Středoškolská technika 2013

Setkání a prezentace prací středoškolských studentů na ČVUT

Sada úloh pro výuku MIT

Lukáš Holek

SPŠ a VOŠ Písek

Karla Čapka 402, Písek

Anotace

Práce je zaměřená na vytvoření sady úloh s mikroprocesorem firmy Atmel a to s mikroprocesorem ATmega32. V této práci jsou souhrnně popsány v dnešní době požívané mikroprocesory různých výrobců, vývojové prostředí AVR Studio a mikroprocesor ATmega32. Na několika úlohách je ukázána práce s tímto mikroprocesorem. Tyto úlohy jsou odzkoušeny na UNI desce, která byla pro tento účel zkonstruovaná.

Abstract

The objective of this work is to make a set of tasks with the Atmel microprocessor ATmega32. This work comprehensively describes currently used microprocessors from various producers, development environment AVR Studio and microprocessor ATmega32. Using of this microprocessor is demonstrated on several tasks. These tasks are tested on UNI board, which was designed for this purpose.

Klíčová slova

UNI deska, ATmega32, AVR Studio 4, jazyk C

Prohlášení

Prohlašuji, že jsem svou práci vypracoval(a) samostatně a použil(a) jsem

pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon)

Poděkování

Chtěl bych poděkovat Mgr. Milanu Janouškovi za jeho rady a pomoc při tvorbě této práce.

Obsah

Úvod

Cílem této práce bylo vytvořit sadu úloh v libovolném programovacím jazyce s libovolným 8 – bitovým mikroprocesorem. K programování byl zvolen vyšší programovací jazyk C a mikroprocesor ATmega32. Úlohy jsou směrovány většinou na ukázku praktického využití, čímž jsou úlohy pro studenty atraktivní a zábavné. V úlohách jsou podrobně popsány jednotlivé práce s periferiemi, takže by měli studenti pochopit práci s periferiemi, jako jsou např. LCD displej, 7 – segmentový displej, AD převodník, LED diody a tlačítka.

K praktickému předvedení činnosti popsaných úloh, byla vytvořena UNI deska s mikroprocesorem ATmega32 . UNI deska obsahuje podpůrné periférie, jako jsou LCD displej, 7 – segmentový displej, LED diody a tlačítka. V práci je také stručně popsáno základní programování v jazyku C, které by mělo naučit studenty základní programování procesoru ATmega32.

1 Přehled současně používaných 8 - bitových MCU

1.1 ATMEL

Tyto mikrořadiče jsou typu RISC (zredukovaná instrukční sada) a jsou postavené na Harvardské architektuře (oddělená paměť dat a paměť programu). Paměť programu je typu Flash. Většina instrukcí má délku 16 bitů, stejně tak slovo programu má délku 16 bitů. Jádro AVR se skládá z třiceti dvou stejných 8 - bitových registrů, které mohou obsahovat data i adresy. Přístup do pole registrů je možný v jednom strojovém cyklu. To znamená, že lze v jednom strojovém cyklu vykonat jednu aritmeticko-logickou operaci. V jednom strojovém cyklu se oba operandy načtou, provede se operace a výsledek se zapíše zpět do pole registrů.

Požívají se v mnoha odvětvích od automobilového průmyslu po zdravotnickou techniku.

Stránky výrobce: http://www.atmel.com/

Řada tinyAVR

Specifické vlastnosti:

- **Malý** (nejmenší měří pouze 1,5 mm x 1,4 mm)
- **Kapacitní dotyk (***QTOUCH ® Knihovna* umožňuje snadné připojení dotykového kapacitního tlačítka a posuvníku, umožňuje flexibilitu i efektivitu při dotykových aplikací)

obr. 1 - ATtiny13A-PU[2]

- **Vysoká integrace**
- **Schopný provozu na pouhých 0,7V**

Tyto MCU obsahují jeden nebo dva osmibitové čítače, některé i šestnáctibitový čítač. Většina typů obsahuje 10 - bitový AD převodník, dva nebo čtyři PWM kanály, SPI, TWI a analogový komparátor. Počet pinů je podle typu od 6 do 32). Tato řada se využívá především v jednoduchých a malých elektronických obvodech.

Typy:

Z rodiny tiny AVR:

a) Nejmenší: jsou mikrokontroléry ATtiny4, ATtiny5, ATtiny9 a ATtiny10. Tyto MCU mají pouze 6 až 8 vývodů, jejich maximální provozní frekvence je 12 MHz a velikost paměti Flash mají 512 B až 1 kB. ATtiny5 a ATtiny10 mají navíc ještě AD převodník.

b) Největší: jsou mikrokontroléry ATtiny828, ATtiny48, ATtiny88. Tyto MCU mají maximální provozní frekvenci 12MHz a typ ATtiny828 má až 20 MHz. Disponují flash pamětí 8 kB. Počet vývodů je až 32.

Řada megaAVR

Specifické vlastnosti:

- **široká rodina**- nabízí nejširší výběr zařízení, pokud jde o paměti, počet pinů a periferií, což umožňuje použití kódu a znalostí na téměř jakémkoliv MCU z této rodiny.
- **picoPower**- technologie která umožňuje u vybraných MCU snížit velmi razantně spotřebu, volitelné klidové režimy
- **Vysoká integrace**- nabízí flash, SRAM, interní EEPROM, SPI, TWI, USART, hlídací časovač, interní oscilátor
- **Analogové funkce**- AD a DA převodníky, vestavěný teplotní senzor, vnitřní zdroj referenčního napětí, rychlý analogový komparátor

Tyto MCU obsahují 2 osmibitové čítače, 1 až 4 šestnáctibitové čítače, 10bitový AD převodník, 3 až 16 PWM kanálů, SPI, TWI, USART, EEPROM (256 B až 4 kB) a analogový komparátor. Počet pinů je podle typu od 23 do 86.

Typy:

Např. ATmega8 (obr. 2 a obr. 3), ATmega16, ATmega32

obr. 2 - ATmega8A-PU pouzdro DIL ***obr. 3 – ATmega8 v pouzdře pro SMD[2]***

1.2 Microchip Technology

Mají oddělenou paměť programu a dat, architektura RISC (nízký počet instrukcí). Délka slova programu je podle typu MCU 12, 14 nebo 16 bitů. Všechny instrukce mají jednotnou délku odpovídajícímu slovu. To znamená, že v každém paměťovém místě je uložena právě jedna instrukce.

Napájecí napětí se pohybuje od 1,8 V do 5,5 V. Proudová spotřeba při 4MHz je méně než 2mA. Typy s nanoWatt technologií mají spotřebu 10x nižší.

Tyto mikrořadiče obsahují obousměrné I/O porty. Jednotlivé piny portů je možno za pomoci řídících registrů nastavit jako výstupní nebo vstupní.

Stránky výrobce: http://www.microchip.com/

MCU této firmy se vyznačují:

- kompatibilita kódu napříč všemi řadami
- vysoké rozlišením PWM, ADC, DAC
- provoz už při napětí 1,8 V
- 375 B až 128 kB Flash paměť
- 16B až 14 kb RAM

- Číslicové řízení oscilátoru
- flash, SRAM, interní EEPROM, SPI, TWI, USART, I2C
- nízké odběry proudu při režimu spánku, cca 20 nA
- aktivní proudy méně jak 35 µA / MHz

Řady

V 8 - bitových MCU má firma Microchip Technology čtyři rodiny:

- ***Baseline*** – základní řada, 6 až 40 vývodů, délka instrukce 12 bitů, (PIC10Fxxx, PIC16F5x)
- ***Mid-Range*** – střední řada, 8 až 64 vývodů, délka instrukce 14 bitů, (PIC12F6xx, PIC16Fxxx)
- ***Enhanced Mid-Range*** – vylepšená střední třída, zvětšená kapacita pamětí, nové periférie, (PIC12F1xxx, PIC16F1xxx)
- ***PIC18*** – nejvyšší řada, 18 až 100 vývodů, délka instrukce 16 bitů, optimalizace architektury pro překlad programů z jazyka C, (PIC18Fxxx)

1.3 Freescale Semiconduktor

Tato firma (dříve patřící pod firmu Motorola) se specializuje hlavně na 8 a 16 - bitové mikropočítače. Vyrábí však i 32 a 64 - bitové procesory pro použití v serverech (řada ColdFire).

Stránky výrobce: http://www.freescale.com/

Řady 8 – bitových MCU:

- HC05
- HC08
- HC11
- RS08

Vlastnosti MCU z řady HC08 - 68HC908QY4CPE:

Tento MCU je CISC s Von Neumannovou architekturou. Tento typ má 16 pinů. Paměť RAM je velká 128 bytů a Flash 4kB. Pracuje na maximální frekvenci 3,2MHz z interního oscilátoru. A/D převodník má 4 kanály s rozlišením 8 – bitů. Dále má jeden 16 – bitový čítač. Vyrábí se v pouzdrech DIP, SOIC, TSSOP.

Obr. 4 – MCU Freescale 68HC908QY4CPE[4]

1.4 STMicroelectronics

Tato firma se specializuje na procesory s malou spotřebou. Procesory vyrábí hlavně pro automobilový průmysl (především v oblasti převodovky a ochranných prostředcích v automobilech). Ale můžeme je najít i v běžné elektronice (např. DVD přehrávač, Set-Top Box atd.).

Jedna z největších světových firem zabývajících se s polovodiči. Spolupracuje např. s firmou Samsung Electronics.

Stránky výrobce: http://www.st.com

Řady 8 – bitových MCU:

- STM8A – automobilové MCU
- STM8L – energeticky málo náročné MCU
- STM8S – tradiční MCU
- STM8T – MCU pro práci se senzory

Vlastnosti MCU ST62E20C:

Jádro ST6 nabízí 40 základních instrukcí. MCU obsahuje paměť RAM 64B, EPROM 4096 B, 8 – bitový A/D převodník se 13 analogovými vstupy, 8 – bitový čítač/časovač a 21 I/O pinů. Maximální pracovní frekvence je 8 MHz. Tento MCU je určen pro středně složité aplikace. V pouzdru má celkem 28 pinů.

Obr. 5 – MCU ST62E20C [5]

1.5 NXP Semiconductors

Mikroprocesory této firmy (dříve patřící pod firmu Philips) poskytují vysoký výkon. Využívají se především v automobilovém průmyslu, v různých identifikačních systémech, v bezdrátové komunikaci a ve spoustě dalších oblastí. Firma se především specializuje na MCU s jádry ARM Cortex, které jsou 32 a 64 bitové.

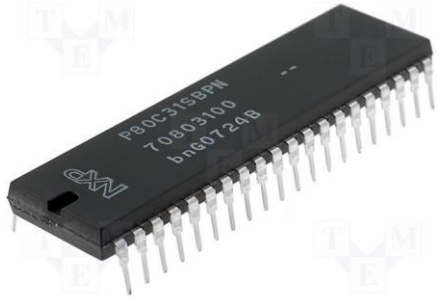
Stránky výrobce: http://www.nxp.com/

Řady 8 – bitových MCU:

- FLASH 80C51 – především v automobilovém průmyslu
- LPC900 – vysoce výkonné, v extrémně malých pouzdrech
- LPC700 – obdobné jako LPC900
- OTP/ROM 80C51– je určena pro aplikace prováděné v reálném čase tzv. Real – Time

Vlastnosti MCU P80C31SBPN:

MCU s jádrem 80C51. Velikost paměti SRAM je 128B. Maximální provozní kmitočet je 16 MHz. V pouzdru má 40 pinů, z nichž má 32 vstupně výstupních. Obsahuje tři 16 - bitové čítače.



Obr. 6 – MCU NXP P80C31SBPN[6]

1.6 Fujitsu Semiconductor

Tato firma se zaměřuje na výrobu MCU pro domácí spotřebiče, elektrická nářadí. Zaměřuje se také na zdravotnictví a automobilový průmysl. Tyto procesory jsou RISC s Harvardskou architekturou. V 32 a 64 bitových verzích MCU jsou použity jádra ARM Cortex.

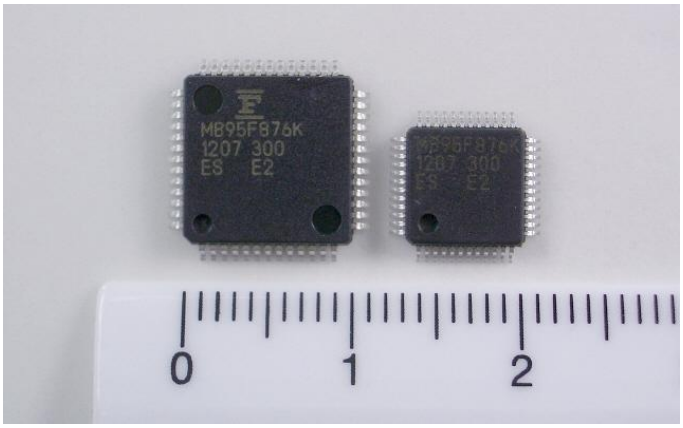
Stránky výrobce: http://www.fujitsu.com/

Řady 8 - bitových MCU:

- MB953xx
- MB954xx
- MB955xx
- MB956xx
- MB957xx
- MB958xx

Vlastnosti MCU MB95F876KPMC z řady MB95870:

Většina MCU této firmy poskytují velmi přesné interní oscilátory a mnoho různých periférií. Nabízí funkci PWM (pulsně šířková modulace). Umožňují zabezpečení paměti Flash, před neoprávněným přístupem. Tento MCU obsahuje i řadič dotykového senzoru (TSC). Velikost paměti Flash má 36 kB, RAM 1 kB, osm 10 – bitových A/D převodníků, jeden ADC komparátor, Watchdog. Maximální provozní frekvenci má 16 MHz. Počet pinů v pouzdře má 48. Z nichž se dá použít až 45 jako I/O.



Obr. 7 – MCU Fujitsu MB95F876KPMC[7]

2 ATmega32

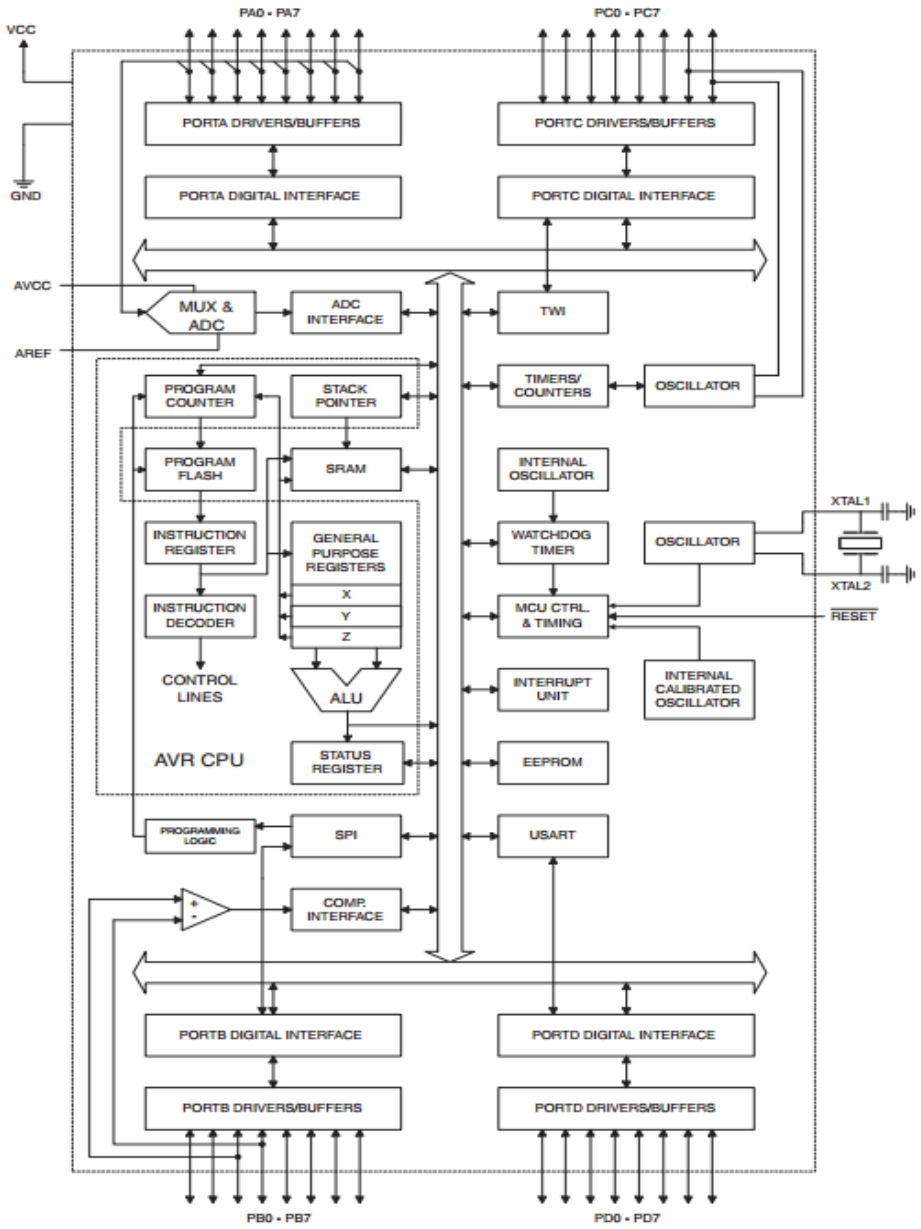
2.1 Obecný popis MCU Atmega32

Mikroprocesor ATmega32 je výkonný 8 – bitový procesor RISC s harvardskou architekturou. Má tedy oddělenou paměť programu a paměť dat. ATmega32 má 32kB programové paměti (Flash), 2kB datové paměti (SRAM) a 1kB EEPROM (až 100000 zápis/ smazání cyklů). Dále má dva 8 – bitové časovače a jeden 16 – bitový časovač. Tyto časovače mají oddělené děličky frekvence. Dále obsahuje 4 PWM kanály, 10 – bitový AD převodník přepínatelný na 8 vstupů. Obsahuje také jednotky SPI, TWI, USART a programovatelný Watchdog. Pro možnost ladění programu přímo v aplikaci má vestavěné standardní rozhraní JTAG (IEEE 1149.1), které zpřístupňuje registry mikroprocesoru. Samotný procesor pak informuje v průběhu práce o své činnosti PC. Obsahuje také interní kalibrovaný RC oscilátor, ale vhodnější je použít externí krystal, který je přesnější. Je to lepší např. pro sériovou komunikaci, ale i pro časovače. V pouzdru DIL má 40 vývodů v pouzdru pro SMD TQFP má 44 vývodů. Z nich nabízí 4 I/O brány po 8mi bitech. Napájecí napětí je potřeba zabezpečit, aby bylo v rozsahu od 4,5V do 5,5V. Samozřejmě existují i nízkonapěťové verze, které lze napájet menším napětím. Má šest možností úsporného režimu. Umožňuje výkon až 16 MIPS při 16 MHz. MCU ATmega32 je podporován s kompletní sadou programů a systémových nástrojů.

Hlavní vlastnosti RISC procesorů:

- Redukovaná sada instrukcí
- Obsahuje převážně jednoduché instrukce
- Délka provádění jedné instrukce je jeden cyklus
- Délka (počet bitů) všech instrukcí je stejná
- Využívá se zde zřetězení instrukcí

V další části rozepíši podrobněji části mikroprocesoru.



Obr. 8 – blokové schéma MCU ATmega32[1]

2.2 Architektura

AVR jsou RISC procesory s Harvardskou architekturou. RISC (Reduced Instruction Set Computer) znamená, že mají zredukovanou instrukční sadu. Kompletní sada pro AVR má 135 instrukcí, tolik instrukcí není u všech modelů. Nižší modely mají 118 instrukcí a nejnižší modely jen 89 instrukcí. Díky nízkému počtu instrukcí daných použitím architektury RISC je možné většinu instrukcí vykonávat v jednom hodinovém cyklu. Z tohoto vyplývá, že při taktovací frekvenci 20 MHz je možné dosáhnou až 20

MIPS (20 miliónů instrukcí za sekundu. Jádro AVR se skládá z 32 x 8 - bitových univerzálních registrů (slouží pro ukládání adres nebo dat). Tyto registry jsou přímo propojené s ALU a doba přístupu k těmto registrům trvá jen jeden strojový cyklus.

Mikroprocesory AVR se stávají čím dál víc oblíbenější zejména pro svůj výpočetní výkon, relativně lehké programování a nízkou cenu. Většina obsahuje 3 druhy pamětí: Flash – programová paměť, EEPROM – paměť dat (slouží na ukládání např. naměřených hodnot, uchování programu…), SRAM – rychlá datová paměť, slouží na rychlé operace s proměnnými. Výrobce zaručuje 10000 R/W cyklů pro paměť Flash a 100000 R/W cyklů pro paměť EEPROM. Velká výhoda je rozhraní JTAG, které umožňuje kontrolu chování MCU a náhled do registrů. Jak již bylo zmíněno, MCU AVR vynikají širokou výbavou periferií např. 8/16 bitové čítače/časovače, PWM výstupy, A/D, D/A převodníky, TWI (I2C), WDT, AD komparátory, interní kalibrovaný oscilátor, USART, RE232, podpora LCD, Ethernet a mnoho dalšího.

Využití mikroprocesorů AVR:

MCU AVR můžeme najít v různých bezpečnostních systémech, v automobilech, herních konzolích, v síťových aplikacích.

ATmega32:

K řešení mého úkolu jsem použil mikroprocesor ATmega32 v pouzdru DIL40. Tento mikroprocesor jsem zvolil z toho důvodu, že má množství nejrůznějších periférií a poměrně mnoho I/O pinů (4 brány po 8 pinech).

Na předešlém blokovém schématu je přehledně znázorněno zapojení jednotlivých bloků v mikroprocesoru a přesně viditelná architektura RISC.

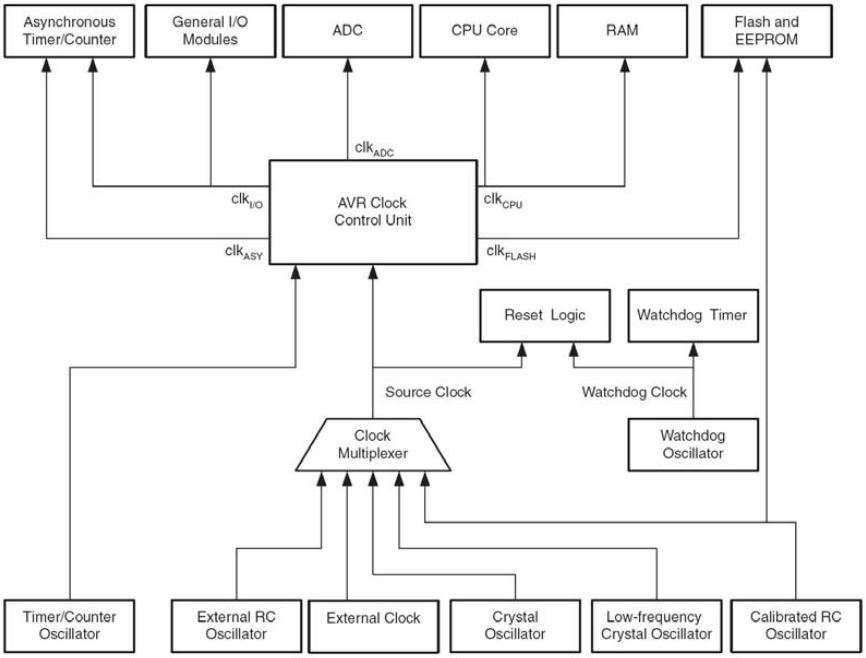
Obsahuje:

- 32kB Flash programové paměti
- 1 kB datové paměti EEPROM
- 2 kB rychlé datové paměti SRAM
- 131 výkonných instrukcí
- Dva 8 - bitové čítače/časovače
- Jeden 16 - bitový čítač/časovač
- 4 PWM kanály
- 10 – bitový A/D převodník

- USART (RS232)
- Kalibrovaný RC oscilátor

Zdroje taktovacího kmitočtu

- Externí krystal
- Externí nízkofrekvenční krystal
- Externí RC oscilátor
- Interní kalibrovaný RC oscilátor
- Externí zdroj taktovacího kmitočtu

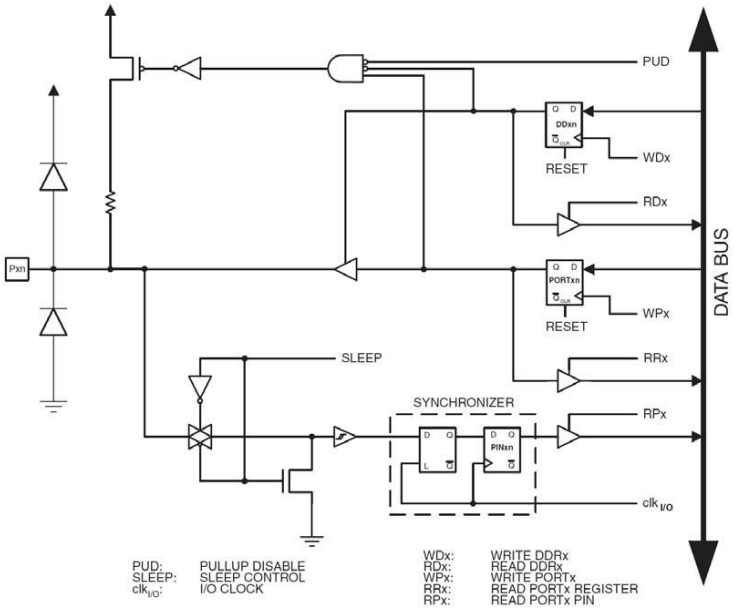


Obr. 9 - rozvedení taktovacích signálů v mikroprocesoru[1]

- Výběr zdroje taktovacího kmitočtu se provádí pomocí tzv. Flash Fuse Bits. Standartně je nový mikroprocesor dodáván s nastaveným interním RC oscilátorem
- 1MHz.

2.3 Vstupně/výstupní porty

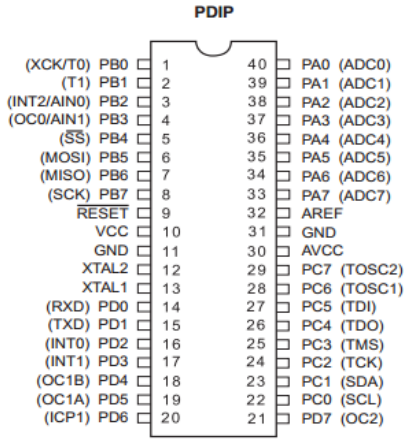
- čtyři 8 - bitové obousměrné brány označené PORT A až PORT D
- Jednotlivé piny portů mohou sloužit jako obecně vstupně / výstupní, anebo mohou být spojeny s příslušnou periferií.



Obr. 10 – interní zapojení jednoho pinu[1]

Vlastnosti:

- Každý pin je na vstupu chráněn diodami proti zemi a proti Ucc
- Každý pin má interní pull-up rezistor, který může a nemusí být aktivní
- Každý pin je zatížitelný proudem až 20 mA při logické 1 i v logické 0
- Každý port (brána má tři ovládací registry – směrový registr DDRx, datový registr PORTx a registr nesoucí informaci o skutečné logické úrovni pinů portu PINx



Obr. 11 – popis vývodů ATmega32[1]

Popis jednotlivých pinů:

VCC	pro připojení napájecího napětí
GND	uzemnění
Port A (PA7..PA0)	Piny portu A slouží jako analogové vstupy pro A/D převodník. Dále slouží také jako 8mi-bitový obousměrný I/O port, pokud není použit A/D převodník. Dále můžeme pro každý pin portu nastavit pull-up rezistory.
Port B (PB7..PB0)	Port B je 8mi-bitový obousměrný I/O port s vnitřními pull-up rezistory.
Port C (PB7..PB0)	Port C je 8mi-bitový obousměrný I/O port s vnitřními pull-up rezistory. Pokud je rozhraní JTAG povoleno, tak pull-ip rezistory na pinech PC5(TDI), PC3 (TMS), PC2(TCK) jsou aktivní, a to i když nastane reset.
Port D (PD7..PD0)	Port D je 8mi-bitový obousměrný I/O port s vnitřními pull-up rezistory.
$\overline{\text{RESET}}$	Obnovení vstupu do výchozího stavu. Přivedením nízké úrovně na tento vstup déle než je minimální délka pulsu se vygeneruje signál reset, i když není přiváděn hodinový signál.
XTAL1	vstup do invertujícího zesilovače oscilátoru a vstup do operačního obvodu vnitřních hodin
XTAL2	výstup z invertujícího oscilátoru
AVCC	AVCC je napájecí napětí pro PORT A jako A/D převodník. Mělo by být připojeno externě na VCC, když ho ADC nepoužívá. Pokud je ADC požit, mělo by být na VCC připojeno prostřednictvím low-pass filtru (propustí jen nízkofrekvenční signály a zeslabuje signály s frekvencemi vyššími než je mezní frekvence.
AREF	AREF je analogový referenční pin pro A/D převodník.

Nastavení portu

Ke každému portu jsou přiřazeny tři registry.

- Registr PINx (Pins Input) je určen pouze pro čtení a jeho obsah odpovídá hodnotě vstupních pinů
- V registru DDRx (Data Direction Registr)nastavujeme, jestli bude příslušný pin vstupní nebo výstupní
- V registru PORTx (datový registr portu) nastavujeme jestli příslušný pin bude v log. 1 nebo v log. 0

Nastavení pinů jako vstup nebo výstup:

Pin je vstupní, jestliže v určitém bitu registru DDRx zapíšeme hodnotu 0

Pin je výstupní, jestliže v určitém bitu registru DDRx zapíšeme hodnotu 1

Po resetování jsou všechny registry DDRx vynulovány, tzn. jsou nastaveny jako vstupy

DDRx registr

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr. 12 – registr DDRA [1]

Pull-up rezistory:

Je - li určitý vývod portu nastaven jako vstupní, pak můžeme nastavit připojení nebo odpojení pull - up rezistorů. Toto nastavení se provádí zapsáním hodnoty do vyrovnávacího registru PORTx.

PORTx registr

Port A Data Register – PORTA

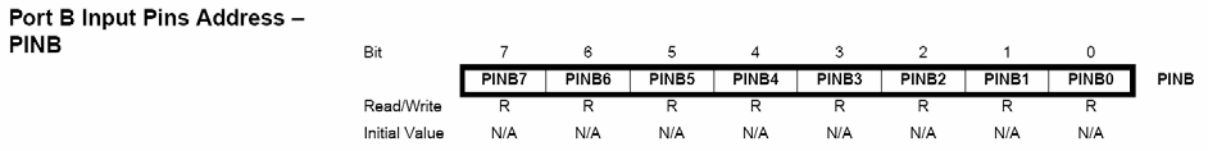
Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr. 13 – datový registr portu A [1]

- Zapíšeme-li 0 v určitém bitu do registru PORTx, tak pull-up rezistor bude na odpovídajícím vstupu odpojen
- Zapíšeme-li 1 v určitém bitu do registru PORTx, tak pull-up rezistor bude na odpovídajícím vstupu připojen

Po resetování se všechny registry PORTx vynulují.

PINx registr



Obr. 14 – registr PINA [1]

- Určen pouze pro čtení
- Obsahuje informaci o fyzickém stavu pinů daného portu

Alternativní funkce pinů/portů

- Aktivace příslušné funkce se provede v řídícím registru dané periférie, které určitý pin náleží (např. PORTA využívá AD převodník, externí přerušení INT0, INT1 na PORTD atd.)
- Při aktivaci těchto funkcí se musí nastavit směr u většiny pinů (určit, jestli je pin vstupní či výstupní v registru DDRx)

Výpis zvláštních funkcí na portech:

Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

obr. 15 – výpis alternativních funkcí portů[1]

Přerušení u MCU ATmega32

ATmega32 disponuje několika zdroji přerušení. V následující tabulce jsou rozděleny podle priority. Každé přerušení má svůj vektor přerušení.

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Externí přerušení

Jedním z nejčastějších přerušení jsou právě externí přerušení. Používají se zejména, potřebujeme – li vykonat nějakou část programu pomocí jiného přerušení než je od zabudovaných periférií v mikroprocesoru. Např. můžeme toto přerušení využít pro tlačítko.

ATmega32 obsahuje celkem tři zdroje externího přerušení.

- INT0 – pin PD2
- INT1 – pin PD3
- INT2 – pin PB2

Mikroprocesor může zareagovat na 4 stavy na I/O pinu

- Na log. 0
- Libovolnou změnu úrovně
- Sestupnou hranu signálu
- Náběžnou hranu signálu

Zapsáním log. 1 do registru GICR zvolíme zdroj externího přerušení

General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr. 16 – registr GICR [1]

V registru MCUCR (pro INT0 a INT1) nastavíme reakci na změnu stavu

MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr. 17 – registr MCUCR[1]

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Table 34. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

2.4 Čítač / časovač

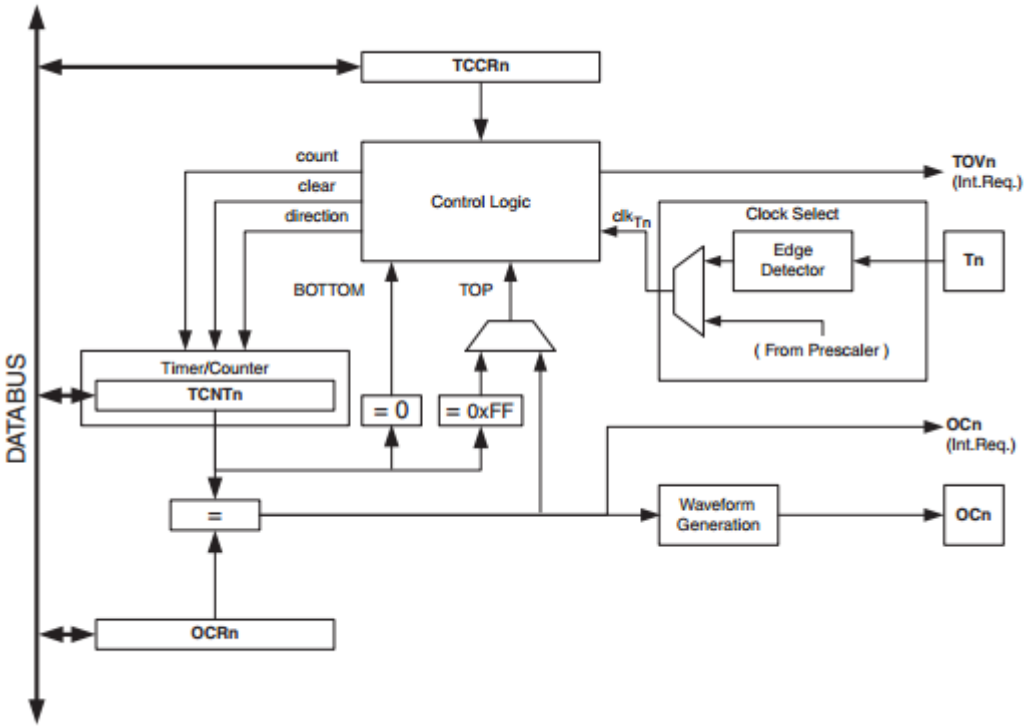
ATmega32 obsahuje celkem tři a to čítač / časovač 0, čítač / časovač 1 a čítač / časovač 2. Každý z nich dokáže vykonávat základní činnosti (čítat, časovat), ale mají i další funkce.

Pro ukázku uvedu čítač / časovač 0. Podrobnější zdroj informací je k nalezení v příslušném datasheetu.

Čítač / časovač 0

Univerzální 8 – bitový modul čítače / časovače 0

V pouzdře má vyvedené dva piny. Jeden je vstupní T0 (PB0), slouží na přivádění vnějších impulsů. Druhý pin je výstupní OC0 (PB3), tento pin se používá hlavně na generování PWM signálu.



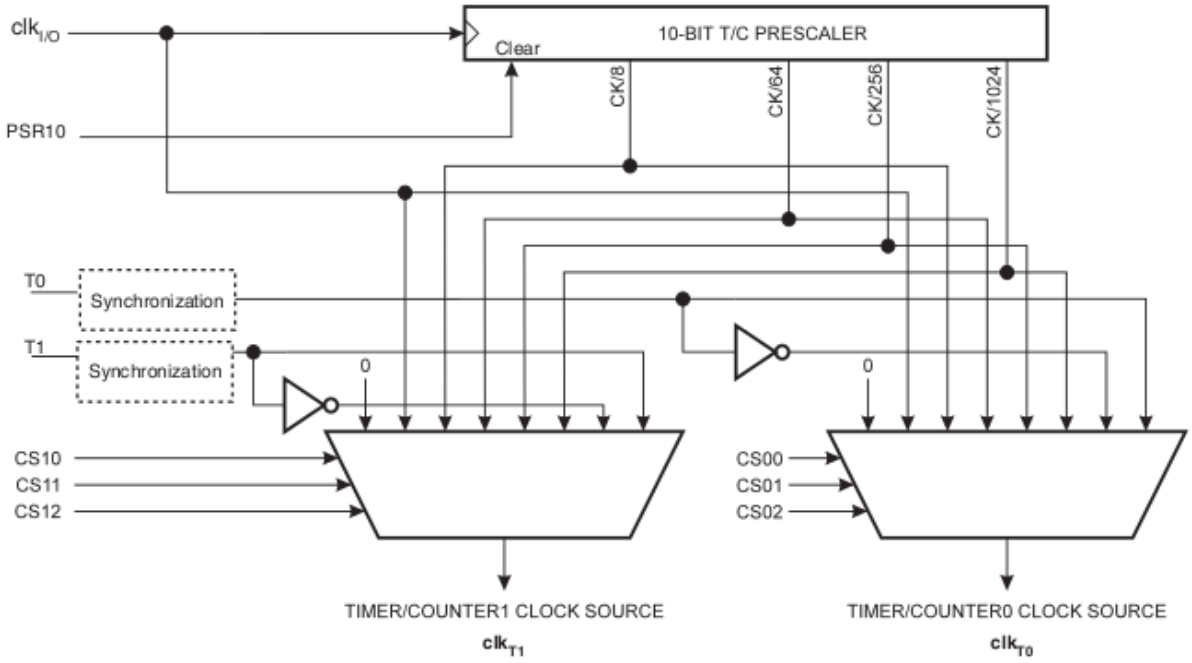
Obr. 18 – blokové schéma čítače/časovače0 [1]

Registry, které slouží na ovládání, jsou:

- TCNT0 (TimerCounterRegistr) – 8 – bitový registr, který slouží na čítání impulzů, obsahuje aktuální stav čítače
- OCR0 (OutoutCompareRegistr) – 8 – bitový registr, který obsahuje hodnotu, se kterou se má porovnávat obsah registru TCNT0
- TIFR (TimerInterruptFlagRegistr) – 8 – bitový registr, který slouží na sledování žádostí o přerušení od čítačů
- TIMSK (TimerInterruptMaskRegistr) – 8 – bitový registr, který slouží na nastavení přerušení od čítačů
- TCCR0 (TimerCounterControlRegistr) – 8 – slouží na řízení čítače / časovače

Dělička čítače / časovače 0:

Čítač / časovač 0 může být řízený vnitřními hodinami přímo anebo přidělenou děličkou, která se nastavuje v registru TCCR0.



obr. 19 – schéma předděličky v MCU ATmega32[1]

Výběr děličky se provádí nastavením bitů v registru TCCR0. Jsou poskytovány děličky 8, 64, 256 a 1024. Dělička se nechá samozřejmě i vypnout. V následující tabulce je vidět nastavení bitů pro jednotlivé děličky.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

obr. 20 – přehled možných předděliček[1]

3 Vývoj programu v prostředí AVR Studio 4

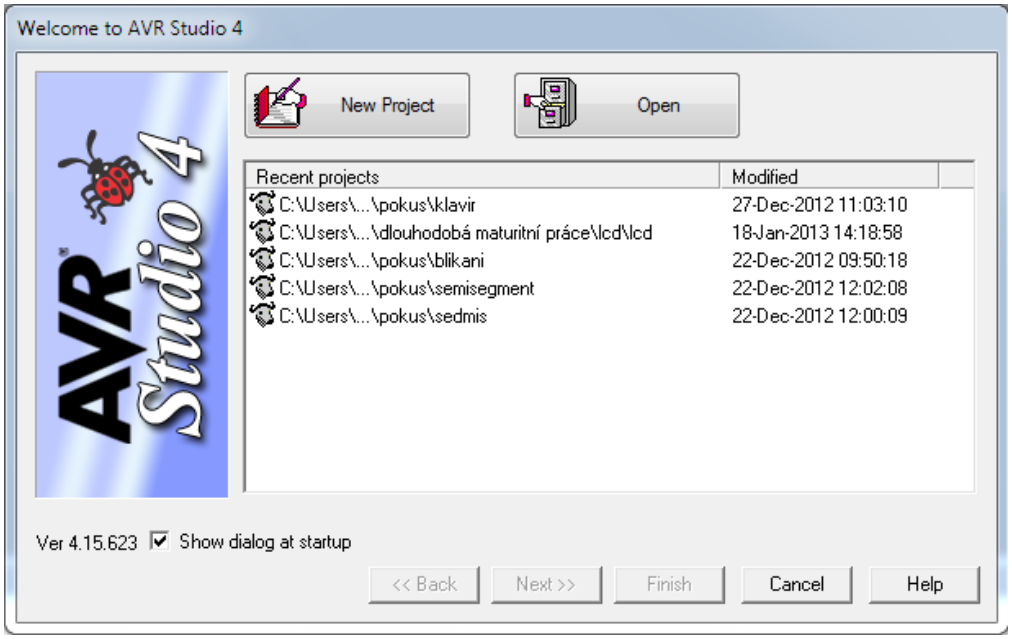
Pro vývoj programů je asi snadnější programovat v prostředí CodevisionAVR, ale mně se lépe pracuje v prostředí AVR Studio. Jedná se o komerčně dostupný, volně šiřitelný nástroj obsahující textový editor se zvýrazněním syntaxe, debugovací nástroj a jednoduchý simulátor.

3.1 Instalace programu

Ze stránek výrobce, tedy společnosti Atmel, si stáhneme program AVR Studio. Na stránkách je mnoho verzí. Já jsem si vybral AVR Studio 4, které lze stáhnou ze stránek AvrStudio4Setup.exe. Ještě před tím než nainstalujete AVR Studio, je vhodné nainstalovat WinAVR (překladač jazyka C, ke stažení zde WinAVR). Při instalaci a při prvním spuštění programu je třeba mít pro operační systémy Windows® 2000, XP, Vista, Windows 7 nebo Windows 8 přístupová práva admistrátora. Program se totiž inicializuje až při prvním spuštění. Po tomto úkonu už můžete pouštět AVR Studio i z limitovaných účtů.

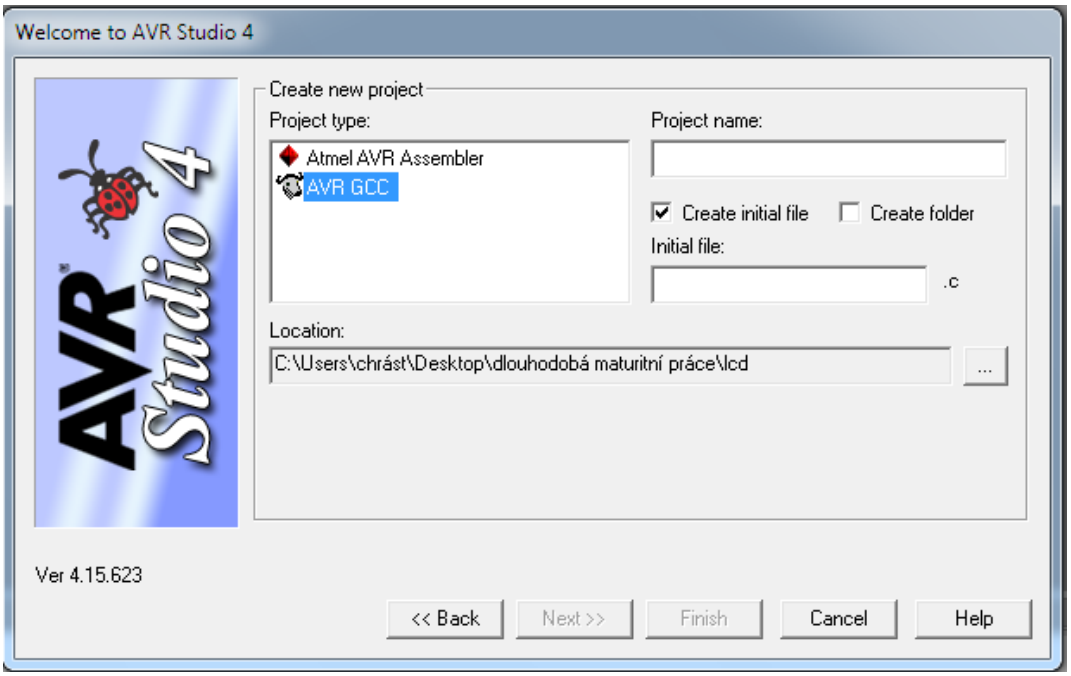
3.2 Vytvoření projektu

Pro vytvoření prvního programu zvolíme ***New Project->Next*** viz obr. 21



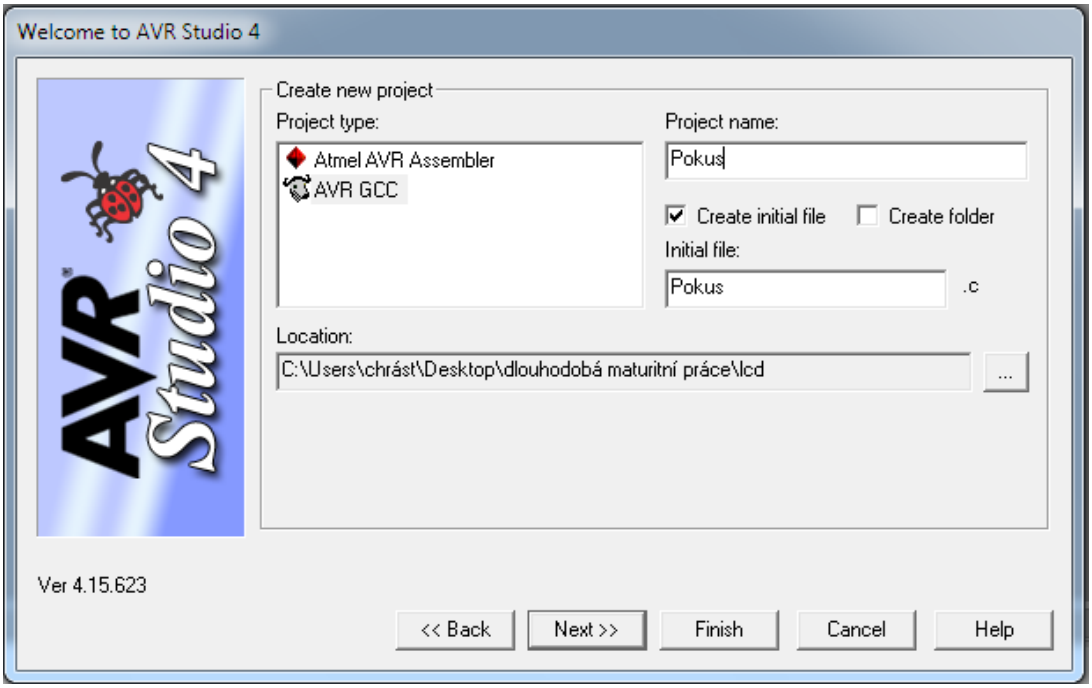
Obr. 21 – vytvoření projektu AVR Studio 4

Dále vybereme, jestli chceme programovat v C (***AVR GCC***) nebo Assembleru (***Atmel AVR Assembler***). A pokračujeme pomocí ***next***



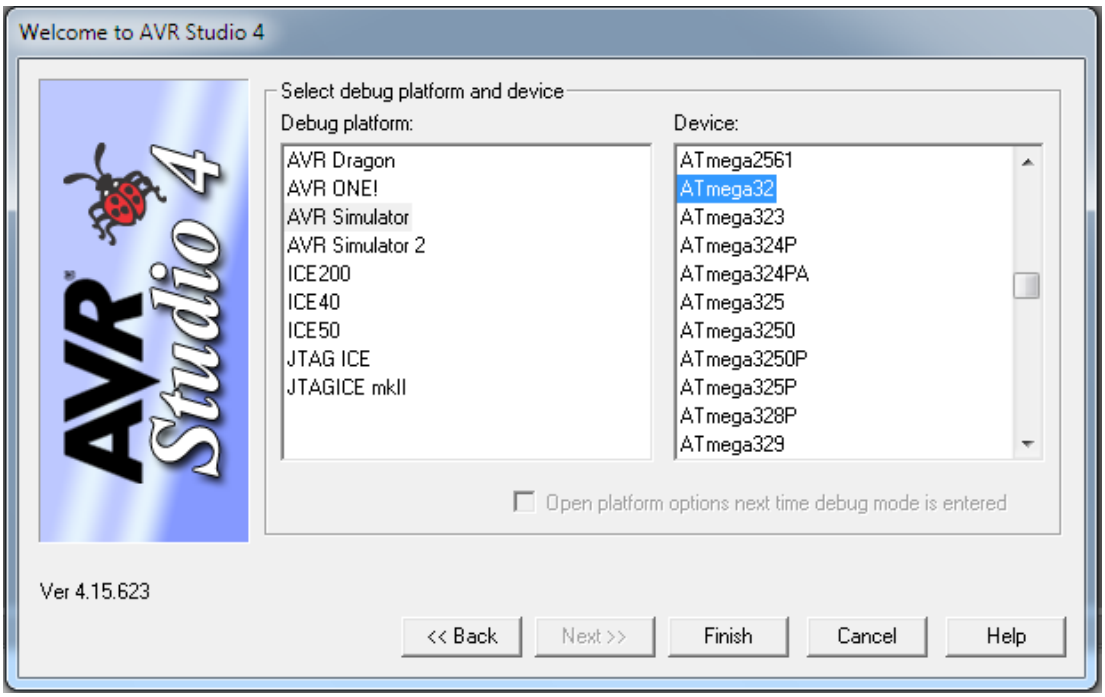
Obr. 22 – zvolení programovacího jazyka

Do položky ***Project name*** napíšeme svůj název projektu, zvolíme místo ukládání v položce ***Location*** a zvolíme ***next***



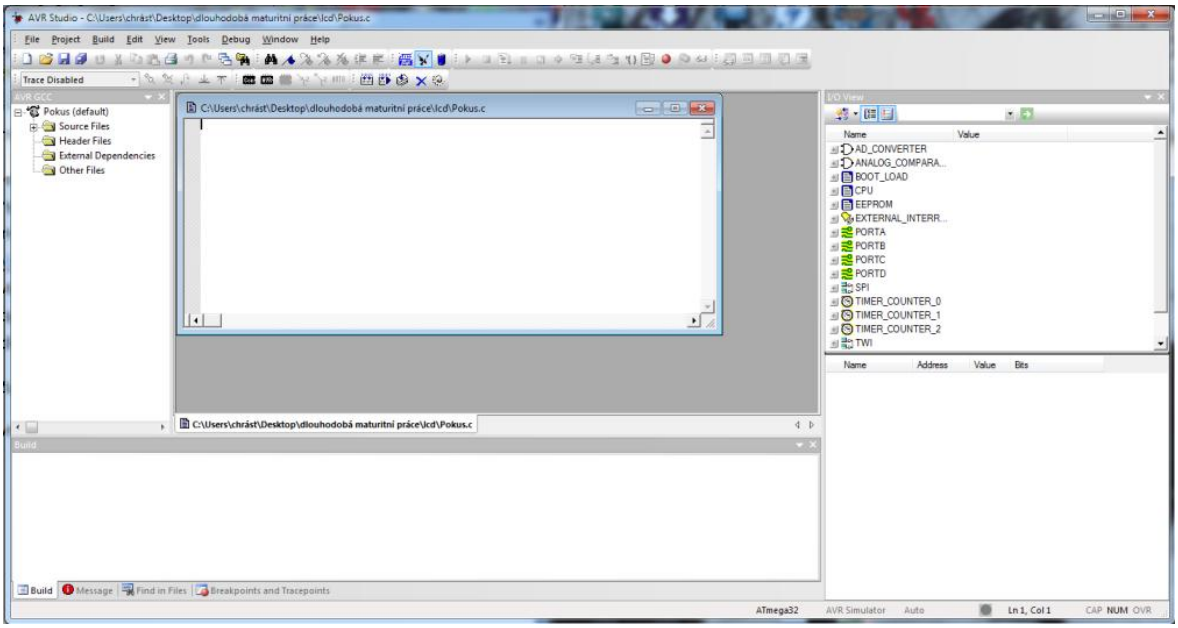
Obr. 23 – název a místo ukládání

Dále v sekci ***Debug platform*** vybereme ***AVR Simulator*** a v sekci ***Device*** vybereme příslušný mikroprocesor. Já jsem zvolil ***ATmega32***. A klikneme na ***Finish***.



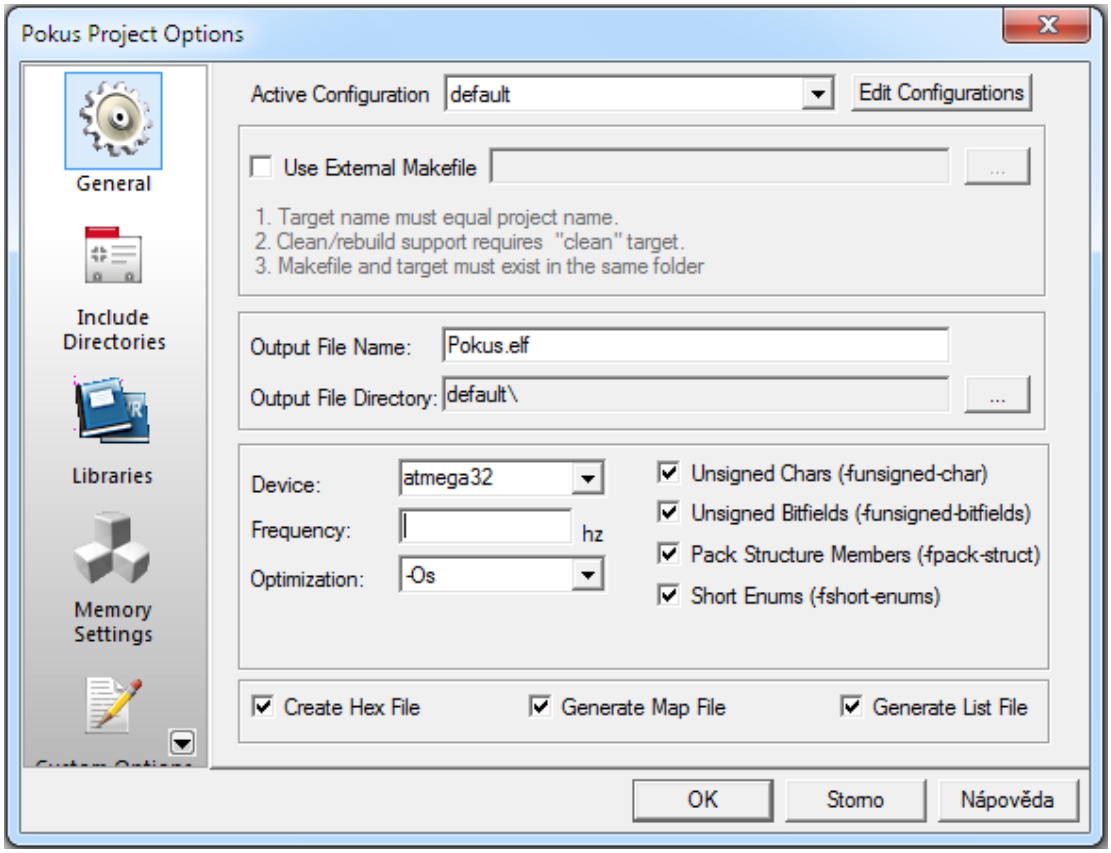
Obr. 24 – výběr procesoru

Po tomto nastavení by se nám měla objevit tato obrazovka. Do okna už můžeme rovnou psát náš program.



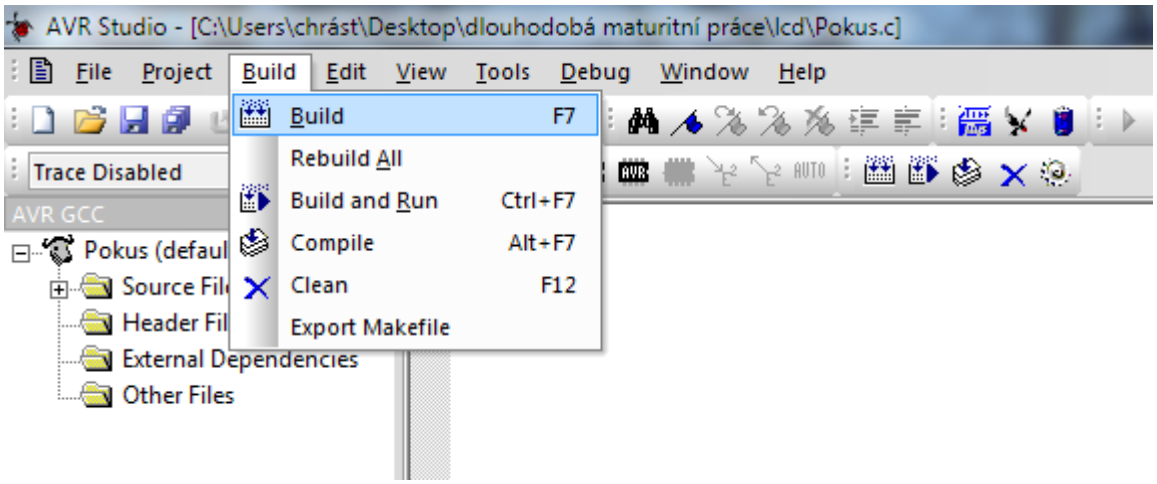
Obr. 25 – úvodní okno nově založeného projektu

V položce Project zvolíme ***Configuration Option*** a v záložce ***General*** zadáme frekvenci v Hz a změníme stupeň optimalizace na standartní-> ***Optimalization -Os***



Obr. 26 – nastavení optimalizace a frekvence procesoru

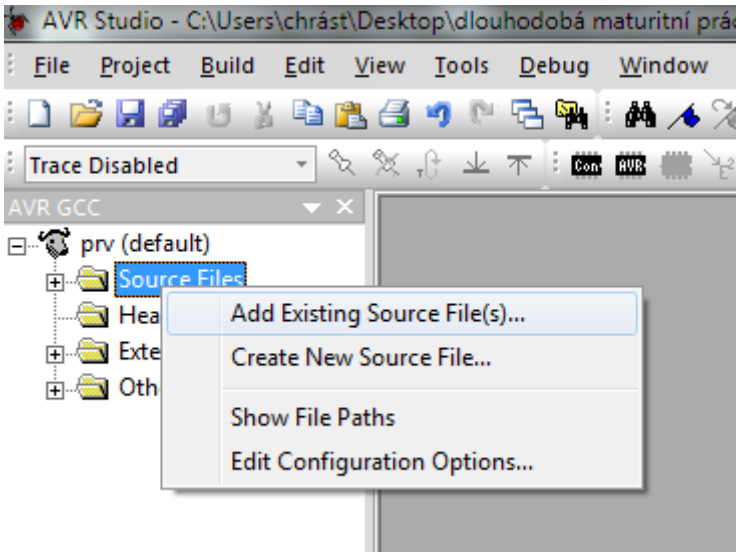
Překlad provedeme v záložce ***Build*** zvolením možnosti ***Build***, také lze provést klávesou F7.



obr. 27 – zkompilování programu (vytvoření souboru *.hex)

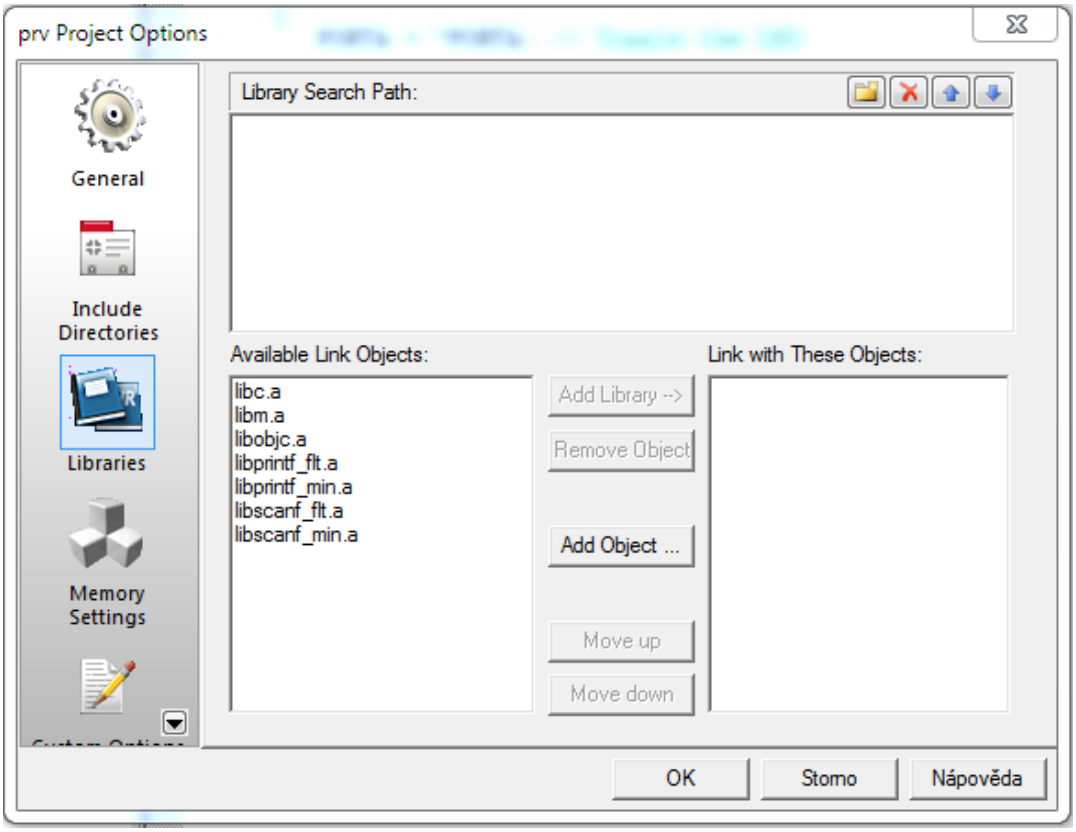
Pro práci s displejem LCD používám knihovnu, která obsahuje soubory **lcd.h** a **lcd.c**, v nichž jsou základní funkce na ovládání LCD displeje. Pro přidání knihovny do projektu musí udělat následovné (viz obr. 28).

Po vytvoření projektu a umístění souborů **lcd.h** a **lcd.c** do adresáře, kde se nachází projekt, můžeme přidat knihovnu. Pravým tlačítkem myši klikneme na položku ***Source Files*** a zvolíme možnost ***Add Existing Source File(s)*** a vybereme ***lcd.c***. To samé uděláme v položce ***Headers Files***, kde přidáme soubor ***lcd.h***.



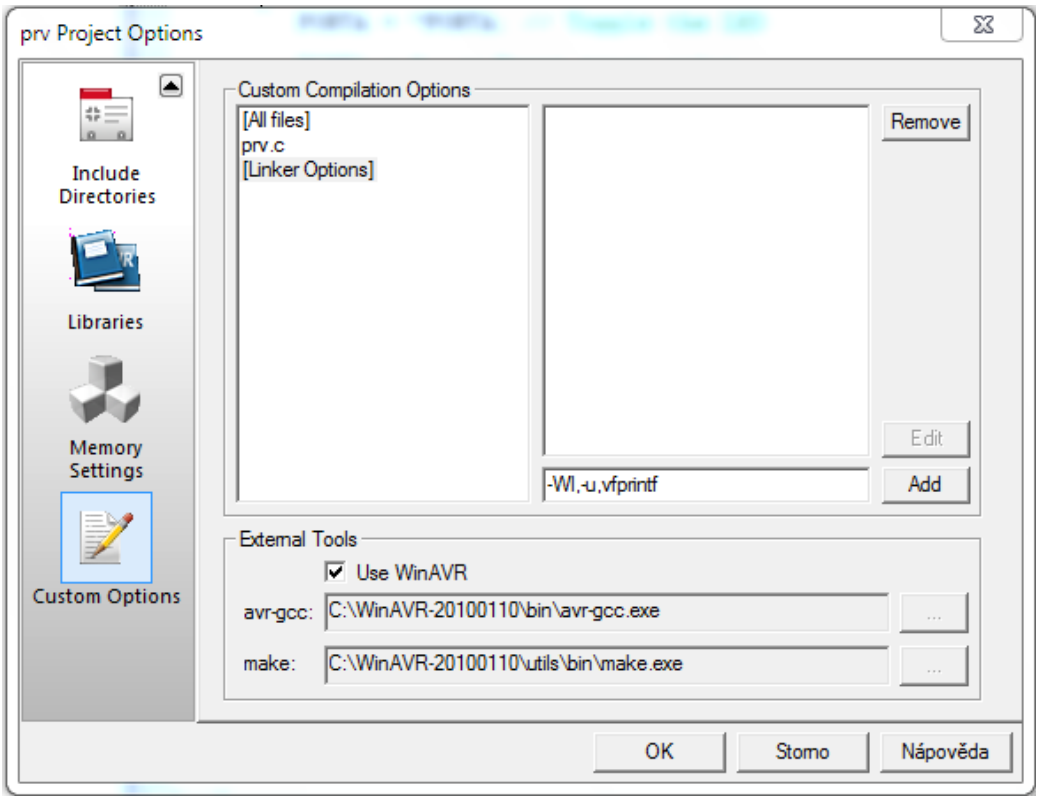
obr. 28 – přidání souboru lcd.c

Problém při vypisování desetinného čísla na LCD displeji. Pro vypsání desetinného čísla na LCD displeji, musíme upravit parametry projektu (viz obr. 29 a 30). Vybereme položku ***Project*** v ní ***Configuration Options*** a zvolíme položku ***Libraries***, kde přidáme soubory ***libprintf_flt.a*** a ***libm.a***.



Obr. 29 – přidání souborů pro práci s desetinnými čísly

Poté v položce ***Custom Option*** zvolíme ***Linker Option*** a přidáme **-Wl,-u,vfprintf**



Obr. 30 – nastavení pro výpis desetinných čísel

4 Postup práce při vývoji aplikace s MCU

4.1 Úvod

Při vývoji mikroprocesorové aplikace, jsou potřebné tyto věci:

- Vývojové prostředí (např. AVR Studio, CodeVision AVR)
- Znalost programovacího jazyka (např. C, Assembler, Basic)
- Programátor, přes který nahrajeme program do MCU
- Programovaný mikroprocesor s různými periferiemi

Mikroprocesor by šlo programovat Assemblerem, který se poměrně do hloubky probírá na škole. Ale já raději používám vyšší programovací jazyk, a to jazyk C. Mimochodem základy tohoto jazyka se na naší škole také vyučují. V jazyku C je programování mnohem přehlednější a dle mého názoru i jednodušší. Takže pro začátečníka je jazyk C rozumnou volbou. Díky tomu se dostane začátečník rychleji do tajů mikroprocesorů.

4.2 Konkrétní vývoj mikroprocesorové aplikace za pomoci UNI desky

Před vývojem mikroprocesorové aplikace musíme nejdříve vymyslet, co chceme udělat. Když už máme nějakou představu, tak si nakreslíme vývojový diagram. Když se nechceme pouštět do tvorby vývojových diagramů, tak si prostě napíšeme jednotlivé kroky programu. Při jednodušších aplikacích není tvorba těchto postupů důležitá, ale při složitějších nám může usnadnit mnoho práce (např. zorientovat se v nějakém programu, objasnit funkci programu atd.).

Když máme hotový postup nebo alespoň představu, tak si na vývojové desce připojíme k mikroprocesoru potřebné periférie (např. LCD display, 7 - segmentový LED display, LED diody, mikrotlačítka).

Poté můžeme začít psát samotný program. Zvolíme si vývojové prostředí, které nám vyhovuje (já používám AVR Studio 4). Dále se musíme rozhodnou, v jakém jazyce budeme chtít tvořit. Já používám jazyk C, protože jak už bylo jednou zmíněno, je jednodušší a přehlednější. Po těchto krocích nám nic nebrání tomu, abychom mohli začít psát vlastní program.

4.3 Programování MCU Atmega32 v jazyce C (základní příkazy)

Nyní zde popíši potřebné minimum pro programování MCU ATmega32 jazyce C

Hlavičkové soubory

#include<avr/io.h>

Tento hlavičkový soubor nám umožňuje práci s registry mikrokontroléru a přístup k nim pomocí jejich názvu.

#include<avr/interrupt.h>

Tento soubor vkládá funkce a makra na obsluhu přerušení

#include<util/delay.h>

Tento soubor nám umožňuje používat čekací funkce _delay_ms(); a _delay_us();

#include<stdio.h>

V tomto souboru se nachází rutiny pro práci se vstupy a výstupy

Registry

Práce s registry je nezbytně nutná. Bez této znalosti nejste schopni udělat nic. Pomocí registrů se nastavují porty, čítače, A/D převodník atd.

V této části popíši nastavení registrů pro porty.

Každý port má tři registry, a to DDRx, PORTx a PINx

Např. chceme-li nastavit celý port A jako výstupní a nastavit na výstup tohoto portu A bitovou kombinaci 0000 1111 (piny PA0 až PA3 jsou v logické 1 a zbylé jsou v logické 0).

Zápis by vypadal takto

DDRA = 0b11111111; //předponou 0b říkáme, že následující znaky jsou v binárním //tvaru, možno zapsat hexadecimálně 0xFF

PORTA = 0b00001111; //možno zapsat v hexadecimálním tvaru 0x0F

Za každým příkazem se musí udělat středník!

Maskování: je vhodné použít, chceme – li nastavit jen určitý pin, když už jsou na daném portu nějaké hodnoty. Při maskování se používají bitové log. operátory. Jsou to: bitový součin **&,** bitový součet **|,** bitová negace ~ a bitový posun <<. Tyto operátory se také používají pro nastavování registrů např. u časovačů, A/D převodníků.

Zápis těchto operátorů by vypadal takto:

Na portu A předpokládejme, že máme hodnotu např. 0b00000011

PORTA |= (1<<PA7); // logický součet, maska 0b10000000

Po této operaci by na portu A byla tato kombinace: 0b10000011

Když chceme zapsat více log. 1 tak zápis bude vypadat takto:

PORTA |= (1<<PA7) | (1<<PA6); //zapsání log.1 na piny PA7 a PA6

Podmíněné příkazy

Příkaz if

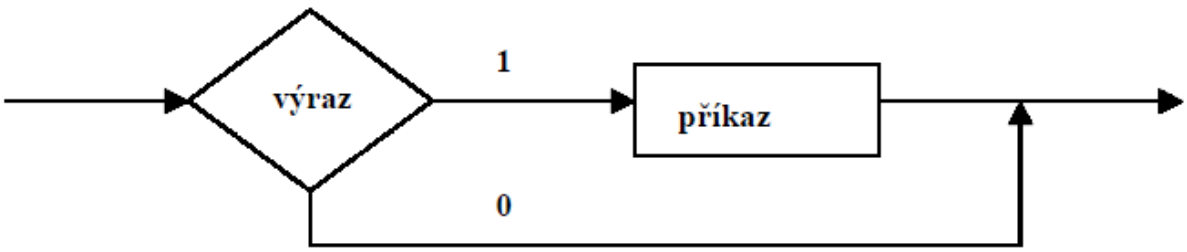
Zápis: ***if (podmínka)***

{příkazy;}

Je-li podmínka splněna, proveď příkaz ve složených závorkách a pokračuj dál. Není-li splněna, nevykonej příkaz ve složených závorkách a pokračuj dál.

Příkaz if else

Zápis: ***if (podmínka)***



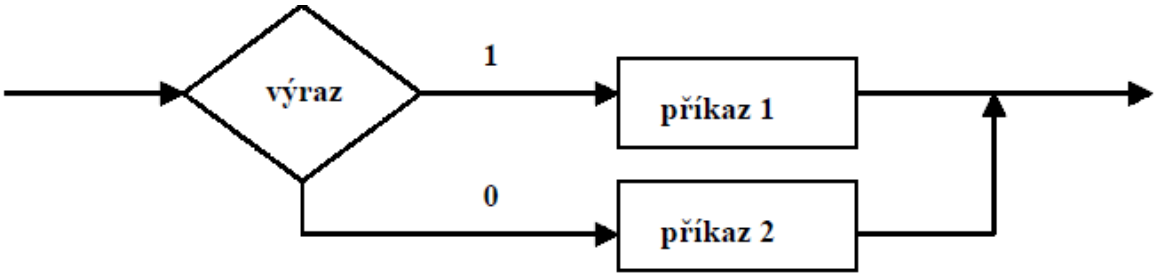
Příkaz **if**

{příkazy;}

else

{příkazy;}

Je-li splněna podmínka, vykonej příkaz u if a vynechej příkaz u else a pokračuj dál. Není-li splněna podmínka, přeskoč příkaz u if a vykonej příkaz u else, a potom pokračuj dál.



Příkaz **if -else**

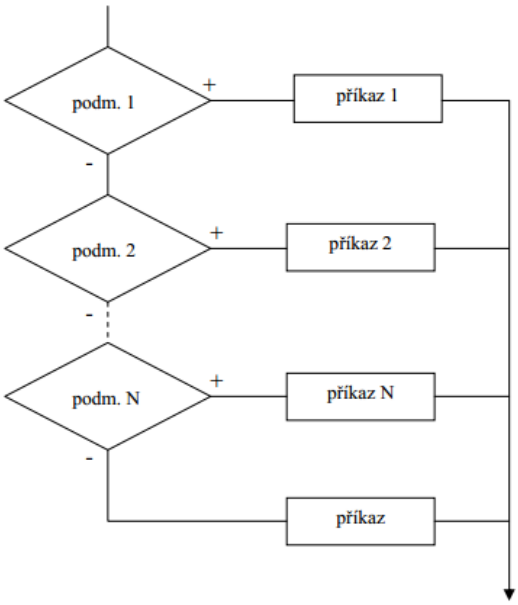
Příkaz if else if

Zápis: ***if(podmínka) {příkazy;}***

else if (podmínka) {příkazy;}

else if (podmínka) {příkazy;}

else {příkazy;}



Přepínač

Přepínač slouží k rozdělení posloupnosti příkazů na části, následné vybrání a provedení některé, či některých z nich.

Zápis: ***switch (celočíselný výraz)***

{

case(celočíselný konstantní výraz) : příkazy;

case(celočíselný konstantní výraz) : příkazy;

case(celočíselný konstantní výraz) : příkazy;

default: příkazy;

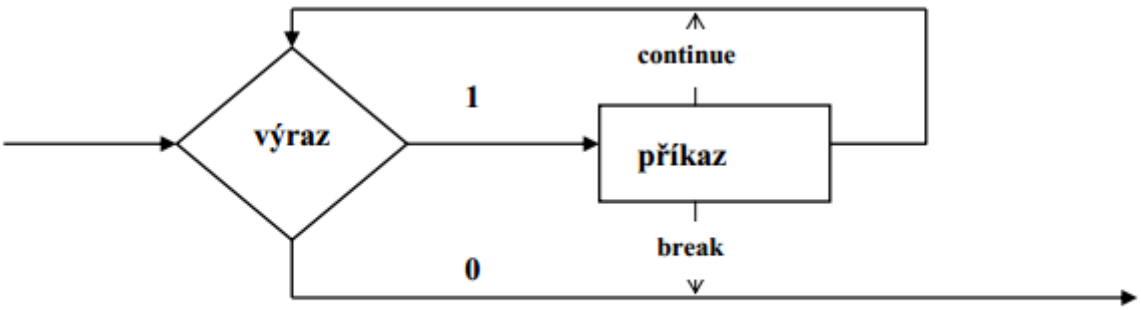
}

Cykly

Cyklus while

Zápis: ***while (podmínka)***

{příkazy;}



Příkaz **while**

Je- li podmínka splněna, provede se příkaz. Příkaz ***break*** má ve všech cyklech stejný význam. Ukončí provádění příkazů těla cyklu a předá řízení prvnímu příkazu za příkazem while. Tímto způsobem můžeme bezprostředně ukončit průběh cyklu bez ohledu na hodnotu podmínky. Příkaz ***continue*** rovněž ukončí provádění těla cyklu, ale vrací se zpět na začátek cyklu. Tyto příkazy jsou nepovinné. Zápis ***while(1)*** nám vytvoří nekonečný cyklus.

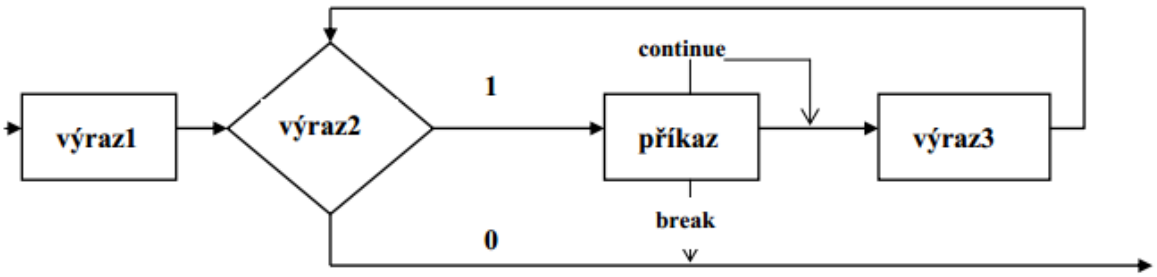
Cyklus for

Zápis:

for (výraz1; výraz2; výraz3)

{příkazy;}

Výraz1 je proveden před prvním vyhodnocením testu. Typicky se používá pro inicializaci proměnných před cyklem. Po každém provedení těla cyklu se provede ***výraz3*** (např. inkrementace proměnné). ***Výraz2*** je testovací výraz. Pokud ho neuvedeme, použije překladač hodnotu 1, a tedy bude provádět nekonečný cyklus.



Příkaz **for**

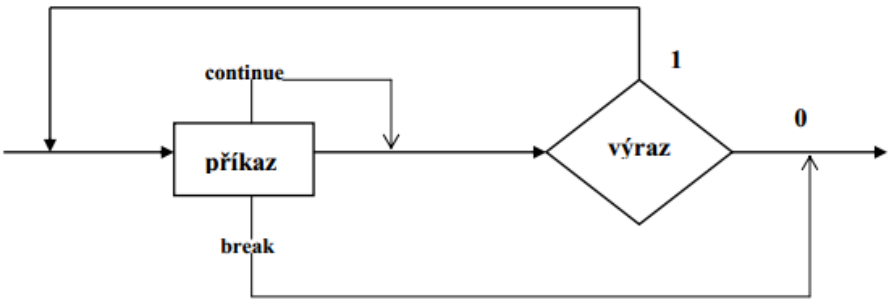
Cyklus do while

Tento cyklus je jediným, který zajišťuje alespoň jedno vykonání těla cyklu. Je to cyklus s podmínkou na konci.

Zápis:

do{příkazy;}

while (podmínka)

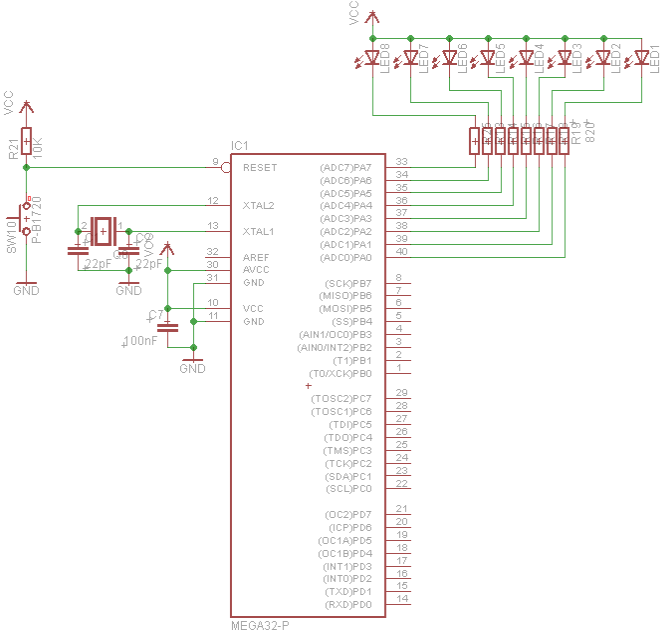


Příkaz **do**

5 Úlohy s mikroprocesorem ATmega 32

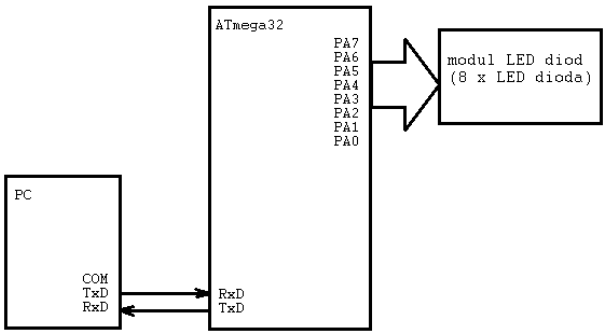
5.1 Úlohy s LED diodami

Schéma zapojení:



Obr. 31 – schéma zapojení LED diod

Blokové schéma:



Obr. 32 – blokové schéma zapojení

5.1.1 Blikač

V prvním programu ukážu nejjednodušší příklad, a to posílání dat na bránu. Aby to nebylo tak jednoduché, tak si uděláme jednoduchý blikač. Data budu vysílána na PORT A, na kterém jsou LED diody připojeny. V mém zapojení (diody se společnou anodou) budou LED diody svítit, pokud bude na příslušném pinu portu A úroveň log. 0. A bude zhasnutá, pokud bude signál na úrovni log. 1. Hodnota jednotlivých bitů v poslaném bytu na bránu nám pak určí, které LED diody budou svítit a které nikoli. Pro blikání diod použijeme čekací funkci *_delay_ms*. Neustálé opakování čekání a posílání bytů zabezpečíme

jednoduše tak, že příkazy na vyslání bytů a čekání umístíme do nekonečného cyklu (např. while(1) nebo for(;;)).

```
#define F_CPU 16000000UL
#include <avr/io.h> //knihovna vstupů a výstupů (PORT, DDR)
#include <util/delay.h> //knihovna čekacích funkcí (např. _delay_ms())
#define CYKL1 0b00001111 //LED připojené na piny A4 až A7 budou svítit
                         //jako první
#define CYKL2 0b11110000 //LED připojené na piny A0 až A3 budou svítit
                         //jako druhé
#define ZPOZD 100 //zadefinování námi určené konstanty
int main (void)   //hlavní funkce
{
DDRA = 0b11111111; //port A je zadefinován jako výstupní, lze také zapsat
                   //hexadecimálně DDRA = 0xFF
 while(1)//Nekonečná smyčka, lze použít místo cyklu while(1)
         //cyklus for(;;)
 {
 PORTA = CYKL1; //námi zadefinovanou konstantu CYKL1 pošleme na piny
                //portu A, rozsvícení LED na pinu A4 až A7
 _delay_ms (ZPOZD);  //zpoždění podle naší konstanty ZPOZD =>100 v tomto
                     //případě 100ms
 PORTA = CYKL2; //námi zadefinovanou konstantu CYKL2 pošleme na port A
                // rozsvícení LED na pinu A0 až A3
 _delay_ms (ZPOZD);  //opět zpoždění podle naší konstanty, tj. 100 ms
 }                   //opět se vracíme na začátek cyklu while
}
```

Nevýhodou této formy napsání programu je v tom, že musíme při definování konstanty napsat druhou přesně opačně. Abychom to nemuseli takto složitě převádět, tak můžeme použít tzv. bitový komplement (bitová negace). Stačí nám tedy nadefinovat jen jednu konstantu a druhá se vytvoří sama pomocí negace. A kód by poté vypadal třeba takto:

```
while(1) // Nekonečná smyčka
 {
 PORTA = ~CYKL1; //zápis bitového komplementu na bránu A. LED diody,
                 //které svítili zhasnou a naopak
 _delay_ms (ZPOZD);//zpoždění podle naší konstanty
 } //opět se vracíme na začátek cyklu while
```

Blikání by šlo také realizovat pomocí čítače. Na jednoduchém příkladu si ukážeme, jak to zrealizovat. Využijeme k tomu čítač/ časovač 1 s předděličkou 64, kterou nastavíme pomocí bitů CS10 a CS11 v registru TCCR1B. K blikání využijeme registr čítače 1 TCNT1. Při dosažení určité hodnoty v tomto registru se změní stav brány A.

```
#define F_CPU 16000000UL
#include <avr/io.h>
int main (void)
{
DDRA = 0xff; // port A výstup
```

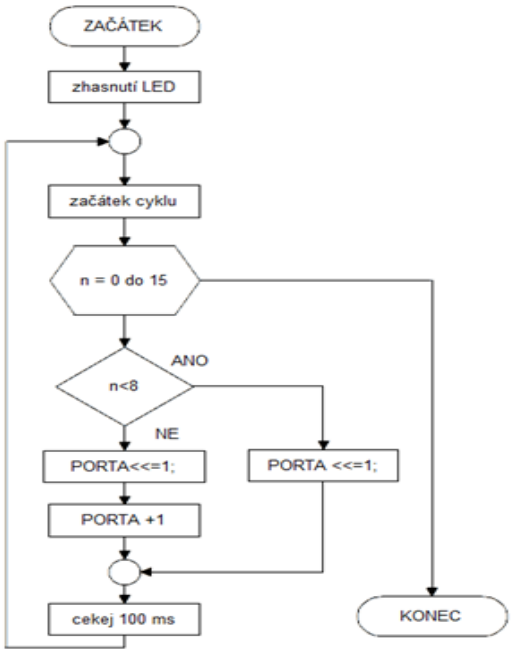
```
6. TCCR1B |= ((1 << CS10) | (1 << CS11)); //čítač/časovač 1 s předděličkou
   64
7. PORTA=0b10101010; //rozsvícení lichých LED diod
8. for (;;)          //nekonečný cyklus
9.  {
10.  if (TCNT1 >= 65535)
11.   {
12.   PORTA = ~PORTA; //změna LED
13.   TCNT1 = 0;      // reset hodnoty čítače/časovače 1
14.   }
15.  }
16.}
```

5.1.2 Postupné rozsvěcování a zhasínání LED diod

Na něco obtížnějším příkladu si ukážeme, jak pomocí bitového posunu rozsvěcovat a zhasínat LED diody. Bitový posun je docela jednoduchá věc. Při tomto úkolu už je zapotřebí užít nějaký cyklus a větvení. Já jsem v příkladu užil cyklus for (je ho vhodné použít, když víme, kolik bude opakování) a větvení if – else.

Zápis bitového posunu vypadá takto: PORTA <<=1

Bitový posun se tedy značí << posun doleva (zprava doplňuje nuly), >> posun doprava (zleva doplňuje nuly). Číslo, v našem případě číslo 1, značí, o kolik se má posun uskutečnit. Např. když máme na portu A bitovou kombinaci 1111111, tak po posunutí doleva bude kombinace vypadat takto 11111110.



Obr. 33 – vývojový diagram

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
DDRA = 0xff; //port A jako výstupní
while(1)
 {
 PORTA = 0b11111111; //zhasnutí všech LED diod
 for(unsigned char n=0;n<16;n++)
  {
  if (n<8)     //dokud nedojdeš na poslední pozici
   {
   PORTA <<=1; //tak posunuj doleva
   }
  else         //když si dojel nakonec
   {
   PORTA <<=1;    //tak posunuj doleva
   PORTA |= 0x01; //a zprava doplň jedničku
   }
  _delay_ms (100); //rychlost kroku 100 ms
  }
 }
}
```

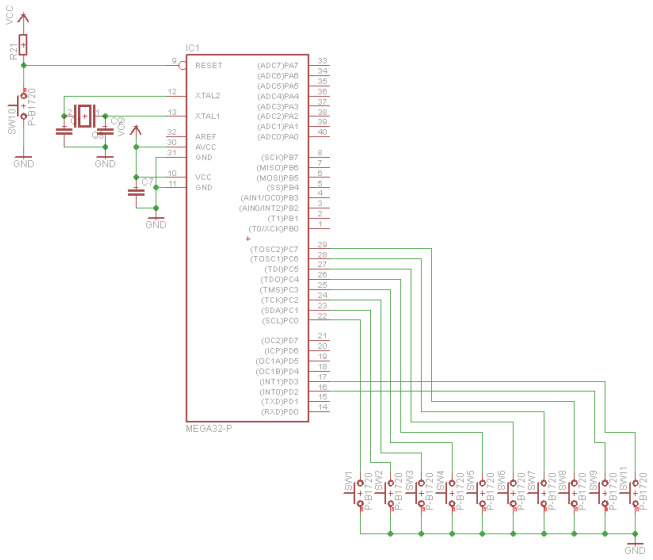
5.1.3 Světelný had

Program vychází z předchozího příkladu. Jen jsme změnili podmínku při větvení.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
DDRA = 0xff;        //port A jako výstupní
PORTA = 0b11111111; //zhasnutí LED diod
while(1)
 {
 for(unsigned char n=0;n<8;n++)
  {
  if (n<4)    //počet dílků hada => 4
   {
   PORTA <<=1; //posun o jednu pozici doleva, zprava se doplní 0
   }
  else
   {
   PORTA <<=1; //bitový posun o jednu pozici doleva
   PORTA |= 0x01; //zprava se doplní jednička bitovým součtem
   }
  _delay_ms (500); //rychlost jízdy hada 500 ms
  }
 }
}
```

5.2 Práce s tlačítky

Schéma zapojení:

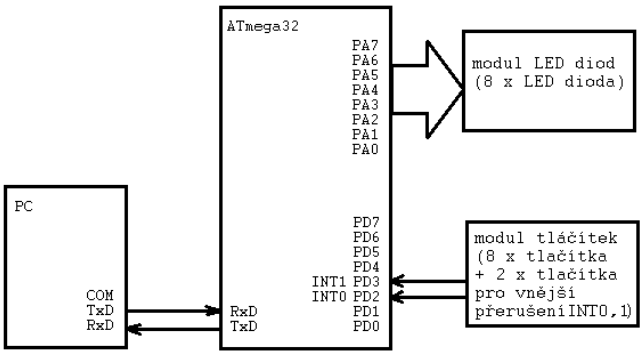


Obr. 34 – schéma zapojení tlačítek

5.2.1 Využití externího přerušení k rozsvícení LED diody

Jedná se o úlohu s využití externího přerušení. Příklad je poměrně jednoduchý, ale je z něj dobře patrná funkce externího přerušení. Na pinech PD2 a PD3 (piny přiřazené externímu přerušení INT0 a INT1) máme připojena tlačítka. Po stlačení tlačítka na INT0 se LED dioda rozsvítí a po stlačení tlačítka na INT1 se LED dioda zhasne. Zvolíme reakci na sestupnou hranu (falling edge). A to z důvodu, že máme tlačítka spínané k zemi. Na těchto vstupech aktivujeme pull – up rezistory.

Blokové schéma:



Obr. 35 – blokové schéma

Potřebné vektory přerušení:

- ISR(INT0_vect) pro INT0
- ISR(INT1_vect) pro INT1

Nastavení reakce na sestupnou hranu podle následujících tabulek:

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Table 34. Interrupt 1 Sense Control

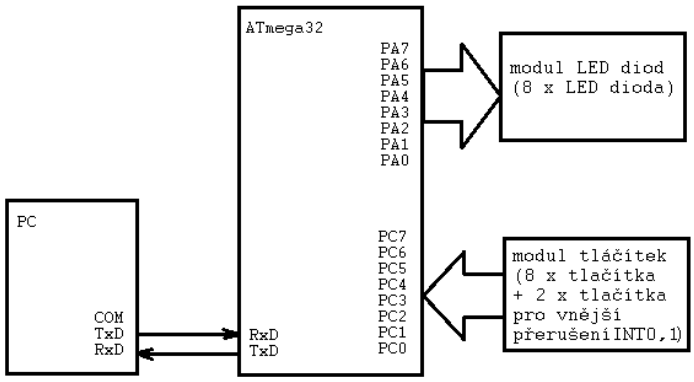
ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

```
1. #define F_CPU 16000000UL
2. #include <avr/io.h> //knihovna vstupů / výstupů
3. #include <avr/interrupt.h> //knihovna přerušení
4. ISR(INT0_vect)       // vektor přerušení INT0
5. {
6. PORTA |= (1 << PD7); // zapni LED
7. }
8. ISR(INT1_vect)       // vektor přerušení INT0
9. {
10. PORTA &= ~(1 << PD7);// vypni LED
11. }
12. int main() //hlavní funkce
13. {
14. DDRA |= (1 << PD7); //PD7 jako výstupní, připojena LED dioda
15. DDRD &= ~( (1 << PD2) | (1 << PD3) ); //PD2,PD3 jako vstupní, externí
16.                                       // přerušení od tlačítka
17. PORTD |= (1 << PD2) | (1 << PD3); // Zapnutí interních pull-up rezistorů
18. MCUCR |= (1 << ISC01); // reakce na sestupnou hranu INT0
19. MCUCR |= (1 << ISC11); // reakce na sestupnou hranu INT1
20. GICR |= (1 << INT1) | (1 << INT0); // povolení přerušení od INT1 a INT0
21. sei();    //globální povolení přerušení
22. while(1); // nekonečný cyklus
23. return 0;
24. }
```

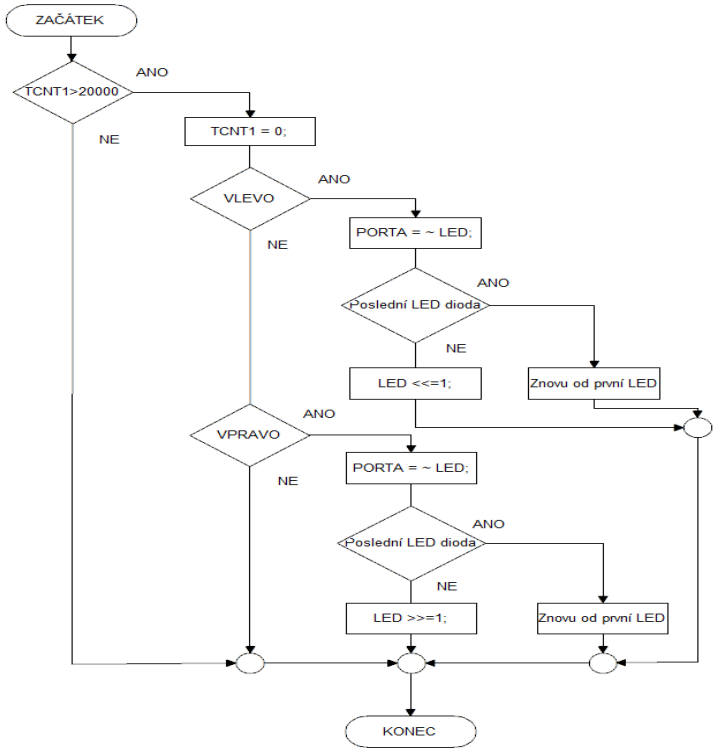
5.2.2 Změna směru rozsvěcování diod pomocí tlačítka

V tomto příkladu si ukážeme, jak stiskem tlačítka změnit směr rozsvěcování diod. Při tomto úkolu využijeme bitového posunu pro realizaci postupného rozsvěcování LED diod. Dále použijeme větvení (pro testování tlačítka a pro změnu směru) a čítač/časovač1 v základním módu s předděličkou 64. Předděličku f_{osc}/ 64 nastavíme v registru TCCR1B zapsáním 1 do bitů CS10 a CS11. Testováním hodnoty v registru TCNT1, ve kterém je uložena hodnota čítače/časovače1, budeme řídit rychlost posunu LED diod a také tento registr využijeme na detekci stisknutého tlačítka (ochrana proti zákmitům).

Blokové schéma:



Obr. 36 – blokové schéma



Obr. 37 – vývojový diagram

```
#define F_CPU 16000000UL
#include <avr/io.h> //knihovna vstupů výstupů
#include <util/delay.h> //knihovna čekacích funkcí

int main(void)
{
unsigned char STISK; //zadefinování proměnných
char SMER = 0xFF; //proměnná na řízení směru
char LED = 0x01; //proměnná na nastavení poslední LED diody
DDRA = 0xFF; //nastavení portu A jako výstupní
PORTA = 0xFF; //nastavení pinů na celém portu A na log.1
DDRD = 0x7F; //nastavení PD7 jako vstupní
PORTD = 0x80; //zapnutí pull-up rezistoru na PD7
STISK = (PIND & (1<<PD7)); // slouží k maskování hodnoty portu D, tj. v
//závislosti na stisknutí tlačítka bude výsledek buď 00000100b nebo
//00000000b
TCCR1B |= ((1 << CS10) | (1 << CS11)); // čítač1 s předděličkou 64
while(1)
 {
 if (STISK == (PIND & (1<<PD7))) //kontrola, jestli je tlačítko stisknuté
  {
  ; //prázdný příkaz
  }
 else if(TCNT1 >= 20000) //ochrana proti zákmitům na tlačítku
  {
```

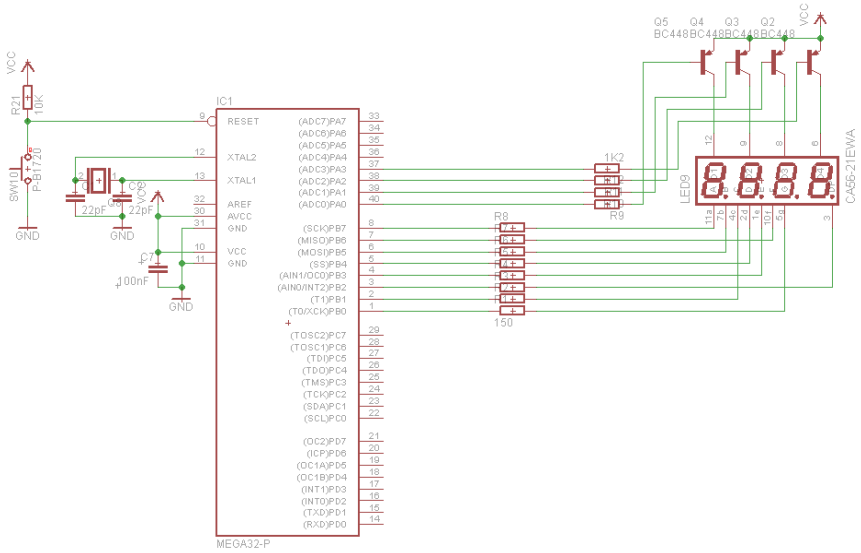
```
  STISK = (PIND & (1<<PD7));
   if (!(PIND & (1<<PD7))) SMER = ~SMER; //jestliže je tlačítko
                                         //stisknuto, změň směr
  }
 if (TCNT1 >= 20000) //nastavuje rychlost posunu
  {
  TCNT1 = 0; //vynulování registru čítače/časovače 1
  if(SMER == 0xFF) //běh LED diod doleva
   {
   PORTA=~LED; //poslání dat na PORTA
   if(LED == 0x80) LED = 0x01; //když dojdeme nakonec LED dioda na PA7,
                               //tak se vrať zpět na začátek=>PA0
   else LED <<=1; //bitový posun o 1 místo doleva, zprava se doplní nula
   }
  if(SMER == 0x00) //běh LED diod doprava
   {
   PORTA=~LED; //poslání dat na PORTA
   if(LED == 0x01) LED = 0x80; //když dojdeme nakonec LED dioda PA0, tak
                               //se vrať zpět na začátek=>PA7
   else LED >>= 1; //bitový posun o 1 místo doleva, zleva se doplní nula
   }
  }
 }
}
```

Umět ovládat tlačítka je poměrně důležité. V dalších kapitolách se o tom přesvědčíme (např. u hodin můžeme nastavovat čas, u kalkulačky zadávat čísla atd.).

5.3 Práce se čtyřmístným 7 – segmentovým LED displejem

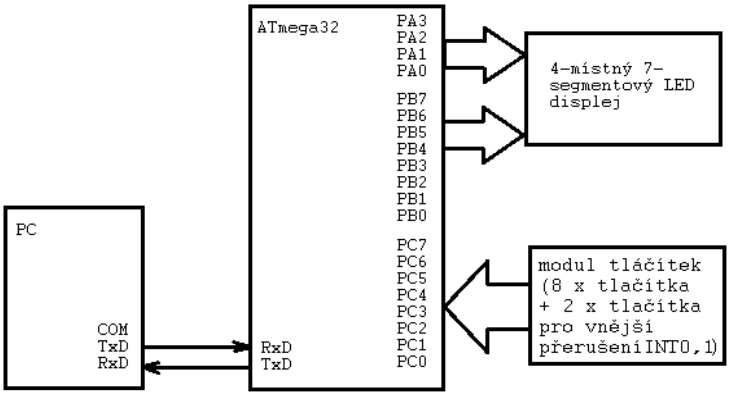
Na úvod bych začal tím, že ovládání 7 – segmentového displeje je, ač se to nemusí na první pohled zdát, jednoduché. Na mé UNI desce je použit čtyřmístný 7 – segmentový LED displej se společnou anodou. Jednotlivé čísla tedy budou zobrazeny, pokud bude na anodu připojena log. 1 (např. 5V) a na jednotlivé segmenty přivedeme log. 0 (např. 0V). Anody displeje jsou připojeny přes spínací tranzistory typu PNP (sepnutí při logické nule na výstupu pinu, pin připojen do báze tranzistoru).

Schéma zapojení:



Obr. 38 – schéma zapojení 4-místného 7-segmentového displeje

Blokové schéma:



Obr. 39 – blokové schéma zapojení 7-segmentového displeje

5.3.1 Náhodné číslo

V tomto příkladu si ukážeme, jak vygenerovat náhodné číslo za pomoci čítače / časovače 0. Využijeme k tomu vektor přerušení *ISR(TIMER0_OVF_vect)* do něhož vložíme proměnnou, která se bude při přerušení inkrementovat. Po stisku tlačítka tuto proměnnou budeme dělit 6. A nyní potřebujeme zbytek po tomto dělení. K tomu slouží operand %, ten nám vrací celočíselný zbytek po dělení. Takže nám může vyjít zbytek – 0, 1, 2, 3, 4, 5. Samozřejmě záleží na tom, čím danou proměnnou dělíme. A poté už budeme jen z pole znaků vybírat příslušné číslo.

```
#define F_CPU 16000000UL
#include <avr/io.h>         //knihovna vstupů/výstupů
#include <util/delay.h>     //knihovna čekacích funkcí
#include <avr/interrupt.h> //knihovna přerušení

unsigned char vyber(unsigned char cislo); //nadefinování funkce „vyber“
void losovani(unsigned char doba);       //nadefinování funkce „losovani“
#define c1 0b11011101 //číslo 1
#define c2 0b01000110 //číslo 2
#define c3 0b01001100 //číslo 3
#define c4 0b10011100 //číslo 4
#define c5 0b00101100 //číslo 5
#define c6 0b00100100 //číslo 6

//definice a inicializace pole znaků
const unsigned char CISLA[] = {c1, c2, c3, c4, c5, c6,};
//pomocná proměnná pro cyklus for ve funkci „vyber“ a „losovani“
unsigned char n = 0;
//proměnná, která „čítá“ přerušení od čítače
unsigned char x = 0;

int main(void)
{
unsigned char cislo = 0;
DDRA = 0x01; //nastavení pinu A0 jako výstup
PORTA = 0xFE; //zapnutí jedné číslice
DDRB = 0xFF; //brána B nastavena jako výstupní
DDRC = 0xFE; //brána C nastavena jako vstupní
PORTC = 0x01; //zapnutí pull-up rezistoru

TCCR0 = 0x01; // čítač0 bez předděličky
TIMSK = 0x01; // přerušení od čítače0
sei(); // globální povolení přerušení

while(1)//nekonečný cyklus
 {
 if(!(PINC & (1<<0))) //když tlačítko na PC0 stisknuto, vykonej příkazy
  {
  losovani(3); //nastavení doby hodu kostkou
  //generování čísla o 1 do 6
  cislo = x % 6; //celočíselný zbytek po dělení 6
  PORTB = vyber(cislo); //pomocí funkce "vyber" vypiš na bráně B
                        //vylosované číslo
```

```
  }
 }
}

//funkce pro vybrání vylosovaného čísla
unsigned char vyber(unsigned char cislo)
{
unsigned char y = 0;
for( n=0; n<6; n++) //6 opakování => 6čísel
 {
 y = CISLA[cislo];  //vybrání čísla pomocí proměnné "cislo"
 }
n=0;      //vynulování pomocné proměnné
return y; //návratová hodnota funkce
}

//funkce pro zobrazování čísel během losování
void losovani(unsigned char doba)
{
unsigned char t;
for(t=0; t<doba; t++) //zobrazování čísel během losování
 {
 for(n=0; n<6; n++) //6 opakování=> 6 čísel
  {
  PORTB = CISLA[n]; //zobrazování čísel od 1 do 6 během losování
  _delay_ms(100); //prodleva mezi přepínáním číslic
  }
 n = 0; //vynulování pomocné proměnné
 }
}

ISR(TIMER0_OVF_vect) //přerušení pro přetečení časovače0
{
x++; // zvýší hodnotu v proměnné x
}
```

5.3.2 Stopky

Tento příklad už patří k těm náročnějším. Ke generování přesného času použijeme čítač/časovač0 s předděličkou 256. Ke generování potřebné frekvence musíme ještě zapsat hodnotu 131 do registru TCNT0. Od této hodnoty začíná čítač čítat. Tyto hodnoty jsou vypočítané pro krystal s frekvencí 16MHz. Nastavení těchto hodnot nám zaručí přerušení po 2ms. Čísla budeme číst z pole znaků, ve kterém máme uloženy binární kombinace odpovídající jednotlivým číslicím od 0 do 9 a tečce. Start/Stop stopek budeme ovládat tlačítkem připojeným na PC0. Reset stopek budeme generovat tlačítkem na PC1.

```
#define F_CPU 16000000UL
#include <avr/io.h>        //knihovna vstupů/výstupů
#include <avr/interrupt.h> //knihovna přerušení

volatile unsigned char pozice = 0;
volatile unsigned char vyber[] = {0,0,0,0};
```

```
7. volatile unsigned char zpozd = 0;
8. volatile unsigned int cas = 0;
9. volatile unsigned char stav = 0;
10. unsigned char stisk = 0; //pomocná proměnná pro obsluhu tlačítek
11.
12. #define c0 0b00000101 //0
13. #define c1 0b11011101 //1
14. #define c2 0b01000110 //2
15. #define c3 0b01001100 //3
16. #define c4 0b10011100 //4
17. #define c5 0b00101100 //5
18. #define c6 0b00100100 //6
19. #define c7 0b01011101 //7
20. #define c8 0b00000100 //8
21. #define c9 0b00011100 //9
22. #define c10 0b11111011 //.
23.
24. unsigned char znaky[] =
25. {c0, c1, c2, c3, c4, c5, c6, c7, c8, c9}; //pole znaků (čísel)
26.
27. unsigned char anoda[] =
28. {0b00001110,0b00001101,0b00001011,0b00000111}; //pole pro multiplex-
    přepínání anod
29.
30. ISR(TIMER0_OVF_vect)
31. {
32. TCNT0 = 131; //námi vypočtená předvolba, od které začíná čítač0 čítat
33. if(stav == 1) //povolení funkce stopek
34.  {
35.  if(++zpozd == 5) //2ms x 5 = 10ms
36.   {
37.   zpozd = 0; //vynulování proměnné zpozd-zajištění opětovného "zpoždění"
38.               // 10ms
39.   if(++cas == 9999) //max hodnota na LED 99,99s
40.    {
41.    cas = 0; //při přesáhnutí max hodnoty vynuluj
42.    }
43.   }
44.  }
45. vyber[0] = cas/1000;             //slouží pro zobrazení desítek s
46. vyber[1] = (cas%1000)/100;       //slouží pro zobrazení jednotek s
47. vyber[2] = ((cas%1000)%100)/10; //slouží pro zobrazení desítek ms
48. vyber[3] = ((cas%1000)%100)%10; //slouží pro zobrazení jednotek ms
49.
50. PORTB = 0b11111111; //zhasnutí všech segmentů displeje
51. if(++pozice == 4) pozice = 0; //přepínání anod
52.
53. //ošetření napsání tečky za jednotkami sekund
54. if(pozice == 1) PORTB = (znaky[vyber[pozice]])&c10;
55. else PORTB = (znaky[vyber[pozice]]); //posílání na bránu B data, které
56. // odpovídají jednotlivým cifrám
57. PORTA = (anoda[pozice]); //multiplex (přepínání anod)
58. }
59. int main(void)
60. {
61. DDRB = 0b11111111;  //PORTB výstup
62. DDRA = 0b00001111;  //PA0-PA3 výstup-spínání anod
63. DDRC = 0b11111100;  //PC0 a PC1 vstup
64. PORTC = 0b00000011; //zapnutí pull-up rezistorů na PC0 a PC1
```

```
65.
66. // Nastavení čítače/časovače 0
67. // 16MHz/256/125 = 500Hz, T = 1/500Hz = 2ms
68. TCNT0 = 131; //předdělička 125 (256 - 125 = 131), čítač
69.              //začne čítat od hodnoty 131
70. TCCR0 = (1<<CS02);  //dělička 256
71. TIMSK = (1<<TOIE0); //povolení přerušení od čítače 0 při přetečení
72. sei(); //globální povolení přerušení
73.
74. while(1)
75.  {
76.  if((PINC&0b00000011) == 3) stisk = 0; //když není tlačítko stisknuté,
77.  // tak vynuluj proměnnou stisk
78.  if(((PINC&0b00000001) == 0)&&(stisk == 0))
79.  //je-li tlačítko stisknuté a proměnná stisk=0, tak proveď následující
80.   {
81.   //spouštění a zastavování stopek
82.   stisk = 1;
83.   if(stav == 1) stav = 0;
84.   //jestliže stopky běží, tak je zastav, jestli ne, tak je zapni
85.   else stav = 1;
86.   }
87.
88.  if(((PINC&0b00000010) == 0)&&(stisk == 0)&&(stav == 0))
89.  //když podmínky platí, tak vynuluj stopky
90.   {
91.   cas = 0;
92.   zpozd = 0;
93.   }
94.  }
95. }
```

5.3.3 Hodiny

V tomto příkladu použijeme čítač/časovač 1 bez předděličky. Opět budeme potřebovat generovat přerušení pro určení poměrně přesného času. Časovač tedy běží s periodou 65536, přerušení je 244,140625 Hz. Interval mezi přerušeními je 4,096 ms. Zavedeme si pomocnou proměnnou *„delic“*, do které budeme přičítat čas intervalu v milisekundách (konkrétně 4096ms). Po dosáhnutí hodnoty 1000000 (1s) se připočte k proměnné *„sek“* jednička. Následně z pomocné proměnné „delic“, která musí být typu long, odečteme hodnotu 1000000 a tím nám v ní zůstane zbytek. Takže počet sekund se bude navyšovat většinou po 244 přerušeních, ale občas po 245 přerušeních. Takto dosáhneme docela přesného generování času. Znaky budeme opět číst z pole, ve kterém máme uloženy odpovídající binární kombinace. Na PC0 budeme přičítat minuty, na PC1 je budeme odečítat. Na PC2 budeme přičítat hodiny a na PC3 je budeme odečítat.

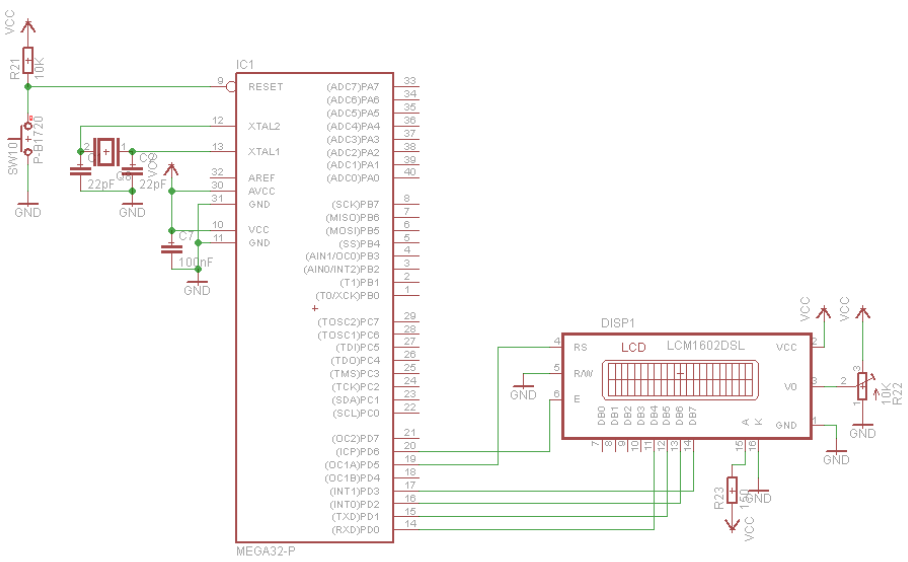
```
1. #define F_CPU 16000000UL
2. #include <avr/io.h>         //Vlož knihovnu vstupů a výstupů (PORT, DDR)
3. #include <util/delay.h>     //vlož knihovnu čekacích funkcí (_delay_ms() )
4. #include <avr/interrupt.h> //vlož knihovnu přerušení(sei() )
5. #include <stdio.h>
6.
7. #define c0 0b00000101 //0
8. #define c1 0b11011101 //1
9. #define c2 0b01000110 //2
10.#define c3 0b01001100 //3
11.#define c4 0b10011100 //4
12.#define c5 0b00101100 //5
13.#define c6 0b00100100 //6
14.#define c7 0b01011101 //7
15.#define c8 0b00000100 //8
16.#define c9 0b00011100 //9
17.#define c10 0b11111011 //.
18.
19.const unsigned char cisla[] = {c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
   c10};
20.
21.#define _ms(n) (17*n)      //součást zpožďovací smyčky
22.
23.void wait(unsigned int a) //zpožďovací smyčka – ochrana proti zákmitům
24.                          //tlačítek
25.{
26.volatile unsigned int b,c;
27.for(b=0;b!= a; b++)for(c=0;c!= 50;c++);
28.return;
29.}
30.
31.unsigned long delic=0;
32.unsigned char sek=0;
33.unsigned char min_1=0;
34.unsigned char min_10=0;
35.unsigned char hod_1=0;
36.unsigned char hod_10=0;
37.unsigned char zobraz=0;
38.
39.ISR(TIMER1_OVF_vect) //přerušení od časovače1, časovač1 běží s
40.                     //periodou 65536
41.{
42.delic = delic+4096;
43.//programové dělení přerušení od čítače1
44.if(delic >= 1000000){delic = delic - 1000000;sek++;};
45.
46.if(sek>59){min_1++;sek=0;}; //když bude 60s, tak přičti k minutám
   //jedničku a vynuluj sekundy
47.
48.if(min_1>9){min_1=0;min_10++;}; //když budou jednotky minut větší než 9,
   //tak vynulujjednotky minut a přičti jedničku k desítkám minut
49.
50.if(min_10>5){min_10=0;hod_1++;}; //když budou desítky minut větší jak
   //5(tzn. 60 minut), tak přičti k jednotkám hodin 1 a vynuluj desítky
   //minut
51.
52.if(hod_1>9){hod_1=0;hod_10++;}; //když budou jednotky hodin větší jak 9,
   //tak přičti jedničku k desítkám hodin a vynuluj jednotky hodin
53.
```

```
54. if(hod_10>1 && hod_1>3){hod_1=0;hod_10=0;}; //když budou desítky hodin
55. //větší než 1 a zároveň jednotky hodin větší než 3 (tzn. 24:00), tak
    //vynuluj jednotky a desítky hodin (čas na displeji 00:00)
56.
57. if(++zobraz==4) zobraz=0;
58. switch(zobraz)
59.  {
60.  case 0: //ukaž jednotky minut
61.   PORTA = 0b11110111;
62.   PORTB = (cisla[min_1]);
63.   break;
64.  case 1: //ukaž desítky minut
65.   PORTA = 0b11111011;
66.   PORTB = (cisla[min_10]);
67.   break;
68.  case 2: //ukaž jednotky hodin
69.   PORTA = 0b11111101;
70.   PORTB = (cisla[hod_1]) & (cisla[10]) ;
71.   break;
72.  case 3: //ukaž desítky hodin
73.   PORTA = 0b11111110;
74.   PORTB = (cisla[hod_10]);
75.   break;
76.  default:
77.   zobraz = 0;
78.   break;
79.  }
80. return;
81. }
82.
83. #define B1() (bit_is_clear(PINC,0))
84. #define B2() (bit_is_clear(PINC,1))
85. #define B3() (bit_is_clear(PINC,2))
86. #define B4() (bit_is_clear(PINC,3))
87. #define B_WAIT 100
88. int main(void)
89. {
90. TIMSK = 0x04; //přerušení od přetečení T1
91. TCCR1B = 0x01;
92.
93. DDRC = 0xF0;  //nastavení PC0 až PC3 jako vstupní
94. PORTC = 0x0F; //zapnutí pull-up rezistorů
95. DDRA = 0x0F;  //nastavení PA0 až PA3 jako výstupní (ovládání anod)
96. DDRB = 0xFF;  //brána B nastavena jako výstupní (ovládání segmentů)
97. PORTB = 0xFF; //zhasnutí všech segmentů
98.
99. sei();   //globální povolení přerušení
100.      while(1)
101.       {
102.       if(B1()) //přičítání minut
103.        {
104.        wait(_ms(B_WAIT));
105.        min_1++; //přičti minutu
106.        sek=0;   //vynuluj sekundy
107.        }
108.       if(B2()) //odečítání minut
109.        {
110.        wait(_ms(B_WAIT));
111.
```

```
112.       if (min_1 > 0) //když jednotky min > 0, odečti minutu
113.        min_1--;
114.       else            //jinak proveď následující příkazy
115.        {
116.        min_1 = 9;     //nastav jednotky min na 9
117.       if (min_10 > 0)//když desítky min větší než 0, tak odečti jedničku
118.                       //od desítek min
119.        min_10--;
120.       else //jinak nastav desítky minut na 5
121.        {
122.        min_10 = 5;
123.        if (hod_1 > 0) //když jednotky hodin větší než 0
124.         hod_1--;      //tak odečti jednotky hodin
125.        else
126.         {
127.         hod_1 = 9; //jinak nastav jednotky hodin na 9
128.         hod_10--;  //a odečti jedničku z desítek hodin
129.         }
130.        }
131.       }
132.      sek=0;   //vynuluj sekundy
133.      }
134.      if(B3()) //přičítání hodin
135.       {
136.       wait(_ms(B_WAIT));
137.       hod_1++; //přičti hodinu
138.       sek=0;   //vynuluj sekundy
139.       }
140.      if(B4())  //odečítání hodin
141.       {
142.       wait(_ms(B_WAIT));
143.       if (hod_1 > 0) //když jednotky hodin větší než 1
144.        hod_1--;      //tak odečti z jednotek hodin jedničku
145.       else
146.        {
147.        hod_1 = 9;   //jinak nastav jednotky hodin na 9
148.        if(hod_10>0) //když desítky hodin větší než 0
149.         hod_10--;   //odečti jedničku z desítek hodin
150.        else
151.         {
152.         hod_10=2; //jinak nastav desítky hodin na 2
153.         hod_1=3;  //a jednotky hodin na 3
154.         }
155.        }
156.       sek=0; //vynuluj sekundy
157.        }
158.       }
159.      }
```

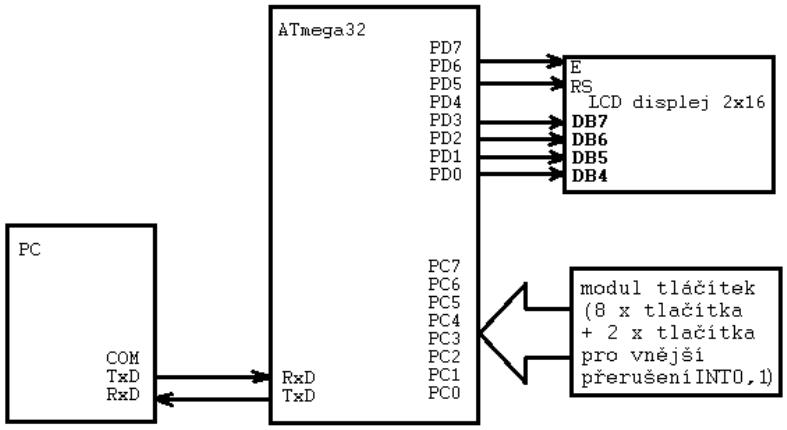
5.4 Práce s LCD displejem

Schéma zapojení:



Obr. 40 – schéma zapojení LCD displeje

Blokové schéma:



Obr. 41 – blokové schéma zapojení LCD displeje

V mém zapojení je LCD displej 2x16 s řadičem HD44780. V kapitole č. 3 jsme si ukázali, jak přidat knihovnu potřebnou pro práci s tímto displejem a jak si ji podle svého zapojení upravit. Nyní si ukážeme několik příkladů s využitím LCD displeje. Displej je připojen na bránu D, konkrétně na PD6-E, PD5 – RS, PD0 – D4, PD1 – D5, PD2 – D6 a PD3 – D7.

V přidané knihovně jsou tyto funkce:

lcd clrscr(void) ; vymazání LCD displeje

lcd home(void); nastaví kurzor na výchozí pozici

lcd gotoxy(uint t x, uint8 t y); nastavení kurzoru na zadanou pozici x a y (indexování od 0)

lcd init(uint8 t dispAttr); inicializace LCD displeje, jako parametr lze zvolit následující možnosti

- LCD_DISP_OFF –displej je vypnutý
- LCD_DISP_ON –displej je zapnutý, kurzor je vypnutý
- LCD_DISP_ ON – displej je zapnutý, kurzor je zapnutý
- LCD_DISP_ ON – displej je zapnutý, kurzor je zapnutý a bliká

lcd putc(char c) – zobraz znak

lcd puts(const char *s); - zobraz řetězec znaků

lcd puts(const char *program s); - zobraz řetězec znaků uložený v paměti Flash

lcd command(uint8 t cmd); - vyšle příkaz na displej

Obr. 42 - Binární kombinace pro zobrazení speciálních znaků na displeji[9]

5.4.1 Výpis požadovaných znaků na LCD displeji

Tento příklad ukazuje základní práci s LCD – výpis určitého textu. Využijeme funkci lcd_puts (vypsání znaků) a funkci lcd_gotoxy (posunutí znaků na požadovanou pozici xy).

```
1. #define F_CPU 16000000UL
2. #include <avr/io.h>
3. #include "lcd.h"
4. int main()
5. {
6. UCSRB = 0;                    //inicializace bez tohoto řádku nejsou
   funkční piny PD0 a PD1 lcd_init(LCD_DISP_ON_CURSOR_BLINK); //inicializace
   displeje
7. lcd_gotoxy(2,0);              // jdi na pozici x=2, y=0
8. lcd_puts("Lukas Holek");      // vypiš text
9. lcd_gotoxy(0,1);              // jdi na pozici x=2, y=0
10. lcd_puts("maturitni prace"); // vypiš text
11. return 0;
12. }
```

5.4.2 Kalkulačka

V tomto příkladu si ukážeme práci s čísly pomocí funkce sprintf(). V kapitole č. 3 jsme si ukázali, jak upravit parametry projektu, abychom mohli vypisovat desetinná čísla. Takže nyní nám nebrání nic v napsání kódu. Hojně zde využijeme větvení programu. Naše kalkulačka bude umět: sčítat, odečítat, násobit, dělit a umocňovat celá čísla. Funkce vstupů: Tlačítka připojeného na PC0 zadáváme první číslo, na PC1 určujeme početní operaci, na PC2 zadáváme druhé číslo, PC3 má funkci „rovná se“, na PC4 nulujeme první číslo, na PC5 nulujeme druhé číslo a na PC6 uvádíme kalkulačku do počátečního stavu.

```
1. #define F_CPU 16000000UL //Makro pro knihovnu delay
2. #include <util/delay.h>
3. #include <avr/io.h>
4. #include <math.h>
5. #include "lcd.h"
6.
7. #define _ms(n) (17*n)       //součást zpožďovací smyčky
8.
9. void wait(unsigned int a)    // smyčka- ochrana proti zákmitům tlačítek
10. {
11.     volatile unsigned int b,c;
12.     for(b=0;b!= a; b++)for(c=0;c!= 50;c++);
13.     return;
14. }
15. #define B1() (bit_is_clear(PINC,0))
16. #define B2() (bit_is_clear(PINC,1))
17. #define B3() (bit_is_clear(PINC,2))
18. #define B4() (bit_is_clear(PINC,3))
19. #define B5() (bit_is_clear(PINC,4))
```

```
#define B6() (bit_is_clear(PINC,5))
#define B7() (bit_is_clear(PINC,6))
#define B_WAIT 300
int main(){
UCSRB = 0;           // aktivace pinů PD0 a PD1, kvůli bootloaderu

DDRC = 0b10000000;  //PC0 a PC1 vstup
PORTC = 0b01111111; //pull-up rezistory
int x = 0;           //první číslo
int y = 0;           //druhé číslo
int r=0;             //početní operace
int vysledek=0;      //tlačítko "rovná se"
float vypocet=0;     //proměnná, do které se ukládá vypočítaná hodnota

while(1){

char pole[32];          //pole pro čísla převedená na řetězce znaků

lcd_init(LCD_DISP_ON);  //inicializace displeje, zapnutí displej
lcd_gotoxy(0,0);        //počáteční pozice

if(B1())                //nastavení prvního čísla
 {
 wait(_ms(B_WAIT));     //ochrana proti zákmitům tlačítka
 x++;
 }
sprintf(pole, "%d", x); //převeď číslo do řetězce
lcd_puts(pole);         //vypiš první číslo

if(B3())    //výběr požadované operace
 {
 wait(_ms(B_WAIT));
 r++;       //změna operace
 }
switch(r)
 {
 case 0: lcd_gotoxy(5,0); //sčítání
         lcd_puts("+");   //zobraz znak na pozici x=5,y=0
         vypocet=(float)x+y; //ulož výsledek operace do proměnné vypocet

         break;
 case 1: lcd_gotoxy(5,0); //odečítání
       lcd_puts("-");
       vypocet=(float)x-y;
       break;
 case 2: lcd_gotoxy(5,0); //násobení
       lcd_puts("*");
       vypocet=(float)x*y;
       break;

 case 3: lcd_gotoxy(5,0); //dělení
       lcd_puts("/");
       vypocet=(float)x/y;
       break;

 case 4: lcd_gotoxy(5,0); //mocnina
       lcd_puts("^");
       vypocet= (double) pow(x,y);
       break;
```

```
78.  default:
79.        r = 0;             //začni vybírat znovu od začátku
80.        break;
81.  }
82. if(B2())    //nastavení druhého čísla
83.  {
84.  wait(_ms(B_WAIT));
85.  y++;
86.  }
87. lcd_gotoxy(10,0);
88. sprintf(pole, "%d", y);
89. lcd_puts(pole);
90.
91. if(B4()) //vypsání výsledku=> tlačítko "="
92.  {
93.  wait(_ms(B_WAIT));
94.  vysledek++;
95.  }
96.
97. lcd_gotoxy(0,1); //vypiš na dolní řádek
98. if(vysledek>=1)  //když bylo stisknuto tlačítko PC3, tak vypiš výsledek
99.  {
100.       if(r==3 && y==0) lcd_puts("nelze!!!"); //podmínka pro dělení
101.
102.       else if(r==4 && x==0 && y==0) lcd_puts("nelze!!!");
103.      //podmínka pro  umocňování
104.       else if(r==4)
105.        {
106.        sprintf(pole, "=%.0f",vypocet); //umocněné číslo do řetězce
107.        lcd_puts(pole); //vypiš umocněné číslo
108.                          }
109.       else
110.        {
111.        sprintf(pole, "=%.3f",vypocet); //číslo - tři desetinná místa
112.        lcd_puts(pole);   //vypiš toto číslo
113.        }
114.       }
115.
116.      if(B5())    //při stisku PC4 vynuluj první číslo
117.        {
118.        wait(_ms(B_WAIT));
119.        x=0;
120.        }
121.
122.      if(B6())    //při stisku PC5 vynuluj druhé číslo
123.        {
124.        wait(_ms(B_WAIT));
125.              y=0;
126.        }
127.      if(B7())    //reset kalkulačky
128.        {
129.        wait(_ms(B_WAIT));
130.        vysledek=0;
131.        x=0;
132.        y=0;
133.        r=0;
134.        }
135.       }
136.      }
```

5.4.3 Výpis hodnoty AD převodníku

AD převodník je elektrické zařízení, které převádí analogový signál na digitální (číslicový). Tyto převodníky se dnes využívají např. pro měření napětí, proudu, teploty…

V tomto příkladu si ukážeme, jak číst hodnoty z AD převodníku. Aby bylo měření přesné, musí být frekvence AD převodníku 50 – 200 kHz. V mém zapojení je krystal s 16 MHz; takže dělící poměr bude 128 (vyšší už není a s nižším poměrem bychom nedosáhli přesnosti). Nastavení dělícího poměru provádíme nastavením bitů ADPS0, ADPS1, ADPS2 v registru ADCSRA. V tomto registru nastavujeme: začátek převodu (bitem ADCS), povolení převodu (bitem ADEN), je zde také příznak přerušení, tento bit se nastaví na log. 1, když skončí převod (bit ADIF). Obsahuje ještě další, ale v tomto příkladu je nepotřebujeme. Další potřebný registr je ADC, který obsahuje výsledek převodu. Poslední registr je ADMUX, ve kterém vybíráme vstupní kanál nastavením bitů MUX3 až MUX0. Dále v něm nastavujeme zdroj referenčního napětí bity REFS0 a REFS1.

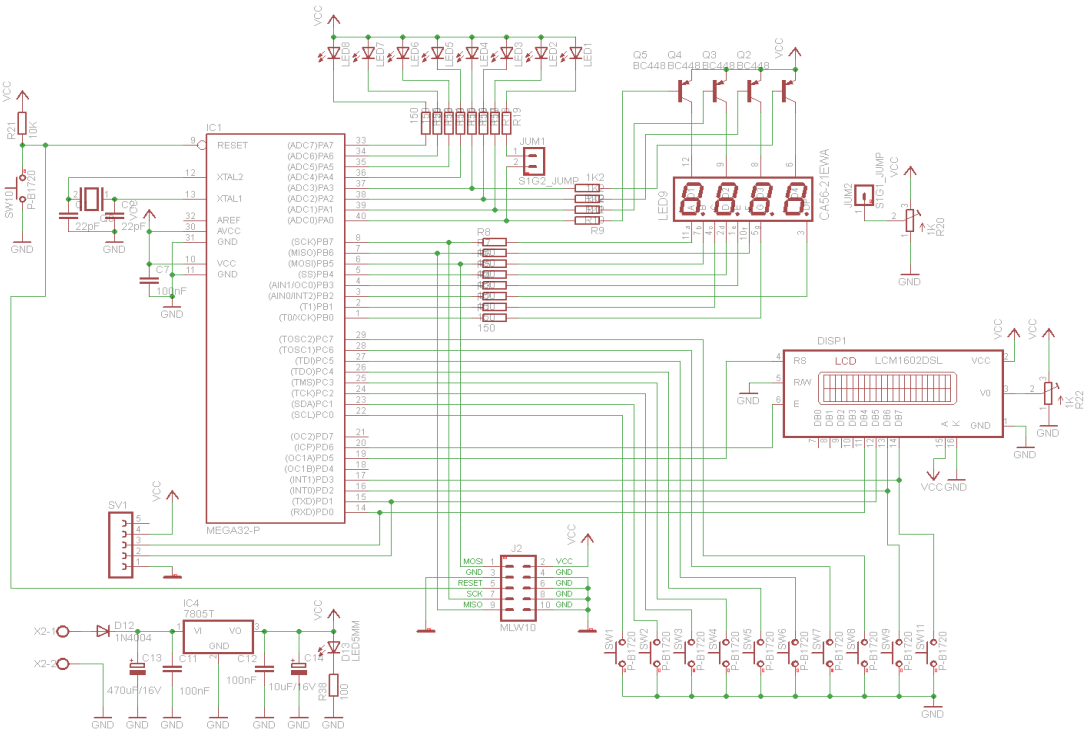
```
1. #define F_CPU 16000000UL //Makro pro knihovnu delay
2. #include <avr/io.h>
3. #include <util/delay.h>
4. #include <avr/interrupt.h>
5. #include <stdlib.h>
6. #include <stdio.h>
7. #include "lcd.h"
8.
9.
10.
11. unsigned int cteniADC(unsigned char kanal)
12. {
13.     ADMUX &= 0xF0;
14.     ADMUX |= kanal & 0x0F;        //vymaskování nepoužitých bitů
15.     ADCSRA |= (1 << ADSC);        // začátek převodu
16.     while(!(ADCSRA & (1<<ADIF))); // čekání na příznak převodu
17.     return ADC;     //  návratová hodnota je výsledek ad převodu
18. }
19.
20. int main()
21. {
22. UCSRB = 0;
23. char pole[32];
24. float u;
25. int adc;
26.
27. ADMUX |=  (1 << REFS0); // referenční napětí 5V
28.
29. // povolení AD převodníku, příznak ukončení převodu předdělička 128
30. ADCSRA |= (1 << ADEN)|(1 << ADIF)|(1 <<ADPS2)|(1 <<ADPS1) |(1 <<ADPS0);

31.
```

```
32. sei(); // globální povolení přerušení
33. lcd_init(LCD_DISP_ON); //zapnutí displeje
34. 
35. while(1)
36.  {
37.  adc = cteniADC(0);          // vybraný nultý kanál AD => PA0
38.  u = ((float)adc*5/1024);    // výpočet napětí ve voltech
39.  sprintf(pole,"ADC:%d \nU:%.3fV",adc,u); //převod na řetězec
40.  lcd_clrscr();           //vymazání displeje
41.  lcd_gotoxy(0,0);    //pozice x=0 a y=0
42.  lcd_puts(pole);     //vypiš hodnoty na displej
43.  _delay_ms(500);     //čekej 0,5 s
44.  }
45. }
```

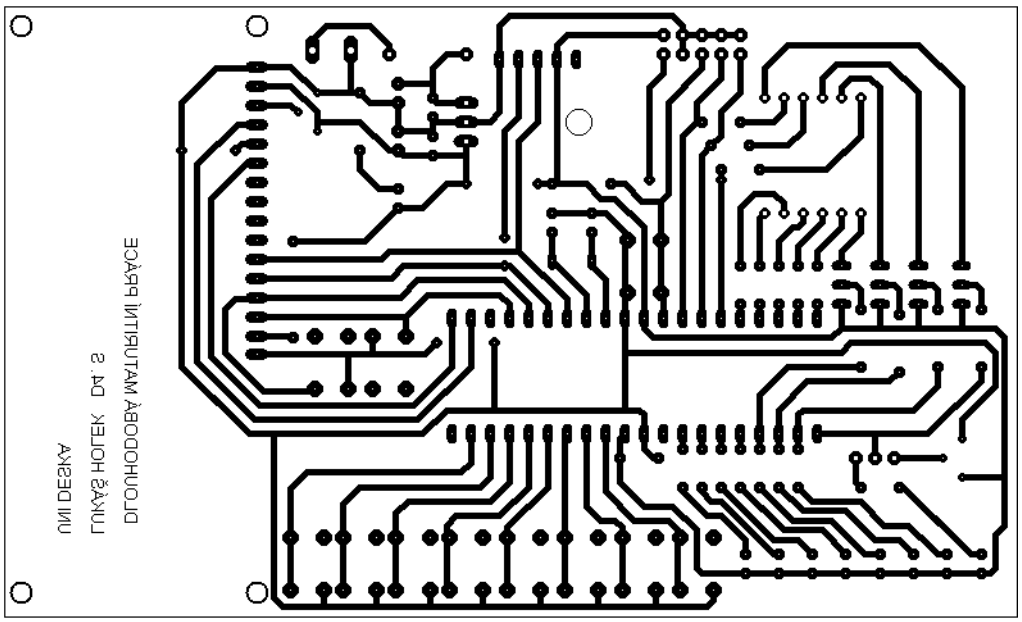
6 UNI deska

Pro práci s mikroprocesorem byla navržena UNI deska. Tato deska je navržena pro typ ATmega32, ale může sloužit i pro vývoj aplikací např. s Atmega16. Deska je navržená tak, že práce s ní je velice jednoduchá. Nemusíme zdlouhavě propojovat příslušné periferie, jelikož už jsou přímo připojené na příslušné brány mikroprocesoru. Deska obsahuje např. 7 – segmentový LED displej, LCD displej, LED diody, tlačítka atd.

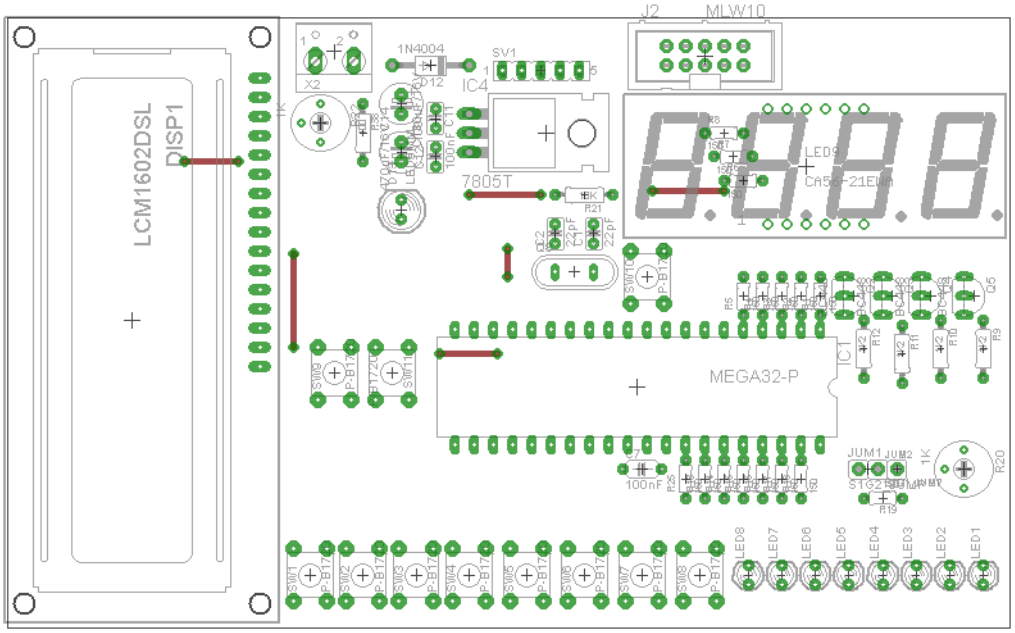
6.1 Schéma



6.2 Návrh plošného spoje



6.3 Rozmístění součástek



6.4 Technické parametry

- Velikost desky
- Procesor ATmega32
- LCD 2x16 znaků s řadičem, podsvícený
- 4 - místný 7 – segmentový LED displej
- 10 tlačítek
- 8 LED diod
- ISP programovací konektor

Závěr

Předložená práce představuje výslednou verzi sady úloh s mikroprocesorem ATmega32 v jazyce C. V této práci jsou popsány dnes používané 8 – bitové mikroprocesory, architektura a vlastnosti mikroprocesoru ATmega32. Dále je zde stručně popsáno vývojové prostředí AVR Studio 4, ve kterém byly úlohy vytvořeny. V této práci je také postup vývoje mikroprocesorové aplikace, základní příkazy v programovacím jazyce C. Nedílnou součástí maturitní práce jsou úlohy, ve kterých je popsána práce s vybranými periférierniemi. K této práci byla zkonstruována UNI deska, na které byly úlohy odzkoušeny.

Seznam zdrojů a použité literatury

[1] ATMEL CORPORATION: *Atmega32 datasheet* [online]. [cit. 2013-02-25]. Dostupné z: http://www.atmel.com/Images/doc2503.pdf.

[2] ATMEL CORPORATION [online]. [cit. 2013-02-25]. Dostupné z: http://www.atmel.com/.

[3] Microchip Technology Inc [online]. [cit. 2013-03-14]. Dostupné z: http://www.microchip.com/

[4] Freescale Semiconduktor [online]. [cit. 2013-03-14]. Dostupné z: http://www.freescale.com/

[5] STMicroelectronics [online]. [cit. 2013-03-14]. Dostupné z: http://www.st.com

[6] NXP Semiconductors [online]. [cit. 2013-03-14]. Dostupné z: http://www.nxp.com/

[7] Fujitsu Semiconductor [online]. [cit. 2013-03-14]. Dostupné z: http://www.fujitsu.com/

[8] VÁŇA, Vladimír. Mikrokontroléry Atmel AVR: programování v jazyce C [online]. Praha:BEN, 2003 [cit. 2013-03-12]. ISBN 80-7300-102-0.

[9] KARAS, Ondřej. *Kvetakov.net: články AVR* [online]. [cit. 2013-02-25]. Dostupné z: http://www.kvetakov.net/clanky/avr.

[10] KRÁL, Václav. Koutek AVR [online]. [cit. 2013-03-14]. Dostupné z: http://www.ok1kvk.cz/