Programová podpora mikropočítačového vývojového kitu 908LJ12

Jaroslav Hnátík

Bakalářská práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav řízení procesů
akademický rok: 2005/2006

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jaroslav HNÁTÍK**
Studijní program:	**B 2807 Chemické a procesní inženýrství**
Studijní obor:	**Automatizace a řídicí technika**
Téma práce:	**Programová podpora mikropočítačového vývojového kitu 908LJ12**

Zásady pro vypracování:

1. Prostudujte hardwarové vlastnosti vývojového kitu 908LJ12 od firmy Beta Control, popište jeho funkce, periferie a použitý mikropočítač.
2. Vytvořte v jazyce symbolických adres programové vybavení pro efektivní obsluhu vybraných integrovaných periferií.
3. Sestavte demonstrační program využívající vytvořené obslužné podprogramy pro ověření jejich správné funkce.
4. Vypracujte přehlednou dokumentaci s ilustračními příklady k realizovaným obslužným podprogramům.

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **CPU08 Central Processor Unit Reference manual. Motorola, 2001**
2. **Starter kit LJ12EVB user's manual. Beta Control, 2002**
3. **MC68HC908LJ12 Technical Data. Freescale semiconductor, 2005**
4. **M68ICS08 68HC08 In-circuit simulator Operators Manual. P&E Microcomputer systems, 2000**
5. **Školní mikropočítač MS 11-A10 STUDENT Příručka pro uživatele. Uherské Hradiště, 1992**

Vedoucí bakalářské práce: **Ing. Petr Dostálek**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **14. února 2006**

Termín odevzdání bakalářské práce: **16. června 2006**

Ve Zlíně dne 25. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Petr Dostál, CSc.
ředitel ústavu

ABSTRAKT

Cílem této práce je prostudovat vývojový kit LJ12EVB firmy Beta Control. Kit je vybaven mikroprocesorem Motorola MC68HC908LJ12CFU.Obsahuje řadu periferií, jako např. LCD, klávesnici a jiné. Tyto periferie mají mnoho nastavení a jejich obsluha vyžaduje znalosti práce s konkrétní periferií. Proto jsou vytvořeny obslužné podprogramy, které umožňují efektivní obsluhu těchto periferií bez nutnosti podrobnějších znalostí těchto periferií. K jednotlivým podprogramům je vytvořena dokumentace a demonstrační program pro ukázku práce s nimi.

Klíčová slova: Motorola, HC08, LJ12EVB, periferie, obsluha

ABSTRACT

This work's goal is to peruse starter kit LJ12EVB made by firm Beta Control. This starter kit contain Motorola MC68HC908LJ12CFU microprocessor. Also contain many peripherals like LCD display, keyboard and any other. This peripherals have many setups and theirs control require expert knowledge. So we develop service subroutines, which provide useful control of this peripherals without depth knowledge of this peripherals. To severally service subroutines was made users documentation. There is also simple demo program presented, that demonstrate work with this subroutines.

Keywords: Motorola, HC08, LJ12EVB, peripheral, service

Děkuji Ing. Petru Dostálkovi za vedení, pomoc a cenné rady při řešení této práce.

Souhlasím s tím, že s výsledky mé práce může být naloženo podle uvážení vedoucího bakalářské práce, ředitele ústavu a institutu. V případě publikace budu uveden jako spoluautor.

Prohlašuji, že na celé bakalářské práci jsem pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně, 13.6.2006 ….....

podpis

OBSAH

ÚVOD

Při vývoji aplikací pro mikropočítač Motorola 68HC908LJ12CFU osazeném v kitu LJ12EVB firmy Beta Control je zapotřebí nejen znalosti příslušného programovacího jazyka, ale při použití periferií tohoto kitu také znalosti práce s těmito periferiemi, jejich nastavení, možnosti a omezení jejich používání.

V této práci se nejprve budeme zabývat hardwarovým řešením tohoto kitu, možnostmi procesoru HC08, možnosti propojení s PC a možnosti programování kitu pomoci PC.

V další části bude rozebrána práce s jednotlivými periferiemi a vytvořeny a popsány obslužné programy pro efektivní práci s jednotlivými periferiemi. Obslužné programy budou jsou řešeny jako samostatné podprogramy. Jejichž použití již dále nebude vyžadovat podrobnou znalost dané periferie a práce s ní, aby se programátor mohl plně věnovat řešení funkce samotné aplikace a nemusel znova řešit programovou obsluhu periferií. Ke každému podprogramu bude vytvořena dokumentace s příklady.

Jednotlivé podprogramy je možno slučovat v jeden celek, vytvářející konkretní aplikaci. Jako příklad takové aplikace bude vytvořen demonstrační program ukazující práci s těmito podprogramy.

1 VÝVOJOVÝ KIT LJ12EVB

Vývojový kit LJ12EVB firmy Beta Control je levnou a přitom efektivní možností jak vytvářet aplikace pro procesory řady HC08. Je vybaven mnoha periferiemi, na kterých je možno ukázat schopnosti těchto výkonných 8-bitových mikropočítačů. Obsahuje jak vstupní periferie, jako například klávesnici, tak výstupní, jako například LCD display a relé. Díky tomu jej má mnoho variant použití a je vhodný i pro výuku.

1.1 Centrální procesorová jednotka CPU

Kit LJ12EVB je osazen mikroprocesorem Motorola MC68HC908LJ12CFU.

Řada mikroprocesorů HC08 má mnoho variant lišící se osazením periferiemi, velikostí a druhem použité paměti, provozní teplotou a typem pouzdra.

Značení mikroprocesorů řady HC08 se řídí tímto systémem [6]:

MC - značení výrobce. V našem případě Motorola

60HC - značení řady mikroprocesoru a obvodové technologie

9 - verze paměti. 9 značí paměť typu FLASH, 7 EEPROM

není li zde uvedeno nic, je použita paměť typu ROM

08 - typové značení řady mikroprocesorů

LJ - značení skladby periferií. Volba periferií je dána využitím v mikropočítače v praxi. Např. písmena GP značí obecné použití.

12 - přibližná velikost paměti v kB. Bývá od 1kB do 64 kB. V našem případě je použito 12kB paměti typu FLASH

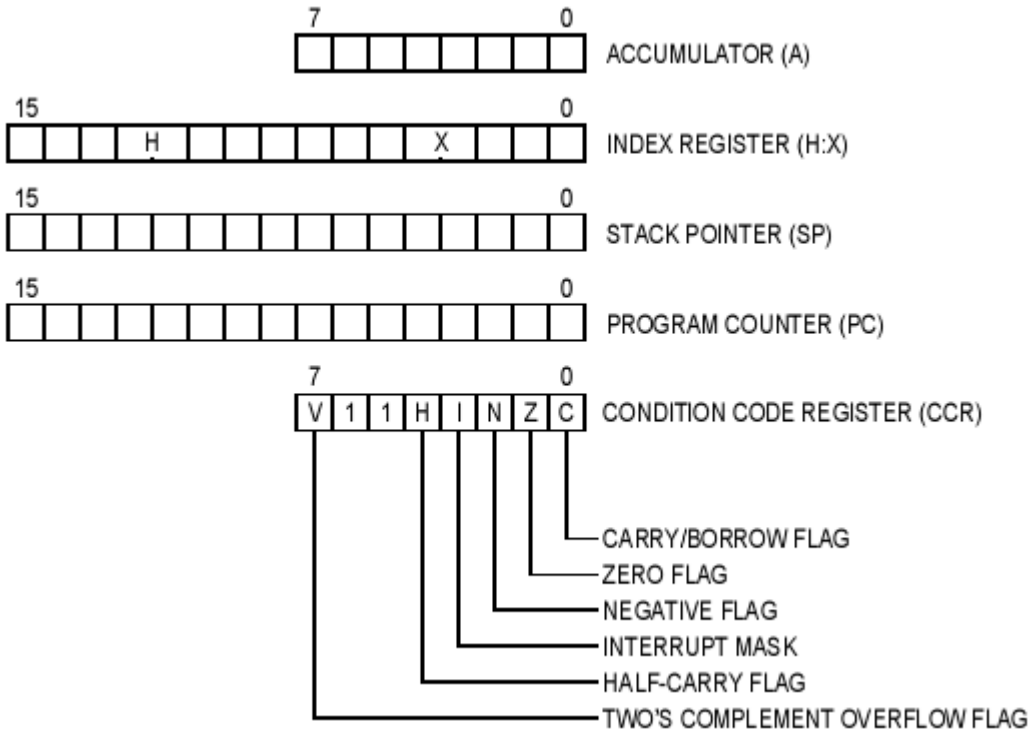
C - teplotní rozsah, který je v našem případě -40 ºC až 85 ºC

FU - typ použitého pouzdra. V našem případě 64 QFP (14 x 14)

Mikroprocesory řady HC08 [2]jsou pokračovateli řady HC05 s níž jsou směrem vzhůru kompatibilní na úrovni strojového kódu. Oproti řadě HC05 přinášejí 16 bitový index registr, 16 bitový ukazatel zásobníku a 78 nových instrukcí. Jedná se o 8 bitové jádro s minimem pracovních registrů s rychlým a flexibilním přístupem do paměti.

Mikroprocesor HC08 využívá tyto registry (viz obr č. 1) :

- Akumulátor A – 8 bitový registr používaný pro uchovávání výsledků aritmetických a logických operací prováděných CPU
- Index registr H:X – 16 bitový registr umožňující adresování paměti.
- Ukazatel zásobníku SP – 16 bitový registr obsahující adresu začátku zásobníku
- Programový čítač PC – 16 bitový registr obsahující adresu paměti odkud je čtena instrukce. Hodnota se mění po každém čtení instrukce nebo při každém skoku
- Registr příznaků C – 8 bitový registr obsahující stavové informace (přenos, přetečení a podobně)



Obr. 1. Registry procesoru HC08

Mezi základní vlastnosti mikroprocesorů řady HC08 patří [2],[6] :

- von Neumanova architektura, tj data i program jsou umístěny ve stejném prostoru
- periferie jsou paměťově mapovatelné
- maximální takt vnitřní sběrnice a CPU 8 MHz

- kompatibilita s kódem 68HC05 směrem vzhůru
- hardwarové násobení a dělení
- minimum registrů
- 16 bitový index registr umožňující adresovat až 64 kB paměti
- rychlý přístup do paměti
- přesuny v paměti bez nutnosti použití akumulátoru
- rozšířená podpora
- možnost použití režimů WAIT a STOP pro snížení spotřeby
- podpora vyšších programovacích jazyků (hlavně jazyka C)
- hlídací obvod COP (WATCHDOG)

1.1.1 Systémový modul SIM

Kontroluje CPU, řídí činnost hlídacího obvodu COP, generuje hodinové signály, řídí režimy STOP a WAIT. Také má na starosti priority přerušení.

1.1.2 Modul hlídání napájení LVI

Tento modul dovoluje kontrolovat napájecí napětí. Dokáže upozorňovat na snížené napětí, případně samočinně resetovat mikropočítač, při poklesu napětí pod limitní hodnotu.

1.1.3 Paměť FLASH

Na čipu je instalována 12kB paměti typu FLASH. Zároveň s touto pamětí je nainstalován také programátor této paměti. To umožní mimo jiné jednoduché programování mikroprocesoru bez nutnosti použití speciálního programátoru. Tato paměť umí emulovat paměť EEPROM. Část této paměti lze navíc uzamknout proti nechtěnému přepisu. Je garantováno 10000 přepisů paměti a 10 let udržení informací uložené v paměti v teplotním rozmezí -40 ºC až 125 ºC.

Při použití rozhraní miniMON (např. ve spojení s kitem JANUS) je paměť FLASH programovatelná po sériovém portu.

1.1.4 Sériové periferní rozhraní SPI

Přenáší data vysokou rychlostí na malé vzdálenosti (na jedné desce plošných spojů). Slouží k propojení mikropočítače s periferiemi jako třeba LCD, A/D D/A převodník a podobně. Jde o plně duplexní synchronní přenos používající vyrovnávací paměť pro vysílání i příjem.

1.1.5 A/D převodník

A/D převodník periodicky vzorkuje vnější analogový signál a na výstupu poskytuje patřičné číslicové hodnoty. Lze převádět jednorázově nebo trvale. Lze nastavit příznak, nebo přerušení po dokončení převodu.

1.1.6 Monitor MON08

Mikropočítače řady HC08 jsou vybaveny podporou ladění a programování tzv. na čipu. To umožňuje vestavěný monitor MON08. Za normálních okolností v tzv. MONITOR MODE, pracuje procesor s uživatelskou aplikací a využívá všechny vývody. V případě, že chce uživatel přejít do tzv. MONITOR MODE dojde k přerušení vykonávání a spustí se speciální vestavěná aplikace tzv. MONITOR. V tomto modu procesor komunikuje s nějakým jiným systémem, např vývojovým prostředím v PC. To se provádí tak, že se jeden vývod vyčlení pro obousměrnou komunikaci. V tomto modu je možno posílat mikropočítači řídící signály, které zajišťují čtení a zápis registrů procesoru a paměti.

Mapa paměti mikropočítače Motorola MC68HC908LJ12CFU je uvedena v příloze P 3. Tato mapa je velmi důležitá. Bez ní není možno mikropočítač naprogramovat. Aplikace vytvořené pro některý konkrétní mikropočítač řady HC08 lze po drobných úpravách kódu pro ovládání periferií používat na jiném typu této řady.

1.2 Vlastnosti kitu

Vývojový kit přidává k mikroprocesoru HC08 mnoho přídavných zařízení, které umožňují lépe demonstrovat možnosti těchto mikroprocesorů a zjednoduší vývoj a ladění aplikací. Existuje mnoho variant těchto kitů. Liší se použitým mikroprocesorem i použitými periferiemi. V České republice ve spolupráci s firmou Freescale (dříve Motorola) vyvíjí a vyrábí vývojové kity pro mikropočítače řady HC08 firma Beta Control, s.r.o. Brno.

Vývojový kit LJ12EVB je levný a jednoduchý vývojový prostředek určený hlavně pro začátečníky.

Mezi základní vlastnosti tohoto kitu patří [1][3]:

- 12 kB paměti typu FLASH umožnující tzv. in-circuit programing (ISP)
- 512 B paměti RAM
- LCD display s osmi sedmi segmentovými moduly a 25 grafickými symboly
- rozhraní pro asynchronní sériový přenos (SCI)

- vnitřní obvod reálného času (RTC) jenž ale nedokáže pracovat s oscilátorem propojovacího kitu JANUS
- šestikanálový, 10-ti bitový A/D převodník
- dva 16-bitové čítače / časovače
- externí 32 kHz oscilátor
- rozhraní miniMON pro programování a ladění aplikací
- 5V napájení
- 2 LED diody
- reléový výstup
- bzučák
- 12+2 tlačítek
- teplotní senzor

2 VÝVOJOVÉ PROSTŘEDKY

2.1 Propojení kitu LJ12EVB s PC

Propojení vývojového kitu LJ12EVB s počítačem je umožněno přes kit JANUS[4]. Tento kit je určen k vývoji software pro skupinu mikropočítačů NITRON, tedy 68HC908QT a 68HC908QY. K propojeni kitu JANUS a PC slouží standardní sériové rozhraní RS232. Tento kit je navíc ale opatřen také konektorem miniMON, díky kterému je jej možno využít jako vývojový a programovací adaptér pro jiné zařízení vybavená tímto konektorem. Tím je i náš kit LJ12EVB. K propojení je zapotřebí vyrobit propojovací kabel. Osazení konektoru najdeme na obrázku č. 2

OSC1	/IRQ	PTA1	PTA2	PTA0
2	4	6	8	10
1	3	5	7	9
VCC	GND	/RES		PTC1

Obr. 2. Zapojení konektoru miniMON

Kit JANUS nám také přes tento kabel zajistí napájení našeho kitu. Svým vestavěným oscilátorem také určuje frekvenci sběrnice. Dále pomocí tlačítka SW1 umožní reset mikropočítače a pomocí tlačítka SW2 umožňuje přepínání USER a MONITOR modu.

2.2 Vývoj a ladění aplikací

Vývoj aplikací pro mikropočítače se běžně provádí jazyce assembler a/nebo v jazyce C. Pro oba jazyky existuje několik různých vývojových prostředků. Většina z nich je zcela zdarma, nebo zdarma s omezením velikosti výsledného kódu.

Mezi nejznámější a nejpoužívanější patří :

- ICC08 firmy ImageCraft
- CodeWarrior Development Studio for HC08 od firmy Metrowerks
- Vývojové prostředí WinIDE od firmy P&E Microcomputer Systems Inc.

Vývojové prostředí WinIDE tvoří kompletní soubor zdarma dostupných prostředků pro tvorbu aplikací v jazyce assembler. Při vývoji obslužných programů bylo použito právě toto vývojové prostředí. Jeho součástí jsou [4] [5] :

- WinIDE – základní prostředí. Jedná se vlastně o editor, jehož součástí je překladač CASM08Z. Ostatní části jsou volány jako externí aplikace.
- ICS08LJZ – in-circuit simulátor. Umožňuje v tzv. MONITOR modu provádět ladění vyvíjené aplikace. Vlastní kód je vykonáván v PC . Kit nemusí být připojen, simulace probíhá celá v PC. Při připojení kitu lze provádět kód v PC ale vstupy a výstupy mikropočítače nejsou simulovány ale odpovídají přímo vývodům mikropočítače.
- PROG08SZ – programátor FLASH paměti mikropočítače. Umožňuje číst a mazat a zapisovat paměť FLASH. Díky tomu lze naprogramovat do připojeného mikropočítače aplikaci, která posléze může být spuštěna v tzv. USER modu
- ICD08SZ – in-circuit debuger. Na rozdíl od simulátoru ICS je kód aplikace prováděn přímo CPU mikropočítače. Aplikace musí být uložena v paměti a jednotlivé instrukce se čtou z této paměti a pomocí CPU provádí. Dává lepší obraz chování aplikace než simulátor.

3 OBSLUŽNÉ PROGRAMY JEDNOTLIVÝCH PERIFERIÍ

3.1 Čítač/časovač, použití čítače pro časové zpoždění

Mikropočítač kitu disponuje dvěma 16 bitovými čítači/časovači. Jelikož jsou základní operace pro oba čítače/časovače stejné, bude dále uváděno jen použití čítače/časovače 1. Použití druhého je stejné, jen je třeba užívat patřičných registrů (T2MOD místo T1MOD, T2SC místo T1SC a tak dále).

Využití časovače není ve spojení Kitu LJ12EVB a Janus příliš vhodné, jelikož krystal Janus není kompatibilní s oscilátorem kitu. Časovač poté počítá čas daleko rychleji. Proto pro pauzu a jiné určení času využívám funkci čítače. Ten po spuštění počítá hodinové impulzy procesoru a porovnává je s hodnotou uloženou v registrech T1MODH a T1MODL. Při shodě hodnoty čítače s hodnotou nastavenou v registrech T1MOD dojde k přetečení čítače a nastavení bitu 7 registru T1SC.

Čítač je třeba před prvním použitím vhodně nastavit. Při frekvenci 2,4576 MHz je vhodné použít předděličku, čímž bude čítač počítat impulzy s frekvencí danou poměrem hodinového taktu a předděličky. Např. při nastavení nejvyššího možného dělícího poměru, který je 64, bude čítač čítat s frekvencí 2457600/64 = 38400 Hz

Chceme-li například dosáhnout pauzy 1 vteřiny, což odpovídá frekvenci 1 Hz, musíme zajistit, aby k přetečení došlo při dosažení hodnoty 38400. Jelikož k přetečení dojde po přivedení N+1 impulzu, musíme nastavit hodnotu o 1 nižší. Tzn. 38399. To je v hexadecimálním kódu hodnota $95FF, kdy do registru T1MODH zapíšeme hodnotu $95 a do registru T1MODL hodnotu $FF.

Dále nastavíme dělící poměr předděličky tzn. 64 zapsáním hodnoty 110 do nejnižších tří bitů registru T1SC. Dále do tohoto registru zapíšeme dvě jedničky do čtvrtého a pátého bitu, čímž zajistíme reset čítače a jeho prozatímní zastavení. Vymazáním registrů T1SC0 a T1SC1 zakážeme všechny capture/compare funkce čítače.
Toto nastavení provádíme v podprogramu `init_citac`

Příklad zápisu:

```
init_citac:     mov     #$76,T1SC
                clr        T1SC0
                clr        T1SC1
                mov     #$95,T1MODH
                mov     #$FF,T1MODL
```

Vlastní inicializaci provedeme voláním podprogramu init_citac.

Příklad volání:

```
jsr   init_citac
```

Start čítače zajistíme voláním podprogramu `start_citace`. Ten nastaví pátý bit registru T1SC na nulu, což čítač spustí.

Příklad volání :

```
jsr   start_citace
```

Zastavení čítač provedeme voláním podprogramu `stop_citace`. Ten nastaví pátý bit registru T1SC na jedničku, čímž se čítač zastaví.

Příklad volání :

```
jsr   stop_citace
```

Případný reset čítače zajistíme voláním podprogramu `reset_citate`. Ten resetuje čítač nastavením čtvrtého bitu T1SC na jedničku.

Příklad volání :

```
jsr   stop_citace
```

Vlastní pauzu v době trvání dle nastavení registru T1MOD zajistíme voláním podprogramu `wait`. Ten bude tak dlouho probíhat ve smyčce skoku na „sebe sama“ dokud nedojde k přetečení čítače. Poté teprve pokračuje na další instrukci, která vynuluje příznak přetečení, abychom mohli čítač použít vícekrát.

Příklad volání:

```
jsr   wait
```

Podprogramy a jejich praktické použití obsahuje soubor `wait.asm`

3.2 LED diody

Kit obsahuje dvě LED diody. Ty rozsvítíme zapsáním logické nuly na bit portu PTB odpovídající dané LED diodě. Pro červenou diodu je to bit 5 a pro zelenou diodu je to bit 4 portu PTB.

Předtím je ale nutno nastavit tyto bity jako výstupy. To provedeme zapsáním logických jedniček do bitu 5, resp. 4 registru DDRB.

Inicializace LED diod se provádí podprogramem `init_LED`.

Příklad volání :

```
jsr   init_LED
```

K rozsvícení diody je zapotřebí zapsat logickou úroveň 0 na patřičný port bitu PTB. Toho dosáhneme voláním podprogramu `LED_1_on` pro rozsvícení červené diody, resp. `LED_2_on` pro rozsvícení zelené diody.

Příklad volání :

```
jsr   LED_1_on    ;rozsvítí červenou diodu
.
.
.
jsr   LED_2_on    ;rozsvítí zelenou diodu
```

Obdobně zhasnutí diody dosáhneme voláním podprogramu `LED_1_off` pro zhasnutí červené diody, resp. `LED_2_off` pro zhasnutí zelené diody. Tyto podprogramy zapíší logickou úroveň 1na patřičný bit portu PTB, čímž se diody zhasnou.

Příklad volání :

```
jsr   LED_1_off   ;zhasne červenou diodu
.
.
.
jsr   LED_2_off   ;zhasne zelenou diodu
```

Pokud chceme dosáhnout změny změny stavu diody, můžeme využít podprogram `LED_1_prepni` pro převrácení stavu červené diody, resp. `LED_2_prepni` pro převrácení stavu zelené diody.

Příklad volání :

```
jsr   LED_1_on    ;rozsvítí červenou diodu
jsr   LED_2_off   ;zhasne, popř. nechá zhasnutou
                   zelenou diodu
.
.
.
jsr   LED_1_prepni  ;změní stav červené diody
                     tzn. zhasne ji
jsr   LED_2_prepni  ;změní stav zelené diody tzn.
                     rozsvítí ji
```

Podprogramy lze nalézt v souboru `led.asm`

3.3 Klávesnice (Tlačítka SA1 až SA12)

Kit obsahuje 12+2 tlačítek. Jelikož jsou tlačítka SA13 a SA14 zapojena zcela odlišně a nezávisle na ostatních, jejich ovládání je vyčleněna do samostatné části. Klávesnice je připojena na porty PTC (sloupce) a PTD (řádky). Pro správnou funkci klávesnice je třeba ji nastavit. Aby klávesnice řádně fungovala, musíme připojit 30kΩ pullup rezistory. To se provede povolením přerušení od klávesnice. Jelikož ale obsluhu klávesnice řešíme ve smyčce podprogramu bez použití přerušení, zároveň toto přerušení maskujeme. Test stisknuté klávesy je prováděn pro každý sloupec zvlášť. A to tak, že zapíšeme logickou 0 na příslušný bit portu PTC (bity 4,5 a 6). Poté čteme hodnotu portu PTD. A pokud se na některém bitu portu PTD reprezentující řádky klávesnice (bity 4,5,6 a 7) objeví logická 0, znamená to, že došlo ke stisku některé z kláves. Pokud toto nastane, vložíme malou pauzu (v našem případě cca 10 ms) a testujeme znova, zda je stále logická nula na daném portu přítomna. Tím zamezíme vyhodnocení případných zákmitů tlačítka, daných konstrukcí tlačítka, jako stisku klávesy. Při stisku tlačítka dojde k zápisu hodnoty podle tabulky č. 1 do proměnné `klavesa`.

Pro zjednodušení se počítá s tím, že v každém okamžiku je stisknuto jen jedno tlačítko. Není-li tomu tak, je vyhodnocen pouze stisk jednoho tlačítka.

SA7 1	SA8 2	SA9 3
SA4 4	SA5 5	SA6 6
SA1 7	SA2 8	SA3 9
SA10 10	SA11 11	SA12 12

Tab. 1. Rozložení tlačítek klávesnice a hodnoty proměnné klavesa pro tyto tlačítka

Inicializaci klávesnice provádíme voláním podprogramu `init_KB`. Ten za nás provede výše uvedená nastavení.

Příklad volání :

```
jsr   init_KB
```

Vlastní zjištění stisknuté klávesy provádí podprogram `get_key`. Jelikož provádí jen jeden test, je vhodné volání tohoto podprogramu umístit do nějaké smyčky. Záleží ale na konkretním použití.

Příklad volání :

```
      mov   #0,klavesa
loop: jsr   get_key     ;zjištění klávesy
      .                 ;vyhodnocení stisknuté klávesy
      .                 ;a případné volání patřičných
                        podprogramů
      .
      mov   #0,klavesa  ;nulování proměnné
      bra   loop        ;skok na loop (nekonečná
                        smyčka)
```

Příslušné podprogramy obsahuje soubor `klavesy.asm`

3.4 Tlačítka SA13 a SA14

Jelikož tlačítka SA_13 a SA_14 jsou zapojeny odlišně od tlačítek klávesnice (SA1 až SA10) je jejich obsluha řešena samostatně v souboru `klavesy2.asm`

Protože tlačítka SA13 a SA14 jsou zapojeny na stejné vstupy jako LED diody nelze je využívat zároveň. Pokud je třeba využití tlačítek SA13 a SA14 ve stejné aplikaci jako diod LED, je třeba před každým použitím obě periferie znovu inicializovat.

Obdobně jako u LED diod je potřeba nastavit bity 4 a 5 portu PTB. V tomto případě ale je třeba nastavit je jako vstupy zapsáním logických nul na dané bity portu PTB. K tomu slouží podprogram `init_KB2`.

Příklad volání :

```
jsr   init_KB2
```

Podprogramem `get_key2` zjistíme, zda došlo ke stisku některého z tlačítek SA_13 a SA_14. Pokud je zjištěn stisk klávesy, je zařazena prodleva cca 10 ms a pak je stisk překontrolován. Opět tím předcházíme vyhodnocení případných zákmitů tlačítka jako stisku tlačítka. Při stisku tlačítka SA_13 bude do proměnné klavesa uložena hodnota 13 a při stisku tlačítka SA_14 bude do proměnné klavesa uložena hodnota 14.

3.5 Relé

Abychom mohli používat relé, musíme nejdříve nastavit druhý bit portu PTA jako výstup. To provedeme voláním podprogramu `init_RELE`. Ten zajistí zapsaní logické jedničky na 2 bit registru DDRA.

Příklad volání :

```
jsr   init_RELE
```

Samotné sepnutí relé nám zajistí podprogram `RELE_on`. Ten nastaví bit 2 portu PTA, což způsobí sepnutí relé

Příklad volání :

```
jsr   RELE_on
```

Obdobně vynulováním bitu 2 portu PTA, které provádí podprogram `RELE_off` způsobí rozepnutí relé.

Příklad volání :

```
jsr   RELE_off
```

Chceme li dosáhnout změnu stavu relé, použijme podprogram `RELE_prepni`.

Příklad volání :

```
jsr   RELE_on     ;sepne relé
.
.
.
jsr   RELE_prepni  ;změní stav relé, tzn. rozepne
                    jej
jsr   RELE_prepni  ;změní stav relé, tzn. sepne
                    jej
```

Všechny podprogramy obsahuje soubor `rele.asm`

3.6 Bzučák

Tón je na bzučáku generován střídáním logických úrovní jedna a nula. Frekvence těchto střídání určuje výsledný tón. Před použitím bzučáku jej musíme nastavit. To nám zajistí podprogram `init_BEEP`. Ten zapsáním logické nuly do bitu jedna registru DDRA nastaví na portu PTA bzučák jako výstup. Také nastaví registry T1MODH, T1MODL a T1SC. Jelikož se v tomto případě generuje relativně vysoká frekvence, nepoužíváme u čítače předdělicku. Protože v tomto podprogramu používáme čítač číslo jedna, který využíváme i pro společné zpožďovací funkce, je třeba zajistit, aby se před použitím tohoto čítače použil vždy správný inicializační podprogram.

Frekvence změn stavu se vypočítá podle vzorce : $\frac{f_B}{(2 \times f_{tónu})}$

Kde f_B je frekvence sběrnice v našem případě 2,4576 MHz. Vypočtená hodnota se zapíše do registru T1MOD. Základní stupnice tónů, jejich frekvence a příslušné hodnoty registru T1MODH a T1MODL jsou uvedena v tabulce č. 2.

Tón	***Frekvence [Hz]***	***T1MODH:TMODL***
c	262	$12:$52
d	294	$10:$54
e	330	$0E:$8C
f	349	$0D:$C1
g	392	$0C:$3F
a	440	$0A:$E9
h	494	$09:B7
c'	524	$09:29

Tab. 2. Frekvence tónů a příslušné hodnoty registru T1MODH:T1MODL

Příklad volání :

```
jsr   init_BEEP
```

Samostatný tón získáme voláním podprogramu `pipni`. Ten v pravidelných intervalech mění logickou úroveň bitu 1 portu PTA, čímž se generuje tón. Doba trvání tónu je dána hodnotou index registru H:X.

Příklad použití :

```
ldhx    #90
jsr     pipni
```

Obslužné podprogramy obsahuje soubor `beep.asm`

3.7 TERMISTOR

Obslužné podprogramy pro termistor obsahuje soubor `termistor.asm`. Jako teplo citlivý prvek je na kitu použit termistor NTC/22k . Termistor je zapojen na napěťovém děliči. Hodnotu napětí na termistoru, jež je závislá na teplotě, zjistíme pomocí A/D převodníku.

Použitý termistor NTC/22K :

katalogové číslo	2322 640 6322
teplotní rozsah	-40 ºC až 125 ºC
ztrátový výkon při 25 ºC	max. 0,5 W
odpor termistoru při 25 ºC	R_T = 25 kΩ ± 5 %
materiálová konstanta	$B_{25,85}$ = 3740 ± 2 %

Odpor termistoru R_T [Ω] při teplotě T [K] nám určuje vzorec : $R_T = R_{25} \times e^{B_{25,85} \times (1/T - 1/T_0)}$

Hodnoty proměnných `teplota1` a `teplota2`, jež nám udávají hodnotu úměrnou teplotě na termistoru nám určuje matematická funkce, kterou lze jen těžko realizovat bez matematické knihovny. Řešením by byla tabulka hodnot proměnných `teplota1`, resp. `teplota2`, které by udávaly skutečnou teplotu pro dané hodnoty. Toto řešení je ale při pokrytí celého rozsahu teplot termistoru poměrně hodně paměťově náročné, proto jsou výstupem podprogramu proměnné teplota1 a teplota2 pouze jako výsledek A/D převodu.

Před použitím podprogramu pro zjištění teploty na termistoru musíme nejdříve nastavit A/D převodník. Budeme využívat nepřetržitý převod, jako zdroj analogového signál bude vybrán port PTB6, jež je přiveden na kanál převodníku ADC4. Využíváme všech 10-bitu převodu, zarovnání vpravo. Toto se nastaví v registrech ADCLK a ADCSR.

Abychom dosáhli správného nastavení využijeme podprogram `init_AD`, který zapíše potřebné hodnoty do těchto registrů.

Příklad volání :

```
jsr   init_AD
```

Výsledek A/D převodu zjistíme voláním podprogramu `teplota`, který vrací v proměnné `teplota1` spodních 8 bitů 10-bitového převodu. Zbývající, nejvyšší dva bity jsou uloženy v bitech 0 a 1 proměnné `teplota2`.

Příklad volání :

```
lda   teplota1
.
.
jsr   teplota
cmp   teplota1    ;porovnání zda došlo ke změně
```

3.8 Sériová komunikace

Sériová komunikace kitu LJ12EVB může probíhat buď pomocí IR přenosu,kdy musí být přes konektor X2 připojen příslušný hardware, nebo pomocí rozhraní RS232 integrovaného na kitu. Přepínání těchto režimů se provádí pomocí propojek JP4 a JP5 na kitu. My budeme používat rozhraní RS232 proto musí být tyto propojky ve stavu 1.

Pro správnou funkci je třeba nastavit vhodnou přenosovou rychlost. Toto nastavení se provádí v registru SCBR. V našem případě jsme zvolili přenosovou rychlost 9600 bps. Jelikož je toto nastavení závisle na vnitřním taktu kitu, při použití jiného zdroje signálu, než námi použitého kitu JANUS je třeba toto nastavení upravit podle dané frekvence.

Toto nastavení zajistíme voláním podprogramu `init_SCI`. Tento podprogram zajistí výše uvedené nastavení přenosové rychlosti. Dále spustí vysílač a přijímač a zakáže přerušení.

Příklad volání : `jsr   init_SCI`

Pro vyslání znaku použijeme podprogram `odeslani` před voláním tohoto podprogramu je vhodné nastavit registr H:X, který je použit pro nastavení maximální doby odesílání. Pokud nebude registr H:X nastaven, použije se maximální doba pro odeslání. Odeslán bude znak uložený v proměnné `znakV`. Odeslání bude provedeno pouze pokud bude v časovém limitu daném registrem H:X uvolněn vysílač po předchozím přenosu.

Příklad volání :

```
mov     #'A',znakV  ;uloží znak do proměnné
ldhx  #55           ;nastaví max čekání
jsr   odeslani
```

Obdobně pro příjem znaku použijeme podprogram `prijem`. Pro nastavení maximální doby odesílání je opět použit registr H:X. Přijatý znak bude uložen do proměnné `znakP`. K odeslání znaku dojde opět za předpokladu, že v době než vyprší doba určená registrem H:X, bude uvolněn vysílač po předchozím přenosu.

Příklad volání :
```
ldhx  #55
jsr   prijem
```

Obslužné podprogramy obsahuje soubor `prenos.asm`

3.9 LCD display

Na kitu LJ12EVB je zapojen LCD display umožňující zobrazit 8 znaků pomocí 8 sedmi segmentových modulů a 25 symbolů.

Abychom mohli zobrazovat všechny znaky a symboly je potřeba v registru CONFIG2 nastavit bit PCEL na logickou jedničku. Tím se bity 0 až 3 portu PTC použijí pro potřeby LCD displaye. V registru LCDCLK je třeba nastavit mód ¼ duty cycle a takt zobrazovače. A v registru LCDCR je potřeba spustit LCD modul a nastavit potřebný kontrast. Jelikož lze jen těžko nastavit univerzálně jak kontrast, tak takt zobrazovače, je námi použité nastavení kompromisem mezi možnými nastaveními.

Samostatné zobrazení je dosaženo zapsáním logické úrovně 1 na patřičný bit registrů LDAT1 až LDAT14. K zjištění patřičného bitu registru je potřeba použít příloha P 4 určující popis segmentů a symbolů a příloha P 5, kde jsou rozepsány patřičné bity v adresní mapě registrů LCD displaye.

Nastavení LCD displaye nám usnadní podprogram `init_LCD`, jež zajistí zapsání patřičných hodnot do daných registrů a vymazání obsahu registrů LDAT1 až LDAT14, čímž dojde ke smazání případných znaků zobrazených na display.

Příklad volání :
```
jsr   init_LCD
```

Pro smazání obsahu LCD displaye je možno kdykoliv použít podprogram `clear_LCD`.

Příklad volání :
```
jsr   clear_LCD
```

Chceme-li zobrazit pouze grafické symboly, můžeme využít podprogram `zobraz_symboly`. V proměnných `tgX`, `tgH`, `tgK` a `tg` nastavíme patřičné bity těchto proměnných podle tabulky č. 3 a přílohy č.

Logická úroveň 1 jednotlivých bitů těchto proměnných při volání podprogramu zobraz_symboly zajistí jejich zobrazení na LCD display.

proměnná	*bit 7*	*bit 6*	*bit 5*	*bit 4*	*bit 3*	*bit 2*	*bit 1*	*bit 0*
tgH	H7	H6	H5	H4	H3	H2	H1	N/A
tgK	K7	K6	K5	K4	K3	K2	K1	N/A
tgX	N/A	X6	X5	X4	X3	X2	X1	N/A
tg	N/A	N/A	Y2	Y1	P3	P2	P1	N/A

Tab. 3. Význam jednotlivých bitů proměnných ovládajících grafické symboly

Příklad volání :

```
mov   #%10110010,tgK
jsr   zobraz_symboly
```

Pro zobrazení čísel na sedmi-segmentových modulech displaye použijeme podprogram `zobraz_znaky`. Ten zajistí smazání LCD displaye, zobrazení číslic a zobrazení grafických symbolů.

Zobrazení grafických symbolů je řízeno proměnnými tgH, tgK, tgX a tg uvedenými v tabulce č 3. Viz výše.

Zobrazení číslic se řídí proměnnými seg1, seg2, seg3 a seg4. V každé proměnné je uloženo číslo k zobrazení na dvou modulech LCD displaye. Jednotlivé moduly a jejich rozložení v daných proměnných je popsáno v tabulce č. 4. Umístění modulů lze nalézt v příloze P 3Např. v proměnné seg1 je na bitech 0 až 3 uloženo číslo jež bude zobrazenu na modulu 2 a na bitech 4 až 7 číslo jež bude zobrazeno na modulu 1.

proměnná	*bit 7*	*bit 6*	*bit 5*	*bit 4*	*bit 3*	*bit 2*	*bit 1*	*bit 0*
seg1	Modul 1				Modul 2			
seg2	Modul 3				Modul 4			
seg3	Modul 5				Modul 6			
seg4	Modul 8				Modul 7			

Tab. 4. Rozpis modulů LCD displaye v ovládacích proměnných

Podprogram `zobraz_znaky` převede číslo uložené v proměnné pro každý modul tak, aby jej bylo možno zobrazit na sedmi-segmentovém zobrazovači. Tento převod je funkční jen pro číslice, jelikož písmena na tomto zobrazovači není možno zobrazit.

Nechceme-li zobrazovat některý modul displaye, můžeme použít proměnnou `tgS`. Jednotlivé bity této proměnné povolují a zakazují zobrazení jednotlivých modulů LCD. Logická úroveň 0 zakáže zobrazení daného segmentu.(Tab. 5)

proměnná	***bit 7***	***bit 6***	***bit 5***	***bit 4***	***bit 3***	***bit 2***	***bit 1***	***bit 0***
tgS	Modul 8	Modul 7	Modul 6	Modul 5	Modul 4	Modul 3	Modul 2	Modul 1

Tab. 5. Význam bitů pro ovládání modulů LCD displaye

Příklad volání :

```
mov     #$42,seg3
mov     #$17,seg2
mov     #%01110000,tgX
mov     #%01011100,tgS
mov     #0,tgK
jsr     zobraz_znaky
```

Obslužné programy pro obsluhu LCD obsahuje soubor `lcd.asm`

4 DEMONSTRAČNÍ PROGRAM

Pro ukázku použití vytvořených podprogramů slouží demonstrační program `demo.asm`. V tomto programu jsou na ukázku použity podprogramy pro obsluhu klávesnice, LCD displaye, relé, LED diod a bzučáku.

Program po spuštění, inicializaci periferií a nastavení počátečních stavů LED diod (podprogramy `LED_1_on`, `LED_2_off`), čeká na stisknutí klávesy (podprogram `get_key`). Je li stisknuta klávesa 1 až 9 (podle Tab. 1) dojde k přesunu z proměnné `klavesa` do proměnné `seg2`, a pomocí podprogramu `zobraz_znaky` k zobrazení hodnoty zmáčknuté klávesy na čtvrtém sedmi-segmentovém modulu LCD displaye. Ostatní moduly jsou pomocí proměnné `tgS` zhasnuté. Dojde-li ke zmáčknutí klávesy 10, pomocí podprogramu `RELE_prepni` dojde k překlopení stavu relé. Aby nedocházelo k rychlému opakování stavu, je vložena přibližně vteřinová mezera pomocí čítače. Dojde-li ke zmáčknutí klávesy 11, pomocí podprogramu `LED_1_prepni` a `LED_2_prepni` se překlopí stavy LED diod. Opět je zařazena malá pauza. A konečně dojde-li ke stisku klávesy 12 rozezní se na chvíli pomocí podprogramu `pipni` bzučák.

Celý program je velmi jednoduché složit z obslužných podprogramů pro jednotlivé periferie. Vlastní tělo programu je díky tomu zkráceno na minimum a jeho tvorba nevyžaduje další znalosti ovládání konkrétních periferií.

ZÁVĚR

Cílem této práce bylo prostudování hardwarových vlastností kitu LJ12EVB firmy Beta Control, vytvoření programového vybavení pro obsluhu periferií, tvorba dokumentace k tomuto softwarovému vybavení a demonstrace práce s nimi v demonstračním programu.

Nejprve byl popsán samotný kit LJ12EVB. A to konkrétně jeho centrální procesorová jednotka, její základní vlastnosti a vlastnosti a vybavení kitu periferiemi. Poté byla popsána možnost propojení kity s PC a základy tvorby aplikací pomoci vývojového prostředí WinIDE.

V další části byly vytvořeny obslužné podprogramy pro obsluhu jednotlivých periferií kitu, popsána jejich funkce a příklady jejich použití. Práce s těmito podprogramy a jejich funkčnost byla ověřena při vytváření demonstračního podprogramu.

SEZNAM POUŽITÉ LITERATURY

[1] 1: Beta Control s.r.o. , Starter Kit LJ12EVB User's Manual, 2002

[2] 2: Motorola Inc. , CPU08 Central Processor Unit Reference manual, 2001

[3] 3: Motorola Inc. , MC68HC908LJ12 Technical Data, 2002

[4] 4: NÁPRSTEK Jiří, Vývojový kit JANUS Uživatelský manuál, 2003

[5] 5: P&E Microcomputer systems. , M68ICS08 68HC08 In-circuit simulator Operators Manual, 2000

[6] VÁŇA,Vladimír. *Začínáme s mikrokontroléry HC08 NITRON*. . Praha : BEN - Technická literatura, 2003. ISBN 80-7300-124-1

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

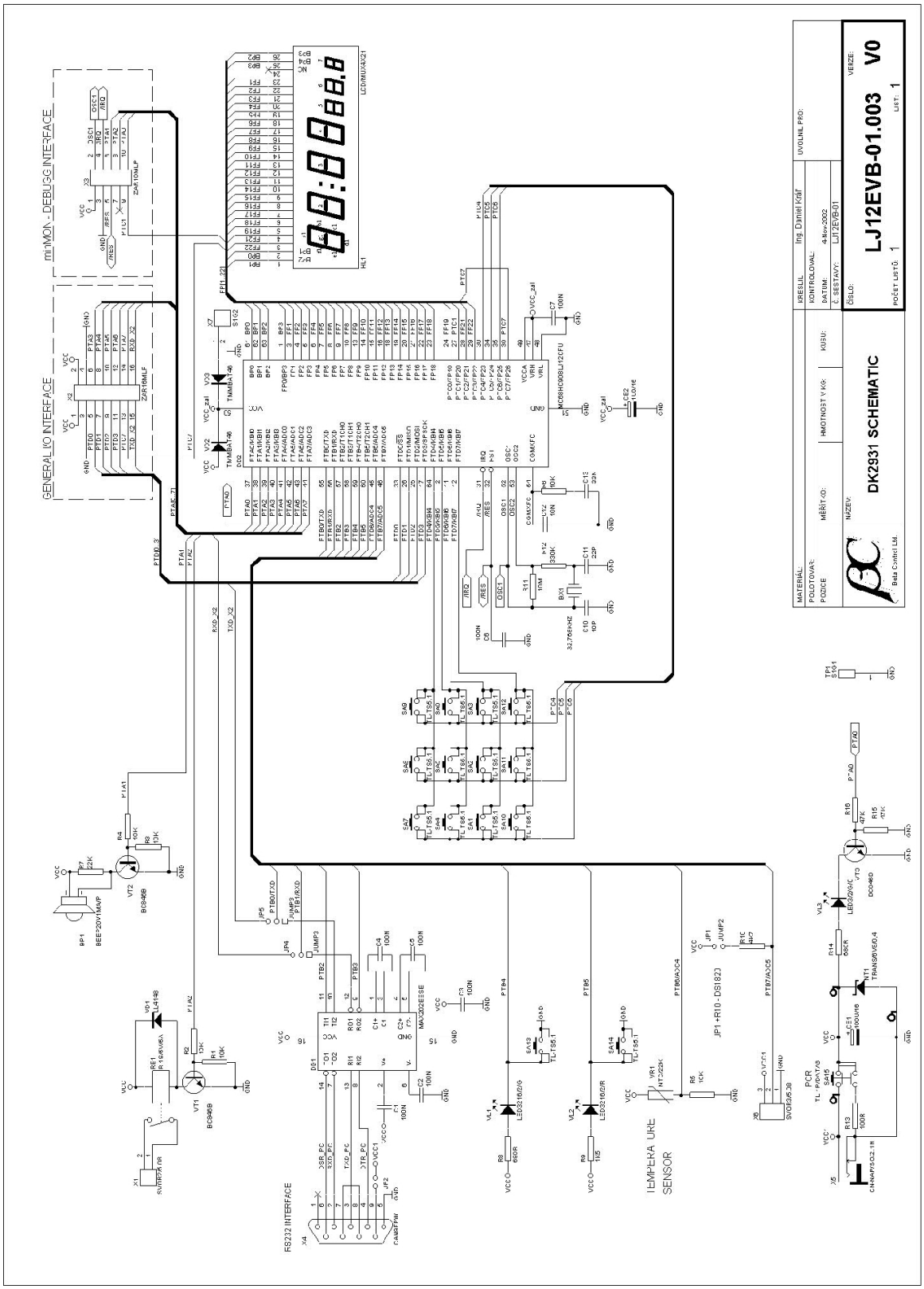
A/D	Analogově digitální
CPU	Centrální procesní jednotka
EEPROM	Elektricky mazatelná a programovatelná paměť
FLASH	Elektricky mazatelná a programovatelná paměť – rychlejší než EEPROM
LED	Světlo vyzařující dioda
LCD	Display z tekutých krystalů
PC	Osobní počítač
RAM	Paměť s libovolným přístupem

SEZNAM OBRÁZKŮ

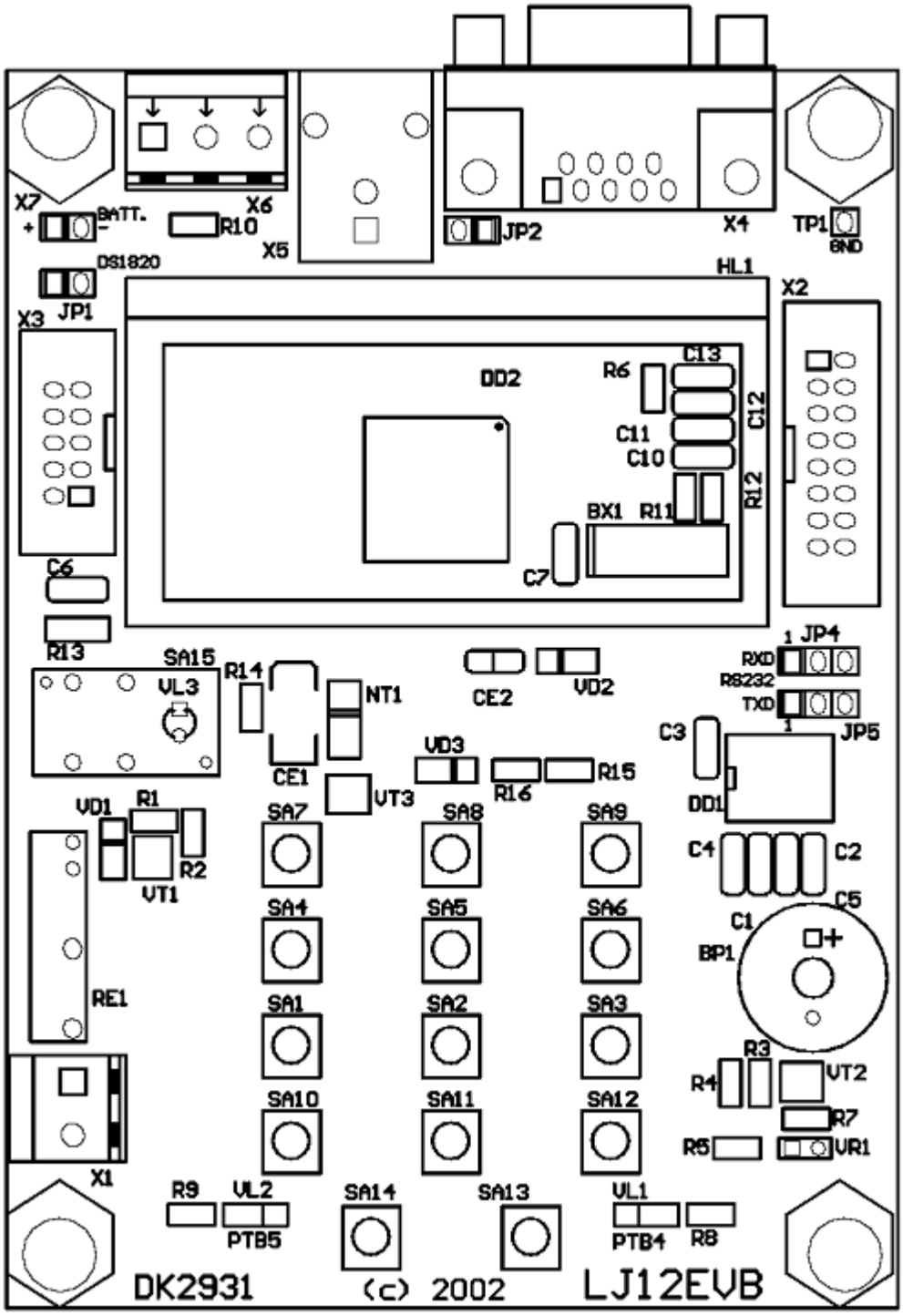
SEZNAM TABULEK

SEZNAM PŘÍLOH

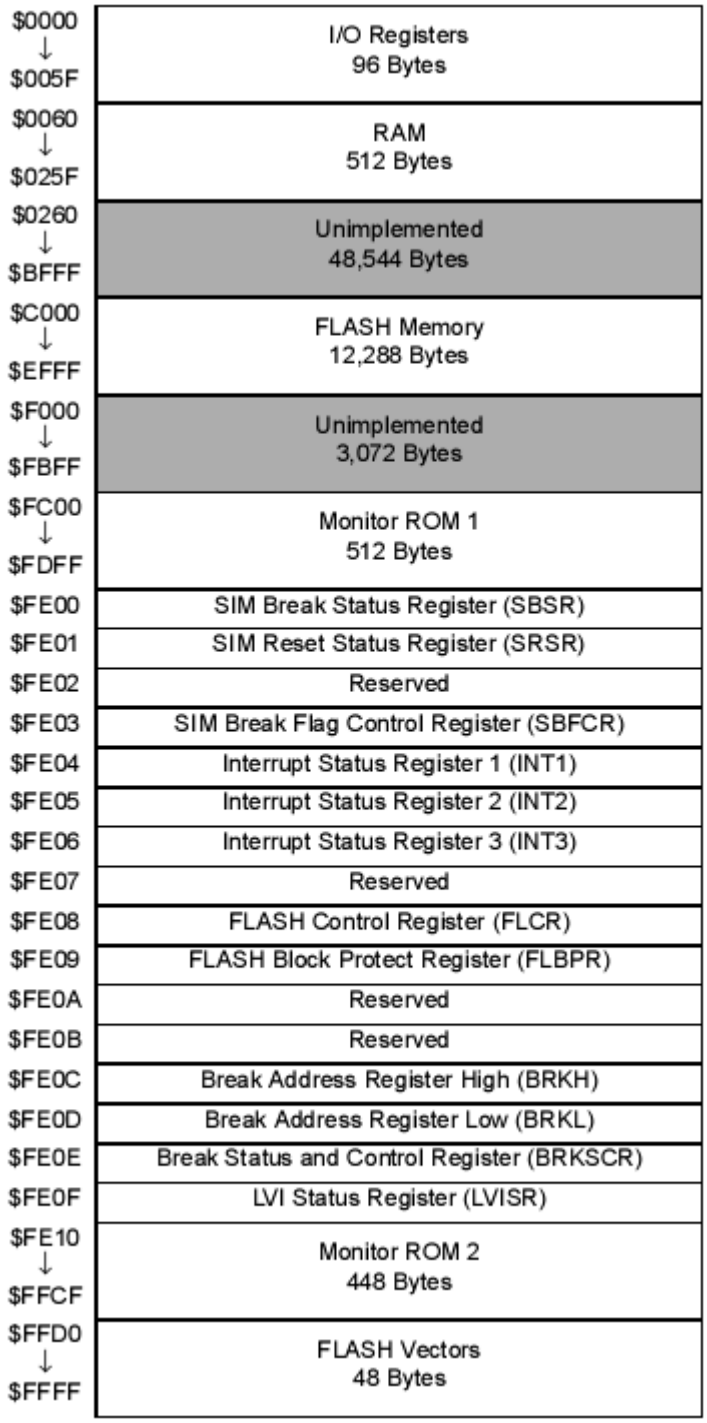
PŘÍLOHA P 1: SCHÉMA ZAPOJENÍ KITU LJ12EVB



PŘÍLOHA P 2: ROZMÍSTĚNÍ SOUČÁSTEK



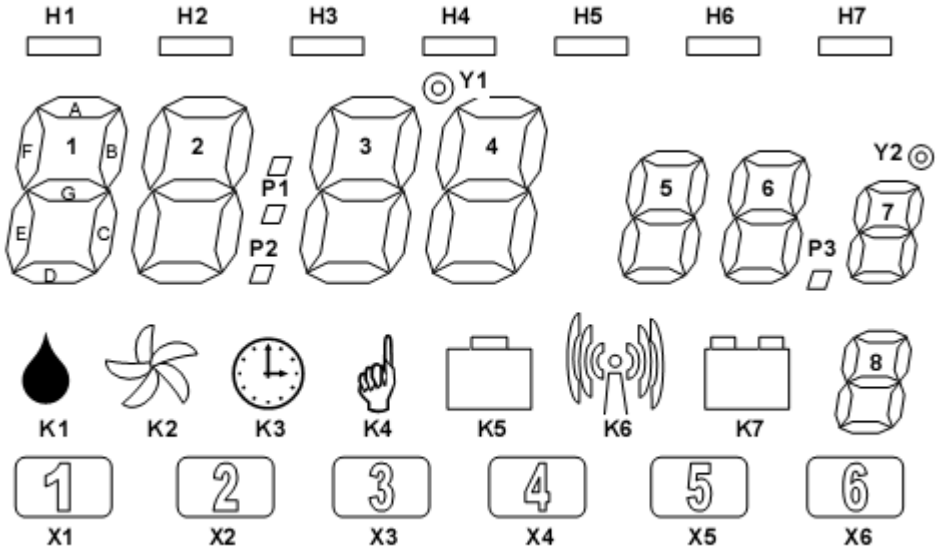
PŘÍLOHA P 3: MAPA PAMĚTI



PŘÍLOHA P 4: TABULKA SIGNÁLŮ A ROZMÍSTĚNÍ SEGMENTŮ . LCD DISPLAYE

BP1	BP0	FP22	FP21	FP19	FP18	FP17	FP16	FP15	FP14	FP13	FP12	FP11
1	2	3	4	5	6	7	8	9	10	11	12	13
	BP0	1F	1A	1B	2F	2A	2B	3F	3A	3B	4F	4A
BP1		1E	1G	1C	2E	2G	2C	3E	3G	3C	4E	4G
		K1	1D	K2	K3	2D	P1	K4	3D	K5	K6	4D
		X1	H1	X2	X3	H2	P2	X4	H3	X5	Y1	H4

FP10	FP9	FP8	FP7	FP6	FP5	FP4	FP3	FP2	FP1		BP3	BP2
14	15	16	17	18	19	20	21	22	23	24	25	26
4B	5F	5A	6F	6A	7F	7A	Y2	8F	8A			
4C	5G	5B	6G	6B	7G	7B		8G	8B			
K7	5E	5C	6E	6C	7E	7C		8E	8C			BP2
X6	5D	H5	6D	P3	7D	H6		8D	H7		BP3	



PŘÍLOHA P 5: REGISTRY PRO PRÁCI S LCD DISPLAYEM

Addr.	Register Name		Bit 7	6	5	4	3	2	1	Bit 0
$004F	LCD Clock Register (LCDCLK)	Read:	0	FCCTL1	FCCTL0	DUTY1	DUTY0	LCLK2	LCLK1	LCLK0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0051	LCD Control Register (LCDCR)	Read:	LCDE	0	FC	LC	LCCON3	LCCON2	LCCON1	LCCON0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$0052	LCD Data Register 1 (LDAT1)	Read:	F1B3	F1B2	F1B1	F1B0	F0B3	F0B2	F0B1	F0B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0053	LCD Data Register 2 (LDAT2)	Read:	F3B3	F3B2	F3B1	F3B0	F2B3	F2B2	F2B1	F2B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0054	LCD Data Register 3 (LDAT3)	Read:	F5B3	F5B2	F5B1	F5B0	F4B3	F4B2	F4B1	F4B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0055	LCD Data Register 4 (LDAT4)	Read:	F7B3	F7B2	F7B1	F7B0	F6B3	F6B2	F6B1	F6B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0056	LCD Data Register 5 (LDAT5)	Read:	F9B3	F9B2	F9B1	F9B0	F8B3	F8B2	F8B1	F8B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0057	LCD Data Register 6 (LDAT6)	Read:	F11B3	F11B2	F11B1	F11B0	F10B3	F10B2	F10B1	F10B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0058	LCD Data Register 7 (LDAT7)	Read:	F13B3	F13B2	F13B1	F13B0	F12B3	F12B2	F12B1	F12B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$0059	LCD Data Register 8 (LDAT8)	Read:	F15B3	F15B2	F15B1	F15B0	F14B3	F14B2	F14B1	F14B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U

U = Unaffected = Unimplemented

$005A	LCD Data Register 9 (LDAT9)	Read:	F17B3	F17B2	F17B1	F17B0	F16B3	F16B2	F16B1	F16B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$005B	LCD Data Register 10 (LDAT10)	Read:	F19B3	F19B2	F19B1	F19B0	F18B3	F18B2	F18B1	F18B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$005C	LCD Data Register 11 (LDAT11)	Read:	F21B3	F21B2	F21B1	F21B0	F20B3	F20B2	F20B1	F20B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$005D	LCD Data Register 12 (LDAT12)	Read:	F23B3	F23B2	F23B1	F23B0	F22B3	F22B2	F22B1	F22B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$005E	LCD Data Register 13 (LDAT13)	Read:	F25B3	F25B2	F25B1	F25B0	F24B3	F24B2	F24B1	F24B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$005F	LCD Data Register 14 (LDAT14)	Read:					F26B3	F26B2	F26B1	F26B0
		Write:								
		Reset:	U	U	U	U	U	U	U	U

U = Unaffected = Unimplemented

PŘÍLOHA P 6: VÝVOJOVÝ DIAGRAM PROGRAMU DEMO.ASM