UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

UROŠ JERMAN

Varna in zanesljiva programska oprema elektronskega števca električne energije MT372-SMART

UNIVERZITETNO DIPLOMSKO DELO

Mentor: doc. dr. Boštjan Murovec

Ljubljana, maj 2008

Zahvala

Na tem mestu bi se najprej zahvalil družini, prijateljem in kolegom, ki so mi pomagali in stali ob strani v času celotnega študija. Še posebej se zahvaljujem mentorju, doc. dr. Boštjanu Murovcu, ki me je usmerjal pri izdelavi diplomske naloge in požrtvovalno pomagal z nasveti in pristopi k reševanju. Prav tako se zahvaljujem mentorju v Iskraemecu, mag. Nenadu Međeralu, ki mi je predstavil zanimiv problem in pomagal z nasveti.

Povzetek

Elektronski števci električne energije se v zadnjem času močno razvijajo. Zgolj merjenju električne energije se pridružujejo nove funkcionalnosti in zahteve. Jedro števca tako predstavlja mikrokrmilnik, ki nadzira delovanje vseh ostalih komponent. Zaradi potreb po čedalje točnejših in obsežnejših meritvah je bilo v Iskraemecu razvito novo merilno vezje, za katerega smo v diplomski nalogi razvili gonilnik in pripadajočo merilno platformo.

Metoda analize možnih odpovedi in njihovih posledic – FMEA je splošno priznana metoda za analizo strojne opreme, medtem ko se jo v zadnjem času uporablja tudi pri analizi programske opreme (SWFMEA). Ker smo pozornost posvetili razvoju varne in zanesljive programske opreme, smo izvedli analizo le-te po omenjeni metodi. Prepoznali smo šest najnevarnejših hazardov in jih ovrednotili ter omejili. Razvili smo svojstven postopek za računanje števila RPN. Pri razvoju programske opreme upoštevamo tudi pravila obrambnega programiranja, na podlagi katerih smo razvili nadzorni seznam, ki služi preverjanju postopkov pri programiranju.

Analiza merilnih rezultatov kaže, da nova programska oprema merilne platforme deluje pravilno, saj je števec prestal teste narejene v skladu s standardom IEC 62053-21.

Ključne besede:
Varno programiranje, števec električne energije, SWFMEA, obrambno programiranje, hazard, Calculator SO7D, Smart Sensor

Abstract

The electronic meters for electrical energy have drastically changed over the years. Previously meters had only measured electrical energy, but today they are accompanied with many new features as well as demands. The core of the electronic meter is represented by microcontroller, which monitors all other components. Due to increasing demands over more accurate and comprehensive measurements Iskraemeco, d.d., has developed new integrated circuit for measuring electrical energy. In this diploma new driver and supporting measurement platform with application interface for new integrated circuit has been build.

Failure mode and effects analysis – FMEA is widely used in engineering hardware systems to help in understanding the effects of potential failures and the faults that cause them to occur. Interest in applying the technique to software has increased in recent years. Therefore SWFMEA has been performed due to focus on development of safe and reliable firmware. Six different hazards have been identified and mitigated. The unique procedure for calculating risk priority number alias RPN has been introduced. Furthermore internal coding standards have been conformed to widely used techniques such as defensive programming.
The analysis of measurement data shows that the firmware has performed well, as the meter has withstood all tests implemented according to IEC 62053-21 standard.

Key words:
Safe programming, electronic meter, SWFMEA, defensive programming, hazard, Calculator SO7D, Smart Sensor

Kazalo

Kazalo tabel

Kazalo slik

Seznam akronimov in izrazov

AMI	(ang.: Advanced Metering Infrastructure) napredna infrastruktura merilnih naprav
AMR	(ang.: Automatic Meter Reading) avtomatsko branje merilnih naprav
ANSI	(ang.: American Standard Code for Information Interchange) ameriški standard za zapis različnih znakov
API	(ang.: Application Programming Interface) aplikacijski programski vmesnik
ASIC	(ang.: Application Specific Integrated Circuit) integrirano vezje izdelano po naročilu
BCPL	(ang.: Basic Combined Programming Language) programski jezik, predhodnik programskega jezika C
C89/C90	ANSI and IEC standard on C) standard za programski jezik C
CCCC	(ang.: C and C++ Code Counter) orodje za analizo C-datotek
CRC	(ang.: cyclic redundancy check) ciklično preverjanje redundance
FMEA	(ang.: Failure Mode and Effects Analysis) analiza možnih odpovedi in njihovih posledic
GPIO	(ang.: General Purpose Input Output) splošno namenska vhodno/izhodna vrata
ICE	(ang.: In-Circuit Emulator) emulator
IEC	(ang.: International Electrotechnical Commission) mednarodna elektrotehniška komisija
OTP	(ang.: One Time Programmable) enkrat programabilne spominske celice
PHA	(ang.: Preliminary Hazard Analysis) predhodna analiza hazarda
RPN	(ang.: Risk Priority Number) število prioritete tveganj
RTC	(ang.: Real Time Clock) ura realnega časa
SPI	(ang.: Serial Peripheral Interface) serijsko vodilo, za komunikacijo med različnimi komponentami
SWFMEA	(ang.: Software Failure Mode and Effects Analysis) analiza možnih odpovedi in njihovih posledic programske opreme
UART	(ang.: Universal Asynchronous Receiver Transmitter) univerzalni asinhroni sprejemnik in oddajnik

1 Uvod

Električna merjenja sežejo daleč v zgodovino človeštva. Stari Grki so opazovali in beležili elektrostatični učinek, medtem ko so Kitajci z magnetnim kompasom izdelali prvi električni instrument. Naslednji nepogrešljivi koraki so nastali v 18. in 19. stoletju, ko so velikani elektrotehniške stroke (C. Coulomb, A. Volta, A. M. Ampere, M. Faraday, J. C. Maxwell) na podlagi merjenj in iz njih izpeljanih zaključkov odkrivali temelje elektrotehniške znanosti [1, str. 2].

Merjenja so postajala čedalje kompleksnejša in zahtevnejša. Ko se je pojavila električna javna razsvetljava, se je pokazala potreba po prenosu meritev iz znanstvenih v komercialne kroge. Tako je E. Thompson v ZDA leta 1889 izdelal prvi pravi števec električne energije, ki so ga uporabljali na različnih gospodarskih področjih [2].

Iskraemeco, d.d., Kranj, se je pridružila proizvajalcem števcev električne energije leta 1948, ko je stekla proizvodnja prvega enofaznega števca. Desetletje kasneje je nastal prvi trifazni števec. Ostali mejniki v razvoju elektromehanskih in elektronskih števcev so bili [3, 4]:

- 1975 leta, precizijski števci (0,5S, 0,2S),
- 1988 leta, AMR (ang.: advanced meter reading) industrijski števci,
- 1992 leta, elektronski monolitni gospodinjski števci ter
- 1998 leta, prvi AMR sistem.

Današnja generacija elektronskih monolitnih števcev električne energije sodi v kategorijo pametnih števcev (ang.: smart meters) in so del tako imenovane napredne merilne infrastrukture, ki zaobjema števce, zbirne centre, ki zbirajo podatke večjega števila števcev, in nadzorne sisteme, ki omogočajo pregledno in temeljito spremljanje vseh parametrov merjenja in iz tega izpeljanih storitev (ang.: AMI – Advanced Metering Infrastructure).

Na sliki 1 prikazujemo razvoj števcev, od indukcijskih do števcev AMI.

Slika 1: trifazni indukcijski števec T3 (levo) in trifazni elektronski števec MT372 (desno)

Sodobni pametni števci električne energije vsebujejo poleg analognih tudi digitalne komponente, s katerimi je izveden pester nabor dodatnih funkcij. Poleg preprostega merjenja in beleženja porabljene energije lahko tovrstne števce električne energije odčitavamo daljinsko, nastavljamo večje število tarif, hranimo podatke drugih merilnikov (poraba vode, plina, ...), nastavljamo največjo dovoljeno porabo moči ali energije in ob prekoračitvi le-teh uporabnika izklopimo, omogočamo predplačniške funkcije in nadziramo kakovost omrežja.

Vendar potrebujejo dodatne funkcionalnosti za pravilno in brezhibno delovanje zanesljive, točne in natančne podatke o merjenih veličinah. S tem namenom je bil v Iskraemecu, d.d., razvit nov merilni sistem, ki je sestavljen iz dveh komponent – pametnih tipal (ang.: smart sensor) ter osrednjega merilnega integriranega vezja SO7D (ang.: Calculator SO7D) [21]. Pametno tipalo meri električno napetost in tok ter posreduje digitalne podatke primerne za nadaljnjo obdelavo vezju SO7D, ki jih obdela in preko vodila SPI (ang.: serial peripheral interface) posreduje merilnemu gonilniku.

Z uvajanjem mikrokrmilnika in integriranega vezja SO7D se je poleg negotovosti pravilnega delovanja analognih komponent pojavila tudi negotovost pri izvrševanju opravil digitalnih komponent. Opravila izvršujemo s pomočjo vgradne programske opreme (ang.: firmware), ki prevzema vedno večjo odgovornost za varno in zanesljivo delovanje sistema. Pri načrtovanju programske opreme skušamo ugotoviti in upoštevati vsa potencialno nevarna stanja sistema, česar se ne lotimo brezglavo, ampak v ta namen razvijemo sistematični postopek na podlagi analitičnih metod za odkrivanje možnih odpovedi in ovrednotenje njihovih posledic.

V diplomski nalogi smo si zadali cilj načrtati in razviti varno ter zanesljivo programsko opremo merilnega dela trifaznih pametnih števcev električne energije MT372-SMART ter umestitev le-te v celotno programsko arhitekturo števca. Programska oprema sestoji iz:

- gonilnika, ki skrbi za komunikacijo med mikrokrmilnikom in merilnim vezjem,
- podsistemom, ki obdela dobljene podatke in
- podsistemom, ki omogoča dostop do podatkov preko programskega vmesnika (ang.: API, Application Programming Interface).

Pri razvoju programske opreme smo razčlenili in z ustreznimi prijemi zmanjšali načrtovalske in programerske napake ter morebitne motnje na vodilu med merilnim vezjem in mikrokrmilnikom.

V nadaljevanju opisujemo splošne postopke pri izgradnji varnega vgradnega sistema (ang.: embedded system) in v tretjem poglavju razčlenimo njegov celoten iterativni življenjski cikel, od zasnove, preko analize zahtev in načrtovalske faze ter faze razvoja programske opreme, do testiranja in predaje sistema naročniku. Ker je analiza celotnega življenjskega cikla sistema obsežno delo, se osredotočamo zgolj na načrtovanje in izvedbo programske opreme.

Iz množice metod, ki pripomorejo k zanesljivi in varni programski opremi, izpostavimo naslednji dve:

- metodo predhodne analize hazardov (ang.: Preliminary Hazard Analysis, PHA) ter
- metodo analize možnih odpovedi in njihovih posledic (ang.: Failure Mode and Effect Analysis, FMEA) in njeno izpeljanko za programsko opremo (ang.: Software Failure Mode and Effect Analysis, SWFMEA)

Obe metodi sta opisani v nadaljevanju. Z njima odkrivamo in odpravljamo potencialno nevarna stanja, ki lahko privedejo do odpovedi sistema, zato sta metodi umeščeni v življenjski cikel le-tega.

Za izvedbo programskega dela števca smo uporabili programski jezik C, zato v poglavjih 6 in 7 analiziramo njegove pomembnejše lastnosti in pasti, ki jih moramo upoštevati pri razvoju zanesljivih in varnih aplikacij. Podan je tudi nadzorni seznam (ang.: checklist) vseh poudarkov, ki jih velja upoštevati. V 9 poglavju prikazujemo koncept celotne programske opreme pametnega števca električne energije Ker pri vgradnih sistemih velja, da sta

programska in strojna oprema med seboj tesno povezani, opisujemo tudi slednjo. Zaključujemo z opisom razvoja in analizo programske opreme merilnega dela in prikazom dobljenih rezultatov.

2 Splošno o načrtovanju varne programske opreme

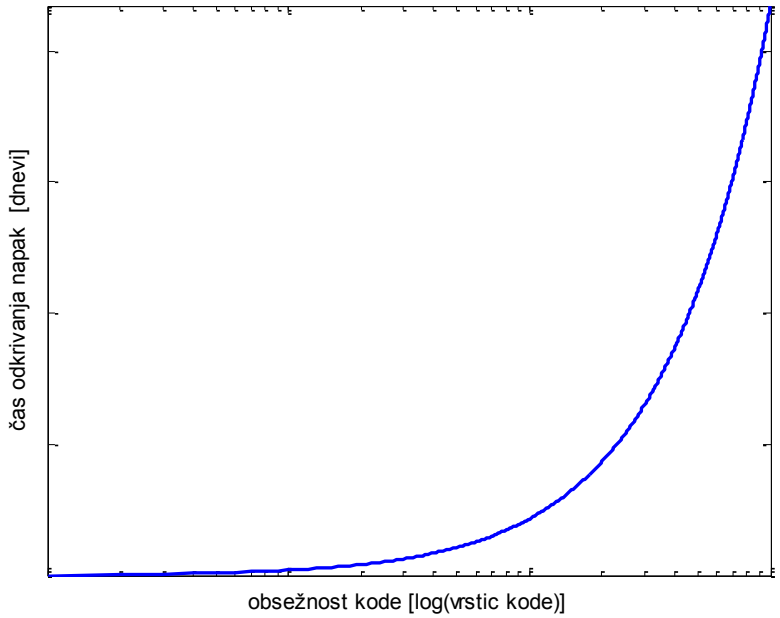
Z razvojem vgradnih sistemov in čedalje kompleksnejšo programsko opremo se je povečal pritisk na programerje. Izdelki se razvijajo v čedalje krajših časovnih intervalih in često vsebujejo kompleksne funkcije, pri tem pa se od razvijalcev zahteva, da rezultirajoča programska oprema ne izkazuje napak pri delovanju.

V takih pogojih je ključno natančno prepoznavanje situacij, ki vodijo k napakam in posledično v odpoved sistema. Okoliščine, ki lahko vplivajo na nepravilno izvajanje programske opreme so [6]:

- različne kombinacije parametrov na vhodu v sistem, ki vodijo k logičnim napakam,
- vpliv različnih motenj na strojno opremo in
- logične napake pri podporni programski opremi (prevajalnikih).

Vendar načrtovalci in programerji niso nezmotljivi in niso vse okoliščine predvidljive. Nekatere vhodne kombinacije parametrov ali napake prevajalnika lahko povzročijo napake v sistemu in posledično njegovo odpoved. Pri pisanju programske opreme velja, da pravilno delovanje sistema v 90 % vseh možnih situacijah ali življenjske dobe pomeni neuporaben izdelek, v 99,9 % pa izdelek s hudimi pomanjkljivostmi. Kompleksnost in velikost kode prispeva h količini skritih napak. Ocenjujejo, da se v 100.000 vrsticah kode skriva okoli 100 napak, kar predstavlja 0,1 % nepravilnost v delovanju sistema [5, str. 55].

Prav tako je programska oprema visoko entropična. Njena neurejenost se z naraščanjem števila vrstic kode povečuje, kar se odraža v radikalnem povečevanju truda in stroškov pri izogibanju različnim napakam in pastem [5, str. 55]. Ocenjeno odvisnost med kompleksnostjo kode in trudom, potrebnim za odkrivanje napak, prikazuje naslednja slika. Na abscisi je podana kompleksnost kode v logaritmu števila programskih vrstic, medtem ko je na ordinatno os nanešen v dneh izražen čas, potreben za odkrivanje in odpravljanje napak.



Slika 2: ocenjeni čas odkrivana napak v odvisnosti od obsežnosti kode

V nadaljevanju definirajmo napake, zmote in odpovedi.

Napaka je vzrok za potencialno nevarno stanje in odpoved sistema. Do napak lahko pride zaradi pomanjkljivosti v specifikacijah, načrtovanju, človeških napak ali fizičnih okvar.

Zmota je pomanjkljivost v načrtovanju ali pa odklon od želenih oziroma predvidenih stanj sistema.

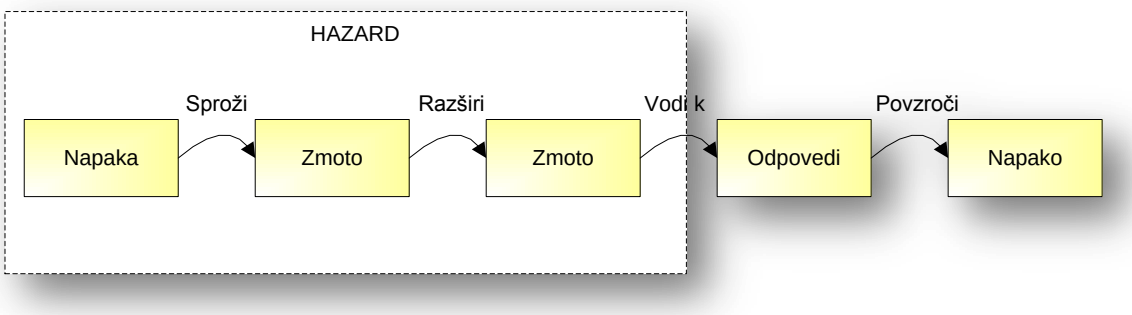
Odpoved je stanje, ko sistem ne deluje po predpisanih zahtevah [17].

Ob uporabi metod za analizo možnih odpovedi in njihovih posledic nato določimo preventivne ukrepe in postopke za odpravo identificiranih napak.

Če načrtujemo vgradni sistem, čigar odpoved lahko pomeni katastrofično stanje (smrt ali poškodbe ljudi, veliko ekonomsko škodo), moramo zagotoviti varno delovanje tudi v primeru odpovedi. Programska oprema naj prispeva k varnemu in funkcionalnemu izvajanju celotnega sistema, za kar je predpogoj, da v začetni fazi razvoja definiramo vse možne hazarde.

Hazard je prisotnost potencialno nevarnega stanja, čigar izid lahko prispeva k odpovedi sistema. Vsak hazard ima vsaj en vzrok nastanka in lahko vodi k številnim nezaželenim posledicam. Tipičen vzrok hazarda je napaka ali okvara strojne oziroma programske opreme. Pri slednji odpovedi izvirajo iz njene arhitekture, medtem ko pri prvi nastanejo zaradi različnih fizikalnih dejavnikov. Za vsak hazard potrebujemo ustrezen postopek, ki ga odpravi. Nadziranje hazarda je postopek, s katerim preprečimo ali zmanjšamo verjetnost nevarnega stanja, tako da prilagodimo strojno ali programsko opremo. Slednjo prilagodimo tako, da ob odpovedi strojne opreme varno zaključi delovanje celotnega sistema. [8, str. 17].

Možnosti za nastanek napak so velike, medtem ko je zmožnost njihovega predvidevanja omejena, zato skoraj vsaka napaka sproži zmoto, ki vodi k odpovedi sistema. Le-ta pa ponovno povzroči napako. Na sliki 3 prikazujemo vzročno odvisnost med napakami, zmotami in odpovedmi ter povezanost le-teh s hazardom.



Slika 3: Vzročna povezanost med napako zmoto in odpovedjo

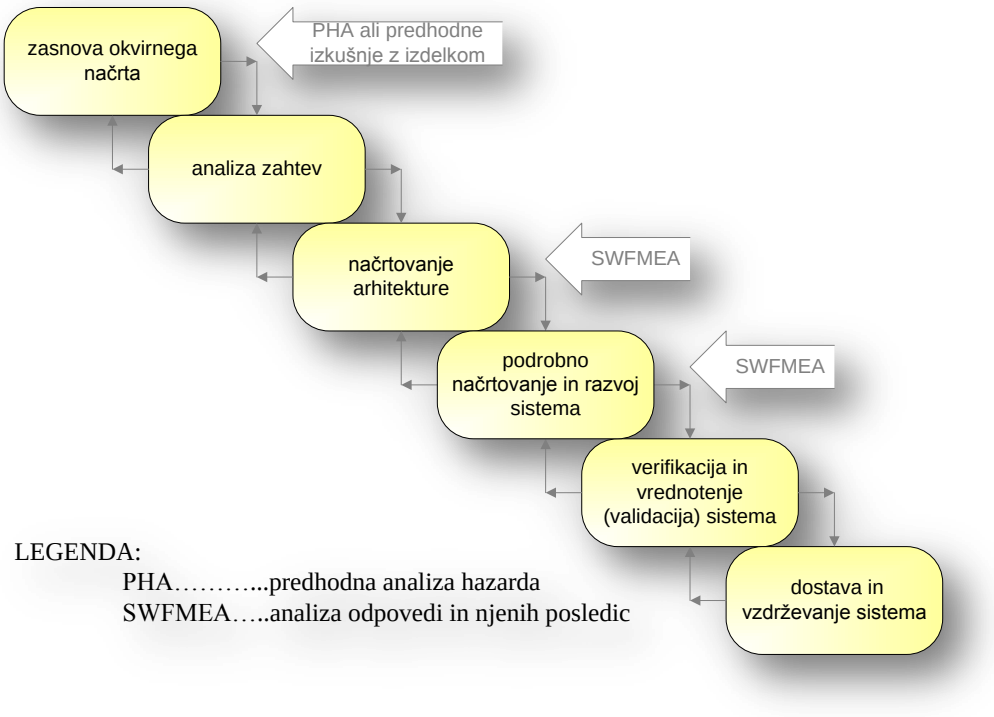
Pri varnem načrtovanju programske opreme je ključnega pomena sistematično beleženje vsakega koraka v procesu razvoja sistema. Za boljše sledenje nastajanja sistema razdelimo postopek izgradnje na več medseboj povezanih modulov. To imenujemo življenjski cikel sistema.

3 Življenjski cikel sistema

Razvoj in vzdrževanje strojne ter programske opreme tvorita življenjski cikel vgradnega sistema. Življenjski cikel sistema ne začnemo z *ad-hoc* pisanjem kode ali načrtovanjem strojne opreme, temveč po predhodnem nastavku koncepta in vzpostavitvi temeljnih zahtev. Prav tako se življenjski cikel sistema ne konča z napisano zadnjo vrstico kode ali vključitvijo vseh strojnih komponent, ampak se zaključi, ko se izdelek umakne iz prodaje in ga ni potrebno več vzdrževati in nadgrajevati. Celoten proces se ne odvija linearno, temveč ga odlikuje mnogo iteracij in popravkov, s katerimi pridemo do želenih rezultatov. Faze življenjskega cikla delimo na:

- zasnova okvirnega načrta,
- analizo zahtev sistema,
- načrtovanje arhitekture,
- podrobno načrtovanje in razvoj sistema,
- verifikacija in vrednotenje (validacija) sistema ter
- dostava in vzdrževanje sistema.

Na Sliki 4 prikazujemo razčlenitev celotnega življenjskega cikla na posamezne faze z možnimi iteracijami. Prikazano je tudi, kdaj je smiselno uporabiti katero izmed metod, ki omogočajo načrtovanje varne programske opreme [6]. Omenjeni metodi PHA in SWFMEA sta podrobneje predstavljeni v poglavjih 4 in 5.



Slika 4: življenjski cikel sistema in vstopne točke analitičnih metod

3.1 Zasnova okvirnega načrta

Pri razvoju izdelka najprej definiramo problem, ki ga želimo rešiti, da dobimo preprosto definicijo problema s stališča uporabnika [7, str. 12]. Prav tako dorečemo morebitne dodatne striktnejše zahteve po razvoju varnega sistema. Na to vprašanje odgovorimo s predhodno analizo hazarda ali na podlagi izkušenj s predhodnimi podobnimi izdelki [6].

3.2 Analiza zahtev sistema

Tu naredimo dokument zahtev oziroma specifikacij, ki nakazuje rešitev problema. V tem delu ne podamo konkretnih metod, temveč osnovna vodila in omejitve za reševanje problema. Izdelani dokument uporabimo tudi za načrtovanje testnega postopka, ki ga razvijamo skupaj z rešitvijo problema. [7, str. 12].

Če smo v zasnovi okvirnega načrta dognali, da potrebujemo načrt za izvedbo varnega sistema, potem opredelimo zahteve, ki odpravijo ali zmanjšajo potencialne hazarde. Včasih so te zahteve že opredeljene v standardih, državnih predpisih ali kupčevih specifikacijah. V tem delu lahko uporabimo različne metode za analizo hazarda.

Dva najpogostejša pristopa, ki jih uporabimo pri predhodno ugotovljenih hazardnih stanjih, sta: analiza od zgoraj navzdol (ang.: top down analysis) in analiza od spodaj navzgor (ang.: bottom up analysis). Pri prvem pristopu iz znanih hazardnih stanj razvijamo drevesno strukturo do najnižjih strojnih (različni ventili, tipala, …) in programskih (funkcijskih blokov) komponent. Pri drugem pristopu pa razvijamo drevesno strukturo od najnižjih komponent preko višjih slojev do ugotovljenih hazardov. Pri tem si pomagamo z metodo SWFMEA. Oba pristopa pokažeta tiste komponente, ki lahko povzročijo hazard, vendar analiza od spodaj navzgor v tej fazi ni vedno mogoča, saj navadno ne poznamo najnižjih funkcij ter komponent sistema in zato to analizo ponavadi naredimo v poznejših fazah življenjskega cikla [8, str. 57].

3.3 Načrtovanje arhitekture sistema

Ta del cikla je najvišji sloj načrtovalskega procesa. Razdeljen je na razvoj funkcijskih sklopov in namenske tehnologije. Pri vgradnih sistemih moramo opredeliti sledeče stvari:

- uporabo strojne ali programske opreme,
- izbiro mikrokrmilnika,
- izbiro programskega jezika,
- strojne in programske module ter
- odločitev o uporabi kupljenih rešitev ali lastnem razvoju.

Uporaba strojne ali programske opreme ima odločilen vpliv na celotno načrtovanje, zato moramo v tem delu določiti tehnologije, ki jih uporabljamo za posamezne funkcijske sklope. Glede na podane ugotovitve se moramo odločiti, ali bomo uporabili programsko, strojno ali obe opremi hkrati. Tipičen primer je programska realizacija univerzalnega asinhronega sprejemnika in oddajnika (ang.: Universal Asynchronous Receiver/Transmitter, v nadaljevanju UART), zaradi zasedenosti ali pomanjkanja strojne izvedbe le-tega. Prav tako se lahko zaradi cene določene strojne komponente, in če nam mikrokrmilnik to omogoča, odločimo za programsko realizacijo ekvivalentne funkcije. Zavedati se moramo, da strojna oprema predstavlja ponavljajoč strošek (z vsakim izdelkom), medtem ko programska oprema predstavlja neponavljajoč strošek. Vendar je začetna investicija pri programski opremi višja.

Pri *izbiri mikrokrmilnika* najprej preverimo možne družine mikrokrmilnikov, nato zožimo izbiro na tiste, ki se najbolj prilegajo našim zahtevam. Pozorni smo na ustrezno število splošno namenskih vrat (ang.: General Purpose Input and Output, v nadaljevanju GPIO), časovnikov, analogno digitalnih pretvornikov in perifernih enot, kot so časovni čuvaj

(ang.:watchdog timer) in vodili SPI ter I^2C. Pri izbiri sta pomembna tudi urni takt mikrokrmilnika in velikost integriranega delovnega (ang.: RAM, Random Access Memory) ter bliskovnega (ang.: flash) pomnilnika. Na koncu preverimo ceno in dobavljivost.

Izbira programskega jezika za vgradne sisteme se nagiba med dvema jezikoma. Prvi je programski jezik C, drugi je zbirni jezik. Ko se odločamo, katerega bomo uporabili, moramo imeti v mislih predvsem prenosljivost, zanesljivost in berljivost programske opreme na eni strani ter hitrost delovanja in velikost nastale programske opreme na drugi [7, str. 19].

Strojne in programske module izberemo glede na periferijo, ki jo izbrani mikrokrmilnik ima oziroma nima, ali jo ima za naše potrebe premalo (npr. uro realnega časa, dodatni bliskovni pomnilnik, zunanji časovni čuvaj). Izbrano periferijo in mikrokrmilnik napajamo preko ustreznega napajalnika. Pri programski opremi zasnujemo važnejše algoritme in začrtamo uporabljene podatkovne strukture.

Odločitev o uporabi kupljenih rešitev ali lastnem razvoju je izredno pomembna. Lahko odloča o tem, ali bo projekt uspel ali ne. Z nakupom določenih podsestavov se izognemo mučnemu in dolgotrajnemu lastnemu razvoju. Podsestave programske in strojne opreme, ki jih ponavadi kupujemo, so: operacijski sistemi dejanskega časa, mrežni gonilniki, matematične funkcije, napajalniki in gonilniki za prikazovalnike. Vendar imajo kupljeni podsestavi tudi šibke točke in lahko se zgodi, da se izkažejo neustrezni za naš projekt. Morebiti nimamo dostopa do izvorne kode, s čimer smo odvisni od pogodbenika, da nam pregleduje in vzdržuje dotične podsklope. Če imamo s pogodbeniki različne prioritete pri zagotavljanju ažurnosti, se zaradi tega lahko celoten projekt zavleče [7, str.20]. S stališča varnega programiranja moramo biti v tem delu cikla pozorni na:

- modularnosti in
- sledljivosti (ang.: traceability).

Modularnost pomeni, da razdelimo vsebinsko podobne funkcijske bloke v zaključene celote. Tako lahko izpostavimo tiste module, ki so kritični s stališča varnosti. Njihovo število in prepletenost (ang.: coupling) morata biti zmanjšana na neko razumno raven (ang.: As Low As Reasonably Possible, v nadaljevanju ALARP).

Sledljivost nam veli, da moramo dosledno upoštevati in dokumentirati vse zapisane točke iz faze zahtev [8, str. 70].

Z analizo dobljenih podatkov iz predhodnega dela cikla določimo tiste funkcijske sklope, ki jih bomo združili v vsebinsko povezane module in nad katerimi bomo opravili analizo z metodo SWFMEA [6]. Opredelimo tudi interni standard kodiranja (ang.: coding standard), s katerim določimo smernice za razvoj programske opreme, kot so uporabljene programske knjižnice in stil pisanja kode (komentarji, oblika, ...). Iz dobljenega standarda izvlečemo nadzorni seznam, s katerim na koncu preverimo ali smo upoštevali vse vidike standarda.

3.4 Podrobno načrtovanje in razvoj sistema

V tem delu podrobno razčlenimo funkcijske bloke posameznih modulov in jih dokončno opredelimo. Najprej razvijemo programsko jedro mikrokrmilnika, nato pa se lotimo posameznih modulov. Če v prejšnjih fazah uporabljamo psevdokodo, jo sedaj zamenjamo z dejansko kodo uporabljenega programskega jezika. Pri tem se lahko poslužimo tudi obrambnega programiranja (ang.: defensive programming), kjer ločimo kritične funkcije od nekritičnih, striktneje preverjamo vhodne parametre, uvajamo redundantne indikatorje stanj in podobno.

Z metodo SWFMEA podrobno analiziramo načrtovanje in nato samo izvedbo sistema. Za kvantitativno ovrednotenje hazarda nam je na voljo ocenitev kompleksnosti kode (ang.: measurement of complexity). Ko smo zadovoljni z oceno sistema, preidemo v naslednjo fazo življenjskega cikla.

3.5 Verifikacija in vrednotenje (validacija) sistema

Najprej naredimo vrednotenje sistema, saj uspešna opravljenost le-tega nakazuje visoko stopnjo gotovosti, da bo sistem deloval znotraj podanih karakteristik. Nato sledijo različni verifikacijski testi, od katerih so najpomembnejši sistemski testi, ki jih delimo na:

- funkcijske teste (ang.: functional testing),
- stresne teste (ang.: stress test),
- teste stabilnosti (ang.: stability test),
- teste učinkovitosti (ang.: performance test).

Pri *funkcijskem testu* opravimo celoten pregled sistema pri nazivnih pogojih. Če funkcijski bloki opravljajo natanko predpisana opravila, ki so bila določena v specifikacijah, jih verificiramo [8, str. 94].

Pri *stresnem testu* želimo videti, koliko sistem prenese, preden odpove. Bodisi krajšamo čase pri komunikaciji med moduli oziroma funkcijskimi bloki, bodisi obremenjujemo centralno procesno enoto [8, str. 94].

Pri *testu stabilnosti* pustimo sistem nemoteno delovati daljše časovno obdobje. Pri tem smo pozorni na periodično pojavljanje pokvarjenih podatkov, občutljivost na različna sosledja dogodkov in puščanje pomnilnika (ang.: memory leakage) [8, str. 94].

Pri *testu učinkovitosti* preverjamo točnost algoritmov, največjo kapaciteto vpisov v posamezni medij, odzivne čase na različne dogodke, dostopnost vseh modulov in prepustnost, ki je definirana kot največje število dogodkov, ki jih sistem lahko obdela na časovno enoto [8, str. 94].

Ob manjših dodatnih posegih oziroma popravkih moramo sistem ponovno verificirati tako, da teste opravimo še enkrat (ang.: regression testing). Vendar pa smo večinokrat časovno omejeni, zato testiramo samo tiste sklope, ki smo jih spreminjali in tiste, kjer lahko pričakujemo nove napake (ang.: minimization) [8, str. 94].

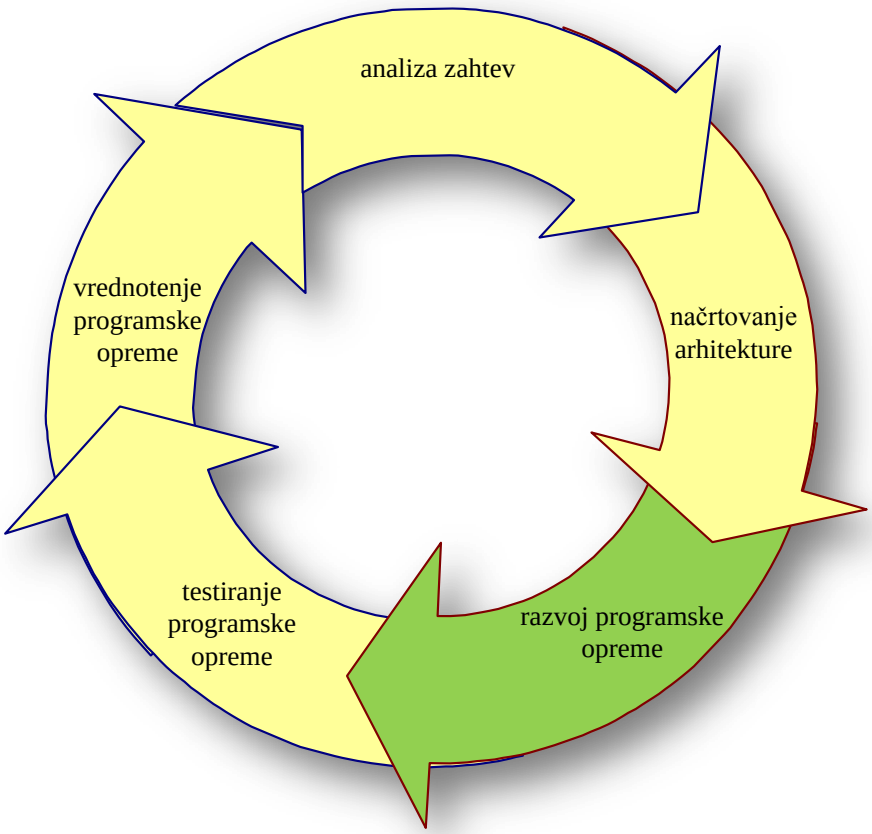
3.6 Dostava in vzdrževanje sistema

Ob dostavi sistema moramo zagotoviti ustrezno dokumentacijo. Vzdrževanje sistema poteka na nivoju strojne in programske opreme. Zaradi staranja ali enkratnih oziroma ponavljajočih stresnih dogodkov moramo strojno opremo po določenem času zamenjati. Ker se programska oprema ne stara, so razlogi za vzdrževanje drugačni:

- popravki na terenu odkritih napak,
- dodane nove funkcionalnosti,
- programska kompenzacija zaradi staranja strojne opreme ali različnih odpovedi.

3.7 Implementacija življenjskega cikla

Analiza celotnega življenjskega cikla sistema je preobsežna in sega preko okvirov diplomskega dela. Osredotočamo se zgolj na načrtovanje programske opreme merilnega dela števcev, pri čemer smo povzeli prikazano shemo na sliki 5 [5, str. 132]. Z zeleno barvo je obarvana faza, ki je v diplomski nalogi najtemeljiteje obdelana in pri kateri smo uporabili metodo SWFMEA. Ostale faze so prav tako predstavljene v nadaljevanju.



Slika 5: življenjski cikel razvoja programske opreme merilnega dela števcev

4 Metoda PHA

Z metodo PHA opredelimo hazarde v začetnih fazah življenjskega cikla sistema. Kvantitativne ocene o resnosti odpovedi uporabimo kot vhodni parameter pri nadaljnjih analizah in uporabljenih metodah.

Ocenjujejo, da je metoda PHA ena izmed najpomembnejših metod pri razvoju varne programske opreme, saj na rezultatih le-te temeljijo vse nadaljnje analize. Z uporabo te metode izgradimo ogrodje (ang.: framework) za določitev hazardov [8, str. 24].

4.1 Nivoji tveganja

PHA vpeljuje nivoje tveganja, s katerimi določimo vrstni red pomembnosti hazardov. Tabela 1 prikazuje stopnjo resnosti hazardov. Uporabljamo lestvico od 1 do 10, s pripadajočimi definicijami resnosti hazardov. Tabela je povzeta po [11 in 8. str 26].

stopnja resnosti	**definicija resnosti hazarda**	**opis**
1	zanemarljiv	brez resnejšega vpliva, manjša motnja
2		manjša poškodba sistema, možna avtokorekcija sistema
3		manjša poškodba sistema, potrebna korekcija sistema s strani pooblaščene osebe; ni izgube parametrov
4	zmeren	manjša poškodba sistema, potrebna korekcija sistema s strani pooblaščene osebe; možna izguba nekaterih parametrov
5		poškodba sistema, izguba parametrov
6	kritičen	poškodba sistema, odpoved manj pomembnih modulov
7		težka poškodba sistema, odpoved večih modulov
8	katastrofičen	izguba sistema, potrebna zamenjava
9		izguba sistema, finančne posledice – tožbe
10		izguba sistema, možna smrt uporabnikov

Tabela 1: stopnja in definicija resnosti hazardov

Meje med posameznimi nivoji niso jasno določene, zato lahko različne skupine analitikov pridejo do različnih rezultatov.

4.2 Postopek PHA

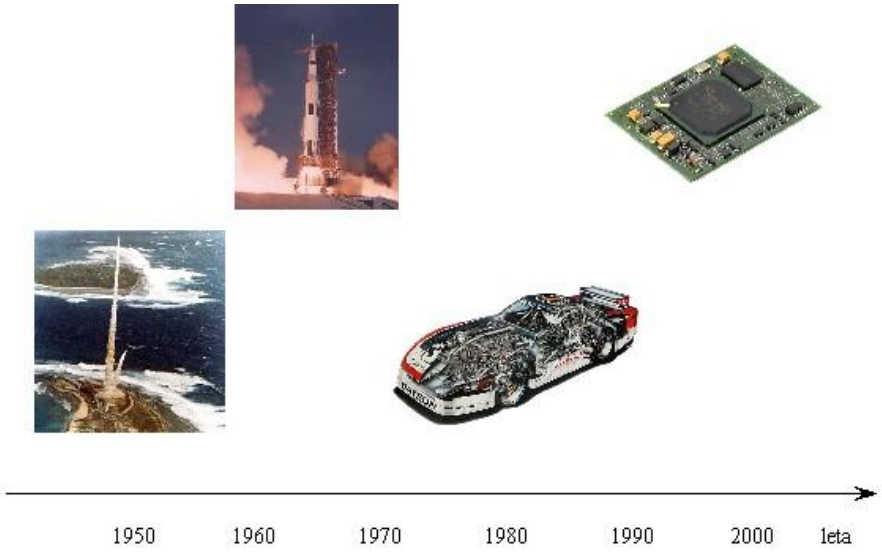
Postopek pri analizi je sledeč [6]:

- pregledamo obstoječo dokumentacijo o možnih hazardih, ali na sestanku prevetrimo ideje (ang.: brainstorming) o možnih hazardih,
- priskrbimo opis možnih hazardov in odpovedi povezanih z njimi,
- opredelimo možne vzroke za nastanek hazardov,
- v tabeli opredelimo resnost hazardov in
- se odločimo ali potrebujemo postopek za odpravo oziroma zmanjšanje tveganja.

Rezultati postopka so tudi vhodni podatki pri določitvi zahtev in specifikacij sistema.

5 Metoda FMEA

Metodo FMEA so zasnovali v ameriški vojski. Prvič je uporaba metode opisana v vojaškem standardu MIL-P-1629, ki je nastal leta 1949. Nato so nanjo pozabili, nakar so jo ponovno odkrili v šestdesetih letih prejšnjega stoletja pri NASA z Apollo odpravami na Luno. V osemdesetih letih so jo uspešno prenesli v avtomobilsko industrijo. Danes opredeljuje uporabo metode v avtomobilski industriji standard SAE J-1739. Ko so ji dodali generičen značaj, je hitro postala popularna v mnogih drugih industrijah. Z digitalno revolucijo so jo prenesli v svet vgradnih sistemov in programske opreme [10, str. 12]. Implementacija metode FMEA v različnih industrijah skozi čas prikazuje naslednja slika.



Slika 6: uporaba FMEA metode v različnih področjih skozi čas

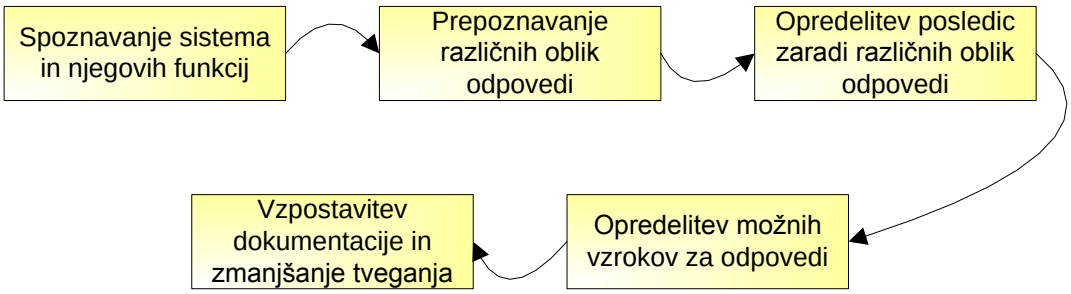
5.1 Splošni pregled

Metoda FMEA je ena izmed mnogih metod analize tveganja, ki jih priporočajo mednarodni standardi. Omogoča sistematični iterativni postopek za odkrivanje vrste možnih hazardov, ki lahko vodijo v odpoved sistema. Prav tako omogoča odkrivanje vzrokov, ki pripeljejo do hazardov in nenazadnje omogoča ovrednotenje in zmanjšanje posledic ob odpovedi sistema.

Metode FMEA razvijamo tako, da iščemo odvisnost med vzroki in posledicami ter jih poskušamo odpraviti. Tako smo pri postopku FMEA osredotočeni na tri glavne cilje:

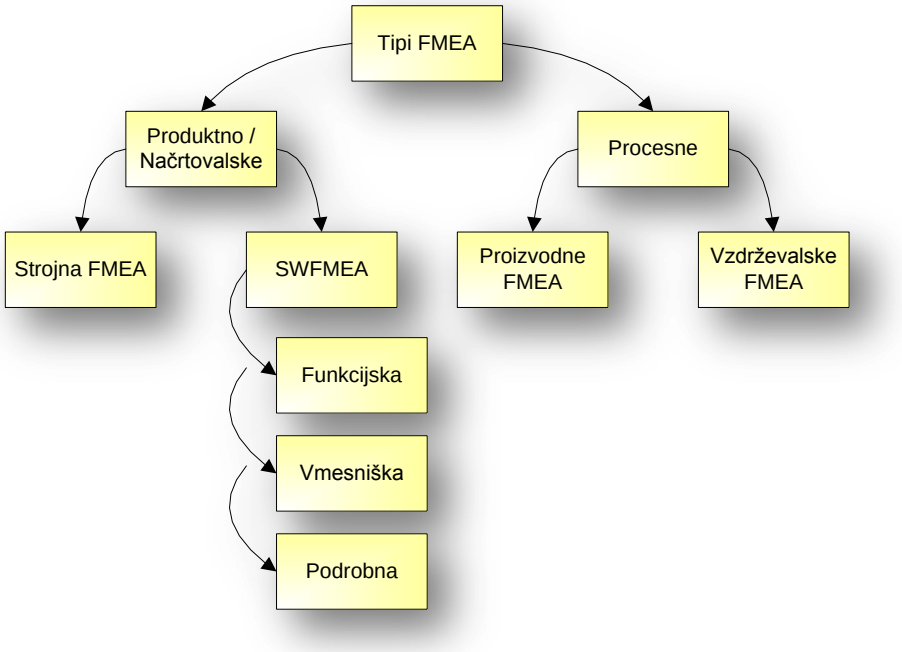
- prepoznavanje in ocenjevanje možnih hazardov, odpovedi in njihovih posledic,
- izgradnja prednostnega seznama ukrepov, ki odpravijo hazarde, možne odpovedi ali zmanjšajo njihovo verjetnost pojavljanja,
- izgradnja dokumentacije, ki ovrednoti podane izboljšave.

Slika 7 prikazuje splošni postopek pri metodi FMEA. Prikazane so glavne faze, s katerim pridemo do končnih rezultatov. Znotraj posameznih faz pa so iterativni postopki.



Slika 7: glavne faze FMEA

V literaturi najdemo različne klasifikacije tipov metode FMEA. Na sliki 8 prikazujemo razvejanost uporabe različnih tipov metode FMEA. Metoda SWFMEA, ki jo bomo podrobneje razčlenili, spada med produktne/načrtovalske tipe FMEA (ang.: Product/Design FMEA).



Slika 8: tipi FMEA

5.2 Delitev metode SWFMEA

Razloge in vzroke za odpovedi posameznih strojnih komponent poznamo. Nastanejo zaradi obrabe, staranja ali nepričakovane obremenitve. Pri programski opremi je situacija nekoliko drugačna. Odpovedi programske opreme so ponavadi vnaprej neznane. Programski moduli ne

odpovejo, temveč zgolj izkažejo nepravilno obnašanje [10, str. 14]. Zato uporabimo v različnih fazah življenjskega cikla različne vrste metode SWFMEA. Le-to razdelimo na tri podtipe, ki jih uporabimo v različnih fazah življenjskega cikla [9, str. 17-3].

5.2.1 Funkcijska SWFMEA

Funkcijsko metodo SWFMEA (ang.: Functional SWFMEA) uporabimo v fazi načrtovanja arhitekture sistema. Rezultati morajo opozoriti na možne slabosti zastavljene arhitekture. V večini primerov se osredotočimo na spodnje tri simptome:

- nepravilno izvrševanje funkcijskih blokov,
- nepopolno izvrševanje funkcijskih blokov in
- izvrševanje ob nepravem času.

5.2.2 Vmesniška SWFMEA

Vmesniško metodo SWFMEA (ang:. Interface SWFMEA) uporabimo pri analizi povezovanja različnih programskih modulov oziroma pri vmesnikih med strojno in programsko opremo. Najpogostejši trije simptomi so:

- nezmožnost vmesnika, da osveži vrednosti,
- nepopolna osvežitev vrednosti in
- osvežitev vmesniških vrednosti se zgodi ob nepravem času.

5.2.3 Podrobna SWFMEA

Podrobno metodo SWFMEA (ang.: Detailed SWFMEA) uporabimo v fazi podrobnega načrtovanja sistema. Podrobno jo izvršimo na posameznem kritičnem funkcijskem bloku. V precep vzamemo posamezno spremenljivko in opazujemo njen vpliv na celotni sistem. Cilj te metode je natančnejši opis dognanja predhodnih dveh postopkov. V praksi jo najpogosteje uporabimo pri načrtovanju prenosa in procesiranja podatkov.

5.3 Postopek SWFMEA

V celotnem postopku analize SWFMEA vgradnih sistemov se priporoča analiza strojne in programske opreme. Diplomska naloga se osredotoča zgolj na analizo programske opreme, medtem ko je strojna oprema predstavljena samo v tisti meri, ki je potrebna za analizo programske opreme [10, str. 21].

Celoten postopek SWFMEA analize lahko strnemo v sledeče točke:

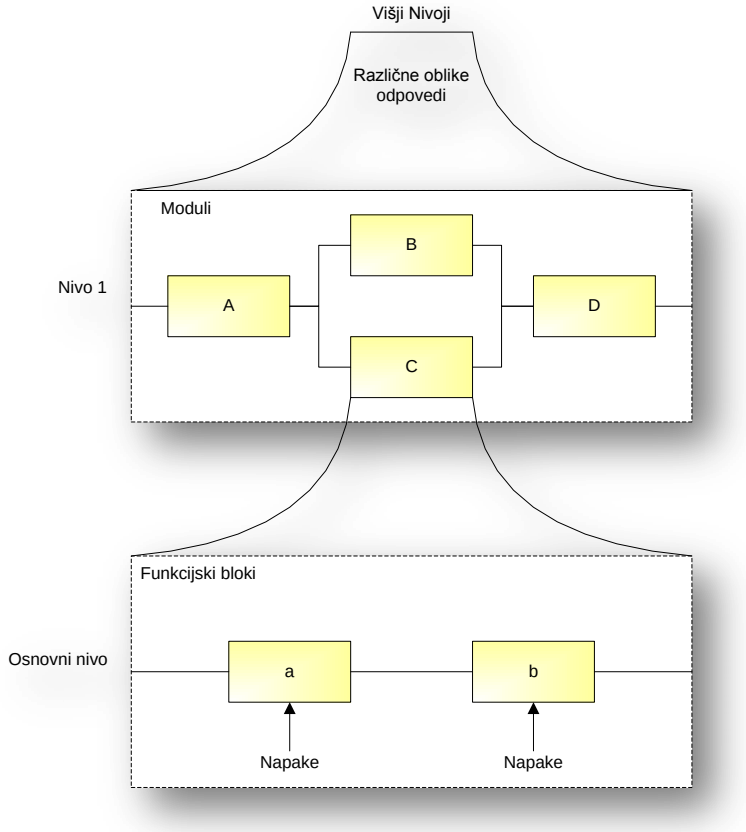
- definiramo meje analize,

- definiramo merilo za klasifikacijo dogodka kot napake,
- razstavimo sistem na osnovne gradnike – funkcijske bloke,
- za posamezen element ugotovimo in zabeležimo hazard,
- povzamemo ugotovitve in ukrepamo.

Podrobneje bomo razčlenili meje analize in določanje hazardnih stanj.

5.3.1 Meje analize

Na začetku določimo globino analize sistema z metodo SWFMEA. Ker se lahko zgodi, da se osredotočimo na napačne vidike analize, sistem hierarhično razdelimo na funkcijske bloke. Tako lahko na najnižjem nivoju opredelimo napake, ki privedejo do hazardov. Na naslednji sliki prikazujemo osnovne funkcijske bloke, v katerih se lahko pojavi napaka in napreduje preko različnih modulov do odpovedi.



Slika 9: SWFMEA hierarhija in napredovanje napak do odpovedi

5.3.2 Določanje hazardnih stanj

Ko razdelimo sistem na module in funkcijske bloke, določimo vse možne hazarde. Za strojno opremo proizvajalci podajajo nekatere parametre, na podlagi katerih sklepamo o deležu in frekvenci odpovedi komponent. Ker pri programski opremi takih podatkov vnaprej ne

poznamo, možne hazarde težko ovrednotimo. Ta del je najtežje uresničiti pri metodi SWFMEA, zato si pri določanju hazardnih stanj pomagamo bodisi z metodo PHA ali s predhodnimi izkušnjami. Osredotočimo se na tiste točke, ki smo jih opisali v poglavjih od 5.2.1 do 5.2.3.

5.3.3 Ocenitev kritičnosti

Ocenitev kritičnosti (ang.: criticality analysis) je kvantitativna nadgradnja metode SWFMEA. Standarda MIL-STD-1629A in IEC 60812 opredeljujeta dva vidika kritičnosti – frekvenco pojavljanja odpovedi (ang.: likelihood of occurrence, F) in resnost odpovedi (ang.: severity of failure, S), medtem ko SAE J-1739 vpeljuje tretji vidik – tako imenovano verjetnost zaznave odpovedi (ang.: detection probability, D) [8, str. 268; 10, str. 21 in 12, str. 3]. Produkt dveh oziroma treh števil imenujemo število prioritete tveganj (ang.: Risk Priority Number) RPN.

$$\mathrm{RPN} = \mathrm{S} \cdot \mathrm{F} \tag{1}$$

Resnost odpovedi povzamemo bodisi iz analize PHA, bodisi jo določimo na podlagi preteklih izkušenj. V tem diplomskem delu uporabljamo lestvico, ki smo jo določili v poglavju 4.1. Frekvenco pojavljanja odpovedi za programsko opremo težko določimo, saj ne poznamo pogostosti pojavljanja okvar vnaprej. V literaturi najdemo vrsto predlogov za njeno nadomestilo, zato v nadaljevanju opisujemo uporabo McCabeove ciklomatične kompleksnosti (ang.: McCabe's cyclomatic complexity) [14]. To je matematična metoda, ki jo je predlagal McCabe leta 1976 in omogoča merjenje kompleksnosti programske opreme. Avtor definira ciklomatično kompleksnost $v(G)$ grafa G z n vozlišči, e robovi in p povezanimi komponentami kot:

$$v(G) = e - n + 2 \cdot p \tag{2}$$

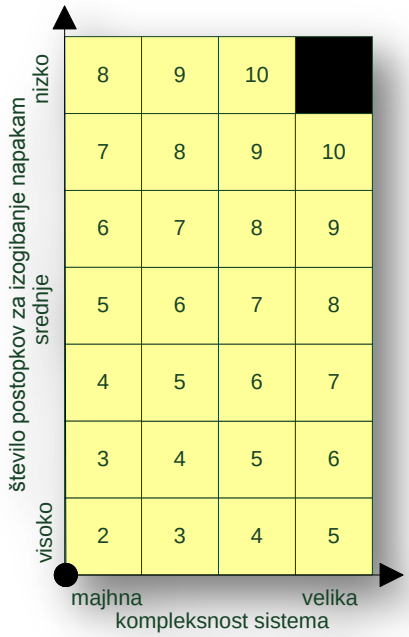
V tesno povezanem grafu G je ciklomatična kompleksnost enaka največjemu številu linearno neodvisnih poti skozi graf. Vsak odločitveni, ponavljajoči ali izbirni stavek v programu predstavlja svoje vozlišče. Vozlišča so medseboj povezana z robovi.

Ciklomatično kompleksnost bi lahko ponazorili na grafu, vendar smo v tem diplomskem delu uporabili odprtokodni program *CCCC* (ang.: C and C++ Code Counter). Program izlušči iz posamezne izvorne C datoteke funkcijske bloke in izračuna ciklomatično kompleksnost [15]. Dobljeno vrednost predstavimo kot absolutno število, kot prikazuje spodnja tabela.

ciklomatična kompleksnost	ocena tveganja
1 -10	preprost F.B[1].
11 -20	srednje kompleksen F.B., zmerno tveganje
21 -50	kompleksen F.B., visoko tveganje
več kot 50	prezapleten F.B., izredno visoko tveganje

Tabela 2: povezava med ciklomatično kompleksnostjo in kompleksnostjo funkcijskih blokov

V [10 str. 25] je opisan obstoj odvisnost med frekvenco pojavljanja odpovedi in obsežnostjo programske kode. Pristop, ki so ga predlagali v [12], ponazarja spodnja slika.



Slika 10: frekvenca pojavljanja napak – število od 1 do 10

Ugotavljajo, da je frekvenca pojavljanja napak odvisna od kompleksnosti programske opreme in upoštevanja postopkov za izogibanje napakam. Ti postopki so strukturirana analiza, objektno orientirana analiza in načrtovanje, načrtovalski in programerski standardi, standardiziran programski jezik, potrjen prevajalnik ter uporaba formalnih in simulacijskih metod. Ker uporabljamo vse zgoraj naštete postopke in elemente razen formalnih in simulacijskih metod, predpostavljamo, da je frekvenca pojavljanja napak različnih funkcijskih blokov odvisna zgolj od kompleksnosti posameznega modula.

[1] funkcijski blok

Definirajmo normalizirano ciklomatično kompleksnost v_n, ki nam bo služila kot vhodni podatek za izračun RPN.

$$v_n = \left\lceil \frac{v(G)}{v_{\max}(G)} \cdot 10 \right\rceil \tag{3}$$

Enačba pomeni, da ciklomatično kompleksnost posameznega funkcijskega bloka $v(G)$ delimo z največjo smiselno vrednostjo ciklomatične kompleksnosti (50) $v_{\max}(G)$. Tako zadostimo zahtevam za primerjalno lestvico od 1 do 10 [17].

Verjetnosti zaznave odpovedi pri izračunu RPN v tem diplomskem delu ne upoštevamo, ker lahko opredelimo s samodiagnostičnimi metodami samo del programskih napak, neznano velik del pa ostane prikrit. Taki rezultati so lahko zavajajoči, zato jih ne upoštevamo.

5.4 Uporabljena analitična orodja

Pri analizi uporabljamo bločne diagrame, tabele in grafe Pareto. Pri metodi SWFMEA delimo bločne diagrame na:

- funkcijske bločne diagrame in
- zanesljivostne bločne diagrame.

Funkcijske bločne diagrame uporabljamo za prikaz bodisi celotnega postopka bodisi povezave med posameznimi funkcijskimi bloki. *Zanesljivostne bločne diagrame* uporabljamo za prikaz medsebojne odvisnosti ali neodvisnosti med moduli pri izvrševanju zahtevane funkcije. Uporabljene simbole v diagramih, povzete po [9, str. 9-1 in 11 str. 60], prikazujemo v spodnji tabeli.

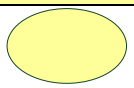
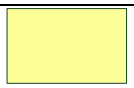
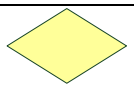
simbol	pomen
	izhodna in izhodna točka v sistem
	blok dejanj, sprejme lahko več vhodov ima samo en izhod
	odločitveni blok, ima vsaj dva izhoda

Tabela 3: uporabljeni simboli v bločnih diagramih pri metodi SWFMEA

Tabele za zapis podatkov so prirejene po [16]. Pri funkcijski in vmesniški metodi SWFMEA uporabljamo spodnjo razpredelnico. Razlika je zgolj v prvem stolpcu, kjer pri prvi zapišemo

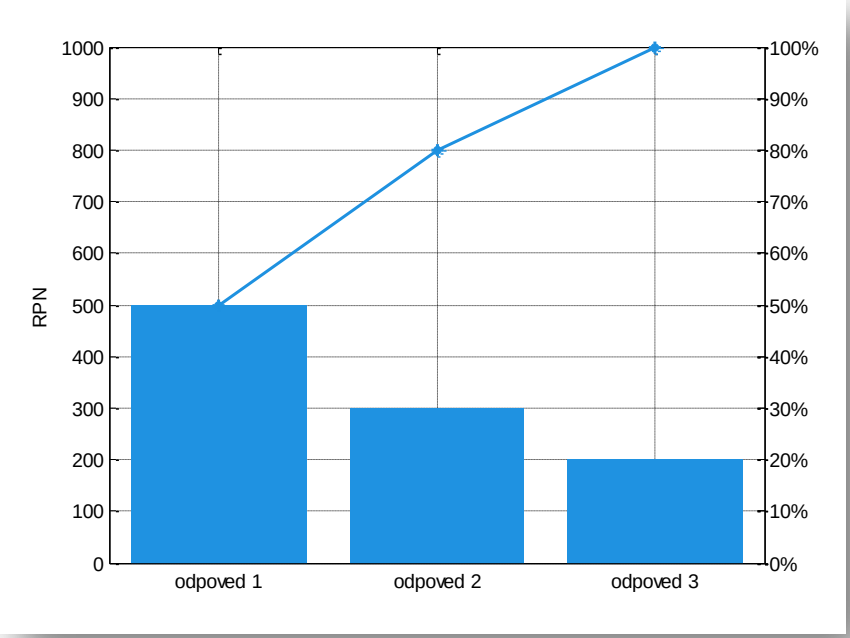
ime elementa, pri drugi pa ime vmesnika. [16] opisuje sistem za uravnavanje kroglice v navpično postavljeni tubi, zato je v tabeli 4 podana funkcijska SWFMEA za A/D pretvornik.

element/ vmesnik	oblika odpovedi	lokalna posledica	posledica na višjem nivoju	posledica na sistem	RPN	opombe
A/D pretvornik	napaka pri zajemanju oziroma pretvorbi na strojni opremi	napačna vrednost na izhodu	hitrost ventilatorja se ne spreminja	nezadostna regulacija ali odpoved regulacije	4	vzpostavitev sistema za periodično preverjanje, ali se vhodni nivoji v mikrokrmilniku spreminjajo

Tabela 4: uporabljena razpredelnica pri SWFMEA metodi

Pri podrobni metodi SWFMEA uporabljamo bodisi zgornjo tabelo, kjer v prvi stolpec zapišemo posamezno spremenljivko, bodisi razširjeno tabelo, kjer prikažemo posamezen vpliv različnih spremenljivk na opazovano spremenljivko.

Graf Pareto sestoji iz stolpčnega in kumulativno črtnega grafa. Upošteva Paretovo pravilo, ki pravi, da 80 odstotkov problemov izhaja iz 20 odstotkov vzrokov [13]. Stolpce postavljamo od leve proti desni v padajočem vrstnem redu. Kumulativni črtni graf prikazuje odstotkovni doprinos posameznega stolpca. Pri metodi SWFMEA uporabimo graf Pareto, ko želimo primerjati RPN različnih oblik odpovedi in tako poiskati najpomembnejše odpovedi [9, str. 9-6]. Spodnja slika prikazuje primer grafa Pareto.



Slika 11: graf Pareto za primerjavo RPN med različnimi oblikami odpovedi

5.5 Omejitve metode

Metoda ima tudi nekaj pomanjkljivosti, zato počasneje prodira v vse veje industrije. Te pomanjkljivosti so [8, str. 260 in 6]:

- časovna zahtevnost,
- težavnost metode,
- rezultat odvisen od znanja analitika,
- statičnost metode.

Časovna zahtevnost. Za natančno analizo potrebujemo veliko časa in ljudi. Zato je pri metodi SWFMEA pomembno, da na začetku podrobno opredelimo meje analize.

Težavnost metode in *rezultat odvisen od znanja analitika*. Metoda zahteva široko znanje o sistemu, zato je smiselno pritegniti k projektu čim večje število ljudi z različnimi znanji. Določanje RPN je subjektivno in odvisno od znanja načrtovalca oziroma analitika.

Statičnost metode. Metoda ne more zajeti vseh dinamičnih lastnosti sistema. Prav tako ne more potrditi stabilnost različnih algoritmov ob pravilnem delovanju sistema.

6 Programski jezik C

Programski jezik C je nastajal v Bellovih laboratorijih od leta 1971 dalje. Oblikoval ga je Dennis M. Ritchie kot razvojno orodje za operacijski sistem UNIX. Izhaja iz starejših programskih jezikov B Kena Thompsona in BCPL (ang.: Basic Combined Programming Language) Martina Ritchardsa [18, str. 1 in 20].

Leta 1978 sta Dennis M. Ritchie in Brian Kernighan napisala knjigo The C Programming Language in v njej opisala programski jezik C. Ta knjiga je znana kot bela knjiga (ang.: white book) in je dolgo veljala za standard. Po tej knjigi je Steve Johnson napisal prevajalnik pcc, ki je služil kot referenčni prevajalnik. V naslednjih letih je sledil razmah računalništva in s tem tudi C-ja. Dodajali so nove funkcionalnosti in lastnosti, ki jih avtorja v beli knjigi nista omenila, ali pa sta jih zgolj bežno opisala. Nastala je potreba po standardizaciji [20].

Leta 1982 so pod okriljem organizacije ANSI ustanovili delovno skupino X3J11. Pot do končnega standarda je bila trnjava, saj so ga sprejeli leta 1989. Danes je poznan kot ISO/IEC 9899-1990 oziroma C90. Leta 1999 so naredili revizijo le-tega in nastal je standard ISO/IEC 9899-1999 oziroma C99. Večina današnjih prevajalnikov in napisane kode še vedno ustreza standardu C90.

6.1 Lastnosti programskega jezika C

C danes uvrščamo med programske jezike srednjega nivoja. Nekateri njegovi operatorji določajo operacije na nivoju registrov (kot zbirni jezik), drugi operatorji in stavki pa določajo širšo dejavnost (kot višji programski jeziki) in se prevedejo v niz strojnih ukazov [18, str.2].

Avtor navaja, da je C pomanjkljiv, a zaradi svoje preprostosti izredno uspešen programski jezik [20]. V 70. letih prejšnjega stoletja je bila potreba predvsem po hitrih in kompaktnih programih, brez odvečne kode za preverjanje ustreznosti podanih argumentov (to je bilo prepuščeno programerju), zato so danes največje hibe C-ja manipulacije s kazalci, odsotnost preverjanja velikosti polj, podanih preko kazalca in šibko pretvarjanje različnih tipov.

Kazalci omogočajo programerju dostopanje do vseh naslovov v pomnilniku, kar je zaželeno kadar pišemo gonilnik, ki zahteva dostop do pomnilniških naslovov perifernih enot. Vendar pa veliko napak izvira ravno iz zahtevne kazalčne aritmetike. Seštevanje, odštevanje in

primerjava kazalcev hitro privede do prekoračitve mej podatkovnih struktur, uporaba neinicializiranih kazalcev vodi do prebitja sklada ali rezerviranega pomnilniškega prostora in posledično neodzivnega sistema.

Programski jezik C ne preverja mej polj in nizov, temveč to prepušča programerju. Ta problem pride do izraza, kadar podajamo polje kot funkcijski parameter preko kazalca. V tem primeru ne poznamo števila elementov polja, zato lahko zlahka pišemo preko mej polja. V ta namen dodamo nov funkcijski parameter, ki podaja velikost polja.

Močno preverjanje tipov pomeni pretvarjanje iz enega tipa v drugega pod točno določenimi pogoji. C omogoča implicitno oziroma šibko pretvarjanje tipov, kar pomeni, da preko nekaterih lastnosti jezika nehote pretvorimo tip (spremenljivko tipa `integer` v tip `char`). Ker C ne pozna močnega pretvarjanja tipov, je zanesljivost napisane programske opreme manjša [8, str.144].

Prav tako so problematične globalne spremenljivke, saj jih lahko nekontrolirano spremenimo iz katerega koli dela programa. Razumemo jih lahko kot dodatne vhodne parametre k vsem funkcijam. Tako lahko nevede spremenimo vrednost pomembnim spremenljivkam in povzročimo napačno izvajanje programa [8, str.146].

Nenazadnje so nekatera področja standarda ohlapno opredeljena ter dovoljujejo razvijalcem prevajalnikov proste roke, zato pri pisanju prenosljive in varne kode pazljivo preučimo tovrstne primere. Omejevanje C-ja z novimi jezikovnimi značilnostmi, ki bi rešile zgoraj naštete probleme, ni smiselno, saj bi mu spremenili funkcionalnost in s tem odvzeli najpomembnejša atributa: generičnost in moč. Varnost pri programiranju dosežemo z lastnim trudom in naporom. V literaturi je mnogo napotkov za uspešno prebroditev pomanjkljivosti različnih definicij C-ja in programiranja nasploh. V tem diplomskem delu opisujemo obrambno programiranje.

7 Obrambno programiranje

Obrambno programiranje (ang.: defensive programming) je metoda, s katero zmanjšamo možnost nastanka odpovedi. Vsako komponento v sistemu razvijemo tako, da preverja svoje delovanje in javi morebitne napake. Obrambno programiranje omogoča, da zaznamo napako zgodaj v sistemu in ne tedaj, ko le-ta že preide v odpoved. Pristopi k zaznavanju napak so različni, v grobem jih lahko razdelimo na tiste, ki so neodvisni in tiste, ki so odvisni od programskih jezikov [19, str. 6].

7.1 Pristopi obrambnega programiranja

V nadaljevanju najprej predstavimo splošne intuitivne prijeme, kasneje pa tiste, ki jih uporabljamo pri programskem jeziku C.

7.1.1 Test centralne procesne enote

Centralna procesna enota (CPE) je občutljiva na elektromagnetna motenja (ang.: electromagnetic interference, EMI), električne razelektritve in šoke. Vsi ti vzroki škodujejo ali delno onemogočajo pravilno delovanje CPE, zato ob inicializaciji naredimo samodiagnostični test. Če je le-ta neuspešen, varno zaključimo z izvajanjem programa in javimo napako [8, str. 197].

7.1.2 Časovni čuvaj

Če števca časovnega čuvaja (ang.: watchdog) ne ponastavimo v določenem času, ponovno zažene CPE. V programih z neskončno zanko števec ponastavimo enkrat na izvajanje zanke. Za ponastavljanje števca ne smemo uporabljati prekinitev, saj le-te lahko še vedno delujejo, medtem ko so ostali procesi zaradi napake zaustavljeni [8, str. 197]

7.1.3 Zaščita pred nenamernim spreminjanjem spremenljivk

Pomembne spremenljivke hranimo v večih kopijah in jih dodatno zaščitimo s cikličnim preverjanjem redundance (ang.: cyclic redundancy check, CRC). Ob napaki pri računanju CRC prve kopije spremenljivke uporabimo njeno drugo kopijo. Če se tudi njen CRC ne ujema, vzamemo ponastavljene vrednosti [8, str. 197].

7.1.4 Preverjanje sklada

Preverjanje sklada omogoča zaznavo preplavitve ali popačenosti le-tega. Ob inicializaciji nastavimo vrednosti sklada na znano vrednost (recimo na število 55 v šestnajstiškem sistemu,

kar predstavlja izmenično postavljanje in brisanje bitov). Nato preverjamo, kdaj se približamo meji sklada in se ustrezno odzovemo s povečanjem sklada (če to lahko naredimo med izvajanjem programa) ali varno zaustavitvijo sistema [8, str. 198].

7.1.5 Uporaba simulatorja ali emulatorja

Vgradni sistemi so za razhroščevanje izredno zahtevni. Včasih razhroščevanje preko UART ni dovolj in za hitrejše razhroščevanje potrebujemo dodatna orodja. Pomagamo si s simulatorji, ki tečejo na osebnih računalnikih in tako izkoriščajo moč le-tega, ter emulatorji (ang.: in-circuit emulator, ICE), ki vsebujejo dodatna vezja za postavljanje dodatnih prekinitvenih točk (ang.: breakpoints).

7.1.6 Prevajanje programa z vključenimi opozorili

Pri prevajanju programa prevajalniki programskega jezika C preverjajo ustreznost kode. Če je napisana koda v neskladju s pravili, prevajalnik javi napako. Večina zmore javiti tudi različna opozorila (ang.: warnings) o možnih logičnih napakah. Ena izmed najpogostejših napak pri pisanju kode v C-ju je zamenjava operatorjev == in =. Prvi je znak za enakost, drugi priredi spremenljivki neko drugo spremenljivko (ang.: lvalue) ali vrednost (ang.: rvalue). Pogosta napaka je tudi uporaba spremenljivke, preden se ji dodeli vrednost. V tem primeru je delovanje programa izrazito naključno in odvisno od podatkov v delovnem pomnilniku. Prevajanje z opozorili imamo zaradi zgoraj naštetih razlogov vedno vključeno. Ko se pojavi opozorilo, ga temeljito preučimo in odpravimo [19, str. 11].

7.1.7 Uporaba varnih podatkovnih struktur

Programski jezik C je zaradi kazalcev izredno močno orodje. Preko njih dostopamo do strojnih registrov perifernih enot. Prav tako pri prenosu funkcijskih parametrov preko kazalcev manj obremenjujemo sklad. Vendar pa ima uporaba kazalcev tudi slabe plati, saj večina varnostnih problemov in napak izvira iz preplavitve pomnilnika (ang.: buffer overrun). Pisanje preko mej polja je tipična napaka. Le-te izvirajo zasnove programskega jezika, ki tesno povezuje polja in kazalce. Elemente polja lahko naslavljamo preko kazalcev in pri tem nevede prečkamo mejo polja. Ker je naslavljanje preko kazalcev legalna operacija, prevajalnik ne javi napake. Programer sam zagotovi varno manipulacijo s kazalci s preverjanjem velikosti ciljnega polja [19, str.12].

7.1.8 Preverjanje statusa funkcij

Funkcije razvijemo tako, da vrnejo status glede na uspešnost opravljene naloge. Vrednosti podajamo na več načinov, bodisi preko pravilnega (ang.: true) oziroma napačnega (ang.: false) stanja bodisi preko deklaracije oštevilčenih tipov (ang.: enumeration declaration). Če vrednosti ne preverjamo, se napake potihoma prikradejo v kodo in povzročijo naključno izvajanje programa ali njegovo nasilno zaustavitev. Kot nazoren primer navedimo funkcije, ki manipulirajo s kazalci. Ob neuspešni inicializaciji javijo napako in vrnejo kazalec, ki kaže na naslov nič. Če ne preverimo statusa, ki ga je funkcija vrnila, napake ne prestrežemo, temveč uporabimo neveljaven kazalec. Delovanje programa je od te točke dalje vprašljivo. Možnih je več primerov, od tega, da CPE dvigne prekinitveno izjemo (ang.: abort exception) ter zaključi izvajanje v pripadajočem prekinitvenem strežniku (ang.: abort handler) do zapisa nepravilne vrednosti na neko lokacijo in spremembe poteka izvajanja programa [19, str. 13].

7.1.9 Deklaracija in inicializacija spremenljivk

Deklaracije spremenljivk opravimo točno takrat, ko jih potrebujemo in ne prej. Tako zagotovimo preglednost in zmanjšamo možnost za njihovo napačno uporabo. Prav tako se izogibamo uporabi iste spremenljivke za različne namene, saj to povzroča pri vzdrževanju kode nejasnosti in vodi k napakam. Prevajalniki so zmožni sami optimizirati uporabo spremenljivk [19, str.14].

Inicializacijo spremenljivk izvršimo v za to namenjeni funkciji, ki se kliče na začetku programa. Če pri deklaraciji pozabimo inicializirati spremenljivke, se program obnaša stohastično, saj je odvisen od podatkov, ki naključno zavzemajo področja delovnega pomnilnika [19, str. 14].

7.1.10 Uporaba standardnih elementov jezika

Programski jezik C je pri uporabi standardnih elementov jezika izredno nehvaležen. Ker se je standard razvijal skoraj desetletje in poskušal obdržati združljivost s starejšimi različicami jezika, je v določenih točkah in podrobnostih le-tega izredno nenatančen in prepušča odločitve proizvajalcem prevajalnikov. Večina sledi standardu C90, vendar so zaradi njegove ohlapnosti med njimi tudi razlike.

Tako je skupni imenovalec programiranja v C-ju pisanje kode v ANSI C-ju, kjer pa je standard nejasen uporabimo svoje rešitve in se ne zanašamo na nestandardne dodatke

prevajalnika. Tipičen primer je obravnava tipa `char`, ki ga nekateri prevajalniki razumejo kot predznačeno, drugi kot nepredznačeno 8-bitno število. Tipe opredelimo preko svoje tabele, ki vsebuje vse uporabljene standardne in lastne tipe, in se tako izognemo nedoločenosti standarda.

Če potrebujemo dostop do posebnih ukazov povezovalnika ali prevajalnika, to storimo preko predprocesorskega ukaza `#pragma`. Ta ukaz omogoča, da varno vnesemo kodo, odvisno od prevajalnika, in jo vključimo v predprocesorski blok. Ob menjavi prevajalnika moramo pregledati samo nekatere bloke kode.

7.1.11 Previdno pretvarjanje tipov

Večina jezikov omogoča pretvarjanje med različnimi podatkovnimi tipi. C-jevi predhodniki niso poznali tipov, zato je tudi C na tem področju dokaj šibek. Preverjanje tipov ni strogo določeno in zato prevajalniki sami naredijo pretvorbo med njimi. Vendar nekateri prevajalniki razumejo tip `integer` kot 2-zložno in drugi kot 4-zložno število. To vodi v težave s prenosljivostjo kode, saj prevajalniki različno tolmačijo podatkovne tipe. Vsako neujemanje tipov zato eksplicitno uredimo z njihovim pretvarjanjem (ang.: type casting).

7.2 Nadzorni seznam

Nadzorni seznam je sestavljen iz dveh delov. V prvem so zbrana vsa opravila dobrega načrtovanja programske opreme, povzeta po podpoglavju 7.1 in iz [8, str. 306 - 310], v drugem pa svojstvene lastnosti C-ja in njihove obrazložitve, povzete po [8, str. 294 - 297]. Upoštevati poskušamo čim večje število predlogov.

Tabela 5 prikazuje zastavljen nadzorni seznam za načrtovanje programske opreme. V prvem stolpcu so našteti uporabljeni postopki, v drugem in tretjem stolpcu pa potrdimo uporabo teh postopkov oziroma zapišemo morebitne komentarje.

načrtovalski postopek	uresničitev	komentar
testiranje CPE		
časovni čuvaj		
zaščita pred nenamernim spreminjanjem spremenljivk		
preverjanje sklada		
prevajanje programa z vključenimi opozorili		
uporaba varnih podatkovnih struktur		
deklaracije in inicializacije spremenljivk		
preverjanje vrednosti, ki jih vračajo funkcije		
uporaba standardnih elementov jezika		
previdno pretvarjanje tipov		
možnost detekcije izpada napajanja in postopna zaustavitev sistema		
seznam vseh možnih odpovedi strojne opreme, ki lahko vpliva na programsko		
zmanjšanje kompleksnosti kode		
uporaba metode vpiši preberi testiraj (z prvo funkcijo vpišemo spremenljivko, z drugo preberemo ter preverimo njeno integriteto		
načrtanje grafa odvisnosti med moduli		

Tabela 5: nadzorni seznam za načrtovanje programske opreme

Tabela 6 prikazuje mehanizme obrambnega programiranja za pisanje preglednejše in varnejše programske opreme v C-ju. V prvem stolpcu je opisan mehanizem obrambnega programiranja, v drugem pa prostor za potrditev realizacije.

mehanizem obrambnega programiranja	obrazložitev	uresničitev
izogibanje uporabe stavka `goto`	prinaša negotovost v programski tok in časovno stohastičnost	
uporaba oklepajev za določitev prioritete izvajanja operatorjev	večja preglednost. Izvajanja stavkov preko izrecne določitve prioritete in ne preko prioritetne liste operatorjev Cja	
omejitev števila in velikosti parametrov podanih funkcijam	preveliko število parametrov vpliva na zmožnost testiranja take funkcije, velike strukture, če niso podane preko kazalcev, lahko preplavijo sklad	
minimalna uporaba rekurzivnih funkcij	enostavna preplavitev sklada, ob uporabi takih funkcij zagotoviti končno rekurzijo	
uporaba direktive `default` pri izbirnem stavku `switch`.	eksplicitna določitev izvajanja programa v primeru neujemanja izbirne vrednosti z delom `case` v izbirnem stavku	
prepovedana uporaba funkcij z nedoločenim številom parametrov	prevajalnik ne more preveriti takih funkcij, zato jih je težko verificirati	
prepovedana uporaba operatorjev ++ in - - z parametri podanimi prek funkcij	izvrševanje stavka `funkcija(spremenljivka++)` je nepredvidljivo in vodi v napake	
uporaba bitnih mask preko makrojev, namesto bitnih polj	bitna polja niso specifično opredeljena v C90 in so odvisna od interpretacije prevajalnika	
uporaba predprocesorskega ukaza `#define` namesto direktne uporabe števil	uporaba makroja definiranega kot `#define` POLMER_ZEMLJE_V_KM 6356 je preglednejši kot samo 6356. Koda postane preglednejša in lažje vzdrževana. Spreminjanje konstane se odvija na enem mestu, pri makroju, in ne pri vsaki spremenljivki	

Tabela 6: nadzorni seznam svojstvenih lastnosti C-ja in pripadajoče obrambno programiranje

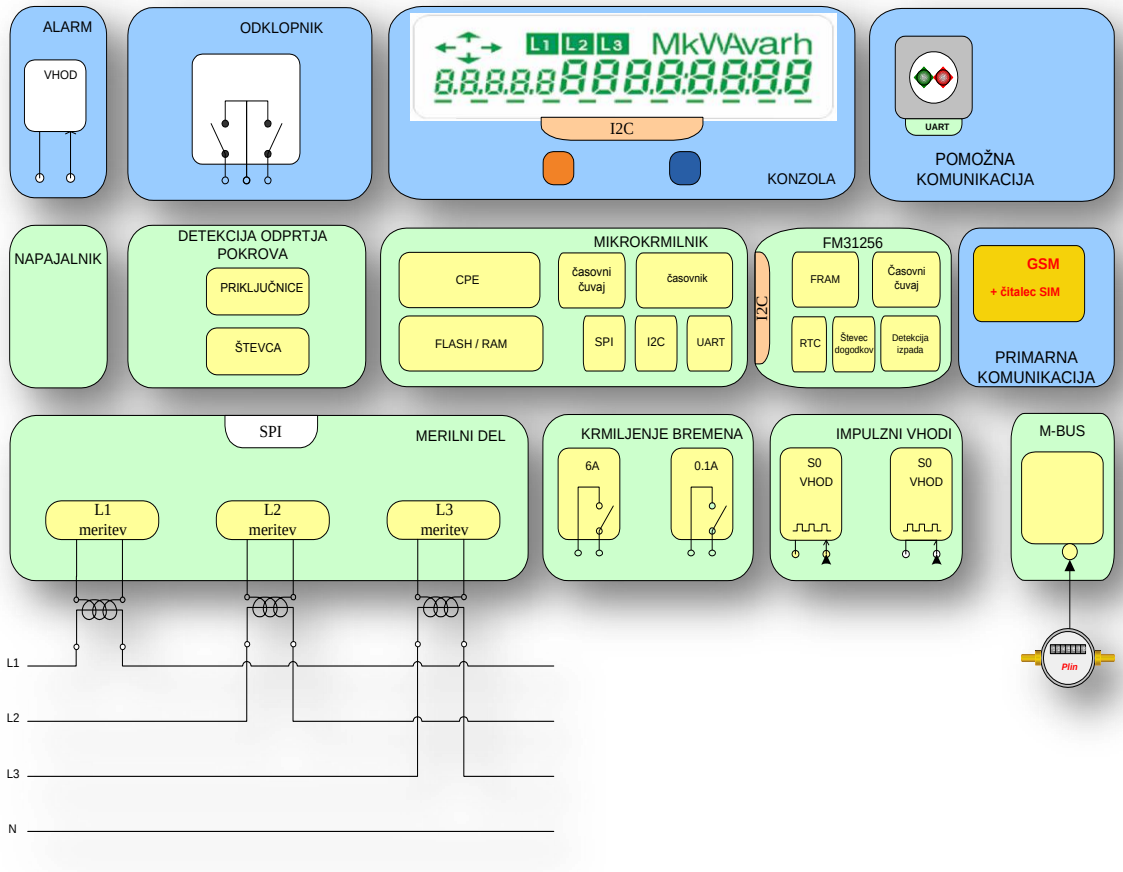
Oba nadzorna seznama pregledamo in izpolnimo v podrobni načrtovalski fazi in razvoju sistema.

8 Gradniki elektronskega števca MT372

Števec električne energije meri tok in napetost ter ju hrani, obdeluje in posreduje nadzornemu centru. Prav tako na podlagi pridobljenih podatkov izvaja različne naloge in akcije (npr. izklop uporabnika). Osnovna gradnika elektronskega števca MT372 sta strojna in programska oprema, ki ju opisujemo v nadaljevanju.

8.1 Strojna oprema

Uporabljena strojna oprema je zasnovana modularno, s čimer zagotovimo šibko povezanost med posameznimi sklopi in možnost preprostega dodajanja novih funkcionalnosti ter realizacijo dodatnih zahtev. Slika 12 predstavlja posamezne pomembnejše module števca MT372-SMART, ki jih podrobneje predstavljamo v nadaljevanju [30].



Slika 12: pomembnejši strojni moduli števca MT372

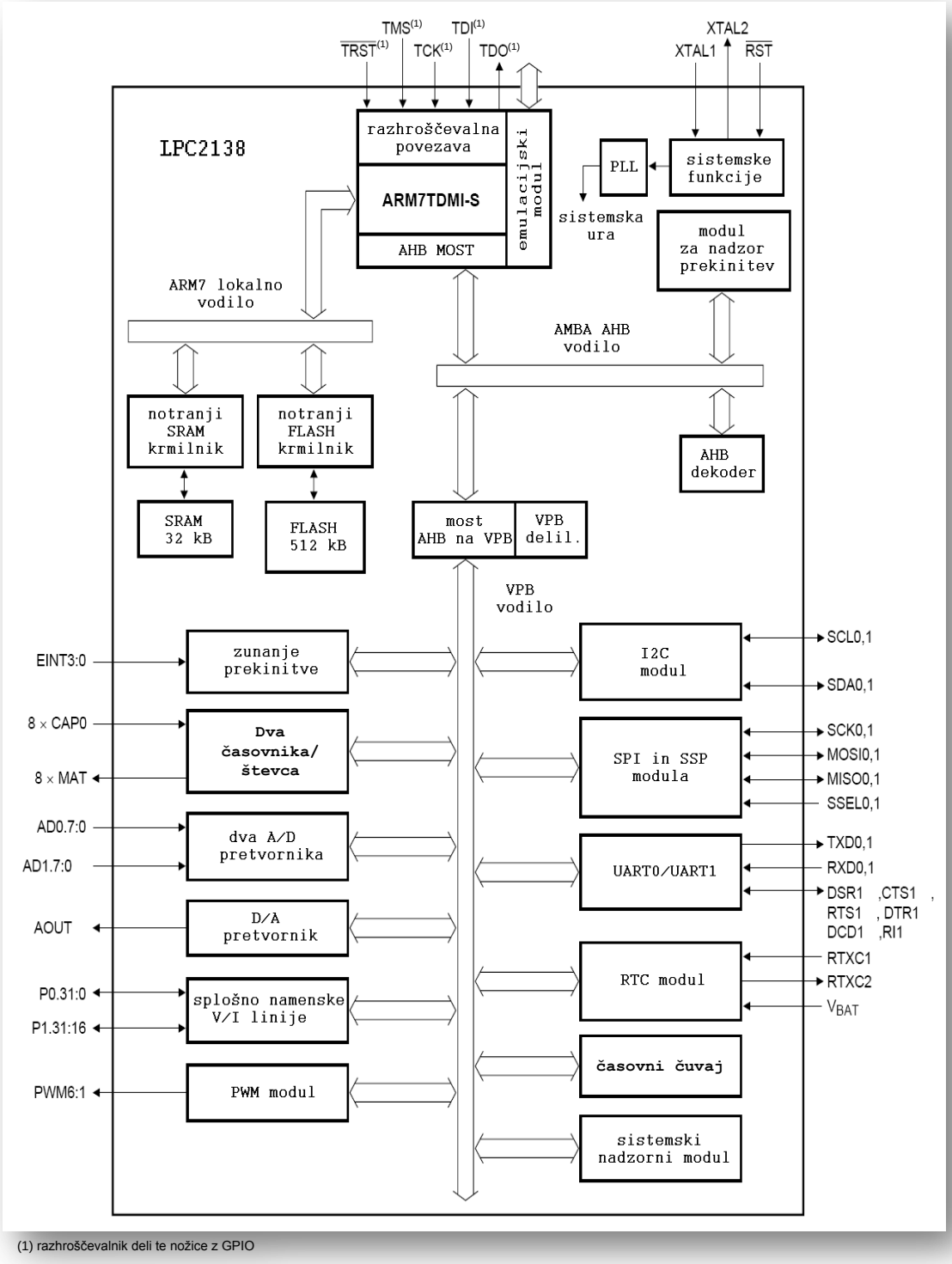
8.1.1 Mikrokrmilnik

Srce števca predstavlja NXP-jev 32-bitni mikrokrmilnik LPC 2138 [24] z vgrajenim bliskovnim in delovnim pomnilnikom, komunikacijskimi vmesniki ter ostalimi periferijami. Centralna procesna enota temelji na 32-bitnem ARM7TDMI-S® [23] jedru z najvišjim možnim taktom delovanja 80 MHz. Programska oprema se izvaja z bliskovnega pomnilnika, katerega dostopni čas je 50 ns, kar zmanjša največji možni urni cikel na 20 MHz. Pri NXP-ju so ta problem rešili z modulom MAM (ang.: memory accelerator module), ki pospeši dostop do bliskovnega pomnilnika. Z uporabo tega modula in modula PLL (ang.: phase locked loop) se maksimalni takt delovanja poveča na 60 MHz.

Pomembnejši moduli so še:

- 512 kB bliskovnega in 32 kB delovnega pomnilnika,
- vektoriziran prekinitveni krmilnik z nastavljivo prioriteto prekinitev (ang.: VIC),
- do skupno 47 splošno namenskih vhodno izhodnih linij (ang.: GPIO),
- dve vodili UART,
- vodili SPI ter SSP,
- dve vodili I^2C,
- dva 32-bitna časovnika/števca,
- modul pulzno širinske modulacije (ang.: PWM),
- dva 10-bitna A/D pretvornika,
- 10-bitni D/A pretvornik,
- časovni čuvaj ter
- modul ure realnega časa.

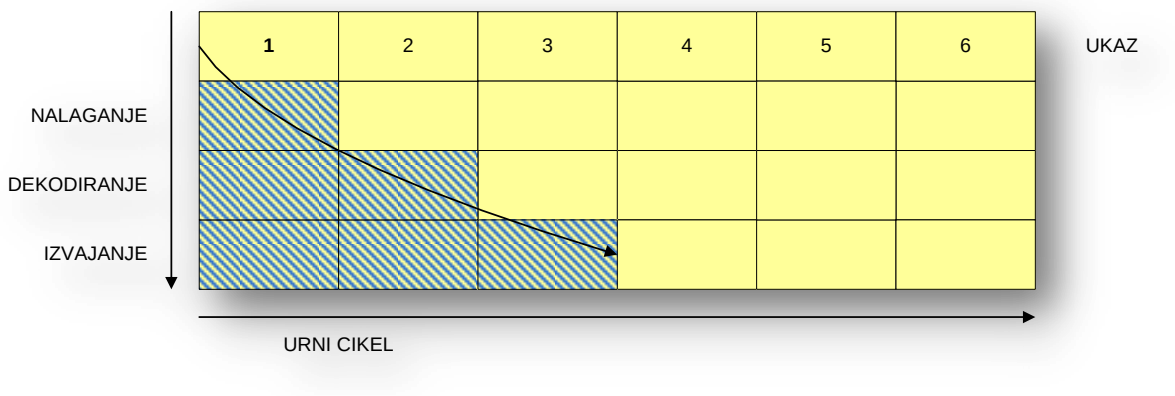
Slika 13 prikazuje blokovni diagram mikrokrmilnika LPC 2138.



Slika 13: blokovni diagram mikrokrmilnika LPC2138

Podrobneje predstavljamo tiste komponente mikrokrmilnika, ki jih potrebujemo, oziroma so bistvenega pomena pri razvoju merilnega dela števca.

Centralno procesna enota je ARMova ARM7TDMI-S®, ki spada med tako imenovane RISC (ang.: reduced instruction set computer) procesorje, kar pomeni, da je njen nabor ukazov majhen. Za hitrejše delovanje uporablja cevovod (ang.: pipeline), ki je sestavljen iz treh stopenj; prva izvaja trenutni ukaz, druga dekodira naslednji ukaz, tretja pa bere ukaz, ki se bo dekodiral v naslednjem ciklu. Tak princip delovanja omogoča, da se večina ukazov v povprečju izvrši v enem samem urinem ciklu. Vendar pa to velja zgolj za linearno kodo brez skokov in vejitev. Ko pride do vejitve se cevovod izprazni in napolni z novimi ukazi. Delovanje cevovoda prikazuje Slika 14.



Slika 14: delovanje cevovoda

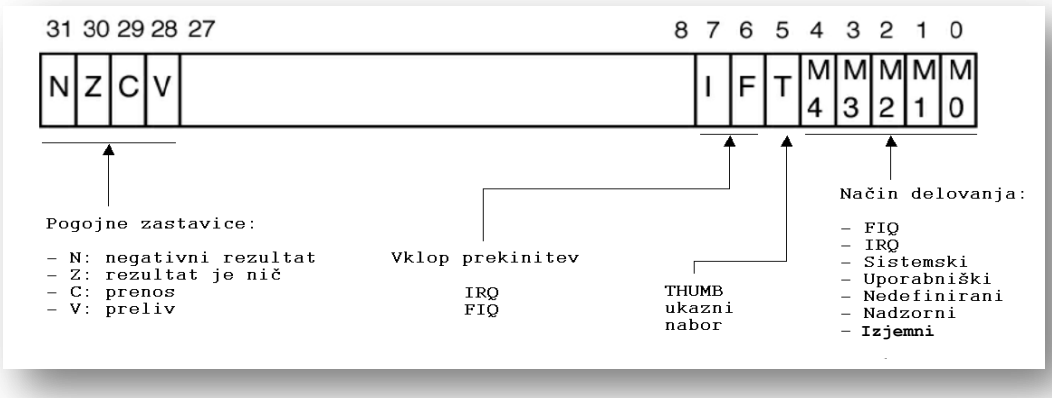
Obdelava podatkov v CPE poteka na sledeči način: podatek se naloži iz pomnilnika v enega od šestnajstih uporabniku dostopnih registrov, nato se le-ta obdela in prenese nazaj na želeno mesto v pomnilniku. Registri so 32-bitni in poimenovani od R_0 do R_{15}.

Prvih 13 registrov je splošno namenskih in nimajo natančno določene funkcije. Preostali trije registri, od R_{13} do R_{15} pa imajo posebno vlogo. Register R_{13} je skladovni kazalec (ang.: stack pointer, SP), R_{14} je povezovalni register (ang.: link register, LR) v katerega se, pri klicu nove funkcije, shrani povratni naslov (ang.: return address) ter R_{15}, ki je programski števec (ang.: program counter, PC).

Poleg naštetih registrov obstaja še kontrolni 32-bitni register CPSR (ang.: current program status register) (slika 15), katerega zgornji štirje biti predstavljajo zastavice, ki jih postavlja aritmetično-logična enota pri operacijah nad podatki. Te zastavice so: N za negativen rezultat (ang.: negative), Z za rezultat nič (ang.: zero), C za prenos (ang.: carry) in V za preliv (ang.: overflow).

Na spodnjih osem bitov posredno vpliva programska oprema. Sedmi in šesti bit omogočata vklop oziroma izklop različnih zunanjih virov prekinitev. Peti bit se imenuje THUMB in z njim preklapljamo med 16-bitnimi THUMB ali 32-bitnimi ARM ukazi. Prve uporabimo, ko želimo kompaktnejšo, a počasneje izvajajočo kodo, slednje pa pri zahtevanem hitrem izvajanju kode.

Spodnjih pet bitov določa enega izmed sedmih načinov delovanja CPE. Koda se normalno izvaja v uporabniškem načinu delovanja (ang.: user mode), z dostopom do zgoraj omenjenih 16-ih registrov. Ko preidemo iz uporabniškega v enega izmed posebnih načinov delovanja, se vsebina registra CPSR prepiše v njegovo sliko, to je register SPSR (ang.: saved program status register), s katerim lahko direktno manipuliramo. Posebni načini delovanja so: dva prekinitvena – navaden oziroma IRQ (ang.: interrupt request, IRQ) in hitri oziroma FIQ (ang.: fast interrupt request, FIQ), nadzorni (ang.: supervisor), izjemni (ang.: abort), nedefinirani (ang.: undefined) in sistemski (ang.: system). [22, str. 6-7; 23, str. 2-16].
Slika 15 prikazuje posamezne bite v registru CPSR.



Slika 15: pomen bitov v registru CPSR

Časovnika sta 32-bitna z nastavljivim 32-bitnim predskalirnikom (ang.: prescaler). Oba časovnika sta enaka in njun programski model vsebuje [24, str. 170]:

- štiri zajemalne kanale (ang.: capture channels), ki si zapomnijo vrednost prostotekočega števca časovnika ob prehodu stanja na digitalnem vhodu; ti kanali omogočajo proženje prekinitev,
- štiri 32-bitne primerjalne registre (ang.: match registers), ki omogočajo:
    - nepretrgano delovanje prostotekočega števca z možnostjo proženja prekinitve ob izenačitvi,

- ustavitev števca z možnostjo proženja prekinitve ob izenačitvi,
- ponastavitev števca ob izenačitvi, z možnostjo proženja prekinitve,
- ob izenačitvi postavitev pripadajočega izhoda bodisi na visoko oziroma nizko stanje, bodisi na stanje nasprotno prejšnjemu (ang.: toggle), ali pa ohranitev prejšnjega stanja.

Modul SSP (ang.:synchronous serial port) uporabljamo pri različnih oblikah serijskega komuniciranja. Prilagojen je za delo z vodilom SPI (ang.: serial peripheral interface), 4-žičnim SSI (ang.: synchronous serial interface) vodilom podjetja Texas Instruments in Microwire™ vodilom podjetja National Standards [24, str. 158]. V nadaljevanju opisujemo zgolj komunikacijo preko vodila SPI, ki ga uporabljamo pri povezovanju mikrokrmilnika in merilnega vezja.

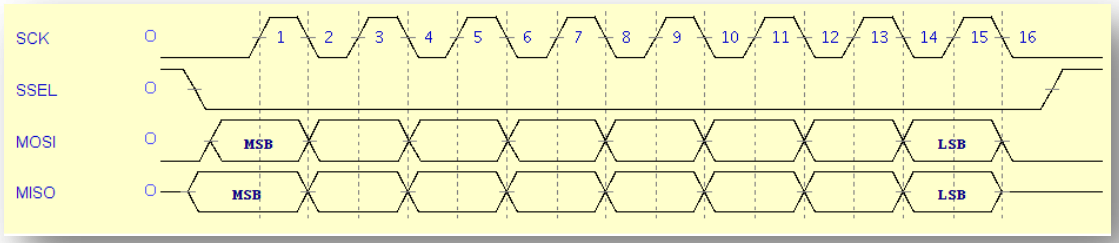
Prenos podatkov po vodilu SPI poteka med gospodarjem in podrejeno enoto z enako hitrostjo ter istočasno v obe smeri (ang.: full duplex). Zunanjo povezavo z ostalimi enotami predstavljajo štiri linije: SCK (ang.: serial clock), CS (ang.: chip select), SDI (ang.: serial data in) in SDO (ang.: serial data out). Vodilo SPI je razvilo podjetje Motorola, ki uporablja drugačne oznake za posamezne linije: SCK (ang.: serial clock), SS (ang.: slave select), MISO (ang.: master in slave out) in MOSI (ang.: master out slave in). Po liniji SCK se prenaša urin takt, z linijo SS izberemo podrejeno napravo na vodilu ter z MISO in MOSI linijami prenašamo podatke. Gospodar vodila z manipuliranjem urinega takta nadzoruje tok podatkov od sebe k izbrani podrejeni enoti in obratno. Nekatere naprave nimajo vseh zgoraj opisanih linij; manjkajo jim lahko MOSI, MISO ali SS linije.

Zanimiva lastnost vodila SPI je, da z bitoma CPOL (ang.: clock polarity) in CPHA (ang.: clock phase) določimo polariteto ter fronto proženja linije SCK. Če je bit CPOL postavljen, SCK počiva v nizkem stanju, aktivno pa je visoko stanje. Kadar bit CPOL ni postavljen, SCK počiva v visokem stanju, aktivno pa je nizko stanje [25, str. 143]. Z bitom CPHA določimo funkcijo posamezni fronti urinega takta. Največji vpliv ima bit pri pošiljanju prvega podatkovnega bita, kjer bodisi dovoli bodisi ne dovoli prehoda urinega takta pred prvim zajemanjem podatkov. Če je bit CPHA postavljen, podatke zajemamo ob zadnji fronti, sicer pa ob prednji fronti. Tabela 7 prikazuje vse možne kombinacije omenjenih bitov in njun vpliv na zajem in prenos podatkov [24, str. 159].

bit		prenos prvega podatkovnega bita	prenos preostalih podatkovnih bitov	zajemanje podatkov
CPOL	CPHA			
0	0	pred prednjo fronto SCK	zadnja fronta SCK	prednja fronta SCK
0	1	prva prednja fronta SCK	prednja fronta SCK	zadnja fronta SCK
1	0	pred zadnjo fronto SCK	prednja fronta SCK	zadnja fronta SCK
1	1	prva zadnja fronta SCK	zadnja fronta SCK	prednja fronta SCK

Tabela 7: oblika prenosa podatkov preko vodila SPI

Slika 16 podrobneje predstavlja prvo obliko prenosa, ko sta bita CPOL = 0 in CPHA = 0.



Slika 16: časovni diagram signalov za CPOL = 0 in CPHA = 0

Značilnosti vodila v neaktivnem stanju so:

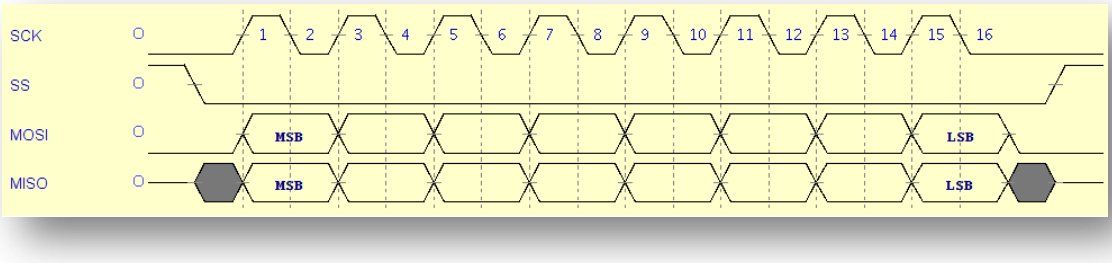
- CLK je v stanju logična 0,
- SS je v stanju logična 1 ter
- MISO/MOSI sta v stanju visoke impedance.

Prenos podatkov se začne, ko gospodar postavi linijo SS na logično 0 in so podatki v oddajnem registru veljavni. Podrejena enota dobi dostop do gospodarjeve linije MISO, prav tako pa se omogoči gospodarjeva linija MOSI. Pol urinega takta kasneje se prenesejo veljavni podatki na linijo MOSI in s tem sta obe komunikacijski liniji pripravljeni za oddajanje in sprejemanje podatkov. Še pol urinega takta kasneje (točka 1 na sliki 16) se linija SCK postavi na logično 1 in komunikacija steče. Podatki se zajemajo na prednjo in pošiljajo na zadnjo fronto.

V primeru enkratnega pošiljanja 1-zložne besede postavimo linijo SS na logično 1 prvi urin takt za zadnjim prejetim bitom. Ker je bit CPHA postavljen na logično 0, med zaporednim pošiljanjem večih zlogov preklopimo linijo SS na logično 1 in nato nazaj na logično 0 ter s tem omogočimo ponovno pisanje v oddajni register. Ko zaključimo z zaporednim prenosom

podatkov, postavimo linijo SS v neaktivno stanje prvi urin takt po zadnjem prejetem bitu [24, str. 159].

Slika 17 prikazuje drugo obliko prenosa, ko sta bita CPOL = 0 in CPHA = 1.



Slika 17: časovni diagram signalov za CPOL = 0 in CPHA = 1

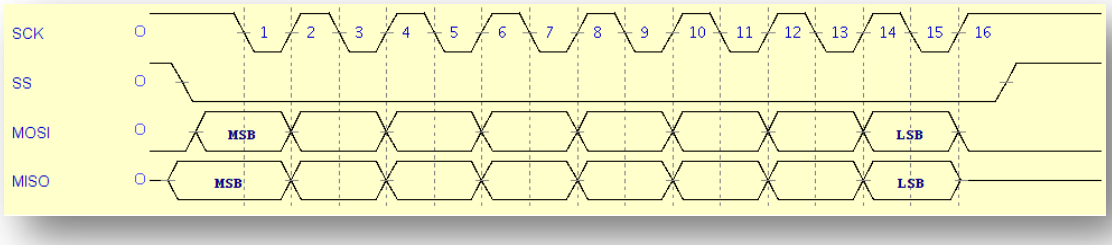
Značilnosti vodila v neaktivnem stanju so:

- CLK je v stanju logična 0,
- SS je v stanju logična 1 ter
- MISO/MOSI sta v stanju visoke impedance.

Prenos podatkov se začne, ko gospodar postavi linijo SS na logično 0 in so podatki v oddajnem registru veljavni. Podrejena enota dobi dostop do gospodarjeve linije MISO, prav tako pa se omogoči gospodarjeva linija MOSI. Pol urinega takta kasneje se prenesejo veljavni podatki na linijo MOSI oziroma MISO in s tem sta obe komunikacijski liniji pripravljeni za oddajanje in sprejemanje podatkov. Istočasno (točka 1 na sliki 17) se linija SCK postavi na logično 1 in komunikacija steče. Podatki se nato pošiljajo ob prednji fronti in zajemajo ob zadnji fronti.

V primeru enkratnega pošiljanja 1-zložne besede postavimo linijo SS na logično 1 prvi urin takt za zadnjim prejetim bitom,. Ker je bit CPHA postavljen na logično 1, med zaporednim pošiljanjem večih zlogov ni potrebno preklapljati stanja linije SS. Ko zaključimo z zaporednim prenosom podatkov, postavimo linijo SS v neaktivno stanje 1 urin takt po zadnjem prejetem bitu [24, str. 160].

Slika 18 prikazuje tretjo obliko, ko sta CPOL = 1 in CPHA = 0.



Slika 18: časovni diagram signalov za CPOL = 1 in CPHA = 0

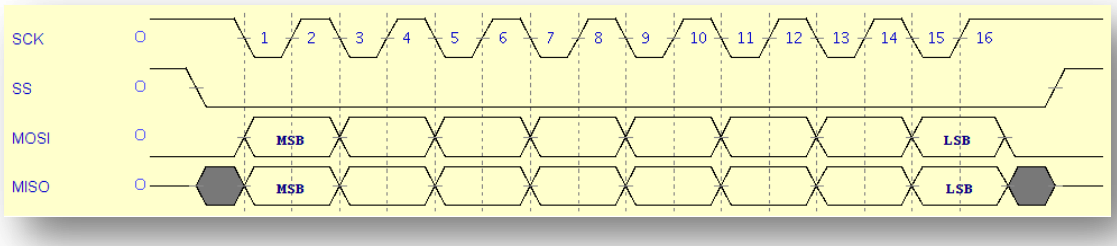
Značilnosti vodila ob neaktivnem stanju so:

- CLK je v stanju logična 1,
- SS je v stanju logična 1 ter
- MISO/MOSI sta v stanju visoke impedance.

Prenos podatkov se začne, ko gospodar postavi linijo SS na logično 0 in so podatki v oddajnem registru veljavni. Ta operacija povzroči takojšnje nalaganje podatkov podrejene enote na linijo MISO, prav tako se omogoči gospodarjeva linija MOSI. Pol urinega takta kasneje se prenesejo veljavni podatki na linijo MOSI in s tem sta obe komunikacijski liniji pripravljeni za oddajanje in sprejemanje podatkov. Še pol urinega takta kasneje (točka 1 na sliki 18) se linija SCK postavi na logično 0 in komunikacija steče. Podatki se zajemajo ob zadnji in pošiljajo ob prednji fronti.

Pošiljanje podatkov za 1-zložne in večzložne besede poteka na enak način, kot je opisano za primer CPOL = 0 in CPHA = 0 [24, str. 161].

Slika 19 prikazuje četrto obliko, ko sta CPOL = 1 in CPHA = 1.



Slika 19: časovni diagram signalov za CPOL = 1 in CPHA = 1

Značilnosti vodila ob neaktivnem stanju so:

- CLK je v stanju logična 1,
- SS je v stanju logična 1 ter
- MISO/MOSI sta v stanju visoke impedance.

Prenos podatkov se začne, ko gospodar postavi linijo SS na logično 0 in so podatki v oddajnem registru veljavni. Pol urinega takta kasneje se prenesejo veljavni podatki na liniji MOSI oziroma MISO in s tem sta obe pripravljeni za oddajanje in sprejemanje podatkov. Istočasno (točka 1 na sliki 19) linija SCK prehaja iz logične 1 v logično 0 in komunikacija steče. Podatki se zajemajo ob prednji in pošiljajo ob zadnji fronti. Pošiljanje podatkov za 1-zložne in večzložne besede poteka na enak način kot za primer CPOL = 0 in CPHA = 1 [24, str. 162].

Pri mikrokrmilniku LPC 2138 uporabnik dostopa do devetih registrov za nastavljanje modula SSP, ki jih podaja Tabela 8 [24, str. 165 - 169].

ime	opis	dostop
SSPCR0	nadzorni register 0: izbiramo med tipom vodila, velikostjo podatkov, polariteto, fazo ter dolžino urinega takta	B/P[2]
SSPCR1	nadzorni register 1: izbiramo med različnimi načini delovanja	B/P
SSPDR	podatkovni register: polni oddajni register in bere ter izprazni bralni register	B/P
SSPSR	statusni register	B
SSPCPSR	register urinega predskalirnika	B/P
SSPIMSC	maskirni register, ki omogoči štiri možne prekinitve	B/P
SSPRIS	statusni register, ki prikaže, kateri izmed možnih štirih pogojev za prekinitev je izpolnjen	B/P
SSPMIS	statusni register, ki prikaže katera izmed štirih možnih prekinitev je aktivna, prikaže samo tiste, ki so aktivirane v SSPIMSC	B
SSPICR	register, ki izbriše pripadajoče bite posameznih prekinitev	P

Tabela 8: registri modula SSP

[2] možno branje (B) in pisanje (P)

Do 47 splošno namenskih vhodno izhodnih linij je razdeljenih med dvoje vrat. Na vratih PORT 0 je od 32 priključnih linij skupno dostopno do 30 vhodno izhodnih in ena izhodna linija. Na vratih PORT 1 pa je dostopnih do 16 vhodno izhodnih linij. Oboja vrata nadziramo preko dveh skupin štirih registrov, kot prikazuje tabela 9.

ime	opis	dostop
IOPIN	register trenutnih vrednosti; iz registra preberemo vrednosti linij ne glede na to, ali gre za vhodno ali izhodno linijo	B/P
IOSET	vpis logične 1 v register postavi ustrezno linijo na visok nivo; zapis logične 0 nima nikakršnega vpliva na dogajanje na izhodu. Kadar je ustrezna linija nastavljena kot vhod, vpis logične 1 nima vpliva	B/P
IODIR	register se uporablja za določitev smeri vhodno izhodne linije, vpis logične 0 nastavi ustrezno linijo kot vhod, logične 1 pa kot izhod	B/P
IOCLR	vpis logične 1 postavi ustrezno linijo na nizek nivo. Vpis logične 0 nima vpliva na dogajanje na izhodu	P

Tabela 9: opis GPIO registrov

Opisane funkcije posameznih registrov so veljavne samo, če je ustrezna linija izbrana kot GPIO, drugače vrednosti v registrih niso reprezentativne. Katere linije so izbrane kot GPIO in katere imajo posebne funkcije, določimo v modulu PCB (ang.: pin connect block) ter registru PINSEL0 in PINSEL1 [24, str.79].

8.1.2 Integrirano vezje S02G − smart sensor

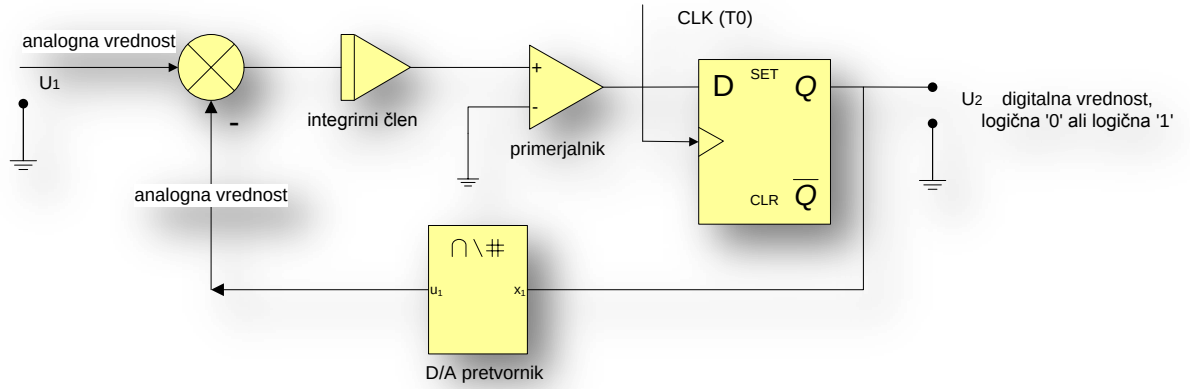
Smart sensor je integrirano vezje ASIC (ang.: application specific integrated circuits) za merjenje toka in napetosti, pri čemer za slednjo uporablja napetostni delilnik, za tokovno tipalo pa uporablja tuljavico Rogowski, tokovni transformator ali šant. Kot osnovni gradnik ga lahko vgradimo v enofazne ali večfazne sisteme [27, str.4].

Analogni del Smart sensor-ja je sestavljen iz predojačevalnika, dveh $\Delta\sum$ A/D pretvornikov prvega reda, bloka temperaturno kompenzirane napetostne reference (ang.: bandgap voltage reference), napetostnega stabilizatorja (ang.: lowdrop voltage regulator) in napetostnih predpomnilnikov. Digitalni del predstavljata generator ure in izhodni multiplekser signalov. Tipične zunanje komponente, njihove funkcije ter občutljivost posameznih tipal, ki spremljajo smart sensor, so prikazane v naslednji tabeli [27, str. 4].

komponenta	funkcija	vrednost	toleranca	Enote
SO7D kalkulator	vezje ASIC za večfazna merjenja in obdelavo signalov.			
napetostni delilnik	vmesni člen pri merjenju napetosti.	1:780	±1 %	V/V
tuljavica Rogowski	vmesni člen pri merjenju toka.	3	±12 %	mV/A
tokovni transformator	vmesni člen pri merjenju toka.	30	±12 %	mV/A
šant	vmesni člen pri merjenju toka.	0,2	5 %	mV/A

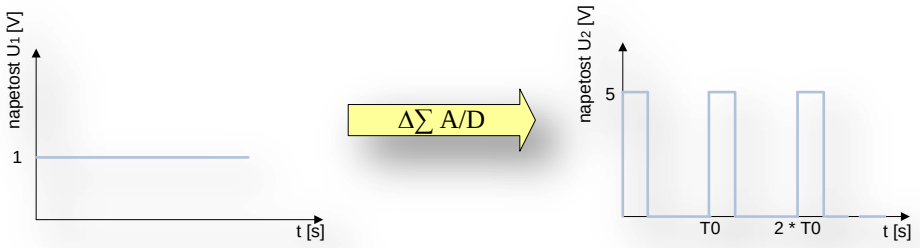
Tabela 10: spremljevalne komponente smart sensorja in njihove karakteristike

Preprosto delovanje ΔΣ A/D pretvornika prvega reda prikazuje spodnja slika.



Slika 20: preprosta shema ΔΣ A/D pretvornika prvega reda

Analogen signal odštejemo na odštevalniku ter razliko vodimo na integrirni člen. Rezultat pripeljemo na primerjalnik in nato še na pomnilno celico D, ki jo krmilimo z digitalno uro visoke frekvence (reda GHz). Izhod iz le-te vodimo na 1-bitni D/A pretvornik in preko povratne zanke na prej omenjeni odštevalnik. Tako dobimo obratovalni cikel (ang.: duty cycle) bitnega toka (ang.: bit stream), iz katerega izračunamo povprečno vrednost vzorčene veličine. Na sliki 21 prikazujemo primer, ko je napetost na vhodni strani U_1 = 1 V. Obratovalni cikel določimo tako, da v določeni periodi preštejemo logične 0 in 1 ter posamezno vrednost delimo s trajanjem periode.



Slika 21: princip delovanja ΔΣ A/D pretvorbe

Smart sensor uporablja za vsak signal (tok in napetost) svoj ΔΣ A/D pretvornik prvega reda. Karakteristike smart sensorja predstavljamo v naslednjih točkah.

Napetostna veja:

- ojačenje A/D dela $A_U = 4$,
- referenčna napetost $U_{ref} = 1{,}2$ V,
- največji dovoljeni razpon napetosti na vhodu $U_{ppmax} = 0{,}54$ V (±0,27 V) ter
- najvišja vhodna efektivna napetost $U_{effmax} = \frac{U_{ppmax}}{2\sqrt{2}} = 0{,}191$ V.

Napetostni delilnik, ki pretvori primarno napetost v napetost, primerno za vhod smartsensorja U_{eff}, načrtano tako, da ne presežemo U_{effmax}. Digitalna vrednost po A/D pretvorbi na izhodu smart sensorja $U_{\Delta\Sigma}$ prikazuje spodnja enačba:

$$U_{\Delta\Sigma} = U_{\text{eff}} \cdot \frac{A_U}{U_{\text{ref}}} \tag{4}$$

Tokovna veja pri uporabi tuljavice Rogowski:

- fiksno ojačenje $A_{pred} = 4$,
- nastavljivo ojačenje A/D prevtornika $A_{A/D}$ (2, 8),
- skupno ojačenje A je zmnožek obeh (8, 32),
- največji dovoljen razpon napetosti na vhodu U_{ppmax} je odvisen od skupnega ojačenja A (0,27 V; 0,0675 V) in
- najvišja vhodna efektivna napetost $U_{effmax} = \frac{U_{ppmax}}{2\sqrt{2}}$ (0,0957 V; 0,024 V).

Tuljavica Rogowski mora na bremenskem uporu zagotoviti pri največjem primarnem toku 180 A napetost 0,0957 V za $A = 8$ oziroma 0,024 V za $A = 32$. Tako se pokrijejo vsa tokovna območja, ki se uporabljajo pri različnih števcih. Izhodna napetost iz tuljavice zaradi odvoda prehiteva primarni tok za 90^o , prav tako je amplituda frekvenčno odvisna. Enačba (5) podaja

povezavo med primarnim tokom I_{vh}, frekvenco omrežja f in konstanto inducirane napetosti odvoda toka (mV/A) pri 50 Hz K_{ir} ter dobljeno napetostjo U_{effir}:

$$U_{\text{effir}} = I_{\text{vh}} \cdot \frac{f}{50} \cdot K_{\text{ir}} \tag{5}$$

K_{ir} je konstanta, ki se določi z razmerjem največje vhodne napetosti na A/D in primarnim tokom:

$$K_{\text{ir}} = \frac{0{,}0957\ \text{V}}{180\ \text{A}} = 0{,}533\ \frac{\text{mV}}{\text{A}}\ (pri\ 50\ \text{Hz}\ za\ A = 8) \tag{6}$$

oziroma

$$K_{\text{ir}} = \frac{0{,}024\ \text{V}}{180\ \text{A}} = 0{,}133\ \frac{\text{mV}}{\text{A}}\ (pri\ 50\ \text{Hz}\ za\ A = 32) \tag{7}$$

Digitalna vrednost $I_{\Delta\sum}$ po A/D pretvrobi napetosti, pridobljne iz odvoda toka, na izhodu smart sensorja je:

$$I_{\Delta\Sigma} \cdot \frac{f}{50} = U_{\text{effir}} \cdot \frac{A}{V_{\text{ref}}} \tag{8}$$

Izhodni multiplekser skrbi za časovno prepletanje izhodnih vrednosti in s tem zmanjšuje število potrebnih izhodnih linij.

8.1.3 Integrirano vezje SO7D – Calculator

SO7D je integrirano vezje ASIC za obdelavo bitnih tokov iz smartsensorjev.Naprava lahko deluje v enofaznih ali večfaznih sistemih samostojno ali kot podrejena enota mikrokrmilniku [21, str. 4]. Analogni del sestavljata temperaturno kompenzirana napetostna referenca in napetostni stabilizator. Digitalni del sestavljajo generator urinega takta, trije PDSP procesorji (ang.: programmable digital signal processor), NDSP (ang.: non-linear digital signal processor) procesor, komunikacijski vmesnik SPI, 112-bitna pomnilna enota OTP (ang.: one-time programmable) in osemindvajset 32-bitnih registrov. Preko vodila SPI lahko dodatno pošljemo še 16 sistemskih signalov, s katerimi kalibriramo, testiramo ali konfiguriramo merilno vezje.

Signal iz Smart sensorja gre v Calculator-ju skozi tri stopnje:

- prva stopnja:
  - časovno razpletanje na napetostni in tokovni bitni tok podatkov ter
  - kalibracija.
- druga stopnja:
  - pretvorba serijskega signala v 17-bitni paralelni signal,
  - integriranje signalov in
  - kompenzacija enosmerne komponente.
- tretja stopnja:
  - frekvenčna kompenzacija.

Delovno in jalovo moč računamo na sledeč način [21, str.6]:

$$P_1 = v_{\mathrm{u}} \cdot v_{\mathrm{iint}}\ ter\ P_2 = v_{\mathrm{uint}} \cdot v_{\mathrm{i}} \rightarrow P = \frac{P_2 - P_1}{2}$$

Rešitev zgornje enaćbe vrne rezultat:

$$P = U_{\mathrm{RMS}} \cdot I_{\mathrm{RMS}} \cdot \cos\varphi \cdot K_{\mathrm{P}} \tag{9}$$

Kjer so v_{u} napetostni signal iz prve stopnje, v_{iint} napetostni signal iz tuljavice Rogowski iz druge stopnje (17-bitni), v_{uint}, napetostni signal iz druge stopnje (17-bitni), v_{i} napetostni signal iz tuljavice Rogowski iz prve stopnje in K_{P} konstanta jalove moči, ki vsebuje konstante integrirnih členov in kalibratorja.

Jalovo moč za 50 Hz omrežje pridobimo na podoben način; pomnožimo napetostni signal iz druge stopnje in napetostni signal iz tuljavice Rogowski (17-bitni) iz druge stopnje. Po nekaj matematičnih korakih dobimo sledečo rešitev.

$$Q = U_{\mathrm{RMS}} \cdot I_{\mathrm{RMS}} \cdot \sin\varphi \cdot K_{\mathrm{Q}} \tag{10}$$

K_{Q} je konstanta jalove moči, ki vsebuje konstante integrirnih členov in kalibratorja.

V Calculatorju se dobljena moč iz (9) in (10) integrira in izračunava energija. Integralni člen je realiziran kot števec gor/dol (ang.: up/down counter). Ker le-ta lahko preplavi, mora zunanja aplikacija poskrbeti, da odčitava energijske prirastke dovolj pogosto, da zazna preplavitev števca.

Calculatorju lahko nastavljamo ali permanentno zapečemo 112 konfiguracijskih bitov. Tiste bite, ki jih uporabljamo pri naši merilni platformi, opisujemo v poglavju 9. Dodatno ima

Calculator še 16 sistemskih signalov, ki omogočajo različne akcije; ponastavitev Calculatorja in zapis oziroma permanentni zapis v blok celic OTP. Uporabljene signale prav tako predstavljamo v poglavju 9.

Za komunikacijo preko SPI je v Calculatorju na voljo pet sponk, katerih pomen prikazuje tabela 11 [34]:

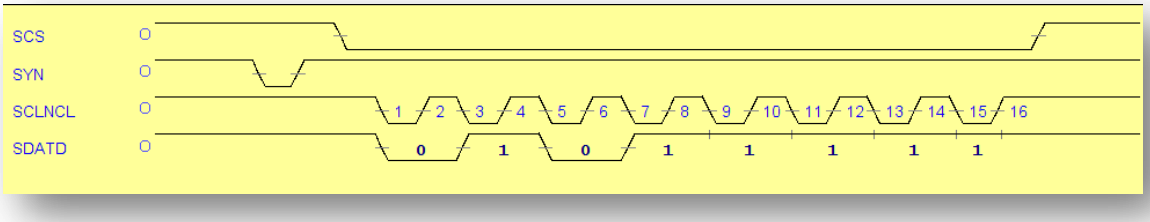
sponke	pomen
SCS	če je na liniji logična 0 je SPI omogočen, drugače je onemogočen
SYN	izbiramo smer SDATD linije; logična 0 pomeni vhodno sponko, logična 1 izhodno sponko
SCLNCL	vhodna sponka za urin takt
SDATD	vhodna sponka za 1-zložen vhodni stavek oziroma izhodna sponka za 32-bitni serijski podatek
VOTP	analogna linija; nanj pripeljemo tokovni vir za trajni vpis v celice OTP

Tabela 11: pomen sponk modula SPI pri Calculatorju

Preko vodila SPI lahko:

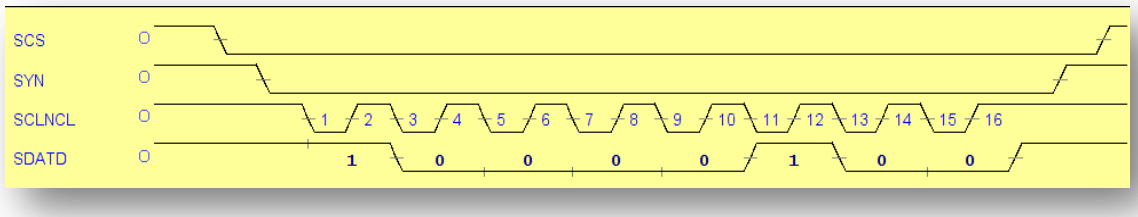
- beremo vrednosti osemindvajsetih 32-bitnih registrov,
- vpisujemo in brišemo 112 začasnih konfiguracijskih bitov oziroma 16 sistemskih signalov in
- trajno vpisujemo v 112 celic OTP.

Pri branju registrov iz Calculator-ja imamo dve možnosti; preberemo lahko sveže podatke ali stare. Če želimo brati prve, zamrznemo podatke v registrih z aktivnim pulzom dolžine vsaj 500 ns na liniji SYN (iz logične 1 v logično 0 in nazaj v logično 1) in visokim nivojem na liniji SCS. Branje svežih podatkov prikazuje slika 22. Uporabljene registre opisujemo v poglavju 9.



Slika 22: branje prvega zloga (0x5F) iz Calculator-ja prek vodila SPI

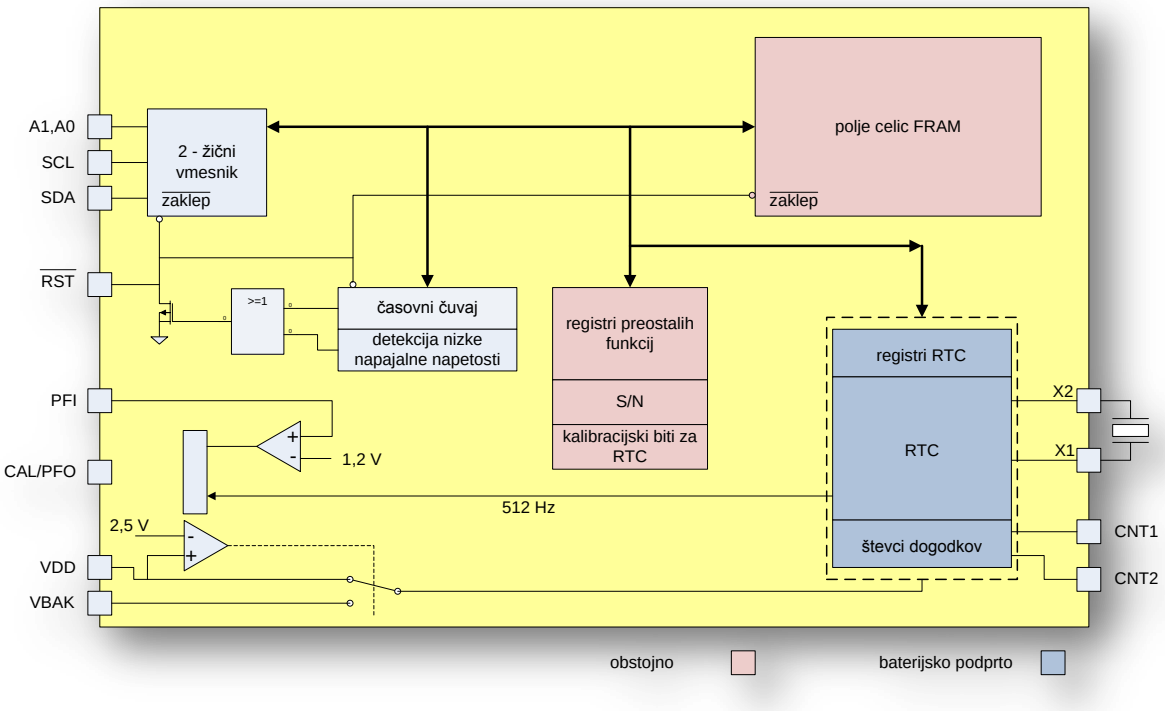
Pri pisanju 1-zlogovnega podatka oziroma ukaza postavimo linijo SCS na logično 0, nato pa še linijo SYN. Slika 23 prikazuje vpis logične 1 v konfiguracijski bit na naslov 4.



Slika 23: pisanje logične 1 na naslov 4

8.1.4 Integrirano vezje FM31256

Integrirano vezje FM31256 vsebuje več različnih naprav. Poleg obstojnega pomnilnika FRAM (ang.: fero-electric random access memory), vsebuje še uro realnega časa in spremljevalno vezje mikrokrmilnika (ang.: processor companion) z vezjem časovnega čuvaja, dvema baterijsko podprtima števcema dogodkov, napetostnim primerjalnikom in vezjem za detekcijo nizke napajalne napetosti. Komunikacija z mikrokrmilnikom poteka preko vodila I2C. Slika 24 prikazuje glavne module vezja [26, str. 3].



Slika 24: blokovni diagram integriranega vezja FM31256

Opis posameznih priključnih sponk predstavljamo v tabeli 12.

sponka	tip	opis
A0, A1	vhodna	liniji za izbiro naprave uporabljamo pri naslavljanju večih integriranih vezij FM31256
CNT1, CNT2	vhodna	vhoda za števca dogodkov, polariteto je možno programsko nastaviti
CAL/PFO	izhodna	v kalibracijskem načinu delovanja je na liniji prisoten signal 512 Hz za kalibracijo RTC, drugače je to izhodna linija za zgodnjo detekcijo izpada napajanja
X1,X2	vhodno izhodna	liniji za priklop kristala s frekvenco 32.768 kHz; namesto njega lahko uporabimo zunanji oscilator, ki ga priključimo na sponko X2, medtem ko sponko X1 pustimo v zraku
$\overline{RST}$	vhodno izhodna	vhod za ponovni ročni zagon vezja ali izhod za ponovni zagon nanj povezane naprave
SDA	vhodno izhodna	serijska komunikacija: prenos podatkov in naslovov
SCL	vhodna	serijska digitalna ura za I^2C komunikacijski vmesnik
PFI	vhodna	nanjo pripeljemo neregulirano napetost za zgodnjo detekcijo izpada napajanja
VBAK	napajalna	pomožno baterijsko napajanje ali napajanje preko kondenzatorja
VDD	napajalna	regulirana napajalna napetost
VSS	napajalna	masa

Tabela 12: opis priključnih sponk integriranega vezja FM31256

Pomnilnik FRAM velikosti 256 kB je zlogovno organiziran. Tehnologija FRAM omogoča izredno hitra branja in pisanja ter obstojnost podatkov po odvzemu napajanja, zato lahko ta pomnilnik, poleg trajnega shranjevanja podatkov, uporabljamo tudi kot delovni pomnilnik. Komunikacija z mikrokrmilnikom poteka preko dvožičnega vmesnika oziroma vodila I^2C.

Vezje za detekcijo nizke napajalne napetosti ima nastavljivo mejo za zaznavanje padca napetosti. V primeru, da napetosti to mejo prekorači, vezje FM31256 postavi RESET linijo na

nizek nivo in s tem ne dovoli zagona mikrokrmilnika. Ob ponovni vzpostavitvi napajanja preide nivo na liniji iz nizkega v visoko stanje po pretečenih 100 ms.

Splošni primerjalnik v števcu MT372-SMART uporabljamo za zgodnjo detekcijo padca napetosti. Nanj preko napetostnega delilnika pripeljemo neregulirano napetost in jo primerjamo z 1,2 V. Ob padcu napetosti pod to vrednost, se proži signal na liniji PFO.

Ura realnega časa je podprta s pomožnim napajanjem, tako da deluje tudi, ko pride do izpada glavnega napajanja. Ura deluje s taktom 32,768 kHz in omogoča programsko kalibracijo. Števca dogodkov štejeta spremembe na liniji. V števcu MT372-SMART ju uporabljamo za detekcijo odprtja pokrova in priključnice števca.

8.1.5 Komunikacijski vmesniki

Števec MT372-SMART podpira optični, GSM/GPRS ter RS485 komunikacijski vmesnik. *Optični vmesnik* je namenjen za lokalno odčitavanje podatkov ali nastavljanje parametrov. Odvisno od želje kupca sta na voljo dva fizična vmesnika [30]:

- IEC optični vmesnik (ang.:IEC optical interface) ter
- ANSI optični vmesnik (ang.: ANSI optical interface), kjer sta zamenjani poziciji diod IR in PIN (physical position of IR and PIN diodes is switched).

GSM/GPRS vmesnik ima vgrajen modem podjetja Wavecom, ki podpira štiri frekvenčna področja (ang.: quad-band) 900/1800 MHz in 850/1900 MHz [31]. Preko GSM/GPRS vmesnika je možno pridobivanje informacije o porabljeni električni energiji, branje posameznih registrov, odklop uporabnika, če ima vgrajen odklopni modul, in prepis programske opreme v števcu. Kupec števcev pred izdelavo pošlje pripadajoče SIM kartice, ki se jih nato pri montaži vstavi v števce. Slika 25 prikazuje uporabljeni modem.

Slika 25: modem Wavecom Q24NG

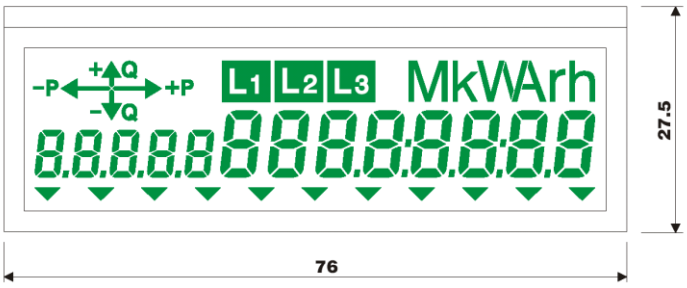
RS-485 vmesnik uporabljamo, ker so števci z GSM/GPRS modemom dragi. Tako lahko zaporedoma vežemo več takih števcev z komunikatorjem P2CC, ki ima vgrajen modem.

8.1.6 Napajalnik

Napajalnik vsebuje elektronski transformator (ang.: switcher). Izveden je tako, da dovaja le toliko energije, kolikor jo števec v danem trenutku potrebuje. Groba regulacija napetosti je izvedena s transformatorjem, ki mu je prenos energije omejen v skladu s trenutnimi potrebami. Za to skrbi vezje VIPER12A [28]. Fina regulacija napetosti na 4,2V se izvaja z vezjem MC33063AD [29].

8.1.7 Konzola

Konzola je sestavljena iz prikazovalnika LCD, dveh tipk in dveh diod LED. Slednji sta namenji preverjanju točnosti števca (slika 26). Prva prikazuje kvant delovne energije, druga pa kvant jalove oziroma navidezne energije. Število impulzov je nastavljivo, navadno pomeni en impulz 1 Wh energije [30, str. 46]. Če posamezna dioda sveti nepretrgoma ni porabe pripadajoče energije.



Slika 26: sedem segmentni prikazovalnik LCD

Sedem segmentni prikazovalnik LCD na sliki 26 vsebuje v spodnjem levem kotu pet števk velikosti 6 mm, ki najavljajo kodo prikazanih objektov na osmih števkah velikosti 8 mm. Dodatni simboli so še [30, str. 47]:

- smer pretoka energije za delovno in jalovo energijo,
- indikator prisotnosti posameznih faz,
- merske enote prikazanih podatkov na osmih števkah ter
- enajst indikatorjev različnih stanj števca na dnu prikazovalnika.

Modra tipka je namenjena navigaciji po menijih in izvajanju različnih ukazev (priklop uporabnega mesta po odklopu odklopnika, …). Oranžna tipka je zapečatena, saj omogoča operacije, ki jih izvajajo samo pooblaščene osebe v izjemnih primerih.

8.1.8 Vhodi

Vhod alarm ob zaznavi prisotnosti napetosti 24 V izvede predprogramirano operacijo. To je lahko beleženje alarma v register ali klic na ustrezno telefonsko številko. Števec je lahko opremljen še z dvema impuznima SO vhodoma. Kratke stike na vhodnih sponkah SO vhoda, ki jih povzročajo izhodi drugih števcev (plinski, toplotni), interpretira kot impulze in tako beleži njihovo porabo, ki jo nato prikaže na prikazovalniku. To rešitev je nadomestila komunikacija preko vodila M-Bus (ang.: meter bus) [30].

Vodilo omogoča priklop do štirih podrejenih naprav. Danes je to največkrat plinski števec, lahko pa priklopimo tudi vodne in toplotne števce. Komunikacijske lastnosti vodila M-Bus v števcu so [30, str. 146]:

- Hitrost prenosa podatkov 2400 baud,
- Števec električne energije je gospodar vodila, ostale enote so podrejene,
- Aplikacijski sloj sledi standardu EN 13757-3 ter
- Fizični ter povezovalni sloj sledita standardu EN 13757-2.

8.1.9 Izhodi

Števec ima en relejski izhod, ki je sposoben preklapljati breme do 250 V, 6 A ter en OPTOMOS rele izhod, ki zmore preklapljati manjša bremena do 250 V, 100 mA. Tretji izhod krmili zunanji odklopni modul, ki se lahko števcu doda na željo kupca. Z njim lahko ponudnik električne energije na daljavo v vsakem trenutku porabniku prekine dostop do

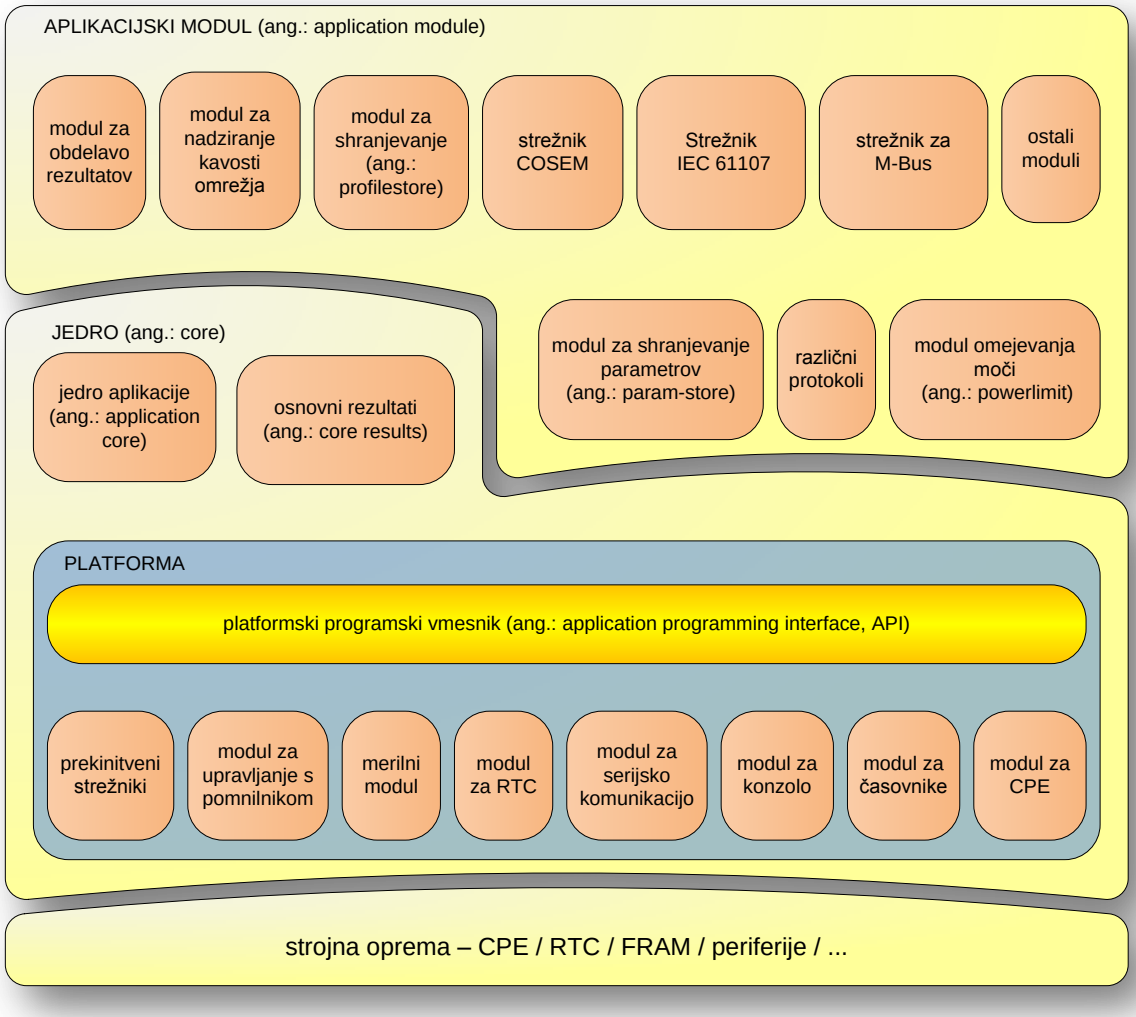
električne energije ali mu ga, preko omejevalnih funkcij v števcu (ang.: powerlimit), omeji na poljubno majhno vrednost [30].

8.2 Koncept programske opreme

Standard [32] zahteva, da je vsa programska oprema overjena s strani pristojnih organov. Dopušča tudi možnost razdelitve programske opreme na zakonodajno odvisno in zakonodajno neodvisno, pri čemer slednja ne sme vplivati na prvo. Tako je programska oprema v števcu MT372-SMART sestavljena iz dveh delov:

- jedra (ang.: core) ter
- aplikacijskega modula (ang.: application module).

Slika 27 prikazuje razdelitev programske opreme v števcu.



Slika 27: razdelitev programske opreme v števcu MT372-SMART

Jedro, ki predstavlja zakonodajno odvisno programsko opremo, vsebuje vse platformske module, njen programski vmesnik ter vse potrebno za minimalno avtonomno delovanje (merjenje energije in shranjevanje podatkov v varne kopije, vendar brez napredne obdelave podatkov). Aplikacijski modul, ki predstavlja zakonodajno neodvisno opremo, vsebuje dodatne funkcionalnosti, kot so: nadzor kakovosti omrežja, omejevanje moči, različni komunikacijski protokoli, registri obremenilne krivulje, obdelava rezultatov, vklapljanje različnih tarifnih območij in drugo. Ko se preveri integriteta in skladnost jedra ter aplikacijskega modula, števec začne delovati s polno funkcionalnostjo. Sedaj ima aplikacijski modul dostop do platformskega programskega vmesnika in posredno do strojne opreme. V števcu se sekvenčno odvija deset glavnih procesov, od tega se dva izvajata v jedru, ostalih osem pa v aplikacijskem modulu.

Tabela 13 opisuje vse omenjene procese [33].

ime procesa	lokacija	opis
platformprocess_run	jedro	proces izvaja vse nedeterministične podprocese in diagnostiko merilnega dela
resultscore_run	jedro	proces dostopa preko merilnega API do merilnih rezultatov porabljene energije in jih shranjuje v pripadajoče registre, prav tako periodično shranjuje podatke v varne kopije v FRAM
systemprocess_run	modul[3]	proces nadzira delovanje števca; zbira podatke o stanju ostalih procesov in proži ustrezne dogodke in posreduje sporočila o napakah v delovanju
consoleprocess_run	modul	proces omogoča prikaz merilnih rezultatov, različnih statusov in parametrov na prikazovalniku LCD ter sprejemanje preprostih ukazov preko konzolnih tipk
serialprocess_run	modul	Process omogoča komunikacijo med centrom ali dlančnikom za odčitavanje števcev; deluje na principu odjemalec-strežnik
resultsprocess_run	modul	proces obdela podatke, pridobljene preko resultscore_run procesa in merilnega dela; podatki se periodično hranijo v varnih kopijah v FRAM
powerquality_run	modul	proces nadzira kakovost dobavljene električne energije; zbira podatke o nihanju napetosti, asimetrični napetosti, največjih in najmanjših vrednostih napetosti
profilestore_run	modul	proces periodično shranjuje podatke v obstojni krožni pomnilnik in preverja njihovo veljavnost
iocontrol_run	modul	proces nadzira dodatne vhodne in izhodne sponke ter zunanje naprave, ki so povezane s števcem (odklopnik, nadzor bremena in alarmni vhodi)
prepay_process_run	modul	proces omogoča predplačniške funkcije v števcu

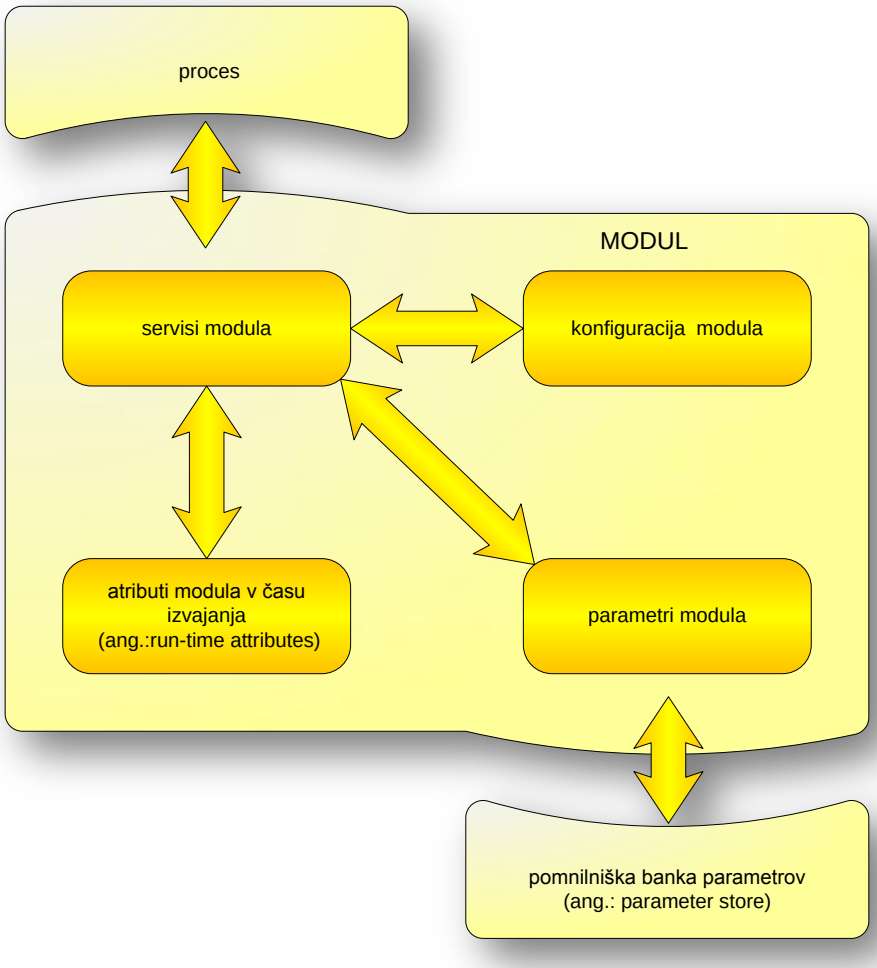
Tabela 13: procesi v števcu MT372-SMART

[3] v tej tabeli modul predstavlja aplikacijski modul

Procesi dostopajo do več kot oseminštiridesetih različnih modulov. Le-ti imajo generično zasnovo, ki jo prikazujemo na sliki 28; sestavljeni so iz:

- parametrov,
- konfiguracije,
- različnih atributov v času izvajanja ter
- servisov.

Prenastavljeni parametri se hranijo v bliskovnem pomnilniku ter se ob zagonu prenesejo v delovni pomnilnik in v pomnilnik FRAM. Spremenjeni parametri se kasneje preko vmesniških funkcij shranjujejo v pomnilniški banki parametrov (ang.: parameter store), ki se nahaja v pomnilniku FRAM.



Slika 28: osnovna sestava modulov števca MT372-SMART

Osnovna konfiguracija modula se izvrši v času prevajanja programa, ki stopi v veljavo pri zagonu in inicializaciji števca. Vsak modul ima svoje atribute, ki hranijo podatke o stanju modula in vrednosti posameznih registrov. V času izvajanja dostopamo do njih preko servisov

modula. Le-ti so tudi programski vmesnik za procese, ki želijo dostopati do atributov ali parametrov posameznega modula.

Programska oprema je pisana modularno v programskem jeziku C. Vsak modul ima pripadajočo C in h-datoteko. Slednja vsebuje deklarirane vse javne funkcije in spremenljivke, medtem ko prva vsebuje njihove definicije in definicije vseh privatnih funkcij ter spremenljivk modula. Konfiguracijska tabela aplikacijskega modula določa, kateri moduli se bodo prevajali in kateri ne. S tem zagotovimo prilagodljivost pri številu različnih programskih oprem, ki jih lahko ponudimo tržišču.

9 Implementacija merilnega dela

Cilj razvoja je zanesljiv ter modularno grajen merilni sistem, ki ga je moč z manjšimi prilagoditvami uporabiti v celotni merilni družini Iskraemecovih sistemskih števcev električne energije. Poseben poudarek namenjamo implementaciji varne programske opreme. Na podlagi izkušenj predhodnega merilnega sistema in metode PHA (poglavje 4) zasnujmo tabelo 14, s katero ovrednotimo možne hazarde.

hazard	oznaka	stopnja resnosti	Opis
nepravilna inicializacija	**a**	**4**	pomanjkljiva ali nepravilna inicializacijska pot vodi v nepravilno delovanje števca
napaka pri komunikaciji z merilnim vezjem	**b**	**9**	zaradi motenj lahko pride do napake pri komunikaciji med merilnim vezjem in mikrokrmilnikom, napačni rezultati lahko vodijo v izklop merilnega mesta
nepravilna kalibracija	**c**	**7**	števec je lahko napačno kalibriran, ker se vrednosti permanentno vpišejo v celice OTP, je izdelek neuporaben
nepravilna konfiguracija	**d**	**3**	v števec lahko vpišemo napačne konfiguracijske vrednosti in s tem onemogočimo normalno delovanje
nepravilno delovanje vmesnika - osveževanje	**e**	**8**	nezmožnost vmesnika, da osveži podatke
nepravilno delovanje vmesnika – časovna neustreznost	**f**	**8**	vrednosti se osvežujejo v napačnih časovnih trenutkih

Tabela 14: tabela ovrednotenih hazardov

V nadaljevanju upoštevamo življenjski cikel, opisan v poglavju 3.7.

9.1 Analiza zahtev

Merilne specifikacije oziroma zahteve za elektronski števec MT372-SMART so:

- merjenje v razredu točnosti 2,
- merjenje delovne, jalove in navidezne energije,
- merjenje smeri pretoka energije,
- podajanje efektivne vrednosti napetosti in toka ter
- merjenje oziroma podajanje trenutne delovne moči.

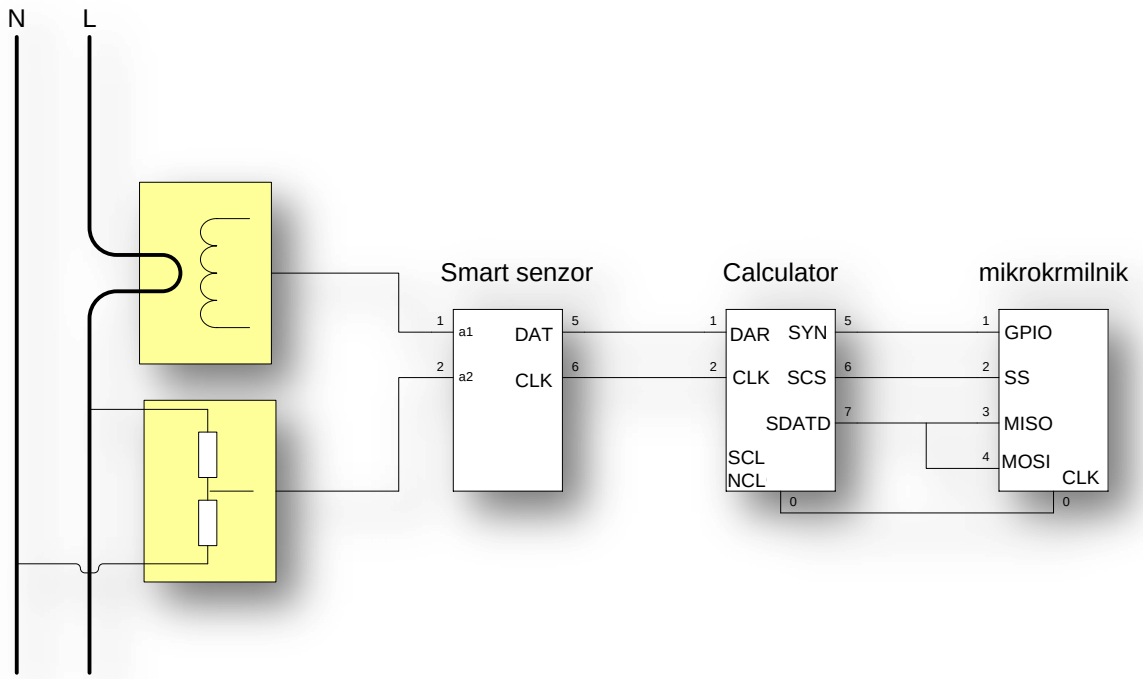
Zahteve opredeljujejo tudi modularno izgradnjo merilnega dela, ki omogoča enostavno povezovanje s preostalimi moduli. V ta namen je razvit in dokumentiran merilni API, ki poskrbi za ločitev strojnega dela od preostale programske opreme. Gonilnik za osrednje merilno vezje mora podpirati avtomatizirano kalibracijo in konfiguracijo Calculator-jevih parametrov ter branje in posredovanje svežih podatkov na 32 ms.

9.2 Načrtovanje in razvoj sistema

Strojni merilni del elektronskega števca MT372-SMART sestavljajo sledeče komponente:

- merilna tuljavica Rogowski,
- pametno tipalo in A/D pretvornik Smart sensor,
- osrednje merilno vezje SO7D Calculator in
- mikrokrmilnik NXP-jeve družine LPC2138.

Slika 29 prikazuje povezave med posameznimi komponentami enofaznega sistema. Trifazni sistem ima ustrezno več Smart sensor-jev.



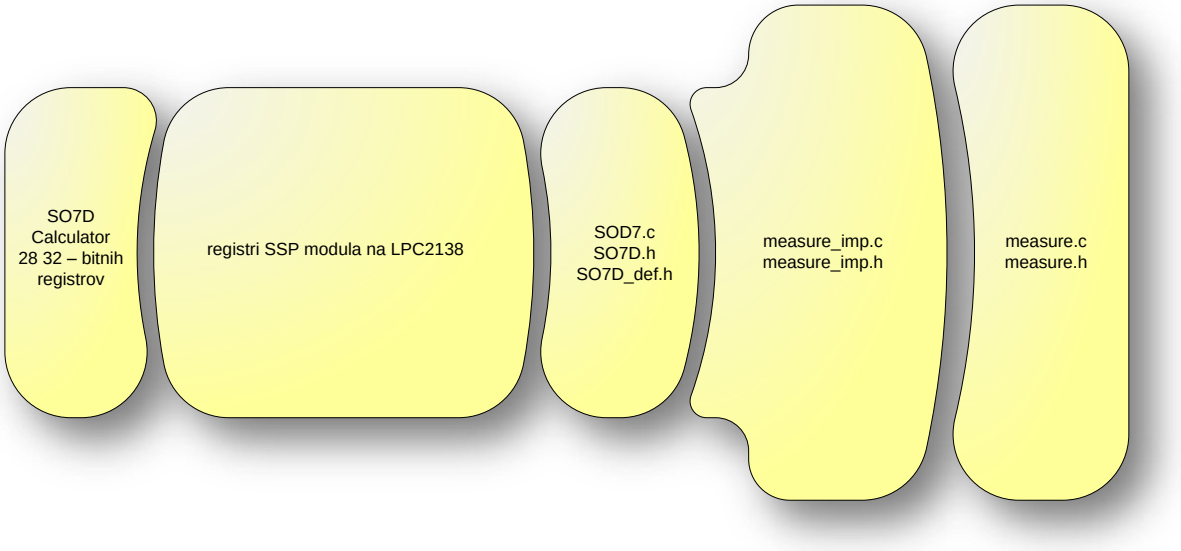
Slika 29: povezave med posameznimi merilnimi komponentami

Tuljavica Rogowski meri spremembo inducirane napetosti, ki je premosorazmerna spremembi toka skozi ovoj. Za vsako fazo sta namenjeni dve tuljavici Rogowski; prvo uporabimo za merjenje energije, drugo pa za merjenje motenj in kasnejšo kompenzacijo signala. Napetost pripeljemo na vhod Smart sensor-ja preko napetostnega delilnika. V Smart sensor-ju se analogne vrednosti pretvorijo v $\Delta\sum$ signale (DAT) in pošljejo Calculator-ju (DAR), kjer se nadaljnje obdelajo. Potek računanja moči in energije je opisan v poglavju 8.1.3. Po obdelavi so vrednosti dostopne v 28 32-bitnih registrih, do katerih lahko dostopamo preko vodila SPI.

Pri načrtovanju smo se odločili za programski jezik C, ker omogoča kompromis med fleksibilnostjo in močjo nizko nivojskih jezikov ter modularnostjo visoko nivojskih jezikov. Tako je naša programska oprema merilnega dela razvita modularno in razdeljena na sledeče module in pripadajoče datoteke programskega jezika C:

- gonilnik za SO7D Calculator:
  - SO7D.c,
  - SO7D.h ter
  - SO7D_def.h
- merilni podsistem, ki obdela podatke:
  - measure_imp.c in
  - measure_imp.h
- merilni API:
  - measure.c in
  - measure.h.

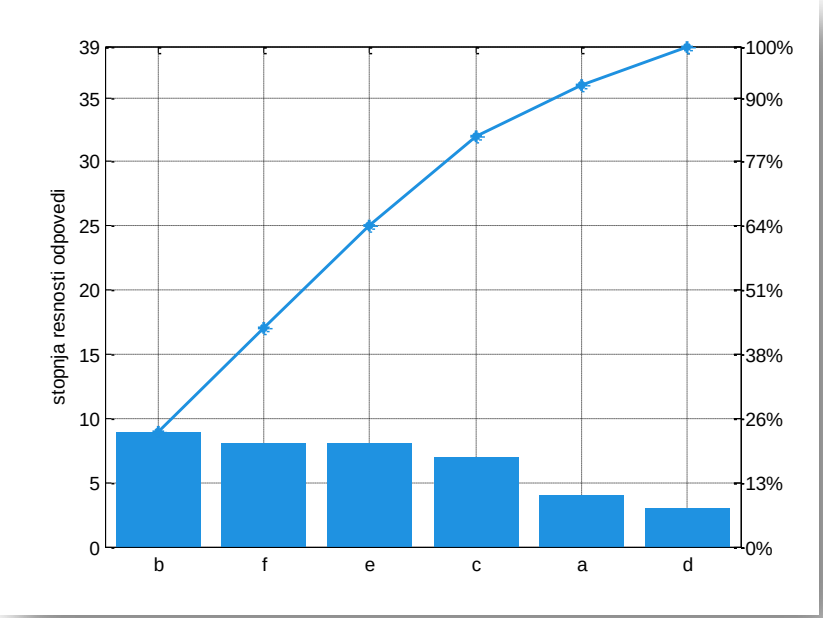
Slika 30 prikazuje merilni del s programskega gledišča.



Slika 30: povezave med posameznimi programskimi deli merilnega dela

S stališča izvajanja se deli programska oprema na tri dele. Prvi je inicializacija merilnega dela na vseh nivojih (SO7D.c, mesaure_imp.c), drugi je glavni merilni proces (vsebuje branje in obdelavo podatkov iz merilnih integriranih vezij, verifikacijo prebranih podatkov, krmiljenje diod LED, ...), ki se izvaja znotraj prekinitvene rutine, medtem ko tretji del predstavljajo pomožne funkcije, ki se izvajajo izven prekinitvene rutine, vendar vseeno vplivajo na delovanje merilnega procesa (npr.: kalibracija, inicializacija, ...).

Ker v tem delu ne opravimo analize po metodi SWFMEA, je naše število RPN sestavljeno samo iz stopnje resnosti odpovedi (tabela14). Na podlagi rezultatov izdelamo graf Pareto in se odločimo za tiste hazarde, ki jih poskušamo najbolj nevtralizirati. Slika 31 prikazuje rezultate analize PHA.



Slika 31: graf Pareto za ugotovljene hazarde

Največ pozornosti pri razvoju programske opreme namenimo hazardom z oznakami *b* – napaka pri komunikaciji z merilnim vezjem, *f* – nepravilno delovanje vmesnika – časovna neustreznost, *e* – nepravilno delovanje vmesnika – osveževanje podatkov, in *c* – nepravilna kalibracija.

9.3 Razvoj programske opreme

V modulu SO7D komuniciramo z merilnim vezjem preko funkcij API generičnega modula SPI, ki kliče API funkcije modula SSP. Le-ta ima dostop do strojnih registrov modula SSP in tako sproži začetek komunikacije. Uporabljene funkcije so opisane v spodnji tabeli.

funkcija	modul	Opis
`SPI_setup`	SPI	kliče ustrezno funkcijo bodisi iz modula LPC21xx_SPI bodisi iz modula LPC21xx_SSP
`SPI_status`	SPI	kliče ustrezno funkcijo bodisi iz modula LPC21xx_SPI bodisi iz modula LPC21xx_SSP
`SPI_transfer`	SPI	kliče ustrezno funkcijo bodisi iz modula LPC21xx_SPI bodisi iz modula LPC21xx_SSP
`LPC21XX_SSP_setup`	SSP	nastavi modul SSP; začetno hitrost, izbrano obliko vodila (SPI) ter izbira polarite in fronto proženja
`LPC21XX_SSP_status`	SSP	vrne trenutno stanje vodila; vodilo zasedeno, vodilo prosto
`LPC21XX_SSP_transfer`	SSP	prenaša sporočila med merilnim vezjem in mikrokrmilnikom; dostop do strojnih registrov modula SSP; vpis v register `SSPDR` za začetek komunikacije in vpis v `SSPIMSC` za vklop prekinitev

Tabela 15: Funkcije API za komuniciranje preko SPI vodila

Komunikacija poteka preko sporočil SPI_MESSAGE, ki si jih moduli podajajo. Le-ta so sestavljena iz naslova prejemnika sporočila, števila zlogov, ki jih prenašamo, ter pomnilniškega naslova, kamor naj se sporočilo shrani, oziroma od kje naj se vsebina pošlje. Funkcije, ki si na različnih slojih podajajo sporočila, imajo v imenu zapisano oznako "transfer". Vse funkcije modula SSP vračajo statuse: `SSP_ERROR` (0xFF), `SSP_OK` (0x00) in `SSP_BUSY` (0x01), s čimer signalizirajo stanja operacije.

9.3.1 Funkcijski bloki modula SO7D

V modulu SO7D je prisotnih 15 funkcij (tabela 16), s katerimi dovoljujemo višjim modulom manipulacijo z merilnim vezjem.

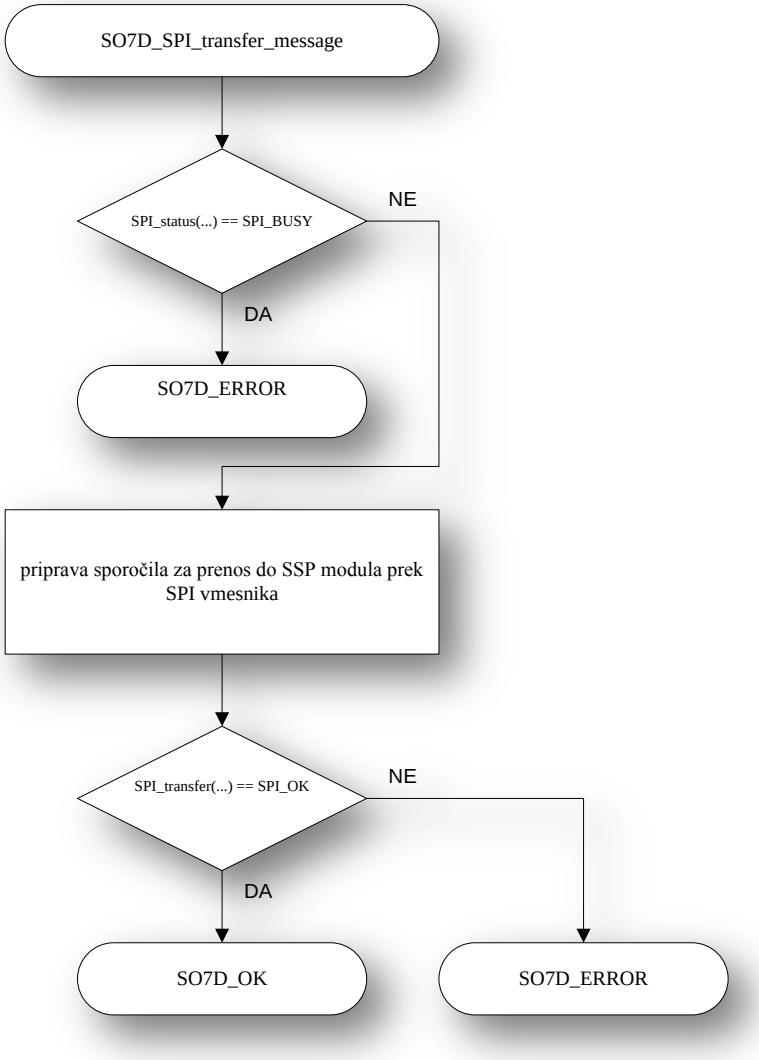
funkcija	opis
SO7D_init	inicializacija merilnega vezja
SO7D_parity_check	preverjanje parnosti na podlagi polzloga (ang.: parity nibble)
SO7D_configurator_clear	brisanje vseh konfiguracijskih bitov
SO7D_read	branje 28-tih 32-bitnih registrov Calculator-ja
SO7D_write	vpis enega izmed 16-ih posebnih signalov ali konfiguracijskih bitov
SO7D_read_registers	dvakratno zaporedno branje podatkov in prenos le-teh iz Calculator-jevih registrov v delovni pomnilnik mikrokrmilnika
SO7D_get_energy_delta_long	računanje energijskega kvanta, saj je merjenje energije realizirano s števci gor-dol
SO7D_configure	nastavljanje konfiguracijskih bitov
SO7D_tstd_programmed	preverjanje, ali je bil bit TSTD vpisan, če je bil, pisanje v celice OTP ni več možno
SO7D_configurator_program_latches	postopek za vpis konfiguracijskih bitov
SO7D_program_tstd	poseben postopek za zapis bita TSTD
SO7D_get_calibration_values	vračanje kalibracijskih bitov za posamezno fazo
SO7D_set_calibration_values	vpis kalibracijskih bitov za posamezno fazo
SO7D_calibrate	kalibracijski postopek za Calculator

Tabela 16: funkcije gonilnika SO7D, vidne višjim modulom

Najosnovnejša funkcija je SO7D_SPI_transfer_message, s katero omogočamo povezavo z moduli nižjih slojev. S funkcijama SO7D_read in SO7D_write pripravimo vse potrebno za branje oziroma pisanje vseh registrov merilnega vezja. Slednji funkciji uporabljajo vsi ostali funkcijski bloki modula SO7D in funkcijski bloki višjih slojev. Zaradi varnosti vgradimo v funkcijske bloke modula SO7D sledeče statuse, ki zagotavljajo informacijo o tem, kako se je določena operacija izvedla: SO7D_ERROR (0xFF), SO7D_OK (0x00), SO7D_INVALID_POINTER

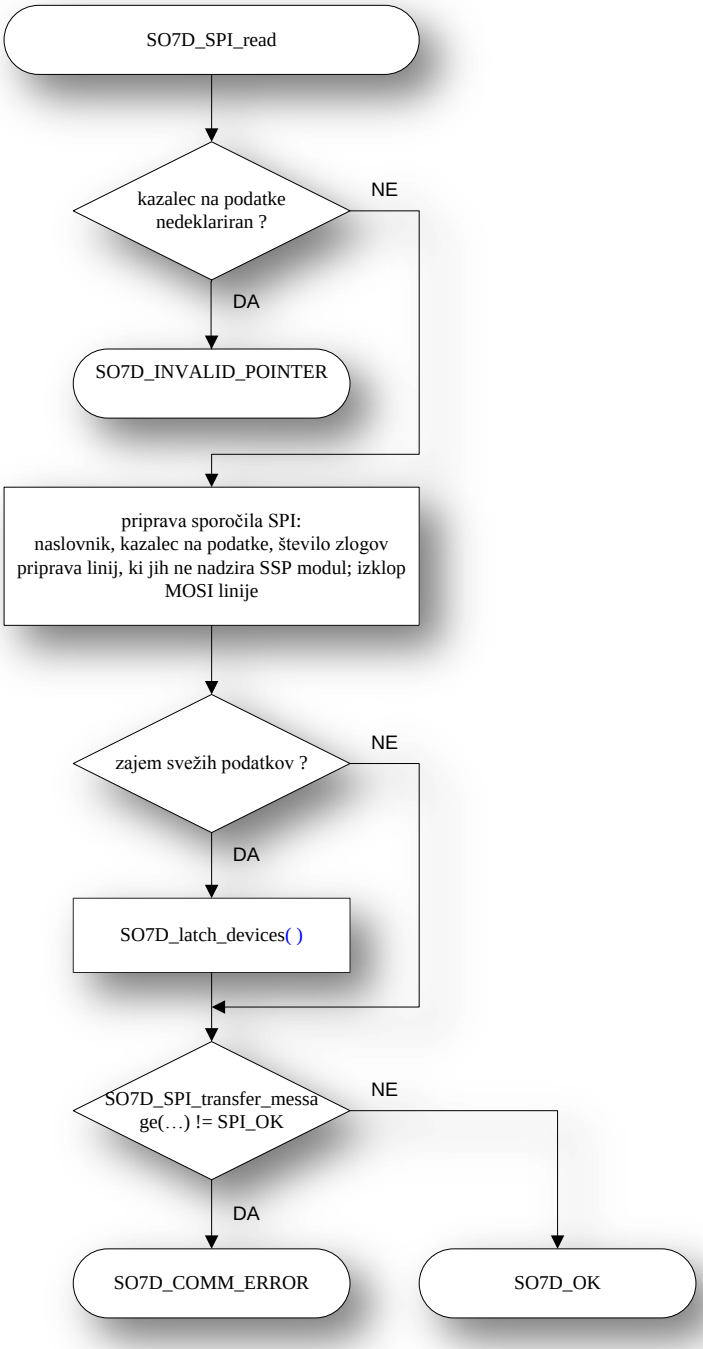
(0x01), `SO7D_PARITY_ERROR` (0x02), `SO7D_COMM_ERROR` (0x03), `SO7D_NOT_IMPLEMENTED` (0x04), `SO7D_CALI_ERROR` (0x05), `SO7D_OTP_FATAL` (0x06).

V nadaljevanju predstavljamo bločne diagrame osnovnih funkcij komunikacije preko vodila SSP. Tako na sliki 32 predstavljamo bločni diagram funkcije `SO7D_SPI_transfer_message`. Če je vodilo zasedeno, prekinemo izvajanje rutine in vrnemo status `SO7D_ERROR`. V nasprotnem primeru pripravimo sporočilo in ga pošljemo nižjemu nivoju (modulu SPI in ta modulu SSP). Po opravljenem prenosu prvega zloga dobimo odgovor `SO7D_OK` oziroma `SO7D_ERROR`. Če prenašamo več sporočil (več zlogov), se vsa nadaljnja sporočila prenašajo preko prekinitvenega strežnika. Glavna zanka se tako nemoteno odvija dalje, medtem ko modul SSP sprejema oziroma oddaja podatke.



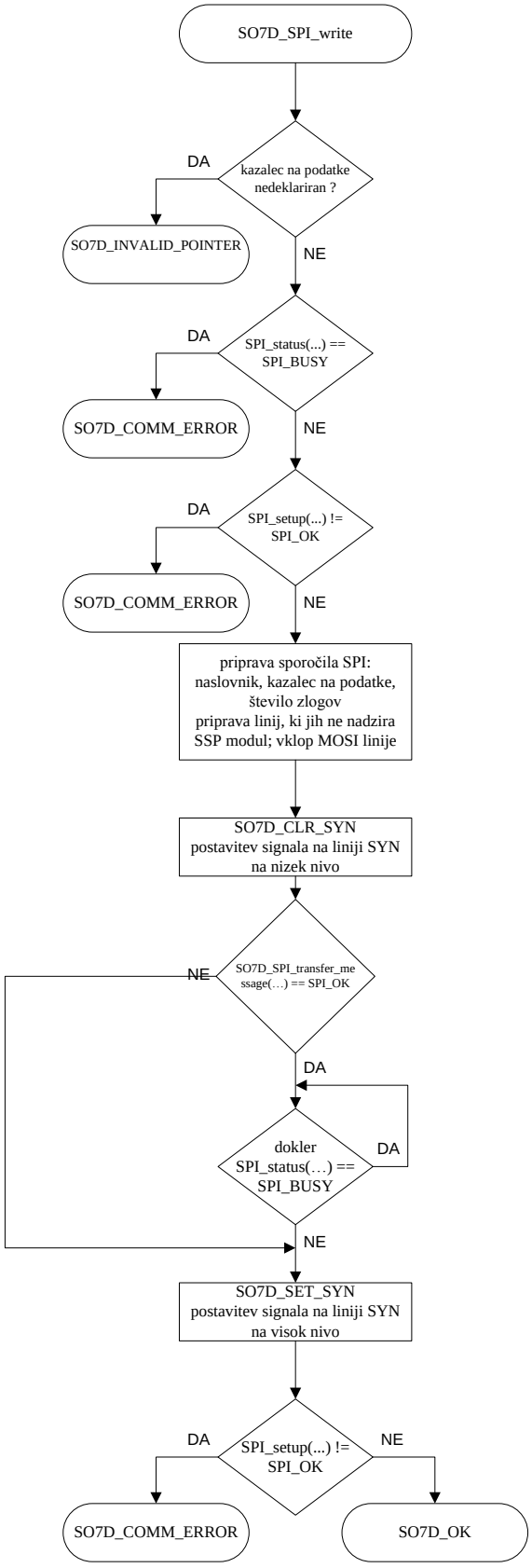
Slika 32: bločni diagram funkcije SO7D_SPI_transfer_message

Funkcijski blok branja je naslednji nivo komunikacije v modulu SO7D. Slika 33 prikazuje bločni diagram funckije `SO7D_read`. Le-ta najprej preveri, ali je kazalec na pomnilnik, kamor se bodo shranjevali podatki, nastavljen. Če ni, potem javimo `SO7D_INVALID_POINTER`, drugače zapišemo vse potrebne podatke v sporočilo in izklopimo MOSI linijo, saj je za prenos podatkov na Calculator dovolj ena linija. Nato preverimo, ali moramo zapahniti registre v merilnem vezju (osveževanje podatkov); če je odgovor pritrdilen, kličemo funkcijo LATCH, drugače ne. V prvem primeru so razmere na vodilu takšne, kot jih prikazuje slika 22, v drugem pa podobne, a brez aktivnega signala na liniji SYN. Nazadnje pošljemo sporočilo in preverimo status, ki ga funkcija vrne.



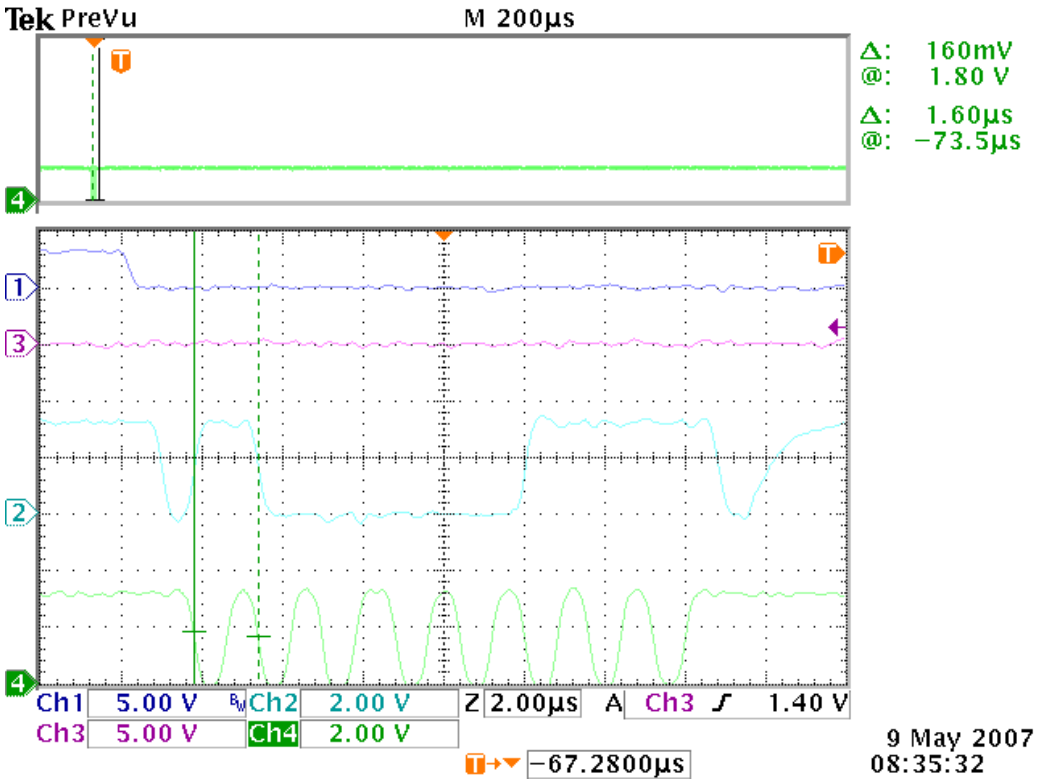
Slika 33: bločni diagram funkcije `SO7D_READ`

Funkcijski blok pisanja predstavljamo na sliki 34. Preverimo, ali je kazalec na podatke nastavljen in vodilo prosto ter zaradi varnosti zmanjšamo hitrost le-tega. Nato pripravimo sporočilo za prenos podatkov in postavimo signal SYN na nizek nivo. Ker je funkcija pisanja sinhrona funkcija, počakamo, da se vsi podatki prenesejo in šele nato nadaljujemo s proceduro. Nazadnje postavimo nivo na liniji SYN na logično 1 in zapustimo funkcijo. Razmere na vodilu pri vpisu prikazuje slika 23.



Slika 34: bločni diagram funkcije `SO7D_WRITE`

Vpis logične 1 na naslov 0x03 v delovni pomnilnik merilnega vezja prikazuje spodnja slika:



Slika 35: vpis logične 1 na naslov 0x03 v delovni pomnilnik merilnega vezja

Prvi signal predstavlja signal SS, drugi SYN, tretji nivo na liniji MISO ter četrti digitalno uro vodila SPI.

9.3.2 Funkcijski bloki modula MEASURE_IMP

Modul Measure_imp vsebuje 18 različnih funkcijskih blokov. V tabeli 17 so opisani samo tisti funkcijski bloki, ki so v začetku poglavja 9 določeni kot potencialno kritični.

funkcija	opis
`meas_init`	inicializacija osnovnih merilnih struktur
`meas_setup`	inicializacija merilnega vezja, klic funkcij modula SO7D za konfiguracijo
`meas_calibrate_imp`	kalibracija merilnega vezja, klic funkcij modula SO7D za kalibracijo
`meas_process`	glavni merilni proces, izvajanje v prekinitveni zanki na 16 ms
`meas_store_so7d_energy_data`	glavna funkcija za obdelavo podatkov o prirastku energije iz merilnega vezja po fazah
`meas_store_so7d_common_data`	glavna funkcija za obdelavo podatkov o frekvenci, napetosti in toku

Tabela 17: pomembnejše funkcije modula MEASURE_IMP

Podrobneje opisujemo funkciji `meas_store_so7d_energy_data` in `meas_store_so7d_common_data`, ostale pa predstavljamo v sledečih poglavjih. S prvo funkcijo preberemo vrednosti ustreznih energijskih števcev (delovno ali jalovo energijo) in jih ovrednotimo s funkcijo `SO7D_get_energy_delta_long`. Če je energijski prirastek večji od postavljene meje, ki jo smatramo za šum, le-tega shranimo v akumulacijske registre, drugače postavimo vrednost prirastka na nič. Te operacije se izvajajo na 32 ms. Vendar v funkciji v daljšem časovnem intervalu (256 ms) določamo tudi smer pretoka energije in beležimo ali smo presegli zaporni tok ali ne.

Funkcija `meas_store_so7d_common_data` shranjuje vrednosti efektivne napetosti in efektivnega toka. V novejših verzijah tudi informacijo o frekvenci omrežja.

9.3.3 Funkcijski bloki modula MEASURE

Funkcijski bloki modula MEASURE predstavljajo vez med merilnim delom in ostalimi moduli, ki pridobivajo merilne rezultate. Prav tako se v teh funkcijskih blokih opravlja zadnje skaliranje merilnih rezultatov, saj so vrednosti, ki jih dobimo iz funkcij modula MEASURE_IMP proporcionalni dejanskim vrednostim. Tabela 18 prikazuje nekatere funkcije API modula MEASURE:

funkcija	opis
`meas_start`	inicializira merilni del
`meas_activeenergy_getadvance`	prebere kumulirano delovno energijo
`meas_ reactiveenergy_getadvance`	prebere kumulirano jalovo energijo
`meas_apparentenergy_getadvance`	prebere kumulirano navidezno energijo
`meas_activeenergy_getflow`	prebere informacijo o smeri pretoka delovne energije
`meas_ reactiveenergy_getflow`	prebere informacijo o smeri pretoka jalove energije
`meas_apparentenergy_getflow`	prebere informacijo o smeri pretoka navidezne energije
`meas_activepower_getvalue`	prebere vrednost aktivne moči
`meas_voltage_getvalue`	prebere vrednost efektivnih napetosti
`meas_current_getvalue`	prebere vrednost efektivnega toka
`meas_frequency_getvalue`	prebere vrednost omrežne frekvence
`meas _phase_getsequence`	prebere informacijo o zaporedju faznih napetosti

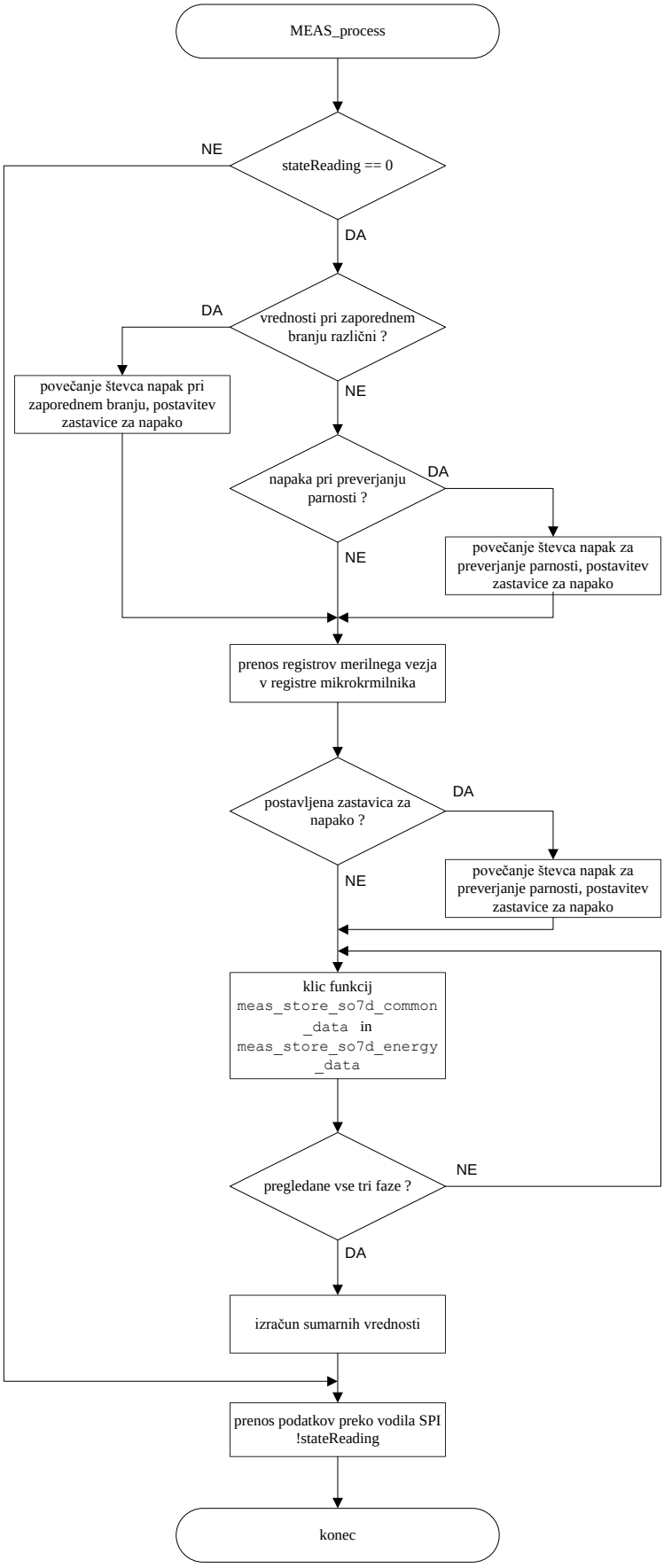
Tabela 18: pomembnejše funkcije modula MEASURE

9.3.4 Glavni merilni proces

Glavni merilni proces lahko razdelimo na dva večja podprocesa. V prvem delu se preveri, ali je potrebno prižgati katero izmed LED diod, ki prikazujejo bodisi kvant jalove ali navidezne energije bodisi kvant delovne energije. Krmiljenje diod LED poteka preko ustreznega avtomata. Stanja sistema so: PULSE, IDLE in CLOSURE. V stanju PULSE se dioda LED prižge, v stanju IDLE se ugasne, medtem ko se stanje CLOSURE uporablja ob prvem zagonu merilnega procesa ali kadar pride do prekinitve električnega toka. V drugem delu se obdelajo vse pomembne električne veličine in izoblikuje napoved, ali je potrebno prižgati katero izmed obeh diod LED; v tem primeru se na avtomatu nastavi stanje PULSE.

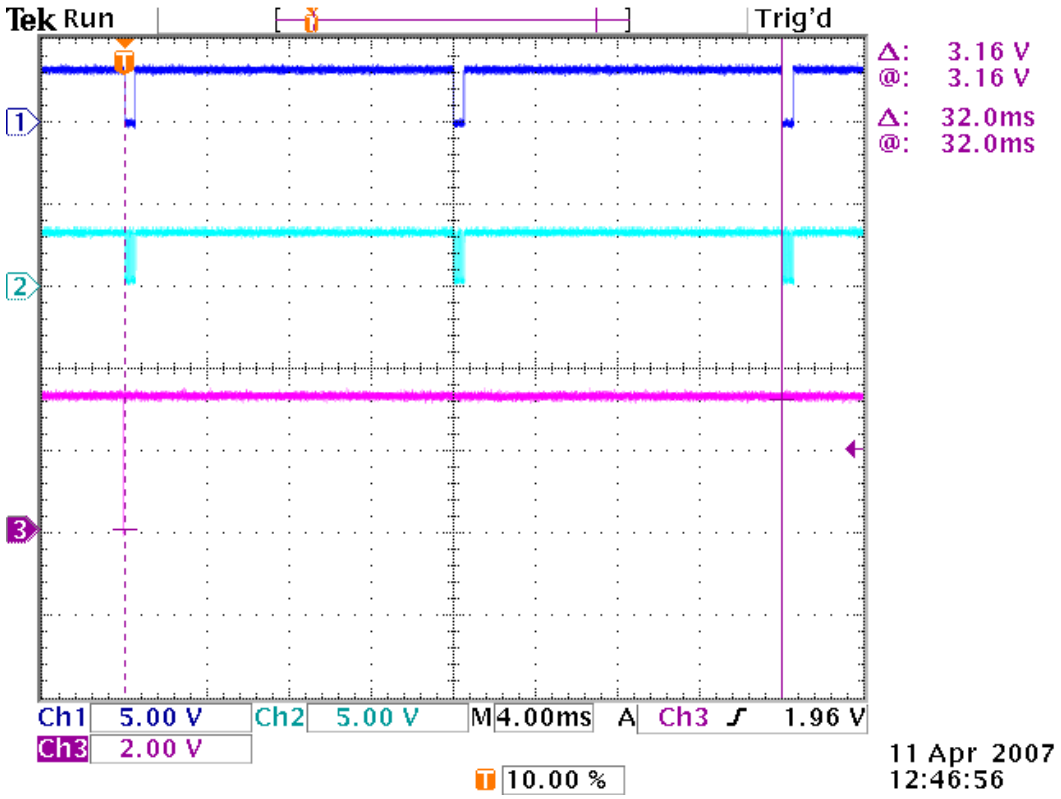
Električne veličine se obdelajo znotraj glavne merilne funkcije `meas_process`. Najprej se preberejo podatki iz merilnega vezja preko vodila SPI. Podatki se prebirajo vsakih 16 ms, osvežujejo pa vsakih 32 ms (spremenljivka stateReading). Isti podatki se vedno preberejo dvakrat in obe vrednosti medseboj primerjata. Tako zagotovimo, da na vodilu ni prišlo do kakršnihkoli napetostnih izbruhov (ang.: bursts). V primeru, da se podatki razlikujejo, javimo napako in zbrišemo vse energijske prirastke. V primeru, da so podatki enaki, sledi preverjanje parnosti. Ob uspešnem preverjanju le-te, sledi nadaljnja obdelava podatkov, ob neuspehu pa se ponovno javi napaka in izbrišejo vsi energijski prirastki.

Obdelava podatkov poteka po določenem zaporedju. Najprej se preberejo trenutne in efektivne vrednosti napetosti in toka, nato se preberejo energijski prirastki po posameznih fazah (odvisno od izbrane konfiguracije: energijski prirastki bremenskega in generatorskega režima delovanja) in določi status posamezne faze. Sledi obravnava sumarnih podatkov. Na koncu se preveri še stanje faz; ali je zaporedje pravilno, ali pa je prišlo pri priključitvi do napačne vezave. Slika 36 prikazuje bločni diagram funkcije `meas_process`.



Slika 36: bločni diagram funkcije `meas_process`

Bralni cikel na 16 ms prikazuje slika 37. Najprej preberemo sveže podatke, nato po 16 ms iste še enkrat in jih primerjamo. Tako zajemanje podatkov se nato ciklično ponavlja.



Slika 37: Bralni cikel merilnega dela

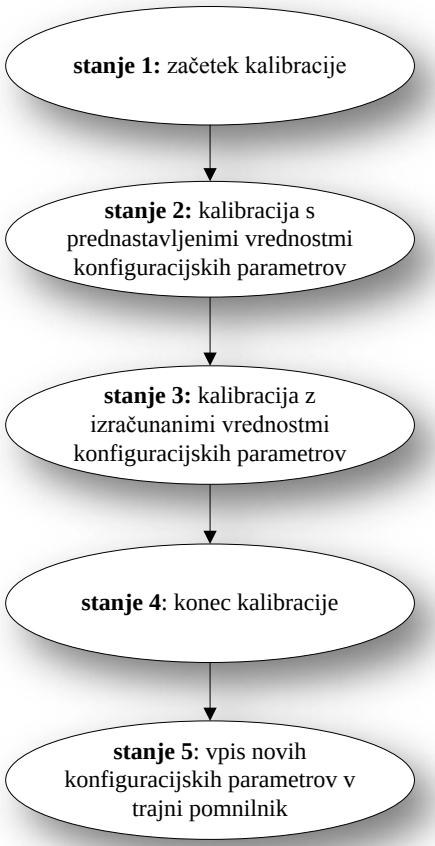
9.3.5 Inicializacija, kalibracija in konfiguracija merilnega vezja

Inicializacijo, konfiguracijo in kalibracijo se šteje pod pomožne funkcionalnosti, saj vplivajo na delovanje merilnega sistema, ker so odgovorne za konfiguracijo merilnih čipov in pripadajočih programskih struktur.

Inicializacija se izvrši ob prvem in vsakem ponovnem zagonu števca, da se nastavijo vse pomembnejše strukture in konstante. Konfiguracija merilnih integriranih vezij ob inicializaciji poteka zgolj prvič, ko zaščitni bit bloka OTP (bit TSTD) še ni postavljen na logično 1.

Pri konfiguraciji se najprej preberejo prenastavljene konfiguracijske vrednosti iz bliskovnega pomnilnika in vpišejo v delovni pomnilnik. Nato sledi varovan vpis izbrane konfiguracije v delovni pomnilnik merilnega vezja. Obenem se nastavijo tudi ostali pomembni registri in vrednosti. Konstante diod LED se nastavijo na prenastavljene vrednosti, stanja avtomatov se inicializirajo na začetne vrednosti, vsi pomembni registri dobijo začetne vrednosti.

Kalibracija se izvrši ob vpisu določene vrednosti v ustrezen register. Kalibracijska funkcija se kliče dvakrat; prvič se kalibracijske vrednosti izračunajo in preverijo, drugič pa trajno zapečejo v ustrezne celice OTP merilnih čipov. Ob prvem klicu se naložijo konfiguracijske vrednosti iz bliskovnega pomnilnika v delovni pomnilnik in zapečejo v ustrezni delovni pomnilnik merilnega vezja. Nato se izvede kalibracija pri 5 A in 230 V. Dobljene vrednosti se vpišejo v ustrezni del delovnega pomnilnika merilnega vezja in nato še v pomnilnik FRAM. Sledi validacija vrednosti vseh izračunanih konstant. Ob drugem klicu se vrednosti iz ustreznega dela delovnega pomnilnika prepišejo v celice OTP. Na koncu se prvi bit celic OTP postavi na logično 1, kar onemogoči kasnejše pisanje in branje po konfiguaracijskih vrednostih merilnih čipov. Slika 38 prikazuje potek kalibracije:



Slika 38: potek kalibracije pri MT372-SMART

10 Analiza programske opreme z metodo SWFMEA

Merilni sistem, opisan v poglavju 9, ovrednotimo s pomočjo metode SWFMEA. Omejimo se zgolj na tista področja, ki smo jih z metodo PHA prepoznali kot nevarna. Za izračun preostalega dela števila RPN bomo analizirali napisane programske module s programom CCCC, ki izračuna McCabovo ciklomatično kompleksnost. Tabela 19 prikazuje ciklomatično kompleksnost *v(G)* in normirano ciklomatično kompleksnost $v_n(G)$ za funkcije, ki jih uporabljamo pri postopkih, opisanih v poglavju 9:

funkcija	tip hazarda[4]	ν(G)	$\nu_n(G)$
`SO7D_SPI_transfer_message`	a,b,c,d	5	1
`SO7D_calibrate`	c	30	6
`SO7D_configure`	d	20	4
`SO7D_program`	c,d	13	3
`SO7D_read`	a,b,c,d	6	2
`SO7D_write`	a,b,c,d	11	3
`SO7D_init`	a	2	1
`meas_calibrate_imp`	c	13	3
`meas_process`	b	17	4
`meas_setup`	a	5	1
`meas_store_so7d_common_data`	e,f	1	1
`meas_store_so7d_energy_data`	e,f	19	4

Tabela 19: ciklomatične vrednosti za posamezne funkcijske bloke

Dobljene ciklomatične vrednosti za posamezni funkcijski blok kažejo, da nobena vrednost ne posega v nedovoljeno kategorijo po tabeli 2 (prezapleten funkcijski blok). Skupno ciklomatično kompleksnost za postopek, ki ga definira več funkcij (inicializacija, kalibracija in konfiguracija) dobimo tako, da vzamemo povprečno vrednost vseh vpletenih funkcij. Tako lahko izpolnimo tabelo Tabela 20 in dobimo analizo sistema:

[4] povzeto po tabeli 14

element/ vmesnik	oblika odpovedi	lokalna posledica	posledica na višjem nivoju	posledica na sistem	RPN	opombe
inicializacija (a)	napaka pri zagonu časovnikov, napačno izbrane vrednosti prenastavljenih vrednosti	napačne začetne vrednosti	nepravilni rezultati	možna odpoved sistema, zaradi nepravilnih rezultatov, ki jih prejmejo izvršni členi	11	implementirano preverjanje pravilne izvršitve funkcije preko statusov
napaka pri komunikaciji z merilnim vezjem (b)	sprejem pokvarjenih paketov z vodila	napačno tolmačenje vrednosti	nepravilni rezultati	možna odpoved sistema, zaradi nepravilnih rezultatov, ki jih prejmejo izvršni členi	52	implementirano dvakratno branje istih podatkov in preverjanje parnosti
nepravilna kalibracija (c)	nepravilne vhodne vrednosti, odpoved analognih komponent, za zagotavljanje primerne napetosti za trajni zapis v blok OTP	ni	nepravilni rezultati	napačno definirani rezultati, sistem ne odpove, ker prikazuje manjše vrednosti od realnih	24	kalibracija vrne status uspešnosti, v primeru neuspeha moramo kalibracijo ponoviti
nepravilna konfiguracija (d)	napačno izbrana konfiguracija, motnje pri prenosu podatkov	ni	nepravilni rezultati	sistem je nestabilen, merilni podatki ne odražajo dejanskega stanja, raztros meritev je velik	8	po konfiguraciji pregledamo statuse in ugotovimo možne napake; v primeru napake, se konfiguracija ponovi
vmesnik – osveževanje (e)	pri prenosu informacij med moduli lahko pride do napake	ni	manko rezultatov	stabilnost delovanja števca je okrnjena, zaradi manka rezultatov	20	vsaka funkcija API vrne status, ali je prišlo do napake
vmesnik – časovna neustreznost (f)	osveževanje rezultatov ob napačnem trenutku, uporaba funkcij, v katere lahko vstopimo v danem intervalu le enkrat (ang.: non-reentrant)	spremenjene vrednosti v ustreznih registrih	nepravilni rezultati	stabilnost delovanja števca je okrnjena, zaradi nepravilnih rezultatov	20	funkcije napisati tako, da se lahko vanje vstopi večkrat v nekem časovnem intervalu

Tabela 20: rezultati analize pridobljene po metodi SWFMEA

Ob upoštevanju smernic nastavljenega standarda varnega programiranja v poglavju 7.2 smo izpolnili spodnja nadzorna seznama.

načrtovalski postopek	uresničitev	komentar
testiranje CPE	NE	implementirano v drugem modulu
časovni čuvaj	NE	implementirano v drugem modulu
zaščita pred nenamernim spreminjanjem spremenljivk	NE	implementirano višje v programu
preverjanje sklada	NE	
prevajanje programa z vključenimi opozorili	DA	
uporaba varnih podatkovnih struktur	DA	
deklaracije in inicializacije spremenljivk	DA	
preverjanje vrednosti, ki jih vračajo funkcije	DA	
uporaba standardnih elementov jezika	DA	striktno v skladu z ANSI C
previdno pretvarjanje tipov	DA	vsako pretvarjanje tipov je opravljeno eksplicitno
možnost detekcije izpada napajanja in postopna zaustavitev sistema	NE	implementirano v drugem modulu
seznam vseh možnih odpovedi strojne opreme, ki lahko vpliva na programsko	DA	merilno vezje in komunikacija preko vodila SPI
zmanjšanje kompleksnosti kode	DA	poskušati zmanjšati ciklomatično kompleksnost
uporaba metode vpiši-preberi-testiraj (s prvo funkcijo vpišemo spremenljivko, z drugo preberemo ter preverimo njeno integriteto	DA	pri kalibraciji in konfiguraciji najprej vpišemo, nato preverimo vpisano
načrtanje grafa odvisnosti med moduli	DA	to možnost lahko vklopimo pri prevajanju programske opreme

Tabela 21: izpolnjeni nadzorni seznam za razvoj programske opreme

mehanizem obrambnega programiranja	obrazložitev	uresničitev
izogibanje uporabe stavka `goto`	prinaša negotovost v programski tok in časovno stohastičnost	DA
uporaba oklepajev za določitev prioritete izvajanja operatorjev	večja preglednost; izvajanja stavkov preko eksplicitne določitve prioritete in ne preko prioritetne liste operatorjev C-ja	DA
omejitev števila in velikosti parametrov podanih funkcijam	preveliko število parametrov vpliva na zmožnost testiranja take funkcije, velike strukture, če niso podane preko kazalcev, lahko preplavijo sklad	DA
minimalna uporaba rekurzivnih funkcij	enostavna preplavitev sklada, ob uporabi takih funkcij zagotoviti končno rekurzijo	DA
uporaba direktive `default` pri izbirnem stavku `switch`.	eksplicitna določitev izvajanja programa v primeru neujemanja izbirne vrednosti z delom `case` v izbirnem stavku	DA
prepovedana uporaba funkcij z nedoločenim številom parametrov	prevajalnik ne more preveriti takih funkcij, zato jih je težko verificirati	DA
prepovedana uporaba operatorjev ++ in - - s parametri, podanimi prek funkcij	izvrševanje stavka `funkcija(spremenljivka++)` je nepredvidljivo in vodi v napake	DA
uporaba bitnih mask preko makrojev, namesto bitnih polj	bitna polja niso specifično opredeljena v C90 in so odvisna od interpretacije prevajalnika	DA
uporaba predprocesorskega ukaza `#define` namesto direktne uporabe števil	uporaba makroja, definiranega kot `#define` POLMER_ZEMLJE_V_KM 6356 je preglednejši kot samo 6356. Koda postane preglednejša in lažje vzdrževana. Spreminjanje konstane se odvija na enem mestu, pri makroju, in ne pri vsaki spremenljivki	DA

Tabela 22: izpolnjeni seznam svojstvenih lastnosti C-ja

11 Analiza merilnih rezultatov

Merilne rezultate smo primerjali z etalonskim merilnikom pogreška TEMP-100. Družina etalonskih merilnikov pogreška TEMP je namenjena merjenju in registraciji električne energije v enofaznih ali trifaznih (tri- ali štiri- vodnih) sistemih. Merilniki omogočajo dvosistemsko ali trisistemsko merjenje in registracijo delovne in jalove energije (ne hkrati) in merjenje pogreška števca ter izvedbo registracije. Slednje pomeni, da merimo števčne pulze za energijo, ki se morajo ujemati s številom pulzov etalonskega merilnika [36].

Postopke za ugotavljanje točnosti vgrajenih algoritmov razvijemo v skladu s standardom IEC 62053-21 [35]. Le-ta opredeljuje največji dovoljeni pogrešek za elektronske števce električne energije razreda 2. Vrednosti v tabeli Tabela 23 prikazujejo dovoljeni pogrešek pri nazivni napetosti (U = 230 V) in simetričnem bremenu.

tokovne vrednosti	faktor moči	Dovoljena napaka v odstotkih za števce razreda 2
$0{,}05 \cdot I_b \leq I < 0{,}1 \cdot I_b$	1	± 2,5
$0{,}1 \cdot I_b \leq I < I_{max}$	1	±2,0

Tabela 23: dovoljena napaka za enofazne in večfazne števce s simetričnim bremenom

Pri tri faznih sistemih ločimo še dodatno možnost, ko breme ni simetrično. Še dovoljene pogreške v tem primeru prikazujemo v tabeliTabela 24.

tokovne vrednosti	faktor moči	Dovoljena napaka v odstotkih za števce razreda 2
$0{,}1 \cdot I_b \leq I \leq I_{max}$	1	±3,0

Tabela 24: dovoljena napaka za več-fazne števce z nesimetričnim bremenom

V zgornjih tabelah predstavlja I_b nazivni tok (5 A), I_{max} pa najvišji dovoljeni tok na vhodu števca (120 A).

V nadaljevanju preverjamo zagonski tok števca. Test izvedemo tako, da vsilimo vsaki fazi napetost U = 230 V in dvigujemo tok simetrično v vseh fazah. Števec mora začeti meriti vsaj pri zagonskem toku, prikazanem v tabeli Tabela 25.

največji dovoljeni zagonski tok za števce razreda 2	faktor moči
$0{,}005 \cdot I_b$	1

Tabela 25: največji dovoljeni zagonski tok

Nazadnje opravimo še preizkus praznega teka (ang.: no-load condition). Referenčna napetost je 115 % nazivne vrednosti, medtem ko čas trajanja Δt določimo na sledeč način.

$$\Delta t \geq \frac{600 \cdot 10^6}{k \cdot m \cdot U_n \cdot I_{max}} \text{ [min]} \tag{11}$$

Oznaka *k* pomeni konstanto števca (1 000 pulzov/kWh), *m* število merilnih elementov (3), U_n referenčno napetost (230 V) in I_{max} največji dovoljeni tok (120 A).
Števec v tem času ne sme narediti več kot enega pulza (pomeriti 1 Wh). Za naš test smo vzeli Δt = 10 min. Števec v tem času ni naredil nobenega pulza, zato je test prestal.

11.1 Merilni pogrešek

Merilni pogrešek merimo z etalonskim merilnikom TEMP tako, da beležimo število pulzov diode LED za delovno energijo števca MT372-SMART. Na podlagi trajanja petih pulzov se izračuna merilni pogrešek. Vrednosti za različne tokove pri simetričnem bremenu prikazuje spodnja tabela.

I_s [A]	I_1 [A]	I_2 [A]	I_3 [A]	*e* [%]	faktor moči
0,750	0,250	0,250	0,250	0,50	1
1,500	0,500	0,500	0,500	0,46	1
3,000	1,000	1,000	1,000	0,36	1
15,000	5,000	5,000	5,000	0,25	1
60,000	20,000	20,000	20,000	0,27	1

Tabela 26: merilni pogrešek pri različnih simetričnih bremenih

I_s predstavlja vsoto tokov posamezne faze. Ker je breme simetrično velja: $I_s = I_1 + I_2 + I_3$. Rezultati kažejo, da je merilni pogrešek bistveno manjši od največjega dovoljenega pogreška razreda 2. Najmanjši pogrešek je pri nazivnem toku 5 A, saj se pri tej vrednosti opravlja kalibracija.

Vrednosti merilnega pogreška za nesimetrično breme prikazujejo naslednje tabele.

I_s [A]	e [%]	faktor moči
0,250	0,55	1
0,500	0,42	1
1,000	0,27	1
5,000	0,14	1
20,000	0,26	1

Tabela 27: merilni pogrešek pri nesimetričnem bremenu $I_s = I_1$; $I_2 = I_3 = 0$

I_s [A]	e [%]	faktor moči
0,250	0,53	1
0,500	0,41	1
1,000	0,25	1
5,000	0,24	1
20,000	0,33	

Tabela 28: merilni pogrešek pri nesimetričnem bremenu $I_s = I_2$; $I_1 = I_3 = 0$

I_s [A]	e [%]	faktor moči
0,250	0,72	1
0,500	0,53	1
1,000	0,25	1
5,000	0,17	1
20,000	0,20	1

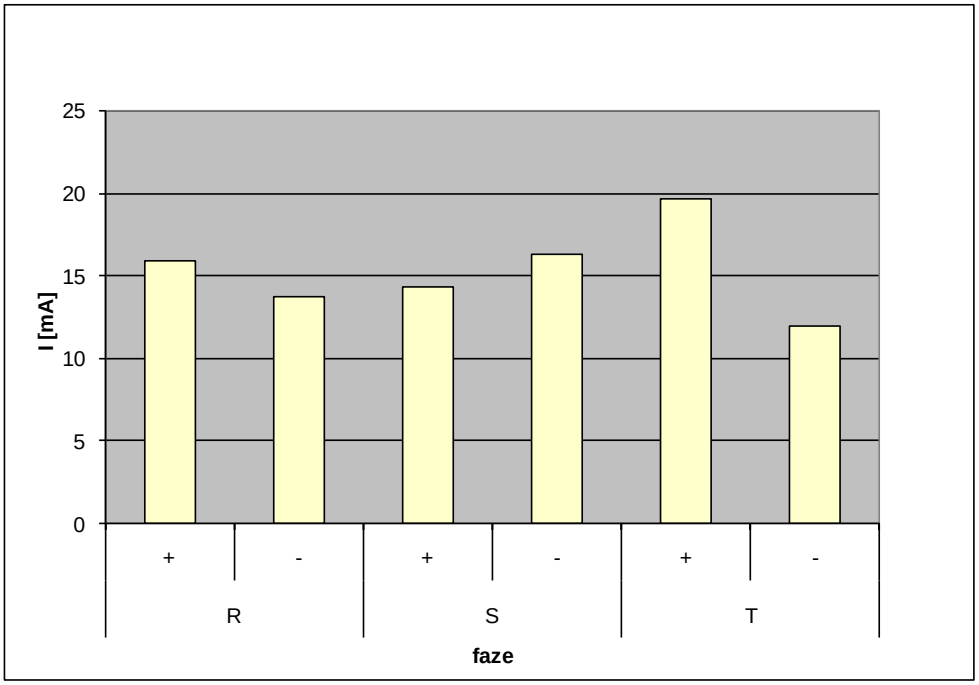
Tabela 29: merilni pogrešek pri nesimetričnem bremenu $I_s = I_3$; $I_1 = I_2 = 0$

Merilni pogrešek je znotraj predpisanih toleranc. Najmanjši je pri kalibracijski vrednosti 5 A. Nekoliko slabši rezultati so pri majhnih tokovih, a še vedno znotraj predpisanega območja, saj tudi standard [35] predvideva večji odklon pri majhnih tokovih.

11.2 Zagonski tok

Meritev opravljamo pri nazivni napetosti U = 230 V na vsaki fazi. Simetrično breme enakomerno spreminjamo in beležimo tok, pri katerem števec začne meriti. Standard [35] zahteva za števce razreda 2 začetek merjenja pri 0,5 % nazivnega toka, kar predstavlja 25 mA pri števcu MT372-SMART. Pri simetričnem bremenu pridejo vse tri faze iz zapore vsaj pri 18 mA.

Interni standard Iskraemeca narekuje dodatni test merjenja zapornega toka pri nesimetričnem bremenu in prisotnosti samo ene faze. Na sliki 39 prikazujemo zagonski tok po posameznih fazah za generatorski (P–) in bremenski način (P+) delovanja pri nesimetričnem bremenu.



Slika 39: Zaporni tok bremenskega in generatorskega režima delovanja za posamezno fazo

Pri nesimetričnem bremenu se zaporni tokovi po posameznih fazah razlikujejo. Očitno je odstopanje pri fazi T, kjer se pozna vpliv tuljavice napajalnega vezja. Kljub temu je tok znotraj predpisanih 25 mA.

12 Zaključek

Cilj diplomske naloge je bil izdelati varno in zanesljivo programsko opremo merilnega dela števca MT372-SMART. Pri tem smo si pomagali z analizo programske opreme po metodi SWFMEA. Analizirali smo tiste ključne točke v merilnem procesu, ki so se izkazale potencialno nevarne. K reševanju problema smo pristopili sistematično, zato smo umestili postopek razvoja programske opreme v življenjski cikel celotnega sistema.

Zaradi obsežnosti takega pristopa, smo se osredotočili zgolj na razvoj in analizo programske opreme. Za celovito analizo pa je smiselno ovrednotiti vsak korak v življenjskem ciklu razvoja sistema. To opravilo je težavno, ker težko pridobimo kvantitativno vrednost za izračun števila RPN pred napisano programsko opremo. Obstajajo postopki, ki poskušajo zaobiti to pomanjkljivost, tako, da definirajo nestabilnost posameznega funkcijskega bloka in celotnega modula. Le-ta je definirana kot stopnja odvisnosti funkcijskega bloka od ostalih in obratno. Na podlagi teh rezultatov se lahko predvidi frekvenca pojavljanja odpovedi.

Kljub temu so naši rezultati zadovoljivo pokazali, kje so možne napake oziroma, kje so bile sistemske slabosti. Le-te smo odpravili, saj merilni sistem v praksi še ni pokazal bistvenih odstopanj od pričakovanih rezultatov.

13 Literatura

[1] Franc Bergelj, Meritve 1, Fakulteta za elektrotehniko, Ljubljana 2002.

[2] (avtor ni podan), A brief history of meter companies and meter evolution, http://watthourmeters.com/history.html, dostopnost preverjena maja 2008.

[3] (avtor ni podan), Najpomembnejši mejniki v razvoju Iskraemeca, http://www.iskraemeco.si/emecoweb/slo/company/milestones.htm, dostopnost preverjena maja 2008.

[4] Stanislav Miklavčič, Iskraemeco AMM system presentation, Kranj 2007.

[5] Jack Gansle, The firmware handbook, Elsevier, Oxford 2004.

[6] B. J. Czerny, J. G. D'Ambrosio, B. T. Murray, P. Sundaram, Effective Application of Software Safety Techniques for Automotive Embedded Control Systems, SAE World Congress, 2005.

[7] Todd D. Morton, Embedded microcontrollers, Prentice Hall, New Jersey 2000.

[8] (avtorji niso podani), NASA Software Safety Guidebook, National Aeronautics and Space Administration, Washington 1995.

[9] (avtorji niso podani), Guidelines for Failure Mode and Effects Analysis for Automotive, Aerospace and General Manufacturing Industries, Dyadem Press, Ontario 2002.

[10] H. Pentti, H. Atte, Failure mode and effects analysis of software-based automation systems, STUK, Helsinki 2002.

[11] R. E. McDermott, R. J. Mikulak, M. R. Beauregard, The Basics of FMEA, Resource Engineering, Portland 1996.

[12] Oliver Mäckel, Software FMEA Opportunities and benefits of FMEA in the development process of software-intensive technical systems, Siemens AG, Mù nchen.

[13] (avtorji niso podani), Pareto Charts; Distribution and Casual Analysis Tools, http://www.hanford.gov/rl/uploadfiles/VPP_Pareto.pdf, dostopnost preverjena maja 2008.

[14] T. J. McCabe, A Complexity Measure, IEEE Transactions on software engineering, vol. SE-2 NO. 4, str. 308-320, December 1976.

[15] (avtor ni znan), User Guide for CCCC, http://stderr.org/doc/cccc/CCCC%20User%20Guide.html, dostopnost preverjena maja 2008.

[16] J. B. Bowles, C. Wan, Software Failure Modes and Effects Analysis For a Small Embedded Control System, Proceedings IEEE, Annual Reliability and Maintainability Symposium, str. 1-6, 2001.

[17] G. Menkhaus, B. Andrich, Metric suite for directing the failure mode analysis of embedded software systems, In Proc. Of the 7th International Conference on Enterprise Information Systems ICEIS, 2005.

[18] Franc Bratkovič, Uvod v C, Fakulteta za elektrotehniko, Ljubljana 1998.

[19] Pete Goodliffe, Code Craft – The practice of writing excellent code, No Starch Press, San Francisco 2007.

[20] Dennis M. Ritchie, The Development of the C Language, http://cm.bell-labs.com/who/dmr/chist.html, dostopnost preverjena maja 2008.

[21] V. Strle, Specification of Application Specific Integrated Circuit, Calculator, R&D Microelectronics, Kranj 2005.

[22] Diplomska naloga, Mikrokrmilniški učni sistem, Aleš Hribar, Ljubljana, avgust 2006.

[23] (avtor ni poznan), ARM7TDMI-S Technical Reference Manual revison r4p3, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0234b/DDI0234.pdf, dostopnost preverjena maja 2008.

[24] (avtor ni poznan), LPC213x User Manual Rev. 01, Phillips, 2005

[25] Marko Munih, Mikrokrmilnik, Fakulteta za elektrotehniko, Ljubljana 1998.

[26] (avtor ni poznan), FM31256 Integrated Processor Companion with Memory User manual, http://www.ramtron.com/files/datasheets/FM31xx_r2.4.pdf, dostopnost preverjena maja 2008.

[27] V. Strle, Specification of Application Specific Integrated Circuit, SmartSensor, R&D Microelectronics, Kranj 2004.

[28] (avtor ni podan), Low Power OFF-Line SMPS Primary Switcher, http://www.st.com/stonline/products/literature/ds/11977.pdf, dostopnost preverjena maja 2008.

[29] (avtor ni podan), MC34063A/MC33063A SMPS Controller http://www.datasheetcatalog.org/datasheet2/2/048rqig2g15f45345ar3uilpijyy.pdf, dostopnost preverjena maja 2008.

[30] (avtor ni podan), Mx37y Functional description, Iskraemeco d.d, Kranj 2008.

[31] (avtor ni podan), Wireless CPU Q24NG; Product technical specification, http://www.kern.hu/KERN_HU/DOCBase/Datasheets/PTS_Q24_Series_rev004.pdf, dostopnost preverjena maja 2008.

[32] (avtor ni podan), WELMEC 7.2 Issue 1, SW Guide (Measuring Instruments Directive 2004/22/EC), http://www.mirs.gov.si/fileadmin/um.gov.si/pageuploads/Dokpdf/SITM/W72Iss120050520.pdf, dostopnost preverjena maja 2008.

[33] N. Međeral, U. Jerman, Mx37y Software description, Iskraemeco, d.d., Kranj 2007.

[34] L. Kunčič, M. Kosmač, Dodatek k specifikacijam ASIC vezja Calculator, Iskraemeco, d.d., Kranj 2007.

[35] IEC 62053-21, IEC standard, 2003.

[36] (avtor ni podan), TEMP-100 etalonski merilnik pogreška, tehnični opis, Iskraemeco, d.d., Kranj 2005.

Izjava:

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Boštjana Murovca. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Ljubljana, 25. 5. 2008 Uroš Jerman