ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

BAKALÁŘSKÁ PRÁCE

Aplikace polohových snímačů pro řízení

Josef Justa **2013**

Anotace

Předkládaná bakalářská práce je zaměřena na využití moderních sensorů pro řízení. Součástí práce bylo vytvoření senzorické rukavice osazené flex senzorem a akcelerometrem. V práci je také řešena komunikace mezi těmito senzory a řízeným systémem, která je zajištěna mikrokontrolérem a modulem bluetooth. Následně byla vytvořena jednoduchá aplikace pro využití systému senzorů.

Klíčová slova

Akcelerometr, mikropočítač, polovodičový ohybový sensor, sběrnice I2C, rukavice

Abstract

Application of position sensors for control

The bachelor's thesis is focused on the application of modern sensors for control. Part of the thesis was the construction of a sensory glove equipped with flex sensor and accelerometer. The communication between these sensors and controlled system is also being addressed. This communication is provided by a microcontroller and a Bluetooth module. Subsequently, a simple application for use of the sensor system was created.

Key words

Accelerometer, microcontroller, flex sensor, I2C bus, glove

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 7.6.2013 Jméno příjmení

……..

Obsah

Seznam symbolů a zkratek

a $\left[\frac{m}{s^2}\right]$	Zrychlení
g $\left[\frac{m}{s^2}\right]$	Gravitační zrychlení. Teoretická střední hodnota gravitačního zrychlení na povrchu Země je g = 9,823 $\left[\frac{m}{s^2}\right]$
AVR	Druh architektury mikrokontrolérů firmy Atmel
SDA	Synchronous Data. Synchronizace datových signálů.
SCL	Synchronous Clock. Synchronizace hodinových signálů.
SPI	Serial Peripheral Interface.
I2C	(IIC) Inter Integrated Circuit.
LGA	Land Grid Array.
k [-]	Konstanta změny odporu v závislosti na změně délky
ρ [Ωm]	Měrný elektrický odpor
ACK	Acknowledgement. Potvrzení.
A/D	Analog to digital converter. Převodník analogového signálu na číslicový.
USART	Universal Synchronous / AsynchronousReceiever and Transmitter. Synchronní / asynchronní sériové rozhraní.
USB	Universal Serial Bus.
CLK	Clock. Hodinový signál.
DPS	Deska plošných spojů.
THT	Trough Hole Technology.
SMT	Surface Mount Technology. Technologie plošné montáže.
C [F]	Elektrická kapacita
R [Ω]	Elektrický odpor
l [m]	Délka
S [m^2]	Plocha
U [V]	Elektrické napětí
I [A]	Elektrický proud
s	Smyčka

Seznam obrázků

Úvod

Předkládaná práce je zaměřena na využití moderních sensorů obecně pro řízení. Součástí této práce je realizovat senzorickou rukavici a popsat práci na jejím vývoji. Aplikace senzorické rukavice (angl. Wired glove) umožňuje snímat polohu a pohyby ruky.

Práce je rozdělena do čtyř kapitol. V první kapitole se zaměřuje na problematiku snímání veličin a umístění snímačů, aby co nejpřesněji snímaly pohyby a polohu ruky. Ve druhé kapitole pojednává o komunikaci mezi těmito snímači a řízeném zařízení. Třetí kapitola je zaměřená na vývoj a výrobu senzorické rukavice. Čtvrtá kapitola pojednává o možnosti využití senzorické rukavice.

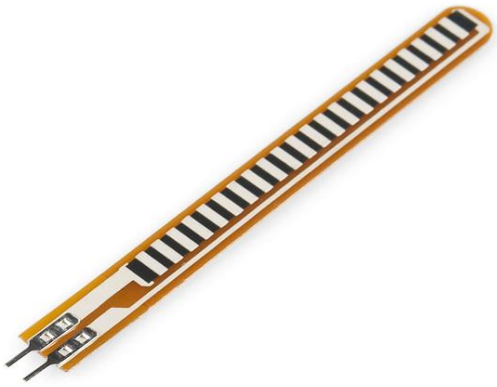
Při tvorbě práce byl kladen důraz na jednoduchost popisování problematiky, aby z ní mohl čerpat informace i člověk, který není úzce zaměřen do daného oboru.

1 Senzory – vhodná volba a umístění

Cílem každého sensoru je získávat informace o měřené veličině, přitom ji co nejméně ovlivňovat a být co nejméně ovlivňován nežádoucími vlivy. Při řešení této problematiky je důležité také neopomenout ekonomickou záležitost. Proto jsem se tedy zaměřil na relativně moderní senzory. Po uvážení a prozkoumání senzorů volně k dostání jsem zvolil pro moji aplikaci kombinaci dvou druhů senzorů - flex senzoru a akcelerometru.

1.1 Polovodičový ohybový senzor (flex senzor)

Tento relativně moderní senzor nabízí oproti konvenčním tenzometrům i stonásobně vyšší konstantu změny odporu v závislosti na změně délky.



Obr. 1 Ukázka flex senzoru

Zdroj: https://www.sparkfun.com/products/10264

- **Tenzometry**

k = cca 2

Změna odporu je značně závislá na změně teploty.

Rovnice popisující princip funkce tenzometru:

$$R = \rho * \frac{l}{S} [\Omega, \Omega m, m, m^2]$$

Napínáním tenkého drátku zvyšujeme délku *l* a snižujeme průřez *S*, díky tomu zvyšujeme odpor *R* a naopak. Pro problematiku snímání ohybu kloubů se jeví jako naprosto nevhodné.

- **Flex senzor**

k > 100

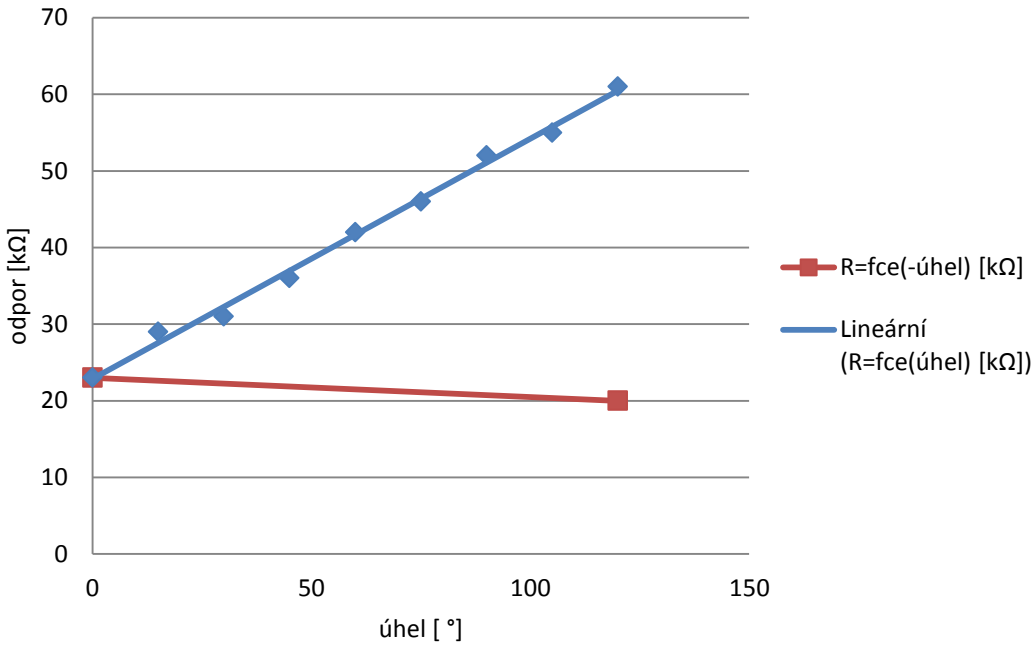
Tento senzor je k dostání ve formě plastového pásku (viz Obr. 1), na kterém je z jedné strany nanesena vrstva polovodiče citlivého na změnu délky a vrstva vytvářející vodivou cestu mezi polovodičem a konektorem. Ohýbáním pásku napínáme polovodič a ten mění odpor. Pásek je konstruovaný takovým způsobem, aby byl odolný vůči podélnému pnutí. Cena na trhu se nyní pohybuje pro použitý senzor délky 2.2“ okolo 200 Kč, takže cenová zátěž pro danou aplikaci několika senzorů se pohybuje na přijatelné ekonomické mezi.

1.1.1 Kontrola parametrů měřením

Pro zjištění parametrů flex senzoru byla měřena změna odporu v závislosti na ohybu flex senzoru okolo válcovitého předmětu o průřezu 14 mm (Tabulka 1). Stupeň ohybu vyjadřuje úhel svírající roviny senzoru před a za ohybem. Odpor obráceně vyjadřuje ohyb na druhou stranu. Měření bylo provedeno do úhlu 120°, protože při vyšším úhlu na daném poloměru hrozilo mechanické poškození senzoru.

úhel [°]	Odpor [kΩ]	Odpor obráceně [kΩ]
0	23	23,0
15	29	22,6
30	31	22,3
45	36	21,9
60	42	21,5
75	46	21,1
90	52	20,8
105	55	20,3
120	61	20,0

Tabulka 1: Naměřené hodnoty



Obr. 2 Závislost změny odporu na úhlu ohybu flex senzoru

Zdroj: Autor

V grafu (Obr. 2) vidíme jisté nepřesnosti oproti lineární charakteristice dané měřením. Měření ohybu poukázalo především na dva zásadní parametry flex senzoru. Za „pozitivní“ vlastnost lze považovat, že změna odporu na velikost ohybu senzoru má lineární charakter. Naopak „negativním“ parametrem pro danou problematiku jsou velmi nesymetrické vlastnosti senzoru vzhledem k ohybu do opačných směrů. Z grafu lze také vypozorovat, že změna odporu vzhledem k úhlu ohybu je v jednom směru vysoká (potvrzuje teoretický vysoký koeficient k).

Následné měření poukázalo na to, že změna odporu je téměř nezávislá na změně teploty v teplotním pásmu snesitelném pro člověka.

Poznámka:

Flex senzory jsou velmi citlivé na zvýšení teploty při pájení kontaktů. Vodivá vrstva nanesená na senzoru, ke které je kontakt mechanicky připojen, při zvýšené teplotě degraduje a kontakt se senzorem se přeruší. Doporučuji proto nevolit techniku připojení kontaktů pájením, nebo pájet velmi krátce a s maximální opatrností.

Tuto drahocennou informaci jsem se dozvěděl příliš pozdě (7 z 10 senzorů bylo po pájení poškozeno). Problém jsem vyřešil použitím vodivého lepidla (na trhu pod anglickým názvem Wired glue) v kombinaci s plastovými výstužemi, aby zmírnily možnost ohybu místa

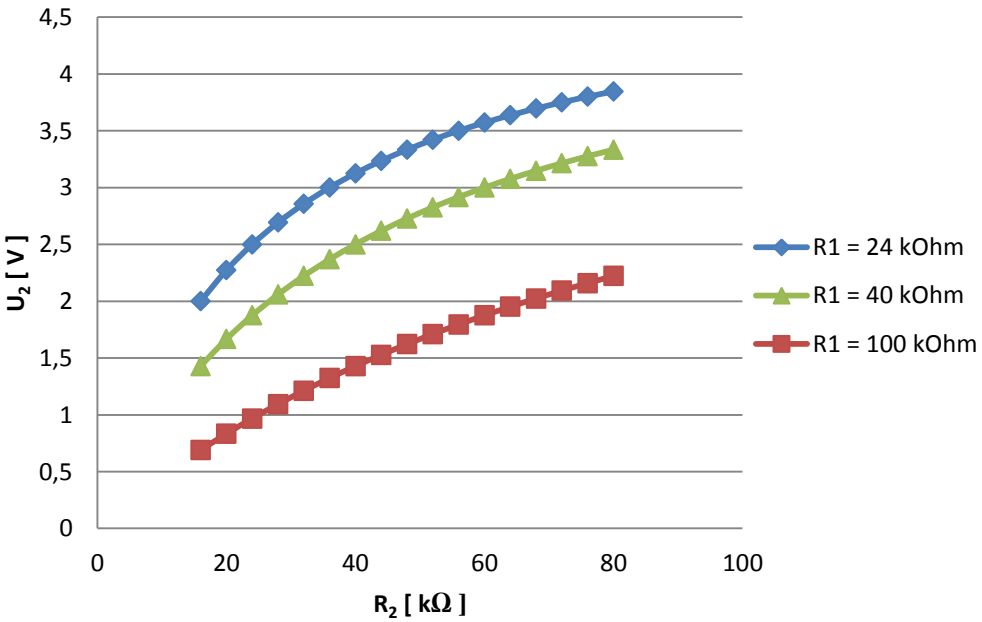
v blízkosti opraveného kontaktu lepidlem. Některé senzory byly poškozeny tak, že je nešlo opravit lepidlem. V těchto případech jsem musel vytvořit nové kontakty krátkým drátkem.

1.2 Měření změny odporu flex senzoru

O tom, že flex senzor má velkou změnu odporu v závislosti na úhlu ohnutí, jsme se přesvědčili v minulé kapitole. Nastává otázka, jak tuto změnu odporu vhodně změřit. U tenzometrických metod se často setkáváme se složitějšími můstkovými zapojeními, abychom dokázali změřit i malou změnu odporu. V našem případě nejsou tyto metody nutné.

Vydal jsem se metodou v zapojení asi nejjednodušší, a to odporovým děličem. Jednoduchost na straně desky plošných spojů však ztěžuje práci s vyhodnocenými daty. Na následujícím Obr. 3 vidíme závislost výstupního napětí děliče na změně odporu. $U_1 = 5$ [V] $R_1 = 24$ [kΩ]. Graf je vykreslen podle známého vztahu pro výpočet výstupního napětí děliče skládajícího se ze dvou rezistorů.

$$U_2 = U_1 \frac{R_2}{R_1 + R_2} \ [\Omega]$$



Obr. 3 Závislost výstupního napětí nezatíženého děliče na odporu flex senzoru

Zdroj: Autor

V grafu (Obr. 3) lze na první pohled zjistit, že charakteristika děliče není lineárně závislá na změně jednoho z odporů. Pokud bychom tuto nelinearitu nekompenzovali (brali bychom napětí jako lineární v závislosti na ohybu), vyvolávala by chybu při měření. Jelikož jsme si

v předešlé kapitole dokázali, že odpor flex senzoru se mění lineárně na ohybu, bude nám stačit podle obráceného výpočtu vypočítat odpor flex senzoru.

$$R_2 = R_1 \frac{U_2}{U_1 - U_2} \ [\Omega]$$

Dá se také odečíst, že nejvyšší citlivosti dosáhneme, pokud nastavíme rezistor R1 do středu odporové charakteristiky flex senzoru. Metodou odporového děliče však ztrácíme rozsah, ve kterém používáme A/D převodník, cca na polovinu. Samozřejmě by bylo možné posunout úroveň napětí až k nule například zenerovou diodou. Mikrokontrolér Atmega8 má také možnost z vnějšku nastavit referenční napětí pro A/D převodník, čímž by bylo možné snížit horní rozsah pro převod. V aplikaci senzorické rukavice pravděpodobně nebudeme nikdy potřebovat dosahovat takovýchto přesností a ani v mém projektu při snímání ohybu není zásadní maximální přesnost (dle mého názoru je sto a tisíc různých možností ohybu kloubu pro člověka nerozeznatelný rozdíl, nehledě na to, že pokud bychom zvýšili přesnost, přenášeli bychom navíc pouze třes ruky a jiné nežádoucí vlivy). Vzdálenost k maximálním hranicím A/D převodníku proto ponechávám jako rezervu. Samozřejmě v řadě jiných možných aplikací pro flex senzor je možné dosahovat násobně vyšší přesnosti.

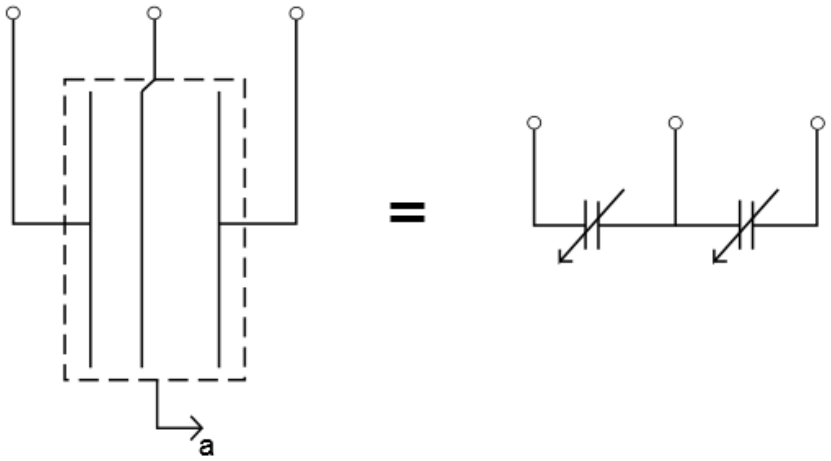
1.3 Akcelerometr

V aplikaci senzorické rukavice zajišťuje akcelerometr měření natočení ruky oproti rovině Země (gravitačnímu zrychlení Země). Na doporučení vedoucího práce jsem vybral digitální akcelerometr, aby se zmenšil rušivý vliv okolního prostředí při komunikaci. Digitální akcelerometry jsou charakteristické komunikací po sběrnici a možností nastavení akcelerometru do optimálního stavu pro měření zrychlení.

Akcelerometr, který jsem zvolil, je připraven v modulu pro snažší použití v praxi. Díky tomu odpadla problematičtější část pájení pouzdra typu LGA. Využit byl v tomto případě akcelerometr ADXL345, který podporuje komunikaci přes I2C i SPI sběrnici. Vzhledem ke konstrukční jednoduchosti sběrnice jsem zvolil pro komunikaci sběrnici I2C (více o této problematice v kapitole 2.1).

Akcelerometr pracuje na kapacitním principu. Pro měření každé osy obsahuje kapacitní snímač. Prostřední deska snímače je umístěna v pružném prostředí a pokud na ni působí zrychlení, vychýlí se proti směru působení daného zrychlení. Díky tomu se změní

kapacita mezi deskami a tu měříme změnou fáze signálu procházejícího skrze snímač (viz Obr. 4). [4]



Obr. 4 Ukázka principu kapacitního akcelerometru

Zdroj: Autor

ADXL345 nabízí možnost pracovat v mnoha režimech maximálního přetížení. Pro moji aplikaci jsem vybral maximální přetížení ±2g. Přitom se nepředpokládá, že by rukavice při standardním užívání někdy dosáhla většího tíhového zrychlení.

Další z možností, které nabízí ADXL345, je nastavení posuvu sbíraných dat od standardních tří os v registrech offset. Díky tomu je možné nastavit výchozí polohu na rukavici, ze které bude natočení sledováno.

Ve srovnání s **gyroskopem** je akcelerometr komplexnějším zařízením, ale nastává u něj problém. Pokud bychom chtěli přesně měřit úhel natočení od gravitační osy i při jiném zrychlení (v mém případě pohyb ruky), tak se nám to pravděpodobně nepodaří. Gyroskop by měl tuto práci zastat mnohem lépe, ale nedokáže změřit zrychlení. V ideálním případě by bylo nutné zkombinovat gyroskop s akcelerometrem, ale vzhledem k ještě větší složitosti jsem se do této problematiky nepouštěl. V praxi se ukázalo, že pokud akcelerometrem příliš prudce nepohybujeme, dokáže změřit natočení velmi uspokojivě.

1.4 Umístění senzorů

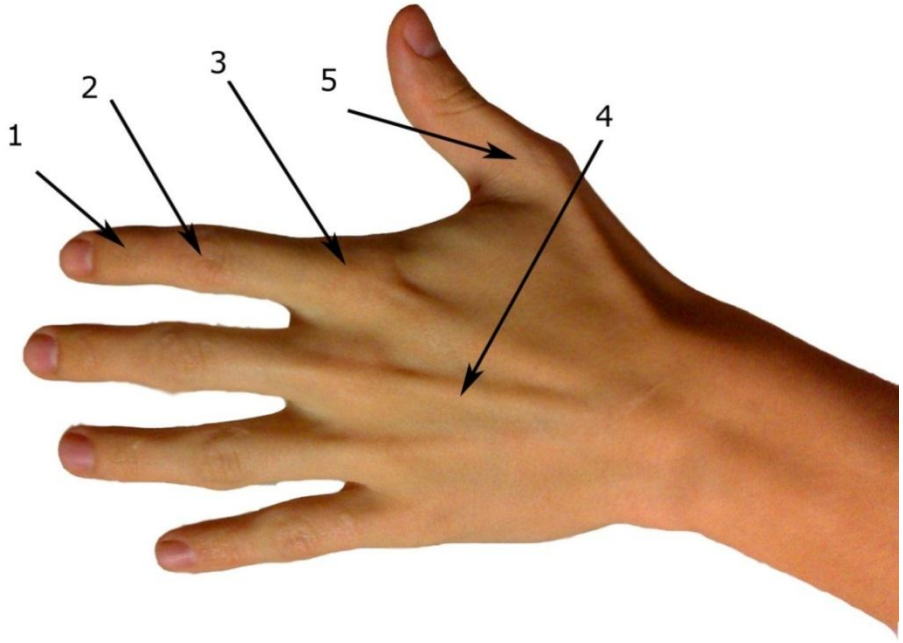
Všechny senzory jsou na rukavici umístěny optimálně, aby zachytily co nejvíce důležitých pohybů ruky. Po prozkoumání pohybů ruky jsem našel pouze jediný pohyb kloubu, který je pro flex senzor problematický, a to přibližování a oddalování kloubu č. 5 směrem

ke vnitřku dlaně (viz Obr. 6). Pro zvětšení tohoto ohybu je na rukavici připevněna výztuha. V praxi posloužil jako výztuha kousek pastelky (viz Obr. 5). Akcelerometr je umístěn na pozici č. 4 ve stejné rovině jako dlaň, aby snímal její natočení v prostoru.

Obr. 5 Výztuha zvyšující ohyb kloubu u palce

Zdroj: Autor

Flex senzory jsou připevněny na každý prst po dvou, aby snímaly ohyb kloubů č. 2 a č. 3 (viz Obr. 6) a aby jednotlivé flex senzory nepřekrývaly tyto klouby najednou. Flex senzor totiž snímá ohyb po celé délce pásku a nedá se rozlišit, jestli je daný ohyb způsobený na začátku pásku nebo na jeho konci. Ve všech projektech, které jsem našel, byl vždy použit pouze jeden dlouhý flex senzor. Kvůli tomu se není možné rozlišit, jestli se ohýbá kloub č. 2 nebo č. 3 (z důvodů uvedených výše).



Obr. 6 Ukázka důležitých oblastí ruky při snímání senzory

Zdroj: Autor

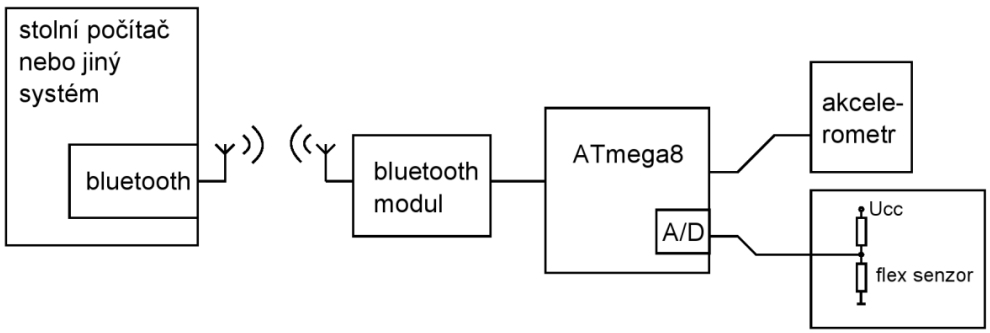
Flex senzory jsou připevněny zároveň tak, aby v nich ohyb kloubu nevyvolával pnutí, které by mohlo pásek mechanicky poškodit. To je zajištěno kapsičkou vytvořenou pro každý pásek, ve které se může pásek mírně posouvat.

Pro bezchybné měření ohybu je nutné zajistit, aby se aktivní strana flex senzoru, která je nanesena vrstvou polovodiče, chránila izolací, nebo se nedotýkala ničeho vodivého. Cokoliv s odporem ve stejném řádu nebo nižším, by dotykem s aktivní částí polovodiče značně ovlivňovalo měření ohybu. Kloub č. 1 pro zjištění polohy prstů snímat nepotřebujeme, protože je pohybově spjatý s kloubem č. 2.

Počet snímačů jsem se snažil optimalizovat, aby byl poměr nákladů na senzory a výsledná zjištěná data z nich v přiměřené míře. Deseti flex senzory lze již velmi účinně snímat jednu ruku. Pro pohyby jako např. roztahování prstů od sebe by se musela využít jiná metoda snímání nebo více vhodně umístěných flex senzorů.

2 Komunikace mezi senzory a počítačem

Jako hlavní řídící jednotka slouží v mém projektu mikrokontrolér architektury AVR. Výhodou této architektury je, že je velmi známá. Pokud řešíte nějaký problém, je velmi pravděpodobné, že už ho někdo řešil dříve nebo řešil obdobný. Velmi mi v začátcích pomohla elektronická kniha od Ondreje Závodského [1]. V mém projektu je aplikace specifikována na demonstrativní ukázku dat, které získávám. Data upravuji do vhodné podoby a posílám je do počítače, kde je pomocí aplikace v Microsoft framework zobrazuji. Znázornění komunikace ukazuje Obr. 7.

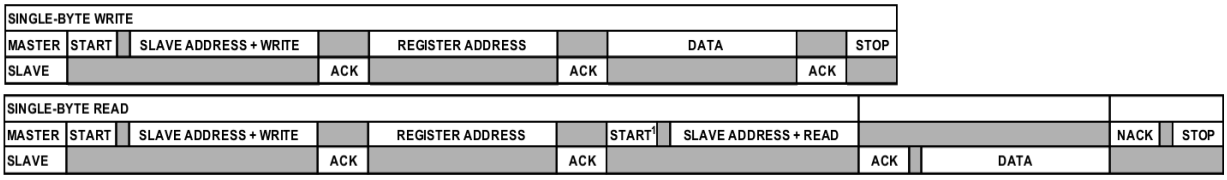


Obr. 7 Blokové schéma zobrazující komunikaci se senzory

Zdroj: Autor

2.1 Komunikace s akcelerometrem

Komunikace s akcelerometrem probíhá přes sběrnici I2C. I2C je dvouvodičová sběrnice pracující na principu otevřených kolektorů. Vodiče jsou v době klidu vytaženy „pull up“ rezistory na logickou „1“. Akcelerometr se před začátkem měření nastaví do režimu, ve kterém bude pracovat a následně se spustí měření. Komunikuje se podle továrního protokolu nastaveného na ADXL345 (viz Obr. 8).



Obr. 8 Grafická ukázka průběhu komunikace s akcelerometrem

Zdroj: Autor

Začátek komunikace je pro všechny typy přenosu stejný. Mikrokontrolér vyšle start podmínku stažením „pull up“ rezistoru na SDA (Synchronous Data) a poté stáhne SCL (Synchronous Clock). SDA v synchronizaci s náběžnou hranou SCL odešle první byte. Ten se skládá ze 7bitové adresy zařízení, se kterým budeme komunikovat, k té je připojen 1bit sdělující zařízení, že do něj chceme zapsat (logická „0“). Na následnou devátou náběžnou hranu zařízení odpoví potvrzovacím signálem ACK (Acknowledgement) tím způsobem, že stáhne SDA do logické „0“. Následně mikrokontrolér odešle další byte s osmibitovou adresou registru zařízení, se kterou chceme pracovat. Zařízení odpoví obdobně signálem ACK. Následná komunikace se liší podle jejího typu.

Zápis jednoho bytu používám pro počáteční nastavení akcelerometru před měřením. Další byte, který odešleme mikrokontrolérem, si zařízení zapíše do adresy odeslané minulým bytem. Zařízení potvrdí příjem signálem ACK a poté mikrokontrolér zastaví komunikaci.

Při **čtení jednoho bytu** mikrokontrolér odešle znovu startovní podmínku a poté byte s adresou zařízení s bitem sdělujícím zařízení, že z něj chceme číst (logická „1“). Zařízení odpoví signálem ACK a na následujících 8 náběžných hran odešle mikrokontroléru byte, který si žádal. Mikrokontrolér již příjem nepotvrzuje a následně ukončí komunikaci.

Čtení a zápis více bytů zároveň spočívá v automatickém posuvu na následnou adresu registru zařízení bez nutnosti znovu začínat komunikaci. Tuto vlastnost v mém projektu nevyužiji, protože registry, ze kterých čtu a zapisuji do nich, nejsou na po sobě jdoucích adresách. Pokud bychom chtěli využít celý 10bitový rozsah přesnosti, bylo by vhodné

používat mnohonásobné čtení pro získávání dat z registrů s hodnotami přetížení. Ty jdou po sobě od adresy 0x32 do 0x37.

2.1.1 Zdrojový kód funkce pro inicializaci I2C[1]:

```
void i2c_init()
      {
            TWSR = 0;                                  //bez děličky
            TWBR = ((8000000/100000)-16)/2;      //hodnota TWBR musí být
                                                       //pro správnou funkci
                                                       //více než 10
      }
```

Hodnoty TWSR a TWBR určíme podle upraveného vztahu z datasheetu.

Pro TWSR=0 platí:

$$TWBR = \frac{fCLK}{2 * fSCL} - 8$$

Jak lze vidět z kódu, využívám standardní frekvenci SCL 100 kHz. Akcelerometr podporuje až 400 kHz, ale já nepoužívám součástky na hranici funkčnosti, pokud to není nutné.

2.1.2 Zdrojový kód funkce pro inicializaci Akcelerometru:

```
int i2c_start(unsigned char address, unsigned char dir)
{
      // pošle start podmínku
      TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

      // čeká na její přijetí
      while(!(TWCR & (1<<TWINT)));

      // kontrola TWSR
      if (((TWSR & 0xF8) != TW_START) && ((TWSR & 0xF8) != TW_REP_START))
return 1;

      // odešle adresu zařízení s určením směru přenosu dat
      address = address<<1;
      TWDR = address | dir;
      TWCR = (1<<TWINT) | (1<<TWEN);

      // čeká na dokončení přenosu
      while(!(TWCR & (1<<TWINT)));

      // kontrola potvrzení ACK
      if (((TWSR & 0xF8) != TW_MT_SLA_ACK) && ((TWSR & 0xF8) !=
      TW_MR_SLA_ACK)) return 2;

      return 0;
}
```

[1] Převážná část kódu v jazyce C byla převzata z knihovny I2C a UART [1], některé části byly editovány autorem. Tento fakt platí i pro následující zdrojové kódy, které textová forma práce obsahuje.

```
int i2c_write(unsigned char data)
{
      //odešle data na předtím adresované zařízení
      TWDR = data;
      TWCR = (1<<TWINT)|(1<<TWEN);

      //čeká na dokončení odeslání
      while(!(TWCR & (1<<TWINT)));

      //zkontroluje stav TWSR
      if((TWSR & 0xF8) != TW_MT_DATA_ACK) return 1;
      return 0;
}

int i2c_stop(void)
{
      //odešle stop podmínku
      TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);

      //počká na dokončení
      while(!(TWCR & (1<<TWSTO)));

      if ((TWSR & 0xF8) != TW_SR_STOP) return 1;

      return 0;
}


void init_akcelerometr(void)
{
            i2c_start(0xA6,I2C_WRITE);    // zarovnání zleva
            i2c_write(0x31);
            i2c_write(0x04);
            i2c_stop();
            _delay_us(50);

            i2c_start(0xA6,I2C_WRITE);    // zpomalení pro snížení
                                          // spotřeby
            . . .
}
```

Můžeme si povšimnout návratových hodnot v jednotlivých funkcích. Ty jsou určeny pro ladění programu.

U ADXL345 máme možnost komunikovat i přes rychlejší sběrnici SPI, avšak tuto variantu jsem v projektu nepoužil vzhledem k tomu, že pokud by někdo chtěl projekt rozšířit o další akcelerometry (např. senzory celé paže), nastal by problém s komunikací s nimi. Pro sběrnici SPI je pro každé zařízení na ni připojené nutné mít jeden volný pin na mikrokontroléru jako „chip select“. U I2C sběrnice tato možnost odpadá díky tomu, že každé zařízení na ni připojené má svoji vlastní adresu, na kterou se před zahájením komunikace adresujeme.

2.2 Komunikace s flex senzory

Atmega8 má v pouzdře PDIP pouze 6 multiplexovaných kanálů určených pro A/D převod a to jen pokud nevyužíváme piny pro komunikaci po sběrnici I2C. Ke komunikaci mezi flex senzory, vzhledem k jejich počtu, používám externí multiplexor. Tím multiplexuji postupně všech 10 výstupů z děličů na jeden kanál A/D převodníku. Tento multiplexor nám vnáší další drobnou chybu do výsledku A/D převodu. A/D převodník Atmega8 zatěžuje při snímání napětí zdroj signálu proudem ±3 μA. Multiplexor má v otevřeném kanále odpor přibližně 1,8 kΩ. To nám vytváří chybu v napětí při průchodu proudu multiplexorem cca ±5,4mV. Tato chyba, které bychom se velmi složitě zbavovali, nám později určuje, že nemá smysl se snažit získávat data z A/D převodníku při maximálním bitovém rozlišení (10bitové rozlišení pro 5 V určuje kvantizační krok cca 5 mV).

Zdrojový kód funkce pro inicializaci A/D převodníku:

```
voidinit_adc(void)
{
      ADMUX = (1<<REFS0)|(1<<ADLAR);      //referenční napětí =Vcc, //kanal
                                          0 (PC0), zarovnání //doleva
      ADCSRA = 0xC8;                 //zapnutí A/D,start převodu,povolení
                                     //přerušení
}
```

Zarovnání vlevo využívám, protože pro moji aplikaci mi dostačuje přesnost jednoho byte. Později pak v programu vyhodnocuji jen horní byte z výsledku převodu. Atmega8 má maximální přesnost A/D převodu 10bit.

2.3 Komunikace mikrokontroléru s počítačem

Většina práce na projektu proběhla s přímým připojením mikrokontroléru k počítači po sériové lince USART (Universal Synchronous / AsynchronousReceiever and Transmitter) v asynchronním režimu. Počítač může tímto rozhraním získat data po úpravě napěťových úrovní standardním sériovým portem. Jelikož u nových počítačů sériové rozhraní mizí a na noteboocích ho téměř vůbec nenalezneme, vydal jsem se cestou využití modulu s převaděčem na USB. Softwarově se jeví toto zařízení podobně jako sériový port a práce s ním je obdobná.

Kvůli využívání sériového přenosu dat je nutné odlišit, jaká data patří kterému senzoru. Využil jsem zde velmi jednoduchou metodu vyslání několika počátečních kódových bytů, díky kterým bude následná série příchozích bytů seřazena podle nastavené sekvence

odeslání dat z mikrokontroléru. Kódové byty mají hodnotu větší než 0xF9 a je programově zabráněno mikrokontroléru vysílat data, která by měla tuto hodnotu. Z těchto šesti kódových bytů využívám pouze jeden a ostatní jsou připraveny pro další využití v upravených aplikacích (např. signalizace chybného převodu A/D převodníku). Data, která by měla dosahovat hodnot kontrolních bytů, jsou saturačně upravena na nejbližší hodnotu (např. při postupném zvyšování zrychlení budou odeslaná data vypadat následovně: 0xF7->0xF8->0xF9->0xF9->0xF9).

Mikrokontrolér má nastavenou rychlost přenosu na standardních 9600 Bd. Rámec je složen také typicky 8N1 (8 datových bitů, bez paritního bitu, jeden stop bit). USART umožňuje nastavit rámec i na 9 datových bitů, není to však příliš standardní pro přenos dat. V případě potřeby lze moji práci případně později upravit nebo vylepšit co se týká přesnosti získávaných dat ze senzorů. Je možné například optimalizovat datový přenos pro tuto aplikaci.

Zdrojový kód funkce pro inicializaci USART:

```
void init_uart(void)
{
      UBRRH = 0;
      UBRRL = 51;                          //rychlost 9600
      UCSRB = (1<<TXEN);                   //zapni vysílač
      UCSRC = (1<<URSEL)|(3<<UCSZ0);       //formát rámce: 8N1
}
```

Nastavení rychlosti podle frekvence CLK lze provést pomocí dvojice registrů UBRRH a UBRRL, které tvoří za sebou šestnácti bitový registr UBRR. Hodnotu UBRR lze pro normální mód asynchronního přenosu vypočítat dle vztahu uvedeného v datasheetu mikrokontroléru [3]:

$$UBRR = \frac{fCLK}{16 * BAUD\_RATE} - 1$$

Následně stačí získanou hodnotu UBRR rozdělit do registrů UBRRH a UBRRL. Je velmi pravděpodobné, že číslo, které jsme vypočetli podle daného vztahu, nebude číslo přirozené. Budeme ho muset zaokrouhlit, což nám vytvoří určitou chybu. Nicméně já jsem zvolil ještě jednodušší metodu. V datasheetu lze u Atmega8 dohledat tabulky pro standardní frekvence krystalových rezonátorů [3]. V nich pouze určíme žádanou rychlost přenosu v Baudech a vyčteme z nich hodnotu UBRR a relativní zaokrouhlovací chybu.

Cílem přenosu mezi mikrokontrolérem a počítačem se ovšem stal přenos bezdrátový, aby se člověk pracující s rukavicí mohl volně pohybovat v dosahu bezdrátového přenosu. Existuje několik variant pro bezdrátové propojení mezi zařízeními. Pro mou aplikaci jsem zvolil přenos standardem bluetooth, který je v poslední době velmi využívaný. Bluetooth je na trhu dostupný i jako modul, díky čemuž se zbavíme téměř veškeré práce s ním. Na vstupní piny je nutné připojit jen napájení a vstup a výstup datové (RxD a TxD) komunikace USART z mikrokontroléru. Tímto se stává senzorická rukavice kompatibilní, co se týče komunikace s různými typy zařízení. Samozřejmě, že díky bezdrátové komunikaci je nutné zajistit alternativní zdroj energie. V mém případě postačila baterie tří článků AA.

3 Konstrukce zařízení

Senzorický systém se skládá ze dvou hlavních částí - desky plošných spojů a senzorické rukavice. DPS je základem pro mikrokontrolér, multiplexor, modul bluetooth a zbytek periferních obvodů. Tyto obvody jsou spojeny vodiči se všemi senzory připevněnými na rukavici přes konektor typu DB15, aby bylo možné jednoduše odpojit rukavici od desky plošných spojů. DPS je při používání umístěna na předloktí spolu s bateriemi. Senzorická rukavice snímá polohu ruky pomocí senzorů. Výsledné signály udávající polohu ruky jsou vyvedeny vodiči k DPS.

3.1 Konstrukce DPS pro mikrokontrolér a periferní obvody

Pro DPS jsem volil kombinované osazení THT a SMT, a to především kvůli ceně, jelikož patice určené pro součástky THT s vývody SMT jsou na trhu špatně dostupné a drahé. Nemám totiž možnost naprogramovat mikrokontrolér přímo na desce a také z praktického hlediska je možnost výměny mikrokontroléru výhodná. Pro mikrokontrolér jsem tedy volil umístění do patice THT. Samozřejmě lze namítnout, že toto řešení je značně nemoderní. Tento názor asi nedokáži vyvrátit, ale mým cílem není postavit nejmodernější rukavici s maximálně miniaturizovanými součástkami pracujícími při maximálním výkonu, které jsou určeny pro sériovou výrobu.

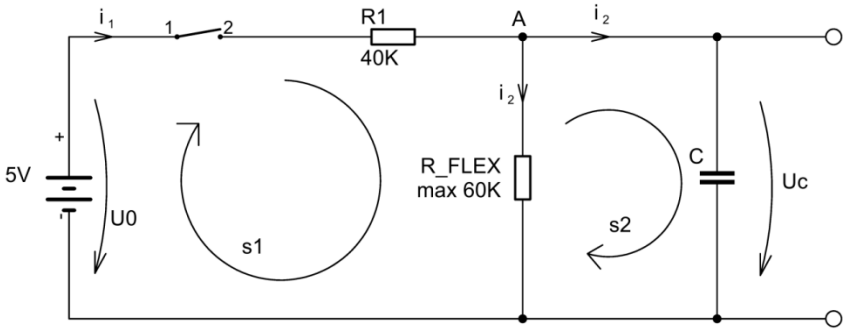
3.1.1 Popis schématu DPS

Součástky mají většinou podobnou funkci jako v obdobných zařízeních, ve kterých se používají, ale pro přehlednost a rychlejší pochopení zde uvedu stručný seznam využití v mém

zapojení. Rezistory R1 a R2 tvoří dělič napětí na polovinu, toto napětí je přivedeno na multiplexor MPC506AP jako klopné napětí pro logické vstupy v souladu s hladinami CMOS technologií mikrokontroléru (viz Obr. 11). Rezistory R3 až R12 tvoří dělič napětí s flex senzory (ty jsou připojeny na konektor) a výstupní napětí děliče je přes multiplexor propojeno na A/D převodník mikrokontroléru. Flex senzory jsou připojeny na desku vodiči zakončenými na schématu v konektoru. Odpory R14 a R15 slouží jako „pull up“ rezistory pro komunikaci po I2C sběrnici. Tlačítko S2 a dioda LED1 slouží pro základní komunikaci a indikaci stavu mikrokontroléru. Kondenzátory C3 a C15 slouží jako blokovací kondenzátory pro mikrokontrolér a multiplexor.

3.1.2 Volba kapacity filtračního kondenzátoru

Tento kondenzátor slouží jako vstupní filtr typu dolní propust, který potlačuje okolní rušení navázané do vodičů k flex senzorům. Kondenzátory jsou na desce navrženy jako pojistka a jak se později ukázalo, nebylo je do běžně rušeného prostředí nutné využít.



Obr. 9 Schéma obvodu s děličem

Zdroj: Autor

$$A: \quad I_1 - I_2 - I_3 = 0$$

$$s_1: \quad I_1 R_1 + I_2 R_{FLEX} = U_0$$

$$s_2: \quad u_c - I_2 R_{FLEX} = 0$$

$$s_2: \qquad I_2 = \frac{u_c}{R_{FLEX}}$$

$$A + s_2: \qquad I_1 = \frac{u_c}{R_{FLEX}} + C\frac{du_c}{dt}$$

$$A + s_1 + s_2: \qquad R_1\left(\frac{u_c}{R_{FLEX}} + C\frac{du_c}{dt}\right) + R_{FLEX}\frac{u_c}{R_{FLEX}} = U_0$$

$$A + s_1 + s_2: \qquad \frac{du_c}{dt} - u_c\frac{R_1 + R_{FLEX}}{C * R_1 * R_{FLEX}} = \frac{U_0}{C * R_1}$$

Předpokládané řešení diferenciální rovnice prvního řádu:

$$u_c(t) = K * e^{\lambda * t} + u_c(\infty)$$

$$\lambda + \frac{R_1 + R_{FLEX}}{C * R_1 * R_{FLEX}} = 0$$

$$\lambda = -\frac{R_1 + R_{FLEX}}{C * R_1 * R_{FLEX}}$$

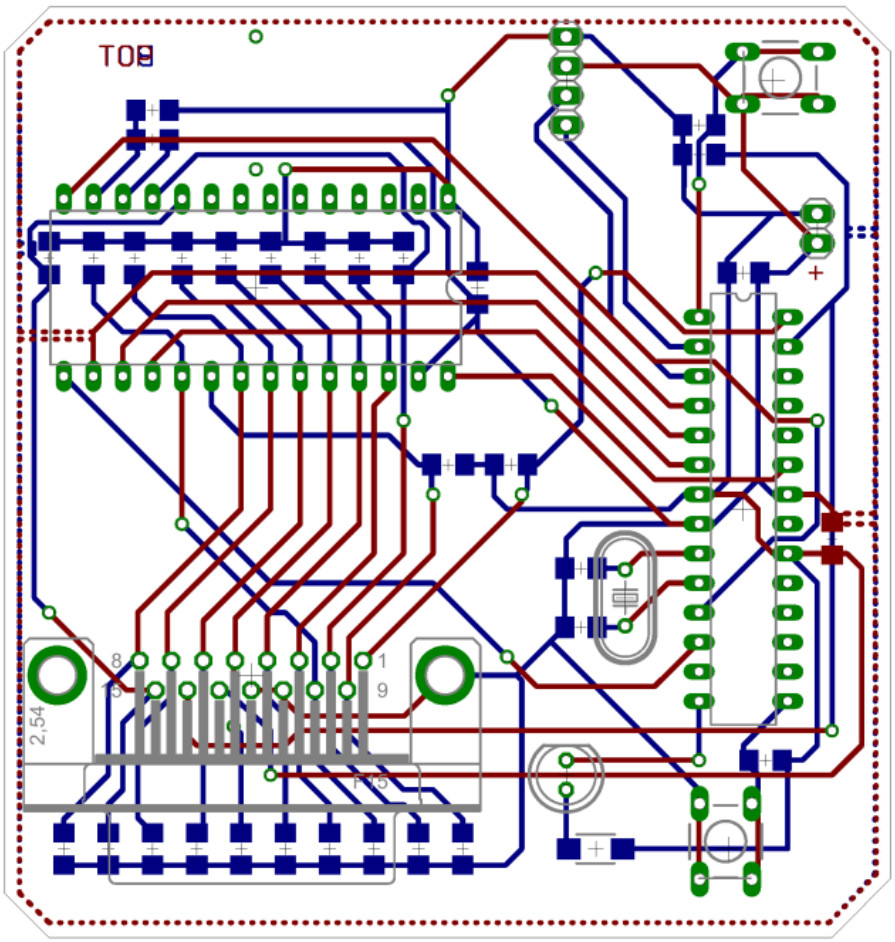
$$\tau = -\frac{1}{\lambda} = \frac{R_1 * R_{FLEX}}{R_1 + R_{FLEX}} C\ [s]$$

Jako ustálenou hodnotu lze považovat 3τ, což odpovídá přibližně 95 % ustálené hodnoty v čase blížícímu se nekonečnu. Pro tuto aplikaci byla zvolena na 0,01 sekundy.

$$3\tau = 0{,}01 = \frac{40 * 10^3 * 60 * 10^3}{40 * 10^3 + 60 * 10^3} C$$

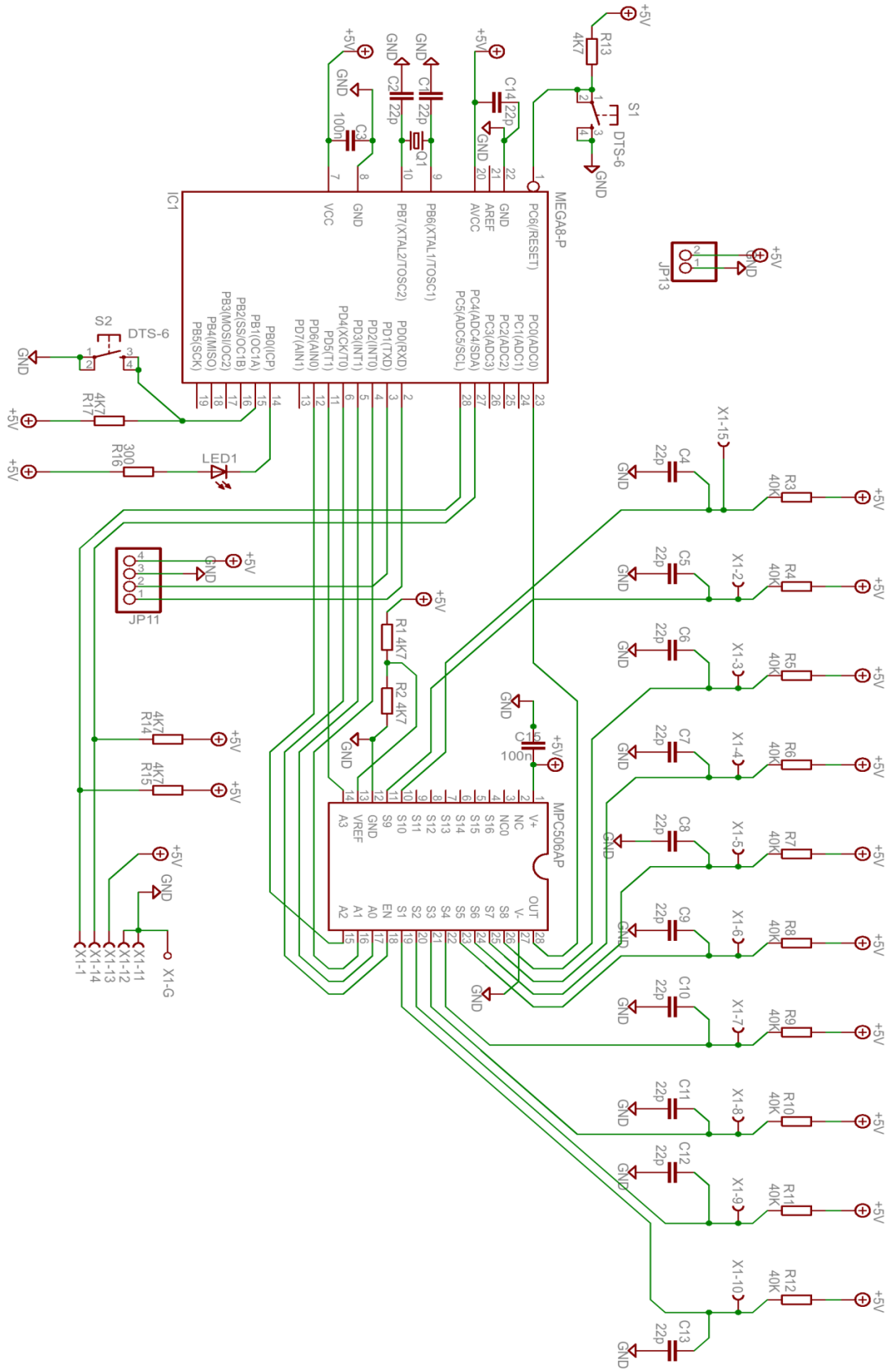
$$C = 417[nF]$$

Pokud vybereme nejbližší kondenzátor z řady (470 nF), získáme mírně vyšší čas do „ustálení“ (0,013 s).



Obr. 10 Návrh desky plošných spojů (bez rozlité mědi)

Zdroj: Autor



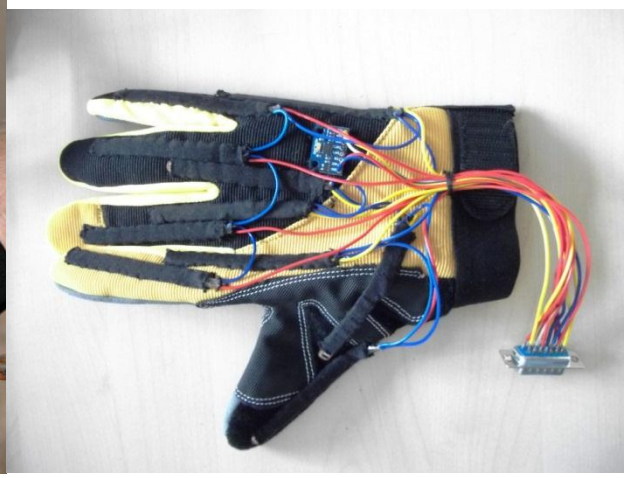
Obr. 11 Schéma desky plošných spojů

Zdroj: Autor

3.2 Konstrukce rukavice

Nalézt na trhu rukavici určenou přímo k tomuto účelu je poněkud problém. Rukavice použitá na projektu je k dostání ve standardním obchodním řetězci pro kutily a zahrádkáře, v sekci ochranných pomůcek (viz Obr. 12). Protože předpokládám delší a častější používání rukavice, volil jsem podle parametrů odolnost, cena, větrání, přiléhavost a vzhled rukavice.

Obr. 12 Rukavice před úpravou



Obr. 13 Rukavice osazená senzory

Zdroj: Obr. 12 http://www.albo.biz/en-us/Free-Hand-gloves/Free-Hand-gloves/Corax Obr. 13 Autor

Jak již bylo zmíněno, rukavice není určená přímo pro tento typ manipulace, a proto jsem ji musel upravit. Nejdůležitější úpravou rukavice je umístění již zmíněných kapsiček pro flex senzor (viz kapitola 1.4). Vhodná volba materiálu pro kapsičky není až tak triviální jak se na první pohled může zdát. Je totiž nutné zajistit, aby se kapsička mohla natahovat s ohybem kloubů na prstech a zároveň, aby nebyla příliš pružná do kolmého směru na ohyb, tj. aby flex senzor „neutíkal“ do stran. Tyto podmínky byly splněny využitím klasického bavlněného materiálu na výrobu oděvů. Při sešívání kapsiček z látky jsem volil klikatý steh (z pohledu zvnějšku vypadá jako trojúhelníkový signál) a pomalejší posuv, aby byla zachována pružnost při napínání. Kapsičky jsou umístěny vedle sebe, aby se zjednodušil přístup a zvýšila ochrana před mechanickým poškozením flex senzoru (viz Obr. 13). Myslím, že by bylo možné umístit senzory i nad sebe, ale obavy o již zmíněnou ochranu senzorů mě dovedli k řešení umístění vedle sebe.

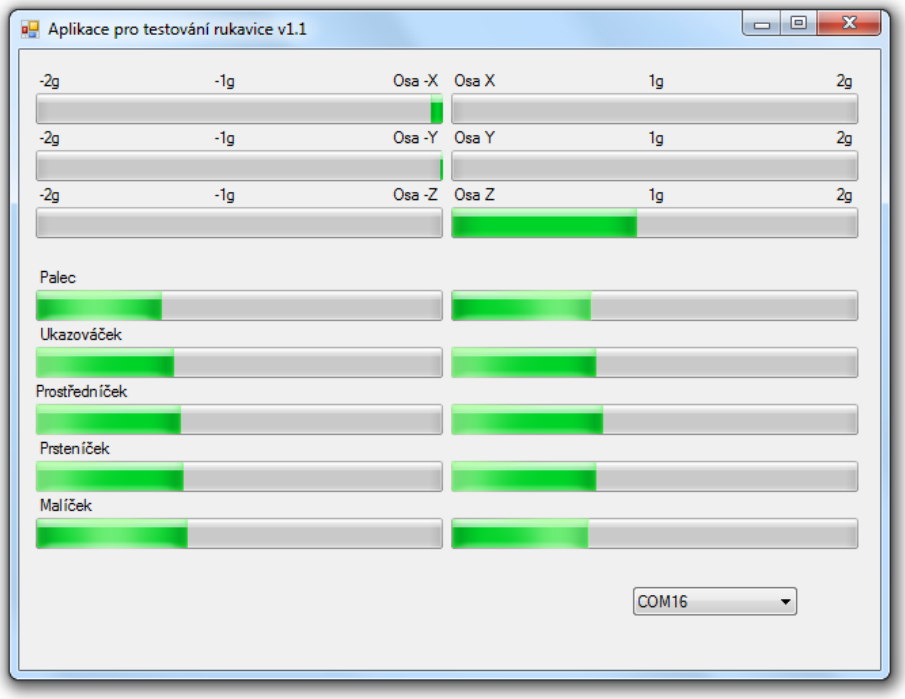
Rukavice má z výroby připevněné výztuže pro ochranu kloubů. Jedna z nich byla odpárána, protože při ohybu prstů k dlani bránila plynulému pohybu.

4 Návrh programového vybavení pro senzorickou rukavici

Rukavice je vytvořena především tak, aby s ní bylo možné řídit co největší počet systémů. Její využití proto není téměř omezené. Možnosti využití takovéto rukavice sahají od ovládání jednoduchých aplikací po složitější software typu vývojových 3D programů, ale lze ji využít také například pro ovládání her. Další zajímavou možností je dálkové ovládání robota. Nejlépe se rukavice hodí pro ovládání robota, který má podobné pohyby, jako jsou pohyby lidské ruky (například robotická ruka). Díky dostatečnému množství výstupních stavů ohybu kloubu lze toto řízení považovat téměř za plynulé.

Aplikaci jsem volil jednodušší kvůli omezenému časovému prostoru. Moje aplikace slouží především pro demonstraci funkce rukavice. Její praktické využití spočívá převážně v doladění správné funkce rukavice. Pro větší jednoduchost při práci s grafickou aplikací jsem využil programovací jazyk C# a strukturu Framework.

Aplikace je složená z převážné části z ukazatelů průběhu (angl. ProgressBar), známých především z instalací programů. Mé využití je poněkud odlišné od tohoto standardního využití. Jednotlivé ukazatele průběhu zobrazují velikost ohybu kloubu nebo velikost přetížení na osu (viz Obr. 14). Výběrové pole (angl. ComboBox) slouží pro výběr sériového portu, po kterém probíhá komunikace přes bluetooth.



Obr. 14 Vzhled aplikace

Zdroj: Autor

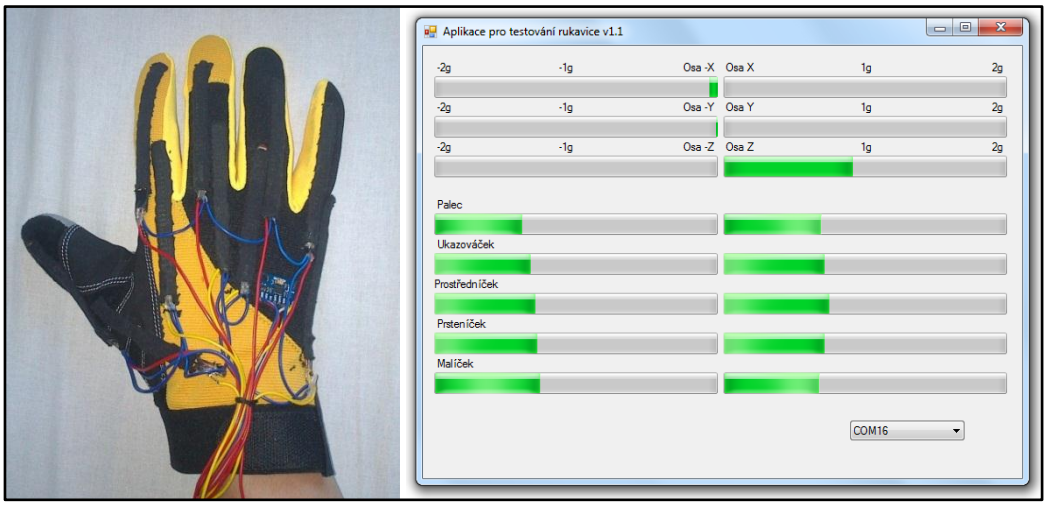
4.1 Hlavní část programu pro grafický výstup:

```
private void zobraz(object o, EventArgs e)
{
    i++;
    if (((int)x[0] == 0xFF)  )//proměnná x[0] uchovává stav příchozího bytu
    {
        if (j == 0xFF)
        { i = 0; }
    }
    j = (int)x[0];       //do proměnné j ukládám stav příchozího byte
                         //a při následujícím průchodu testuji, jestli
                         //nebyly vyslané synchronizační byty
switch (i)               //proměnná i uchovává postupný posuv při příchodu
                         //bytu
    {
            Case 0: break;          // v případě příchodu 0xFF proskočení
                                    //switchem
            case 1:
            if (((int)x[0]) > 127)  //rozdělení rozsahu přetížení na kladné
                                    //a záporné pro osu X
            {
                progressBar4.Value = 255 - (int)x[0];
                progressBar1.Value = 0;
            }
            else
            {
                progressBar1.Value = (int)x[0];
                progressBar4.Value = 0;
            }
            break;


            case 2:
            if (((int)x[0]) > 127)  //rozdělení rozsahu přetížení na
                                    //a záporné pro osu Y
            {
                progressBar5.Value = 255-(int)x[0];
                progressBar2.Value = 0;
            }
            else
            {
                progressBar2.Value = (int)x[0];
                progressBar5.Value = 0;
            }
            break;

            case 3:
            if (((int)x[0]) > 127) //rozdělení rozsahu přetížení na kladné
                                    //a záporné pro osu X
            {
                progressBar6.Value = 255 - (int)x[0];
                progressBar3.Value = 0;
            }
            else
            {
                progressBar3.Value = (int)x[0];
                progressBar6.Value = 0;
            }
            break;
```

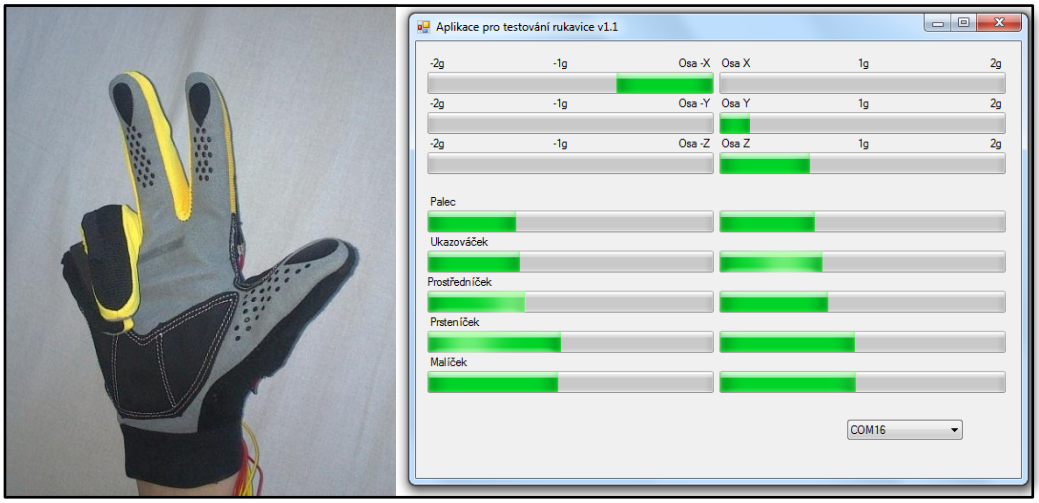
```
                case 4: progressBar7.Value = (int)x[0]; //zápis hodnoty od flex
                                                        //senzoru
                        break;
                case 5: progressBar8.Value = (int)x[0]; //zápis hodnoty od flex
                        break;                          //senzoru
                .
                .
    }
}
```

4.2 Ukázky poloh ruky v rukavici a jejich vykreslení v aplikaci

Z následujících obrázků je patrné, že změna napětí na děliči se nepohybuje od nuly k 5 V (viz Obr. 15 až Obr. 19). V opačném případě by se musel odpor flex senzoru měnit od nuly k nekonečnu (podrobněji viz kapitola 1.1.1). Nicméně tato změna je pro řízení většiny aplikací dostatečná. Když uvážíme, že celý ProgressBar je rozdělen na 256 hodnot, je i z obrázků patrné, že většímu ohybu kloubu připadá změna cca 50 hodnot.

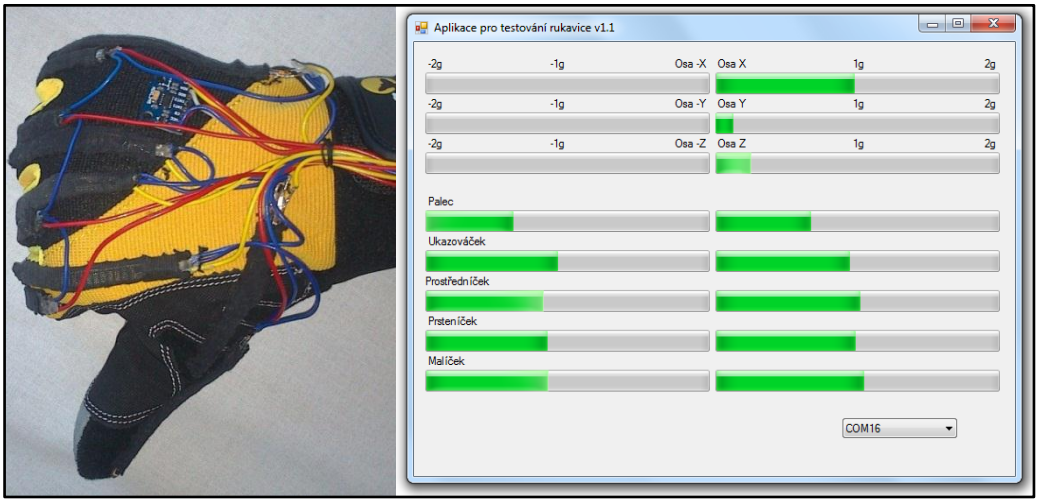


Obr. 15 Poloha narovnaných prstů



Obr. 16 Poloha tří narovnaných prstů

Obr. 17 Palec vztyčený vzhůru



Obr. 18 Palec směřuje k zemi

Obr. 19 Ukázka ohnutých kloubů na prstech, bez ohnutí celého prstu

Zdroj Obr. 14 až Obr. 18: Autor

Závěr

Práce se zabývala návrhem a realizací senzorického systému. Součástí práce byla vlastní konstrukce systému v podobě senzorické rukavice. V písemné části byly popsány jednotlivé součásti senzorického systému, komunikace s nimi a byl popsán postup realizace. Dále bylo diskutováno možné využití systému.

Volba vhodných senzorů pro systém je popsána v první kapitole. Toto řešení se ukázalo jako velmi vhodné pro danou problematiku, ale po praktických zkušenostech s těmito senzory bych chtěl doporučit zvýšenou opatrnost při manipulaci a instalaci především flex senzorů. Senzory využívané v mé práci jsou dostatečné pro základní ovládání systému, ale pro zlepšení by bylo nutné přiřadit k akcelerometru ještě jiný nezávislý snímač, který by měřil natočení. Tyto snímače by se navzájem kompenzovaly a data získaná z této kombinace by byla mnohem komplexnější.

Na komunikaci s řízeným systémem a získávání dat ze senzorů jsem se zaměřil v druhé kapitole. Cílem práce není součástky zatěžovat maximálním možným výkonem, ale zajistit funkční komunikaci mezi senzory a řízeným systémem.

Následná kapitola je zaměřena na konstrukci senzorické rukavice a obvodů důležitých pro její funkci. Je zde ukázán jeden z možných způsobů jak k této problematice přistupovat. Základ celého zde popisovaného systému tvoří deska plošných spojů, na kterou je přes konektor připojena senzorická rukavice. Při tvorbě desky plošných spojů byl také kladen důraz na základy elektromagnetické kompatibility zařízení.

V poslední kapitole o využití senzorického systému je popsána tvorba jednoduché aplikace pro otestování rukavice. Senzorický systém má velmi málo omezení a umožňuje tak velkou variabilitu využití tohoto systému.

Systém byl realizován prakticky podle popisu v daných kapitolách. Byla také provedena zkouška funkčnosti celého systému rukavice, která proběhla úspěšně. Tímto je systém senzorické rukavice připraven pro další práce, které mohou plně využít jeho potenciál.

Použitá literatura

[1] ZÁVODSKÝ, Ondrej. [online]. 2012, s. 66 [cit. 2013-06-03]. Dostupné z: http://svetelektro.com/clanky/kniha-programujeme-avr-v-jazyku-c-557.html

[2] SHARP, John. Microsoft Visual C# .NET krok za krokem. 1. vyd. Brno: Mobil Media, 2002, 655 s. ISBN 80-865-9327-4.

[3] ATMEL CORPORATION. ATmega8(L). 2013, 325 s. [online]. Dostupné z: http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf [cit. 2013-06-03]

[4] ANALOG DEVICES. ADXL345. [online]. 2013, s. 40 [cit. 2013-04-012]. Dostupné z: http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

[5] Spectrasymbol. Flex senzor. [online]. s. 2 [cit. 2013-01-016]. Dostupné z: https://www.sparkfun.com/datasheets/Sensors/Flex/flex22.pdf

Přílohy

Příloha A – Hlavní část programu pro mikrokontrolér

Zdroj pro knihovny uart.h a i2c_master.h [3]

```
# define F_CPU 8000000UL

#include <avr/io.h>
#include <avr/delay.h>
#include <avr/interrupt.h>
#include "i2c_master.h"
#include "uart.h"

static unsigned char fl[11],x,y,z,mux=0;

void init_akcelerometr(void)
{
            i2c_start(0xA6,I2C_WRITE);    // zarovnání zleva
            i2c_write(0x31);
            i2c_write(0x04);
            i2c_stop();
            _delay_us(50);

            i2c_start(0xA6,I2C_WRITE);    // zpomalení pro nižší spotřebu
            i2c_write(0x2C);
            i2c_write(0x07);
            i2c_stop();
            _delay_us(50);


            i2c_start(0xA6,I2C_WRITE);    // nulový offset
            i2c_write(0x1E);
            i2c_write(0x0);
            i2c_stop();
            _delay_us(50);


            i2c_start(0xA6,I2C_WRITE);    //zapnutí měření
            i2c_write(0x2D);
            i2c_write(0x08);
            i2c_stop();
}

ISR(ADC_vect)
{
      unsigned char ad;
      ad=(char)(ADC>>2);           // zápis 8 horních bitů výsledku převodu


      switch(mux)
      {
            case 1: PORTD= 0x10 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 2: PORTD= 0x18 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 3: PORTD= 0x14 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
```

```
            case 4: PORTD= 0x1C | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 5: PORTD= 0x50 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 7: PORTD= 0x58 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 9: PORTD= 0x54 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 6: PORTD= 0x5C | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 8: PORTD= 0x30 | (PORTD & 0x01);
                        fl[mux]=ad;
                        break;
            case 10: PORTD= 0x38 | (PORTD & 0x01);
                        fl[mux]=ad;
                        mux=0;
                        break;

            default:mux=0;
                        break;
      }
      mux++;
}

void cti_akcelerometr(void)
      {
            unsigned char c,a;
            for(c=0x33;c<0x38;c=c+2) // načte hodnoty přetížení po I2C do
                                     //x,y,z
                  {
                        i2c_start(0xA6,I2C_WRITE);
                        i2c_write(c);
                        i2c_start(0xA6,I2C_READ);
                        a=i2c_read_nAck();
                        i2c_stop();
                        _delay_ms(5);
                        if(a>0xF9)a=0xF9;
                        switch(c)
                        {
                              case 0x33: x=a;break;
                              case 0x35: y=a;break;
                              case 0x37: z=a;break;
                        }

                  }
      }

int main(void)
{
      unsigned char poc1;
      _delay_ms(3);
      DDRB=0xFF;
      PORTB=0x00;
      PORTC= 0x30;
      DDRD=0xFF;
      PORTD=0x00;
```

```
	UBRRH = 0x00;
	UBRRL = 51;
	UCSRB = (1<<TXEN);                  //zapni vysílač
	UCSRC = (1<<URSEL)|(3<<UCSZ0);      //nastav formát rámce: 8data,
                                        //1stop bity

	ADMUX = 0x40;                       // referenční napětí REFS0=>Vcc,
                                        // kanál 0 (PC0)
	ADCSRA |= 0xEF;                     // povolení činnosti a/d, start
                                        //převodu
                                        // opakovací režim, přeruš,
                                        //clk/128


	i2c_init();                         // inicializace i2c rozhraní
	_delay_ms(3);
	init_akcelerometr();

	sei();
	while(1)
	{
		_delay_ms(20);              //zpomalení pro snížení spotřeby
		cti_akcelerometr();
		uart_putc(0xFF);            //postupné odeslání znaků v hlavní
                                        //smyčce
		uart_putc(0xFF);
		uart_putc(0xFF);
		uart_putc(x);
		uart_putc(y);
		uart_putc(z);
		uart_putc(fl[3]);           //seřazení znaků do logické
                                        //posloupnosti
		uart_putc(fl[2]);
		uart_putc(fl[5]);
		uart_putc(fl[4]);
		uart_putc(fl[8]);
		uart_putc(fl[6]);
		uart_putc(fl[10]);
		uart_putc(fl[7]);
		uart_putc(fl[1]);
		uart_putc(fl[9]);
	}
}
```

Příloha B – Hlavní část programu aplikace v C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication4
{
public partial class Form1 : Form
{
```

```
public Form1()
{
    InitializeComponent();
}

byte[] x = new byte[1];
private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
{
    while (serialPort1.BytesToRead > 0)//vyčítej obsah bufferu dokud není
prázdný
    {
        serialPort1.Read(x, 0, 1);
        this.Invoke(new EventHandler(zobraz));
    }
}

private int i=0,j=0;
private void zobraz(object o, EventArgs e)
{

    i++;
    if (((int)x[0] == 0xF0)  )      //proměnná x[0] uchovává stav
příchozího bytu
    {
        if (j == 0xF0)
        { i = 0; }
    }
    j = (int)x[0];                  //do proměnné j ukládám stav příchozího
                                    //byte
                                    //a při následujícím průchodu testuji,
                                    //jestli
                                    //nebyly vyslané synchronizační byty

    switch (i)                      //proměnná i uchovává postupný posuv
                                    //při příchodu
                                    //bytu
    {
        case 0: break;              // v případě příchodu 0xF0 proskočení
                                    //switchem
        case 1:
            if (((int)x[0]) > 127)  //rozdělení rozsahu přetížení na kladné
                                    //a záporné
                                    //pro osu X
            {
                progressBar4.Value = 255 - (int)x[0];
                progressBar1.Value = 0;
            }
            else
            {
                progressBar1.Value = (int)x[0];
                progressBar4.Value = 0;
            }
            break;


        case 2:
           if (((int)x[0]) > 127)  //rozdělení rozsahu přetížení na
                                   //kladné a záporné
                                   //pro osu Y
            {
```

```
                progressBar5.Value = 255-(int)x[0];
                progressBar2.Value = 0;
            }
            else
            {
                progressBar2.Value = (int)x[0];
                progressBar5.Value = 0;
            }
            break;

        case 3:
            if (((int)x[0]) > 127)  //rozdělení rozsahu přetížení na kladné
                                    //a záporné
                                    //pro osu X
            {
                progressBar6.Value = 255 - (int)x[0];
                progressBar3.Value = 0;
            }
            else
            {
                progressBar3.Value = (int)x[0];
                progressBar6.Value = 0;
            }
            break;
            case 4: progressBar7.Value = int)x[0]; //zápis hodnot
                                                   // od flex senzorů
                    break;
            case 5: progressBar8.Value = (int)x[0];
                    break;

            case 6: progressBar9.Value = (int)x[0];
                    break;
            case 7: progressBar10.Value = (int)x[0];
                    break;                              //konstanty
                            //kompenzující snížený ohyb některých senzorů
            case 8: progressBar11.Value = (int)(x[0] * 1.1);
                    break;
            case 9: progressBar12.Value = (int)x[0];
                    break;
            case 10:progressBar13.Value = (int)x[0];
                    break;
            case 11:progressBar14.Value = (int)(x[0] / 1.07);
                    break;
            case 12:progressBar15.Value = (int)x[0];
                    break;
            case 13:progressBar16.Value = (int)(x[0]/1.2);
                    break;
    }
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    serialPort1.Close();
    serialPort1.PortName = comboBox1.Text;
    serialPort1.Open();
}
private void Form1_FormClosed(object sender, FormClosingEventArgs e)
{
            serialPort1.Close();
}
}
}
```