Sem vložte zadání Vaší práce.

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

Bakalářská práce

Univerzální řadič displejů

Vojtěch Miškovský

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

16. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce, panu Kubalíkovi, za pomoc s výběrem tématu a jeho realizací a za zapůjčení potřebného hardwaru. Dále bych chtěl poděkovat Tomáši Novákovi za pomoc s pájením konektorů a odhalením problému se sériovou linkou.

Prohlášení

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

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

V Praze dne 16. května 2013 .....

Odkaz na tuto práci

Miškovský, Vojtěch. *Univerzální řadič displejů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

The objective of this work is the creation of a platform for connecting character and graphical LCD displays to a serial port (RS232). The project should become a foundation of an universal LCD display driver providing a possibility of connecting every common monochromatic LCD display.

Keywords LCD display, character display, graphical display, KS0108, HD44780, RS232 display

Abstrakt

Cílem této práce je vytvoření platformy pro připojování znakových a grafických LCD displejů k sériové lince, resp. rozhraní RS232. Práce by měla sloužit jako základ pro zcela univerzální řadič displejů umožňující připojit každý běžně dostupný monochromatický displej.

Klíčová slova LCD displej, znakový displej, grafický displej, KS0108, HD44780, RS232 displej

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem této práce je vytvoření základu zařízení pro snadné ovládání znakových a grafických monochromatických LCD displejů. Výsledkem by mělo být zařízení, které bude pomocí rozhraní RS232 přijímat textové příkazy a na jejich základě zobrazovat informace na téměř libovolném připojeném displeji.

Jelikož displejů je na trhu velké množství, nemohu si z časových ani finančních důvodů dovolit implementovat opravdu všechny. Výstupem mé práce tedy bude modulární platforma vydaná pod otevřenou licencí, která umožní snadné přidávání dalších displejů. Vytvoření této práce tedy bude spíše odrazovým můstkem, než hotovým univerzálním řešením.

V rámci textu práce se budu zabývat v první kapitole zejména analýzou trhu s displeji a v kapitole druhé použitím vytvořeného zařízení, strukturou jeho firmware a možnostmi dalších úprav. Dále se budu věnovat grafické aplikaci, kterou pro účely snadného testování a používání zařízení vytvořím.

Kapitola 1

Analýza a návrh

1.1 Hardware

1.1.1 LCD displeje

LCD dnes patří k nejrozšířenějším zobrazovacím technologiím nejen ve světě informačních technologií, ale také ve spotřební elektronice.

Běžné konzumní LCD monitory se připojují prostřednictvím standardizovaných rozhraní (D-sub, DVI, HDMI, DiplayPort,...). Naproti tomu menší průmyslové LCD displeje takové standardy obvykle nemají a způsob jejich připojení je tudíž velmi rozmanitý. Údělem každého tvůrce zařízení, který chce takový displej využít, je, aby dopředu zvážil veškeré výhody a nevýhody a své zařízení připravil pro použití s konkrétním displejem.

LCD displeje se dělí na dvě hlavní skupiny: barevné a monochromatické (nejčastější český termín černobílé je zde značně zavádějící, jelikož tato barevná kombinace se v LCD nevyskytuje). Barevné displeje se využívají zejména ve složitějších zařízeních, jako jsou výše zmiňované monitory, mobilní telefony apod. Mé zařízení však bude mířit do jednodušších vestavných zařízení, v nichž se nejčastěji využívá displejů monochromatických, jejichž připojování je řádově jednodušší.

Monochromatické displeje se v naprosté většině případů dodávají na jednom tištěném spoji společně s řadičem. Řadič nám umožňuje relativně jednoduché ovládání displeje a nabízí nám základní operace s většími bloky pixelů (např. přímé zobrazování znaků). Tomuto řadiči sekunduje ještě paměť pro uložení zobrazovaných dat. Ovládání pak probíhá prostřednictvím nastavování řídících signálů a čtení signálů stavových. Bohužel však jednotlivé řadiče nejsou vzájemně kompatibilní, což je problém, který se se svým zařízením budu snažit v co největší míře vyřešit.

Hitachi HD44780	
1	napájení GND
2	napájení +5V
3	nastavení kontrastu
4	RS - volba instrukce/data
5	RW - volba čtení/zápis
6	Enable signál
7	data bit DB0
8	data bit DB1
9	data bit DB2
10	data bit DB3
11	data bit DB4
12	data bit DB5
13	data bit DB6
14	data bit DB7
15	anoda podsvětlení
16	katoda podsvětlení

Tabulka 1.1: Hitachi HD44780 pinout [2]

Displeje můžeme rozdělit do dvou hlavních kategorií. Jsou to displeje znakové a displeje grafické. Za další skupinu mohou být považovány displeje kombinované, které kromě grafických operací podporují i snadné psaní znaků, blíže však mají k displejům grafickým.

Znakové displeje jsou takové displeje, které jsou určeny ke zobrazování pouze textových informací. Jejich pixely jsou seskupovány do skupin (nejčastěji 5x8 či 7x8), kde každá tato skupina slouží ke zobrazení právě jednoho znaku.

Naprostá většina znakových displejů je párována s řadičem společnosti Hitachi s kódovým označením HD44780, který využívá pro komunikaci 16bitové paralelní rozhraní. Ostatními řadiči znakových displejů se zabývat nebudu.

U displejů grafických je situace značně komplikovanější. Zde můžeme řadiče rozdělit podle způsobu komunikace do dvou skupin - sériové a paralelní.

Výhoda sériově připojovaných displejů spočívá v nízkém počtu potřebných vodičů. Nicméně hlavní nevýhodou je relativně složitá implementace časování a pro mé účely také téměř nemožné sjednocení rozhraní se znakovými displeji (řadičem HD44780). Naštěstí sériově připojované displeje nejsou v praxi příliš využívané, jejich hlavní doménou byly starší mobilní

Samsung KS0108	
1	napájení GND
2	napájení +5V
3	nastavení kontrastu
4	RS - volba instrukce/data
5	RW - volba čtení/zápis
6	Enable signál
7	data bit DB0
8	data bit DB1
9	data bit DB2
10	data bit DB3
11	data bit DB4
12	data bit DB5
13	data bit DB6
14	data bit DB7
15	výběr řadiče pro levou část LCD
16	výběr řadiče pro pravou část LCD
17	reset modulu
18	výstup záporného napětí pro LCD (asi -12V)
19	anoda podsvětlení
20	katoda podsvětlení

Tabulka 1.2: Samsung KS0108 pinout [6]

telefony. Z těchto důvodů se jimi ve své práci nebudu zabývat.

Paralelně připojované displeje jsou využívané výrazně hojněji, nicméně i zde nalezneme několik problémů. Narozdíl od sériově připojovaných využívají pro komunikaci vyšší počet pinů, což ovšem koresponduje se znakovými displeji, takže pro mé zařízení se vlastně o problém nejedná. Hlavním problémem je, narozdíl od znakových displejů, velká množina používaných řadičů, jejichž způsob zapojení se často velmi liší. Mezi nejčastěji používané řadiče patří Samsung KS0108, Toshiba T6963 a další.

V tabulkách 1.1, 1.2 a 1.3 můžete najít pinové zapojení nejčastěji používaných řadičů. Jak je vidět, jsou zde relativně malé rozdíly, z nichž některé půjdou implementovat softwarově a některé hardwarově (pro některé displeje zkrátka bude třeba použít jednoduchou redukci pro změnu pořadí pinů).

Toshiba T6963	
1	napájení GND
2	napájení +5V
3	nastavení kontrastu
4	RS - volba instrukce/data
5	RW - volba čtení/zápis
6	Enable signál
7	data bit DB0
8	data bit DB1
9	data bit DB2
10	data bit DB3
11	data bit DB4
12	data bit DB5
13	data bit DB6
14	data bit DB7
15	aktivace modulu
16	reset modulu
17	výstup záporného napětí pro LCD (asi -12V)
18	ovládání podsvětlení
19	katoda podsvětlení
20	anoda podsvětlení

Tabulka 1.3: Toshiba T6963 pinout [7]

1.1.2 Dostupná řešení

Hledáním na internetu jsem objevil několik projektů, které jsou částečně podobné mému cíli. Žádný z nich však neumí připojovat displeje s různými řadiči.

- `http://obsoletetechnology.wordpress.com/projects/rs232-lcd-display-driver/`

  Tento modul umožňuje jednoduché ovládání textových displejů pomocí RS232. Podporuje řadič HD44780 a ovládá se pomocí několika jednobytových příkazů.

- `http://electronics-diy.com/electronic_schematic.php?id=759`

  Modul velice podobný předchozímu.

- `http://kitsrus.com/pdf/K192.pdf`

Opět modul určený pro HD44780, tentokrát ovšem v sofistikovanějším provedení a velmi pěkně vypracovanou dokumentací. Podporuje například komunikaci na úrovních TTL a uživatelsky definované znaky. Ovládání probíhá opět pomocí jednobytových příkazů.

- `http://www.ozitronics.com/docs/sLCD.pdf`

  Další z řady modulů pro řadič HD44780. Tento navíc kromě RS232 podporuje také rozhraní MT (podrobnosti o tomto rozhraní jsem nezjišťoval). Co se mi u tohoto modulu líbilo a také jsem to ve své práci použil, je používání textových příkazů.

- `https://www.sparkfun.com/products/9352`

  Modul pro použití se dvěma různými grafickými displeji - jedním s řadičem KS0108 a druhým s řadičem 6963C. Komunikace probíhá pomocí jednobytových příkazů.

- `http://users.fit.cvut.cz/~xkubalik/lib/exe/fetch.php?media=project:2009:2009_hajrapetjan_armen.pdf`

  Modul pro řadič KS0108, jednobytové příkazy.

Jak je vidět z předchozího seznamu, pro některé z nejběžnějších řadičů existují zařízení, která umožní jejich připojení přímo na sériovou linku. Tato zařízení se však vždy dají použít pouze s jedním konkrétním řadičem (resp. v jednom případě dvěma). Žádné univerzální řešení se mi nepodařilo objevit.

Komunikační protokoly u těchto zařízení jsou obvykle navrhovány s ohledem na zaměření na znakové, nebo grafické displeje, kde u znakových displejů bývá komunikace jednodušší. Povětšinou pro jednotlivé příkazy využívají netisknutelné znaky. Tento způsob komunikace se mi příliš nelíbí, je nevhodný například pro nastavení a ovládání displeje prostřednictvím terminálu.

1.1.3 Použité prostředky

Pro realizaci mého řadiče je potřeba zvolit vhodné hardwarové prostředky.

Základní otázkou bylo, zda použít již hotové řešení, nebo navrhnout své vlastní. Jelikož na trhu existuje velké množství nejrůznějších periferních desek s rozmanitými mikrokontroléry a s ohledem na mou nezkušenost v oblasti návrhu a realizace tištěných spojů jsem se rozhodl použít řešení hotové.

Požadavky na vývojovou desku byly následující:

- alespoň 8bitový mikrokontrolér
- velké množství pinů pro připojení displeje
- napájení +5V
- hardwarově ralizované rozhraní UART
- cenová dostupnost

Těmto požadavkům by vyhovělo značné množství komerčních desek, nakonec jsem však zvolil desku navrženou bývalým studentem FEL [1].

Tato deska vznikla jako bakalářská práce pod vedením pana Kubalíka, čímž pro mě byla dobře dostupným řešením. Jelikož původním účelem byla komunikace s displejem s řadičem KS0108 prostřednictvím RS232, byla pro mě vhodným kandidátem. Navíc jejím srdcem je 8bitový mikrokontrolér PIC 18f4520 od společnosti Microchip, s jejímiž čipy mám zkušenosti.

Pro programování mikrokontroléru jsem využil programátor PICkit3 přímo od výrobce čipu. Pro seriovou komunikaci jsem použil převodník USB-RS232. Oboje mi ochotně zapůjčil pan Kubalík.

1.2 Software

1.2.1 Programovací prostředí

Pro vývoj firmwaru a jeho nahrávání jsem využil vývojové prostředí MPLAB IDE v8.83 společně s balíkem překládacích nástrojů MPLAB C18, oboje od společnosti Microchip. S tímto prostředím jsem měl předchozí zkušenosti. Navíc pochází přímo od výrobce mikrokontroléru a je dodáváno společně s množstvím užitečných knihoven [3]. Programovacím jazykem byl jazyk C.

1.2.2 Ovládací aplikace

Grafickou ovládací aplikaci jsem programoval v jazyce C#. Jeho nevýhoda sice spočívá v nutnosti instalace runtime prostředí .NET framework (resp. Mono), ovšem výhodou je multiplatformnost a snadná práce se sériovou linkou. Jako vývojové prostředí jsem zvolil Visual Studio 2010.

Důvodem, proč jsem nezvolil podobný, konkureční jazyk Java, byl fakt, že s ním nemám tolik zkušeností.

Kapitola 2

Realizace

2.1 Použití

Zařízení je ovládáno pomocí příkazů přijímaných protokolem RS232 a to buď z počítače připojením pomocí přímého kabelu k sérové lince, nebo propojením s jiným nízkonapěťovým zařízením s možností komunikace na TTL úrovních.

Komunikační rozhraní je nastaveno na baudrate 9600, 8 datových bitů, 1 startbit, 1 stopbit, bez parity.

Kromě datového kabelu je třeba k zařízení připojit napájení (+5V) a požadovaný displej.

Dále je třeba dát pozor na nastavení kontrastu, který se reguluje pomocí trimmeru na horní straně zařízení. Potýkal jsem se s problémem, že mi displej s řadičem HD44780 nic nezobrazoval a nakonec jsem zjistil, že jeho viditelná úroveň kontrastu je úplně jiná něž u displeje s řadičem KS0108.

Ovládací příkazy jsou textové ve formátu *kod-příkazu(parametry oddělené čárkou)* a zapisují se bezprostředně za sebe. Případné oddělení netisknutelnými znaky je zařízením ignorováno.

Úspěšné zpracování příkazu zařízením je signalizováno odesláním znaku *'D'*. Před přijetím tohoto znaku by neměly být do zařízení zasílány žádné další příkazy.

Zařízení pro jednodušší ovládání pomocí terminálu podporuje přepnutí do *echo* režimu, kdy všechny přijaté příkazy jsou zaslány zpět uživateli. Uživatel tak vidí, co poslal, a může odhalit případný překlep, který zabránil úspěšnému zpracování příkazu. Defaultně je *echo* režim vypnutý.

2.1.1 Textové příkazy

V následujícím seznamu jsou popsány všechny v současnosti implementované příkazy a jejich parametry. Jiné příkazy popř. příkazy s chybnými parametry budou ignorovány.

() Zapne/vypne režim *echo*.

disp(n,x,y) Slouží k vybrání připojeného displeje.

- **n** kód připojeného displeje
  - **1** řadič Samsung KS0108
  - **2** řadič Hitachi HD44780
- **x** šířka displeje (pro grafické v pixelech, pro znakové ve znacích)
- **y** výška displeje (pro grafické v pixelech, pro znakové ve znacích)

clr(barva) Vymaže obsah displeje.

- **barva** určuje, zda vymazání proběhne zapnutím, nebo vypnutím všech pixelů
  - **0** displej bude vyplněn vypnutými pixely
  - **1** displej bude vyplněn zapnutými pixely

curs(x,y) Posune znakový kurzor na zadanou pozici.

- **x** určuje pozici následujícího zadaného znaku na řádku
- **y** určuje pořadí řádku, na který bude následující znak zapsán

str(text,barva) Vypíše na displej text od aktuálního umístění kurzoru.

- **text** text k zapsání, omezen maximálně na 63 znaků

  Pro zapsání znaků se speciálním významem {,()\} musí těmto znakům předcházet znak '\'. Pro přesunutí kurzoru na nový řádek slouží dvojice znaků '\n'. V případě, že za znakem '\' bude následovat jiný znak, znak '\' bude ignorován.
- **barva** určuje, zda jednotlivé znaky budou zapsány zapnutím, nebo vypnutím pixelů
  - **0** text bude zapsán vypnutými pixely
  - **1** text bude zapsán zapnutými pixely

pt(x,y,barva) Vykreslí na displej bod.

x vodorovná souřadnice bodu

y svislá souřadnice bodu

barva určuje, zda bod bude vykreslen zapnutím, nebo vypnutím pixelu

0 bod bude vykreslen vypnutým pixelem

1 bod bude vykreslen zapnutým pixelem

line(x1,y1,x2,y2,barva) Vykreslí na displej úsečku od bodu 1 do bodu 2.

x1 vodorovná souřadnice prvního bodu úsečky

y1 svislá souřadnice prvního bodu úsečky

x2 vodorovná souřadnice posledního bodu úsečky

y2 svislá souřadnice posledního bodu úsečky

barva určuje, zda jednotlivé body úsečky budou vykresleny zapnutím, nebo vypnutím pixelů

0 úsečka bude vykreslena vypnutými pixely

1 úsečka bude vykreslena zapnutými pixely

rect(x1,y1,x2,y2,barva) Vykreslí na displej obdélník.

x1 vodorovná souřadnice první svislé strany obdélníku

y1 svislá souřadnice první vodorovné strany obdélníku

x2 vodorovná souřadnice druhé svislé strany obdélníku

y2 svislá souřadnice druhé vodorovné strany obdélníku

barva určuje, zda jednotlivé body obdélníku budou vykresleny zapnutím, nebo vypnutím pixelů

0 obdélník bude vykreslen vypnutými pixely

1 obdélník bude vykreslen zapnutými pixely

circ(x,y,r,barva) Vykreslí na displej kružnici.

x vodorovná souřadnice středu kružnice

y svislá souřadnice středu kružnice

r poloměr kružnice v pixelech

barva určuje, zda jednotlivé body kružnice budou vykresleny zapnutím, nebo vypnutím pixelů

0 kružnice bude vykreslena vypnutými pixely

1 kružnice bude vykreslena zapnutými pixely

2.1.2 Grafická aplikace

Návod na použití grafické aplikace najdete v sekci 2.3.1

2.2 Firmware

Jak již bylo zmíněno dříve, pro realizaci firmwaru jsem zvolil vývojové prostředí od výrobce mikrokontroléru, MBLAB IDE, a programovací jazyk C.

2.2.1 Struktura

Během realizace jsem usiloval o to, aby výsledný kód byl co nejsnadněji rozšiřitelný. Primárně mi pak šlo o jednoduchost při přidávání podpory pro další displeje, aby se maximalizovala univerzálnost tohoto řešení.

Celý projekt je rozdělený do několika knihoven, které spolu úzce spolupracují, ale zároveň jsou snadno nahraditelné pro úpravu na míru individuálním potřebám užvatele. Tyto knihovny si nyní popíšeme. Pro lepší představu o závislostech knihoven slouží diagram na obrázku 2.1.

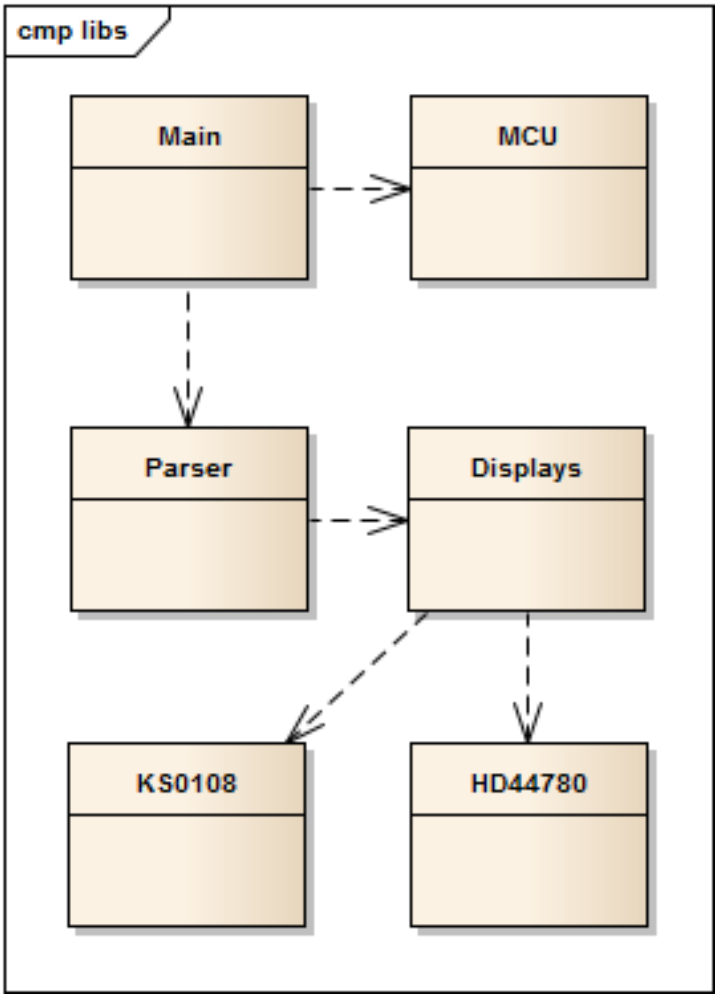
2.2.1.1 MCU

mcu.h, mcu.c

Tato knihovna obsahuje veškerý kód, který je závislý na použitém mikrokontroléru. Pro svůj chod potřebuje základní knihovnu pro práci s mikrokontrolérem, v mém případě *p18f4520.h*, a knihovnu pro časování, *delays.h*, které jsou dodávány s vývojovým prostředím MPLAB. Knihovna *MCU* bsahuje:

- definice přiřazení jednotlivých pinů pro displej odpovídajícím portům mikrokontroléru
- definice pro nastavení portů jako vstup/výstup
- funkci pro počáteční inicializaci mikrokontroléru
- funkce pro ovládání rozhraní UART
- funkce pro čekání

Díky oddělení všech těchto součástí od zbytku kódu stačí pro použití mé práce na jiném mikrokontroléru pouze nahradit tuto knihovnu.



Obrázek 2.1: Diagram závislostí mezi knihovnami

2.2.1.2 Displays

displays.h, displays.c

Displays je zastřešovací knihovna pro knihovny jednotlivých displejů. Právě na přítomnosti těchto knihoven je také závislá.

Obsahuje definice ukazatelů na funkce sloužící k vykreslování na displej. Podle zvoleného displeje se k ukazatelům přiřadí odpovídající funkce z knihovny daného řadiče.

Veškeré úpravy potřebné pro přidání podpory pro další displej jsou prováděny právě v této knihovně. Blíže se těmto úpravám věnuji v sekci 2.2.2.

2.2.1.3 Knihovny displejů

ks0108.h, ks0108.c, hd44780.h, hd44780.c

Tyto knihovny jsou základním stavebním kamenem celého řešení. Knihovna *displays.c* je využívá pro vykreslování dat na displejích. V současnosti jsou součástím aplikace knihovny pro displeje Samsung KS0108 (grafický) a Hitachi HD44780 (znakový).

Knihovny v aktuální verzi nabízejí tyto funkce pro práci s displejem:

disp_ON aktivace displeje

disp_CLR vymazání obsahu displeje

disp_Set_Cursor nastavení pozice kurzoru

disp_Put_String vypsání řetězce na displej

disp_Draw_Point vykreslení pixelu

disp_Draw_Line vykreslení úsečky

disp_Draw_Rect vykreslení obdélníku

disp_Draw_Circle vykreslení kružnice

V jednotlivých knihovnách je *disp* nahrazeno kódovým označení displeje (např. *ks0108_Draw_Line* a *hd44780_Draw_Line*).

Možnosti přidání dalších funkcí pro vykreslení na displej se podrobněji věnuji v sekci 2.2.2.

Co se týče rozdílů mezi znakovými a grafickými displeji, rozhodl jsem se je v návrhu nerozlišovat. Reálný rozdíl bude tedy v tom, že příkazy určené pouze pro grafické displeje budou implementovány prázdnou funkcí. Důvodem pro toto rozhodnutí je zejména to, že v procedurálním jazyce C nemohu využít dědičnost, která by se v tomto případě velmi hodila, a jiné řešení by způsobilo komplikace při přidávání nových displejů.

Jelikož mám k dispozici pouze jeden displej s řadičem KS0108 a jeden s řadičem HD44780, nemohu zaručit funkčnost na všech displejích s těmito řadiči. Konkrétně je funkčnost ověřena pro displej s 20x4 znaky s řadičem HD44780 a displej s 128x64 pixely s řadičem KS0108. Nicméně příkaz *disp* zpracovává rozměry displeje, takže případná optimalizace pro displeje s jinými rozměry je otázkou drobných úprav ve vykreslovacích funkcích v knihovnách řadičů.

2.2.1.4 Parser

parser.h, parser.c

Knihovna Parser zpracovává textové příkazy a jejich parametry. Poté volá odpovídající prováděcí funkce. Knihovna je závislá na *displays.c* a lze ji poměrně snadno nahradit v případě, že uživatel by chtěl např. využívat jiný způsob komunikace.

Bližší popis textového protokolu, který tato knihovna zpracovává naleznete v sekci 2.1.1.

2.2.1.5 Hlavní program

main.c

Na začátku souboru se nachází konfigurační příkazy mikrokontroléru.

Hlavní program vyvolá na svém počátku funkce pro inicializaci mikrokontroléru a jeho UART rozhraní. Následuje nekonečná smyčka, která přijímá příkazy z UART a poté je předává parseru.

Závisí na knihovnách *parser.c* a *mcu.c*. Kvůli konfiguračním příkazům je třeba zde také zahrnout hlavičku *p18f4520.h*.

2.2.2 Úpravy

V této sekci se budu věnovat možnostem rozšíření resp. přizpůsobení aplikace. Právě na možnost budoucího rozšíření, zejména o podporu dalších displejů, jsem se v návrhu zaměřil, aby se můj řadič mohl v budoucnu stát maximálně univerzálním.

2.2.2.1 Přidání podpory pro další displej

Základem pro přidání podpory pro další displej je vytvoření knihovny pro práci s ním. Ta musí implementovat všechny funkce uvedené v sekci 2.2.1.3 pojmenované v zavedeném formátu.

Budeme-li chtít přidat displej z názvem *newdisplay*, bude hlavičkový soubor jeho knihovny vypadat následovně:

```
/* newdisplay.h */

#ifndef _NEWDISPLAY
#define _NEWDISPLAY

void newdisplay_ON(void);
void newdisplay_CLR(char color);
void newdisplay_Draw_Point(char x,char y, char color);
```

```
void newdisplay_Draw_Rect(char x1,char y1,char x2,char y2, char
    color);
void newdisplay_Draw_Line(char x1,char y1,char x2,char y2, char
    color);
void newdisplay_Draw_Circle(char x0, char y0, char r, char
   color);
void newdisplay_Put_String(char * string, char color);
void newdisplay_Set_Cursor(char x, char y);

#endif
```

Dalším krokem je přidat tento displej do knihovny *displays.c*. Konkrétně je třeba přidat link na hlavičkový soubor *newdisplay.h*, přidat definici číselného identifikátoru, podle kterého se bude displej volit příkazem *disp*, a přidat pro tento identifikátor *case* do konstrukce *switch* ve funkci *set_display*, který vyvolá makro *SET*. Toto makro slouží pro přiřazení funkcí daného displeje k ukazatelům, pomocí kterých jsou volány.

Po zmíněných úpravách může soubor *displays.c* vypadat například takto (znakem % jsou označeny přidané řádky kódu):

```
/* displays.c */

#include "ks0108.h"
#include "hd44780.h"
#include "newdisplay.h"        %
#include "displays.h"

#define NONE 0
#define DISP_KS0108 1
#define DISP_HD44780 2
#define DISP_NEWDISPLAY 3      %

#define SET(TYPE) disp_Draw_Line = TYPE ## _Draw_Line;
    disp_Draw_Circle = TYPE ## _Draw_Circle; disp_Draw_Rect =
    TYPE ## _Draw_Rect; disp_Put_String = TYPE ## _Put_String;
    disp_CLR = TYPE ## _CLR; disp_Set_Cursor = TYPE ##
    _Set_Cursor; disp_ON = TYPE ## _ON

extern void (*disp_Draw_Line)(char x1,char y1,char x2,char y2,
   char color);
extern void (*disp_Draw_Rect)(char x1,char y1,char x2,char y2,
   char color);
extern void (*disp_Draw_Circle)(char x,char y,char r, char
   color);
extern void (*disp_Put_String)(char * string, char color);
extern void (*disp_CLR)(char color);
extern void (*disp_ON)(void);
extern void (*disp_Set_Cursor)(char x,char y);
```

```
void set_display(char n) {
  switch (n) {
    case DISP_KS0108:
      SET(ks0108);
      break;
    case DISP_HD44780:
      SET(hd44780);
      break;
    case DISP_NEWDISPLAY:      %
      SET(newdisplay);         %
      break;                   %
    default:
      return;
  }
  disp_ON();
  disp_CLR(0);
  disp_selected = n;
}
```

Tímto jsme úspěšně přidali displej *newdisplay* mezi podporované displeje.

Na co je však třeba také dbát, je hardwarová kompatibilita. Některé řadiče displejů (viz. například 1.3) mají nekompatibilní rozložení pinů. Takovou situaci je potřeba řešit jednoduchou redukcí.

Pro připojení zmíněného displeje s řadičem Toshiba T6963 bychom například museli vytvořit redukci s následujícím zapojení pinů:

Zařízení	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Displej	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	18	16	20	19

Zapojení displejů s řadiči KS0108 a HD44780 jsou si velmi podobná (u HD44780 jsou některé piny nevyužité) až na zapojení elektrod podsvícení. Jelikož na zařízení, pro které je má práce určena, je podsvícení špatně zapojeno (přímo na pinu mikrokontroléru, který nedodává dostatečný proud, tudíž podsvícení nefunguje), elektrody jsem pro zapojení řadiče HD44780 ignoroval a tudíž jsem žádnou redukci vytvářet nemusel.

2.2.2.2 Přidání nového příkazu a nové funkce displeje

Přidání nového příkazu je oproti přidání displeje o něco složitější. Je potřeba nejen naprogramovat tuto funkčnost pro daný displej, ale také přidat do parseru podporu pro nový příkaz.

Pro přidání příkazu musíme provést následující kroky:

1. Do všech knihoven displejů přidáme funkci, která bude tento nový příkaz provádět. Neměli bychom zapomenout na přidání funkce také do hlavičkových souborů.

2. Přidáme ukazatel na tuto funkci do hlavičkového souboru *displays.h*.

3. Do makra *SET* v knihovně *displays.c* přidáme přiřazení dané funkce do ukazatele. Například *disp_Nazev_Funkce = TYPE ## _Nazev_Funkce;*

4. Do knihovny *parser.c* přidáme deklaraci ukazatele na tuto novou funkci.

5. V knihovně *parser.c* dále přidáme definici číselného identifikátoru funkce. Tento identifikátor by měl být o jedna vyšší, než identifikátor doposud posledního příkazu. Také musíme zvýšit hodnotu *CMDS_N*.

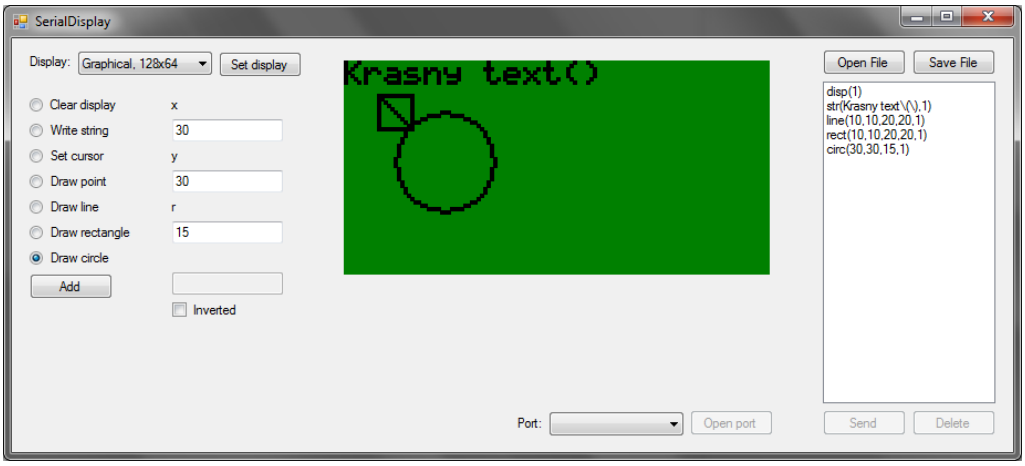
6. V téže knihovně přidáme do pole *cmds* text pro vyvolání příkazu a do pole *params* parametry přijímané příkazem. Prvky v těchto polích musí pořadím odpovídat identifikátoru definovanému pro tento příkaz (viz. bod 5).

7. V knihovně *parser.c* přidáme do konstrukce *switch* obstarávající volání funkcí displeje *case* odpovídající identifikátoru příkazu (viz. bod 5).

Soubor *parser.c* by měl nyní vypadat asi takto (nově přidané, nebo upravené řádky jsou označeny znakem %, soubor je pro úsporu místa zkrácen o nepodstatné části):

```
/* parser.c */

   ...
#define CURS 6
#define DISP 7
#define POINT
#define NEW 9                                          %

#define CMDS_N 10                                       %

#define NONE 0
#define NUMBER 1
#define STRING 2
#define BOOL 3
   ...
void (*disp_ON)(void);
void (*disp_Set_Cursor)(char x,char y);
```

```
void (*disp_Nova_Funkce)(char n,char color);    %

const char cmds[][5] = {
  "",
  "rect",
  "circ",
  "str",
  "line",
  "clr",
  "curs",
  "disp",
  "pt",
  "new"                                          %
};

const char params[][5] = {
  "",
  {NUMBER,NUMBER,NUMBER,NUMBER,BOOL},
  {NUMBER,NUMBER,NUMBER,BOOL,NONE},
  {STRING,BOOL,NONE,NONE,NONE},
  {NUMBER,NUMBER,NUMBER,NUMBER,BOOL},
  {BOOL,NONE,NONE,NONE,NONE},
  {NUMBER,NUMBER,NONE,NONE,NONE},
  {NUMBER,NONE,NONE,NONE,NONE},
  {NUMBER,NUMBER,BOOL,NONE,NONE},
  {NUMBER,BOOL,NONE,NONE,NONE}                   %
};

  ...
char parseParams(char * string, char cmd) {
  ...
  switch (cmd) {
    case RECT:
      disp_Draw_Rect(nparam1,nparam2,nparam3,nparam4,nparam5);
      break;
      ...
    case DISP:
      set_display(nparam1);
      break;
    case POINT:
      disp_Draw_Point(nparam1,nparam2,nparam3);
      break;
    case NEW:                                    %
      disp_Nova_Funkce(nparam1,nparam2);         %
      break;                                     %
  }
  return 0;
}
```



Obrázek 2.2: Náhled grafické aplikace

V tuto chvíli bychom měli mít program ve stavu, kdy po přeložení a nahrání do zařízení bude schopen zpracovat nově přidaný příkaz. Ten bude zadáván ve formátu *new(cislo,bool)* a bude volat funkci displeje *disp_Nova_Funkce*.

Pozor! Parametr typu *STRING* lze použít pouze pro první parametr příkazu. Pokud jej použijeme, je třeba předávat v konstrukci *switch* volané funkci proměnnou *param1*, nikoli *nparam1*.

2.3 Grafická aplikace

Pro ještě snadnější ovládání displejů a jako demonstraci funkčnosti svého řešení jsem vytvořil jednoduchou grafickou aplikaci.

Tato aplikace komunikuje se zařízením prostřednictvím textového protokolu, jemuž se blíže venuji v sekci 2.1.1.

Kromě snadného zadávání jednotlivých příkazů umožňuje také načtení zdrojového souboru, který má syntaxi shodnou s komunikačním protokolem. Této funkce se dá využít pro dávkové zasílání příkazů do displeje. Aplikace navíc obsahuje náhled výsledné podoby displeje, díky kterému lze příkazy odladit i bez připojení displeje.

Grafická aplikace je psána v jazyce C#, její chod zajišťuje rozhraní .NET framework. Funkčnost aplikace jsem ověřoval na Windows 7 s nainstalovaným .NET frameworkem verze 4, nicméně je více než pravděpodobné, že bude bez problémů fungovat i se staršími verzemi či v prostředí Mono.

Náhled grafické aplikace můžete vidět na obrázku 2.2

2.3.1 Použití

Ačkoli je dle mého názoru rozhraní aplikace dostatečně přehledné a intuitivní, uvedu zde stručný návod pro její použití.

2.3.1.1 Zadávání příkazů

Ovládací prvky pro zadávání příkazů nalezneme v levé části aplikace.

Před zadáváním příkazů je třeba zvolit displej, který budeme k zařízení připojovat. To provedeme vybráním zvoleného displeje z nabídky a potvrzením tlačítkem *Set display*. Samotné zadávání příkazů probíhá v těchto krocích:

1. zvolení příslušného příkazu
2. zadání všech potřebných parametrů (jejich označení se mění v závislosti na zvoleném příkazu)
3. v případě grafických displejů volitelné použití zatržítka *Inverted* pro vykreslení zhasnutím pixelů
4. potvrzení výběru tlačítkem *Add*.

Všechny takto vytvořené příkazy se zobrazují v seznamu příkazů v pravé části okna. Zde je můžeme vybrat a kliknutím na tlačítko *Delete* smazat. Příkazy umístěné v seznamu se nám také automaticky zobrazují v náhledu v prostřední části okna.

2.3.1.2 Práce se soubory

Pokud vytvoříme seznam příkazů, který bychom rádi využívali opakovaně, můžeme si pomoci uložením příkazů do souboru. K tomu slouží tlačítko *Save file*. Po kliknutí na toto tlačítko se nám otevře nabídka pro vybraní cesty k souboru. Po zvolení cesty a názvu souboru se veškeré příkazy v seznamu do tohoto souboru pro případné další načtení.

Pokud máme uložený seznam příkazů, nebo jsme si soubor s příkazy ručně vytvořili, můžeme jej otevřít kliknutím na tlačítko *Open file*. Zobrazí se nám dialog pro vybrání souboru. Po vybrání se příkazy v souboru uložené zobrazí v seznamu příkazů v pravé části okna. Navíc se v prostřední části okna vytvoří náhled výsledného zobrazení na displej.

2.3.1.3 Odeslání sériovou linkou

Až budeme se zobrazenými daty spokojeni, stačí již pouze zvolit z nabídky *COM* port, ke kterému máme připojeno zařízení, potvrdit volbu kliknutím na tlačítko *Open port* a tlačítkem *Send* odeslat všechny příkazy do zařízení resp. na displej.

2.3.2 Struktura

Zdrojový kód grafické aplikace je rozdělen do několika hlavních tříd, kdy každá taková třída se nachází ve vlastním zdrojovém souboru. Aplikace obsahuje následující hlavní třídy:

2.3.2.1 Display

Display.cs

Tato třída obstarává vykreslování náhledu displeje. Funguje podobně jako knihovna *displays.c* v kódu firmwaru (podrobnosti najdete v sekci 2.2.1.2).

Třída vykresluje dva typy displejů, buď grafický, nebo textový. Jelikož je nezávislá na řadiči displeje, nemá žádné podtřídy pro jejich implementaci.

2.3.2.2 Parser

Parser.cs

Tato třída je v podstatě identická s knihovnou *parser.c*, pouze je přepsána do jazyka C#. Podrobnosti o knihovně *parser.c* najdete v sekci 2.2.1.4.

2.3.2.3 Commander

Commander.cs

Tato třída se stará o přiřazení radio button prvků k jednotlivým příkazům. Dále se stará o správné zpřístupnění a pojmenování textových polí jednotlivých parametrů.

2.3.2.4 Form1

Form1.cs, Form1.Designer.cs

Tato třída je zodpodědná za vykreslování okna aplikace a jednotlivých prvků (vyjma displeje). Také se v ní nachází ovládání seriové linky.

2.3.3 Úpravy

V této sekci se budu věnovat základním možnostem úpravy grafické aplikace, díky kterým se může vylepšovat současně s vylepšováním firmwaru samotného zařízení.

2.3.3.1 Přidání podpory pro další displej

Jak jsem již zmínil v sekci 2.3.2.1, grafická aplikace není závislá na vnitřním fungování displeje. Díky tomu funkčně rozlišujeme pouze mezi displeji grafickými a textovými a specifikujeme jejich rozměry. Řadič displeje specifikujeme pouze kvůli správnému odeslání jeho identifikátoru do zařízení. Tím se přidání nového displeje stává o mnoho jednodušší.

Pro přidání řadiče displeje je potřeba vytvořit pro něj nový identifikátor. Ten by měl být o 1 větší, než předchozí. Dále je třeba do pole *driverType* přidat identifikátor typu displeje (grafický/znakový) na pozici odpovídající identifikátoru řadiče.

Po přidání řadiče může přiřazení vypadat například takto:

```
public const int KS0108 = 1;
public const int HD44780 = 2;
public const int T6963 = 3;

/* Assigning display type identifiers to display driver
    identifiers */
static public int [] driverType = {
  NONE,
  GRAPHIC,
  CHARACTER,
  GRAPHIC
};
```

Pro přidání nového displeje je třeba editovat pouze třídu *Display*. Do statického pole *types* typu *DisplayType* přidáme nový prvek.

Konstruktor třídy *DisplayType* požaduje tyto parametry:

***int* type** řadič displeje z následující množiny konstant (pokud nebyly přidány další řadiče):

NONE žádný (používá se pouze pro inicializaci po spuštění)

KS0108 grafický

HD44780 znakový

***int* width** šířka displeje, v pixelech pro grafický a ve znacích pro znakový displej

***int* height** výška displeje, v pixelech pro grafický a ve znacích pro znakový displej

***string* name** název displeje zobrazovaný v seznamu pro výběr

A takto může ve výsledku přiřazení vypadat, pokud chceme používat dvouřákový znakový displej s 10 znaky na řádek s řadičem HD44780 (znakem % jsou označeny nově přidané řádky):

```
/* Display.cs */

static public DisplayType[] types = {
  new DisplayType(NONE, 0, 0, ""),
  new DisplayType(KS0108, 128, 64, "Graphical, 128x64"),
  new DisplayType(HD44780, 20, 4, "Character, 20x4"),
  new DisplayType(HD44780, 10, 2, "Character, 10x2")      %
};
```

Pokud budeme nahrávat příkazy ze souboru, můžeme používat i displeje, které nemáme v grafické aplikaci implementované. Náhled však nebude k dispozici.

2.3.3.2 Přidání nového příkazu a nové funkce displeje

Přidání nového příkazu do třídy *Parser* probíhá velmi podobně jako v případě přidávání do firmwaru.

1. Do třídy *Display* přidáme funkci obsluhující nový příkaz.
2. Do třídy *Parser* přidáme definici číselného identifikátoru fukce. Tento identifikátor by měl být o jedna vyšší, než identifikátor doposud posledního příkazu. Také musíme zvýšit hodnotu *CMDS_N*.
3. V téže třídě přidáme do pole *cmds* text pro vyvolání příkazu a do pole *parameters* parametry přijímané příkazem. Prvky v těchto polích musí pořadím odpovídat identifikátoru definovanému pro tento příkaz (viz. bod 2).
4. V třídě *Parser* přidáme do konstrukce *switch* obstarávající volání funkcí displeje *case* odpovídající identifikátoru příkazu (viz. bod 2).
5. Ve třídě *Commander* přidáme do statického konstruktoru přiřazení do pole *commands* typu *Command* nový prvek na pozici určenou identifikátorem příkazu (viz. bod itm:funcid2).

   Konstruktor třídy *Command* přijímá tyto parametry:

***string* label1** textový popis prvního parametru

***string* label2** textový popis druhého parametru

***string* label3** textový popis třetího parametru

***string* label4** textový popis čtvrtého parametru

***int* param1** typ prvního parametru (odpovídá typům v poli *cmds*, viz. bod 3)

***int* param2** typ druhého parametru

***int* param3** typ třetího parametru

***int* param4** typ čtvrtého parametru

***bool* isCharacter** udává, zda tento příkaz mohou zpracovávat textové displeje

***bool* getsColor** udává, zda pro tento příkaz má být k dispozici zatržítko na invertování barvy

Do vlastností parametrů nepíšeme parametr pro volbu barvy. Aplikace však počítá s tím, že tento parametr, pokud je k dispozici, se v syntaxi příkazu zapisuje jako poslední.

6. Ve třídě *Form1* vytvoříme pro daný příkaz prvek radio button.

7. Ve třídě *Form1* přiřadíme tomuto radio button jako handler události *CheckedChanged* metodu *radioDisp_CheckedChanged*.

8. Ve třídě *Form1* ve funkci *initCommander* přiřadíme pomocí metody *initRadio* třídy *Commander* radio button danému příkazu.

Ačkoli se může postup jevit na první pohled složitý, jsou všechny kroky snadno pochopitelné při pouhém pohledu na současný zdrojový kód a výsledku se snadno dobereme jeho kopírováním. Část je navíc velmi podobná postupu z úpravy firmwaru (viz. sekce 2.2.2). Z tohoto důvodu zde nebudu uvádět příklady.

2.4 Testování

Všechny níže uvedené testy byly prováděny dvakrát. Jednou pro znakový displej a jednou pro grafický. Většina testů probíhala zároveň na zařízení i v grafické aplikaci.

Všechny testy dopadly úspěšně.

2.4.1 Správné zobrazování

Úkolem tohoto testu bylo ověřit, zda všechny příkazy správně plní svou funkci. Kolekce testovacích dat obsahovala všechny příkazy zobrazitelné na daném displeji. Každý příkaz byl zastoupen v počtu řádově jednotek opakování pokaždé s jinak nastavenými parametry tak, aby se jednotlivá opakování co nejvíce různila.

2.4.2 Správné nezobrazování

Úkolem testu bylo zjistit, zda syntakticky nesprávné příkazy jsou ignorovány. Kolekce testovacích dat obsahovala neplatné příkazy a platné příkazy s neplatnými parametry. Celkem řádově desítky příkazů.

2.4.3 Odolnost programu

Úkolem těchto testů bylo ověřit stabilitu programu při ztížených podmínkách.

2.4.3.1 Velké množství příkazů

Řádově stovky příkazů bylo zasláno dávkově. Poté bylo ověřeno, že zařízení i grafická aplikace jsou schopny další komunikace.

2.4.3.2 Časová odolnost

Do zařízení bylo zasláno několik příkazů a poté asi 15 hodin volně běželo. Následovalo ověření schopnosti nadále se zařízením komunikovat.

Závěr

Cílem této práce bylo vytvořit základ univerzální platformy pro připojování LCD displejů.

Čím se mé řešení odlišuje od existujících je zejména to, že žádné jiné neumožňuje komunikovat prostřednictvím jednoho zařízení s různými řadiči displejů.

Dalším významným rozdílem je, že zařízení komunikuje prostřednictvím textového protokolu. To má nespornou výhodu pro čitelnost příkazů, zejména pokud bude uživatel psát delší skripty. Ladění příkazů prostřednictvím terminálu si pak lze ještě zpříjemnit zapnutím módu pro vracení přijatých znaků. Hlavní nevýhodou je fakt, že příkazy jsou datově objemnější, než v konkurenčních řešeních často používané jednobytové příkazy. Není však žádný problém přepsat si knihovnu *parser* tak, aby používala naprosto odlišnou syntaxi.

Dalším užitečným prvkem mého řešení je grafická aplikace, která umožňuje ještě snadnější práci se zařízením.

Současná podoba práce má k úplné univerzálnosti ještě daleko. Zatím podporuje pouze po jednom řadiči ze znakových a grafických displejů. Ovšem s ohledem na to, že je aplikace značně modulární, součástí práce je podrobný návod na rozšiřování a že jsem se ji rozhodl vydat pod otevřenou licencí, má šanci stát se řešením opravdu univerzálním.

Možnosti budoucího rozšíření jsou jasné. Základem je přidat podporu pro další řadiče, rozšířit zobrazovací schopnosti či rozšířit zařízení o další způsoby komunikace.

S přihlédnutím ke všemu výše zmíněnému si dovolím tvrdit, že jsem cíl práce splnil a vytvořil jsem produkt, který je ve svém zaměření inovativní a jedinečný, ačkoli prozatím nedokonalý.

Literatura

[1] Hajrapetjan, A.: *Programovatelný řadič displeje*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, 2009.

[2] Hitachi: *HD44780U (LCD-II)*. 1999. Dostupné z: `http://lcd-linux.sourceforge.net/pdfdocs/hd44780.pdf`

[3] Microchip Technology Inc.: *MPLAB C18 C compiler libraries*. 2004. Dostupné z: `http://ww1.microchip.com/downloads/en/devicedoc/51297c.pdf`

[4] Microchip Technology Inc.: *PIC18F2420/2520/4420/4520 Data Sheet*. 2004. Dostupné z: `http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf`

[5] Rodrigues, C.: Philips PCD8544-based LCD library for Arduino. 2010, charset.cpp. Dostupné z: `http://code.google.com/p/pcd8544/source/browse/trunk/charset.cpp?r=11`

[6] SAMSUNG Electronics: *KS0108B LCD Driver IC*. 1997. Dostupné z: `ftp://docenti.ing.units.it/arc_stud/Carrato_Giulio/Calcolatori_Elettronici_3/Docum/LCD%20ks0108%20%28chip%29.pdf`

[7] Toshiba: *Toshiba T6963*. 1997. Dostupné z: `http://www.lcd-module.de/fileadmin/eng/pdf/zubehoer/t6963c.pdf`

Příloha A

Seznam použitých zkratek

LCD Liquid Crystal Display

UART Universal Asynchronous Receiver and Transmitter

USART Universal Synchronous / Asynchronous Receiver and Transmitter

TTL Transistor-Transistor Logic

MCU MicroController Unit

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... textový soubor s popisem obsahu CD
- `exe` ..... adresář se spustitelným souborem grafické aplikace
- `src` ..... adresář se zdrojovými soubory
  - `firmware` ..... zdrojové kódy firmwaru zařízení
  - `graphapp` ..... zdrojové kódy grafické aplikace
  - `thesis` ..... zdrojový text práce ve formátu LaTeX
- `text` ..... adresář s textem práce
  - `BP_Miskovsky_Vojtech_2013.pdf` ..... text práce ve formátu PDF