SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

Juraj Slačka

Autoreferát dizertačnej práce

RIADIACI POČÍTAČ SLOVENSKEJ DRUŽICE TYPU CUBESAT

na získanie akademickej hodnosti doktor (philosophiae doctor, PhD.)

v doktorandskom študijnom programe: **Kybernetika**
v študijnom odbore 2647 – 9.2.7 kybernetika

Miesto a dátum: Bratislava, 26.8.2015

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Juraj Slačka

Autoreferát dizertačnej práce

RIADIACI POČÍTAČ SLOVENSKEJ DRUŽICE TYPU CUBESAT

na získanie akademickej hodnosti doktor (philosophiae doctor, PhD.)

v doktorandskom študijnom programe:
Kybernetika

Miesto a dátum: Bratislava, 26.8.2015

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia

Na Fakulte Elektrotechniky a Informatiky STU Bratislava
Ústav robotiky a kybernetiky

Predkladateľ: Ing. Juraj Slačka, Ústav robotiky a kybernetiky, FEI STU Bratislava

Školiteľ: doc. Ing. Miroslav Halás, PhD.
Ústav robotiky a kybernetiky, FEI STU Bratislava

Oponenti: doc. Ing Martin Hromčík, Ph.D.
ČVUT v Prahe, Elektrotechnická fakulta
doc. Ing. Peter Fodrek, PhD.
Slovenská technická univerzita v Bratislave, FEI

Autoreferát bol rozoslaný: 15.7.2015

Obhajoba dizertačnej práce sa koná: 26.8.2015

Na Fakulte Elektrotechniky a Informatiky STU
Ústav robotiky a kybernetiky
Ilkovičova 3, Bratislava

Obsah

Tézy dizertačnej práce

- **Návrh hardvéru riadiaceho palubného počítača**
- **Návrh plánovača reálneho času s dôrazom na jeho bezpečnosť a robustnosť**
- **Návrh vhodných filtračných algoritmov na spracovanie signálov zo senzorov satelitu**
- **Implementácia stabilizačného algoritmu pre stabilizáciu cubesatu na orbite**

Úvod

Hlavným cieľom tejto práce bolo navrhnúť úplne od základov hardvér a softvér palubného počítača pre malú Slovenskú družicu. Pri návrhu hardvéru bolo dbané hlavne na jeho odolnosť voči rozličným vplyvom radiácie, pričom bola navrhnutá technika supervízora, ktorý prepína v prípade problémov medzi dvomi procesormi. Z hľadiska softvérového vybavenia palubného počítača bol kompletne navrhnutý operačný systém reálneho času, ktorý plne spĺňa požiadavky pre kritické systémy a zároveň má v sebe implementované mechanizmy na detekciu potenciálnych chýb spôsobených radiáciou. Záver práce je venovaný problematike stabilizačného algoritmu satelitu a jeho testovania v laboratórnych podmienkach na Zemi.

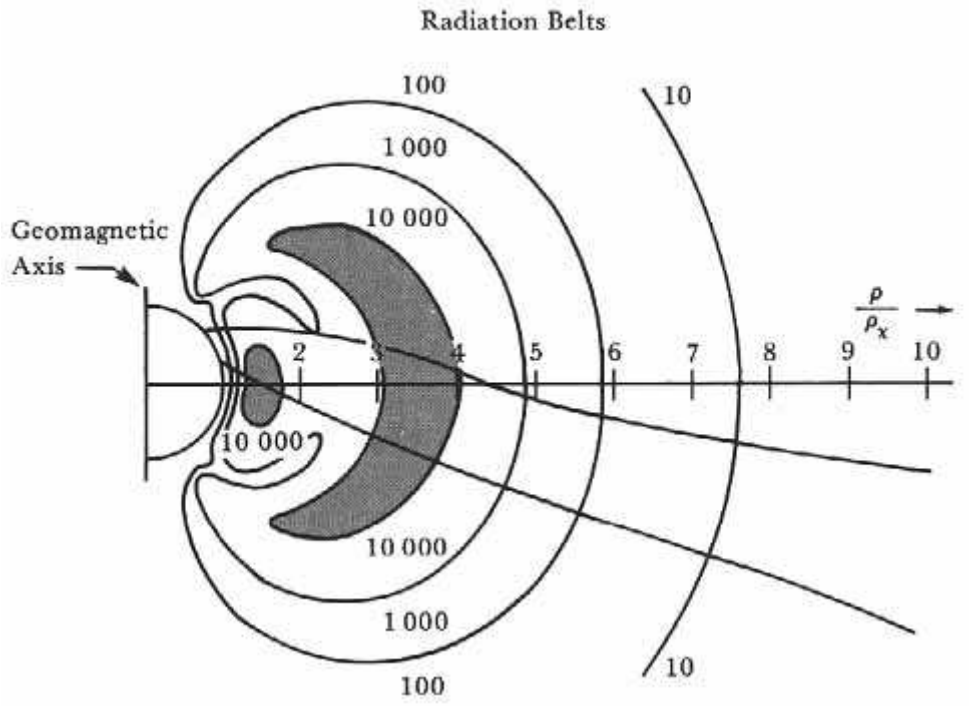
Hardvér palubného počítača

Palubný počítač bude hlavným riadiacim a komunikačným uzlom satelitu skCube. K palubnému počítaču budú pripojené všetky senzory na meranie stavu a orientácie satelitu, experiment, zdroj, modul rádiovej telemetrie a systém na riadenie magnetických cievok. Tým, že dátový tok v rámci satelitu bude hviezdicovej topológie, ktorej hlavným centrálnym komunikačným uzlom bude palubný počítač, pri jeho zlyhaní budú nefunkčné alebo neovládateľné všetky komponenty satelitu s výnimkou rádia, ktoré je do istej miery autonómne. Palubný počítač bude obsahovať okrem riadiacej jednotky (procesora a podporných obvodov) aj dátové úložisko pre operačný systém, aplikácie a vedecké dáta. Ďalšími komponentmi budú prípoje všetkých komunikačných zberníc satelitu, systém pre riadenie budiaceho prúdu cievok, teplomery a MEMS senzory na určenie orientácie satelitu.

Riadiaci procesor palubného počítača

Pri voľbe procesora pre palubný počítač bolo zohľadnené množstvo architektúr dostupných na komerčnom trhu. Ako prvá bola vylúčená architektúra x86 z dôvodu jej vysokej spotreby a produkovania nadmerného tepla. Táto architektúra je stará a nie vhodná pre vnorené systémy. Bolo jasné, že procesor musí byť kvôli spotrebe zo segmentu mikročipov určených pre vnorené zariadenia. Druhou architektúrou, ktorá bola vylúčená, je architektúra MIMPS, ktorej procesor je použitý napríklad aj v prenosnej hracej konzole Playstation Portable. Túto architektúru sme vylúčili preto, že sa jedná o mŕtvu architektúru, ktorej procesory produkuje minimum výrobcov. Ako prvým vhodným kandidátom sa javila architektúra ARM podrobne popísaná v [11]. Spoločnosť ARM vydáva licencie pre širokú škálu procesorov od vysoko výkonných, určených pre tablety a inteligentné telefóny až po jednoduché mikroprocesory s nízkou spotrebou určené na použitie v jednoduchých aplikáciách ako práčky, diaľkové ovládače a pod. (napríklad jadro Cortex M0). Vyššia rada procesorov ARM ako ARMv9 alebo ARMv11 by z programátorského hľadiska bola vhodnejšia z dôvodu, že tieto procesory obsahujú MMU (memory management unit), ktorá je nevyhnutná pre použitie moderných operačných systémov ako napríklad Linux. Pri použití tohto riešenia sa priam ponúka použiť linuxový kernel a real time preemptive patch, pre možnosti aplikácie reálneho času. Žiaľ v mnohých prípadoch MMU tvorí až 30% spotreby procesora, čo si pri odhadovanom príkone zo solárnych panelov 0,81W za jeden obeh Zeme nemôžeme dovoliť. Preto padlo rozhodnutie vybrať jednoduchší mikroprocesor bez MMU a naprogramovať vlastný plánovač reálneho času. Tu sa ponúkla voľba nízkoenergetických procesorov Cortex M3, pripadne M4 alebo M0. Ich veľkou výhodou je 32-bitová architektúra spolu s 32-bitovou aritmeticko-logickou jednotkou. Pri jadre M4 aj FPU (floating point unit) pre výpočty s plávajúcou desatinnou čiarkou. Z pohľadu pomeru spotreby a výpočtového výkonu pre výpočty orientácie a stabilizácie satelitu sú tieto procesory ideálne. Žiaľ ako väčšina moderných procesorov, tieto

procesory sú navrhnuté dynamickou CMOS architektúrou, ktorá je v porovnaní so statickou CMOS architektúrou dva krát menej odolná voči radiácii [12]. Satelit skCube bude na obežnej dráhe, ktorá sa nachádza pod Van Allenovými radiačnými pásmi (obr. 8). Tieto pásy by mali ochrániť komponenty satelitu od najintenzívnejšej radiácie, ale stále bude satelit čeliť vyššej radiácii ako na Zemi, čo dokázali aj naše merania pomocou stratosférických sond Julo.



Obr. 8. Van Allenove pásy namerané sondami Pioneer 1 a 3 [13]

Vieme, že napríklad kolegovia z Estónska použili procesory architektúry Cortex M3, konkrétne STM32 F103 pri návrhu svojho palubného počítača a po roku vo vesmíre im až na pár náhodných resetov funguje tento procesor spoľahlivo. Pre nás však táto informácia o nižšej radiačnej odolnosti predstavovala potenciálne bezpečnostné riziko, preto sme sa nakoniec rozhodli hľadať ďalšiu alternatívu. Pri ďalšom hľadaní vhodného kandidáta sme narazili na architektúru MSP430 od firmy Texas Instruments. Táto architektúra ponúka 16bitové jadro s veľmi nízkou spotrebou, množstvo periférií a zaujímavých vlastností ako napríklad asynchrónne čítače. Keď sa pozrieme na ich odolnosť voči vplyvom vesmírneho prostredia a hlavne voči kozmickej radiácii zistíme, že podľa meraní ESA (European Space Agency) [14] procesory z rodiny MSP430 vydržali bez väčších problémov dávky radiácie 40 Krad. Jediným parametrom, ktorý vykazoval neštandardné hodnoty bol zvýšený odber prúdu mikroprocesora. Toto ale nepovažujeme za kritický problém pre našu misiu. Meranie vplyvu radiácie na jednotlivé kritické komponenty palubného počítača sme realizovali aj vlastným meraním. Konkrétne sme z rodiny MSP430 ako riadiaci mikroprocesor pre palubný počítač zvolili MSP430F5438A-EP. Jedná sa o verziu určenú pre vojenský a vesmírny sektor, ktorá podlieha

prísnejším výrobným testom a väčšina chýb z dokumentov Errata ostatných procesorov danej rodiny je u nich opravená. V prvej fáze návrhu bol zvažovaný aj mikroprocesor z rodiny MSP430FR ktorý miesto FLASH pamäte pre program obsahuje feritovú FRAM pamäť, ktorá je podstatne viac odolná voči radiácii. U pamätí typu FLASH dochádza vplyvom radiácie často k porušeniu nábojovej pumpy, pomocou ktorej sa vykonáva samotné prepálenie pamäťových buniek, pričom FRAM pamäť pracuje na princípe RAM a k tomuto problému u nej nedochádza. Žiaľ procesory MSP430FR, ktoré túto FRAM obsahujú sú dostupné s maximálnou kapacitou SRAM iba 2KB, ktorá by pre potreby stabilizačného algoritmu, algoritmu na určenie orientácie a operačného systému bola nedostačujúca.

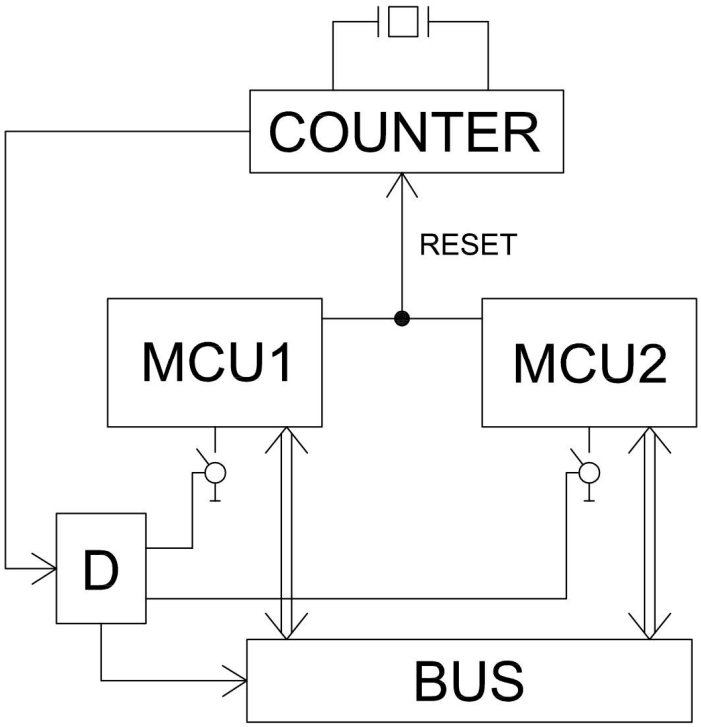
Návrh palubného počítača

Ako bolo už v úvode do tejto kapitoly spomenuté, riadiaci palubný počítač predstavuje kritický komponent satelitu, ktorý je pre jeho správnu funkčnosť spoločne s rádiom najdôležitejší. Tomuto faktu bol prispôsobený aj návrh hardvéru palubného počítača, pri ktorom bol kladený dôraz na čo najväčšiu elimináciu potenciálnych chýb a defektov, ktoré by mohli vo svojom dôsledku znefunkčniť palubný počítač.

Prvým a asi najhlavnejším rozhodnutím bolo použitie dvoch procesorov MSP430F5438A-EP miesto jedného. Tieto procesory sú zapojené v takzvanom režime „Cold Swap“, pri ktorom je aktívny (zapnutý) len jeden z týchto procesorov a v prípade akéhokoľvek problému či už softvérového alebo hardvérového supervízor, ktorý kontroluje tieto procesory vykoná ich prepnutie. Pri prepnutí procesorov sa aktívny procesor, ktorý spôsobil svojou chybou akciu supervízora odpojí od napájania a teda vypne, pričom doteraz vypnutý procesor sa pripojí na napájanie a naštartuje sa. Pri svojom štarte zapíše túto udalosť do systémového logu, pre ďalšiu možnú analýzu. Pri samotnom prepnutí procesora sú prepnuté aj všetky riadiace zbernice k tomuto procesoru pomocou trojstavového prepínača. Takto je schopný záložný procesor plnohodnotne ovládať všetky zariadenia palubného počítača. Samozrejmosťou je, že v prípade ďalšej chyby, alebo poruchy vieme takto druhý procesor prepnúť späť na hlavný. Jediným rozdielom v zapojení týchto procesorov je jeden ich digitálny IO pin, ktorý je pomocou odporu pripojený na logickú „1“ respektíve „0“. Takto vieme v programe identifikovať na ktorom fyzickom procesore systém práve pracuje. Blokovú schému zapojenia palubného počítača možno nájsť na obrázku č. 12.

Z vyššie uvedenej logiky prepínania vyplýva, že síce sme zvýšili odolnosť procesoru voči radiácii a silne nabitým časticiam jeho zdvojením (redundanciou), ale zároveň sme do systému zaviedli nový kritický prvok, ktorým je supervízor a prepínač zberníc. Tieto prvky sme sa rozhodli zostaviť nie z komerčne dostupných ale radiačne tvrdených súčiastok. K tomuto rozhodnutiu nás viedol aj ekonomický fakt, pretože tieto komponenty sú elektronicky podstatne jednoduchšie a tým pádom aj lacnejšie v porovnaní s radiačne odolným procesorom. Podľa ústnych informácii od jedného

dodávateľa je cena jednoduchého základného 8-bitového procesora rodiny 8051, ktorý je odolný voči radiácii 8 až 10 tisíc eur, čo je pre náš projekt neobhájiteľná suma.



Obr. č. 12 Bloková schéma palubného počítača

Samotný supervízor pozostáva z dvoch radiačne tvrdených súčiastok. Jedná sa o počítadlo (counter) a D klopný logický obvod. Počítadlo je taktované vlastným zdrojom frekvencie ktorý je vytvorený pomocou RC člena. Pri dosiahnutí limitnej hodnoty počítadla nastanie pretečenie, ktoré vygeneruje impulz, ktorý následne vstupuje do D klopného obvodu. Po pretečení sa počítadlo vynuluje a celý proces začína odznova.

Zopnutie D klopného obvodu má za následok prepnutie napájania jednotlivých procesorov a zmenu prepnutia zberníc pomocou trojstavových prepínačov. Tento D obvod môžeme chápať aj ako výhybku, ktorá pri vstupnom signále automaticky predáva riadenie vypnutému procesoru.

Samotné počítadlo musí byť periodicky nulované predtým ako nastane jeho pretečenie. Toto vynulovanie zabezpečí aktívny procesor, zmenou logickej hodnoty na príslušnom výstupnom pine z log „0“ na log „1“. Táto akcia nulovania počítadla nesmie byť zakomponovaná v riadiacom softvéri ako prerušenie, ale musí byť volaná priamo z programového kódu. Týmto zabezpečíme detekciu chyby priamo v čipe riadiaceho procesora, alebo softvérovej chyby, ktorá mohla byť spôsobená či už programátorom, alebo tzv. „single upset event-om“, ktorý môže zapríčiniť napríklad zmena bitu v pamäti vplyvom radiácie.

MEMS senzory – gyroskop, akcelerometer a magnetometer má každý procesor vlastné. Pre misiu sa nejedná o kritické súčiastky a z dôvodu jednoduchšieho návrhu dosky plošných spojov neboli tieto senzory umiestnené na zbernicový prepínač. Ďalšie senzory pre určenie orientácie ako senzory slnka a detektor horizontu Zeme ako aj komunikačné linky k ostatným subsystémom satelitu sú pripojené cez zbernicový prepínač.

Samotná doska palubného počítača obsahuje ešte termistory pripojené na analógovo digitálny prevodník určené na meranie vnútornej teploty v satelite a elektroniku pre riadenie prúdu do cievok. Táto elektronika je riadená vstupným PWM signálom (pulzovo šírkovo modulovaný signál), ktorý ovláda H-mostík. Takéto signály a mostíky sú dokopy tri, každý pre cievku v jednej osi *x*, *y*, *z*. Toto zapojenie umožňuje pomocou jedného pinu riadiť smer prúdu ktorý preteká cievku a pomocou PWM signálu riadiť jeho intenzitu.

Pre potreby konfigurácie a kontroly stavu satelitu po osadení do vypúštacieho zariadenia P-POD musel mať palubný počítač vytvorený servisný konektor. V našom prípade sme ako servisný konektor zvolili konektor typu microUSB ktorý bude na jednej strane satelitu, ktorá je pri inštalácii satelitu otočená smerom k servisnému technikovi. Konektor microUSB bol zvolený preto, že má malé rozmery, teda jeho umiestnenie na stenu satelitu nezaberie veľa dôležitého miesta, ktoré je potrebné pre solárne panely. Ako druhý dôvod tejto voľby bol fakt, že USB káble pozostávajú zo štyroch vodičov. Dva sú komunikačné – Rx a Tx (čítanie/zápis) a cez ďalšie dva sa privádza napájanie a zem. Toto nám umožní cez jeden konektor previesť napájanie aj komunikáciu, čo je mimoriadne výhodné, keďže podľa noriem majú byť batérie pred štartom satelitu vybité a všetky systémy bez trvalého prísunu energie. Samotná komunikácia so satelitom bude prebiehať po UART linke v textovej podobe, pričom v rámci USB servisného kábla bude použitý prevodník FT232, ktorý na strane počítača emuluje sériový COM port a prevádza potrebné signály do TTL logiky.

Pri pripojení napájania k palubnému počítaču a jeho samotnému štartu nie je dopredu dané ktorý z dvoch použitých procesorov sa zapne ako prvý. Ich „voľba“ je čisto náhodná. Preto aj servisný UART port oboch procesorov je privedený na automatický prepínač zberníc. V prípade ak by sme chceli overiť aj správnu funkčnosť druhého procesora, musíme softvérovým príkazom vyvolať ich zmenu a reštart celého systému.

.

Operačný systém reálneho času

V histórii počítačov a operačných systémov nájdeme niekoľko možných riešení ako zabezpečiť beh programu v reálnom čase. Najskôr by sme si mali ale priblížiť čo ten reálny čas vlastne znamená. Pri klasických operačných systémoch ako napríklad Linux, alebo Windows sa po vzniknutí udalosti (napríklad kliknutie myšou na ikonu) vykoná patričná reakcia (otvorenie okna programu) vtedy, keď príde na obslúženie danej udalosti rada. V istých prípadoch vzniknuté oneskorenie ani nepostrehneme a niekedy, hlavne pri veľmi vyťažených systémoch môže byť odozva niekoľko stoviek ba až tisícok milisekúnd. Toto oneskorenie sa vo všeobecnosti nazýva Jitter. Úlohou operačného systému reálneho času (RTOS) je zabezpečiť, aby odozva na udalosť nebola dlhšia ako požadovaný Jitter. V závislosti od plnenia podmienky aby nebol prekročený Jitter delíme operačné systémy na takzvané tvrdé (hard) a mäkké (soft). Pri mäkkých RTOS občasné nesplnenie časovej podmienky (Jitter) nepredstavuje problém pre danú aplikáciu. Jedná sa hlavne o systémy na prúdový prenos videa a podobne, kde vypadnutie jedného obrázku nespôsobí pád aplikácie, bezpečnostné riziko, alebo výrazne nezníži užívateľský komfort. Naopak pri tvrdých RTOS sa vyžaduje bezpodmienečné splnenie časovej podmienky (Jitter) za každých okolností. Ide napríklad o brzdové systémy, letecké a robotické systémy a podobne, kde by čo len jedno nesplnenie termínu na odozvu mohlo mať fatálne následky.

Od minulosti až po súčasnosť prešli RTOS systémy veľa zmenami. Prvé hard RTOS systémy pracovali na princípe takzvanej „All mighty main funciton“ [19, 20]. V takomto prístupe boli všetky úlohy zakomponované do jednej veľkej main funkcie programu, ktorá sa vykonávala počas behu systému stále dookola. Matematicky bola potom spočítaná najdlhšia odozva jednotlivých úloh v slučke a tým pádom bolo aj dokázané splnenie Jitteru za každých podmienok. Neskôr prišli do procesorovej architektúry prerušenia a jednotlivé úlohy reálneho času mohli byť volané na základe týchto prerušení. Toto riešenie však pri komplexnejších systémoch predstavovalo rôzne úskalia. Najnebezpečnejším bolo to, že pokiaľ úloha obsluhovala periférium procesora a bola v tom momente prerušená prerušením od úlohy reálneho času, mohla túto periférium odovzdať systému v nedefinovanom (nežiaducom) stave. Preto bolo potrebné prísť so systémom ktorý by tieto stavy a prepínanie úloh strážil.

Ako sa systémy vyvíjali a stávali sa čoraz komplexnejšie, takáto jedna programová slučka začala byť neprehľadná a vývoj takýchto systémov bol programátorsky nekomfortný. Preto prišli takzvané real time executive (RTE) kernely [19], ktoré sa kompilovali spoločne s jednotlivými úlohami (funkciami) a ponúkali základné funkcie, ktoré poznáme z moderných operačných systémov ako plánovač, blokovacie mechanizmy a pod.

V moderných a komplexných systémoch potom prišli veľké operačné systémy, ako Unix, Dos, prípadne Windows, ktoré obsahovali všetky vlastnosti štandardných edícií týchto operačných systémov. Navyše od nich obsahovali vložku (alebo patch, prípadne celé modifikované jadro), ktoré umožňovalo okrem behu štandardných programov aj beh programov reálneho času.

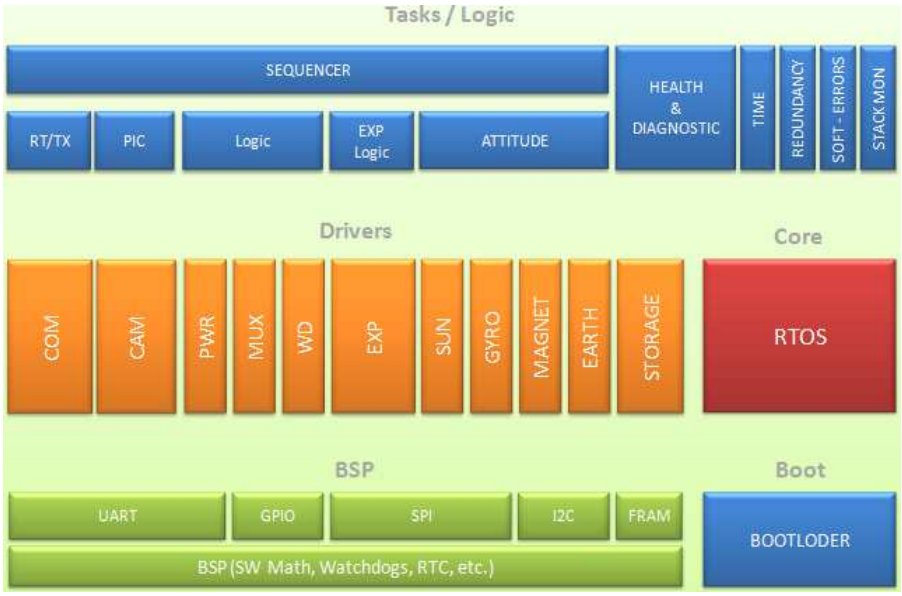
Na nami navrhnutý hardvér palubného počítača môžeme uvažovať kvôli komplexnosti a pamäťovým nárokom iba systémy na báze „All mighty main function“, alebo na báze jednoduchých kernelov ktoré sa skompilujú spoločne s jednotlivými úlohami. Preto ako prvý krok sme sformulovali naše požiadavky ktoré na operačný systém palubného počítača máme:

- OS má zaberať čo najmenej pamäte RAM a FLASH
- Pre logiku a funkčnosť našich aplikácii stačia jednoduché blokovacie a medzi procesové komunikačné mechanizmy
- OS má spĺňať štandardy pre bezpečný softvér MISRA C
- OS má mať v sebe implementované mechanizmy na detekciu a opravu soft errors ako napríklad bit flip (prepnutie logickej hodnoty bitu vplyvom radiácie)
- OS by mal byť ľahko portovatelný na rôzne hardvérové platformy
- Plánovač operačného systému by mal mať ochranný mechanizmus proti vzniku soft errors
- OS musí byť voľne dostupný

Pri pohľade na dostupné RTOS systémy na trhu sme prišli k záveru, že žiaden z bežne dostupných systémov nevyhovuje našim požiadavkám. Žiaden z voľne dostupných RTOS systémov nespĺňal požiadavku na splnenie MISRA C [21] štandardu a u väčšiny RTOS sme nenašli žiadnu ochranu proti soft errors. Samozrejme sme uvažovali aj možnosť upustiť od našej prísnej bezpečnostnej politiky a použiť RTOS ktorý nespĺňa bezpečnostné štandardy. Žiaľ tu sme narazili na ďalší problém a to je pamäťová náročnosť a celková komplexnosť jednotlivých systémov. Napríklad nami navrhnutý RTOS oproti bežne používanému FreeRTOS v najzredukovanejšej verzii spotrebuje jednu štvrtinu pamäte FLASH – 1KB proti 4KB u FreeRTOS [22]. Z týchto dôvodov sme sa nakoniec rozhodli pre návrh a vývoj vlastného operačného systému. Jednou z hlavných výhod tohto rozhodnutia je okrem získaných skúseností aj dokonalá znalosť interných mechanizmov RTOS systému, ktorých pochopenie v prípade prebraného riešenia by nám zabralo dosť vývojového času.

Návrh RTOS systému

Pri návrhu nášho operačného systému reálneho času sme sa najskôr zamerali na jeho jednoduché portovanie na rôzne hardvérové platformy. Pre tento účel sme zvolili rozdelenie OS do troch úrovní ktoré medzi sebou spolupracujú a tieto jednotlivé úrovne sa dajú bez ovplyvnenia druhých jednoducho preprogramovať. Samotné úrovne a model operačného systému je znázornený na obrázku č. 16 a jeho funkčnosť si je možné analogicky predstaviť podobne ako pri OSI modeli v problematike počítačových sieti.



Obr. 16. Úrovňový model operačného systému

Na najnižšej úrovni sú zobrazené hardvérovo závislé časti programového kódu. Jedná sa o board support package BSP, ktorý zahŕňa nastavenie a prácu s jednotlivými rozhraniami procesora a o zavádzač (bootloader), ktorý má za úlohu naštartovať a inicializovať celý systém. Pri portovaní nami navrhnutého operačného systému je potrebné prispôsobiť túto vrstvu novej hardvérovej platforme.

Druhá stredná vrstva obsahuje programový kód, do ktorého priamo užívateľ alebo programátor nemusí zasahovať a bude pracovať s pomocou nižšej vrstvy. Nachádza sa tu jadro operačného systému a jednotlivé ovládače zariadení s ktorými palubný počítač pracuje. V našom prípade sú to hlavne senzory, driver na cievky a UART komunikačné zbernice. V prípade ak by programátor chcel využívať nové iné senzory, treba pre ne vytvoriť nový ovládač. Posledná najvyššia vrstva obsahuje užívateľom vytvorené programy a úlohy. Ako aj nami vytvorené nástroje na monitorovanie funkčnosti OS, spracovanie príkazov zo zeme, monitorovanie stavu a zdravia satelitu a tak podobne.

Samotný programový kód všetkých vrstiev operačného systému je naprogramovaný v ansi C pre jeho jednoduchšiu prenositeľnosť na iné platformy. Túto vlastnosť sme overili tým, že sme behom

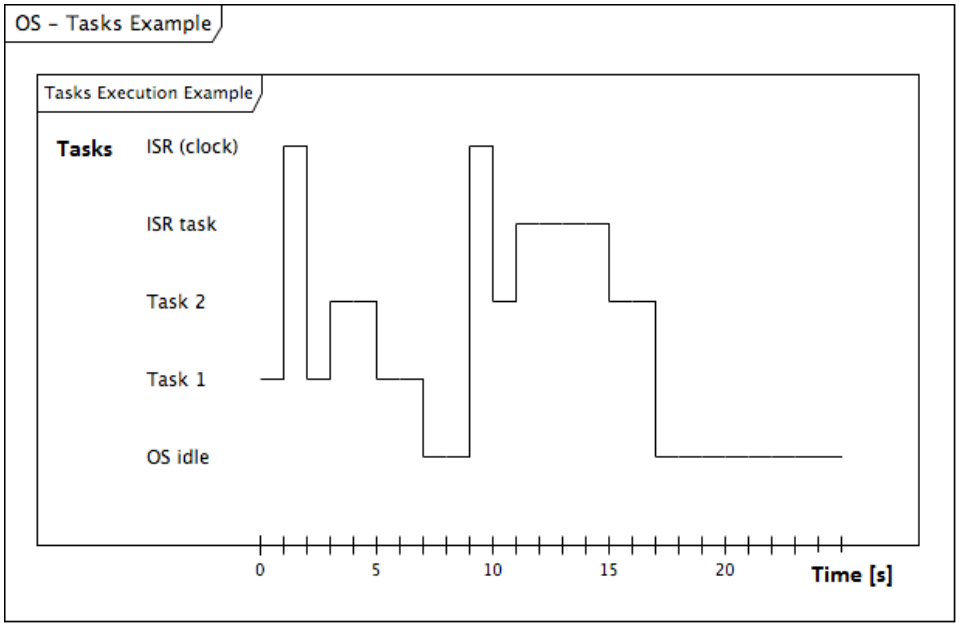
jedného dňa preniesli operačný systém na prvú verziu testovacej dosky s ARMv7 procesorom a takisto sme boli schopný spustiť operačný systém na PC a MAC počítačoch pod operačným systémom Unix.

Všetok softvér a všetky časti operačného systému plne spĺňajú bezpečnostné štandardy vyvinuté pre automobilový priemysel MISRA C (Motor Industry Software Reliability Association). Pre overenie splnenia týchto kritérií sme použili statický programový analyzátor PC Lint od spoločnosti Gimpel.

Pri návrhu logiky a prepínacieho algoritmu operačného systému sme využívali simulátor ktorý sme navrhli v prostredí Simulink programu Matlab. Pri návrhu logiky sme zvažovali dva možné prístupy k prepínaniu jednotlivých úloh a to kooperatívny alebo preemptívny multitasking. Pri preemptívnom multitaskingu sa úlohy delia podľa svojej priority rovnomerne o procesorový čas, pričom samotnú úlohu preruší svojvoľne plánovač a v prípade potreby predá procesorový čas inej úlohe. Takéto prepínanie predstavuje férovejší prístup k predávaniu procesorového času a zabezpečuje, že sa úloha s vyššou prioritou dostane k vykonaniu najneskôr za periódu času s ktorou je volaný plánovač. Nevýhodou tohto systému je jeho zložitosť aj fakt, že každá úloha musí mať svoj vlastný pamäťový zásobník. Pri kooperatívnom multitaskingu ide o podstatne jednoduchšiu schému predávania procesorového času. Môžeme si ju predstaviť tak, že práve vykonávaná úloha sa sama rozhodne kedy ponúkne procesor inej úlohe s vyššou prioritou. Ak v momente tohto ponúknutia žiadna úloha s vyššou prioritou nečaká na procesor pôvodná úloha pokračuje ďalej vo svojom vykonávaní. Keď sa vykonávaná úloha skončí, predá znova riadenie plánovaču a ten určí ďalšiu úlohu s najvyššou prioritou. Pre dodržanie integrity zásobníka je potrebné, aby boli všetky úlohy s vyššou prioritou vykonané predtým ako sa začne vykonávať úloha s nižšou prioritou.

Tento typ prepínania je podstatne jednoduchší z už spomenutého dôvodu, že programátor sám určí kedy môže byť jeho úloha prepnutá a kedy nie. Ďalším zjednodušením je, že jednotlivé úlohy môžu zdieľať spoločný zásobník. Takisto sa pri kontrolovanom predávaní procesora jednoduchšie spočíta maximálny čas ktorý aplikácia s vyššou prioritou bude čakať. Jednou vecou na ktorú si však pri takomto plánovaní treba dať pozor je dostatočne časté volanie plánovača v tele jednotlivých úloh, lebo naopak by potom takéto plánovanie strácalo svoj význam.

Pre svoju jednoduchosť a aj menšiu pamäťovú náročnosť sme zvolili kooperatívny režim prepínania. Funkčnosť tohto režimu spolu s vyťažením zásobníka vysvetľuje nasledujúci obrázok.



Obr. 17. Príklad kooperatívneho multitaskingu

Kde v čase:

0: Task 1 sa práve vykonáva.

1: Vznikne prerušenie od hodín. OS zmení stav pre Task 2 na ready.

2: Task 1 pokračuje vo svojom vykonávaní.

3: Task 1 dobrovoľne "predá" procesor pre Task 2 (ktorý bol nedávno nastavený do stavu ready).

5: Task 2 ukončí svoje vykonávanie a zásobník sa znova rozbalí do stavu počas vykonávania Task 1.

7: Task 1 sa ukončí. Systém sa prepne do režimu IDLE.

9: Vznikne prerušenie od hodín. Task 2 má zmenený stav na "ready".

10: Keďže sa žiadna úloha nevykonáva, Task 2 sa začne bezodkladne vykonávať.

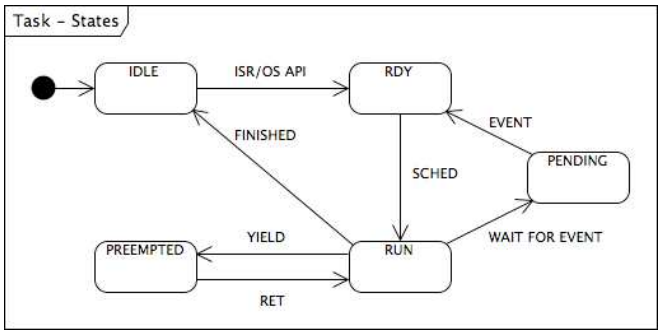
11: Task 2 je prepnutý úlohou ISR task, ktorá predstavuje krátku tvrdú úlohu reálneho času. Tieto úlohy majú svoj vlastný zásobník, preto systémový zásobník s úlohou Task 2 nie je porušený.

15: "ISR task" ukončí svoje vykonávanie a procesor je predaný pre Task 2. Aj keď Task 2 ponúkne procesor inej úlohe, bude pokračovať vo svojom vykonávaní, keďže sa jedná o úlohu s najvyššou prioritou.

17: Task 2 ukončí svoje vykonávanie. Systém sa prepne do režimu IDLE.

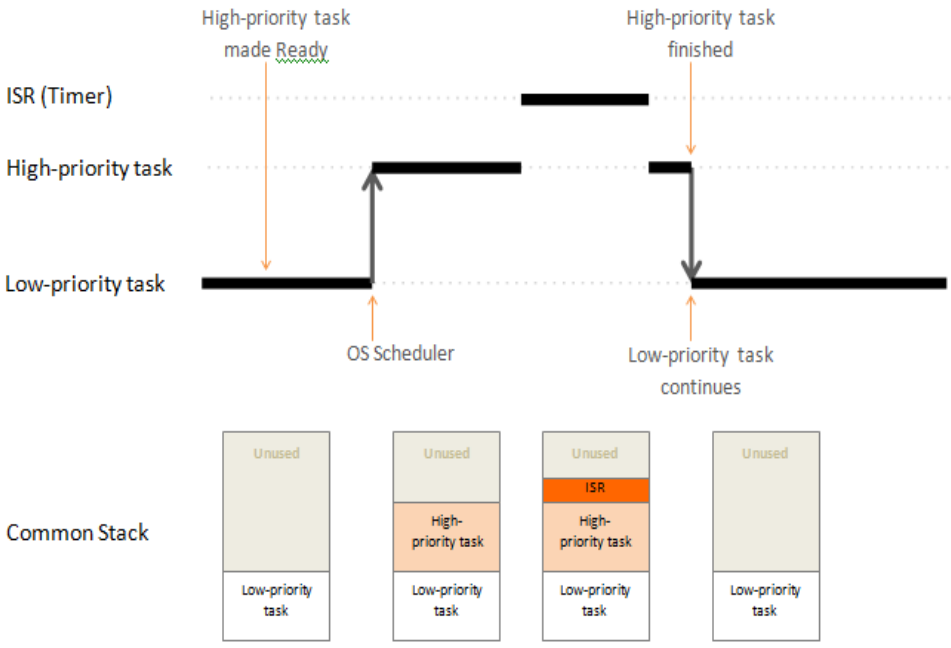
Pri vytvorení novej úlohy je táto úloha automaticky v stave IDLE. Zavolanie API funkcie operačného systému pomocou prerušenia, alebo v programe užívateľa môže tento stav zmeniť na READY. Tento stav umiestni úlohu do fronty úloh, kde sú tieto úlohy triedené podľa ich priority.

V momente ako je zavolaný plánovač, tak vyberie úlohu s najvyššou prioritou a odovzdá jej riadenie procesora. Ďalším zo stavov v ktorom sa môže úloha nachádzať je ten, kedy čaká na dáku udalosť (Pending on Event), napríklad na príznak o dokončení merania. V momente keď sa táto udalosť naplní, tak sa stav danej úlohy zmení opäť na READY a úloha sa zaradí do fronty. Podrobný vzťah medzi jednotlivými stavmi možno vidieť na diagrame v obrázku č. 18.



Obr. 18. Možné stavy úlohy

Na priblíženie kooperatívneho prepínania štandardných úloh so spoločným zásobníkom slúži graf č. 19. Všetky úlohy reálneho času ktoré sú ovládané pomocou prerušenia majú svoj vlastný zásobník a teda v prípade ak preberú riadenie počas vykonávania štandardnej úlohy nedôjde k porušeniu zdieľaného zásobníka.

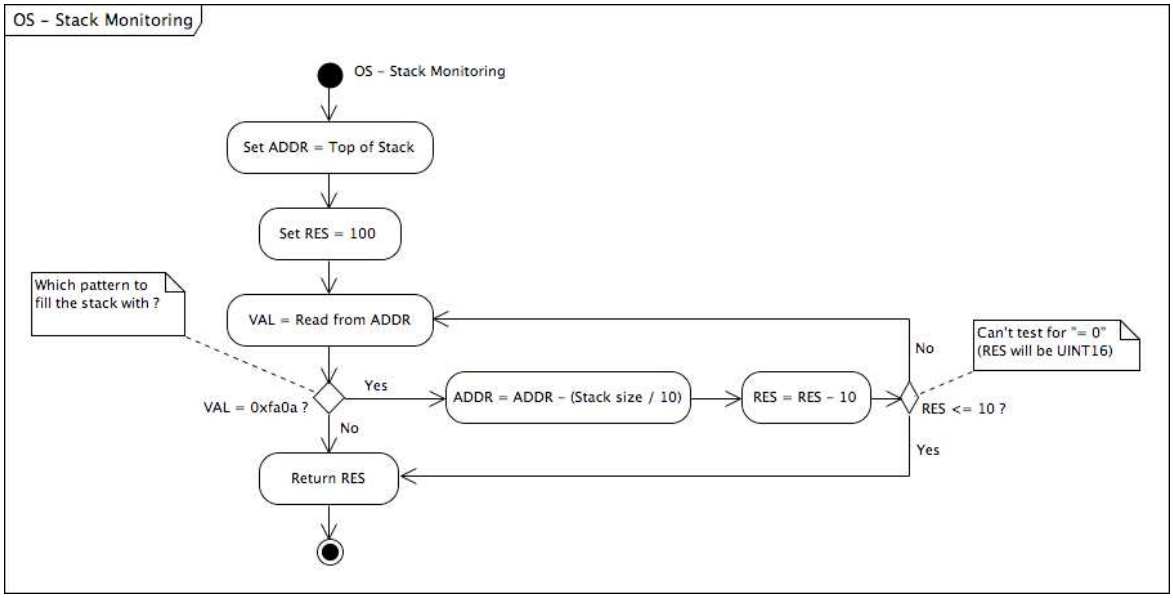


Obr. 19. Správanie sa zásobníka pri kooperatívnom multitaskingu

Najskôr vidíme na spodku zásobníku iba úlohu s nízkou prioritou. Hneď ako sa začne vykonávať úloha s vyššou prioritou (a neskôr úloha reálneho času) sú jej údaje umiestnené vyššie v zásobníku nad údajmi úlohy s nižšou prioritou. V tomto momente sa nemôže začať vykonávať úloha s najnižšou prioritou kým sa prirodzene nevyprázdni zásobník nad ňou tým, že sa ukončia úlohy s vyššou prioritou umiestnené nad ňou. Preto sa tieto systémy občas nazývajú aj run to complete. V prípade ak by sa táto úloha s nižšou prioritou vykonala, zásobník by sa stal nekonzistentným a nastal by pád celého systému. Tým, že celý programový kód dodržiava prísne požiadavky štandardu Misra C, tak v celom operačnom systéme sa nachádza iba pár smerníkov a všetky sú statické – teda ich hodnota je vypočítaná v čase kompilácie zdrojového kódu. Keďže nemôžeme dynamicky alokovať pamäť celý zásobník na jednotlivé úlohy musí byť statický s dopredu vypočítanými hranicami na základe predpokladaného vyťaženia systému úlohami. Ďalší z problémov, ktorý môže nastať práve v tomto prípade je to, že v prípade veľkého počtu prepnutých úloh v zásobníku môže tento zásobník narásť natoľko že pretečie.

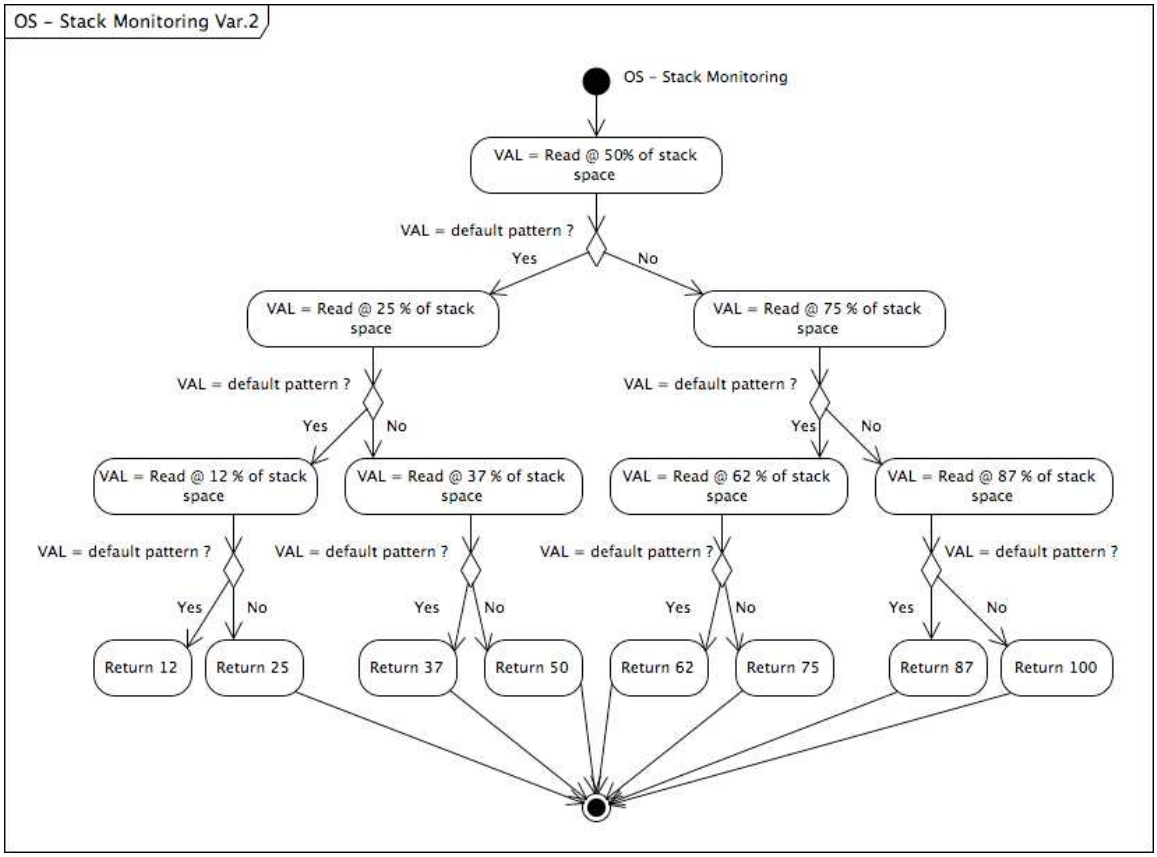
Na to aby sme tomuto problému zabránili musíme vyťaženie zásobníka pravidelne monitorovať. Zásobník monitorujeme jednoduchým algoritmom, kedy na začiatku pred vykonávaním úloh naplníme zásobník jednoduchým jednotným vzorom 0xA5. Potom pomocou funkcie hľadáme pokiaľ medzi hranicami zásobníka je táto hodnota prepísaná na inú. Informácia o polohe takto prepísaného bajtu nám prezradí ako je zásobník práve vyťažený.

Za účelom nájdenia približnej polohy tohto bitu sme navrhli dva algoritmy. Presná poloha tohto bajtu pre naše potreby nie je potrebná, lebo jej presné hľadanie by nás stálo zbytočný procesorový čas. Dôležité je vedieť aký približný rozsah zásobníka je obsadený a či sa obsadenosť neblíži ku kritickej hodnote. V prípade dosiahnutia kritickej hodnoty sa táto informácia zapíše do systémového logu a počítač vykoná reštart. Na obrázku č. 20 je vidieť prvý algoritmus na zistenie obsadenosti zásobníka.



Obr. 20. Monitor obsadenia zásobníka 1

Ako vidno, tento algoritmus na svoj výpočet využíva procesorovú aritmetiku, čo v prípade poruchy typu soft error nepredstavuje najvhodnejšie riešenie. Takisto tým, že používa aritmetiku a cyklus, potrebuje väčší procesorový čas na svoje vykonanie. Jeho nespornou výhodou ale je, že po svojej kompilácii spotrebuje iba pár bajtov kódu a teda zaberá málo miesta v pamäti FLASH. Pri druhom algoritme zobrazenom na obrázku č. 21 je možné vidieť iný prístup. Samotný algoritmus používa iba porovnávanie v rozvetvených podmienkach, pričom algoritmus začína v strede zásobníka a postupne miesto kontroly delí dvomi. Tento algoritmus je podstatne rýchlejší, žiaľ jeho nevýhodou je, že zaberá niekoľko desiatok násobne viac bajtov ako algoritmus s aritmetikou.



Obr. 21. Monitor obsadenia zásobníka 2

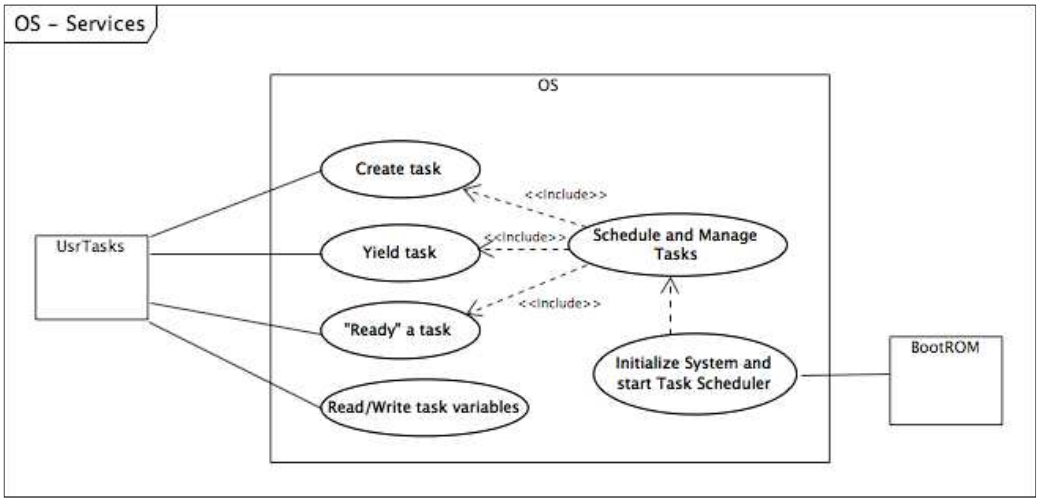
Do operačného systému sme okrem monitorovania vyťaženia zásobníka (stacku) implementovali celý systém na zber diagnostických údajov z ktorých následne generujeme dlhodobú štatistiku vyťaženosti systému. Jednotlivé diagnostické dáta sú znázornené v tabuľke č. 6. Tieto údaje sú periodicky nahrávané do pamäte RAM a po následnom spracovaní sa aktualizuje dlhodobá štatistika v pamäti FLASH, ktorá je potom prenášaná pomocou telemetrie do pozemnej stanice.

Tab.6: Diagnostické údaje

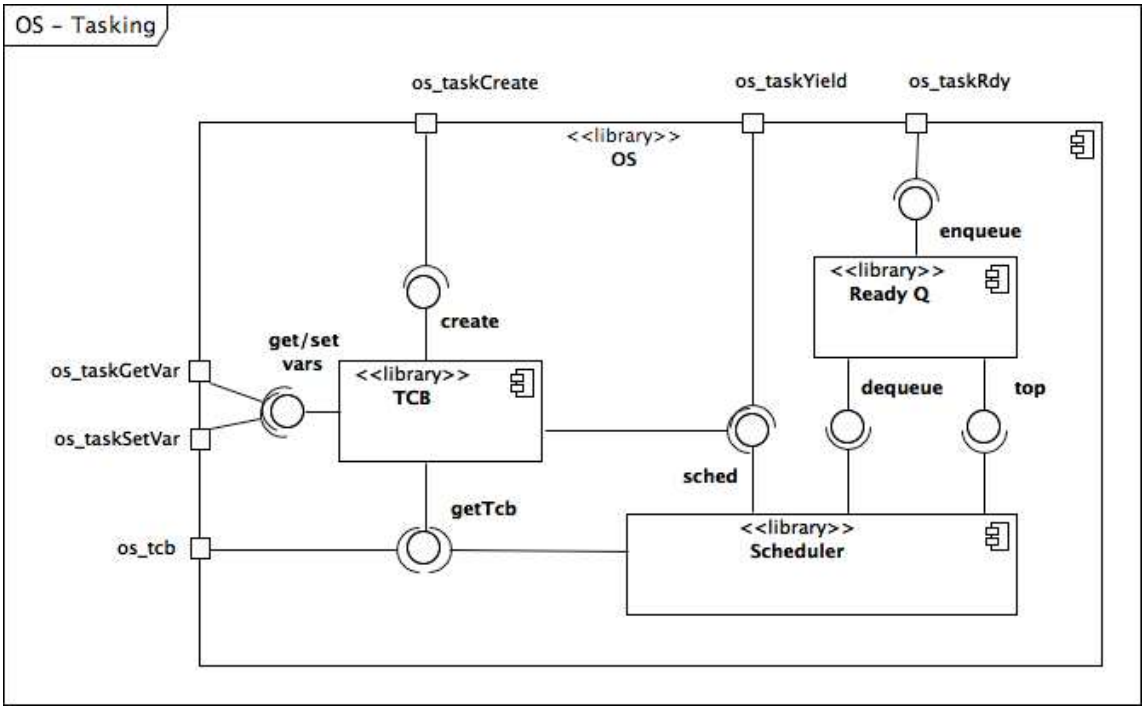
ul			Bytes)
		Max usage	
		e distribution %	
		e variation %	
	duler	Max calls / min	
		s. between calls distrib.	
		ID distrib. / hour	
		Max tasks waiting distrib.	
	y Q	Max usage	
		tasks in Q distribution %	

m	OPS	codes for last executed operations system-wide (for post-mortem analysis)	

Na správu úloh obsahuje operačný systém API (application interface – zobrazené na obrázku č. 22), ktorý umožňuje užívateľovi vytvoriť novú úlohu, ponúknuť procesor druhej úlohe (yield), zmeniť stav úlohy na READY, prípadne čítať alebo zapisovať globálne premenné pre jednotlivé úlohy, ktoré slúžia na medziprocesovú komunikáciu. Prácu s úlohami v operačnom systéme riadia tri hlavné bloky (obrázok č. 23). Blok TCB tvorí register vytvorených úloh. Scheduler (plánovač) vyberie vždy úlohu s najvyššou prioritou zo zásobníku pripravených úloh (Ready Q). Zásobník pripravených úloh (Ready Queue) je FIFO (first in first out) zásobník všetkých úloh, ktoré sú pripravené na spustenie. V tomto zásobníku je okrem iného jediný smerník, ktorý sa používa v celom systéme. Tento smerník ukazuje na funkciu úlohy a je počítaný staticky pri kompilácii zdrojového kódu kompilátorom. Preto nehrozí, že by v prípade nesprávneho výpočtu programátora smerník ukazoval na nesprávne miesto v pamäti. Takisto dynamické a v čase meniace sa smerníky vylučuje štandard Misra C kvôli ich vysokému bezpečnostnému riziku.

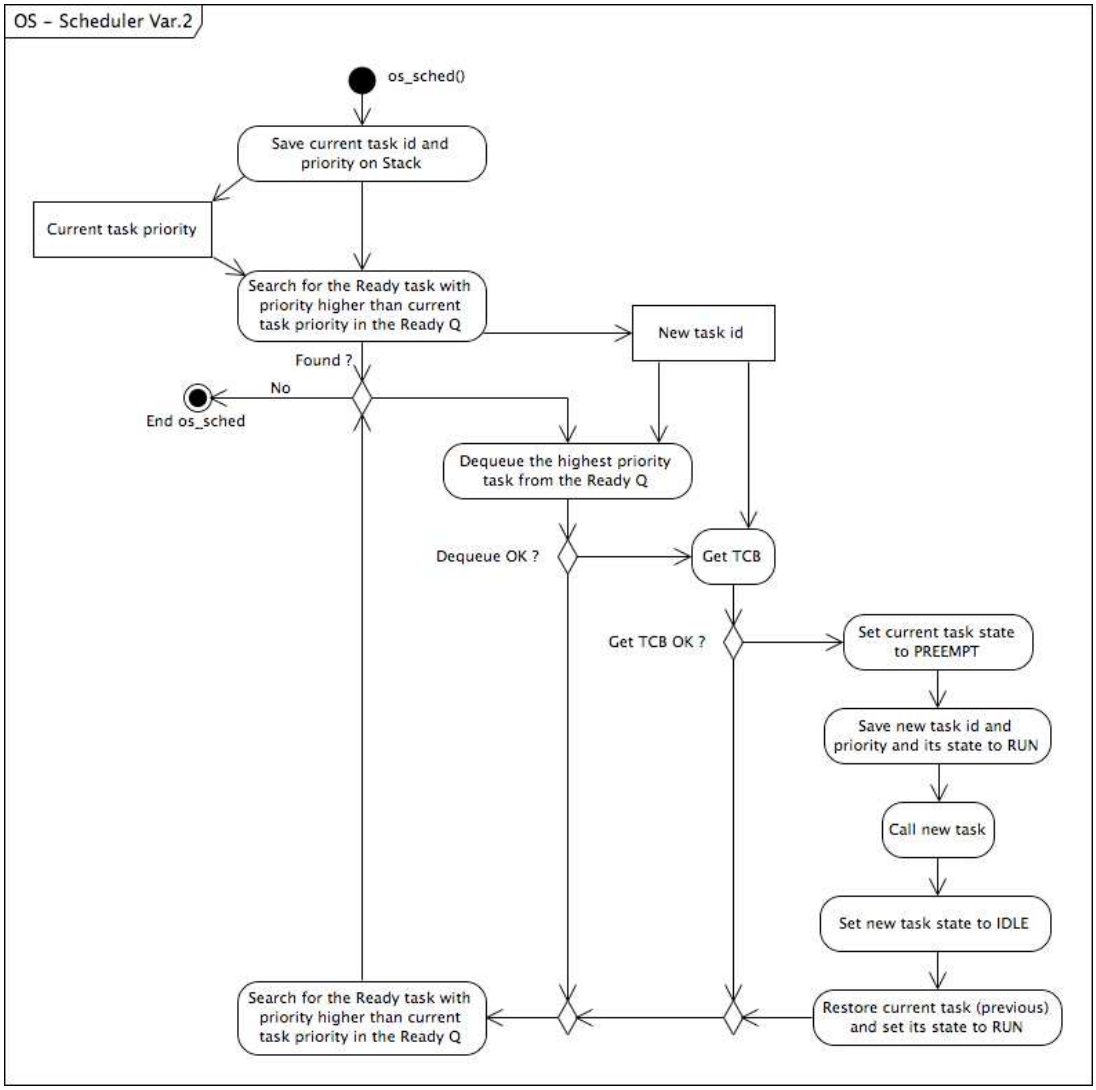


Obr. 22. API operačného systému na prácu s úlohami



Obr. 23. Bloky operačného systému na manažment úloh

Plánovač úloh (task scheduler) je jednoduchá funkcia ktorá sa vykoná pri zavolaní jeden krát bez rekurzie alebo slučky (vo všeobecnosti rekurzie aj tak štandard Misra C zakazuje kvôli možnosti pretečenia zásobníku). Jej úlohou je určiť úlohu s najvyššou prioritou ktorá sa má vykonávať.

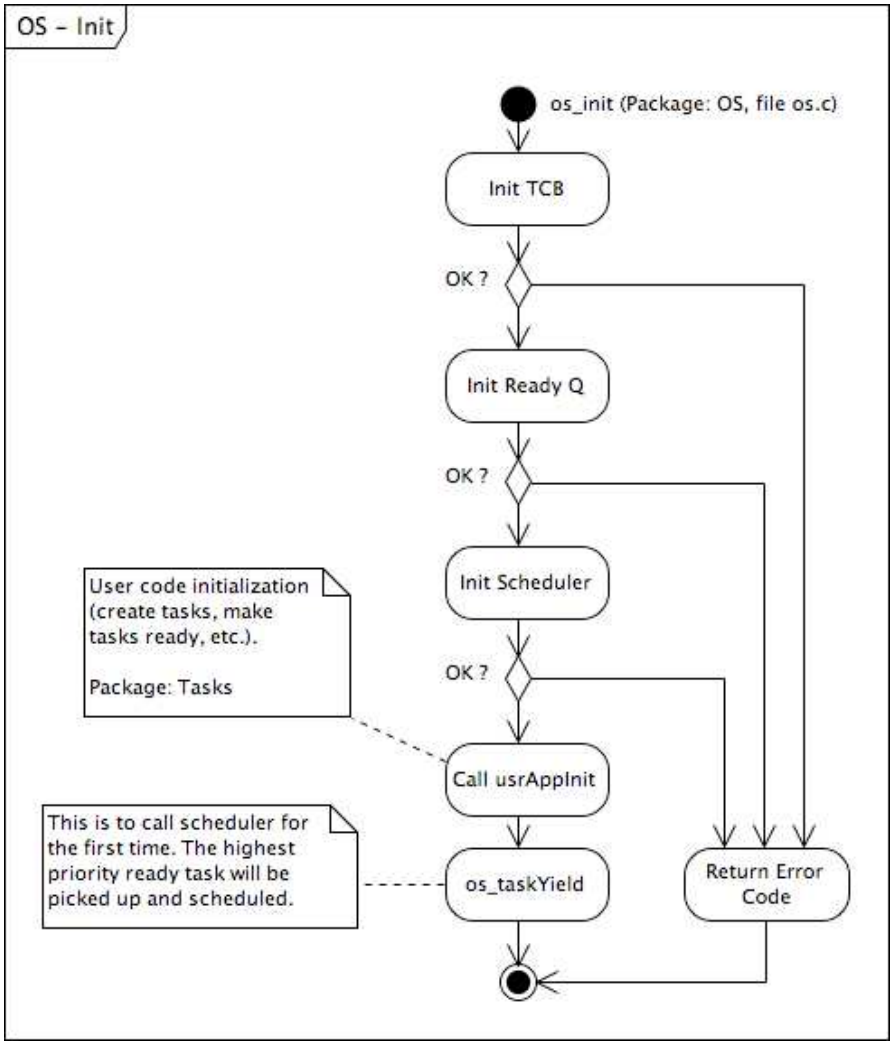


Obr. 24. Algoritmus plánovača úloh

Samotný algoritmus plánovača úloh musí byť atomický, to znamená že počas jeho vykonávania nemôže byť ničím prerušený. Z tohto dôvodu sa pred spustením algoritmu globálne vypnú všetky prerušenia a po jeho ukončení sa následne znova zapnú. Plánovač najskôr skontroluje či v Ready Q nečakajú pripravené úlohy ktoré sú zoradené podľa priority tak, že úloha s najvyššou prioritou sa nachádza na vrchu. Ak sú pripravené úlohy, tak sa porovná priorita najvyššej úlohy v Ready Q s aktuálne bežiacou úlohou. V prípade, že úloha v Ready Q má vyššiu prioritu, vyberie sa zo zásobníku a jej funkcia sa zavolá. Ak by nebola pripravená žiadna úloha s vyššou prioritou tak vo svojom vykonávaní bude pokračovať aktuálna úloha.

Ešte pred tým ako bude plánovač prvýkrát zavolaný systémom, je potrebné aby prebehla inicializácia operačného systému. Inicializácia sa vykoná vždy ako prvá a je spúšťaná zavádzačom. Ako možno vidieť vo vývojovom diagrame na obrázku č. 25 najskôr sa inicializujú tri bloky popísané vyššie TCB, Ready Q a plánovač. Následne sa zavolá vstupná užívateľská funkcia, ktorá musí byť vždy

obsiahnutá v každom obraze systému. V tejto funkcii môže užívateľ vytvoriť nové úlohy, zmeniť im stav na READY a pod. Po skončení tejto funkcie systém zavolá plánovač a celý proces pokračuje spustením pripravenej úlohy s najvyššou prioritou.



Obr. 25. Inicializácia operačného systému

Bezpečnosť operačného systému

Hlavným uplatnením prezentovaného operačného systému majú byť hlavne vesmírne a letecké aplikácie. Z tohto dôvodu je potreba dbať na zvýšenú robustnosť systému a odolnosť voči rôznym náhodným chybám, ktoré môžu nastať.

Z hľadiska softvéru nás trápia hlavne dva druhy chýb. Prvá je programátorská, ktorej sa snažíme vyvarovať pomocou použitia štandardov Misra C a NASA JPL C coding quide [23], prípadne pomocou použitia rôznych statických analyzátorov kódu a unit testov. Druhá chyba, ktorá môže spôsobiť problémy systému, je viacej dynamická a môže nastať v ktoromkoľvek čase. Ide o jav, kedy silne nabitá častica, ktorá napríklad vznikne slnečnou erupciou a smeruje smerom k Zemi, zasiahne pamäťovú bunku nášho palubného počítača. V takomto prípade môže dôjsť k prevráteniu logickej hodnoty jedného alebo viacerých bitov. Tento jav nazývame bit flip. Vplyv tejto chyby vieme výrazne

redukovať použitím rôznych algoritmov. Niektoré techniky má v sebe implementovaný samotný štandard Misra C, ako je napríklad druhá ukončovacia podmienka v cykle v prípade ak by sa z nejakého dôvodu (napríklad aj bit flip) prvá podmienka nenaplnila. Ďalšie algoritmy sme museli navrhnúť a naprogramovať do operačného systému sami.

Pamäť RAM a všetky premenné v nej uložené chráni dvojitý zápis pomocou inverznej formy. Pri tejto technike sa jednotlivé premenné nachádzajú v pamäti dva krát, jeden krát v normálnej a druhý krát v bitovo inverznej forme. Na príklad môžeme uvažovať napríklad číslo 0xA5 ktoré má v bitovej podobe tvar 0b10100101 a v inverznej bitovej forme 0b01011010. V prípade ak nad týmito dvoma formami prevedieme logickú operáciu XOR musíme dostať výsledok 0xFF teda 0b11111111. V prípade, že výsledok bude iný, vieme s istotou prehlásiť, že došlo k porušeniu hodnoty premennej.

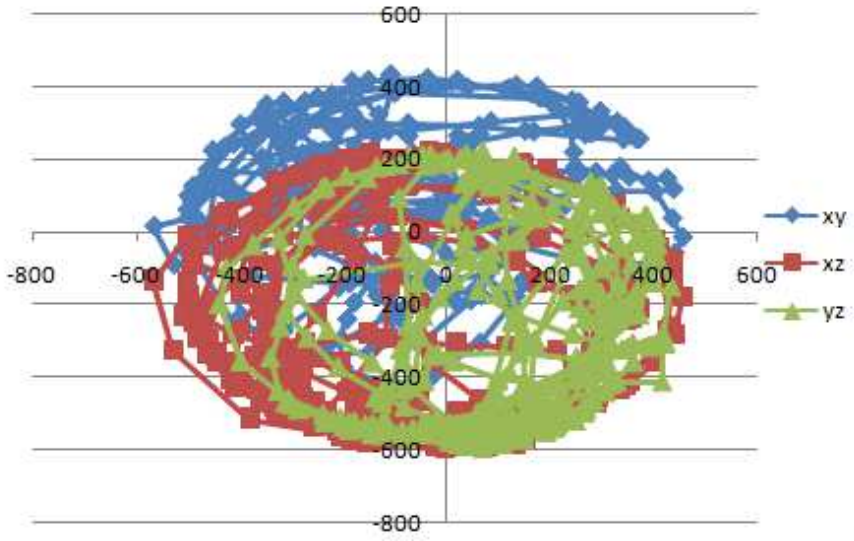
Taká istá situácia môže nastať nie len v užívateľskej pamäti RAM ale aj v registroch procesora a môže pre kritické algoritmy spôsobiť problém. V operačnom systéme sme ako kritický algoritmus vyhodnotili práve plánovač, ktorého úlohou je správne vybrať úlohu na spustenie. Pri riešení ochrany tohto algoritmu sme vychádzali z technológie „lock step“, kedy je jeden a ten istý program s časovým posunom spustený na viacerých procesoroch zároveň a jednotlivé výpočty sa porovnávajú, či sa zhodujú. Obdobu tohto systému využíva napríklad aj novo vyvíjaná vesmírna loď NASA Orion[24], alebo Dragon[25] od spoločnosti SpaceX. V našom prípade nemôžeme použiť dva alebo viac procesorov súčasne. Môžeme však algoritmus plánovača spustiť hneď dva krát po sebe a porovnať jeho výsledok. V prípade, ak by sa tento výsledok nezhodoval, udalosť sa zapíše do systémového logu a nastane reštart systému.

Náhodná zmena hodnoty bitu môže však okrem pamäte RAM nastať aj v pamäti FLASH kde je uložený program s operačným systémom a úlohami. Konzistencia a správnosť tohto programu sa periodicky testuje pomocou výpočtu sumy CRC, ktorej výsledok sa následne porovnáva s uloženou hodnotou. Táto hodnota je znova chránená pomocou inverzného zápisu. V prípade nájdenia chyby sú v palubnom počítači v odolnejších pamätiach typu FRAM (ferite ram) uložené systémové obrazy ktoré sa opätovne nahrajú.

Napriek všetkým prijatým opatreniam a nízkej pravdepodobnosti chyby typu soft error s pravdepodobnosťou približne 1 bit flip za trilión hodín môže nastať stav, kedy by bol program v palubnom počítači nefunkčný. Na riešenie tohto problému sme navrhli špeciálny zavádzač s možnosťou preprogramovania. Pre zvýšenie bezpečnosti sme čiastočne implementovali aj štandardy SAE ARP 4761 (FMEA a FTA analýzu)..

Spracovanie signálov zo senzorov

Niektoré signály zo senozorov satelitu skCube budú potrebovať dodatočné spracovanie, prípadne pred samotným meraním bude nutné vykonať kalibráciu senzora. Signály z niektorých senzorov stačí upraviť pomocou jednoduchých vzorcov, pomocou ktorých sa potlačí napríklad teplotný drift merania. Takáto kompenzácia sa robí napríklad pri detektore horizontu Zeme, pomocou jednoduchého vzorca, ktorý je uvedený v katalógovom liste senzoru [26]. Takéto jednoduché kompenzácie sú aplikované na každý senzor satelitu a nebudú v práci ďalej rozoberané. Zastavíme sa však pri kalibrácii magnetometra, ktorú sme museli navrhnúť z dôvodou, že bežné MEMS magnetometre majú rozdielnu citlivosť v každej z ich osí merania. Najlepšie tento fakt naznačuje graf na obrázku č. 28. Tento graf zobrazuje roviny *xy*, *xz* a *yz* vytvorené z hodnôt nameraných v každej osi magnetometra. Meranie týchto hodnôt prebiehalo tak, že sme magnetometrom náhodne otáčali tak, aby sme magnetometer zorientovali do všetkých možných orientácií a dosiahli tak čo najväčšiu variabilitu meraných údajov.



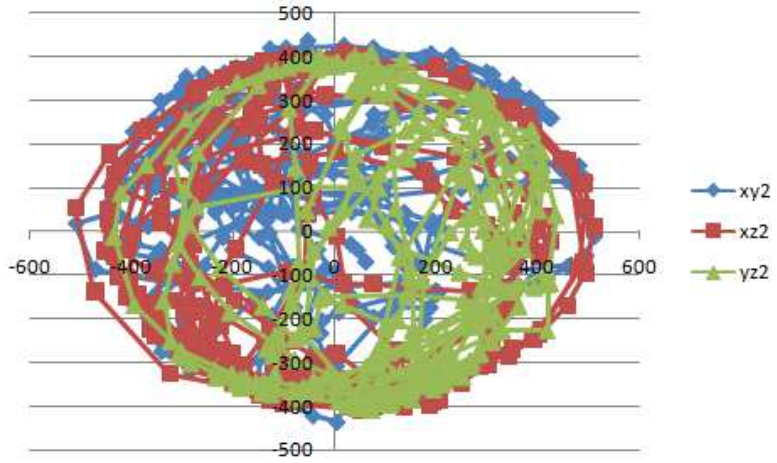
Obr. 28. Meranie magnetometra pred kalibráciou

Ako môžeme vidieť, hranice jednotlivých rovín (zelenej *yz*, modrej *xy* a červenej *xz*) nie sú totožné, ale posunuté, pričom sa jednotlivé roviny neprekrývajú. Tento rozdiel je spôsobený rodielným ofsetom magnetometra pre každú jeho meranú os. Tento ofset vieme eliminovať štatistickou metódou z nameraných údajov grafu č.28 pomocou nasledujúceho vzorca:

$$Offset_x = \min(Data_x) + \frac{\max(Data_x) - \min(Data_x)}{2} \qquad \textbf{(4.1)}$$

Týmto vzorcom, ktorý využíva nájdené maximá a minimá vieme vypočítať offset pre každú jednú os. Takto vypočítaný ofset je potom uložený v pamäti počítača a používa sa počas každého jedného merania magnetometrom. Výsledné meranie magnetometra potom tvorí meraný údaj od ktorého je

odpočítaný offset. Výsledný graf zostrojený z pôvodných meraní posunutých o vypočítaný offset možeme nájsť v obrázku č. 29.



Obr. 29. Meranie magnetometra po kalibrácii

Z grafu vidno, že jednotlivé hranice rovín sú približne totožné a jednotlivé roviny sa prekrývajú. Touto metódou sme kalibrovali každý zo 4-roch magnetometrov na satelite zvlášť a každý magnetometer má svoje vlastné konštanty offsetov.

Z použitých senzorov v satelite skCube sa ako najviac problematický javí práve mems gyroskop. Jedná sa o pomerne citlivú súčiastku (na teplotu, zrýchlenia, a pod.), ktorá trpí veľkým zašumením. Tento šum spôsobujú hlavne vibrácie a zmeny teploty. Takisto všeobecne známym problémom je drift strednej hodnoty gyroskopou, ktorý je prítomný aj pri kľudových podmienok bez pohybu a rotácie gyroskopu a pri konštantnej teplote. Pri získavaní hodnoty otočenia v stupňoch pomocou integrácie uhlovej rýchlosťi gyroskopu by mohla nastať situácia, kedy by sme vypočítali otočenie senzoru kľudne aj o 10-40 stupňov bez toho aby sa senzor vôbec pohol. Takýto naintegrovaný šum treba potlačiť a čo najviac obmedziť. Pri prvom pohľade by sa javil ako vhodný kandidát na potlačenie tohto šumu rozšírený kalmanov filter. Jeho nevýhodou je žiaľ pomerne vysoká výpočtová náročnosť z dôvodu použitých matematických a goniometrických operácií. Takúto vysokú výpočtovú náročnosť by nemusel byť riadiaci počítač schopný realizovať s dostatočne rýchlou periódou. Z tohto dôvodu sme zvolili algoritmus fúzie filtrov, ktorý pre svoje výpočty používa quaternióny a matice v ktorých sa vykonávajú len jednoduché aritmetické operácie.

Testovanie stabilizačného algoritmu

V CubeSate skCube budeme na detumbling využívať algoritmus B-dot, navrhnutý Sicklerom a Alfriendom [30]. Ako názov napovedá jedná sa o algoritmus, ktorý využíva deriváciu magnetického poľa. Takáto technika znižovania rotácie sa dá použiť iba pri nižších orbitách typu LEO, keďže vo väčšej výške je magnetické pole už prakticky nemerateľné [31].

Popisovaný algoritmus využíva ako vstup namerané hodnoty geomagnetického poľa Zeme magnetometrom a jeho výstupom je magnetické pole generované cievkami. Tu vzniká pri meraní geomagnetického poľa, ktoré obklopuje satelit, konflikt, keďže použitie cievok bude ovplyvňovať hodnoty namerané magnetometrom. Kvôli tomuto problému sa odporúča použiť prepínaciu stratégiu riadenia [32]. Takáto forma riadenia nám umožňuje zmerať magnetometrom hodnotu geomagnetického poľa, potom magnetometer vypnúť a počas tejto doby zapnúť cievky na vykonanie akčného zásahu bez toho, aby magnetické pole generované cievkami ovplyvnilo meranie. Tento prístup nám takisto umožňuje použiť filter na odhad derivácie magnetického poľa, ktorým sa odstráni nízkofrekvenčný šum[32].

Tento odhad môžeme vyjadriť pomocou Laplace=ovho obrazu ako:

$$H(s) = \frac{\dot{B}}{B} = \frac{\omega_c s}{s + \omega_c} \tag{5.1}$$

Kde ω_c je uhlová rýchlosť, ktorej hodnota pre dosiahnutie najlepších výsledkov v našom prípade bude 0,7 rad/s. Po prevedení filtra do diskrétnej oblasti Z-transformáciou dostávame:

$$H(z) = \frac{z - 1}{z - e^{\omega_c T_s}} = \frac{b_2(1 - z^{-1})}{1 - a_2 z^{-1}} \tag{5.2}$$

Kde je perióda vzorkovania. Následne dostávame diskrétny tvar filtra, ktorý môžeme využiť v programovom kóde riadenia.

$$\dot{B}_k = a_2 \dot{B}_{k-1} + b_2(B_k - B_{k-1}) \tag{5.3}$$

Výsledná hodnota magnetického poľa sa použije v riadiacom B-dot algoritme, ktorý opisuje rovnica 5.4 v ktorej predstavuje výstupný magnetický moment, je kladné zosilnenie a je výsledná hodnota nameraného magnetického poľa. Tým, že sa bude meniť magnetické pole v okolí satelitu

podľa jeho rýchlosti otáčania, tak záporná spätná väzba regulátora zabezpečí, že sa bude znižovať energia satelitu a tým aj jeho uhlová rýchlosť. Výhodou tohto algoritmu je, že nepotrebuje pre svoju funkčnosť poznať žiadne parametre obežnej dráhy a preto je jeho implementácia pomerne jednoduchá a často obľúbená.

$$m^k = -k\dot{B}_k \quad ; k > 0 \tag{5.4}$$

Nevýhodou tohto riešenia je, že sa nedá analyticky dokázať jeho stabilita, lebo nevieme vždy garantovať že magnetické pole bude vždy kladne semi definitné. Zo simulácií však vyplýva, že algoritmus je asymptoticky stabilný [31].

Testovanie stabilizačného algoritmu

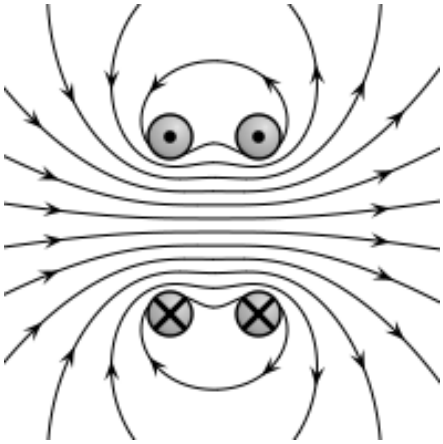
Väčšina tímov testovala funkčnosť B-dot algoritmu pomocou matematických simulácií. Po našich rozhovoroch s kolegami z Estónska ktorý pracovali na satelite EstCube sme dospeli k záveru, že takáto forma overenia správnej funkčnosti tohto algoritmu je nedostačujúca. Väčšina tímov sa stretla s tým, že im simulačne odladený algoritmus vo vesmíre nefungoval správne a museli ho dolaďovať počas priebehu misie experimentálne.

Z tohto dôvodu sme sa rozhodli zhotoviť zariadenie, pomocou ktorého budeme schopný priblížiť čo najvernejšie podmienky na obežnej dráhe Zeme a budeme pomocou neho schopný odsimulovať naše algoritmy na stabilizáciu a orientáciu satelitu.

Výsledkom tohto snaženia je zariadenie, ktoré dokáže simulovať magnetické podmienky pôsobiace na satelit na obežnej dráhe a takisto dokáže simulovať bezváhový stav vo vesmíre. Toto zariadenie pozostáva z troch hlavných častí, ktoré budú podrobne rozpísané v nasledujúcich podkapitolách. Jedná sa o sústavu Helmholzových cievok, vzduchové guľové ložisko a testovaciu platformu.

Sústava Helmholzových cievok

Helmholzove cievky sú kruhového tvaru a sú uložené vždy v páre pozdĺž spoločnej osi. Týmto umiestnením docielime to, že v strede oblasti medzi takýmito dvoma cievkami bude vždy generované homogénne magnetické pole, tak ako znázorňuje nákres na obrázku č. 31.



Obr. 31. Helmholzove cievky [33]

Na vytvorenie homogénneho magnetického poľa vo všetkých troch osiach potrebujeme 6 cievok, teda 3 páry. Výslednú konštrukciu týchto cievok je možné vidieť na obrázku číslo 32. Výslednú intenzitu magnetického poľa v strede medzi dvoma cievkami vieme vyjadriť z Biot-Savartovho zákona ako:

$$B\left(\frac{R}{2}\right) = \left(\frac{8}{5\sqrt{5}}\right)\frac{\mu_0 nI}{R} \qquad \textbf{(5.5)}$$

Kde:

- je premeabilita prostredia

- je prúd prechádzajúci cievkou v ampéroch

– je polomer cievky v metroch

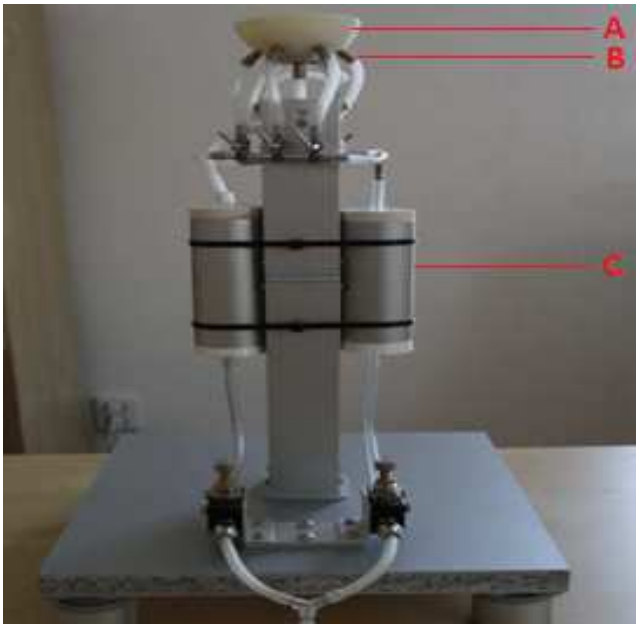
- je počet závitov jednej cievky

Takáto sústava troch párov Helmholzových cievok je schopná vo svojom strede vygenerovať ľubovoľný magnetický vektor a vyrušiť zemský magnetizmus. Tým pádom môžeme generovať aj magnetické pole slabšej intenzity ako magnetické pole Zeme merané na jej povrchu. Tento fakt nám umožňuje simulovať variabilné magnetické pole, ktoré bude pôsobiť na satelite počas obehu Zeme. O riadenie tohto magnetického pola sa stará navrhnutý driver, ktorý komunikuje s aplikáciou v počítači. Pomocou tejto aplikácie je možné nastaviť statické alebo aj dynamické magnetické pole ktoré sa otáča. Samotný driver reguluje podľa príkazov prúd, ktorý prechádza jednotlivými pármi cievok na základe spätnej väzby, ktorú mu poskytuje digitálny precízny magnetometer MAG3310 umiestnený tesne pri plošine v samotnom strede magnetickej sústavy.

Obr. 32. Sústava Helmholzových cievok

Vzduchové guľové ložisko

Na to aby sme boli schopný vhodne simulovať bezváhový stav objektu v priestore sme museli navrhnúť špeciálne zariadenie. Pre naše potreby simulácie rotácie a natáčania satelitu potrebujeme aby sa testovací objekt mohol ľubovoľne otáčať v primeranom rozsahu vo všetkých osiach bez pôsobenia trení a iných síl ktoré by na objekt vo vesmíre nepôsobili. Tým že je kladený dôraz na otáčanie objektu a nie na jeho presun v priestore, tak sme zvolili položenie testovacieho objektu na guľové ložisko. Štandardne takéto ložisko pozostáva z gule a kĺbu v ktorom guľa sedí. Takéto vyhotovenie by síce poskytovalo dostatočnú voľnosť vo všetkých osiach, ale do rotačného pohybu by stále vstupovalo trenie v ložisku. Na odstránenie tohto problému sme navrhli vzduchový systém, ktorý vháňa cez trisky do ložiska vzduch. V ložisku sa potom vytvorí tlak, ktorý nadnáša guľu na jemnom vzduchovom vankúši. Takéto otáčanie gule v kĺbe vytvára minimálne trenie a predstavuje dokonalú simuláciu bezváhového stavu. Konštrukciu takéhoto vzdušného ložiska (SAB – spherical air bearing) možno nájsť na obrázku č. 33.

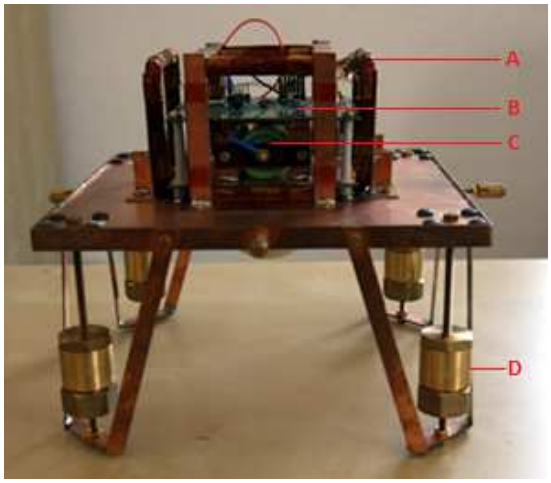


Obr. 33. Vzdušné guľové ložisko SAB

Na obrázku môžeme vidieť ložisko (bod A), ktoré je presne vytvorené ako odliatok – protikus gule zo špeciálnej látky na báze epoxidu. Tento odliatok musí okrem veľkej presnosti dosahovať aj veľmi hladký vnútorný povrch bez pórov a jamiek. Tieto drobné póry spôsobujú turbulencie obtekajúceho vzduchu a tým sa zvyšuje trenie v systéme. Po odliati ložiska boli doňho presne navŕtané otvory pre trisky (bod B) pomocou ktorých sa do ložiska vháňa rovnomerne vzduch. Pri prvých testoch sme prišli na problém, že občas začala testovacia plošina v ložisku naskakovať. Zistili sme, že tento jav nastával preto, lebo pri otáčaní sa občas v niektorých miestach rýchlo zmenil tlak a v systéme nebol dostatok vzduchu aby túto stratu kompenzoval. Z tohto dôvodu sme do celého systému pridali ešte vzduchový kondenzátor (bod C), ktorý tento problém úplne odstránil. Pri správne nastavenom a vyváženom ložisku sme uskutočnili test pri ktorom sme testovaciu plošinu roztočili na počiatočnú uhlovú rýchlosť 20 stupňov za sekundu a merali sme čas kým dôjde k svojvoľnému zastaveniu tejto rotácie. Po úplne zastavenie sa plošina bez akéhokoľvek zdroja pohybu zastavila po 47 minútach.

Testovacia platforma

Testovacia platforma bola skonštruovaná ako nosná konzola s vyvažovacími závažiami na ktorej je uchytená elektronika určená na testovanie riadiacich algoritmov tak ako možno vidieť na obrázku č. 34.



Obr. 34. Testovacia platforma

Túto elektroniku je v neskoršej fáze možné vymeniť za letovú verziu satelitu. Samotná platforma je skonštruovaná tak, aby sa dala použiť aj v budúcnosti pri misiách s väčším satelitom s rozmermi 2 aj 3U. Na obrázku 27 môžeme v rohoch platformy vidieť vyvažovacie závažia (bod D). Okrem týchto závaží obsahuje ešte platforma malé dovažovacie závažia, ktoré slúžia na presné doladenie ťažiska platformy. Ovládacia elektronika sa nachádza na mieste bodu B. Táto elektronika je napájaná pomocou LiFe batérie s kapacitou 2200 mAh (bod C). Pre interakciu s vonkajším magnetickým poľom, ktoré generujú Helmholzove cievky slúžia cievky satelitu (bod A). Na tejto testovacej zostave sú použité cievky bez jadra, ale do letovej verzie satelitu plánujeme použiť cievky s jadrom kvôli ich vyššej účinnosti.

Ovládacia elektronika obsahuje v sebe riadiaci procesor STM32 s jadrom ARM Cortex M3. Tento procesor slúži na vykonávanie výpočtov riadiacich algoritmov a spracovanie údajov zo senzorov umiestnených na platforme. Tieto údaje sa následne bezdrôtovo prenášajú pomocou Bluetooth modulu priamo do počítača kde sú v aplikácii zobrazené v reálnom čase.

Záver

Táto práca priblížila problémy, ktoré museli byť vyriešené počas návrhu riadiaceho počítača prvej Slovenskej družice od rozdelenia Československa s názvom skCube. V úvode boli priblížené jednotlivé subsystémy, ktoré táto družica bude obsahovať ako aj unikátny experiment, ktorý sa bude snažiť zrealizovať.

Jadro práce bolo venované návrhu hardvéru palubného počítača. Postupne bol opísaný výber jednotlivých komponentov spolu s prvotnými testami výkonnosti procesora v aritmetických a goniometrických funkciách, ktoré budú potrebné v riadiacich a stabilizačných algoritmoch. Následne vybrané komponenty boli osadené na testovaciu dosku, ktorá sa pomocou rengenového žiariča ožiarila na lekárskej fakulte. Týmto testom sme preverili odolnosť vybraných súčiastok voči radiácii, ktorá ich čaká v kozmickom prostredí. Takisto sme spomenuli systém režimu prepínania dvoch záložných procesorov pomocou jedného externého supervízora. V časti venovanej operačnému systému reálneho času sme podrobne popísali dizajnovú logiku a jednotlivé algoritmy. V práci sú uvedené aj informácie k jednotlivým nástrojom, ktoré bolo nutné navrhnúť pri vývoji tohto operačného systému ako simulátor v prostredí Simulink a emulátor pre Unix systémy. Na konci tejto kapitoly sme sa venovali špeciálnym metódam ako zabezpečiť všeobecnú odolnosť tohto systému voči mnohým vonkajším poruchám. Tým, že plánujeme po misii skCube zverejniť zdrojové kódy tak sa bude jednať o prvý rte RTOS s týmito technikami implementovanými priamo v jadre systému.

V dvoch posledných častiach práce sme sa zamerali na spracovanie signálov zo senzorov a na testovanie stabilizačného algotimu. Pri spracovaní signálov sme navrhli filter, ktorý využíva pre svoj odhad quaterniónové výpočty. Tento prístup nám podstatne zjednodušil výpočtovú náročnosť tohto algoritmu vďaka čomu sme schopný dosiahnuť rýchlejšie periódy odhadu orientácie satelitu. V poslednej časti sme sa venovali hlavne problému stabilizovania počiatočnej rotácie satelitu na Zemskom orbite. Za týmto účelom bol prezentovaný algoritmus B-dot, pre ktorého testovanie bolo zhotovené praktické laboratórne simulačné zariadenie. Toto zariadenie sa skladá zo sústavy Helmholzových cievok, guľového vzdušného ložiska a testovacej platformy. Samozrejmosťou je aj bezdrôtové získavanie dát z testov v reálnom čase.

S pomedzi bežných európskych aj svetových tímov pracujúcich na vlastných CubeSatoch sa ten náš vyznačuje hlavne tým, že v satelite používa svoj vlastný operačný systém navrhnutý špeciálne pre podmienky práce vo vesmíre a takisto tým, že všetok hardvér družice je vlastného návrhu aj výroby. S pomedzi európskych tímov patríme k jediným, ktoré boli schopné vyrobiť a navrhnúť testovaciu stolicu na ktorej by bolo možné testovať stabilizáciu satelitov v bezváhovom stave. Ďalší tým s veľmi úspešnou misiou EstCube problémy stabilizácie a testov algoritmov riešil až po vypustení svojho

satelitu na Zemskú orbitu, čím stratili drahocenný čas, ktorý je pre realizáciu nášho experimentu detekcie gama záblesков veľmi dôležitý.

Literatúra

Okrem prác autora

[1] *Projekt SolarSail*, SolarSail team, dostupné na internete: http://sail.planetary.org/

[2] *Ion Electrospray Engines Could Take CubeSats to the Moon and Beyond*, Evan Ackerman, IEEE Spectrum, dostupné na internete: http://spectrum.ieee.org/tech-talk/aerospace/satellites/ion-electrospray-engines-give-a-boost-to-tiny-satellites

[3] Electrospray Propulsion System for CubeSats, Paulo Lozano, Space Propulsion Laboratory, http://web.mit.edu/aeroastro/labs/spl/research_ieps.htm

[4] Webová stránka spoločnosti SpaceX, dostupné na internete: http://www.spacex.com

[5] Projekt stratosférickej sondy, kolektív SOSA, dostupné na internete: http://sosa.sk/stratosfericky-balon/

[6] Pioneer Galileo navigation fixes, ESA, dostupné na internete: http://www.esa.int/Our_Activities/Navigation/Pioneer_Galileo_navigation_fixes_recognised_by_ESA

[7] Štandard MIL-STD-1553, dostupné na internete: https://en.wikipedia.org/wiki/MIL-STD-1553

[8] Tongyue Gao, Zhenbang Gong, Jun Luo, Wei Ding, Wei Feng, *An Attitude Determination System for a Small Unmanned Helicopter using Low-Cost Sensors*, Robotics and Biometrics IEEE International Conference, 2006, ISBN: 1-4244-0570-X

[9] Jungmin Kim, Yountae Kim, Sungshin Kim, *An accurate localization for mobile robot using extended Kalman filter and sensor fusion*, Neural Networks IEEE World Congress, 2008, ISBN: 978-1-4244-1820-6

[10] Sebastian O. H. Madgwick, *An efficient orientation filter for intertial and inertial/magnetic sensor arrays*, Internal report, 2010

[11] Steve Furber, *ARM System on a Chip*, ADDISON-WESLEY, 2000, ISBN 13: 978-0-201-67519-1

[12] F. Faccio, G. Anelli, M. Campbell, and co., *Total dose and single event effects (SEE) in a 0.25um CMOS technology*, dostupné online: http://ilc.fnal.gov/detectors/rd/New%20Folder/tidsee.pdf

[13] Van Allenove pásy zmerané sondami Pioneer, NASA, dostupné online: http://image.gsfc.nasa.gov/poetry/tour/AAvan.html

[14] Tanya Vladimirova, Christoper P. Bridges, George Prassions and co., *Characterising Wireless Sensor Motes for Space Applications*, Second NASA/ESA Conference on Adaptive Hardware and Systems 2007, ISBN: 0-7965-2866-X/07

[15] Lol software team, *Better function approximations*, dostupné online: http://lolengine.net/blog/2011/12/21/better-function-approximations

[16] Jack Ganssle, *A Guide to Approximations*, Ganssle Group, dostupné online: http://www.ganssle.com/approx.htm

[17] Lol software team, *Playing with the CPU pipeline*, dostupné online: http://lolengine.net/blog/2011/9/17/playing-with-the-cpu-pipeline

[18] NanoMind A712 – CubeSat OnBoard Computer, dostupné na internete: http://www.cubesatshop.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=68&category_id=8&option=com_virtuemart&Itemid=75&vmcchk=1&Itemid=75

[19] Rob Williams, *Real Time Systems Developement*, Elsevier, 2006, ISBN: 13:978-0-7506-6471-4

[20] Chowdary Venkateswara Penumuchu, *Simple real-time operating system: a kernel inside view for beginer*, Trafford, 2007, ISBN: 978-1-4251-1782-5

[21] Misra Limited, *MISRA-C:2004 Guidelines for the use of the C language in critical systems*, 2004, ISBN: 978-0-9524156-4-0

[22] Indrek Sünter, *SOFTWARE FOR THE ESTCUBE-1 COMMAND AND DATA HANDLING SYSTEM*, University of Tartu Faculty of Science and Technology, 2014 dostupné online: http://www.tuit.ut.ee/sites/default/files/tuit/arvutitehnika-thesis-msc-2014-synter-indrek-text-20140528.pdf

[23] NASA JPL, *JPL Institutional Coding Standard for the C Programming Language*, Jet Propulsion Laboratory, California Institute of Technology, 2009, dostupné online: http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf

[24] Mitch Fletcher, *Progression of an Open Architecture: from Orion to Altair and LSS*, Honeywell International, 2009, dostupné online: http://www.zettaflops.org/spc09/S69-5000-20-0-Fault-Tolerant-Open-Computing-System.pdf

[25] Amy Svitak, *Dragon's "Radiation-Tolerant" Design*, Aviation week, 2012, dostupné online: http://aviationweek.com/blog/dragons-radiation-tolerant-design

[26] Katalógový list k produktu Melexis MLX90620, dostupný online: http://www.melexis.com/Asset/Datasheet-IR-thermometer-16X4-sensor-array-MLX90620-DownloadLink-6099.aspx

[27] S.O.H. Madgwick, A.J.L. Harrison, R. Vaidyanathan, *Estimation of IMU and MARG orientation using a gradient descent algorithm*, Rehabilitation Robotics, IEEE Conference, 2011, ISBN: 978-1-4244-9861-1

[28] Estcube team, *Projekt EstCube*, dostupné online: http://www.estcube.eu/

[29] Masat-1, dostupné online: http://cubesat.bme.hu/?lang=en

[30] A. Stickler a K. Alfriend, *An elementary magnetic attitude control system*, American Instutute of Aeronautics and Astronautics, Mechanics and Control of Flight Conference, Anaheim, California vol. 5, 1974.

[31] Gaute Bråthen, *Design of Attitude Control System of a Double CubeSat*, Master thesis, Norwegian University of Science and Technology, dostupné online: http://www.diva-portal.org/smash/get/diva2:617039/FULLTEXT01.pdf

[32] Torsten Lorentzen, *Attitude Control and Determination System for DTUsat1*, Cubesat Project, University of Denmark, 2002

[33] Obrázok prevzatý z internetu: https://upload.wikimedia.org/wikipedia/commons/thumb/9/9c/VFPt_helmholtz_coil_thumb.svg/255px-VFPt_helmholtz_coil_thumb.svg.png

[34] Video testovacej platformy, Kolektív SOSA, dostupné online: https://youtu.be/1X7-In85QnA

Publikované práce autora

KAJAN, Slavomír - SLAČKA, Juraj. Parallel Computing on Graphics Cards. In *AT&P Journal Plus*. Č. 1: Systémy automatického riadenia (2010), s.49-52. ISSN 1336-5010.

KAJAN, Slavomír - SLAČKA, Juraj. Computing of Neural Network on Graphics Card. In *Technical Computing Bratislava 2010 : 18th Annual Conference Proceedings. Bratislava, Slovak Republic, 20.10.2010*. Bratislava : RT Systems, 2010, s.CD-Rom. ISBN 978-80-970519-0-7.

SLAČKA, Juraj - MIKLOVIČOVÁ, Eva. Control of Nonlinear Systems. In *Technical Computing Bratislava 2012 [elektronický zdroj] : 20th Annual Conference Proceedings. Bratislava, 7.11. 2012*. Bratislava : RT Systems, 2012, s.CD-ROM [6] s. ISBN 978-80-970519-4-5.

SLAČKA, Juraj - PETRÍK, Slavomír - HALÁS, Miroslav. Embedded RTOS for skCube satellite. In *Technical Computing Bratislava 2014 [elektronický zdroj] : 22nd Annual Conference Proceedings. Bratislava, SR, 4. 11. 2014*. 1. vyd. Prague : Institute of Chemical Technology, 2014, CD-ROM, [6] s. ISBN 978-80-7080-898-6.

SLAČKA, Juraj. IMU filter for embedded device. In *ELITECH´14 [elektronický zdroj] : 16th Conference of Doctoral Students; Bratislava, Slovakia, 4 June 2014*. 1.vyd. Bratislava : Nakladateľstvo STU, 2014, CD-ROM, [4] p. ISBN 978-80-227-4171-2.

SOÓS, Dávid - SLAČKA, Juraj. Experimenting with simplest dead time compensator. In *ICETA 2014 [elektronický zdroj] : 12th IEEE International Conference on Engineering eLearning Technologies and Applications. December 4-5, 2014 Starý Smokovec, Slovakia*. 1. vyd. Danvers : IEEE, 2014, CD ROM, s. 447-451. ISBN 978-1-4799-7738-3.

SLAČKA, Juraj - HALÁS, Miroslav. Safety Critical RTOS for Space Satellites. In *Proces Control 2015.Štrbské pleso, SR, 9. 6. 2015*. 2015, CD-ROM, [5] s. ISBN 978-1-4673-6626-7.

Hrečko Lukáš - SLAČKA, Juraj - HALÁS, Miroslav. Bicopter Stabilization Based on IMU Sensors. In *Proces Control 2015.Štrbské pleso, SR, 9. 6. 2015*. 2015, CD-ROM, [6] s. ISBN 978-1-4673-6626-7.

SLAČKA, Juraj - HALÁS, Miroslav – Ondrej Závodský. Test platform for magnetic stabilization of a small satellite. In *INTECH 2015.Dubrovnik, HR, 9. 9. 2015*. 2015, Accepted paper to be published

Zoznam citácií

Musa Peker, Baha Şen, Hüseyin Gürüler - *Rapid Automated Classification of Anesthetic Depth Levels using GPU Based Parallelization of Neural Networks*, Journal of medical systems, DOI 10.1007/s10916-015-0197-3

Tomislav Bacek, Dubravko Majetic, Danko Brezak - *GPU implementation of the feedforward neural network with modified Levenberg-Marquardt algorithm*, Neural networks (IJCNN), 2014