UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Tkalčič

Strojna implementacija algoritma za prenapetostno zaščito

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00402/2008
Datum: 01.09.2008

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALEŠ TKALČIČ**

Naslov: **STROJNA IMPLEMENTACIJA ALGORITMA ZA PRENAPETOSTNO ZAŠČITO**

A HARDWARE IMPLEMENTATION OF THE OVERVOLTAGE PROTECTION ALGORITHM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Na osnovi podane specifikacije implementirajte algoritem za prenapetostno zaščito v jeziku VHDL. Preverite delovanje algoritma v logičnem simulatorju na osnovi simuliranih vhodnih podatkov. Raziščite možnosti implementacije algoritma v vezju FPGA Spartan III.

Mentor:

doc. dr. Branko Šter

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:

prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Aleš Tkalčič** ,

z vpisno številko **63030069** ,

sem avtor/-ica diplomskega dela z naslovom:

Strojna implementacija algoritma za prenapetostno zaščito

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc.dr. Branko Šter

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se mentorju doc. dr. Branku Šteru, za strokovno pomoč in koristne nasvete. Hvala vam, posebej za vašo prijaznost, pripravljenost in izkazano pomoč pri izvedbi diplomske naloge.

Hvala bratu, očetu in mami za podporo skozi celotni študij. Hvala, ker ste verjeli vame. Ati in mami, to diplomo posvečam vama.

Hvala dekletu, za vzpodbujanje pri pisanju naloge. Hvala ti za lep zaključek študija.

Hvala vsem prijateljem, s katerimi sem šel skozi študentska leta. Doživeli smo marsikaj, ostali so nam pa nepozabni trenutki.

Kazalo:

Kazalo slik:

Povzetek

Namen diplomskega dela je izvedba prenapetostnega zaščitnega algoritma v jeziku VHDL. Algoritem se uporablja za zaščito naprav v elektroenergetskem sistemu (EES). Namen zaščitnih algoritmov, ki delujejo v realnem času, je, da delujejo po sistemu 24/7/365 (vse dni v letu). Pri tem pa si ne smemo dovoliti kakršnega koli izpada delovanja zaščite, saj bi poleg ogromne materialne škode ogrožali človeška življenja.

Da pa bi bralcu lažje razložil, kaj zaščita sploh je, sem v prvem delu diplomske naloge predstavil EES in namen zaščit v njem. Kronološko sem razdelil zaščitne naprave v skupine glede na njihove skupne značilnosti in jih opisal. Tako sem predstavil, kam bi uvrstil prenapetostni algoritem, njegovo delovanje, kje je uporabljen in pa na katere zahteve moramo biti pozorni, ko razvijamo takšne algoritme.

V diplomski nalogi sem izvedel takšen algoritem, ki mora zadoščati tem zahtevam. Pred samim začetkom kodiranjem sem naredil dokumentiran načrt (dizajn dokument) tega algoritma. Pri tem sem skušal predstaviti delovanje algoritma in določiti robne pogoje. Potrebno je bilo poiskati vse pasti, v katerih bi se lahko ob določenih trenutkih algoritem oziroma zaščita nahajala. Z dokumentiranem načrtu se programer zaščiti pred samim delovanjem in zagovori način izvedbe. Kot je že napisano, je algoritem kodiran v jeziku VHDL, za njegovo pisanje pa sem uporabljal orodje Active-HDL. Na koncu sem še s taistim orodjem simuliral in testiral delovanje, da sem prišel do zaključnih rezultatov.

Ključne besede: elektroenergetski sistem, zaščita, prenapetostni algoritem, VHDL

Abstract

The objective of my Bachelor's thesis is to implement an over-voltage protective algorithm in VHDL hardware description language. The algorithm is used as a protection in power system protection. A protective algorithm is intended to run as a real-time application on the basis of 24/7/365 (365 days a year) system. It is imperative that no falling out or stopping happens with a protective device based on such system. This is not only because enormous damage/cost would be caused, but also because it would put human lives in danger. My final work provides such a reliable algorithm.

With readers of the thesis in mind, I prepared a simple introduction to EES and the basics of protection for better understanding of my work. I classify protective devices according to chronology into groups and describe their characteristics. This helped me to classify my over-voltage protection algorithm and to define its working, use and demands in programming.

Before coding the program, I prepared a design document of such an algorithm. I tried to present its functioning and define all the possible conditions to be kept in mind. It is important to foresee and predict any problems and complications with the device at any time during its performing. With the design document, a programmer also defends the way of working on the assigned project. I used the Active-HDL tool to code the program in VHDL language. I also used the same tool to test and simulate the program. Algorithm performance and the result are provided.

Keywords: power system protection, protection, over-voltage algorithm, VHDL

1 UVOD

Dandanes se brez električne energije ustavi čas. Seveda v prenesenem pomenu. Dejansko prenehajo delovati številne naprave. Na primer, ugasnejo luči na semaforju, ustavijo se aparature v bolnišnicah, prekine se šolski pouk in delo. Prenehajo voziti sodobni vlaki, ugasnejo televizijski sprejemniki in prenosni telefoni ne najdejo omrežja … Je že res, da si za nekaj trenutkov lahko pomagamo z raznimi generatorji, ampak globalno trenutno ni druge alternative po nadomestitve električne energije, kot proizvajanje nje same z različnimi tipi elektrarn. Le-te so priključene na električno omrežje (EES), ki je mednarodno povezano. Zato je pomembno, da se znotraj omrežja držimo standardov glede napetosti vodov, ki nam prinašajo električno energijo in pri tem vzdržujemo stabilnost EES. Vendar zmeraj ni tako. Razne motnje, ki se pojavijo v EES, povzročajo uporabnikom težave v obliki izpada električne energije. Zato je nujno, da imamo v takšnem sistemu, kot je EES, nadzor, in da motnje, ki so neizbežne, ublažimo, pri tem pa uporabnik in hkrati odjemalec najmanj trpi.

Nadzor, ki ga izvajamo, je med drugim tudi zaščita EES. Nad EES izvajamo vrsto zaščitnih algoritmov, ki so med drugim tudi zaščita za uporabnika. Skozi zgodovino se je razvoj zaščit spreminjal in izpopolnjeval.

Varovalko, ki je bila predstavljena že več kot sto let nazaj, poznamo kot prvo obliko zaščite, uporabljene na električnem omrežju. Kljub temu pa ni sposobna zaščite obširnih motenj pretoka velikih napetosti v električnem krogu sistema napajanja. Uvedba trifaznega sistema je pospešila razvijanje elektroenergetskega sistema, ki se je dopolnil z iznajdbo transformatorja, ki je omogočal prenos električne energije na daljavo oziroma po daljnovodih. Prvi prenos električne energije je bil leta 1891 med mestoma Lauffen in Frankfurt v Nemčiji s kapaciteto 200kW/15kV. S tem pa se je pojavila potreba po zaščiti elektroenergetskega sistema, ki služi za nadzor in kontrolo faz in električnega kroga s primarnimi releji (ang. circuit breaker).

Okvare, ki so posledica strel, poškodb izolatorjev, atmosferskih vplivov, onesnaženja, mehanskih poškodb, staranja izolacije, prekomernega segrevanja, preobremenitev, itd. so stalnica in neizogibno spremljajo delovanje vsakega EES in jih ni mogoče preprečiti. Lahko le znižamo verjetnosti nastopa in omejujemo posledice. Posledico okvar, ki je kot izguba stabilnosti v delu povezanega EES, pa uporabniki čutimo kot izpad v dobavi električne energije. Medtem pa si ne želimo, da pride do večjih okvar, ki bi pomenile ogrožanje življenj prebivalstva in ostalih katastrof, kot se je na primer zgodilo v Novi Angliji leta 1965, ko je zaradi človeške napake pri nastavitvi zaščitnega releja, le-ta odpovedal in se je naenkrat v kaosu znašlo 30 milijonov ljudi. Dogodek ponazarja neustrezno delovanje zaščite in nakazuje pomen zaščite za varno obratovanje EES-ov, ki pravilno zasnovana in izvedena omejuje posledice okvar na najmanjšo možno mero.

Sodobne zaščitne naprave dandanes predstavljajo proizvod visoke tehnologije, izvedene z izjemno zmogljivimi paralelnimi signalnimi procesorji s hitrimi perifernimi enotami, optični komunikacijskimi vmesniki in bogato programsko opremo, ki že počasi posega na področje umetne inteligence.

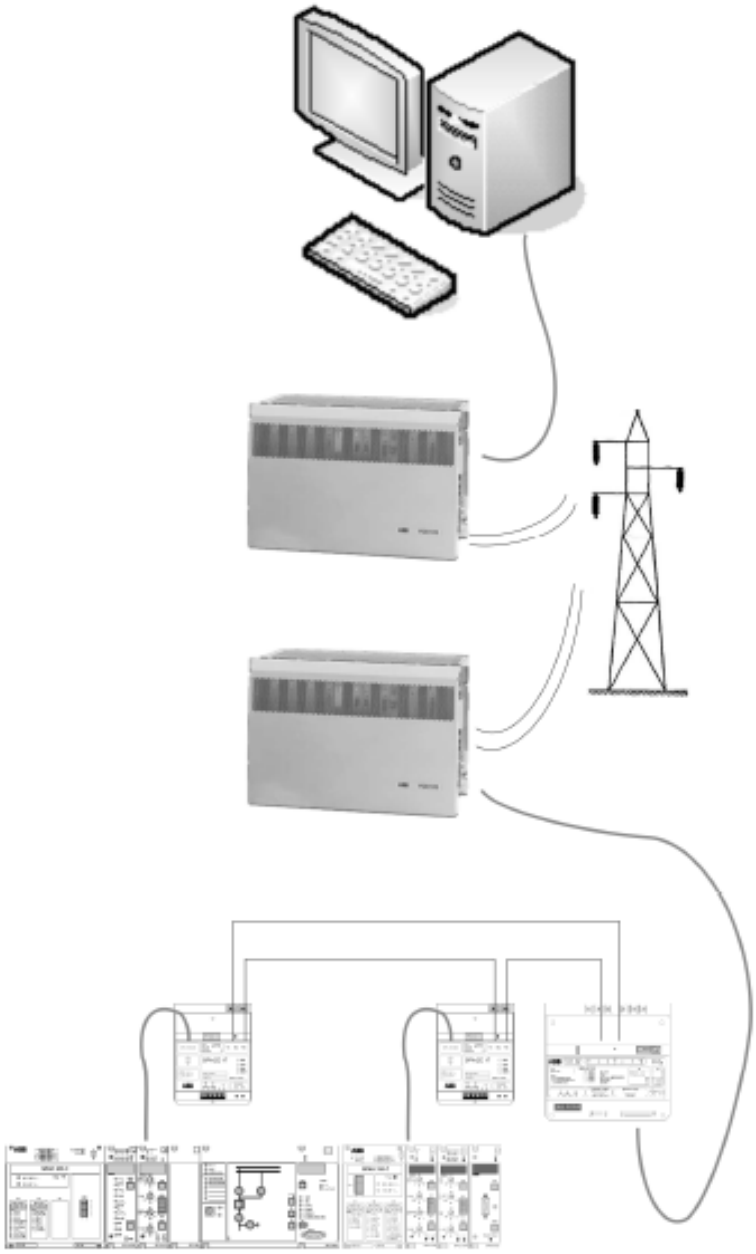
Relejna zaščita se ukvarja predvsem z zaščito delov primarnega elektroenergetskega omrežja in pa primarnega elektroenergetskega sistema v celoti. Deli primarnega sistema, ki imajo svojo zaščito, so:

- generatorji,
- transformatorji,
- zbiralke in
- vodi.

Primarni EES mora zagotavljati odjemalcem stalno razpoložljivo električno energijo predpisane kvalitete, ki jo definira oblika, efektivna vrednost in frekvenca omrežne napetosti. Naloga sekundarnega pa je zajemanje podatkov iz primarnega sistema, njih prenašanje, obdelava in na tej osnovi tudi ustrezno ukrepanje. Del tega sistema je tudi človek – operator. Obdelava, shranjevanje in ukrepanje je lahko ročno, delno ročno ali pa avtomatsko. Naloge zaščit so omejitve okvar, izločitev minimalnega dela EES, ki je v okvari, sprožitev samodejnih (avtomatskih) postopkov za vzpostavitev normalnega stanja, informiranje sistemov vodenja in nadzora o času, kraju in zaporedju dogodkov… Sam pojem releji pa se nanaša na naprave in sestave sekundarnega elektroenergetskega sistema, ki v primeru okvar ali nedopustnih obratovalnih stanj selektivno vplivajo na strukturo sistema in omogočijo vzpostavitev normalnega stanja.

Sekundarni sistem se lahko nahaja samo v treh področjih delovanja: v normalnem obratovalnem stanju, motenem obratovalnem stanju in pa havarijskem obratovalnem stanju. V **normalnem** obratovalnem stanju so vsi sistemski parametri znotraj dopustnih področij (napetost, frekvenca, pretok moči, harmonska popačenja), v sistemu obstaja zadostna rotacijska rezerva. Osnovno vlogo v nadzoru sistema opravlja primarna in sekundarna regulacija v elektrarnah in centrih vodenja. Osnovna zahteva za zaščito je, da ne deluje pri maksimalnih vrednostih tokov, pri minimalnih obratovalnih napetostih in v primeru prehodnih pojavov. **Moteno** stanje nastopi kot posledica izpadov proizvodnih, prenosnih ali distribucijskih kapacitet. Za indikatorje motenega stanja lahko uporabimo: odstopanje napetosti od nazivne vrednosti, tokovne obremenitve, ki presegajo nazivne vrednosti, pojav nične ali inverzne komponente toka, prekomerno segrevanje elementov sistema. Ko smo v motenem stanju, si najbolj želimo čim prej vrnitev v normalno obratovalno stanje. Za vzpostavitev normalnega stanja imajo osnovno vlogo avtomatika v elektrarnah, transformatorskih postajah in centrih vodenja. **Havarijsko** stanje sistema nastopi kot posledica okvar na elementih EES, ki se običajno pojavijo kot kratki stiki. Osnovno nalogo ima zaščita, ki mora čim prej ločiti mesto okvare od preostalega omrežja.

Slika 1 prikazuje uporabo računalnika (daljinsko vodenje in nadzor) z numerično zaščito na delu EES.



Slika 1: Daljinsko vodenje in nadzor. Vir: [3]

2 ZAŠČITNE NAPRAVE

Najvažnejši in običajno najzahtevnejši člen zaščitnega sistema je zaščitni rele. Zaščitni rele lahko predstavlja le en rele, lahko pa je to relejni sestav z več releji in sestavnimi deli v enem okrovu. Pogosto predstavlja zaščitni rele tudi relejna skupina z več električno oz. mehansko povezanimi relejnimi sestavi. Skozi čas so se razvili različni releji, ki jih lahko umestimo po času in načinu delovanja v naslednje skupine:

- I. generacija: elektromehanski releji
- II. generacija: statični releji
- III. generacija: numerični releji
- IV. generacija: adaptivni in »inteligentni« releji

Po funkciji razlikujemo naslednje vrste relejev:

- merilni (z dovolj veliko natančnostjo zaznava spremembo vzbujalne veličine),
- pomožni (za povečanje števila kontaktov, ločevanje tokokrogov, povečanje stikalne zmogljivosti kontaktov) in
- časovni (za doseganje zakasnitve delovanja zaščite).

Po osnovnem principu delovanja delimo releje na:

- elektromagnetne,
- bimetalne,
- elektrodinamične,
- indukcijske,
- elektromotorske ter
- elektronske.

Glede na zaznavano veličino so releji lahko:

- tokovni (pretokovni – nadtokovni),
- napetostni (podnapetostni ali prenapetostni),
- diferenčni,
- distančni,
- frekvenčni (podfrekvenčni),
- močnostni ali smerni,
- plinski,
- termični in
- časovni.

2.1 I. Generacija

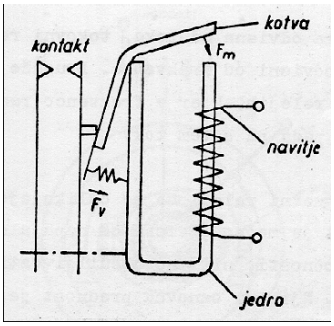
2.1.1 Elektromehanski releji

Večino zaščitnih relejev predstavljajo še vedno elektromehanski releji. Glede na fizikalne principe delovanja jih delimo na: elektromagnetne (Slika 2 in Slika 3), magnetno električne (Slika 4), indukcijske in termične.

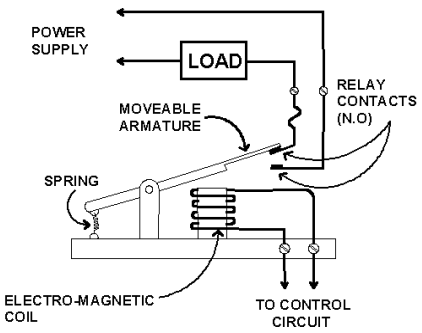
Elektromehanski releji imajo pomične in nepomične dele. Silo na pomične dele povzroča karakteristična veličina. Če je ta večja od nastavljive mehanske sile, rele deluje. Obvezni del releja je kontaktni stavek, ki ima običajno svoj pomični in nepomični del. Ker imajo merilni releji običajno le po en kontakt s sorazmerno nizko stikalno zmogljivostjo, uporabljamo običajno še izhodne pomožne releje z večjim številom kontaktov in z večjo stikalno zmogljivostjo.

Če je potrebno hitro delovanje zaščite, je potrebno, da delujeta hitro tako merilni rele kot tudi njegov pomožni rele. Ker je delovni čas običajnih pomožnih relejev približno 15 milisekund, uporabljamo v tem primeru posebno hitre releje z delovnim časom približno 1 milisekunde.

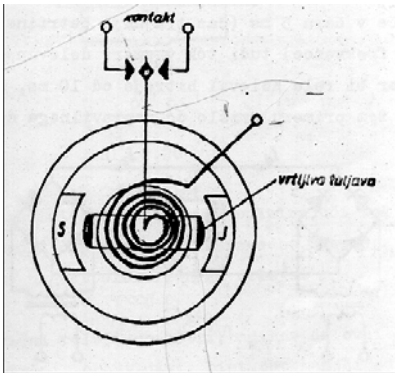
Njihova osnovna prednost je, da so enostavni in robustni in zato sorazmerno poceni in neobčutljivi na motnje, ki jih povzročajo prehodni pojavi v visokonapetostnem omrežju. Slaba pa je, da imajo veliko lastno porabo, so omejeni v realizaciji zaščitnih funkcij, omejene imajo možnosti povezovanja z drugimi releji, zahtevna izdelava, visoka cena in pa potreba po pogostem vzdrževanju.



Slika 2: Elektromagnetni del z gibljivim delom. Vir: [9]



Slika 3: Elektromagnetni rele. Vir: [13]



Slika 4 : Magnetno električni rele z vrtljivo tuljavo. Vir: [9]

2.2 II. Generacija

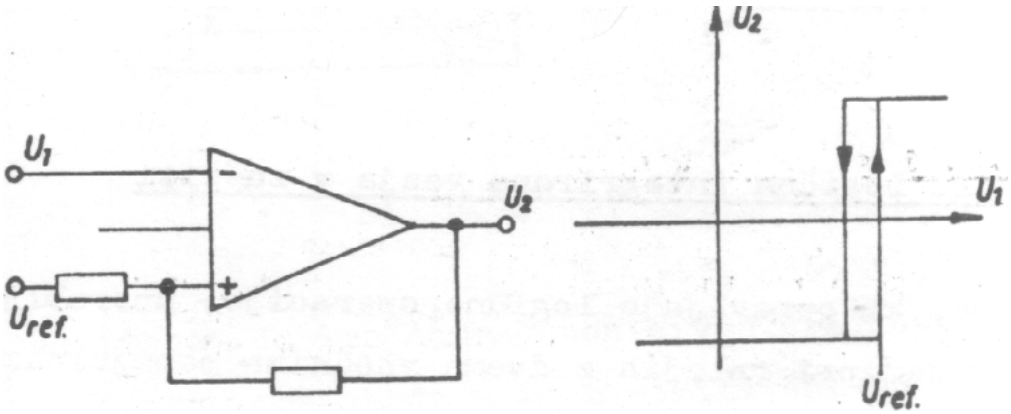
2.2.1 Statični releji

Statični releji so grajeni na osnovi analognih in digitalnih elektronskih komponent in integriranih vezij. Pri statičnih relejih se izvaja konverzija brez mehanskega gibanja in to predvsem z elektronskimi elementi. Statični rele je na izhodni strani lahko tudi opremljen z elektromehanskim relejem, obstajajo pa tudi popolnoma statične izvedbe, ki pa so sorazmerno redke. Tako imamo statične releje z izhodnimi kontakti in pa izvedbe brez izhodnih kontaktov.

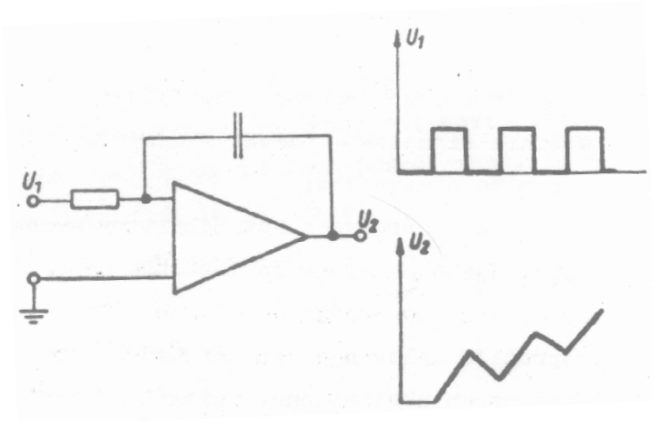
Kot že napisano, se v statičnih zaščitah uporabljajo linearna integrirana vezja, to pa so predvsem operacijski ojačevalniki, ki jih potrebujemo pri preoblikovanju vhodnih signalov.

Primerni so za:

- nivojske detektorje, predvsem kot Schmittov sprožilnik (trigger). Njihova naloga je ista kot naloga merilnih relejev z eno vhodno napajalno veličino, namreč, da sprožijo delovanje pri porastu kontrolirane veličine preko nastavljene, referenčne vrednosti. Schmittov sprožilnik omogoča dodatno, da je lahko tudi povratno razmerje nastavljivo.
- preoblikovalnike iz sinusnih v pravokotne oziroma iglaste impulze. Tako preoblikovanje je potrebno predvsem pri faznih komparatorjih.
- integratorje, ki integrirajo vhodne impulze (Slika 6)
- kot aktivne filtre. Iz sistema dobivamo v rele poleg osnovnega vala tudi višje harmonske komponente. Te komponente včasih koristno izrabljamo za delovanje ali pa za zadrževanje delovanja zaščite.



Slika 5 : Schmitov sprožilnik. Vir: [9]



Slika 6: Integrator vhodnih impulzov. Vir: [9]

Poleg tega pa se operacijske ojačevalnike uporablja kot komparatorje dveh vrednosti, od katerih je ena referenčna.

Poleg linearnih integriranih vezij se uporabljajo še digitalna integrirana vezja. Logične operacije, ki so nam na voljo, se uporabljajo predvsem pri relejih z dvema napajalnima veličinama, ki jim predhodno sinusno obliko pretvorimo oziroma spremenimo ali v pravokotno ali v iglasto obliko, s katero potem lahko operiramo.

2.2.2 Kombifleks

Kombifleks (kot primer statičnega releja z vsemi funkcionalnostmi) je modularen montažni sistem, ki nam omogoča skupno montažo raznih električnih in elektromehanskih elementov v funkcionalni kompletni sistem. Njegova značilnost je majhna lastna poraba, omejitve v realizaciji zaščitnih funkcij, možnosti povezovanja; vsaka zaščitna funkcija zahteva svoj modul in pa potrebo po vzdrževanju v rednih intervalih. Prednosti so, da so vse zaščite (tako imenovani moduli) integrirane na enem mestu in so med seboj povezljive.

Slika 7 : Kombifleks. Vir: [8]

2.3 III. Generacija

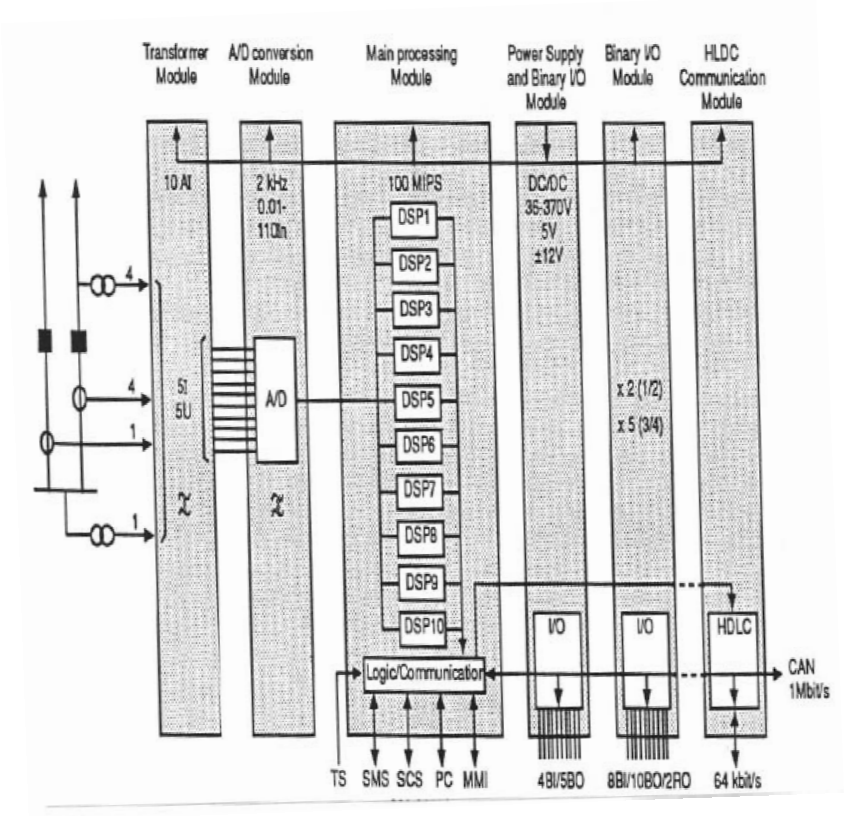
2.3.1 Numerični releji

Glede na releje (zaščite) prve in druge generacije, torej elektromehanske in statične, je numerični rele (zaščita) zasnovan predvsem na prilagodljivosti integracij različnih funkcij, razpoložljivosti in zanesljivosti. Izboljšanje razpoložljivosti je mogoče doseči predvsem na račun samonadzora, ki omogoča, da rele med obratovanjem nadzira delovanje procesorja, signalne tokokroge, vhodno-izhodne in pomnilniške enote. Odlikuje jih tudi sposobnost medsebojne komunikacije in povezanost s sistemom za vodenje in nadzor.

Razvoj teh relejev je povezan z dvema ključnima mejnikoma. Prvi, bolj teoretični, je bil konec šestdesetih ob raziskovanju možnosti digitalizacije naprav sekundarnega sistema (idejni začetnik je bil Rockefeller), drugi pravi razvoj pa je povezan z vpeljavo 8-bitnih mikroprocesorjev sredi sedemdesetih. Zaradi omejenih zmožnosti so 8-bitne hitro zamenjali zmogljivejši 16-bitni mikroprocesorji (predvsem Intel 8086 in Motorola 68000). Kljub izboljšanim zmogljivostim tudi ti procesorji postavljajo resne omejitve pri izvajanju računskih operacij, predvsem zaradi velikega števila množenj in seštevanj, ki so značilni za digitalno signalno obdelavo. Za skrajševanje časa obdelave je šel razvoj numeričnih relejev naprej. Tako so se sredi osemdesetih pojavili digitalni signalni procesorji (DSP), ki so se odlikovali po integriranih množilnikih znotraj samega procesorja, tako da je bilo mogoče opraviti množenje in seštevanje v enem ciklu. Šele pojav zmogljivih signalnih procesorjev v enojni in

kasneje v paralelni (večprocesorski) konfiguraciji je omogočil izvedbo tudi najbolj kompleksnih in računsko zahtevnih algoritmov. Numerični rele sestavlja več povezanih komponent, ki jih lahko razdelimo v tri osnovne funkcijske sklope: pripravo signalov, signalno pretvorbo in digitalno obdelavo. Prva dva sklopa sta skupna večini numeričnih relejev, medtem ko je sama digitalna obdelava odvisna od aplikacije in zgradbe algoritma zaščite.

Osnovne značilnosti naprav z mikroračunalniki so, da se lahko opravljajo različne logične operacije in iz njih izvedene aritmetične operacije sorazmerno hitro, da je možna uporaba sorazmerno velikega števila pomnilnih elementov in da je možno z eno napravo izvajati več različnih funkcij.



Slika 8 : Zgradba numeričnega releja Vir: [3]

2.3.2 Distančna zaščita

Distančna zaščita je ena izmed temeljnih zaščitnih naprav v elektroenergetskih sistemih. Primarni gradniki teh sistemov so vodi, generatorji, transformatorji, bremena ter redkeje še posebne naprave. Distančna zaščita se v veliki večini primerov uporablja za zaščito vodov in v manjši meri tudi za zaščito transformatorjev. V zadnjih letih se uporabljajo izključno numerični zaščitni releji.

Pred tem so se v ta namen uporabljali statični in še prej elektromehanski zaščitni releji. Primarni cilj distančne zaščite je selektivno delovanje v primeru napak (npr. na vodu). Distančna zaščita meri tokove in napetosti v vseh fazah na začetku (ali koncu) voda. Ker ima zgolj lokalne podatke o obeh veličinah, se v primeru pojava vplivnih faktorjev (dvojni vodi,

vmesno napajanje, kombinacija z prečnimi transformatorji) pojavi problem neselektivnega delovanja. Nasprotna konca enega voda sta navadno povezana s komunikacijsko povezavo. Preko te se izvajajo različne zaščitne sheme (PUTT, POTT, DTT, BTT, ..). V osnovi distančna zaščita zagotavlja selektivno delovanje za napake na ščitnem objektu ter rezervno zaščito za objekte zunaj osnovnega dosega (odvisno od stopnjevalnega načrta).

2.4 IV. Generacija

2.4.1 Inteligentni releji

Delovanje numeričnih zaščit se v zadnjem času skuša še dodatno izboljšati na osnovi prilagajanja (adaptacije) karakteristik in vključitve elementov umetne inteligence (mehka logika, umetna nevronska mreža, ekspertni sistemi, ...). Izboljšave se nanašajo predvsem na zaznavanje oziroma detekcijo, razvrščanje (kvalifikacijo) in lokacijo okvar pri spreminjajočih se obratovalnih razmerah EES-a. Značilnost strojne opreme teh relejev je predvsem na več namembnosti in povezljivosti, programske opreme pa po prilagajanju postopkov specifičnim zahtevam uporabnikov in poenostavitvi postopkom pri nastavljanju, konfiguriranju in usklajevanju zaščitnih funkcij.

2.5 Ocenjevanje osnovnih zahtev za zaščito

Pri ocenjevanju obstoječega, kot tudi pri načrtovanju novega zaščitnega sistema je potrebno upoštevati predvsem njegovo

- hitrost,
- občutljivost,
- selektivnost,
- zanesljivost in
- ekonomičnost.

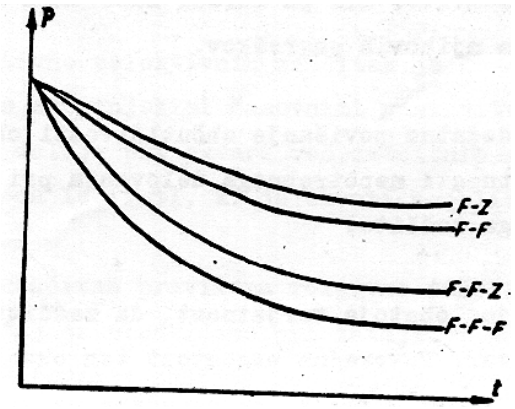
2.5.1 Hitrost delovanja

Stopnja uničenja opreme v primeru okvare bo manjša, če bo mesto okvare čim prej izločeno iz preostalega omrežja. Zaradi prehodnih pojavov ob nastopu okvare lahko pride do okvar še na drugih elementih sistema.

Vzroki so predvsem naslednji:

- posledice kratkega stika so pogosto sorazmerne času trajanja okvare,

- pri okvari pogosto prihaja v sistemu do napetostnih, tokovnih, termičnih in mehanskih preobremenitev. To lahko povzroči pri dalj časa trajajoči okvari še drugo okvaro v bližnji ali pa daljni okolici,



Slika 9: Hitrost. Vir: [9]

- zaradi okvare lahko pride do nestabilnega obratovanja. Slika 9 prikazuje podano možno trajanje okvare v vodu v odvisnosti od moči P, katero je vod prenašal pred okvaro. Krivulje podajajo za nek konkreten primer čase dopuščenega trajanja za štiri vrste okvar in sicer za trofazni kratki stik (P-P-P), za dvofazni kratki stik (F-F), za dvofazni zemeljski stik (P-P-Z) in za enofazni stik (F-Z).
- oblok se v zraku običajno s časom podaljšuje. S tem se povečuje tudi njegova upornost, kar lahko otežuje delovanje zaščite.

Vendar pa so zelo hitre zaščite običajno tudi dražje, manj selektivne, včasih pa tudi manj občutljive. Z nastopom okvare namreč dobimo tako v primarnih kot v sekundarnih tokokrogih prehodne pojave, ki pa se običajno hitro iznihajo. Zato se odločamo prvenstveno za hitre zaščite pri strojih in pri vodih visokih in najvišjih napetostih z delovnim časom releja približno 20 milisekund. V distribucijskih vodih, kjer cena zaščite igra važno vlogo, pa dopuščamo tudi delovne čase zaščite do nekaj sekund.

2.5.2 Selektivnost

Od zaščite zahtevamo, da izloči le tisti del sistema, na katerem je nastala okvara. Zato je celotni primarni sistem razdeljen na posamezna območja, ki ga pokrivajo različne zaščite. Ta območja zajemajo en, včasih pa tudi dva dela kot npr. blok generator - transformator oz. blok transformator - vod. Seveda se morajo območja med seboj delno prekrivati in to tako, da dobimo zanesljivo delovanje pri okvari na kateremkoli delu sistema.

Glede na selektivnost razlikujemo absolutno in relativno selektivne zaščite. Absolutno selektivne zaščite delujejo le pri okvarah v svojem območju in jih zato imenujemo tudi zaščite enote. Prednost teh zaščit je predvsem v tem, da delujejo sorazmerno hitro pri okvarah v celotnem območju zaščite, ki je točno definirano. Pri teh zaščitah pa je potrebna ustrezna rezerva, ki deluje pri zatajitvi te zaščite.

Pri relativno selektivnih zaščitah je dosežena selektivnost s stopenjskimi časovnimi nastavitvami. Pri teh zaščitah deluje pri okvari več zaščitnih relejev istočasno, toda le tisti, ki je okvari najbližji, opravi izklop. Pri teh zaščitah praviloma rezervne zaščite niso potrebne, saj je predvideno pri zatajitvi zaščite enega območja delovanje zaščite sosednjega območja. Vendar pa je območje delovanja le grobo določeno in zato imajo te zaščite tudi daljše čase delovanja. Ker se obe vrsti zaščite v glavnem dopolnjujeta, se vse pogosteje uporablja kombinacija obeh.

Doseganje selektivnosti:

- omejeno področje delovanja (diferenčna zaščita),
- s časovnim stopnjevanjem (stopnjevana nad tokovna zaščita),
- z uporabo zaščit, ki delujejo hitreje pri bližjih okvarah in zakasnjeno pri oddaljenih okvarah glede na relejno točko,
- z vpeljavo smernega releja in
- s povezovanjem zaščit na obeh koncih elementa.

2.5.3 Občutljivost

Pri zaščiti želimo, da bi delovala že pri minimalnih okvarah, pri tem pa se ne smejo odzivati pri maksimalnih vrednosti nadzorovanih veličin v motenih stanjih in ob prehodnih pojavih. Minimalne neodkrite okvare sicer običajno ne povzročajo večje škode, lahko pa postanejo zelo nevarne pri nastopu druge okvare. Pri nekaterih zaščitah smo omejeni z možnostjo zviševanja občutljivosti in to zaradi zaščitnega releja in njegove nastavitve ali pa zaradi zaščitnih transformatorjev in njihovih pogreškov.

Včasih pa dodatno povišanje občutljivosti občutno poviša verjetnosti nepotrebnega delovanja pri okvarah izven obsega zaščite. Zato običajno obstoja verjetnost, da zaščita ne odkrije vseh okvar.

Faktorji za omejevanje občutljivosti so:

- pogreški merilnih pretvornikov, motnje v signalnih in komunikacijskih tokokrogih,
- nizko razmerje med minimalnim kratkostičnim in maksimalnim bremenskim tokom v zankastih omrežjih in
- izraziti prehodni pojavi ob vklopu naprav in pri stikalnih manipulacijah.

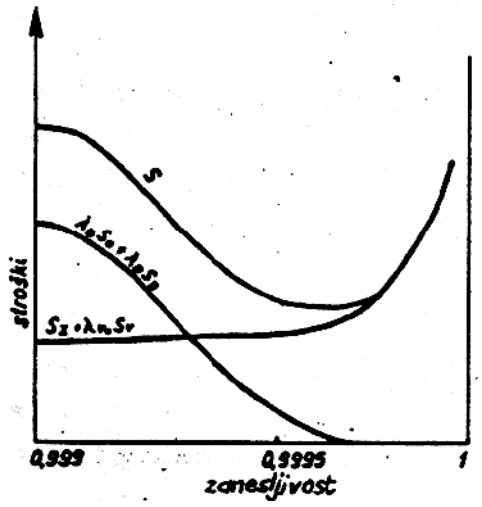
Zadostno občutljivost delovanja dosegamo predvsem z ustreznimi nastavitvami, omejenim področjem delovanja in zaščitami.

2.5.4 Ekonomičnost

Stroški za nabavo, vgradnjo, preizkušanje in vzdrževanje zaščitnih sistemov so v absolutnem merilu relativno visoki. Nekoliko drugačno sliko pa dobimo, če stroške za zaščito primerjamo

s stroški za primarno opremo. Iz primerjalni študij v različni EES-ih izhaja, da investicijski stroški za zaščito v splošnem ne presegajo 3 % vrednosti primarne opreme. V ekonomski analizi, ki vpliva na izbiro zaščitnega sestava, pa je potrebno upoštevati tudi ostale stroške, ki so vezani na obratovanje (vzdrževanje, šolanje operaterjev, stroški nedobavljene energije zaradi napačnega delovanja zaščit, stroški zaradi nepotrebnega delovanja, stroški samega delovanja naprave), pa tudi pomembnost objekta za konkreten EES in s tem povezane posebne zahteve po zanesljivosti in razpoložljivosti. Zaščitne sisteme torej načrtujemo tako, da dosežemo razumno razmerje investicijskih in obratovalnih stroškov med primarno in sekundarno opremo.

S poviševanjem zanesljivosti rastejo stroški za zaščitne naprave in za njihovo vzdrževanje, padajo pa stroški, povezani z odpovedmi delovanja zaščite ali zaradi nepotrebnega delovanja zaščite. Za konkreten primer so podani stroški v odvisnosti od zanesljivosti (Slika 10). Kot sledi iz slike, so minimalni stroški za ta primer pri zanesljivosti 0,99965.



Slika 10: Ekonomičnost. Vir: [9]

2.5.5 Zanesljivost

Zaščitne naprave morajo delovati zanesljivo, kar pomeni, da pri okvarah ne zatajijo in da ne delajo po nepotrebnem v motenih stanjih in ob prehodnih pojavih. Zanesljivost je tako podana kot verjetnost, da bo zaščita opravila predvideno funkcijo v danih pogojih in v danem časovnem intervalu. Verjetnost, da zaščita ne bo zatajila je podana z razpoložljivostjo, verjetnost, da zaščita ne bo delovala po nepotrebnem, pa s sigurnostjo. Nepravilno delovanje je lahko posledica osnovne odpovedi zaradi napak v načrtovanju, nastavitvah ali uporabi zaščite. Odpoved zaščite pa razumemo kot nepravilno delovanje zaradi napak v elementu zaščite.

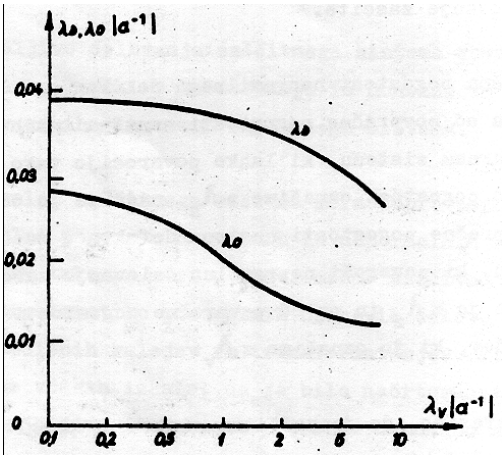
Na zanesljivost klasičnih elektromehanskih in statičnih zaščit je mogoče vplivati z rednim periodičnim vzdrževanjem, medtem ko lahko zanesljivost numeričnih zaščit bistveno povečamo z vpeljavo funkcij sprotnega samonadzora.

Na razpoložljivost in sigurnost lahko vplivamo tudi z redundanco, tako da določeno zaščitno funkcijo izvaja več relejev, neodvisno drug od drugega. Redundanco lahko izvedemo z vzporedno in zaporedno vezavo relejev. Pri vzporedni vezavi se zniža verjetnost izostalega delovanja, poveča pa nepotrebno delovanje. Vzporedno uporabljamo tam, kjer bi bile posledice izostalega delovanja izredno hude (generatorji, transformatorji, pomembnejši vodi). Nasprotno pa z zaporedno vezavo relejev znižamo verjetnost nepotrebnega delovanja, povečamo pa verjetnost izostalega delovanja (npr. pri zaščiti zbiralk).

Največ težav pa povzročajo defekti v zaščitnih napravah. Ker delujejo nekatere zaščite zelo redko, se lahko defekt odkrije le takrat, ko zaščita deluje nepotrebno ali pri okvari zataji. Lahko pa se odkrije tudi pri vzdrževalnih pregledih.
Zaščita lahko deluje nepravilno zaradi defekta v zaščitnem sistemu, ki zaradi povišane občutljivosti povzroči nepotrebno delovanje, ali pa zaradi defekta, ki zmanjša občutljivost zaščite oz. celo prepreči potrebno delovanje zaščite.

Povprečna pogostost nepravilnega delovanja zaščite je odvisna od povprečne pogostosti nastopa takih razmer v primarnem sistemu, ki lahko povzročijo tako delovanje.

Slika 11 nam prikazuje podani krivulji, ki podajata pogostost nepotrebnih delovanj zaščite oz. izostalih delovanj zaščite v odvisnosti od pogostosti pregledov.



Slika 11 : Zanesljivost. Vir: [9]

Spodnja krivulja, ki prikazuje pogostost izostalih delovanj, je izračunana za primer, da nastopita na leto dve okvari na varovanem delu, da pride do defekta na zaščitnem sistemu, ki povzroči zatajitev delovanja zaščite 1 krat na 100 let in da zaščita enkrat na sto let ne deluje pri okvari zaradi nezadostne občutljivosti.

Zgornja krivulja prikazuje pogostost nepotrebnih delovanj. Določena je za primer, da nastopa v sistemu na leto 20 okvar, pri katerih pa naj zaščita varovanega dela ne deluje, da pride

1 krat na 100 let do defekta v zaščitnem sistemu, zaradi katerega zaščita po nepotrebnem deluje pri okvarah v sistemu in pa da 1 krat na 100 let deluje zaradi svoje karakteristike neselektivno.

2.5.6 Dodatne zahteve

Dodatne zahteve, za ocenjevanje kakovosti zaščite:

- Povezljivost (kompatibilnost); od zaščit zahtevamo, da so povezljive z obstoječimi napravami primarnega in sekundarnega sistema.
- Zmožnost, da se zaščita vključi v informacijski sistem za nadzor EES in da komunicira ostalimi zaščitami.
- Prilagodljivost (fleksibilnost); zmožnost prilagajanja karakteristike zaščite obratovalnim pogojem in konfiguracijam omrežja.
- Samonadzor; zmožnost numeričnih zaščit; tako, da sproti preverjamo stanje napajalnih in krmilnih tokokrogov, primerjamo aktualne signale z referenčnimi, nadzorujemo delovanje procesorja (watch-dog) in občasno sprožimo testiranje strojne opreme.

3 IMPLEMENTACIJA PRENAPETOSTNE ZAŠČITE V VHDL

3.1 Uvod

V naslednjih dveh opisih bomo spoznali, kaj je prenapetostna zaščita ter zakaj in kako jo implementirati v VHDL-u. Orodje, ki sem ga uporabil, je program Active-HDL, ki ga uporabimo za opis in simulacijo digitalnih vezij. Tretji del bo natančna obrazložitev delovanja algoritma, ki bo ponazorjena s slikami in pa samo kodo.

3.2 Prenapetostna zaščita

Prenapetostna zaščita je namenjena detekciji in zaščiti izvoda in uporabnikov pred prevelikimi napetostmi.

Obratovanje pri povišani napetosti lahko nastopi pri delih primarnega sistema, kot posledica okvar na sistemu napetostne regulacije ali zaradi nenadnih razbremenitev v omrežju. Povišane napetosti dodatno obremenjujejo izolacijo, povečujejo pa se tudi vrtinčne izgube.

Relejna prenapetostna zaščita je namenjena zaščiti EES pred prenapetosti obratovalnih frekvenc. Za zaščito pred drugimi prenapetostmi, predvsem pred atmosferskimi in stikalnimi prenapetostmi, ki jih včasih namestimo tudi na generatorske sponke.

Do povišanja napetosti obratovalnih frekvenc prihaja tudi v normalnem obratovanju in sicer pri trenutnih razbremenitvah. Tem razbremenitvam sledi tudi povišanje vrtljajev, ki je pri turbo generatorjih le nekaj procentov, občutno višje pa je pri nekaterih vodnih turbinah in to zaradi omejitev pri zapiranju turbine. Prenapetostna zaščita deluje pri prenapetostih, ki so višje od prenapetosti pri popolni razbremenitvi agregata. Pri hidro agregatih, kjer nastopi dodatno povišanje vrtljajev, pa uporabimo običajno tudi zakasnjeno prenapetostno stopnjo, ki je nastavljena na nižjo vrednost kot prva, trenutna stopnja. Pri prenapetostnih zaščitah EES-a, ki se jim lahko vrtljaji občutno dvignejo, mora biti rele frekvenčno neodvisen.

Prenapetostni rele in napetostni regulator priključujemo običajno na različne napetostne transformatorje. S tem dobimo pravilno delovanje prenapetostne zaščite tudi pri okvari na napetostnem transformatorju napetostnega regulatorja oz. pri okvari na njegovem sekundarnem tokokrogu.

Najbolj pogosti izvori prenapetosti:

- ***Prenapetosti zaradi slučajnih napak*** - povzroči jih človek z nepazljivim ali nestrokovnim posegom.
- ***Prenapetosti zaradi statičnih razelektritev***
- ***Prenapetosti zaradi različnih potencialov ozemljitev*** - napaka se pogosto pojavi, ko skušamo povezati dve voda, ki imata različne ozemljitvene potenciale.

- ***Prenapetosti zaradi elektromagnetnih motenj*** - vodi ustvarjajo elektromagnetna polja, ki ob spreminjanju jakosti ali smeri polja v vodnikih inducirajo napetosti.

3.3 VHDL

Tehnologija programirljivih vezij (FPGA, CPLD) je danes zelo razširjena, saj lahko z majhnimi stroški naredimo celoten sistem na integriranem vezju. Zmogljivost programirljivih vezij izredno narašča, z njo pa se povečujejo izdelave različnih aplikacij. Z visokonivojskim jezikom VHDL lahko opišemo in modeliramo vezja. Programska oprema za sintezo vezij nam iz takšnega opisa naredi vezje na nivoju logičnih vrat. Zgradbo vezja nam določajo strukture, ki jih razdelimo na krmilne avtomate in podatkovne strukture. To je funkcijski model za obdelavo podatkov oziroma signalov na nivoju RTL-a (Register Transfer Level). RTL nam natančno določa stanje vhodnih in izhodnih signalov ob vsakem urnem ciklu.

Primeri aplikacij, realiziranih na FPGA z jezikom VHDL:

- digitalna obdelava signalov, zvoka in videa,
- digitalni radio in televizija,
- omrežne naprave (Ethernet, CAN, brezžična omrežja),
- vmesniki (PCI, DDR) in
- hitre vzporedne računske naprave.

Dandanes je pri načrtovanju produkta oziroma neke aplikacije pomembnih več dejavnikov oziroma lastnosti. Nekateri najpomembnejši izmed njih so vsekakor:

- cena,
- zmogljivost,
- hitrost,
- nizka poraba energije,
- kakovost itd.

V primeru, da gradimo produkt na osnovi *FPGA* oz. *logičnih programirljivih vezij*, dobimo vse te lastnosti. Digitalni sistemi, zgrajeni s programirljivimi vezji, imajo več prednosti glede na enake sisteme, ki so zgrajeni z mikroprocesorjem. Najpomembnejše prednosti so hitrost delovanja, majhna poraba energije in zanesljivost delovanja aplikacij v realnem času. Obdelava podatkov poteka z majhnimi zakasnitvami. Vgrajeno aplikacijo pri namensko načrtovanih sistemih odlikuje velika prilagodljivost. Velika prednost sistemov FPGA je tudi sočasnost delovanja logičnih gradnikov in sklopov ter reprogramiranje. Pri veliko manjši frekvenci ure, kot je potrebna za delovanje mikroprocesorja, izvajamo operacije v realnem času. Sisteme z FPGA vezji običajno uporabljamo v različnih kombinacijah z mikroprocesorjem.

Če se vrnemo nazaj na že napisano, da se trenutno za zaščito EES-a uporabljajo v večini zaščite tretje generacije in sicer numerični releji z mikroprocesorji in zgoraj navedene lastnosti VHDL-a, potem lahko to povežemo z idejo, da bi nek zaščitni algoritem implementirali v VHDL-u za numerični rele. Pri tem nam je v pomoč zgradba numeričnega releja, ki že vsebuje vezje FPGA za pripravo in obdelavo signalov za mikroprocesor. Z

implementacijo algoritma v FPGA razbremenimo mikroprocesor, ki ga lahko tudi odstranimo. Pri tem pa nič ne izgubimo, ampak kvečjemu lahko pridobimo na zgoraj navedenih lastnostih.

3.4 Opis delovanja

Pri zasnovi algoritma moramo upoštevati določila, navedena v poglavju 2.5. S tem bomo implementirali algoritem zaščite, ki bo ustrezal standardom, ki jih določa IEC. Delovanje algoritma mora biti jasno in robni pogoji morajo biti natančno določeni, ker si ne smemo privoščiti, da bi algoritem napačno deloval, saj to lahko povzroči enormne stroške. Zato je dobro, da pred samim kodiranjem naredimo dober načrt (dizajn), s katerim podkrepimo naše razmišljanje in samo izvedbo. Pomembno je, da si razjasnimo kako algoritem deluje, katere vhodne parametre imamo na voljo, kako jih moramo upoštevati, da dobimo pravilno delovanje v obliki pravilnega izhoda. Za programski jezik VHDL je jasnost vhodov in izhodov zelo pomembna.

V naslednjem koraku si bomo pogledali vse potrebne vhode in izhode za realizacijo prenapetostne zaščite. Z njihovim opisom se bomo seznanili z vlogo, ki jim je dodana, s slikami/grafi pa si bomo ogledali njihovo delovanje.

Opis veličin parametrov vhodov in izhodov, ki jih bomo pri realizaciji potrebovali:

- *pickupvalue* – definira mejo območja, kjer zaščita prime (možna relativna nastavitev)
- *dropoutvalue* – definira mejo, kjer zaščita odpusti (možna relativna nastavitev)
- *pickup* (signal) – izhodni signal, ki se postavi, ko poteče *pickupdelay* in je opazovana veličina še vedno v območju delovanja zaščite.
- *pickupdelay* - časovna stabilizacija elementa *pickup* (*pickupvalue* + *dropoutvalue*) pred prijemom. Čas, ko je vrednost opazovane veličine v območju delovanja zaščite (*pickup*), vendar zaščita še ne prime. Kot filter za motnje in večjo robustnost. Časovnik na stalen signal.
- *dropoutdelay* - časovna stabilizacija elementa *pickup* (*pickupvalue* + *dropoutvalue*) po prijemu. Čas, ko je vrednost opazovane veličine izven območja delovanja zaščite (*pickup*), vendar zaščita še ne spusti. Časovnik na stalen signal.
- *tripdelay* – zakasnitev izhoda *trip*. Časovnik na stalen signal *Pickup*.
- *minimal_pulse* – minimalni izhodni pulz samo za izhod *trip*. Časovnik se sproži vedno, ko se postavi trip. Časovnik na prehod.
- *alarm* – izhodni signal, ko je nekaj narobe v delovanju algoritma
- *amplituda* – vhodna opazovana veličina, s katero se preverja območje delovanja zaščite
- *trip* – izhodni signal, ko je opazovana veličina v delovanju zaščite in poteče *tripdelay*.
- *pickup_block* – vhodni signal, ki ponazarja blokado za signal *trip*.
- *pickup_blocked* – izhodni signal, ko je blokiran *pickup*.

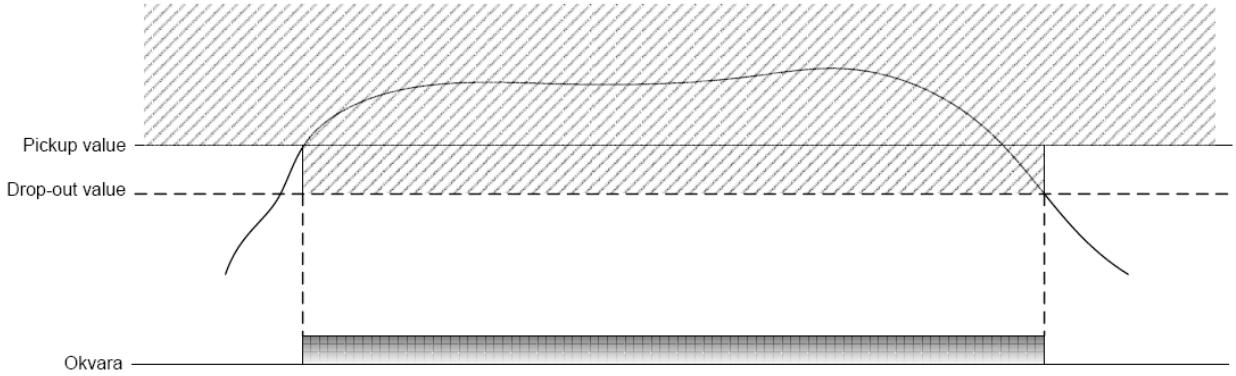
Časovnik (ang. timer) je razred oziroma komponenta, ki omogoča izvajanje programa v več nitih. Z njim lahko nastavimo, po kolikšnem času oziroma ob katerem točno določenem času naj se začne izvajati neka nit programa.

Tipi časovnikov, katere uporabljamo:

- Časovnik na stalen signal: časovnik steče, ko je signal prisoten. Če se signal vmes prekine, se časovnik resetira in ustavi.
- Časovnik na prehod: časovnik steče, ko se signal postavi, t.j. preide iz 0 na 1. Časovnik teče do konca, ne glede na signal.

3.4.1 Zaznavanje okvare

Okvara je nezaželeno, vendar neizbežno stanje prenapetostne zaščite. Z njo se identificira napaka. Na njeni podlagi se vklopi zaščita. Območje delovanja zaščite je vedno nad *pickupvalue*, ko opazovana veličina preseže vrednost *pickupvalue* in preidemo v stanje okvare ter do *dropoutvalue*, dokler na pade pod vrednost *dropoutvalue*, ko okvara ni več prisotna. Slika 12 nam prikazuje prisotnost okvare.



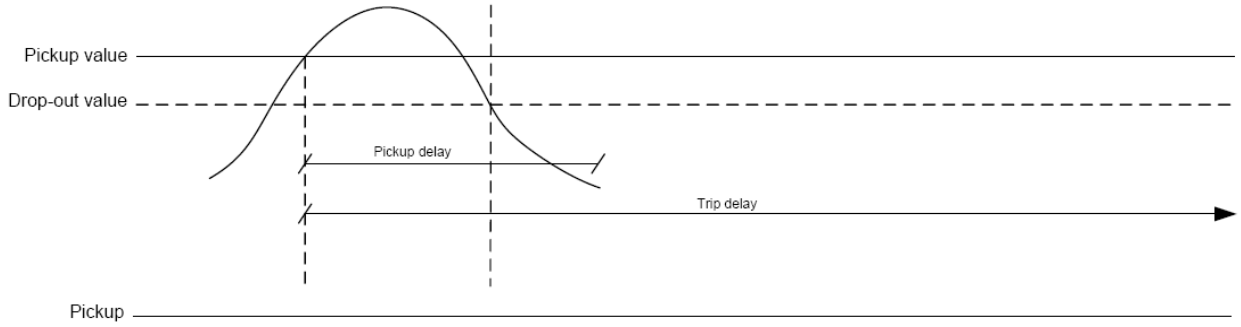
Slika 12: Okvara

3.4.2 Zaznavanje signala *pickup*

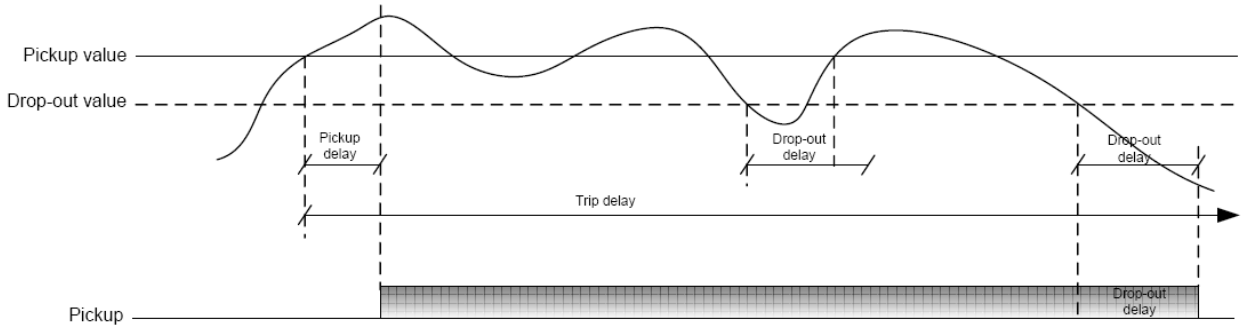
Signal *pickup* prime, ko poteče *pickupdelay* in je opazovana veličina še vedno v območju delovanja zaščite. Slika 13 prikazuje primer, ko je opazovana veličina iz območja okvare. Slika 14 pa prikazuje postavljen signal *pickup*.

Signal *pickup* pade v dveh primerih:

- Pred nastopom signala *trip*: ko opazovana veličina pade iz območja delovanja zaščite in poteče *dropoutdelay*.
- Po nastopu signala *trip*: ko opazovana veličina pade iz območja delovanja zaščite.



Slika 13: *Pickupdelay* ne poteče, ni pristne okvare.



Slika 14: Postavljen signal *pickup*.

3.4.3 Zaznavanje signala *trip*

Signal *trip* je signal, ki nam vklopi zaščito. Ko ga določimo, se identificira vklop zaščite. Na njeni podlagi pa se izklopijo npr visokonapetostne naprave (odklopniki, ločilniki, …).

Signal *trip* se postavi:

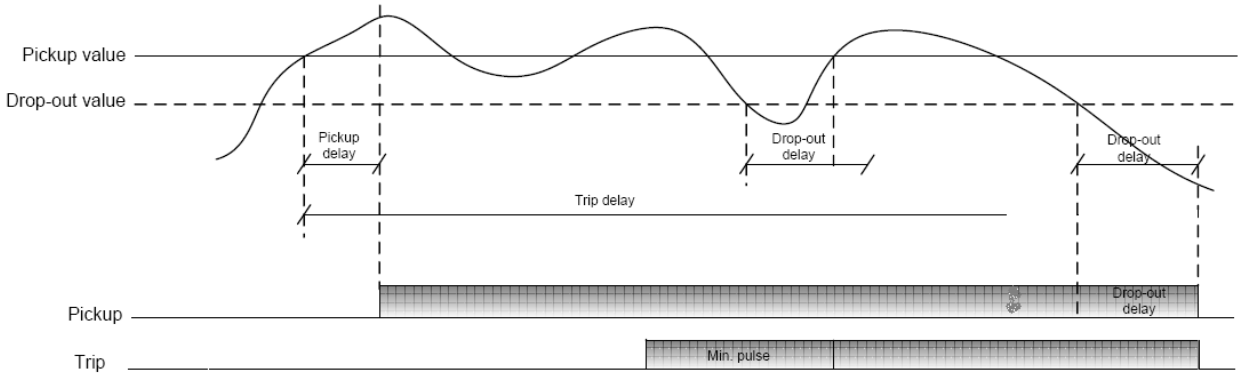
- ko je veličina v območju delovanja zaščite in poteče *tripdelay* (Slika 15).

Signal *trip* pade:

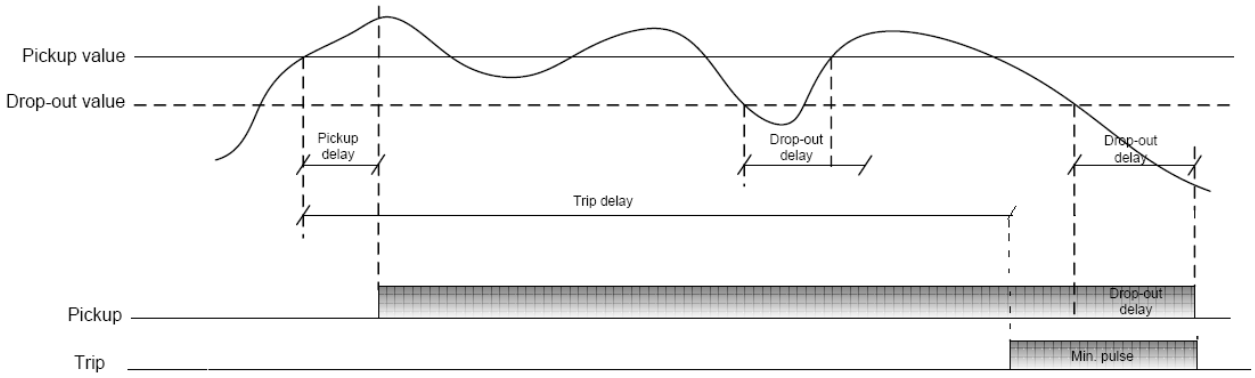
- ko signal *pickup* pade in poteče čas *minimal pulse* (Slika 16).

Časovnik *tripdelay* starta, ko pride opazovana veličina v območje delovanja zaščite. Časovnik *tripdelay* se resetira:

- Pred nastopom signala *pickup*: ko opazovana veličina pade iz območja delovanja zaščite.
- Po nastopu signala *pickup*: ko signal *pickup* pade.



Slika 15: Signal *trip* (1)



Slika 16: Signal *trip* (2)

3.4.4 Blokade

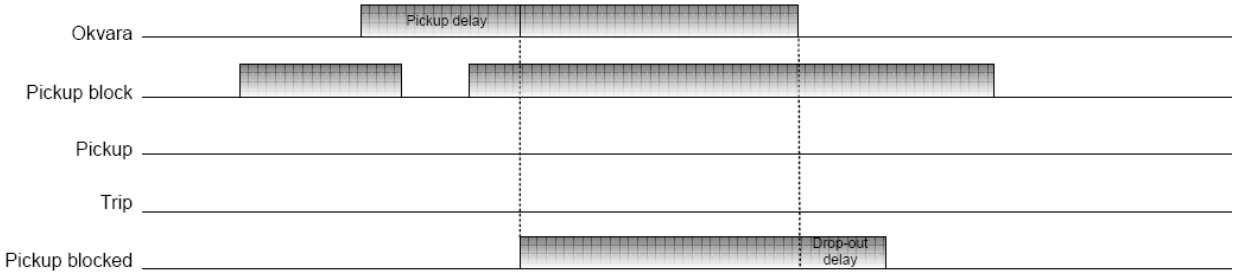
Blokade so zelo pomembne v zaščiti EES. Marsikdaj pridemo v situacijo, ko vemo vnaprej, da bo prišlo do povečane napetosti na primer ob direktnem zagonu velikih motorjev ali ob vklopu visokonapetostnih kondenzatorskih baterij. Pri teh prehodnih pojavih pa ne želimo, da se vklopi zaščita, saj razlog za povečano napetost nadzorujemo in ne le, da bi bil vklop brez potrebe, tudi napačen bi bil. Saj, ko bi želeli nekaj priklopiti na omrežje, kar bi sprožilo prehodni pojav, nam bi zaščita to izklopila. Za takšne primere je idealna uporaba blokad.

3.4.4.1 Delovanje blokad

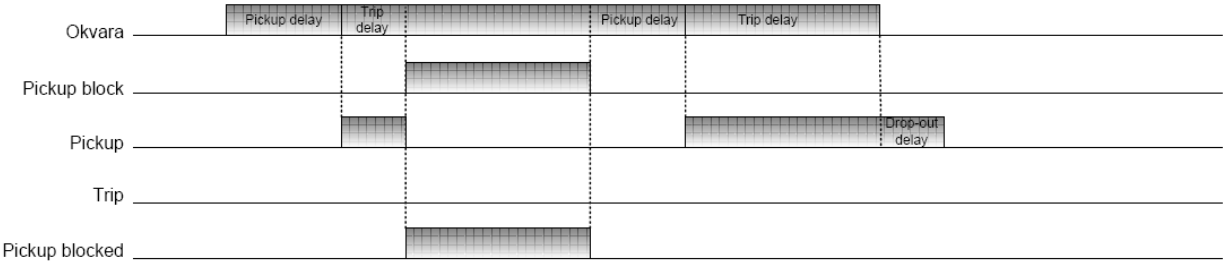
Signal *pickup* je blokiran, ko se postavi blokada *pickup block*. Namesto njega se postavlja signal *pickup blocked*. Če se signal *pickup blocked* postavi v času, potem:

- Signal *tripdelay*: prekine signal *pickup* in resetira časovnik.
- Signal *trip*: prekine signal *trip*, vendar ne pred iztekom *minimal pulse time*.
- Blokiran trip: prekine signal *pickup* in resetira časovnik.

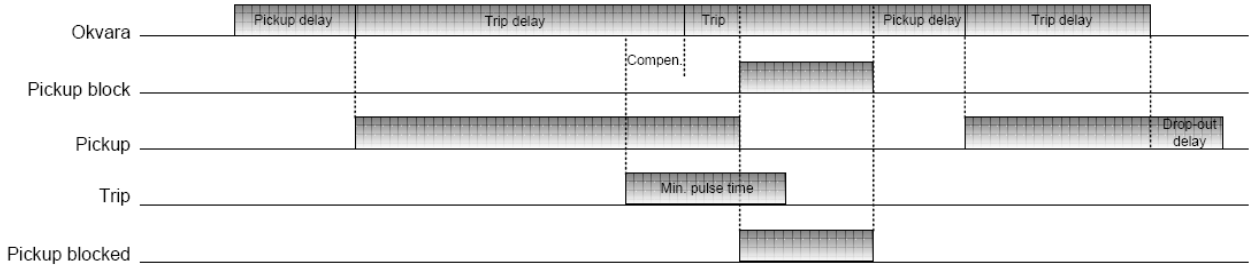
Ko *pickup_block* signal pade, se ponovno starta *pickupdelay*.



Slika 17: Signal *pickup_block* pred okvaro



Slika 18: Signal *pickup_block* pride in pade med *Tripdelay*-em



Slika 19: Signal *pickup_block* pride in pade med *tripdelay*-em

3.5 Opis VHDL kode

Za realizacijo takšnega algoritma sem uporabil možnost končnega avtomata (ang. finite state machine). Pri opisu končnega avtomata se moramo držati določenih pravil in sicer:

- pravilo je, da se pogoj za fronto ure pojavi le enkrat in le na enem mestu končnega gradnika in
- če v vezju ne želimo imeti zapahov, moramo definirati vrednosti signalov v vseh primerih.

Realizacija končnega avtomata je vezje, ki generira izhodne signale na podlagi notranjega stanja in vhodnih signalov. Za zapis stanj avtomata v jeziku VHDL definiramo nov podatkovni tip, v katerem naštejemo vsa možna stanja. Spremenljivko, ki hrani stanje avtomata, deklariramo kot notranji signal.

Tukaj je treba omeniti še konstrukt *proces*, ki nam omogoča izvajanje zaporednih končnih stavkov. Stavki v procesu se izvedejo vsakokrat, ko se spremeni vrednost kateregakoli signala iz seznama signalov, na katere je proces občutljiv in so mu podani ob deklaraciji procesa. Znotraj arhitekture je možnih več procesov, ki so ločeni z unikatno oznako.

V algoritmu sem uporabil tri procese: *ura_in_reset*, *iskanje_okvare* in *izvrsitev_okvare*.

Prvi proces je občutljiv na uro in na vhodni signal *reset*. V primeru *reseta* se postavimo v začetno stanje in resetiramo časovnike. V drugem primeru spremembe procesa, torej ob prehodu ure iz 0 v 1, nastavimo notranja signala, ki hranita stanje avtomata na njihovo naslednje stanje. Ostala dva proces pa nam služita, kot je že razvidno iz njihovega imena za iskanje okvare in izvršitev zaščite. V obeh procesih sta realizirana za delovanje končna avtomata.
Slika 21 in Slika 22 nam prikazujeta diagram prehajanja stanj za ta dva procesa, ki je osnova za opis končnega avtomata.

Zaradi trivialnosti rešitve prvega procesa je zanj podana le koda brez diagrama (Slika 20).

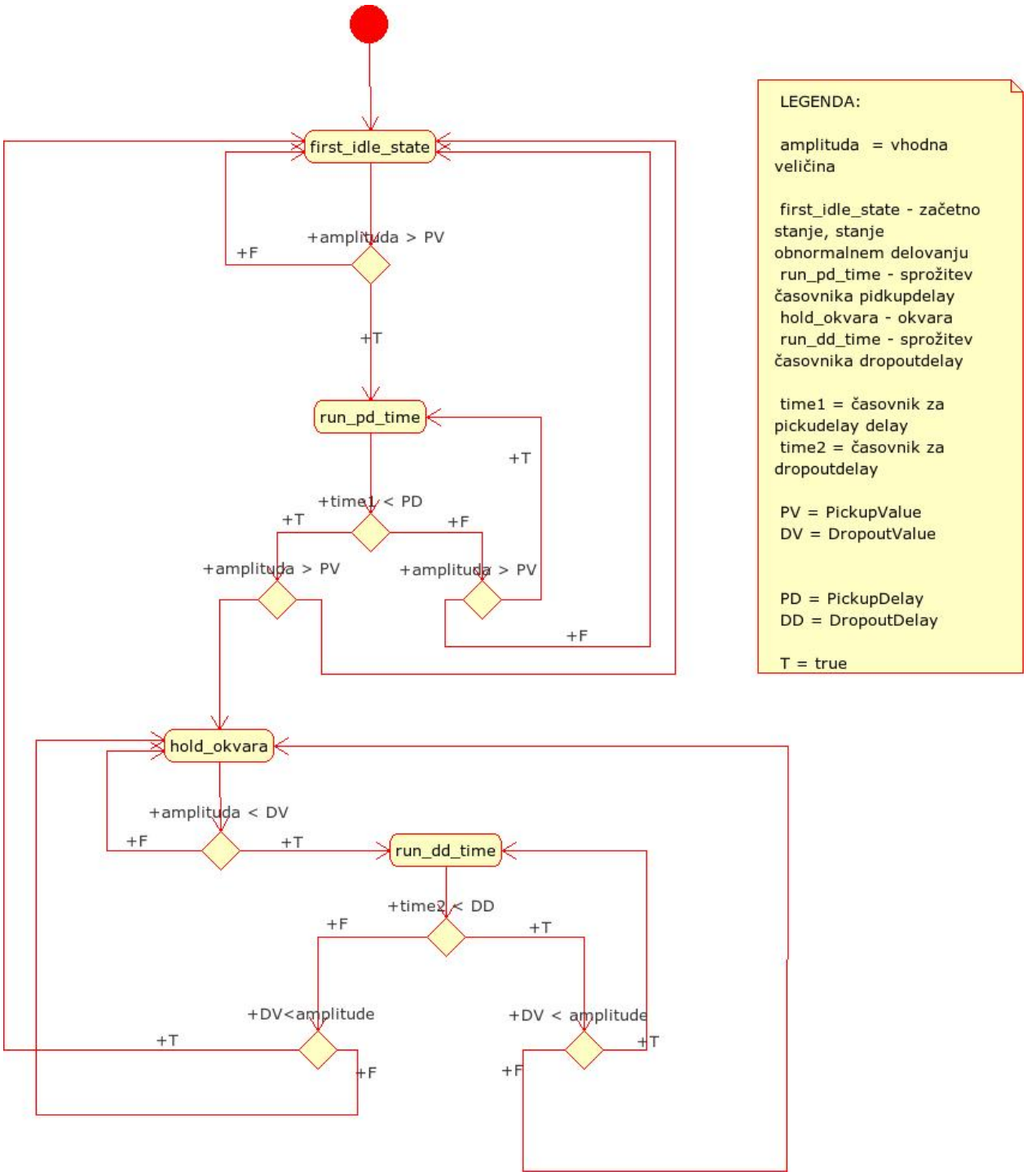
```
URA_IN_RESET: process (clk, reset)
begin
    if reset = '1' then
        stanje <= first_idle_state;
        stanje_izvrsitev <= zacetno_stanje_izvrsitev;
        time1 <= 0;
        time2 <= 0;
        time3 <= 0;
        time4 <= 0;
    elsif clk'event and clk='1' then
        stanje <= naslednje;
        stanje_izvrsitev <= naslednja_izvrsitev;
        time1 <= time1d;
        time2 <= time2d;
        time3 <= time3d;
        time4 <= time4d;
        if error_1 = '1' and error_2 = '1' then
            alarm <= '1';
        end if;
    end if;
 end process;
```

Slika 20: Proces URA_IN_RESET

3.5.1 Proces iskanja okvare

Avtomat za iskanje okvare nam služi, da identificiramo okvaro. Pri opisu se za lažje razumevanje sklicujemo na sliko diagrama (Slika 21). V procesu iskanja imamo štiri stanja: *first_idle_state*, *run_pd_time*, *hold_okvara* in *run_dd_time*.

Začetno stanje, ki je hkrati tudi idealno stanje, je stanje, ko je zaščita v normalnem obratovanju. Takrat je vhodna *amplituda* v območju delovanja. V praksi pomeni, da je amplituda manjša od vrednosti *pickup*. V primeru, da *amplituda* preseže to vrednost sprožimo časovnik za *pickupdelay* in se prestavimo v naslednje stanje – »*run_pd_time*«. V tem stanju preverjamo, ali je potekel čas *pickupdelay*. V primeru, da je čas potekel in je *amplituda* še vedno nad dovoljeno vrednostjo *pickupvalue*, se prestavimo v naslednje stanje »*hold_okvara*«. Če *amplituda* pade iz območja okvare, potekel pa še ni *pickupdelay*, se vrnemo v prvo stanje (Slika 13). Dogodek, ki se nam je zgodil, pa označimo za motnjo. Ob vsakem urinem prehodu iz 0 v 1 preverjamo, kakšna je vhodna vrednost *amplitude* glede na nastavljeno histerezo. Stanje »*hold_okvara*« je stanje, ko je prisotna okvara (Slika 12). V tem stanju nastavimo notranji signal *okvara* na vrednost 1. Na notranji signal *okvara* je občutljiv naslednji proces, ki izvršuje *okvaro*. Vedno, ko je prisotna *okvara* ima signal vrednost 1. V primeru, da vhodna *amplituda* pade pod vrednost *dropoutvalue*, se sproži časovnik *dropoutdelay* in se prestavimo v naslednje stanje »*run_dd_time*«. Če ne, ostanemo v trenutnem stanju. V stanju »*run_pd_time*« je še vedno v območju okvare in notranji signal *okvara* je še vedno prisoten! V tem stanju, ki je tudi zadnje stanje v procesu iskanje okvare, preverjamo ali je vrednost vhodne *amplitude* dovolj dolgo (*dropoutdelay*) pod spodnjo vrednostjo histereze (*dropoutvalue*). Če v času *dropoutdelay* vhodna vrednost *amplitude* preseže dovoljeno, dogodek označimo kot motnja in se vrnemo v predhodno stanje »*hold_okvara*«. Ko pa čas poteče, in je *amplituda* pod spodnjo vrednostjo, se vrnemo v začetno stanje. Nahajamo se v idealnem stanju. Notranji signal *okvara* se nastavi na vrednost 0. V praksi to pomeni, da je vhodna *amplituda* v območju dovoljenih vrednostih in zaščita ni vklopljena.



Slika 21: Avtomat za iskanje okvare

3.5.2 Proces izvajanja zaščite

Slika 22 prikazuje proces izvajanje zaščite. Ima štiri stanja: *zacetno_stanje_izvrsitev*, *izvrsite_trip_delay*, *izvrsitev_takoj* in *izvrsitev_blokada*. V vseh stanjih se najprej preverja pogoj, če je notranji signal okvara postavljen, razen v stanju *izvrsitev_takoj*, ki se preverja nekoliko kasneje, ampak o tem pozneje.

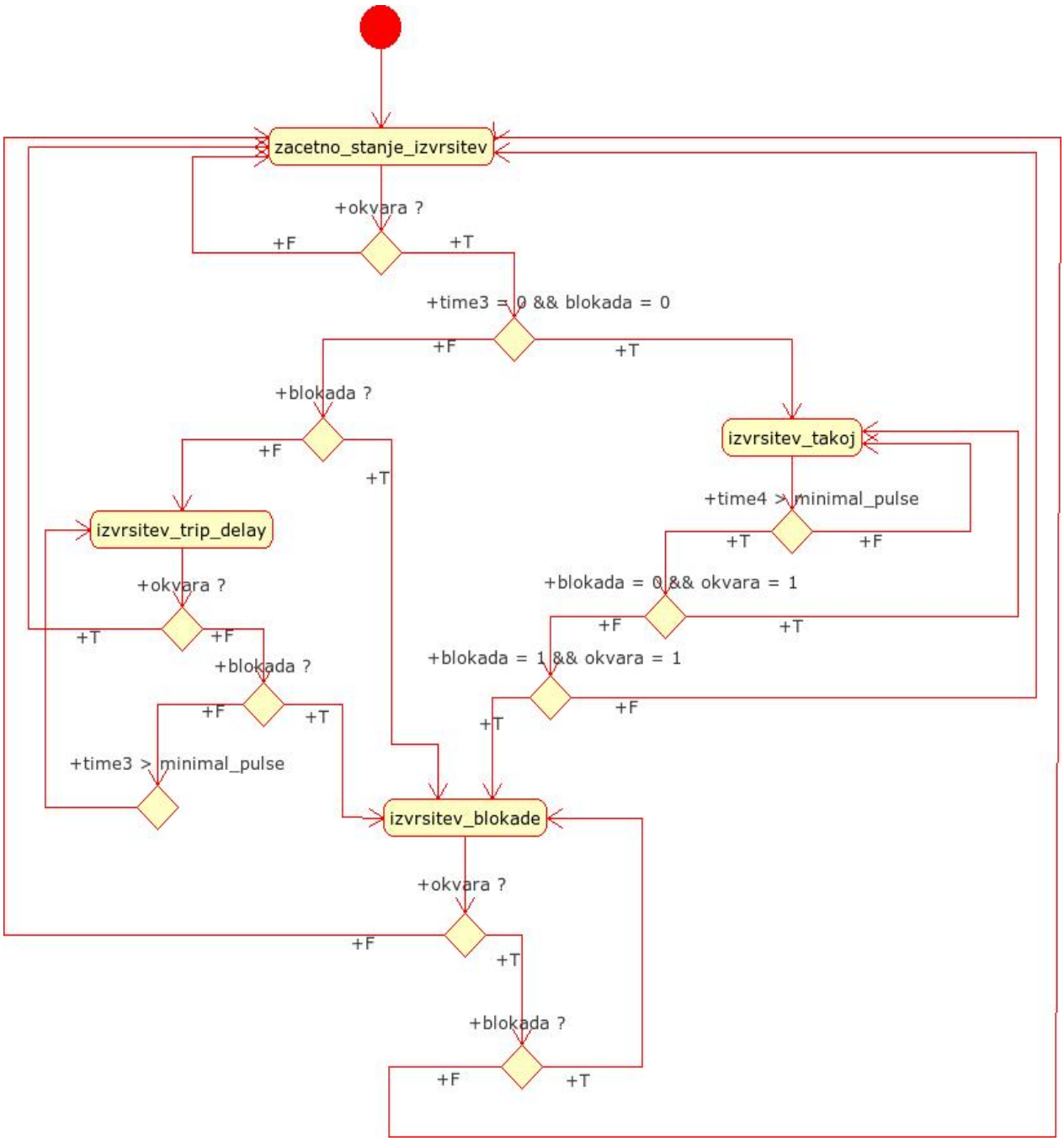
Ta proces nam služi, da ob določenih pogojih postavimo izhodni signal *trip* na 1, kar v praksi pomeni, da damo zeleno luč za aktivacijo zaščite. Na podlagi signala *trip* se aktivirana zaščita izklopi iz elektroenergetskega omrežja.

V prvem stanju smo vedno, kadar ni zaznana okvara, ki se identificira v procesu iskanja okvare. Če smo ob določeni urini periodi v nekem drugem stanju in se v tem času spremeni okvara, se takoj vrnemo v začetno stanje procesa. V tem stanju ni postavljen noben izhodni signal. Ob prisotnosti okvare se v začetnem stanju na podlagi ostalih vhodnih pogojev odločamo v katero izmed treh ostalih stanj bomo šli.

V primeru, da imamo blokado *pickup_block* in *okvaro* hkrati, je naslednje stanje *izvrsitev_blokade*. V tem stanju postavimo samo izhodni signal *pickup_blocked*. V tem stanju se nahajamo toliko časa, dokler je postavljena blokada *pickup_block* in prisotna *okvara*. V primeru, da *okvare* ni več, se vrnemo v začetno stanje tega procesa. V primeru, da je *okvara* še vedno prisotna ni pa več prisotna blokada *pickup_block*, se zopet vrnemo v začetno stanje, z razliko od prej pa tokrat dodatno postavimo notranji signal *reset_blokade*. Ta signal se preverja v procesu iskanja okvare v stanjih »*hold_okvara*« in »*run_dd_time*«. Z njim povzročimo, da znova preverjamo ali je prisotna *okvara* v procesu iskanje okvare (postavimo se v stanje *run_pd_time*).

Če imamo nastavljen čas *trip_delay* in ni prisotne blokade *pickup_block* gremo v stanje *izvrsitev_trip_delay*. V tem stanju smo toliko časa, dokler ne poteče čas *trip_delay* (seveda ob prisotnosti okvare) razen, če pride do signala za blokado. Takrat gremo v stanje za izvršitev blokade. Ko pa čas poteče, se naslednje stanje prestavi v *izvrsitev_takoj*. V tem stanju imamo postavljen le signal *pickup*.

Zadnje stanje in edino stanje v katerem se postavi »zelena luč« za signal *trip*, je *izvrsitev_takoj*. V tem stanju smo minimalno toliko časa, kolikor je dolg časovnik za *minimal_pulse*. V tem času se postavijo vsi izhodni signali: *trip*, *pickup* in *pickup_blocked* (če je postavljen vhodni *pickup_block*). Postavljeni so minimalni časi ne glede na to, če v tem času pade *okvara* ali se pojavi blokada *pickup_block*. Če je po preteku časa prisotna blokada, gremo v stanje za izvršitev blokade, če ni *okvare*, v začetno stanje in če ni zadržkov, da ne bi ostali v istem stanju držimo postavljena signala *trip* in *pickup*.



Slika 22: Avtomat za izvršitev zaščite

4 SIMULACIJA IN REZULTATI

S pomočjo orodja Active-HDL, s katerim sem kodiral prenapetostni zaščitni algoritem, sem ga tudi testiral in simuliral. Orodje tudi izrisuje poteke signalov (ang. waveform), ki nam ponazorijo vrednosti vseh vhodnih in izhodnih kakor tudi notranjih signalov ob določenem času. S simulacijo sem skušal zajeti čim več možnosti, v katerih se lahko znajdemo ob nekem času. Slika poteka signalov nam bo tudi predstavila rezultate, ki jih bomo lahko primerjali z želenimi slikami oziroma grafi, ki smo si jih zastavili v opisu delovanja. Z simulacijo bom skušal dokazati pravilnost delovanja algoritma.

Pri simulaciji bom uporabljal uro s frekvenco 100Mhz. Za časa časovnikov (*pickupdelay*, *tripdelay*, …) bom privzel, da je ena urina perioda enaka eni enoti zakasnitve danega časovnika. Za vrednost časovnikov bom uporabljal nizke številke. S tem bom dosegel bolj pregledno sliko in bolj jasne rezultate.

Nastavljene vrednosti prevzamemo kot privzete in fiksne. Med samo simulacijo se ne bodo spreminjale. Nastavljive oziroma variabilne bodo vrednosti, ki se bodo med delovanjem simulacije spreminjale. Simulirali bomo tako, da jih bomo spreminjali ob vnaprej določenem času (orodje Active-HDL omogoča to možnost s formulo).

4.1 Simulacije

1. **Simulacija: Slika 23**

   Nastavljene vrednosti:
   - *pickup_block*(blokada): 0
   - *clock*: 100Mhz (10ns je ena urina perioda)
   - *pickupvalue*: 120
   - *dropoutvalue*: 80
   - *pickupdelay*: 3
   - *dropoutdelay*: 3
   - *tripdelay*: 0
   - *minimal_pulse*: 4

   Nastavljive vrednosti:
   - *amplituda*: 125 30 ns, 65 50 ns,150 80 ns, 40 150 ns

Na sliki simulacije se opazuje, kako se spreminja amplituda. Rezultat spreminjanja je v signalu *trip*. Za dobro sledenje opazujmo še *okvaro*, *stanje*, *stanje_izvršitev* in pa *time1d* (števec za *pickupdelay*) in *time2d* (števec za *dropoutdelay*).

Kot lahko razberemo, je začetna *amplituda* nedefinirana. Po 30ns ali 3 urinih periodah vrednost *amplitude* naraste na 125. Vrednost *amplitude* je večja od dovoljene – *pickupvalue*.

Sproži se časovnik *time1d*. Okvara se aktivira, ko je čas *time1d* večji od *pickupdelaya*. Po 5 urinih periodah vrednost amplitude pade na 65. Pogoj za okvaro je padel (Slika 13 nam to tudi prikazuje). Po 8 periodah (80ns) znova naraste vrednost *amplitude*, ki je večja od vrednosti *pickupvalue*. Tokrat se vrednost obdrži dlje časa kot pa traja *pickupdelay*. Po preteku *pickupdelaya* se zazna okvara, ki takoj sproži signala *pickup* in *trip*. Zaščita je aktivirana. Pri petnajsti periodi *amplituda* pade na 40ns. Sproži se začetek časovnika *time2d*, ki se primerja z časom *dropoutdelaya*. Dokler *time2d* ne preteče *dropoutdelaya*, so vsi trije signali postavljeni na 1. Če sedaj še pogledamo stanji signala avtomata: stanje in stanje_izvršitev. Stanji se spreminjata ob prehodu ure iz 0 v 1. Najbolj zanimivo je, da se signala *trip* in *pickup* postavita na 1, eno urino periodo preden smo v stanju *izvršitev_takoj*. To je zato ker, ko je *okvara* zaznana in obstajajo pogoji za vklop zaščite, postavimo signal avtomata za naslednjo izvršitev, ki pa se izvede ob naslednji urini periodi. Da pa se ne čaka na ta prehod, takoj vklopimo zaščito in nadaljujemo njeno izvajanje v naslednjem stanju avtomata. Podobno se zgodi, ko ni več potrebe po zaščiti in se ta ena urina perioda izvede preden se prestavimo v začetno stanje. To velja za vse naslednje simulacije.

2. Simulacija: Slika 24

Nastavljene vrednosti:
- *pickup_block*(blokada): 0
- *clock*: 100Mhz (10ns je ena urina perioda)
- *pickupvalue*: 120
- *dropoutvalue*: 80
- *pickupdelay*: 3
- *dropoutdelay*: 3
- *tripdelay*: 0
- *minimal_pulse*: 10

Nastavljive vrednosti:
- *amplituda*: 125 30 ns, 65 80 ns

Slika druge simulacije (Slika 24) nam prikazuje, kaj se zgodi, če *okvara* ni več prisotna, postavljena sta pa še vedno signala *pickup* in *trip*. To se pa zgodi zaradi časovnika *minimal_pulse*. V praksi se to uporablja zaradi zakasnitev na odklopnikih, ločilnikih, …

Po 3 periodah *amplitude* se začne preverjanje zaznavanja okvare, ki se po pretečenem času *time1d* glede na *pickupdelay* tudi zazna. Okvara je prisotna 5 period potem pa zaradi pretečenega časa *time2d* nič več. Vendar čas *time4d*, ki se meri proti časovniku *minimal_pulse* še ni potekel. Zaradi tega pogoja morata biti signala *trip* in *pickup* postavljena najmanj še 5 period. Potem pa padeta, ker ni več prisotne *okvare*.

3. Simulacija: Slika 25

Nastavljene vrednosti:
- *clock*: 100Mhz (10ns je ena urina perioda)
- *pickupvalue*: 120
- *dropoutvalue*: 80
- *pickupdelay*: 3

dropoutdelay: 3
tripdelay: 0
minimal_pulse: 2

Nastavljive vrednosti:

amplituda: 125 30 ns, 65 110 ns
pickup_block(blokada): 1 60 ns, 0 170 ns

Tretjo simulacijo smo zasnovali tako, da smo prikazali kaj se zgodi, če je prižgana blokada za signal *trip*. Opazujmo signale *pickup_block*, *trip*, *okvara* in *pickup_blocked*. V praksi se takšen primer pričakuje, ko priklapljamo kakšen vod na omrežje in vemo, da bo prišlo do povečane vrednosti amplitude, vendar se zavedamo, da je to kratkotrajno in ker nadzorujemo potek si ne želimo, da bi se vklopila zaščita. Zato pred tem vklopimo blokado, katero tudi izklopimo po določenem času.

Slika 25 prikazuje vklop signala *pickup_block* v 6. periodi in njegov izklop v 17. periodi. V tretji periodi pride do povečane *amplitude*, ki po preteku *pickupdelay-a* povzroči *okvaro*. Vklopit se morata signala *pickup* in *trip*. Namesto njiju se vklopi signal *pickup_blocked*, ki nam pove, da bi v tem času mogel biti vklopljen signal *trip* in posledično signal *pickup*.

4. Simulacija: Slika 26

Nastavljene vrednosti:

clock: 100Mhz (10ns je ena urina perioda)
pickupvalue: 120
dropoutvalue: 80
pickupdelay: 3
dropoutdelay: 3
tripdelay: 1
minimal_pulse: 2

Nastavljive vrednosti:

amplituda: 125 30 ns, 65 110 ns
pickup_block(blokada): 1 60 ns, 0 170 ns

Četrta simulacija je skoraj podobna kot tretja. Razlika je le, da ima nastavljen čas. Zakasnilni *tripdelay* je čas za zakasnitev signal *trip*. Pogosto je ta čas kar nič. V redkih primerih pa želimo zakasnitev vklopa zaščite. Ta čas je zelo kratek.

Medtem, ko se zazna *okvara*, se zraven sproži še preverjanje *tripdelay-a*. Če je ta čas nastavljen, mora signal *trip* počakati toliko časa, kolikor traja *tripdalay*, da se lahko postavi. Ker pa imamo v tem primeru vklopljeno blokado *pickup_block*, se namesto signala *trip* vklopi signal *pickup_blocked*.

5. Simulacija: Slika 27

Nastavljene vrednosti:

clock: 100Mhz (10ns je ena urina perioda)

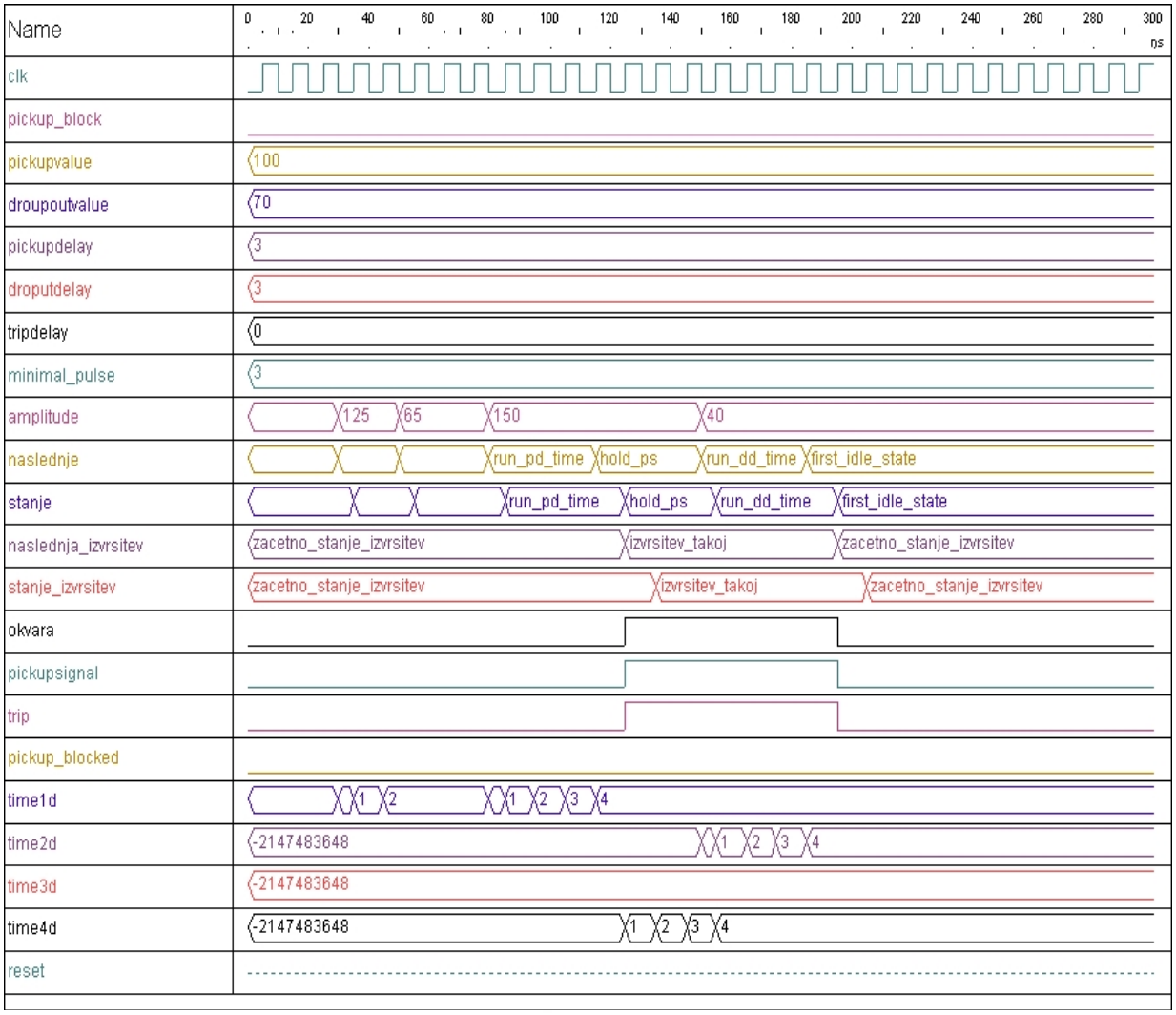
pickupvalue: 120
dropoutvalue: 80
pickupdelay: 3
dropoutdelay: 3
tripdelay: 1
minimal_pulse: 2

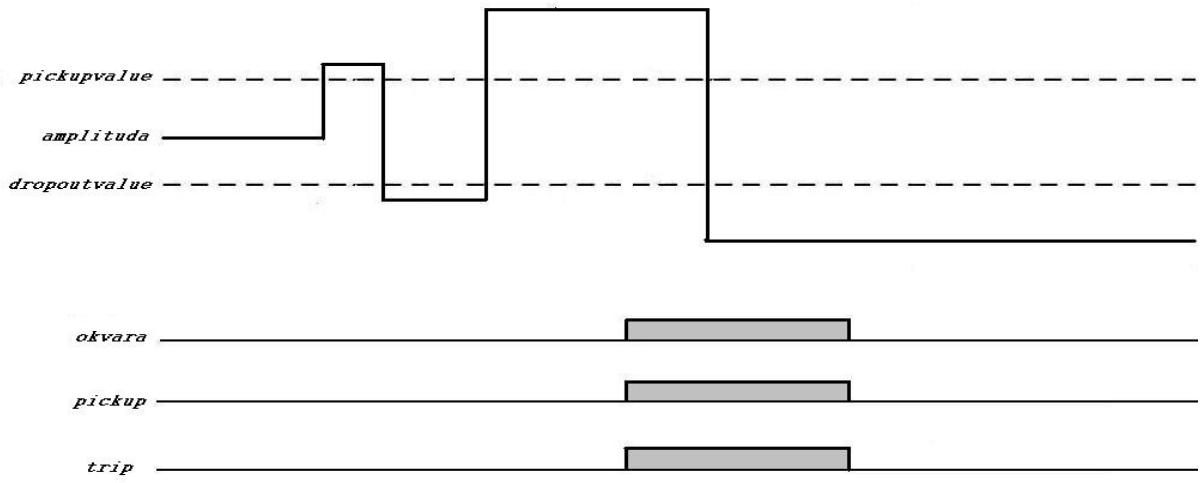
Nastavljive vrednosti:

amplituda: 125 20 ns, 65 220 ns
pickup_block(blokada): 1 85 ns, 0 120 ns

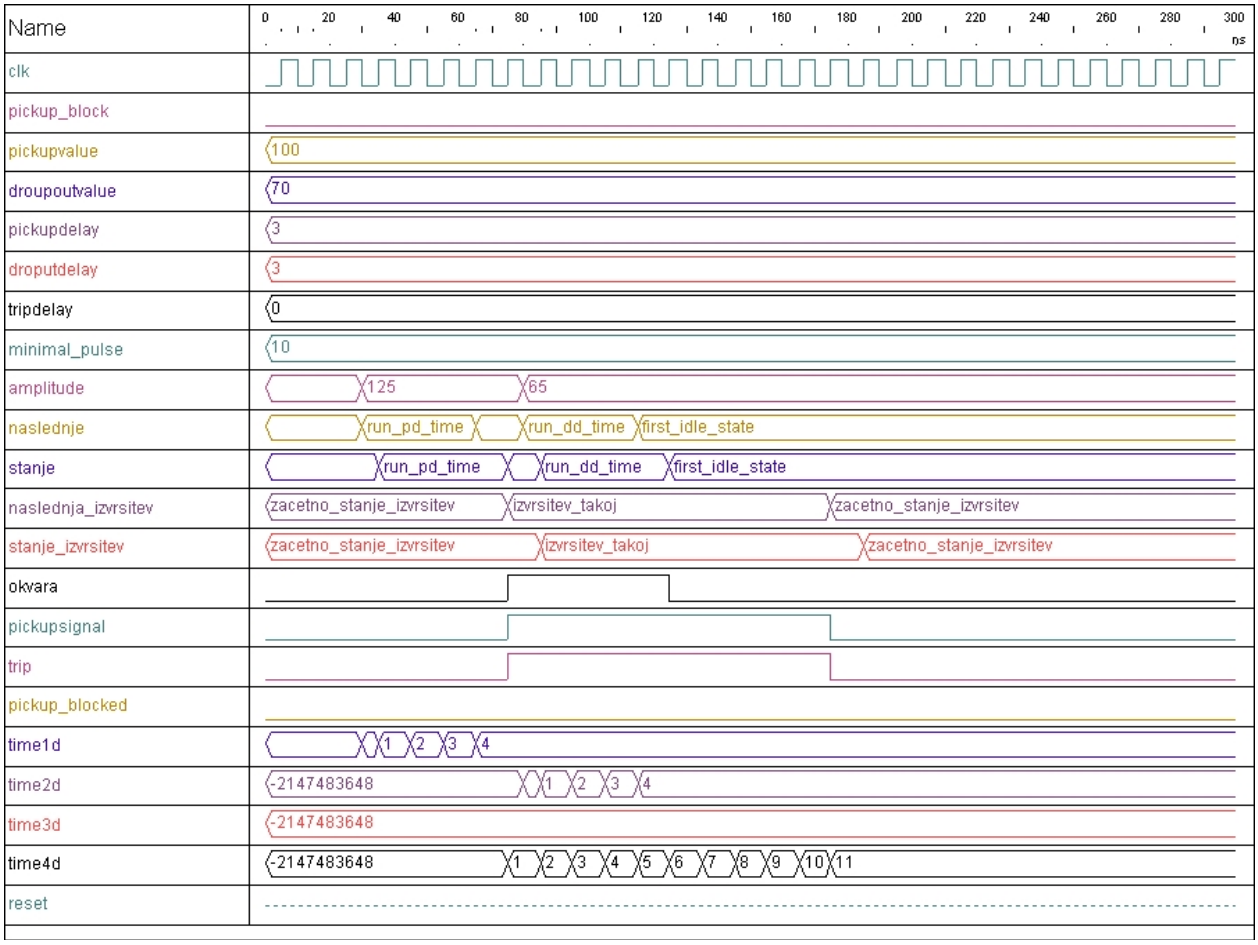
Pri peti simulaciji sem podaljšal čas nastopa povečane vrednosti *amplitude* na 20 period. Pri tem sem dodal možnost, da se blokada *pickup_block* vklopi med izvajanjem. Pri opazovanju slike bodimo pozorni na naslednje signale: *pickup_block*, *okvara*, *trip*, *pickup*, *pickup_blocked* in stanje končnega avtomata. Na sliki najprej opazim, da se signala *pickup_blocked* in *trip* prekrivata. S tem ni nič narobe. To smo si tudi želeli. Zgodilo se je pa ob naslednjih pogojih oziroma dogodkih.

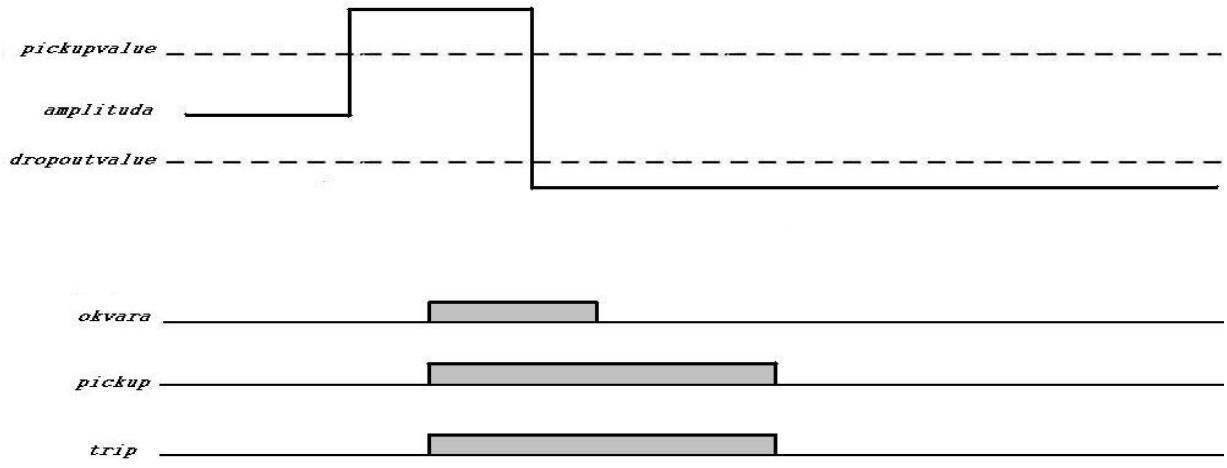
V drugi periodi, smo dobili preveč povečano *amplitudio* glede na dovoljeno. Ta je prevelika celih 20 period. Po preteku *pickupdelay*-a se zazna *okvara*. Za njo moramo upoštevati še zakasnitev *trip delaya*. Po preteku le-tega se vklopi signal *trip*. Do tega časa blokada *pickup_block* še ni vklopljena. V 85. nanosekundi se vklopi blokada *pickup_block*, ker je takrat tudi sprememba urinega prehoda iz 0 v 1 se ta tudi takoj upošteva. Sedaj bi se moral signal *trip* izklopiti in vklopiti *pickup_blocked*, vendar se ne. To pa zato, ker še ni potekel čas za časovnik *minimal_pulse*. To je čas od začetka signala *trip* in traja do konca nastavljene vrednosti. Ko smo v tem stanju izvrševanje *tripdelaya*, mora biti signal *trip* postavljen ne glede na prisotnosti blokade *pickup_block* ali *okvare*. Slika 27 to tudi prikazuje, ko sta signala *trip* in *pickup_blocked* postavljena istočasno. Po preteku časa *tripdelay*, se signal *trip* postavi na nič, namesto njega se pa za tisti čas, ko bi moral biti postavljen postavi *pickup_blocked*. Ta signal je postavljen, dokler sta prisotna signala *pickup_block* in *okvara*. Ker *pickup_block* pade in je zahteva algoritma takšna, da se ponovno preverja stanje za *okvaro*, se ponovno preverja čas *pickupdelay* in ali je *okvara* še vedno prisotna. Po tem, ko zopet zaznamo *okvaro*, *tripdelay* poteče in blokada ni vklopljena (*pickup_block*), se signal *trip* postavi. Zaščita je aktivirana. Zopet je zaradi *tripdelay*-a signal *pickup* postavljen pred signalom *trip*.



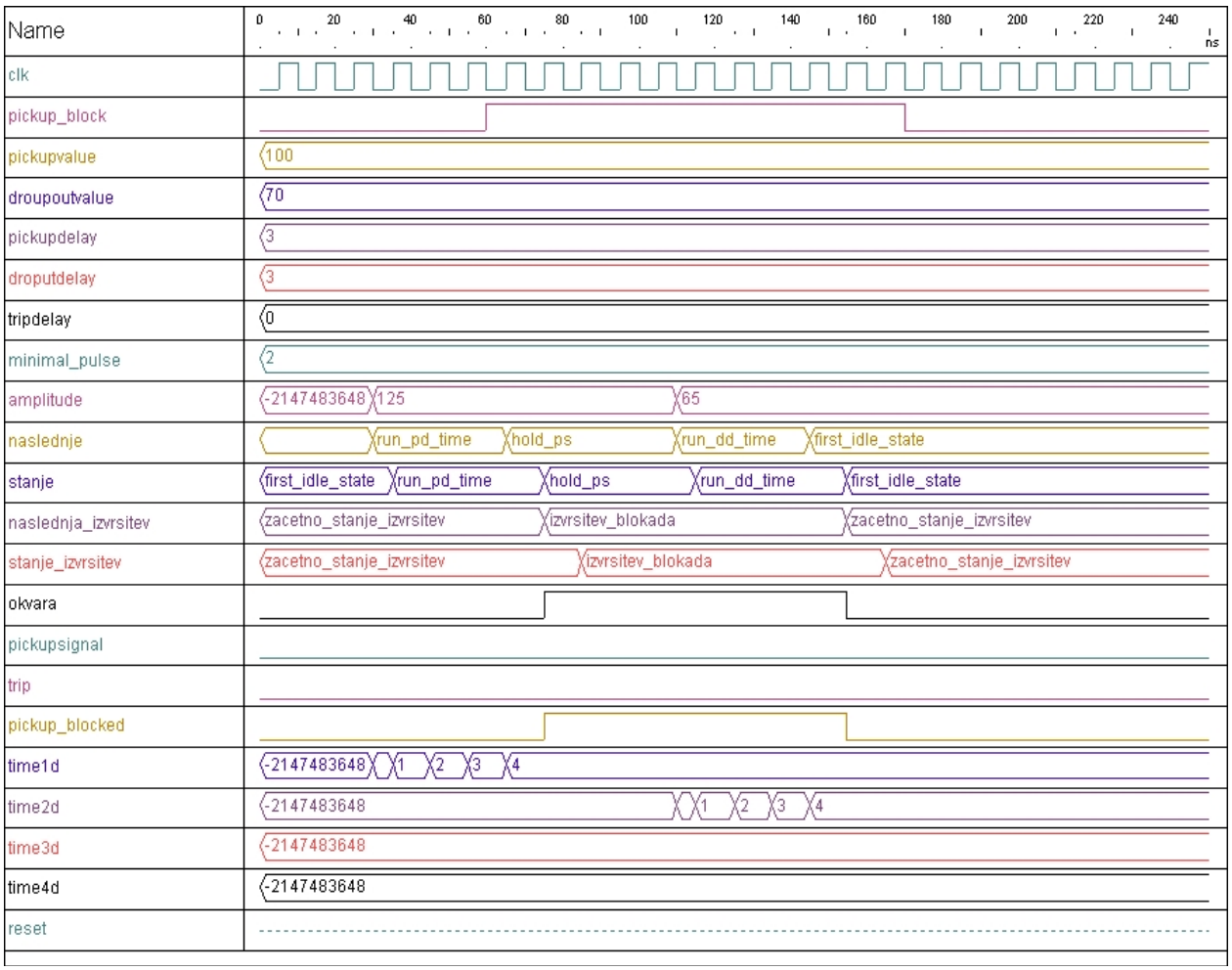


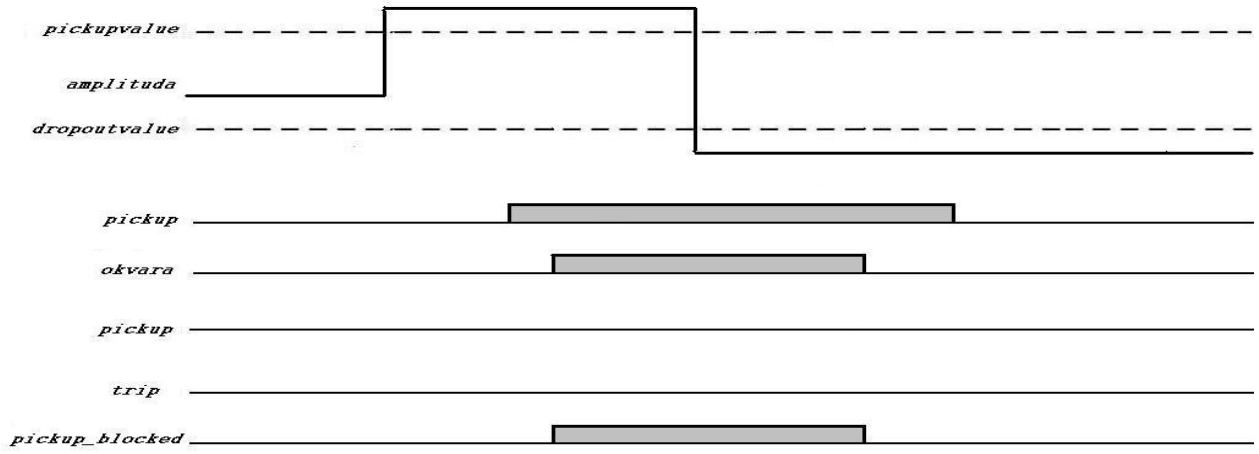
Slika 23: Simulacija 1 in graf



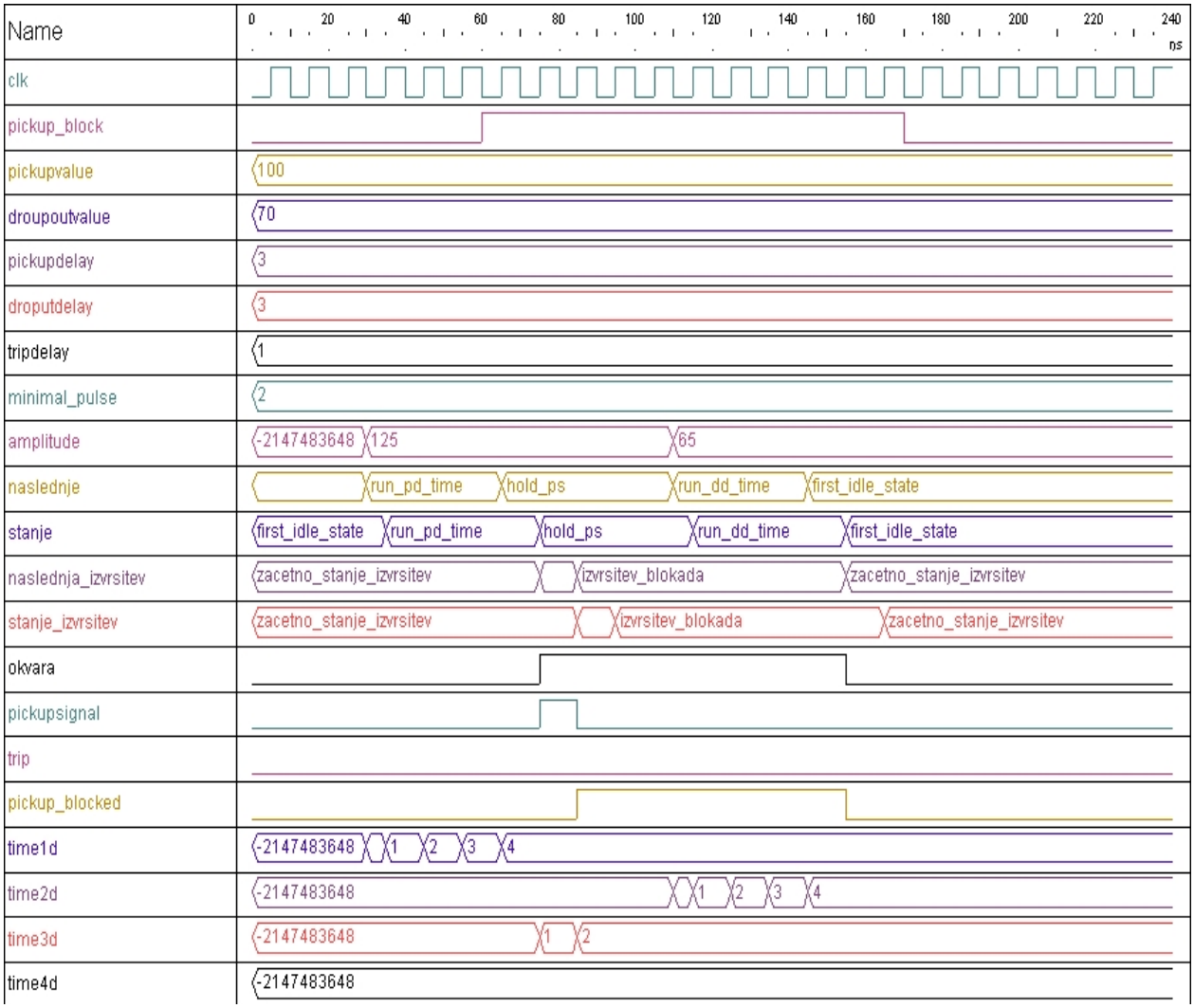


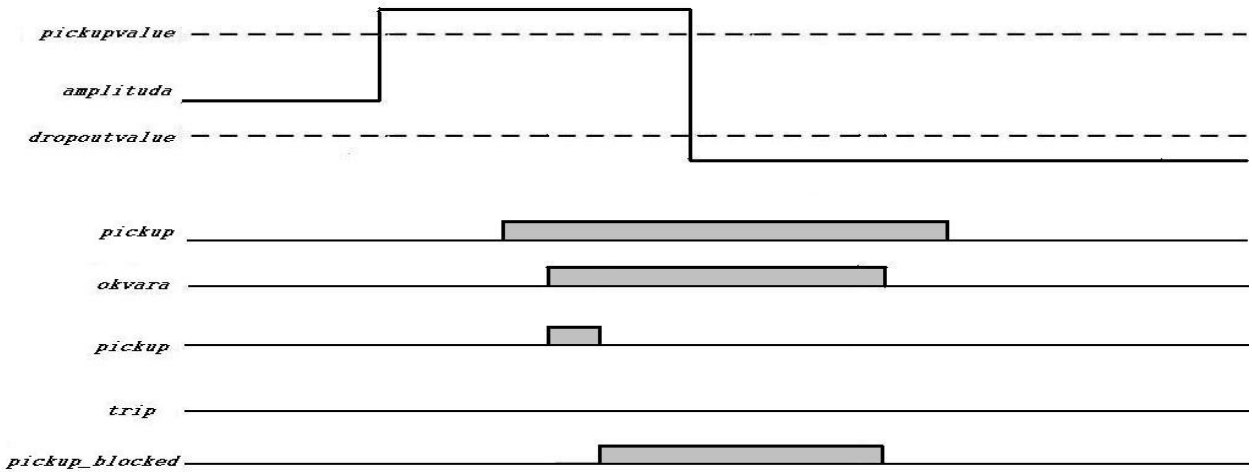
Slika 24: Simulacija 2 in graf



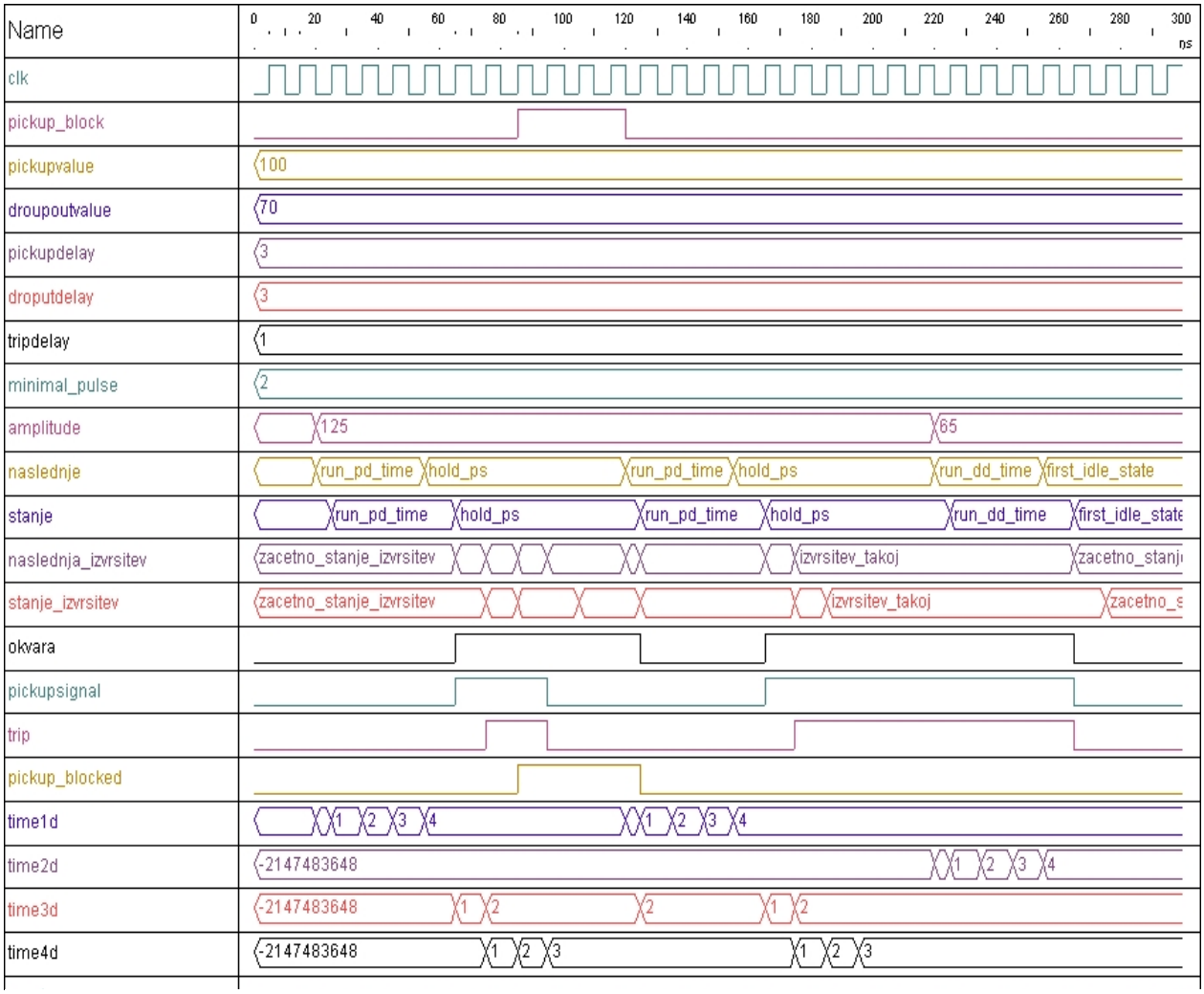


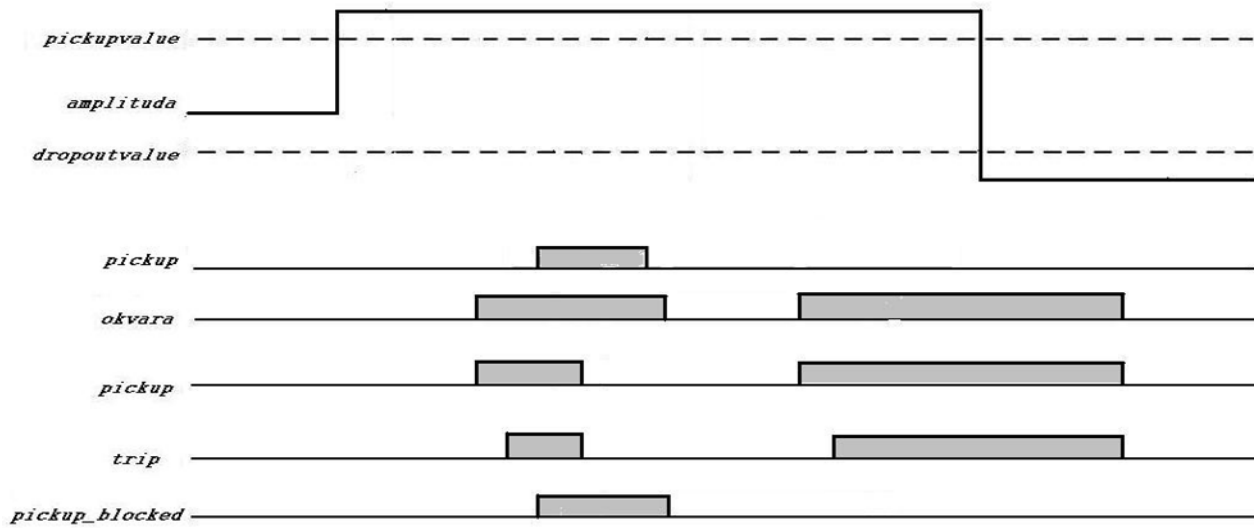
Slika 25: Simulacija 3 in graf (prikaz blokade)





Slika 26: Simulacija 4 in graf





Slika 27: Simulacija 5 in graf

5 ZAKLJUČEK

Ideja za implementacijo zaščitnega prenapetostnega algoritma v jeziku VHDL se je rodila v podjetju Iskra Sistemi, kjer sem najprej opravljal praktično usposabljanje, potem pa sem še nadaljeval sodelovanje v razvojnem oddelku kot študent – programer. V Iskri se med drugim ukvarjajo tudi z zaščitami v energetiki. Z lastnim razvojem, ki je prisoten že iz sedemdesetih (takrat še pod blagovno znamko Iskra Sysen), razvijajo zaščitne releje od mehanskih in sedaj do numeričnih. Aktualen produkt, ki sloni na numerični zaščiti, ima oznako FPC. FPC je naprava za zaščito, nadzor in krmiljenje SN (srednje napetostnih) vodov. Naprava že ima obstoječe FPGA vezje, pri zajemanju in obdelavi podatkov. Želja je bila, da bi predstavil implementacijo enega izmed zaščitnih algoritmov v jeziku VHDL za vezje FPGA, ki se bi ga lahko uporabilo na FPC produktu. Ta zaščitni algoritem bi se zamenjalo s tistim, ki je že realiziran na mikroprocesorju.

Da bi realiziral takšno idejo, sem moral najprej pridobiti splošno znanje o zaščitah, zaščitnih relejih, kje se uporabljajo in na kakšen način. S tem nisem dobil samo vpogleda, kje se to uporablja, ampak tudi, na kakšen način mora biti zaščitni algoritem realiziran, kakšne so njegove zahteve za delovanje, … Še preden sem se lotil naloge, me je zanimalo, ali že obstaja kakšna podobna rešitev implementacije zaščitnih algoritmov v VHDL. Do kakšni bistvenih rešitev nisem prišel. Neka ideja o implementaciji prenapetostnega algoritma se je sicer pojavila na konferenci PES (power energy society) združenja IEEE (Institute of Electrical and Electronics Engineers) [14]. Ampak kaj več kot ideje nisem našel. Mogoče to ni toliko aktualno, ker se sedaj za zaščito vodov, generatorjev… usmerja bolj v uporabo umetne inteligence.

Preden sem se lotil kodiranja sem se poleg splošnega znanja o relejih moral seznaniti in raziskati, kaj sploh je prenapetostni algoritem, kakšne so njegove zahteve in kako deluje. Znanje o tem, kakšen način in kako deluje, sem pridobil v podjetju Iskra Sistemi. Pomagal sem si tudi s podatki delovanja Siemensovih in ABB-jevih naprav (ABB je vodilno podjetje na svetu pri zaščiti sekundarnega EES). Potem je sledilo spoznavanje in učenje jezika VHDL in orodja Active-HDL. Nekaj začetnih osnov sem pridobil že na fakulteti med študijem. Spoznal sem, da je VHDL nekoliko drugačen od ostalih jezikov, kot so na primer Java ali C/C++.

Pred pisanjem kode sem si naredil nekakšen osnutek oziroma načrt, kako bom izvedel kodiranje. Tako sem imel začrtano pot do cilja. Šlo je brez večjih težav. Te so se začele pojavljati, ko sem začel testirati in simulirati algoritem. Na začetku se je pojavila tista najbolj neumna, ker sem vzel za primerjanje prevelike vrednosti. Vendar te niso toliko vplivale na delovanje algoritma, kot pa na porabo časa pri testiranju. Zanimivo je bilo bolj pri takšni, ko se postavitve signalov niso najbolj ujemale in so imeli med seboj razliko postavitve na primer v urini periodi. Razlog za takšno napako je bil bolj zaradi bodisi prehitrega ali prepoznega postavljanja signalov. Z malo »kozmetike« v kodi se je to hitro našlo in odpravilo. Bolj zanimivo je bilo takrat, ko sem že kar nekaj možnosti obnašanja algoritma stestiral in prišel do napačnega želenega rezultata. Tam se je porodilo vprašanje ali sem prav zastavil načrt algoritma, kako odpraviti napako in na kaj bo vse to vplivalo. Takrat sem prišel do spoznanja, da sem naletel na robni pogoj, ki ga nisem upošteval. Spregledal sem pogoj, ga nisem

upošteval in šel prehitro v izvajanje. Rezultat - napačen odziv (to je bilo pri signalu *pickup*, ko sem testiral oziroma simuliral simulaciji 4 in 5).

Osnovne faze projekta so: analiza, načrtovanje, kodiranje, testiranje in vzdrževanje. Nekje sem slišal, da podjetje IBM posveča 40% časa, ki ga ima na voljo za določen projekt, analizi, načrtovanju in kodiranju, 40% testiranju in 20% časa za vzdrževanje oziroma pisanje uporabniške dokumentacije. Če po tem skušam povzeti mojo diplomsko nalogo, pridem do podobnega zaključka z malce odstopanja. Za analizo orodja in jezika nisem porabil skoraj nič časa, saj je bila izbira jasna. Načrtovanje mi je vzelo že več časa. Skice na papir in zmečkani papirji v košu so postali praksa. Kodiranje je šlo zelo hitro (seveda po prebrani knjigi o VHDL-u in nekaj predelanih internetnih učnih ur (ang.tutorial)), saj sem osnutke in načrt dobesedno spreminjal v kodo. Testiranje je vzelo tudi kar nekaj časa. Nekajkrat sem se moral vrniti nazaj na kodiranje in popraviti kakšno malenkost, kot rečeno enkrat tudi večjo. Po resnici povedano mi je pa (uporabniška) dokumentacija vzela največ časa. To je bila diploma. Kar veliko časa mi je vzelo pridobivanje, raziskovanje zaščit in predelovanje, izdelovanje in izvedba v logično celoto za diplomsko delo. Posebej zato, ker sem moral začetno znanje, kako se projekt izdela, šele pridobiti. Če bi moj porabljen čas diplomske naloge razdelil po prej omenjenih procentih, pridem do spoznanja, da je pri meni bilo približno tako: 30%:30%:40%. Mogoče za prvič tudi ne tako slabo.

6 LITERATURA

[1] A. Trost: Načrtovanje digitalni vezij v jeziku VHDL, Fakulteta za elektrotehniko, Ljubljana 2007

[2] A. Ogorelec, F. Gubina, »Protection: From relays to computers,« v zborniku 10th International Conference on Power System Protection PSP'96, Bled, Slovenia, 7. - 9. October 1996

[3] B. Grčar, Zaščita v EES - Skripta predavanj, dostopno na http://www.labie.uni-mb.si/un_pred_op.htm#uni_zasc_ESS, marec 2009

[4] A. Ogorelec, Relejni zaščitni sistemi, Fakulteta za elektrotehniko, Ljubljana 1977

[5] A. Ogorelec, Nove smeri v zaščiti in lokalni avtomatizaciji, Fakulteta za elektrotehniko, Ljubljana 1986

[6] Iskra Sistemi d.d., dostopno na: http:/www.iskrasistemi.si/, marec 2009

[7] Numerični releji. Dostopno na: http://pdf.directindustry.com/pdf/siemens-power-transmission-distribution/siprotec-numerical-protection-relays-siemens-30064-19268.html, marec 2009

[8] ABB Group, dostopno na: http://www.abb.com/ProductGuide/Alphabetical.aspx, marec 2009

[9] A. Ogorelec, Skripta predavanj 12.11.2002, dostopno na http://lees.fe.uni-lj.si/www/staticAdminMgr.php?action=read&menu=1128596748, marec 2009

[10] Relay, dostopno na: http://en.wikipedia.org/wiki/Relay, marec 2009

[11] Distančna zaščita, dostopno na: http://sl.wikipedia.org/wiki/Distan%C4%8Dna_za%C5%A1%C4%8Dita, marec 2009

[12] Overvoltage, dostopno na: http://en.wikipedia.org/wiki/Overvoltage, marec

[13] Electro mechanical relay, dostopno na: http://pinpointcncblog.com/, marec 2009

[14] The implementation of digital protection in power system using FPGA, dostopno na: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=982603, marec 2009

7 PRILOGA

VHDL koda:

```
--------------------------------------------------------------------------------
-
--Title                       :     Degree task
--Design                      :     Protection algorithm
--Author                      :     Ales Tkalcic,     ales.tkalcic@gmail.com
       +38640827572
--Counsellor, Mentor    :     Branko Ster
--Company                     :     /
--------------------------------------------------------------------------------
-
--Description                 :     Overvoltage algorithm
--------------------------------------------------------------------------------
-

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity avtomat is
    port (  clk:              in std_logic;
            reset:            in std_logic;
            pickupvalue:      in integer := 100;
            droupoutvalue:    in integer := 70;
            pickupdelay:      in integer := 3;
            droputdelay:      in integer := 3;
            amplitude:        in integer;
            pickupsignal:     out bit;
            pickup_blocked:   out bit;
            trip:             out bit;
            pickup_block:     in bit := '0';
            tripdelay:        in integer := 0;
            minimal_pulse:    in integer := 2;
            alarm:            out bit := '0'                  );
end avtomat;

architecture RTLopis of avtomat is
type vsa_stanja_okvara is (first_idle_state, run_pd_time, hold_ps,
run_dd_time);

            signal stanje, naslednje: vsa_stanja_okvara;


type vsa_stanja_izvrsitev is (zacetno_stanje_izvrsitev, izvrsitev_blokada,
izvrsitev_trip_delay, izvrsitev_takoj );
            signal stanje_izvrsitev, naslednja_izvrsitev:
vsa_stanja_izvrsitev;

             signal time1: integer;
             signal time2: integer;
             signal time1d: integer;
             signal time2d: integer;
             signal time3: integer;
```

```
             signal time4: integer;
             signal time3d: integer;
             signal time4d: integer;
             signal okvara: bit;
             signal reset_okvara: bit;
             signal error_1: bit;
             signal error_2: bit;

begin

URA_IN_RESET: process (clk, reset)
begin
      if reset = '1' then
            stanje <= first_idle_state;
            stanje_izvrsitev <= zacetno_stanje_izvrsitev;
            time1 <= 0;
            time2 <= 0;
            time3 <= 0;
            time4 <= 0;
      elsif clk'event and clk='1' then
            stanje <= naslednje;
            stanje_izvrsitev <= naslednja_izvrsitev;
            time1 <= time1d;
            time2 <= time2d;
            time3 <= time3d;
            time4 <= time4d;
            if error_1 = '1' and error_2 = '1' then
                  alarm <= '1';
            end if;
    end if;
 end process;


 ISKANJE_OKVARE: process (stanje, naslednje, amplitude,reset_okvara, time1,
time2)

begin

      time1d <= time1;
      time2d <= time2;

         case stanje is
           when first_idle_state =>
                  okvara <= '0';
                        if pickupvalue < amplitude then
                              naslednje <= run_pd_time;
                              time1d <= 0;
                    else
                       naslednje <= first_idle_state;
                    end if;

           when run_pd_time =>
                   okvara <= '0';
                   time1d <= time1 + 1;--ali lahko tako
                    if  time1 < pickupdelay then

                              if pickupvalue <= amplitude then
                                    naslednje <= run_pd_time;
                              else
                                    naslednje <= first_idle_state;
                                          end if;
```

```
                    else
                        if pickupvalue < amplitude then
                                    naslednje <= hold_ps;
                              else
                                    naslednje <= first_idle_state;
                                    end if;
                    end if;


               when hold_ps =>
                   okvara <= '1';
                        if reset_okvara = '1' then
                              time1d <= 0;
                              naslednje <= run_pd_time;

                        elsif droupoutvalue > amplitude then
                                    naslednje <= run_dd_time;
                              time2d <= 0; --time1 <= pickupdelay;--
start_timer

                        else
                              naslednje <= hold_ps;

                    end if;



           when run_dd_time =>

                   okvara <= '1';
                   time2d <= time2 + 1;

                        if reset_okvara = '1' then
                              time1d <= 0;
                              naslednje <= run_pd_time;

                        elsif  time2 < droputdelay then

                              if droupoutvalue >= amplitude then
                                    naslednje <= run_dd_time;

                              else
                                    naslednje <= hold_ps;
                              end if;

                        else
                                    if droupoutvalue >= amplitude then
                                                naslednje <=
first_idle_state;
                                    else
                                          naslednje <= hold_ps;
                                                end if;

                    end if;


           when others =>
                   error_1 <= '1';
                   naslednje <= first_idle_state;
```

```
        end case;

 end process;


IZVRSITEV_ZASCITE: process (stanje_izvrsitev, naslednja_izvrsitev,time3,
time4, pickup_block, tripdelay, minimal_pulse, okvara, time3d,
time4d,amplitude, time1d, time2d, time1, time2, stanje, naslednje)
begin
      time3d <= time3;
      time4d <= time4;

       case stanje_izvrsitev is
           when zacetno_stanje_izvrsitev =>
                  reset_okvara <= '0';

                  if okvara = '1' then
                        if tripdelay = 0 and pickup_block = '0' then
                              pickupsignal <= '1';
                              trip <= '1';
                              pickup_blocked <= pickup_block;
                              time4d <= 1; --zazenemo minimal_pulse casovnik
                              naslednja_izvrsitev <= izvrsitev_takoj;

                        elsif tripdelay > 0 and pickup_block = '1' then
                              pickupsignal <= '1';
                              trip <= '0';
                              pickup_blocked <= '0';
                              time3d <= 1; --zazenemo minimal_pulse casovnik
                              naslednja_izvrsitev <= izvrsitev_trip_delay;


                        elsif pickup_block = '0' then
                              pickupsignal <= '1';
                              trip <= '0';
                              pickup_blocked <= pickup_block;
                              time3d <= 1;
                              naslednja_izvrsitev <= izvrsitev_trip_delay;


                        else
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= pickup_block;
                              naslednja_izvrsitev <= izvrsitev_blokada;
                        end if;

                  else
                        pickupsignal <= '0';
                        trip <= '0';
                        pickup_blocked <= '0';
                        naslednja_izvrsitev <= zacetno_stanje_izvrsitev;
                  end if;



            when izvrsitev_takoj =>
                  time4d <= time4 + 1;
                  if time4 >= minimal_pulse then
                        time4d <= minimal_pulse+1;
```

```
                        if pickup_block = '0' and okvara = '1' then
                              pickupsignal <= '1';
                              trip <= '1';
                              pickup_blocked <= pickup_block;
                              naslednja_izvrsitev <= izvrsitev_takoj;

                        elsif pickup_block = '1' and okvara = '1' then --
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= pickup_block;
                              naslednja_izvrsitev <= izvrsitev_blokada;

                        else
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= '0';
                              naslednja_izvrsitev <=
zacetno_stanje_izvrsitev;
                        end if;

                  else--čas ni potekel
                        pickupsignal <= '1';
                        trip <= '1';
                        pickup_blocked <= pickup_block;
                        naslednja_izvrsitev <= izvrsitev_takoj;
                  end if;

            when izvrsitev_blokada =>
                  if okvara = '1' then
                        if pickup_block = '1' then
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= pickup_block;
                              naslednja_izvrsitev <= izvrsitev_blokada;
                        else
                              reset_okvara <= '1';
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= '1';
                              naslednja_izvrsitev <=
zacetno_stanje_izvrsitev;

                        end if;
                  else
                        pickupsignal <= '0';
                        trip <= '0';
                        pickup_blocked <= '0';
                        naslednja_izvrsitev <= zacetno_stanje_izvrsitev;
                  end if;

            when izvrsitev_trip_delay =>
                  time3d <= time3 + 1;

                  if okvara = '1' then
                        if pickup_block = '1' then
                              pickupsignal <= '0';
                              trip <= '0';
                              pickup_blocked <= pickup_block;

                              naslednja_izvrsitev <= izvrsitev_blokada;
```

```
                        elsif time3 < tripdelay then
                              pickupsignal <= '1';
                              trip <= '0';
                              pickup_blocked <= pickup_block;
                              naslednja_izvrsitev <= izvrsitev_trip_delay;
                        else
                              pickupsignal <= '1';
                              trip <= '1';
                              pickup_blocked <= '0';
                              time4d <= 1;
                              naslednja_izvrsitev <= izvrsitev_takoj;
                        end if;
                  else
                        pickupsignal <= '0';
                        trip <= '0';
                        pickup_blocked <= '0';
                        naslednja_izvrsitev <= zacetno_stanje_izvrsitev;
                  end if;
            when others =>
                   error_2 <= '1';
            end case;

 end process;
end RTLopis;
```