ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

PROVOZNĚ EKONOMICKÁ FAKULTA

Katedra informačních technologií

BAKALÁŘSKÁ PRÁCE

Programování 8-bitového mikrokontroléru Atmel

Vedoucí bakalářské práce:
Ing. Martin Havránek

Autor:
Martin Neruda © 2009

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Programování 8-bitového mikrokontroléru Atmel" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne _____

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Havránkovi za odborné vedení, cenné rady, trpělivost a ochotu při vedení mé bakalářské práce. Velké poděkování patří také rodině a přítelkyni za podporu.

Programování 8-bitového mikrokontroléru Atmel

Programming of Atmel 8-bit microcontroller

Souhrn

Tato práce se zabývá 8bitovými mikrokontroléry, konkrétně typem ATmega16 od firmy Atmel. Zachycuje vnitřní architekturu, typy periférií a možnosti programování v jazyce C. Je vysvětlen způsob kompilace programu a nároky kladené na kompilátor a optimalizaci kódu. Program v jazyce C je demonstrován na kuchyňské minutce, které se věnuje poslední část práce.

Summary

This thesis is dealing with 8-bit microcontrollers, specifically type ATmega16 made by Atmel Corporation. It describes its architecture, types of peripherals and possibilities of programming in C programming language. There is also explained the way of compiling program and requirements placed on compiler and on code optimization. Program written in C programming language is demonstrated on kitchen minute timer, which is described in final chapters of this paper.

Klíčová slova: mikrokontrolér, kompilátor, registr, AVR, ATmega16, vnitřní paměť

Keywords: microcontroller, compiler, register, AVR, ATmega16, internal memory

Obsah

1 Úvod

S mikrokontroléry se v dnešní době nepřímo setkáváme téměř na každém kroku našeho života. Jsou v dálkových ovladačích k televizním přístrojům, v mobilních telefonech, v zabezpečovacích systémech, termostatech a nespočetné řadě dalších přístrojů. Uplatní se všude tam, kde je zapotřebí něco řídit, kontrolovat nebo vyhodnocovat. Hlavní výhodou mikrokontrolérů je jejich velikost, která se díky technickému vývoji stále zmenšuje. Vzhledem k velkému množství firem zabývajících se jejich vývojem a výrobou si můžeme vybrat vhodný typ, který se pro zvolenou aplikaci nejlépe hodí z hlediska rychlosti, velikosti paměti, počtem vstupů/výstupů, množstvím periférií atd..

Dříve se k programování malých 8bitových mikrokontrolérů používal výhradně jazyk symbolických adres – assembler, kvůli jeho blízkosti se strojovým kódem. Výhodou takového programu byla rychlost, velikost paměti, kterou zabíral, i to, že programátor přímo ovlivňoval každou instrukci vykonávanou programem. Nevýhodou byla špatná čitelnost takového programu a nepřenositelnost na jiné platformy.

Proto se postupně začaly používat programovací jazyky vyšší úrovně. Např. jazyk C, který má největší praktické využití v aplikaci mikrokontrolérů. Pro efektivní využití však bylo nutné, aby výrobci přizpůsobili architekturu mikrokontrolérů jeho požadavkům. U softwaru je největší důraz kladen na kompilátor a optimalizaci kódu, která začíná již u programátora. Nadbytečný kód vygenerovaný z jazyka C (oproti stejnému programu napsaném v assembleru) se téměř eliminuje při použití vhodné architektury, kompilátoru a správně napsaného programu.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je především vysvětlit, co je to mikrokontrolér, jak funguje a jaké je jeho vnitřní uspořádání. Práce dále prezentuje programování 8bitových mikrokontrolérů v jazyce C a nároky kladené na kompilátor tohoto jazyka. Praktické využití mikrokontrolérů bude demonstrováno na jednoduché kuchyňské minutce.

2.2 Metodika práce

Teoretické poznatky této bakalářské práce budou zpracovány na základě prostudování odborné literatury a dostupných online zdrojů.

Úvodní část bude věnována struktuře mikrokontrolérů, používaným architekturám a jejich kladům a záporům. Dále bude uvedena stručná historie firmy Atmel Corporation a představení jejich 8bitového mikrokontroléru ATmega16 s architekturou AVR, na který bude práce zaměřena. U ATmegy16 bude vysvětlen princip zpracování instrukcí, struktura vnitřních pamětí, vstupně/výstupní periferie a registry.

Následující kapitola nastíní funkci integrovaných vývojových prostředí a přiblíží využití programovacího jazyka C pro programování 8bitových mikrokontrolérů. Bude zde uveden průběh kompilace a optimalizace programu a stručný vývoj jazyka C.

V závěru bude programování mikrokontroléru ATmega16 demonstrováno na obvodu kuchyňské minutky. Schéma obvodu bude doplněno rozborem jednotlivých funkčních součástí a okomentovaným zdrojovým kódem napsaným v jazyce C. K vývoji projektu bude použito integrované vývojové prostředí.

3 Mikrokontrolér

3.1 Co je to mikrokontrolér

Mikrokontrolér, MCU (Micro Controller Unit) nebo také jednočipový počítač, je programovatelná polovodičová součástka tvořená z velkého počtu tranzistorů integrovaných na malé křemíkové destičce. Křemíkový čip je hermeticky uzavřen v pouzdru čtvercového nebo obdélníkového tvaru a z čipu jsou jednotlivé vstupy a výstupy vyvedeny na kontakty (piny) na pouzdru. Pouzdro chrání křemíkový čip před poškozením, zvyšuje jeho mechanickou odolnost a může plnit i funkci chlazení.

Mikrokontrolér představuje spojení mikroprocesoru s vnitřní pamětí (Flash, RAM, ROM, SRAM, EPROM, EEPROM) a periferními obvody (analogově/digitální převodník, čítače/časovače, PWM (pulzně šířková modulace), vstupně/výstupní porty, sériové rozhraní, analogový komparátor...) na jednom čipu. Hlavní výkonnou jednotkou mikrokontroléru je jeho CPU (Central Processing Unit), která čte instrukce z paměti a na jejich základě vykonává program.

Výkon mikrokontroléru je ovlivněn:

- Architekturou (registry a šířky sběrnic pro data, adresy).
- Rozsahem příkazů.
- Maximální hodinovou frekvencí a počtem hodinových cyklů na příkaz.
- Efektivností programového kódu.

V závislosti na použité architektuře rozdělujeme mikrokontroléry na 4, 8, 16 nebo 32bitové. To, že je např. mikrokontrolér plně 8bitový, znamená, že je vybaven 8bitovou vnitřní architekturou, která mu dovoluje během jediné instrukce zpracovat jednu či dvě 8bitové hodnoty.

Díky malé velikosti a velké spolehlivosti se mikrokontroléry nejčastěji používají v jednoúčelových zařízeních pro regulaci či řízení nebo ve vestavěných systémech (tzv. embedded systémech). Mezi embedded zařízení lze zařadit např. bankomaty, pračky, herní konzole, routery…

3.2 Architektura mikroprocesorů

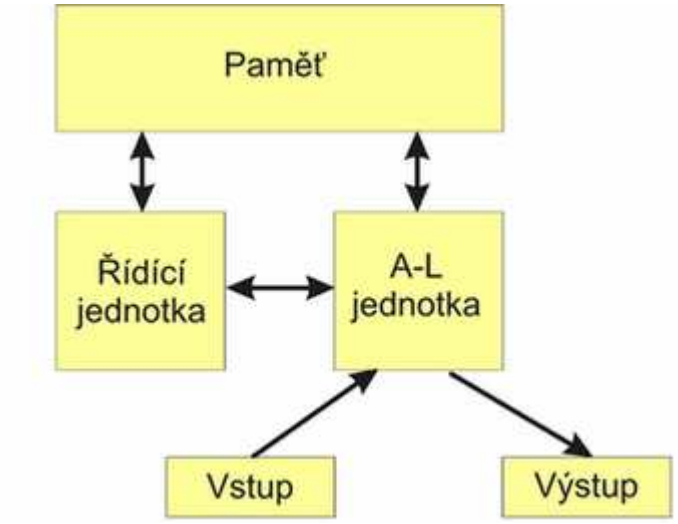
Architekturu můžeme chápat jako pohled na vnitřní uspořádání počítače/mikrokontroléru. Je složen z menších částí, které jsou vzájemně propojené a tvoří funkční celek. Lze popsat strukturu jednotlivé části, jak funguje a jakým způsobem komunikuje s ostatními částmi. Mezi základní koncepce vnitřního uspořádání patří von Neumannova a Harvardská architektura.

3.2.1 Von Neumannova architektura

Tato koncepce je pojmenována po americkém vědci a matematikovi maďarského původu Johnu von Neumannovi (28.12.1903-8.2.1957), který kromě práce na vývoji počítačů se ještě podílel na založení matematické teorie her, vývoji kvantové teorie a také se po určitou dobu podílel na vývoji první atomové bomby.

Základní myšlenkou je zásada, že vnitřní struktura počítače by se neměla nijak měnit v závislosti na zpracovávané úloze. Vnitřní struktura by tedy měla být maximálně univerzální, aby dokázala vyjít vstříc potřebám úloh nejrůznějšího typu. Veškeré přizpůsobení konkrétním úkolům má být řešeno výhradně programem.

Programy i jejich data by se měly uchovávat v téže operační paměti. Von Neumannova koncepce poprvé přišla s myšlenkou, že jak programy, tak i data jsou ve své podstatě jedno a totéž (posloupnosti nul a jedniček) a rozdíl mezi nimi je dán pouze významem, který jim přisuzuje ten, kdo s nimi pracuje. Počítač vycházející z von Neumannovy koncepce proto uchovává programy i data v jediné společné paměti. Záleží pak na právě probíhajícím programu, jak bude interpretovat to, co v operační paměti najde [16].



Obrázek 1: Von Neumannova architektura [http://cs.wikipedia.org]

Výhody:

- Není nutné rozlišovat instrukce pro přístup k paměti dat a paměti programu, z toho vyplývá zjednodušení čipu.
- Potřeba pouze jedné datové sběrnice (stejná pro data i program) – jednodušší výroba.

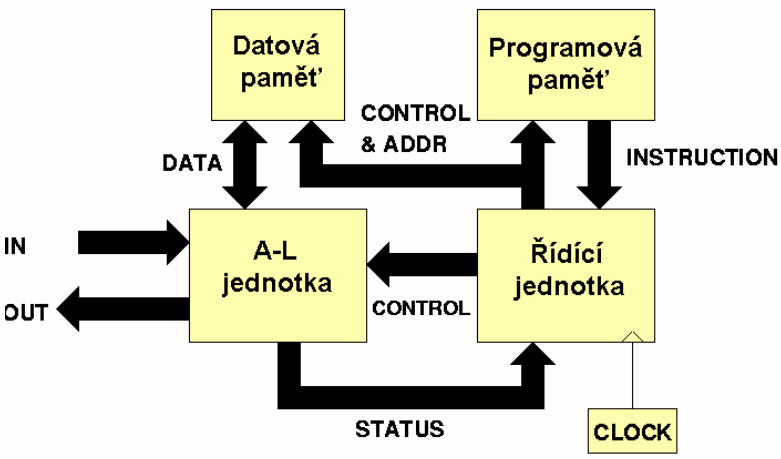
Nevýhody:

- Přenos obou typů dat po jedné sběrnici je pomalejší.
- Společné uložení dat a kódu může mít při chybě za následek přepsání vlastního kódu [5].

3.2.2 Harvardská architektura

Tento název je odvozen od sálového počítače Harvard Mark I z roku 1944, který byl vyvinut na harvardské univerzitě. Je považován za jeden z prvních univerzálních počítačů. V této architektuře je od sebe fyzicky oddělena paměť programu a dat. U Harvardské architektury není potřeba mít paměť stejných parametrů a vlastností pro data a pro program. Paměti můžou být naprosto odlišné, mohou mít různou délku slova, časování, technologii a způsob adresování. V některých systémech se pro paměť programu používá typ paměti ROM (read only memory) nebo Flash, přičemž paměť dat vyžaduje typ paměti RAM (random-access memmory). Díky oddělenému sběrnicovému

systému pro datovou a adresovou paměť je možné současně pracovat s daty (v datové paměti) i příkazy (v programové paměti), což je základní předpoklad pro vysokou rychlost zpracování [6].



Obrázek 2: Harvardská architektura [http://www.ee.nmt.edu]

Výhody:

- Program nemůže přepsat sám sebe.
- Technologie výroby datové a programové paměti se může lišit.
- Každá paměť může mít jinou velikost nejmenší adresovací jednotky.
- Dvě sběrnice umožňují jednoduchý paralelizmus, kdy lze přistupovat k datům a instrukcím současně.

Nevýhody:

- Dvě sběrnice kladou vyšší nároky na vývoj řídící jednotky procesoru a zvyšují i náklady na výrobu výsledného mikroprocesoru.
- Nevyužitou část paměti dat nelze použít pro program a obráceně [5].

3.3 Architektura souboru instrukcí

3.3.1 CISC

CISC (Complex Instruction Set Computer) procesor obsahuje kompletní instrukční sadu, která je v procesoru vytvořena na hardwarové úrovni. Je zde mnoho výkonných instrukcí, takže zadáním jediného příkazu je možné realizovat složité

zpracování dat, čímž se ušetří paměťové místo. Každá složitější instrukce je tvořena mikrokódem, který je rozdělen na několik elementárních instrukcí (mikroinstrukcí), které je procesor schopen vykonat. Většina složitějších instrukcí je tvořena různým počtem těchto mikroinstrukcí a z toho plyne různý počet hodinových cyklů potřebných k vykonání této instrukce. Programování procesorů CISC je díky velkému množství integrovaných instrukcí výrazně jednodušší než programování procesorů RISC [1, 8].

3.3.2 RISC

RISC (Reduced Instruction Set Computer) procesor má redukovanou instrukční sadu, která obsahuje jen nejpoužívanější instrukce. Všechny instrukce mají stejnou bitovou šířku, jsou méně složité a vyžadují méně složitý interpret příkazů. Tím je podstatně zefektivněno dekódování a zpracování instrukce, které zpravidla proběhne v jednom hodinovém cyklu. Rychlost zpracování je vyšší než u procesorů CISC, ale je zde větší délka programového kódu [1, 7].

3.4 Mikrokontroléry Atmel

Tato kapitola se podrobněji věnuje 8bitovému mikrokontroléru od firmy Atmel založeném na jádře AVR.

3.4.1 Firma Atmel

Atmel Corporation s hlavním sídlem v San Jose v Kalifornii byla založena v roce 1984. Je výrobcem a vývojářem široké řady mikrokontrolérů (8051 a jejich klony, mikrokontroléry s architekturou AVR a AVR32, AT91CAP - jejich vnitřní logika lze upravit podle požadavků aplikace, AT91SAM s 32-bitovou architekturou ARM, MARC4 s 4-bitovou architekturou…). K dalším výrobkům patří čipy pro RF identifikaci, GPS moduly, dotykové technologie, permanentní paměti (Flash, EPROM …). Atmel Corporation je držitelem certifikátů ISO 9001, ISO/TS 16949 a ISO 14001 [13].

3.4.2 Historie architektury AVR

Architekturu AVR poprvé popsali v roce 1992 dva studenti norského technického institutu (Norwegian Institute of Technology) Alf-Egil Bogen a Vergard

Wollan ve své práci High Speed Analog to Digital Converters in CMOS (Rychlý CMOS analogově digitální převodník). Svoji studii prodali firmě Atmel, která založila v Norsku dceřinou společnost, kde zaměstnala tyto dva studenty, aby technologii AVR dále rozvíjeli. 8bitové mikrokontroléry AVR se liší velikostí pamětí, množstvím periférií, počtem instrukcí, maximálním výkonem a pouzdrem. Vyráběné 8bitové mikrokontroléry s architekturou AVR lze rozdělit do devíti skupin podle použití (každá skupina obsahuje od jednotek po desítky konkrétních typů mikrokontrolérů).

Skupiny 8bitových mikrokontrolérů AVR:

- Automotive AVR.
- Battery Management AVR.
- CAN AVR.
- LCD AVR.
- Lighting AVR.
- megaAVR.
- tinyAVR.
- USB AVR.
- XMEGA.

3.5 Mikrokontrolér Atmel s architekturou AVR – ATmega16

Mikrokontrolér ATmega16 je nízkopříkonový 8bitový mikrokontrolér ze skupiny megaAVR založený na rozšířené architektuře AVR RISC a vyráběný v pouzdru DIP 40 nebo TQFP 44.

Obrázek 3: Pouzdro DIP40 (vlevo) a TQFP44 (vpravo) [http://cz.farnell.com]

Parametry mikrokontroléru ATmega16:

- Instrukční soubor obsahuje 131 instrukcí.
- 32 registrů délky 8 bitů.
- Čtyři 8bitové vstupně/výstupní porty (celkem 32 vstupů/výstupů).
- Hodinový kmitočet 0 až 16 MHz, maximální výpočetní výkon až 16 MIPS.
- Paměť programu je tvořena zabudovanou Flash, kapacita je 16 KB – počet přeprogramování je 1000 cyklů.
- Datová paměť RAM kapacity 1 KB.
- Datová paměť EEPROM kapacity 512 B – počet přeprogramování je 100 000 cyklů.
- Flash a EEPROM jsou programovatelné přímo v systému pomocí rozhraní SPI nebo JTAG.
- Dva 8bitové čítače/časovače, jeden 16-bitový čítač/časovač.
- Čtyři PWM kanály.
- Analogový komparátor a 10bitový A/D převodník.
- Jednotky USART, SPI, TWI.
- Jednotky WDT, Power-on reset.
- Zabudovaný RC oscilátor.

3.5.1 Kombinace RISC a CISC

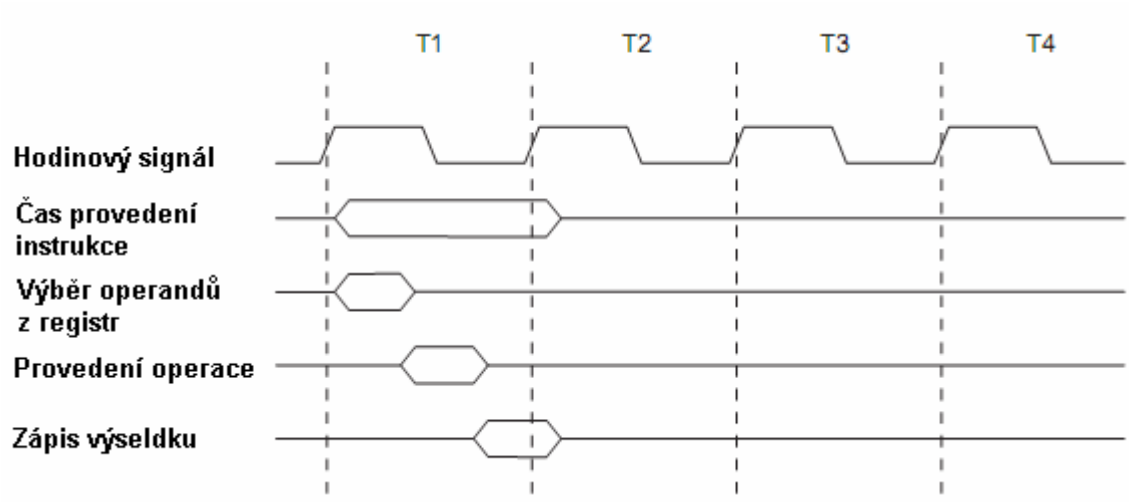
S počtem 133 instrukcí se CPU mikrokontroléru ATmega16 přibližuje architektuře CISC. Všechny ostatní charakteristiky jako stejná bitová šířka instrukcí a zpracování instrukcí v jednom hodinovém cyklu jsou naopak typické vlastnosti architektury RISC. Proto lze říci, že architektura AVR využívá výhod obou architektur, tedy výkonnost architektury RISC a rozsahem instrukční sady blížící se CISC. Tímto je možné udržet menší velikost programu při vyšších rychlostech zpracování.

3.5.2 Zpracování instrukcí

Jednotná délka většiny instrukcí je 16 bitů. V těchto 16 bitech je uložen operační znak (určuje typ instrukce), ale i informace o operandech. Tento prvek zjednodušuje realizaci dekodéru instrukcí, což přináší zvýšení rychlosti.

AVR architektura vychází z koncepce rychle přístupného registrového pole (register file), které obsahuje 32 obecně použitelných registrů délky 8 bitů. Přístup do registrového pole je prováděn v jediném strojovém cyklu. To znamená, že během jednoho strojového cyklu lze vykonat jednu aritmeticko-logickou operaci.

Oba operandy aritmeticko-logické instrukce jsou načteny z registrového pole s náběžnou hranou hodinového signálu, operace je provedena při sestupné hraně a výsledek uložen do cílového registru před náběžnou hranou následujícího hodinového impulzu. Strojový cyklus mikrokontrolérů AVR (ATmega16) přímo odpovídá hodinovému cyklu, což umožňuje velký výpočetní výkon (při hodinovém kmitočtu 16 MHz je výkon 16 MIPS, tedy 16 milionů instrukcí za sekundu) [3].

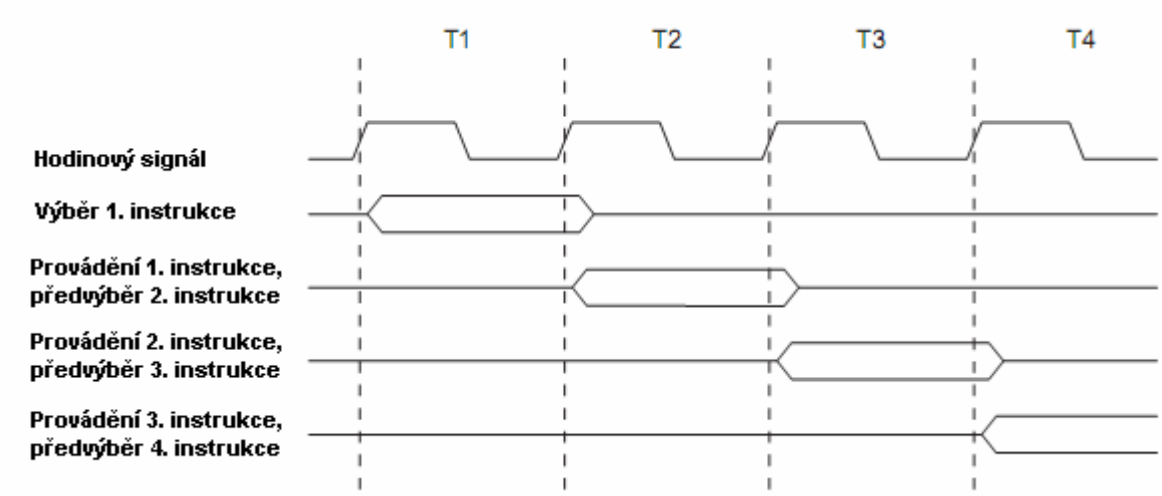


Obrázek 4: Jednocyklová aritmeticko-logická operace [4]

3.5.3 Předvýběr instrukcí

Mikrokontroléry AVR (ATmega16) využívají jednoduchý, ale účinný předvýběr instrukcí. V mikrokontroléru je implementován jednofázový pipelining (zřetězené provádění instrukcí). Z programové paměti se nejdříve načte první instrukce. V následujícím hodinovém cyklu se tato instrukce provádí a současně s tím probíhá

předvýběr druhé instrukce. S dalším hodinovým cyklem se provádí druhá instrukce a z instrukční fronty se načítá třetí instrukce. Tímto způsobem probíhá načítání dalších instrukcí [3].



Obrázek 5: Předvýběr instrukce a zřetězení [4]

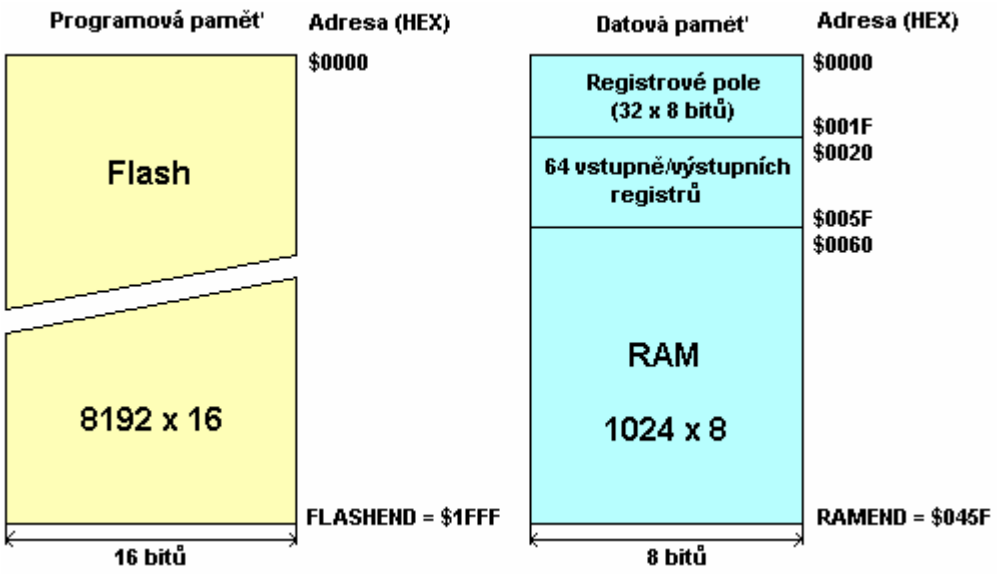
3.5.4 Paměťový prostor

Mikrokontroléry řady AVR používají harvardskou architekturu, tj. paměť programu a paměť dat jsou odděleny. Vnitřní paměť mikrokontroléru tvoří Flash, RAM a EEPROM.

Vnitřní programová paměť

Vnitřní programová paměť pro ATmega16 je realizována jako 16 KB paměť typu Flash. Paměť je adresována po slovech (16 bitů). Nejvyšší adresa je 8191 (FLASHEND = $1FFF). Programovou paměť je možno rozdělit na dvě sekce: aplikační sekce (Application Flash Section) a bootovací sekce (Boot Flash Section).

Bootovací sekce a tzv. bootloader (speciální zaváděcí program) poskytuje mechanizmus pro zavedení nebo zpětné čtení programového kódu samotným procesorem – tímto způsobem lze naprogramovat aplikace, které se sami aktualizují pomocí boot loaderu [4].



Obrázek 6: Struktura programové a datové paměti u ATmega16 [3]

Vnitřní datová paměť

Vnitřní datová paměť je tvořena statickými buňkami (nepotřebují obnovovat jako dynamická paměť a jsou rychlejší) paměti typu SRAM (Static RAM). Datová paměť začíná blokem 32 bajtů, které odpovídají registrům registrového pole (R0 až R31). Následuje 64 vstupně/výstupních registrů (přes tyto registry se ovládají periferie). Poslední blok datové paměti o velikosti 1 KB je libovolně použitelná paměť [3].

Vnitřní paměť EEPROM

Mikrokontrolér ATmega16 obsahuje paměť EEPROM, do které lze zapisovat po jednom bajtu (při zápisu nových údajů není nutné obsah paměti vymazat – mazací operace při zápisu jednoho bajtu je řešena vnitřně). EEPROM se používá v případech, kdy je potřeba uchovat údaje (např. uživatelská nastavení) i po vypnutí napájení.

3.5.5 Registry

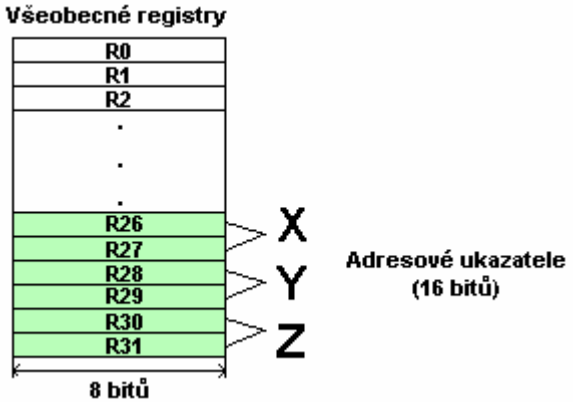
Pod pojmem registr si lze představit malou část paměti RAM obsažené v čipu. Do těchto registrů je možné zapisovat nebo z nich číst data. Dále lze hodnoty v jednotlivých registrech sčítat, odčítat a provádět logické operace (AND, OR, XOR…) s nějakou konstantou. Slouží jako dočasná buňka pro uložení operandu, který se účastní instrukce nebo k uchování výsledků instrukcí. U mikrokontrolérů Atmel s jádrem AVR lze registry rozdělit do několika skupin podle způsobu použití [9].

Registrové pole (Register File)

Je skupina 32 registrů, kde každý registr má délku 8 bitů. Většinu těchto registrů můžeme použít jako vstupní nebo výstupní operand instrukcí. Registrové pole lze tedy chápat jako 32 všeobecně použitelných registrů, které jsou umístěny v prvních 32 bitech datové paměti. Protože se registrové pole nachází v datové paměti, lze ke každému z registrů přistupovat buď jeho jménem (R0 až R31) nebo jeho adresou ($0000 až $001F).

Ukazatele (pointery)

Posledních 6 registrů registrového pole můžeme použít jako 16-bitové ukazatele pro nepřímé adresování paměti. Ukazatel je složen z dvojice registrů pojmenovaných X (registry R26 a R27), Y (R28 a R29) a Z (R30 a R31). Pomocí nepřímého adresování můžeme adresovat prostor 64 KB (2^{16}) paměťových buněk. Nepřímé adresování je výhodné při blokovém zpracování dat. ATmega16 disponuje instrukcemi **post-inkrement** (automatické zvýšení ukazatele o jedničku po provedení instrukce-ukazatel se posune na následující buňku v paměti) a **pre-dekrement** (snížení ukazatele o jedničku před přístupem do paměti – ukazatel se posune na předcházející paměťovou buňku). Výhodou těchto instrukcí je, že změna ukazatele proběhne v rámci instrukce nepřímého adresování a není třeba další instrukce na změnu pozice ukazatel [3].



Obrázek 7: Ukazatele X, Y a Z [3]

Ukazatel vrcholu zásobníku (Stack Pointer - SP)

Registr SP je ukazatelem na vrchol zásobníku, který je typu LIFO (Last In First Out). Ukazatel vrcholu zásobníku se obvykle nastavuje na konec datové paměti na RAMEND.

Programový čítač (Program Counter - PC)

Tento registr obsahuje adresu právě prováděné instrukce v programové paměti. Obsah registru se automaticky mění při provádění instrukcí. Skoky v programové paměti jsou zajištěny tak, že se do programového čítače uloží cílová adresa skoku.

Stavový registr (Status Register - SREG)

Je 8bitový vstupně/výstupní registr, který obsahuje příznaky definující stav mikrokontroléru po vykonání instrukce. V příznaku je uložena informace o úspěšnosti provedení instrukce [12].

Význam jednotlivých bitů:

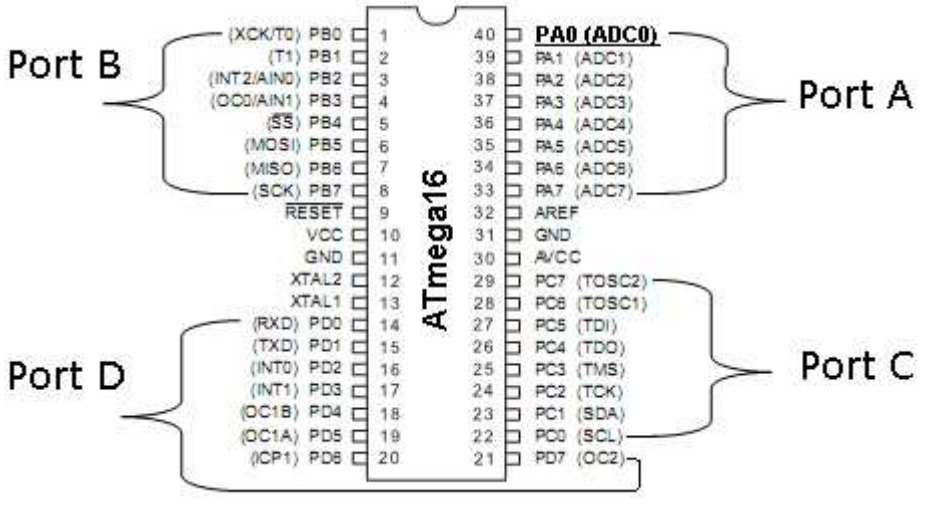
- **C** – příznak přetečení (**Carry Flag**). Indikuje přetečení po aritmetické nebo logické operaci.
- **Z** – příznak nulového výsledku (**Zero Flag**). Indikuje nulový výsledek aritmetické nebo logické operace.
- **N** – příznak negativního výsledku (**Negative Flag**). Indikuje záporný výsledek po aritmetické nebo logické operaci.
- **V** – příznak přetečení čísla v druhém doplňku (**Two's Complement Overflow Flag**). Podporuje aritmetiku v druhém doplňku.
- **S** – znaménkový bit (**Sign Bit**). Indikuje znaménko čísla v druhém doplňku (S = N xor V).
- **H** – pomocný příznak přetečení (**Half Carry Flag**). Indikuje přenos mezi dolní a horní polovinou výsledku.
- **T** – kopírovací bit (**Bit Copy Storage**). Slouží jako zdrojový nebo cílový bit instrukcí BLD (Bit LoaD) – načtení bitu a BST (Bit STore) – uložení bitu.
- **I** – globální povolení přerušení (**Global Interrupt Enable**). Pro příjem přerušení musí být tento bit nastaven (I = 1). Po vstupu do obsluhy přerušení je tento bit vynulován a tím je příjem dalších přerušení zablokován [3].

Vstupně/výstupní registry (I/O Memory)

Slouží k řízení periferií a jádra mikrokontroléru. Jsou implementovány jako buňky datové paměti a práce s těmito registry je naprosto totožná jako s všeobecnými. Do těchto registrů patří i stavový registr SREG a ukazatel zásobníku SP.

3.6 Periferie ATmega16

Mikrokontrolér ATmega16 je vybaven řadou periferií a čtyřmi obousměrnými 8bitovými porty (PA až PD). Vstupy/výstupy portů jsou vyvedeny na jednotlivé piny (kontakty) pouzdra mikrokontroléru. Kvůli omezenému počtu vývodů mikrokontroléru, jsou všechny bity daných portů sdíleny se zabudovanými periferiemi. Např. rozhodneme-li se použít vývod 40 (pouzdro DIP 40) jako výstupní/vstupní port (PA0), nemůžeme již současně použít nultý vstupní kanál A/D převodníku (ADC0), který sdílí port PA0. Použití tohoto kanálu A/D převodníku je možné až po přenastavení příslušného bitu, aby vývod sloužil jako vstup A/D převodníku.



Obrázek 8: Porty mikrokontroléru ATmega16 [12]

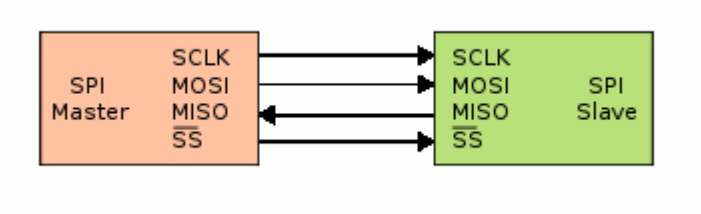
3.6.1 Sériová rozhraní

Mikrokontrolér ATmega16 obsahuje tři různá rozhraní pro sériovou komunikaci.

SPI

SPI (Serial Peripheral Interface) slouží pro přenos dat mezi mikrokontrolérem a periferními zařízeními nebo jinými mikrokontroléry podporující přenos po sběrnici SPI.

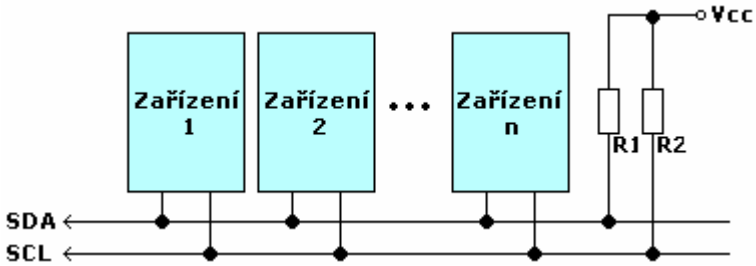
Zařízení spolu komunikují v módu master (řídící obvod) / slave (řízený obvod), kde master začíná komunikaci. Pro přenos se využívají 4 vodiče, SCLK (Serial Clock - hodinový signál), MOSI (Master Out Slave In – výstup z master obvodu), MISO (Master In Slave Out – výstup ze slave obvodu) a SS (Slave Select – výběr slave obvodu) [3, 10].



Obrázek 9: Schéma zapojení SPI [9]

TWI

TWI (Two-Wire Interface) se používá pro komunikaci mezi mikrokontrolérem a až 128 různými zařízeními za použití dvou linek, sběrnice SCL (Serial Clock - hodinový signál) a SDA (Serial Data – Sériová Data). Všechna zařízení připojená na sběrnici mají individuální adresy. Na obě linky je nutné připojit tzv. pull-up rezistor připojený na Vcc (napájecí napětí), které v případě nečinnosti na sběrnici „zdvihnou“ napětí na obou linkách na hodnotu Vcc [4, 11].



Obrázek 10: TWI komunikace [archiv autora]

USART

USART (Universal Synchronous Asynchronous Receiver Transmitter) je jednotka schopná obousměrné komunikace, a to jak synchronní, tak i asynchronní s možností volby přenosové rychlosti. Umožňuje přenos 5-9 datových bitů s 1-2 stopbity. Validitu přenesených dat zajišťuje hardwarová kontrola paritního bitu (sudá/lichá parita).

3.6.2 A/D převodník

Slouží k převodu napětí nebo jiných fyzikálních veličin (po převodu na napětí) na digitální informaci. ATmega16 má 10bitový A/D převodník, který umožňuje měřit rychlostí až patnáct tisíc vzorků za sekundu.

3.6.3 Analogový komparátor

Umožňuje porovnávat hodnoty dvou vstupních napětí. Lze použít pro realizaci regulačních úloh nebo při měření fyzikálních veličin. Porovnávaná napětí se přivádí na vývody neinvertujícího vstupu (AIN0 - vývod u ATmega16) a invertujícího vstupu (AIN1 - vývod u ATmega16). Výstup komparátoru pak nabývá hodnoty log.1, když je napětí na AIN0 větší než na AIN1 a log.0 při opačné situaci.

3.6.4 Čítač/časovač

Podle [3] je čítač obvod, který počítá impulzy vnějšího signálu (měří jeho kmitočet) a časovač čítá pevný kmitočet, který je odvozen od hodinového signálu mikrokontroléru (napočítáním určitého počtu impulzů se zajistí odměření časového úseku).

Mikrokontrolér ATmega16 je vybaven třemi čítači/časovači (čítač/časovač 0 až 3).

4 Programové vybavení

4.1 Vývojové prostředí

Integrated development environment (IDE) je soubor softwarových nástrojů, které usnadňují programování mikrokontrolérů. Výhoda těchto prostředí je, že obsahují vše potřebné pro vývoj celého projektu od napsání zdrojového kódu, přes jeho kompilaci a ladění, až po nahrání výsledného kódu do mikrokontroléru. Vytvoření programu ve vývojovém prostředí se podle [14] skládá z několika kroků, které se mohou opakovat:

- Napsání zdrojového kódu v jazyce, který dané vývojové prostředí podporuje.
- Simulace programu a nahrání kódu do mikrokontroléru (přes programátor).
- Ladění programu (debug) přímo v konkrétní aplikaci.
- Upravování zdrojového kódu a odstraňování chyb.

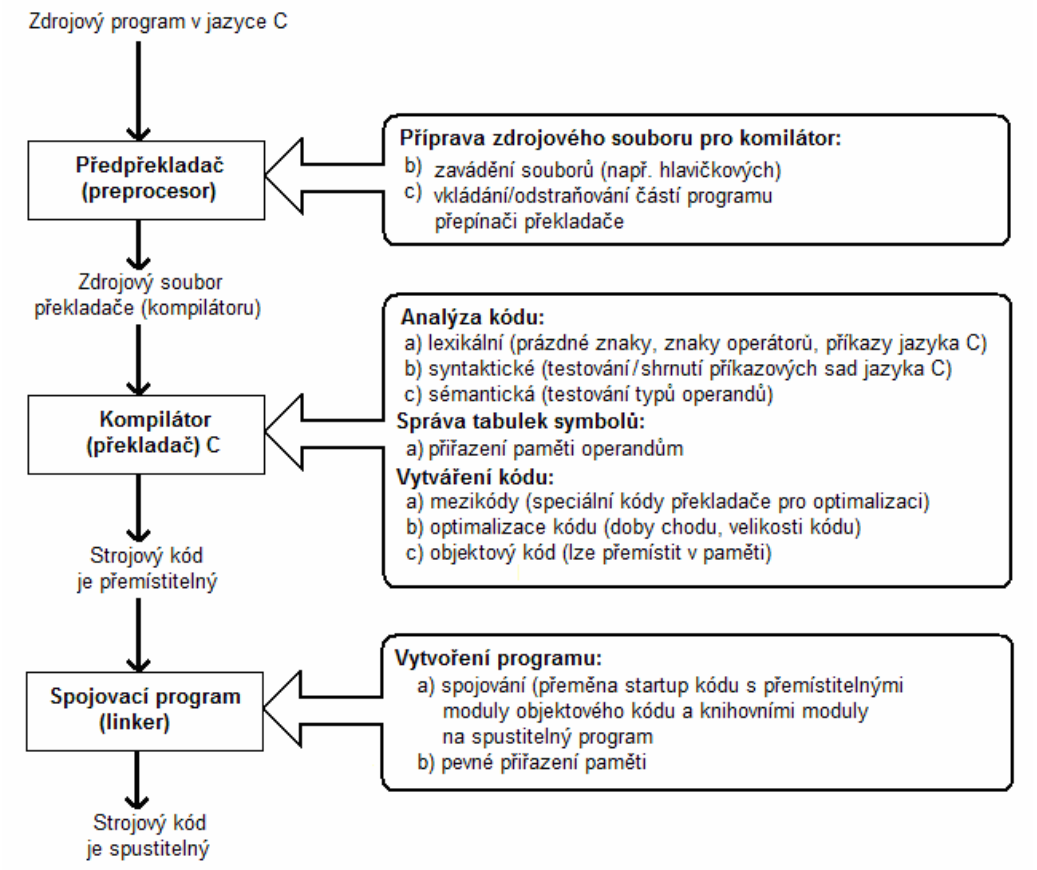
4.2 Kompilátor C pro mikrokontroléry

V mnoha oblastech a aplikacích mikrokontrolérů je již delší dobu běžné použití jazyka C. Nabídek vyšších programovacích jazyků je dostatek, od kompilátoru C až po interpret Basicu. Výrobci mikrokontrolérů v tomto směru nezůstávají pozadu a vytvářejí architektury mikrokontrolérů s podporou programovacích jazyků vyšší úrovně HLL (High Level Language). Jedná se o oboustranně přizpůsobené řešení architektury mikrokontroléru a kompilátoru C. Tím se nyní změnila situace zejména pro malé (8bitové) mikrokontroléry, kde se jazyk C používá ve stále větší míře. Kvůli omezeným systémovým zdrojům malých MCU jsou kladeny vyšší nároky na kompilátor jazyka C, který musí podporovat:

- Řízení rozmisťování (location) dat a kódu.
- Speciální datové typy (např. bitová proměnná, registrová proměnná).
- Zvláštní metody adresování (short addressing).
- Modely paměti specifické pro aplikaci.
- Systémové řízení na úrovni přerušení.

Ideální kompilátor C by vytvořil strojový kód tak dobře jako optimálně napsaný program v assembleru. Díky stále se zlepšujícím optimalizačním metodám se tomuto teoretickému ideálnímu případu dnešní kompilátory jazyka C stále více přibližují [1].

Průběh vytváření kódu kompilátorem je složitý proces. V první fázi se připravuje zdrojový text předpřekladačem (preprocesorem) pro vlastní běh kompilátoru. V závislosti na direktivách předpřekladače se zavádějí nebo odstraňují části kódu, zavádí se obsah souborů include a dosazují se hodnoty symbolických konstant a maker. Teprve potom začíná práce kompilátoru, který vytváří přirozený zdrojový kód v závislosti na architektuře a instrukční sadě mikrokontroléru [1].



Obrázek 11: Průběh kompilace programu [1]

Funkce, která si u kompilátoru zasluhuje mimořádnou pozornost, je optimalizace kódu. Optimalizace začíná již u programátora, který musí vytvořit základnu pro optimalizaci kompilátoru použitím účinného algoritmu vhodně zvolených typů proměnných a paměťových tříd. Své optimalizační moduly si společnosti velmi dobře

hlídají a dále je rozvíjí a zdokonalují. Existuje řada standardních metod, které vždy najdeme v optimalizačních modulech a je možné je rozdělit na dvě skupiny podle závislosti optimalizace na hardwaru. Vybrané druhy optimalizace v tab. 1 a 2.

Tabulka 1: Optimalizace závislá na hardware [1]

Způsob optimalizace	Popis
Register Variable (registrové proměnné)	Automatické proměnné a parametry funkcí se pokud možno, co nejvíce umísťují do registrů. Tím je přístup na ně efektivnější a nezabírají žádné paměťové místo v RAM.
Simple Access Optimalization (optimalizace jednoduchým přístupem)	Přístupy na interní datové a bitové adresy jsou optimalizovány strojovými instrukcemi závislými na MCU, využívají se speciální posloupnosti strojových příkazů.
Code Rearrangenment (reorganizace kódu)	Je-li smyčka FOR efektivnější než programátorem použitá smyčka WHILE, kompilátor kód změní.

Tabulka 2: Optimalizace nezávislá na hardware [1]

Způsob optimalizace	Popis
Constant folding (zpracování konstant)	Výpočty, které obsahují konstanty, jsou v co největší míře prováděny již kompilátorem.
Local Common/Global Common Subexpression Elimination (vyloučení opakujících se výpočtů nebo částí výrazů)	Vícenásobné (opakované) výpočty nebo stejné části výrazů uvnitř výrazů nebo funkce jsou pokud možno co nejvíce eliminovány a počítají se jen jednou, přičemž výsledek prvního výpočtu se ukládá do registru. To vyžaduje velmi přesnou analýzu prgramu prováděnou kompilátorem.
Dead Code Elimination (vyloučení "mrtvého" kódu)	Odstranění nepoužívaného pasivního zdrojového kódu z programu.
Cross-jumping and hoisting (náhrada opakujících se úseků programu skoky)	Indentické úseky kódu na různých místech v programu se vytváří jen jednou a zpracovávají pomocí skokových příkazů.
Jump Negation (negace skoků)	Testy pro podmíněné skoky se invertují, lze-li tím odstranit jiné skoky nebo nepoužívaný kód.
Loop Rotation (rotace smyček)	V programových smyčkách se zamění uspořádání kódu, dosáhne-li se tím rychlejšího a efektivnějšího kódu.
Control Flow Optimalization (optimalizace řídícího toku)	Výrazy typu switch - case se optimalizují a zjednodušují jako skokové tabulky nebo skokové řetězce. Proměnné z různých oblastí pamětí se zavádějí přímo do operace.

U většiny kompilátorů je možnost optimalizaci řídit nebo volit stupeň optimalizace. Programátor se musí rozhodnout, zda bude optimalizovat z hlediska

velikosti kódu (Size) nebo doby běhu (Speed). Při vysokém stupni optimalizace nemusí být již laditelný na úrovni jazyka C.

4.3 Jazyk C

Jazyk C je univerzální a flexibilní programovací jazyk, který je nezávislý na konkrétní hardwarové a softwarové platformě. Mezi jeho výhody patří podle [15] zejména:

- 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).

Vývoj jazyka C

Vývoj jazyka C začal v Bellových laboratořích AT&T mezi roky 1967 a 1973. V roce 1978 byla vydána kniha *The C programming Language* jejímiž autory jsou Brian W. Kernighan a Denis M. Ritchie. Tato kniha se stala základní učebnicí jazyka C a také prvním neoficiálním standardem jazyka, který se běžně označuje jako „K&R“ podle počátečních písmen příjmení autorů.

Dnešním oficiálním standardem, který z K&R vychází, je tzv. ANSI C z roku 1990. Součástí standardu je i přesná specifikace množiny knihovních funkcí a hlavičkových souborů, které musí každá implementace ANSI C kompilátoru obsahovat.

Jazyk C neumožňuje přímo práci s řetězci a poli ani neobsahuje nástroje pro vstupy a výstupy. Tyto akce se provádí prostřednictvím volání funkcí, z čehož plyne jednoduchost jazyka a jeho nezávislost na hardware [16].

Inicializace mikrokontroléru

Pro programování mikrokontrolérů je jazyk C velmi výhodný z hlediska jeho blízkosti ke strojovému kódu, takže nárůst kódu (code-overhead) vůči assembleru je malý, přesto však má stupeň abstraktnosti potřebný pro vyšší programovací jazyk.

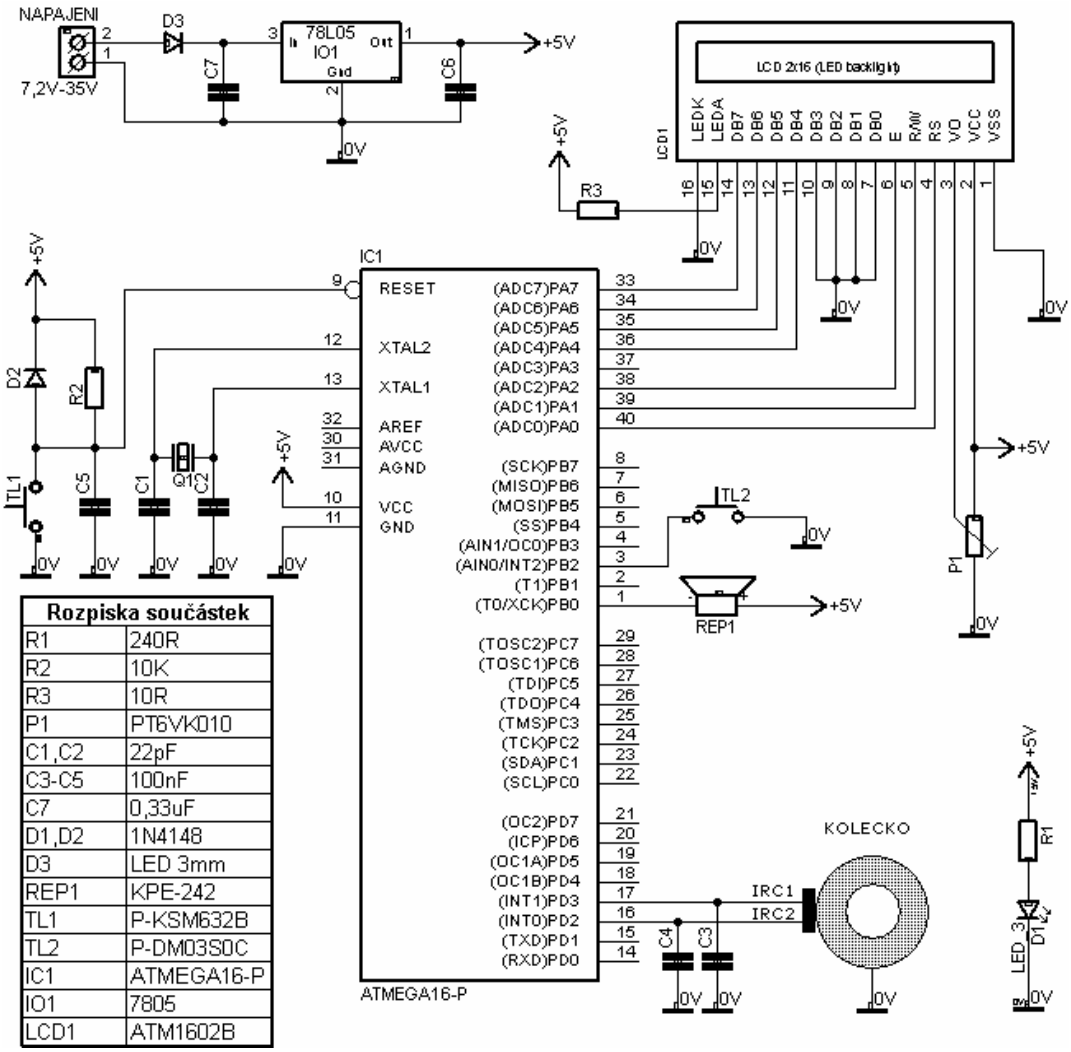
U mikrokontrolérů se před vyvoláním programu v C spouští počáteční (startup) modul, který je většinou napsaný v assembleru. Startup modul se stará o počáteční inicializaci různých registrů mikrokontroléru, oblastí paměti, ukazatele zásobníku a vektorů přerušení. Po dokončení inicializace následuje běh programu C vždy funkcí s názvem main(), která je vyvolána startup modulem. Funkce main() převezme řízení a zpracovává postupně příkazy zdrojového kódu C a volá další funkce. Po resetu mikrokontroléru se pokaždé spouští jako první startup kód [1].

5 Realizace kuchyňské minutky

Pro demonstraci programování mikrokontroléru ATmega16 v jazyce C byla zvolena konstrukce jednoduché kuchyňské minutky. Na minutce lze nastavit čas od 1 do 60 min, po uplynutí nastaveného času se ozve zvukový signál. Odpočet lze kdykoli zastavit a tím se nastavený čas vynuluje.

5.1 Elektronické řešení

Pro vývoj obvodu minutky bylo zvoleno nepájivé kontaktní pole, které umožňuje snadné a rychlé propojování elektronických součástek jen pomocí drátků. Další výhodou je také snadná změna zapojení realizovaná jen přepojením součástek. Na obr. 12 je schéma zapojení elektronické minutky.



Rozpiska součástek	
R1	240R
R2	10K
R3	10R
P1	PT6VK010
C1,C2	22pF
C3-C5	100nF
C7	0,33uF
D1,D2	1N4148
D3	LED 3mm
REP1	KPE-242
TL1	P-KSM632B
TL2	P-DM03S0C
IC1	ATMEGA16-P
IO1	7805
LCD1	ATM1602B

Obrázek 12: Elektronické schéma kuchyňské minutky [archiv autora]

5.1.1 Napájecí zdroj

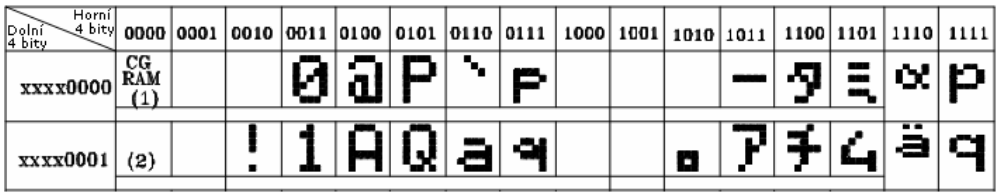
Napájení celého zařízení zajišťuje zdroj se stabilizátorem LM7805 v katalogovém zapojení, který stabilizuje stejnosměrné napětí na vstupu (v rozsahu 7,2 – 35V) na 5V na výstupu. Dioda D3 zabraňuje případnému přepólování napájecího napětí.

5.1.2 Displej

Pro zobrazení časových údajů je použit LCD displej ATM1602B. Displej pracuje v textovém režimu (2 řádky, 16 znaků) a je řízen řadičem HD44780 od firmy Hitachi. Komunikace mikrokontroléru a řadiče LCD může probíhat po čtyřech nebo osmi datových linkách. V případě komunikace po čtyřech linkách probíhá zápis nadvakrát (nejdříve horní a pak dolní polovina bajtu) po vodičích DB7 až DB4. O komunikaci se starají linky RS (výběr přenosu dat nebo příkazu), R/$\overline{W}$ (čtení nebo zápis) a E (povolovací vstup) [3]. U minutky byl zvolen 4-bitový režim, aby bylo možné použít k ovládání displeje pouze jeden 8bitový port ATmegy16 (port A). Mikrokontrolér tedy obsluhuje čtyři datové a tři řídící linky LCD.

Displej disponuje dvěma typy pamětí:

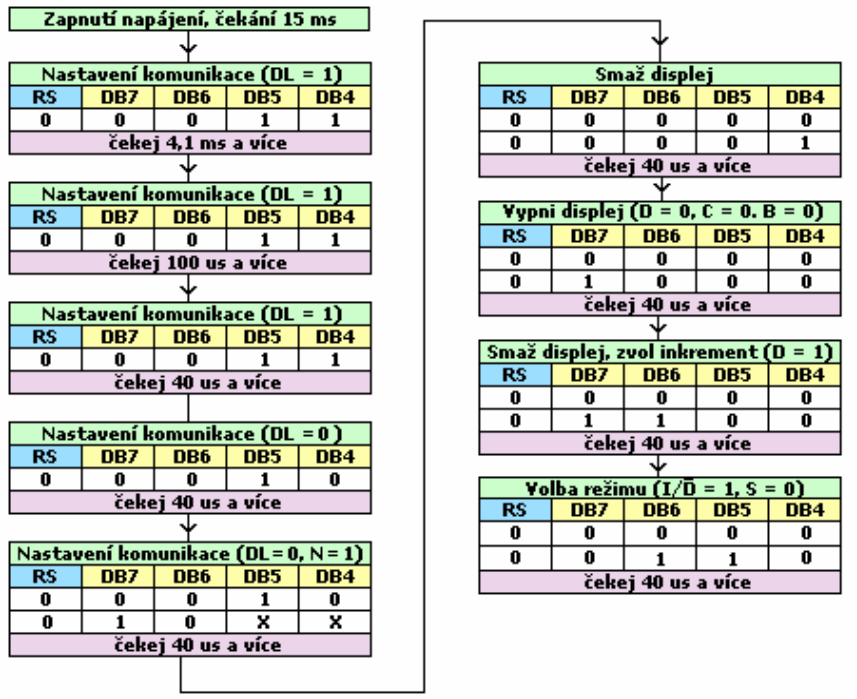
- DDRAM obsahuje výrobcem definovanou sadu znaků, které se dají zobrazit na displeji.
- CGRAM umožňuje uživateli vytvořit až osm vlastních znaků. Tato možnost se obvykle využívá pro definici české diakritiky.



Obrázek 13: Část paměti DDRAM a CGRAM [http://www.mathar.com]

Inicializační sekvence

Pro komunikaci s displejem je nutné provést počáteční inicializační sekvenci. Počáteční nastavení pro 4-bitovou komunikaci je na obr. 14. a význam nastavovaných bitů je uveden v tabulce 3.



Obrázek 14: Inicializační sekvence pro 4bitovou komunikaci [4]

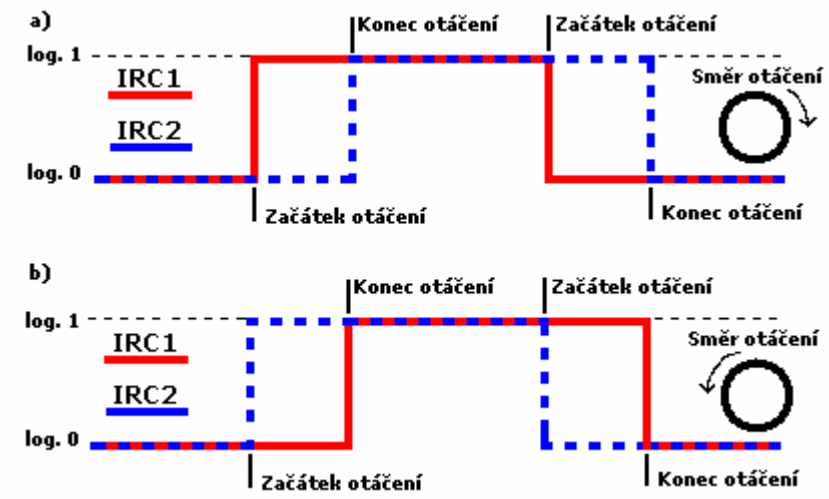
Tabulka 3: Význam jednotlivých bitů displeje [4]

X	Libovolná hodnota (0 nebo 1).
$I/\overline{D}$	Inkrementace (1), dekrementace (0).
S	Režim displeje (0 = normální práce, 1 = kombinovaný posun displej).
D	Displej zapnut (1), vypnut (0).
C	Zobrazení kurzoru zapnuto (1), vypnuto (0).
B	Blikání kurzoru zapnuto (1), vypnuto (0).
$S/\overline{C}$	Posun displeje jsou-li data zapsána (1), posun kurzoru jsou-li data zapsána (0).
$R/\overline{L}$	Posun doprava (1), doleva (0).
DL	8 bitová komunikace (1), 4 bitová komunikace (0).
N	Počet řádků: dva (1), jeden (0).

5.1.3 Ovládání minutky

Pro ovládání minutky bylo vybráno „scrollovací“ kolečko z počítačové myši. Otáčením na jednu stranu se minuty zvyšují, na druhou stranu se snižují. Spuštění a zastavení odpočtu je zajištěno integrovaným mikrospínačem v kolečku (TL2 ve

schématu). Kolečko má tři vývody – první je připojen na 0V a zbylé dva (IRC1 a IRC2) jsou buď spojeny s prvním, nebo ne, v závislosti na poloze kolečka. Vývody IRC1 a IRC2 jsou připojeny k portu D ATmegy16 na vstupy INT0 a INT1. Tyto vstupy jsou nastaveny, aby reagovaly na změnu stavu (z log 0 na log 1 a naopak), kdy každá tato změna vyvolá přerušení. Obslužná rutina tohoto přerušení zajistí buď inkrementaci proměnné pro uložení času nebo její dekrementaci s ohledem na směr otáčení kolečka. Kolečko pracuje na podobném principu jako vypínač (vypnuto/zapnuto), na obr. 15 lze vidět jednotlivé průběhy napětí na vývodech IRC1 a 2.



Obrázek 15: Průběhy napětí při otáčení kolečkem [archiv autora]

Například uvažujeme-li směr otáčení doprava podle obr. 15 a). Vývody kolečka IRC1 a IRC2 jsou v log. 0, při otáčení přechází vývod IRC1 jako první do stavu log. 1, to vyvolá v mikrokontroléru přerušení od INT1 a následně začne probíhat rutina na jeho ošetření. V této rutině také dojde k vynulování požadavku na přerušení od INT0 (vyvolané přechodem IRC2 do log. 1), který ale nastal až po příchodu přerušení INT1, proto je nutné tento příznak vynulovat, aby nebylo generováno další přerušení.

5.1.4 Zvolený mikroprocesor

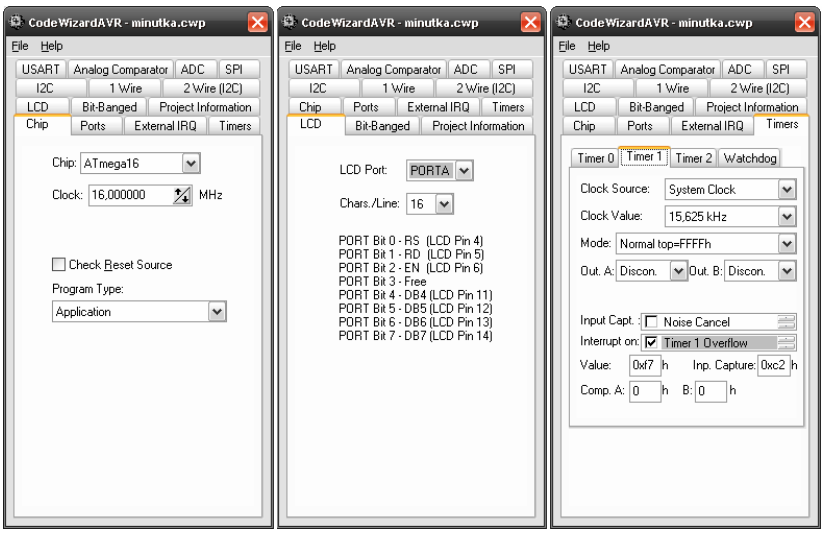
Pro řízení celého obvodu byl vybrán mikrokontrolér ATmega16 s krystalovým oscilátorem o hodnotě 16 MHz. Reset mikrokontroléru po připojení napájecího napětí je zajištěn obvodem složeným z rezistoru R2, kondenzátoru C5 a diody D2. Tlačítko TLl slouží k manuálnímu resetu.

5.1.5 Zvukový výstup

Siréna (REP1) slouží jako zvukový výstup. Je připojena mezi napájecí napětí a port PB0, který je nastaven jako výstup s počáteční hodnotou log.1. Oba vývody sirény jsou v klidovém stavu na stejném potenciálu. Po uplynutí nastaveného času se stav výstupu PB0 změní na log.0 a dojde ke spuštění zvukového signálu.

5.1.6 Použitý software

Pro napsání zdrojového kódu bylo vybráno vývojové prostředí CodeVisionAVR od HP InfoTech S.R.L., které je pro nekomerční účely volně ke stažení. Jediné omezení je maximální délka kódu 2 kB. Toto omezení je však plně dostačující pro menší projekty. Velkou výhodou tohoto prostředí je průvodce pro vytvoření projektu (CodeWizard AVR), pomocí kterého je možné počáteční inicializaci mikrokontroléru nastavit jen „zaškrtnutím“ požadovaných vlastností v dialogovém okně a uvedený průvodce vygeneruje zdrojový kód s inicializací. Prostředí poskytuje řadu knihoven, např. pro práci s řádkovými LCD, což velmi usnadňuje práci.



Obrázek 16: Ukázka nastavení v CodeWizard AVR [archiv autora]

Ladění programu probíhalo ve vývojovém prostředí AVR Studio 4 od firmy Atmel. AVR Studio umí spolupracovat se zdrojovým kódem napsaným v CodeVisionAVR pomocí tzv. COFF (Common Object File Format) souboru, který toto prostředí generuje. Při ladění zdrojového souboru je možné sledovat chování všech registrů a vstupně/výstupních periférií. Také lze prostřednictvím tohoto prostředí

odladěný zdrojový kód po kompilaci nahrát prostřednictvím podporovaných programátorů přímo do mikroprocesoru.

5.1.7 Program

Celý program pracuje v nekonečné smyčce, ve které se vypisují údaje na displej. Ošetření otočení kolečkem a stisknutí tlačítka start/stop je řešeno přes vnější přerušení INT0 až INT2.

Odpočet minut je zajištěn čítačem, který je nastaven tak, aby po vteřině, kdy dojde k jeho přetečení, vyvolal přerušení. V obsluze tohoto přerušení dojde k dekrementaci proměnné, která obsahuje vteřiny. Když je počet vteřin roven nule, sníží se i proměnná s minutami. V této funkci se také testuje, zda se počet minut nerovná nule, pokud ano, rozezní se siréna.

Příloha A obsahuje kompletní okomentovaný zdrojový kód v jazyce C. Některé komentáře jsou ponechány tak, jak je generuje program CodeWizard AVR. V příloze B nalezneme fotografie minutky při vývoji na nepájivém poli.

6 Závěr

Vývoj mikrokontrolérů jde neustále kupředu. Vznikají nové typy a roste i počet integrovaných periférií. Velký důraz je kladen na nízkou spotřebu elektrické energie. Např. novinkou firmy Atmel je v současné době 8bitový mikrokontrolér ze skupiny tinyAVR (ATtiny43U), který dokáže pracovat již od 0,7V, což je značný rozdíl oproti typickým napájecím napětím mikrokontrolérů začínajících až na 1,8V. Takto nízké napájecí napětí jej např. předurčuje pro použití v miniaturních zařízeních napájených jen z jedné tužkové baterie.

Napsání programu pro mikrokontroléry v jazyce C se díky vyspělým vývojovým prostředím velmi ulehčuje a zrychluje. Většina prostředí obsahuje i softwarový emulátor mikrokontrolérů umožňující odladit program bez nutnosti připojení ke konkrétnímu obvodu.

I v jazyce C se nadále používá assembler, který si najde své místo např. při programování přesných časových smyček, kdy je nutné počítat s dobou trvání každé instrukce. K výhodám psaní programu v jazyce C patří, že je možné se v něm snadno vyznat i po delší době, je však dobré psát do programu komentáře, které nám později značně usnadní porozumění programu.

8bitové mikrokontroléry nachází uplatnění v nenáročných elektronických obvodech, jako jsou teploměry, termostaty, nabíječky, voltmetry, hodiny atd. V dnešní době je možné většinu těchto výrobků pořídit poměrně levně a domácí výroba se proto po finanční stránce nevyplatí.

Výsledkem této práce je praktická ukázka realizace kuchyňské minutky řízená mikrokontrolérem ATmega16 s programem napsaným v jazyce C. K zobrazení nastaveného času slouží dvouřádkový textový LCD displej a ovládání je zajištěno „scrollovacím“ kolečkem z počítačové myši. O uplynutí nastaveného času informuje zvukový signál.

7 Seznam literatury

1. MANN, Burkhard. *C pro mikrokontroléry*. Přel. V. Losík. 1. vydání. Praha: BEN - technická literatura, 2003, 280 s., 1 CD-ROM. ISBN 80-7300-077-6.
2. VÁŇA, Vladimír. *Mikrokontroléry ATMEL AVR programování v jazyce C*. 1. vydání. Praha: BEN - technická literatura, 2003, 216 s., 1 CD-ROM. ISBN 80-7300-102-0.
3. MATOUŠEK, David. *Práce s mikrokontroléry ATMEL AVR ATmega16 4. díl*. 1. vydání. Praha: BEN - technická literatura, 2006, 320 s., 1 CD-ROM. ISBN 80-7300-174-8.
4. MATOUŠEK, David. *Práce s mikrokontroléry ATMEL AVR AT90S*. 2. vydání. Praha: BEN - technická literatura, 2006, 376 s., 1 CD-ROM. ISBN 80-7300-209-4.
5. POLI WEB SERVER. *Architektura počítačů* [online]. [cit. 2008-11-08]. <http://poli.cs.vsb.cz/edu/arp/down/pocitace.pdf>.
6. WIKIPEDIA. *Harvardská architektura* [online]. [cit. 2008-11-10]. <http://cs.wikipedia.org/wiki/Harvardská_architektura>.
7. DŘÍNEK, Milan. *Architektura počítačů a mikropočítačů* [online]. [cit. 2008-11-17]. < http://hw.cz/Teorie-a-praxe/Dokumentace/ART1525-Architektura-pocitacu-a-mikropocitacu.html>.
8. TIŠNOVSKÝ, Pavel. *Architektury mikroprocesorů* [online]. [cit. 2008-11-17]. <http://www.root.cz/clanky/architektury-mikroprocesoru/>.
9. OLEJÁR, Martin. *Mikrokontroléry PIC 1.díl: začínáme* [online]. [cit. 2008-12-08]. <http://www.elweb.cz/clanky.php?clanek=64>
10. EMBEDDED. *Introduction to Serial Peripheral Interface* [online]. [cit. 2008-12-08]. < http://www.embedded.com/story/OEG20020124S0116>.
11. TOMÍČEK, Petr. *Použití rozhraní TWI (I2C) v procesorech AVR I.* [online]. [cit. 2008-12-08]. <http://www.mcu.cz/print.php?news.621>.
12. ATMEL. *AVR 8-Bit RISC* [online]. [cit. 2008-12-08]. <http://www.atmel.com/products/avr/default.asp>.
13. SCRIBD. *Exploring C for Microcontrollers* [online]. [cit. 2008-12-08]. <http://www.scribd.com/doc/4769859/Exploring-c-for-Microcontrollers>.
14. ŘEHÁK, JAN. *Překladače jazyka C pro mikroprocesory řady x51* [online]. [cit. 2009-02-01]. <http://hw.cz/Teorie-a-praxe/Programovani/ART122-Prekladace-jazyka-C-pro-mikroprocesory-rady-x51.html>.
15. HEROUT, Pavel. *Učebnice jazyka C*. 3. upravené vydání. České Budějovice: KOPP, 1994, 269 s., ISBN 80-85828-21-9
16. PETERKA, Jiří. *Von Neumannova architektura* [online]. [cit. 2008-11-08]. <http://www.earchiv.cz/a93/a321c120.php3>.

8 Seznam obrázků

9 Seznam tabulek

10 Přílohy

Příloha A: Zdrojový kód

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.03.6 Evaluation
Automatic Program Generator
© Copyright 19982008 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project   : Kuchynska minutka
Version   : 1.0
Date      : 4.3.2009
Author    : Martin Neruda

Chip type              : ATmega16
Program type           : Application
Clock frequency        : 16,000000 MHz
Memory model           : Small
External RAM size      : 0
Data Stack size        : 256
*****************************************************/

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>

// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x1B; PORTA
#endasm
#include <lcd.h>

unsigned char pom[8];                  // Pomocna prom. pro prevod cisla na retezec
unsigned char minuty,vteriny;          // Promenna pro ulozeni nastavenych minut
unsigned char start;

// Definice stavu portu se sirenou
#define SIRENA_ON   PORTB.0=0
#define SIRENA_OFF  PORTB.0=1

// Externi preruseni INT0 - od inkrementalniho cidla
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Inkrementace pocitadla
   #asm("cli")                         // Zakaz preruseni
   if(++minuty>60) minuty=0;           // Vynuluje minuty pri prekroceni 60min
```

```
   if(MCUCR==0x0F) MCUCR=0x0A; // 0x0F = nabezna hrana
   else MCUCR=0x0F;            // 0x0A = sestupna hrana
   GIFR=0xC0;                  // Vynuluje priznaky preruseni
   #asm("sei")                 // Povoleni preruseni
}

// Externi preruseni INT1 - od inkrementalniho cidla
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Dekrementace pocitadla
   #asm("cli")                 // Zakaz preruseni
   if(--minuty==255) minuty=60;
   if(MCUCR==0x0F) MCUCR=0x0A; // 0x0F = nabezna hrana
   else MCUCR=0x0F;            // 0x0A = sestupna hrana
   GIFR=0xC0;                  // Vynuluje priznaky preruseni
   #asm("sei")                 // Povoleni preruseni
}

// Externi preruseni INT02 - od tlacitka
interrupt [EXT_INT2] void ext_int2_isr(void)
{
   #asm("cli")                 // Zakaz preruseni
   if((start==0)&&(minuty!=0))
   {
      GICR=0x20;               // Zakaz preruseni INT0 a INT1
      GIFR=0xC0;               // Vynulovani priznaku preruseni INT0 a INT1
      TIMSK=0x04;              // Zapnuti casovace 1
      lcd_gotoxy(0,1);
      lcd_putsf("Run ");
      start=1;
   }
   else
   {
      GICR|=0xC0;              // Povoleni preruseni INT0 a INT1
      GIFR=0xC0;               // Vynulovani priznaku preruseni INT0 a INT1
      TIMSK=0x00;              // Vypnuti casovace 1
      lcd_gotoxy(0,1);
      lcd_putsf("Stop");
      lcd_gotoxy(11,1);
      lcd_putsf("    ");
      start=0;
      minuty=0;
      vteriny=0;
   }
   #asm("sei")
}
```

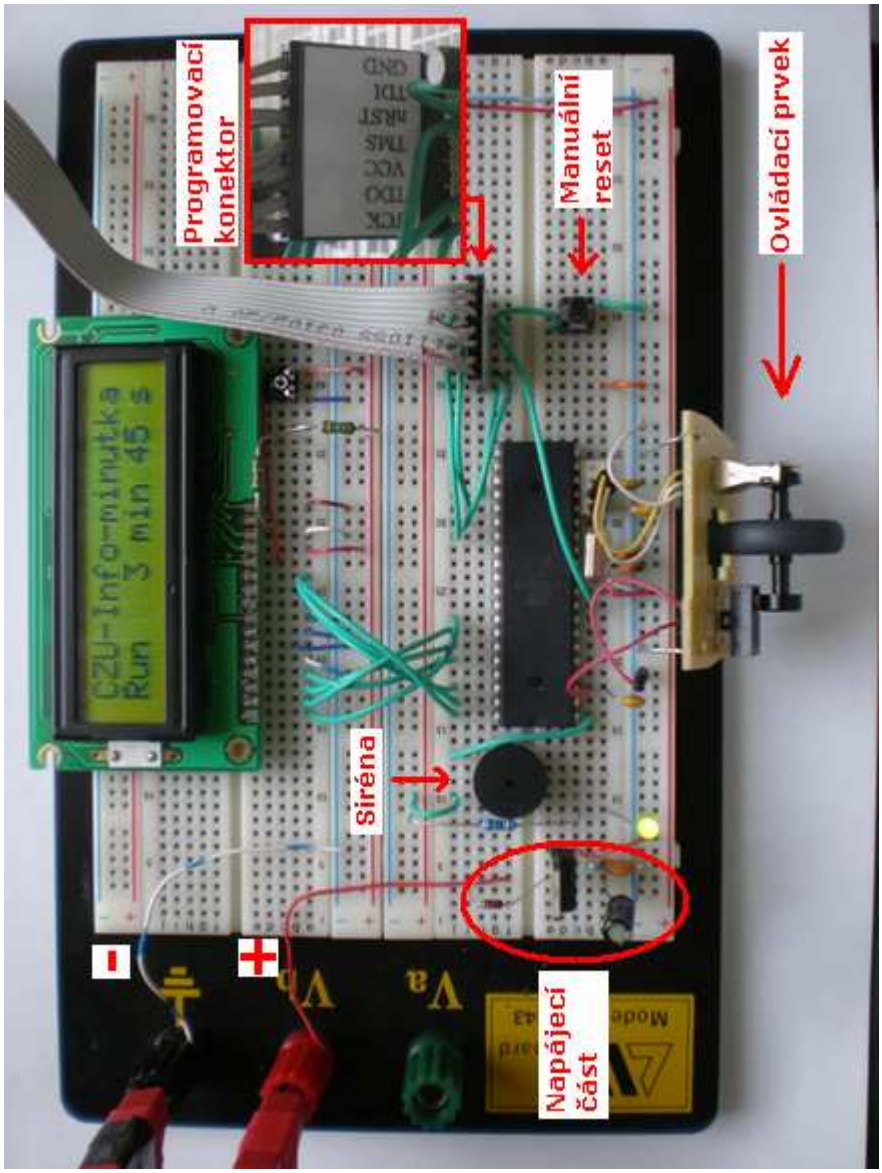
```
// Preruseni od casovace 1
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
   #asm("cli")                          // Zakaz preruseni
   // Odecte minutu z nastaveneho casu
   if((vteriny==0)&&(minuty!=0))
   {
      --minuty;
      vteriny=59;
   }
   else --vteriny;

   // Po uplynuti nastaveneho casu se spusti sirenka
   if((minuty==0)&&(vteriny==0))
   {
      lcd_gotoxy(0,1);
      lcd_putsf("Stop");
      lcd_gotoxy(12,1);
      lcd_putsf(" 0");
      start=0;
      //2x kratke a 1x dlouhe pipnuti
      SIRENA_ON;
      delay_ms(200);
      SIRENA_OFF;
      delay_ms(200);
      SIRENA_ON;
      delay_ms(200);
      SIRENA_OFF;
      delay_ms(200);
      SIRENA_ON;
      delay_ms(500);
      SIRENA_OFF;

      GICR|=0xC0;                       // Povoleni preruseni INT0 a INT1
      GIFR=0xC0;                        // Vynulovani priznaku preruseni INT0 a INT1
      TIMSK=0x00;                       // Vypnuti casovace 1
   }
   // Reinitialize Timer 1 value
   TCNT1H=0xC2;
   TCNT1L=0xF7;
   #asm("clv")
   #asm("sei")
}

// Declare your global variables here

void main(void)
```

```
{
// Deklarace lokalnich promennych
minuty=0;
vteriny=0;
start=0;

// Input/Output Ports initialization
//DDRX - urcuje smer toku dat vstup(0)/vystup(1)
//PORTX - odpovida hodnote zapsane do vyrovnavaciho registru portu
//kdyz DDRX = 1, PORTX = 0 => vystup v log.0
//                PORTX = 1 => vystup v log.1
//kdyz DDRX = 0, PORTX = 0 => pull-up odpojen
//                PORTX = 1 => pull-up pripojen

// Port A initialization
PORTA=0x00;
DDRA=0x00;

// Port B initialization
PORTB=0x05;
DDRB=0x01;

// Port C initialization
PORTC=0x00;
DDRC=0x00;

// Port D initialization
PORTD=0x0C;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 15,625 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
```

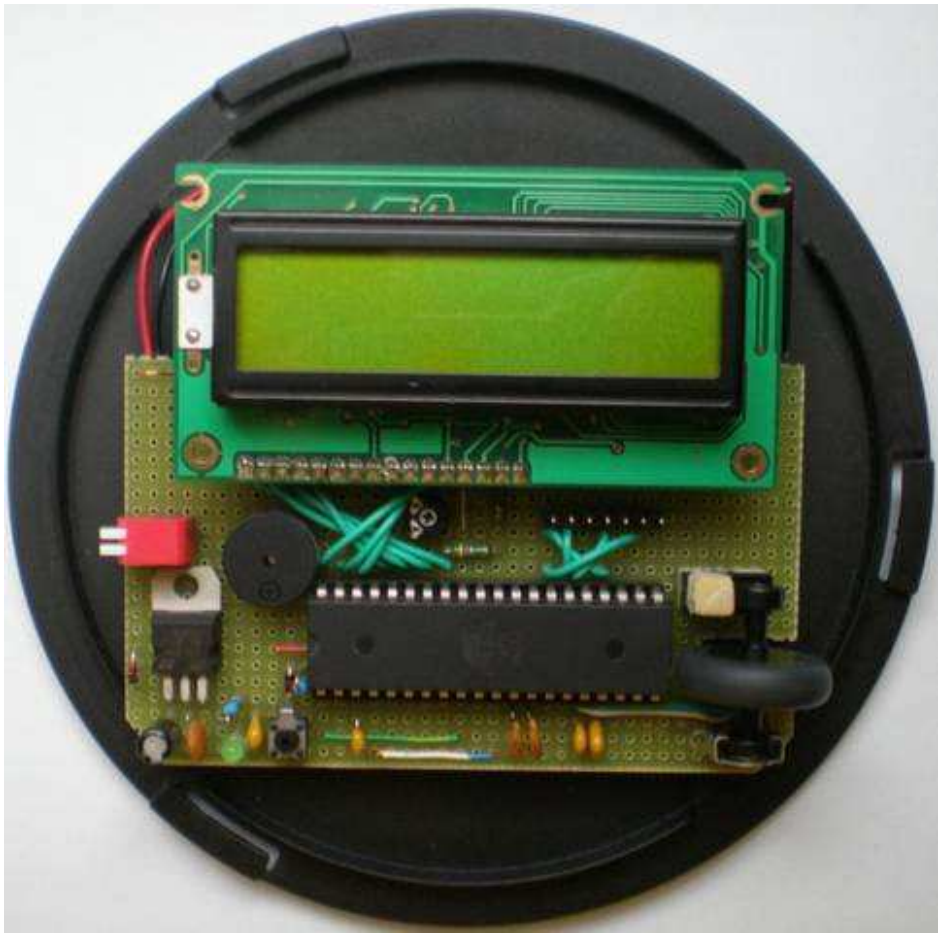
```
// Timer 1 Overflow Interrupt: On
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0xC2;
TCNT1L=0xF7;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
// INT1: On
// INT1 Mode: Rising Edge
// INT2: On
// INT2 Mode: Falling Edge
GICR|=0xE0;
MCUCR=0x0F;
MCUCSR=0x00;
GIFR=0xE0;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Inicializace LCD displeje
```

```
lcd_init(16);
// Uvodni napis na displeji
lcd_gotoxy(0,0);                    // Umisti kurzor na zadanaou pozici
lcd_putsf("CZU-Info-minutka");      // Vypis textu na LCD
lcd_gotoxy(0,1);
lcd_putsf("Stop");
lcd_gotoxy(6,1);
lcd_putsf("0");
lcd_gotoxy(8,1);
lcd_putsf("min");
lcd_gotoxy(15,1);
lcd_putsf("s");

// Povoleni globalniho preruseni
#asm("sei")

// Nekonecna smycka
while (1)
    {
    // Vypis minut
    if(minuty<10)
    {
     lcd_gotoxy(5,1);
     lcd_putsf(" ");
     lcd_gotoxy(6,1);
    }
    else lcd_gotoxy(5,1);
    itoa(minuty,pom);
    lcd_puts(pom);

    // Vypis vterin
    if(vteriny<10)
    {
     lcd_gotoxy(12,1);
     lcd_putsf(" ");
     lcd_gotoxy(13,1);
    }
    else lcd_gotoxy(12,1);
    itoa(vteriny,pom);
    lcd_puts(pom);

    delay_ms(100);
    };
}
```

Příloha B: Fotografie



Elektronický obvod minutky sestavený na nepájivém poli.



Fotografie minutky zkonstruované na univerzálním plošném spoji.