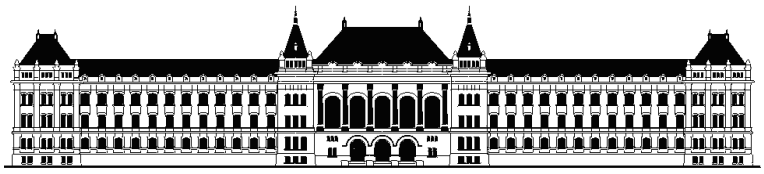


M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

SMOG-1 fedélzeti számítógép fejlesztése

BSc szakdolgozat

Készítette
Kristóf Timur

Konzulens
Dudás Levente

2017. december 8.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Kristóf Timur*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

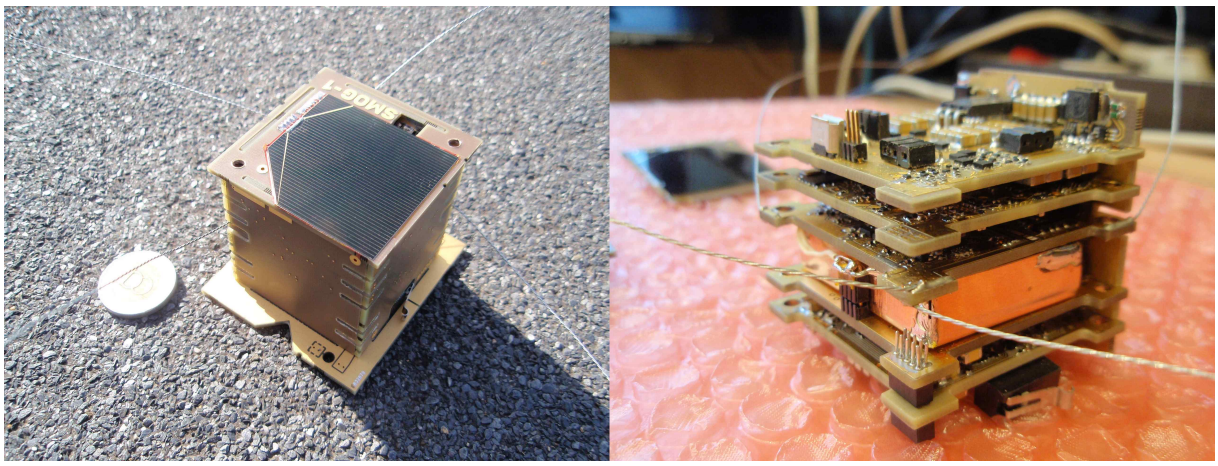
Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. december 8.

Kristóf Timur
hallgató

Kivonat

A SMOG-1 Magyarország következő műholdja, amely a PocketQube szabvány szerint készül Egyetemünkön. Küldetése az elektromágneses spektrum monitorozása és az ionizáló sugárdózis mérése alacsony Föld körüli pályáról.



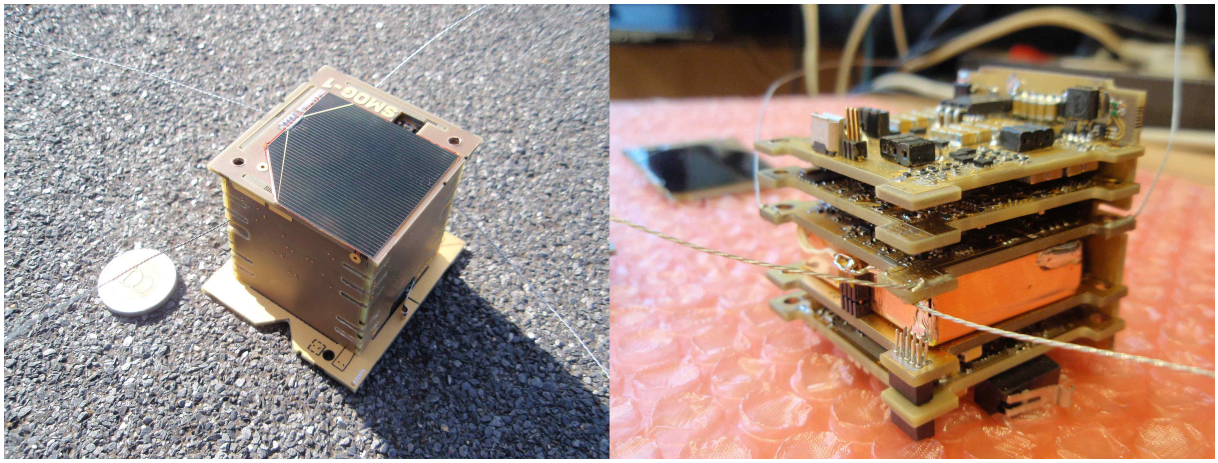
Saját feladatom a projektben elkészíteni műhold fedélzeti számítógépét és a rajta futó szoftvert. Dolgozatomban azt mutatom be, milyen szempontok szerint terveztem és hogyan készült a SMOG 1 fedélzeti számítógép (on-board computer, OBC), ami kapcsolódik a műhold összes többi alrendszeréhez és ezek működését vezérli, monitorozza és összehangolja.

Hardveres szempontból a fedélzeti számítógép összetevői: egy ARM Cortex-M mikrokontroller, RTCC (óra és naptár), flash memória, és a szenzorok, amelyek a műhold „érzékszervei" lesznek. Az OBC-hez kapcsolódik perifériaként a többi alrendszer is: a kommunikációs rendszer (COM), valamint az energiaellátó rendszerek, amelyektől telemetriát gyűjt. A nagy megbízhatóság érdekében mindegyik egységből kettő lesz a fedélzeten, amelyek hideg redundanciával, egy-pont meghibásodásra méretezetten működnek.

Az OBC szoftverének feladata, hogy vezérelje a műhold működését, a méréseket ütemezze, telemetriát gyűjtsön, valamint irányítsa a kommunikációt a földi állomással, amely egy 4.5 m-es parabola antennával fogja venni a SMOG 1 jeleit a BME „E" épület tetején. A szoftver „event loop" felépítéssel valós időben fut.

Abstract

SMOG-1 is Hungary's next satellite which is made according to the PocketQube standard. Its mission is to monitor the electromagnetic spectrum and measure ionizing radiation dose from low Earth orbit.



My task is to develop the on-board computer and its software for the satellite. In this work I will present how I designed the SMOG 1 on-board computer (OBC) and how it was made. The OBC is the subsystem which connects to all other parts in the satellite and controls, monitors and coordinates them.

The on-board computer hardware consists of an ARM Cortex-M microcontroller, RTCC (clock and calendar), flash memory and sensors that are the "eyes and ears" of the satellite. All other subsystems connect to the OBC as peripherals: the communication subsystem (COM) and the electrical power subsystems from which the OBC collects telemetry. For high reliability, there are going to be two of each unit on board which will work in cold redundancy without any single point of failure.

The on-board computer software is tasked with controlling the satellite, collecting telemetry and managing the communications with the ground station which is going to receive signals from SMOG 1 using a 4.5 m parabolic antenna on top of BME building "E". The software is written using an "event loop" architecture and runs in real time.

Első fejezet

Bevezető

A SMOG-1 Magyarország következő műholdja. Az első a MASAT-1 volt, amely az eredeti CubeSat szabvány szerint 10 × 10 × 10 cm-es volt. Ennek egyik utódprojektje a SMOG-1, amely már az újabb, kisebb méretű PocketQube szabvány szerint készül a Budapesti Műszaki és Gazdaságtudományi Egyetemen a Villamosmérnöki és Informatikai Kar és a Gépészmérnöki kar együttműködésével. [1] [3] [4] [5] [11] [10] [7] [8] [9] Mérete legfeljebb 5 × 5 × 5 centiméter, tömege legfeljebb 250 gramm. A műhold elektronikáját a Szélessávú Hírközlés és Villamosságtan Tanszék hallgatói tervezik és valósítják meg.

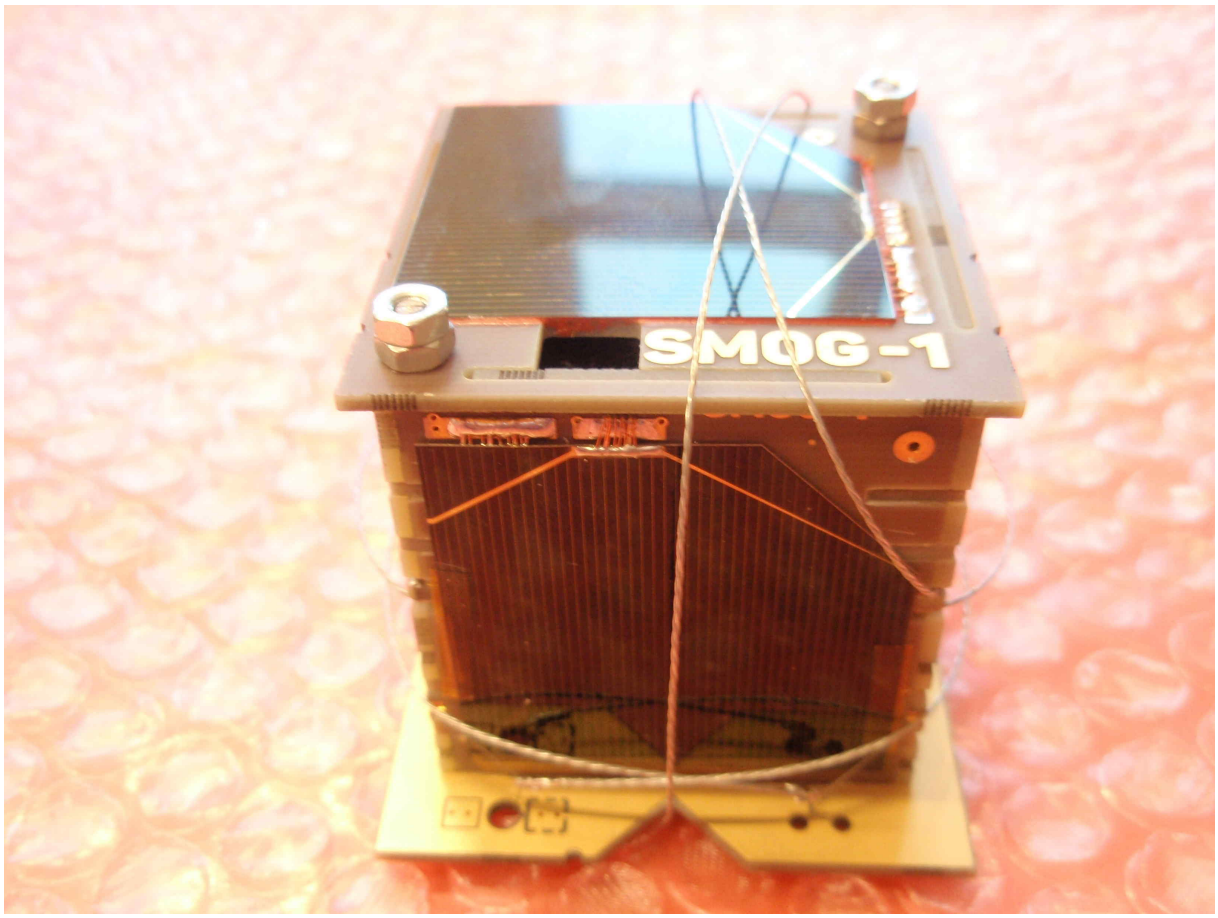
A projekt 2014-ben kezdődött. A mérnöki példány 2016-ban készült el, a kvalifikációs példány pedig 2017-ben. Jelenlegi terveink szerint 2018 első felében várható a repülő példány elkészülte. Ezt a példányt is (akárcsak a korábbiakat) alapos tesztelésnek fogjuk alávetni, mielőtt elengednénk a világűrbe.

Saját feladatom a projektben elkészíteni műhold fedélzeti számítógépét és a rajta futó szoftvert. Jelen dolgozat a BSc önálló laboratórium feladatom [6] folytatásaként született. Dolgozatomban azt mutatom be, milyen szempontok szerint terveztem és hogyan készült a SMOG-1 fedélzeti számítógép (on-board computer, OBC), ami a műhold összes többi alrendszeréhez kapcsolódik és ezek működését vezérli és összehangolja. Mivel az OBC minden más alrendszerrel kapcsolatban van, elengedhetetlen számomra, hogy a többi alrendszer működésével is tisztában legyek. Ezért vállaltam részt a műhold rendszertervének kidolgozásában is, és fektetek hangsúlyt a rendszertervre is a dolgozatom elején.

1.1. Küldetés

Műholdunk fő küldetése az elektromágneses spektrum monitorozása a Föld körüli pályáról DVB-T sávban (digital video broadcasting terrestrial, vagyis a földi digitális műsorszóró adók). Azt mérjük, hogy a földi DVB-T műsorszóró adók jele mennyire vehető a világűrből. [1] (A SMOG-1-et kb. 500-600 km-es magasságú pályára tervezzük.)

A műsorszóró adók célpontja a földi lakosság, ezért minden olyan elektromágneses jel, ami vehető ezektől az adóktól az űrben, valójában elpazarolt, kidobott rádiós teljesítmény, amely



1.1. ábra. SMOG-1 kívülről

olyan mértékeket ölt, hogy már akadályozza a műholdas kommunikációt. Küldetésünk képet ad a földi DVB-T lefedettségről, és segít a műsorszórók hatékonyabbá tételében.

1.1.1. Kihívások

Energia

Előzetes becslésünk szerint összesen 0,3 watt (körátlag) [9] teljesítménnyel kell gazdálkodnunk, a fedélzeti alrendszerek legfeljebb ennyit fogyaszthatnak.

Az energia viszonyok miatt a kommunikációval áthidalt távolság is kihívás. A földi állomáshoz a csapatunk beszerzett egy 4.5 m átmérőjű parabola antennát, amelynek primer sugárzóját Légrádi Máté készítette [13]. A földi állomás vezérlő szoftverét pedig Kálmán Tibor fejleszti. [5]

Redundancia

Hogy ellenálljon a világűr viszontagságainak, a SMOG-1 alrendszereit redundáns módon kell megtervezni, mégpedig úgy, hogy bármely pont meghibásodása esetén a teljes rendszer zökkenőmentesen működhessen tovább. Ez azt jelenti, hogy a tervezés során különös gondot kell fordítani arra, hogy ha bármelyik alkatrész elromlik, azzal ne veszélyeztesse a többi komponenst.

Alkatrészek

A SMOG-1 a magyarországi műholdak oktatási vonalát képviseli. Ez azt jelenti, hogy nagy részét a Műegyetem hallgatói tervezik, és a költségvetése is az egyetemi kasszához mért, vagyis drága, űrminősített eszközök helyett a kisműholdak (picosatellite) tervezői a piacon széles körben elérhető ipari alkatrészekből építkeznek. Legfőbb szempontok az alkatrészek mérete és megbízhatósága.

Mechanikai terhelés és termikus viszonyok

Gépészmérnök kollégáink foglalkoznak ezekkel a kérdésekkel. Meg kell győződni arról, hogy a műhold olyan hőmérsékletű lesz a világűrben, ami az alkatrészek, különösen az akkumulátor üzemi hőmérséklettartományába esik. [10] [11]

Helymeghatározás

Műholdunkon GPS vevő nincs, mert a kereskedelmi forgalomban kapható GPS vevő áramkörök korlátozzák a maximális magasságot, sebességet és gyorsulást (hogy ne lehessen belőlük fegyvert készíteni). Ehelyett a műholdat a Földről követjük TLE (two-line element set) segítségével. [5]

1.2. Felépítés

1.2.1. Mechanikai szerkezet

A műhold belsejében FR-4 áramköri lemezekből kialakított „polcok" vannak. Egy-egy áramköri lapon található egy-egy alrendszer. Az alrendszerek összeköttetése a közöttük levő rendszerbusz csatlakozó segítségével valósul meg. A „polcokhoz" illeszkednek a szintén FR-4 oldallemezek, amelyekkel mintegy „puzzle" szerűen rakható össze a műhold. A belső paneleket két menetes szár tartja össze a kocka alsó és felső napelem oldalával. [8] [7]

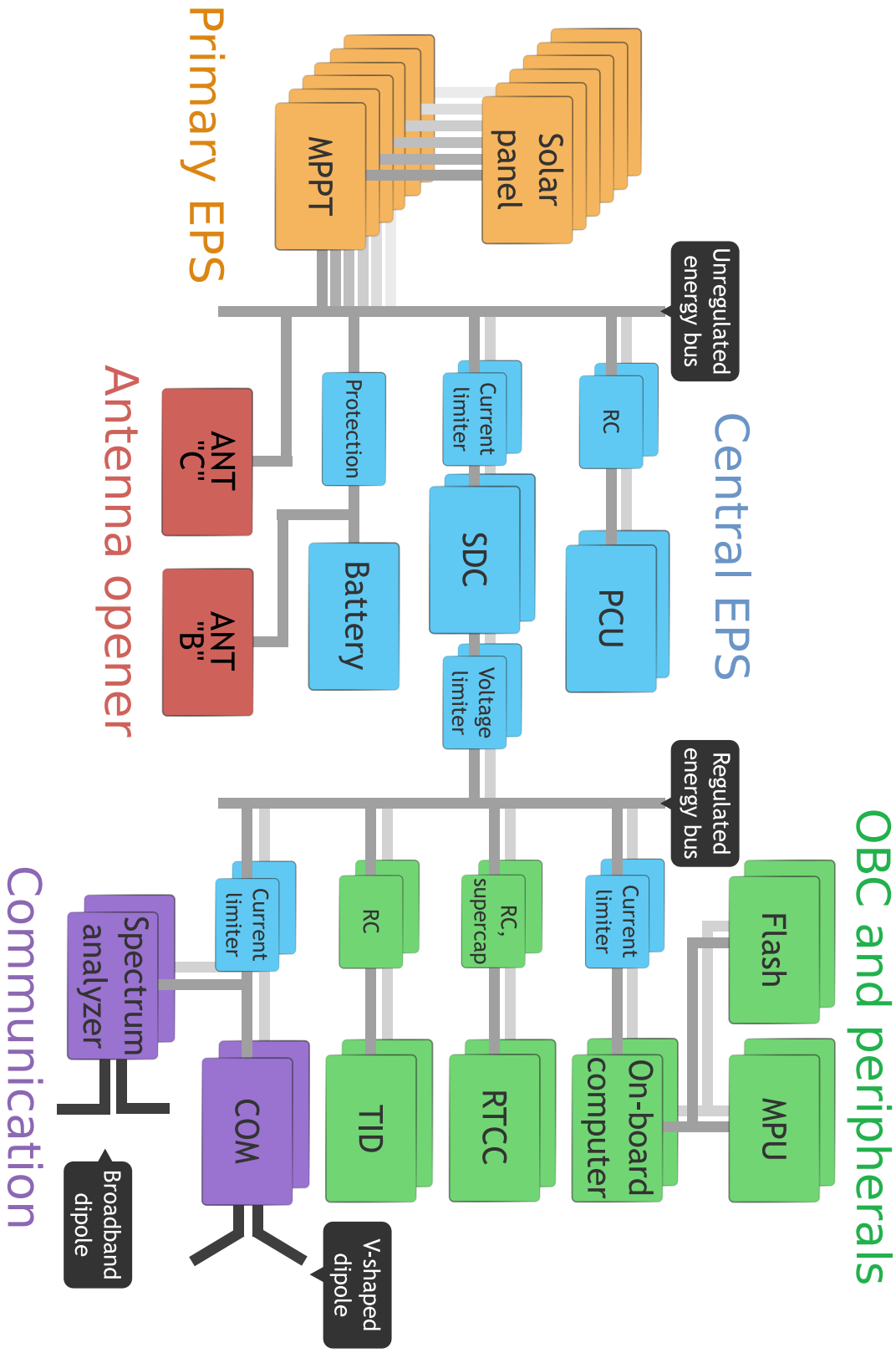
1.2.2. Rendszerterv

Alábbiakban röviden ismertetem a műhold további alrendszereit és az OBC helyét ezek között. A teljes rendszertervet láthatjuk az 1.2 ábrán.

Lényeges elemek védelméről áramkorlátozó és túlfeszültségtől védő kapcsolók gondoskodnak. [3] Az alacsony fogyasztású alkatrészeket egyszerű RC tagokkal kötjük a szabályozott energiabuszra (ennek következtében nem „húzzák le" a buszt, ha tönkremennek), az RTCC számára pedig szuperkapacitás biztosítja a redundáns energiaellátást.

EPS1 (Elsődleges energiaellátás)

Elsődleges energiaellátásunkat kb. 4×4 cm-es napelemek biztosítják, amelyek a kocka 5 cm-es oldallapjain helyezkednek el, és MPPT áramkör által optimalizálva táplálják a SMOG-1-et.



1.2. ábra. Részletes rendszerterv

[4]

EPS2 (Központi energiaellátó rendszer)

A napelemek egy ún. szabályozatlan energiabuszra vannak rákötve. Erről üzemel az EPS2, ami tartalmaz egy akkumulátort, egy PCU (power control unit) vezérlő egységet, egy feszültségcsökkentő kapcsolást (SDC, step-down konverter), amely a fedélzeti számítógép és perifériái részére állít elő megfelelő feszültséget. Ezen áramkörök mindegyike megfelelő védelemmel (túláram és túlfeszültség, redundancia, stb.) van ellátva. [3]

OBC (Fedélzeti számítógép)

Ennek az alrendszernek a kidolgozása a saját feladatom. Az OBC feladata az összes többi alrendszert vezérelni, monitorozni és összehangolni. Részegységei: egy mikrokontroller, flash memória, RTCC (real time clock and calendar, vagyis lényegében egy pontos kvarcóra), és MPU (motion processing unit, amely tartalmaz giroszkópot, gyorsulásmérőt és mágneses térerősségmérőt). Bővebben ld. a további fejezeteket.

COM (Kommunikációs alrendszer)

A COM rendszert szétbonthatjuk a földi állomással kommunikáló egységre és a spektrumanalizátorra. [1] A két spektrumanalizátor és a két rádiókommunikációs rendszer egy-egy közös antennával rendelkezik. Ezen antennák egymásra merőlegesen helyezkednek el a műhold külsején.

ANT (Antennanyitó alrendszer)

Legkritikusabb folyamat a műholdunkon az antennanyitás. Ha ez nem történik meg, akkor lehetetlen lenne felevenni a kapcsolatot a Földről a műholddal. Pályára állításkor a helytakarékosság végett az antennák a műhold köré lesznek tekerve (kellően rugalmas anyagból készülnek ehhez), és egy polimerszállal lesznek rögzítve, amit nyitáskor el kell olvasztani. [12]

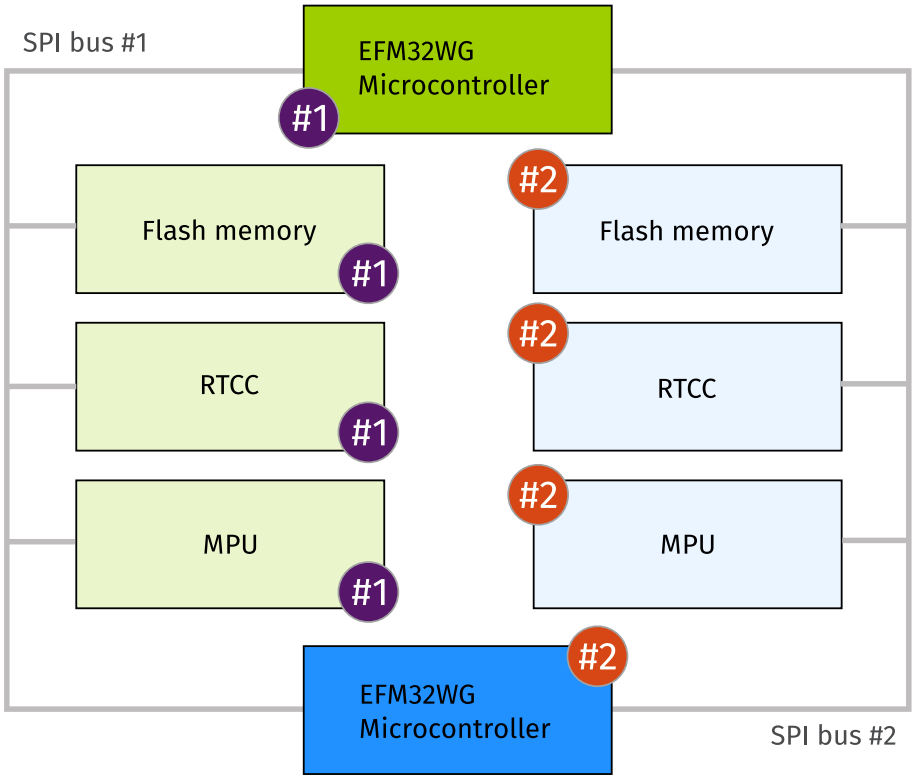
TID (Sugárdózis mérő)

Másodlagos küldetésünk az ionizáló sugárzás (total ionization dose, TID) mérése, amelyet egy RAD FET alapú totáldózismérő segítségével valósítunk meg. [3]. Azért különleges, mert ilyen kis méretben még senki sem valósította meg, és LEO (low-Earth orbit) pályán kevesen végeztek hasonló méréseket.

Második fejezet

Fedélzeti számítógép hardvere

2.1. Felépítés



2.1. ábra. OBC blokkvázlata

Minden hardverelemből két példány foglal helyet az OBC panelen (redundancia megfontolásokból, ld. lentebb). A perifériák (RTCC, flash, MPU) egy-egy példánya külön SPI (serial peripheral interface) buszon fognak a mikrokontrollerekkel kommunikálni. Ez úgy értendő, hogy egy mikrokontroller két különböző SPI buszon is master módban működik, és a redundáns párja is rajta lesz mindkét SPI buszon szintén master módban. Fontos megemlíteni, hogy habár mindkét OBC nem lesz egyszerre bekapcsolva (kettejük között ún. hideg redundancia valósul

meg), a busz el lesz látva védődiódákkal és ellenállásokkal, hogy a kettejük egyidejű (téves) bekapcsolt állapota biztosan ne okozza a slave eszközök meghibásodását. Tehát a memória, RTCC és szenzorok egyik példánya az egyik, másik példánya a másik SPI buszon foglal majd helyet slave módban. Így, ha akár mindegyik fajta egység közül egy megsérül, a teljes rendszer még mindig üzemképes marad.

Az OBC felépítésének blokkvázlata megtekinthető a 2.1 ábrán.

2.1.1. Kapcsolási rajz

A függelékben megtalálható az OBC mikrokontroller, valamint a perifériák egy példányának a kapcsolási rajza. Természetesen az OBC panelen mindegyikből két példány foglal helyet.

- Kezdem a teljes rendszer kapcsolási rajzával, amely mindegyik példányt tartalmazza, és ezek kapcsolatát mutatja be egymás között és a rendszerbusz csatlakozóval (amellyel a többi panelhez kapcsolódik az OBC).
- Utána következik a mikrokontroller kapcsolási rajza, amely a szükséges védőáramköröket is tartalmazza. Ezen a fentiekben részletezett egyvezetékes védelmet már csak egy „1-wire protection” feliratú doboz jelzi.
- Végül következik a perifériák egy-egy példányának kapcsolási rajza az RTCC redundáns tápellátásával és az SPI busz védelemmel együtt.
- Az áramlimiterek kapcsolási rajzát itt nem közlöm, megtalálható Géczy Gábor dolgozatában. [3]

2.2. Áramköri megvalósítás, megfontolások

2.2.1. Redundancia

Az egész műhold tervezésekor az egyik legfontosabb áramköri szempont az „egy pont” meghibásodásra (single point of failure) vonatkozó redundancia. Ez azt jelenti, hogy a műholdon belül bármely alkatrész (bármely „egy pont”) meghibásodása esetén a teljes rendszernek tovább kell tudnia üzemelni. Amennyiben lehetséges, funkcionalitás vesztés nélkül, azonban ez nem mindig kivitelezhető a korábban ismertetett korlátok mellett. (Pl. akkumulátorból csak egy van.)

Lehetséges problémák

- Ellenállások: szakadássá változhatnak, ezért kritikus helyekre legalább két párhuzamos ellenállást helyezünk el, vagy funkcionálisan tartalékoljuk az egységet, amelyben az adott ellenállás van.
- Kondenzátorok: rövidzárrá vagy szakadássá is változhatnak, ezért kritikus helyre négy kondenzátort kell elhelyezni: két-két sorbakapcsolt kondenzátort egymással párhuzamosan. Ha ezek közül bármelyik bárhogyan elromlik, a kapcsolás még mindig kondenzátorként működik, csak a kapacitása változik meg. Azonban ez a fajta tartalékolás nem mindig kifizetődő, sok helyütt megelégszünk a funkcionális tartalékolással.
- Diódák, tranzisztorok: rövidzárrá vagy szakadássá is változhatnak. Ha a rövidzárrá változás ellen akarunk védekezni, akkor két darab sorbakapcsolt elem, ha pedig a szakadássá változás ellen, akkor két párhuzamosan kapcsolt elem kell. Tehát négy elemmel mindkét problémát orvosolhatjuk, azonban itt fokozottan igaz az, hogy ez sokhelyütt felesleges, és elegendő csak az egyik problémával foglalkozni, a másikat pedig funkcionális tartalékolással kezelni.
- Integrált áramkörök (IC), aktív eszközök: bármelyik alkatrészláb rövidre záródhat akár a föld (logikai 0) vagy a tápfeszültség (logikai 1) felé, vagy tirisztorhatás jöhet létre. Ezt szinte mindig funkcionális redundanciával küszöböljük ki, vagyis az adott funkcionális egységből, amelynek része az adott áramkör, legalább kettőt helyezünk el a fedélzeten. Azonban minden esetben gondoskodni kell a következőkről:
  - Az integrált áramkör meghibásodása esetén ne „húzza le” a tápfeszültséget, vagyis ne csináljon táp-föld zárlatot. Ezt tipikusan diódákkal és áramkorlátozó ellenállásokkal lehet megoldani.
  - Ha az IC valamilyen kommunikációs buszra csatlakozik (pl. SPI), akkor meghibásodás esetén ne tegye lehetetlenné a többi eszköz számára a kommunikációt. Például ha tönkremegy valamelyik periféria egy SPI buszon, akkor ettől még a többi (működő) periféria és a mikrokontroller közötti kommunikációnak továbbra is üzemelnie kell.

– Az IC véletlenül se kapjon tápfeszültséget valamelyik I/O lábon (meghibásodás esetén sem), amikor egyébként nem szabadna. Ez triviálisnak tűnhet, de például egy mikrokontroller lehet annyira alacsony fogyasztású, hogy egy kommunikációs vonalon (pl. UART, SPI) megjelenő logikai magas szint hatására is bekapcsoljon. (Erről az OBC-hez használt EFM32WG esetén meg is győződtük, meglepetésünkre az is bekapcsol ilyenkor.)

Funkcionális tartalékolás

A fedélzeti számítógép mikrokontrolleréből két redundáns példány is helyet kap az áramkörben, amelyek egymás *funkcionális tartalékai*. Közöttük *hideg redundancia* (cold redundancy) valósul meg, ami azt jelenti, hogy egyszerre csak egyikük lesz bekapcsolva. Annak eldöntése, hogy melyiket kell bekapcsolni, a PCU (power control unit) feladata, amely az EPS (energiaellátó rendszer) része.

A funkcionális tartalékolás úgy valósul meg, hogy az OBC ún. *heartbeat* („szívdobbanás") jeket küld periodikusan a PCU-k felé. Ha ez a jel megszűnik, vagyis a PCU nem látja a heartbeat jelet bizonyos ideig, akkor újraindítja (reseteli) az aktuálisan bekapcsolt OBC példányt úgy, hogy az energiaellátását lekapcsolja, majd rövid idő múlva visszakapcsolja. Ha a heartbeat jel ezután sem áll helyre, akkor ezt a példányt a PCU elkönyveli „halottnak" és helyette a másikat indítja el.

A mikrokontrollerekhez fizikailag is legközelebb eső elemek (ezek vele azonos nyomtatott huzalozású lemezen helyezkednek el) is természetesen funkcionálisan tartalékoltak. Például ez azt jelenti, hogy a mikrokontroller valamelyik lábához amennyiben pl. diszkrét diódát helyeztünk el, nincs szükség belőle 4 darabra, hanem elég kettő (sorosan kapcsolva, hogy a rövidzárrá változástól védekezzük), amely biztosítja, hogy a mikrokontroller nem tud azon a lábán feltápolódni. Mivel mindegyik funkcionális egységből két példány fog elhelyezkedni az áramkörben, így valamely dióda szakadássá változása esetén az adott egység funkcióvesztéssel esetleg üzemelhet tovább, de a másik egység funkcióvesztés nélkül ép marad.

2.2.2. Kiemelt kapcsolások

Felhasznált passzív alkatrészek

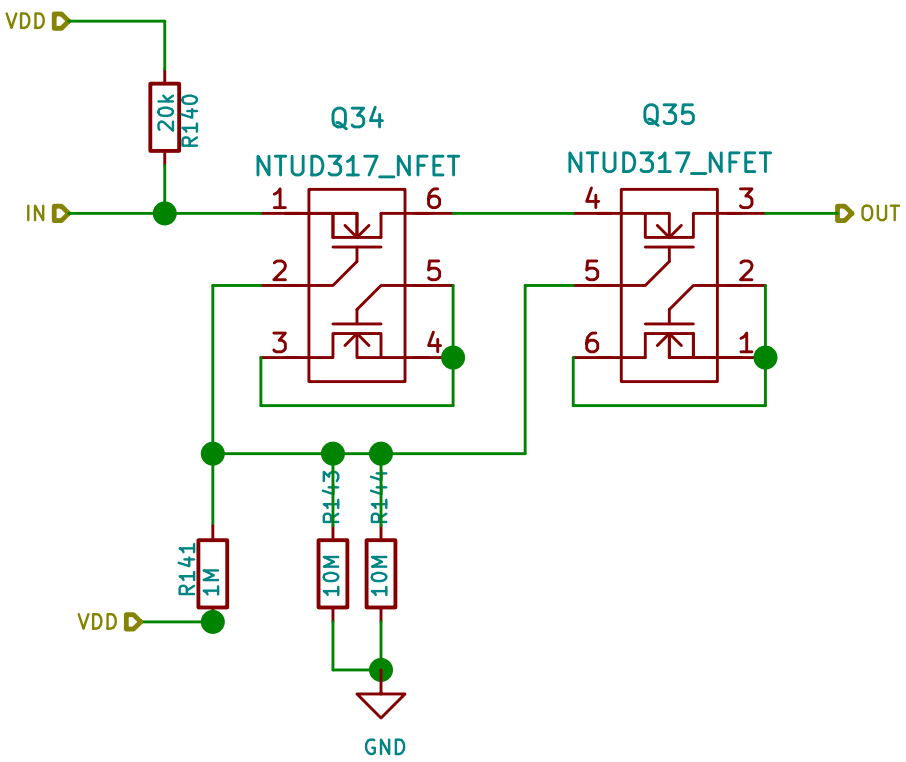
- Ellenállások: mindenhol 0402 (tápellátásnál pedig 0603) méretű Vishay és Panasonic gyártmányú, legfeljebb 1%-os szorású ellenállásokból építkezem.
- Kondenzátorok: szintén 0402 (amiből nincs, abból 0603) méretű, legalább 6.3 V névleges feszültségű, Murata gyártmányú X7R (pontosabb) illetve X5R (kevésbé pontos) típusú kerámiából készült kondenzátorokat tartalmaznak a kapcsolások. (Kivéve, természetesen, a szuperkapacitást, amelyet a Taiyo Yuden készített.)
- FET: jelátvitelhez (de teljesítményátvitelhez nem) az ON Semiconductor által gyártott NTUD317 FET termékcsaládot használom, mert tervezéskor ezek voltak az elérhető leg-

kisebb méretűek (mindössze 1 × 1 mm-es tokban 2 db FET található), amiket még kézzel lehet forrasztani.

- Dióda: a szintén ON Semiconductor által gyártott NSR0x40 dióda termékcsaládot használom az összes jelvezetéken, mert tervezéskor ezek voltak az elérhető legkisebb méretűek (mindössze 0.6 × 1 mm, ami kisebb, mint a 0402-es szabvány), amiket még kézzel lehet forrasztani.

Egyvezetékes kommunikáció védőkapcsolása

Az OBC és több alrendszer (PCU, TID, napelem oldalak) között egyvezetékes half-duplex UART kommunikáció zajlik. Ez azt jelenti, hogy magát a kommunikációt egyetlen jelvezetéken bonyolítják. Ezt a jelvezetéket meg kell védeni többféle problémától is. Egy ilyen védőkapcsolást mutat a 2.2 ábra. Természetesen a kommunikáció mindkét oldalára (tehát az OBC-hez és az adott alrendszerhez is) el kell helyezni egy ilyen kapcsolást.



2.2. ábra. Egyvezetékes kommunikáció védőkapcsolása

„VDD” a mikrokontroller tápfeszültsége, „IN” a mikrokontroller azon lába, amelyen a kommunikáció zajlik, „OUT” pedig az alrendszer felé menő vezeték. Az adott alrendszer esetén: ugyanennek a kapcsolásnak az „OUT” oldala mutat az OBC felé, az „IN” az alrendszerben levő kontroller megfelelő GPIO lába, a „VDD” pedig annak tápfeszültsége.

A kapcsolás gondoskodik a következőkről:

- Az „OUT” vezetéken megjelenő logikai magas szint ne kapcsolja be a kontrollert. Emiatt van n-csatornás FET beiktatva, amely zárva van, ha a kontroller épp nem kap tápfeszültséget. Ennek teljesülnie kell akkor is, ha a FET rövidzárrá változik, ezért van két sorbakapcsolt FET (az ábrán Q34 és Q35).
- Az „OUT” vezetéken legyen logikai magas szint, ha bármely kontroller be van kapcsolva. Emiatt a FET-ek gate-jét és az „IN”-t felhúzó ellenállásokkal láttam el (R141 és R140).
- Kapcsoljon ki a kapcsolás, ha a kontroller is kikapcsol. Ezért van a FET-ek gate-je nagy ellenállással a földre is húzva. A tartalékolás miatt két párhuzamos ellenállás van (az ábrán R143, R144), hiszen akkor is ki kell tudnia kapcsolni, ha az egyik ellenállás szakadássá változik.
- Ha a kontroller táp-föld zárlatot szenved, vagy az „IN” láb tápra vagy földre záródik, nem szabad a túloldalon levő kontrollert tönkretennie. Ezt a célt szolgálja a túloldalon elhelyezett ugyanilyen védőkapcsolás.

Megjegyzések:

- Mivel az egy tokban levő két FET azonos szubsztráton helyezkedik el, ezért meghibásodás szempontjából azok egy alkatrésznek („egy pontnak”) számítanak, vagyis jelen kapcsolásban nem szabad egy tokban levő két FET-et sorbakötni.
- Az egyes tokokból nem használt FET-ek lábait összekötöttem, hogy mechanikailag stabilabbak legyenek a forrasztási felületek (padek).
- Még úgy is, hogy a tokból csak egy FET-et használunk, az NTUD317 volt az elérhető legkisebb méretű, még kézzel forraszható FET.

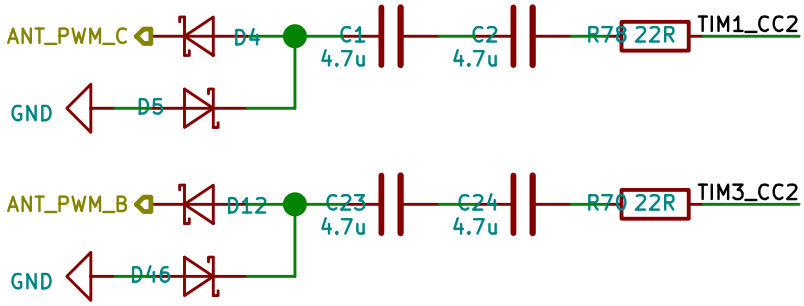
Ha valamelyik FET szakadássá változik, akkor a kommunikáció megszűnik. Ebben az esetben azonban a funkcionális tartalék továbbra is védi a műholdat, hiszen az adott OBC kontroller példány még mindig tud kommunikálni az alrendszer másik példányával, illetve a másik OBC példány továbbra is kommunikálni tudhat mind a két alrendszer példánnyal.

A továbbiakban erre a kapcsolásra „1-wire protection” névvel is hivatkozom.

Antennanyitó vezérlése

A két antennanyitó berendezést („ANT B” és „ANT C”, ld. rendszerterv) az OBC mikrokontrollere vezérli PWM (pulse width modulation) jel segítségével.

Redundancia végett két antennanyitásra alkalmas áramkör is van a fedélzeten, ezek egyike („ANT B”, mint battery) az akkumulátor feszültségét közvetlenül egy ellenállásra vezeti, amely a keletkező hő által elolvasztja az antennákat rögzítő polimerszálat. A másik („ANT C”, mint capacitor) arra az esetre készült, ha az akku vagy az „ANT B” tönkremenne. Ez a szabályozatlan



2.3. ábra. Antennanyitók vezérlő kapcsolása

busz feszültségéről kondenzátorokat tölt fel, és az ezek által tárolt energia segítségével olvasztja el a polimerszálat. Mind a két antennanyitó áramkört az OBC vezérli.

A vezérlő kapcsolásban két egymással sorbakapcsolt kondenzátor található. Így ha ugyanis az egyik kondenzátor rövidzárrá változik, akkor a mikrokontroller jele nem jut közvetlenül a diódára.

Mindkét OBC példány tudja mindkét antennanyitó áramkört vezérelni, ezért szükséges a vezérlő kapcsolás kimenetére is egy diódát rakni, hogy valamelyik OBC példány meghibásodása esetén a másikat ne rontsa el.

A vezérlőkapcsolás nem védett a benne található diódák vagy ellenállás tönkremenetele, valamint a kondenzátor szakadássá változása ellen. Ha ezek valamelyike mégis bekövetkezik, akkor a funkcionális tartalékot, illetve a még ép antennanyitó vezérlőt használjuk.

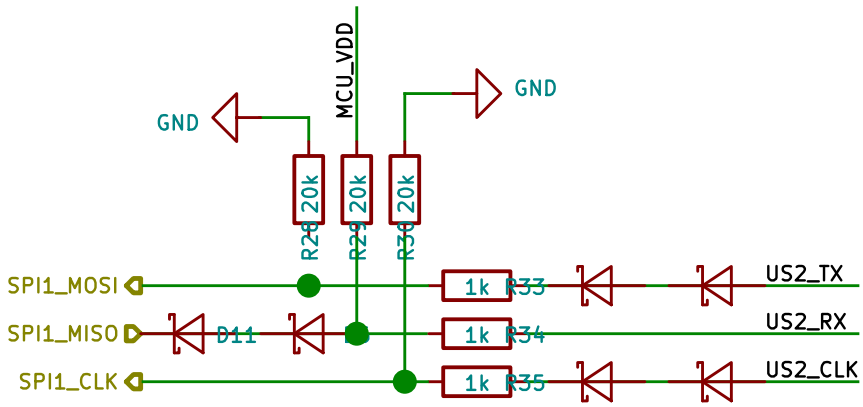
SPI busz védelme

Mivel mind a két OBC példány master módban van bekötve mindkét SPI buszra, valamint a buszokat több periféria is használja (ld. a 2.1 ábra), emiatt szükségessé válik az SPI busz védelme a következő problémák ellen:

- Ha az egyik master használja az SPI buszt, akkor a buszon megjelenő logikai magas szint ne tápolja fel a másik mastert.
- Ha valamelyik master táp-föld zárlatot szenved, attól még a másik számára használható maradjon a busz.
- Ha valamelyik periféria táp-föld zárlatot szenved, attól még a többiek számára használható maradjon a busz.

Ennek a védelemnek a mikrokontroller felőli oldalát mutatja be a 2.4 ábra.

- Táp-föld zárlat bekövetkezése esetén létrejövő túláram ellen véd mindegyik lábon egy-egy áramkorlátozó ellenállás.
- Mindegyik vonalat két dióda védi. Egy nem elég, mert annak rövidzárrá változásakor már nem teljesülne az, hogy a buszon levő logikai magas szint ne tápolja fel a mikrokontrollert.



2.4. ábra. SPI busz védelme a mikrokontroller felől

- Az EFM32WG-ben az SPI kommunikációt is az USART periféria végzi, ezért a mikrokontroller oldalon az USART periféria „RX”, „TX”, „CLK” lábaira van kötve a fenti kapcsolás.
- Az SPI CS vonalakat is védem (ez a teljes kapcsolási rajzon látszik): active high esetben két sorbakapcsolt diódával, active low esetben pedig egy FET-tel.

A periféria felőli oldalon egyszerűbb dolgom van:

- A MISO vonalat egy FET segítségével védem.
- A MOSI, CLK és CS vonalakra elég mindössze egy áramkorlátozó ellenállás, amely biztosítja, hogy zárlat esetén a többi áramkör ne menjen tönkre, illetve a busz használható maradjon számukra.

Áramkorlátozó kapcsolók

A fedélzeti számítógéphez tartozik két darab limiter switch (áramkorlátozó kapcsoló), amelyeket Géczy Gábor tervezett. Ezek a kapcsolók a COM (kommunikációs) rendszer egy-egy redundáns egységét védik és felügyelik. Afféle „okos biztosítóként” egy bizonyos megengedett áramérték felett kioldanak, így megakadályozva, hogy az adott áramkörben egy rövidzárlat energiaveszteséget okozzon vagy a műhold egészét tönkretegye. [3] Ezek a kapcsolók nem csak az automatikus kioldásra jók, hanem ennek megtörténte esetén képesek digitális jelet adni a mikrokontroller felé, valamint áram- és feszültségmérő kapcsolásokkal is el vannak látva, amelyektől az OBC telemetriát gyűjt.

Az OBC-nek feladata két ilyen kapcsoló vezérlése és tőlük áram- és feszültségtelemetria mérése (analóg-digitál átalakítók segítségével) és rögzítése. Természetesen a fedélzeti számítógép mikrokontrollerei (a redundáns pár mindkét tagja) is ilyen kapcsolókkal vannak megvédve, de azokat a kapcsolókat az EPS vezérli (és az EPS panelen foglalnak helyet).

2.2.3. Mikrokontroller

A fedélzei számítógép „lelke” egy EFM32WG családba tartozó „Wonder Gecko” típusú mikrokontroller, melyet a Silicon Laboratories (Silabs) gyárt. Ez tartalmaz egy saját kategóriájában jónak mondható ARM Cortex-M4 processzort (CPU) és megfelelő mennyiségű operatív memóriát (RAM).

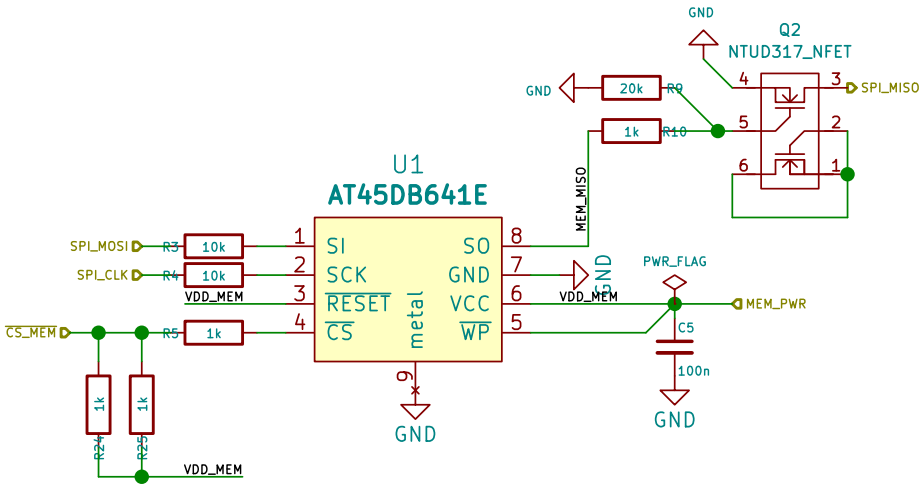
(*Megjegyzés.* A tervezés korai fázisában egy PIC24 családba tartozó mikrokontrollert választottam a feladathoz, azonban világossá vált, hogy annak számítási kapacitása alkalmatlan lenne a kódolási és titkosítási feladatok megvalósítására.)

A mikrokontroller feladata, hogy az OBC fő szoftverét futtassa. Ez a szoftver vezérli az összes többi alrendszert (az energiaellátó rendszer kivételével), legfontosabbként említendő a kommunikációt, valamint gyűjt majd telemetriaadatokat az egész műhold összes alrendszere felől (beleértve az EPS-t és a napelem oldalakat is). Lásd bővebben a szoftver felépítéséről szóló fejezetet.

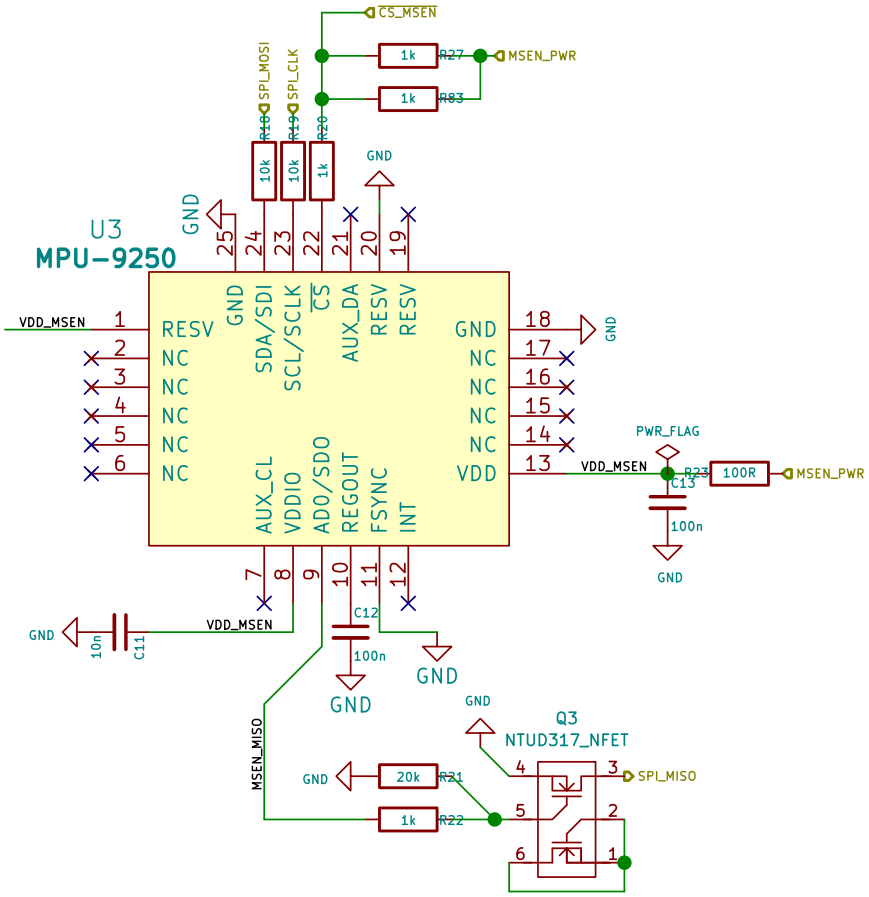
A mikrokontroller órajelgenerátora egy precíziós hőmérséklet-kompenzált kristály oszcillátor (TCXO), amelyet a TAI-SAW gyártott. Ez ugyanaz a típus, amely a MASAT-1 fedélzetén már sikeresen teljesített.

Bizonyos perifériák tápellátását a mikrokontroller GPIO lábairól oldottam meg (két diódán keresztül, hogy redundáns legyen). Így bármelyik mikrokontroller tud tápfeszültséget adni bármelyik perifériának úgy, hogy meghibásodás esetén sem megy tönkre a másik. Gondoskodtam arról, hogy a GPIO tud elegendő tápáramot biztosítani, illetve arról is, hogy akkor sem ég le a mikrokontroller, ha a GPIO lábra logikai magas szintet kapcsolok, de a periférián már táp-föld zárlat van.

Használom a mikrokontroller analóg perifériáit is, például mérem az áramkorlátozó kapcsolókon átfolyó áram értékét, és két darab, az EPS2 panelen levő hőmérőt, amelyek analóg-digitál konverziójára a PCU-n nincs lehetőség. (Ezen kívül mérem a mikrokontroller saját tápfeszültségét és áramát.)



2.5. ábra. Flash memória kapcsolása



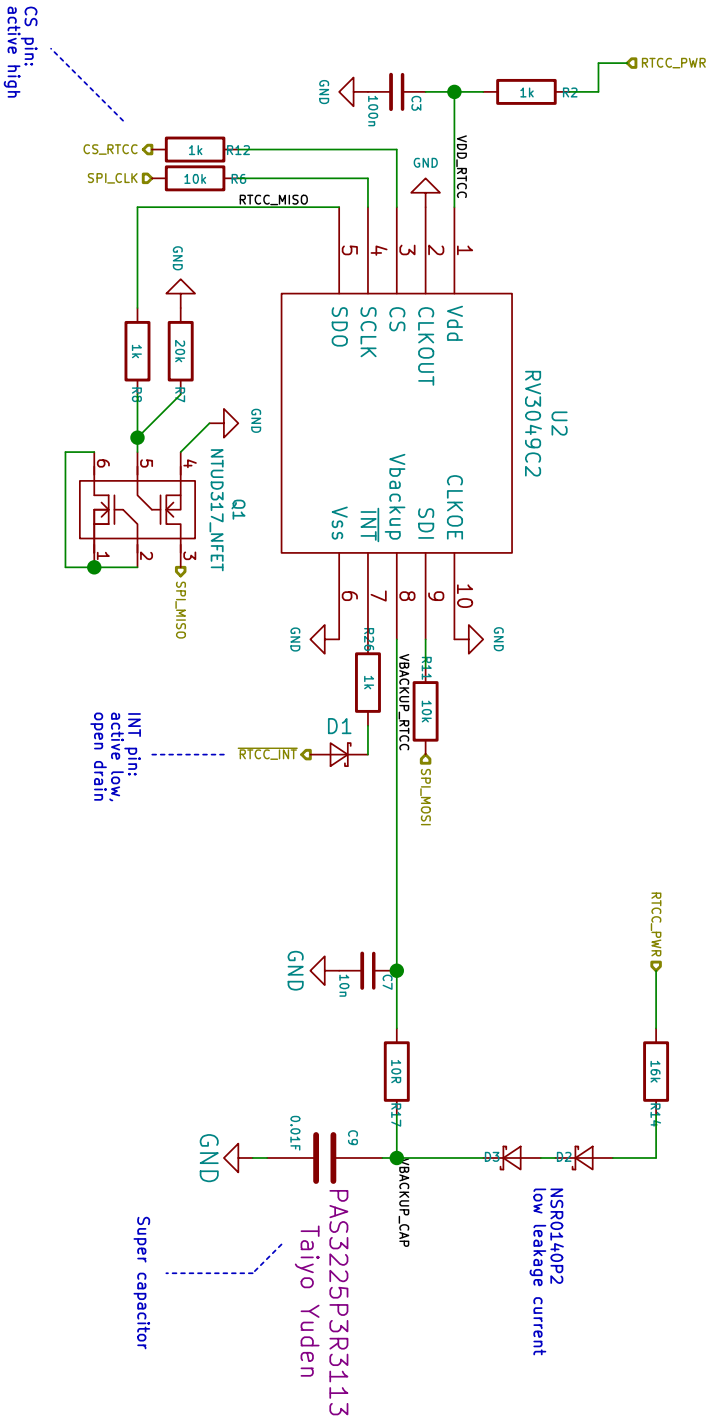
2.6. ábra. MPU kapcsolása

2.2.4. Flash memória

Küldetéséből kifolyólag a SMOG-1 mérőrendszere folyamatosan adatokat generál. Egyrészt a spektrumanalizátor kimenetét (ld. bevezető / küldetés), másrészt a műhold különféle alrendszereinek telemetriaadatait. Fontos tehát, hogy legyen egy megbízható háttértár, ami kellő kapacitással rendelkezik. Szóba jönnek az EEPROM modulok és a flash memóriák. Lényeges szempont, hogy a memória ne fogyasszon sokat és lehetőleg a mikrokontrollerrel azonos (vagy tágabb) feszültségtartományban üzemeljen. Arra nincs szükség, hogy hosszú ideig tároljunk sok adatot a műholdon (a tárolt adatokat akár naponta többször is letölthetjük róla), de a mikrokontroller beépített memóriája kevés.

A választás az Adesto által gyártott AT45DB641E chipre esett, amely 64 Mbit (8 Mbyte) tárterülettel rendelkezik és képes az 1,7 - 3,6 V-os tartományban működni. [32] További modulok, amiket találtunk, csak lényegesen nagyobb feszültségen (többnyire 3 V körül) kezdtek működni, vagy lényegesen szűkebb tartományban (pl. 1,65 - 1,95 V).

A 2.5 ábrán látható az AT45DB641E chip, a gyártó ajánlása szerint bekötve. Láthatjuk az ábrán az előzőekben tárgyalt, SPI busz védelmére szolgáló FET-et és áramkorlátozó ellenállásokat is. A gyártó nem kéri (de nem is tiltja), én viszont a mechanikai stabilitás növelése érdekében úgy döntöttem, hogy a chip alján található forraszfelületet (amelyet általában ther-



2.7. ábra. RTCC kapcsolása

mal pad-nek hívnak, de az Adesto „metal pad”-nek nevez) a földre kötöm. A flash memória tápellátását a mikrokontroller egy GPIO lába biztosítja két diódán keresztül. A mikrokontroller pár bármelyike tud tápfeszültséget adni bármelyik flash memóriának.

MPU (motion processing unit)

Helyet kap a fedélzeti számítógép részei között egy olyan szenzor is, amely képes három tengely mentén gyorsulást, szögsebességet és mágneses térerősséget mérni.

Választásunk az Invensense MPU-9250 [34] termékére esett. Főbb szempontjaink a szenzor minél kisebb mérete és ehhez képest minél nagyobb tudása voltak. Ezt a szenzort használjuk többek között arra, hogy adatokat gyűjtsünk arról, mikor milyen mozgásállapotban van a műhold. Ez az információ jelterjedési szempontokból is érdekes. (Pl. segítségünkre lehet az ún. „spin fade”, vagyis a forgás miatti jelveszteség jelenségének vizsgálatában.)

A 2.6 ábrán látható az MPU-9250 chip, a gyártó ajánlása szerint bekötve. Láthatjuk az ábrán az előzőekben tárgyalt, SPI busz védelmére szolgáló FET-et és áramkorlátozó ellenállásokat is. Mechanikai stabilitás végett ennek a thermal pad-jét is a földre kötöm. Az MPU tápellátását a mikrokontroller egy GPIO lába biztosítja két diódán keresztül. A mikrokontroller pár bármelyike tud tápfeszültséget adni bármelyik MPU-nak.

RTCC (real time clock and calendar)

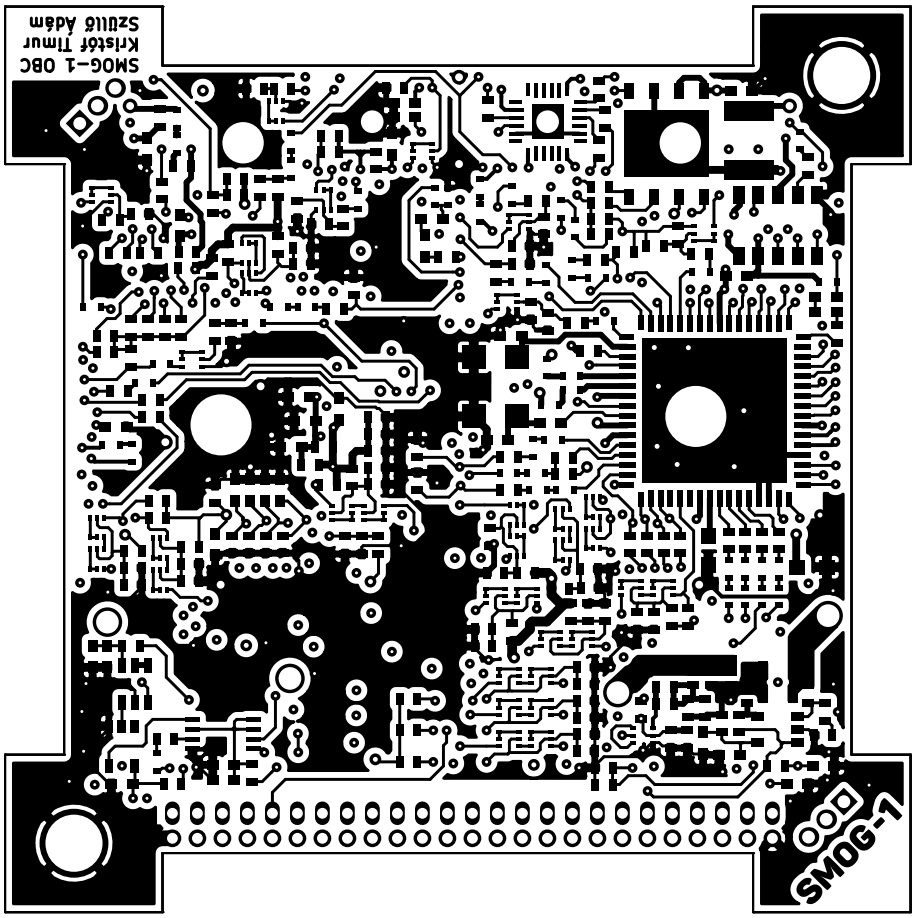
A svájci Micro Crystal cég RTCC megoldását, az RV3049-et [33] választotta a SMOG-1 csapat. Felmerülhet a kérdés, hogy miért nem használtuk valamely mikrokontrollerbe épített RTCC megoldást. Ezek a megoldások korántsem elég precízek, valamint redundancia okokból ilyenkor a fedélzeti számítógép mindkét redundáns egységének egyszerre kellene működnie, hogy az egyikük meghibásodása esetén a másik se veszítse el a pontos időt. Ehelyett inkább egy rendkívül alacsony fogyasztású áramkört használunk, amelynek redundáns energiaellátást is bizosítunk 11 mF-os szuperkapacitások segítségével. [3]

A 2.7 ábrán látható az RTCC modul. Láthatjuk az ábrán az előzőekben tárgyalt, SPI busz védelmére szolgáló FET-et és áramkorlátozó ellenállásokat. Rendelkezik redundáns tápegységgel is, amelyet egy szuperkapacitás valósít meg. Arra az esetre, ha a műhold akkumulátora tönkremenne, az RTCC energiaigényét ez is képes ellátni. A szuperkapacitást diódákkal védjük, hogy táp-föld zárlat esetén ne sülhessen ki.

2.3. Nyomtatott huzalozás

Az OBC nyomtatott huzalozását Szüllő Ádámmal (aki szintén a Mikrohullámú Távérzékelés Laboratórium munkatársa) közösen terveztük. Amint a 2.8 (top oldal) és a 2.9 (bottom oldal) ábrán is látható, a nyomtatott huzalozás kialakítása során arra törekedtünk, hogy a terv szimmetrikus legyen, mert akkor egyrészt jobban átlátható, másrészt könnyebb a beültetése is.

Tervezési szempontjaink a következők voltak:

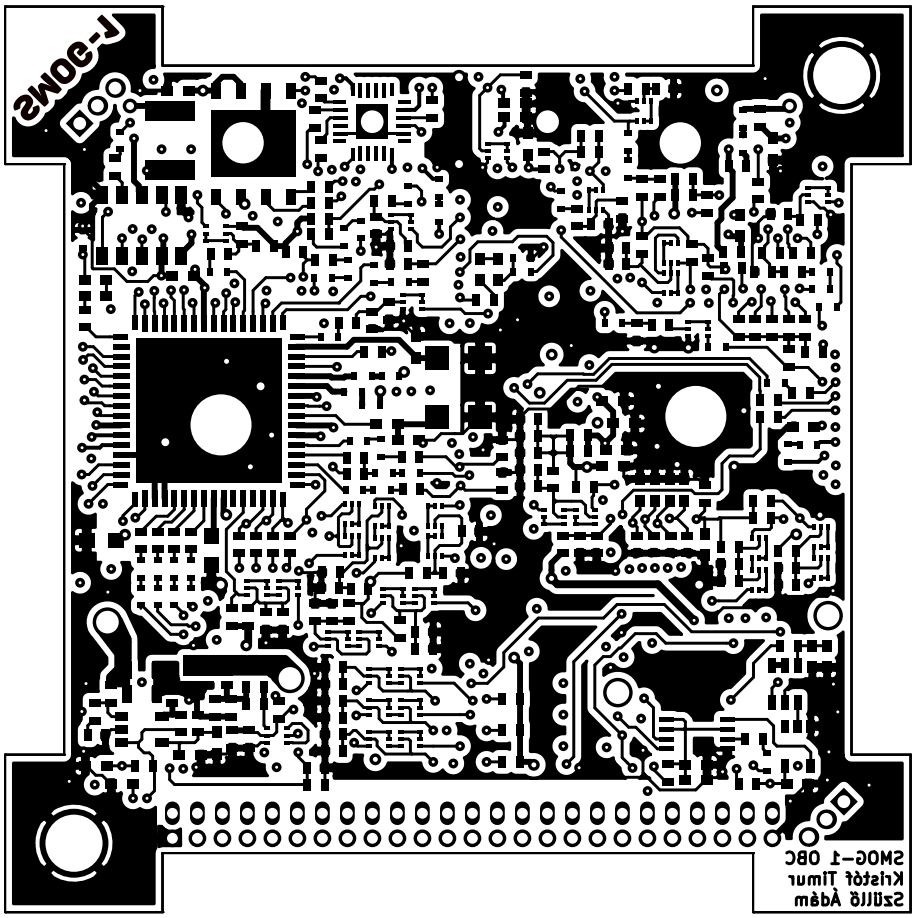


2.8. ábra. OBC nyomtatott huzalozási terve (top oldal)

- Mind a két mikrokontroller egymástól függetlenül programozható legyen.
- A teljes panel kézzel beültethető legyen. Például ezért van a QFN (quad flat no-lead) tokozású alkatrészek alatt nagy fémezett falú furat (via), mert annak segítségével ezen QFN tokozású alkatrészek alján található ún. thermal pad beforraszthető.
- A nagyobb áramot szállító vezetékek minél vastagabbak legyenek, pl. a COM tápfeszültségét szabályzó áramlimiter be- és kimenete (1 mm), az OBC tápfeszültségét szállító vezeték (0.5 mm), és a perifériák tápfeszültségeit szállító vezeték (0.3 mm).
- A jelvezetékek és ezek viái a technológia által megengedett legkisebbek.
- Minden alkatrészből a lehető legkisebb, mechanikailag legstabilabb, még kézzel forrasztható tokozást választottuk. (Például a QFP tok leszakadna a rázkódás hatására, ezért QFN-t használunk.)

2.3.1. Technológia

A nyomtatott huzalozású lemezeket a Eurocircuits cégnél készíttettük, ezért az ő technológiai korlátaikhoz [39] igazítottuk a lehetőségeinket.



2.9. ábra. OBC nyomtatott huzalozási terve (bottom oldal)

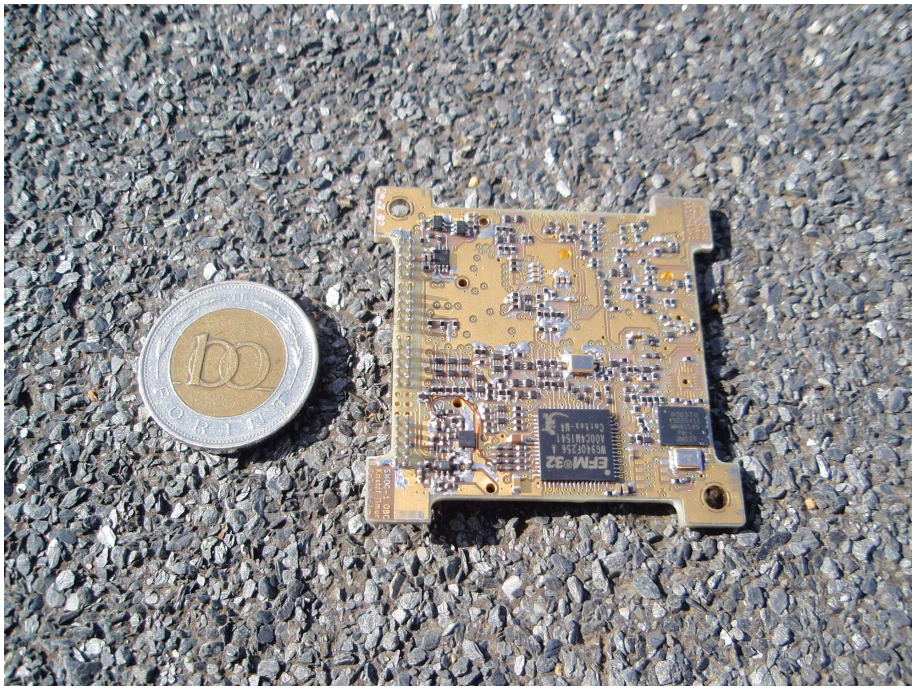
Sem ólommentes forrasztási technológiát, sem gépi forrasztást az űriparban általában nem használják, mert:

- Az ólommentes forrasztás alkalmatlan az űreszközök mechanikai és termikus követelményeinek kielégítésére.
- Az ólommentes forraszanyag a leghajlamosabb whiskerek képződésére. [40]
- A gépi forrasztás általában véve (a fogyaszói cikkek árának alacsonyan tartása érdekében) inkább spórol a forraszanyaggal, nem célja tartós és stabil kötés kialakítása.

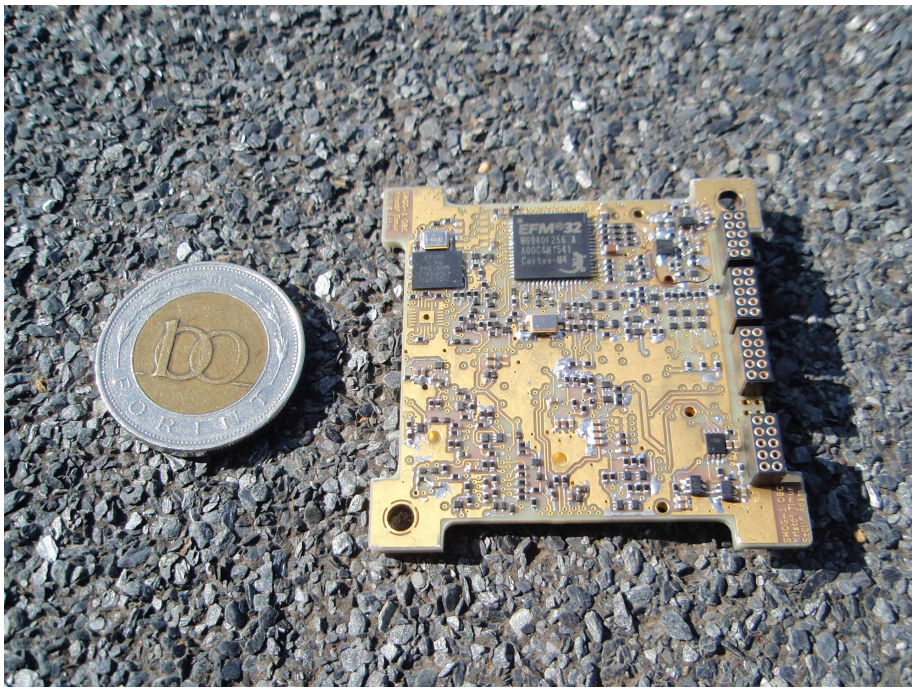
A fenti megfontolások alapján a következőre jutottunk:

- Minden alkatrészt kézzel ültetünk be az áramkörbe ólmos (a régi hagyományos ón-ólom eutektikum) forraszanyaggal.
- Az áramköri lemezeket aranyozott bevonattal kérjük, mert az aranynak alacsony hajlandósága van whiskerek képződésére. [40] Az aranyréteget feloldó (degolding) eljárásra nincs szükség.
- Forradásgátló maszkot nem kérrünk, mert anyaga vákuumban hajlamos kipárolgásra.

2.3.2. Kész hardver



2.10. ábra. OBC panel mérnöki példány (top oldal)



2.11. ábra. OBC panel mérnöki példány (bottom oldal)

Harmadik fejezet

Fedélzeti számítógép szoftvere

3.1. Működési elv

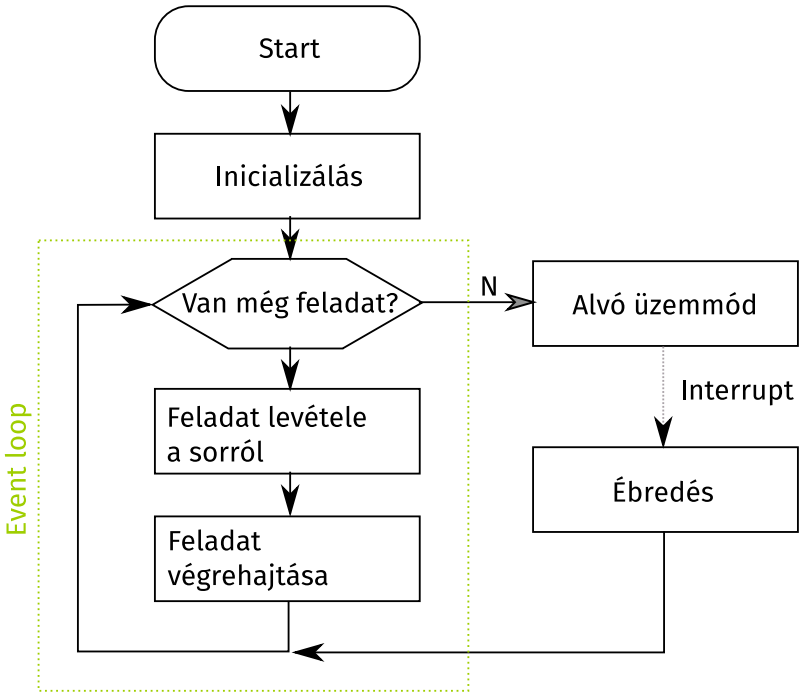
A fedélzeti számítógépet működtető szoftver az „event loop” mintát követi, ami az eseményvezérelt programozás egyik alappillére. [18] Ennek lényege a következő: az OBC (on-board computer, fedélzeti számítógép) egy FIFO queue-t (várakozási sort) tart fenn az elvégzendő feladatok számára (FIFO: first in first out, vagyis a feladatok a beérkezés sorrendjében végződnek el). Valamely esemény bekövetkeztekor mindössze csak rögzíti, hogy újabb dolog vár feldolgozásra, aztán folytatja az addigi teendőit. Amikor pedig éppen kiürült a queue, akkor alvó üzemmódba helyezi magát a következő esemény bekövetkeztéig.

3.1.1. Event loop

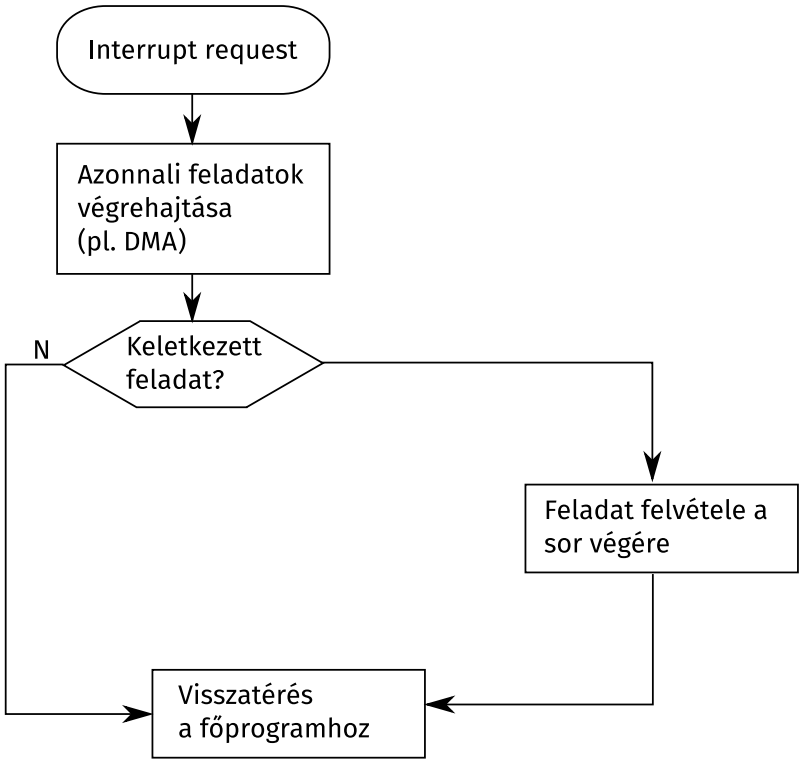
A gyakorlatban az event loop elv azt jelenti, hogy a processzor folyamatosan egy végtelen ciklust futtat, ami ellenőrzi, hogy van-e elvégzendő feladat. Ha nincs, akkor a processzor átvált alvó üzemmódba és így várakozik egészen addig, amíg valamilyen számára érdekes esemény be nem következik. Ez úgy valósul meg, hogy valamelyik periféria megszakítás kérést (interrupt request) küld a processzornak [19], amitől az felébred. Perifériák alatt most nem csak a mikrokontrollerbe integráltakat értem, hanem az összes további egységet, amely kapcsolatban áll még a fedélzeti számítógéppel.

A főprogram egy queue-ban tartja nyilván az elvégzendő feladatokat. Amikor bekövetkezik egy esemény, a processzor, ha aludt, felébred és lefut az adott eseményhez tartozó megszakításkezelő rutin (interrupt service routine, ISR), és jelzi a főprogram számára az adott esemény bekövetkeztét, oly módon, hogy felvesz egy feladatot a már említett queue-ba. A főprogram algoritmusa tehát a 3.1 ábra szerint működik, egy megszakításkezelő rutin pedig a 3.2 ábra szerint.

Előfordulhat, hogy az ISR-nek „azonnali” feladatot kell végrehajtania. Ezalatt olyan tennivalókat értek, amiknek muszáj az interrupt bekövetkeztekor rögtön megtörténniük és nem érnek rá addig, amikor a végrehajtás visszatér főprogramba. Ilyen például a DMA folyamatok vezérlése (ld. később), amelyek elmulasztása hibás működést von maga után.



3.1. ábra. Event loop vázlata



3.2. ábra. Megszakításkezelő rutin vázlata

Ha új feladat kerül az event loop-ba, akkor mindig a sor végére kerül. Az event loop egy-egy feladat befejezte után minden esetben a sor elejéről leveszi az éppen következő feladatot és azt hajtja végre. Ha viszont nincs több elvégzendő feladat, akkor a program alvó üzemmódba állítja a processzort. Ez egy olyan működés, amikor a központi feldolgozóegység le van kapcsolva, viszont a mikrokontrollerbe épített többi periféria még működik. (Lásd bővebben az energiatakarékosságról szóló fejezetet.) Ezzel a módszerrel elérhető például többek között az is, hogy az UART periféria, amikor külső egységtől üzenetet kap, felébressze a processzort, ami elkezdheti feldolgozni a beérkező üzenetet. (Erre részletesebben kitérek a fedélzeti számítógép többi egységgel való kapcsolódásáról szóló fejezetben.)

(*Megjegyzés.* Korábban az event loop-ot egy bonyolultabb koncepció szerint prioritásos sorral valósítottam meg. Ez azonban több problémát vetett fel, mint amennyit megoldott, és a gyakorlatban a tesztelések során szükségtelennek bizonyult. Mivel nincsenek egyszerre hosszú időt igénybevevő feladatok, illetve nincs egyszerre sok elem a sorban, ezért egy FIFO használata is elegendő, és ezáltal a megoldás áttekinthetőbb és egyszerűbb is lett.)

3.1.2. Energiatakarékosság

Az energiatakarékosságot sokan hardveres kérdésnek tekintik, azonban azonok a hardvereken, melyeken szoftver fut, bonyolódik a helyzet, mert a hardver hiába teremt lehetőséget különféle energiamegtakarító módszerek alkalmazására, ha a szoftver nem használja ki ezeket, akkor semmit nem érnek.

Eseményvezéreltség jelentősége az energiatakarékosságban

Klasszikus példa a szoftvertechnológiában a „busy wait” és a „sleep wait” közötti különbség. A „busy wait” esetben a szoftver, amikor vár valamilyen esemény bekövetkeztére, egy végtelen ciklust futtat, amely folyamatosan ellenőrzi, hogy az esemény bekövetkezett-e már (ez a módszert „polling” néven is ismert). Ellenben a „sleep wait”, amennyiben a várt esemény még nem következett be, alvó üzemmódba helyezi a processzort (vagy ha a szoftver egy operációs rendszer alatt fut, akkor átadja az operációs rendszernek a vezérlést), és egészen addig nem fut tovább egyáltalán, amíg valamilyen módon jelzést nem kap, hogy az esemény bekövetkezett. Hardverközeli esetben ilyen lehet egy interrupt. [19] [20]

Tehát a busy wait esetben a program folyamatosan utasításokat ad a processzornak, míg a sleep wait esetében a processzor akár alhat, akár más feladatokat hajt végre. A sleep wait megközelítés megfelel az úgynevezett Hollywood elvnek (amely úgy szól: „majd mi hívunk, te ne hívj minket”). [18]

Természetesen olyan helyeken, ahol a fogyasztás másodlagos (például asztali számítógépben) megengedhető a busy wait jellegű megközelítés, de manapság a hordozható eszközök terjedésével az energiakészlet szűkössége miatt egyre inkább előtérbe kerül a fogyasztás. Egy átlagos mai számítógép (és okos eszköz, pl. telefon vagy tablet) processzora normál használat mellett is az ideje nagyrészét valamilyen készenléti vagy alvó üzemmódban tölti.

Egy zsebműhold olyan rendszer, ahol minden egyes milliamper számít [3] [4], tehát a busy wait megközelítés alkalmazhatósága teljességgel kizárt. Szerencsére az eseményvezérelt programozás lehetővé teszi a processzor alacsonyabb energiafogyasztású üzemmódjainak használatát.

A fedélzeti számítógépben használt Silicon Laboratories „Wonder Gecko" EFM32WG mikrokontroller is többféle energiatakarékos üzemmóddal rendelkezik. Ezek elnevezése EM1, EM2, EM3 és EM4. [24] (az EM itt az „energy mode" rövidítése.) Ezen üzemmódokba a szoftver a kontroller energiamenedzselő egységének (energy management unit, EMU) használatával juthat el. [24] [21]

- EM1 módban lekapcsol az EFM32WG-ben levő processzormag és csak a perifériák működnek.
- EM2 módban a mikrokontroller lekapcsolja a nagyfrekvenciájú belső oszcillátorát is, így az arról működő perifériák is megállnak.
- EM3 módban már csak az néhány alacsony órajelről működő belépített periféria marad bekapcsolva, ezek azonban továbbra is képesek lehetnek felébreszteni a processzort.
- EM4 módból már csak reset (teljes újraindítás) segítségével lehet kijutni, amelyet például a backup RTC (real time counter) idézhet elő.

Természetesen lehetőség van az EFM32WG órajelének megváltoztatására is, és a processzormag alacsonyabb órajelen kevesebb energiát fogyaszt. Ez a módszer azonban kétélű, hiszen alacsonyabb órajelen ugyanazokat a műveleteket tovább tart elvégezni, tehát a processzor kevesebb időt tölthet energiatakarékos módban. Hiába csökken a pillanatnyi fogyasztás, az átlagfogyasztás nőhet is. [21] [26]

Egyes mérések [21] szerint jobban megéri futás közben a processzort magas órajelen működtetni, hogy a feladatait hamar befejezze és minél több időt tölthessen alvó üzemmódban, mert így lehetséges az átlagfogyasztást a lehető legalacsonyabbra csökkenteni.

Sajnos az OBC számára jelenleg csak az EM1 mód áll rendelkezésre, mert ennél alacsonyabb energiaszintű üzemmódokban leáll a COM rendszerrel kommunikáló UART periféria. Ezt nem engedhetem meg, mert a COM bármikor (aszinkron módon) küldhet üzenetet az OBC felé.

Energiatakarékosság a CPU tehermentesítésével: DMA

Bizonyos esetekben azzal spórolhatjuk meg a legtöbb energiát, ha a processzort úgy tehermentesítjük, hogy bizonyos feladatokat specializált hardverekre bízunk. Ilyenkor a processzormag akár alvó módban is töltheti az idejét, miközben a specializált hardveregység ugyanazt a feladatot gyorsabban és alacsonyabb fogyasztással tudja elvégezni. Ilyen feladat a legtöbb I/O (input / output) művelet is, mint például az adatok mozgatása a memóriában vagy a memória és a perifériák között.

Erre a feladatra találták ki a közvetlen memóriaelérést (direct memory access, DMA), amely lehetővé teszi, hogy különféle hardverelemek egymás között a processzor beavatkozása nélkül

mozgathassanak adatokat. Ennek eszköze egy ún. DMA vezérlő. A CPU-nak mindössze közölnie kell a DMA vezérlővel, hogy mennyi adatot, honnan és hová kell mozgatni, az pedig teszi a dolgát, miközben a CPU más feladatokkal foglalkozhat vagy alvó üzemmódba helyezheti magát. A DMA vezérlő, ha elkészült a feladattal, megszakításkérés (interrupt request) segítségével jelzi azt a processzornak. [25]

Az EFM32WG-ben is található DMA vezérlő, amely egyszerre akár több adatmozgatást is képes több csatornán végezni (megfelelő időosztással az egyes csatornák között). [24] [23] Ezt a funkciót használja ki a SMOG-1 fedélzeti számítógépe is a perifériákkal való kommunikáláskor.

Alacsony fogyasztás elérése soros port használatakor

Az EFM32WG soros port vezérlő perifériái képesek kihasználni a DMA vezérlőt, tehát az onnan jövő adatok mozgatásához nincs szükség a CPU beavatkozására. [24] [22]

Ezen kívül azonban az egyik periféria, az ún. LEUART (low energy UART, kis fogyasztású soros port) rendelkezik még egy hasznos funkcióval, ez pedig a frame detection (keret érzékelés) [22], ami azt jelenti, hogy bizonyos előre beállítható adatbájtokat képes érzékelni a hardver, és ezek érkezésekor képes megszakításkérést küldeni a processzor felé.

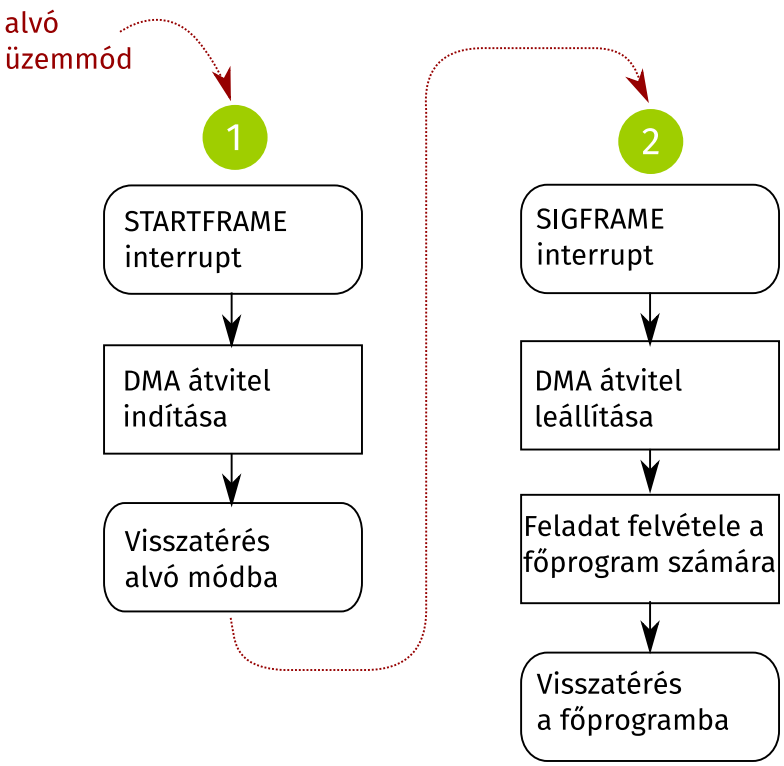
Két ilyen detektálható bájt állítható be:

- Start frame: egy üzenet kezdetét jelző bájt.
  Érkezésekor a STARTFRAME interruptot küldi nekünk a soros port.
- Signal frame: egy üzenetben valamilyen jelentéssel bíró bájt, például az üzenet befejeztének felismerésére használható.
  Egy ilyen bájt beérkezésekor a soros porttól SIGFRAME interruptot kapunk.

A fentiek segítségével olyan soros porti kommunikáció valósítható meg, amely a lehető legkevésbé igényli a processzor beavatkozását. Ennek igénybevételéhez az alábbiakra van szükség:

1. Soros port konfigurálása
   Beállítjuk a soros port vezérlőjében a start frame-et és a signal frame-et.
2. DMA vezérlő konfigurálása
   Létrehozunk egy DMA csatornát a soros porti adatok küldésére és még egy csatornát a soros portról érkező adatok fogadására. A DMA vezérlő üzemmódjai [23] közül ebben az esetben a „Basic” üzemmódra van szükség.
3. DMA vezérlő bekapcsolása
   Órajelet adunk a DMA vezérlőnek és engedélyezzük a működését.
4. Soros port bekapcsolása
   Órajelet adunk a soros port vezérlőjének és engedélyezzük a működését.

A kommunikáció beállítása után használatba is vehetjük azt. Ezt a 3.3 ábra mutatja.
A folyamat tehát a következő:



3.3. ábra. Alacsony fogyasztás megvalósítása soros porton.

1. A soros port észleli az üzenet elejét jelző bájt beérkezését, küld egy STARTFRAME megszakítást.

2. A megszakításkezelő rutin elindítja a DMA átvitelt azon a fogadó csatornán, amit az előző oldal leírása szerint beállítottunk.

3. A processzor visszatérhet alvó üzemmódba (vagy folytathatja egyéb elfoglaltságait, ha voltak).

4. A soros port észleli az üzenet végét jelző bájtot, és küld egy SIGFRAME interruptot a CPU felé, ezzel felébresztve azt, ha aludt.

5. A megszakításkezelő rutin leállítja az előzőleg indított DMA átvitelt.

6. A megszakításkezelő rutin feladatot ad a főprogram számára azáltal, hogy felveszi azt a korábbiakban taglalt prioritásos sorba.

Ez a megoldás nagyságrendekkel energiatakarékosabb a szokásos naiv megvalósításnál, amelyben a processzor busy wait jelleggel folyamatosan figyeli a soros porton beérkező adatokat és vár, amíg az összes adat meg nem érkezett. Az energiatakarékos megoldás nem csak azt teszi lehetővé, hogy a CPU aludjon, miközben az üzenetet a soros port fogadja, hanem azt is, hogy közben akár egyéb feladatokat lásson el.

A LEUART-nak is megvannak a maga korlátai, ezért kb. 10 kbit / sec feletti adatsebességekhez már nem ajánlják. Így a COM rendszert nem LEUART-on keresztül vezérli az, OBC, viszont a többi alrendszert igen:

- TID (total ionizing dose sensor, vagyis a sugárdózis mérő egység)
- PCU (power control unit, ami az EPS programozott logikáit valósítja meg)
- Napelem oldalak

Fordítás, optimalizálás

Az OBC szoftverét a Silicon Laboratories által is preferált GCC (GNU Compiler Collection) fordítóprogrammal fordítom le, amely számos lehetőséget ad a készült kód optimalizálására. Az optimalizálás nem csak a sebesség, hanem ennek következményeképp az energiatakarékosság miatt is fontos, hiszen ha a feladatait gyorsabban végzi el a szoftver, akkor a processzor több időt tölthet alvással.

Azonban az optimalizálás veszélyes is lehet. Körültekintőnek kell lenni, mert a modern optimalizációs módszerekkel rendelkező fordítóprogramok olyan lehetőségeket használnak fel a C / C++ nyelvek specifikációiban, amelyek gyakran a programozó által nem várt mellékhatásokkal járnak. [27] Nem várt következményeket elkerülendő a SMOG-1 OBC szoftvere csak „debug optimalizálással" lesz fordítva. Ilyenkor a fordító csak olyan optimalizálást hajt végre, ami nem zavarja a debugolást, tehát könnyen tesztelhető marad vele a program.

3.1.3. Analógia operációs rendszerekkel

Párhuzamot találhatunk az OBC szoftvere és egy operációs rendszer között. Az előbb részletezett event loop megvalósítás tulajdonképpen úgy működik, mint egy nagyon kezdetleges ütemező. Ha így nézzük, akkor az elvégezendő feladatok processzeknek, az OBC szoftvere pedig egyfajta operációs rendszernek fogható fel. Hardveres korlátok miatt nem lenne gazdaságos ennél bonyolultabb rendszert kialakítani, mert az EFM32WG processzorában nincs sem MMU (memory management unit), sem elég operatív memória ahhoz, hogy érdemes legyen több folyamatot párhuzamosan futtatni. [24]

Használhatnánk valamelyik ún. valósidejű operációs rendszert (real-time operating system, RTOS), ezt azonban nem tesszük. Mivel a műhold szoftverének nagy megbízhatóságúnak kell lennie, semmilyen olyan szoftverelemet nem kívánunk feljuttatni, amit nem mi fejlesztettünk ki, mert third party szoftverelemek ellenőrzése és hibajavítása nehezen megvalósítható feladat. Valamint a feladat komplexitása sem indokolja bonyolultabb rendszer használatát.

Természetesen lehetne preemptív multitaszkingot megvalósítani, úgy, hogy context switch esetén az operatív memória teljes tartalmát átmásoljuk a flash memóriába, de ez mikrokontrolleres hardveren lassú és gazdaságtalan folyamat lenne.

3.2. Szoftver megvalósítása

3.2.1. Felhasznált alap szoftverek

A szoftver alapját a Silicon Labs által az EFM32WG chiphez adott „ARM CMSIS" library, egy harveres alapfunkciókat támogató „emlib" library és driverek képezik. Ezen szoftverelemek forrása szabadon hozzáférhető és módosítható is, valamint kellően egyszerűek ahhoz, hogy a működésüket én is átlássam, és ha szükséges, rajtuk kisebb javításokat eszközöljek vagy funkciókat adjak hozzá. Ezen kívül készítettem néhány saját drivert (a flash és az MPU szenzor számára, valamint az UART, LEUART perifériák számára is).

3.2.2. Implementáció nyelve

Az alacsony szintű elemeket C nyelven írtam, mert:

- Jól kézre áll a közvetlenül a hardvert vezérlő programkódok esetén.
- Nem kell bonyolult absztrakciókat megfogalmazni.
- Nem túl bonyolult szoftverelemek egyszerűen átláthatóak C nyelven.

A komplexebb funkciókat további, magasabb szintű szoftverelemek látják el, amelyek a hardvert már nem közvetlenül, csak a drivereken keresztül érik el. Ezeket a szoftverelemeket újrafelhasználhatóan írtam meg, és a programban több helyen is használom őket.

A magasabb szintű elemeket C++ nyelven írtam, amely ezen esetekben előnyösebb, mint a C, mert:

- A C++ szigorúbb és biztonságosabb, mint a C. (Szigorúbb a pointer aritmetika, stb.)
- A fenti megszorításoktól eltekintve (majdnem) minden érvényes C kód egyben érvényes C++ kód is.
- Alapvető dolgokat a nyelv automatikusan megtesz, ezzel csökkenti a hibázás lehetőségét. (Pl. konstruktor és destruktor hívása automatikus, stb.)
- Összetartozó függvényeket és adatokat egy osztályba csoportosítva lehetőség van újrafelhasználható kódot készíteni.
- Lehetőség van generikus programozásra (template), amelynek köszönhetően kevesebb a kódduplikáció, tehát csökken a hibázás lehetősége.
- Általában véve könnyebb modulárisabb, és ezáltal tesztelhetőbb kódot készíteni.

Mivel a C++ a beágyazott fejlesztésben kevésbé gyakran fordul elő, ezért alaposan utánajártam, hogy használata mennyire javallott és kik használják (különös tekintettel az űriparra).

Az ESA [37] és a NASA [36] is használ C++-t és rendelkezésre bocsát ezzel kapcsolatos javaslatokat, dokumentumokat, valamint az autóipari szabványokról ismert MISRA [38] is támogatja. Sőt, maga a GCC (amivel a C kódot is lefordítom) is C++ nyelven íródott.

Tehát összességében elmondhatjuk, hogy — habár még nem elterjedt — a C++ nyelv alkalmas erre a feladatra.

Használtam ezen kívül — ahol szükség volt rá — az ARM processzorokhoz való C / C++ nyelvi kiegészítéseket [16] is, amelyeket a GCC fordító is ismer [17].

3.2.3. Irányadó megfontolások

Legfőbb irányadónak a NASA által is [35] használt „power of ten" szabályokat tekintem, amelyeket az alábbiakban ismertetek:

1. Hogy a program menete átlátható maradjon, ne használjunk goto, setjmp, longjmp utasításokat és rekurziót.
2. Gondoskodni kell róla, hogy a program ne „ragadjon be" végtelen ciklusba.
3. Ne használjunk dinamikus memóriakezelést. (Mert heap fragmentációhoz és ezáltal nem várt következményekhez vezethet.)
4. Minden függvény legyen áttekinthető méretű. (A NASA szerint egy nyomtatott oldalnyi a még áttekinthető terjedelem, ezzel körülbelül én is egyetértek.)
5. A függvények feltételezéseiket assert-ként fogalmazzák meg. (Vagyis a program ne haladhasson tovább, ha nem teljesülnek.)
6. Minden adatváltozót a lehető legkisebb scope-ban deklaráljunk. (Ezzel csökkentjük a hibázás esélyét, mert a változókat véletlenül sem tudják máshonnan módosítani.)
7. Minden függvényhívás visszatérési értékét ellenőrizni kell.
8. Az előfordító (feltételes fordítás, makrók) használatát amennyire lehet, mellőzzük. (Mert átláthatatlanná teszi a programot.)
9. A pointerek használatát tartsuk ésszerű szinten. Függvénypointert ne használjunk (hacsak nem muszáj).
10. Minden állományt a fordító legszigorúbb beállítása mellett fordítsunk, sem hiba, sem figyelmeztetés (warning) nem engedélyezett.

Az OBC szoftverében törekszem arra, hogy ezeket a lehető legjobban betartsam. Például a linkert úgy konfigurálom, hogy a malloc, calloc, stb. függvények használatát tiltsa. Így még véletlenül sem lehet a programban dinamikus memóriakezelés. Sajnos eseményvezérelt programozás esetén a függvénypointereket nem lehet teljesen mellőzni, de használatukat korlátozom két helyre: a driverekben, ahol callback-ként van szükség rájuk, illetve az event loop-ban, amely

a következő végrehajtandó feladatokat tartja nyilván. Szerencsére C++ nyelvben rendelkezésre állnak a functorok is, amelyek lényegesen egyszerűbb, áttekinthetőbb kódot eredményeznek, mint a függvénypointerek.

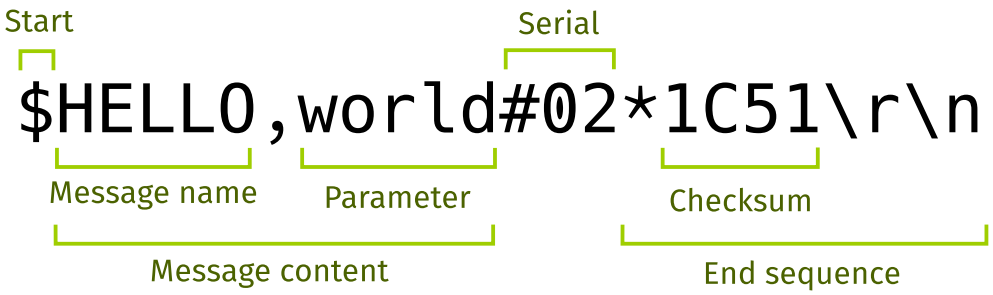
Negyedik fejezet

Fedélzeti számítógép feladatai

4.1. Kapcsolat a többi alrendszerrel

Azt a tervezési elvet választottuk, hogy minden alrendszerben található egy „helyi intelligencia”, vagyis egy kicsi teljesítményű mikrokontroller, ami az adott alrendszert egy egyszerű saját szoftverrel közvetlenül irányítja. Az OBC ezekkel a mikrokontrollerekkel tartja a kapcsolatot UART-on keresztül.

Feladatom volt kidolgozni a protokollt, amellyel az alrendszerek kommunikálhatnak. Az eredmény egy NMEA-hoz hasonló protokoll lett, amely az „S1 Protokoll” nevet kapta. (Az S1 ebben az esetben a SMOG-1 nevét rövidíti.) A 4.1 ábrán egy példa S1 protokoll üzenet felépítését mutatom be.



4.1. ábra. S1 protokoll üzenet felépítése

Így épül fel egy üzenet:

- Start karakter: `$`.
- Üzenet tartalma: az üzenet neve (kötelező) és paraméterei (opcionális) vesszővel elválasztva.
- Sorozatszám: `#` után egy byte üzenetsorozatszám 2 byte ASCII hexadecimális karakterként ábrázolva.
- Befejező szekvencia: `*` után egy 2 byte Fletcher-16 checksum (4 byte ASCII hexadecimális karakterként ábrázolva), végül `\r\n`.

Megjegyzés: a start karakterre és befejező szekvenciára azért van szükség, hogy együtt tudjon működni a hardveres UART frame detection funkcióval.

A protokollt kétféle módon használhatjuk:

- Half-duplex: egyvezetékes összeköttetés esetén. Ilyenkor az OBC a master, és az adott alrendszer a slave, amely csak az OBC üzeneteire válaszol.
- Full-duplex: kétvezetékes összeköttetés. Ilyenkor az OBC egyenrangú a másik alrendszerrel, mindkettő küldhet üzeneteket a másiknak aszinkron módon.

4.1.1. COM (rádiókommunikáció és spektrumanalizátor)

Rádiókommunikációért egy Silicon Laboratories gyártmányú Si1062 integrált áramkör lesz a felelős. Ez a chip tartalmaz a rádiófrekvenciás áramkörön kívül egy beépített 8051-es mikrokontrollert is, amely által „helyi intelligencia" valósítható meg a COM rendszerben. [1]

A spektrumanalizátor egy Si4464 típusú IC, amelyet az Si1062-ben levő mikrokontroller fog vezérelni SPI buszon keresztül. Ebben az elrendezésben tehát a földi állomással való kommunikáció és a spektrumanalizátor az OBC szemszögéből egyetlen egységes egészként viselkedik. A COM és az OBC között full-duplex UART digitális összeköttetés lesz, amelyen spektrum mérési adatokat, földről vett, és a földre küldendő információkat cserélhetnek egymással nagy sebességgel. (Terveink szerint akár száz kilobyte / sec sebességgel.) Ezen kívül a COM tápellátását is az OBC vezérli két áramkorlátozó kapcsoló segítségével (ld. fentebb).

Üzenetek az OBC-től a COM felé:

- `PING`: adatkapcsolat tesztelésére szolgál, amelyre a másik fél `ACKMS`-sel válaszol.
- `SPAST`: spektrumanalízis elindítását kérő üzenet a kért spektrum analízis paraméterekkel (frekvenciatartomány, lépésköz, stb.).
- `SDPKT`: csomag kiküldését kéri a Föld felé.
- `RXMOD`: a COM vételi üzemmódba állítását kéri.

Üzenetek az COM-től az OBC felé:

- `PING`: adatkapcsolat tesztelésére szolgál, amelyre a másik fél `ACKMS`-sel válaszol.
- `SPADN`: jelzi, hogy a kért spektrumanalízis elkészült, és küldi az adatokat.
- `PTXDN`: jelzi, hogy a kért kimenő csomag küldése sikeres.
- `RCVPK`: jelzi, hogy bejövő csomag érkezett a földi állomásról.

4.1.2. PCU (power control unit)

A PCU egy PIC24 családba tartozó mikrokontroller, ami az EPS-hez tartozik. A PCU feladatai közé tartozik a szabályozott energiabusz és az akkumulátor felügyelete, valamit az OBC-t ellátó LSW-k vezérlése, és annak eldöntése, hogy a redundáns pár közül melyik OBC üzemeljen. [3]

Az OBC és a PCU között „one-wire" half-duplex UART digitális összeköttetés van, amelyen telemetriaadatokat és egyéb információkat cserélhetnek egymással. Master-slave viszony valósul meg, vagyis a TID kizárólag az OBC üzeneteire válaszol.

Legfontosabb üzenetek:

- `PINGD`: adatkapcsolat tesztelésére szolgál, amelyre a másik fél `ACKMS`-sel válaszol.
- `HEART`: heartbeat jel a PCU felé, ami jelzi a PCU-nak az OBC helyes működését.
- `RQDEP`, `RQBAT`, `RQSDC`, `RQBUS`: kérés a deployment switch (műhold felbocsátását jelző kapcsoló), akkumulátor, step-down converterek és szabályozott és szabályozatlan buszok mérései adatainak lekérésére.
- `PDBAT`: az OBC által az akkumulátor leválasztásáról szóló döntést közli a PCU-val.

4.1.3. TID (total ionizing dose sensor)

A TID „helyi intelligenciáját" egy NXP gyártmányő LPC812 mikrokontroller valósítja meg. Ezzel van az OBC „one-wire" half-duplex UART digitális összeköttetésben, amelyen master-slave viszony valósul meg, vagyis a TID kizárólag az OBC üzeneteire válaszol.

Lehetséges üzenetek:

- `PINGD`: adatkapcsolat tesztelésére szolgál, amelyre a másik fél `ACKMS`-sel válaszol.
- `RESET`: a TID újraindítását kéri. A TID `ACKMS`-sel válaszol, azután újraindul.
- `START`: dózismérési folyamat elkezdését kéri. A TID `ACKMS`-sel válaszol.
- `RQDAT`: mért adatok lekérése. A TID `MEDAT` üzenetben közli a választ (ha van), vagy `NODAT`-tal jelzi, ha nincs válasz.

4.1.4. Napelem oldalak

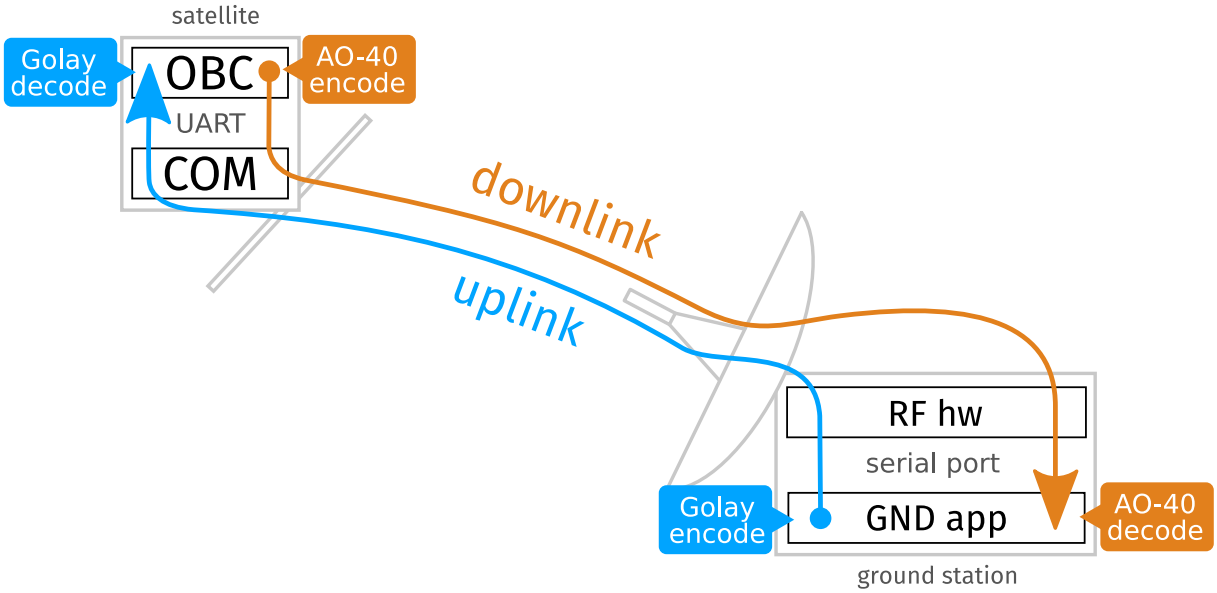
Az OBC-nek adatokat kell gyűjtenie a napelemekről is, viszont az EFM32WG-nek nincs több lehetséges UART portja, amire ré lehetne kötni. Ezesetben azonban nincs szükség bonyolult üzenetváltásra, elegendő csak a napelemről jövő méréseket fogadni bizonyos időközönként. Úgyhogy egy hardveres „trükk" segítségével az EFM32WG-ben található PRS (peripheral reflex system) használatával egy-egy GPIO portot össze tudunk kötni egy UART RX-szel minden napelem oldal számára. (Viszont a PRS nem teszi lehetővé a TX láb összekötését is.)

Ebben az esetben az OBC nem tud ezen a porton üzenetet küldeni, csak GPIO funkcióval „lehúzza” az adott lábat logikai alacsony szintre, vár 300 mikroszekundumot, és fogadja a napelemoldalról jövő `MPPTD` üzenetet, ami minden mérési adatot tartalmaz.

4.2. Adattárolás

Az OBC-hez tartozik 8 Megabyte redundáns flash memória. Ezt a tárterületet használjuk majd a mért adatok és a telemetria tárolására. Amikor a műhold a földi állomással kommunikál, akkor innen olvassa majd ezeket az adatokat és küldi le őket.

4.3. Kapcsolattartás a földi állomással



4.2. ábra. Földi állomás kommunikáció vázlatos rajza

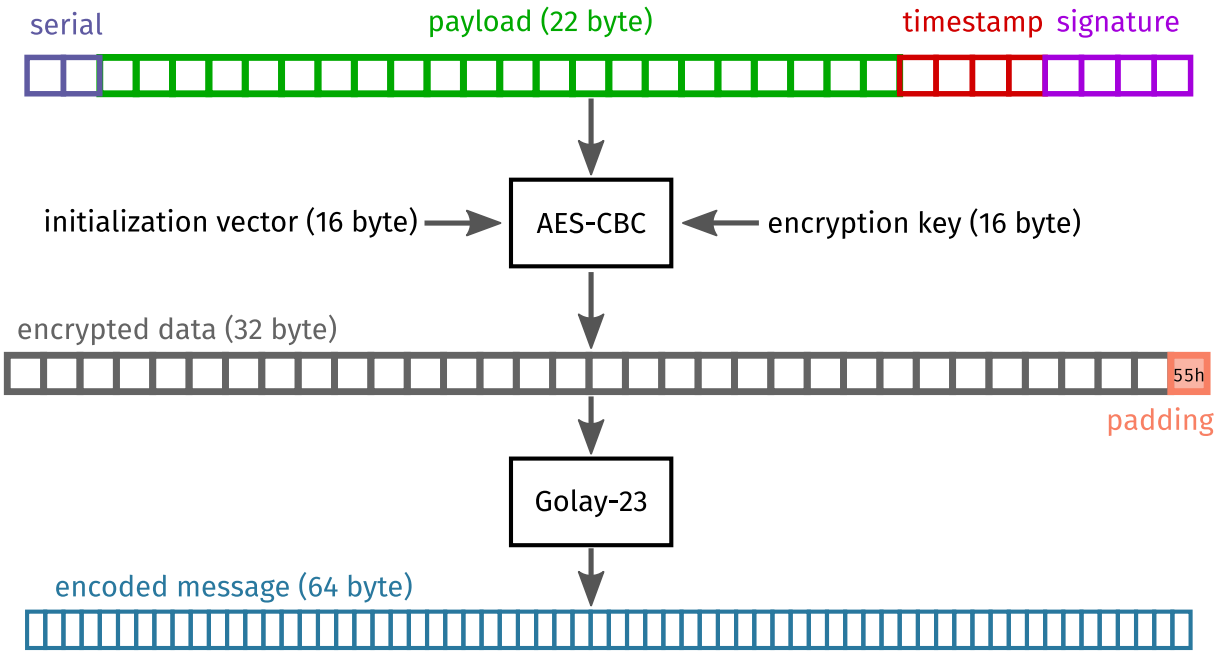
Az OBC egyik legfontosabb feladata, hogy tartsa a kapcsolatot a földi állomással (amelyet a csapatban gyakran csak „GND”-ként rövidítünk). Ez a gyakorlatban azt jelenti, hogy az OBC szoftverének együtt kell tudnia működni a földi állomás vezérlő szoftverével („GND app”), amelyet Kálmán Tibor fejleszt. [5]

A lérehozott adatkapcsolatot a 4.2 ábra szemlélteti vázlatosan. Természetesen kétirányú kapcsolatot valósítunk meg. A műholdtól a földi állomás felé küldött üzenet irányt *downlink*-nek (lefelé irányuló), a földi állomástól a műhold felé pedig *uplink*-nek (felfelé irányuló) nevezzük.

Üreszközökkel való kommunikáció folyamán elengedhetetlen a hibajavító kódolás alkalmazása. Ezen kódolások teszik lehetővé, hogy a korlátozott energia ellenére a SMOG-1 üzenetei eljussanak hozzánk. A hibajavító kódolás az algoritmuselmélet egyik részterülete, és arra alkalmas, hogy zajos csatornán keresztül növelje az átviteli rendszer jel-zaj viszony tűrőképességét, s ezáltal annak valószínűségét, hogy a fogadó oldalra az az üzenet jut el, amit a küldő feladott.

[31]

4.3.1. Uplink irány



4.3. ábra. Uplink kódoló működése

A földi állomásról induló, a műhold felé közlekedő üzenetek a 4.3 ábrán látható séma szerint épülnek fel. A GND appban található ennek a kódoló oldala, az OBC szoftverben pedig a dekódoló (mind a kettőt én dolgoztam ki).

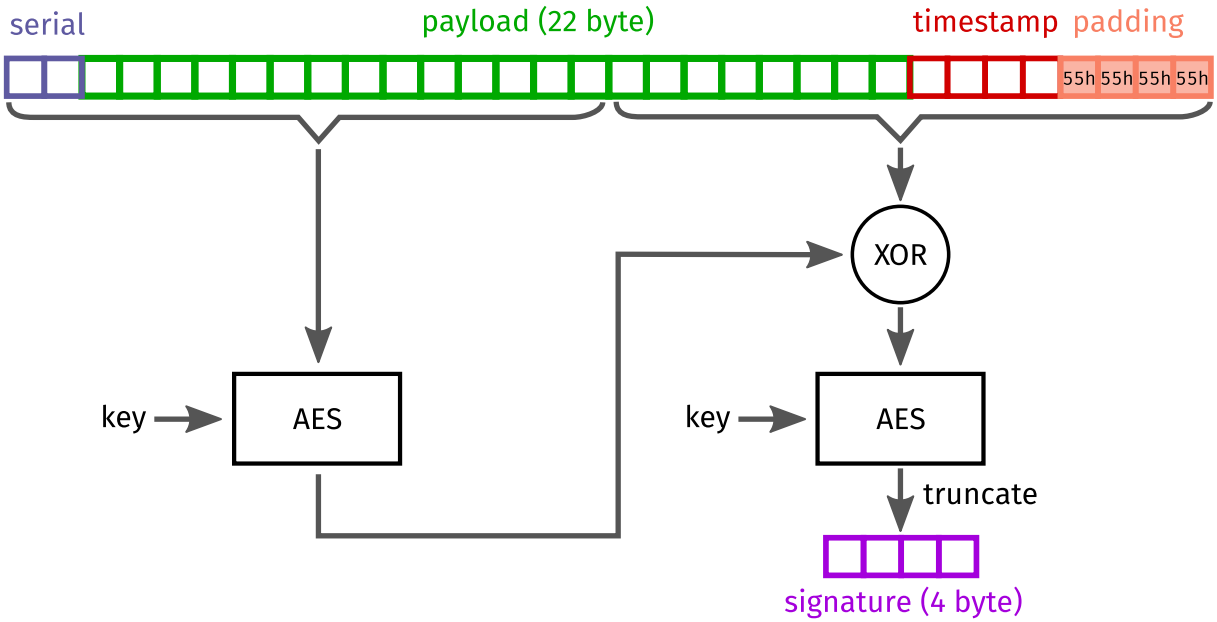
A műhold felé tartó üzeneteket Golay hibajavító kóddal [29] védjük meg, mert a tapasztalatok szerint kis adatmennyiségű üzenetek esetén ez válik be a legjobban, mert ennek a hibaarány-görbéje (BER görbéje) a legjobb. Szükség van a felküldött adatok titkosítására is, hogy ne tudjon akárki parancsokat küldeni a műholdnak. Az üzenetek integritását pedig egy az üzenet tartalmából, az aktuális időbélyegből és egy üzenetsorozatszámból generált aláírás segítségével védjük. Az OBC számontartja az utoljára sikeresen vett üzenet sorszámát.

Így néz ki tehát az uplink kódolási folyamat:

1. A felküldendő adat (payload) elé helyezünk egy sorozatszámot (2 byte), amely az üzenetet azonosítja, és az aktuális időbélyeget (4 byte). A végére 4 byte-nyi 55h byte-okból álló paddinget helyezünk el (azért, mert 55h = 01010101b).

2. Ebből kiszámoljuk az üzenethez tartozó aláírást a 4.4 ábrán látható módon, és az üzenet utolsó 4 byte-jába másoljuk.

3. A keletkezett 32 byte-os tömböt titkosítjuk AES-CBC eljárással egy titkosítási kulcs és inicializációs vektor felhasználásával. (Az aláírás számolásához használt kulcs természetesen



4.4. ábra. Uplink aláírásgenerálás menete

nem ugyanaz, mint a titkosító kulcs.)

4. A titkosított tömböt egy 55h padding byte-tal egészítjük ki (hogy a Golay-23 kódoló számára megfelelő input legyen).

5. Golay-23 kódoló segítségével (amely 12 bites inputból 23 bites outputot állít elő) kódoljuk a 33 byte-nyi adatot (264 bit, azaz 22 kódblokknyi). A kódoló kimenete 506 bit, amelynek a végére tesszük a 15h sorozatot, hogy így 64 byte-os kimenetet kapjunk.

Az OBC oldalon dekódolás pedig ennek inverze:

1. Golay-23 dekódolóval dekódoljuk a vett 64 byte első 506 bitjét, így kapunk egy 33 byte-os tömböt.

2. Eldobjuk a padding byte-ot és visszafejtjük az AES-CBC titkosítást.

3. Az üzenet első 28 byte-jából kiszámoljuk az aláírást a 4.4 ábrának megfelelően.

4. Összehasonlítjuk a vett (dekódolot) aláírást a most kiszámolt aláírással. Ha nem egyezik, akkor az üzenetet sérültnek tekintjük és eldobjuk.

5. Az üzenetsorszám és időbélyeg alapján következtetünk arra, hogy az üzenetet már feldolgoztuk-e (így kivédve a „visszajátszásos” támadást).

6. Az RTCC-k óráját a vett üzenet időbélyegéhez állítjuk.

Az uplink kódoló séma a következő problémáktól véd:

- A Golay kódolás képes blokkonként tetszőleges 3 bithibát javítani.
- Az aláírás alkalmazásával meggyőződünk a vett üzenet integritásáról (vagyis, hogy történte a Golay által nem javítható hiba).
- AES titkosítással gondoskodunk róla, hogy illetéktelenek ne tudjanak érvényes parancsot előállítani.
- Az időbélyeg és üzenetsorszám gondoskodik arról, hogy két különböző időpontban kiadott üzenet (akkor is, ha a payload azonos) mindig különböző legyen. Egyúttal véd a visszajátszásos támadás ellen is. (Ha valaki felveszi a GND közelében az általunk kiadott parancsot és azt megpróbálja később a műholdnak elküldeni.)

Golay kódolás és dekódolás

A Golay hibajavító kódolást Marcel Golay alkotta meg 1949-ben. [29] Legismertebb alkalmazása a NASA Voyager űszondáin történt, ahol a Golay(24, 12, 8) kódot használták. Ma már természetesen ismerünk ennél lényegesen jobb hibajavító képességgel rendelkező kódokat is, azonban a Golay kód továbbra is jó választás olyan alkalmazásoknál, ahol az egyszerre átküldendő csomagok mérete kicsi és/vagy a rendelkezésre álló hardveres erőforrások szerények és emiatt nincs lehetőség bonyolultabb kódokat megvalósítani.

A SMOG-1 uplinkhez a szisztematikus [31] Golay(24, 12, 8) kódot [29] használom. Ebből egy bit elhagyásával képezek egy Golay(23, 12, 7) kódot. [15] Munkám során igyekeztem olyan megoldást kitalálni, ami kellően hatékonyan fut beágyazott rendszeren is. Szerencsére a szisztematikus kódolót könnyen meg lehet valósítani bitműveletek és paritásszámolás segítségével.

A dekódoláshoz többféle módszer ismert. A leggyakrabban szindrómaalapú dekódolást alkalmaznak, amely problémásan megvalósítható, mert vagy lassú (mert sok iterációt igényel a hibajavítás számolásához), vagy memóriaigényes (mert tárolja a szindrómákhoz tartozó javítandó hibákat). Saját megoldásom egy olyan dekódoló algoritmus, amelyhez elegendő a Golay-féle generátormátrix tárolása a memóriában, és ennek használatával bitműveletekkel és paritásszámítással ciklusok nélkül képes egy-egy blokk dekódolására.

4.3.2. Downlink irány

A lefelé irányuló rádiós összeköttetésen a rádióamatőr műholdakban már bevált AO-40 kódot [30] használjuk. Ez a kódoló Szabó Lóránt munkája, ezért itt nem fejtem ki részletesen.

Kétféle variációját alkalmazzuk:

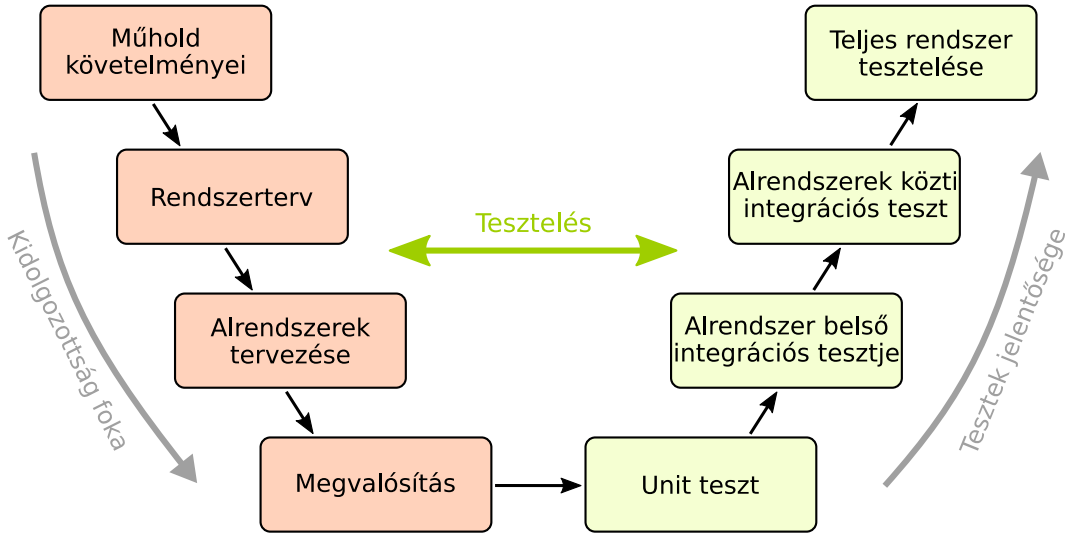
- Rövid: 128 byte-ból állít elő 333 kódolt byte-ot, ezt a műhold normál, periodikus telemetria adásai folyamán használjuk.
- Normál: 256 byte-ból állít elő 650 kódolt byte-ot, ezt pedig a műholdról földi állomás parancsra letöltött adatcsomagok, és a spektrumadatok esetén használjuk.

Ötödik fejezet

Tesztelés, minősítő mérések

A műholdat tesztelni kell, hogy meggyőződjünk a helyes működéséről. Sokan alábecsülik a tesztelés jelentőségét, pedig manapság bármely műszaki projekt szignifikáns részét teszik ki. 2015-ben egy átlagos projekt *teljes költségének 35 %-át* költötték tesztelésre (és ennek növekedését jósolják az elkövetkezendő évekre), azonban ez az arány még nagyobb kritikus ágazatokban, mint például az ipari automatizálás vagy healthcare (gyógyászati eszközök). [41]

Fontos tehát, hogy minden projektnek legyen tesztelési stratégiája. A miénket az 5.1 ábrán szemléltetem. (Ez a stratégia a klasszikus V modell egy változata.)



5.1. ábra. Teszteléshez használt módszertan

Az 5.1 ábrán látszik, hogy a fejlesztés minden szintjéhez tartozik egy tesztelési mód is. Vagyis nem elég pusztán csak az egyes komponensek működését tesztelni, hanem ezek együttműködéséről meg kell győződni az alrendszeren belül. Ha minden rendben működik, akkor az egyes alrendszereket más alrendszerekkel együtt tesztelve meggyőződhetünk a köztük levő kommunikáció, vezérlés, tápellátás, stb. funkcionalitásról. Ezután pedig a teljes műholdat összerakjuk és az összes alrendszer működését egyben teszteljük.

Minél magasabb tesztelési szinten vagyunk az 5.1 ábrán, annál nagyobb jelentőségű a teszt,

mert annál többféle problémát kimutat. Ha azonban a teszt eredménye az, hogy az adott dolog nem működik, akkor egy magas szintű tesztet igen nehéz elemezni, hogy a hiba pontos helyét megtaláljuk. Ezért van szükség az alacsonyabb szintű tesztekre is.

5.1. Hardver tesztelése

5.1.1. Alapműködés

Az OBC hardver működéséről az alábbi módon győződtem meg:

1. Az OBC panelen a rendszerbusz csatlakozót bekötöttem egy labortápra, valamint egy Silicon Laboratories EFM32 fejlesztői készletre a az 5.3 ábrán látható módon, amivel a debugolást végzem. A szabályozott buszfeszültségre és az OBC1 (első redundáns példány) tápfeszültségére adtam 3.3 V-ot. A labortáp áramkorlátjának beállítottam 120 mA-t.

2. Meggyőződtem arról, hogy a panel fogyasztása nem „túl nagy”, mert az táp-föld zárlatot jelentene. A fogyasztás kb. 7 mA (kikapcsolt perifériákkal), ami elfogadható.

3. Egy multiméter segítségével ellenőriztem, hogy a mikrokontroller minden táplában és az RTCC táplában 3.3 V-os feszültség van.

4. Meggyőződtem arról, hogy az EFM32 fejlesztői készlet felismeri és debug módban tud csatlakozni az OBC-ben levő EFM32WG mikrokontrollerhez.

5. Feltöltöttem a kontrollerre egy egyszerű teszt programot, amely konfigurál minden perifériát és megkísérel kommunikálni velük SPI buszon.

6. Megnézem, hogy milyen áramot mérnek az OBC panelen levő COM áramkorlátozó kapcsolók. Ha nem folyik áram, akkor néhány mA pozitív ofszet áramot kell mérnie.

7. Ezután elvégeztem ugyanezt a lépéssorozatot az OBC2-re (második redundáns példány) is.

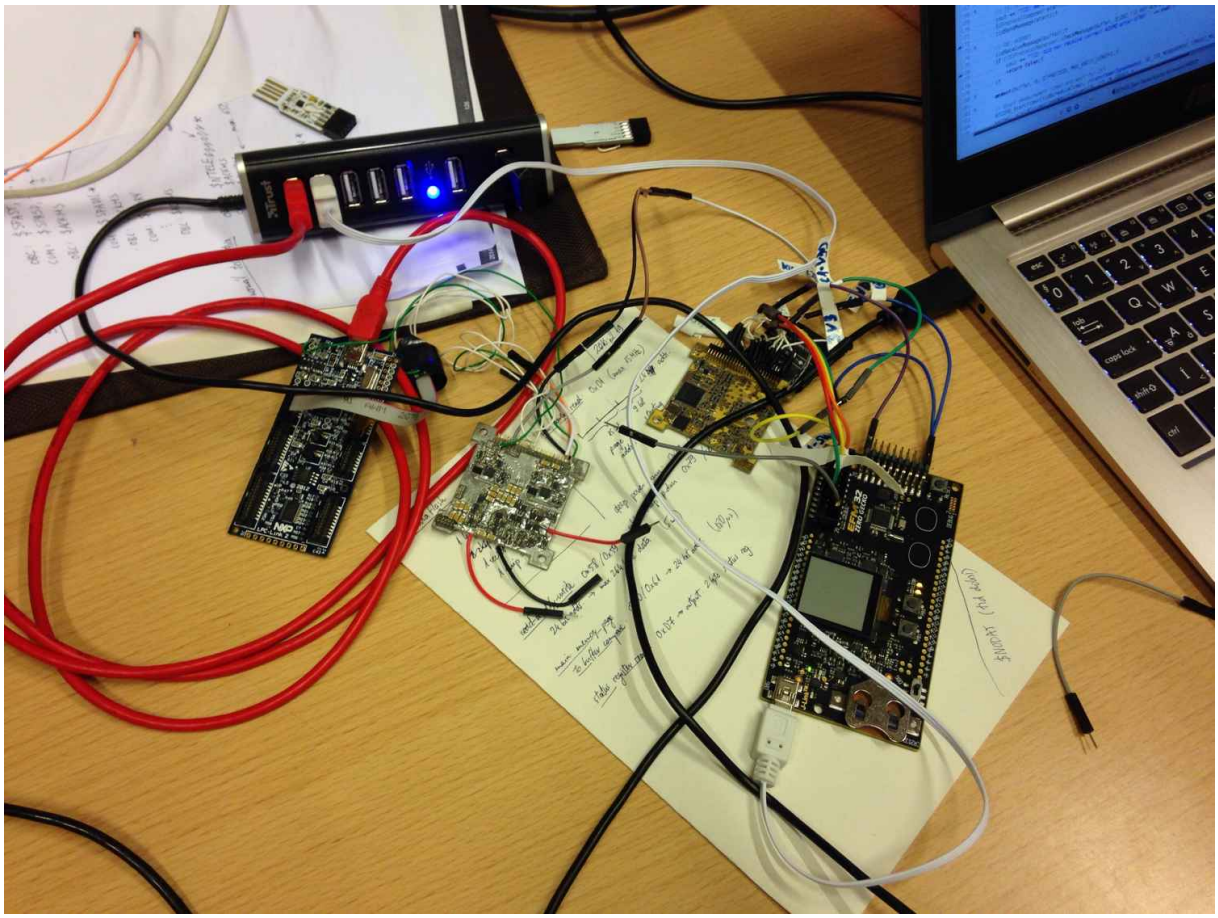
5.1.2. Összemérés más alrendszerekkel

Miután az alapfunkciók helyes működéséről meggyőződtem, teszteltem a kapcsolatot más alrendszerekkel. Például az az 5.2 ábrán látható, ahogyan az OBC és a TID alrendszerek közötti kommunikációt és a visszaérkező mérési adatok helyességét teszteltem.

5.2. Szoftver tesztelése

5.2.1. Unit tesztek

Az alacsony szintű szoftverelemeket (driverek, hardver konfigurációja, stb.) egy egyszerű teszt program segítségével próbáltam ki, ami minden perifériát konfigurál, majd megkísérli fel-



5.2. ábra. OBC és TID összemérése

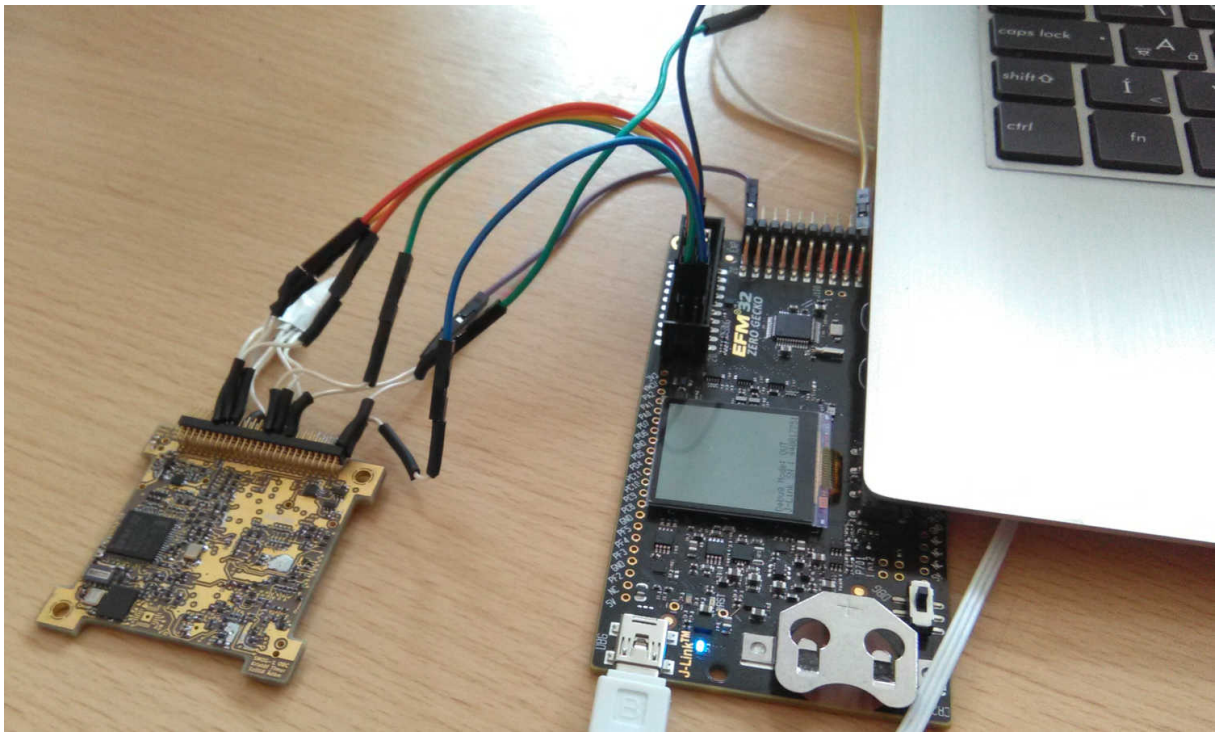
venni a kapcsolatot az SPI buszon a flash memóriával, RTCC-vel és MPU-val. Ezt szemlélteti az 5.3 ábra.

A szoftveres funkcionalitás magas szintű részeit (S1 protokoll megvalósítása, alrendszerek vezérése) olyan unit teszt programmal próbálom ki, ami nem a mikrokontrolleren, hanem számítógépen fut. Az alacsony szintű részeket lecserélem egy teszt implementációra [42]. Például a protokoll értelmező teszteléséhez nincs szükség igazi soros portra, bőven elég, ha a számítógép memóriájából a teszt által előre megadott értéket olvassa. Ezzel a módszerrel könnyebb tesztelni, mintha mindent a mikrokontrolleren futtatnék.

5.2.2. Tesztprogramok

S1 protokoll tesztelő

Mivel a rendszer egyik legkritikusabb pontja az S1 protokoll helyes megvalósítása az összes alrendszerben, ezért készítettem egy grafikus felülettel rendelkező programot, amivel valódi soros porton lehet az egyes SMOG-1 alrendszerek és a számítógép egy soros portja között kommunikálni. A soros port és UART közötti összeköttetést egy FTDI által gyártott FT234 fejlesztői készlettel valósítottuk meg.



5.3. ábra. OBC debugolása

Ez a program gondoskodik a start, sorszámozás és záró szekvencia generálásáról, a felhasználónak csak az üzenet tartalmát kell beírnia. Így lehet az alrendszerek közötti összeköttetést tesztelni akkor is, ha a másik alrendszer épp nem áll rendelkezésre.

Spektrumanalizátor segédprogram

A spektrumanalizátor adatok vizualizálásához készítettem egy segédprogramot. [43] Az ebben használt megjelenítést Kálmán Tibor továbbfejlesztette és beépítette a földi állomás szoftverbe. [5]

Uplink teszt kódoló

Az uplink kódolás tesztelésének megkönnyítésére készítettem egy egyszerű parancssori alkalmazást. Ennek parancsorban megadhatók a használandó kulcsok és a kódolandó állomány, és a kódolt adatokat egy külön állományba menti. Lehetővé teszi csak a Golay-kód tesztelését is. Ez a program tette lehetővé, hogy a különböző kódolások összehasonlításakor a Golay kód BER (bit error rate, bithibaarány) görbéjét is felvegyük.

5.2.3. Integrációs tesztek

Egyes alrendszerek bonyolultabb teszteseteit a Pápay Levente által fejlesztett GSE (ground support equipment) valósítja meg [14], így ezzel jelen dolgozatban nem foglalkozom.

5.2.4. Földi állomás és OBC kapcsolatának tesztje

Dudás Levente készített egy működő földi állomás modellt egy Raspberry Pi kártyaszámítógépből. Ezt Kálmán Tiborral összekötöttük a földi állomás szoftverrel. Ezután összeraktuk az OBC és COM alrendszereket, az OBC-t pedig rákötöttük az EFM32 fejlesztői készletre. Így le tudtuk tesztelni az összes fajta üzenetváltást, ami az OBC és a földi állomás szoftver között előfordul.

Hatodik fejezet

Összefoglaló

A fedélzeti számítógép legfőbb célja tehát az, hogy a műhold egészét vezérelje és nagy megbízhatósággal álljon rendelkezésre. „Bombabiztosnak” kell lennie, mert a műholdon nem engedhető meg az, mint napjaink fogyasztói termékeiben, ahol elterjedt a „ha nem működik, indítsuk újra” szemlélet. (Nem tudunk kimenni és újraindítani, ha nem működik.) Ezt a megbízhatóságot hardveres és áramköri redundanciával, valamint a szoftverelemek körültekintő megtervezésével és implementálásával; valamint a teljes rendszer alapos tesztelésével érem el.

Mivel az OBC minden más alrendszerrel kapcsolatban van, elengedhetetlen számomra, hogy a többi alrendszer működésével is tisztában legyek. Ezért vállaltam részt a műhold rendszertervének kidolgozásában is, és ezért fektettem hangsúlyt a rendszertervre is a dolgozatom elején.

6.1. Kitekintés

Projektünk hosszabb távú célja, hogy a műholdfejlesztés oktatási vonalát külföldön is népszerűsítsük, ezért a SMOG-1 küldetés sikerét követően azt tervezi a csapat, hogy a projekt teljes forrását (beleértve a kapcsolási rajzokat és nyomtatott huzalozási terveket, valamint a szoftverek forráskódját is) közzé fogjuk tenni nyílt forrású licensszel.

Emiatt arra törekszünk, hogy az általunk készült anyagok a nagyközönség számára is érthetőek legyenek. Tehát a szemléltető ábrákon, kapcsolási rajzokon és a szoftverben a dokumentáció és a kommentek nyelve *angol*, valamint olyan szoftverek segítségével készítettem munkámat, amelyek nyílt forrásúak és bárki számára hozzáférhetőek.

6.2. Köszönetnyilvánítás

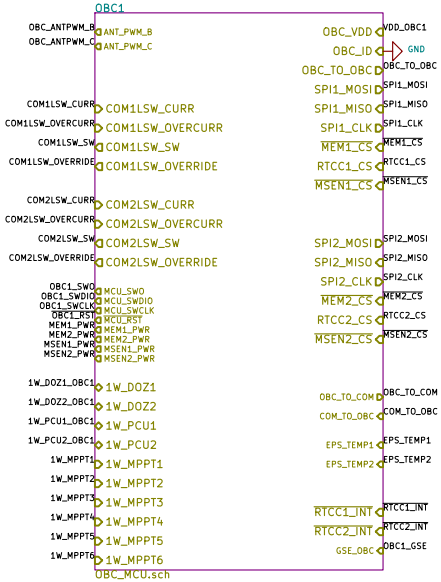
Szeretném megköszönni elsők között konzulensemnek, Dudás Leventének áldozatos munkáját, amellyel felbecsülhetetlen mértékben hozzájárult szakmai előmenetelemhez. Neki köszönhetem, hogy elmélyültem a hardvertervezés világában, és tőle tanultam a villamosmérnöki szakma gyakorlati oldalát is.

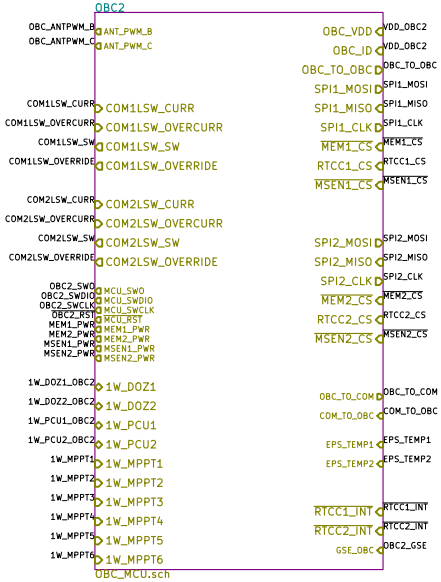
Hálával tartozom továbbá Herman Tibornak és Géczy Gábornak, akik tehetségükkel és szakértelmükkel segítettek a fedélzeti számítógép hardverének fejesztésében.

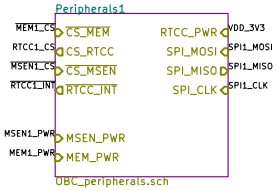
Köszönöm Pápay Leventének, Légrádi Mátének, Merényi Mártonnak és Szüllő Ádámnak, hogy a különféle OBC példányok (mérnöki, kvalifikációs, és repülő) alkatrészeinek beforrasztásában segítettek. Külön köszönet Szüllő Ádámnak a nyomtatott huzalozás elkészítésében nyújtott segítségéért.

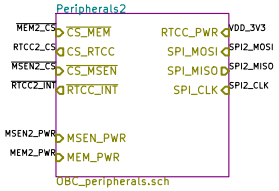
Köszönöm Kálmán Tibornak, hogy gördülékenyen csinálhattuk az OBC és a földi állomás szoftverének összehangolását, összemérését, és a sok tesztelést.

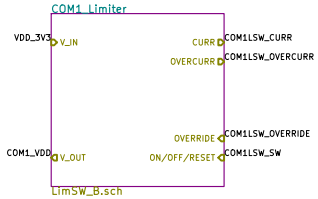
Függelék

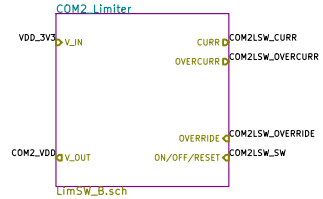

OBC1
OBC_ANTPWM_B
OBC_ANTPWM_C
ANT_PWM_B
ANT_PWM_C
COM1LSW_CURR
COM1LSW_OVERCURR
COM1LSW_SW
COM1LSW_OVERRIDE
COM2LSW_CURR
COM2LSW_OVERCURR
COM2LSW_SW
COM2LSW_OVERRIDE
OBC1_SWO
OBC1_SWDIO
OBC1_SWCLK
OBC1_RST
MEM1_PWR
MEM2_PWR
MSEN1_PWR
MSEN2_PWR
MCU_SWO
MCU_SWDIO
MCU_SWCLK
MCU_RST
MEM1_PWR
MEM2_PWR
MSEN1_PWR
MSEN2_PWR
1W_DOZ1_OBC1
1W_DOZ2_OBC1
1W_PCU1_OBC1
1W_PCU2_OBC1
1W_MPPT1
1W_MPPT2
1W_MPPT3
1W_MPPT4
1W_MPPT5
1W_MPPT6
1W_DOZ1
1W_DOZ2
1W_PCU1
1W_PCU2
1W_MPPT1
1W_MPPT2
1W_MPPT3
1W_MPPT4
1W_MPPT5
1W_MPPT6
OBC_VDD
OBC_ID
OBC_TO_OBC
SPI1_MOSI
SPI1_MISO
SPI1_CLK
MEM1_CS
RTCC1_CS
MSEN1_CS
SPI2_MOSI
SPI2_MISO
SPI2_CLK
MEM2_CS
RTCC2_CS
MSEN2_CS
OBC_TO_COM
COM_TO_OBC
EPS_TEMP1
EPS_TEMP2
RTCC1_INT
RTCC2_INT
GSE_OBC
VDD_OBC1
GND
OBC_MCU.sch

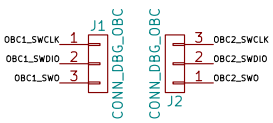

OBC2
OBC_ANTPWM_B
OBC_ANTPWM_C
ANT_PWM_B
ANT_PWM_C
COM1LSW_CURR
COM1LSW_OVERCURR
COM1LSW_SW
COM1LSW_OVERRIDE
COM2LSW_CURR
COM2LSW_OVERCURR
COM2LSW_SW
COM2LSW_OVERRIDE
OBC2_SWO
OBC2_SWDIO
OBC2_SWCLK
OBC2_RST
MEM1_PWR
MEM2_PWR
MSEN1_PWR
MSEN2_PWR
MCU_SWO
MCU_SWDIO
MCU_SWCLK
MCU_RST
MEM1_PWR
MEM2_PWR
MSEN1_PWR
MSEN2_PWR
1W_DOZ1_OBC2
1W_DOZ2_OBC2
1W_PCU1_OBC2
1W_PCU2_OBC2
1W_MPPT1
1W_MPPT2
1W_MPPT3
1W_MPPT4
1W_MPPT5
1W_MPPT6
1W_DOZ1
1W_DOZ2
1W_PCU1
1W_PCU2
1W_MPPT1
1W_MPPT2
1W_MPPT3
1W_MPPT4
1W_MPPT5
1W_MPPT6
OBC_VDD
OBC_ID
OBC_TO_OBC
SPI1_MOSI
SPI1_MISO
SPI1_CLK
MEM1_CS
RTCC1_CS
MSEN1_CS
SPI2_MOSI
SPI2_MISO
SPI2_CLK
MEM2_CS
RTCC2_CS
MSEN2_CS
OBC_TO_COM
COM_TO_OBC
EPS_TEMP1
EPS_TEMP2
RTCC1_INT
RTCC2_INT
GSE_OBC
VDD_OBC2
VDD_OBC2
OBC_TO_OBC
SPI1_MOSI
SPI1_MISO
SPI1_CLK
MEM1_CS
RTCC1_CS
MSEN1_CS
SPI2_MOSI
SPI2_MISO
SPI2_CLK
MEM2_CS
RTCC2_CS
MSEN2_CS
OBC_TO_COM
COM_TO_OBC
EPS_TEMP1
EPS_TEMP2
RTCC1_INT
RTCC2_INT
OBC2_GSE
OBC_MCU.sch

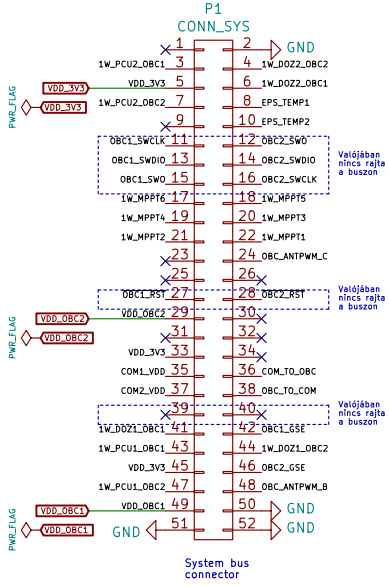

Peripherals1
MEM1_CS
RTCC1_CS
MSEN1_CS
RTCC1_INT
MSEN1_PWR
MEM1_PWR
CS_MEM
CS_RTCC
CS_MSEN
RTCC_INT
MSEN_PWR
MEM_PWR
RTCC_PWR
SPI_MOSI
SPI_MISO
SPI_CLK
VDD_3V3
SPI1_MOSI
SPI1_MISO
SPI1_CLK
OBC_peripherals.sch


Peripherals2
MEM2_CS
RTCC2_CS
MSEN2_CS
RTCC2_INT
MSEN2_PWR
MEM2_PWR
CS_MEM
CS_RTCC
CS_MSEN
RTCC_INT
MSEN_PWR
MEM_PWR
RTCC_PWR
SPI_MOSI
SPI_MISO
SPI_CLK
VDD_3V3
SPI2_MOSI
SPI2_MISO
SPI2_CLK
OBC_peripherals.sch

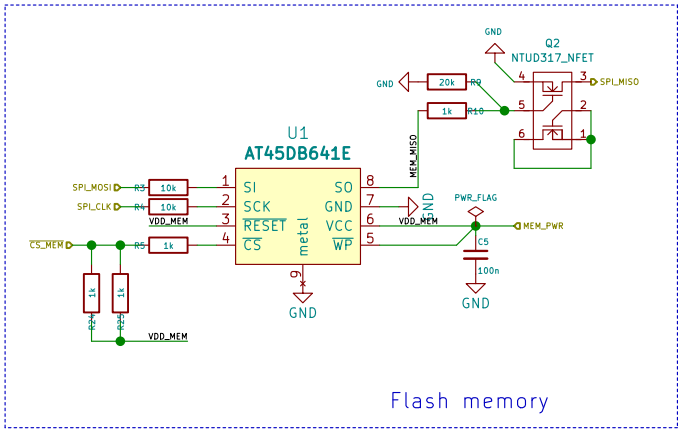

COM1_Limiter
VDD_3V3
COM1_VDD
V_IN
V_OUT
CURR
OVERCURR
OVERRIDE
ON/OFF/RESET
COM1LSW_CURR
COM1LSW_OVERCURR
COM1LSW_OVERRIDE
COM1LSW_SW
LimSW_B.sch

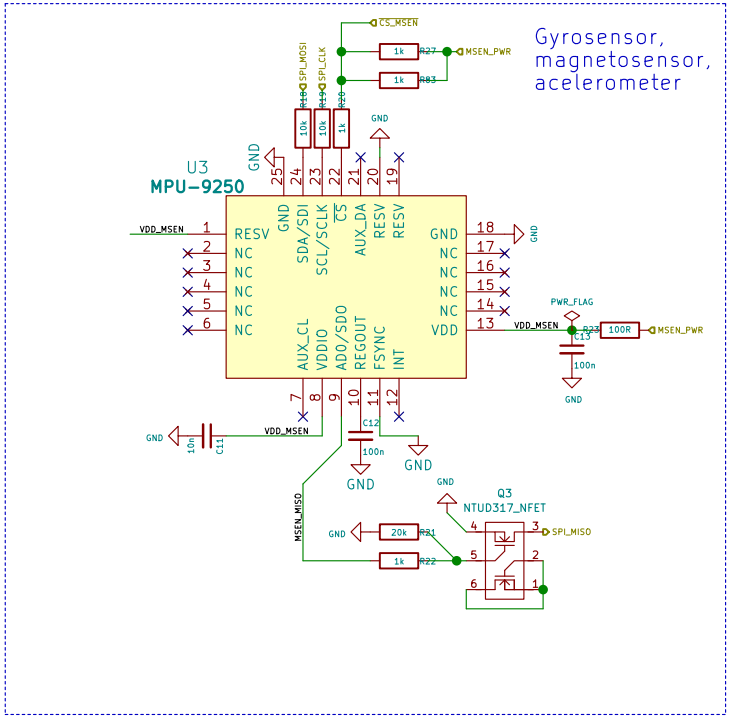

COM2_Limiter
VDD_3V3
COM2_VDD
V_IN
V_OUT
CURR
OVERCURR
OVERRIDE
ON/OFF/RESET
COM2LSW_CURR
COM2LSW_OVERCURR
COM2LSW_OVERRIDE
COM2LSW_SW
LimSW_B.sch

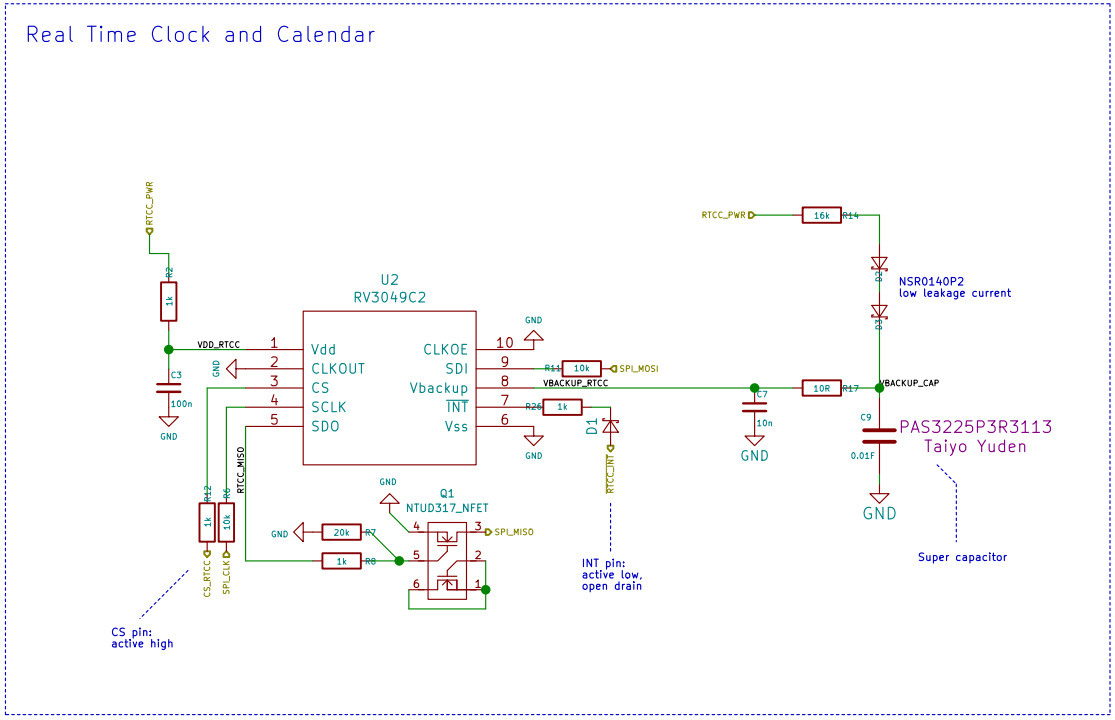

J1
CONN_DBG_OBC
OBC1_SWCLK
OBC1_SWDIO
OBC1_SWO
J2
CONN_DBG_OBC
OBC2_SWCLK
OBC2_SWDIO
OBC2_SWO


P1
CONN_SYS
PWR_FLAG
VDD_3V3
VDD_OBC2
VDD_OBC1
GND
1W_PCU2_OBC1
1W_DOZ2_OBC2
1W_DOZ2_OBC1
EPS_TEMP1
EPS_TEMP2
OBC1_SWCLK
OBC2_SWO
OBC1_SWDIO
OBC2_SWCLK
OBC1_SWO
1W_MPPT6
1W_MPPT5
1W_MPPT4
1W_MPPT3
1W_MPPT2
1W_MPPT1
OBC_ANTPWM_C
OBC1_RST
OBC2_RST
Valójában nincs rajta a buszon
COM1_VDD
COM2_VDD
COM_TO_OBC
OBC_TO_COM
1W_DOZ1_OBC1
OBC1_GSE
1W_PCU1_OBC1
1W_DOZ1_OBC2
OBC2_GSE
1W_PCU1_OBC2
OBC_ANTPWM_B
System bus connector

Prevent the MCU from startup when a resistor fails on some pins).
Digital connection between the two OBCs. In case one is accidentally powered up by the EPS when it shouldn't be.
This pin identifies which OBC is the current one.
Output of CDH limiter analogue current sensors
Output of EPS analogue temperature sensors
No pull-down resistors needed because they are in the limiters.
Power filtering


U1
AT45DB641E
SI
SCK
RESET
CS
SO
GND
VCC
WP
metal
SPI_MOSI
SPI_CLK
CS_MEM
VDD_MEM
MEM_MISO
MEM_PWR
PWR_FLAG
Q2
NTUD317_NFET
SPI_MISO
Flash memory


Gyrosensor,
magnetosensor,
acelerometer
U3
MPU-9250
RESV
NC
GND
SDA/SDI
SCL/SCLK
CS
AUX_DA
AUX_CL
VDDIO
ADO/SDO
REGOUT
FSYNC
INT
VDD
VDD_MSEN
MSEN_PWR
MSEN_MISO
PWR_FLAG
Q3
NTUD317_NFET
SPI_MISO


Real Time Clock and Calendar
U2
RV3049C2
Vdd
CLKOUT
CS
SCLK
SDO
CLKOE
SDI
Vbackup
INT
Vss
RTCC_PWR
VDD_RTCC
VBACKUP_RTCC
BACKUP_CAP
NSR0140P2
low leakage current
PAS3225P3R3113
Taiyo Yuden
Super capacitor
Q1
NTUD317_NFET
SPI_MISO
SPI_MOSI
RTCC_MISO
RTCC_INT
CS_RTCC
SPI_CLK
CS pin:
active high
INT pin:
active low,
open drain

Irodalomjegyzék

[1] Dudás Levente, *The Spectrum Monitoring System of Smog-1 Satellite*, MAREW 2015 konferencia
http://gnd.bme.hu/smog1/files/publikaciok/levi_com/PID3627859.pdf
(elérés dátuma: 2017. december 8.)

[2] Dudás Levente, Pápay Levente, Gschwindt András, Seller Rudolf, *A MASAT-1 automatizált és távvezérelt földi vezérlő állomásai*, Repüléstudományi Konferencia 2014, Szolnok, Repüléstudományi Közlemények XXVI. évfolyam, 2014. 2. szám, pp 426-442
http://epa.oszk.hu/02600/02694/00065/pdf/EPA02694_rtk_2014_2_426-442.pdf
(elérés dátuma: 2017. december 8.)

[3] Géczy Gábor, *A SMOG-1 PocketQube másodlagos energiaellátó rendszere*, TDK dolgozat, 2016
http://gnd.bme.hu/smog1/files/publikaciok/gabor_eps2/A-Smog1-PocketQube-masodlagos-energiaellato-Dolgozat-5.pdf
(elérés dátuma: 2017. december 8.)

[4] Herman Tibor, *SMOG-1 Elsődleges Energiaállító Rendszere*, MSc diplomaterv, 2015
http://gnd.bme.hu/smog1/files/publikaciok/tibi_eps1/A-Smog1-PocketQube-elsodleges-energiaellato-Dolgozat-4.pdf
(elérés dátuma: 2017. december 8.)

[5] Kálmán Tibor, *SMOG-1 műhold földi állomás kliens szoftver fejlesztése*, BSc szakdolgozat, 2016
https://diplomaterv.vik.bme.hu/hu/Theses/SMOG1-muhold-foldi-allomas-kliens-szoftver/Attachment/32504
(elérés dátuma: 2017. december 8.)

[6] Kristóf Timur, *PocketQube műhold fedélzeti számítógépének tervezése*, BSc Önálló laboratórium beszámoló, 2015
http://gnd.bme.hu/smog1/files/publikaciok/timur_obc/KristofTimur-onlab-201505084.pdf
(elérés dátuma: 2017. december 8.)

[7] Jáger Dávid és Török Péter, *PocketQube műhold numerikus hőtani szimulációja*, TDK dolgozat, 2014
http://gnd.bme.hu/smog1/files/publikaciok/tdk_2014/PocketQube_muhold_numerikus_hotani_szimulacioja.pdf
(elérés dátuma: 2017. december 8.)

[8] Katona Krisztina, Ötvös Vivien, Sipos Anna Ilona, Tomasics Sára, *PocketQube műhold vázszerkezetének fejlesztése és hőtechnikai elemzése*, TDK dolgozat, 2014
http://gnd.bme.hu/smog1/files/publikaciok/tdk_2014/PocketQube_muhold_vazszerkezetenek_

fejlesztese_es_hotechnikai_elemzese.pdf
(elérés dátuma: 2017. december 8.)

[9] Olaszi Bálint, *A BME-1 „pocketqube" műhold ciklusonkénti napelemes energiatermelésének meghatározása*, TDK dolgozat, 2014
http://gnd.bme.hu/smog1/files/publikaciok/tdk_2014/PocketQube_ciklusonkenti_napelemes_energiatermeles.pdf
(elérés dátuma: 2017. december 8.)

[10] Welsz Ágnes, *A SMOG-1 műhold hőáramhálózatos modellezése*, TDK dolgozat, 2016
http://gnd.bme.hu/smog1/articles/20161117_smog1_tdkk/Welsz_Agnes_TDK.pdf
(elérés dátuma: 2017. december 8.)

[11] Petróczi Balázs, *Thermal finite element modelling of the SMOG-1 nanosatellite*, TDK dolgozat, 2016
http://gnd.bme.hu/smog1/articles/20161117_smog1_tdkk/Thermal_finite_element_modelling_of_the_SMOG1_nanosatellite.pdf
(elérés dátuma: 2017. december 8.)

[12] Pápay Levente, *SMOG-1 Antennanyitás*, 2015
http://gnd.bme.hu/smog1/files/publikaciok/levi_ant/SMOG_1_antennanyitas.docx
(elérés dátuma: 2017. december 8.)

[13] Légrádi Máté, *Körpolarizált primersugárzó tervezése parabolatükörbe*, MSc önálló laboratórium beszámoló, 2015
http://gnd.bme.hu/smog1/files/publikaciok/gnd/Legradi_Mate/bmegnd_parabola45_legradi_mate_onlab2.pdf
(elérés dátuma: 2017. december 8.)

[14] Pápay Levente, *GSE berendezés fejlesztése a SMOG-1 hallgatói műhold kvalifikációs tesztjeinek támogatására*
http://gnd.bme.hu/smog1/files/publikaciok/levi_gse/GSE-berendezes-fejlesztese-a-Smog1-hallgatoi-Dolgozat-2.pdf
(elérés dátuma: 2017. december 8.)

[15] Aleksziev Rita Antónia, *Golay-kódok*, BSc szakdolgozat, 2015
https://web.cs.elte.hu/blobs/diplomamunkak/bsc_alkmat/2015/aleksziev_rita_antonia.pdf
(elérés dátuma: 2017. december 8.)

[16] ARM Infocenter, *ARM C Language Extensions*
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0053c/IHI0053C_acle_2_0.pdf
(elérés dátuma: 2017. december 8.)

[17] GCC dokumentáció, *ARM C Language Extensions*
https://gcc.gnu.org/onlinedocs/gcc/ARM-C-Language-Extensions-_0028ACLE_0029.html
(elérés dátuma: 2017. december 8.)

[18] Stephen Ferg, *Event-Driven Programming: Introduction, Tutorial, History*
http://sourceforge.net/projects/eventdrivenpgm/
(elérés dátuma: 2017. december 8.)

[19] Scott Rosenthal, *Interrupts might seem basic, but many programmers still avoid them*
http://www.sltf.com/articles/pein/pein9505.htm
(elérés dátuma: 2017. december 8.)

[20] Red Hat, *Hardware Interrupts*
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.3/html/Realtime_Reference_Guide/chap-Realtime_Reference_Guide-Hardware_interrupts.html
(elérés dátuma: 2017. december 8.)

[21] Silabs, *AN0007 - Energy Modes*
https://www.silabs.com/Support%20Documents/TechnicalDocs/AN0007.pdf
(elérés dátuma: 2017. december 8.)

[22] Silabs, *AN0017 - Low Enery UART*
https://www.silabs.com/Support%20Documents/TechnicalDocs/AN0017.pdf
(elérés dátuma: 2017. december 8.)

[23] Silabs, *AN0013 - DMA*
https://www.silabs.com/Support%20Documents/TechnicalDocs/AN0013.pdf
(elérés dátuma: 2017. december 8.)

[24] Silabs, *EFM32WG Reference Manual*
http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32WG-RM.pdf
(elérés dátuma: 2017. december 8.)

[25] National Instruments, *DMA Fundamentals on Various PC Platforms*
http://cires1.colorado.edu/jimenez-group/QAMSResources/Docs/DMAFundamentals.pdf
(elérés dátuma: 2017. december 8.)

[26] Bhavin Turakhia, *Understanding CPU utilization and Optimization*
http://careers.directi.com/display/tu/Understanding+CPU+Utilization+and+Optimization
(elérés dátuma: 2017. december 8.)

[27] Chris Lattner, *What Every C Programmer Should Know About Undefined Behavior*
http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html
(elérés dátuma: 2017. december 8.)

[28] *GCC Options That Control Optimalization*
(elérés dátuma: 2015. október 26.)
https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
(elérés dátuma: 2017. december 8.)

[29] Golay, Marcel J. E. (1949). *Notes on Digital Coding*. Proc. IRE 37: 657.

[30] Phil Karn, *Proposed Coded AO-40 Telemetry Format*
http://www.ka9q.net/papers/ao40tlm.html
(elérés dátuma: 2017. december 8.)

[31] Györfi-Győri-Varga, *Információ- és kódelmélet*

[32] Adesto Technologies, *AT45DB641E adatlap*
https://www.adestotech.com/wp-content/uploads/DS-45DB641E-027.pdf
(elérés dátuma: 2017. december 8.)

[33] Micro Crystal Switzerland, *RV-3049-C3 adatlap*
http://www.microcrystal.com/images/_Product-Documentation/02_Oscillator_&_RTC_Modules/01_Datasheet/RV-3049-C3.pdf
(elérés dátuma: 2017. december 8.)

[34] Invensense, *MPU-9250 adatlap*
http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
(elérés dátuma: 2017. december 8.)

[35] NASA, *The Power of 10: Rules for Developing Safety-Critical Code*
http://web.eecs.umich.edu/~imarkov/10rules.pdf
(elérés dátuma: 2017. december 8.)

[36] NASA, *C++ Coding Standards and Style Guide*
https://ntrs.nasa.gov/search.jsp?R=20080039927
(elérés dátuma: 2017. december 8.)

[37] ESA BSSC, *C/C++ Coding Standard*
http://www.esa.int/TEC/Software_engineering_and_standardisation/TECT5CUXBQE_0.html
(elérés dátuma: 2017. december 8.)

[38] MISRA, *MISRA C++*
https://www.misra.org.uk/?TabId=171
(elérés dátuma: 2017. december 8.)

[39] Eurocircuits műszaki specifikációk
https://www.eurocircuits.hu/pcb-prototype-and-small-series-services-offered-by-eurocircuits-made-in-europe/
(elérés dátuma: 2017. december 8.)

[40] NASA, *Tin Whisker (and Other Metal Whisker) Homepage*
https://nepp.nasa.gov/Whisker/
(elérés dátuma: 2017. december 8.)

[41] ISTQB, *Worldwide Software Testing Practices Report*, 2015-2016
https://www.istqb.org/documents/ISTQB_Worldwide_Software_Testing_Practices_Report.pdf
(elérés dátuma: 2017. december 8.)

[42] Martin Fowler, *Mocks Aren't Stubs*
https://martinfowler.com/articles/mocksArentStubs.html
(elérés dátuma: 2017. december 8.)

[43] Kristóf Timur, *Frequency analyzer app*
https://github.com/Venemo/frequency-analyzer
(elérés dátuma: 2017. december 8.)