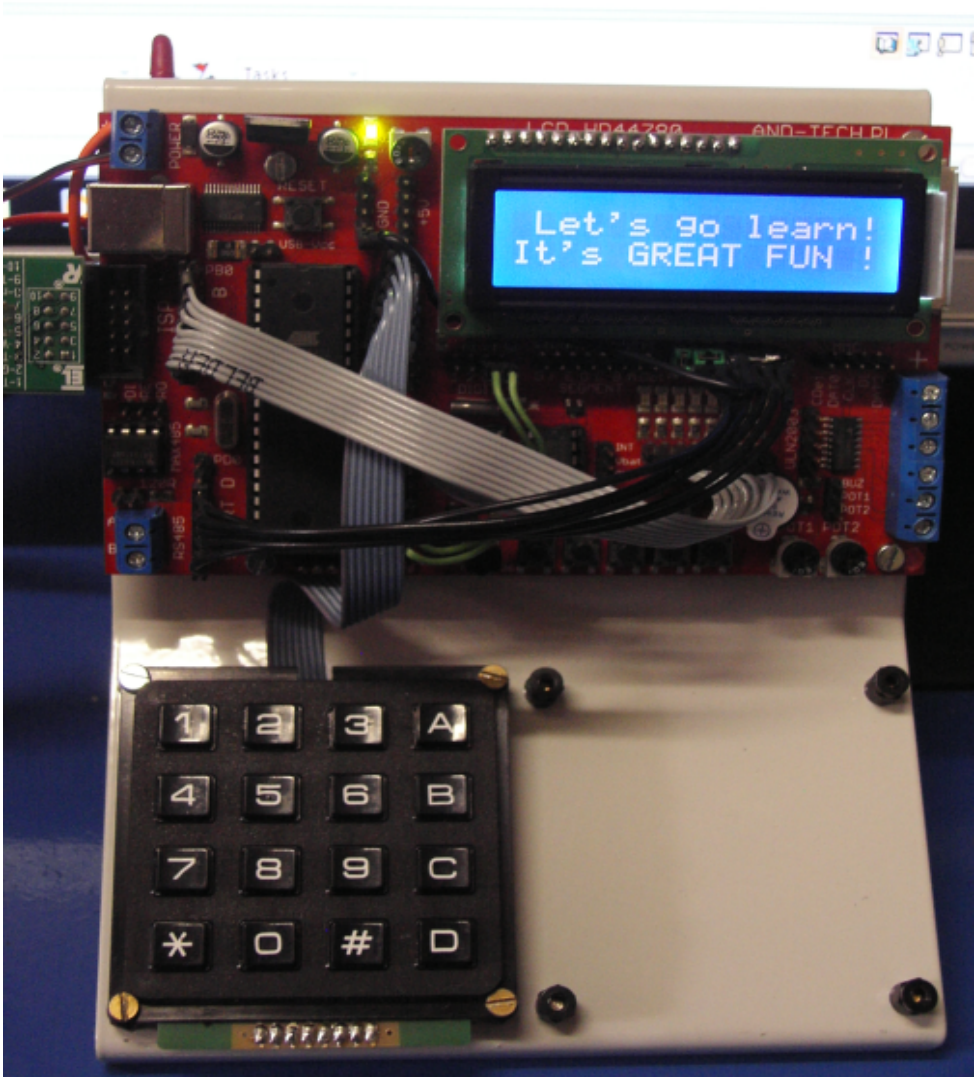
evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
PLZEŇSKÝ KRAJ
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Praktické úlohy řešené technikou jednočipových mikropočítačů

Miroslav Valečka



Obsah

1 Úvod

V dnešní době se téměř na každém kroku setkáváme v různých aplikacích s použitím jednočipových mikropočítačů a mikrokontrolérů. Uživatel těchto zařízení ani neví, že činnost těchto přístrojů je založena na využití této techniky. Mikropočítače a mikrokontroléry (rozdíl mezi nimi bude upřesněn později) nám usnadňují obsluhu různých přístrojů, zkvalitňují jejich funkci, snižují spotřebu energie, zvyšují bezpečnost zařízení, zlepšují komunikaci, … , zkrátka výčet výhod použití je velmi rozsáhlý.

Kurz, který právě začíná si klade za cíl seznámit zájemce (v našem případě hlavně žáky naší školy) s použitím mikroprocesorové techniky. Někteří žáci již absolvovali kurz mikroprocesorové techniky, ve kterém používali mikropočítač založený na jádře světoznámého mikropočítače firmy Intel řady I8051. Přestože jde o typ starý vývojově více než 30 let, je tento mikropočítač u mnoha konstruktérů na celém světě velmi oblíbený a používaný a dodnes jej vyrábí mnoho světových výrobců. Vývoj však jde nezadržitelně kupředu, proto jsme se rozhodli opustit výuku mikroprocesorové techniky s tímto typem mikropočítače a používat novější typ, který má mnoho výhod oproti typu I8051. Byl zvolen mikropočítač vyráběný firmou Atmel z řady AVR, konkrétně typ ATmega32.

Žádný mikropočítač není schopen funkce bez vloženého programu. Program vytváří uživatel, konstruktér zařízení. Pro programování se (pomineme-li úplné začátky, kdy se používal tzv. „strojový kód“, což byla kombinace „nul“ a „jedniček“ a programování bylo značně náročné) nejvíce používaly jazyky symbolických adres, pro něž se vžil název „assembler“ (i když výraz assembler je vlastně nesprávně používaný, jelikož je to překladač do strojového kódu mikropočítače). Tyto assemblery přinášely značné zjednodušení oproti programování ve strojovém kódu, jelikož pro vyjádření jednotlivých instrukcí používaly symboly - zkratky anglických slov a umožňovaly též symbolické vyjádření adres a proměnných. Tento typ programování je však značně náročný na čas a vyžaduje velkou praxi programátora. I přes některé jeho výhody je postupně opouštěn, v současné době se používají „vyšší“ programovací jazyky, pro jednočipové mikropočítače se stal nejvíce používaný jazyk C. Jazyk C byl proto zvolen pro vytváření našich úloh.

Pro vývoj aplikací nestačí vlastnit samotný mikropočítač, ten je bez připojených vnějších zařízení prakticky nepoužitelný. Aby nemusel konstruktér ztrácet čas při vývoji aplikací připojováním různých vnějších zařízení, jsou na trhu tzv. „vývojové kity“, což jsou desky osazené konkrétním typem mikropočítače, ke kterému je možné připojit celou řadu nejpoužívanějších periferních obvodů, umožňujících komunikaci s vnějším prostředím. Jsou to např. vstupní tlačítka, přepínače, potenciometry, displeje LED a LCD, piezo reproduktory, výkonové spínače, ale i speciální periferie pro komunikaci po různých sběrnicích, čtečky paměťových karet apod., záleží na tom, co umožňuje daný typ mikropočítače. Pro vývoj , resp. výuku byl zvolen vývojový kit EvB 4.3, jehož popis bude uveden v 5. kapitole skript.

Pro výuku a vývoj aplikací vyvíjí mnoho firem tzv. „vývojová prostředí“, což jsou speciální programová prostředí, umožňující napsání programu, jeho komfortní editaci, překlad do strojového kódu mikropočítače, jeho naprogramování a hlavně obsahující komfortní prostředky pro „ladění“ programu, tzv. debugging. Ne vždy (a to

hlavně při začátcích programování bez velké praxe) námi napsaný program funguje hned napoprvé. Potom je třeba najít „chybu“ v našem programu, což bývá někdy velmi náročné! V našem kurzu budeme používat vývojové prostředí „CrossWorks for AVR“ od anglické firmy Rowley.

Lze říci, že nás čeká poměrně náročný úkol, jelikož je nutné se seznámit s úplně novými pojmy, s novým programovacím jazykem, novým typem mikropočítače a s vývojovým prostředím. Tyto nové znalosti se budou prolínat takovým způsobem, aby bylo pokud možno co nejdříve řešit jednoduché úlohy s touto novou technikou. Naším cílem není vychovat z našich žáků zkušené programátory a konstruktéry (na to není bohužel dostatečný časový prostor), jde o seznámení s touto novou technikou a o vzbuzení zájmu o tento velmi zajímavý a perspektivní obor činnosti. Předpokládám, že se námi stanovený cíl podaří syplnit a někteří z řady žáků budou pokračovat ve studiu této perspektivní techniky buďto samostudiem nebo na vyšším typu škol.

2 Základní pojmy z mikroprocesorové techniky

Tato část textu je určena pro úplné začátečníky z oboru mikroprocesorové techniky. I když jsou tyto pojmy pro většinu z vás běžné, určitě není na škodu si je zopakovat (připomenout):

Bit *(z* anglického *binary digit)*

Je to nejmenší jednotka nesoucí informaci, používaná v číslicové a výpočetní technice. Značí se malým písmenem *b* , např. 8b, ale používá se též označení 8bit. Jeden bit může nabývat hodnoty 1 a 0 binární číslice, odpovídající odpovědi pravda/nepravda (TRUE/FALSE) na otázku, zda určitá skutečnost nastala či nenastala.

Bajt (anglicky *byte*)

Též se používá název slabika**,** obsahuje 8 bitů a může reprezentovat desítkové číslo v rozsahu 0 až 255 (tj. 256 kombinací, 2^8 = 256) nebo jeden znak (např. abecedy). Jeden bajt je nejmenší objem dat, se kterým pracují číslicové počítače. U většiny pamětí je velikost paměťové buňky shodná s rozměrem jednoho bajtu.

Slovo (anglicky *word*)

Je to dvojice bajtů(u 8 a 16 bitových mikroprocesorů), tj. 16 bitů. Je to počet bitů, se kterým procesor (počítač) pracuje. V programovacím jazyce *C* se označuje *int* . Má rozsah čísel 0 až 65535 (2^{16}=65536). Slovo se též používá pro adresování buněk pamětí.

Pozn.: protože bajt je malá jednotka informace, používají se před slovem bajt předpony známé z matematiky, zejména *kilo* (kB = kilobajt = 1024 bajtů) a *mega (*MB = megabajt = 1 048576 bajtů). Neplatí tedy, že 1kB = 1000B a 1MB = 1 000000B, i když se toto často používá!

Mikroprocesor

Je to složitý sekvenční číslicový obvod, tvořící základní součást počítače, který vykonává převážně aritmetické a logické operace. Postupně zpracovává jednotlivé instrukce programu, čímž realizuje námi požadovanou funkci. Program je uložen v paměti programu počítače. Aby mohl mikroprocesor komunikovat s okolím, potřebuje též různá vstupně/výstupní zařízení (rozhraní).

Jednočipový mikropočítač

Vznikne spojením mikroprocesoru, pamětí a vstupně/výstupních rozhraní do jednoho integrovaného obvodu – *čipu* (proto jednočipový). Tento integrovaný obvod má mnoho vývodů (často několik desítek) a je velmi univerzální. Bývá doplněn mnoha přídavnými funkcemi, existuje tedy mnoho typů jednočipového mikropočítače se stejným typem mikroprocesoru, říkáme se stejným *„jádrem“,* který používá stejný soubor instrukcí programu.

Jednočipový mikropočítač v sobě zahrnuje zpravidla vše potřebné k tomu, aby mohl obsáhnout celou aplikaci bez potřeby dalších podpůrných obvodů. V technické praxi se též používá pro jednočipový mikropočítač název *mikrokontrolér*, který se poněkud liší od jednočipového mikropočítače, i když pevnou hranici mezi oběma typy není možné vždy určit.

Mikrokontrolér

Je jednočipový mikropočítač vhodný pro využití v řízení. Kromě vstupních a

výstupních obvodů jsou v něm integrovány i mnohé další obvody – např. analogově-digitální nebo digitálně-analogový převodník, čítač, časovač, komparátor, synchronní sériový port, USB, PWM (pulsně-šířkový modulátor), EEPROM a další. Díky tomu, že jsou tyto obvody v mikrokontroléru již integrovány, není potřeba je realizovat externě. Mikrokontroléry se obvykle používají pro tzv. vestavné (embedded) aplikace, tj. jsou součástí nějakého dalšího zařízení, kde plní nějakou specifickou funkci. V tomto kontextu jsou na mikrokontroléry často kladeny speciální požadavky, např. nízká spotřeba, malé rozměry, nízká cena, velký rozsah pracovních teplot apod.

Paměť

Slouží k uložení programu (programová paměť) a dat (datová paměť) v mikrokontroléru.

V mikrokontrolérech se používá často tzv. Harvardská struktura, kdy je paměť programu oddělena od paměti dat (existuje též tzv. Von Neumannova struktura, kdy je společný paměťový prostor pro program i data).

Adresa

Je to číslo ve dvojkové soustavě, které udává umístění dané paměťové buňky v paměti. Počet bitů adresy určuje kapacitu paměti. Např. pro 16-bitovou adresu bude maximální kapacita paměti 2^{16} = 65536 buněk (adresy od 0 do 65535). Jelikož by bylo psaní adresy ve dvojkové soustavě nepřehledné a zdlouhavé, používáme pro zápis adresy soustavu hexadecimální (šestnáctkovou).

Sběrnice

Jsou to skupiny vodičů, které propojují vstupně/výstupní zařízení, paměti a jiná speciální zařízení s procesorem počítače. Pro sběrnice jsou definovány napěťové úrovně a časové posloupnosti přicházejících signálů. Uvnitř počítače (i mikrokontroléru) se používají tyto typy sběrnic: datová (přenáší data), adresová (přenáší adresu osloveného zařízení) a řídící (slouží k synchronizaci přenosu signálů).

Registr

Používá se k dočasnému uložení dat, operandů instrukce a výsledků operace. Registry jsou součástí mikrokontroléru, podle typu mikrokontroléru jich je několik málo až několik desítek.

Zásobník (anglicky *stack*)

Je to speciální paměť, která se používá k dočasnému ukládání obsahu registrů (pokud registry potřebuje použít jiná část programu) a k ukládání návratové adresy při volání podprogramů. Způsob ukládání je typu LIFO (Last In First Out = „poslední dovnitř první ven“), tzn. naposledy uložená informace je na vrcholu zásobníku a při výběru se jako první přečte. Tzv. „vrchol“ zásobníku je uložen v registru SP.

Instrukce

Je to nejmenší programová jednotka, provádí elementární operace, např. součet obsahu dvou registrů a uložení výsledku operace do registru. Smysluplná posloupnost instrukcí se nazývá program.

Instrukční soubor

Je to soubor všech instrukcí, které mikrokontrolér umí vykonat. Jejich počet je několik desítek (některý mikrokontrolér zná i více než sto instrukcí).

ALU (Arithmetical Logical Unit)

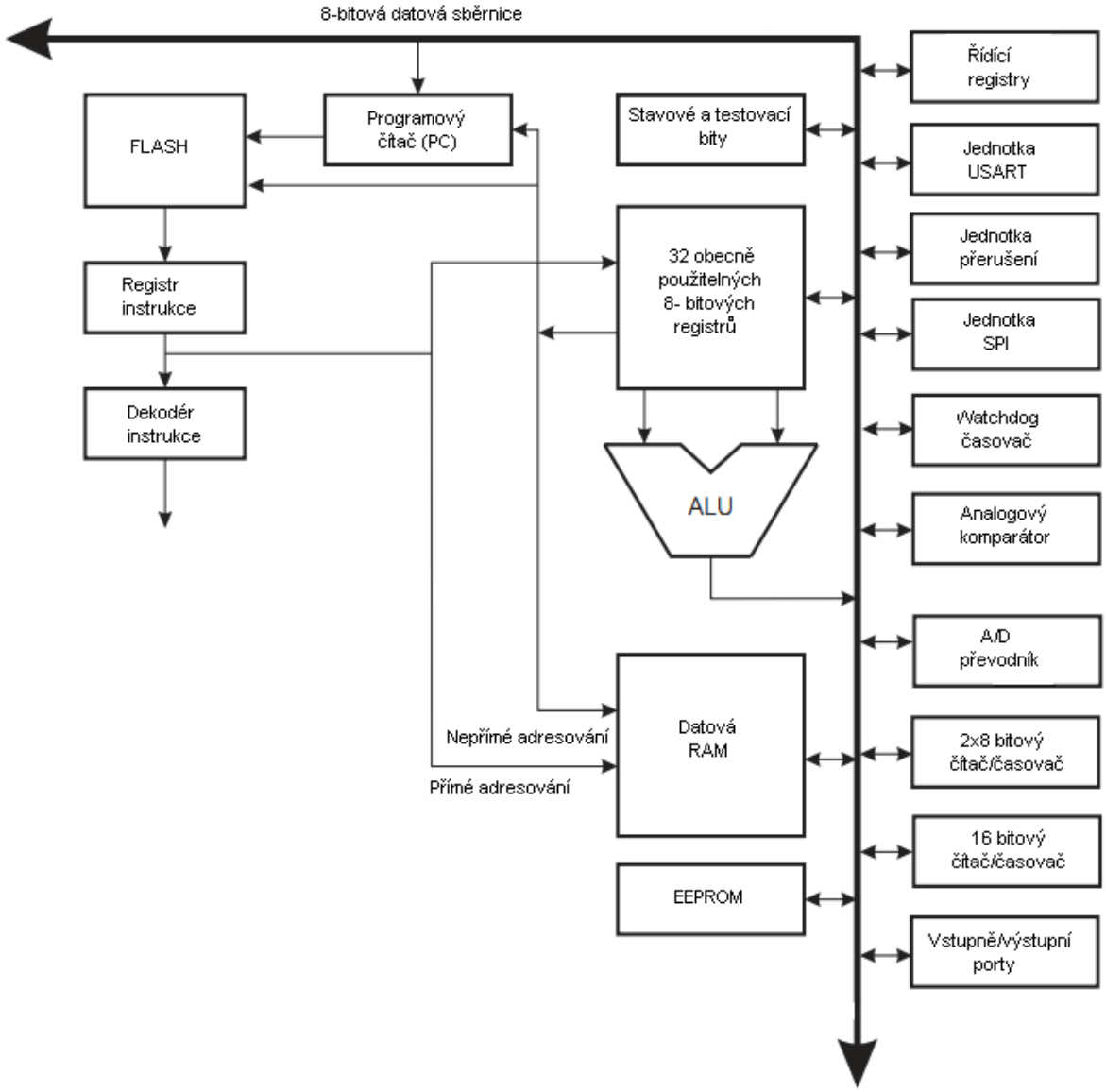
Je to výkonná část procesoru, která provádí aritmetické (součet, rozdíl, součin, podíl) a logické operace (logický součet, logický součin, negaci, …).

3 Základní vlastnosti mikrokontroléru AT mega 32

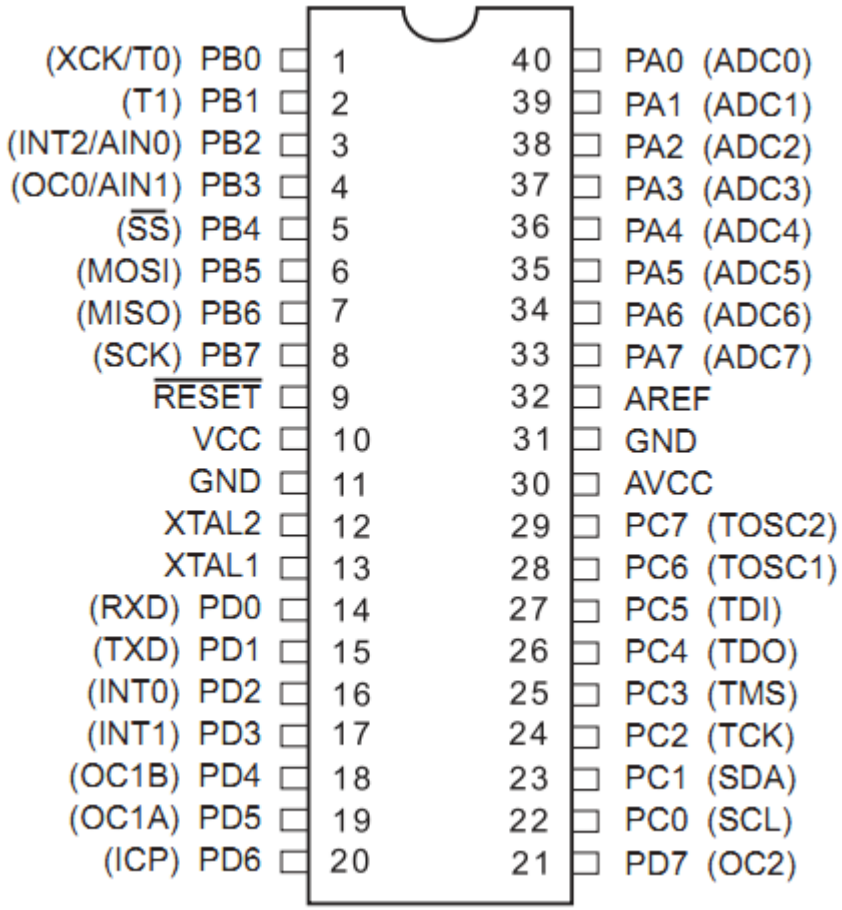
Rodina mikrokontrolérů AVR (nehledejte pod touto zkratkou žádné technické termíny, je to zkratka vytvořená z počátečních písmen jmen tvůrců Alf (Egil Bogen), Vegard (Wollan) RISC procesor) je zdařilým výsledkem architektury mikrokontrolérů s architekturou RISC (Redused Instruction Set Computing) – procesorů s redukovanou instrukční sadou. Základní rysy jsou tyto:

- Architektura typu Harvard a RISC.
- 32 identických 8-bitových registrů pro všeobecné použití (Register File), které jsou všechny použitelné jako akumulátor.
- Programové a datové paměti
  - 32 kB paměti typu Flash pro program
  - 1024 B datové paměti EEPROM pro ukládání dat
  - 2 kB statické paměti RAM pro registry, oblast I/O a data
- JTAG rozhraní, použitelné pro
  - Programování Flash, EEPROM, Fuses a Lock Bits
  - Ladění (debugging) programu
  - Boundary Scan – technika dovolující sledování a nastavení úrovní na I/O bez přídavných periferií pouze za podpory vývojového prostředí
- Periferie:
  - Dva 8-bitové čítače/časovače
  - Jeden 16-bitový čítač/časovač
  - Hodiny reálného času se samostatným oscilátorem
  - Čtyři PWM výstupní kanály
  - 8-kanálový, 10-bitový A/D převodník
  - TWI (2-vodičová seriová sběrnice)
  - SPI sběrnice (Serial Peripheral Interface)
  - Sériová komunikace typu USART
  - Programovatelný Watchdog časovač se samostatným oscilátorem na čipu
  - Analogový komparátor
  - 32 programovatelných I/O linek

4 Programátorský model AT Mega 32



Obr.1: Blokové schema AT Mega 32



Obr.2: Zapojení pouzdra DIP 40

Popis vývodů:

VCC napájecí napětí

GND zem

Port A (PA7 ..PA0) 8 bitový obousměrný I/O port, alternativní použití pro vstupy A/D převodníku

Port B (PB7 ..PB0) 8 bitový obousměrný I/O port, alternativní použití vývodů bude popsáno v dalším textu

Port C (PC7 ..PC0) 8 bitový obousměrný I/O port, alternativní použití vývodů bude popsáno v dalším textu

Port D (PD7 ..PD0) 8 bitový obousměrný I/O port, alternativní použití vývodů bude popsáno v dalším textu

RESET vstup reset kontroléru, aktivní na „0“

XTAL1 připojení krystalu

XTAL2 připojení krystalu

AVCC napájecí napětí pro A/D převodník

AREF referenční napětí pro A/D převodník

Všechny 8 bitové I/O porty (PA až PD) jsou obousměrné, umožňují připojit vnitřní zvyšující (Pull Up) rezistory. Maximální výstupní proud jednotlivých pinů je 40mA, což umožňuje přímé připojení LED, celkový odebíraný proud z I/O portů nesmí překročit 400mA, podrobnější informace jsou uvedeny v katalogovém listu. Většina pinů portů má alternativní použití pro speciální periferie. Pokud se tyto piny použijí, nemohou se dále použít jako obecné I/O. Význam alternativního použití je popsán stručně v Tab.1.

Vývod	Význam
RESET	nulovací vstup
AIN0	vstup analogového komparátoru (+)
AIN1	vstup analogového komparátoru (-)
SS	Slave Select kanálu SPI
MOSI	Master Out/Slave In kanálu SPI
MISO	Master In/Slave Out kanálu SPI
SCK	hodinový signál kanálu SPI
RxD	vstup USART
TxD	výstup USART
XCK	hodiny pro synchronní režim USART
INT0	vstup vnějšího přerušení 0
INT1	vstup vnějšího přerušení 1
INT2	vstup vnějšího přerušení 2
T0	hodinový vstup čítače/časovače 0
T1	hodinový vstup čítače/časovače 1
OC0	Output Compare čítače/časovače 0
OC1A	Output Compare čítače/časovače 1 (kanál A)
OC1B	Output Compare čítače/časovače 1 (kanál B)
OC2	Output Compare čítače/časovače 2
ICP	Input Capture čítače/časovače 1
TDI	vstupní data JTAG rozhraní
TDO	výstupní data JTAG rozhraní
TMS	výběr režimu JTAG rozhraní
TCK	hodinový signál JTAG rozhraní
SDA	datový signál TWI (I2C) rozhraní
SCL	hodinový signál TWI (I2C) rozhraní
ADC0 až ADC7	kanály A/D převodníku

Tab.1: Alternativní použití vývodů ATmega32

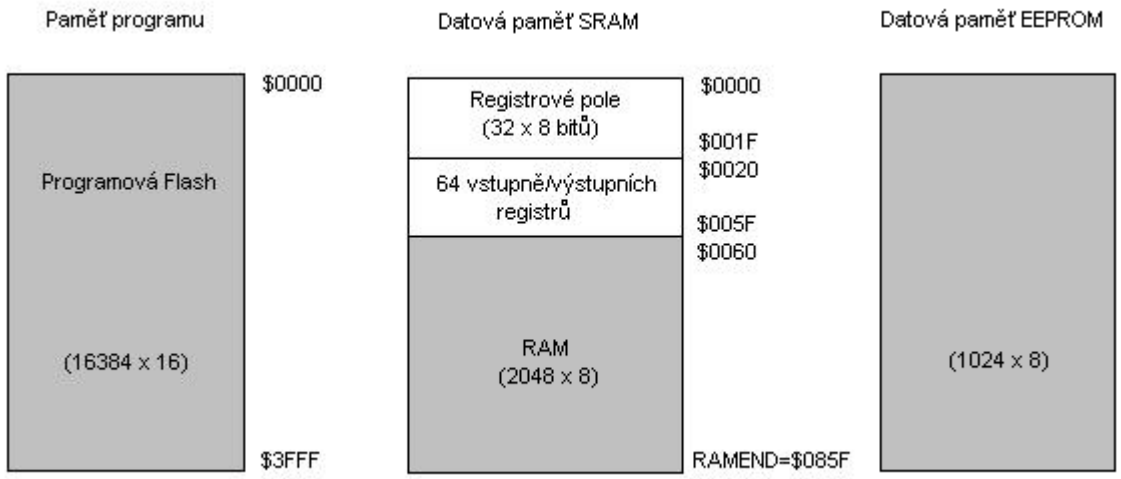
4.1 Paměťový prostor

Mikrokontroléry řady AVR používají harvardskou strukturu, kde je oddělen paměťový prostor pro program a pro data. Mikrokontrolér ATmega32 nemá podporu pro vytvoření vnějšího paměťového prostoru. Používá tedy pouze vnitřní paměti, které jsou typu Flash, RAM a EEPROM. Mapa vnitřních pamětí je na Obr.3.

4.1.1 Vnitřní programová paměť

ATmega32 obsahuje 32kB v systému reprogramovatelnou Flash paměť pro uložení programu. Jelikož instrukce AVR procesorů jsou 16bitové nebo 32bitové, je organizace paměti 16kB x 16bitů. Tato paměť je minimálně 10000x reprogramovatelná.

Program Counter (PC, programový čítač) ATmega32 je 14bitový, může adresovat 16k programové paměti.



Obr.3: Mapa vnitřních pamětí ATmega32

4.1.2 Vnitřní datová paměť SRAM

Tato paměť je tvořena statickými buňkami (Static RAM). Obsahuje pole 32 obecně použitelných registrů R0 až R31, pole 64 vstupně/výstupních registrů pro řízení periferií a blok volně použitelné paměti.

4.1.3 Vnitřní datová paměť EEPROM

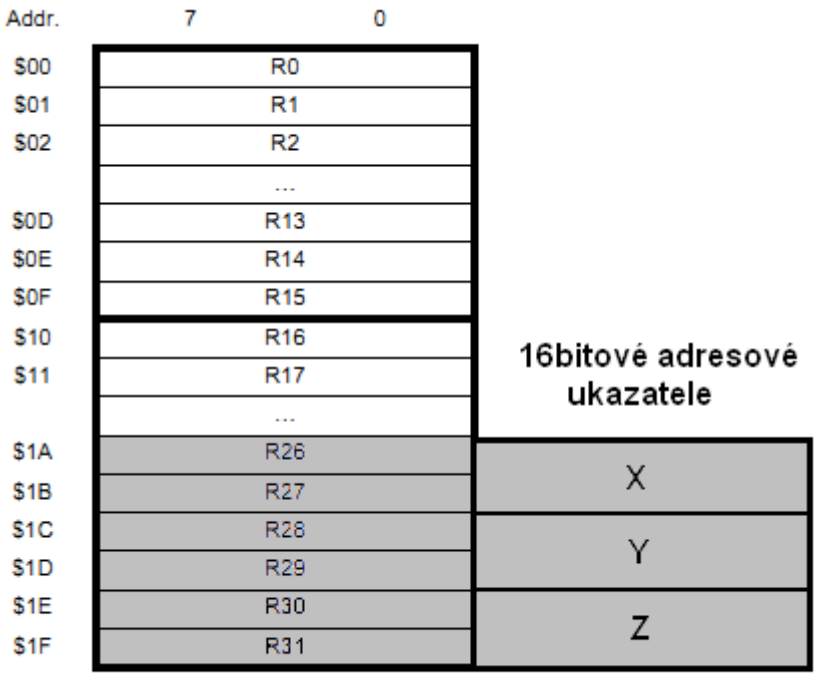
EEPROM (Electrically Eraseable Programable ROM) je paměť, která si uchová svůj obsah i po vypnutí napájení mikrokontroléru (až několik desítek let), používá se tedy pro uchování určitých dat, např. nastavení předvolených stanic autorádia, konfiguračních parametrů různých přístrojů apod. Zápis do této paměti je časově podstatně delší než zápis do SRAM (asi 8,5 ms/1byte), pro přístup k této paměti se používají speciální registry mikrokontroléru. Počet přeprogramování je min. 100000x. Mikrokontrolér ATmega32 je vybaven celou řadou registrů. Nyní uvedeme stručný popis základních registrů.

4.2 Registrové pole (Registr File)

Je tvořeno z 32 všeobecně použitelných 8bitových registrů. S těmito registry pracují přímo instrukce programu, jsou v nich uloženy operandy vstupující do ALU (viz blokové schema architektury AVR) a ukládá se do nich i výsledek operace. Ke každému registru je možné přistupovat jeho jménem (R0 až R31) nebo jeho adresou ($0000 až $001F).

4.2.1 Ukazatele (pointery)

Ukazatele se používají k nepřímému adresování paměti. Mikrokontroléry řady AVR mají 3 ukazatele: X,Y a Z. Ukazatele jsou 16bitové a jsou tvořeny dvojicemi registrů, počínaje registrem R26. Adresy registrů jsou znázorněny na Obr.4.



Obr. 4: Registry ATmega32

Při nepřímém adresování je do ukazatele vložena 16bitová adresa paměťové buňky, se kterou chceme pracovat. Při blokovém zpracování dat (přesun bloku dat, prohledávání bloku dat, …) se do ukazatele vloží počáteční adresa bloku, po provedení patřičné instrukce se pouze inkrementuje (zvýší o 1) nebo dekrementuje (sníží o 1) obsah ukazatele a tím pracujeme s následující buňkou paměti. Tuto inkrementaci či dekrementaci obsahují některé speciální instrukce a provádějí ji automaticky před (predekrement) přístupem do paměti nebo po (postinkrement) přístupu do paměti v rámci zpracování přesunu a není tedy potřebná další instrukce pro modifikaci obsahu ukazatele, čímž se značně urychlí běh programu.

4.3 Vstupně/výstupní registry (I/O Memory)

Tyto registry jsou uloženy v datové paměti mikrokontroléru od adresy $0020 až $005F (64 buněk). Slouží k řízení zabudovaných periferií a jádra mikrokontroléru (čítačů/časovačů, přerušovacího systému, komunikačních rozhraní SPI, TWI, USART, EEPROM, A/D převodníku, obsahují též stavový registr a registr SP...). Přehled registrů je uveden v Tab.2.

Adresa	Registr	Popis funkce
$00 ($20)	TWBR	Registr přenosové rychlosti TWI rozhraní
$01 ($21)	TWSR	Stavový registr TWI rozhraní
$02 ($22)	TWAR	Adresový registr TWI rozhraní
$03 ($23)	TWDR	Datový registr TWI rozhraní
$04 ($24)	ADCL	Datový registr A/D převodníku (dolní bajt)
$05 ($25)	ADCH	Datový registr A/D převodníku (horní bajt)
$06 ($26)	ADCSRA	Řídící a stavový registr A/D převodníku
$07 ($27)	ADMUX	Řídící registr multiplexeru A/D převodníku
$08 ($28)	ACSR	Řídící a stavový registr analogového komparátoru
$09 ($29)	UBRRL	Registr přenosové rychlosti USART (dolní bajt)
$0A ($2A)	UCSRB	Řídící/stavový registr B USART
$0B ($2B)	UCSRA	Řídící/stavový registr A USART
$0C ($2C)	UDR	Datový registr USART
$0D ($2D)	SPCR	Řídící registr SPI
$0E ($2E)	SPSR	Stavový registr SPI
$0F ($2F)	SPDR	Datový registr SPI
$10 ($30)	PIND	Vstupní vývody portu D
$11 ($31)	DDRD	Směrový registr portu D
$12 ($32)	PORTD	Datový registr portu D
$13 ($33)	PINC	Vstupní vývody portu C
$14 ($34)	DDRC	Směrový registr portu C
$15 ($35)	PORTC	PORTC
$16 ($36)	PINB	Vstupní vývody portu B
$17 ($37)	DDRB	Směrový registr portu B
$18 ($38)	PORTB	Datový registr portu B
$19 ($39)	PINA	Vstupní vývody portu A
$1A ($3A)	DDRA	Směrový registr portu A
$1B ($3B)	PORTA	Datový registr portu A
$1C ($3C)	EECR	Řídící registr EEPROM
$1D ($3D)	EEDR	Datový registr EEPROM
$1E ($3E)	EEARL	Adresový registr EEPROM (dolní bajt)
$1F ($3F)	EEARH	Adresový registr EEPROM (horní bajt)
$20 ($40)	UCSRC	Řídící/stavový registr C USART
$20 ($40)	UBRRH	Registr přenosové rychlosti USART (horní bajt)
$21 ($41)	WDTCR	Řídící registr WDT

$22 ($42)	ASSR	Stavový registr asynchronního režimu čítače/časovače 2
$23 ($43)	OCR2	Kompareční registr čítače/časovače 2
$24 ($44)	TCNT2	Registr čítače/časovače 2
$25 ($45)	TCCR2	Řídící registr čítače/časovače 2
$26 ($46)	ICL1L	Záchytný registr čítače/časovače 1 (dolní bajt)
$27 ($47)	ICL1H	Záchytný registr čítače/časovače 1 (horní bajt)
$28 ($48)	OCR1BL	Komparační registr B čítače/časovače 1 (dolní bajt)
$29 ($49)	OCR1BH	Komparační registr B čítače/časovače 1 (horní bajt)
$2A ($4A)	OCR1AL	Komparační registr A čítače/časovače 1 (dolní bajt)
$2B($4B)	OCR1AH	Komparační registr A čítače/časovače 1 (horní bajt)
$2C($4C)	TCNT1L	Registr čítače/časovače 1 (dolní bajt)
$2D ($4D)	TCNT1H	Registr čítače/časovače 1 (horní bajt)
$2E ($4E)	TCCR1B	Řídící registr B čítače/časovače 1
$2F ($4F)	TCCR1A	Řídící registr A čítače/časovače 1
$30 ($50)	SFIOR	Registr speciálních funkcí I/O
$31 ($51)	OSCCAL	Kalibrační registr vnitřního RC oscilátoru
$31 ($51)	OCDR	Registr pro podporu ladění na čipu
$32 ($52)	TCNT0	Registr čítače/časovače 0
$33 ($53)	TCCR0	Řídící registr čítače/časovače 0
$34 ($53)	MCUCSR	Řídící/stavový registr mikrokontroléru
$35 ($55)	MCUCR	Řídící registr mikrokontroléru
$36 ($56)	TWCR	Řídící registr TWI
$37 ($57)	SPMCR	Řídící registr pro zápis do programové paměti
$38($58)	TIFR	Registr příznaků přerušení od čítačů/časovačů
$39 ($59)	TIMS	Masky přerušení od čítačů/časovačů
$3A ($5A)	GIFR	Stavový registr přerušovacího systému
$3B ($5B)	GICR	Hlavní řídící registr přerušení
$3C ($5C)	OCR0	Komparační registr čítače/časovače 0
$3D ($5D)	SPL	Ukazatel zásobníku (nižší bajt)
$3E ($5E)	SPH	Ukazatel zásobníku (vyšší bajt)
$3F($5F)	SREG	Stavový registr

Tab.2: Přehled registrů Atmega32

Nyní popíšeme podrobně význam a obsah dvou obecně používaných registrů mikrokontrolérů:

4.3.1 Stavový registr SREG (State Register)

Tento registr obsahuje příznaky, které definují stav mikrokontroléru po vykonání programové instrukce. Obsah stavového registru je definován následovně:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Výchozí hodnota	0	0	0	0	0	0	0	0	

Význam jednotlivých bitů:

- C – (Carry Flag) příznak přetečení – indikuje přetečení po aritmetické operaci, tzn. výsledek je větší než č. 255 (největší číslo v 8bitovém registru).
- Z – (Zero Flag) příznak nulového výsledku – indikuje nulový výsledek po aritmetické nebo logické operaci.
- N – (Negative Flag) příznak negativního výsledku – indikuje záporný (negativní) výsledek po aritmetické nebo logické operaci.
- V – (Two's Complement Overflow Flag) příznak přeplnění čísla v druhém doplňku – podporuje aritmetiku dvojkového doplňku.
- S – (Sign) znaménkový bit – indikuje znaménko čísla v dvojkovém doplňku.
- H – (Half Carry Flag) – indikuje poloviční přenos (mezi dolní a horní polovinou bajtu) výsledku, používají jej instrukce pracující s BCD kódem.
- T – (Bit Copy Storage) – kopírovací bit, používají jej instrukce BLD (Bit LoaD) a BST (Bit STorage) jako zdrojový nebo cílový bit. Bit z registru registrového pole může být kopírován do bitu T instrukcí BST a opět uložen z T do bitu registru z registrového pole instrukcí BLD.
- I – (Global Interrupt Enable) – globální povolení přerušení. Je-li I=0, je příjem přerušení zakázán, což se provede po vstupu do obsluhy přerušení. Po vykonání obsluhy přerušení se instrukcí RETI opět uvede bit I do stavu I=1.

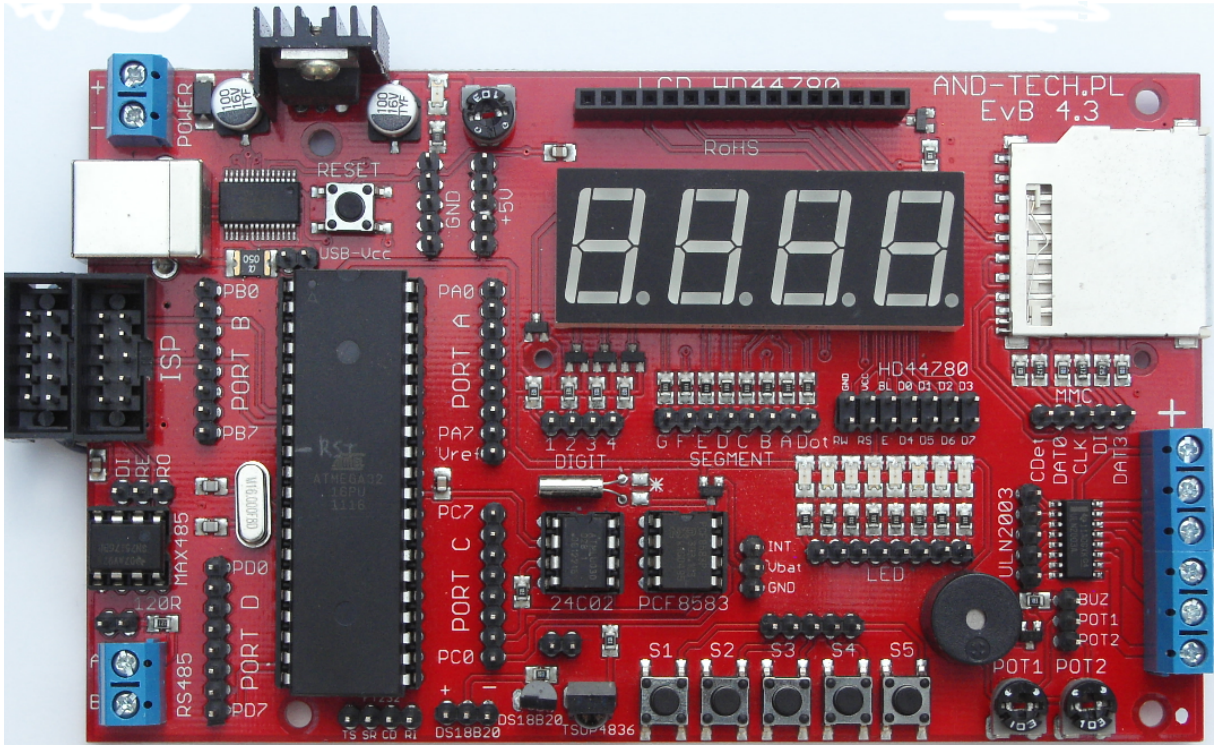
4.3.2 Ukazatel vrcholu zásobníku SP (Stack Pointer)

Zásobníková paměť je používána zejména pro dočasné uložení dat, pro uložení lokálních proměnných a pro uložení návratové adresy po volání podprogramů nebo podprogramů pro obsluhu přerušení. Tento registr (tvořený dvěma registry SPL a SPH) ukazuje na vrchol zásobníkové paměti, tj. na posledně uložený bajt. Při vložení dalšího údaje se obsah SP (adresa) snižuje, při výběru údaje se obsah SP zvyšuje (jde o paměť typu LIFO – Last In First Out). Prostor pro zásobníkovou paměť v datové oblasti SRAM musí být v programu definován před prvním voláním podprogramu nebo před povolením obsluhy přerušení, jelikož výchozí hodnota je $00. Registr SP musí být nastaven nad adresu $60, kde začíná oblast volně použitelné RAM. Obvyklá hodnota je RAMEND=$085F. Obsah ukazatele vrcholu zásobníku je definován následovně:

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Výchozí hodnota	0	0	0	0	0	0	0	0	
Výchozí hodnota	0	0	0	0	0	0	0	0	

Ostatní registry budou popsány při popisu obsluhy jednotlivých zabudovaných periferií mikrokontroléru.

5 Vývojový kit EvB 4.3



Tato vývojová deska (kit) je pomůcka pro vývoj aplikací s mikrokontroléry Atmel AVR AT mega 16, AT mega 32 a AT mega 64.

Deska je osazena mnoha periferiemi, čidly a akčními členy, které jsou připojeny na hřebínkové konektory. Toto provedení umožňuje velmi snadnou a rychlou konfiguraci desky podle právě vyvíjené aplikace, v našem případě řešené školní úlohy.

Jednotlivé periferie se připojují bez nutnosti pájení jednopólovými nebo vícepólovými propojovacími kablíky. Všechny periferie a konektory jsou pečlivě popsány, změna zapojení je intuitivní a lze ji provést bez potřeby použití manuálu.

Kit EvB 4.3 je určen jak pro začátečníky, tak i pro zkušené vývojáře, kteří její pomocí minimalizují čas potřebný k vývoji aplikace. Deska je velmi vhodná pro výuku mikroprocesorových aplikací jak na středních školách a učilištích, tak i na odborných školách vyšších stupňů.

Obsah vývojové desky EvB 4.3

- mikrokontrolér AVR AT mega 32 v provedení DIL 40
- tlačítko – 5x
- LED indikátor – 8x
- 4místný sedmisegmentový LED displej
- podsvícený LCD displej 2x16 znaků
- analogové otenciometry – 2x
- výkonový výstup 500mA - 6x
- interface USB
- interface ISP (pro programování)
- převodník RS232/RS485
- sériová paměť EEPROM I2C – AT24C02
- hodiny reálného času – RTC (Real Time Clock) – PCF 8583
- přijímač IR (infračerveného záření) na 36kHz – TSOP4836
- teplotní senzor – 1wire Dallas sběrnice – DS18B20
- patice pro MMC/SD paměťové karty
- volné napájecí piny +5V DC – 5x
- volné zemnicí piny – 5x

Podrobný popis jednotlivých periferií bude uveden při jejich prvním použití v jednotlivých úlohách.

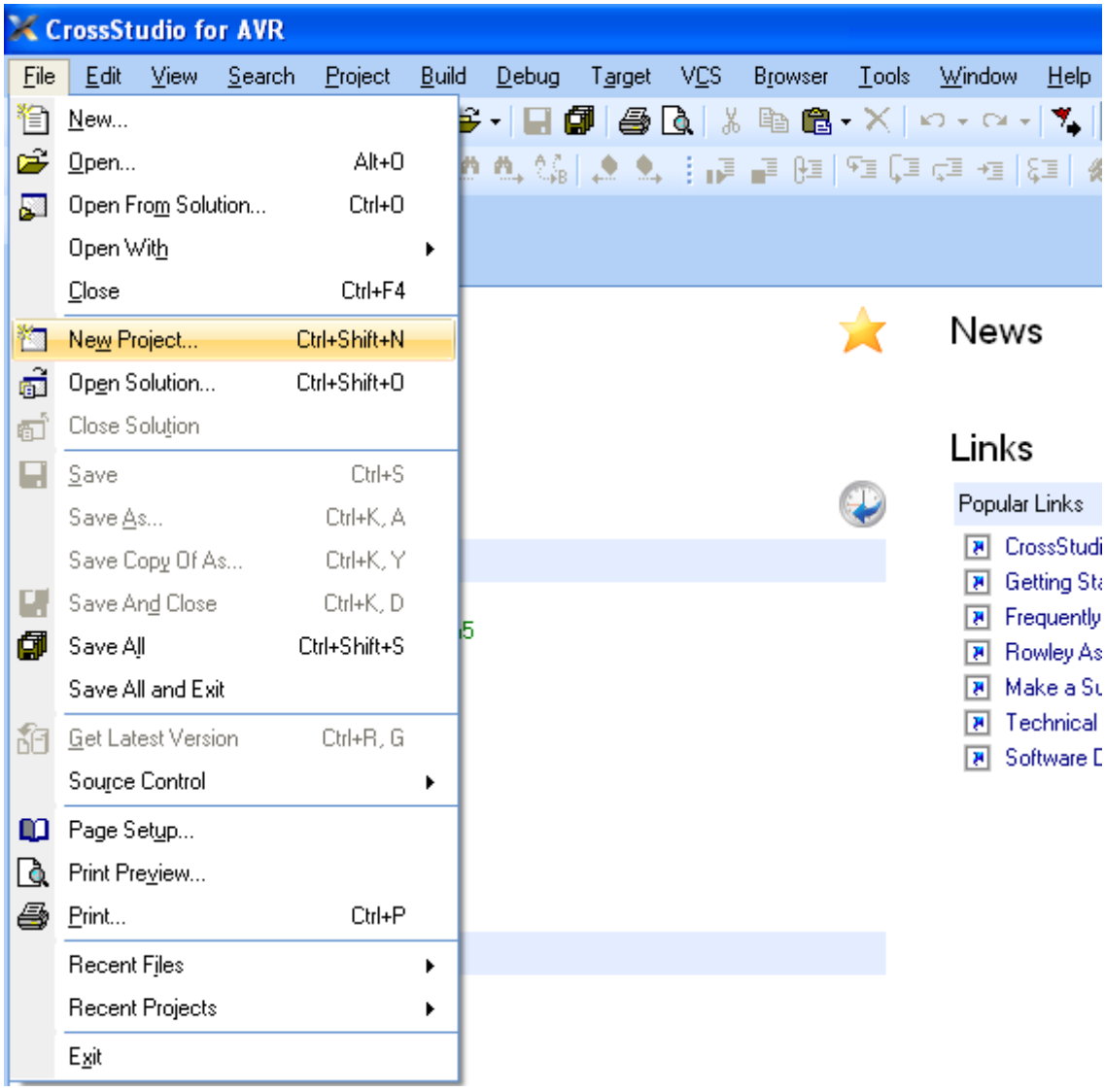
6 Vývojové prostředí CrossStudio

V následujícím textu se seznámíme se základní obsluhou vývojového prostředí a se způsobem vložení programu do vnitřní paměti mikrokontroléru.

6.1 Vytvoření nového projektu

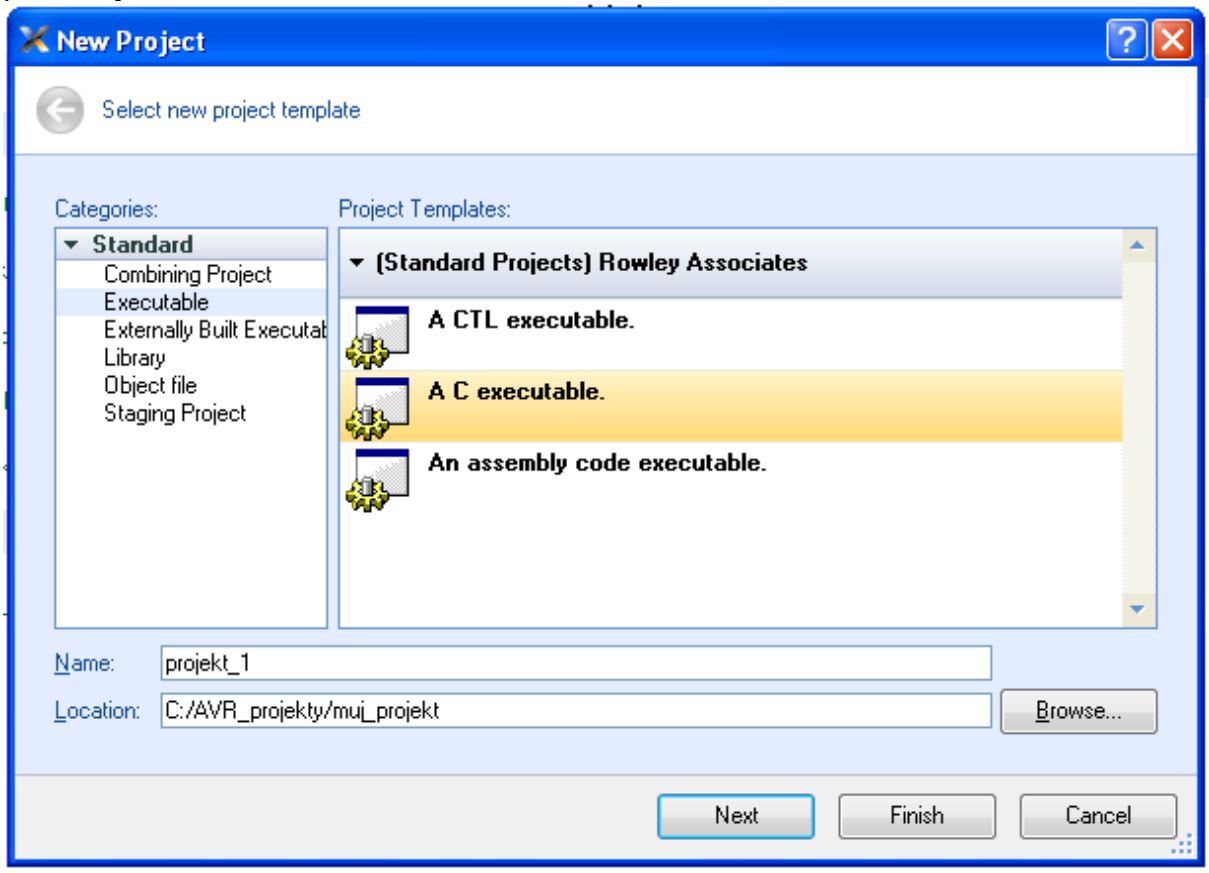
Je možné vytvořit novou „solution“ (česky se hodí snad nejlépe výraz „řešení“, jde o soubor programů podobného typu, projektu, ...) pro každý projekt nebo je možné vytvořit projekt v již existující „solution“. Každý projekt (v našem případu příklad) může navíc obsahovat více souborů .C, pouze v jednom souboru však může být použita funkce s označením „main“.

V **File** menu, vyberte **New Project**, abychom zobrazili průvodce vytvoření nového projektu:

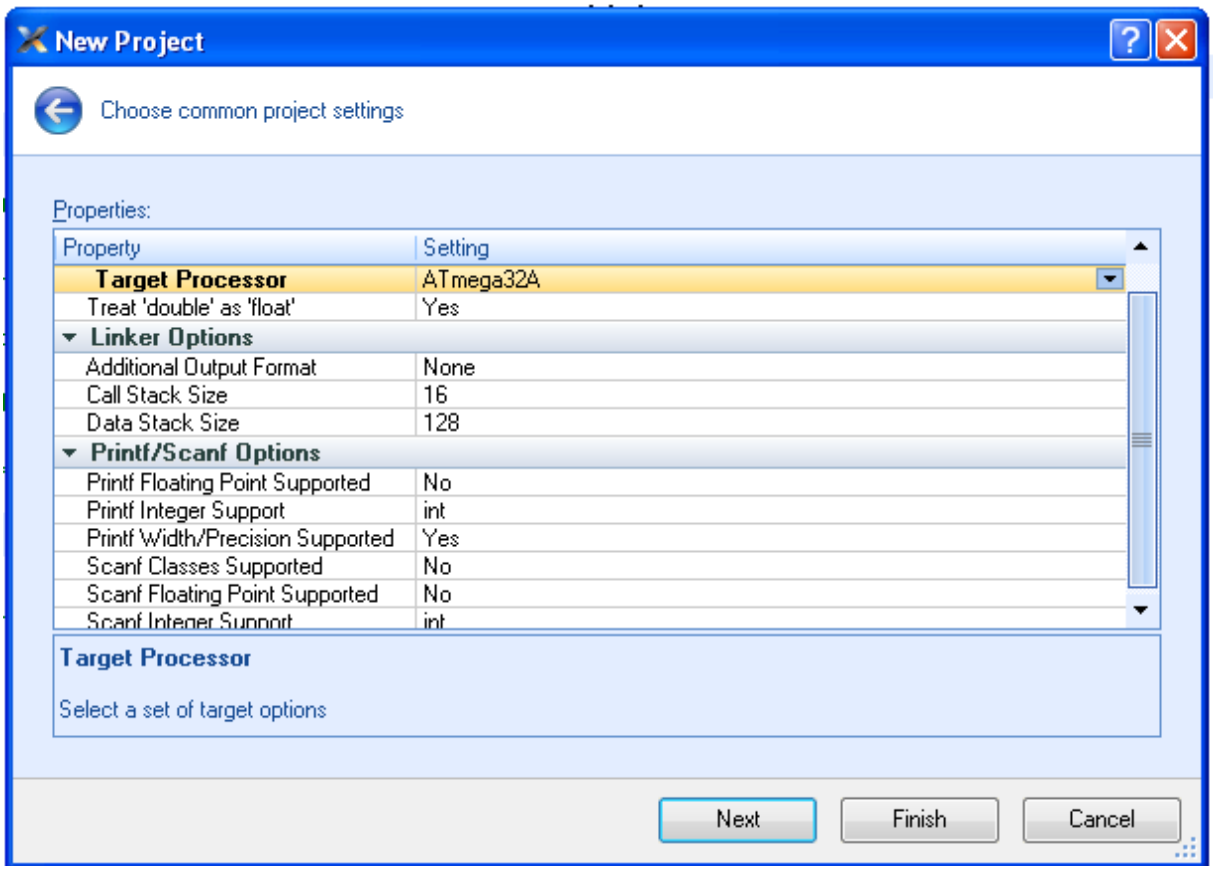


V okně **New Project** vyberte v poli **Categories** položku **Executable** a v poli **Project Templates** položku **A C executable.**

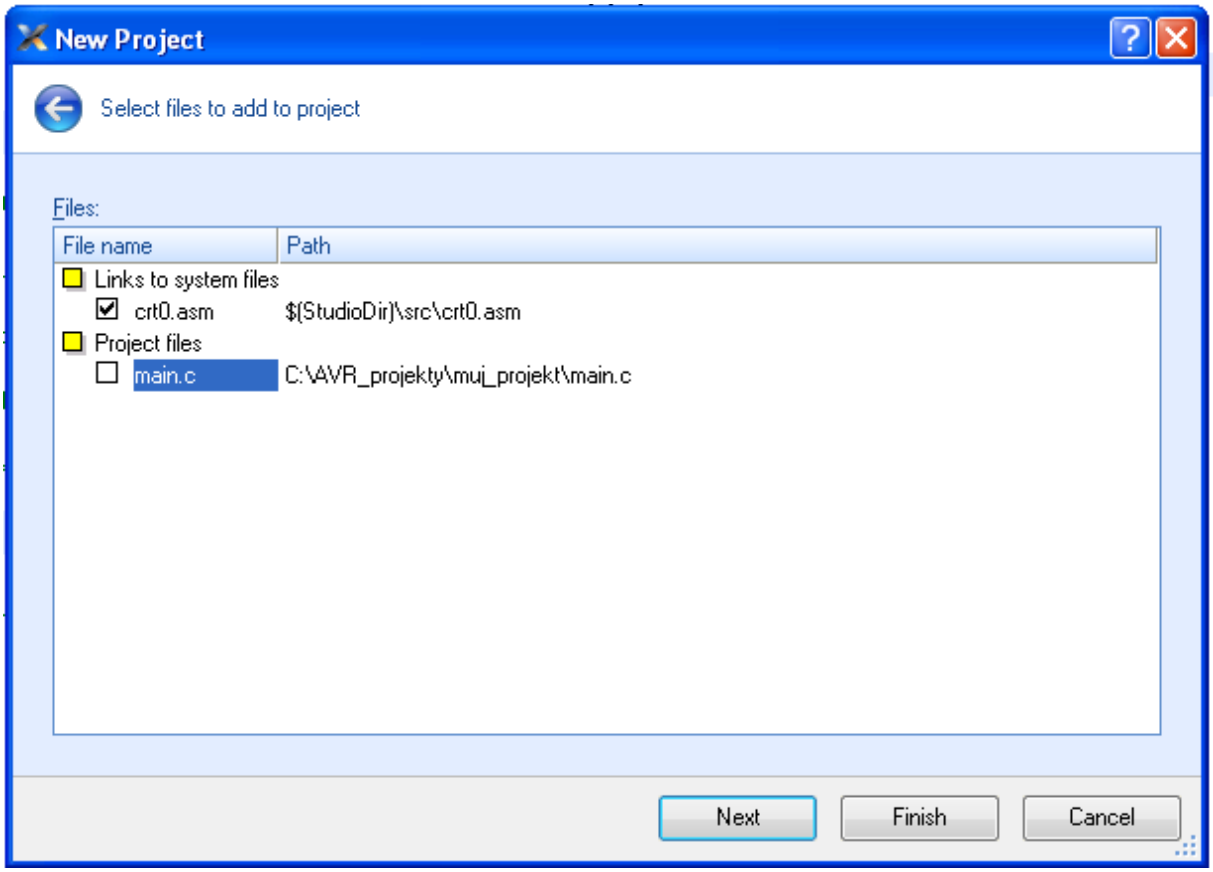
Zvolte umístění projektu v poli **Location** a jméno projektu v poli **Name**, pokračujeme **Next**



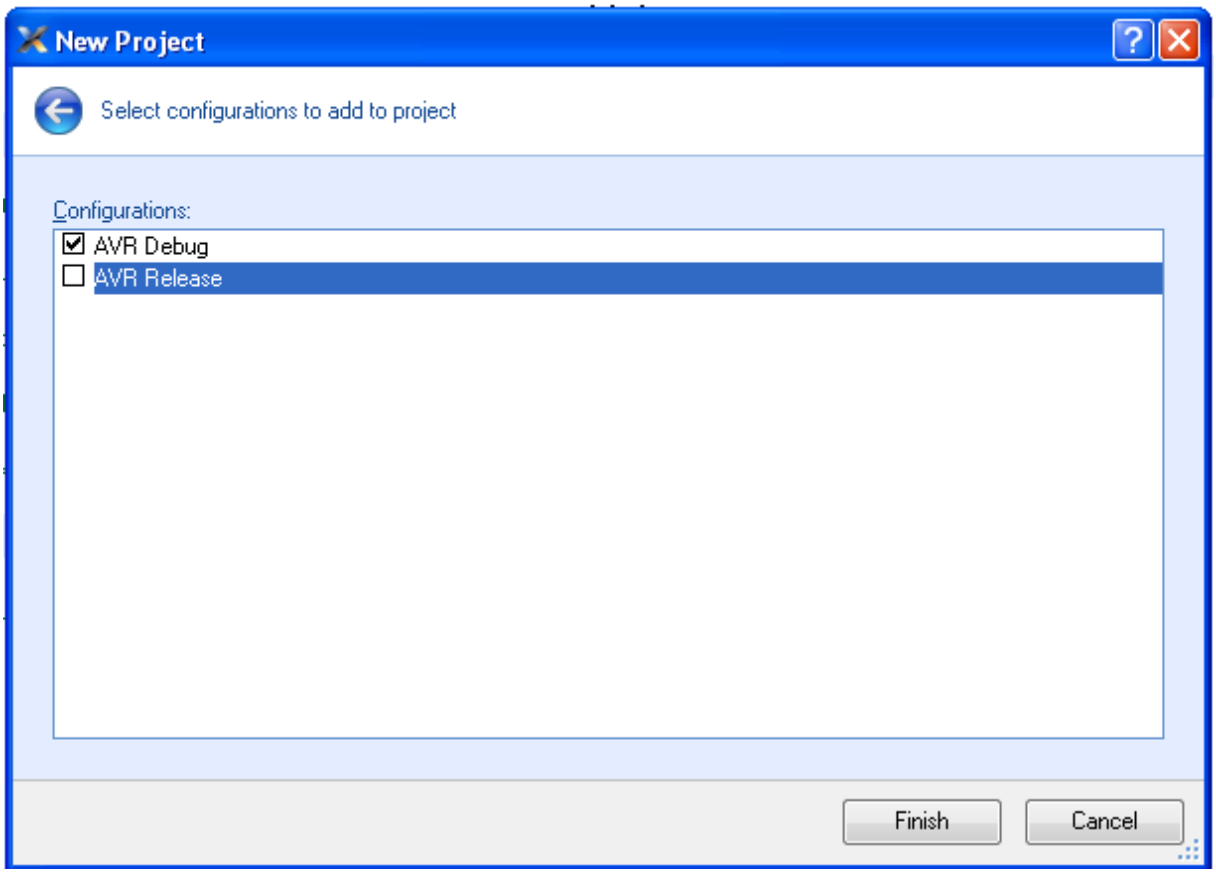
V okně **New Project** zvolte typ procesoru v řádku **Target Processor,** v našem příkladu typ: **ATmega32A** , pokračujte **Next**



V okně **New Project** odstraňte volbu vytvoření souboru **main.c** v zaškrtávacím poli Project files. V opačném případě se založí krátký demonstrační program se jménem main.c, který bychom stejně přepsali a název přejmenovali podle našich požadavků.



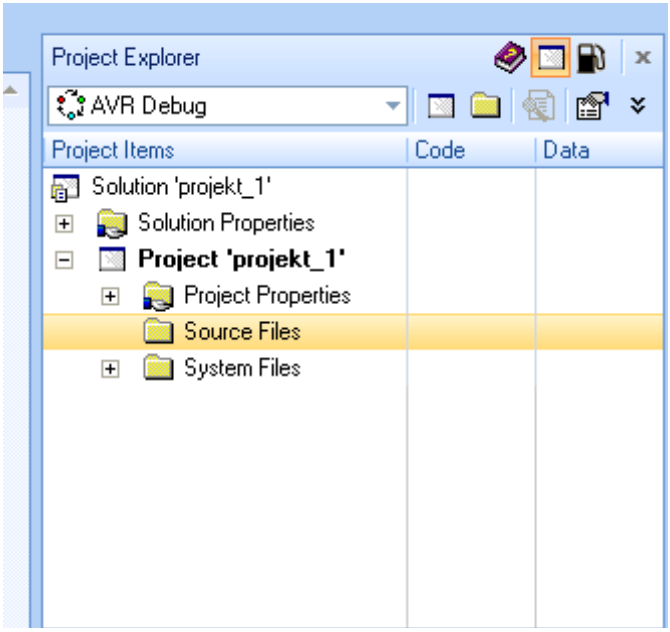
V okně **New Project** odstraníme volbu **AVR Release** v poli **Configurations.** Volba **Release** umožňuje vytvořit optimalizovaný kód, např. na minimální velikost nebo na rychlost chodu programu, což pro naše účely není podstatné.

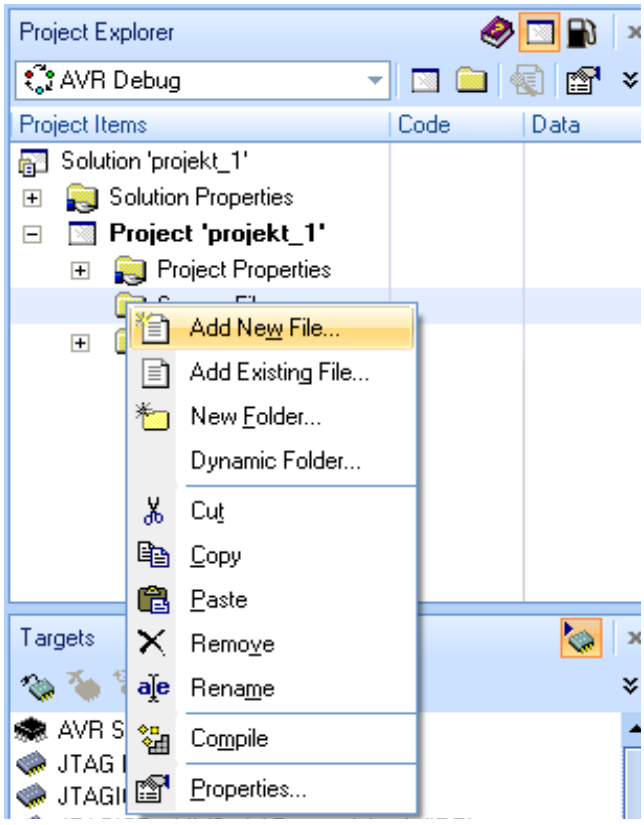


Po volbě **Finish** se otevře nové okno **Project Explorer,** ve kterém se zobrazí název „**Solution**“ se jménem, v našem příkladu „**projekt_1**“. Stejné jméno má i nově zavedený projekt, toto jméno lze však změnit poklepáním pravého tlačítka myši a výběrem položky **Rename**.

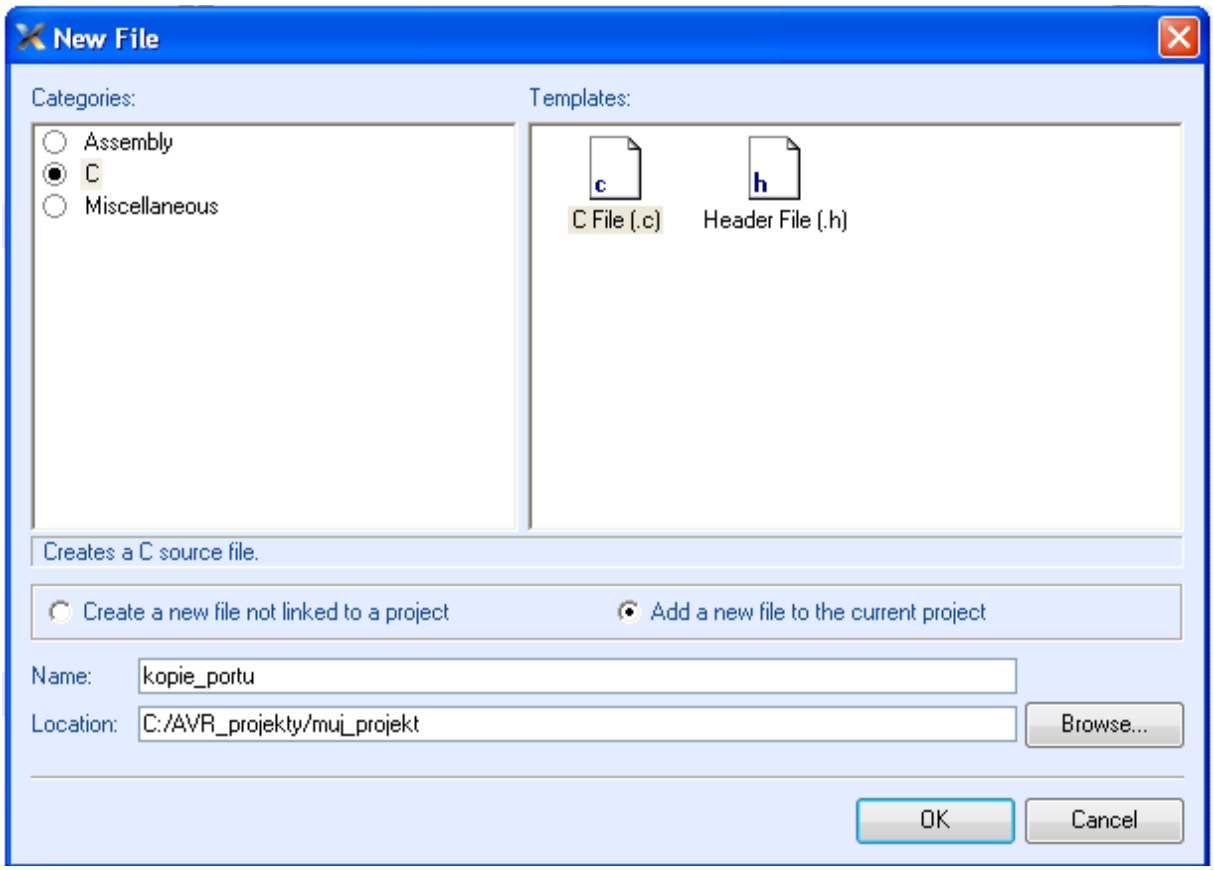
V Solution „projekt_1“ je možné založit další nové projekty, např. tématicky podobné. V nově otevřeném projektu není zatím žádný „zdrojový“ soubor, námi vytvořený program.

Okno pro editaci programu vytvoříme poklepáním pravým tlačítkem myši na položku **Source Files** a zvolením položky **Add New File**.





Následně se otevře okno New File.



Zvolíme volbu **C** v poli **Categories** a **C File(.c)** v poli **Templates**. Zvolíme jméno, např. „**kopie_portu**“ pro náš **.C** soubor, pro námi vytvářený program. Dále zvolíme **Add a new file to the current project** a potom **OK**. Otevře se editovací okno a můžeme začít psát program.

Pozn.: Jeden projekt může obsahovat (u větších projektů tomu tak běžně je) více souborů typu *.c. Tyto soubory obsluhují např. jednotlivé periferie systému (spínače, zobrazení na LCD, sériový kanál, paměť EEPROM,...). Řešení jednotlivých částí oddělenými programovými moduly zvyšuje přehlednost celého systému, umožňuje týmovou spolupráci, využití již dříve vyřešených úloh a má i jiné výhody**.** Tento způsob (více .c souborů) budeme používat i v některých našich příkladech. **Pouze jeden modul *.c však může obsahovat funkci main.c!**

6.2 Vytvoření nového projektu v existující „solution“

Solution může obsahovat více projektů. Nový projekt v existující solution založíme následujícím postupem:

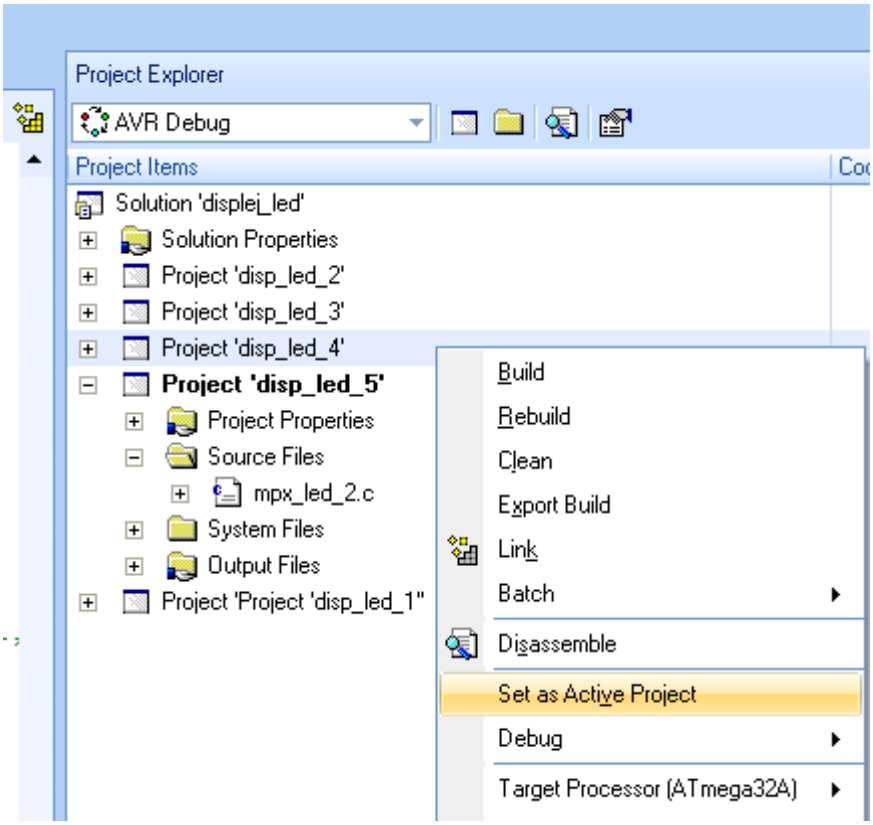
- V menu **File** vybereme položku **New Project**, zvolíme **Add the project to the current solution.**
- V okně **New Project** a podokně **Categories** vybereme volbu **Executable**, v podokně **Project Templates** položku **A C executable**. V poli **Name** zadáme jméno nového projektu, např. **projekt_2** a zadáme **Next**.
- V následujících oknech zadáme známým postupem typ procesoru **ATmega32A**, odstraníme předvolený soubor **main.c** a výběr configurace **release.**
- Zadáme jméno nového zdrojového souboru, např. „**tlacitka**“, po stisku **OK** jej můžeme začít editovat.

6.3 Editace programu

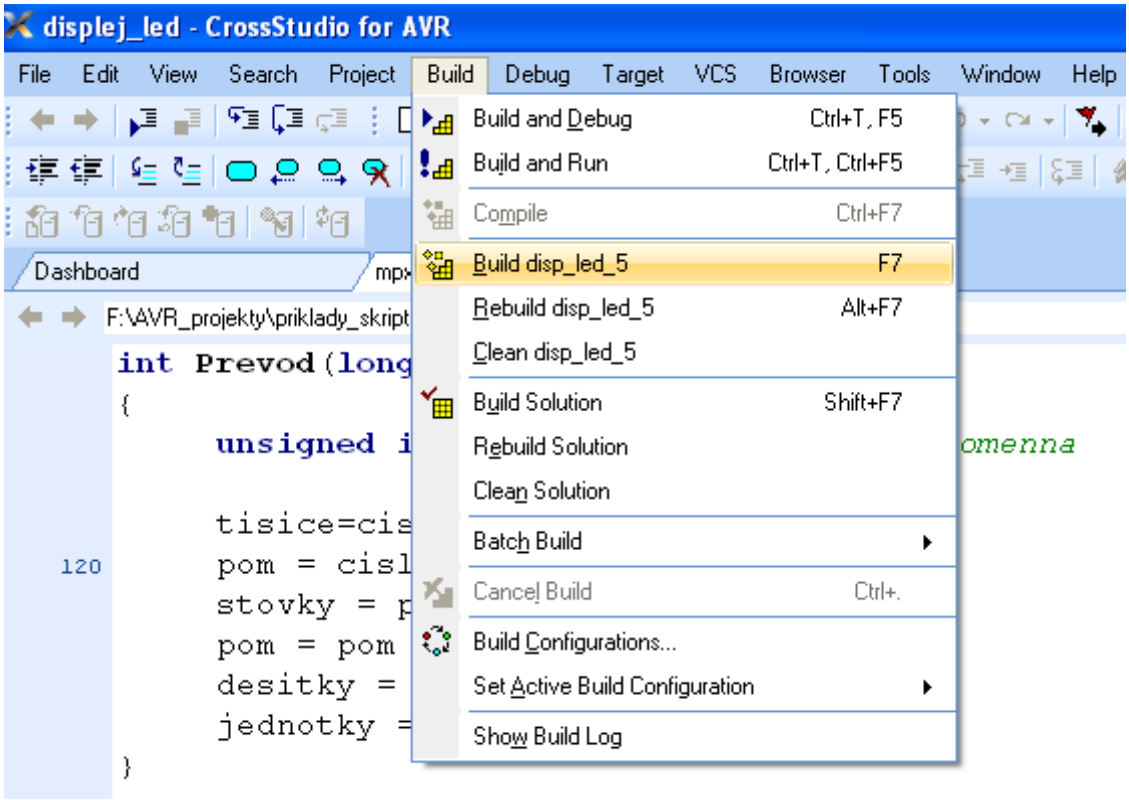
Editace využívá podobný editor, na který jsme zvyklí např. z balíků „office“ programů. Navíc má funkce vhodné pro psaní zdrojového programu v jazyce C. Jsou to např. tabulátory pro odsazení bloku, viditelné spojení párových závorek, nastavitelné barevné schema pro komentáře programu, definice a číslice, vyhledávání a záměna symbolů a mnoho dalších užitečných funkcí, které nám velmi ulehčují psaní zdrojových souborů. Zásady pro vhodný styl psaní programu jsou uvedeny na začátku kapitoly 7.

6.4 Sestavení programu

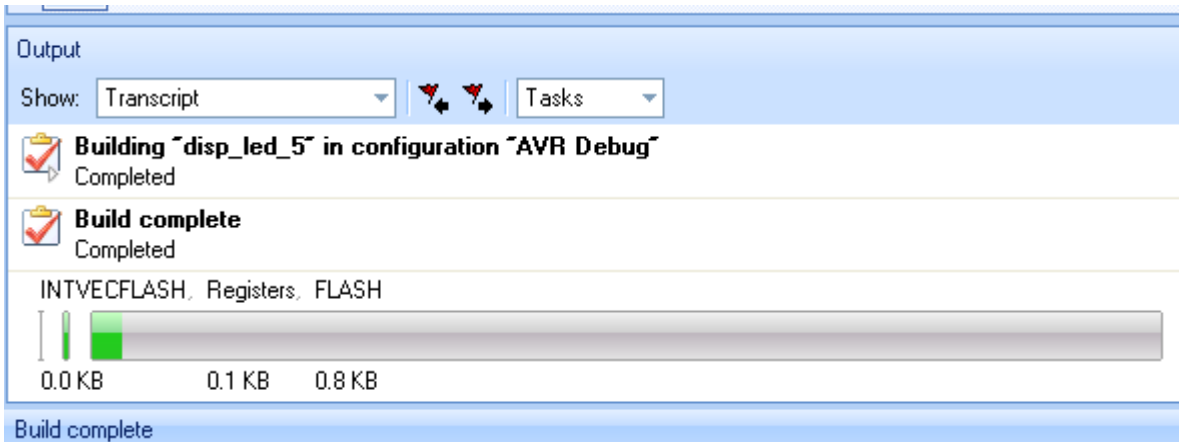
Po napsání zdrojového souboru ***.C** nastavíme daný projekt jako aktivní tak, že poklepeme 2x levým tlačítkem myši v okně **Project Explorer** na zvolený projekt, popř. poklepeme pravým tlačítkem 1x na zvolený projekt a vybereme volbu **Set as Active Project**. Aktivní projekt zůstane zvýrazněný tučným písmem. V našem případě na následujícím obrázku je aktivní projekt **disp_led_5**.



Na hlavní liště prostředí CrossStudia vybereme nabídku **Build** (sestavení), po rozbalení nabídky vybereme ikonu v řádku **Build** název projektu. Poté se provede překlad a sestavení našeho zdrojového programu.



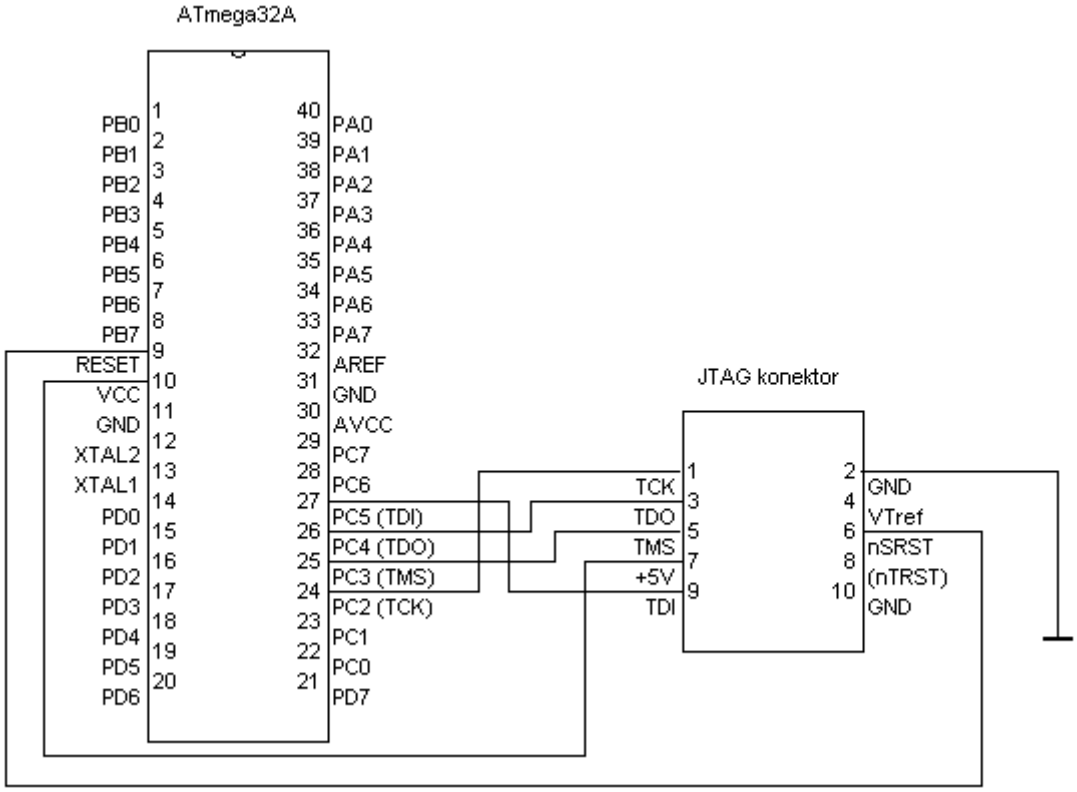
Ve spodní části obrazovky v okně Output se objeví hlášení o překladu. Pokud tato zpráva končí hlášením **Build complete**, vše proběhlo v pořádku a můžeme nahrát program do mikrokontroléru. V opačném případě se zobrazí seznam chybových hlášení, jedná se o formální chyby ve zdrojovém souboru, které je potřeba opravit. Většinou jsou to nepárové závorky, zapomenutý středník na konci příkazu, nedefinivaná proměnná či název funkce a podobně.



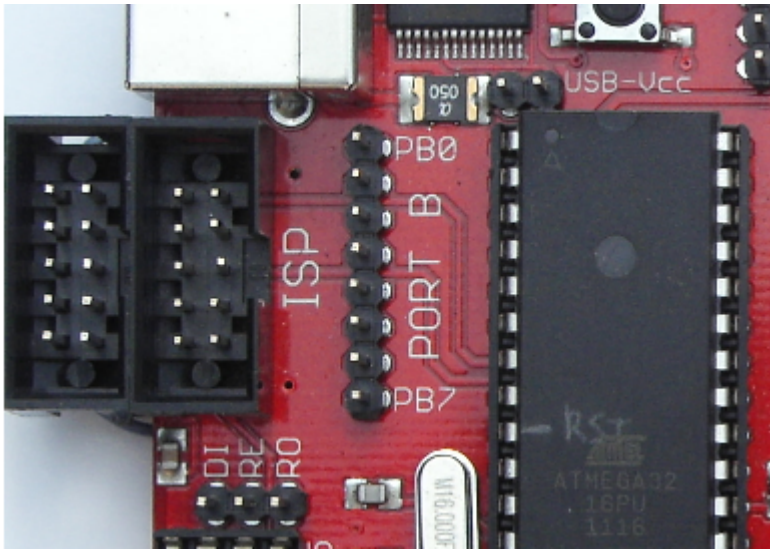
6.5 Vložení programu do mikrokontroléru a jeho spuštění

Máme-li již hotový a sestavený program, je třeba jej vložit do mikrokontroléru. Pro tento účel je možné použít několik možností, resp. programátorů. V našem systému použijeme programátor, který využívá rozhraní JTAG, konkrétně typ JTAG ICE (Joint Test Actoin Group In Circuit Emulator). Důvodem jeho použití je možnost odladění úlohy (debugging) přímo v mikrokontroléru. Popis ladění bude uveden v podkapitole 6.6.

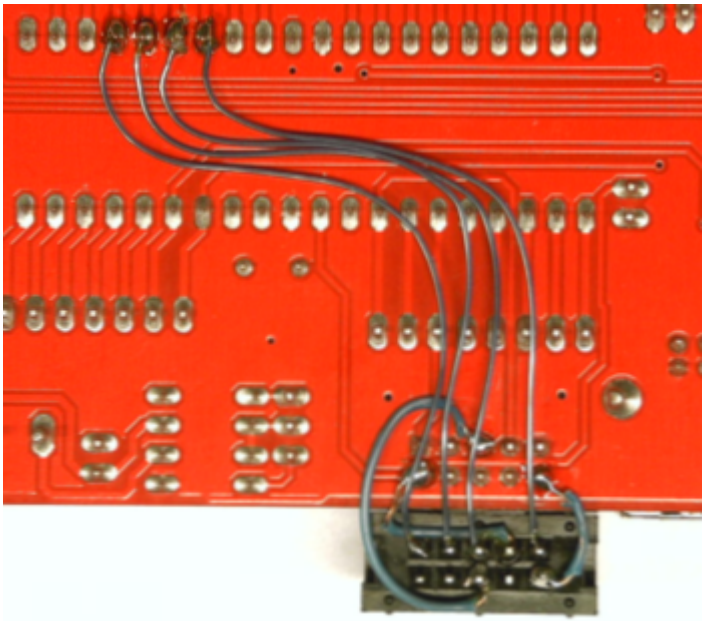
Vývojová deska EvB 4.3 neobsahuje rozhraní JTAG. Toto rozhraní je zde vytvořeno dodatečnou montáží - přilepením 10pólového konektoru na bok konektoru rozhraní ISP (In System Programing) vteřinovým lepidlem, viz. Obr.6. Propojení s patřičnými bity portu C mikrokontroléru znázorňuje schema na Obr. 5, realizaci drátového propojení ukazuje Obr. 7.



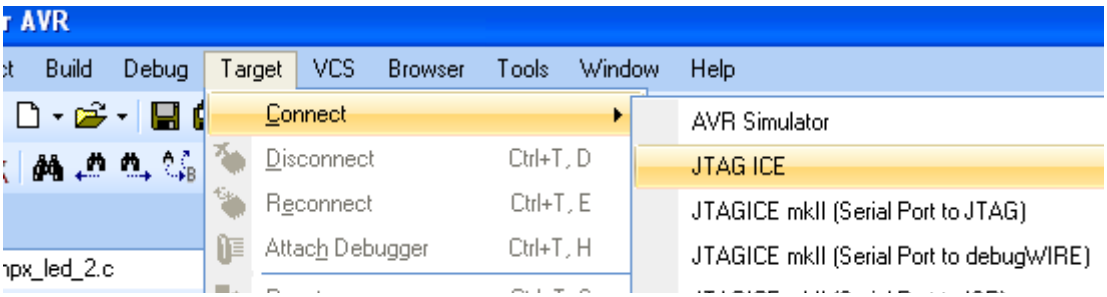
Obr.5: Připojení JTAG rozhraní k mikrokontroléru



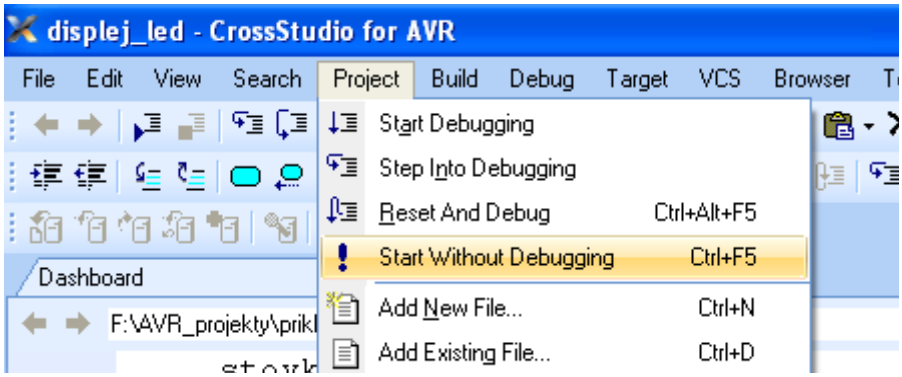
Obr. 6: Montáž konektoru pro JTAG rozhraní přilepením k ISP konektoru


Obr.7: Připojení JTAG rozhraní k mikrokontroléru ze strany spoje desky EvB 4.3

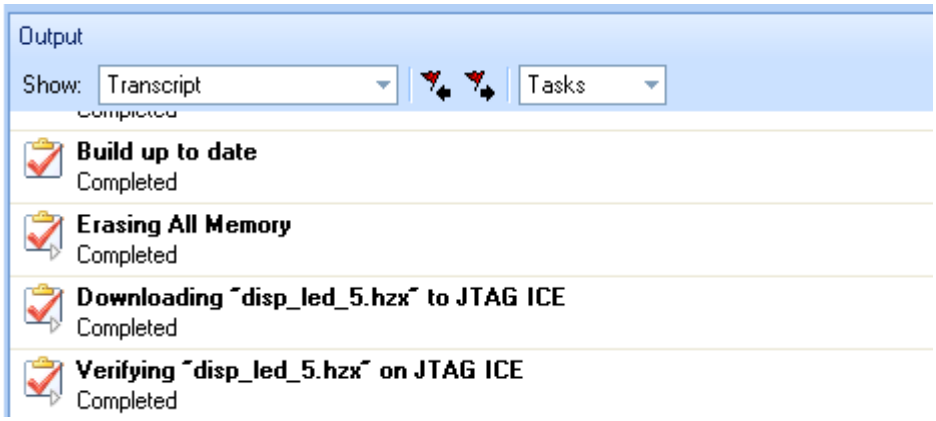
V CrossStudiu je potřeba zvolit použitý typ programátoru. Tento se vybere z hlavní lišty CrossStudia v okně **Targets.** V podokně **Connect** zvolíme položkou **JTAG ICE**. Programátor je připojen.



Vlastní nahrání programu do mikrokontroléru se provede z hlavní lišty CrossStudia v nabídce **Project** výběrem položky **Start Without Debugging**.



V okně **Output** se objeví hlášení o průběhu nahrání programu do mikrokontroléru.



Pokud je vše v pořádku, končí hlášení **Verifying** název programu **Completed** a program se v mikrokontroléru spustí.

6.6 Odladění úlohy

Po spuštění programu ve vývojové desce ověříme jeho činnost. Zvláště zpočátku, pokud nemáme příliš velké zkušenosti s programováním v C se stává, že program nepracuje buďto vůbec, popřípadě neprovádí přesně to, co jsme požadovali. V tomto případě použijeme tzv. **Debugger** CrossStudia, což je ladicí prostředí. Toto prostředí nám umožňuje testovat program po částech, popřípadě po jednotlivých příkazech. Debugger umožňuje umístit do programu tzv. Breakpointy (body zastavení), na nichž se činnost programu zastaví a máme možnost prohlédnout si např. obsahy portů, registrů, I/O registrů a jiné další možnosti.

Tento způsob ladění programu přímo v mikrokontroléru umožňuje právě rozhraní JTAG. Právě toto je důvod, proč byl zvolen typ mikrokontroléru s tímto rozhraním.

V následujícím textu budou stručně popsány základní příkazy debuggeru. Tyto příkazy je možné spustit po otevření okna **Debug** z hlavní lišty CrossStudia, pomocí ikon v liště nástrojů debuggeru nebo pomocí klávesových zkratek.

GO (F5) – spustí program v cílovém zařízeni (zde EvB 4.3) nebo pokračuje v jeho vykonávání po jeho přerušení až do bodu zastavení (Breakpoint)

Break (Ctrl +) - zastaví běh programu a vrací řízení debuggeru

Stop (Shift + F5) – zastaví debugging programu a vrací se do okna editoru

Restart (Ctrl + Shift + F5) – restartuje cílové zařízení a spustí debugging programu

Step Into (F11) – krokuje k dalšímu příkazu nebo instrukci a spouští C-funkce a podprogramy v assembleru. Pokud při krokování narazí na Breakpoint, debugger se v tomto bodě zastaví.

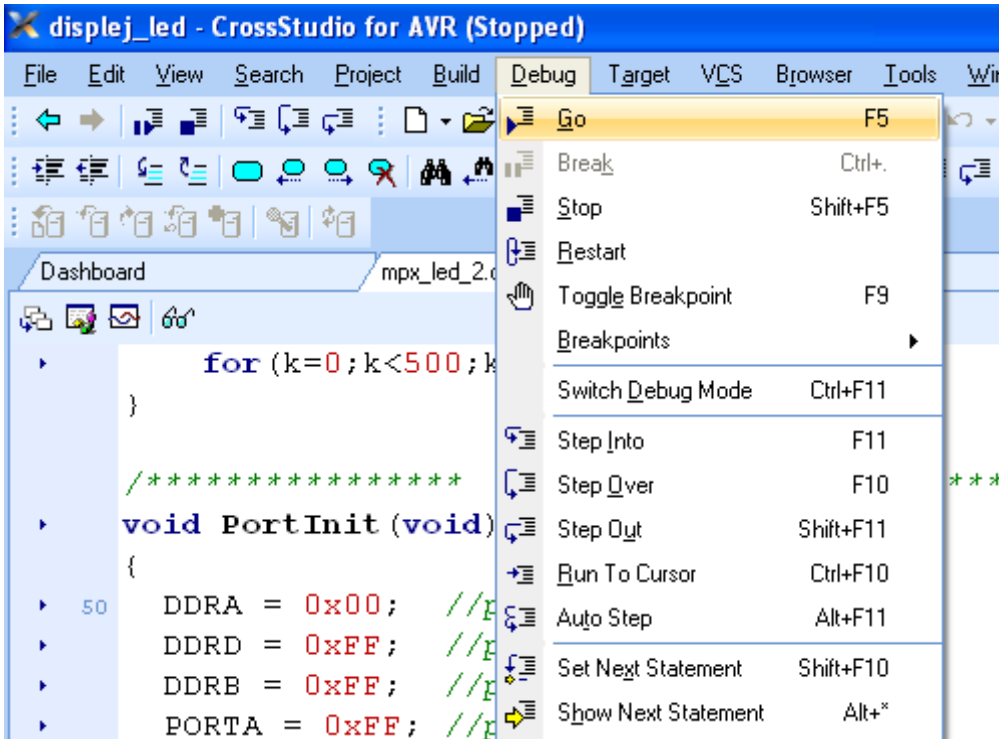
Step Over (F10) – přeskočí následující příkaz nebo instrukci aniž vstoupí do C funkce nebo do podprogramu v assembleru. Pokud při krokování narazí na Breakpoint, debugger se v tomto bodě zastaví.

Step Out (Shift + F11) – opustí C funkci nebo ASM podprogram a skočí až za instrukci, která volala tuto funkci nebo ASM podprogram. Pokud při krokování narazí na Breakpoint, debugger se v tomto bodě zastaví.

Run To Cursor (CTRL + F10) – spustí program na příkazu nebo instrukci, na které je kurzor. Pokud při krokování narazí na Breakpoint, debugger se v tomto bodě zastaví.

Auto Step (Alt + F11) – vykonávání programu včetně obnovování všech oken debugeru po vykonání každého příkazu nebo instrukce.

Set Next Statement (Shift + F10) – nastaví programový čítač na příkaz nebo instrukci, kde leží kurzor. Toto může vést k nepředvídanému nebo nesprávnému vykonávání programu.



Show Next Statement (Alt + *) - zobrazí zdrojovou řádku nebo instrukci asociovanou s programovým čítačem. Tento příkaz se používá, když hledáme ve výpisu programu řádek, který se bude zpracovávat.

Locals (Ctrl + Alt + L) – zobrazí obsah lokálních proměnných

Call Stack (Ctrl + Alt + S) – zobrazí seznam spuštěných funkcí v okamžiku zastavení programu

Pokud se běh programu v debug módu zastaví, ručně, při krokování nebo třeba na breakpointu (bod zastavení zpracování programu), zobrazí se v námi otevřených oknech obsahy registrů mikrokontroléru, lokálních či globálních proměnných, pamětí apod.. Hodnoty, které se změnily od předcházejícího zastavení programu se probarví červeně a jsou na první pohled viditelné. Tímto je možné podrobně testovat běh programu a odhalit v něm případné chyby.

Otevření oken se provede z hlavní lišty CrossStudia v menu **Debug** > **Debug Windows** > požadované okno.

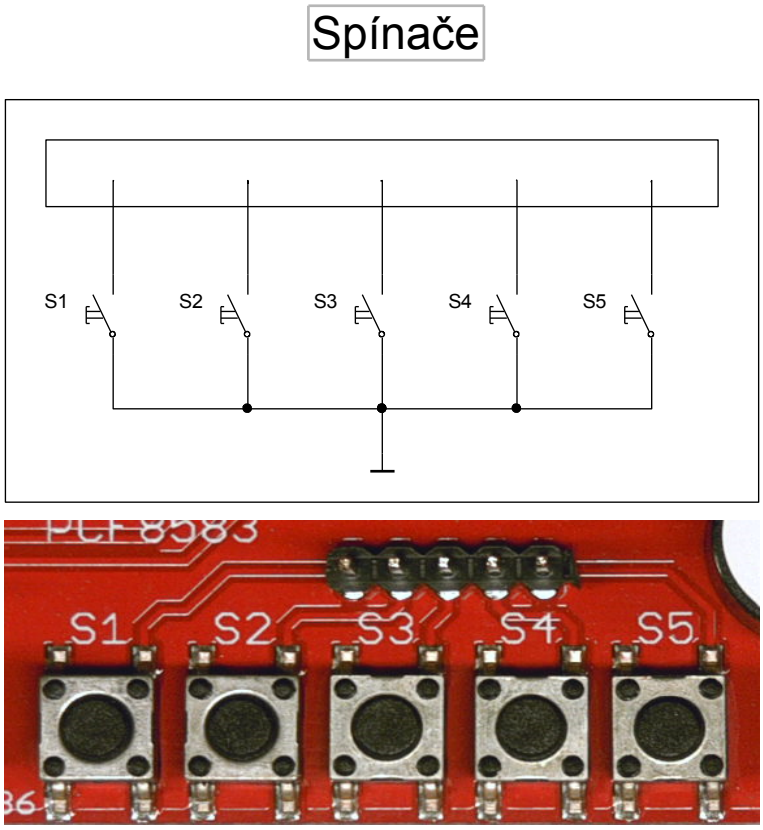
7 Úlohy

Těžištěm práce bude vytváření jednotlivých úloh, které budou používat různé periferie vývojové desky EvB 4.3. Úlohy budou koncipovány od nejjednodužších, kde se budou využívat hlavně tlačítka a indikátory LED při využití vstupně výstupních portů mikrokontroléru. Po získání jisté zkušenosti s programovacím jazykem C a práci ve vývojovém prostředí CrossWorks budou úlohy postupně nabývat na složitosti a budou využívat složitější periferie mikrokontroléru ATmega32 a vývojové desky EvB 4.3. Postup výuky bude takový, že se žáci nejprve seznámí s ukázkovými příklady, které budou po jejich dokonalém pochopení sami modifikovat, později pak budou vytvářet své vlastní projekty podle zadání.

Úlohy jsou rozděleny tématicky do jednotlivých kapitol, jejich seznam je uveden v obsahu těchto skript v kapitole 7.

Ve většině úloh budeme port A využívat právě pro čtení stavu tlačítek a port B k připojení indikátorů LED. Propojení je provedeno buďto jednopólovými kablíky, nebo (což je méně pracné a přehlednější) plochými 8pólovými kablíky.

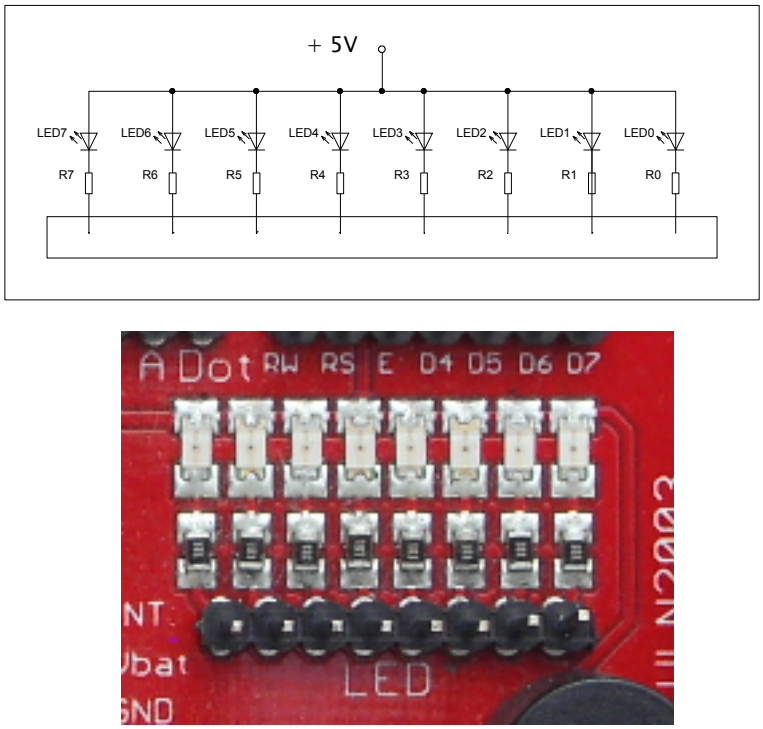
Pole tlačítek na vývojové desce EvB 4.3 je zapojeno podle schematu na Obr.8. Tlačítka jsou zapojena proti „zemi“, jejich stisknutí vytváří signál log. „0“. Signál log. „1“ je vytvořen připojením vnitřních pull-up rezistorů na vstupním portu A. Připojenípull-up rezistorů se provádí programově a bude vysvětleno při popisu programu.



Obr. 8: Zapojení modulu spínačů a jeho umístění na desce EvB 4.3

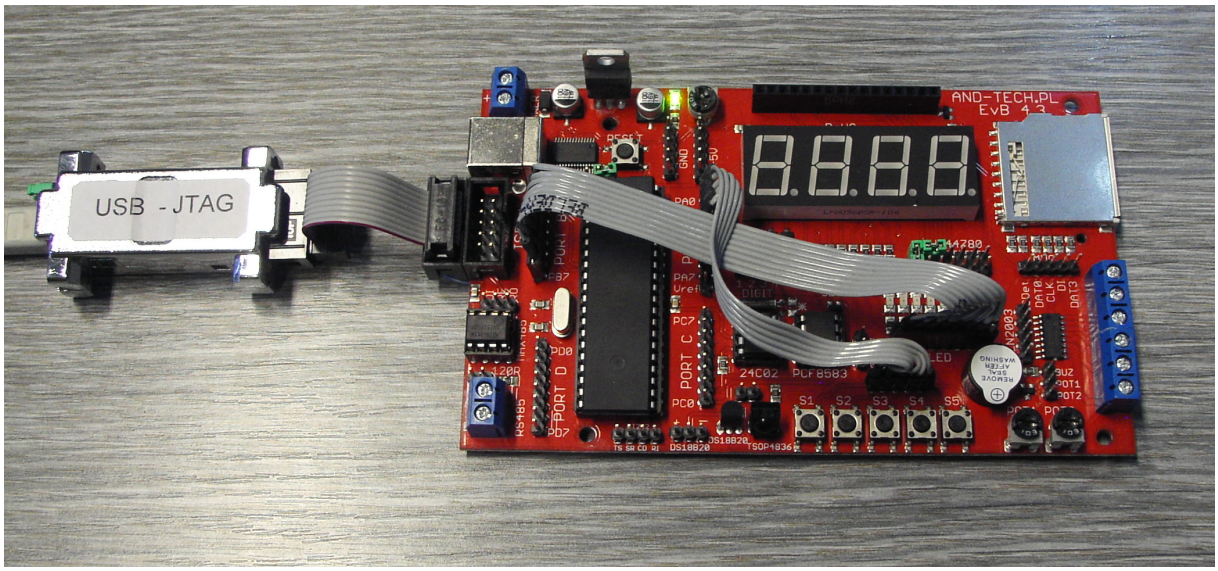
Pole indikátorů LED je zapojeno podle schematu na Obr.9. Indikátory LED jsou spojeny anodami na plus pól, katody jsou zapojeny přes omezovací rezistory na příslušné vývody portu. Indikátory jsou aktivní na signál log. „0“. Výstupní proud jednotlivých bitů portu je 40 mA, což umožňuje budit LED přímo z portu B bez nutnosti spínacích tranzistorů.

Indikátory LED



Obr.9: Zapojení modulu indikátorů LED a jeho umístění na desce EvB 4.3

Na Obr.10 je názorně vidět propojení mikrokontroléru s jednotlivými periferiemi.Toto propojení bude stejné pro všechny úlohy s porty mikrokontoléru a nebude již u ostatních úloh uváděno.



Obr.10: Propojení periferií s porty mikrokontroléru

Propojovací kablíky orientujeme tak, aby tlačítko označené S5 (umístěné na vývojové desce vpravo bylo připojeno na 0.bit portu A, stejně i LED umístěná vpravo (budeme ji nazývat LED0 podle vžitých zvyklostí z číslicové a mikroprocesorové techniky, kdy 0.bit binárního čísla je vždy vpravo) bude připojena na 0.bit portu B.

Na portu A zůstanou nepřipojeny bity 5 až 7.

Poznámky ke stylu psaní programu

Než začneme psát první program, je dobré si zvyknout na určitý styl psaní, na určitou úpravu. Přehledně napsaný program s patřičnými komentáři je dobře čitelný, snadněji se v něm odstraňují formální i logické chyby programu. Každý program by měl obsahovat tzv. „hlavičku“, která obsahuje název projektu, název .c souboru, stručný popis funkce programu (v našem případě to bude zadání úlohy), datum vytvoření programu a jméno autora. V následujícím textu uvedeme základní zásady úpravy a psaní programu, které jsou ověřeny letitou praxí zkušených programátorů, nesnažme se je proto měnit.

- vlastní program začíná vždy hlavičkou podobnou této:

```
/*****************************************************************************
*
*  Projekt:     Proj_4              název projektu
*  Soubor:      prikl_4.c           název souboru
*
*  Obsah:       Na displeji zobrazi hexacislici odpovidajici
*               hexa kodu zadanem  S5-S2,pri uvolneni tlac
*               sviti desetinna tecka,pri jine nez hexa
*               kombinaci nesviti nic
*
*  Vytvoreno: 25/10/2012
*
*****************************************************************************/
```

- jména objektů (proměnných, funkcí) psát zásadně malými písmeny s využitím podtržítek
- nepoužívat podobné identifikátory, např. **tlacit** a **tlacitka**
- nikdy nepoužívat dva stejné identifikátory rozlišené jen typem písma, např.: **CITAC** a **citac**
- běžně používané významové identifikátory:
  i j k -indexy, parametry cyklů
  c ch -znaky
  m n -čítače
  f r -reálná čísla
  s -řetězce
  p -ukazatele(pointery)
- před každou logicky ucelenou částí kódu uvádět komentář
- u kratších funkcí uvádět popis funkce před definicí funkce
- komentář má obsahovat pouze užitečnou informaci
- každá proměnná by měla být definována na samostatné řádce a okomentována
- mezi definicemi a příkazy je prázdná řádka
- každé vnoření logicky podřízeného úseku programu je o 2 mezery doprava
- prázdné řádky se vkládají podle vlastního uvážení kdekoliv, kde mohou zlepšit čitelnost programu:

```
if (x == 5)
  if (y == 6)
    z = 7;
                                /* prázdná řádka je zde velmi vhodná */
 if (k == 8)
  j = 9;
```

- levá složená závorka ´ { ´ začínající tělo funkce je vždy sama na řádce v 1. sloupci. Totéž platí pro uzavírací závorku funkce ´ } ´ .
- mezi: **if for while switch return** a následující otevírací závorkou ´ (´ musí být jedna mezera

```
while  (x == 5)
```

- binární operátory, např.: *** / + -**
  přiřazovací operátory, např.: **= += -= *=**
  operátory porovnání, např.: **== <= >**
  a ternární operátor: **?**

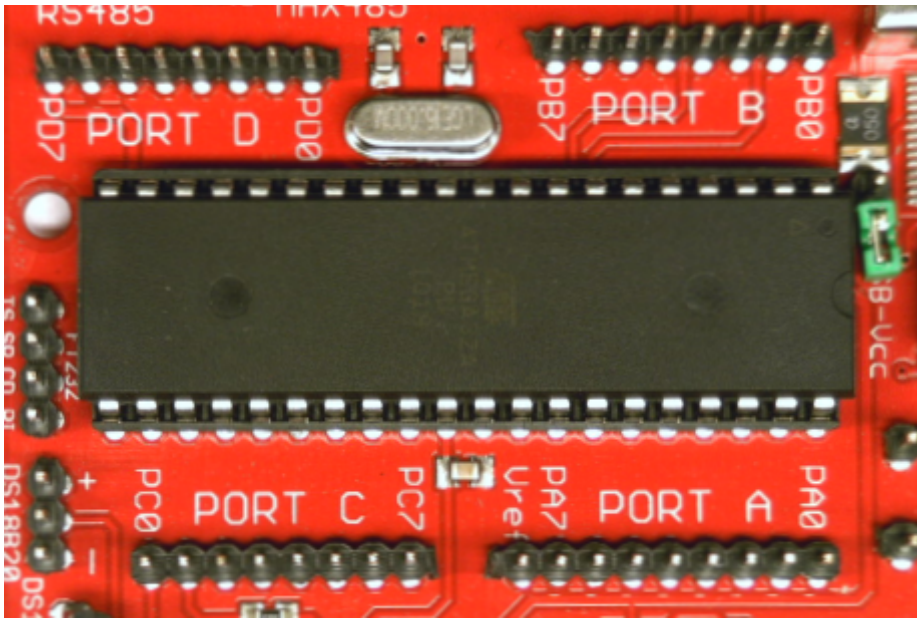
  musí mít jednu mezeru před a jednu mezeru za:

```
if (i  <=  5)
{
  j = k  *  8  +  3;
  k  +=  5;
}
```

- unární operátory, např.: **! ++ -- * &**
  nesmí být odděleny od opeandu mezerou:

```
i++;
j  +=  --i;
```

7.1 Obsluha I/O portů



I/O porty jsou používané k připojení nejrůznějších periferních obvodů. Mikrokontrolér ATmega32 obsahuje čtyři 8bitové porty, označené PORT A až PORT D.

7.1.1 Kopie portů

Tento příklad je velmi jednoduchý, je uveden zejména pro úplné začátečníky v práci s mikrokontroléry.

Zadání:
Program bude cyklicky číst stav portu A, ke kterému je připojeno 5 spinačů (S1 až S5). Stav spinačů se bude kopírovat na port B, ke kterému je připojeno 8 indikátorů LED.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       porty_1
*   Soubor:        kopie_portu.c
*
*   Obsah:         Kopie spinacu (port A) na ledky (port B)
*
*   Vytvoreno:   25/10/2012
*   Autor:
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA
#define LEDKY PORTB


/************  prototypy lokalnich funkci ****************************************/
void main(void);
void copy_portA_to_portB(void);

/************  kod  ***************************************************/

/*******  kopie portu A (tlacitka) do portu B (ledky)  ********************/
void copy_portA_to_portB(void)
{
   LEDKY = SPINACE;
}

/****************  main funkce ***************************************/
void main(void)
{
   DDRA = 0x00;    //port A je vstup
   DDRB = 0xff;     //port B je vystup
   PORTA = 0xff;     //pull-up pripojeny

   while(1)             //nekonecna smycka
   {
     copy_portA_to_portB();
   }
}
```

Popis programu:

Po úvodní hlavičce je příkazem **#include <atmega32.h>** vložen při překladu programu tzv. „hlavičkový soubor“, který obsahuje informace o použitém mikrokontroléru (ATmega32), např. makra pro definici adres vektorů přerušení (zatím nevíme, oč jde, přerušovacim systémem se budeme zabývat později, např. až budeme pracovat s interními časovači), makra pro definici bitů speciálních registrů, portů, atd.

Následuje definice konstant. V naší úloze je registr **PINA** (používá se pro čtení obsahu portu A) pojmenován výmluvným názvem **SPINACE** (jelikož k portu A jsou skutečně připojeny spinače) a registr **PORTB** pojmenován **LEDKY** (k portu B jsou skutečně připojeny indikátory LED). Tyto názvy můžeme volit libovolně, překladač pak při překladu programu nahradí náš text názvem příslušného registru portu.

Následuje seznam použitých lokálních funkcí. V našem případě je to funkce **main()**, což je „hlavní“ funkce, kterou obsahuje každý program a to právě pouze jednu funkci main(). Dále je to funkce **copy_port_A_to_port_B()**, kterou by bylo možné nahradit v main() funkci jedním příkazem **LEDKY = SPINACE.**. Použití funkce s pouze jedním příkazem není obvyklé, zde je použita pro ilustraci možností využití funkcí. Seskupením několika příkazů do funkce se zlepšuje přehlednost a čitelnost programu. Některé užitečné a často používané funkce je možné použít i v jiných programech. Můžeme si vytvořit knihovnu „svých“ funkcí, usnadní nám to řešení dalších úloh.

Následuje definice použitých funkcí s jejich popisem, zde je to pouze jedna funkce. Naše funkce je uvedena slovem **void** (prázdná, nevrací žádnou hodnotu) a za jménem funkce je též v závorce uvedeno **void**, jelikož se při volání funkce nepředávají žádné parametry.

Funkce **main()** začíná příkazy pro nastavení orientace portů a příkazem pro připojení pull-up rezistorů k portu A. Tyto rezistory vytvářejí úroveň log. „1“ při neaktivních (nesepnutých) spínačích, sepnuté spinače vytvářejí log. „0 “.

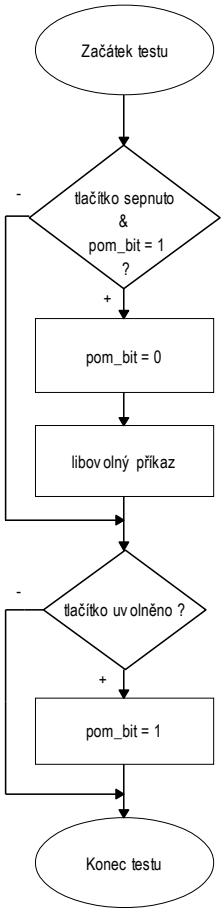
Smyčka **while(1)** je tzv. nekonečná smyčka, která zpracovává příkazy uzavřené do složených závorek, dokud je hodnota výrazu v závorce „TRUE“ (má libovolnou nenulovou hodnotu, což číslice 1 splňuje) . Tímto je dosaženo nepřetržité čtení stavu tlačítek a jeho kopie na indikátory LED. Tato smyčka se provádí několik set tisíckrát za sekundu, záleží na pracovním kmitočtu mikrokontroléru a na časové náročnosti zpracovávaných příkazů. V našem případě je jako zdroj „hodinového“ kmitočtu použit vnitřní kalibrovaný RC oscilátor o kmitočtu 1MHz (je to defaultní nastavení výrobce mikrokontroléru), při kterém je v našem jednoduchém příkladu kmitočet volání funkce kopie portů 142 kHz (změřeno osciloskopem).

7.1.2 Detekce náběžné hrany stisknutého tlačítka, odstranění zákmitů

Zadání:
Program bude číst stav spínačů na portu A, po stisku spínače S5 (0.bit portu A) se bude negovat stav Led0 (0.bit portu B), po stisku S4 (1.bit portu A) se bude negovat stav Led1 (1.bit portu B).

Pokud budeme číst periodicky velmi rychle stav tlačítka, při velké rychlosti čtení nastane i při velmi krátkém stisku tlačítka několikanásobné vykonání určité akce, v našem příkladu negace LED. V našem příkladu budeme požadovat při každém stisku pouze jednu negaci stavu LED.

Mechanismus detekce náběžné hrany stisku tlačítka využívá pomocného bitu, který se nuluje při detekci první hrany stisku tlačítka a nastavuje se až po jeho uvolnění. Teprve po uvolnění je možná detekce dalšího stisku a znovuprovedení příkazu. Činnost funkce je znázorněna vývojovým diagramem na Obr.11.



Obr.11: Vývojový diagram testu stisku tlačítka

Dalším problémem jsou zákmity mechanických kontaktů. Kontakt je připevněn na pružném pásku, který při stisku a uvolnění několikrát odskočí, čímž může způsobit několik sepnutí a rozepnutí. Jedním z možných řešení je několikanásobné čtení stavu sepnutého tlačítka po určitém časovém intervalu, pokud se stav tlačítka nemění, je považován za stabilní stav. Tento způsob bude použit i v následujícím příkladu.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:      porty_2
*
*   Soubor:       tlacitko.c
*
*   Obsah:        Test stisku spinacu, po stisku S5 neguj stav LED0,
*                  po stisku S4 neguj stav LED1
*                  Pouzijte funkci pro opakovane cteni stavu spinacu
*                  pro odstraneni jejich zakmitu
*
*   Vytvoreno:    2013
*
*****************************************************************************/

/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice globalnich promennych ******************************/
unsigned char SPIN_STAV;   //byty pro pomocne bitove promenne aktivity spinacu
unsigned char SPIN_ON;     //byty pro pomocne bitove promenne stavu ON spinacu

/************  definice konstant, jmen a maker ******************************************/
#define LEDKY PORTB
#define SPINACE PINA

#define OFF 0xff
#define ON 0x00

#define TRUE 1
#define FALSE 0

#define S5 0
#define S4 1
#define S3 2
#define S2 3
#define S1 4

#define Led0 0
#define Led1 1

#define SETBIT(BRANA, BIT) ((BRANA) |= (1<<(BIT)))      //nastaveni bitu
#define CLRBIT(BRANA, BIT) ((BRANA) &= ~(1<<(BIT)))  //nulovani bitu
#define NEGBIT(BRANA, BIT) ((BRANA) ^= (1<<(BIT)))    //negace bitu
#define TESTBIT(BRANA, BIT) ((BRANA) & (1<<(BIT)))    //test nahozeneho bitu
```

```
#define TESTNEGBIT(BRANA,BIT) (~BRANA & (1<<BIT))   //test nuloveho bitu


/************  prototypy lokalnich funkci *********************************/
void main(void);
void Delay(void);
unsigned char RepeatReadSwitch(unsigned char);

/************  kod  ****************************************************/

/************   casove zpozdeni  ***************************************/
void Delay(void)
{
  unsigned int i;
  for (i = 0;i < 1500;i++);           //asi 10ms
}

/************  opakovane cteni aktivniho stavu tlacitka ***************/

/*funkce filtruje zakmity spinacu,vraci hodnotu TRUE (1) pri opakovanem
potvrzeni aktivniho stavu spinace (Switch) po 5nasobnem cteni po 10ms,
v opacnem pripade vraci hodnotu FALSE (0)
*/

unsigned char RepeatReadSwitch(unsigned char Switch)
{
  unsigned char j;

  SETBIT(SPIN_ON,Switch);   //prvotni nahozeni priznaku sepnuti
  for (j = 0; j < 5; j++)   //opakuj cteni 5x po urcitych intervalech
  {
    Delay();               //cekej na nasledujici test spinace

    if (!(TESTNEGBIT(SPINACE,Switch) && TESTBIT(SPIN_ON,Switch)))
    {
      CLRBIT(SPIN_ON,Switch);  //pri neaktivnim spinaci nuluj priznak sepnuti
      return FALSE;                    //aktivita spinace nepotvrzena, funkce vraci log. 0
    }
  }
     return TRUE;           //pokud spinac 5x aktivni, funkce vraci log. 1
}


/****************  main funkce ************************************************/
void main(void)
{
    DDRA = 0x00;    //port A je vstup
    DDRB = 0xff;    //port B je vystup
    PORTA = 0xff;   //pull-up pripojeny

  LEDKY = OFF;

  while(1)           //nekonecna smycka
  {
     if (TESTNEGBIT(SPINACE,S5) && TESTBIT(SPIN_STAV,S5))  //test stisku S5
```

```
    {
      CLRBIT(SPIN_STAV,S5);

      if (RepeatReadSwitch(S5))   //pokud opakovane spinac sepnuty
      {
        //zde jakekoliv akce spustene stiskem S5
        NEGBIT(LEDKY,Led0);         //aby se neco delalo
      }
    }


    if (TESTNEGBIT(SPINACE,S4) && TESTBIT(SPIN_STAV,S4))  //test stisku S4
    {
      CLRBIT(SPIN_STAV,S4);

      if (RepeatReadSwitch(S4))    //pokud opakovane spinac sepnuty
      {
        //zde jakekoliv akce spustene stiskem S4
        NEGBIT(LEDKY,Led1);         //aby se neco delalo
      }
    }

    if (TESTBIT(SPINACE,S5)) SETBIT(SPIN_STAV,S5); //test uvolneni S5
    if (TESTBIT(SPINACE,S4)) SETBIT(SPIN_STAV,S4); //test uvolneni S4
  }
}
```

Popis programu:

V tomto programu používáme bitové proměnné. Mikrokontroléry AVR však nedisponují bitovými proměnnými. Pro práci s bity jsou použita makra, která provádí běžné bitové operace: nastavení bitu, nulování bitu, negaci bitu, test log. „1“ bitu a test log. „0“ bitu.

Tyto bity jsou uloženy v paměti RAM v globálních proměnných **SPIN_STAV** a **SPIN_ON**. Při testu sepnutí spínače **S5** využíváme 0. pomocný bit (#define S5 0) a při testu sepnutí spínače **S4** využíváme 1. pomocný bit (#define S4 1) bajtu **SPIN_STAV** jako příznak sepnutí či rozepnutí spínačů. Při sepnutí spínače se tento bit nuluje (makro **CLRBIT**), při uvolnění se nastavuje (makro **SETBIT**). Pro test sepnutého spínače se používá makro **TESTNEGBIT,** jelikož spínače jsou aktivní na log.“0“, pro test uvolnění makro **TESTBIT**. Obě tato makra testují port A pojmenovaný příkazem **#define** jménem **SPINACE**. Jednotlivé bity číslujeme pořadově, počínaje bitem 0, posledním bitem by byl bit 7, v tomto příkladu ale nejsou použity. V případě potřeby většího počtu bitů nadefinujeme další proměnné, např. v příkladu použitý bajt **SPIN_ON**.

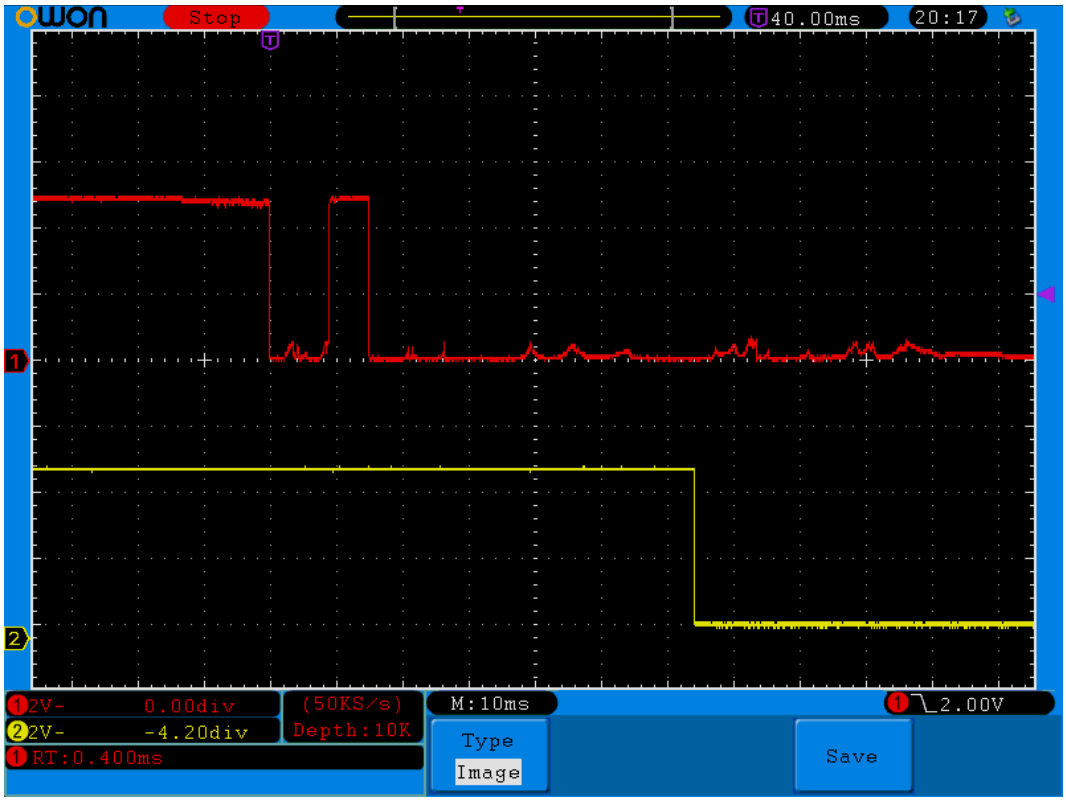
***Detekce hrany sepnutí spínače*:**

V hlavní smyčce programu se periodicky testuje stav spínačů S5 a S4. Při sepnutí spínače se nuluje odpovídající pomocný bit v proměnné SPIN_STAV, čímž se znemožní opakovaný test stisku spínače a následné provedení určité akce několikrát při jednom stisku spínače. Teprve po uvolnění spínače se pomocný bit nastaví a je možné akci provézt znovu.

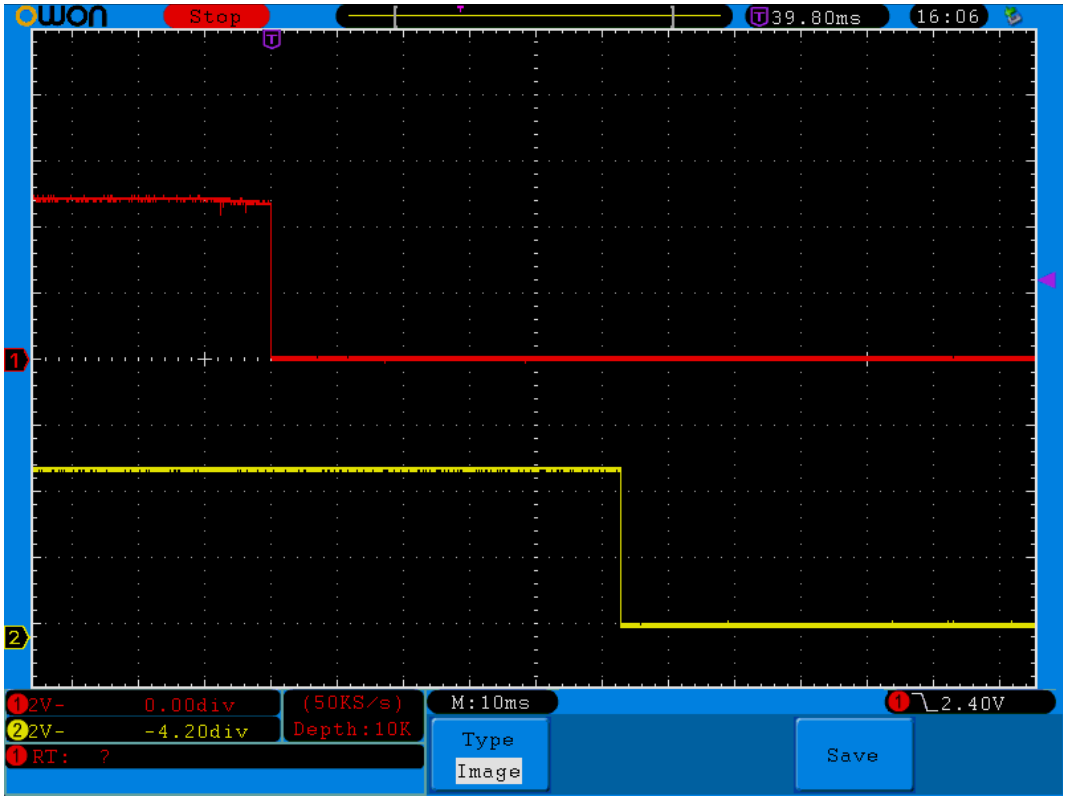
***Potlačení zákmitů spínače*:**

Po detekci sestupné hrany při stisku spínače se volá funkce **RepeatReadSwitch(),** která 5x opakovaně čte stav spínače po 10ms. Pokud je stav spínače stále log.0, funkce vrací hodnotu TRUE a požadovaná akce se provede. Při výskytu zákmitů spínače nebo při neurčitých log. úrovních proběhne test se záporným výsledkem, funkce vrací hodnotu FALSE a požadovaná akce se neprovede.

Na následujících obrázcích je záznam z paměťového osciloskopu, který demonstruje chování spínače a odstranění nežádoucích zákmitů. Kanál 1 záznamu (červený záznam) je výstupní signál spínače, kanál 2 (žlutý záznam) je výstup obvodu po filtraci zákmitů. Obvod synchronizace osciloskopu spouští záznam při sestupné hraně signálu o úrovni 2V a na obrázku je tento okamžik vyznačen písmenem **T** v horní části displeje. Od tohoto okamžiku, čili od sepnutí spínače, je spuštěn záznam průběhu.



Obr. 12 Zákmity spínače a jejich filtrace



Obr. 13 Sepnutí spínače bez zákmitů

7.1.3 Test tlačítek, následné akce

Zadání:
Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

1. S1-negace stavu LED
2. S2-LED zhasnou
3. S3-sviti vsechny LED
4. S4-LED kombinace 11001100
5. S5-LED kombinace 01010101

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       porty_3
*   Soubor:        tlacitka_akce.c
*
*   Obsah:         test tlacitek, nasledne akce na portu s ledkami:
*                  S1-negace stavu led
*                  S2-led zhasnou
*                  S3-sviti vsechny led
*                  S4-kombinace 00111100
*                  S5-kombinace 10101010
*                  pouziti maker nastaveni,nulovani a test bitu
*
*   Vytvoreno:     26/10/2012
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA  //cte stav portu A
#define LEDKY PORTB   //na portB pripojeny LED
#define S5  0
#define S4  1
#define S3  2
#define S2  3
#define S1  4
#define bS5  0
#define bS4  1
#define bS3  2
#define bS2  3
#define bS1  4
#define OFF 0xff
#define ON 0x00
//makra
#define  SETBIT(BRANA,BIT)   (BRANA |=(1<<BIT))          //nahozeni bitu
#define  CLRBIT(BRANA,BIT) (BRANA &= ~(1<<BIT))          //nulovani bitu
#define  NEGBIT(BRANA, BIT)  (BRANA ^= (1<<BIT))         //negace bitu
#define  TESTBIT(BRANA,BIT) (BRANA & (1<<BIT))           //test nahozeni bitu
```

```
#define  TESTNEGBIT(BRANA,BIT) (~BRANA & (1<<BIT))      //test nuloveho bitu

/************  definice globalnich promennych *************************/
unsigned char BAJT1;   //pole 8 bitu

/************  prototypy lokalnich funkci *********************************/
void main(void);
void PortsInit(void);
void Delay(void);
void NegujLed(void);
void ZhasniLed(void);
void RozsvitLed(void);
void Komb1Led(void);
void Komb2Led(void);

/************  kod  ***************************************************/
/************  inicializace portu  ************************************/
void PortsInit(void)
{
  DDRA = 0x00;    //port A je vstup
  DDRB = 0xff;    //port B je vystup
  PORTA = 0xff;   //pull-up pripojeny
}
/************  negace portu B  ****************************************/
void NegujLed(void)
{
  LEDKY = ~LEDKY;   //rozsvit/zhasni
}
/************  zhasnuti led  ******************************************/
void ZhasniLed(void)
{
  LEDKY = OFF;   //zhasni led
}
/************  rozsvit led  *******************************************/
void RozsvitLed(void)
{
  LEDKY = ON;   //rozsvit led
}
/************  led kombinace1  ****************************************/
void Komb1Led(void)
{
  LEDKY = 0x3c;   //led kombinace 1
}
/************  led kombinace2  ****************************************/
void Komb2Led(void)
{
  LEDKY = 0xaa;   //led kombinace 1
}
/************   casove zpozdeni  **************************************/
void Delay(void)
{
  unsigned int i;
  for (i = 0;i < 100;i++);
}
/****************  main funkce ***************************************/
```

```
void main(void)
{
 PortsInit();
 LEDKY = 0x55;

 while(1)       //nekonecna smycka
 {
  if(TESTNEGBIT(SPINACE,S5) & TESTBIT(BAJT1,bS5))    //test stisku S5
  {
     CLRBIT(BAJT1,bS5);
     NegujLed();
     Delay();             //casove zpozdeni pro ustaleni stavu tlacitka
  }
  if(TESTNEGBIT(SPINACE,S4) & TESTBIT(BAJT1,bS4))    //test stisku S4
  {
     CLRBIT(BAJT1,bS4);
     ZhasniLed();
     Delay();
  }
  if(TESTNEGBIT(SPINACE,S3) & TESTBIT(BAJT1,bS3))    //test stisku S3
  {
     CLRBIT(BAJT1,bS3);
     RozsvitLed();
     Delay();
  }
  if(TESTNEGBIT(SPINACE,S2) & TESTBIT(BAJT1,bS2))    //test stisku S2
  {
     CLRBIT(BAJT1,bS2);
     Komb1Led();
     Delay();
  }
  if(TESTNEGBIT(SPINACE,S1) & TESTBIT(BAJT1,bS1))    //test stisku S1
  {
     CLBIT(BAJT1,bS1);
     Komb2Led();
     Delay();
  }
  if (SPINACE == OFF)                                 //test uvolneni tlacitek
  {
     SETBIT(BAJT1,bS5);
     SETBIT(BAJT1,bS4);
     SETBIT(BAJT1,bS3);
     SETBIT(BAJT1,bS2);
     SETBIT(BAJT1,bS1);
  }
 }
}
```

Popis programu:

Tento program je rozšířenou verzí předcházejícího programu. Testujeme všechna tlačítka (S1 až S5), stisk každého z nich vyvolá určitou funkci. Funkce jsou opět velmi jednoduché. Funkce **Delay()** je použita v bloku testu každého tlačítka, pak je volána pouze při stisku tlačítka a nezatěžuje zbytečně mikrokontrolér.

7.1.4 Rotace LED ovládaná tlačítky

Zadání:
Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

1. S5 - rotace LED o 1 bit doprava, po zhasnutí LED0 se rozsvítí LED7
2. S4 - rotace LED o 1 bit doleva, po zhasnutí LED7 se rozsvítí LED0

Ukázkový program:

```
 /*****************************************************************************
*   Projekt:        porty_4
*
*   Soubor:       rotace_l_p.c
*
*   Obsah:        test tlacitka S5, po stisku rotuje LEDKY doprava o 1 bit
*                 test tlacitka S4, po stisku rotuje LEDKY doleva o 1 bit
*
*   Vytvoreno:    4.12.2012
*
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA  //cte stav portu A
#define LEDKY PORTB   //na portB pripojeny LED
#define S5  0
#define S4  1
#define bS5  0
#define bS4  1
#define OFF 0xff
#define ON 0x00

//makra pro bitove operace
#define  SETBIT(BRANA,BIT)   (BRANA |=(1<<BIT))            //nahozeni bitu
#define  CLEARBIT(BRANA,BIT) (BRANA &= ~(1<<BIT))      //nulovani bitu
#define  NEGBIT(BRANA, BIT)  (BRANA ^= (1<<BIT))           //negace bitu
#define  TESTBIT(BRANA,BIT) (BRANA & (1<<BIT))             //test nahozeni bitu
#define  TESTNEGBIT(BRANA,BIT) (~BRANA & (1<<BIT))    //test nuloveho bitu

/************  definice globalnich promennych ************************/
unsigned char BAJT1;   //pole 8 bitu

/************  prototypy lokalnich funkci ********************************/
void main(void);
void Delay(void);
void PortsInit(void);

/************  kod  ******************************************************/
/************  inicializace portu  ***************************************/
void PortsInit(void)
```

```
{
  DDRA = 0x00;    //port A je vstup
  DDRB = 0xff;    //port B je vystup
  PORTA = 0xff;   //pull-up pripojeny
}
/************  rotace led doprava  ***************************************/
void RotujLedP(void)
{
  LEDKY = ((LEDKY >> 1) | 0x80);  //po kazde rotaci zleva doplnuj "1"
  if(LEDKY == 0xff) LEDKY = 0x7f; //po zhasnuti vsech LED rozsvit LED7
}
/************  rotace led doleva  ****************************************/
void RotujLedL(void)
{
  LEDKY = ((LEDKY << 1) | 0x01);  //po kazde rotaci zprava doplnuj "1"
  if(LEDKY == 0xff) LEDKY = 0xfe; //po zhasnuti vsech LED rozsvit LED0
}
/************   casove zpozdeni  *****************************************/
void Delay(void)
{
  unsigned int i;
  for (i = 0;i < 100;i++);
}
/****************  main funkce ***************************************/
void main(void)
{
  PortsInit();
  LEDKY = OFF;
  while(1)       //nekonecna smycka
  {
    if(TESTNEGBIT(SPINACE,S5) & TESTBIT(BAJT1,bS5))    //test stisku S5
    {
      CLEARBIT(BAJT1,bS5);
      RotujLedP();
      Delay();             //casove zpozdeni pro ustaleni stavu tlacitka
    }
    if(TESTNEGBIT(SPINACE,S4) & TESTBIT(BAJT1,bS4))    //test stisku S4
    {
      CLEARBIT(BAJT1,bS4);
      RotujLedL();
      Delay();
    }

    if (SPINACE == OFF)                 //test uvolneni tlacitek
    {
      SETBIT(BAJT1,bS5);
      SETBIT(BAJT1,bS4);
    }
  }
}
```

Popis programu:

Tento program je rozšířenou verzí předcházejícího programu. Pro posuv LED jsou použity operátory posunu bitů doleva **<<** a doprava **>>** o 1 bit.

Při posuvu doleva se doplňuje zprava na pozici LED0 log.1, aby se tato nerozsvítila, při posunu doprava se doplňuje zleva na pozici LED7 log.1. Toto se provede operátorem logického součtu **„ | ”**s bytem 0x01 při posuvu doleva, resp. 0X80 při posuvu doprava.

7.1.5 Rotace LED ovládaná časovačem (časová smyčka)

Zadání:
Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

S5 - rotace LED o 1 bit doprava po 0,5 s dokola
S4 - zastavení rotace, rozsvícení LED3
S3 - rotace LED o 1 bit doleva po 0,5 s dokola

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:        porty_5
*   Soubor:         rotace_l_p_stop.c
*
*   Obsah:         -vychozi stav:po zapnuti sviti LED3
*                  -po stisku S5 postupne blikaji LED doprava po 0,5s
*                   dokola
*                  -po stisku S4 se pohyb zastaví a sviti vychozi stav
*                  -po stisku S3 postupne blikaji LED doleva po 0,5s
*                   dokola
*
*   Vytvoreno:   20.12.2012
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA  //cte stav portu A
#define LEDKY PORTB   //na portB pripojeny LED
#define S5  0
#define S4  1
#define S3  2
#define bDoleva 0
#define bDoprava 1
#define bTimeOut 2
#define OFF 0xff
#define ON 0x00
#define vychozi_stav 0xf7
//makra
#define  SETBIT(BRANA,BIT)   (BRANA |=(1<<BIT))          //nahozeni bitu
#define  CLRBIT(BRANA,BIT) (BRANA &= ~(1<<BIT))          //nulovani bitu
```

```
#define  NEGBIT(BRANA, BIT)  (BRANA ^= (1<<BIT))          //negace bitu
#define  TESTBIT(BRANA,BIT) (BRANA & (1<<BIT))            //test nahozeni bitu
#define  TESTNEGBIT(BRANA,BIT) (~BRANA & (1<<BIT))     //test nuloveho bitu

/************  definice globalnich promennych *****************************/
unsigned char BAJT1;   //pole 8 bitu

/************  prototypy lokalnich funkci *********************************/
void main(void);
void Delay(void);
void PortsInit(void);
void RotujLedP(void);
void RotujLedP(void);
void TestTlac(void);

/************  kod  ***************************************************/
/************  inicializace portu  ************************************/
void PortsInit(void)
{
  DDRA = 0x00;    //port A je vstup
  DDRB = 0xff;    //port B je vystup
  PORTA = 0xff;   //pull-up pripojeny
}
/************  rotace led doprava  ************************************/
void RotujLedP(void)
{
  LEDKY = ((LEDKY >> 1) | 0x80);  //po kazde rotaci zleva doplnuj "1"
  if(LEDKY == 0xff) LEDKY = 0x7f; //po zhasnuti vsech LED rozsvit LED7
}
/************  rotace led doleva  ************************************/
void RotujLedL(void)
{
  LEDKY = ((LEDKY << 1) | 0x01);  //po kazde rotaci zprava doplnuj "1"
  if(LEDKY == 0xff) LEDKY = 0xfe; //po zhasnuti vsech LED rozsvit LED0
}
/************   casove zpozdeni  ************************************/
void Delay(void)
{
  unsigned int i;
  for (i = 0;i < 100;i++);
}

/************  kod  ***************************************************/
//***********  funkce testuje tlacitka a nahazuje smerove bity  **********/
void TestTlac(void)
{
  if(TESTNEGBIT(SPINACE,S5))
  {
    CLRBIT(BAJT1,bDoleva);
    SETBIT(BAJT1,bDoprava);
  }
  if(TESTNEGBIT(SPINACE,S3))
  {
    CLRBIT(BAJT1,bDoprava);
    SETBIT(BAJT1,bDoleva);
```

```
  }
  if(TESTNEGBIT(SPINACE,S4))
  {
    CLRBIT(BAJT1,bDoprava);
    CLRBIT(BAJT1,bDoleva);
  }
}
//************  neprava casova zakladna  *********************
// po 6500 cyklech nahazuje bit b_timeout, je nutne jej nulovat
// po detekci volajici funkci pro dalsi casovani

void TimeBase(void)
{
  static unsigned int i;
  i++;
  if(i == 16500)  //konstanta pro priblizny cas 0,5 sec
  {
    i = 0;
    SETBIT(BAJT1,bTimeOut);
  }
}

/****************  main funkce ****************************************/
void main(void)
{
  PortsInit();

  while(1)
  {
    TestTlac();
    TimeBase();

    if(TESTBIT(BAJT1,bDoprava) && TESTBIT(BAJT1,bTimeOut))
    {
      CLRBIT(BAJT1,bTimeOut);
      LEDKY = ((LEDKY >> 1)|0x80);
      if (LEDKY == 0xff) LEDKY = 0x7f;
    }

    if (TESTBIT(BAJT1,bDoleva) && TESTBIT(BAJT1,bTimeOut))
    {
      CLRBIT(BAJT1,bTimeOut);
      LEDKY = ((LEDKY << 1)|0x01);
      if(LEDKY == 0xff) LEDKY = 0xfe;
    }

    if (TESTNEGBIT(BAJT1,bDoprava) && TESTNEGBIT(BAJT1,bDoleva))  //nic neni
                                                                  //stisknuto
    {
      LEDKY = vychozi_stav;   //sviti led3
    }
  }
}
```

Popis programu:

V programu je použita funkce **TestTlac()**, která nahazuje a nuluje směrové bity **bDoprava** a **bDoleva** podle stisknutého tlačítka **S5**, **S4** nebo **S3**. Tato funkce se periodicky volá v hlavní smyčce programu, po jejím vykonání se směrové bity testují. Je- li nahozen některý ze směrových bitů a při současném dosažení času přibližně 0,5 s (testuje se bit **bTimeOut**, nastavovaný ve funkci **TimeBase()**) se provede vynulování bitu bTimeOut (pro nový odpočet času) a následně rotace brány A (LEDKY) o 1 bit na příslušnou stranu. Po rotaci se nahazuje vždy vstupní bit brány ve směru rotace, jelikož operátor rotace dosazuje na uvolněný bit log.0 a patřičná krajní LED by svítila a dál rotovala (LED jsou aktivní na „0“). Funkce **TimeBase()** se volá periodicky v hlavní smyčce programu, po spuštění se inkrementuje proměnná **„i“** a testuje se dosažení hodnoty **TimeOver**, která určuje čas přibližně 0,5 s a musí se experimentálně nastavit. Proměnná „i“ je typu static, pak si uchová svou hodnotu i po opuštění a znovuvolání funkce TimeBase().

Řešení časování pomocí funkce TimeBase() je výhodnější, než použití funkce pro časové zpoždění realizované časovou smyčkou. V tomto případě by byl mikrokontrolér po dobu zpracování smyčky zcela zaneprázdněn a nemohl by vykonávat jinou činnost. Podstatně lepší by ale bylo použití vnitřního čítače/časovače mikrokontroléru, který čítá kmitočet systémových hodin zcela autonomně a nezatěžuje mikrokontrolér. Použití čítačů/časovačů bude vysvětleno v pozdějších kapitolách skript.

7.1.6 Úlohy k samostatnému řešení

7.1.6.1: Rotace1

Program bude číst stav spínačů na portu A, při sepnutých spínačích bude vykonávat následující akce:

Po stisku S5 se spustí rotace LED počínaje LED0 doleva s periodou 1s (časování programovou smyčkou)

Rotování LED se zastaví po 5 cyklech nebo po stisku S4 a LED zhasnou.

7.1.6.2: Rotace2

Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

- po 3-násobném stisku S5 - rotace LED o 1 bit doprava dokud LED nezhasne
- po 3-násobném stisku S1 - rotace LED o 1 bit doleva dokud LED nezhasne

S2 – rozsviť LED0
S4 – rozsviť LED7
S3 – zhasnou rozsvícené LED

7.1.6.3: Blikač1

Program bude číst stav spínačů na portu A, při sepnutých spínačích bude vykonávat následující akce:

- Při současném stisku S5 a S4 rozblikej trvale LED0 s periodou 0,5s
- Při současném stisku S1 a S2 rozblikej trvale LED7 s periodou 0,5s
- Při stisku S3 nuluj rozsvícené LED

Použijte makra pro operaci s bitovými proměnnými.

7.1.6.4: Blikač2

Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

- S1 – blikání LED0 s periodou 2 s
- S2 – blikání LED0 s periodou 1 s
- S3 - blikání LED0 s periodou 0,5 s
- S4 – střídavé blikání LED0 / LED1 s periodou 0,5 s
- S5 – střídavé blikání LED0 / LED1 s periodou 1 s

7.1.6.5: Padající vločky

Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

S1 – Led se rozsvěcují periodicky dokola podle následujícího schematu:

00000001
00000011
00000111
00001110
00011100
00111000
01110000
11100000
11000000
10000000

S2 – Led se rozsvěcují periodicky dokola podle následujícího schematu:

10000000
11000000
11100000
01110000
00111000
00011100
00001110

00000111
00000011
00000001

S3 – Stop blikání

Pozn.1: 0 – Led nesvítí, 1 – Led svítí, perioda přepínání 0,2 s

Pozn.2: Vytvořte a použijte funkce **RotaceL()** a **RotaceP()**, příkaz cyklu a příkaz pro rotaci bitů

7.1.6.6: Dekodér binárního kódu na kód 1 z 8.

Program bude číst stav spínačů na portu A, spínače S3 až S5 budou vytvářet vstupní binární kód dekodéru (S3 = 2^2, S4 = 2^1, S5 = 2^0). Výstupem dekodéru budou LED na portu A (Led7 = výstup7, Led6 = výstup6, …, Led0 = výstup0).

Pozn.: Vytvořte 2 řešení:

1. Pro převod použijte příkaz **switch(…);**
2. Pro převod použijte funkci **BinToOut()**

7.1.6.7: Dekodér hexadecimálního kódu na kód Gray-úv

Program bude číst stav spínačů na portu A, spínače S2 až S5 budou vytvářet vstupní hexadecimální kód dekodéru (S2 = 2^3, S3 = 2^2, S4 = 2^1, S5 = 2^0), výstupem dekodéru budou LED na portu A (Led4 až Led0), které se budou rozsvěcovat podle Grayova kódu.

Pozn.: Vytvořte 2 řešení:

1. Pro převod použijte příkaz **switch(…);**
2. Pro převod použijte funkci **HexToGray()**

7.1.6.8: Lékárnická míchačka

Program bude číst stav spínačů na portu A

Při sepnutém spínači Start (S1) zapne chod lékárnické míchačky.

Míchačka má motor, který se otáčí 4s doprava (Led0), následuje 2s motor v klidu, poté 4s doleva (Led1), 2s klid a tento cyklus se opakuje až do stisku tlačítka Stop (S2).

Po stisku tlačítka Cisteni (S3) se reverzuje motor s periodou 1s po dobu 5s.

7.1.6.9: Schodišťový automat

Program bude číst stav spínačů na portu A a bude provádět následující akce:

- Při sepnutém spínači S1 bude svítit světlo (Led0) po dobu 5s

- Při dvojnásobném stisku v časovém intervalu 1s bude svítit světlo trvale
- Při stisku delším než 2s světlo zhasne.

7.1.6.10: Postupný rozběh motoru

Program bude číst stav spínačů na portu A a bude provádět následující akce:

- Při sepnutém spínači Start (S1) se zapne slykač K0 (Led0), dále vždy po 3s stykač K1 (Led1), K2 (Led2) a K3 (Led3)
- Po stisku tlačítka Stop (S2) se vypne ihned stykač K3, potom opět po 3s postupně stykače K2, K1 a K0.

7.1.6.11: Rozběh světelného bodu

Program bude číst stav spínačů na portu A a bude provádět následující akce:

- Po stisku spínače S1 se postupně rozsvěcí světelný bod doprava od Led7 do Led0 s rostoucí rychlostí, první časová prodleva mezi posuvem bude 0,7s, pak 0,6s, 0,5s, …0,2s, 0,1s
- Po stisku S5 to samé doleva, první prodleva 0,1s, 0,2s, ..., 0,7s.

Pozn.: Pro rotaci světelného bodu použijte operátor rotace a cykly for(; ;)

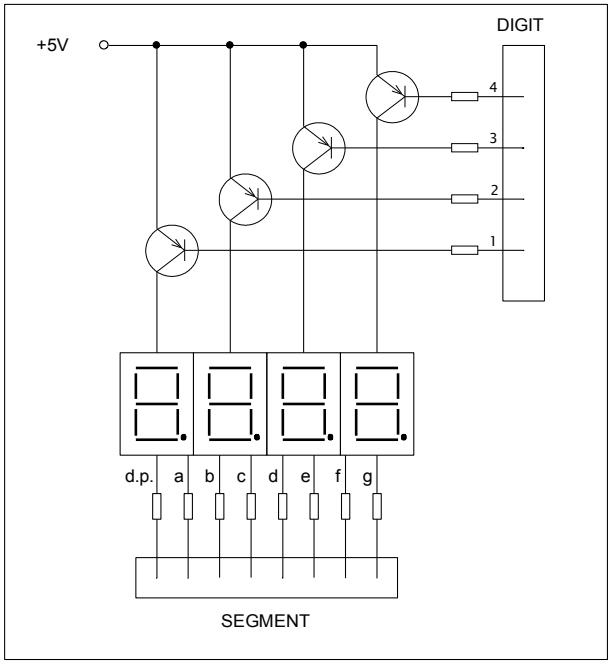
7.1.6.12: Kódový zámek

Po zadání následující sekvence stisků tlačítek (kódu) a po potvrzení tlačítkem S5 se otevře kódový zámek, což je indikováno 10x rychlé bliknutí všech LED.

Kód: S1 … 2x stisk, pak S2 … 4x stisk, pak S3 … 5x stisk

7.2 Řízení 7-segmentového displeje

4 - místný 7-segmentový displej LED je určen pro multiplexní režim řízení, má všechny shodné segmenty všech 4 znakovek propojeny. Na vývojové desce Evb 4.3 je použit typ displeje se společnou anodou, katody jsou aktivní na log. „0. Zapojení displeje je na Obr.14.



Obr.14: Zapojení multiplexně řízeného 7segmentového displeje LED

7.2.1 Zobrazení hexadecimálního znaku zadaného kombinací tlačítek

Zadání:

Program bude číst stav spínačů na portu A a rozsvítí na číslicovce LED řádu jednotek hexadecimální znak odpovídající stisknuté kombinaci spínačů. Nebude-li stisknut žádný spínač, bude na displeji zobrazen znak „0“.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:        disp_led_1
*   Soubor:       znak_hex.c
*
*   Obsah:        Na displeji zobrazi hexacislici odpovidajici
*                    hexa kodu zadanem na S5-S2,pri uvolneni tlac
*                    sviti "0"
*   Vytvoreno:  25/10/2012
*   Autor:
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA
#define LEDKY PORTB
#define ANODY PORTC
#define SEGMENTY PORTD

#define MASKA 0x0f    //vyber dolnich 4 bitu
#define OFF 0xff

//dekoder pro 7-segmentovy displej LED,spol.anody (katody akt. na "0")
unsigned char znak[16] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                          0x01,0x21,0x11,0x07,0x8d,0x43,0x0d,0x1d};

/************  prototypy lokalnich funkci ****************************/
void main(void);
void PortsInit(void);

/************  kod  ***************************************************/
/*****************  funkce pro nastaveni orientace portu *************/
void PortsInit(void)
 {
   DDRA = 0x00;         //port A je vstup
   DDRB = 0xFF;         //port B je vystup
   DDRC = 0xc3;         //bity 0,1,6 a 7 vystupni, pouzity pro spolecne
                        //anody displeje, bity 2-6 pouzity pro JTAG
   DDRD = 0xFF;         //port D je vystup
   PORTA = 0xFF;         //pull-up pripojeny
 }
```

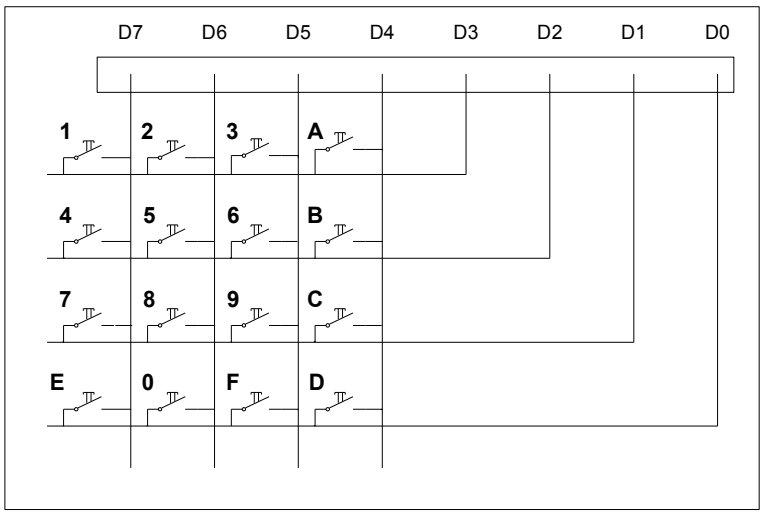
```
/***************  main funkce ****************************************/
void main()
{
  PortsInit();

  LEDKY = OFF;      //vypni LED, jinak pri zapnuti sviti (akt. "0")
  ANODY = 0xc2;     //sviti pouze rad "jednotky"

  while(1)
  {
    SEGMENTY = znak[~SPINACE & MASKA];  //spinace akt."0", proto negace (~)
                                 //MASKA vybira dolni 4 bity portu A
  }
}
```

Popis programu:

Dekodér znaků pro LED displej je realizován polem **znak[16]** o 16 prvcích. V hlavní programové smyčce se periodicky posílá na port C (**SEGMENTY**) ten z prvků pole, jehož index odpovídá kombinaci stisknutých spínačů. Jelikož jsou na všech bitech portu A připojeny pull-up rezistory, musí se horní 4 bity přečteného portu A „maskovat“, což je provedeno logickým součinem s bytem **MASKA** (0x0f).

7.2.2 Zobrazení hexadecimálního znaku zadaného maticovou klávesnicí

Maticová klávesnice je tvořena polem šestnácti tlačítek, zapojených do matice, viz schema zapojení na Obr.15.



Obr.15: Zapojení maticové klávesnice

Maticová klávesnice je používána v mnoha aplikacích, kde je potřebné zadat číselný kód, např. u kódového zámku, zabezpečovacích zařízení, telefonní klávesnice a podobně. Princip maticové klávesnice je použit i u klávesnic běžně používaných u osobních počítačů. Hlavní výhodou je použití malého počtu připojovacích vstupů při velkém počtu tlačítek. V následujícím příkladu bude použit jeden z jednodušších způsobů ovládání klávesnice, který neumožňuje číst kombinaci současně stisknutých kláves.

Zadání:
Na displeji LED zobrazte hexadecimální znak zadaný hexadecimální maticovou klávesnicí.

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:        keyb_disp_led
*   Soubor:       keyb_displ.c
*
*   Obsah:        Na displeji zobrazi hexacislici odpovidajici
*                 stisknute klavese na maticove klavesnici
*
*   Vytvoreno:    25.12.2012
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SLOUPCE PINA                    //maticova klavesnice, horni 4 bity
#define RADEK PORTA           //maticova klavesnice, dolni 4 bity
#define LEDKY PORTB
#define ANODY PORTC                     //spol ANODY 7-segm. displeje
#define SEGMENTY PORTD    //segmenty 7-segm displeje

#define radek1 0xfe                     //spodni radek klavesnice
#define radek2 0xfd
#define radek3 0xfb
#define radek4 0xf7                     //horni radek klavesnice

#define OFF 0xff

// dekoder znaku pro 7-segmentovy displej (na komb. 0xff displej zhasnuty)
unsigned char ZNAK[17] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                    0x01,0x21,0x11,0x07,0x8d,0x43,0x0d,0x1d,0xff};

/************  globalni promenne *********************************************/
static unsigned char dekod_znak;   //dekodovana hodnota klavesy v hexa kodu

/************  prototypy lokalnich funkci ************************************/
int main(void);
void PortInit(void);
void Delay(void);
unsigned char ReadKeyb(void);

/************  kod  **********************************************************/

/*****************  inicializace portu    ************************************/
void PortsInit(void)
{
 DDRA = 0x0f;           //port A horni 4 bity = vstup, dolni 4 bity = vystup
 DDRB = 0xFF;           //port B je vystup (ledky)
```

```
  DDRC = 0xc3;          //bity 0,1,6 a 7 vystupni, pouzity pro anody displeje
  DDRD = 0xFF;          //port D je vystup
  PORTA = 0xff;         //pull-up pripojeny
}
/************  casove zpozdeni  ***************************************/
void Delay(void)
{
  unsigned char i;
  for (i = 0;i < 10;i++);
}
/************ obsluha maticove klavesnice  ********************************/
//vraci hexa kod  naposledy stisknuteho tlacitka

//  zapojeni klavesnice:
//  b7    b6    b5    b4    b3    b2    b1    b0
//  sl1   sl2   sl3   sl4  rad4  rad3  rad2  rad1   (sl1=vlevo,rad1=spodni)

unsigned char ReadKeyb(void)
{
   unsigned char klav_nactena;          //nactena hodnota ze sloupcu klavesnice
   unsigned char pom;

   RADEK = radek4;                      //na 4.radek(horni) log.0

   Delay();                             //ustaleni stavu urovni na portech
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)   //stisknuta klavesa=>najdi sloupec,prirad kod
   {
     switch (klav_nactena)
      {
       case 0x07:dekod_znak = 1;break;    //1.sloupec
       case 0x0b:dekod_znak = 2;break;    //2.sloupec
       case 0x0d:dekod_znak = 3;break;    //3.sloupec
       case 0x0e:dekod_znak = 10;break;   //4.sloupec
      }
   }

   RADEK = radek3;          //na 3.radek log.0
   Delay();
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 4;break;
       case 0x0b:dekod_znak = 5;break;
       case 0x0d:dekod_znak = 6;break;
       case 0x0e:dekod_znak = 11;break;
     }
   }

   RADEK = radek2;        //na 2.radek log.0

   Delay();
```

```
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 7;break;
       case 0x0b:dekod_znak = 8;break;
       case 0x0d:dekod_znak = 9;break;
       case 0x0e:dekod_znak = 12;break;
     }
   }

   RADEK = radek1;         // na1.radek(spodni)log.0
   Delay();
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 14;break;
       case 0x0b:dekod_znak = 0;break;
       case 0x0d:dekod_znak = 15;break;
       case 0x0e:dekod_znak = 13;break;
     }
   }
}
/*****************  main funkce *******************************************/
int main(void)
{
 ANODY = 0xc2;     //aktivovan pouze nejnizsi rad displeje (jednotky)
 LEDKY = OFF;      //vypni LED, jinak pri zapnuti sviti (akt. "0" )

 PortsInit();

 while(1)               //nekonecna smycka
 {
    ReadKeyb();
    SEGMENTY = ZNAK[dekod_znak];
 }
}
```

Popis programu:

Maticová klávesnice je připojena k portu A podle schematu na obrázku Obr. 15. Stisknutá klávesa je vyhodnocována funkcí **ReadKeyb()**, která modifikuje globální proměnnou **dekod_znak**. Hodnota této proměnné je použita jako index pole **ZNAK**, prvky tohoto pole aktivují příslušné segmenty displeje LED. Displej pracuje ve statickém režimu.

Popis funkce **ReadKeyb()** :

Funkce postupně aktivuje všechny čtyři řádky klávesnice tak, že posílá na řádky log. „0“ a poté čte stav sloupců klávesnice. Pokud je některá klávesa stisknuta (if(klav_nactena != 0x0f)), přečte se kombinace na sloupcích klávesnice a pomocí přepínače **switch(klav_nactena)** se přiřadí odpovídající znak proměnné

dekod_znak.

7.2.3 Reverzibilní čítač MOD10 (0 až 9) ovládaný tlačítky

Zadání:
Na displeji LED se zobrazí stav reverzibilního čítače MOD10. Inkrementace spínačem S5, dekrementace spínačem S4, nulování čítače spínačem S1.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:        disp_led_3
*   Soubor:         citac_mod10.c
*
*   Obsah:          Reverzibilni citac MOD10,inkrementace S5,dekrementace S4
*                   nulovani S1
*   Vytvoreno:      25/10/2012
*   Autor:
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA
#define LEDKY PORTB
#define ANODY PORTC
#define SEGMENTY PORTD

#define OFF 0xff

#define S5 0
#define S4 1
#define S3 2
#define S2 3
#define S1 4

#define bS5 0
#define bS4 1
#define bS3 2
#define bS2 3
#define bS1 4

/************  makra pro bitove operace ******************************/
#define  SETBIT(BIT_POLE,BIT) (BIT_POLE|=(1<<BIT))       //nahozeni bitu
#define  CLRBIT(BIT_POLE,BIT) (BIT_POLE &= ~(1<<BIT))    //nulovani bitu
#define  TESTBIT(BIT_POLE,BIT) (BIT_POLE & (1<<BIT))     //nahozeni bitu
#define  TESTNEGBIT(BIT_POLE,BIT) (~BIT_POLE & (1<<BIT)) //test nuloveho bitu
#define  NEGBIT(BIT_POLE, BIT)  (BIT_POLE ^= (1<<(BIT))) //negace bitu

/************  definice globalnich promennych ************************/
unsigned char bajt1;
```

```
//dekoder pro 7-segmentovy displej LED,spol.anody (katody akt. na "0")
unsigned char znak[16] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                          0x01,0x21,0x11,0x07,0x8d,0x43,0x0d,0x1d};

/************  prototypy lokalnich funkci ******************************/
void main(void);
void PortsInit(void);
void Delay(void);

/****************  kod  ************************************************/
/****************  funkce pro nastaveni orientace portu *****************/
void PortsInit(void)
{
  DDRA = 0x00;     //port A je vstup
  DDRB = 0xFF;    //port B je vystup
  DDRC = 0xc3;    //bity 0,1,6 a 7 vystupni, pouzity pro spolecne
                      //anody displeje, bity 2-6 pouzity pro JTAG
  DDRD = 0xFF;    //port D je vystup
  PORTA = 0xFF;   //pull-up pripojeny
}
/*************  casove zpozdeni  ***************************************/
void Delay(void)
{
  unsigned char i;
  for (i = 0;i < 10;i++);
}
/****************  main funkce ****************************************/
void main(void)
{
  PortsInit();

  LEDKY = OFF;      //vypni LED, jinak pri zapnuti sviti (akt. "0")
  ANODY = 0xc2;     //sviti pouze rad "jednotky" MPX displeje

  while(1)
  {
    static unsigned char citac;                 //aktualni stav citace

    if (TESTNEGBIT(SPINACE,S5) && TESTBIT(bajt1,bS5))    //test sepnuti S5
    {
      CLRBIT(bajt1,bS5);
      citac ++;
      if (citac == 10) citac = 0;             //"preteceni" citace MOD10
    }
    if (TESTNEGBIT(SPINACE,S4) && TESTBIT(bajt1,bS4))    //test sepnuti S4
    {
      CLRBIT(bajt1,bS4);
      citac --;
      if (citac == 255) citac = 9;  //"podteceni" citace pod "0"
    }
    if (TESTNEGBIT(SPINACE,S1) && TESTBIT(bajt1,bS1))     //test sepnuti S1
    {
      CLRBIT(bajt1,bS1);
      citac = 0;
```

```
    }
    Delay();
    if (TESTBIT(SPINACE,S5)) SETBIT(bajt1,bS5);       //test uvolneni S5
    if (TESTBIT(SPINACE,S4)) SETBIT(bajt1,bS4);       //test uvolneni S4
    if (TESTBIT(SPINACE,S1)) SETBIT(bajt1,bS1);       //test uvolneni S1

    SEGMENTY = znak[citac];
  }
}
```

Popis programu:

Čítač je realizován inkrementací spínačem S5, dekrementací S4 a nulováním S1. Čtení stavu spínačů se provádí pomocí maker pro práci s bity. Pro detekci sestupné a náběžné hrany spínačů jsou použity pomocné bity bS5, bS4 a bS1 uložené v proměnné bajt1.

7.2.4 Multiplexní režim displeje

Vícemístné 7 segmentové displeje LED mají vzájemně propojeny souhlasné segmenty všech číslicovek. Je to z důvodu menšího počtu vývodů displeje a hlavně menšího počtu výstupů řídícího obvodu, v našem případě mikrokontroléru.

U těchto displejů se používá pro jejich řízení tzv. „multiplexní režim“, kdy v jednom okamžiku svítí pouze jedna číslice z celého zobrazeného čísla a tyto se postupně přepínají stále dokola. Díky nedokonalosti oka není při dostatečném kmitočtu přepínání blikání číslicovek patrné, zobrazené číslo se jeví klidné jako při statickém řízení.

Zadání:
Na 4-místném 7-segmentovém displeji LED zobrazte číslo 1234.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       disp_led_4
*   Soubor:        mpx_led_1.c
*
*   Obsah:         Multiplexne rizeny displej, na displeji se zobrazi
*                  zleva cislice 1234
*
*   Vytvoreno:   8.12.2012
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>

/************  definice konstant *********************************************/
#define SPINACE PINA
#define LEDKY PORTB
```

```
#define ANODY PORTC  //ANODY cislicovek,pouzity bity 7,6,1 a 0, zbyle vyuziva JTAG
#define SEGMENTY PORTD

#define OFF 0xff
#define ON  0x00

#define DIS4  0xc2      //                DIS4
#define DIS3  0xc1      //            DIS3
#define DIS2  0x83      //       DIS2
#define DIS1  0x43      // DIS1
#define VYPNUTO    0xff
#define ZHASNUTO  0xff

/************  definice globalnich promennych *****************************/

//pole - dekoder 7-segmentove hodispleje
unsigned char znak[16] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                          0x01,0x21,0x11,0x07,0x8d,0x43,0x0d,0x1d};

/************  prototypy lokalnich funkci *********************************/
int main(void);
void Delay (void);

/************  kod  ******************************************************/
/*************** Delay  ***************************************************/
void Delay (void)
{
  int k;
  for(k=0;k<500;k++);
}
/***************  main funkce ***********************************************/
int main(void)
{
          DDRA = 0x00;  //port A je vstup
          DDRD = 0xFF;  //port D je vystup
      DDRB = 0xFF;  //port B je vystup
      PORTA = 0xFF; //pull-up pripojeny

      DDRC = 0xc3; //bity 0,1,6 a 7 vystupni
      LEDKY = OFF;  //vypni LED, jinak pri zapnuti sviti, akt. "0"

  while(1)
  {
     SEGMENTY = znak[4];     //kombinace znaku jednotek na katody
     ANODY = DIS4;           //aktivuj anodu jednotek
     Delay();                    //nech jednotky chvili svitit
     ANODY = ZHASNUTO;     //zhasni displej

     SEGMENTY = znak[3];     //kombinace znaku desitek na katody
     ANODY = DIS3;           //aktivuj anodu desitek
     Delay();                //nech desitky chvili svitit
     ANODY = ZHASNUTO;     //zhasni displej

     SEGMENTY = znak[2];     //kombinace znaku stovek na katody
     ANODY = DIS2;           //aktivuj anodu stovek
```

```
    Delay();                    //nech stovky chvili svitit
    ANODY = ZHASNUTO;    //zhasni displej

    SEGMENTY = znak[1];    //kombinace znaku tisicu na katody
    ANODY = DIS1;              //aktivuj anodu tisicu
    Delay();                    //nech tisice chvili svitit
    ANODY = ZHASNUTO;    //zhasni displej
  }
}
```

Popis programu:

Na katody displeje se pošle kombinace nul a jedniček, která odpovídá požadovanému 1. rozsvícenému znaku. Poté se aktivuje společná anoda první 7-segmentovky, nechá se krátkou dobu svítit (**Delay()**), pak se deaktivuje. Dále se změní kombinace nul a jedniček odpovídající 2. rozsvícenému znaku, aktivuje se společná anoda 2. znakovky, opět se nechá chvíli svítit, zhasne se, atd. Celý postup se opakuje dokola pro všechny znakovky. V tomto demonstračním programu je ovládání displeje jedinou činností, která se zde provádí. V následujících programech bude ovládání displeje řešeno funkcí.

7.2.5 Reverzibilní čítač MOD 10000 ovládaný tlačítky

Zadání:
Na displeji LED se zobrazí stav reverzibilního čítače MOD1000. Inkrementace spínačem S5, dekrementace spínačem S4, nulování čítače spínačem S1.

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:       disp_led_5
*   Soubor:        mpx_led_2.c
*
*   Obsah:          Multiplexne rizeny displej, na displeji se zobrazi
*                   obsah 4-mistneho citace  MOD 10000
*                   inkrementace S5, dekrementace S4, nulovani S1
*   Vytvoreno:    8.12.2012
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>
#include "dis_led.h"
#include "spin.h"
#include "makra.h"
#include "evb43_1.h"

/************  definice konstant *********************************************/
#define OFF 0xff
#define ON  0x00
```

```
/************  definice globalnich promennych *****************************/
unsigned char desitky,jednotky,stovky,tisice;
long int citac = 0;
unsigned char bajt1;

/************  prototypy lokalnich funkci *********************************/
int main(void);
void Delay(void);
void PortsInit(void);
void AktivujCitac(void);
void AktivujDisplej(void);
int Prevod(long int cislo);

/************  kod  ******************************************************/

/***************** kratke casove zpozdeni  *********************************/
void Delay(void)
{
   int k;
   for(k=0;k<500;k++);
}

/*****************  inicializace portu ***************************************/
void PortsInit(void)
{
  DDRA = 0x00;  //port A je vstup
  DDRD = 0xFF;  //port D je vystup
  DDRB = 0xFF;  //port B je vystup
  PORTA = 0xFF; //pull-up pripojeny
  DDRC = 0xc3;  //bity 0,1,6 a 7 vystupni
}
/**********  aktivace citace  ***********************************************/
void AktivujCitac(void)
{
    if(TESTNEGBIT(SPINACE,0) & TESTBIT(bajt1,0))   //inkrementace citace
    {
      CLRBIT(bajt1,0);
      citac ++;
      if(citac > 9999) citac = 0;
      Prevod(citac);                    //prevod pouze pri zmene obsahu citace
      Delay();                          //casove zpozdeni pro ustaleni stavu tlacitka
    }
    if(TESTNEGBIT(SPINACE,1) & TESTBIT(bajt1,1))   //dekrementace citace
    {
      CLRBIT(bajt1,1);
      citac --;
      if(citac < 0) citac = 9999;
      Prevod(citac);
      Delay();
    }
    if(TESTNEGBIT(SPINACE,4) & TESTBIT(bajt1,4))   //nulovani citace
    {
      CLRBIT(bajt1,4);
      citac = 0;
      Prevod(citac);
```

```
      Delay();
    }
    if (SPINACE == OFF)
    {
       SETBIT(bajt1,0);
       SETBIT(bajt1,1);
       SETBIT(bajt1,4);
    }
}
/***************  AktivujDisplej  ***************************************/
void AktivujDisplej(void)
{
 SEGMENTY = znak[jednotky];  //kombinace znaku jednotek na anody
 ANODY = DIS4;               //aktivuj anodu jednotek
 Delay();                    //nech jednotky chvili svitit
 ANODY = OFF;                //zhasni displej

 SEGMENTY = znak[desitky];   //kombinace znaku desitek na anody
 ANODY = DIS3;               //aktivuj anodu desitek
 Delay();                    //nech desitky chvili svitit
 ANODY = OFF;                //zhasni displej

 SEGMENTY = znak[stovky];    //kombinace znaku stovek na anody
 ANODY = DIS2;               //aktivuj anodu stovek
 Delay();                    //nech stovky chvili svitit
 ANODY = OFF;                //zhasni displej

 SEGMENTY = znak[tisice];    //kombinace znaku tisicu na anody
 ANODY = DIS1;               //aktivuj anodu tisicu
 Delay();                    //nech tisice chvili svitit
 ANODY = OFF;                //zhasni displej
}

/*************** funkce pro vypocet hodnot displeju  **********************/
/*** vypocte z 4-mistneho desitkoveho cisla hodnoty jednotlivych radu *********/
int Prevod(long int cislo)
{
    unsigned int pom;       //pomocná promenna

    tisice=cislo/1000;
    pom = cislo - (tisice * 1000);
    stovky = pom/100;
    pom = pom - (stovky * 100);
    desitky = pom/10;
    jednotky = pom - (desitky * 10);
}

/***************  main funkce ***************************************************/
int main(void)
{
 PortsInit();
 LEDKY = OFF;

 while(1)
 {
```

```
    AktivujCitac();    //cte stav spinacu,inkrementuje/dekrementuje/nuluje citac
    AktivujDisplej();  //zobrazi stav citace
  }
}
```

Popis programu:

Funkce **Aktivuj Citac()** čte stav spínačů, je-li některý aktivní, provede inkrementaci, dekrementaci popř. nulování proměnné **citac**.

Funkce **Prevod()** vypočítává ze čtyřmístného desítkového čísla hodnoty jednotlivých řádů čísla, modifikuje globální proměnné **tisice**, **stovky**, **desitky**, **jednotky**, které jsou zobrazeny na displeji aktivací funkce **AktivujDisplej()**. Je zde použito celočíselné dělení, kterým získáme hodnotu příslušného řádu od nejvyššího (tisíce) až po nejnižší (jednotky). Proměnná **pom** slouží pro uložení vypočteného zbytku po celočíselném dělení.

V programu jsou použity tzv. „**hlavičkové soubory**“, značí se příponou ***.h**. V těchto souborech jsou definice konstant, maker apod. Jejich použitím se stává program přehlednější, výpis programu je kratší. Překladač vloží tyto soubory do programu při překladu zdrojového souboru ***.C**.

Existují dva druhy hlavičkových souborů, liší se svým umístěním:

- Pokud jsou názvy těchto souborů uvozeny závorkami **<** a **>** , hledá překladač tyto soubory v tzv. knihovnách, což jsou soubory s příponou ***.lib** (library). Toto jsou knihovny překladače jazyka C, příslušné danému typu mikrokontroléru, viz příkaz **#include <atmega32.h>**, který používáme ve všech našich ukázkových programech. Tento soubor obsahuje definice vstupně/výstupních registrů mikrokontroléru Atmega32, definice jejich bitů, adresy vektorů přerušení a podobně.
- Pokud jsou názvy hlavičkových souborů uvozeny uvozovkami **“** a **“** , hledá překladač tyto soubory ve složce, kde je umístěn zdrojový soubor ***.C** . Uživatelsky vytvářené hlavičkové soubory využívají tento způsob uložení a v našem programu je použit.

Ve vývojovém prostředí CrossStudio, které používáme, můžeme zobrazit obsahy hlavičkových souborů poklepáním pravým tlačítkem myši na název souboru a výběrem položky „**Go To Definition**“.

V následujícím textu je uveden obsah vytvořených hlavičkových souborů, některé z nich budou použity i v následujících ukázkových programech a jejich obsah již nebude uváděn.

spin.h

```
//definice bitu spinacu
#define S5 0
#define S4 1
#define S3 2
#define S2 3
#define S1 4

//definice pomocnych bitu pro spinace
```

```
#define bS5 0
#define bS4 1
#define bS3 2
#define bS2 3
#define bS1 4
```

dis.h

```
//konstanty pro aktivaci spolecnych anod znakovek LED displeje
#define DIS4  0xc2    //                DIS4
#define DIS3  0xc1    //           DIS3
#define DIS2  0x83    //      DIS2
#define DIS1  0x43    // DIS1

//dekoder 7-segmentoveho displeje bez zobrazeni desetinne tecky
//pri posledni kombinaci je displej zhasnuty
unsigned char znak[11] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                          0x01,0x21,0xff};

//dekoder 7-segmentoveho displeje vcetne rozsvicene desetinne tecky
unsigned char znak_dt[11] = {0x80,0xf2,0x48,0x60,0x32,0x24,0x04,0xf0,
                          0x00,0x20,0xfe};
```

makra.h

```
// makra pro bitove operace
#define  SETBIT(BIT_POLE,BIT) (BIT_POLE|=(1<<BIT))                //nahozeni bitu
#define  CLRBIT(BIT_POLE,BIT) (BIT_POLE &= ~(1<<BIT))             //nulovani bitu
#define  TESTBIT(BIT_POLE,BIT) (BIT_POLE & (1<<BIT))              //nahozeni bitu
#define  TESTNEGBIT(BIT_POLE,BIT) (~BIT_POLE & (1<<BIT))          //test nuloveho bitu
#define  NEGBIT(BIT_POLE, BIT)  (BIT_POLE ^= (1<<(BIT)))          //negace bitu
```

evb43_1.h

```
//definice prirazeni periferií k portum
#define SPINACE PINA          //spinace S1 až S5
#define LEDKY PORTB
#define ANODY PORTC           //spol ANODY 7-segm. displeje
#define SEGMENTY PORTD  //segmenty 7-segm displeje
```

7.2.6 Úlohy k samostatnému řešení

7.2.6.1: Vajíčkový časovač

Po startu programu se zobrazí na displeji čas 02:30 (min : sek). Po stisku S5 se začne čas odečítat, při dosažení času 00:00 se rozsvítí Led0 a na displeji se zobrazí znovu čas 02:30. Při opětovném stisku S5 Led0 zhasne a spustí se znovu odpočet času.

7.2.6.2: Sekundové stopky

Po startu programu se zobrazí na displeji čas 00:00 (min:sek).

Ovládání stopek:

- S1 – start
- S2 – zamrznutí displeje, opětovným stiskem S2 normální zobrazení času
- S3 – stop
- S4 – nulování

7.2.6.3: Nápis HALO SOUE

Po stisku S1 se na 7-segmentovém LED displeji objevuje střídavě po 1 s nápis „HALO“ a „SOUE“. Po stisku S2 displej zhasne.

7.2.6.4: Běžící text

Po zapnutí programu bude na 7-segmentovém displeji rolovat dokola směrem doleva text: „ HALO SOUE “. Před každým slovem textu jsou dvě mezery. Rychlost rolování přibližně 0,5 s.

7.2.6.5: Násobilka 5

Po zapnutí programu se budou na 7-segmentovém displeji po 0,5 s zobrazovat čísla odpovídající násobilce 5 ve tvaru: 0 0, 1 5, 2 10, 3 15, 4 20, 5 25, 6 30, atd až do 1999 9995 .

7.2.6.6: Generátor pseudonáhodného čísla

Po každém stisku S1 se na 7-segmentovém displeji zobrazí aktuální stav volně běžícího čítače MOD 100 kde zůstane až do dalšího stisku S1.

7.2.6.7: Závodník na okruhu

Po stisku S5 se budou postupně přibližně po 0,3s rozsvěcovat vnější segmenty 4-místného 7-segmentového displeje doprava, čímž se vytvoří dojem běžícího závodníka na okruhu. Prvním rozsvíceným segmentem bude segment „b“ na pravé znakovce.

Po stisku S1 to samé doleva, počínaje segmentem „f “ na levé znakovce.

Po stisku S3 zůstane svítit poslední rozsvícený segment.

7.2.6.8: Start rakety

Po stisku S5 se postupně odečítá na 7-segmentovém LED displeji čas od 9 do 0 po 1s , potom 5x zablikají rychle (asi po 0,2s) všechny segmenty celého LED displeje a pak vše zhasne. Po stisku S5 se vše opakuje.

7.2.6.9: Světelný had

Na 4-místném 7-segmentovém LED displeji vytvořte postupným rozsvěcením segmentů iluzi pohybu hada po dráze široké osmičky (přes všechny znakovky). Po dobu stisku S5 doprava, po dobu stisku S1 doleva, při uvolnění tlačítka had zmizí. Rychlost přepínání segmentů asi 0.3 s.

7.2.6.10: Odhad času v sekundách

Po stisku S1 odstartujte měření času v sekundách, po uvolnění S1 se údaj zobrazí na displeji. Po opětovném stisku displej zhasne a začíná nový odpočet času. Zkalibrujte časovou smyčku pro maximální odchylku 1s za dobu 1 minuty.

7.2.6.11: Generátor PWM signálu.

Program generuje PWM signál s periodou 10s. Doba zapnutí výstupu je nastavitelná čítačem MOD10 ovládaným tlačítkem S1 a zobrazeným na displeji. Výstup generátoru bude na portu B, bit0.

7.2.6.12: Speciální čítač1

Na 3-místném 7-segmentovém displeji LED se budou zobrazovat následující číslice synchronně:

- Na pozici jednotky: číslice od 0 do 8 a zpět po 1s (0,1,2,3,4,5,6,7,8,7,6...1,0,1,..)
- Na pozici desítky: číslice od 0 do 8 po 2 a zpět po 2s (0,2,4,6,8,6,4,2,0,2,...)
- Na pozici stovky: číslice od 0 do 8 po 4 a zpět po 4s (0,4,8,4,0,4,...)

Pozn.: Generování číslic pomocí příkazu cyklu „for ( )“

7.2.6.13: Speciální čítač2

Na 4-místném 7-segmentovém displeji LED se bude zobrazovat následující běžící posloupnost číslic: 0,1,2,3,4,5,6,7,8,9,8,7,6,5,4,3,2,1,0,1,2... po 1s.

7.2.6.14: Kódový zámek 2

Po zadání kódu **1832** a jeho potvrzení spinačem S5 se otevře zámek dveří, což je signalizováno 10x rychlým bliknutím všech Led na portu B. Kód se zadává spínači S1 až S4 a zobrazuje se na 4-místném 7-segmentovém displeji LED. Každý ze spínačů ovládá jednu číslici displeje dokola od 0 do 9. Při nesprávném zadání kódu a potvrzení S5 se kód vynuluje.

7.3 Obsluha přerušení

Přerušení přeruší běh zpracovávaného programu a přesměruje jej na určitou obslužnou rutinu, po jejímž vykonání se vrátí zpět na místo, kde se program přerušil. Pokud přijde při vykonávání obslužného programu přerušení požadavek na další přerušení a má-li tento požedavek vyšší prioritu, obslužný program přerušení s nižší prioritou se přeruší a začne se vykonávat program pro obsluhu přerušení s vyšší prioritou.

7.3.1 Externí přerušení INT0

Zadání:

Hlavní program bude blikat Led0 s periodou 10s. Externí přerušení způsobí negaci Led0 v libovolném stavu. Externí přerušení bude vyvoláno stiskem spínače S1, který je připojen ke vstupu INT0 proti zemi, přerušení bude vyvoláno sestupnou hranou impulsu.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       ext_int
*   Soubor:        ext_int0.c
*
*   Obsah:         Hlavni program blika Led0 s periodou 10s
*                  S1 pripojene na INT0 neguje lED0 okamzite
*   Vytvoreno:     26.3.2013
*
*
*****************************************************************************/
/************  include files ************************************************/
#include <atmega32.h>
#include "makra.h"
```

```
/************  definice globalnich promennych *****************************/
unsigned char b_timeout;

/************ definice konstant ****************************************/
#define LEDKY PORTB   //na portB pripojeny LED

#define LED0 0
#define OFF 0xff
#define ON 0x00

/************  prototypy lokalnich funkci *******************************/
void main(void);
void Delay(void);
void Time100ms(unsigned char nasobek);
void PortsInit(void);
void ExtIntInit(void);
/************  kod  ****************************************************/

/************   casove zpozdeni  ***************************************/
void Delay(void)
{
  unsigned int i;

  for (i=0;i<1000;i++);
}

/***************** funkce casoveho zpozdeni *******************************/
void Time100ms(unsigned char nasobek)
{
  static unsigned int i,j;
  i++;
  if (i == 3018)           //3018 cas konst pro pribl 100ms zpozdeni
  {
     i=0;
     j++;

     if(j==nasobek)
     {
       j = 0;
       b_timeout=1;       //priznak dosazeni casu (100ms x nasobek)
     }                     //ktery se testuje v hlavnim programu a nuluje pro dalsi
  }                        //odecet casu
}
/************** funkce nastaveni orientace portu  ***********************/
void PortsInit(void)
{
   DDRB = 0xff;                    //port B je vystup

   CLRBIT(DDRD,DDD2);        //nastaveni pinu INT0 jako vstup (lze i jako vystup)
   SETBIT(PORTD,PORTD2); //pull-up na pin INT0 (nutne)
}
/************** nastaveni prerusovaciho systemu ************************/
void ExtIntInit(void)
{
  SETBIT(GICR,INT0);           //nastaveni preruseni od INT0
```

```
  SETBIT(MCUCR,ISC01);      //preruseni od INT0 na sestupnou hranu
  SETBIT(SREG,7);           //globalni povoleni preruseni
}

/************** interruptova rutina INT0 ******************************/
//pouze neguje Led0
void ext_inter_0(void)__interrupt[INT0_vect]
{
  NEGBIT(LEDKY,LED0);
  Delay();
}
/****************  main funkce ****************************************/
void main(void)
{
  PortsInit();
  ExtIntInit();

  LEDKY = OFF;

  while(1)
  {
    Time100ms(50);
    if(b_timeout)         //po 5s (50x100ms)
    {
      b_timeout = 0;
      NEGBIT(LEDKY,LED0);  //Led0 rozsviti/zhasne
    }
  }
}
```

Popis programu:

Ve smyčce while(1) je pouze volána funkce pro časové zpoždění a periodicky přibližně po 5s mění stav Led0 (Led0 bliká).

Externí přerušení je vyvoláno stiskem spínače připojeným na vstup **INT0** mikrokontroléru. Toto přerušení způsobí okamžitou změnu stavu (svitu) **Led0**, nezáleží na tom, ve kterém stavu se Led0 nachází. Interruptová rutina se jménem „ **ext_inter_0()** “ je založena jako běžná funkce **C**, liší se však tím, že za jménem funkce je (nutně) umístěno „ **__interrupt[INT0_vect]** “. V hranaté závorce je umístěn tzv. „ vektor přerušení“, což je vlastně adresa, kde je příslušná rutina umístěna v paměti programu. Překladač pozná, že jde o rutinu přerušení, po jejíž vyvolání zajistí zákaz dalšího přerušení, uklizení registrů používaných touto rutinou (aby rutina nepřepsala původní obsah registrů používaných v hlavním programu) a po ukončení návrat do hlavního programu za místo, odkud byla rutina volána.

Před použitím přerušení musí být příslušný typ přerušení povolen zápisem log.1 do příslušného bitu povolovaného přerušení v registru **GICR** (General Interrupt Control Register).

Dále je nutné nastavit druh aktivace přerušení, resp. citlivost na náběžnou, sestupnou nebo na obě hrany signálu nebo na úroveň log.0. Toto se provede nastavením bitů **ISC00, ISC01** v registru **MCUCR**.

Též musí být nastaven bit **I** globálního povolení přerušení v registru **SREG** .

Hlavičkový soubor **makra.h** je již použit v předchozích příkladech (obsahuje definici maker pro práci s bity) a musí být umístěn ve složce, kde je zdrojový soubor programu **ext_int0.c** .

7.3.2 Externí přerušení INT0, INT1

Zadání:
Hlavní program bude blikat Led0, doba svitu a doba zhasnuti Led0 bude nastavitelná tlacitky S1 a S2 připojenými na interuptové vstupy INT0 a INT1 proti zemi, přerušení budou vyvolána sestupnou hranou impulsu. Výchozí doba bude 1000ms, inkrementace a dekrementace doby po 100ms v rozsahu konstant **perioda1** a **perioda2**. Nastavená doba bude zobrazena na na 7-segmentovém LED displeji v milisekundách.

Ukázkový program:

```
/*****************************************************************************
*   Projekt:        ext_int_0_1
*   Soubor:         ext_int0_int1.c
*
*   Obsah:          tlacitky na interuptovych vstupech INT0 a INT1 se
*                   zkracuje (INT0) nebo prodluzuje (INT1) doba svitu
*                   a doba zhasnuti LED0 po 100 ms od hodnoty perioda1
*                   do perioda2, doba bude zobrazena v ms na LED displeji
*   Vytvoreno:      30.3.2013
*
*****************************************************************************/
/************  include files ************************************************/
#include <atmega32.h>
#include "makra.h"

/************  definice globalnich promennych ******************************/
unsigned int perioda_blik;
unsigned char b_byte;                       //byte pro bitove promenne
unsigned char desitky,jednotky,stovky,tisice;

/************ definice konstant ********************************************/
#define LEDKY PORTA
#define DISPLEJ PORTB
#define KATODY PORTC                    //katody cislicovek,pouzity bity 7,6,1 a 0

#define DIS4  0xc2              //                DIS4
#define DIS3  0xc1              //           DIS3
#define DIS2  0x83              //      DIS2
#define DIS1  0x43              // DIS1

#define LED0 0
#define bit_timeout 0
#define bit_pozice 1
```

```
#define OFF 0xff
#define ON 0x00

#define perioda1 1                // * 100ms (= 0.1s)
#define perioda2 50               // * 100ms (= 5s)

//pole - dekoder 7-segmentoveho displeje
unsigned char znak[11] = {0x81,0xf3,0x49,0x61,0x33,0x25,0x05,0xf1,
                          0x01,0x21,0xff};

/************  prototypy lokalnich funkci *********************************/
void main(void);
void Time100ms(unsigned char nasobek);
int Prevod(long int cislo);
void ZobrazZnakMpxDispl(void);
void PortsInit(void);
void ExtIntInit(void);
/************  kod  ****************************************************/

/***************** funkce casoveho zpozdeni *********************************/
void Time100ms(unsigned char nasobek)  //funkce nahazuje po case:100ms * nasobek
                    // bit "b_timeout",
                    //ktery shazuje po detekci volajici program
{
//vnejsi casova smycka
 static unsigned int i,j,l;
 i++;
 l++;

 if(l > 50)          //l urcuje periodu multiplexu displeje
 {
     l=0;
     SETBIT(b_byte,bit_pozice);
 }

 if (i == 1500)      //1500 tato konstanta zpusobi pribl.100ms zpozdeni
 //vnitrni casova smycka
 {
     i=0;
     j++;
     if(j > nasobek) j = 0;  //pri zmene nasobku, je-li aktualni hodnota
                       //j > nove zadany nasobek casu 100ms
     if(j == nasobek)
     {
       j = 0;
       SETBIT(b_byte,bit_timeout); //priznak dosazeni casu (100ms x nasobek)
     }                           //ktery se testuje v hlavnim programu
 }                               //a nuluje pro dalsi odecet casu

}

/******************** funkce pro vypocet hodnot displeju  *********************/
int Prevod(long int cislo)
{
   unsigned int pom;
```

```
    tisice=cislo/1000;
    pom = cislo - (tisice * 1000);
    stovky = pom/100;
    pom = pom - (stovky * 100);
    desitky = pom/10;
    jednotky = pom - (desitky * 10);
}

/***************** funkce zobrazeni znaku na MPX displeji *********************/
void ZobrazZnakMpxDispl(void)
{
     static unsigned char count_pozice;

     if(TESTBIT(b_byte,bit_pozice))   //po urcitem case se prepina multiplex
     {                                // displeje
        CLRBIT(b_byte,bit_pozice);
        count_pozice ++;
        if(count_pozice == 4) count_pozice = 0;

        if(count_pozice == 0)
        {
          KATODY = OFF;                 //zhasni displej
          DISPLEJ = znak[jednotky];     //kombinace znaku jednotek na anody
          KATODY = DIS4;                //aktivuj katodu jednotek
        }
        if(count_pozice == 1)
        {
          KATODY = OFF;                 //zhasni displej
          DISPLEJ = znak[desitky];      //kombinace znaku jednotek na anody
          KATODY = DIS3;                //aktivuj katodu jednotek
        }
        if(count_pozice == 2)
        {
          KATODY = OFF;                 //zhasni displej
          DISPLEJ = znak[stovky];       //kombinace znaku jednotek na anody
          KATODY = DIS2;                //aktivuj katodu jednotek
        }
        if(count_pozice == 3)
        {
          KATODY = OFF;                //zhasni displej
          DISPLEJ = znak[tisice];      //kombinace znaku jednotek na anody
          KATODY = DIS1;               //aktivuj katodu jednotek
        }
     }
}

/************** funkce nastaveni orientace portu  ************************/
void PortsInit(void)
{
   DDRA = 0xff;                    //port A je vystup
   DDRB = 0xff;                    //port B je vystup
   DDRC = 0xc3;                    //bity 0,1,6 a 7 vystupni

   CLRBIT(DDRD,DDD2);         //nastaveni pinu INT0 jako vstup (lze i jako vystup)
```

```
   CLRBIT(DDRD,DDD3);                //nastaveni pinu INT1 jako vstup (lze i jako vystup)
   SETBIT(PORTD,PORTD2);             //pull-up na pin INT0 (nutne)
   SETBIT(PORTD,PORTD3);             //pull-up na pin INT1 (nutne)
}
/************** nastaveni prerusovaciho systemu ************************/
void ExtIntInit(void)
{
   SETBIT(GICR,INT0);          //nastaveni preruseni od INT0
   SETBIT(GICR,INT1);          //nastaveni preruseni od INT1
   SETBIT(MCUCR,ISC01);        //preruseni od INT0 na sestupnou hranu
   SETBIT(MCUCR,ISC11);        //preruseni od INT1 na sestupnou hranu
   SETBIT(SREG,7);             //globalni povoleni preruseni
}

/*************** interruptova rutina INT0 *******************************/
//zkracuje periodu blikani do hodnoty perioda1
void ext_inter_0(void)__interrupt[INT0_vect]
{
 if (perioda_blik > perioda1) perioda_blik--;
 Prevod(perioda_blik * 100);
}

///*************** interruptova rutina INT1 *******************************/
//prodluzuje periodu blikani do hodnoty perioda2
void ext_inter_1(void)__interrupt[INT1_vect]
{
 if (perioda_blik < perioda2) perioda_blik++;
 Prevod(perioda_blik * 100);
}

/***************  main funkce ****************************************/
void main(void)
{
 PortsInit();
 ExtIntInit();

 LEDKY = OFF;
 perioda_blik = 10;              //vychozi perioda blikani Led0 (priblizne 500ms)
 Prevod(perioda_blik * 100);

 while(1)
 {
  Time100ms(perioda_blik);
  if(TESTBIT(b_byte,bit_timeout))
  {
   CLRBIT(b_byte,bit_timeout);
   NEGBIT(LEDKY,LED0);     //Led0 rozsviti/zhasne
  }
  ZobrazZnakMpxDispl();
 }
}
```

Popis programu:

Interuptové rutiny inkrementují (INT0) a dekrementují (INT1) globální proměnnou

perioda_blik, která je vstupním parametrem funkce **Time100ms(perioda_blik),** určující dobu svitu a zhasnutí **Led0**. Tato funkce nahazuje bitovou proměnnou **bit_timeout** po uplynutí doby (100ms * perioda_blik). Tato bitová proměnná se testuje při každém průchodu nekonečnou smyčkou while(1), je-li tato proměnná nastavena, nuluje se pro další odpočet doby a zároveň se neguje stav **Led0**. Funkce **Time100ms(perioda_blik)** nahazuje (po 50 voláních) též bitovou proměnnou **bit_pozice**, který je použit pro přepínání multiplexu 7-segmentového displeje LED.

Funkce **ZobrazZnakMpxDispl()** je volána v hlavní smyčce **while()**, při každém volání se testuje bit **bit_pozice** a je-li nastaven, nuluje se a inkrementuje se proměnná **count_pozice**, která tvoří programový čítač MOD4. Aktuální stav proměnné count_pozice určuje, která číslicovka 7-segmentového displeje LED je aktivní. Zobrazená čísla na jednotlivých řádech displeje jsou uložena v globálních proměnných j**ednotky**, **desítky**, **stovky** a **tisíce**, které generuje funkce **Prevod()**.

Funkce **Prevod(perioda_blik * 100)** je volána pouze při změně proměnné **perioda_blik**, pak neplýtvá zbytečně časem mikrokontroléru. Vstupní parametr **perioda_blik** je násoben konstantou 100, pak je čas svitu Led0 zobrazen na displeji v ms.

Vložený hlavičkový soubor **makra.h** je umístěn v adresáři, kde se nachází zdrojový text programu. Tento soubor byl použit v předcházejících příkladech, definuje makra pro práci s bity.

Pozn.: Použití funkce **Time100ms(unsigned char nasobek)** je jakýmsi kompromisem, dokud ještě neovládáme práci s vnitřními čítači/časovači. Těmito čítači/časovači se budeme zabývat v příští kapitole textu a po zvládnutí jejich teorie již tuto funkci používat nebudeme.

7.3.3 Externí přerušení INT0, INT1,INT2

Zadání:
Tlacitka S1, S2 a S3 připojená na interuptové vstupy INT0, INT1 a INT2 proti zemi budou inicializovat přerušení sestupnou hranou impulsu. Rutiny pro obsluhu přerušení budou negovat Led7 (INT0), Led6 (INT1) a Led5 (INT2).

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       ext_int_0_1_2
*   Soubor:        ext_int0_int1_int2.c
*
*   Obsah:         tlacitkem na interuptovem vstupu INT0 neguj svit LED7
*                  tlacitkem na interuptovem vstupu INT1 neguj svit LED6
*                  tlacitkem na interuptovem vstupu INT2 neguj svit LED5
*   Vytvoreno:     5.4.2013
*
*****************************************************************************/
/************  include files **********************************************/
#include <atmega32.h>
#include "makra.h"
```

```
/************ definice konstant ****************************************/
#define LEDKY PORTA

#define LED7 7
#define LED6 6
#define LED5 5
#define OFF 0xff
#define ON 0x00

/************  prototypy lokalnich funkci *******************************/
void main(void);
void PortsInit(void);
void ExtIntInit(void);

/************  kod  *****************************************************/
/************** funkce nastaveni orientace portu  ***********************/
void PortsInit(void)
{
   DDRA = 0xff;                         //port A je vystup

   CLRBIT(DDRD,DDD2);                   //nastaveni pinu INT0 jako vstup (lze i jako vystup)
   CLRBIT(DDRD,DDD3);                   //nastaveni pinu INT1 jako vstup (lze i jako vystup)
   CLRBIT(DDRB,DDB2);                   //nastaveni pinu INT2 jako vstup (lze i jako vystup)
   SETBIT(PORTD,PORTD2);                //pull-up na pin INT0 (nutne)
   SETBIT(PORTD,PORTD3);                //pull-up na pin INT1 (nutne)
   SETBIT(PORTB,PORTB2);                //pull-up na pin INT2 (nutne)
}
/************** nastaveni prerusovaciho systemu *************************/
void ExtIntInit(void)
{
   SETBIT(GICR,INT0);           //nastaveni preruseni od INT0
   SETBIT(GICR,INT1);           //nastaveni preruseni od INT1
   SETBIT(GICR,INT2);           //nastaveni preruseni od INT2
   SETBIT(MCUCR,ISC01);         //preruseni od INT0 na sestupnou hranu
   SETBIT(MCUCR,ISC11);         //preruseni od INT1 na sestupnou hranu
   CLRBIT(MCUCSR,ISC2);         //preruseni od INT2 na sestupnou hranu
   SETBIT(SREG,7);              //globalni povoleni preruseni
}

/*************** interruptova rutina INT0 *******************************/
//neguj LED7
void ext_inter_0(void)__interrupt[INT0_vect]
{
  NEGBIT(LEDKY,LED7);
}

///*************** interruptova rutina INT1 *******************************/
//neguj LED6
void ext_inter_1(void)__interrupt[INT1_vect]
{
   NEGBIT(LEDKY,LED6);
}

///*************** interruptova rutina INT2 *******************************/
```

```
//neguj LED5
void ext_inter_2(void)__interrupt[INT2_vect]
{
  NEGBIT(LEDKY,LED5);
}

/****************  main funkce ******************************************/
void main(void)
{
  PortsInit();
  ExtIntInit();

  LEDKY = OFF;

  while(1)
  {
    //zde je misto pro hlavni program
  }
}
```

Popis programu:

Tento program je modifikací předchozích dvou ukázkových programů, využívá všechny tři externí vstupy přerušení. Zajímavostí je, že piny pro interruptové vstupy mohou být orientovány jako vstupní i výstupní, pull-up rezistory jsou však nutné. Smyčka **while()** je prázdná, může však obsahovat libovolný program.

7.3.4 Úlohy k samostatnému řešení

7.3.4.1: Rotace LED

Program bude modifikací příkladu 7.1.4. Rotace LED budou ovládány interruptovými vstupy takto:

INT0 - rotace LED o 1 bit doprava, po zhasnutí LED0 se rozsvítí LED7
INT1 - rotace LED o 1 bit doleva, po zhasnutí LED7 se rozsvítí LED0

7.3.4.2.: Rotace1 INT0 INT1

Program bude sledovat stav externích interuptových vstupů INT0 (ovládán spinačem S1) a INT1 (ovládán spinačem S2).

Po aktivaci INT0 se spustí rotace LED počínaje LED0 doleva s periodou 1s (časování časovou smyčkou)

Rotování LED se zastaví po 5 cyklech nebo po aktivaci INT1, poté LED zhasnou.

Pozn.: Program je modifikací úlohy 7.1.6.1. Rotace1.

7.3.4.3: Rotace2 INT0 INT1

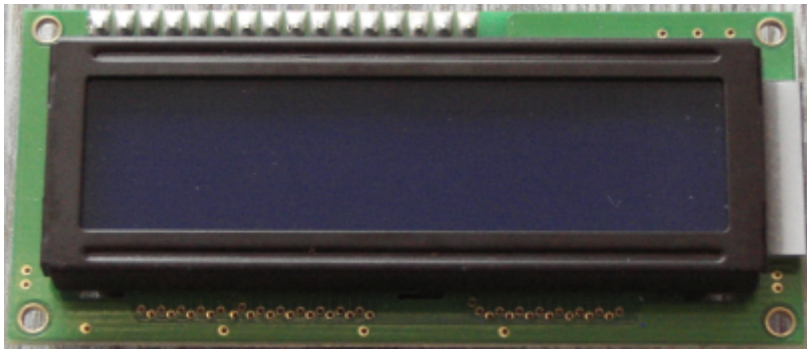
Program bude sledovat stav externích interuptových vstupů INT0 (ovládán spinačem S1) a INT1 (ovládán spinačem S2).

Po každé 2. aktivaci INT0 se provede rotace LED7 o 1 bit doprava dokud LED nezhasne.

Po každé 2. aktivaci INT1 se provede rotace LED0 o 1 bit doleva dokud LED nezhasne.

S3 – rozsvití LED7
S4 – rozsvití LED0
S5 – zhasnou rozsvícené LED

7.4 Obsluha LCD displeje



Tato kapitola se bude zabývat použitím dvouřádkového alfanumerického displeje LCD (**L**iquid **C**rystal **D**isplay). V našich příkladech budeme používat displej typu 1602A, umožňuje zobrazení dvou řádků po šestnácti znacích. Tento displej využívá řadič typu HD44780 (Hitachi). Datasheet tohoto řadiče je umístěn na webových stránkách projektu v položce Datasheety.

7.4.1 Paměti LCD displeje

LCD displej obsahuje dva typy pamětí, DD RAM a CG RAM.

DD RAM obsahuje znaky, které se zobrazují na displeji. Každému zapsanému údaji odpovídá jeden zobrazený znak podle tabulky Tab.4. DD RAM se adresuje 7bitovou adresou. Pro dvouřádkový displej se 16 znaky na jeden řádek jsou pro první řádek platné adresy 0x00 až 0x0F, pro druhý řádek adresy 0x40 až 0x4F, viz Tab.3.

Ve skutečnosti obsahuje paměť DD RAM 40 znaků, je možné však zobrazit pouze 16 znaků, posunem displeje doleva či doprava se zobrazí postupně všech 40 znaků.

Pozice znaku	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adresa DD RAM	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Tab.3: Adresy znaků na LCD dvouřádkovém displeji.

CG RAM slouží pro uložení uživatelských znaků. Uživatel může vytvořit vlastní sadu 8 znaků, např. pro grafické znaky nebo pro písmena s českou diakritikou. Kódy těchto znaků jsou 0x00 až 0x07. CG RAM se adresuje 6bitovou adresou. Každý znak je definován osmi po sobě jdoucími buňkami, rozměr znaku je 5x7 bodů. Horní tři bity nejsou použity, zbylých pět bitů je použito pro definování podřádku. Těchto podřádků může být 8, spodní podřádek se obvykle nepoužívá, využívá se k zobrazení kurzoru. Příklad vytvoření tří znaků (šipka nahoru, šipka dolů a srdíčko) je uveden v Tab.5.

Tab.4: Adresy znaků v DD RAM

Kód znaku (DDRAM data)									CGRAM adresa							Předloha znaku (CGRAM data)								Znak číslo	Hex kód podřádku
7	6	5	4	3	2	1	0		5	4	3	2	1	0		7	6	5	4	3	2	1	0		
												0	0	0		*	*	*	0	0	1	0	0		0x04
												0	0	1		*	*	*	0	1	1	1	0		0x0e
												0	1	0		*	*	*	1	0	1	0	1		0x15
0	0	0	0	*	0	0	0		0	0	0	0	1	1		*	*	*	0	0	1	0	0	1	0x04
												1	0	0		*	*	*	0	0	1	0	0		0x04
												1	0	1		*	*	*	0	0	1	0	0		0x04
												1	1	0		*	*	*	0	0	1	0	0		0x04
												1	1	1		*	*	*	0	0	0	0	0		0x00
												0	0	0		*	*	*	0	0	1	0	0		0x04
												0	0	1		*	*	*	0	0	1	0	0		0x04
												0	1	0		*	*	*	0	0	1	0	0		0x04
0	0	0	0	*	0	0	1		0	0	1	0	1	1		*	*	*	0	0	1	0	0	2	0x04
												1	0	0		*	*	*	1	0	1	0	1		0x15
												1	0	1		*	*	*	0	1	1	1	0		0x0e
												1	1	0		*	*	*	0	0	1	0	0		0x04
												1	1	1		*	*	*	0	0	0	0	0		0x00
												0	0	0		*	*	*	0	1	0	1	0		0x0a
												0	0	1		*	*	*	1	0	1	0	1		0x15
												0	1	0		*	*	*	1	0	0	0	1		0x11
0	0	0	0	*	1	1	1		1	1	1	0	1	1		*	*	*	1	0	0	0	1	8	0x11
												1	0	0		*	*	*		1	0	1			0x0a
												1	0	1		*	*	*			1				0x04
												1	1	0		*	*	*			1				0x04
												1	1	1		*	*	*	0	0	0	0	0		0x00

Tab.5: Příklad sestavení uživatelských znaků

7.4.2 Rutiny pro ovládání LCD displeje

Obsluha LCD displeje je poměrně náročná, pokud budeme používat jednotlivé příkazy pro elementární činnosti (mazání displeje, určení pozice znaku, posun znaku, zápis znaku nebo příkazu, sekvence příkazů pro inicializaci displeje, …). Podrobný popis obsluhy je uveden v datasheetu řadiče HD4470.

V našich příkladech si obsluhu displeje zjednodušíme použitím několika funkcí. Tyto funkce budou umístěny v samostatném souboru **lcd.c**, který musí být součástí námi vytvářeného projektu. Jsou to tyto funkce:

- **LCDwriteCommand**(unsigned char b) – pošle příkaz do LCD
- **LCDwriteData**(unsigned char b) – pošle data do LCD
- **LCDclear**() – vymaže obsah LCD
- **LCDposition**(unsigned char x,unsigned char y) – nastaví pozici kurzoru na daný řádek (x) a sloupec (y)
- **LCDinit**() - inicializace displeje
- **LCDprintText**(const char* text) - vypíše textový řetězec
- **LCDprintDec**(unsigned char hodnota) - vypíše desítkovou číslici, vstupem je spodní nibl bytu "hodnota"
- **LCDprintTwoDec**(unsigned char hodnota) - vypíše 2mistné desítkové číslo, vstupem je hodnota bajtu "hodnota"
- **LCDprintByteToDec**(unsigned char hodnota, unsigned char plna_delka) - vypíše 3místné desítkové číslo, vstupem je hodnota bajtu "hodnota"
- **LCDcharDef**(void) - nahraje 8 vlastních znaků do CG RAM

V následujícím textu je výpis kódu funkcí pro ovládání displeje:

```
/*********************************************************************
*
*   Soubor:       lcd.c
*
*   Obsah:        Rutiny pro obsluhu LCD dvouradkoveho displeje
*
*   Vytvoreno:    26.11.2013
*
*********************************************************************/
#include <atmega32.h>
#include <inavr.h>
#include "makra.h"       //definice maker pro bitove operace
#include "lcd.h"         //pripojeni LCD a funkcni prototypy funkci pro
                         //obsluhu LCD

// definice prikazu

#define delay(us)   __delay_cycles(FOSC * us)   //makro pro casove zpozdeni
                        // FOSC = hodinovy kmitocet  CPU [MHz] defin. v lcd.h
#define FUNSET 0x28    //4bit komunikace,2 radky
#define LCDOFF 0x08    //vypnuti displeje
#define LCDON  0x0f    //zapnuti displeje,blikani kurzoru
#define CLEAR  0x01    //mazani displeje
#define ENTRY  0x06    //inkrementace

/****************** inicializace displeje *************************/
void LCDinit(void)
{
  unsigned int i;

  delay(4500);           //ceka 4.5ms
  delay(4500);           //ceka 4.5ms
  delay(4500);           //ceka 4.5ms

//1. prikaz nast. komunikace (0x30)
   LCD = ((0x30 >> 4) << 4) | (1 << EN);   // priprav prikaz pro odeslani
   CLRBIT(LCD,EN);                    //konec zapisu
   delay(4100);           //ceka 4.1ms

//2. prikaz nast. komunikace (0x30)
   LCD = ((0x30 >> 4) << 4) | (1 << EN);
   CLRBIT(LCD,EN);
   delay(120);            //ceka 120us

   LCD = ((0x30 >> 4) << 4) | (1 << EN);
   CLRBIT(LCD,EN);
   delay(40);            //ceka 40us

//4. 4bitova komunikace (0x20)
   LCD = ((0x20 >> 4) << 4) | (1 << EN);
   CLRBIT(LCD,EN);
```

```
  delay(40);              //ceka 40us

//5. nastaveni rezimu displeje (0x28)  dvouradkovej,5x8
  LCDwriteCommand(FUNSET);

//6. vymazani displeje (CLEAR 0x01) - vymaze LCD
  LCDwriteCommand(CLEAR);
  delay(1640);            //ceka 1640us

//7. zapne displej,blikajici kurzor(0x0f)
  delay(40);              //ceka 40us
  LCDwriteCommand(LCDON);

//8. entry mod,inkrementace (ENTRY 0x06)
  LCDwriteCommand(ENTRY);
  delay(40);            //ceka 40us
}

/****************** posle prikaz na LCD ****************************/
void LCDwriteCommand(unsigned char b)
{
 unsigned char i;
 unsigned char kopie = b;

 //horni 4 bity
 CLRBIT(LCD,RS);
 b = (((b >> 4) << 4) | (1 << EN)); //maskuje horni nibl a nahodi EN bit
 LCD = b;                           //posle na LCD
 CLRBIT(LCD,EN);                    //nuluje EN bit

 delay(40);
 //dolni 4 bity
 b = kopie;
 b =((b << 4) | (1 << EN));        //maskuje dolni nibl a nahodi EN bit
 LCD = b;                          //posle na LCD
 CLRBIT(LCD,EN);                   //nuluje EN bit
  delay(40);           //ceka 40us
}

/****************** posle znak na LCD ******************************/
void LCDwriteData(unsigned char b)
{
 unsigned char i;
 unsigned char kopie = b;

 //horni 4 bity
 SETBIT(LCD,RS);
 b = (((b >> 4) << 4) | (1 << EN) | (1 << RS)); //maskuje horni nibl a nahodi EN a RS bit
 LCD = b;                                  //posle na LCD
  delay(40);             //ceka 40us
 CLRBIT(LCD,EN);                           //nuluje EN bit

 //dolni 4 bity
 b = kopie;
 b =((b << 4) | (1 << EN) | (1 << RS));  //maskuje dolni nibl a nahodi EN bit
```

```
  LCD = b;                          //posle na LCD
  delay(40);                        //ceka 40us
  CLRBIT(LCD,EN);                   //nuluje EN bit
   delay(40);                       //ceka 40us
}

/******************* vymaze displej *********************************/
void LCDclear(void)
{
   unsigned int i;

   LCDwriteCommand(CLEAR);   //vymaze displej a nastavi pozici kurzoru na 0,0

   delay(1640);                      //ceka 1640us
}

/****************** nastaveni pozice kurzoru ***********************/
void LCDposition(unsigned char x,unsigned char y)
{
    if ((x >= 0) &&  (x <= 15))
    if ((y >= 0) &&  (y <= 1))  LCDwriteCommand(0x80 | x + y * 0x40);
}


/****************** vypise text ************************************/
void LCDprintText(const char* text)
{
  while(*text)  LCDwriteData(*text++);
}

/****************** vypise desitkovou cislici **********************/
//vstupem je spodni nibl bytu "hodnota"
void LCDprintDec(unsigned char hodnota)
{
  LCDwriteData(hodnota | 0x30);     //dle tabulky znaku pricti 0x30
}

/****************** vypise 2mistne desitkove cislo ****************/
//vstupem je hodnota bajtu "hodnota"
void LCDprintTwoDec(unsigned char hodnota)
{
  LCDprintDec(hodnota / 10);         //desitky
  LCDprintDec(hodnota % 10);       //jednotky
}

/******** vypise 3mistne desitkove cislo  **************************/
//vstupem je hodnota bajtu "hodnota"
void LCDprintByteToDec(unsigned char hodnota, unsigned char plna_delka)

//plna_delka = 1 ... zobrazeni nevyznamnych nul, = 0 ... potlaceni
{
  unsigned char Sto,Des,Jed;

  Sto = hodnota / 100;
```

```
 if (plna_delka)
    LCDwriteData(Sto + 0x30);           //0x30 se pricte viz tabulka znaku
 else                                   //potlaceni nevyznamne nuly na radu Sto
   {
    if (Sto != 0)
       LCDwriteData(Sto + 0x30);       //vypis cislici
    else
       LCDwriteData(' ');              //vypis prazdny znak
   }
 Sto *= 100;
 Des = (hodnota - Sto) / 10;

 if (plna_delka)
    LCDwriteData(Des + 0x30);
 else                                   //potlaceni nevyznamne nuly na radu Des
    {
     if ((Des == 0) && (Sto == 0))
        LCDwriteData(' ');
     else
        LCDwriteData(Des + 0x30);
    }
 Des *= 10;
 Jed = (hodnota - Sto - Des);
 LCDwriteData(Jed + 0x30);
}

/******** vypise 5mistne desitkove cislo  *************************/
//vstupem je hodnota wordu "hodnota"
void LCDprintWordToDec(unsigned int hodnota, unsigned char plna_delka)

//plna_delka = 1 ... zobrazeni nevyznamnych nul, = 0 ... potlaceni
{
   unsigned int DesTis = 0,Tis = 0,Sto = 0,Des = 0,Jed = 0;
   DesTis = hodnota / 10000;
   if (plna_delka)
      LCDwriteData(DesTis + 0x30);     //0x30 se pricte viz tabulka znaku
   else                                 //potlaceni nevyznamne nuly na radu Sto
     {
      if (DesTis != 0)
         LCDwriteData(DesTis + 0x30);  //vypis cislici
      else
         LCDwriteData(' ');            //vypis prazdny znak
     }
   DesTis *= 10000;

   Tis = (hodnota - DesTis) / 1000;
   if (plna_delka)
      LCDwriteData(Tis + 0x30);        //0x30 se pricte viz tabulka znaku
   else                                 //potlaceni nevyznamne nuly na radu Sto
     {
      if ((DesTis == 0) && (Tis == 0))
         LCDwriteData(' ');            //vypis prazdny znak
      else
        LCDwriteData(Tis + 0x30);      //vypis cislici
     }
```

```
   Tis *= 1000;

   Sto = (hodnota - DesTis - Tis) / 100;
   if (plna_delka)
      LCDwriteData(Sto + 0x30);        //0x30 se pricte viz tabulka znaku
   else                                //potlaceni nevyznamne nuly na radu Sto
     {
      if  ((DesTis == 0) && (Tis == 0) && (Sto == 0))
       LCDwriteData(' ');
      else
       LCDwriteData(Sto + 0x30);     //vypis cislici            //vypis prazdny znak
     }
   Sto *= 100;

   Des = (hodnota - DesTis - Tis - Sto) / 10;
   if (plna_delka)
      LCDwriteData(Des + 0x30);
   else                                //potlaceni nevyznamne nuly na radu Des
      {
        if ((DesTis == 0) && (Tis == 0) && (Sto == 0) && (Des == 0))
            LCDwriteData(' ');
        else
            LCDwriteData(Des + 0x30);
      }
   Des *= 10;

   Jed = (hodnota - DesTis - Tis - Sto - Des);
   LCDwriteData(Jed + 0x30);
}

/******************  nahrani vlastniho znaku do CG RAM ************/
void LCDcharDef(void)
{
  LCDwriteCommand(0x40);           //nastav adresu CG RAM
/*1.znak-sipka nahoru, adresa 0x00*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x0e);     //2.
  LCDwriteData(0x15);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x04);     //5.
  LCDwriteData(0x04);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*2.znak-sipka dolu, adresa 0x01*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*3.znak-srdicko, adresa 0x02*/
  LCDwriteData(0x0a);     //1.radek
  LCDwriteData(0x15);     //2.
```

```
  LCDwriteData(0x11);     //3.
  LCDwriteData(0x11);     //4.
  LCDwriteData(0x0a);     //5.
  LCDwriteData(0x04);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.radek
/*4.znak-sipka dolu, adresa 0x03*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*5.znak-sipka dolu, adresa 0x04*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*6.znak-sipka dolu, adresa 0x05*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*7.znak-sipka dolu, adresa 0x06*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
/*8.znak-sipka dolu, adresa 0x06*/
  LCDwriteData(0x04);     //1.radek
  LCDwriteData(0x04);     //2.
  LCDwriteData(0x04);     //3.
  LCDwriteData(0x04);     //4.
  LCDwriteData(0x15);     //5.
  LCDwriteData(0x0e);     //6.
  LCDwriteData(0x04);     //7.
  LCDwriteData(0x00);     //8.
}
```

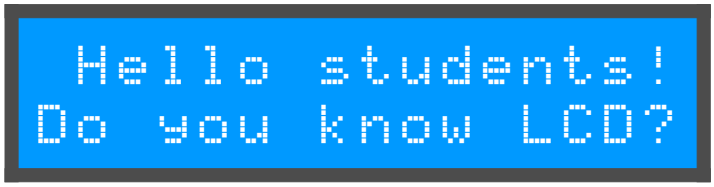
Jelikož funkce LCD je závislá na přesném časování, je nutné zohlednit použitý hodinový kmitočet mikrokontroléru. Časování LCD využívá makra **delay(us),** které

způsobí časové zpoždění v mikrosekundách, udané jako parametr makra. Je zde využito knihovní funkce **__delay_cycles(n)** námi používaného prostředí **CrossWorks.** Tato knihovní funkce způsobí zastavení vykonávání programu na daný počet **n** cyklů procesoru. Pro hodinový kmitočet 1MHz je doba cyklu 1us, při větším hodinovém kmitočtu je tato doba patřičně kratší. Z tohoto důvodu se musí parametr funkce **__delay_cycles(us)** násobit kmitočtem oscilátoru v jednotkách MHz. Toto je provedeno v definici makra násobením konstantou **FOSC**, jejíž hodnota se zadává v hlavičkovém souboru **lcd.h**. Tímto je dosažena nezávislost činnosti LCD na kmitočtu mikrokontroléru.

Podle potřeby je možné dodefinovat jiné speciální funkce, pro naše začátky s použitím LCD zatím postačují výše uvedené. Některé z těchto funkcí jsou použity v ukázkovém příkladu.

Zadání:

Po zapnutí se na LCD displeji vypíše na dobu přibližně 5s úvodní nápis :



Následuje nápis:

Po stisku S1 se vypíše text:

Po přibližně 5s se opakuje výzva :

Ukázkový program:

```
/*********************************************************************
*
*   Projekt:        displ_LCD
*   Soubor:       displ_LCD.c
*
*   Obsah:        uvodni osloveni, test stisku S1, potom dalsi text
*
*   Vytvoreno:   10.6.2013
*
*********************************************************************/

/************  include soubory ************************************/
#include <atmega32.h>
#include "makra.h"   //definice maker pro bitove operace
#include "spin.h"
#include "led.h"
#include "lcd.h"        //pripojeni LCD a funkcni prototypy funkci pro
                        //obsluhu LCD

/************  definice globalnich promennych *********************/
unsigned char BAJT1;   //pole 8 bitu

/************  definice konstant a maker **************************/
#define OFF 0xff
#define ON 0x00

/**************** nastaveni orientace portu ************************/
void PortsInit(void)
{
 DDRA = 0x00;  //port A je vstup
 DDRB = 0xFF;  //port B je vystup
 DDRD = 0xFF;  //port D je vystup
 PORTA = 0xFF; //pull-up pripojeny
 LEDKY = OFF;
}

/****************  main funkce ***********************************/
int main(void)
{
 PortsInit();           //nastaveni orientace portu

 LCDinit();             //inicializace LCD
 LCDwriteCommand(0x0c);  //vypni kurzor
 LCDclear();

 LCDposition(0,0);
 LCDprintText(" Hello students!");

 LCDposition(0,1);
 LCDprintText("Do you know LCD?");

 while(1)               //nekonecna smycka
```

```
 {
   static long i;

   i++;
   if (i == 100000)
   {
     i = 0;
     LCDclear();
     LCDposition(0,0);
     LCDprintText("    Press S1 !");
   }

   if (TESTNEGBIT(SPINACE,S1) && TESTBIT(BAJT1,S1)) //test stisku S1
   {
     CLRBIT(BAJT1,S1);
     LCDclear();
     LCDposition(0,0);
     LCDprintText(" Let's go learn!");
     LCDposition(0,1);
     LCDprintText("It's GREAT FUN !");
   }
   if (TESTBIT(SPINACE,S1)) SETBIT(BAJT1,S1);     //test uvolneni S1
 }
}
```

Popis programu:

V úvodu jsou vloženy „hlavičkové“ soubory **makra.h** (makra pro bitové operace), **spin.h** (definování portu pro připojení spínačů a přiřazení jednotlivých spínačů S1 až S5 k bitům tohoto portu), **led.h** (definování portu pro připojení LED a přiřazení jednotlivých indikátorů LED0 až LED7 k bitům tohoto portu) a **lcd.h** (definování portu pro připojení LCD, přiřazení řídících bitů LCD a seznam funkčních prototypů pro ovládání LCD):

```
/*****************************************************************************
*
*   Soubor:       makra.h
*
*   Obsah:        definice maker pro bitove operace
*
*   Vytvoreno:    26.5.2013
*
*****************************************************************************/
//makra pro bitove operace
#define  SETBIT(BIT_POLE,BIT) (BIT_POLE|=(1<<BIT))            //nahozeni bitu
#define  CLRBIT(BIT_POLE,BIT) (BIT_POLE &= ~(1<<BIT))        //nulovani bitu
#define  TESTBIT(BIT_POLE,BIT) (BIT_POLE & (1<<BIT))          //test nahozeni bitu
#define  TESTNEGBIT(BIT_POLE,BIT) (~BIT_POLE & (1<<BIT)) //test nuloveho bitu
#define  NEGBIT(BIT_POLE, BIT)  (BIT_POLE ^= (1<<(BIT)))      //negace bitu
```

```
/*****************************************************************************
*
*   Soubor:       spin.h
*
```

```
*   Obsah:        definice pripojeni spinacu
*
*   Vytvoreno:    26.5.2013
*
*******************************************************************************/
#define SPINACE PINA

//definice bitu spinacu
#define S5 0
#define S4 1
#define S3 2
#define S2 3
#define S1 4

//definice pomocnych bitu pro spinace
#define bS5 0
#define bS4 1
#define bS3 2
#define bS2 3
#define bS1 4
```

```
/*******************************************************************************
*
*   Soubor:       led.h
*
*   Obsah:        definice pripojeni indikatoru LED
*
*   Vytvoreno:    26.5.2013
*
*******************************************************************************/
#define LEDKY PORTB

//definice bitu LED
#define LED0 0
#define LED1 1
#define LED2 2
#define LED3 3
#define LED4 4
#define LED5 5
#define LED6 6
#define LED7 7
```

```
/*******************************************************************************
*
*   Soubor:       lcd.h
*
*   Obsah:        definice pripojeni LCD dvouradkoveho displeje
*                   prototypy funkci
*
*   Vytvoreno:    26.5.2013
*
*******************************************************************************/

#define LCD PORTD
```

```
#define RS  2       /* pripojeni bitu -R-egister -S-elect */
#define EN  3       /* pripojeni bitu -EN-able pin */

//definice kmitocu oscilatoru CPU
#define FOSC 16 //hodinovy kmitocet  CPU [MHz]

/********** funkcni prototypy  ************************************/
//inicializace displeje
void LCDinit(void);
//posle prikaz na LCD
void LCDwriteCommand(unsigned char b);
//posle znak na LCD
void LCDwriteData(unsigned char b);
//vymaze displej
void LCDclear(void);
//nastaveni pozice kurzoru
void LCDposition(unsigned char x,unsigned char y);
//vypise textovy retezec
void LCDprintText(const char* text);
//vypise desitkovou cislici, vstupem je spodni nibl bytu "hodnota"
void LCDprintDec(unsigned char hodnota);
//vypise 2mistne desitkove cislo, vstupem je hodnota bajtu "hodnota"
void LCDprintTwoDec(unsigned char hodnota);
//vypise 3mistne desitkove cislo, vstupem je hodnota bajtu "hodnota"
void LCDprintByteToDec(unsigned char hodnota, unsigned char plna_delka);
//vypise 5mistne desitkove cislo, vstupem je hodnota wordu "hodnota"
void LCDprintWordToDec(unsigned int hodnota, unsigned char plna_delka);
//nahraje 8 vlastnich znaku do CG RAM
void LCDcharDef(void);

/*Nektere uzitecne prikazy:

posuv displeje L ... 0x18
posuv displeje R ... 0x1C
posuv kurzoru L ... 0x10
posuv kurzoru R ... 0x14
kurzor a displej domu ... 0x02
*/
```

Před použitím LCD displeje je **nutné** provézt jeho inicializaci funkcí **LCDinit()**, vypnutí kurzoru zapsáním příkazu s kódem **0x0c** (viz tabulka příkazů v datasheetu řadiče HD4477) použitím funkce **LCDwriteCommand(0x0c)** a vymazání displeje funkcí **LCD clear()**.

Před zapsáním úvodního textu „**Hello students!**“ se nastaví pozice začátku textu na 0. sloupec a 0. řádek displeje funkcí LCDposition(0,0), podobně před zápisem textu „**Do you know LCD?**“ nastavíme pozici prvního znaku na 0. sloupec a 1. řádek displeje.

V hlavní programové smyčce je časovač způsobující zpoždění přibližně 5s, poté se po opětovném vymazání obsahu displeje a nastavení pozice začátku textu vypíše text „ **Press S1!** “.

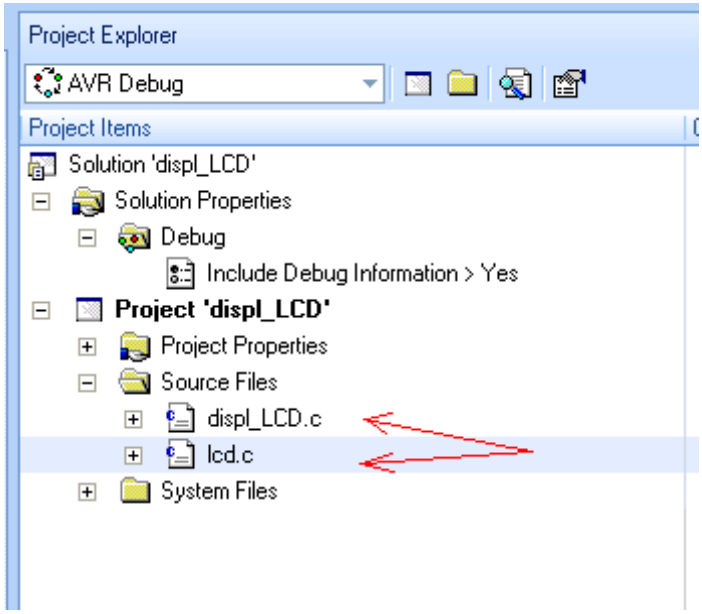
Po stisku S1 se vypíše text na 0.řádku „**Let's go learn!**“ a na 1. řádku text

„**It's GREAT FUN !**“.

Výzva „ **Press S1** „ se po 5s opakuje.

Projekt ukázkového programu obsahuje dva *.c soubory : **displ_LCD.c** a **lcd.c**. Funkce main() smí být obsažena **pouze v jednom** z těchto souborů, v našem projektu v souboru displ_LCD.c !

Do projektu vložíme nový soubor takto: pravým tlačítkem myši poklepeme na položku **Source Files** a z nabídky vybereme položku **Add New File** .



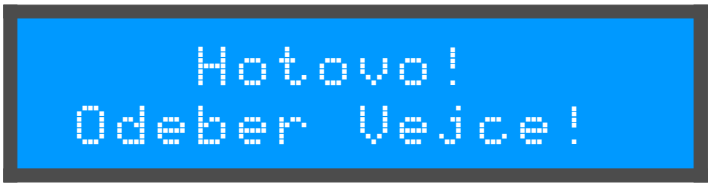
Toto je běžná praxe u větších projektů, vede k větší přehlednosti projektu, umožňuje týmovou spolupráci. Ke každému ze souborů *.c by měl existovat hlavičkový soubor se stejným jménem, který obsahuje příslušné definice, makra popř. prototypy funkcí, související s tímto souborem.

7.4.3 Úlohy k samostatnému řešení

7.4.3.1: Vajíčkový časovač LCD

Po zapnutí programu se zobrazí na LCD displeji nápis:

Po stisku S5 se začne čas na displeji po 1s odečítat, po uplynutí času bude na displeji 10x blikat nápis:



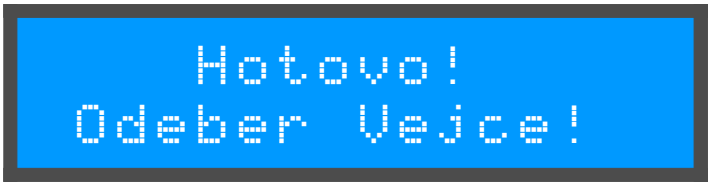
Potom se bude celý cyklus opakovat.

Pozn.: Tato úloha je modifikací úlohy 7.2.6.1.

7.4.3.2: Vajíčkový časovač LCD plus

Po zapnutí programu se zobrazí na LCD displeji nápis:

Po stisku S5 se začne čas na displeji po 1s odečítat, po uplynutí času bude na displeji 10x blikat nápis:



Potom se bude celý cyklus opakovat.

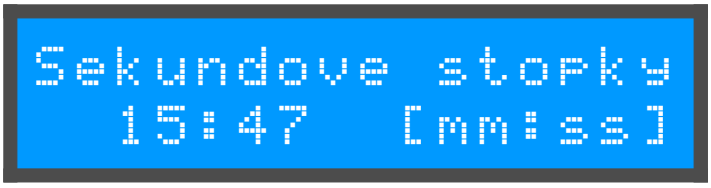
Automat umožňuje nastavení času spínači S1 a S2 takto:

- S1: nastavení minut v rozsahu 1 až 9 cyklicky
- S2: nastavení sekund v rozsahu 0 až 50 po 10s cyklicky

Pozn.: Tato úloha je modifikací úlohy 7.4.3.1.

7.4.3.3: Sekundové stopky LCD

Po startu programu se zobrazí na displeji nápis:



Ovládání stopek:

- S1 – start
- S2 – zamrznutí displeje, opětovným stiskem S2 normální zobrazení času
- S3 – stop
- S4 – nulování

Pozn.: Tato úloha je modifikací úlohy 7.2.6.2.

7.4.3.4: Nápis HALO SOUE LCD

Po zapnutí programu se zobrazí na LCD displeji nápis:

Po stisku S1 10x blikne na displeji nápis:

a na displeji bude vypsán znovu úvodní text. Vše se opakuje.

7.4.3.5: Běžící text LCD

Po stisku S1 bude v 1.řádku LCD displeje rolovat dokola směrem doleva text: „HALO SOUE! JAK SE DNES MATE ? “. Před každým slovem textu jsou dvě mezery. Rychlost rolování přibližně 0,3 s.

Po stisku S2 se běžící text zastaví a vše se opakuje po opětovném stisku S1.

7.4.3.6: Násobilka 5 LCD

Po zapnutí programu se bude na LCD displeji po 0,5 s zobrazovat malá násobilka 5 od 1 do 100 takto:

Pozn.: Tato úloha je modifikací úlohy 7.2.6.5.

7.4.3.7: Generátor pseudonáhodného čísla LCD

Po každém stisku S1 se na LCD displeji zobrazí aktuální stav volně běžícího čítače MOD 100, kde zůstane až do dalšího stisku S1. Zobrazení na displeji:

Pozn.: Tato úloha je modifikací úlohy 7.2.6.6.

7.4.3.8: Odhad času v sekundách LCD

Po stisku S1 odstartujte měření času v sekundách, po uvolnění S1 se na dobu 3s zobrazí na LCD displeji údaj skutečného času do 99 s. Po opětovném stisku začíná nový odpočet času. Zkalibrujte časovou smyčku pro maximální odchylku 1s za dobu 1 minuty. Zobrazení na displeji:

Pozn.: Tato úloha je modifikací úlohy 7.2.6.10.

7.5 Čítač / časovač 0

Čítače a časovače umožňují přesně časovat běh nějaké události, počítat vnější impulsy, generovat, dělit a měřit kmitočet, generovat signál PWM (pulsně šířková modulace) pro fázové řízení a jiné činnosti.

Atmega32 obsahuje tři čítače/časovače, které mají částečně jiné možnosti a budou postupně v dalším textu vysvětleny. Rozdíl mezi čítačem a časovačem je následující:

Čítač je obvod, který počítá impulsy vnějšího signálu přivedeného na vstupní piny mikrokontroléru. Po načítání určitého počtu impulsů se provede uživatelem definovaná činnost, např. zastavení pohybu dopravníku po napočtení určitého počtu výrobků.

Časovač čítá pevný kmitočet odvozený od hodinového kmitočtu mikrokontroléru. Napočítáním určitého počtu impulsů je provedeno určení přesného časového úseku, např. po 1s se posune vteřinová ručka hodin.

7.5.1 Čítač / časovač 0 v módu CTC

Zadání:

S využitím Čítače/časovače 0 v módu CTC generujte signál na výstupu OC0 (PB3) se střídou 1:1, s periodou 0,5 s. Na bit PD7 portu D bude připojena LED7, která bude blikat s periodou 10s. Kopie S5 (PA0) na LED0 (PD0).

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:      timer_0
*   Soubor:     timer0_ctc.c
*   Obsah:      -generovani signalu se stridou 1:1 s periodou 0.5s na
*                 vystupu OC0 (PB3) signalizovane LED3
*                -blikani LED7 (PD7) se stridou 1:1 s periodou 10s
*                -zdroj taktovaciho kmitoctu je vnitrní kalibrovany
*                 RC oscilátor 1MHz
*                -kopie S5 (PA0) na LED0 (PD0)
*
*   Vytvoreno:  5/4/2013
*
*****************************************************************************/
/************  include soubory ************************************************/
#include <atmega32.h>
#include "makra.h"

/************ definice konstant ***********************************************/
#define SPINACE PINA    //na portA pripojeny spinace
#define LEDKY PORTD     //na portB pripojeny LED
```

```
#define OFF 0xff
#define LED7 7
#define LED0 0
#define S5  0
/************  prototypy lokalnich funkci  *********************************/
void main(void);
void PortInit(void);
void Timer0Init(void);

/**************  kod  ****************************************************/
//inicializace portu
void PortInit(void)
{
  DDRA = 0x00;          //port A je vstup
  PORTA = 0xFF;         //pull-up pripojeny
  DDRD = 0xFF;          //port D je vystup
  SETBIT(DDRB,3);     //bit PB3 (OC0) je vystupni
}

//inicializace casovace 0 v modu ctc
void Timer0Init(void)
{
  TCCR0 = 0x1d;                 //ctc mod,preddelicka :1024,OC=toggle
  SETBIT(TIMSK,OCIE0);          //povoleni prerus pri shode TCNT0 = OCR0
  OCR0 = 0xf4;                  ///vrchol citace = 244,pri
                                //shode TCNT0 = OCR0 se citac nuluje
  SETBIT(SREG,7);               //globalni povoleni preruseni (bit I)
}

//interuptova rutina citace/casovace 0,vyvola se kdyz TCNT0 = OCR0
void Timer_0(void)__interrupt[TIMER0_COMP_vect]
{
 static unsigned char citac;      //citac poctu vstupu do interuptove rutiny pouzit
                                  //pro akce konane pri dosazeni nasobku casu 250 ms
 citac++;
 if (citac == 20)                  //20 x 250ms = 5s
 {
  citac = 0;
  NEGBIT(LEDKY,LED7);          //neguj LED7
 }
}

void main(void)
{
  PortInit();
  Timer0Init();
  OSCCAL = 0xb5;        //kalibracni bajt RC oscilatoru (presne dostaveni kmitoctu)
  LEDKY = OFF;          //po spusteni programu zhasni ledky

  while(1)
  {
//  hlavni program,zde cokoliv,pro ilustraci kopie spinace S5(PA0) na LED0(PD0)

    if(TESTNEGBIT(SPINACE,S5)) CLRBIT(LEDKY,LED0);     //aktivni S5 => rozsvit LED0
    else SETBIT(LEDKY,LED0);                           //jinak zhasni LED0
```

```
  }
}
```

Popis programu:

V režimu **CTC** se čítač vynuluje, když **TCNT0** dosáhne hodnoty **OCR0**. Toto chování je již v názvu módu CTC (Clear Timer on Compare Match - nuluj časovač při shodě s kompararačním registrem). Při shodě se nastaví příznak **OCF0** (Output Compare Flag 0). Je-li povoleno přerušení, vyvolá se.

Pro generování výstupního průběhu se střídou 1 :1 na vývodu **OC0** (PB3) je zvolen režim **toggle**, pak se při shodě obsahu registru čítače **TCNT0** s komparačním registrem **OCR0** vývod **OC0** neguje.

Pro generování signálu se střídou 1 : 1 a periodou 0,5s je potřeba odměřit časový úsek 0,25s. Je použit vnitřní kalibrovaný RC oscilátor o kmitočtu 1MHz, vydělený předděličkou čítače s dělicím poměrem 1:1024. Kmitočet za předděličkou bude mít periodu T = 1,024 ms. Z těchto pulsů vytvoříme dobu 0,25s tak, že zkrátíme cyklus čítače TCNT0. Hodnotu komparačního registru OCR0, při kterém čítač dosáhne svého vrcholu a vynuluje se, vypočteme takto:

0,25s / 0,001024s = 244,14 .

Konstantu 244 zapíšeme do OCR0 ve funkci **Timer0Init()**, kde se též konfiguruje režim čítače/časovače naplněním TCCR0 hodnotou 0x1d, povolí se přerušení při shodě TCNT0 = OCR0 nastavením bitu OCIE0 a též se povolí globální přerušení nastavením bitu **I** v registru SREG.

Kontrola požadovaného časového intervalu: 244 * 1,024 ms = 0,249856s. Určitá malá nepřesnost se příliš neprojeví, už z důvodu, že použitý vnitřní RC oscilátor není příliš přesný a teplotně stabilní, je od výrobce mikrokontroléru kalibrovaný při teplotě 25 ^{0}C. Kmitočet tohoto RC oscilátoru lze jemně dostavit změnou hodnoty kalibračního bajtu **OSCCAL** v teplotních podmínkách, ve kterých bude zařízení pracovat. Toto nastavení kalibračního bajtu je v ukázkovém programu použito. Při kalibrování provádíme měření periody, lze použít např. digitální osciloskop nebo logický analyzátor. Plně vyhoví i měřicí přístroje v provedení pro připojení k PC přes USB rozhraní. V dalším textu si ukážeme použití levného 8kanálového logického analyzátoru od firmy SALEASE, který je plně vyhovující a budeme jej v některých aplikacích používat.

Při shodě TCNT0 s OCR0 se současně vyvolá přerušení. V rutině pro obsluhu přerušení nazvané **Timer_0()__interrupt[TIMER0_COMP_vect]** je vytvořen programovým čítačem s proměnnou **citac** časový interval 5s, který nastane po 20násobném volání rutiny přerušení. Potom se proměnná čítač nuluje a neguje se stav LED7. Tato bliká s periodou 10s.

V hlavní smyčce **while(1)** může být umístěna libovolná posloupnost příkazů. V uvedeném příkladu je pro názornost ukázky nezávislosti chodu hlavní programové smyčky a událostí vyvolané přerušením pouze jediný podmíněný příkaz, který po dobu stisku S5 rozsvěcuje LED0, přičemž LED7 bliká nezávisle s periodou 10s.

Pozn.: **Pin OC0 (PB3) musí být nastaven jako výstupní, jinak se výstupní signál na tomto výstupu neobjeví !**

Pro snadné sestavení konfiguračního bajtu čítače/časovače 0, který se zapíše do registru TCCR0 poslouží následující tabulka Tab.6. Do prázdných políček bajtu se zapíší patřičné hodnoty 0/1 podle volby režimu, chování bitu OC0 a požadovaného dělicího poměru předděličky.

0		0	normální režim
0		1	CTC režim
1		0	fázově korigovaný PWM režim
1		1	rychlý PWM režim

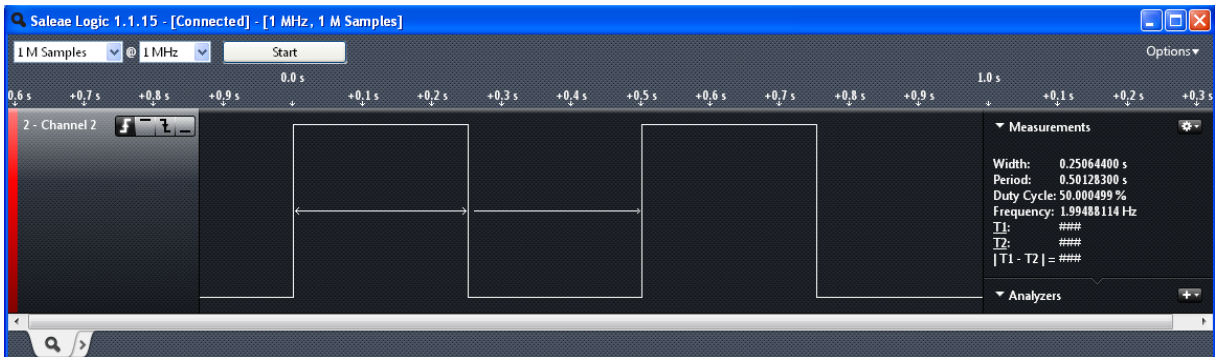
	0							
bit	7	6	5	4	3	2	1	0
TCCR 0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

0	0	OC0 odpojen
0	1	negace OC0 při shodě
1	0	nulování OC0 přishodě
1	1	nastavení OC0 při shodě

C/T0 zastaven	0	0	0
předdělička :1	0	0	1
předdělička : 8	0	1	0
předdělička : 64	0	1	1
předdělička : 256	1	0	0
předdělička : 1024	1	0	1

Tab.6: Sestavení konfiguračního bajtu pro nastavení činnosti C/T 0

Na obrázku Obr.16 je záznam výstupního kmitočtu na výstupu OC0, pořízený logickým analyzátorem značky SELEAE, připojeného pomocí USB rozhraní k PC. V pravé části obrázku jsou hodnoty šířky pulsu (Width: 0.25064400) a doby periody (Period: 0.50000499).



Obr.16: Záznam výstupního kmitočtu na výstupu OC0

7.5.2 Čítač / časovač 0 v módu rychlého PWM

Zadání:

S využitím Čítače/časovače 0 v módu rychlého PWM generujte PWM signál na výstupu OC0 (PB3) s periodou 256 µs. Na bit PB3 bude připojena LED0, která bude měnit svůj jas od zhasnutí po maximální svit.

Ovládání: S5 … inkrementace PWM s autorepeatem
S4 … dekrementace PWM s autorepeatem

Signálem PWM proveďte změnu jasu LED0 event. změnu otáček stejnosměrného motoru, připojeného přes výstupní zesilovač k externímu napájecímu zdroji.

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:        timer0_PWM_fast
*   Soubor:         timer0_PWM.c
*   Obsah:         -generovani PWM signalu na vystupu OC0 (PB3) signalizovane LED3
*                  -sirka PWM se zadava spinacem S5 (inkrement) a S4 (dekrement)
*                  -spinace maji funkci autorepeat
*                  -zdroj taktovaciho kmitoctu je vnitrní kalibrovany
*                   RC oscilátor 1MHz
*
*   Vytvoreno:  20/4/2013
*
*****************************************************************************/
/************  include soubory ************************************************/
#include <atmega32.h>
#include "makra.h"

/************ definice globalnich promennych *********************************/
unsigned char BAJT1;        //bajt pro bitove promenne
unsigned char sirkaPWM;   //inkrement/dekrement PWM (0-255)

/************ definice konstant **********************************************/
```

```
#define SPINACE PINA      //na portA pripojeny spinace
#define LEDKY PORTD       //na portB pripojeny LED
#define OFF 0xff
#define ON 0x00
#define LED7 7
#define S5  0
#define S4  1
#define bKeyRepeat 2

/*************  prototypy lokalnich funkci  ********************************/
void main(void);
void PortInit(void);
void Timer0Init(void);

/*************  kod  ******************************************************/
//inicializace portu
void PortInit(void)
{
  DDRA = 0x00;          //port A je vstup
  PORTA = 0xFF;         //pull-up pripojeny
  DDRD = 0xFF;          //port D je vystup
  SETBIT(DDRB,3);     //bit PB3 (OC0) je vystupni
}

//inicializace casovace 0 v modu ctc
void Timer0Init(void)
{
  TCCR0 = 0x69;               //PWM mod,bez preddelicky,OC=down
  SETBIT(TIMSK,OCIE0);  //povoleni prerus pri shode TCNT0 = OCR0
  SETBIT(SREG,7);           //globalni povoleni preruseni (bit I)
}

//interuptova rutina citace/casovace 0,vyvola se kdyz TCNT0 = OCR0
void Timer_0(void)__interrupt[TIMER0_COMP_vect]
{
  static unsigned int i;    //i je pomocna promenna pro repeat funkci tlacitek

  i++;
  if(i == 50)                 //konst 50 urcuje rychlost zmeny PWM tlacitky
  {
    SETBIT(BAJT1,bKeyRepeat); //bKeyRepeat pouzit pro repeat funkci tlacitek
    i = 0;
  }
  OCR0 = sirkaPWM;      //zadani hodnoty PWM (0-255)
}
void main(void)
{
  PortInit();
  Timer0Init();
  OSCCAL = 0xb5;  //kalibracni bajt RC oscilatoru (presne dostaveni kmitoctu)
  OCR0 = 0;           //vychozi hodnota predvolby komparacniho registru

  while(1)
  {
    if(TESTNEGBIT(SPINACE,S5) && TESTBIT(BAJT1,bKeyRepeat))    //test stisku S5
```

```
   {
     NEGBIT(BAJT1,bKeyRepeat);
     if(sirkaPWM < 255) sirkaPWM+=1;   //inkrement hodnoty PWM
   }

   if(TESTNEGBIT(SPINACE,S4) && TESTBIT(BAJT1,bKeyRepeat))    //test stisku S4
   {
     NEGBIT(BAJT1,bKeyRepeat);
     if(sirkaPWM > 0) sirkaPWM-=1;     //dekrement hodnoty PWM
   }
  }
}
```

Tento režim poskytuje vysokorychlostní generování PWM průběhu. Čítač čítá od $00 do maxima ($FF) a po přetečení se vrací opět do $00. Narozdíl od fázově korigovaného režimu PWM (který používá dvoufázový průběh a bude použit v následujícím příkladu) poskytuje rychlý PWM režim 2x vyšší pracovní kmitočet.

V použitém neinvertujícím režimu je výstup OC0 nastaven do log.1 (začátek PWM impulsu) po přetečení obsahu čítače. Po dosažení shody TCNT0 = OCR0 výstup přejde do log.0 (konec PWM impulsu). Hodnota OCR0 přímoúměrně určuje dobu trvání log.1 na výstupu OCR0, což je vlastně šířka PWM impulsu. Tuto dobu můžeme měnit změnou obsahu registru OCR0. V ukázkovém programu se obsah OCR0 modifikuje při každém přetečení čítače v obslužné rutině přerušení proměnnou **sirkaPWM**.

Změna hodnoty této proměnné se provádí aktivací spínačů S4 (dekrementace) a S5 (inkrementace) v nekonečné programové smyčce **while(1)**. Zároveň s testem stisknutí tlačítek se testuje bit **bKeyRepeat**, který zajišťuje funkci autorepeatu spínačů. Inkrementace (dekrementace) proměnné sirkaPWM se provede při stisku a držení spinače až po nahození tohoto bitu. Tento bit se ihned nuluje a začíná nový odpočet času pro jeho nahození. Odpočet času se provádí počítáním počtu vstupů do rutiny pro obsluhu přerušení (proměnná i), při dosažení určitého počtu vstupů (zde 50) se bit bKeyRepeat znovu nahodí. Při konstantě 50 trvá změna PWM od 0 do 100 % přibližně 4s.

Při inkrementaci (dekrementaci) hodnoty proměnné sirkaPWM se testují její krajní meze 0 a 255, které nesmí být překročeny.

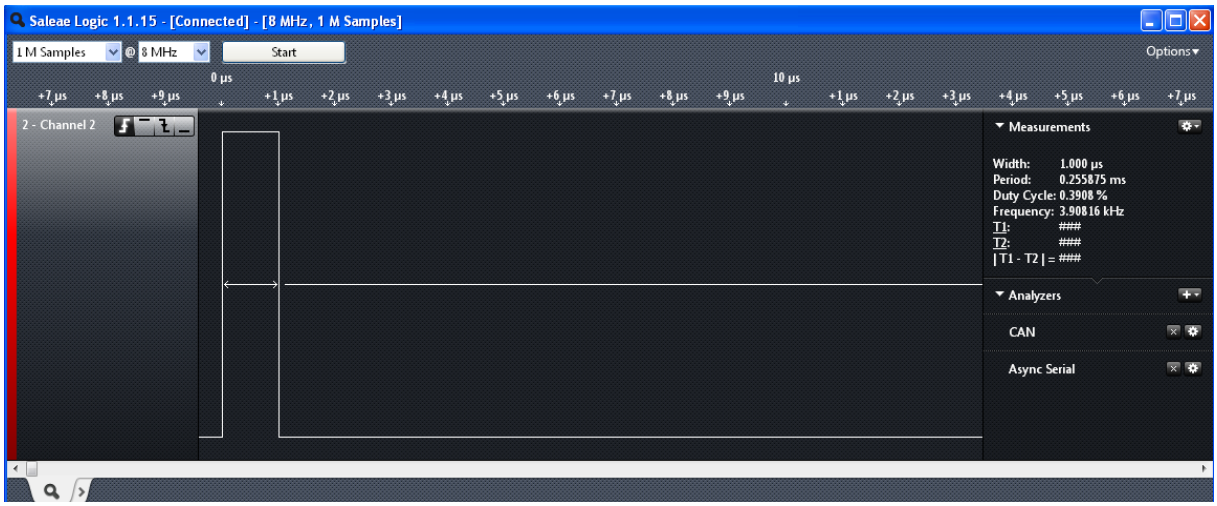
V ukázkovém příkladu je použit opět vnitřní kalibrovaný RC oscilátor 1MHz, bez předděličky. Při tomto kmitočtu bude výstupní kmitočet PWM signálu přibližně 3,9 kHz (1000000 / 256).

Při nastavení minimální šířky PWM proměnnou **sirkaPWM = 0** není, jak by se předpokládalo, výstupní impuls PWM nulový, na výstupu se objeví „jehlové“ impulsy o šířce odpovídajíci periodě hodinového kmitočtu, v našem přikladu o šířce T = 1 μs. Tento „nedostatek“ se neprojeví v režimu fázově korigovaného PWM.

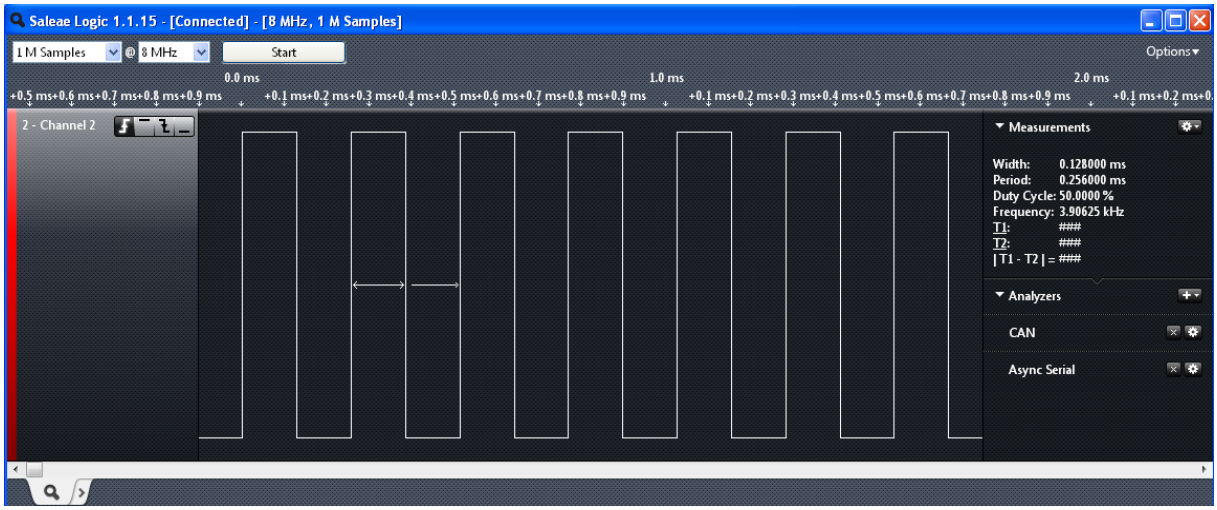
Při nastavení maximální šířky PWM proměnnou **sirkaPWM = 255,** bude na výstupu OC0 (PB3) trvale výstupní signál v log.1.

Na následujících obrázcích je patrný průběh signálu PWM při různých hodnotách proměnné **sirkaPWM**, průběhy jsou získány z PC logického anylyzátoru SALEALE. Průběh se šířkou PWM 100% nedokáže tento logický anylyzátor zobrazit, jelikož jej

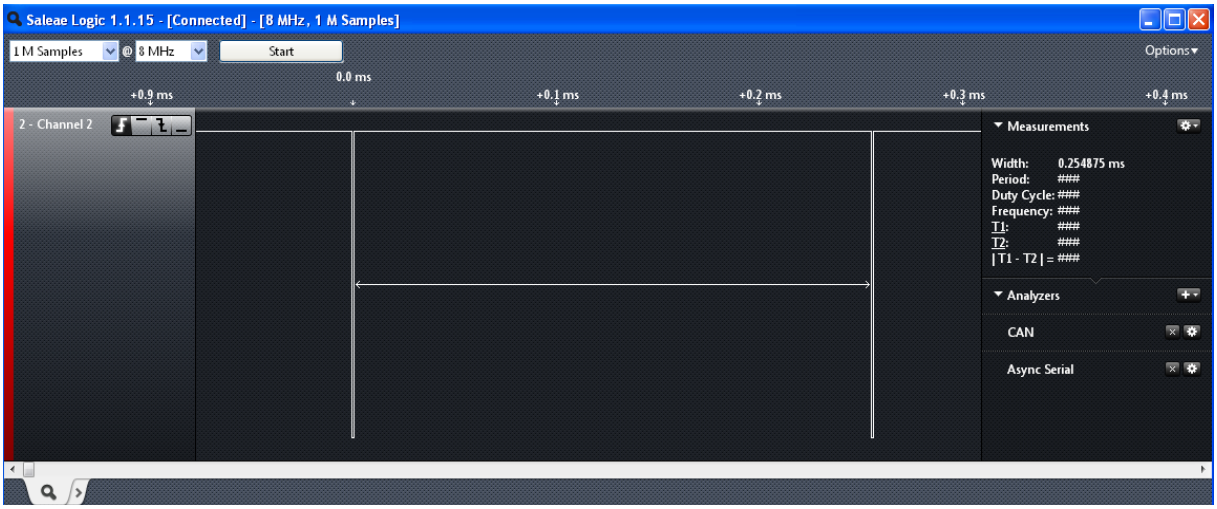
nedokáže zasynchronizovat, chybí náběžná čí sestupná hrana signálu, signál je trvale v log.1.



Obr.17: PWM signál 0%, šířka PWM 1µs, perioda PWM 256µs



Obr.18: PWM signál 50%, šířka PWM 125µs, perioda PWM 256µs



Obr.19: PWM signál 99%, šířka PWM 255µs, perioda PWM 256µs

7.5.3 Čítač / časovač 0 v módu fázově korigovaného PWM

Zadání:
S využitím Čítače/časovače 0 v módu fázově korigovaného PWM generujte PWM signál na výstupu OC0 (PB3) s periodou 512 µs. Na bit PB3 bude připojena LED0, která bude měnit svůj jas od zhasnutí po maximální svit.
Ovládání:
S5 … inkrementace PWM s aotorepeatem
S4 … dekrementace PWM s autorepeatem
S2 … PWM 100% okamžitě
S1 … PWM 0% okamžitě
LED displej bude zobrazovat velikost signálu PWM v rozsahu (0-255)
Signálem PWM proveďte změnu jasu LED0 event. změnu otáček stejnosměrného motoru, připojeného přes výstupní zesilovač k externímu napájecímu zdroji.

Ukázkový program:

```
/*****************************************************************************
*
*  Projekt:      timer0_PWM_displ_led
*  Soubor:     timer0_PWM_displ_led.c
*  Obsah:      -generovani fazove korigovaneho PWM signalu na vyst.OC0(PB3)
*                 -sirka PWM se zadava spinaci S5 (inkrement) a S4 (dekrement)
*                -S1...0% (OC0=0) PWM okamzite, S2...100% (OC0=255) PWM okamzite
*                -spinace maji funkci zrychlujiciho autorepeatu
*                -zdroj taktovaciho kmitoctu je vnitrní kalibrovany RC osc. 1MHz
*                -zobrazeni hodnoty PWM na LED displeji (0-255)
```

```
*
*   Vytvoreno:  2/5/2013
*
*******************************************************************************/
/************  include soubory ************************************************/
#include <atmega32.h>
#include "makra.h"
#include "dis_led.h"
#include "spin.h"

/************ definice globalnich promennych *********************************/
unsigned char BAJT1;                    //bajt pro bitove promenne
unsigned char zadanaPWM;                    // PWM (0-255)
unsigned char des,jed,sto,tis;
long int cislo;
unsigned int repeat = 1000;                    //rychlost autorepeatu inkr/dekr PWM tlacitky
static unsigned char keyCounter;    //pomocna promenna

/************ definice konstant **********************************************/
#define SPINACE PINA                    //na portA pripojeny spinace
#define KATODY PORTC                //katody cislicovek,pouzity bity 7,6,1 a 0
#define DISPLEJ PORTD

#define OFF 0xff
#define ON 0x00
#define LED7 7

//bity bajtu BAJT1
#define bKeyRepeat 5

/*************  prototypy lokalnich funkci  ********************************/
void main(void);
void portInit(void);
void timer0Init(void);
int prevod(long int cislo);
void zobrazZnakMpxDispl(void);
/**************  kod  ****************************************************/
```

```
//inicializace portu
void portInit(void)
{
  DDRA = 0x00;              //port A je vstup
  PORTA = 0xFF;       //pull-up pripojeny
  SETBIT(DDRB,3);     //bit PB3 (OC0) je vystupni
  DDRC = 0xc3;              //bity 0,1,6 a 7 vystupni
  DDRD = 0xFF;              //port D je vystup
}

//inicializace casovace 0 v modu ctc
void timer0Init(void)
{
//  TCCR0 = 0x69;                   //rychly PWM mod,preddelicka:1 ,OC=down,
  TCCR0 = 0x61;                   //fazove korigovany PWM mod,preddelicka:1 ,OC=down,
  SETBIT(TIMSK,OCIE0);    //povoleni prerus pri shode TCNT0 = OCR0
  SETBIT(SREG,7);           //globalni povoleni preruseni (bit I)
}

/**************** funkce pro prevod 4-mistneho cisla na jednotl. rady  *******/
int prevod(long int cislo)
{
   unsigned int pom;

   tis=cislo/1000;
   pom = cislo - (tis * 1000);
   sto = pom/100;
   pom = pom - (sto * 100);
   des = pom/10;
   jed = pom - (des * 10);
}

/**************** funkce zobrazeni znaku na MPX DISPLAYi *********************/
void zobrazZnakMpxDispl(void)
{
  static unsigned char pozice;
  pozice++;
```

```
  if(pozice == 4) pozice = 0;

  switch (pozice)
  {
  case 0: KATODY = OFF; DISPLEJ = znak[jed]; KATODY = DIS4;break;
  case 1: KATODY = OFF; DISPLEJ = znak[des]; KATODY = DIS3;break;
  case 2: KATODY = OFF; DISPLEJ = znak[sto]; KATODY = DIS2;break;
  //  case 3: KATODY = OFF; DISPLEJ = znak[tis]; KATODY = DIS1;break;
  case 3: KATODY = OFF;                    //cislicovka radu tisice zhasnuta,zde nepouzita
  }
}

//interuptova rutina citace/casovace 0,vyvola se kdyz TCNT0 = OCR0
void Timer_0(void)__interrupt[TIMER0_COMP_vect]
{

  static unsigned int i,j;              //i je pomocna promenna pro repeat funkci tlacitek
                                        //j je pomocna promenna pro obnovu MPX LED displeje
  OCR0 = zadanaPWM;                     //zadani hodnoty PWM (0-255), vrchol citace
  j++;
  if(j==10)
  {
    j = 0;
    zobrazZnakMpxDispl();               //po 10 volani interruptu inkrement MPX displeje
  }
  i++;
  if(i == repeat)                       //konst repeat urcuje rychlost zmeny PWM tlacitky
  {
    SETBIT(BAJT1,bKeyRepeat); //bKeyRepeat pouzit pro repeat funkci tlacitek
    i = 0;
  }
}

/*************** funkce main ************************************************/
void main(void)
{
  portInit();
```

```
timer0Init();
OSCCAL = 0xb4;            //kalibracni bajt RC oscilatoru (presne dostaveni kmitoctu)
OCR0 = 0;                 //vychozi hodnota predvolby komparacniho registru

while(1)
{
 if(TESTNEGBIT(SPINACE,S5) && (TESTBIT(BAJT1,bKeyRepeat)))    //test stisku S5
 {
  keyCounter++;
  if(keyCounter > 7) repeat = 50; //po 7 krocich(pribl.3[%]rozsahu PWM)rychly autorepeat
  else repeat = 1000;                                  //pomaly autorepeat

  NEGBIT(BAJT1,bKeyRepeat);
  if(zadanaPWM < 255) zadanaPWM+=1;          //inkrement hodnoty PWM
  prevod(zadanaPWM);
 // prevod(((zadanaPWM*39.2)/100)+1);          //udaj hodnoty PWM v [%]
 }

 if(TESTNEGBIT(SPINACE,S4) && TESTBIT(BAJT1,bKeyRepeat))    //test stisku S4
 {
  keyCounter++;
  if(keyCounter > 7) repeat = 50;
  else repeat = 1000;

  NEGBIT(BAJT1,bKeyRepeat);
  if(zadanaPWM > 0) zadanaPWM-=1;            //inkrement hodnoty PWM
  prevod(zadanaPWM);
 }
 if(SPINACE == OFF) keyCounter = 0;          //po uvolneni spinacu pomaly autorepeat

 if(TESTNEGBIT(SPINACE,S2))                  //test stisku S2
 {
  zadanaPWM = 255;                           //PWM 100[%]
  prevod(zadanaPWM);
 }

 if(TESTNEGBIT(SPINACE,S1))                  //test stisku S1
```

```
      {
       zadanaPWM = 0;                                  //PWM 0[%]
       prevod(zadanaPWM);
      }
     }
    }
```

Popis programu:

Program je modifikací předchozího programu. Navíc má tyto změny:

- zobrazuje na displeji LED hodnotu PWM v rozsahu 0 až 255
- spínače S4 a S5 mají zrychlený autorepeat (po 7 krocích pomalého autorepeatu se tento zrychlí, čímž je umožněno v počátku pomalé nastavení šířky PWM signálu, po 7 krocích je zapnuta rychlá změna šířky PWM)
- spínačem S1 se provede okamžitá změna PWM na 0
- spínačem S2 se provede okamžitá změna PWM na hodnotu 255

Funkce zrychleného autorepeatu se provádí modifikací proměnné **repeat** podle počtu kroků autorepeatu. Je-li počet kroků „pomalého“ autorepeatu větší než 7, sníží se hodnota proměnné repeat z 1000 na 50, čímž se 20x urychlí nahození bitu **bKeyRepeat** v interuptové rutině pro obsluhu čítače/časovače 0.

Při každém desátém volání této interuptové rutiny se též volá funkce **zobrazZnakMpxDispl()**, čímž je zaručena pravidelná obsluha 7segmentového displeje LED. Četnost obsluhy displeje LED je potřeba nastavit experimentálně, při příliš velké četnosti se zbytečně plýtvá strojním časem mikrokontroléru, při malé četnosti obsluhy displeje dochází k jeho blikání.

V hlavní smyčce se testuje aktivita spínačů S1, S2, S3 a S4. Při stisku kteréhokoliv z nich se mění hodnota proměnné **zadanaPWM**, která se zobrazuje na displeji LED. Pouze po každé změně této hodnoty se volá funkce **prevod(zadanaPWM)**, která vypočítává hodnoty jednotlivých řádů čísla zobrazujícího se na displeji. Tyto hodnoty se přenášejí do funkce pro zobrazení prostřednictvím globálních proměnných **jed**, **des**, **sto** a **tis**. Jelikož má v našem příkladu zobrazené číslo hodnotu maximálně 3místnou, je po celou dobu zobrazení číslice řádu tisíců číslicovka neaktivní (KATODY = OFF;).

Čítač/časovač v módu fázově korigovaného PWM má při zadání nulové šířky PWM trvale výstup v log.0, při maximální hodnotě PWM (255) v log.1. Vše je dobře patrné z následujících obrázků Obr.14 až Obr.17, na kterých je zachycena hodnota nastavené šířky signálu PWM na displeji LED a skutečná šířka signálu změřená osciloskopem.

7.5.4 Úlohy k samostatnému řešení

7.5.4.1: Generátor kmitočtu

Pomocí čítače/časovače 0 generujte na vývodu OC0 (PB3) signál se střídou 1 :1 o kmitočtu přibližně 100 Hz.

Pozn.: Použijte mód CTC a zvolte režim negování výstupu při shodě obsahu registrů TCNT0 a OCR0. Výstupní kmitočet změřte osciloskopem (popř. jeho vzorek logickým analyzátorem zn. SALEAE).

7.5.4.2: Generátor PWM

Pomocí čítače/časovače 0 generujte na vývodu OC0 (PB3) fázově korigovaný PWM signál, kterým budete měnit svit připojené LED0. Šířka PWM signálu (tím i svit LED0) bude odpovídat stisknutému spinači S1 až S5 takto:

nestisknuté žádné tlačítko = 0% => LED0 nesvítí

S1 ... 20%
S2 … 40%
S3 ... 60%
S4 ... 80%
S5 ... 100% => LED0 svítí naplno

Šířku výstupního PWM signálu ověřte osciloskopem (popř. jeho vzorek logickým analyzátorem zn. SALEAE).

7.5.4.3: Rotace1 CT0

Program bude číst stav spínačů na portu A, při sepnutých spínačích bude vykonávat následující akce:

Po stisku S5 se spustí rotace LED počínaje LED0 doleva s periodou 1s (časování čítačem CT0)

Rotování LED se zastaví po 5 cyklech nebo po stisku S4 a LED zhasnou.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.1: Rotace1.

7.5.4.4: Blikač1 CT0

Program bude číst stav spínačů na portu A, při sepnutých spínačích bude vykonávat následující akce:

Při současném stisku S5 a S4 rozblikej trvale LED0 s periodou 0,5s
Při současném stisku S1 a S2 rozblikej trvale LED7 s periodou 0,5s
Při stisku S3 nuluj rozsvícené LED

Použijte makra pro operaci s bitovými proměnnými. Pro časování intervalu 0,5s použijte čítač/časovač 0.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.3: Blikač1.

7.5.4.5: Blikač2 CT0

Program bude číst stav spínačů na portu A, při sepnutém spínači bude vykonávat následující akce:

S1 ... blikání LED0 s periodou 2 s
S2 ... blikání LED0 s periodou 1 s
S3 ... blikání LED0 s periodou 0,5 s
S4 ... střídavé blikání LED0 / LED1 s periodou 0,5 s
S5 ... střídavé blikání LED0 / LED1 s periodou 1 s

Pro časování příslušných intervalů použijte čítač/časovač 0.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.4: Blikač2.

7.5.4.6: Lékárnická míchačka CT0

Program bude číst stav spínačů na portu A, při sepnutém spínači Start (S1) zapne chod lékárnické míchačky. Míchačka má motor, který se otáčí 4s doprava (Led0), následuje 2s motor v klidu, poté 4s doleva (Led1), 2s klid a tento cyklus se opakuje až do stisku tlačítka Stop (S2). Po stisku tlačítka Cisteni (S3) se reverzuje motor s periodou 1s po dobu 5s.

Časování intervalů 1s, 2s a 4s realizujte pomocí čítače/časovače 0.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.8: Lékárnická míchačka.

7.5.4.7: Schodišťový automat CT0

Program bude číst stav spínačů na portu A, při sepnutém spínači S1 bude svítit světlo (Led0) po dobu 5s. Při dvojnásobném stisku v časovém intervalu 1s bude svítit světlo trvale. Při stisku delším než 2s světlo zhasne.

Časování intervalu 5s realizujte pomocí čítače/časovače 0.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.9: Schodišťový automat.

7.5.4.8: Postupný rozběh motoru CT0

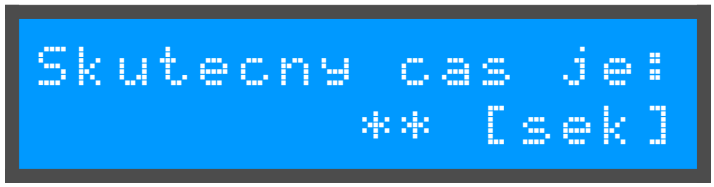
Program bude číst stav spínačů na portu A, při sepnutém spínači Start (S1) se zapne stykač K0 (Led0), dále vždy po 3s stykač K1 (Led1), K2 (Led2) a K3 (Led3). Po stisku tlačítka Stop (S2) se vypne ihned stykač K3, potom opět po 3s postupně stykače K2, K1 a K0.

Časování intervalu 3s realizujte pomocí čítače/časovače 0.

Pozn.: Tato úloha je modifikací úlohy 7.1.6.10: Postupný rozběh motoru.

7.5.4.9: Odhad času v sekundách LCD CT0

Po stisku S1 odstartujte měření času v sekundách, po uvolnění S1 se na dobu 3s zobrazí na LCD displeji údaj skutečného času do 99 s. Po opětovném stisku začíná nový odpočet času. K odečítání času použijte čítač/časovač 0 . Zobrazení na displeji bude vypadat následovně:



Pozn.: Tato úloha je modifikací úlohy 7.3.4.8

7.6 Čítač / časovač 1

7.6.1 Čítač/časovač 1 v normálním módu

Zadání: S využitím Čítače/časovače 1 v normálním módu vytvořte program, který bude blikat Led7 (PB7) s periodou 1Hz. Pro taktování použijte vnitřní kalibrovaný oscilátor 1MHz.

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:     timer1_nor
*   Soubor:      timer1_nor.c
*   Obsah:       Blikani LED7 na portu B s periodou priblizne 1s, zdroj
*                taktovaciho kmitoctu je vnitrní kalibrovany RC oscilator 1MHz
*
*   Vytvoreno:  17.11.2013
*
*****************************************************************************/
/************  include soubory ***************************************/
#include <atmega32.h>
#include "makra.h"
#include "led.h"

/************ definice konstant ***************************************/

#define SPINACE PINA    //na portA pripojeny spinace
#define LEDKY PORTB     //na portB pripojeny LED
#define OFF 0xff

/************  prototypy lokalnich funkci ****************************/
void main(void);
void PortsInit(void);
void Timer1Init(void);
void Timer_1(void);


/********** kod  ************************************************/
/********** nastaveni orientace portu ********************************/
void PortsInit(void)    //nastaveni orientace portu
{
  DDRA = 0x00;          //port A je vstup
  PORTA = 0xFF;         //pull-up pripojeny
  DDRB = 0xFF;          //port B je vystup
}

/********** inicializace citace 1 ************************************/
```

```
void Timer1Init(void)
{
   SETBIT(TIMSK,TOIE1);       //povoleni prerus od preteceni citace 1
   TCCR1A = 0x00;             //normal mod
   TCCR1B = 0x02;             //normal mod, :8, citac pretece po asi 0,52s
   //TCCR1B = 0x05;           //normal mod, :1024, citac pretece po asi 64s
}

/********** obsluzna rutina  pri preteceni TCNT1 **********************/
void Timer_1(void)__interrupt[TIMER1_OVF_vect]
{
  NEGBIT(LEDKY,LED7);       //po preteceni citace 1 neguje LED7
}

/********** main funkce ***********************************************/
void main(void)
{
 PortsInit();
 Timer1Init();
 SETBIT(SREG,7);            //globalni povoleni preruseni

 LEDKY = OFF;               //po spusteni programu zhasni ledky
 CLRBIT(LEDKY,LED7);        //zapni LED7 ihned po spusteni programu

 while(1)
 {
    // zde muze byt cokoliv
 }
}
```

Popis programu:

V příkladu je použit Čítač/časovač 1 v normálním režimu. Pro taktování je použit vnitřní kalibrovaný RC oscilátor 1MHz, který je vydělen (:8), perioda taktovacího kmitočtu bude :

T = 1/(1 000000 MHz / 8) = 8 µs .

Šestnáctibitový čítač přeteče za dobu :

65 5536 * 8 µs = 0,524 s.

Po přetečení se vyvolá obsluha přerušení, kde se změní stav Led7. Tato změna stavu Led7 je provedena pomocí makra NEGBIT().

7.6.2 Čítač/časovač 1 v CTC módu

Zadání: S využitím Čítače/časovače 1 v módu CTC vytvořte měřič kmitočtu. Vstupem měřeného kmitočtu bude vstup T1 (PB1). Periodu měřicího cyklu zvolte 1s a pro vytvoření této periody použijte Čítač/časovač 0. Zdrojem taktovacího kmitočtu bude krystalový oscilátor 16MHz. Změřený kmitočet zobrazte na LCD displeji takto:

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:     timer1_freq_meas
*   Soubor:      timer1_freq_meas.c
*   Obsah:       Meric kmitoctu, cita vnejsi události  na vstupu T1 (PB1)
*                oscilator 16MHz
*  zdroj taktovaciho kmitoctu je vnejsi krystalovy
*
*
*   Vytvoreno:  26/11/2013
*
*****************************************************************************/
/************  include soubory ***************************************/
#include <atmega32.h>
#include <inavr.h>
#include "makra.h"
#include "led.h"
#include "lcd.h"

/************ definice konstant ***************************************/
#define LEDKY PORTB       //na portB pripojeny LED
#define OFF 0xff
#define ON 0x00
```

```
#define START 0x07;
#define STOP 0x00;

/************  prototypy lokalnich funkci ****************************/
void main(void);
void PortsInit(void);
void Timer0Init(void);
void Timer1Init(void);
void Timer_0(void);

/********** kod  ***************************************************/

/********** nastaveni orientace portu ********************************/
void PortsInit(void)                    //nastaveni orientace portu
{
  DDRA = 0x00;                          //port A je vstup
  DDRB = 0x00;          //port B je vstup
  DDRD = 0xFF;          //port D je vystup

  PORTA = 0xFF;         //pull-up pripojeny
}

/********** inicializace citace 0 ***********************************/
// citac se pouziva pro odmereni mericiho intervalu 1.000 s
void Timer0Init(void)
{
  SETBIT(TIMSK,OCIE0);   //povoleni prerus pri shode TCNT0 = OCR0
  TCCR0 = 0x0b;                         //ctc mod, preddelicka :64
   OCR0 = 0xf9;                         //OC0 = 249, pak se citac nuluje
}

/********** inicializace citace 1 ***********************************/
void Timer1Init(void)
{
   TCCR1A = 0x00;           //NORMAL mod (WGM =0)
   TCCR1B = 0x00;           //NORMAL mod (WGM =0)
}
```

```
/********** obsluzna rutina  pri preteceni TCNT0 **********************/
//timer pretece kazdych 0,001s

void Timer_0(void)__interrupt[TIMER0_COMP_vect]
{
 static unsigned int citac_mod1000;     //citac mod1000

  SETBIT(PORTB,7);                      //pouzito pro ladeni
  CLRBIT(PORTB,7);                      //pouzito pro ladeni

 citac_mod1000 ++;
 if (citac_mod1000 >999)                //po 1s se provede nasledujici
 {
   TCCR1B = STOP;                       //STOP citace1
   citac_mod1000 = 0;                   //nulovani citace mod1000

   CLRBIT(LEDKY,LED0);                  //indikace vzorkovani mereni

   LCDposition(4,1);
   LCDprintWordToDec(((TCNT1H << 8) + TCNT1L),0);  //vypis hodnotu kmitoctu

   TCNT1H = 0;                          //nulovani citace1, nejprve horni bajt !!!
   TCNT1L = 0;                          //nulovani citace1, potom dolni bajt !!!

   TCCR1B = START;                      //START citace1

   SETBIT(LEDKY,LED0);                  //indikace vzorkovani mereni
 }
}

/********** main funkce ******************************************/
void main(void)
{
  PortsInit();
  Timer0Init();
  Timer1Init();
```

```
  SETBIT(SREG,7);                 //globalni povoleni preruseni

  LEDKY = OFF;                    //po spusteni programu zhasni ledky

  LCDinit();                      //inicializace LCD
  LCDwriteCommand(0x0c);          //vypni kurzor
  LCDclear();

// uvodni napis na displeji
  LCDposition(0,0);
  LCDprintText("Meric kmitoctu");
  LCDposition(0,1);
  LCDprintText("f = ");
  LCDposition(10,1);
  LCDprintText("[Hz]");

  Timer0Init();
  Timer1Init();

  while(1)
  {
    // zde muze byt cokoliv
  }
}
```

Popis programu:

V příkladu je použit Čítač/časovač 1 v režimu CTC. Vstupem měřiče kmitočtu je vstup T1 (PB1). Měření probíhá po dobu 1s, načtený počet impulsů odpovídá měřenému kmitočtu. Pro odměření měřicího intervalu 1,000 s je použit Čítač/časovač 0, který je řízen krystalovým oscilátorem 16 MHz. Tento kmitočet je vydělený předděličkou (:64), výsledná doba periody je tudíž 0,000004s. Když čítač dosáhne hodnoty 249 (čítač MOD 250), vynuluje se a vyvolá přerušení. V obslužné rutině přerušení Čítače/časovače 0 je realizován čítač MOD 1000 pro odměření periody měření 1,000s. Po dosažení 1s se provádějí následující činnosti:

- obsah 16bitového Čítače/časovače 1 se vypíše na LCD displeji jako hodnota naměřeného kmitočtu
- čítač/časovač 1 se vynuluje

- odstartuje se nová perioda měření

Šestnáctibitový Čítač/časovač 1 pracuje v normálním módu, maximální měřený kmitočet je tudíž dán maximálním obsahem čítače, což je 65535 Hz. Pokud by čítač přetekl před dosažením konce měřicího cyklu 1s, údaj na LCD zobrazovači by neodpovídal skutečnému měřenému kmitočtu. Pro měření vyšších kmitočtů bychom museli použít kratší periodu čítání, což se běžně používá.

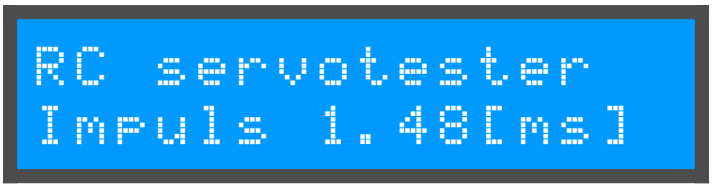
Měření kmitočtu se provádí pouze v obslužných rutinách přerušení obou čítačů/časovačů. V hlavní smyčce programu v našem školním příkladu neprovádí mikrokontrolér žádnou činnost.

7.6.3 Úlohy k samostatnému řešení

7.6.3.1: Servotester

S využitím čítačů/časovačů vytvořte generátor impulsů proměnné šířky v rozsahu od 0,80ms až 2,20ms. Impulsy se budou opakovat po 20ms. Změnu šířky impulsů proveďte spínači S1 (dekrementace) a S2 (inkrementace) v krocích po 0,01ms. Inkrementace a dekrementace bude provedena s autorepeatem v uvedeném rozsahu šířky impulsu. Spínačem S3 se nastaví pevná šířka výstupního impulsu 1,00ms, S4 šířka impulsu 1,50ms a S5 šířka impulsu 2,00ms.

Aktuální šířka výstupního impulsu se bude zobrazovat na LCD displeji takto:



Jako zdroj kmitočtu mikrokontroléru využijte krystalový oscilátor 16MHz. Výstupní signál generujte na portu B. Kontrolu funkce servotesteru ověřte digitálním osciloskopem.

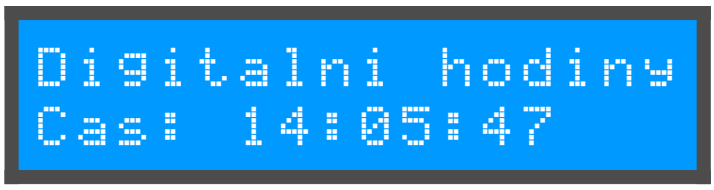
7.7 Čítač / časovač 2 (digitální hodiny)

7.7.1 Čítač/časovač 2 v CTC módu

Zadání:

S využitím Čítače/časovače 2 v módu CTC vytvořte digitální hodiny se zobrazením

času na LCD displeji. Zdrojem kmitočtu bude krystalový oscilátor 16MHz. Zobrazení časového údaje bude vypadat následovně:



Nastavení hodin spínačem S1, nastavení minut spínačem S2 a nulování hodin spínačem S5. Nastavení časového údaje bude mít funkci autorepeatu s opakováním 0,5 s.

Ukázkový program:

```
/*********************************************************************
*
*   Projekt:      citac_2
*   Soubor:       hodiny.c
*
*   Obsah:        digitalni hodiny, zdroj hodin kmitoctu krystalovy
*                 oscilator 16MHz s casovacem 2
*                 S1...nast.hod, S2...nast.min, S5...nulovani hodin
*
*   Vytvoreno:    20.9.2013
*
*********************************************************************/

/************  include soubory ************************************/
#include <atmega32.h>
#include "makra.h"       //definice maker pro bitove operace
#include "spin.h"        //pripojeni spinacu
#include "led.h"         //pripojeni LED indikatoru
#include "lcd.h"         //pripojeni LCD a funkcni prototypy funkci pro
                         //obsluhu LCD

/************  definice globalnich promennych *********************/
unsigned char BAJT1;               //pole 8 bitu
unsigned char hod,min,sek;         //citace hodin, minut a sekund
unsigned char KeyRepeatTime;  //perioda autorepeatu
/************ seznam lokalnich funkci **************************/
void PortsInit(void);
void Timer2Init(void);
void CountHourMinSek(void);
void ResetSetClock(void);
void RefreshLCD(void);


/************  definice konstant a maker **************************/
#define bTimeout1s    0                    //0.bit BAJT1..dosazeni 1sek
#define bKeyRepeatTimeout    1      //1.bit BAJT1..dosazeni casu autorepeatu spinacu
#define OFF 0xff
```

```
/*************** nastaveni orientace portu ***********************/
void PortsInit(void)
{
  DDRA = 0x00;          //port A je vstup
  DDRB = 0xFF;          //port B je vystup
  DDRD = 0xFF;          //port D je vystup
  PORTA = 0xFF;          //pull-up pripojeny
}

/*************** inicializace casovace 2 v modu ctc *********************************/
void Timer2Init(void) {
  TCCR2 = 0x0e;          //ctc mod,preddelicka :256 (16MHz/256 => T=16us)
  OCR2 = 0xf9;           //vrchol = 249,pri shode TCNT2 = OCR2 se TCNT2 nuluje (16us*250 =
4ms)
  SETBIT(TIMSK,OCIE2);  //povoleni prerus pri shode TCNT2 = OCR2
  SETBIT(SREG,7);          //globalni povoleni preruseni (bit I)
}

/*************** citac sekund,minut a hodin ****************************************/
void CountHourMinSek(void) {
  sek++;
  if (sek == 60){
    sek = 0;
    min++;
  }
  if (min == 60){
    min = 0;
    hod++;
  }
  if (hod == 24){
    hod = 0;
  }
}

/*********** nulovani citacu hodin a nastaveni aktualniho casu *********************/
void ResetSetClock(void) {

    static unsigned char RepeatCount;     //citac cyklu autorepeatu tlacitek pro
                                          //zrychleny autorepeat
    if (TESTNEGBIT(SPINACE,S5)) {      //nulovani citacu hodin
      sek = 0;
      min = 0;
      hod = 0;
    }
    if (TESTNEGBIT(SPINACE,S1) && TESTBIT(BAJT1,bKeyRepeatTimeout)) {        //nastaveni
hod
      CLRBIT(BAJT1,bKeyRepeatTimeout);

      RepeatCount ++;                                  //pocitej cykly autorepeatu
      if (RepeatCount > 5) KeyRepeatTime = 30;       //po 5 cyklech zrychli autorepeat

      hod++;
      if (hod > 23) hod = 0;
      LCDposition(5,1);
```

```
    LCDprintTwoDec(hod);
   }
   if (TESTNEGBIT(SPINACE,S2) && TESTBIT(BAJT1,bKeyRepeatTimeout)) {       //nastaveni
min
    CLRBIT(BAJT1,bKeyRepeatTimeout);

    RepeatCount ++;                                    //pocitej cykly autorepeatu
    if (RepeatCount > 5) KeyRepeatTime = 30;     //po 5 cyklech zrychli autorepeat

    min++;
    if (min > 59) min = 0;
    LCDposition(8,1);
    LCDprintTwoDec(min);
   }
   if (SPINACE == OFF) {
    KeyRepeatTime = 125;                               //nastav pocatecni rychlost autorepeatu
    RepeatCount = 0;
   }
}

/*************** obnova textu na LCD ************************************/
void RefreshLCD(void) {
   LCDposition(5,1);
   LCDprintTwoDec(hod);
   LCDposition(8,1);
   LCDprintTwoDec(min);
   LCDposition(11,1);
   LCDprintTwoDec(sek);
}

/**************** rutina obsluhy preruseni od  casovace 2 ****************************/
//timer2 v rezimu CTC pouzit pro vytvoreni sekundovych impulsu

void Timer_2(void)__interrupt[TIMER2_COMP_vect]
{
 static unsigned char InterruptCounter,KeyRepeatCounter;

 InterruptCounter++;         //citac poctu preruseni pro mereni casu 1 s
 KeyRepeatCounter++;     //citac pro autorepeat tlacitek

 if (KeyRepeatCounter > KeyRepeatTime) {   //KeyRepeatTime = 125...rychlost autorepeatu 0.5 s
  KeyRepeatCounter = 0;
  SETBIT(BAJT1,bKeyRepeatTimeout);
 }

 if (InterruptCounter > 249) {       //dosazeni 1 s (250 * 4ms = 1s)
  InterruptCounter = 0;
  NEGBIT(LEDKY,LED0);          //indikace sekundovych impulsu
  CountHourMinSek();              //inkrementace sec,min,hod
 }
}

/****************  main funkce ************************************/
int main(void)
{
```

```
 unsigned int   ScheduleTmr;  //planovac cetnosti opakovani akci ve smycce while
                                        //podle dulezitosti
 PortsInit();                 //nastaveni orientace portu
 Timer2Init();                //inicializace timeru 0
 LCDinit();                   //inicializace LCD
 LCDwriteCommand(0x0c);               //vypni kurzor
 LCDclear();                  //nulovani LCD

/************ uvodni napis na LCD **********************************/
 LCDposition(0,0);
 LCDprintText("Digitalni hodiny");
 LCDposition(0,1);
 LCDprintText("Cas:");
 LCDposition(7,1);
 LCDprintText(":");
 LCDposition(10,1);
 LCDprintText(":");

 while(1)
 {
  if( ScheduleTmr == 100 ) {    //nasledujici se provede pri
                                //kazdem 100. pruchodu smyckou while
    RefreshLCD();               //obnova obsahu LCD
    ResetSetClock();            //nulovani a nastaveni hodin
  }

  if( ScheduleTmr > 200 ) ScheduleTmr = 0;

  //zde budou funkce, ktere musi byt volany co nejcasteji

  ScheduleTmr++;
 }
}
```

Popis programu:

Pro získání přesného sekundového signálu 1,00s je použit Čítač/časovač 2, který je řízen krystalovým oscilátorem 16 MHz. Tento kmitočet je vydělený předděličkou (:256), výsledná doba periody vstupního kmitočtu je:

T = 1/(16 000 000 / 256) = 0,000016s (= 16µs).

Tento signál se čítá 8bitovým čítačem. Při shodě obsahu čítače s obsahem registru TCCR2 (kde je přednastavena hodnota 249 => čítač MOD250) se čítač nuluje a vyvolá se přerušení. Toto nastane každé 4ms. V obslužné rutině přerušení Čítače/časovače 2 je realizován čítač MOD 250 pro odměření intervalu 1,00s. Každou 1s se volá funkce **CountHourMinSek(),** ve které je realizován čítač minut (MOD60) a čítač hodin (MOD24). Tato funkce modifikuje globální proměnné **sek**, **min** a **hod**.

Nulování čítačů sekund, minut a hodin a nastavení aktuálního času se provádí voláním funkce **ResetSetClock()**, která se volá při každém 100. průchodu smyčkou while(1), stejně jako funkce pro obsluhu LCD displeje **RefreshLCD()**. Příliš častá

obsluha spínačů a displeje LCD by zbytečně spotřebovávala strojní čas mikrokontroléru. Počítání průchodů nekonečné smyčky je provedeno čítačem využívajícím proměnnou **ScheduleTmr.**

Funkce pro nastavení hodin **ResetSetClock()** testuje příslušné spínače **S1**, **S2** a **S5**. Po stisku S1 a při současném nahození bitu **bKeyRepeat** (tento bit je nastavován periodicky v obslužné rutině přerušení po 0,5s) dojde k inkrementaci čítače hodin, při držení S1 k tomuto dojde 2x za sekundu. Tímto je dosaženo funkce „autorepeat“ k pohodlnému nastavení hodin. Po 5 přírůstcích čítače hodin (**RepeatCount > 5**) se zkrátí perioda autorepeatu (KeyRepeatTime = 30 místo původní hodnoty 125), čímž se podstatně zrychlí nastavování hodin. Po uvolnění S1 je opět nastaven pomalý autorepeat (KeyRepeatTime = 125). Po každé změně hodnoty čítače se modifikuje údaj na LCD. Stejným způsobem probíhá nastavení čítače minut spínačem S2.

Projekt digitálních hodin obsahuje dva soubory ***.c**, **hodiny.c** (popsaný v předešlých odstavcích) a **lcd.c**. Tento soubor obsahuje rutiny pro obsluhu LCD displeje. Součástí těchto souborů jsou hlavičkové soubory **makra.h**, **spin.h**, **led.h** a **lcd.h**, které obsahují definice připojení k mikrokontroléru, použité I/O porty popř. funkční prototypy funkcí použitých ve stejně pojmenovaných souborech *.c. Výpis souboru **lcd.c** a hlavičkových souborů byl uveden již v příkladech v předcházejících kapitolách a budou součástí přiloženého CD s programy a manuály.

7.7.2 Úlohy k samostatnému řešení

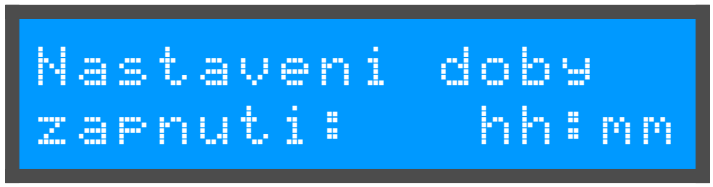
7.7.2.1: Časový spínač 1

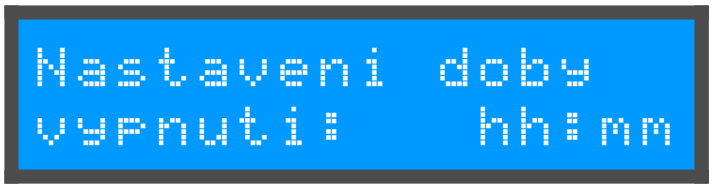
S využitím čítače/časovače 1 vytvořte časový spínač pro spínání elektrospotřebiče. Spínač má umožnit nastavení aktuálního času a jednoho času zapnutí a vypnutí jednoho elektrospotřebiče během 24 hodinového cyklu.

Ovládání:

- S1...nastavení hodin cyklicky (0,1,2,...23, 0,...)
- S2...nastavení minut cyklicky (0, 1, ...59, 0,...)
- S3...přepínání režimu nastavení časů a zobrazení na LCD
- S4...aktivace/deaktivace spínání elektrospotřebiče
- S5...nulování čítačů hodin, minut, sekund a registrů času spínání

Zobrazení na LCD v režimu nastavení:





Zobrazení v provozním režimu bude vypadat následovně:

Eventuálně při neaktivním spínači:

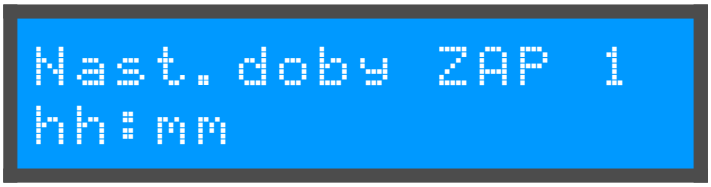
7.7.2.2: Časový spínač 2

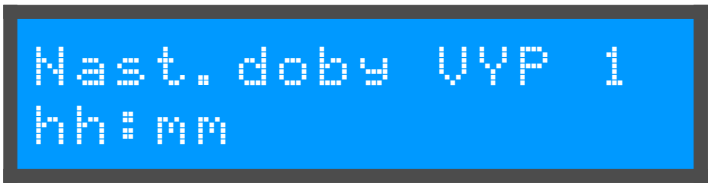
S využitím čítače/časovače 1 vytvořte časový spínač pro spínání elektrospotřebičů. Spínač má umožnit nastavení aktuálního času a 4 časů zapnutí a vypnutí jednoho elektrospotřebiče během 24 hodinového cyklu.

Ovládání:

- S1...nastavení hodin cyklicky (0,1,2,...23, 0,...)
- S2...nastavení minut cyklicky (0, 1, ...59, 0,...)
- S3...přepínání režimu nastavení časů a zobrazení na LCD
- S4...aktivace/deaktivace spínání elektrospotřebiče
- S5...nulování čítačů hodin, minut, sekund a registrů času spínání

Zobrazení na LCD v režimu nastavení:





Stejným způsobem bude provedeno zobrazení pro ostatní 3 časy zapnutí a vypnutí.

Zobrazení v provozním režimu bude vypadat následovně:

Eventuálně při neaktivním spínači:

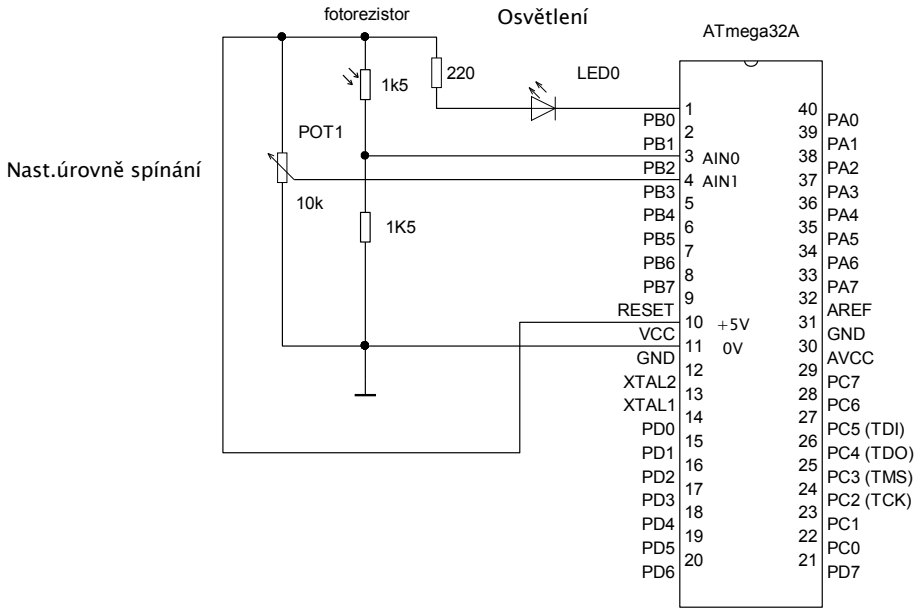
Jako zdroj kmitočtu mikrokontroléru využijte krystalový oscilátor 16MHz. Zapnutí spotřebiče indikujte (kromě zobrazením na LCD) též rozsvícením LED0, aktivní režim spínače (volba pomocí S4) indikujte LED7.

7.8 Analogový komparátor (soumrakový spínač osvětlení)

Vestavěný analogový komparátor umožňuje porovnávat hodnoty dvou vstupních napětí. Při dosažení shody dojde k překlopení výstupu komparátoru a vyvolá se přerušení (pokud je povoleno). V obslužném programu přerušení je možné provádět různé akce, v závislosti na náběžné nebo sestupné hraně nebo při obou hranách výstupního signálu komparátoru. Toto je možné využít např. k realizaci regulačních úloh nebo k měření fyzikálních veličin.

Zadání:

S využitím vestavěného analogového komparátoru realizujte soumrakový spínač osvětlení. Při určité hodnotě osvětlení (nastavené potenciometrem POT1 na vývojové desce EvB 4.3) dojde k překlopení stavu analogového komparátoru a rozsvítí se nebo zhasne LED0 (PortB.0, osvětlení). Osvětlení lze též ovládat ručně spínačem S1 (zapnutí) a S2 (vypnutí) nezávisle na automatickém spínání. Intenzita osvětlení bude snímána fotorezistorem. Schema připojení fotorezistoru a potenciometru pro nastavení úrovně spínání osvětlení k mikrokontroléru je na Obr.20.



Obr.20: Schema připojení fotorezistoru a osvětlení k mikrokontroléru

Ukázkový program:

```
/*****************************************************************************
*
*   Projekt:   AnalogComparator
*   Soubor:    analog_komp.c
*
*   Obsah:     Analogovy komparator ve funkci spinace
*                  venkovniho osvetleni
*                  vstup + (AIN0)...napeti delice s fotoodporem
*                  vstup - (AIN1)...napeti odporoveho delice pro
*                nastaveni urovne spinani
*                  S1...zapnuti svetla rucne
*                  S2...vypnuti svetla rucne
*   Vytvoreno: 2014
*
*****************************************************************************/

/************  include soubory ***********************************************/
#include <atmega32.h>
#include "makra.h"      //definice maker pro bitove operace
#include "spin.h"               //pripojeni spinacu
#include "led.h"          //pripojeni LED indikatoru
#include "lcd.h"          //pripojeni LCD a funkcni prototypy funkci pro
                          //obsluhu LCD
#define OFF 0xff
#define ON 0x00

/************ definice globalnich promennych ******************************/
unsigned char BAJT1;      //bajt pro pomocné bity

/************  prototypy lokalnich funkci *********************************/
void PortsInit(void);
void AnalogCompInit(void);

/************  kod  ******************************************************/
/**************** nastaveni orientace portu ******************************/
void PortsInit(void) {
 DDRA = 0x00;  //port A je vstup
 DDRB = 0xF3;  //port B.2 a B.3 vstup,ostatni bity vystup
 DDRD = 0xFF;  //port D je vystup
 PORTA = 0xFF; //pull-up pripojeny
}

/**************** nastaveni analog. komparatoru ***************************/
 void AnalogCompInit(void) {
 CLRBIT(SFIOR,ACME);         //zakazani uziti analog. multiplexeru
 ACSR = 0x08;                //povol preruseni od anal.komparatoru
                             //na nabeznou i sestupnou hranu vystupu
 SETBIT(SREG,7);             //globalni povoleni preruseni
}
/**************** rutina obsluhy preruseni od analog. komp. ***************/
//tato rutina se vola pri kazde zmene vystupu komparatoru
```

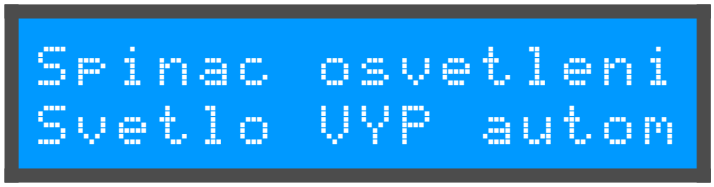
```
void AnalogComp(void)__interrupt[ANA_COMP_vect]
{
 if (TESTBIT(ACSR,ACO)) {
    SETBIT(LEDKY,LED0);    //vypni osvetleni
    LCDposition(0,1);
    LCDprintText("Svetlo VYP autom");
 }
 else {
    CLRBIT(LEDKY,LED0);    //zapni osvetleni
    LCDposition(0,1);
    LCDprintText("Svetlo ZAP autom");
 }

}
/****************  main funkce ******************************************/
int main(void)
{
  PortsInit();
  LCDinit();
  AnalogCompInit();

  LEDKY = OFF;                    //vypni LED

  /************ uvodni napis na LCD ***************************************/
  LCDclear();
  LCDposition(0,0);
  LCDprintText("Spinac osvetleni");
  LCDposition(0,1);
  LCDprintText("S1=ZAP  S2=VYP  ");

  while(1)
  {
  //zapni osvetleni rucne spinacem S1
  if (TESTNEGBIT(SPINACE,S1) & TESTBIT(BAJT1,bS1)) {
    CLRBIT(BAJT1,bS1);
    CLRBIT(LEDKY,LED0);
    LCDposition(0,1);
    LCDprintText("Svetlo ZAP rucne");
  }
  if (TESTBIT(SPINACE,S1)) SETBIT(BAJT1,bS1); //test uvolneni S1

  //vypni osvetleni rucne spinacem S2
  if (TESTNEGBIT(SPINACE,S2) & TESTBIT(BAJT1,bS2)) {
      CLRBIT(BAJT1,bS2);
      SETBIT(LEDKY,LED0);
      LCDposition(0,1);
      LCDprintText("Svetlo VYP rucne");
  }
  if (TESTBIT(SPINACE,S2)) SETBIT(BAJT1,bS2); //test uvolneni S2
  }
}
```

Popis programu:

Analogový komparátor porovnává napětí z potenciometru POT1 a napětí z děliče složeného z fotorezistoru a rezistoru. Při poklesu vnějšího osvětlení stoupá napětí na neinvertujícím analogovém vstupu AIN0. Jakmile toto napětí překročí hodnotu určenou nastavením potenciometru POT1, dojde ke komparaci, na výstupu komparátoru se objeví náběžná hrana signálu. Při nárůstu intenzity vnějšího osvětlení dojde k opačnému jevu, na výstupu komparátoru vznikne sestupná hrana signálu.

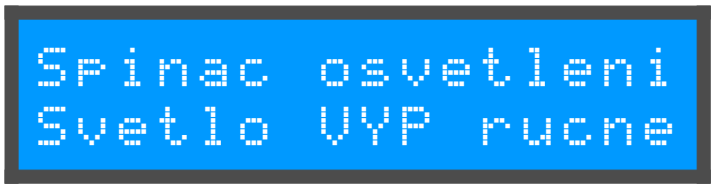
V programu je povoleno přerušení od analog. komparátoru při libovolné hraně změny jeho výstupu a je zakázáno použití analogového multiplexeru (hodnota bitů ACIE = 1, ACIS1 = ACIS0 = 0, => hodnota reg. ACSR = 0x08). Při libovolné změně na výstupu komparátoru se vyvolá obslužná rutina, která testuje výstup komparátoru (bit ACO z registru ACSR).

Je- li napětí na neinvertujícím vstupu AIN0 větší než napětí na invertujícím vstupu AIN1, je bit ACO nahozen, vypne se LED0 (osvětlení) a na LCD displeji se vypíše následující nápis:



Při opačné změně se LED0 zapne a změní se nápis na LCD:

Program umožňuje i ruční zapnutí (spínač S1) a vypnutí osvětlení spínač S2). Toto je provedeno testováním spínačů v hlavní programové smyčce způsobem již v mnoha příkladech použitým. Signalizace ručního zapnutí a vypnutí je též zobrazena na displeji LCD:



Ruční ovládání je zapotřebí např. pro testování osvětlení a při prvním uvedení zařízení do provozu. Pokud by se zapnulo napájení automatického spínače osvětlení ve tmě, nedošlo by k zapnutí osvětlení, jelikož změna stavu spínače nastane až při komparaci vstupních napětí, k tomu dojde ale až při rozednění.

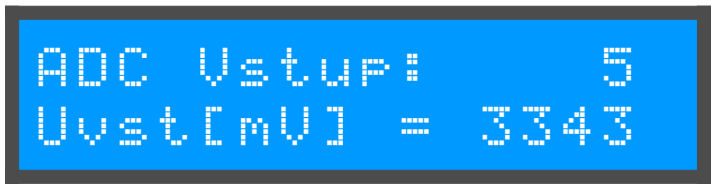
7.9 A/D převodník (voltmetr)

Analogově/digitální převodník se používá k měření napětí nebo jiných fyzikálních veličin na napětí převedených.

Mikrokontrolér Atmega32 má vestavěný 10bitový A/D převodník. Jeho vstup je připojen na 10kanálový analogový multiplexer, je tedy možné snímat až 8 vstupních napětí (fyzikálních veličin) a navíc napětí vnitřního referenčního zdroje napětí a analogové nuly.

V následujícím ukázkovém příkladu je použit A/D převodník k periodickému měření napětí na všech 8 externích vstupech analogového multiplexeru v intervalu 1s.

Zadání:
Vytvořte program, který bude měřit stejnosměrná napětí přivedená na vstupy multiplexeru vestavěného analogovéhgo komparátoru. Napětí budou mít maximální hodnotu 5V a budou se postupně po 1s zobrazovat na LCD displeji následovně:



Zdrojem kmitočtu bude krystalový oscilátor 16MHz. LCD displej bude připojen k portu D. Aktivace převodů a zobrazení bude řízena čítačem /časovačem 0.

Ukázkový program:

```
/*************************************************************************
*
*   Projekt:        voltmetr3
```

```
*   Soubor:       voltmetr3.c
*
*   Obsah:   -ADC v rezimu jednoducheho prevodu, je spousten
*                 62-nasobnym pretecenim C/T 0 v normalnim modu (po 1s)
*                -postupne prepinani vstupu ADC0 az ADC7 dokola
*                -zdroj hodin. kmitoctu krystal. oscilator 16MHz
*                -LCD pripojen k portu D
*
*   Vytvoreno:    10.4.2014
*
*********************************************************************/

/************  include soubory ************************************/
#include <atmega32.h>
#include "makra.h"   //definice maker pro bitove operace
#include "led.h"
#include "lcd.h"        //pripojeni LCD a funkcni prototypy funkci pro
                        //obsluhu LCD

/**************** definice konstant ********************************/
#define ON 0x00
#define OFF 0xff

/************ seznam lokalnich funkci ****************************/
void PortsInit(void);
void Timer0Init(void);
void ADCInit(void);

/**************** nastaveni orientace portu ************************/
void PortsInit(void) {
  DDRA = 0x00;    //port A je vstup
  DDRB = 0xFF;    //port B je vystup
  DDRD = 0xFF;    //port D je vystup (LCD)
}

//inicializace casovace 0
void Timer0Init(void) {
  TCCR0 = 0x15;                  //normal mod,preddelicka :1024,OC=toggle
  SETBIT(TIMSK,TOIE0);        //povoleni prerus pri preteceni
  SETBIT(SREG,7);               //globalni povoleni preruseni (bit I)
}

// inicializace ADC
void ADCInit(void) {
  ADMUX = 0x40;              //Aref = Ucc, vstup ADC0
  ADCSRA =0xcf;              //jednoduchy prevod, ADclk=16MHz/1024=125kHz
}

//interuptova rutina citace/casovace 0
void Timer_0(void)__interrupt[TIMER0_OVF_vect] {
  static unsigned int Interrupt_Counter;  //citac poctu vstupu do interuptove rutiny pouzit
                                  //pro akce konane pri dosazeni nasobku casu 16ms
  static unsigned char ADC_Input_Counter; //citac prepinani vstupu ADC

  Interrupt_Counter++;
```

```
  if (Interrupt_Counter == 62) {                    //62 x 16ms = 0.992 s
   Interrupt_Counter = 0;

   ADC_Input_Counter++;
   if (ADC_Input_Counter == 8) ADC_Input_Counter = 0;

   ADMUX = 0x40 + ADC_Input_Counter;        //prepinani vstupu ADC

   LCDposition(14,0);
   LCDprintDec(ADC_Input_Counter);                    //vypis cisla kanalu ADC

   SETBIT(ADCSRA,ADSC);                               //start AD prevodu po 1s
  }
}

//interuptova rutina obsluhy ADC,vyvola se po ukonceni AD prevodu
void ADC_0(void)__interrupt[ADC_vect] {
  unsigned int Conversion_Result;
  float Input_Voltage;

  Conversion_Result = (ADCH * 256) + ADCL;        //secti horni + dolni bajt prevodu
  Input_Voltage = Conversion_Result * 4.8875;       //1023x4.8875 = 4999 mV

  LCDposition(10,1);
  LCDprintWordToDec(Input_Voltage,0);          //vysledek AD prevodu na LCD
}

/*****************  main funkce ************************************/
int main(void)
{
  PortsInit();                                  //nastaveni orientace portu
  Timer0Init();                                 //inicializace C/T 0
  ADCInit();                                    //inicializace ADC

  LCDinit();                                    //inicializace LCD
  LCDwriteCommand(0x0c);          //vypni kurzor
  LCDclear();

  LEDKY = OFF;

/************* uvodni napis na displeji *********************************/
  LCDposition(0,0);
  LCDprintText("ADC vstup:");
  LCDposition(0,1);
  LCDprintText("Uvst[mV] = ");

  while(1)
  {
    //zde cokoliv
  }
}
```

Popis programu:

Projekt **voltmetr3** obsahuje dva soubory, **lcd.c** a **voltmetr3.c**. Soubor lcd.c již byl popsán v předcházejících příkladech. Soubor voltmetr3.c provádí vlastní obsluhu A/D převodníku. Využívá čítač/časovač 0 pro odměření časového intervalu přibližně 1s. Tento čítač/časovač přeteče vždy po 16ms, po 62. přetečení je dosaženo času přibližně 1s (62 x 16ms = 0.992s). Po každé 1s se inkrementuje proměnná **ADC_Input_Counter**, která slouží pro přepínání kanálů analogového multiplexeru ADC0 až ADC7. Číslo právě měřeného kanálu se zapíše na LCD a spustí se A/D převod (nastavením bitu **ADSC** v registru **ADCSRA**).

Po ukončení A/D převodu se vyvolá interuptová rutina **ADC_0(void)__interrupt[ADC_vect]**. Výsledek převodu se získá součtem výstupních registrů A/D převodníku **ADCH** a **ADCL** a uloží se do proměnné **Conversion_Result**. Tento výsledek je 10bitový, při vstupním napětí odpovídajícím referenčnímu napětí AVCC = 5V je výsledek převodu 1023. Jelikož potřebujeme dostat výsledek 5000 (mV), násobí se konstantou 4.8875 (1023 x 4.8875 = 4999 mV) a uloží se do proměnné **Input_Voltage**, který se poté zobrazí na LCD.

V hlavní funkci programu (main()) se pouze provádí inicializace portů, časovače, A/D převodníku a LCD displeje a vypíše se úvodní nápis na LCD. Nekonečná smyčka programu (while(1)) je v tomto příkladu prázdná, veškerá obsluha voltmetru je řešena pomocí přerušení.

7.10 Paměť EEPROM (řízení krokového motoru)

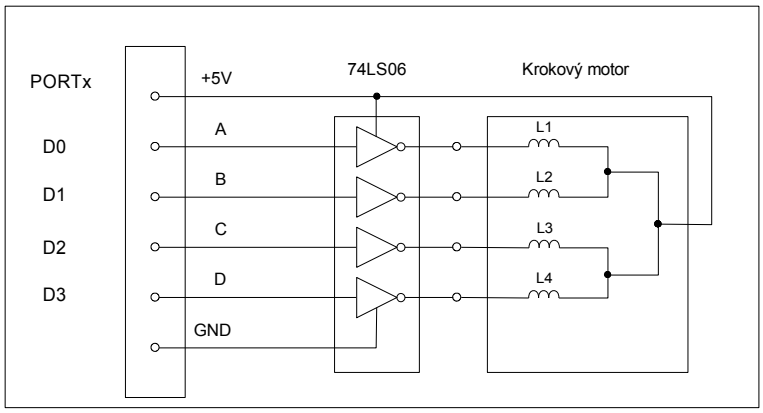
Vnitřní paměť EEPROM se používá např. pro uložení konfiguračních parametrů zařízení, které je třeba uchovat i po vypnutí jeho napájení.

Mikrokontrolér Atmega32 disponuje 1kB paměti EEPROM. Zápis do tohoto druhu paměti vyžaduje nastavení některých speciálních registrů, v praxi jsou vytvořeny funkce pro zápis nebo čtení jednoho bajtu do (z) dané adresy paměti.

Tyto funkce jsou použity v následujícím programu řízení krokového motoru pro uložení nastavené rychlosti otáčení a budou vysvětleny. Tento ukázkový příklad využívá též maticovou klávesnici pro ovládání motoru a zadání požadované rychlosti otáčení. Pro interaktivní ovládání je zde použit LCD displej. Činnost maticové klávesnice i LCD displeje již byla vysvětlena v příkladech v předcházejících kapitolách.

Zadání:

Vytvořte program pro řízení krokového motoru. Krokový motor bude připojen k portu B mikrokontroléru podle schematu na Obr.21.



Obr.21: Připojení krokového motoru k portu mikrokontroléru

Ovládání krokového motoru bude z maticové klávesnice, která bude připojena k portu A mikrokontroléru podle schematu na Obr. 15 .

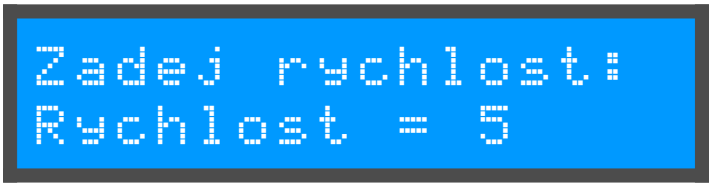
Ovládání bude mít následující možnosti:

- klávesa A ... motor doprava
- klávesa B ... motor stop
- klávesa C ... motor doleva
- klávesy 1 až 9 … zadání rychlosti otáčení
- klavesa # ... potvrzeni zadane hodnoty rychlosti a její okamžitá změna

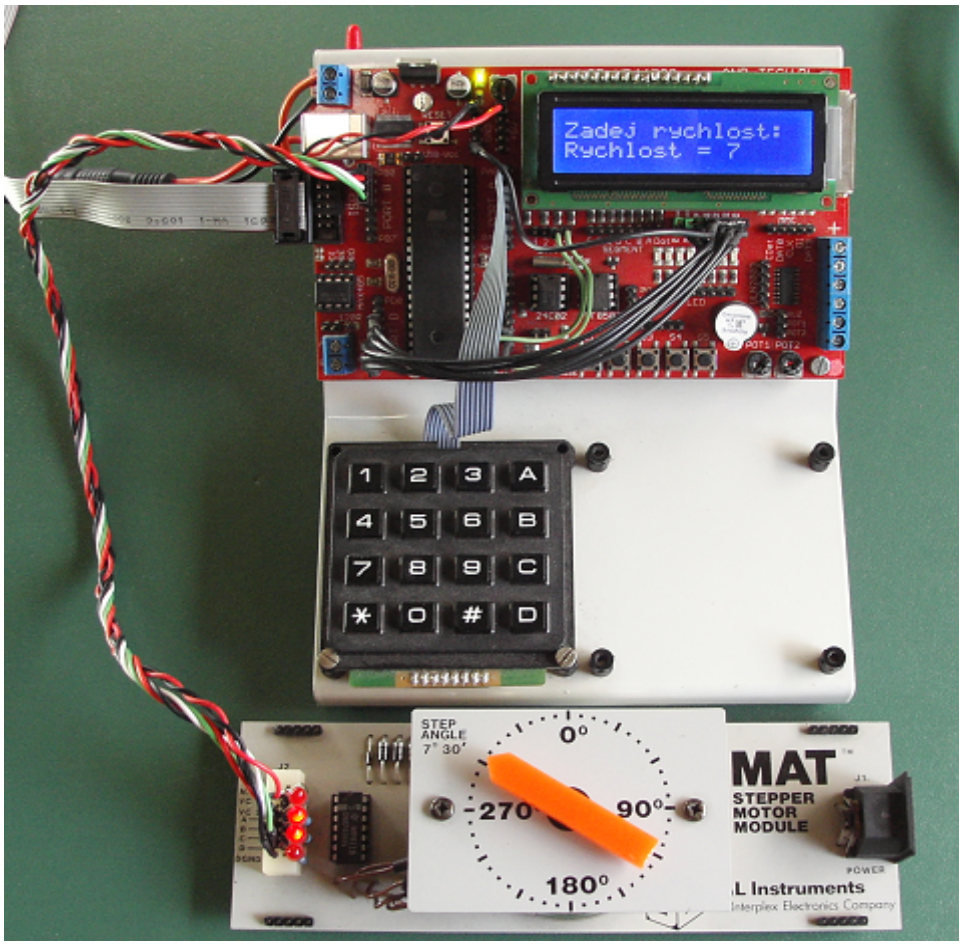
Zadaná rychlost bude po každé její změně uložena do paměti EEPROM a po zapnutí napájení se z této paměti načte => motor se bude otáčet stejnou rychlostí, jakou měl při vypnutí zařízení (napájení mikrokontroléru).
Maximální rychlost otáčení (rychlost 9) nastavte experimentálně podle použitého krokového motoru tak, aby motor „nevypadl“ ze synchronizace. Minimální rychlost otáčení (rychlost 1) volte přibližně 1ot/sek.

Na LCD displeji se po zapnutí objeví úvodní nápis:

Po zadání rychlosti a potvrzeni klávesou # se na LCD objeví nápis:



Zdrojem kmitočtu bude vnitřní kalibrovaný oscilátor 1MHz.



Obr.22 Fotografie sestavy krokového motoru s modulem EvB4.3

Ukázkový program:

```
/*****************************************************************************
*   Projekt:       step_motor5
*   Soubor:        stepper_motor5.c
*
*   Obsah:         klavesa A ... motor doprava
*                  klavesa B ... motor stop
*                  klavesa C ... motor doleva
*                  klavesa # ... potvrzeni zadane hodnoty rychlosti
*
*   Vytvoreno:   5.1.2014
*
*****************************************************************************/
/************  include soubory ***********************************************/
#include <atmega32.h>
#include "makra.h"   //definice maker pro bitove operace
#include "spin.h"      //definice bitu spinacu a pomocnych bitu Bajt1 a Bajt2
#include "km.h"        //definice pripojeni krokoveho motoru a postup spinani civek
#include "lcd.h"       //pripojeni LCD a funkcni prototypy funkci pro obsluhu LCD
#include "eep.h"
#include "key.h"
```

```
/************  definice konstant ************************************************/
unsigned int Speed[10] = {0,9600,8450,7300,6150,5000,3850,2700,1550,400};

/************  definice globalnich promennych ***********************************/
unsigned char Bajt1,Bajt2;                //bajty pro bitove promenne
unsigned int Actual_Speed = 8500;   //aktualni rychlost

/****************  prototypy lokalnich funkci ***************************************/
int main(void);
void Delay (void);
void PortsInit(void);
void KeybTest(void);
void StepperMotorControl(void);
unsigned int SestavCislo(void);

/****************  kod  *************************************************************/

/**************** casova smycka  ***************************************************/
//urcuje rychlost pohybu motoru, nejmensi hodnotu promenne  Actual_Speed je
//nutno urcit pri ladeni tak, aby motor "nevypadl" ze synchronizace

void Delay(void)
{
  unsigned  int  k;
  for(k = 0; k < Actual_Speed; k++);
}

/**************** nastaveni orientace portu ***************************************/
void PortsInit(void)
{
  DDRA = 0x0f;    //port A horni 4 bity = vstup, dolni 4 bity = vystup
  DDRB = 0xFF;    //port B je vystup
  DDRD = 0xFF;    //port D je vystup
  PORTA = 0xFf;   //pull-up pripojeny
}

/**************** test stisku klavesy a nahozeni smerovych bitu ************************/
// klavesa "A" nahazuje bit bRight a nuluje bLeft
// klavesa "B" nuluje bity bRight i bLeft
// klavesa "C" nahazuje bit bLeft a nuluje bRight

void KeybTest(void)
{
//motor doprava
  if (ReadKeyb() == 10)   //klavesa "A"
  {
    CLRBIT(Bajt1,bLeft);
    SETBIT(Bajt1,bRight);
  }
//motor stop
  if (ReadKeyb() == 11)  //klavesa "B"
  {
    CLRBIT(Bajt1,bRight);
    CLRBIT(Bajt1,bLeft);
```

```
  }
//motor doleva
  if (ReadKeyb() == 12)   //klavesa "C"
  {
    CLRBIT(Bajt1,bRight);
    SETBIT(Bajt1,bLeft);
  }
}

//************** rizeni krokoveho motoru *********************************/
void StepperMotorControl(void)
{
  static unsigned char Step;

//pohyb doprava trvale
     if (TESTBIT(Bajt1,bRight))
     {
        if (Step < 4) Step ++;
        else Step = 1;

//      MOTOR = motor[Step];       //jednofazove rizeni (vzdy aktivni pouze jedna civka KM)
        MOTOR = motor_2f[Step];   //dvoufazove rizeni (vzdy aktivni dve civky KM)
        Delay();                  //urcuje dobu kroku KM
     }

//pohyb doleva trvale
    if (TESTBIT(Bajt1,bLeft))
    {
        if (Step > 1) Step --;
        else Step = 4;

//      MOTOR = motor[Step];          //jednofazove rizeni (vzdy aktivni pouze jedna civka KM)
        MOTOR = motor_2f[Step];       //dvoufazove rizeni (vzdy aktivni dve civky KM)
        Delay();                      //urcuje dobu kroku KM
    }
}


/****************  main funkce ******************************************/
int main(void)
{
    unsigned char Ask_Speed  = 0;     //pozadovana rychlost
    unsigned char Read_Character;     //nacteny znak z klavesnice

    MOTOR = motor[2];                 // motor neaktivní
    PortsInit();

    LCDinit();                        //inicializace LCD
    LCDwriteCommand(0x0c);            //vypni kurzor
    LCDclear();

//uvodni napis na LCD
    LCDposition(0,0);
    LCDprintText("Zadej rychlost  ");
    LCDposition(0,1);
```

```
  LCDprintText("otac.1-9,stisk.#");

  Actual_Speed = ((EEPread(1) * 256) +  EEPread(0));          //nacteni rychl. z EEPROM

 while(1)
 {
  Read_Character = ReadKeyb();                                //cti znak maticove klavesnice

  if ((Read_Character != 0x00) && (Read_Character != 0xff) && (Read_Character < 10) &&
TESTBIT(Bajt2,bPom0))   //stisknuta nejaka klavesa
  {
     CLRBIT(Bajt2,bPom0);

     Ask_Speed  = Read_Character;
     LCDposition(0,0);
     LCDprintText("Zadej rychlost:  ");
     LCDposition(0,1);
     LCDprintText("Rychlost = ");
     LCDprintDec(Ask_Speed);
     LCDprintText("    ");                                   //smaz zbytek radku z predchaz. vypisu
  }

  if (Read_Character == 0xff) SETBIT(Bajt2,bPom0);            //test uvolneni klavesy

  if ((Read_Character == 15)  && TESTBIT(Bajt2,bPom1))    //potvrzeni zadani klavesou "#"
  {
    CLRBIT(Bajt2,bPom1);

    LCDclear();
    LCDposition(0,0);
    LCDprintText("Zadej rychlost:  ");
    LCDposition(0,1);
    LCDprintText("Rychlost = ");
    LCDprintDec(Ask_Speed);

    Actual_Speed = Speed[Ask_Speed];

    EEPwrite(0,Actual_Speed);                  //dolni bajt do EEP na adresu 0
    EEPwrite(1,Actual_Speed / 0xff);           //horni bajt do EEP na adresu 1
  }

   if (Read_Character == 0xff) SETBIT(Bajt2,bPom1);

   KeybTest();                                 //test ridicich klaves (A,B,C)  z matic. klavesnice
   StepperMotorControl();
  }
}
```

Popis programu:

Projekt řízení krokového motoru **step_motor5** obsahuje čtyři soubory ***.c**: **stepper_motor5.c** (popsaný v předešlých odstavcích), **lcd.c,** (popsaný v příkladu digitálních hodin v kapitole 7.7.1), dále soubor **key.c** pro obsluhu maticové klávesnice a soubor **eep.c** pro obsluhu EEPROM paměti**.** Výpis těchto dvou

posledních souborů je uveden v následujících řádcích a bude též součástí doprovodného CD.

```
/*****************************************************************************
*
*   Soubor:       key.c
*
*   Obsah:        Rutiny pro obsluhu maticove klavesnice
*
*   Vytvoreno:    26.11.2013
*
*****************************************************************************/
#include <atmega32.h>
#include "key.h"

/************ obsluha maticove klavesnice  ******************************/
//funkce vraci desitkovou hodnotu (0 az 15) hexacislice  naposledy stisknuteho tlacitka
//pri nestisknute zadne klavese vraci hodnotu 0xff

unsigned char ReadKeyb(void)
{

   unsigned char klav_nactena;         //nactena hodnota ze sloupcu klavesnice
   unsigned char pom, i;
//cteni klavesy 4.radku
   RADEK = radek4;                     //na 4.radek(horni) log.0
   for (i = 0;i < 10;i++);             //ustaleni stavu urovni na portech
   klav_nactena = SLOUPCE >> 4;   //cte stisknutou klavesu a posle na spodni 4 bity
   if(klav_nactena != 0x0f)            //stisknuta klavesa=>najdi sloupec, prirad kod, jinak nic
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 1;break;          //1.sloupec ,znak 1
       case 0x0b:dekod_znak = 2;break;          //2.sloupec ,znak 2
       case 0x0d:dekod_znak = 3;break;          //3.sloupec ,znak 3
       case 0x0e:dekod_znak = 10;break;         //4.sloupec ,znak A
     }
     return dekod_znak;
   }

//cteni klavesy 3.radku
   RADEK = radek3;                             //na 3.radek log.0
   for (i = 0;i < 10;i++);
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 4;break;          //1.sloupec ,znak 4
       case 0x0b:dekod_znak = 5;break;          //2.sloupec ,znak 5
       case 0x0d:dekod_znak = 6;break;          //3.sloupec ,znak 6
       case 0x0e:dekod_znak = 11;break;         //4.sloupec ,znak B
     }
     return dekod_znak;
```

```
   }
//cteni klavesy 2.radku
   RADEK = radek2;                              //na 2.radek log.0
   for (i = 0;i < 10;i++);
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 7;break;          //1.sloupec ,znak 7
       case 0x0b:dekod_znak = 8;break;          //2.sloupec ,znak 8
       case 0x0d:dekod_znak = 9;break;          //3.sloupec ,znak 9
       case 0x0e:dekod_znak = 12;break;         //4.sloupec ,znak C
     }
     return dekod_znak;
   }
//cteni klavesy 1.radku
   RADEK = radek1;                              // na1.radek(spodni)log.0
   for (i = 0;i < 10;i++);
   klav_nactena = SLOUPCE >> 4;
   if(klav_nactena != 0x0f)                     //pokud je stisknuta nejaka klavesa
   {
     switch (klav_nactena)
     {
       case 0x07:dekod_znak = 14;break;   //1.sloupec ,klavesa *, znak E
       case 0x0b:dekod_znak = 0;break;    //2.sloupec ,znak 0
       case 0x0d:dekod_znak = 15;break;   //3.sloupec ,klavesa #, znak F
       case 0x0e:dekod_znak = 13;break;   //4.sloupec ,znak
     }
     return dekod_znak;
   }
   else return 0xff;                            //nic nestisknuto
}
```

Popis souboru **key.c** :

Funkce postupně aktivuje všechny čtyři řádky klávesnice tak, že posílá na řádky log. „0“. Po každé aktivaci nového řádku čte stav sloupců klávesnice. Pokud je některá klávesa stisknuta (if(klav_nactena != 0x0f)), přečte se kombinace na sloupcích klávesnice a pomocí přepínače **switch(klav_nactena)** se přiřadí odpovídající znak proměnné **dekod_znak.** Příkazem **return** se funkce ukončí a předává (volání funkce vlastně vrací) hodnotu stisknuté klávesy. Při nestisknuté klávese vrací funkce hodnotu 0xff.

Do souboru **key.c** je vložen soubor **key.h**, který obsahuje informace o zapojení klávesnice k portu a funkční prototyp funkce **ReadKeyb()**.

```
/*****************************************************************************
*
*   Soubor:       key.h
*
*   Obsah:        prototypy funkci
```

```
*
*   Vytvoreno:    26.5.2013
*
*******************************************************************************/
//  zapojeni klavesnice:
//  b7    b6    b5    b4    b3    b2    b1    b0
//  sl1   sl2   sl3   sl4  rad4  rad3  rad2  rad1   (sl1=vlevo,rad1=spodni)



/********** funkcni prototypy  ************************************/

static unsigned char dekod_znak;    //dekodovana hodnota klavesy v hexa kodu

//funkce vraci desitkovou hodnotu (0 az 15) hexacislice  naposledy stisknuteho tlacitka
unsigned char ReadKeyb(void);

//********** definice ***************************************************/

#define SLOUPCE PINA          //maticova klavesnice, horni 4 bity
#define RADEK PORTA           //maticova klavesnice, dolni 4 bity

#define radek1 0xfe           //spodni radek klavesnice
#define radek2 0xfd
#define radek3 0xfb
#define radek4 0xf7           //horni radek klavesnice
```

```
/*******************************************************************************
*
*   Soubor:       eep.c
*
*   Obsah:        Rutiny pro obsluhu EEPROM pameti
*
*   Vytvoreno:    2.1.2014
*
*******************************************************************************/
#include <atmega32.h>
#include "eep.h"                    //prototypy funkci pro obsluhu EEP

/******************* cteni z EEP *************************/
unsigned char EEPread(unsigned int Adress)
{
   while(EECR & (1 << EEWE));  //cekej na dokonceni predchoziho zapisu do EEP
   EEARH = 0;                   //napln adresovy registr
   EEARL = Adress;
   EECR |= (1 << EERE);         //povoleni cteni
   return EEDR;                 //vraci data z prislusne adresy EEP
}

/******************* zapis do EEP *************************/
void EEPwrite(unsigned int Adress, unsigned char Data)
{
   while(EECR & (1 << EEWE));  //cekej na dokonceni predchoziho zapisu do EEP
```

```
   EEARH = 0;                    //napln adresovy registr
   EEARL = Adress;
   EEDR = Data;                  //napln datovy registr
   EECR |= (1 <<EEMWE);          //povol zapis
   EECR |= (1 << EEWE);          //zapis data na prislusnou adresu
}
```

Popis souboru **eep.c** :

Tento soubor obsahuje funkci EERead() pro čtení z paměti EEPROM a funkci EEPwrite() pro zápis do paměti EEPROM. Čtení a zápis je prováděno po jednotlivých bajtech.

Operace čtení:

- testuje se, zda bit EEWE = 0, což indikuje dokončení předchozího zápisu do EEPROM
- zapíše se adresa požadované buňky EEPROM do adresového registru
- nastaví se bit povolení čtení EERE v řídícím registru EECR
- funkce vrací hodnotu přečteného bajtu

Operace zápisu:

- testuje se, zda bit EEWE = 0, což indikuje dokončení předchozího zápisu do EEPROM
- zapíše se adresa požadované buňky EEPROM do adresového registru
- zapíší se data do datového registru EEDR
- povolí se zápis nastavením bitu EEMWE v řídícím registru EECR
- aktivuje se zápis dat nastavením bitu EEWE v řídícím registru EECR

```
/*****************************************************************************
*
*   Soubor:       eep.h
*
*   Obsah:        prototypy funkci
*
*   Vytvoreno:    26.5.2013
*
*****************************************************************************/


/*********** funkcni prototypy  ************************************/

/******************* cteni z EEP *************************/
unsigned char EERead(unsigned int Adress);

/******************* zapis do EEP *************************/
void EEPwrite(unsigned int Adress, unsigned char Data);
```

Nově je použit hlavičkový soubor **km.h**, který obsahuje popis připojení krokového motoru k bitům patřičného portu mikrokontroléru a různé způsoby řízení motoru.

```
/*****************************************************************************
*
*   Soubor:       km.h
*
*   Obsah:        definice pripojeni krokoveho motoru
*
*   Vytvoreno:    20.12.2013
*
*****************************************************************************/
#define MOTOR PORTB

//motor pripojen k dolnim 4 bitum portu, A=bit0, B=bit1, C= bit2, D=bit3

//jednofazove rizeni s plnym krokem (vzdy aktivni pouze jedna civka)
//doprava: A,D,B,C
//doleva:  C,B,D,A

unsigned char motor[5] = {0xff,0xfe,0xf7,0xfd,0xfb};         //jednofazove rizeni (4 kombinace)


//dvoufazove rizeni s plnym krokem (vzdy aktivni dve civky)
//doprava: AD,DB,BC,CA

unsigned char motor_2f[5] = {0xff,0xf6,0xf5,0xf9,0xfa};    //dvoufazove rizeni (4 kombinace)




//dvoufazove rizeni s polovicnim krokem
//doprava: A,AD,D,DB,B,BC,C,CA


unsigned char motor_2f_pul[5] = {0xff,0xfc,0xf6,0xf7,0xf5,0xfd, 0xf9, 0xfb,0xfa};    //dvoufazove
rizeni (8 kombinaci)
```

Popis souboru **km.h**:

Krokový motor je možné řídit několika způsoby, vždy je řízení provedeno postupným spínáním jednotlivých cívek motoru nebo jejich kombinací. Tím dosáhneme různého krokování, t.j. posun rotoru o určitý úhel.

Při jednofázovém řízení je vždy aktivní pouze jedna cívka krokového motoru, rotor se při přepínání otočí o 7,5°, na jednu otáčku je nutné provézt 48 kroků. O stejný úhel se motor otáčí i při dvoufázovém řízení s plným krokem, vždy jsou aktivní dvě sousední cívky motoru, motor má větší kroutící moment a snese rychlejší krokování než při jednofázovém řízení.

Jemnějšího krokování dosáhneme použitím dvofázového řízení s polovičním krokem, rotor se při přepínání otočí o 3,75°, na jednu otáčku je nutné provézt 96 kroků.

V souboru **km.h** jsou pro všechny způsoby řízení uvedena datová pole s kombinacemi spínání cívek. První prvek pole má vždy hodnotu 0xff, cívky motoru jsou odpojeny (aktivní je log.0).

7.10.1 Úlohy k samostatnému řešení

7.10.1.1: Lékárnická míchačka KM

Program bude číst stav spínačů na portu A.

Při sepnutém spínači Start (S1) zapne chod lékárnické míchačky.

Míchačka využívá krokový motor, který se bude otáčet 10 otáček doprava, následuje 2s motor v klidu, poté 10 otáček doleva, 2s klid a tento cyklus se opakuje až do stisku tlačítka Stop (S2).

Po stisku tlačítka Cisteni (S3) se otočí motor 2 otáčky doprava, následuje 0,5s klid, pak 2 otáčky doleva, 0,5s klid. Tento cyklus se opakuje 10krát.

7.10.1.2: Řízení otáček krokového motoru

Program bude číst stav spínačů na portu A. Ovládání bude následující:

- S1 … zapnutí otáčení doleva
- S2 … stop otáčení
- S3 … zapnutí otáčení doprava
- S4 … snižování rychlosti otáčení skokově (10 kroků)
- S5 … zvyšování rychlosti otáčení skokově (10 kroků)

Nastavte experimentálně rozsah nejvyšších otáček, které motor při daném způsobu řízení zvládne. Minimální otáčky přibližně 100 ot/min.

8 Životní prostředí

8.1 Životní prostředí a svět

Životní prostředí se v posledních desetiletích značně změnilo. Zdroje pitné vody se zhoršily, v zemědělských produktech se objevují dusičnany a další nežádoucí látky, stále častěji se setkáváme s negativními vlivy ovzduší na lidský organizmus a ve výrobcích se vyskytují toxické materiály. Každý výrobek má určitý vliv na životní prostředí, ať už při své výrobě, při užívání, nebo při likvidaci. Proto se ukazuje stále více nutnost zavádění managementu životního prostředí především ve výrobních podnicích. Cílem je nejen zdokonalit materiálové složení výrobků, ale i úspora energií, a tak omezit negativní dopad výrobků na životní prostředí v průběhu celého jejich životního cyklu. K této skutečnosti se váže legislativa na celém světě, v EU pak v podobě některých směrnic a zákonů, jež bude nutné respektovat jak z hlediska jednotlivých členských států, tak z hlediska firem a také spotřebitelů.

Výrobní a návrhové firmy musí pochopit, proč jsou otázky životního prostředí stále důležitější a závažnější, a musí reagovat na požadavky kladené nejen legislativou, ale i trhem zahrnujícím širokou veřejnost. Promyšlený a aktivní přístup k ekologickému návrhu a konstrukci výrobků může navíc vést k mnoha tvůrčích změnám, novým nápadům a tím podpořit lidský pokrok. Vývoj naznačuje, že management životního prostředí, někdy nazývaný také environmentální management, se stane nezbytnou součástí firem, tak jak je to dnes již běžné v případě managementu jakosti. Nový přístup k řešení této problematiky představuje zavedení ekologického návrhu (eco-design) do inovace a návrhu nových výrobků.

8.2 Ekologie a elektrotechnický sektor

Elektronický průmysl je významnou součástí světové ekonomiky a dnes je prakticky využíván ve všech odvětvích průmyslu a hospodářství. Nejenom velké podniky, ale i malé a střední firmy sehrávají ve využívání elektrických a elektronických systémů značnou roli týkající se procesu proměny a inovace výrobků. Pokrok a dosahované úspěchy v tomto sektoru přináší celou řadu nových skutečností souvisejících s používáním nových materiálů a technologií, což však současně vyvolává i jisté obavy týkající se vlivů na životní prostředí. Například zařízení používaná v domácnostech a kancelářích spotřebují více než 25% celkové spotřeby elektrické energie a osvětlení domácností se podílí na spotřebě energie v bytovém sektoru jen 17%, přičemž mnohem větší část této energie se spotřebuje na topení. Navíc rychlý vývoj, inovace a dostupnost elektroniky je úzce spojená s dnes typickým chováním spotřební společnosti, jež se vyznačuje tendencí stále častěji nakupovat nové výrobky a zbavovat se starých. Sleva elektronických výrobků je dnes běžnou praxí, ale málokdo si uvědomí, že když se ocitne nějaký elektronický výrobek ve slevě, je pravděpodobné, že byl vyroben ze surovin a součástek pocházejících z různých částí světa, které s velkou pravděpodobností svět již také několikrát procestovaly. Stále větší složitost elektrických a elektronických přístrojů znamená, že obsahují celou řadu různých materiálů, které ovlivňují životní prostředí jak při své přípravě a zpracování, tak i při likvidaci. Některé jsou specifické a také nebezpečné lidem i životnímu prostředí.

Toto všechno jsou důvody, proč musí v co nejkratší době elektronický průmysl sehrát zásadní roli v otázkách ochrany životního prostředí.

Lze položit otázku, jak může být elektronický průmysl životnímu prostředí prospěšný. V důsledku jeho rozšířenosti a skutečnosti, že elektronické výrobky dnes používá převážná část lidstva, může koordinovaný a systematický přístup přispět významným způsobem k pozitivnímu vývoji životního prostředí. Jako velmi obecný příklad lze uvést proces miniaturizace, která vede k nižší spotřebě materiálů a k nárůstu objemu informací soustředěných v menším objemu hmoty (výrobku). To podporuje celkový vývoj vytváření globálních databází prostřednictvím internetu, čímž se rozšiřuje možnost vzdělávání, čímž vzniká prostor pro navazování kontaktů a pro spolupráci jedinců i firem z celého světa. Zvyšuje se také výkonnost výrobních kapacit díky automatizaci procesů a zařízení. Pro pochopení všech souvislostí je nutné si uvědomit, jak to vše souvisí s životním prostředím a co vše vlastně životní prostředí znamená.

8.2.1 Životní prostředí

Z diskusí o ohrožení životního prostředí vychází obvykle jako nejožehavější problém globální oteplování. Naléhavých otázek je však mnohem více, např. vyčerpání zdrojů surovin či vysoká spotřeba a znečisťování vody. Spotřeba vody není v převážné části evropských zemích zásadní problém, ale kritická začíná být situace v mnoha regionech světa, kde se vyrábějí elektronické součástky. Znečištění vody toxickými látkami a vysušování vodních ploch rovněž značně komplikují a zhoršují celkový stav životního prostředí. Některé regiony jsou zatíženy smogem,jenž může být vyvolán výfukovými plyny, jež jsou příčinou fotochemického smogu, který způsobuje kyselé deště a další šíření toxických látek. Hluk, zápach a záření rovněž patří mezi další vážné problémy především ve velkých městech.

Málokdo si však uvědomuje, že všechny tyto negativní aspekty souvisí s životním cyklem každého výrobku, jenž působí na životní prostředí, často i několikanásobně. Konkrétní výrobek se může podílet na životním prostředí v jednotlivých fázích celkového životního cyklu, které sestávají z řady dílčích kroků jimiž jsou:

- získání surovin
- výroba součástek
- montáž sestav
- distribuce a prodej
- používání výrobku
- opravy a modernizace
- někdy opětovné použití
- likvidace (nebo recyklace materiálů) použitého, již nepotřebného výrobku

V průběhu životního cyklu působí řada vztahů, jako např. vztah mezi dodavateli a zákazníky, spotřebiteli a případně těmi, kdo výrobek opravují, modernizují recyklují. To znamená, že jednotlivé subjekty mají ať už přímý či nepřímý vliv, a také odpovědnost za působení výrobku na životní prostředí během celého životního cyklu.

8.2.2 Ekologické výrobky se lépe prodávají

Ekologická uvědomělost souvisí s tvůrčím přístupem a inovací. Samotné dodržovaní legislativy je sice přínosem, ale vyžaduje určitý stupeň byrokracie přinášející spíše nevýhody. Proto musí být jedním z prvních kroků rozpoznání ekonomických přínosů spojených se zaváděním strategií návrhu ekologických výrobků na cestě k aktivnějšímu přístupu k ochraně životního prostředí.

Ekologická uvědomělost a vyspělost souvisí s vytvořením dobrého „image“ značky na trhu. Spotřebitelům, kteří chápou potřebu ochrany životního prostředí a někdy si také uvědomují, že ekologické výrobky jsou také výkonnější, se „zelená prodává lépe“. Existuje celá řada ekologických značení, jež informují spotřebitele o ekologických vlastnostech výrobku. Kromě toho, že tyto výrobky jsou ekologické, jsou v mnoha případech také výkonnější, bezpečnější pro spotřebitele, spolehlivější a často i jakostnější.

Častou námitkou bývá tvrzení, že strategie podpory životního prostředí je pro firmy nákladná. Ve skutečnosti je ale cílem zavedení účinného ekologického systému, tak jako v případě zavádění systému jakosti, dosáhnout naopak snížení nákladů. Například nižší spotřeba materiálů a nižší ztráty během výroby, výroba s nižší spotřebou energie jsou pro výrobce výhodné, nehledě na snížení vnitřního rizika a zvyšování motivace zaměstnanců. Strategie uplatňování ekologického návrhu a zavedení ekologického systému navíc souvisí s vývojem a modernizací výrobků a zvýšením jejich výkonnosti a tím také konkurenceschopnosti. V neposlední řadě potom ekologický návrh nových výrobků znamená aktivní přístup ke splnění legislativních požadavků. Jak ekologická uvědomělost celosvětově stoupá, právě koncoví spotřebitelé se stávají jedním z hlavních motivů pro prosazování ekologické strategie výrobků.

S jistými regionálními rozdíly je prevence znečištění životního prostředí chápána jako úkol zcela zásadní. Proto mnoho spotřebitelů dokáže ocenit ekologickou šetrnost výrobků.

V řadě zemí existují různé ekologické značky pro různé typy výrobků. V roce 2003 mělo více než 10 000 výrobků celoevropské či regionální ekologické značení. Např. v Německu v roce 2004 asi 83% spotřebitelů uvedlo, že znají značku „Blue Angel“.

Z toho 49% uvedlo, že tato značka hraje roli při nákupu výrobku. Ekologické značení není důležité pouze pro soukromý sektor, ale hraje roli i ve velké části veřejných zakázek, kde ekologické vlastnosti výrobků mají svůj rozhodující význam. Cena, funkčnost a obsluha mají zásadní vliv na rozhodnutí k nákupu, nicméně přívlastek ekologický může být dalším důvodem, proč daný produkt koupit. Dle průzkumu Německé federální agentury pro životní prostředí odpovědělo na otázku, zda jsou zákazníci ochotni zaplatit více za ekologicky šetrný výrobek, 10% německých spotřebitelů „určitě ano“, dalších 53% „spíše ano“. Je třeba si uvědomit, že ekologický výrobek neznamená, že musí být nutně dražší. Ve skutečnosti ekologické výrobky často bývají levnější, zvláště když budeme uvažovat náklady na jejich životní cyklus.

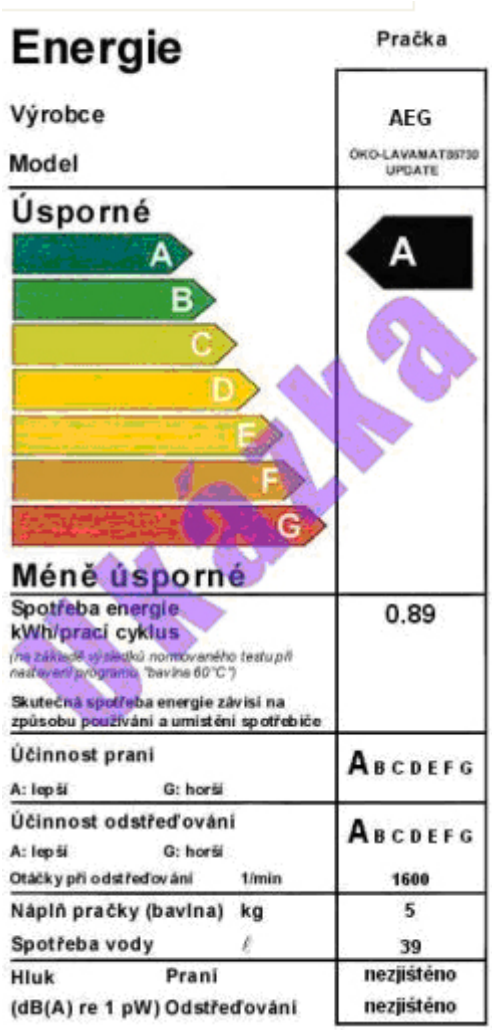
8.2.3 Ekologické značení výrobků v ČR

Značku propůjčuje Ministerstvo životního prostředí ČR, které pro ekologicky šetrné výrobky připravuje i směrnice k jejich hodnocení. Pro udělení značky jsou zavedena výběrová kritéria, které stanovují ekologické parametry výrobků jak při jejich provozu (např. emise, spotřeba energie, uvolňování chemických látek), tak během životního cyklu výrobku (např. spotřeba energie a surovin při výrobě nebo zda a jak se dá recyklovat), ale posuzuje se i obal. Výrobce musí o udělení značky sám požádat a zaplatit určitý poplatek. Udělení značky je podmíněno certifikaci produktu nezávislou třetí stranou.

Příklady ekologických značení výrobků:

U počítačů, tiskáren, kopírek a další kancelářské techniky se můžeme setkat s označením "Energy star", které mají zařízení s nízkou spotřebou. Většinu osobních počítačů lze nastavit tak, aby při nečinnosti spotřebovávaly co nejméně. Pokud je však nezbytné, aby počítač běžel nepřetržitě, je možné ho alespoň zaměstnat nějakým užitečným úkolem. Existují mezinárodní projekty, v nichž může i váš počítač spolupracovat na vědeckém výzkumu, např. modelování klimatických změn nebo výzkum proteinů využitelný v boji s rakovinou (viz např. http://www.boinc.cz).

TCO 99 a TCO 03
Značky Ecology energy emissions ergonomics vypovídají o zdravotní bezpečnosti/nezávadnosti označených přístrojů, jež v sobě zahrnuje působení elektrostatických polí, el. vlnění, rentgenové záření. Druhá značka poukazuje navíc na kvalitu obrazu a barev a snížení emisí a látek, které poškozují životní prostředí během výroby a recyklace. Ruku v ruce s oběma značkami jde nižší spotřeba energie. Značky naleznete na LCD displejích, přenosných počítačích, klávesnicích, kopírkách, monitorech, atd. Přímo na energetickém štítku může být i česká nebo evropská ekoznačka (viz ekologicky šetrný výrobek) .



Označování elektrospotřebičů energetickými štítky je v ČR povinné od roku 2001. Stejná povinnost platí v EU již několik let. Na každé chladničce, pračce atp. je výrazná barevná samolepka, která i laikovi řekne, jaký je výrobek z hlediska spotřeby elektrického proudu.

Průmyslové zákazníky mohou motivovat také různé strategie ekologického návrhu, především pak mezinárodní projekty s vlastní politikou v oblasti životního prostředí, což může mít zásadní vliv na dodavatele. Přinejmenším mohou požadovat, aby dodavatelé do jisté míry jednali podle zásad managementu životního prostředí. Poměrně často bývá žádána specifikace materiálního složení dodávaných výrobků, ať už detailní seznam použitých látek nebo popis materiálu.

Další obchodní výhodou ekologických výrobků je změna komplexního pohledu na výrobek. Návrh výrobků s ohledem na ekologické aspekty může vést k novému, vysoce inovačnímu pojetí. Ekologická analýza výrobků vede k lepšímu porozumění složení součástek a funkcí a zároveň vztahů v rámci dodavatelského řetězce. Dobré řízení dodavatelského řetězce je nezbytným předpokladem pro vysokou jakost výrobku.

8.3 Význam návrhu nového výrobku

Tradiční přístup k ochraně životního prostředí spočívá v prevenci znečištění prostředí nebo v hospodaření s odpadem. Avšak tyto činnosti se zaměřují pouze na

to, jak se vyhnout nebo minimalizovat případné negativní dopady na životní prostředí, ale neřeší otázku jak předcházet těmto negativním vlivům ekologickým návrhem výrobků.

Na počátku přistoupení ke strategii ekologického návrhu stojí zvážení výrobních nákladů. Jaká část výrobních nákladů souvisí se surovinami, výrobním zařízením, spotřebou vody a energie? Není snadné přesně určit tato čísla v rámci celého dodavatelského řetězce, ale je jisté, že zde existují nemalé možnosti dosáhnout úspor. Například u výrobců desek s plošnými spoji souvisí 20-40% celkových výrobních nákladů se surovinami a spotřebou energie. Minimalizací objemu materiálu použitého na jeden výrobek se sníží náklady a zároveň se zvýší jeho ekologická šetrnost. Omezení množství a různorodosti výrobních prostředků, např. chemikálií vede k nižší potřebě interní logistiky. Vyřazení nebezpečných materiálů z výroby může snížit manipulační náklady, menší rozměry výrobků souvisí s nižší spotřebou obalových materiálů a použití recyklovaných materiálů rovněž přináší úspory. Konečně jednoduchá montáž výrobků snižuje náklady na výrobu a navíc usnadňuje demontáž při opětovném použití, opravě či recyklaci.

Ekologický návrh (eco-design) je zaměřen na fáze předcházející vlastní výrobě, to je především vlastní návrh a vývoj výrobků. Cílem je navržení výrobku v takové podobě, aby se minimalizoval negativní dopad na životní prostředí výrobku i výrobního procesu a aby přitom bylo dosaženo úspor.

Přestože návrh samotný je z pohledu životního prostředí “neškodný” proces, závisí na něm většina ekologických vlivů daného výrobku. Jakmile je návrh výrobku dokončen a je rozhodnuto o výrobních technologiích, zbývají již pouze minimální možnosti, jak dodatečně ovlivnit výkonnost výrobku, jak minimalizovat emise spojené s výrobním procesem a jak všeobecně omezit působení na okolí. Navíc i ty nejmodernější technologie recyklace musí řešit to, co bylo rozhodnuto v průběhu návrhu výrobku.

Přibližně 80% celkového dopadu výrobku na životní prostředí se rozhoduje v průběhu fáze jeho návrhu (obr. 2). S náklady na životní cyklus výrobku je situace podobná. Proto je nesmírně důležité zvážit ekologické a ekonomické stránky hned na samém počátku a uvažovat řešení této problematiky jako neoddělitelnou část návrhu výrobku.

Závěr:

Důležitým krokem pro zahájení jakékoliv činnosti je začátek, který vyžaduje pochopení dané problematiky a také patřičnou dávku motivace. Hlavní motivačním aspektem je dosažení vlastních úspor a tím získání určitých výhod oproti konkurenci. Jako podnět pro inovaci, optimalizaci a progresivní přístup k návrhu nových ekologických výrobků může posloužit dnes již všeobecně známá filozofie „šesti RE“, jež shrnuje základní pravidla v následujících šesti krocích:

- Stále promýšlejte (Re-think) návrh výrobku a jeho funkce, především jak ho vyrobit a využít efektivněji.
- Snižte (Re-duce) spotřebu energie a materiálů během celého životního cyklu výrobku.

- Nahraďte (Re-place) škodlivé látky látkami méně zatěžujícími životní prostředí.
- Umožněte recyklaci (Re-cycle). Vybírejte materiály, které je možné recyklovat, a navrhněte takový výrobek, který se dá snadno rozebrat a recyklovat.
- Znovu použijte (Re-use). Navrhněte výrobek tak, aby se jeho části daly znovu použít při výrobě a servisu.
- Opravujte (Re-pair). Vytvořte takový výrobek, aby se dal snadno opravit a nemusel být nahrazen co nejdříve novým.

8.4 Elektronické přístroje používané v rámci projektu

Jsou to tyto přístroje:

- počítač PC včetně monitoru,klávesnice a myši
- programátor JTAG ICE
- vývojová deska EvB 4.3
- LCD maticový displej
- maticová klávesnice
- krokový motor
- logická sonda
- logický analyzátor
- napájecí zdroj
- propojovací kabely
- individuálně vyrobené učební pomůcky

U těchto přístrojů se předpokládá životnost několik let. Přesto jednou přijde doba, kdy přístroje doslouží a kdy budeme nuceni řešit jejich ekologickou likvidaci. Podívejme se proto, jakým působem lze ekologicky výrobek zlikvidovat.

8.5 Způsoby likvidace elektrotechnického a elektronického odpadu (EEO)

V první řadě bychom měli při poruše zařízení zvážit možnost jeho opravy. Neznamená to, že bychom se měli snažit udržet v činnosti zařízení, která jsou z hlediska spotřeby energie nahraditelná přístroji podstatně úspornějšími. Tím, že zařízení opravíme, minimalizujeme množství elektroodpadu a tím přispějeme výrazně ekologii. Pokud se rozhodneme zařízení již dále nepoužívat, rozhodně nesmí skončit v nejbližším kontejneru. Na nás je tedy vybrat vhodný a především

dostupný způsob likvidace našeho elektroodpadu. V následujícím textu bude uveden příklad tří českých společností, popsána jejich strategie likvidace elektroodpadu a jejich dostupnost.

8.6 Organizace zabývající se zpětným sběrem EEO

V české republice existuje několik organizací, které řeší systémově sběr odpadů včetně EEO. Představme ty hlavní: **Retela, Asekol a Elektrowin**

8.6.1 RETELA , s.r.o.

RETELA, s.r.o. byla založena za účelem provozování kolektivního systému RETELA, na nějž mohou v souladu s §37h odst (1) c) Zákona č. 7/2005 Sb. výrobci elektrozařízení přenést své povinnosti vyplývající ze zmíněného zákona.

Důvodem vzniku bylo zavedení nového režimu pro nakládání s elektrozařízením v Evropě i ČR pro jejich výrobce a dovozce.

Legislativa umožňuje kolektivní plnění povinností výrobců a dovozců EEZ. Vzhledem k tomu, že se výrobci nechtějí nadměrně zatěžovat novou agendou, využívají tuto možnost a volí si partnerství kolektivního systému RETELA.

Kolektivní řešení nakládání s elektroodpadem nejméně zatěžuje výrobce a dovozce elektrozařízení, zefektivňuje administrativu, technickou i finanční stránku nového režimu nakládání s elektroodpadem. Mezi hlavní výhody patří:

- Výrobce splní povinnost zákona administrativně nejjednodušší formou
- Podání návrhu na zápis do Seznamu MŽP provede kolektivní systém
- Roční zprávy a další zákonem dané povinnosti zajišťuje kolektivní systém
- Výrobce nemá povinnost finančního zajištění a nemusí mít účelově vázaný bankovní účet (nemusí poskytovat záruku)

8.6.1.1 Principy činnosti kolektivního systému RETELA

Kolektivní systém RETELA v souladu s §37h odst (1) c) Zákona č. 7/2005 Sb. přebírá zodpovědnost za výrobce a dovozce elektrozařízení (povinnou osobu) za odpad, který vznikne po skončení životnosti elektrozařízení, tj. zajistí jeho sběr, demontáž, recyklaci a následné využití, jak ukládá zákon. Povinná osoba odvádí do systému RETELA recyklační příspěvek, jehož výše je stanovena dle skupiny elektrozařízení, pod kterou povinná osoba spadá, a množství zboží uvedeného na trh v předešlém období.

8.6.1.2 Působnost

Kolektivní systém RETELA pokrývá všech deset kategorií daných legislativou.

1. Velké domácí spotřebiče
2. Malé domácí spotřebiče
3. Zařízení informačních technologií a telekomunikační zařízení
4. Spotřebitelská zařízení
5. Osvětlovací zařízení
6. Elektrické a elektronické nástroje (s výjimkou velkých stacionárních průmyslových nástrojů)
7. Hračky, vybavení pro volný čas a sporty
8. Lékařské přístroje (s výjimkou všech implantovaných a infikovaných výrobků)
9. Přístroje pro monitorování a kontrolu
10. Výdejní automaty

To vše jak v režimu zařízení z domácností, tak profesionálních, a to bez ohledu na datum uvedení na trh.

RETELA, s.r.o. ve smyslu svých stanov nerozděluje zisk společníkovi. Případný zisk se vynaloží na nakládání s elektroodpadem.

RETELA je podporována Svazem průmyslu a dopravy České republiky. Jedním z partnerů je Českomoravská komoditní burza Kladno. Napojení na burzu, která může fungovat pouze se státním dozorem dle zákona 229/1992 Sb. § 5 odst. 1, umožňuje využít burzovní zázemí pro vysokou ochranu a agregaci obchodních informací a pro objektivní cenotvorbu. Toto ve svých důsledcích vede ke snižování nákladů při zachování hlavního cíle, jímž je dosažení kvót sběru a recyklace při dodržení přísných pravidel ochrany životního prostředí.

Základním východiskem pro zavedení nového režimu nakládání s odpadovými elektrickými a elektronickými zařízeními (OEEZ) je to, že množství OEEZ vznikajícího v Evropských společenstvích rychle roste. Obsah nebezpečných součástí v elektrických a elektronických zařízeních představuje závažný problém při nakládání s odpady a k recyklaci OEEZ nedochází v dostatečném rozsahu.

Vzhledem k tomu, že cílem ekologické politiky Společenství jsou zejména zachovat, chránit a zlepšovat kvalitu životního prostředí, chránit lidské zdraví a zajistit racionální využívaní přírodních zdrojů, přijaly Evropský parlament a Rada Evropské Unie Směrnici 2002/96/ES.

Hlavním cílem směrnice je prevence vzniku OEEZ, jeho opětovné použití, recyklace a další formy jeho využívání ve snaze snížit množství odpadu určeného k odstranění. Směrnice 2002/96/ES v příloze IA definuje deset kategorií EEZ spadajících do působnosti této směrnice. V příloze IB je dále specifikován seznam výrobků spadajících do kategorií uvedených v příloze IA.

Proč je nový režim nakládání s elektrozařízením správný? Každý tak pomáháme

chránit životní prostředí a pomáháme návratu cenných materiálů zpátky do oběhu.

Motto: **„každý výrobce je zodpovědný za financování nakládání s odpadem ze svých vlastních výrobků“**

Vysloužilá elektrická a elektronická zařízení lze shrnout pod jedním pojmem elektrošrot. Narozdíl od „klasického“ odpadu tato zařízení velmi často obsahují materiály, které jsou vzácné a které jsou znovu využitelné, např. měď, platina, zlato apod.

Jaké jsou obecné cíle nového režimu nakládání s EEZ?

Prevence v oblasti OEEZ

- Zvýšení opětovného použití, recyklace a ostatních způsobů jejich využití
- Minimalizace hromadění elektroodpadu jako netříděného komunálního odpadu
- Minimalizace používání nebezpečných látek pro výrobu EEZ
- Stimulování spotřebitele k bezplatnému vracení a oddělenému sběru OEEZ
- Využití nejlepších dostupných technik pro zpracování, využití a recyklaci

Problém:

Množství OEEZ rychle roste; jeho nebezpečnost představuje závažnou zátěž pro životní prostředí; recyklace OEEZ je nedostatečná; materiály nenávratně mizí na skládkách nebo ve spalovnách.

Řešení:

Vyrábět pouze taková EEZ, která umožní jejich opětovné použití a usnadní následnou demontáž a recyklaci OEEZ; minimalizovat používání nebezpečných látek pro výrobu EEZ; dosáhnout pro OEEZ vysoké úrovně odděleného sběru; stimulovat spotřebitele k bezplatnému vracení a separovanému sběru OEEZ; dodržovat značení materiálů pro jejich možnou identifikaci; využít nejlepších dostupných technik pro zpracování, využití a recyklaci;

Nástroje:

Zavést odpovědnost výrobce za OEEZ – každý výrobce by měl být odpovědný za financování nakládání s odpady ze svých vlastních výrobků; stát by měl podporovat takové návrhy a výrobu EEZ, které umožní jejich opětovné použití a usnadní následnou demontáž a recyklaci OEEZ; zřídit dostatečný počet sběrných míst k bezplatnému vracení a separovanému sběru OEEZ; přijmout vhodná opatření k dosažení vysoké úrovně třídění komunálního odpadu; zajistit, aby zpracovatelé OEEZ splňovali určité minimální normy pro zpracování a recyklaci OEEZ.

Označování zařízení:

Podle zákona existuje označení dvojího typu:

1. Označení výrobce elektrozařízení
2. Označení elektrozařízení

Obě tato označení musí být umístěna na výrobku nebo v průvodní dokumentaci, a to jedním z níže uvedených způsobů

- vyznačením data uvedení na trh, nebo
- vyznačením symbolu " 8/05" , nebo
- vyznačením grafického symbolu "přeškrtnuté podtržené popelnice"

Grafická podoba symbolu přeškrtnuté popelnice je dána evropskou normou EN 50419.

Pouze pro
Evropskou unii.

Označení výrobce:

- vyznačením obchodní firmy
- uvedením značky, pod kterou výrobce dováží nebo uvádí elektrozařízení na trh a kterou uvede v návrhu na zápis
- evidenčním číslem výrobce v seznamu

Další instrukce:

Není stanovena minimální velikost označení. Je však povinností, aby označení bylo umístěno tak, aby bylo viditelné, čitelné a nesmazatelné při běžném používání výrobku. Pouze není-li to možné lze použít výše zmíněnou průvodní dokumentaci (návody, záruční listy).

RETELA se snaží přispět k plnění cílů stanovených pro Českou republiku.

8.6.2 ASEKOL

ASEKOL je neziskově hospodařící společnost, která v zastoupení výrobců a dovozců elektrozařízení organizuje celostátní systém zpětného odběru elektrozařízení, tj. sběr, dopravu a recyklaci elektrozařízení včetně financování celého systému. ASEKOL je tzv. kolektivní systém zpětného odběru elektrozařízení, jehož služeb mohou na základě smlouvy využít výrobci nebo dovozci elektrozařízení. ASEKOL při zajišťování chodu systému zpětného odběru úzce spolupracuje s městy a obcemi, posledními prodejci a servisy, svozovými společnostmi a zpracovateli elektrozařízení.

ASEKOL byl založen v červenci roku 2005 nejvýznamnějšími představiteli na trhu spotřební elektroniky, kancelářské, telekomunikační a výpočetní techniky. Společníky kolektivního systému jsou firmy:

- ASBIS CZ, spol. s r.o.
- BaSys CS, s.r.o.
- FAST ČR, a.s.
- LG Electronics CZ, s.r.o.
- MASCOM s.r.o.
- Philips Česká republika s.r.o.
- Panasonic Marketing Europe GmbH
- Samsung Electronics Czech and Slovak, s.r.o.
- SONY EUROPE LIMITED

Společnost ASEKOL je na základě rozhodnutí ministerstva životního prostředí (MŽP) jako jediná v ČR oprávněna ke zpětnému odběru historických elektrozařízení ve skupinách 3, 4 a 7. Rozhodnutí MŽP nabylo právní moci dnem 22. 12. 2005.

Dále pak společnost ASEKOL zajišťuje na základě rozhodnutí Ministerstva životního prostředí ze dne 13. 2. 2009, plnění povinností pro zpětný odběr a opětovné použití elektrozařízení, oddělený sběr, zpracování, využití a odstranění elektroodpadu ve skupinách 6 a 9. Rozhodnutí nabylo právní moci dnem 17. 2. 2009.

Dále je kolektivní systém ASEKOL zapsán ze strany MŽP pro tyto skupiny elektrozařízení:

	zařízení historická (vyrobena před 13. 8. 2005)	zařízení do domácností nová (vyrobena po 13. 8. 2005)	zařízení pro podnikání
skupina 3 (výpočetní a kancelářská technika)	Ano (zapsán pouze ASEKOL)	Ano	Ano
skupina 4 (spotřební elektronika)	Ano (zapsán pouze ASEKOL)	Ano	Ano
skupina 6 (Elektrické a elektronické nástroje (s výjimkou velkých stacionárních průmyslových nástrojů)	Ne	Ano	Ano
skupina 7 (hračky, volný čas)	Ano (zapsán pouze ASEKOL)	Ano	Ano
skupina 8 (lékařské přístroje)	Ne	Ano	Ano
skupina 9 (přístroje pro monitorování a kontrolu)	Ne	Ano	Ano
skupina 10 (výdejní automaty)	Neexistují	Neexistují	Ano

ASEKOL velmi úzce spolupracuje s Asociací spotřební elektroniky (ASE). Většina jeho zakladatelů patří mezi členy ASE. V České republice je členem Sdružení veřejně prospěšných služeb. Na mezinárodní úrovni je ASEKOL členem evropské asociace kolektivních systémů Weee forum se sídlem v Bruselu.

Poslání firmy ASEKOL:

Zajistit jménem výrobců a dovozců sběr a ekologické zpracování vyřazených elektrospotřebičů

- být důvěryhodným partnerem pro orgány státní správy a samosprávy
- dbát na efektivitu vynaložených nákladů
- přísně dodržovat kvalitu ekologického nakládání s elektroodpadem
- provádět osvětu široké veřejnosti

Skupiny elektrozařízení:

Společnost ASEKOL je zapsána ministerstvem životního prostředí jako jediný

kolektivní systém pro realizaci zpětného odběru historických elektrozařízení ve skupinách 3, 4 a 7. Dále je ASEKOL oprávněn zajišťovat zpětný odběr pro nová elektrozařízení z těchto skupin:

6. Elektrické a elektronické nástroje
8. Lékařské přístroje
9. Přístroje pro monitorování a kontrolu
10. Výdejní automaty

Klienti:

V kolektivním systému ASEKOL je registrováno okolo 500 výrobců a dovozců elektrozařízení. Počet registrovaných výrobců a dovozců se trvale zvyšuje. Struktura klientů je různorodá – od velkých nadnárodních společností, přes obchodní řetězce až po menší společnosti. Klienti sdružení v kolektivním systému ASEKOL uvádějí na trh cca 40 000 tun nových elektrozařízení ročně.

Systém financování:

Výrobci a dovozci elektrozařízení financují systém prostřednictvím příspěvků. Příspěvky jsou odváděny ASEKOLu čtvrtletně na základě množství elektrozařízení uvedených v daném čtvrtletí výrobcem nebo dovozcem na trh.

Příspěvek se sestává ze tří částí:

- Systémový poplatek – jedná se o paušální poplatek, který kryje náklady na administrativní provoz systému, je pro všechny klienty stejně vysoký.
- PHE – příspěvek na historická elektrozařízení je vyčíslen na kus elektrozařízení uvedených na trh. Slouží k financování likvidace staré zátěže. Je uváděn viditelně a je tedy vybírán od koncových spotřebitelů.
- PNE – příspěvek na nová elektrozařízení je vyčíslen na kus elektrozařízení uvedených na trh. Slouží k vytváření rezervy na budoucí likvidaci nových elektrozařízení. Není uváděn viditelně a je tedy hrazen z prostředků dovozce/výrobce.

Příspěvky na historická a nová elektrozařízení jsou vyčísleny pro každý druh elektrozařízení odděleně. Nedochází tak ke křížovému financování, tj. že by z příspěvků vybraných za jeden druh elektrozařízení byla financována likvidace jiného druhu elektrozařízení. V případě že jsou elektrozařízení, za která byly zaplaceny příspěvky vyvezena do zahraničí, je možné požádat o refundaci příspěvků.

Z vybraných příspěvků je hrazen provoz systému – sběr, doprava a recyklace elektrozařízení a osvětové aktivity.

Systém sběru:

Sběr je realizován prostřednictvím sběrné sítě, která je tvořena:

- Sběrnými dvory a mobilními svozy v obcích
- Prodejnami a opravnami elektrozařízení

Doplňkovými sběrnými místy jsou kontejnery umístěné na veřejných prostranstvích nebo ve veřejných budovách.

Pro firmy a instituce je možný individuální odvoz většího množství elektrozařízení nebo nadměrně velkých elektrozařízení.

V současné době je sběrnou sítí koletivního systému ASEKOL pokryto téměř 90% obyvatel ČR.

Systém dopravy a zpracování:

Pro logistické účely byla ČR rozdělena na 40 svozových oblastí. Svozové společnosti jsou vybírány na základě výběrových řízení, které se opakují každé dva roky. Pro každou svozovou oblast je určen jeden dopravce pro realizaci odvozů ze sběrných dvorů a jeden dopravce pro realizaci odvozu z ostatních sběrných míst. Mobilní svozy jsou prováděny svozovou společností, kterou si vybere daná obec. ASEKOL také využívá služeb kurýrních a zásilkových společností.

I zpracovatelé jsou vybíráni na základě výběrových řízení opakujících se ve dvouletém intervalu. ASEKOL spolupracuje s více než 20 zpracovateli, rozmístěnými rovnoměrně na území ČR. Skoro polovina z nich jsou chráněné dílny, které zaměstnávají pracovníky se změněnou pracovní schopností. Elektrozařízení sebraná v rámci jedné svozové oblasti se předávají vždy jednomu zpracovateli. ASEKOL dbá na ekologickou kvalitu zpracování a to následujícími postupy:

- vydáním závazné směrnice pro zpracovatele (popisuje, jak mají zpracovatelé při zpracování postupovat)
- pravidelnými audity zpracovatelů (minimálně 1x ročně u každého zpracovatele), nejen prostřednictvím vlastních kontrolorů, ale také prostřednictvím auditů od renomovaných auditorských firem
- smlouvou se smluvními sankcemi
- zavedením systému ISO 9001, ISO 14 001 a ISO 18 001

PR a komunikace:

ASEKOL se dlouhodobě věnuje osvětě a vzdělávání v oblasti zpětného odběru elektrozařízení u široké i odborné veřejnosti. Vzhledem k tomu, že aktivity ASEKOLu v oblasti PR a osvěty jsou rozsáhlé a rozmanité, vybíráme jen několik nejzajímavějších projektů.

Pro širokou veřejnost byla určena velmi úspěšná road-show „Nakrmte Šrotozemšťana“, která byla oceněna v soutěži Česká cena za public relations. Ve spolupráci s Českou televizí natočil ASEKOL pětidílný televizní seriál “Kam s nimi”. Pro školní mládež organizoval ASEKOL ve spolupráci s organizací Ecobat školní zábavný program “Hrátky s Asíkem a Batem”.

Pro odbornou veřejnost ASEKOL v roce 2011 uspořádal již třetí ročník konference “Zpětný odběr”, jejíž kvalitu dokládaly pozitivní ohlasy 500 domácích i zahraničních návštěvníků. Pravidelně je vydáván časopis “Zpětný odběr”, který byl ohodnocen cenou “Zlatý středník”, jako nejlepší časopis státní, veřejné a neziskové sféry. Pozornost veřejnosti si získala interaktivní putovní výstava Muzeum spotřebičů. Úspěchy sklízí také dlouhodobý školní vzdělávací projekt Recyklohraní, do kterého bylo ke konci roku 2011 zapojeno zhruba 750 000 žáků a studentů.

ASEKOL pořádal rovněž projekt „Zahoď mobil“, neboli mistrovství republiky v hodu mobilním telefonem. Jednalo se o unikátní zábavně soutěžní road-show v 13 krajských městech, která vyvrcholila v červnu 2012 v podobě celorepublikového finále. Cílem bylo zábavnou formou propagovat třídění mobilních telefonů a dalších vysloužilých elektrozařízení. Ve výčtu osvětových aktivit nemůže chybět ani rozsáhlá kampaň na podporu červených stacionárních kontejnerů – Najdi si svůj červený kontejner, která má za sebou jarní část a celkem oslovila nebo ještě na podzim osloví obyvatele 52 českých a moravských měst.

8.6.3 Elektrowin

ELEKTROWIN, a.s. provozuje kolektivní systém pro zpětný odběr, oddělený sběr, zpracování, využití a odstranění elektrozařízení a elektroodpadu (velké domácí spotřebiče, malé domácí spotřebiče, elektrické nástroje a nářadí). Společnost je nezisková, jejími akcionáři jsou přední výrobci velkých a malých domácích spotřebičů.

Díky našemu kolektivnímu systému je po celé České republice k dispozici **kolem 9 000 sběrných míst**, na kterých lze odevzdat **elektrospotřebiče k ekologické recyklaci**. Spolupracujeme s více než 900 sběrnými dvory, elektrická zařízení také můžete odevzdat v prodejnách domácích spotřebičů, ve školách nebo i v rámci mobilních svozů odpadů.

Poskytujeme Wintejnery – speciální kontejnery na zpětně odebrané elektrozařízení / elektroodpad, které usnadňují manipulaci a zajišťují ochranu elektrospotřebičů na sběrných dvorech.

Ve spolupráci s kolektivními systémy realizujeme projekt pro školy s názvem „Recyklohraní aneb Ukliďme si svět!“. Školy, které shromáždí nejvíce vysloužilých elektrospotřebičů, získávají sportovní potřeby, učební pomůcky, atraktivní výlety pro žáky a další zajímavé ceny. Do systému sběru elektroodpadu se také zapojují sbory dobrovolných hasičů.

Naším cílem je maximální využití surovin získaných z elektroodpadu. V současnosti **dosahujeme výtěžnost recyklace až 90%**. Naší snahou je proto i nadále přibližovat sběrnou síť k domácnostem.

Vysvětlení některých základních pojmů:

Co je recyklační příspěvek

Zákazník, který chce zakoupit elektrozařízení, by měl na účtence, faktuře, cenovce nebo letáku vystaveném na prodejně vždy nalézt informaci o výši recyklačního příspěvku, tedy PHE.

Částka recyklačního příspěvku je určena na financování zpětného odběru elektrozařízení tj. sběr, svoz a recyklaci. Do kolektivního systému jí odvádí výrobci resp. dovozci elektrozařízení.

Pojmy:

Recyklační příspěvek

- slouží k financování sběru, svozu a zpracování tzv. historických elektrozařízení, touto částkou výrobci přispívají do kolektivního systému.

Kolektivní systém

- zajišťuje povinnosti výrobců v oblasti zpětného odběru, zpracování a využití

elektrozařízení.
Rozlišujeme dva druhy recyklačních příspěvků:

- **PHE - příspěvek na historická elektrozařízení, tzv. viditelný příspěvek:** je příspěvek na recyklaci historických elektrozařízení (vyrobených před 13. 8. 2005), tato částka by měla být uváděna viditelně tj na cenovce výrobku, letáku nebo na dokladu vydávaném při prodeji.
- **PNE -** příspěvek za recyklaci nových elektrozařízení (vyrobených po 13. 8. 2005). Tato částka není uváděna viditelně.

Symbol přeškrtnutá popelnice

Pro účely separovaného sběru elektroodpadu a zpětného odběru elektrozařízení je výrobce povinen označit elektrozařízení grafickým symbolem („přeškrtnutá popelnice“).

Není-li možné elektrozařízení takto označit vzhledem k jeho velikosti nebo funkci, označí se grafickým symbolem obal nebo návod k použití nebo záruční list elektrozařízení.

Výrobce elektrozařízení uvedeného na trh po 13. srpnu 2005 (kdy nabyla účinnosti novela zákona o odpadech zapracovávající předpisy EU, která upravuje mimo jiné nakládání s elektrozařízením z domácností a jeho kategorizaci) má dále povinnost zajistit, aby z označení elektrozařízení bylo patrné, že bylo na trh uvedeno po tomto datu, a bylo možné zjistit jeho výrobce.

Tento symbol znamená ,že výrobek je určen k recyklaci a pro sběrná místa :

Zařízení, které bylo uvedené na trh do 13. srpna 2005, se označuje jako tzv. „historické elektrozařízení“ a nemá uvedené značení.

Co je elektrozařízení

Přesnou definici „elektrozařízení“ neboli elektrické či elektronické zařízení najdeme v zákoně o odpadech § 37g. Zjednodušeně můžeme tedy elektrozařízením označit malý i velký domácí elektrospotřebič „vše co lze zapojit do elektrické zásuvky nebo funguje na baterie“

Jsou stanoveny i výjimky spotřebičů, které do zpětného odběru nepatří: např.

bojlery, klimatizační jednotky, pokud jsou stálou, pevnou a nedílnou součástí staveb, akumulační kamna….

Elektrozařízení jsou podle zákona rozděleny do 10. skupin:

1. Velké domácí elektrospotřebiče (pračky ledničky , sporáky apod.)
2. Malé domácí elektrospotřebiče (vysavače, topinkovače , fritovací hrnce apod.)
3. Zařízení informačních technologií a telekomunikační zařízení (notebooky, mobily)
4. Spotřebitelská zařízení (video, audio, hudební nástroje)
5. Osvětlovací zařízení (žárovky, trubice,výbojky)
6. Elektrické a elektronické nástroje (vrtačky, pily, zahradní technika)
7. Hračky, vybavení pro volný čas a sporty (hračky,vybavení pro sport a volný čas)
8. Lékařské přístroje (lékařská technika)
9. Přístroje pro monitorování a kontrolu (detektory, termostaty..)
10. Automaty

Jaký elektrospotřebič můžete odevzdat zdarma?

Ke zpětnému odběru lze odevzdat jen kompletní a nerozebrané elektrospotřebiče z domácností nebo jemu podobné od právnických osob a podnikatelů (tzn., že pokud máte např. ledničku nebo rychlovarnou konvici v kanceláři a dosloužíl, můžete ji bezplatně odevzdat na kterémkoliv místě zpětného odběru a pokud budete potřebovat, obsluha vám vystaví potvrzení).

Ze zákona vyplývá, že elektrozařízení pocházející z domácností je takové elektrozařízení, které svým stavem odpovídá výrobku s ukončenou životností bez bližšího určení důvodu. Stejně jednoznačně lze určit, že takovým elektrozařízením není výrobek, který již prošel jakoukoli fází ‚zpracování' či dílčího využití, tedy činností, která je zákonem určena pouze osobám s příslušnými oprávněními. Takové elektrozařízení, tedy nekompletni , musí být posuzováno jako odpad a pro zacházení s ním nelze využívat výhod pro zpětný odběr elektrozařízení.

Nekompletní spotřebiče jsou tedy považovány za odpad a náklady spojené s jejich recyklací a odstraněním jsou k tíži obci, následně se tedy promítají do poplatku za odpady.

Elektrospotřebič odevzdaný na místo zpětného odběru se stává odpadem až ve chvíli předání osobě oprávněné k jeho využití nebo odstranění, tedy do rukou specializované recyklační firmy.

Jak sbírat a seznam sběrných míst

V ČR již v současnosti existuje více než 12 000 sběrných míst, kam mohou lidé odnést vysloužilý elektrospotřebič. Sběrná místa jsou tvořena obecními sběrnými dvory, mobilními svozy a koncovými prodejci, neboli prodejnami s elektrospotřebiči. Dále také ELEKTROWIN spolupracuje i se servisy, se školami v rámci projektu Ukliďme si svět a se sbory dobrovolných hasičů pod hlavičkou programu Recyklujte s hasiči.

Označení místa zpětného odběru:

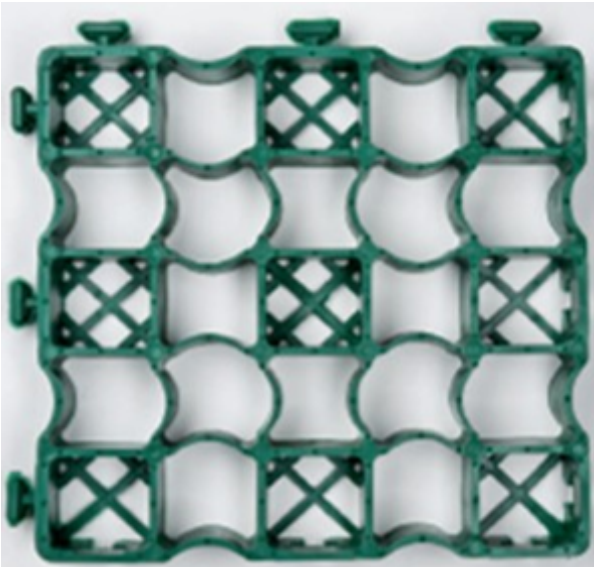
Co se s ním stane dál?

Ze sběrných míst jsou vysloužilé elektrospotřebiče odvezeny do zpracovatelských zařízení, kde jsou přístroje nejprve rozebrány na jednotlivé komponenty, ze kterých jsou odstraněny součásti obsahující nebezpečné látky (např. kondenzátory, baterie, atd.). Následně se součásti bez nebezpečných látek drtí a třídí podle druhu materiálu pro recyklaci. Recyklovatelné materiály jsou použity pro výrobu nových produktů, kde nahrazují primární suroviny .

Příklady nových produktů:
Disk na kola

Střecha s PUR- pěnou

Zatravňovací dlaždice



Z čeho jsou elektrospotřebiče vyrobeny?

Elektrospotřebič, jak ho známe v dnešní době, je v zásadě složen ze tří hlavních složek. Nejpodstatnější částí elektrospotřebiče je železo, které tvoří jak základ pro design, tak je především hlavním materiálem funkčních částí elektrospotřebiče.

Další důležitou součástí každého elektrospotřebiče jsou plasty, jež jsou nejen součástí designu, ale také v mnoha případech slouží jako bezpečnostní či funkční součást (např. jako poličky v ledničce).

A v neposlední řadě je to také sklo, které je z větší části použito na doplnění vzhledu nebo jako izolační materiál.
Typickou ukázkou elektrospotřebiče je například mikrovlnná trouba

Podíl jednotlivých materiálu je závislý na druhu elektrospotřebiče. V zásadě se však dá říci, že železo, sklo a plasty tvoří jeho většinovou část.

Je také dobré vědět, že součástí elektrospotřebiče jsou i nebezpečné látky, které v žádném případě není možné zlikvidovat například pouhým postavením lednice vedle popelnice. Každé elektrozařízení obsahuje nebezpečné látky, jež je nutné profesionálně zrecyklovat.

Kde končí elektrospotřebiče?

Sběrné dvory

místa zpětného odběru elektrozařízení jsou tak vytvářena ve sběrných dvorech a na sběrných místech obcí (sběrný dvůr je zařízení města/obce a je přístupný pro odevzdání veškerého elektrozařízení z domácnosti, kterého se chcete zbavit)

1056 sběrných dvorů v 920 obcích

Mobilní svozy

Pokud nemáte v obci sběrný dvůr, můžete využít možnosti mobilního svozu nebo tzv.:

Putující kontejner

Putující kontejner je program kolektivního systému ELEKTROWIN pro

mikroregiony a svazky obcí. Spočívá v přistavení kontejneru na místo určené těmito subjekty dle připraveného harmonogramu.

> 4000

Prodejce elektro

Prodejce je povinen odebírat spotřebiče pouze kus za kus – tedy v případě, že nakupujete zařízení stejného typu a funkce

> 2600

Malý kontejner

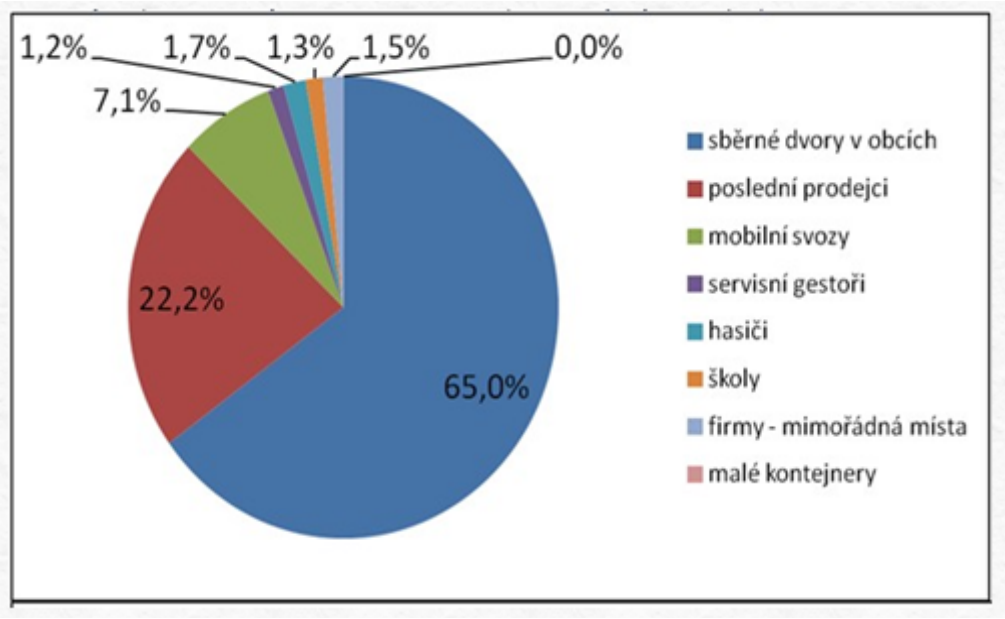
Je umístěný zpravidla u sběrných hnízd, tedy tam, kde se nacházejí sběrné nádoby na papír, plasty, sklo apod. Je určen výhradně pro malé elektrospotřebiče (příklady - skupiny 2 a 6). Do kontejneru se nesmí vhazovat světelné zdroje (žárovky, zářivky, výbojky), cartridge a elektrozařízení obsahující obrazovku (PC monitory, TV).

Netradiční sběrná místa

> 1 300

Elektrowin dále spolupracuje i se:

- servisy
- školami v rámci projektu **Recyklohraní** aneb **Ukliďme si svět**
- sbory dobrovolných hasičů v rámci programu **Recyklujte s hasiči**



Podíl odběru elektrospotřebičů na jednotlivých sběrných místech

Nebezpečné látky

Může se Vám zdát, že z nefunkčního spotřebiče se dá ještě ledascos využít – šroubek, klička, motor, kompresor – víte ale, co tím můžete způsobit?

Ke zpracování elektrozařízení musejí firmy získávat obtížně souhlas krajského úřadu – musí splnit spoustu podmínek – stavební úpravy v dílnách, technické vybavení, odbornou způsobilost. Při demontáži musí dodržet zákonem stanovený postup, odejmout všechny části obsahující nebezpečné látky tak, aby neohrozily ani zdraví ani životní prostředí.

Nebezpečné? Ano, ale jen při nesprávném zacházení!

Elektrospotřebiče obsahují takové látky, jako je kadmium, rtuť, olovo, šestimocný chrom, polybromové bifenyly, polybromové difenyletheny nebo azbest.

V chladničkách se nacházejí také freony a to jak v chladicím okruhu, tak ve většině případů také v izolační pěně v korpusu chladničky. Pokud dojde k poškození okruhu – např. odstřižením kompresoru nebo chladící mřížky, uniká nejen olej, ale právě i freon do ovzduší.

Proto, aby nedocházelo k poškozování zdraví a životního prostředí, uložila Evropská unie výrobcům a dovozcům elektrozařízení, aby tyto látky již nadále pro výrobu nových spotřebičů nepoužívali, případně pro jejich použití tam, kde jsou nenahraditelné, omezili na nezbytné minimum.

Ve starších spotřebičích jsou ale stále! Pokud z takového spotřebiče někdo odmontuje to, co si myslí, že jednou použije nebo o čem je přesvědčen, že se podaří výhodně prodat, vystavuje sebe i své okolí zdravotnímu riziku.

Odevzdávejte na místa zpětného odběru kompletní spotřebiče! Chráníte tím zdraví sobě i svému okolí! Seznam míst zpětného odběru naleznete na: www.elektrowin.cz

Ochrana ozónové vrstvy

V období let 1960–1985 začala prudce narůstat spotřeba látek, které poškozují ozonovou vrstvu, a to zejména chlorfluoruhlovodíků (CFC).

Protože přímá souvislost mezi vzrůstající spotřebou CFC a úbytkem stratosférického ozonu byla objevena až v polovině sedmdesátých let, tyto látky pro své velmi dobré technické vlastnosti ve velkém měřítku pronikly do mnoha odvětví lidské činnosti a objem jejich spotřeby nadále vzrůstal. Až když vznikla rozsáhlá ozonová díra nad Antarktidou a začal prudce stoupat výskyt kožních nádorových onemocnění.

Zdaleka nejvýznamnějším odvětvím používání látek, které poškozují ozonovou vrstvu Země, je chladicí a klimatizační technika. Jedná se o komplexní obor, který dnes zasahuje téměř do všech oborů lidské činnosti tím, že ovlivňuje řadu průmyslových a hospodářských odvětví a některé z nich přímo podmiňuje. Jedná se především o potravinářský a chemický průmysl, klimatizaci, zdravotnictví, kulturu a sport.

V oboru chladicí techniky byly a stále ještě jsou používány jako chladicí média dvě skupiny látek, které významným způsobem poškozují ozonovou vrstvu Země.

První skupinou jsou chlorfluoruhlovodíky (CFC), které se řadí k nejnebezpečnějším látkám z hlediska potenciálu poškozování stratosférického ozonu.

Druhou skupinou jsou hydrochlorfluoruhlovodíky (HCFC), které patří k méně rizikovým a jejichž potenciál poškozování ozonu je řádově nižší (pětkrát až padesátkrát, podle druhu látky).

Pokud jde o zařízení staršího data výroby (zejména se jedná o chladicí zařízení vyrobená před rokem 2000), je v nich stále ještě možno jako chladivo používat HCFC a tyto látky je rovněž možno používat také při jejich servisu a to až do roku 2014.

Mezi největší rizika pro zdraví člověka patří:

- zvýšení výskytu nádorových onemocnění kůže (tzv. spálení kůže při opalování, zrychlené stárnutí kůže, kožní nádorová onemocnění)
- narušení imunitního systému
- poškození zraku: sněžná slepota – jedná se o reakci rohovky na zvýšenou expozici záření typu UVB, které způsobuje zarudnutí očí, přecitlivělost na světlo, bolesti a může v některých případech rovněž vést ke vzniku nádorových onemocnění, šedý zákal – toto onemocnění může vést až k úplnému oslepnutí
- UVB-záření má navíc schopnost aktivovat některé druhy virů, mezi jinými např. HIV, viry tuberkulózy nebo lymské boreliózy

Co se vyrábí dál?

Nové přicházejí, staré se recyklují

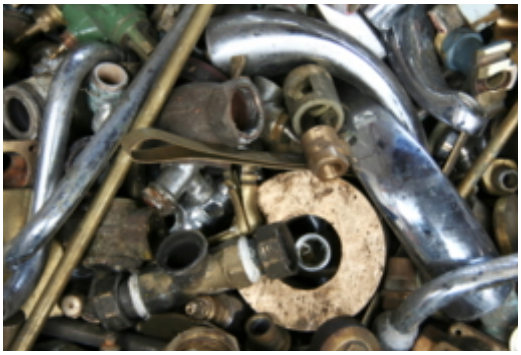
S příchodem nových spotřebičů do domácností se z našich starších elektro

pomocníků stává spíše elektroodpad. Někdy nás ke koupi nového spotřebiče donutí porucha stávajícího, jindy s námi cloumá čistě jen chuť pořídit si něco nového. Jak ale správně naložit s vysloužilci? Recyklovat! To však stále ještě není pravidlem. Někteří raději uloží rozbitý spotřebič do skříně takříkajíc „na pak“, jiní chybně odloží např. plastovou rychlovarnou konvici do popelnice na plastové láhve nebo ji hodí přímo do popelnice s komunálním odpadem.

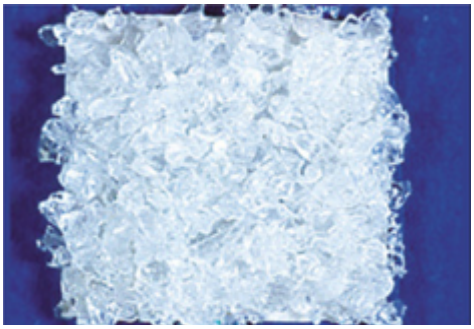
Přitom je to tak snadné. Odevzdáním spotřebiče na správné místo lze znovu využít až 90% materiálů. Výroba nových spotřebičů či dalších výrobků je pak mnohem efektivnější a přívětivější k životnímu prostředí.

Výrobky z materiálů, které jsou ve spotřebičích zastoupeny v největší míře:

- plast - plotové laťky, zatravňovací dlaždice; součásti nových elektrospotřebičů – např. kryty (vysavače), roury; plastové prvky dětských hřišť;
- železo – cokoliv ze železa - trubky, rámy nebo se zpracovává na ocel: dráty, plechy, kolejnice, hřebíky, mříže, dráty, sudy, plechovky, lana,
- barevné kovy - MĚĎ - měděné dráty, střešní krytiny, okapy, nebo se používá k výrobě bronzu (mince, medaile)či mosazi; NIKL - ochrana proti korozi – např. u šroubováků nebo klíčů, chirurgických nástrojů, nikl se používá v bateriích; HLINÍK - mince, alobal, nádobí, CDčka, beton (silnice)



- ***sklo*** – technické sklo (např. police do ledniček, talíř do mikrovlnky), stavebnictví-izolace,



Dalším velmi důležitým aspektem je úspora energie a primárních surovin, jako např. železné a jiných rud, ropy. Ve spotřebičích je spousta materiálů, které po správném a odborném zpracování dokáží tyto suroviny nahradit.

Co jsou akumulátory a baterie:

Akumulátor

Je elektrochemický zdroj elektrické energie. Pro upřesnění je nutné dodat, že se jedná o zdroj stejnosměrného elektrického proudu. Základní vlastnost akumulátoru, či schopnost chceme-li, je akumulace elektrické energie (kumulovat čili hromadit, odtud název akumulátor).

Baterie

Je bezesporu nejrozšířenější a nejpoužívanější termín pro chemický zdroj elektrické energie. Bývá často, avšak nesprávně, používán v souvislosti s obyčejnými tzv. primárními články (např. AA tužkové baterie, apod.). Výrazem BATERIE lze totiž správně označit pouze elektrochemický zdroj, sestavený z více článků.

Zajišťování **zpětného odběru přenosných baterií v České republice** se zabývá kolektivní systém ECOBAT (www.ecobat.cz).

Kam se starými?

Použité baterie a akumulátory, nesprávně vyhazované s běžným odpadem, mohou vážně narušit životní prostředí. Po čase se z nich uvolňují škodlivé látky (zejména

tzv. těžké kovy), které mohou znečistit půdu nebo spodní a povrchové vody. Těžké kovy obsažené v bateriích mají prokazatelně škodlivý vliv na lidské zdraví.

Použité a vybité baterie nebo akumulátory můžete bezplatně odevzdávat k likvidaci a recyklaci na mnoha místech, která jsou označena jako „místa zpětného odběru“. Nebo je můžete odevzdat společně s elektrozařízením.

Použité články a baterie (např. tužkové baterky z dálkového ovládání, knoflíkové baterky s náramkových hodinek, dobíjecí akumulátor z mobilních telefonů), můžete odevzdat na nejen následujích pobočkách prodejen:

- maloobchodech s potravinami
- prodejny a hypermarkety ELEKTRO
- hobby centra
- prodejny s drogistickým a jiným zbožím

- na školách
- na veřejných místech
- ve sběrných dvorech apod.

8.7 A co na to my?

A nyní už jen zbývá vybrat si vhodný způsob, jak s naším elektroodpadem naložit. V uvedeném přehledu organizací si rozhodně vybereme. Rozhlédněme se po okolí našeho bydliště, jistě najdeme ne jedno sběrné místo, kam elektroodpad uložit s vědomím, že jsme udělali něco užitečného a velmi důležitého pro životní prostředí.

9 Závěr

Účelem a cílem těchto skript je přijatelným způsobem uvézt žáky naší odborné školy do problematiky mikroprocesorové techniky, konkrétně využitím mikrokontroléru ATmega32 výrobce ATMEL.

Nově oproti výuce v minulých letech, kdy jsme používali více než 30 let starý mikrokontrolér AT89c51, je použití mikrokontroléru s novou architekturou RISC se zabudovaným rozhraním JTAG. Toto rozhraní spolu s vhodným programátorem a vývojovým prostředím umožňuje pohodlné ladění úloh.

Další novinkou ve výuce je použití pro aplikaci mikrokontrolérů vhodnějšího programovacího jazyku C oproti dosud používanému použití jazyku symbolických adres. Jsem přesvědčen, že právě aplikace jednočipových mikrokontrolérů díky častému ladění úloh je velmi vhodným prostředkem pro prvotní seznámení s programovacím jazykem C.

Jsem si vědom, že ukázkové úlohy použité ve skriptech nejsou příliž složité a nemají mnohdy praktický význam. Snažil jsem se neodradit žáky příliš složitými úlohami, vybral jsem úlohy, které obsahují obsluhu běžných periferií, jako jsou spínače, LED indikátory, 7segmentový displej LED, dvouřádkový displej LCD, maticová klávesnice, krokový motor apod.

V ukázkových příkladech jsou použity některé vnitřní bloky mikrokontroléru ATmega32, jako např. I/O porty, přerušovací systém, čítače/časovače, generátor PWM signálu, A/D převodník, analogový komparátor.

Ve skriptech je uvedeno mnoho zadání úloh pro samostatné řešení žáky. Tyto úlohy lze často realizovat úpravou uvedených ukázkových příkladů. Někdy jde o relativně snadnou úpravu, v mnoha úlohách je však nutné poněkud jiné, originální řešení. Je to způsob, který donutí žáky se zamyslet a najít "to svoje řešení". Tímto začíná proces učení.

Při řešení některých úloh je zapotřebí provádět odborná měření. Např. při generování přesného kmitočtu nebo časového intervalu využitím čítačů/časovačů, při měření zákmitů spínačů, při použití A/D převodníku a pod. Při těchto úlohách se žáci seznámí s použitím moderní měřicí techniky, jako např. digitálního paměťového osciloskopu, generátoru signálů, logického analyzátoru, logické sondy.

V uvedených příkladech nejsou využity všechny vnitřní bloky mikrokontroléru ATmega32, stejně tak nejsou využity všechny periferie vývojového kitu EvB 4.3. Též programovací techniky použité v ukázkových příkladech odpovídají úrovni žáka, který s mikroprocesorovou technikou začíná. Podle prvních zkušeností při výuce vzbudila výuka této techniky u mnoha žáků velký zájem.

Autor

Seznam použité literatury:

1. B.Maan: „C pro mikrokontroléry“, BEN – technická literatura, Praha 2003
2. D.Matoušek: „Práce s mikrokontroléry ATMEL AVR“, BEN – technická literatura, Praha 2006
3. D.Matoušek: „C pro mikrokontroléry ATMEL AT89S52“, BEN – technická literatura, Praha 2007
4. P.Herout: „Učebnice jazyka C“, KOPP, 1994
5. V.Váňa: „Mikrokontroléry ATMEL AVR – programování v jazyce C“, BEN – technická literatura, Praha 2003
6. Rowley Associates Ltd.: „CrossWorks for AVR Reference Manual“,http://www.rowley.co.uk
7. Atmel Corporation: „Datasheet mikrokontroléru Atmega32“, http://www.atmel.com
8. http://www.asekol.cz
9. http://www.retela.cz
10. http://www.elektrowin.cz
11. http://www.mzp.cz/cz/elektrozarizeni
12. http://martin.feld.cvut.cz/
13. http://www.umel.feec.vutbr.cz/