TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Fluxmetr ovládaný mikrokontrolerem s rozhraním pro přípojení k PC

Bakalářská práce

Studijní program:	N2612 – Elektrotechnika a informatika
Studijní obor:	2612R011 – Elektronické informační a řídící systémy
Autor práce:	**Vojtěch Kuliš**
Vedoucí práce:	Ing. Miroslav Novák, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Fluxmetr ovládaný mikrokontrolerem s rozhraním pro připojení k PC

Jméno a příjmení:	**Vojtěch Kuliš**
Osobní číslo:	M15000101
Studijní program:	B2612 Elektrotechnika a informatika
Studijní obor:	Elektronické informační a řídicí systémy
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	**2018/2019**

Zásady pro vypracování:

1. Navrhněte koncepci přístroje fluxmetru s displejem a možností komunikace s počítačem pomocí SCPI příkazů. Předpokládá se použití mikrokontroleru, displeje s rozhraním, ovládacích tlačítek nebo jednoduché klávesnice, AD a DA převodníku a hotového modulu integrátoru s operačním zesilovačem.
2. Navržený systém sestavte a oživte.
3. Naprogramujte firmware fluxmetru, který bude obsluhovat vlastní měření. Umožní přepínání rozsahů integrátoru, jeho resetování, nulování driftu a umožní vzdálené měření z počítače.
4. Hotové zařízení otestuje.

Liberec 2019

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] PUNČOCHÁŘ, Josef. Operační zesilovače v elektronice. 5. vyd. Praha: BEN, 2002. ISBN 80-7300-059-8.

[2] DOSTÁL, Jiří. Operační zesilovače. 1. vyd. Praha: BEN, 2005. ISBN 80-7300-049-0.

[3] PINKER, Jiří. Mikroprocesory a mikropočítače: [obecné principy konstrukce současných mikroprocesorů a mikropočítačů]. Praha: BEN – technická literatura, 2004. ISBN 80-7300-110-1.

[4] VÁŇA, Vladimír. Mikrokontroléry ATMEL AVR: popis procesoru a instrukční soubor. Praha: BEN, 2003. ISBN 80-7300-083-0.

[5] MANN, Burkhard a Václav LOSÍK. C pro mikrokontroléry: ANSI-C, kompilátory C, spojovací programy – linkery, práce s ATMEL AVR a MSC-51, příklady programování v jazyce C, nástroje programování, tipy a triky, ... Praha: BEN – technická literatura, 2003, ISBN: 80-7300-077-6.

Vedoucí práce:	Ing. Miroslav Novák, Ph.D. Ústav mechatroniky a technické informatiky
Konzultant práce:	Ing. Martin Černík, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2018

Prohlášení

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

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom po-vinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

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

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: Podpis:

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce panu doktoru Novákovi za rady a tipy, které mi značně ulehčily práci. Také bych mu chtěl poděkovat za umožnění zprovoznění práce v jeho elektrotechnické laboratoři a za možnost využití zařízení, bez kterých by nebylo možné práci dokončit.

Abstrakt

Cílem této práce je navržení a sestavení prototypu přístroje fluxmetr. Fluxmetr je měřící přístroj, který měří sílu magnetického toku. Fluxmetr v této práci je řízen mikrokontrolerem. Mikrokontroler zprostředkovává zpracování signálu z analogového obvodu, převádění signálu pomocí AD a DA převodníků, zajišťuje nastavení vstupních konstant, umožňuje přepínání rozsahu, poskytuje výpočet výstupní hodnoty a její následné zobrazení. Tento systém se dá pomocí SCPI příkazů řídit přímo z PC. V práci je uveden návrh zařízení tak, aby docházelo k co nejmenší ztrátě signálu s ohledem na cenu a časovou náročnost.

Abstract

The objective of this work is to desing and build the prototype of magnetic fluxmeter. Magnetic fluxmeter is measuring device, which can measure magnitude of magnetic flux. Fluxmeter in this work is controlled by microcontroler, Microcontroller provides signal processing from analog circuit, converting of signal from AD and DA convertors, provide setting input constants, allows switching range, provides calculation of output value and its sunsequent display. This system can be controlled by SCPI commands from PC. The desing of the device is designed to minimize signal loss with respect to cost and time.

Obsah

Seznam tabulek

Seznam obrázků

Seznam zkratek a pojmů

µC	mikrokontrolér
OZ	operační zesilovač
M. Flux	magnetický tok
LSB	nejméně významný bit
MSB	nejvíce významný bit
CS	chip select
SPI	Serial Peripheral Interface
CLK	časování
CS_AD	chip select pro AD převodník
CS_DA	chip select pro AD převodník
log.	logická/é (ve smyslu logická úroveň)

Úvod

Tato práce navazuje na bakalářský projekt, ve kterém vznikla hlavní část fluxmetru, tedy analogového integrátoru. Ten se vždy před měřením musel připojit k měřící desce propojené s PC. To sebou neslo řadu nevýhod. Měření se muselo předem připravit. Algoritmy nastavení offsetu byly provizorní. Tyto a další nevýhody o dost prodloužily měření. Proto vznikl nápad na samostatný přístroj, který by dokázal obstarat měření a zároveň ho zrychlit.

Přístroj fluxmetr není z hlediska obsluhy nikterak složitý. Prvně je potřeba připojit periferii, tedy Helmholtzovu cívku, na které se naindukuje impulz magnetu. Plocha cívky a počet závitů cívky je nutné zadat do přístroje před začátkem měření. Než se začne měřit, je nutné vložit magnet do cívky.

Dále je před vytažením magnetu nutné eliminovat rušivé vlivy, zejména vliv offsetu. Pro konstantní hodnotu výstupu přístroj umožňuje nastavit napětí, vstupující do invertujícího vstupu OZ pomocí DA převodníku. V konečné fázi je také nutné zvolit rozsah. Rozsah je ovládán pomocí vhodné volby odporů integrátoru. Volbu odporů zajišťuje spínání pomocí relé.

Poté je nutné zmáčknout tlačítko reset, které vybije kondenzátor integrátoru a tím vynuluje hodnotu. Po vytažení magnetu mikrokontrolér okamžitě vypočítá hodnotu a ukáže ji na displej. Touto hodnotou může být magnetický tok, magnetická indukce, intenzita magnetického pole. Přístroj umí zobrazit intenzitu magnetického pole v anglosaské jednotce Oersted. Stejně tak umí zobrazit magnetickou indukci v anglosaských jednotkách Gauss. Proto se práce také detailně věnuje fyzikálním odvození vzorců, nutných pro výpočet těchto veličin.

Jedná se tedy o nastavení dvou konstant, nastavení kompenzace, zvolení vhodného rozsahu a požadované veličiny a konečně resetování integrátoru, vytažení magnetu a následné přečtení veličiny. Kdybych tento přístroj měl přirovnat k běžnému multimetru, je doplněn právě o nastavení konstant a kompenzaci s resetem. Z toho důvodu je k přístroji přistupováno velmi jednoduše. Pro zobrazení hodnot slouží běžný dvouřádkový displej a o zadání konstant se stará klávesnice.

Další možností je řízení fluxmetru pomocí počítače. K tomu slouží příkazy SCPI. Všechny důležité funkce je tedy možné řídit komfortně z PC, pomocí terminálu, jako je např. software Termite.

Tato práce se zaobírá právě realizací řízení výše uvedeného procesu. Zabývá se jak hardwarovou, tak softwarovou částí. Jelikož je softwarová část velmi obsáhlá, v práci uvádím pouze zásadní algoritmy. Různé pomocné proměnné a funkce v práci pro přehlednost uvedeny nejsou. Stejně tak práce pro přehlednost v práci nejsou uvedeny běžné hardwarové řešení, jako je například napájení.

1 Princip fluxmetru a odvození vzorců

Práce se zaobírá především programem a převodem signálu. Pro správnou funkci fluxmetru je nutné zadat vstupní konstanty, aby celé měření mělo smysl. Vše začíná u Faradayova indukčního zákona

$$U_i = -\frac{d\Phi}{dt}$$

který nám říká, že indukované napětí U_i se rovná záporné derivaci magnetického toku.
V našem případě se bude jednat o magnet vytahovaný z Helmholtzovi cívky. Je tedy vhodné upravit Faradayův zákon do následující podoby.

$$U_i = -N\frac{d\Phi_p}{dt} - N\frac{d\Phi_0}{dt}$$

Tento vzorec počítá, že se napětí indukuje na cívce s N závity. Na následujícím vztahu je dobré si uvědomit, že celkový magnetický tok $\Phi_c = \Phi_p + \Phi_0$, kde ϕ_p je proměnný magnetický tok a ϕ_0 je magnetický tok v ustáleném stavu. Pokud s magnetem nehýbeme, stále je v jeho okolí magnetické pole, tedy i magnetická indukce a potom v určené ploše i konstantní magnetický tok ϕ_0. Derivace konstanty je nula a proto vztah $-N\frac{d\Phi_0}{dt}$ úplně zmizí. V praxi to znamená, že jsme tímto způsobem schopni získat magnetický tok magnetu jen v případě, že se s ním bude pohybovat v okolí cívky.

Dále uvedu vzorec pro Millerův integrátor. Millerův integrátor je hlavní nejdůležitější elektronický obvod analogového integrátoru. Slouží ke zpracování (zintegrování) nestálého signálu Helmholtzovi cívky na určitou hladinu napětí. Díky napěťovému výstupu se signál dá jednodušeji zpracovat AD převodníkem.

$$U_{out} = -\frac{1}{\tau}\cdot\int_0^t U_{in}dt$$

U_{out} je výstupní napětí na výstupu integrátoru, τ je časová konstanta daná součinem R·C, kde R je odpor integrátoru a C je kapacita kondenzátoru Millerova integrátoru. U_{in} je potom vstupní napětí, které přivedeme na svorky integrátoru a t je doba, po kterou toto napětí budeme přivádět. Pokud by napětí U_{in} bylo po dobu t konstaní, můžeme po zintegrování n napsat, že

$$U_{out} = -\frac{1}{\tau}\cdot(U_{in}\cdot t + U_0)$$

Z toho plynou hned dvě věci. Za prvné U0 je napětí, které je již „naintegrováno“. Toto napětí se objeví na svorkách kondenzátoru. Proto je analogový integrátor vybaven resetovacím relé, který vybije kondenzátor C. Před každým měřením je tedy nutné vybít tento kondenzátor. Když bude kondenzátor vždy vybitý, můžeme tento člen vynechat. Druhá věc je právě zbylý člen U_{in}·t. Při rozměrové analýze si všimneme, že výsledek vychází v V·s. Poté mohu napsat

$$[V]\cdot[s] = [Wb]$$

$$\int_0^t U_{in} dt = \Phi = \Phi_p$$

Výsledek integrátoru je tedy magnetický tok násobený konstantou. Pokud si vezmu vstupní napětí dané vztahem

$$U_i = -N \frac{d\Phi_p}{dt}$$

Po úpravě

$$\int_0^t U_i \, dt = -N \cdot \Phi_p$$

$$-\frac{1}{N} \int_0^t U_i \, dt = \Phi_p$$

Tedy pokud bych chtěl dosazovat do Millerova vzorce pak

$$-\frac{1}{N} \int_0^t U_i \, dt = \int_0^t U_{in} dt$$

$$-\frac{1}{N} (U_i \cdot t + U_{i0}) = U_{in} \cdot t + U_0$$

U0 viz. výše je nulové. Napětí Ui0 by mělo být napětí naindukované na cívce, a to by mělo být v době měření taktéž nulové. Za těchto okolností mohu napsat:

$$-\frac{1}{N} U_i \cdot t = U_{in} \cdot t$$

$$-\frac{1}{N} U_i = U_{in}$$

Po dosazení do Millerova vzorce je pak

$$U_{out} = -\frac{1}{\tau} \cdot -\frac{1}{N} \int_0^t U_i \, dt$$

$$U_{out} = \frac{1}{N\tau} \int_0^t U_i \, dt$$

$$U_{out} = \frac{1}{N\tau} \Phi_p$$

Z toho: $$\Phi_p = U_{out} \cdot N \cdot \tau$$

Jelikož je potřeba měřit i magnetickou indukci lze si ji snadno vyjádřit ze vztahu,

$$B = \frac{\Phi_p}{S} = \frac{U_{out} \cdot N \cdot \tau}{S}$$

kde S je plocha cívky a dále je potřeba zobrazit i intenzitu magnetické pole H, které je rovno

$$H = \frac{B}{\mu} = \frac{U_{out} \cdot N \cdot \tau}{S \cdot \mu}$$

V cílech zadání je zobrazovat magnetickou indukci i v jednotkách Gauss a intenzitu magnetické pole i v jednotkách Oersted. Tyto jednoty se totiž stále v některých zemích používají. Pro převod platí

$$1T = 10\,000G$$

$$1A/m = 0.013Oe$$

Do paměti µC musely být tedy zapsány převodní konstanty pro anglosaské jednotky a kapacita kondenzátoru. Hodnota odporu závisí na právě zvoleném rozsahu. Dále musel být program doplněn o nastavení převodní konstanty cívky a o nastavení plochy cívky S.

2 Hardwarová realizace

2.1 Vývojový kit EvB 5.1

2.1.1 Co je to vývojový kit a proč byl v práci použit

Výsledek práce má být pouze prototyp zařízení. Kdybych místo vývojového kitu rovnou navrhoval vlastní zařízení, neskutečně by vzrostla časová náročnost práce. Dále by také rostly náklady na vývoj zařízení. Další problém by vznikl v případě, že by finální zařízení nefungovalo správně. Každá změna na zařízení by si žádala velké množství času a financí. I malá změna by mohla mít za následek objednávku nové DPS. Tomu dokáže předejít právě vývojový kit. Vývojové kity jsou osazeny běžně používanými prvky, se kterými se v elektronice nejčastěji setkáváme. Vývody kitů jsou uzpůsobeny pro rychlé zapojování/přepojování signálů. Zařízení obsahuje i různé ochrany proti zkratu atd.

Nevýhodou kitů je, že téměř vždy nevyužijeme všechny součástky. Kdyby se tedy kit použil v sériové výrobě, platilo by se i za součástky, které se nikdy nevyužijí. Navíc pro snadnou

manipulaci jsou součástky větších rozměrů. Díky konektorům prakticky k jakémukoliv vývodu se rozměry ještě zvětšují. Toto provedení je navíc citlivější vůči negativním rušivým vlivům. Je tedy na místě použít kit jen pro výrobu prototypu a jeho odladění. Pro výrobu více kusů je nutné zhodnotit finanční náročnost a místo kitu zhotovit vlastní zařízení.

K výrobě prototypu se dá použít i nepájivé či pájivé pole. Vývojový kit je ale odolnější vůči vnějším vlivům. Lepší kity jsou vybaveny různými typy ochran jako je např. ochrana proti zkratu atd. Celkově se s vývojovými kity i lépe manipuluje. Nevýhodou je ovšem jejich cena. V dnešní době se tato cena pohubuje v řádek stokorun. Využití je tedy z časového hlediska vhodné i zde. Bohužel tato práce není výjimkou, velmi často dochází k nutnosti využit kombinaci kitu i nepájivého, případně pájivého, pole. I tak se ale práce zrychlí.

2.1.2 Proč vývojový kit EvB 5.1?

Na dnešním trhu se vyskytují i jiné vývojové kity. Velmi dominantní jsou vývojové kity platformy Arduino. Tato platforma má velkou podporu z hlediska periferií a hlavně SW podpory. Téměř každá periferie má již zhotovenou svoji knihovnu. Programování Arduina probíhá v prostředí Wiring. Wiring je program založený na jazyku C. Program a celá struktura je dělaná systémem DIY, kde se počítá i se zapojením široké veřejnosti. Tedy řešení spousty problémů je dohledatelné na fórech.

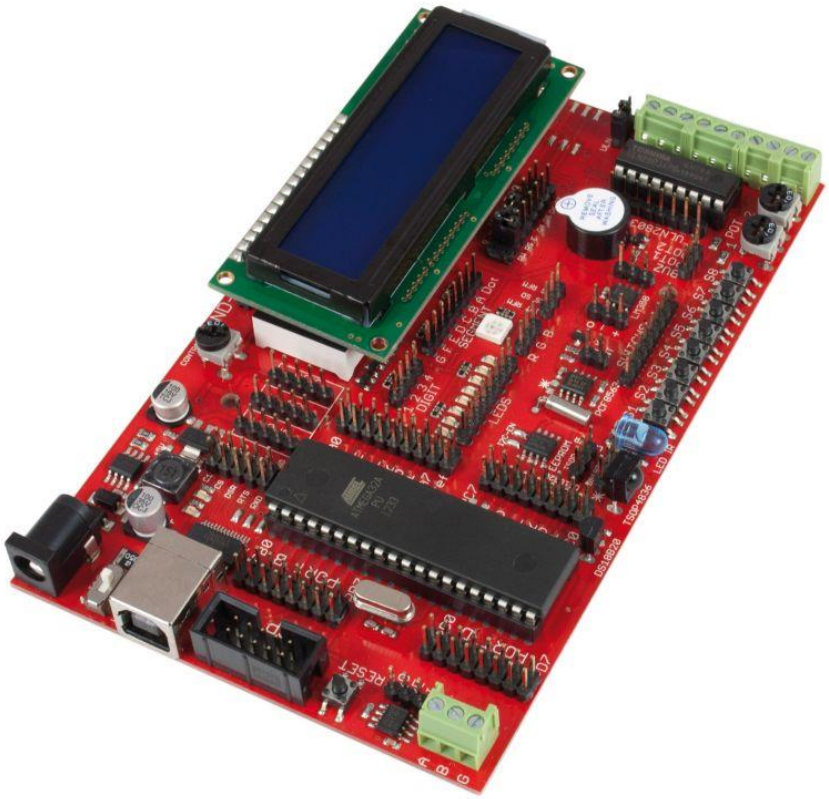
Oproti tomu EvB nemá ani zdaleka tak velkou podporu. Čip se programuje přes Atmel studio, které je taktéž na bázi jazyku C. Tento SW je dělaný přímo pro uC Atmel a architekturu avr. Programování spoléhá na oborovou gramotnost uživatele. Díky ne tak dobré podpoře jsem byl často nucen vytvořit si části kódu sám. Tento systém je však univerzální. Vývojové kity Arduino jsou taktéž vybaveny uC od Atmelu a lze je tedy programovat přes Atmel studio. Rozhodující faktor, proč jsem nakonec vybral kit EvB byly moje zkušenosti s touto platformou a vlastnictví tohoto zařízení.

2.1.3 Prvky kitu EvB 5.1

Vývojový kit je vybaven následujícími prvky:

- AVR ATMega16 uC (je možné osadit i silnějšími verzemi ATMega32 a ATMega644p
- Časovačem PCF8583
- Pamětí EEPROM AR24C02
- TSOP4836 infra-červeným přijímačem
- Teplotním čidlem DS18B20
- Sběrnicí RS485
- Pouzdrem na MMC/SD kartu
- 5 tlačítky
- 8x LED
- 2x tranzistorovými výstupy 1A každý
- 3x tranzistorovými výstupy 0,5A každý
- 2 analogovými potenciometry
- 4x7 segmentovým displejem

- USB portem
- ISP portem
- Pěti +5V piny
- Pěti GND piny
- LCD se 2x16 znaky

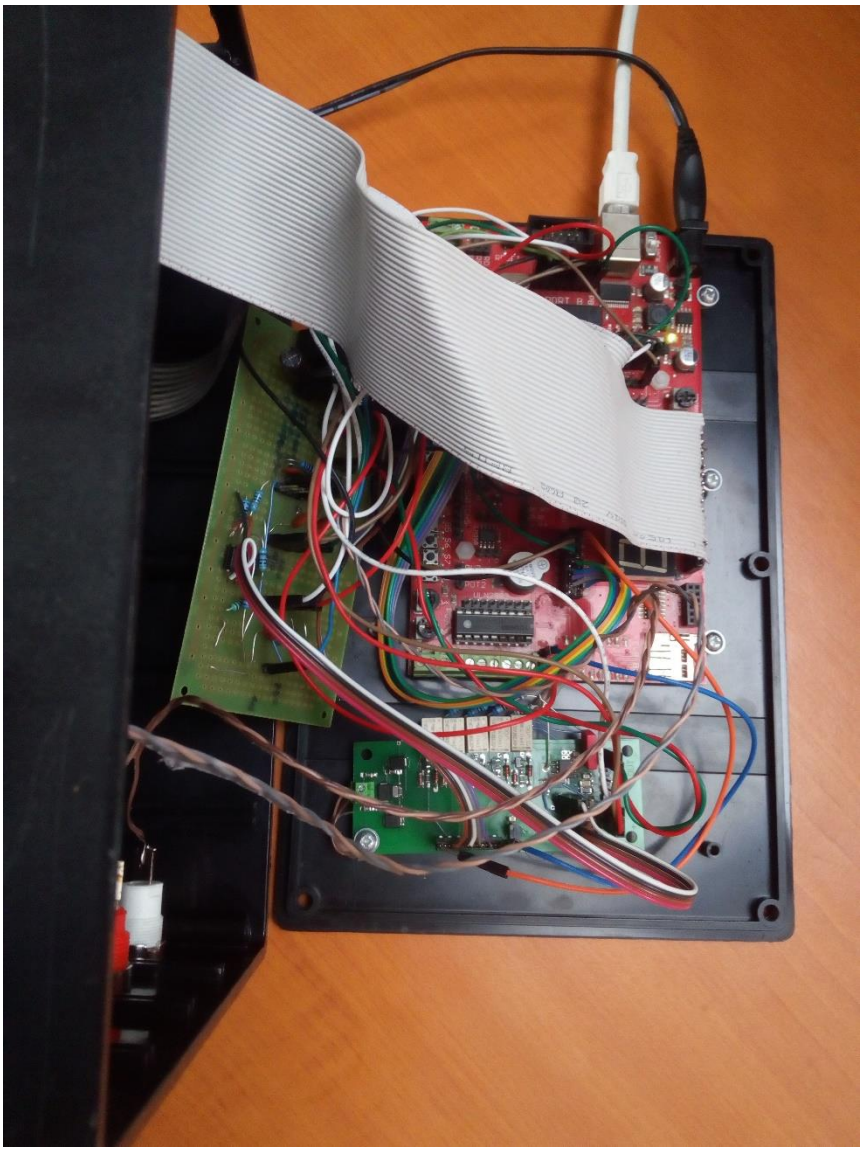


Obrázek 1 Vývojový kit EvB 5.1

2.2 Celková hardwarová realizace

Většinu prvků již obsahuje vývojový kit, nicméně některé periferie bylo třeba přidělat. Jedná se například o AD a DA převodník. K tomu bylo využito pájivé pole. Na toto pole bylo také umístěno napájení pro OZ. Pokud jde o napětí, byl použit DC/DC měnič se dvěma kanály +15V a -15V. Zařízení se dělí na tři části, jež můžeme vidět na Obrázek 2 vnitřní zapojeníObrázek 2 . První červená část vpravo je samotný vývojový kit. Vlevo můžeme vidět AD a DA převodníky včetně napájení. Dole se nachází analogový

integrátor. Vše je spojeno pomocí vodičů přes headery. Většinu vodičů jsem pro úsporu času dělal sám.



Obrázek 2 vnitřní zapojení

Realizace AD a DA převodníku je pro přehlednost uvedena u vysvětlení, a to v kapitolách Realizace AD Převodníku a Úprava signálu DA převodníku.

Celý vývoj byl uskutečněn v laboratoři vedoucího práce pana doktora Nováka. Čas byl tudíž omezený. Méně důležité součástky byly vybírány z dostupných zdrojů, nikoliv těch nejvhodnějších. Některé obvody byly navrhovány na místě.

3 Program

3.1 Úvod k programu

Možností, jak a v čem napsat program je vždy více. Program by měl plnit svou funkci co nejefektivněji. Tedy být co nejmenší a nejrychlejší. Toho jsem se snažil dosáhnout. Nejrychlejší a nejlepší řešení se občas stávalo zbytečně komplikovaným a složitým řešením po přidání další funkce.

Jelikož se jedná o čipy Atmel, vybral jsem si software přímo do výrobce. Jedná se o Atmel Studio 7.0. Ten dává na výběr ze dvou jazyků, a to Assembler a C/C++. Pro prototyp fluxmetru této práce je naprosto dostačující přehlednější jazyk C. Výhoda assembleru je v rychlosti a menší velikosti programu. V tomto případě je dobré zmínit možnost kombinace obou jazyků.

Jelikož se práce skládá z velké části z kódu, uvedu zde jeho části a nejdůležitější funkce. Ač důležité, ale přesto doplňkové části kódu zde nebudou nebo je pro přehlednost velmi zestručním.

3.2 Hlavní bloky programu

Mezi hlavní bloky programu patří:

```
int key();
void writeNumber_key();
void range();
void reset();
void offset();
void coil_threads();
uint32_t AD_SPI();
void DA_SPI();
void coil_surface_function();
char* fluxmetr_result();
void units();
void physical_quantity_set();
void print_display();

int main(void);
```

Funkcí key() a procedurou writeNumber_key() jsem se zabýval už v kapitole klávesnice. Funkce `AD_SPI()` je popsaná v kapitole Program AD převodníku . Podobně jsem popisoval proceduru `DA_SPI()` v kapitole Řízení DA převodníku. Proto se jimi v této kapitole zabývat nebudu.

Základní funkcí celého programu je `main(void)`. V hlavičce této funkce jsou deklarovány úvodní parametry globálních a lokálních proměnných. Dále jsou zde definovány log. základní úrovně pinů např. chipselecty do log. 1 značící deaktivaci AD a DA převodníků. Poté program najede do nekonečné while smyčky.

```
while (1)
    {

       LCD_goto(1,1);
       LCD_print("M. Flux:");
```

```
        DA_SPI();
        units();

        LCD_goto(2,1);
        LCD_print(fluxmetr_result());

        _delay_ms(5);

        switch(key())
              {
                    case 13:
                    range();
                    break;

                    case 14:
                    coil_threads();
                    break;

                    case 15:
                    coil_surface_function();
                    break;

                    case 9:
                    offset();
                    break;

                    case 10:
                    physical_quantity_set();
                    break;

                    default:;
                    break;
              }


        reset();

    }
}
```

Jelikož v této funkci muselo být uskutečněno měření, musela být velmi krátká. Nejdříve se zapíše na displej „M. Flux“ značící anglickou zkratku Magnetic Flux, tedy magnetický tok. Poté se na druhý řádek zavolá výpočtová funkce `fluxmetr_result()`, o které se zmíním později, a výsledek vypočteného magnetického toku se zapíše na druhý řádek displeje. Mezi tím se ještě zavolají procedury `DA_SPI()`, která nastaví DA převodník a procedura `units()`, která nastavuje předpony soustavy SI.

Na následujícím switchi je použita další výhoda vlastní knihovny klávesnice a podstata rozdělení funkce key() na funkci a proceduru writeNumber_key().Připomínám, že funkce key() vrací integer o rozsahu 0-16, s tím že 0 znamená „nic není zmáčknuto“. Původně bylo zamýšleno udělat pouze jednu funkci, která by řešila zápis znaků a v tomto switchi by byl jako argument použit char. Tím by se ale výpočet zbytečně prodloužil. Funkce by se také značně zkomplikovala. V metodě writeNumber_key() značím všechny nečíselné znaky N. Když pak vypisuji číselnou hodnotu, ptám se pouze jednou, zda-li byl zmáčknu znak N, tedy nečíselný.

V případě rozdílných charů bych se musel dotazovat na každý zvlášť, či hledat řešení rozpoznání čísla od písmene v typu char. Výsledkem by tedy byla zbytečná komplikace, kterou se tímto elegantně vyhnu.

Zmáčkne-li obsluha na klávesnici jedničku, program spustí metodu pro nastavení odporu integrátoru. Pokud obsluha zmáčkne čtyřku, spustí proceduru pro zadání offsetu. Pokud nezmáčkne nic, je vrácena nula. Tedy provede se to, co je pod default, takže nic.

3.2.1 Procedura offset

Jak funguje procedura a návrat zpět je dobře vidět na funkci `offset()`. Proto ji zde uvedu:

```
int charlenght;
	_Bool stop = 1;
	_delay_ms(button_delay);
	LCD_clear();
	LCD_goto(1,1);
	LCD_print("Offset:");
	_delay_ms(800);
	while(stop)
	{
		reset();
		writeNumber_key();

		LCD_goto(2,1);
		LCD_print(writekey_char_number);
		if(key()==12)		//potvrzeni cisla
		{
			//stop=0;
			offset_value = atof(writekey_char_number); //zapise nulu na
			zacatek
			if (offset_value>2.048)
			{

				char *a="2.048";
				strcpy(writekey_char_number,a);

				LCD_clear();
				LCD_goto(1,1);
				LCD_print("overflow");
				LCD_goto(2,1);
				LCD_print(writekey_char_number);
				_delay_ms(delay_confirm);
				LCD_clear();
				LCD_goto(1,1);
				LCD_print("Offset:");
				LCD_goto(2,1);
				LCD_print(writekey_char_number);
				counter=6;

			}
			else{
			LCD_clear();
			offset_value = atof(writekey_char_number);
			charlenght=strlen(writekey_char_number);
			writekey_char_number[0]='0';
			writekey_char_number[1]='\0';
```

```
                counter=1;
                if (charlenght>3)
                {
                        for (int i=2;i<charlenght;i++)
                        {
                                writekey_char_number[i]=(int)NULL;
                        }
                }
                LCD_goto(1,1);
                LCD_print("Confirmed");
                _delay_ms(delay_confirm);
                LCD_clear();
                stop=0;

                }
        }

        if(key()==16)
        {
                stop=0;
                charlenght=strlen(writekey_char_number);
                LCD_clear();
                counter=1;
                writekey_char_number[0]='0';
                writekey_char_number[1]='\0';
                if (charlenght>3)
                {
                        for (int i=2;i<charlenght;i++)
                        {
                                writekey_char_number[i]=(int)NULL;
                        }
                }
                LCD_goto(1,1);
                LCD_print("Canceled");
                _delay_ms(delay_confirm);
                LCD_clear();

        }

    }
}
```

Na začátku programu se smaže displej a zapíše se na něj „Offset:“ jasně indikující nastavení menu. Jelikož by se procedura okamžitě vrátila do zpět do main, je zde while a také nastavení lokální boolovské proměnné na log. 1. Dokud nebude v programu nastavena na do log. 0, program zůstane v této metodě. Následně je spuštěna funkce `writeNumber_key()`. Ta umožňuje zápis a mazání znaku do globální proměnné `writekey_char_number`. Tato globální proměnná se okamžitě zapíše na displej. Pokud je hodnota potvrzena stiskem A na klávesnici, globální proměnná `writekey_char_number` se přeloží na double a uloží se do globální proměnné. Před tím se pomocí několika řádků nuluje globální proměnná `writekey_char_number` kvůli využití v jiných funkcích. Následně je na displeji zobrazena hláška „confirmed“, která obsluhu informuje o přenastavení hodnoty. V případě, že by se obsluha rozhodla z různých důvodů hodnotu neuložit a celý proces stornovat, může zmáčknout na klávesnici A. To spustí prakticky totožnou funkci. Podstatným rozdílem oproti potvrzení je, že nedojde k přepsání hodnoty globální proměnné. Následně se také obsluze zobrazí hláška „Canceled“. Prakticky totožně jsou psány metody coil_threads() a coil_surface_function(). S tím

rozdílem, že přenastavují jiné globální proměnné a samozřejmě obsluha je informována na displeji, kterou z konstant nastavuje.

Na popis jsem si vybral metodu offset(), jelikož v ní je oproti konstantám potřeba kontrolovat, zde-li uživatel nezadal napětí větší než je DA převodník schopen zvládnout. Pokud tedy uživatel zadá napětí větší jak 2.048V a potvrdí výsledek, je informován o tom, že přesáhl maximální hranici hláškou „overflow“. Následně je větší číslo přepsáno maximální hodnotou DAC a tato max. hodnota je zobrazena na displeji. Obsluha si pak může vybrat, zda-li potvrdí nejvyšší hodnotu a nebo tuto hodnotu může pomocí mazání smazat a zadat jinou hodnotu offsetu.

Za povšimnutí stojí způsob, jakým způsobem bylo zapsáno do proměnné writekey_char_number. Jelikož C umí zobrazovat string pouze jako pole charů, v předchozím nulování a přepisování jsem k němu tak přistupoval. To je sice jasně definující styl zápisu, ale značně zdlouhavý. Proto jsem postupem času přešel na zápis stringu pomocí ukazatele a jeho uložení do jiného stringu pomocí knihovny string.h. Tento zápis je přehlednější. Pozor se ale musí dát na délku kopírovaných stringů. Jelikož místo v paměti tímto zápisem přímo neadresuji a nevím, kde je hodnota uložená, použitím této knihovny a porušením vstupních podmínek může tato knihovna přepsat hodnotu na dalším místě v paměti. To se pak může projevit zdánlivě nesmyslným přepisem proměnné třeba v úplně jiné funkci programu, což může být pro programátora dost matoucí a taková chyba se velmi špatně hledá.

3.2.2 Procedura physical_quantity_set()

Jak již bylo zmíněno, přístroj umí kromě magnetického toku změřit také magnetickou indukci B a intenzitu magnetického pole H. To dokáže i v anglosaských jednotkách Gauss a Oersted. Proto je zde procedura physical_quantity_set(), která slouží k nastavení těchto jednotek. Toto menu je pod tlačítkem č. 5 a obsluha pomocí tlačítek 1-5 může provést požadované nastavení.

```
while(stop)
	{
		reset();

		switch(key())
		{
				case 13:
				unit_storage=1;
				stop=0;
				print_display("O");
				break;

				.
				.
				.
				.
				.
				.
				.

				case 10:
				unit_storage=5;

				stop=0;
				print_display("Oe");
				break;
```

```
                case 16:
                stop=0;
                LCD_clear();
                LCD_goto(1,1);
                LCD_print("Canceled");
                _delay_ms(delay_confirm);
                LCD_clear();
                break;

                default:;
                break;

        }
```

V proceduře je naprosto klíčová funkce switche. Proto se v popisu omezím jen na ní. Tento switch funguje velmi podobně, jako výše popsaný switch v hlavní smyčce programu. V každém casu switche, s výjimkou zrušení, se uloží do globální jednobajtové bezznaménkové proměnné `unit_storage`. Tato proměnná bude posléze řídit ve funkci `fluxmetr_result()`výběr výpočtu žádané veličiny. Při potvrzení je se spustí procedura `print_display()`, která informuje obsluhu pomocí displeje jakou fyzikální jednotku zvolila. Tato procedura se nespouští pouze při zrušení. Zde je obsluha informována hláškou „canceled“. V případě zvolení jiné jednotky, či zrušení, se díky zastavení smyčky while pomocí proměnné stop program vrátí zpět do hlavní smyčky programu.

3.2.3 Procedura fluxmetr_result()

Druhou nejdůležitější funkcí celého programu je fluxmetr_result(). Zde probíhá výpočet a zde se také používají téměř všechny globální proměnné. Pro zjednodušení popisu uvedu pointu celé funkce. Zapsání na displej probíhá pomocí funkce knihovny *LCD_print*(). Vstupní parametr této funkce je string. Uživatele je potřeba obeznámit s tím, jaká veličina je měřena a s jakou předponou. Dále je potřeba převést číselný výsledek, na string a to celé je potřeba zřetězit a uvést na displej.

```
        int bitvoltagestorage=AD_SPI();

        if(bitvoltagestorage&0b00000000100000000000000000000000)
        {
                return "overflow low";
        }

        if (bitvoltagestorage&0b00000000001000000000000000000000)
        {
                return "overflow high";
        }
```

```
double calculation=((double)referencevoltage)*
((double)(bitvoltagestorage&0b00000000000011111111111111111111)/(double)2097152)-
((double)referencevoltage/(double)2); //vlozeni napeti do výpočtu
```

Než proběhne výpočet je nutné provést kontrolu přetečení. Pokud by bylo měřené napětí větší než referenční, převodník by ho správně nevyčíslil a vznikla by tak chyba. Kontrola přetečení tudíž má přednost před výpočtem. K tomu slouží dvě podmínky před výpočtem. Pomocí

maskování je separován overflow bit a pokud je v log. 1, pomocí return se obsluha informuje o přetečení. Program v tomto případě dál nepokračuje. V následujícím řádku je vypočítána hodnota napětí jako:

$$U = U_{refv}.\frac{AD_{out}\&\&M_{21\text{bits}}}{2^{22}-1} - \frac{U_{refv}}{2}$$

Kde U je výsledné napětí uložené jako double. U_{refv} je výstupní hodnota integrátoru. V našem případě ±10V, tedy rozsah 20V celkově. O převodu napětí se blíže zmiňuji v kapitole Realizace AD Převodníku. AD_{out} je 32. bitová informace od AD převodníku, M_{21bits} je maska, sloužící k nulování bitů od 32. do 22. bitu, pro získání 22. bitové informace.

Pozn. kvůli chybě v kapitole Realizace AD Převodníku je hodnota U_{refv} =22,5225V

Následující kód switche je jádrem všech výpočtů, získaných v kapitole Princip fluxmetru a odvození vzorců. Do těchto vzorců dosazuji proměnné. Pro přehlednost zde tyto proměnné shrnu. Do proměnné `Tau` jsem v proceduře `range()` uložil konstantu kondenzátoru a odporu integrátoru. `Proměnná coil_threads_number` obsahuje počet závitů cívky a lze ji změnit pomocí procedury `coil_threads()`. Proměnná `coil_surface` obsahuje plochu cívky a lze ji měnit pomocí procedury `coil_surface_function()`. Všechny tyto proměnné lze měnit také pomocí SCPI příkazů přímo z počítače.

```
switch(unit_storage)
{
        case 1: // výpočet a zápis pro magnetický tok

        calculation=Tau*coil_threads_number*calculation*
        pow(1000,exponent);
        unit="Wb";
        break;

        case 2://vypocet pro mag. indukci B
        calculation=(Tau*coil_threads_number*calculation
        *pow(1000,exponent)) /coil_surface;
        unit="T";
        break;

        case 3://vypocet pro intenzitu mag. pole H
        calculation=(Tau*coil_threads_number*calculation*pow(1000,exp
        onent)) /(coil_surface*(double)1.256637061*(double)(1e-6));
        unit="A/m";
        break;

        case 4:
        calculation=(Tau*coil_threads_number*calculation*(double)1e4
        *pow(1000,exponent))/coil_surface;
        unit="G";
        break;

        case 5:
        calculation=(Tau*coil_threads_number*calculation*(double)0.01
        3*pow(1000,exponent))/(coil_surface*(double)1.256637061*(doub
        le)(1e-6));
```

```
                unit="Oe";
                break;

                default:;
                break;

        }
```

Zde přichází konečně implementace proměnné unit_storage z menu nastavování jednotek z procedury physical_quantity_set(), kde si uživatel nastaví interpretaci fyzikální veličiny. Kromě výpočtu se také nastaví proměnná, uchovávající v sobě string požadované jednotky.

Následující switch je doplňkem navíc. Výsledek výpočtu je na displeji prezentován v exponenciálním tvaru. Obsluha si ale může pomocí klávesnice změnit předponu pro lepší interpretaci.

Pod tímto switchem následuje převod výsledku z double do stringu. Poté následuje zřetězení stringů v následujícím pořadí: String s číselným výsledkem, string s předponou, string s fyzikálním rozměrem a string s mezerou. Mezera slouží k mazání přebytečných znaků. Tato situace nastává např. v případě, že fyzikální rozměr předchozí veličiny vyžadoval více prvků v poli. Pokud tedy nedošlo k přetečení, funkce vrátí string jehož podoba vypadá nějak takto „1.2345E+01mWb“. Jelikož je funkce volaná z hlavní smyčky, pokud program není v jiném menu nebo pokud nedojde k přerušení od UART, se výsledek v každém cyklu programu přepočítá, zřetězí a zapíše na displej. Zařízení z pohledu člověka jako pozorovatele reaguje okamžitě, bez zásek.

```
switch(exponent)
{
        case 3: // výpočet a zápis pro magnetický tok
        exponent_char="n";
        break;

        case 2: // výpočet a zápis pro magnetický tok
        exponent_char="u";
        break;

        case 1: // výpočet a zápis pro magnetický tok
        exponent_char="m";
        break;

        case 0: // výpočet a zápis pro magnetický tok
        exponent_char="";
        break;

        case -1: // výpočet a zápis pro magnetický tok
        exponent_char="k";
        break;

        default:;
        break;

}
sprintf(result, "%.4E", calculation);
strcpy(result_string,result);
strcat(result_string,exponent_char);
strcat(result_string,unit);
strcat(result_string,"  ");
return result_string;
```

3.3 Ovládání a zobrazování hodnot fluxmetru

3.3.1 Displej s řadičem HD44780

Pro zobrazování fluxmetru byl vybrán běžný displej HD44780. Zde se nabízela možnost pojmout zařízení modernějším způsobem a použít barevný dotykový displej. V tomto případě je ale dobré si uvědomit funkci fluxmetru. Pokud se změní cívka, bude třeba před měřením zadat dvě konstanty. Před každým měřením je třeba nastavit offset. Poté se zvolí vhodný rozsah a může se měřit. Zároveň se nepředpokládá časté měření na přístroji. Zásadní přínos by byl tedy estetický a obsluha by měla při nastavování větší komfort. Na druhou stranu by se zvedla cena a časová náročnost celého projektu. Navíc zařízení se dá ovládat i z PC. To vedlo ke konečnému rozhodnutí použít naprosto dostačující displej s řadičem HD44780.

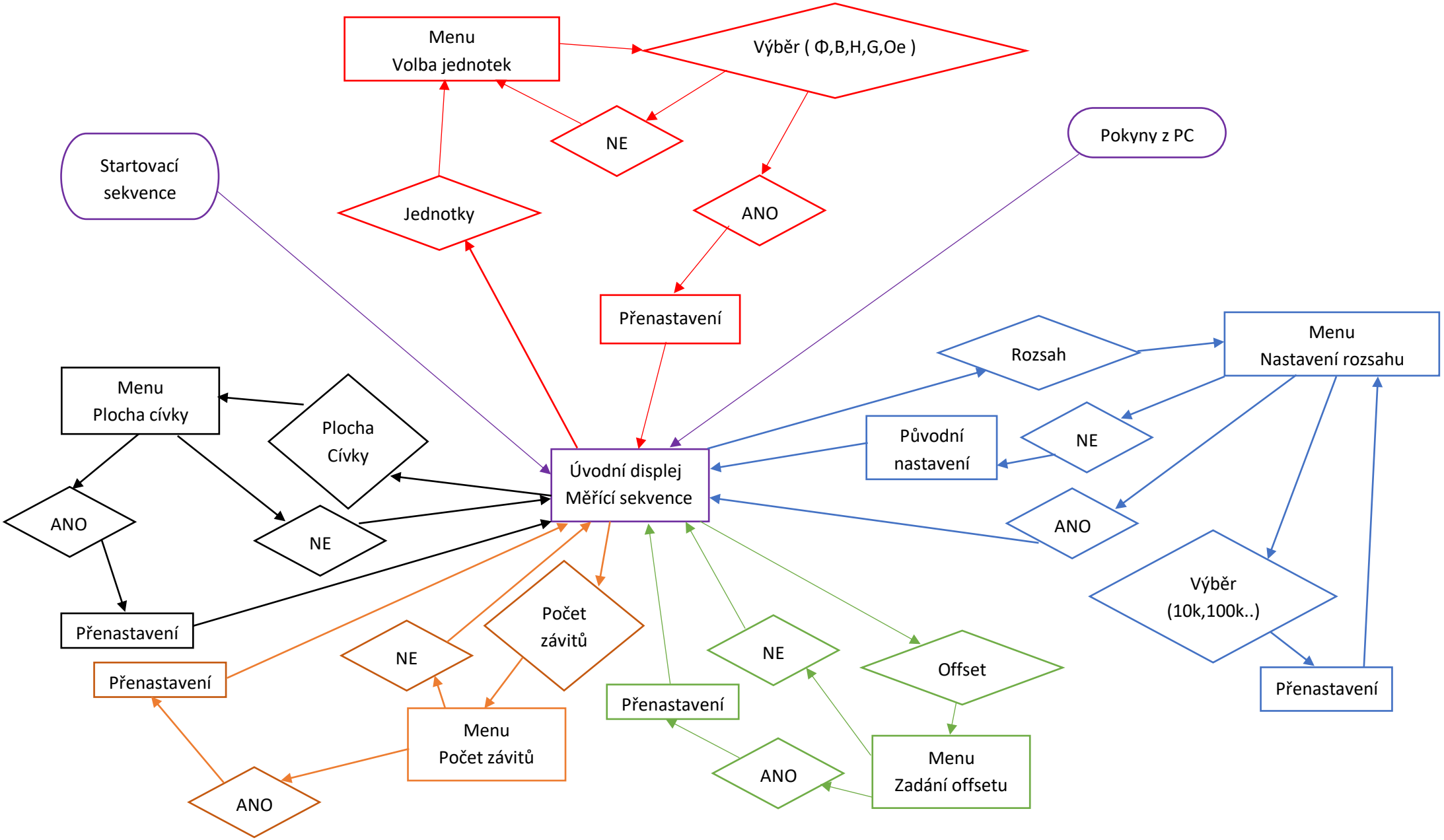
Pro fluxmetr byla nakonec zvolena dvouřádková verze displeje se 16 znaky a podsvícením. Napájení displeje je řešeno pomocí kitu. Ostatní piny jsou připojeny napřímo k µC. Velmi často se pro tento displej používá I2C konvertor, který zredukuje počet pinů na 4. Jelikož 32 pinů µC nebylo v práci využito, nebylo potřeba tento modul přidávat.

3.4 Klávesnice

Pro ovládání přístroje byla použita maticová klávesnice 4x4. Celá obsluha je řešená právě zde. Jde o výběr menu, potvrzování, zrušení, resetování, zápis konstant. Vše jde řídit pomocí této klávesnice.

3.4.1 Program klávesnice

Program klávesnice jsem rozdělil jako funkci key(), která vrací integer 0-16 a proceduru void writeNumber_key() jež řeší zápis znaku na displej a do paměti. Toto rozdělení zrychlí a zjednoduší program. Důvod je vidět v pozdější aplikaci této funkce v kapitole Program.


Menu
Volba jednotek
Výběr (Φ,B,H,G,Oe)
NE
Jednotky
ANO
Přenastavení
Startovací
sekvence
Pokyny z PC
Úvodní displej
Měřící sekvence
Menu
Plocha cívky
Plocha
Cívky
ANO
NE
Přenastavení
Rozsah
Menu
Nastavení rozsahu
Původní
nastavení
NE
ANO
Výběr
(10k,100k..)
Přenastavení
Počet
závitů
NE
Přenastavení
Menu
Počet závitů
ANO
NE
Offset
Přenastavení
Menu
Zadání offsetu
ANO

3.4.1.1 Funkce Key()

```
int key()
{
Keypad_port_set=0b00001111;// piny PB0-PB3 jako výstupy a piny PB4-PB7 jako
vstupy
Keypad_port|=0b11110000;   //aktivace pull up rezistorů pro piny PB4-PB7
int keynumber=0;
uint8_t setrow=0b11110111;
uint8_t maskcolumns=0b00010000;
uint8_t maskrows =  0b11110000;
for (int row=0;row<4;row++)
      {
                  Keypad_port=setrow|maskrows;
                  maskcolumns=0b00010000;
                        for (int columns=1;columns<5;columns++)
                        {
                              if ((Keypad_port_pin&maskcolumns)==0)
                              {
                              keynumber=row*4+columns;
                              return keynumber;
                              }
                              maskcolumns=maskcolumns<<1;

                        }
                        setrow=setrow>>1;
      }
      _delay_ms(button_delay);

      return keynumber;
}
```

Klávesnice má 4 řádky a 4 sloupce. V tomto případě jsou řádky připojeny na piny PBO-PB3 a jsou deklarovány jako výstupy. Sloupce jsou připojeny n piny PB4-PB7 a jsou deklarovány jako vstupy. µC je vybaven interními pull-up rezistory, které je možné aktivovat na každém pinu individuálně. Pro jasnou definici pinu jsou na vstupních pinech PB4-PB7 aktivovány. Zmáčknutím tlačítka na klávesnici se příslušný pin uzemní. Log. 0 poté jasně definuje jaké tlačítko je zmáčknuto.

Program funguje následovně. První for pouští postupně log. 0 do řádků. Druhý for poté testuje postupně každý sloupec. V případě, že tlačítko není zmáčknuto, je na pinu vždy hodnota z pull-up rezistoru, tedy log. 1. V případě, že je tlačítko zmáčknuto, je na něm log. 1 do té doby, dokud se k danému řádku nedostane první for. Ve chvíli, kdy se k němu dostane, je tento pin pomocí µC uzemněn a testování na daný výstupní pin splní podmínku. Pokud je podmínka splněna, je do proměnné `keynumber` zapsána hodnota aktuálního řádku a sloupce. Součástí podmínky je return, který zapříčiní ukončení dalšího testování. Tím se zaručí ochrana proti dvojímu stisku kláves, jelikož při nalezení prvního zmáčknutého tlačítka program dále nezjišťuje, zda je zmáčknuto i jiné tlačítko.

Co stojí za povšimnutí je hodnota návratové proměnné v pro znak „1“ na klávesnici a to 13. Program totiž musí začít pouštět log. 0 od spodu, nikoli od jedničky a první řádek je na klávesnici v programu ten poslední. Pokud by začal z vrchu, bylo by nutné na konci programu provést bitovou rotaci doleva a nikoli doprava. Tím by se ale na pozici LSB dostala log. nula a

v tomto případě by byly nastaveny dva řádky na logickou nulu. To v konečném důsledku uzemní všechny řádky. Když se potom ptám, jaký řádek je aktivovaný ve třetím sloupci, odpovědí může být „každý“. Tím pádem přicházím o jasnou definici. Řešení jsou dvě. Buďto najít způsob, který při bitové rotaci nahrazuje nový bit log. 1, nebo rotovat na druhou stranu. Zde využívám toho, že bity PB4-PB7 jsou nastaveny jako vstupy a je na nich pomocí zápisu pro výstup log. 1 nastaven pull-up. Po rotaci bude tedy na předchozí řádek zapsána log. 1 a nikoli log. 0. Abych nevypnul pull-up rezistory, musím ještě přičíst log. součtem na bity PB4-PB7 log. 1.

3.4.1.2 Procedura writeNumber_key()

Minulá funkce řešila pouze jaká klávesa je zmáčknuta, což je dostačující pro výběr menu, nebo u potvrzování hodnot. Tato procedura úzce navazuje na funkci key(). Procedura writeNumber_key() řeší přidávání číselných znakůvčetně tečky. Dosahuje toho pomocí zápisu do globální proměnné writekey_char_number[]. Tato proměnná se cyklicky zapisuje na displej. Při přidání, či odebrání znaku se tedy okamžitě přepíše tato proměnná. Proto nemůže být napsána jako funkce, ale jako procedura. Při potvrzení výsledku se v jiných procedurách z této globální proměnné převede string v ní uchovávaný na double a uloží se do paměti. Jelikož je tato proměnná využívána ve více procedurách, každá procedura, která využívá globální proměnnou writekey_char_number ji na svém konci smaže.

Tato procedura se na začátku nejdříve ptá. Je zmáčknutá klávesa? Pokud je, vrátí místo nulového integeru integer, s hodnotou 1-16. Podle něj se vybere z pole charů znak. Pokud tento znak je číselný, tedy není N, prohledá pomocí *strchr*() proměnnou writekey_char_number zda-li neobsahuje tečku. Pokud jí již obsahuje a vybraný znak je tečka, nedělá nic. Tím je vyřešená ochrana proti dvojitému zápisu tečky. Program se poté ještě ptá, zda-li nebyl překročen maximální počet znaků. Zápis do neexistujícího znaku pole způsobí zamrznutí µC, které je třeba řešit resetem. Pokud ani tato chyba nehrozí, zapíše vybraný znak do globální proměnné, podle ukazatele counter, což je globální proměnná uchovávající pozici dalšího místa v poli, určené pro následující znak.

```
char symbol[16]=
{'N','0','.','N','7','8','9','N','4','5','6','N','1','2','3','N'};

	if ((key())!=0)
	{

		char symbol_write[2] = {symbol[key()-1], '\0'};

		if (symbol_write[0]!='N')
		{

			if((strchr(writekey_char_number ,'.')!=NULL) &&
			(symbol[symbol_number]=='.'))
			{
			}
			else
			{
				if (counter<maxcharlenght)
				{
```

```
                        writekey_char_number[counter-
                        1]=symbol[symbol_number];
                        counter++;
                        _delay_ms(writekeydelay);
                }

        }


}
if (key()==1)
{
        if (writekey_char_number[1]=='\0')
        {
                _delay_ms(writekeydelay);
                writekey_char_number[0]='0';
                writekey_char_number[1]='\0';
                LCD_goto(2,1);
                LCD_print("                ");
                counter=1;

        }
        else
        {
        counter--;
        writekey_char_number[counter-1]='\0';
        LCD_goto(2,1);
        LCD_print("                  ");


        }
        _delay_ms(writekeydelay);
}
```

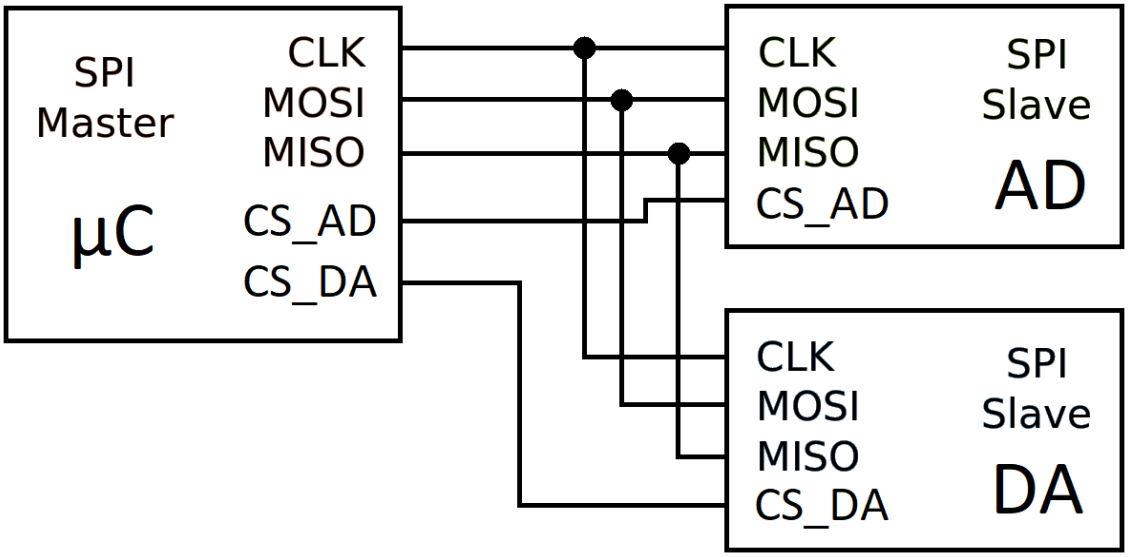
Dále je v této funkci řešeno mazání znaku. Pokud je zmáčknuta hvězdička a předposlední znak není ukončovací, do aktuálního místa v poli zapíše nulovací znak a dekrementuje pozici v poli (`counter`) o 1. Pokud je předposlední znak ukončovací, značí to mazání posledního symbolu. V tomto případě je místo mazání do `writekey_char_number` zapsána nula. Ta se poté okamžitě zobrazí uživateli na displeji. Za nulováním je potřeba přemazat poslední hodnotu na displeji, což se provádí zápisem mezery za proměnnou `writekey_char_number.`

3.5 SPI sběrnice

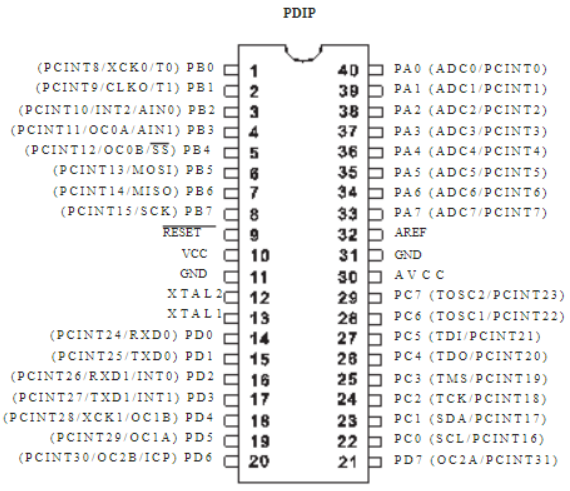
SPI je sériová sběrnice master-slave. Tato sběrnice se velmi často používá při komunikaci s externími moduly, jako jsou AD a DA převodníky, displeje atd. Proto mikrokontrolér avr umožňuje plně duplexní přenos. Nejvyšší rychlost je polovina taktu procesoru. Mikrokontroléry avr umožňují pracovat jak v roli mastera, tak v roli slave obvodu. Adresace na této sběrnici se provádí pomocí vodiče tzn. Chipselectu. Chipselect se značí obvykle CS, ale někdy také SS. Pomocí CS aktivuje master daný slave obvod. Platí, že sběrnice má pouze jednoho mastera a každý připojený slave musí mít vlastní chipselect, tedy i vodič. Dále má sběrnice linky MOSI, pro komunikaci Master -> Slave, linku MISO pro komunikaci Slave -> Master a CLK. CLK zprostředkovává časování mezi masterem a slavem. Po spuštění CLK začíná přenos dat po

sběrnici, pro daný slave aktivovaný CS. Pokud je deaktivovaný chipselect u slave, tak je linka MISO nastavená na vysokou impedanci. Parafrázování z [3] „vysoká impedance High-Z v tomto případě účinně odpojí účinky slave na sběrnici. Zaprvé toto řešení předchází zkratům a za druhé předchází kolizi log. 0 od slave obvodu č. 1 a log. 1 slave obvodu č.2 při komunikaci s µC“.

V případě této práce bude pomocí SPI řešena komunikace mezi mikrokontrolérem, AD a DA převodníkem. Jejich zapojení pomocí sběrnice SPI je na následujícím obrázku.



Obrázek 3 SPI komunikace



Obrázek 4 Pinout Atmega16

AVR mikrokontrolery mají HW uzpůsobení pro SPI a proto je potřeba připojit sběrnici na správné piny. Tyto piny jsem našel v datasheetu použitého µC. Piny jsou dále spojeny s ISP programátorem na vývojovém kitu. Těchto pinů bylo využito při nahrání bootloaderu.

3.5.1 Registry pro SPI

K řízení SPI slouží tři osmibitové registry. Registr pro manipulaci dat SPDR. To je v podstatě posuvný registr. Zápisem do něj se odstartuje přenos. Data k odeslání pro master se do něj zapíší a zároveň se posílají data do něho zapsané pro slave. Do tohoto registru se nesmí během přenosu zapisovat.

Pokud by se tak stalo, tuto informaci můžeme získat z SPSR (SPI Status Registr) registru. Přesněji díky bitu WCOL. Program nikdy do registru během přenosu zapsat nemůže, proto tento registr není v programu kontrolován. Nejdůležitější bit tohoto registru je SPIF. Tento bit se nastaví do log. 1 po přenesení celého bajtu programu. Čtením z něj se stavový bit nastaví na log. 0. Tento stavový bit je tedy pouze pro čtení. Pokud by byl CS mikrokontroléru nastaven

SPCR								
Bit	7	6	5	4	3	2	1	0
Jméno	SPIE	SPE	DODR	MSTR	CPOL	CPHA	SPR1	SPR0
Nastevení:	0	1	0	1	1	1	0	1

jako vstupní a přišla by na něj log. 0, znamenalo by to přepnutí do role slave a bit SPIF by se taktéž nastavil do log. 1. Jelikož je tento pin před hlavní smyčkou „natvrdo“ nastaven jako výstupní, SPIF se zaručeně bude nastavovat pouze při dokončení přenosu celého bajtu.

Nejvíce nastavování proběhlo v registru SPCR (SPI Control registr). Bitem SPE se povoluje SPI komunikace. Pokud by bylo potřeba vývody pro SPI použít i jinak, bylo by nutné SPI zakazovat. Pokud je nastaven bit SPIE, po nastavení bitu SPIF by se vyvolalo externí přerušení. To nebylo využito viz. kapitola AD převodník. Bit DODR určuje směr posuvného registru. V log. jedničce se nejprve odesílá LSB. Nastavení DODR je důležité hlavně pro programátora, aby věděl, jak má posléze data zpracovávat.

Tabulka 1 SPCR registr

Bity CPOL a CPHA se nastavuje průběh hodinového signálu. Leading edge v tomto případě znamená jeden puls ve tvaru obdélníku vedoucí z log. 0 do log. 1 a zpět. V případě trailing edge je to přesně opačně, tedy z puls ve tvaru obdélníku z log. 1 do log. 0 a zpět. Rising edge poté označuje náběžnou a Falling edge sestupnou hranu signálu. Toto nastavení bylo vybráno podle datasheetu AD a DA převodníku na třetí mód. Bity SPR1, SPR0 a SPR2 fungují pro nastavení rychlosti CLK, tedy nastavuje se tak rychlost přenosu. Rychlost přenosu se dá nastavit podle předdefinovaných hodnot stanovených jako podíl frekvence mikrokontroléru lomeno konstanta viz Tabulka 2 SPI mode/clock. Rychlost použitého µC je 16MHz. Tabulka uvádí pouze 4 hodnoty, ale díky bitu SPR2, který najdeme v registru SPSR se dají nastavit ještě další 4 rychlosti např. fosc/2. Obecně platí, že čím vyšší rychlost, tím vyšší je možnost ztráty informace. Proto jsem nenastavoval rychlost co nejvyšší, ale zvolil jsem rychlost na fosc/16, která je pro uživatelskou obsluhu nerozeznatelná od vyšších rychlostí. Jelikož práce nemá problém s náročnými výpočty, které by zdržovali rychlost měření, není potřeba rychlost

přenosu řešit do detailů a v práci jsem se tímto problémem dále nezabývám. Stejně tak neuvádím tabulku pro kompletní výpis všech nastavení registrů, ale pouze orientační.

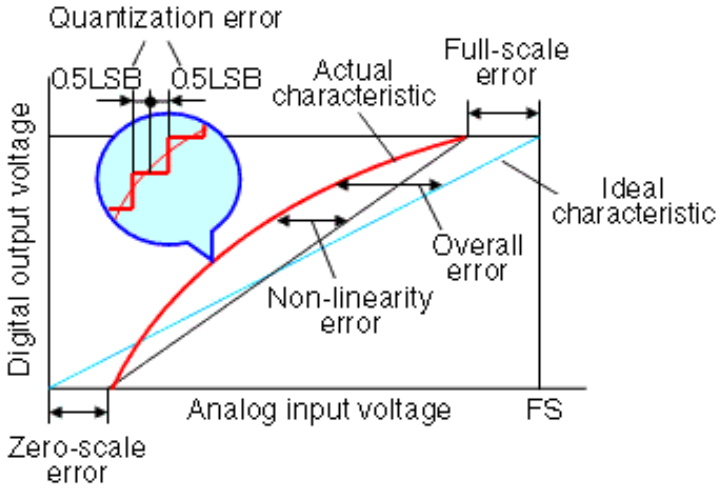
SPI Mode	CPOL	CPHA	Sample	SPR1	SPR0	SCK frequency
0	0	0	Leading (Rising) Edge	0	0	fosc/4
1	0	1	Trailing (Falling) Edge	0	1	fosc/16
2	1	0	Leading (Falling) Edge	1	0	fosc/64
3	1	1	Trailing (Rising) Edge	1	1	fosc/128

Tabulka 2 SPI mode/clock

Na závěr této kapitoly je dobré připomenou, že AD a DA převodník jsou od stejné firmy a mají stejné nastavení v registru SPCR. Tento registr je tedy možné nastavit na začátku programu a dále do něj není potřeba zapisovat.

3.6 AD převodník

A/D převodník v této práci zprostředkovává převod analogové hodnoty napětí U_{OUT} z výstupu integrátoru na digitální hodnotu, kterou je µC schopen zpracovat. Jelikož se jedná o AD převodník pro měřící přístroj, je důležité zohlednit přesnost AD převodníku. Přesněji jeho chyby. Mezi známé chyby AD převodníků patří chyba nuly, chyba linearity, chyba plného rozsahu a především kvantovací chyba. Kvantováním se vždy ztratí část informace. Tuto ztrátu lze minimalizovat výběrem převodníku s vysokým počtem bitů. Parafrázováno z [4].



Obrázek 5 Chyby AD převodníku [5]

Pro práci se jevil jako vhodný 22 bitový AD převodník MCP3551/3 od firmy Microchip. Byl vybrán především díky vyššímu počtu bitů, ale jeho díky nízkým chybám. Jeho chyba nuly je pouze 3µV, chyba plného rozsahu 2ppm, chyba nelinearity 6 ppm.

3.6.1 Aplikace AD převodníku

Převodník dokáže pracovat ve dvou režimech. Režim continous conversion a režim single conversion. Oba režimy komunikace probíhají po sběrnici SPI. Režim single conversion po definovaném signálu provede jednu konverzi a po přenesení všech dat se převodník přepne do režimu shutdown, tedy se vypne. V režimu continous conversion probíhá konverze neustále. Při čtení v režimu continous conversion se hodnota bufferu, kde se ukládá výsledek převodu v AD převodníku, nemění, dokud neskončí přenos. Program AD převodníku je volán v hlavní programové smyčce, která je časově nenáročná. Pro aplikaci AD převodníku je důležitá pouze koncová hodnota převodu. Samozřejmě za předpokladu, že integrátor samovolně nemění hodnotu měření např. špatně nastaveným offsetem. Proto není třeba volat AD převodník v přerušení od externího timeru. Z těchto důvodů lze využít oba mody. V případě této aplikace jsem využil modifikovaný single conversion mode.

3.6.2 Program AD převodníku

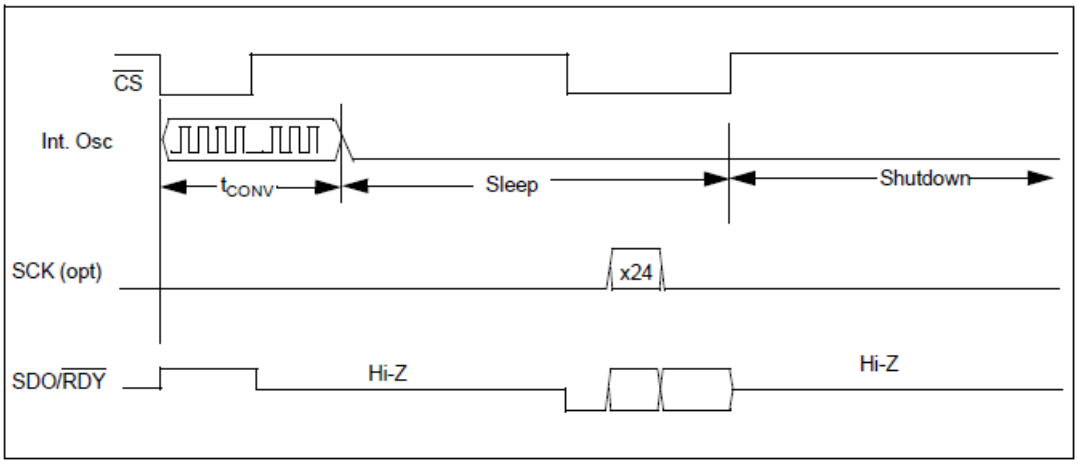


FIGURE 5-2: *Single Conversion Mode.*

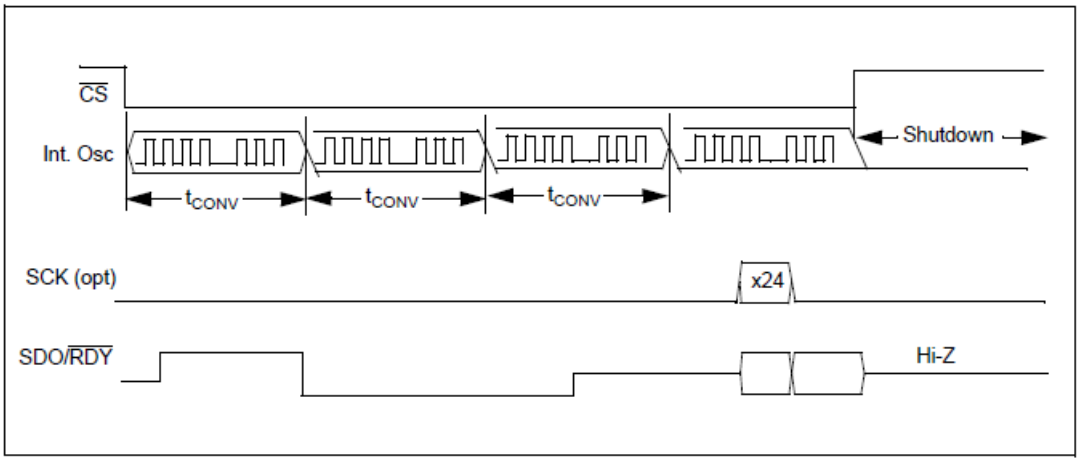


FIGURE 5-3: *Continuous Conversion Mode.*

Obrázek 6 Řízení AD převodníku

Na předchozím obrázku jsou vyobrazeny oba typy přenosů. Tyto přenosy jsou uvedeny v datasheetu AD převodníku.

```
uint32_t AD_SPI()
{
	CS_AD_L;
	_delay_us(11);
	if (((PINB &(1<< SPI_MISO))))
	{
		CS_AD_H;
		_delay_ms(17);
	}
	else
	{
		SPDR = 0;
		while(!(SPSR & (1<<SPIF)));

		AD_data=0;
		AD_data = (uint32_t)SPDR*(uint32_t)(65536);

		SPDR = 0;
		while(!(SPSR & (1<<SPIF)));
		AD_data = AD_data+(uint32_t)SPDR*(uint32_t)255;

		SPDR = 0;
		while(!(SPSR & (1<<SPIF)));
		AD_data = AD_data+(uint32_t)SPDR;

	}
	return AD_data;
}
```

Program je psán jako funkce, jejíž návratová hodnota je 32bitová proměnná, kterou je vždy potřeba přepočítat na interpretovatelnou veličinu. Tento výpočet se provádí v Proceduře fluxmetr_result(), kde je i popsán. Nejdříve program nastaví CS_AD tedy chipselect pro AD převodník do Low. To spustí konverzi. Ve chvíli, kdy převodník provádí konverzi z něj není možné číst. Informaci o stavu dává pinem RDY, který je součástí MISO. Log. 1 značí probíhající konverzi. Log. 0 Značí, že z převodníku je možné číst. Proto bylo nutné vytvořit podmínku. Pokud je převodník ready, podmínka nastaví CS do stavu log. 1. Tím se spustí režim single conversion viz. Obrázek 6 Řízení AD převodníku. Ještě je nutné dodržet minimální délku impulsu CS, kterou výrobce uvádí na 10µS. Proto je vložen 11µS delay. Poté je vloženo zpoždění 17ms. To je hodnota konverze uváděná v datasheetu. Jelikož cyklus programu trvá více jak 17ms, toto zpoždění není v této aplikaci nutné. V prvním cyklu je návratová hodnota hodnotou předchozí. Proto je potřeba návratovou proměnnou ukládat do globální proměnné. V každém volání funkce po předchozím impulsu na CS je zaručeno, že na pinu RDY je hodnota Hi-Z, tedy podmínka splněná není a přejde se k plnění „else“. Nulováním registru SPDR se spustí CLK a podmínka while čeká na signální bit SPIF, který signalizuje přenesení bajtu.

Hodnota jednobajtového registru SPDR je posunuta a následně uložena na danou pozici 32bitové návratové hodnoty.

Analog Input Voltage	OVL	OVH	Digital Output Code																				Decimal Code	Hexa
	B23	B22	B21	B20	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0		*B[23:0]*
Vref +1 LSB	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2097153	600001
Vref	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2097152	600000
Vref - 1 LSB	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2097151	1FFFFF
2 LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	000002
1 LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	000001
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	000000
-1 LSB	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	3FFFFF
-2 LSB	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	-2	3FFFFE
-Vref	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2097152	200000
-Vref -1LSB	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-2097153	9FFFFF
-Vref -2LSB	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	-2097154	9FFFFE

Tabulka 3 ADC Výstupní kód

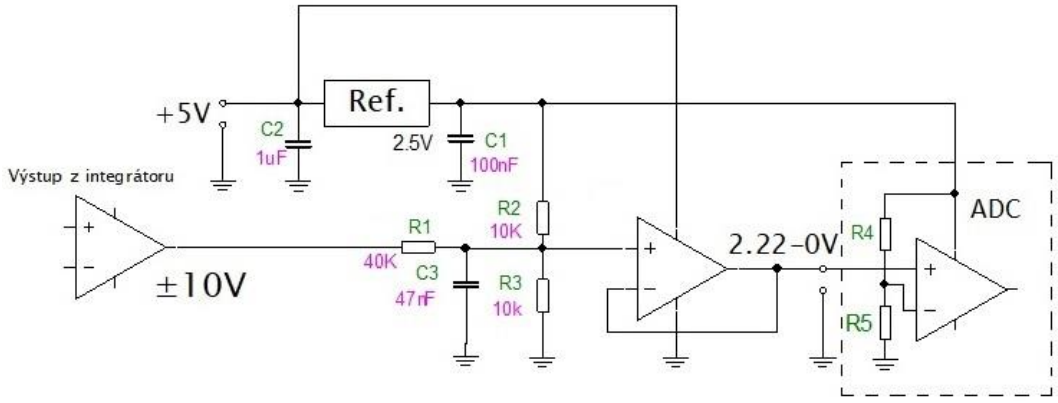
V 32bitové funkci je tedy 24 bitů AD převodníku. Jak jsem zmiňoval výše, AD převodník je 2bitový s tím, že poslední bit je znaménkový. To můžeme vidět na předchozím obrázku. Zde jsou uvedeny ještě další tři bity OVH a OVL jsou bity tzn. přetečení overflow. OVH (Overflow high) je log. 1 tehdy pokud vstupní napětí $U_{IN} > +U_{ref}$ a OVL (Overflow Low) je v log. 1 tehdy, pokud $U_{IN} < -U_{ref}$. Záporná čísla jsou kodována ve dvojkovém doplňku. Nejvyšší bit tedy určuje znaménko.

V předchozí kapitole jsem hovořil o modifikované verzi single conversion modu. V případě, že není podmínka splněna rovnou, může se stát, že převodník nevypne a může chovat jako v modu continous conversion. Na konečnou funkci to však vůbec nemá vliv.

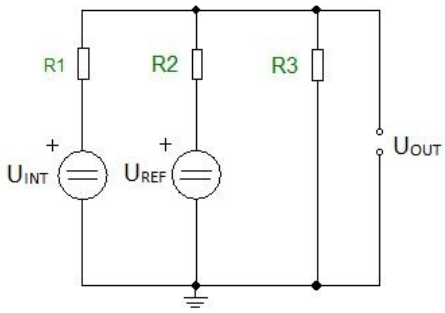
Navržení programu jako dvoucyklového má jednu podstatnou výhodu. Jelikož se nečeká na provedení konverze, hlavní smyčka programu je rychlejší. Samozřejmě, že by šlo vše umístit do jednoho cyklu. Zpoždění v jednom cyklu by mohlo odhadem nabrat kolem 35ms. Při krátkém stisku by poté třeba nemusela klávesnice správně zareagovat.

3.6.3 Realizace AD Převodníku

Za účelem vyšší přesnosti je použitý přesný zdroj referenčního napětí 2,5V. Dále je zde převodník napětí z ±10V na 0-2,22V. Převodník umí konverzi $\pm U_{REF}$, tedy na vstupu by stačil pouze dělič napětí. Nicméně to by znamenalo předělat celý program AD převodníku. V tomto případě je o dost jednodušší přidat jeden odpor mezi referenci a neinvertující vstup napěťového sledovače.



Obrázek 7 Zapojení ADC



Pro výpočet převodníku využiji náhradní schéma na Obrázek 8 Náhradní schéma převodníku. U_{INT} je napětí z integrátoru, U_{REF} je napětí reference a U_{OUT} je napětí na výstupu děliče, tedy na neinvertujícím vstupu napěťového sledovače.

Obrázek 8 Náhradní schéma převodníku

Nejdříve pomocí metody uzlových napětí sestavím rovnici pro výpočet napětí U_{OUT} jako

$$0 = \frac{U_{INT} - U_{OUT}}{R_1} + \frac{U_{REF} - U_{OUT}}{R_2} + \frac{-U_{OUT}}{R_3}$$

Pokud U_{OUT} položím rovný nule z rovnice vypadne poslední člen. Po dosazení dostanu:

$$0 = \frac{-10}{R_1} + \frac{2{,}5}{R_2}$$

Odpor R1 musí tedy být 4x větší než odpor R_2. Proto volím R_1 40k a R_2 10k. Nyní dosadím rovnici pro U_{OUT}=2.22V a U_{INT}=10V.

$$0 = \frac{10 - 2{,}22}{40k} + \frac{2{,}5 - 2{,}22}{10k} + \frac{-2{,}22}{R_3}$$

$$0 = 7{,}78 + 4 \cdot 0{,}28 + \frac{-88{,}8}{R_3} => R_3 \approx 10k$$

Zde musím uvést proč byla zvolena tak podivná hodnota výstupu 2,22V. Jelikož doma nemám vybavenou laboratoř, využíval jsem při kompletování laboratoř vedoucího práce. Některé věci bylo třeba řešit rychle namístě z důvodu úspory času. Hodnota R_3 byla zvolena odhadem na 10k. Při dosazení U_{REF} 2,5V do rovnice by hodnota tohoto odporu vyšla na 13,33k. Jelikož byl odhad relativně přesný, z časových důvodů nebyla tato hodnota měněna.

3.7 DA převodník

DA převodník v této aplikaci nepatří do přímé větve, a proto na něj nejsou kladeny tak velké nároky na přesnost. Jeho úlohou je zajistit pokud možno stálost výstupního napětí. Toho dosáhne kompenzací offsetu OZ. K neinvertujícímu vstupu je připojen nepřímo přes dělič napětí. Tento vstup převádí napětí ±5V na napětí ±15mV.

Pro přístroj byl navržen DA převodník MCP4822 od stejné firmy jako DA převodník. To hlavně z důvodů podobnosti řízení s AD převodníkem. DA převodník MCP 4822 je taktéž řízen po sběrnici SPI a má může mít stejné nastavení SPCR jako AD převodník, což značně zjednoduší práci.

MCP4822 je 12bitový duální DA převodník s vlastním referenčním napětím 2,048V. Jedná se o unipolární převodník se dvěma výstupy. Výstupy dokáží být řízeny individuálně. Práce využívá pouze jednoho výstupu.

3.7.1 Řízení DA převodníku

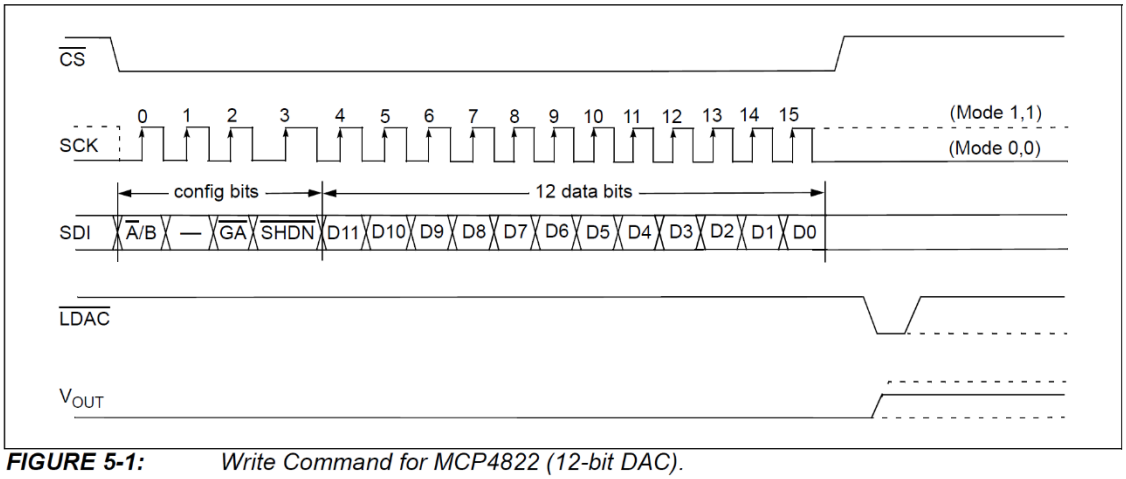


FIGURE 5-1: *Write Command for MCP4822 (12-bit DAC).*

Obrázek 9 Řízení DA převodníku

V kapitole AD převodníku jsem uváděl a porovnával dva mody ovládání AD převodníku. Zde stačilo pouze vybrat průběh pro 12-bitovou verzi.

```
uint8_t AD_dataset;
uint16_t binaryoffsetvalue;

binaryoffsetvalue=0x0FFF&((uint16_t)(offset_value*2000));
AD_dataset=0b01110000|(uint8_t)(binaryoffsetvalue/256);

_delay_us(10);
```

```
CS_DA_L;

SPDR=AD_dataset;

while(!(SPSR & (1<<SPIF)));

AD_dataset=(uint8_t)(binaryoffsetvalue);
SPDR=AD_dataset;

while(!(SPSR & (1<<SPIF)));
_delay_us(1);
CS_DA_H;

Latch_L;
_delay_us(1);
Latch_H;
```

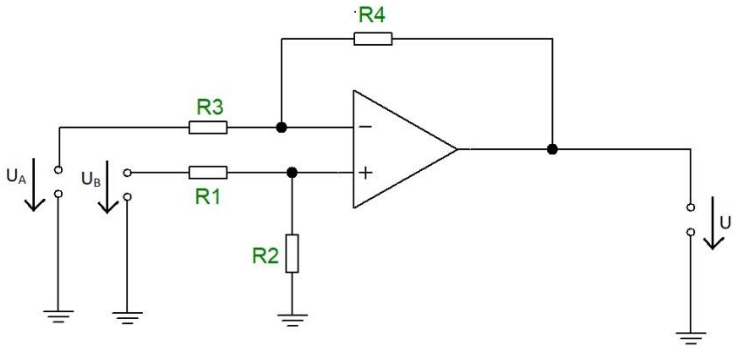
Program je psaný taktéž podle Obrázek 9 Řízení DA převodníku. Prvně se nastaví 16 bitová hodnota AD převodníku. K tomu je potřeba znát hodnotu offsetu, kterou je třeba nastavit. Ta se získá z globální proměnné offset_value typu float. Připomínám, že tato proměnná se nastavovala v menu offset pomocí funkce offset(). Lze jí také nastavit vzdáleně přes PC. Hodnota offset_value se pohybuje mezi 0-2,048V. To je softwarově zaručeno. Jelikož na data je vyčleněno 12 bitů, tedy 2^12 hodnot, je potřeba vynásobit proměnnou offset_value konstantou 2000. Rozsah se poté pohybuje mezi 0-2^12. První 4 nejvyšší bity jsou pro jistotu vymaskovány.

Protože SPI pracuje s 8 bitovým posuvným registrem, je potřeba tuto hodnotu ještě rozdělit na dva samostatné bajty. Na Obrázek 9 Řízení DA převodníku je vidět, že první hodnota je od MSB. Proto binaryoffsetvalue nejdříve vydělím 256, čímž ji posunu o 8 bitů doprava a poté přičtu řídící bity. (pozn. překladač při překladu kódu dělení 256 převede na bitový posun, který je značně rychlejší) Řídící bity jsou první 4 bity od MSB. Log.0 na ¬A/B znamená zápis na výstup A. Bit GA řídí zesílení. Log. 1 znamená žádné zesílení. Log. 0 na SHDN znamená vypnutí daného kanálu. Proto je na prázdné místo prvního odesílaného bajtu log. součtem přičtena hodnota 0b0111000.

AD převodník se aktivuje log. 0 na chipselect. Poté je zapsáno do SPDR, což odstartuje přenos. Po nastavení signálního bitu SPIF do log. 1 je potřeba zapsat do registru SPDR zbylý bajt. Nejjednodušší způsob, jak to udělat je přetypovat uint_16 na unit_8, čímž se ztratí informace na prvním bajtu. Po skončení přenosu je třeba nastavit chipselect do log. 1. Poté je třeba aktualizovat stávající hodnotu na novou. Toho se dosáhne pomocí impulsu na pinu Latch. Podle datasheetu musí tento puls mít alespoň 100ns, čehož se dosáhne vložením zpoždění.

3.7.2 Úprava signálu DA převodníku

Jelikož se jedná o unipolární převodník s rozsahem 0-2,048V je potřeba předělat jeho napěťový výstup na ±5V tak, aby vyhovoval vstupu řízení offsetu integrátoru. Návrh takového zapojení je uveden v datasheetu DA převodníku v kapitole 6.6 Bipolar operation. Výpočet hodnot pro DA převodník mi ale nefungoval. Nicméně řešení problému, proč tento postup nemohl fungovat mě dovedlo k fungujícímu řešení. Proto začnu právě nefunkčním řešením.



Obrázek 10 Rozdílový zesilovač

V datasheetu je uvedeno zapojení podobné s Obrázek 10 Rozdílový zesilovač. Toto zapojení je obyčejný rozdílový zesilovač. Jeho funkce je následující. Pokud uzemním vstup U_B, pak se jedná o invertující operační zesilovač a můžu tedy napsat, že $-\frac{R_4}{R_3} \cdot U_A = U$. V případě, že uzemním vstup U_A, jedná se o neinvertující zesilovač, který je doplněn o dělič napětí na neinvertujícím vstupu. Pokud mám uzemněný vstup U_A, můžu napsat že $U = U_B \cdot \frac{R_2}{R_1+R_2} \cdot \frac{R_3+R_4}{R_3}$. V případě neuzemnění obou vstupů můžu konečně napsat že:

$$U = U_B \cdot \frac{R_2}{R_1 + R_2} \cdot \frac{R_3 + R_4}{R_3} - \frac{R_4}{R_3} \cdot U_A$$

Pro lepší pochopení si nyní udělám substituci $U_B \cdot \frac{R_2}{R_1+R_2} \cdot \frac{R_3+R_4}{R_3} = U_2$ a $\frac{R_4}{R_3} \cdot U_A = U_1$ tedy:

$$U = U_2 - U_1$$

Chci-li docílit výstupu ±x volt, můžu toho dosáhnout dvěma způsoby. Pokud budu mít referenci na vstupu U_B pak $U_2=2*U_1 \Rightarrow U_2=5V \wedge U_1 \in (0V,10V)$. V tomto případě by při maximálním rozsahu DA převodníku bylo na výstupu -5V. Druhý způsob je připojit referenci na vstup U_A. Pak platí, že $2*U_2=U_1 \Rightarrow U_1=5V \wedge U_2 \in (0V,10V)$. V tomto případě je při maximálním výstupu DA převodníku na výstupu +5V. Proto je právě tento druhý způsob uveden primárně v datasheetu DA převodníku.

Nyní udělám ještě jednu substituci $\frac{R_2}{R_1+R_2} = X, \frac{R_3+R_4}{R_3} = Y, \frac{R_4}{R_3} = Z$. X, Y a Z jsou zesílení potřebné k nastavení napětí U_1 a U_2.

Pokud budu uvažovat právě druhý způsob, potřebuji za U_1 dostat hodnotu 5V. V případě reference +5V toho docílím tak, že X=1 tedy $R_3=R_4$. Zesílení Y se váže na zesílení X. V tomto případě

$$Y = \frac{R_3 + R_4}{R_3} = 2$$

Pro X platí: 0<X<1. X tedy dokáže napětí U2 pouze zeslabovat. Aby byl převod ±5V funkční, musí být na U2 hodnota 10V. Jelikož výstup U_B je roven maximálnímu výstupu DA převodníku, tedy 2,048V, je evidentní, že ani při maximálním zesílení 2x hodnoty 10V nedosáhnu, a proto metodu v datasheetu nelze použít.

Řešením je právě první způsob a to připojení reference na vstup UB. Připomínám že hodnoty ±5V se dá prvním způsobem dosáhnout když: $\pm 5 = U_2 - U_1 => U_2 = +5, U_1 \in (0\text{V}, 10\text{V})$.

Tedy:

$$U_B \cdot X \cdot Y = 5 \ \wedge Z \ \cdot U_A = 10$$

Vím, že U_B=5V a U_A=2.048V a také vím, že Y je závislé na Z proto nejdříve:

$$Z = \frac{10}{2{,}048} = 4{,}883$$

Pokud zvolím R_3=10k a vím, že Z=R_4/R_3, pak R_4=48.83k. Jelikož

$$Y = \frac{R_3 + R_4}{R_3} = \frac{48{,}83 + 10}{10} = 5{,}883$$

Protože U_B=5 a R_2 volím 10k:

$$5 \cdot X \cdot Y = 5 => X = \frac{1}{Y} = \frac{10}{48{,}83 + 10} = \frac{R_2}{R_1 + R_2}$$

Z toho plyne, že R2=R3=10k a R4=R1=48.83k

4 Ovládání přes PC

4.1 Stručný Popis sériové linky

K běžné výbavě mikrokontrolérů dnešní doby patří rozhraní USART. V této práci se využívá asynchronní režim sériové linky RS232 a to hned ve dvou případech. Ten první je samotné programování mikrokontroléru. V druhém případě jde o řízení přístroje fluxmetru přes PC. Jelikož PC je v dnešní době vybaveno výkonnějším rozhraním USB, je potřeba využít převodníku RS-232 na USB, kterým je kit vybaven. Sériová linka je řízena programem pomocí knihovny uváděné v online knize [1].

4.2 SCPI příkazy

Jedná se o běžně využívané příkazy pro řízení měřících zařízení přes počítač. Kdybych měl zařadit SCPI příkazy do referenčního modelu OSI pak bych je zařadil nejspíše do prezenční vrstvy. Hlavním úkolem prezenční vrstvy je transformování dat do tvaru, které používají aplikace. V tomto případě jde o transformaci dat a pokynů do textových řetězců. Jedná se tedy

o vyšší vrstvu, než na které je UART, o kterém jsem hovořil v předchozím odstavci.
U jiných zařízení např. při měření napětí můžou mít SCPI příkazy např. následující vzhled:

```
:MEASure
    :VOLTage
        :DC?
        :AC?
```

V aplikaci fluxmetru jsou příkazy definovány následovně:

OFST:(číslo) -> Pro nastavení ofsetu
CTHR:(číslo) -> Pro nastavení počtu závitů cívky (Coil THReads)
CSUR:(číslo) -> Pro nastavení plochy cívky (Coil SURface)

RELY:(10k,100k,1M,10M) -> Pro nastavení rozsahu odporu

PQAN:(O,B,H,G,Oe) -> Pro nastavení fyz. Veličiny (Physical QUAntity)

MEAS? -> Vrátí výsledek měření
SMEAS -> Každé 3s vrátí výsledek měření (Start MEASuring)

DMEAS -> Zastaví kontinuální měření (Disable MEASuring)

IDN? -> Vrátí infomaci o zařízení

***RST** -> Reset integrátoru

RSTD -> Reset celého zařízení včetně konstant

4.3 Programy ovládání přes PC

Celý program využívá spoustu knihoven. Všechny doposud použité knihovny byly buďto běžnou součástí Jazyka C, nebo součástí Atmel Studia. Knihovna pro UART byla jediná z volně dostupného kódu, kterou jsem použil. Tuto knihovnu mám z [1]. Využívám z ní dvě funkce a to uart_gets(), která spustí přerušení a vyčte řetězec znaků z PC a uart_puts(), která pošle řetězec znaků zpět do PC.

```
if(uart_gets(SCPICharacter))  //jestliže přišel řetězec znaků
        {
                if (strlen(SCPICharacter)>36) //a jestliže nebyl delší jak 36 znaků
                {
                        uart_puts("Too Long");
                }
                else
                {
                        //zde leží jednotlivé kody
                }
```

V prvním odkazu kódu se při odeslání znaku do zařízení spustí přerušení a SCPI se stane

nenulovou. Následně se zkontroluje velikost příchozího souboru. V případě, že přijde nesmyslně dlouhý řetězec, je obsluze automaticky odeslána chybová hláška.

4.3.1 Vzdálené nastavení konstant a offsetu

```
if (memcmp(SCPICharacter,"OFST:",5)==0)
{
        for ( step=0;step<32;step++)
        {
                if ((SCPIcharnumber[step]='\0')){break;}
                SCPIcharnumber[step]=SCPICharacter[step+5];
        }



        float SCPIfloatnumber =atof(SCPIcharnumber);
        if (SCPIfloatnumber>2.048)
        {
                uart_puts("overflow: offset set to maximal value 2.048V");
                offset_value=2.048;
        }
        else
        {
                offset_value=SCPIfloatnumber;
                char *SCPIcharreturn="        ";
                sprintf(SCPIcharreturn,"%.3f",SCPIfloatnumber);
                uart_puts("Offset set to:");
                uart_puts(SCPIcharreturn);
                uart_puts("V");
        }

}
```

Zde je využita funkce *memcmp*(), která vezme první string a hledá ve druhém stringu shodu znaků znaků o určitém počtu. Pokud se tento určitý počet shoduje v s druhým stringem, výsledkem funkce je nula. Následuje cyklus for, který postupně uloží číselnou hodnotu za hlavičkou. Ve chvíli, kdy zjistí konec, automaticky vyskočí, čímž se celý cyklus zrychluje. Následuje překlad onoho stringu na číslo. Jelikož se jedná o nastavení offsetu, je zde podmínka přetečení. Obsluha je následně informována o nastavení, či případném překročení maximálního rozsahu DA přebodníku.

Pokud jde o nastavení konstant, smysl kódu je v podstatě totožný. Kód se mění se samozřejmě v hlavičce a u nastavení konstant cívky chybí také ochrana proti překročení maximálnímu rozsahu. Pro přehlednost zde tyto kódy uvedeny nebudou.

4.3.2 Vzdálené přepínání rozsahů a přepínání měřených veličin

```
if (memcmp(SCPIcharnumber,"10k",3)==0) //10k
```

```
                        {
                                Relay_port = 0b00100000;
        //10k
                                Tau=0.0001;
                                uart_puts("Relay set to 10k");
                        }
                        else if (memcmp(SCPIcharnumber,"100k",4)==0)
                        {
                                Relay_port = 0b00010000;
        //100k
                                Tau=0.001;
                                uart_puts("Relay set to 100k");
                        }
                        else if (memcmp(SCPIcharnumber,"1M",2)==0)
                        {
                                Relay_port = 0b00001000;
        //1M
                                Tau=0.01;
                                uart_puts("Relay set to 1M");
                        }
                        else if (memcmp(SCPIcharnumber,"10M",3)==0)
                        {
                                Relay_port = 0b00000100;
        //10M
                                Tau=0.1;
                                uart_puts("Relay set to 10M");
                        }
                        else
                        {
                                uart_puts("wrong set");
                        }
```

Předchozí kód je ukázka vzdáleného řízení roszahů odporu. Zde je taktéž využita funkce *memcmp*. V tomto případě využívám druhou vlastnost a to, že funkce hledá shodu kdekoliv ve stringu a nejen na začátku stringu. Pokud najde shodu např. se stringem „10k“, přepne se relé a zároveň je přenastavena konstanta Tau pro výpočet.

Při prvním pokusu jsem v kódu využíval funkci pro porovnání strcmp(), která pouze porovnávala stringy od první pozice, a proto byl pokus o napsání kódu s funkcí strcmp() značně komplikovanější.

Funkce pro nastavení fyzikálních veličin je do značné míry podobná, proto ji taktéž nebudu uvádět. Přepisuje pouze jiné globální konstanty a vrací obsluze jinou zpětnou vazbu.

4.3.3 Reset integrátoru a reset zařízení

```
if (memcmp(SCPICharacter,"*RST",4)==0)
{
        Relay_port_set|=(1<<Relay_port_reset);//nastaveni vystupu

        Relay_port|=(1<<Relay_port_reset);
        _delay_ms(1500);

        Relay_port&=~(1<<Relay_port_reset);
        uart_puts("Reset done:");

}
```

Reset integrátoru popsaný v předešlém kódu se nikterak neliší od resetu přes tlačítko. V případě testování řetězce je zde využit stejný princip, jako v kapitole 4.3.1 a 4.3.2 . Jiné je to však u resetu celého zařízení. Podle webu [2] se zde využívá watchdog timeru. Ten se nastaví na nejkratší časový limit a poté přejde do nekonečné nic nedělající smyčky. Watchdog poté resetuje processor. Kód využívá knihovnu Atmel Studia <avr/wdt.h> a vypadá následovně:

```
if (memcmp(SCPICharacter,"RSTD",4)==0)
{
	wdt_enable(WDTO_15MS);
}
```

4.3.4 Vzdálené měření

Poslední zajímavou částí je měření. Zde jsem naprogramoval dvě možnosti. Zaprvé jednoduché odeslání aktuální hodnoty výsledku. Za druhé kontinuální měření po 3 vteřinách. První možnost pouze otestuje příchozí řetězec a na dotaz pošle zpět řetězec aktuální hodnoty fluxmetru. Uvedu ale kód právě pro druhou možnost měření, jelikož je o poznání zajímavější.

```
if (memcmp(SCPICharacter,"SMEAS",5)==0)
	{
		int stop =1;
		while(stop)
			{
				_delay_ms(3000);
				uart_puts(fluxmetr_result());
				if (memcmp(SCPICharacter,"*RST",4)==0)
			{

	Relay_port_set|=(1<<Relay_port_reset);//nastaveni vystupu
	Relay_port|=(1<<Relay_port_reset);
	_delay_ms(1500);
	Relay_port&=~(1<<Relay_port_reset);
	uart_puts("Reset done:");
	}
	if (uart_gets(SCPICharacter))
	{
		if (strlen(SCPICharacter)>36) //a jestliže nebyl delší jak 36 znaků
		{
			uart_puts("Too Long");
		}
	else
	{
	if ((memcmp(SCPICharacter,"MSRCS",5)==0)){stop=0;}
	}
}
```

Nejdříve program čte hlavičku. Pokud obsluha zadá příkaz ve formě SMEAS pro měření kontinuálně, každé 3 vteřiny vrátí string s hodnotou měření. Ve smyčce while se ale program musí chovat stejně jako v hlavní funkci programu *main*(). Proto se i zde následně ptá, jestli nepřišel ukončovací znak, a i zde tento příchozí string testuje proti přesáhnutí limitu pole charů. Poté se v každé smyčce ptá, jestli nepřišel string s informací o ukončení. Za povšimnutí také stojí resetovací sekvence, která je tu přítomná.

Použitá lieratura

[1]ZÁVODSKÝ, Ondrej. *Programujeme AVR v jazyku C* [online]. 2012 [cit. 2019-08-26]. Dostupné z: zawin@svetelektro.com

[2]*Microchip* [online]. [cit. 2019-08-28]. Dostupné z: https://www.microchip.com/webdoc/AVRLibcReferenceManual/FAQ_1faq_softreset.html

[3]Three-state logic. *Wikipedia* [online]. [cit. 2019-08-29]. Dostupné z: https://en.wikipedia.org/wiki/Three-state_logic

[4]HAASZ, Vladimír a Miloš SEDLÁČEK. *Elektrická měření: přístroje a metody*. Vyd. 2. Praha: Vydavatelství ČVUT, 2003. ISBN 80-01-02731-7.

[5]Chyby AD převodníku. *Renesas* [online]. [cit. 2018-09-14]. Dostupné z: Obrázek z https://en-support.renesas.com/knowledgeBase/16978566 dne 14.4. 2019

[6]BLAŽEK, Jan, VIK, Ing. Vladimír, ed. *Aplikovaná elektronika: Elektronika II*. 2013. Jičín: Vyšší odborná škola a střední průmyslová škola Jičín, 2013.

[7]PLÍVA, Z., J. DRÁBKOVÁ, J. KOPRNICKÝ a L. PETRŽÍLKA. Metodika zpracování bakalářských a diplomových prací. 2. upravené vydání. Liberec: Technická univerzita v Liberci, FM, 2014. ISBN 978-80-7494-049-1. Dostupné z: doi:10.15240/tul/002/2014-11-002

[8] Mikrokontrolery AVR Atmega. *tajned* [online]. [2019-08-29]. Dostupné z: http://www.tajned.cz/category/programming/mikrokontrolery/avratmega/

[9]Programování v jazyku C/C++. *Sallyx* [online]. [2019-08-29]. Dostupné z: https://www.sallyx.org/sally/c/

Závěr

Finálním produktem této práce je prototyp zařízení fluxmetru. Tento prototyp dokáže změřit kromě magnetického toku také magnetickou indukci a intenzitu magnetického pole. Poslední dvě veličiny dokonce dokáže zobrazovat v anglosaských jednotkách. Zařízení se dá ovládat jak pomocí klávesnice, tak pomocí SCPI příkazů přímo z PC, ke kterému se dá připojit pomocí USB.

Velkou výhodou zařízení jsou vlastní funkce a procedury AD a DA převodníku a také vlastní funkce pro klávesnici. Tato profesionální forma přístupu umožňuje komerční využití. Z volně dostupného specifického kódu práce využívá pouze knihovnu pro UART. Vývoj vlastních funkcí a procedur byl časově velmi náročný. V případě AD a DA převodníku nezbytný. V případě klávesnice došlo k usnadnění pozdější práce a ke zrychlení programu.

Další výhodou zařízení je velké množství softwarových ochran a chybových hlášek. Zařízení má slušnou zpětnou vazbu. Při řízení z PC vždy reaguje na každé nastavení a ve většině případů, kdy nastane chybné zadání, dokáže obsluhu upozornit.

Při vývoji zařízení jsem neustále narážel na dva problémy. Jelikož se jedná o komplexní firmware s více periferiemi, které spolu navzájem komunikují pomocí mikroprocesoru, bylo potřeba je navzájem propojit. Toto propojení s sebou neslo jakousi „předtuchu“ jak by mohla daná periferie komunikovat a jak ji nejlépe propojit s druhou. K tomu se těsně váže časový odhad, jak dlouho bude vývoj trvat. Časový odhad je velice důležitý, protože školní rok je pro vývoj takového zařízení krátký a jedná se tedy o kompromis co bude možné vytvořit.

V prvním případě se mi velmi často stávalo, že jsem zvolil nejvhodnější časově a z hlediska rychlosti tu nejlepší verzi, jak danou část naprogramovat. Po připojení další části se stala tato metoda nevhodnou, a proto bylo třeba hned několik částí programu přepracovat.

Tedy některé funkce byly přepracovány hned několikrát. Každé přepracování muselo být testováno kvůli bugům v programu. Tyto chyby byly nepředvídatelné. V případě zapisování pole charů pomocí pointeru často nastával v případě posledního znaku ve formě mezery zmatek. V jednom případě při použití funkce na zřetězení stringů tento zápis dokonce překročil zadaný počet míst. Funkce pro zřetězení se stala nestabilní a přepsala místo v navazující paměti. Protože žádná adresace není přímá, chyba se projeví náhodně v úplně jiné funkci. Oprava je značně komplikovaná a časově náročná. Některé zápisy kódu jsou právě z těchto důvodů napsány dosti rozdílně, nicméně jsou funkční.

Jelikož se jedná o prototyp, je zde spoustu věcí a funkcí, které by šlo vylepšit a na které nezbyl čas. Jedná se například o využití vnitřní EEPROM mikroprocesoru k ukládání konstant. Dalším bodem možného vylepšení by mohl být vlastní tištěný spoj. Pájivé pole a jeho provizorní zapojení není úplně nejvhodnější pro měřící přístroj. Automatické nastavení offsetu by jistě také práci zrychlilo. Raději jsem ale čas věnoval do opravy chyb a chybových hlášek, aby to, co již bylo zprovozněno bylo odolnější a především funkční.

Co by bylo pro práci jistě dobré je porovnání přístroje s jiným přístrojem a porovnání přesnosti. Přístroj k porovnání přišel bohužel pozdě, a proto byl poslední bod testování splněn bez porovnání, není tedy známá přesnost.

Všechny body práce byly splněny, dokonce v některých případech vznikla drobná vylepšení. Tato vylepšení jsou: měření více veličin jako je např. mag. indukce, přepínání předpon soustavy jednotek SI a chybové hlášky, které zjednodušují obsluhu.