ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROMECHANIKY A VÝKONOVÉ ELEKTRONIKY

BAKALÁŘSKÁ PRÁCE

Univerzální bootloader pro mikrokontroléry Atmel AVR

Petr Rösch **2015**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr RÖSCH**
Osobní číslo:	**E13B0058K**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Název tématu:	**Univerzální bootloader pro mikrokontroléry Atmel AVR**
Zadávající katedra:	**Katedra elektromechaniky a výkonové elektroniky**

Zásady pro vypracování:

1. Prostudujte princip bootloaderu.
2. Seznamte se s podporovanými typy mikrokontrolérů Atmel AVR.
3. Navrhněte a realizujte bootloader s těmito parametry: a) možnost volby sériového rozhraní: USART, IIC, SPI; b) možnost broadcast programování v režimu IIC.
4. Navrhněte univerzální aplikaci pro PC, která bude umožňovat prostřednictvím vhodného převodníku nahrání kódu do FLASH i EEPROM paměti v mikrokontroléru.
5. Vytvořené technické a programové prostředky podrobně popište.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **20 - 30 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Pinker, J.: Mikroprocesory a mikropočítače.**
2. **Burkhard, M.: C pro mikrokontroléry.**
3. **AVR109: Self Programming (Aplikační list).**
4. **Elektronické informační zdroje.**

Vedoucí bakalářské práce: **Ing. Jaroslav Freisleben**
Regionální inovační centrum elektrotechniky

Datum zadání bakalářské práce: **15. října 2014**

Termín odevzdání bakalářské práce: **8. června 2015**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

V Plzni dne 15. října 2014

Abstrakt

Cílem této práce je vysvětlení pojmu zavaděč, jeho základního principu a využití v různých zařízeních. Dále je podrobně rozebrána jeho funkce v mikrokontrolérech Atmel AVR z hlediska spouštění, činnosti a organizace paměťového prostoru. Ovládání a přenos dat je zajištěn pomocí volitelného sériového rozhraní. Pro tento účel jsou ze softwarového pohledu představeny sběrnice USART, SPI a TWI. Za pomoci uvedených prostředků je realizován zavaděč pro zvolený mikrokontrolér Atmel ATmega8. Jeho předností je možnost práce s vnitřní pamětí FLASH i EEPROM pomocí kterékoliv ze tří popsaných sériových sběrnic. Výjimečnou vlastností je práce v režimu umožňujícím použití zavaděče jako mostu mezi rozhraním USART a SPI, nebo mezi USART a TWI. Na TWI lze připojit více mikrokontrolérů najednou a nahrávat jejich paměti v režimu broadcast. Zavaděč je vyvíjen v jazyce C a laděn na demonstračním plošném spoji osazeném čipem ATmega8 s převodníkem pro USB port. Ovládání je prováděno počítačem prostřednictvím speciální aplikace napsané ve vývojovém prostředí Delphi 7. Výsledkem práce je funkční systém, který dosáhl požadovaných parametrů.

Klíčová slova

Zavaděč, mikrokontrolér Atmel AVR, USART, I2C, TWI, SPI, CodeVisionAVR, Delphi 7, Broadcast, Paralelní programování, Sebeprogramování

Abstract

The aim of this work is to explain concept of bootloader, its basic principal and application in different devices. Then there is detailed analysis of its function in Atmel AVR microcontrollers in viewpoint of launching, operation and memory space organization. Controlling and data transmission is provided by suitable serial interface. For this purpose there is software introduction of USART, SPI and TWI buses. By using these resources and selecting Atmel ATmega8 microcontroller the bootloader is implemented. Its advantage is possibility of work with both FLASH and EEPROM internal memories by select any of three serial buses described. Exceptional quality is operation mode allowing bootloader to convert USART to SPI or USART to TWI. It is possible to connect more devices to TWI bus for broadcast memory work. The bootloader is developed in C language and debugged on demonstration PCB with ATmega8 chip and USB converter mounted. Data handling is performed by computer with special application running. Development environment Delphi 7 was used to write program. The result of this work is functional system that reached required parameters.

Keywords

Bootloader, microcontroller Atmel AVR, USART, I2C, TWI, SPI, CodeVisionAVR, Delphi 7, Broadcast, Parallel Programming, Selfprogramming

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení §270 trestního zákona č.40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne ……..

……..

Podpis

Obsah

Seznam symbolů a zkratek

FLASH ..... Paměť obsahující programový kód

EEPROM..... Electrically Erasable Programmable Read Only Memory - Paměť sloužící jako úložiště aplikace

USART..... Universal Synchronous Asynchronous Receiver Transmitter - Rozhraní pro synchronní nebo asynchronní sériovou komunikaci

CAN ..... Controller Area Network - Sběrnice používaná v automobilech a průmyslu

BSL ..... Bootstrap Loader - Zavaděč

UART..... Universal Asynchronous Receiver Transmitter - Rozhraní pro asynchronní sériovou komunikaci

I2C..... Inter-Integrated Circuit - Sériová sběrnice vyvinutá firmou Philips

TWI ..... Two-wire Interface - Označení I2C používané firmou Atmel

SPI..... Serial Peripheral Interface - Sériová periferní sběrnice

OS ..... Operační Systém

ROM..... Read Only Memory - Paměť pouze pro čtení

USB..... Universal Serial Bus - Univerzální sériová sběrnice

RX..... Receive - Brána pro příjem dat

TX ..... Transmit - Brána pro odesílání dat

RTS ..... Request To Send - Požadavek k odeslání, brána rozhraní RS-232

LED..... Light-Emitting Diode - Dioda emitující světlo

SRAM ..... Static Random Access Memory - Statická paměť s náhodným přístupem

MOSI..... Master Out, Slave In - Brána rozhraní SPI

MISO..... Master In, Slave Out - Brána rozhraní SPI

SCK..... Serial Clock - Brána rozhraní SPI

SS ..... Slave Select - Brána rozhraní SPI

SDA..... Serial Data - Brána rozhraní TWI, I2C

SCL ..... Serial Clock - Brána rozhraní TWI, I2C

RWW ..... Read-While-Write - Sekce paměti FLASH

NRWW..... No Read-While-Write - Sekce paměti FLASH

DPS ..... Deska plošných spojů

Úvod

V první části se práce zabývá vysvětlením pojmu zavaděč, neboli bootloader. Jako příklad jsou uvedena zařízení, která tento systém mohou využívat.

Druhá část se věnuje podpoře bootloaderu u modelu Atmel ATmega8. Je vytvořena struktura programu zaváděcí sekce a navržen komunikační protokol. Dojde k zahájení vývoje ovládací aplikace, která je nakonec schopná inicializovat zavaděč pomocí sériového portu zasláním příkazu do zásobníku v paměti mikrokontroléru.

Obsahem třetí části je vypracování metod pro přesun pamětí FLASH i EEPROM po sběrnici USART. Data mohou být zobrazena a upravena v uživatelském rozhraní. Nahraný programový kód lze spustit příkazem pro skok do aplikační sekce.

Čtvrtá část řeší schopnost zavaděče přijímat příkazy po sběrnicích SPI a TWI. Dále uvádí možnost použít bootloader jako převodník z USART na TWI, nebo SPI.

Poslední část je zaměřena na návrh testovací DPS. Popsány jsou 2 osazovací varianty podle role, kterou deska zastupuje na příslušném rozhraní.

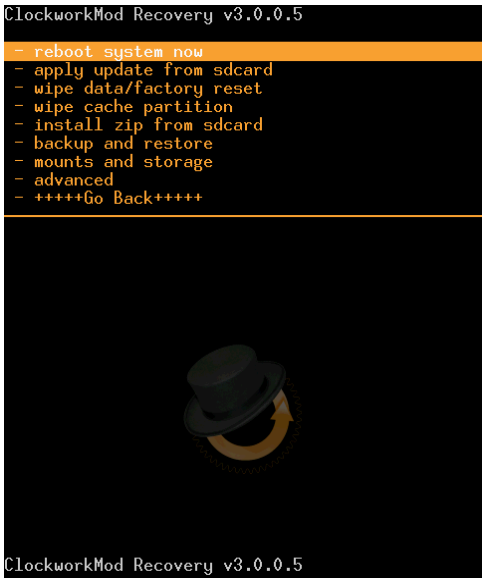
1 Princip bootloaderu

Hlavní myšlenkou je spuštění určitého programu po startu zařízení. Z hlediska paměťového prostoru je tento kód uložen mimo operační systém a není ani jeho součástí. Slouží obecně pro jeho zavedení. Celkový rozsah funkcí se odvíjí od typu přístroje.

1.1 Mobilní telefony

Velkou skupinu tvoří chytré telefony s operačním systémem Google Android. Originální zavaděč vykoná nahrání jádra operačního systému. Obvykle lze najít nástroje pro testování hardwarových vstupů a výstupů, nebo diagnostiku při selhání systému. Alternativní bootloader (obrázek 1.1) přináší možnost číst a psát do paměti Read Only Memory (ROM). Díky tomu lze systém zálohovat, nebo zaměnit. Zobrazení voleb je často možné stiskem několika kláves současně během zapínání [1].

Výrobce může povolit zákazníkovi aktualizaci. Ušetří tím náklady za reklamace způsobené vadným softwarem dodaném v zařízení. Dále dává možnost využít i neoficiální operační systém, který často má některé funkce navíc. Specialisté i amatéři v oboru mobilních telefonů se předhání s modifikacemi ROM pamětí pro chytré telefony. V odvětví spotřební elektroniky použití zavaděče usnadní vývoj na dané platformě.



Obrázek 1.1: ClockworkMod Recovery pro chytré telefony [1]

1.2 Mikrokontroléry

Existuje celá řada mikrokontrolérů s podporou zavaděče. Pro výrobce Microchip byl nalezen Tiny PIC bootloader. Pracuje s modely řady PIC16F, PIC18F a dsPIC. Po resetu čeká sekundu na zprávu z počítače. Pokud ji nedostane, spustí aplikaci. Je schopný nahrát program do paměti FLASH. Jeho výhodou je malá velikost (100 slov) [2]. Mezi další nalezené projekty patří WLoader, ZPL nebo PICLOADER.

Firma STMicroelectronics během výroby procesorů řady STM32 nahrává vlastní bootloader podporující více rozhraní pro přenos aplikačního programu. Je možné využít USART (Universal Synchronous and Asynchronous Receiver Transmitter), CAN (Controller Area Network), USB (Universal Serial Bus), I2C (Inter-Integrated Circuit) nebo SPI (Serial Peripheral Interface). Pro každé z nich je definovaný komunikační protokol [3].

Texas Instruments uvádí možnost využít BSL (Bootstrap Loader) u svých mikrokontrolérů MSP430. Zajišťuje komunikaci s vestavěnou pamětí a nachází se v chráněném úložišti. K dispozici jsou komunikační protokoly pro rozhraní UART (Universal Asynchronous Receiver Transmitter), USB nebo I2C. Přístup k paměti může být chráněn uživatelským heslem [4]. MSP430F5xx a MSP430F6xx mají BLS umístěn v paměti FLASH a je možné ho změnit. Jiné rodiny (MSP430G2xx a MSP430FR5xx) používají pro uložení BLS paměť ROM a zároveň podporují pouze UART. Pokud je potřeba u těchto modelů použít jinou sběrnici, lze do hlavní paměti nahrát alternativní bootloader MSPBoot. Schopnost práce s pamětí je tak doplněna o I2C, SMBUS, SPI a jiné [5].

U procesorů Freescale je využíván bootloader Kinetis. U modelů řady K02_100, K22_100, K22_120, K24_120 a KV3x je nahrán v paměti FLASH. KL0, KL1x, KL2x, KL3x a KL4x mají zavaděč předprogramovaný v ROM. Jeho funkce jsou podobné jako u ostatních výrobců [6].

Atmel podporuje použití zavaděče v rozsahu celé rodiny AVR. Využíván je protokol STK500 nazvaný podle stejnojmenné vývojové sady [7]. Samotný procesor zajistí obousměrný přenos programového kódu. Umožňuje vytvářet aplikace, které dokážou sami sebe aktualizovat pomocí zavaděče. Adresový prostor paměti FLASH (paměť obsahující programový kód) je rozdělen na 2 sekce [8]:

- ***Aplikační sekce***

  Obsahuje programový kód aplikace reprezentující činnost zařízení

- ***Zaváděcí sekce***

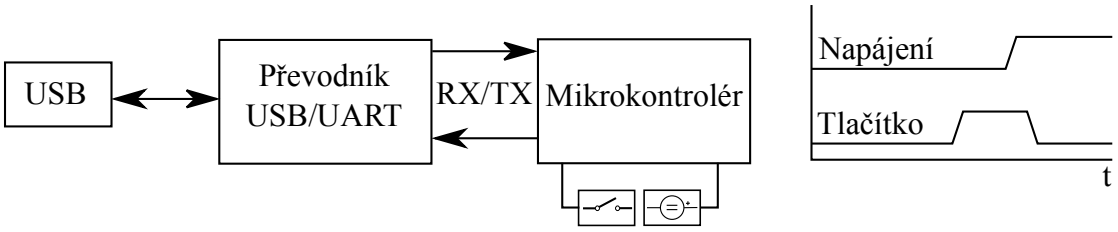
  Sekce obsahující kód pro nahrávání programu z/do aplikační sekce

Správná kombinace konfiguračních bitů zajistí, že se po startu spustí program na adrese zavaděče. Kód následně definuje podmínku, kterou je potřeba splnit pro setrvání v zaváděcí sekci. V případě nesplnění dojde ke spuštění aplikace [7].

Zařízení je připojeno k PC pomocí USB portu. Pokud mikrokontrolér nepodporuje USB, je nutné použít převodník na UART. Ten je vybaven piny RX (Receive - Brána pro příjem dat) a TX (Transmit - Brána pro odesílání dat), které lze použít pro komunikaci se širokou škálou procesorů. Na základě úvahy byly namodelovány 2 způsoby spuštění bootloaderu.

- ***Přidržením tlačítka během připojení napájení***

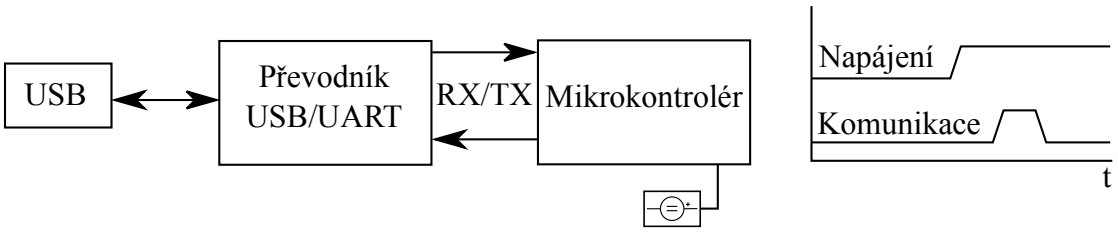
Tlačítko připojené k libovolného digitálnímu vstupu musí být sepnuto v okamžik připojení napájení. Program na začátku přečte stav a vyvolá podle toho bootloader nebo aplikační sekci.



Obrázek 1.2: Spuštění zavaděče přidržením tlačítka během připojení napájení

- ***Zahájením komunikace po připojení napájení***

Na začátku program určitý okamžik, například 3 sekundy, naslouchá příchozí komunikaci. Pokud v tomto časovém intervalu přijme příslušný povel, dojde k vyvolání zaváděcí sekce. Po vypršení doby však dojde ke spuštění aplikace.

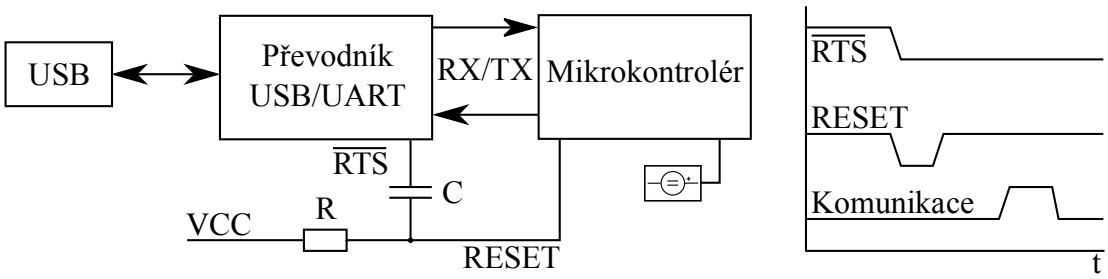


Obrázek 1.3: Spuštění zavaděče zahájením komunikace po připojení napájení

Výše uvedené způsoby jsou velmi jednoduché, ale vyžadují jistou pozornost při provádění jednotlivých kroků. Obsluha musí znát způsob aktivace bootloaderu. Při konzultaci s vedoucím práce proběhla diskuze o další variantě využívané na vývojové platformě Arduino. Ta se zdá být uživatelsky nejjednodušší.

- ***Resetováním a zahájením komunikace***

Výstup RTS (Request To Send - Požadavek k odeslání) převodníku USB/UART je přes kondenzátor připojen na reset vstup mikrokontroléru. Při otevření sériového portu dojde k resetování a není nutné mačkat tlačítko při nahrání nového kódu [9].



Obrázek 1.4: Spuštění zavaděče resetem a komunikací

2 Návrh bootloaderu a jeho propojení s PC

Mikrokontrolér by měl být schopný v režimu zavaděče komunikovat po zvoleném rozhraní. Nejjednodušší přenos dat je po sběrnici USART. Existuje pro ni levný a dostupný USB převodník od FTDI (FT230x). Navíc je možné využít RTS výstup k resetování mikrokontroléru. Z těchto důvodů je zamýšleno rozhraní využít pro připojení k počítači.

Dalším předpokládaným prostředkem pro nahrání kódu je sběrnice TWI (Two-wire Interface - Označení používá Atmel, protože I2C je registrovanou značkou firmy Philips). Zde se vytrácí rovnocennost jednotlivých připojených obvodů. Zařízení na sběrnici může zastupovat roli master nebo slave. Procesor, který přijímá programový kód k účelu jeho naprogramování bude vystupovat jako slave. Vzniká potřeba převodu komunikace z PC na TWI master. Bylo usouzeno, že má smysl vyzkoušet samotnému zavaděči přidat funkci převodníku z USART na TWI. Dalším úskalím je adresace. Není nutné, aby každý slave měl svou adresu. V případě broadcast (adresa 0x00) vyslechnou povel všichni účastníci. Problém však nastává při kontrole zapsaných dat. Vyčítání je možné provádět pouze jednotlivě a každý účastník musí být identifikovatelný svou adresou, kterou by mohl mít uloženou například v paměti EEPROM. Ta však slouží jako úložiště aplikace a hrozí, že ji bude potřebovat využít v plném rozsahu. Vhodné místo pro adresu se stává paměť FLASH. Vznikne tím nutnost více souborů hex (pro každou adresu zvlášť). Tento problém lze jednoduše vyřešit. Zaváděcí sekce se bude nahrávat pouze přes ovládací aplikaci, která otevře originální zkompilovaný soubor, upraví v něm adresu a až potom nahraje pomocí programátoru.

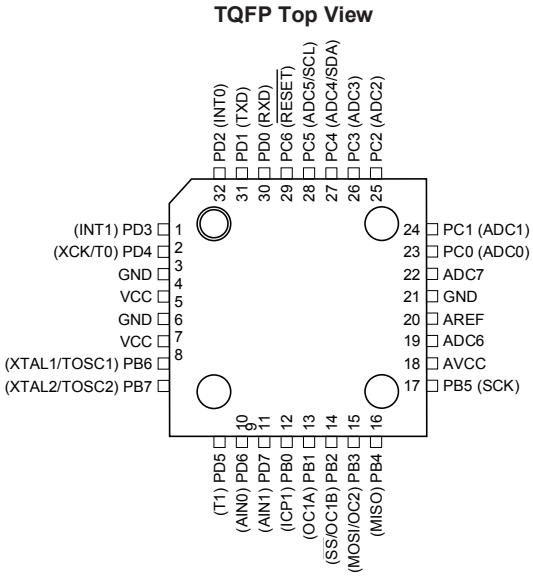
I v případě sběrnice SPI vzniká potřeba převodníku. Princip řešení je naplánován stejně jako u TWI. Navíc se nemusí řešit adresování, protože není předpokládáno více slave zařízení na SPI.

2.1 Výběr softwarového a hardwarového vybavení

Úkolem je zvolit takové prostředky, aby bylo možné realizovat bootloader s podporou rozhraní USART, SPI a TWI včetně broadcast programování. Ovládací aplikace by měla být vyvíjena v prostředí, které umožní pomocí vhodného převodníku nahrávat data.

2.1.1 Výběr mikrokontroléru

Na základě zadání práce je potřeba zvolit mikrokontrolér podporující zavaděč a uvedené sběrnice. Jedná se o rozhraní USART, nebo její asynchronní variantu označenou jako UART. Další je TWI a poslední je SPI. Pro indikaci činnosti je potřeba několik volných vstupně/výstupních pinů pro LED diody (Light-Emitting Diode - Dioda emitující světlo). Užitečný může být reset vstup, který umožní spustit program od začátku bez odpojení napájení. Po konzultaci s vedoucím byl jako vhodný mikrokontrolér vybrán Atmel ATmega8. Podle uspořádání pinů znázorněném na obrázku 2.1 je až 12 vstupně výstupních pinů volně použitelných. Ostatní jsou rezervované pro sběrnice, krystalový oscilátor a reset. Katalogový list dále uvádí možnost využití interního i externího přerušení. Zavaděč má nastavitelnou velikost (256, 512, 1024 nebo 2048 bajtů) a může upravovat aplikační sekci paměti [10].



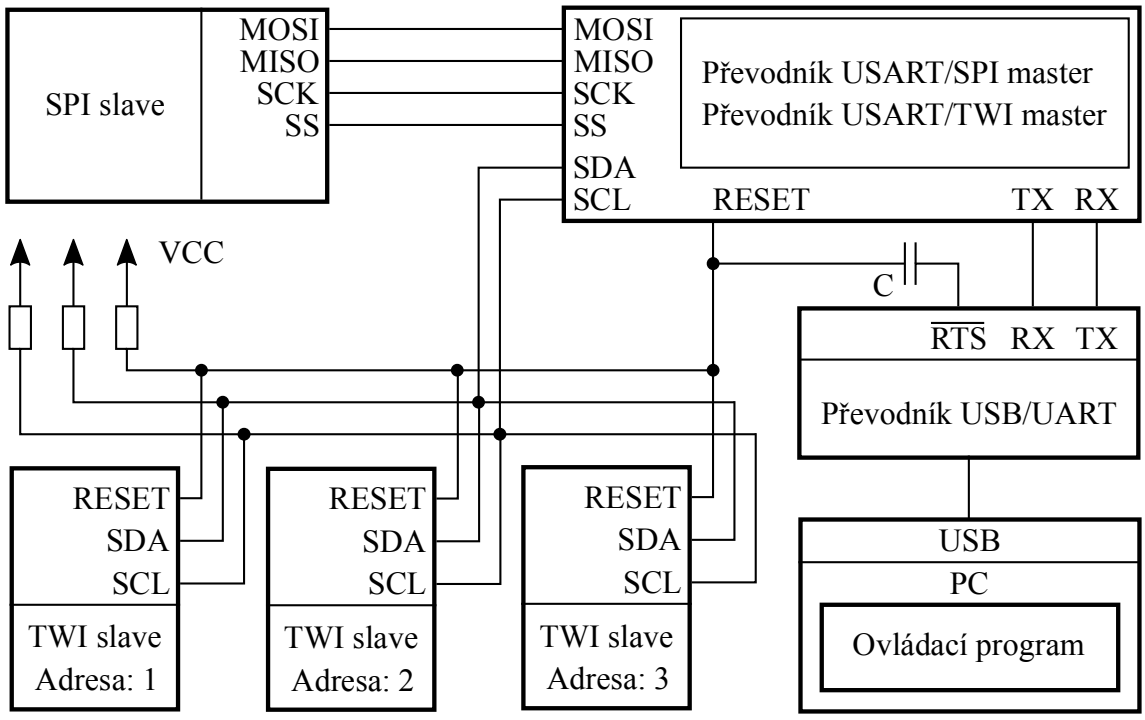
Obrázek 2.1: Mikrokontrolér Atmel ATmega8 [10]

2.1.2 Výběr vývojového prostředí pro mikrokontrolér

Na základě dřívějších zkušeností byl jako vývojové prostředí vybrán CodeVisionAVR od společnosti HP InfoTech. Velkou výhodou je schopnost automaticky vygenerovat programový kód, který zahrnuje nastavení registrů, vytvoření podpůrných funkcí a rutin přerušení. Na úrovni (programovacího) jazyka C lze dokonce přiřadit proměnným přesnou adresu v paměti, což umožňuje jejich sdílení s úrovní strojového kódu.

2.1.3 Výběr vývojového prostředí pro ovládací software

Vývoj aplikací pro operační systém Windows je možný například v Microsoft Visual Studio. Univerzita navíc poskytuje licenci pro studijní použití. Bohužel dosud nebyly s tímto prostředím získány žádné vlastní zkušenosti. Vzhledem ke dřívějším zkušenostem byl jako optimální vývojový nástroj zvolen Borland Delphi 7, který je ve variantě personal zdarma. Pro moji aplikaci je tato verze naprosto dostačující, neboť umožňuje vytvářet grafické formuláře, pracovat se soubory a ovládat sériovou komunikaci.



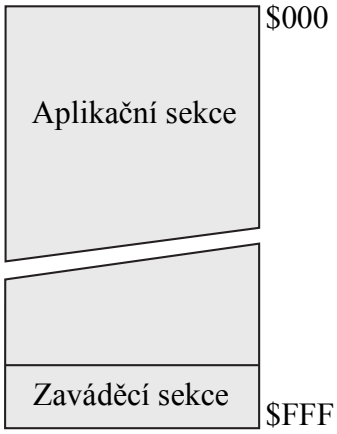
Obrázek 2.2: Schéma propojení

2.2 Rozdělení paměti FLASH

Kapacita paměti mikrokontroléru ATmega8 je 8Kb. Velikost slova na jedné adrese je 16 bitů. Prostor je rozdělen na aplikační a zaváděcí sekci. Velikost sekcí závisí na naprogramování pojistek BOOTSZ1 a BOOTSZ0 podle tabulky 2.1.

Tabulka 2.1: Rozdělení paměťového prostoru [10]

BOOTSZ1	BOOTSZ0	Paměť zavaděče	Adresa aplikační sekce		Adresa zaváděcí sekce	
			Začátek	Konec	Začátek	Konec
1	1	128 slov	0x000	0xF7F	0xF80	0xFFF
1	0	256 slov		0xEFF	0xF00	
0	1	512 slov		0xDFF	0xE00	
0	0	1024 slov		0xBFF	0xC00	



Obrázek 2.3: Rozdělení paměťového prostoru [10]

Paměť FLASH je také rozdělena na 2 sekce podle možnosti současného zápisu a čtení. Jejich velikosti není možné nastavit [10].

- ***RWW (Read-While-Write) sekce***

Během zápisu, nebo mazání dat v této sekci je možné číst a provádět programový kód v NRWW sekci.

- ***NRWW (No Read-While-Write) sekce***

Při provádění změn v této sekci paměti je procesor zastaven a není možný běh programu, dokud není operace s pamětí dokončena

Tabulka 2.2: Rozdělení paměti na RWW a NRWW sekce [10]

Sekce	Počáteční adresa	Konečná adresa
Read-While-Write	0x000	0xBFF
No Read-While-Write	0xC00	0xFFF

2.3 Spuštění zaváděcí sekce

Z katalogového listu bylo zjištěno, že počáteční adresa při startu mikrokontroléru je závislá na naprogramování pojistky BOOTRST [10].

Tabulka 2.3: Možnosti naprogramování pojistky BOOTRST [10]

BOOTRST	Adresa po resetu
1	0x000
0	Začátek bootloader sekce podle pojistek BOOTSZ0, BOOTSZ1

Na nepájivém poli byl sestaven jednoduchý obvod se vzorkem ATmega8. K jeho vývodům XTAL1 a XTAL2 byl připojen krystalový oscilátor spolu s kondenzátory podle doporučení výrobce [10]. Pro možnost nahrávání paměti byl vyveden kabel zakončený konektorem PFL10. Ten zajistil propojení s programátorem a napájení. Nakonec proběhla volba vhodné indikace. K volným vstupně výstupním pinům bylo připojeno 8 LED diod. Obvod byl následně připojen k programátoru. Přečtením signatury čipu byla ověřena správná funkce komunikačního rozhraní. V prostředí CodeVisionAVR byl založen nový projekt. V něm bylo za pomoci časovače vytvořeno jednoduché blikání LED diody.

Před nahráním úspěšně zkompilovaného kódu bylo potřeba správně naprogramovat pojistky. Veškeré hodnoty je možné získat studií katalogového listu.

- ***CKSEL0 = 0, CKSEL1..3 = 1***

  Aplikuje použití krystalového oscilátoru o frekvenci 3-8 MHz [10].

- ***BOOTRST = 0, BOOTSZ0..1 = 0***

  Program po připojení napájení nebo resetu začne na adrese zavaděče (viz tabulka 2.3). Velikost zaváděcí sekce bude 1024 slov podle tabulky 2.1.

Více informací o práci s pamětí v režimu zavaděče bude uvedeno v kapitolách 4.7 až 4.10. Nejprve je nutné oživit komunikaci, aby bylo možné zapisovat a číst aplikační sekci. Možným řešením by bylo generování programového kódu uvnitř mikrokontroléru. Zapsaná data by pak mohla být zpětně vyčtena programátorem.

2.4 Návrh komunikačního protokolu

Návrh předchází shrnutí všech možných operací, které by měl bootloader schopný provést. Ke každému úkonu je třeba po příslušné sběrnici dodat množinu informací. Její součástí mohou být i data programového kódu, pokud se jedná o zápis paměti. Tento balík informací bude dále označen jako příkaz.

Mikrokontrolér se musí na základě přijatých dat rozhodnout, jestli má být příslušná operace provedena na jeho vlastní aplikační sekci, nebo bude v roli převodníku požadovat o vykonání jednotku připojenou na sběrnici SPI, nebo TWI. Zde se projevuje potřeba znalosti cílové adresy, která může být navíc nulová pro broadcast požadavek. Tato informace bude vyžadovat rezervaci dvou bajtů uvnitř příkazu. První bude obsahovat kód cílového zařízení a druhý případnou adresu.

Jádrem příkazu je navolení požadovaného úkonu s pamětí. Jedná se o zápis a čtení EEPROM, nebo aplikace uvnitř paměti FLASH. Tyto zásahy by bylo vhodné orámovat inicializací zaváděcí sekce a spuštěním aplikačního kódu. Celkově vyplynulo 6 možností pro zvolenou operaci. Bude stačit použití jednoho bajtu.

Práce s některou z pamětí může být vykonána v určitém adresovém rozsahu. Užitečné tedy bude předat údaj o počáteční a koncové adrese. Proto bylo nutné prozkoumat systém adresování. Udávaná kapacita FLASH je 4096 slov, nebo také 8192 bajtů [10]. To znamená, že velikost jednoho slova jsou 2 bajty. EEPROM má velikost 512 bajtů a její adresa je reprezentována devíti bity [10]. Ani jedné z pamětí nestačí samostatný bajt pro uložení adresy. Bude ji tedy symbolizovat dvojice bajtů resp. čtveřice bajtů ponese rozsah paměťového prostoru, nad kterým bude zvolená operace provedena.

Poslední přenášený prvek je samostatný obsah programového kódu. Jeho délka je definována počáteční a cílovou adresou. Zavaděč, který přijímá data, musí vědět předem počet bajtů zbývající do konce přenosu. Důsledkem je potřeba výpočtu na straně mikrokontroléru. Úspornější způsob je určení hodnoty v ovládací aplikaci a zaslání délky datového bloku uvnitř příkazu.

Po dokončení příslušného úkonu je očekávána odpověď. Jeden bajt bude signalizovat úspěšnost dokončení požadavku. Pokud bylo zvoleno čtení, budou následovat data o délce dané rozsahem adres. Použitý návrh příkazu a odpovědi lze najít v tabulce 2.4 a 2.5.

Tabulka 2.4: Navržený příkaz

Bajt	Význam	
0	Velikost datového bloku příkazu v bajtech. Pro čtení paměti má nulovou hodnotu.	
1		
2	Velikost datového bloku očekávané odpovědi v bajtech. Pro zápis paměti má nulovou hodnotu.	
3		
4	Počáteční adresa paměti.	
5		
6	Koncová adresa paměti.	
7		
8	Zvolená operace dle hodnoty.	
	0x64	Čtení FLASH
	0x65	Čtení EEPROM
	0x66	Zápis FLASH
	0x67	Zápis EEPROM
	0x68	Spuštění aplikační sekce
	Ostatní	Inicializace zaváděcí sekce
9	Zvolené cílové zařízení dle hodnoty.	
	0x01	Provést požadavek na své vlastní aplikační sekci
	0x02	Přeposlat požadavek přes sběrnici SPI
	0x03	Přeposlat požadavek přes sběrnici TWI
10	TWI adresa cílového zařízení.	
11	Rezervováno pro další vývoj.	
12+	Přenášený programový kód.	

Tabulka 2.5: Odpověď

Bajt	Význam	
0	Výsledek operace dle hodnoty.	
	0x14	Úspěšné dokončení
	Ostatní	Chyba
1-3	Rezervováno pro další vývoj.	
4+	Přenášený programový kód.	

2.5 Automatický skok do aplikační sekce

Ve zdrojovém kódu pro Atmel došlo k definici čtyř indikačních LED diod. Jejich význam byl diskutován s vedoucím práce. Výsledná podoba indikace je definována v tabulce 2.6.

Tabulka 2.6: LED indikace

Symbol	Port	Stav	Význam
LEDB	PORTC.0	Bliká	Po zapnutí. Okamžik, kdy lze vyvolat zaváděcí sekci.
		Svítí	Zaváděcí sekce je vyvolána.
		Nesvítí	Program se nachází v aplikační sekci.
LEDU	PORTC.1	Svítí	Zavaděč byl inicializován prostřednictvím sběrnice USART.
LEDI	PORTC.2	Svítí	Zavaděč byl inicializován přes sběrnici TWI.
		Blikne	Přeposlání příkazu po rozhraní TWI.
LEDS	PORTC.3	Svítí	Zavaděč byl inicializován přes sběrnici SPI .
		Blikne	Přeposlání příkazu po rozhraní SPI.

Jednotlivé porty bylo potřeba nastavit jako digitální výstupy. Registru DDRC je na začátku programu přiřazena hodnota 0x0F, protože logická jednička znamená výstup a spodní 4 bity ovládají směr nultého až třetího pinu. Není třeba dbát na horní 4 bity, protože jejich funkce bude pod kontrolou TWI sběrnice a obvodu resetu [10].

Po připojení napájení je možné aktivovat zaváděcí sekci. Tento stav je indikován blikáním diody LEDB. Po vypršení času určeného pro setrvání v zaváděcí sekci dojde ke skoku do aplikační sekce. Blikání i spuštění aplikace bylo realizováno za pomoci rutin přerušení časovačů TIMER0 a TIMER1. Osmibitový časovač TIMER0 je ovládaný registrem TCCR0. Zapsáním hodnoty 0x05 a předchozím výběrem krystalového oscilátoru (7,3728 MHz) je zajištěna čítací frekvence 7200 Hz. Rutina přerušení bude vyvolána jednou za 0,36 sekund a pokaždé invertuje stav diody LEDB. Předdělička šestnáctibitového čítače TIMER1 je nastavena registrem TCCR1B. Použitím hodnoty 0x04 je nastaveno čítání o frekvenci 28800 Hz. Rutina přerušení obsahuje skok do aplikační sekce a k jejímu vyvolání dojde za 2,28 sekundy od začátku programu. Přerušení obou časovačů bylo zapnuto zapsáním hodnoty 0x05 do registru TIMSK. Spouštění všech rutin přerušení je nutné globálně povolit pomocí instrukce *sei*. Zdrojový kód byl následně přeložen a nahrán do zaváděcí sekce mikrokontroléru. Na místo aplikace byl přidán testovací program spočívající v pomalém blikání dvojice LED diod. Hned po připojení napájení se rychle rozblikala LEDB. Po uplynutí přibližně dvou sekund zhasnula a blikání dvojice LED potvrdilo spuštění aplikace.

2.6 Příprava zásobníku pro přenos dat

ATmega8 disponuje 1024 bajty vnitřní pamětí SRAM. Pro podporu zásobníku bylo definováno pole s názvem „buffer" o velikosti 524 bajtů (12 pro příkaz a 512 pro přenášený programový kód). Je předpokládáno, že pokud bude zavaděč vystupovat v roli převodníku, bude potřeba přenášet obsah po zvolené sběrnici. Vyskytla se potřeba dalších pomocných proměnných. Ty jsou popsány v tabulce 2.7.

Tabulka 2.7: Proměnné využívané zásobníkem

Typ	Název	Význam	
unsigned int	wr_index	Index zápisu. Ukládá pozici v zásobníku, kam bude uložen následující bajt.	
unsigned int	rd_index	Index čtení. Ukládá pozici v zásobníku, z které bude přečten následující bajt.	
unsigned char	buffer_action	Akce zásobníku. Význam podle hodnoty.	
		0x00	Žádná operace se zásobníkem neprobíhá.
		0x01	Příjem příkazu. Příkaz je přenášen do zásobníku po sběrnici, která vyvolala zaváděcí sekci.
		0x02	Odeslání odpovědi. Odpověď je odesílána zpět po sběrnici, která vyvolala zaváděcí sekci.
		0x03	Přeposlání odpovědi. Zavaděč vystupuje v roli převodníku a přijímá odpověď od cílového zařízení po zvolené sběrnici.
		0x04	Přeposlání příkazu. Zavaděč vystupuje v roli převodníku a odesílá příkaz do cílového zařízení zvolené sběrnici.
union	datalength	Délka dat. Počet plánovaných přenášených znaků. Ukládá součet velikostí ovládací a datové části příkazu, nebo odpovědi.	

Union umožnil definovat pro jednu proměnnou více datových typu o stejné velikosti. Např. datalength může být zpracován jako integer, nebo jako pole dvou bajtů. Pro zápis do zásobníku byla navržena funkce putchar. Datový typ jejího parametru je unsigned char. Předanou hodnotu zapíše do pole buffer na pozici wr_index. Proměnou wr_index je následně nutné inkrementovat, jinak by při každém dalším volání funkce docházelo k opakovanému přepisování stejného indexu. Analogicky je realizováno čtení. Funkce getchar vrátí hodnotu zásobníku vybranou podle proměnné rd_index. Ta musí být opět inkrementována.

2.7 Inicializace zavaděče po sběrnici USART

2.7.1 Mikrokontrolér

Cílem je úspěšně přenést příkaz z počítače do mikrokontroléru po sběrnici USART a získat správnou odpověď. Ke spojení byl použit standardní převodník USB/UART. Atmel připravený na nepájivém poli používá napájení z programátoru 5 V. Ze softwarového hlediska bylo nutné nastavit registry sběrnice USART. Všechny hodnoty byly vybírány na základě katalogového listu a vlastního posudku. Přesné znění se nachází v tabulce 2.8

Tabulka 2.8: Nastavení sběrnice USART

Registr	Hodnota	Význam
UCSRA	0x00	Horních 6 bitů signalizuje stav. Spodní 2 s tímto nastavením zakážou víceprocesorovou komunikaci a dvojnásobnou rychlost.
UCSRB	0xD8	Zapnutí příjmu a odesílání po sběrnici včetně povolení rutin přerušení vyvolaných dokončením příjmu a odesílání.
UCSRC	0xA6	Asynchronní režim, sudá parita, 1 stop bit a délka znaku 8 bitů (ta je navíc závislá na 2. bitu registru UCSRB).
UBRRH	0x00	Při použití krystalového oscilátoru 7,3728 MHz nastaví přenosovou rychlost 115200 bd.
UBRRL	0x03	

Dále byla vytvořena rutina přerušení vyvolaná příjmem po sběrnici USART. Ta přečtením registru UCSRA zkontroluje bezchybnost přenosu. Úspěšně přijatý znak vloží do zásobníku pomocí funkce putchar (dojde k inkrementaci wr_index). Předpokládá se, že hodnota proměnné buffer_action (akce zásobníku) je rovna 1 (příjem příkazu) a datalength nese naplánovaný počet příchozích znaků. Rovnost hodnot datalength a wr_index znamená dokončení přenosu celého zásobníku a vynuluje buffer_action.

Na počátku hlavní programové smyčky je vynulován index zápisu, na hodnotu 12 nastavena délka dat (podle příkazu navrženého protokolu) a akcí zásobníku se stává příjem příkazu. V této fázi lze zasílat data a plnit jimi zásobník pomocí rutiny přerušení. Program čeká, dokud není přijata informace o délce přenášeného programového kódu (první 2 bajty). Pak o tuto hodnotu navýší plánovaný počet příchozích znaků. Průběh je pozastaven, dokud rutina přerušení vyvolaná příjmem po sběrnici USART nenastaví akci zásobníku na hodnotu 0.

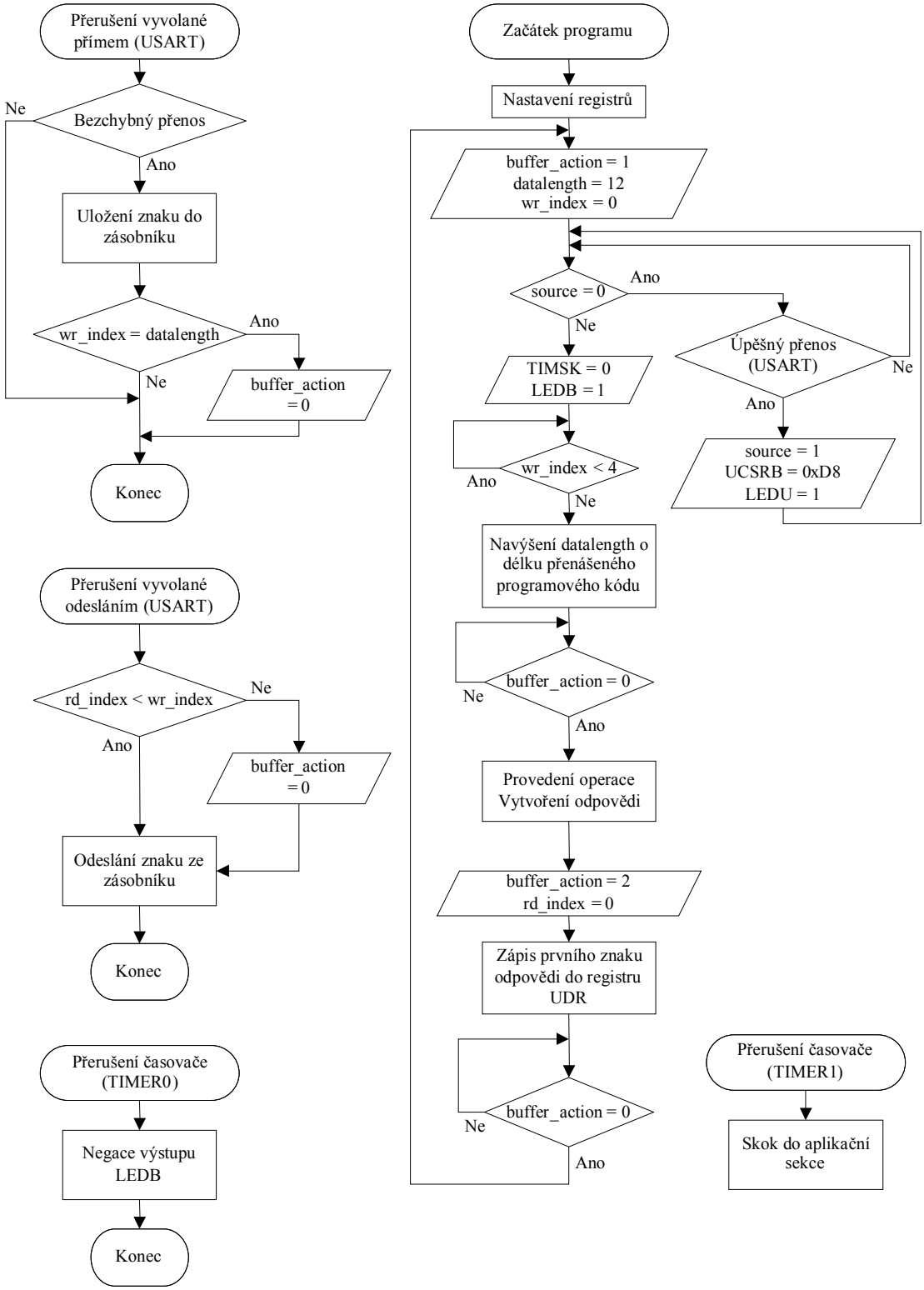
Příkaz je celý uvnitř zásobníku a zapsáním 0x00 do registru TIMSK jsou deaktivována přerušení časovačů. Tím je zabráněno blikání LEDB a skoku do aplikační sekce. V dalším kroku je potřeba zajistit schopnost vrátit odpověď. Analogicky k příjmu byla vytvořena rutina přerušení vyvolaná dokončením odesílání po rozhraní USART. Je předpokládáno, že zásobník s odpovědí bude naplněn pomocí funkce putchar a tak bude pozice posledního znaku uložena v proměnné wr_index. Dokud bude index čtení menší, než index zápisu, odesílá se další znak přečtený pomocí funkce getchar (inkrementuje rd_index). Rovnost indexů způsobuje vynulování akce zásobníku. Před vytvořením odpovědi je nutné vynulovat index čtení i zápisu. Data je pak možné naplnit funkcí putchar. Odeslání je zahájeno zapsáním prvního znaku zásobníku do registru UDR. O zbytek se stará přerušení vyvolané odesíláním. Nakonec je vhodné čekat, dokud nebude akce zásobníku nulová.

Celý tento program byl laděn rozsvícením LED diod v různých fázích programu a odesíláním testovacího příkazu pomocí terminálu. Odpověď byla generována na základě porovnávání hodnot zásobníku s konstantami zapsanými v kódu. Tím byla ověřena úspěšnost přenosu. Během pokusů bylo zjištěno nestandardní chování. Když byl převodník odpojený od vývodů RX a TX, docházelo cyklicky k vyvolání přerušení příjmem po sběrnici USART. Zároveň nikdy nedošlo k naplnění zásobníku, takže se jednalo o chybný přenos. Problém byl vyřešen změnou nastavení sběrnice. Do registru UCSRB byla zapsána hodnota 0x18. Rutiny přerušení vyvolané dokončením příjmu nebo odesílání byly deaktivovány. Byla definována nová proměnná s názvem source (zdroj). Datový typ je unsigned char. Její význam (tabulka 2.9) byl promyšlen do budoucího vývoje.

Tabulka 2.9: Proměnná source a její význam

Hodnota	Význam
0	Žádná sběrnice dosud neinicializovala zavaděč
1	Zavaděč inicializován sběrnicí USART
2	Zavaděč inicializován sběrnicí SPI
3	Zavaděč inicializován sběrnicí TWI

Na začátku hlavní programové smyčky byla vytvořena podsmyčka, která se opakuje v případě nulové proměnné source. V těle podsmyčky je podmínka, kterou lze splnit bezchybným příjmem znaku před rozhraní USART. Následně dojde k přiřazení hodnoty 1 proměnné source (program se již do této podsmyčky nikdy nedostane, zdroj byl již vybrán). Rozsvítí se dioda LEDU a zapnou se příslušná přerušení. Celý algoritmus je vyjádřen vývojovým diagramem na obrázku 2.4.



Obrázek 2.4: Vývojový diagram ve fázi oživení komunikace přes rozhraní USART

2.7.2 Ovládací aplikace

Inicializace zaváděcí sekce mikrokontroléru byla dosud testována pouze pomocí hyperterminálu. Smyslem dalšího vývoje bylo implementovat odeslání příkazu a příjem odpovědi do vlastního ovládacího softwaru. Nejprve bylo nutné navrhnout sestavu objektů včetně jejich úkolů. Ladění je v mnoha případech usnadněno textovým výpisem. Byl vytvořen formulář frmBase doplněný o procedury uvedené v tabulce 2.10.

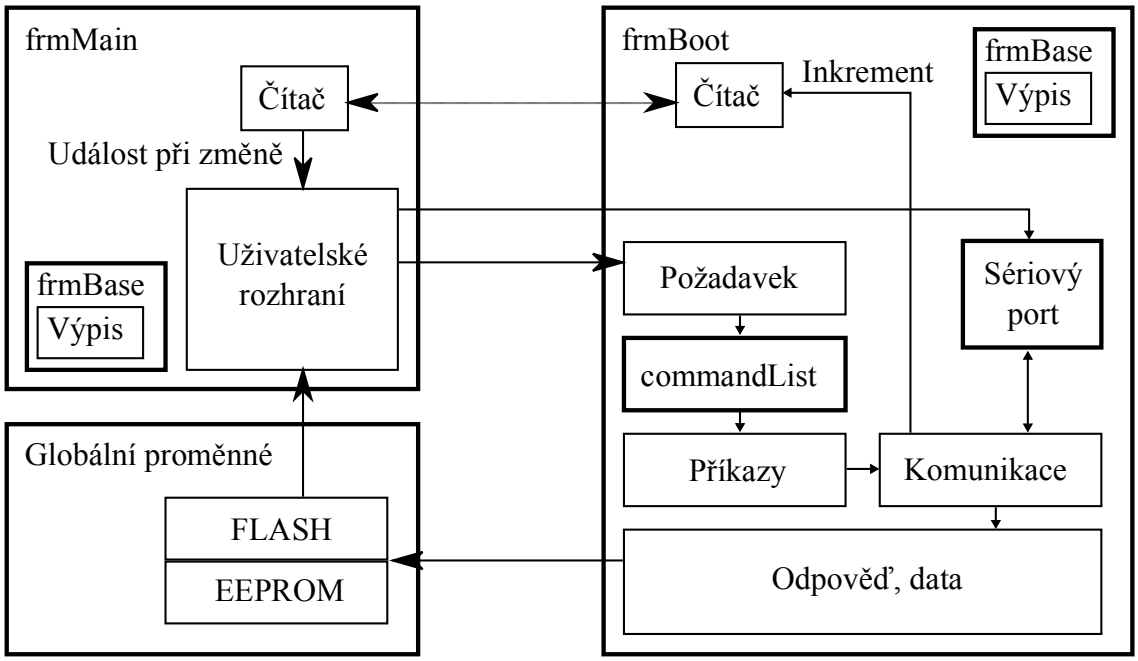
Tabulka 2.10: Formulář frmBase - Textový výpis

Název	Parametr	Význam
setMemo	Objekt textového okna	Nastaví okno, ve kterém bude výpis zobrazen.
MEMO_Add	String	Vypíše text předaný parametrem.

Předpokládá se, že frmBase bude univerzálním předchůdcem všech dalších formulářů. Je neviditelný, protože jeho jediný účel je poskytnout dalším formulářům své metody pomocí dědičnosti.

Další formulář byl pojmenován frmMain a v projektu hraje roli uživatelského rozhraní. Jeho úkolem je formulovat uživatelský požadavek dalšímu objektu, získat informaci o jeho vyřízení a zobrazit případné výsledky (např. přečtený programový kód). Dokončení je signalizováno inkrementací čísla uvnitř neviditelného textového pole. To funguje jako čítač dokončených operací a musí být předáno objektu zpracovávajícímu požadavek. Změna hodnoty vyvolá událost a dává tak aplikaci možnost zareagovat. První umístěný viditelný ovládací prvek je okno textového výpisu.

Veškeré požadavky vytvořené pomocí frmMain jsou předány formuláři frmBoot. Ten zajišťuje ovládání sérové komunikace a ukládá do globálních proměnných přenášenou paměť. Důležitá je znalost navrženého protokolu. Za účelem vytváření příkazů byl navržen pomocný objekt typu TCommand pojmenovaný jako commandList. Jeho úkolem je převádět požadavky na příkazy, kterým mikrokontrolér rozumí. Základní princip aplikace je pro názornost nastíněn na obrázku 2.5.



Obrázek 2.5: Princip ovládací aplikace

Vývoj byl zaměřen na objekt typu TCommand (proměnná s názvem commandList). Délka příkazu včetně přenášených dat je omezena velikostí paměti SRAM. Přenos celé paměti FLASH je nutné rozdělit na více příkazů. Proto vzniknul požadavek TCommand navrhnout tak, aby na základě jednoho požadavku v případě potřeby vytvořil více příkazů. Formulář frmBoot může k TCommand přistupovat pomocí metod v tabulce 2.11.

Tabulka 2.11: Metody objektu TCommand

Název	Význam
Create	Konstruktor.
setAction	Nastavení prováděné akce, parametr může být např. init.
ready	Připraví uvnitř objektu pole textových řetězců obsahující příkazy.
getNext	Vrátí další příkaz.
isFirst	Jestliže byl příkaz získaný pomocí getNext první v poli, vrátí pravdu.
isLast	Jestliže byl příkaz získaný pomocí getNext poslední v poli, vrátí pravdu.

Do formuláře frmBoot byl vložen ovládací prvek sériového portu. Jeho parametry byly nastaveny tak, aby odpovídaly konfiguraci mikrokontroléru. Aby bylo možné předat vše potřebné z uživatelského rozhraní, byly napsány procedury setComportNumber a init. Procedura init otevře sériový port, pokud je neaktivní. Potom volá sekvenci prvních čtyř metod uvedených v tabulce 2.11 a získá příkaz k inicializaci, který je přenesen procedurou USART_Transmit. Tu lze volat kdykoliv v kódu. Odpověď je získána pomocí události USART_Receive, která je vyvolána přítomností nepřečtených znaků v zásobníku. Pokud je obsah správný, textový výpis zobrazí hlášení o úspěšné inicializaci.

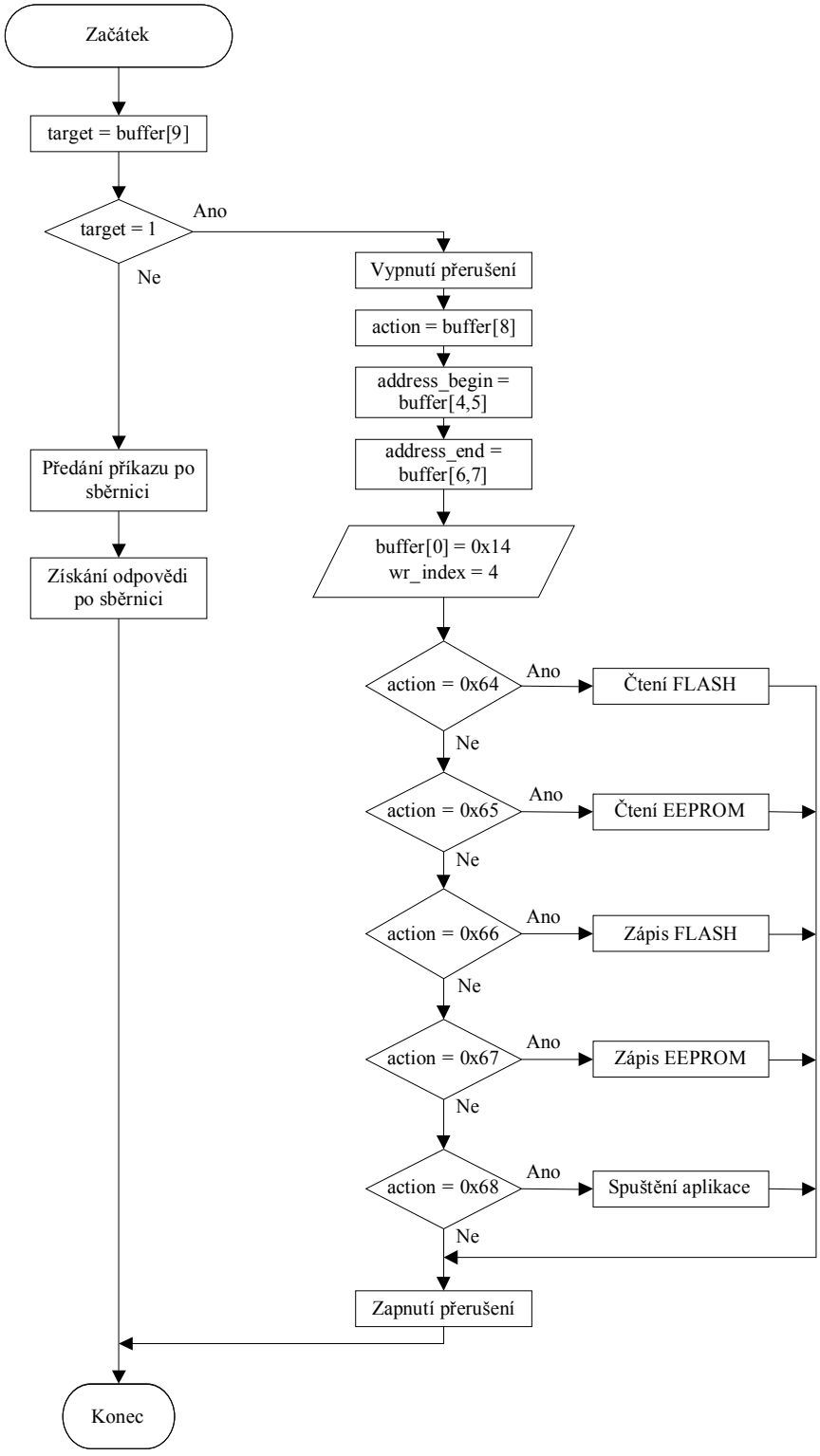
Stisknutí tlačítka Init v uživatelským rozhraní předá číslo portu formuláři frmBoot a zavolá jeho proceduru init. Program byl přeložen a spuštěn, vzorek na nepájivém poli připojen do USB portu a stiskem tlačítka Init došlo k zobrazení hlášky „Initialize OK". Také došlo k trvalému rozsvícení LEDB a LEDU. Pokus byl úspěšný i opakovaně bez resetu nebo znovuspuštění programu.

3 Paměťové operace

Tato část se zabývá strukturou bloku „Provedení operace, vytvoření odpovědi" na obrázku 2.4. Při návrhu programu je potřeba počítat s budoucím rozvojem a tak bylo vzhledem k plánovaným funkcím vyhodnoceno vhodné větvení zdrojového kódu. Nejdůležitější rozhodnutí spočívá ve výběru cílového zařízení. Devátý bajt zásobníku definuje, jestli bude zavaděč pracovat se svou vlastní pamětí, nebo bude vystupovat v roli převodníku na jinou sběrnici. Další větvení vyplývá z výčtu operací, které je možné provést. Z hlediska směru toku dat se jedná o zápis a čtení. Lze zvolit paměť FLASH nebo EEPROM. Užitečný je také skok do aplikační sekce. Osmý bajt zásobníku nese kód akce. Některé kroky jsou společné pro více možností. Všechny přenosy programového kódu potřebují znát počáteční a cílovou adresu. Zásobník má zaznamenané tyto hodnoty ve čtvrtém až sedmém bajtu. Během práce s pamětí je vypnuto přerušení, protože program nemá připravenou odpověď a tak není možné reagovat na události sběrnic. Vývojový diagram výše zmíněného bloku je na obrázku 3.1. Použité proměnné jsou vysvětleny v tabulce 3.1.

Tabulka 3.1: Proměnné bloku „Provedení operace, vytvoření odpovědi"

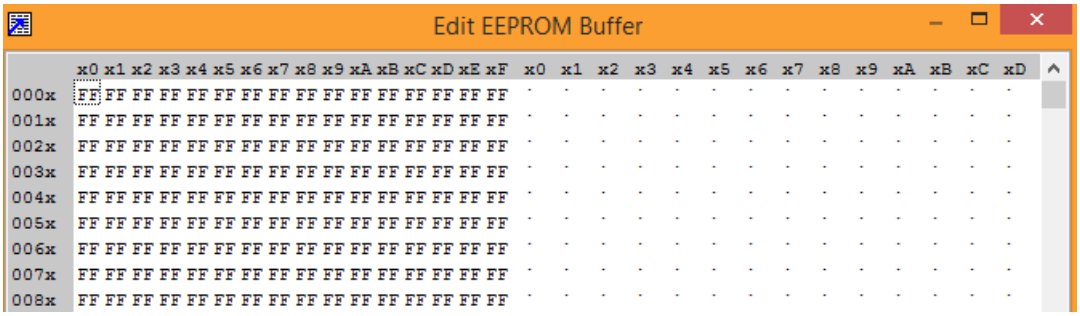
Typ	Název	Význam
unsigned char	target	Kód cílového zařízení podle tabulky 2.4.
unsigned char	action	Kód akce podle tabulky 2.4.
union	address_begin	Počáteční adresa paměti.
union	address_end	Cílová adresa paměti.



Obrázek 3.1: Vývojový diagram bloku „Provedení operace, vytvoření odpovědi"

3.1 Čtení paměti EEPROM

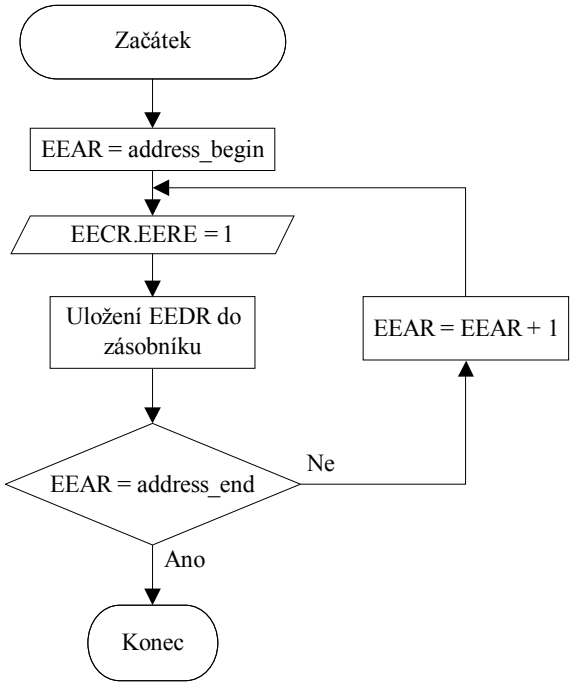
Výsledkem by měla být schopnost inicializovaného zavaděče přečíst paměť EEPROM v rozsahu definovaném příkazem a vrátit její obsah v odpovědi do počítače. Ovládací aplikace provede vhodné zobrazení kódu v mřížce podobně jako software pro programátor.



Obrázek 3.2: Zobrazení EEPROM v CodeVisionAVR

3.1.1 Mikrokontrolér

Na obrázku 3.1 se jedná o blok „Čtení EEPROM". Inspirací se stal příklad kódu jazyka C v katalogovém listu [10]. Adresa je předem uložena do EEAR (EEPROM Address Register). Čtení je aktivováno nastavením bitu EERE (EEPROM Read Enable) registru EECR (EEPROM Control Register). Přečtený znak je získán z EEDR (EEPROM Data Register). Odpověď byla naplněna přečtenými znaky podle diagramu na obrázku 3.3.



Obrázek 3.3: Čtení EEPROM

3.1.2 Ovládací aplikace

Jednotlivé části aplikace byly doplněny o objekty a funkce, které se podílí na celém procesu čtení paměti EEPROM. Nejprve byla věnována pozornost typu TCommand. Došlo k doplnění procedur pro nastavení počáteční a cílové adresy společně s délkou programového kódu přenášeného v rámci jednoho příkazu a odpovědi (velikost zásobníku). Jejich výčet je v tabulce 3.2.

Tabulka 3.2: Procedury objektu TCommand vzniklé při vývoji čtení EEPROM

Název	Význam
setDataMaxLength	Nastavení maximální délky zásobníku.
setAddressBegin	Nastavení počáteční adresy paměti.
setAddressEnd	Nastavení konečné adresy paměti.
getAddressBegin	Vráti nastavenou počáteční adresu.
getAddressEnd	Vrátí nastavenou konečnou adresu.

Bude-li maximální délka zásobníku 128 bajtů, počáteční adresa 0x000 a konečná adresa 0x1FF, vytvoří TCommand podle navrženého protokolu čtveřici příkazů reprezentující čtení paměti EEPROM v daných úsecích. Tabulka 3.3 vypisuje tyto příkazy v hexadecimálním tvaru.

Tabulka 3.3: Hexadecimální tvar příkazů pro čtení celé EEPROM po 128 bajtech

Délka dat z PC do mikrokontroléru		Délka dat z mikrokontroléru do PC		Počáteční adresa		Konečná adresa		Akce	Cíl	TWI adresa	Rez.
00	00	00	80	00	00	00	7F	65	01	00	00
00	00	00	80	00	80	00	FF	65	01	00	00
00	00	00	80	01	00	01	7F	65	01	00	00
00	00	00	80	01	80	01	FF	65	01	00	00

Různé mikrokontroléry mají rozdílné parametry včetně velikosti pamětí. Aby bylo možné v budoucím vývoji rozšířit aplikaci o podporu dalších modelů, proběhla volba způsobu tyto informace programu poskytnout. Uživatelsky přijatelná je tabulka ve formátu excel. V adresáři aplikace byl vytvořen podadresář data. Do něj byl uložen soubor models.xls. Jediný list sešitu byl pojmenován „Models" a jeho obsah (tabulka 3.4) byl prozatím doplněn počáteční a konečnou adresou paměti EEPROM pro ATmega8.

Tabulka 3.4: Obsah models.xls ve fázi vývoje čtení EEPROM

Model	EEPROM begin	EEPROM end
atmega8	0000	01FF

Do formuláře frmBoot byl umístěn ovládací prvek mřížka (jméno nastaveno na gridModels). Obsah je při startu aplikace importován ze souboru models.xls. Pro snadné získání parametrů byla vytvořena funkce GRID_Find. Je navržena tak, aby podle známé hodnoty jednoho sloupce vrátila obsah jiné buňky ve stejném řádku. Uplatnění se našlo v nových funkcích getEepromBegin a getEepromEnd (hledání počáteční a koncové adresy paměti EEPROM podle modelu v mřížce gridModels). Pro podporu nastavení velikosti zásobníku uživatelem byly vytvořeny funkce getBufferSize a setBufferSize. Hodnota je formulářem frmBoot předávána objektu commandList pomocí funkce setDataMaxLength. Událost vyvolaná příjmem po sériovém portu byla doplněna o ukládání dat do vnitřní proměnné a odesílání příkazů (commandList.getNext). Poslední procedura s názvem readEeprom spustí samotné čtení. Parametrem je počáteční a koncová adresa. Před jejím voláním je předpokládáno nastavení velikosti zásobníku pomocí setBufferSize.

Zbývajícím krokem bylo vytvoření prvků uživatelského rozhraní ve formuláři frmMain. Velikost zásobníku je nastavitelná textovým polem editBufferSize. Pro zobrazení paměti byla umístěna mřížka gridEeprom. Čtení je zahájeno stiskem tlačítka Read EEPROM. Dojde k předání velikosti zásobníku a získání počáteční i koncové adresy pomocí getEepromBegin a getEepromEnd. S těmito hodnotami dojde k zavolání procedury readEeprom (bude přečtena celá paměť). Dokončení je zachyceno událostí změny čítače (inkrementací). Přečtená paměť je uložena do globální proměnné eeprom.

Správná funkce celého procesu byla ověřena úpravou paměti pomocí programátoru, vyčtením, převedením proměnné eeprom do hexadecimálního tvaru a textovým výpisem. Po odstranění chyb zbývalo zobrazit obsah proměnné eeprom (datový typ string) v mřížce gridEeprom. Pro tento účel byla napsána funkce MEMORY_String2Grid. Jejím parametrem je cílová mřížka, paměť ve tvaru textového řetězce, počet sloupců a počet bajtů na buňku.

3.2 Čtení paměti FLASH

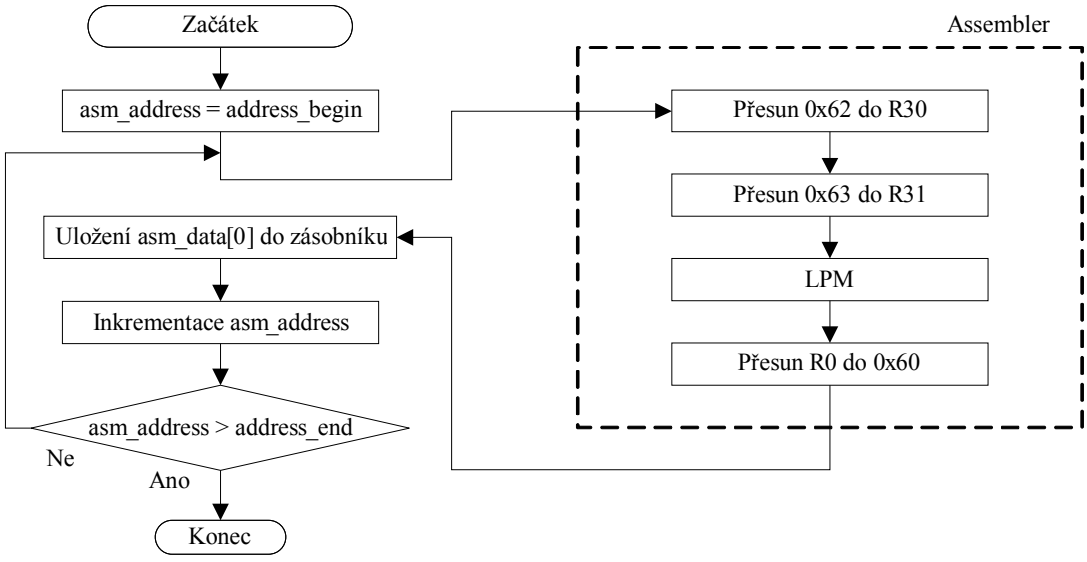
Úkolem je naučit zavaděč číst paměť FLASH v rozsahu adres podle přijatého příkazu, naplnit zásobník programovým kódem a odeslat odpověď do počítače. V ovládací aplikaci by mělo dojít k zobrazení obsahu v mřížce.

3.2.1 Mikrokontrolér

V instrukční sadě ATmega8 byl využit příkaz *LPM* (Load Program Memory), který provádí načtení paměti FLASH na adrese Z do registru R0 [10]. Z je ukazatel sestavený z registrů R30 a R31. Adresování není pro instrukci *LPM* provedeno po slovech, ale po jednotlivých bajtech [10]. Bylo rozhodnuto, že počáteční a konečná adresa komunikačního protokolu bude postavena na bajtové adresaci. Hodnoty tedy budou dvojnásobné oproti skutečným adresám. R0 až R31 jsou obecně účelné registry a jsou rezervovány pro použití kompilátoru [11]. Práce s nimi může být provedena pouze na úrovni strojového kódu. Aby bylo možné mezi jednotlivými jazykovými úrovněmi předávat hodnoty, vznikla potřeba definovat proměnné na přesné adrese paměti SRAM. CodeVisionAVR řeší tento problém pomocí deklarace proměnné ve tvaru „typ název @adresa;" [12]. V tabulce 3.5 jsou proměnné deklarované pro tento účel. Algoritmus je zobrazen vývojovým diagramem na obrázku 3.4. Zároveň je strukturou bloku „Čtení FLASH" na obrázku 3.1.

Tabulka 3.5: Proměnné pro předání hodnot mezi jazyky C a assembler

Typ	Název	Adresa	Význam
union	asm_data	0x60	Předávání dat.
union	asm_address	0x62	Předávání adresy



Obrázek 3.4: Čtení FLASH

3.2.2 Ovládací aplikace

Schopnost čtení paměti FLASH byla vytvořena analogicky k EEPROM s několika rozdíly. Objekt TCommand generuje příkazy s bajtovou adresací i když funkce setAddressBegin a setAddressEnd předpokládají adresování po slovech. Formulář frmBoot byl doplněn o metody readFlash, getFlashBegin a getFlashEnd. Dokončení operace vyvolá událost (inkrementací čítače). Přečtený kód je uložen do globální proměnné flash. Uživatelské rozhraní bylo doplněno o tlačítko „Read FLASH" a mřížku gridFlash. Soubor models.xls byl doplněn o potřebné informace (tabulka 3.6). Bylo provedeno několik úspěšných pokusů. Paměť byla editována pomocí programátoru a vyčtena ovládací aplikací. Byly odladěny některé drobné chyby a čtení paměti fungovalo v celém rozsahu.

Tabulka 3.6: Obsah models.xls ve fázi vývoje čtení FLASH

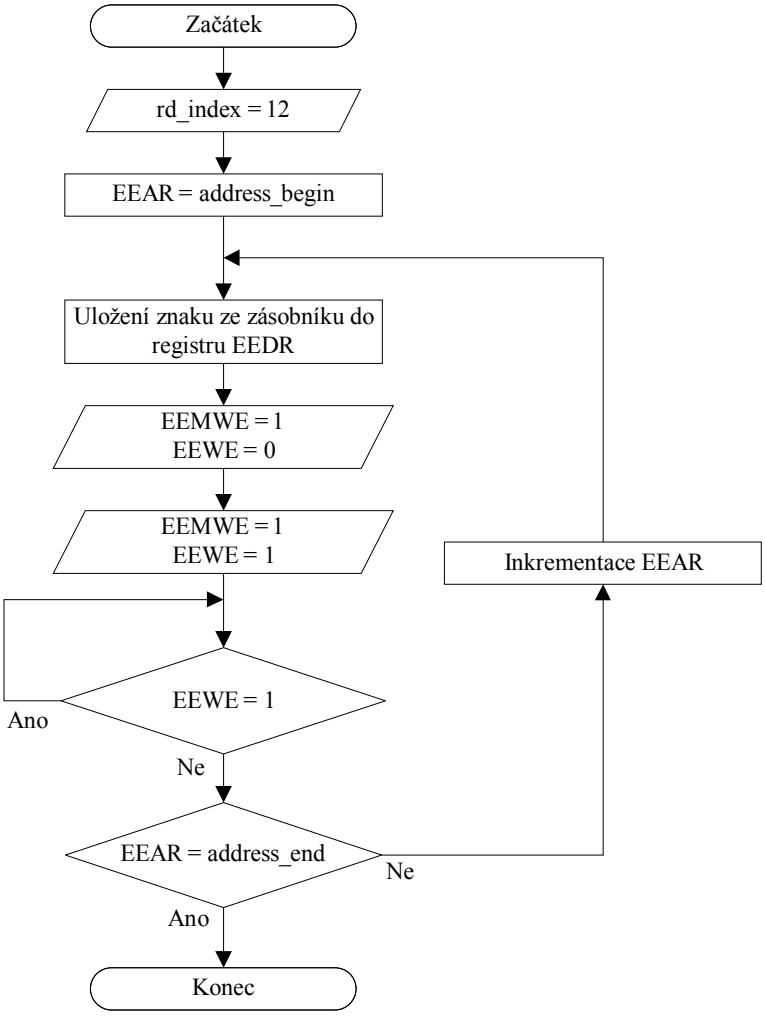
Model	Flash begin	Flash end	EEPROM begin	EEPROM end
atmega8	0000	0BFF	0000	01FF

3.3 Zápis paměti EEPROM

Jedná se o opačný přenos kódu. Příkazy budou obsahovat paměť a odpovědi budou podle komunikačního protokolu pouze čtyřbajtové. Tato část si dává za úkol přesunout obsah mřížky gridEeprom do EEPROM paměti mikrokontroléru pomocí zavaděče.

3.3.1 Mikrokontrolér

Zápis paměti využívá stejné registry jako čtení. Byla nalezena posloupnost kroků, kterými lze docílit zápis jednoho bajtu. Nejprve je potřeba počkat, až bude bit EEWE (EEPROM Write Enable) registru EECR nulový. Potom je možné naplnit adresu a data do registrů EEAR a EEDR. Zápis je nutné povolit současným nastavením jedničky bitu EEMWE (EEPROM Master Write Enable) a nuly EEWE. Od té doby se musí do čtyř hodinových cyklů zároveň přiřadit jednička do bitů EEMWE a EEWE [10]. Na obrázku 3.5 je vývojový diagram této operace pro celý rozsah daný příkazem. Jedná se zároveň o strukturu bloku „Zápis EEPROM" na obrázku 3.1. Doporučený postup byl mírně pozměněn, ale jeho správná funkce byla potvrzena.



Obrázek 3.5: Zápis EEPROM

3.3.2 Ovládací aplikace

Objekt TCommand byl doplněn o metody getData a setData. Slouží k předání obsahu paměti ve tvaru textového řetězce. Přibyla schopnost rozdělit data podle velikosti zásobníku. Na základě jednotlivých částí je generováno odpovídající pole příkazů.

Procedura writeEeprom byla vytvořena ve formuláři frmBoot. Jejími parametry jsou data k zapsání (string) a počáteční adresa. Konečná není nutná, protože je dopočítána podle délky předaných dat. Událost vyvolaná příjmem po sériovém portu nemusí plnit žádnou proměnnou přijatými zprávami. Pouze očekává jednoduchou odpověď bez datové části a odesílá příkazy. Inkrementace čítače není nezbytně nutná (i když je provedena z důvodu zachování pravidel návrhu aplikace). Rozhraní neprovádí žádnou obnovu mřížky. O dokončení operace je uživatel informován textovým výpisem.

Do formuláře frmMain bylo umístěno tlačítko „Write EEPROM". Jeho stisknutí provede volání procedury writeEeprom. Obsah gridEeprom je potřeba převést do datového typu string. Pro tento účel byla doplněna funkce MEMORY_Grid2String. Aby bylo umožněno lépe provádět pokusy s přesunem paměti, byla oblast mřížky paměti EEPROM doplněna o ovládací prvky umožňující její editaci.

Správnou funkci zápisu paměti bylo vhodné otestovat. Paměť byla nejprve přečtena. V dalším kroku byla upravena tak, aby byl její obsah snadno zapamatovatelný. Tlačítkem byl zahájen zápis. Mřížka byla vyplněna jinými hodnotami a nakonec byla paměť opět přečtena. Obsah mřížky se obnovil a byl stejný jako před spuštěním zápisu.

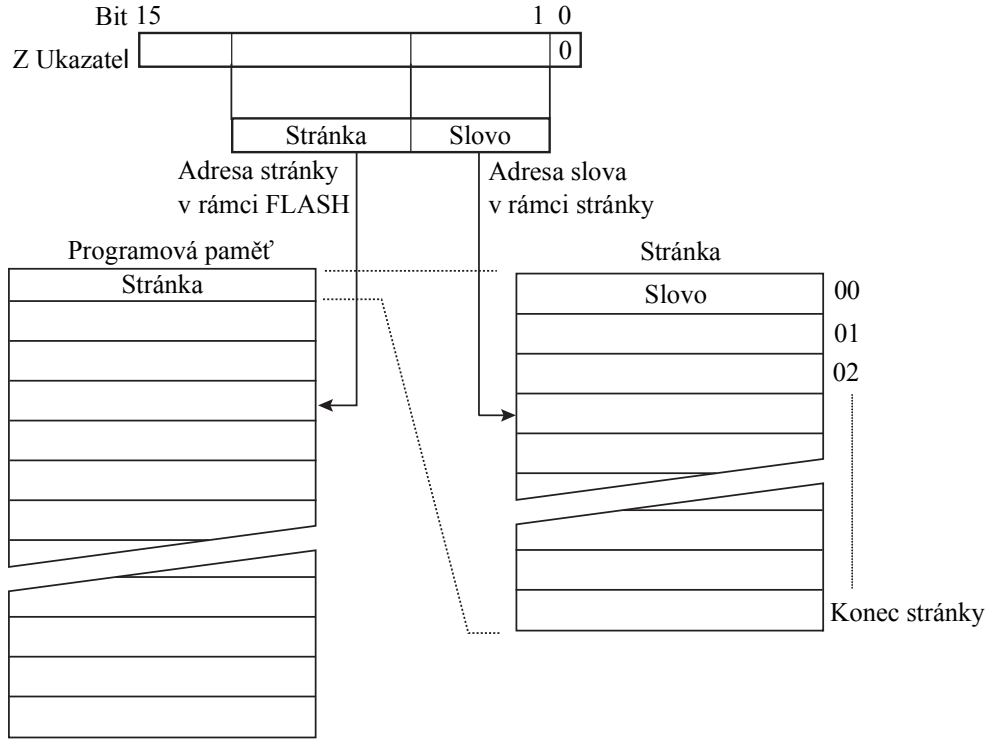
3.4 Zápis paměti FLASH

Cílem je doplnit dosud vytvořený software o schopnost přenést obsah mřížky gridFlash v ovládací aplikaci do programové paměti mikrokontroléru. Pokusem je potřeba potvrdit správnou činnost celé operace podobně jako u EEPROM.

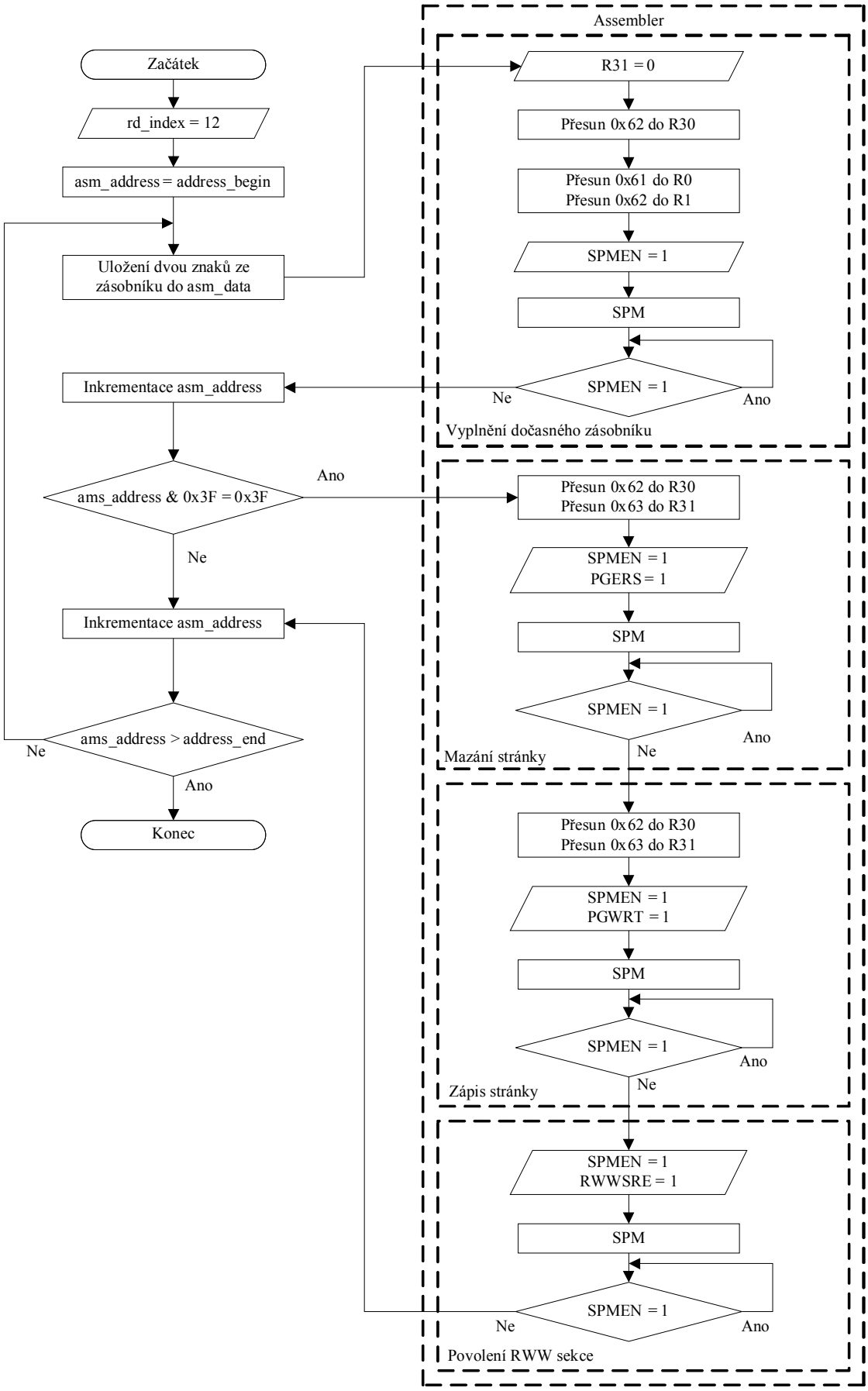
3.4.1 Mikrokontrolér

Bylo zjištěno, že paměť FLASH je rozdělena na stránky. Každá obsahuje 32 slov. Číslo slova je reprezentováno pěti dolními bity adresy. 7 horních bitů udává číslo stránky. Zápisové operace se provádějí nastavením SPMCR (Store Program Memory Control Register) a spuštěním instrukce *SPM*. Programování je možné provádět pouze stránku po stránce. Nejprve je nutné naplnit dočasný zásobník. Ukazatel Z musí obsahovat číslo slova v rámci stránky. R0 a R1 jsou naplněny zapisovanými daty. Slovo je uloženo zapsáním jedničky do bitu SPMEN (Store Program Memory Enable) registru SPMCR a spuštěním instrukce *SPM*. Další krok je mazání stránky v paměti FLASH. Do ukazatele Z je vloženo číslo stránky. Registr SPMCR musí být vyplněn takovou hodnotou, aby došlo k současnému zapsání jedničky do bitu SPMEN a PGERS (Page Erase). Provedením instrukce *SPM* dojde k vymazání paměti FLASH v rozsahu zvolené stránky. Následuje samotné programování. V registru SPMCR jsou současně zapsány jedničky na pozice bitů SPMEN a PGWRT (Page Write). Bez prodlení je opět spuštěna instrukce *SPM*. Zápis zablokuje RWW sekci pro čtení. Obnovení této schopnosti se provádí současným zápisem jedničky do bitu SPMEN a RWWSRE (Read-While-Write Section Read Enable). Stejně jako u předchozích kroků je nutné vyvolat instrukci *SPM*.

Dokončení každé *SPM* operace je signalizováno samovolným zápisem nuly na pozici bitu SPMEN. Proto není vhodné dovolit programu pokračovat, dokud k této události nedojde [10]. Význam ukazatele Z během programování je na obrázku 3.6. Algoritmus bylo nutné přizpůsobit komunikačnímu protokolu. Pokud bude rozsah počáteční a konečné adresy menší než jedna stránka, bude potřeba více příkazů pro naplnění dočasného zásobníku a zápisu paměti. Většina kroků je proveditelná pouze na úrovni strojového kódu. Proto se opakuje použití proměnných asm_address a asm_data pro sdílení hodnot s jazykem C. Příkaz respektuje adresování po bajtech. Adresa je po uložení slova do dočasného zásobníku inkrementována dvakrát. Tím je ošetřeno, že její dolní bit bude vždy nulový podle požadavku na adresování během programování. Zároveň jsou z příkazu přečteny 2 znaky najednou. Tím je zajištěn převod bajtové adresace na slovní. Mezi první a druhou inkrementací je vložena detekce konce stránky. Pokud je splněna podmínka, dočasný zásobník je naplněn. Dojde k zahájení mazání a zápisu. Vývojový diagram celé akce je na obrázku 3.7.



Obrázek 3.6: Adresování paměti FLASH během programování [10]



Obrázek 3.7: Zápis FLASH

3.4.2 Ovládací aplikace

V objektu TCommand byly provedeny drobné úpravy, aby bylo možné správně vytvářet příkazy týkající se zápisu FLASH. Většina kroků je ekvivalentní s pamětí EEPROM. Formulář frmBoot byl doplněn o proceduru writeFlash. Její parametry jsou shodné s writeEeprom. Do uživatelského rozhraní ve formuláři frmMain bylo umístěno tlačítko „Write FLASH". Jeho stisknutí provádí převod gridFlash do textového řetězce (funkce MEMORY_Grid2String) a následné volání writeFlash. Okolí mřížky bylo osazeno ovládacími prvky pro její úpravu. Bylo provedeno několik úspěšných pokusů s přesunem a změnou paměti.

3.5 Skok do aplikační sekce

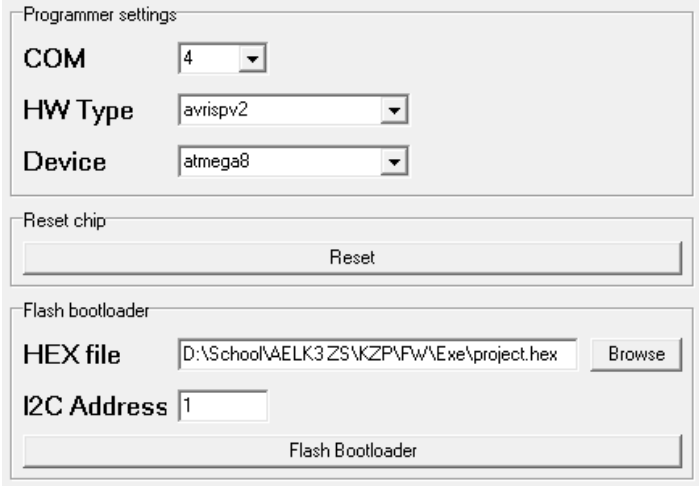
TCommand byl doplněn o podporu příkazu k tomuto účelu. Formulář frmBoot byl rozšířen o proceduru appLaunch. Do uživatelského rozhraní bylo umístěno tlačítko „Launch application", které tuto proceduru volá. Mikrokontrolér je při získání příkazu donucen přímo skočit do aplikační sekce. Ovládací aplikace nezískala odpověď o úspěšném provedení operace. Proto byla v programu zavaděče definována proměnná launch. Její hodnota je přepsána v bloku „Spuštění aplikace" na obrázku 3.1. Teprve po odeslání odpovědi dojde k vyhodnocení, jestli má být skok proveden.

4 Podpora rozhraní TWI a SPI

Celý projekt vyžadoval doplnění takových prostředků, aby bylo možné přenést příkaz prostřednictvím jednoho mikrokontroléru do druhého a zpětně přeposlat odpověď. Komunikace po rozhraní USART zůstává zachována. Liší se pouze informací o zvolené sběrnici popř. TWI adrese.

4.1 Přiřazení TWI adresy mikrokontroléru

Na základě předchozího rozhodnutí byl vytvořen systém nahrávání zaváděcí sekce pomocí ovládací aplikace z důvodu přiřazení adresy sběrnice TWI. Do formuláře frmMain byly umístěny prvky pro nastavení programátoru a modelu. Ostatní položky slouží k výběru souboru ve formátu hex (zkompilovaný zavaděč) a číslo adresy. Tlačítko „Flash Bootloader" provede potřebné kroky pro přepsání adresy a naprogramování zavaděče.



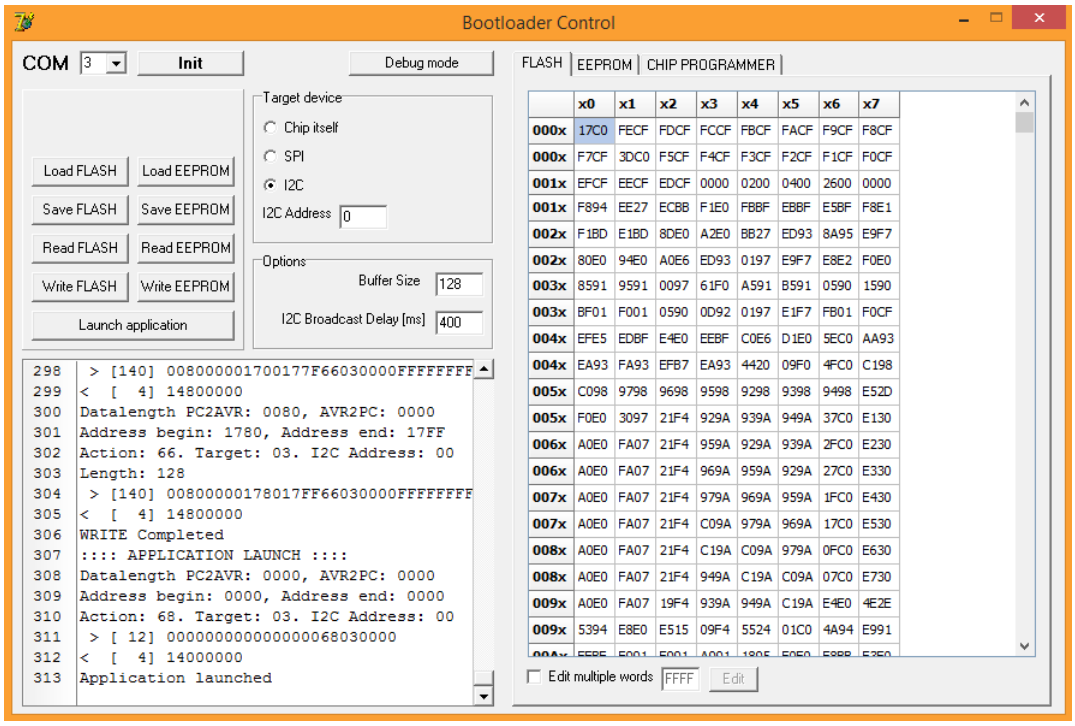
Obrázek 4.1: Ovládací prvky pro nahrávání zaváděcí sekce

Na začátku programu mikrokontroléru byla definována proměnná asm_i2c_address. Pomocí strojového kódu ji bylo přiřazeno několik hodnot. Pro každou byl zavaděč přeložen a výstupní hex soubory porovnány. Bylo zjištěno, že adresa 0xAB vytvoří v kódu slovo ve tvaru 0x0BEA. Do podadresáře data ovládací aplikace byly doplněny pomocné nástroje hex2bin, bin2hex a avrdude. Zvolený hexadecimální soubor zavaděče je nejprve převeden do binárního. To umožní jeho obsah zpracovávat jako textový řetězec, na který je aplikována funkce hledání a nahrazení. Dojde k úpravě kódu podle vyplněné TWI adresy a zpětnému převodu pomocí nástroje bin2hex. Nakonec je výsledný soubor nahrán pomocí avrdude.

Konverze mezi binárním a hexadecimálním formátem se stala jádrem funkcí getRawData a saveRawAsHex. Ze souboru vytvářejí textový řetězec a naopak. Společně s metodami MEMORY_String2Grid a MEMORY_Grid2String položily základ pro souborový vstup a výstup. Do formuláře frmMain byly pro tento účel umístěny tlačítka „Load FLASH", „Save FLASH", „Load EEPROM" a „Save EEPROM".

4.2 Volba sběrnice v uživatelském rozhraní

Objekt typu TCommand a formulář frmBoot byly doplněny o metody getTarget, setTarget, getI2cAddress a setI2cAddress. Jejich účel spočívá v předání informace o cílovém mikrokontroléru podle rozhraní. Pokud je pomocí setTarget předána hodnota „i2c" a zároveň je nastavena nulová adresa procedurou setI2cAddress, jedná se o broadcast požadavek. Je sice možné zaslat příkaz všem účastníkům sběrnice, ale nelze zpětně vyčíst odpověď. Mikrokontrolér sloužící jako převodník z USART na TWI zašle požadavek a sám vygeneruje odpověď, kterou vrátí do PC. Ovládací aplikace na odpověď zareaguje a odešle na sériový port další příkaz. Operace s pamětí ještě nemusí být dokončena a proto je nutné vytvořit časovou prodlevu mezi příjmem odpovědi a odesláním příkazu na straně počítače. Pro nastavení této doby byla vytvořena procedura setBroadcastDelay. Nakonec bylo veškeré nastavení zpřístupněno ovládacími prvky ve formuláři frmMain.



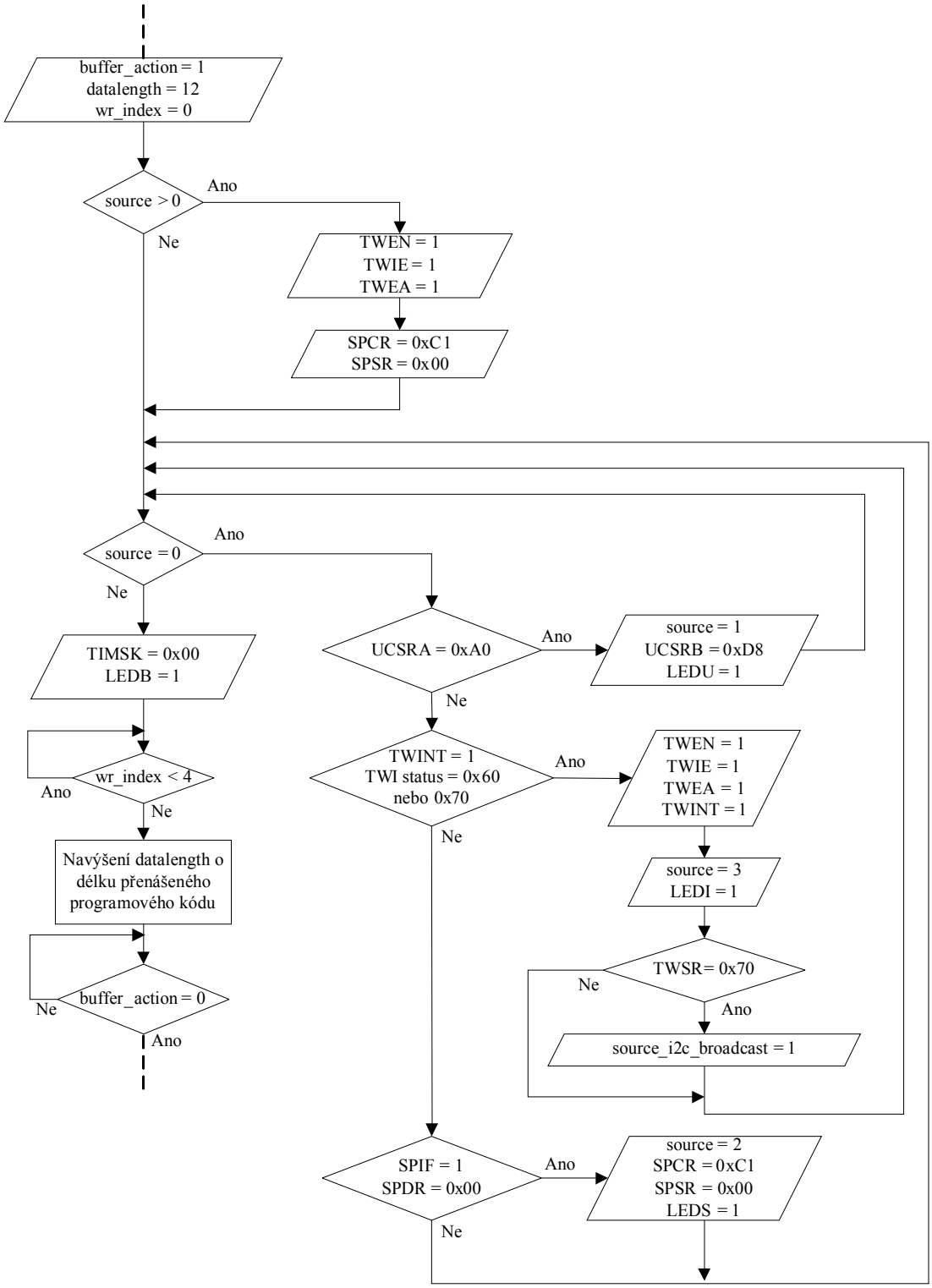
Obrázek 4.2: Ovládací aplikace

4.3 Příjem příkazu po libovolné sběrnici

Byla upravena část programu mikrokontroléru, ve které probíhá čekání na příchozí data po sběrnici USART. Byla doplněna detekce příjmu po TWI a SPI. Pro všechna rozhraní se předpokládá vypnuté přerušení, dokud není zaznamenán první příchozí bajt. Pokud dojde k příjmu, uloží se do proměnné source (zdroj) kód sběrnice, která zareagovala jako první. Zapne se její rutina přerušení (v ní vyřešen algoritmus pro příjem příkazu) a rozsvítí se příslušná LED dioda.

Činnost na TWI je signalizována bitem TWINT (TWI Interrupt Flag) registru TWCR (TWI Control Register). O přesné události informuje pět dolních bitů TWSR (TWI Status Register). Hodnota TWSR s maskovanými třemi horními bity je v následujících diagramech pojmenována jako „TWI status". Pokud je TWINT roven jedné a TWI status má hodnotu 0x60, došlo na sběrnici k příjmu vlastní adresy, která je uložena v registru TWAR (TWI Address Register). TWAR je na začátku programu nastaven podle proměnné asm_i2c_address. Hodnota je posunuta o jeden bit doleva, protože dolní bit TWAR zapíná reakci na broadcast požadavek. Pokud je mikrokontrolér volán adresou 0x00, TWI status je roven 0x70 [10]. Tato skutečnost je zapamatována pomocí proměnné source_i2c_broadcast. Volání na základě vlastní i nulové adresy nastaví sběrnici TWI jako zdroj. Rozsvítí se LEDI a je zapnuto potvrzování a rutina přerušení. Ve vývojových diagramech lze ve společném bloku najít přiřazení hodnoty 1 některým z bitů TWINT, TWEA (TWI Enable Acknowledge), TWSTA (TWI Start Condition), TWSTO (TWI Stop Condition), TWEN (TWI Enable Bit) nebo TWIE (TWI Interrupt Enable Bit). Význam takového bloku není postupný zápis jednoho bitu po druhém, ale přiřazení takové hodnoty registru TWCR, aby jmenované bity byly rovné log. 1 a ostatní zůstaly nulové. Toto pravidlo vzniklo pro jednodušší formulaci hodnoty registru.

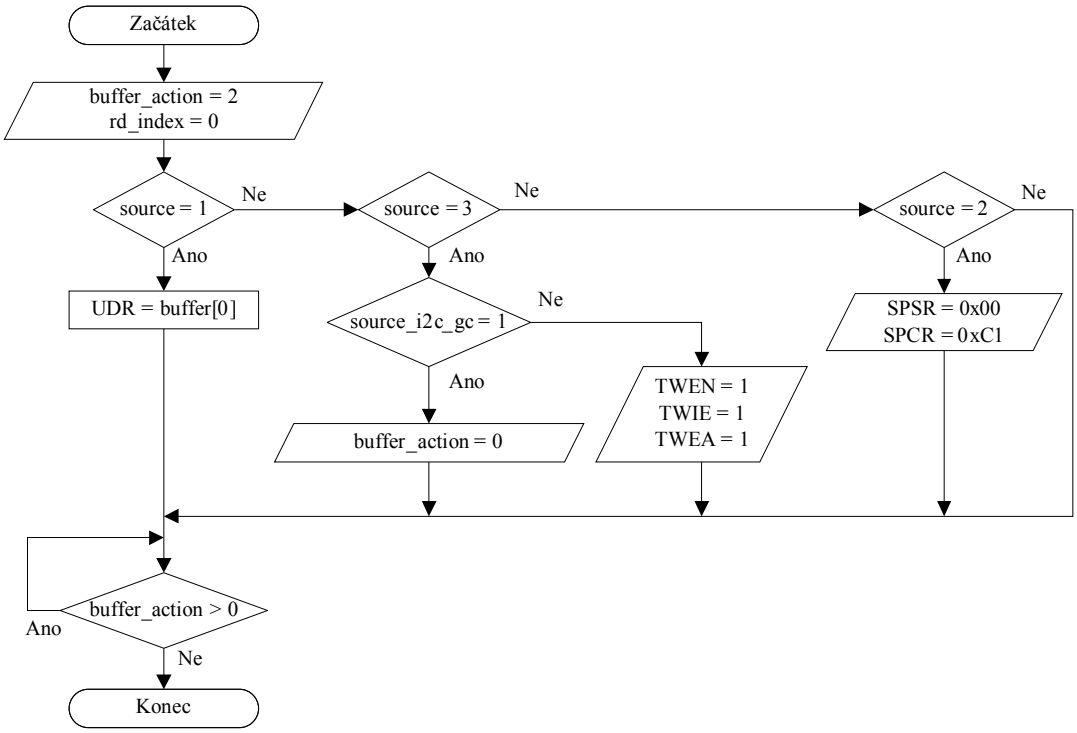
Příjem bajtu přes rozhraní SPI je signalizován bitem SPIF (SPI Interrupt Flag) registru SPSR (SPI Status Register) [10]. Bylo rozhodnuto, že výběr SPI jako zdroje bude umožněn zasláním hodnoty 0x00. Zápisem 0xC1 do registru SPCR (SPI Control Register) dojde ke konfiguraci rozhraní do režimu slave a zapnutí rutiny přerušení.



Obrázek 4.3: Volba zdrojové sběrnice a příjem příkazu

4.4 Odeslání odpovědi po libovolné sběrnici

Další rozšíření se týká části programu odesílající odpověď po rozhraní USART. Sběrnice je zvolena podle proměnné source (zdrojová sběrnice). Pokud byl bootloader inicializován přímo počítačem prostřednictvím sériového portu, dojde k prostému zapsání prvního bajtu odpovědi do UDR (USART Data Register). SPI a TWI jsou nastaveny jako slave a jejich rutiny přerušení zajišťující přenos odpovědi jsou vypnuty. Master nemá možnost získat odpověď, dokud nedojde k jejich zapnutí. To je provedeno až v této fázi (data v zásobníku jsou již připravena). Pokud byl mikrokontrolér volán adresou 0x00 (TWI broadcast), je přenos odpovědi přeskočen.

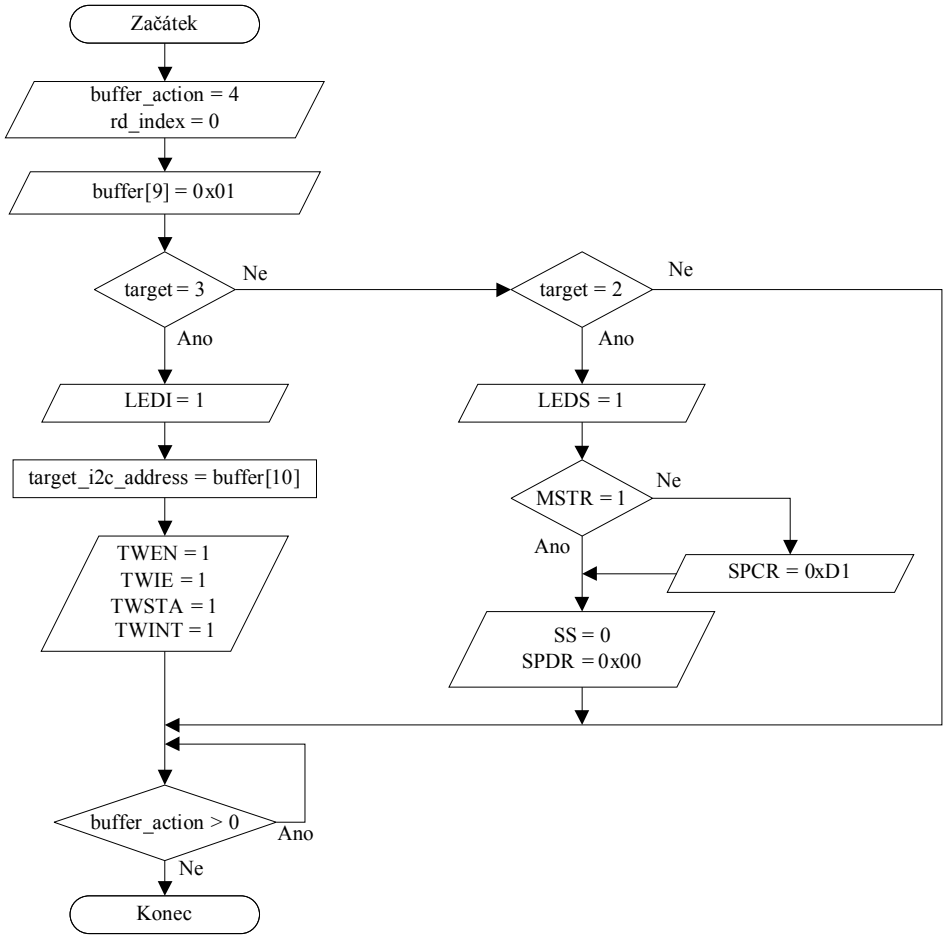


Obrázek 4.4: Odeslání odpovědi po zdrojové sběrnici

4.5 Převod USART na SPI nebo TWI - Přenos příkazu

Jedná se o strukturu bloku „Předání příkazu po sběrnici" na obrázku 3.1. Je předpokládáno, že proměnná target (cílová sběrnice) obsahuje příslušný kód. Devátý bajt zásobníku (stejný obsah jako target) je přepsán na hodnotu 1, jinak by cílový mikrokontrolér přesměroval příkaz sám na sebe. Podle použitého rozhraní je rozsvícena dioda LEDI, nebo LEDS.

V případě TWI je do pomocné proměnné target_i2c_address uložena hodnota desátého bajtu zásobníku. Přenos je zahájen odesláním Start Condition a zapnutím přerušení. Tato kombinace zároveň přepne mikrokontrolér do režimu master. Před odesláním po SPI probíhá kontrola nastavení role. V případě potřeby, je do registru SPCR zapsána hodnota 0xD1 (master, zapnuté přerušení). Nakonec je přepnut výstup SS (Slave Select) a odeslán bajt (0x00) zahajující SPI komunikaci.

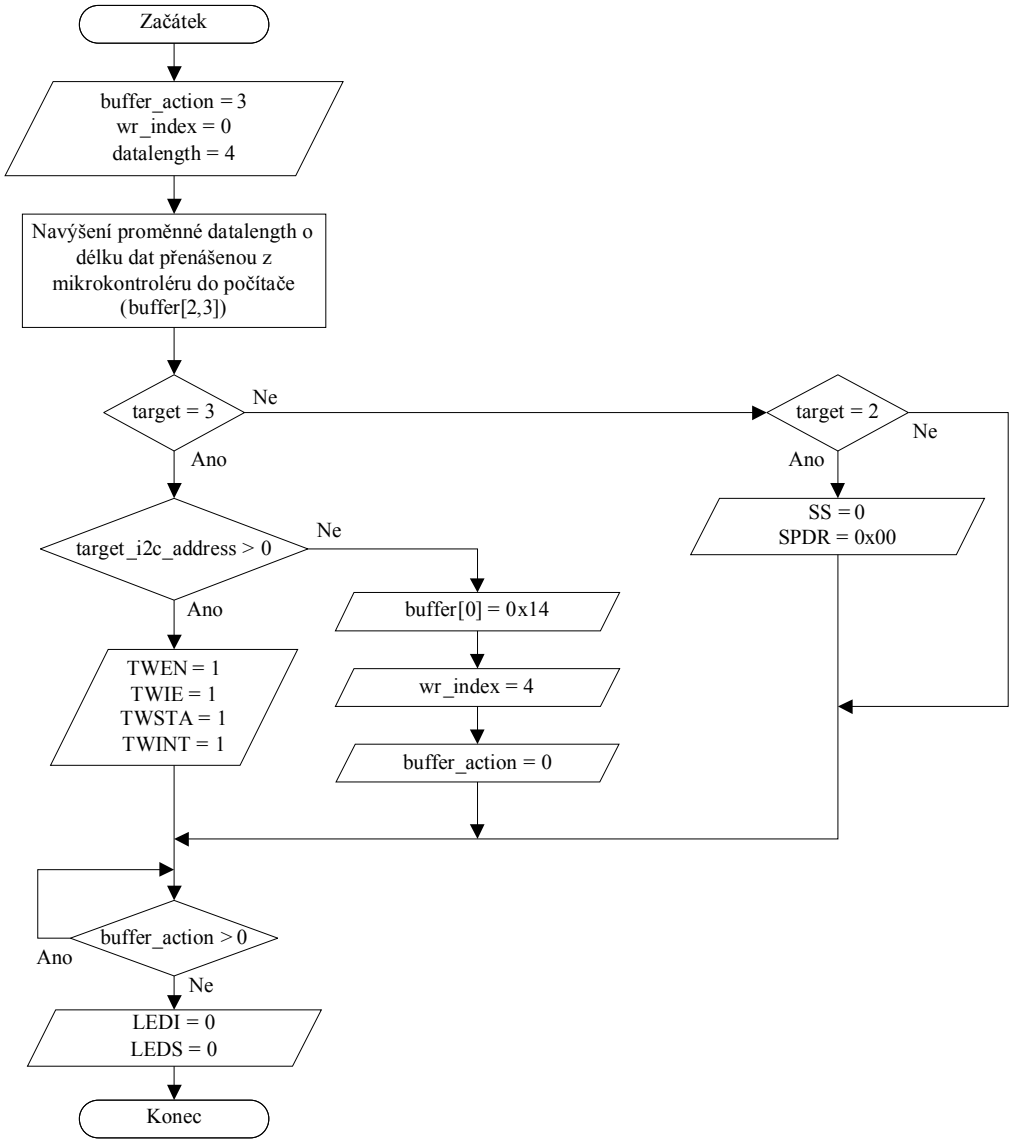


Obrázek 4.5: Přeposlání příkazu na zvolenou sběrnici

4.6 Převod USART na SPI nebo TWI - Přenos odpovědi

Jedná se o strukturu bloku „Získání odpovědi po sběrnici" na obrázku 3.1. Nejprve je provedeno nastavení zásobníku a výběr sběrnice, která zahájí přenos odpovědi.

Pokud byl předchozí příkaz přijat po TWI a mikrokontrolér byl adresován jako 0x00, bude celý postup přenosu odpovědi přeskočen. V opačném případě je zahájena komunikace odesláním „Start Condition". SPI zahájí příjem přepnutím výstupu SS a odesláním bajtu (0x00). Obě rozhraní řeší přenos ve svých rutinách přerušení.



Obrázek 4.6: Příjem odpovědi z cílové sběrnice

5 Návrh testovací DPS

Navržený bootloader bylo nutné otestovat na hardwaru, který disponuje použitými prostředky a zajišťuje požadovanou propojitelnost. Byla navržena testovací deska osazená mikrokontrolérem Atmel ATmega8. Použitý krystalový oscilátor má frekvenci 7,3728 MHz (jako obvod zapojený na nepájivém poli). Pro indikaci rozhraní byly umístěny diody LEDU, LEDI a LEDS. Aktivitu zavaděče signalizuje LEDB. Pro účel aplikační sekce byla deska doplněna o diody LED1 - LED8. Dále bylo vytvořeno 5 ukázkových programů s různým světelným efektem využívajících těchto LED. Tlačítko slouží jako reset mikrokontroléru.

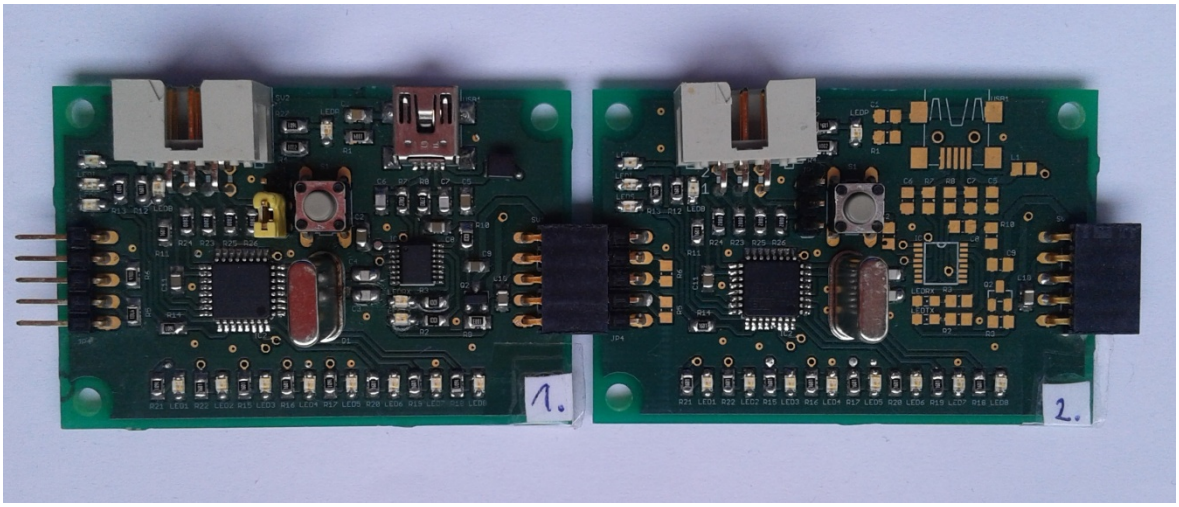
Bylo vyrobeno 6 desek ve dvou osazovacích variantách. Každá byla po nahrání zaváděcí sekce označena číslicí podle použité TWI adresy. Podrobnosti návrhu jsou uvedené v přílohách B až F.

- ***Slave (4x)***

Obsahuje konektor PFL06 pro připojení programátoru a komunikaci po rozhraní SPI. Jeho pátý pin lze jumperem propojit s vývodem reset, nebo SS. Po levém a pravém boku jsou umístěny pinheadery, které poskytují rozvod napájení, signálu reset a datových vývodů SDA a SCL sběrnice TWI.

- ***Master (2x)***

Je osazen všemi součástkami varianty slave. Deska je navíc doplněna o USB/UART převodník včetně indikačních LED tohoto rozhraní. Jeho RTS výstup je zapojen tak, aby byl schopný resetovat všechny propojené desky.



Obrázek 5.1: Propojené desky master a slave

Závěr

Popisovaný vývoj dospěl k realizaci bootloaderu se zadanými parametry. Zavaděč je schopný po volitelném rozhraní obousměrně přenášet paměť FLASH i EEPROM. Program zaváděcí sekce byl navíc rozšířen o schopnost vystupovat jako master sběrnic TWI nebo SPI. Ovládací aplikace poskytuje jednoduché rozhraní pro provádění jednotlivých operací. Její objektová struktura je připravena pro snadnou rozšiřitelnost o další možnosti. Správná funkce byla ověřena na navržených deskách plošných spojů.

Využití by tento systém mohl najít v oblasti automatizace a testování. V kombinaci se vzdálenou správou je možné přístroje připojené k počítači programově aktualizovat na dálku včetně zařízení na sběrnici. V oblasti výroby by mohl být vstupním materiálem mikrokontrolér s naprogramovaným zavaděčem. Konečné výrobky by pak mohly využít broadcast nahrávání aplikace podle daného modelu zařízení nebo továrních dat (např. výrobní linka, směna nebo časové období).

Další rozvoj by mohl zajistit překlad zavaděče pro další modely rodiny Atmel AVR na straně ovládací aplikace. Protože bootloader zabírá v zaváděcí sekci Atmel ATmega8 přibližně 92 % paměti, není možné dosavadní funkce rozšířit. Modely s vyšší pamětí by mohly být doplněny o ověřování přeneseného programového kódu kontrolními součty a zpracování chybových stavů na straně mikrokontroléru.

Použitá literatura

[1] Androidmarket. Jak vypadá Android uvnitř aneb co je ROM, kernel, bootloader a další? [online]. [Citace: 10.5.2015]. Dostupné z: http://androidmarket.cz/android/jak-vypada-android-uvnitr-aneb-co-je-rom-kernel-bootloader-a-dalsi/

[2] Tiny PIC bootloader. [online]. [Citace: 25. 5. 2015]. Dostupné z: http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm

[3] STMicroelectronics. AN2606: STM32 microcontroller system memory boot mode. [Citace: 25. 5. 2015]. Dostupné z: http://www.st.com/web/en/resource/technical/document/application_note/CD00167594.pdf

[4] Texas Instruments. MSP430 Programming Via the Bootstrap Loader (BSL). [Citace: 25. 5. 2015]. Dostupné z: http://www.ti.com/lit/ug/slau319i/slau319i.pdf

[5] Texas Instruments. MSPBoot – Main Memory Bootloader for MSP430™ Microcontrollers. [Citace: 25. 5. 2015]. Dostupné z: http://www.ti.com/lit/an/slaa600a/slaa600a.pdf

[6] Freescale Semiconductor. Kinetis Bootloader. [online]. [Citace: 25.5.2015]. Dostupné z: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=KBOOT

[7] HW server. STK500 Protocol AVR Bootloader. [online]. [Citace: 17.5.2015]. Dostupné z: http://www.hw.cz/navrh-obvodu/software/stk500-protocol-avr-bootloader.html

[8] MATOUŠEK, David. *Práce s mikrokontroléry ATMEL AVR ATmega16 4. díl*. Praha: BEN - technická literatura, 2006. ISBN 80-7300-174-8.

[9] Arduino. Guide to the Arduino Mini. [online]. [Citace: 17.5.2015]. Dostupné z: http://www.arduino.cc/en/Guide/ArduinoMini

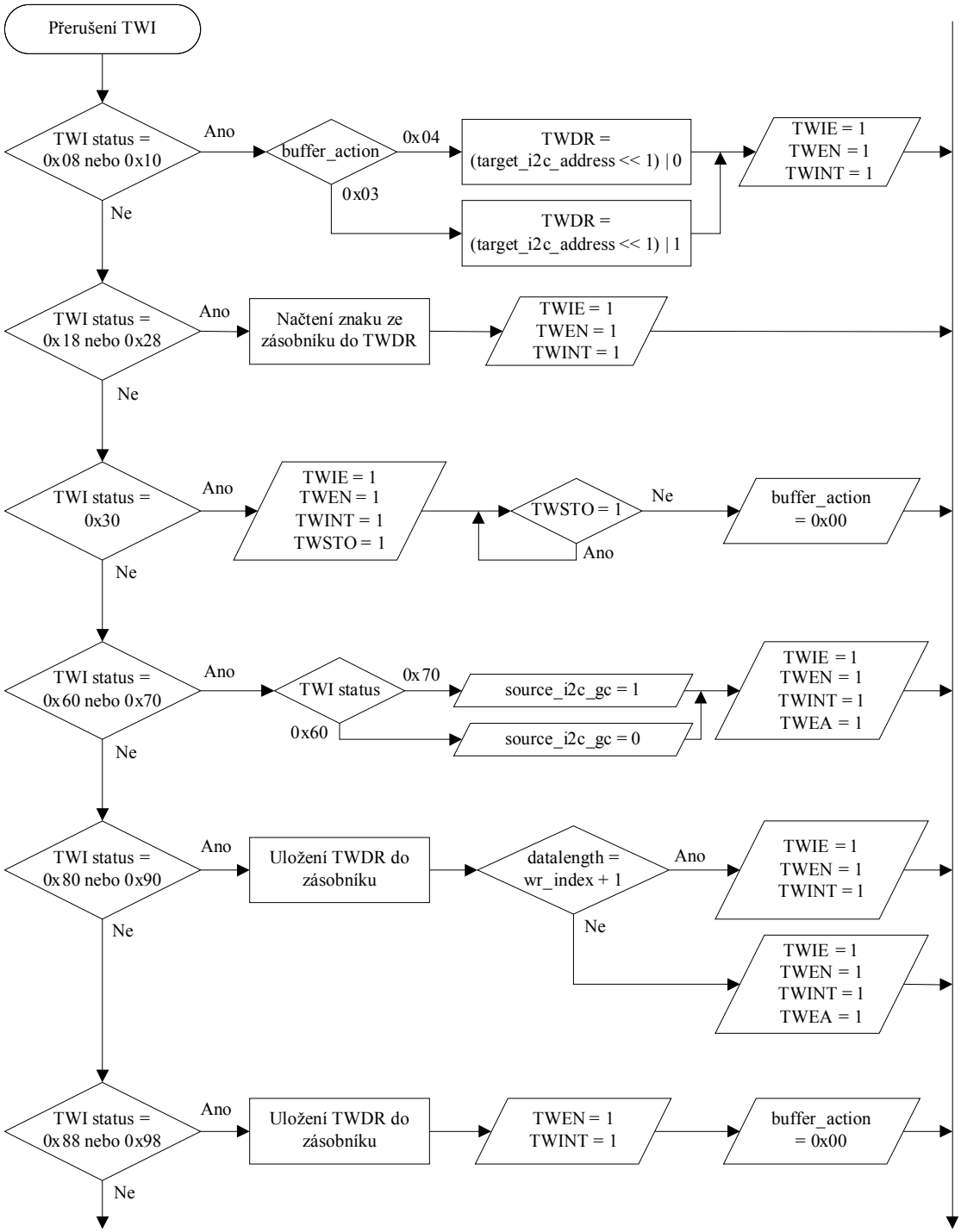
[10] Atmel Corporation. ATmega8(L) datasheet. [Citace: 20. 6. 2014]. Dostupné z: http://www.atmel.com/images/atmel-2486-8-bit-avr-microcontroller-atmega8_l_datasheet.pdf

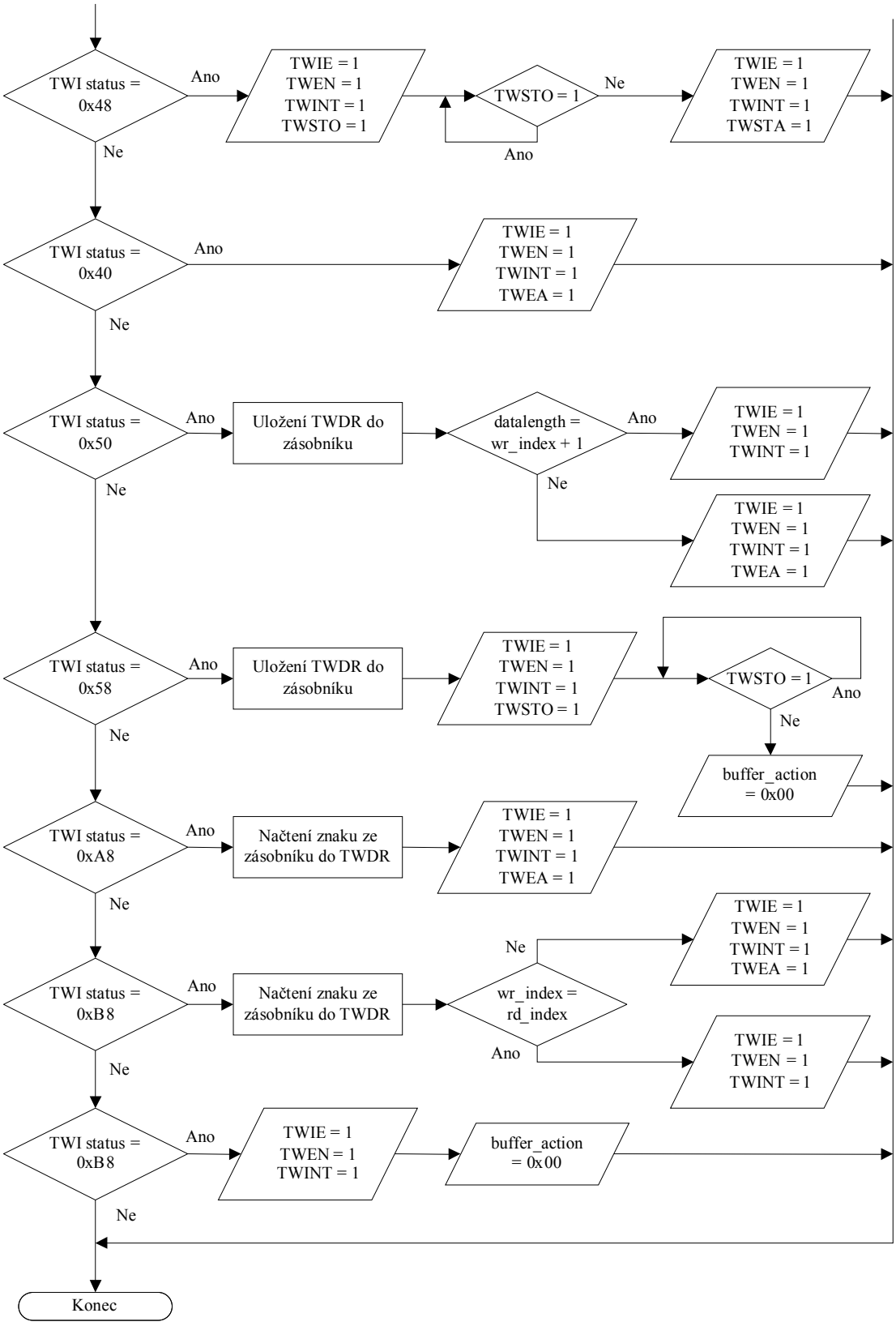
[11] Atmel Corporation. how to use general purpose register in c. [online]. [Citace: 22.6.2014]. Dostupné z: http://www.avrfreaks.net/forum/how-use-general-purpose-register-c

[12] stack exchange. Initializing a variable and specifying the storage address the same time: is it possible? [online]. [Citace: 24.6.2014]. Dostupné z: http://stackoverflow.com/questions/11771377/initializing-a-variable-and-specifying-the-storage-address-the-same-time-is-it

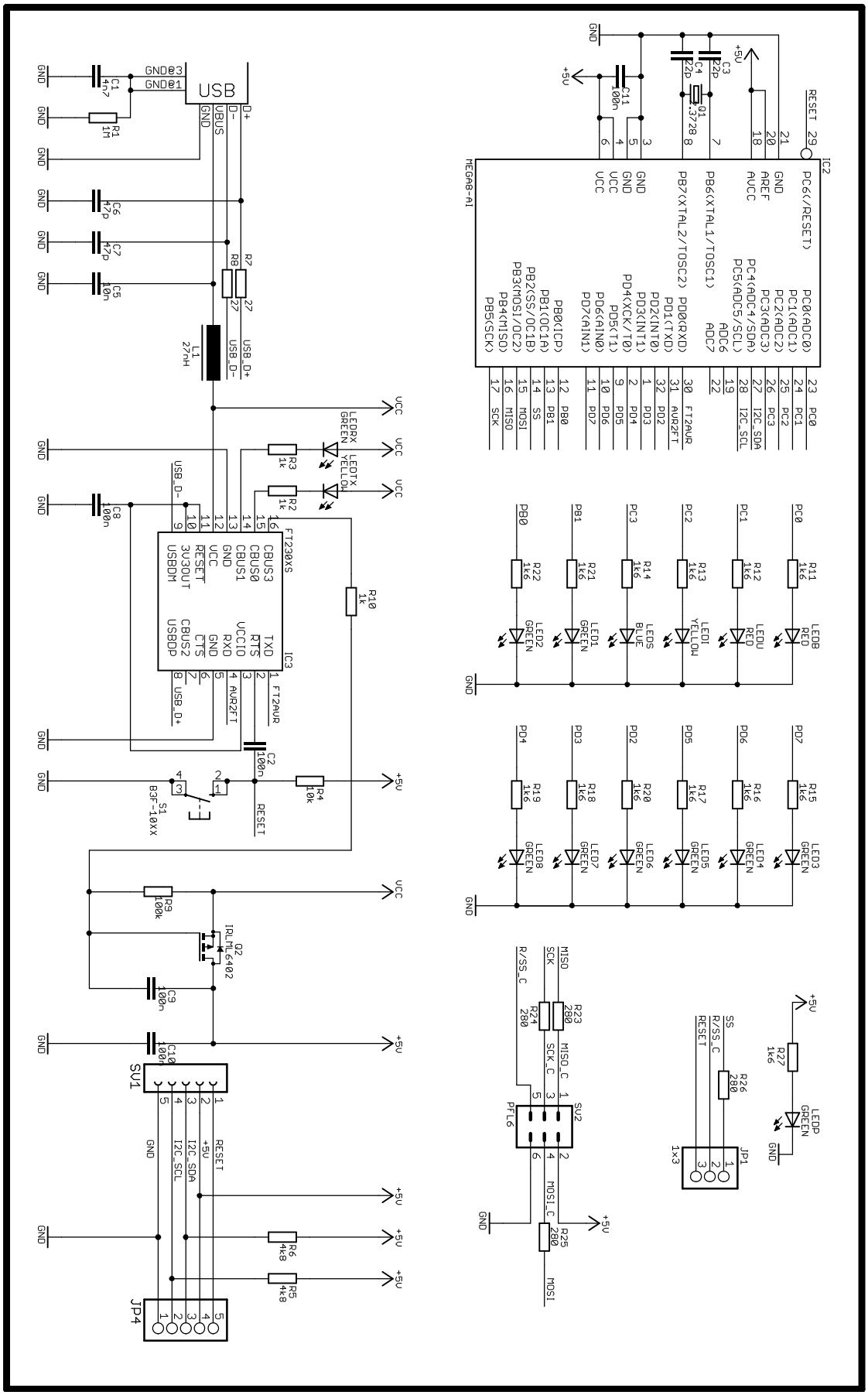
Přílohy

Příloha A - Rutina přerušení sběrnice TWI

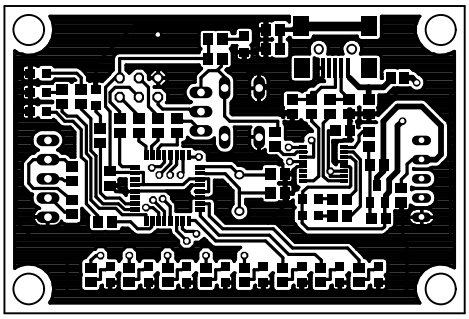



TWI status = 0x48
Ano
TWIE = 1
TWEN = 1
TWINT = 1
TWSTO = 1
TWSTO = 1
Ne
TWIE = 1
TWEN = 1
TWINT = 1
TWSTA = 1
Ano
Ne
TWI status = 0x40
Ano
TWIE = 1
TWEN = 1
TWINT = 1
TWEA = 1
Ne
TWI status = 0x50
Ano
Uložení TWDR do zásobníku
datalength = wr_index + 1
Ano
TWIE = 1
TWEN = 1
TWINT = 1
Ne
TWIE = 1
TWEN = 1
TWINT = 1
TWEA = 1
Ne
TWI status = 0x58
Ano
Uložení TWDR do zásobníku
TWIE = 1
TWEN = 1
TWINT = 1
TWSTO = 1
TWSTO = 1
Ano
Ne
buffer_action = 0x00
Ne
TWI status = 0xA8
Ano
Načtení znaku ze zásobníku do TWDR
TWIE = 1
TWEN = 1
TWINT = 1
TWEA = 1
Ne
TWI status = 0xB8
Ano
Načtení znaku ze zásobníku do TWDR
wr_index = rd_index
Ne
TWIE = 1
TWEN = 1
TWINT = 1
TWEA = 1
Ano
TWIE = 1
TWEN = 1
TWINT = 1
Ne
TWI status = 0xB8
Ano
TWIE = 1
TWEN = 1
TWINT = 1
buffer_action = 0x00
Ne
Konec

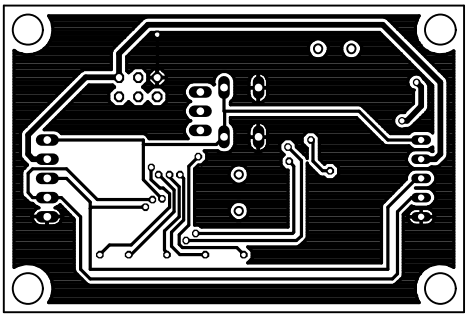
Příloha B - Schéma zapojení navržené DPS



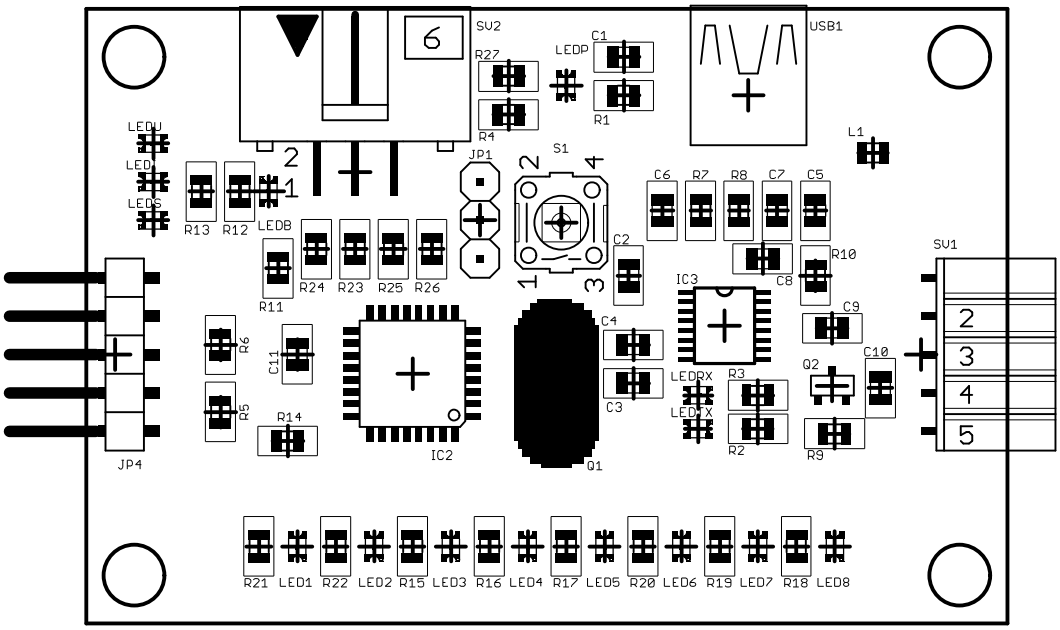
Příloha C - Horní vrstva navržené DPS



Příloha D - Spodní vrstva navržené DPS



Příloha E - Rozmístění součástek navržené DPS



Příloha F - Seznam součástek navržené DPS

Součástka	Hodnota	Pouzdro	Master	Slave
C1	4n7	C0805	Ano	Ne
C2, C8, C9	100n	C0805	Ano	Ne
C10, C11	100n	C0805	Ano	Ano
C3, C4	22p	C0805	Ano	Ano
C5	10n	C0805	Ano	Ne
C6, C7	47p	C0805	Ano	Ne
IC2	MEGA8-AI	TQFP32-08	Ano	Ano
IC3	FT230XS	SSOP16	Ano	Ne
JP1	1x3	PINHD-1X3_2.54	Ano	Ano
JP4	1x5	PINHD-1X5_2.54-90°	Ano	Ano
L1	27nH	FB805	Ano	Ne
LED1 - LED8	GREEN	CHIPLED_0805	Ano	Ano
LEDB	RED	CHIPLED_0805	Ano	Ano
LEDI	YELLOW	CHIPLED_0805	Ano	Ano
LEDP	GREEN	CHIPLED_0805	Ano	Ano
LEDRX	GREEN	CHIPLED_0805	Ano	Ne
LEDS	BLUE	CHIPLED_0805	Ano	Ano
LEDTX	YELLOW	CHIPLED_0805	Ano	Ne
LEDU	RED	CHIPLED_0805	Ano	Ano
Q1	7.3728	HC49U-V	Ano	Ano
Q2	IRLML6402	SOT23	Ano	Ne
R1	1M	R0805	Ano	Ne
R2, R3, R10	1k	R0805	Ano	Ne
R4	10k	R0805	Ano	Ano
R5, R6	4k8	R0805	Ano	Ne
R7, R8	27	R0805	Ano	Ne
R9	100k	R0805	Ano	Ne
R11 - R22, R27	1k6	R0805	Ano	Ano
R23 - R26	280	R0805	Ano	Ano
S1	B3F-10XX	B3F-10XX	Ano	Ano
SV1	FE05W	FE05W	Ano	Ano
SV2	PFL6	ML6L	Ano	Ano
USB1	USB-MB-S	USB-MB-S	Ano	Ne