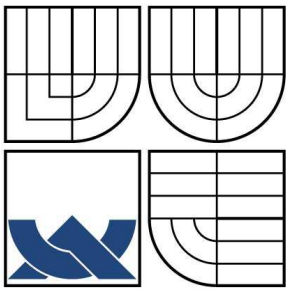
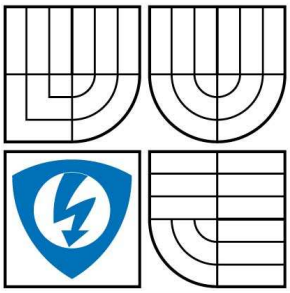


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

INTELIGENTNÍ SIMULÁTOR ODPOROVÝCH ČIDEL PRO ŽIVOTNOSTNÍ TESTY

SMART SIMULATOR OF RESISTIVE SENSORS FOR LIFE-CYCLE TESTS

BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

AUTOR PRÁCE **JAKUB POSPÍŠIL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADIM ČÍŽ,**
SUPERVISOR

BRNO 2008

ZDE VLOŽIT LIST ZADÁNÍ

(Kvůli správnému číslování)

ZDE VLOŽIT LIST LICENČNÍ SMLOUVY

(Kvůli správnému číslování)

ZDE VLOŽIT DRUHÝ LIST LICENČNÍ SMLOUVY

(Kvůli správnému číslování)

ABSTRAKT

Abstrakt práce v češtině

Práce je zkonstruována jako ovládací prvek při vyhodnocování životnostních testů připojených zařízení. Je dodána sada knihoven, pomocí kterých lze zařízení jednoduše ovládat. V zařízení je použit digitální potenciometr, jako simulátor odporových čidel. Zařízení komunikuje s PC. Jádrem celého zařízení je mikrokontroler ATmega324p.

KLÍČOVÁ SLOVA

Klíčová slova v češtině

Životnostní test, digitální potenciometr, simulátor odporových čidel, procesor ATmega324p

ABSTRACT

Abstract in English

The document is made as an operating item at evaluating of life tests for connected equipments. There is set of bookcases, thanks to them we can use the equipment simply. The digital potenciometer is used like a stimulator of resistance sensor in this equipment. The equipment is able to communicate with PC. The mikrocontroler ATmega324p is the most important part of the equipment.

KEYWORDS

Keywords in English

life-cycle test, digital potentiometer, simulator of resistive sensors, processor ATmega324p

POSPÍŠIL J. *Inteligentní simulátor odporových čidel pro životnostní testy.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 54 s., 7 s. příloh. Bakalářská práce. Vedoucí práce byl Ing. Radim Číž.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma "Inteligentní simulátor odporových čidel pro životnostní testy" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne …… ……..

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce Ing. Radimu Čížovi, Ing. Radomíru Svobodovi a panu Ing. Petrovi Kolesnikovi za příkladnou metodickou, pedagogickou a odbornou pomoc a za další cenné rady při zpracování mé bakalářské práce.

V Brně dne …… ……..

(podpis autora)

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

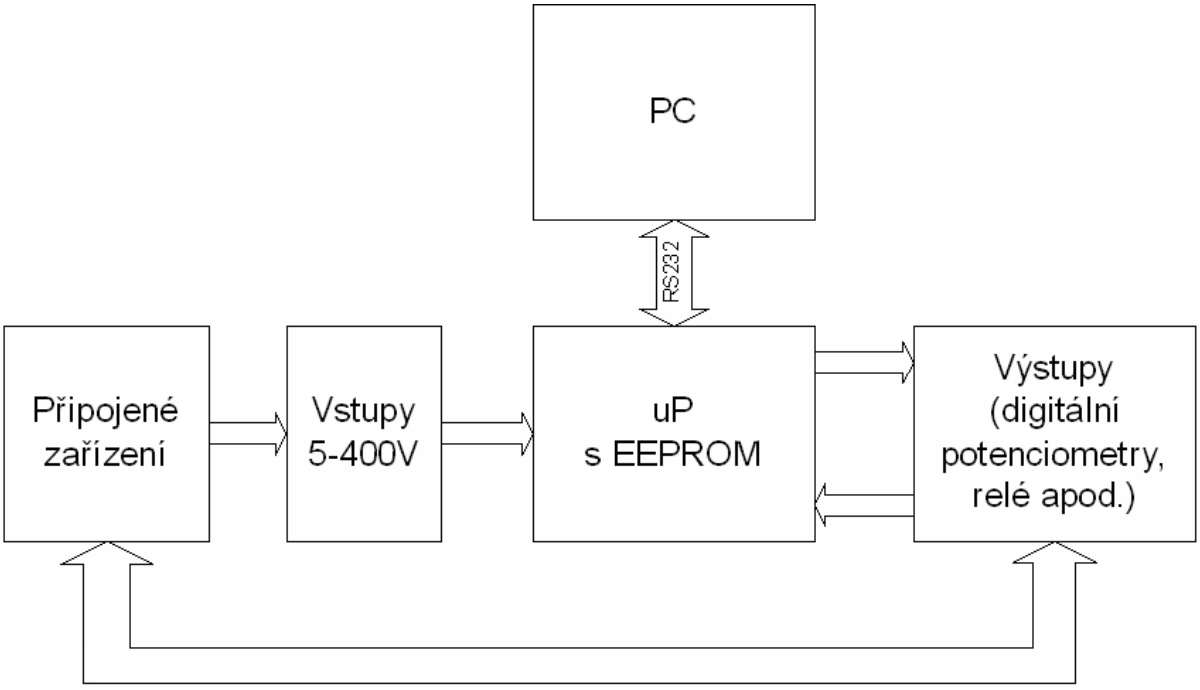
Cílem bakalářské práce je seznámení s rodinou mikrokontrolérů ATmega od firmy Atmel a jejich programování v jazyce C. V tomto případě konkrétně s typem ATmega324p.

Úkolem je sestrojit koncept zařízení, které bude sloužit jako ovládací prvek testeru, který bude provádět životnostní testy na připojených zařízeních. K testeru bude dodána sada knihoven, které budou sloužit k jeho ovládání. Uživatel potom bude moci napsat svůj vlastní program podle potřeby, kde tyto knihovny využije k vyvolání potřebných funkcí (zvýšení/snížení odporu digitálního potenciometru, vyhodnocení vstupního napětí, spínání sady relé) .Digitální potenciometr zde slouží jako simulátor odporových čidel, které by měly za úkol měřit neelektrické veličiny, jako je například teplota, vlhkost, tlak apod.

K testeru budou tedy připojena zařízení, která budeme chtít otestovat, z jejich výstupů půjdou signály o hodnotě stejnosměrného napětí až do 400V okamžité hodnoty.

Veškerá data bude možno přenášet do počítače pomocí rozhraní RS232, kde bude možno s výsledky dále pracovat.

Blokové schéma testeru je na obrázku 1.1.



Obr. 1.1: Blokové schéma testeru

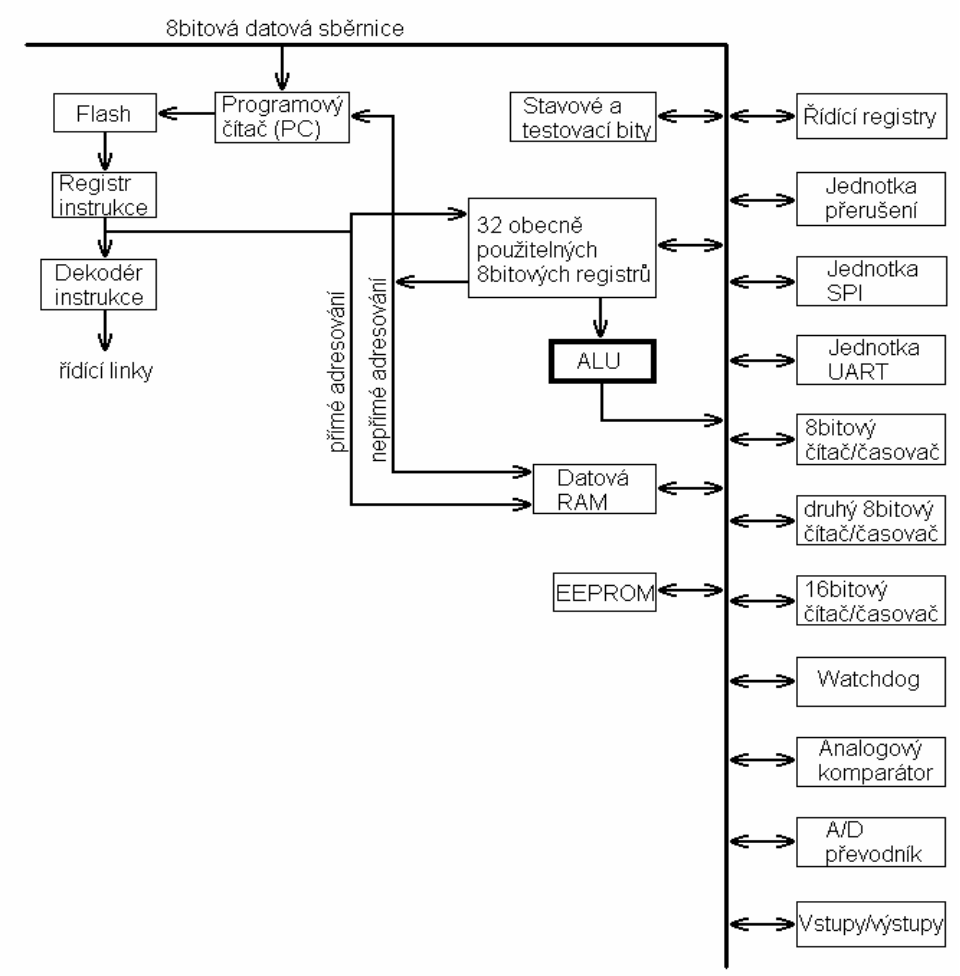
2 Mikroprocesor

2.1 Použitý mikroprocesor Atmel ATmega324p[1]

Volba procesoru Atmel AVR typu ATmega324p byla dohodnuta s vedoucím ve firmě Honeywell. Vzhledem k tomu, že bude využita velká část zabudovaných periferií, tak je volba tohoto procesoru optimální. Není tedy potřeba připojovat externí periferie, pouze digitální potenciometr, tímto bude ušetřena velká část portů, které lze v budoucnu využít.

Jedná se o nízkopříkonový 8 bitový mikroprocesor, stejně jako ostatní procesory AVR provádí výkonné instrukce v jediném hodinovém cyklu, takže dosahují výpočetního výkonu 1MIPS na 1MHz.

Zjednodušené blokové schéma jádra AVR procesoru (viz obr. 2.1).



Obr. 2.1: Zjednodušené schéma jádra procesoru AVR

Procesor má k dispozici 32 registrů délky 8 bitů, všechny jsou schopny pracovat s ALU (aritmeticko-logickou jednotkou), zjednodušeně lze říct, že procesor obsahuje 32 akumulátorů, což je oproti procesorům Atmel řady 8051 velká výhoda, protože tyto procesory obsahovaly akumulátor jen jeden.

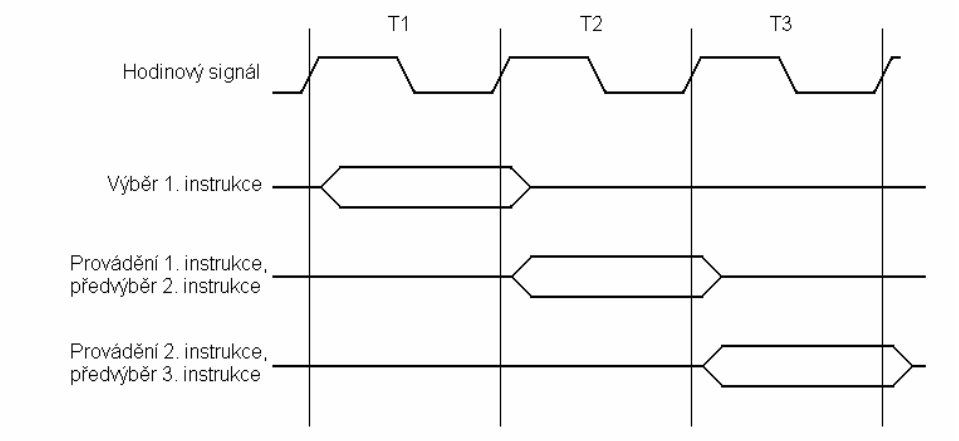
Jako paměť programu je použita zabudovaná paměť typu FLASH, která má u tohoto typu procesoru velikost 32KB. Tuto paměť je možno programovat buď přes paralelní rozhraní, sériové rozhraní SPI nebo rozhraní JTAG.

Jako datová paměť je u tohoto procesoru použita paměť typu RAM, která má velikost 2KB, procesor však obsahuje i paměť typu E^2 PROM, která má velikost 1KB.

Mikroprocesor obsahuje čtyři 8bitové vstupně výstupní porty PA až PD, výstupní proud dosahuje velikosti až 20mA, čili je zde možnost přímo budit např. LED (diodu), maximální proud, který lze dohromady odebírat z výstupů je však 40mA. Procesor dále obsahuje obvod reálného času RTC, dva 8bitové čítače/časovače a jeden 16bitový, dále má 8 kanálů na jeden 10bitový A/D převodník a 6 kanálů pro pulsně šířkovou modulaci PWM. Procesor také umožňuje použít tzv. interní WatchDog časovač, což je hlídač správného běhu programu. U tohoto procesoru není potřeba použít žádný vnější oscilátor, či vnější hodinový vstup, protože procesor již oscilátor obsahuje.

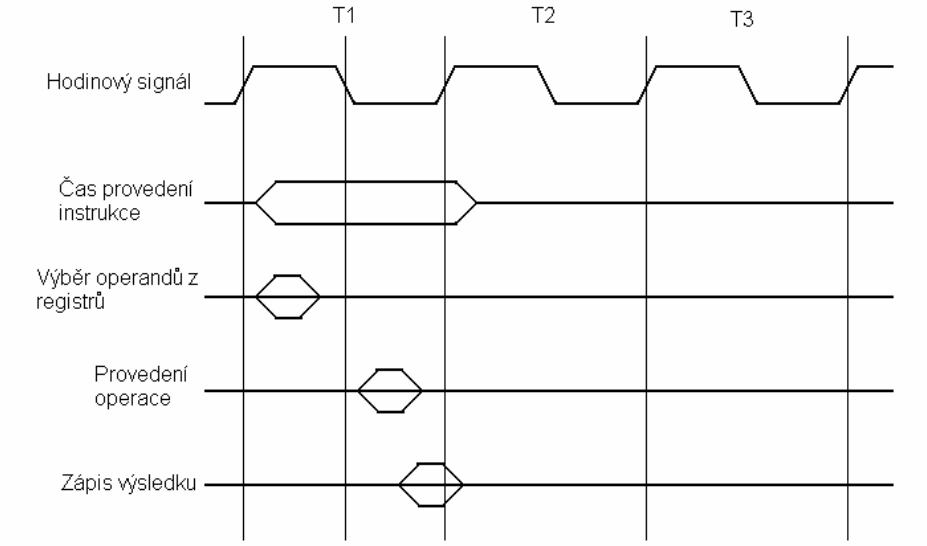
Strojový cyklus je u AVR stejný jako hodinový cyklus (u 8051 obsahuje jeden strojový cyklus 12 hodinových cyklů).

Další vlastností procesorů AVR je tzv. předvýběr instrukce (prefetch), znamená to, že procesor si z paměti programu vybere instrukci, kterou má provést a v dalším hodinovém cyklu ji provede, ale zároveň si i přečte další instrukci, v dalším hodinovém cyklu se provede druhá instrukce a předvybere se třetí atd. Vše je zobrazeno na obrázku 2.2.



Obr. 2.2: Předvýběr instrukce

Mikroprocesory AVR se také vyznačují tím, že dokáží v jednom hodinovém cyklu načíst oba operandy aritmeticko-logické instrukce, provést výpočet a výsledek uložit. Oba operandy se načtou při náběžné hraně hodinového signálu a aritmeticko-logická operace je provedena při sestupné hraně hodinového signálu. Výsledek se ukládá do cílového registru před náběžnou hranou dalšího hodinového cyklu, čili ten již pracuje s aktuálními daty (viz obr. 2.3).



Obr. 2.3: Časování jednocyklové aritmeticko-logické instrukce

Dalším prvkem, který zvyšuje výpočetní výkon procesorů AVR je ten, že procesory mají stejnou délku instrukce a to 16bitů (pouze instrukce LDS a STS mají délku 32 bitů). V těchto 16 bitech jsou uloženy veškeré informace o instrukci a použitých operandech v dané instrukci. Tato vlastnost zjednodušuje realizaci dekodéru instrukcí, díky čemuž se zvyšuje rychlost procesoru.

Procesory AVR obsahují tři bajty tzv. signatury, což jsou bajty, které jsou do procesoru zapsány výrobcem a podle těchto tří bajtů lze poznat o jaký se jedná procesor.

2.2 Způsob programování

Při výběru způsobu programování byla volba téměř jasná, protože od firmy Honeywell byl k dispozici poskytnut programátor AVR Dragon, který umožňuje hned několik programovacích a ladících rozhraní.

Při výběru programovacího jazyka se nabízejí dvě možnosti, buď programovací jazyk ASSEMBLER, nebo C. K řešení této úlohy byl vybrán programovací jazyk C.
S programovacím jazykem ASSEMBLER jsem pracoval na střední škole a s programovacím

jazykem C v předchozích dvou ročnících studia na VUT. Po těchto zkušenostech se mi programování v jazyce C jeví jako výhodnější a jednodušší.

V porovnání s ASSEMBLEREM se ve zdrojovém kódu jazyku C daleko lépe orientuje a celkově práce v něm je pohodlnější.

Jako vývojové prostředí byl zvolen program AVR Studio a to z toho důvodu, že je ho možno stáhnout i na stránkách ATMEL, čili lze předpokládat, že to bude optimální volba.

Možností také bylo také použít přímo WinAVR i jako vývojové prostředí, ale práce s ním se jeví složitější. Sice lze psát program například pouze jen v poznámkovém bloku a potom jej pouze pomocí části programového balíčku MFile zkompilovat pro překladač. Po nainstalování WinAVR i AVRStudia však vše funguje i v AVRStudiu a navíc se v tomto programu daleko lépe orientuje a jsou zde vypsány veškeré registry použité u daného procesoru, což je obrovskou výhodou, zvláště pokud chceme používat debbuger, který slouží k simulaci běhu programu.

2.2.1 Programátor AVR Dragon[3]

AVR Dragon patří k programátorům, které nejsou příliš drahé, ale jsou výkonné. Tento programátor umí pracovat se všemi procesory rodiny Atmel, dále může fungovat také jako JTAG rozhraní pro ladění aplikace přímo v mikroprocesorech, což znamená, že můžeme nahrát zdrojový kód do procesoru a necháme jej připojen v patici programátoru. Pomocí programu AVRStudio se k němu přes programátor připojíme a můžeme po jednotlivých strojových cyklech sledovat jak se naprogramovaný procesor chová aniž bychom jej museli připojovat k nějakému zařízení. Tato funkce je však možná jen pro mikroprocesory, které mají FLASH paměť do 32kB.

Programátor podporuje následující programovací a ladící rozhraní:

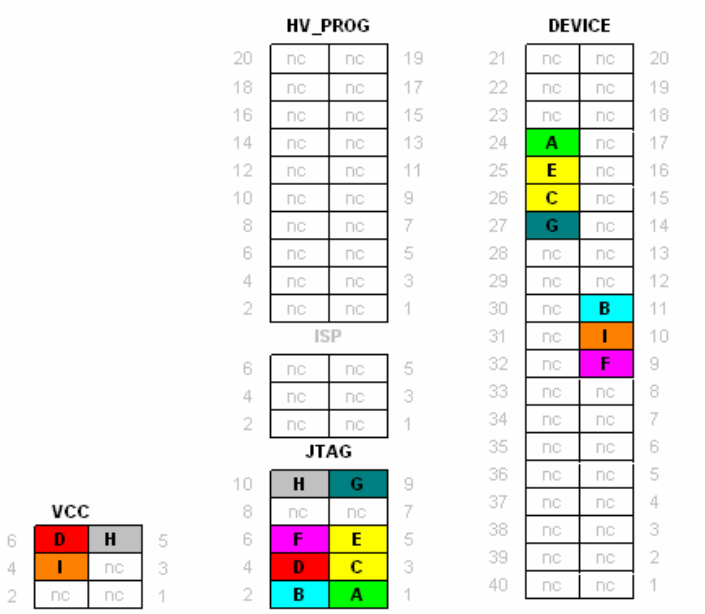
- ISP – sériové programování přímo v aplikaci, 3 vodičové (plus napájecí vodiče).
- JTAG – 4 vodičové programování (plus napájecí vodiče).
- Sériové programování vyšším napětím.
- Paralelní programování.
- JTAG ladění pro procesory s FLASH pamětí do 32kB.
- DebugWIRE jednovodičové ladící rozhraní pro procesory s malým počtem pinů.

S počítačem programátor komunikuje přes USB rozhraní, přes které je zároveň i napájen, čili není zapotřebí žádných dalších zdrojů, ale zároveň je zde možnost připojení

externího napájení v případě, že zařízení bude odebírat větší proud, než je schopno USB rozhraní dodat, tento problém může však nastat jen v případě, že budeme používat programátor v režimu JTAG ladění a budeme k mikroprocesoru připojovat zařízení, která mají vyšší odběr proudu.

Pro programování bylo vybráno rozhraní JTAG, protože jeho zapojení je stejné, jako když chceme použít JTAG ladění.

Zapojení programátoru na JTAG rozhraní je na obrázku 2.4:



Obr. 2.4: Zapojení programu na JTAG rozhraní

Programátor je plně kompatibilní s programovacím prostředím AVR Studio4, které je využito k programování. Je zde i návod na jeho zapojení a možnosti jeho využití.

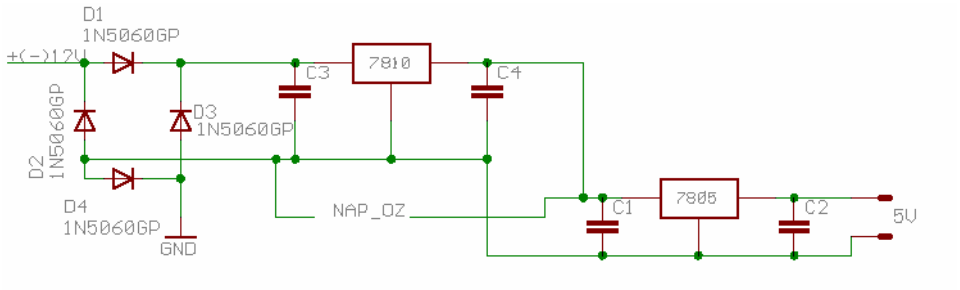
Pokud však chceme programovat v AVR Studiu v jazyce C, tak je zapotřebí si nainstalovat i kompilátor WinAVR, se kterým AVR Studio spolupracuje a pomocí něho vytváří hexadecimální program, který je možno přímo nahrát do paměti programu procesoru.

Při použití těchto dvou programů je však zapotřebí nejprve nainstalovat WinAVR a až poté AVR Studio, protože AVR Studio vyhledává vhodné nástroje pro spolupráci a pokud již máme nainstalované WinAVR, zvolí ho jako výchozí a bude s ním spolupracovat.

3 Napájení zařízení

Zařízení je napájeno stejnosměrným napětím o hodnotě 5V, jako zdroj napětí je použit adaptér o výstupním napětí 12V. Výstupní napětí adaptéru je stabilizováno na 10V pomocí

obvodu 7810. Před stabilizátor je zároveň přidán diodový můstek (obr. 3.1), který je zde použit proto, aby zařízení mohlo být napájeno jak kladným, tak i záporným napětím. V katalogovém listu obvodu 7810 i 7805 [6] je uvedena maximální hodnota vstupního napětí a to 35V, čili lze předpokládat, že bychom pro napájení zařízení mohli použít i takto vysoké napětí. V našem případě však bude z 10V napájen operační zesilovač, který je použit pro ošetření vstupů proti přepětí a 5V budeme napájet ostatní zařízení (mikrokontrolér, MAX232, digitální potenciometr).



Obr. 3.1: Schéma zapojení napájení

4 Vstupy zařízení

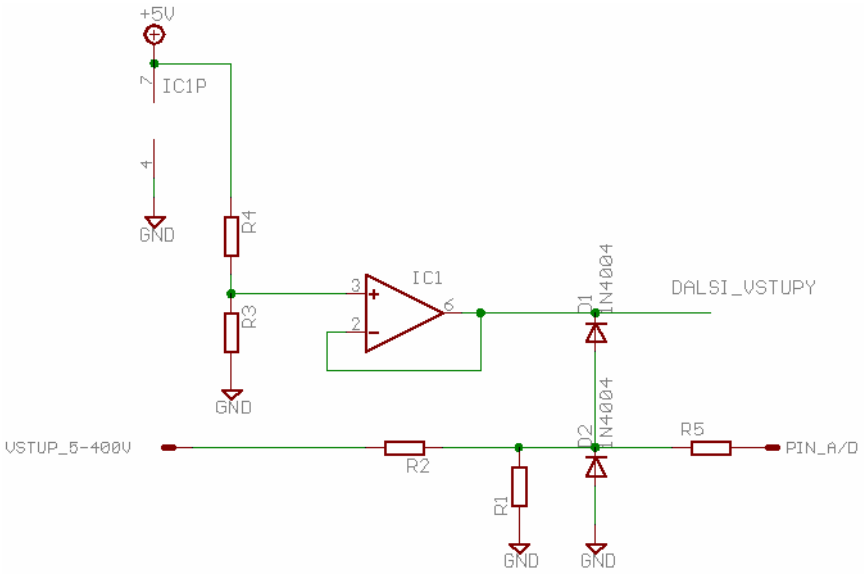
Zařízení bude mít nejméně čtyři vstupy, které budou konfigurovatelné pro čtyři rozsahy napětí - do 5V, do 50V, do 200V a do 400V (okamžitá hodnota). Aktivní může být vždy pouze jeden vstup a pro správnou funkci je potřeba, aby neaktivní vstupy byly uzemněny. Pro rozsahy s vyšším maximálním napětím než je 5V se zařadí do cesty signálu odpovídající odporový dělič (viz obr. 4.1), který zajistí, že nebude na vstup nikdy přivedeno napětí vyšší, než 5V. Pokud však bude aktivní vstup, který je konfigurován na 400V, musí se tato hodnota následně přepočítat, abychom věděli jaké napětí je na vstup vlastně přivedeno.

Tyto vstupy budou přivedeny na piny A/D převodníku procesoru, který bude v pravidelných intervalech vstupní napětí vzorkovat.

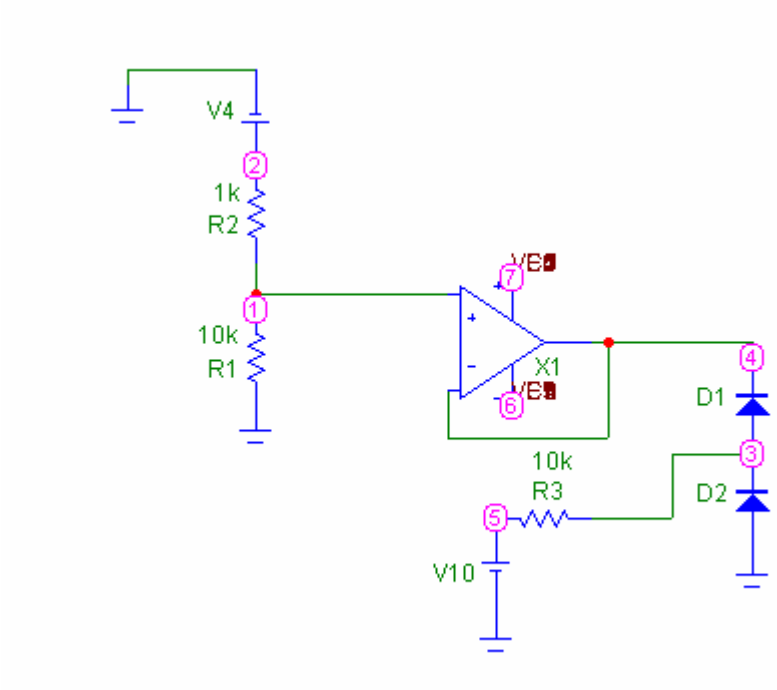
Vstupy mikroprocesoru jsou dále ošetřeny proti přepětí, tedy kdybychom na vstup, který je nastaven například na 5V přivedli napětí vyšší, tak nedojde k jeho destrukci. Schéma zapojení vstupů je znázorněno na obrázku 4.1. Na dalších obrázcích je znázorněna simulace tohoto zapojení v programu Micro-Cap 7.0.9. Na obrázku 4.2 je znázorněno schéma nakreslené v simulačním programu, na obrázku 4.3 jsou znázorněny hodnoty napětí na uzlu 3, pokud měníme hodnoty zdroje V10 od 1 do 400V, jak je vidět napětí na uzlu 3 je zhruba 5,3V při 400V na vstupu. Na obrázku 4.4 je zobrazeno napětí na uzlu 3 při napětích na vstupu od 1 do 5V, napětí je jen o několik setin voltů nižší, než je napětí na vstupu.

Poměr odporů R1 a R2 je pro vstupy následující:

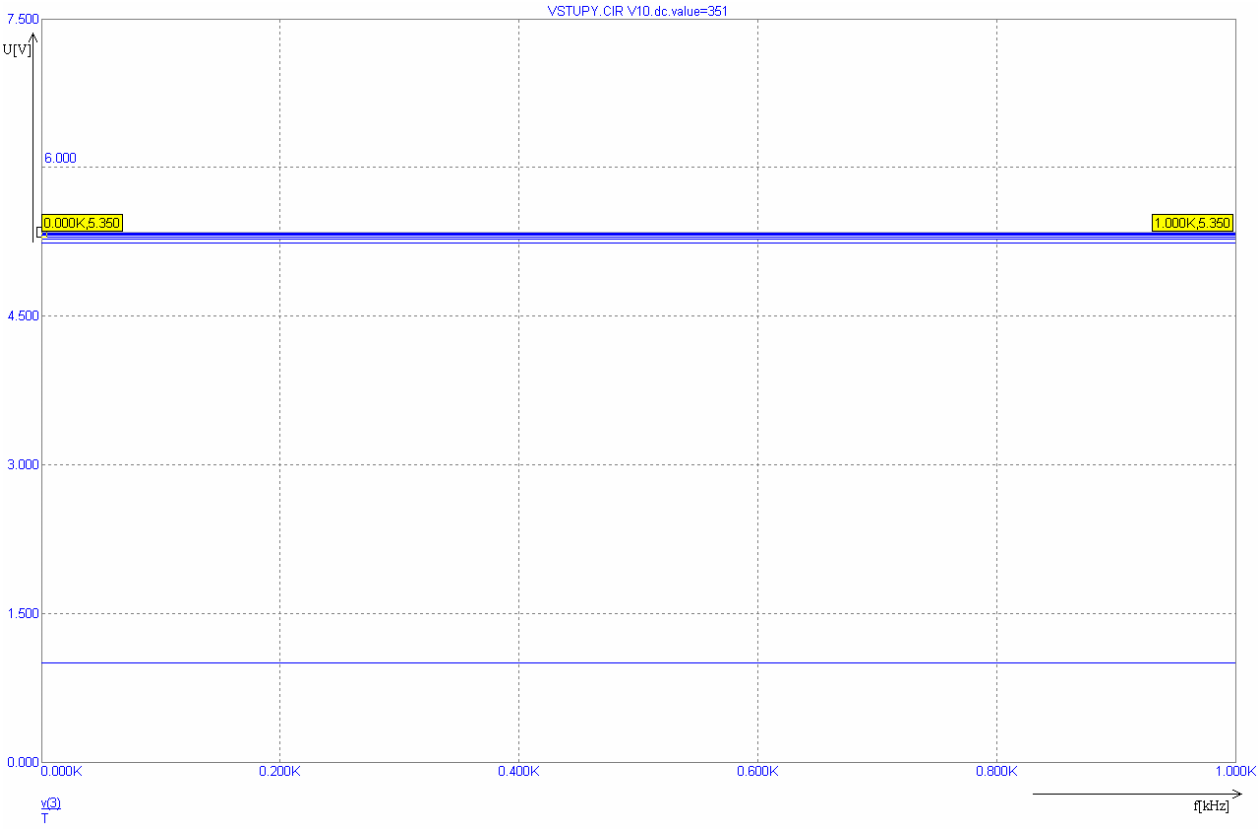
- Vstup do 400V – R1=10kΩ, R2=126Ω
- Vstup do 200V – R1=10kΩ, R2=256Ω
- Vstup do 50V – R1=10kΩ, R2=1111Ω
- Vstup do 5V – R1=10kΩ



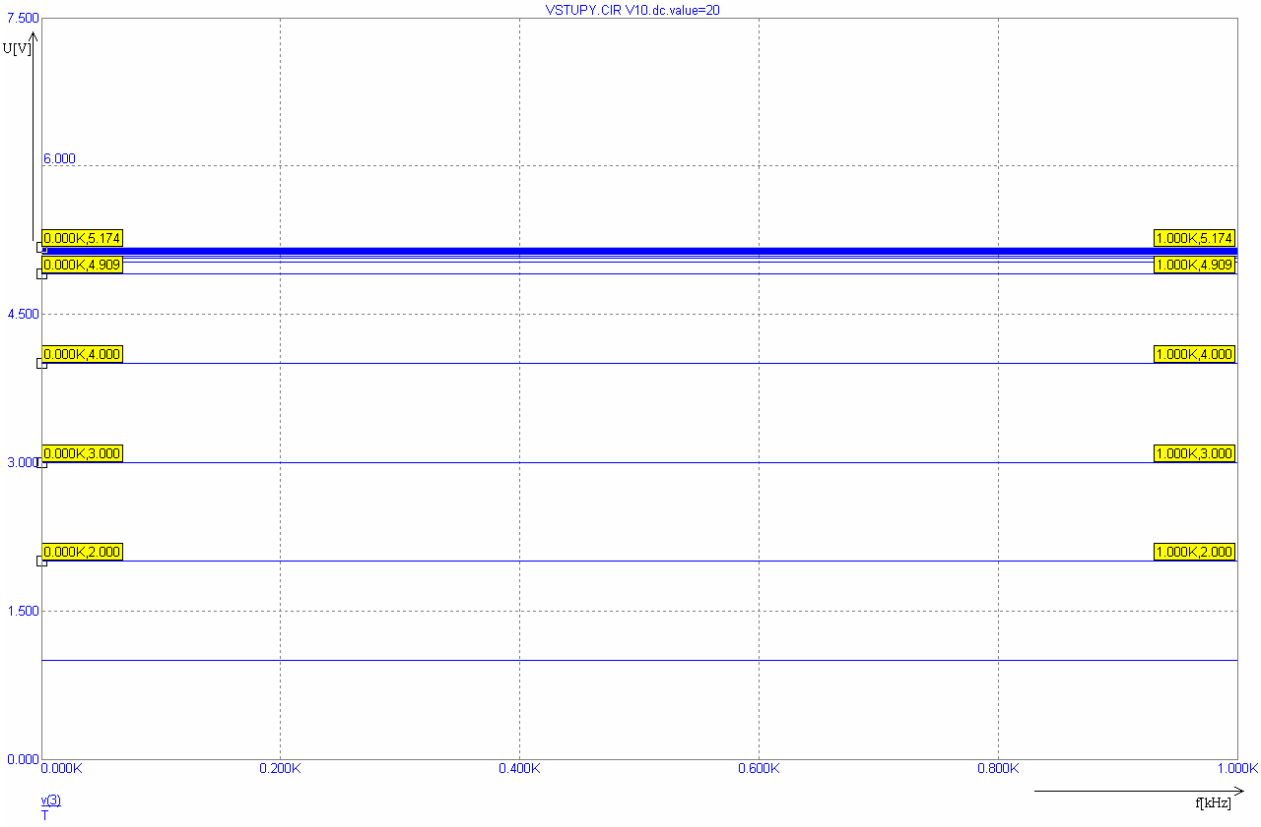
Obr. 4.1: Schéma zapojení vstupů



Obr. 4.2: Schéma vstupů vytvořené v programu MC7



Obr. 4.3: Vyobrazení hodnot napětí na uzlu 3, při napětí 1-400V na zdroji V10



Obr. 4.4: Vyobrazení hodnot napětí na uzlu 3, při napětí 1-20V na zdroji V10

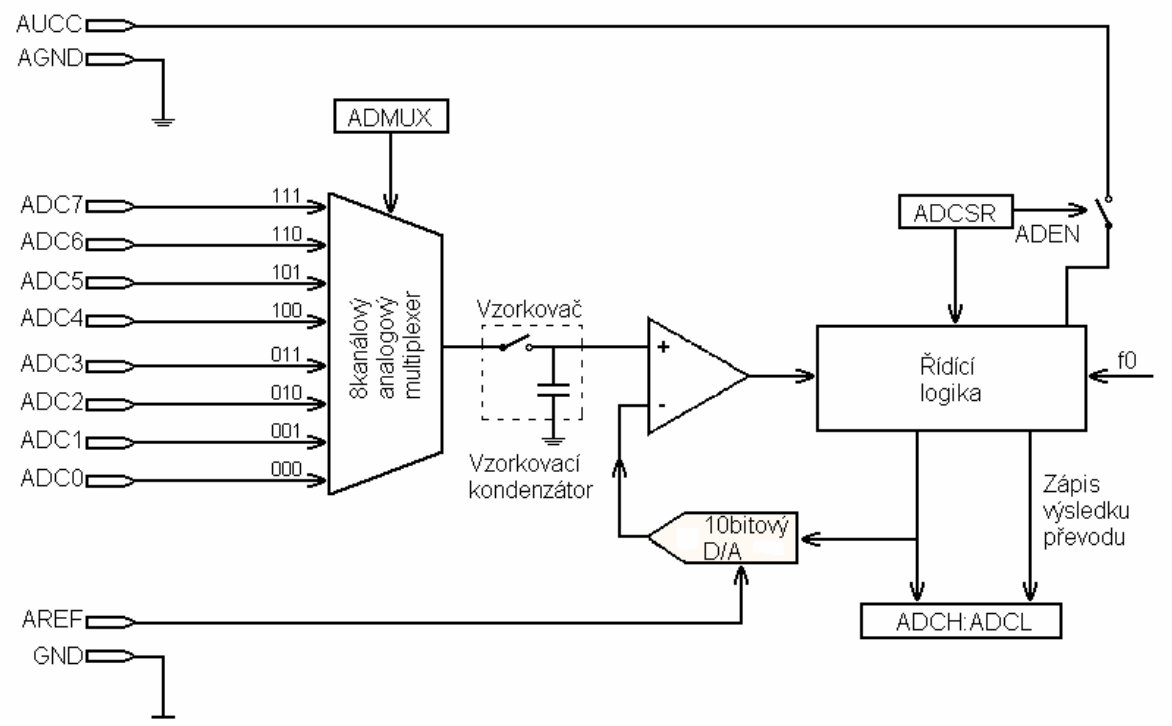
4.1 A/D převodník [1]

Jak již bylo výše uvedeno mikroprocesor AVR ATmega324p obsahuje zabudovaný A/D převodník, což umožňuje měření napětí bez nutnosti připojení vnějších periferií. Proto jej zde využijeme k vyhodnocování přivedeného napětí na vstupy.

Jedná se o 10bitový převodník pracující na principu postupné aproximace, který je připojen na 8kanálový analogový multiplexer, takže je možno připojit a sledovat až 8 vstupů, v našem případě použijeme nejméně 4.

Převodník obsahuje vzorkovač, který je spojený se zesilovačem, který udržuje vstupní napětí v průběhu vzorkování stabilní.

A/D převodník má oddělené napájení, je realizováno pomocí dvou vývodů, GND (na 31 pinu) jako digitální zem, která musí být spojená s analogovou zemí pouze v jednom bodě a vývod AVCC, které nesmí být vyšší než $\pm$ 0,3V.



Obr. 4.5: Blokové schéma A/D převodníku

4.1.1 Operace s A/D převodníkem [1]

Převodník převádí analogové napětí na vstupu na 10bitovou číslicovou hodnotu, kde je minimum zadané hodnotou na vstupu digitální země GND a maximum je zadané vstupem AREF, které nesmí překročit VCC.

To z jakého kanálu chceme hodnoty číst záleží na tom jakou hodnotu zadáme do registru ADMUX.

A/D převodník může pracovat ve dvou režimech „Jednoduchý převod“ a „Volný běh“. O jaký režim se bude jednat se volí bitem **ADATE** z registru **ADCSRA**. „Jednoduchý převod“ (Single conversion) se vyznačuje tím, že každý převod musí být vyvolán uživatelem, kdežto „volný běh“ (Free running) převody se provádí neustále v určitém časovém intervalu bez nutnosti dalšího programového spouštění.

Celý A/D převodník se spouští pomocí bitu **ADEN** z registru **ADCSRA** a to jeho nastavením do log 1, pokud bude v **ADEN** log 0, tak bude A/D převodník vypnut a nespotřebovává žádnou energii.

Celý převod je však odstartován nastavením bitu **ADSC** z registru **ADCSRA** do log 1, tento bit je nastaven až do doby dokud neproběhne celý převod a poté je hardwarově vynulován. Pokud dojde během převodu ke změně nastavení registru **ADMUX**, čili ke změně kanálu, tak není tato změna respektována až do té doby, dokud se nedokončí právě probíhající převod.

Výsledkem převodu je 10bitový výsledek, který se ukládá do registrového páru **ADCH** a **ADCL**. Pokud chceme výsledek z registrů číst, musíme nejprve přečíst z registru **ADCL**, aby se zajistilo že **ADCH** obsahuje výsledek stejného převodu, protože pokud je registr **ADCL** jednou přečten je přístup k němu dále zablokován. To znamená, že **ADCL** nelze číst dvakrát za sebou, pokud by to možné bylo došlo by k přečtení chybného výsledku. Pokud však dojde k přečtení i registru **ADCH**, tak je přístup k **ADCL** opět povolen.

4.1.2 Řídící registry

Pro A/D převodník jsou stěžejní tři registry, kterými je ovládán, jsou to **ADMUX**, **ADCSRA** a **ADCH:ADCL**.

- **ADMUX:** tento registr slouží k výběru kanálu, na kterém budeme napětí pomocí A/D převodníku měřit a to konkrétně pomocí dolních tří bitů **MUX0**, **MUX1** a **MUX2**, z čehož plyne, že můžeme vybírat mezi osmi kanály ADC7 až ADC0. Bity **REFS1** a **REFS1** slouží k nastavení normálového napětí (viz Tab.4.1).

REFS1	**REFS0**	Výběr referenčního napětí
0	0	Napětí na AREF
0	1	Napětí na AVCC s externím kondenzátorem na AREF

1	0	Vnitřní napětí 1,1V s externím kondenzátorem na AREF
1	1	Vnitřní napětí 2,56V s externím kondenzátorem na AREF

Tab. 4.1: Význam bitů REFS1 a REFS0 z registru ADMUX

V našem případě budeme používat jako referenční napětí z pinu AREF (**REFS0**=**REFS1**= log „0"). Bit **ADLAR** zde slouží jako příznak toho, jestli byly bity **ADCH**:**ADCL** správně přečtené.

Registr **ADMUX** je zobrazen v tabulce 4.2.

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Tab. 4.2: Registr ADMUX

- **ADCSRA:** tento registr obsahuje řídící bity a příznaky A/D převodníku, jednotlivé bity jsou vyobrazeny v tabulce 4.3.

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Tab. 4.3: Registr ADCSRA

1. **ADEN (zapnutí A/D převodníku)** – nastavením tohoto bitu se celý A/D převodník zapíná, pokud se vynuluje dříve, než je převod dokončen, tak se převod předčasně přeruší.
2. **ADSCA (start převodu)** – pokud používáme režim „jednoduchý převod", tak je potřeba tento bit nastavit do log „1" před každým převodem.
   V režimu „volný běh" stačí nastavit bit do log „1" jen při prvním převodu.
3. **ADATE (výběr režimu)** – pokud je nastaven do log „1", tak je zvolen režim „Volný běh" a pokud je nastaven do log „0", tak je zvolen režim „jednoduchý převod".
4. **ADIF (příznak přerušení převodníku)** – tento bit se nastaví do log „1", po dokončení převodu a zaktualizování registrového páru **ADCH:ADCL**. Pokud je tento bit a bit **ADIE** roven log „1", tak se provede obslužná rutina přerušení odpovídající dokončení převodu. Při vstupu do obslužné rutiny se tento bit hardwarově nuluje, programově jej lze vynulovat tak, že do něj zapíšeme log „1".

5. **ADIE (povolení přerušení převodníku)** – pokud je tento bit roven log „1“, tak je po konci převodu vyvoláno přerušení odpovídající dokončení převodu.
6. **ADPS2** až **ADPS0 (dělička převodníku)** – těmito bity se nastavují vlastnosti děličky.

- **ADCH:ADCL:** do těchto dvou registrů se ukládá výsledek A/D převodníku. Z registru **ADCH** jsou dostupné pouze dva spodní bity, které odpovídají dvěma nejvyšším bitům A/D převodníku.

  Pokud je **ADCL** přečten, neobnoví se jeho hodnota do té doby, dokud nebude přečten i registr **ADCH**. Tímto je zaručeno, že **ADCL** nebude přečten dvakrát za sebou.

4.1.3 Měření napětí na vstupu pomocí A/D převodníku

A/D převodník nám zde slouží k měření napětí na vstupu. Protože budeme používat 4 vstupy, každý nadefinovaný na jiné napětí (5, 50, 200, 400V), musíme určit konstantu, kterou budeme provádět přepočet z maximálních 5V na skutečnou hodnotu, která je na vstup přivedena. Dalším problémem je to, že převodník je 10 bitový, což znamená, že maximální hodnota 5V odpovídá číslu 1024. Pokud bychom se chtěli vyhnout složitějším výpočtům, bylo by řešení přivést na vstup AREF přivést napětí vyšší, než je napětí napájecí tak, aby maximální hodnota 5V odpovídala číslu 1000, bohužel v datasheetu mikroprocesoru je výslovně zakázáno přivést na vstup AREF vyšší napětí, než je napětí napájecí. Proto je potřeba ještě upravit vstupní děliče tak, aby nebylo přivedeno maximální napětí jako 5V, ale zhruba 4,9V, proto bude výhodné použít trimry (400V – R2=126,5Ω; 200V – R2=256,3Ω; 50V – 1109,6Ω).
Část zdrojového kódu řešící tento problém:

```
do
{
ADCSRA=0xC3;                    //nepovolené přerušení a
                                //jednoduchý převod
delay2();
prevod=ADCL;                    //čtení výsledku převodu
prevod= ADCL | (ADCH<<8);
AktivVstup=ADMUX;               //který vstup je aktivní
ADMUX++;                        //další vstup
PocTestVstup++;                 //počet testu
if (ADMUX==4) ADMUX=0;          //testování 4 vstupů
                                //proběhne znovu
if (PocTestVstup==8) prevod=1;     //aby program nebezel
                                   //stále pokud neni
}                                  //zadny vstup aktivni
```

```
while(prevod==0);                    //cekej na aktivni
                                     //vstup
prevod--;
PocTestVstup=0;


if (AktivVstup==0)                   //přepočty napětí
     {                               //podle aktivního
     C=prevod/2;                     //vstupu
     A=prevod/200;
     B=(C-(100*A));
     }
if (AktivVstup==1)
     {
     C=prevod/2;
     A=prevod/20;
     B=(C-(10*A));
     }
if (AktivVstup==2)
     {
     A=prevod/5;
     B=0;
     }
if (AktivVstup==3)
     {
     A=(10*prevod)/25;
     B=0;
     }
```

Příklad:

Přivedeme napětí, které odpovídá číslu 845 na vstup konfigurovaný na maximální napětí 50V. Podle vzorce 4.1 vypočteme, že hodnota přivedeného napětí je 42,25V.

$$U_{50V} = \frac{50V}{1000} \cdot 845[V] \qquad (1)$$

Budeme postupovat podle programu $C = \frac{845}{2} = 422$, $A = \frac{845}{20} = 42$,

$B = (422 - 10 \cdot 42) = 2$, výsledek je v proměnných A a B a to 42,2V.

U vstupů do 200V a 400V budou zobrazovány pouze jednotky voltů, protože půjde o hodnoty napětí, kde bude přesnost na desetinná místa nepotřebná.

5 Komunikace zařízení s PC

Jedním ze základních požadavků na zařízení je, aby bylo schopno komunikovat s osobním počítačem. Pomocí počítače by mělo být možno provádět jednoduchá nastavení a načítat ze zařízení uložená data, se kterými pak bude možno dále pracovat.

Mikroprocesor ATmega324p obsahuje dva zabudované asynchronní sériové kanály (UART), které se pro tuto činnost dají perfektně použít. Oba UART tohoto procesoru jsou plně duplexní přijímače a vysílače.

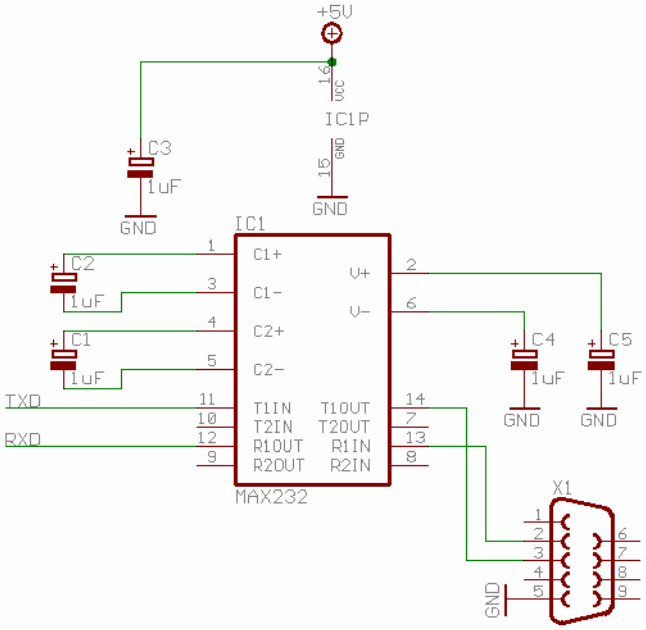
5.1 Vlastnosti UART [1]

UART u tohoto procesoru umožňuje nastavení velkého počtu přenosových rychlostí pomocí generátoru přenosové rychlosti, narozdíl od procesorů řady 8051 není generátor realizován časovačem, takže použitím UART neztratíme jeden časovač, jak tomu u 8051 je.

Dále UART dosahuje vysokých rychlostí i při nízkých kmitočtech krystalu, délka znaku je 8/9 bitů, umožňuje filtraci šumu na straně příjmu a detekci chyb (ztráta znaku, chyba rámce, falešný start-bit). UART má tři zvláštní přerušení (dokončení vysílání, vyprázdnění vysílacího datového registru, dokončení příjmu).

5.2 Propojení PC a mikrokontroléru

Pokud připojíme mikrokontrolér k počítači přes sériový port RS-232, tak tomu musíme mikrokontrolér přizpůsobit, protože mikrokontrolér využívá logiku TTL. K převedení úrovní TTL na úrovně RS-232 se dá perfektně použít obvod MAX232. K obvodu je třeba připojit několik kondenzátorů, pro jeho správnou funkci (obr. 5.1). K mikrokontroléru se obvod připojí pomocí pinu T1IN do TxD a pinu R1OUT do RxD (obr. 5.1).



Obr. 5.1: Schéma zapojení obvodu MAX232 a způsob připojení k mikrokontroléru

5.3 Řídící registry [1]

UART je ovládán pomocí čtyř vstupně/výstupních registrů, datový registr **UDR1 (UDR0)**, který obsahuje přijatou, nebo vyslanou hodnotu, dále registrový pár na nastavení rychlosti přenosu **UBRR1H:UBRR1L (UBRR0H:UBRR0L)**, stavový registr **UCSR1A (UCSR0A)**, ve kterém jsou příznaky stavu přenosu a posledním registrem je řídící registr **UCSR1B (UCSR0B).**

- **UDR1:** tento registr je fyzicky představován jako registry dva, protože pokud chceme vysílat, musíme vysílací registr naplnit daty, které chceme vyslat a pokud z něj čteme, tak čteme přijímací registr.
- **UBRR1H:UBRR1L:** tento registr ovládá přenosovou rychlost UART a to podle následujícího vzorce:

$$PR = \frac{f_0}{16 \cdot (UBRR + 1)} \qquad (2)$$

kde PR je výsledná přenosová rychlost v Bd, f_0 je kmitočet krystalu procesoru v Hz a UBRR je obsah registrů **UBRR1H** a **UBRR1L**.

- **UCSR1A:** tento registr obsahuje stavové příznaky přenosu. Význam jednotlivých bitů:
    1. **RXC1** – příjem kompletní. Příznakem je nastavení do log „1“, ten se nastaví, pokud přijde znak přenesen z přijímacího registru **UDR1**. Tento příznak se nastavuje i při případné chybě rámce. Pokud bude bit **RXCIE1** nastaven, tak se po provedení příjmu provede vyvolá přerušení. **RXC1** se nuluje hardwarově čtením registru **UDR1**. Pokud obslužná rutina tohoto přerušení nepřečte registr **UDR1**, vyvolá se přerušení po jeho obsluze znovu.
    2. **TXC1** – vysílání kompletní. Tento bit se nastaví po vyslání stop-bitu, po jeho vyslání lze do vysílacího registru přes **UDR1** vložit nová data. Tento příznak se používá hlavně při plně duplexním přenosu. Pokud je nastaven bit **TXCIE1**, tak se provádí to samé jako u **RXC1**, jen s tím rozdílem, že **TXC1** lze nulovat také tím, že se do něj zapíše log „1“.
    3. **UDRE1** – datový registr je prázdný. Tento bit se nastaví při přenesení obsahu **UDR1** do vysílacího registru a **UDR1** je připraven přijmout další data. Pokud je nastaven **UDRIE1** vyvolá nastavení **UDRE1** přerušení. Obslužná rutina přerušení musí provést zápis do **UDR1**. Jinak dojde po

ukončení obslužné rutiny tohoto přerušení k opětné aktivaci. Po resetu je **UDRE1** rovno 1, aby se indikovala připravenost na vysílání.

4. **FE1** – chyba rámce. Bit **FE1** se nastaví, pokud dojde k chybě rámce, tzn. že bit odpovídající stop-bitu je roven 0. Bit se hardwarově nuluje, je-li rámec v pořádku.
5. **DOR1** – ztráta znaku. Bit je nastaven, pokud dojde ke ztrátě znaku (znak dříve uložený do **UDR1** nebyl přečten před příjmem dalšího znaku). Bit **DOR1** je nastaven pouze jednou, což znamená, že je buferován. Nulování bitu se provádí hardwarově čtením dat z **UDR1**.

- **UCSR1B:** tento registr je řídící registr UART. Význam jednotlivých bitů:
    1. **RXCIE1** – povolení přerušení po dokončení příjmu. Pokud bude tento bit nastaven, tak nastavení bitu **RXC1** vyvolá příslušné přerušení.
    2. **TXCIE1** – povolení přerušení po dokončení vysílání. Zde je to stejný případ jako u **RXCIE1**, ale přerušení se vyvolá nastavením **TXC1**.
    3. **UDRIE1** – povolení přerušení při vyprázdnění datového registru. Pokud je tento bit nastaven, vyvolá nastavení bitu **UDRE1** přerušení odpovídající vyprázdnění vysílacího datového registru.
    4. **RXEN1** – příjem povolen. Povoluje příjem. Je-li tento bit nastaven do nuly, tak se nemůžou nastavit příznaky **RXC1**, **DOR1** a **FE1**. Pokud jsou však tyto příznaky nastaveny a dojde k vynulování **RXEN1**, tak nedojde k jejich vynulování.
    5. **TXEN1** – vysílání povoleno. Povoluje vysílání. Pokud se tímto bitem zakáže vysílání v průběhu vysílání znaku, nedojde k jeho přerušení, ale odešle se znak uložený v **UDR1**.
    6. **UCSZ12** – tento bit určuje zda se jedná o příjem či vysílání devíti bitů. Devátý je čten z **RXB81** a je zapisován do **TXB81**. Tento devátý bit se většinou používá jako parita, nebo přídavný stop-bit. Čili pokud je tento bit v log „1“, tak **RXB81** obsahuje devátý bit přijatého znaku a **TXB81** devátý bit vyslaného znaku.

5.4 Vysílání dat [1]

Vysílání dat je vždy spuštěno zápisem znaku do registru **UDR1**, ze kterého jsou data přenesena do posuvného vysílacího registru, viz obr. 5.2.

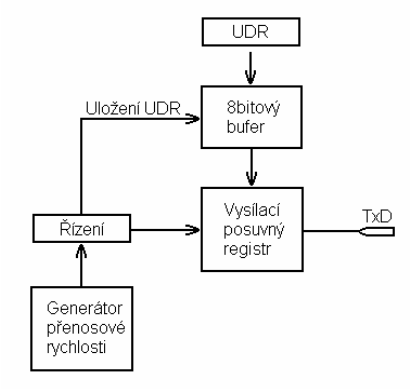
Pokud je zápis nového znaku do **UDR1** proveden po stop-bitu znaku, který byl právě vysílán, tak je vysílací posuvný registr naplněn novými daty okamžitě, ale pokud byl nový znak do **UDR1** zapsán před stop-bitem aktuálně vysílaného znaku, tak je zápis do vysílacího posuvného registru odložen až do odvysílání právě odesílaného znaku.

Jestli je vysílací posuvný registr prázdný, tak se data z **UDR1** přenesou okamžitě, současně se i nastaví příznak **UDRE1**, který značí že je datový registr prázdný, je-li tento příznak nastaven je UART schopen přijmout další znak pro vysílání. Současně s přenesením dat do vysílacího posuvného registru dochází k vynulování nultého bitu vysílacího posuvného registru (start-bit).

V hodinovém impulsu generátoru přenosové rychlosti, který následuje po přenosu dat z **UDR1** do vysílacího posuvného registru, je vyslán start-bit na vývod TxD. Pak následují zapsaná data (první je LSB). Když je vyslán stop-bit, nahrává se do vysílacího posuvného registru další znak (byl-li uložen do **UDR1** v průběhu vysílání). V průběhu tohoto přenosu je nastaven příznak **UDRE1**. Nejsou-li v **UDR1** nová data, zůstane příznak **UDRE1** nastaven až do zápisu dat do **UDR1**. Nejsou-li zapsána nová data v **UDR1** a stop-bit byl na vývodu TxD po dobu jednoho impulsu generátoru přenosové rychlosti, nastaví se příznak **TXC1** (přenos je kompletní).

Jak již bylo zmíněno, tak bit **TXEN1** je bit, kterým se povoluje vysílání, je-li tedy tento bit nastaven, tak je vývod PD3(PD1) brán jako vývod TxD, bez ohledu na nastavení **DDRD**, pokud bit nastaven není, tento vývod je brán jako normální vstupně výstupní vývod.

Zjednodušené schéma vysílače je na obrázku 5.2.



Obr. 5.2: Zjednodušené schéma vysílače UART

Protože je použit pouze terminál Hercules a není zhotoven žádný program, který by byl schopen vyčítat hodnoty z komunikačního protokolu, tak je před každým odesláním hodnot do počítače využíváno převádění veškerých hodnot na string. Toto nám umožňuje zobrazovat veškeré hodnoty na terminálu.

Část programu, sloužící k vysílání dat do PC:

```
itoa(1,tempstr,10);     // číslo int1 převeď na string tempstr
u_puts(tempstr);           // a zapiš do UART

void u_puts( char *text )
{
unsigned char i=0,temp;
     do
     {
     temp = text[i];
     if(temp==0) break;       //pokud v temp není žádný znak
     u_putc(temp);            //smyčka se ukončí
     i++;
     }
     while(temp>0);           //probíhá dokud jsou přiváděny
}                             //znaky

void u_putc( char data )
{
while ( !( UCSR0A & (1<<UDRE0)) );      // čekej do
                                        //vyprázdnění bufferu

UDR0 = data;
}
```

5.5 Přijímání dat [1]

Přijímání dat funguje tím způsobem, že přijímač vzorkuje signál vývodu RxD kmitočtem 16x vyšším, než je přenosová rychlost, tzn., že na jeden bit tedy připadá 16 vzorků (obr. 5.3). Pokud je linka nečinná, tedy je v log „1“, tak je jeden vzorek v log „0“ brán jako sestupná hrana start-bitu, čímž se spustí detekce start-bitu. Pro kontrolu jsou posuzovány ještě 8. 9. a 10. vzorek, jsou-li dva nebo více vzorků v log „1“, tak se tento děj bere jako zákmit a start-bit je zamítnut a čeká se na nový.

Pokud je start-bit platný, tak se provádí vzorkování datových bitů následujících za start-bitem, důležité jsou 5. 8. 9. a 10. vzorky. Hodnota, která je ve dvou a více z těchto vzorku je brána jako hodnota bitu. Takto přijaté bity jsou vkládány do posuvného přijímacího registru.

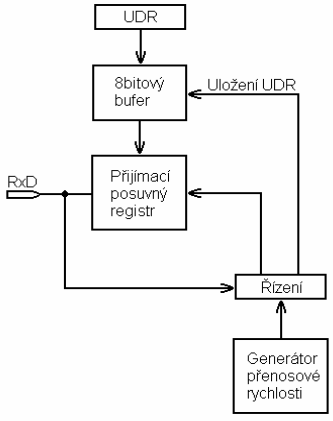
Když do přijímače vstoupí stop-bit, musí být většina vzorků v log „1“. Jsou-li dva nebo tři vzorky v log „0“, tak se nastaví příznak **FE1**, proto by program měl testovat tento příznak před čtením dat z registru **UDR1**.

Ovšem bez ohledu na to, zda je na konci přijímacího cyklu stop-bit detekován, tak jsou data z přijímacího registru přenesena do **UDR1** a současně se nastaví příznak **RXC1**. Registr **UDR1** je fakticky tvořen dvojicí registrů (pro vysílaná data a pro přijímaná data). Při čtení **UDR1** se přistupuje k přijatým datům a při zápisu do něj se přistupuje do vysílacího registru.

Jestliže nedojde po příjmu znaku ke čtení registru **UDR1** do příjmu dalšího znaku, nastaví se příznak **DOR1**, což značí, že posledně přijatý znak nemohl být přesunut do **UDR1** a je tedy ztracen. Příznak **DOR1** je buferovaný, tzn. že si zachovává svou hodnotu do čtení znaku z **UDR1** (po jeho čtení se vždy nuluje). Proto je vždy potřeba testovat stav příznaku **DOR1** před čtením **UDR1**, aby zjistil případnou ztrátu znaku, zvláště je-li nastavena vysoká přenosová rychlost.

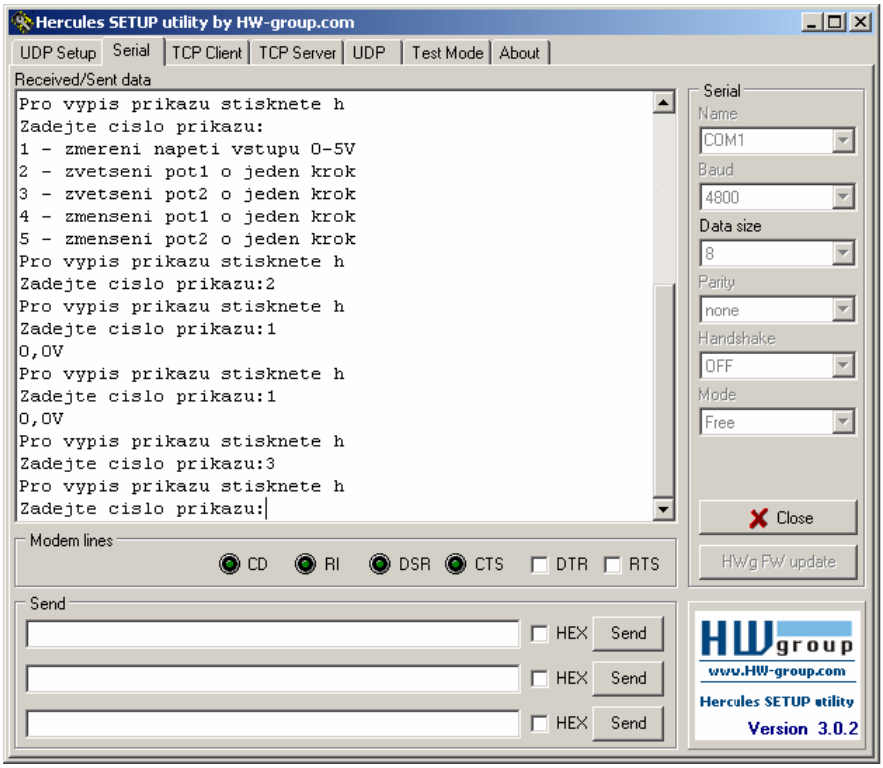
Nastavení bitu **RXEN1** zde má stejnou funkci, jako u vysílání dat bit **TXEN1**.

Zjednodušené schéma přijímače je zobrazeno na obrázku 5.3.



Obr. 5.3: Zjednodušené schéma přijímače UART

Aby bylo možno celé zařízení používat i samostatně, bez potřeby udělání programu, který pomocí přiložených knihoven bude zařízení ovládat, tak je dodán program, kterým lze zařízení ovládat prostřednictvím klávesnice. Program je dodán pouze na přiloženém CD, není součástí bakalářské práce. Komunikace s počítačem je velmi jednoduchá a ovládání je intuitivní (Obr. 5.4).



Obr. 5.4: Zadávání příkazů z klávesnice a jejich vyhodnocování

5.5.1 Vyhodnocování přijatých dat [5]

Pokud chceme data, která přijmeme z procesoru do počítače nějakým způsobem vyhodnotit, musíme vytvořit aplikaci, která toto bude umožňovat a nebo použijeme programy, které toto umožňují a jsou volně přístupné na internetu.

V této práci byla zvolena druhá varianta a byl použit program Hercules a to kvůli jeho přehlednosti a jednoduchosti ovládání.

Jedná se o univerzální program, který zastává funkce všech základních TCP a UDP utilit. Pro nás je však důležitý sériový terminál, který budeme využívat.

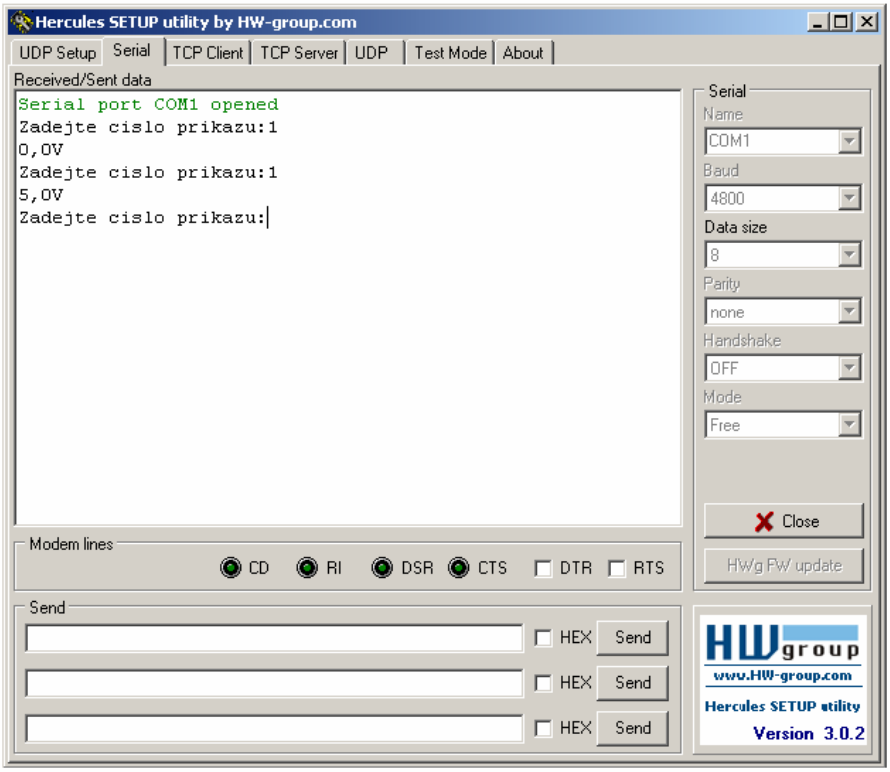
Použití a nastavení programu jako jednoduchý sériový terminál pro sériový port standardu RS-232 je na obrázcích 5.5 a 5.6 Na obrázku 5.6 jsou zobrazeny hodnoty, které lze nastavovat.

- **Name** vybírá, který kanál sériového portu bude použit pro komunikaci.
- **Band** nastaví jakou rychlostí se bude komunikovat, rychlost je v baudech [Bd].
- **Data size** určuje počet datových bitů na znak.

- **Parity** určuje jakým způsobem bude probíhat kontrola přenosu. Možnosti jsou none (žádná kontrola), even (kontrola sudé parity), odd (kontrola liché parity) a mark (kontrola znaménka).
- **Handshake**

  Kontrola přenosu způsobující přerušení přenosu dat v okamžiku, kdy je buffer druhého zařízení zaplněný.
- **Mode**

  Tato volba je určena pro tetování zařízení HW group, my budeme využívat defaultní nastavení „free“.
- Pomocí tlačítka **Open/Close** se otevře/uzavře nastavené spojení.



Obr. 5.5: Použití programu Hercules

Obr. 5.6: Nastavení hodnot programu Hercules při použití jako terminál pro sériový port

6 Ukládání dat

Jako nejschůdnější možnost ukládání dat se v tomto případě jeví použití paměti E^2PROM. U použitého procesoru je navíc paměť E^2PROM již zabudována, její kapacita je 1kB, což pro tento účel bohatě stačí. Jelikož ale není zcela jasné, jaká data a kolik se jich má ukládat, tak není program pro ovládání paměti E^2PROM součástí přiloženého programu. Lze jej ale jednoduše doplnit. Zde je naznačena funkce pro zápis a čtení dat do paměti E^2PROM:

```
EEAR=adr;                    //adresa, kam se bude
                             //ukládat
EEDR=A;                      //uložení hodnoty napětí na
                             //vstupu, pokud tedy tento
                             //program použijeme společně
                             //s A/D převodníkem
setb(EECR,2);
clrb(EECR,1);
delay4k();
setb(EECR,2);
setb(EECR,1);
adr++;                       //zvýšení ukazatele adresy
EEAR=adr;                    //uložení hodnoty desetin
EEDR=B;                      //voltů
```

```
setb(EECR,2);
clrb(EECR,1);
delay4k();
setb(EECR,2);
setb(EECR,1);
adr++;



EEAR=adr;                          //čtení z paměti
setb(EECR,0);
data=EEDR;
adr--;
```

6.1 Vnitřní paměť E^{2}PROM [1]

Mikroprocesory AVR obsahují zabudovanou paměť E^{2}PROM, do které lze zapisovat po jednom bajtu a to jak programově, tak i pomocí programátoru.

Nejvýznamnější vlastností pamětí E^{2}PROM je, že si uchovávají data i po odpojení napájení, narozdíl např. od pamětí RAM. Životnost dat, které paměť E^{2}PROM uchovává je až několik desítek let. Zaručovaný počet programování je 100.000.

Mikroprocesory se liší velikostí paměti E^{2}PROM, v našem případě je její velikost 1kB. S tímto parametrem také souvisí délka registru, který je použit pro adresování E^{2}PROM. V našem případě se tedy jedná o délku 10 bitů a o registrový pár **EEARH:EEARL**.

E^{2}PROM dále disponuje datovým registrem **EEDR**, který má délku 8 bitů (obsahuje přečtená data nebo data pro zápis) a řídícím registrem **EECR**, který povoluje čtení nebo zápis do paměti.

6.1.1 Přístup do paměti E^{2}PROM

Registry pro přístup k E^{2}PROM jsou uloženy ve vstupně/výstupním prostoru. Zápis do paměti je časově náročnou operací, která trvá 2,5 až 4 ms v závislosti na velikosti napájecího napětí. Funkce automatického časování zápisu dovoluje, aby program detekoval okamžik, kdy lze učinit zápis dalšího bajtu.

V okamžiku, kdy program zapisuje do paměti E^{2}PROM, tak musí být zajištěny určité podmínky:

- musí být dostatečně vyhlazeno napájecí napětí,
- pro zvýšení bezpečnosti se doporučuje používat vnější nulovací obvod,
- aby nedošlo k náhodnému zapsání do E^2PROM, je nutno vykonat specifickou zapisovací sekvenci,
- při zápisu do E^2PROM je jádro procesoru zastaveno na 2 hodinové cykly před provedením další instrukce,
- při čtení je jádro procesoru zastaveno na 4 hodinové cykly před provedením další instrukce.

6.1.2 Ovládací registry paměti E^2PROM

Paměť E^2PROM je řízena třemi vstupně výstupními registry (**EEARH:EEARL**, **EEDR** a **EECR**).

- **EEARH:EEARL**: adresový registrový pár paměti, určuje adresu buňky E^2PROM pro čtení či zápis. V tomto případě se tedy jedná o celý registr **EEARL** a dva spodní bity registru **EEARH**.
- **EEDR**: datový registr paměti, který obsahuje data, která se mají zapsat, nebo je naplněn přečtenými daty. Délka registru je 8 bitů, přístup do E^2PROM je po celých bajtech.
- **EECR**: řídící registr paměti, v tomto registru se zadává zda půjde o zápis či čtení z paměti a povoluje se zde přerušení spojené s E^2PROM. Procesor ATmega324p obsahuje 6 řídících bitů v registru **EECR**. Jejich význam je uveden zde:
    1. **EERE** – povolení čtení z paměti. Pokud nastavíme tento bit, tak oznamujeme paměti, že chceme z paměti číst, tento bit však musíme nastavit až po uložení adresy do registru **EEARH:EEARL**. Po přečtení údaje z paměti se bit hardwarově vynuluje a tím se potvrdí platnost dat v registru **EEDR**. Čtení z paměti trvá jednu instrukci, proto není bit **EERE** nutno testovat. Po nastavení bitu **EERE** se totiž jádro procesoru zastaví na 4 hodinové cykly.
    2. **EEWE –** povolení zápisu do paměti. Nastavením tohoto bitu oznamujeme paměti, že chceme provést zápis, nastavovat ho musíme až pokud jsou uloženy platné adresy a data v registrech **EEAR** a **EEDR**. Dále musí být před nastavením bitu **EEWE** nastavit bit **EEMWE**, jinak nebude zápis úspěšný. Zápis spolu s mazáním je časován hardwarově. Po uplynutí doby zápisu je bit **EEWE** hardwarově vynulován. Hiearchicky by měl zápis do paměti vypadat takto:

- čekej, až je **EEWE** = 0 (dokončení předchozího zápisu),
- zapiš adresu do **EEARH:EEARL**,
- zapiš data do **EEDR**,
- zapiš **EEMWE** = 1, **EEWE** = 0,
- do 4 cyklů od kroku 4. zapiš **EEMWE** = 1, **EEWE** = 1.

3. **EEMWE -** hlavní povolení zápisu do paměti. Pokud tento bit není nastaven, tak není možno do paměti zapisovat. Procesor hardwarově nuluje tento bit po uplynutí 4 hodinových cyklů od jeho nastavení, je tak zabráněno náhodným zápisům.
4. **EERIE** – povolení přerušení po dokončení zápisu do paměti. Pokud jsou nastaveny bity **EERIE** a **I**, tak je povoleno přerušení, které se generuje v okamžiku, kdy je E^2PROM připravena na zápis dalšího bajtu (nastane po vynulování bitu **EEWE**).

7 Výstupy

Jak plyne ze zadání, tester musí plnit funkci simulátoru různých snímačů neelektrických veličin a spínačů.

Jako snímač neelektrických veličin by měla být použita součástka, která by umožňovala digitálně měnit svůj odpor. Jako nejlepší řešení se jeví použití digitálního potenciometru.

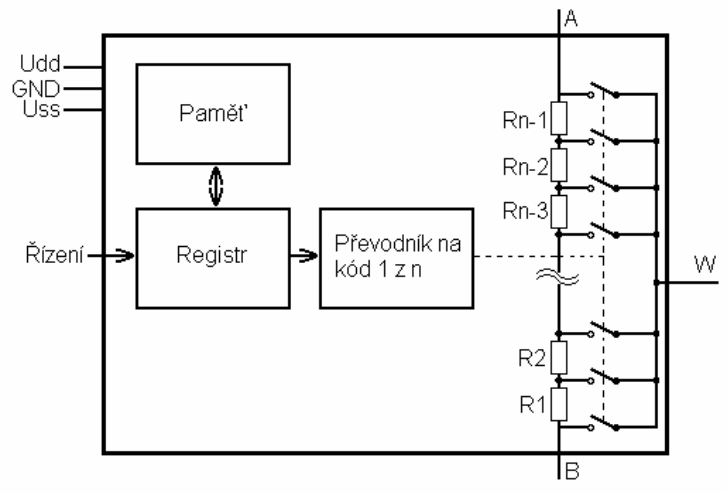
Jako spínače budou použity relé, pro spínací napětí až 400VAC/1A, která budou řízena procesorem.

7.1 Digitální potenciometr

7.1.1 Obecný popis součástky a kritéria výběru určitého typu [2]

Tuto součástku dnes nabízí jen několik firem, např. ANALOG DEVICES, DALLAS SEMOCONDUCTOR, MAXIM, XICOR a MICROCHIP.

Tyto firmy mají za cíl přiblížit se vlastnostem obyčejných mechanických potenciometrů a odstranit zároveň jejich nevýhody. Řešení potenciometru je zobrazeno na obrázku obr.7.1.1.



Obr. 7.1: Vnitřní zapojení digitálního potenciometru

Hlavním parametrem, který určuje použitelnost je počet poloh jezdce, pokud je počet poloh N, potom je mezi vývody A a B sériově zapojeno N-1 rezistorů, ze kterých je sestavena odporová dráha. Jezdec je představován vývodem W, který je do rezistorové sítě připojen pomocí elektronických spínačů, které jsou ovládány řídícím kódem. Pro řízení může být použita komunikace I^2C, paralelní sběrnice a nebo 3vodičová sériová sběrnice SPI a také pomocí sběrnice 1-WIRE, kde by byl použit pouze jeden vývod procesoru, ale tento způsob řízení nabízí pouze firma MAXIM. Nejlepším řešením jak ovládat tyto potenciometry je pomocí mikroprocesoru, ale lze použít také některý port počítače a vyrábí se také potenciometry, které obsahují dvě tlačítka „nahoru dolů“, kterými lze potenciometr mechanicky ovládat. Součástí většiny potenciometrů je paměť na ukládání posledního nastavení potenciometru, které není závislé na napájecím napětí. Po zapnutí potenciometru se z této paměti hodnota načte a potenciometr se tak nastaví do poslední nastavené hodnoty.

Počet poloh jezdce bývá většinou mocnina dvou (32, 64, 128, 256, ...). Doposud nejvyšší rozlišovací schopnost je 1024 poloh, tyto potenciometry vyrábějí firmy Xicor a Analog Devices. Výrobci nabízejí potenciometry s odporem dráhy 1kΩ, 10kΩ, 50kΩ, 100kΩ, 500kΩ, 1MΩ. Přesnost potenciometrů není příliš veliká, dosahuje hodnot 20 – 30%, toto však není příliš velký problém, protože lze jednoduše změřit hodnoty potenciometru a pomocí procesoru vytvořit korekci, kterou budeme nastavovat požadovanou hodnotu. Výrobci potenciometrů dále zaručují vysokou linearitu. Nelinearita bývá většinou menší, než nejnižší bit. Dobrou vlastností je, že změna jezdce není doprovázena rušivými vlivy.

Nevýhodou je, že potenciometr má nezanedbatelný odpor jezdce, který je způsoben nenulovým odporem spínače, s touto hodnotou je třeba počítat zejména při nastavování malých hodnot odporu potenciometru.

Výrobci dále umožňují použít více potenciometrů v jednom pouzdře.

7.1.2 Volba potenciometru

Potenciometr jsem vybíral na internetových stránkách výše uvedených firem.

Hlavním požadavkem bylo, aby měl potenciometr krok 1024, tímto se výběr velice zúžil, protože jak už bylo uvedeno, tuto možnost nabízí pouze firma XICOR a ANALOG DIVICES, pokud bychom navíc chtěli použít nejvýhodnější způsob řízení pomocí sběrnice 1-Wire, tak nenalezneme vhodný potenciometr. Proto musíme vybrat potenciometr, který lze řídit pomocí sběrnice SPI.

Další parametr, který ovlivnil výběr byla maximální hodnota odporu. Požadavek byl, že by měl potenciometr dosahovat hodnoty až 250kΩ. V tomto případě jsem již mohl vybírat mezi konkrétními typy. Protože cena nebyla u různých typu příliš rozdílná, rozhodl jsem se pro výběr potenciometru AD5235, jehož popis je v následující kapitole.

7.1.3 Digitální potenciometr AD5235 ANALOG DEVICES[4]

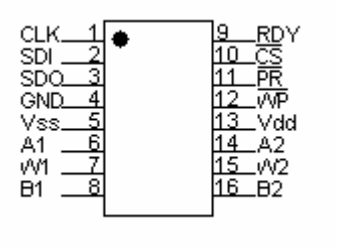
Vlastnosti – dvoukanálový; 1024 kroků; nominální resistence 250Ω; paměť pro nastavení jezdce a opětovné nastavení jezdce; ochrana proti nepovolenému zápisu do paměti; odchylka odporu uložená do paměti; předdefinované lineární inkrementační/dekrementační instrukce; předdefinované ±6dB inkrementační/dekrementační instrukce; ovládání pomocí SPI sběrnice; napájení od 3 do 5V, nebo ±2,5V; 26 bajtů paměti pro uživatelská data; načtení předchozího nastavení z paměti při zapnutí.

Hlavní popis – AD5235 je dvoukanálový digitální potenciometr s permanentní pamětí a s 1024 pro nastavení jeho hodnoty. Zařízení vykonává stejné elektronické funkce jako mechanický potenciometr s rozšířeným nastavením, stejnou spolehlivostí a s nižším teplotním koeficientem.

U potenciometru lze naprogramovat data přímo do registru RDAC, čímž se přímo nastaví hodnota odporu mezi vývody W-A a W-B. Toto nastavení se uloží do paměti EEMEM a je znovu vloženo do registru RDAC po opětovném připojení napájení na potenciometr.

Obsah paměti je vyčten dynamicky nebo pomocí externího resetu na $\overline{PR}$. Nastavení jezdce potenciometru o lineární krok nahoru či dolů lze udělat v jediném strojovém cyklu. Pro

logaritmické kroky o 6dB je potřeba provést nastavení ve dvou strojových cyklech. Tolerance, o kterou se může lišit výsledná resistence je uložená v paměti EEMEM. Aktuální hodnota konečné velikosti odporu je také uložena v paměti, proto uživatel může nastavovat vhodný krok, tak aby odpovídal požadované hodnotě resistence potenciometru.



Obr. 7.2: Rozvržení pinů digitálního potenciometru

Číslo pinu	Název pinu	Popis
1	**CLK**	Vstup pro hodinový signál
2	**SDI**	Vstupní data. Při pozitivním nastavení CLK se zapíše jeden bit dat. První se zapíše bit s nejvyšší významovou hodnotou (MSB).
3	**SDO**	Datový výstup, slouží ke čtení nastavených dat na potenciometru.
4	**GND**	Zem.
5	**V_{SS}**	Záporné napájení, pokud použijeme napájení 0-5V, tak bude připojen na 0.
6	**A1**	Svorka A pro RDAC1.
7	**W1**	Svorka jezdce pro RDAC1.
8	**B1**	Svorka B pro RDAC1.
9	**B2**	Svorka B pro RDAC2.
10	**W2**	Svorka jezdce pro RDAC2.
11	**A2**	Svorka A pro RDAC2.
12	**V_{DD}**	Kladné napájecí napětí.
13	**WP**	Ochrana proti nechtěnému zápisu do paměti EEMEM.
14	**PR**	Ochrana proti nechtěnému zápisu do paměti EEMEM.
15	**CS**	Zápis a čtení z potenciometru může probíhat pouze pokud je CS v log 0.
16	**RDY**	Tento pin indikuje, kdy jsou dokončeny instrukce 2, 3, 8, 9, 10 a PR.

Tab. 7.1: Významy jednotlivých pinů AD5235

Ovládání potenciometru - AD5235 je navržen tak, aby pracoval jako opravdový potenciometr. Hodnota jezdce digitálního potenciometru je dána hodnotou, která je zapsána do registru RDAC. Nastavení potenciometru je provedeno pomocí sériového rozhraní SPI.

Nastavení se provádí na základě 24 bitů, první 4 bity jsou příkazy, další 4 jsou adresy a posledních 16 jsou data. Nastavení potenciometru se ukládá do paměti EEMEM, tento zápis trvá zhruba 25ms a odebírá zhruba 35mA, během tohoto času je posuvný registr RDAC zablokován. Až když je pin RDY nastaven do log 0, tak je zápis dokončen a je možné provézt další nastavení.

Rozdělení funkcí:

0 – nedělej nic.
1 – nastavení RDAC podle EEMEM.
2 – vložení RDAC do EEMEM.
3 – nastavení RDAC podle uživatelských dat v EEMEM.
4 – snížení o 6dB.
5 – snížení všeho o 6dB.
6 – snížení o 1 krok.
7 – snížení všeho o jeden krok.
8 – reset, obnovení všeho, podle uložených dat v EEMEM.
9 – odesílání dat z uživatelské paměti
10 – načtení dat do RDAC z SDO v dalším rámci
11 – zapiš hodnoty z D0 až D9 do RDAC.
12 – inkrementace o 6dB
13 – inkrementace všeho o 6dB
14 – inkrementace o 1 krok.
15 – inkrementace všeho o 1 krok.

Z výše uvedených funkcí jsou vytvořeny knihovny ovládající funkce 2, 4, 5, 6, 7, 12, 13, 14, 15. Veškeré programy jsou v příloze A, podle komentáře u nich lze jednoduše poznat o jakou funkci se jedná.

7.1.4 Popis sériového rozhraní SPI [1]

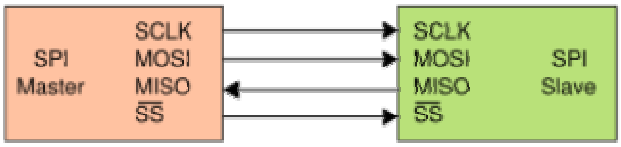
Rozhraní zajišťuje vysokorychlostní přenos dat mezi mikrokontrolérem a zařízením které je vybaveno SPI sběrnicí.

Schopnosti SPI:

- Plný duplex (schopnost současně přijímat i vysílat).
- 3vodičový synchronní přenos dat.
- Může pracovat jako master (řídící obvod) nebo slave (řízený obvod).

- 4 programovatelné rychlosti, maximální 250kb/s.
- Lze volit pořadí bitů (LSB až MSB a naopak).
- Příznaky konce a kolize přenosu.

Způsob propojení mezi masterem a slavem je na obrázku 7.3. CLK je hodinový signál. Odesílání dat se odstartuje, když jsou data zapsána do datového registru SPDR. Z masteru jsou data vysouvána vývodem MOSI a slavem jsou na stejném přijímána. Po vysunutí celého bajtu se hodinový generátor zastaví a nastaví se příznak konce přenosu SPIF.



Obr. 7.3: Propojení master – slave u SPI

Vývod SS je použit k detekování, pokud je nastaven do log „0“ je možno vysílat a přijímat data, pokud je nastaven do log „1“ lze vývod MOSI použít jako normální vstup.

SPI kanál je ovládán třemi registry:

- SPCR je řídící registr (rychlost a formát přenosu),
- SPSR je stavový registr (příznaky dokončení a kolize),
- SPDR datový registr (vstupní a výstupní data),

- **SPCR:** obsahuje 8 bitů, které konfigurují schopnosti SPI kanálu
  1. **SPIE** – povoluje přerušení SPI. **SPIE**=0 přerušení zakáže.
  2. **SPE** – aktivuje SPI kanál. SPI je aktivováno pro **SPE**=1, pak jsou aktivovány i vývody PB4 až PB7.
  3. **DORD** – určuje pořadí bitů při přenosu dat. Je-li DORD=1, tak začíná přenos nejnižším bitem LSB.
  4. **MSTR** – pokud je nastaven **MSTR**=1, tak je mikrokontrolér nastaven jako master.
  5. **CPOL** – volí polaritu hodinového signálu **CLK**.
  6. **CPHA** – určuje fázi hodin. **CPHA** a **CPOL** řídí vzájemné vztahy hodin a dat mezi masterem a slavem.
  7. **SPR1, SPR0** – volí kmitočet **CLK** (Tab. 7.2).

SPR1	SPR0	CLK	CLK pro f_0=1MHz
0	0	$f_0/4$	250kHz

0	1	$f_0/16$	62,5kHz
1	0	$f_0/64$	15,625kHz
1	1	$f_0/128$	7,813kHz

Tab. 7.2: Volba přenosové rychlosti SPI

- **SPSR:** obsahuje 2 příznaky indikující stav SPI kanálu
    1. **SPIF** – příznak konce přenosu. Po dokončení přenosu je tento bit nastaven SPIF=1.
    2. **WCOL** – je příznak kolize zápisu. Nastaví se WCOL=1, pokud dojde k zápisu do datového registru během přenosu.
- **SPDR:** 8 bitový registr jehož bity jsou po zapsání do něj vysílány. Přijaté bity se na konci přenosu do tohoto registru zapíší.

Příklad zápisu dat do digitálního potenciometru je uveden zde, jde o funkci zvýšení odporu o jeden krok:

```
void init_SPI(void)
{
   DDRB = 0xFF;         //celý portB je výstupní
   SPCR=0x52;           //rychlost 15,625kHz, CPHA a CPOL=0
                        //master, aktivování SPI
}

void zvetsi_o_1_1(void)
{
clrb(PORTB,4);          //SS se nastaví do 0
SPDR=0xE0;              //inkrementace o 1 krok
delay2();               //čekej 2uS
SPDR=0xE0;              //nepodstatná data
delay2();               //jde o naplnění 24 bitového
SPDR=0xE0;              //registru
delay2();
setb(PORTB,4);          //ukončení přenosu
delay4k();
clrb(PORTB,4);
delay4k();
```

```
setb(PORTB,4);
delay();              //čekej 1S
}
```

8 Závěr

Hlavním cílem bakalářské práce bylo seznámení se s ovládáním použitého procesoru (AVR ATmega324p), dalším cílem bylo navrhnout a zároveň realizovat zařízení, které bude sloužit jako ovládací prvek k testeru životnostních testů.

V práci jsou uvedena řešení všech HW částí zařízení a zároveň byla realizována sada knihoven, které slouží k ovládání jednotlivých prvků a funkcí zařízení, jako je ovládání digitálního potenciometru, ovládání A/D převodníku a sady relé, veškeré programy jsou v příloze A, kde jsou programy zřetelně popsány. Zařízení komunikuje s PC, kam mohou být odesílána veškerá potřebná data. Je zde také možnost ovládat jednotlivé funkce z klávesnice, ale spíše se bude zařízení nastavovat tak, aby běželo v určitém cyklu, který bude zařízení testovat, aniž by bylo potřeba jej nějakým způsobem ovládat.

Jako digitální potenciometr byla použita součástka AD5235, její odpor není příliš stálý, jeho hodnota se neustále mění v řádu několika desítek ohmů, proto pokud bychom chtěli simulovat čidlo, které má malý teplotní součinitel odporu, museli bychom použít digitální potenciometr, který má nižší maximální hodnotu odporu, v našem případě je to 250kΩ. Navíc potenciometr dosahuje na prvním potenciometru maximální hodnoty 135kΩ a minimální 298Ω a druhý maximální 127kΩ a minimální 297Ω.

V programu AVR Studio4 došlo pravděpodobně k chybě v pojmenování registrů a jejich jednotlivých bitů u sběrnice SPI, veškerá jména jsou uvedena s 0 na konci, ale správný zápis je bez 0 (SPCR0 – SPCR). Je třeba na správné pojmenování dbát, jinak program nebude fungovat.

Seznam literatury

[1] MATOUŠEK, David: *Práce s mikrokonroléry ATMEL AVR, BEN – technická literatura, Praha 2003*

Prameny

[2] Digitální potenciometr.
http://www.elektrorevue.cz/clanky/02050/index.html

[3] Nápověda programu AVR Studio4.

[4] Datasheet digitálního potenciometru AD5235.
http://www.analog.com/en/prod/0,2877,AD5235,00.html

[5] Program Hercules
http://www.hw-group.com/products/hercules/index_cz.html

[6] Datasheet obvodů 7805 a 7910.
http://www.gme.cz/_dokumentace/dokumenty/330/330-058/dsh.330-058.1.pdf

Seznam použitých zkratek

A, B	vývody digitálního potenciometru
A/D	analogově digitální
ALU	aritmeticko-logická jednotka
AREF	referenční napětí
AVCC	napájecí napětí mikrokontroléru
CD	kompaktní disk
EEMEM	paměť digitálního potenciometru
E^2 PROM	datová paměť
f_0	frekvence oscilátoru mikrokontroléru
GND	zem
HW	fyzická část
I^2C	multi-masterová počítačová sériová sběrnice
ISP	sériové programování přímo v aplikaci, 3 vodičové (plus napájecí vodiče)
JTAG	standard definovaný na programováni Flash paměti, 4 vodičové
LED	elektroluminiscenční dioda
MIPS	milion instrukcí za sekundu
PA až PD	port A až port D
PC	osobní počítač
PR	přenosová rychlost
PWM	pulsně šířkovou modulaci
RAM	operační paměť
RDY	indikace dokončení instrukce
RS-232	standart rozhraní pro sériovou komunikaci
RTC	obvod reálného času
RxD	přijímací vývod
SDI	vstupní data
SDO	datový výstup
SPI	sériové periferní rozhraní
TCP	Transmission Control Protocol jeden ze základních protokolů sady protokolů internetu
TxD	vysílací vývod
UART	synchronní a asynchronní sériové rozhraní

UDP	User Datagram Protocol jeden ze základních protokolů sady protokolů internetu
USB	universální sériová sběrnice
VCC	napájecí napětí
V_{DD}	kladné napájení
V_{SS}	záporné napájení
W	jezdec digitálního potenciometru
WP, PR	ochrana proti nechtěnému zápisu do paměti EEMEM

A Programové vybavení pro ATmega324p

```
// tester životnostních testů


#include <avr/io.h>                 //definice io registrů atd
#include <stdlib.h>                 //itoa(), ltoa()
                                    //(převodyčíslo-string)


#include "MyLib.c"                  //Zahrnutí potřebných funkcí
#include "AD.c"
//////////////////////////////MAIN//////////////////////////

int i=5, j=0;                       //i udává kolikrát má
                                    //proběhnout určitá
                                    //funkce
int main (void)
{
     init_AD();                     //inicializace
                                    //A/D převodníku
     init_uart(4800);               //inicializace UART 4800
                                    //baud
     init_SPI();                    //inicializace SPI
     DDRC = 0xFF;                   //nastaví port C jako
                                    //výstupní


     for(;;)                        //hlavní smyčka
     {
     AD();                          //změření napětí na vstupu

     Do                             //zvětšení odporu prvního
     {
     zvetsi_o_1_1();                //potenciometru o 5 kroků

     i--;
     }
     while(i>0);
     i=5;
     uloz1();                       //uložení aktuální hodnoty
                                    //prvního potenciometru
     negb(PORTC,0);                 //sepnutí/rozepnutí relé1
```

```
    do                              //zvětšení odporu druhého
    {                               //potenciometru o 5 kroků
    zvetsi_o_1_2();
    i--;
    }
    while(i>0);
    i=5;
    uloz2();

    negb(PORTC,1);                  //sepnutí/rozepnutí relé2

    do                              //zmenšení odporu prvního
    {                               //potenciometru o 5 kroků
    zmensi_o_1_1();
    i--;
    }
    while(i>0);
    i=5;
    uloz1();

    do                              //zmenšení odporu druhého
    {                               //potenciometru o 5 kroků
    zmensi_o_1_2();
    i--;
    }
    while(i>0);
    i=5;
    uloz2();
    }

}
//////////////////////////END MAIN//////////////////////////
//////////////////////////knihovna pro AD//////////////////
#include <avr/io.h>                 // definice IO registru atd.
#include <stdlib.h>                 // itoa() ltoa() (převody
                                    //int-str)
// Změna bitu portu, IO registru nebo proměnné:
#define setb(port,pin)    port |= 1<<pin    //nastav bit
#define clrb(port,pin)    port &= ~(1<<pin) //nuluj bit
#define negb(port,pin)    port ^= 1<<pin    //neguj bit

void init_AD(void);
void AD(void);
void init_uart (unsigned int baud);//inicializace
                                    //UART
unsigned char u_getc( void );       //přijmi znak z
                                    //UART
void u_putc( char data );           //napiš znak do
                                    //UART
void u_puts( char *text );          //pošli string
```

```
void AD(void)                        //změření vstupního napětí
{
char tempstr[8], znak;
int prevod=0, A=0, B=0, C=0, AktivVstup=0, PocTestVstup=0;
     do
     {
     ADCSRA=0xC3;        //nepovolene preruseni a jednoduchy
                         //prevod
     delay2();           //čekej 2uS
     prevod=ADCL;                    //cteni vysledku prevodu
     prevod= ADCL | (ADCH<<8);
     AktivVstup=ADMUX;               //ktery vstup je aktivni
     ADMUX++;
     PocTestVstup++;                 //pocet testu
     if (ADMUX==4) ADMUX=0;
     if (PocTestVstup==8) prevod=1; //aby program nebezel stale
                                     //pokud neni
     }                               //zadny vstup aktivni
     while(prevod==0);               //cekej na aktivni
                                     //vstup
     prevod--;
     PocTestVstup=0;


     if (AktivVstup==0)              //vyhodnocení vstupního
                                     //napětí
          {
          C=prevod/2;
          A=prevod/200;
          B=(C-(100*A));
          }
     if (AktivVstup==1)
          {
          C=prevod/2;
          A=prevod/20;
          B=(C-(10*A));
          }
     if (AktivVstup==2)
          {
          A=prevod/5;
          B=0;
          }
     if (AktivVstup==3)
          {
          A=(10*prevod)/25;
          B=0;
          }

// zapis celeho cisla:
     itoa(A,tempstr,10);     // číslo int1 převeď na string
                              //tempstr,
```

```
        u_puts(tempstr);        // a zapiš do UART
        u_putc(',');
        itoa(B,tempstr,10);     // číslo int1 převeď na string
                                //tempstr,
        u_puts(tempstr);           // a zapiš do UART
        u_putc('V');
        u_puts("\r\n");         // odřádkuj
        delay();

}


void init_AD(void)
{
        ADMUX=0x00;
        ADCSRA=0x00;        //nepovolene preruseni a jednoduchy
                            //prevod
        ADCSRB=0x00;
}

// UART pošli znak:
void u_putc( char data )
{
while ( !( UCSR0A & (1<<UDRE0)) );     // cekej do vyprázdnení
                                        //bufferu

UDR0 = data;
}
// UART pošli string:
void u_puts( char *text )
{
unsigned char i=0,temp;
        do
        {
        temp = text[i];
        if(temp==0) break;
        u_putc(temp);
        i++;
        }
        while(temp>0);
}

// UART přijmi znak:
unsigned char u_getc( void )
{
while ( !(UCSR0A & (1<<RXC0)) );      // čekej do přijetí dat
return UDR0;
}
```

```
////////////////////////////UART////////////////////////////
/////////////////////////inicializace/////////////////////////
void init_uart (unsigned int baud)
{
if(baud==1200) {UBRR0H=2;UBRR0L=  8;}
if(baud==2400) {UBRR0H=1;UBRR0L=  3;}
if(baud==4800) {UBRR0H=0;UBRR0L=12;}
if(baud==9600) {UBRR0H=0;UBRR0L=6;}
UCSR0B=0x18;    // příjem zapnut, vysílání zapnuto
UCSR0C=0x06;    // 8-bit, 1 stopbit, bez parity

}
////////////////////////////////////////////////////////////////
/////////////////////MyLib//////////////////////////////////
void delay(void);
void delay2(void);
void delay4k(void);
void init_SPI(void);
void posli(unsigned char B);
void zvetsi_o_1_1(void);
void zvetsi_o_1_2(void);
void zmensi_o_1_1(void);
void zmensi_o_1_2(void);
void zvys_vse_o_1(void);
void sniz_vse_o_1(void);
void zvetsi_o_6dB1(void);
void zvetsi_o_6dB2(void);
void sniz_o_6dB1(void);
void sniz_o_6dB2(void);
void zvys_vse_o_6dB(void);
void sniz_vse_o_6dB(void);
void uloz1(void);
void uloz2(void);
////////////////////////zpozdeni/////////////////////////
void delay(void)                  //1S
{
     unsigned int i=0xFFFF;
     while(--i);
}

void delay2(void)                 //2,5uS
{
     unsigned int i=0xFF;
     while(--i);
}

void delay4k(void)                //25nS
{
     unsigned int i=0x04;
     while(i--);
}
```

```
//inicializace SPI
void init_SPI(void)
{
     DDRB = 0xFF;        // cely portB je vystupni
     SPCR=0x52;          //zakazane preruseni, aktivovani SPI,
                         //odesilani od MSB, MASTER
                         //15,625kHz
}
////////////////////////////////////////////////////////
//////////////funkce pro odeslání//////////////////////
void posli(unsigned char B)
{
     clrb(PORTB,4);              //CS = 0
     SPDR=0xB;              //číslo funkce
     delay2();                   //cekej 2,5uS
     SPDR=0xB;              //tyto udaje nemaji vzynam
     delay2();                   //muzeme poslat cokoli
     SPDR=0xB;              //jde o naplneni 24 bitoveho
     delay2();              //ridiciho registru potenciometru
     setb(PORTB,4);         //ukonceni prenosu
     delay4k();
     clrb(PORTB,4);
     delay4k();
     setb(PORTB,4);
     delay();           //cekej 1S
}
//////////////funkce zvetsi o 1 krok první potenciometr/////
void zvetsi_o_1_1(void)
{
posli(0xE0);
}
//////////////funkce zvetsi o 1 krok druhý potenciometr/////
void zvetsi_o_1_2(void)
{
posli(0xE1);
}
//////////////funkce zmenší o 1 krok první potenciometr/////
void zmensi_o_1_1(void)
{
posli(0x60);
}
//////////////funkce zmenší o 1 krok druhý potenciometr/////
void zmensi_o_1_2(void)
{
posli(0x61);
}
//////////////funkce zvětší vše o 1 krok //////////////////
void zvys_vse_o_1(void)
{
posli(0xF0);
}
```

```
//////////////funkce zmenší vše o 1 krok ////////////////
void sniz_vse_o_1(void)
{
posli(0x70);
}
//////////////funkce zvětší o 6dB první potenciometr////////
void zvetsi_o_6dB1(void)
{
posli(0xC0);
}
//////////////funkce zvětší o 6dB druhý potenciometr////////
void zvetsi_o_6dB2(void)
{
posli(0xC1);
}
//////////////funkce zmenší o 6dB první potenciometr////////
void sniz_o_6dB1(void)
{
posli(0x40);
}
//////////////funkce zmenší o 6dB druhý potenciometr////////
void sniz_o_6dB2(void)
{
posli(0x41);
}
//////////////funkce zvětší vše o 6dB ////////////////////
void zvys_vse_o_6dB(void)
{
posli(0xD0);
}
//////////////funkce zmenší vše o 6dB ////////////////////
void sniz_vse_o_6dB(void)
{
posli(0x50);
}
//////funkce pro uložení nastavení prvního potenciometru////
void uloz1(void)
{
posli(0x20);
}
//////funkce pro uložení nastavení druhého potenciometru////
void uloz2(void)
{
posli(0x21);
}
```