ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

Katedra technologií a měření

DIPLOMOVÁ PRÁCE

Systém pro určování pozice modelu letadla

vedoucí práce: **Ing. Kamil Kosturik, Ph.D.**

autor: **Michal Hrách**

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal HRÁCH**
Osobní číslo:	**E10N0022P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Komerční elektrotechnika**
Název tématu:	**Systém pro určování pozice modelu letadla**
Zadávající katedra:	**Katedra technologií a měření**

Zásady pro vypracování:

Navrhněte systém, umožňující pilotovi letadla sledovat polohu modelu.

1. Prostudujte možnosti a způsoby lokalizace polohy modelu letadla.
2. Navrhněte vhodný způsob lokalizace modelu a přenosu dat do vizualizačního systému.
3. Navrhněte způsob vizualitace polohy letadla tak, aby byla využitelná pro tréninky akrobacie.
4. Navržené řešení realizujte.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Kamil Kosturik, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **17. října 2011**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Ing. Vlastimil Skočil, CSc.
vedoucí katedry

V Plzni dne 17. října 2011

Anotace

Tématem předkládané diplomové práce je Systém pro určování pozice modelu letadla. Práce je zaměřena na problematiku vizualizace polohy křídel na PC v průběhu letu. Cílem je návrh a realizace zařízení.

Klíčová slova

Model letadla, akcelerometr, mikroprocesor, program, vysílací a přijímací modul, hardware, schéma zapojení, deska plošných spojů, aplikace, software, náklon, vizualizace, umělý horizont, FTDI převodník, zrychlení, citlivost.

Abstract

The subject of this Graduate thesis is Model airplane attitude sensing and displaying device. The thesis is focused on visualization of the position of wings during flight. The aim of the work is design and construction of the equipment.

.

Key words

Airplane model, accelerometer, microcontroller, program, transmitter and receiver module, hardware, wiring diagram, printed circuit board, application, software, bank, visualization, artificial horizon, FTDI converter, acceleration, sensitivity.

Prohlášení

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

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

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

V Plzni dne 10.5.2012 Jméno příjmení

……..

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Kamilovi Kostrukovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Přehled zkratek

Li-Pol	Lithium-Polymer (akumulátor)
USB	Universal serial bus (sériová sběrnice)
PC	Osobní počítač
MEMS	Micro electro mechanical system
SMD	Surface mount device
A/D	analog/digital
RISC	Reduced instruction set computer
CPU	central processing unit
ALU	Arithmetic logic unit
SPI	Serial peripheral unit
ISP	In-system programming
TTL	Tranzistorově-tranzistorová logika
H	High
L	Low
U_{cc}	Napájecí napětí
U_r	Rozhodovací úroveň napětí
U_{OHmin}	U output high minimum
U_{OLmax}	U output low maximum
U_{IHmin}	U input high minimum
U_{ILmax}	U input low maximum
Kbps	Kilobit za sekundu
FM	Frekvenční modulace
DPS	Deska plošných spojů
BMP	Bitmapa

Přehled zkratek

GND	Zem
EEPROM	Electrical erasable programmable read-only memory
RAM	Random-access memory
UART	Universal asynchronous receiver and transmitter
FTDI	Future technology devices international
ASM	Assembler

Přehled symbolů

f [Hz]	Frekvence
I [A]	Proud
U [V]	Napětí
C [F]	Kapacita
S [m^2]	Plocha
d [m]	Vzdálenost
ε_r [$F.m^{-1}$]	Relativní permitivita
ε_0 [$F.m^{-1}$]	Permitivita vakua
F [N]	Síla
m [kg]	Hmotnost
a [$m.s^{-2}$]	Zrychlení
g [$m.s^{-2}$]	Tíhové zrychlení
P [W]	Příkon
Q [C, Ah]	Elektrický náboj
v_m [Bd]	Modulační rychlost
λ [m]	Vlnová délka
v [$m.s^{-1}$]	Rychlost
l [m]	Délka
g_m [-]	Vektor gravitační síly

Úvod

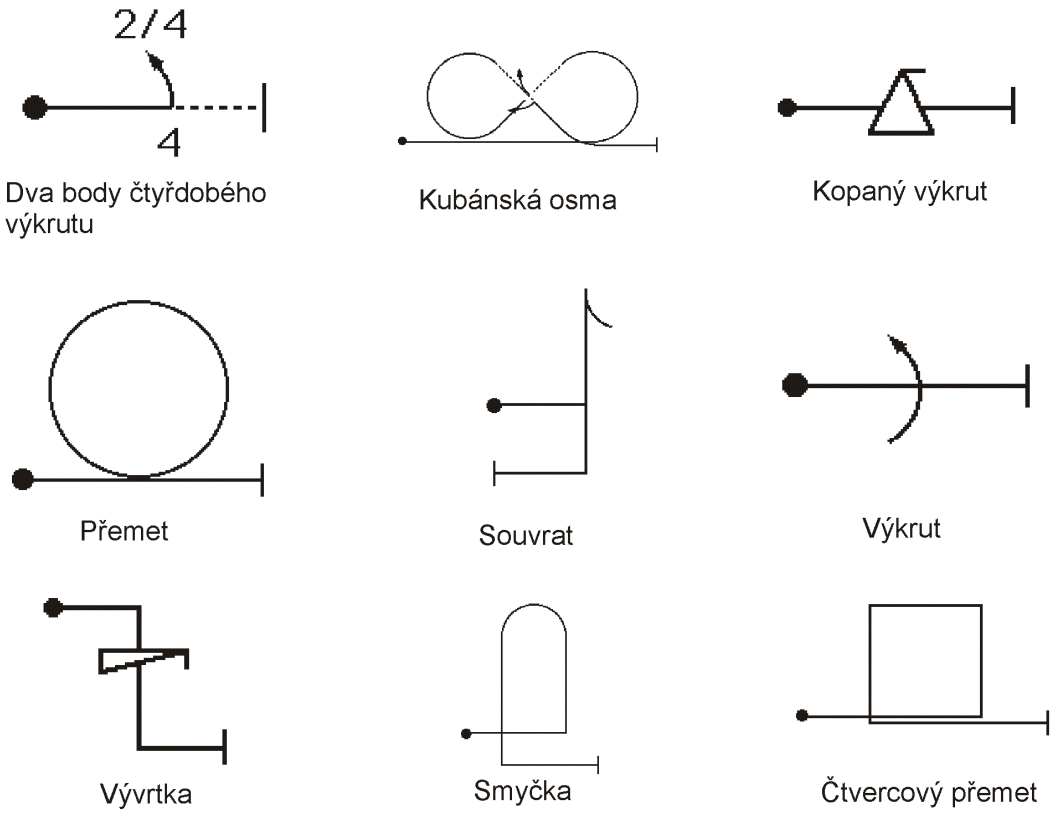
Předkládaná práce je zaměřena na problematiku měření náklonu křídel modelu letadla během letu a grafického online zobrazení PC.

Práce byla prakticky realizována, takže veškeré návrhy a zdrojové kódy programů jsou ověřeny. Dokumentace je rozdělena do pěti částí. První představuje principy létání akrobacie a význam měření náklonu křídel. Druhá kapitola popisuje návrh a funkci jednotlivých bloků zařízení. Ve třetí části jsou představeny hlavní díly hardwaru. Čtvrtá kapitola popisuje schémata zapojení a výrobu desek plošných spojů, včetně seznamů použitých součástek. V závěrečné části je popsán program mikroprocesoru, použitý software a vytvořená grafická aplikace pro zobrazení měřených dat.

1 Létání akrobacie

Při pilotáži akrobatického modelu letadla jsou prováděny akrobatické figury předepsané Arestiho symboly. Ty jsou zkompletovány katalogem Arestiho symbolů, ze kterého se vybírají a vkládají v určeném pořadí do sestavy akrobatických obratů. Sestava se skládá ze středových a krajových figur. Každý z obratů je v katalogu obsažen v několika variantách, protože obraty na sebe musí navazovat. Například, pokud jedna figura končí v horní letové hladině, následující obrat musí začínat ve stejné výšce. Ukázka vybraných Arestiho symbolů je zobrazena na obr. 1.1.

Obr. 1.1 Příklad Arestiho symbolů

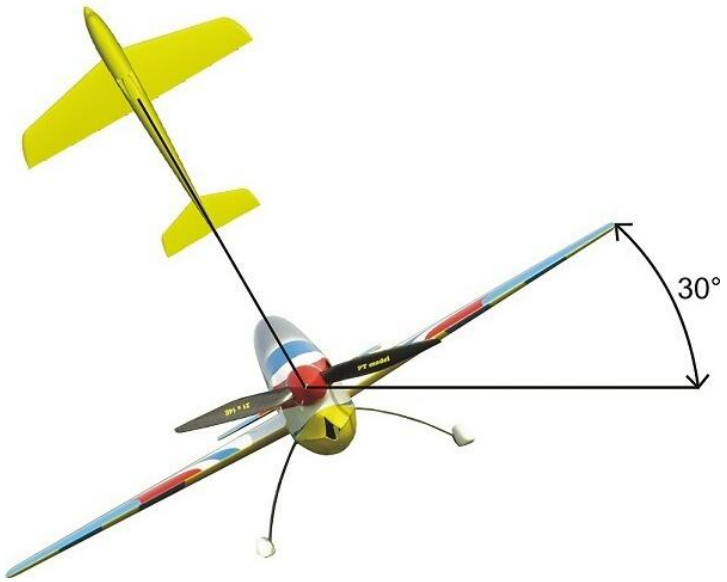


Kvalitu provedení jednotlivých prvků hodnotí rozhodčí známkou 0 až 10. Kritéria bodování jsou přesně definována. Základním předpokladem pro správné provedení obratu je rovný nálet. Pokud rozhodčí vidí, že poloha křídel není vodorovná, je to důvod ke snížení hodnocení. Ale mnohem větší význam poloha křídel nabývá při přechodu z vodorovného do vertikálního letu. Pokud nejsou křídla rovně, po přitažení výškového kormidla model začne stoupat, ale neletí kolmo vzhůru, stoupá šikmo. Tento princip je na obr. 1.2 a obr. 1.3.

Ačkoliv to není zcela patrné, je udržení správné polohy křídel jednou z nejobtížnějších a nejdůležitějších záležitostí pilotáže. Pilot se dívá na model zdola a navíc se poloha modelu vůči pilotovi neustále mění. Z těchto důvodů je správné určení polohy pouhým pohledem velmi náročné.

Obr. 1.2 Vodorovná poloha křídel

Obr. 1.3 Měřitelný náklon letadla



1.1 Umělý horizont

Skutečná letadla mají na přístrojové desce takzvaný umělý horizont. To je přístroj, který zobrazuje skutečnou vodorovnou polohu horizontu bez ohledu na síly, působící na letadlo. Tento mechanický přístroj funguje na principu setrvačníku. Základem je pohyblivá otočná

koule, uvnitř je namontován gyroskop roztáčený elektrickým napětím. Tento mechanický systém doposud nebyl nahrazen sofistikovanějším zařízením. Pomyslný horizont je nakreslen linií rozdělující světlou (nejčastěji modrou) a tmavou polokouli. Tato barevná volba představuje oblohu a zemský povrch stejně jako při pohledu z kokpitu. Díky ocejchovaným stupnicím je možné odečítat číselnou hodnotu náklonu[1].

Pro aplikace do modelu letadla je tento sytém nepoužitelný. Hmotnost a rozměry zařízení jsou značné. Dalším problémem je přenést informaci pilotovi, který stojí na zemi. Případným řešením by mohlo být umístění kamery před umělý horizont a přenášet bezdrátově video obraz. Tento způsob by však nebyl ideální a je značně finančně náročný. Vhodnější je získat elektronickou informaci o náklonu modelu, která je mnohem méně náročná na kapacitu datového toku, než přenášený video signál. Také lze dosáhnout levnějšího způsobu řešení a nižší hmotnosti zařízení. Požadovaného výsledku lze dosáhnout pomocí akcelerometru, nebo také gyroskopu. Samozřejmě spolehlivost a přesnost takového přístroje je podstatně nižší, než u klasického umělého horizontu. Přístroj v klasickém podání na přístrojové desce skutečného letadla je na obr. 1.4.

Obr. 1.4 Umělý horizont na přístrojové desce letadla

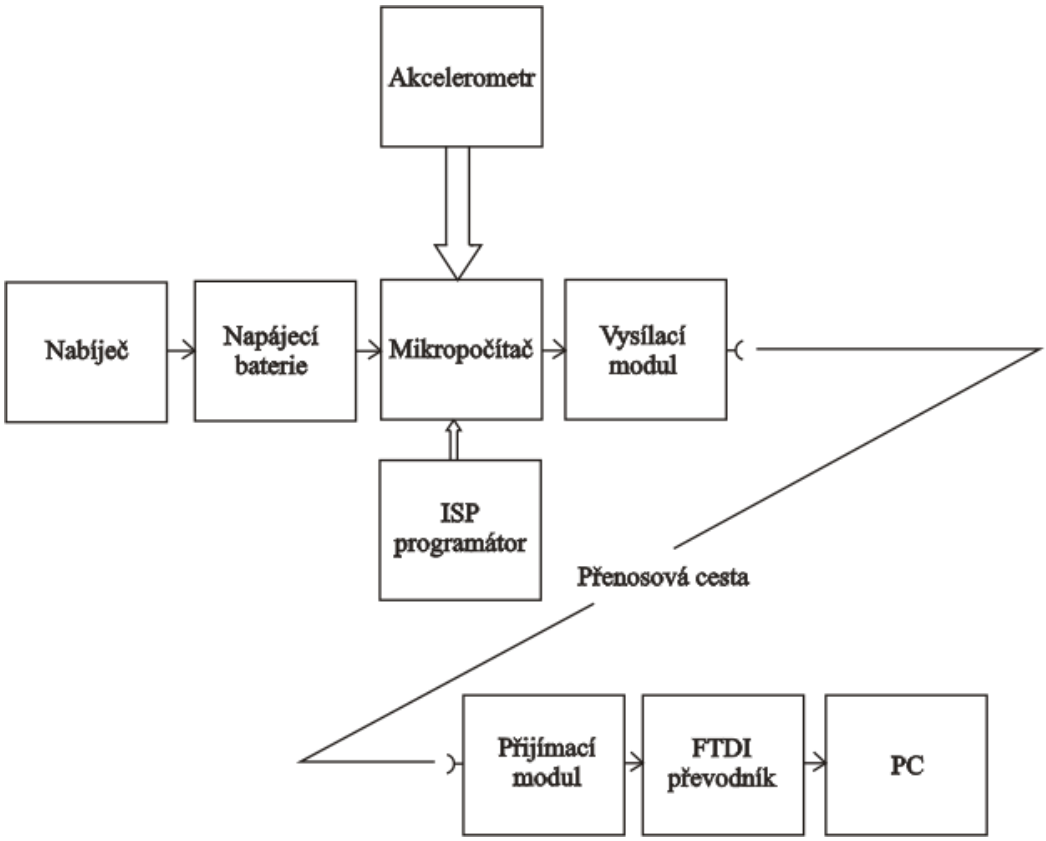
Výstřižek z Microsoft Flight Simulator 2004

2 Návrh měřícího zařízení

Hlavní požadavky na parametry zařízení pro určení polohy křídel modelu letadla jsou následující:

- Rozpoznat směr náklonu
- Určit velikost náklonu
- Grafické zobrazení na monitoru počítače
- Rozlišovat let v normální poloze a na zádech
- Bezdrátový přenos dat z modelu do přijímače připojeného k počítači
- Dostatečný počet měření za časový úsek
- Připojení k PC přes rozhraní USB
- Napájení obvodů v modelu z 2 Li-Pol článků
- Přijímací modul napájet z USB počítače
- Letadlo je ovládáno systémem na frekvenci 2,4 GHz. Mezi těmito přenosy nesmí docházet k rušení.

Obr. 2.1 Blokové schéma navrhovaného zařízení



Na palubě modelu letadla je zdroj pro napájení přijímače a ovládacích servomotorů. Napětí baterie ze dvou Li-Pol článků je stabilizováno na 5V. K tomuto zdroji je možné připojit i měřící zařízení, ale z bezpečnostních důvodů je použit samostatný zdroj ze dvou Li-Pol akumulátorů.

Dále je důležité správné uložení obvodů v letadle. Pokud by poloha tištěného spoje nebyla rovnoběžná s polohou křídel, bylo by možné kalibrovat rovnovážnou polohu, ale nastal by problém s rozeznáváním letu v normální poloze a letu na zádech.

Uložení přístroje musí být pevné, aby nedocházelo k chvění během letu. Na druhou stranu nesmí být upevnění příliš tvrdé, aby se vibrace od motoru a vrtule přenášely k akcelerometru co nejméně. Vhodným řešení mohou být silentbloky. Celý upevňovací systém je potřeba udělat tak, aby měl co nejnižší hmotnost.

Při umisťování vysílací antény je potřeba zohlednit, že motor odebírá proud kolem 80A. Regulátor otáček je zároveň střídačem, takže motor je napájen střídavým napětím. Zde vzniká určité riziko rušení. Z těchto důvodů je vhodné umístit vysílací modul s anténou co nejdále od obvodů zajišťujících pohon letadla.

3 Hardware

3.1 Akcelerometr

Akcelerometr je součástka, která měří velikost zrychlení v určitém směru a převádí hodnotu na velikost elektrického napětí. Tyto součástky bývají použité v široké řadě aplikací, například:

- Měření vibrací a otřesů
- Měření zrychlení
- Měření odstředivé síly
- Měření setrvačných sil (airbag v automobilu)
- Měření náklonu
- Detekce pohybu
- Detekce a měření otřesů při pádu při přepravě
- Měření seizmické aktivity

Akcelerometr se skládá z:

1. Základny- Pevně spojena s objektem.
2. Setrvačné hmoty- Pružně uložena a její výchylka vůči základně je vyhodnocována.
3. Tlumení- Pokud je tlumení příliš malé, vzniká problém s překmity. Nebo naopak při velkém tlumení, dochází ke snížení citlivosti a zvýšení fázového posuvu.

3.1.1 MEMS akcelerometr

Mikro – elektro - mechanické systémy integrují mechanické senzory a elektroniku na jednom čipu křemíkového substrátu. Výsledkem výroby je elektromechanická součástka. Elektronika je vyráběna standardním postupem výroby integrovaných obvodů. Zatímco mechanické části jsou vyráběny obráběcími procesy leptání a přidávání vrstev křemíku.

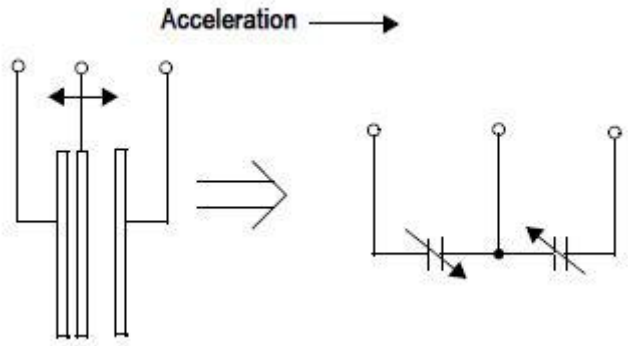
Akcelerometry jsou součástky, které umožňují měřit zrychlení. Kromě dynamického zrychlení měří i statické, pokud je obvod v klidovém stavu. To znamená, že dokáže měřit tíhové zrychlení. Při rovnovážné klidové poloze naměří zrychlení 1g v ose Z. A dynamické zrychlení je způsobeno změnou rychlosti pohybu. Při správném vyhodnocování měřených údajů lze zjistit o kolik stupňů a v jakém směru se vychýlí rovina akcelerometru oproti rovině kolmé na vektor gravitačního zrychlení [4].

Princip funkce MEMS akcelerometrů je založen na proměnné kapacitě tříelektrodového kondenzátoru. Princip funkce vychází z nelineární závislosti kapacity na vzdálenosti elektrod kondenzátoru. Rovnice (3.1) zobrazuje výpočet kapacity deskového kondenzátoru.

$$C = \epsilon_0 \epsilon_r \frac{S}{d} [F] \tag{3.1}$$

Na obr. 3.1 je znázorněn základní princip. Prostřední elektroda se pohybuje mezi dvěma pevnými elektrodami a její pohyb je závislý na působení sil zrychlení. Takto vznikne kapacitní akcelerometr.

Obr. 3.1 Princip MEMS akcelerometru



http://www.pololu.com/file/download/MMA7260QT.pdf?file_id=0J87

Jedním z hlavních požadavků je tzv. měřící rozsah senzoru. Tím je myšleno minimální a maximální měřitelné zrychlení.

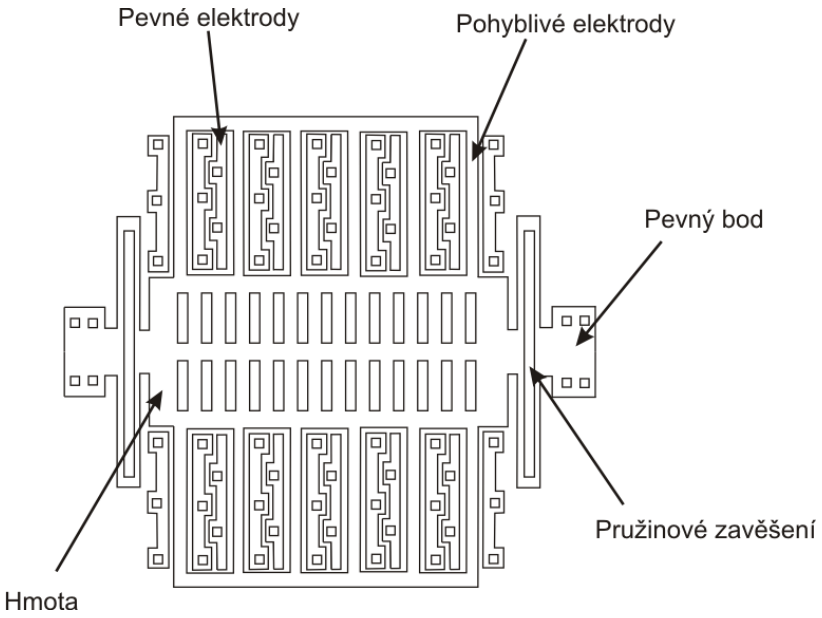
Mechanická struktura MEMS akcelerometru je zobrazena na obr. 3.2. Princip je založen na rovnici pro působení síly při zrychlení (3.2).

$$F = m \,.\, a \; [N] \tag{3.2}$$

Vzniká síla působením zrychlení na pevný bod. Tato síla se pak přes pružinové uložení přenáší na hmotu. Některé části tvoří pohyblivé elektrody vzduchového kondenzátoru, jejichž pozice se vlivem působení sil mění oproti pozici pevných elektrod. Takto je určena velikost kapacity vzniklého vzduchového kondenzátoru.

Struktura nosníků a pružin je leptána do polykrystalického křemíku. Zlepšující se technologie umožňují protahování pohyblivých elektrod z původních jednotek mikrometrů až na desítky mikrometrů. To umožňuje získat podstatně lepší odstup signál-šum, menší vliv zrychlení v jedné ose na osy jiných směrů a hlavně lepší odezvu na změnu velikosti zrychlení.

Obr. 3.2 Mechanická struktura MEMS akcelerometru



Popsaná struktura umožňuje měřit zrychlení v jednom směru. Přidáním druhé vrstvy, pootočené o 90°, vzniká akcelerometr pro měření zrychlení ve dvou osách. A po přidání výškově pohyblivé struktury vznikne 3D akcelerometr.

Měřená změna kapacity je vnitřní elektronikou zpracována a převedena na změnu velikosti elektrického napětí. To je filtrováno a linearizováno dalšími obvody a nakonec se provádí kompenzace vlivu teploty. Výstupem je zesílený, lineární a kompenzovaný napěťový signál. Citlivostí je definována informace o kolik se musí změnit hodnota měřeného zrychlení, aby došlo ke změně výstupního napětí o 1V. Každá osa funguje samostatně [2].

3.1.2 Akcelerometr MMA7260QT

Je schopen měřit zrychlení ve všech třech osách (X, Y, Z). Výstupní signály ze všech tří os jsou analogové, takže je nutné u mikropočítače použít A/D převodník. Na výstupu je signál upravován dolní propustí, teplotní kompenzací a funkcí g-Select, která umožňuje přepínat mezi čtyřmi rozsahy citlivostí.

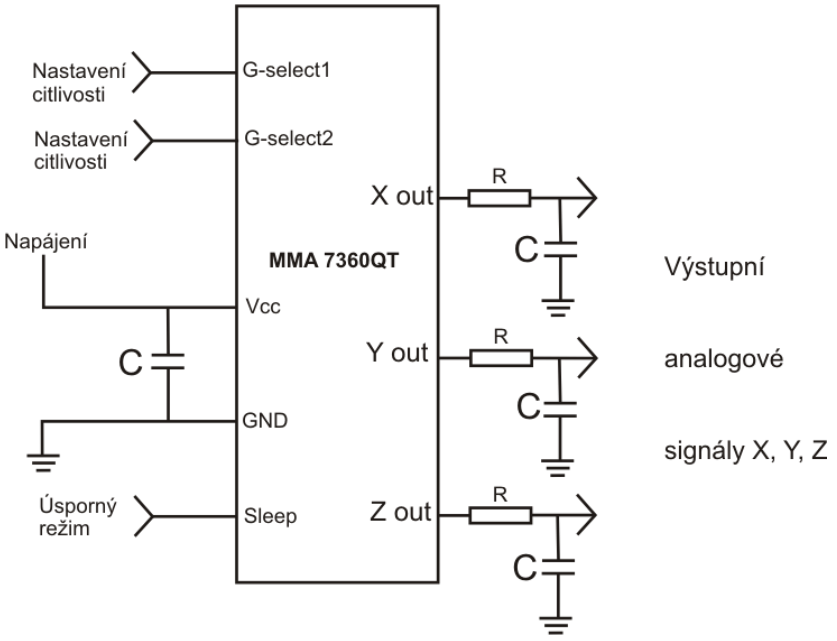
Vlastnosti:

- Volitelná citlivost (1,5g/2g/4g/6g)
- Nízká proudová spotřeba: 500µA
- Spotřeba v uspaném režimu: 3µA
- Napájecí napětí: 2.2V – 3.6V
- Reakční doba: 0,5ms

- Teplotní rozsah: -40°C až +105°C
- Maximální zrychlení, které nepoškodí akcelerometr: 5000g
- Křížová citlivost: 5%
- Nelinearita: 1%
- Test vlastní funkce
- Logický výstup detekce pádu (0g-Detect)

Výstupním napěťovým analogovým signálům lze nastavit rozsah a s tím související citlivost pomocí pinů g-select (obr. 3.3).

Obr. 3.3 Akcelerometr MMA 7360QT



Tabulkou (3.1) je znázorněno nastavení citlivosti a rozsahu měření v závislosti na stavu pinů g-select [3].

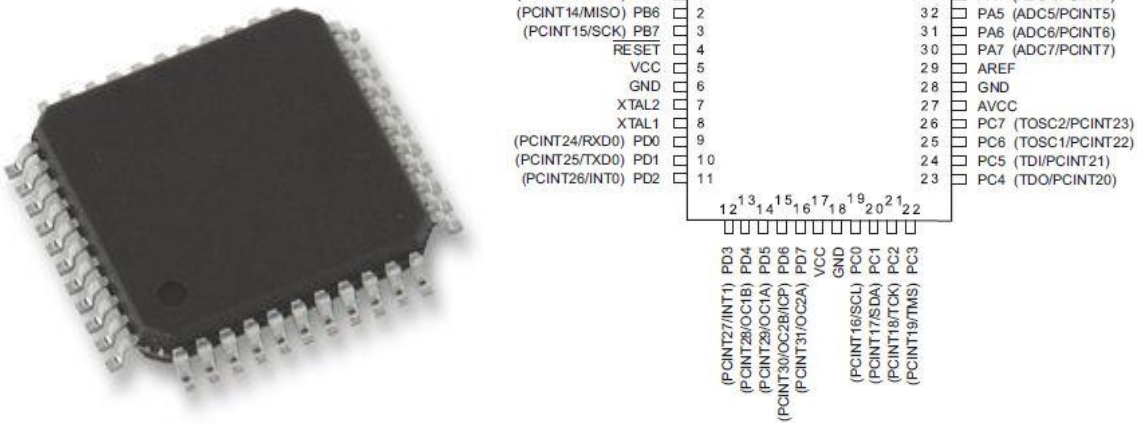
Tabulka 3.1 Nastavení rozsahu a citlivosti

g-select2	g-select1	Rozsah	Citlivost
0	0	1.5g	800mV
0	1	2g	600mV
1	0	4g	300mV
1	1	6g	200mV

3.2 Procesor ATmega 644

Na desku plošného spoje měřicího přístroje byl vybrán jako procesor produkt firmy Atmel ATmega644. Jedná se o 8bitový mikropočítač v pinové konfiguraci TQFP.

Obr. 3.4 ATmega 644, TQFP



http://www.atmel.com/Images/doc2593.pdf
http://cz.farnell.com/atmel/atmega644v-10au/8bit-mcu-64k-flash-1-8v-5-5v-644/dp/1288341

Mikroprocesory AVR jsou stavěny na Harwardské architektuře. Tím je myšleno použití samostatných pamětí pro data a program. Parametry mikropočítače jsou pro danou aplikaci naprosto vyhovující. Je stavěn podle tzv. *RISC* modelu, který je odlišný od běžně používaných procesorů *Intel* nebo *AMD.* RISC je označení pro procesory s redukovanou instrukční sadou. Návrh je zaměřen na jednoduchou optimalizaci strojových funkcí [5].

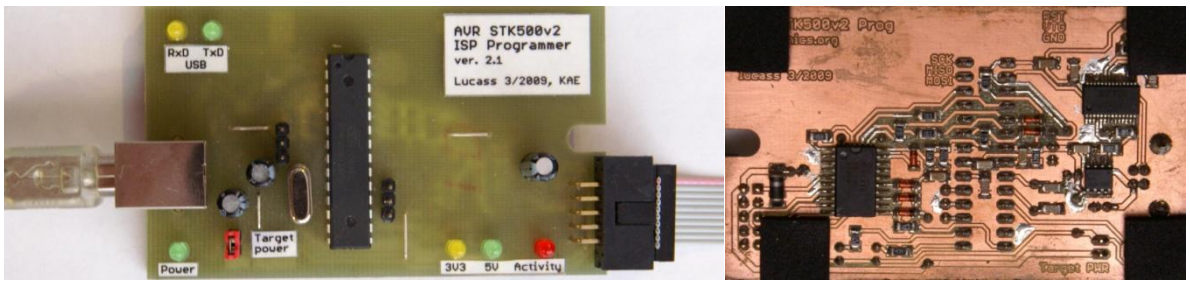
Vlastnosti

- 64 kB vlastní programovatelná programová paměť
- 2 kB EEPROM paměť
- USART - komunikace přes sériový port
- 6 x PWM pro pulsně šířkovou modulaci (řízení motorů apod.)
- 8 kanálů 10bit A/D převodníku
- Watchdog timer - automatický reset v případě zamrznutí programu
- Interní RC oscilátor, pokud není použit externí krystal [7]

3.2.1 Připojení ISP programátoru

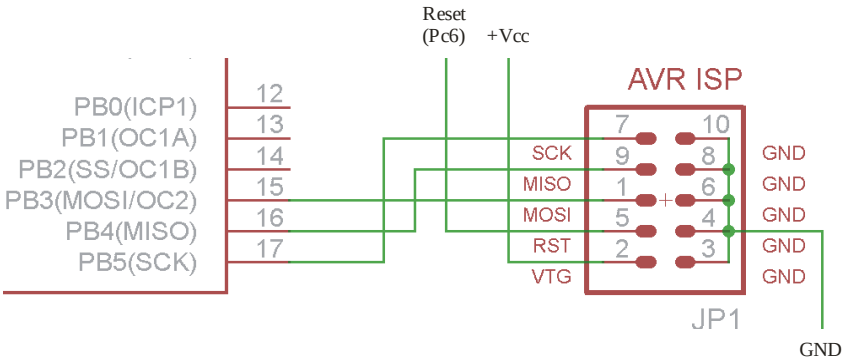
Programování lze zvolit paralelní, nebo sériové. Nevýhodou paralelního programování je nutnost odpojit procesor od veškerých obvodů a umístit jej do programátoru. To je v řadě aplikací dosti nepraktické, zvláště pokud je potřeba přehrát software v čipu. Tato nevýhoda odpadá při volbě sériového programování. Mikroprocesor zůstává zapojen v obvodu a lze ho jednoduše naprogramovat pomocí signálů SCK, MISO, MOSI a RESET. Tato vlastnost je nazvána ISP programování. Při práci s mikropočítačem bylo využito ISP programátoru AVR STK500 (obr. 3.5) [6].

Obr. 3.5 Programátor AVR STK500v2



Ten je připojen k obvodu s mikropočítačem přes ISP konektor a spojení s PC je realizováno pomocí USB. Na obr. 3.6 je zobrazeno zapojení mikroprocesoru a konektoru programátoru.

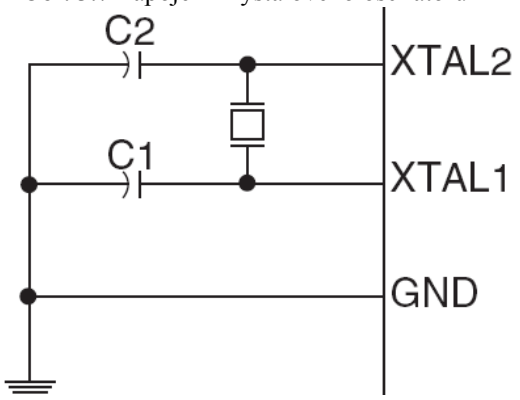
Obr 3.6 Připojení ISP konektoru



3.2.2 Oscilátor procesoru

Přestože mikropočítač umožňuje využití interního oscilátoru, je vhodné použít externí. Důvodem je potřeba přesnosti, aby nedocházelo ke zbytečným chybám, které by mohla způsobovat nepřesná frekvence oscilátoru. Nicméně, interní oscilátor by pravděpodobně postačil. Zapojení oscilátoru je zobrazeno na obr. 3.7.

Obr. 3.7 Zapojení krystalového oscilátoru

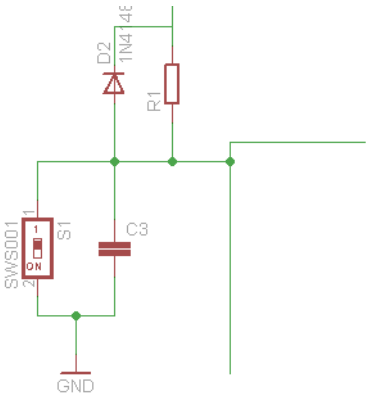


http://www.gme.cz/_dokumentace/dokumenty/958/958-102/dsh.958-102.1.pdf

K frekvenci 8MHz externího krystalu jsou podle datasheetu doporučovány kondenzátory o kapacitě 12 - 22pF. Do zapojení byly použity 22pF. Krystal je připojen na piny XTAL1 a XTAL2, dále je zde nutné spojení s GND. Hodnota krystalu 8MHz je vhodná pro následné nastavení rychlosti sériové komunikace. Spotřeba procesoru je závislá na frekvenci oscilátoru, takže výběr nižších frekvencí vede k menšímu proudovému odběru. Tato vlastnost je vzhledem k připojení na akumulátor palubní elektroniky letadla velmi žádoucí.

3.2.3 Resetovací obvod procesoru

Obr. 3.8 Resetovací obvod procesoru



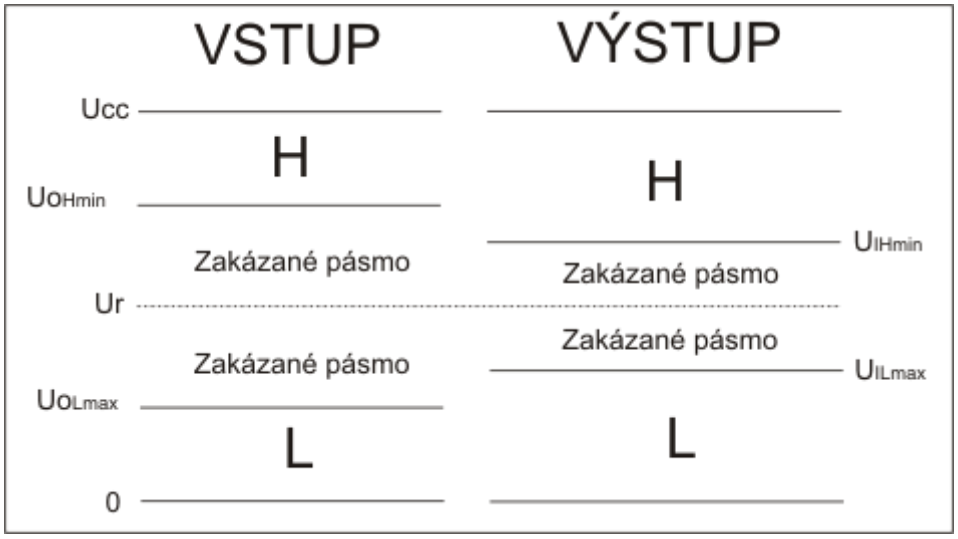
Zapnutí mikropočítače bez vykonání platné funkce RESET způsobí, že procesor rozběhne program z náhodné polohy v paměti programu, protože obsah čítače nebyl vynulován a jeho obsah je tedy náhodný. Na obr. 3.8 je návrh resetovacího obvodu, který je připojen na pin RESET. Krátce po zapnutí napájení dochází v obvodu k přechodovému ději. V tomto okamžiku se procesor ocitá mimo definované parametry a to může vést až k nespuštění programu, v důsledku neočekávaných stavů. RC článek způsobí krátké zpoždění startu procesoru. Pokud dojde k ojedinělé kolizi je vhodným řešením stisk resetovacího tlačítka. Za předpokladu, že

nedošlo k jiné chybě, nebo poškození bude obvod pracovat opět korektně. Hodnoty součástek byly zvoleny R_1=10kΩ a C_3=100nF.

3.3 Převodník sériového rozhraní v TTL úrovních na USB

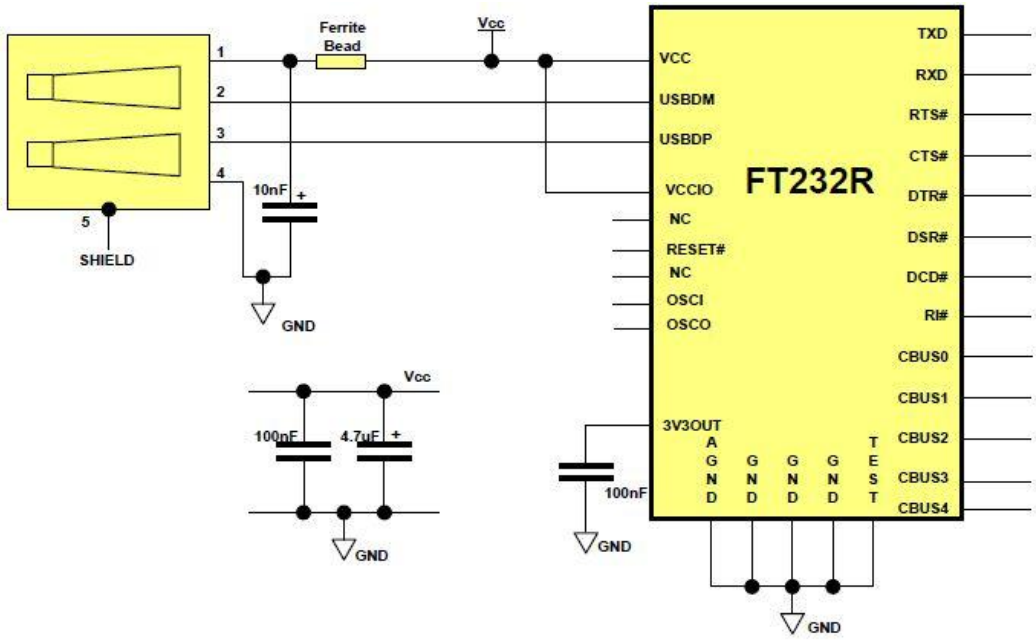
Tranzistorově-tranzistorová logika je standardem v integrovaných logických obvodech. Technologie vychází z použití bipolárních křemíkových tranzistorů. Na obr. 3.9 je zobrazen základní princip funkce TTL logických obvodů. Rozeznávají logické stavy 0 a 1. Mezi těmito úrovněmi je tzv. zakázané pásmo. V napěťových úrovních zakázaného pásma se signál nesmí nacházet a při přechodu z jednoho logického stavu do druhého musí být toto pásmo co nejrychleji překonáno. Z obr. 3.9 je dále patrné, že logický obvod musí mít zakázané pásmo na vstupu širší, než na výstupu. To je požadavek pro kompatibilitu s dalšími připojenými obvody. Jinak jasně definovaný výstup z jednoho logického členu může vstoupit do druhého v úrovních zakázaného pásma. Uprostřed zakázaného pásma je tzv. rozhodovací úroveň U_r, kdy dochází k překlopení logických stavů 0 a 1.

Obr. 3.9 Úrovně signálů TTL



Důvodem využití USB bylo, že ne všechna zařízení jsou vybavena sériovým rozhraním RS-232C. Z hlediska technické realizace to neznamená žádné komplikace. Při využití rozhraní RS-232C je často využíván převodník MAX232 pro převod z TTL logických úrovní. Pokud RS-232C chceme nahradit USB, je možné nahradit MAX232 převodníkem FTDI. Konkrétně byl použit v zapojení obvod FT232RL. Výhodou převodníku je, že nevyžaduje instalaci žádného specifického firmware. Spojení s počítačem je navázáno automaticky po připojení. Bylo využito konfigurace, která umožňuje napájení převodníku z PC přes USB (obr. 3.10) dle datového listu součástky [10].

Obr. 3.10 FTDI převodník



http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf

3.4 Vysílací a přijímací modul

Pro danou aplikaci byly zvoleny vysílací a přijímací moduly XTR-434 od firmy Aurel. Modul je stejný pro funkci vysílače i přijímače. Nastavení vysílacího, nebo přijímacího režimu se realizuje pinovou konfigurací podle datového listu (tab. 3.2).

Tab. 3.2 Nastavení XTR-434

PIN 15 TX enable	PIN 16 RX enable	Funkce
1	1	Modul není funkční
1	0	Přijímač
0	1	Vysílač
0	0	Zakázaná konfigurace

Jedná se moduly využívající FM modulaci na frekvenci 433,92 MHz s maximální přenosovou

rychlostí 100 Kbps. Zařízení disponuje malými rozměry (33mm x 23mm x 8mm) a díky nízké hmotnosti je vhodné pro použití v modelu letadla. V přijímacím režimu má modul odběr 11 mA, takže je možné na přijímací straně napájet zařízení přímo z USB připojeného počítače. Takže přijímací část měřícího zařízení nepotřebuje externí zdroj napájení. Ve vysílacím režimu je odběr modulu poněkud vyšší (28 mA), ale tato hodnota je pro napájení z Li-Pol baterie stále velmi příznivá. Moduly odpovídají evropským normám EN 300 220 a EN 301 489 [8].

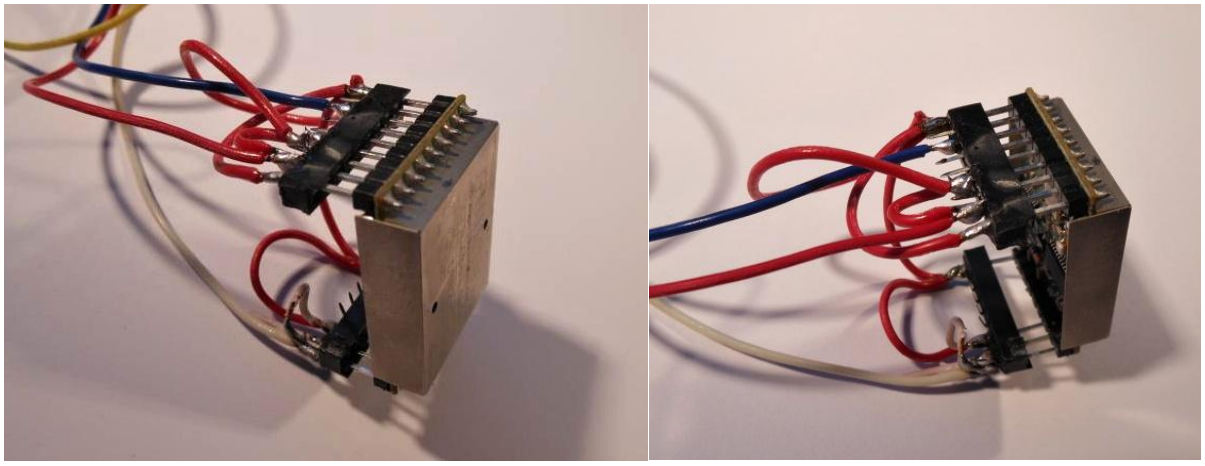
Nebyla použita žádná speciální anténa. V takovém případě postačí drát o délce λ/4. Výpočet vlnové délky je znázorněn v rovnici (3.3). Délku antény pak lze vypočítat dle vzorce (3.4).

$$\lambda = \frac{v}{f} = \frac{3*10^8}{433{,}92*10^6} = 0.69m \tag{3.3}$$

$$l = \lambda/4 = 17.25\text{cm} \tag{3.4}$$

Na vysílací a přijímací modul jsou kladeny celkem vysoké nároky. Dosah v daném případě je potřeba aspoň 400 metrů. Při využití zařízení v modelu poháněným elektrickým motorem, je situace komplikována z hlediska odrušení. Letadlo je zdrojem silného elektromagnetického rušení. Elektromotor o výkonu 2 kW je napájen baterií 10 Li-Pol akumulátorů o kapacitě 5 Ah. Z baterie je odebírán proud až kolem 80 A. Regulátor otáček zároveň plní funkci střídače, protože motor je 3 fázový. Z těchto důvodů je potřeba umístit vysílací modul v trupu letadla co nejdále od pohonné jednotky.

Obr. 3.11 Moduly Aurel XTR-434



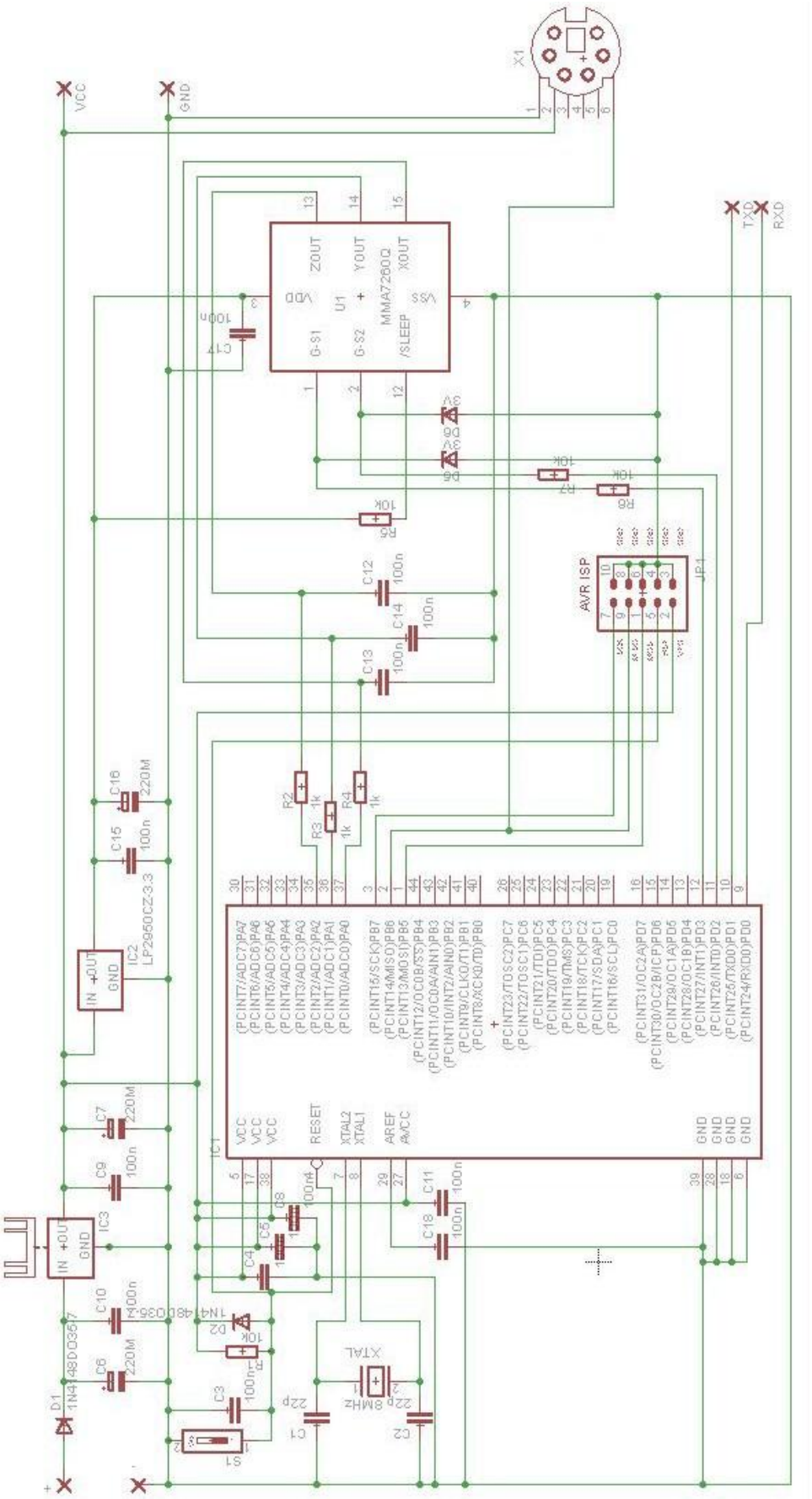
4 Schéma zapojení a DPS

Obvod musí být rozdělen na dvě samostatné desky plošných spojů. První část je obsažena v modelu letadla a druhý obvod s funkcí přijímače dat je připojen k PC. Na obvod v modelu letadla jsou kladeny nároky, aby byl napájen z baterie a dosahoval co nejnižší hmotnosti. Dále je nutné zařízení v trupu správně připevnit.

4.1 Obvod v modelu letadla

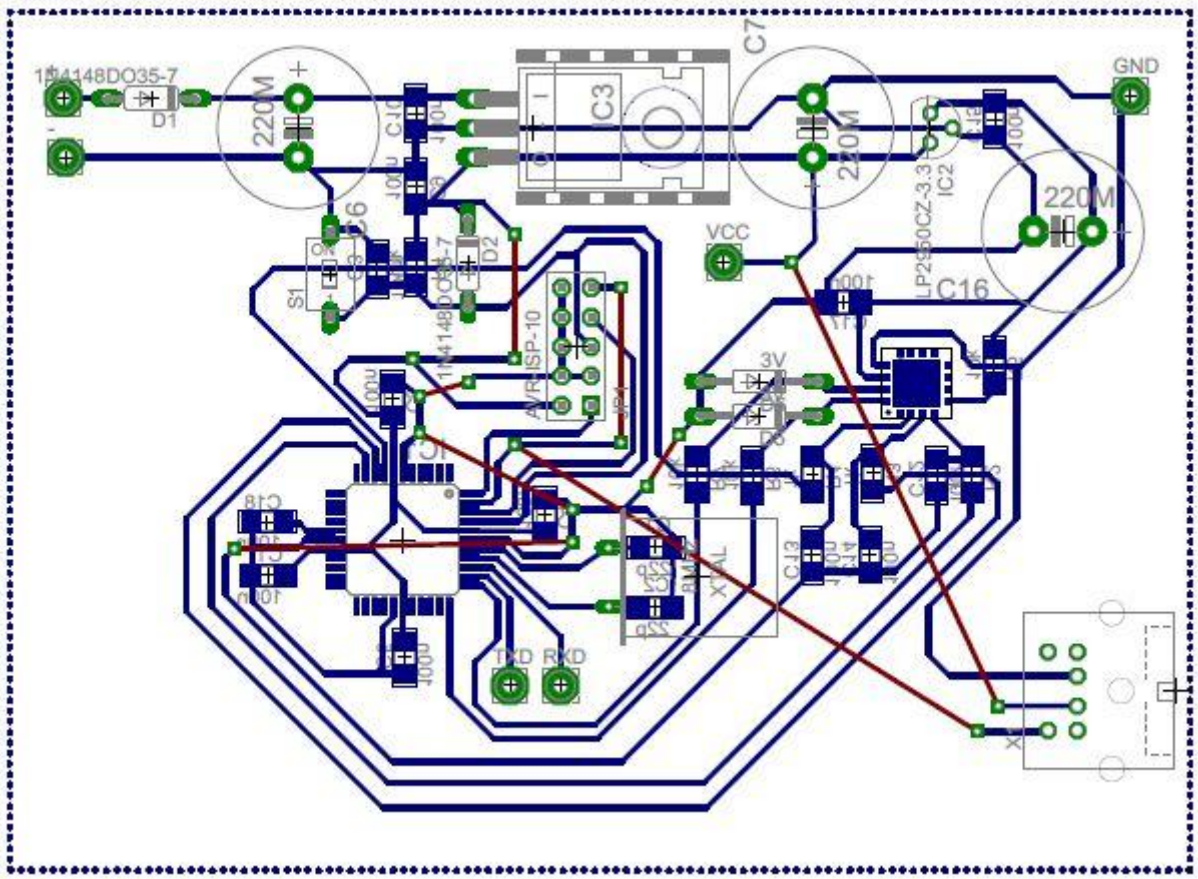
Na palubě letadla slouží jako zdroj baterie ze dvou Li-Pol akumulátorů. Napětí sady je závislé na stavu nabití, a tak jsou baterie připojeny na stabilizátor napětí. Zařízení, včetně vysílacího modulu pracuje na 5V. Z toho plyne výběr integrovaného stabilizátoru 7805. Dále jsou zapojeny obvody, které byly popsány v kapitole 3 (Hardware). Schéma zapojení a následný návrh DPS byl realizován v programu EAGLE, což je produkt firmy Cadsoft se sídlem v Německu. Program je možné používat ve verzi freeware při návrhu desky o rozměrech maximálně 100x80mm, mohou být použity jen dvě signálové vrstvy spojů (spodní a vrchní strana) a schéma může být vytvořeno pouze na jednom listu [9]. Freeware verze programu byla pro daný návrh naprosto dostačující. Schéma zapojení obvodu umístěného ve vrtulníku je na obr. 4.1. Návrh DPS následuje na obr. 4.2.

Obr. 4.1 Schéma zapojení obvodu v letadle



Deska plošných spojů má rozměr 100 mm x 70 mm. Návrh byl realizován jako jednostranný, s tím, že bylo potřeba osmi drátových propojek, které jsou zobrazeny na obr. 4.2 červenými spoji. V případě použití dvouvrstvého plošného spoje by nebylo drátových propojek potřeba.

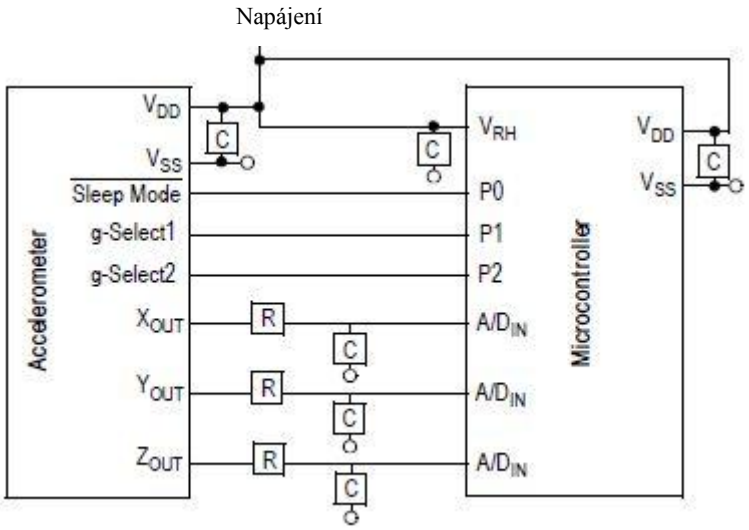
Obr. 4.2 DPS v modelu letadla



Výstupem obvodu jsou komunikační porty RXD a TXD. TXD (transmitter) kanál je vysílán a dále zpracováván FTDI převodníkem po přijetí signálu přijímačem. Dalším výstupem je stabilizované napětí 5V, určené jako napájení pro vysílací obvod. Nicméně použitý akcelerometr MMA7260QT je určen pro napájení 3,3 V. Z tohoto důvodu je za stabilizátorem 7805 zařazen další stabilizátor LP2950, který stabilizuje napájecí napětí pro akcelerometr na 3,3V. Existuje možnost použít procesor pracující na 3,3 V. Ale 5 V je v obvodu potřeba pro vysílací modul nevyhnutelně. Z tohoto důvodu byl zvolen procesor na 5 V. V této kombinaci je již pouze potřeba snížit napěťovou úroveň signálů vstupujících z procesoru do akcelerometru. To je realizováno stabilizačními Zenerovými diodami BZX83V003.3 na vstupech akcelerometru a sériově řazenými odpory. Jedná se pouze o dva piny, kterými se nastavuje citlivost akcelerometru (G-select). Výstupy akcelerometru jsou připojeny, dle

doporučeného zapojení v datovém listu (obr. 4.3), přes RC filtry, za účelem snížení úrovně šumu. Dalším předpokladem správné funkčnosti je co nejkratší vzdálenost mezi součástkami.

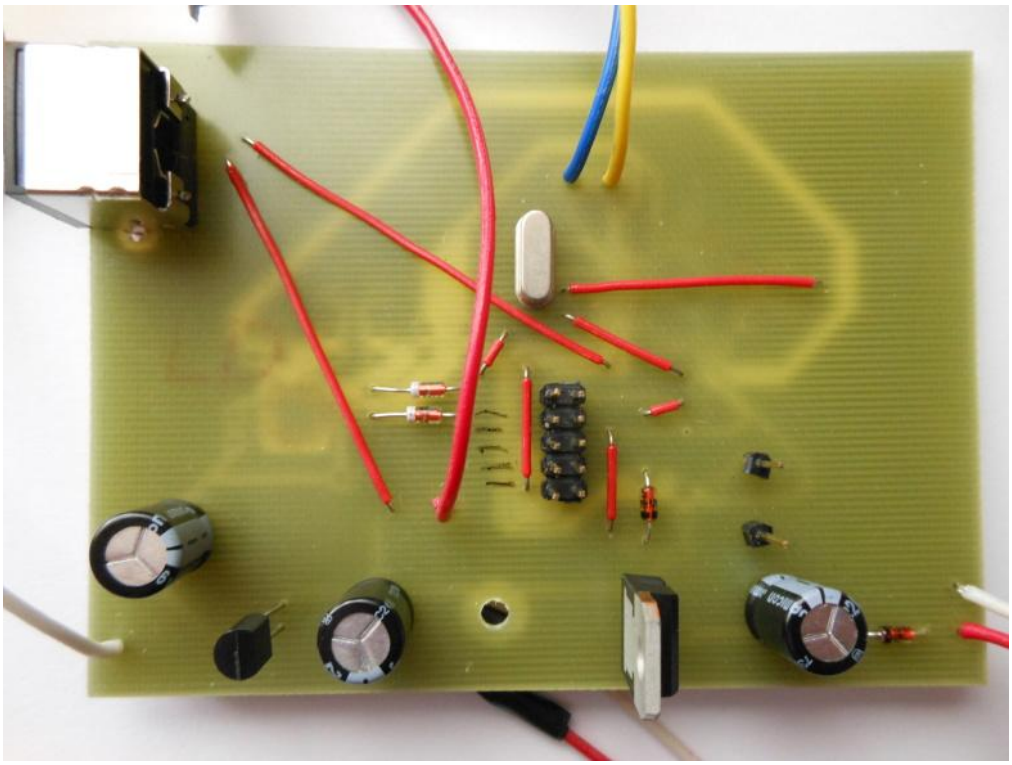
Obr. 4.3 Doporučené připojení akcelerometru k procesoru



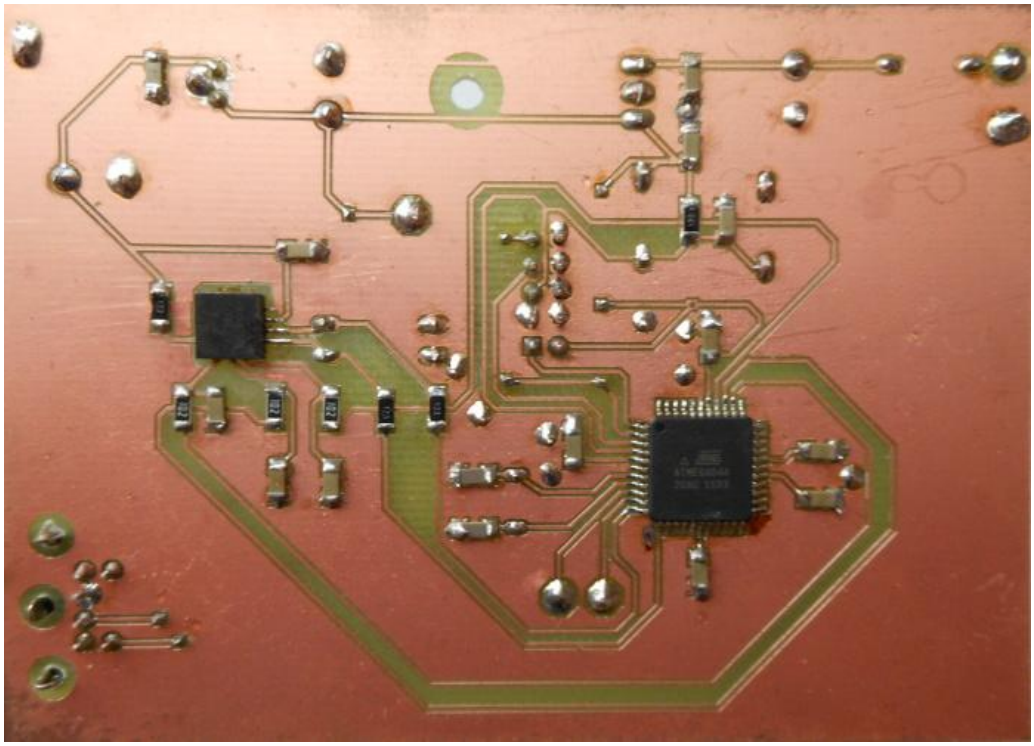
http://www.pololu.com/file/download/MMA7260QT.pdf?file_id=0J87

V desce plošných spojů jsou technologické otvory pro snadnou montáž do trupu modelu. Na obr. 4. 4. je pohled shora na realizovanou DPS a na obr. 4. 5 pohled zdola.

Obr. 4.4 Realizovaná DPS shora



Obr. 4.5 Realizovaná DPS zdola



Tabulka 4.1 Seznam součástek – Obvod v modelu letadla

Součástka	**Počet kusů**	**Označení**	**Typ**
Kondenzátor	2	C1, C2	22p
Kondenzátor	13	C3 - C5, C8 – C15, C17, C18	100n
Elektrolytický kondenzátor	3	C6, C7, C16	220M
Dioda	2	D1, D2	1N4148
Zenerova dioda	2	D5, D6	BZX83V003.3
Procesor	1	IC1	ATmega644
Stabilizátor napětí	1	IC2	LP2950CZ-3.3
Stabilizátor napětí	1	IC3	7805

ISP konektor	1	JP1	AVR-ISP-10
Rezistor	3	R2, R3, R4	1k
Rezistor	4	R1, R5, R6, R7	10k
Zkratovací spínač	1	S1	-
Akcelerometr	1	U1	MMA7260QT
Krystal	1	XTAL	8MHz
Konektor	1	X1	MDD6BB

4.1.1 Instalace zařízení do trupu letadla

Celé zařízení je namontováno v kabině modelu. Tím je dosaženo největší možné vzdálenosti od pohonné sady a celý systém je snadno dostupný obsluze. Přichycení musí být pevné, aby nedocházelo k pružení, a zároveň nesmí mít velkou hmotnost. Dalším požadavkem bylo provádět co nejméně technických zásahů do trupu letadla. Deska plošného spoje s akcelerometrem je přichycena dvěma šrouby M3 přes plastové silentbloky, vysílací modul je přilepen na laminát kabiny a baterii drží suchý zip (obr. 4.6).

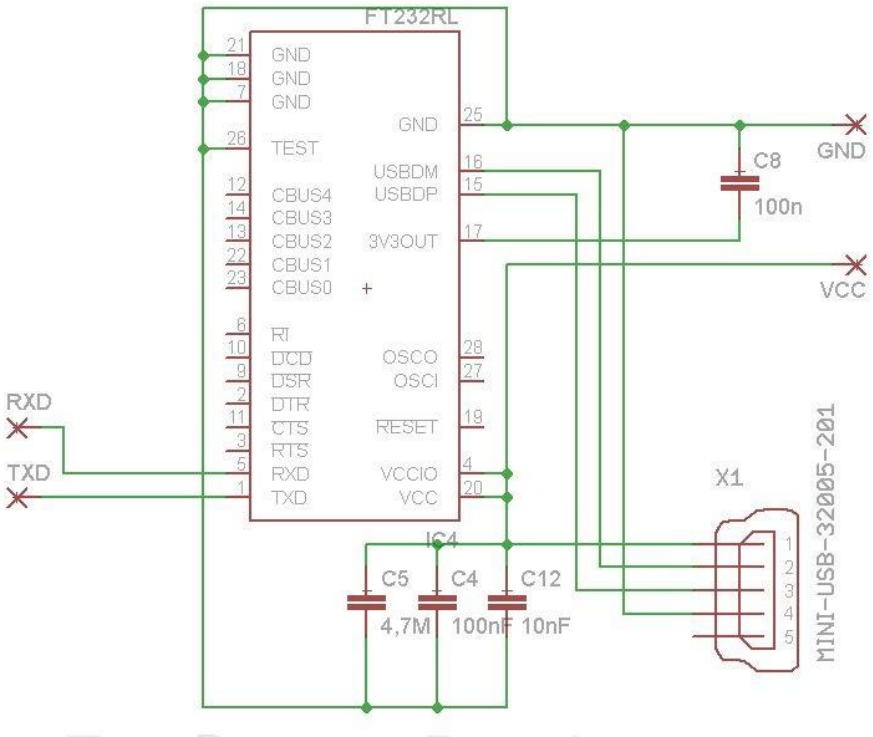
Obr. 4.6 Instalace zařízení do trupu modelu

Zapnutí se provádí zapojením konektoru mezi baterií a DPS.

4.2 Obvod připojený k počítači

Zde se nabízí možnost napájet FTDI převodník z PC přes USB konektor. Vzhledem k nízkému příkonu přijímacího modulu je možné ho napájet z USB konektoru také. Díky tomu není potřeba žádného externího zdroje napětí. Periferní součástky převodníku byly vybrány a zapojeny dle datového listu [10]. Tentokrát je z komunikačních portů využit RXD (receiver). Pro připojení desky k PC byl vybrán USB konektor MINI USB. Dalším výstupem je stabilizované napětí 5V, určené jako napájení pro přijímací modul. Na obrázku 4.7 je zobrazen návrh DPS, který byl nakreslen stejným postupem, jako první obvod v programu EAGLE. Rozměry desky jsou 30 mm x 30 mm.

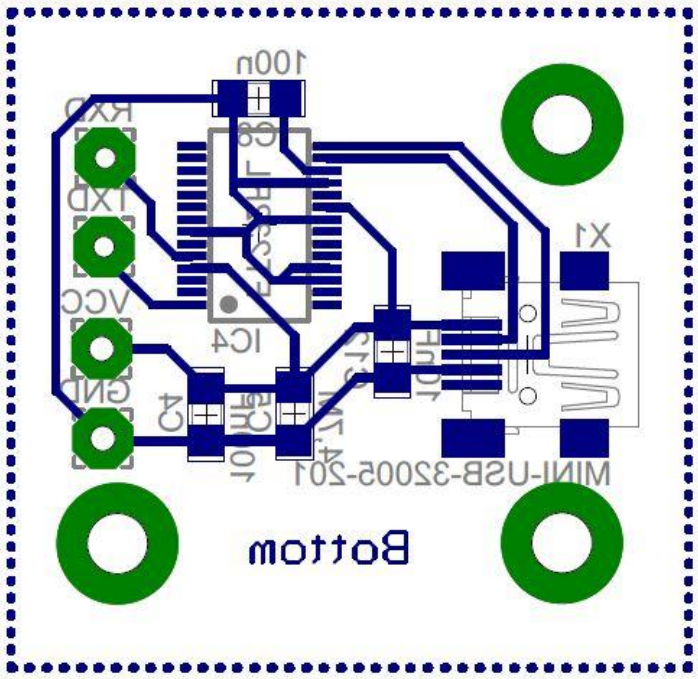
Obr. 4.7 Schéma zapojení obvodu připojeného k PC



Obvod byl opět realizován jako jednovrstvý. Nebylo zde potřeba druhé vrstvy, ani drátových propojek (obr. 4.8).

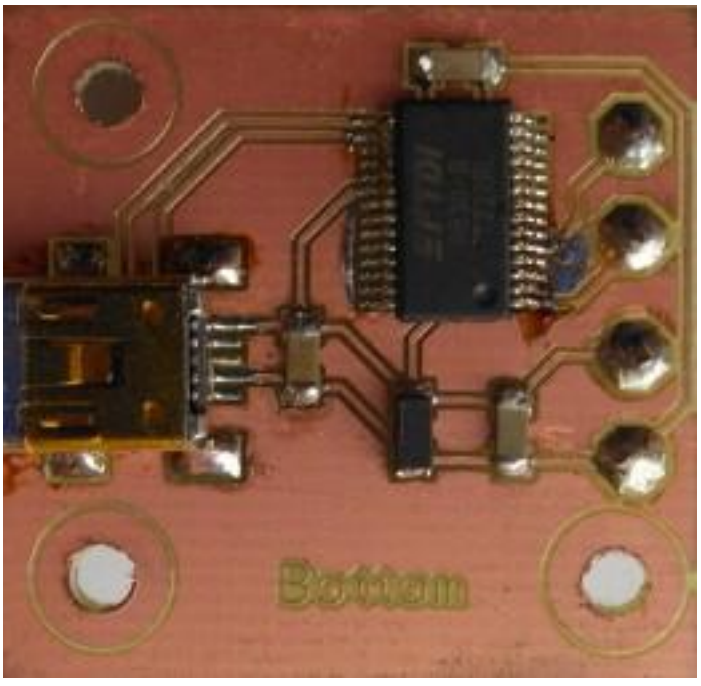
Schéma zapojení a DPS

Obr. 4.8 DPS obvodu připojeného k PC



Na obr. 4.9 je realizovaný a funkční obvod.

Obr. 4.9 Vyrobená DPS připojená k PC



Tabulka 4.2 Seznam součástek – Obvod připojený k PC

Součástka	**Počet kusů**	**Označení**	**Typ**
Kondenzátor	2	C4, C8	100nF
Kondenzátor	1	C5	4,7MF
Kondenzátor	1	C12	10nF
FTDI převodník	1	IC4	FT232RL
USB konektor	1	X1	Mini-USB

Tím je dokončena hardwarová část řešení daného problému. Vyrobené desky plošných spojů jsou funkční. Dále je potřeba naprogramovat procesor ATmega 644 a připravit program pro zpracování a zobrazení výsledků měření na PC.

5 Software

Informace o poloze je složena ze tří hodnot, protože dochází k měření ve třech osách. Z těchto tří vzorků gravitačního zrychlení je možně spočítat úhel náklonu.

Nejprve je potřeba spočítat vektor gravitační síly (rovnice 5.1).

$$g_m = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{5.1}$$

Vzhledem k tomu, že v následujících výpočtech se nepočítá se skutečnou velikostí zrychlení, ale s poměry mezi zrychlením v dané ose a vektorem gravitační síly, není potřeba údaje z akcelerometru na skutečnou hodnotu zrychlení přepočítávat. Za pomoci goniometrických funkcí je možné počítat náklon ve dvou směrech (rovnice 5.2 a 5.3).

$$Náklon\ 1 = arcsin\frac{a_x}{g_m} * \frac{180}{\pi}\ [°] \tag{5.2}$$

$$Náklon\ 2 = arcsin\frac{a_y}{g_m} * \frac{180}{\pi}\ [°] \tag{5.3}$$

Vynásobením vzorce $\frac{180}{\pi}$ je výsledek převeden z úhlových radiánů na stupně [4].

5.1 Programování procesoru ATmega 644

Výstupem akcelerometru je analogové napětí. Velikost této hodnoty musí být zpracována na vstupu procesoru A/D převodníkem. Každý výstup akcelerometru je připojen na jeden kanál A/D převodníku.

Úkolem procesoru je zpracovat data, která přicházejí ze všech tří kanálů akcelerometru a posílat je po UART sériové komunikaci přes FTDI převodník do PC. Při frekvenci 8 MHz krystalového oscilátoru byla zvolena přenosová rychlost 4800 Bd.

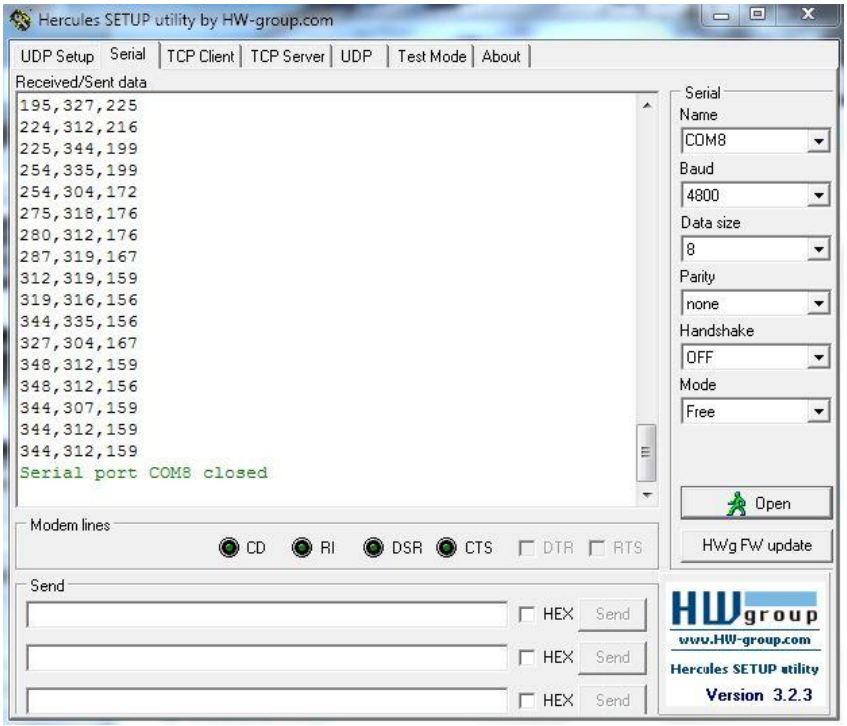
Je potřeba definovat tvar odesílané informace pro jednu polohu tak, aby přijímací strana byla schopna data správně identifikovat a zpracovat. Byla zvolena formulace pro jeden údaj polohy taková, že všechna tři čísla jsou na jednom řádku a odděluje je čárka. Pro jednoduchou kontrolu odesílaných informací z procesoru do PC byl použit program Hercules (obr. 5.1).

5.1.1 Aplikace Hercules

Jako software pro zobrazování dat byla zvolena aplikace Hercules. Jedná se o terminál pro obsluhu sériových portů. Práce s programem je velmi jednoduchá. Po zvolení záložky *Serial*

je potřeba vybrat příslušný COM port, na který je zařízení připojeno. Následuje nastavení přenosové rychlosti v baudech, počtu datových bitů na znak, paritního bitu a kontroly přenosu způsobujícího přerušení přenosu dat v okamžiku, kdy je *buffer* zařízení naplněný. Pokud vše pracuje správně, po stisknutí volby *open* se začnou vypisovat údaje o poloze (obr. 5.1). Lze napsat, že formát odesílaného řádku je: a_x, a_y ,a_z.

Obr. 5.1 Hercules terminal

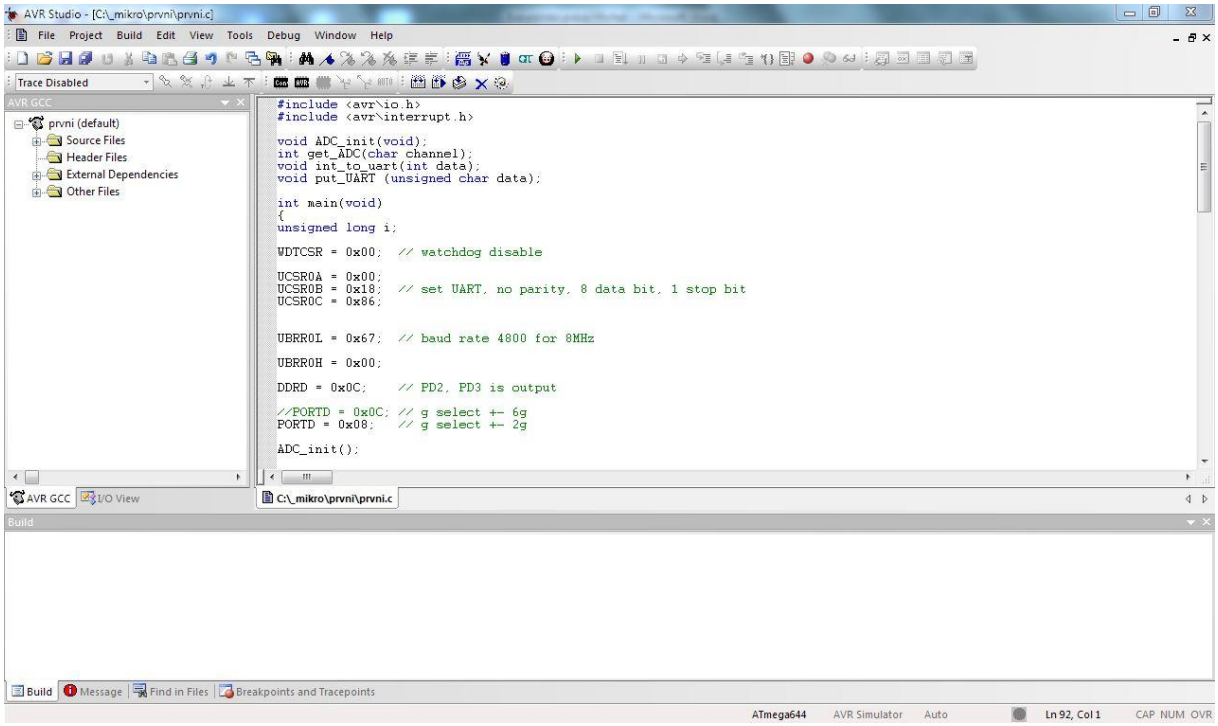


5.1.2 AVR Studio

Na programování AVR procesorů je velké množství softwaru. A to jak v jazyce ASM, C, nebo třeba Pascal. Zdrojový kód procesoru je napsán v jazyce C. Pro odladění a kompilaci kódu byl vybrán program AVR Studio. Je to uživatelsky přívětivý a zdarma stažitelný program.

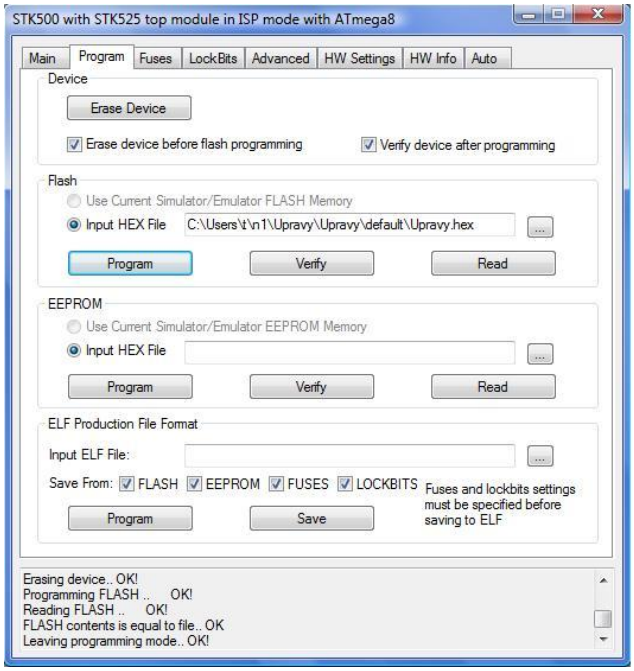
Jedná se o vývojové prostředí firmy Atmel. Obsahuje editor na psaní vlastního programu, ve kterém barevně odlišuje komentáře, klíčová slova atd. Obsahuje *debugger* pro kontrolu a nalezení chyb v programu. Dále také užitečný simulátor AVR procesorů. Kompilátor pro psaní programu v jazyce C je třeba použít externí. WinAVR je sada open source spustitelných nástrojů pro vývoj softwaru procesorů Atmel AVR používané na platformě Windows [6].

Obr. 5.2 AVR Studio



Po odladění programu už zbývá, pouze program nahrát pomocí programátoru do mikroprocesoru.

Obr. 5.3 Naprogramování procesoru



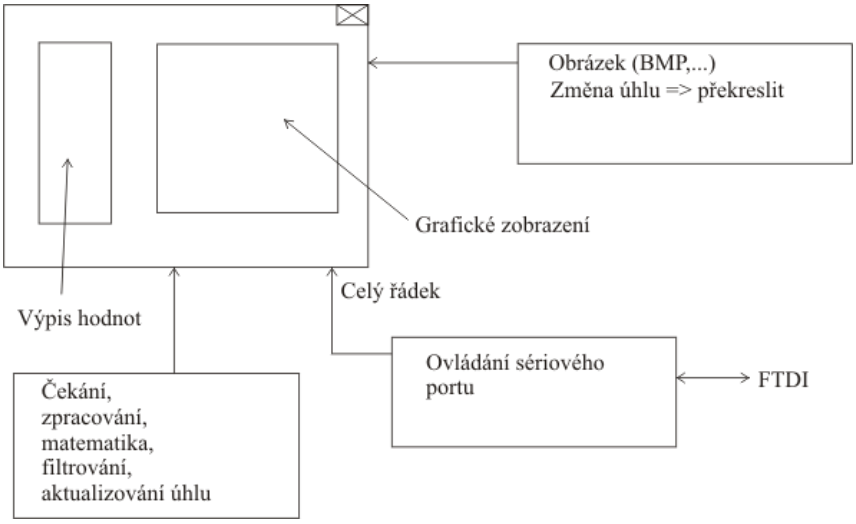
5.2 Zobrazovací aplikace na PC

Ke zpracování a přehlednému zobrazení bylo potřeba vytvořit program. Požadovaným výsledkem je pohyblivý obraz na monitoru počítače, jako kdyby pilot měl k dispozici umělý horizont na přístrojové desce.

Vytvoření takového programu bylo realizováno v Microsoft Visual Studiu 2010 Express. Tato volně stažitelná verze je pro dané využití dostačující. Celý program je napsán v jazyce C#. Jedná se o produkt firmy Microsoft umožňující objektové programování.

Kromě grafického výstupu naklánění umělého horizontu je vhodné vypisovat přijímané hodnoty. V případě, že není navázán signál s vysílací částí zařízení, je tato skutečnost okamžitě patrná a snadněji detekujeme chybu. Na obr. 5.4 je nákres struktury programu.

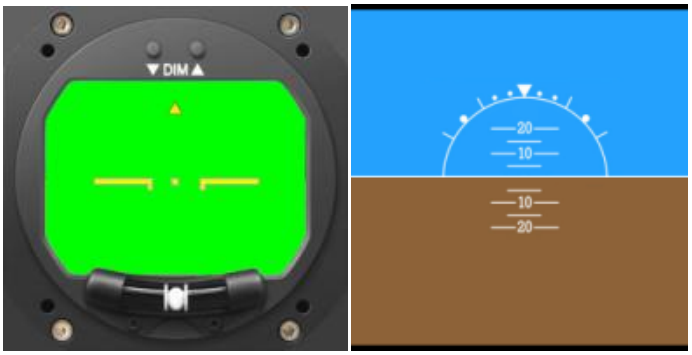
Obr. 5.4 Nákres struktury programu



5.2.1 Grafika

Obrázek je vykreslován ve dvou fázích. Pracuje se s formátem BMP. Rozdělení je patrné na obr. 5.5.

Obr. 5.5 Bitmapy grafiky



Panel přístroje je vykreslován v pevné pozici a druhý obrázek, který představuje zem a oblohu, je pohyblivý [10]. Zelená barva v panelu přístroje je označena jako transparentní. Tím je dosaženo toho, že z každé bitmapy je vidět jen co je potřeba. Princip otáčení je takový, že nejprve dojde k otočení o požadovaný úhel kolem levého horního rohu vykreslené plochy a následně dojde k posunutí obrázku do požadované polohy.

5.2.2 Obsluha sériového portu

Nastavení:

1) 1 start bit, 8 datových bitů, bez parity, 1 stop bit

2) Vytvoření objektu SerialPort

Funkce:

3) Otevření portu

4) Načítání dat

5) Uzavření portu při ukončení programu

Program je ošetřen tak, aby při příjmu chybného formátu řádky data nepoužil a čekal na korektní příjem informace.

5.2.3 Filtrování přijatých dat

Vzhledem k tomu, že motor v letadle je zdrojem vibrací je potřeba zpracovaná data filtrovat. Byl zvolen softwarový filtr (klouzavý průměr). Ten načítá do paměti posledních 10 hodnot. Při příchodu nové hodnoty odebere tu nejstarší. Na obr. 5.6 je příklad funkce filtru se třemi hodnotami.

Obr. 5.6 Filtr

17, 25, 6

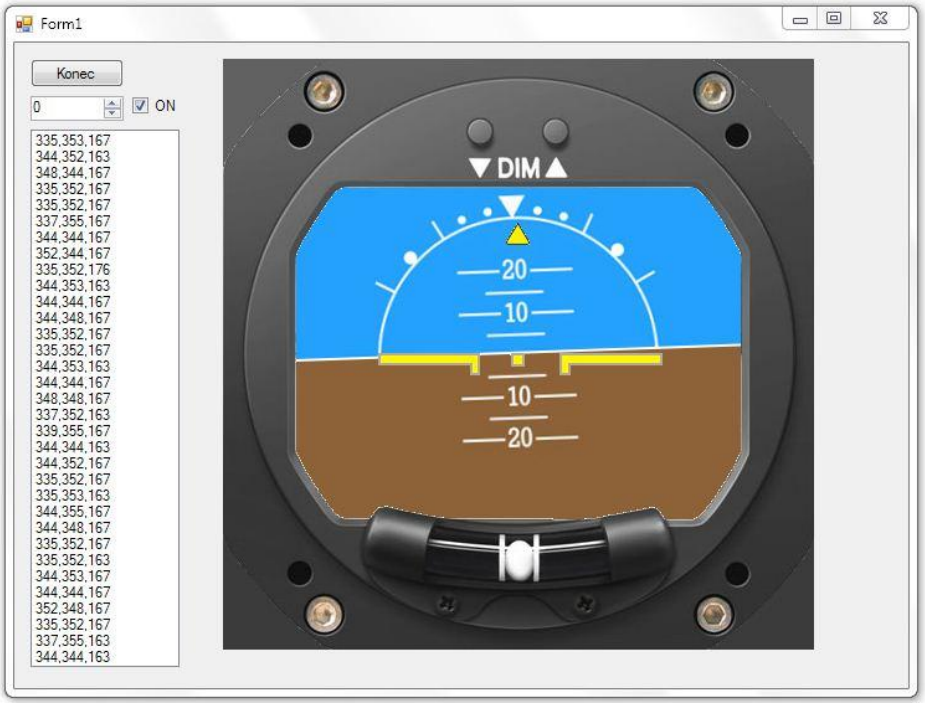
3, 17, 25

-8, 3, 17

5.2.4 Funkce aplikace

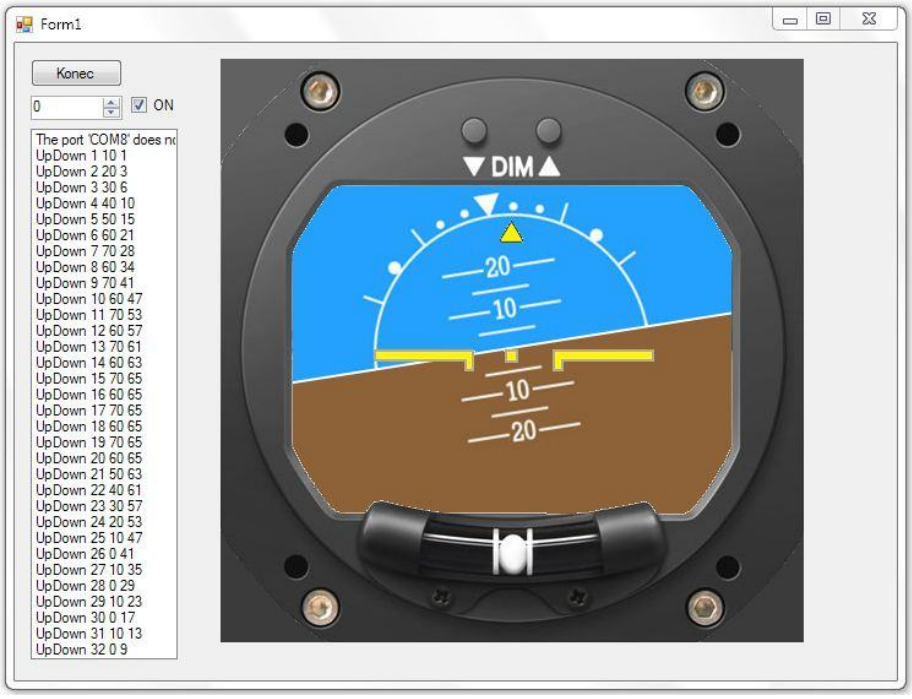
Pokud po sériovém portu přicházejí data, jsou vyhodnocována a zpracována. Kromě funkce grafické části aplikace je zobrazen v levé části okna také ListBox. Zde je vidět posledních 32 hodnot, které přišli po sériovém portu. To je výhodné pro kontrolu navázání komunikace (obr. 5.7).

Obr. 5.7 Aplikace pracující se sériovým portem



Zaškrtnutím okna CheckBox hodnoty nejsou přijímány ze sériového portu, ale je možné je vkládat ručně. Potom jsou v ListBoxu zobrazena tři čísla. První pouze počítá množství hodnot, druhé číslo představuje aktuálně vložený úhel a třetí reprezentuje velikost výsledného úhlu po filtraci. Tato funkce je využívána převážně při ladění aplikace.

Obr. 5.8 Aplikace v ručním režimu



5.2.5 Kalibrace a nastavení

Po připojení k počítači je potřeba nastavit správnou přenosovou rychlost a vhodný port, kam byl FTDI převodník připojen. To je možné ve Visual Studiu, v tabulce nastavení. Dále tato tabulka umožňuje kalibrovat nulovou polohu (obr. 5.9).

Obr. 5.9 Nastavení a kalibrace nulové polohy

	Name	Type	Scope	Value
	comName	string	Application	COM8
	comBitRate	int	Application	4800
	calibrationX	int	Application	5
	calibrationY	int	Application	0
	calibrationZ	int	Application	0
*				

Závěr

Výsledkem práce je přístroj, který je užitečný pro trénink přesné akrobacie modelů letadel. Představené řešení je plně funkční. Finanční náročnost zařízení je nízká, jen vysílací moduly představují vyšší investici. Celková cena se pohybuje kolem 3.000 Kč.

V rámci realizace zadání byl vytvořen hardware na bázi mikroprocesoru, včetně patřičného firmwaru umožňující monitorovat náklon letadla a přenášet naměřená data do PC. Dále byla vytvořena vizualizační aplikace pro PC, která umožňuje přehledně online vizualizovat získaná data, čímž byly splněny všechny body zadání.

Rychlost vyhodnocování dat se během letu ukázala jako dostatečná, takže pilot je schopen udržet rovnovážnou polohu křídel podle přístroje.

Frekvence vysílacího a přijímacího modulu 433,92 GHz je výhodná v tom směru, že řízení letadla je realizováno na 2,4 GHz. Systémy jsou dostatečně frekvenčně oddělené a nedochází k žádnému rušení. Dosah zařízení je dostatečný, a pokud by drát o vlnové délce λ/4 byl nahrazen 50Ω anténou, dosah by se ještě zvýšil.

K měření by bylo možné použít namísto akcelerometru i gyroskop, ale to by značně zvýšilo cenu zařízení.

Výhodou kalibrace je to, že akcelerometr nemusí být naprosto přesně umístěn v trupu modelu letadla. Poté lze snadno rovnovážnou polohu nastavit.

Vzhledem k vibracím letadla během letu se ukázala nejnižší citlivost akcelerometru jako nejvhodnější volba. Jinak je obraz během letu poněkud roztřesený.

Použitá literatura

[1] Wikipedia: Umělý horizont [online]. Aktualizace 11.11.2011 [cit. 12.3.2012]. Dostupné na: <http://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%BD_horizont>

[2] HW: 3D akcelerometry MEMS freescale [online]. Aktualizace 30.4.2007 [cit. 26.3.2012]. Dostupné na: <http://www.hw.cz/soucastky/jak-pracuji-nove-3d-mems-akcelerometry-freescale.html >

[3] Datasheet akcelerometr MMA7260QT [online]. Aktualizace 12.5.2011 [cit.26.3.2012]. Dostupné na: <http://www.pololu.com/file/download/MMA7260QT.pdf?file_id=0J87>

[4] HW: Digitální vodováha [online]. Aktualizace 15.6.2009 [cit. 30.3.2012]. Dostupné na: <http://www.hw.cz/teorie-a-praxe/konstrukce/digitalni-vodovaha.html >

[5] ROOT: Mikroprocesory s architekturou RISC [online]. Aktualizace 15.6.2009 [cit. 5.4.2012]. Dostupné na: < http://www.root.cz/clanky/mikroprocesory-s-architekturou-risc-i/>

[6] HW: Začínáme s mikroprocesory Atmel AVR [online]. Aktualizace 1.12.2006 [cit. 16.4.2012]. Dostupné na: <http://www.hw.cz/teorie-a-praxe/zaciname-s-mikroprocesory-atmel-avr.html>

[7] Datasheet procesor ATmega 644 [online]. Aktualizace 12.5.2011 [cit.15.3.2012]. Dostupné na: <http://www.atmel.com/Images/doc2593.pdf>

[8] Datasheet modul XTR 434 [online]. Aktualizace 16.12.2002 [cit.23.4.2012]. Dostupné na: <http://www.aurelwireless.com/wp-content/uploads/user-manual/650200588G_um.pdf>

[9] Eagle [online]. Aktualizace: 12.12.2011 [cit. 25.4.2012]. Dostupné na: <http://www.eagle.cz/>

[10] Sarasota Avionics international [online]. Cit. 2.5.2012. Dostupné na: <http://sarasotaavionics.com/avionics/rca2600_3in#>

Seznam příloh

Příloha A:

```
#include <avr\io.h>
#include <avr\interrupt.h>

void ADC_init(void);
int get_ADC(char channel);
void int_to_uart(int data);
void put_UART (unsigned char data);

int main(void)
{
unsigned long i;

WDTCSR = 0x00;      // watchdog disable

UCSR0A = 0x00;
UCSR0B = 0x18;      // set UART, no parity, 8 data bit, 1 stop bit
UCSR0C = 0x86;


UBRR0L = 0x67;      // baud rate 4800 for 8MHz

UBRR0H = 0x00;

DDRD = 0x0C;        // PD2, PD3 is output

//PORTD = 0x0C;     // g select +- 6g
PORTD = 0x08;       // g select +- 2g

ADC_init();



while(1){

for(i=0; i<300; i++);  // zpomalovaci smycka

i=get_ADC(0);
//put_UART('x');
int_to_uart(i);
put_UART(',');
//put_UART(10);  //enter

i=get_ADC(1);
//put_UART('y');
int_to_uart(i);
put_UART(',');
//put_UART(10);
```

```
i=get_ADC(2);
//put_UART('z');
int_to_uart(i);
//put_UART(10);
put_UART(10);

}

}

void put_UART (unsigned char data){
UDR0 = data;
while( (UCSR0A & 0x40) == 0);      // wait until data are not send
UCSR0A = UCSR0A | 0x40;
}

void int_to_uart(int data){
unsigned char c;


c = (unsigned char) (((data % 1000)/100)+ 48);
put_UART(c);

c = (unsigned char) (((data % 100)/10)+ 48);
put_UART(c);

c = (unsigned char) ((data % 10) + 48);
put_UART(c);

}

void ADC_init(void){

ADMUX = 0x40;
ADCSRA = 0x80;
ADCSRB = 0x00;
}

int get_ADC(char channel){

int result;

channel = channel & 0x07;    // channel has max value 7

ADMUX = ADMUX & 0xF8;
ADMUX = ADMUX | channel;        // set ADC channel


ADCSRA = ADCSRA | 0x40;          // start conversion
```

```
while( (ADCSRA & 0x10) == 0);     // wait until conversion is not completed
ADCSRA = ADCSRA | 0x10;           // clear completed flag



result = ((int) (ADCL));
result = result | (((int) (ADCH))<<8);
return result;
}
```

Příloha B:

Modul Horizont:

```
using System;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
using System.IO;
using Horizont2W.Properties;

namespace Horizont2W
{
    public partial class Horizont2 : Form
    {
        private Settings cfg;
        matika mat = new matika();
        public int pocitadlo = 0;
        public Horizont2()
        {
            InitializeComponent();
            cfg = Settings.Default;
            mat.setCalibraion(cfg.calibrationX, cfg.calibrationY, cfg.calibrationZ);
            serialPort1.BaudRate = cfg.comBitRate;
            serialPort1.PortName = cfg.comName;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            int angle = (int)numericUpDown1.Value;


            if (checkBox1.Checked)
            {
                int xangle = mat.processAngle(angle);   //xangle hodnota po filtraci
                obrazek1.SetAngle(xangle);
                pocitadlo++;
                status(String.Format("UpDown {0} {1} {2}", pocitadlo, angle, xangle));
            }
        }

        private void parseLine(String s)
        {
            status(s);

            double angle = mat.parseLineToAngle(s);

            if (!checkBox1.Checked)
                obrazek1.SetAngle(mat.processAngle(angle));
        }

        delegate void statusDelegate(String s);
        void status(String s)
        {
            if (this.InvokeRequired)
            {
                this.Invoke(new statusDelegate(status), new object[] { s });
                return;
            }
```

```
            if (listBox1.Items.Count > 32)
                listBox1.Items.RemoveAt(0);

            listBox1.Items.Add(s);
        }

        char[] radek = new char[32];
        int radekUkaz = -1;

        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
            // dialog nastaveni seriaku, prijem od konce radku do konce radku
        {
            if (e.EventType != SerialData.Chars)
                return;

            while (serialPort1.BytesToRead > 0)
            {
                int x = serialPort1.ReadByte();
                if (x == '\n')
                {
                    if (radekUkaz == -1)
                    {
                        radekUkaz = 0;
                    }
                    else
                    {
                        parseLine(new String(radek));
                        radekUkaz = 0;
                    }
                }
                else
                {
                    if (radekUkaz >= 0)
                    {
                        radek[radekUkaz] = (char)x;
                        radekUkaz++;
                        // TEST na > 32 !!!
                    }
                }
            }

        }

        private void Horizont2_Load(object sender, EventArgs e)
        {
            try
            {
                serialPort1.Open();
            }
            catch (IOException ex)
            {
                status(ex.Message);
            }
        }

        private void Horizont2_FormClosed(object sender, FormClosedEventArgs e)
        {
        }

        private void Horizont2_FormClosing(object sender, FormClosingEventArgs e)
```

```
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.DataReceived -= serialPort1_DataReceived;
                serialPort1.DiscardInBuffer();
                serialPort1.DiscardOutBuffer();

                Thread.Sleep(500);

                serialPort1.Close();
                Thread.Sleep(100);
            }
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked)
            {
                if (serialPort1.IsOpen)
                {
                    serialPort1.DiscardInBuffer();
                    serialPort1.DiscardOutBuffer();
                    serialPort1.Close();
                }
            }
            else
            {
                try
                {
                    serialPort1.Open();
                }
                catch (IOException ex)
                {
                    status(ex.Message);
                }
            }
        }
    }
}
```

Modul Obrázek:

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace Horizont2W
{
    public class Obrazek : Control
    {
        private Bitmap imgHorizontBg = null;    // obrazek s pozadim
        private Bitmap imgHorizontFg = null;    // obrazek s pristrojem
        private int angle = 0;      // uhel nakloneni letadla (po smeru hod. rucicek)

        public Obrazek()
        {
            try
            {
            imgHorizontBg = new Bitmap("images\\horizontBG.bmp"); // obrazek s pozadim
            imgHorizontFg = new Bitmap("images\\horizontFG.bmp"); // obrazek s
pristrojem
```

```
                imgHorizontFg.MakeTransparent(Color.FromArgb(0, 255, 0));    // zelena
barva bude transparentni
            }
            catch (Exception)
            {
            }

            this.DoubleBuffered = true;

        }

        public void SetAngle(int a)
        {
            angle = a;
            Invalidate();
        }

        protected override void OnPaint(PaintEventArgs pe)
        {


                float rotationAngle = (float) angle;   // uhel otoceni obrazku (opacne
naklopeni nez naklon letadla)
                Point originPt = new Point(0, 0);       // pocatek souradnic

                if ((imgHorizontBg != null) && (imgHorizontFg != null))
                {
                    Bitmap rotatedBg = rotateImage(imgHorizontBg, rotationAngle);   //
otoceni pozadi o dany uhel
                    pe.Graphics.DrawImage(rotatedBg, originPt);        // vykresleni
otoceneho obrazku
                    pe.Graphics.DrawImage(imgHorizontFg, originPt);    // vykresleni
vzhledu pristroje
                }

            // Calling the base class OnPaint
            base.OnPaint(pe);
        } // OnPaint()

        /// <summary>
        /// Otoceni obrazku o dany uhel.
        /// </summary>
        /// <param name="img">Obrazek k otoceni.</param>
        /// <param name="angle">Uhel otoceni ve stupnich po smeru hodinovych
rucicek.</param>
        /// <returns>Otoceny obrazek.</returns>
        private Bitmap rotateImage(Bitmap img, float angle)
        {
            float moveX = (float)img.Width / 2;
            float moveY = (float)img.Height / 2;
            Bitmap newImage = new Bitmap(img.Height, img.Width);

            Graphics g = Graphics.FromImage(newImage);
            g.TranslateTransform(moveX, moveY);
            g.RotateTransform(angle);
            g.TranslateTransform(-moveX, -moveY);
            g.DrawImage(img, new Point(0, 0));
            return newImage;
        }
```

```
    }
}
```

Modul Matematika:

```
using System;

namespace Horizont2W
{
    public class matika
    {
        private double calibX = 335;    // vychozi namerena kalibrace v ose X
        private double calibY = 312;    // vychozi namerena kalibrace v ose Y
        private double calibZ = 316;    // vychozi namerena kalibrace v ose Z


     private const double xAxisMin = 170;   // vychozi rozsah hodnot v ose X (minimum)
     private const double xAxisMax = 500;   // vychozi rozsah hodnot v ose X (maximum)
     private const double yAxisMin = 130;   // vychozi rozsah hodnot v ose Y (minimum)
     private const double yAxisMax = 490;   // vychozi rozsah hodnot v ose Y (maximum)
     private const double zAxisMin = 150;   // vychozi rozsah hodnot v ose Z (minimum)
     private const double zAxisMax = 490;   // vychozi rozsah hodnot v ose Z (maximum)

        /// <summary>
        /// Nastavi kalibraci v jednotlivych osach (dokalibrovani vychozich hodnot).
        /// </summary>
        /// <param name="calibrationX">Kalibrace osy X.</param>
        /// <param name="calibrationY">Kalibrace osy Y.</param>
        /// <param name="calibrationZ">Kalibrace osy Z.</param>
        public void setCalibraion(double calibrationX, double calibrationY, double
calibrationZ)
        {
            this.calibX += calibrationX;
            this.calibY += calibrationY;
            this.calibZ += calibrationZ;
        }

        /// <summary>
        /// Overi, zda je dane cislo v intervalu (min,max)
        /// </summary>
        /// <param name="value">Overovana hodnota</param>
        /// <param name="min">Zacatek intervalu</param>
        /// <param name="max">Konec intervalu</param>
        /// <returns>TRUE pokud se cislo nachazi v danem intervalu, jinak
FALSE</returns>
        private bool checkLimits(double value, double min, double max)
        {
            if (min < value && value < max)
            {
                return true;
            }
            return false;
        }

        private double[] angleBuffer = new double[10];   // poslednich 10 hodnot uhlu

        /// <summary>
        /// Prumeruje poslednich 10 nactenych hodnot.
        /// </summary>
        /// <param name="newAngle">Posledni nactena hodnota</param>
        /// <returns>Prumerna hodnota</returns>
```

```
        private int filterAngle(double newAngle)
        {
            double sum = newAngle; // soucet pole
            for (int i = 1; i < angleBuffer.Length; i++)
            {
                sum += angleBuffer[i];
                angleBuffer[i - 1] = angleBuffer[i];

            }
            angleBuffer[angleBuffer.Length - 1] = newAngle;

            int avgAngle = (int)Math.Round(sum / angleBuffer.Length);
            return avgAngle;
        }

        /// <summary>
        /// Parsuje vstupni radek se souradnicemi na 3 ciselne souradnice
        /// </summary>
        /// <param name="line">Radek obsahujici souradnice oddelene carkou v poradi
X,Y,Z</param>
        public double parseLineToAngle(String line)
        {
                double xAxis, yAxis, zAxis;

                string[] parts = line.Split(',');
                if ((parts.Length == 3) &&
                    (double.TryParse(parts[0], out xAxis)) &&
                    (double.TryParse(parts[1], out yAxis)) &&
                    (double.TryParse(parts[2], out zAxis)))
                {
                    if (checkLimits(xAxis, xAxisMin, xAxisMax) && // kontrola limitu
hodnot
                        checkLimits(yAxis, yAxisMin, yAxisMax) &&
                        checkLimits(zAxis, zAxisMin, zAxisMax))
                    {
                        xAxis -= calibX;    // kalibrace hodnot podle konfiguracniho
souboru
                        yAxis -= calibY;
                        zAxis -= calibZ;


                        const double rad2deg = 180 / Math.PI;   // prepocet radianu na
stupne
                       double gm = Math.Sqrt(Math.Pow(xAxis, 2) + Math.Pow(zAxis, 2));
// vypocet bez Y osy
                        double angle = (Math.Asin(xAxis / gm) * rad2deg);
// goniometricky vzorec


                        if (zAxis > 0)                  //  rozpoznani letu na zadech
                        {
                            Console.WriteLine("vzhuru nohama");
                            angle = 180 - angle;        //  zobrazeni letu na zadech

                        }

                       Console.WriteLine("x=" + xAxis + ", y=" + yAxis + ", z=" +
zAxis);


                        return angle;
                    }
                }
```

```
                else
                {
                    Console.WriteLine("Chyba pri parsovani radky ze serioveho
portu!");
                }


            return 0;
        }

        public int processAngle(double angle)
        {

            Console.WriteLine("angle=" + angle);


            int guiAngle = filterAngle (angle);

            Console.WriteLine("guiangle=" + guiAngle);

            Console.WriteLine("---------------------------");

            return guiAngle;

        }
    }

}
```