SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2021-72

Programska potpora za upravljanje kamerom na CubeSat nanosatelitu

Nikola Gudan

Zagreb, srpanj 2022.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Hvala Karli Salamun na savjetima i pomoći pri izradi ovog rada.
Hannon allen. Savo ’lass a lalaith.

SADRŽAJ

1. Uvod

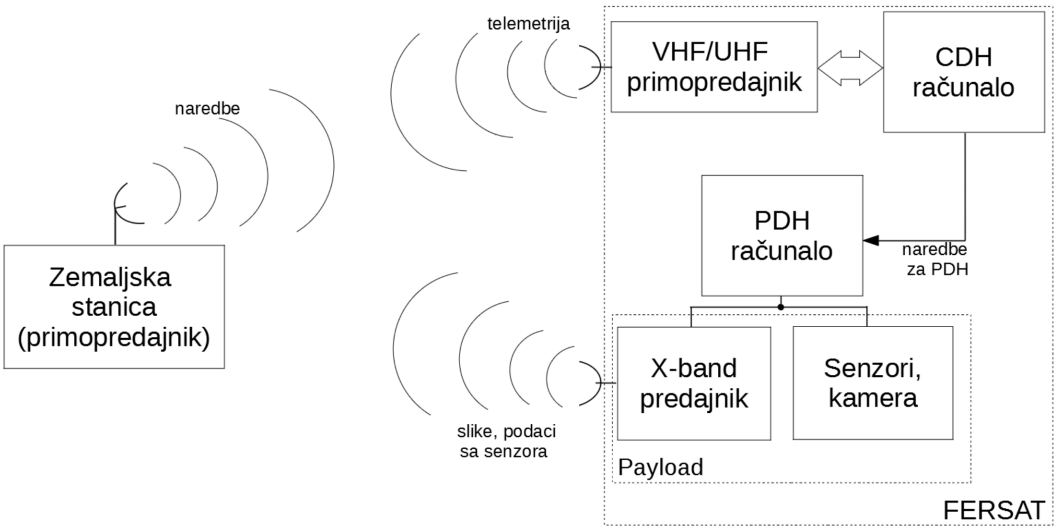
Ovaj završni projekt se izvodi u sklopu projekta FERSAT, koji se od 2018. godine provodi na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu [10]. Cilj projekta je izrada, lansiranje i korištenje jednog nanosatelita u CubeSat formatu, dimenzija 10 cm x 10 cm x 10 cm, volumena jedne litre i težine ne veće od 4/3 kg. Navedene dimenzije satelita odgovaraju formatu CubeSat 1U. Planirana visina orbite satelita je između 500 i 600 km, a očekivano trajanje misije je 3 godine. Korisni teret satelita (engl. *payload*) se sastoji od 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.

Kako bi se moglo upravljati radom korisnog tereta, na satelit će biti ugrađeno PDH (engl. *Payload Data Handler*) računalo, čija će zadaća biti prikupljanje podataka s kamere i senzorskog podsustava, pohranjivanje prikupljenih podataka u trajnu memoriju (engl. *non-volatile memory*), te slanje podataka na Zemlju pomoću komunikacijskog sustava. Izabrani mikrokontroler za ulogu PDH računala je STM32L471VGT6 proizvođača ST Microelectronics.

Ostalim podsustavima, koji nisu direktno vezani uz koristan teret, upravlja CDH (engl. *Command and Data Handler*) računalo. CDH računalo može upravljati položajem i orijentacijom satelita, slanjem telemetrijskih podataka na Zemlju, a također upravlja i napajanjem korisnog tereta i šalje naredbe PDH računalu preko CAN (engl. *Controller Area Network*) sučelja. U trenutku pisanja ove dokumentacije, konkretno CDH računalo još nije odabrano.

Slika 1.1 prikazuje blok dijagram cijelog sustava. U okviru ovog projekta razvijena je programska potpora PDH računala za upravljanje kamerom i *flash* memorijom.



Slika 1.1: Blok dijagram FERSAT-a i komunikacija sa zemaljskom postajom [3]

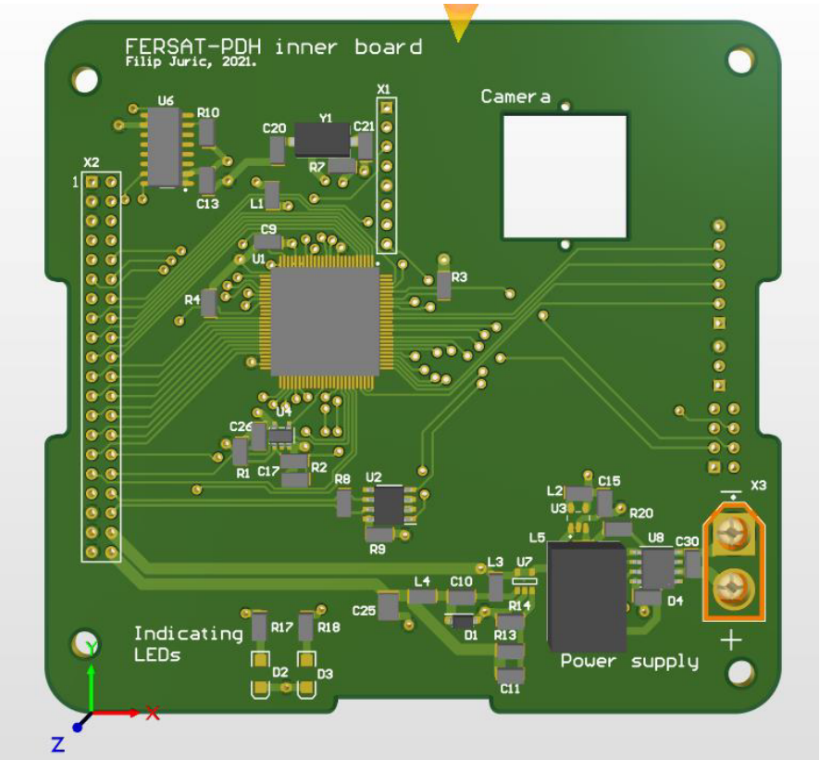
Sustav za upravljanje kamerom se sastoji od Arducam Mini 5MP Plus kamere. Upravljanje kamerom se sastoji od konfiguracije kamere i samog korištenja kamere, odnosno slikanja i spremanja slike. Konfiguracija kamere je nužna kako bi se ispravno podesili parametri trajanja ekspozicije, pojačanje i formata u kojem se slika želi spremiti.

2. Arhitektura sustava

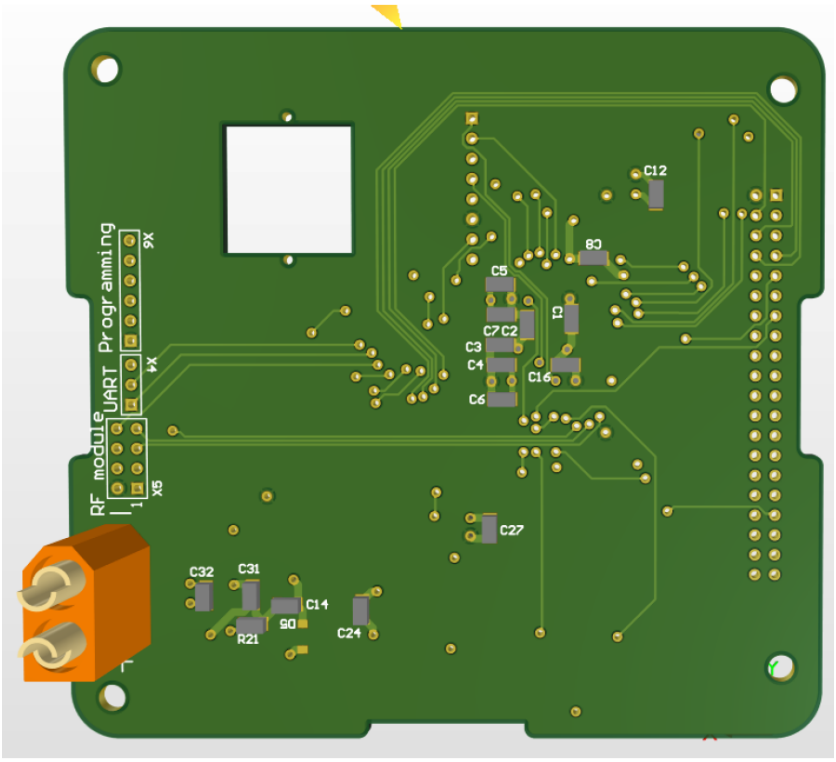
Slanjem određenog signala na sklop za kameru može se uslikati slika, a nakon slikanja slika se spremi na vlastiti međuspremnik kamere. Cilj je spremljenu kameru pročitati iz međuspremnika kamere i spremiti ju na *flash* memoriju koja se nalazi na pločici PDH-a, gdje može biti spremljena dok se ne zatraži slanje slike preko X-band predajnika na Zemlju.

Flash memorija, osim što služi za pohranu slike, služi i za pohranu podataka s drugih senzora. Ona prima i šalje podatke ovisno o poslanoj naredbi putem SPI komunikacije s mikrokontrolerom.

Upravljačko sklopovlje PDH računala se sastoji od STM32F471VGT6 mikrokontrolera, spomenute vanjske *flash* memorije, konektora za povezivanje s ostalim dijelovima sustava (uključujući i konektor za povezivanje s kamerom), sustava za napajanje, upravljačkog sklopovlja za CAN komunikaciju i sklopa za kontrolu izvođenja programa (engl. *watchdog*) [1]. Izgled tiskane pločice upravljačkog sklopovlja PDH računala prikazan je na slikama 2.1 i 2.2. Konektor X1 služi za povezivanje sustava s kamerom.



Slika 2.1: Prikaz gornje strane tiskane pločice upravljačkog sklopovlja PDH računala [1]



Slika 2.2: Prikaz donje strane tiskane pločice upravljačkog sklopovlja PDH računala [1]

Programska podrška za PDH računalo već je razvijena [3]. Međutim, u međuvre-

menu je došlo do promjene izbora mikrokontrolera PDH računala, te je stoga postojeću programsku podršku bilo potrebno prilagoditi trenutačnom sklopovlju.

S obzirom na prirodu ovog završnog projekta, gdje je naglasak bio na prilagođavanju postojeće programske podrške, u ovom radu će biti raspravljeni izazovi i izmjene do kojih je došlo tijekom prilagođavanja programske podrške. U poglavlju 2 dan je detaljan opis I^2C (engl. *Inter-Integrated Circuit*) komunikacije, te su istaknute razlike između starog i novog sklopovlja koje su bile ključne za prilagođavanje programske podrške. Na isti način opisani su SPI (engl. *Serial Peripheral Interface*) komunikacija i DMA (engl. *Direct Memory Access*) prijenos u poglavlju x. Detaljan pregled razvijene programske podrške dan je u poglavlju y, gdje je opisana integracija programske podrške za kameru i *flash* memorija u FreeRTOS operacijski sustav za rad u stvarnom vremenu.

3. Sučelja za komunikaciju

3.1. I^2C sučelje

Za konfiguraciju kamere Arducam 5MP Mini Plus PDH računalo koristi I^2C komunikaciju. U nastavku slijedi općeniti opis I^2C protokola kao i razlike između I^2C periferijski sklopova na prethodno korištenom (STM32F407VGT6) i trenutačnom (STM32L471VGT6) mikrokontroleru.

3.1.1. I^2C protokol

I^2C je jednostavna dvosmjerna sinkrona serijska sabirnica razvijena od strane *Philips Semiconductors* (sada *NXP Semiconductors*) 1982. godine [8]. Koristi dvije linije:

- serijska podatkovna linija (SDA, *Serial Data Line*),
- serijska taktna linija (SCL, *Serial Clock Line*).

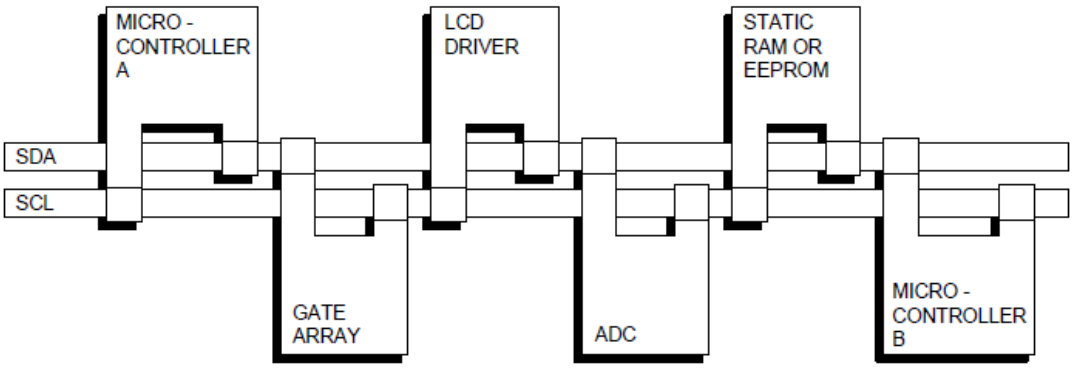
Obje linije su pritegnute na visoku logičku razinu preko *pull-up* otpornika. Moguće brzine prijenosa su:

- do 100 kbit/s u *Standard-mode* načinu rada,
- do 400 kbit/s u *Fast-mode* načinu rada,
- do 1 Mbit/s u *Fast-mode Plus* načinu rada,
- do 3.4 Mbit/s u *High-speed* načinu rada.

Navedene brzine se koriste kod dvosmjernog prijenosa, a moguća je i brzina do 5 Mbit/s u jednosmjernom prijenosu. Više uređaja se može spojiti na jednu sabirnicu, a svaki uređaj je prepoznatljiv po svojoj jedinstvenoj adresi i može se ponašati kao prijamnik ili odašiljač, ovisno o funkciji uređaja [2]. Protokol najčešće, a tako i u ovom slučaju, koristi 7-bitno adresiranje, a moguće je i korištenje 10-bitnog adresiranja. Osim konfiguracije prijamnika i odašiljača, uređaj također može biti *master* uređaj ili *slave* uređaj tijekom prijenosa podataka. *Master* uređaj je uređaj koji inicijalizira

prijenos podataka na sabirnici i generira signal takta kako bi omogućio prijenos. U tom trenutku, bilo koji uređaj koji je adresiran smatra se *slave* uređajem.

Na I^2C sabirnicu se također može spojiti više *master* uređaja, a primjer jednog takvog spoja sa dva mikrokontrolera dan je na slici 3.1. Prijenos podataka bi možda



Slika 3.1: Primjer I^2C sabirnice sa spojena dva mikrokontrolera [2]

mogao izgledati ovako:

1. Mikrokontroler A želi poslati podatke mikrokontroleru B:
    - mikrokontroler A (*master* uređaj) adresira mikrokontroler B (*slave* uređaj)
    - mikrokontroler A (*master*-odašiljač) šalje podatke mikrokontroleru B (*slave* uređaj-prijamnik)
    - mikrokontroler A prekida prijenos
2. Mikrokontroler A želi primiti podatke sa mikrokontrolera B:
    - mikrokontroler A (*master* uređaj) adresira mikrokontroler B (*slave*)
    - mikrokontroler A (*master*-prijamnik) prima podtke sa mikrokontrolera B (*slave*-odašiljač)
    - mikroknotroler A prekida prijenos.

U svakom od navedenih slučajeva mikrokontroler A je generirao takt i prekidao prijenos. Kod prijenosa podataka na I^2C sabirnici, *master* uređaj uvijek generira signal takta. U ovom radu korišten je samo jedan mikrokontroler, odnosno *master* uređaj, pa će se u daljnjem tekstu podrazumijevati samo taj slučaj.

Opis komunikacije i vremenski dijagram

I^2C komunikacija započinje sa *start* simbolom i završava sa *stop* simbolom. Komunikacijom se može čitati ili pisati ovisno o R/W bitu u adresi. Struktura adresiranja kod

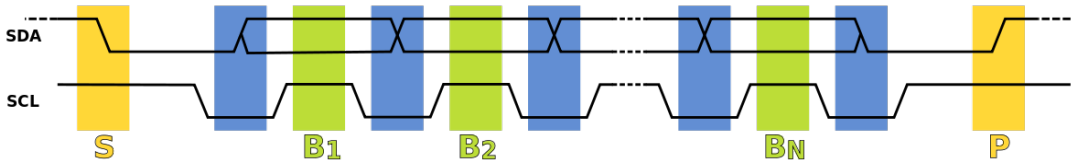
7-bitne adrese je prikazana u tablici 3.1.

Tablica 3.1: Struktura adresiranja kod 7-bitne adrese [8]

	Adresno polje							R\W
Pozicija bita u bajtu	7	6	5	4	3	2	1	0
Značenje	MSB						LSB	1=READ, 0=WRITE

Iz tablice je vidljivo da najmanje značajan bit označava želi li se podatak čitati ili pisati.

Imajući na umu izgled adresnog bajta, vremenski dijagram tipčne I^2C komunikacije prikazan je na slici 3.2.



Slika 3.2: Vremenski dijagram I^2C komunikacije [8]

- Prijenos podataka se inicijalizira *start* uvjetom (S) tako da SDA linija prijeđe u nisku logičku razinu dok SCL linija ostaje u visokoj logičkoj razini.
- (Plavo područje) SCL prelazi u nisku logičku razinu i SDA postavlja prvi podatkovni bit dok je SCL u niskoj logičkoj razini.
- (Zeleno područje) Podaci se primaju dok SCL poraste za prvi bit (B_1). Kako bi podaci bili valjani, SDA se ne smije promijeniti između rastućeg brida SCL-a i sljedećeg padajućeg brida.
- Postupak se ponavlja, SDA se postavlja dok je SCL u niskoj razini, a podaci se čitaju dok je SCL u visokoj razini (B_2 do B_n).
- Nakon posljednjeg bita slijedi taktni impuls, tijekom kojeg SDA prelazi u nisku razinu pripremajući se za *stop* uvjet.
- Signalizira se *stop* uvjet kada SCL prijeđe u visoku logički razinu, nakon čega slijedi prelazak u visoku logičku razinu SDA signala.
- (Plavo područje) SCL prelazi u nisku logičku razinu i SDA postavlja

Start i *stop* uvjete uvijek generira *master* uređaj. Nakon svakog bajta prijamnik šalje odašiljaču ACK bit kojim se signalizira uspješno primanje podatka, odnosno NACK bit kojim se signalizira neuspješno primanje podatka. ACK i NACK bitovi se nazivaju signalom potvrde i definiraju se na sljedeći način: odašiljač otpušta SDA liniju tijekom potvrdnog takta kako bi prijamnik mogao spustiti SDA na nisku razinu na kojoj i ostaje tijekom visoke razine takta. Ako SDA ostaje u visokoj razini tijekom devete periode takta, to predstavlja NACK (engl. *Not Acknowledge*) signal, a suprotan slučaj predstavlja ACK (engl. *Acknowledge*) signal. Ako je došlo do NACK signala, *master* uređaj može generirati *stop* uvjet kako bi prekinuo prijenos ili može ponovno generirati *start* uvjet kako bi započeo novi prijenos. Vremenski dijagram cijele komunikacije s potvrdnim signalima prikazan je na slici 3.3.

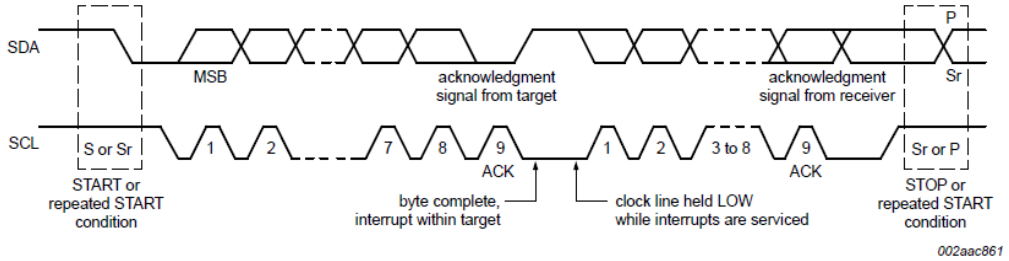


Figure 6. Data transfer on the I²C-bus

Slika 3.3: Prijenos podataka na I^2C sabirnici [2]

3.1.2. Razlika I^2C periferije na STM32L471VGT6 i STM32F407VGT6 mikrokontrolerima

Tijekom prijenosa koda sa starog mikrokontrolera na novi, primjećeno je da postoji razlika između struktura I^2C periferija. Točnije, postoji razlika između registarskih mapa na dvama periferijama, koje su vidljive usporedbom tablica 3.2 i 3.3 (*napomena*: u tablicama nisu prikazani svi registri, jer se neki registri niti ne koriste, ili se koriste samo tijekom konfiguracije periferija. Za puni prikaz tablica treba provijeriti dokumentacije mikrokontrolera [5], [6]).

Tablica 3.2: Registarska mapa I^2C periferije STM32L471VGT6 mikrokontrolera [6]

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0	I2C_CR1	Res.								PECEN	ALERTEN	SMBDEN	SMBHEN	GCEN	WUPEN	NOSTRETCH	SBC	RXDMAEN	TXDMAEN	Res.	ANFOFF	DNF[3:0]				ERRIE	TCIE	STOPIE	NACKIE	ADDRIE	RXIE	TXIE	PE
	Reset value									0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
0x4	I2C_CR2	Res.					PECBYTE	AUTOEND	RELOAD	NBYTES[7:0]								NACK	STOP	START	HEAD10R	ADD10	RD_WRN	SADD[9:0]									
	Reset value						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	I2C_ISR	Res.								ADDCODE[6:0]							DIR	BUSY	Res.	ALERT	TIMEOUT	PECERR	OVR	ARLO	BERR	TCR	TC	STOPF	NACKF	ADDR	RXNE	TXIS	TXE
	Reset value									0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	1	0
0x1C	I2C_ICR	Res.																		ALERTCF	TIMOUTCF	PECCF	OVRCF	ARLOCF	BERRCF	Res.		STOPCF	NACKCF	ADDRCF	Res.		
	Reset value																			0	0	0	0	0	0			0	0	0			
0x24	I2C_RXDR	Res.																								RXDATA[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x28	I2C_TXDR	Res.																								TXDATA[7:0]							
	Reset value																									0	0	0	0	0	0	0	0

Tablica 3.3: Registarska mapa I^2C periferije STM32F407VGT6 mikrokontrolera [5]

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0	I2C_CR1	Res.																SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NOSTRETCH	ENGC	ENPEC	ENARP	SMBTYPE	Res.	SMBUS	PE
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0		0	0
0x4	I2C_CR2	Res.																			LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Res.		FREQ[5:0]					
	Reset value																				0	0	0	0	0			0	0	0	0	0	0
0x10	I2C_DR	Res.																								DR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	I2C_SR1	Res.																SMBALERT	TIMEOUT	Res.	PECERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
	Reset value																	0	0		0	0	0	0	0	0	0		0	0	0	0	0
0x18	I2C_SR2	Res.																PEC[7:0]								DUALF	SMBHOST	SMBDEFAUL	GENCALL	Res.	TRA	BUSY	MSL
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0

Vidljiva je razlika između količine registara, raspodjele i značenja njihovih bitova, kao i njihovih imena, što implicira različite funkcionalnosti pojedinih registara. Tako, npr. I^2C periferija kod STM32F407VGT6 sadržava 2 status registra: I2C_SR1 i I2C_SR2, dok kod STM32L471VGT6 postoji samo jedan status registar I2C_ISR. Ta razlika je bitna zato što se tijekom prijenosa podataka na I^2C sabirnici trebaju

provjeravati razne zastavice koje se mijenjaju tijekom komunikacije, kao što je npr. zastavica za prazni odašiljački registar (STM32F407VGT6: registar I2C_SR1 bit 7, STM32L471VGT6: bit 0), zastavica za puni prijamnički registar (STM32F407VGT6: registar I2C_SR1, bit 6, STM32L471VGT6: bit 2), zastavica za završetak prijenosa (STM32F407VGT6: ne postoji, STM32L471VGT6: bit 6) itd.

Vidljivo je također da kod STM32L471VGT6 postoji zastavica ADDR, koja inače kod STM32F407VGT6 signalizira uspješan primitak adrese uređaja mete, a kod STM32L471VGT6 ta zastavica se koristi isključivo u *slave* načinu rada, tako da ta zastavica nije bitna za ovaj projekt. Kako onda mikrokontroler zna da je poslana adresa točna? Naime, STM32L471VGT6 ima poseban registar za pohranu adrese uređaja mete, pa kada mikrokontroler pošalje *start* uvjet on automatski nakon završetka *start* uvjeta pošalje i adresu uređaja mete, a uspješan primitak adrese signalizira zastavica I2C_ISR_TXIS kod slanja podataka, odnosno I2C_ISR_RXNE zastavica kod primitka podataka.

Vidljive su i razlike u raspodjeli zastavica u registrima, kao i razlike u funkcijama koje zastavice signaliziraju. Inače bi te razlike stvarale probleme kod konfiguracije I^2C periferije, no, kako je tu brigu riješio kod generator ugrađen u STM32CubeIDE razvojno okruženje, nije bila posvećena pažnja tim razlikama. Način implementacije spomenutih razlika u programsku podršku opisan je u poglavlju z.

3.2. SPI sučelje

Protokol SPI se koristi za prijenos podataka između *flash* memorije i mikrokontrolera, odnosno međuspremnika kamere. Kako bi se oslobodili resursi na mikrokontroleru, za prijenos podataka između kamere i *flash* memorije se, u kombinaciji sa SPI protokolom, koristi i DMA prijenos. U ovom poglavlju bit će opisan SPI protokol i DMA prijenos i bit će istaknute razlike i problemi kod prilagođavanja programske podrške za STM32L471VGT6 mikrokontroler.

3.2.1. SPI protokol

SPI je sinkrono serijsko komunikacijsko sučelje koje se koristi za komunikaciju na kratkim udaljenostima, pretežito u ugradbenim računalnim sustavima [9].

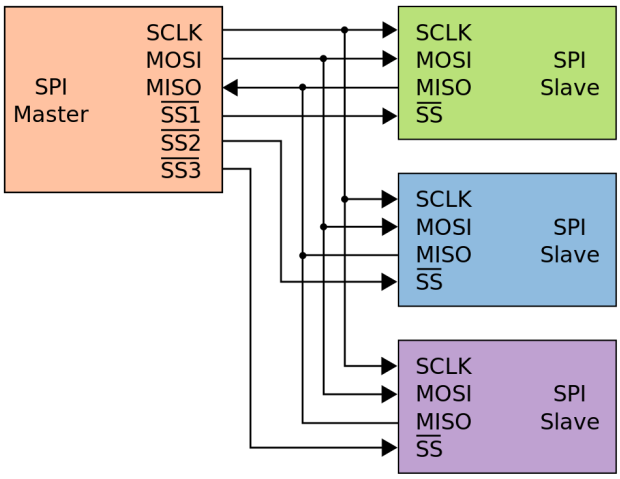
SPI uređaji komuniciraju u *full-duplex* načinu rada koristeći *master-slave* arhitekturu, obično sa jednim *master* uređajem. Više *slave* uređaja može biti spojeno na jedan upravljač tako da se aktivira određeni *chip select* signal za pojedini uređaj.

Opis sučelja

SPI sabirnica se sastoji od četiri signala:

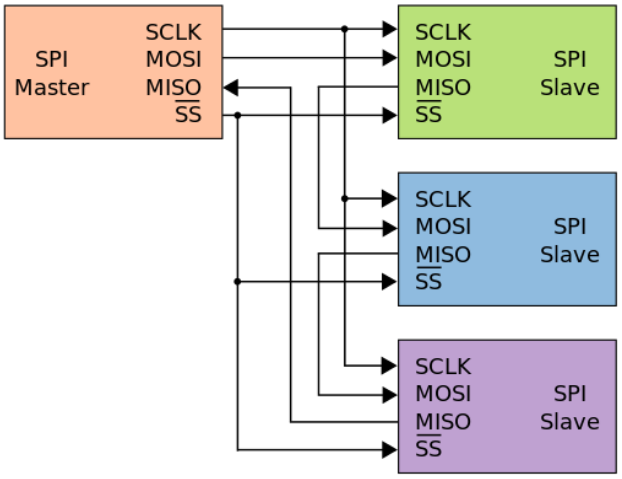
- SCLK: Serijski takt (izvor je *master* uređaj),
- MOSI: *Master Output Slave Input* (izvor podataka iz *master* uređaja),
- MISO: *Master Input Slave Output* (izvor podataka iz *slave* uređaja),
- CS/SS: *Chip/Slave Select* (aktivan nisko, signal iz *master* uređaja, označava da se prenose podaci).

MOSI na *master* uređaju se spaja na MOSI na *slave* uređaju, dok se MISO na *master* uređaju se spaja na MISO na *slave* uređaju. CS/SS se koristi za pokretanje komunikacije između *slave* i *master* uređaja. Za svaki *slave* uređaj postoji zaseban CS/SS priključak na *master* uređaju. Takav način spajanja se naziva neovisni *slave* uređaj. Primjer spajanja tri *slave* uređaja na jedan *master* uređaj u konfiguraciji neovisnog *slave* uređaja prikazan je na slici 3.4.



Slika 3.4: Spoj tri *slave* uređaja na jedan *master* uređaj u konfiguraciji neovisnog *slave* uređaja. Vidljivo je da *master* uređaj ima tri SS priključka, a svaki odgovara jednom *slave* uređaju, dok se SCLK, MOSI i MISO linije međusobno dijele između *slave* uređaja [9]

Moguće je još spojiti uređaje u konfiguraciju ulančavanog *slave* uređaja. U toj konfiguraciji *slave* uređaji dijele isti CS/SS, a ulančavanjem preko MISO/MOSI linija podaci se prenose prema načelu posmačnog registra, koji je objašnjen u sljedećem potpoglavlju. Prikaz spajanja tri *slave* uređaja se nalazi na slici 3.5.



Slika 3.5: Spoj tri *slave* uređaja na jedan *master* uređaj u konfiguraciji ulančavanog *slave* uređaja [9]

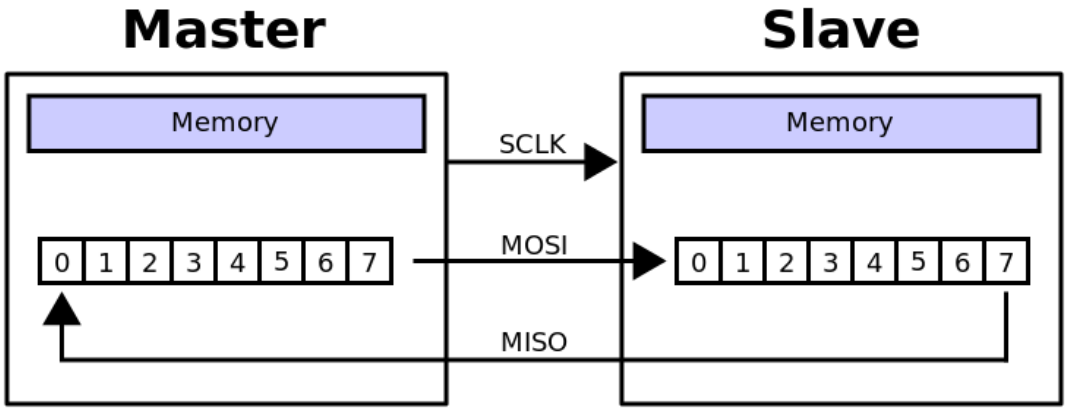
Način rada

SPI sabirnica radi s jednim *master* uređajem i jednim ili više *slave* uređaja. Ako se koristi jedan *slave* uređaj, onda CS signal može biti postavljen u nisku logičku razinu, ako *slave* uređaj to dopušta. Neki *slave* uređaji zahtijevaju padajući brid CS signala kako bi započela komunikacija. Ako se koristi više *slave* uređaja potreban je zaseban CS signal *master* uređaja za svaki *slave* uređaj.

Prijenos podataka Za početak komunikacije *master* uređaj konfigurira takt koristeći frekvenciju koju podržava *slave* uređaj, obično do nekoliko MHz. *Master* uređaj zatim odabire *slave* uređaj postavljanjem CS linije u nisko logičko stanje. Ako je potreban period čekanja, npr. za AD (analogno-digitalnu) pretvorbu, *master* uređaj mora pričekati minimalno taj period vremena prije puštanja takta.

Tijekom svakog perioda takta obavlja se prijenos podataka u *full-duplex* načinu rada. To znači da *master* uređaj pošalje jedan bit na MOSI liniju, koji *slave* uređaj pročita, dok u isto vrijeme *slave* uređaj šalje jedan bit na MISO liniju, koji *master* uređaj pročita. Takva sekvenca se održava čak i kada se izvodi jednosmjerni prijenos podataka.

Prijenosi podataka uključuju dva posmačna registra zadane veličine, npr. 8 bitova, jedan u *master* i drugi u *slave* uređaju. Registri su spojeni u topologiji virtualnog prstena (slika 3.6).

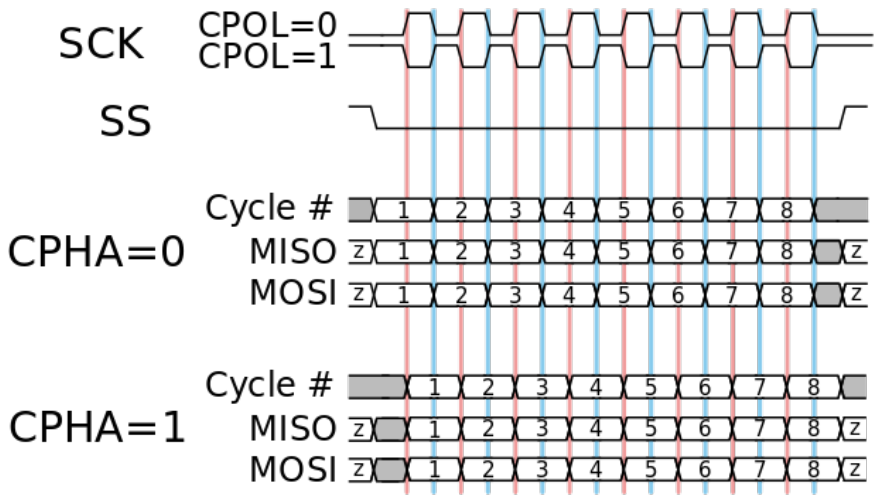


Slika 3.6: Tipičan spoj dvaju posmačna registra koji formiraju kružni međuspremnik [9]

Podaci se obično pomiču tako da se prvo pomakne najznačajniji bit. Na brid takta, *master* i *slave* uređaj pomaknu bit i pošalju ga na prijenosnu liniju. Na sljedeći brid takta, na svakom prijamniku bit se uzorkuje s prijenosne linije i postavlja se kao novi najmanje značajni bit u posmačnom registru. *Master* i *slave* uređaji u potpunosti razmjene podatke u registrima nakon što se svi bitovi u registrima prebace. Ako je potrebno razmijeniti još podataka, posmačni registri se ponovno napune te se postupak ponavlja, a prijenos se može obavljati za bilo koji broj perioda takta. Kada je prijenos dovršen, *master* uređaj prestaje davati takt i obično isključi CS signal, odnosno postavi ga na visoku razinu.

Prijenos se obično obavlja u riječima širine 8 bitova, no moguća je i širina riječi od 16 bita, ili čak 12 bitova, koji se koristi za digitalno-analogne i analogno-digitalne pretvornike.

Polaritet takta i faza Osim što mora podesiti frekvenciju takta, *master* uređaj mora isto tako podesiti polaritet takta (CPOL, engl. *Clock Polarity*) i fazu (CPHA, engl. *Clock Phase*) ovisno o podacima. Vremenski dijagram je prikazan na slici 3.7.



Slika 3.7: Vremenski dijagram koji pokazuje polaritet takta i fazu. Crvene linije označuju vodeće bridove, a plave linije označavaju prateće bridove [9].

CPOL određuje polaritet kanala. Polaritet može biti invertiran jednostavnim inverterom.

- Ako je CPOL = 0, onda takt miruje u niskom logičkom stanju, a svaki period se sastoji od impulsa visokog logičkog stanja. To znači da je vodeći brid rastući brid, a prateći brid padajući brid.
- Ako je CPOL = 1, onda takt miruje u visokom logičkom stanju, a svaki period se sastoji od impulsa niskog logičkog stanja. To znači da je vodeći brid padajući brid, a prateći brid rastući brid.

CPHA određuje fazu podatkovnih bitova u odnosu na takt.

- Ako je CPHA = 0, strana koja šalje podatke mijenja podatak na prateći brid prethodnog perioda takta, dok strana koja prima podetke prihvaća podatak na (ili ubrzo nakon) vodeći brid perioda takta. Izlazna strana zadržava valjani podatak sve do pojave pratećeg brida trenutnog perioda takta.
- Ako je CPHA = 1, strana koja šalje podatke mijenja podatak na vodeći brid trenutnog perioda takta, dok strana koja prima podatke prihvaća podatak na (ili ubrzo nakon) pratećeg brida perioda takta. Izlazna strana zadržava valjani podatak do pojave vodećeg brida sljedećeg perioda takta. Na zadnji period, *slave* uređaj zadržava valjani podatak na MISO liniji sve dok *slave* uređaj ne bude deselektiran.

MOSI i MISO signali su obično stabilni za vrijeme pola perioda takta, sve do sljedeće promjene takta. SPI *master* i *slave* uređaji mogu uzorkovati podatke u bilo kojem

vremenu unutar te polovice periode takta.

Kombinacije različitih konfiguracija CPOL i CPHA bitova predstavljaju načine rada. Konvecija je da CPOL predstavlja viši bit, dok CPHA predstavlja niži bit. Načini rada kod ARM-ovih mikrokontrolera su prikazani u tablici 3.4.

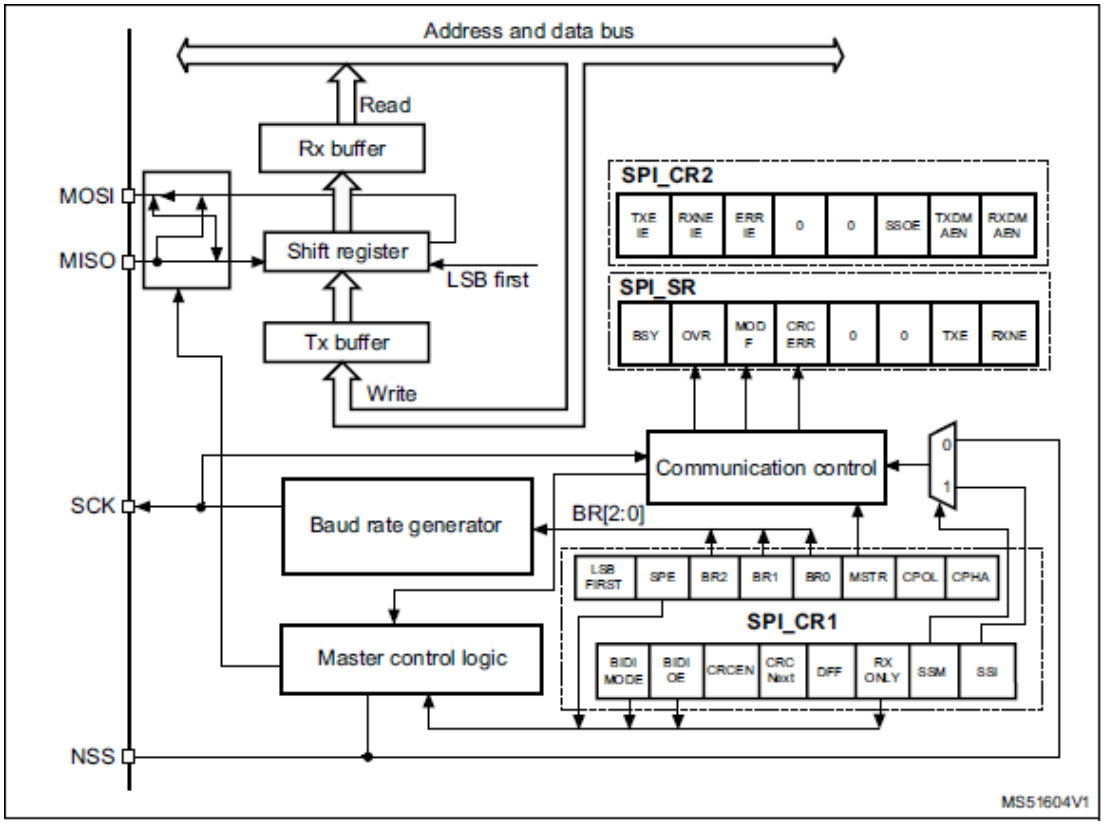
Tablica 3.4: SPI načini rada kod ARM-ovih mikrokontrolera [9]

SPI način rada	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

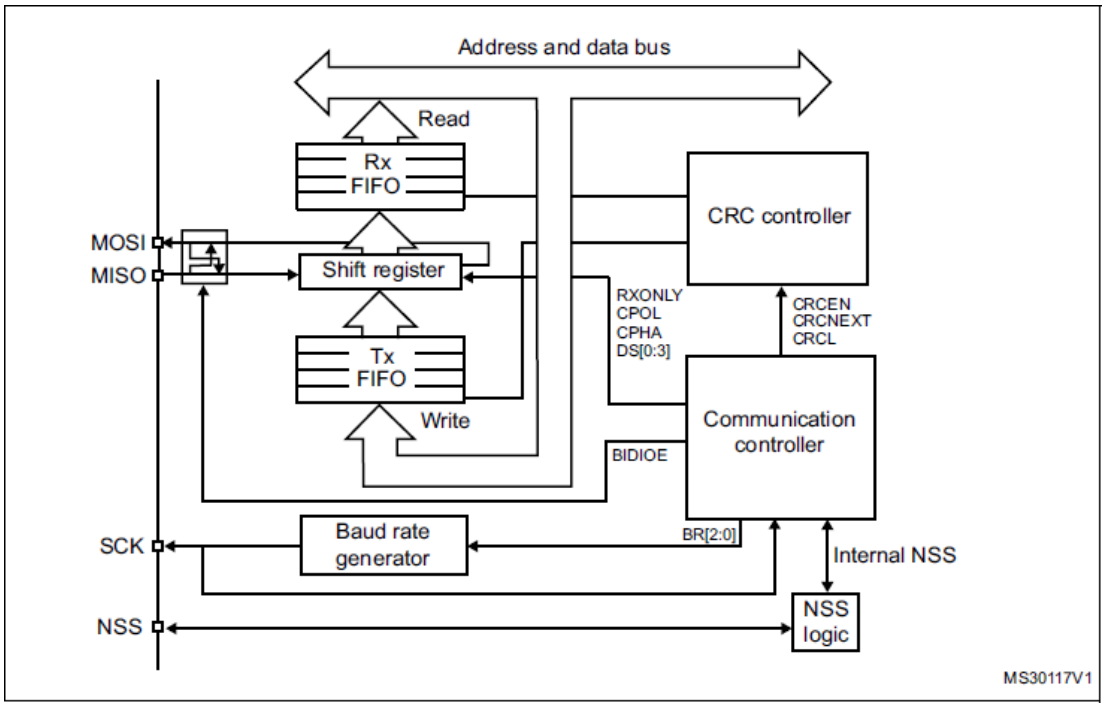
3.2.2. Prilagodba programske podrške sa STM32F407VGT6 mikrokontrolera na STM32L471VGT6 mikrokontroler

Što se tiče programske podrške za SPI komunikaciju između mikrokontrolera, *flash* memorije i kamere, nije došlo do nikakvih poteškoća kod prijenosa programske podrške s prethodno korištenog mikrokontrolera na trenutni. Pogledom na blok dijagrame SPI periferije (slike 3.8 i 3.9) vidljiva je velika sličnost između mikrokontrolera. Razlike između kontrolnih registara nisu problem, s obzirom na to da njih podešava generator koda, dok se razlike u statusnim registrima mogu zanemariti radi korištenja definicija registara i zastavica u programskoj podršci koju je pružao proizvođač mikrokontrolera.

Došlo je, međutim, do poteškoća kod prijenosa programske podrške za DMA prijenos, koje će biti objašnjene u sljedećem poglavlju.



Slika 3.8: SPI blok dijagram mikrokontrolera STM32407VGT6 [5, str. 876]



Slika 3.9: SPI blok dijagram mikrokontrolera STM32L471VGT6 [6, str. 1451]

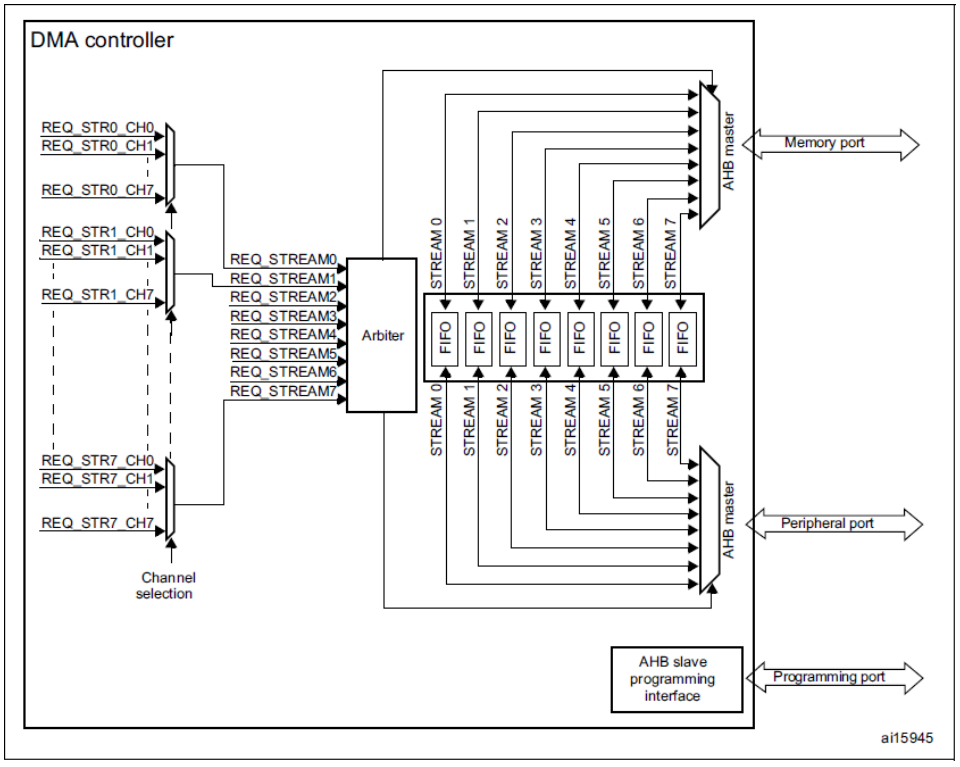
3.2.3. DMA prijenos

DMA se koristi kako bi se omogućio prijenos podataka visokih brzina između periferijskih sklopova i memorije ili između dviju memorijskih jedinica [5]. Podatci se brzo mogu prenijeti bez posredovanja procesora. Na taj se način oslobađaju resursi procesora kako bi se mogle izvoditi druge operacije za vrijeme prijenosa.

U ovom projektu DMA prijenos se koristi između međuspremnika kamere i radne memorije mikrokontrolera.

Razlike DMA periferije na STM32F407VGT6 i STM32L471VGT6 mikrokontrolerima

STM32F407VGT6 mikrokontroler ima dva DMA kontrolera. Blok dijagram jednog DMA kontrolera je prikazan na slici 3.10.



Slika 3.10: Blok dijagram DMA kontrolera na STM32F407VGT6 mikrokontroleru [5]

DMA kontroler obavlja izravan prijenos memorije; kao AHB (engl. *AMBA High-performance Bus*, AMBA na engl. *Advanced Microcontroler Bus Architecture*) *master*, može u bilo kojem trenutku preuzeti kontrolu nad AHB sabirnicom i pokrenuti AHB transakcije. DMA kontroler može pokrenuti sljedeće transakcije:

- prijenos sa periferije na memoriju,

- prijenos sa memorije na periferiju,
- prijenos sa memorije na memoriju.

S obzirom na to da se u ovom radu koriste prijenosi s periferije na memoriju i obrnuto, u daljnjem tekstu će se podrazumijevati samo ti slučajevi.

DMA periferija ima dvije AHB *master* sabirnice, jedna se koristi za pristup memoriji, a druga za pristup periferijama. AHB *slave* sabirnica se koristi za programiranje DMA kontrolera.

Pojedini kontroler ima 8 tokova (engl. *stream*), te za svaki tok postoji 8 kanala (engl. *channel*). Svaki tok je spojen na određeni sklopovski DMA kanal. Tokovi i kanali služe kako bi se ostvarila veza između ostalih periferija i DMA mikokrontrolera, tako da periferije mogu slati zahtjev za DMA prijenos.

Svaki DMA prijenos se sastoji od tri operacije:

- učitavanje sa periferijskog podatkovnog registra ili lokacije u memoriji, čije su adrese zapisane u DMA_SxPAR ili DMA_SxM0AR registru
- spremanje podataka na periferijski podatkovni registar ili lokaciju u memoriji, čije su adrese zapisane u DMA_SxPAR ili DMA_SxM0AR registru
- naknadno dekrementiranje DMA_SxNDTR registra, koji sadržava broj podataka koji se još trebaju prenjeti

Spomenuti registri su prikazani u tablici 3.5. Kada periferija želi pristupiti DMA kontroleru, periferijski sklop šalje zahtjev DMA kontroleru. DMA kontroler poslužuje zahtjev ovisno o prioritetima kanala. Čim DMA kontroler pristupi periferiji, DMA kontroler periferiji šalje signal potvrde. Periferni uređaj ukida svoj zahtjev čim dobije potvrdni signal iz DMA kontrolera. Nakon što periferna jedinica ukine zahtjev, DMA kontroler ukida signal potvrde. Ako ima više zahtjeva, periferna jedinica može pokrenuti sljedeću transakciju.

Tablica 3.5: Registri DMA_SxPAR, DMA_SxM0AR i DMA_SxNDTR kod STM32F407VGT6 mikrokontrolera. *x* označava broj toka [5]

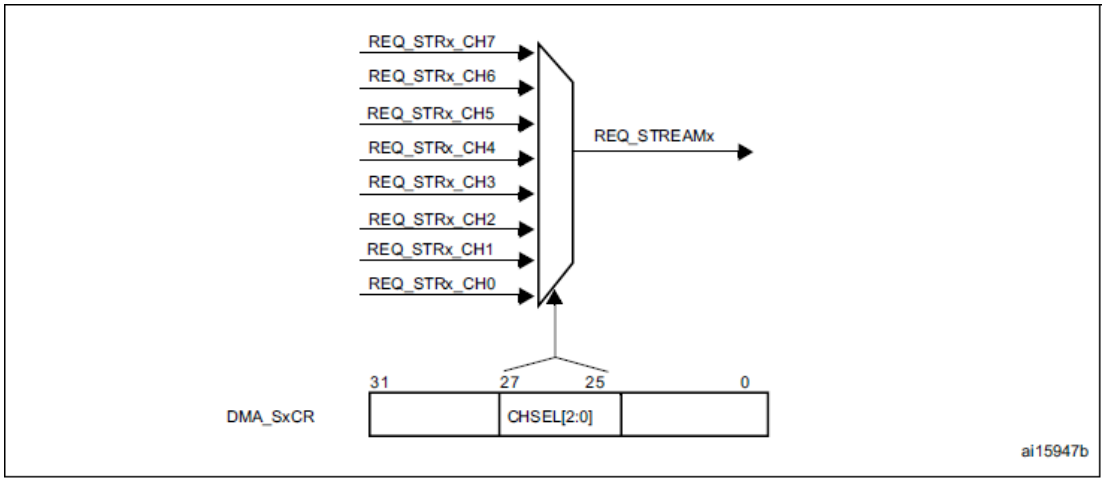
Register name	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
DMA_SxPAR	PA[31:0]																															
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DMA_M0AR	M0A[31:0]																															
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DMA_SxNDTR	Res.																NDT[15:0]															
Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Svaki tok je povezan sa DMA zahtjevom, koji može biti odabran između 8 mogućih

kanalnih zahtjeva. Odabir kanala određuju bitovi CHSEL[2:0] u DMA_SxCR registru (tablica 3.7). Odabir kanala prikazan je na slici 3.11.

Tablica 3.6: Registar DMA_SxCR kod STM32F407VGT6 mikrokontrolera [5]

Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA_SxPAR	Res.				CHSEL[2:0]			MBURST[1:0]		PBURST[1:0]		Res.	CT	DBM	PL[1:0]		PINCOS	MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR[1:0]		PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN
Reset value					0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Slika 3.11: Odabir kanala [5]

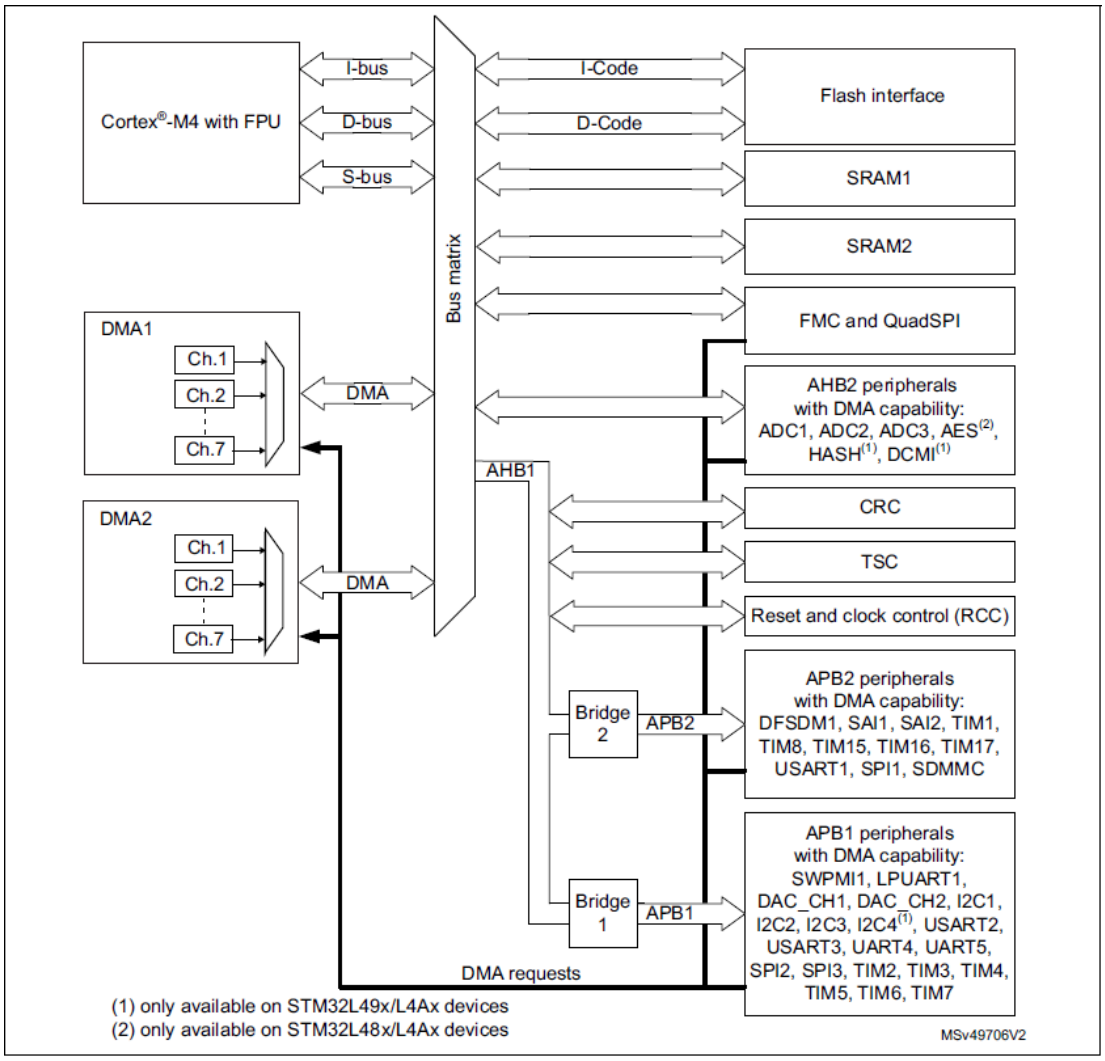
Proučavanjem dokumentacije mikrokontrolera, zaključeno je da je za SPI prijenos putem DMA sklopa potrebno koristiti kanal 0 i tokove 3 (SPI2_RX) i 4 (SPI2_TX) [5, str. 307]. Odabrani tokovi se koriste zato što je kamera spojena na SPI2 periferiju mikrokontrolera.

Blok dijagram DMA periferije na STM32L471VGT6 mikrokontroleru je prikazan na slici 3.12. Vidljivo je da oba mikrokontrolera sadržavaju dvije DMA periferije. Za razliku od SMT32F407VGT6 mikrokontrolera, trenutačni mikrokontroler, STM32L471VGT6, nema dvije AHB *master* sabirnice, već samo jednu AHB *master* sabirnicu.

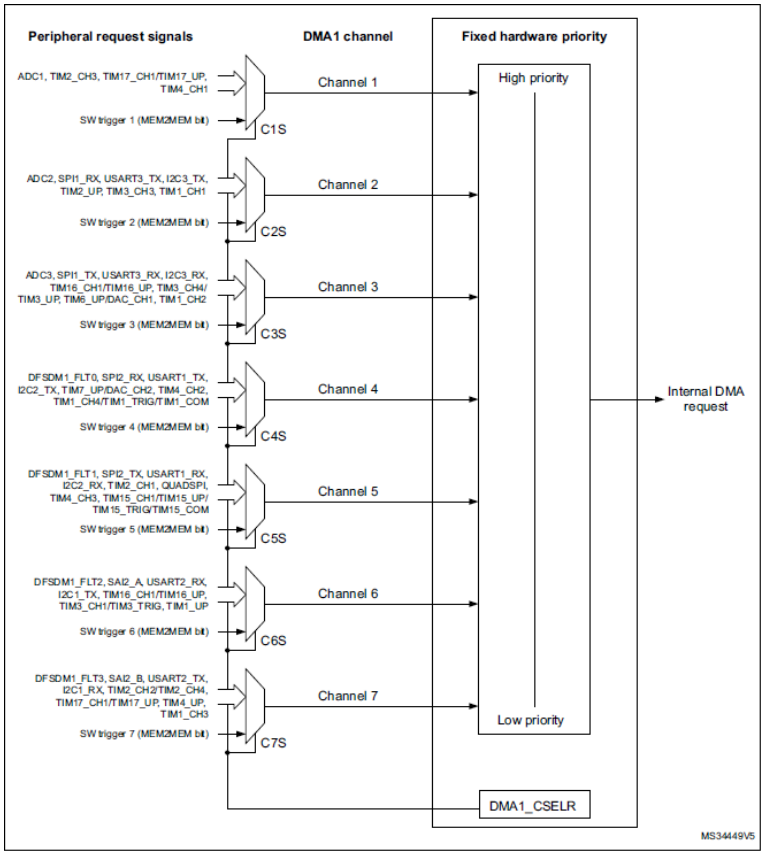
STM32L471VGT6 nema tokove za ostvarivanje veze između periferija i DMA sklopa, već ima samo kanale, te se stoga veza između ostalih periferija i DMA kontrolera ostvaruje na drugačiji način, prikazan na slici 3.13. Iz slike je vidljivo da se na ovom mikrokontroleru trebaju koristiti kanali 4 (SPI2_RX) i 5 (SPI2_TX).

Još jedna razlika između DMA periferija dvaju mikrokontrolera se krije u prekidima koje DMA kontrolera i zastavicama za prekide. STM32F407VGT6 ima 5 razli-

čitih prekida: završetak prijenosa (TC, engl. *Transfer Complete*), obavljena polovica prijenosa (HT, engl. *Half Transfer*), greška u prijenosu (TE, engl. *Transfer Error*), FIFO greška (FE, engl. *FIFO Error*) i greška u direktnom načinu rada (DME, engl. *Direct Mode Error*). STM32L471VGT6 ima 3 različita prekida: završetak prijenosa završetak prijenosa (TC), obavljena polovica prijenosa (HT), greška u prijenosu (TE). Kod STM32L471VGT6 postoji još globalni prekid (GI, engl. *Global Interrupt*) koji se aktivira u svim slučajevima. Ako se, na primjer, želi DMA kontroler namjestiti da zahtijeva prekid kod završetka prijenosa, polovice prijenos i kod greške u prijenosu, to se može podesiti jednostavno aktiviranjem globalnih prekida. S obzirom na to STM32F407VGT6 sadržava više vrsta prekida, on sadržava i 2 prekidna registra, dok STM32L471VGT6 sadržava 1 prekidni registar.



Slika 3.12: Blok dijagram DMA periferije na STM32L471VGT6 mikrokontroleru [6]



Slika 3.13: Mapiranje zahtjeva za DMA1 kontroler kod STM32L471VGT6 mikrokontrolera [6, str. 338]

Što se tiče ostalih dijelova DMA periferija, mikrokontroleri funkcioniraju isto. Postoje, međutim, razlike u načinu konfiguracije DMA kontrolera, međutim, to nije predstavljalo problem, s obzirom na to da je konfiguracija prepuštena generatoru koda. Nazivi registara koji se koriste su također različiti između dva mikrokontrolera, iako su im funkcije iste. To također nije bio problem jer su se koristile definicije registara u programskoj podršci koju je pružao proizvođač mikrokontrolera, pa je bilo potrebno samo promijeniti nazive tih registara.

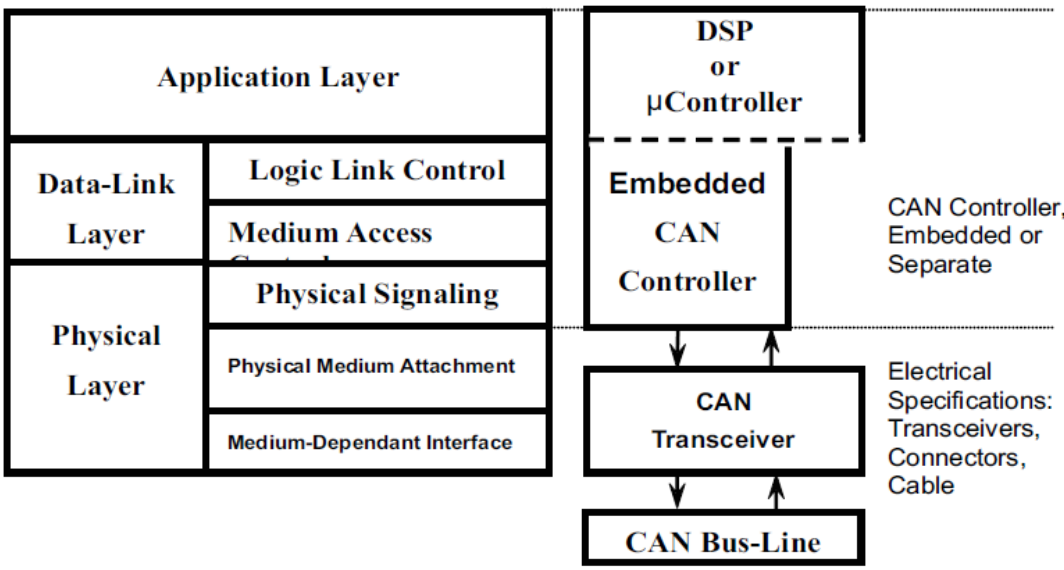
3.3. CAN protokol

S obzirom na to da je na tiskanoj pločici PDH sustava sklopovlje za CAN komunikaciju krivo povezan, nije bilo moguće napisati programsku podršku za CAN komunikaciju. Međutim, u ovom poglavlju dati će se opis protokola, kao i njegova implementacija na mikrokontroleru, te će se izložiti mogućnosti implementacije u ovaj projekt.

3.3.1. Opis protokola

CAN je serijska komunikacijska sabirnica koju je standardizirao ISO (engl. *International Standardization Organization*), a razvijena je od strane tvrtke BOSCH za automobilsku industriju s ciljem da se zamijeni složeni žičani kabel s dvožičnom sabirnicom [7]. Specifikacija zahtijeva su visoka otpornost na električne smetnje i sposobnost otkrivanja i ispravljanja greški kod prijenosa podataka.

Komunikacijski protokol CAN opisuje kako se informacija prenosi između uređaja na mreži i kako odgovara OSI (engl. *Open Systems Interconnection*) modelu koji je definiran u slojevima (slika 3.14). Stvarna komunikacija između uređaja spojenih fizičkim medijem je definirana fizičkim slojem modela.



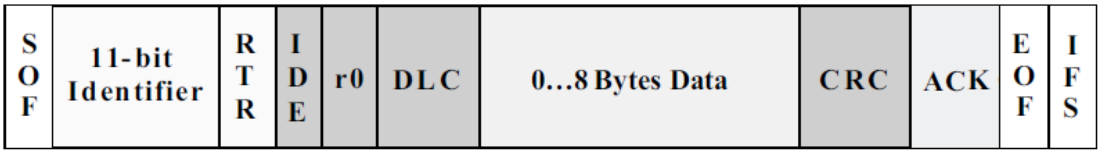
Slika 3.14: OSI model CAN protokola [7, str. 2]

CAN komunikacijski protokol je protokol s višestrukim pristupom, osluškivanjem nosioca, detekcijom sudara i arbitražom na paritet poruka (CSMA/CD+AMP). CSMA znači da svaki čvor na sabirnici mora čekati određeni period neaktivnosti prije nego što pokuša poslati poruku. CD+AMP znači da se sudari rješavaju bitovnom (*bit-wise*) arbitražom, koja se temelji na prethodno namještenom prioritetu svake poruke u identifikacijskom polju poruke. Viši prioritet uvijek dobiva pristup sabirnici.

Standardni CAN protokol s identifikatorom širine 11 bita omogućava brzine prijenosa od 125 kb/s do 1 Mb/s. Standardni protokol u praktičnim primjenama je zamijenjen s proširenim protokolom s identifikatorom širine 29 bita. Standardni 11-bitni identifikator omogućava 2^{11}, ili 2048 različitih identifikatora poruka, dok prošireni 29-bitni identifikator omogućava 2^{29}, ili 536870912 različitih identifikatora.

Standardni CAN protokol

Standardni CAN sa 11-bitnim identifikatorom je prikazan na slici 3.15.



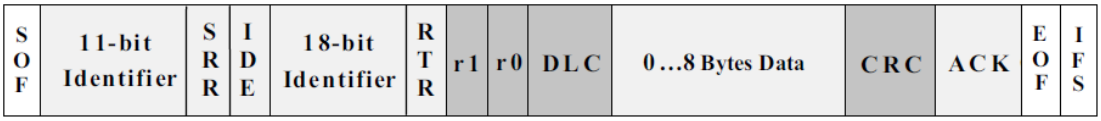
Slika 3.15: Standardna CAN poruka: 11-bitni identifikator [7, str. 3]

Značenje pojedinih bitova na slici 3.15 su:

- SOF - jedan bit, početak okvira (engl. *Start Of Frame*), označava početak poruke i koristi se za sinkronizaciju čvorova na sabirnici nakon mirovanja,
- Identifikator - 11-bitni identifikator standardnog CAN protokola, uspostavlja prioritet poruke. Manja binarna vrijednost znači viši prioritet,
- RTR - jedan bit, zahtjev za udaljenim prijenosom (engl. *Remote Transmission Request*) dominantan je kada se traži informacija s drugog čvora. Svi čvorovi prime zahtjev, ali identifikator određuje traženi čvor. Povratna informacija se također šalje na sve čvorove i svaki čvor ju može iskoristiti ako je potrebno. Na taj su način svi podatci koji se koriste u sustavu uniformni,
- IDE - jedan bit, proširenje identifikatora (engl. *Identifier Extension*), označava da se šalje standardni CAN identifikator bez proširenja,
- r0 - rezervirani bit (za moguću upotrebu kod budućih dopuna standarda),
- DLC - 4-bitna širina podatkovnog koda (engl. *Data Length Code*), sadržava broj bajtova podatka koji se šalje,
- Podatci - može se slati do 64 bitova aplikacijskih podataka,
- CRC - 16-bitna (15 bitova plus granični bit) ciklička provjera redundancije (engl. *Cyclic Redundancy Check*) sadržava kontrolni zbroj (*checksum*, broj poslanih bitova) prethodnih aplikacijskih podataka za detekciju grešaka,
- ACK - svaki čvor koji primi točnu poruku prepisuje ovaj recesivni bit u izvornoj poruci s dominantnim bitom, što znači da je poslana poruka bez greške. Ako prijemni čvor otkrije pogrešku i ostavi ovaj bit recesivnim, on odbacuje poruku i čvor koji šalje poruku ponavlja poruku nakon rearbitraže. Tako svaki čvor priznaje (ACK) integritet svojih podataka. ACK sadrži 2 bita, jedan je bit potvrde, a drugi je graničnik,

- EOF - 7-bitno polje, kraj okvira (engl. *End Of Frame*), označava kraj poruke i onemogućuje trpanje bitova, ukazujući na grešku kod trpanja u slučaju da je dominantan. Kada 5 bitova iste logičke razine nastanu u slijedu kod normalne operacije, bit suprotne logičke razine se *natrpa* u podatke,
- IFS - 7-bitno polje, međuokvirni prostor (engl. *Interframe Space*), sadržava vrijeme potrebno da kontroler pomakne ispravno primljen okvir na njegovu valjanu poziciju u međuspremniku poruka.

Prošireni CAN protokol



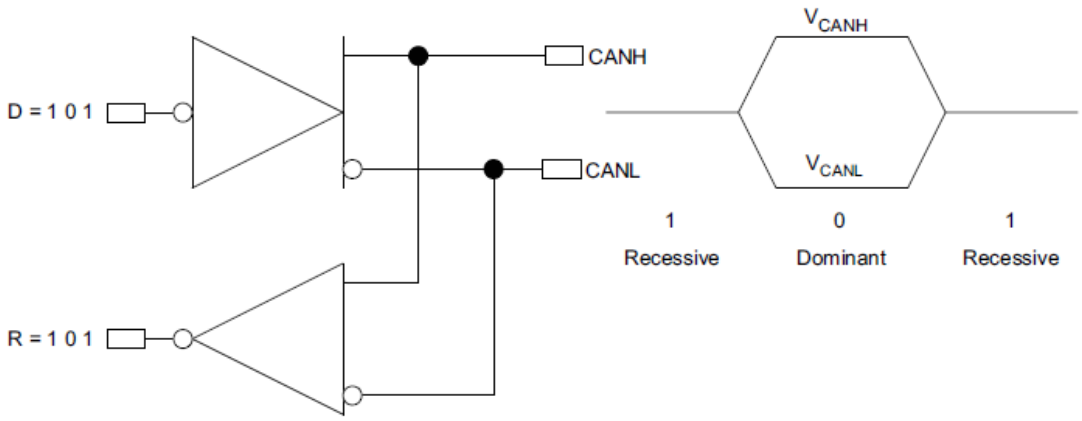
Slika 3.16: Proširena CAN poruka: 29-bitni identifikator [7, str. 4]

Na slici 3.16 je vidljivo da je proširena CAN poruka ista kao i standardna, uz dodatak:

- SRR - zamjenski udaljeni pristup (engl. *Substitute Remote Request*), zamjenjuje RTR bit u standardnoj poruci kao rezervirano mjesto u proširenom formatu,
- IDE - recesivni bit u proširenju identifikatora (engl. *Identifier Extension* označava da slijedi još identifikatorskih bitova. 18-bitno proširenje slijedi nakon IDE,
- r1 - nakon RTR i r0 bitova, dodan je još jedan rezervirani bit prije DLC bita.

CAN poruka

Arbitraža Temeljna karakteristika CAN protokola je suprotno logičko stanje između sabirnice, upravljačkog ulaza i izlaza prijamnika (slika 3.17). U općenitom slučaju visoka logička razina se poistovjećuje s jedinicom, a niska logička razina se poistovjećuje s nulom, međutim, tako nije na CAN sabirnici. Pristup sabirnici se događa nasumično. Ako dva čvora pokušavaju istovremeno zauzeti sabirnicu, pristup se omogućuje pomoću nedestruktivne bitovne arbitraže. Nedestruktivno znači da postupak arbitraže ni na koji način ne utječe na poruke, odnosno čvor koji dobije arbitražu može nastaviti sa slanjem poruke.

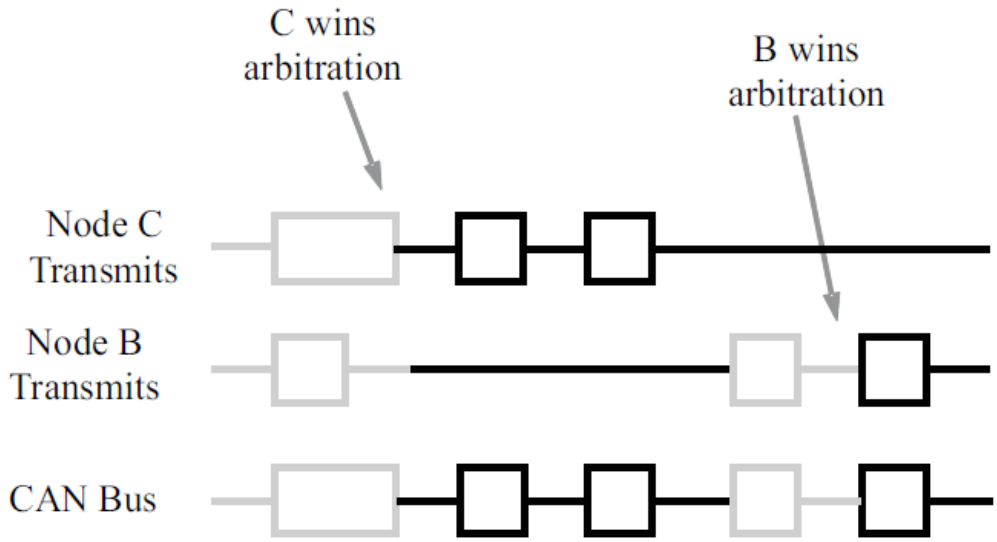


Slika 3.17: Obrnuta logika CAN sabirnice [7, str. 4]. D je upravljački ulaz, a R je izlaz prijamnika

Alokacija prioriteta porukama u identifikatoru je svojstvo CAN protokola koje ga čini pogodnim za upotrebu u sustavima za rad u stvarnom vremenu. Što je binarni broj identifikatora poruke niži, to je prioritet poruke viši. Identifikator koji se u potpunosti sastoji od nula čini poruku najvišeg prioriteta na mreži jer zadržava sabirnicu dominantnom najdulje. Iz tog razloga, ako dva čvora počnu sa slanjem poruke istovremeno, čvor koji pošalje posljednji identifikatorski bit kao nulu (dominantan) dok drugi čvor pošalje jedinicu (recesivan) zadržava kontrolu nad CAN sabirnicom i nastavlja sa slanjem poruke. Dominantan bit uvijek prebriše recesivan bit na CAN sabirnici.

Treba imati na umu da čvor koji šalje poruku stalno nadzire svaki bit svog prijenosa. Ovo je razlog za konfiguraciju primopredajnika na slici 3.17 u kojoj su CANH i CANL izlazne stezaljke upravljačkog sklopa interno povezane s ulazom prijemnika. Kašnjenje radi propagacije signala u unutarnjoj petlji od upravljačkog ulaza do izlaza prijemnika se obično koristi kao kvalitativna mjera CAN primopredajnika. Ovo propagacijsko kašnjenje se naziva vrijeme petlje (engl. *loop time*).

Slika 3.18 prikazuje proces arbitraže kojim CAN kontroler automatski upravlja. Budući da svaki čvor kontinuirano nadzire vlastiti prijenos, dok dominantni bit čvora C prebrisuje recesivni bit čvora B, čvor B detektira da se stanje sabirnice ne poklapa sa bitom koji je poslao. Posljedično, čvor B prestaje sa prijenosom dok čvor C nastavlja sa slanjem poruke. Čvor B ponovno pokušava pristupiti sabirnici nakon što čvor C otpusti sabirnicu.



Slika 3.18: Arbitraža na CAN sabirnici [7, str. 5]

Tipovi CAN poruka Postoje četiri tipova poruka, odnosno okvira, a to su:

- Podatkovni okvir - najčešći oblik poruka, sastoji se od arbitražnog polja, podatkovnog polja, CRC polja, i potvrdnog polja. Arbitražno polje se sastoji od 11-bitnog identifikatora na slici 3.15 i RTR bita, koji je dominantan za podatkovne okvire. Na slici 3.16 sadržava 29 bitova i RTR bit. Slijedi podatkovno polje koje sadržava od 0 do 8 bajtova podataka, a zatim slijedi CRC polje koje sadržava 16-bitni kontrolni zbroj koji se koristi za detekciju pogrešaka. Potvrdno polje je posljednje,
- Daljinski okvir - svrha daljinskog okvira je traženje prijenosa podataka s drugog čvora. Daljinski okvir sličan je podatkovnom okviru, s dvije važne razlike. Prvo, ovaj tip poruke je eksplicitno označen kao daljinski okvir preko recesivnog RTR bita u arbitražnom polju, a druga razlika je da nema podataka,
- Okvir pogreške - ovo je posebna poruka koja krši pravila formata CAN poruke. Odašilje ju čvor kada detektira pogrešku u poruci, i uzrokuje da svi ostali čvorovi u mreži počnu slati okvir pogreške. Izvorni odašiljač zatim automatski ponovno pošalje poruku. Sustav brojača pogrešaka u CAN kontroleru osigurava da čvor ne zauzima sabirnicu uzastopnim slanjem okvira pogreške,
- Okvir preopterećenja - okvir preopterećenja se spominje radi cjelovitosti. Sličan je okviru pogreške u smislu formata i odašilje ga čvor koji postane previše zauzet. Primarno se koristi kao dodatna odgoda između poruka.

Ispravan okvir Kada je posljednji bit završnog EOF polja primljen bez grešaka u recesivnom stanju, smatra se da je poruka bez greške. Dominantan bit u EOF polju uzrokuje da odašiljač ponovi slanje.

Provjera pogrešaka i ograničenje grešaka

Robusnost CAN protokola može se djelomično pripisati njegovim brojnim postupcima provjere pogrešaka. CAN protokol uključuje pet metoda provjere grešaka: tri na razini poruke i dvije na razini bita. Ako poruka ne uspije proći na bilo kojoj od ovih metoda otkrivanja pogreške, ona se ne prihvaća, te prijemni čvor generira okvir pogreške. Ovime se prisiljava odašiljački čvor da ponovno pošalje poruku dok se ne primi ispravna poruka. Ako neispravni čvor prekine sabirnicu kontinuiranim ponavljanjem pogreške, njegov kontroler uklanja sposobnost prijenosa nakon što se dosegne ograničenje broja pogrešaka.

Provjeru pogreške na razini poruke provode CRC i ACK bitovi prikazani na slikama 3.15 i 3.16. 16-bitni CRC sadrži kontrolni zbroj podataka prethodne aplikacije za otkrivanje pogreške s 15-bitnim kontrolnim zbrojem i 1-bitnim graničnikom. Polje ACK je dugo dva bita i sastoji se od bita potvrde i graničnog bita.

Na razini poruke provodi se također i provjera formata. Ova provjera pretražuje polja u poruci koja uvijek moraju sadržavati recesivne bitove. Ako se detektira dominantan bit, stvara se pogreška. Bitovi koji se provjeravaju su SOF, EOF, ACK granični bit i CRC granični bit.

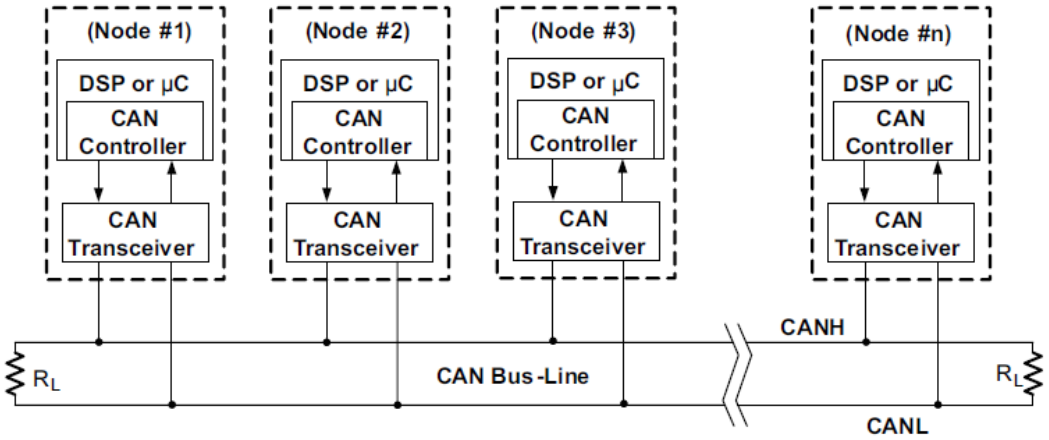
Na razini bita svaki odaslani bit nadzire odašiljač poruke. Ako je podatkovni bit (ne arbitražni bit) zapisan na sabirnicu, a pročitan je suprotan bit, generira se pogreška. Jedine iznimke su kod identifikatorskog polja poruke koja se koriste kod arbitraže i potvrdno polje koje zahtjeva da recesivni bit bude prebrisan od strane dominantnog bita.

Posljednja metoda detekcije pogreške je pravilo umetanja redundantnih bitova (engl. *bit stuffing*), gdje nakon pet uzastopnih bitova iste logičke razine, ako sljedeći bit nije komplement, generira se pogreška. Umetanje redundantnih bitova osigurava dostupnost rastućih bridova za tekuću sinkronizaciju mreže. Umetanje također osigurava da se tok bitova ne zamijeni s okvirom pogreške ili 7-bitnim međuokvirnim prostorom koji predstavlja kraj poruke. Umetnute bitove miče kontroler prijemnog čvora prije nego se podatci proslijede aplikaciji. S ovom logikom, aktivni okvir pogreške sastoji se od šest bitova, što krši pravilo umetanja redundantnih bitova. Ovo se tumači kao pogreška od strane svih CAN čvorova koji zatim generiraju vlastiti okvir pogreške. To

znači da okvir pogreške može biti dugačak od originalnih šest do dvanaest bitova sa svim odgovorima. Nakon ovog okvira pogreške slijedi polje razgraničenja od osam recesivnih bitova i razdoblje mirovanja sabirnice prije ponovnog slanja oštećene poruke. Važno je napomenuti da ponovno poslana poruka još uvijek mora proći arbitražu kako bi dobila pristup sabirnici.

CAN sabirnica

Podatkovna veza i slojevi fizičke signalizacije na slici 3.14, koji su inače transparentni operateru sustava, uključeni su u svaki kontroler koji implementira CAN protokol. Veza s fizičkim medijem se onda implementira kroz linijski primopredajnik kako bi se formirao sistemski čvor prikazan na slici 3.19.

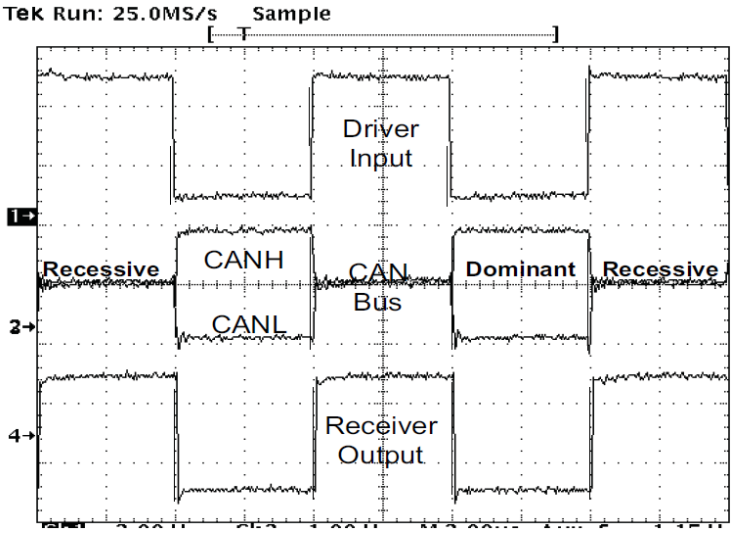


Slika 3.19: Detalji CAN sabirnice [7, str. 7]

Signalizacija je diferencijalna, odakle CAN dobiva svoju robusnu otpornost na šum i toleranciju na pogreške. Uravnoteženo diferencijalno signaliziranje smanjuje šum i dopušta visoku brzinu signalizacije preko kabela s upletenom paricom. Uravnoteženo znači da je tok struje u svakoj signalnoj liniji jednak, ali suprotan po smjeru, što rezultira efektom poništavanja polja, što je ključno za niske emisije šuma.

Specifikacije standarda visoke brzine ISO 11898 dane su za maksimalnu brzinu prijenosa 1 Mbps s duljinom sabirnice od 40 m, s najviše 30 čvorova. Preporučuje se da se linije terminiraju na oba kraja s otpornicima R_L, koji odgovaraju karakterističnoj impedanciji linije kako bi se spriječila refleksija signala.

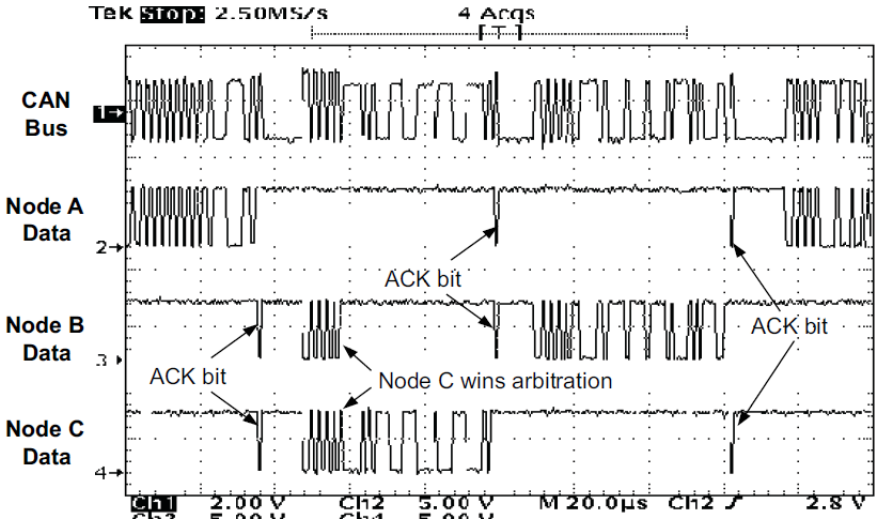
Izgled recesivnih i dominantnih bitova na signalnim linijama CANH i CANL prikazan je na slici 3.20.



Slika 3.20: Dominantno i recesivno stanje na CAN sabirnici [7, str. 8]

CAN standard definira komunikacijsku mrežu koja povezuje sve čvorove spojene na sabirnicu i omogućava im međusobnu komunikaciju. Može, ali ne mora postojati središnji kontrolni čvor, a čvorovi se mogu dodavati u bilo kojem trenutku, čak i kad se na mreži odvija komunikacija (engl. *hot-plugging*).

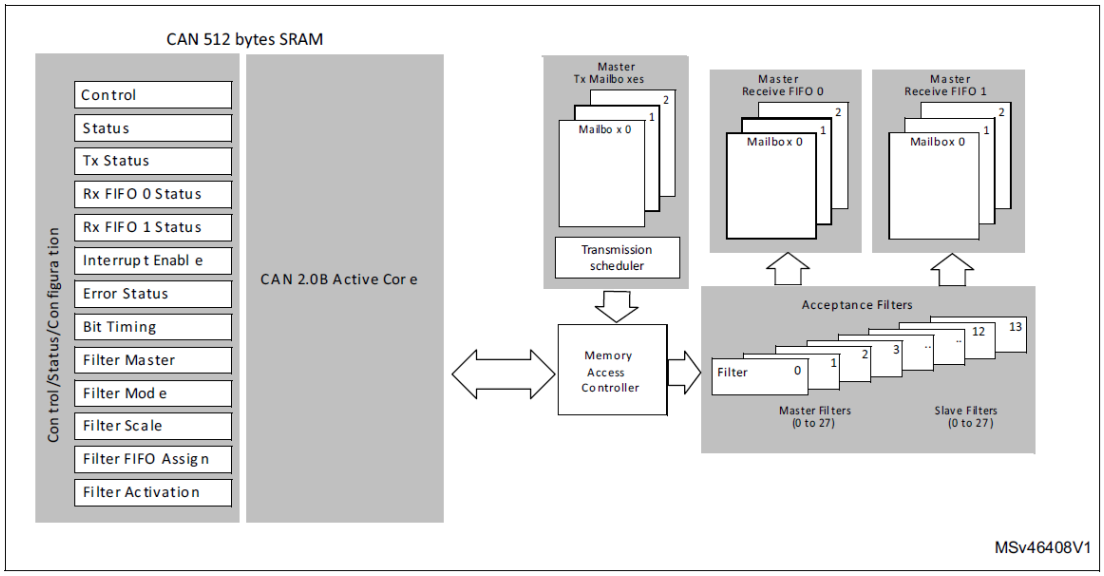
Primjer komunikacije između tri čvora dan je na slici 3.21. Čvor A završi slanje svoje poruke kada čvorovi B i C potvrde primitak ispravne poruke. Čvorovi B i C zatim započnu arbitražu, čvor C dobiva arbitražu i pošalje svoju poruku. Čvorovi A i B potvrde primitak poruke čvora C, te zatim čvor B nastavi sa slanjem svoje poruke. Obratiti pažnju na suprotni polaritet upravljačkog ulaza i izlaza na sabirnici.



Slika 3.21: Promet na CAN sabirnici [7, str. 9]

3.3.2. CAN periferijsko sklopovlje na STM32L471VGT6 mikrokontroleru

CAN periferija kod STM32L471VGT6 mikrokontrolera se naziva bxCAN (engl. *Basic Extended CAN*). Podržava CAN protokole verzije 2.0A i B. Osim aplikacijskih poruka dodane su još i dijagnostičke poruke i poruke za upravljanje mrežom. Blok dijagram CAN periferijskog sklopovlja prikazan je na slici 3.22.



Slika 3.22: Blok dijagram CAN periferije na STM32L471VGT6 mikrokontroleru

Modul bxCAN rukuje s prijenosom CAN poruka potpunosti automatski. Sklopovlje podržava standardne (11-bitne) i proširene (29-bitne) identifikatore. Aplikacija koristi kontrolne, statusne i konfiguracijske registre kako bi:

- konfigurirala CAN parametre, npr. baud rate,
- zahtijevala slanja,
- rukovala prijemima,
- upravljala prekidima,
- dobivala dijagnostičke podatke.

Za postavljanje poruka, programska podrška na raspolaganju ima tri odašiljačke *mailbox* strukture. Planer slanja odlučuje koja *mailbox* struktura će se prvo slati. bxCAN pruža do 14 skalabilnih/podesivih banaka filtara za odabir nadolazećih poruka koje programska podrška treba, a ostale odbacuje. Sklopovlje koristi dvije primajuće FIFO (engl. *First Input First Output*) strukture za spremanje poruka. FIFO znači da se prvi

podatak koji se unese prvi obrađuje. Tri kompletne poruke se mogu pohraniti u svaku FIFO strukturu. Sklopovlje u potpunosti upravlja FIFO strukturama.

Načini rada bxCAN sklopovlja

Sklopovlje bxCAN može raditi u tri glavna načina rada, a to su inicijalizacijski, normalni i spavajući. Nakon sklopovskog reseta, bxCAN je u spavajućem načinu rada kako bi se smanjila potrošnja snage, te se aktivira unutarnji *pull-up* otpornik na CANTX liniji. Programska podrška može zahtijevati da bxCAN uđe u inicijalizacijski ili spavajući način rada tako da postavi INRQ ili SLEEP bitove u visoku logičku razinu u CAN_MCR registru. Nakon što se odabere način rada, bxCAN potvrđuje odabir tako da postavi INAK ili SLAK bitove u visoku logičku razinu u CAN_MSR registru, te se unutarnji *pull-up* otpornik onemogućuje. Kada INAK i SLAK bitovi nisu postavljeni, bxCAN radi u normalnom načinu rada. Prije ulaska u normalan način rada bxCAN se mora sinkronizirati s CAN sabirnicom. Kako bi se sinkronizacija provela, bxCAN čeka da CAN sabirnica bude u mirovanju, a to znači da CANRX linija mora primijetiti 11 recesivnih bitova.

Inicijalizacijski način rada Inicijalizacijalizacija programske podrške se može obaviti dok je sklopovlje u inicijalizacijskom načinu rada. Kako bi se ušlo u ovaj način rada programska podrška treba postaviti INRQ bit u CAN_MCR registru u visoku logičku razinu i pričekati dok sklopovlje ne potvrdi zahtjev postavljanjem INAK bita u CAN_MSR registru u visoku logičku razinu.

Kako bi se inicijalizacijski način rada napustio, programska podrška treba očistiti INRQ bit. bxCAN napušta inicijalizacijski način rada kada sklopovlje očisti INAK bit.

Dok je u inicijalizacijskom načinu rada, sve poruke prema i od CAN sabirnice se zaustavljaju, a stanje izlaza CAN sabirnice CANTX je u recesivnom stanju. Ulaskom u inicijalizacijski način rada konfiguracijski registri se ne mijenjaju.

Kako bi se inicijalizirao CAN kontroler, potrebno je podesiti CAN_BTR (engl. *(*)Bit Timing Register) i CAN_MCR registre. Za inicijalizaciju registara povezanih s CAN bankama filtera (način rada, razmjer, FIFO dodjela, aktivacija i vrijednost filtera), potrebno je postaviti FINIT bit u visoku logičku razinu u CAN_FMR registru. Inicijalizacija filtera se također može obaviti i van inicijalizacijskog načina rada.

Normalni način rada Kada je inicijalizacija gotova, programska podrška mora zahtijevati od sklopovlja da uđe u normalni način rada kako bi se sklopovlje sinkroniziralo s CAN sabirnicom i započelo primanje i slanje poruka.

Zahtjev za ulaskom u normalni način rada se izdaje čišćenjem INRQ bita u CAN_MCR registru. bxCAN ulazi u normalni način rada i spreman je za sudjelovanje u aktivnostima sabirnice kada se sinkronizira s prijenosom podataka na CAN sabirnici. To se radi čekanjem pojave 11 uzastopnih recesivnih bitova, što označuje mirovanje sabirnice. Prelazak u normalni način rada potvrđuje sklopovlje čišćenjem INAK bita u CAN_MSR registru.

Inicijalizacija vrijednosti filtera je neovisno od inicijalizacijskog načina rada, ali se mora obaviti dok filter nije aktivan, tj. dok je odgovarajući FACTx bit u niskoj logičkoj razini. Razmjer filtera i način rada se moraju podesiti prije ulaska u normalni način rada.

Način rada s niskom potrošnjom energije (engl. *sleep mode*) Kako bi se smanjila potrošnja snage, bxCAN ima način rada s niskom potrošnjom energije. U ovaj način rada se ulazi kada programska podrška postavi SLEEP bit u CAN_MCR registru u visoku logičku razinu. U ovom načinu rada, takt bxCAN periferije se zaustavlja, a *mailbox* strukture su još uvijek dostupne programskoj podršci.

Ako programska podrška pošalje zahtjev za ulazak u inicijalizacijski način rada tako da postavi INRQ bit dok je bxCAN u načinu rada s niskom potrošnjom energije, potrebno je i očistiti SLEEP bit. bxCAN može izaći iz načina rada s niskom potrošnjom energije ako se očisti SLEEP bit ili ako se primijeti aktivnost na CAN sabirnici.

Na primijećenu aktivnost CAN sabirnice, sklopovlje automatski provede proces buđenja tako da očisti SLEEP bit, pod uvjetom da je AWUM bit u CAN_MCR registru postavljen u visoku logičku razinu. Ako je AWUM bit u niskoj logičkoj razini, programska podrška mora očistiti SLEEP bit kada se dogodi prekid za buđenje kako bi se izašlo iz načina rada s niskom potrošnjom energije.

Nakon što se očisti SLEEP, bxCAN izlazi iz načina rada s niskom potrošnjom energije kada se sinkronizira sa CAN sabirnicom. Izlazak iz načina rada s niskom potrošnjom energije je dovršen nakon što sklopovlje očisti SLAK bit.

Funkcionalni opis bxCAN periferije

Slanje poruke Kako bi se odaslala poruka aplikacija mora odabrati jednu praznu odašiljačku *mailbox* strukturu, postaviti identifikator, duljinu podatkovnog koda (DLC) i podatke prije zahtijevanja slanja postavljanjem odgovarajućeg TXRQ bita u CAN_TIxR registru. Kada *mailbox* struktura više nije prazna, programska podrška više nema pristup pisanju u registre *mailbox* strukture. Odmah nakon što je TXRQ bit postavljen,

mailbox struktura ulazi u stanje čekanja i čeka da postane *mailbox* struktura najvišeg prioriteta. Čim *mailbox* struktura dobije najviši prioritet, zakaže se njezino slanje. Slanje poruke zakazane *mailbox* strukture počne kada CAN sabirnica postane neaktivna. Kada se poruka *mailbox* strukture uspješno pošalje, ona opet postane prazna. Sklopovlje naznačuje uspješno slanje poruke postavljanjem RQCP i TXOK bitova u CAN_TSR registru u visoku logičku razinu. Ako prijenos zakaže, uzrok se naznačuje ALST bitom u CAN_TSR registru u slučaju gubitka arbitraže i/ili TERR bitom u slučaju detekcije pogreške kod prijenosa.

Prioritet slanja se može odrediti identifikatorom ili redoslijedom zahtjeva za prijenos. Kod prioriteta određenog identifikatorom, kada su više od jedne odašiljačke *mailbox* strukture u stanju čekanja, prioritet slanja se određuje identifikatorom poruke koja je pohranjena u *mailbox* strukturi. Poruka s najnižom vrijednosti identifikatora ima najveći prioritet. Ako su vrijednosti identifikatora iste prioritet se daje *mailbox* strukturi koja ima najmanji broj. Kod prioriteta određenog redoslijedom zahtijeva za prijenos, odašiljačke *mailbox* strukture se mogu konfigurirati kao FIFO strukture za prijenos postavljanjem TXFP bita u CAN_MCR registru. U ovom načinu rada prioritet prijenosa zadan je redoslijedom zahtijeva za prijenos.

Zahtjev za slanjem se može prekinuti ako korisnik postavi ABRQ bit u CAN_TSR registru. Ako je *mailbox* struktura u stanju čekanja ili je zakazana, slanje se odmah prekida. Ako je u stanju slanja, moguća su dva ishoda. Ako je poruka *mailbox* strukture uspješno poslana, struktura postane prazna, a ako prijenos zakaže, struktura postane zakazana za prijenos, te se prijenos prekida i struktura postane prazna.

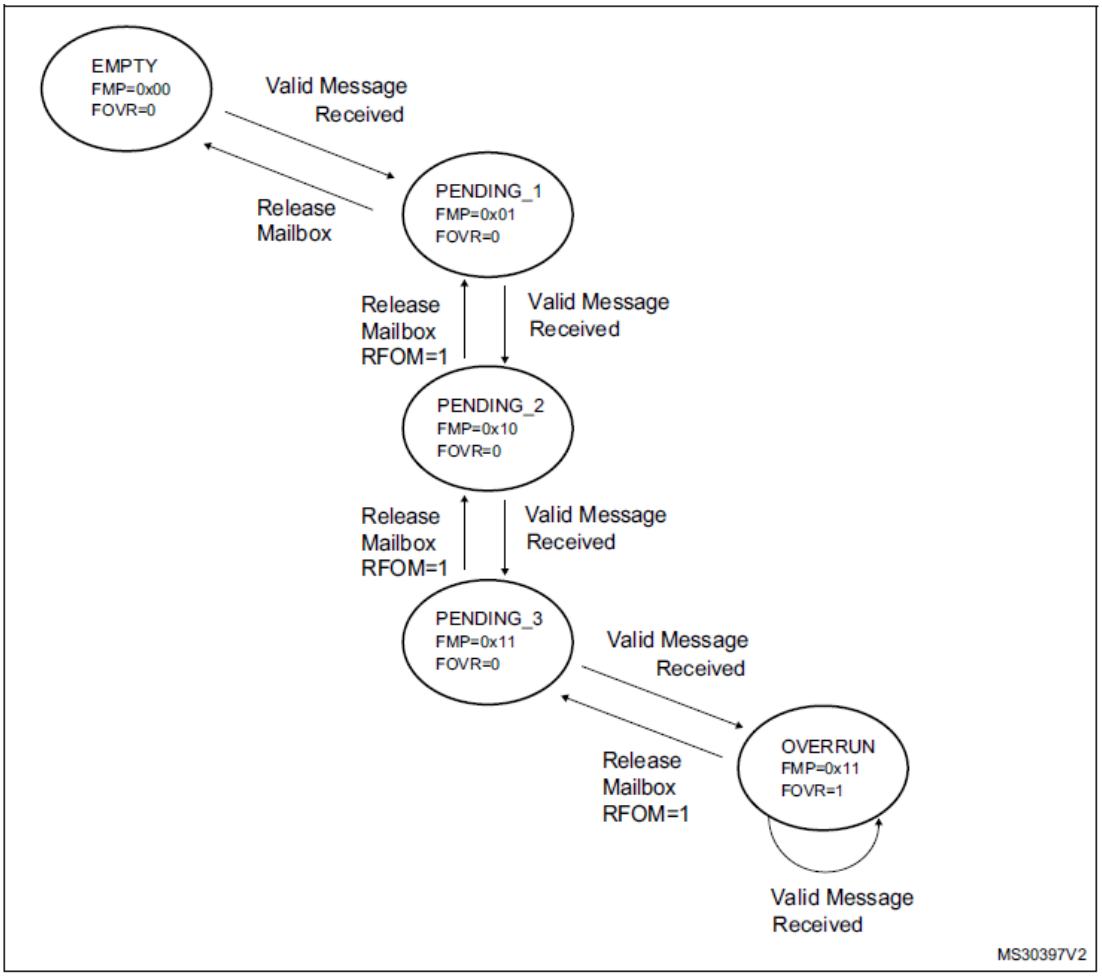
Kako bi se ispunio zahtjev za vremenski okinutom komunikacijom CAN standarda, implementiran je način rada bez automatskog ponovnog slanja. Kako bi se sklopovlje podesilo da radi u ovom načinu rada, NART bit u CAN_MCR registru mora biti postavljen u visoku logičku razinu. U ovom načinu rada svako slanje se pokreće samo jednom. Ako prvi pokušaj zakaže, radi gubitka arbitraže ili pogreške, sklopovlje neće automatski pokrenuti ponovno slanje poruke. Na kraju pokušaja slanja, sklopovlje smatra slanje završenim i postavlja RQCP bit u CAN_TSR registru u visoku logičku razinu. Rezultat slanja se može saznati pregledom bitova TXOK, ALST, TERR u CAN_TSR registru.

Način rada s vremenski okinutom komunikacijom U ovom načinu unutarnje brojilo CAN sklopovlja se aktivira i koristi za generiranje vrijednosti vremenske oznake pohranjene u registrima CAN_RDTxR/CAN_TDTxT (za Rx i Tx *mailbox* strukture). Unutarnje brojilo se poveća za jedan nakon isteka svakog CAN bitovnog vremena.

Unutarnje brojilo bilježi se na točki uzorkovanja na bitu početka okvira u slanju i primanju.

Primanje poruke Za prijem CAN poruke na raspolaganju su tri *mailbox* strukture posložene u FIFO strukturu. Kako bi se smanjila opterećenost procesora, pojednostavnila programska podrška i zagarantirala dosljednost podataka, FIFO strukturom upravlja sklopovlje. Aplikacija pristupa porukama koje su pohranjene u FIFO strukturama kroz izlazne FIFO *mailbox* strukture.

Primljena poruka smatra se ispravnom kada se primi na ispravan način opisan CAN protokolom (nema greške do pretposljednjeg bita EOF polja) i kada uspješno prođe kroz filtriranje identifikatora.



Slika 3.23: Stanja prijemnih FIFO strukuta [6]

Upravljanje FIFO strukturom ilustrirano je slikom 3.23. Počevši od praznog stanja (engl. *empty*), prva primljena ispravna poruka se sprema u FIFO strukturu koja prelazi u *pending_1* stanje. Sklopovlje signalizira taj događaj postavljanjem bitova FMP[1:0]

u CAN_RFR registru na vrijednost 01b. Poruka postaje dostupna u izlaznoj FIFO *mailbox* strukturi. Programska podrška pročita sadržaj *mailbox* te ju otpušta postavljanjem RFOM bita u CAN_RFR registru, te FIFO struktura ponovno postane prazna. Ako je, u međuvremenu, primljena još jedna ispravna poruka FIFO struktura ostaje u *pending_1* stanju i nova poruka je dostupna u izlaznoj FIFO *mailbox* strukturi. Ako aplikacija ne otpusti *mailbox* strukturu, sljedeća ispravna poruka se sprema u FIFO strukturu te se ulazi u *pending_2* stanje(FMP[1:0]=10b). Proces pohrane se ponavlja za sljedeću ispravnu poruku te FIFO ulazi u *pending_3* stanje (FMP[1:0]=11b). U ovom trenutku programska podrška mora otpustiti *mailbox* strukturu postavljanjem RFOM bita, kako bi struktura bila dostupna za pohranu sljedeće ispravne poruke, u suprotnom sljedeća ispravna poruka uzrokuje gubitak poruke.

Kada je FIFO struktura u *pending_3* stanju, sljedeća ispravna poruka dovodi do prekoračenja i poruka se gubi. Sklopovlje signalizira prekoračenje postavljanjem FOVR bita u CAN_RFR registru. Koja se poruka točno gubi ovisi o konfiguraciji FIFO strukture:

- ako je funkcija zaključivanja FIFO strukture isključena (RFLM bit u CAN_MCR registru je očišćen) posljednja poruka koja je pohranjena u FIFO strukturi se prepisuje novom porukom. U ovom slučaju najnovija poruka je uvijek dostupna aplikaciji,
- ako je funkcija zaključivanja FIFO strukture uključena (RFLM bit u CAN_MCR registru je postavljen u jedinicu) najnovija poruka se odbacuje i programska podrška ima pristup trima najstarijim porukama.

Jednom kada se poruka pohrani u FIFO strukturu, ažuriraju se bitovi FMP[1:0] i generira se zahtjev za prekidom, pod uvijetom da je FMPIE bit u CAN_IER registru postavljen u visoku logičku razinu. Kada FIFO struktura postane puna FULL bit u CAN_RFR registru se postavlja u visoku logičku razinu i generira se zahtjev za prekid, pod uvijetom da je FFIE bit u CAN_IER registru postavljen u jedinicu. Ako je FOVR bit u CAN_IER registru postavljen u jedinicu, generira se zahtijev za prekidom svaki put kada se dogodi prekoračenje.

Filtriranje identifikatora U CAN protokolu identifikator poruke nije povezan s adresom čvora nego je povezan sa sadržajem poruke. Posljedično, odašiljač šalje svoju poruku svim čvorovima. Na primitak poruke čvor odlučuje da li mu je poruka potrebna ili ne, ovisno o vrijednosti identifikatora. Ako je poruka potrebna kopira se u SRAM, a ako nije onda se odbacuje bez intervencije programske podrške. Kako bi se ispunio

ovaj zahtjev, na raspolaganju su 14 podesivih i skalabilnih banaka filtera. Svaka banka filtera se sastoji od da 32-bitna registra, CAN_FxR0 i CAN_FxR1.

Kako bi se filteri optimizirali i prilagodili prema potrebama aplikacije, svaka banka filtera se može neovisno skalirati. Ovisno o skali filtera, banka filtera pruža:

- jedan 32-bitni filter za STDIT[10:0], EXTID[17:0], IDE i RTR bitove,
- dva 16-bitna filtra za STDIT[10:0], RTR, IDE i EXTID[17:15].

Filteri mogu biti podešeni tako da rade u maskiranom načinu rada ili u načinu rada s popisom identifikatora. U maskiranom načinu rada identifikatorski registri su povezani s registrima maske koja određuje koji bitovi identifikatora se moraju podudarati (engl. *must match*) ili koji nisu bitni (engl. *don't care*). U načinu rada s popisom identifikatora registri maske se koriste kao identifikatorski registri. Tako umjesto da se definira identifikator i maska, definiraju se dva identifikatora. Svi se bitovi nadolazećeg identifikatora moraju podudarati s bitovima specificiranim u registrima filtra.

Banke filtera se podešavaju preko odgovarajućeg CAN_FMR registra. Kako bi se podesila banka filtera, prvo se mora deaktivirati banka čišćenjem FACT bita u CAN_FAR registru. Skala filtera se podešava preko odgovarajućeg FSCx bita u CAN_FS1R registru. Način rada za odgovarajuće registre maske ili identifikatorske registre se podešava preko FBMx bita u CAN_FMR registru.

Jednom kada FIFO struktura primi poruku, ona je dostupna aplikaciji. Aplikacijski podaci se obično kopiraju u neku lokaciju u SRAM memoriji. Kako bi se podatak kopirao na pravu lokaciju, aplikacija mora identificirati podatak uz pomoć identifikatora. Kako bi se to izbjeglo i olakšalo pristup lokacijama u SRAM memoriji, CAN kontroler pruža indeks podudaranja filtra. Indeks se pohranjuje u *mailbox* strukturu zajedno sa porukom prema pravilima prioriteta filtra. Tako svaka primljena poruka ima pridružen indeks podudaranja filtra. Index podudaranja filtra se može iskoristiti na dva načina:

- uspoređivanje indeksa s popisom očekivanih vrijednosti,
- korištenje indeksa na nizu za pristup lokaciji podatka.

Za nemaskirane filtre programska podrška ne treba uspoređivati identifikator. Ako je filter maskiran programska podrška smanjuje uspoređivanje samo na maskirane bitove.

Pravila prioriteta filtra.

Ovisno o kombinaciji filtra može se dogoditi da identifikator prođe uspješno kroz nekoliko filtera. U ovom slučaju vrijednost podudaranja filtra pohranjena u primajućoj *mailbox* strukturi se odabire prema sljedećim pravilima prioriteta:

- 32-bitni filtar ima veći prioritet od 16-bitnog filtra,

- za filtre jednake skale, prioritet se daje filtru s načinom rada s popisom identifikatora, a ne filtru s maskiranim načinom rada,
- za filtre s istom skalom i načinom rada, prioritet se određuje prema broju filtra (manji broj, veći prioritet).

Pohrana poruke Sučelje između programske podrške i sklopovlja za CAN poruke je implementirano preko *mailbox* struktura. *Mailbox* struktura sadržava sve informacije povezane s porukom; informacije o identifikatoru, podacima, kontroli, statusu i vremenskoj oznaci.

Programska podrška sastavlja poruku koja se treba poslati u praznoj *mailbox* strukturi. Status odašiljanja naznačuje sklopovlje u CAN_TSR registru.

Kada se poruka primi, ona je dostupna programskoj podršci u FIFO izlaznoj *mailbox* strukturi. Kada programska podrška pročita poruku, programska podrška mora otpustiti *mailbox* strukturu preko RFOM bita u CAN_RFR registru kako bi sljedeća nadolazeća poruka bila dostupna. Indeks podudaranja filtera se pohranjuje u MFMI polju u CAN_RDTxR registru. 16-bitna vrijednost vremenske oznake se pohranjuje u TIME[15:0] polju u CAN_RDTxR registru.

Upravljanje pogreškama Upravljanje pogreškama kao što je opisano u CAN protokolu odrađuje sklopovlje koristeći brojač odašiljačkih pogrešaka (engl. *Transmit Error Counter*, TEC vrijednost u CAN_ESR registru) i brojač prijamničkih pogrešaka (engl. Receive Error Counter, REC vrijednost u CAN_ESR registru) koji se inkrementiraju ili dekrementiraju ovisno o stanju pogreške. Obje vrijednosti se mogu pročitati kako bi se ustanovila stabilnost mreže. Nadalje, CAN sklopovlje pruža detaljne informacije o trenutačnom statusu greške u CAN_ESR registru. Preko CAN_IER registra programska podrška može podesiti stvaranje prekida na detekciju pogreške.

Kada je vrijednost TEC polja veća od 255 dolazi do *Bus-Off* stanja koje je naznačeno preko BOFF bita u CAN_ESR polju. U *Bus-Off* stanju bxCAN periferija više ne može slati i primati poruke. Ovisno o ABOM bitu u CAN_MCR registru, bxCAN se oporavi od *Bus-Off* stanja automatski ili na zahtjev programske podrške. U oba slučaja bxCAN mora čekati redoslijed oporavka specificiran u CAN standardu (128 pojava 11 uzastopnih recesivnih bitova na CANRX liniji).

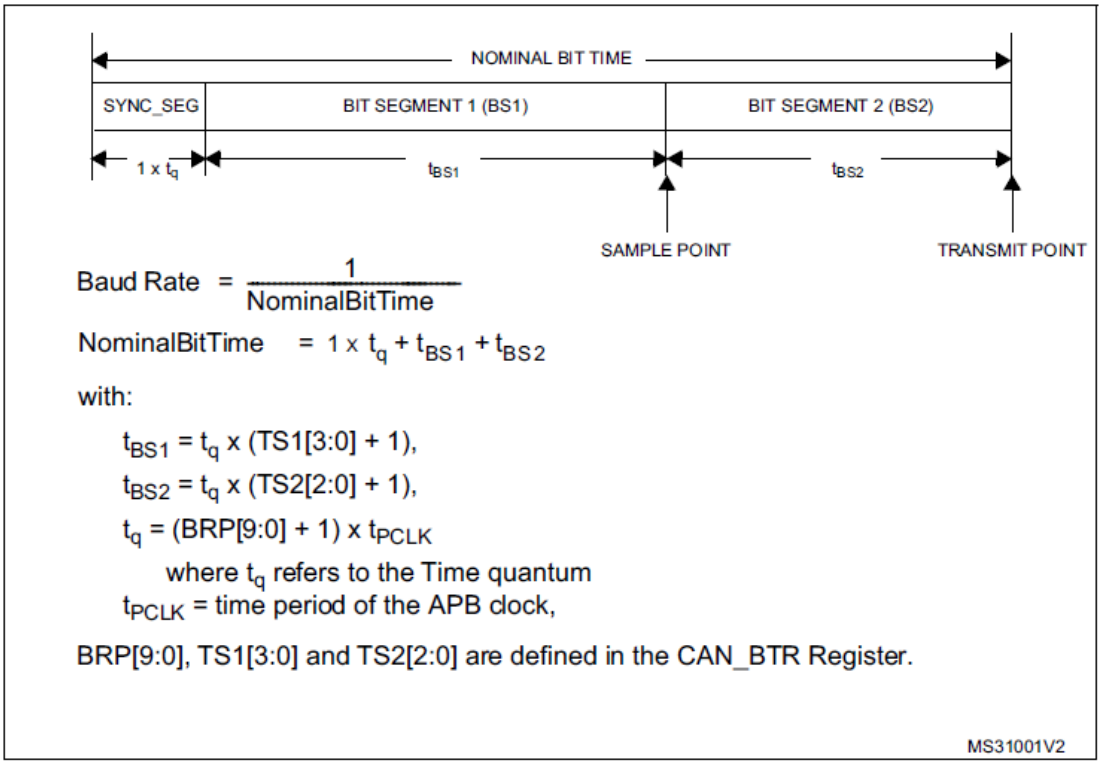
Vrijeme bita Logika vremena bita nadzire liniju serijske sabirnice i izvodi uzorkovanje i podešavanje točke uzorkovanja sinkronizacijom na rubu početnog bita i ponovnom sinkronizacijom na sljedećem rubovima.

Način rada logike vremena bita se može objasniti dijeljenjem nominalnog vremena bita na tri segmenta (slika 3.24):

- sinkronizacijski segment (SYNC_SEG) - u ovom segmentu se očekuje promjena bita. Njegova duljina je fiksna i iznosi jedan vremenski kvant (1 x t_q),
- prvi segment bita (BS1) - određuje lokaciju točke uzorkovanja. Njegovo trajanje je programabilno između 1 i 16 vremenskih kvanta, ali može se automatski produžiti kako bi se kompenzirao pozitivni fazni pomak koji nastaje radi razlika u frekvencijama raznih čvorova na mreži,
- drugi segment bita (BS2) - određuje lokaciju točke odašiljanja. Njegovo trajanje je programabilno između 1 i 8 vremenskih kvanti, ali može se automatski smanjiti kako bi se kompenzirao negativni fazni pomak.

Širina skoka ponovne sinkronizacije (*resynchronization jump width*, SJW) određuje gornju granicu iznosa produženja ili skraćivanja segmenata bita. Programabilna je između 1 i 4 vremenskih kvanta.

Valjan rub se definira kao prva promjena u vremenu bita iz dominantnog u recesivno stanje sabirnice, pod uvjetom da kontroler sam ne pošalje recesivni bit. Ako se valjani rub detektira u BS1 umjesto SYNC_SEG, BS1 se produžuje za do SJW, kako bi se odgodila točka uzorkovanja. Ako se valjanu rub detektira u BS2 umjesto u SYNC_SEG, BS2 se skraćuje za do SJW, kako bi se točka odašiljanja pomaknula ranije.



Slika 3.24: Prikaz i proračun vremena bita [6, str. 1645]

Tablica 3.7: Registri CAN periferije kod STM32L471VGT6 mikrokontrolera [6, poglavlje 46]

Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAN_MCR	Res.															DBF	RESET	Res.							TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
Reset value																1	0								0	0	0	0	0	0	1	0
CAN_MSR	Res.																				RX	SAMP	RXM	TXM	Res.			SLAKI	WKUI	ERRI	SLAK	INAK
Reset value																					1	1	0	0				0	0	0	1	0
CAN_TSR	LOW[2:0]			TME[2:0]			CODE[1:0]		ABRQ2	Res.			TERR2	ALST2	TXOK2	RQCP2	ABRQ1	Res.			TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res.			TERR0	ALST0	TXOK0	RQCP0
Reset value	0	0	0	1	1	1	0	0	0				0	0	0	0	0				0	0	0	0	0				0	0	0	0
CAN_RF0R	Res.																										RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]	
Reset value																											0	0	0		0	0
CAN_RF1R	Res.																										RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]	
Reset value																											0	0	0		0	0
CAN_IER	Res.														SLKIE	WKUIE	ERRIE	Res.			LECIE	BOFIE	EPVIE	EWGIE	Res.	FOVIE1	FFIE1	FMPIE1	FOVIE0	FFIE0	FMPIE0	TMEIE
Reset value															0	0	0				0	0	0	0		0	0	0	0	0	0	0
CAN_ESR	REC[7:0]								TEC[7:0]									Res.									LEC[2:0]		Res.	BOFF	EPVF	EWGF
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										0	0	0		0	0	0
CAN_BTR	SILM	LBKM	Rot.				SJW[1:0]		Res.	TS2[2:0]			TS1[3:0]					Res							BRP[9:0]							
Reset value	0	0					0	0		0	1	0	0	0	1	1							0	0	0	0	0	0	0	0	0	0
CAN_TIxR	STID[10:0]/EXID[28:18]								EXID[17:0]																				IDE	RTR	TXRQ
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
CAN_TDTxR	TIME[15:0]																	Res.						TGT	Res.				DLC[3:0]			
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X								X					X	X	X	X
CAN_TDLxR	DATA3[7:0]								DATA2[7:0]									DATA1[7:0]							DATA0[7:0]							
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_TDHxR	DATA7[7:0]								DATA6[7:0]									DATA5[7:0]							DATA4[7:0]							
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_RIxR	STID[10:0]/EXID[28:18]								EXID[17:0]																				IDE	RTR	Res.
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_RDTxR	TIME[15:0]																	FMI[7:0]							Res.				DLC[3:0]			
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_RDLxR	DATA3[7:0]								DATA2[7:0]									DATA1[7:0]							DATA0[7:0]							
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_RDHxR	DATA7[7:0]								DATA6[7:0]									DATA5[7:0]							DATA4[7:0]							
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CAN_FMR	Res.																		CANSB[5:0]						Res.							FINIT
Reset value																			0	0	1	1	1	0								1
CAN_FM1R	Res.				FMB[27:0]																											
Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CAN_FS1R	Res.				FSC[27:0]																											
Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CAN_FFA1R	Res.				FFA[27:0]																											
Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CAN_FA1R	Res.				FACT[27:0]																											
Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CAN_FiRx	FB[31:0]																															
Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

4. Programska podrška

Prilikom razvoja programske podrške korišteno je integrirano razvojno okruženje STM32CubeIDE, PDH sustav i programator ST-LINK/V2. Programska potpora razvijena je u jeziku C uz korištenje GCC prevodioca.

4.1. Korišteni programski paketi i biblioteke

Razvojno okruženje STM32CubeIDE nudi mogućnost grafičkog podešavanja parametara periferija i automatsko generiranje koda, čime su olakšane inicijalizacije raznih perifernih sklopova. Prije generiranja koda moguće je odabrati hoće li se koristiti HAL (engl. *Hardware Abstraction Layer*) ili LL (engl. *Low-Layer*) biblioteke.

HAL biblioteke nude visoku razinu abstrakcije i lakše su za koristiti, međutim, one rade po principu „crne kutije", te ih stoga korisnik teže razumije. HAL biblioteke također zauzimaju više radne memorije od LL biblioteka jer u sebi sadržavaju kojekakve provjere vrijednosti i modifikacije unutarnjih HAL struktura, pa time nisu osigurane optimalne performanse.

Iz tog razloga odlučeno je da će se koristiti LL biblioteke. LL biblioteke omogućuju izravan pristup registrima periferija i korisnik ih daleko jednostavnije razumije, s obzirom na to da se često sastoje od samo jedne linije koda.

4.2. Programska potpora za kameru

Za rad s kamerom razvijene su korisničke funkcije prikazane u isječku koda 4.1.

```
1 uint8_t ACAM_TestComms(void);
2 uint8_t ACAM_SPI_Read(uint8_t reg);
3 void ACAM_SPI_Write(uint8_t reg, uint8_t val);
4 void ACAM_spi_read_package(uint8_t * buff, uint16_t size);
5 void ACAM_I2C_Setup();
6 uint8_t ACAM_I2C_Read(uint16_t reg);
```

```
7  void ACAM_I2C_Write(uint16_t reg, uint8_t command);
8  void ACAM_I2C_WriteSeq(const struct ACAM_I2C_Register commandSeq
       []);
9  uint8_t ACAM_TestComms(void);
10 void ACAM_select_JPEG(void);
11 void ACAM_select_RAW(uint8_t resolution);
12 void ACAM_start_capture(void);
13 uint32_t ACAM_get_image_size();
14 void ACAM_set_exposure(uint16_t nr_lines, uint8_t nr_lines_frac);
15 void ACAM_Reset(void);
16 void ACAM_exp_gain_manual(void)
17 void ACAM_exp_gain_auto(void)
18 uint8_t ACAM_is_cap_complete(void)
19 void ACAM_set_gain(uint8_t gain);
```

Isječak koda 4.1: Korisničke funkcije za Arducam 5MP Mini Plus

Za testiranje SPI i I^{2}C komunikacije između mikrokontrolera i kamere postoji funkcija `ACAM_TestComms()`, za odabir između RAW i JPEG formata slike na raspolaganju su funkcije `ACAM_select_RAW()` i `ACAM_select_JPEG()`. Način upravljanja ekspozicijom odabire se funkcijama `ACAM_exp_gain_manual()` i `ACAM_exp_gain_auto()`, a ukoliko se odabere ručni način upravljanja ekspozicijom parametri vremena ekspozicije i pojačanja pojačala se podešavaju funkcijama `ACAM_set_exposure()` i `ACAM_set_gain()`. Komanda za početak slikanja se šalje funkcijom
`ACAM_start_capture()`, a provjera završetka slikanja se obavlja funkcijom `ACAM_is_cap_complete()`. Navedene funkcije nije trebalo mijenjati u odnosu na prethodnu programsku podršku, pa one rade na isti način kao i u prethodnom radu [3]. Funkcije koje direktno rade sa SPI i I^{2}C periferijom su jedine mijenjane s obzirom na to da se rad periferija između prethodno korištenog i trenutačno korištenog mikrokontrolera razlikuju, kako je i pokazano u poglavlju 3. U nastavku će biti istaknute razlike između starih i novih funkcija kao i obrazloženje radi čega je došlo do promjena.

4.2.1. Funkcije za rad sa I^{2}C periferijom

Funkcije za rad s I^{2}C periferijom navedene su u isječku koda 4.2.

```
1  void ACAM_I2C_Setup();
2  uint8_t ACAM_I2C_Read(uint16_t reg);
```

```
3 void ACAM_I2C_Write(uint16_t reg, uint8_t command);
4 void ACAM_I2C_WriteSeq(const struct ACAM_I2C_Register commandSeq
      []);
```

Isječak koda 4.2: Funkcije za rad s I^2C periferijom

U odnosu na prethodnu programsku podršku dodana je funkcija `ACAM_I2C_Setup()` koja podešava I^2C periferiju. Funkcija postavlja adresu uređaja s kojim se želi komunicirati, odnosno SADD registar, i govori periferiji želi li se na uređaj nešto čitati ili pisati, odnosno podešava RD_WRN bit u I2C_CR2 registru. Tu funkciju je važno pozvati prije svakog pokušaja komunikacije s kamerom. Funkcije `ACAM_I2C_Write()`, `ACAM_I2C_Read()` i `ACAM_I2C_WriteSeq()` služe za prijenos podataka između kamere i mikrokontrolera, i one su prošle manje promjene, konkretno, bilo je potrebno omogućiti drugačije prekide i provjeravati drugačije statusne zastavice, zato što registarska mapa I^2C periferije izgleda drugačije na STM32L471VGT6 mikrokontroleru nego na STM32F407VGT6 mikrokontroleru. Navedene promjene nije bilo teško implementirati zato što LL funkcije koriste intuitivna imena. Na primjer, funkcija koja omogućava TC prekid (prekid za završetak prijenosa) je `LL_I2C_EnableIT_TC()`, a funkcija koja provjerava da li je podignuta zastavica BUSY (prijenos je u tijeku) je `LL_I2C_IsActiveFlag_Busy()`.

Prijenos preko I^2C protokola u ovom radu funkcionira tako da se u jednoj od funkcija koje su zadužene za prijenos podataka postavi adresa uređaja s kojim se želi komunicirati (u ovom slučaju kamera), nakon toga generira se *start* uvjet i omogućuju se prekidi relevantni za vrstu prijenosa koja se želi obaviti (slanje ili primanje podataka). Stanje komunikacije se prati preko globalne zastavice koja se osvježava u prekidnoj podrutini `I2C1_EV_IRQHandler()`. Jedina promjena koju je prekidan funkcija doživjela jest ta da je izbrisan kod za provjeru je li poslana adresa *slave* uređaja. Naime, na prethodno korištenom mikrokontroleru nije postojao posebni registar za adresu *slave* uređaja, nego se adresa slala preko izlaznog registra periferiju. Kod novog mikrokontrolera, čim se pošalje *start* bit šalje se adresa *slave* uređaja preko SADD registra i automatski se provjerava da ACK bit za potvrdu primitka točne adrese *slave* uređaja. Ostatak funkcije funkcionira na isti način kao i na prethodnom mikrokontroleru.

4.2.2. Funkcije za rad sa SPI periferijom

Funkcije za rad s SPI periferijom navedene su u isječku koda 4.3.

```
1 uint8_t ACAM_SPI_Read(uint8_t reg);
```

```
2 void ACAM_SPI_Write(uint8_t reg, uint8_t val);
3 void ACAM_spi_read_package(uint8_t * buff, uint16_t size);
```

Isječak koda 4.3: Funkcije za rad s SPI periferijom

Kao što je već rečeno, SPI periferija funkcionira na isti način na oba mikrokontrolera, te stoga funkcije `ACAM_SPI_Read()` i `ACAM_SPI_Write()` funkcioniraju na isti način kao i u programskoj podršci za prethodni mikrokontroler.

Do razlike, međutim, dolazi u funkciji `ACAM_spi_read_package()`, jer ta funkcija koristi DMA prijenos, a kao što je već pokazano, DMA prijenos funkcionira drugačije na STM32L471VGT6 mikrokontroleru nego na STM32F407VGT6 mikrokontroleru. U prethodnoj programskoj podršci koristili su se takozvani tokovi kod DMA prijenosa. Na primjer, za slanje podataka preko SPI protokola pomoću DMA prijenosa koristio se tok 4, a u programskoj podršci tok 4 je označavala definicija `LL_DMA_STREAM_4`. U programskoj podršci za trenutačni mikrokontroler ne postoje takve definicije za tokove, s obzirom na to da se prijenos obavlja preko takozvanih kanala. Ako se žele poslati podatci preko SPI protokola pomoću DMA prijenosa mora se koristiti kanal 5, a za primanje podataka koristi se kanal 4, a u programskoj podršci kanale 4 i 5 predstavljaju definicije `LL_DMA_CHANNEL_4` i `LL_DMA_CHANNEL5`. Navedene promjene bilo je jednostavno implementirati jer se koriste globalne definicije preko kojih se programskoj podršci govori koje dijelove periferije se koriste, tako da je bilo potrebno promijeniti svega nekoliko linija koda (isječak koda 4.4).

```
1 #define ACAM_SPIn   SPI2
2
3 #define ACAM_LL_DMA_CHANNEL_TX  LL_DMA_CHANNEL_5
4 #define ACAM_LL_DMA_CHANNEL_RX  LL_DMA_CHANNEL_4
5
6 #define ACAM_DMA_CHANNEL_TX_IRQn    DMA1_Channel5_IRQn
7 #define ACAM_DMA_CHANNEL_RX_IRQn    DMA1_Channel4_IRQn
8
9 #define ACAM_DMAn   DMA1
```

Isječak koda 4.4: Zaglavlje dma.h datoteke u kojima se definira koji dijelovi periferija se koriste

Konačno, trebalo je izbrisati funkcije koje se bave prekidima koji ne postoje na trenutačnom mikrokontroleru i prilagoditi prekidne funkcije da rade s prekidima kanala 4 i 5. Na primjer, kod STM32L471VGT6 mikrokontrolera ne postoje DMEIF i FEIF zastavice, pa je, na primjer, funkcije `LL_DMA_ClearFlag_DMEx()` i

`LL_DMA_ClearFlag_FEx()` jednostavno bilo potrebno izbrisati (x u nazivu funkcije može biti 3 ili 4, što predstavlja broj toka). Funkcije koje rade s HT, TC i TE prekidima, na primjer `LL_DMA_ClearFlag_HTx()`, `LL_DMA_ClearFlag_TCx()` i `LL_DMA_ClearFlag_TEx()`, je trebalo promijeniti da rade s brojem kanala koji se koriste, a to znači promijeniti x u 4 ili 5, što predstavlja broj kanala. Na isti način modificirane su druge funkcije koje rade s prekidima kanala

Na taj je način postojeća programska podrška prethodnog mikrokokontrolera za kameru prilagođena za trenutačni mikrokontroler.

4.3. Programska potpora za *flash* memoriju

Za rad s *flash* memorijom razvijene su korisničke funkcije prikazane u isječku koda 4.5.

```
1 uint8_t W25N_init(void);
2 uint8_t W25N_block_erase(uint16_t block_nr);
3 uint8_t W25N_page_data_read(uint16_t page_addr);
4 uint8_t W25N_read_data(uint16_t col_addr, uint8_t * data,
      uint16_t nr_bytes);
5 uint8_t W25N_prog_data_load(uint16_t col_addr, uint8_t * data,
      uint16_t nr_bytes);
6 uint8_t W25N_rand_prog_data_load(uint16_t col_addr, uint8_t *
      data, uint16_t nr_bytes);
7 uint8_t W25N_prog_execute(uint16_t page_addr);
```

Isječak koda 4.5: Korisničke funkcije za *flash* memoriju

Inicijalizacija perifernih jedinica potrebnih za rad s *flash* memorijom, provjera ispravnosti SPI komunikacije izvršava se funkcijom `W25N_init()`. Brisanje bloka izvršava se funkcijom `W25N_block_erase()`. Prebacivanje odabrane stranice u međuspremnik *flash* memorije izvodi se pomoću funkcije `W25N_page_data_read()`, a nakon toga željeni niz okteta se može pročitati funkcijom `W25N__read_data()`. Funkcijama `W25N_prog_data_load()` i `W25N_rand_prog_data_load()` izvodi se upisivanje u međuspremnik. Funkcija `W25N_prog_execute()` upisuje podatke iz međuspremnika u stranicu u memoriji.

Funkcije koje još rade sa *flash* memorijom su one koje upravljaju neispravnim blokovima (isječak koda 4.6).

```
1 void W25N_scan_factory_BB(uint32_t * BBBT);
2 void W25N_bb_manage(uint16_t lba, uint16_t pba);
3 void W25N_read_bbm_lut(uint8_t * container);
```

Isječak koda 4.6: Funkcije za upravljanje neispravnim blokovima

Funkcija `W25N_scan_factory_BB()` skenira blokove u tablici bitova i označuje one koji su tvornički neispravni, a tablica bitova koju treba skenirati dobiva preko pokazivača `BBBT`. Funkcija `W25N_bb_manage()` omogućuje specificiranje zamjenskog bloka za blok koji je uočen kao neispravan. Funkcijom `W25N_read_bbm_lut()` se iščitava prozivna tablica (LUT), čime se dobivaju adrese neispravnih i zamjenskih blokova.

Zajedničko svim prethodnim funkcijama je korištenje funkcije prikazane u isječku koda 4.7.

```
1 void W25N_instruction_execute( W25N_instruction_nr instr, uint8_t
      *params, uint8_t *data_or_rbs, uint32_t nr_data_bytes);
```

Isječak koda 4.7: Funkcija za izvođenje instrukcije na *flash* memoriji

Funkcija `W25N_instruction_execute()` se izvršavaju instrukcije na flash memoriji i ona radi direktno s registrima SPI periferije. U ovoj funkciji su jedino nastajali problemi, a greška se događala kod upisa u podatkovni registar SPI periferiji. Na primjer, ako se želi poslati neki podatak preko SPI protokola, željeni podatak potrebno je upisati u registar SPI1_DR, a način na koji se je to radilo u prethodnoj programskoj prikazan je u isječku koda 4.8.

```
1 SPI1->DR = 0x69;
```

Isječak koda 4.8: Upis podatka u SPI2_DR registar

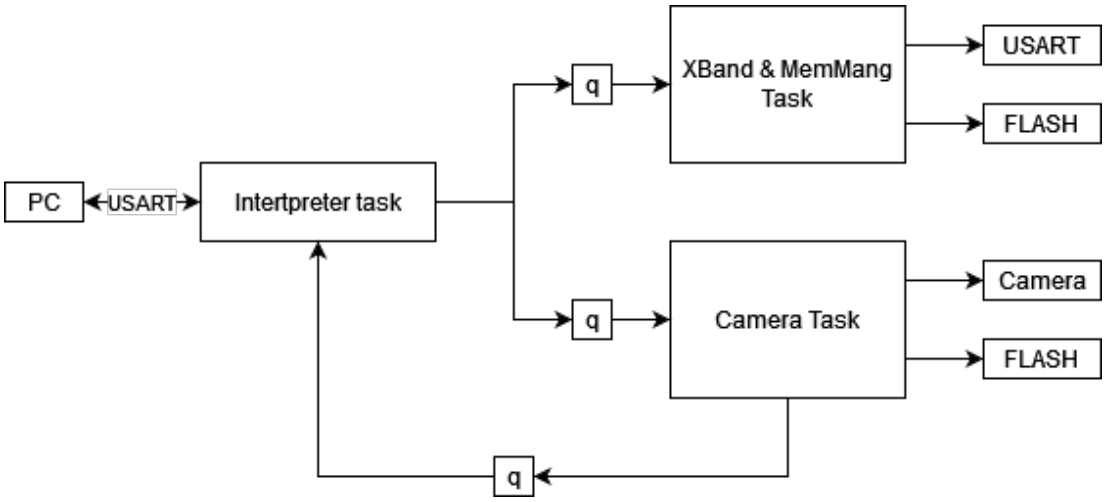
Naime, SPI komunikacija u ovom projektu radi s 8-bitnim podatcima, međutim, promatranjem linije kroz osciloskop, primijećeno je da se šalje 16-bitni podatak, s tim da se traženi podatak nalazi u gornja dva bajta. Ovaj problem riješen je korištenjem odgovarajuće LL funkcije, a u slučaju slanja podatka, primjer u isječku koda 4.8 prikazan je na ispravno napisan način u isječku koda 4.9.

```
1 LL_SPI_TransmitData8(SPI1, 0x69);
```

Isječak koda 4.9: Ispravan upis podatka u SPI2_DR registar

4.4. Integracija s operacijskim sustavom FreeRTOS

Za povezivanje razvijene programske potpore i ostvarivanja višezadaćnosti korišten je operacijski sustav za rad u stvarnom vremenu FreeRTOS. Razvijeni su zadaci *Camera Task* za upravljanje kamerom i *XBand & MemMang Task* za upravljanje memorijom i slanjem podataka preko USART sučelja na računalo u svrhu demonstracije. *Interpreter task* zadatak služi za upravljanje ostalim zadacima. Preko *Interpreter* zadatka zadaju se parametri koje će koristiti kamera (vrijeme ekspozicije, pojačanje i format) kao imena datoteka u koje će se spremiti slika (imena imaju nazive brojeva u intervalu od 0 do 1023). Preko *Interpreter* zadatka se također može odabrati datoteka za slanje preko USART sučelja, a željena datoteka se može i izbrisati. Također, ako je došlo do nekakvih problema, moguće je ponovno inicijalizirati datotečni sustav *flash* memorije. Prikaz zadataka u obliku blok dijagrama prikazan je na slici 4.1.



Slika 4.1: Zadaci u operacijskom sustavu PDH računala

Dio programa koji uzima i sprema sliku, kao i dio programa koji šalje podatke na USART sučelje, predstavljaju kritične sekcije, jer bi njihovo prekidanje moglo dovesti do grešaka pri upravljanju s podacima.

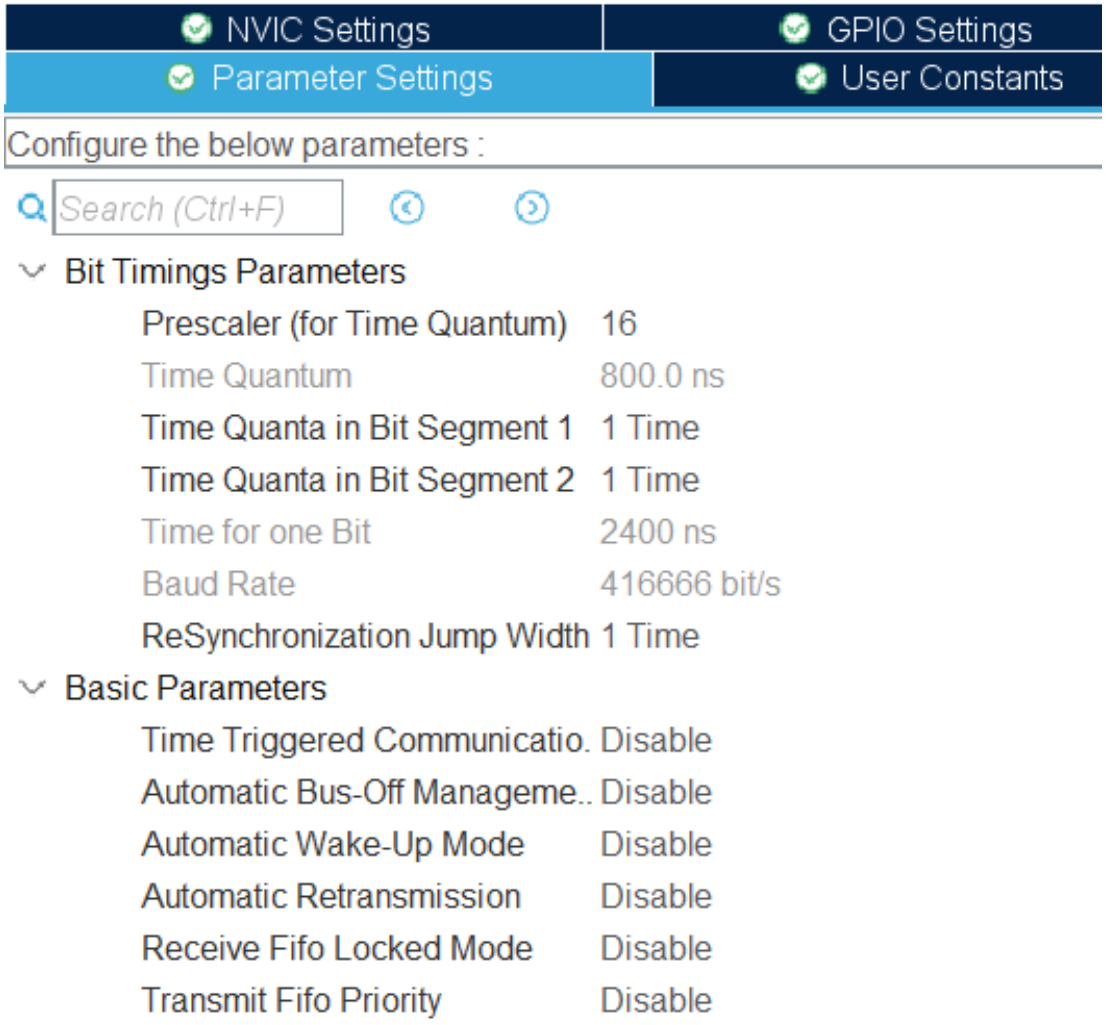
4.5. Programska podrška za CAN komunikaciju

S obzirom na neispravnost sklopa za CAN komunikaciju na tiskanoj pločici PDH sustava, programska podrška za CAN sučelje nije razvijena. Ipak, u ovom poglavlju dat će se pregled funkcija koje će se najčešće koristiti jednom kada se programska podrška počne razvijati. Za razliku od ostalih sučelja na STM32L471VGT6 mikrokontroleru

ne postoje LL biblioteke koje rukuju CAN periferijom, pa će se stoga koristiti HAL biblioteke.

4.5.1. Inicijalizacija

Konfiguracija CAN periferije može se izvesti korištenjem generatora koda u STM32CubeIDE razvojnom okruženju, a prikaz osnovne konfiguracije prikazan je na slici 4.2.



Slika 4.2: Konfiguracija CAN periferije

Generator koda će napraviti kod za inicijalizaciju CAN periferije implementiranjem `HAL_CAN_MspInit()` funkcije, u kojoj se omogućuje takt za CAN sučelje, podešavaju se stezaljke koje će CAN koristiti i omogućit će i podesiti prioritete prekida. CAN periferija će se inicijalizirati preko `HAL_CAN_Init()` funkcije koja koristi `HAL_CAN_MspInit()` funkciju za *low-level* inicijalizaciju.

Generator koda ne nudi opciju konfiguracije filtra, pa stoga korisnik mora sam implementirati funkciju za konfiguraciju filtra `HAL_CAN_ConfigFilter()`, čiji je prototip dan u isječku koda **??**.

```
1 HAL_StatusTypeDef HAL_CAN_ConfigFilter(CAN_HandleTypeDef *hcan,
    const CAN_FilterTypeDef *sFilterConfig)
```

Isječak koda 4.10: Prototip funkcije HAL_CAN_ConfigFilter()

Funkcija prima pokazivač na strukturu `*sFilterConfig`, koja sadržava željene parametre filtra. Kako bi se filter podesio, korisnik prvo treba napraviti strukturu tipa `CAN_FilterTypeDef` koja sadržava sve potrebne parametre filtra (isječak koda 4.11).

```
1  typedef struct
2  {
3    uint32_t StdId;
4    uint32_t ExtId;
5    uint32_t IDE;
6    uint32_t RTR;
7    uint32_t DLC;
8    FunctionalState TransmitGlobalTime;
9
10 } CAN_TxHeaderTypeDef;
```

Isječak koda 4.11: CAN_FilterTypeDef struktura

Kada se napravi inicijalizacija, CAN modul se uključuje `HAL_CAN_Start()` funkcijom. Čvor je sada aktivan na sabirnici i može primati i slati poruke.

4.5.2. Korištenje

Slanjem poruka se upravlja sljedećim funkcijama:

- `HAL_CAN_AddTxMessage()` - zahtjev za slanjem nove poruke,
- `HAL_CAN_AbortTxRequest()` - zahtjev za prekidom poruke u tijeku slanja,
- `HAL_CAN_GetTxMailboxesFreeLevel()` - vraća broj slobodnih odašiljačkih *mailbox* struktura,
- `HAL_CAN_IsTxMessagePending()` - provjerava da li je poruka u tijeku slanja

- `HAL_CAN_GetTxTimestamp()` - vraća vremensku oznaku poslane poruke, u slučaju da je omogućen način rada s vremenski okinutom komunikacijom.

Poruka primljena u FIFO strukturi se može pročitati `HAL_CAN_GetRxMessage()` funkcijom, broj poruka pohranjenih u FIFO strukturi se može dobiti preko funkcije `HAL_CAN_GetRxFifoFillLevel()`.

Primanje i slanje podataka se može izvoditi preko prekida, koji se mogu uključiti funkcijom `HAL_CAN_ActivateNotification()`. CAN periferija se može prebaciti u način rada s niskom potrošnjom energije uz pomoć `HAL_CAN_RequestSleep()` funkcije, a povratak u normalan način rada ostvaruje funkcija `HAL_CAN_WakeUp()`.

4.6. Učitavanje programa

Kako bi se programska podrška mogla učitati na mikrokontroler PDH sustava nužno je koristiti programator, u ovom slučaju ST-LINK/V2. Na raspolaganju je STM32F4DISCOVERY razvojni sustav koji ima ugrađen ST-LINK/V2 programator. Programator na razvojnom sustavu se može koristiti za programiranje mikrokontrolera na razvojnom sustavu ili programiranje mikrokontrolera na vanjskoj pločici. Koji mikrokontroler programator programira određuju prespojnici CN3 na razvojnom sustavu:

- ako su oba prespojnika spojena, programira se mikrokontroler na razvojnom sustavu,
- ako su oba prespojnika odspojena, programira se mikrokontroler na vanjskoj pločici.

Programiranje se izvodi preko CN2 konektora na razvojnom sustavu i X6 konektora na PDH sustavu. Funkcije stezaljki na pojedinim konektorima su dane u tablici 4.1.

Tablica 4.1: Opis stezaljki CN2 i X6 konektora [1], [4]

Redni broj stezaljke	CN2	X6
1.	VDD	VDD
2.	SWCLK	SWDIO
3.	GND	SWCLK
4.	SWDIO	SWO
5.	NRST	NRST
6.	SWO	GND

Potrebno je spojiti istoimene stezaljke kako bi se učitavanje programa uspješno izvelo. Osim za programiranje, razvojni sustav se koristi kako bi PDH sustav imao napajanje, s obzirom na to da sustav napajanja na PDH sustavu trenutačno nije ispravno.

5. Rezultati

Komunikacija između osobnog računala i PDH sustava u ovom slučaju predstavlja komunikaciju između CDH računala i PDH sustava. Komunikacija se izvodi preko USART sučelja, te je stoga potreban emulator terminala koji može koristiti UART komunikaciju. U tu svrhu odabran je program CoolTerm, koji može primati i slati podatke preko UART komunikacije i može spremati rezultate komunikacije, što je potrebno kako bi se na računalu mogla učitati slika. UART komunikacija radi na brzini prijenosa podataka od 115200 i ona predstavlja slanje podataka preko XBand sustava.

Kada se sustav uključi prvo se ispisuje rezultat inicijalizacije sustava, ako je inicijalizacija dobro prošla ispisuje se poruka `startup_ok`. Nakon toga nudi se izbor ponovne inicijalizacije datotečnog sustava, te se nudi izbor datoteke za slanje. Ako korisnik želi, može odmah nakon slanja datoteke odabranu datoteku izbrisati, a može i izbrisati datoteke bez prethodnog slanja. Korisnik može odabrati da se slanje datoteka preskoči. Sustav zatim traži od korisnika da koristi uobičajene, odnosne prethodne, postavke kamere ili da sam podesi postavke kamere ili da se korištenje kamere preskoči. Kada korisnik podesi sustav i odabere što želi raditi s njim, sustav ponudi korisniku pokretanje operacija PDH sustava. Kada sustav završi s izvršavanjem operacija ispisuje se rezultat operacija i status sustava, te sustav krene ispočetka i ponovno ponudi korisniku reinicijalizaciju datotečnog sustava. Primjer komunikacije između računala i PDH sustava dan je u isječku koda 5.1.

```
1 sustav: startup_ok
2 sustav: Type 'reinit' to reinitialize the filesystem, 'n' not to
3 korisnik: n
4 sustav: Press 'y' to select file for transmission, 'n' to skip
5 korisnik: n
6 sustav: Any [other] file you would like to delete?['y'/'n']
7 korisnik: n
8 sustav: Press 'y' to set camera params, 'd' to use default/
     previous, 'n' to skip
9 korisnik: y
```

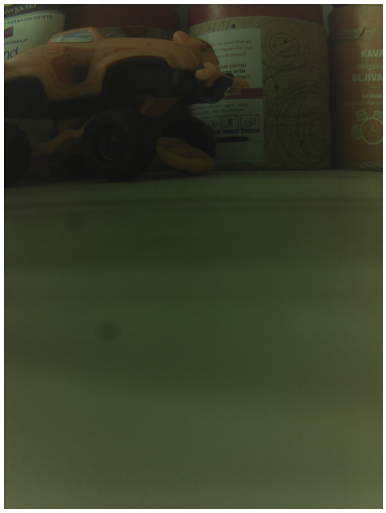
```
10 sustav: Set exposure nr_lines (uint16_t).
11 korisnik: 2000
12 sustav: Set exposure nr_lines frac (uint8_t).
13 korisnik: 10
14 sustav: Set gain (uint8_t), see acam.h.
15 korisnik: 9
16 sustav: Select format: 'r' for raw, 'j' for jpeg,
17 korisnik: j
18 sustav: Enter file name [0-1023] to store image measurments.
19 korisnik: 0
20 sustav: Press 's' to start PDH operations
21 korisni: s
22 sustav: Device camera status ok
23 sustav: Type 'reinit' to reinitialize the filesystem, 'n' not to
24 korisnik: n
25 sustav: Press 'y' to select file for transmission, 'n' to skip
26 korisnik: y
27 sustav: Enter file name [0-1023] of file to be transmitted via x-
       band.
28 korisnik: 0
29 sustav: Press 'y' to delete file after transmission, 'n' to keep
       it
30 korisnik: y
31 sustav: Any [other] file you would like to delete?['y'/'n']
32 korisnik: n
33 sustav: Press 'y' to set camera params, 'd' to use default/
       previous, 'n' to skip
34 korisnik: n
35 sustav: Press 's' to start PDH operations
36 korisnik: s
37 sustav: ÿÖÿ
38 ...
39 slanje slike
40 ...
41 ÿÜ Device xband status ok
42 sustav: Type 'reinit' to reinitialize the filesystem, 'n' not to
```

Isječak koda 5.1: Komunikacija između računala i PDH sustava

Ovisno o svjetlini okoline gdje se nalazi kamera preporučuje se vrijednost parametra `nr_lines` 2000 za relativno svjetlo područje i 5000 za relativno mračno područje. Za parametar `nr_lines frac` se preporučuje mala vrijednost, na primjer 10. Isto tako, male vrijednosti se preporučuju za parametar `gain`. Primjer slika koje su usli-

kane s navedenim parametrima u jpeg formatu prikazane su na slikama 5.2 i 5.1.

Slika 5.1: Slika ZESOI-knjižnice u relativno svjetlim uvjetima



Slika 5.2: Slika narančastog autića u relativno mračnim uvjetima

6. Zaključak

U ovom projektu opisan je razvoj programske podrške za upravljanje kamerom i *flash* memorijom nanosatelita CubeSat u sklopu projekta FERSAT. Kratko su opisani FERSAT projekt i dosadašnje aktivnosti u sklopu projekta. Opisana je arhitektura korisnog tereta FERSAT satelita i sklopovlje PDH sustava. Detaljno su objašnjeni I^2C, SPI i CAN protokoli i objašnjena je razlika između I^2C i SPI periferijskih sklopova prethodno korištenog (STM32F407VGT6) i novog (STM32L471VGT6) mikrokontrolera. Dan je pregled prilagodbe postojeće programske podrške za stari mikrokontroler na novi mikrokontroler i objašnjene su poteškoće do kojih je došlo kod prilagodbe. Objašnjen je način korištenja CAN periferije i dan je pregled funkcija koje će se koristiti za CAN komunikaciju. Programska potpora za upravljanje kamerom i *flash* memorijom je integrira u operacijski sustav za rad u stvarnom vremenu FreeRTOS. Predstavljeni su zadaci razvijeni u FreeRTOS operacijskom sustavu kojima se ostvaruje višezadaćnost PDH sustava i objašnjena je veza između zadataka. Objašnjen je način učitavanja razvijene programske podrške na PDH sustav. Predstavljeni su rezultati projekta i objašnjen je način upotrebe sustava, te su dane preporučene postavke za korištenje sustava.

Razvijenu programsku podršku moguće je nadograditi i poboljšati. Za primjer se uzima brzina takta SPI komunikacije između. Trenutačne brzine takta su male (1.25 Mbit/s za kameru i 625 kbit/s za *flash* memoriju) jer je kod većih brzina primijećeno izobličenje takta, što može stvarati probleme u komunikaciji. Moglo bi se iterativno povećavati brzina takta dok se ne dobije dovoljno velika brzina uz minimalno izobličenje signala takta.

Radi pogrešaka na tiskanoj pločici PDH sustava neke dijelove programske podrške nije bilo moguće razviti. Na primjer podsustav za CAN komunikaciju sadržava krivo spojene stezaljke za napajanje, a pogreške se nalaze i na samom podsustavu za napajanje PDH sustava, pa projekt u trenutačnom stanju ne može raditi bez osobnog računala i programatora. Kako bi se riješili ti problemi potrebno je napraviti novu ispravnu tiskanu pločicu, što je van okvira ovog projekta.

LITERATURA

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

[2] *UM10204 I^2C-bus specification and user manual*. NXP Semiconductors, 2021. Rev. 7.0.

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

[4] *UM1472 User manual Discovery kit for STM32F407/417 lines*. ST Microelectronics, 2014. Rev 4.

[5] *RM0090 Reference manual STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs*. ST Microelectronics, 2021. Rev. 9.

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

[7] *SLOA101B Introduction to the Controller Area Network (CAN)*. Texas Instruments, 2002. Revised May 2016.

[8] Wikipedia. I^2 c, 2022. URL `https://en.wikipedia.org/wiki/I%C2%B2C`. Preuzeto: 30. 05. 2022.

[9] Wikipedia. Serial peripheral interface, 2022. URL `https://en.wikipedia.org/wiki/Serial_Peripheral_Interface`. Preuzeto: 20. 06. 2022.

[10] FER ZKIST. FERSAT - opis projekta, 2022. URL https://www.fer.unizg.hr/zkist/FERSAT/projekt. Preuzeto: 18. 06. 2022.

Programska potpora za upravljanje kamerom na CubeSat nanosatelitu

Sažetak

U ovom projektu razvijena je programska podrška za upravljanje kamerom i *flash* memorijom na CubeSat nanosatelitu u sklopu projekta FERSAT. Dan je pregled aktivnosti projekta FERSAT koje su prethodile ovom projektu. Detaljno su opisane SPI i I^2C komunikacije pomoću kojih se upravlja kamerom i *flash* memorijom. Dana je usporedba između SPI i I^2C periferijskih sklopova STM32F407VGT6 i STM32L471VGT6 mikrokontrolera. Objašnjen je postupak prilagodbe programske podrške s prethodno korištenog (STM32F407VGT6) na trenutačno korišteni (STM32L471VGT6) mikrokontroler. Opisana je integracija razvijene programske podrške u FreeRTOS operacijski sustav za rad u stvarnom vremenu. Objašnjen je način upotrebe CAN periferije ne STM32L471VGT6 mikrokontroleru.

Ključne riječi: CubeSat, FERSAT, STM32, kamera, *flash* memorija, programska podrška, SPI, I^2C, FreeRTOS, CAN

Software for Camera Control on CubeSat Nanosatellite

Abstract

In this project, software support was developed for managing the camera and flash memory on the CubeSat nanosatellite as part of the FERSAT project. An overview of the activities of the FERSAT project that preceded this project was given. The SPI and I^2C communications used to control the camera and flash memory are described in detail. A comparison is given between the SPI and I^2C peripheral circuits of STM32F407VGT6 and STM32L471VGT6 microcontrollers. The procedure for adapting the firmware from the previously used (STM32F407VGT6) to the currently used (STM32L471VGT6) microcontroller is explained. The integration of the developed software support into the FreeRTOS operating system for real-time operation is described. The method of using the CAN peripherals for the STM32L471VGT6 microcontroller is explained.

Keywords: CubeSat, FERSAT, STM32, camera, flash memory, software support, SPI, I^2C, FreeRTOS, CAN