TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Zařízení pro analýzu protokolu komunikační linky

Bakalářská práce

Autor: **Jiří Scheuer**

Vedoucí práce: Ing. Jaroslav Buchta

Konzultant:

Zadání:

Název tématu: **Analyzátor sériové komunikační linky**

Zásady pro vypracování:

1. Seznamte se s mikrokontroléry řady ATMEGA
2. Navrhněte hardware a komunikační rozhraní
3. Vytvořte software pro mikrokontrolér
4. Vytvořte testovací aplikaci pro načtení a zobrazení zaznamenaných údajů

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: 40 stran

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářské práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Rád bych poděkoval Ing. Jaroslavu Buchtovi za veškerou pomoc, cenné rady a podmětné připomínky při vedení mé práce.

Anotace

Tato bakalářská práce se zabývá analýzou komunikační linky pomocí mikrokontroléru. Cílem je navrhnout jednoduché zařízení a obslužný software, který umožní zobrazení dat v PC. Praktická část popisuje konkrétní realizaci zařízení založenou na mikrokontroléru Atmel ATMEGA16 jeho propojení s PC a obslužný software vytvořený v prostředí Delphi.

OBSAH:

Seznam použitých symbolů

0b…	prefix binárních čísel
0x …	prefix hexadecimálních čísel
b, bit	binary digit – dvojková číslice s hodnotami 0 nebo 1
B, Byte	jednotka kapacity, slovo o délce 8 bitů
GND	GrouND – zemní potenciál
EEPROM	Elektronický programovatelná paměť
RISC	Reduced Instruction Set Computer – Počítač s redukovanou instrukční sadou
SMD	Elektronické součástky pro povrchovou montáž
Word	slovo o délce 16 bitů

Úvod

V dnešní době řídí spousty technologických procesů, měření a testů počítače, které komunikují po několika datových vodičích. Ne vždy je však možné použít několik klasických či průmyslových PC nebo osciloskopů, které by zpracovaly komunikační linku. Lepším řešením je použití jednoduchého jednoúčelového zařízení na analýzu dat, které se připojí přes běžné standardizované rozhraní k PC a zobrazí naměřená data.

Základní požadavky na takového zařízení jsou:

- dostatečná rychlost ukládání dat,
- velikost paměti,
- kompaktní rozměry, malá spotřeba, mechanická odolnost,
- kompatibilní pro připojení k PC a cena.

Pro snadný přenos dat do PC jsme zvolili připojení přes sériové rozhraní RS232, které obsahuje většina počítačů a je podporován většinou operačních systémů.

V následujícím textu bude nejprve popsán mikrokontrolér ATMEGA16 jeho programování pomocí softwaru AVR Studio v programovacím jazyce C a komunikační protokol mezi PC a zařízením. V praktické části je dále popsán návrh a realizace analyzátoru, komunikačního rozhraní RS232. Poslední část se zabývá návrhem softwaru, pro zobrazení získaných dat v aplikaci vytvořené pomocí Borland Delphi 7.

1. Teoretická část

V teoretické části jsou rozebrány základní vlastnosti mikrokontroléru ATMEGA16, software AVR Studio a komunikační protokol mezi PC a zařízením.

1.1. Mikrokontrolér Atmel ATMEGA

Pro danou úlohu jsem zvolil RISCový mikrokotrolér ATmega z rodiny AVR firmy Atmel. ATMEGA16 je 8-bitový jednočipový mikrokontrolér, který má velkou řadu periférií potřebnou pro naše použití a je u nás dobře dostupný za rozumné ceny. Nabízí dobrý poměr cena/výkon (periférie).

Stručně uvedu některé výhody:

- Vysoký výkon 16MIPS (16MHz),
- většina instrukcí je provedena v jednom cyklu,
- nízká spotřeba,
- možnost programování přímo na desce pomocí JTAG,
- dostatečná velikost vnitřních pamětí,
- zdarma vývojové nástroje + překladač jazyka C.

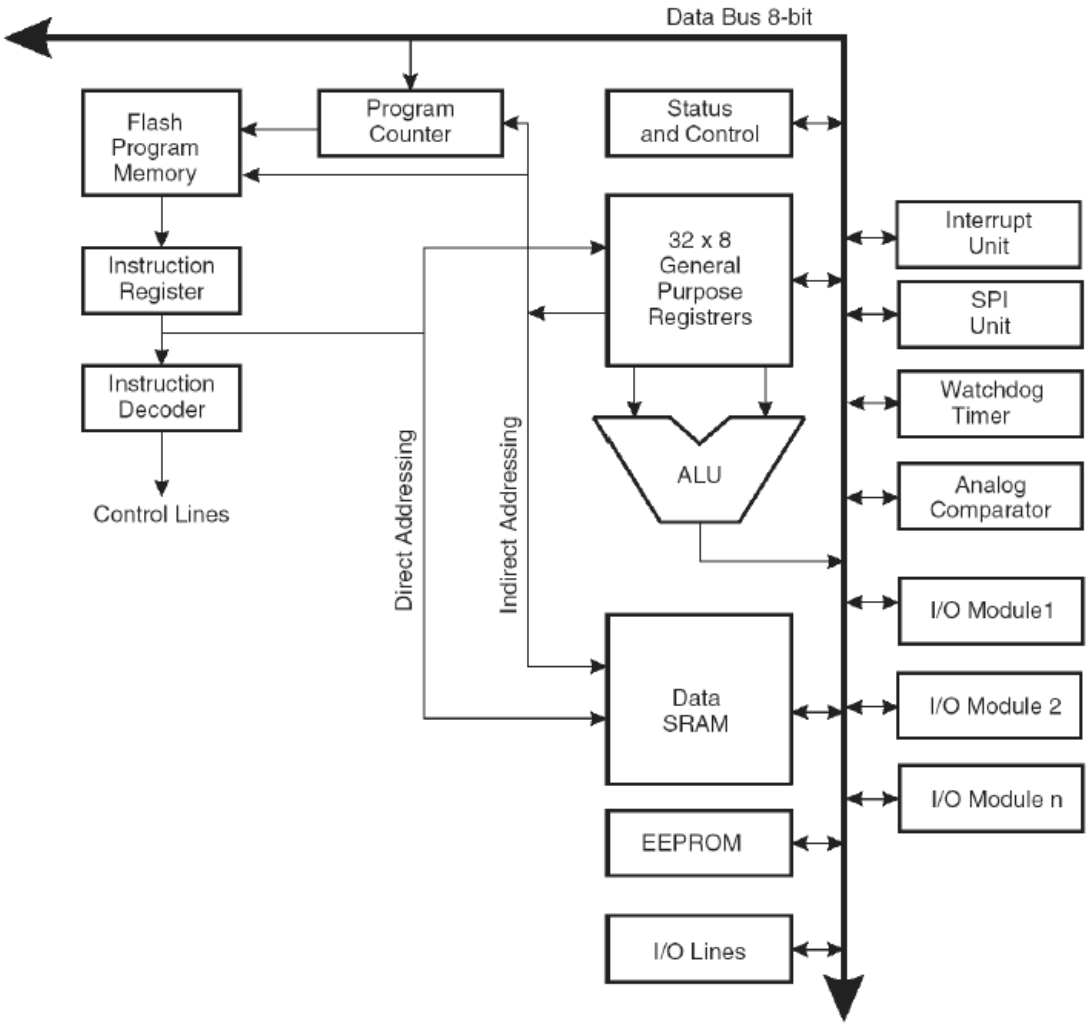
Vlastnosti použitého ATMEGA16 (více v [6]):

- napájecí napětí 4,5 – 5,5 V, při aktivitě odběr max. 1,1 mA
- 131 instrukcí,
- 16 kB Flash EEPROM pro uložení dat a programu, možnost přepisu až 10 000 cyklů,
- 512 B EEPROM pro uložení dat, možnost přepisu až 100 000 cyklů,
- 1 kB SRAM pro uložení dat,
- JTAG rozhraní pro programování mikrokontroléru nebo Flash, EEPROM, konfiguračních pojistek(ochrana proti kopírování programu),
- 2 8-bitové čítače/časovače se samostatnými předděličkami,
- 16-bitový čítač/časovač se samostatnou předděličkou,
- 8-kanálový 10 bitový AD převodník, 2 kanály s programovatelným ziskem až 200krát,
- programovatelný watchdog,

- analogový komparátor,
- sériové rozhraní I2C,
- programovatelný sériový USART,
- programovatelný reset po zapnutí a při detekci podpětí
- 6 úsporných režimů
- 2 8-bitové čítače/časovače se samostatnými předděličkami

1.1.1. RICS jádro ATMEGA16

Procesor ATMEGA16 je založen na RISCovém jádře Harvardské architektury, kde je oddělený paměťový prostor pro data a pro program blokové schéma je na obrázku 1-1.



Obr. 1-1: Blokové schéma jádra

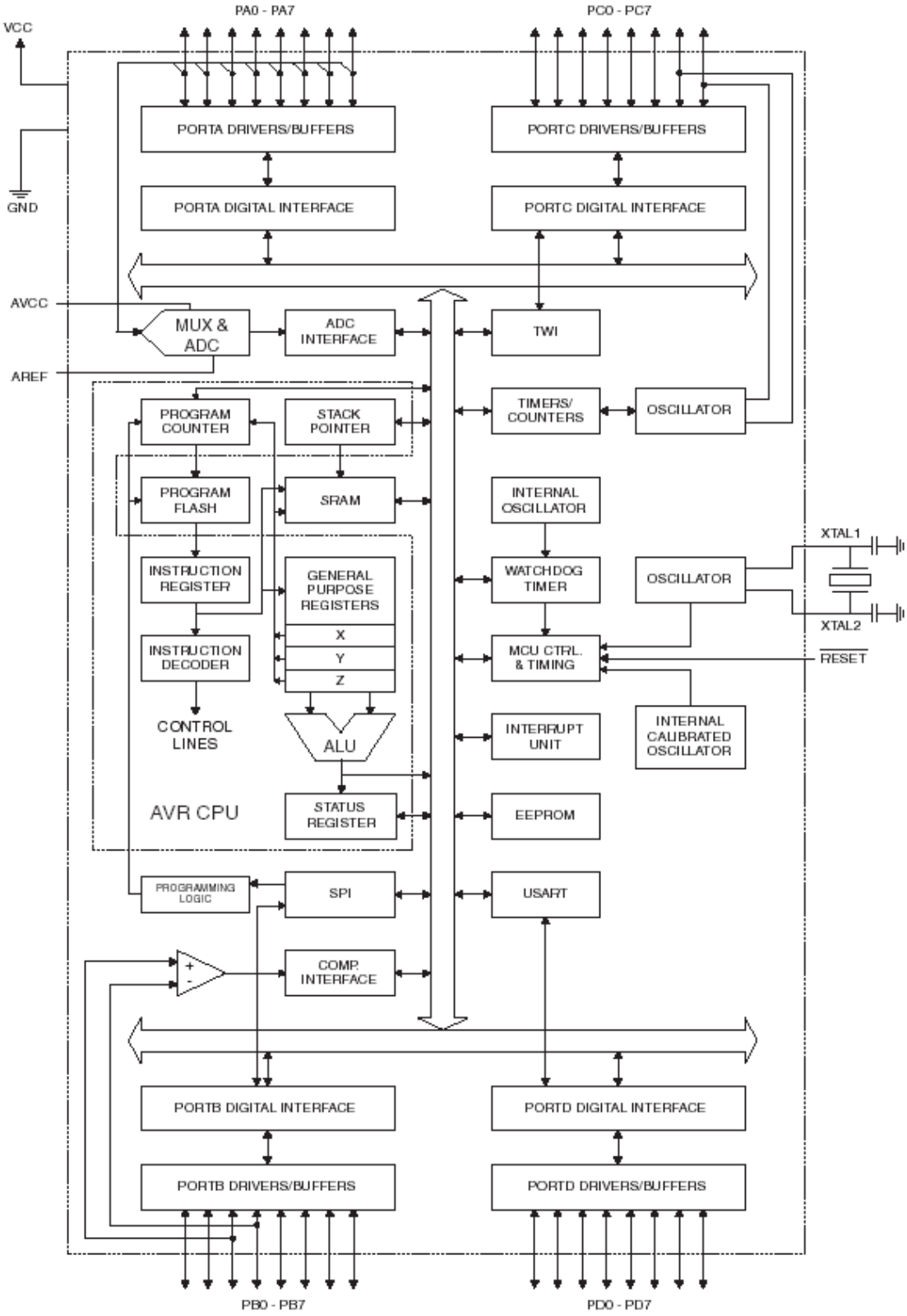
Jádro AVR se skládá ze 32 stejných 8bitových registrů, které mohou obsahovat jak data, tak adresy. Během jednoho hodinové cyklu se provede jedna ALU operace. Přitom vstupem jsou dva operandy uložené v souboru registrů, výstup operace je pak uložen nazpět do registru. ALU umožňuje aritmetické a logické operace mezi registry, nebo mezi registrem a konstantou.

Paměťový prostor dále obsahuje 64 adres I/O řídicích registrů pro nastavení čítače/časovače, A/D převodníku a dalších I/O funkcí. Program umístěný v programové paměti je prováděn s jednoduchým překrýváním instrukcí (pipeline). Zatímco jedna instrukce je prováděna, druhá se načítá z programové paměti.

Běh programu (registr PC) lze řídit podmíněnými a nepodmíněnými skoky, voláním a návratem z podprogramu a voláním obsluh přerušení.

Systém přerušení má vlastní řídicí registry umístěné v I/O prostoru a navíc bit ve stavovém registru pro zákaz/povolení všech přerušení. Všechny přerušení jsou umístěna v tabulce vektoru přerušení a jejich umístění určuje prioritu. Čím nižší má vektor přerušení adresu, tím větší má prioritu.

Hlídací obvod Watchdog je časovač, který může být spuštěn po resetu mikropočítače nebo někdy později. Slouží k hlídání programu, tak aby se nezacyklil nebo neudělal jinou chybu při, které by přestal fungovat. Watchdog je časovač, který při přetečení čítaní vyvolá reset, proto do programu vkládáme nulovaní čítače, tak aby k němu nedošlo a tím náš program se nemusel resetovat.



Obr. 1-2. Kompletní blokové vnitřní struktury ATMEGA16

1.1.2. Programovatelné konfigurační pojistky

Kromě běžné Flash a EEPROM paměti obsahuje procesor ATmega ještě čtyři 8bitovéEEPROM buňky, tzv. pojistky (fuses), které se programují zvlášť. Tyto pojistky určují specifická nastavení mikrokontroléru a dále už není potřeba je měnit. Slouží např. pro nastavení zdroje hodinového signálu, znemožnění vypnutí watchdogu atd.

Nastavení ochranných bitů má znemožnit další čtení/zápis programové paměti. Tyto pojistky mají hodnotu 1, pokud jsou nenaprogramované. V tab. 3.1.2.1 je význam pojistkové byte Lock Bits.

Tabulka. 1-1: Pojistkový byte Lock Bits

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	-	**-**	**BLB12**	**BLB11**	**BLB02**	**BLB01**	**LB2**	**LB1**

Bity BLBxx se týkají zaváděcí oblasti a bity BLx aplikační oblasti. Např. kombinací LB2 = 1, LB1 = 0 se zabrání dalším Flash, EEPROM a ostatních pojistek přes SPI/JTAG. Pokud navíc LB2 = 0, nelze z oblastí ani číst.

Tabulka. 1-2: Pojistkový byte High Fuse

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	OCDE	**JTAGE**	**SPIE**	**CKOP**	**EESAV**	**BS1**	**BS0**	**BRST**

OCDE – zapne funkci On-chip debut pro ladění JTAG.

JTAGE – zapne rozhraní JTAG .

SPIE – zapne programovací rozhraní SPI .

CKOP – nastavuje režim oscilátoru – pokud je bit naprogramovaný, bude budič externího krystalového oscilátoru používat plný rozkmit napětí což je nutné pro frekvence vyšší jak 8 MHz nebo v zarušeném prostředí. Tento režim má ale větší spotřebu. Pokud je bit nenaprogramovaný, používá budič menší amplitudu napětí.

EESAV – pokud je bit naprogramovaný, zůstane zachován obsah EEPROM při programování Flash.

BS0, BS1 – nastavuje velikost zaváděcí oblasti.

BRST – nastavuje vektor resetu – pokud je bit naprogramovaný, spustí se po resetu zavaděč ze zaváděcí oblasti, jinak se spustí program z aplikační oblasti.

Tabulka. 1-3: Pojistkový byte Low Fuse

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	BODL	**BODE**	**SUT1**	**SUT0**	**CKS3**	**CKS2**	**CKS1**	**CKS0**

BODL – určuje úroveň, při jakém poklesu napájecího napětí vyvolá detektor podpětí reset. Pro hodnotu 1 je to 2,7 V a pro 0 je to 4,0V

BODE – zapíná detektor podpětí, tzv. Brown out detector

SUT0, SUT1 – nastavuje přídavné časové zpoždění rozběhu procesoru po zapnutí napájecího.

CKS0-CKS3 – slouží pro volbu zdroje hodinového signálu (X – externí krystal, LX Xtal – krystal o nízké frekvenci, RC – interní/externí RC oscilátor), viz. Tab. 1-5

Tabulka 1-4: Hodnoty přídavného zpoždění náběhu po zapnutí napájení

CKS:	**X, CKS0=0**	**X, CKS0=1**	**LX Xtal**	**Int. RC**	**Ext. RC**
SUTx	**Doba[ms]**	**Doba[ms]**	**Doba[ms]**	**Doba[ms]**	**Doba[ms]**
0b00	4,1	65	4,1	-	-
0b01	65	-	65	4,1	4,1
0b10	-	4,1	65	65	65
0b11	4,1	65	-	-	4,1

Tabulka 1-5: Hodnoty přídavného zpoždění náběhu po zapnutí napájení

CKSx	Popis atributy	
0b111x[1]	Externí krystal 3-8 MHz; pro CK0PT=0	1-16MHz
0b110x[1]	Externí krystal 0,9 - 3 MHz; pro CK0PT=0	1-16MHz
0b101x[1]	Externí krystal 0,4-0,9 MHz; pro CK0PT=0	1-16MHz
0b1001	Externí nf krystal, typ 32kHz	
0b0100	Externí RC oscilátor 8-12 MHz	
0b0111	Externí RC oscilátor 3-8 MHz	
0b0110	Externí RC oscilátor 0,9 – 3 MHz	
0b0101	Externí RC oscilátor 0,1 – 0,9 Mhz	
0b0100	Kalibrovaný interní RC oscilátor 8 MHz	
0b0011	Kalibrovaný interní RC oscilátor 4 MHz	
0b0010	Kalibrovaný interní RC oscilátor 2 MHz	
0b0001	Kalibrovaný interní RC oscilátor 1 MHz	
0b000	Externí zdroj hodinového signálu	

[1]bit CKS0 slouží spolu se SUT0, SUT1 pro výběr přídavné doby náběhu

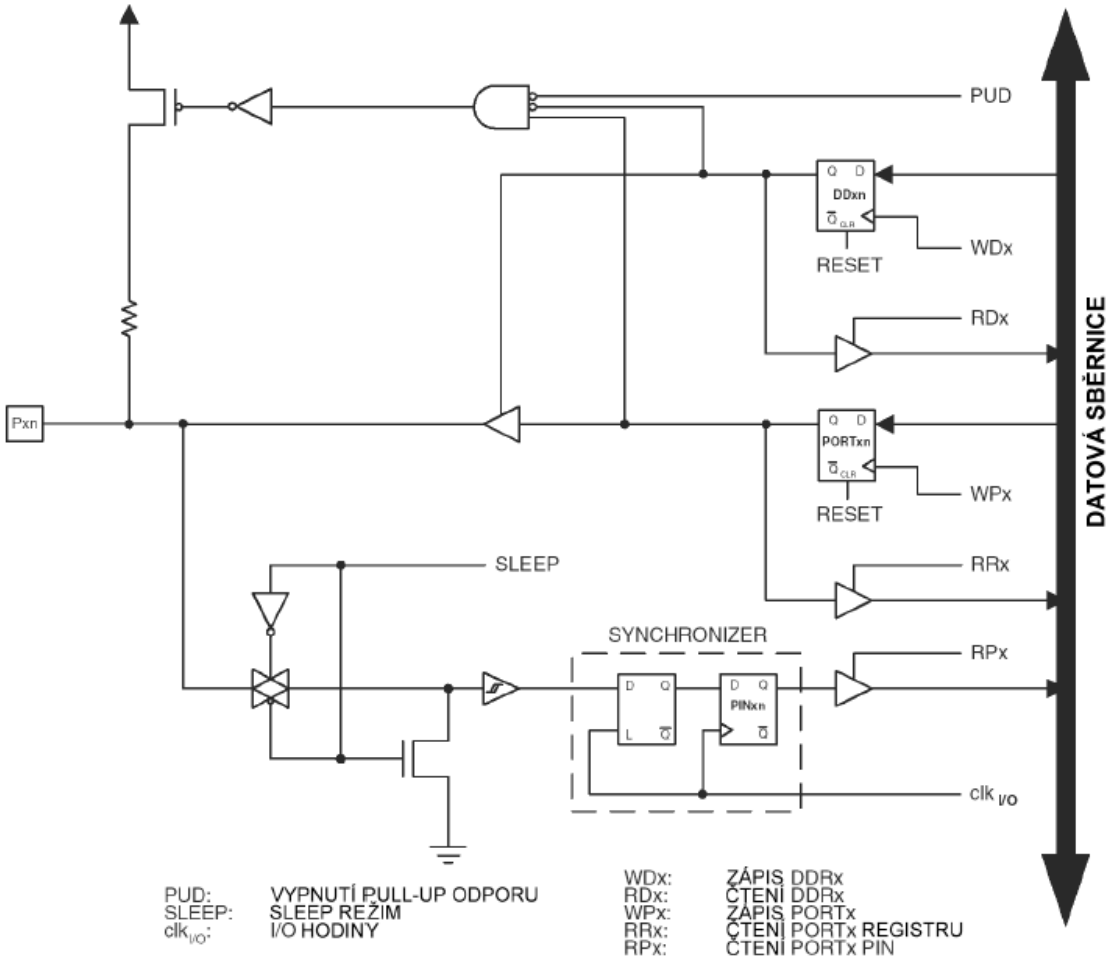
1.1.3. I/O porty

Mikrokontrolér ATMEGA16 je vybaven čtyřmi 8bitovými obousměrnými branami, které se označují PORTA, PORTB, PORTC, PORTD. Většina těchto bran(portů) má další alternativní význam, který se aktivuje pomocí řídicího registru dané periférie. Na obrázku 1-3. je řídicí logika I/O portu. Každý port má tři 8bitové řídicí registry.

PORTx – slouží k nastavení hodnoty na I/O na portu případně pinu.

DDRx – slouží pro konfiguraci portu buď jako vstup (hodnota 0) nebo výstup (hodnota 1).

PINx – slouží ke čtení aktuální hodnoty na daném pinu, synchronizuje se s hodinových signálem.



obr. 1-3. Řídicí logika I/O portu.

Všechny možné konfigurace portu jsou popsány v tab. 1-1. Pull-up rezistor z Vcc je řízen registrem **SFIOR**, bitem **PUD**. Čtením registru PORTx nám vrací předchozí zapsanou hodnotu, nikoliv skutečný stav pinu a tak ke čtení požíváme příkaz PINx. Při přechodu z režimu vstup do režimu výstup s danou hodnotou je potřeba nejdříve zapsat **PORTX** a pak **DDRx**. Po resetu mikrokontroléru jsou všechny porty nastavené jako vstup a pull-up rezistory vypnuté.

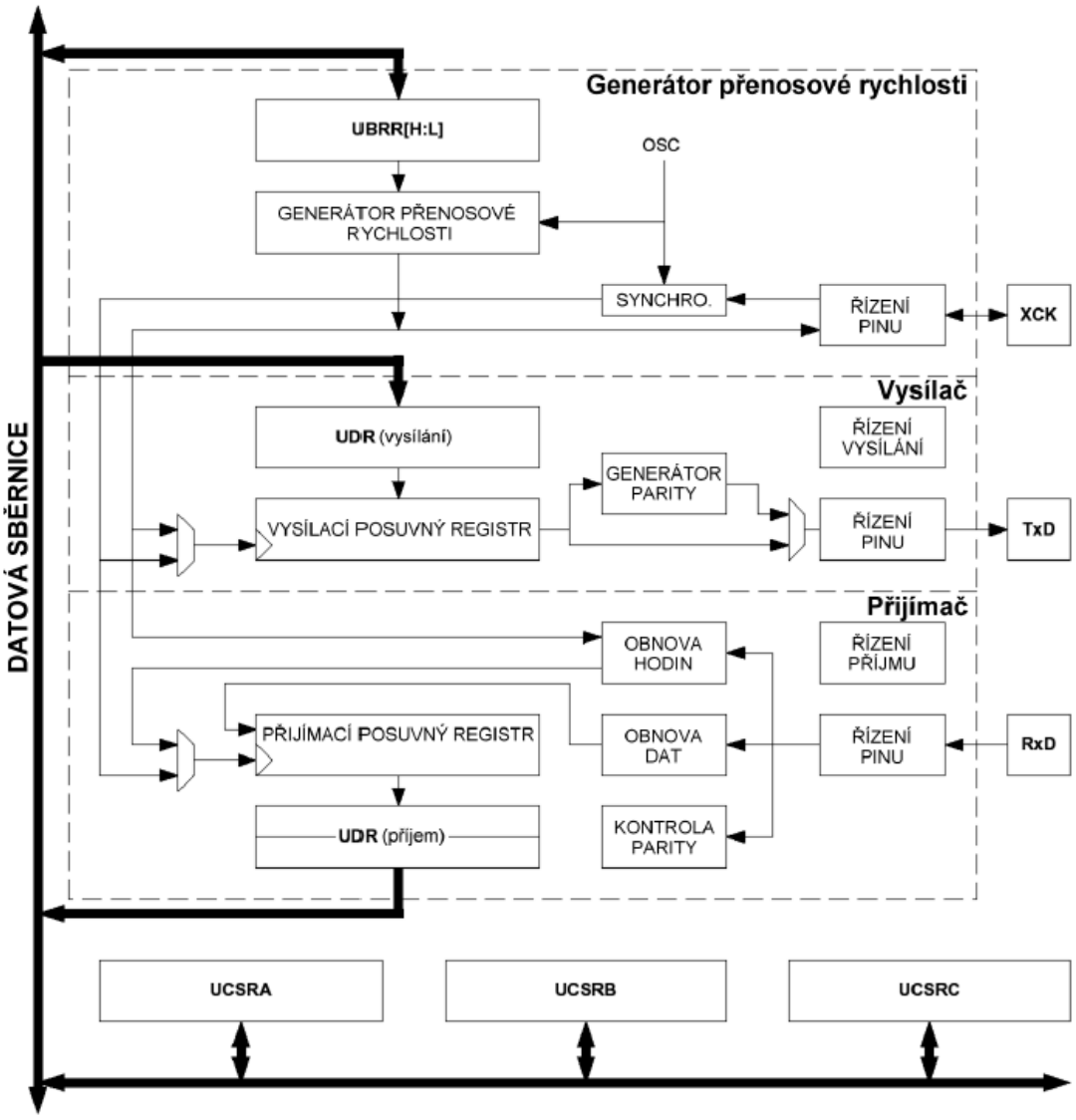
Tab.1-6. Možné konfigurace I/O portů

DDRx	PORTx	PUD	I/O	pull-up	popis
0	0	x	vstup	Ne	velká impedance 3. stav
0	1	0	vstup	Ano	připojen zatěžovací odpor, zdroj proudu
0	1	1	vstup	Ne	velká impedance, 3. stav
1	0	x	výstup	Ne	výstup v log. 0, otevřený kolektor
1	1	x	výstup	Ne	výstup v log. 1, otevřený kolektor

1.1.4. USART

Mikrokontrolér ATMEGA16 má jeden programovatelný **USART** (Universal Syncronous/Asynchronous Receiver and Transmiter) pro sériovou komunikaci. Základní úlohou jednotky USART je vyslat datové slovo převzaté od CPU a samostatně je sériově vyslat na linku TxD, popřípadě načíst bajt z linky RxD. Blokové schéma USARTu je na obr. 1-4. Základní vlastnosti USARTu:

- plně duplexní sériový kanál,
- vlastní generátor přenosové rychlosti
- možnost až 9bitové komunikace,
- detekce falešného start bitu, chybného znaku,
- filtrace šumu,
- 3 samostatné vektory přerušení (vysílání dokončeno, vysílací reg. prázdný, příjem kompletní)



Obr. 1-4. Blokové schéma USART

USART se skládá ze 3. základních celků: generátor rychlosti, vysílač a přijímač. Pro nastavení přenosové rychlosti slouží 16bitový registr UBRR. Konkrétní rychlost podle hodnoty UBRR registru vypočteme ze vztahu 1-1. a potřebnou hodnotu UBRR pro požadovanou rychlost ze vztahu 1-2.

Vztah 1-1.

$$\mathbf{BAUD = f_{osc}/[16 \bullet (UBRR+1)]}$$

kde:

$\mathbf{f_{osc}}$ je hodinový kmitočet procesoru

UBRR je hodnota UBRR registru

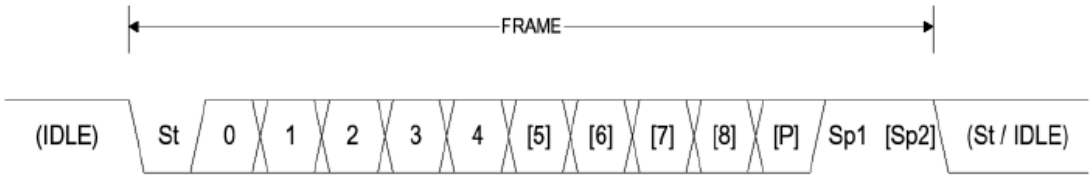
BAUD je bitová rychlost v baudech

Vztah 1-2.

$$\mathbf{UBRR = f_{osc}/(16 \cdot BAUD) - 1}$$

Datový registr **UDR** slouží k vysílaní a přijímáni bajtů a je fyzicky rozdělen do dvou registrů, které se adresují se stejnou adresou. Liší se jen způsobem přístupu pro čtení/zápis. Při zápisu se uplatňuje vysílací registr (**Tx**) a při čtení přijímací registr (**Rx**). V závislosti na zápisu/čtení se tedy uplatňují dva zcela odlišné registry.

Vysílač obsahuje posuvný registr do nejž se zápisem **UDR** vloží vysílaná data, ty jsou pak automatický zpracována. Po odeslaní všech bitů z posuvného registru je vyvoláno přerušení oznamující připravenost **UDR** zpracovat další data. Po výpočtu parit řídící logika v rytmu generátoru bitové rychlosti vysílá jednotlivé bity doplněné o start, paritu a stop bit na pin **TxD**. Při dokončení vysílaní je vyvoláno další přerušení(pokud je povoleno). Rámec přenosu je na obr. 1-5.



Obr. 1-5. Rámec sériové přenosu.

Přijímač vzorkuje každý bit 16 krát, tedy 16násobkem kmitočtu nastavené přenosové rychlosti. Vlastní vyhodnocení vzorků k vyhodnocení úrovně se u všech bitů provádí vždy 8., 9. a 10. vzorkem. Po odstranění šumu se bity střádají v posuvném registru. Z přijatých dat se spočítá a zkontroluje parita. Po dokončení přijmu slova je vyvolano přerušení a přijatá data jsou uložena v **UDR**.

Další nastavení USARTu a čtení stavových informaci se provádí pomocí tří 8bitové regitry **UCSRA**, **UCSRB**, **UCSRC** viz tab. 1-2, 1-3 a 1-5.

Tabulka 1-2. Konfigurační stavový registr UCSRA

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	RXC	**TXC**	**UDRE**	**FE**	**DOR**	**UPE**	**U2X**	**MPCM**

- bit 7 – **RXC**: 1 = v UDR je přijatý bajt
- bit 6 – **TXC**: 1 = výsílaní bajtu je hotovo a v UDR není žádny nový bajt
- bit 5 – **UDRE**: 1 = vysílaný bajt z UDR je převzat do Tx registru
- bit 4 – **FE**: 1 = signalizuje chybu rámce v přijatém slovu
- bit 3 – **DOR**: 1 = signalizuje přetečení dat
- bit 2 – **UPE**: 1 = chyba parity přijmutého slova v registru UDR
- bit 1 – **U2X**: nastavením do 1 se zvýší bitová rychlost UARTu na dvojnásobek
- bit 0 – **MPCM**: nastavením do 1 se zapíná podpora multiprocesorové komunikace

Tabulka 1-3. Konfigurační stavový registr UCSRB

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	RXCIE	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**

- bit 7 – **RXCIE**: 1 = povolení přerušení při dokončení RX (RXC = 1)
- bit 6 – **TXCIE**: 1 = povolení přerušení při dokončení TX (TXC = 1)
- bit 5 – **UDRIE**: 1 = povolení přerušení při dokončení vyprázdnění **UDR** (**UDRE** = 1)
- bit 4 – **RXEN**: 1 = zapíná blok přijímače
- bit 3 – **TXEN**: 1 = zapíná blok vysílače
- bit 2 – **UCSZ2**: slouží pro nastavení počtu datových bitů podle tab. 1-4
- bit 1 – **RXB8**: datový bit 8 (příjem) pro 9-ti bitové slovo
- bit 0 – **TXB8**: datový bit 8 (zápis) pro 9-ti bitové slovo

Tabulka 1-4. Konfigurační stavový registr UCSRB

UCSZ2	**UCSZ1**	**UCSZ0**	**Počet datových bitů**
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

Tabulka 1-5. Konfigurační stavový registr UCSRB

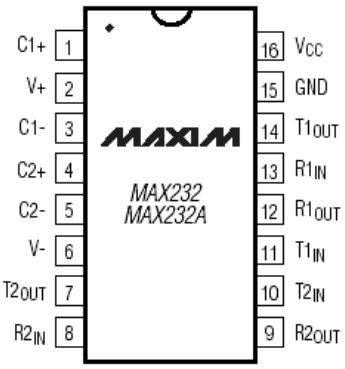
Bit	6	5	4	3	2	1	0
	UMSEL	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**

- bit 6 – **UMSEL**: 1 = synchronní režim, 0 = asynchronní režim.
- bit 5,4 – **UPM1**: volba parity: 0b00 žádná, 0b10 sudá, 0b11 lichá.
- bit 3 – **USBS**: nastavuje počet stop bitů, 0 = jeden, 1 = dva.
- bit 2 – **UCSZ1**: slouží pro nastavení počtu datových bitů podle tab. 1-4.
- bit 1 – **UCSZ0**: slouží pro nastavení počtu datových bitů podle tab. 1-4.
- bit 0 – **UCPOL**: nastavení polarity hodin pro synchronní přenos,
  0 = příjem na sestupnou hranu, 1 = příjem na vzestupnou hranu.

1.2. RS232

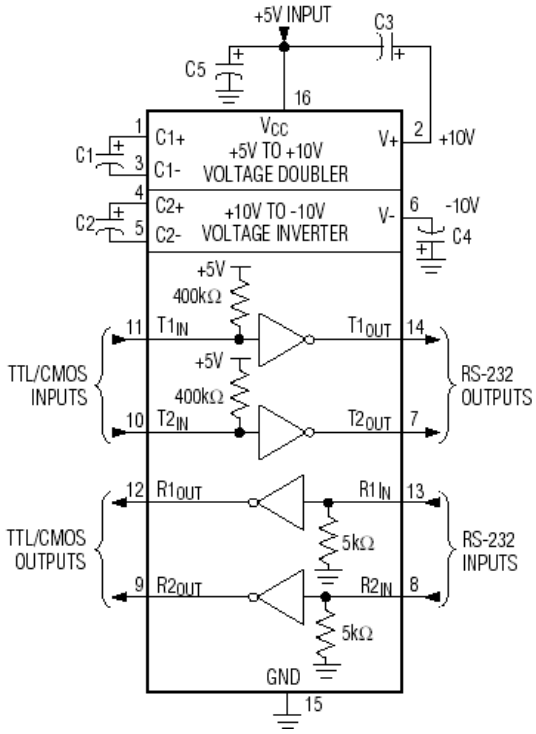
1.2.1. Převodník úrovní RS232/TTL

Pomoci RS232 se propojuje počítač s dalšími periferiemi např. modem, PC, měřící přístroj, myš atd. Pro lepší odolnost proti rušení a zvýšení dosahu využívá počítač logiky -15/+15 V. Mikrokontrolér **ATMEGA16** používá standardních 0/5V. Proto je potřeba použít převodník napěťových úrovní Max232. Pro naše zařízení jsem použil obvod Max 232 od firmy Maxim v provedení **SMD**. Blokové schéma a zapojení vývodů je na obr. 1.2-1. Hlavní výhodou je, že tento obvod potřebuje ke své funkci napětí pouze 5V. Obsahuje totiž zdvojovače a invertory na principu nábojové pumpy, která vytvoří potřebné napětí buzení pro **RS232**. Obvod vyžaduje pouze minimum součástek. 4 kondenzátory (tantalové) a jeden blokovací kondenzátor napájecího napětí. Detailní popis obvodu MAX232 viz [11].



DIP/SO

CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

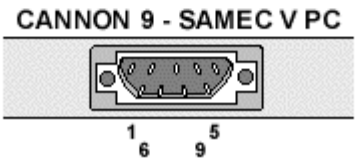


Obr. 1.2-1: Blokové schéma a zapojení vývodů MAX232

1.2.2. Zapojení RS232

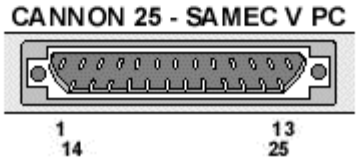
V dnešní době se v počítačích setkáme už pouze s jedním nebo žádným konektorem RS232 a to pouze cannon 9, který je zobrazen na obrázku 1.2-2. Verze cannon 25 je na obrázku 1.2-3.

Cannon 9			
PIN	**NÁZEV**	**SMĚR**	**POPIS**
1	CD	<--	Carrier Detect
2	RXD	<--	Receive Data
3	TXD	-->	Transmit Data
4	DTR	-->	Data Terminal Ready
5	GND	---	System Ground
6	DSR	<--	Data Set Ready
7	RTS	-->	Request to Send
8	CTS	<--	Clear to Send
9	RI	<--	Ring Indicator



Obr. 1-2-2: Zapojení konektoru cannon 9

Cannon 25			
PIN	NÁZEV	SMĚR	POPIS
1	SHIELD	---	Shield Ground
2	TXD	-->	Transmit Data
3	RXD	<--	Receive Data
4	RTS	-->	Request to Send
5	CTS	<--	Clear to Send
6	DSR	<--	Data Set Ready
7	GND	---	System Ground
8	CD	<--	Carrier Detect
9-19	N/C	–	-
20	DTR	-->	Data Terminal Ready
21	N/C	-	-
22	RI	<--	Ring Indicator
23-25	N/C	-	-

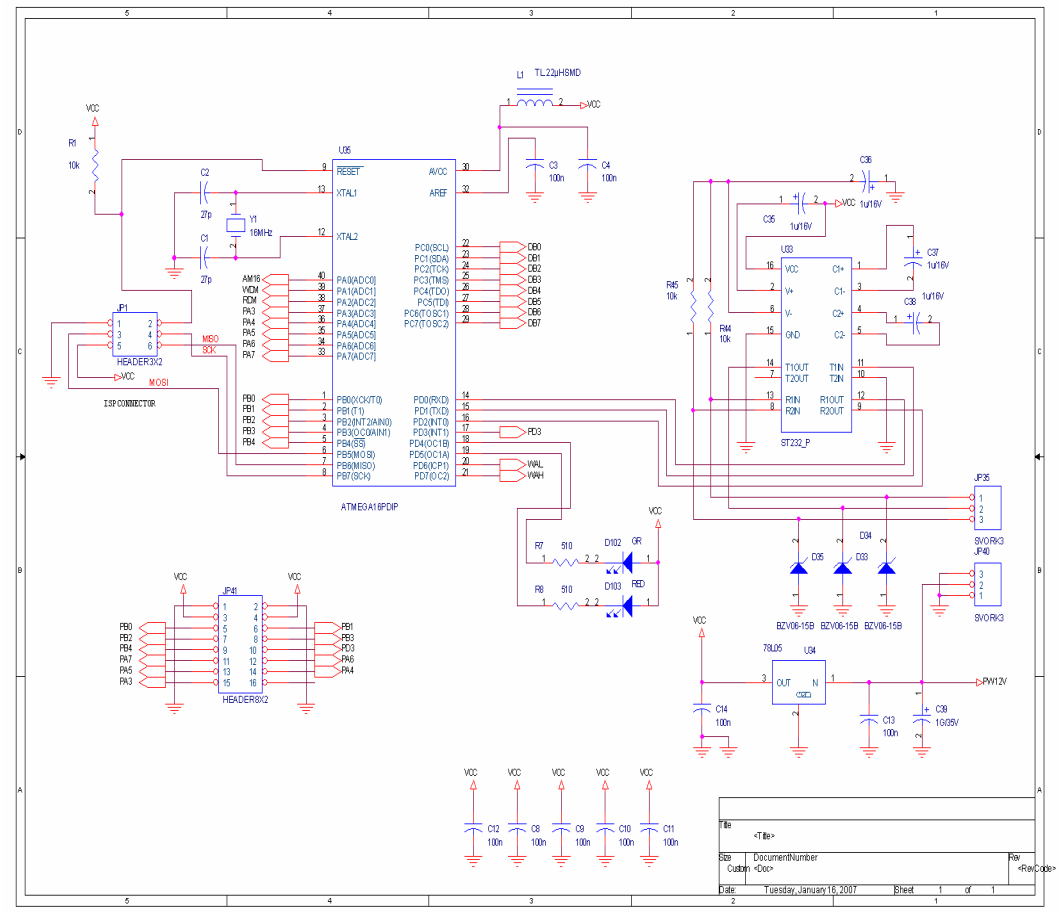


Obr. 1.2-3: Zapojení cannon 25

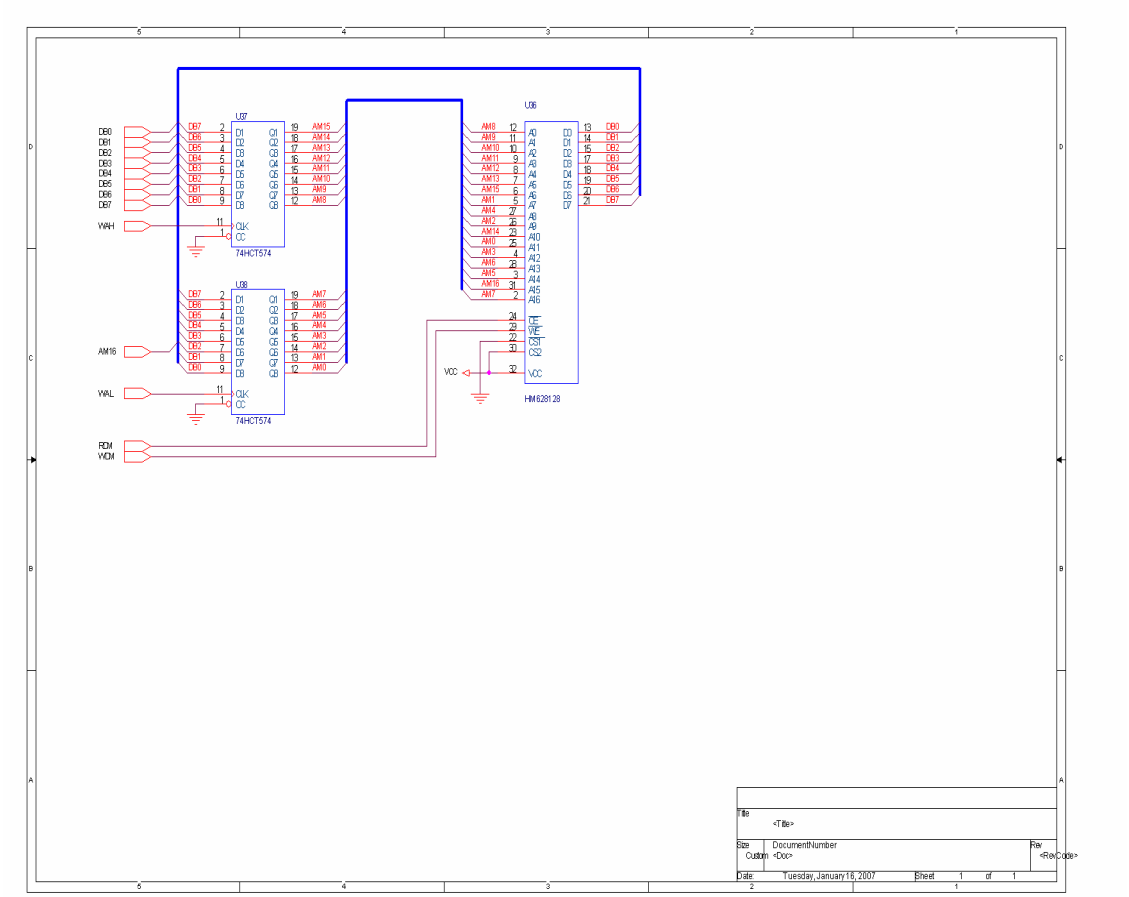
2. Návrh hardware

2.1. Schéma zapojení

Schéma zapojení a návrh plošného spoje jsme udělali v programu OrCAD. Všechny vývody součástek jsme propojili podle jednotlivých katalogů k součástkám. Schéma zapojení je na obrázku 2.1-1 a 2.1-2. Plošný spoj jsme nechali vyrobit v laboratořích TUL pomocí fotocesty. Obvodové zapojení je realizováno na jednostrané desce plošného spoje. Všechny **SMD** součástky jsou osazeny ze spodní strany a všechny **THD** součástky jsou osazeny z vrchu. Osazení jsem provedl ručně mikropájkou.



Obr. 2.1-1: Schéma zapojení analyzátoru I.



Obr. 2.1-2: Schéma zapojení analyzátoru II.

2.2. *Komunikační rozhraní*

Analyzátor komunikuje s PC po lince **RS 232**. Pro komunikaci jsou využity 4 vstupy/výstupy **Rx**,**Tx**,**Gnd**,**DTR**. Signál **DTR** je využíván jako zdroj spouštění měření a je připojen přes převodník MAX232 na **INT0** mikrokontroléru.

3. Software pro mikrokontrolér

3.1. *Vývojové nástroje*

Dříve bylo běžné programování jednočipových mikropočítačů v jazyce symbolických adres (assembleru). Výrobci dodávali ke svým produktům vlastní překladače, debugery a simulátory.

Hlavní výhodou programování v asebleru je, že výsledný kód přímo píše programátor a tak při dobré znalosti hardwaru ho může velmi efektivně optimalizovat na rychlost a velikost. Toto ma především význam u levných jednočipů s malou pamětí

a nižším výkonem, kam by se výsledný kod z překladačů s výšších jazyků vůbec nevešel.

Nevýhodou je pak to, že kód v asembleru je vázán na konkrétní instrukční soubor daného procesoru nebo rodiny a nelze ho pak jednoduše použít na jiné platformě.

3.2. Jazyk C

S příchodem moderních rychlých jednočipů s dostatečnou pamětí se stále častěji používá překladačů výšší programovacích jazyků, zejména C. Vývoj jazyka C začal v Bellových laboratořích AT&T mezi léty 1969 a 1973. Vyvinul ho Ken Thompson a Denis Ritchie pro OS UNIX. Stručná charakteristika:

- univerzální programovací jazyk nízké úrovně,
- obsahuje velký soubor operandů a moderní datové struktury,
- je strukturováný a má úspornou syntaxi,
- není specializovaný na jednu oblast používání,
- jeho kód se efektivitou skoro vyrovná asembleru
- je nezávislý na použité platformě

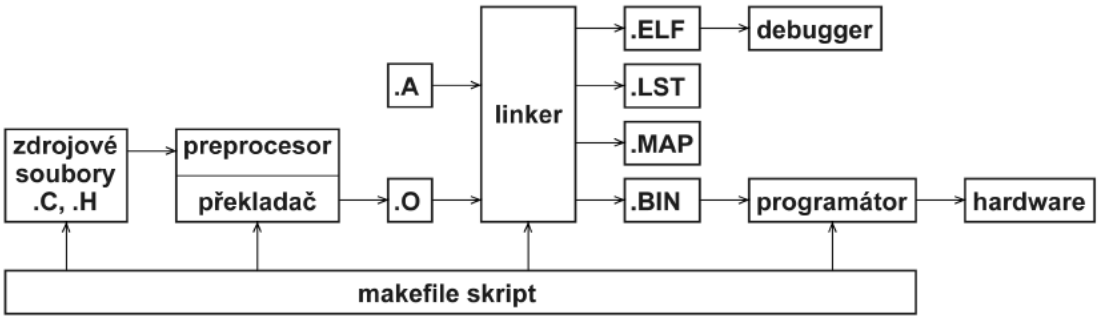
Pro platformu AVR je celá řada překladačů CodeVisionAVR C Compiler, Imagecraft, ICCAVR a plno dalších open-source překladačů. Já jsem požil balík WinAVR.

3.3. WinAVR

Balík WinAVR obsahuje vše potřebné pro přeložení kódu a naprogramování jednočipu.

- AVR-libC – knihovna standardních C funkcí pro AVR
- GCC – překladač ANSI C a C++
- Binutils – nástroj pro tvorbu a praci s binarními soubory
- AVRdude – programátor jednočipů s širokou podporou hardware
- Simul AVR – podpora simulace pro GDB

Překlad probíhá podle schématu na obr. 3.3-1.



Obr. 3.3-1: Proces překladu zdrojových kódů

3.4. Obsluha přerušení

Nejdříve je třeba definovat funkci pro obsluhu přerušení a do tabulky vektorů přerušení na příslušnou pozici zapsat pointer na onu funkci. To lze provést např. pomocí makra SIGNAL(jméno signálu) místo klasické hlavičky funkce, které je definované v tomtéž souboru INTERRUPT.H (dříve SIGNAL.H). Jméno signálu přerušení je definováno v tomtéž souboru. V hlavním programu je pak třeba globálně povolit přerušení makrem sei() a také v příslušném řídicím registru periférie povolit konkrétní přerušení. Pokud funkce obsluhy přerušení pracuje s nějakou globální proměnou, je nutné aby tato proměnná byla označena atributem volatile. Ukázka definice obslužné funkce je v následujícím příkladu.

```
#include <avr/interrupt.h>
#include<avr/io.h>
volatile char a;
SIGNAL(SIG_UART_RECV)
{
        if (a==0)
                return;
        a=UDR;
}
int main(void)
UCSRB=(1<<RXEN) | (1<<TXEN) | (1<<RXCIE);
sei();
return 0;
}
```

4. Testovací aplikace

4.1. Vývojové prostředí

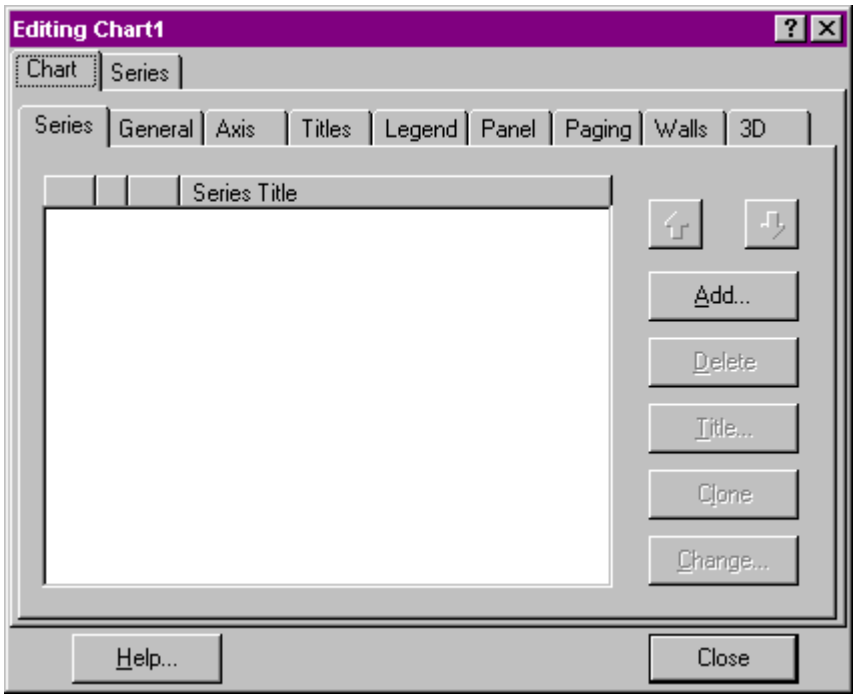
Aplikaci pro komunikaci s mikrokontrolérem jsem naprogramoval ve vývojovém prostředí Borland Deplhi VI. Delphi vychází z Turbo Pascalu, velmi kvalitního Pascalovského kompilátoru. Delphi vzniklo hlavně pro zjednodušení vývoje grafických aplikací, kde se nemusí namáhavě programovat všechny prvky programu. Jednoduše přesuneme např. tlačítko na formulář a už sním můžeme pracovat. Každá komponenta nabízí řadu vlastností, na jejichž základě je možno měnit chování komponenty a událostí, na které může komponenta reagovat.

Všechny komponenty jsou v době designu viditelné, ale v době chodu programu můžou být skryté.

4.1.1. Komponenta Chart

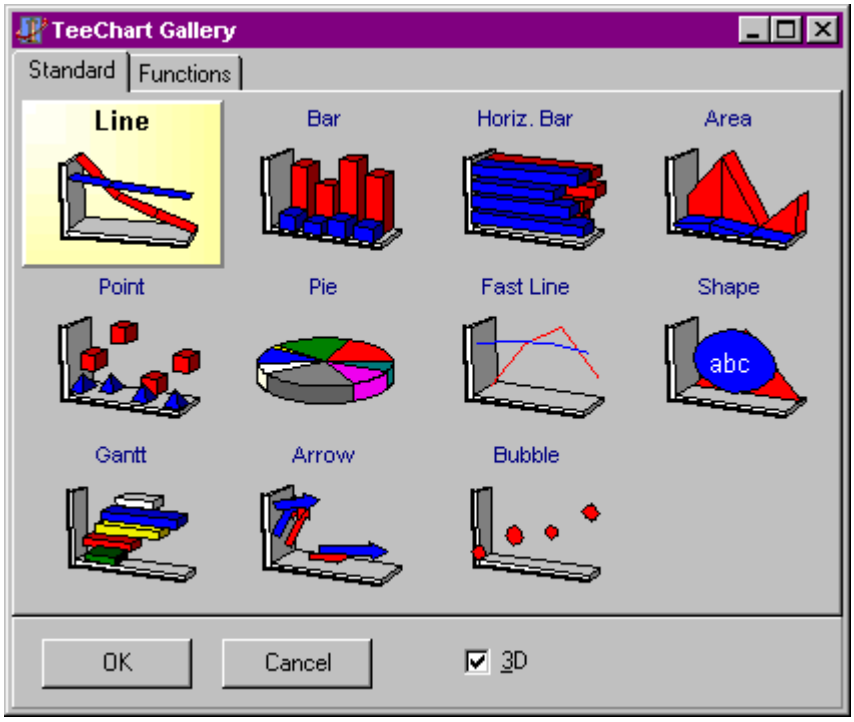
Komponentu TChart jsem použil pro zobrazení analogového signálu získaného z analyzátoru. Tato komponenta je součástí Delphi a nabízí velmi elegantně vypadající grafy.

Po umístění komponenty na formulář, stisknutím právého tlačítka myši se nám zobrazí editor grafů Chart Editor viz. obrázek 4-1.



Obr. 4-1: Editor grafu

Poté klikneme na tlačítko Add, abychom mohli přidat další řadu(series) v grafu. Otevře se další okno v němž jsme zvolili klasický řádkový graf (line). Viz obr. 4-2.



Obr. 4-2: Galerie grafů

Zaškrtneme tlačítko 3D a klepneme na OK. Nyní jsme vytvořili první datovou řadu. Základní záložky pro nastavení:

- Series: seznam existujících datových řad. V našem případě obsahuje jedinou, právě vytvořenou sérii, která se nazývá Series1. Pomocí tlačítek je možné přejmenovat sérii, smazat ji, klonovat (vytvořit identickou kopii), případně změnit její druh.
- General: obecná nastavení, která zahrnují možnosti tisku a exportu do obrázku, a dále nastavení týkající se zoomu a scrollování.
- Axis: nastavení týkající se os.
- Titles: nastavení titulků a popisků, včetně barvy nebo vzoru pozadí apod.
- Legend: nastavení legendy a pomocných popisků, včetně fontu, barvy, stínování apod.
- Panel: nastavení panelu, na němž je zobrazen graf na formuláři (okraje, plasticita, barva apod.)
- Paging: stránkování
- Walls: nastavování vzhledu levé, spodní a zadní „stěny“ grafu
- 3D: nastavování trojrozměrného vzhledu grafu

Tímto jsme vytvořili graf a nyní se musí naplnit daty. Jako příklad uvedu kousek zdrojového kódu programu, který plní graf při reálném čtení dat z analyzátoru. Zobrazuje posledních 100 přečtených hodnot.

```
if i < 101  then Chart1.Series[0].AddXY(i mod 100,cislo) else
        Chart1.Series[0].YValue[i mod 100]:=cislo;
```

4.1.2. Komponenta pro přístup k sériovému rozhraní

K sériovému rozhraní se dá přistupovat i bez pomocných komponent, přes API funkce. Je to však velmi složité a proto jsem použil freewarový balíček Varian Async32. Pracuje pod 32-bitovými Delphi a C++ Buildrem 4[18].

Umístíme komponentu na plochu a provedeme její nastavení. Nastavení pro analyzátor jsem uložil do INI souboru z kterého jsou později data načtena. Ukázka načtení nastavení komponenty.

```
var ini: TIniFile;
begin
 ini:=TIniFile.Create('AVRBoard.ini');

 VaComm1.PortNum   := TVaPortInt(ini.ReadInteger('komunikace','COM',0)+1);
 VaComm1.BaudRate := TVaBaudrate(ini.ReadInteger('komunikace','BAUD',7)+1);
 VaComm1.Databits  := TVaDataBits(ini.ReadInteger('komunikace','DATA',4));
 VaComm1.Parity    := TVaParity(ini.ReadInteger('komunikace','PARITY',0));
 VaComm1.StopBits  := TVaStopBits(ini.ReadInteger('komunikace','STOP',0));
```

Ukázka otevření a zavření portu.

```
VaComm1.Close; //zavře příslušný port (COM1)
VaComm1.PortNum:=2; //nastavení čísla nového portu COM2
VaComm1.Open; //otevření tohoto portu (COM2).
```

V Komponentě VaComm v Object Inspectoru v Properties je spoustu položek - k přípravě přenosu jsou potřebné tyto:

- Baudrate - přenosová rychlost - na kratší zdálenosti doporučuji rychlost 115 200 Baudů/s, na delší nižšší
- Databits - počet datových bitů - při přenosu znaků z ASCII tabulky ponechte implicitní nastavení (8 datových bitů)
- FlowControl - řízení toku dat - RTS/CTS a DTR/DSR je hardwarové, XOn/XOff softwarové a None žádné
- Parity - vkládání paritního (kontrolního) bitu - sudého, lichého, mezerového, značeného nebo žádného
- Stopbits - počet stopbitů - 1, 1,5 a 2

K nastavení řízení toku dat jsem vybral None; při použití softwarového řízení pomocí znaků XOn/XOff by mohlo dojít při přenosu datových souborů (všechny kromě čistě textových) k přerušení přenosu - kdyby soubor obsahoval znaky XOn/XOff, tak by je příjmač detekoval jako konec přenosu.

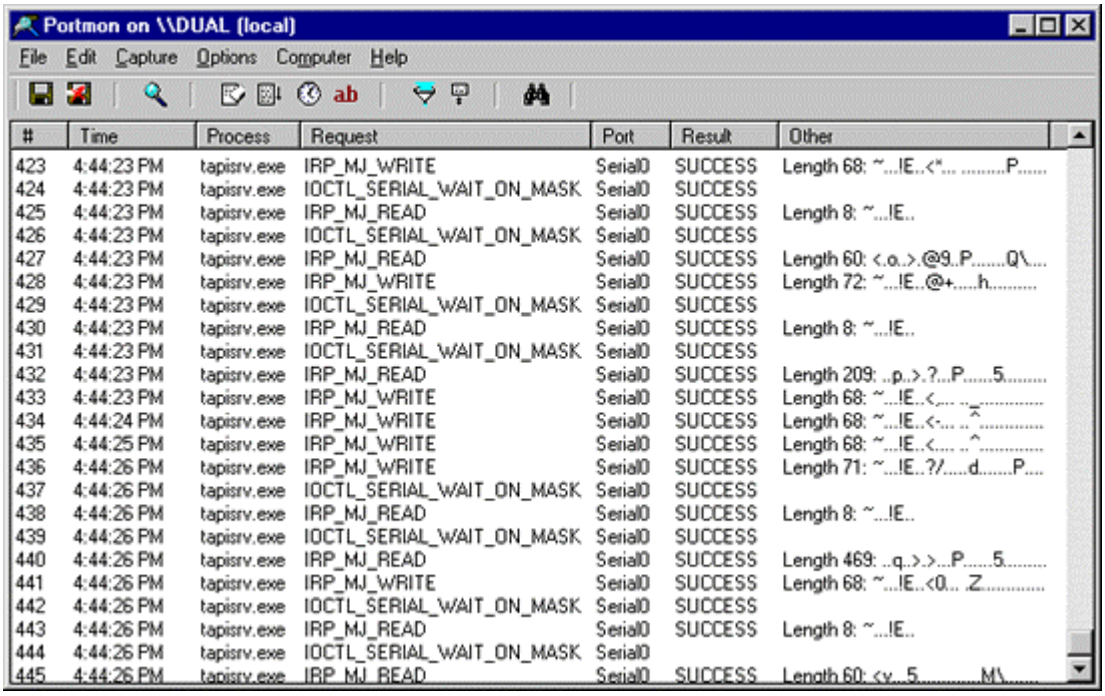
K vysílání dat používám funkci VaComm1.WriteText('text') viz příklad.
Na příkladu je odeslán testovací řetězec, zda je analyzátor připojen.

```
s:= 'r'+#10#11#12#13#14#15;
VaComm1.WriteText(s);
```

Z příkladu je zřejmé, že se data typu string (řetězec) zapisují funkcí VaComm.WriteText(s), kde s je řetězec znaků, které chceme zapsat a výsledek funkce je typu boolean (true = pravda nebo false = nepravda). Data se čtou až po vyvolání události OnRxChar, která indikuje příchozí znak, a to funkcí VaComm.ReadText, kde výsledkem jsou přečtená data typu string (řetězec).

4.2. *Pomocný software*

Při komunikaci s analyzátorem jsem veškerou komunikaci monitoroval a odlaďoval pomocí software PortMon od společnosti Sysinternals [19]. Tento nastroj je velmi užitečný při komunikaci pomocí sériového portu. Na obr. 4-3 je zobrazen PortMon při zachytávání dat.



Obr. 4- 3: PortMon

5. Závěr

Závěrem bych chtěl zhodnotit výsledky a průběh své práce. V první etapě jsem prostudoval specifikaci procesoru ATmega16 a vyzkoušel si jeho jednoduché zapojení. Na nepájivém poli jsem si otestoval zápis do paměti a vytvořil si jednoduché funkce programu pro komunikaci s PC, přerušení, čtení z AD převodníku a čtení/zápis do paměti.

Poté jsem navrhnul celkové zapojení a plošný spoj. Vše zkompletoval a odladil software mikrokontroléru a programu v PC.

Výsledkem je funkční analyzátor, který by se dal v budoucnu rozšířit o komunikaci s CF kartou, na kterou se dají uložit mnohem větší objemy dat.

Použitá literatura

[1] Specifikace mikrokontroléru ATMEGA - www.atmel.com

[2] Překladač WIN AVR, dokumentace k projektu

[3] MSDN, programováni s MFC, www.microsoft.cz

[4] datasheet MK681000 [CD-ROM], [Cit. 20.1.2007]

[5] datasheet 74HC574 [CD-ROM], [Cit. 20.1.2007]

[6] datasheet ATMEGA16 [CD-ROM],[Cit. 20.1.2007]

[7] C pro mikrokontroléry, Burkhard Mann, 2004

[8] Atmel AVR programování v jazyce C,Vladimír Váňa, 2003

[9] Atmel AVR popis procesorů a instrukční soubor, Vladimír Váňa, 2003

[10] Jan Voplakal, http://www.builder.cz/art/delphi/delphiser.html

[11] datasheet MAX232 [CD-ROM], [Cit. 20.1.2007]

[12] http://cs.wikipedia.org/wiki/C_(programovac%C3%AD_jazyk)

[13]Jazyky C a C++ podle normy ANSI / ISO, Dirk Louis, Petr Mejzlík, Miloslav Virius, 1999

[14] Jazyk C, Dalibor Kačmář, 2000

[15] RNDr. Klára Císařová , http://www.fm.tul.cz/~ksi/cz/mater/ksi_mat.html

[16]Pavel Kubal, http://www.builder.cz/art/delphi/dserial_trocha_historie.html

[17]Václav Kadlec, http://www.zive.cz/h/Programovani/Ar.asp?ARI=110213

[18]Varian Async, http://www.builder.cz/data/ASYNC32.ZIP

[19]PortMon,http://download.sysinternals.com/Files/PortMon.zip

Přílohy