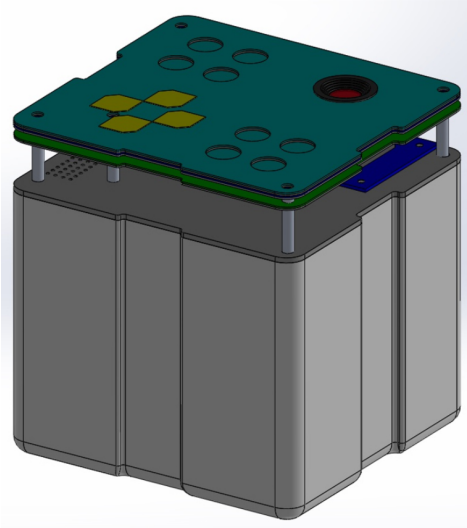
SADRŽAJ

1. Uvod

Projekt FERSAT, koji se od 2018. godine provodi na Fakultetu elektrotehnike i računarstva, uključuje izradu i lansiranje satelita CubeSat te korištenje satelita u svrhu prikupljanja informacija o svjetlosnom zagađenju i debljini ozonskog omotača, fotografiranja površine Zemlje i horizonta, te ispitivanja sustava za komunikaciju u radijskom X-pojasu. Satelit u izradi dimenzija je približno 10 cm x 10 cm x 10 cm, volumena jedne litre i ne teži od 4/3 kilograma, što ga svrstava u skupinu satelita formata CubeSat 1U [17]. Očekivani životni vijek satelita je 3 godine, a bit će postavljen u niskoj Zemljinoj orbiti na visini između 500 i 600 kilometara. Slika 1.1 prikazuje planirani izgled satelita.



Slika 1.1: Skica planiranog izgleda FERSAT-a. Prikazani su korisni teret (engl. *payload*) i platforma (engl. *bus*) [17]

Ovaj rad opisuje razvijenu programsku potporu koja ostvaruje prikupljanje i obradu podataka o svjetlosnom onečišćenju i debljini ozonskog omotača. Programska potpora izvodi se na *Payload Data Handler* (PDH) računalu koje upravlja radom korisnog tereta. U razvoju je korišteno postojeće sklopovlje za prikupljanje podataka, opisano u poglavlju 3.

Blok dijagram sustava FERSAT-a relevantnih za ovaj rad prikazan je slikom **??**.

Glavno računalo koje upravlja radom satelita je *Command and Data Handler* (CDH) računalo. Kada PDH računalo primi naredbu od CDH računala, prikuplja uzorke sa senzora, obavlja potrebnu obradu i sprema podatke u vanjsku *Flash* memoriju. Na zahtjev CDH računala, podaci se prosljeđuju putem radio veze do zemaljske postaje.

Nastavak rada strukturiran je na sljedeći način. U poglavlju 2 opisana je arhitektura korisnog tereta FERSAT-a. Poglavlje 3 opisuje komunikacijsko sučelje između mikrokontrolera STM32L471VGT6 i senzorskog podsustava (SPI - *Serial Peripheral Interface*). Sklopovlje korišteno u senzorskom podsustavu opisano je u poglavlju 4. U poglavlju 5 opisani su razvijeni upravljački programi za pojedine sklopovske komponente, cjelokupna programska potpora za senzorski podsustav, i integracija s ostalim dijelovima programske potpore PDH računala korištenjem operacijskog sustava za rad u stvarnom vremenu FreeRTOS.

2. Arhitektura korisnog tereta FERSAT-a

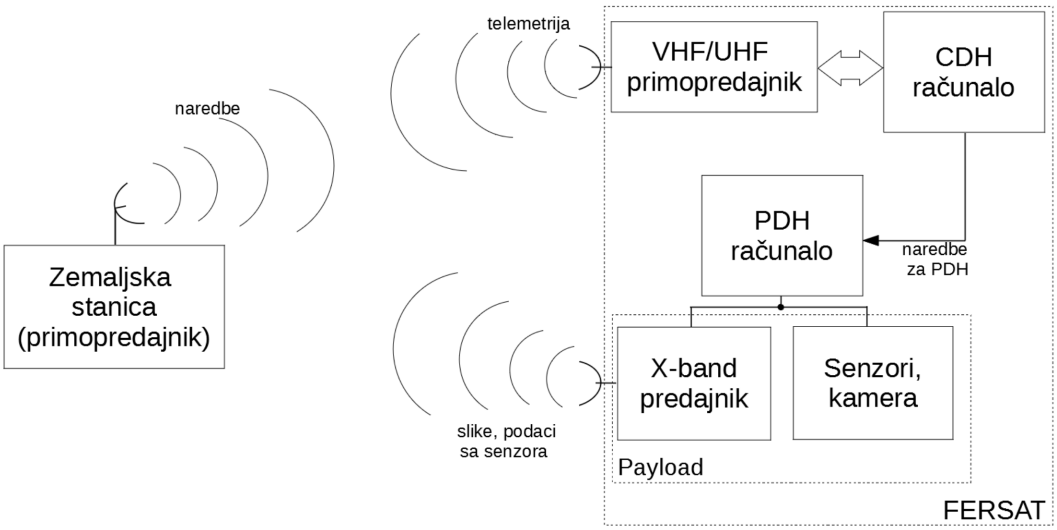
Planirani korisni teret (engl. *payload*) FERSAT-a podijeljen je na tri podsustava:

- kamera za snimanje površine Zemlje i zemaljskog horizonta,
- detektori svjetla u vidljivom i ultraljubičastom dijelu spektra za mjerenje svjetlosnog onečišćenja i debljine stupca ozona,
- komunikacijski sustav u radijskom X-pojasu (10.45 GHz) za prijenos podataka na Zemlju.

Radom korisnog tereta upravlja *Payload Data Handler* (PDH) računalo. Zadaća je PDH računala prikupiti podatke iz senzorskog podsustava i kamere, pohraniti ih u trajnu memoriju (engl. *non-volatile memory*) te poslati prikupljene podatke na Zemlju korištenjem komunikacijskog podsustava. Kao PDH računalo odabran je mikrokontroler STM32L471VGT6 proizvođača ST Microelectronics.

Za rad ostalih podsustava satelita koji nisu direktno vezani uz koristan teret (npr. upravljanje položajem satelita, slanje telemetrijskih podataka na Zemlju) brine se *Command and Data Handler* (CDH) računalo. CDH računalo također upravlja napajanjem korisnog tereta i šalje naredbe PDH računalu. Komunikacija CDH i PDH računala odvija se korištenjem sučelja CAN (*Controller Area Network*). Konkretno CDH računalo u trenutku pisanja ovog teksta još nije odabrano. Slika 2.1 prikazuje blok dijagram cijelog sustava.

Senzorski podsustav ima dvije temeljne zadaće. Prva od njih je korištenjem fotosenzora koji rade u vidljivom dijelu elektromagnetskog spektra prikupiti podatke na temelju kojih će biti moguće odrediti udio LED rasvjete u naseljenim mjestima u odnosu na konvencionalnu natrijevu, metal-halidnu i fluorescentnu javnu rasvjetu. U sklopu projekta FERSAT već je razvijen algoritam koji na temelju obrade signala multispektralnog svjetla sa Zemlje može odrediti ovu informaciju [15]. Mjerenje udjela LED rasvjete zanimljivo je zbog mogućih negativnih utjecaja plavog svjetla na ljudsko



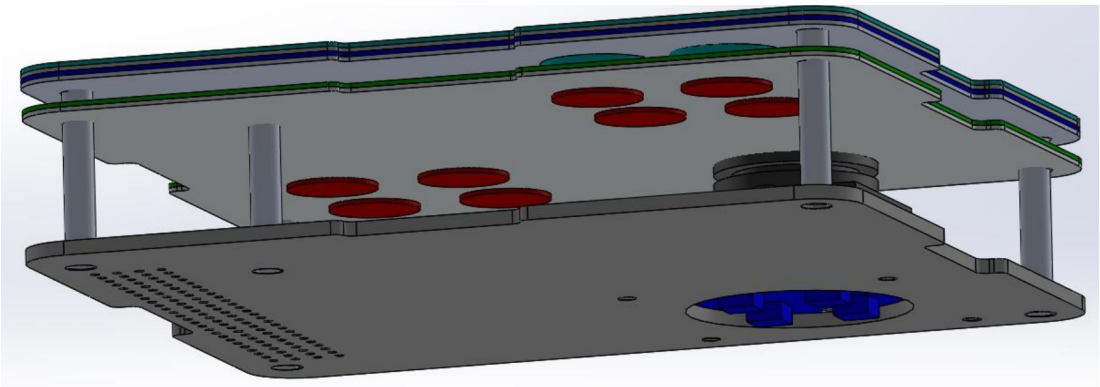
Slika 2.1: Blok dijagram FERSAT-a i komunikacija sa zemaljskom postajom [8]

zdravlje, koje LED rasvjeta emitira u znatno većem intenzitetu nego konvencionalna [4].

Druga je zadaća senzorskog podsustava mjerenje propusnosti i refleksije atmosfere za ultraljubičasto svjetlo u svrhu određivanja debljine ozonskog omotača. Za mjerenje se koriste PureB detektori ultraljubičastog zračenja razvijeni na FER-u [3] i algoritmi razvijeni za tu namjenu [12]. Uspješna mjerenja po prvi put bi potvrdila mogućnost korištenja ove tehnologije u mjerenjima debljine ozonskog omotača iz svemira.

Upravljačko sklopovlje potrebno za rad PDH računala već je razvijeno [6]. Tiskana pločica PDH računala, osim mikrokontrolera STM32L471VGT6, sadrži i vanjsku *Flash* memoriju, sustav za napajanje, sklop za kontrolu izvođenja programa (engl. *watchdog*), upravljački sklop za CAN komunikaciju i konektore za povezivanje s ostalim dijelovima sustava. PDH pločica bit će smještena ispod senzorske pločice, u takozvanoj *stack-up* konfiguraciji (slika 2.2).

Također, u sklopu projekta FERSAT razvijen je i dio programske potpore PDH računala [8]. No, kako je u međuvremenu došlo do promjene izbora mikrokontrolera PDH računala i promjene dijela sklopovlja senzorskog podsustava, dijelove te programske potpore bilo je potrebno prilagoditi ili ponovno razviti.



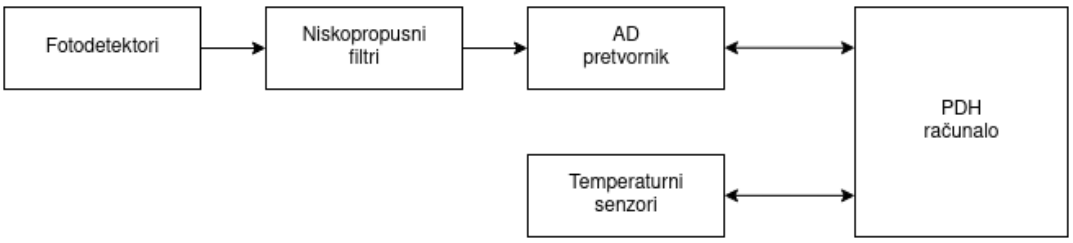
Slika 2.2: Trodimenzionalni model korisnog tereta FERSAT-a. PDH računalo smješteno je na donjoj pločici, a senzorski podsustav na srednjoj [6]

3. Opis korištenog sklopovlja

3.1. Senzorska pločica FERSAT-a

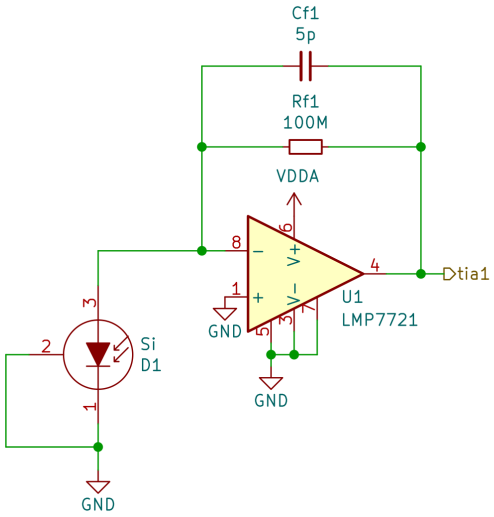
Slika 3.1 prikazuje blok dijagram senzorskog podsustava FERSAT-a. Sve komponente sustava smještene su na takozvanoj senzorskoj tiskanoj pločici. Komponente koje se na njoj nalaze su:

- 8 fotodetektora (6 detektora vidljive svjetlosti i 2 detektora ultraljubičastog zračenja) s pojačalima LMP7721,
- 8 RC filtara, po jedan za svaki detektor,
- 3 temperaturna senzora ADT7301,
- analogno-digitalni pretvornik ADS131M08.



Slika 3.1: Blok dijagram senzorskog podsustava

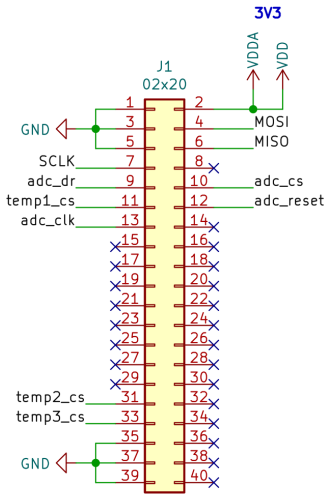
Jedan fotodetektor sastoji se od fotodiode i transimpedancijskog pojačala izvedenog pojačalom LMP7721. Slika 3.2 prikazuje jedan takav sklop. Transimpedancijsko pojačalo pretvara struju fotodiode u naponski signal i ujedno pojačava signal. S obzirom da pojačanje transimpedancijskog pojačala linearno ovisi o impedanciji u povratnoj vezi [13], korištena je velika vrijednost otpora i mala vrijednost kapaciteta kako bi se postigao velik iznos pojačanja. Na svaku od dioda postavljen je optički filtar (staklo ispred diode) koji propušta samo svjetlost u određenom području valnih duljina. Područje propuštanja razlikuje se za svaku fotodiodu.



Slika 3.2: Električna shema fotodetektora

Izlaz svakog fotodetektora propušta se kroz niskopropusni RC filtar, koji djeluje kao *anti-aliasing* filtar, a ujedno i kao teret pojačala. Signal se na kraju dovodi na ulaz analogno-digitalnog pretvornika ADS131M08. Na pločici se nalaze i tri temperaturna senzora ADT7301 koji služe za kompenzaciju rezultata mjerenja s obzirom na promjenu temperature.

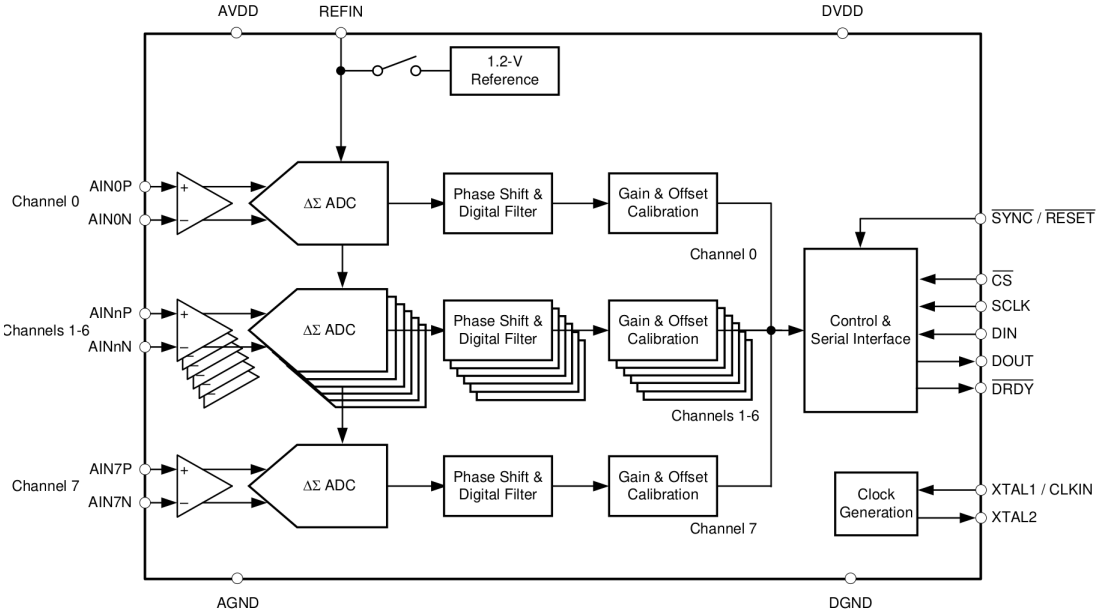
ADC i temperaturni senzori povezani su s PDH računalom jednim SPI sučeljem. Prijenosne linije SPI sučelja, kao i CS linije pojedinih sklopova i drugi upravljački signali, dovedene su do pločice na kojoj se nalazi PDH računalo putem konektora s 40 priključaka, čiji je raspored izvoda prikazan na slici 3.3.



Slika 3.3: Raspored izvoda konektora na senzorskoj pločici

3.2. AD pretvornik ADS131M08

ADS131M08 je delta-sigma analogno-digitalni pretvornik (engl. *analog-to-digital converter, ADC*) s 8 kanala i rezolucijom od 24 bita proizvođača Texas Instruments [14]. Za komunikaciju s mikrokontrolerom koristi se SPI sučelje. Svih 8 kanala uzorkuje se istovremeno, a za svaki kanal moguće je postaviti programibilno pojačanje od 1 do 128. Frekvencija uzorkovanja također je programibilna i može iznositi do 32 tisuće uzoraka u sekundi. Slika 3.5 prikazuje blok dijagram sklopa.

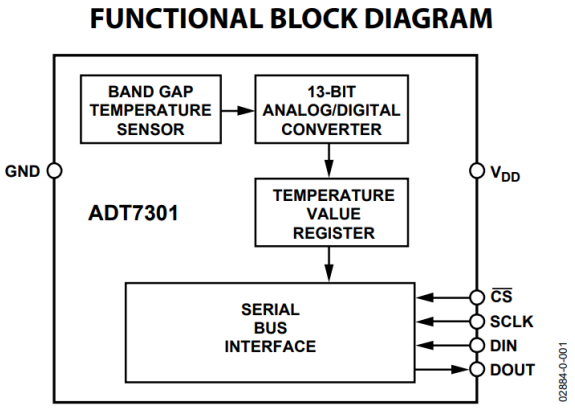


Slika 3.4: Blok dijagram sklopa ADS131M08 [14]

Sklop zahtijeva odvojeno napajanje digitalnog i analognog dijela. Analogno napajanje može biti u rasponu 2.7 - 3.6 V, a digitalno napajanje treba biti 1.8 V ili 3.3 V. Referentni napon može se dovesti na vanjski priključak, ili se može koristiti unutarnji izvor referentnog napona iznosa 1.2 V. Signal takta može se generirati unutar sklopa ili može biti doveden na vanjski priključak.

3.3. Temperaturni senzor ADT7301

Sklop ADT7301 proizvođača Analog Devices je temperaturni senzor s integriranim 13-bitnim analogno-digitalnim pretvornikom i serijskim sučeljem SPI. Omogućuje mjerenje temperature u rasponu od -40°C do 150°C, s rezolucijom 0.03125°C i tipičnom preciznošću ± 0.5°C [1]. Blok dijagram sklopa prikazan je na slici 3.5.



Slika 3.5: Blok dijagram sklopa ADT7301 [1]

Senzor uzima mjerenja temperature svakih 1.5 sekundi, što je regulirano internim sklopom za mjerenje vremena (engl. *timer*). Između dva mjerenja, napajanje analognog sklopovlja senzora je ugašeno te ono postaje neaktivno. Digitalno sklopovlje uvijek je aktivno, ali ako se pokuša pročitati vrijednost temperature više puta unutar jednog intervala mjerenja, senzor će uvijek vraćati istu vrijednost (onu koju je izmjerio na početku intervala).

Dodatna mogućnost sklopa je takozvani *shutdown* način rada. U ovom načinu rada sklop troši vrlo malo struje (oko 1 µA), što je korisno ako postoji dulje vremensko razdoblje u kojem se neće uzimati uzorci temperature. *Shutdown* način rada omogućuje se upisom odgovarajućeg bita u kontrolni registar putem serijskog sučelja.

Prilikom ispitivanja senzora u uvjetima sobne temperature, primijećeno je kako nakon nekoliko minuta kontinuiranog rada očitana temperatura počinje rasti, te može pokazivati vrijednosti čak i do 55°C. Zaključeno je da je navedeno posljedica zagrijavanja samog senzora. Zato je odlučeno da će se senzor između mjerenja stavljati u *shutdown* način rada, kako bi se smanjila potrošnja struje, i samim time disipacija toplinske energije.

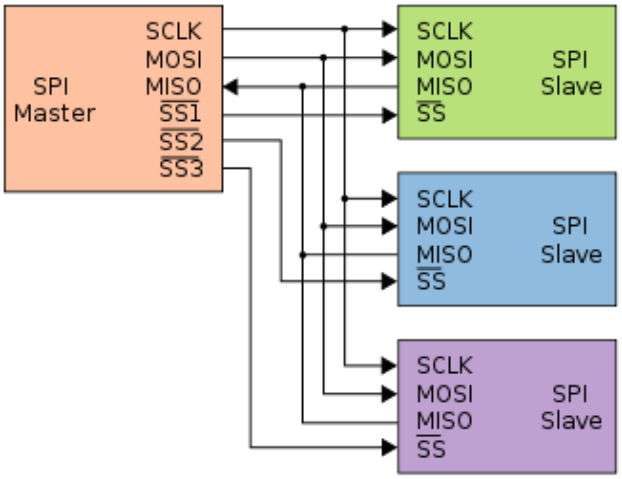
4. Sučelje za komunikaciju sa senzorskim podsustavom

Komponente senzorske pločice FERSAT-a komuniciraju s PDH računalom putem sučelja SPI. Radi boljeg razumijevanja načina komunikacije između komponenata i PDH računala, u nastavku ovog poglavlja prvo je dan općeniti opis SPI protokola i opis SPI periferije porodice mikrokontrolera STM32L4, a zatim je opisan konkretan način komunikacije s AD pretvornikom ADS131M08 i temperaturnim senzorima ADT7301.

4.1. SPI protokol

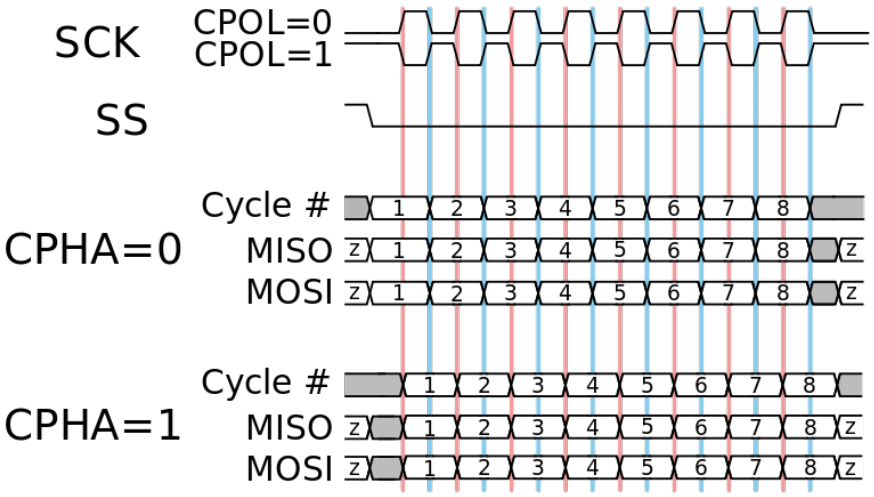
Serial Peripheral Interface (SPI) sinkrono je serijsko komunikacijsko sučelje, razvijeno u tvrtki Motorola. S obzirom da SPI sučelje omogućava brzinu prijenosa do nekoliko desetaka Mbit/s, obično se koristi za prijenos signala visokih frekvencija između računala ili mikrokontrolera i periferijskih sklopova. Podaci se prenose između jedne upravljačke jedinice (engl. *master*) i više upravljanih jedinica (engl. *slave*) korištenjem četiri prijenosne linije: SCLK (*Serial Clock*), MISO (*Master Input Slave Output*), MOSI (*Master Output Slave Input*) i CS (*Chip Select*, ponekad se naziva i *Slave Select*). Signal takta pogoni *master* uređaj, a pomoću linije CS *master* uređaj odabire koji *slave* uređaj smije komunicirati preko linija MISO i MOSI. Slika 4.1 prikazuje tipičan način spajanja uređaja SPI sučeljem, u konfiguraciji jednog *master* uređaja i tri *slave* uređaja.

Postoje 4 temeljna načina rada (engl. *modes*) SPI sučelja, a razlikuju se po polaritetu signala takta (engl. *Clock Polarity, CPOL*) i načinu čitanja podatka sa strane master uređaja (engl. *Clock Phase, CPHA*). CPOL određuje logičku razinu u koju će signal takta poprimiti u neaktivnom stanju (engl. *idle state*). CPOL=0 će postaviti nisku razinu, a CPOL=1 visoku. CPHA određuje na koji brid signala takta će *master* uređaj čitati podatak koji se nalazi na liniji. CPHA=0 obično znači da će to biti prvi brid, a CPHA=1 drugi. Ovisno o postavci CPOL, taj brid može biti rastući ili padajući. Ako je CPOL=0 prvi brid će biti rastući, a ako je CPOL=1, prvi brid će biti padajući. Slika 4.2 prikazuje



Slika 4.1: Povezivanje uređaja SPI sučeljem [16]

vremenske odnose za različite postavke CPOL i CPHA.

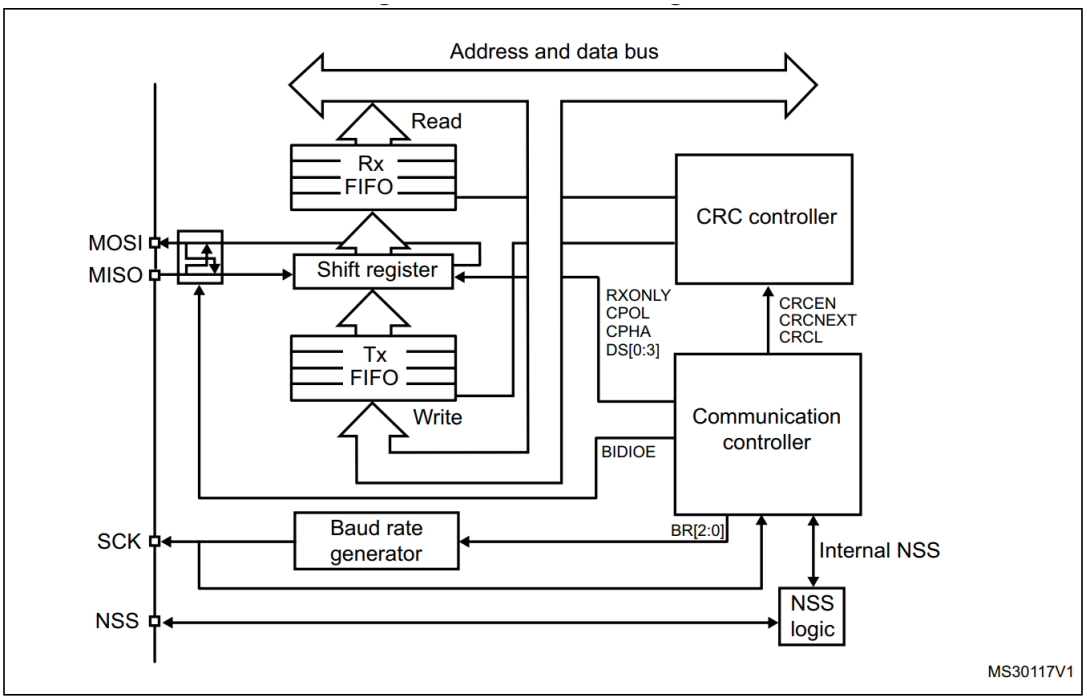


Slika 4.2: Vremenski dijagram SPI prijenosa s različitim postavkama CPOL i CPHA. Crvene linije označavaju prvi, a plave drugi brid signala takta [16]

4.2. Struktura SPI periferije STM32L4

Slika 4.3 prikazuje blok dijagram SPI periferije porodice mikrokontrolera STM32L4 [10].

Primanje podataka odvija se na način da riječ koja se prima po MISO liniji prvo ulazi u posmačni registar (engl. *shift register*), pri čemu se na svaki period SPI takta posmiče



Slika 4.3: Blok dijagram SPI periferije STM32L4 [10, str. 1451]

za jedno mjesto. Kada je primljena cijela riječ, ona se na sljedeći brid takta prebacuje na kraj reda Rx FIFO (*First In First Out*), pri čemu se postavlja zastavica RXNE (*Receiver Buffer Not Empty*) u registru stanja SPI periferije(SPIx_SR). Prvom podatku u redu programski se može pristupiti preko SPI podatkovnog registra (SPIx_DR). Čitanje ovog registra automatski čisti zastavicu RXNE ukoliko je popunjeno manje od četvrtine maksimalnog kapaciteta reda.

Slanje podataka odvija se na sličan način. Riječ upisana u SPIx_DR sprema se na kraj reda Tx FIFO. Prva riječ u redu prebacuje se u posmačni registar, te se izlazni bit pri svakom posmaku šalje po liniji MOSI. Ako Tx FIFO sadrži manje podataka od pola svog kapaciteta, postavlja se zastavica TXE (*Transmitter Buffer Empty*). Postavljanje te zastavice signalizira programu da se sljedeća riječ može upisati u red.

Važno je naglasiti da SPIx_DR nije fizički registar, već se radi o virtualnom registru koji služi za pristup redovima Rx FIFO i Tx FIFO. Pisanje u ovaj registar umeće podatak na kraj reda Tx FIFO, a čitanje sadržaja registra vraća prvi podatak u redu Rx FIFO. Oba reda su veličine 32 bita, odnosno mogu primiti 4 8-bitne riječi. Nivo popunjenosti, tj. broj 8-bitnih riječi u redu može se dobiti čitanjem bitova FTLVL[1:0] za Tx FIFO, odnosno FRLVL[1:0] za Rx FIFO u SPIx_SR.

SPI kontroler može raditi s duljinama riječi od 4 do 16 bita. U izradi ovog rada korištena je duljina riječi 8 bita.

Kontroler omogućuje hardversko izračunavanje CRC zaštitnog koda. Ova je mogućnost nakon reseta isključena, no može se omogućiti postavljanjem bita CRCEN u registru SPIx_CR1. Tada će pogreška u prijenosu koju otkrije CRC biti signalizirana postavljanjem zastavice CRCERR u registru SPIx_SR.

4.3. Postupak slanja i primanja podataka

SPI sučelje može funkcionirati u nekoliko načina rada s obzirom na smjer komunikacije. Podržani načini rada su: *Full Duplex Master, Full Duplex Slave, Half Duplex Master, Half Duplex Slave, Simplex Receive Only* i *Simplex Transmit Only*. U upravljačkim programima izrađenim u sklopu ovog rada korišten je način rada *Full Duplex Master*, pa će zato u nastavku ovog potpoglavlja biti opisan postupak koji upravljački program mora izvršiti za ispravnu komunikaciju u tom načinu rada.

Važno je naglasiti da je u *Full Duplex Master* načinu rada signal SPI takta određen slanjem podataka. SPI kontroler počinje generirati signal takta upisom prve riječi podatka u podatkovni registar, te nastavlja generirati signal takta dok sve riječi nisu poslane. Ako je zadnja riječ poslana i nema novih riječi u redu Tx FIFO, kontroler prestaje s generiranjem takta do sljedećeg upisa u podatkovni registar.

Upravljački program mora slijediti sljedeću proceduru[1]:

1. Omogućiti SPI postavljanjem bita SPE u registru SPIx_CR1.
2. Upisati prvu riječ za slanje u podatkovni registar.
3. Čekati dok se ne postavi zastavica TXE i zatim upisati sljedeću riječ u podatkovni registar. Čekati dok se ne postavi zastavica RXNE i zatim pročitati riječ iz podatkovnog registra. Ponavljati ovaj korak do (uključivo) predzadnje pročitane riječi.
4. Čekati dok se ne postavi zastavica RXNE i pročitati zadnju riječ iz podatkovnog registra.

Čekanje zastavice TXE prilikom upisa druge riječi nije obavezno jer će ona sigurno biti postavljena, odnosno Tx FIFO sigurno može primiti dvije 8-bitne riječi. Međutim, čekanje je potrebno za svaki sljedeći upis.

Gornja procedura implementirana je funkcijom `SPI_TransmitReceive()`, koja pruža praktično sučelje za SPI prijenos po uzoru na HAL biblioteke.

[1] Ova procedura je prilagođena verzija procedure opisane u priručniku mikrokontrolerske porodice STM32F4 [9, str. 887].

```
1  void SPI_TransmitReceive(SPI_TypeDef *SPIx, uint8_t len, uint8_t
       *tx_buffer, uint8_t *rx_buffer)
2  {
3    LL_SPI_Enable(SPIx);
4
5    LL_SPI_TransmitData8(SPIx, tx_buffer[0]);
6    for (int i = 0; i < len - 1; i++) {
7      while ( !LL_SPI_IsActiveFlag_TXE(SPIx) );
8      LL_SPI_TransmitData8(SPIx, tx_buffer[i + 1]);
9      while ( !LL_SPI_IsActiveFlag_RXNE(SPIx) );
10     rx_buffer[i] = LL_SPI_ReceiveData8(SPIx);
11   }
12
13   while ( !LL_SPI_IsActiveFlag_RXNE(SPIx) );
14   rx_buffer[len - 1] = LL_SPI_ReceiveData8(SPIx);
15 }
```

Odsječak koda 4.1: Funkcija za slanje i primanje podataka putem SPI sučelja

Ako je nakon dovršenog SPI prijenosa potrebno staviti mikrokontroler u način rada male potrošnje, tada treba na ispravan način onemogućiti SPI. U nekim drugim SPI načinima rada potrebno je onemogućiti SPI nakon svakog prijenosa, npr. ako u *Receive Only Master* načinu rada mikrokontroler komunicira sa sklopom koji kontinuirano šalje podatke, kako bi se spriječilo slanje dodatnih neželjenih podataka. U *Full Duplex* načinu rada to obično nije potrebno, međutim dobra je praksa slijediti pravilnu proceduru za onemogućavanje SPI sučelja nakon završetka prijenosa kako bi se spriječila pogreška tijekom prijenosa zadnje odlazne riječi. Procedura za onemogućavanje je:

1. Čekati dok se ne isprazni odlazni red (TX FIFO) provjerom bitova FTLVL[1:0] u SPI statusnom registru.
2. Čekati dok se ne spusti zastavica BSY u SPI statusnom registru.
3. Onemogućiti SPI čišćenjem bita SPE u registru CR1.

Ova je procedura implementirana funkcijom `SPI_Disable()`:

```
1  void SPI_Disable(SPI_TypeDef *SPIx) {
2    while (LL_SPI_GetTxFIFOLevel(SPIx) != LL_SPI_TX_FIFO_EMPTY);
3    while (LL_SPI_IsActiveFlag_BSY(SPIx));
4    LL_SPI_Disable(SPIx);
```

```
5 }
```

Odsječak koda 4.2: Funkcija za onemogućavanje SPI sučelja

Tijekom uhodavanja SPI komunikacije sa sklopom ADT7301 uočena je greška u implementaciji HAL biblioteke za rad s SPI periferijom kada je SPI u *Receive Only* načinu rada. Naime, u navedenom načinu rada potrebno je očistiti SPE bit točno jedan ciklus SPI takta nakon primitka predzadnje riječi kako bi se spriječilo da uređaj koji šalje podatak inicira prijenos nove riječi [9, str. 894]. HAL funkcija `HAL_SPI_TransmitReceive()` čisti SPE bit tek nakon primitka zadnje riječi, zbog čega senzor šalje još jednu riječ, pa ukupna duljina prijenosa iznosi 24 bita umjesto 16. Također, funkcija čeka da se sklopovski očisti zastavica RXNE prije nego što završi s izvođenjem, a s obzirom da se zadnja riječ nikad ne pročita, zastavica uvijek ostaje postavljena. Zbog toga funkcija uvijek čeka do isteka *timeout* intervala, što nije poželjno ponašanje. No, to nije predstavljalo problem u daljnem tijeku izrade ovog rada, jer su za razvoj korištene LL biblioteke i *Full Duplex* način rada.

4.4. Prijenos podataka korištenjem DMA sklopa

Prijenos podataka SPI sučeljem između PDH računala i AD pretvornika podrazumijeva slanje relativno velike količine podataka, što može prilično dugo trajati, ovisno o brzini prijenosa. Zato je korisno koristiti DMA (*Direct Memory Access*) sklop za prijenos podataka između SPI periferije i memorije. Za vrijeme DMA prijenosa procesor je rasterećen pa može obavljati druge korisne zadatke, odnosno moguće je ostvarenje neblokirajućih funkcija za SPI prijenos. Dodatna prednost je i smanjenje ukupnog trajanja komunikacije. Naime, kada se SPI komunikacija obavlja programski, program mora prvo upisati odlaznu riječ u podatkovni registar, zatim čekati primitak cijele dolazne riječi, te pročitati riječ prije upisa sljedeće riječi. To ponekad može uzrokovati diskontinuirani prijenos, odnosno periode čekanja između slanja dvije riječi [9, str. 890]. Korištenjem DMA slanje i primanje može se obavljati istovremeno, što eliminira ovaj problem.

Odabrani mikrokontroler sadrži dva DMA sklopa, a svaki od njih ima 7 prijenosnih kanala. Periferija SPI3 spojena je na sklop DMA2, i to tako da je linija zahtjeva za dolazni prijenos (RX) spojena na kanal 1, a odlazni prijenos (TX) na kanal 2.

Prije korištenja DMA sklopa potrebno je obaviti određenu inicijalizaciju. Većina te inicijalizacije, kao što je omogućavanje prekida, postavljanje smjera prijenosa i slično, može se obaviti u alatu CubeMX, pa će ovdje biti opisani samo oni koraci koji se moraju

obaviti nakon CubeMX inicijalizacije, s obzirom da su oni posebno važni za razvoj programske potpore. Ne smije se zaboraviti ni inicijalizacija SPI periferije, kako bi ona mogla slati zahtjeve za DMA prijenos. U nastavku je opisana potrebna procedura i programski kod kojim se ta procedura implementira, te je uz svaki korak procedure naveden broj linije koja implementira taj korak.

1. Postaviti adresu periferije za svaki kanal upisom u registar CPAR. Ova se adresa neće inkrementirati nakon svakog prijenosa (2).

2. Postaviti adresu memorije za svaki kanal upisom u registar CMAR. Ova će adresa biti inkrementirana nakon svakog prijenosa, a iznos inkrementa ovisi o veličini riječi (3).

3. Omogućiti *Transfer Complete* (TC) prekide za svaki kanal upisom bita TCIE u registru CCR (4).

4. Postaviti duljinu prijenosa, odnosno broj riječi koje se trebaju prenijeti za svaki kanal upisom u registar CNDTR (8).

5. Omogućiti SPI RX zahtjeve za DMA prijenos upisom bita RXDMAEN u SPI registru CR2 (12).

6. Omogućiti odgovarajuće DMA kanale upisom bita EN u registru CCR za svaki kanal (16, 17).

7. Omogućiti SPI TX zahtjeve za DMA prijenos upisom bita TXDMAEN u SPI registru CR2 (21).

8. Omogućiti prekide koje okidaju SPI zastavice RXNE i TXE upisom bitova RXNEIE odnosno TXEIE u SPI registru CR2 (25, 26).

```
1  void DMA_Channel_Init(DMA_TypeDef *DMAx, uint32_t channel,
       uint32_t periph_addr, uint8_t *mem_addr) {
2    LL_DMA_SetPeriphAddress(DMAx, channel, periph_addr);
3    LL_DMA_SetMemoryAddress(DMAx, channel, (uint32_t)mem_addr);
4    LL_DMA_EnableIT_TC(DMAx, channel);
5  }
6
7  void DMA_Set_Channel_Data_Length(DMA_TypeDef *DMAx, uint32_t
       channel, uint32_t length) {
8    LL_DMA_SetDataLength(DMAx, channel, length);
```

```
 9 }
10
11 void SPI_Enable_DMA_Rx_Request(SPI_TypeDef *SPIx) {
12   LL_SPI_EnableDMAReq_RX(SPIx);
13 }
14
15 void DMA_Enable_CH1_CH2(DMA_TypeDef *DMAx) {
16   LL_DMA_EnableChannel(DMAx, LL_DMA_CHANNEL_1);
17   LL_DMA_EnableChannel(DMAx, LL_DMA_CHANNEL_2);
18 }
19
20 void SPI_Enable_DMA_Tx_Request(SPI_TypeDef *SPIx) {
21   LL_SPI_EnableDMAReq_TX(SPIx);
22 }
23
24 void SPI_Enable_RXNE_TXE_Interrupts(SPI_TypeDef *SPIx) {
25   LL_SPI_EnableIT_RXNE(SPIx);
26   LL_SPI_EnableIT_TXE(SPIx);
27 }
```

Odsječak koda 4.3: Inicijalizacija DMA sklopa za SPI prijenos u *Full Duplex* načinu rada

Po završetku DMA prijenosa za svaki kanal, bit će generiran *Transfer Complete* (TC) prekid. Bitno je primijetiti da se prekid generira za svaki kanal posebno. U prekidnoj rutini tih prekida potrebno je očistiti zastavicu CTCFIFx (za kanal x) u DMA registru IFCR i onemogućiti odgovarajući kanal kako bi se spriječio nastavak prijenosa na sljedeći SPI zahtjev kada se postavi zastavica TXE ili RXNE. Kada završe oba prekida, potrebno je onemogućiti SPI RX i TX DMA zahtjeve. Programski kod koji obavlja ovu zadaću dan je u nastavku. S obzirom da prekidne rutine TC prekida izgledaju gotovo identično za oba kanala, navedena je samo jedna od njih.

```
 1 void DMA_Transfer_Complete_RX_interrupt_handler() {
 2   LL_DMA_ClearFlag_TC1(DMA2);
 3   LL_DMA_DisableChannel(DMA2, LL_DMA_CHANNEL_1);
 4   if (tc == 1) {
 5     DMA_Disable(SB_SPIx);
 6     tc = 0;
 7   } else {
 8     tc = 1;
 9   }
10 }
11
```

```
12 void DMA_Disable(SPI_TypeDef *SPIx) {
13   SPI_Disable(SPIx);
14   SPI_Disable_DMA_Requests(SPIx);
15   // Postavi CS prikljucak u visoku razinu
16   LL_GPIO_SetOutputPin(ADC_CS_GPIOx, ADC_CS_PIN);
17   // Omoguci DRDY prekide ADC-a
18   NVIC_EnableIRQ(EXTI4_IRQn);
19 }
```

Odsječak koda 4.4: Prekidna rutina prekida *DMA Transfer Complete*

Globalna varijabla `tc` je zastavica koja signalizira je li se prekid drugog kanala već dogodio. Tek kada oba kanala završe s prijenosom može se onemogućiti SPI i omogućiti nove prekide ADC-a.

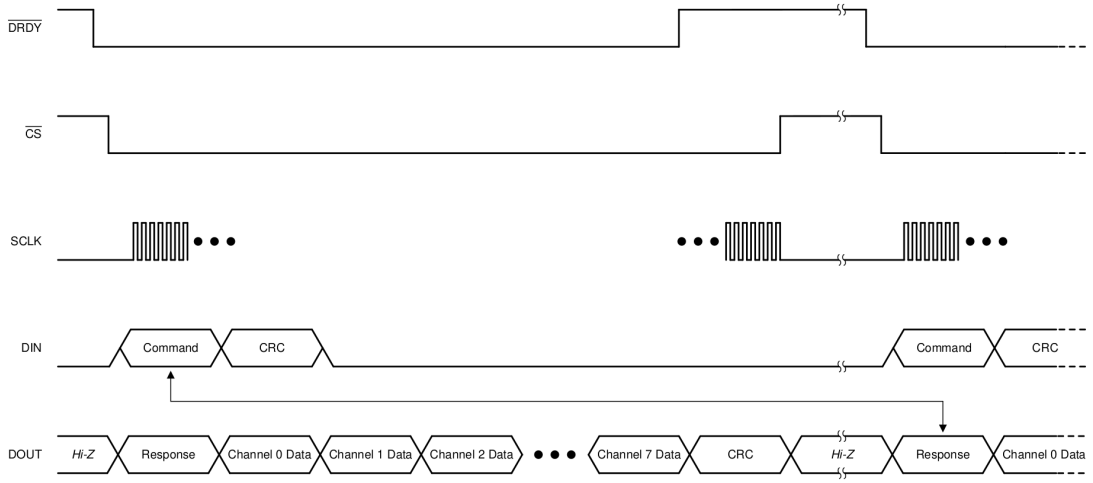
Prije pokretanja novog prijenosa potrebno je ponoviti korake 4 do 8 inicijalizacijske procedure.

4.5. SPI sučelje sklopa ADS131M08

SPI sučelje sklopa ADS131M08 koristi postavke CPOL = 0 i CPHA = 1, što znači da niska logička razina odgovara neaktivnom stanju takta i da se podatak čita na drugi brid takta (padajući). Za komunikaciju se koriste standardni SPI priključci (SCLK, MOSI, MISO i $\overline{\text{CS}}$) i dva dodatna priključka: $\overline{\text{DRDY}}$ (*Data Ready*) i $\overline{\text{SYNC/RESET}}$.

$\overline{\text{DRDY}}$ priključak postaje aktivan u trenutku kada je sklop spreman poslati rezultate konverzije. Aktiviranje priključka može se iskoristiti za okidanje prekida na mikrokontroleru, što omogućuje pravovremeno čitanje. Posebnu pažnju treba obratiti kada se podaci čitaju prvi put nakon uključenja ili kada je prošlo dulje vrijeme od zadnjeg čitanja. Naime, ADC ima unutarnji međuspremnik u kojem se spremaju rezultati konverzije ako nisu pročitani. Međuspremnik ima dovoljno kapaciteta za spremanje posljednje dvije konverzije. Kada je međuspremnik pun, ponašanje priključka $\overline{\text{DRDY}}$ neće biti dosljedno. Proizvođač zato preporuča dva načina za sinkronizaciju: prvi je pročitati dva uzorka zaredom bez čekanja aktivacije $\overline{\text{DRDY}}$ priključka i tako isprazniti međuspremnik [14]. Drugi način sinkronizacije je postavljanje pravokutnog impulsa trajanja duljeg od jednog perioda signala takta ADC-a, ali kraćeg od 2048 perioda na priključak $\overline{\text{SYNC/RESET}}$ [14]. Prilikom postavljanja parametara SPI komunikacije, treba voditi računa da frekvencija SPI takta bude dovoljno velika da prijenos podataka traje kraće od perioda uzorkovanja ADC-a, odnosno perioda aktivacije signala $\overline{\text{DRDY}}$. U suprotnom ponašanje signala $\overline{\text{DRDY}}$ neće biti u skladu sa specifikacijom.

Podaci se prenose preko SPI sučelja u takozvanim okvirima. Svaki okvir sastoji se od 10 riječi (ili manje, ako su neki kanali onemogućeni). Duljina riječi može se postaviti na 16, 24 ili 32 bita, a pretpostavljena duljina je 24 bita. Sučelje radi u *Full-Duplex* načinu rada. Vremenski dijagram jednog okvira prikazan je na slici 4.4.



Slika 4.4: Vremenski dijagram SPI komunikacije. Strelica označava naredbu i pripadajući odgovor. [14]

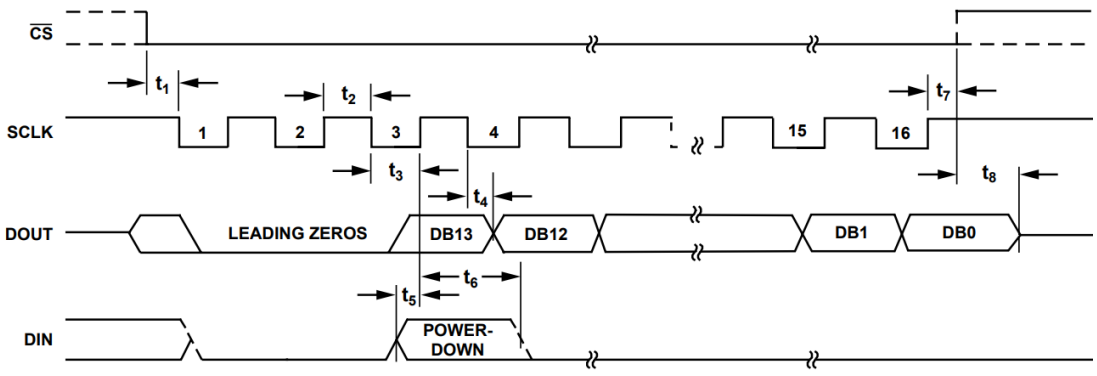
Prva riječ okvira na liniji MOSI, na slici označeno kao DIN, jest naredba koju *master* uređaj šalje ADC-u. Najčešće korištena naredba je *Null* naredba, čiji je instrukcijski kod 0x0000. Kada primi tu naredbu, ADC neće obaviti nikakvu operaciju, već će samo poslati rezultate konverzije. Neke druge naredbe su RREG (čitanje registra), WREG (upis u registar) i RESET. Druga riječ okvira je CRC (*Cyclic Redundancy Check*) zaštitni kod poruke (ili riječ bitova u niskoj razini ako je CRC onemogućen), nakon čega slijedi 8 riječi s bitovima u niskoj logičkoj razini.

Prva riječ okvira na liniji MISO, na slici označeno kao DOUT jest odgovor na naredbu u prethodnom okviru. Ako je naredba bila *Null*, tada je odgovor ispis sadržaja registra stanja ADC-a. U slučaju neke druge naredbe odgovor se može sastojati i od više riječi, primjerice ako se radi o ispisu sadržaja nekoliko registara. Nakon odgovora slijedi 8 riječi, po jedna za svaki kanal, koje sadrže rezultate AD pretvorbe. Moguće je onemogućiti neke kanale korištenjem naredbe WREG, u tom će slučaju riječ koja odgovara tom kanalu biti izostavljena. Posljednja riječ u okviru je CRC zaštitni kod poruke.

4.6. SPI sučelje sklopa ADT7301

Temperaturni senzor ADT7301 koristi SPI postavke CPOL = 1 i CPHA = 1, što znači da visoka logička razina signala takta odgovara neaktivnom stanju i da se podatak čita na drugi brid takta (rastući).

ADT7301 ima mogućnost istovremenog slanja i primanja podataka (engl. *Full Duplex*). Na svojem priključku DOUT, koji je spojen na SPI liniju MOSI, sklop daje 16-bitni izlazni podatak na način da najznačajniji bit podatka izlazi prvi. Bitovi 15 i 14 su u niskoj logičkoj razini, bit 13 je bit predznaka, a ostali bitovi predstavljaju apsolutnu vrijednost očitane temperature. Na priključku DIN, koji je spojen na SPI liniju MISO, sklop prima 16-bitni podatak, gdje svi bitovi osim trećeg najznačajnijeg bita moraju biti u niskoj logičkoj razini. Treći najznačajniji bit označava hoće li sklop prijeći u *shutdown* način rada po završetku ciklusa slanja. Ako je taj bit u visokoj logičkoj razini, sklop će prijeći u *shutdown* način rada, a ako nije, sklop će nastaviti rad u dosadašnjem načinu rada. Slika 4.5 prikazuje jedan SPI ciklus čitanja/pisanja.

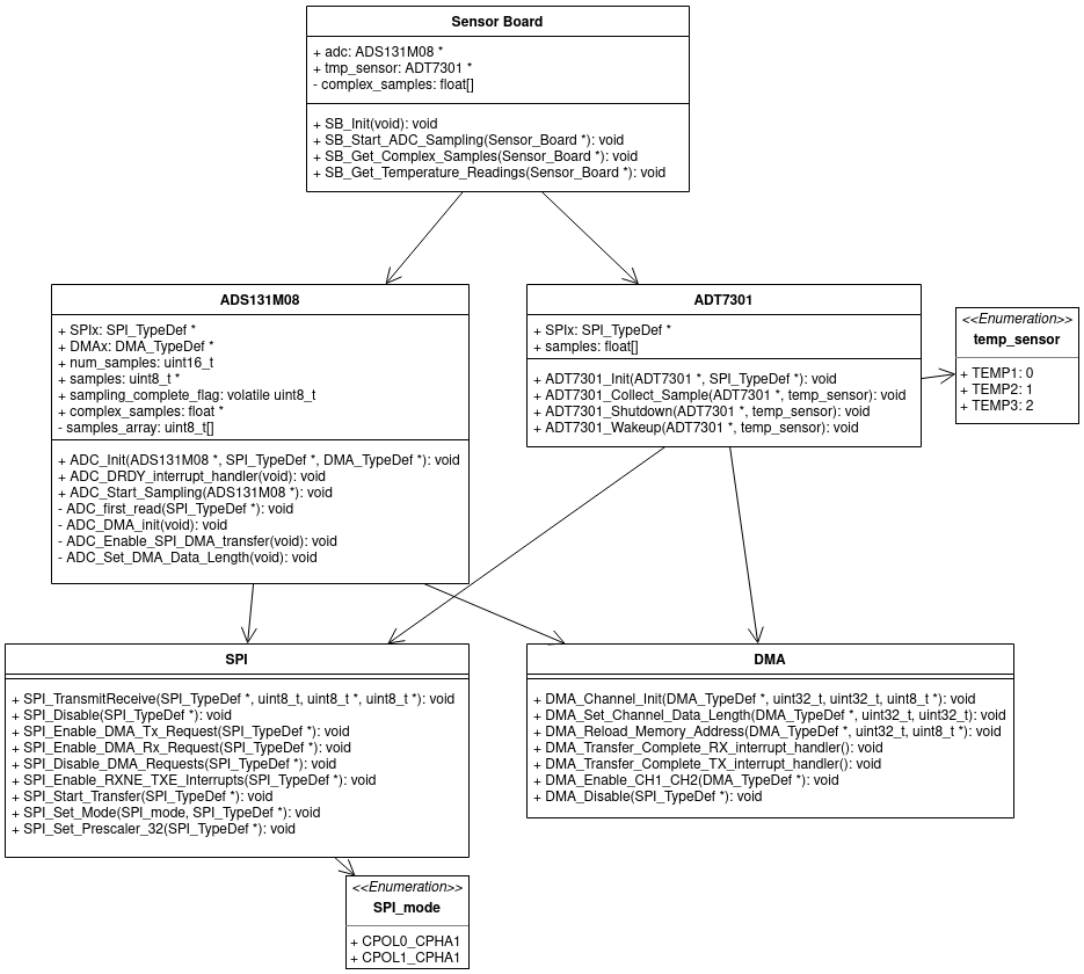


Slika 4.5: Vremenski dijagram SPI komunikacije sklopa ADT7301 [1]

5. Razvijena programska potpora

5.1. Struktura programske potpore

Na slici 5.1 prikazana je struktura razvijene programske potpore u obliku UML dijagrama razreda. U nastavku poglavlja bit će detaljnije opisani upravljački programi za sklopove ADS131M08 i ADT7301, kao i funkcije za prikupljanje i obradu podataka u modulu *Sensor Board*.



Slika 5.1: Prikaz strukture programske potpore UML dijagramom razreda

5.2. Korištene biblioteke i alati

Tvrtka ST Microelectronics nudi dva skupa programskih biblioteka za razvoj programske potpore namijenjene njihovim mikrokontrolerima: *Hardware Abstraction Layer* (HAL) i *Low Level* (LL) biblioteke. HAL biblioteke nude višu razinu apstrakcije sklopovlja od LL biblioteka, što omogućuje jednostavniji i brži razvoj, te olakšanu prenosivost između različitih mikrokontrolera istog proizvođača. S druge strane, LL biblioteke nude vrlo nisku razinu apstrakcije, što zahtijeva vrlo dobro poznavanje strukture i načina rada mikrokontrolera od strane programera, ali zato omogućuje pisanje programa koji zauzimaju vrlo malu količinu memorije [11]. Također, korištenje LL biblioteka smanjuje ukupno vrijeme izvođenja jer se eliminiraju česte provjere parametara funkcija ugrađene u HAL biblioteke. Upravo zbog vrlo ograničenih resursa dostupnih na odabranom mikrokontroleru, tijekom razvoja programske potpore opisane u radu [8] donesena je odluka o korištenju LL biblioteka. Iz tog razloga LL biblioteke korištene su i u razvoju programske potpore opisane u ovom radu.

Programska potpora razvijena je u programskom jeziku C, uz korištenje prevoditelja GCC u sklopu paketa *GNU ARM Embedded Toolchain*. Za razvoj je korišteno razvojno okruženje STM32CubeIDE, s integriranim alatom CubeMX. Navedeni alat omogućuje automatsko generiranje programskog koda za inicijalizaciju periferija mikrokontrolera i jednostavnu konfiguraciju signala takta kroz grafičko sučelje.

5.3. Upravljački program za AD pretvornik ADS131M08

Zadaća upravljačkog programa za ADS131M08 je prikupiti rezultate AD pretvorbe putem SPI sučelja. Zbog velike količine podataka koje je potrebno prikupiti, za prijenos podataka između SPI periferije i memorije koristi se DMA sklop. Upravljački su programi pisani tako da budu neovisni o bilo kakvoj programskoj logici više razine, pa ih je moguće ponovno iskoristiti u budućim nadogradnjama programske potpore ili u drugim projektima. Funkcionalnost programa ostvarena je pomoću tri glavne funkcije:

```
1 void ADC_Init(ADS131M08 *adc_struct, SPI_TypeDef *SPIx,
     DMA_TypeDef *DMAx);
2 void ADC_Start_Sampling(ADS131M08 *adc);
3 void ADC_DRDY_interrupt_handler();
```

Odsječak koda 5.1: Funkcije upravljačkog programa za sklop ADS131M08

Također, definirana je i struktura `ADS131M08` preko koje se razmjenjuju konfiguracijski podaci između pozivajućeg modula i upravljačkog programa, te čuvaju pokazivači na prikupljene podatke:

```
1 typedef struct {
2     SPI_TypeDef *SPIx;
3     DMA_TypeDef *DMAx;
4     uint16_t num_samples;
5     uint8_t *samples;
6     volatile uint8_t sampling_complete_flag;
7     float *complex_samples;
8 } ADS131M08;
```

Odsječak koda 5.2: Definicija tipa `ADS131M08`

Funkcija `ADC_Init()` inicijalizira članove strukture `ADS131M08`, postavlja parametre SPI komunikacije tako da odgovaraju onima koje podržava sklop ADS131M08, i poziva pomoćnu funkciju `ADC_DMA_Init()` postavlja izvorišnu i odredišnu adresu DMA prijenosa i omogućuje DMA prekide.

Nakon što je inicijalizacija dovršena, dohvaćanje podataka može se pokrenuti funkcijom `ADC_Start_Sampling()`. Ova funkcija prvo poziva pomoćnu funkciju `ADC_First_Read()` koja obavlja čišćenje međuspremnika ADC-a s dva uzastopna čitanja kao što je opisano u potpoglavlju 3.2. Zatim se omogućuje prekid kojeg izaziva padajući brid signala na priključku $\overline{\text{DRDY}}$ ADC-a, čime započinje prikupljanje podataka.

Navedeni prekid doveden je na prekidnu liniju EXTI4 prekidnog kontrolera. Rutina koja obrađuje prekid poziva funkciju `ADC_DRDY_interrupt_handler()`. Funkcija postavlja duljinu prijenosa i odredišnu adresu, omogućuje DMA prijenos pozivanjem funkcija koje postavljaju odgovarajuće bitove u registrima SPI i DMA periferija, te postavlja priključak $\overline{\text{CS}}$ ADC-a u nisku logičku razinu. Nakon toga SPI prijenos se odvija automatski sklopovskom sinkronizacijom između SPI i DMA periferija. Po završetku zadanog broja prijenosa onemogućuje se $\overline{\text{DRDY}}$ prekid.

Podaci prikupljeni s ADC-a spremaju se u statički alocirano polje 8-bitnih cijelih brojeva. Jedan uzorak svakog od kanala veličine je 24 bita, odnosno 3 bajta. Ukupna duljina podataka jednog okvira, koji sadrži uzorke svih 8 kanala, ispis statusnog registra ADC-a i CRC zaštitni kod poruke, je 30 bajtova. Prikuplja se 256 okvira, pa ukupna duljina podataka iznosi 7680 bajtova.

5.4. Upravljački program za temperaturni senzor ADT7301

Upravljački program za temperaturni senzor prikuplja uzorke temperature od senzora ADT7301. Razvijene su četiri funkcije za rad sa senzorom:

```
1 void ADT7301_Init(ADT7301 *adt7301_struct, SPI_TypeDef *SPIx);
2 void ADT7301_Collect_Sample(ADT7301 *adt7301, temp_sensor ts);
3 void ADT7301_Shutdown(ADT7301 *adt7301, temp_sensor ts);
4 void ADT7301_Wakeup(ADT7301 *adt7301, temp_sensor ts);
```

Odsječak koda 5.3: Funkcije upravljačkog programa za sklop ADT7301

Na senzorskoj pločici nalaze se tri temperaturna senzora, pa zato funkcije za prikupljanje podataka primaju parametar `ts` koji označava s kojeg senzora treba prikupiti podatak. Parametar je enumeracijskog tipa `temp_sensor`:

```
1 typedef enum {
2     TEMP1 = 0,
3     TEMP2 = 1,
4     TEMP3 = 2
5 } temp_sensor;
```

Odsječak koda 5.4: Definicija tipa `temp_sensor`

Slično kao i kod upravljačkog programa za AD pretvornik, parametri i prikupljeni podaci čuvaju se u strukturi ADT7301:

```
1 typedef struct {
2   SPI_TypeDef *SPIx;
3   float samples[3];
4 } ADT7301;
```

Odsječak koda 5.5: Definicija tipa `ADT7301`

Funkcija `ADT7301_Init()` inicijalizira članove strukture `ADT7301` i konfigurira SPI komunikaciju. Ostale tri funkcije imaju međusobno sličnu funkcionalnost - šalju i primaju dva bajta podataka s odabranog temperaturnog senzora korištenjem funkcije `SPI_TransmitReceive()`.

Funkcija `ADT7301_Collect_Sample()` obavlja pretvorbu primljenog podatka u stvarnu vrijednost temperature (iskazanu u °C) u formatu realnog broja s pomičnim zarezom, i zapisuje vrijednost u polje `samples` strukture `ADT7301`. Mjesto u polju

na koje će biti zapisana vrijednost ovisi o odabranom senzoru `ts` (indeks 0, 1 ili 2).

Funkcija `ADT7301_Shutdown()` šalje naredbu za postavljanje senzora u način rada niske potrošnje (*shutdown*). Podatak primljen sa senzora zanemaruje se.

Funkcija `ADT7301_Wakeup()` „budi“ senzor iz načina rada niske potrošnje. Primljeni podatak također se zanemaruje.

5.5. Programska potpora za prikupljanje i obradu podataka

Programska potpora za prikupljanje i obradu podataka senzorskog podsustava grupirana je u modul `sensor_board`. Razvijena programska potpora uključuje prikupljanje podataka korištenjem prethodno opisanih upravljačkih programa za sklopove ADS131M08 i ADT7301, te pripremu podataka o svjetlosnom onečišćenju prikupljenih s ADC-a za računanje spektra algoritmom *Fast Fourier Transform* (FFT).

Za prikupljanje podataka razvijene su sljedeće funkcije:

```
1 void SB_Init(Sensor_Board *sb);
2 void SB_Start_ADC_Sampling(Sensor_Board *sb);
3 void SB_Get_Temperature_Readings(Sensor_Board *sb);
```

Odsječak koda 5.6: Funkcije za prikupljanje podataka senzorskog podsustava

Funkcija `SB_Init()` inicijalizira strukturu `Sensor_Board`, koja enkapsulira strukture `ADS131M08` i `ADT7301`:

```
1 typedef struct {
2     ADS131M08 *adc;
3     ADT7301 *tmp_sensor;
4 } Sensor_Board;
```

Odsječak koda 5.7: Definicija tipa `Sensor_Board`

Funkcija `SB_Start_ADC_Sampling()` inicijalizira ADC, a zatim pokreće postupak dohvaćanja uzoraka s ADC-a pozivom funkcije `ADC_Start_Sampling()`. Funkcija je neblokirajuća, dakle moguće je obavljati druge zadatke za vrijeme dohvaćanja uzoraka.

Za dohvaćanje podataka s temperaturnih senzora namijenjena je funkcija `SB_Get_Temperature_Readings()`. Funkcija inicijalizira temperaturne sen-

zore, a zatim svaki od tri senzora prvo „budi“ iz *shutdown* načina rada, prikuplja uzorak temperature sa senzora i vraća ga u *shutdown* način rada.

Kako bi se iz podataka dobivenih od fotodetektora mogla dobiti informacija o spektralnoj gustoći svjetla sa Zemlje, potrebno je izvršiti određenu obradu podataka. Prvi je korak u toj obradi računanje spektra diskretnog signala algoritmom FFT. Uzastopni uzorci s jednog fotodetektora, odnosno uzorci s jednog kanala ADC-a, promatraju se kao vremenski niz, tj. diskretni signal u vremenskoj domeni. Uzima se 256 uzoraka svakog kanala kako bi se mogla izračunati transformacija u 256 točaka. To je najveći broj uzoraka koji se može prikupiti s obzirom na ograničenja memorije[1]. Za računanje FFT koristi se implementacija iz CMSIS (*Cortex Microcontroller Software Interface Standard*) DSP (*Digital Signal Processing*) biblioteke. Ova biblioteka pruža programsko sučelje za korištenje algoritama za digitalnu obradu signala koji su posebno optimirani za izvođenje na procesorskim jezgrama Cortex-M koje proizvodi tvrtka Arm [2].

Funkcija iz CMSIS DSP biblioteke koja implementira brzu Fourierovu transformaciju nad realnim nizovima 32-bitnih brojeva s pomičnim zarezom ima sljedeću deklaraciju:

```
1 void arm_rfft_fast_f32( const arm_rfft_fast_instance_f32 *S,
2                         float32_t *p,
3                         float32_t *pOut,
4                         uint8_t ifftFlag );
```

Odsječak koda 5.8: Deklaracija funkcije `arm_rfft_fast_f32`

Parametar `S` je struktura s konfiguracijskim podacima za algoritam (između ostalog sadrži i duljinu niza), `p` je pokazivač na ulazni niz, `pOut` je pokazivač na izlazni niz, a `ifftFlag` je zastavica koja označava radi li se o inverznoj transformaciji.

Iako navedena funkcija radi s realnim ulaznim nizovima (pa se taj algoritam još naziva i *Real Fast Fourier Transform* - RFFT), brza Fourierova transformacija definirana je nad kompleksnim brojevima (*Complex Fast Fourier Transform* - CFFT). Zato se polje s ulaznim nizom mora formatirati na način da se svaki podatak prikazuje kao zbroj svoje realne i imaginarne vrijednosti. Primjerice, formatirano polje s N ulaznih podataka simbolički možemo prikazati na sljedeći način, gdje `real[i]` označava realnu vrijednost podatka s rednim brojem `i`, a `imag[i]` njegovu imaginarnu vrijednost:

[1]Da bi FFT algoritam radio efikasno, broj uzoraka treba biti potencija broja 2. Za računanje FFT u 512 točaka potrebno je više radne memorije nego što je dostupno na odabranom mikrokontroleru, pa se zato računa FFT u 256 točaka.

`[real[0],imag[0],real[1],imag[1],...,real[N-1],imag[N-1]]`

Izlazno je polje formatirano na isti način. No, zbog svojstva simetrije spektra realnih signala, druga polovica spektra u potpunosti je određena prvom polovicom. Za takve signale, realni dio spektra je parna funkcija, a imaginarni dio neparna, pa se svaki uzorak druge polovice spektra može prikazati kao konjugirano-kompleksna vrijednost odgovarajućeg uzorka iz prve polovice [7]. Zato je spektar signala duljine N uzoraka moguće u prikazati s $\frac{N}{2} + 1$ kompleksnih brojeva. Osim toga, prvi i posljednji od tih brojeva su čisto realni, pa se oni prikazuju kao prva dva elementa izlaznog polja. Zato izlazno polje ima dvostruko manju duljinu od ulaznog, i može se prikazati u obliku:

`[real[0],real[N/2],real[1],imag[1],...,real[N/2-1],imag[N/2-1]]`

FFT obrada signala zahtijeva znatnu količinu procesorskih resursa, pa će se zato obavljati u vremenu kada su ostali sustavi korisnog tereta satelita neaktivni (engl. *down-time*), a programski će za nju biti zadužen zadatak neovisan od zadatka za upravljanje senzorskim podsustavom. U okviru ovog rada realizirano je poravnanje podataka dobivenih s ADC-a tako da oni budu u obliku pogodnom za korištenje kao ulazno polje funkcije `arm_rfft_fast_f32()`, i zapisivanje takvih podataka u trajnu memoriju (engl. *non-volatile memory*). Program za obradu podataka tada može jednostavno učitati podatke iz trajne memorije, provesti FFT transformaciju i ostalu obradu koja slijedi nakon nje, i zapisati rezultate natrag u trajnu memoriju.

Formatiranje podataka u polje kompleksnih realnih brojeva s pomičnim zarezom ostvaruje sljedeća funkcija:

```
1 void SB_Get_Complex_Samples(Sensor_Board *sb);
```

Odsječak koda 5.9: Deklaracija funkcije `SB_Get_Complex_Samples`

Funkcija pretvara 24-bitne rezultate AD pretvorbe u 16-bitne (vodeći računa da se preskoče nepotrebni podaci dobiveni s ADC-a kao što je ispis statusnog registra), zatim ih pretvara u stvarne vrijednosti napona s obzirom na referentni napon ADC-a i zapisuje ih u izlazno polje kao kompleksne brojeve s imaginarnim dijelom jednakim 0. Izlazno polje je statički alocirano i veličine je 16 384 bajta, a u njemu se nakon poziva funkcije nalaze redom uzorci sa svih 8 fotodetektora (prvo 256 kompleksnih uzoraka s kanala 0, zatim 256 kompleksnih uzoraka s kanala 1, itd.).

Ispravan rad funkcije ispitan je tako što je nakon njenog poziva izvršena FFT transformacija rezultata AD pretvorbe jednog kanala korištenjem funkcije `arm_rfft_fast_f32()`. Za ispitivanje je korišten sljedeći odsječak programskog koda:

```
1 arm_status status;
2 arm_rfft_fast_instance_f32 S;
3 static float32_t fft_out[NUM_SAMPLES];
4 status = arm_rfft_fast_init_f32(&S, NUM_SAMPLES);
5 arm_rfft_fast_f32(&S, (float32_t *) sb.adc->complex_samples,
      fft_out, 0);
```

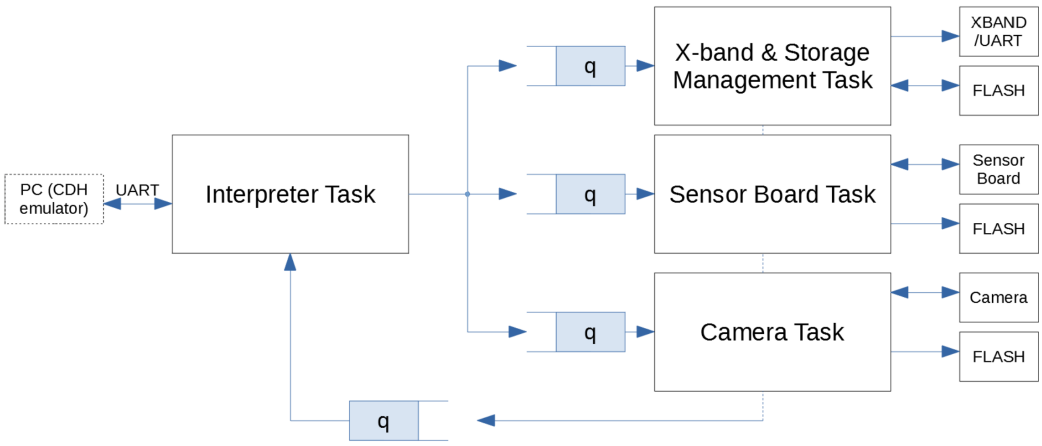
Odsječak koda 5.10: Program za računanje FFT spektra signala

5.6. Integracija s operacijskim sustavom FreeRTOS

Razvijena programska potpora za upravljanje senzorskim podsustavom integrirana je s cjelokupnom programskom potporom PDH računala. Programska potpora koristi operacijski sustav za rad u stvarnom vremenu FreeRTOS. Raspodjela poslova u FreeRTOS-u obavlja se korištenjem takozvanih zadataka (engl. *tasks*). Programska potpora PDH računala podijeljena je na 4 zadatka, koji su ovdje navedeni redom od zadatka s najvišim prioritetom do zadatka s najnižim prioritetom [8]:

1. *Interpreter Task*: komunicira s CDH računalom, odnosno osobnim računalom u demonstracijskoj inačici programa, i postavlja parametre za izvršavanje ostalih zadataka (*device tasks*),
2. *X-band and Storage Management Task*: upravlja komunikacijskim podsustavom i datotečnim sustavom za trajnu memoriju,
3. *Camera Task*: upravlja radom kamere,
4. *Sensor Board Task*: upravlja senzorskim podsustavom.

Zadaci međusobno komuniciraju putem redova poruka (engl. *message queue*). Zadatak *Interpreter* postavlja parametre za svaki od *device* zadataka (npr. ime datoteke u koju se podaci spremaju, duljinu ekspozicije kamere, itd.) te postavlja poruku s parametrima u red poruka odgovarajućeg zadatka. Kada su postavljeni parametri za sve zadatke, zadatak *Interpreter* spušta svoj prioritet na najniži u sustavu. To omogućuje aktivaciju ostalih zadataka, koji čitaju poruke iz pripadajućih redova i obavljaju svoje poslove. Nakon što neki od zadataka završi s poslom, šalje poruku o uspješnosti u red poruka zadatka *Interpreter*, a zatim biva blokiran čekajući na poruku iz (sada praznog) reda poruka. Kada svi zadaci obave svoj posao, ponovno se aktivira zadatak *Interpreter*



Slika 5.2: Zadaci programske potpore PDH računala [8]

i ciklus se ponavlja. Komunikacija između zadataka prikazana je slikom 5.2. Eventualni dijeljeni resursi ne štite se nikakvim posebnim mehanizmima jer je sustav zamišljen tako da se zadaci međusobno ne prekidaju.

U sklopu rada [8] razvijen je i datotečni sustav za trajnu *Flash* memoriju koja se nalazi na tiskanoj pločici PDH računala. Za rad s datotečnim sustavom razvijene su korisničke funkcije po uzoru na standard POSIX (*Portable Operating System Interface*). Funkcije omogućuju jednostavno pisanje i čitanje datoteka s *Flash* memorije, te se koriste u svim *device* zadacima.

U sklopu ovog rada razvijen je zadatak *Sensor Board Task*. Putem zadatka *Interpreter* korisnik može zadati imena datoteka u koje će se spremiti uzorci sa fotosenzora (u kompleksnom zapisu, kao što je opisano u potpoglavlju 5.5) i uzorci s temperaturnih senzora. Imena datoteka su cijeli brojevi u intervalu od 0 do 1023. Nakon što se parametri pročitaju iz reda poruka, pokreće se postupak prikupljanja podataka, kao što je prikazano u odsječku koda 5.11.

```
1  Sensor_Board sb;
2  SB_Init(&sb);
3  taskENTER_CRITICAL();
4  SB_Get_Temperature_Readings(&sb);
5  taskEXIT_CRITICAL();
6  SB_Start_ADC_Sampling(&sb);
7  SB_Get_Complex_Samples(&sb);
```

Odsječak koda 5.11: Prikupljanje podataka u sklopu zadatka za upravljanje senzorskim podsustavom

Dio programa koji čita podatke s temperaturnih senzora mora se štititi kao kritična sekcija, jer bi njegovo prekidanje od strane drugih zadataka moglo dovesti do grešaka pri spremanju podataka. Dio programa koji čita podatke s ADC-a ne mora se štititi jer prekidanje zadatka neće dovesti do prekida u prijenosu podataka do memorije, s obzirom da je prijenos ostvaren DMA sklopom, a ne programski.

Po završetku izvođenja ovog odsječka programskog koda, uzorcima fotosenzora može se pristupiti preko pokazivača `adc->complex_samples` u strukturi `sb`, a uzorcima temperaturnih senzora preko pokazivača `tmp_sensor->samples`, također u strukturi `sb`. Prikupljeni podaci zapisuju se u datotečni sustav u datotekama čija su imena zadana parametrima zadataka, korištenjem prethodno spomenutih POSIX funkcija. Za potrebe demonstracije dodan je i ispis prikupljenih podataka putem sučelja UART (*Universal Asynchronous Receiver Transmitter*).

5.6.1. Mapiranje dijelova memorije u RAM2

Memorija potrebna za izvršavanje zadatka Sensor Board Task iznosi približno 23.5 kB. Najveći dio te memorije zauzima polje s kompleksnim uzorcima fotosenzora (oko 16 kB) i polje s primljenim podacima s ADC-a (oko 7.5 kB). Prije integracije programske potpore za upravljanje senzorskim podsustavom, količina slobodne memorije u primarnoj RAM memoriji aplikacije PDH računala iznosila je svega 3.8 kB, što nije ni približno dovoljno za spremanje svih potrebnih podataka. No, mikrokontroler STM32L471VGT6 ima i sekundarnu RAM memoriju naziva RAM2 [10]. Ta je memorija kapaciteta 32 kB i mapirana je na adrese memorijskog prostora od 0x10000000 do 0x10008000.

Prema početnim postavkama programa povezivača (engl. *linker*) koje definira proizvođač mikrokontrolera u *linker* skripti `STM32L471VGTX_FLASH.ld`, sve statički alocirane varijable smještaju se u memorijsku sekciju `.bss` memorije RAM. U navedenoj *linker* skripti definirana je i memorija RAM2, međutim nije joj dodijeljena nijedna memorijska sekcija. Kako bi se kreirala memorijska sekcija u koju će se spremati podaci čije će memorijske adrese biti mapirane na adrese pridijeljene memoriji RAM2, potrebno je u *linker* skriptu dodati sljedeći odsječak koda:

```
1 SECTIONS
2 {
3     .ram2 :
4     {
5         KEEP(*(.ram2))
```

```
6       } >RAM2
7
8       /* ... ostatak koda linker skripte ... */
```

Odsječak koda 5.12: Smještanje memorijske sekcije `.ram2` u memoriju RAM2

Naredba `SECTIONS` definira kako se sekcije u ulaznim datotekama programa povezivača trebaju mapirati u izlaznu datoteku [5]. Linija 3 gornjeg odsječka definira izlaznu sekciju `.ram2`. Izraz na liniji 5 znači da se ulazna sekcija `.ram2` definirana u bilo kojoj ulaznoj datoteci (označeno *wildcard* simbolom `*`) treba mapirati na adresu izlazne sekcije `.ram2`. Naredba `KEEP()` govori programu povezivaču da ulazne sekcije `*.ram2` ne smije ukloniti tijekom *garbage collection* postupka. Izraz na liniji 6 mapira adresu izlazne sekcije `.ram2` u memoriju RAM2.

Preostaje još samo signalizirati programu povezivaču koje programske varijable treba smjestiti u sekciju `.ram2`. To se može postići dodjeljivanjem atributa `section` varijabli navođenjem izraza `__attribute__((section(".ram2")))` uz njenu deklaraciju. Radi bolje čitljivosti koda, definiran je makro izraz:

```
1  #define _SECTION_RAM2 __attribute__((section(".ram2")));
```

Odsječak koda 5.13: Makro izraz za dodjelu atributa `section` programskim varijablama

Makro izraz je zatim naveden uz deklaracije polja u kojima se spremaju kompleksni uzorci fotosenzora i podaci dohvaćeni s ADC-a. Raspored memorije nakon ove modifikacije prikazan je slikom 5.3. Iz slike je vidljivo da se podaci smješteni u poljima `samples_array` i `complex_samples` sada nalaze u memoriji RAM2 počevši od adrese 0x10000000.

RAM2	0x10000000		32 KB
.ram2	0x10000000	0x10000000	23.5 KB
samples_array	0x10000000	0x10000000	7.5 KB
complex_samples	0x10001e00	0x10001e00	16 KB
RAM	0x20000000		96 KB
.data	0x20000000	0x08012bb4	472 B
.bss	0x200001d8		90.24 KB
._user_heap_stack	0x20016ad0		1.5 KB

Slika 5.3: Raspored memorije nakon smještanja polja `complex_samples` i `samples_array` u RAM2

6. Zaključak

U ovom radu opisana je razvijena programska potpora za prikupljanje i obradu senzorskih podataka nanosatelita CubeSat u okviru projekta FERSAT. Dan je kratki opis projekta FERSAT, dosadašnjih aktivnosti na projektu i objavljenih radova na koje se ovaj rad nadovezuje. Opisana je arhitektura korisnog tereta FERSAT-a i postojeća programska potpora i sklopovlje korisnog tereta. Detaljno je proučeno SPI sučelje PDH računala koje se koristi za komunikaciju sa sklopovljem senzorskog podsustava i navedeni su dijelovi programske potpore koji omogućuju upravljačkim programima korištenje tog sučelja. Opisan je način na koji komponente senzorskog podsustava koriste SPI sučelje za komunikaciju s mikrokontrolerom. Razvijeni su upravljački programi za prikupljanje podataka sa senzora, kao i programska potpora za pripremu podataka za daljnju obradu, odnosno računanje spektra signala algoritmom FFT. Programska potpora za upravljanje senzorskim podsustavom integrirana je u programsku potporu PDH računala. Razvijen je zadatak u sklopu operacijskog sustava za rad u stvarnom vremenu FreeRTOS koji upravlja aktivnostima senzorskog podsustava i zapisuje prikupljene podatke u trajnu memoriju korištenjem datotečnog sustava, te je integriran u postojeću programsku potporu. Opisan je i način mapiranja dijelova memorije u sekundarnu RAM memoriju odabranog mikrokontrolera.

Programsku potporu razvijenu u sklopu ovog rada moguće je nadograditi na nekoliko načina. Primjerice, nisu korištene naprednije mogućnosti programiranja AD pretvornika ADS131M08. Programiranjem registara AD pretvornika mogla bi se smanjiti duljina rezultata pretvorbe s 24 na 16 bita, što bi ubrzalo prijenos podataka bez gubitka preciznosti koji bi bio značajan za ovu primjenu. Bilo bi vrijedno i istražiti mogućnost povećanja frekvencije uzorkovanja. Također, nije korištena mogućnost izračunavanja CRC zaštitnog koda kako bi se podaci zaštitili od grešaka tijekom prijenosa.

Na kraju, moglo bi biti od koristi istražiti mogućnost korištenja vanjske serijske RAM memorije kako bi se riješio problem velike zauzetosti radne memorije mikrokontrolera.

LITERATURA

[1] *ADT7301 Temperature Sensor Datasheet*. Analog Devices, 2011. Rev. B.

[2] *CMSIS DSP Software Library Version 1.10.0 Documentation*. Arm, 2022. URL `https://arm-software.github.io/CMSIS_5/DSP/html/index.html`. Preuzeto: lipanj 2022.

[3] Filip Bogdanović. Sklopovi za mjerenje ultra-ljubičastog zračenja na satelitima. Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2020.

[4] Fabio Falchi, Pierantonio Cinzano, Christopher D. Elvidge, David M. Keith, i Abraham Haim. Limiting the impact of light pollution on human health, environment and stellar visibility. *Journal of Environmental Management*, 92(10), 2011.

[5] GNU Contributors. GNU Linker ld version 2.38 documentation. URL `https://sourceware.org/binutils/docs/ld/index.html`. Preuzeto: lipanj 2022.

[6] Filip Jurić. Upravljačko sklopovlje za prikupljanje i obradu senzorskih signala na CubeSat nanosatelitu. Završni rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021.

[7] Tomislav Petković. Osnove obradbe signala: Spektar. Materijali za predavanja, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021.

[8] Goran Petrak. Programska potpora ugradbenog računalnog sustava za udaljeno prikupljanje fotografija putem satelita. Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021.

[9] *RM0351 Reference manual STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Axxx advanced Arm®-based 32-bit MCUs*. ST Microelectronics, 2021. Rev. 9.

[10] *RM0090 Reference manual: STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs*. ST Microelectronics, 2021. Rev. 19.

[11] *UM7125 Description of STM32F4 HAL and low-layer drivers*. ST Microelectronics, 2021. Rev. 7.

[12] Kristijan Stepančić. Satelitska mjerenja koncentracije ozona. Završni rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2020.

[13] *LMP7721 3 Femtoampere Input Bias Current Precision Amplifier Datasheet*. Texas Instruments, 2008.

[14] *ADS131M08 8-Channel, Simultaneously-Sampling, 24-Bit, Delta-Sigma ADC*. Texas Instruments, 2021. Rev. B.

[15] Jakov Tutavac. Spectral unmixing of incoherent light for the analysis of light pollution. Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2020.

[16] Wikipedia. Serial peripheral interface, 2022. URL `https://en.wikipedia.org/wiki/Serial_Peripheral_Interface`. Preuzeto: svibanj 2022.

[17] Zavod za komunikacijske i svemirske tehnologije Fakulteta elektrotehnike i računarstva. FERSAT - opis projekta, 2022. URL `https://www.fer.unizg.hr/zkist/FERSAT/projekt`. Preuzeto: lipanj 2022.

Programska potpora za prikupljanje i obradu senzorskih podataka na CubeSat nanosatelitu

Sažetak

U ovom radu razvijena je programska potpora za upravljanje sustavom za prikupljanje podataka o svjetlosnom onečišćenju Zemlje, namijenjena za korištenje na nanosatelitu FERSAT. Dan je pregled aktivnosti u sklopu projekta FERSAT koje su prethodile radu. Opisano je postojeće sklopovlje i programska potpora. Opisano je komunikacijsko sučelje SPI (*Serial Peripheral Interface*) pomoću kojeg komponente senzorskog sustava komuniciraju s mikrokontrolerom. Razvijeni su upravljački programi za prikupljanje senzorskih podataka i programska potpora za pripremu podataka za daljnju obradu te računanje spektra signala korištenjem algoritma FFT (*Fast Fourier Transform*). Razvijena programska potpora integrirana je u postojeću programsku potporu za PDH (*Payload Data Handler*) računalo koja je izvedena pomoću operacijskog sustava FreeRTOS.

Ključne riječi: ugradbeni računalni sustavi, CubeSat, FERSAT, STM32, upravljački program, digitalna obrada signala, analogno-digitalna pretvorba, FreeRTOS

Software for CubeSat Nanosatellite Sensor Data Acquisition and Processing

Abstract

This thesis describes the development of software for acquisition and processing of sensor data from light pollution sensors on a CubeSat nanosatellite. A brief overview of the FERSAT project is provided, along with a description of existing software and hardware. The SPI (Serial Peripheral Interface) communication interface used for data transfer between sensor subsystem components is described in detail. Drivers for an analog-to-digital converter and a temperature sensor are developed, as well as software support for preparing the acquired data for further processing and for calculating the signal spectrum of acquired data using FFT (Fast Fourier Transform). All developed software components are integrated into an existing FreeRTOS application that executes on the PDH (Payload Data Handler) computer.

Keywords: embedded systems, CubeSat, FERSAT, STM32, driver, digital signal processing, analog-to-digital conversion, FreeRTOS