České vysoké učení technické v Praze

Fakulta elektrotechnická

Skupina inteligentní a mobilní robotiky

Technický manuál
řídicí deska multicopter

Tomáš Báča

Obsah

1 Hardware

1.1 Letoun

Letoun, jinak běžně nazývaný kvadrukoptera, je helikoptera se čtyřmi rotory. Listy mají pevný úhel náběhu a jejich tah se řídí změnu rychlosti jejich otáčení. Díky tomu je stroj konstrukčně jednoduchý a relativně odolný proti poškození (v porovnání s běžnou helikopterou). Z toho důvodu je vhodný pro experimentální použití. Stroj je napájen z Li-Poly (Lithium-Polymerová) baterie. Doba letu se pohybuje od 5 do 15 minut v závislosti na zatížení.

Samotný letoun je nestabilní systém vyžadující stabilizaci. O tu se stará výrobcem dodaná stabilizační deska FlightCTRL (Obrázek 2). Ta je osazená 3-osými MEMS gyroskopy a akcelerometry, ze kterých syntetizuje údaj o úhlu náklonu v ose Pitch a Roll. Ty potom používá pro stabilizaci letounu. Dále se deska stará o mixování vstupních signálů (výroba virtuálních vstupů) a řízení jednotlivých vrtulí.

Vstupními řídicími signály do desky jsou:

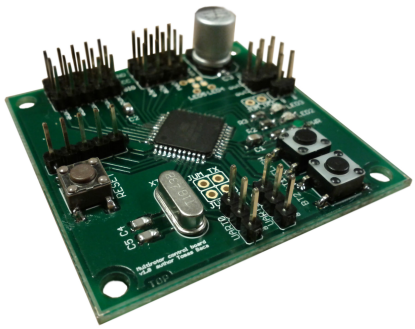
THROTTLE - Kolektivní tah všech rotorů v procentech jejich max. otáček
ELEVATOR - Náklon letounu v ose dopředu/dozadu (pitch, výškovka), úhlový rozměr
AILERON - Náklon letounu v ose doleva/doprava (roll, křidélka), úhlový rozměr
RUDDER - Rotace kolem svislé osy (yaw, směrovka), rozměr úhlové rychlosti

Vstupní signál se do FlightCTRL zavádí po jednom vodiči pomocí Pulze Poziční Modulace (PPM). V desce je stabilizace implementována formou **úhlového regulátoru**. Všechny tři osy (yaw, pitch, roll) jsou stabilizovány na nulový úhel. Před startem je vždy třeba provést kalibraci na rovné zemi, kde si jednotka nastaví referenční nulový úhel.

Obrázek 1: Stabilizační deska FlightCTRL

1.2 Řídicí deska

Řídicí deska plní funkci komunikačního uzlu mezi moduly a malé výpočetní jednotky pro další stabilizaci a řízení letounu. Je osazena mikrokontrolerem ATmega164p s 1kb RAM a 16kb ROM (pro program). Procesor kontroleru trpí absencí FPU (floating point unit). Ke komunikaci jsou zde 2 porty UART. Programování mikrokontroleru probíhá přes SPI.



Obrázek 2: Řídicí deska kvadrukoptery

Deska má 9 vstupů pro Pulzně Šířkovou Modulaci (PWM). Použity jsou pro příjem signálů z RC přijímače. Dále jsou zde dva volitelné výstupy, z nichž jeden je použit pro PPM výstup do stabilizační desky FlightCTRL a druhý je volitelný.

Pro signalizaci jsou k dispozici 2 LED, žlutá a červená, třetí se dá případně připojit do volitelného výstupu.

Deska je osazena dvěma volitelnými tlačítky a jedním tlačítkem **reset**.

V minimální konfiguraci plní funkci konvertoru signálů z RC soupravy. Převádí PWM z přijímače na PPM pro FlightCTRL a umožňuje tedy ruční řízení letounu. Z důvodu bezpečnosti doporučuji vždy **zachovat majoritní vliv ručního řízení** a vyvíjené regulátory mixovat se signály z RC soupravy se značnou saturací. Zamezíte tím případné neovladatelnosti stroje. Více viz kapitola 2.1.2.

2 Software

2.1 Struktura software

Software pro kontroler je napsán v programovacím jazyce C. Hlavním souborem je **main.c**. Zde je hlavní smyčka programu a předpis funkcí pro přerušení přerušeními procesoru.

Většinu času tráví procesor v nekonečné smyčce ve funkci **main()**. Zde čeká na asynchronní obsluhu komunikace, nebo periodické volání řídicích funkcí apod. Je velmi důležité, aby co možná všechny výpočty, zpracování a volání funkcí probíhaly voláním z funkce **main()**. Jinak bude procesor zablokován (přerušení nebude vyvoláno, pokud procesor vykonává funkci jiného přerušení) a nebude zpracovávat komunikaci, což může vést k neovladatelnosti letounu.

Kód je strukturován pomocí podmínek preprocesoru. Pokud např. nebudete potřebovat kamerový modul s počítačem gumstix, lze v souboru **config.h** definovat `GUMSTIX_DATA_REVEICE` na hodnotu `DISABLED`. Všechen kód a proměnné týkající se příjmu dat a volání funkcí, s tímto modulem spojených, bude vyřazeno z kompilace.

2.1.1 main.c

Kód **main.c** obsahuje definici globálních proměnných. Všechny proměnné, které si mají uchovat hodnotu, musí být deklarovány zde. Všechny proměnné doporučuji definovat jako **volatile**, předejdete nečekaným problémům s jejich měnícím se obsahem.

Je zde definována funkce **main()** v níž probíhá konfigurace procesoru po spuštění a následně zanoření do nekonečné smyčky.

Dále se zde nachází obsluha přerušení, viz. Tabulka 1.

V nekonečné smyčce jsou kontrolovány vlajky pro asynchronní volání funkcí. Např., poté, co přijde poslední znak zprávy z px4flow, nastaví se vlajka **px4flowDataFlag** na hodnotu 1. Ta je poté v nekonečné smyčce odchycena a jsou vykonány potřebné úkony spojené s příjmem dat - filtrace, uložení aktuálních hodnot do stavových proměnných, apod.

2.1.2 controllers.c

controllers.c obsahuje funkce pro řízení letounu.

2.1.3 system.c

system.c obsahuje funkce pro obsluhu kontroleru a letounu. Viz. Tabulka 2.

ISR(USART0_RX_vect)	příjem z UART0
ISR(USART1_RX_vect)	příjem z UART1
ISR(TIMER1_COMPA_vect)	Komparační přerušení A k 16bit čítači. Používá se pro generování vstupního PPM signálu (start pulzu). **Neupravovat, pokud nevím, co dělám!!**
ISR(TIMER1_COMPB_vect)	Komparační přerušení B k 16bit čítači. Používá se pro generování vstupního PPM signálu (konec pulzu). **Neupravovat, pokud nevím, co dělám!!**
ISR(PCINT0_vect)	Zpracování změny na vstupních PWM pinech 1..4 **Neupravovat, pokud nevím, co dělám!!**
ISR(PCINT1_vect)	Zpracování změny na vstupních PWM pinech 5..9 **Neupravovat, pokud nevím, co dělám!!**
ISR(TIMER0_OVF_vect)	Zpracování přetečení 8bit čitače (hrubé měření času).

Tabulka 1: Použitá přerušení

initializeMCU()	Volá se jednou, po zapnutí. Provede nastavení mikrokontroleru. **Neupravovat, pokud nevím, co dělám!!**
enableController()	Zapne automatickou stabilizaci letounu.
disableController()	Vypne automatickou stabilizaci letounu.
armVehicle()	Provede automatické "armování"letounu. NEPOUŽÍVAT!.
disarmVehicle()	Provede automatické "disarmování"letounu.
button1check()	Kontrola stisku tlačítka 1.
button2check()	Kontrola stisku tlačítka 2.

Tabulka 2: Funkce v system.c

USART0_init()	inicializace UART0
USART1_init()	inicializace UART1
Uart0_write_char()	zapíše bajt na UART0
Uart0_write_string()	zapíše string na UART1
atomParseChar()	zpracuje příchozí znak z "Atomového"počítače (použití pro surfnav)
gumstixParseChar()	zpracuje příchozí znak z Gumstixu
flightCtrlParseChar()	zpracuje příchozí znak z FlightCTRL stabilizační desky
Decode64()	dekóduje zprávu z FlightCTRL
parseFlightCtrlMessage()	zpracuje zprávu z FlightCTRL stabilizační desky
mergeSignalsToOutput()	provádí míchání signálů regulátorů a RC vysílače. **Neupravovat, pokud nevím, co dělám!!**
px4flowParseChar()	zpracuje příchozí znak ze senzoru px4flow
my_mavlink_parse_char()	upravená funkce z knihovny MavLink. zpracuje znak ze senzoru px4flow.
capturePWMInput()	měří délku PWM pulzů z RC soupravy. **Neupravovat, pokud nevím, co dělám!!**

Tabulka 3: Funkce v communication.c

2.1.4 communication.c

communication.c obsahuje funkce pro obsluhu a zpracování komunikace s externími moduly. Některé funkce zpracovávají jednotlivé příchozí bajty a jsou tedy určeny pro volání zevnitř přerušení. Jiné zpracovávají celou komunikační zprávu a musejí být volání asynchronně, zevnitř smyčky, po přijetí posledního bajtu.

2.1.5 config.h

config.h obsahuje direktivy preprocesoru pro konfiguraci celého firmwaru. Lze zde zapínat jednotlivé moduly (px4flow, gumstix, atomový PC, FligthCTRL) a nastavovat chování firmwaru. Dále se zde konfigurují seriové linky (jejich BAUD rate) a jejich přiřazení k modulům.

Důležité je nastavení konstant pro PWM a PPM. Jsou zde hodnoty délek pulzů (min, střední, max), délka PPM rámce a délka dělícího pulzu v PPM.

Dále je zde namapování kanálů z RC na jednotlivé PWM vstupy.

FRAME_ORIENTATION	Nastavení orientace letounu (PLUS_COPTER, nebo X_COPTER). Má vliv na úhly vyčítané z FlightCTRL, ty jsou vždy relativně k desce FlightCTRL.
GUMSTIX_CAMERA_POINTING	Kam míří kamera Gumstix počítače (FORWARD, nebo DOWNWARD), důležité pro rotaci souřadnic.
disableController()	Vypne automatickou stabilizaci letounu.
armVehicle()	Provede automatické "armování"letounu. NEPOUŽÍVAT!.
disarmVehicle()	Provede automatické "disarmování"letounu.
button1check()	Kontrola stisku tlačítka 1.
button2check()	Kontrola stisku tlačítka 2.

Tabulka 4: Obsah config.h

2.2 Kompilace pro ATmega164p

Ke kompilaci je třeba mít nainstalovaný kompilátor **avr-gcc**. Pro windows je obsažen v balíčku **WinAVR**, ke stažení na adrese http://winavr.sourceforge.net/. Pro Linux je potřeba balíčky **gcc-avr**, **binutils-avr**, **avr-libc**, **avrdude**, **gdb-avr**.

```
sudo apt-get install gcc-avr binutils-avr gdb-avr avr-libc avrdude
```

Pro samotnou kompilaci je přítomen soubor Makefile, který obsahuje předpis pro všechny výše popsané soubory. Výstupem kompilace je soubor **main.hex**, který je vstupním parametrem pro upload do mikrokontroleru. V Linuxu je kompilace prostá - zavoláním příkazu `make all` ve složce se zdrojovými soubory. Ve Windows je postup stejný, pokud jste si nainstalovali výše zmíněný toolchain. Ten do windows doinstaloval program make, který umí interpretovat Makefile stejně, jako v Linuxu.

2.3 Upload progamu do ATmega164p

Po úspěšné kompilaci máte ve složce se zdrojovými soubory soubor **main.hex**. K jeho nahrání do kontroleru potřebujete programátor (např. USBasp) a program **avrdude**. Příkaz pro upload souboru je shodný pro Windows i Linux, pouze v Linuxu je nutné volat ho s právy superuživatele.

```
avrdude -p m164P -c usbasp -U flash:w:main.hex
```

POZOR! Před nahráváním si vždy zkontrolujte, zdali je do řídicí desky řádně přivedeno napájení. Pokud je deska bez napájení, může nahrávání firmwaru nenávratně poškodit kontroler.