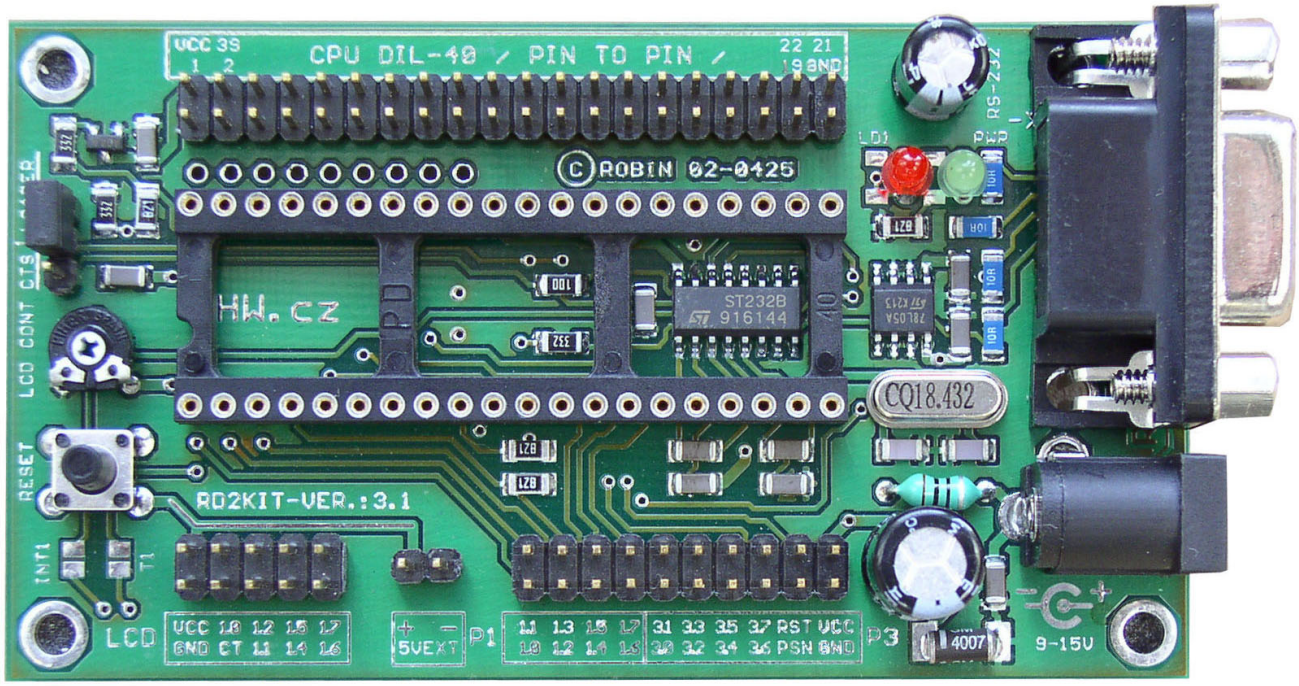
RD2 Kit

Naučte se programovat jednočipové mikroprocesory (MCU) nejen z rodiny x51 s pomocí 2x 40 řešených příkladů v jazyku C.



Starting guide - Než začnete pracovat s RD2 Kitem

Nainstalujte si software :

Dřív než začnete pracovat s kitem, měli by jste mít nainstalován software splňující následující potřeby :

- Internetový prohlížeč (MSIE, NN, Mozzila nebo další ..)
- Prohlížeč .PDF souborů (například **acrobat_reader_500enu.exe** z adresáře /sw/)
  Katalogové listy k procesoru a některé další (LCD displej..) najdete v adresáři Datasheets.
- Terminál pro přístup na sériový port RS232 (vyhoví **Teraterm.zip** nebo **setup_serial_term.zip** z adresáře /sw/, případně jiný ale preferujte takový, který lze stiskem jednoho tlačítka odpojit od portu a znovu jej pak k portu připojit..
- Pokud nebudete používat náš RD2 flasher ovládaný z příkazové řádky, doporučujeme hlavně začátečníkům nainstalovat si **Atmel Flip** (/Flasher/Atmel_Flip/dev_tools3bc6c0cebce3f.zip) který se snadno ovládá z prostředí Windows. Neumí však programovat interní EEPROM procesoru (viz. dále).
- Nainstalujte si demonstrační verzi **Keil µVision2** (najdete ji na druhém CD, které doporučujeme také později projít, protože i zde je mnoho užitečného) případně **SDCC (**podrobný popis instalace SDCC najdete v souboru **sdcc.install.txt** v adresáři **/SDCC/**), ideálně obojí..

Připojte LCD displej

RD2 Kit v řadě příkladů počítá s připojením LCD displeje se standardním řadičem kompatibilním s Hitachi HD44780. Pokud jste si tento displej koupili, připojte jej dále popsaným způsobem a otestujte nahráním některého z příkladů 2.x (strana 33).

```
* Target system : RD2 KIT - Version 3.1                                  *
*                 T89c51RD2 18,4320 MHz, UART: 19200,N,8,1              *
*                 LCD SC1602A - 4.bit mode                              *
* Pin assignment:                                                       *
*  LCD SC1602A - 4.bit mode                  T89c51RD2 - KIT LCD        *
*     1  GND                                    GND                     *
*     2  Vcc                                    Vcc                     *
*     3  V0  (contrast)                         CT                      *
*     4  RS  (register selection)               P1^0                    *
*     5  RW  (read/write)                       P1^1                    *
*     6  EN  (chip enable)                      P1^2                    *
*     7  DB0 (data bit 0)                       not connected           *
*     8  DB1 (data bit 1)                       not connected           *
*     9  DB2 (data bit 2)                       not connected           *
*     10 DB3 (data bit 3)                       not connected           *
*     11 DB4 (data bit 4)                       P1^4                    *
*     12 DB5 (data bit 5)                       P1^5                    *
*     13 DB6 (data bit 6)                       P1^6                    *
*     14 DB7 (data bit 7)                       P1^7                    *
*     15 Vcc (backlight Vcc)                    not connected           *
*     16 GND (backlight GND)                    not connected
```

LCD displeje 1x16, 2c16 nebo 1x20 mají všechny prakticky stejný interface. Liší se prakticky pouze implementovanou znakovou sadou pro horních 128 znaků (Ruská / Anglická / Japonská), mechanickými parametry, umístěním konektoru a polaritou případného LED podsvícení.

Obecně jsou displeje typu A užší a konektor je vlevo nahoře, typ B má širší plošný spoj a konektor vlevo dole (zleva = `D7|D6|...|D0|En|Rw|Rs|V0|+|-|LED-|LED+`)

Otestujte funkčnost HW

- Připojte RD2 Kit pomocí prodlužovacího kabelu k cannon 9 k RS232 vašeho PC. Pusťte si terminálový program a připojte se na tento port (v nejhorším postačí HyperTerminál, ale to je ta nejhorší volba z možných).
- Připojte ze síťového adaptéru cca 9-20V (přepólování nevadí, ale kit nefunguje) – rozsvítí se zelená LED.
- Zkontrolujte pozici troj-jumperu vlevo, nad trimrem pro ovládání kontrastu LCD displeje. Pro funkci nahraného programu musí být jumper v poloze LOADER nebo musí být jumper jako takový rozpojený.
- Nastavte si v terminálovém programu parametry sériového portu na 19200 8N1
  Stiskněte tlačítko RESET na RD2 Kitu. Standardně je kit dodáván s příkladem 1.1 – komunikace po RS232. V terminálu se vám proto musí zobrazit následující :

```
T89c51RD2 RS232 - PART I. ver 0.1.0 Sep 07 2002, 13:21:30
+---=[ RS232 Demo Application ver 0.1.0 (C) 2002 by www.HW.cz ]=---+
|                                                                  |
| Elementary functions needed for asynchronous communication       |
| between Atmel T89c51RD2 CPU and external devices - 19200,N,8,1.  |
|                                                                  |
+ command -+- function ---------------------------------------------+
|    a     | ASCII table (only characters >= 32)                   |
|    s     | display strings stored in RAM and ROM                 |
|    t     | terminal emulation                                    |
+-------------------------------------------------------------------+
```

  Nyní již komunikujete s mikroprocesorem RD2 Kitu, jak je popsáno v popisu příkladu 1.1 a hlavně v jeho zdrojovém kódu (**Examples/rd2.src.keil/01.01.RS232/main.c**)

Vytvořte a spusťte program BLIK

- Nainstalujte si prostředí **Keil µVision2** (doporučujeme shlédnout i demonstrační presentace) a přečtěte si popis na straně 26.
- Vytvořte LED.hex pomocí podrobného popisu na straně 24-25.
- Odpojte případný terminál ze sériového portu a spusťte si program **Atmel Flip**
- Před volbou „**Connect**“ v programu Flip ověřte napájení RD2 Kitu a pozici jumperu JMP3 v pozici „**LOADER**“, případně jumper nastavte a RD2 Kit resetujte.
- Pokud je vše OK, Flip se připojí a stačí vybrat soubor a naprogramovat jej do RD2 Kitu.
- Potom odpojte Flip z sériového portu, nastavte JMP3 na CTS nebo jej odpojte úplně a resetujte RD2 Kit.
- Pokud jste program napsali přesně podle návodu, měla by červená LED blikat s přibližně půlsekundovým intervalem.

RD2 Kit - C Programming

Umíte ASM – začněte programovat i v C

Pro vývoj aplikací v C lze použít řadu platforem, vývojových kitů atd.. Vzhledem ke specifické situaci v ČR, kde je stále ještě většina běžných vývojářů používá ASM a cca 50% lidí programuje x51 kompatibilní procesory jsme připravili dále popsaný vývojový kit, který je cílen především na usnadnění migrace od assembleru k jazyku C pro x51 procesory.

Čím se liší projekt RD2 Kit :

- Jednoduchý HW, který je snadno a levně dostupný.
- Použitý CPU RD2 zajišťuje funkční jednočipové řešení C programu.
- Cca 40 příkladů, které demonstrují základní funkce a rychle Vás vtáhnou do problematiky.
- Všechny příklady jsou k dispozici pro Keil C i SDCC které je k dispozici zdarma.
- Aplikace nejsou omezeny pouze na jedno C konkrétního výrobce.
- Součástí dokumentace je i popis rozdílů jednotlivých kompilerů C.

Pokud vás tedy dále popsaný RD2 Kit zaujal, můžete si jej vyrobit sami, podle popsaných schémat a návodů, nebo si jej můžete objednat z HW serveru.

Vývojový kit s procesorem T89c51RD2

RD2 Kit je jednoduchý a levný vývojový prostředek pro rychlý vývoj nových aplikací a výuku programování v jazyce C. Kit je osazen jednočipovým mikropočítačem T89c51RD2 a základními periferiemi. Programování interní paměti FLASH a EEPROM v procesoru je řešeno ISP programováním. U procesorů RD2 je programování v aplikaci (In Systém Programming) řešeno malým programem (loader) který je umístěn na posledním 1 kB kódu programu. takže na aplikace zbývá 63 kB (0000h – FC00h). Loader je spuštěn skokem na nastavenou adresu FC00h, pokud je při resetu nastavena určitá kombinace na pinech MOVC, ALE a hlavně PSEN=0V. V RD2 Kitu je tak procesor RD2 programován prostým přizemněním pinu PSEN a následným resetem. V CPU spuštěný program loader se po sériové lince RS232 dohodne s programovacím programem v PC „Flasher“, ten mu pošle program pro jednočipový procesor v HEX formátu a loader tento program naprogramuje do spodních 63 kB interní FLASH procesoru RD2. Námi dodávaný loader a flasher je kompatibilní s originální a navíc umí i obsluhovat interní EEPROM procesoru.

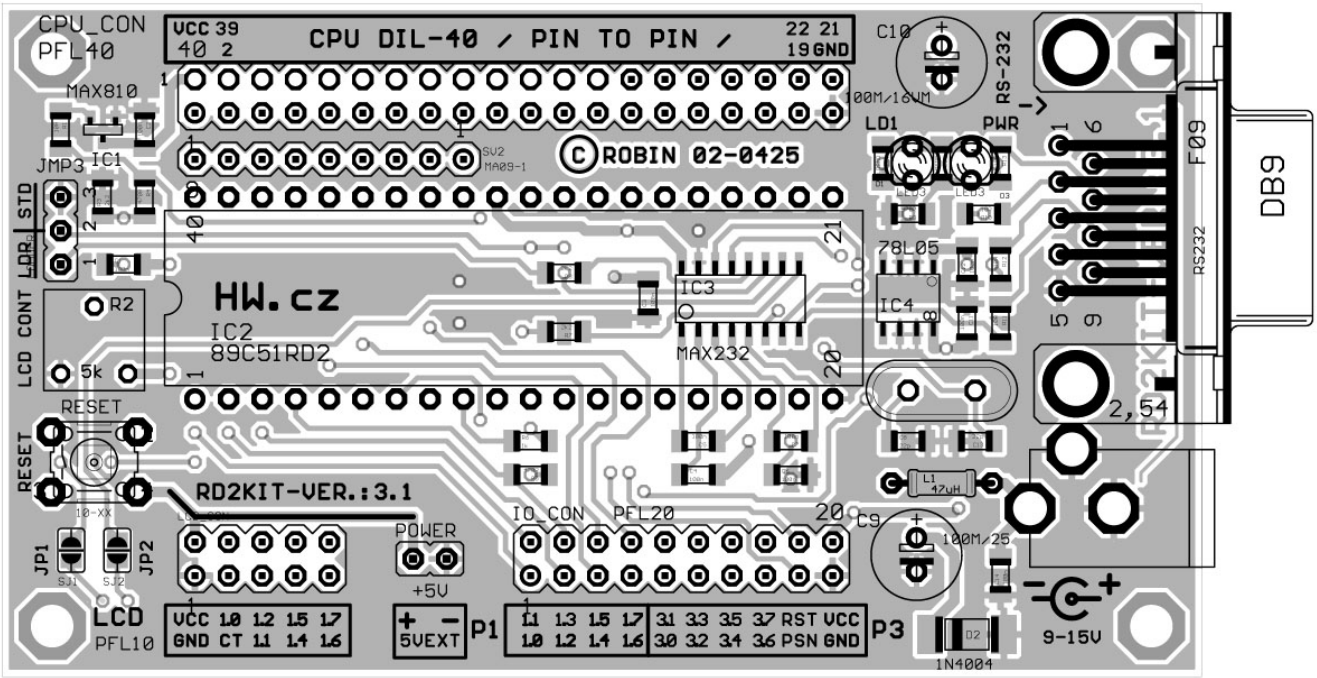
Pro práci s RD2 Kitem tak nepotřebujete žádný programátor, ani spoustu kabelů na stole. Stačí vám PC se sériovým portem RS232, editor a jeden z popsaných C kompilerů, napáječ a RD2 Kit.

Základní parametry RD2-kitu

- Mikroprocesor T89c51RD2 v pouzdře DIL, Xtal 18,4352 MHz.
- Indikační led dioda - P1.3
- RS232 rozhraní s převodníkem úrovní TTL/RS232.
- Jumper k aktivaci ISP režimu programování.
- Power-reset obvod + RESET tlačítko.
- Stabilizátor napájecího napětí (9-15 V, 200 mA).
- 10-pinový LCD konektor pro připojení displeje + nastavení kontrastu na RD2 Kitu.
- 20-pinový univerzální I/O konektor s vyvedenými bránami P1 a P3.
- 40-pinový expanzní konektor 1:1 s vývody procesoru (pro připojení RAM..).
- Rozměry plošného spoje kitu 90 x 50 mm.

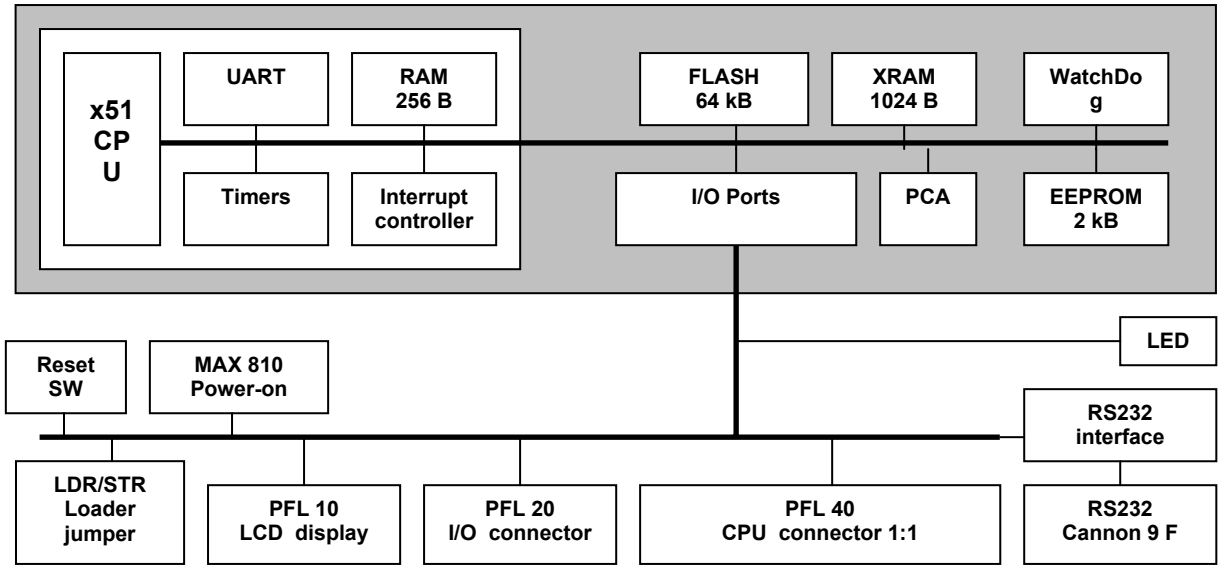
Pouzdro DIL u Jednočipového Mikroprocesoru (dále jenom JM) bylo zvoleno pro možnost náhrady za jiný pinově kompatibilní CPU a pro snadné programování obvodů v pouzdře DIL 40. Celý RD2-kit byl od počátku koncipován jako prostředek určený pro výuku a vývoj nových aplikací. Pokud vystane nutnost naprogramovat JM v klasickém programátoru není třeba speciálních konvertorů pouzder.

Popis osazeni RD2-kitu



Blokové schéma RD2-kitu

T89c51RD2 CPU



T89c51RD2

Procesor T89c51RD2 je zpětně kompatibilní s klasickou rodinnou 8051, kterou rozšiřuje o :

- 64 kB FLASH paměť na čipu programovatelnou metodou ISP a IAP s uživatelským zavaděčem,
- 2 kB EEPROM pro 100 000 zápisových cyklů,
- 1 kB interní paměti RAM,
- softwarové přepínání násobičky hodin, 6 a 12 hodinových taktů na cyklus (X2 - Mode),
- max. kmitočet 20 Mhz (6 taktů/cyklus) nebo 40 Mhz (12 taktů/cyklus),
- 7 zdrojů přerušení ve 4 úrovních priorit,
- podpora režimů se sníženou spotřebou,
- programovatelný výstup hodin,
- čítačem podporované programovatelné pole PCA,
- druhý ukazatel dat DPTR,
- možnost vypnutí signálu ALE,
- hardwarový watchdog.

Podrobnější informace o jednotlivých periferiích a jejich obsluze lze získat z katalogového listu firmy Atmel na adrese http://www.atmel.com/ nebo z katalogov0ho listu na CD RD2 Kitu.

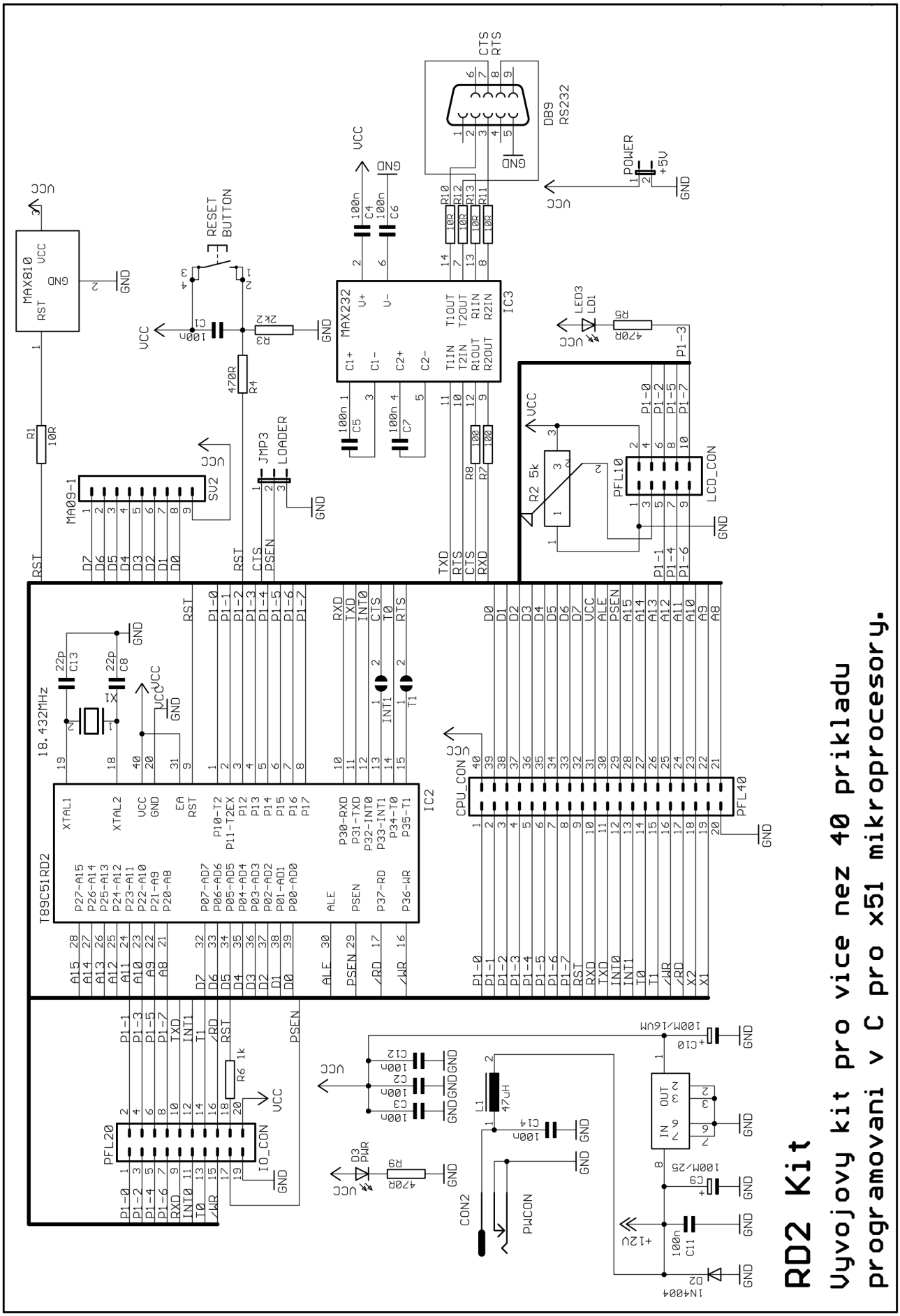
Procesoru jako takovému je věnován samostatný článek na CD.

Napájení RD2-kitu

K napájení kitu je nutno použít externí zdroj stejnosměrného napětí (nemusí být stabilizovaný). Doporučený rozsah výstupního napětí zdroje je 9 až 15 V při maximálním odběru asi 200 mA (odběr závisí na konkrétní aplikaci). Napájecí část kitu nemá ochranu proti přepětí, a proto maximální napětí zdroje je 15 V. Při překročení této hodnoty může dojít k poškození nebo zničení kitu. Kit je napájen z souosého konektoru (+ uprostřed) o průměru 2,5 mm do stabilizátoru 78L05 (5 V) s filtračními a blokovacími kondenzátory. Napájení 5V je přístupné na všech konektorech kitu.
Spotřeba kitu se pohybuje v jednotkách rozsahu (10 - 20 mA) a další odběr je tvořen použitými výstupy. Pokud chceme přímo z vyvedených výstupů JM budit vnější součástky, je nutno dát pozor na maximální přípustný odběr výstupních pinů uvedený v katalogovém listu (cca 10mA proti Vcc a cca 1mA proti GND). Přítomnost napájení je signalizována zelenou led diodou D3 - PWR.

Procesor

Reset (nulování) procesoru provádí integrovaný power-on reset MAX810, výstup. Oscilátor tvoří krystal Q1 a kondenzátory C1 a C2. Použitý krystal v miniaturním provedení HC49U/S je zaletován, ale lze jej snadno nahradit paticí a pak si vyměňovat krystaly podle potřeby. Při změně kmitočtu krystalu není třeba provádět žádné hardwarové zásahy do kitu, nezapomínejte však změnit konstanty ve vašem programu, hlavně pokud používáte sériovou linku RS232. Loader a Flasher funguje prakticky se všemi krystaly, protože se po startu synchronizuje,


RD2 Kit
Vyvojovy kit pro vice nez 40 prikladu
programovani v C pro x51 mikroprocesory.

Popis konektorů a přípojných míst

Komunikace s okolními periferiemi je realizována celou řadou konektorů na kterých jsou vhodně zapojeny jednotlivé vývody procesoru. Zapojení konektorů samozřejmě vyplývá především ze schématu.

Konektor LCD display

Konektor „**PFL10 – LCD display**“ je určen pro připojení LCD displeje v 4-bitovém režimu komunikace. Jednotlivé vývody konektoru jsou zapojeny podle následující tabulky.

Konektor PFL10		LCD displej		Popis
1	GND	1	GND	společný vodič (zem)
2	VCC	2	VCC	napájení +5 V
3	CT	3	V0	nastavení kontrastu
4	P1.0	4	RS	volba registru příkazů/dat
5	P1.1	5	RW	operace čtení/zápis
6	P1.2	6	EN	aktivace LCD (platná data)
7	P1.4	11	DB4	datový bit 4
8	P1.5	12	DB5	datový bit 5
9	P1.6	13	DB6	datový bit 6
10	P1.7	14	DB7	datový bit 7

Nastavení kontrastu LCD displeje se provádí odporovým trimrem R2 umístěným vedle resetovacího tlačítka. Datové bity DB0 .. DB3 LCD displeje v 4-bitovém režimu zůstávají nezapojeny.

Konektor PFL20 – I/O Connector

Na konektoru „**PFL20 – I/O Connector“** jsou vyvedeny brány P1 a P3. Tento konektor je určen pro obecné použití a je zapojen podle následující tabulky. Část pinů má konektor společných s LCD konektorem, tyto jsou popsány v tabulce.

Konektor PFL20		Popis	LCD displej	
1	P1.0	T2 … externí vstup/výstup čítače/časovače T2	4	RS
2	P1.1	T2EX …ovládání čítače/časovače T2	5	RW
3	P1.2	ECI …externí hodinový signál pro PCA	6	EN
4	P1.3	CEX0 … záchytný/komparační registr modulu PCA 0	-	-
5	P1.4	CEX1 … záchytný/komparační registr modulu PCA 1	11	DB4
6	P1.5	CEX2 … záchytný/komparační registr modulu PCA 2	12	DB5
7	P1.6	CEX3 … záchytný/komparační registr modulu PCA 3	13	DB6
8	P1.7	CEX4 … záchytný/komparační registr modulu PCA 4	14	DB7
9	P2.0	RxD … příjem dat po sériovém kanálu	-	-
10	P2.1	TxD … vysílání dat po sériovém kanálu	-	-
11	P2.2	/INT0 … vstup externího přerušení 0	-	-
12	P2.3	/INT1 … vstup externího přerušení 1	-	-
13	P2.4	T0 … vstup čítače vnějších událostí T0	-	-
14	P2.5	T1 … vstup čítače vnějších událostí T1	-	-
15	P2.6	/WR … zápis do vnější datové paměti	-	-
16	P2.7	/RD … čtení z vnější datové paměti	-	-
17	PSEN	/PSEN … čtení vnější paměti programu	-	-
18	RESET	nulovací vstup	-	-
19	GND	společný vodič (zem)	-	-
20	VCC	napájení +5 V	-	-

Na výstupu P1.3 je zapojena červená led dioda, kterou lze programově ovládat. V demonstračních příkladech je připojena maticová klávesnice na vývodech P3.5 .. P3.7 (sloupce) a P1.4 .. P1.7 (řádky). Vodiče obsluhující řádky maticové klávesnice jsou sdíleny společně s LCD displejem což je třeba patřičně programově ošetřit.

Konektor PFL40 – CPU connector

Na konektoru „**PFL40 – CPU connector“** jsou zapojeny symetricky (1:1) všechny vývody procesoru T89c51RD2. Ideálně zde lze použít jakékoliv ladící prostředky, nebo například připojit externí paměť RAM..

Konektor PFL 40 – CPU connector							
1	P1.0	11	P3.2	21	P2.0	31	/EA
2	P1.1	12	P3.2	22	P2.1	32	P0.0
3	P1.2	13	P3.3	23	P2.2	33	P0.1
4	P1.3	14	P3.4	24	P2.3	34	P0.2
5	P1.4	15	P3.5	25	P2.4	35	P0.3
6	P1.5	16	P3.6	26	P2.5	36	P0.4
7	P1.6	17	P3.7	27	P2.6	37	P0.5
8	P1.7	18	P3.8	28	P2.7	38	P0.6
9	RST	19	XTAL2	29	/PSEN	39	P0.7
10	P3.0	20	XTAL1	30	ALE	40	VCC

Konektor JMP3 - LOADER

Jumperový konektor **LOADER** nastavuje jak bude aktivován LOADER, kterým se po sériové lince RS232 programuje FLASH a EEPROM procesoru. Jumper ovládá pin PSEN, který je testován po resetu CPU a je-li na úrovni 0V (GND – nebo velmi „tvrdá“ log. 0) spustí program LOADER na adrese FC00h a tím dojde k aktivaci ISP režimu programování.

- V poloze **LDR (1-2)** je výstup PSEN připojen přes převodník úrovní MAX232 na signál CTS z sériové linky. Tento pin je dodávaným FLASHEREM programově ovládán z PC, takže není nutné stále přepínat jumper JMP3 při vývoji SW.
- V poloze **STD (2-3)** je výstup PSEN připojen trvale na zem a po ukončení programování musí být zkratová propojka na JMP3 z této pozice odstraněna, jinak se po resetu mikroprocesoru opět spustí interní program LOADER.

Pokud je jumper JMP3 – LOADER bez jumperu, neaktivujete ISP programování a po resetu se vždy spustí program od adresy 0000h.

Konektor RS232 – Cannon9

Komunikaci procesoru s okolím zabezpečuje plně duplexní sériový kanál. Logické úrovně TTL a RS232 jsou mezi sebou převáděny obvodem MAX232 či jeho ekvivalentem. Ten obsahuje dvojici převodníků TTL/RS232, a měnič napětí 5 V/±10 V. Zabudovaný měnič napětí na principu nábojové pumpy potřebuje ke své činnosti pouze čtyři externí kondenzátory C6, C7, C14 a C15. Zapojení umožňuje doplnit komunikaci přes TxD/RxD o jednoduché řízení toku dat pomocí signálů RTS/CTS. Další rozšiřování a programování procesoru je řešeno přes sériový kanál.

Konektor RS232 CAN 9F (DB9)		PC - cannon 9Male	
2	TxD	2	RxD
3	RxD	3	TxD
5	GND	5	GND
7	CTS	7	RTS
8	RTS	8	CTS

Konektor POWER - +5V

Na konektoru POWER je vyvedeno stabilizované napětí +5V a společná zem. Při použití tohoto konektoru k napájení periferních obvodů je třeba brát ohled na doporučení uvedená v textu o napájení RD2-kitu (max. cca 200mA).

Závěr

Výběr správného JM je jedním z kritických rozhodnutí, které často ovlivňují úspěch či selhání celého projektu. Hlavním naším cílem při návrhu RD2-kitu byla volba moderního JM schopného splnit požadavky kladené současným trendem v oblasti vývoje embedded zařízení. Stále více inteligentních periferií je integrováno přímo na čipu. Tato skutečnost se positivně projevuje do nárůstu spolehlivosti (testování již během výroby, snížení počtu externích periferií) a komfortu při ovládání (dedikované instrukce procesoru). Vyloučení externích prvků sebou přináší i snížení požadavků na velikost DPS a spotřebu celého zařízení.
Poměr pamětí RAM ku ROM se v minulosti pohyboval 1:32, dnešní trend však směřuje k 1:16 (1:8). Více paměti RAM umožňuje použít při vývoji perspektivnějších vyšších programovacích jazyků. Pro aplikace kde se předpokládá změna kódu programu jsou vhodné JM s pamětí programu typu FLASH a rozhraním ISP.
V komplexnějších projektech je těžiště při samotné realizaci v oblasti návrhu a testování firmwaru. Vhodná volba vývojových a podpůrných nástrojů nám ušetří spoustu zbytečných komplikací nyní i v budoucnu. Není důvod se obávat nasazení vyšších programovacích jazyků. Sestavený kód programu v kvalitním překladači jazyka C je dnes plně srovnatelný s programem napsaným v jazyku symbolických adres.
Přestože původní architektura JM s jádrem 8051 pochází z roku 1980 těší se stále neutuchající oblibě u řady vývojářů. To je zapříčiněno tím, že dochází k neustálému obohacování funkcí a vlastností původního procesoru při zachování zpětné kompatibility. Nevýhodou procesoru 8051 je omezení vyplívající z 8-bitové architektury a velikosti adresního prostoru 64 kB.

T89c51RD2 patří mezi jedny z posledních typů jednočipových mikropočítačů řady 8051 uvedených firmou Atmel na trh. Samozřejmě se nejedná o žádný ideální JM, takový totiž neexistuje. Má však řadu vlastností, které se jevily jako velice vhodné pro jeho použití ke stavbě vývojového kitu. RD2-kit může najít své uplatnění u začátečníka se zájmem o JM, učitele či studenta na škole nebo zkušeného vývojáře embedded zařízení.

RD2 - Flasher verze 2.2 for MS Windows

Program RD2-Flasher slouží k programování jednočipového mikropočítače Atmel T89c51RD2 metodou ISP přes sériový port v operačních systémech rodiny MS Windows.

Metoda ISP (In System Programming) přes sériový port programuje ve spolupráci s programem LOADER v procesoru RD2 jeho interní FLASH paměť. Program Flasher pracuje v řádkovém režimu v operačních systémech rodiny MS Windows. Jedná se o plně nativní Win32 konzolovou aplikaci určenou pro snadnou integraci do libovolného stávajícího vývojového prostředí (Keil μVision2, SDCC, atd.).

RD2 – Flasher představuje alternativní řešení k originálnímu softwaru FLIP firmy Atmel. Ve spojení s námi vyvinutým zavaděčem (plně kompatibilní s FLIP) pro JM T89c51RD2 nabízí celou řadu nových funkcí a vlastností.

Vlastnosti RD2 – Flasheru

- čtení, zápis a verifikace paměti FLASH a EEPROM,
- detekce úrovně uzamčení JM,
- uzamčení JM,
- vymazání paměti FLASH JM,
- zabezpečení paměti FLASH pomocí kontrolního součtu,
- validace vstupního souboru Intel – HEX,
- ovládání vývodu PSEN u JM přes sériový port vodičem RTS,
- přenosová rychlost 9600 až 115200 Bd,
- snadná integrace do stávajících IDE.

Ovládání programu je řešeno pomocí parametrů z příkazové řádky. U parametrů, které mohou nabývat dvou stavů zapnuto/vypnuto je přiřazena hodnota '0' stavu vypnuto a '1' stavu zapnuto. Komunikaci mezi PC a programovaným JM lze provádět ve dvou módech. Mód 0 posílá celý řádek načtený z textového souboru Intel – HEX a následně čeká na odpověď. Mód 1 vyšle znak a čeká na odpověď.
Běh programu lze přerušit stisknutím kláves CTRL-C nebo CTRL-BREAK.

Testované platformy

Windows 95, Windows 98se, Windows 2000 Server + SP2, Windows XP Professional

Možnosti sériové komunikace na straně PC

Program je schopen využít libovolný volný sériový port, který je mu dán k dispozici operačním systémem. Volba portu se provádí parametrem –c COMx. Ovládání sériového portu je řešeno pomocí volání standardních API funkcí. Zkontrolujte, prosím nastavení vyrovnávací paměti FIFO u sériových portů. V případě že je vypnuta, tak ji zapněte.

Podporované přenosové rychlosti pro sériovou komunikaci
110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200

Nastavená přenosová rychlost sériového portu parametrem –b na PC musí být podporována krystalem připojeným k programovanému JM. Zavaděč obsahuje rutinu autodetekce přenosové rychlosti čímž je odstraněn problém vazby na fixní hodnotu přenosové rychlosti. Prakticky použitelné rychlosti se pohybují od 9600 Bd.

Zapojení sériového programovacího kabelu

Procesor RD2 je připojen běžným sériovým kabelem (TxD, RxD, GND). Komunikace nevyužívá žádné formy řízení toku dat. Zapojením vodiče RTS a následným propojením na vývod PSEN u JM T89c51RD2 lze přímo řídit průběh programování z obslužného softwaru na PC.

Následující tabulka popisuje připojení jednotlivých zařízení z HW serveru k loaderu.
Popsaná polarita (Male / Female) se popisuje konektor zařízení.

PC CAN9M	RD2 Kit CAN9F
>- RxD (2)	-> TxD (2)
-> TxD (3)	>- RxD (3)
-- GND (5)	-- GND (5)
-> RTS (7)	>- CTS (7)

Test funkčnosti zavaděče a sériového kabelu
Zavaděč programu se aktivuje uzemněním vývodu PSEN během nulování JM. Následně zavaděč provádí autodetekci přenosové rychlosti (PC vysílá po sériové lince periodicky znak 'U'). Po zesynchronizování posílá zavaděč všechny přijaté znaky zpět do PC. Manuální test funkčnosti kabelu a zavaděče lze provést s libovolným sériovým terminálem. Vypneme lokální odezvu odesílaných znaků. Držíme klávesu 'U' a čekáme na odezvu od zavaděče.

Práce s programem FLASHER

Programováni paměti FLASH JM
U programovaného JM aktivujeme zavaděč (uzemníme vývod PSEN a vynulujeme JM). Procesor musí být napájen ze stabilizovaného zdroje 5V s dostatečnou proudovou rezervou. Při nižším napájení může docházet ke stochastickým jevům v průběhu programování a komunikace s PC.
Vstupní soubor Intel – HEX se uvádí za parametrem –f. Obsah souboru se načte do paměti, kde dojde ke kontrole CRC a následně je vytvořen dočasný soubor rd2f.hex, který je použit při samotném programování. Tento postup odstraňuje problémy, které se vyskytovaly při programování některých souborů Intel – HEX.

Programováni paměti EEPROM JM
Postup programování paměti EEPROM je shodný s programováním paměti FLASH. Soubor Intel - HEX je pouze doplněn o informace pro zavaděč, udávající celočíselnou část kmitočtu v MHz použitého krystalu. Např. hodnota parametru –q pro kmitočet 22,1184 MHz bude 22.

Verifikace paměti EEPROM a FLASH JM
Parametrem –v lze povolit provedení verifikace obsahu paměti EEPROM a FLASH po ukončení programování.

Čtení paměti EEPROM a FLASH JM
Obsah paměti EEPROM (2 kB) a FLASH (64 kB) lze vyčíst pokud použijeme parametr –r. Soubor, kde bude obsah vyčtené paměti uložen se udává parametrem –f. Čtení nelze provádět u JM, který je chráněn proti čtení.

Uzamčení JM
JM lze po úspěšném naprogramování uzamknout parametrem -l. Úroveň 2 poskytuje ochranu proti zápisu a úroveň 3 doplňuje dále ochranu proti čtení.

Ovládání vývodu PSEN z PC
Možnost ovládání vývodu PSEN z PC, předpokládá zapojení kabelu s RTS vodičem. Použitím parametru –p dojde po spuštění programu k automatickému uzemnění vývodu PSEN. Následným manuálním vynulováním JM se aktivuje zavaděč. Dále postupujeme podle pokynů uvedených v programu.

Textový výstup programu
Parametrem -t lze zvolit barvu, kterou použije program při výstupu na obrazovku. K dispozici je základních 16 barev používaných pro zobrazování v textovém režimu na VGA adaptéru.

Zabezpečení paměti FLASH JM pomocí kontrolního součtu
Pro účely diagnostiky funkčnosti JM přímo v aplikaci byla navržena metoda ochrany paměti FLASH pomocí kontrolního součtu. Oblast paměti FLASH začíná na adrese 0000h a je z hora ohraničena adresou udanou parametrem –s. Takto vymezená oblast je zabezpečena kontrolním součtem používaným u souboru Intel – HEX. Výsledné číslo je uloženo hned na následující adrese v paměti FLASH.

Tipy, triky, praxe

- **Uzamčení procesoru**
  Procesor T89c51RD2 umožňuje aktivaci několika úrovní ochrany kódu programu. Vedle klasické ochrany proti čtení a zápisu existuje možnost chránit kód programu proti paralelnímu nebo ISP přístupu. RD2-Flasherem lze nastavovat pouze přístupová práva pro ISP metodu. V případě, že jsou však špatně nastaveny bity u paralelního přístupu (hardwarovým programátorem), lze uzamčený JM metodou ISP bez problémů vyčíst v klasickém programátoru. Situace je obdobná i v opačném případě.

- **Zápis a verifikace FLASH paměti**
  Seznam chyb JM T89c51RD2 uveřejněný firmou Atmel uvádí, že v určitých situacích může docházet k chybám během programování paměti flash. V doporučení je uvedeno provádět celkové vymazání JM před samotným programováním a verifikaci obsahu paměti po ukončení programování.

- **Postup integrace RD2-Flasheru do Keil μVision2 IDE**
  Vybereme menu *Tools → Customize Tools Menu…* a zadáme název nového menu *RD2F with @P.hex*. @P reprezentuje název aktuálního projektu v prostředí μVision2. Do pole *Command* zadáme cestu k programu rd2f.exe. V poli *Arguments* uvedeme parametry, které chceme použít např. -f @P.hex -c COM1 -b 115200 –p 1. Na závěr zaškrtneme volbu *Run Independent*. V menu *Tools* vznikne nová položka označená *RD2F with …,* pomocí níž lze následně spustit RD2-Flasher.

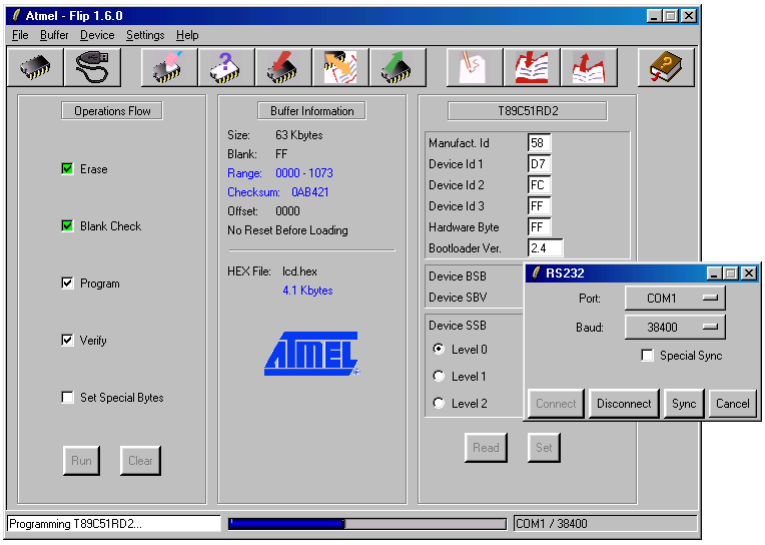
- **Komunikace s originálním zavaděčem firmy Atmel**
  RD2-Flasher byl vyvinut a optimalizován pro práci s upraveným zavaděčem napsaným Radkem Benediktem. Originální zavaděč firmy Atmel je však mnohem pomalejší a pro komunikaci s ním je třeba použít volbu –m 0.

- **Zobrazení konzolové aplikace na celou obrazovku ve Win9x**
  Zvýšení výkonu a stability lze dosáhnout spouštěním RD2 – Flasheru v zobrazení na celou obrazovku. Konzolové aplikace jsou v operačním systému Win9x spouštěny pomocí programu *„c:\windows\system\conagent.exe".* Změnou vlastností pro tento program lze globálně nastavit chování všech konzolových aplikací (např. zobrazení na celou obrazovku).

- **Problémy s používáním ve Windows 2000**
  Při testování funkčnosti programu ve Windows 2000 jsme narazili na následující jev. Na stejném počítači ve Windows98se fungovalo vše bez jediného náznaku výpadku či chyby při přenosu dat. Ve Windows 2000 nedošlo ani k zesynchronizování se zavaděčem v programovaném JM. Při sériové komunikaci docházelo ke ztrátám při přenosu a ukončování komunikace. Funkčnost komunikace byla obnovena poté co jsme vyměnili sériový kabel.

Atmel FLIP

Pokud preferujete „klikací“ software, který není možné volat přímo z vývojového prostředí, zkuste použít FLIP od firmy ATMEL. Jedná se o totéž jaké náš Flasher, ale originální software neumí ovládat EEPROM, protože původní loader uvnitř procesoru je kompatibilní s loadery pro procesory RD2 od Philipsu, které nemají EEPROM.



Obrázek zobrazuje FLIP během nahrávání příkladu 2.2 pro obsluhu LCD displeje do RD2 Kitu..

Nesporná nevýhoda také je, že FLIP neumí přepnout RD2 Kit přímo do programovacího režimu pomocí ovládání pinu PSEN z procesoru vývodem CTS z RS232. Při používání FLIPu proto musíte před každým downloadem software přepnout jumper a stisknout RESET, zatímco Flasher si jumper pomocí CTS přepne sám a vy jenom zresetujete aplikaci..

Závěr

Ať už použijete náš Flasher, nebo originální program FLIP, používejte ISP programování. Doba programování procesorů v programátoru a přendávání z patice do patice během vývoje firmware je již dávno pryč..

C51 - Jak a proč vznikl

Zvyšováním integrované paměti RAM a FLASH integrovaných na čipu jednočipových mikroprocesorů se v poslední době usnadňuje použití vyšších programovacích jazyků i pro malá zařízení. Mnoho vývojářů proto přemýšlí o přechodu na vyšší programovací jazyk, většinou na ANSI C. Chtěl bych zde popsat vývoj tohoto jazyka pro procesory z rodiny x51.

Jazyk C je spíše hutný a matoucí jazyk. Obecně je uváděn mezi vyššími jazyky. A opravdu obsahuje mnoho rysů odpovídajících definici jazyka vyšší úrovně - strukturované programování, definici volání procedur, předávání parametrů, podporu struktur atd. Nicméně hodně z síly C leží v jeho schopnosti spojit jednoduché, nízko úrovňové příkazy do komplikovaných funkcí a dovolit přístup k reálným prostředkům hostitelského procesoru. Zjednodušeně by se dalo říci, že C je druh univerzálního jazyka symbolických instrukcí.

Většina programátorů, kteří píší v C, jej využívá pro psaní programů pro relativně výkonné procesory s dostatkem paměti. Ta umožňuje aby i v omezených 640kB paměti MS-DOSu, byla nejmenší proměnná integer o velikosti 16 bitů. Vlastní rozhraní s reálným světem je zprostředkováno přes přerušení OS a voláním funkcí OS. A tak se programátor může většinou soustředit na manipulaci a zpracování proměnných, řetězců, polí atd.
U 8-bitových mikroprocesorů je nicméně situace poněkud jiná. Vezmeme-li 8052 jako příklad, může být program umístěn do 8 KB a využívat pouze 256B RAM. Reálné periferie, jako jsou porty, sériový kanál, časovače, SFR musí být přímo přístupné z jazyka C. Přerušení musí být ošetřeno pomocí volání vektorů na absolutních adresách. Velkou pozornost je třeba věnovat přidělování paměti reetrantním funkcím, snadno totiž může dojít k přepsání aktuálních dat. (*Reetrantní je typ funkce, která volá ze sebe jinou funkci – například funkce kresli čáru, která volá funkci kresli bod.*) Jeden ze základních rysů jazyka C je skutečnost, že parametry a výsledky funkcí jsou předávány na zásobníku. A samozřejmě funkce mohou být volány jak v hlavním programu tak i přerušením a stále musí být zabezpečena ochrana před ztrátou dat.

Největším omezením rodiny MCS51 je právě malý rozsah zásobníku. Většina ostatních procesorů má ukazatel na zásobník 16-bitový a je možné případně použít i jiné registry jako dočasné ukazatele na zásobník. Jazyk C má opravdu extremní požadavky na správu zásobníku a schopnost přístupu k datům jeho prostřednictvím. Mikroprocesory rodiny MCS51 jsou obdařeny zásobníkovým systémem, který je reálně schopný zvládat jen manipulaci s návratovými adresami. S 256 bajty zásobníku je opravdu těžké si představit funkční rozsáhlejší program, s několikanásobně vnořeným voláním funkcí a předáváním parametrů a návratových hodnot. A to stále za předpokladu, že úplně pomineme paměťový prostor pro globální proměnné.

Z výše zmíněného by se dalo usuzovat, že implementace vyššího jazyka, intenzivně využívajícího zásobník jako je právě jazyk C, není na MCS51 realizovatelná. A opravdu velmi dlouho taky nebyla žádná implementace reálně použitelná. Většina kompilátorů byla totiž pouze adaptována z překladačů pro mnohem výkonnější procesory a byla velmi neefektivní. Přístup k rozsáhlému zásobníku v externí paměti byl realizován prostřednictvím knihovních podprogramů, které byly volány vždy při vstupu/výstupu do/z funkce. Na jednu stranu to přineslo dostatek prostoru pro zásobník a reetrantnost funkcí, ale za cenu velmi pomalého běhu vlastního programu, pomalé reakce na přerušení atd. Běžnou realitou bylo, že výsledný kód měl režii větší než vlastní výkonný algoritmus.

Vedle neefektivní správy zásobníku byl i generovaný programový kód špatně optimalizován na specifické vlastnosti procesorů rodiny MCS51. Všechny tyto důvody vedly k dalšímu zvyšování nároků na velikost použité paměti a následně k nutnosti řešit problémy s případným přepínáním paměťových stránek pomocí IO portů. A tak se zdálo, že programování malých a rychlých aplikací je možné jedině v assembleru.
Nicméně, vývoj jde stále vpřed. A tak Intel již v roce 1980 představil částečné řešení problému implementace vyššího jazyka pro jeho MCS51 - jednalo se o jazyk PLM51. Tento kompilátor nebyl dokonalý, byl adaptován z PLM85 (8085), ale Intel si jako první realisticky uvědomil, že implementace založená výhradně na zásobníku není jednoduše možná. Intel použil na prví pohled velice nesystémové, ale jednoduché řešení - parametry a návratové hodnoty funkcí nejsou předávány pomocí zásobníku, ale ve vyhrazené, předem definované oblasti paměti.

Použití vnější paměti sice tento proces zpomaluje, ale je stále rychlejší než použití umělého (programově implementovaného) zásobníku. Nevýhoda tohoto řešení je jasná - není možná reetrantnost funkcí. Toto podstatné omezení ale vlastně v typických programech pro MCS51 nevytváří žádný větší problém. Pokud je třeba, nechá se funkce navrhnout tak, aby reetrantnost umožňovala. Neřeší to však obecně překladač, ale programátor. *Poznámka: Novější verze C51 již umožňují určit kompilátoru funkce, které mají být reetrantní.* A tak případná reetrantnost nijak významně nesnižuje výkonnost celého programu.

Další problémy, které musí kompilátor jazyka C řešit:

- přístup k interním a externím periferiím
- správa přerušení
- optimální využití omezené instrukční sady
- specifické vlastnosti rozdílných paměťových prostorů
- podpora různé konfigurace pamětí ROM a RAM
- vysoká úroveň optimalizace pro maximální využití kódového prostoru
- přepínání registrových bank
- podpora různých klonů standardní 8051

Překladače jako je například „Keil C51“ obsahují dnes všechna tato nutná rozšíření jazyka C pro použití ve světě mikroprocesorů MCS51. C51 je postavený na technikách navržených Intelem a zároveň přidává vlastnosti jazyka C - aritmetiku v pohyblivé řádové čárce, formátovaný vstup a výstup, atd. C51 se tak stal prakticky nepsanou standardní implementací ANSI C pro mikroprocesory rodiny MCS51.

Pamětové modely

Keil C51 umožňuje C programátorům psát rychle a efektivně programy pro systémy na bázi MCS51 bez nutnosti rozsáhlého učení nových specifik překladače a cílové platformy. Nicméně, pro dosažení nejlepších výsledků je nutné zvládnou základní znalosti o architektuře procesorů rodiny MSC51. Základním rozhodnutím je volba paměťového modelu.

Paměťová konfigurace - fyzické umístění paměťových prostorů
Většinu začátečníku plete Harvardská architektura procesorů MCS51. Pokud předtím pracovali s procesory s Von Neumanovou architekturou (68Hxx) je to značně matoucí. Místo jednoho sekvenčního adresového prostoru jsou najednou postaveni před různé paměťové prostory, které začínají na stejné adrese. A aby to bylo ještě matoucí, některé jsou implementovány fyzicky stejnými HW prostředky, a odlišují se pouze přístupem (poznámka - vše je uváděno z pohledu C51):

- **CODE** - prostor začíná na 0x0000 a končit na adrese 0xFFFF. Jedná se o oblast primárně určenou pro uložení programu (ROM), ale můžeme v ní ukládat i konstanty, tabulky, a podobně. Přístup je přes speciálních instrukce MOVC A,@DPTR a MOVC A,@A+PC, a data můžeme pouze číst.
- **DATA** - prostor začíná na adrese 0x00 a končí na adrese 0xFF. Je přímo adresovatelný (MOV A, x) a aby to nebylo tak jednoduché, dělí se na dvě oblasti. První je oblast 0x00-0x7F kde jsou uloženy registrové banky, bitové proměnné a mohou zde být uloženy pracovní proměnné programu. Druhou je oblast 0x80..0xFF kde jsou speciální funkční registry (SFR), tedy registry sloužící ke konfigurování a přístupu k integrovaným periferiím, akumulátoru, zásobníku a podobně.
- **IDATA** - prostor začíná na adrese 0x00 a končí na adrese 0xFF. Je přístupný pouze nepřímým adresováním (MOV A,@Ri) a opět je rozdělen na dvě oblasti. První je fyzicky sdílena s prostorem DATA, druhá je samostatná a překrývá oblast SFR.
- **BDATA** - prostor začíná na adrese 0x00 a končí na adrese 0xFF. A opět je tento prostor rozdělen na dvě oblasti. První je oblast bitových proměnných. Fyzicky je tato oblast umístěna v oblasti DATA na adresách 0x20 - 0x2F. Do druhé oblasti 0x80 - 0xFF jsou namapovány některé z registrů speciálních funkcí. Tak je umožněn bitový přístup k těmto registrům.
- **PDATA** - jedná se vlastně o prostor XDATA, ale se stránkovým přístupem realizovaným instrukcí MOVX A,@Ri. Stránka o velikosti 256B je určena obsahem registru (portu) P2.
- **XDATA** - prostor začíná na 0x0000 a končí na 0xFFFF. Jedná se o oblast primárně určenou k ukládání dat. Přístup je přes speciální instrukce MOVX A, @DPTR.

Paměťové modely C51

Při psaní aplikace v C51 je nutné nejdříve rozhodnout který paměťový model použijeme. Zatímco PC programátor si může vybrat mezi TINY, SMALL, MEDIUM, COMPACT, LARGE a HUGE modelem a neřeší problém s různými prostory pro program a data, v C51 je tomu poněkud odlišně. Paměťový model zde vlastně řeší implicitní umístění parametrů, automatických proměnných a deklarací bez explicitní definice umístění. C51 podporuje následující paměťové modely:

- **SMALL** - všechny proměnné a parametry jsou umístěny v x51 interní paměti (IDATA)
- **COMPACT** - proměnné jsou uloženy v stránkované vnější paměti (PDATA) adresované porty P0 (a P2). Je použito nepřímé adresování. Interní paměť je použita pro lokální proměnné a parametry.
- **LARGE** - proměnné jsou umístěny v externí paměti (XDATA). Interní paměť je použita pro lokální proměnné a parametry.
- **BANKED** - zvláštní model určený pro rozsáhlé programy. Kód může být uložen až v 1MB s tím, že se používá stránkování. Uvnitř každého 64KB paměťového bloku musí existovat společná oblast pro uložení knihovních funkcí. Volání a skoky mezi bankami jsou samozřejmě možné.

Existuje samozřejmě možnost použít jeden model globálně a explicitně určit jednotlivým proměnným a objektům paměťový prostor, kde budou uloženy. A samozřejmě lze paměťové modely i kombinovat pro různé části kódu. Zde je však třeba pamatovat na možnost vzniku problému při linkování. Typický program v C51 je překládán s jedním globálním modelem, například COMPACT. Pro rychlé rutiny - přerušení je však vhodné použít model SMALL. Nejjednodušší a všem zřejmý přístup s použitím #pragma MODEL (případně samostatný překlad modulů s odlišně nastavenými paměťovými modely). Některé chyby, které mohou tímto způsobem vzniknout a jejich řešení jsou popsány v dlouhé verzi článku na www.MCU.cz.

Typ	Vhodné	Nevhodné
CODE	• konstanty • tabulky	• proměnné !!!
DATA	• často používané proměnné • proměnné v rutinách přerušení	• větší pole a struktury
IDATA	• malá pole a struktury s rychlým přístupem • zásobník	• proměnné větší než CHAR
PDATA	• běžné proměnné • pole a struktury	• proměnné LONG a FLOAT • často používané proměnné • pole a struktury větší než 256B
XDATA	• rozsáhlá pole a struktury • relativně méně často používané proměnné	• často používané proměnné • proměnné s rychlým přístupem

M. Kostomlatský

Kapitolu „C51 - Jak a proč vznikl“ jsme převzali se svolením autora ze serveru www.MCU.cz

Překladače jazyka C pro mikroprocesory řady x51

K vývoji aplikací pro jednočipové mikropočítače lze použít celou řadu programovacích jazyků (jazyk symbolických instrukcí, C, Pascal, java, Basic, atd.). Dominantní postavení si stále udržuje jazyk symbolických instrukcí s více než 50 %. Současný trend však stále více směřuje, k použití vyšších programovacích jazyků, jmenovitě jazyka C. Pro mnohé vývojáře se však jedná o zásadní problém, který lze nazvat strachem z něčeho nového, složitého a neznámého. Proč přecházet od zavedeného programování v jazyku symbolických instrukcí k něčemu méně efektivnímu? Na tuto otázku a další by měl odpovědět následující text.

Pohled na vývoj z hlediska aplikace

Komplexnost současných jednočipových aplikací z oblasti automobilového průmyslu, GSM a dalších oblastí, spolu s narůstajícím tlakem na zpracování aplikací jasně směřují k použití vyšších programovacích jazyků.

Výhody použití vyšších programovacích jazyků při zpracování aplikací :

- snížení doby vývoje a tím spojených nákladů,
- zpřehlednění zdrojových kódů,
- dlouhodobou správa a údržbu projektu po garantovanou dobu podpory pro dané zařízení,
- důraz na multiplatformnost (jeden zdrojový kód pro různé typy mikroprocesorů),

Výrobci mikroprocesorů tento trend podporují a stále častěji integrují větší množství paměti RAM a FLASH přímo na čipu. V základním popisu těchto mikroprocesorů je přímo vyzdvihováno použití ve spojitosti s vyššími programovacími jazyky.

Proč použít jazyk C?

Jazyk C byl od prvopočátku navrhován k systémovému (nízkoúrovňovému) programování. Díky tomu se prosadil všude tam, kde je třeba přímo ovládat libovolný hardware. V jazyku C je napsaná celá řada operačních systému, uživatelských a jednočipových aplikací. Překladače jazyka C existují pro většinu dostupných typů procesorů.

Výhody jazyka C

- jednoduchá přenositelnost zdrojových kódů programu (multiplatformnost),
- přehlednost zdrojových kódů programu a celkové zjednodušení při správě složitých projektů,
- rychlejší vývoj aplikací (použití standardních knihoven),
- nástroje na optimalizaci a validaci výsledného kódu programu,
- zavedený programovací jazyk, podporovaný výrobci hardwaru (mikroprocesorů) a softwaru (kompilátorů).

Nevýhody jazyka C

- vyšší cena kvalitního vývojového prostředí,
- relativně větší nároky na paměť dat a programu výsledné aplikace,
- složitější na osvojení.

Implementace jazyka C u mikroprocesorů řady x51

Díky neutuchající oblibě mikroprocesorů řady x51 je k dispozici dostatečné množství kompilátorů jazyka C. Základní vlastností jazyka C je, že využívá intenzivně práce se zásobníkem (předávání parametrů, lokální proměnné, návratové adresy funkcí, reetrantní funkce, atd.). To je kamenem úrazu u mikroprocesorů řady x51, která není obdařena vhodným zásobníkovým systémem. Úspěšnost implementace jazyka C u mikroprocesorů řady x51 závisí na tom, jak se daný výrobce kompilátoru vypořádal s tímto nelehkým problémem. V další fázi se pak řeší specifické problémy s implementací jazyka C na 8bitový mikroprocesor řady x51

- přístup k interním a externím periferiím,
- správa přerušení,
- optimální využití omezené instrukční sady,
- specifické vlastnosti rozdílných paměťových prostorů,
- podpora různé konfigurace pamětí ROM a RAM,
- vysoká úroveň optimalizace pro maximální využití kódového prostoru,
- přepínání registrových bank,
- podpora různých klonů standardní 8051.

Počet kompilátorů jazyka C pro mikroprocesory řady x51, které můžeme nalézt na Internetu se hravě vyšplhá k číslu dvacet. Jak se v takovém počtu kompilátorů jednoduše zorientovat a podle čeho vybírat ten správný?

První věc co by měl každý zájemce udělat, je navštívit internetové stránky daného výrobce. U většiny výrobců lze volně získat ukázkovou verzi jejich produktu k vyzkoušení.

Tyto volné verze jsou různě limitovány například :

- velikostí výsledného kódu aplikace,
- počáteční adresou, na které je aplikace sestavena,
- časovým omezením doby používání,
- absencí některých knihoven (např. práce s plovoucí čárkou),
- dostupnými paměťovými modely,
- počtem podporovaných klonů mikroprocesorů x51,
- atd.

Takto volně získaná verze programu, často plně uspokojí potřeby začátečníka, studenta či amatéra.
S dostupnými ač limitovanými nástroji, lze vytvořit spoustu funkčních a zajímavých aplikací. Speciálně vhodných pro základní řadu mikroprocesorů firmy Atmel, velice oblíbených u této skupiny vývojářů.

V oblasti GNU kompilátorů pro mikroprocesory řady x51 je k dispozici pouze jeden zástupce s názvem SDCC (Small Device C Compiller). Svými vlastnostmi výsledného kódu programu se řadí do střední třídy kompilátorů. Při realizaci složitějších projektů, je však třeba počítat, se zvýšeným úsilím, ze strany vývojáře k dosažení patřičného výsledku. Pokud se hodláte zabývat vývojem aplikací na profesionální úrovni nezbude vám nic než se poohlédnout po komerčním řešením.

Jak by měl vypadat nástroj pro vývoj aplikací v C

Pokud hodláme investovat nemalé peníze do nákupu vývojového prostředí měli-by jsem si uvědomit co od daného produktu chceme a k čemu ho budeme používat. Při výběru se pak zaměřit na následující body

- integrované vývojové prostředí (IDE),
- správa projektu a překlad (inteligentní make),
- podporované klony x51 (jsou využívány nové instrukce),
- softwarový debugger (simulace nových periferií u klonů x51),
- návaznost na další externí programy (CVS, PC-Lint, programovací SW, …),
- výstupní formát kompilátoru (BIN, Intel-HEX, OMF-51, …),
- optimalizace kódu aplikace (je možné vytvořit aplikaci > 64 kB),
- paměťové modely (interní/externí CODE a XRAM),
- náchylnost k chybám při překladu,
- podpora standardu ANSI C (do jaké míry, je kompilátor schopen přeložit zdrojový kód v C),
- rozsah implementovaných standardních funkcí jazyka C,
- pokračuje výrobce v dalším vývoji kompilátoru, updaty, uživatelská podpora,
- dostupná dokumentace k danému kompilátoru.

Seznam kompilátorů jazyka C

- Dunfield Development Systems - http://www.dunfield.com/
- HI-TECH Software - http://www.htsoft.com/
- IAR Systems - http://www.iar.com/
- Keil Software, Inc. - http://www.keil.com/
- Franklin Software - http://www.fsinc.com/
- Altium Limited - www.tasking.com
- Avocet Systems, Inc. - http://www.avocetsystems.com/
- Small Device C Compiler - http://sdcc.sourceforge.net,
- Wickenhäuser Elektrotechnik µC51 - http://www.wickenhaeuser.com/
- Raisonance - www.raisonance.com
- Crossware Products - www.crossware.com
- Rigel Corporation Reads51 - www.rigelcorp.com/
- a další.

Podstatné rozdíly mezi kompilátory

Objektivně zhodnotit vlastnosti všech kompilátorů je pomalu nadlidský a časově náročný úkol. Proto bych se pokusil podělit o pár osobních zkušeností, které jsem nabil u kompilátorů co jsem měl možnost vyzkoušet (Keil, SDCC, Franklin, Hi-Tech, Reads51, µC51).

1. Vždy se podívejte na datum poslední aktualizace daného kompilátoru. Používejte pokud možno poslední verzi programu (SDCC). Je z podivem, že někdo je schopen prodávat kompilátor, který už několik let nevyvíjí (Franklin Software).

2. Jaké klony x51 jsou opravdu podporované, nejedná se pouze o základní 8051/52 (SDCC). Je možné zvolit překlad aplikace pro daný typ procesoru, rozšířenou instrukční sadu, … (Keil).

3. Napište si triviální aplikaci a tu zkuste přeložit pod vícero překladači. Podívejte se jestli je pro váš procesor k dispozici hlavičkový soubor. V jakém formátu jsou zapisovány hlavičkové soubory, lze použit moje stávající hlavičkové soubory i jinde?

4. Jedná se o zavedený kompilátor (Keil, Hi-Tech, …) nebo se někdo snaží vytvořit svoji vlastní variantu C (µC51)?

5. Je to plnohodnotné C nebo kompilátor z řady small C (Reads51), který je spíš vhodný na hraní než na praktické aplikace.

6. Jak pracuje daný kompilátor se zásobníkem? Nemá kompilátor tendenci obsazovat paměť a neuvolňovat ji (SDCC).

7. Je vývojové prostředí opravdu Win32 aplikací, nespouští se náhodou pouze v DOSovém okně (Hi-Tech)?

8. Má výrobce zkušenosti i s kompilátory pro jiné typy procesorů než x51?

9. Jaký formát ukládání dat používá daný kompilátor big/little endian (Keil/SDCC)?

Obecně lze říct, že většina kompilátorů nemá problémy, pokud programovaná aplikace vystačí s prostředky dostupnými přímo na čipu (model small). Skutečným testem, kde lze vidět co kompilátor dokáže natropit, je přibližně aplikace kolem 1000 a více řádků zdrojového kódu. Nejlépe kompilovaná v modelu large s externí pamětí XRAM a základními periferiemi.

Bohužel takto složitou aplikaci, jako celek není vůbec jednoduché převést z jednoho kompilátoru do druhého. Obzvláště pokud tak činíme poprvé, kdy je vyžadována nutná dávka entuziasmu. Důvod je prostý. Řada věcí se dá zapsat v jazyku C různými způsoby, ale každý kompilátor si zdrojový kód interpretuje trochu jinak. Pokud tyto zkušenosti vývojář jednou nabude, pak mu nečiní obtíže vyvíjet aplikace pro více typů kompilátorů.

SDCC ač vyvíjené skupinou nadšenců si v porovnání s některými komerčními produkty vede docela obstojně. Chybí spousta vymožeností, které je dána absencí IDE a s tím spojený komfort při vývoji. Rozšířená syntaxe jazyka C používaná u mikroprocesorů řady x51 je u SDCC a Keil C51 shodná. To umožňuje psát programy, které lze jednoduše přeložit v obou kompilátorech.

Keil C51 je odbornou veřejností považován za jeden z nejlepších kompilátorů pro x51. Je to pravda a na celém produktu je vidět několik let intenzivního vývoje, který má za sebou. Pokud vážně uvažujete o nákupu komerčního kompilátoru, tak by jste při výběru neměli opomenout Keil C51.

Závěr

Vzhledem k popsanému, jsme do RD2 kitu zvolili jako příklady kompilerů právě Keil a SDCC. Pro oba kompilátory najdete na CD ke kitu více než 40 příkladů…

Úvod do programování v Keil C51 a SDCC

Popis základních principů použití vyšších programovacích jazyků pro vývoj jednočipových aplikací. Konkrétní příklady použití nástrojů Keil C51 a SDCC. Tvorba jednoduchého programu, překlad, ladění a spuštění aplikace na cílovém jednočipovém mikropočítači T89c51RD2 (RD2-Kit).

Úvod

V dnešní době existuje celá řada nástrojů pro vývoj jednočipových aplikací ve vyšších programovacích jazycích. Mezi jedny z nejvyspělejších patří komerční produkty firmy Keil. Seznámení s problematikou návrhu aplikací ve vyšších programovacích jazycích si předvedeme na omezené verzi vývojového prostředí Keil C51, které je volně k dispozici na Internetu. Protipól při výkladu nám bude tvořit GNU kompilátor SDCC (Small Device C Compiler). Porovnáním těchto dvou nástrojů máme možnost si vytvořit náhled na problematiku vývoje jednočipových aplikací. Demonstrace funkčnosti výsledné aplikace je provedena s vývojovým kitem osazeným procesorem T89c51RD2, který díky svým vlastnostem (ISP, FLASH 64 kB, XRAM 1kB, integrované periferie na čipu,…) je velice vhodný pro daný účel. V dalším textu se předpokládá základní znalost konstrukcí jazyka C a architektury 8051. Hlavní důraz bude kladen na specifické použití jazyka C v jednočipových aplikacích demonstrovaný na praktických příkladech.

Keil C51 – evaluation version 7.01

Volně dostupná verze vývojových nástrojů Keil C51 (www.keil.com) pro procesory x51 má následující omezení:

µVision2 IDE

- max. 5 zdrojových souborů v rámci jednoho projektu,
- editace souborů menších jak 16 kB.
- C51
- kompilátor, linker a debugger pracují pouze se objektovými soubory do 2 kB,
- výsledná aplikace je uložena od adresy 800h a max. velikost je omezena na 2 kB,
- není podporována práce s čísly v plovoucí čárce,
- není podporována práce s více hardwarovými ukazateli dat (DPTR),
- není k dispozici RTX51 Tiny Real-time Operating System.

Pro pochopení principů a návrh jednoduché aplikace jsou však dostupné prostředky plně postačující. Vývoj se provádí v IDE µVision2, které sdružuje management projektu, editaci zdrojových kódů, překladač, linker a debugger v jeden celek. Instalace programu je obdobná běžně používanému postupu známého z prostředí MS Windows a není třeba jí věnovat další pozornost.

Prostředí µVision2

Základní seznámení s prostředím provedeme na příkladě jednoduché aplikace, která rozbliká diodu připojenou na výstupu P1.3 JM. Zdrojové soubory navrhované aplikace jsou sdruženy v rámci projektu. Projekt obsahuje informace potřebné pro překladač, linker, debugger, atd. Nový projekt vytvoříme v menu *Project → New Project*, kde zadáme název projektu např. *LED*. Následně jsme vyzváni k volbě použitého jednočipového mikropočítače a vybereme *Atmel → T89C51RD2*. V okně projektu se objeví položky *Target 1* a *Source Group 1*, které lze pro větší názornost libovolně přejmenovat. Myší zvolíme položku *Target 1* a přejmenujeme na *BLED*. Obdobně změníme *Source Group 1* na *SRC*.

Vytvořený projekt dosud, ale neobsahuje žádné zdrojové soubory. V menu *File → New* vytvoříme nový soubor, který uložíme přes *File → Save* a zadáme jméno souboru *led.c*. Vybereme položku *SRC* v okně projektu, zmáčkneme pravé tlačítko a v zobrazeném roletovém menu zvolíme *Add Files to Group 'SRC'*. Označíme soubor *led.c* a postupně zmáčkneme tlačítka *Add* a *Close*.

Do zdrojového souboru led.c napíšeme následující kód programu, který rozbliká diodu připojenou na výstupu P1.3.

```
#include <89C51RD2.H>                  // header file for AtmelWM T89C51RD2
#define LD1      P1_3                  // LED attached to pin P1.3

void main( void )
{ unsigned int i = 0;

  while(1)
  { LD1 = ~LD1;                        // toggle LED
    while(--i);                        // small delay
  }

}
```

Zdrojový kód obsahuje některé specifické části typické pro jednočipové aplikace napsané v programovacím jazyku C. Na prvním řádku připojujeme k zdrojovému souboru hlavičkový soubor s popisem registrů JM T89c51RD2. Funkce main nemá žádné vstupní parametry a ani nepředává dále návratovou hodnotu. Aplikace napsaná pro JM nesmí nikdy ukončit funkci main. Ukončení by mělo za následek vykonání instrukcí uložených v paměti programu na navazujících adresách a tím zcela nepředvídatelné chování celého programu.

Před samotným zahájením sestavení programu a jeho dalším laděním můžeme v menu *Project → Options for Target 'LED'* upřesnit jednotlivé parametry překladu a výsledného kódu programu. V záložce *Output* zaškrtneme *Create HEX File*, ostatní parametry ve všech záložkách můžeme ponechat pro začátek jak jsou přednastaveny.

Sestavení programu provedeme v menu *Project → Build target* čímž dojde k vytvoření několika souborů s informacemi o výsledcích překladu.

Jméno souboru	Popis
LED	vstupní data pro debugger
LED.HEX	sestavený kód programu
LED.M51	tabulka symbolů
LED.OPT	aktuální nastavení μVison2 (otevřené soubory, velikost oken, …)
LED.UV2	soubor projektu
LED.C	zdrojový kód programu
LED.LST	výpis programu po provedení sestavení

Ladění programu spustíme v menu *Debug → Start/Stop Debug Session*. Krokování programu lze provádět na úrovni zdrojového kódu jazyka C nebo přímo po jednotlivých instrukcích v jazyce symbolických adres. Přepínání mezi jednotlivými okny se provádí v menu *View* popř. kliknutím myši na patřičnou ikonu. Do programu lze libovolně umisťovat body přerušení (breakpoints).

Klávesová zkratka	Popis
F5	GO … spuštění programu
F11	STEP … krokování po instrukcích
F10	STEP OVER … automatické vykonání vnořených funkcí
CTRL - F11	STEP OUT OF CURRENT FUNCTION … dokončení prováděné funkce
CTRL - F10	RUN TO CURSOR LINE … vykonání programu po aktuální pozici kurzoru

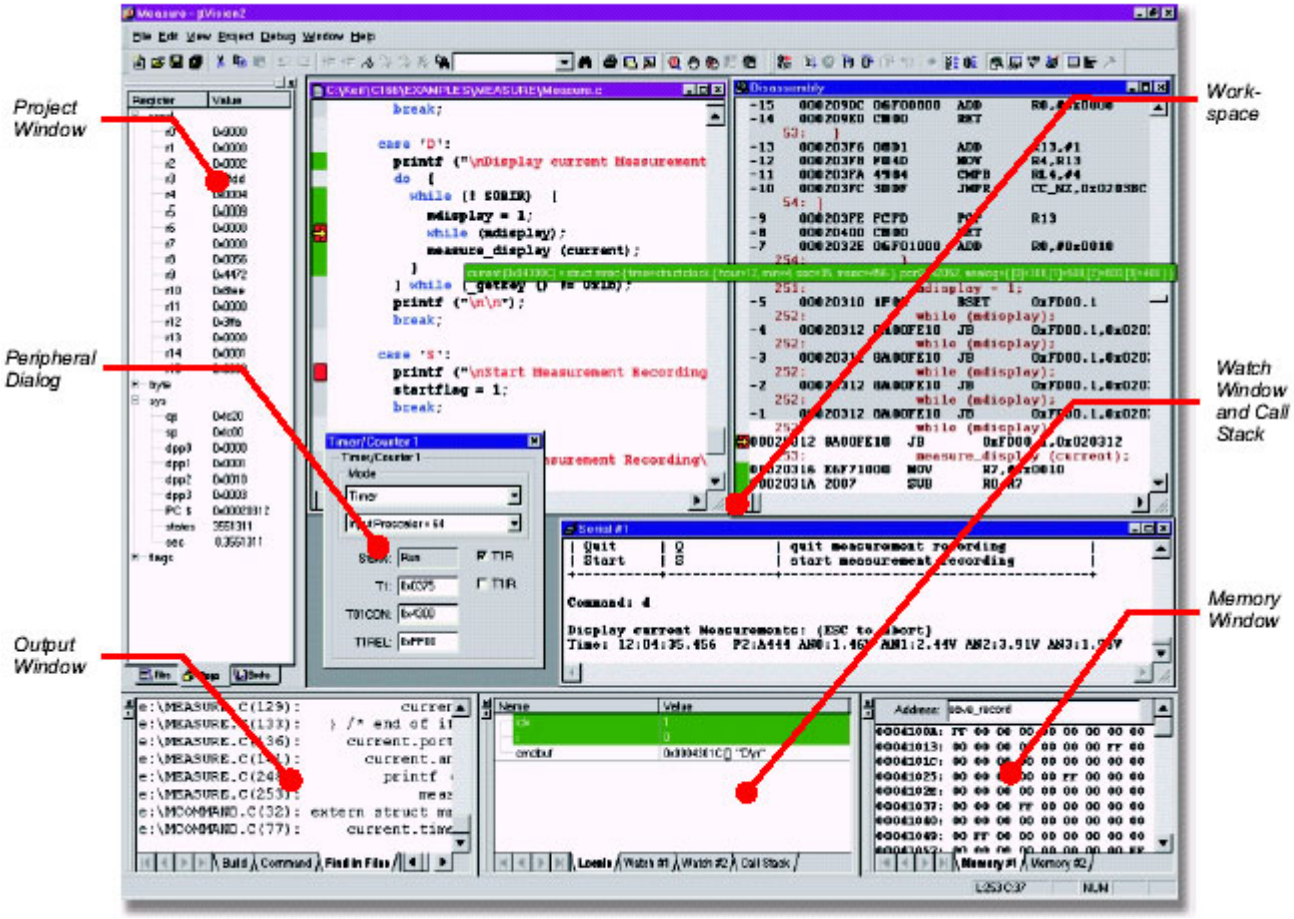
V režimu ladění máme k dispozici přístup ke všem periferiím integrovaných na JM v menu *Peripherals*. Provedení jakýchkoliv změn ve zdrojovém kódu se projeví teprve po ukončení režimu ladění a opětovném sestavení celého programu.

Správnost funkce programu ověříme následujícím postupem. Zobrazíme bránu P1 ke které je připojena led dioda *Peripherals → I/O Ports → Port 1.* Následně v menu *View* zatrhneme položku *Window Periodic Update* a zmáčkneme klávesu *F5*. Dojde k automatickému spuštění krokování programu. Na portu P1.3 sledujeme periodickou změnu stavu výstupu.

Nyní již nic nebrání v naprogramování T89c51RD2 a spuštění aplikace na reálném JM. Postup ISP programování T89c51RD2 je detailně uveden v textu zabývajícím se problematikou použití programu RD2-Flasher. Po úspěšném naprogramování a vynulování JM se můžeme těšit na blikající led diodu což bylo cílem našeho úvodního příkladu.

Popis prostředí µVision2 IDE

Tato kapitola obsahuje stručný popis vývojového prostředí µVision2. V žádném případě si neklade za cíl detailní popis jednotlivých funkcí, který lze nalézt v originální dokumentaci. Text má posloužit pro získání základního přehledu k čemu lze toto vývojové prostředí použít a kde najít tu či onu funkci..



µVision2 IDE V2.30

Vývojové prostředí Keil µVision2 v sobě sdružuje management projektu, editaci zdrojových kódů, překladač, linker, debugger a další části v jeden kompaktní celek. Instalace programu je obdobná běžně používaným postupům známým z prostředí MS Windows a není třeba jí věnovat další pozornost.

Celé prostředí se ovládá z menu a zapnutých lišt nástrojů. Zdrojové soubory navrhované aplikace jsou sdruženy v rámci projektu, který tvoří základ všeho co se děje v µVision2 IDE. Projekt obsahuje potřebné informace pro překladač, linker, debugger, atd.

Menu File

Menu File obsahuje základní operace pro práci se soubory (načtení, uložení, přehled naposledy otevřených, tisk, atd.). Dále lze zde provádět manipulaci s databází podporovaných JM (přidání nového JM, modifikace vlastností stávajícího JM).

Menu Edit

Menu Edit obsahuje funkce pro editaci zdrojových textů programu (vkládání, kopírování, historie změn, vyhledávání, atd.).

Menu View

V menu View lze zapnout jednotlivé lišty nástrojů, zobrazovaná okna a nastavit celkový vzhled vývojového prostředí (barevné zobrazení syntaxe, klávesové zkratky, atd.).

Menu Project

V tomto menu se provádí operace s projekty. Menu je rozděleno to čtyřech logických částí. První část je určeno pro vytvoření nového projektu, otevření, zavření a importu stávajících projektů. Druhá část nastavuje vlastnosti jednotlivých částí projektu (typ JM, použitý paměťový model, atd.). Třetí část slouží k překladu zdrojových kódů a sestavení výsledné aplikace. Čtvrtá část zobrazuje naposledy otevřené projekty.

Menu Debug, View, Peripherals

V menu Debug jsou umístěny všechny funkce, které mají spojitost s laděním aplikace v simulátoru. Provádí se zde krokování aplikace, práce s body přerušení, modifikace instrukcí, analýza výkonu navržené aplikace. Těsně s tímto menu jsou spjata menu View a Peripherals. Položky v menu Peripherals se objevují v závislosti na integrovaných periferiích JM, které lze během krokování volně nastavovat. Menu View umožňuje zobrazit (přepínat) jednotlivá okna použitá při krokování (výstup sériové linky, okno výpisu paměti, atd.).

Menu Tools

Menu Tools slouží k nastavení externích programů. První skupinu tvoří programy pro validaci zdrojových kódů programu PC - Lint a Siemens Easy Case. Další programy si může uživatel libovolně přidat (např. RD2-Flasher).

Menu SVCS

V menu SVCS se nastavuje spolupráce µVision2 se systémem SVCS (Software Version Control System), který je používán pro správu zdrojových kódů během vývoje aplikace (vývojové stromy, správa verzí programu, atd.).

Menu Window, Help

V menu Window se provádí všechny operace s okny aplikace. Menu Help poskytuje nápovědu a informace o použitém vývojovém prostředí.

SDCC (Small Device C Compiler) - 2.3.2 (Jul 14 2002)

SDCC je volně dostupný ANSI C kompilátor pro procesory 8051, Z80, AVR, DS390 a PIC. Ve své podstatě SDCC představuje kompilátor jazyka C, assembler a linker což umožňuje jednoduchou přenositelnost na jiné platformy (Windows, Linux, …). Není k dispozici integrované vývojové prostředí podobné μVision2. Toto vede k zcela odlišné filozofii používání celého programu při vývoji jednočipových aplikací a je velice podobné programátorským technikám známým z prostředí OS rodiny UNIX. Všechny programy jsou ovládány pomocí parametrů z příkazové řádky a je na uživateli samotném jakou metodu zvolí pro sestavení výsledného programu a správu projektu.

Instalace SDCC

Domovská stránka projektu je na adrese http://sdcc.sourceforge.net/. Zde lze volně získat poslední verzi kompilátoru ve formě zdrojových kódů nebo předkompilovanou distribuci pro OS Windows a Linux. V dalším textu budeme vycházet z instalace SDCC verze 2.3.2 (Jul 14 2002) na OS Windows.

SDCC vyžaduje striktně instalaci do adresáře *c:\sdcc*, kde se vytvoří adresáře *bin* a *share*. Adresář *bin* obsahuje všechny spustitelné soubory a musí být uveden v rámci proměnné prostředí *PATH*. Pokud je vše správně nastaveno, tak po zadání příkazu *sdcc –v* se vypíše následující text

```
SDCC : mcs51/gbz80/z80/avr/ds390/pic14/TININative/xa51 2.3.2 (Jul 14 2002) (UNIX)
```

Adresář *share* obsahuje programovou dokumentaci, hlavičkové soubory a knihovny. Veškerá dokumentace uvádí, že hlavičkové soubory a knihovny jsou standardně uloženy v adresářích *c:\sdcc\share\sdcc\include* a *c:\sdcc\share\sdcc\lib*. Bohužel to neplatí u verze 2.3.2, která tyto soubory hledá v adresářích *c:\sdcc\sdcc\include* a *c:\sdcc\sdcc\lib*. Chybu lze odstranit přejmenováním patřičných adresářů nebo použitím direktiv kompilátoru *-I* a *-L*. Do adresáře *include* doplníme hlavičkový soubor 89c51rd.h pro JM T89c 51RD2, který je součástí vzorových příkladů pro SDCC kompilátor.

Tvorba programů v SDCC

Úvodní seznámení s SDCC si ukážeme na příkladu blikající diody. Zdrojový kód programu se zcela shoduje s příkladem použitým u Keil C51. Překlad programu provedeme příkazem

```
sdcc -c -Ic:\sdcc\share\sdcc\include --stack-after-data --model-small led.c
```

Význam jednotlivých parametrů

-c … proveď pouze překlad zdrojového souboru led.c,
-I … explicitní zadání cesty k hlavičkovým souborům,
--stack-after-data … zásobník je uložen v datové paměti až za použitými proměnnými,
--model-small … použij vnitřní datovou paměť procesoru.

Po úspěšném ukončení překladu jsou vytvořeny následující soubory

Jméno souboru	Popis
LED.ASM	přeložený program LED.C do assembleru
LED.LST	výpis přeloženého programu LED.ASM
LED.SYM	tabulka symbolů
LED.REL	relativní soubor, který použije linker pro sestavení výsledného programu

Sestavení programu provedeme příkazem

```
sdcc --Lc:\sdcc\share\sdcc\lib\small --stack-after-data --model-small led.rel.
```

V případě, že provádíme sestavení více relativních modulů je třeba uvést jako první soubor, kde je obsažena funkce main(). Výsledný program je uložen ve formátu Intel-HEX v souboru led.ihx, který pomocí RD2-Flasheru nahrajeme do JM T89c51RD2. Po úspěšném naprogramování a vynulování JM začne na výstupu P1.3 blikat připojená dioda.

Správa projektů

Vypisovat jednotlivé příkazy vždy když chceme přeložit znovu program je velice nepohodlné a nepraktické. Celý tento proces lze automatizovat použitím programu make a patřičného souboru makefile pro daný projekt.

Základním cílem programu make je sestavit soubor v postupných krocích. Jestliže je k finálnímu spustitelnému souboru nutné zpracovat velké množství zdrojových souborů, pak máme možnost po změně jednoho zdrojového souboru znovu spustitelný soubor sestavit, aniž by jsme museli překládat všechny zdrojové soubory. Příkaz make si zaznamenává jména souborů, které je nutno před sestavením přeložit. Dochází tedy ke značné úspoře času, zejména v případě, když takto překládáme velmi rozsáhlé programy. Sled jednotlivých instrukcí prováděných příkazem make je uložen v souboru makefile.

V následujícím příkladu si ukážeme použití příkazu make a strukturu souboru makefile.

Obsah souboru makefile pro projekt blikající diody

```
#############################################################################
#          T89c51RD2  Blinking - LED demo application ver 0.1.0             #
#############################################################################
VER    = 0.1.0 08.08.2002
NAME   = BLED
IHEX   = led.ihx

CC     = sdcc
RM     = @rm
RD2F   = @rd2f
TERM   = @serialterm
CLS    = @cls

COM    = COM1
FBAUD  = 115200
CBAUD  = 19200
PSEN   = 1
ERASE  = 0
VERI   = 0

INC    = -Ic:\sdcc\share\sdcc\include
LIB    = -Lc:\sdcc\share\sdcc\lib\small

LFLAGS = --stack-after-data --model-small --xram-loc 0x0000
CFLAGS = $(LFLAGS) $(INC) -c

image: cmp_msg led.rel
       $(CC) $(LFLAGS) $(LIB) led.rel

cmp_msg:
       @echo ----------------------------------------------------------------
       @echo Compiling $(NAME) project for T89c51RD2 version $(VER).
       @echo ----------------------------------------------------------------

flash_msg:
       @echo ---------------------------------------
       @echo Programming T89c51RD2 microcontroller.
       @echo ---------------------------------------

led.rel: led.c
       $(CC) $(CFLAGS) led.c

flash: flash_msg
       $(RD2F) -f $(IHEX) -c $(COM) -b $(FBAUD) -p $(PSEN) -e $(ERASE) -v $(VERI)

pause:
       @pause

all: image pause flash

clean:
       $(RM) -f *.asm *.cdb *.rel *.lst *.map *.rst *.sym *.lnk *.lib *.bin log.txt
```

V uvedeném souboru makefile si všimněme následujících cílových objektů

- all – překlad, sestavení a naprogramování jednočipového mikropočítače,
- flash – naprogramování jednočipového mikropočítače,
- clean – odstranění temporárních souborů vytvořených během překladu a sestavení.

Jednotlivé cílové objekty voláme z příkazové řádky jako parametry příkazu make

- make all,
- make flash,
- make clean.

Závěr

SDCC je zajímavá varianta, která je určena především zkušenějším uživatelům, kteří již mají nějaké zkušenosti c podobnými kompilátory. Pro začátečníky je například Keil výrazně přívětivější hlavně tím, že obsahuje interní debuger, programy lze trasovat a podobně.

Na druhé straně je SDCC zdarma, zatímco Keil stojí nemalé peníze. Pokud nato máte čas a naučíte se dělat základní věci v Keilu, lze časem přejít celkem úspěšně na SDCC. Ladění aplikace záleží na konkrétních vývojářích, některým stačí displej v aplikaci, jiní mají rádi simulátory, emulátory a další vývojové prostředky. V praxi má překlad v SDCC o něco horší výsledky, než Keil, ale pokud se s ním naučíte pracovat a znát jeho chyby, jsou výsledky srovnatelné.

40 řešených příkladů v C pro Keil i SDCC

Pro studijní účely a pochopení základních principů návrhu jednočipových aplikací s procesory řady 8051 je RD2-kit vybaven sadou příkladů v jazyce C. Ukázkové příklady jsou určeny pro vývojové nástroje Keil C51 evaluation version a SDCC. Porovnáním těchto dvou nástrojů máme možnost si vytvořit konkrétní náhled na problematiku vývoje jednočipových aplikací ve vyšším programovacím jazyce. Naprogramování vytvořené aplikace do JM z PC se provádí metodou ISP přes sériové rozhraní. Tato metoda nevyžaduje žádný další podpůrný hardware a zcela eliminuje použití klasického programátoru.

Kromě zdrojových kódů najdete na CD také .HEX soubory, které lze pomocí Flasheru nebo programu FLIP rovnou nahrát do aplikace a spustit.

Jednotlivé příklady jsou členěny do několika tématických skupin

Úvodní program v C na T89c51RD2

- popis překladu a sestavení jednoduchého programu v jazyce C, naprogramování JM metodou ISP.

Sériový kanál

- nastavení 8-bitové sériové komunikace, vlastní funkce pro čtení a zápis dat přes sériový kanál,
- použití standardních funkcí jazyka C pro formátovaný vstup/výstup dat přes sériový kanál,
- celočíselný kalkulátor.

LCD displej 2 x 16 znaků v 4-bitovém režimu komunikace

- nastavení 4-bitového režimu komunikace, funkce pro čtení a zápis dat na LCD displej,
- uživatelská znaková sada, posuvy textu a animace,
- zasílání příkazů LCD displeji přes sériový kanál,
- přesměrování standardního výstupu funkce printf na LCD a sériový kanál.

Maticová klávesnice 4 x 3

- funkce pro čtení klávesnice, diagnostika klávesnice s výstupem na sériový kanál,
- sdílení datové sběrnice LCD displeje s klávesnicí, výstup dat na LCD.

Systém přerušení

- časovač T2 s obvodovým 16-bitovým přednastavením, obsluha led diody v rutině přerušení,
- hodiny reálného času s časovačem T2,
- obsluha vnějšího přerušení INT0.

Časovač watchdog

- ovládání časovače watchdog a jeho použití při kontrole běhu aplikace,
- řízené nulování JM časovačem watchdog.

Čítačem podporované programovatelné pole PCA, časovač T2

- 8-bitová pulsně šířkové modulace,
- měření délky pulsu,
- programovatelný generátor pulsů.

Paměť programu FLASH

- ověření integrity kódu programu na základě výpočtu kontrolního součtu souvislého bloku paměti FLASH,
- volání API funkcí uživatelského zavaděče pro práci s pamětí FLASH,
- přístup do paměti XAF (eXtra Array Flash).

Paměť EEPROM

- obsluha paměti EEPROM, čtení, zápis a výpis obsahu paměti přes sériový kanál.

Vnější paměť dat XRAM

- nastavení dostupné velikosti vnější paměti dat XRAM,
- test paměťových buněk a výpis obsahu paměti přes sériový kanál.

Speciální funkce procesoru

- vypnutí generování signálu ALE,
- režimy se sníženou spotřebou (Idle Mode, Power-Down Mode),
- programové přepínání násobičky hodin (X2 - Mode).

Diagnostika JM

- ověření funkčnosti jednotlivých bloků JM a připojených periferií.

Pokročilé použití vývojových nástrojů

- vkládání instrukcí jazyka symbolických adres do zdrojového kódu jazyka C,
- sestavení výsledného programu na uživatelem definovaných adresách.

Sériový kanál

Příklad 1.1

Navrhněte program pro komunikaci po sériovém kanálu s přenosovými parametry 19200, N, 8, 1. Napište vlastní funkce pro operace čtení a zápisu dat na sériovém kanálu. Funkčnost demonstrujte na příkladě vysílání textového řetězce zakončeného znakem 00h a jednoduchém emulátoru terminálu (vrací zpět stištěné klávesy).

Sériový kanál přepneme do módu 1 – 8bitový UART. Přenosová rychlost sériového kanálu je volitelná a je dána periodou přetečení časovače T1 a hodnotou bitu SMOD v registru PCON. Časovač T1 pracuje v módu 2 s obvodovým přednastavením a pro přenosovou rychlost platí následující vztah

$$Přenosová\ \ rychlost = \frac{2^{SMOD}}{32} \cdot \frac{F_{OSC}}{12 \cdot [256 - TH1]} \qquad (1)$$

Výpočet hodnoty TH1 časovače T1 pro různé přenosové rychlosti zautomatizujeme pomocí makra bez parametrů (příkaz define) Timer1Mode1ReloadValue. Ve všech následujících příkladech pracujících se sériovým kanálem budeme implicitně používat přenosovou rychlost 19200 baudů.

Obsluha sériového kanálu je řešena bez využití přerušovacího systému. Vysílání a příjem dat provádíme programově pomocí patřičných podprogramů pro vysílání (ser_putch(), ser_puts()) a příjem (ser_getch()).

Sestavení programu a naprogramování JM je shodné s vzorovým popisem uvedeným u programu blikající led diody. K zobrazení dat posílaných po sériovém kanálu použijeme libovolný sériový terminál. Nastavíme přenosové parametry 19200, N, 8, 1 a vypneme řízení toku dat. V případě, že využíváme programové řízení vývodu PSEN při ISP programování je nutné zkontrolovat, aby sériový terminál nenastavoval signál RTS. Nastavením signálu RTS by došlo k uzemnění vývodu PSEN JM během nulování JM a spuštění zavaděče na adrese FC00h místo uživatelské aplikace.

Příklad 1.2

Nahraďte vlastní funkce z příkladu 1.1 za standardní funkce jazyka C pro formátovaný vstup/výstup dat na sériovém kanálu. Použijte funkci printf() a zobrazte numerickou hodnotu čísel datového typu unsigned char, long a int co nejvíce možnými způsoby. Využijte různé typy řídícího řetězce formátu funkce printf().

Pro vstupy a výstupy používáme funkce, které při vstupu čísla načtou celé číslo jako řetězec a převedou ho automaticky do číselné podoby (scanf()) nebo naopak, při výstupu samy konvertují hodnotu číselné proměnné na odpovídající posloupnost číslic (printf()).

Použití standardních funkcí vstupu/výstupu je stejné jak známe z programování na PC. Implicitně se u těchto funkcí předpokládá práce se sériovým kanálem. Drobné odchylky jsou u řídících řetězců formátu, které jsou částečně optimalizované pro použití na platformě x51.

Pokud používáme v SDCC funkci printf() je třeba napsat vlastní funkci putchar() viz příklad 2.4.

Příklad 1.3

Navrhněte program jednoduchého kalkulátoru schopného provádět operace sčítání, odčítání, dělení a násobení s celými čísly. Způsob realizace vstupně/výstupních funkcí volte podle vlastního uvážení.

V případě absence některých standardních funkcí jazyka C implementovaných na platformě x51 musíme využít alternativních řešení. Program kalkulátoru načítá dvě celá čísla a symbol matematické operace. Načtení čísel může obstarat funkce scanf(), která lze však v tomto případě nahradit např. funkcemi sscanf() nebo atoi(). Zobrazení výsledku na sériovém kanálu využívá funkcí popsaných v předchozích příkladech.

LCD displej 2 x 16 znaků v 4 - bitovém režimu komunikace

Příklad 2.1

Navrhněte program pro obsluhu LCD displeje 2 x 16 znaků v 4 - bitovém režimu komunikace. Zaměřte se zejména na správnou inicializaci displeje a programovou kontrolu příznaku BUSY. Napište funkce pro zasílání příkazů a znaků řadiči displeje. Funkčnost demonstrujte na příkladě vysílání textového řetězce zakončeného znakem 00h.

Obsluha displeje v 4 - bitovém režimu komunikace přináší úsporu čtyř datových vodičů D0 až D3. Na druhou stranu využívá složitější komunikační protokol. Displej je připojen na bránu P1 a využívá všech vývodů mimo P1.3. Programově je ošetřeno, že vývod P1.3 není ovlivňován komunikací s displejem a lze ho libovolně použít pro v/v operace.

Použitý LCD displej je osazen řadičem kompatibilním s HD44780. Pro řízení displeje jsou důležité tři základní funkce na kterých založíme naše LCD API rozhraní. Jedná se o inicializaci displeje v 4 - bitovém režimu komunikace, zasílání dat řadiči (příkazy, znaky) a kontrola příznaku BUSY. Detailní popis jednotlivých funkcí je nad rámec tohoto textu. Potřebné informace a průběhy řídících signálů jsou uvedeny v katalogovém listu řadiče HD44780.

V některých případech po ne zcela korektním zapnutí napájecího napětí docházelo k problémům s inicializací displeje a celého JM. Problém byl odstraněn použitím funkce PowerOnReset() detailně popsané v příkladu 5.2.

Příklad 2.2

Příklad 2.1 doplňte o možnost dohrání uživatelské znakové sady. Navrhněte funkce pro plynulý posuv textového řetězce na displeji. Uživatelskou znakovou sadu využijte k jednoduché animaci.

Prvních osm znaků ve znakové sadě lze nastavit, což umožňuje na displeji tvořit různé efekty, grafické symboly, češtinu nebo animace. Použitá zobrazovací matice displeje a nastavený režim určují, zda k zobrazení uživatelsky definovaných znaků bude použit rastr 5 x 7 nebo 5 x 10 bodů.

Definice jednotlivých znaků je uložena v paměti kódu programu. Jedná se celkem o 64 bajtů a znak definuje 8 bajtů, které v paměti řadiče displeje CGRAM na sebe postupně navazují.

Vytvořená animace zobrazuje postupně řetězec, který je posouván na displeji zprava do leva. Vhodnou volbou zobrazované posloupnosti vznikne jednoduchá grafická animace.

Příklad 2.3

Po sériovém kanálu zobrazte tabulku základních příkazů pro ovládání displeje. Uživatel má možnost výběru jednotlivých příkazů, kterými ovládá chování displeje.

Program slouží pro demonstraci vytvořeného API rozhraní displeje. Uživatel vidí jednotlivé popisy API funkcí na sériovém terminálu. Zmáčknutím klávesy přidělené dané funkci vyvolá patřičnou odezvu na displeji.

Příklad 2.4

Využijte možnosti přesměrování standardního výstupu funkce printf() do více zařízení. Funkce printf() pro výpis dat na nejnižší úrovni využívá volání funkce putchar(). Napište vlastní funkci putchar() s možností zobrazení dat na LCD a sériovém kanálu. Funkčnost demonstrujte na příkladu volání funkce printf().

Funkce printf() na základě řídícího řetězce formátu vytváří textový řetězec, který je následně zobrazen pomocí funkce putchar(). Implicitně funkce printf() předpokládá výstup na sériový kanál. V případě, že nahradíme funkci putchar() svou vlastní funkcí naskýtá se nám možnost přesměrovat výstup na další zařízení. Je však třeba mít na paměti následující věc. Při definici vlastní funkce putchar() nesmí dojít k rozporu s deklarací funkce putchar() uvedenou v hlavičkovém souboru stdio.h použitého překladače. Globální proměnnou poté volíme aktuální výstupní zařízení použité funkcí putchar() pro výstup dat.

Maticová klávesnice 4 x 3 tlačítka

Příklad 3.1

Navrhněte program pro čtení maticové klávesnice 4 x 3 tlačítka. Vyzvěte uživatele k zmáčknutí jednotlivých tlačítek a detekované tlačítko zobrazte po sériovém kanálu.

Maticová klávesnice je připojena pomocí sedmi vodičů. K výběru sloupce klávesnice slouží vývody P3.5 až P3.7. Aktuální stav jednotlivých řádků klávesnice čteme na vstupech P1.4 až P1.7. Postupně jednotlivé sloupce přepínáme do stavu log.0 a čtením řádků zjišťujeme zmáčknuté klávesy. Navržené řešení zjišťuje pouze jednu zmáčknutou klávesu. Ověření funkčnosti je demonstrováno na jednoduchém programu, který zmáčknuté klávesy zobrazuje po sériovém kanálu.

Příklad 3.2

Program obsluhy klávesnice upravte tak, aby pro čtení řádků klávesnice využil datové sběrnice použité u LCD displeje v 4 - bitovém režimu komunikace. Funkčnost demonstrujte na příkladu přístupového systému s kódovým zámkem (zadání nového hesla a ověření správnosti hesla) a výstupem na LCD.

U aplikací kde jsme limitováni počtem volných vývodů JM se může uplatnit následující řešení. Data na sběrnici LCD displeje jsou platná pokud se nachází signál ENABLE v log. 1. V opačném případě je sběrnice volně k dispozici ostatním periferiím. Stávající funkce pro obsluhu klávesnice doplníme o instrukci, která přepne signál ENABLE do log. 0 vždy, když provádíme čtení klávesnice. Nechtěnému ovlivňováni komunikace LCD s JM ze strany klávesnice zamezíme přepnutím sloupců klávesnice do stavu log. 1.

Systém přerušení

Příklad 4.1

Navrhněte program obsluhující rutinu přerušení od časovače T2 s obvodovým 16 - bitovým přednastavením. Nastavte periodu blikání led diody 2000 ms. Využijte vstupu P2.0 k prodloužení periody blikání (log. 0 … zvětšení periody).

Přerušení od časovače T2 se vyvolá nastavením příznaku TF2. Výhoda použití časovače T2 v režimu s 16 - bitovým přednastavením je, že současně s přetečením časovače dojde k automatickému přednastavení. Funkce časovače je tím pádem nezávislá na hlavním programu. Přednastavenou hodnotu časovače T2 vypočteme podle následujícího vztahu

$$[TH2, TL2] = 10000h - \frac{F_{OSC} \cdot t}{12} \qquad (2)$$

kde t je čas v sekundách za jak dlouho má dojít k přerušení. F_{OSC} kmitočet použitého oscilátoru v Hz a konstanta 12 je odvozena od počtu period oscilátoru na jeden strojový cyklus.

Definice funkce obsluhy přerušení v jazyce C se vyznačuje použitím atributu interrupt s číslem přerušení. Klíčové slovo using umožňuje volbu použité banky registrů. Pro časovač T2 bude definice funkce obsluhy přerušení vypadat následovně.

```
static void Reload_Timer2_ISR(void) interrupt 5 using 3
{ LD1 = ~LD1;                      // toggle LED every 1000 ms
  TF2 = 0;                         // clear timer 2 overflow flag
}
```

Příklad 4.2

Navrhněte program hodin reálného času úpravou příkladu 4.1. Výstup hodin ve formátu HH:MM:SS zobrazte na LCD nebo sériovém kanálu.

Přesnost hodin je dána dobou mezi dvěmi přetečeními časovače T2. Obsluha přerušení by měla obecně vždy obsahovat jen ty nejnutnější instrukce, např. pro výpočet času postačují proměnné typu unsigned char. Zobrazení času na výstupním zařízení provádíme v hlavní smyčce programu.

Příklad 4.3

Na vstup externího přerušení INT0 připojte tlačítko. Provádějte detekci vstupního signálu externího přerušení INT0 na sestupnou hranu. Vykonání rutiny přerušení INT0 indikujte změnou stavu led diody.

Vnější přerušení /INT0 a /INT1 mohou být vyvolána buď logickou úrovní nula nebo sestupnou hranou na patřičném vstupu JM. Programová obsluha rutiny přerušení je obdobná jako v předchozích dvou příkladech. Při práci s vnějšími přerušeními je důležité jak vypadá průběh vstupního signálu. Konkrétně v našem případě zákmity generované stiskem tlačítka mohou vyvolat nechtěnou žádost o přerušení. Pokud používáme detekci nulové úrovně je třeba, aby tato trvala minimální dobu uvedenou v katalogovém listu výrobce.

Časovač watchdog

Příklad 5.1

Navrhněte program využívající funkce časovače watchdog. Vstupy reálné aplikace simulujte stisknutím klávesy v sériovém terminálu. Včasný stisk klávesy nuluje čítač časovače watchdog v opačném případě se hodnota čítače inkrementuje a může dojít k nulování celého JM.

Časovač watchdog slouží ke kontrole běhu aplikace a nulování JM v případě, že došlo k selhaní programu. Funkci časovače watchdog aktivujeme postupným zápisem hodnot 1Eh a E1h do registru WDTRST. Hodnota čítače časovače watchdog se postupně inkrementuje. Dojde-li k přetečení, JM se automaticky vynuluje. Přetečení můžeme zamezit periodickým zápisem hodnot 1Eh a E1h do registru WDTRST. Periodu časovače watchdog lze prodloužit nastavením dalšího čítače 2^7 v registru WDTPRG.

Příklad 5.2

Navrhněte program využívající informace uvedené v registru PCON příznak Power-Off. Rozlište stav kdy došlo k zapnutí napájecího napětí u JM nebo nulování obvodu nulovacím tlačítkem. Časovač watchdog použijte k programem řízenému nulování JM.
Hodnota příznaku Power-Off se nastaví po zapnutí napájecího napětí do log. 1 a není ovlivněna nulováním JM nulovacím tlačítkem. Této vlastnosti využijeme k detekci stavu po zapnutí napájecího napětí. Spustíme časovač watchdog a po jeho přetečení se automaticky provede cílené nulování JM.

Čítačem podporované programovatelné pole PCA, časovač T2

Příklad 6.1

Navrhněte program využívající funkce čítačem podporovaného programovatelného pole k vytváření 8 - bitové pulsně šířkové modulace. Výstup PWM signálu přiveďte na led diodu P1.3. Měňte hodnotu střídy a na led diodě pozorujte postupnou změnu svítivosti.
PCA se skládá z 16 - bitového čítače/časovače a pěti 16 - bitových komparačních nebo záchytných modulů. Čítač/časovač je společný pro všechny moduly. Jednotlivé moduly lze provozovat v komparačním režimu(16 - bitový časovač, rychlý výstupní mód, pulsně šířkový modulátor) nebo záchytném režimu (měření délky pulsů).
Samotný proces generování PWM signálu spočívá v nastavení patřičného modulu do komparačního režimu. Pulsně šířkový generátor pracuje jako 8 - bitový čítač s obvodovým přednastavením. Po přetečení je registr CCAPLn obnoven z registru CCAPHn. Změnou hodnoty CCAPHn lze plynule nastavovat střídu generovaného signálu.

Příklad 6.2

Navrhněte program využívající funkce čítačem podporovaného programovatelného pole k měření délky pulsů. Zdrojem pulsů je teplotní čidlo SMT160 - 30 převodník teplota/střída. Změřené údaje numericky zpracujte a výslednou teplotu zobrazte na displeji.

V záchytném režimu lze nastavit požadavek o přerušení od náběžné, sestupné nebo obou hran signálu připojeného na vstup PCA modulu. Společně s požadavkem o přerušení dojde k zápisu obsahu čítače CH, CL do registrů CCAPnH a CCAPnL.

Teplotní čidlo SMT160 - 30 připojíme na vstup PCA modulu 0. V rutině přerušení ukládáme hodnotu registrů CCAP0H a CCAP0L do připravené struktury popisující obdélníkový puls. Na základě těchto údajů vypočteme podle následujícího vztahu aktuální teplotu t [° C] a zobrazíme ji na displeji.

$$t = \frac{1446 \cdot T_1 - 681 \cdot T_0}{T_1 + T_0} \qquad (3)$$

T_1 je stav log. 1 a T_0 stav log. 0

Příklad 6.3

Navrhněte program využívající funkce časovače T2 ke generování pulsů na výstupu P1.0. Zadaný kmitočet signálu čtěte ze sériového kanálu. Zobrazte hodnotu požadovaného a reálně generovaného kmitočtu na výstupu P1.0. Vzniklou odchylku zdůvodněte.

Časovač T2 přepneme do režimu generování pulsů (16 - bitový čítač s obvodovým přednastavením). Kmitočet generovaný na výstupu P1.0 je dán vztahem 4

$$F_{OUT} = \frac{F_{OSC}}{4 \cdot [65536 - [RCAP2H, RCAP2L]]} \qquad (4)$$

Zadanou hodnotu kmitočtu dosadíme do vztahu 4 a vypočteme hodnotu registrů RCAP2H, RCAP2L. U vyšších kmitočtů dochází k numerické chybě při zaokrouhlování na celé 16 - bitové číslo. To má za následek vznik odchylky mezi zadaným a generovaným kmitočtem.

Příklad 6.4

Časovač/čítač T0, T2 - měření kmitočtu
Navrhněte program využívající funkce čítače T0 pro čítání externích pulsů na vstupu P3.4 JM. Zdroj pulsů realizujte časovačem T2 ve funkci generátoru pulsů na výstupu P1.0, který připojte na vstup P3.4. Vypočtěte hodnotu kmitočtu a periodu změřeného signálu a zobrazte na sériovém kanálu. Maximální kmitočet, který lze měřit (přivést) na vstupu P3.4 je roven hodnotě 1/24 použitého oscilátoru u JM. Měření provádějte pro kmitočtový rozsah 100 Hz – 750 kHz.

Příklad 6.5

Čítačem podporované programovatelné pole PCA, časovač T2 - měření periody

Navrhněte program využívající funkce čítačem podporovaného programovatelného pole k měření délky pulsů. Zdroj pulsů realizujte časovačem T2 ve funkci generátoru pulsů na výstupu P1.0, který připojte na vstup PCA P1.3. Vypočtěte hodnotu kmitočtu a periodu signálu změřeného signálu a zobrazte na sériovém kanálu. Měření provádějte pro kmitočtový rozsah 100 Hz – 10 kHz.

Příklad 6.6

Systém přerušení - Alarm
Navrhněte program spínající čtyři výstupy JM po uplynutí přednastavené doby u jednotlivých výstupů. Využijte hodin reálného času z příkladu 4.2. pro časovou základnu celého zařízení. Režim nastavení hodin aktivujte zmáčknutím klávesy „*“ na maticové klávesnici. Nastavení jednotlivých alarmů proveďte klávesou „#“ a patřičným číslem alarmu (0 … 3). Aktuální stav jednotlivých alarmů a čas zobrazte na LCD.

Příklad 6.7

Časovač/čítač T0, T2 - měření kmitočtu
Navrhněte program využívající funkce čítače T0 pro čítání externích pulsů na vstupu P3.4 JM. Zdroj pulsů realizujte časovačem T2 ve funkci generátoru pulsů na výstupu P1.0, který připojte na vstup P3.4. Vypočtěte hodnotu kmitočtu a periodu změřeného signálu a zobrazte na sériovém kanálu. Maximální kmitočet, který lze měřit (přivést) na vstupu P3.4 je roven hodnotě 1/24 použitého oscilátoru u JM. Měření provádějte pro kmitočtový rozsah 100 Hz – 750 kHz.

Příklad 6.8

Čítačem podporované programovatelné pole PCA, časovač T2 - měření periody
Navrhněte program využívající funkce čítačem podporovaného programovatelného pole k měření délky pulsů. Zdroj pulsů realizujte časovačem T2 ve funkci generátoru pulsů na výstupu P1.0, který připojte na vstup PCA P1.3. Vypočtěte hodnotu kmitočtu a periodu signálu změřeného signálu a zobrazte na sériovém kanálu. Měření provádějte pro kmitočtový rozsah 100 Hz – 10 kHz.

Paměť programu FLASH

Příklad 7.1

Navrhněte program ověření integrity kódu programu na základě výpočtu kontrolního součtu souvislého bloku paměti FLASH. K tomuto účelu využijte funkci programu RD2 - Flasheru s parametrem –s. Algoritmus výpočtu kontrolního součtu je shodný s algoritmem použitým u souborů typu Intel - HEX. Čtení paměti FLASH provádějte pomocí ukazatele do paměti kódu programu.

Program RD2 - Flasher umožňuje výpočet kontrolního součtu souvislého bloku paměti FLASH. Vypočtená hodnota je uložena na adrese následující za zabezpečeným blokem paměti. Obsah paměti je v pořádku, pokud suma všech bajtů v kontrolovaném bloku paměti FLASH a kontrolního součtu je rovna nule.

Rozšíření jazyka C na platformě 8051 umožňují mimo jiné optimalizovat práci s ukazateli. Programátor může určit paměť, kde je ukazatel uložen a paměť do které odkazuje. Ukazatel uložený v interní paměti JM ukazující do paměti programu definujeme následovně.

```
Keil C51        unsigned char code *data ptr;
SDCC   code unsigned char *data ptr;
```

Příklad 7.2

Navrhněte program komunikující pomocí API rozhraní s uživatelským zavaděčem pro práci s pamětí FLASH. Zobrazte všechny dostupné informace Manufacturer ID, ID1, ID2, … na sériový kanál.

API rozhraní uživatelského zavaděče je detailně popsáno v katalogovém listě JM T89c51RD2 na straně 61 až 63. Princip používání jednotlivých API funkcí spočívá v naplnění patřičných registrů (R1, DPH, DPL, ACC) a volání funkce na adrese FFF0h. Legitimní řešení předávání parametrů v registrech JM si vynutilo napsání podprogramu v jazyce symbolických adres. Tento podprogram připojíme k hlavnímu programu v jazyce C odkud ho následně voláme.

Konvence volání a předávání parametrů se v jednotlivých vývojových nástrojích liší. Zde se osvědčil následující trik. V jazyce C napíšeme funkci, která má vstupní a výstupní parametry shodné s plánovanou funkcí v jazyce symbolických adres. Překladem této funkce získáme funkční kód v jazyce symbolických adres, který doplníme o námi požadované operace a jsme hotovi.

Příklad 7.3

Navrhněte program přistupující přímo do paměti XAF (eXtra Array Flash). Zobrazte obdobně jako v příkladě 7.2 dostupné informace na sériový kanál.

Paměť XAF má velikost 128 B a je zcela oddělena od paměti programu. XAF obsahuje informace používané během ISP režimu programování. Tyto informace lze číst buď voláním API rozhraní zavaděče (viz příklad 7.2) nebo přímým přístupem do paměti FLASH. Čtení a zápis paměti FLASH je řízeno registrem FCON. Adresa jednotlivých údajů v paměti XAF je uvedena katalogovém listu JM T89c51RD2 na straně 68 a 69.

Paměť EEPROM

Příklad 8.1

Navrhněte program pracující s integrovanou pamětí EEPROM na čipu JM. Napište funkce pro čtení a zápis dat do paměti EEPROM. Obsah paměti zobrazte pomocí jednoduchého prohlížeče paměti s výstupem na sériový kanál.
Integrovaná paměť EEPROM je přístupná na adresách 0000h až 07FFh a překrývá se s vnější datovou pamětí XRAM. Výběr paměti EEPROM provedeme nastavením bitu EEE v registru EECON. Následně lze libovolně přistupovat do paměti EEPROM instrukcí MOVX. Registr EETIM nastavuje časování vnitřních operací paměti EEPROM v závislosti na připojeném krystalu. Zápis do paměti lze provádět po jednotlivých bajtech v rámci jedné stránky nebo po stránkách (64 B). Operaci zápisu potvrdíme sekvenčním zápisem hodnot 50h a A0h do registru EECON. Ukončení zápisu je ze strany hardwaru indikováno vynulováním příznaku EEBUSY.

Poznámka: *Seznam chyb JM T89c51RD2 uveřejněný firmou Atmel uvádí, že je třeba provést zápis dat třikrát za sebou, aby došlo ke korektnímu naprogramování paměti EEPROM.*

Vnější paměť dat XRAM

Příklad 9.1

Navrhněte program pracující s integrovanou pamětí XRAM na čipu JM. Nastavte skutečnou velikost dostupné paměti XRAM. Otestujte jednotlivé paměťové buňky a adresové vodiče dekodéru paměti. Obsah paměti zobrazte pomocí jednoduchého prohlížeče paměti s výstupem na sériový kanál.

Velikost integrované paměti XRAM na čipu je 1024 B. Po zapnutí/nulování JM je však k dispozici pouze 768 B (kompatibilní režim s T87c51RD2). V registru AUXR lze programově nastavit velikost dostupné paměti. Jednotlivé paměťové buňky ověříme postupným zápisem a následným čtením konstant AAh a 55h. Přístup do paměti je řešen pomocí makra XBYTE, které slouží k zápisu/čtení jednotlivých položek paměti XRAM. Obdobně lze použít další makra (CBYTE, DBYTE, PBYTE, XWORD, …) pro přístup do ostatních paměťových prostorů JM.

Speciální funkce procesoru

Příklad 10.1

Navrhněte program schopný vypnout/zapnout generování signálu ALE na příslušném výstupu JM.

Pro řízení vnější paměti je JM vybaven řídícími signály ALE (pro zápis spodní poloviny platné adresy A0 až A7), /PSEN (pro čtení z vnější paměti programu), /RD a /WR (pro čtení nebo zápis z vnější datové paměti). Signál ALE je periodicky generován (F_{OSC}/6) nezávisle na tom jestli je připojena externí datová paměť programu nebo dat. Nastavením bitu AO v registru AUXR lze vypnout automatické generování signálu ALE a tím snížit generované rušení. Signál ALE se poté generuje pouze během provádění instrukcí MOVX a MOVC.

Příklad 10.2

Navrhněte program využívající režimů se sníženou spotřebou (Idle Mode, Power - Down Mode). Na led diodě P1.3 zobrazte aktuální stav JM (svítící dioda indikuje probíhající program a naopak). Použijte vstup P2.0 v log. 0 k aktivaci režimu snížené spotřeby. Tlačítkem připojeným na vstup externího přerušení /INT0 jej ukončete.

U aplikací nevyžadujících soustavnou činnost a mají požadavek na nízkou spotřebovanou energie, je možné využít režimů se sníženou spotřebou. JM T89c51RD2 je vybaven dvěmi režimy se sníženou spotřebou (Idle Mode, Power - Down Mode), do kterých lze JM uvést nastavením příslušných bitů v registru PCON. Instrukce nastavující patřiční režim je poslední instrukcí provedenou před přechodem do režimu se sníženou spotřebou. Tento režim v našem případě opustíme po výskytu požadavku o vnější přerušení na vstupu /INT0.

Příklad 10.3

Navrhněte program obsluhující rutinu přerušení od časovače T2 s obvodovým 16 - bitovým přednastavením. Nastavte periodu blikání led diody 2000 ms. Využijte vstupu P2.0 k programovému přepínání násobičky hodin (X2 - Mode … log. 0 … zvětšení periody).

Postup nastavení časovače T2 s obvodovým 16-bitovým přednastavením je popsáno v příkladu 4.1. Programové přepínání hodin se provádí bitem X2 v registru CKCON. U jednotlivých periferií JM lze selektivně nastavit počet period (6 nebo 12) oscilátoru na jeden strojový cyklus.

Diagnostika RD2 Kitu

Příklad 11.1

Navrhněte program ověřující funkčnost jednotlivých bloků JM a připojených periferií.

Zaměřte se na následující testy :

- sériový kanál čtení a zápis dat,
- led dioda na výstupu P1.3,
- integrita kódu programu chráněná kontrolním součtem,
- externí vstup přerušení INT0 s tlačítkem,
- LCD displej 2 x 16 znaků v 4 - bitovém režimu komunikace,
- maticová klávesnice 4 x 3 tlačítka.

Jednotlivé body zadání jsou detailně probrány v rámci dosavadních příkladů zabývajících danou problematikou. Záměrem tohoto příkladu je zopakovat a shrnout znalosti získané v předešlých příkladech. Zároveň vytvoříme testovací nástroj, který ověřuje funkčnost základních bloků JM a připojených periferií. Aplikace je vhodná pro situace, kdy máme podezření na nefunkčnost některé periferie.

Pokročilé použití vývojových nástrojů

Příklad 12.1

Navrhněte program určující paritu jednobajtového čísla zadaného na sériovém kanále. K zjištění parity využijte příznaku parity uloženého ve stavovém slově. Je-li v akumulátoru lichý počet jedniček, potom příznak parity je nastaven na P = 1. Práci s akumulátorem a určení parity proveďte v jazyku symbolických instrukcí, který vložte do zdrojového kódu programu jazyka C.

Keil C51

V okně projektu vybereme zdrojový kód programu *main.c*, zmáčkneme pravé tlačítko a zvolíme položku *Options for file main.c*. Zobrazené okno obsahuje nezaškrtnuté volby *Generate Assembler SRC file* a *Assemble SRC file.* Tyto volby je nutné zaškrtnout, aby došlo k překladu programu obsahující následující pragma pro vkládaný kód jazyka symbolických adres.

```
#pragma asm

zdrojový kód programu v jazyku symbolických adres

#pragma endasm
```

Sestavení programu vyžaduje navíc manuální doplnění všech použitých knihoven. Toto provedeme v okně projektu, kde postupně vložíme použité knihovny z adresáře */C51/LIB*. V našem případě se jedná o knihovnu *C51S.LIB*.

SDCC

Použití vkládaného kódu jazyka symbolických adres je v překladači SDCC mnohem jednodušší. Ve zdrojovém kódu jazyka C stačí pouze uvést klíčová slova _asm a _endasm;, která ohraničují kód v jazyce symbolických adres. Překlad a sestavení programu není třeba nikterak modifikovat.

```
_asm

zdrojový kód programu v jazyku symbolických adres

_endasm;     // je třeba zakončit středníkem
```

Při použití vkládaných instrukcí jazyka symbolických adres je třeba dbát na to, že se jedná o zásah do struktury programu napsaného v jazyce C. Neopatrné použití může vést k fatálním následkům.

Příklad 12.2

Navrhněte program vypisující textový řetěz na sériový kanál. Zaměřte se na fázi sestavení programu linkerem kdy dochází k přidělení relativního kódu absolutním adresám. V závislosti na možnostech použitých vývojových nástrojů nastavte počáteční adresu programu, uložení textových řetězců na stanovenou adresu v paměti programu, atd.

Keil C51

Linker BL51 umožňuje cílené řízení sestavení výsledného programu. Lze specifikovat adresy kde mají být uloženy jednotlivé funkce, moduly, … V menu *Project → Options for Target …→ BL51 Locate* pole *CODE* uvedeme názvy jednotlivých funkcí(modulů) s patřičnou počáteční adresou. Při zadávání musíme použít názvy uvedené v souboru .M51 (tabulka symbolů). Např. funkci ser_init() umístíme na adresu 900h následujícím zápisem ?PR?SER_INIT?MAIN(0900h).

SDCC

Možnosti SDCC při sestavování programu jsou omezeny pouze na určení počáteční adresy odkud bude uložen výsledný kód programu. Adresa je předána linkeru parametrem *--code-loc* s udanou počáteční adresou.

Závěr

Zdrojové kódy popsaných aplikací, najdete na CD k RD2 Kitu, které bude možné koupit také samostatně v internetovém obchodu HW serveru, nebo v běžné distribuci.

Součástí CD je i sériový kód, který vám zpřístupní aktualizovanou databázi zdrojových kódů a dalších příkladů pro RD2 Kit.

Další informace o této problematice najdete na těchto internetových adresách :

- **www.HW.cz** - HW server : Výrobce RD2 Kitu, stránky celého projektu..
- **www.keil.com** - Výrobce Keil µVision2 v jehož demoverzi jsou popsané příklady kompilovatelné..
- **sdcc.sourceforge.net** – Domovská stránka projektu SDCC

Kolektiv autorů z HW serveru