

Spojená škola, o. z.
Stredná priemyselná škola elektrotechnická S. A. Jedlika
Komárňanská 28, Nové Zámky

Stredoškolská Odborná Činnosť

č. odboru: 12
Elektronika, hardware, mechatronika

DRONELOGGER3

2019
Nové Zámky

riešitelia:
Richard Kováč
Marco Pintér
ročník štúdia: štvrtý

Spojená škola, o. z.
Stredná priemyselná škola elektrotechnická S. A. Jedlika
Komárňanská 28, Nové Zámky
Nitriansky samosprávny kraj

Stredoškolská Odborná Činnosť

č. odboru: 12
Elektronika, hardware, mechatronika

DRONELOGGER3

2019
Nové Zámky

riesitelia:
Richard Kováč
Marco Pintér
ročník štúdia: štvrtý

konzultant:
Ing. František Hortai

Čestné vyhlásenie

Týmto čestne vyhlasujem, že sme celú túto prácu, výskum, vývoj a návrh hardvéru robili samostatne, s použitím uvedenej literatúry.

Vyhlasujem, že danú prácu som neprihlásil a neprezentoval v žiadnej inej súťaži, ktorá je schválená Ministerstvom školstva, vedy, výskumu a športu SR.

Som si vedomý, že pokiaľ by mnou uvedené vyhlásenie nebolo pravdivé, budem čeliť všetkým z toho vyplývajúcim následkom.

Nové Zámky, 21. január 2019

vlastnoručný podpis

Pod'akovanie

Chceli by som sa pod'akovať nášmu konzultantovi, Ing. Františkovi Hortaiovi, za vedenie a cenné pripomienky pri záverečnom spracovaní práce a Ing. Tiborovi Molnárovi za cenné rady pri riešení práce.

Obsah

Úvod	6
Cieľ práce	7
Metodika práce	8
1 Hardvér tretej verzie - časť pilota	9
1.1 Plastové diely častí na a v RC súprave	9
1.1.1 Plastové diely zobrazovacieho panelu	9
1.1.2 Plastové diely modulu v RC súprave	10
1.2 Elektronika častí na a v RC súprave	12
1.2.1 Elektronika zobrazovacieho modulu	12
1.2.2 Elektronika modulu RC súpravy	13
1.2.3 Schéma hlavnej logiky	13
1.2.4 Schéma bezdrôtovej komunikácie	14
1.2.5 Schéma napájanie a VS	15
1.3 Dizajn DPS	16
2 Hardvér tretej verzie - časť bezpilotného stroja	17
2.1 Dizajn a konštrukčné riešenie plastových dielov	17
2.1.1 Horný a spodný diel krabičky	18
2.2 Schéma zapojenia, odôvodnenie výberu súčiastok	20
2.2.1 Úsek mikrokontroléra	22
2.2.2 Úsek bezdrôtovej komunikácie	22
2.2.3 Úsek snímania pozície	23
2.2.4 Úsek správy interného akumulátora	25
2.2.5 Úsek regulácie napäťia	25
2.2.6 Úsek vstupov / výstupov a úsek filtrovania	26
2.3 Dizajn DPS	28

Závery práce	29
Resumé	30
Zoznam použitej literatúry	31
Prílohy A, B, C	33
Príloha D - Hardvérové revízie	38
2.4 Hardvérové revízie	38
2.4.1 Rev. nultá	38
2.4.2 Rev. prvá a druhá	39
2.4.3 rev. tretia	40
Príloha E - Fungovanie systému	41
2.5 Modul RC súpravy	41
2.6 Modul Bezpilotného stroja	42
Príloha F - Softvér - vývojové prostredia	44
2.7 Vývojové prostredie Arduino IDE	44
2.8 Vývojové prostredie Nextion Editor	45
2.8.1 HMI displej Nextion	46
Príloha G - Softvér tretej a druhej verzie - Arduino kód	47
2.9 Bezdrôtová sériová komunikácia	47
2.9.1 DroneLogger Air – odosielanie dát	48
2.9.2 DroneLogger Ground – prijímanie odoslaných dát	49
2.10 Uhol náklonu	51
2.10.1 Načítavanie údajov zo senzora	51
2.10.2 Výpočet uhlu náklonu	51
2.10.3 Aplikovanie Kalmanovho filtra na výpočet uhlu náklonu	52
2.11 GPS	53
2.11.1 Komunikácia s GPS modulom	53
2.11.2 Čítanie hodnôt z GPS modulu	53
2.11.3 Výpočet vzdialenosťi od miesta štartu	54
2.11.4 Výpočet azimutu k miestu štartu	54
2.12 Kompas	55
2.12.1 Komunikácia s kompasom	55

2.12.2	Načítavanie údajov zo senzora	55
2.13	Barometer	56
2.13.1	Komunikácia s barometrom	56
2.13.2	Načítavanie údajov z barometra	57
2.14	SD Karta	57
2.14.1	Komunikácia s SD kartou	57
2.14.2	Formát SD karty	58
Príloha H - Softvér tretej verzie - Nextion HMI		59
2.15	Editor Nextion	59
2.16	Ovládanie grafiky cez UART	60
2.16.1	Ošetrenie výpisu	60
2.16.2	Funkcia writeToNextion()	61
2.17	Domovská obrazovka	61
2.18	Hlavná obrazovka s letovými prístrojmi	62
2.18.1	Tvorba a dizajn letových prístrojov	62
2.18.2	Tvorba ručičky kompasu	62
2.18.3	Tvorba umelého horizontu	63
Príloha I - Softvér druhej verzie - Displeje na báze ILI9341		66
2.19	Výpis letových údajov na displej	66
2.19.1	Aktualizovanie textu podľa aktuálnej predlohy	67
2.20	Komunikácia s displejom ILI9341	67
2.21	Vykresľovanie letových prístrojov	68
2.21.1	Grafika umelého horizontu	68
2.21.2	Aktualizovanie grafiky umelého horizontu	69
2.21.3	Grafika kompasu	69
2.21.4	Aktualizovanie grafiky kompasu	70
2.22	Dotyková vrstva	70
2.22.1	Načítavanie vstupov dotykovej vrstvy	70

Úvod

Bezpilotné lietajúce stroje, po Anglicky UAV (Unmanned Aerial Vehicle) alebo ľudovo „Drony“ sú každým dňom prístupnejšie aj pre neskúsených užívateľov. Tento rýchly nárast amatérov bez základných znalostí konštrukcie či ovládania spôsobil aj nárast havarijných situácií a incidentov [1] – ako je vidieť aj v médiách.

Na trhu sa nachádza aj druhá skupina užívateľov – profesionáli, ktorí majú lietajúce stroje osadené drahým vybavením a aj malá chyba môže mať fatálne (finančné) následky. Rozhodli sme sa vyriešiť problémy oboch týchto skupín jedným zariadením DroneLogger ktoré zaznamenáva polohu pripojeného stroja v priestore – náklon na všetkých osiach a jeho polohu v priestore. Zariadenie varuje pilota pri prekročení maximálneho dovoleného náklonu a navedie ho na jeho stabilizáciu.

Zariadenie pozostáva z dvoch častí – „vysielacej“ – DroneLogger Air ktorá je umiestnená na lietajúcim stroji a „prijímacej“ – DroneLogger Ground ktorá je umiestnená priamo v RC súprave užívateľa. Užívateľ môže sledovať polohu lietajúceho stoja na farebnom LCD displeji a zároveň sa údaje o polohe zaznamenávajú na SD kartu. Záznam je štandardizovaný a po lete je možný jeho import do Google máp.

V práci sme sa zaobrali vývojom ako hardvéru pre obe zariadenia - od návrhu 3D modelov krabičiek cez výber elektronických komponentov až po vytvorenie schém a výroby plošných spojov na mieru tak aj softvérom - od návrhu bezdrôtovej komunikácie, komunikácie s jednotlivými senzormi až po graficko-užívateľské rozhranie. V práci vás oboznámim s vývojom minulých aj terajších verzií systému.

Cieľ práce

Cieľom práce bolo vytvoriť systém schopný:

- zaznamenávať polohu lietajúceho zariadenia, teda jeho GPS súradnice, výšku, rýchlosť a náklon
- zobrazovať zaznamenané dátá na displeji podobe dynamickej grafiky – umelého horizontu a kompasu
- navrhnuť spôsob bezdrôtovej komunikácie medzi zariadením na lietajúcom stroji a zariadením umiestnenom v RC súprave užívateľa
- zaznamenávanie získaných dát na SD kartu

Pri návrhu hardvéru sme museli brať ohľad na rôzne aspekty systému – užívateľské pohodlie, napájanie z RC súpravy a lietajúceho stroja, rušenie z motorov vibračné a elektromagnetické a mnoho ďalších.

Metodika práce

Vývoj hardvéru a softvéru určeného pre náš systém bol komplexný proces ktorý sme rozdelili na niekoľko úkonov a otázok ktoré sme sa počas vývoja pýtali:

Najskôr sme si musíme definovať pre koho systém je – pre profesionálov ako aj amatérov, teda jeho dizajn a ovládanie musí byť intuitívne a jednoduché ale zároveň plnohodnotné.

Ďalší v poradí bol výber uhlopriečky zobrazovacieho panela – displeja od firmy Nextion, okolo ktorého je postavená zobrazovacia časť prijímacieho modulu. Zvolili sme ideálnu uhlopriečku – 5“ ktorá je dostatočne veľká aj na komplexnú grafiku a zároveň dostatočne kompaktná aby nezaťažovala užívateľa svojimi rozmermi a váhou.

Ku zvolenému displeju bolo nutné vybrať mikrokontrolér a komplementárnu elektroniku – Tu je dôležité napájanie z univerzálneho modulového slotu, komunikácia s „lietajúcim“ modulom, vstup užívateľa cez dotykový displej a celková univerzálnosť zobrazovacieho modulu – možnosť pripojiť ho na tablet ako aj položiť na stôl prípadne chytiť do ruky. „Lietajúci“ modul má priority úplne odlišné – musí byť ľahký, kompaktný, s nízkou spotrebou a vysokou odolnosťou voči nárazom, tekutinám a prachu.

Pre celý systém je dôležitý dlhý dosah a odolnosť voči externým aj interným interferenciám.

Celá práca je rozdelená na dva úseky pojednávajúce o vývoji softvéru a hardvéru. Tieto úseky sa kvôli prehľadnosti ďalej delia na kapitoly bližšie približujúce danú tematiku.

Z dôvodu obmedzenia rozsahu práce na 25 strán sme boli nútený jej veľkú časť presunúť do príloh. Pre plné pochopenie konceptu fungovania projektu ako celku ako aj vývoju softvéru preň odporúčame naštudovanie príloh A až I.

Kapitola 1

Hardvér tretej verzie - časť pilota

Hardvér časti pilota systému DroneLoger3 sa zkladá z mnohých častí, rozdelíme si ho teda na sekcie:

- Plastové diely častí na a v RC súprave
- Elektronika častí na a v RC súprave
- Dizajn DPS

1.1 Plastové diely častí na a v RC súprave

1.1.1 Plastové diely zobrazovacieho panelu

Celý vývojový proces plastových dielov zobrazovacej časti sa musel podriadiť dvom veciam – Vybranému zobrazovaciemu panelu od firmy Nextion [2] a nutnosti kompatibility s univerzálnym JR modulom nachádzajúcim sa v zadnej časti väčšiny RC súprav na trhu.

Výsledný dizajn krabičky zobrazovacieho zariadenia je ľahký (minimum materiálu na tlač), kompaktný, dostatočne pevný, ponúka množstvo možností uchytenia (dizajn „spôn“ na pripnutie na tablet) a je esteticky príjemný.

Skladá sa z troch, resp. štyroch častí – čelného krytu displeja ktorý samotný displej chráni a poskytuje miesto na logo, samotného displeja, zadného/stredného krytu displeja ktorý displej tak tiež chráni a zadného výstupného/napájacieho ktorý obsahuje USB konektor na napájanie a komunikáciu s modulom vo vysielači pilota. Všetky 3 diely sú spojené metrickými skrutkami M3 10mm. Proces 3D návrhu prebehol v softvéri Fusion360, neskôr boli diely vytlačené z ABS¹ plastu na 3D tlačiarni. ABS plast bol zvolený z viacerých dôvodov – je lacný, ľahký, odolný voči teplu ($T_g = 105C$) a pevný v ťahu aj v ohybe (UTS = 6600 PSI; FS = 10800 PSI).



Obr. 1.1: Zobrazovací modul DroneLogger Air

1.1.2 Plastové diely modulu v RC súprave

Počas návrhového a vývojového procesu bolo nutné zohľadniť niekoľko dôležitých faktov: Modul obsahujúci elektroniku musí byť kompatibilný s JR modulovým slotom ktorým disponuje väčšina RC súprav na trhu a modul nesmie prekážať v komfortnom využívaní RC súpravy svojimi rozmermi či umiestnením externej antény.



Obr. 1.2: Modul vo vysielačke FrSky Taranis

¹Akrylonitrilbutadiénstyrén

Výsledný dizajn spĺňa všetky tieto parametre a skladá sa zo štyroch časťí:

- Vrchného krytu ktorý zabezpečuje ochranu a chladenie elektroniky
- Samotnej elektroniky
- Spodného krytu pripojiteľného do modulu JR
- Antény pripojenej k internému RF modulu



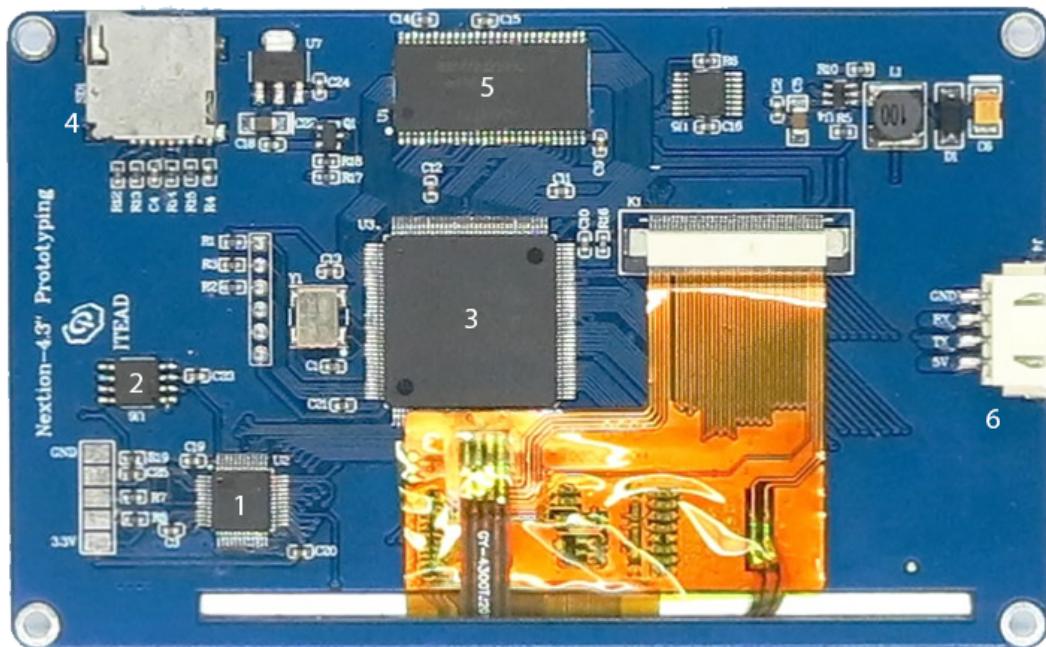
Obr. 1.3: Časti modulu

Vrchný kryt obsahuje perforované bočné steny umožňujúci lepšie chladenie elektroniky a otvory na konektor USB typu A a konektor typu SMA na anténu interného RF modulu. Spodný kryt obsahuje otvor pre hrebienkový konektor ktorý umožňuje napájanie modulu a komunikáciu s RC súpravou a slot na SD kartu slúžiacu na záznam letových údajov. Návrh všetkých častí prebehol v 3D CAD softvéri Fusion360, diely boli následne vytlačené na 3D tlačiarni z ABS¹ plastu z rovnakých dôvodov ako pri zobrazovacom paneli.

1.2 Elektronika častí na a v RC súprave

1.2.1 Elektronika zobrazovacieho modulu

Zobrazovací modul je primárne plastový, jedinými elektronickými súčiastkami sú DPS zabezpečujúca komunikáciu cez USB s modulom vo vysielači a HMI riešenie NX4827T043 od firmy Nextion. Je to LCD panel o uhlopriečke 4.3", rozlíšení 480 x 272 obrazových bodov podporujúci 65K farieb s rezistívou dotykovou vrstvou. Ked'že sa jedná o HMI, panel obsahuje aj 16MB integrovanej Flash pamäte (pre vykreslovanú grafiku), riadiaci mikrokontrolér (STM32 [3]) a grafický procesor (ALTERA MAX II FPGA [4]) doplnený o RGB buffer v podobe RAM pamäte. Vývojové WYSIWYG prostredie je dodávané výrobcom, softvér obsluhujúci panel je nezávislý od pripojeného mikrokontroléra (v našom prípade Atmega328pb) s ktorým komunikuje pomocou obojsmernej UART zbernice. Jej primárny účel je zadávanie obslužných príkazov displeju, teda mu "hovorí" čo zobraziť. Táto zbernice taktiež odovzdáva informácie o stlačení dotykovej vrstvy displeja riadiacemu mikrokontroléru. Absolútne nezávislosť má niekoľko výhod - softvér displeja bežiaci z SD karty v integrovanom slote je veľmi ľahko inovateľný bez nutnosti zásahov do prostredia riadiaceho mikrokontroléra. Toto značne znižuje výkonové požiadavky na externú ovládaci elektroniku.



Obr. 1.4: 1)STM32 MCU, 2)16M FLASH, 3)FPGA chip, 4)Micro-SD slot, 5)RGB Buffer, 6)UART int.

1.2.2 Elektronika modulu RC súpravy

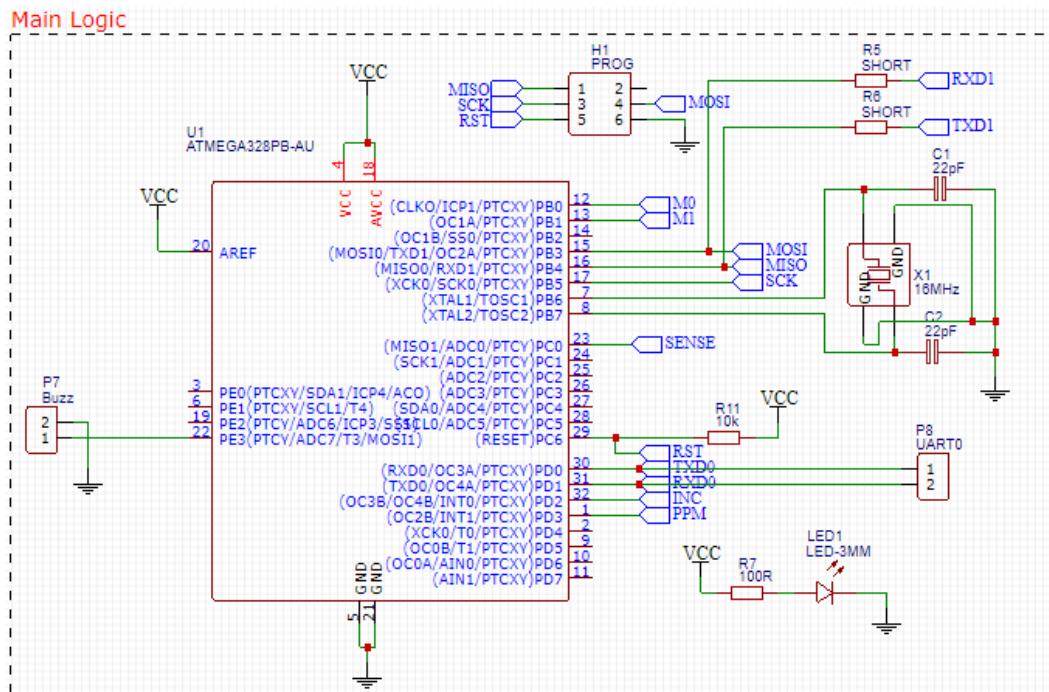
Všetky schémy a DPS sme navrhovali v cloudovom CAD softvéri EasyEda z dôvodu jeho dostupnosti kdekoľvek, širokej komunite používateľov a jednoduchosti následného objednania DPS. Z dôvodu prehľadnosti si túto sekciu rozdelíme na niekoľko častí:

- Schéma hlavnej logiky
- Schéma bezdrôtovej komunikácie
- Schéma napájanie a VS
- Dizajn DPS

Celú schému je možné nájsť v prílohoch (A)

1.2.3 Schéma hlavnej logiky

V tejto časti sa zameriame na hlavnú časť celkovej schémy modulu RC súpravy - schému hlavnej logiky:



Obr. 1.5: Schéma hlavnej logiky

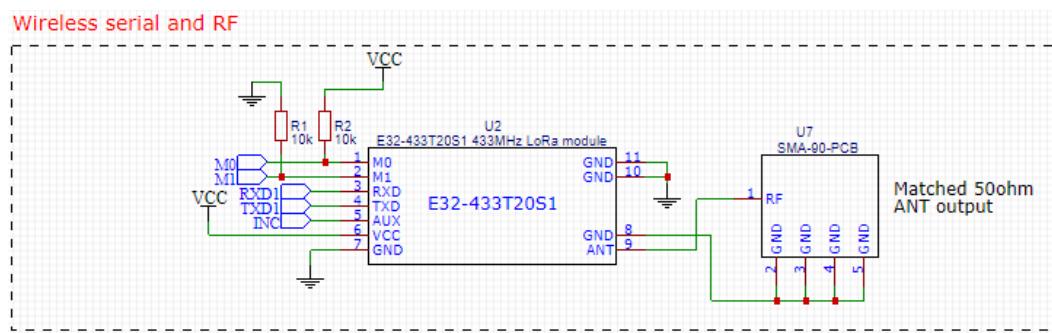
Tu je jedna z najdôležitejších súčasti celého modulu - mikrokontrolér Atmega328pb[5].

Je to jednoduchý a všeobecný mikrokontrolér spoločnosti Atmel (Microchip po akvizícii) AVR architektúry. O jeho všeobecnosť sa stará vysoký počet periférií.

Taktiež pre našu aplikáciu dôležitá súčasť je intergrovaný 16bit DAC. Zvolili sme modernizovanú verziu Atmega328pb ktorá má dve hardvérové uart zbernice nutné na paralelnú komunikáciu s HMI riešením a bezdrôtovým sériovým modulom. Ďaľej na schéme môžeme vidieť konektor H1 slúžiaci na programovanie pomocou SPI zbernice (ISP), ledku signalizujúcu funkčné napájanie napájanie (LED1), konektor P8 umožňujúci programovanie jednoduchým USB to UART mostíkom po vypálení zavádzacieho programu a externý 16Mhz oscilátor. Taktiež vydíme konektor na piezoelektrický bzučiak na doplnenie vizuálnych informácií z LCD panela audiom a možnosť prerušíť sériové linky zbernice UART1 smerujúce do bezdrôtového modulu. Táto časť schémy má niekoľko funkcií, medzi hlavné patria komunikácia s bezdrôtovým modulom, spracovanie komunikačného packetu, odovzdanie inštrukcií HMI riešeniu, ovládanie piezoelektrického bzučiaka a čítanie vstupného napájacieho napäťa pomocou DAC.

1.2.4 Schéma bezdrôtovej komunikácie

Tento úsek hlavnej schémy obsahuje hlavne bezdrôtový sériový modul zabezpečujúci komunikáciu medzi modulmi v RC súprave a na lietajúcom stroji:

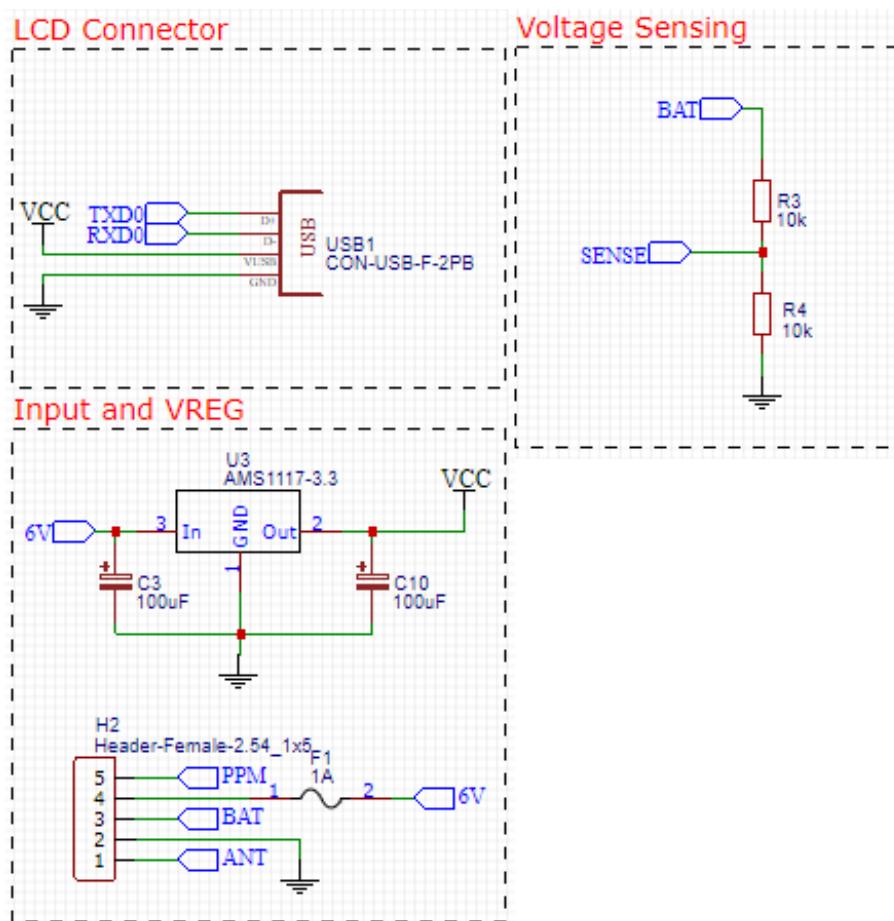


Obr. 1.6: Schéma bezdrôtovej komunikácie

Tejto časti schémy dominuje bezdrôtový sériový modul E32-433T20S1[6]. Je založený na čipe SX1278[7] spoločnosti Semtech. Kľúčove vlastnosti tohto čipu, teda aj nami vybraného modulu sú podpora LoRa™ FHSS Spread-Spectrum modulácie, -148dBm vstupná citlosť a +20dBm vysielací výkon v pásme 433Mhz s veľmi nízkou spotrebou. Okrem horeuvedených vlastností tento čip disponuje až 19.2kbps (0.3kbps @ -148dBm) bezdrôtovým bitratom. Všetky tieto vlastnosti spolu s až 256B CRC zabezpečujú konštatné výsledky aj vo vysoko zarušenom prostredí. V testoch[8] dosahuje modul až 3000m komunikačnú vzdialenosť v realistických podmienkach. K modulu je možné pripojiť anténu pomocou SMA konektora.

1.2.5 Schéma napájanie a VS

Tu sa zameriame na napájanie, konektor zabezpečujúci komunikáciu s HMI riešením, pripojenie ku RC súprave a čítanie napäťa na vstupe napájania:



Obr. 1.7: Schéma napájania a VS

V poslednom úseku hlavnej schémy je dominantou napájanie - Lineárny regulátor AMS1117-3.3[9] dostupný od rôznych výrobcov. Bol zvolený z niekoľkých dôvodov - jednoduchosť, teda ja cena zapojenia a nízky výkon celého modulu - nízke ztraty aj pri lineárnom napájaní. Ďalšími súčasťami sú konektor na komunikáciu s HMI riešením - zvolili sme USB typu A. Toto riešenie nám umožňuje spojiť modul v RC súprave so zobrazovacím modulom aj na dlhšie vzdialenosť a to pomocou bežne dostupných USB káblov. Spojenie s RC súpravou je realizované pomocou hrebienkového konektora s rastrom 2.54mm a čítanie napäťacieho napäťa pomocou napäťového deliča v pomere $U_{výs} = \frac{1}{2} U_{vs}$.

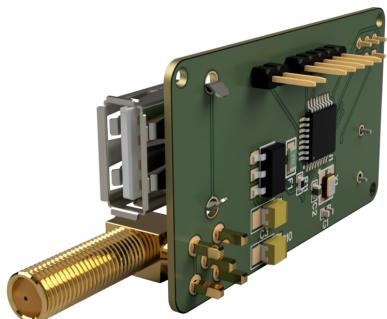
Celú schému je možné nájsť v prílohoch.

1.3 Dizajn DPS

Dizajn plošného spoja prebehol taktiež v clouдовom nástroji EasyEda. O ich následné vyhotovenie sa postarala Čínska firma JLCPCB. Výsledný plošný spoj je obdĺžnikového tvaru o rozmeroch 55mm*31mm a štandardnej hrúbky 1.6mm. Samotný laminát je typu FR4 ($T_g = 140C$).

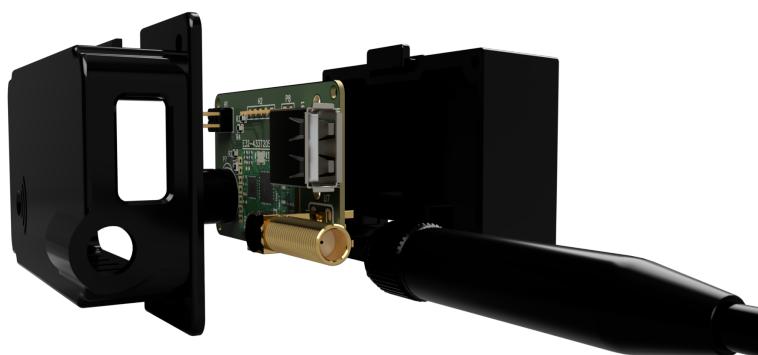


Obr. 1.8: Spodná vrstva DPS



Obr. 1.9: Vrchná vrstva DPS

DPS musí mať v tomto prípade jednu hlavnú vlastnosť - výjsť do formátu JR modulu. Taktiež je dôležité správne rozmiestnenie konektorov - SMA a USB konektory je nutné umiestniť na vrchnej (kratšej) strane z dôvodu polohy modulu v RC súprave - smer antény a pripojeného USB káblu by mal byť totožný s integrovanou anténou RC súpravy. Ideálna pozícia bzučiaku je taktiež smerom hore (na hornej vrstve) - od tela RC súpravy. Hrebienkové konektory napájania a sériovej zbernice UART1 musia smerovať dolu - do tela RC súpravy. Ideálna pozícia bezdrôtového modulu je na vrchnej vrstve z termálnych dôvodov, z rovnakých dôvodov sme umiestnili zvyšok citlivej elektroniky na vrstvu opačnú.



Obr. 1.10: Osadenie DPS v krabičke

Kapitola 2

Hardvér tretej verzie - časť bezpilotného stroja

V tejto kapitole sa pozrieme na návrh a fungovanie modulu bezpilotného stroja teda na dizajn jeho plastových dielov, schémy a objasníme dôvody zvolenia konkrétnych komponentov resp. technologických riešení. Pre sprehľadnenie si kapitolu opäť rozdelíme na niekoľko sekcií:

- Dizajn a konštrukčné riešenie plastových dielov
- Schéma zapojenia, odôvodnenie výberu súčiastok
- Dizajn DPS

2.1 Dizajn a konštrukčné riešenie plastových dielov

Správne vyhotovenie a dizajn dielov krabičky je pre fungovanie systému podľa špecifikácií nanajvýš dôležité. Tieto plastové diely musia byť extrémne odolné a zároveň presné. Musia elektroniku umiestnenú vo vnútri chrániť pred prachom, tekutinami a otrasmami ktoré sa vyskytujú počas bežnej prevádzky bezpilotného stroja aj počas havarijných situácií. Zároveň je nutné udržať váhu celého modulu na minimum z dôvodu nutnosti zachovania využitia bezpilotného stroja v prípade umiestnenia na krídlo/rameno.



Obr. 2.1: Plastové diely a elektronika v nich

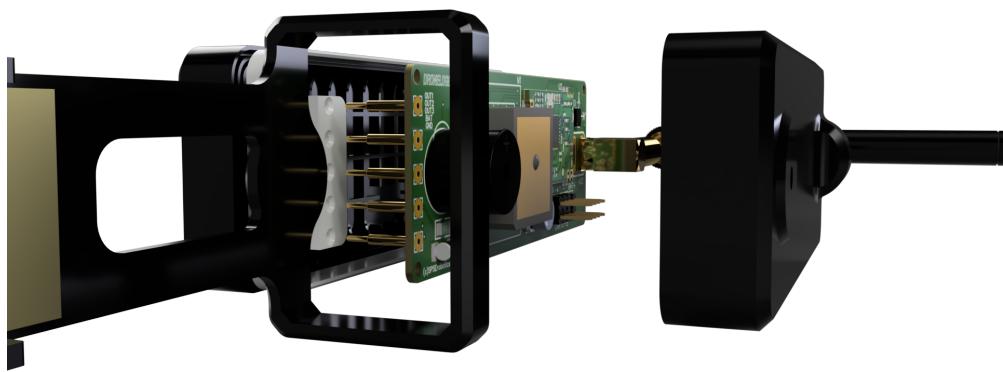
2.1.1 Horný a spodný diel krabičky

Dizajn dielov krabičky je podrobený špecifickým požiadavkám prostredia, v ktorom sa bude nachádzať. Vrchný a spodný diel sú spojené pomocou oceľových skrutiek typu M3 o dĺžke 10mm. Je nutné, aby bola krabička tesná aj voči vode pod tlakom, nielen vode kvapkajúcej. Z toho dôvodu sme sa rozhodli utesniť všetky otvory pomocou 3D tlačených tesniacich krúžkov z materiálu TPU² - je relatívne jednoduchý na tlač a je vysoko ohybný. Tento materiál sa bežne používa na výrobu ohybných puzdier na smart telefóny.

V našom prípade bol použitý na výrobu troch tesnení - jednej tesniacej membrány ktorá zabraňuje vstupu neželaných častíc do priestoru piezoelektrického bzučiaka a tesniaceho valca ktorý vytvára tlak po obvode barometra a zabraňuje prieniku neželaných častíc do priestoru elektroniky. Tvar tohto tesnenia je prispôsobený tak, aby umožnil barometru prístup ku okolitému prostrediu. Samotný barometer je vodeodolný.

Tretím tesnením je krúžok, resp. obdlžník po obvode spodného dielu krabičky. Tento prvak zabezpečuje vodotesný a prachotesný spoj medzi vrchným a spodným dielom krabičky. Toto tesnenie nie je samostatným prvkom, je integrované (natavené, "natlačené") do spodného dielu krabičky. Tento spôsob zaručuje jeho absolútne priľnutie k ABS¹ plastu z ktorého je samotná krabička bez použitia špeciálnych adhezív na TPU² plast.

²Termoplastický Polyuretán



Obr. 2.2: Plastové diely a elektronika v nich

Posledným dielom z TPU² plastu je tesniaci blok obalujúci tzv. pogo piny slúžiace na komunikáciu s bezpilotným strojom. Tento atypický spôsob prepojenia bol zvolený z dôvodu jeho vysokej odolnosti oproti bežnému konektoru - v prípade tesnosti voči vode a prachu aj odolnosti proti nárazom a prudkým pohybom - v prípade nárazu dôjde k rozpojeniu kontaktu nedeštruktívne.

Špecifickým prvkom prvkom spodného dielu krabičky je zasadenie v mieste pogo kontaktov - tie nepresahujú za úroveň spodnej steny krabičky je teda malá šanca ich zachytenia priu manipulácii a poškodenia plošného spoja a kontaktov samotných. Vrchný diel obsahuje tiež mosadzné závitové termovložky typu M3. Pri dizajne vnútornej štruktúry krabičky bolo nutné myslieť na jej odolnosť voči zvýšenému tlaku zvonka (napr. prejdenie modlu autom) teda bola navrhnutá štruktúra 0.8mm hrubých výstužných stien ktorá je prispôsobená tvarom a umiestneniu jednotlivých komponentov na plošnom spoji - pri uzavretí krabičky dochádza ku "spojeniu" vrchného a spodného dielu bez medzere cez plošný spoj - pri zvýšenom tlaku zvonka nedochádza k ohýbaniu a praskaniu.



Obr. 2.3: Plastové diely a elektronika v nich

Posledným dielom modulu lietajúceho stroja je obruč držiaca modul na ramene stroja. Táto obruč tiež drží plošný spoj vytvárajúci kontakt s pogo pinmi zabezpečujúci priame prepojenie napájacieho napäťa a komunikácie s lietajúcim strojom. Výsledný dizajn krabičky je kompaktný, ľahký a extrémne odolný voči poveternostným vplyvom.

2.2 Schéma zapojenia, odôvodnenie výberu súčiastok

Pred výberom konkrétnych komponentov si musíme stanoviť čoho má byť "vysielač" modul schopný:

- Spoľahlivo komunikovať "so zemou" aj na dlhšie vzdialenosť
- Zbierať dátá o polohe v priestore pomocou viacerých snezorov
- Snímať okolitý tlak a teplotu
- Systém musí byť schopný samostatného fungovania (interné napájanie)
- Systém musí byť taktiež vysoko odolný a veľmi ľahký

Komunikácia medzi "zemou" a lietajúcim strojom je zabezpečená LoRa modulom E32-433T20S1[6] a rovnako ako v module vo vysielačke bol zvolený z dôvodov nízkej spotreby a výbornej vstupnej a výstupnej citlivosti a jeho vysielačieho výkonu. Zbieranie dát o polohe v priestore je vykonávané prostredníctvom GPS modulu Quectel L86 [10]. Tento modul bol zvolený z niekoľkých dôvodov - je extrémne malý, má vysokú citlivosť a integruje GNSS chipset s panelovou keramickou anténou v

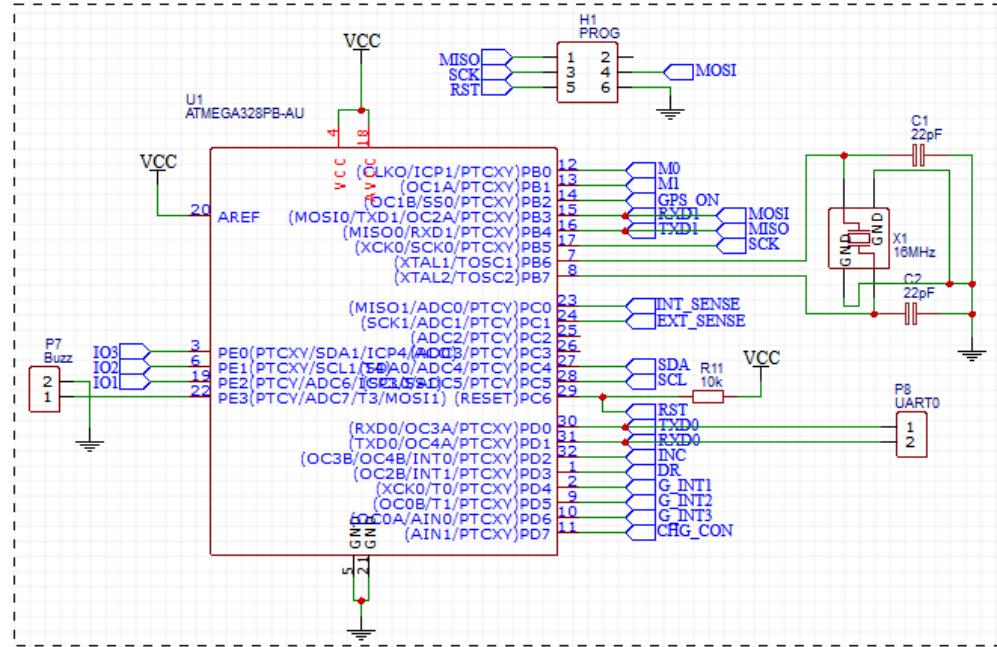
jednom “balení”. Dáta o barometrickom tlaku okolia, teda o vertikálnom umiestnení modulu zbiera altimeter HP206C[11] čínskej firmy HOPERF. Je kompaktný, vysoko presný (rozlíšenie až 0.1m) a odolný voči poveternostným vplyvom. Na zbieranie dát o akcelerácii, náklone a magnetickom náklone bol vybratý Inerciálny modul iNEMO LSM9DS1[3] od Švajčiarskej spoločnosti ST Microelectronics. Jeho kompaktné rozmery, vysoká miera integrácie, presnosti a možnosti interného škálovania meraných jednotiek bol vybraný ako jeden z hlavných článkov zberu dát pre modul lietajúceho stroja. Tento systém taktiež disponuje interným napájaním článkom Li-Ion o kapacite 1.5Wh s integrovanou elektronikou na správu a nabíjanie. Vysokú odolnosť modulu zabezpečuje okrem dizajnu krabičky aj vhodný návrh plošného spoja.

Pre zprehľadnenie si schému rozedelíme na 7 úsekov:

- Úsek mikrokontroléra
- Úsek bezdrôtovej komunikácie
- Úsek snímania pozície
- Úsek vstupov a ochrany
- Úsek správy interného akumulátora
- Úsek regulácie napäťia a
- Úsek filtrovania

2.2.1 Úsek mikrokontroléra

Hlavnou časťou tohto úseku je riadiaci mikrokontrolér Atmega328pb[12]:

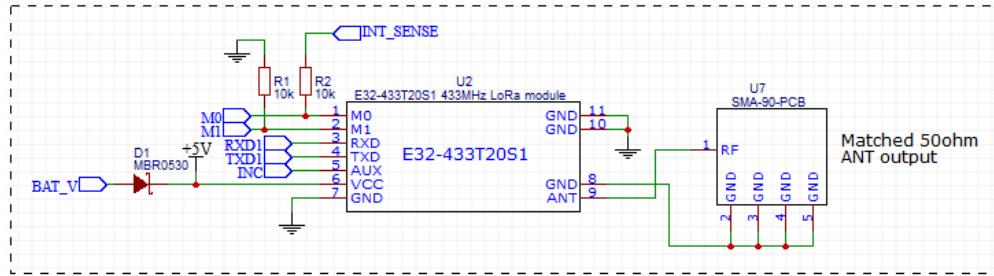


Obr. 2.4: úsek schémy s riadiacim mikrokon.

Tento integrovaný obvod Amerického výrobcu je všestranný a jednoduchý, založený na AVR architektúre. Jeho veľké množstvo periférií v podobe ADC a hardvérom akcelerovaných sériových zbernic spolu s veľmi jednoduchým “minimálnym zapojením” z neho robí ideálneho kandidáta pre tento modul. V tejto časti schémy môžeme taktiež vidieť externý oscilátor s frekvenciou $F_{osc} = 16MHz$ a programovací konektor H1 ATMEL ISP 6P. Tento úsek je srdcom celého modulu - spracúva dátu zo senzorov, GPS a sériového modulu a tieto spracované dátu odosiela vo forme dvoch periodických paketov na “zem”.

2.2.2 Úsek bezdrôtovej komunikácie

V tomto úseku je dominantný LoRa RF modul E32-433T20S1[6]. Tu je okrem už spomenutých kľúčových vlastností tohto modulu - kom. čipu SX1278[12], vysokej vstupnej a výstupnej citlivosti, vysokého výkonu a dosahu v pásme 433mhz vďaka pokročilím algoritmom a dostatočným bezdrôtovým bitratom dôležitá možnosť implementovať WOR (Wake On Receive) - je teda možné vynechať zo zapojenia modulu lietajúceho stroja tlačítko ON/OFF a modul aktivovať a deaktivovať na diaľku v prípade potreby.

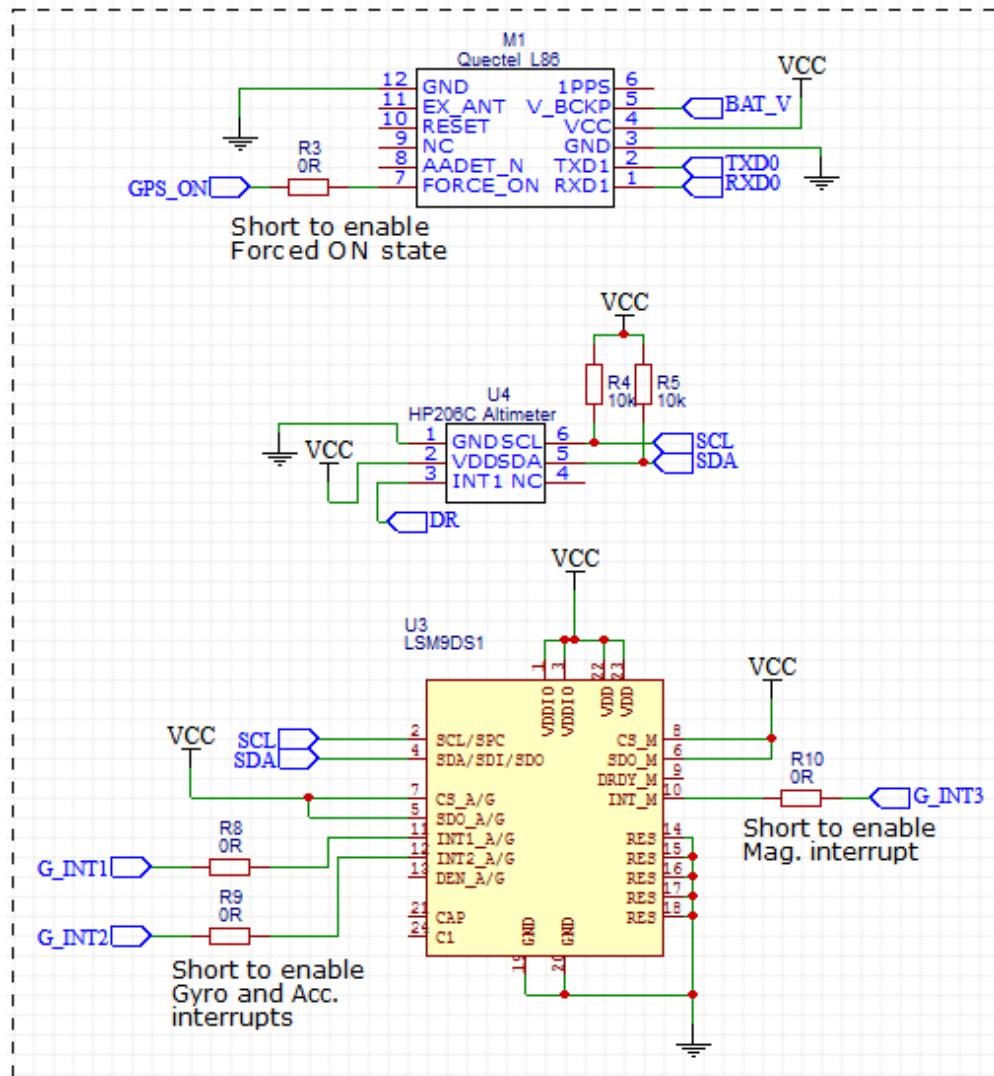


Obr. 2.5: úsek schémy RF modulom

Okrem bezdrôtového komunikačného modulu sa tu nachádza tiež SMA RF konektor na pripojenie antény.

2.2.3 Úsek snímania pozície

Jedna z najdôležitejších častí modulu lieatjúceho stroja je úsek snímania polohy:



Obr. 2.6: úsek snímania polohy

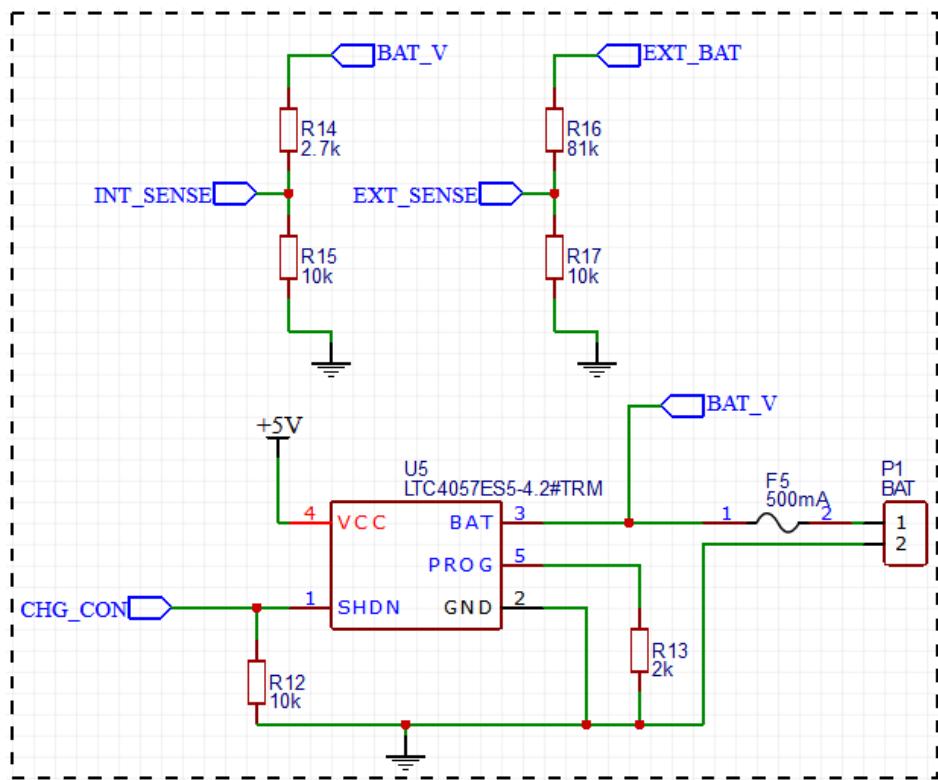
Tu sa nachádzajú dva senzory a GPS moudul Quectel L86[10]. Tento modul má niekoľko priaznivých vlastností ktoré ho predurčujú ako vhodného kandidáta pre náš projekt - modul L86 je vysoko kompaktný GPS modul konfigurácie POT (Patch On Top) s integrovanou anténou využívajúci moderný GNSS SOC MT3333[13] firmy MediaTek čo ho predurčuje na využitie v miniatúrnych zariadenia kde je vyžadovaná nízka váha a rozmery a vysoká pozičná presnosť. Tento GNSS modul taktiež umožnuje napájanie RTC z jednočlánkovej batérie čo znížuje vybíjací prúd integrovanej záložnej batérie modulu v režime spánku a zrýchľuje získanie pozície pri opäťovnom zapnutí.

Dôležitou súčasťou modulu ktorá sa nachádza na tomto úseku schémy je inerciálny modul od firmy STMicroelectronics LSM9DS1[3]. Modul LSM9DS1 nie je obyčajný senzor polohy ale systém troch 3D digitálnych senzorov: lineárneho senzoru akcelerácie, senzoru náklonu a senzoru magnetického poľa. Tento modul umožnuje nastaviť škálovanie na všetkých senzoroch čo umožnuje čítať dané veličiny v maximálnom možnom rozlíšení. Tiež podporuje programovateľné prerušenia všetkých senzorov a vďaka pokročilej internej logike sú výstupy precízne korigované a filtrované, teda výstupné dátá sú pripravené na použitie bez výkonovo náročného filtrovania v hlavnom mikrokontroléri.

Posledným aktívnym komponentom v tejto časti schémy je altimeter HP206C[11]. Tento modul bol zvolený z niekoľkých dôvodov: jeho integrovaný altimeter je vysoko presný ($\pm 5\text{cm}$), má malé rozmery a vďaka nepriepustnej membráne je vysoko odolný voči poveternostným vplyvom.

2.2.4 Úsek správy interného akumulátora

V tomto úseku sa nachádza elektronika slúžiaca na nabíjanie a čítanie napäťa int. akumulátora:



Obr. 2.7: úsek nabíjania int. akumulátora

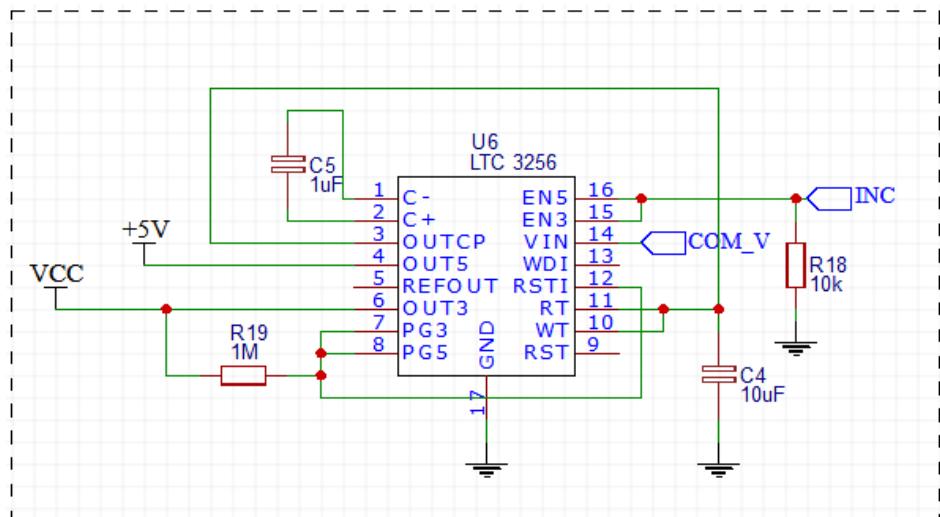
Tu je prominentným integrovaný obvod LTC4057-4.2[14] spoločnosti Linear Technology. Je to jednoduchý ale veľmi schopný obvod slúžiaci na nabíjanie článkov technológie Li-Ion. Bol vybraný vďaka jeho jednoduchému zapojeniu, kompaktným rozmerom, širokému rozsahu vstupného napäťa a nastaviteľnému prúdu až do 800mA jedným rezistorom. Samozrejmostou je poistka chrániaca obvod v prípade skratu Li-Ion článku.

Tiež sa tu nachádzajú dva napäťové deliče umožňujúce čítať napätie externého aj interného akumulátora.

2.2.5 Úsek regulácie napäťa

V tomto úseku je riešený problém nutnosti dvoch napájacích vetiev (3.3V a 5V) a ich regulácie zo značne vyššieho napäťa (až 25V). Je nutné, aby naše regulačné zapojenie pozostávalo z malého počtu komponentov a malo nízke výstupné zvlnenie. Ideálnym zapojením by bol pulzný regulátor s lineárny výstupným regulátorom. Presne toto

zapojenie do seba integruje obvod od Linear Technology LTC3256[15]:



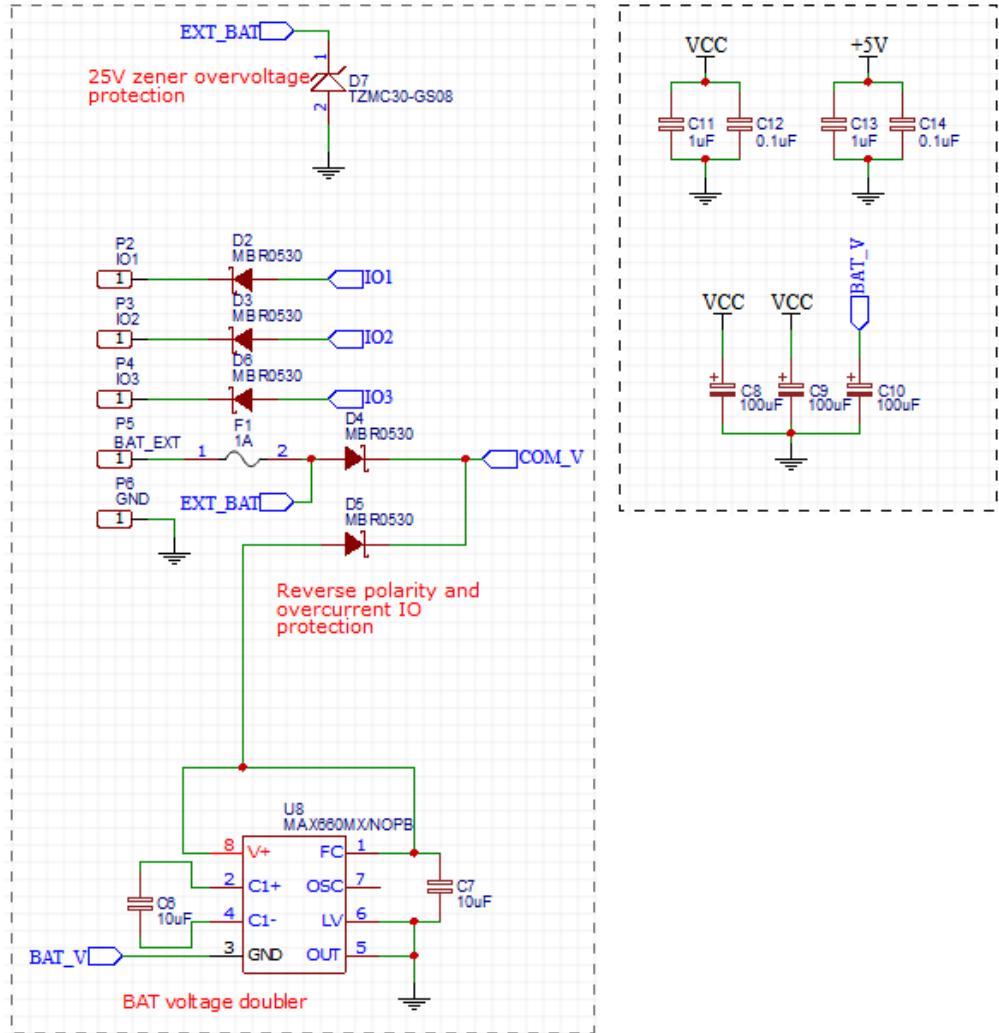
Obr. 2.8: úsek regulácie napäťia

Tento integrovaný obvod obsahuje vstupný pulzný DC/DC regulátor poskytujúci 5V výstup ktorý je dostupný na 5V vetve a tiež ďalej regulovaný na 3.3V. Toto zapojenie umožňuje vysoké vstupné napätie zároveň s nízkym stratovým teplom. Výstupné prúdy obvodu sú 100mA na 5V vetve a 250mA na vetve s napäťím 3.3V čo plne postačuje na beh modulu bezpilotného stroja. Ďaľšou nutnou vlastnosťou je možnosť obvod jednoducho zobudiť a uspať externým mikrokontrolérom pričom jeho spánkový prúd je len $0.5\mu A$.

2.2.6 Úsek vstupov / výstupov a úsek filtrovania

Úsek filtrovania obsahuje dva druhy zapojení kondenzátorov: oddeľovacie zabezpečujúce odstránenie šumu vzniknutého v regulátore napäťia a kondenzátory vyššej kapacity zabezpečujúce stabilitu v prípade výkyvov prúdov na danej napäťovej vetve (RF modul začne vysielať, interný akumulátor sa začne nabíjať)

V úseku vstupov / výstupov je prominentný integrovaný obvod MAX660MX[16] zabezpečujúci jednoduchým zdvojením vstupného napäťia vstup do už spomenutého regulátora napäťia. Tento úsek schémy ďaľej zabezpečuje ochranu pred prekročením maximálneho vstupného napäťia (25V), diódy zabezpečujúce uzavorenie vybíjania interného akumulátora v prípade pripojenia externého napäťia, obsahuje Polymérovú poistku chrániacu celé zapojenie a diódy chrániace výstupy pred pripojením externého napäťia.

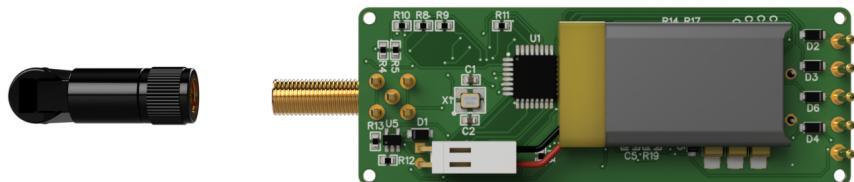


Obr. 2.9: úsek filtrovania

Celú schému je možné nájsť v prílohách.

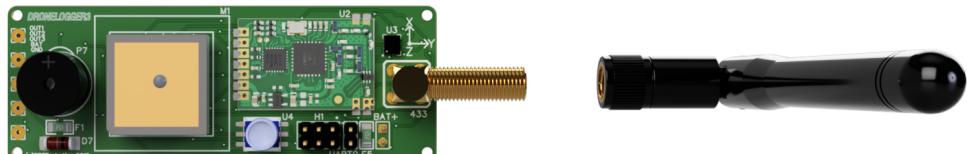
2.3 Dizajn DPS

Je nutné, aby mal plošný spoj vhodné rozmery na bezproblémové uchytenie na krídle lietajúceho stroja. Taktiež je nutné myslieť na konektor na anténu a "pogo-piny" na komunikáciu s externými zariadeniami a napájanie samotného modulu.



Obr. 2.10: pohľad "zospodu"

Na spodnej časti plošného spoja môžeme vidieť hlavný riadiaci mikrokontrolér, integrovaný obvod zabezpečujúci nabíjanie interného akumulátora, diódy chrániace vstupy a výstupy a samotný interný akumulátor. Tu je nutné brať do úvahy interný akumulátor a jeho pozíciu a taktiež "pogo-piny" slúžiace na komunikáciu a napájanie.



Obr. 2.11: pohľad "zhora"

Pri pohľade zhora môžeme vidieť GPS modul, bzučiak, bezdrôtový sériový modul, inerciálny modul a altimeter. Tu je nutné správne umiestniť bzučiak a konektor typu SMA na externú anténu.

Pri návrhu celého plošného spoja bolo nutné myslieť aj na výrobu krabičky, tesnení a praktického umiestnenia na bezpilotnom stroji.

Závery práce

Počas našich dlhoročných skúseností s bezpilotnými strojmi a RC modelmi vysvitla potreba oddeleného, absolútne autonómneho navigačného a záznamového systému disponujúceho vysokou versatilitou, spoľahlivosťou a nízkou váhou, rozmermi a cenou.

Výsledkom niekoľkoročného vývoja systému spĺňajúceho horeuvedené kritériá je DroneLogger3 - kompaktný a versatilný navigačný systém spĺňajúci požiadavky ako začiatočníkov vyžadujúcich jednoduchosť pri inštalácii a používaní tak aj pokročilých pilotov očakávajúcich absolútну spoľahlivosť a presnosť.

Výsledkom tejto práce je dokument po hardvérovej stránke popisujúci fungovanie systému DroneLogger3 - ako oba moduly vzájomne aj samostatne fungujú a zdôvodnenie výberu nielen elektronických súčiastok počas ich vývoja, pôvod systému - prečo sa vývoj uberal týmto smerom a z čoho sme počas vývoja vychádzali a výsledný vzhľad a užívateľská prívetivosť systému - kompatibilita so širokým spektrom RC súprav a bezpilotných strojov.

Resumé

In this thesis we have described the needs comming from our years long experience with unmanned aircraft and RC models on which the developement and feature set of this assistnace and datalogging system is based on.

We have also described the developement of the system from both hardware and software perspective - selected materials, manufacturing processes, electrical components and design choices leading to the final feature set and user experience of the DroneLogger3.

To furhter explain the developement path and hardware choices leading to a polished and fully functional assistance and datalogging system ready for end user usage we have also looked into the previous versions of the system and its hardware and software.

Zoznam použitej literatúry

1. BBC. *Heathrow airport: Drone sighting halts departures*. Dostupné tiež z: <https://www.bbc.com/news/uk-46803713>.
2. NEXTION. *NX4827T043 Datasheet*. Dostupné tiež z: https://nextion.itead.cc/resources/datasheets/nx4827t043_011/.
3. ST. *STM32 MCUs*. Dostupné tiež z: <https://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html>.
4. INTEL. *ALTERA MAX II Datasheet*. Dostupné tiež z: <https://www.intel.com/content/www/us/en/products/programmable/cpld/max-ii.html>.
5. ATMEL. *Atmega328 Datasheet*. Dostupné tiež z: <https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf>.
6. EBYTE. *E32 Series Datasheet*. Dostupné tiež z: <http://www.ebyte.com/en/downpdf.aspx?id=229>.
7. SEMTECH. *SX1278 Datasheet*. Dostupné tiež z: https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf.
8. EBYTE. *Test vzdialenosťi*. Dostupné tiež z: <http://www.ebyte.com/en/product-view-news.aspx?id=229>.
9. MONOLITHIC. *AMS1117 Datasheet*. Dostupné tiež z: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>.
10. QUECTEL. *Quectel L86 Datasheet*. Dostupné tiež z: https://www.quectel.com/UploadFile/Product/Quectel_L86_GNSS_Specification_V1.0.pdf.
11. HOPERF. *HP206C Datasheet*. Dostupné tiež z: <https://cdn.sos.sk/productdata/d8/cc/27577aa5/hp206c.pdf>.
12. ILITEK. *ILI9341 Datasheet*. Dostupné tiež z: <https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf>.

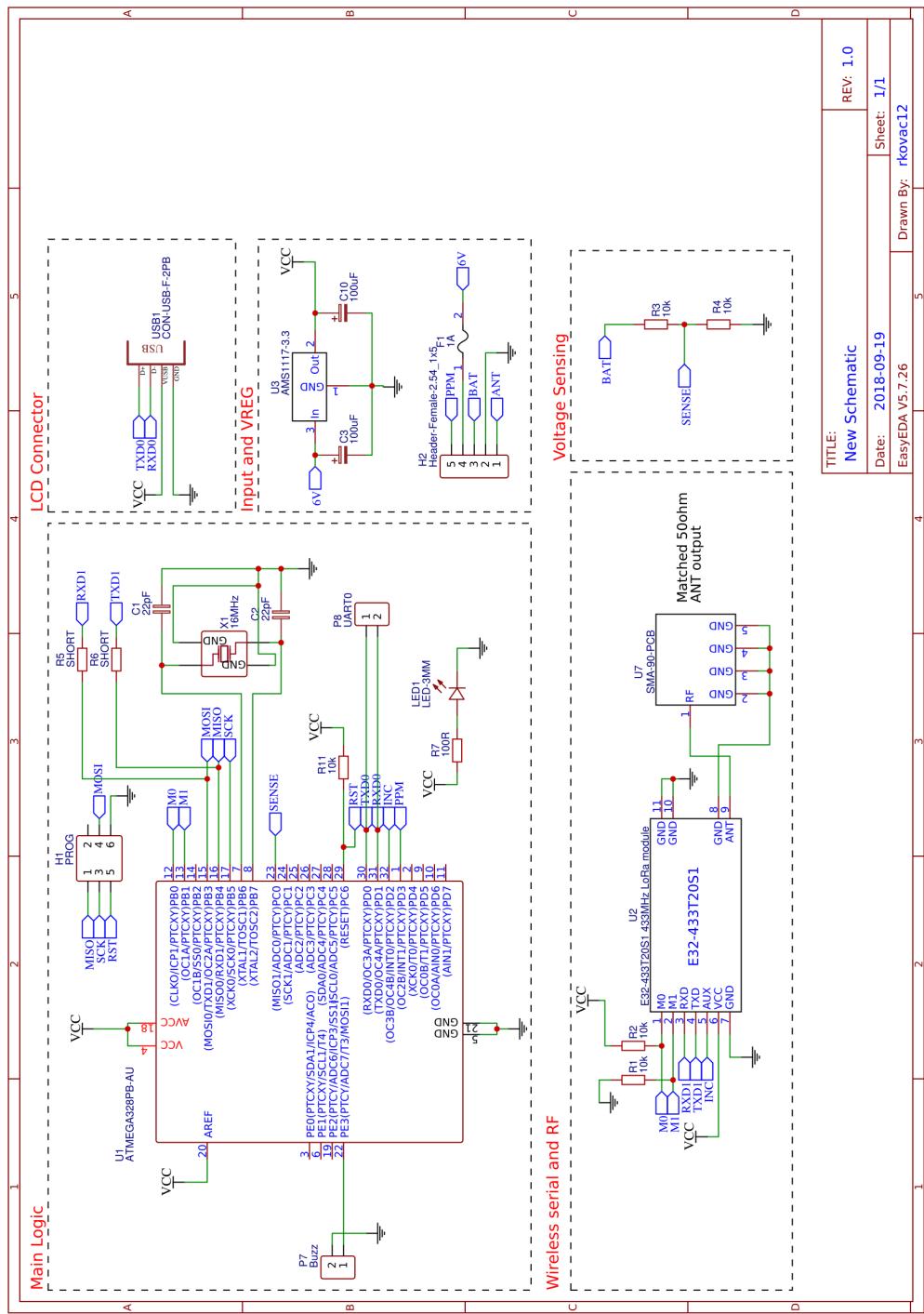
13. MEDIATEK. *MT3333 Datasheet*. Dostupné tiež z: <https://labs.mediatek.com/en/chipset/MT3333>.
14. TECHNOLOGY, Linear. *LTC4057-4.2 Datasheet*. Dostupné tiež z: <https://www.analog.com/media/en/technical-documentation/data-sheets/4057f.pdf>.
15. TECHNOLOGY, Linear. *LTC3256 Datasheet*. Dostupné tiež z: <https://www.analog.com/media/en/technical-documentation/data-sheets/3256fb.pdf>.
16. INSTRUMENTS, Texas. *MAX660 Datasheet*. Dostupné tiež z: <http://www.ti.com/lit/ds/symlink/max660.pdf>.
17. NXP. *MK20DX256VLH7 Datasheet*. Dostupné tiež z: http://cache.freescale.com/files/32bit/doc/data_sheet/K20P64M72SF1.pdf.

Prílohy

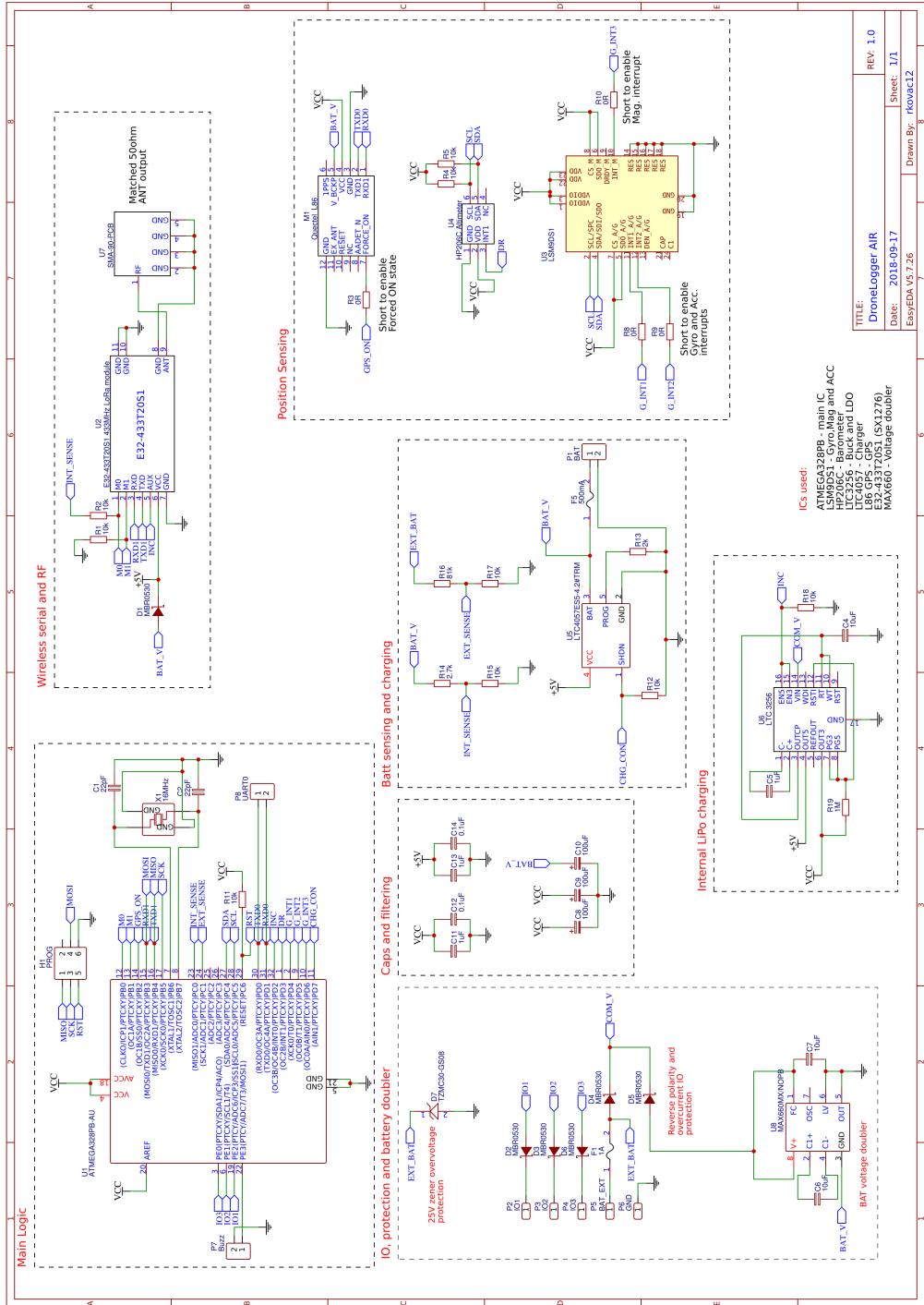
Zoznam príloh:

- Príloha A: Schéma modulu RC súpravy
- Príloha B: Schéma modulu bezpilotného stroja
- Príloha C: Rendery plošných spojov modulov
- Príloha D: Použité vývojové prostredia
- Príloha E: Softvér druhej a tretej verzie - Arduino kód
- Príloha F: Softvér tretej verzie - Nextion HMI
- Príloha G: Softvér druhej verzie - Displeje na báze ILI9341

Príloha A

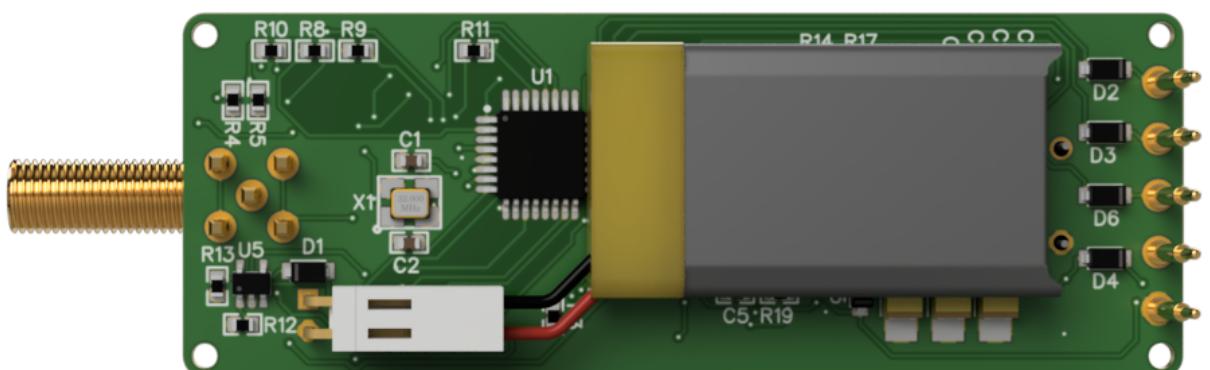
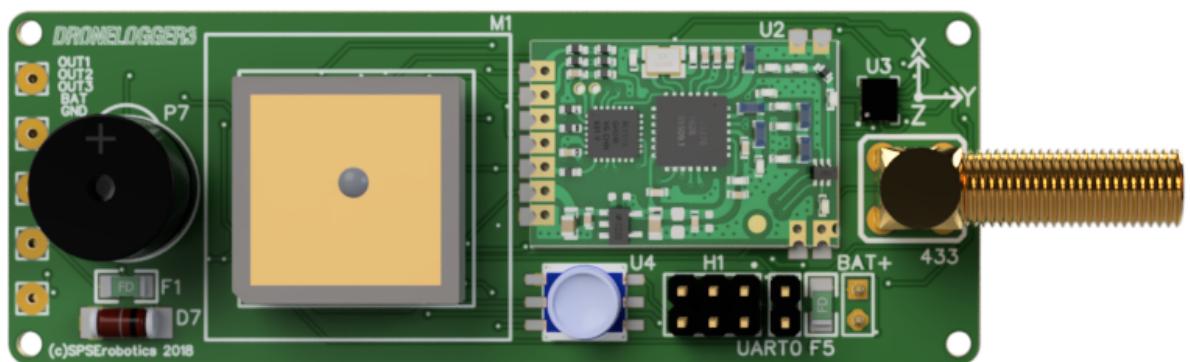


Príloha B

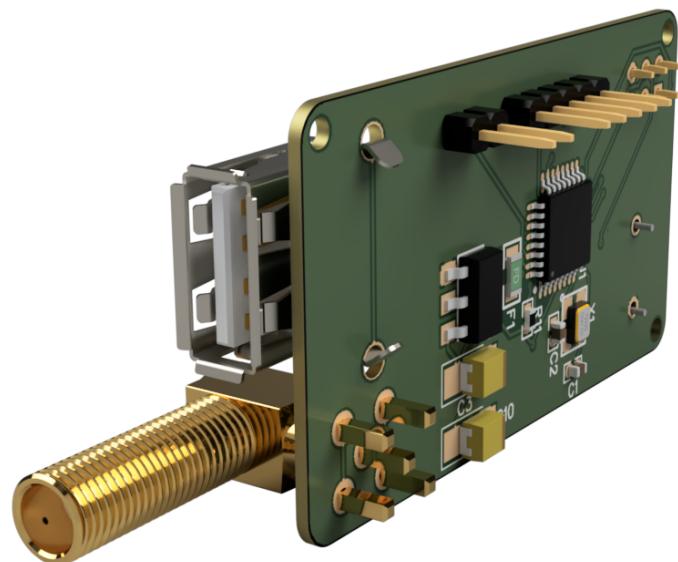
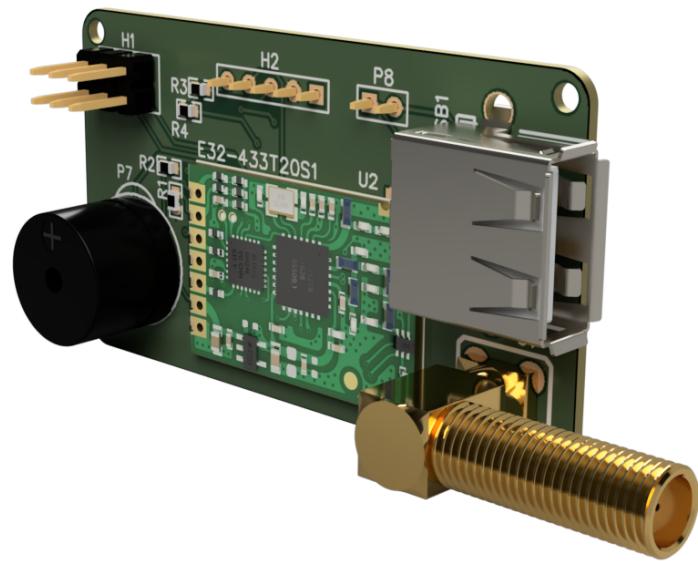


Príloha C

Modul DroneLogger3 AIR



Modul DroneLogger3 GROUND



Hardvérové revízie

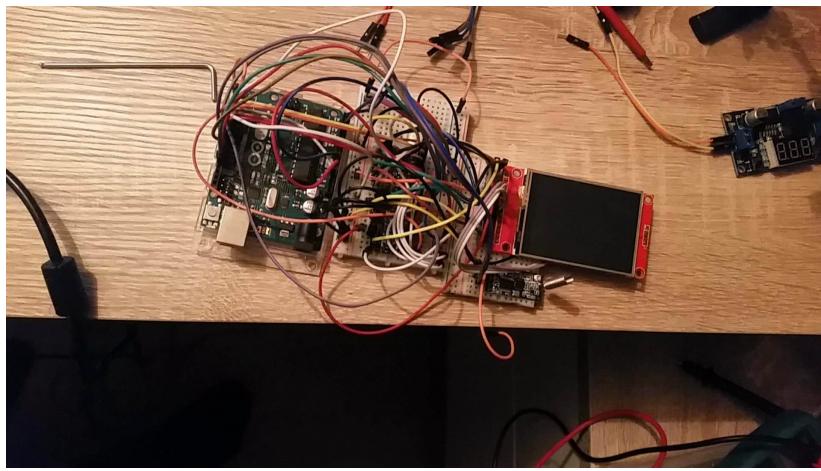
Ked'že je systém DroneLogger niekoľkoročný projekt prešiel viacerými hardvérovými aj softvérovými revíziami, od preliminárnych zapojení do kontaktného poľa až po optimalizovaný hardvér a softvér na mieru viditeľný v terajšej verzii systému.

V tejto kapitole sa pozrieme na jednotlivé iterácie systému a popíšeme ich základné vlastnosti a poznatky získané počas ich vývoja.

2.4 Hardvérové revízie

2.4.1 Rev. nultá

Ked'že náš projekt vychádza z našej vlastnej myšlienky zjednodušiť a zlepšiť navigovanie bezp. strojov v priestore aj hardvér vychádzal "z ničoho". Prvý nutný krok bol overenie myšlienok a technologických postupov v praxi zapojením rôznych senzorov do kontaktného poľa. Toto zapojenie bolo statické a slúžilo iba ako tzv. "*Feasibility study*" pre ďalšie revízie.



Obr. 2.12: Prvotné zapojenie "na stole"

2.4.2 Rev. prvá a druhá

Tieto prvé iterácie boli určené primárne na výber a vývoj hardvéru, softvéru, materiálov, umiestnenia samotného systému na lietajúcom stroji ako aj vysielači pilota a jeho komfortu používania. Tieto verzie neboli určené pre koncových užívateľov. Prvé verzie systému využívali jeden, neskôr dva grafické 2.8" displeje na báze čipu ILI9341 [12] na ktoré vypisuje dátu a grafiku priamo mikrokontrolér (najskôr Atmega328p [5], neskôr MK20DX256VLH7 [17]) ktorý ich ovláda. Toto bol jedným z hlavných dôvodov prechodu na dosku Teensy 3.2 ktorá nahradzuje pomalý AVR procesorom moderným s ARM architektúrou, zachováva si však plnú kompatibilitu s Arduino IDE. Neskôr sa začali prejavovať problémy s vykreslovaním dynamickej grafiky, hlavne umelého horizontu. Okrem technických sa vyskytli aj praktické problémy, riešenie s dvoma menšími displejmi po stranách je z užívateľského hladiska značne nepraktické. Z tohto dôvodu sme sa rozhodli prekopať hardvér a softvér a začať vývoj novej verzie.



Obr. 2.13: Prvá revízia s jedným panelom

2.4.3 rev. tretia

Pri jeho návrhu sme už mysleli na finálnu aplikáciu do praxe a tak sme „prijímacej“ časti prispôsobili komfortu užívateľa a hardvér „vysielacej“ časti jednoduchosti pripojenia a integrácie s lietajúcim strojom. Taktiež sme mysleli na húževnatosť „prijímacej“ časti čo je vidieť na tvare a prevedení krabičky. Problém s vykreslovaním dynamickej



Obr. 2.14: Tretia, finálna revízia plastových dielov

grafiky sme vyriešili implementovaním päťpalcového displeja od firmy Nextion ktorý má vlastnú pamäť a ovládanie elektroniku pre graficko-užívateľské rozhranie. Displej komunikuje s mikroprocesorom „prijímacieho“ modulu pomocou zbernice UART. Týmto sa vyriešil aj problém s nedostatočným výkonom predošej verzie a nutnosťou ARM mikrokontroléra. Ďalej sme prispôsobili hardvér tak aby ho bolo možné pripojiť priamo do RC súpravy pilota.

Fungovanie systému

V tejto kapitole si zhrnieme spoločné fungovanie jednotlivých modulov systému, ich pripojenie k jednotlivému vybaveniu pilota a sledovaného bezpilotného stroja.

2.5 Modul RC súpravy

Začnime od pilota. Ten má v rukách vysielačku ktorou ovláda bezpilotný stroj. Táto RC súprava má v zadnej časti priestor na univerzálne moduly typu JR. Do tohto priestoru je vložený modul RC súpravy. Tento modul je z vysielačky napájaný teda je uvedený do chodu pri jej zapnutí.



Obr. 2.15: Modul vo vysielačke

Je taktiež vyžadované externé napájanie z dôvodu zachovania kompatibility so všetkými RC súpravami - v závislosti od zvoleného zobrazovacieho panelu môže dôjsť k prekročeniu max. prúdu pre externý modul a dôjde k jeho vypnutiu. z tohto dôvodu

je nutné pripojiť externý akumulátor s napäťom v rozsahu 6 - 12V. K modulu je cez SMA konektor pripojená externá anténa - jej smerosť a polarizáciu užívateľ zvolí v závislosti od jeho konkrétnych požiadaviek. Možnosť pripojiť externú anténu robí systém vysoko flexibilný. Hned' vedľa SMA konektora sa nachádza USB konektor typu A ktorý poskytuje sériové rozhranie na pripojenie zobrazovacieho panelu. USB konektor bol zvolený z dôvodu vysokej flexibility - užívateľ vyberá dĺžku kábla prepojujúceho panel s modulom podľa potreby

2.6 Modul Bezpilotného stroja

Iked' je modul schopný plne autonómneho chodu z interného napájacieho zdroja, je doporučené mať pripojené hlavné napájacie napäťie sledovaného bezpilotného stroja z niekoľkých dôvodov:

- Kapacita interného akumulátora je relatívne nízka a jedná sa skôr o záložný zdroj ako o akumulátor na trvalé napájanie počas bežného chodu
- Pripojenie externého napájania umožňuje toto napäťie sledovať a upozorniť pilota v prípade vybitia pripojeného externého akumulátora.

Ked'že modul bezpilotného stroja nemá žiadne "tlačítko" na vypnutie či zapnutie je táto funkcia implementovaná pomocou WOR funkcie LoRa modulov - po zapnutí modulu RC súpravy a odoslaní náhodných dát je po ich prijatí protiľahlým modulom, ktorý je napájaný interným akumulátorom a udržiavaný v režime spánku aktivuje svoj WOR výstup a tým prebudí interný regulátor čo má za následok prebudenie zvyšnej elektroniky a uvedenie celého systému do chodu.



Obr. 2.16: zapnutý zobrazovací panel

Po uvedení systému do chodu sa zobrazujú nasledovné dátá na zobr. paneli:

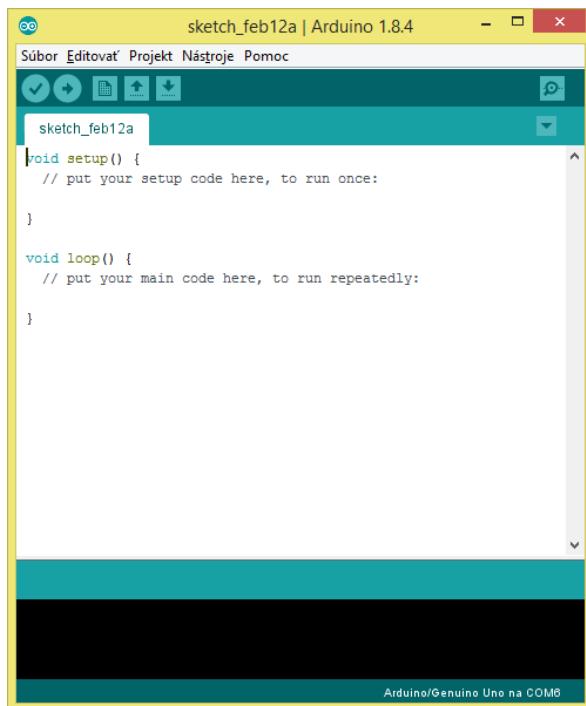
- GPS súradnice domovského miesta (miesta prvého zapnutia modulu v danom napájacom cykle) a GPS súradnice bodu, na ktorom sa modul práve nachádza.
- GPS čas a dátum, indikátor prichádzajúcich dát a indikátory úrovne nabitia akumulátorov.
- Štyri digitálne prístroje: Výškomer zobrazujúci nadmorskú výšku v metroch, rýchlomer zobrazujúci (pozemnú) rýchlosť v kilometroch za hodinu, kompas ukazujúci pozíciu bezpilotného stroja v danom momente a ukazovateľ smeru "domov" a umelý horizont

Softvér - vývojové prostredia

2.7 Vývojové prostredie Arduino IDE

Arduino IDE (Integrated Development Environment) je multiplatformová aplikácia pre operačné systémy Windows, MacOS a Linux. Prostredie Arduino IDE je určené na vytváranie, kompliaciu a nahrávanie softvéru do vývojových dosiek Arduino a kompatibilných mikroprocesorov (AVR a ARM). Arduino IDE podporuje upravené programovacie jazyky C a C++ pomocou špeciálnych pravidiel štruktúrovania.

Používateľov zdrojový kód nutne vyžaduje len dve základné funkcie - funkciu `setup`, ktorá sa spustí na začiatku a funkciu `loop`, ktorá beží v slučke. Tieto dve funkcie sú následne skompilované a spojené so stub `main()` do spustiteľného cyklického exekučného programu.



Obr. 2.17: Vývojové prostredie Arduino IDE

Celý náš softvér je napísaný vo vývojovom prostredí Arduino IDE v programovacom

jazyku C++. Pri jeho tvorbe sme zvolili metódu OOP (Objektovo orientované programovanie). Tento spôsob zvyšuje efektívnosť programu, a taktiež zjednoduší jeho údržbu, prípadné úpravy za účelom zefektívnenia chodu.

2.8 Vývojové prostredie Nextion Editor

Je určený na tvorbu graficko-užívateľských rozhraní pre HMI displeje Nextion. V editore Nextion je možné rýchlo a efektívne vytvárať grafiku pomocou drag-and-drop (ťahaj a pusti) komponentov (grafika, text, tlačidlá, posuvníky, ...). Má vlastnú inštrukčnú sadu pre programovanie interakcií s komponentmi displeja. Toto vývojové prostredie má taktiež možnosť simulácie navrhnutej grafiky priamo v editore.

2.8.1 HMI displej Nextion

Je HMI (Human Machine Interface) riešenie, ktoré kombinuje TFT dotykový displej, grafický processor a internú pamäť. S displejom je možné komunikovať pomocou UART sériovej zbernice.

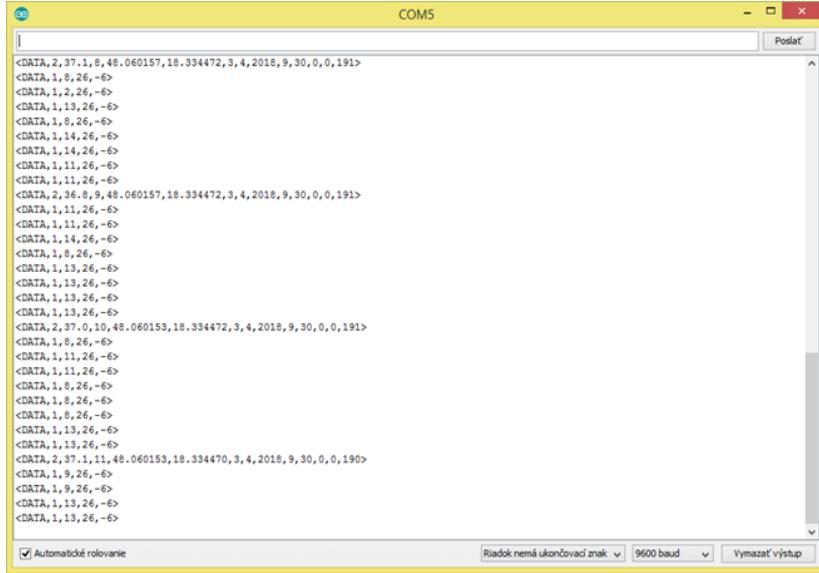


Obr. 2.18: HMI panel

Softvér tretej a druhej verzie - Arduino kód

2.9 Bezdrôtová sériová komunikácia

Na prenos údajov medzi zariadeniami DroneLogger Air a DroneLogger Ground sú implementované LoRa (Long Range – dlhý dosah) bezdrôtové sériové moduly. Zariadenie komunikuje so sériovým modulom pomocou UART zbernice. Taktiež sme museli myslieť aj na to, že na frekvencii na ktorej pracujú naše sériové moduly, môžu byť vysielané aj iné, cudzie dáta. Na zabezpečenie komunikácie sme museli navrhnúť akou formou posieláť packety s jednotlivými dátami. Každý packet ktorý je vyslaný z lietajúceho zariadenia má začiatočný a ukončovací znak, zvolili sme zobáčikové zátvorky “< >”. Tým sme zabezpečili oddelenie každej správy, ktorá bude vyslaná. Bolo taktiež nutné navrhnúť ako oddelovať jednotlivé údaje v jednotlivých packetoch. Ako oddelovací znak sme zvolili čiarku. Taktiež sme sa zamysleli nad tým, či je potrebné odosieláť všetky údaje naraz. Po úvahe sme sa rozhodli, že budeme odosieláť dva druhy packetov, jeden obsahujúci údaje o náklone lietajúceho stroja a jeho smere, a druhý obsahujúci údaje o GPS pozícii, čase, výške a rýchlosťi. Z dôvodu odosielania väčšieho počtu packetov bolo potrebné pridať parameter ktorý rozhoduje o type packetu.



Obr. 2.19: Vizualizácia paketov

2.9.1 DroneLogger Air – odosielanie dát

Komunikácia je rozdelená na dva typy packetov, ktoré sú odosielané v rôznych intervaloch. Packet s údajmi o náklone a smere je vysielaný každých 100ms. Ostatné údaje o GPS pozícii, čase, výške a rýchlosťi sú odosielané v intervale 1000ms. Meranie intervalu medzi packetmi je zabezpečené pomocou časovača v mikroprocesore. Ako môžeme vidieť premennú *currentMillis*, ktorá predstavuje aktuálny počet milisekúnd na počítadle. Podmienka (*currentMillis - previousMillis >= interval*) nám zabezpečí odosielanie dát v intervaloch ktoré nastavíme. Pri každom splnení podmienky sa do premennej *previousMillis* zapíše aktuálna hodnota počítadla a je následne odoslaný konkrétny packet.

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    Serial.print("<DATA,1,");
    Serial.print(heading);
    Serial.print(",");
    Serial.print(pitch_final);
    Serial.print(",");
    Serial.print(roll_final);
    Serial.println(">");
}
```

Obr. 2.20: premenná *currentMillis*

2.9.2 DroneLogger Ground – prijímanie odoslaných dát

Ako už bolo spomenuté softvér je napísaný pomocou OOP. Pre prijímanie dát je vytvorená trieda *Communication*. Obsahom triedy *Communication* sú metódy pre inicializáciu komunikácie, prijímanie a delenie prijatých údajov do jednotlivých premenných. Metóda *initialization()* slúži na inicializáciu sériovej komunikácie. Metóda *recvWithStartEndMarkers()* zabezpečí načítanej správy zo sériovej komunikácie len od začiatočného znaku až po ukončovací znak. Metóda *parseData()* vykonáva rozdelenie prijatej správy pomocou oddelovacieho znaku na jednotlivé premenné. Metóda *receive()* spája metódy *recvWithStartEndMarkers()* a *parseData()* do jednej a zároveň zabezpečuje, aby sa metóda *parseData()* vykonávala len vtedy, ak bola prijatá ďalšia, nová správa

```
class Communication {
public:
    void initialization() {
        Serial.begin(9600);
    }
    void receive() {
        recvWithStartEndMarkers();
        if (newData == true) {
            strcpy(tempChars, receivedChars);
            parseData();
            newData = false;
        }
    }
    void recvWithStartEndMarkers() {
        static boolean recvInProgress = false;
        static byte ndx = 0;
        char startMarker = '<';
        char endMarker = '>';
        char rc;
        while (Serial.available() > 0 && newData == false) {
            rc = Serial.read();
            if (recvInProgress == true) {
                if (rc != endMarker) {
                    receivedChars[ndx] = rc;
                    ndx++;
                    if (ndx >= numChars) {
                        ndx = numChars - 1;
                    }
                } else {
                    receivedChars[ndx] = '\0';
                    recvInProgress = false;
                    ndx = 0;
                    newData = true;
                }
            } else if (rc == startMarker) {
                recvInProgress = true;
            }
        }
    }
}
```

Obr. 2.21: trieda Communication

V metóde *parseData()* môžeme vidieť rozdelenie typov prijatých packetov pomocou premennej *packet*. Táto premenná je vždy na prvom mieste v každom type packetu. Po načítaní hodnoty z prvého miesta v pakete sa rozhodne o type prijatého packetu. Potom sa už jednotlivé hodnoty rozdeľujú pomocou oddelovacieho znaku. Pomocou funkcie *strtok()* vieme rozdeliť packet, ktorý sme prijali vo forme reťazca, na jednotlivé dátá. Jednotlivé dáta konvertujeme pomocou funkcií *atof*, *atoi*, *atol* a následne

uložíme do premenných. Rôzne dĺžky packetov sú z dôvodu potrebného dlhšieho času pri prijímaní dlhších packetov. Pri prijímaní packetu z dlhším počtom premenných je potrebné viac času na spracovanie. Všetky parametre nie je potrebné aktualizovať každých 100ms. Napríklad pri údajoch o polohe (GPS) je úplne postačujúce aktualizovanie každých 1000ms, čiže každú sekundu. Rozdelením packetov na dva tipy sa výrazne zrýchliло prijímanie dát. Každých 100ms sa odosielajú údaje o uhle stúpania , uhle náklonu a azimute, Ostatné údaje ako je poloha, rýchlosť ,výška a čas každú sekundu.

```

void parseData() {
    int packet;
    char * strtokIndx;
    strtokIndx = strtok(tempChars, ",");
    strcpy(messageFromDroneLogger, strtokIndx);
    strtokIndx = strtok(NULL, ",");
    packet = atoi(strtokIndx);
    if (packet == 1) {
        strtokIndx = strtok(NULL, ",");
        smoothHeadingDegrees = atoi(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        pitch = atoi(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        roll = atoi(strtokIndx);
        pitch += 5;
    }
    else if (packet == 2) {
        strtokIndx = strtok(NULL, ",");
        temperature = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        fix_data = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        latitude = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        longitude = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        gps_month = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        gps_day = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        gps_year = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        time_hour = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        time_minute = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        fix_age = atol(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        speed_kmph = atoi(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        bmp_altitude = atoi(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        gps_course = atoi(strtokIndx);
    }
}

```

Obr. 2.22: metóda parseData()

2.10 Uhol náklonu

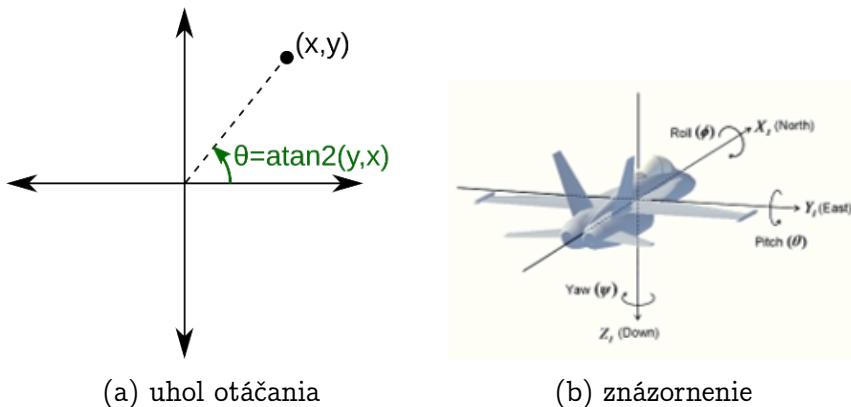
2.10.1 Načítavanie údajov zo senzora

Pre komunikáciu mikroprocesora s senzorom (akcelerometer, gyroskop) sme použili I2C zbernicu. Súčasťou vývojového prostredia Arduino IDE je na tento úkon uspôsobená knižnica Wire. Ako prvé musíme inicializovať I2C komunikáciu na adresu senzora. Potom už následne môžeme čítať hodnoty jednotlivých registrov a ukladať do konkrétnych premenných.

```
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr, 14, true);
accX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
accY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
accZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
Temp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
gyroX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
gyroY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
gyroZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
```

Obr. 2.23: knižnica Wire

2.10.2 Výpočet uhlu náklonu



Pri výpočte uhlu osi otáčania (roll) sme vychádzali zo vzorca $\theta = \text{atan}2(y, x)$, ktorý nám vracia uhol medzi rovinou a bodmi x, y. Pre výpočet potrebujeme za body x, y dosadiť akceleráciu v osiach y a z ($accY$, $accZ$). Teda vzorec na výpočet uhlu otáčania v radiánoch je

$$\text{roll} = \text{atan}2(accY, accZ)$$

Aby sme premenili výsledok v radiánoch na stupne do vzorca sme doplnili konštantu RAD_TO_DEG = 57.2957795130823.

$$\text{roll} = \text{atan}2(accY, accZ) * \text{RAD_TO_DEG}$$

Pri výpočte uhlu osi stúpania (pitch) sme vychádzali zo vzorca $\text{atan}\left(\frac{x}{\sqrt{y^2+z^2}}\right)$. Pre výpočet potrebujeme za x,y,z dosadiť akceleráciu vo všetkých troch osiach ($accX$, $accY$, $accZ$), teda vzorce pre výpočet uhlu stúpania v radiánoch a stupňoch budú vizerať nasledovne:

$$\text{pitch} = \text{atan}\left(\frac{-accX}{\sqrt{accY^2 + accZ^2}}\right)$$

$$\text{pitch} = \text{atan}\left(\frac{-accX}{\sqrt{accY^2 + accZ^2}}\right) * \text{RAD_TO_DEG}$$

```
roll = atan2(accY, accZ) * RAD_TO_DEG;
pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
```

Obr. 2.25: v jazyku C++

2.10.3 Aplikovanie Kalmanovho filtra na výpočet uhlu náklonu

Tu sme implementovali knižnicu Kalman.h, konkrétnie metódy pre zadanie uhlu a čítanie vypočítaného uhlu náklonu, čiže metódy *setAngle* a *getAngle*. Metóda *setAngle* je použitá na nastavenie počiatočného uhlu náklonu pri výpočte. Metóda *getAngle* je použitá na získanie uhlu s aplikovaným filtrom. Metóda *getAngle* má tri parametre. Prvým parametrom je uhol, druhým je výchylka tretím parametrom je delta time (čas medzi meraniami).

```
kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
```

Obr. 2.26: Kalmanov filter

Výchylka je načítaná priamo z gyroskopu, avšak je nutné ju prepočítať do stupňov za sekundu. Citlivosť gyroskopu nastavíme podľa aplikácie v ktorú ho chceme použiť, pri lietajúcich zariadeniach je dôležitá najmä citlivosť, čiže čo možno najvyššia citlivosť je želaná. Pri najcitlivejšom nastavení gyroskopu je hodnota LSB($^{\circ}/\text{s}$) = 131. Hodnotu načítanú z gyroskopu vydelíme číslom zisteným z datasheetu daného gyroskopu podľa nastavenej citlivosti.

```
double gyroXrate = gyroX / 131.0;
double gyroYrate = gyroY / 131.0;
```

Obr. 2.27: prepočet na stupne za sekundu

Výpočet času medzi meraniami sme realizovali pomocou interného časovaču mikroprocesora. Prostredie Arduino IDE má implementovanú funkciu *micros()* ktorej návratová hodnota je čas v mikrosekundách. Výpočet sa teda rovná $dt = \text{aktuálny_čas} - \text{minulý_čas}$. Keďže do metódy *getAngle* musíme dosadiť čas v sekundách vzorec ešte vydelíme číslom 1000000.

```
double dt = (double)(micros() - timer) / 1000000;  
timer = micros();
```

Obr. 2.28: výpočet delta time

2.11 GPS

2.11.1 Komunikácia s GPS modulom

Komunikácia s GPS modulom je zabezpečená pomocou UART zbernice, čiže sériovej zbernice. Mikroprocesor Atmega328p má jednu UART zbernicu, ktorá je už použitá na bezdrôtovú sériovú komunikáciu. Riešením je implementácia softvérovej sériovej zbernice pomocou knižnice *SoftwareSerial*. Táto knižnica emuluje UART zbernicu na pinoch, ktoré si zvolíme.

```
static const int RXPin = A2, TXPin = A3;  
SoftwareSerial ss(RXPin, TXPin);
```

Obr. 2.29: vytvorenie konštruktora triedy SoftwareSerial

2.11.2 Čítanie hodnôt z GPS modulu

Pre čítanie hodnôt z GPS modulu sme implementovali knižnicu TinyGPS++. Pomocou tejto knižnice je možné čítať všetky parametre, ktoré nám GPS modul poskytuje. Táto knižnica je napísaná pomocou OOP (Objektovo orientovaného programovania), a tak je nutné vytvoriť konštruktor triedy TinyGPS++ ktorý sme nazvali *gps*.

```
TinyGPSPlus gps;
```

Obr. 2.30: vytvorenie konštruktora triedy TinyGPS++

Následne môžeme pomocou metód čítať jednotlivé parametre, ktoré budeme potrebovať. Aby sme zabránili zbytočnému spomaľovaniu mikroprocesora tým, že bude aktualizovať parametre každý cyklus, parametre sa aktualizujú iba vtedy, keď

sa daný parameter zmení. Táto možnosť je zabezpečená pomocou metódy triedy TinyGPS++ *isUpdated()*. Metóda má boоловskú návratovú hodnotu.

```
if (gps.location.isUpdated()) {
    latitude = gps.location.lat();
    longitude = gps.location.lng();
}
```

Obr. 2.31: aktualizovanie iba v prípade zmeny

Pomocou tejto knižnice je načítavaná zemepisná šírka (*latitude*), zemepisná dĺžka (*longitude*), gps rýchlosť, dátum a čas. Tieto parametre nám postačujú na zaznamenávanie polohy.

2.11.3 Výpočet vzdialenosť od miesta štartu

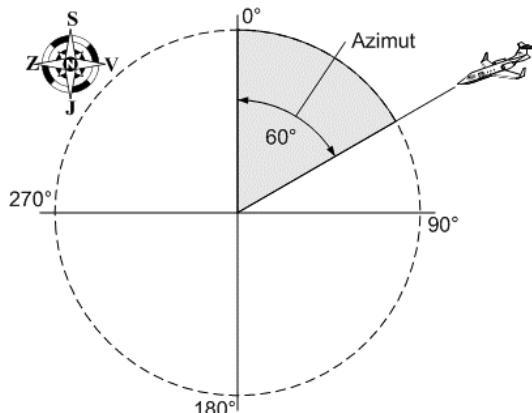
Je realizovaný pomocou metódy *distanceBetween*, ktorá je súčasťou knižnice TinyGPS++. Táto metóda má štyri parametre. Prvé dva sú zemepisná šírka a zemepisná dĺžka bodu A a druhé dva parametre sú zemepisná šírka a zemepisná dĺžka bodu B. Návratová hodnota tejto metódy je vzdialenosť z bodu A do bodu B v metroch. Bod A, teda zemepisnú šírku štartu, získame zapísaním prvej platnej polohy.

```
distanceToHome = TinyGPSPlus::distanceBetween(latitude, longitude, start_latitude, start_longitude);
```

Obr. 2.32: vytvorená premenná triedy TinyGPS++

2.11.4 Výpočet azimutu k miestu štartu

Azimut (kurz) je smerovanie lietajúceho stroja k určitému bodu. V letectve sa používa na navigáciu v priestore. V našom prípade chceme vypočítať azimut z miesta, kde sa aktuálne lietajúci stroj nachádza k miestu, odkiaľ vzlietol.



Obr. 2.33: azimut

Tak ako aj výpočet vzdialenosťi od miesta štartu tak aj výpočet kurzu k miestu štartu je realizovaný pomocou knižnice TinyGPS++ metódou *courseTo*. Táto metóda má taktiež štyri parametre. Prvé dva sú zemepisná šírka a zemepisná dĺžka bodu A, a druhé dva parametre sú zemepisná šírka a zemepisná dĺžka bodu B. Návratová hodnota metódy *courseTo* je azimut v rozsahu 0° až 359° stupňov.

```
courseToHome = TinyGPSPlus::courseTo({double}latitude, {double}longitude, {double}start_latitude, {double}start_longitude)
```

Obr. 2.34: vytvorená metóda uložená do premennej courseToHome

2.12 Kompas

2.12.1 Komunikácia s kompasom

Kompas komunikuje s mikroprocesorom pomocou I2C zbernice. Tak ako pri gyroskope sme implementovali knižnicu *Wire.h*. Ako prvé musíme inicializovať komunikáciu s kompasom pomocou metódy *begin()* knižnice *Wire*. Po inicializácii nastavíme I2C adresu pomocou knižnice *MechaQMC5883.h* metódou *init()*.

```
Wire.begin();
qmc.init();
```

Obr. 2.35: inicializácia a nastavenie adresy

2.12.2 Načítavanie údajov zo senzora

Na získavanie údajov zo senzora sme implementovali metódu *read()* knižnice *MechaQMC5883.h*. Ako prvé je potrebné vytvoriť objekt, pomenovali sme ho *qmc*.

```
MechaQMC5883 qmc;
```

Obr. 2.36: vytvorenie objektu qmc

Vytvorenie premenných x,y,z a azimuth na uloženie načítaných hodnôt. Vstupné hodnoty metódy *read()* sú len adresy premenných x,y,z a azimuth

```

int x, y, z;
int azimuth;
qmc.read(&x, &y, &z, &azimuth);

void MechaQMC5883::read(uint16_t* x,uint16_t* y,uint16_t* z,int* a){
    read(x,y,z);
    *a = azimuth(y,x);
}

```

Obr. 2.37: metóda read

Metóda *read()* je preťažovaná, v našom prípade sme použili metódu *read()* so štyrmi vstupnými premennými. Pri použití tejto možnosti si metóda ako prvú zavolá metódu *read()* s troma vstupnými parametrami. Táto možnosť načítava údaje priamo z registrov kompasu. Ako druhú zavolá metódu *azimuth()*, ktorá vypočíta azimut na základe vstupných hodnôt.

```

void MechaQMC5883::read(uint16_t* x,uint16_t* y,uint16_t* z){
    Wire.beginTransmission(address);
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.requestFrom(address, 6);
    *x = Wire.read(); //LSB x
    *x |= Wire.read() << 8; //MSB x
    *y = Wire.read(); //LSB z
    *y |= Wire.read() << 8; //MSB z
    *z = Wire.read(); //LSB y
    *z |= Wire.read() << 8; //MSB y
}

float MechaQMC5883::azimuth(uint16_t *a, uint16_t *b){
    float azimuth = atan2((int)*a,(int)*b) * 180.0/PI;
    return azimuth < 0?360 + azimuth:azimuth;
}

```

Obr. 2.38: metóda read

2.13 Barometer

2.13.1 Komunikácia s barometrom

Barometer komunikuje s mikroprocesorom pomocou I2C zbernice. Implementovaná je knižnica umožňujúca komunikáciu s I2C zariadeniami *Wire.h*. Ako prvé je potrebné inicializovať komunikáciu s barometrom pomocou metódy *begin()* knižnice *Adafruit_BMP280.h*. Nakoľko knižnica *Adafruit_BMP280.h* ponúka metódy *begin()*, nie je potrebné použiť metódu *begin()* knižnice *Wire*.

```
bme.begin(0x76);
```

Obr. 2.39: metóda read

2.13.2 Načítavanie údajov z barometra

Na získavanie údajov z barometra som implementoval metódu *readAltitude()*. Táto metóda ma jeden vstupný parameter a to je momentálny tlak vzduchu v hPa. Priemerná denná hodnota tlaku vzduchu je okolo 1005 hPa. Táto hodnota je dosadená pri výpočte výšky. Ako prvý krok je potrebné vytvoriť objekt, pomenovali sme ho *bme*.

```
Adafruit_BMP280 bme;
```

Obr. 2.40: vytvorenie objektu bme

Knižnica ponúka aj ďalšie metódy ako napr. *readTemperature()* a *readPressure()*. Metóda *readTemperature()* vracia teplotu v °C. Metóda *readPressure()* vracia tlak v Pa. Vytvorili sme triedu *Pressure*, ktorá má dve metódy *initialization()* a *getValuePressure()*.

```
class Pressure {
public:
    void initialization() {
        bme.begin(0x76);
    }
    void getValuePressure() {
        bmp_temperature = bme.readTemperature();
        bmp_pressure = bme.readPressure();
        bmp_altitude = (bme.readAltitude(1005));
    }
};
```

Obr. 2.41: vytvorenie triedy Pressure

2.14 SD Karta

2.14.1 Komunikácia s SD kartou

SD karta komunikuje s mikroprocesorom pomocou SPI zbernice. Na zabezpečenie komunikácie som implementoval knižnicu SPI.h a SD.h. Ako prvé je potrebné definovať pin, na ktorom je pripojený CS (*Chip Select*) SD karty. Po zadefinovaní pinu CS je

už možné začať komunikáciu s SD kartou pomocou metódy `begin()` knižnice `SD.h`. Taktiež je potrebné zistit, či je SD karta pripojená. Ak nie je, do sériového portu sa vypíše chybové hlásenie „*Card failed, or not present*“ (SD karta zlyhala alebo nie je pripojená).

```
if (!SD.begin(chipSelect)) {  
    Serial.println("Card failed, or not present");  
}
```

Obr. 2.42: chybové hlásenie

Na zápis údajov na SD kartu sme vytvorili súbor `datalog.txt` do ktorého sú zapisované dátá – číslo záznamu, dátum, čas a GPS poloha. Ak sa podarilo otvoriť súbor `datalog.txt` tak je možné zapísat údaje do súboru.

```
File dataFile = SD.open("datalog.txt", FILE_WRITE);  
  
if (dataFile) {  
    dataFile.print(i);  
    dataFile.print(",");  
    dataFile.print(gps_day);  
    dataFile.print("/");  
    dataFile.print(gps_month);  
    dataFile.print(",");  
    dataFile.print(time_hour);  
    dataFile.print(":");  
    dataFile.print(time_minute);  
    dataFile.print(",");  
    dataFile.print(latitude, 6);  
    dataFile.print(",");  
    dataFile.println(longtitude, 6);  
    dataFile.close();  
    i++;  
}
```

Obr. 2.43: otvorenie .txt súboru a podmienka na zápis

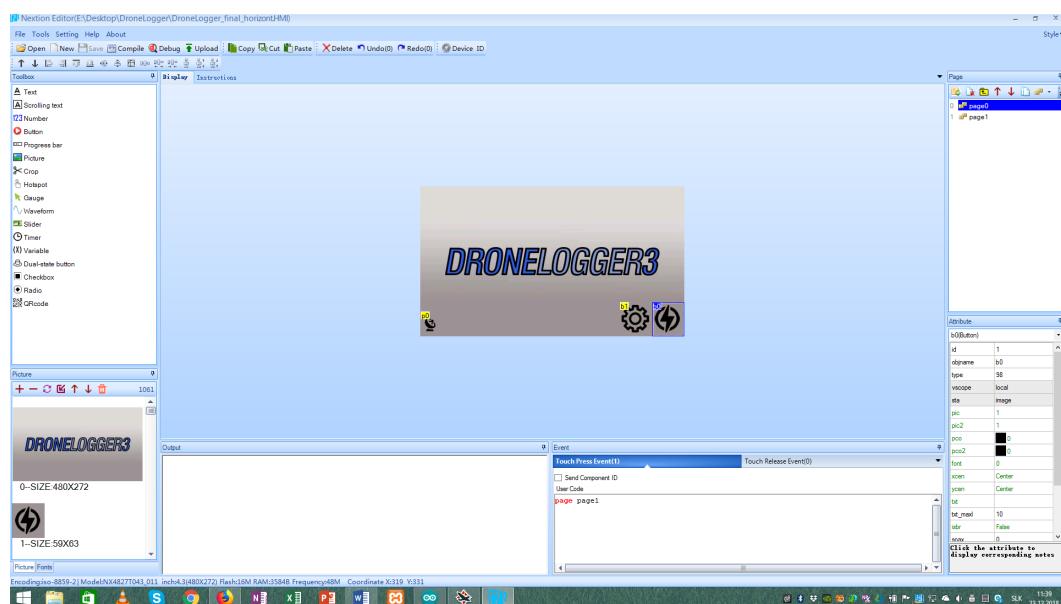
2.14.2 Formát SD karty

SD karta musí byť naformátovaná do formátu FAT16 alebo FAT32. Iné formáty SD karty nie sú podporované knižnicou `SD.h`.

Softvér tretej verzie - Nextion HMI

2.15 Editor Nextion

Návrh celého graficko-užívateľského rozhrania pre displej Nextion je vytvorený v editore dodávanom výrobcom - Nextion Editor. V tomto editore je možné jednoducho a efektívne vytvoriť graficko-užívateľské rozhranie s vlastným jedinečným dizajnom.



Obr. 2.44: editor Nextion

Celá grafika sa skladá z obrázkov ktoré sú uložené v pamäti displeja. Na výpis textu na displej je implementovaná funkcia Text. Každému textovému poľu je nutné definovať rozmery, pozadie a meno. Následne do vytvoreným polí je možné zapisovať akýkoľvek reťazec, buď nadefinovaný v prostredí Nextion alebo poslaný z DroneLogger Ground.



Obr. 2.45: vytvorený dizajn hlavnej obrazovky

2.16 Ovládanie grafiky cez UART

Displej Nextion komunikuje s DroneLogger Ground pomocou UART (sériovej) zbernice. Do displeja sú odosielané potrebné dátá na aktualizáciu grafiky podľa aktuálnych hodnôt. Pri zápisе hodnoty do displeja musíme ako prvé zvoliť stranu (*page1*), funkciu (*z0*) a parameter funkcie, chceme zmeniť hodnotu čiže meníme parameter *val*. Potom následne ukončíme sériovú komunikáciu pomocou funkcie *writeToNextion()*.

```

Serial.print("page1.z0.val=");
Serial.print(smoothHeadingDegrees);
writeToNextion();

if (last_bmp_altitude != bmp_altitude and bmp_altitude > 0) {
    Serial.print("page1.t0.txt=");
    Serial.print("'");
    Serial.print((int)bmp_altitude);
    Serial.print("'");
    writeToNextion();
    animacia();
}

```

Obr. 2.46: sériová komunikácia

2.16.1 Ošetrenie výpisu

Aby sa hodnota aktualizovala len, vtedy ak sa jej hodnota zmení, výpis ošetríme podmienkou. Na konci každého cyklu zapíšeme aktuálne hodnoty do premenných z názvom *last_nazov-premnnej* a potom ich porovnávame s aktuálnou hodnotou.

2.16.2 Funkcia writeToNextion()

Z inštrukčnej sady displeju vieme, že všetky inštrukcie, ktoré sú vyslané cez sériovú zbernicu musia byť ukončené tromi bajtami 0xFF. Nakoľko sme si uvedomili, že budeme vysielať veľa hodnôt, na zjednodušenie budúceho zápisu sme naprogramovali funkciu, ktorá vyšle do displeja tri bajty 0xFF.

```
void writeToNextion() {  
    Serial.write(0xff);  
    Serial.write(0xff);  
    Serial.write(0xff);  
}
```

Obr. 2.47: ukončenie zápisu

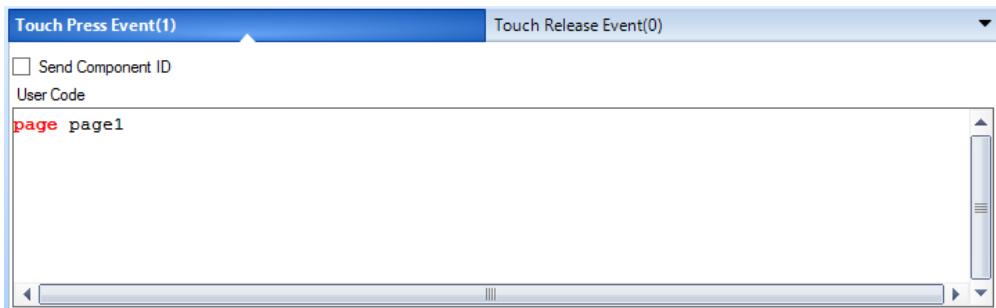
2.17 Domovská obrazovka

Tvorí ju pozadie s logom DroneLogger. V ľavom dolnom rohu je ukazovateľ ,ktorý znázorňuje príjem signálu. V pravom dolnom rohu je vstupná ikona odkazu na obrazovku s letovými prístrojmi. Vedľa nej je ikona odkazu smerujúceho do nastavení.



Obr. 2.48: domovská obrazovka

Každej ikone je možné definovať rozmery, pozíciu a pozadie. Displej Nextion má svoju vlastnú inštrukčnú sadu pomocou ktorej je možné ovládať displej. Jednou z inštrukcií je napr. *page*. Pomocou tejto funkcie môžeme meniť aktuálnu snímku, ktorú displej zobrazuje.



Obr. 2.49: zmena aktuálnej snímky

2.18 Hlavná obrazovka s letovými prístrojmi

2.18.1 Tvorba a dizajn letových prístrojov

Pri tvorbe dizajnu letových prístrojov sme sa inšpirovali prístrojmi zo skutočných lietadiel. Pre výškomer a rýchlomer sme zvolili jednoduchý dizajn s digitálnym ukazovateľom. Oba prístroje tvorí jeden obrázok na pozadí a pole, v ktorom sa zobrazuje hodnota daného prístroja. Kompas je taktiež vytvorený pomocou jedného obrázku na pozadí a dvoch ukazovateľov smeru, červený pre aktuálne smerovanie lietajúceho stroja a druhý ukazujúci smer ku miestu štartu. Umelý horizont je tvorený veľkým množstvom obrázkov. Umelý horizont je možné vykresľovať v osi otáčania $\pm 60^\circ$ a v osi stúpania $\pm 20^\circ$. Toto obmedzenie je len kvôli nedostatočnej pamäti displeja - 16MB.

2.18.2 Tvorba ručičky kompasu

Ako už bolo spomenuté dizajn kompasu tvorí obrázok ktorý je na pozadí. Ručičky prístroja tvoria komponenty softvéru Nextion, konkrétnie funkcia *Gauge*. Do tejto funkcie stačí zapísť hodnotu od 0° do 359° na nastavenie ručičky kompasu. Pri tvorbe v prostredí Nextion treba nastaviť rozmery, pozíciu, pozadie a názov funkcie. Ako pozadie je možné nastaviť tzv. *Crop Image* (Orezaný obrázok), čiže na pozadie použije tú časť obrázku, ktorú zaberá.



Obr. 2.50: kompas

Nakoľko editor Nextion neponúka možnosť posunu vstupnej hodnoty (hodnota nula funkcie Gauge sa rovná 270° požadovanej hodnoty), museli sme naprogramovať funkciu, ktorá posúva vstupnú hodnotu v rozsahu od 0° po 359° . Vstupnú hodnotu je potrebné rotovať o 90° .

```

if (course_offset >= 0)  {
    if (smoothHeadingDegrees + course_offset < 359)  {
        smoothHeadingDegrees += course_offset;
    }
    else  {
        smoothHeadingDegrees = smoothHeadingDegrees + course_offset - 359;
    }
}
else  {
    if (smoothHeadingDegrees + course_offset <= 0)  {
        smoothHeadingDegrees = 359 + (smoothHeadingDegrees + course_offset);
    }
    else  {
        smoothHeadingDegrees += course_offset;
    }
}
Serial.print("page1.z0.val=");
Serial.print(smoothHeadingDegrees);
writeToNextion();
    
```

Obr. 2.51: posun hodnoty

2.18.3 Tvorba umelého horizontu

Umely horizont tvorí funkcia *Picture*, ktorá nahradí určený rozmer obrázkom ktorý zvolíme. Funkcií *Picture* je nutné tak ako u funkcie *Gauge* definovať názov, rozmery a prvy obrázok, ktorý má zobrazit. Ako môžeme vidieť, veľkosť obrázku sme sa snažili zmeniť na čo najmenšiu tak, aby sa do displeja zmestilo čo najviac obrázkov. Dospeli sme k najmenšej možnej veľkosti funkcie *Picture* ktorá by nenarušovala dizajn umelého horizontu - 84x84 pixlov.



Obr. 2.52: umelý horizont

Po vytvorení obrázkov umelého horizontu bolo nutné vymyslieť spôsob, ako ich priradovať k jednotlivým hodnotám uhlu stúpania a uhlu náklonu. Obrázky bolo potrebné uložiť v poradí a to tak, aby obrázky v jednom uhle stúpania nasledovali numericky zoradené v celom rozsahu uhlu náklonu. Týmto spôsobom je potom možné pomocou funkcie *map* naprogramovať funkciu, ktorá namapuje hodnoty uhlu náklonu -50° až 50° na obrázky v rozsahu 60 až 11. Táto funkcia by fungovala iba pri uhle stúpania 0° . Tento výpočet je uložený do premennej *rollToPrint*. Ak chceme pridať ďalšie rozsahy jednoducho uložíme ďalší rozsah s iným uhlom stúpania a definujeme posun o 50 medzi obrázkami.

```

void pitchMode() {
    if (pitch >= -1 and pitch <= 1) {
        rollToPrint += 0;
    }
    //61-110
    if (pitch == -2 or pitch == -3) {
        rollToPrint += 50;
    }
    if (pitch == 2 or pitch == 3) {
        rollToPrint += 100;
    }
    if (pitch == -4 or pitch == -5) {
        rollToPrint += 150;
    }
    if (pitch == 4 or pitch == 5) {
        rollToPrint += 200;
    }
    if (pitch == -6 or pitch == -7) {
        rollToPrint += 250;
    }
    if (pitch == 6 or pitch == 7) {
        rollToPrint += 300;
    }
    if (pitch == -8 or pitch == -9) {
        rollToPrint += 350;
    }
    if (pitch == 8 or pitch == 9) {
        rollToPrint += 400;
    }
}

```

Obr. 2.53: mapovanie hodnoty náklonu

Pre posúvanie rozsahu sme vytvorili funkciu *pitchMode()*, ktorá na základe uhlu stúpania posunie rozsah obrázkov. Keď sa pozrieme na prvú podmienku môžeme vidieť, že ak je hodnota uhlu stúpania medzi hodnotami -1 a 1 rozsah neposúvame. Pri druhej podmienke už môžeme vidieť posun rozsahu o 50, ak je hodnota uhlu stúpania -2 alebo -3. Rozsah sa týmto spôsobom posúva až po hodnotu uhlu stúpania 20°.

```
void printHorizont() {
    pitchToPrint = pitch;
    rollToPrint = roll;
    if (roll >= -50 and roll <= 50) {
        rollToPrint = map(rollToPrint, -50, 50, 60, 11);
        pitchMode();
        Serial.print("page1.p3.pic=");
        Serial.print(rollToPrint);
        writeToNextion();
        animacia();
    }
}
```

Obr. 2.54: posun

Softvér druhej verzie - Displeje na báze ILI9341

Pri prvej verzii DroneLogger Ground sme sa rozhodli použiť dva grafické displeje ILI9341. Celá grafika letových prístrojov a letových údajov je vykreslovaná pomocou knižnice *Adafruit_ILI9341*. Táto knižnica nám poskytuje metódy pre vykreslovanie čiar, kruhov, štvorcov a písmen. Pre získavanie údajov z dotykovej vrstvy displeja sme implementovali knižnicu *URTouch*.

2.19 Výpis letových údajov na displej

Je realizovaný pomocou knižnice *Adafruit_ILI9341*. Ako prvé pri výpise textu je potrebné vyplniť pozadie textu. Na vyplnenie pozadia sme použili metódu *fillScreen*. Ďalej je potrebné nastaviť rotáciu displeja, je možné nastaviť 4 rotácie. Taktiež je potrebné nastaviť veľkosť textu a umiestnenie. Po nastavení parametrov už môžeme vypísat daný text pomocou metódy *print*. Týmto spôsobom je možné vypisovať text ak ho nie je potrebné aktualizovať počas cyklu.



(a) Zobrazené údaje

```
tft2.fillScreen(ILI9341_WHITE);
tft2.setRotation(0);
tft2.setTextColor(ILI9341_BLACK);
tft2.setTextSize(2);
tft2.setCursor(5, 2);
tft2.print("Incoming data: ");
```

(b) metóda fillScreen

2.19.1 Aktualizovanie textu podľa aktuálnej predlohy

Text, ktorý nie je statický, je potrebné aktualizovať podľa aktuálnej hodnoty. Bolo potrebné vymyslieť spôsob výpisu textu tak, aby nedochádzalo k prekrývaniu textu. Pre každý parameter ktorý je vypisovaný sme vytvorili funkciu, ktorá má tri vstupné hodnoty: umiestnenie v osiach x a y a veľkosť textu. Obsahom funkcie je podmienka ktorá aktualizuje text iba vtedy, ak sa zmenila vstupná hodnota. Funkcia pri aktualizovaní textu ako prvá vypíše text predošej hodnoty vo farbe pozadia, v našom prípade je to biela. Potom už následne vypíše novú hodnotu.

```
void printLatitude(int x, int y, int size)  {
    if (last_latitude != latitude) {
        tft2.setTextSize(size);
        tft2.setTextColor(ILI9341_WHITE);
        tft2.setCursor(x, y);
        tft2.print(last_latitude, 6);
        tft2.setTextSize(size);
        tft2.setTextColor(ILI9341_BLACK);
        tft2.setCursor(x, y);
        tft2.print(latitude, 6);
    }
}
```

Obr. 2.56: aktualizovanie textu

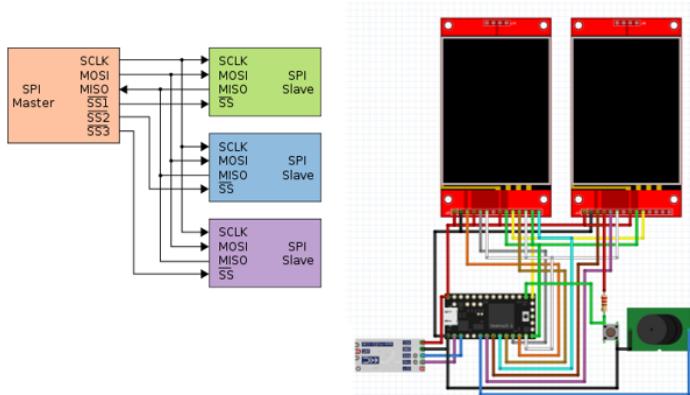
2.20 Komunikácia s displejom ILI9341

Displej ILI9341 komunikuje s mikroprocesorom cez SPI (Serial Peripheral Interface - Sériové periférne rozhranie) zbernicu. Pre komunikáciu s displejmi sme implementovali knižnicu *SPI.h* a *Adafruit_ILI9341*. Zapojenie displejov na SPI zbernicu je paralelné. Každý displej má samostatný CS (Change Slave) a DC. Pomocou makra v jazyku C *define* sme zadefinovali zámennu textu v zdrojovom kóde za číslu pinu kam je fyzicky pripojený pin displeju. Týmto spôsobom je možné rýchlo upravovať číslo pinu na ktorý je displej pripojený. Po nadefinovaní pinov už môžeme vytvoriť objekt displeja. Vytvoril som dva - *tft* a *tft2*.

```
#define TFT_CS 20
#define TFT_DC 21
#define TFT_CS_2 10
#define TFT_DC_2 6
#define t_SCK 14
#define t_CS 7
#define t_MOSI 8
#define t_MISO 3
#define t_IRQ 2
ILI9341_t3 tft = ILI9341_t3(TFT_CS, TFT_DC);
ILI9341_t3 tft2 = ILI9341_t3(TFT_CS_2, TFT_DC_2);
URTouch ts(t_SCK, t_CS, t_MOSI, t_MISO, t_IRQ);
```

Obr. 2.57: definícia pripojenia a vytvorenie objektov

V zapojení displejov mikroprocesor predstavuje SPI Master, displeje predstavujú SPI Slave. Na obrázku vpravo je znázornené zapojenie displejov k vývojovej doske Teensy 3.2.



Obr. 2.58: Zapojenie displejov

2.21 Vykresľovanie letových prístrojov

Celá grafika letových prístrojov je vytvorená pomocou knižnice *Adafruit_ILI9341*. Pri tvorení grafiky sme implementovali všetky možnosti knižnice. Použili sme metódy na vykreslovanie čiar, kruhov a štvorcov.

2.21.1 Grafika umelého horizontu

Ako prvé po zapnutí sa vykreslí umelý horizont s nulovými vstupnými hodnotami uhlu stúpania a uhlu náklonu. Hornú polovicu displeja vyplníme modrou farbou, ktorá symbolizuje oblohu, spodnú polovicu vyplníme hnedou farbou, ktorá symbolizuje zem. Po vykreslení hornej a dolnej polovice umelého horizontu sa vykreslí ukazovateľ uhlu stúpania pomocou funkcie *drawInfo()*.

```
tft.setRotation(0);
tft.fillRect(0, 0, 240, 160, ILI9341_BLUE);
tft.fillRect(0, 160, 240, 160, ILI9341_BROWN);
drawInfo();
```



(a) funkcia drawInfo()

(b) umelý horizont

Vykreslenie pomocou metódy *fillRect* má päť vstupných hodnôt, sú to súradnice x a y bodu odkiaľ má štvorec začínať, súradnice x a y bodu kde má štvorec končiť a farba ktorou má byť štvorec vyplnený.

2.21.2 Aktualizovanie grafiky umelého horizontu

Na vykreslovanie grafiky umelého horizontu sme vytvorili funkciu *drawHorizont*. Táto funkcia má dva vstupné parametre - *roll* (uhôl náklonu) a *pitch* (uhôl stúpania). Pomocou tejto funkcie je možné vykresliť umelý horizont. Funkcia vykresluje určitý počet čiar v určitom uhle podľa vstupných parametrov. Aktualizovanie zabezpečuje funkcia *updateHorizont()*. Funkcia aktualizuje umelý horizont iba vtedy, ak sa zmení aktuálna hodnota stúpania alebo aktuálna hodnota náklonu.

```
void updateHorizont() {
    if (last_roll != roll) {
        drawHorizon(roll, pitch);
        drawInfo();
    }
    if (last_pitch != pitch) {
        drawHorizon(roll, pitch);
        drawInfo();
    }
}
```

Obr. 2.60: funkcia *updateHorizont()*

2.21.3 Grafika kompasu

Celá grafiku kompasu je tvorená pomocou metód knižnice *Adafruit_ILI9341*. Červená ručička ukazuje aktuálny azimut, modrá ručička ukazuje azimut k miestu štartu. Ďalším prvkom grafiky je azimutová kružnica s rozdelením po desiatich stupňoch.

```
drawCourseToHome(courseToHome);
drawCompass(smoothHeadingDegrees);
```

(a) funkcie kompasu



(b) vykreslený kompas

2.21.4 Aktualizovanie grafiky kompasu

Na vykresľovanie grafiky kompasu sme vytvorili funkcie *drawCompass()* a *drawCourseToHome()*. Tieto funkcie majú iba jeden vstupný parameter - azimut. Funkcie vykreslia ručičky podľa zadaného azimutu. Aktualizovanie zabezpečuje funkcia *updateCompass()*. Súčasťou tejto funkcie je podmienka, ktorá aktualizuje kompas iba pri zmene azimutu.

```
void updateCompass() {
    if (last_smoothHeadingDegrees != smoothHeadingDegrees) {
        if (course_offset >= 0) {
            if (last_smoothHeadingDegrees + course_offset < 359) {
                last_smoothHeadingDegrees += course_offset;
            }
        } else {
            last_smoothHeadingDegrees = last_smoothHeadingDegrees + course_offset - 359;
        }
    } else {
        if (last_smoothHeadingDegrees + course_offset <= 0) {
            last_smoothHeadingDegrees = 359 + (last_smoothHeadingDegrees + course_offset);
        } else {
            last_smoothHeadingDegrees += course_offset;
        }
    }
    drawCourseToHome(courseToHome, 0);
    drawCompass(smoothHeadingDegrees, 0);
}
```

Obr. 2.62: funkcia *updateCompass()*

2.22 Dotyková vrstva

2.22.1 Načítavanie vstupov dotykovej vrstvy

Na načítavanie údajov z dotyковej vrstvy sme implementovali knižnicu *URTouch.h*. Ako prvé je potrebné definovať piny na ktorých je dotyková vrstva pripojená. Po nadefinovaní pinov je potrebné vytvoriť objekt. Pomenovali sme ho *ts*.

```
#define TFT_CS 20           #define t_SCK 14
#define TFT_DC 21           #define t_CS 7
                           #define t_MOSI 8 .
                           #define t_MISO 3 .
                           #define t_IRQ 2
URTouch ts(t_SCK, t_CS, t_MOSI, t_MISO, t_IRQ); |
```

Obr. 2.63: definícia vstupov a vytvorenie objektu

Po vytvorení objektu je nutné dotykovú vrstvu inicializovať. Tú inicializujeme pomocou metódy *InitTouch()*. Musíme taktiež nastaviť presnosť dotykovej vrstvy, tú nastavíme pomocou metódy *setPrecision()*.

```
        while (ts.dataAvailable())  {  
            ts.read();  
            x = ts.getX();  
            y = ts.getY();  
        }  
    }  
    ts.InitTouch();  
    ts.setPrecision(PREC_MEDIUM);
```

- (a) inicializácia dotykovej vrstvy (b) načítavanie hodnôt
z dotykovej vrstvy

Po inicializácí už môžeme načítať hodnoty z dotyковej vrstvy. Aby sme zabránili zbytočnému načítavaniu hodnôt, ošetrili sme tento problém pomocou metódy *dataAvailable()* knižnice *URTouch.h*. Návratová hodnota tejto metódy je typu *bool*, čiže true(pravda) alebo false(nepravda). Cyklus while sa bude vykonávať iba ak je hodnota true. Ak sa cyklus vykoná tak ako prvé prečíta údaje z dotykovej vrstvy. Pomocou metód *getX()* a *getY()* uložíme hodnoty do premenných x a y.