České vysoké učení technické v Praze
Fakulta elektrotechnická

Programovatelná řídicí jednotka

Diplomová práce

Autor: Evžen Thöndel

leden 2004

Anotace

Tématem této diplomové práce je návrh a realizace programovatelného řídicího modulu. Práce se zabývá nejen hardwarovým řešením, ale i vývojem grafického návrhového systému, který metodou „Drag and Drop“ umožňuje jednoduchý vývoj a ladění uživatelských aplikací určených pro tento modul. Práce je tématicky členěna do dvou částí. První část popisuje strukturu a postup konstrukce tohoto modulu. Zde budou popsány vlastnosti použitých součástek a návrh desek plošných spojů. Druhá část se podrobně zabývá především popisem zmíněného grafického návrhového systému. Funkce celého zařízení bude v závěru vyzkoušena na reálné soustavě.

Annotation

The subject of this graduation theses is the design and implementation of a programmable control module. The work deals not only with hardware solution but also with development of graphic design „Drag and Drop“ system that enables an easy development and tuning of user applications designed for this module. The work is thematically structured into two parts. The first part describes the structure and the technique of the construction of this module. Properties of used components and design of the printed circuit cards are described here. The second part deals in detail above all with description of the mentioned graphic design system. The function of the whole facilities will be finally tested on a real system.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) 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 Praze dne …….. ……

podpis

Poděkování

Rád bych poděkoval všem, kteří mi pomohli při vzniku této práce. Především svému školiteli Ing. Martinu Hlinovskému za konzultace a odbornou pomoc. Dále bych rád poděkoval firmě Riston za materiálovou podporu při výrobě modulu a firmě Sea za pomoc při výrobě plošných spojů. V neposlední řadě děkuji také Gabriele Novákové za závěrečnou korekturu této práce.

Obsah

ÚVOD **8**

1. NÁVRH HARDWARU **9**

1.1. Požadavky na řešení **9**

1.2. Architektura **9**

1.3. Hardwarové funkční bloky **10**
1.3.1. Napájení 10
1.3.2. Logické vstupy 11
1.3.3. Logické výstupy 12
1.3.4. Tlačítka 13
1.3.5. LCD displej 14
1.3.6. Procesor AT89C51-RD2 14
1.3.7. Rozhraní CAN a řadič sběrnice SJA1000 15
1.3.8. Programování modulu a rozhraní RS232 16
1.3.9. Zapojení sériového programovacího kabelu 19

1.4. Konstrukce **19**
1.4.1. Horní deska plošného spoje 21
1.4.2. Spodní deska plošného spoje 22

2. SOFTWARE **23**

2.1. Požadavky na řešení **23**

2.2. Knihovny funkcí **23**
2.2.1. Knihovna pro obsluhu LCD displeje 23
2.2.2. Knihovna pro obsluhu sběrnice I^2C 25
2.2.3. Knihovna pro obsluhu rozhraní CAN 26
2.2.4. Systémová knihovna 29

2.3. Modul Programátor **32**
2.3.1. Základní vlastnosti a funkce programu 32
2.3.2. Struktura programu a uživatelské prostředí 32
2.3.3. Knihovna funkčních bloků 35
2.3.4. Simulátor 39
2.3.5. Kompilátor 40
2.3.6. Loader 41
2.3.7. Programový cyklus 42
2.3.8. Instalace programu Modul Programátor 42
2.3.9. Postup vytváření uživatelských aplikací 44

3. POUŽITÍ PRO ŘÍZENÍ **46**

3.1. Příklad použití – řízení pásového přepravníku ... 46
3.1.1. Popis systému ... 46
3.1.2. Úkol řízení ... 47
3.1.3. Řešení ... 47
3.1.4. Ověření funkce a závěr ... 50

4. PODOBNÁ ŘEŠENÍ ... 51

5. ZÁVĚR ... 52

REFERENCE ... 53

PŘÍLOHA ... 54

A. Hardware ... 54
A.1. Horní deska ... 54
A.2. Spodní deska ... 57
A.3. Obrázky modulu ... 59

B. Software ... 60
B.1. i2c.h ... 60
B.2. i2c.c ... 61
B.3. lcd.h ... 64
B.4. lcd.c ... 65
B.5. system.h ... 67
B.6. system.c ... 68
B.7. sja1000_reg.h ... 72
B.8. sja1000.h ... 77
B.9. sja1000.c ... 79
B.10. can.h ... 81
B.11. can.c ... 82
B.12. can_demo.c ... 84
B.13. Vzorový program (program.c) ... 85
B.14. Instrukční soubor LCD displeje ... 86

Obsah přiloženého CD

- Katalogové listy nejdůležitějších součástek
- Elektrická schémata
- Zdrojové kódy všech knihoven
- Program Modul Programátor (včetně zdrojových kódů)
- Program Loader
- SDCC kompilátor
- Video ukazující použití modulu v praxi
- Fotografie modulu
- Tento dokument

Obsah nevázané přílohy

- Elektrická schémata
- Obrázky desek plošných spojů

Úvod

Cílem této diplomové práce je návrh a realizace programovatelného řídicího modulu. Vlastnosti, které by měl tento modul splňovat, jsou podrobně uvedeny v oficiálním zadání. Modul by měl být určen pro nasazení v menších aplikacích, kde množina vstupních a výstupních signálů řízeného procesu není příliš veliká. Typická aplikace je ovládání přepravníkových pásů, jak bude ukázáno v závěru této práce.

Aby byl modul použitelný v praxi, je třeba vytvořit i programovou podporu. Část této práce se zabývá vývojem grafického návrhového systému, který jednoduchým způsobem umožňuje návrh uživatelských aplikací.

Obsahem práce je i srovnání mého modulu s podobným řešením, které nabízí firma Siemens.

1. Návrh hardwaru

Následující část práce se podrobně zabývá návrhem a popisem realizace univerzálního řídicího modulu. První kapitola specifikuje podrobněji požadavky na řešení, které přímo neplynou ze zadání. Druhá kapitola popisuje architekturu modulu. Zavádí pojem hardwarových funkčních bloků. Třetí kapitola se detailně zabývá popisem jednotlivých funkčních bloků. Jedná se o nejrozsáhlejší kapitolu v hardwarové části práce. Poslední, čtvrtá kapitola popisuje postup konstrukce modulu.

1.1. Požadavky na řešení

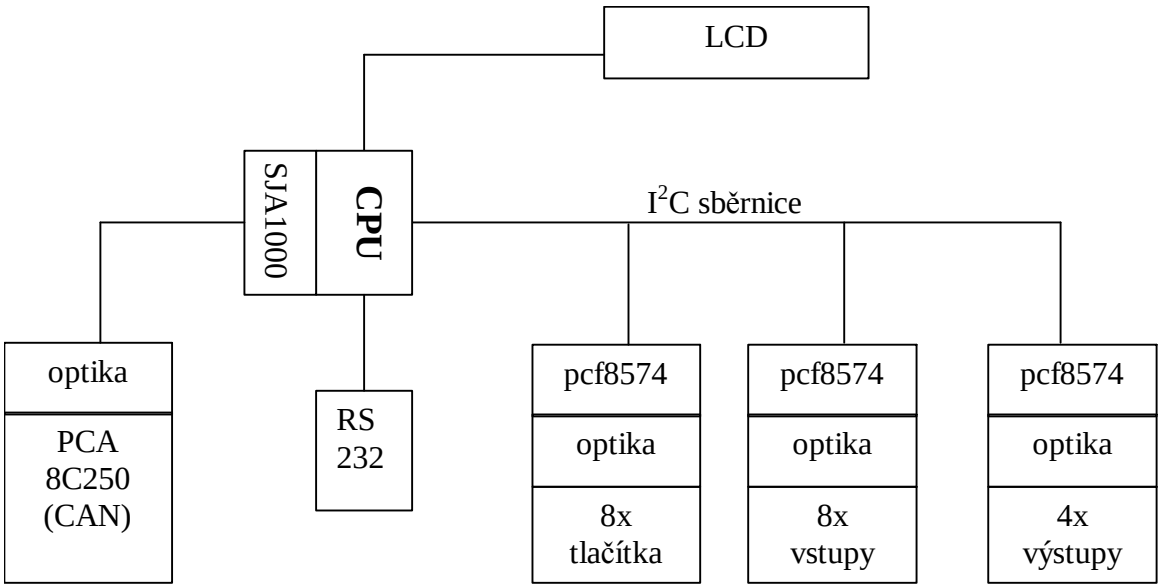
Většina požadavků na řešení plyne již ze zadání. Pokud má být řešení použitelné v praxi, musí tvořit kompaktní modul, který bude zabudován v jednom pouzdře. Nejčastěji se používají pouzdra, která umožňují připevnění na DIN-lištu. Tuto vlastnost budeme brát jako hlavní kritérium při výběru pouzdra. Další důležitou vlastností je jednoduché a univerzální napojení modulu do technologického celku. Pod touto vlastností je skryt především výběr vhodných konektorů. Nejjednodušší a nejuniverzálnější se jeví použití svorkovnice, na kterou jsou vyvedeny všechny vstupní a výstupní signály modulu.

1.2. Architektura

Bloková struktura modulu je schematicky znázorněna na obr. 1.1. Strukturu tvoří několik hardwarových funkčních bloků (HFB), které budou podrobně popsány v následujících kapitolách. HFB tvoří funkční celek, který v modulu zajišťuje určitou činnost.

Centrální HFB je výpočetní blok tvořený jednočipovým mikropočítačem řady C51 (AT89C51-RD2) a řadičem sběrnice CAN (SJA1000). Ostatní HFB jsou na tento výpočetní blok napojeny buď přímo nebo přes sběrnici.

Všechny HFB zajišťující komunikaci modulu s okolím jsou galvanicky oddělené optočleny. Výjimku tvoří sériové rozhraní RS232, které slouží pro programování modulu a není primárně určeno pro použití v aplikaci (viz kap. 1.3.8).



obr. 1.1: Blokové schéma struktury modulu

Pro zjednodušení návrhu plošného spoje jsou všechny digitální vstupy a výstupy včetně tlačítek napojeny na společnou sériovou sběrnici I^2C. Knihovna popisující čtení a zápis přes tuto sběrnici je podrobně popsána v kapitole 2.2.2.

1.3. Hardwarové funkční bloky

Následující kapitoly se podrobně zabývají popisem všech hardwarových funkčních bloků uvedených na obr. 1.1.

1.3.1. Napájení

Hlavní požadavek kladený na obvod, který zajišťuje napájení modulu, je dodržení galvanického oddělení všech funkčních bloků a ochrana modulu před změnami v napájecím napětí (přepětí). Napájecí obvod musí být též dostatečně robustní a správně pracovat v co možná nešírším spektru vstupního napájecího napětí. Rozsah vstupního napájecího napětí uvádí tab. 1.1.

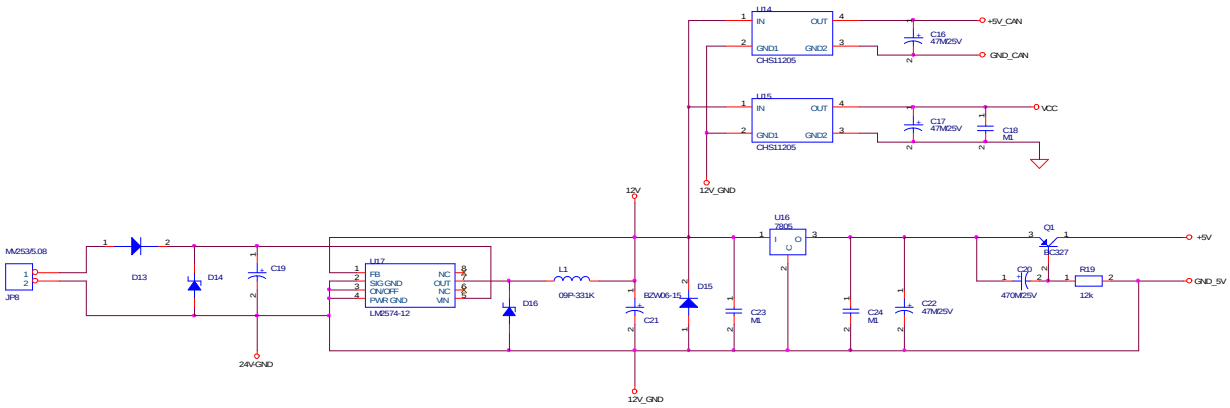
Napájecí napětí se přivádí na svorku JP8 (viz obr. 1.2). Modul je optimalizován pro napájecí napětí 24V. Za vstupní svorkou je umístěna dioda D13, která chrání obvod proti přepólování. Za touto diodou následuje transil D14, který dále nepropustí napětí větší než 33V. Správnou funkci napájecího obvodu indikuje LED dioda D18, která v případě připojeného napájení svítí zeleně.

Jádrem napájecího obvodu je obvod LM2574. Jedná se o spínaný zdroj, který pro svou funkci potřebuje jen minimum externích součástek. Pro správnou funkci spínaného zdroje

stačí zapojit Shotkyho diodu D16 a cívku L1. Hodnoty těchto součástek byly zjištěny z katalogového listu výrobce [2].

Výstupní napětí tohoto spínaného zdroje je 12V. Napětí vstupuje do DC-DC převodníků od firmy HYPEL [3], ve kterých se vytváří napájecí napětí pro funkční bloky.

Transistor Q1 spolu s tranzistorem C20 a odporem R19 tvoří zpožďovací člen, který přivede napájecí napětí na výstupní tranzistory až po náběhu procesorové části.



obr. 1.2: Schéma napájení

	min	max	jednotka
Napájecí napětí	18	33	V

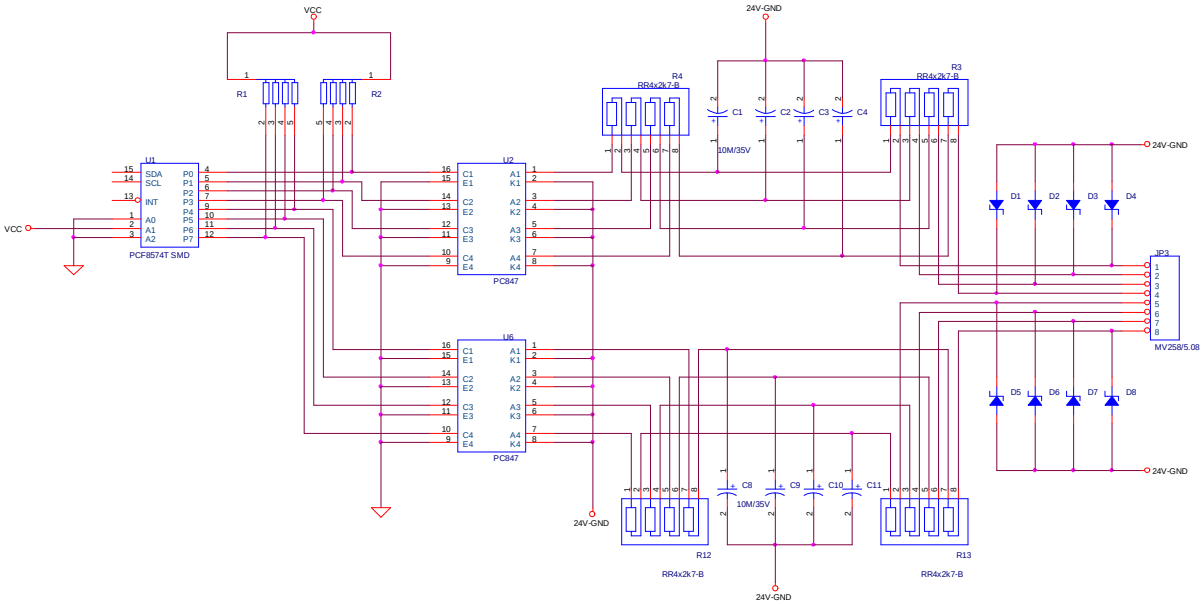
tab. 1.1: Vlastnosti napájení

1.3.2. Logické vstupy

Pro získávání informací o okolním světě a o stavu řízeného procesu je modul vybaven osmi logickými vstupy. Signál vstupuje do modulu přes svorku JP3 (viz obr. 1.3). Transily zapojené za touto svorkou hlídají překročení vstupního napětí přes kritickou hodnotu (33V). Galvanické oddělení zde zajišťují optické oddělovače PC847.

Celý blok logických vstupů je napojen na procesorovou část přes sériovou sběrnici I^2C. K tomuto účelu zde slouží obvod PCF8574 [4]. Jedná se o osmibitový expandér pro sběrnici I^2C. Adresu pro komunikaci určuje zapojení pinů A0 až A2. V případě bloku logických vstupů je tato adresa 116. Výstupem z expandéru jsou signály SDA a SCL, které jsou připojeny na společnou sběrnici I^2C. V případě změny hodnoty na vstupu se generuje signál pro přerušení na výstupu expandéru, který vyvolá v procesoru externí přerušení INT1. Toho lze s výhodou využít při programování procesoru, a to např. v případě, že potřebujeme

rychlou odezvu na změnu vstupních signálů. Přerušení procesoru INT1 je sdíleno s blokem ovládacích tlačítek (viz kap. 1.3.4).



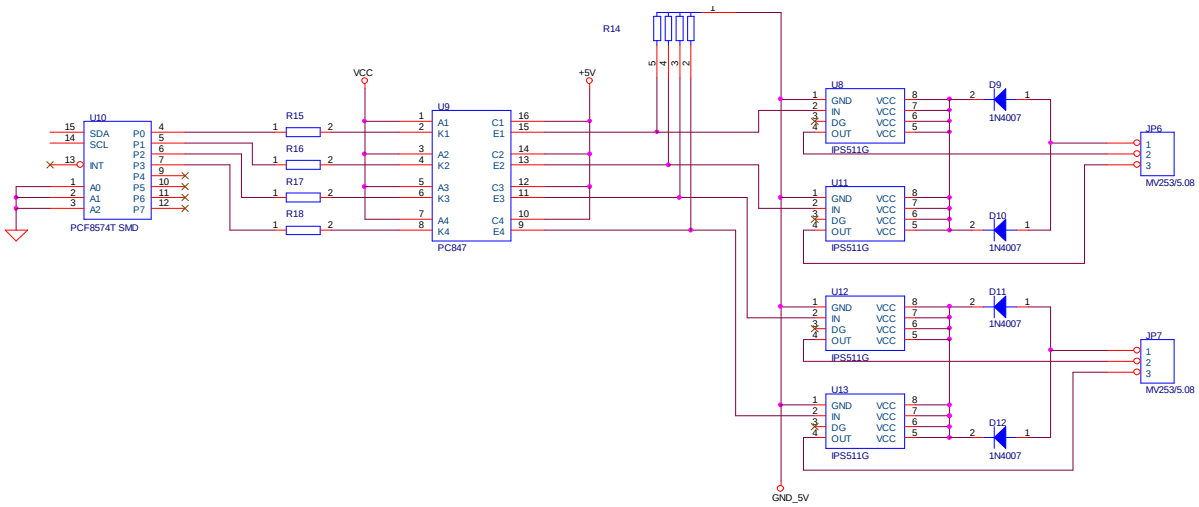
obr. 1.3: Schéma bloku logických vstupů

	Max.	Min.	Jednotka
Vstupní napětí	33	18	V

tab. 1.2: Vlastnosti logických vstupů

1.3.3. Logické výstupy

Logické výstupy slouží k vytvoření akčního zásahu do řízeného procesu. Modul obsahuje dva páry nezávislých výstupů, které mohou spínat napětí přivedené na vstupní svorku dané dvojice (obr. 1.4). Blok logických výstupů je také připojen na sběrnici I^2C. Jeho adresa je 112.



obr. 1.4: Schéma bloku logických výstupů

Základním stavebním prvkem bloku je tranzistor IPS511G. Obsahuje na čipu integrovanou logiku, která ho chrání proti zkratu a přehřátí. Tato logika automaticky vypíná tranzistor, pokud dojde k překročení kritické hodnoty proudu nebo pokud teplota na čipu stoupne nad kritickou mez. Souhrnné vlastnosti tranzistoru IPS511G jsou uvedeny v tab. 1.3. Kompletní vlastnosti a popis tohoto tranzistoru lze najít na stránkách výrobce [5].

	Hodnota	Jednotka
Max. spínané napětí	50	V
Max. proud tranzistorem	1.4	A
Kritický proud	3-7	A
Kritická teplota	165	°C

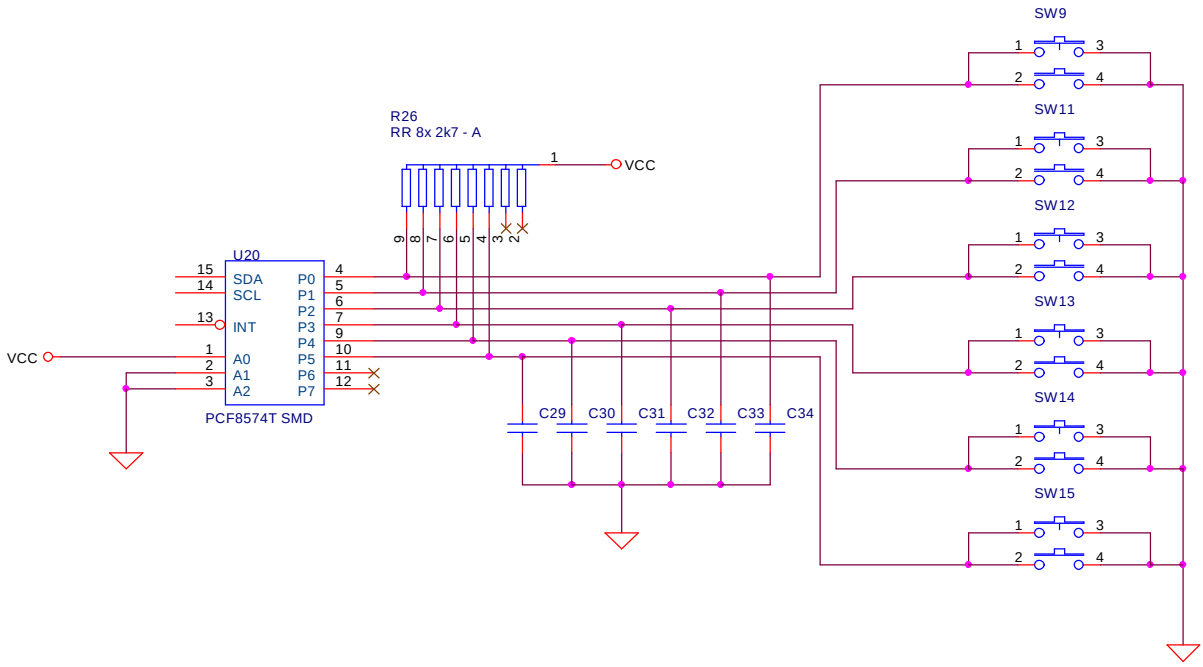
tab. 1.3: Vlastnosti tranzistoru IPS511G

Diody D9 až D12 uvedené ve schématu (obr. 1.4) chrání výstupní obvod proti přepólování. Galvanické oddělení je zde realizováno optickým oddělovačem PC847.

1.3.4. Tlačítka

Modul je vybaven šesti tlačítky, jejichž stav lze číst přes sběrnici I^2C na adrese 66. Tlačítka mohou být využita spolu s LCD displejem například pro ovládání uživatelského programu nahraného do procesoru. Na obr. 1.5 je uvedeno schéma celého bloku tlačítek. Stejně jako v případě bloku logických vstupů nebo výstupů je i zde použit obvod PCF8574,

který převádí osmibitovou bránu P0 až P7 na sériovou komunikaci. Signál přerušení je zaveden do procesoru, kde v případě stisku nebo uvolnění tlačítka vyvolá přerušení INT1.



obr. 1.5: Schéma bloku tlačítek

1.3.5. LCD displej

Pro zobrazování stavu řízeného procesu nebo stavu uživatelského programu je k procesoru připojen displej MC1602E-SYL [1]. Tento dvouřádkový displej dokáže zobrazit 2x16 znaků. Programově lze řídit také jeho podsvícení. K tomuto účelu zde slouží tranzistor Q1, který lze ovládat na portu P1.3. Pro nastavení kontrastu je na horní desce plošného spoje umístěn trimr R20 (viz nevázaná příloha).

Ovládání displeje je popsáno v knihovně LCD (kap. B.3). Základní je funkce *lcd_init*, která musí být volána vždy před zahájením práce s displejem. Tato funkce inicializuje komunikaci a nastavuje základní vlastnosti displeje. Podrobnější popis viz kapitola 2.2 a příloha.

1.3.6. Procesor AT89C51-RD2

Jádro modulu tvoří jednočipový mikropočítač řady 51 od firmy Atmel typ AT89C51-RD2. Tento osmibitový procesor zajišťuje provádění uživatelského programu a realizaci řídicích algoritmů. Všechny periférie jsou na procesor napojeny buď přímo nebo přes sběrnici. Kompletní katalogový list lze najít například na internetových stránkách výrobce [6].

AT89C51-RD2 je vyroben technologií CMOS. Na čipu je k dispozici 64kB paměti FLASH pro program a data. Navíc je na čipu rozšířená paměť XRAM o velikosti 1024 bytů a paměť EEPROM o velikosti 2048 bytů. Tato velikost paměti dovoluje použít pro vývoj programů vyšší programovací jazyky, jako je například jazyk C. Frekvence oscilátoru může nabývat hodnoty až 40MHz, což dovoluje implementovat i časově složitější algoritmy nebo použít modul pro řízení v reálném čase.

Přístup do rozšířené paměti XRAM je možný prostřednictvím instrukce MOVX. Podmínkou je nastavení registru EXTRAM na hodnotu 0. Tento registr zapíná nebo vypíná rozšířenou paměť. Při vypnuté XRAM paměti (EXTRAM=1) se instrukce MOVX odkazuje na paměť připojenou na port P0. Toho se využívá pro komunikaci procesoru s řadičem sběrnice CAN. Proto je třeba dávat dobrý pozor při implementaci programu na správné nastavení registru EXTRAM.

Port P0 je sdílen čipem SJA1000 (řadič CAN) a LCD displejem. Řízení přístupu k jednotlivým zařízením je umožněno pomocí signálů CS (P3.4) a ENABLED (P1.2).

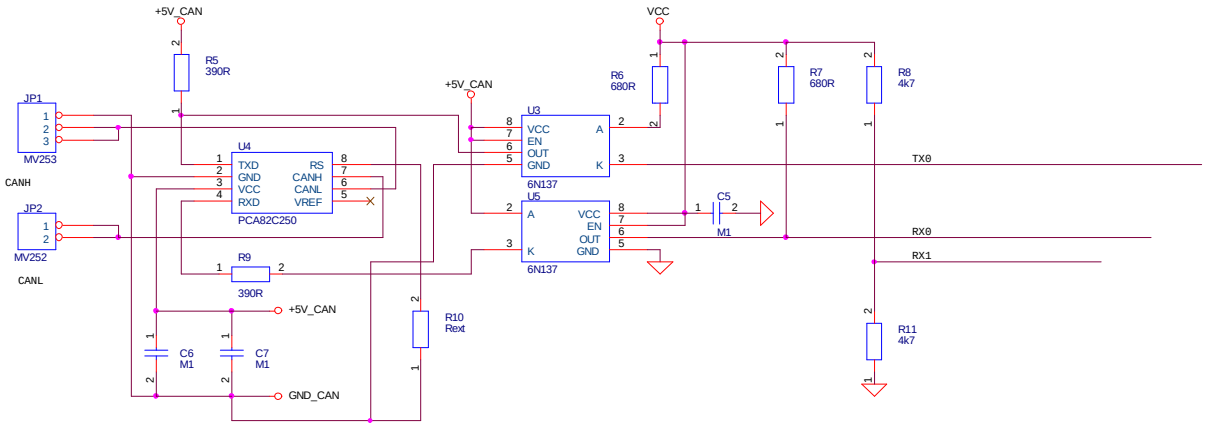
Procesor dále obsahuje na čipu časovač typu WATCHDOG. Ten může být použit pro hlídání správného chodu programu nebo jako prevence proti uváznutí.

1.3.7. Rozhraní CAN a řadič sběrnice SJA1000

Sběrnice CAN byla vyvinuta firmou Bosch v roce 1983 pro nasazení do automobilového průmyslu [7]. Postupem času se pro svou spolehlivost a nízkou cenu začala používat i v jiných oblastech, jako je například automatizační technika.

Rozhraní pro sběrnici CAN bylo do modulu zařazeno z důvodu rozšiřitelnosti modulu o další komponenty, např. o modul dalších logických vstupů a výstupů nebo modul analogových vstupů a výstupů. Tato práce se nezabývá návrhem těchto rozšiřujících modulů, ani návrhem komunikačního protokolu. Na tomto místě se autor odkazuje na případné budoucí práce. Spektrum použití zabudovaného rozhraní CAN není omezeno jen na rozšiřující moduly. Toto rozhraní může být například použito i pro komunikaci s nadřazeným řídicím systémem nebo se systémem HMI (Human Machine Interface).

Rozhraní CAN je postaveno na řadiči SJA1000. Tento čip je následníkem čipu PCA82C200, se kterým je plně kompatibilní [4]. Kompatibilita je však zaručena pouze v BASIC_CAN módu. V PELI_CAN módu pracuje SJA1000 s naprosto jinou strukturou registrů a nabízí další funkce. SJA1000 může pracovat v obou režimech, ale pouze v PELI_CAN módu jsou implementovány funkce, které jsou uvedeny ve specifikaci CAN 2.0B [7]. Popis programování je uveden podrobně v kapitole 2.2.3.



obr. 1.6: Schéma rozhraní CAN

Rozhraní mezi řadičem sběrnice (SJA1000) a fyzickou sběrnicí uvádí obr. 1.6. Ve schématu není uvedeno impedanční zakončení. Předpokládá se připojení na společnou sběrnici, která je po obou stranách správně impedančně zakončena. Rozhraní galvanicky odděluje vnitřní část modulu od vnějšího prostředí. Toto oddělení je prakticky realizováno optickými oddělovači 6N137. Základním prvkem rozhraní je obvod PCA82C250. Tento čip je převodníkem napěťových úrovní na hodnoty vhodné pro komunikaci po sběrnici CAN. Maximální přenosová rychlost je 1Mbaud. Úplný katalogový list této součástky je možné najít na internetu [4].

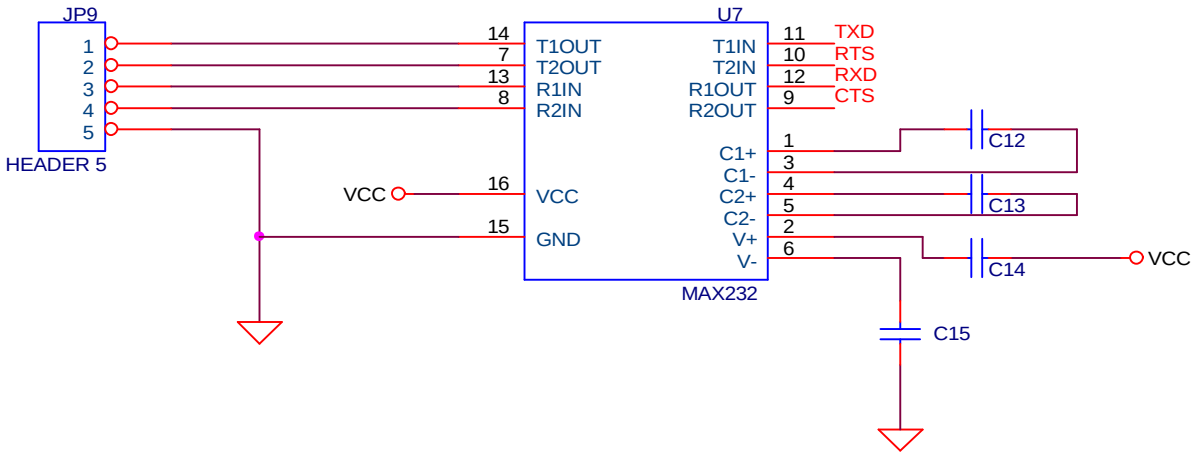
Zvláštní pozornost si zaslouží odpor R10, který nastavuje pracovní režim čipu PCA82C250. Ten může pracovat ve třech režimech: high-speed, slope-control a standby. Volbou odporu R10 lze přepínat mezi režimy high-speed a slope-control. V režimu high-speed se výstupní tranzistory spínají maximální možnou rychlostí. Je zde doporučeno používat stíněný kabel. Tento režim je nastaven, pokud je odpor R10 zkratován. Pokud má odpor R10 nenulovou hodnotu, tak je doba náběžné a sestupné hrany úměrná proudu, který protéká přes tento odpor, což má význam v případě použití nestíněného kabelu.

Rozhraní CAN je v modulu navrženo především pro zapojení rozšiřujících modulů. Předpokládá se použití krátké sběrnice se stíněným kabelem. Odpor R10 je tedy v modulu zkratován.

1.3.8. Programování modulu a rozhraní RS232

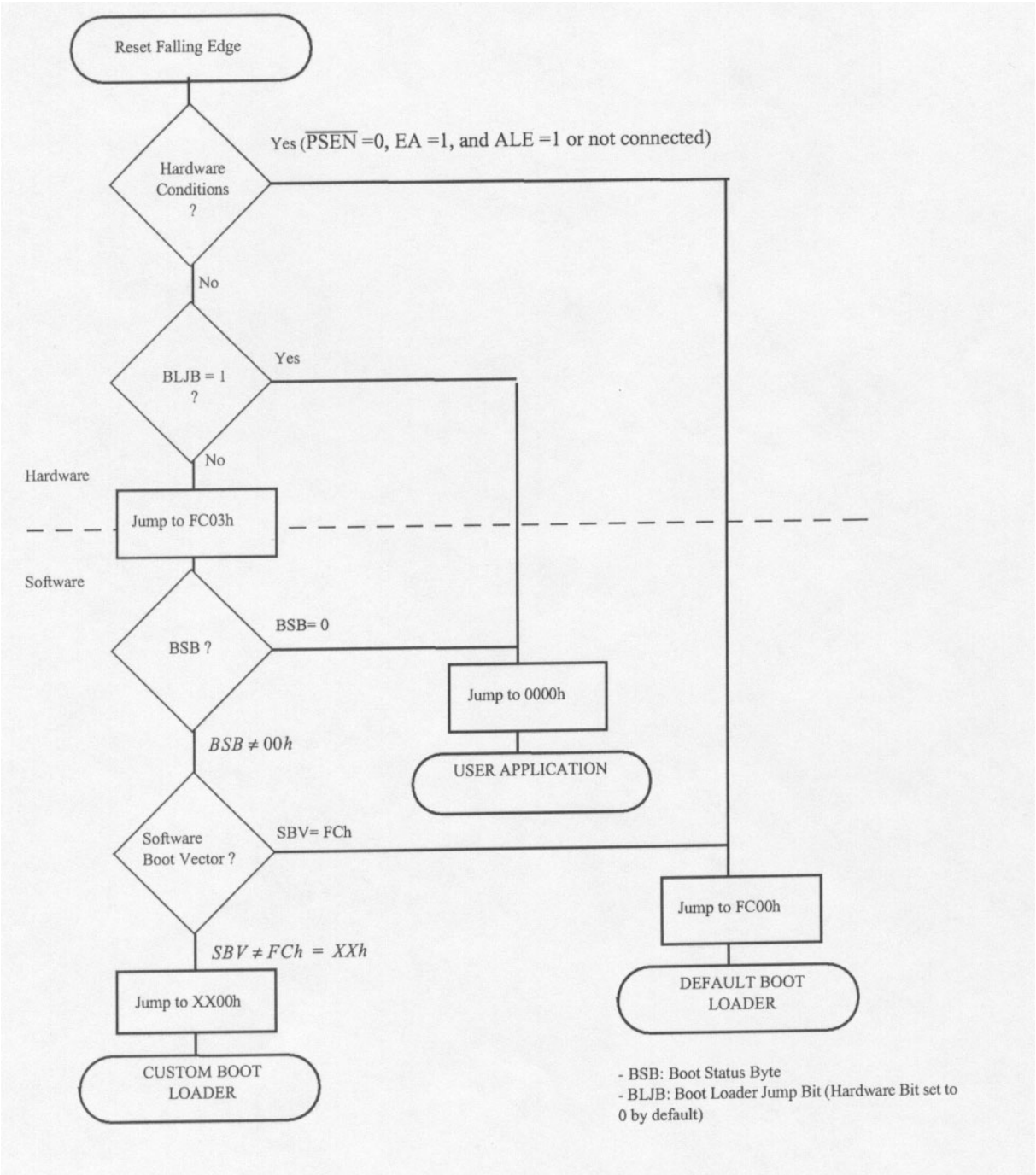
V zadání této práce se uvádí, že modul by měl umožňovat změnu řídicího programu z PC pomocí programovacího kabelu. Nyní podrobně popíšeme způsob programování modulu.

Vestavěný procesor AT89C51-DR2 (viz kap. 1.3.6) lze naprogramovat přes sériovou bránu metodou ISP (In System Programing). Tato metoda dovoluje naprogramovat procesor (modul) bez jeho vyjmutí z obvodu. Pro komunikaci s nadřazeným PC je do modulu zařazeno rozhraní RS232 (viz obr. 1.7), které převádí signál z PC na TTL úroveň a zpět. Rozhraní je postaveno na čipu MAX232, který pro svou správnou činnost potřebuje jen minimum externích součástek. Vstupy a výstupy tohoto obvodu jsou vyvedeny na konektor JP9. Popis zapojení programovacího kabelu je uveden v následující kapitole.



obr. 1.7: Schéma rozhraní RS232

Kromě základních signálů pro přenos dat po sériovém kanálu (TxD – vysílání dat, RxD – příjem dat) je zde navíc vyveden signál CTS a RTS. Signál RTS (Request To Send) je zaveden na pin P1.7 a může být využit pro libovolné účely. Standardně se tento signál používá pro žádost o vysílání [12]. Signál CTS (Clear To Send), standardně používaný pro indikaci připravenosti k vysílání dat [12], je zde napojen na signál PSEN. Hodnota signálu na vývodu PSEN řídí programování procesoru. Po resetu čte procesor tuto hodnotu a pokud je zde nastavena hodnota LOW, přechází do programovacího režimu. Na obr. 1.8 je uveden kompletní stavový diagram vývoje procesoru po resetu převzatý z katalogového listu výrobce [6].

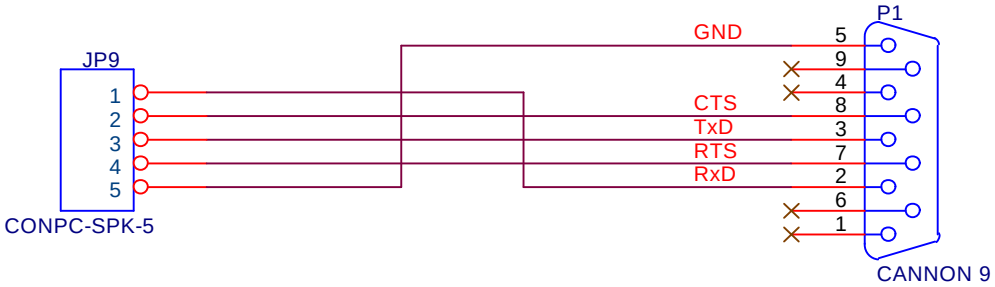


obr. 1.8: Stavový diagram bootování procesoru AT89C51-RD2

Rozhraní RS232 je primárně určeno pro programování modulu. Nic však nebrání použití tohoto rozhraní v uživatelské aplikaci, kde může například sloužit k přenosu dat do systému HMI (Human Machine Interface). Rozhraní RS232 není galvanicky odděleno od modulu, proto je třeba dávat větší pozor při použití v uživatelské aplikaci, aby nedošlo k poškození modulu. Při zapojování a rozpojování kabelu je doporučeno nejdříve celý modul odpojit od napájení.

1.3.9. Zapojení sériového programovacího kabelu

Programovací kabel slouží k přenosu dat mezi modulem a PC. Jeho zapojení uvádí obr. 1.9. Vodiče jsou popsány ze strany PC. Na straně modulu dochází k překřížení vodičů TXD a RXD.

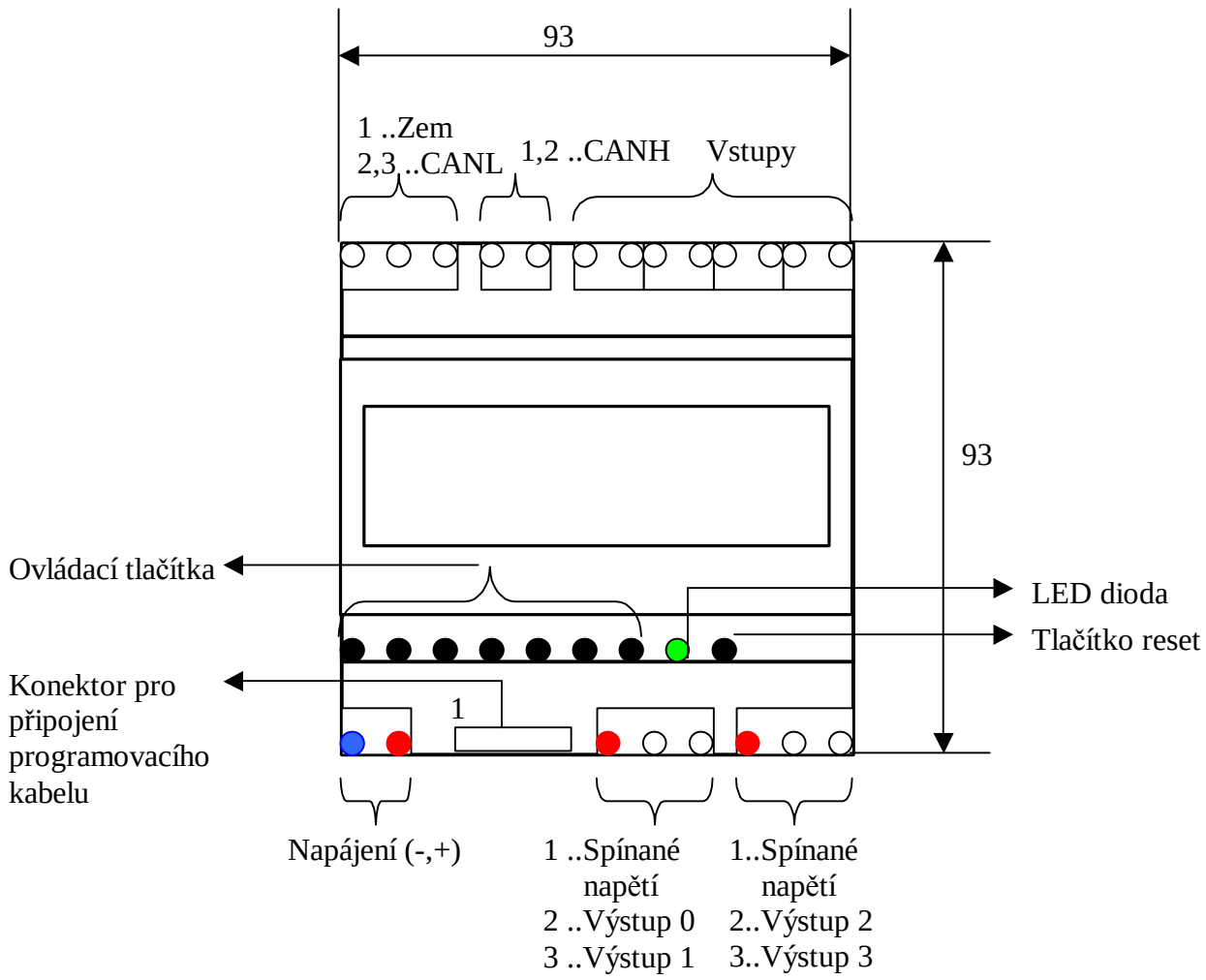


obr. 1.9: Zapojení programovacího kabelu

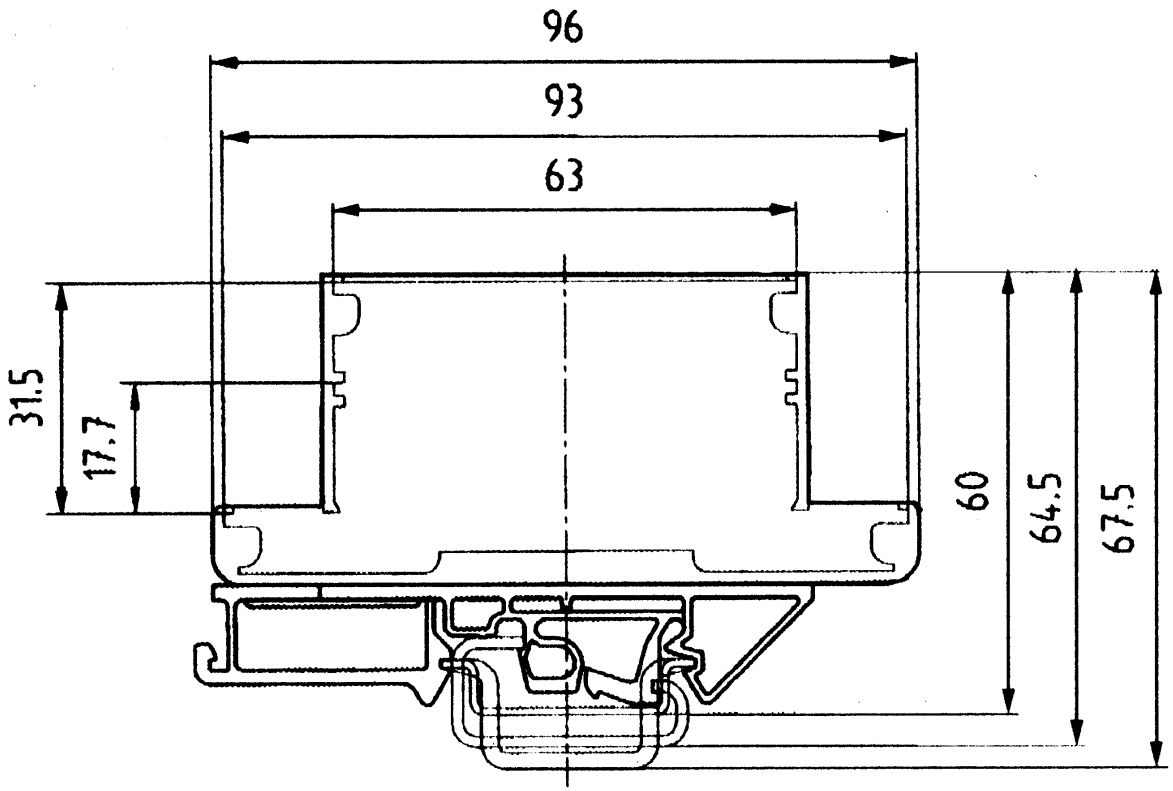
Názvy konektorů jsou převzaty z katalogového listu GM Electronic [1]. U této firmy je možné konektory i zakoupit. Kabel může být složen z několika dílů, ale jeho délka by neměla přesáhnout 15 m [12].

1.4. Konstrukce

Celé zařízení je umístěno v pouzdře WEB-B8 [1], které umožňuje připevnění na DIN lištu. Celé provedení tak vytváří kompaktní modul, který je možné nasadit v průmyslové praxi. Z vlastností pouzdra WEB-B8 (viz obr. 1.11) vyplývá rozdělení celého řešení na dvě desky plošného spoje, které jsou spolu propojeny pomocí konektorů JP4 a JP11 (viz nevázaná příloha). Celkové uspořádání a rozmístění vývodů uvádí obr. 1.10.



obr. 1.10: Rozložení vývodů



obr. 1.11: Rozměry pouzdra

Následující dvě kapitoly se věnují konstrukci obou desek plošného spoje. Pro návrh byl použit program OrCAD verze 9.1. Kompletní podklady pro výrobou obou desek jsou umístěny na přiloženém CD.

1.4.1. Horní deska plošného spoje

Horní deska plošného spoje obsahuje výpočetní blok, ovládací tlačítka a rozhraní pro připojení LCD displeje.

LCD displej je k horní desce připojen přes konektor JP10 pomocí šestnáctižilového plochého kabelu. Dříve než displej zapojíme, je nutné jej upravit tak, aby bylo možné ovládat jeho podsvícení. Z výroby je podsvícení natrvalo připojeno na +5V. Úprava spočívá v odstranění odporu R12 o hodnotě 0R a posunutí odporů R7 a R8 o hodnotě 15R na pozici R9 a R10. Pozice jednotlivých odporů jsou názorně zobrazeny na spodní straně tohoto displeje. Nastavení kontrastu se provádí pomocí trimru R20, který je třeba nastavit při oživování modulu. Displej je k horní desce připojen přes distanční sloupky, pro které jsou na odpovídacích místech připraveny otvory.

Zkratovací propojky JP5 a JP12 umístěné na horní desce jsou pozůstatky vývoje a pro správnou funkci celého modulu by měly zůstat propojeny. Význam může mít propojka JP5, která po rozpojení zabrání přeprogramování modulu.

První verze horní desky plošného spoje obsahuje chybu, kterou je třeba odstranit. Chyba spočívá v připojení řadiče CAN (SJA1000) na port P2 procesoru. Čip SJA1000 se chová jako rozšířená paměť, a proto musí být připojen na port P0 (viz kap. 1.3.6). Chybu odstraníme propojením portu P0 s portem P2 na desce plošného spoje. K propojení je třeba napájet osm drátků na spodní stranu desky. Procesor zasuneme do patice tak, aby nožičky brány P2 zůstaly nezapojeny. Tím máme zapojen čip SJA1000 na port P0 společně s displejem. Řízení přístupu na jednotlivá zařízení je třeba řešit pomocí signálů CS a E. V následujících verzích desky plošného spoje bude třeba zaměnit zapojení portů P0 a P2.

Konektor JP11 slouží k propojení se spodní deskou plošného spoje. K propojení se používá dvouřadá vidlice, kterou připájíme na stranu spojů. Na spodní desce je na odpovídajícím místě na straně součástek napájen konektor, který umožňuje zasunutí uvedené vidlice (viz následující kapitola). Spojení obou desek je tedy rozebíratelné, což usnadňuje oživování modulu.

Obrázek desky plošného spoje spolu se seznamem součástek a schématem osazení je uveden v příloze a na přiloženém CD.

1.4.2. Spodní deska plošného spoje

Spodní deska plošného spoje obsahuje bloky vstupů a výstupů, blok rozhraní RS232 a CAN a napájecí blok.

Deska je spojena s horní deskou pomocí konektoru JP4. Tento konektor je v katalogu GM Electronic [1] označen jako BL214G.

Stejně jako první verze horní desky, tak i první verze spodní desky obsahuje chybu. Chyba je v příliš úzkém pouzdru pro tranzistory IPS511G. Chybu lze odstranit tak, že nožičky těchto součástek lehce přihneme pod pouzdro a součástku pak opatrně připájíme do odpovídajícího místa.

Podklady pro výrobu a osazení desky jsou opět uvedeny v příloze a na přiloženém CD.

2. Software

Druhá část této práce se zabývá programovou podporou pro daný modul. Tématicky je dělena do tří kapitol. První kapitola specifikuje požadavky kladené na programové řešení. Zde budou blíže rozvedeny body uvedené v zadání. Druhá kapitola popisuje základní knihovny funkcí, které byly napsány pro obsluhu jednotlivých částí modulu, jako je například LCD displej nebo sběrnice I^2C. Poslední, třetí kapitola je věnována grafickému návrhovému programu. Ten umožňuje jednoduchý vývoj uživatelských aplikací.

2.1. Požadavky na řešení

Hlavním úkolem této části práce je vyvinout grafický návrhový systém. Aby toto bylo možné, je třeba nejprve vytvořit knihovny, které popisují obsluhu jednotlivých hardwarových funkčních bloků. Samotný návrhový systém by měl mít intuitivní ovládání plynoucí ze zásad operačního systému. Samozřejmostí by měla být i podrobná nápověda ke všem funkcím programu, která bude dostupná z jakéhokoliv místa v programu.

2.2. Knihovny funkcí

Knihovny funkcí obsahují základní funkce pro obsluhu jednotlivých periférií modulu. Knihovny jsou napsány v jazyce C a na přiloženém CD jsou k dispozici i ve verzi přeložené překladačem SDCC [18]. V následujících podkapitolách budou podrobně popsány jednotlivé knihovny. Úplné výpisy zdrojových kódů lze najít v příloze nebo na přiloženém CD.

2.2.1. Knihovna pro obsluhu LCD displeje

Tato knihovna obsahuje základní funkce pro obsluhu LCD displeje, jehož vlastnosti a popis zapojení byly uvedeny v kapitole 1.3.5.

Celá komunikace s displejem je obsažena v jedenácti instrukcích, jejichž úplný popis převzatý ze stránek firmy Data Image Corporation [19] lze najít v příloze (viz kap. B.14).

Dále je uveden popis funkcí obsažených ve zmíněné knihovně. Tyto funkce obalují některé LCD instrukce. Všechny funkce z této knihovny začínají prefixem *lcd*.

void lcd_init(unsigned char setup)

Tato funkce musí být volána před všemi operacemi s LCD displejem. Parametrem funkce je nastavení kurzoru. Parametr může nabývat kombinací následujících hodnot.

CR_SHOW	Zobrazí kurzor na aktuální pozici.
CR_BLINK	Pokud je nastaven parametr CR_SHOW, bude zobrazený kurzor blikat.

tab. 2.1: Význam hodnot parametru SETUP

Funkce při svém spuštění zapne displej a nastaví způsob komunikace na osmibitový. Displej je teoreticky schopen komunikovat i čtyřbitově, každá instrukce je potom posílána ve dvou cyklech. Dále tato funkce vymaže obsah displeje, nastaví vlastnosti kurzoru podle výše uvedeného parametru a přesune ho na počáteční pozici.

void lcd_close(void)

Funkce vypne displej. Pokud dále v aplikaci displej nebudeme potřebovat, je vhodné jej vypnout. Sníží se tím celková energetická spotřeba modulu.

void lcd_putch(char znak)

Funkce zapíše znak na aktuální pozici displeje a posune kurzor o jednu pozici směrem doprava. Parametrem funkce je znak, který se má vypsat.

void lcd_write(char *str)

Funkce napíše na displej řetězec znaků zakončený podle konvence jazyka C znakem NULL. Parametrem je ukazatel na tento nulou zakončený řetězec. Funkce nevrací žádnou hodnotu.

void lcd_clr(void)

Funkce vymaže displej a přesune kurzor na počáteční pozici.

void lcd_ledon(void)

Funkce zapne podsvícení displeje.

void lcd_ledoff(void)

Funkce vypne podsvícení displeje.

void lcd_at(unsigned char x, unsigned char y)

Nastaví kurzor na pozici danou parametry. Parametr *x* může nabývat hodnot v intervalu od 0 do 15. Tento parametr určuje pozici v rámci jednoho řádku. Parametr *y* nabývá hodnoty 0 nebo 1 a definuje číslo řádku.

2.2.2. Knihovna pro obsluhu sběrnice I^2C

Většina periférií není připojena přímo na procesor, ale komunikuje s ním přes sdílenou sběrnici I^2C. Tato sériová sběrnice byla použita z důvodu zjednodušení návrhu a realizace plošného spoje. Jsou na ni napojeny bloky logických vstupů a výstupů a tlačítka. Každé periférii připojené na sběrnici musí být přidělena jednoznačná adresa. V případě modulu jsou adresy pevně zadány hardwarovým uspořádáním a jejich hodnoty jsou uvedeny v následující tabulce.

Název bloku	Adresa
Logické vstupy	116
Logické výstupy	112
Tlačítka	66

tab. 2.2: Adresy bloků připojených na sběrnici I^2C

Knihovna pro obsluhu této sběrnice obsahuje základní funkce pro zápis a čtení dat. Všechny funkce z této knihovny začínají prefixem *i2c*.

unsigned char i2c_readbyte(unsigned char addr)

Funkce přečte jeden byte ze zadané adresy. Adresa se zadává do parametru funkce. Možné hodnoty uvádí tab. 2.2. Funkce vrací přečtenou hodnotu.

unsigned char i2c_sendbyte(unsigned char addr, unsigned char value)

Funkce pošle jeden byte na zadanou adresou. Adresa se zadává do parametru *addr*. Hodnota vysílaného bytu je uložena v parametru *value*. Pokud je operace úspěšná, vrací funkce hodnotu 1.

2.2.3. Knihovna pro obsluhu rozhraní CAN

Knihovna pro obsluhu rozhraní CAN nabízí v současné verzi jen základní funkce pro obsluhu rozhraní. Případné navazující práce, zabývající se tímto tématem, by mohly knihovnu dále rozšířit. Rovněž by se tyto práce mohly zabývat návrhem a realizací komunikačního protokolu mezi rozšiřujícími moduly, pro které je toto rozhraní v modulu primárně určeno.

Jak již bylo zmíněno v kapitole 1.3.7, je celé rozhraní pro sběrnici CAN postaveno na řadiči SJA1000. Čip se vůči procesoru chová jako rozšířená paměť, a proto programování spočívá v zápisu dat do jeho příslušných registrů .

Nejdůležitější soubor poskytující základ pro další vývoj celé knihovny je soubor popisující strukturu registru čipu SJA1000. Tento soubor se jmenuje *SJA1000_reg.h* a jeho kompletní výpis je uveden v příloze a na přiloženém CD. Lze jej také najít na stránkách výrobce [4].

Dále se knihovna dělí do dvou částí. První část popisuje ovládání čipu SJA1000 v basic-módu. Rozšířené vlastnosti, které čip nabízí v PeliCAN-módu, nejsou v současné verzi knihovny implementovány. Případné rozšíření by nemělo být problematické za použití již zmíněného souboru *SJA1000_reg.h*. Druhá část knihovny nabízí funkce pro vysílání a příjem dat přes sběrnici CAN.

Pro odzkoušení funkce komunikace přes sběrnici CAN je na přiloženém CD k dispozici demonstrační program, který čte data ze sběrnice a vypisuje je na LCD displej. Po stisknutí prvního ovládacího tlačítka se na sběrnici zapíše řetězec znaků: „CAN TEST“.

Funkce pro ovládání čipu SJA1000 v basic-módu

Funkce ovládající čip SJA1000 v basic módu jsou obsaženy v souborech *SJA1000.h* a *SJA1000.c*. Všechny funkce z těchto souborů začínají prefixem *sja*. Před jakýmkoli přístupem na čip je třeba nejprve zavolat funkci *sja_enable*, která uvolní přístup. Po ukončení komunikace s čipem je vhodné tento přístup opět zakázat, protože během komunikace není možný přístup do rozšířené paměti integrované na procesoru.

void sja_enable(void)

Funkce uvolní přístup na čip SJA1000 tím, že nastaví signál CS (Chip Select) na hodnotu LOW a nastaví příznak EXTRAM v registru AUXR. Nastavením příznaku EXTRAM se veškeré přístupy do externí paměti přesměrují přes port P0 na procesoru, kde se

očekává připojené zařízení, které realizuje rozšířenou paměť. V našem případě se jedná o čip SJA1000.

void sja_disable(void)

Funkce zakáže přístup na čip SJA1000. Veškeré odkazy do externí paměti budou směrovány do paměti integrované na čipu.

void sja_reset(void)

Funkce přepne čip do režimu Reset. V tomto režimu je možná změna nastavení na čipu (přenosová rychlost, maska zprávy atd.).

void sja_clrreset(void)

Funkce přepne čip zpět do provozního režimu.

void sja_baudrate(int baudrate)

Funkce nastaví přenosovou rychlost. Přenosovou rychlost je možné nastavit pouze v režimu Reset. Maximální rychlost je 1 Mbit/s.

void sja_waitfor_readytosend(void)

Funkce čeká, dokud nebude čip připraven k vysílání. Čip nemusí být připraven, pokud předchozí odesílaná zpráva nebyla ještě úspěšně zpracována a odesílací registr je tedy zaplněný.

unsigned char sja_newdata(void)

Funkce vrací nenulovou hodnotu, pokud byla přijata nová zpráva. Přijímané zprávy se ukládají do přijímacího registru.

Ovládací funkce sběrnice CAN

Ovládací funkce sběrnice CAN jsou obsaženy v souborech *can.h* a *can.c*. Funkce pro svou činnost využívají ovládací funkce čipu SJA1000 a všechny funkce začínají prefixem *can*.

void can_init(struct can_init_data *init_data)

Funkce inicializuje rozhraní CAN. Parametrem funkce je ukazatel na strukturu, která obsahuje základní parametry pro nastavení přenosu. Struktura obsahuje položky *AcceptCode (ACR)* a *AcceptMask (AMR)*, které slouží pro nastavení filtru zpráv. Každá zpráva přijímaná čipem SJA1000 prochází nejdříve tímto filtrem, ve kterém je porovnáno prvních osm bitů identifikátoru zprávy s bity v registru *AcceptCode*. Obsah registru *AcceptMask* určuje, které bity se budou porovnávat. Porovnávají se jen ty bity, u kterých je v registru *AcceptMask* nastavena nula. Příklad nastavení filtru je uveden v tab. 2.3.

	MSB										LSB
ACR	0	1	1	1	0	0	1	0			
AMR	0	0	1	1	1	0	0	0			
ID	0	1	x	x	x	0	1	0	x	x	x

tab. 2.3: Příklad nastavení filtru zpráv. Pouze zprávy s identifikátorem ID budou akceptovány (x znamená, že na dané hodnotě nezáleží).

Pokud identifikátor zprávy neodpovídá zadanému filtru, je zpráva zahozena. Tím je ulehčena práce procesoru, kterému odpadá práce se tříděním zpráv. Výše popsaný způsob filtrovaní zpráv funguje jen v basic-módu čipu SJA1000. V PeliCan-módu se nabízí mnohem více možností nastavení filtru zpráv [4].

Poslední položkou zmíněné struktury je baudrate, kde je obsažena informace o přenosové rychlosti. Položka je typu integer a její hodnota odpovídá přenosové rychlosti udané v kbit/s. Možné hodnoty jsou 100, 250, 1000.

void can_send(struct can_message *message)

Funkce odešle zprávu. Parametrem funkce je ukazatel na strukturu obsahující data zprávy. Struktura obsahuje dvoubytové pole *ID*, ve kterém je uložen identifikátor zprávy. Význam má jen prvních jedenáct bitů, které budou fyzicky odeslány na sběrnici. Další položkou struktury je osmibytové pole *can_data*, ve kterém jsou uložena samotná data.

unsigned char can_receive(struct can_message *message)

Funkce přečte zprávu. Funkce nejprve testuje, jestli je nová zpráva obsažena v přijímacím registru čipu SJA1000. Pokud ano, zkopíruje data do struktury, jejíž ukazatel je

funkci předán v parametru. Funkce v případě úspěšného přijetí nové zprávy vrací nenulovou hodnotu.

2.2.4. Systémová knihovna

Systémová knihovna obsahuje funkce, které se používají při sestavování uživatelského programu, který byl vytvořen v grafickém návrhovém systému (viz kap. 2.3). Program vytvořený v tomto návrhovém systému tvoří logickou síť vzájemně propojených bloků. Tyto bloky představují různé funkce (například logická konjunkce nebo disjunkce). Funkce obsažené v této knihovně představují realizaci jednotlivých bloků.

Každá funkce realizující nějaký blok má jeden společný parametr *ID*. Toto jednobytové číslo představuje jednoznačný identifikátor bloku a je automaticky přidělováno návrhovým systémem. Další parametry představují vstupní hodnoty bloku.

Každý blok má v paměti procesoru přiděleny dva bity paměti pro ukládání výsledků. Jeden bit slouží pro ukládání nových výsledků v daném cyklu a druhý obsahuje výsledek bloku v minulém cyklu. Struktura každé funkce realizující některý blok vypadá následovně. Funkce načte vstupní parametry a provede požadovanou operaci (konjunkce, disjunkce atd.). Výsledek uloží do paměti na místo dané parametrem *ID*. Funkce vrací výsledek operace. Další informace o sestavování uživatelského programu lze najít v kapitole 2.3.5.

Dále budou podrobně popsány jednotlivé funkce z této knihovny.

void ReadInput(void)

Funkce přečte hodnoty vstupních logických signálů a hodnoty uloží do pole *Input*. Tato funkce je součástí hlavního programového cyklu, jehož popis je uveden v kapitole 2.3.7.

void WriteOutput(void)

Tato funkce přepíše hodnoty uložené v poli *Output* na fyzické výstupy. Pole *Output* je aktualizováno při každém průchodu programovým cyklem. Tato funkce je také součástí hlavního programového cyklu.

void ReadButtons(void)

Funkce přečte stavy tlačítek a hodnoty uloží do pole *Button*. Funkce je součástí hlavního programového cyklu.

unsigned char OR(unsigned char ID, unsigned char LValue, unsigned char RValue)

Realizace bloku logické disjunkce. Jak bylo popsáno výše, představuje parametr *ID* jednoznačný identifikátor bloku a parametry *LValue* a *RValue* jsou vstupní hodnoty bloku.

unsigned char AND(unsigned char ID, unsigned char LValue, unsigned char RValue)

Realizace bloku logické konjunkce.

unsigned char NOT(unsigned char ID, unsigned char Value)

Realizace bloku logické negace.

unsigned char UP_firstcall(unsigned char ID, unsigned char Value)

Realizace bloku, který reaguje na náběžnou hranu vstupního signálu. Tato funkce se volá při prvním průchodu daným blokem v rámci programového cyklu. Při dalším průchodu se volá funkce *UP_nextcall*.

unsigned char UP_nextcall(unsigned char ID)

Tato funkce se volá při druhém a dalším průchodu blokem reagujícím na náběžnou hranu vstupního signálu.

unsigned char DOWN_firstcall(unsigned char ID, unsigned char Value)

Funkce realizující blok, který reaguje na sestupnou hranu vstupního signálu. Realizace je stejná jako v případě bloku reagujícího na náběžnou hranu. Tato funkce se volá při prvním průchodu daným blokem.

unsigned char DOWN_nextcall(unsigned char ID)

Funkce se volá při druhém a dalším průchodu blokem, který reaguje na sestupnou hranu vstupního signálu.

unsigned char d(unsigned char ID, unsigned char Value)

Realizace operátoru zpoždění o jeden krok. Funkce vrací hodnotu, která byla na vstupu v minulém programovém cyklu.

unsigned char RS(unsigned char ID, unsigned char R, unsigned char S)

Tato funkce realizuje RS klopný obvod. Vstupní parametry mají následující význam: R je signál reset, S je signál set. Funkce vrací výstupní hodnotu tohoto klopného obvodu.

unsigned char RSnon(unsigned char ID, unsigned char R, unsigned char S)

Funkce má stejný význam jako funkce předešlá. Rozdíl spočívá v tom, že tato funkce vrací negovanou hodnotu výstupu klopného obvodu.

unsigned char GetNewValue(unsigned char ID)

Funkce vrací hodnotu z pole *newmem*. V tomto poli jsou uloženy výsledné hodnoty bloků. Parametr *ID* specifikuje identifikátor bloku, jehož výsledná hodnota má být přečtena. Tato funkce se používá v případě vícenásobného průchodu bloku v rámci jednoho programového cyklu. Díky tomu se operace každého bloku provede maximálně jednou v rámci jednoho cyklu (více viz kap. 2.3.5).

unsigned char GetValue(unsigned char ID)

Tato funkce vrací výslednou hodnotu bloku z minulého programového cyklu. Používají ji některé speciální bloky (např. blok náběžné a sestupné hrany nebo operátor zpoždění).

void InitScan(void)

Tato funkce se volá na začátku programového cyklu. Funkce kopíruje hodnoty obsažené v poli *newmem* do pole *mem* (pole obsahující předchozí výsledné hodnoty).

void LCD(char *Line1, char *Line2, unsigned char LED, unsigned char Value)

Funkce realizující blok LCD displeje. Parametry Line1 a Line2 obsahují ukazatele na nulou zakončené řetězce znaků, které mají být vypsány na první, respektive druhý řádek displeje. Parametr LED udává, zda má být podsvícení displeje zapnuto nebo vypnuto.

void mem_init(void)

Funkce nuluje obsah polí *mem* a *newmem*.

2.3. Modul Programátor

2.3.1. Základní vlastnosti a funkce programu

Cílem této práce bylo vytvořit nejen hardwarové řešení, ale i plnou programovou podporu. Následující kapitoly popisují podrobně návrhový software, který umožňuje jednoduchým způsobem vytvořit a odladit aplikaci pro výše popsaný modul. Tento program neklade na uživatele žádné speciální požadavky. Postačí jen základní znalost práce pod operačním systémem Windows. Uživatelé, kteří nemají žádnou zkušenost s programováním jednočipových mikropočítačů, mohou tak pomocí tohoto nástroje vytvořit a odladit celou aplikaci.

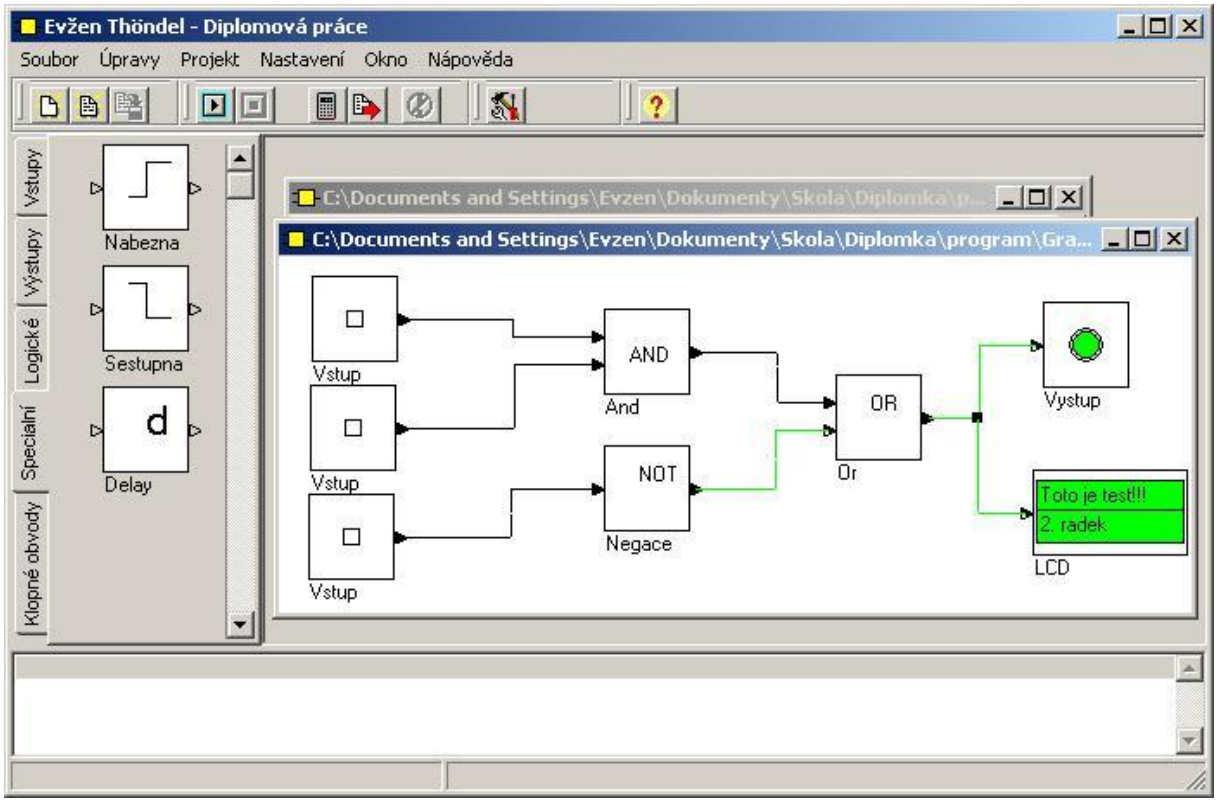
Modul Programátor je grafický návrhový systém. Aplikace vytvořené v tomto programu tvoří logickou síť navzájem propojených funkčních bloků. Velkou výhodou je možnost simulace během návrhu, která umožňuje odhalit chyby ještě před odesláním aplikace do modulu.

Aby bylo možné navrhovat i složité aplikace, je potřeba dostatečně velká množina funkčních bloků. Program v této verzi nabízí zatím jen základní bloky. Rozšíření je možné, jak bude popsáno v kapitole 2.3.3. Na tomto místě se autor odvolává na případné budoucí práce na toto téma.

Program byl vyvíjen v prostředí C++ Builder ve verzi 5.0.

2.3.2. Struktura programu a uživatelské prostředí

Program byl vytvořen pro operační systém Windows. Struktura programu a ovládání vychází z koncepce tohoto operačního systému. Na obr. 2.1 je základní okno programu.



obr. 2.1: Hlavní okno programu s otevřeným projektem

Hlavní okno se skládá z ovládacího menu, panelu nástrojů, knihovny funkčních bloků a stavového panelu. Může také obsahovat dceřiná okna otevřených projektů.

Ovládací menu

Ovládací menu obsahuje všechny operace, které lze s programem provádět. Výjimku tvoří funkční bloky, pro které je vytvořeno zvláštní ovládací menu (viz kap. 2.3.3). Struktura menu vypadá následovně.

- Soubor
  - Nový – Založí nový projekt.
  - Otevřít – Otevře existující projekt.
  - Uložit – Uloží otevřený projekt do souboru.
  - Uložit jako – Uloží otevřený projekt do souboru pod jiným jménem.
  - Zavřít – Zavře otevřený projekt.
  - Zavřít vše – Zavře všechny otevřené projekty.
  - Konec – Ukončí program.
- Úpravy
  - Smazat výběr – Smaže všechny vybrané objekty.

  - Vlastnosti – Zobrazí vlastnosti vybraného objektu.
- Projekt
  - Kompilace – Přeloží aktivní projekt (více viz kap. 2.3.5).
  - Naprogramovat – Pošle přeložený projekt do modulu (viz kap. 2.3.6).
  - Start simulace – Spustí simulátor (více viz kap. 2.3.4).
  - Stop simulace – Zastaví simulátor.
- Nastavení
  - Vlastnosti – Zobrazí základní nastavení programu (více níže).
- Okno – Uspořádá okna otevřených projektů.
  - Kaskáda
  - Uspořádat horizontálně
  - Uspořádat vertikálně
  - Minimalizovat vše
  - Uspořádat ikony
- Nápověda
  - Obsah nápovědy
  - O programu – Základní informace o programu.

Panel nástrojů

Panel nástrojů slouží k rychlé volbě nejčastějších funkcí. V hlavním okně je umístěn přímo pod menu. Jeho struktura je popsána v tab. 1.1.

	Založí nový projekt.
	Otevře existující projekt uložený na disku počítače.
	Uloží otevřený projekt na disk počítače.
	Spustí simulaci.
	Zastaví simulaci.
	Přeloží aktivní projekt.
	Pošle přeložený projekt do modulu.
	Přeruší probíhající překlad nebo nahrávání dat do modulu.
	Základní vlastnosti programu.
	Nápověda.

tab. 2.4 Struktura panelu nástrojů

Nastavení programu

Nastavení programu se spustí volbou položky *Vlastnosti* z menu *Nastavení*. V zobrazeném dialogovém okně jsou dvě záložky pro nastavení vlastností kompilátoru a loaderu, který nahrává uživatelem vytvořenou aplikaci do modulu. Více informací lze získat v kapitolách 2.3.5 a 2.3.6.

Formát zdrojových souborů

Modul Programátor ukládá vytvořené projekty do souborů s příponou DFM. Jedná se o čistě textové soubory obsahující veškerou informaci potřebnou pro znovuotevření projektu pro další práci. Stejnou strukturu souboru pro ukládání obsahu a vlastností formuláře používá i vývojové prostředí C++ Builder a Delphi.

Nápověda programu

Nápovědu lze vyvolat kdekoliv v programu stisknutím klávesy F1 nebo volbou položky z menu. Soubor nápovědy je napsán v jazyce HTML a je uložen v podadresáři s názvem HELP.

Program dále využívá tzv. bublinových nápověd, které se objeví, pokud uživatel podrží kurzor myši nad některým ovládacím prvkem. Zároveň s bublinovou nápovědou se ve stavovém řádku aplikace objeví stručný popis dané funkce.

Základní informace o programu (verze programu, autor) lze vyvolat volbou položky *O programu* z menu *Nápověda*.

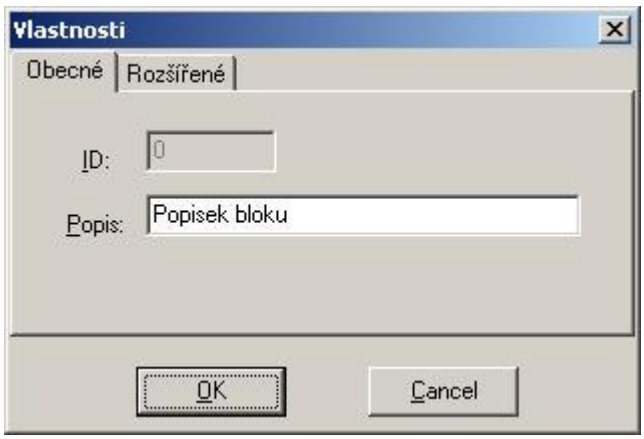
Nejdůležitější částí programu je knihovna funkčních bloků, a proto jí budeme věnovat samostatnou kapitolu.

2.3.3. Knihovna funkčních bloků

Knihovna funkčních bloků umístěná v levé části hlavního okna obsahuje seznam všech funkčních bloků, které mohou být využity pří návrhu aplikace. Funkční bloky jsou tématicky rozděleny do pěti skupin. Jsou to skupiny vstupů, výstupů, logických funkcí, speciálních funkcí a klopných obvodů.

Jednotlivé bloky mohou být přidány do aplikace tažením neboli metodou „Drag and Drop“. To znamená stisknutím a držením levého tlačítka myši nad příslušným blokem a přetažením do návrhového okna. Úpravy v návrhovém okně se provádějí stejným způsobem.

Každý z bloků má určité vlastnosti, které mohou být nastaveny v dialogovém okně (viz obr. 2.2) a které lze vyvolat buď poklepáním nebo označením příslušného bloku a následnou volbou položky *Vlastnosti* z menu *Úpravy*. Základní vlastností každého bloku je jeho jméno, které se objevuje při návrhu pod jeho grafickou značkou. Další společnou vlastností je ID. Toto jednobytové číslo je jednoznačný identifikátor, a proto jej nelze měnit. Některé bloky mají i další vlastnosti, které se nacházejí pod záložkou rozšířených vlastností tohoto dialogového okna.



obr. 2.2: Dialogové okno vlastností funkčního bloku

Každý z bloků je komponenta napsaná v jazyce C++ ve vývojovém prostředí C++ Builder. Kompletní stromová struktura tříd je uvedena na následující straně. Při vytváření nových bloků se vychází z nějakého existujícího bloku, od kterého se dědí jeho vlastnosti. Pro pokrytí celého spektra je zde vytvořeno několik abstraktních komponent, které vytvářejí základní třídu pro danou skupinu.

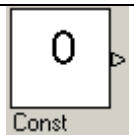
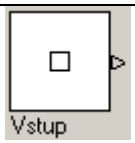
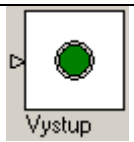
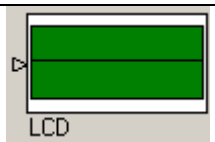
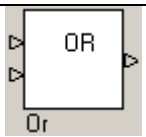
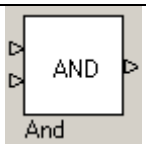
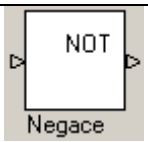
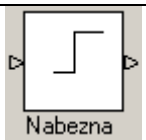
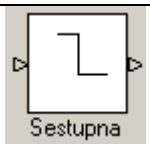
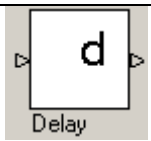
- *TSink* je rodičovská třída komponent definujících výstup. Potomek této třídy je například blok výstupu nebo LCD displej.
- *TSource* je rodičovská třída všech vstupů. Od této třídy je odvozen například blok LCD displeje.
- *TSoucastka* je základní abstraktní třída všech bloků.

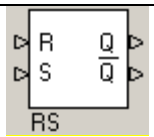
Kompletní stromová struktura tříd, které představují realizaci jednotlivých funkčních bloků.

TGraphControl - Základní třída pro všechny prvky

- TPin - Vstupní nebo výstupní brána uzlu
- TUzel - Uzel grafu
  - TSoucastka - Abstraktní třída všech funkčních bloků
    - TSource - Abstraktní třída všech vstupních bloků
      - TConst - Konstanta
      - TVstup - Logický vstup nebo tlačítko
    - TSink - Abstraktní třída všech výstupů
      - TLCD - Třída bloku LCD displeje
      - TVystup - Třída logického výstupu
    - TAnd - Třída logické konjunkce
    - TOr - Třída logické disjunkce
    - TNot - Třída logické negace
    - TRS - Třída RS klopného obvodu
    - TNabezna - Třída bloku náběžná hrana
    - TSestupna - Třída bloku sestupná hrana
    - TDelay - Třída bloku operátor zpoždění
  - TUzelHrana - Uzel spojující hrany
- THrana - Hrana

V následujícím odstavci podrobně popíšeme vlastnosti jednotlivých funkčních bloků, které se v současné verzi nacházejí.

	Konstanta. V rozšířených vlastnostech lze nastavit hodnotu. Výchozí hodnota je 0.
	Definuje vstup. V rozšířených vlastnostech je možné nastavit, zda se jedná o tlačítko nebo logický vstup. Dále je třeba nastavit pořadové číslo žádaného vstupu nebo tlačítka.
	Logický výstup. Opět je třeba ve vlastnostech nastavit žádané pořadové číslo výstupu, které se může pohybovat od hodnoty 0 do hodnoty 3.
	Ovládání LCD displeje. Do rozšířených vlastností se zadává text obou řádků. Případně je také možné zapnout podsvícení displeje.
	Logická disjunkce. Výstup je aktivní, pokud je alespoň na jednom vstupu log. 1.
	Logická konjunkce. Výstup je aktivní, pokud je na obou vstupech log. 1.
	Logická negace. Výstup je negací vstupu.
	Reaguje na náběžnou hranu vstupního signálu. Je aktivní pouze pro jeden programový cyklus, pokud se signál na vstupu změní z log. 0 na log. 1.
	Reaguje na sestupnou hranu vstupního signálu. Je aktivní pouze pro jeden programový cyklus, pokud se signál na vstupu změní z log. 1 na log 0.
	Operátor zpoždění o jeden krok. Na výstupu je hodnota, která byla na vstupu v minulém programovém cyklu.

	R-S klopný obvod. Pravdivostní tabulka převzatá z [8] viz tab. 2.6.

tab. 2.5: Popis jednotlivých funkčních bloků

S	R	Q	NOT Q
0	0	Q_{-1}	NOT Q_{-1}
0	1	0	1
1	0	1	0
1	1	0	0

tab. 2.6: Pravdivostní tabulka obvodu R-S

Množina všech funkčních bloků není zatím příliš veliká, ale postačující. Z výrokové logiky je známo, že pomocí negace, konjunkce a disjunkce je možné vyjádřit libovolnou logickou operaci [20].

Případné rozšíření množiny funkčních bloků je možné výše popsaným způsobem. Nově vytvořenou komponentu pak stačí umístit do knihovny funkcí. Nejjednodušší způsob je použití vizuálního programovacího nástroje C++ Builder.

2.3.4. Simulátor

Pro vyzkoušení správnosti návrhu uživatelské aplikace slouží simulátor. Pomocí tohoto nástroje je možné celou aplikaci vyzkoušet ještě před samotným nahráním do modulu. Názorná indikace stavu signálu v jednotlivých částech programu umožňuje uživateli rychleji odhalit chyby a hazardní stavy a celkově tak urychluje vývoj aplikace.

Simulátor se ovládá buď z přístrojové lišty tlačítky start a stop nebo z hlavního menu programu. Simulátor lze spustit pouze tehdy, je-li otevřen alespoň jeden projekt.

Při simulaci se používá zpětného šíření signálu logickou sítí. To znamená postup od výstupních bloků k blokům vstupním. Během postupu se průběžně vyhodnocují stavy jednotlivých bloků. Každý blok se během jednoho postupu vyhodnocuje pouze jednou. Při dalším volání v rámci jednoho programového cyklu se vrací již získaná hodnota. Metoda jednoho vyhodnocení bloku v rámci programového cyklu zjednodušuje výpočet a zabraňuje zacyklení programu.

Pro názornou orientaci se spoje v síti s pozitivní hodnotou signálu obarvují zeleně. Stav aplikace indikují také výstupní bloky.

Simulace probíhá cyklicky s periodou 500ms. Výpočet jednoho průběhu je umístěn do speciálního vlákna a nezatěžuje tak hlavní vlákno aplikace, které může přijímat uživatelské povely.

2.3.5. Kompilátor

Kompilátor překládá uživatelský program do formy, kterou je možné nahrát do modulu (viz kap. 2.3.6). Kompilace probíhá ve dvou krocích a v programu ji lze spustit z přístrojové lišty, z hlavního menu nebo klávesovou zkratkou F9.

1. Převod logické sítě do jazyka C.
2. Překlad programu z jazyka C a tvorba souboru ve formátu Intel hex.

Celý průběh kompilace je umístěn do samostatného vlákna a stejně jako v případě simulace nezatěžuje hlavní vlákno aplikace.

Převod logické sítě do jazyka C

Pro převod logické sítě do jazyka C se používá podobné metody jako v případě simulace (kap. 2.3.4). Opět se postupuje od výstupních bloků směrem k blokům vstupním. Během postupu přidá každý blok do výstupu svou část kódu. Kód každého bloku se přidá pouze jednou. Při dalším volání se přidá jen odkaz do paměti, kde je uložen výsledek daného bloku.

Každému bloku je v paměti procesoru vyhrazen jeden bit pro uložení výsledku. V současné verzi je alokováno pole o velikosti 32B v paměti RAM, což dovoluje použití maximálně 256 bloků. Případné rozšíření je možné zavedením dvoubytového identifikátoru bloku a umístěním pole do rozšířené paměti XRAM, kde je možné alokovat až 1024B paměti. Maximální počet použitelných součástek tak vzroste na 4096. Současně tak klesne rychlost výpočtu, protože přístup do rozšířené paměti XRAM je pomalejší než do paměti RAM.

V paměti je alokováno ještě jedno pole o stejné velikosti, které uchovává výsledky bloků v minulém programovém cyklu. Hodnoty z tohoto pole využívají některé speciální součástky, jako např. blok reagující na náběžnou a sestupnou hranu a blok zpoždění o jeden krok.

Kód získaný zpětným postupem sítí se spojí se vzorovým programem tak, že se kód zapíše do funkce *SetOutput()*. Úplný výpis vzorového programu s popisem je uveden v příloze (viz kap. B.13). Vzorový program obsahuje základní metodu *main*, ve které je implementován programový cyklus a alokace výše uvedených polí.

Prototypy všech funkcí použitých při sestavování uživatelského programu jsou uvedeny v systémové knihovně (viz kap. 2.2.4). Úplný výpis této knihovny je opět uveden v příloze (viz kap. B.5).

Překlad programu v jazyce C a tvorba souboru ve formátu Intel hex

Soubor získaný výše popsaným postupem se dále překládá C kompilátorem pro osmibitové mikroprocesory. Zde je konkrétně použit kompilátor SDCC (Small Device C Compiler). Tento nástroj lze volně stáhnout z internetu [18].

Kompilátor je k programu Modul Programátor připojen technologií PIPE. To znamená přesměrování standardního vstupu a výstupu. Průběh kompilace a případné chyby se objeví ve stavovém panelu hlavního okna aplikace.

Výsledkem kompilace je soubor ve formátu Intel hex, který může být nahrán do modulu (viz následující kapitola).

2.3.6. Loader

Loader slouží k nahrávání uživatelských programů do modulu. Programování modulu lze spustit buď z nástrojové lišty, z hlavního menu programu nebo klávesovou zkratkou F5. Podrobný popis programování modulu byl uveden v kapitole 1.3.8. Zde uvedeme programové řešení.

Podobně jako v případě kompilace se pro programování modulu používá externí program, který lze volně stáhnout z internetu [16]. I zde je použita technologie přesměrování standardního vstupu a výstupu do stavového panelu (PIPE).

Aby program bezchybně pracoval, je třeba správně nastavit číslo sériového portu COM, přes který je modul připojen k počítači. Toto číslo je možné zadat v základním nastavení Modul Programátoru.

Postup nahrávání uživatelské aplikace do modulu

Loader po svém spuštění nastaví signál PSEN na hodnotu LOW a čeká na resetování modulu. Resetujte modul a potvrďte dialogové okno, které vás informuje o spuštění procesu. Modul přejde do režimu programování. Od této chvíle probíhá kopírování dat do modulu. Tento proces můžete kdykoliv ukončit stisknutím tlačítka *Přerušit* z nástrojové lišty programu.

Po ukončení přenosu modul znovu resetujte. Programování je ukončeno a v modulu běží nová uživatelská aplikace.

2.3.7. Programový cyklus

Uživatelský program je v modulu vykonáván periodicky v tzv. programovém cyklu a jeho délka závisí na složitosti uživatelského programu. Programový cyklus se skládá z následujících kroků.

1. Inicializace.
2. Čtení hodnot vstupních signálů.
3. Čtení stavu tlačítek.
4. Provedení uživatelského kódu.
5. Zápis výstupu.

Inicializace

V této části dochází k inicializaci programového cyklu. Během inicializace se zkopíruje obsah pole nových výsledků do pole minulých výsledků bloků. Pole výsledků obsahuje výstupní hodnoty jednotlivých funkčních bloků (viz. kap. 2.3.5).

Čtení hodnot vstupních signálů

Přečtou se stavy digitálních vstupů. Přečtené hodnoty se uloží do pole *Input*, které má velikost osm bytů. Každý byte tohoto pole odpovídá jednomu vstupu.

Čtení stavu tlačítek

Přečtou se stavy tlačítek. Získané hodnoty se uloží do pole *Button*.

Provedení uživatelského kódu

Uživatelský kód je uložen ve funkci *SetOutput()*. Během provádění této funkce se podle stavu vstupů, tlačítek a minulých výsledků bloků nastavují výstupy. Výstupy se nezapisují přímo, ale ukládají se do pole *Output*.

Zápis výstupu

Pole *Output* modifikované v minulém kroku se pošle na fyzické výstupy.

2.3.8. Instalace programu Modul Programátor

Pro úplnou funkci programu je nejprve nutné stáhnout z internetu kompilátor jazyka C pro osmibitové mikropočítače a program pro programování procesoru AT89C51-RD2.

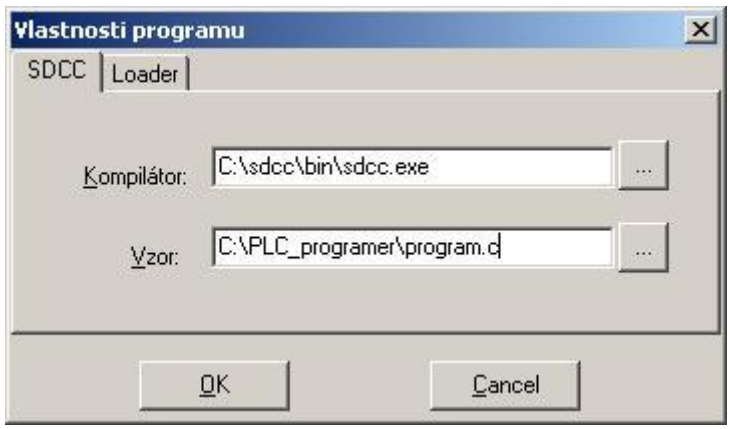
Jako kompilátor doporučuji použít SDCC (Small Device C Compiler), který lze volně stáhnout z internetu [18]. Na této adrese se prakticky denně objevují nové verze. Distribuce

SDCC je *opensource*. To znamená, že kromě samotného kompilátoru je možné stáhnout kompletní zdrojové kódy. Teoreticky je možné použít i jiný kompilátor, který splňuje normu ANSI-C. Správná funkce byla však ověřena pouze se jmenovaným kompilátorem.

Po instalaci SDDC kompilátoru je nutné nahrát knihovny *System*, *LCD* a *I2C*. To lze provést tak, že přeložené knihovny překopírujeme do adresáře „SDCC\LIB\SMALL“ a do textového souboru „libsdcc.lib“ dopíšeme na konec seznamu názvy výše zmíněných knihoven.

Další důležitou součástí je program pro kopírování dat do procesoru AT89C51-RD2 metodou ISP přes sériový port počítače. Z internetu lze volně stáhnout program RD2-FLASHER (dále jen RD2F), který je možné najít na stránkách HW-Serveru [16]. Jedná se o alternativní řešení originálního softwaru FLIP od firmy ATMEL [6]. Jeho výhodou je, že program je řešen jako konzolová aplikace ovládaná z příkazové řádky. Integrace do vývojového prostředí je pak snadná.

Základní vlastnosti Modul Programátoru se nastavují v dialogovém okně, jak je ukázáno na obr. 2.3 a obr. 2.4.



obr. 2.3: Nastavení SDCC kompilátoru. V editačním poli *Vzor* je cesta ke vzorovému programu, do kterého jsou vkládány uživatelské aplikace (viz kap. 2.3.5).

obr. 2.4: Nastavení programu RD2-FLASHER

Dialogové okno se skládá ze dvou záložek. První slouží k nastavení vlastností SDDC kompilátoru. Druhá obsahuje vlastnosti programu RD2F. Důležité je správně nastavit číslo sériového portu, na kterém je modul připojen.

2.3.9. Postup vytváření uživatelských aplikací

V této kapitole bude stručně popsán postup, jak vytvářet uživatelské programy pro modul pomocí grafického nástroje Modul Programátor.

Založení nového projektu provedeme volbou z hlavního menu *Soubor/Nový*. Otevře se nové prázdné dceřiné okno projektu. Projektů může být najednou otevřeno i více.

Do tohoto okna již můžeme začít vkládat funkční bloky. Vkládání nových bloků je řešeno metodou „Drag and Drop“ neboli tažením. Všechny funkční bloky použitelné pro návrh jsou umístěny v knihovně funkcí. Změna polohy funkčního bloku v rámci návrhového okna se opět provádí tažením. Pokud je funkční blok již zapojen do logické sítě, rozpojí se po přemístění všechny spoje vedoucí do daného bloku nebo z něj.

Pokud máme v návrhovém okně alespoň dva funkční bloky, můžeme začít vytvářet funkční spojení. Nové funkční spojení se vytváří opět metodou „Drag and Drop“. Tažením pinu funkčního bloku vytvoříme pravoúhlou kombinaci dvou hran. V návrhu spoje pokračujeme tak, že opět táhneme za konec předchozí hrany, který je označen malým černým čtvercem. Takto postupujeme, dokud se v návrhu nedostaneme ke druhému funkčnímu bloku, kde napojením na odpovídající pin ukončíme návrh spoje.

Pokud chceme provést změnu návrhu, můžeme celý spoj nebo jeho část smazat. Mazání hrany se provádí tak, že myší klikneme na příslušnou hranu. Označená hrana se obarví červeně a stisknutím klávesy *delete* se označená část smaže. Pokud chceme smazat více hran najednou, postupujeme stejným způsobem, přičemž musí být stále stisknuta klávesa Ctrl.

Vytvořený projekt můžeme uložit na disk počítače. Projekt je na disku uložen v souboru s příponou DFM. Tento soubor obsahuje veškeré potřebné informace pro znovuotevření projektu a pokračování v práci.

Jestliže máme vytvořený projekt, můžeme jeho funkci ověřit v simulátoru. Význam a funkce simulátoru byly vysvětleny v kapitole 2.3.4.

Vytvořený a odzkoušený projekt je připraven k nahrání do modulu. Nejprve je třeba celý projekt přeložit. Překlad lze nejsnáze spustit klávesovou zkratkou F9. Průběh překladu je možné sledovat ve stavovém panelu. Během překladu se v tomto panelu zobrazí výsledný program v jazyce C.

V tomto stavu již můžeme projekt nahrát do modulu. Proces nahrávání je možné spustit klávesou F5. Na začátku procesu se objeví výzva k resetování modulu. Jakmile je modul resetován a dialogové okno s výzvou potvrzeno, začíná vlastní přenos dat do modulu. Po úspěšném přenosu dat je nutné modul znovu resetovat. Od tohoto okamžiku běží v modulu nová aplikace.

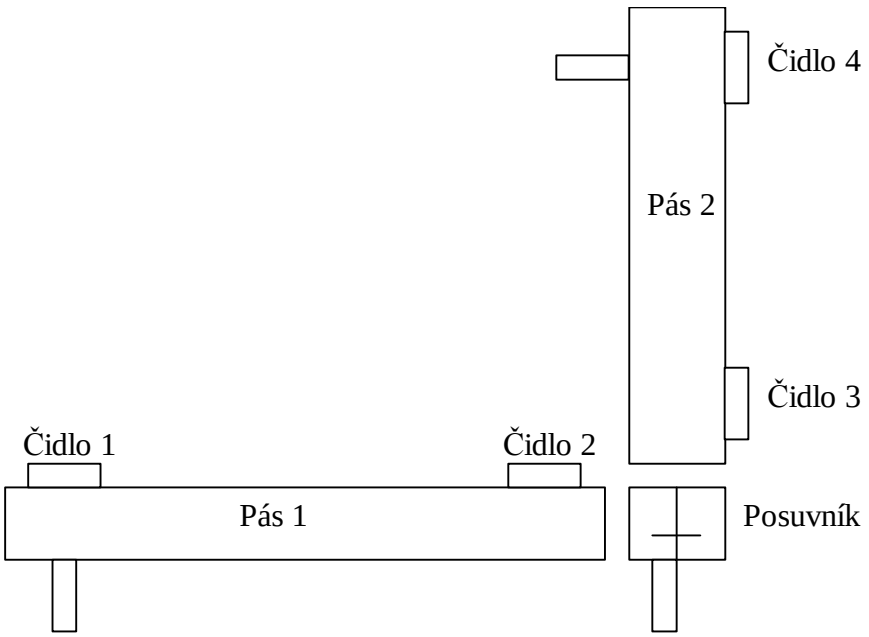
3. Použití pro řízení

Navržený modul je určen pro malé aplikace, jako je například řízení dopravníkových pásů (jak bude uvedeno dále), řízení osvětlení v budovách a podobně. Modul byl navržen jako alternativa k modulu LOGO! od firmy Siemens.

3.1. Příklad použití – řízení pásového přepravníku

3.1.1. Popis systému

Jako příklad aplikace zde uvádím řízení dopravníkových pásů. Struktura řízeného systému je znázorněna na obr. 3.1.



obr. 3.1: Schéma řízeného systému

Systém se skládá ze dvou přepravníků a jednoho posuvníku. Každý z přepravníků je vybaven dvěma čidly, která detekují objekt na začátku a na konci pásu. Posuvník obsahuje také dvě čidla ve formě koncových spínačů, které informují o krajní poloze posuvníku.

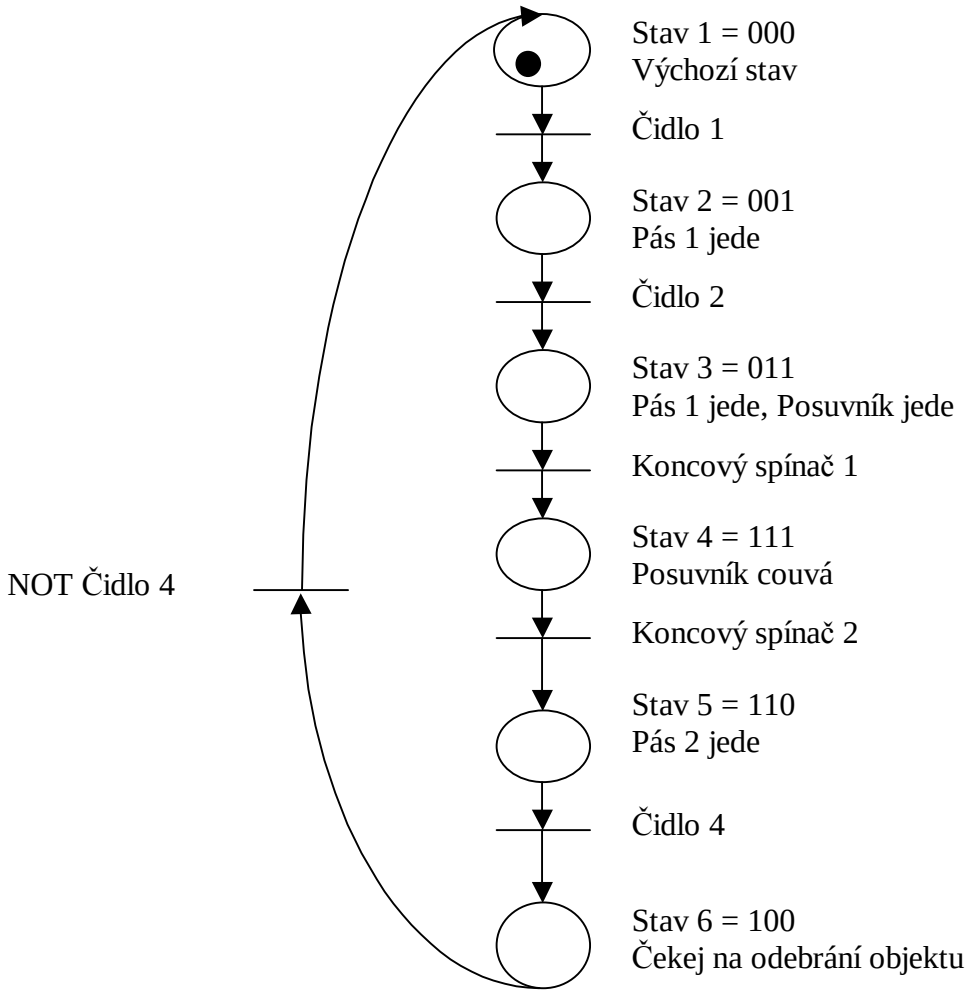
Vstupem do systému je ovládání motorů přepravníku 1 a 2, ovládání motoru posuvníku a signál ovlivňující směr posunu posuvníku.

3.1.2. Úkol řízení

Cílem řízení je dopravit objekt vložený na začátek přepravníku 1 na konec přepravníku 2. Systém je řešen jako dávkový proces. To znamená, že další objekt může vstoupit do systému až po odebrání předešlého objektu.

3.1.3. Řešení

Řešení lze názorně popsat pomocí Petriho sítě (viz obr. 3.2).



obr. 3.2: Petriho síť

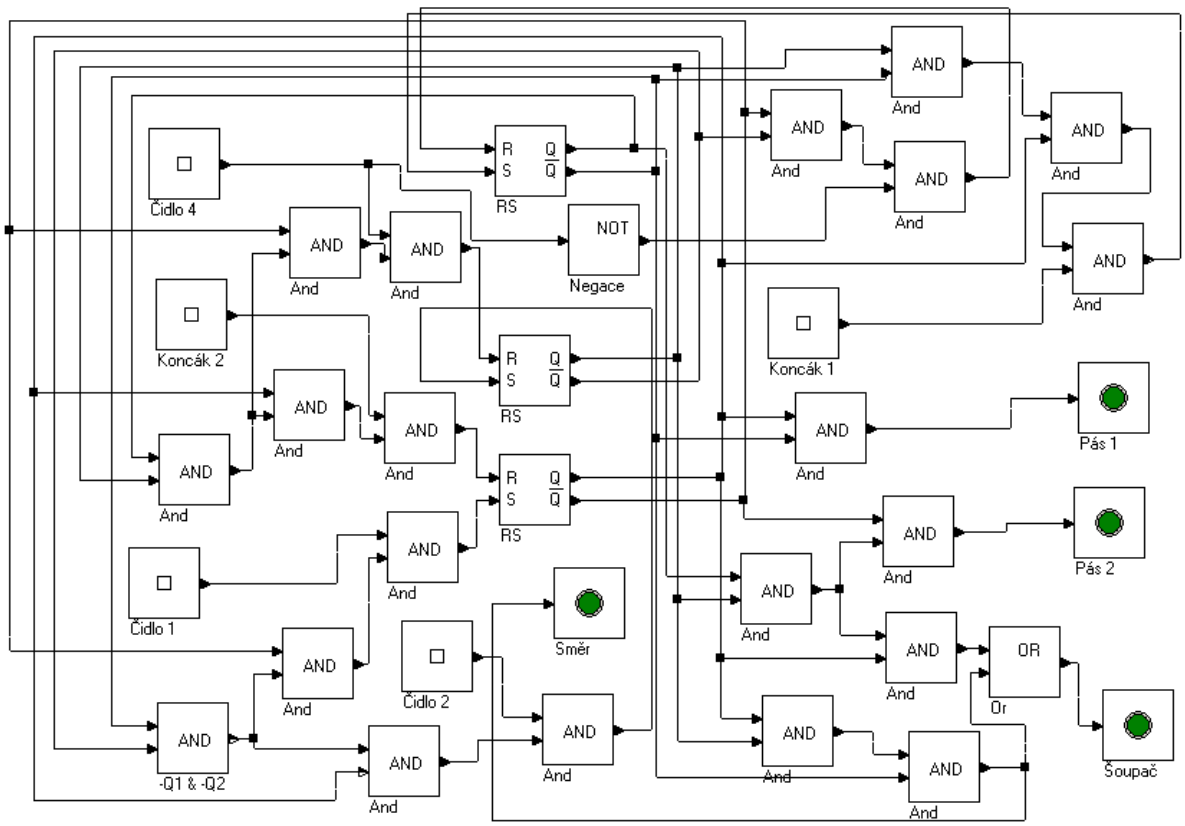
Jedná se o sekvenční logický systém se šesti stavy. Pro řešení budeme muset použít klopné obvody pro uložení stavu [8]. Stavy systému můžeme zakódovat například následujícím způsobem.

	q_1	q_2	q_3
Stav 1	0	0	0
Stav 2	0	0	**1**
Stav 3	0	**1**	1
Stav 4	**1**	1	1
Stav 5	1	1	**0**
Stav 6	1	**0**	0

tab. 3.1: Zakódování stavů

Při kódování musíme dodržet podmínku, aby při přechodu z jednoho stavu do druhého docházelo ke změně pouze v jedné stavové proměnné. Dodržením této podmínky eliminujeme případné hazardní stavy [8].

Po zakódování stavů již stačí navrhnout vstupní a výstupní logickou síť. Celé řešení je znázorněno na obr. 3.3. Vstupní logická síť nastavuje na základě hodnot vstupů a stavů nové stavy automatu. Výstupní logická síť nastavuje výstupní signály na základě stavu automatu.



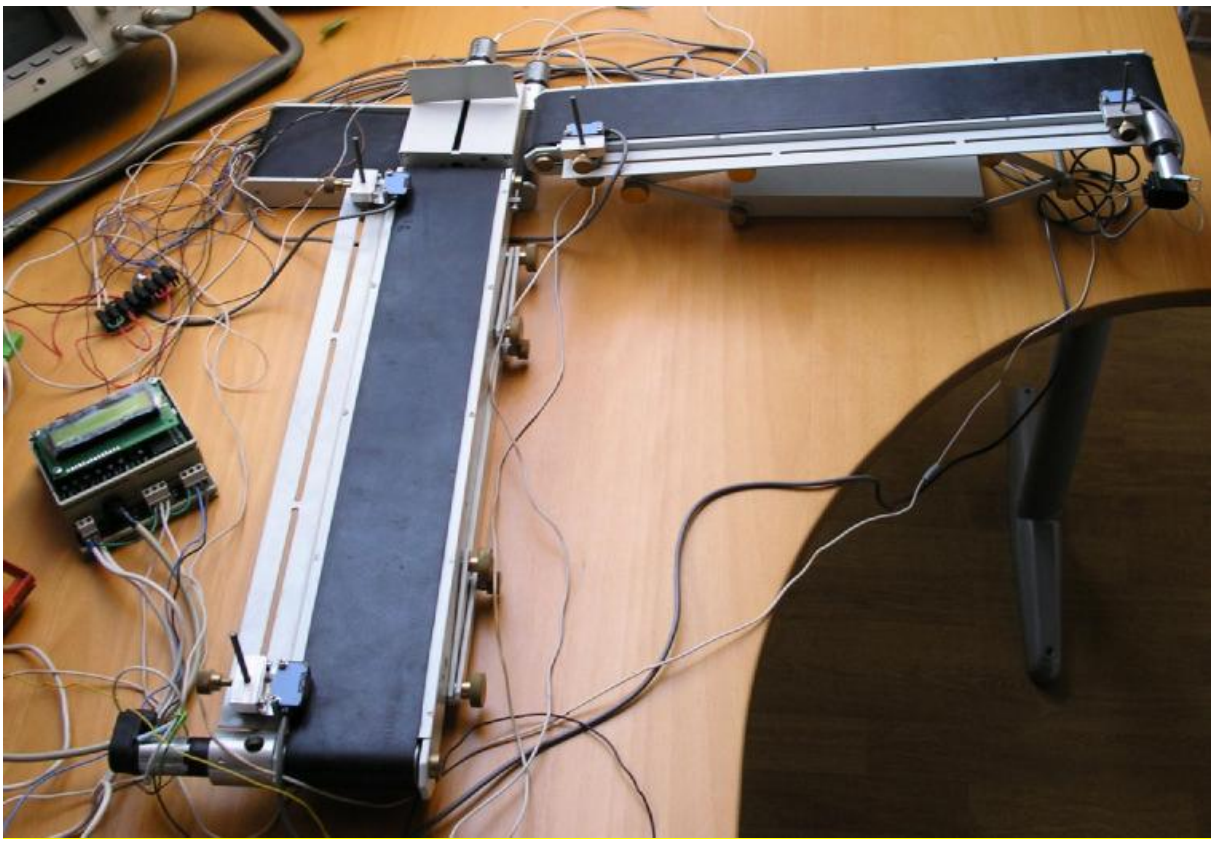
obr. 3.3: Řešení řízení přepravníků

Pro zajímavost uvádím také výpis převodu výše uvedené logické sítě do jazyka C.

```
void SetOutput( void )
{
        AND(16, RSnon(2,GetNewValue(11),GetNewValue(7)), GetNewValue(8));
        AND(17, !Input[7], GetNewValue(16));
        NOT(19,!Input[7]);
        AND(20, !GetNewValue(2), !GetNewValue(1));
        AND(21, GetNewValue(20), GetNewValue(19));
        AND(8, RS(0,GetNewValue(21),GetNewValue(25)),
                RS(1,GetNewValue(17),GetNewValue(15)));
        AND(10, GetNewValue(2), GetNewValue(8));
        AND(11, !Input[2], GetNewValue(10));
        AND(13, GetNewValue(3), RS(2,GetNewValue(11),GetNewValue(7)));
        AND(15, !Input[5], GetNewValue(13));
        AND(3, !GetNewValue(0), RSnon(1,GetNewValue(17),GetNewValue(15)));
        AND(4, !GetNewValue(2), GetNewValue(3));
        AND(7, !Input[4], GetNewValue(4));
        AND(22, GetNewValue(1), !GetNewValue(0));
        AND(23, GetNewValue(22), RS(2,GetNewValue(11),GetNewValue(7)));
        AND(25, GetNewValue(23), !Input[3]);
        AND(27, GetNewValue(2), RSnon(0,GetNewValue(21),GetNewValue(25)));
        Output[0] = GetNewValue(27);
        AND(31, GetNewValue(0), GetNewValue(1));
        AND(32, !GetNewValue(2), GetNewValue(31));
        Output[1] = GetNewValue(32);
        AND(34, GetNewValue(2), GetNewValue(1));
        AND(35, GetNewValue(34), !GetNewValue(0));
        AND(33, GetNewValue(31), GetNewValue(2));
        OR(37, GetNewValue(33), GetNewValue(35));
        Output[3] = GetNewValue(37);
        Output[2] = GetNewValue(35);
}
```

3.1.4. Ověření funkce a závěr

Řešení bylo ověřeno na reálné soustavě. Reálná soustava se skládala z modelů přepravníků a posuvníku (viz obr. 3.4). Výsledné řešení splňovalo požadavky uvedené v zadání.



obr. 3.4: Ověření funkce na reálné soustavě

Na přiloženém CD je také možné najít krátké video ukazující funkci celého zařízení. Soubor s videem je kódován algoritmem MPEG1.

4. Podobná řešení

Podobné řešení programovatelného řídicího modulu nabízí firma Siemens. Systém se jmenuje LOGO! a lze ho najít například na internetu [17]. Systém je podle výrobce určen především pro menší aplikace, jako je například řízení osvětlení nebo vytápění budov.

Základní jednotka systému LOGO! má podobné vlastnosti jako výše popsaný modul.

- Svorky pro napájení 24V DC.
- 8 digitálních vstupů.
- 4 digitální výstupy.
- Rozhraní pro programový modul nebo kabel do PC.
- Podsvětlený LCD displej 4 řádky x 12 sloupců.
- Tlačítkové ovládací pole.

K základní jednotce lze připojit rozšiřující moduly. Velkou nevýhodou systému LOGO! je omezení maximálního počtu digitálních vstupů a výstupů. Umožňuje maximálně 24 digitálních vstupů, 16 digitálních výstupů a 8 analogových vstupů. LOGO! nemá žádné analogové výstupy. Modul popsaný v této práci žádná omezení na rozšiřující moduly neklade.

Návrh uživatelských programů pro LOGO! je založen na podobném principu. Jedná se rovněž o grafický návrhový systém založený na metodě „Drag and Drop“. Firma Siemens jej nabízí za cenu 1500,- Kč. Program zde popsaný nemůže v současné verzi konkurovat komerčnímu programu od firmy Siemens. Autor se na tomto místě odkazuje na budoucí práce, které by se mohly zabývat návrhem dalších funkčních bloků a optimalizací uživatelského rozhraní.

Cena základní jednotky LOGO! se pohybuje kolem 3000,- Kč. Cena rozšiřujících modulů je okolo 1500,-. Tyto ceny jsou přibližně srovnatelné s řešením popsaným v této práci.

5. Závěr

Cílem této diplomové práce byl návrh a realizace programovatelného řídicího modulu, který by obsahoval osm digitálních vstupů, čtyři digitální výstupy a rozhraní CAN. Součástí práce byl i vývoj grafického návrhového systému, který by jednoduchým způsobem umožňoval návrh uživatelských aplikací pro tento modul.

Funkčnost celého řešení byla ověřena na modelu, který se skládal ze dvou přepravníků a jednoho posuvníku. Cílem řízení bylo přemístit objekt umístěný na začátek jednoho pásu na konec pásu druhého. Řídicí program pro tento systém byl navržen v uvedeném návrhovém programu. Celý průběh procesu řízení je možné vidět na videu, které je obsaženo na přiloženém CD. Ukázalo se, že návrhový systém skutečně ulehčuje návrh uživatelských aplikací a modul spolehlivě řídí danou soustavu. Velkou výhodou je možnost simulace funkčnosti aplikace před samotným nahráním do modulu.

Již ze zadání plyne, že tato práce je prací úvodní. Další práce zabývající se tímto tématem by měly dále rozpracovat možnost napojení dalších rozšiřujících modulů (další logické vstupy a výstupy atd.). V této práci je popsáno rozhraní pro sběrnici CAN, které by do budoucna mohlo sloužit právě pro napojení těchto rozšiřujících modulů.

Grafický návrhový systém nabízí v současné verzi jen základní soubor funkčních bloků. Rozšíření je možné postupem, který byl popsán v kapitole 2.3.3. Také uživatelské prostředí obsahuje několik drobných chyb, které je možné odstranit až při delším používání programu v praxi.

Reference

[1] *Součástky pro elektrotechniku.* Praha: GM electronic, 2003.392 s.
[2] Micrel Semiconductor. *Micrel Semiconductor* [online]. [cit. 2003-12-13] <http://www.micrel.com>.
[3] Hypel. *Hypel* [online]. [cit. 2003-12-13] <http://www.hypel.cz>.
[4] Philips Semiconductors. *Philips Semiconductors* [online]. [cit. 2003-12-13] <http://www.semiconductors.philips.com>.
[5] International Rectifier. *Internation Rectifier* [online]. [cit. 2003-12-13] <http://www.irf.com>.
[6] Atmel. *Atmel* [online]. [cit. 2003-12-13] <http://www.atmel.com>.
[7] CAN in Automation (CiA). *CAN in Automation* [online]. Poslední revize 2003-11-21 [cit. 2003-12-13] . <http://www.can-cia.de>.
[8] Bayer, J. *Logické řídicí systémy – Sylaby přednášek.* Praha: 1992. 206 s
[9] Podlešák, J. *Přístrojové aplikace mikroprocesorů*. Praha: Vydavatelství ČVUT, 1999. 164 s.
[10] Šubrt, V. *Jednočipové mikropočítače Intel 8048 – 8096*. Praha: Grada, 1992. ISBN 80-85424-66-5.
[11] Cantu, M. *Delphi 4*. Praha: Grada, 1999. ISBN 80-7169-800-8.
[12] Šnorek, M. *Standarní rozhraní PC*. Praha: Grada, 1992. ISBN 80-85424-80-0.
[13] Versick, D. *Linux-Gerätetreiber für CAN-Controller*. Diplomová práce. Rostock: Lehrstuhl für Informations- und Kommunikationsdienste, Universität Rostock, 2001. 34 s.
[14] Nenadál, K. Václavíková, D. *Turbo C – popis jazyka*. Praha: Grada, 1991. ISBN 80-85424-08-8.
[15] Hlavenka, J. Sedlář, R. Holčík, T. Kubeš, J. Mach, J. *Vytváříme WWW stránky*. Praha: Computer Press, 2000. 520 s.
[16] HW server. *HW server* [online]. [cit. 2003-12-13] <http://www.hw.cz>.
[17] Siemens, *Mikrosystémy Siemens* [online]. [cit. 2003-12-13]. <http://www.siemens.cz/micro>.
[18] Helton, D. *SDCC – Small Device C Compiler* [online]. Poslední revize 2003-05-30. [cit. 2003-12-13]. <http://sdcc.sourceforge.net>.
[19] Data Image Corporation. Data Image Corporation [online]. [cit. 2003-12-13]. <http://www.dataimagecorp.com>.
[20] Demlová, M. Pondělíček, B. *Matematická logika*. Praha: Vydavatelství ČVUT, 1997. 160 s.

Příloha

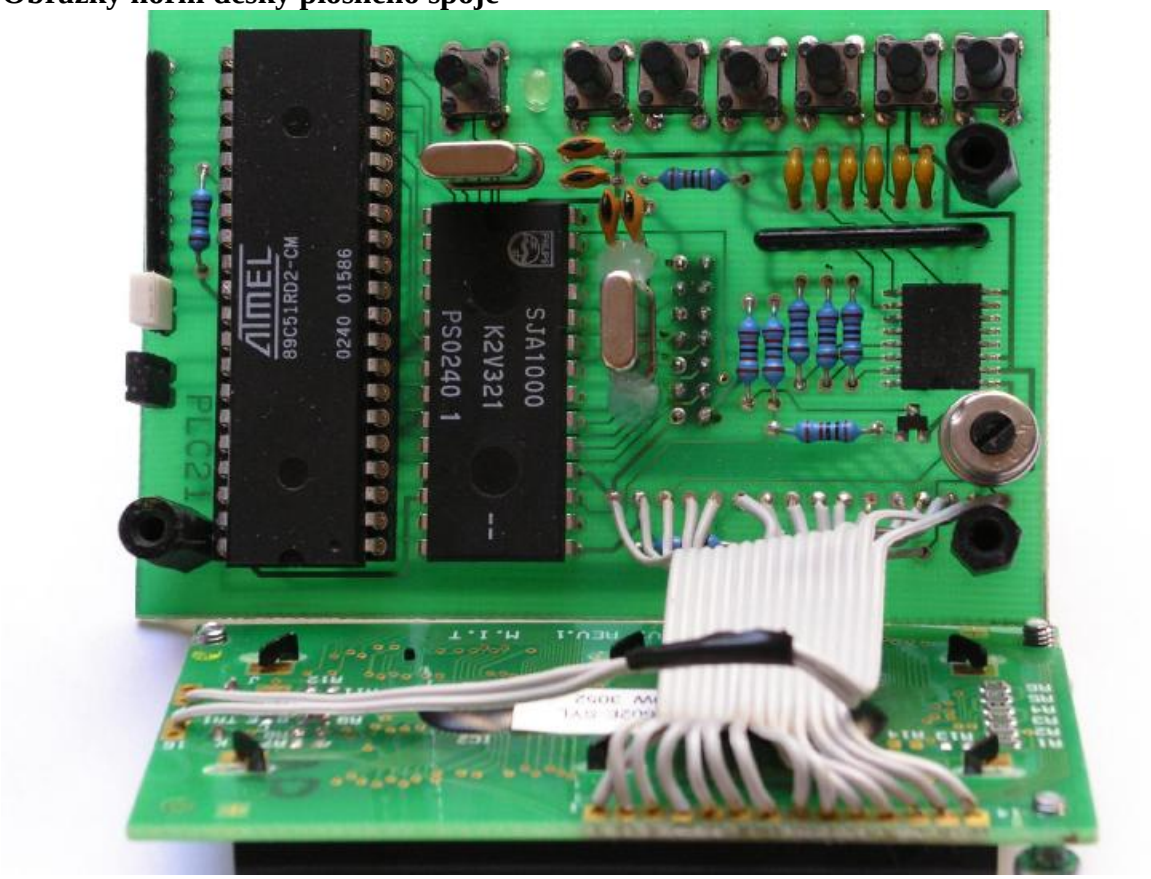
A. Hardware

A.1. Horní deska

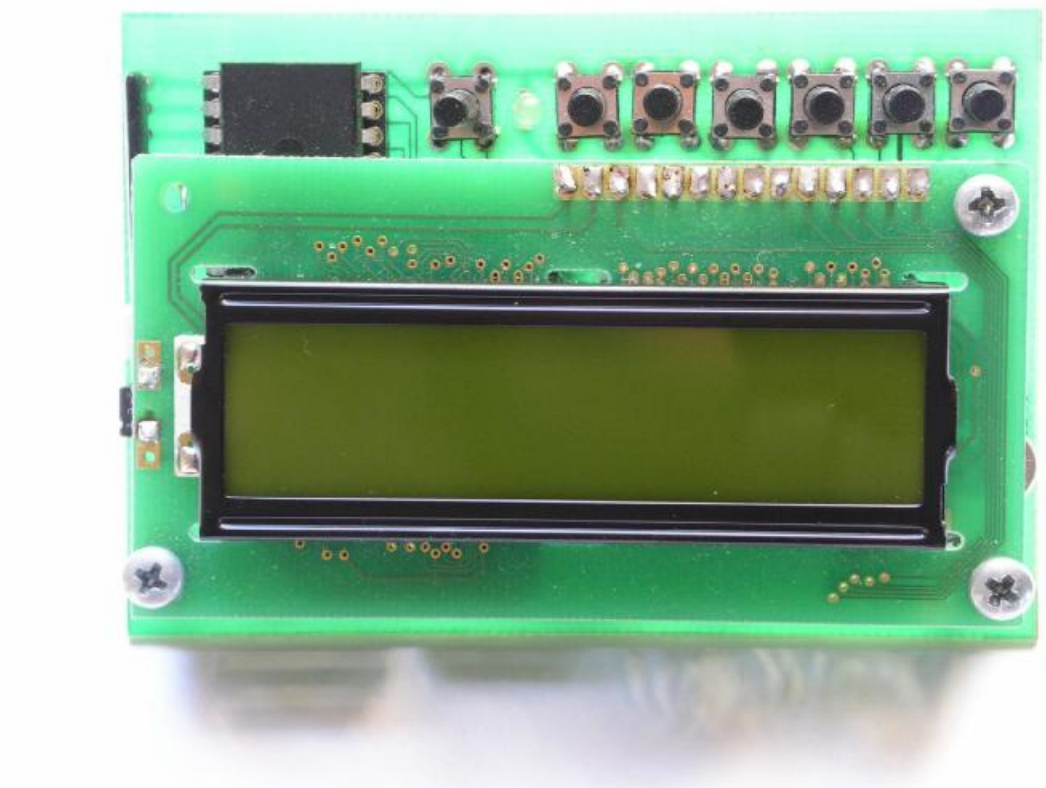
Seznam součástek

Položka	Počet	Reference	Hodnota
1	2	C25, C26	15pF
2	2	C27, C28	22pF
3	6	C29, C30, C31, C32, C33, C34	100nF
4	1	D18	L-HLMP-1790
5	1	JP10	Prázdné
6	1	JP11	ASS21438Z
7	2	Q1, Q2	BC847AL
8	3	R20, R28, R33	10k
9	2	R22, R26	RR 8x 2k7 – A
10	4	R23, R24, R25, R29	2k7
11	2	R31, R27	2k2
12	1	Patice	DIL40PZ
13	1	R30	1k5
14	7	SW8, SW9, SW11, SW12, SW13, SW14, SW15	P-B1720D
15	1	U18	AT89C51-RD2
16	1	U19	SJA1000
17	1	U20	PCF8574T-SMD
18	1	Y1	24MHz
19	1	Y2	18.432MHz

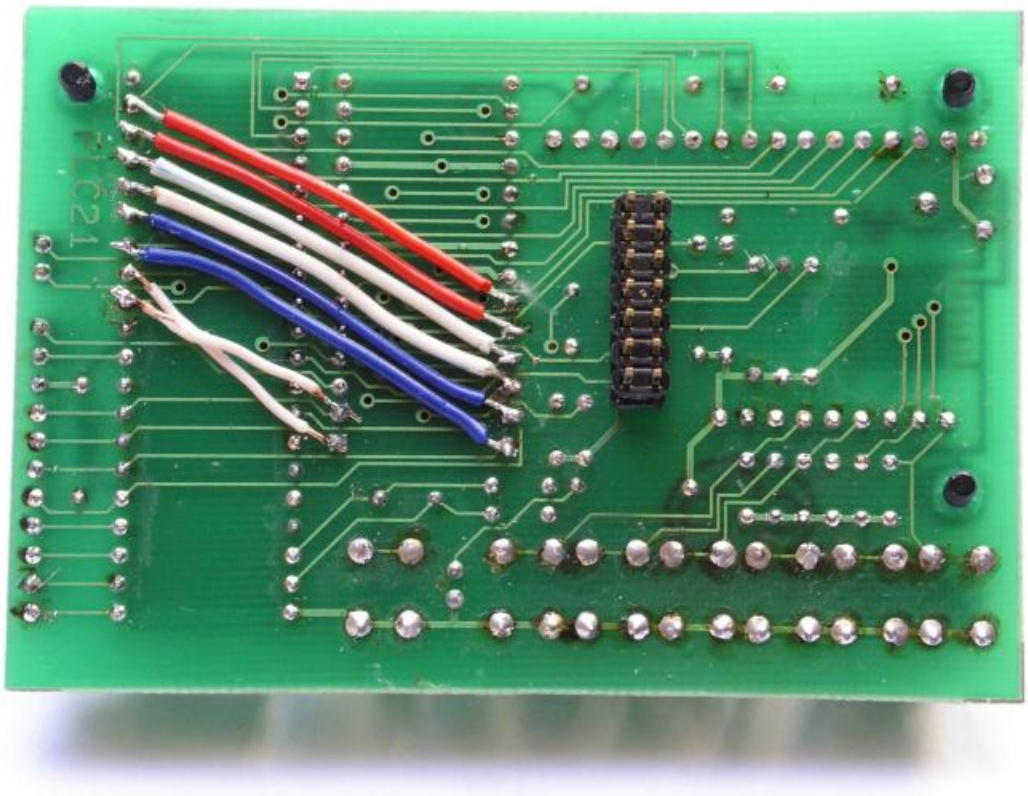
Obrázky horní desky plošného spoje



obrázek 1: Pohled na horní desku ze strany součástek s odklopeným LCD displejem



obrázek 2: Pohled na horní desku ze strany součástek s namontovaným LCD displejem



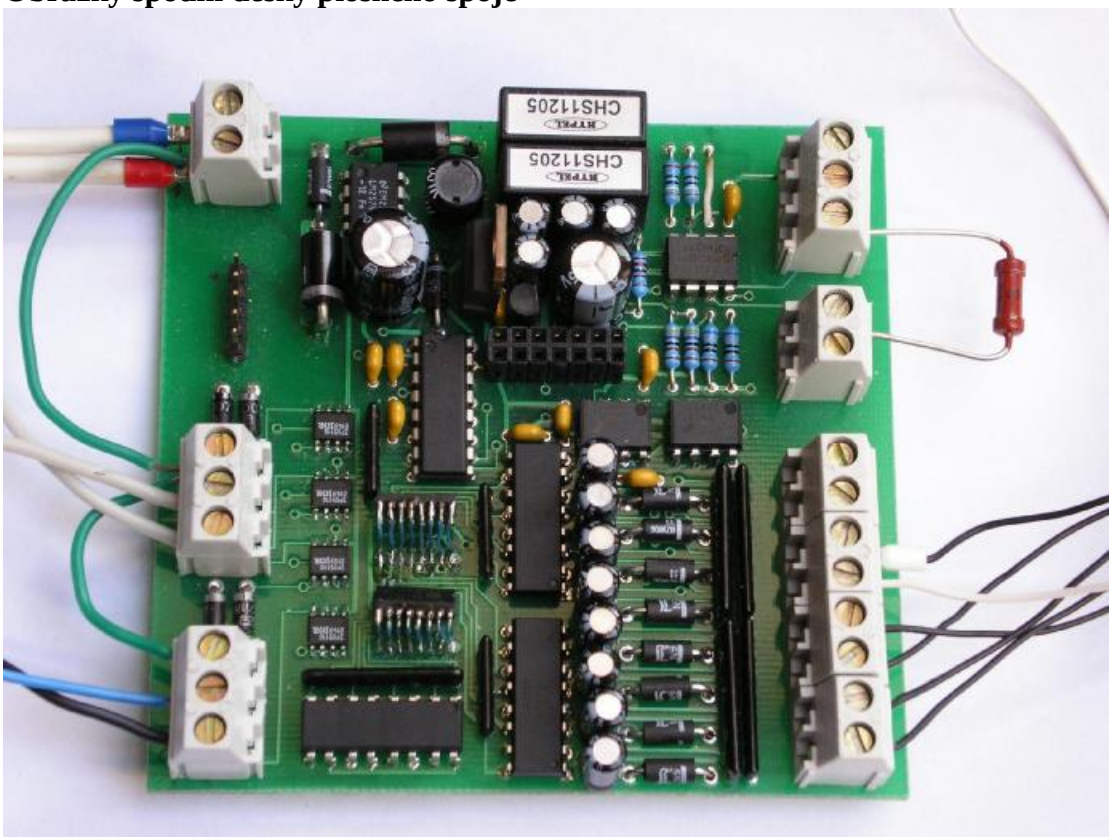
obrázek 3: Pohled na horní desku ze strany spojů

A.2. Spodní deska

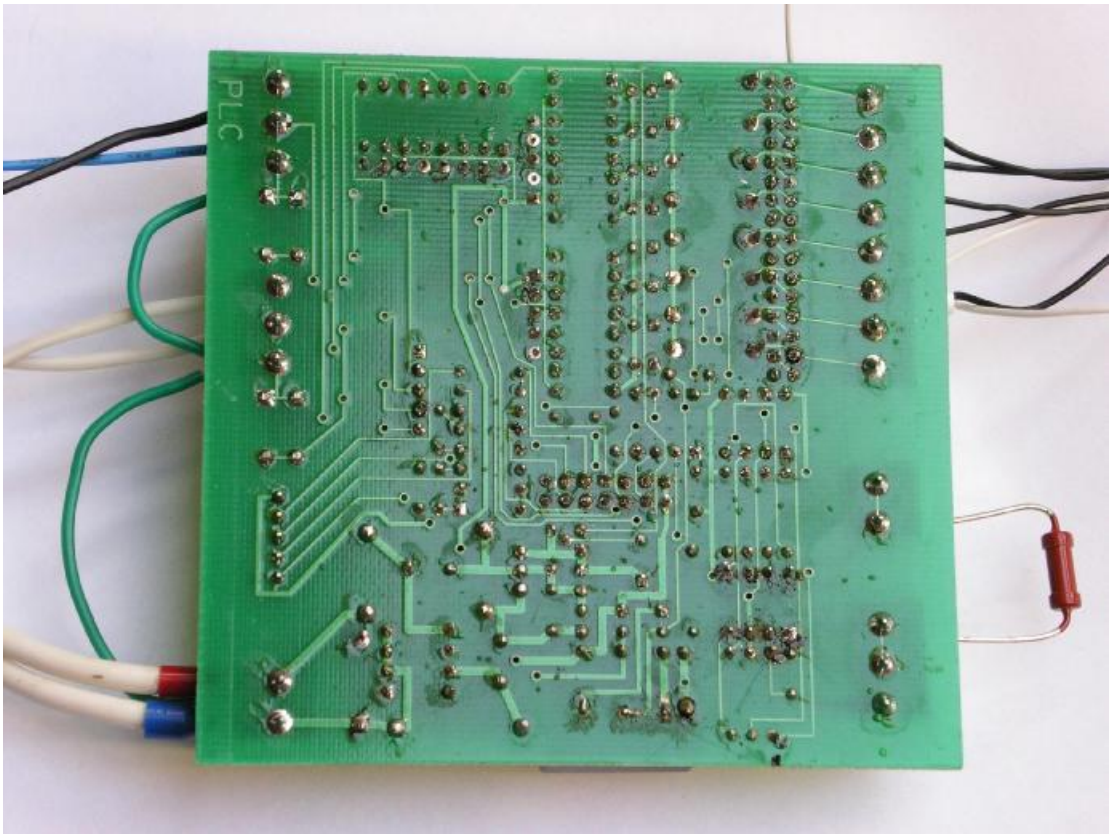
Seznam součástek

Položka	Počet	Reference	Hodnota
1	4	D9, D10, D11, D12	1N4007
2	1	D13	1N5408
3	1	D15	BZW06-15
4	1	D16	SB360
5	1	JP1, JP6, JP7	ARK120/3
6	1	JP2, JP8	ARK120/2
7	1	JP3	4xARK120/2
8	1	JP4	BL214G
9	1	JP9	SJG20
10	1	L1	09P-331K
11	1	Q1	BC327
12	2	R1, R2	6k8
13	4	R3, R4, R12, R13	RR4x2k7-B
14	2	R5, R9	390R
15	2	R6, R7	680R
16	2	R8, R11	4k7
17	1	R10	0
18	1	R14	2k7
19	4	R15, R16, R17, R18	1k
20	1	R19	12k
21	2	U10, U1	PCF8574-SMD
22	3	U2, U6, U9	PC847
23	2	U3, U5	6N137
24	1	U4	PCA82C250
25	1	U7	MAX232
26	4	U8, U11, U12, U13	IPS511G
27	2	U15, U14	CHS11205
28	1	U16	7805
29	1	U17	LM2574-12
30	9	C1, C2, C3, C4, C8, C9, C10, C11, C21	10M/35V
31	6	C5, C6, C7, C18, C23, C24	M1
32	4	C12, C13, C14, C15	100n
33	3	C16, C17, C22	47M/25V
34	1	C19	470M/35V
35	1	C20	470M/25V
36	9	D1, D2, D3, D4, D5, D6, D7, D8, D14	BZW06-33

Obrázky spodní desky plošného spoje

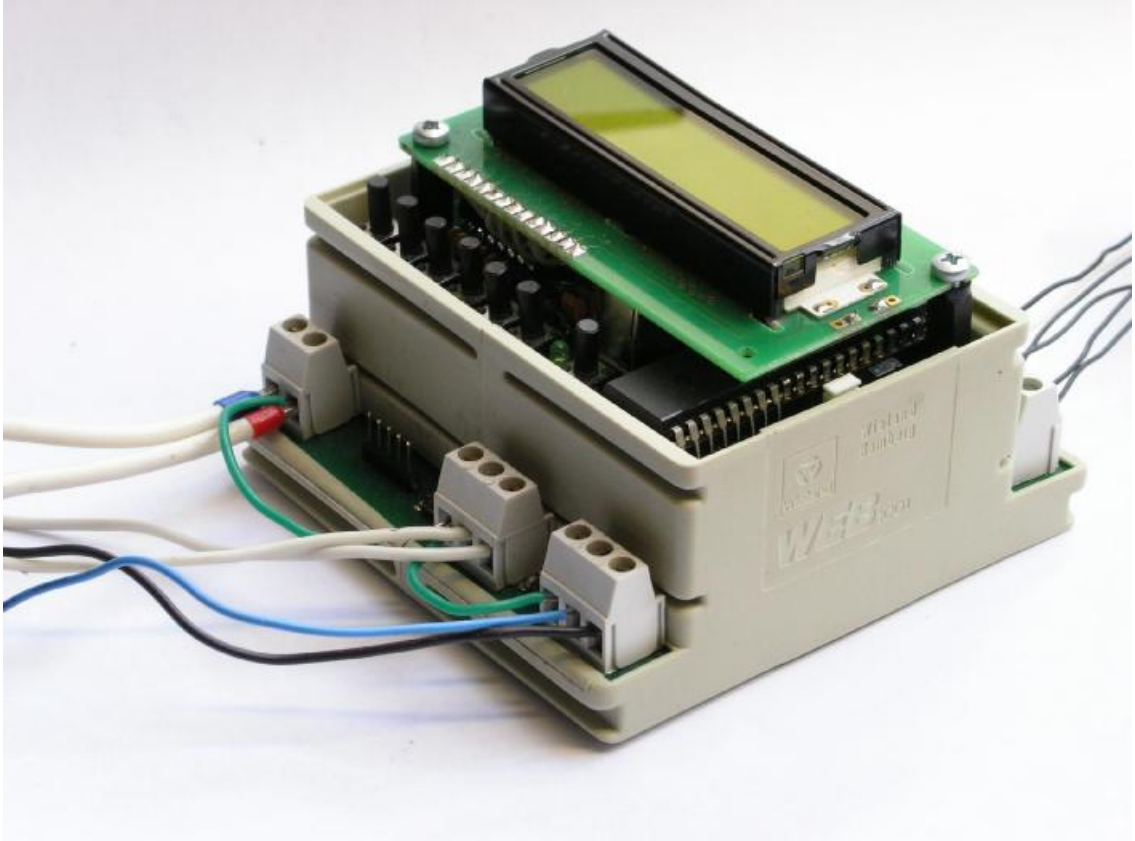


obrázek 4: Pohled na spodní desku ze strany součástek

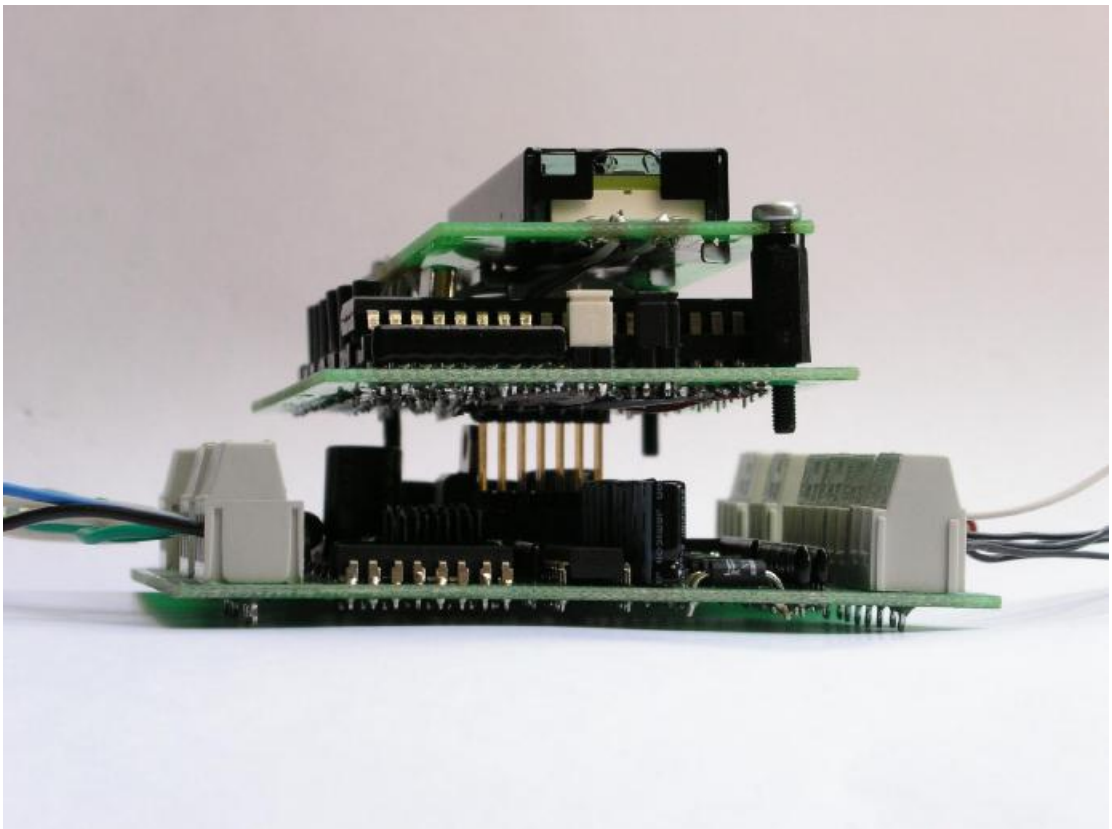


obrázek 5: Pohled na spodní desku ze strany spojů

A.3. Obrázky modulu



obrázek 6: Pohled na výsledný modul



obrázek 7: Pohled na spojení obou desek

B. Software

B.1. i2c.h

```
/*                                                    */
/* i2c.h 1.0.0                (c) Evzen Thoendel      */
/*                              14.8. 2003            */
/*                                                    */
/* Knihovna pro obsluhu I2C sbernice                  */
/*                                                    */
/*                                                    */
/*                                                    */

#ifndef I2C_H
#define I2C_H

//Adresy jednotlivych modulu
#define ADDR_TLACITKA 66
#define ADDR_VSTUPY 116
#define ADDR_VYSTUPY 112

unsigned char i2c_readbyte( unsigned char addr );
//Precte byte ze zadane adresy

unsigned char i2c_sendbyte( unsigned char addr, unsigned char value );
//Posle byte na zadanou adresu. Vraci 1 pokud je operace uspesna.

#endif
```

B.2. i2c.c

```
/*                                                    */
/* i2c.c 1.0.0                (c) Evzen Thoendel      */
/*                                14.8. 2003          */
/*                                                    */
/* Knihovna pro obsluhu I2C sbernice                  */
/*                                                    */
/*                                                    */
/*                                                    */


#include "i2c.h"

sbit at 0x94 SCL;
sbit at 0x95 SDA;

unsigned char i2c_sendbyte( unsigned char addr, unsigned char value )
//Posle byte na zadanou adresu. Vraci 1 pokud je operace uspesna.
{
   unsigned char i;

   //posli startbit
   SDA = 1;
   SCL = 1;
   SDA = 0;
   SCL = 0;

   //posli adresu
   for (i=7; i>0; i--) {
      SDA = (addr & (1 << i)) == (1 << i);
      SCL = 1;
      SCL = 0;
   }

   //Posledni bit oznamujici zapis
   SDA = 0;
   SCL = 1;
   SCL = 0;

   //Precti ACK
   SDA = 1;
   SCL = 1;
   i = SDA;
   SCL = 0;
   if (i) return 0;

   //posli data
   for (i=0; i<8; i++) {
      SDA = (value & 0x80) == 0x80;
      value = value << 1;
```

```
        SCL = 1;
        SCL = 0;
    }

    //Precti ACK
    SDA = 1;
    SCL = 1;
    i = SDA;
    SCL = 0;

    //posli stopbit
    SDA = 0;
    SCL = 1;
    SDA = 1;
    SCL = 0;

    return !i;
}

unsigned char i2c_readbyte( unsigned char addr )
//Precte byte ze zadane adresy
{
    unsigned char i;
    unsigned char value=0;

    //posli startbit
    SDA = 1;
    SCL = 1;
    SDA = 0;
    SCL = 0;

    //posli adresu
    for (i=7; i>0; i--) {
        SDA = (addr & (1 << i)) == (1 << i);
        SCL = 1;
        SCL = 0;
    }

    //Posledni bit oznamujici cteni
    SDA = 1;
    SCL = 1;
    SCL = 0;

    //Precti ACK
    SDA = 1;
    SCL = 1;
    i = SDA;
    SCL = 0;

    //Precti data
```

```
    for (i=0; i<8; i++) {
        SDA = 1;
        SCL = 1;
        value = (value << 1) | SDA;
        SCL = 0;
    }

    //NO ACK
    SDA = 1;
    SCL = 1;
    SCL = 0;

    //posli stopbit
    SDA = 0;
    SCL = 1;
    SDA = 1;
    SCL = 0;

    return value;
}
```

B.3. lcd.h

```
/*                                                    */
/* lcd.h 1.0.0                (c) Evzen Thoendel      */
/*                                13.8. 2003          */
/*                                                    */
/* Knihovna pro obsluhu LCD displeje                  */
/*s 8-mi bitovym pripojenim                           */
/*                                                    */
/*                                                    */

#ifndef LCD_H
#define LCD_H

#define CR_SHOW 2
#define CR_BLINK 1

void lcd_init( unsigned char setup );
// Inicializace displaye.
// setup ... nastaveni vlastnosti cursoru. (CR_SHOW | CR_BLINK)

void lcd_close();
// Vypne display

void lcd_putch( char znak );
// Napise znak

void lcd_write( char *str );
// Napise retezec znaku zakonceny \0

void lcd_clr();
// Vymaze display a presune cursor na zacatek

void lcd_ledon();
// Zapne podsviceni

void lcd_ledoff();
// Vypne podsviceni

void lcd_at( unsigned char x, unsigned char y );
// Presune cursor na zadanou pozici

#endif
```

B.4. lcd.c

```
/*                                                    */
/* lcd.c 1.0.0                (c) Evzen Thoendel      */
/*                                13.8. 2003          */
/*                                                    */
/* Knihovna pro obsluhu LCD displeje                  */
/*s 8-mi bitovym pripojenim                           */
/*                                                    */
/*                                                    */

#include "lcd.h"

#define TICK2US 10
#define DELAYTIME 1000

sfr at 0x80 DB;
sbit at 0x90 RS;
sbit at 0x91 RW;
sbit at 0x92 E;
sbit at 0x93 LED;

void lcd_delay( int us )
//Ceka zadany pocet milisekund. Neni pesny. Da se nastavit konstantou TICK2US.
{
   int i;
   for (i = 0; i < us/TICK2US; i++);
}

void lcd_out( bit command, unsigned char dat )
// Posle data na display.
// command  ...   0 - instrukce
//                1 - data
// dat      ...   posilana data
{
   E = 0;
   RS = command;
   lcd_delay( DELAYTIME );

   E = 1;
   DB = dat;
   lcd_delay( DELAYTIME );
   E = 0;
}

void lcd_init( unsigned char setup )
// Inicializace displaye.
// setup ... nastaveni vlastnosti cursoru. (CR_SHOW | CR_BLINK)
{
   RW = 0;
   lcd_out( 0, 56 );          //Nastavi typ komunikace na 8-bitovou, 2-radkovy display.
```

```
   lcd_out( 0, 12 | setup );   //Zapne display a nastavi vlastnosti cursoru
   lcd_out( 0, 1 );            //Vymaze display a presune cursor na zacatek
}

void lcd_close()
// Vypne display
{
   lcd_out( 0, 8 );
}

void lcd_putch( char znak )
// Napise znak
{
   lcd_out(1, znak );
}

void lcd_write( char *str )
// Napise retezec znaku zakonceny \0
{
   unsigned char i=0;
   while ((str[i]!='\0') && (i<255))   //Delka retezce nesmi byt delsi nez 255 znaku
   {
      lcd_out( 1, str[i] );
      i++;
   }
}

void lcd_clr()
// Vymaze display a presune cursor na zacatek
{
   lcd_out( 0, 1 );
}

void lcd_ledon()
// Zapne podsviceni
{
   LED = 1;
}

void lcd_ledoff()
// Vypne podsviceni
{
   LED = 0;
}

void lcd_at( unsigned char x, unsigned char y )
// Presune cursor na zadanou pozici
{
   lcd_out( 0, 0x80 + (y*0x40) + x );
}
```

B.5. system.h

```
/*                                          */
/* system.h 2.0.0          (c) Evzen Thoendel */
/*                         23.11.2003       */
/*                                          */
/* Systemova knihovna                       */
/*                                          */
/*                                          */
/*                                          */

#ifndef SYSTEM_H
#define SYSTEM_H

extern unsigned char Input[8];
extern unsigned char Output[8];
extern unsigned char Button[8];

void ReadInput( void );
void WriteOutput( void );
void ReadButtons( void );
unsigned char OR( unsigned char ID, unsigned char LValue, unsigned char RValue );
unsigned char AND( unsigned char ID, unsigned char LValue, unsigned char RValue );
unsigned char NOT( unsigned char ID, unsigned char Value );
unsigned char UP_firstcall( unsigned char ID, unsigned char Value );
unsigned char UP_nextcall( unsigned char ID );
unsigned char DOWN_firstcall( unsigned char ID, unsigned char Value );
unsigned char DOWN_nextcall( unsigned char ID );
unsigned char d( unsigned char ID, unsigned char Value );
unsigned char RSnon( unsigned char ID, unsigned char R, unsigned char S );
unsigned char RS( unsigned char ID, unsigned char R, unsigned char S );
unsigned char GetNewValue( unsigned char ID );
unsigned char GetValue( unsigned char ID );
void InitScan( void );
void LCD( char *Line1, char *Line2, unsigned char LED, unsigned char Value );
void mem_init( void );

#endif
```

B.6. system.c

```
/*                                                  */
/* system.c 2.0.0              (c) Evzen Thoendel   */
/*                                   23.11.2003     */
/*                                                  */
/* Systemova knihovna                               */
/*                                                  */
/*                                                  */
/*                                                  */


#include <system.h>
#include <i2c.h>
#include <lcd.h>

unsigned char Input[8];
unsigned char Output[8];
unsigned char Button[8];

unsigned char mem[32];
unsigned char newmem[32];

unsigned char GetNewValue( unsigned char ID )
{
   return newmem[ID/8] & (1<<(ID%8));
}

unsigned char GetValue( unsigned char ID )
{
   return mem[ID/8] & (1<<(ID%8));;
}

void SetNewValue( unsigned char ID, unsigned char Value )
{
   if (Value) newmem[ID/8] = newmem[ID/8] | (1<<(ID%8));
      else newmem[ID/8] = newmem[ID/8] & ~(1<<(ID%8));
}

void SetValue( unsigned char ID, unsigned char Value )
{
   if (Value) mem[ID/8] = mem[ID/8] | (1<<(ID%8));
      else mem[ID/8] = mem[ID/8] & ~(1<<(ID%8));
}

void ReadInput( void )
{
   unsigned char I = i2c_readbyte( ADDR_VSTUPY );
   unsigned char i;
   for(i=0; i<8; i++)
      Input[i] = !(I & (1<<i));
```

```
}

void WriteOutput( void )
{
   unsigned char O = 0;
   unsigned char i;
   for (i=0; i<8; i++)
      O = O | ( (!Output[i]) <<i);
   i2c_sendbyte( ADDR_VYSTUPY, O );
}

void ReadButtons( void )
{
   unsigned char B = i2c_readbyte( ADDR_TLACITKA );
   unsigned char i;
   for(i=0; i<8; i++)
      Button[i] = !(B & (1<<i));
}

void InitScan( void )
{
      char i;
      for (i=0; i<32; i++) mem[i] = newmem[i];
}

unsigned char OR( unsigned char ID, unsigned char LValue, unsigned char RValue )
{
   unsigned char fNewValue = LValue || RValue;
   SetNewValue( ID, fNewValue );
   return fNewValue;
}

unsigned char AND( unsigned char ID, unsigned char LValue, unsigned char RValue )
{
   unsigned char fNewValue = LValue && RValue;
   SetNewValue( ID, fNewValue );
   return fNewValue;
}

unsigned char NOT( unsigned char ID, unsigned char Value )
{
   unsigned char fNewValue = !Value;
   SetNewValue( ID, fNewValue );
   return fNewValue;
}

unsigned char UP_firstcall( unsigned char ID, unsigned char Value )
{
   SetNewValue( ID, Value );
   return Value && !GetValue( ID );
```

```
}

unsigned char UP_nextcall( unsigned char ID )
{
   return GetNewValue(ID) && !GetValue(ID);
}

unsigned char DOWN_firstcall( unsigned char ID, unsigned char Value )
{
   SetNewValue( ID, Value );
   return !Value && GetValue( ID );
}

unsigned char DOWN_nextcall( unsigned char ID )
{
   return !GetNewValue(ID) && GetValue(ID);
}

unsigned char d( unsigned char ID, unsigned char Value )
{
   SetNewValue( ID, Value );
   return GetValue( ID );
}

unsigned char RS( unsigned char ID, unsigned char R, unsigned char S )
{
   unsigned char fNewValue;
   if (!S && !R) fNewValue = GetValue( ID );
   else if (!S && R) fNewValue = 0;
   else if (S && !R) fNewValue = 1;
   else fNewValue = 0;
   SetNewValue( ID, fNewValue );
   return fNewValue;
}

unsigned char RSnon( unsigned char ID, unsigned char R, unsigned char S )
{
   return !RS( ID, R, S );
}

void LCD( char *Line1, char *Line2, unsigned char LED, unsigned char Value )
{
   if (Value) {
      lcd_at(0,0);
      lcd_write( Line1 );
      lcd_at(0,1);
      lcd_write( Line2 );
      if (LED) lcd_ledon(); else lcd_ledoff();
   }
}
```

```
void mem_init( void )
{
   char i;
   for (i=0; i<32; i++) {
      mem[i] = 0;
      newmem[i] = 0;
   }
}
```

B.7. sja1000_reg.h

```
/*                                                  */
/* sja1000_reg.h 1.0.0          (c) Evzen Thoendel  */
/*                                     11.10. 2003  */
/*                                                  */
/* Registry chipu sja1000                           */
/*                                                  */
/*                                                  */
/*                                                  */


#ifndef SJA1000_REG_H
#define SJA1000_REG_H


/* definition for direct access to 8051 memory areas */
#define XBYTE ((unsigned char volatile xdata *) 0)


/* address and bit definitions for the Mode & Control Register */
#define ModeControlReg XBYTE[0]

#define RM_RR_Bit 0x01                  /* reset mode (request) bit */
#if defined (PeliCANMode)
#define LOM_Bit 0x02                    /* listen only mode bit */
#define STM_Bit 0x04                    /* self test mode bit */
#define AFM_Bit 0x08                    /* acceptance filter mode bit */
#define SM_Bit 0x10                     /* enter sleep mode bit */
#endif


/* address and bit definitions for the Interrupt Enable & Control Register */
#if defined (PeliCANMode)
#define InterruptEnReg XBYTE[4]         /* PeliCAN mode */

#define RIE_Bit 0x01                    /* receive interrupt enable bit */
#define TIE_Bit 0x02                    /* transmit interrupt enable bit */
#define EIE_Bit 0x04                    /* error warning interrupt enable bit */
#define DOIE_Bit 0x08                   /* data overrun interrupt enable bit */
#define WUIE_Bit 0x10                   /* wake-up interrupt enable bit */
#define EPIE_Bit 0x20                   /* error passive interrupt enable bit */
#define ALIE_Bit 0x40                   /* arbitration lost interr. enable bit*/
#define BEIE_Bit 0x80                   /* bus error interrupt enable bit */
#else                                   /* BasicCAN mode */
#define InterruptEnReg XBYTE[0]         /* Control Register */

#define RIE_Bit 0x02                    /* Receive Interrupt enable bit */
#define TIE_Bit 0x04                    /* Transmit Interrupt enable bit */
#define EIE_Bit 0x08                    /* Error Interrupt enable bit */
#define DOIE_Bit 0x10                   /* Overrun Interrupt enable bit */
```

```
#endif


/* address and bit definitions for the Command Register */
#define CommandReg XBYTE[1]
#define TR_Bit 0x01                    /* transmission request bit */
#define AT_Bit 0x02                    /* abort transmission bit */
#define RRB_Bit 0x04                   /* release receive buffer bit */
#define CDO_Bit 0x08                   /* clear data overrun bit */
#if defined (PeliCANMode)
#define SRR_Bit 0x10                   /* self reception request bit */
#else                                  /* BasicCAN mode */
#define GTS_Bit 0x10                   /* goto sleep bit (BasicCAN mode) */
#endif


/* address and bit definitions for the Status Register */
#define StatusReg XBYTE[2]
#define RBS_Bit 0x01                   /* receive buffer status bit */
#define DOS_Bit 0x02                   /* data overrun status bit */
#define TBS_Bit 0x04                   /* transmit buffer status bit */
#define TCS_Bit 0x08                   /* transmission complete status bit */
#define RS_Bit 0x10                    /* receive status bit */
#define TS_Bit 0x20                    /* transmit status bit */
#define ES_Bit 0x40                    /* error status bit */
#define BS_Bit 0x80                    /* bus status bit */


/* address and bit definitions for the Interrupt Register */
#define InterruptReg XBYTE[3]
#define RI_Bit 0x01                    /* receive interrupt bit */
#define TI_Bit 0x02                    /* transmit interrupt bit */
#define EI_Bit 0x04                    /* error warning interrupt bit */
#define DOI_Bit 0x08                   /* data overrun interrupt bit */
#define WUI_Bit 0x10                   /* wake-up interrupt bit */
#if defined (PeliCANMode)
#define EPI_Bit 0x20                   /* error passive interrupt bit */
#define ALI_Bit 0x40                   /* arbitration lost interrupt bit */
#define BEI_Bit 0x80                   /* bus error interrupt bit */
#endif


/* address and bit definitions for the Bus Timing Registers */
#define BusTiming0Reg XBYTE[6]
#define BusTiming1Reg XBYTE[7]

#define SAM_Bit 0x80                   /* sample mode bit
                                       1 == the bus is sampled 3 times
                                       0 == the bus is sampled once */
```

```
/* address and bit definitions for the Output Control Register */
#define OutControlReg XBYTE[8]
                        /* OCMODE1, OCMODE0 */
#define BiPhaseMode 0x00            /* bi-phase output mode */
#define NormalMode 0x02             /* normal output mode */
#define ClkOutMode 0x03             /* clock output mode */

                        /* output pin configuration for TX1 */
#define OCPOL1_Bit 0x20             /* output polarity control bit */
#define Tx1Float 0x00               /* configured as float */
#define Tx1PullDn 0x40              /* configured as pull-down */
#define Tx1PullUp 0x80              /* configured as pull-up */
#define Tx1PshPull 0xC0             /* configured as push/pull */

                        /* output pin configuration for TX0 */
#define OCPOL0_Bit 0x04             /* output polarity control bit */
#define Tx0Float 0x00               /* configured as float */
#define Tx0PullDn 0x08              /* configured as pull-down */
#define Tx0PullUp 0x10              /* configured as pull-up */
#define Tx0PshPull 0x18             /* configured as push/pull */


/* address definitions of Acceptance Code & Mask Registers */
#if defined (PeliCANMode)
#define AcceptCode0Reg XBYTE[16]
#define AcceptCode1Reg XBYTE[17]
#define AcceptCode2Reg XBYTE[18]
#define AcceptCode3Reg XBYTE[19]
#define AcceptMask0Reg XBYTE[20]
#define AcceptMask1Reg XBYTE[21]
#define AcceptMask2Reg XBYTE[22]
#define AcceptMask3Reg XBYTE[23]
#else /* BasicCAN mode */
#define AcceptCodeReg XBYTE[4]
#define AcceptMaskReg XBYTE[5]
#endif


/* address definitions of the Rx-Buffer */
#if defined (PeliCANMode)
#define RxFrameInfo XBYTE[16]
#define RxBuffer1 XBYTE[17]
#define RxBuffer2 XBYTE[18]
#define RxBuffer3 XBYTE[19]
#define RxBuffer4 XBYTE[20]
#define RxBuffer5 XBYTE[21]
#define RxBuffer6 XBYTE[22]
#define RxBuffer7 XBYTE[23]
#define RxBuffer8 XBYTE[24]
```

```
#define RxBuffer9 XBYTE[25]
#define RxBuffer10 XBYTE[26]
#define RxBuffer11 XBYTE[27]
#define RxBuffer12 XBYTE[28]
#else /* BasicCAN mode */
#define RxBuffer1 XBYTE[20]
#define RxBuffer2 XBYTE[21]
#define RxBuffer3 XBYTE[22]
#define RxBuffer4 XBYTE[23]
#define RxBuffer5 XBYTE[24]
#define RxBuffer6 XBYTE[25]
#define RxBuffer7 XBYTE[26]
#define RxBuffer8 XBYTE[27]
#define RxBuffer9 XBYTE[28]
#define RxBuffer10 XBYTE[29]
#endif


/* address definitions of the Tx-Buffer */
#if defined (PeliCANMode)
/* write only addresses */
#define TxFrameInfo XBYTE[16]
#define TxBuffer1 XBYTE[17]
#define TxBuffer2 XBYTE[18]
#define TxBuffer3 XBYTE[19]
#define TxBuffer4 XBYTE[20]
#define TxBuffer5 XBYTE[21]
#define TxBuffer6 XBYTE[22]
#define TxBuffer7 XBYTE[23]
#define TxBuffer8 XBYTE[24]
#define TxBuffer9 XBYTE[25]
#define TxBuffer10 XBYTE[26]
#define TxBuffer11 XBYTE[27]
#define TxBuffer12 XBYTE[28]
/* read only addresses */
#define TxFrameInfoRd XBYTE[96]
#define TxBufferRd1 XBYTE[97]
#define TxBufferRd2 XBYTE[98]
#define TxBufferRd3 XBYTE[99]
#define TxBufferRd4 XBYTE[100]
#define TxBufferRd5 XBYTE[101]
#define TxBufferRd6 XBYTE[102]
#define TxBufferRd7 XBYTE[103]
#define TxBufferRd8 XBYTE[104]
#define TxBufferRd9 XBYTE[105]
#define TxBufferRd10 XBYTE[106]
#define TxBufferRd11 XBYTE[107]
#define TxBufferRd12 XBYTE[108]
#else /* BasicCAN mode */
#define TxBuffer1 XBYTE[10]
```

```
#define TxBuffer2 XBYTE[11]
#define TxBuffer3 XBYTE[12]
#define TxBuffer4 XBYTE[13]
#define TxBuffer5 XBYTE[14]
#define TxBuffer6 XBYTE[15]
#define TxBuffer7 XBYTE[16]
#define TxBuffer8 XBYTE[17]
#define TxBuffer9 XBYTE[18]
#define TxBuffer10 XBYTE[19]
#endif


/* address definitions of Other Registers */
#if defined (PeliCANMode)
#define ArbLostCapReg XBYTE[11]
#define ErrCodeCapReg XBYTE[12]
#define ErrWarnLimitReg XBYTE[13]
#define RxErrCountReg XBYTE[14]
#define TxErrCountReg XBYTE[15]
#define RxMsgCountReg XBYTE[29]
#define RxBufStartAdr XBYTE[30]
#endif


/* address and bit definitions for the Clock Divider Register */
#define ClockDivideReg XBYTE[31]
#define DivBy1 0x07                 /* CLKOUT = oscillator frequency */
#define DivBy2 0x00                 /* CLKOUT = 1/2 oscillator frequency */
#define ClkOff_Bit 0x08             /* clock off bit, control of the CLK OUT pin */
#define RXINTEN_Bit 0x20            /* pin TX1 used for receive interrupt */
#define CBP_Bit 0x40                /* CAN comparator bypass control bit */
#define CANMode_Bit 0x80            /* CAN mode definition bit */



#endif
```

B.8. sja1000.h

```
/*                                                  */
/* sja1000_basic.h 1.0.0        (c) Evzen Thoendel  */
/*                                     11.10. 2003  */
/*                                                  */
/* Ovladaci funkce chipu sja1000                    */
/* v basic modu.                                    */
/*                                                  */
/*                                                  */

#ifndef SJA1000_H
#define SJA1000_H


/*- definition of bus timing values for different examples -------*/

/* bus timing values for
   - bit-rate                                : 250 kBit/s
   - oscillator frequency                    : 24 MHz, 1,0%
   - maximum propagation delay               : 1630 ns
   - minimum requested propagation delay     : 120 ns */
#define Presc_kB250_24 0x02                  /* baud rate prescaler : 3 */
#define SJW_kB250_24 0xC0                    /* SJW : 4 */
#define TSEG1_kB250_24 0x0A                  /* TSEG1 : 11 */
#define TSEG2_kB250_24 0x30                  /* TSEG2 : 4 */

/* bus timing values for
   - bit-rate                                : 1 MBit/s
   - oscillator frequency                    : 24 MHz, 0,1%
   - maximum tolerated propagation delay     : 747 ns
   - minimum requested propagation delay     : 45 ns */
#define Presc_MB_24 0x00                     /* baud rate prescaler : 1 */
#define SJW_MB_24 0x00                       /* SJW : 1 */
#define TSEG1_MB_24 0x08                     /* TSEG1 : 9 */
#define TSEG2_MB_24 0x10                     /* TSEG2 : 2 */

/* bus timing values for
   - bit-rate                                : 100 kBit/s
   - oscillator frequency                    : 24 MHz, 1,0%
   - maximum tolerated propagation delay     : 4250 ns
   - minimum requested propagation delay     : 100 ns */
#define Presc_kB100_24 0x07                  /* baud rate prescaler : 8 */
#define SJW_kB100_24 0xC0                    /* SJW : 4 */
#define TSEG1_kB100_24 0x09                  /* TSEG1 : 10 */
#define TSEG2_kB100_24 0x30                  /* TSEG2 : 4 */

/* bus timing values for
   - bit-rate                                : 1 MBit/s
   - oscillator frequency                    : 16 MHz, 0,1%
   - maximum tolerated propagation delay     : 623 ns
```

```
   - minimum requested propagation delay  : 23 ns */
#define Presc_MB_16 0x00                  /* baud rate prescaler : 1 */
#define SJW_MB_16 0x00                    /* SJW : 1 */
#define TSEG1_MB_16 0x04                  /* TSEG1 : 5 */
#define TSEG2_MB_16 0x10                  /* TSEG2 : 2 */

/* bus timing values for
   - bit-rate                             : 100 kBit/s
   - oscillator frequency                 : 16 MHz, 1,0%
   - maximum tolerated propagation delay  : 4450 ns
   - minimum requested propagation delay  : 500 ns */
#define Presc_kB100_16 0x04               /* baud rate prescaler : 5 */
#define SJW_kB100_16 0xC0                 /* SJW : 4 */
#define TSEG1_kB100_16 0x0A               /* TSEG1 : 11 */
#define TSEG2_kB100_16 0x30               /* TSEG2 : 4 */

/*- end of definitions of bus timing values ----------------------*/

void sja_enable( void );
void sja_disable( void );
void sja_reset( void );
void sja_clrreset( void );
void sja_baudrate( int baudrate );
void sja_waitfor_readytosend( void );
unsigned char sja_newdata( void );


#endif
```

B.9. sja1000.c

```
/*                                                    */
/* sja1000_basic.c 1.0.0 (c)   Evzen Thoendel         */
/*                                        11.10. 2003 */
/*                                                    */
/* Ovladaci funkce chipu sja1000                      */
/* v basic modu.                                      */
/*                                                    */
/*                                                    */

#define ClrByte 0x00
#define EXTRAM 0x02
#undef PeliCANMod

#include "sja1000.h"
#include "sja1000_reg.h"

sbit at 0xB4 CS ;
sfr at 0x8E AUXR ;
sbit at 0xA8 EX0  ;
sbit at 0xAF EA   ;

void sja_enable( void )
{
   CS = 0;
   AUXR = AUXR | EXTRAM;
}

void sja_disable( void )
{
   CS = 1;
   AUXR = AUXR & ~EXTRAM;
}

void sja_reset( void )
{
   while((ModeControlReg & RM_RR_Bit ) == ClrByte){
      ModeControlReg = ModeControlReg | RM_RR_Bit ;
   }
   EA = 0;     //vypne vsechna preruseni
   EX0 = 0;
}

void sja_clrreset( void )
{
   do {
      ModeControlReg = ClrByte;
   } while((ModeControlReg & RM_RR_Bit ) != ClrByte);
   EA = 1;
   EX0 = 1;
```

```
}

void sja_baudrate( int baudrate )
{
   switch (baudrate) {
      case 100:
         BusTiming0Reg = SJW_kB100_24 | Presc_kB100_24;
         BusTiming1Reg = TSEG2_kB100_24 | TSEG1_kB100_24;
         break;
      case 250:
         BusTiming0Reg = SJW_kB250_24 | Presc_kB250_24;
         BusTiming1Reg = TSEG2_kB250_24 | TSEG1_kB250_24;
         break;
      case 1000:
         BusTiming0Reg = SJW_MB_24 | Presc_MB_24;
         BusTiming1Reg = TSEG2_MB_24 | TSEG1_MB_24;
         break;
      default:
         BusTiming0Reg = SJW_kB100_24 | Presc_kB100_24;
         BusTiming1Reg = TSEG2_kB100_24 | TSEG1_kB100_24;
         break;
   }
}

void sja_waitfor_readytosend( void )
{
   while((StatusReg & TBS_Bit ) != TBS_Bit );
}

unsigned char sja_newdata( void )
{
   unsigned char c;
   c = (StatusReg & RBS_Bit ) == RBS_Bit;
   return c;
}
```

B.10. can.h

```
/*                                                    */
/* can.h 1.0.0                (c) Evzen Thoendel     */
/*                                     11.10. 2003    */
/*                                                    */
/* Ovladaci funkce sbernice CAN                       */
/*                                                    */
/*                                                    */
/*                                                    */


#ifndef CAN_H
#define CAN_H

struct can_init_data {
   unsigned char AcceptCode;
   unsigned char AcceptMask;
   int BaudRate;
};

struct can_message {
   unsigned char ID[2];
   unsigned char can_data[8];
};

void can_init( struct can_init_data *init_data );
void can_send( struct can_message *message );
unsigned char can_receive( struct can_message *message );

#endif
```

B.11. can.c

```
/*                                                  */
/* can.c 1.0.0                (c) Evzen Thoendel    */
/*                                     11.10. 2003  */
/*                                                  */
/* Ovladaci funkce sbernice CAN                     */
/*                                                  */
/*                                                  */
/*                                                  */

#include "can.h"
#include "sja1000.h"
#include "sja1000_reg.h"

void can_init( struct can_init_data *init_data )
{
   sja_enable();                                    //CS = 0, Externi pamet: vypnuta
   sja_reset();

   ClockDivideReg = ClkOff_Bit | CBP_Bit;           //Basic mode,
                                                    //Vystupni oscilator: vypnut,
                                                    //Vstupni komparator: vypnut,
                                                    //Interrupt impuls do TX1: vypnut.
   AcceptCodeReg = init_data->AcceptCode;           //Nastaveni filtru
   AcceptMaskReg = init_data->AcceptMask;           //Nastaveni masky filtru

   sja_baudrate( init_data->BaudRate );             //Nastaveni prenosove rychlosti

   OutControlReg = Tx1Float | Tx0PshPull | NormalMode;

   sja_clrreset();
   sja_disable();
}

void can_send( struct can_message *message )
{
   sja_enable();

   sja_waitfor_readytosend();

   TxBuffer1 = message->ID[0];
   TxBuffer2 = message->ID[1];

   TxBuffer3 = message->can_data[0];
   TxBuffer4 = message->can_data[1];
   TxBuffer5 = message->can_data[2];
   TxBuffer6 = message->can_data[3];
   TxBuffer7 = message->can_data[4];
   TxBuffer8 = message->can_data[5];
   TxBuffer9 = message->can_data[6];
```

```
    TxBuffer10= message->can_data[7];

    CommandReg = TR_Bit ;

    sja_disable();
}

unsigned char can_receive( struct can_message *message )
{
    sja_enable();
    if ( sja_newdata() ) {
        message->ID[0] = RxBuffer1;
        message->ID[1] = RxBuffer2;

        message->can_data[0] = RxBuffer3;
        message->can_data[1] = RxBuffer4;
        message->can_data[2] = RxBuffer5;
        message->can_data[3] = RxBuffer6;
        message->can_data[4] = RxBuffer7;
        message->can_data[5] = RxBuffer8;
        message->can_data[6] = RxBuffer9;
        message->can_data[7] = RxBuffer10;

        CommandReg = RRB_Bit;
        sja_disable();
        return 1;
    }
    sja_disable();
    return 0;
}
```

B.12. can_demo.c

```
#include <lcd.h>
#include <i2c.h>
#include "can.h"

struct can_init_data init_data;
struct can_message rcv_message;
struct can_message snd_message;

void main()
{
   unsigned char i;
   unsigned char b;

   snd_message.ID[0] = 0xFF;
   snd_message.ID[1] = 0x28;
   snd_message.can_data[0] = 'C';
   snd_message.can_data[1] = 'A';
   snd_message.can_data[2] = 'N';
   snd_message.can_data[3] = ' ';
   snd_message.can_data[4] = 'T';
   snd_message.can_data[5] = 'E';
   snd_message.can_data[6] = 'S';
   snd_message.can_data[7] = 'T';

   lcd_init( 0 );

   init_data.AcceptCode = 0xFF;
   init_data.AcceptMask = 0x00;
   init_data.BaudRate = 1000;
   can_init( &init_data );

   while(1) {
      b = i2c_readbyte( ADDR_TLACITKA );
      if ( can_receive( &rcv_message ) ) {
         lcd_at(0,0);
         for(i=0; i<8; i++) lcd_putch( rcv_message.can_data[i] );
      }
      if (!(b & 1)) {
         lcd_at(0,1);
         lcd_write("Posilam data: ");
         can_send( &snd_message );
         lcd_write("OK");
      }
   }
}
```

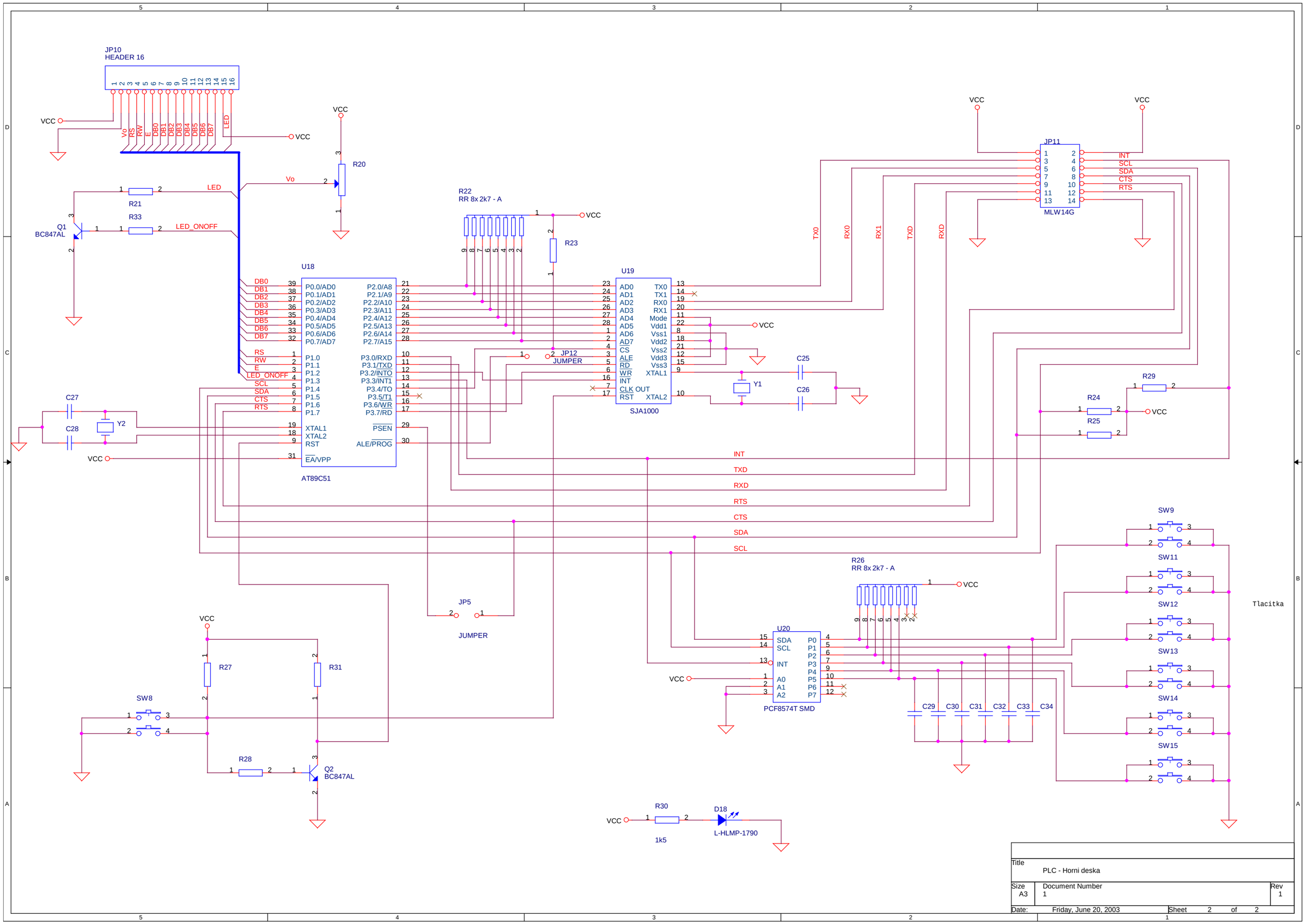
B.13. Vzorový program (program.c)

```
/*                                                  */
/* program.h 1.0.0              (c) Evzen Thoendel  */
/*                                     7.11. 2003   */
/*                                                  */
/* Vzorovy program                          */
/*                                                  */
/*                                                  */
/*                                                  */

//section include
#include <system.h>
#include <lcd.h>

void SetOutput()
{
//section program
}

void main()
{
   lcd_init( 0 );
   mem_init();
   while(1) {
      InitScan();
      ReadInput();
      ReadButtons();
      SetOutput();
      WriteOutput();
   }
}
```

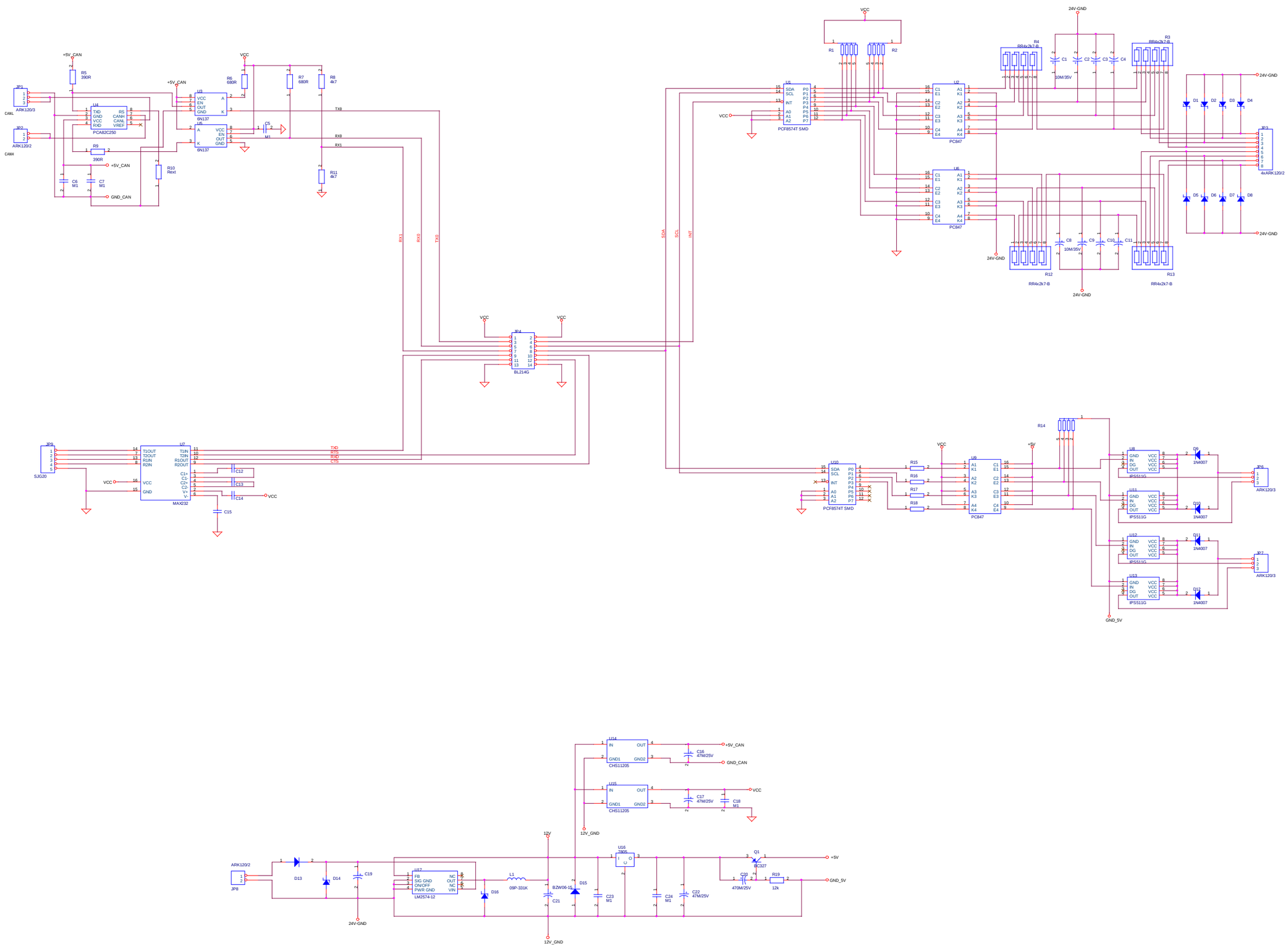
B.14. Instrukční soubor LCD displeje

Instruction	CODE										DESCRIPTION	Executed Time(max) focs=250KHz
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clear all display and returns the cursor to the home position (Address 0)	1.64mS
Cursor At Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position DDRAM contents remain unchanged	1.64mS
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40µS
Display On/Off Control	0	0	0	0	0	0	1	D	C	B	Sets the ON/OFF of all display (D) cursor ON/OFF (C), and blink of cursor position character (B)	40µS
Cursor/Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DDRAM contents	40µS
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines(N) and character font (F).	40µS
CGRAM Address Set	0	0	0	1	A_{CG}						Sets the CGRAM, data is sent and received after this setting.	40µS
DDRAM Address Set	0	0	1	A_{DD}							Sets the CGRAM, data is sent and received after this setting.	40µS
Busy Flag/ Address Read	0	1	BF	AC							Reads Busy flag (FB) indicating internal operation is being performed and reads address counter contents.	0µS
CGRAM/DDRAM Data Write	1	0	W_{RITE} D_{ATA}								Writes data into DDRAM or CGRAM	40µS
CGRAM/DDRAM Data Read	1	1	R_{EAD} D_{ATA}								Reads data into DDRAM or CGRAM	40µS

Code		Description		Executed Time (max.)
I/D=1: Increment I/D=0: Decrement S=1: With display shift S/C=1: Display shift S/C=0: Cursor movement R/L=1: Shift to the right R/L=0: Shift to the left DL=1 : 8-bit	DL = 0:4-bit 1/16 duty 1/8 duty, 1/11 duty F= 1: 5x10 dots F=0: 5x7 dots BF=1: Internal Operation is being performed BF=0: Instruction acceptable	DDRAM: Display Data RAM CGRAM: Character Generator RAM ACG: CGRAM Address ADD: DDRAM Address Corresponds to cursor address AC: Address Counter, used for both DDRAM and CGRAM *: Invalid		Fcp or focs = 250kHz However, when frequency changes, execution time also changes Example If fcp or focs is 270kHz 40µS x ~~250~~ = 37µS 270

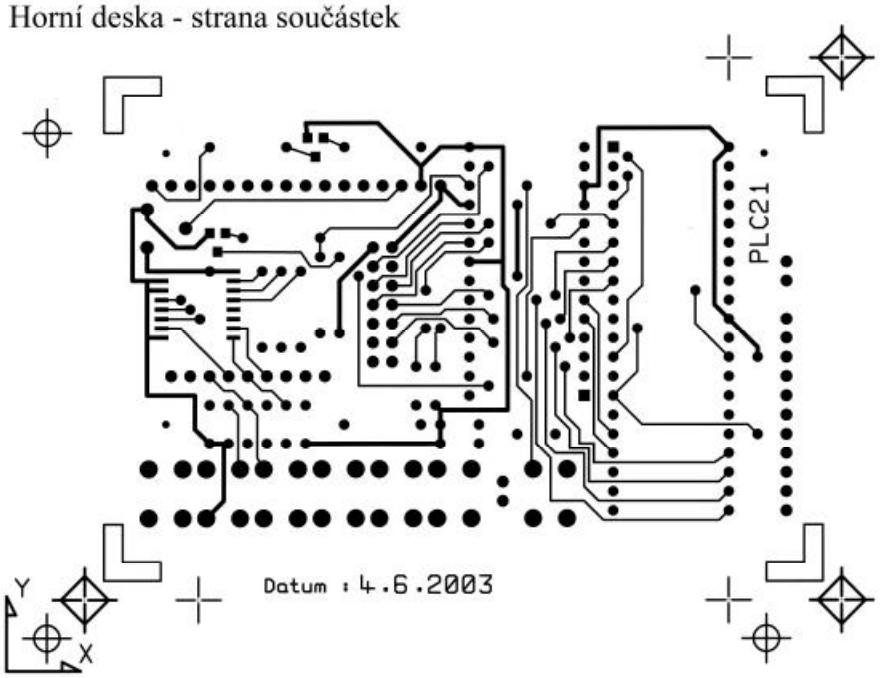


Title	PLC - Horni deska	
Size A3	Document Number 1	Rev 1
Date: Friday, June 20, 2003	Sheet 2 of 2	

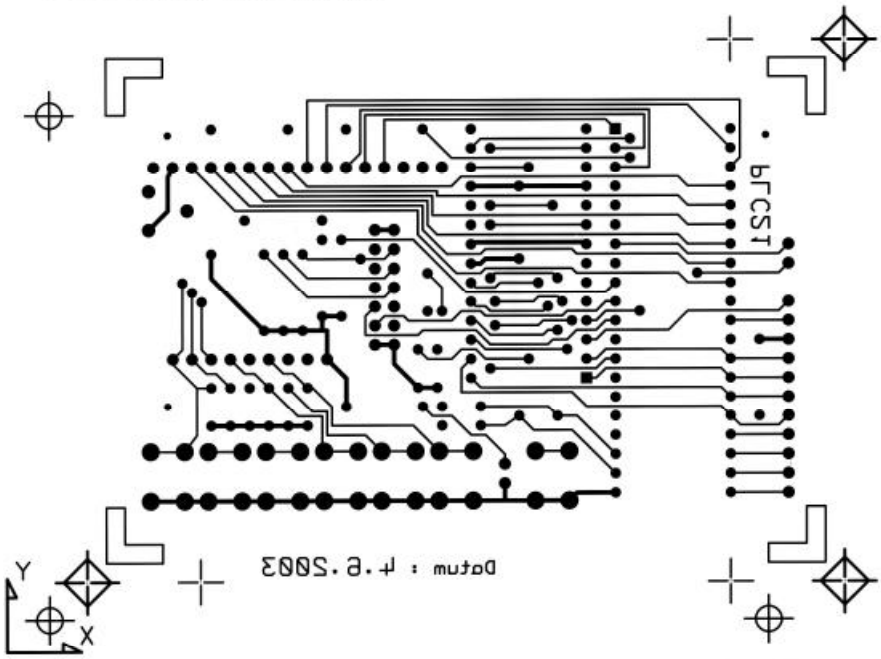


Title	PLC - Dolni deska
Size A1	Document Number 1
Rev	1
Date:	Wednesday, December 17, 2003
Sheet	1 of 2

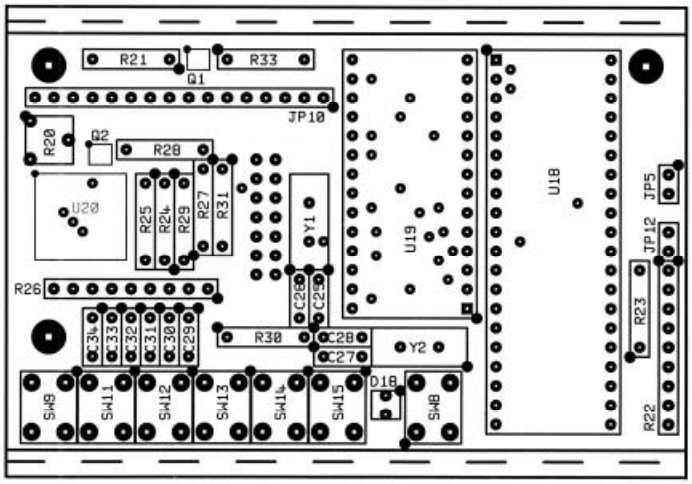
Horní deska - strana součástek



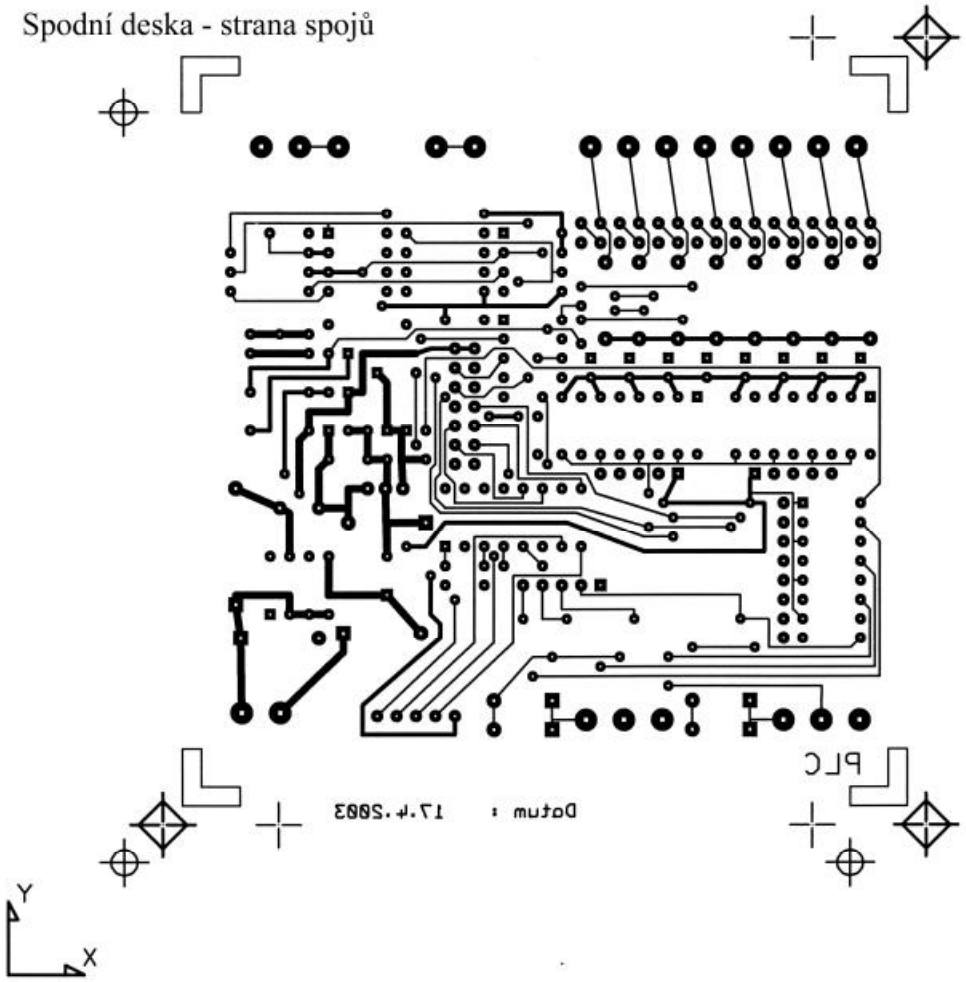
Horní deska - strana spojů



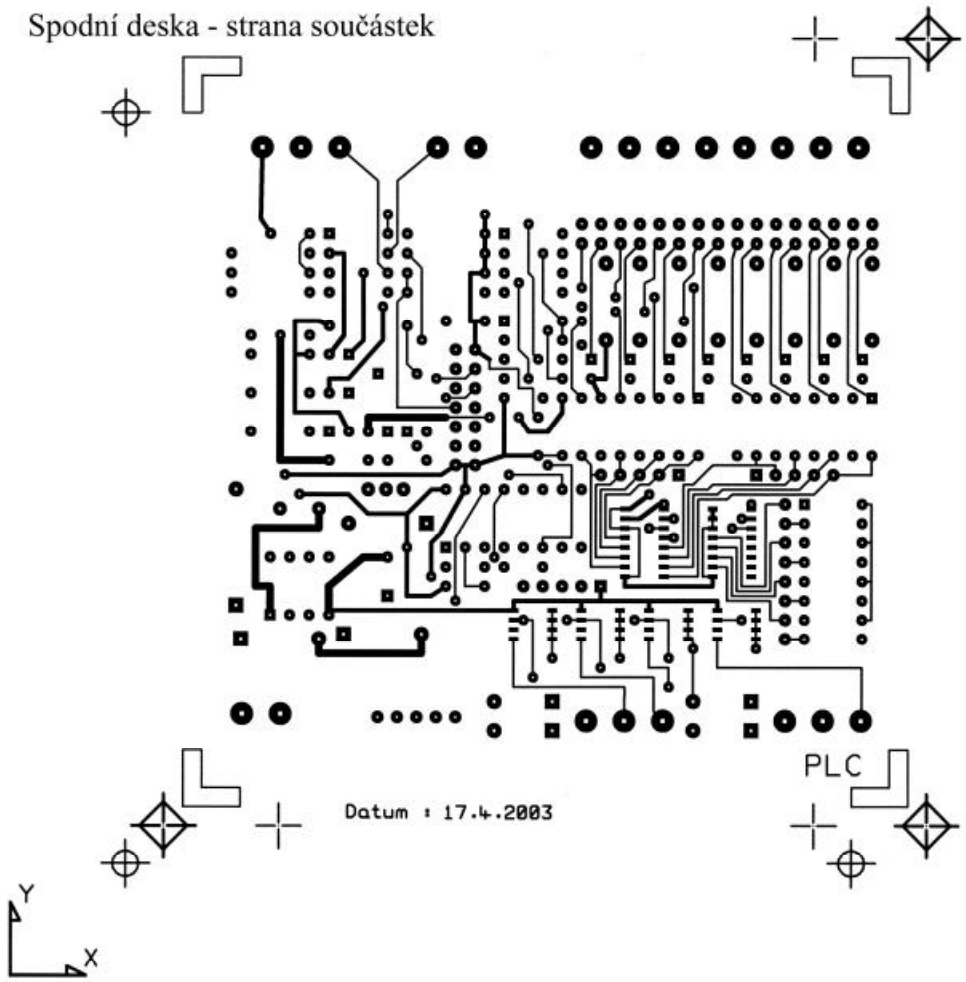
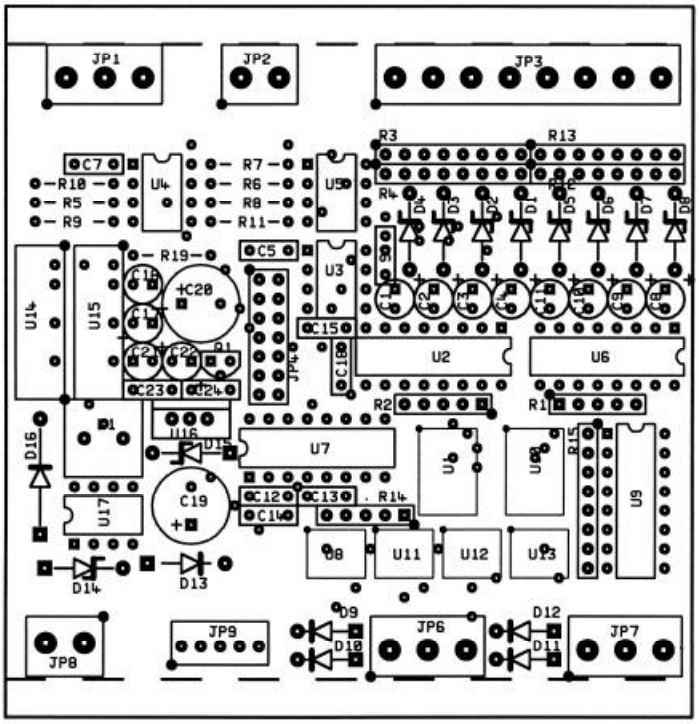
Horní deska - rozložení součástek



Spodní deska - strana spojů



Spodní deska - strana součástek



Spodní deska - rozložení součástek