čítače, se skáče do stavu WaitForInstr, jinak se vracíme do stavu NextInstr.

Do stavu HandleLDS se přejde ze stavu HandleInstr, při isntrukci LDS. V prvním taktu se čeká na data z registru. Ve druhém taktu se vyšle řídící signál DM_rd a na adresovou sběrnici se vystaví hodnota konstanty, která je uložená v druhých 16 bitech instrukčního slova. Jeden takt počkáme na data z SRAM a ve čvrtém taktu uložíme načtená data do registru a přesuneme se do stavu NextInstr.

HandleSTS ukládá data z registru do datového adresního prostoru adresovaného 16 bitovou adresou, uloženou v druhých 16 bitech instrukčního slova. Doba provádění jsou dva takty.

HandleLpmZ je obsluha instrukce LPM, která načte jeden byte adresovaný registrem Z z programové paměti do registru. V prvním taktu se čeká na data z programové paměti a ve druhém taktu se přečtená data uloží do registru.

Stav HandleLpmZPlus je stejný jako stav HandleLpmZ, jenom trvá o jeden takt déle a právě v posledním taktu je inkrementován registr Z.

4.2.2 Registrové pole

Registrové pole je implementováno dvěma pamětmi, ve VHDL kódu značené reg_regs_h a reg_regs_l. První paměť reg_regs_l obsahuje všechny sudé registry, tedy R0, R2 a postupně až R30 a reg_regs_h obsahuje všechny liché registry. Adresu konkrétního registru získáme při dekódování instrukce. Nejnižším bitem je vybrána paměť, pokud je nejnižší bit 0, vybraná paměť uchovává sudé registry, pokud 1 jedná se o lichý registr. Bity 1 až 4 jsou adresa vybraného registru. Pro zápis do jednoho registru musíme nastavit příznak reg_dWE_b a pro zápis do dvou po sobě jdoucích registrů příznak reg_dWE_w. Data z registrů jsou pro instrukci k dispozici jeden takt po dekódování instrukce.

4.2.3 Přeřušení

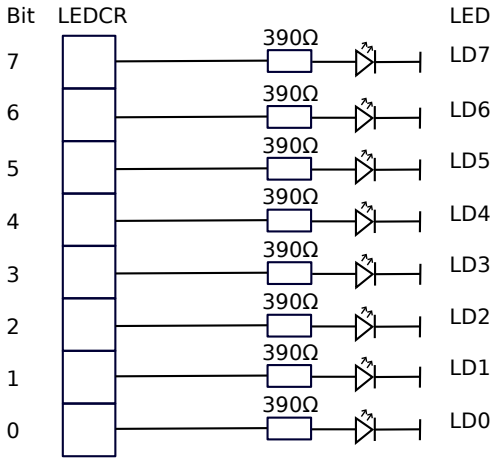
O obsluhu přerušení se stará stav HandleInterrupt celkem ve čtyřech taktech. V první taktu je resetován příznak I ve stavovém registru. V dalším taktu je uloženo spodních 8 bitů programového čítače a ukazatel zásobníku je snížen o 1. Ve třetím taktu je uloženo vrchních 8 bitů programového čítače a ukazatel zásobníku opět o 1 dekrementován. V posledním čtvrtém taktu je do programového čítače uložen vektor přerušení. O výběr přerušení s nejvyšší prioritou se stará prioritní kodér.

4.3 Programová paměť

Programová paměť je realizována VHDL kódem, který je vygenerován z přeloženého hex souboru, spuštěním programu hex2vhdl.hdl. Programový čítač (PC) je implementován jako 16 bitový registr, tudíž umožňuje adresovat až 64K instrukcí, tedy 128 KB programové paměti. Bohužel jsem nenašel, jak změnit nastavení kompilátoru pro adresování větší paměti než je daná výrobcem pro konkrétní typ. Po překročení velikosti 8KB, což je velikost pro typ ATtiny861, vyhodí překladač chybu.

4.3.1 Datová paměť

Datová paměť je fyzicky implementovaná v block RAM. Datový prostor může mít celkem velikost až 64KB, díky 16 bitové adrese a 16 bitovému SP. Velikost je opět omezena hlavně překladačem, jelikož na začátku je provedena inicializace zásobníku a tu překladač provádí podle konkrétního typu AVR jádra. Datová paměť v této konkrétní implementaci má 512B.



Obrázek 4.3: Připojení LED

4.4 LED

LEDky jsou přímo připojeny na příslušný registr v IO prostoru. Automaticky reagují, zhasnou nebo se rozsvítím po změně dat v tomto registru.

LEDs Control Register - LEDCR

Bit	7	6	5	4	3	2	1	0
0x0E	MSB							LSB
čtení/zápis	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	0	0	0	0	0	0	0	0

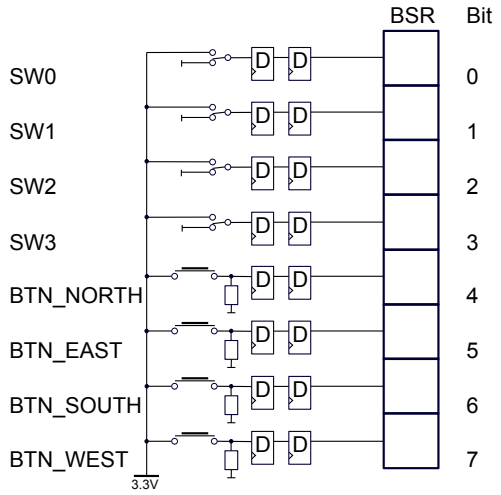
- 7. až 0. bit
  Pokud je bit nastaven, příslušná LEDka svítí.

4.4.1 Přepínače a tlačítka

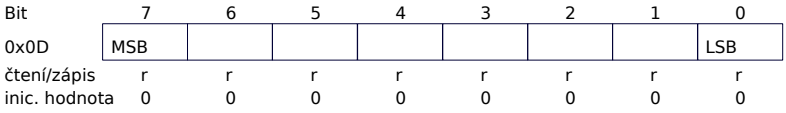
Přepínače a tlačítka jsou vstupní asynchronní signály a ty musí být ošetřeny proti metastabilitě. Ochranu proti metastabilnímu chováni zajišťují dva D klopné obvody, které vidíte na obrázku 4.4 ihned za vstupy z tlačítek nebo přepínačů. První registr navzorkuje asynchronní signál a na jeho výstupu může docházet k metastabilnímu chování. Po jednom taktu hodin druhý registr navzorkuje výstup prvního registru a to by již měl být signál ustálený a výstup druhého D klopného obvodu je správný.

Výstupy ze synchronizačních buněk jsou přivedeny přímo na jednotlivé bity speciálního registru v IO prostoru.

Buttons Status Register - BSR



Obrázek 4.4: Připojení přepínačů a tlačítek



- 7. až 0. bit
  Výstupy přepínačů jsou přivedeny na spodní 4 bity, tlačítka jsou připojena na horní bity registru BSR. Logická 1 se na příslušném bitu objeví, pokud konkrétní tlačítko nebo přepínač připojí ke vstupu napětí.

4.4.2 Rotační enkodér

Prvním bodem implementace je odfiltrovat z vstupního signálu zákmity. Vstupní filtr vstupního signálu A je 1 bitový registr, který je nastaven do 1 v případě, že jsou oba vstupní signály v 1 a resetován, pokud jsou oba vstupní signály 0, v ostatních případech si pamatuje předchozí hodnotu. Filtrační registr vstupu B je nastaven, pokud vstup A se rovná 0 a vstup B 1 a resetován, když A je rovno 1 a vstup B se rovná 0. Jinak se obsah registru nemění.

Následně již pracujeme s odfiltrovaným signálem. Při náběžné hraně vyfiltrovaného signálu A, dojde k vyvolání přerušení. Následně pokud se vyfiltrovaný signál B rovná 0 nastaví se příznak RL, otočení vlevo, jinak se nastaví příznak RR, tedy došlo k rotaci vpravo.

Rotary Encoder Register - RER

Bit	7	6	5	4	3	2	1	0
0x12	-	-	-	-	REIE	RCB	RL	RR
čtení/zápis	r	r	r	r	r/w	r	r	r
inic. hodnota	0	0	0	0	0	0	0	0

- **RR** - Rotary Right
  Tento bit je nastaven po otočení enkodéru vpravo. Je nastaven dokud nedojde k otočení vlevo.
- **RL** - Rotary Left
  Indikuje otočení vlevo. Po nastavení je vynulován pouze otočením vpravo.
- **RCB** - Rotary Center Button
  Pokud je stlačeno tlačítko CENTER je tento bit nastaven.
- **REIE** - Rotary Encoder Interrupt Enable
  Povolení nebo zakázání přerušení od rotačního enkodéru.
- 4. až 7. bit
  Jsou nevyužity, při čtení se přečte 0.

4.4.3 UART

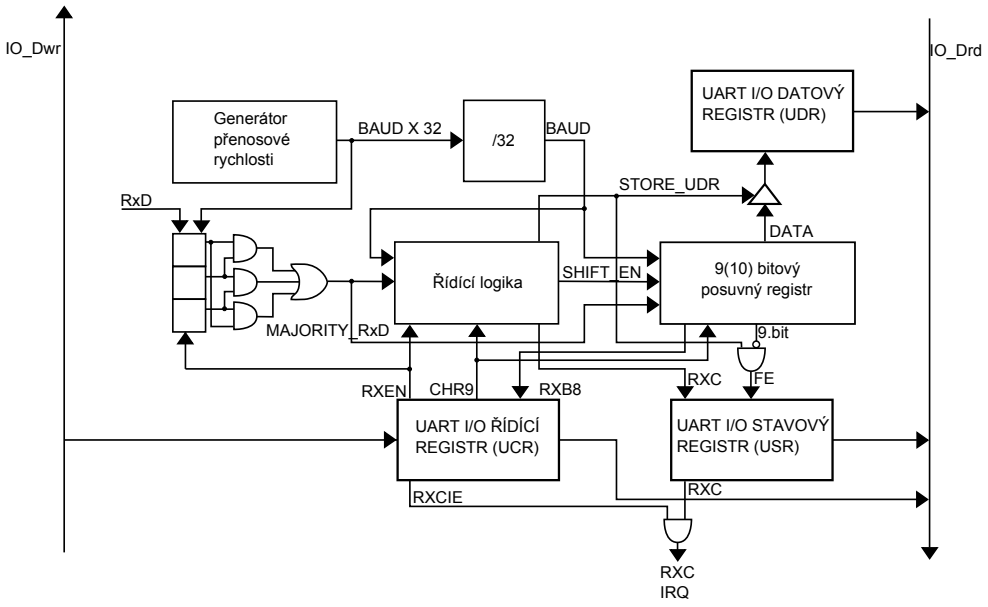
Univerzální asynchronní sériový kanál je implementován jako dva nezávislé bloky, přijímač a vysílač , což umožňuje plně duplexní přenos. Pro práci se sériovým kanalém jsou k dispozici čtyři registry, datový, stavový, řídící a registr pro nastavení přenosové rychlosti. Navrhovaný design sériového kanálu se vyznačuje:

- Možnost volby mezi 8 a 9 bitovým přenosem
- Kontrola STOP bitu
- Vstupní filtr
- Programové určení přenosové rychlosti
- Generuje tři druhy přerušení

4.4.3.1 Přijímač

Blok přijímače 4.5 je implementován jako čtyři základní části, které jsou mezi sebou navzájem provázané.

První část je vstupní filtr, což je 3 bitový posuvný registr. Tento obvod si vzorkuje vstupní signál s frekvencí 32krát vyšší, než je rychlost přenosu sériového kanálu. Všechny bity z tohoto posuvného registru jsou připojeny k majoritní funkci. Výstup majority, tedy vyfiltrovaný signál (MAJORITY_RxD) je vstupním signálem kontrolní logiky.



Obrázek 4.5: UART - blokové schéma přijímače

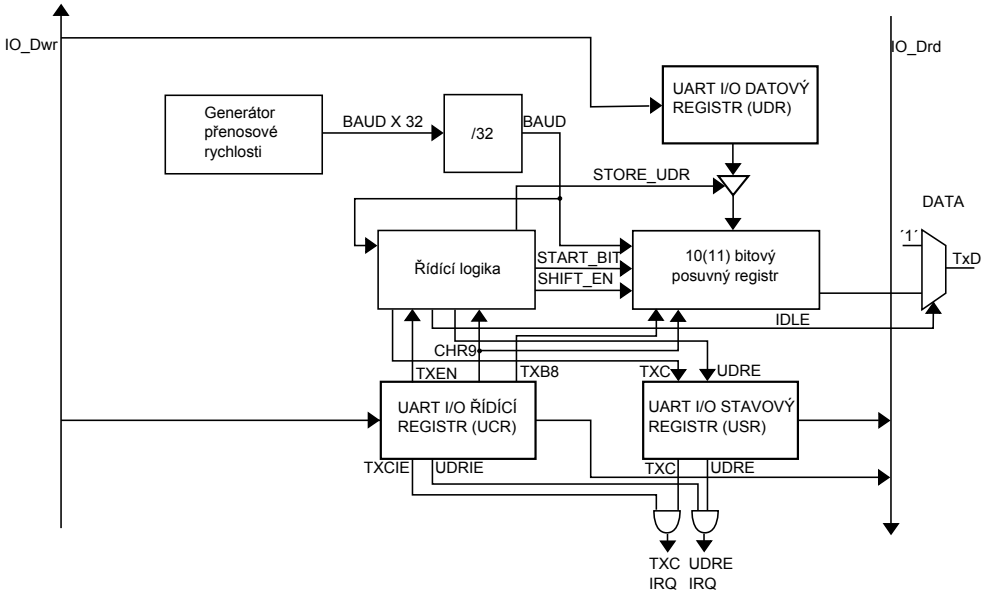
Kontrolní logika řídí celý obvod. Na začátku přenosu kontrolní logika čeká na logickou 0 na vstupním signálu (MAJORITY_RxD), což značí start bit. Následně nastaví čítač pro příjem dat na 9 nebo 10, v závislosti na signálu CHR9 a čeká jeden a půl BAUDu, tedy přesně do půlky prvního přijímaného bitu. Po přijetí start bitu se nastaví příznak na logickou 1 příznak SHIFT_ENABLE. Tím začíná postupný přesun dat i s závěrečným stop bitem do posuvného registru.

Samotná data společně se stop bitem, se postupně po jednotlivých bitech posouvají do posuvného registru. Po kompletním přijetí všech dat, nastaví kontrolní logika signál STORE_UDR, tím se zapíše první až osmý bit do UDR, v případě že šlo o 8 bitový přenos. Při 9 bitovém přenosu se do UDR zapíše nultý až sedmý bit a osmý bit posuvného registru se zapíše do RXB8 v řídícím registru. Devátý bit se zneguje a zapíše do příznaku FE ve stavovém registru.

Poslední část přijímače je generátor přenosové rychlosti. Generátor je rozdělen do dvou částí. První část generuje 32krát vyšší frekvenci než je frekvence posílaných dat po sériovém kanále. Jedná se o čítač, který při resetu nebo nulové hodnotě čítače nastaví na hodnotu z registru přenosové rychlosti (UBRR) a postupně čítá do nuly. Pokud je čítač v nule nahodí se signál BAUDX32. Druhá část generátoru je obyčejná dělička, která vydělí signál BAUDX32 32krát.

4.4.3.2 Vysílač

Vysílací část UARTu 4.6 se skládá ze 3 základních částí, zde není potřeba řešit vstupní filtr, tedy vysílač má o jeden blok méně než přijímač.



Obrázek 4.6: UART - blokové schéma vysílače

Kontrolní logika ovládající celý obvod čeká než budou k dispozici data k odeslání, to znamená, že data budou zapsána do UDR a tím bude resetován příznak UDRE. Následně v závislosti na příznaku CHR9 se nastaví datový čítač na 10 nebo 11. Nastaví se příznak SHIFT_ENABLE a postupně se odešlou všechna data, začínající start bitem a končící logickou 1 neboli stop bitem.

Posuvný registr je po nastavení signálu STORE_UDR naplněn daty. Na nultý bit se uloží 0, následují data z datového registru, na devátý bit je přiveden signál TXB8 z řídícího registru nebo uložena logická 1, v závislosti na příznaku CHR9. Desátý bit je vždy na začátku nahozen. Data se odesílají pokaždé, když signály BAUD a SHIFT_ENABLE jsou zároveň nastaveny.

Třetí, poslední blok vysílače je generátor přenosové rychlosti, který je podrobně popsaný u přijímače.

UART Data Register - UDR

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

I/O datový registr je ve skutečnosti implementován dvěma různými registry. Oba registry sdílejí shodnou I/O adresu. Po přijetí dat se data fyzicky zapíší do RXD_UDR a následně i z toho registru čtou. Programově se zápisuje do registru TXD_UDR.

UART Control Register - UCR

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

- **TXB8** - Transit Data Bit 8
  Devátý odesílaný datový bit, pokud je příznak CHR9 v logické 1.
- **RXB8** - Recieve Data Bit 8
  Je-li nastaven CHR9, do tohoto příznaku se přesune osmý bit z posuvného registru, po přijetí kompletních dat.
- **CHR9** - I-bit Character
  Bit který nastavuje velikost odesílaných nebo přijímaných dat. Pokud je tento bit nastaven mají data velikost 9 bitů. Jinak mají data o 1 bit méně, tedy 8 bitů.
- **TXEN** - Transmitter Enable
  Nastavením tohoto bitu se zapne vysílač UARTu. Pokud je tento bit vynulován během odesílání dat, aktuálně posílaná data se dopošlou.
- **RXEN** - Reciever Enable
  Povolení nebo zakázání přijímače UARTu.
- **UDRIE** - UART Data Register Empty Interrupt Enable
  Tento bit povoluje přerušení pokud je vysílací datový registr TXD_UDR prázdný.
- **TXCIE** - TX Complete Interrupt Enable
  Povolení přerušení, které je vyvoláno po odeslání všech dat.
- **RXCIE**- RX Complete Interrupt Enable
  Nastavením tohoto bitu povoluje přerušení po přijetí celého znaku.

UART Status Register - UCR

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

- 0. až 3. bit
  Jsou nevyužity, při čtení se přečte 0.

- **FE** - Framing Error
  Pokud je u přicházejícího znaku nulový stop bit je tento bit nastaven.

- **UDRE** - UART Data Register Empty
  Tento bit značí, že vysílač je schopen přijmout další znak k vysílání. UDRE je nastaven, pokud se data z vysílacího datového registru, přesunou do posuvného registru. Pokud jsou data zapsána do vysílacího datového registru je tento bit resetován. Při povoleném přerušení, tedy ldyž jsou nastaveny příznaky UDRIE a ve status registru příznak globálního přerušení, je vyvolávána obslužná rutina, dokud je UDRE nastaven.

- **TXC** - UART Transmit Complete
  Je-li tento bit nastaven, tak jsou již odeslány všechny data, ve vysílacím datovém registru již není další znak k odeslání. Při povoleném přerušení je vyvoláno přerušení vysílání dokončeno. Tento bit je nutné programově resetovat v obslužné rutině, jinak by nastalo ihned po skončení obslužné rutiny nové přerušení.

- **RXC** - UART Receive Complet
  Při přijetí kompletních dat, dojde k nastavení bitu RXC, k vynulovaní dojde při čtení z UDR. Nastavením RXC, pokud je povolené přerušení program skočí do přerušovací rutiny, kde musí dojít k přečtení UDR.

UART Baud Rate Register - UBRR

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

- 0. až 7. bit
  Nastavením obsahu registru UART registru přenosové rychlosti určuje rychlost přenosu po sériovém kanálu.

$$BAUD = \frac{f_{CLK}}{32 * (UBRR + 1)}$$

$$UBRR = \frac{f_{CLK}}{32 * BAUD} - 1$$

f = 50 MHz		
Baud rate	UBRR	Chyba [%]
9600	162	0,15
14400	108	0,45
19200	80	0,47
28800	53	0,47
38400	40	0,76
57600	26	0,47
76800	19	1,73
115200	13	3,12

Tabulka 4.1: Tabulka používaných přenosových rychlostí a k nim příslušné hodnoty registru UBRR.

4.4.4 Znakové LCD

Ovládání dvouřádkového znakového LCD je řešeno v samotné logice FPGA a programově k displeji lze přistupovat přes tři registry, které se nachází V I/O adresním prostoru. Příznak lcd_rw je stále vynulovám, jelikož se z displeje nebude číst. Signál lcd_enable musí být nastaven vždy, když na displej posíláme jakýkoliv příkaz nebo data. Pokud je lcd_enable vynulován, displej ignoruje všechny vstupy. Samotný řadič displeje, implementovaný v logice FPGA, je implementován stavovým automatem 4.7 a čítačem.

Nejprve je potřeba automat inicializovat, což obstará prvních 5 stavů automatu. Ve stavu START se čeká alespoň 15 ms, což je 750000 taktů při frekvenci 50 MHz, než se nakonfiguruje samotné FPGA. Následně proběhne samotná inicializace displeje. V dalším stavu POWER_UP_1 se po dobu 12 taktů nastaví na datové vodiče hodnota 3 a nahodí příznak lcd_enable. Ve stavu POWER_UP_1 se setrvá, dokud čítač nedočítá na hodnotu 205014. V následujícím stavu POWER_UP_2 opět na 12 taktů nastavíme hodnotu 3 na čtyři datové vodiče a až se čítač bude rovnat 5014 přejdeme do stavu POWER_UP_3. Opět odešleme hodnotu 3 a v tomto stavu vyčkáme 2014 hodinových taktů. V posledním inicializačním stavu na 12 taktů na datové vodiče vystavíme hodnotu 2 a po dočítání čítače do hodnoty 2014 je inicializace hotova.

Ještě před samotným zobrazováním znaků přímo na displej je potřeba displej počátečně nakonfigurovat. Počáteční konfigurace obsahuje 4 příkazy, které obstarává následných 8 stavů automatu. Každý příkaz je rozdělen do dvou stavů automatu, jelikož všechny příkazy a data jsou 8 bitové a tedy jsou posílána na 2krát 2.10. První příkaz nastaví znakovou sadu, deska Starter Kit obsahuje pouze jednu sadu. V prvních dvou stavech FUNCTION_SET_H/L odešleme hodnotu 28 hexadecimálně. A počkáme 40 μs. Další příkaz nastaví jakým směrem se bude posouvat kurzor, tedy jestli se bude adresa, na kterou

kursor ukazuje inkrementovat nebo dekrementovat. Naše implementace při zápisu na displej bude kurzor klasicky posouvat doprava, tedy adresu inkrementovat. Displeji odešleme hexadecimální hodnotu 6 a vyčkáme zde taktéž 40 μs. V následných dvou stavech DISPLAY_ONOFF_H/L displeji odešleme hexadecimální hodnotu C a tím zapneme LCD. Prováděcí doba je opět 40 μs. Poslední konfigurační příkaz je vymazání celého displeje. Hexadecimální hodnota, kterou musíme předat displeji, je 1. Celkem v těchto dvou stavech CLEAR_DISPLAY_H/L setrváme výrazně delší dobu a to konkrétně 82078 hodinových taktů. Příkaz vymaž displej totiž fyzicky přepíše celý displej mezerami a nastaví kursor na adresu 0, což je levý horní roh znakové LCD.

Po počáteční inicializaci a konfiguraci se nacházíme ve stavu čekajícím na příkaz programátora, který má čtyři možnosti. První z možností je vypsat na displej libovolný znak z vybrané sady, druhá možnost je nastavit pozici kursoru, další možnost je vrátit kursor do levého horního rohu a poslední možností je vymazání celého displeje.

LCDDR - LCD Data Register

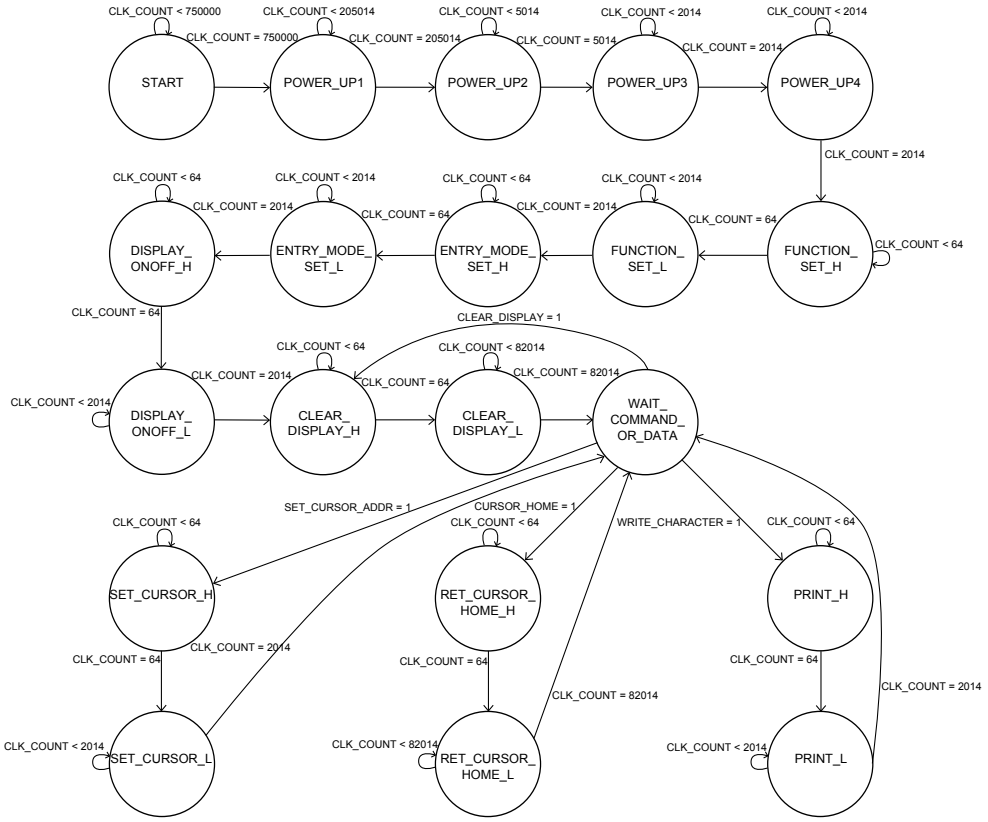
Bit	7	6	5	4	3	2	1	0
0x0E	MSB							LSB
čtení/zápis	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	0	0	0	0	0	0	0	0

- 0. až 7. bit
  8 bitů označujících znak, který při současně nastaveném příznaku WCH se zobrazí na displeji.

LCDCR - LCD Control Register

Bit	7	6	5	4	3	2	1	0
0x10	LCDR	-	-	-	CH	CD	SCA	WCH
čtení/zápis	r/w	r	r	r	r/w	r/w	r/w	r/w
inic. hodnota	0	0	0	0	0	0	0	0

- **WCH** - Write Character
  Nastavením WCH se data uložená LCDDR vytisknou na displej.
- **SEA** - Set Address Cursor
  Po nastavení signálu SEA, se změní pozice kursoru na pozici zapsanou v registr LCDCAR.
- **CD** - Clear Display
  Nastavením tohoto příznaku smažeme celý displej.
- **CH** - Cursor Home
  Nastaví kursor do levého horního rohu obrazovky, první řádek první sloupec.



Obrázek 4.7: Přechodová funkce stavového automatu ovládajícího znakový LCD

- 4. až 6. bit
  Jsou nevyužity, při čtení se přečte 0.

- **LCDR** - LCD Ready
  Je-li nastaven, LCD je připravenu zpracovat další příkaz.

LCDCAR - LCD Cursor Address Register

Bit	7	6	5	4	3	2	1	0
0x11	MSB							LSB
čtení/zápis	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w
inic. hodnota	1	0	0	0	0	0	0	0

- 0. až 6. bit
  7 bitů adresy kursoru.

- 7. bit
  Tento bit je vždy nastaven na logickou 1, protože příkaz pro nastavení kursoru má sedmý bit 1.

4.5 Knihovna v programovacím jazyku C

Pro všechny periferie byla vytvořena knihovna v jazyku C. Všechny periferie mohou číst nebo zapisovat do datových registrů. Dále mohou pomocí vytvořených funkcí inicializovat a konfigurovat periferie. Periferie vyvolávají také 4 různá přerušení, 3 UART a 1 rotační kodér. Jelikož procesor ATtiny861 nemá sériovou linku ani rotační kodér, tyto přerušení museli být implementovány na místa, kde byla původně jiná přerušení. Konkrétně přerušení vyvolané přijetím znaku, má vektor shodný s externím přerušením 0 (INT0), přerušení od rotačního kodéru je implementováno na místo PCINT, přerušení od hlídacího obvodu (WDT) je nahrazeno přerušením odeslání dokončeno a při prázdném odesílacím datovém registru je vyvoláno externí přerušení 1 (INT1). Každý překladač má pro konkrétní typ procesoru napevno deklarovanou tabulku vektorů přerušení, tedy při deklarování obsluhy přerušení je nutné použít původní názvy přerušovacích vektorů.

Kapitola 5

Testování

Nejdůležitější testování, proběhlo již na samotném začátku, při výběru procesoru. Procesoru byli za pomocí jazyka symbolických adres otestovány všechny implementované instrukce. Hlavní důraz bylo otestovat nastavování příznaků a následně testování podmíněných skoků. V tomto se vyskytuje nejvíce chyb. Všechny instrukce vykazovali správné výstupy. Následně byl program otestován programem napsaným v jazyce C ,vytvořeným za pomocí vývojových nástrojů firmy Atmel. Program seřadil pole pomocí jednoduchého řadícího algoritmu BubleSort. Procesor se podle výsledků ze simulátoru ModelSim [13] jevil plně funkční.

Další testování proběhlo při vývoji sériové linky a LCD. Stejně jako sériová linka, byl i její test rozdělen do dvou částí. Samostatně byl testován přijímač a vysílač. Pro oba bloky byl napsán testbench soubor. Na vstupy obou bloků byli přivedeny všechny možné vstupní kombinace. Po behaviární simulaci se přijímač i vysílač sériové linky jevily taktéž plně funkční.

Simulace LCD zobrazovače byla zaměřena na otestování dvou základních funkčností. Nejprve byla zkontrolována přechodová funkce automatu. A následně změřeny příslušné intervaly doby provádění jednotlivých příkazů. Všechny intervaly seděly podle specifikace.

Fyzické a komplexní testování proběhlo přímo na desce vzorovou aplikací. Vzorová aplikace umožňuje vyzkoušet stlačení tlačítka nebo změnu polohy přepínače rozsvícením, případně zhasnutím příslušné LED. Ten samý test je zvolen pro testování LED. Samotné vybírání položky z menu testuje rotační enkodér. Test sériové linky je rozdělen do dvou částí. První část „Test RxD“ testuje přijímač. Vypisuje znaky přijaté po sériovém kanálu cyklicky na první řádek displeje. Test vysílače odešle po sériovém kanálu text „"Test transmitter"“. Tyto dílčí testy si můžete sami vyzkoušet po nahrání vygenerovaného bitstreamu do FPGA.

Závěr

Vybrat procesor s ohledem na minimální spotřebu s možností využití stávajících nástrojů firmy Atmel, umožňující programování jádra jazykem C, byl první bod zadání. Na základě těchto dvou požadavků se jako nejschůdnější volba jevilo jádro AVRtinyx61 [1], poskytované pod LGPL licencí. Procesor byl zdokumentován a upraven pro implementaci do FPGA Spartan 3E a konečné parametry jádra jsou:

- Maximální frekvence 106 MHz, procesor je taktován 50 MHz oscilátorem, který je přímo na desce Spartan-3E Starter kit.
- 120 výkonných instrukcí
- 32 8bitových registrů
- Až 4096 instrukcí
- Datová paměť 512B
- Implementovány 4 přerušení

Ve VHDL kódu je vytvořeno rozhraní pro komunikaci procesoru s obvodem RS232, znakovým LCD, rotačním kodérem, tlačítky, přepínači a LED. Periferie lze ovládat prostřednictvím speciálně funkčních registrů, mapovaných do IO prostoru procesoru. K ovládání periferií slouží vytvořená C knihovna.

Změnou konstant ve VHDL kódu, lze paměť fyzicky rozšířit až do kapacity zdrojů na vybraném FPGA. Proti rozšíření byl překladač, který má pro konkrétní typ pevně stanovenou velikost paměti.

Na vytvořené vzorové aplikaci si můžete vyzkoušet funkčnost procesoru i s přidanými periferiemi. Procesor je možno využít jako stavební kámen pro další práci. Je možno k němu doimplementovat složitější periferie (Ethernet, USB) a využít ho pro reálnou aplikaci.

Literatura

[1] Hilvarsson, A.: AVRtinyX61core. [online], 2008, [cit. 2014-04-28]. Dostupné z: http://opencores.org/project,avrtinyx61core

[2] Xilinx: *Spartan-3E FPGA Family Data Sheet*. Xilinx, 2013, [cit. 2014-05-05]. Dostupné z: www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[3] Atmel: *ATtiny261/461/861*. Atmel, 2013, [cit. 2014-05-07]. Dostupné z: http://www.atmel.com/Images/Atmel-2588-8-bit-AVR-Microcontrollers-tinyAVR-ATtiny261-ATtiny461-ATtiny861_Datasheet.pdf

[4] Xilinx: *Spartan-3E FPGA Starter Kit Board User Guide*. Xilinx, 2011, [cit. 2014-05-08]. Dostupné z: http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[5] Xilinx: *PicoBlaze 8-bit Embedded Microcontroller User Guide*. Xilinx, 2011. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

[6] Xilinx: *MicroBlaze Processor Reference Guide*. Xilinx, 2008. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[7] Altera: *Nios II Processor Reference Handbook*. Altera, 2014. Dostupné z: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[8] OpenCores: OpenCores Projects. [online], 1999, [cit. 2014-04-28]. Dostupné z: www.http://opencores.org/projects

[9] Jakub Šťastný: *FPGA prakticky*. Praha: BEN - technická literatura, první vydání, ISBN 978-80-7300-261-9.

[10] Atmel: Atmel AVR 8-bit and 32-bit Microcontrollers. [online], 2014, [cit. 2014-05-07]. Dostupné z: http://www.atmel.com/products/microcontrollers/avr/

[11] Ken Chapman: *Rotary Encoder Interface for Spartan-3E Starter Kit.* Xilinx, 2006, [cit. 2014-05-10]. Dostupné z: http://www.eng.utah.edu/~cs3710/xilinx-docs/examples/s3esk_rotary_encoder_interface.pdf

[12] Vladimír Váňa: *Mikrokontroléry ATMEL AVR.* Praha: BEN - technická literatura, první vydání, ISBN 80-7300-083-0.

[13] ModelSim PE Student Edition. [online], [cit. 2014-04-28]. Dostupné z: http://www.mentor.com/company/higher_ed/modelsim-student-edition

Příloha **A**

Seznam použitých zkratek

FPGA Field Programmable Gate Array

RISC Reduced Instruction Set Computing

LUT Look Up Table

KCPSM (K)constand Coded Programmable State Machine

RAM Random Access Memory

EDK Embedded Development Kit

SBT Software Build Tools

ALU Arithmetic Logic Unit

SRAM Static Random Access Memory

RAM Input Output

PC Program Counter

VHDL Very High Speed Integrated Circuit Hardware description language

DDR SDRAM Double Data Rate Synchronous Dynamic Random Access Memory

SPI Serial Peripheral Interface

LCD Liquid Crystal Display

VGA Video Graphics Array

USB Universal Serial Bus

LED Light Emitting Diode

UART Universal Asynchronous Reciever and Transmitter

LVTTL Low Voltage Tranzistor-Tranzistor Logic

LVCMOS Low Voltage Complementary Metal Oxide Semiconductor

ROM Read Only Memory

ASCII American Standart Code for Information Interchange

LGPL Lesser General Public License

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bit` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS