VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

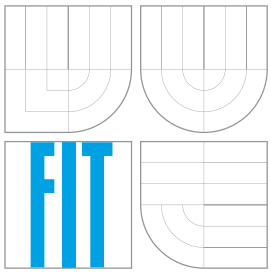
FITKIT JAKO PROGRAMÁTOR MIKROKONTROLERŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN MAREŠ
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FITKIT JAKO PROGRAMÁTOR MIKROKONTROLERŮ

FITKIT AS A PROGRAMMER OF MICROCONTROLLERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN MAREŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá možností využití FITkitu jako programátoru mikrokontrolérů Microchip PIC a Atmel AVR. Rozvádí možnosti programování mikrokontrolérů a postupy, které je třeba při programování dodržet. V další části popisuje praktickou implementaci programu pro FPGA, který je schopen tyto mikrokontroléry programovat a také implementaci příslušné ovládací aplikace pro PC. Nakonec práce zhodnocuje výsledky s ohledem na rychlost programování.

Abstract

This thesis explores the possibilites of using FITkit as a programmer of Microchip PIC and Atmel AVR microcontrollers. It describes methods of microcontroller programming and procedures that need to be followed to program a microcontroller. It also details the implementation of a FPGA design able to program these microcontrollers and a corresponding application for PC. At last, it evaluates the results, with focus at programming speed.

Klíčová slova

Mikrokontrolér, MCU, FITkit, AVR, Atmel, PIC, Microchip, FPGA

Keywords

Microcontroller, MCU, FITkit, AVR, Atmel, PIC, Microchip, FPGA

Citace

Jan Mareš: FITkit jako programátor mikrokontrolerů, bakalářská práce, Brno, FIT VUT v Brně, 2012

FITkit jako programátor mikrokontrolerů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka.

.....
Jan Mareš
14. května 2012

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce za jeho odborné rady, které mi pomohly dokončit tuto práci.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout a implementovat zařízení a ovládací aplikaci tohoto zařízení, které budou dohromady schopny programovat vybrané typy mikrokontrolérů. Požadavkem je, aby zařízení vyžadovalo minimální počet externích komponent a dosahovalo co možná nejvyšších programovacích rychlostí. Rozsah podporovaných mikrokontrolérů by měl zahrnovat nejdostupnější typy.

Programováním mikrokontrolérů se zde nemyslí samotný proces implementace zdrojového programu pro řízení mikrokontroléru, ale zápis binárních dat do programové paměti mikrokontroléru a případně do datové a konfigurační paměti, pokud ji mikrokontrolér obsahuje.

Pro tuto práci jsem zvolil mikrokontroléry od společností Microchip PIC a Atmel AVR.

Motivací k vytvoření této práce mi byla relativní složitost použití těchto zařízení, hlavně pro potřeby výuky. Mikrokontroléry od společností Atmel a Microchip jsou velmi rozšířené a dostupné, ale využívají pro nahrávání dat proprietární protokoly a není jednoduché je připojit přímo k počítači. Zařízení pro programování těchto obvodů jsou pro vyukové použití nákladná a student, který by si chtěl postavit programátor vlastní, by narážel na různé překážky. Například nejjednodušší programátory se k počítači připojují přes paralelní port, který je ale zastaralý a většina moderních počítačů (hlavně notebooků) ho neobsahuje.

FITkit je k tomuto využití velmi vhodný. Obsahuje mikrokontrolér i hradlové pole, které je schopno při správném použití dosahovat působivých rychlostí u paralelizovatelných aplikací. Připojuje se přes rozhraní USB, které je v současné době velmi rozšířené. Každý student FIT si ho může zapůjčit.

Tato práce se tedy bude zabývat implementací takového programátoru do FITkitu a příslušné ovládací aplikace. Rozeberu zde vlastnosti mikrokontrolérů Atmel AVR a Microchip PIC a jejich možnosti programování. Poté se budu zabývat implementací programátoru do FITkitu a ovládací aplikace v PC. Obrázky a tabulky, které by v samotném textů působily rušivě, jsou umístěny v přílohách, přičemž text práce se na ně pochopitelně odkazuje. Hexadecimální čísla jsou v této práci zapsána jako v jazyce C, například 0xA5.

Kapitola 2

Základní prvky

2.1 Mikrokontrolér

Mikrokontrolér je zařízení, které kombinuje mikroprocesor, paměť pro ukládání programu, vstupně-výstupní periferie a často i RAM paměť. Na rozdíl od mikroprocesorů se používají ve vestavěných zařízeních. Ročně se na celém světě prodají miliardy takových mikrokontrolérů a obsahují je rozmanitá zařízení od kuchyňských minutek až po ovládání prvků letadel. Ačkoliv existují mikrokontroléry s velkým množstvím paměti a rozmanitých periferií, zařízení často nemají velké nároky na paměť a výkon a tudíž se ve velkém množství případu využívají právě ty nejmenší (a pochopitelně nejlevnější) typy mikrokontrolérů. [9] Mikrokontroléry vyrábí mnoho společností a k dispozici jsou různé typy pro všechna možná použití.

Mikrokontroléry se běžně vyrábí 8bitové, 16bitové a 32bitové. Většina 32bitových a značná část 16bitových mikrokontrolérů se programuje pomocí rozšířeného rozhraní JTAG, které umožňuje často provádět i ladění přímo na čipu. 8bitové mikrokontroléry jsou ale nejrozšířenější a z toho důvodu se v této práci budu věnovat právě těmto typům.

2.2 FITkit

FITkit je platforma vytvořená pro výuku programování vestavěných systémů. Jde o samostatné zařízení, které obsahuje mikrokontrolér *MSP430* od společnosti Texas Instruments a hradlové pole *XC3S50* od firmy Xilinx. Dále obsahuje řadu periferních zařízení jako FTDI čip pro USB komunikaci, dynamickou RAM paměť, displej, klávesnici a jiné. Díky možnosti neomezeně reprogramovat svůj obsah vytváří skvělou platformu pro vzdělávání studentů v oblasti vestavěných systémů. [16]

Hradlové pole, které FITkit obsahuje, umožňuje vytvářet hardwarové logické obvody bez nutnosti použití diskrétních součástek a mnoha integrovaných obvodů. Jeho použití je tedy vhodné u aplikací, které je možné paralelizovat a zapsat v podobě logického obvodu.

Kapitola 3

Mikrokontroléry Atmel AVR

3.1 Popis

AVR je rodina mikrokontrolérů od společnosti Atmel, které jsou hojně užívány jak amatéry, tak v profesionálních aplikacích. Atmel tyto mikrokontroléry dodává v rozmanitých pouzdrech a variantách. Dělí se do několika základních tříd pro všeobecné použití. Kromě nich je k dispozici množství různých typů pro konkrétní aplikace (automobilové, řízení baterií, radiačně odolné), kterými se dále zabývat nebudu. [1]

tinyAVR Je rodina nejjednoduších mikrokontrolérů z řady AVR. Mají nízký počet pinů a velmi malou spotřebu. Neobsahují takové množství periferií jako pokročilejší typy, ale svou jednoduchostí a nízkou cenou jistě nachází své využití. Většinu těchto mikrokontrolérů lze programovat napětím 3,3 V, ale některé vyžadují napájecí napětí 5 V, takže je složitější je programovat z FITkitu.

megaAVR Je rodina pokročilejších mikrokontrolérů z řady AVR. Mají vyšší počet pinů a často rozsáhlé periferie k dispozici, A/D převodníky, několik časovačů a UART moduly. Prakticky všechny podporují programování pomocí napětí 3,3 V, čímž jsou naprosto vhodné pro použití s FITkitem.

AVR XMEGA Jsou nejpokročilejší z mikrokontrolérů AVR. Mají 8/16bitová jádra. Vybrané typy obsahují dokonce podporu USB. Bohužel se nedodávají v DIP pouzdrech a tím pádem je jejich amatérské použití poměrně omezené. Způsob programování se také od předchozích typů změnil, proto jsem jejich podporu nezahrnul.

3.2 Programování mikrokontrolérů AVR

Tato část je kompilací několika zdrojů od společnosti Atmel [2] [4] [5] [3] a knihy [10].

3.2.1 Možnosti programování

Mikrokontroléry Atmel AVR obsahují několik možností programování:

- Sériové programování — toto programování je základní, používá pouze několik pinů a umožňuje programovat čip i v obvodu. Dále se budu zabývat tímto způsobem programování, protože jako jediné nevyžaduje 12 V napájení.

- Vysokonapěťové sériové programování — tento typ programování se využívá v zařízeních s menším počtem pinů a používá se v případě, že není možné využít základní sériovou verzi. Vyžaduje napětí 12 V přivedené na mikrokontrolér a proto je jeho použití složitější. Užitečné je hlavně v případě, že nastavení FUSE bytů zablokuje standardní sériové programování.

- Paralelní programování — tato metoda programování je dostupná u zařízení s větším počtem pinů a jeho použití je podobné jako u vysokonapěťového sériového programování. Také vyžaduje napětí 12 V a jeho použití je tudíž pro programování v obvodu nevhodné.

Sériové programování využívá protokol SPI a několik dalších vstupů, které je třeba nastavovat. [2]

3.2.2 Zapojení mikrokontroléru

Na obrázku 3.1 jsou zobrazeny vstupy a výstupy, které je třeba využívat při programování mikrokontroléru pomocí sériového programování.

- VCC, GND — napájení mikrokontroléru. Většina mikrokontrolérů dovede pracovat s napětím 1,8 V—5 V.

- SCK — připojení SPI, vstupní hodinový signál. SPI je synchronní přenos a tudíž je třeba řídícího hodinového signálu.

- MISO — připojení SPI, *Master-in, slave-out*, signál, po kterém mikrokontrolér odesílá svá data.

- MOSI — připojení SPI, *Master-out, slave-in*, signál, na kterém miktrokontrolér přijímá svá data.

- $\overline{\text{RESET}}$ — reset mikrokontroléru, signál je invertován, při programování je třeba být v resetu, což znamená, že na tomto vstupu musí být logická nula. Vzhledem k vlastnostem programátoru nestačí tento signál uzemnit.

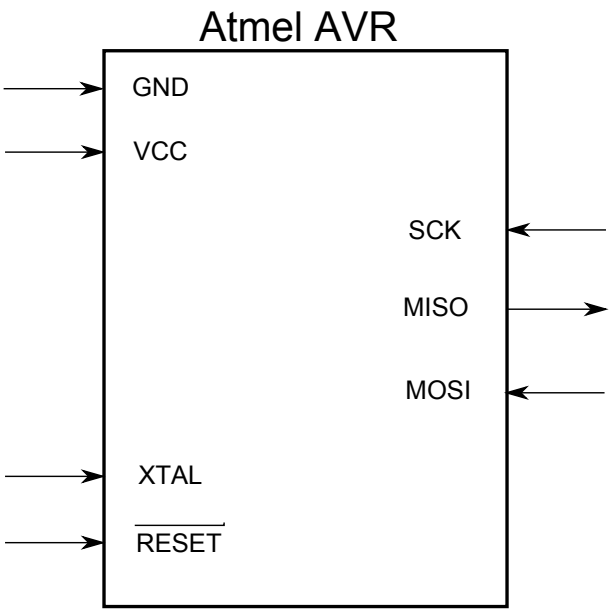
- XTAL — hodinový signál pro jádro mikrokontroléru. Je třeba pouze v případě, že je mikrokontrolér nastavený tak, aby využíval externí oscilátor.

Většina mikrokontrolérů AVR má poměrně velký rozsah povolených vstupních napětí 1,8 V — 5 V, některé menší typy ovšem striktně vyžadují programovací napětí 5 V[6]. Tím pádem je není jednoduché programovat z FPGA, které má maximálně 3,3 V výstupy. Pro většinu typů AVR tedy platí, že je možné je připojit přímo k FPGA. Ačkoliv by použití např. optočlenů mohlo být vhodné, tato vlastnost dokáže zapojení výrazně zjednodušit.

3.2.3 Průběhy signálů

Komunikace probíhá pomocí rozhraní SPI. Parametry SPI jsou standardní, první se odesílá nejvyšší významový bit. Data se nastavují na sestupné hraně hodinového signálu a čtou na vzestupné hraně. Hodinový signál musí být v logické nule v případě, že se zrovna nevyužívá. Programátor data odesílá na pinu MOSI a přijímá na pinu MISO. Zároveň musí generovat signál SCK.

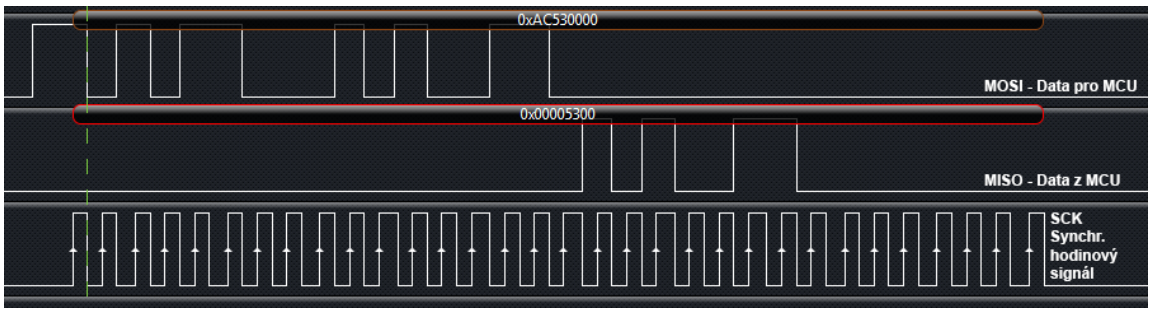
Rychlost komunikace musí splňovat podmínku v závislosti na taktu interního oscilátoru:



Obrázek 3.1: Zapojení mikrokontroléru pro sériové programování

- Pokud je frekvence oscilátoru vyšší nebo rovna 12 MHz, tak frekvence komunikace musí být nižší nebo rovna 1/6 frekvence oscilátoru.
- Pokud je frekvence oscilátoru nižší než 12 MHz, tak frekvence komunikace musí být nižší nebo rovna 1/4 frekvence oscilátoru.

Takt oscilátoru se odvíjí od nastavení FUSE bytů mikrokontroléru. Je možné při programování použít interní kalibrovaný oscilátor, schopný generovat frekvence 1 MHz, 2 MHz, 4 MHz a 8 MHz. Při správném nastavení mikrokontrolér dokáže využívat i externí zdroj hodinového signálu. Dále je třeba zohlednit fakt, že při použití externího zdroje hodinového signálu a napájecím napětí 3,3 V je maximální povolená hodnota frekvence 8 MHz[1]. Tím pádem je maximální frekvence komunikace 2 MHz při použití napájecího napětí 3,3 V.



Obrázek 3.2: Komunikace s mikrokontrolérem (vstup do programovacího módu)

[1]Prakticky mi bez problémů fungovaly i vyšší frekvence okolo 10 MHz, což ovšem pochopitelně nelze spolehlivě využít.

1.byte	**2.byte**	**3.byte**	**4.byte**	**Název**
Instrukce načítání do vyrovnávací paměti				
0xAC	0x53	0x00	0x00	Zahájení programování
0xAC	0x80	0x00	0x00	Smazání obsahu čipu
0xF0	0x00	0x00	0x00	Polling RDY/$\overline{\text{BUSY}}$
0x48	adresa	adresa	data	Zápis vyššího byte do Flash vyr. pam.
0x40	adresa	adresa	data	Zápis nižšího byte do Flash vyr. pam.
0xC1	0x00	adresa	data	Zápis byte do EEPROM vyr. pam.
Instrukce čtení z mikrokontroléru				
0x28	adresa	adresa	data(čtení)	Čtení vyššího byte slova z Flash paměti
0x20	adresa	adresa	data(čtení)	Čtení nižšího byte slova z Flash paměti
0xA0	adresa	adresa	data(čtení)	Čtení byte z EEPROM (datové) paměti
0x58	0x00	0x00	data(čtení)	Čtení zámků kódu
0x30	0x00	číslo	data(čtení)	Čtení záznamu typu mikrokontroléru
0x50	0x00	0x00	data(čtení)	Čtení FUSE bytu
0x58	0x08	0x00	data(čtení)	Čtení vyššího FUSE bytu
0x50	0x08	0x00	data(čtení)	Čtení rozšířeného FUSE bytu
0x38	0x00	číslo	data(čtení)	Čtení kalibrace oscilátoru
Instrukce zápisu do mikrokontroléru				
0x4C	adresa	adresa	0x00	Zápis stránky do programové (Flash) paměti
0xC0	adresa	adresa	data	Zápis dat do datové (EEPROM) paměti
0xC2	adresa	adresa	data	Zápis stránky do datové (EEPROM) paměti
0xAC	0xE0	0x00	data	Zápis zámků kódu
0xAC	0xA0	0x00	data	Zápis FUSE bytu
0xAC	0xA8	0x00	data	Zápis vyššího FUSE bytu
0xAC	0xA4	0x00	data	Zápis rozšířeného FUSE bytu

Tabulka 3.1: Seznam dostupných příkazů pro AVR mikrokontroléry

3.2.4 Zahájení programování

Při zahájení programování je třeba, aby byl mikrokontrolér v resetovaném stavu. Existuje několik způsobů jak toho dosáhnout. Pokud dokážeme zajistit, že napětí VCC bude připojeno dříve, než bude na pin $\overline{\text{RESET}}$ přivedena logická nula, tak je možno předpokládat, že mikrokontrolér je ve stavu resetu ihned po přivedení napájecího napětí.

Pokud nejsme schopni zajistit, že pin VCC bude připojen jako první, tak je třeba přivést na pin $\overline{\text{RESET}}$ logickou 1 po dobu alespoň dvou taktů oscilátoru, poté přepnout hodnotu $\overline{\text{RESET}}$ na logickou 0 a vyčkat minimálně 20 milisekund. Poté lze předpokládat, že mikrokontrolér je ve stavu resetu.

Ve chvíli, když je mikrokontrolér resetován, lze odeslat příkaz pro zahájení programování. Při správném postupu mikrokontrolér odešle zpět jako třetí byte hodnotu odeslanou v druhém byte (0x53), čímž lze ověřit, zda je zařízení připojeno a schopno komunikace.

3.2.5 Zápis a čtení programové paměti a EEPROM paměti

Programová paměť se zapisuje po stránkách, jejichž délka se liší dle konkrétního typu mikrokontroléru. Poté, co se zapíší všechny hodnoty, které do stránky spadají, do vyrovnávací paměti v mikrokontroléru, tak se příkazem zahájí zápis stránky do paměti Flash. Při zápisu je třeba zohledňovat to, že nižší byte slova je třeba zapisovat před vyšším bytem a nepřekročit hranice stránky. Hodnoty, které odpovídají 0xFF (což je implicitní hodnota všech míst v paměti u smazaného čipu) není třeba zapisovat, což může urychlit komunikaci.

Paměť EEPROM pro uživatelská data se u některých konkrétních typů mikrokontrolérů zapisuje také po stránkách, stejným způsobem jako paměť Flash. U některých typů se ovšem musí zapisovat po jednotlivých bytech, jelikož tyto neobsahují potřebnou paměť pro hromadný zápis.

Polling Po zápisu každé stránky, případně po zápisu každého bytu (u EEPROM), je třeba čekat určitou dobu stanovenou v datasheetu mikrokontroléru než zápis proběhne. Po tuto dobu není dovoleno odesílat žádné příkazy, které by se týkaly zápisu do jakékoliv paměti.

Druhou možností je využití aktivního dotazování (pollingu). Toto umožňuje pomocí opakovaných dotazů zjistit, zda už je zápis dokončen a tudíž není třeba čekat celou uvedenou dobu. Existují dvě metody pollingu, přičemž jejich použití se liší dle konkrétního typu mikrokontroléru:

- Polling pomocí čtení zapsaného bytu — některý z právě zapisovaných bytů se opakovaně čte. Dokud mikrokontrolér vrací jeho hodnotu jako 0xFF, tak je byte zapisován a pokud vrátí vlastní hodnotu bytu, pak je zápis dokončen. Což pochopitelně znamená, že tuto metodu nelze použít pro kontrolu bytu, jehož zapisovaná hodnota je právě 0xFF. Takové byty ale není nutné ani zapisovat.

- Polling pomocí speciálního příkazu RDY/$\overline{\text{BUSY}}$ — tento příkaz vrací jako poslední byte odpovědi hodnotu 0x01, pokud zápis probíhá, případně 0x01, pokud je dokončen. Využití je jednodušší, jelikož není třeba kontrolovat hodnotu zapisovaných dat.

Některé typy mikrokontrolérů AVR podporují první metodu a některé metodu druhou, je tedy pro využití pollingu třeba implementovat obě.

Verifikace Po zápisu dat je nutné tyto data ověřit, zda byl zápis úspěšný na všech zapsaných paměťových pozicích. U mikrokontrolérů AVR je toto jednoduché, stačí přečíst pozici v paměti a zkontrolovat s daty určenými k zápisu. Vzhledem k tomu, že AVR dokáží adresovat každý byte paměti zvlášť, je tento postup naprosto přímočarý.

Při zápisu dat po stránkách je možné ověřit zapsání stránky okamžitě po dokončení zápisu. Stejně tak při zápisu po jednotlivých bytech je možné provést kontrolu ihned po dokončení zápisu daného bytu.

3.2.6 FUSE byty

FUSE byty se programují pomocí zvláštních příkazů, jejichž přesné tvary se liší podle typu mikrokontroléru. Po zápisu každého bytu je třeba čekat určitou dobu, která je definovaná v datasheetu, pro dokončení programování. Po této době lze poté byte přečíst a porovnat se zamyšlenými daty, čímž dojde k verifikaci.

Některé části FUSE bytů se dají využít i k zablokování sériového programování. Ačkoliv sériové programování není možné zakázat při samotném sériovém programování, tak je možné přenastavit pin $\overline{\text{RESET}}$ tak, aby fungoval jako běžný vstupně-výstupní pin, což efektivně zablokuje sériové programování taktéž.

Mezi FUSE byty je možné započítat také Lock byte. Jde o jeden byte, který se používá stejně jako FUSE byty a umožňuje zamykat programovou paměť proti čtení a zápisu. Jeho použití se liší dle typu mikrokontroléru.

Kapitola 4

Mikrokontroléry Microchip PIC

4.1 Popis

PIC je rodina RISC mikrokontrolérů vycházejících z PIC1650 vytvořeného již neexistující společností General Instruments.

Mikrokontroléry PIC jsou oblíbené pro svou jednoduchost, dostupnost a poměrně příznivou pořizovací cenu. Firma Microchip Technology, která mikrokontroléry vyrábí, má v nabídce přibližně 290 různých typů, s rozdílnými parametry a rozšířeními.

PIC můžeme rozdělit do několika základních tříd [12][13]':

Základní zařízení (*Baseline core devices*) Jsou jednoduché typy mikrokontrolérů PIC. Používají 12bitové instrukce, obsahují poměrně málo paměti (protože mají jen 32 bytů adresového prostoru, což je ale u některých typů řešeno pomocí stránkování). Zástupci této třídy jsou hlavně PIC10, ale i některé PIC12 a PIC16.

Střední třída (*Mid-range core devices*) Jsou pokročilejší typy, které mají 14bitové instrukce. Oproti nižší třídě obsahují navíc několik nových instrukcí a hlavně rozšiřují počet adresovatelných registrů na 128. I zde se používá stránkování, pokud je paměti k dispozici více. Většina PIC12 a PIC16 spadá do této kategorie.

Rozšířená střední třída (*Enhanced mid-range core devices*) Hardwarově velice podobná střední třídě. Mají stejnou délku instrukcí, ale přibylo několik nových typů (hlavně jako optimalizace pro použití jazyka C). Patří sem několik typů z PIC16 a PIC12. Tyto mikrokontroléry jsou pro nás zajímavé, o čemž se zmiňuji dále.

Vyšší třída (*High-end core devices*) Nejvyšší třída mikrokontrolérů PIC. Využívají 16bitové instrukce, mají poměrně hodně paměti a podporují některé velmi užitečné periferie (např. USB 2). Do této kategorie spadají mikrokontroléry PIC17 a PIC18. Nutno říct, že PIC18 jsou značně rozšířenější a těší se výrazně vyšší oblibě. Programování těchto mikrokontrolérů se liší od všech předchozích (prakticky se odesílají instrukce přímo CPU, které poté zapisuje data do paměti). Při programování využívají velké množství různých příkazů, tudíž se jejich programováním dále zabývat nebudu.

4.2 Programování mikrokontrolérů PIC

Tato část je kompilací několika zdrojů od společnosti Microchip. [9][14][11]

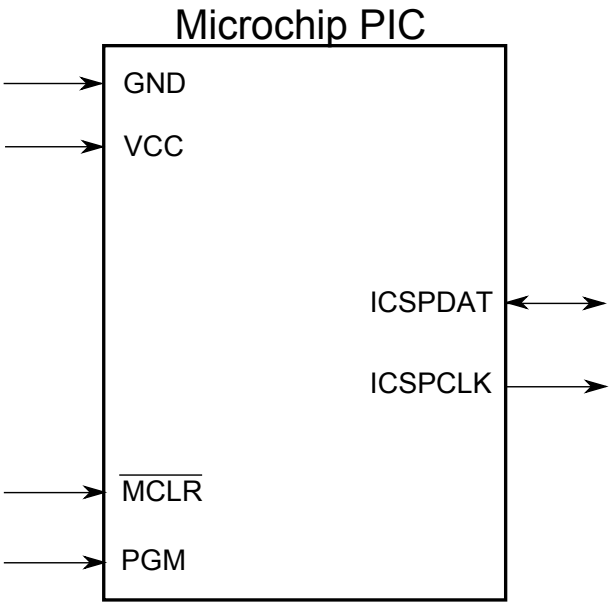
4.2.1 Možnosti programování

Možnosti programování jsou dvě, přičemž každá má svoje výhody a nevýhody:

Vysokonapěťové programování Vyžaduje přivedení 12 V na jeden z pinů mikrokontroléru. To přepne mikrokontrolér do programovacího módu, kde je možné mu zadávat další příkazy. Nevýhoda je jasná — je třeba 12 V zdroj (navíc schopný na tento pin připojit ještě zemi a ideálně 5 V). Další nevýhoda je, že při tomto typu programování je v drtivé většině případů třeba použít 5 V napájení, což může situaci trochu zkomplikovat. Výhoda je, že tento typ programování podporují všechny mikrokontroléry PIC.

Nízkonapěťové programování Do programovacího módu lze vstoupit i bez 12 V napětí. To se provádí dvěma různými způsoby (viz. dále). Navíc lze často využít pro napájení mikrokontroléru napětí 3,3 V. Bohužel tento typ programování podporují hlavně novější typy mikrokontrolérů PIC.

4.2.2 Zapojení mikrokontroléru



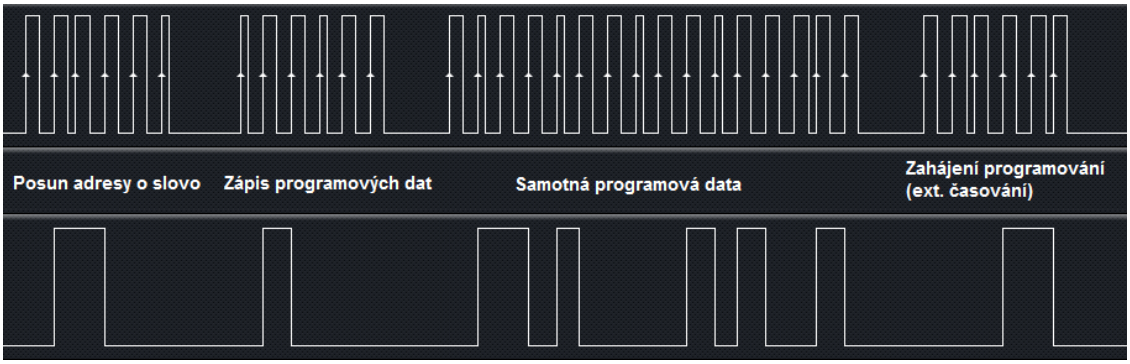
Obrázek 4.1: Připojení mikrokontroléru PIC

Na obrázku 4.1 jsou zobrazeny vstupy a výstupy, které je třeba zapojit při programování mikrokontroléru.

- ICSPCLK — synchronizační hodinový signál komunikace. Na rozdíl od Atmelu, kde jsou pravidla pro frekvenci komunikačního signálu rozmanitá a složitá, zde stačí dodržet maximální frekvenci 5 MHz.

- ICSPDAT — datový signál komunikace. Je obousměrný, tudíž je třeba využít třístavového vstupně/výstupního prvku na komunikačním zařízení.
- $\overline{\text{MCLR}}$ — resetovací pin, na tento pin je třeba přivádět 12 V při vysokonapěťovém programování a je také nutné ho nastavit do odpovídající hodnoty pro vstup do programovacího módu.
- PGM — pin, který spolu s pinem $\overline{\text{MCLR}}$ určuje, zda mikrokontrolér vstoupí do programovacího módu. Používá se pouze v případě nízkonapěťového programování. Některé nové typy mikrokontrolérů ho nevyužívají, je tedy třeba rozlišovat, zda je nutné ho připojit nebo ne.

4.2.3 Průběh signálů



Obrázek 4.2: Ukázka komunikace s mikrokontrolérem PIC

Programování mikrokontrolérů PIC využivá svůj vlastní protokol pro komunikaci mezi prográmatorem a programovaným čipem. Obdobně jako u Atmelu je třeba synchronizovat data s hodinovým signálem, ale tím podobnost s Atmelem končí. Je třeba zapisovat data na náběžné hraně, první se zapisuje nejnižší významový bit. Po každém odeslaném příkazu, případně zapsaných a přečtených datech je nutno čekat minimálně 1 µs.

Rozlišujeme 4 základní druhy průběhů:

- Příkaz — odeslání příkazu mikrokontroléru. Příkazy mají délku 6 bitů, je tedy třeba vygenerovat 6 pulzů hodinového signálu a na ně synchronizovat odesílaný příkaz.
- Data (odeslání) — odeslání dat. Délka dat může být 14 bitů, 12 bitů (podle typu jádra mikrokontroléru), případně 8 bitů pro datovou paměť. Způsob odesílání je ale stále stejný. Programátor musí vygenerovat 16 hodinových pulzů a na tyto synchronizovat data tak, že první a poslední puls se nevyužije a data se zarovnávají k nejnižšímu významovému bitu (odesílají se na začátku, protože nejnižší významový bit se odesílá první).
- Data (příjem) — příjem dat. Pokud je po některém příkazu třeba číst data, je nutné aby programátor přepnul pin ICSPDAT na vstup a poté vygeneroval 16 pulzů hodinového signálu. Pro vlastní data platí to stejné co pro jejich odesílání, první a poslední bity jsou nulové a data jsou zarovnána k nejnižšímu významovému bitu.
- MCHP — odeslání „MCHP“ v ASCII kódu je třeba provést při vstupu do programovacího módu některých typů mikrokontrolérů. Při odesílání se vygeneruje 33 hodinových

Příkaz	Název
0x00	Skok na adresu konfigurace
0x02	Načíst data pro zápis do programové/konfigurační paměti
0x04	Přečíst data z programové/konfigurační paměti mikrokontroléru
0x03	Načíst data do vyrovnávací paměti pro zápis do datové paměti
0x05	Přečíst data z datové paměti mikrokontroléru
0x06	Navýšit adresu interního ukazatele o jedno slovo
0x16	Reset hodnoty interního ukazatele
0x08	Zahájení interně časovaného programování
0x18	Zahájení externě časovaného programování
0x0A	Konec externě časovaného programování
0x09	Smazání celého obsahu programové paměti
0x0B	Smazání celého obsahu datové paměti
0x1F	Smazání celého obsahu čipu
0x11	Smazání aktuálního řádku paměti (interní časování)
0x08	Smazání aktuálního řádku paměti (externí časování)
0x0A (0x17)[1]	Ukončení externě časovaného programování

Tabulka 4.1: Seznam dostupných příkazů pro PIC mikrokontroléry, každý typ podporuje podmnožinu těchto příkazů

pulzů a na ně se synchronizují odesílaná data. Poslední puls se nevyužije (ale je nutné ho generovat).

4.2.4 Zahájení programování

Metoda vstupu do programovacího módu závisí na zvoleném módu a také na konkrétním typu mikrokontroléru, který používáme.

Vysokonapěťové programování Je třeba nastavit signály ISCPCLK a ICSPDAT do logické 0 a poté přivést napětí 12 V na pin $\overline{\text{MCLR}}$. Mikrokontrolér takto vstoupí do programovacího módu. Po vstupu je nutné čekat určitý čas, který se liší podle konkrétního typu.

Nízkonapěťové programování u starších zařízení (střední třída) Přivedeme napětí *VDD* na pin $\overline{\text{MCLR}}$ a poté přivedeme napětí *VDD* na pin PGM. Takto mikrokontrolér vstoupí do programovacího módu. Podobně jako u vysokonapěťového programování musíme čekat určitý čas, který se liší dle typu mikrokontroléru.

Nízkonapěťové programování u novějších zařízení (rozšířená střední třída) Zde je situace mírně komplikovanější. Tyto typy neobsahují pin PGM pro vstup do programovacího módu. Je tedy nutné nastavit pin $\overline{\text{MCLR}}$ na hodnotu *GND*. Poté odeslat po datovém kanále data „MCHP“ s tím, že data se odesílají klasicky — nejnižší významový bit první. Za hodnotou „MCHP“, která je dlouhá 32 bitů je třeba odeslat ještě 33. bit, s libovolnou hodnotou (jako stop bit).

4.2.5 Mazání obsahu čipu

Před zápisem dat je nutné smazat obsah paměti, do které potřebujeme zapisovat. Nejjednodušší metoda je pomocí příkazu pro smazání obsahu čipu, ať už při mazání programové a datové paměti zvlášť, nebo celého obsahu. U některých typů je ale tato možnost využitelná pouze při použití 5 V napájení. Pokud je využito napájení pomocí 3,3 V, tak je nutno využít mazání po řádcích, které je ovšem pomalejší. Časování při mazání může být jak interní, tak externí, použití se liší podle konkrétního typu mikrokontroléru.

4.2.6 Programování obsahu čipu

Metoda programování se výrazně liší od mikrokontrolérů Amtel, což je vidět i na tabulce 4.1. Mikrokontroléry PIC využívají adresový čítač, který je možné pomocí příkazu *Increment counter* posunout na další adresu v paměti. V těchto adresách je zapsána programová paměť i konfigurační paměť (obdoba FUSE bytů u Atmelu).

Po vstupu do programovacího módu je adresový čítač nastaven na hodnotu 0, přičemž neexistuje příkaz na jeho snížení. Pokud je potřeba posunout čítač zpět, je nutné opustit programovací mód a vrátit se do něj zpět, případně použít příkaz *Reset address*, pokud mikrokontrolér umožňuje tuto operaci.

Programování se provádí tak, že se data nejdříve zapíší do vyrovnávací paměti mikrokontroléru a poté se příkazem zahájí programování a data se zapíší do nevolatilní paměti. Při zápisu dat je možné u některých typů zapisovat před samotným programováním několik hodnot do vyrovnávací paměti. Po každém takovém zápisu se navýší adresový čítač a poté stačí provést jedno samotné programování paměti.

Programování paměti existují dva základni druhy, jejich použití se liší podle konkrétního typu zařzení a typu paměti. Někdy je možné si i vybrat. [14]

Externě časované programování Po zahájení programování musí programátor počkat stanovenou dobu a poté odeslat příkaz k ukončení programování. Po odeslání příkazu je nutno znovu počkat 300 µs než se vybijí programovací kondenzátory. Tento typ programování se využívá výrazně častěji.

Interně časované programování Po zahájení stačí počkat určitou dobu a poté je možné pokračovat dalším příkazem. Nevýhoda je, že tento typ programování je pomalejší než externí programování. Občas je nutné tento způsob použít.

Verifikace

Ověření zápisu je možno provést kdykoliv po ukončení programování. V případě, že se data programují pouze po jedné hodnotě naráz, je možné provádět verifikaci okamžitě po zápisu. Způsob je takový, že se před odesláním příkazu k navýšení čítače data ještě přečtou. Pokud se data zapisují po větších blocích, není možné data verifikovat okamžitě po zápisu (z důvodu nemožnosti snížení čítače) a je tudíž nutno verifikovat až všechna data naráz po dokončení zápisu celého čipu.

4.2.7 Konfigurační data

Konfigurační data se nachází na některých adresách, které jsou specifikovány v manuálu ke konkrétním mikrokontrolérům. U některých typů je možno využít příkaz *Load configuration*,

který posune programový čítač na adresu konfigurace. Pokud toto není umožněno, tak je třeba posouvat interním ukazatelem, dokud se neposune na potřebnou adresu.

Podobně jako u Atmelu je možno nastavením konfigurace zakázat nízkonapěťové programování. U mikrokontrolérů PIC ovšem nelze toto provést přímo z nízkonapěťového programování žádným způsobem, na rozdíl od Atmelů.

Kapitola 5

Návrh a implementace

5.1 Požadavky

Požadavky této části vycházejí z předchozích kapitol. Vlastním cílem je pochopitelně schopnost programovat mikrokontroléry, ale aby toho mohlo být dosaženo optimálně, je třeba zohlednit některé další nutné vlastnosti. Při tvorbě jsem vycházel z následujících požadavků:

- Využívat FITkit, který je dostupný pro všechny studenty FIT a tudíž je ideální platformou pro studium programování mikrokontrolérů.
- Schopnost komunikovat s mikrokontroléry Atmel AVR a Microchip PIC, kvůli jejich dostupnosti.
- Schopnost generovat hodinový signál pro komunikaci s mikrokontroléry ve frekvenci, která musí být nastavitelná.
- Komunikace s PC musí být dostatečně rychlá, pro dosažení maximální rychlosti samotného programování.
- Komunikace musí být co nejplynulejší, pro dosažení maximální rychlosti. Pokud není dostatečně plynulá, je třeba implementovat vyrovnávací paměť, která tento problém vyřeší.
- Přípojení musí využívat minimum externích komponent, aby použití bylo z hlediska zapojení co možná nejjednodušší.
- Zařízení musí umět programovat data, vyčítat data, programovat a číst konfigurační data a provádět všechny ostatní operace, které jsou pro úspěšné použití mikrokontroléru nutné.

5.2 Cílová platforma

5.2.1 Zařízení

Pro co nejlepší splnění předchozích požadavků jsem zvolil použití FPGA na FITkitu. Vzhledem k vlastnostem FPGA obecně lze dosáhnout splnění několika požadavků s poměrně malým úsilím. Použití mikrokontroléru je pro řešení této práce nadbytečné, pouze by přeposílal data z PC do FPGA a zbytečně komplikoval celý systém. Na FITkitu mikrokontrolér využiji

pouze pro generování hodinového signálu pro řízeni FPGA. Cílová platforma je tedy *Xilinx Spartan 3*, konkrétně *XC3S50*. FPGA *XC3S50* je hradlové pole od společnosti Xilinx, které obsahuje 1728 logických bloků. Každý logický blok je tvořen nastavitelnou 4-vstupou LUT (*Look-up-table*), která umožňuje v hardware vytvořit libovolnou 4-vstupou funkci a D klopný obvod, který lze využít jako registr. Výhoda těchto LUT je, že je možné je využít i jako 16bitový posuvný registr, nebo 16bitovou RAM paměť (protože jsou tvořeny statickou RAM pamětí). Toto FPGA dále obsahuje 9 kB blokové statické RAM paměti, která jde zapojit i jako dvouvstupová RAM paměť. [17]

Projekt je vytvořen takovým způsobem, aby ho bylo možné použít na libovolném FPGA s dostatečnou kapacitou, například některé od firem Actel nebo Altera.

5.2.2 Komunikace

Vzhledem k tomu, že FITkit je běžně připojen k PC přes rozhraní USB a při zohlednění rozšířenosti USB je volba komunikačního kanálu mezi PC a FPGA zřejmá. FITkit obsahuje USB převodník od společnosti FTDI, který je schopen převádět data z USB na několik různých forem komunikace, např. UART nebo paralelní FIFO. Právě UART jsem zvolil jako metodu pro komunikaci a to z důvodů jednoduchosti jeho použití, rozšířenosti (v případě, že by bylo v jiném zapojení nutné využívat jiný typ převodníku) a velkému rozsahu možných přenosových rychlostí.

UART

UART (*Universal asynchronous receiver/transmitter*) je elektronické zařízení, které je schopné měnit paralelní komunikaci na sériovou. [15] Od toho se odvíjí používaný protokol. Tento protokol je definován parametry přenosu:

- Rychlost komunikace — počet odeslaných bitů za sekundu. Do této hodnoty se počítají i start a stop bity. Většinou se používá násobek 300, např. 9600, 57600, 115200, 921600, atd.
- Počet stop bitů — kolik stop bitů se umisťuje na konec komunikace. Vzhledem k tomu, že stop bity mají stejnou hodnotu jako nevyužívaná linka, je tato informace využívána pouze vysílací částí. Běžně používané hodnoty jsou 1, 1, 5 a 2.
- Paritní bit — zda se má na konec komunikace přidávat paritní bit. Parita může být sudá nebo lichá. Tento bit má logickou hodnotu 1 v případě, že počet jedniček v odesílaném bytu je sudý/lichý. Pomocí paritního bitu lze zjistit, že jsou data nesprávná. Ale není možné zjistit, který bit je špatně.
- Řízení toku — zařízení mezi sebou komunikují a pomocí informací druhému zařízení dokáží zastavit/povolit přenos. Metody řízení toku jsou tři (kromě možnosti, kdy je řízení toku vypnuté):
  - XON/XOFF — nebo také softwarové řízení toku. Zařízení může odeslat příkaz XON (0x11), pokud chce povolit odesílání a příkaz XOFF (0x13), pokud chce odesílání zakázat. Toto má zřejmou nevýhodu — není možné použít tyto dvě hodnoty k ničemu jinému.
  - RTS/CTS — nebo také hardwarové řízení toku. Povolení vysílání určují hodnoty signálů RTS a CTS, každý jedním směrem. Nastavení hodnoty do logické 1 zakáže odesílání. Nevýhoda je, že jsou třeba další dva vodiče.

– DTR/DSR — stejné jako RTS/CTS, pouze na jiných pinech.

FTDI čip

Tento čip od společnosti FTDI má několik vlastností, které je třeba zohlednit. Funguje na bázi USB packetů, tudíž přenos není plynulý jako u integrovaného UART řadiče, ale pracuje po blocích. Délka bloku a pauza mezi bloky se liší podle nastavené rychlosti komunikace. Čip podporuje všechny parametry komunikace, které může UART nabídnout, včetně paritního bitu, řízení toku, nastavení počtu stop bitů a podobně. Jediné co nepodporuje je vyvolání přerušení procesoru, takže je třeba se dotazovat na přijatá data.

Parametry komunikace

Rychlost komunikace je nutno zvolit tak, aby při programování čipů nedocházelo ke zbytečným prodlevám. Mikrokontrolér s nejvyššími nároky na přenosovou rychlost, který tento prográmator dokáže programovat, je *PIC16F1518*. Tento mikrokontrolér vyžaduje 64 bytů dat každých 2,5 ms, což znamená požadavek na přenosovou rychlost 25600 bytů za sekundu. Vzhledem k tomu, že FTDI čip odesílá data v blocích a s pauzou přibližně 10 ms mezi bloky, je třeba využít rychlost vyšší. Při použití běžné rychlosti 921600 bit/s je efektivní rychlost třetinová (5 ms odesílání dat a 10 ms pauza), což při přepočtu na byty za sekundu znamená 30720 bytů/s. Tato rychlost je tedy dostačující. Neplynulost komunikace je ale třeba řešit použitím vyrovnávací paměti, aby programování bylo plynulé.

Využití řízení toku je velmi užitečné, protože kapacita paměti na FPGA je poměrně nízká a potřebné objemy dat jsou poměrně vysoké. Softwarové řízení toku by bylo naprosto nevhodné, protože by znemožnilo použití některých hodnot pro samotné programování. Vzhledem k tomu, že FTDI čip je již připojen k FPGA pomocí vodičů umožňujících použití RTS/CTS, je volba jasná.

Parita není nijak užitečná. Při zapojení, kdy jsou obě dvě komponenty na jedné desce tištěných spojů, je pravděpodobnost selhání velmi nízká. Další odesílaný bit by pouze zpomaloval přenos.

Stop bit stačí jeden, důvod je podobný jako u parity. Pro moderní zařízení s přesnými oscilátory není třeba využívat větší počet stop bitů.

Parametry komunikace jsou tedy následující:

- Rychlost komunikace: 921600 bit/s
- Řízení toku: RTS/CTS
- Stop bity: 1
- Parita: Žádná

5.2.3 Takt hodinového signálu

Dále bylo třeba zvolit vhodnou frekvenci hodinového signálu pro řízení FPGA. Pro ni je třeba přihlédnout k následujícím požadavkům:

- Frekvence musí být dělitelná 921600, pro použití pro generování UART signálů.
- Z frekvence musí jít dělením sudým číslem (pro jednoduchost) vytvořit frekvence co možná nejbližší 8 MHz, jako externí hodinový signál pro zařízení Atmel.

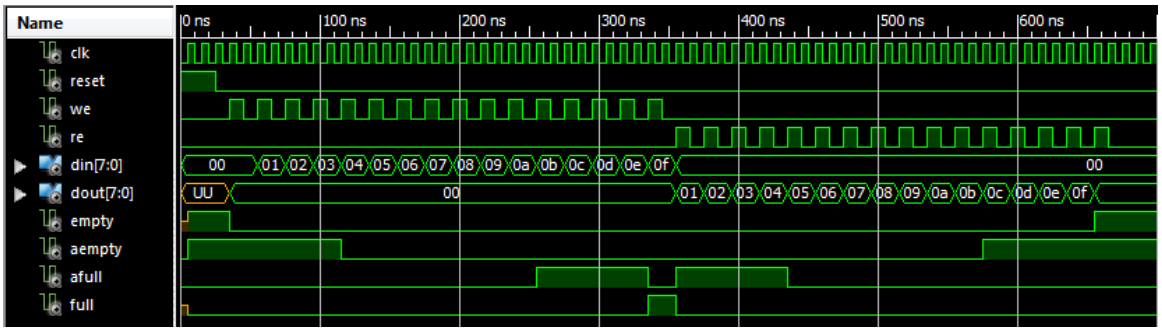
- Z frekvence musí jít dělením sudým číslem vytvořit frekvence co možná nejbližší 5 MHz, jako komunikační signál pro mikrokontroléry PIC.
- Z frekvence musí jít dělením sudým číslem vytvořit frekvence co možná nejbližší 2 MHz, jako komunikační signál pro mikrokontroléry Atmel s externím oscilátorem nebo 8 MHz interním oscilátorem.
- Z frekvence musí jít dělením sudým číslem vytvořit frekvence co možná nejbližší 1 MHz, 0,5 MHz a 0,25 MHz jako komunikační signál pro mikrokontroléry Atmel s interním oscilátorem o frekvenci 4 MHz, 2 MHz a 1 MHz.
- Frekvence musí jít vytvořit v DCM modulu v FPGA ze vstupní frekvence 7,3728 MHz.

V příloze C je tabulka, která obsahuje seznam frekvencí, které splňují první parametr a zohledňuje míru splnění ostatních parametrů. Z této tabulky jasně vyplývá, že nejvhodnější dostupná frekvence je 47,9232 MHz.

5.3 Popis jednotlivých modulů

5.3.1 Modul FIFO

Vzhledem k vlastnostem FTDI čipu je nutné využívat na výstupu UART modulu frontu, která bude zadržovat přečtená data, dokud je nevyčtou ostatní moduly. Jde o základní implementaci synchronní fronty, která nastavuje signály *EMPTY* (pokud je fronta prázdná), *FULL* (pokud je fronta plná), *AEMPTY* (pokud je fronta skoro prázdná) a *AFULL* (pokud je fronta skoro plná). Základní princip je naznačený v následujícím obrázku:



Obrázek 5.1: Funkce a chování FIFO

Signály *AEMPTY* a *AFULL* jsou nutné pro nastavování hardwarového řízení toku. Vzhledem k tomu, že čip FTDI odesílá ještě maximálně 3 byty dat po zákazu odesílání, je nutné mít vyrovnávací paměť schopnou pojmout další alespoň 4 byty. [7] Z toho důvodu je zvolena velikost fronty 16 bytů. Přičemž signál *AEMPTY* je nastaven v případě, že zaplnění klesne na 4 byty nebo méně. Signál *AFULL* je nastaven v případě, že zaplnění vzroste na 12 bytů nebo více.

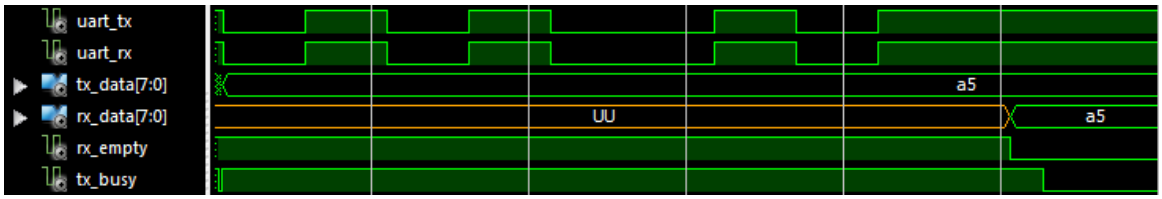
Paměť je schopna zároveň data číst a zapisovat. V případě zápisu do plné paměti nebo vyčtení prázdné paměti dojde k přetečení. Toto není nijak ošetřeno, nicméně vzhledem k návrhu designu a chování ostatních komponent prakticky nemůže k takové situaci dojít. Tato chyba by se projevila nesprávným chováním zařízení.

FIFO je vytvořeno tak, že první slovo je automaticky dosazováno na výstup, takže je platné už v momentě vyčítání. Toto zapojení umožňuje číst data ve stavovém stroji s minimem spotřebovaných stavů. Zapojení je řešeno použitím asynchronního čtení z paměti, což znemožňuje využit blokovou RAM paměť v FPGA. Syntetizátor tudíž využívá distribuovanou RAM paměť, což je vlastně zapojení LUT jako 16bitových registrů. Těch je tedy potřeba 8. Řízení takovéhoto vyčítání z blokové RAM paměti by zabíralo více než 8 LUT, takže nejde o neúsporu.

5.3.2 Modul UART

UART modul je vytvořen tak, aby zabíral co možná nejmenší prostor na FPGA. Je tvořen ze dvou základních částí. Jedna část odesílá data přes komunikační kanál a druhá je přijímá. Modul je schopen nastavovat signály CTS a řídit se signálem RTS.

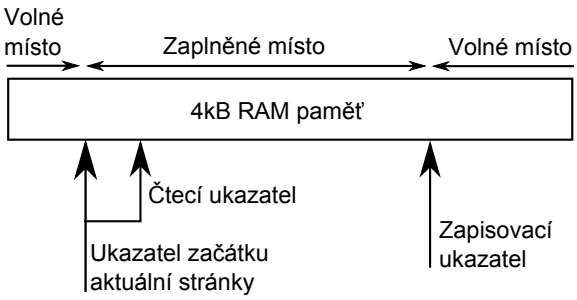
Po přijetí jsou přečtená data uložena do vyrovnávací paměti, odkud je čtou ostatní moduly. Pro odesílání žádná vyrovnávací paměť není implementována, vzhledem k průběhu operací by byla v naprosté většině případů zbytečná. Modul nastavuje signál CTS pomocí RS klopného obvodu. V případě nastavení signálu *AFULL* se odesílání dat z FTDI čipu zakáže a v případě nastavení signálu *AEMPTY* se zase povolí. Přetečení FIFO je tímto zamezeno, podtečení je třeba hlídat v modulech, které data využívají. Chování UART modulu je patrné v následujícím obrázku.



Obrázek 5.2: Chování modulu UART — loopback (vstup je připojen na výstup)

5.3.3 Modul kruhové vyrovnávací paměti

Při programování dat je třeba využít vyrovnávací paměť, která má dva důležité úkoly. Zaprvé vyrovnává nepravidelnosti v chování odesílání dat z FTDI čipu a zadruhé ukládá právě čtená data, aby bylo možné snadno provádět verifikaci.



Obrázek 5.3: Modul kruhové paměti

Modul se chová jako inteligentní FIFO, ale s několika zásadními rozdíly. Pracuje po stránkách nastavitelné délky. Maximálně je schopen pracovat s 1024 stránkami délky 256

bytů. Toto umožňuje programovat mikrokontroléry až do kapacity 128 kB (např. *ATMega128*). Využívá 4 kB blokové RAM paměti, do které automaticky načítá data z UART modulu. V rámci jedné stránky je možnost číst libovolná data, což je používáno při verifikaci zapsaných dat. Po postoupení na další stránku tento modul případně přepíše původní stránku novými daty. Interně využívá čtecí ukazatel, který se počítá jako součet ukazatele začátku stránky s nastaveným offsetem a zapisovací ukazatel, který se posouvá automaticky při čtení dat z UART modulu.

5.3.4 Hlavní modul programátoru

Tento modul propojuje všechny ostatní moduly a hlavně řídí chování celého programátoru. Dekóduje příchozí příkazy a obsahuje několik pomocných prvků. Přijatý příkaz se ihned po přijetí odesílá zpět, což usnadňuje řazení odpovědí. Poté je ve stavovém stroji zpracován a proveden. Pokud je příkaz určen pro některý jiný programátor, je tento programátor spuštěn a příkaz je mu předán. Po ukončení provádění příkazu konkrétním programátorem má programátor možnost vyvolat odeslání zprávy o úspěchu, případně neúspěchu, což zároveň odešle adresu chyby.

Tento modul ukládá informace, které jsou společné pro všechny komponenty, konkrétně délku stránky, počet stránek a globální adresu (hodnotu pro různé využití programátory). Zároveň je možné tyto hodnoty nastavovat příkazy tomuto modulu.

Hlavní modul také obsahuje 17bitový jednosměrný čítač pro použití ostatními komponentami, modul schopný provádět čekání s přesností 0,5 ms pro použití ostatními komponentami a generátor pulzů s nastavitelnou frekvencí, který se využívá pro generování komunikačního hodinového signálu. Tento generátor vytváří střídající se pulzy na dvou signálech (*UP* a *DOWN*). Puls na signálu *UP* určuje, kdy má přijít náběžná hrana komunikačních dat a puls na signálu *DOWN* určuje pozici sestupné hrany.

Dále tento modul ukládá adresu selhání programování od ostatních komponent.

5.3.5 Modul programátoru Atmel

Tento modul je využíván pro programování mikrokontrolérů typu Atmel. Obsahuje několik stavových strojů a dalších modulů schopných provádět kompletní programování.

Komunikační modul

Generuje komunikační hodinový signál a odesílá data, která dostane zadána. Během odesílání také do registru uloží data, která během komunikace odesílal mikrokontrolér. Tímto způsobem implementuje protokol SPI pomocí posuvného registru. Pro komunikaci s mikrokontroléry Atmel je třeba zajistit, že hodinový signál dodržuje synchronizaci s použitým oscilátorem. Tudíž je nevhodné, aby se komunikační hodiny posouvaly vůči oscilátoru. Toho je dosaženo využitím signálů *CK-UP* a *CK-DOWN*. Tato metoda ale za určitých okolností vytváří mírné zpomalení, jelikož je před začátkem odesílání třeba čekat na sestupnou hranu pro nastavení prvního bitu dat.

Z tohoto důvodu komunikační modul generuje signál o dokončení ještě před samotným dokončením. Pokud je poté během odesílání posledního bitu vytvořen další požadavek na odesílání dat. Tato data se odešlou okamžitě po dokončení původního odesílání. To umožňuje výrazně zrychlit programování až na téměř optimální míru.

Hlavní stavový stroj Atmel

Přijímá příkazy od hlavního modulu a stará se o jejich zpracování. Přímo využívá komunikační modul pro komunikaci s mikrokontrolérem. Dokáže odeslat libovolný příkaz mikrokontroléru (např. nastavení FUSE, smazání obsahu čipu) a odpověď odeslat zpět. Dokáže zahájit programování, vyčítání a vstoupit do programovacího módu, vše pomocí stavových strojů k tomu určených.

Dále je schopen nastavit čekací doby pro ostatní prvky (jelikož se liší dle typu mikrokontroléru) a konfiguraci konkrétního čipu, určující zda se EEPROM paměť programuje po stránkách a typ pollingu.

Stavový stroj pro vstup do programovacího módu

Po spuštění tohoto stroje se pokusí přejít do programovacího módu mikrokontroléru. Provede automaticky celou operaci za použití komunikačního modulu. Na konci ohlásí úspěch nebo neúspěch, podle toho zda mikrokontrolér odpověděl nebo ne.

Programovací a vyčítací modul

Tento modul se využije pro programování a vyčítání obsahu programové a datové paměti mikrokontroléru. Před naprogramováním každé stránky odešle přes sériovou linku informaci o postupu, která se poté využívá pro zobrazení průběhu. Dělí se na tři části:

Část pro stránkové programování Zapisuje data do programové i datové paměti a to po stránkách libovolné délky. Využívá kruhovou vyrovnávací paměť, jejíž parametry se použijí i pro programování čipu (délka stránky a jejich počet). Dokáže využít oba druhy pollingu pro urychlení programování. Na konci programování informuje o úspěchu nebo neúspěchu programování.

Část pro nestránkové programování Dokáže zapisovat data pouze do datové paměti. Také využívá kruhovou paměť, ale protože programování neprobíhá po stránkách, tak jsou parametry nastavení stránek libovolné. Dokáže využívat obě metody pollingu. Na konci programování odesílá informace o úspěchu, případně neúspěchu programování.

Část pro vyčítání Dokáže vyčítat programovou i datovou paměť a vyčtená data rovnou odesílá pomocí sériové linky. Využívá globální adresu a 17bitový čítač ze společného bloku.

5.3.6 Modul programátoru PIC

Tento modul se využívá k programování mikrokontrolérů typu PIC. Funguje podobně jako varianta pro Atmel. Obsahuje komunikační část a stavové stroje používané pro zpracovávání dat a také samotné programování.

Komunikační modul

Generuje komunikační hodinový signál a odesílá, popřípadě přijímá, data dle zadaných parametrů. Modul má čtyři metody použití, které jsou popsány v kapitole 4.2.3. Po každém dokončení odesílání nebo přijímání, provede ještě automaticky čekání po dobu 1 µs, jak je mikrokontroléry PIC vyžadováno. Stejně jako komunikační modul Atmel tento modul

využívá signálů *CK-UP* a *CK-DOWN* pro generování komunikačního hodinového signálu. Ačkoliv jejich použití zde není tak zásadní jako v případě Atmelu, je vhodné z hlediska sdílení zdrojů.

Hlavní stavový stroj PIC

Tento stavový stroj dokáže zahájit programování. Toto se neděje pomocí zvláštního stavového stroje, jako v případě Atmelu, ale přímo v hlavním stavovém stroji. Dále dokáže nastavit konfiguraci PIC, posouvat adresovým ukazatelem v mikrokontroléru, smazat obsah čipu a zahájit programování nebo vyčítání čipu. Zpět odesílá informace o úspěchu.[1]

Programovací a vyčítací modul PIC

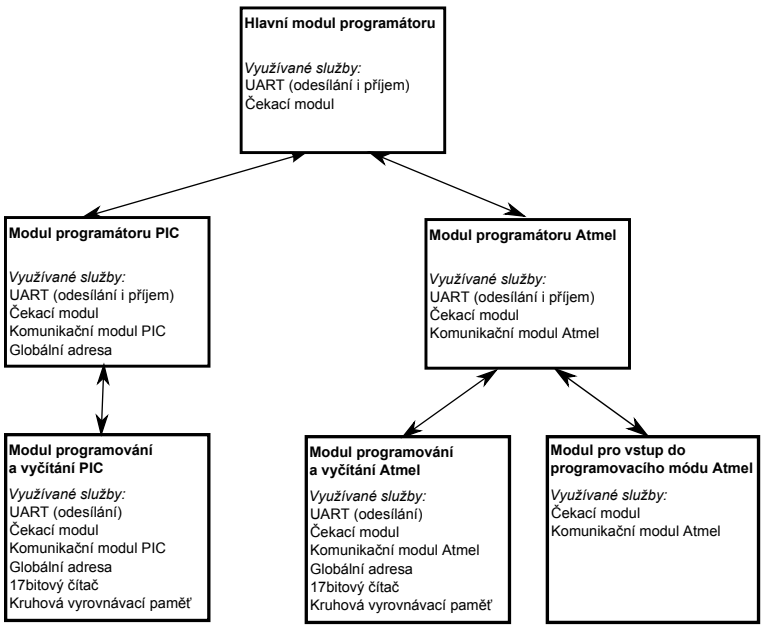
Tento modul je určen k programování a vyčítání samotných dat. Zohledňuje nastavenou konfiguraci při programování, což je nutné vzhledem k velkému počtu různých způsobů programování pro různé typy mikrokontrolérů. Dokáže provádět i mazání po řádcích, interní i externí programování, verifikaci přímo při zápisu, všechna potřebná čekání a ostatní náležitosti. To činí stavový stroj poměrně komplikovaný. Využívá globální 17bitový čítač.

Vyčítací modul využívá 17bitový čítač i globální adresu, která v tomto případě určuje počet slov, která se mají vyčíst. Vzhledem k tomu, že každé slovo má 16 bitů, tak tento modul odesílá 2 byty na každé přečtené slovo. Tato odesílaná data jsou formátována tak, aby bylo možné je rovnou zapisovat do *.hex* souboru.

5.3.7 Předávání řízení

Moduly mohou využívat společných služeb kdykoliv, kdy potřebují. Aby bylo zajištěno, že nedojde k využití jednoho prvku dvěma moduly naráz, tak si hlavní moduly mezi sebou předávají řízení. Toto je provedeno hlavně z důvodu přehlednosti kódu, principielně by všechny moduly mohly být zahrnuty v jednom stavovém stroji. Tedy vždy je aktivní pouze jeden modul, přičemž ostatní čekají na jeho ukončení. Způsob předávání řízení je zobrazen na obrázku níže.

[1] Žádná z uvedených operací nemůže selhat, protože na ně mikrokontrolér nijak neodpovídá, není tedy nutné informovat o neúspěchu.



Obrázek 5.4: Diagram předávání řízení

Kapitola 6

Návrh a implementace ovládací aplikace

6.1 Požadavky

Při tvorbě ovládací aplikace a výběru vhodného prostředí jsem vycházel z několika požadavků:

- Ovládací aplikace by měla být multiplatformní.
- Aplikace by měla být uživatelsky příjemná na použití (konzolová aplikace by byla nevhodná). Měla by být přehledná a informativní.
- Aplikace musí umět pracovat s programátorem přes sériový port počítače a být schopná programátor řídit a zpracovávat odpovědi.
- Aplikace musí umět programovat a vyčítat programovou paměť mikrokontroléru, datovou paměť mikrokontroléru a konfigurační data.
- Aplikace musí před koncovým uživatelem skrýt operace, jejichž zobrazení by uživatel považoval za nadbytečné.

6.2 Prostředí a implementace

Ovládací aplikace je vytvořena v jazyce Python, což příjemným způsobem řeší problémy s více platformami. Konkrétně jsem využil jazyk *Python 2.7*.

Jako grafické rozhraní jsem zvolil knihovnu *Qt*, konkrétně *pyQt*. Mezi jeho výhody patří poměrně jednoduchá implementace (v porovnání s některými jinými prostředími) a hlavně obrovská výhoda přirozeného vzhledu dle platformy. Dá se předpokládat, že uživatel například Windows 7 je zvyklý na jiný vzhled oken, než uživatel Gnome, což *Qt* řeší automaticky.

Pro sériovou komunikaci jsem zvolil knihovnu *pySerial*, která umožňuje provést připojení k sériovému portu, nastavit parametry připojení a odesílat a přijímat data. Výhodou je opět multiplatformnost, knihovna je schopna provádět ještě další užitečné operace, například vytvořit seznam všech dostupných portů v počítači.

Aplikace dále využívá několik modulů, které jsou ale v Pythonu vestavěny, např. *threading*, *os*, *time* a *re*.

6.3 Implementace

6.3.1 Podpora více jazyků

Aplikace podporuje více jazyků. Implementována je čeština a angličtina. Pro přidání dalšího jazyka není třeba zasahovat do zdrojového kódu, stačí pouze přidat další jazyk k ostatním souborům ve složce *lang/*. Program sám prohledá potřebné soubory a načte jeden z nich, podle toho, který jazyk je nastaven. Pro změnu jazyka je třeba program restartovat.

Použití je provedeno pomocí asociativního pole, které obsahuje překlady textů pro daný jazyk. Ve zdrojovém kódu jsou tedy pouze textové odkazy na řádky v souborech překladů, přičemž modul pro zpracování jazyků provede potřebný převod.

6.3.2 Uživatelské rozhraní

Uživatelské rozhraní je vytvořeno tak, aby jeho použití bylo co nejvíce intuitivní. Hlavní část vyplňuje modul se třemi záložkami, mezi kterými lze přepínat:

- Flash — zobrazuje obsah programové paměti. Toto ovšem není aktuální paměť mikrokontroléru, ale data načtená v paměti počítače. Obsah je zobrazen jako hexadecimální kódy po 16 bytech na jeden řádek. Dále je zde možnost tato data zapsat do paměti mikrokontroléru, případně vyčíst data z mikrokontroléru a zobrazit je v tomto okně.
- EEPROM — zobrazuje obsah datové paměti. Tato záložka nemusí být vždy přístupná. Pokud konkrétní mikrokontrolér neobsahuje datovou paměť, záložka nepůjde otevřít. Je velice podobná jako zobrazení programové paměti a stejně tak je možné tuto paměť zapsat a případně vyčíst obsah zařízení.
- Fuse/konfigurace — zobrazuje nastavení konfiguračních parametrů a umožňuje je změnit. Pochopitelně lze také tyto parametry zapsat a vyčíst.

Na pravou stranu byl umístěn ovládací blok, ve kterém je možné nastavit komunikační frekvenci, typ a rodinu mikrokontroléru a provést další operace, jako smazání obsahu čipu a hromadná programování a vyčítání.

Ve spodní části je informační okno, do něhož různé moduly vpisují informace u úspěchu a neúspěchu prováděných operací. Okno umožňuje nastavit barvu textu, který tak bude přehlednější. Pod informačním oknem se nachází progress bar, který zobrazuje postup právě prováděné operace. U něj je také možné změnit barvu. Implicitně je progress bar zelený, červenou barvu používám, pokud dojde k problému.

6.3.3 Vícevláknové zpracování (threading)

Aplikace používá několik vláken, která vytváří pomocí modulu *threading*. Vlákna jsou využita, aby uživatel měl pocit naprosté plynulosti aplikace a aby nedocházelo k různým zádrhům a nutnosti čekání na dokončení operace. Aplikace sestává ze tří vláken:

- Vlákno GUI — toto je původní vlákno, které je vytvořeno spuštěním aplikace. Využívá se ke zobrazování dat pro uživatele a běží v něm celé grafické uživatelské rozhraní a celá interní logika, např. načítání informací o mikrokontrolérech, výpočet hodnot FUSE bytů a různé další operace, které přímo nekomunikují s mikrokontrolérem.

- Vlákno sériové komunikace — toto vlákno komunikuje se sériovým portem. Vzhledem k tomu, že nelze využít přerušení, je jedinou možností opakované dotazování (polling). Toto vlákno tedy přijímá a odesílá data ze sériového portu pomocí dvou front (vstupní a výstupní).

- Vlákno komunikace s mikrokontrolérem — toto vlákno využívá vlákno sériové komunikace pro komunikace s mikrokontrolérem. Je odděleno od vlákna GUI, protože některé operace trvají poměrně dlouho (desítky sekund) a uživatel by mohl mít pocit, že aplikace havarovala, pokud by např. nešlo klikat na tlačítka. Toto vlákno využívá systém řetězení příkazů pro dosažení maximální rychlosti komunikace.

Komunikace mezi vlákny

Vlákna mezi sebou komunikují několika způsoby. všechny tyto způsoby jsou chráněny semafory, aby nedošlo k problémům způsobeným společným přístupem.

Vlákno sériové komunikace obsahuje dvě fronty, jednu pro čtení dat a druhou pro zápis. Ostatní moduly tedy mohou vkládat data do zapisovací fronty a číst data z čtecí fronty. Tato operace je chráněna semafory, pro každou frontu zvlášť. To zaručuje, že dva moduly nepřečtou stejná data a nezapíší data chybně. Metody užité pro čtení umožňují číst s čekáním, bez čekání, jeden byte i více, pro maximální flexibilitu ostatních modulů.

Vlákno GUI ukládá všechny potřebné údaje o mikrokontroléru a umožňuje přístup k nim ostatním modulům. Tyto informace jsou uloženy ve slovníku (asociativním poli). Pro přístup k těmto datům je tedy nutné znát pouze název klíče. Přístup je hlídán semaforem, který je stejný pro čtení a zápis.

Vlákno GUI dále obsahuje frontu, do které lze uložit pokyny k překreslení části uživatelského rozhraní, pokyn k výpisu informací do stavového okna, případně pokyn pro nastavení hodnoty a barvy progress baru. Tyto hodnoty jsou vybírány pomocí časovače přímo z knihovny Qt a zpracovávány.

Vlákno komunikace s mikrokontrolérem přijímá příkazy z grafického rozhraní. Tyto obsahují pouze příkaz k provedení, všechny potřebné údaje si toto vlákno získá samo dotazy na asociativní pole. Chování je kontrolováno semaforem, takže celé vlákno čeká, dokud mu není zadán příkaz. Pokud je zrovna nějaký příkaz prováděn a je zadán další, tak se neprovede, ale vypíše se chybová hláška do stavového okna.

6.3.4 Řetězení příkazů

Vzhledem k tomu, že při komunikaci s mikrokontrolérem může být nutné využít mnoho různých příkazů postupně (např. nastavení globální adresy, nastavení délky stránky, nastavení konfigurace) a také s přihlédnutím k tomu, jak pracuje FTDI čip, není vhodné provádět pouze jeden příkaz naráz. V případě, že by byl odeslán příkaz a poté aplikace čekala na odpověď, zpoždění by dosahovalo i desítek milisekund na jeden příkaz, což by mohlo být u některých operací nepříjemné i pro uživatele. Z toho důvodu jsem vytvořil systém řetězení příkazů.

Každý příkaz pro programátor se zapíše jako asociativní pole, které obsahuje data k odeslání, počet dat, která se mají přijmout jako odpověď, odkaz na funkci, které se mají tato přijatá data předat a parametry, které se předávají stejné funkci. Dále obsahuje pokyn, zda se má odesílání zastavit až do doby, kdy bude ukončeno přijímání dat pro tento příkaz. Tato pole se poté ukládají do fronty příkazů.

Zpracování se provádí tak, že se nezávisle na sobě odesílají data určená k odeslání. Pokud je přijatý dostatek dat k danému příkazu, tak se zavolá funkce, která tato data zpracuje. Toho je dosaženo pomocí dvou ukazatelů, čtecího a odesílacího. Pouze v případě, že je rozkázáno čekat na přijetí dat, se odesílací ukazatel zastaví a nepřekročí na další příkaz. Toto využívám u příkazů, u nichž je poměrně vysoké riziko selhání (vstup do programovacího módu Atmel, programování s verifikací).

Tato metoda umožňuje dosahovat mnohem vyšších rychlostí, protože FTDI čip tak odesílá příkazy v blocích. Programátor je navržen způsobem, aby s tímto postupem počítal (řízení toku, odesílání kódu příkazu okamžitě zpět).

Při přijímání dat se posouvá progress bar na hlavním okně, aby měl uživatel přehled o postupu (a aby aplikace nevypadala příliš staticky). Postup se počítá podle počtu přijatých bytů, proto programátor odesílá informaci o zahájení programování každé stránky. Vzhledem k tomu, že počet očekávaných bytů je znám, je možno tuto informaci přepočítat na procenta a zobrazit jako progress bar.

6.3.5 Mikrokontroléry a jejich nastavení

Pro ukládání informací o mikrokontrolérech se využívá konfiguračních souborů, které jsou parsovány pomocí vestavěného modulu *ConfigParser*. Tyto soubory obsahují všechny potřebné informace pro programování daného mikrokontroléru jako velikost pamětí, přístup k ní, různé pokyny pro programovací postupy a podobně. Dále obsahují seznam FUSE / konfiguračních dat, jejich originální hodnoty a popisy jednotlivých bitů.

Tyto popisy obsahují pouze názvy jednotlivých bytů (často oficiální). Pro ně platí, že každý může být použitý v celém systému pouze jednou. Důvodem je, že tyto byty jsou popsány v konfiguraci FUSE bytů (*cfg/fuse.ini*). Tato konfigurace umožňuje spojit nastavené bity do skupin, přiřadit jim názvy a volitelné hodnoty. Nastavené bity se poté v uživatelském rozhraní zobrazí jako seznamy, ze kterých je možno vybrat jednu možnost. Pomocí funkcí v programu jsou tyto výběry poté přepočítány na vlastní hodnotu, kterou je možné zapsat do mikrokontroléru. Při vyčtení hodnot z mikrokontroléru se naopak v seznamech automaticky vyberou načtené hodnoty. Užitá implementace je výhodná, protože řada mikrokontrolérů od stejných firem spolu sdílí nastavení a tudíž je možné jeden popis využít vícekrát.

6.3.6 Načítání *.hex* souborů

Intel HEX je formát zápisu binárních dat do textového souboru specifikovaný společností Intel v roce 1988.[8]

Překladače kódů pro mikrokontroléry používají ve většině případů formát Intel HEX pro zápis dat. Proto je třeba, aby aplikace dokázala tento formát přečíst a uložit.

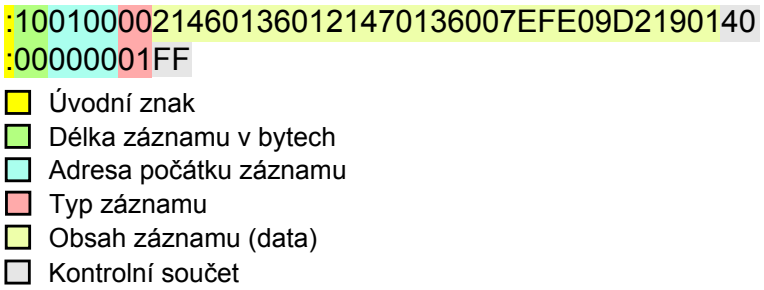
Formát Intel HEX je textový formát vytvořený tak, aby bylo možné ho přepsat na papír a zpět do zařízení. Proto je poměrně jednoduché ho číst a zapisovat.

Na obrázku je vidět formát souboru Intel HEX. Typů záznamu je šest:

0. Datový záznam — obsahuje přímo data a 16ti bitovou adresu.

1. Konec souboru — tento záznam musí být použit právě jednou a to na konci souboru. Adresa se použije 0x0000.

2. Rozšířený segmentový záznam — tento záznam vychází ze stylu reálného adresování používaného u procesorů x86. 16 bitů dat, které jsou vloženy v tomto záznamu se



Obrázek 6.1: Formát Intel HEX

posune o 4 bity doleva a adresy v následných datových záznamech se k této hodnotě přičítají.

3. Záznam počátečního segmentu — tento záznam nastavuje registry CS:IP. Je platný pro procesory x86 a u mikrokontrolérů se nepoužívá.

4. Rozšířený lineární záznam — tento záznam je podobný typu 2. Pouze se data zapsaná u tohoto záznamu posouvají doleva o 16 bitů a ne o 4 bity, tudíž je možno adresovat až 32 bitů adresy. Tento záznam se používá poměrně často. Adresa u takovéhoto záznamu je nulová.

5. Záznam počáteční lineární adresy — tento záznam je podobný typu 3. Nastavuje registr EIP. Protože tento registr mikrokontroléry nemají, tento záznam se nepoužívá.

Záznam takto může obsahovat až 2^{32} hodnot, což je 4 GB dat. Tolik paměti v současné době nemá žádný mikrokontrolér, který by toto zařízení dokázalo programovat, tudíž je zápis pomocí Intel HEX dostačující.

Data pro mikrokontrolér Atmel se zapisují do dvou souborů, jeden je pro programovou paměť a druhý pro datovou. Hodnoty FUSE bohužel u Atmelu do *.hex* souboru vložit nejdou, tudíž je nutné je před programováním nastavit ručně.

Data pro mikrokontrolér PIC se dodávají v jednom souboru *.hex*, ve formátu který lze rovnou odeslat programátoru. Tzn. například 8bitová data pro EEPROM stejně zabírají 2 byte v *.hex* souboru. Záznam je zapsán jako little-endian. Aplikace dokáže přečíst všechna potřebná data ze souboru, včetně nastavení konfiguračních dat.

Aplikace dokáže i uložit vyčtená data do souboru *.hex* pro použití v jiných programátorech, případně pro jejich úschovu. Pokud je mikrokontrolér typu PIC, tak aplikace uloží všechny nutné parametry, včetně nastavení konfiguračních dat.

Kapitola 7

Analýza rychlosti programování

V této části proberu výsledky práce, které se zabývají jedním z hlavních kritérií, na které jsem se soustředil — rychlostí programování. Zvolil jsem dva zástupce z řad mikrokontrolérů, jeden AVR a druhý PIC, na kterých jsem analýzu provedl.

7.1 Mikrokontrolér AVR ATMega8

7.1.1 Teoretická maximální rychlost programování

V případě, že využijeme možnosti pollingu při zápisu, tak není možné dopředu vědět, jak dlouho bude samotná operace zápisu stránky trvat. Proto je třeba při výpočtu teoretické maximální rychlosti programování nutné vycházet z toho, že se polling nevyužívá a počítat se zadanou rychlostí.

Dobu potřebnou k odeslání každého příkazu lze spočítat z komunikační frekvence dle vzorce $t = \frac{32}{f}$, kde f je frekvence komunikace. Budu uvažovat nejvyšší možnou komunikační frekvenci dosažitelnou při použitém napětí, což je 2 MHz. Doba potřebná k odeslání jednoho příkazu je tedy 16 µs, neboli 0,016 ms.

Každou stránku je nutné zapsat do vyrovnávací paměti mikrokontroléru. Stránka má 64 bytů a tudíž je třeba odeslat 64 příkazů na zápis každé stránky. Poté se vydá příkaz pro zápis (1 příkaz). Čekání na zápis dle datasheetu trvá 4,5 ms, přičemž stránek je celkem 128. Nakonec se data verifikují, což znamená 8192 příkazů pro verifikaci. Z těchto údajů lze vypočítat teoretickou maximální rychlost programování:

$$128 * (64 * 0,016\,ms + 0,016\,ms + 4,5\,ms) + 8192 * 0,016\,ms = 840,192\,ms$$

Teoretická maximální rychlost zápisu, včetně verifikace, je tedy $840,192\,ms$.

7.1.2 Naměřená rychlost programování

Rychlost programování byla změřena pomocí logického analyzátoru, pro maximální přesnost měření. Naměřená rychlost programování, včetně verifikace je $821,776\,ms$. Tato rychlost je vyšší než teoretická z toho důvodu, že využití pollingu mírně zkracuje nutnou čekací dobu mezi stránkami, což se ale na 128 stránkách projeví. Postup programování navíc probíhá tak, že mezi jednotlivými odeslanými příkazy není vkládána pauza ani jeden takt hodinového signálu, což je možné díky paralelizaci při použití hradlového pole.

7.2 Mikrokontrolér PIC16F1518

7.2.1 Teoretická maximální rychlost programování

Podobně jako u mikrokontrolérů AVR je třeba spočítat dobu potřebnou k odeslání jednoho příkazu. Vzhledem k tomu, že po odeslání každého příkazu nebo dat je třeba čekat $1\,\mu s$, přičteme tuto hodnotu k vypočtenému času. Čas pro odeslání se tedy vypočte jako $t = \frac{x}{f} + 1\,\mu s$. Zde x je počet taktů hodinového signálu v příkazu a f je frekvence komunikace. Při dodržení maximální povolené rychlosti programování 5 MHz lze odeslat příkaz (který má 6 taktů) za 2,2 µs a data (která mají 16 taktů) za 4,2 µs.

Podobně jako u mikrokontroléru AVR je třeba zapsat každou stránku do vyrovnávací paměti mikrokontroléru a poté spustit programování. Každá stránka má délku 32 slov, přičemž po každém zápisu je nutné zvýšit adresu zápisu v mikrokontroléru. Stránek je 512. Poté je třeba vydat pokyn k zápisu, přičemž zápis trvá 2 ms.[1] Po každém zápisu je nutné odeslat příkaz k ukončení programování a poté čekat ještě 300 µs na vybití programovacích kondenzátorů. Toto sice není nutné při interně časovaném programování, ale tato metoda je přesto rychlejší. Čas potřebný k programování tedy spočítáme jako:

$$512 * (32 * (2,2\,\mu s + 4,2\,\mu s + 2,2\,\mu s) + 2,2\,\mu s + 2000 + 2,2\,\mu s + 300) = 1320755,2\,\mu s$$

Doba potřebná k programování je tedy 1320755,2 µs, což je přibližně 1320,755 ms.

Doba potřebná k verifikaci (kterou je u tohoto typu nutné provádět až po programování) se spočítá obdobně. Po přečtení každého slova se navýší adresový ukazatel a oba dva přečtené byty se odešlou po sériové lince. Prakticky stejně funguje vyčítání dat. Doba potřebná k vyčítání se tedy spočítá takto:

$$16384 * (2,2\,\mu s + 4,2\,\mu s + 2,2\,\mu s) = 140902,4\,\mu s$$

Doba nutná k vyčtení/verifikaci je tedy 140902,4 µs, po převodu a zaokrouhlení 140,902 ms.

7.2.2 Naměřená rychlost programování

Rychlost programování byla také změřena logickým analyzátorem. Doba potřebná k naprogramování zařízení byla 1422,711 ms. Tato hodnota je vyšší než teoretická naměřená z několika důvodů.

Nutné čekání 300 µs se provádí jako čekání o délce 500 µs, protože čekací modul umí provádět čekání pouze v násobcích půl milisekundy. Zvýšení přesnosti by přineslo vyšší složitost programu pro FPGA a způsobilo problémy s jeho umístěním do zařízení, které je už tak plné.

Frekvence komunikace není naprosto optimálních 5 MHz, ale 4,8 MHz, jelikož optimální frekvence není jednoduchá na vypočítání. Viz. tabulka v příloze C.

Rychlost vyčítání byla změřena také logickým analyzátorem na 358,214 µs. Tato rychlost je také vyšší než teoretická naměřená, což je způsobeno rychlostí sériové linky. Po každém provedeném čtení se odesílají dva byte dat, což způsobuje potřebu čekat na dokončení odesílání i přes to, že odesílání a komunikace s mikrokontrolérem probíhají paralelně. Tento problém není bohužel snadné vyřešit, kvůli problematické podpoře vyšších rychlostí sériového portu než 921600 bit/s u většiny počítačů.

[1] V datasheetu je uvedena hodnota 1 ms — 2,1 ms. S hodnotami nižšími než 2 ms se mi ovšem nedařilo zapisovat data konzistentně správně. Vliv na to bude mít pravděpodobně napájecí napětí 3,3 V nebo jsem měl k dispozici chybný vzorek mikrokontroléru.

Tato doba je ale tak krátká, že v podstatě nemá žádný vliv na uživatelský komfort. U mikrokontrolérů Atmel se takto po každém čtení odešle pouze jediný byte, což tento problém nevyvolá.

Kapitola 8

Závěr

Tato práce měla za cíl využít FITkit jako zařízení, které bude schopné programovat mikrokontroléry z vybraných rodin a k tomu odpovídající ovládací aplikaci pro PC. Zvolil jsem mikrokontroléry Microchip PIC a Atmel AVR.

Jedním z cílů bylo navrhnout zařízení tak, aby pro jeho použití bylo potřeba minimální množství externích komponent. Tento bod zařízení splňuje, jelikož pro jeho použití není potřeba žádná externí komponenta, pouze propojky mezi piny na FITkitu a programovaným mikrokontrolérem. Podařilo se dosáhnout velmi vysokých rychlostí programování, které se blíží teoretickým maximálním rychlostem a v případě mikrokontrolérů Atmel jich i dosahují. Ovládací aplikace je vytvořena tak, aby její použití bylo jednoduché pro uživatele. Rozsah podporovaných obvodů zahrnuje nejpoužívanější typy od společnosti Atmel. U společnosti PIC je tato situace zkomplikována potřebou 12 V napájení u většiny starších typů mikrokontrolérů. Novější typy jsou podporovány.

Práce je navržena tak, aby principielně nevyžadovala FITkit. Při použití FPGA (ideálně při zachování FPGA *Spartan 3*) a vhodného USB řadiče je možnost navrhnout jednoduché zařízení, které dokáže využívat tuto aplikaci pouze s minimálními změnami.

Při tvorbě práce jsem si navíc ověřil možnosti, výhody a nevýhody využití FPGA pro programování mikrokontrolérů. Použití FPGA pro programování mikrokontrolérů se ukázalo jako velmi rychlé, hlavně vzhledem k užití paralelizace. Schopnost přijímat data, zpracovávat je a zároveň řídit mikrokontrolér umožňila komunikaci s mikrokontrolérem AVR bez pauz v odesílaných datech. Nevýhoda použití FPGA je hlavně ve složitosti implementace. Některé části, které by na procesoru byly poměrně jednoduché, jsou s hradlovým polem velmi komplikované na implementaci. Testování je také složitější s nutností simulovat i velké úseky obvodu a následně využívat logický analyzátor pro ověření funkčnosti. Je třeba brát v ohled problémy s metastabilitou klopných obvodů a zpoždění signálů.

Možností rozšíření je mnoho. Zřejmá varianta je rozšířit podporu o další mikrokontroléry, hlavně PIC18. Jednodušší možnost je vytvořit přídavný modul, který bude umět generovat napětí 12 V a tím bude schopen programovat i ty mikrokontroléry PIC, které toto napětí vyžadují. Přidání podpory pro ladění přímo na čipu by mohlo být lákavé, bohužel se provádí pomocí rozhraní JTAG a jeho přidání by bylo velmi komplikované, i vzhledem k zaplnění hradlového pole.

Literatura

[1] Atmel Corporation. *AVR 8- and 32-bit microcontrollers* [online]. [cit. 2012-4-27]. Dostupné na: <http://www.atmel.com/products/microcontrollers/avr/default.aspx>.

[2] Atmel Corporation. *AVR910: In-System Programming* [online]. Revize 0943E-AVR-08/08. 2008 [cit. 2012-4-24]. Dostupné na: <http://www.atmel.com/Images/doc0943.pdf>.

[3] Atmel Corporation. *ATmega16, ATmega16L* [online]. Revize 2466T-AVR-07/10. 2010 [cit. 2012-4-28]. Dostupné na: <http://www.atmel.com/Images/doc2466.pdf>.

[4] Atmel Corporation. *ATtiny13, ATtiny13V* [online]. Revize 2535J-AVR-08/10. 2010 [cit. 2012-4-30]. Dostupné na: <http://www.atmel.com/Images/doc2535.pdf>.

[5] Atmel Corporation. *ATmega8* [online]. Revize 2486Z-AVR-02/11. 2011 [cit. 2012-5-7]. Dostupné na: <http://www.atmel.com/Images/doc2486.pdf>.

[6] Atmel Corporation. *ATtiny4/5/9/10* [online]. Revize 8127E-AVR-11/11. 2011 [cit. 2012-4-27]. Dostupné na: <http://www.atmel.com/Images/doc8127.pdf>.

[7] Future Technology Devices International Ltd.. *FTDI Support* [online]. 2012 [cit. 2012-4-29]. Dostupné na: <http://www.ftdichip.com/Support/FAQs.htm>.

[8] Intel Corporation. *Hexadecimal Object File Format Specification* [online]. Revize A. 1988 [cit. 2012-4-14]. Dostupné na: <http://microsym.com/editor/assets/intelhex.pdf>.

[9] Katzen, S. *The quintessential PIC microcontroller*. 3. vyd. [b.m.]: Springer, 2004. ISBN 1-85233-309-X.

[10] Matoušek, D. *Práce s mikrokontroléry Atmel ATmega16*. 1. vyd. [b.m.]: BEN - technická literatura, 2006. ISBN 80-7300-174-8.

[11] Microchip Technology, Inc.. *PIC16F818/819 Flash Memory Programming Specification* [online]. 2004 [cit. 2012-4-28]. Dostupné na: <http://ww1.microchip.com/downloads/en/DeviceDoc/39603c.pdf>.

[12] Microchip Technology, Inc.. *8-Bit PIC Microcontrollers* [online]. 2006 [cit. 2012-4-30]. Dostupné na: <http://ww1.microchip.com/downloads/en/DeviceDoc/39630C.pdf>.

[13] Microchip Technology, Inc.. *PIC Microcontrollers with Enhanced Mid-Range Core* [online]. 2009 [cit. 2012-4-30]. Dostupné na: <http://ww1.microchip.com/downloads/en/DeviceDoc/41382a.pdf>.

[14] Microchip Technology, Inc.. *PIC16(L)F151X/152X Memory Programming Specification* [online]. Revize B. 2010-2011 [cit. 2012-4-28]. Dostupné na: <http://ww1.microchip.com/downloads/en/DeviceDoc/41442B.pdf>.

[15] Osborne, A. *An Introduction to Microcomputers Volume 1 - Basic Concepts*. 1. vyd. [b.m.]: Adam Osborne and Associates, 1976. ISBN 0-931988-02-0.

[16] Vašíček, Z. *FITkit* [online]. [cit. 2012-4-29]. Dostupné na: <http://merlin.fit.vutbr.cz/FITkit/>.

[17] Xilinx, Inc.. *Spartan-3 FPGA Family Data Sheet* [online]. Verze 2,5. 2009 [cit. 2012-4-12]. Dostupné na: <http://ww1.microchip.com/downloads/en/DeviceDoc/41442B.pdf>.

Příloha A

Seznam podporovaných mikrokontrolérů

A.1 Atmel AVR

A.1.1 tinyAVR

ATtiny1634, ATtiny13A, ATtiny13, ATtiny24A, ATtiny24, ATtiny44A, ATtiny44, ATtiny84A, ATtiny84, ATtiny261A, ATtiny261, ATtiny461A, ATtiny461, ATtiny861A, ATtiny861, ATtiny26, ATtiny2313A, ATtiny2313, ATtiny4313, ATtiny43U, ATtiny28L, ATtiny48, ATtiny88, ATtiny87, ATtiny167.

A.1.2 megaAVR

ATmega48, ATmega48A, ATmega48P, ATmega48PA, ATmega8, ATmega8515, ATmega8535, ATmega88, ATmega88A, ATmega88PA, ATmega8A, ATmega16, ATmega162, ATmega164A, ATmega164PA, ATmega165P, ATmega165PA, ATmega168, ATmega168A, ATmega168P, ATmega168PA, ATmega16A, ATmega32, ATmega324A, ATmega324P, ATmega324PA, ATmega325, ATmega3250, ATmega325A, ATmega325P, ATmega325PA, ATmega328, ATmega328P, ATmega32A, ATmega64, ATmega640, ATmega644, ATmega644A, ATmega644P, ATmega644PA, ATmega645, ATmega6450, ATmega6450A, ATmega6450P, ATmega645A, ATmega645P, ATmega64A, ATmega128, ATmega1280, ATmega1281, ATmega1284, ATmega1284P, ATmega128A, AT90CAN128, AT90CAN32, AT90CAN64, ATmega16M1, ATmega32M1, ATmega64M1, AT90PWM1, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM81, AT90PWM161, AT90USB1286, AT90USB1287, AT90USB162, AT90USB646, AT90USB647, AT90USB82, ATmega16U2, ATmega16U4, ATmega32U2, ATmega32U4, ATmega8U2, ATmega169A, ATmega169P, ATmega169PA, ATmega329, ATmega3290, ATmega3290A, ATmega3290P, ATmega329A, ATmega329P, ATmega329PA.

A.2 Microchip PIC

A.2.1 PIC12

PIC12F1501, PIC12F1822, PIC12F1840.

A.2.2 PIC16

PIC16F1454, PIC16F1455, PIC16F1458, PIC16F1459, PIC16F1503, PIC16F1507, PIC16F1508, PIC16F1509, PIC16F1512, PIC16F1513, PIC16F1516, PIC16F1517, PIC16F1518, PIC16F1519, PIC16F1526, PIC16F1527, PIC16F1782, PIC16F1783, PIC16F1784, PIC16F1786, PIC16F1787, PIC16F1823, PIC16F1824, PIC16F1825, PIC16F1826, PIC16F1827, PIC16F1828, PIC16F1829, PIC16F1847, PIC16F1933, PIC16F1934, PIC16F1936, PIC16F1937, PIC16F1938, PIC16F1939, PIC16F1946, PIC16F1947, PIC16F818, PIC16F819, PIC16F87, PIC16F88, PIC16F882, PIC16F883, PIC16F884, PIC16F886, PIC16F887, PIC16LF1902, PIC16LF1903, PIC16LF1904, PIC16LF1906, PIC16LF1907.

Příloha B

Návod k použití

Na FITkitu musí být nahrán ovládací program, ideálně pomocí programu QDevKit. QDevKit by během programování neměl být spuštěn, aby se nepokoušel připojovat k sériovým portům. FITkit musí být během připojení spuštěn, je tedy třeba odstranit jumper JP8. Poté FITkit připojíme k mikrokontroléru podle obrázku B.1 a k počítači přes rozhraní USB.

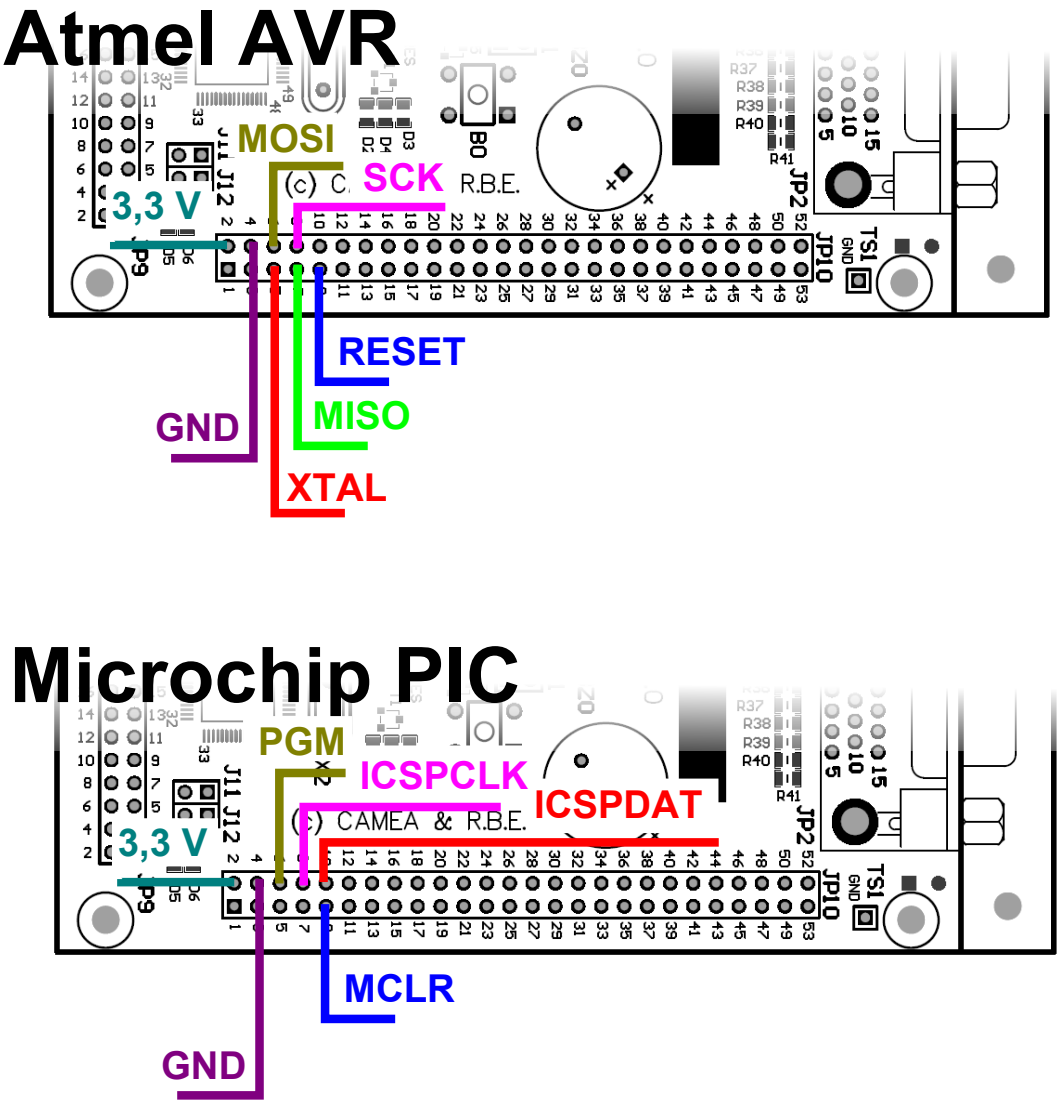
Ovládací aplikace je vytvořena tak, aby její použití bylo co nejvíce intuitivní. Aplikace se pokouší připojit okamžitě po spuštění, což napoprvé pravděpodobně selže. V menu Nastavení je třeba nastavit port, přes který je připojen FITkit, respektive FPGA[1].

Pokud je připojen mikrokontrolér Atmel, tak je možné využít automatickou detekci připojeného typu mikrokontroléru. U PIC této možnosti bohužel nelze využít. Při volbě rodiny mikrokontrolérů se komunikační frekvence nastaví automaticky, ale pouze pokud uživatel tuto frekvenci nezmění. Pokud není frekvence správná, pak je nutné ji nastavit ručně, podle parametrů mikrokontroléru (nejnižší frekvence by měla fungovat vždy).

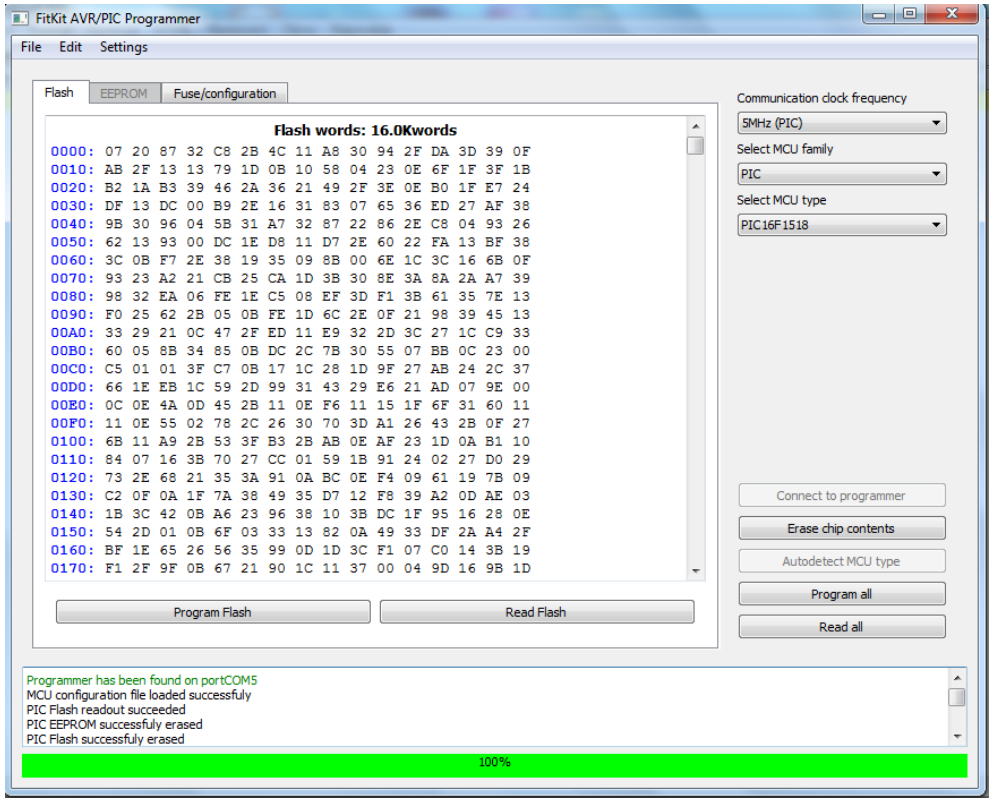
Načítání dat z .hex souborů je přímočaré, u mikrokontrolérů PIC obsahují .hex soubory všechna potřebná data, která se všechna vyčtou automaticky. U mikrokontrolérů AVR není možné do .hex souboru umístit nastavení FUSE bytů, je nutné je nastavovat ručně. EEPROM paměť musí být umístěna ve zvláštním souboru, s čímž programátor počítá.

Před zápisem je třeba obsah mikrokontroléru smazat, pokud je to možné.

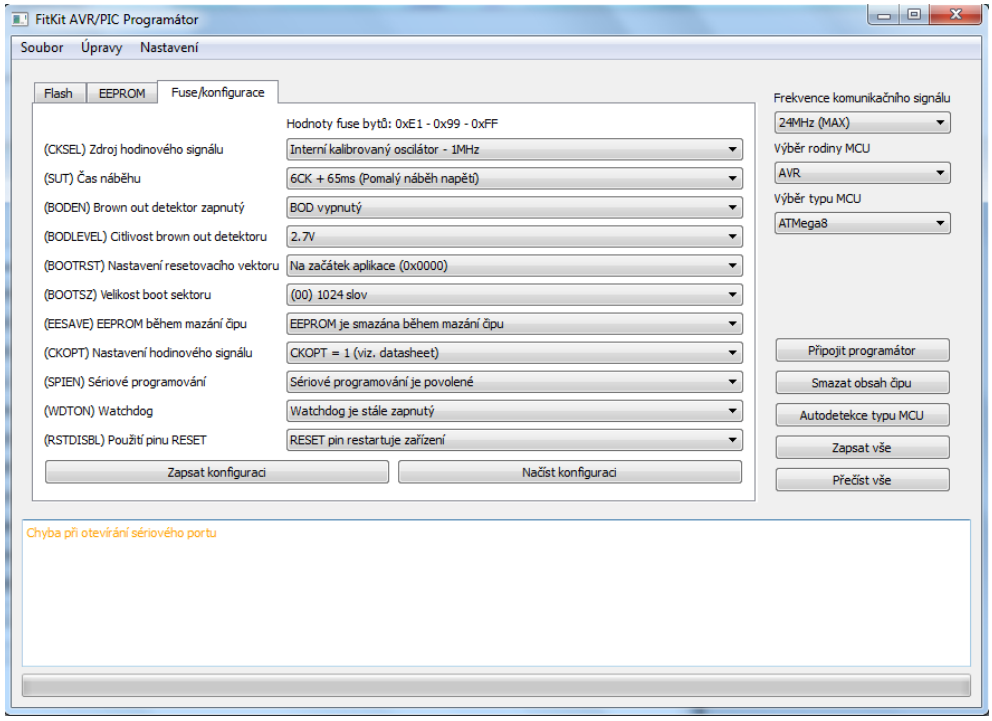
[1] Většinou jde o port s nižším číslem



Obrázek B.1: Připojení programovaného mikrokontroléru k FITkitu



Obrázek B.2: Snímek aplikace — Angličtina a Flash paměť



Obrázek B.3: Snímek aplikace — FUSE byty

Příloha C

Frekvence oscilátoru

Tabulky níže zobrazují možné frekvence hlavního oscilátoru, které splňují požadavky zadané v části 5.2.3. Zde jsou zobrazeny pouze ty rychlosti, které splňují požadavek na dělitelnost pro UART (dělitelnost 921600) a ty, které je možné vytvořit z frekvence 7.3728 MHz v DCM modulu FPGA.

První sloupec tabulky ukazuje samotnou rychlost a druhý sloupec ukazuje nutné nastavení DCM modulu pro vytvoření této frekvence. Další sloupce už ukazují blízkost cílovým frekvencím 8 MHz, 5 MHz, 2 MHz, 1 MHz a 250 kHz. Hodnota ve sloupci ukazuje odchylku od požadované frekvence v kHz, v procentech a nutné podělení základní frekvence pro získání cílové frekvence. Barva slouží pro snadnější orientaci. Zelená ukazuje odchylku 2% a lepší, Oranžová ukazuje odchylku 5% a lepší a nakonec Červená ukazuje odchylku horší než 5%.

Z těchto tabulek je jasně patrné, proč byla zvolena frekvence 47,9232 MHz. Pro výpočet většiny frekvencí je velmi vhodná, pouze programování mikrokontrolérů PIC s 5 MHz frekvencí bude pomalejší o 4,33% od optima.

Frekvence	DCM	8MHz	5MHz
0,9216MHz	*1 /8	7078,40kHz(/1 - 768,06%)	4078,40kHz (/1 - 442,53%)
1,8432MHz	*1 /4	6156,80kHz(/1 - 334,03%)	3156,80kHz (/1 - 171,27%)
2,7648MHz	*3 /8	5235,20kHz(/1 - 189,35%)	2235,20kHz (/1 - 80,84%)
3,6864MHz	*1 /2	4313,60kHz(/1 - 117,01%)	1313,60kHz (/1 - 35,63%)
4,6080MHz	*5 /8	3392,00kHz(/1 - 73,61%)	392,00kHz (/1 - 8,51%)
5,5296MHz	*3 /4	2470,40kHz(/1 - 44,68%)	2235,20kHz (/2 - 80,84%)
6,4512MHz	*7 /8	1548,80kHz(/1 - 24,01%)	1774,40kHz (/2 - 55,01%)
7,3728MHz	*1 /1	627,20kHz(/1 - 8,51%)	1313,60kHz (/2 - 35,63%)
8,2944MHz	*9 /8	3852,80kHz(/2 - 92,90%)	852,80kHz (/2 - 20,56%)
9,2160MHz	*5 /4	3392,00kHz(/2 - 73,61%)	392,00kHz (/2 - 8,51%)
10,1376MHz	*11 /8	2931,20kHz(/2 - 57,83%)	2465,60kHz (/4 - 97,29%)
11,0592MHz	*3 /2	2470,40kHz(/2 - 44,68%)	2235,20kHz (/4 - 80,84%)
11,9808MHz	*13 /8	2009,60kHz(/2 - 33,55%)	2004,80kHz (/4 - 66,93%)
12,9024MHz	*7 /4	1548,80kHz(/2 - 24,01%)	1774,40kHz (/4 - 55,01%)
13,8240MHz	*15 /8	1088,00kHz(/2 - 15,74%)	1544,00kHz (/4 - 44,68%)
14,7456MHz	*2 /1	627,20kHz(/2 - 8,51%)	1313,60kHz (/4 - 35,63%)
15,6672MHz	*17 /8	166,40kHz(/2 - 2,12%)	1083,20kHz (/4 - 27,66%)
16,5888MHz	*9 /4	3852,80kHz(/4 - 92,90%)	852,80kHz (/4 - 20,56%)
17,5104MHz	*19 /8	3622,40kHz(/4 - 82,75%)	622,40kHz (/4 - 14,22%)
18,4320MHz	*5 /2	3392,00kHz(/4 - 73,61%)	392,00kHz (/4 - 8,51%)
19,3536MHz	*21 /8	3161,60kHz(/4 - 65,34%)	161,60kHz (/4 - 3,34%)
20,2752MHz	*11 /4	2931,20kHz(/4 - 57,83%)	1620,80kHz (/6 - 47,96%)
21,1968MHz	*23 /8	2700,80kHz(/4 - 50,97%)	1467,20kHz (/6 - 41,53%)
22,1184MHz	*3 /1	2470,40kHz(/4 - 44,68%)	1313,60kHz (/6 - 35,63%)
23,0400MHz	*25 /8	2240,00kHz(/4 - 38,89%)	1160,00kHz (/6 - 30,21%)
23,9616MHz	*13 /4	2009,60kHz(/4 - 33,55%)	1006,40kHz (/6 - 25,20%)
24,8832MHz	*27 /8	1779,20kHz(/4 - 28,60%)	852,80kHz (/6 - 20,56%)
25,8048MHz	*7 /2	1548,80kHz(/4 - 24,01%)	699,20kHz (/6 - 16,26%)
26,7264MHz	*29 /8	1318,40kHz(/4 - 19,73%)	545,60kHz (/6 - 12,25%)
27,6480MHz	*15 /4	1088,00kHz(/4 - 15,74%)	392,00kHz (/6 - 8,51%)
28,5696MHz	*31 /8	857,60kHz(/4 - 12,01%)	238,40kHz (/6 - 5,01%)
29,4912MHz	*4 /1	627,20kHz(/4 - 8,51%)	84,80kHz (/6 - 1,73%)
31,3344MHz	*17 /4	166,40kHz(/4 - 2,12%)	1083,20kHz (/8 - 27,66%)
33,1776MHz	*9 /2	2470,40kHz(/6 - 44,68%)	852,80kHz (/8 - 20,56%)
35,0208MHz	*19 /4	2163,20kHz(/6 - 37,06%)	622,40kHz (/8 - 14,22%)
36,8640MHz	*5 /1	1856,00kHz(/6 - 30,21%)	392,00kHz (/8 - 8,51%)
38,7072MHz	*21 /4	1548,80kHz(/6 - 24,01%)	161,60kHz (/8 - 3,34%)
40,5504MHz	*11 /2	1241,60kHz(/6 - 18,37%)	944,96kHz (/10 - 23,30%)
42,3936MHz	*23 /4	934,40kHz(/6 - 13,22%)	760,64kHz (/10 - 17,94%)
44,2368MHz	*6 /1	627,20kHz(/6 - 8,51%)	576,32kHz (/10 - 13,03%)
46,0800MHz	*25 /4	320,00kHz(/6 - 4,17%)	392,00kHz (/10 - 8,51%)
47,9232MHz	*13 /2	12,80kHz(/6 - 0,16%)	207,68kHz (/10 - 4,33%)
49,7664MHz	*27 /4	1779,20kHz(/8 - 28,60%)	23,36kHz (/10 - 0,47%)
51,6096MHz	*7 /1	1548,80kHz(/8 - 24,01%)	699,20kHz (/12 - 16,26%)
53,4528MHz	*29 /4	1318,40kHz(/8 - 19,73%)	545,60kHz (/12 - 12,25%)
55,2960MHz	*15 /2	1088,00kHz(/8 - 15,74%)	392,00kHz (/12 - 8,51%)
57,1392MHz	*31 /4	857,60kHz(/8 - 12,01%)	238,40kHz (/12 - 5,01%)
58,9824MHz	*8 /1	627,20kHz(/8 - 8,51%)	84,80kHz (/12 - 1,73%)

Tabulka C.1: Tabulka s analýzou frekvencí pro frekvence 8 MHz a 5 MHz

Frekvence	DCM	2MHz	1MHz	250kHz
0,9216MHz	*1 /8	1078,40kHz (/1 - 117,01%)	78,40kHz (/1 - 8,51%)	19,60kHz (/4 - 8,51%)
1,8432MHz	*1 /4	156,80kHz (/1 - 8,51%)	78,40kHz (/2 - 8,51%)	19,60kHz (/8 - 8,51%)
2,7648MHz	*3 /8	617,60kHz (/2 - 44,68%)	308,80kHz (/4 - 44,68%)	19,60kHz (/12 - 8,51%)
3,6864MHz	*1 /2	156,80kHz (/2 - 8,51%)	78,40kHz (/4 - 8,51%)	19,60kHz (/16 - 8,51%)
4,6080MHz	*5 /8	848,00kHz (/4 - 73,61%)	232,00kHz (/6 - 30,21%)	19,60kHz (/20 - 8,51%)
5,5296MHz	*3 /4	617,60kHz (/4 - 44,68%)	78,40kHz (/6 - 8,51%)	19,60kHz (/24 - 8,51%)
6,4512MHz	*7 /8	387,20kHz (/4 - 24,01%)	193,60kHz (/8 - 24,01%)	1,88kHz (/26 - 0,76%)
7,3728MHz	*1 /1	156,80kHz (/4 - 8,51%)	78,40kHz (/8 - 8,51%)	4,24kHz (/30 - 1,73%)
8,2944MHz	*9 /8	617,60kHz (/6 - 44,68%)	170,56kHz (/10 - 20,56%)	6,05kHz (/34 - 2,48%)
9,2160MHz	*5 /4	464,00kHz (/6 - 30,21%)	78,40kHz (/10 - 8,51%)	7,47kHz (/38 - 3,08%)
10,1376MHz	*11 /8	310,40kHz (/6 - 18,37%)	155,20kHz (/12 - 18,37%)	8,63kHz (/42 - 3,57%)
11,0592MHz	*3 /2	156,80kHz (/6 - 8,51%)	78,40kHz (/12 - 8,51%)	9,58kHz (/46 - 3,99%)
11,9808MHz	*13 /8	3,20kHz (/6 - 0,16%)	1,60kHz (/12 - 0,16%)	0,40kHz (/48 - 0,16%)
12,9024MHz	*7 /4	387,20kHz (/8 - 24,01%)	78,40kHz (/14 - 8,51%)	1,88kHz (/52 - 0,76%)
13,8240MHz	*15 /8	272,00kHz (/8 - 15,74%)	12,57kHz (/14 - 1,27%)	3,14kHz (/56 - 1,27%)
14,7456MHz	*2 /1	156,80kHz (/8 - 8,51%)	78,40kHz (/16 - 8,51%)	4,24kHz (/60 - 1,73%)
15,6672MHz	*17 /8	41,60kHz (/8 - 2,12%)	20,80kHz (/16 - 2,12%)	5,20kHz (/64 - 2,12%)
16,5888MHz	*9 /4	341,12kHz (/10 - 20,56%)	78,40kHz (/18 - 8,51%)	6,05kHz (/68 - 2,48%)
17,5104MHz	*19 /8	248,96kHz (/10 - 14,22%)	27,20kHz (/18 - 2,80%)	6,80kHz (/72 - 2,80%)
18,4320MHz	*5 /2	156,80kHz (/10 - 8,51%)	78,40kHz (/20 - 8,51%)	0,92kHz (/74 - 0,37%)
19,3536MHz	*21 /8	64,64kHz (/10 - 3,34%)	32,32kHz (/20 - 3,34%)	1,88kHz (/78 - 0,76%)
20,2752MHz	*11 /4	310,40kHz (/12 - 18,37%)	78,40kHz (/22 - 8,51%)	2,74kHz (/82 - 1,11%)
21,1968MHz	*23 /8	233,60kHz (/12 - 13,22%)	36,51kHz (/22 - 3,79%)	3,53kHz (/86 - 1,43%)
22,1184MHz	*3 /1	156,80kHz (/12 - 8,51%)	78,40kHz (/24 - 8,51%)	4,24kHz (/90 - 1,73%)
23,0400MHz	*25 /8	80,00kHz (/12 - 4,17%)	40,00kHz (/24 - 4,17%)	4,89kHz (/94 - 2,00%)
23,9616MHz	*13 /4	3,20kHz (/12 - 0,16%)	1,60kHz (/24 - 0,16%)	0,40kHz (/96 - 0,16%)
24,8832MHz	*27 /8	222,63kHz (/14 - 12,53%)	42,95kHz (/26 - 4,49%)	1,17kHz (/100 - 0,47%)
25,8048MHz	*7 /2	156,80kHz (/14 - 8,51%)	7,51kHz (/26 - 0,76%)	1,88kHz (/104 - 0,76%)
26,7264MHz	*29 /8	90,97kHz (/14 - 4,77%)	45,49kHz (/28 - 4,77%)	2,53kHz (/108 - 1,02%)
27,6480MHz	*15 /4	25,14kHz (/14 - 1,27%)	12,57kHz (/28 - 1,27%)	3,14kHz (/112 - 1,27%)
28,5696MHz	*31 /8	214,40kHz (/16 - 12,01%)	47,68kHz (/30 - 5,01%)	3,71kHz (/116 - 1,51%)
29,4912MHz	*4 /1	156,80kHz (/16 - 8,51%)	16,96kHz (/30 - 1,73%)	0,07kHz (/118 - 0,03%)
31,3344MHz	*17 /4	41,60kHz (/16 - 2,12%)	20,80kHz (/32 - 2,12%)	1,31kHz (/126 - 0,53%)
33,1776MHz	*9 /2	156,80kHz (/18 - 8,51%)	24,19kHz (/34 - 2,48%)	2,41kHz (/134 - 0,97%)
35,0208MHz	*19 /4	54,40kHz (/18 - 2,80%)	27,20kHz (/36 - 2,80%)	3,38kHz (/142 - 1,37%)
36,8640MHz	*5 /1	156,80kHz (/20 - 8,51%)	29,89kHz (/38 - 3,08%)	0,92kHz (/148 - 0,37%)
38,7072MHz	*21 /4	64,64kHz (/20 - 3,34%)	32,32kHz (/40 - 3,34%)	1,88kHz (/156 - 0,76%)
40,5504MHz	*11 /2	156,80kHz (/22 - 8,51%)	34,52kHz (/42 - 3,57%)	2,74kHz (/164 - 1,11%)
42,3936MHz	*23 /4	73,02kHz (/22 - 3,79%)	36,51kHz (/44 - 3,79%)	0,63kHz (/170 - 0,25%)
44,2368MHz	*6 /1	156,80kHz (/24 - 8,51%)	38,33kHz (/46 - 3,99%)	1,48kHz (/178 - 0,60%)
46,0800MHz	*25 /4	80,00kHz (/24 - 4,17%)	40,00kHz (/48 - 4,17%)	2,26kHz (/186 - 0,91%)
47,9232MHz	*13 /2	3,20kHz (/24 - 0,16%)	1,60kHz (/48 - 0,16%)	0,40kHz (/192 - 0,16%)
49,7664MHz	*27 /4	85,91kHz (/26 - 4,49%)	4,67kHz (/50 - 0,47%)	1,17kHz (/200 - 0,47%)
51,6096MHz	*7 /1	15,02kHz (/26 - 0,76%)	7,51kHz (/52 - 0,76%)	1,88kHz (/208 - 0,76%)
53,4528MHz	*29 /4	90,97kHz (/28 - 4,77%)	10,13kHz (/54 - 1,02%)	0,22kHz (/214 - 0,09%)
55,2960MHz	*15 /2	25,14kHz (/28 - 1,27%)	12,57kHz (/56 - 1,27%)	0,92kHz (/222 - 0,37%)
57,1392MHz	*31 /4	95,36kHz (/30 - 5,01%)	14,84kHz (/58 - 1,51%)	1,57kHz (/230 - 0,63%)
58,9824MHz	*8 /1	33,92kHz (/30 - 1,73%)	16,96kHz (/60 - 1,73%)	0,07kHz (/236 - 0,03%)

Tabulka C.2: Tabulka s analýzou frekvencí pro frekvence 2 MHz, 1 MHz a 250 kHz

Příloha D

Komunikace s programátorem

Tabulky v této příloze popisují komunikaci se zařízením přes linku UART. První tři obsahují seznam příkazů, které je možné použít, ostatní obsahují popis konfiguračních bytů uvedených v těchto příkazech.

Při přijetí příkazu zařízení jako první odpovídá kódem přijatého příkazu. Tato odpověď platí pro všechny příkazy a není v těchto tabulkách zobrazena.

Příkaz	**Název**	**Data**	**Odpověď**	**Popis**
0x00	Test přítomnosti	žádná	0xAA	Otestuje, zda je zařízení přítomno. Odpovídá pouze 0xAA.
0x01	Nastavení komunikační rychlosti	1 byte	0xC0	Nastaví frekvenci komunikace podle vzorce $f = 24/(x+1)$, kde x je hodnota přijatého byte.
0x02	Nastavení počtu stránek	2 byte	0xC0	Nastaví počet stránek, v potaz se bere pouze 8 nejnižších významových bitů. Počet stránek odpovídá vzorci $n = x+1$, kde x je přijatá hodnota.
0x03	Nastavení délky stránky	1 byte	0xC0	Nastaví délku stránky v bytech, délka stránky odpovídá vzorci $n = x+1$, kde x je přijatý byte.
0x04	Nastavení globální adresy	3 byte	0xC0	Nastaví globální adresu, v potaz se bere pouze nejnižších 17 bitů.

Tabulka D.1: Seznam všeobecných příkazů pro programátor

Příkaz	Název	Data	Odpověď	Popis
0x41	Zahájení programování	žádná	1 byte	Pokusí se zahájit programování, odpovídá 0xC0 při úspěchu a 0xF0 při neúspěchu.
0x42	Zadaný příkaz	4 byte	1 byte	Odešle mikrokontroléru všechny 4 přijaté byte a odpovídá poslední přijatý byte od mikrokontroléru. Používá se např u čtení FUSE bytů.
0x4A	Zadaný příkaz s čekáním	4 byte	1 byte	Odešle mikrokontroléru všechny 4 přijaté byte, poté počká hodnotu pro čekání u FUSE bytů a odpovídá poslední přijatý byte od mikrokontroléru. Používá se u zápisu FUSE bytů.
0x5A	Zadaný příkaz s mazacím čekáním	4 byte	1 byte	Odešle mikrokontroléru všechny 4 přijaté byte, poté počká hodnotu pro čekání u mazání obsahu a odpovídá poslední přijatý byte od mikrokontroléru. Používá se u mazání obsahu čipu.
0x44	Nastavení konfigurace	4 byte	0xC0	Nastaví konfiguraci pro programátor, konfigurace je popsána níže.
0x45	Nastavení čekacích časů	3 byte	0xC0	Nastaví časy čekání pro programátor, formát je popsaný níže.
0x46	Programování	stránky	různé	Zapíše do čipu přijatá data. Po načtení každé stránky odpovídá 0xA0 a po úspěšném dokončení odpovídá 0xC0. Pokud kdykoliv odpoví 0xF0 tak programování selhalo a další 3 byte obsahují adresu selhání.
0x4E	Programování EEPROM	stránky	různé	Stejné jako předchozí, jen zapisuje do datové (EEPROM) paměti.
0x47	Vyčítání	žádná	glob. adr. + 1	Přečte z čipu data o délce odpovídající nastavené globální adrese zvýšené o jedna. Čte od počátku paměťi.
0x4F	Vyčítání EEPROM	žádná	glob. adr. + 1	Stejné jako předchozí, pouze vyčítá EEPROM paměť.

Tabulka D.2: Seznam příkazů pro programátor, při programování Atmel AVR

Příkaz	Název	Data	Odpověď	Popis
0xC1	Zahájení programování	žádná	0xC0	Zahájí programování. Odpovídá 0xC0, protože tato operace nemůže selhat
0xC2	Posun ukazatele	žádná	0xC0	Posune adresu ukazatele o globální adresu + 1, poté odpoví 0xC0
0xC3	Posun ukazatele na konfiguraci	žádná	0xC0	Posune adresu ukazatele na adresu konfigurace
0xC5	Programování	stránky	různé	Zapíše data do čipu dle nastavené konfigurace. Odesílá stejné informace jako programování AVR
0xD5	Vyčítání	žádné	glob. adr. * 2 + 2	Vyčte data z čipu (ze současné adresy). Globální adresa udává počet slov mínus jedna, které se mají vyčíst. Každé slovo má dva byte, které se odesílají.
0xD7	Nastavení konfigurace	3 byte	0xC0	Nastaví konfiguraci pro programování, její popis je níže.
0xD9	Reset ukazatele	žádné	0xC0	Resetuje adresu ukazatele na začátek adresového prostoru.
0xDA	Smazání programové paměti	žádné	0xC0	Smaže programovou paměť čipu.
0xEA	Smazání datové paměti	žádné	0xC0	Smaže datovou paměť čipu.
0xFA	Smazání celého obsahu čipu	žádná	0xC0	Smaže obsah všech pamětí čipu.

Tabulka D.3: Seznam příkazů pro programátor, při programování Microchip PIC

Příkaz	Název	Data	Odpověď	Popis
0xC1	Zahájení programování	žádná	0xC0	Zahájí programování. Odpovídá 0xC0, protože tato operace nemůže selhat
0xC2	Posun ukazatele	žádná	0xC0	Posune adresu ukazatele o globální adresu + 1, poté odpoví 0xC0
0xC3	Posun ukazatele na konfiguraci	žádná	0xC0	Posune adresu ukazatele na adresu konfigurace
0xC5	Programování	stránky	různé	Zapíše data do čipu dle nastavené konfigurace. Odesílá stejné informace jako programování AVR
0xD5	Vyčítání	žádné	glob. adr. * 2 + 2	Vyčte data z čipu (ze současné adresy). Globální adresa udává počet slov mínus jedna, které se mají vyčíst. Každé slovo má dva byte, které se odesílají.
0xD7	Nastavení konfigurace	3 byte	0xC0	Nastaví konfiguraci pro programování, její popis je níže.
0xD9	Reset ukazatele	žádné	0xC0	Resetuje adresu ukazatele na začátek adresového prostoru.
0xDA	Smazání programové paměti	žádné	0xC0	Smaže programovou paměť čipu.
0xEA	Smazání datové paměti	žádné	0xC0	Smaže datovou paměť čipu.
0xFA	Smazání celého obsahu čipu	žádná	0xC0	Smaže obsah všech pamětí čipu.

Tabulka D.4: Seznam příkazů pro programátor, při programování Microchip PIC

Bit	Název	Popis
7	`CFG_EEPROM_BLOCK`	Určuje, zda se má při programování EEPROM paměti využívat stránkový režim programování.
6	`CFG_NOPOLLING`	Zakáže využití pollingu při zápisu.
5	`CFG_OLDPOLLING`	Bude využívat polling typu RDY/BUSY.
4-0	Vyhrazeno	Tyto bity nejsou využity.

Tabulka D.5: Seznam bitů při konfiguraci Atmel (1 byte, 0x44)

Bit	Název	Popis
24 - 19	`FUSE_WAIT_LEN`	Délka čekání při zápisu FUSE bytů.
18 - 13	`ERASE_WAIT_LEN`	Délka čekání při mazání obsahu čipu.
12 - 7	`EEPROM_WAIT_LEN`	Délka čekání při zápisu EEPROM.
6 - 0	`FLASH_WAIT_LEN`	Délka čekání při zápisu Flash.

Tabulka D.6: Seznam bitů při nastavení délky čekání Atmel (3 byte, 0x45)

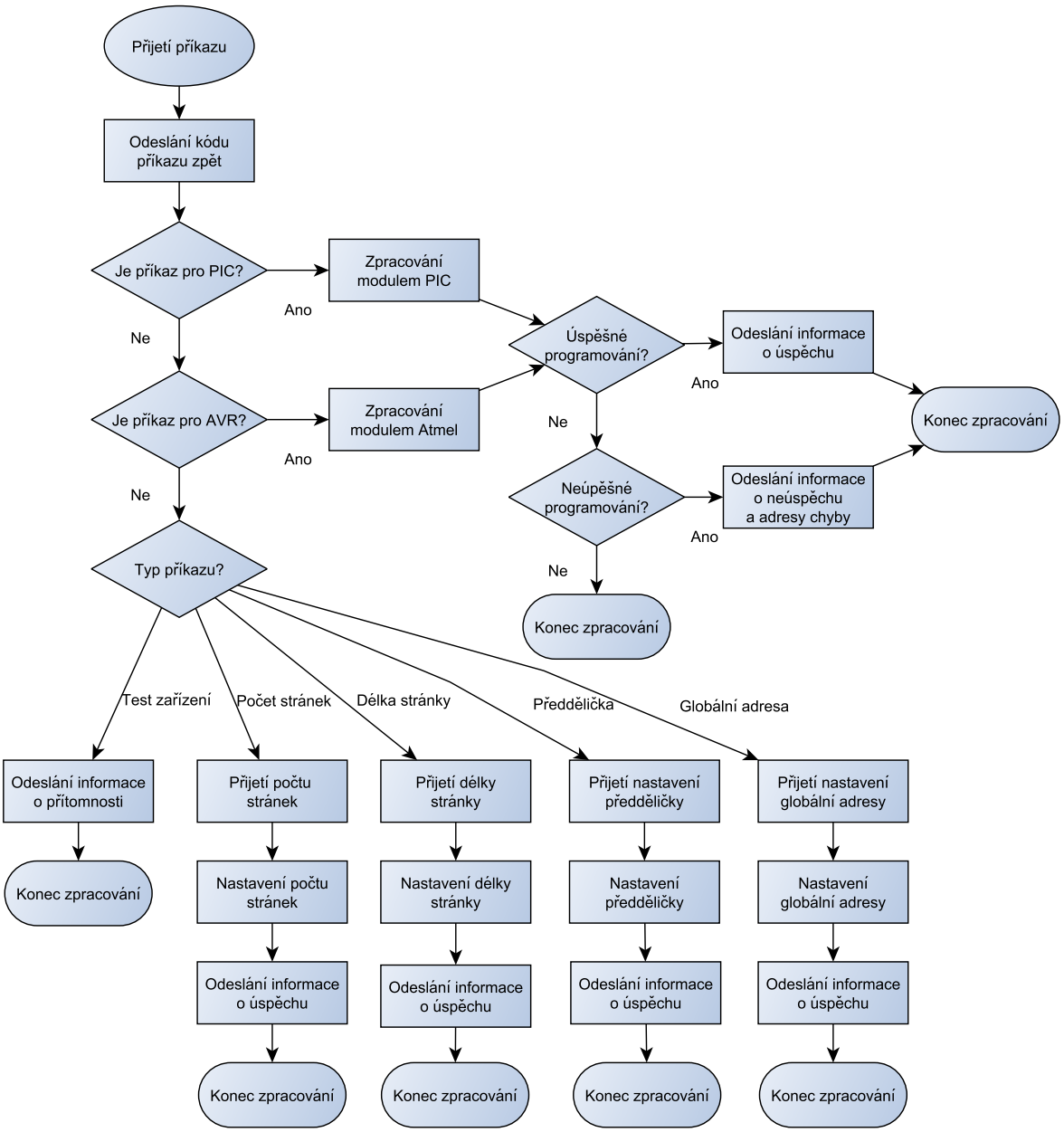
Bit	Název	Popis
23 - 18 (6 bit)	CFG_ERASE_TIME	Doba mazání čipu / řádku v jednotkách 1/2 milisekundy. Použije se pro všechny typy mazání.
17 - 12 (6 bit)	CFG_PROG_TIME	Čas programování v jednotkách 1/2 milisekundy. Použije se pro všechny typy programování.
11 - 9 (3 bit)	CFG_BLOCK_LENGTH	Délka bloku k zápisu (kolik slov se zapíše před zahájením programování). Spočítá se podle vzorce $n = 2 * *x$, kde x je zadaná hodnota.
8	CFG_USE_ALTERNATE_PROGEND	Použije při programování alternativní příkaz ukončení programování (0x06 místo 0x17).
7	CFG_VERIFY	Provést verifikaci okamžitě po zápisu (vhodné pouze pokud je délka bloku 0).
6	CFG_USE_INTERNAL_TIMING	Použít interní časování při zápisu.
5	CFG_ROW_ERASE	Smazat řádek před zápisem (32 slov).
4	CFG_USE_DATAMEM	Použít datovou paměť při programování/vyčítání.
3	CFG_USE_PRELOAD	Před programováním použít jeden zápis navíc (potřeba, pokud má mikrokontrolér i datovou paměť).
2	CFG_USE_MCHP	Odeslat při zahájení programování „MCHP“.
1	CFG_PGM_SETUP	Výběr hodnoty PGM v programovacím módu.
0	CFG_MCLR_SETUP	Výběr hodnoty $\overline{\text{MCLR}}$ v programovacím módu.

Tabulka D.7: Seznam bitů při konfiguraci PIC (3 byte, 0xD7)

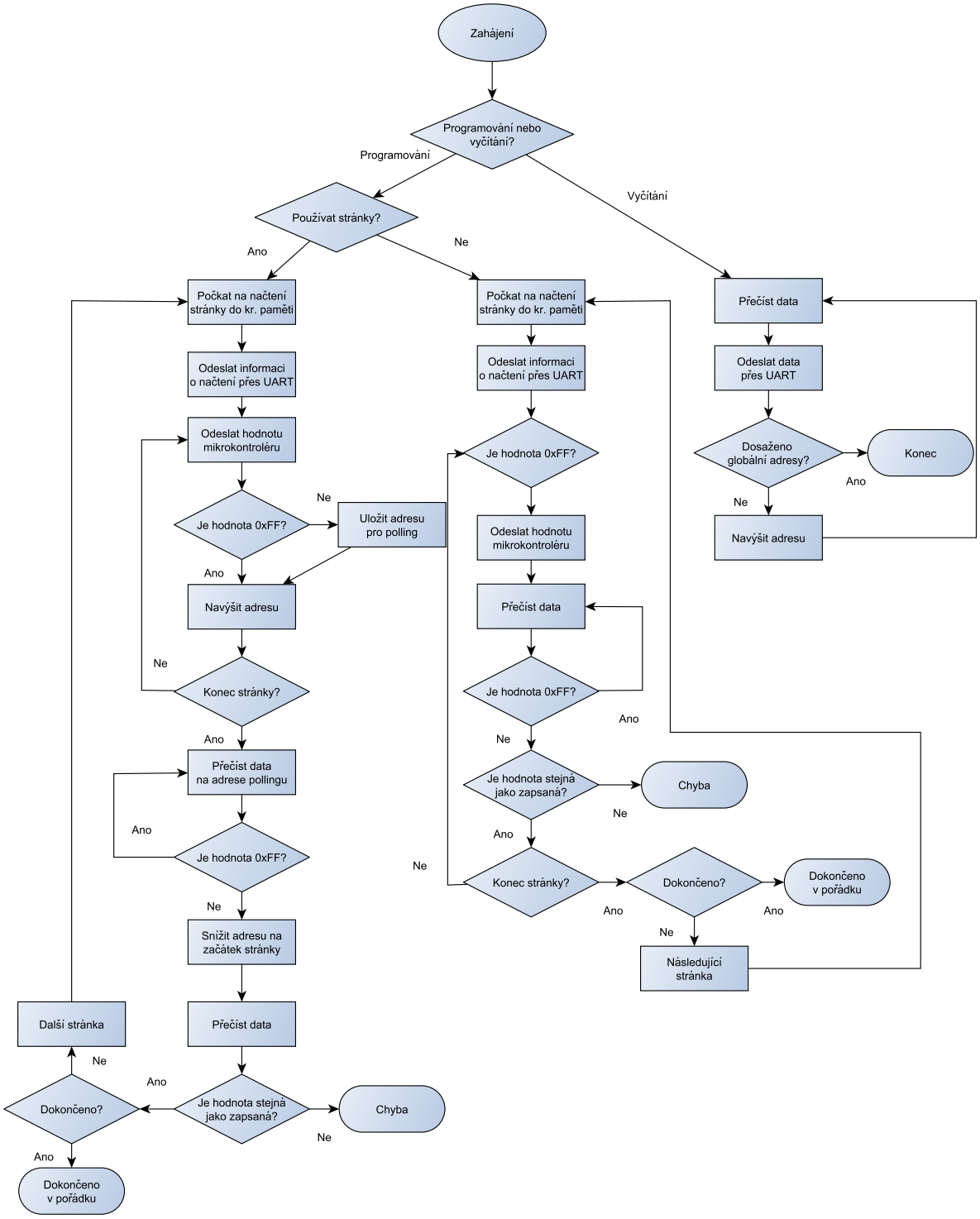
Příloha E

Konečné stavové stroje

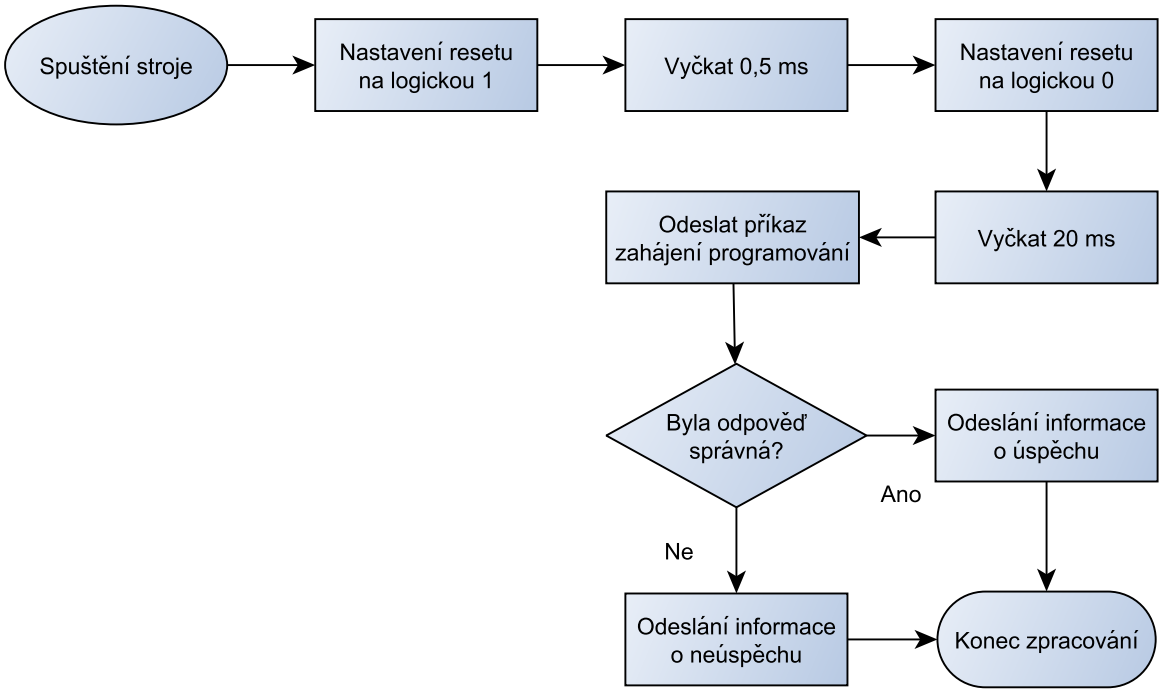
V této příloze se nachází grafy popisující stavové stroje jednotlivých modulů. Záměrně jsou vytvořeny jako vývojové diagramy a ne jako stavové diagramy. Vzhledem ke komplikovanosti stavových strojů by stavové diagramy byly příliš nepřehledné. Jeden prvek grafu tedy může obsahovat více skutečných stavů stavových strojů, nebo naopak, jeden stav může být popsán více prvky grafu.



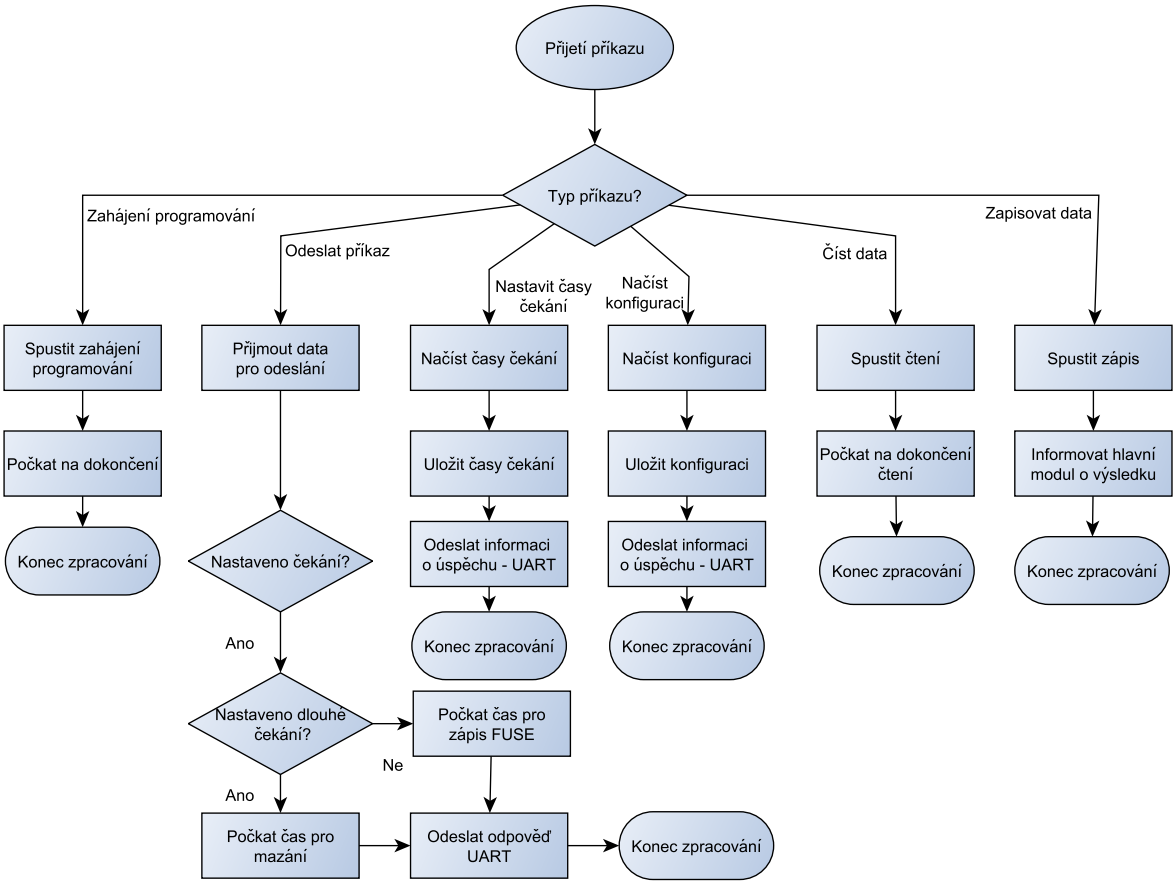
Obrázek E.1: Stavový stroj hlavního modulu programátoru



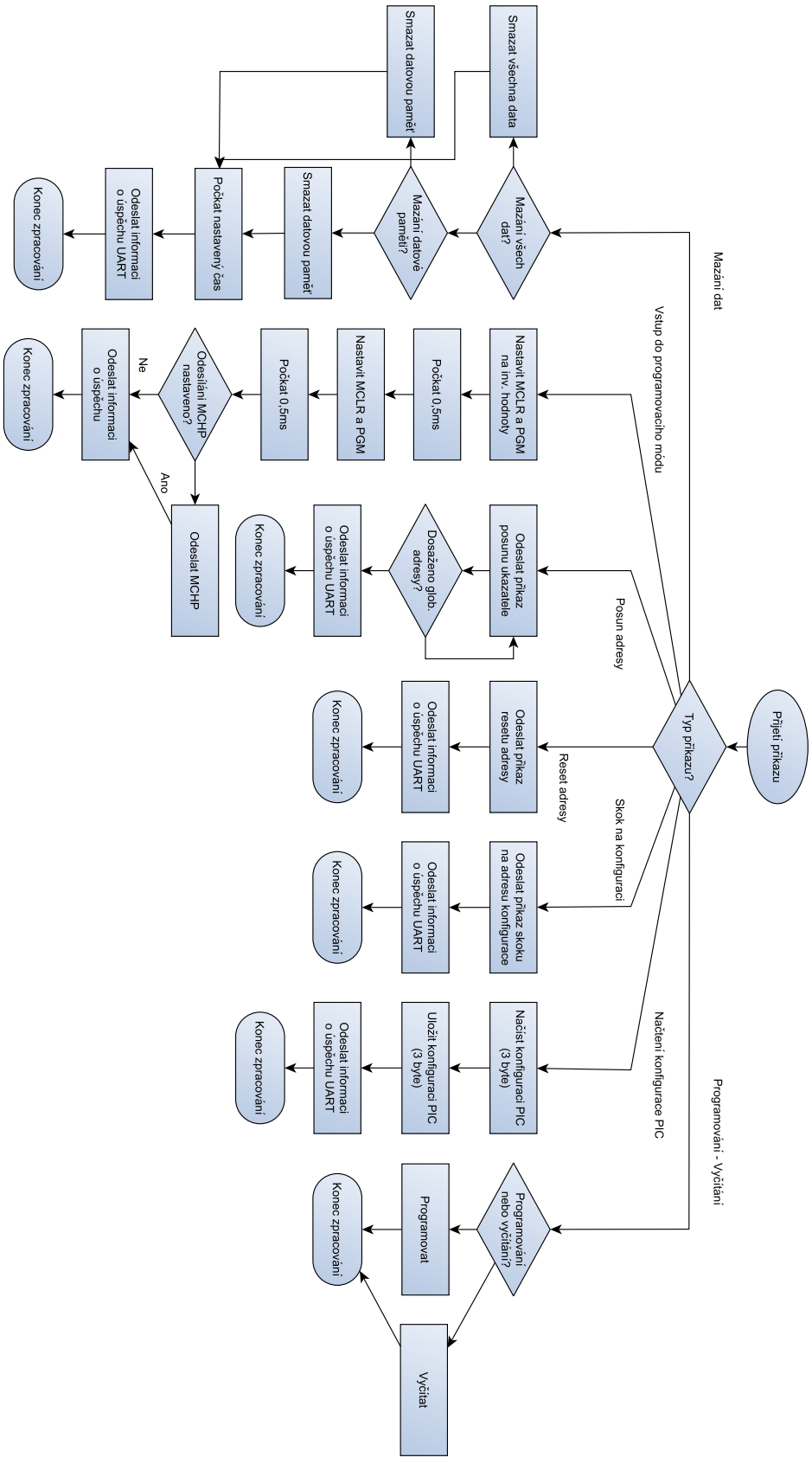
Obrázek E.2: Stavový stroj programování a vyčítání Atmel



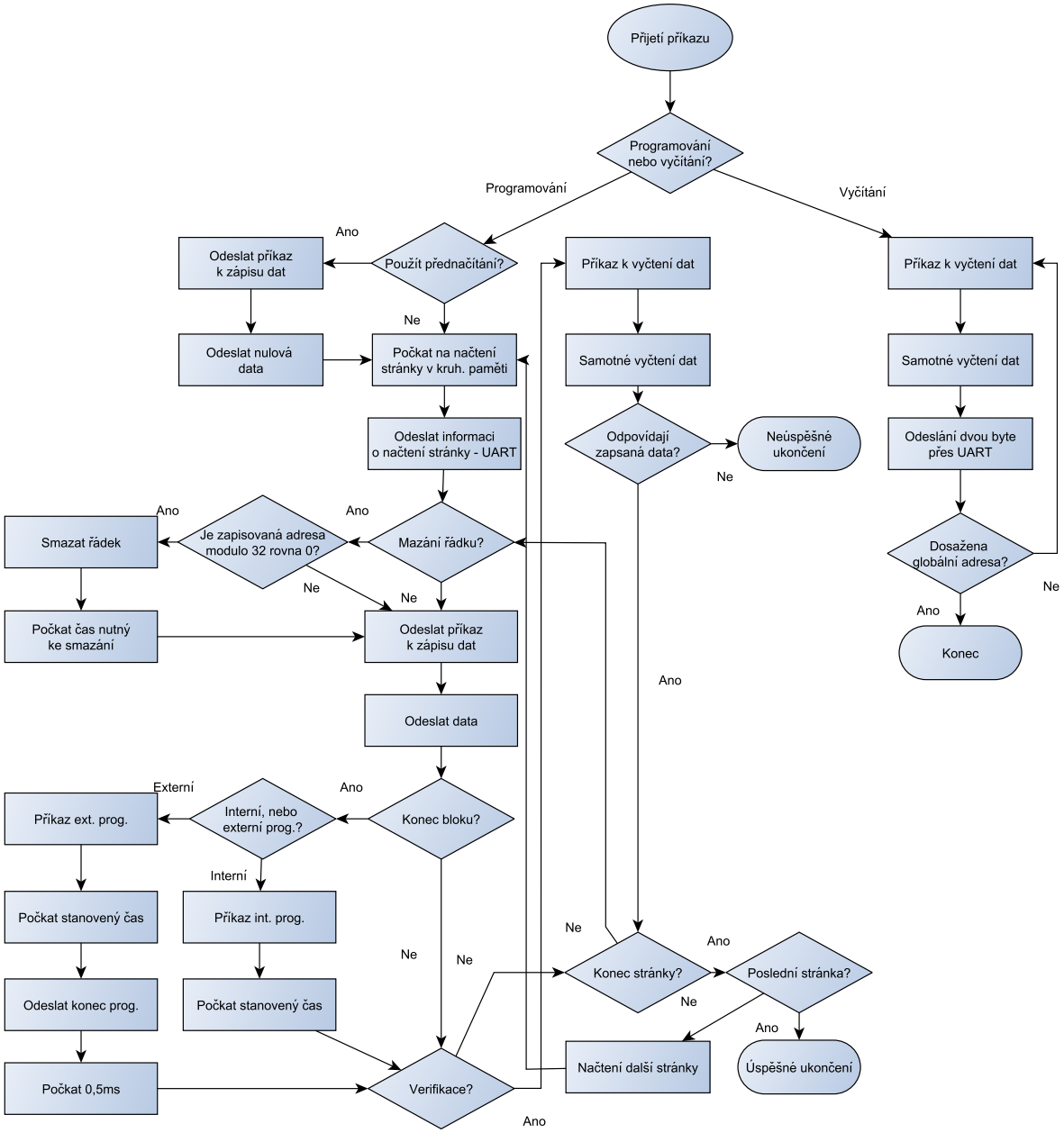
Obrázek E.3: Stavový stroj vstupu do programovacího módu Atmel



Obrázek E.4: Hlavní stavový stroj modulu Atmel



Obrázek E.5: Hlavní stavový stroj modulu PIC



Obrázek E.6: Stavový stroj programování a vyčítání PIC

Příloha F

Obsah CD

Obsah přiloženého CD je následující, rozepsán je dle adresářů:

- `/docs/` — obsahuje text této práce
- `/docs/src/` — obsahuje zdrojové kódy ve formátu LaTeXpro tuto práci
- `/vhdl/` — obsahuje zdrojové kódy VHDL pro FPGA jako projekt pro QDevKit, prosím čtěte *README* soubor v tomto adresáři
- `/app/src/` — obsahuje zdrojové kódy ovládací aplikace v jazyce Python. Prosím čtěte *README* soubor v tomto adresáři
- `/app/bin/` — obsahuje přeloženou ovládací aplikaci spustitelnou pod Windows