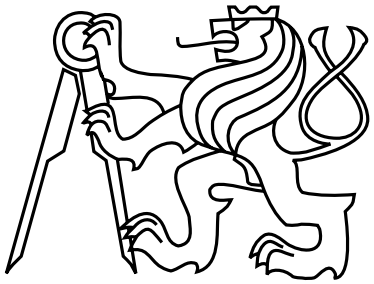
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Aplikace pro komunikaci s univerzální periferní deskou

Jan Mikolášek

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

Studijní program: Elektrotechnika a informatika, strukturovaný, Bakalářský

Obor: Výpočetní technika

23. května 2011

Poděkování

Rád bych poděkoval vedoucímu mé práce Ing. Pavlu Kubalíkovi, Ph.D. za veškerý čas, který mi věnoval a za jeho cenné rady. Veliký dík dále patří mé rodině, která mě podporovala po celou dobu mého studia.

Prohlášení

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

V Nelahozevsi dne 22. 5. 2011 .....

Abstract

The object of this thesis is a creation of libraries for communication with universal peripheral board in Java programming language. Created libraries enable to control individual peripheries of the board from the computer. During the work firmware, communication protocol and demo application were created beside the libraries.

Abstrakt

Předmětem této práce je vytvoření knihoven pro komunikaci s univerzální periferní deskou v programovacím jazyce Java. Vzniklé knihovny umožňují ovládat jednotlivé periferie desky z počítače. Během práce byl vedle knihoven vytvořen firmware pro desku, komunikační protokol a demonstrační aplikace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Univerzální periferní deska navržená Ing. Josefem Nouzákem[19] představuje alternativu ke komerčním produktům. Je postavena okolo procesoru AVR ATmega 2560 a obsahuje tyto periferie: obvod reálného času, paměť EEPROM, 3 rozhraní UART, USB rozhraní, PS/2 konektor, slot pro SD kartu, LCD displej a 2 rozšiřující konektory umožňující připojit k desce další moduly.

Cílem této práce bude vytvořit sadu knihoven v programovacím jazyce Java. Tyto knihovny umožní ovládat periferie desky prostřednictvím počítače propojeného s deskou USB kabelem. Vytvořené knihovny budou následně využity pro tvorbu demonstrační aplikace s grafickým uživatelským prostředím. Požadovanými funkcemi jsou například zobrazování textu na displeji, získávání dat z obvodu reálného času anebo kopírování souborů z/na SD kartu. Tyto funkce jsou uvedeny v kapitole 2. Součástí práce bude návrh komunikačního protokolu vhodného pro účely ovládání desky, vytvoření firmware pro periferní desku a také krátká rešerše podobných komerčních řešení.

Kapitola 2

Popis problému, specifikace cíle

Hlavním úkolem je vytvořit soubor knihovních prvků v programovacím jazyce Java, které umožní zjišťování a nastavování stavu jednotlivých periferií procesorové desky prostřednictvím osobního počítače propojeného s procesorovou deskou USB kabelem. Součástí této práce bude:

- Návrh komunikačního protokolu pro komunikaci s deskou
- Vytvoření firmware pro procesorovu desku
- Vytvoření javovských knihovních prvků pro ovládání periferií desky
- Naprogramování aplikace s grafickým uživatelským prostředím demonstrující možnosti desky s použitím těchto knihovních prvků

2.1 Funkční požadavky

Pro každou periferii desky je potřeba definovat základní operace, které bude možné provádět prostřednictvím výsledné javovské aplikace. Tyto požadavky jsou vymezeny dále.

2.1.1 LCD displej

- Zobrazení zadaného textu na určitém řádku
- Smazání zadaného řádku
- Smazání celého displeje

2.1.2 Obvod reálného času

- Nastavení a zjištění času
- Nastavení a zjištění datumu

2.1.3 Pamět EEPROM

- Čtení a zapisování stránek z/do paměti
- Čtení a zapisování skupin znaků z/do paměti od určité adresy

2.1.4 Klávesnice

- Výpis stisklých kláves

2.1.5 SD karta

- Kopírování souborů z/na SD kartu
- Zjištování adresářové struktury na SD kartě
- Vytváření adresářů
- Mazání adresářů/souborů
- Zjišťování údajů o celkové/volné kapacitě karty

2.1.6 Rozšiřující konektor

Pro rozšiřující konektor je k dispozici redukce, která umožňuje připojení jednak modulu výstupů s relátky a jednak modulu vstupů. Požadované operace jsou:

- Spínání a rozepínání relátek
- Zjišťování stavu vstupů

Kapitola 3

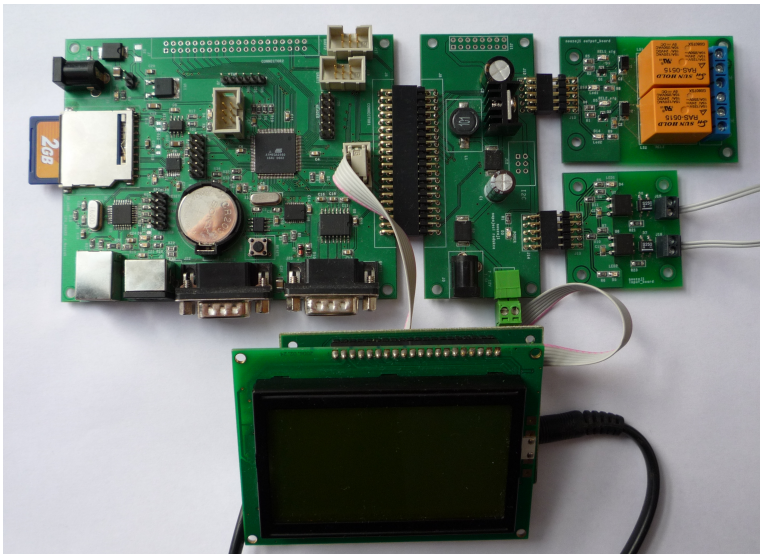
Analýza a návrh řešení

3.1 Popis Nouzákovy desky

O desce Ing. Josefa Nouzáka, se kterou budu pracovat, jsem se zmiňoval již v úvodu. Pro úplnost zde uvedu její základní parametry. Deska s připojeným LCD displejem a redukcí se vstupy a výstupy je uvedena na obrázku 3.1.

- procesor AVR ATmega2560 [8]
- obvod reálného času
- paměť EEPROM 256 Kbit
- USB port
- PS/2 rozhraní
- SPI rozhraní
- JTAG
- 2x konektor Canon 9M
- 2x rozšiřující 40-ti pinový konektor
- slot pro SD kartu
- modul LCD displeje
- moduly vstupů a výstupů
- modul Ethernetu
- cena bez Ethernet modulu - 1600 Kč

K desce byla vytvořena sada knihovních funkcí v programovacím jazyce C pro práci s periferiemi, která ulehčuje mikroprogramování.



Obrázek 3.1: Nouzákova deska včetně připojených modulů vstupů a výstupů a modulu LCD displeje

3.1.1 Napájení desky

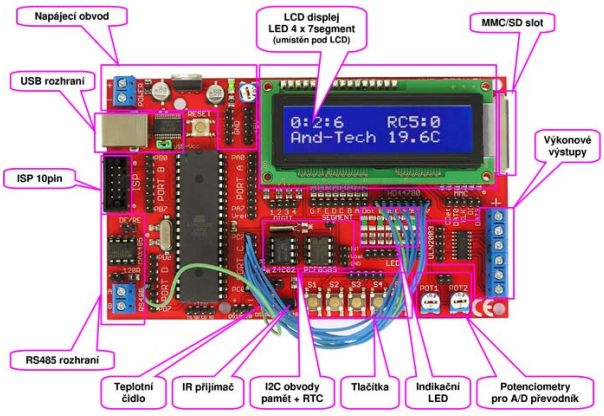
Dlouhou dobu jsem zjišťoval jakým zdrojem desku a modul displeje napájet, protože tuto informaci zapoměl autor desky zmínit. Nakonec jsem zjistil, že desku, modul LCD displeje a redukci pro rozšiřující konektor lze napájet zdrojem stejnosměrného napětí 9V/500 mA.

3.2 Komerční řešení periferních desek

3.2.1 AVR EvB

Vývojová deska AVR EvB [3] nabízí za cenu kolem 1500 Kč poměrně dobré vybavení. Deska může být osazena procesory AVR ATmega32, ATmega16 nebo ATmega644p. Její součástí je RTC, paměť EEPROM, infračervený přijímač, teplotní čidlo, slot pro SD/MMC kartu, sériové rozhraní, 8 indikačních LED, 5 tlačítek, 2 potenciometry, 5 výkonových výstupů, 4 sedmisegmentové zobrazovače, dvouřádkový LCD displej, USB a ISP konektory. Součástí desky v revizi 4 vyráběné od roku 2010 je samovybuzovací piezoměnič. Rozmístění periferií je ilustrováno na obrázku 3.2.

Vzhledem k tomu, že je deska osazena procesorem typu AVR, má uživatel k dispozici na výběr z mnoha vývojových prostředků. Existují pro programovací jazyky Basic, Pascal a C. Podrobněji se o nich zmíním v některé z dalších kapitol, protože budu tvořit firmware také pro procesor AVR.



Obrázek 3.2: AVR EvB(převzato z[3])

3.2.2 MB-ATmega128 3.0

Tento výrobek pochází z dílny českého výrobce PK Design. Jak již název napovídá, základem desky je procesor AVR ATmega128. Deska je význačná velikou modularitou. V základním provedení neobsahuje téměř žádné periferie. Naleznete na ní dva odpojitelné RS232 konektory, RTC, 4 konektory s I/O porty a resetovací tlačítko. Výrobce desky nabízí další přídavné moduly s tlačítky, LED diodami, přepínači nebo LCD displejem. Dále pak moduly s USB, RS232, PS/2 nebo VGA rozhraním, moduly se segmentovými displeji a moduly s pamětí SRAM. Cena desky se pohybuje kolem 1500 Kč, přídavné moduly pak lze pořídit od 60 do 550 Kč.

Vývojové prostředky jsou obdobné jako u desky AVR EvB.

3.2.3 STM32F103

Tato periferní deska je určena pro procesor ARM Cortex M3. Její součástí je USB konektor, slot pro SD kartu, potenciometr, RTC, 8x LED, 46 I/O pinů, JTAG konektor, konektor pro LCD displej a 2x UART. Cena se pohybuje kolem 1200 Kč. Podrobnější specifikaci a obrázky si lze prohlédnout na stránkách obchodu Gravitech[1].

Pro procesory ARM je k dispozici sada vývojových nástrojů Keil, kterou poskytuje přímo výrobce. Obsahuje C/C++ překladač, programy pro ladění a simulaci. Demo nástroje lze zdarma stáhnout ze stránek výrobce[10], práce v něm je omezena maximální velikostí kódu 16KB. Pro tuto desku je poskytována sada standartních knihoven.

3.2.4 Srovnání desek

V tabulce 3.1 je srovnáno vybavení a cena všech uvedených desek.

	Nouzák	AVR EvB	MB-ATmega128	STM32F103
RTC	ano	ano	ano	ano
EEPROM	ano	ano	ne	ne
USB	ano	ano	sam. modul	ano
PS/2	ano	ne	sam. modul	ne
RS232	ano	ne	ano	ano
JTAG	ano	ne	ne	ano
LCD	sam. modul	ano	sam. modul	sam. modul
SD slot	ano	ano	ne	ano
Ethernet	sam. modul	ne	ne	ne
I/O piny	ano	jen output	ano	ano
Základní cena	1600 Kč	1500 Kč	1500 Kč	1200 Kč

Tabulka 3.1: Srovnání periferních desek Ing. Josefa Nouzáka a podobných komerčních produktů

3.3 Firmware

3.3.1 Vývojové prostředky

3.3.1.1 Volba programovacího jazyka a překladače

Firmware desky bude rozsáhlejší aplikace. Tvořit ho v assembleru by bylo značně neefektivní. Navíc autor desky již vytvořil knihovny funkcí pro jednotlivé periferie, které ulehčí programování firmware. Tyto knihovny jsou napsány v jazyce C. To bude zřejmě určující pro volbu programovacího jazyka.

Pro procesory AVR existují překladače pro různé programovací jazyky. Mikroprogramy je možné psát v Basicu za využití nástroje BASCOM-AVR [5], v Pascalu (nástroj AVRco [2]) anebo v C. Pro programovací jazyk C existují placené nástroje CodevisionAVR (cena asi 150 Euro) [6]a mikroC PRO for AVR (cena kolem 200 USD)[12]. Konkurencí je jim volně dostupná sada WinAVR [18], která obsahuje i balík užitečných knihoven avr-libc.

Já jsem se rozhodl pro mikroprogramování v jazyce C. S mikroprogramováním sice nemám větší zkušenosti, ale jazyk C narozdíl od ostatních uvedených znám. Ve prospěch jazyka C navíc hovoří bezplatný nástroj WinAVR a knihovny funkcí od autora desky.

3.3.1.2 Vývojové prostředí

Pro vývoj firmware použiji vývojové prostředí AVR Studio 4 [4], které poskytuje firma Atmel zdarma. Po instalaci nástroje WinAVR se do AVR Studia integruje GCC překladač a mikroprogramy je možné psát v jazyce C. Toto prostředí sice nenabízí takový komfort jako je například kontrola deklarace proměnných známá z vývojových prostředí pro desktopové aplikace, jeho výhoda je ale v možnosti okamžitého překladu programu pro procesor. Součástí AVR Studia je nástroj pro nahrání programu do paměti procesoru.

3.3.2 Práce s SD kartou

Deska by měla podporovat práci se souborovým systémem FAT32 tak, aby bylo možné zapisovat, číst a editovat soubory jak deskou, tak i ve standartní čtečce paměťových karet na počítači. Realizace mikroprogramu, který by toto zajišťoval, by byla velice náročná, možná by byla v rozsahu celé jedné bakalářské práce. Je tedy potřeba použít některou z existujicích knihoven pro práci s FAT32, přičemž je nutné, aby dokázala pracovat s SD kartou a byla otestována na mikrokontrolérech AVR.

Našel jsem 3 řešení vhodné pro procesory AVR. První z nich používá ve svém projektu autor desky. Je dostupné zde[17]. Pro mě však nemá význam, protože funkce pro práci s SD kartou jsou příliš provázány se zbytkem aplikace (přímo v tělech funkcí se odesílají data po sériové lince). Většina funkcí by se proto musela pozměnit. To by nebylo moc efektivní. Lépe jsou na tom další dvě řešení, která poskytují interface pro práci s FAT32, respektive SD kartou. Při jejich použití by mělo stačit změnit konfigurační soubory definující elektrická zapojení slotu pro SD kartu. Obě knihovny jsou open source a jsou poměrně dobře zdokumentovány. První z nich je dílem japonského inženýra a je volně dostupná na[9]. Druhou vytvořil Němec Roland Riegel, k dispozici je na[14].

Zatím nevím pro kterou z knihoven se rozhodnu, budu muset obě vyzkoušet při nasazení v mém projektu.

3.3.3 Programování procesoru

Programovat procesor na této desce je možné několika způsoby. První možností je programátor Ponyprog, který využívá sériovou linku a je možné si ho levně vyrobit doma. Návod na jeho stavbu lze nalézt například na[13]. Další možností je ISP programátor s rozhraním STK500. U tohoto typu programátoru je procesor programován přes SPI. Rozhraní STK500 pochází přímo ze stejnojmeného vývojového kitu firmy Atmel, proto takový programátor spolupracuje s AVR studiem[4]. Deska obsahuje JTAG konektor, čili je možné použít některý ze sofistikovaných programátorů, které umožňují ladění programu v procesoru. Takové programátory jsou ovšem velmi drahé. Autor desky zmiňuje také možnost programování přes USB. Tento způsob je ale velmi pomalý, a proto prakticky nepoužitelný.

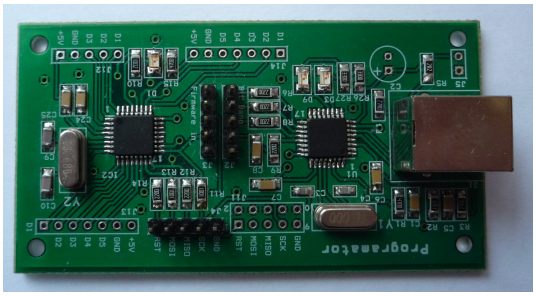
Rozhodl jsem se použít programátor s rozhraním STK500, viz obrázek 3.3. Umožňuje pohodlné programování, připojení přes USB rozhraní a mám možnost si ho zapůjčit od vedoucího práce.

3.3.3.1 Užitečné rady pro programování

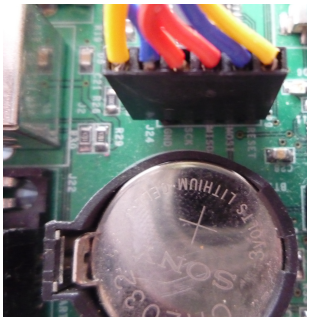
Programátor, který jsem použil, komunikuje s deskou po SPI. Je nutné správně propojit programátor s deskou. SPI konektor na straně programátoru je vidět na obrázku 3.3 úplně dole. Na straně desky se připojuje k pravé straně konektoru, který je umístěn vedle pouzdra baterie, nad PS/2 konektorem, viz obrázek 3.4.

Při programování je dobré odpojit periferie využívající SPI, tzn. modul Ethernetu a SD kartu. Pokud jsou tyto periferie připojené nemusí se podařit procesor naprogramovat.

Po propojení programátoru s deskou a počítačem pak v AVR Studiu[4] v menu Tools vyberete Program AVR - Connect. Jako platformu vyberete STK500. Pokud neznáte port,



Obrázek 3.3: Programátor s rozhraním STK500



Obrázek 3.4: SPI konektor pro připojení programátoru

můžete vybrat Auto a poté potvrdit. Důležíté je nepotrvdit dialogové okno, které se následně zobrazí. Po kliknutí na tlačítko Storno by se měla zobrazit nabídka pro nahrání programu do procesoru. Vyberete přeložený program, kliknete na tlačítko Program a pak už jen čekáte na dokončení procesu programování.

3.4 Java aplikace

3.4.1 Vývojové prostředí

Při výběru vývojového prostředí pro tvorbu knihovních prvků a demonstrační aplikace mám jasno. Jediné, se kterým mám zkušenosti je Netbeans IDE. Nemám důvod ho měnit. Konkrétně použiji verzi 6.8.

3.4.2 Grafické uživatelské prostředí

Java aplikace by měla umožňovat pohodlné intuitivní ovládání, proto bude její součástí grafické uživatelské prostředí. Pro tvorbu GUI nabízí Java tři knihovny – AWT, Swing a SWT.

Knihovna AWT (Abstract Windowing Toolkit) se objevila už v první verzi JDK (Java Development Kit). Filozofií AWT je, že každá komponenta v Javě má svůj nativní protějšek v systému. Vytvořené GUI tedy ladí s komponentami vašeho operačního systému. Tímto

je omezena přenositelnost aplikace, protože různé komponenty mají rozdílné vlastnosti v různých operačních systémech.

Od verze Javy 1.2 lze používat novou knihovnu Swing. V tomto případě není GUI napasováno na nativní komponenty, čímž se získává větší platformová nezávislost. Nevýhodou této knihovny jsou nároky na paměť a rychlost procesoru. To se mělo odstranit v knihovně SWT, ale moc se to nepovedlo.

Ve své aplikaci budu zřejmě kombinovat knihovny AWT a Swing. Převažovat bude AWT, jednak má měnší nároky na počítač a její komponenty se mi v operačním systému Windows líbí víc. AWT má ovšem omezený počet typů komponent, proto budu muset možná použít některé komponenty knihovny Swing. Příkladem takové komponenty může být JTree pro vizualizaci stromových struktur.

3.4.3 Komunikace s deskou

Deska se propojuje s počítačem USB kabelem. Za USB konektorem na desce je ale FTDI převodník, takže se toto připojení jeví jako sériová linka. Standartní edice Javy neobsahuje knihovny pro sériovou komunikaci, musím tedy použít externí knihovnu.

Společnost Sun Microsystems uvedla API pro sériovou komunikaci s názvem JavaComm, to ovšem není dostupné pro všechny platformy. Roku 2005 bylo vydáno JavaComm API pro Windows, později ovšem Sun upustilo od podpory této platformy, pro 64-bitové operační systémy Windows tak JavaComm není vůbec použitelné. Od roku 2006 je podporována Linuxová platforma. Vzhledem k tomu, že vyvíjím aplikaci pro Windows a pracuji na počítači s Windows 7 64-bit, nepřipadá pro mě použití této knihovny v úvahu.

Mnohem lépe se jeví použití open source knihovny RXTX[15], kterou doporučuje i autor desky. Knihovny jsou k dispozici pro operační systémy Windows i Linux, pro 64-bitové systémy jsou k dispozici na[16]. Součástí balíku knihovny je i textový soubor popisující její instalaci. Ta bude podrobněji rozebrána v některé z příloh. Použití knihovny RXTX bohužel snižuje přenositelnost aplikace mezi různými platformami, lepší řešení jsem ale nenalezl.

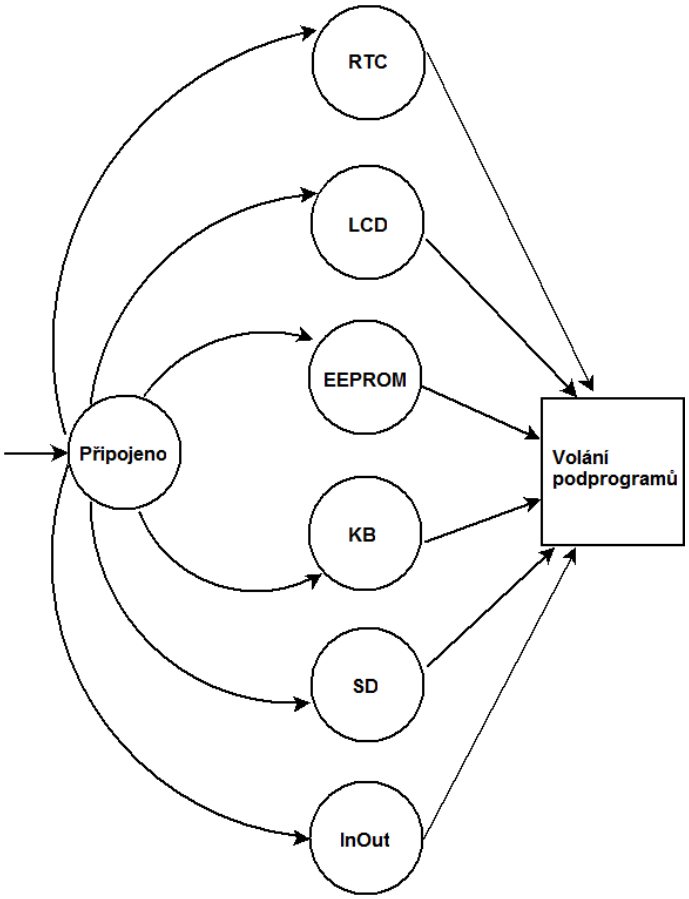
3.5 Komunikační protokol

Jak už bylo zmíněno v podsekci věnující se Java aplikaci(3.4.3) , komunikace s deskou probíhá po sériové lince (fyzické propojení je realizováno USB kabelem).

U firmware desky není vyžadováno, aby vykonával jiné operace než obsluhu komunikace a plnění úkolů přijatých v této komunikaci. Různé operace s různými periferiemi vyžadují proměnlivé množství a typy dat. Mohl bych po sériové lince posílat pakety určité struktury, ty následně ukládat do bufferu a zpracovávat. Jejich analýza by ale byla podobně složitá jako průběh řízení komunikace, který jsem se rozhodl použít já. V mém řešení je firmware stavový automat, který přechází mezi stavy na základě přijatých bytů. Ve stavech, které vyžadují operace s periferiemi, jsou volány podprogramy, které samy řídí komunikaci. Řešení bude podrobněji rozebráno v dalších částech této sekce za pomoci obrázku 3.5.

Byte může nabývat 256 různých hodnot. Takto navržený protokol tedy umožňuje firmwaru ovládat až 256 periferií a s každou z nich provádět až 256 operací. Pokud několik hodnot vyhradíme pro speciální účely, dostaneme se na čísla o něco menší. Bez problému jsou pokryty

všechny stávající periferie a ještě nám zbývá veliký prostor pro případ rozšíření možností desky.



Obrázek 3.5: Stavový automat pro firmware desky

3.5.1 Navázání komunikace

Po restartu deska čeká na příjem inicializačního bytu od počítače, po jeho přijetí je tento byte potvrzen a deska odešle identifikační kód. Poté čeká na potvrzení správnosti kódu od počítače. V případě, že tato akce neproběhne správně, je opakována.

3.5.2 Další průběh komunikace

Pokud se podaří komunikaci navázat, přechází deska do stavu *Připojeno*. Zde čeká na příjem dalšího bytu. Ten udává s jakou periferií se bude pracovat. Jeho přijetí deska potvrzuje odesláním stejného bytu. V následujícím stavu deska znova čeká na přijetí bytu a dle jeho hodnoty se rozhoduje jaký podprogram bude zavolán. Po vykonání podprogramu se deska vrací zpátky do stavu, ze kterého byl tento podprogram zavolán. Aby se deska mohla vrátit

znovu do stavu *Připojeno*, musí jí být poslán byte oznamující konec práce s periferií[1] (ve stavech *RTC, LCD, EEPROM, KB, SD a InOut*).

3.5.3 Detekce chyb v datech

Pro detekci chyb v posílaných blocích dat jsem se rozhodl využít metodu cyklického redundatního součtu. Využiji 16-ti bitový CRC, protože knihovna avr-libc, kterou používám obsahuje funkce pro rychlý výpočet CRC-16.

[1] Přechody vedoucí do stavu *Připojeno* ze všech ostatních stavů nejsou ve stavovém diagramu kvůli přehlednosti uvedeny

Kapitola 4

Realizace

4.1 Konkrétní hodnoty pro komunikační protokol

4.1.1 Parametry přenosu

Parametry přenosu po sériové lince, které jsem zvolil, jsou uvedeny v tabulce 4.1.

Přenosová rychlost	230 400 b/s
Počet datových bitů	8
Počet stopbitů	1
Parita	ne

Tabulka 4.1: Parametry přenosu po sériové lince

4.1.2 Realizace komunikace

Pro komunikaci po sériové lince jsou ve firmware používány funkce ze souboru *UART0_ routines.c* od autora desky[19].

V javovské aplikaci byla použita knihovna RXTX [15]. S její pomocí jsem vytvořil třídu *Komunikace*. Tato třída poskytuje metody pro navázání spojení a komunikaci.

4.1.2.1 Třída Komunikace

Třída *Komunikace.java* obsahuje důležité konstanty pro navázání spojení s deskou. Jedná se o hodnotu inicializačního bytu a kód použitý pro ověření spojení. Tyto konstanty nelze měnit. Spojení s deskou by jinak bez změny firmware nebylo možné. Jsou to tyto konstanty:

```
private final byte initByte = 0xFF;
private final String code = "77504";
```

Třída dále pracuje s objekty typu *SerialPort, InputStream a OutputStream*. Pro uživatele, který bude tuto třídu používat pouze jako rozhraní ke komunikaci, není nutné znát význam

těchto objektů. Důležité je pouze vědět, že za pomoci knihovny RXTX můžeme sériovou linku používat jako vstupně-výstupní proudy.

Dále jsou popsány důležité metody této třídy:

```
public void pripoj(String jmenoPortu) throws NoSuchPortException,
PortInUseException, UnsupportedCommOperationException, IOException
```

Metoda *pripoj* zajišťuje nastavení parametrů přenosu, otevření portu, získání vstupních a výstupních proudů. Její základ je převzat z práce autora desky[19]. Parametrem je jméno sériového portu, typicky řetězec "COMx", kde x zastupuje číslo portu. Metoda vyhazuje vyjímky v těchto případech: port v parametru neexistuje(*NoSuchPortException*), port v parametru je používán jiným procesem(*PortInUseException*), s portem je prováděna nepodporovaná operace(*UnsupportedCommOperationException*) anebo nastala chyba při získávání vstupního nebo výstupního proudu(*IOException*). Parametry přenosu nelze pro aplikaci, která je předmětem této bakalářské práce, měnit. Pokud by někdo tuto třídu chtěl využít pro jiné účely, lze je změnit prostřednictvím konstant objektu *SerialPort* na následujícím řádku:

```
SerialPort.setSerialPortParams(230400, SerialPort.DATABITS_8,
SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
```

```
public ArrayList<String> listPorts()
```

Metoda *listPorts* vrací seznam názvů portů v počítači. Při její tvorbě bylo vycházeno z kódu v článku[11].

```
public boolean pripojDesku() throws IOException
```

Tato metoda zajišťuje navázání spojení s deskou v souladu s postupem popsaným v 3.5.1. Její součástí je implementace timeoutu. V případě, že se spojení podaří navázat vrátí *true*. Pokud se spojení nedaří 5 sekund navázat, vrátí *false*.

```
public int prijmi() throws IOException
```

Vrací přijaté číslo v rozsahu 0 - 255. Navrátový typ je *int*, protože v Javě neexistuje typ *unsigned byte* a použití standartního *byte* se znaménkem by komplikovalo práci s deskou.

```
public void posli(byte b) throws IOException
public void posli(int i) throws IOException
public void posli(char znak) throws IOException
```

Metoda *posli* slouží pro odeslání bytu po sériové lince. Byla přetížena i pro datové typy *int* a *char* hlavně z toho důvodu, že velikost datového typu *char* je v programovacím jazyce Java 16 bitů. Já ale používám pouze standartní znaky tvořené spodními 8 bity. Při odesílání typu *int* by měl mít programátor na paměti, že může odesílat pouze čísla v rozsahu 0 - 255. Přesto jsou v metodě větší čísla redukována, viz. příklad:

```
public void posli(int i) throws IOException {
        if(i > 255) out.write(255);
        else if(i < 0)out.write(0);
        else out.write(i);
}
```

Pozor, jako první věc před odesíláním nebo příjímaním dat je nutné zavolat metodu *pripoj* (volá se pouze jednou, na začátku).

4.1.3 Hodnoty pro stavový automat ve firmware

Stavy automatu jsou v kódu firmware reprezentovány proměnnou *stav* typu *int*. Přehled hodnot proměnné a odpovídajících stavů je uveden v tabulce 4.2.

Hodnota	**Stav**
0	Připojeno
1	LCD
2	RTC
3	KB
4	EEPROM
5	SD
6	InOut

Tabulka 4.2: Reprezentace stavu proměnnou

4.1.3.1 Tabulka přechodů

Po úspěšném navázání spojení s počítačem přechází deska do stavu *Připojeno*. V tomto stavu čeká na přijatý byte a podle něj se rozhoduje do jakého stavu přejde. To je popsáno v tabulce přechodů 4.3.

Hodnota bytu	**Následující stav**
200	LCD
199	RTC
198	KB
197	EEPROM
196	SD
195	InOut
jiný byte	Připojeno

Tabulka 4.3: Přechody ze stavu *Připojeno*

4.1.3.2 Volání podprogramů

Pokud je deska v některém ze stavů přislušejících periferiím, tzn. *LCD, RTC, KB, EEPROM, SD, InOut*, volá se na základě přijatého bytu určitý podprogram nebo se vrací do stavu *Připojeno*. Tento postup včetně významů jednotlivých podprogramů je přehledně zobrazen v tabulkách níže.

- LCD displej

Hodnota bytu	**Volaný podprogram**	**Význam podprogramu**
190	zobrazData_LCD()	zobrazí na LCD přijatý řetězec
189	smaz_displej()	smaže displej
188	animace()	zobrazí animaci
187	smazaniRadku()	smaže daný řádek displeje
186	zobrazByte()	rozsvítí body segmentu LCD určené přijatým bytem
185	nastavitRadek()	nastaví kurzor na zadaný řádek
185	nastavitSloupec	nastaví kurzor na zadaný sloupec

Tabulka 4.4: Volané podprogramy ve stavu *LCD*

Byte pro návrat do stavu *Připojeno*: 181

- RTC

Hodnota bytu	**Volaný podprogram**	**Význam podprogramu**
180	posliDatum()	odešle datum
179	posliCas()	odešle čas
178	prijmiDatum()	nastaví datum
177	prijmiCas()	nastaví čas

Tabulka 4.5: Volané podprogramy ve stavu *RTC*

Byte pro návrat do stavu *Připojeno*: 171

- Klávesnice (KB)
  Ve stavu *KB* se volá pouze jediný podprogram *posliZnak_ kb()* při přijmutí bytu *170*. Tento podprogram vybere znak z bufferu klávesnice a odešle ho.
  Byte pro návrat do stavu *Připojeno*: 161

- EEPROM
  Byte pro návrat do stavu *Připojeno*: 151

Hodnota bytu	Volaný podprogram	Význam podprogramu
160	posliZnaky_EEPROM()	odešle určitý počet znaků přečtených z dané adresy
159	prijmiZnaky_EEPROM()	přijme a zapíše určitý počet znaků na danou adresu
158	posliStranku_EEPROM()	přečte a odešle stránku určitého čísla
157	prijmiStranku_EEPROM()	přijme a zapíše stránku určitého čísla

Tabulka 4.6: Volané podprogramy ve stavu *EEPROM*

- SD karta

Hodnota bytu	Volaný podprogram	Význam podprogramu
150	createDirSD()	vytvoří adresář v zadaném umístění
149	copyFile()	zkopíruje soubor z počítače na SD kartu
148	isFile()	zjistí jestli je zadaná položka soubor
147	sendFile()	odešle soubor z SD karty do počítače
145	sendSpace()	odešle kapacitu SD karty
144	sendFreeSpace()	odešle údaj o volném místu
143	sendDirContent()	odešle názvy položek v daném adresáři
142	deleteItem()	smaže danou položku

Tabulka 4.7: Volané podprogramy ve stavu *SD*

Byte pro návrat do stavu *Připojeno*: 141

- Rozšiřující konektor (InOut)

Hodnota bytu	Volaný podprogram	Význam podprogramu
140	set_s1rel1L()	sepne levé relé
139	res_s1rel1L()	rozepne levé relé
138	set_s1rel2L()	sepne pravé relé
139	res_s1rel2L()	rozepne pravé relé
135	sendStateLeft()	odešle informace o stavu levého vstupu
134	sendStateRight()	odešle informace o stavu pravého vstupu

Tabulka 4.8: Volané podprogramy ve stavu *InOut*

Byte pro návrat do stavu *Připojeno*: 131

Podprogramy *set_s1rel1L, set_s1rel2L, res_s1rel1L a res_s1rel2L* jsou převzaté z knihovny *output_routines.c* od autora desky[19].

4.2 Firmware

Při tvorbě firmware jsem využil knihovny *UART0_routines, UART2_LCD, RTC_routines, EEPROM_routines, kb_routines a output_routines*, které vytvořil autor desky[19]. Dále jsem použil standartních prostředků knihovny *avr-libc*, která je součástí balíku WinAVR[18].

Pro práci s SD kartou jsem nakonec použil knihovnu od Rolanda Riegela[14], protože při testování druhé knihovny od Elm Chana[9] se mi nedařilo správně číst jména souboru a odesílat soubory přes UART.

Důležité části firmware jsou popsány v tabulkách v 4.1.3.2.

Firmware zabírá 34,19 KB programové a 1,29 KB datové paměti procesoru.

4.2.1 Konfigurace a použití Riegelovy knihovny

Riegelova knihovna není napsána přímo pro procesor AVR ATmega2560, ale po menší úpravě se dá použít i pro tento procesor. ATmega2560 má stejné elektrické zapojení pro SPI jako ATmega64, ATmega128 a ATmega169, pro které je knihovna připravena. Stačí přidat jediný řádek do konfiguračního souboru *sd_raw_config.h*:

```
...
#elif defined(__AVR_ATmega64__) || \
      defined(__AVR_ATmega128__) || \
      defined(__AVR_ATmega169__) || \
      defined(__AVR_ATmega2560__)                    //pridany radek
    #define configure_pin_mosi() DDRB |= (1 << DDB2)
    #define configure_pin_sck() DDRB |= (1 << DDB1)
...
```

Dále je potřeba v souboru *sd_raw_config.h* zapnout podporu SDHC karet, podporu zápisu dat a podporu bufferování.

```
#define SD_RAW_SDHC 1
#define SD_RAW_WRITE_SUPPORT 1
#define SD_RAW_WRITE_BUFFERING 1
```

V souboru *partition_config.h* je třeba nadefinovat počet diskových oddílů:

```
#define PARTITION_COUNT 1
```

V souboru *fat_config.h* je potřeba zapnout podporu zápisu dat a podporu dlouhých jmen souboru a adresářů:

```
#define FAT_WRITE_SUPPORT 1
#define FAT_LFN_SUPPORT 1
```

Následující příklad poodhaluje práci s knihovnou včetně inicializace SD karty. Konkrétně se jedná o čtení 128 bytů dat ze souboru.

```
/* potřebné datové struktury */
struct partition_struct* partition;
struct fat_fs_struct* fs;
struct fat_dir_entry_struct directory;
struct fat_file_struct* ff;
unsigned char data[128];

/*inicializace SD karty */
sd_raw_init();
partition = partition_open(sd_raw_read,sd_raw_read_interval,
 sd_raw_write,sd_raw_write_interval,0);
fs = fat_open(partition);

/* otevření souboru */
fat_get_dir_entry_of_path(fs, "/soubor.txt", &directory);
ff = fat_open_file(fs, &directory);

/* čtení ze souboru a zavření souboru */
fat_read_file(ff, data, 128);
sd_raw_sync();
fat_close_file(ff);

/* ukončení práce s file systémem */
fat_close(fs);
partition_close(partition);
```

Dokumentaci ke všem nabízeným funkcím lze nalézt na internetových stránkách projektu[14].

4.3 Knihovny v Javě

Pro každou z periferií byla vytvořena samostatná třida, poskytující metody pro jejich ovládání. Při tvorbě instance třídy se jejímu konstruktoru musí předat objekt typu *Komunikace*. Součástí všech tříd jsou následující dvě metody:

`public boolean startCom() throws IOException` - Tuto metodu je nutné volat před tím než započnete volání dalších metod zajišťujících práci s danou periferiií. Vrací *true* pokud se podařilo periferii inicializovat a je možné s ní pracovat.

`public void ukoncitPraci() throws IOException` - Tuto metodu je nutné volat vždy po skončení práce s danou periferií. Pokud tak neučiníte, není možné začít pracovat s jinou periferií.

Některé třídy používají třídu *TestCRC16*, která zajišťuje počítání cyklického redundantního součtu. Její základ byl převzat z[7]. Konkrétně je používáno XMODEM-CRC.

Vetšina metod vyhazuje vyjímku *IOException*. Ta nastává pokud se vyskytne nějaká chyba s odesíláním nebo přijímáním dat, potažmo s výstupními nebo vstupními proudy.

Dále jsou popsány všechny vytvořené třídy.

4.3.1 LCD.java

`public void zobrazData(String s, int radek) throws IOException` - Slouží pro zobrazení řetězce předaného v parametru na určitém řádku. Je třeba pamatovat na to, že řádek displeje dokáže zobrazit pouze 21 znaků a má 8 řádku. Před voláním metody je proto nutné zajistit správný rozsah parametrů.

`public void smazat() throws IOException` - Smaže displej.

`public void smazatRadek(int radek) throws IOException` - Smaže řádek předaný parametrem. Znovu je potřeba zajistit parametr *radek* v rozsahu 1 - 8.

`public void zobrazByte(byte b) throws IOException` - Na displeji je možné rozsvítit sloupec 1 - 8 bodů. To jaké body budou rozsvíceny určuje parametr *b*. MSB rozsvicí nejspodnější bod, LSB nejhořejší. Pokud bude v parametru 0xFF rozsvítí se všech 8 bodů, v případě 0x01 se rozsvítí pouze horní bod, při 0x03 horní dva body a tak dále.

`public void nastavRadek(int radek) throws IOException` - Nastaví kurzor na daný řádek. Parametr *radek* musí být v rozsahu 1 - 8.

`public void nastavSloupec(int sloupec) throws IOException` - Nastaví kurzor do daného sloupce. Parametr *sloupec* musí být v rozsahu 0 - 127.

4.3.2 RTC.java

`public String getDate() throws IOException` - Získá datum z obvodu reálného času. Získané datum je ve formátu dd/mm/rrrr.

`public String getTime() throws IOException` - Získá čas z RTC. Získaný čas je ve formátu hh:mm:ss.

`public void setDate(int den, int mesic, int rok) throws IOException` - Nastaví datum RTC. Je třeba dbát na správné rozsahy parametrů. Parametr *den* musí být v rozsahu 1 - 31, *mesic* v rozsahu 1 - 12.

`public void setTime(int hodina, int minuta) throws IOException` - Nastaví čas RTC. Parametr *hodina* musí být v rozsahu 0 - 23, *minuta* v rozsahu 0 - 59.

4.3.3 EEPROM.java

`public char[] prectiZnaky(int ha, int la, int pocetZnaku) throws IOException` - Přečte určitý počet znaků(bytů) z dané adresy, *ha* je horních 8 bitů adresy, *la* je spodních 8 bitů adresy. Je třeba dbát na správný rozsah adres. Minimální adresa je 0x0000, tzn. *ha = 0x00, la = 0x00*. Maximální adresa je 0x7FFF, tzn. *ha = 0x7F, la = 0xFF*. Parametr *pocetZnaku* musí být v rozsahu 0 - 64. V případě, že se čtení nepovedlo vrací *null*.

`public boolean zapisZnaky(int ha, int la, String s) throws IOException` - Zapíše na danou adresu řetězec. Pravidla pro adresu jsou stejná jako v případě metody *prectiZnaky*. Řetězec *s* může být dlouhý maximálně 64 znaků. V případě úspěšného zápisu vrací *true*.

`public char[] prectiStranku(int cislo) throws IOException` - Přečte a vrátí stránku(64 byte) z paměti. Parametrem je číslo stránky nabývající hodnot od 0 do 511. Pokud se čtení nepovede, vrátí *null*.

`public boolean zapisStranku(int cislo, String s) throws IOException` - Zapíše stránku daného čísla. Číslo stránky nabývá hodnot od 0 do 511. Řetězec *s* může být dlouhý maximálně 64 znaků. Vrací *true* v případě úspěšného zápisu.

4.3.4 Keyboard.java

`public char getChar() throws IOException` - Vrátí znak vybraný z bufferu klávesnice.

`public String getString() throws IOException` - Vybere celý buffer klávesnice a vrátí ho jako řetězec.

4.3.5 SD_card.java

Jako oddělovač v cestách funguje znak "/".

`public int createDir(String nazev, String cesta) throws IOException` - Vytvoří adresář daného názvu v zadaném umístění(parametr *cesta*). Vrací 1, pokud se podaří vytvořit adresář.

`public int copyFile(File s, String umisteni) throws IOException` - Zkopíruje soubor z počítače předaný v parametru na zadané umístění. Vrací 1 v případě úspěchu, 0 pokud je vyčerpaná kapacita SD karty, 2 pokud se nepodařilo zkopírovat soubor a 3 pokud nastala chyba v komunikaci.

`public int copyFile(File s, String umisteni, Label status) throws IOException` - Přetížené metodě *copyFile* můžete předat jako parametr grafický objekt typu *Label (Java.awt.Label)*, na tom pak bude zobrazován stav kopírování, z kolika procent je již soubor zkopírován.

`public boolean isFile(String cesta) throws IOException` - Vrací *true* pokud je položka v parametru soubor. Paremetrem *cesta* se předává cesta k položce včetně jejího názvu.

`public int deleteItem(String cesta) throws IOException` - Smaže danou položku. Paremetrem *cesta* se předává cesta k položce včetně jejího názvu. Vrací 1 pokud se podaří položku smazat, 3 pokud nastane chyba v komunikaci.

`public int getFile(String name, String path, String sd_path) throws IOException` - Zkopíruje soubor z SD karty do počítače. Cesta k souboru, který chceme kopírovat, včetně jeho jména je v parametru *sd_path*. Soubor bude uložen pod názvem *name* v umístění daném parametrem *path*. Vrací 1 v případě úspěchu, 2 v případě neúspěchu, 3 pokud nastane chyba v komunikaci.

`public int getFile(String name, String path, String sd_path, Label status) throws IOException` - Podobně jako metodě *copyFile* i metodě *getFile* lze v parametr předat objekt typu Label, na který bude vypisována informace o stavu kopírování souboru.

`public int getSpace() throws IOException` - Vrátí kapacitu SD karty v bytech.

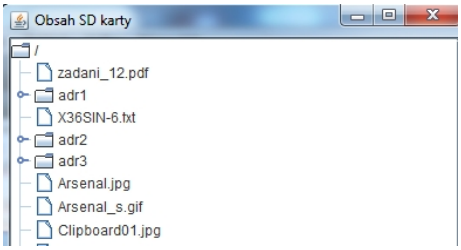
`public int getFreeSpace() throws IOException` - Vrátí volné místo na SD kartě v bytech.

`public ArrayList<String> getDirContent(String cesta) throws IOException` - Vrátí seznam se jmény položek obsažených v adresáři na daném umístění. Vrací *null* pokud nastane chyba v komunikaci.

`public boolean getSDTree(ArrayList<String> root, String cesta, DefaultMutableTreeNode koren) throws IOException` - Tato metoda je užitečná pro aplikace s GUI. Parametr *root* je seznam položek v kořenovém adresáři, *cesta* je zde pro účely rekurze, při volání metody se jako *cesta* použije *"/"*, *koren* je objekt, který se po zavolání této metody použije jako parametr pro konstruktor třídy JTree, což je komponenta pro vizualizaci stromů. Celé použití této metody pak může vypadat například takto:

```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("/");
sd.getSDTree(sd.getDirContent("/"), "/", root);
tree = new JTree(root);
```

V kódu výše je *sd* instance třídy SD_card. Pokud takto získaný JTree přidáte do nějakého kontejneru a necháte zobrazit, může to vypadat podobně jako na obrázku 4.1.



Obrázek 4.1: JTree získaný za pomoci metody *getSDTree*

4.3.6 InOut.java

Předpokládá se, že modul s výstupy je zapojen v levém konektoru - J13 a modul se vstupy je zapojen v pravém konektoru - J14.

`public void sepniRele1() throws IOException` - Sepne levé relé.

`public void rozepniRele1() throws IOException` - Rozepne levé relé.

`public void sepniRele2() throws IOException` - Sepne pravé relé.

`public void rozepniRele2() throws IOException` - Rozepne pravé relé.

`public boolean stavLevyVstup() throws IOException` - Vrací *true* pokud je na levém vstupu napětí(vstup je v logické 0).

`public boolean stavPravyVstup() throws IOException` - Vrací *true* pokud je na pravém vstupu napětí(vstup je v logické 0).

4.4 Ukázka práce se vzniklými knihovnami

Pro lepší pochopení toho, jak se vzniklými prvky pracovat jsem se rozhodl uvést krátký příklad. Cílem programu je vypsat volné místo na SD kartě, zkopírovat do adresáře *test* na

SD kartě soubor *test.txt* z počítače, zobrazit na 3. řádku displeje nápis *TESTOVACI TEXT* a sepnout levé relé.

```
/* navázání spojení */
Komunikace kom = new Komunikace();
kom.pripoj("COM1");
kom.pripojDesku();

/* vytvoření objektů pro periferie */
SD_card sd = new SD_card(kom);
LCD displej = new LCD(kom);
InOut io = new InOut(kom);

/* práce s SD kartou */
sd.startCom();
System.out.println("Volné místo:" + sd.getFreeSpace() + "bytes");
File f = new File("test.txt");
sd.copyFile(f, "/test/");
sd.ukoncitPraci();

/* práce s LCD */
displej.startCom();
displej.zobrazData("TESTOVACI TEXT", 3);
displej.ukoncitPraci();

/* práce s výstupy */
io.startCom();
io.sepniRele1();
io.ukoncitPraci();
```

4.5 Ukázka zobrazení dat na displeji

Na obrázku 4.2 je ilustrováno použití několika metod pro práci s displejem. Po navázání spojení s deskou byl vykonán následující kód:

```
LCD displej = new LCD(kom);
displej.startCom();
displej.nastavRadek(4);
displej.nastavSloupec(60);
displej.zobrazByte(0xFF);
displej.zobrazByte(0x7E);
displej.zobrazByte(0x3C);
displej.zobrazByte(0x18);
displej.zobrazByte(0x18);
displej.zobrazByte(0x18);
```

```
displej.zobrazByte(0x3C);
displej.zobrazByte(0x7E);
displej.zobrazByte(0xFF);
```

Obrázek 4.2: Zobrazení dat na displeji

4.6 Demonstrační aplikace

Demonstrační aplikace využivá vzniklé knihovní prvky a demonstruje možnosti periferní desky. Díky jednoduchému GUI si ovládání desky může vyzkoušet i laik.

Rozhodl jsem pro tvorbu GUI nepoužít návrháře, protože jsem s takovým nástrojem neměl zkušenosti. To se posléze ukázalo jako celkem nešťastná volba, protože čas strávený psaním kódu pro GUI byl zřejmě stejný jako čas, který bych strávil studiem práce s návrhářem. Výsledné GUI sice nepůsobí veliký "wow efekt", ale myslím, že je přehledné a pro danou aplikaci dostačující.

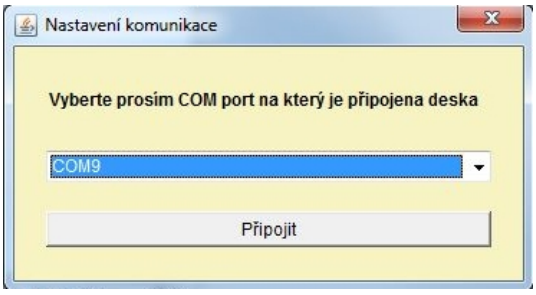
Aplikace je ošetřena proti nevhodným vstupům, upozorňování uživatele je realizovánou systémem dialogů. Stejně je uživatel informován o odchycených vyjímkách. Správné chování aplikace jsem testoval já a také můj kamarád Petr, viz. kapitola 5.

4.6.1 Popis GUI

GUI bylo logicky rozděleno na hlavní okno (obrázek 4.3) a 6 vedlejších oken. Hlavní okno je jednoduché, obsahuje 6 tlačítek pro otevírání vedlejších oken. Každé vedlejší okno odpovídá jedné periferii. V jedné chvíli lze ovládat pouze jednu periferii, po otevření vedlejšího okna je hlavní okno skryto a zobrazí se až po uzavření vedlejšího okna.

Po startu aplikace je zobrazen dialog *Nastavení komunikace* (obrázek 4.4), který nabídne uživateli na výběr všechny COM porty v počítači. Od uživatele se očekává, že vybere ten, ke kterému je připojena deska. Po stisku tlačítka *Připojit* se aplikace pokusí navázat spojení s deskou. Pokud se navázaní spojení podaří, je o tom uživatel informován a zobrazí se hlavní okno. V opačném případě je uživatel také informován, ale aplikace je ukončena.

Obrázek 4.3: Hlavní okno aplikace



Obrázek 4.4: Okno nastavení komunikace

Kapitola 5

Testování

V této fázi jsem se zaměřil hlavně na uživatelské testování, protože se mi jevilo jako velmi složité vytvořit nějaké automatické testy pro dvě spolupracující zařízení. Jediná možnost verifikace dat bylo jejich zobrazení na displeji, případně jejich porovnávání s konstantami v paměti. Při ladění firmware jsem použil nástroje v AVR Studiu.

První testování probíhalo již při seznamování s deskou, kdy jsem ověřoval funkčnost knihoven dodaných k desce. Zjistil jsem jediný závažný nedostatek, LCD displej zobrazuje nekorektně znaky G a H. Bohužel jsem neměl k dispozici zdrojový kód k firmware displeje, proto jsem tuto chybu nemohl napravit.

Další testování probíhalo souběžně s tvorbou firmware a knihovních prvků v javě. Některé funkce šlo testovat pouhým pohledem (operace s LCD displejem, spínání relé, získávání stavu vstupů, vybírání bufferu klávesnice), další byly ověřovány jinak. Například při nastavování času RTC nebo zápisu dat do EEPROM byla správná funkce ověřována zpětnými operacemi(čtení času nebo dat) a zobrazením dat na displeji. Při testování správné práce s SD kartou bylu využívána čtečka karet v počítači. Při kopírování souboru na SD kartu jsem porovnával púvodní soubor a zkopírovaný soubor programem *cmp* dostupným z příkazové řádky OS Windows, který soubory porovnává byte po bytu. Stejně jsem postupoval i při kopírování souborů z SD karty. Ostatní funkce byly ověřeny porovnáním údajů z počítače(vytváření stromové struktury, kapacita karty, mázání položek, vytváření adresářů). V průběhu testování se bohužel ukázalo, že deska nespolupracuje se všemi kartami. Zatímco práce s SD kartou Transcend 2 GB a MMC kartou Pretec 256 MB byla v pořádku, SD kartu SanDisk 2 GB se nepodařilo ani iniciovat. Více karet jsem bohužel neměl k dispozici. Další problém, který se objevil je rychlost kopírování souborů na SD kartu. Zkopírovat soubor veliký 50 kB na SD kartu trvá asi 3 a půl minuty, zatímco jeho zkopírování z SD karty trvá asi 20 sekund. Tento problém se mi nepodařilo vyřešit, zdá se, že je zakořeněn již v použité knihovně pro SD kartu. Testování probíhalo na dvou počítačích. Prvním byl stolní počítač s OS Windows 7 64-bit, druhým byl notebook s OS Windows Vista 32-bit.

5.1 Uživatelské testování

5.1.1 Závěrečné testování aplikace

Jako tvůrce aplikace jsem mohl otestovat její potenciální slabiny. Provedl jsem několik testů při kterých jsem se choval jako nevyzpytatelný uživatel.

5.1.1.1 Navazování spojení

Při navazování spojení jsem záměrně vybral špatný COM port. Aplikace správně odpověděla chybovým dialogem, ale nesprávným typem dialogu, který po odsouhlasení neukončí aplikaci. Musel jsem tuto chybu odstranit tak, že jsem na všech místech v aplikaci, kde byl nesprávně použit dialog *WarnDialog*, použil na místo něj dialog *ErrDialog*.

Ve druhém pokusu jsem sice vybral správný COM port, ale desku jsem nerestartoval do počátečního stavu. Aplikace správně po 5 sekundách odpověděla timeoutem a ukončila se.

Nakonec jsem provedl navázání spojení v souladu se standartním postupem. Aplikace správně odpověděla dialogem o navázání spojení a mohl jsem testovat dál.

Během testů jsem narazil ještě na jeden problém. Dialog pro navazování spojení bylo možné zavřít standartním zavíracím tlačítkem, spojení nebylo navázáno a aplikace přesto zobrazila hlavní okno. Dialog *KomDialog* bylo tedy nutné přeprogramovat.

5.1.1.2 Odpojení kabelu za běhu aplikace

V tomto testu jsem ověřoval jak se aplikace vyrovná s odpojením kabelu za běhu. Vyzkoušel jsem to u všech periferií. Aplikace vždy správně odpověděla chybovým dialogem a ukončila se.

5.1.1.3 Odolnost aplikace vůči nestandartním vstupům

Jediné místo, kde uživatel může škodit nestandartním vstupem je okno pro práci s EEPROM pamětí. Zde se nachází několik textových polí, na kterých by aplikace mohla ztroskotat. Testoval jsem tedy jak se aplikace zachová při zadávání nepovolených hodnot do těchto polí. Aplikace nezklamala a správně mě vždy informovala o provedené chybě. Jednalo se o chyby vkládání znaků místo čísel, zadávání hodnot mimo požadovaný rozsah a zadání příliš dlouhého řetězce.

5.1.1.4 Správná funkce vstupů

Správná funkce detekce stavu vstupů, byla testována tak, že jsem vstupy připojil ke zdroji a nechal je spínat relátkem z modulu výstupů. Mohl jsem tak okamžitě testovat, jestli při sepnutí relátka detekuje aplikace sepnutí vstupu. Testy proběhly v pořádku.

5.1.2 Testování nezasvěceným uživatelem

Do testování jsem zapojil také svého kamaráda Petra, abych získal objektivní názor uživatele. Pomohl jsem mu s uvedením desky do provozu. Poté dostal seznam úkolů a po jejich splnění mi měl odevzdat jeho poznámky k aplikaci. Pro účely testování jsem mu zapůjčil svůj notebook s OS Windows Vista 32-bit.

5.1.2.1 Seznam úkolů

1. Pokusit se o navázání spojení s deskou
2. Vyzkoušet si práci s displejem - zobrazování textu, mazání řádků, zobrazování samostatných skupin bodů.
3. Vyzkoušet nastavování a zjišťování data a času na RTC.
4. Ověřit správnost zobrazování znaků tištěných na klávesnici.
5. Vyzkoušet čtení a zapisování do EEPROM
6. S pomocí čtečky si vytvořit adresářovou strukturu na SD kartě, nakopírovat na SD kartu nějaké soubory. Zkusit kopírování souborů z/na SD kartu. Vyzkoušet správné zobrazování adresářové struktury, mazání položek, vytváření adresářů. Ověřit informace o kapacitě a volném místě na SD.
7. Ověřit správné spínání výstupů a detekování stavu vstupů.

5.1.2.2 Poznámky k aplikaci

Dostal jsem od Petra následující zpětnou vazbu:

Když chci zobrazit text na displeji na řádku, na kterém už něco je zobrazeno, měl by se tenhle řádek nejdřív smazat a ne přepisovat původní text. Displej špatně vykresluje znak H. Při nastavování času bys možná měl použít textová pole místo rozbalovacích seznamů. U nastavení datumu se špatně zobrazuje tlačítko. Kopírování souborů na SD kartu je hodně pomalé.

5.1.2.3 Zapracování připomínek

Petr při testování narazil většinou na problémy, se kterými jsem se už dříve potýkal já. Některé z nich pro mě nejsou řešitelné(nesprávné zobrazení některých znaků na displeji, rychlost kopírování souborů), některé jsou pouze subjektivním názorem na GUI. Upraveno tak bylo pouze chybně zobrazované tlačítko.

Kapitola 6

Závěr

V rámci této práce se mi podařilo vytvořit sadu tříd v programovacím jazyce Java s jejichž pomocí lze ovládat univerzální periferní desku propojenou s počítačem USB kabelem. Vytvořil jsem demonstrační aplikaci s GUI, která tyto třídy využívá a umožňuje i laikovi vyzkoušet si možnosti desky. Výsledkem práce je kromě toho firmware pro desku a návrh jednoduchého komunikačního protokolu.

Práce na tomto projektu byla zajímavá, pro mě ale trochu náročná, protože sem neměl větší zkušenosti s mikroprogramováním. V první fázi jsem se tedy seznamoval s deskou, hledal vhodné nástroje pro implementaci, zkoušel tvořit jednoduché programy pro ovládání jednotlivých periferií a hledal knihovnu pro práci s SD kartou.

V další fázi probíhal návrh komunikačního protokolu, tvorba firmware a knihovních prvků pro ovládání periferií desky v souladu s funkčními požadavky definovanými v kapitole 2. Tyto požadavky jsem splnil.

Nakonec jsem vytvořil demonstrační aplikaci s grafickým uživatelským prostředím, při jejíž tvorbě jsem využil výsledky mé předchozí práce.

Myslím, že mě práce obohatila o spoustu nových znalostí z oblasti mikroprogramování, vývojových nástrojů a knihoven pro mikrokontroléry.

Vytvořená sada knihovních prvků se dá do budoucna rozšiřovat. Jako dobré vylepšení mi připadá třeba nástroj pro kreslení grafů a obrázků na LCD displeji. Dále lze rozšířit funkcionalitu obvodu reálného času o nastavování alarmu a timeru. K desce existuje také modul ethernetu, pro který by mohla být vytvořena ovládací třída a upraven firmware.

Literatura

[1] STM32F103 ARM Cortex M3 Development Board. Dostupné z: <http://www.gravitech.us/starmcom3deb.html>. stav z 24. 4. 2011.

[2] AVRco, . Dostupné z: <http://www.e-lab.de/AVRco/index_en.html>. stav z 22. 4. 2011.

[3] Hardwarové pomůcky pro podporu vývoje aplikací s mikrokontroléry ATMEL AVR, . Dostupné z: <http://hw.cz/teorieapraxe/programovani/art3679-hardwarove-pomucky-pro-podporu-vyvoje-aplikaci-s-mikrokontrolery-a>. stav z 3. 3. 2011.

[4] AVR Studio 4, . Dostupné z: <http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725>. stav z 22. 4. 2011.

[5] BASCOM-AVR. Dostupné z: <http://www.mcselec.com/>. stav z 22. 4. 2011.

[6] CodevisionAVR. Dostupné z: <http://www.codevision.be/>. stav z 22. 4. 2011.

[7] CRC-CCITT (0xFFFF) function. Dostupné z: <http://stackoverflow.com/questions/5139480/crc-ccitt-0xffff-function>. stav z 5. 5. 2011.

[8] Datasheet AVR ATmega2560. Dostupné z: <http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf>. stav z 22. 4. 2011.

[9] FatFs Generic FAT File System Module. Dostupné z: <http://elm-chan.org/fsw/ff/00index_e.html>. stav z 24. 4. 2011.

[10] KEIL Tools. Dostupné z: <https://www.keil.com/>. stav z 24. 4. 2011.

[11] Enumerate hardware ports in Java. Dostupné z: <http://thomascannon.net/misc/java_portlist/>. stav z 30. 4. 2011.

[12] mikroC PRO for AVR. Dostupné z: <http://www.mikroe.com/eng/products/view/228/mikroc-pro-for-avr/>. stav z 22. 4. 2011.

[13] Ponyprog - sériový programátor. Dostupné z: <http://www.lancos.com/prog.html>. stav z 24. 4. 2011.

[14] Roland Riegel's MMC/SD/SDHC card library. Dostupné z: <http://www.roland-riegel.de/sd-reader/>. stav z 24. 4. 2011.

[15] Open source knihovna RXTX, . Dostupné z: <http://rxtx.qbang.org>. stav z 24. 4. 2011.

[16] RXTX pro 64-bitové operační systémy, . Dostupné z: <http://www.cloudhopper.com/opensource/rxtx/>. stav z 24. 4. 2011.

[17] SD/SDHC Card Interfacing with ATmega8 /32 (FAT32 implementation). Dostupné z: <http://www.dharmanitech.com/2009/01/sd-card-interfacing-with-atmega8-fat32.html>. stav z 24. 4. 2011.

[18] WinAVR. Dostupné z: <http://winavr.sourceforge.net/>. stav z 22. 4. 2011.

[19] NOUZáK, J. *Konfigurovatelný řídicí modulární systém*. 2010. (Diplomová práce).

Příloha A

Seznam použitých zkratek

API Application Programming Interface
CRC Cyclic Redundancy Check
EEPROM Electrically-Erasable Programmable Read-Only Memory
GCC GNU C Compiler
GUI Graphical User Interface
IDE Integrated Development Environment
ISP In System Programming
JTAG Joint Test Action Group
LCD Liquid Crystal Display
LED Light Emitting Diode
MMC Multimedia Card
OS Operating System
RTC Real Time Clock
SD Secure Digital
SDHC Secure Digital High Capacity
SPI Serial Peripheral Interface
SRAM Static Random Access Memory
UART Universal Asynchronous Receiver Transmitter
USB Universal Serial Bus
VGA Video Graphics Array

Příloha B

Instalační a uživatelská příručka

B.1 Překlad a spuštění aplikace

Pro překlad a spuštění aplikace je nutné mít nainstalované NetBeans IDE a Java Runtime Environment(JRE).

B.1.1 Instalace knihovny RXTX

1. Stáhněte ze stránek[16] knihovnu pro váš operační systém a rozbalte archiv. Knihovnu také najdete na přiloženém CD v adresáři *RXTX*
2. Najděte adresář pro JRE - typicky *C:\Program Files\Java\jre*
3. V tomto adresáři nakopírujte do podadresáře *\lib\ext* soubor *RXTXcomm.jar*(najdete ho v adresáři stažené knihovny) a do podadresáře *\bin* soubory *rxtxSerial.dll a rxtxParallel.dll*

B.1.2 Spuštění aplikace

1. Otevřete v NetBeans projekt z přiloženého CD(*\src\project\Win32* nebo *\src\project\Win64*)
2. Přeložte a spusťte aplikaci
3. Pokud projekt nepůjde přeložit, zkuste vytvořit nový projekt, zkopírovat do něj package z projektu na CD a přidejte do projektu knihovnu *RXTXcomm.jar* z adresáře stažené knihovny

B.2 Uživatelská příručka

B.2.1 Před zahájením komunikace

1. Připojte k desce všechny periferie, které zamýšlíte používat - klávesnici, SD kartu, modul LCD displeje, moduly vstupů a výstupů

2. Připojte zdroje napájení
3. Propojte desku s počítačem USB kabelem
4. Stiskněte tlačítko reset(blízko baterie)

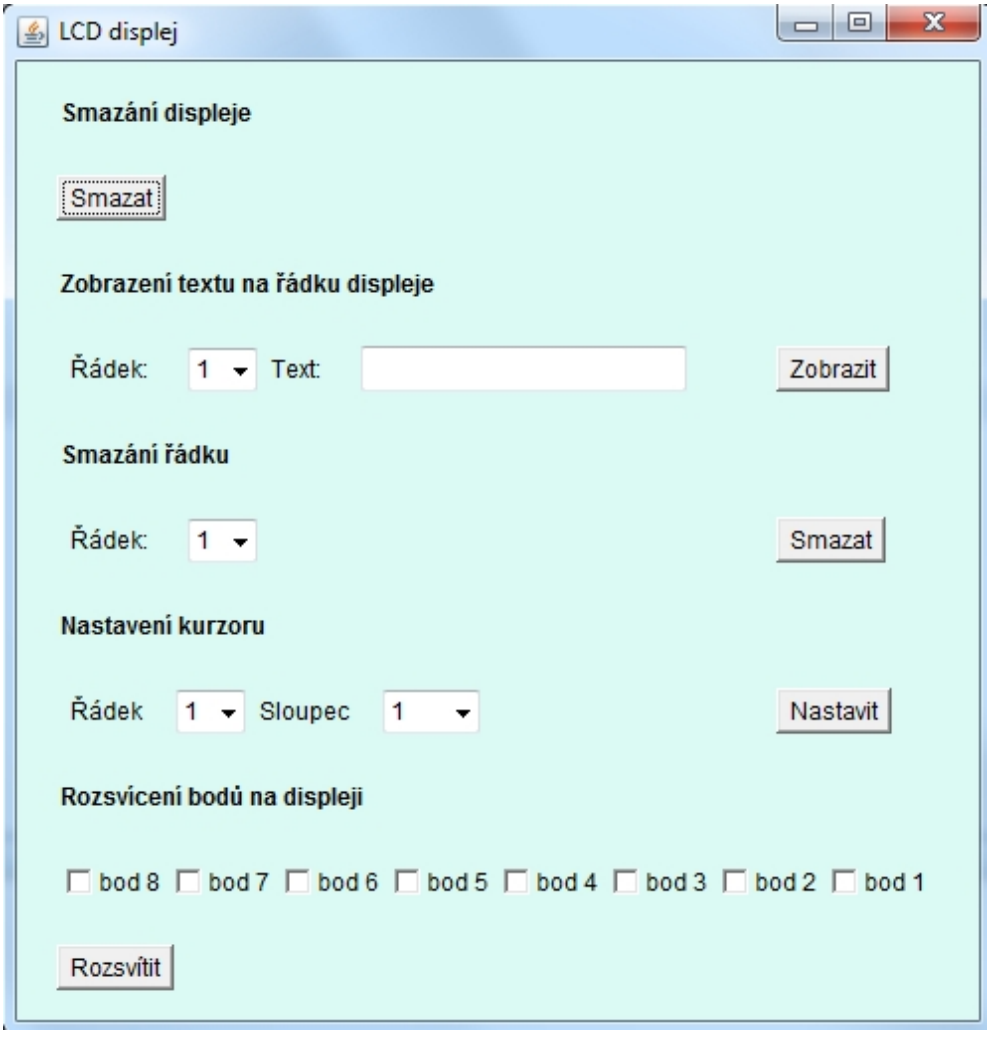
B.2.2 Ovládání aplikace

GUI aplikace by mělo být dostatečně intuitivní, přesto některé postupy nebo chování nemusí být úplně jasné. Vytvořil jsem seznam užitečných tipů pro ovládání.

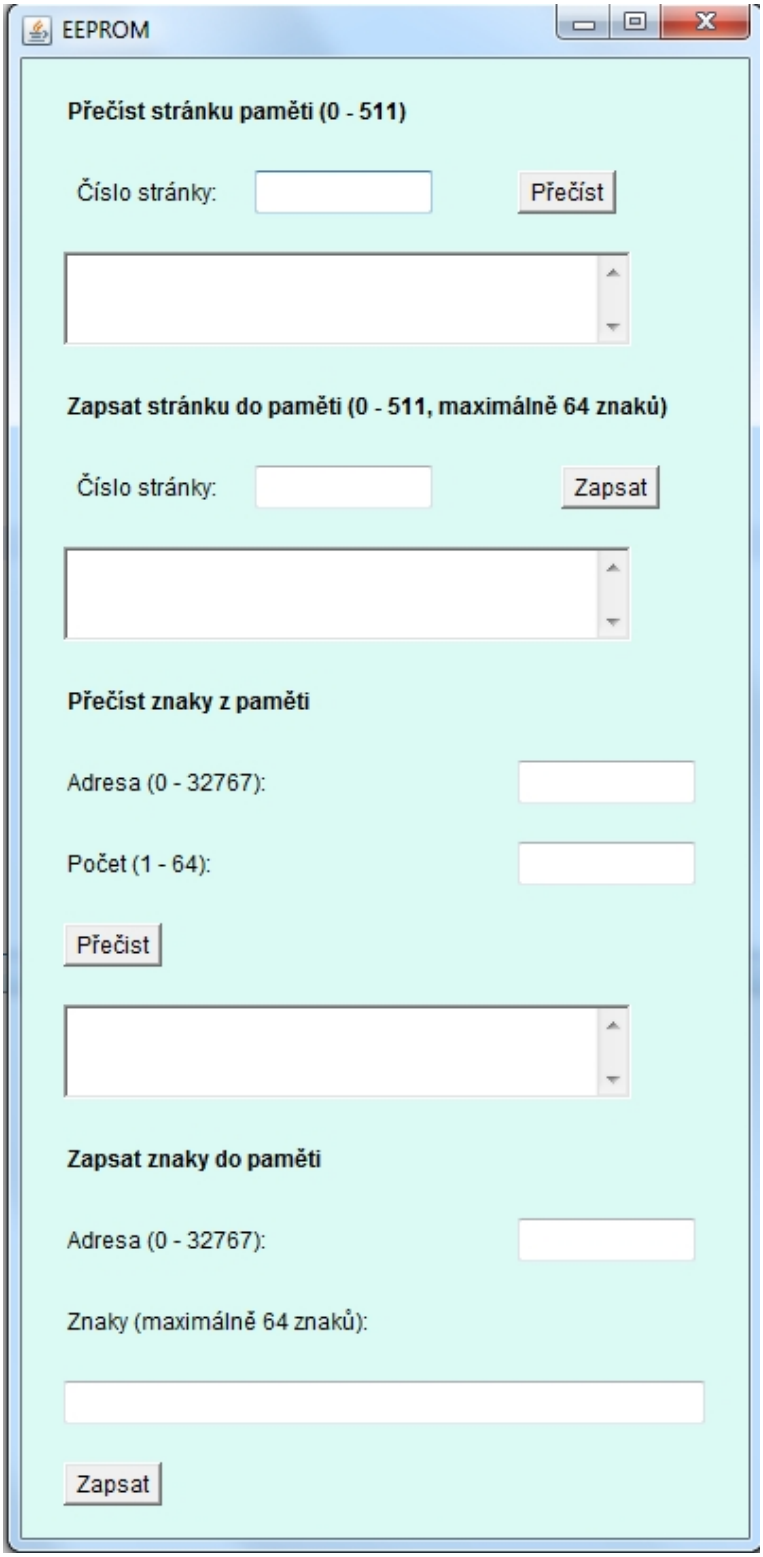
1. Po smazání displeje je kurzor nastaven na začátek prvního řádku.
2. V oknu pro práci s SD kartou je užitečný informační řádek(červený), často vám napoví, pokud se vám bude zdát, že se nic neděje.
3. Při kopírování souborů z/na SD kartu je potřeba nejprve vybrat cíl na SD kartě. Po stisknutí tlačítka *Zvolit cíl* se vygeneruje stromová struktura(obrázek C.5). Kliknutím na její položku jí vyberete jako cíl.
4. Při kopírování souborů z SD karty se tyto soubory ukládají do adresáře s projektem. Je tedy dobré mít projekt zkopírovaný na lokální disk.

Příloha C

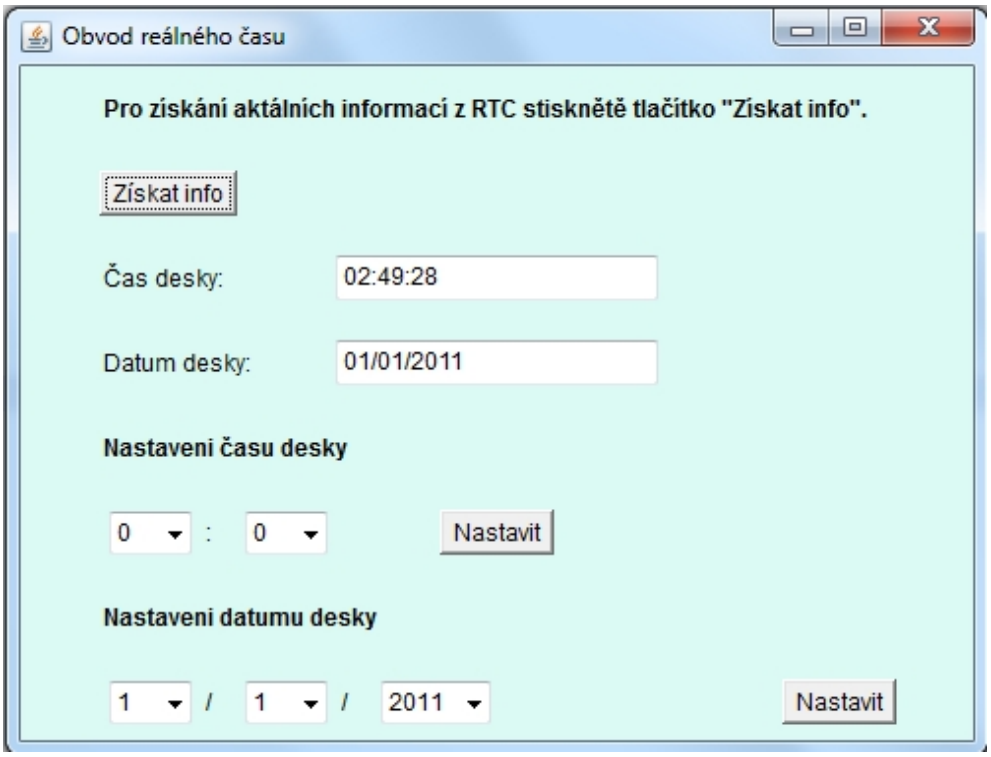
Screenshoty aplikace



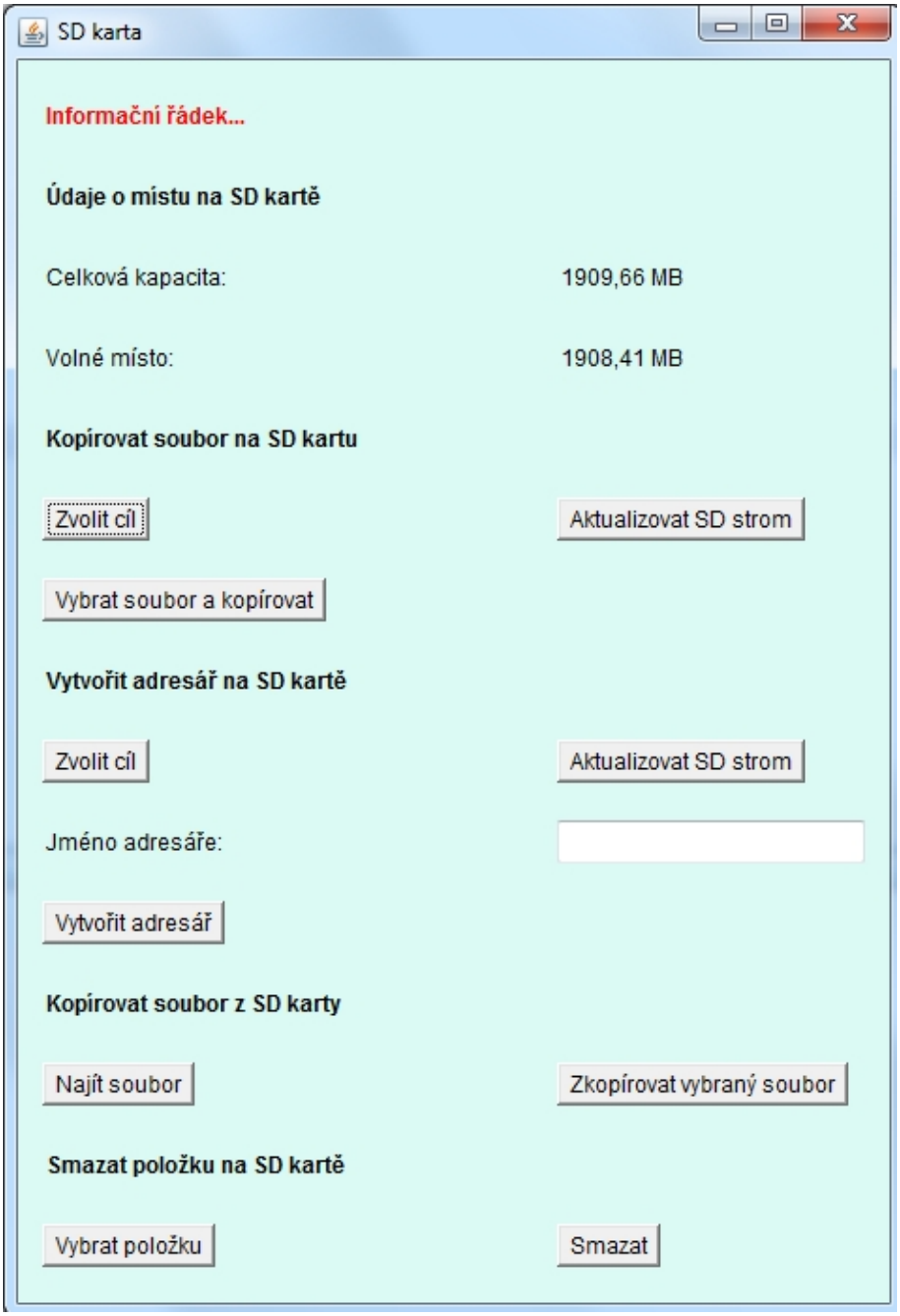
Obrázek C.1: Okno pro práci s LCD displejem



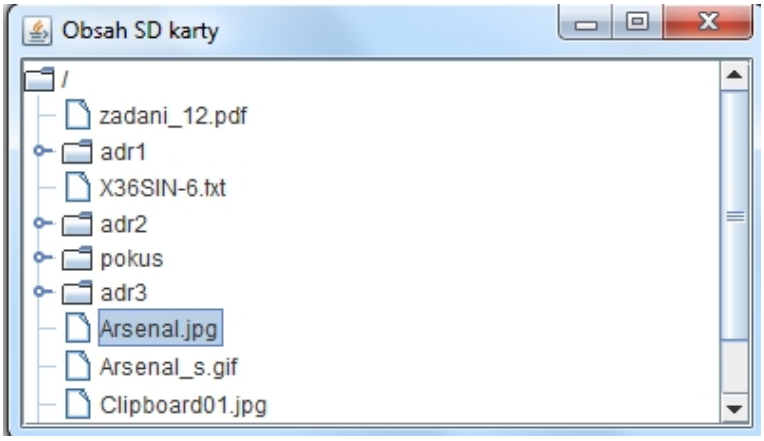
Obrázek C.2: Okno pro práci s EEPROM



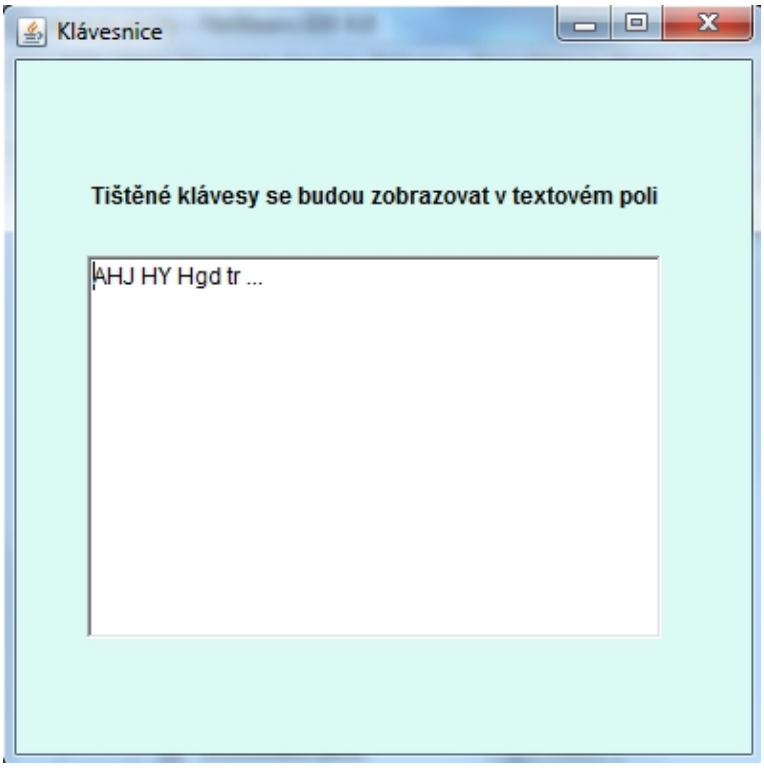
Obrázek C.3: Okno pro práci s RTC



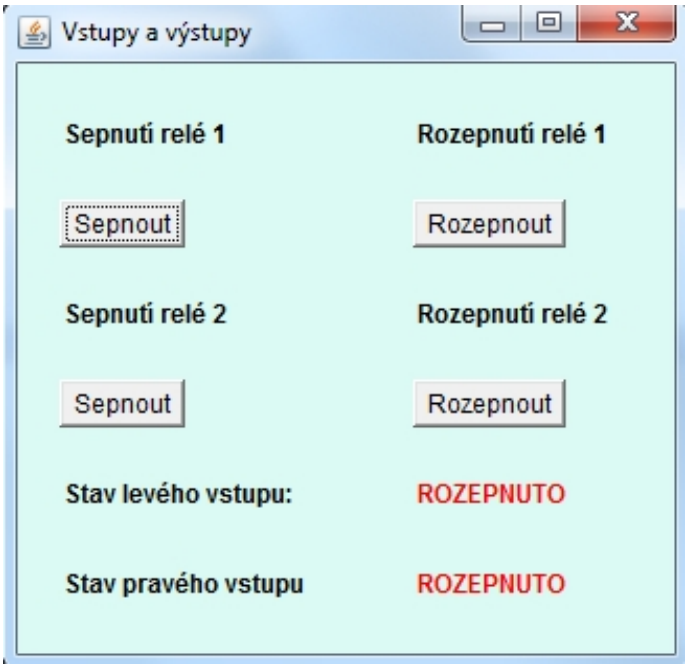
Obrázek C.4: Okno pro práci s SD kartou



Obrázek C.5: Stromová struktura SD karty



Obrázek C.6: Okno pro práci s klávesnicí



Obrázek C.7: Okno pro práci se vstupy a výstupy

Příloha D

Obsah přiloženého CD

Obrázek D.1: Obsah přiloženého CD