Univerzitet u Nišu
Elektronski fakultet

SANDRA M. ĐOŠIĆ

TOLERISANJE GREŠAKA I ENERGETSKA EFIKASNOST KOD SISTEMA ZA RAD U REALNOM VREMENU SA VREMENSKOM REDUNDANSOM

doktorska disertacija

Niš, 2015

University of Niš
Faculty of Electronic Engineering

SANDRA M. ĐOŠIĆ

FAULT TOLERANCE AND ENERGY EFFICIENCY OF REAL-TIME SYSTEMS WITH TIME REDUNDANCY

doctoral dissertation

Niš, 2015

Podaci o mentoru i članovima komisije

Mentor:

dr Goran Lj. Đorđević, redovni profesor, Elektronski fakultet Niš, Univerzitet u Nišu

Komisija za ocenu i odbranu doktorske disertacije:

1. dr Goran Lj. Đorđević, redovni profesor, Elektronski fakultet Niš, Univerzitet u Nišu
2. dr Milun Jevtić, redovni profesor, Elektronski fakultet Niš, Univerzitet u Nišu
3. dr Branislav Petrović, redovni profesor, Elektronski fakultet Niš, Univerzitet u Nišu
4. dr Milunka Damnjanović, redovni profesor u penziji, Elektronski fakultet Niš, Univerzitet u Nišu
5. Prof. dr Zlatko Bundalo, redovni profesor, Elektrotehnički fakultet Banjaluka, Univerzitet u Banjaluci

Datum odbrane: _____

Mojoj porodici

Podaci o doktorskoj disertaciji

Naslov doktorske disertacije:

Tolerisanje grešaka i energetska efikasnost kod sistema za rad u realnom vremenu sa vremenskom redundansom

Rezime:

Koncept sistema za rad u realnom vremenu (RTS) prisutan je u računarstvu već nekoliko decenija unazad. Primena RTS-a je evaluirala od sistema koji su pretežno razvijani za specijalne namene, do sistema koji se danas, u najrazličitijim oblicima, prisutni svuda oko nas. U današnje vreme, nove oblasti primene RTS-a diktiraju i novine vezane za projektantske zahteve, što se pre svega odnosi na sve strožije zahteve koji se tiču pouzdanosti kao i zahteve vezane za smanjenja potrošnje energije. Ovi zahtevi su u osnovi oprečni, s obzirom da se konačno slobodno vreme procesora koristi i za povećanje energetske efikasnosti RTS-a i za poboljšanje njegove tolerantnosti na grešaka. Centralni problem koji se razmatra u ovoj doktorskoj disertaciji je kako, na optimalan način, raspodeliti slobodno vreme procesora između tehnika za smanjenje potrošnje i tehnika za poboljšanje tolerantnosti na greške.

Rešenje postavljenog problema predstavljeno je u vidu novog heurističkog FT-DVFS algoritma koji ima za cilj smanjenje potrošnje procesora uz poštovanje svih vremenskih zahteva i zahteva vezanih za prevazilaženje otkaza kod RTS-a. U osnovi, FT-DVFS pripada kategoriji algoritama za dinamičko skaliranje napona (DVS) sa integrisanom analizom vremena odziva RTS-a radi provere ispunjenosti vremenskih ograničenja zadataka i unapred

postavljenih zahteva u pogledu nivoa tolerantnosti grešaka. U okviru disertacije, na bazi FT-DVFS algoritma, razvijen je simulator koji omogućava brzu i efikasnu analizu rada RTS-a sa različitih aspekata, kako vremenskog tako i aspekta vezanog za energetsku efikasnost i aspekta vezanog za tolerisanje grešaka. Rezultati simulacija pokazuju da se primenom FT-DVFS algoritma ostvaruje znatna ušteda u potrošnji procesora čak i kada se koriste samo dva frekventna/naponska nivoa kao i da ušteda u potrošnji procesora raste sa porastom broja dostupnih frekventnih/naponskih nivoa. Takođe, simulacijama je pokazano da na rezultate FT-DVFS algoritma utiče količina raspoloživog slobodnog vremena procesora, izražena faktorom iskorišćenosti procesora i to tako da ušteda u potrošnji opada sa povećanjem faktora iskorišćenosti. Simulacije vezane za analiza rada RTS-a sa aspekta vezanog za tolerisanje grešaka pokazale su da se primenom FT-DVFS algoritma može ostvariti manja ušteda u potrošnji ako su zahtevi vezani za prevazilaženje otkaza strožiji i obrnuto da što je ušteda u potrošnji procesora veća to je manja mogućnost prevazilaženja otkaza. U okviru disertacije predstavljeni su rezultati simulacija koje vrše poređenje heurističkog FT-DVFS algoritma sa optimalnim algoritmom za dodelu frekventnih/naponskih nivoa zadacima, koji pokazuju da FT-DVFS algoritam dolazi do rezultata u znatno kraćem vremenu i da u većini slučajeva, čak i za obimne skupove zadataka, FT-DVFS algoritam generiše skoro optimalna rešenja.

Ključne reči:
Sistemi za rad u realnom vremenu, tolerisanje grešaka, vremenska redundansa, analiza vremena odziva, tehnika dinamičkog skaliranja napona.

Naučna oblast:
Elektrotehničko i računarsko inženjerstvo

Uža naučna oblast:
Elektronika

UDK broj: ((621.31:065.011)+004.052.4):004.31

Doctoral dissertation information

Doctoral dissertation title:

Fault tolerance and energy efficiency in real-time systems with time redundancy

Abstract:

The concept of real-time systems (RTSs) is presented in the computer science for decades. During that period, the RTSs have evolved from special purpose microcomputer systems for industrial application to various forms of embedded system that are deeply ingrained in wide segments of daily life. The new application domains pose new design requirements and goals to RTSs, which are now often required to provide both fault tolerance and energy efficiency in addition to their main objective to compute and deliver correct results within a specified period of time. There is a fundamental tradeoff between these two additional requirements because fault tolerance techniques use slack time to improve reliability while low energy consumption techniques exploits slack time to increase energy efficiency. The central problem considered in the dissertation is how to optimally distribute the slack time between these techniques.

Dynamic voltage scaling (DVS) is known as one of the most effective low-energy technique for RTSs. However, most existing DVS techniques only focus on minimizing energy consumption without taking the fault-tolerant capability of RTSs into account. In order to solve specify problem in this dissertation, a new heuristic-based fault-tolerant dynamic voltage and frequency scaling (FT-DVFS) algorithm is developed. The goal of the

proposed algorithm is to minimize the amount of energy consumed by a real-time system under fault tolerance constraints while guaranteeing that all real-time tasks can complete successfully before their deadlines. Basically, the FT-DVFS is a DVS algorithm with integrated response time analysis (RTA) to check both the schedulability and the fault tolerant constraints of real-time task sets. The performances of FT-DVFS algorithm are evaluated by simulation in a custom build simulator. The simulation results are analyzed from three different points of view: the schedulability, the energy consumption, and the fault tolerance. The simulation results show that the proposed algorithm saves a significant amount of energy even with only two frequency/voltage levels, and the savings further increases with the increase of the number of frequency levels. Also, the simulations show that the reduction in power consumption, which can be achieved with FT-DVFS algorithm decreases with the increase of the processor utilization factor (i.e. processor spare time). The simulation results from the fault tolerant point of view show that the higher level of fault tolerance can only be attained through sacrificing a part of savings in power consumption, and vice versa. The proposed heuristic FT-DVFS algorithm is compared with the optimal DVS algorithm. The simulation analysis show that FT-DVFS algorithm achieves near-optimal solutions in very short computation time even for large task sets.

Keywords:
Real-time systems, fault-tolerances, time redundancy, response time analysis, dynamic voltage scaling technique.

Scientific area:
Electrical and Computer Engineering

Major scientific area:
Electronics

UDC number: ((621.31:065.011)+004.052.4):004.31

Sadržaj

Spisak slika

Spisak tabela

1 Uvod

Sistemi za rad u realnom vremenu (eng. *Real-Time System* - RTS) su računarski sistemi koji upravljaju i nadgledaju fizičke procese. Da bi RTS ispravno funkcionisao potrebno je da odziv na ulaznu pobudu bude ne samo funkcionalno korektan već i da bude generisan u tačno definisanom vremenskom intervalu pri čemu izostanak pravovremene reakcije može da dovede do katastrofalnih posledica [Bur97]. RTS-i danas nalaze široku primenu u komercijalnoj, vojnoj, medicinskoj, obrazovnoj i kulturnoj infrastrukturi savremenog društva. Sistemi za kontrolu procesa, sistemi za proizvodnju, sistemi za komunikaciju, upravljanje i nadzor, telekomunikacioni sistemi su samo neki primeri RTS-a.

Projektovanje savremenih RTS-a zahteva primenu rigoroznih metodologija projektovanja i analize u cilju povećanja efikasnosti i poboljšanja predviljivosti razvoja, gde se pod predvidljivošću smatra mogućnost da se još u fazi projektovanja uoče, kvalifikuju i optimizuju sve bitne karakteristike gotovog proizvoda. RTS, kao računarski sistem, može se predstaviti skupom funkcionalnih i skupom ne-funkcionalnih karakteristika [Nis97]. Funkcionalne karakteristike se odnose na glavne aktivnosti ili servise koje dati RTS pruža krajnjem korisniku, od računarskih igara i multimedijalnih aplikacija do kontrole automobila, aviona ili svemirskih letelica. Ne-funkcionalne karakteristike predstavljaju kvantitativne i kvalitativne mere funkcionalnog ponašanja sistema. Primarna ne-funkcionalna karakteristika RTS-a jeste *pravovremenost*, odnosno garancija izvršenja vremenski kritičnih funkcija u zadatim vremenskim rokovima. Pouzdanost, bezbednost, otpornost na otkaze i energetska efikasnost spadaju u bitne sekundarne ne-funkcionalne karakteristike. Cilj istraživanja obuhvaćenog ovom disertacijom jeste unapređenje metodologije projektovanja RTS-a u

pravcu poboljšanja predvidljivosti tri važne ne-funkcionalne karakteristike RTS-a: pravovremenost, otpornost na otkaze i energetska efikasnost.

Tradicionalna podela RTS-a je na *hard* RTS-e i na *soft* RTS-e. Kod *hard* RTS-a apsolutni imperativ je da sistem generiše odziv na spoljašnju pobudu u zadatom roku pri čemu prekoračenje roka može da dovede do katastrofalnih posledica na živote ljudi, materijalna sredstva ili okolinu (primer je sistem za kontrolu nuklearne elektrane, sistem za upravljanje letom aviona, i sl.). Kod *soft* RTS-a rokovi su važni, ali se povremeno mogu i prekoračiti, sve dok performanse sistema ostaju u zadatim okvirima (primer je telefonska centrala, "gubitak" pri prenosu i/ili obradi digitalne TV slike i sl.).

Osnovna jedinica softverske arhitekture RTS-a je zadatak (eng. *task*). Zadatak obuhvata jednu jasno definisanu aktivnost RTS-a, koja se inicira kao reakcija RTS-a na protok vremena ili na pojavu određenog događaja u fizičkom okruženju. RTS je tipično razložen na skup od više zadataka, koji kooperativno realizuju zahtevanu funkcionalnost. Pri tom, svakom zadatku su pridruženi određene vremenske karakteristike, kao: rok za izvršenje (eng. *deadline*), vreme izvršenja u najgorem slučaju (eng. *worst-case execution time*), vreme odziva (eng. *response time*), i minimalno vreme između iniciranja zadatka (eng. *minimum inter-arrival time*). Vreme između iniciranja zadatka i rok za njihovo izvršenje nametnuti su od strane okruženja, dok vreme izvršenja zadatka zavisi od hardverskih karakteristika RTS-a. Cilj projektovanja RTS-a je da se obezbedi da vreme odziva ni jednog zadatka ne prekorači rok za izvršenje pri bilo kom mogućem sledu događaja koji iniciraju izvršenje zadataka. Ukoliko je ovaj cilj ispunjen, za skup zadataka se kaže da je izvodljiv (engl. *schedulable*).

Postojanje više zadataka u RTS-u, koji se nezavisno iniciraju i nadmeću za korišćenje deljivih i ograničenih hardverskih resursa (npr. jedna CPU) dovodi do problema planiranja redosleda njihovog izvršenja. Redosledom izvršenja RT (eng. *real-time*) zadataka upravlja proces kernela koji se naziva planer (eng. *scheduler*). Uloga planera je da rasporedi izvršenje iniciranih zadataka u skladu sa njihovim prioritetima. Planeri mogu raditi na dva načina. Jedan je planiranje bez istiskivanja (eng. *non-preemptive*) kada se izvršenje zadatka ne može prekinuti, a drugi je planiranje sa istiskivanjem (eng. *preemptive*) kada se izvršenje zadatka može prekinuti usled pojave zadatka višeg prioriteta. Način na koji se određuje prioritet zadatka zavisi od politike (tj. strategije) planiranja koju planer sprovodi. Zadatak planera tj. određivanje rasporeda izvršenja zadataka nije nimalo lako zbog vremenskih parametara svih zadataka koji moraju biti ispunjeni. Često se dešava da problemi planiranja pripadaju klasi NP kompletnih problema. Prioriteti zadataka mogu biti statički, kada se prioriteti dodeljuju

zadacima u fazi projektovanja sistema i ne mogu se menjati u toku rada sistema i dinamički, kada prioriteti zadataka nisu fiksni, unapred dodeljeni, već se određuju i menjaju u toku rada sistema, i zavise od trenutnog stanja sistema.

Za potrebe analize RTS-a sa stanovišta izvodljivosti skupa zadataka razvijene su brojne matematičke metode. Jedna od ovih metoda koja nalazi široku primenu poznata je pod nazivom analiza vremena odziva (eng. *Response Time Analysis* - RTA). Ulazni parametri RTA su vremenski parametri zadataka, a izlaz odgovor na pitanje da li je skup zadataka izvodljiv [Jos96], [Lim03], [Bin04]. RTA ne zavisi od izabranog planera što je čini veoma pogodnom za široku primenu. U ovoj disertaciji tretiraju se *hard* RTS-i koji se modeliraju skupom periodičnih, međusobno nezavisnih zadataka sa statički prioritetom i planeri sa istiskivanjem koji se analiziraju korišćenjem RTA.

RTS-i su u često neposrednom kontaktu sa fizičkim okruženjem, pa su samim tim i podložni negativnim uticajima iz okruženja. Naime, pojave iz okruženja, kao što su pojačano elektromagnetsko zračenje, povišena temperatura, vibracije i sl., mogu da izazovu različite greške u radu RTS-a. Zbog toga, neophodno je u postupku projektovanja RTS-a razmatrati i aspekt tolerisanja grešaka.

RTS-i koji imaju mogućnost da nastave sa radom čak i nakon pojave otkaza nazivaju se sistemi za rad u realnom vremenu sa mogućnošću tolerisanja otkaza (eng. *Fault-Tolerant Real-Time Systems* – FT RTS). Postoje tri tipa otkaza koji se mogu javiti kod RTS-a: stalni, povremeni i prolazni. Stalni otkazi (eng. *permanent faults*) pojave se i traju sve dok se komponenta kod koje se javio otkaz ne popravi ili zameni drugom. Povremeni otkazi (eng. *imtermittent faults*) javljaju se s vremena na vreme u nepravilnim vremenskim intervalima. Prolazni otkazi (eng. *transient faults*) pojave se, traju neko vreme, a zatim nestanu. Danas se prolazni otkazi smatraju najzastupljenijim tipom otkaza [Cam92], [Sri04], [Bau05]. U ovoj disertaciji pažnja će biti usmerena ka prevazilaženju grešaka nastalih zbog prolaznih otkaza.

Osobina tolerisanja grešaka kod FT RTS-a se postiže uvođenjem nekog oblika redundanse. Redundansa može biti: hardverska, softverska, informaciona ili vremenska. Hardverska redundansa podrazumeva postojanje dodatnog hardvera sa svrhom detektovanja i/ili tolerancije otkaza. Softverska predstavlja dodavanje softvera, iznad onog neophodnog za obavljanje zadatih funkcija, a u cilju detekcije i tolerisanja otkaza. Informaciona redundansa ogleda se u dodavanju dodatnih informacija iznad zahtevanih za izvršavanje zadate funkcije, dok vremenska koristi dodatno vreme za izvršavanje funkcija oporavka sistema pri nastanku grešaka. U ovoj disertaciji akcenat će biti na vremenskoj redundansi kao tehnici za

prevazilaženje grešaka. Razlog je činjenica da je ova tehnika ne zahteva dodatni hardver, relativno je jeftina i posebno pogodna za primenu kod aplikacija gde postoje ograničenja vezana za težinu, veličinu i potrošnju. Pored toga, vremenska redundansa posebno je pogodna za toleranciju prolaznih otkaza što je još jedan razlog za njeno razmatranje u disertaciji [Kan03], [Đoš11b].

Metoda vremenske redundanse, koja će biti korišćena u disertaciji, je metoda ponovljenog izvršavanja zadatka (eng. *re-execution*). Osnovni princip ove metode sastoji se u ponovljenom izvršavanju celokupnog zadatka u toku čijeg izvršenja je detektovana greška sa ciljem njenog prevazilaženja. Ponovljeno izvršavanja zadatka je često korišćeno u literaturi kao osnova za razvoj tehnika za prevazilaženje prolaznih otkaza kod FT RTS-a. Treba napomenuti da imperativ neprekoračenja vremenskih rokova čini problem projektovanje FT RTS-a težim u odnosu na projektovanje računarskih sistema opšte namene otpornih na otkaze. Angažovanje RTS-a na detekciju i otklanjanje posledica otkaza može da dovede do manjka vremena za izvršenje RT zadatka, a time i do opasnosti od prekoračenja vremenskih rokova. S tim u vezi, uvođenje bilo kog mehanizma za tolerisanje grešaka nameće potrebu za modifikacijom i prilagođenjem matematičkog aparata za analizu izvodljivosti skupa zadataka [Kan03], [Pop07], [Gho95], [Bur96], [Izo08].

Danas, kao i za svaki elektronski sistem, bitna nefunkcionalna karakteristika RTS-a jeste njihova energetska efikasnost. Sve češći zahtevi vezani za proces projektovanja RTS-a odnose se na tehnike kojima se povećava energetska efikasnost RTS-a odnosno kontroliše potrošnja sistema sa ciljem da ona bude minimalna moguća [Uns03], [Jev08]. DVS (eng. *dynamic voltage scaling*) je tehnika koja se bavi upravljanjem potrošnjom procesora i njena osnovna ideja proistekla je iz činjenice da je potrošnja procesora srazmerna radnoj frekvenciji i kvadratu napona napajanja. Smanjivanjem napona napajanja uporedo sa smanjenjem radne frekvencije može se ostvariti ušteda u potrošnji procesora [Bur00]. Ako se DVS tehnike primene kod RTS-a dolazi do promene vremena potrebnog za izvršenja zadatka tako da se vreme potrebno da procesor izvrši zadatke RTS-a povećava usled smanjenja radne frekvencije procesora. U literaturi se često može naći pretpostavka da se skaliranjem radne frekvencije procesora za faktor α vreme izvršenja zadatka menja za faktor $1/\alpha$ [Fen08], [Ahm10], [Pin08]. Može se reći da DVS tehnika koristi slobodno vreme procesora kako bi smanjila njegovu potrošnju ali samo do mere koja ne dovodi do prekoračenja vremenskih rokova [Ahm10], [San09].

Tolerisanje grešaka korišćenjem vremenske redundanse odnosno metode ponovljenog izvršavanja zadatka, kao i upravljanje potrošnjom korišćenjem DVS tehnika su dva problema često odvojeno razmatrana u literaturi u kontekstu RTS-a. U literaturi je posvećeno mnogo manje pažnje objedinjenom razmatranju ova dva problema [Ahm10], [San09]. Ono što je zajedničko za ova dva problema jeste da koriste slobodno vreme procesora kako bi ostvarili svoje ciljeve. Naime, tolerisanje grešaka kod RTS-a korišćenjem metode ponovljenog izvršavanja zadatka koristi slobodno vreme procesora kako bi se zadatak, u toku čijeg izvršenja se desila greška, ponovo izvršio. DVS tehnike primenjene kod RTS-a kao posledicu imaju povećanje vremena izvršenja zadataka koristeći slobodno vreme procesora. Slobodno vreme procesora je ograničen resurs i ako DVS tehnike iskoriste veći deo tog resursa manje će ostati za tehnike tolerisanja grešaka i obrnuto. Pitanje koje se nameće jeste kojoj tehnici dodeliti koliko slobodnog vremena. Ova dilema je obrađena u disertaciji. Glavni cilj disertacije vezan je upravo za pitanje kako naći kompromis između zahteva vezanih za energetsku efikasnost i zahteva vezanih za mogućnost prevazilaženja prolaznih otkaza kod RTS-a.

Na putu do pronalaženja odgovora na prethodno pitanje bilo je neophodno modifikovati odnosno prilagoditi postojeće tehnike i razraditi nove metode projektovanja i analize RTS-a. Jedan od zadataka, koji je predstavljen u disertaciji, vezan je za RTA i kompleksnost računice za veći skup RT zadataka. Naime, za veći skup RT zadataka, broj iteracija kao i računica tokom svake iteracije je veoma složena. Kako bi se brže došlo do rešenja, za potrebe disertacije, realizovan je efikasan program za izračunavanje vremena odziva u okviru softverskog paketa Matlab.

U disertaciji je takođe obrađen problem primene RTA kod FT RTS-a, konkretno, određivanje najmanjeg vremena između moguće pojave dve uzastopne greške - T_{Fmin} koje dati RTS može da prevaziđe i nastavi sa normalnim funkcionisanjem. Rešenja ovog problema dato je u vidu programa koji omogućava brzo i efikasno izračunavanje parametra T_{Fmin} čak i za veće skupove RT zadataka.

Još jedna modifikacija RTA, koja je predstavljena u disertaciji, odnosi se na slučaj kada u RTS-u postoji jedan ili nekoliko zadataka kritičnih po pitanju bezbednosti sistema. Odnosno slučaj kada pojava greške u izvršenju kritičnog zadatka zahteva momentalni oporavak tj. kada izvršenje procedure za oporavak mora početi neposredno nakon što je greška detektovana.

U osnovnoj verziji analize vremena odziva podrazumeva se tehnika ponovljenog izvršavanja zadataka. U okviru disertacije modifikovana je osnovna verzija RTA za slučaj

metode tolerancije grešaka kod kojih se oporavak od greške postiže izvršavanjem alternativnog zadatka. Prilikom rešavanja ovog problema uzeta je u obzir činjenica da alternativni zadaci imaju različito vreme izvršenja (najčešće kraće) od vremena izvršenja zadataka RT sistema.

U okviru disertacije analiziran je rad RTS-a sa stanovišta korišćenja različitih planera i sa stanovišta korišćenja različitih načina oporavka od greške. Stanovište vezano za planere odnosi se na način dodele prioriteta zadacima RTS-a, dok se stanovište vezano za način oporavka odnosi se na oporavak RTS-a izvršavanjem alternativnog zadatka ili ponavljanjem zadatka u toku čijeg izvršenja je došlo do pojave greške.

Glavni doprinos disertacije je rešenje predstavljeno u vidu novog heurističkog algoritma za preraspodelu slobodnog vremena procesora na deo koji se dodeljuje tehnikama energetske efikasnosti i na deo koji se dodeljuje tehnikama za prevazilaženje otkaza [Djo12a], [Djo12b], [Đoš12a], [Đoš12b], [Djo13]. Algoritam je nazvan FT-DVFS (*fault tolerant dynamic voltage frequency scaling*) i ima za cilj smanjenje potrošnje procesora uz poštovanje svih vremenskih zahteva i zahteva vezanih za prevazilaženje otkaza kod RTS-a. Glavna novina predloženog FT-DVFS algoritma je korišćenje RTA za proveru ispunjenja zahteva vezanih za vremenska ograničenja RTS-a prilikom promene radnog napona odnosno radne frekvencije procesora. Prema dosadašnjim saznanjima ovo je prvi put da je RTA integrisana u neki DVS algoritam.

Za potrebe istraživanja obuhvaćenih ovom disertacijom razvijen je simulator u C++ programskom jeziku, koji omogućava proveru predloženog koncepta, olakšava iznalaženje optimalnih algoritamskih rešenja i procenu performansi predložene metode. Simulator omogućava brzu i efikasnu proveru izvodljivosti RTS-a, procenu mogućnosti tolerisanja grešaka kod RTS-a, kao i analizu RTS-a sa stanovišta energetske efikasnosti. Uz određene modifikacije, razvijeni simulator efikasno se može iskoristiti u postupku projektovanja energetski efikasnih FT RTS-a.

Ova disertacija, pored uvodnog dela, sadrži još sedam poglavlja. U drugom poglavlju definiše se pojam RTS-a i predstavlja njihov značaj. Takođe, predstavljena je struktura RTS-a i objašnjen pojam zadatka. Deo poglavlja posvećen je operativnim sistemima za rad u realnom vremenu i planerima RTS-a.

Treće poglavlje govori o FT RTS-ima, njihovoj primeni i značaju. U okviru ovog poglavlja objašnjavaju se osnovni pojmovi FT RTS-a, dati su uzroci nastanka otkaza i predstavljeni projektantski pristupi za prevazilaženje otkaza. Takođe predstavlja se pojam

redundanse, sa posebnim osvrtom na tehnike za prevazilaženje otkaza koje koriste vremensku redundansu.

Četvrti deo disertacije bavi se analizom vremena odziva. Razmatra se primena RTA kod RTS-a kod kojih se ne uzima u obzir mogućnost pojave greške i kod RTS-a kod kojih se greške javljaju. U okviru ovog dela date su i modifikacije RTA rađene za specijalne slučajeve *hard* RTS-a kao i rezultati eksperimentalnih analiza rada RTS-a za modifikovane verzije RTA.

Peto poglavlje govori o energetski efikasnim RTS-ima. Ovom poglavlje predstavlja pregled osnovnih tehnika upravljanja potrošnjom RTS-a. Centralni deo petog poglavlja rezervisan je za DVS tehnike, podele DVS tehnika i koncept njihove primene i specifičnosti kod RTS-a.

Šesti deo disertacije daje detaljan opis modela RTS-a na osnovu kojeg je razvijen heuristički FT-DVFS algoritam, rešenje koje omogućava projektovanje energetski efikasnih FT RTS-a. FT-DVFS algoritam nalazi frekvencije tako da potrošnja procesora bude najmanja moguća a da se zadovolje svi zahtevi vezani za prevazilaženje otkaza. U okviru ovog dela disertacije razmatra se i složenost heurističkog FT-DVFS algoritma.

U sedmom poglavlju predstavljaju se rezultati analize performansi FT-DVFS algoritma za različite modele procesora i različite slučajeve RT skupova zadataka: sintetičke i one iz "realnog" sveta. Analiziraju se slučajevi RTS-a bez mogućnosti i sa mogućnošću tolerisanja grešaka. U okviru ovog poglavlja izvršeno je i poređenje FT-DVFS metode sa optimalnom metodom iscrpljivanja. Kao zaključak, predstavljeni su sumirani najvažniji rezultati i usmerenja za moguća dalja istraživanja u oblasti energetski efikasnih RTS-a.

2 Sistemi za rad u realnom vremenu

Kako je računar postao manji, brži, pouzdaniji i jeftiniji, tako je spektar primene računara i računarskih sistema postao sve širi. Projektovani i realizovani prvenstveno kao mašine za rešavanje matematičkih problema, danas imaju uticaj na skoro sve segmente modernog života.

Posebnu grupu računarskih sistema predstavljaju sistemi za rad u realnom vremenu (eng. *Real-Time System* - RTS). Osnovna razlika RTS-a u odnosu na druge računarske sisteme je da valjanost rezultata obrade zavisi i od vremenskog trenutka kada su ti rezultati generisani. Znači, nije dovoljno da rezultat bude samo funkcionalno ispravan, već je neophodno i da bude dostupan u okviru unapred definisanog vremena.

Danas, RTS-i imaju veoma bitnu ulogu u mnogim oblastima svakodnevnog života, počevši od kontrole jednostavnih električnih uređaja pa do kontrole svemirskih stanica. Neke od tipičnih primena RTS-a su: robotika, kontrola leta u avionima, kontrola saobraćaja, kontrola procesa u industriji, u automatizovanim laboratorijama kao i u nuklearnim centralama, daljinska istraživanja podvodnih prostranstava, svemira kao i kontaminiranih ili oblasti visokog rizika, [Jev09a].

U ovom poglavlju je definisan pojam RTS-a, predstavljena je njegova struktura i objašnjen pojam zadatka RTS-a. Drugi deo poglavlja posvećen je operativnim sistemima za rad u realnom vremenu, sa posebnim osvrtom na planere RTS-a.

2.1 Definicija RTS-a

U literaturi se može naći više definicija odnosno više različitih interpretacija tačne prirode sistema za rad u realnom vremenu.

Prema [Dai08] sistem za rad u realnom vremenu je definisan kao svaki sistem kod koga je od velike važnosti vreme tj. trenutak kada se generiše izlaz. Ovo je najčešće zbog toga što je ulaz povezan sa nekim spoljašnjim procesom ili promenama iz fizičkog sveta, dok je izlaz taj koji opet na neki način utiče na te iste spoljašnje procese odnosno ostvaruje povratno dejstvo na promene iz fizičkog sveta. Vreme koje je potrebno da sistem generiše izlaz, na osnovu odgovarajuće ulazne informacije, mora biti dovoljno kratko da bi izlazni rezultat mogao pravovremeno da utiče na ulaz.

Sistem za rad u realnom vremenu može se definisati i kao bilo koja aktivnost obrade informacija ili sistem koji treba da odgovori na spolja generisan ulaz u ograničenom i određenom vremenskom periodu, [You82].

Još jedna definicija RTS-a, prema [Bur97] bila bi: RTS je sistem od kojeg se zahteva da reaguje na stimulacije iz okruženja (uzimajući u obzir i tok realnog vremena) u vremenskim intervalima određenim od strane okruženja.

Može se primetiti da se u svim definicijama pominje vreme potrebno da sistem generiše izlaz od odgovarajuće ulazne informacije. Ne postoji apsolutna odrednica koja bi definisala koliko malo to vreme odziva RTS-a treba da bude. Ono treba samo da bude "dovoljno malo" posmatrano relativno za dati sistem. To može biti od nekoliko milisekundi (sistem za vođenje projektila), desetina ili stotina milisekundi (telefonska centrala), do čak nekoliko sekundi (kontrola inertnog industrijskog procesa).

Prema svojoj prirodi RTS-i mogu biti klasifikovani kao *hard* RTS-i i kao *soft* RTS-i.

Za *hard* RTS-e je karakteristično postojanje striktnih krajnih rokova za završetak pojedinih kritičnih aktivnosti. Konsekvence propuštenih rokova ili nedeterminističkog ponašanja sistema mogu biti katastrofalne po okolinu, ljude i druge sisteme (tipičan primer je auto-pilot u avionu). Fokus u ovoj disertaciji je na *hard* RTS-ima.

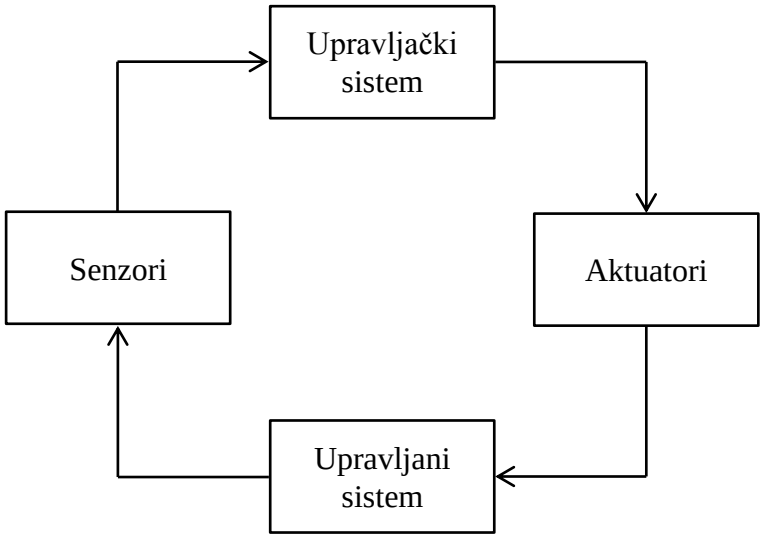
Kod *soft* RTS-a konsekvence propuštenih rokova i varijacije u vremenu odziva zadataka su relativno blaže i nemaju za posledicu katastrofalne posledice. Varijacije u vremenu odziva zadataka obično rezultuju u trenutnom i/ili delimičnom smanjenju kvaliteta funkcionisanja sistema bez izazivanja tragičnih posledica (tipičan primer je "privremeni gubitak slike" pri prenosu i/ili obradi digitalne TV slike).

Prema prethodnoj klasifikaciji RTS-a mogu se izdvojiti njihove bitne karakteristike. Recimo, za *hard* sisteme, bitno je teorijski dokazati njihovu *izvodljivost* (eng. *feasibility*), tj. pokazati da se zadati rokovi neće prekoračiti ni u kom slučaju, pri zadatim uslovima i resursima. Analiza izvodljivosti najčešće podrazumeva analizu *rasporedivosti* (engl. *schedulability*) definisanih poslova na raspoložive procesne jedinice.

Za *soft* sisteme, bitno je teorijski, simulaciono ili praktično pokazati da su *performanse* (engl. *performance*) sistema zadovoljavajuće, tj. ne odstupaju od zadatih graničnih vrednosti pod svim uslovima. To podrazumeva statističku analizu (npr. srednje vrednosti i disperzije) parametara performansi, kao što su kašnjenje (engl. *delay*) ili propusnost (engl. *throughput*).

2.2 Struktura RTS-a

Kod većine RTS-a, računar predstavlja samo jedan deo složenijeg sistema, tako da se njegov krajnji efekat može videti kroz interakciju mehaničkih ili nekih drugih sistema sa okolinom. Prema tome, za RTS se može reći da se sastoji od upravljačkog i upravljanog sistema Sl. 2-1 [Nis97].



Sl. 2-1: Struktura RTS-a

Upravljački sistem predstavlja elektronski podsistem fizičkog sistema. Upravljani sistem se može posmatrati kao okruženje sa kojim upravljački sistem interaguje. Upravljački sistem interaguje sa okruženjem posredstvom senzora i aktuatora.

Na primer, u slučaju automatizovane fabrike, upravljani sistem je fabrički pogon koji se sastoji od: robota, manipulatora, pokretnih traka, itd, dok upravljački sistem čini računar zajedano sa interfejsom prema operateru, koji upravlja i koordiniše sve aktivnosti u fabričkom pogonu.

Od ključne važnosti za ispravan rad RTS je da stanje okruženja, kako ga upravljački sistem "vidi", bude usklađeno sa stvarnim stanjem okruženja. U suprotnom, efekti akcija, koje upravljački sistem preduzima na bazi pogrešne predstave o tekućem stanju upravljanog sistema, mogu imati neželjene posledice. Iz tog razloga, neophodno je obezbediti periodično prikupljanje i pravovremenu obradu informacija.

Osnovne funkcije RTS sistema su sledeće:

- prikuplja informacije o tekućem stanju upravljanog sistema, npr. nadgleda fizičke veličine kao što su: temperatura, pritisak, hemijski sastav;
- obrađuje prikupljene informacije na bazi matematičkog modela upravljanog sistema;
- generiše izlazne signale koji utiču na ponašanje upravljanog sistema, a u cilju upravljanja ili ostvarivanje neke zadate performanse mere.

2.3 Zadatak RTS-a

Zadatak RTS-a ili RT zadatak (eng. *real-time task*) predstavlja jednu jasno definisanu aktivnost RTS-a sa pridruženim vremenskim karakteristikama [Sun96]. RT zadatak se može definisati kao softverski entitet ili program namenjen za izvršavanje određene aktivnosti, [Nis97]. Po pravilu RT zadatak je ponovljiv u smislu da se iznova izvršava uvek kada se javi potreba za obavljanjem aktivnosti koju realizuje. RTS tipično sadrži više RT zadataka koji se konkurentno izvršavaju. RT zadaci sa vremenskim ograničenjima u vidu krajnjih rokova su vremenski-kritični ili *hard* RT zadaci, dok su ostali zadaci *soft* zadaci.

Zadaci se razvrstavaju i po prioritetu u izvršavanju. Naime, ako RTS poseduje jedan CPU (jednoprocesorski sistem) onda se u jednom trenutku može izvršavati najviše jedan RT zadatak. Ako u isto vreme postoje zahtevi za izvršenjem više od jednog zadatka, svi inicirani zadaci, osim jednog, moraju da čekaju oslobađanje CPU-a. Odlaganje početka izvršenja zadatka, zbog zauzetosti CPU-a izvršenjem drugih zadataka, može da dovede do prekoračenja krajnjeg roka. Da ne bi dolazilo do prekoračenja rokova i nepoštovanja drugih vremenskih karakteristika zadataka, redosled izvršenja iniciranih zadataka kontroliše se uz pomoć prioriteta. Naime, svakom zadatku se pridružuje nenegativan ceo broj (prioritet) koji

odražava stepen hitnosti (kritičnosti) zadatka. Uvek kada postoji više od jednog iniciranog zadatka, za izvršenje se bira zadatak najvišeg prioriteta.

Prioritet RT zadatka može biti statički ili dinamički. Ako RT zadatak ima statički (fiksni) prioritet to znači da prioritet ostaje nepromenjen tokom celokupnog životnog veka RTS-a. Dinamički prioritet označava da se pri svakim ponovnim izvršenjem RT zadatka njegov prioritet može promeniti.

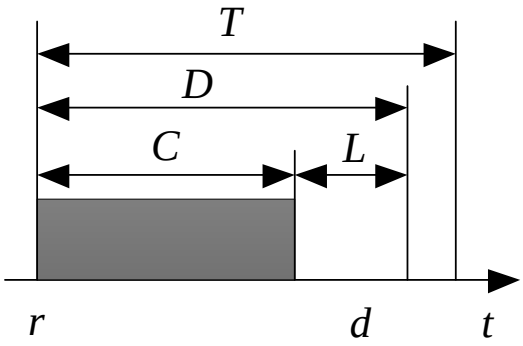
Još jedna moguća podela RT zadataka je na:

- zadatke čije izvršenje može biti prekinuto radi izvršenja zadatka višeg prioriteta (eng. *preemptive task*) i
- zadatke čije izvršenje ne može biti prekinuto radi izvršenja zadatka višeg prioriteta (eng. *non-preemptive*).

Zadaci se mogu podeliti i na međusobno zavisne odnosno nezavisne RT zadatke. Jedan od razloga međusobne zavisnosti RT zadataka može biti posledica deljenja resursa. Recimo, ako isti resurs koriste dva RT zadatka, izvršenje jednog mora biti blokirano dok se resurs, koji trenutno koristi drugi RT zadatak, ne oslobodi. Neki od protokola koji omogućavaju što bolju iskorišćenost deljivih resursa kako kod jednoprocesorksih tako i kod više procesorskih sistema mogu se naći u [Bas10], [Gua11], [Nol09]. U ovoj tezi razmatraju se samo međusobno nezavisni zadaci i jedini resurs koji zadaci dele je procesor.

Izvršenje RT zadataka je posledica ispunjenja nekog unapred definisanog uslova. Sa stanovišta načina iniciranja razlikuju se sledeća dva tipa zadataka:

- zadatak iniciran događajem (*event-driven*) odnosno iniciran pojavom određenog događaja (eksternog ili internog). U ovom slučaju, tačan vremenski trenutak pojave događaja kao i sled događaja nije unapred poznat, već se eventualno poznaje minimalno vreme između dva događaja, prosečno vreme između dva događaja i sl. Zato se ovi zadaci nazivaju i aperiodični ili asinhroni zadaci.
- vremenski iniciran zadatak (*time-driven*) gde se zadatak ponavlja u regularnim vremenskim intervalima, a perioda iniciranja zadatka se zadaje unapred, u fazi projektovanja sistema. Zato se ovi zadaci nazivaju sinhronim zadacima. Tipično, zadaci koji prikupljaju informacije od senzora su po svojoj prirodi periodični.



Sl. 2-2: *Osnovni parametri RT zadatka*

Bez obzira na vrstu zadatka, svaki zadatak karakterišu osnovni vremenski parametri, prikazani na Sl. 2-2, kao što su:

- *r*, trenutak iniciranja zadatka odnosno vremenski trenutak kada je zadatak dostupan za izvršenje;
- *C*, vremenski interval potreban za izvršenja zadatka pri čemu treba naglasiti da vreme izvršenja zadatka zavisi od složenosti obrade obuhvaćene zadatkom i brzine rada CPU-a, odnosno računarskim performansama upravljačkog sistema;
- *D*, relativni rok za izvršenje zadatka tj. maksimalni vremenski interval u toku kojeg zadatak mora biti izvršen pri čemu su krajnji rokovi određeni zahtevima koji potiču od upravljanog sistema;
- *d*, apsolutni rok za izvršenje zadatka je karakterističan parametar za *hard* RT zadatke koji se računa kao
  $d = r + D$;
- *T*, period zadatka (veličina karakteristična samo za periodične zadatke). Period se može definisati kao vremenski interval između dve uzastopna iniciranja periodičnog zadatka;
- *L*, (eng. *laxity*) vremenska rezerva tj. vremenski interval za koji zadatak može maksimalno zakasniti sa početkom izvršenja a da ne dođe do prekoračenja roka, pod uslovom da se nakon početka izvršenja zadatak izvršava u kontinuitetu, tj. bez prekidanja.

Za trenutke iniciranja periodičnog RT zadatka važi

$r_k = r_0 + kT$

gde je r_0 prvi trenutak iniciranja zadatka, a r_k je k+1 trenutak iniciranja zadatka, dok je T period RT zadatka. Takođe za pravilno definisan periodičan zadatak treba da bude ispunjen i uslov vezan za apsolutni rok za izvršenje tj. treba da važi

$$d_k = r_k + D.$$

Ako važi da je $D = T$ onda periodični zadatak ima relativni rok za izvršenje jednak periodu zadatka. Kod dobro definisanih RT zadataka važi da je $0 < C \leq D \leq T$.

Veličina karakteristična za procesor, a koja predstavlja deo procesorskog vremena potrebnog za izvršenje skupa RT zadataka, naziva se faktor iskorišćenosti procesora U i definisana je kao

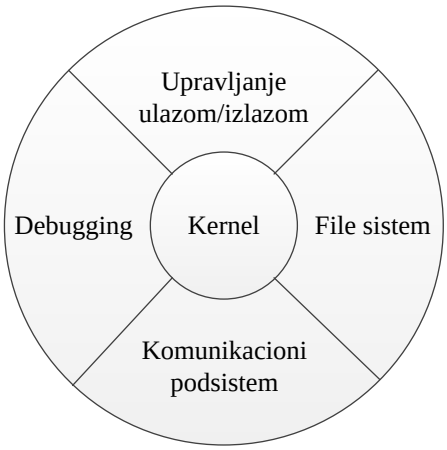
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

gde je n broj periodičnih zadataka RT skupa.

2.4 Operativni sistemi za rad u realnom vremenu

Interakcija između korisnika i RTS-a ostvarena je preko operativnog sistema za rad u realnom vremenu (eng. *Real-Time Operating Systems* – RTOS) koji predstavlja skup servisa koji korisniku omogućavaju pokretanje aplikativnog softvera. Iz ugla projektanta, uloga RTOS-a je da "sakrije" hardversku složenost sistema, tj. da za pristup hardveru obezbedi standardizovane softverske procedure. Takođe, RTOS treba da obezbedi podršku za multitasking, komunikaciju i sinhronizaciju zadataka, kontrolu pristupa deljivim resursima... Na taj način, RTOS stvara privid "virtuelne" mašine, nezavisne od konkretne hardverske platforme, što omogućava projektantu da se usredsredi na rešavanje konkretnog problema na višem nivou (identifikacija zadataka, komunikacija/sinhronizacija zadataka, obezbeđivanje zahtevanih performansi...) i da se u što većoj meri oslobodi implementacionih detalja niskog nivoa.

RTOS-i su projektovani tako da imaju modularnu i slojevitu strukturu kao što je prikazano na Sl. 2-3. Osnovu svakog RTOS-a čini jezgro (kernel) koje obezbeđuje podršku za: planiranje izvršenja zadataka, upravljanje memorijom, zaštitu deljivih resursa, obradu prekida, komunikaciju i sinhronizaciju zadataka.



Sl. 2-3: *Struktura RTOS-a*

U skladu sa temom disertacije, od posebnog interesa je kernel koji je zadužen za manipulaciju zadacima. Kernel planira redosled izvršenje zadataka i obezbeđuje mehanizme za kreiranje, startovanje i suspenziju zadataka. U toku rada sistema, svaki zadatak u svakom trenutku se nalazi u jednom od sledeća tri stanja:

- *Running* (izvršenje) - zadatak se trenutno izvršava od strane CPU-a. U jednoprocesorskom sistemu, u svakom trenutku, najviše jedan zadataka može biti u ovom stanju.
- *Ready* (spreman za izvršenje) - zadatak je aktiviran i spreman za izvršenje; svi resursi neophodni za izvršenje zadatka, osim CPU-a, su obezbeđeni. Kod jednoprocesorskih sistema, proizvoljan broj zadataka može biti u ovom stanju i svi oni su uređeni u tzv. red čekanja (*queue*). Zadatak koji se nalazi na početku reda čekanja je prvi sledeći zadatak koji će biti izvršen.
- *Waiting* (čekanje) - zadatak je neaktivan i čeka na događaj koji će ga aktivirati. Događaj može biti: isteklo zadato vreme, signal iz okruženja (npr. od senzora), interni signal generisan od strane nekog aktivnog zadatka. Broj zadataka koji se nalazi u stanju *waiting* nije ograničen.

Osim kernela postoji i "drugi sloj" RTOS-a koji sadrži nekoliko delova. Deo nazvan upravljanje ulazom-izlazom obezbeđuje drajvere standardnih hardverskih uređaja i omogućava hardversku nezavisnost aplikativnog softvera. Deo nazvan *File* sistem pruža podršku za kreiranje i brisanje fajlova i direktorijuma, upis/čitanje fajlova kod sistema koji poseduju *hard*-disk. Deo komunikacioni podsistem implementira komunikacione protokole niskog nivoa i drajvere za mrežnu komunikaciju. Deo *debugging* obezbeđuje nadgledanje

izvršenja zadataka, startovanje/zaustavljanje zadataka što se koristi u toku razvoja i testiranja korisničke aplikacije.

Većina RTOS je tako organizovana da se moduli, ili čak celi nivoi, mogu izostaviti, ako nisu neophodni, prilikom instalacije na ciljnoj hardverskoj platformi.

2.5 Algoritmi planera RTS-a

Algoritmi planera se mogu posmatrati kao skup pravila na osnovu kojih kernel dodeljuje procesorsko vreme RT zadacima u određenom redosledu. Izbor algoritma planera kod RTS-a u velikoj meri zavisi od konfiguracije RTS-a tj. da li je RTS jednoprocesorski, višeprocesorski ili distribuirani, [Lin10], [Jev02], [Jev04b], [Đoš07].

Jednoprocesorski algoritmi planera podrazumevaju jedan procesor na kome se mogu izvršavati RT zadaci. Kod jednoprocesorskih RTS-a u svakom trenutku može se izvršavati samo jedan RT zadatak dok ostali spremni zadaci čekaju svoj red na izvršenje. Višeprocesorski algoritmi planera podrazumevaju procesore sa više jezgara ili više procesora na kojima se mogu izvršavati RT zadaci. Kod višeprocesorskih RTS-a u svakom trenutku može se izvršavati više RT zadataka što zavisi od broja dostupnih procesora odnosno jezgara i slobodnih resursa. Distribuirani algoritmi planera podrazumevaju postojanje više fizički razdvojenih nezavisnih procesora na kojima se mogu izvršavati RT zadaci. Distribuirani RTS-a podrazumevaju postojanje RT komunikacionog protokola koji omogućava razmenu informacija između RT zadataka raspoređenih na različite procesore.

Planiranje izvršenja zadataka kod većine RTOS-a zasnovano je na prioritetima. Naime, svakom zadatku se pridružuje prioritet i uvek kada postoji više od jednog spremnog zadatka, za izvršenje se bira zadatka sa najvišim prioritetom.

Generalno, postoje dva načina za dodelu prioriteta zadacima:

- statički (ili fiksni) gde se prioriteti dodeljuju zadacima u fazi projektovanja sistema i ne mogu se menjati u toku rada sistema. Prioriteti se dodeljuju na bazi procenjene kritičnosti ili urgentnosti zadatka. Kriterijumi za procenu kritičnosti mogu biti: učestanost iniciranja zadatka (veća učestanost, viši prioritet), urgentnost (kraće zahtevano vreme odziva, viši prioritet), vreme izvršenja (duže vreme izvršenja, viši prioritet). Tipično, prioritet se određuje na bazi jednog ili kombinacijom više navedenih kriterijuma (npr. količnik vremena izvršenja i vremena odziva).

- dinamički (ili promenljivi) gde se prioriteti određuju i menjaju u toku rada sistema. Posebna jedinica kernela RTOS-a, tzv. dispečer, na bazi trenutnog stanja sistema dodeljuje prioritete zadacima. Dispečer neprekidno prati stanje sistema i pokušava da organizuje izvršenje iniciranih zadataka na način koji će obezbediti da svi oni ispoštuju zadate krajnje rokove. Informacije koje su pri tome dostupne dispečeru, a na bazi kojih on donosi svoje odluke (koji zadatak izvršavati, kada istisnuti zadatak koji se trenutno izvršava) su: očekivana vremena izvršenja zadataka i njihovi krajnji rokovi.

Način na koji se određuju statički prioriteti, u prvom slučaju, odnosno način na koji dispečer donosi odluke, u drugom slučaju, naziva se algoritam planiranja izvršenja zadataka (ili skraćeno algoritmi planera RTS-a). U skladu sa načinima dodele prioriteta razlikuju se statički i dinamički algoritmi planera RTS-a.

U zavisnosti od tipa RT zadataka, algoritmi planera mogu se podeliti i na, [Nis97]:

- algoritme sa mogućnošću istiskivanja zadataka (eng. *preemptive*) i
- algoritme bez mogućnosti istiskivanja zadataka (eng. *non-preemptive*).

U prvu grupu spadaju algoritmi planera kod kojih je karakteristično da se zadatak koji se trenutno izvršava može prekinuti dolaskom zadatka višeg prioriteta. Nakon izvršenja zadatka višeg prioriteta, prekinuti (istisnuti) zadatak nastavlja sa izvršenjem. Odgovornost planera je da obezbedi da istisnuti zadatak ne prekorači rok za izvršenje. Opšti zaključak je da ovakvi planeri zahtevaju veći prostor manevrisanja.

Za drugu grupu planera karakteristično je da izvršenje zadatka ne može biti prekinuto bez obzira na prioritet dolazećeg zadatka. Prednosti ovih planera su manja složenost, jednostavnija verifikacija, kao i mogućnost garancije pristupa bilo kom zajedničkom resursu ili zajedničkim podacima bez konflikta.

2.5.1 Osnovni algoritmi planera

U okviru ove sekcije biće predstavljeni osnovni algoritmi planera i to statički: RM (eng. *rate monotonic scheduling*) i DM (eng. *deadline monotonic scheduling*) i dinamički EDF (eng. *earliest deadline first scheduling*) i LLF (eng. *least laxity first scheduling*) algoritmi, [Bru04].

2.5.1.1 RM algoritam

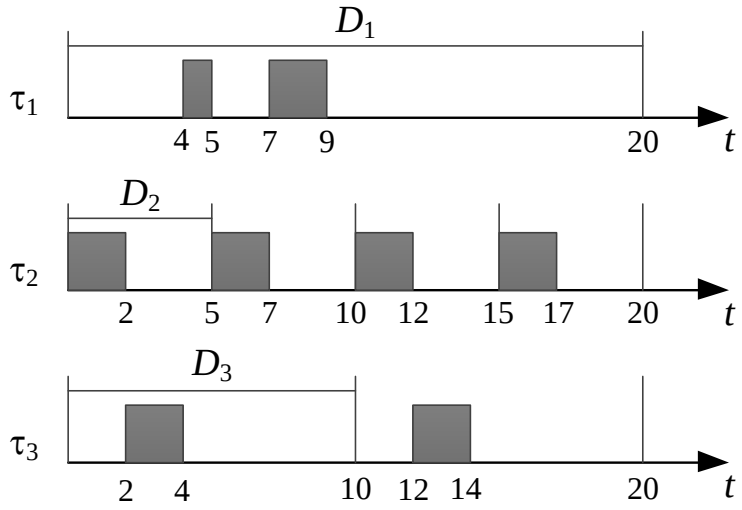
RM algoritam predstavlja bazični algoritam planera kada se govori o planiranju izvršenja zadataka sa fiksnim prioritetima. Kod RM algoritma perioda zadatka predstavlja osnovni kriterijum prilikom određivanja rasporeda izvršenja zadataka [Đoš10b]. Ovaj algoritam, RT zadatku sa kraćim periodom iniciranja dodeljuje veći prioritet prilikom izvršenja. Kako je period zadatka konstantna veličina, RM algoritam spada u grupu statičkih algoritama planera.

Tab. 2-1: *Primer za RM algoritam*

Zadatak	C_i	D_i	T_i
τ_1	3	20	20
τ_2	2	5	5
τ_3	2	10	10

RM algoritam može se ilustrovati primerom tri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i) dati u Tab. 2-1. U skladu sa vremenskim parametrima zadataka na Sl. 2-4 prikazana je raspodela procesorskog vremena za slučaj RM algoritma.

Kako je period zadatka τ_2 najkraći, u skladu sa RM algoritmom ovaj zadatka ima najveći prioritet. Nakon njega slede, po prioritetu, zadaci τ_3 i τ_1. Na Sl. 2-4 je prikazan interval od 20 vremenskih jedinica što predstavlja najmanji zajednički sadržalac perioda svih zadataka tako da je dovoljno prikazati raspored izvršenja zadataka baš u tom intervalu. Nakon perioda od 20 vremenskih jedinica, raspored izvršenja zadataka se ponavlja sa periodom 20.



Sl. 2-4: *Primer RM algoritma za slučaj tri periodična zadatka*

Kao zadatak sa najvišim prioritetom, τ_2 zauzima intervale tj. procesorsko vreme od [0, 2], [5, 7], [10, 12] i [15, 17]. Kako je period zadatka τ_2 5 vremenskih jedinica, to znači da će se on izvršiti 4 puta u posmatranom intervalu od 20 vremenskih jedinica. Sledeći po prioritetu je zadatak τ_3, čiji je period izvršenja 10 vremenskih jedinica, tako da će se on izvršiti ukupno 2 puta u posmatranom intervalu. Ako pretpostavimo da se zahtevi za izvršenje zadataka poklapaju sa periodom T_i, to znači da se zahtev za izvršenje zadatka τ_3 javlja u trenucima t=0 i t=10. U oba slučaja je procesor zauzet baš u tom trenutku, tako da ovaj zadatak čeka prve slobodne intervale za izvršenje. Odmah nakon izvršenja zadatka τ_2 izvršava se zadatak τ_3 i zauzima intervale od [2, 14] i [12, 14]. Kao zadatak sa najmanjim prioritetom, τ_1 se izvršava poslednji, pri čemu i on čeka na prve slobodne intervale za svoje izvršenje. Kao što se i sa Sl. 2-4 može videti zadatak τ_1 zauzima intervale od [4, 5] i [7, 9] i izvršava se iz dva dela usled pojave zahteva za izvršenjem zadatka većeg prioriteta τ_2. Sa Sl. 2-4 može se videti da se svi zadaci izvršavaju na vreme, pre njihovog roka za izvršenje.

2.5.1.2 DM algoritam

Kao i RM algoritam, i DM algoritam spada u grupu algoritama planera zadataka sa fiksnim prioritetima. Za razliku od RM algoritma kod koga je veličina T osnovni kriterijum za dodelu prioriteta zadacima, kod DM algoritma to je veličina D. Odnosno, DM algoritam prioritete zadacima dodeljuje na osnovu njihovog relativnog roka za izvršenje i to tako da se zadatku sa kraćim relativnim rokom za izvršenje dodeljuje veći prioritet. Ovaj algoritam spada u grupu statičkih algoritama planera.

DM algoritam može se ilustrovati primerom tri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i) dati u Tab. 2-2. U skladu sa vremenskim parametrima zadataka na Sl. 2-5 prikazana je raspodela procesorskog vremena za slučaj DM algoritma.

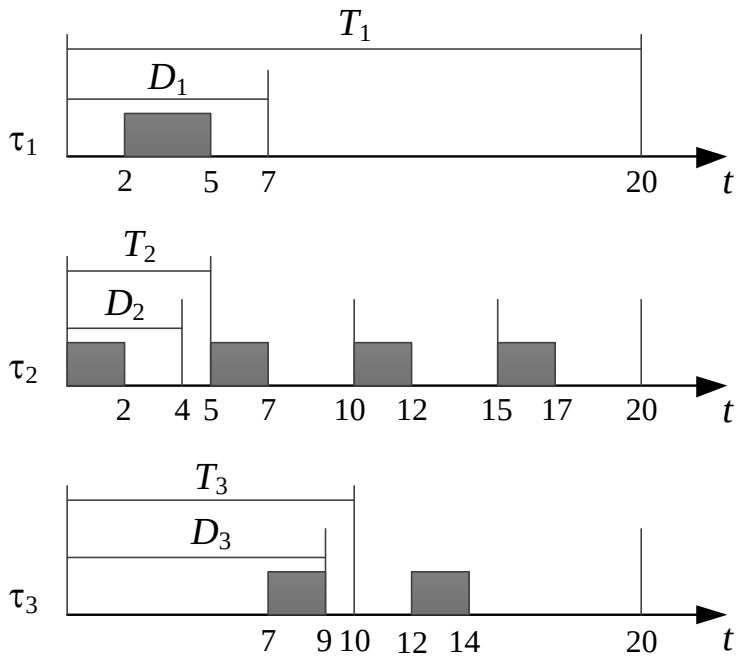
Tab. 2-2: *Primer za DM algoritam*

Zadatak	C_i	D_i	T_i
τ_1	3	7	20
τ_2	2	4	5
τ_3	2	9	10

Kako je rok za izvršenje zadatka τ_2 najkraći, prema DM algoritmu, ovaj zadatak ima najveći prioritet. Nakon njega, po prioritetu slede zadaci τ_1 i τ_3. Kako su periodi izvršenja

zadataka 20, 5 i 10 i ovde će od interesa biti interval od 20 vremenskih jedinica kao što je i prikazano na Sl. 2-5.

Kao zadatak sa najvišim prioritetom, τ_2 zauzima intervale tj. procesorsko vreme od [0, 2], [5, 7], [10, 12] i [15, 17]. Kako je period zadatka τ_2 5 vremenskih jedinica, to znači da će se on izvršiti 4 puta u posmatranom intervalu od 20 vremenskih jedinica. Sledeći po prioritetu je zadatak τ_1, čiji je period izvršenja 20 vremenskih jedinica, tako da će se on izvršiti samo jednom u posmatranom intervalu. Ako i ovde pretpostavimo da se zahtevi za izvršenje zadataka poklapaju sa periodom T_i, to znači da se zahtev za izvršenje zadatka τ_1 javlja u trenutku t=0 i kako je tada procesor zauzet, ovaj zadatak čeka prvi slobodan interval za svoje izvršenje, a to je odmah nakon prvog izvršenja zadatka τ_2 odnosno u periodu od [2, 5]. Kao zadatak sa najmanjim prioritetom, τ_3 se izvršava poslednji. Zahtevi za izvršenje ovog zadatka javljaju se u trenucima t=0 i t=10, ali je u oba slučaja u tom trenutku procesor zauzet tako da τ_3 čeka prve slobodne intervale za svoje izvršenje. Kao što je i prikazano na Sl. 2-5, zadatak τ_3 zauzima intervale od [7, 9] i [12, 14]. Sa Sl. 2-5 može se videti da se svi zadaci izvršavaju na vreme, pre njihovog roka za izvršenje.



Sl. 2-5: *Primer DM algoritma za slučaj tri periodična zadatka*

2.5.1.3 EDF algoritam

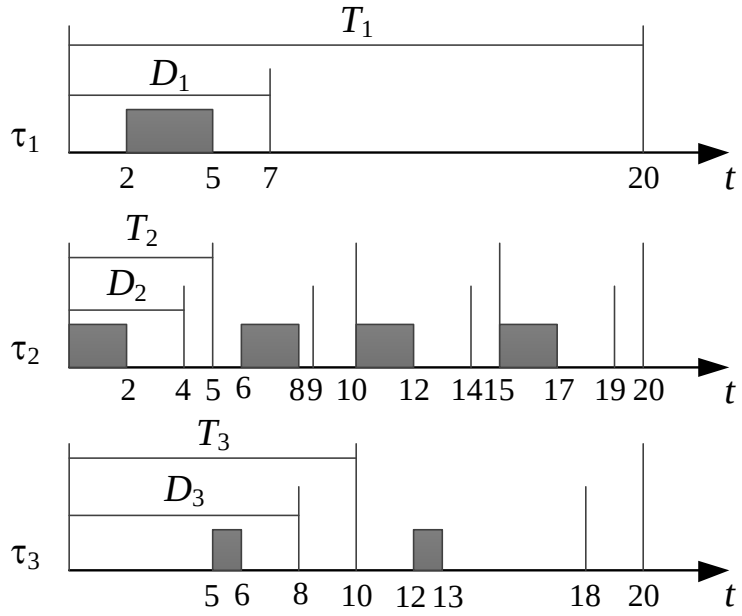
EDF je dinamički algoritam planera kod koga se prioriteti zadacima dodeljuju na osnovu njihovih apsolutnih rokova, odnosno najveći prioritet ima zadatak koji je najbliži svom krajnjem roku. Drugim rečima, prioritet zadatka koji čeka da bi bio izvršen vremenom raste.

Tab. 2-3: *Primer za EDF algoritam*

Zadatak	C_i	D_i	T_i
τ_1	3	7	20
τ_2	2	4	5
τ_3	1	8	10

EDF algoritam može se ilustrovati primerom tri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i) dati u Tab. 2-3. Za sva tri RT zadatka važi da je prvo vreme iniciranja $r_0 = 0$. U skladu sa vremenskim parametrima zadataka na Sl. 2-6 prikazana je raspodela procesorskog vremena za slučaj EDF algoritma.

Kako su periodi izvršenja zadataka 20, 5 i 10 od interesa je interval od 20 vremenskih jedinica kao što je i prikazano na Sl. 2-6.



Sl. 2-6: *Primer EDF algoritma za slučaj tri periodična zadatka*

U trenutku t=0 sva tri zadatka su spremna za izvršenje, međutim kako je apsolutni rok zadatka τ_2 najkraći (d_1=7, d_2=4 i d_3=8), prema EDF algoritmu, ovaj zadatak imaće najveći

prioritet. Nakon što se zadatak τ_2 izvrši, u trenutku t=2 spremni za izvršenje su τ_1 i τ_3 i kako je trenutku apsolutni rok zadatka τ_1 kraći ovaj zadatak se naredni izvršava i zauzima procesorsko vreme od 3 vremenske jedinice tj. vremenski interval [2, 5].U trenutku t=5 spremni za izvršenje su τ_2 i τ_3, sada je potrebno ponovo izračunati apsolutni rok zadatka τ_2 i uporediti ga sa apsolutnim rokom zadatka τ_3. Kako je d_2=9 i d_3=8 prioritet se dodeljuje zadatku τ_3 za čije izvršenje je potrebno procesorsko vreme od 1 vremenske jedinice tj. izvršenje ovog zadatka je u vremenskom intervalu [5, 6]. U trenutku t=6 jedino spreman zadatak za izvršenje je zadatak τ_2 koji se izvršava i zauzima vremenski interval [6, 8]. Naredni trenutak od interesa je trenutak t=10 kada su spremni za izvršenje zadaci τ_2 i τ_3. Kako je d_2=14 i d_3=18 zaključuje se da zadatak τ_2 ima veći prioritet, on se i izvršava i zauzima vremenski interval [10, 12]. U trenutku t=12 jedino spreman zadatak za izvršenje je zadatak τ_3 koji se izvršava u periodu [12, 13]. Još jedan trenutak od interesa je t=15 kada je jedino spreman za izvršenje zadatak τ_2, on se izvršava i zauzima vremenski interval [15, 17]. Sa Sl. 2-6 može se zaključiti da se svi zadaci izvršavaju na vreme, pre njihovog roka za izvršenje.

2.5.1.4 LLF algoritam

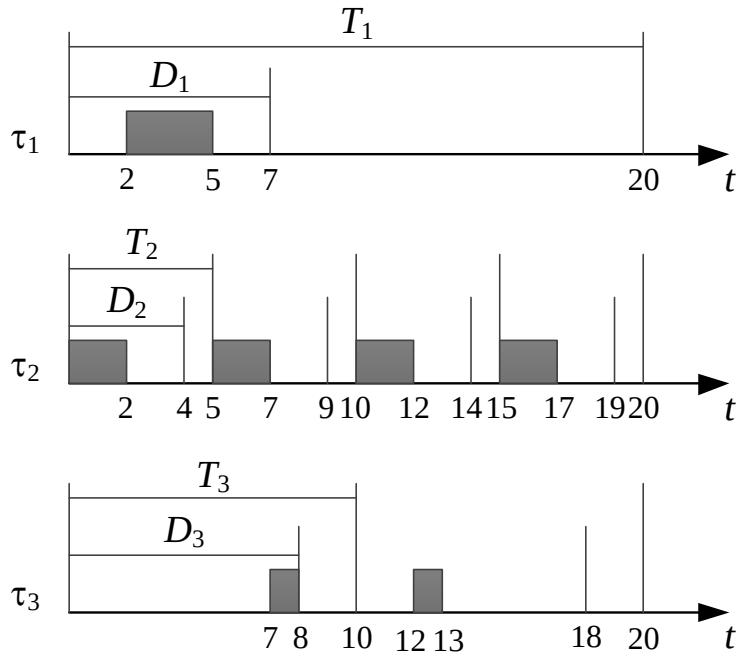
LLF je dinamički algoritam planera kod koga se prioriteti zadacima dodeljuju na osnovu vremenske rezerve L, (eng. laxity) i to tako da najveći prioritet dodeljuje zadatku sa najmanjom vremenskom rezervom L.

LLF algoritam može se ilustrovati primerom tri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i) dati u Tab. 2-4. Za sva tri RT zadatka važi da je prvo vreme iniciranja r_0 = 0. U skladu sa vremenskim parametrima zadataka na Sl. 2-7 prikazana je raspodela procesorskog vremena za slučaj LLF algoritma.

Tab. 2-4: *Primer za LLF algoritam*

Zadatak	C_i	D_i	T_i
τ_1	3	7	20
τ_2	2	4	5
τ_3	1	8	10

Kako su periodi izvršenja zadataka 20, 5 i 10 od interesa je interval od 20 vremenskih jedinica kao što je i prikazano na Sl. 2-7.



Sl. 2-7: *Primer LLF algoritma za slučaj tri periodična zadatka*

U trenutku t=0 sva tri zadatka su spremna za izvršenje, međutim kako je vrednost vremenske rezerve zadatka τ_2 najmanja (L_1=4, L_2=2 i L_3=7), prema LLF algoritmu, ovaj zadatak imaće najveći prioritet. Nakon što se zadatak τ_2 izvrši, u trenutku t=2 spremni za izvršenje su τ_1 i τ_3 ali kako je vremenska rezerva zadatka τ_1 najmanja ovaj zadatak se naredni izvršava i zauzima procesorsko vreme od 3 vremenske jedinice tj. vremenski interval [2, 5]. U trenutku t=5 spremni za izvršenje su τ_2 i τ_3. Kako je L_2=2 a takođe L_3=2 sada je moguće izabrati bilo koji od ova dva zadatka. Neka to bude recimo zadatak τ_2 koji izvršavanjem zauzima interval [5, 7]. Nakon njega izvršava se zadatak τ_3 koji zauzima vremenski interval [7, 8]. Naredni trenutak od interesa je trenutak t=10 kada su spremni za izvršenje zadaci τ_2 i τ_3. Kako su sada veličine parametra L_2=2 i L_3=8 naredni zadatak koji se izvršava je τ_2 koji zauzima vremenski interval [10, 12]. Nakon zadatka τ_2 izvršava se i zadatak τ_3 koji zauzima interval [12, 13]. Još jedan trenutak od interesa je t=15 kada je jedino spreman za izvršenje zadatak τ_2, on se izvršava i zauzima vremenski interval [15, 17]. Sa Sl. 2-7 može se zaključiti da se svi zadaci izvršavaju na vreme, pre njihovog roka za izvršenje.

3 Sistemi za rad u realnom vremenu sa mogućnošću tolerisanja grešaka

RTS-i otporni na otkaze (eng. *fault-tolerant real-time systems* – FT RTS) definišu se kao visoko-pouzdani RTS-i koji i u vrlo nepovoljnim, pa i ekstremnim uslovima, zahvaljujući pre svega svojoj sposobnosti da tolerišu greške, nastavljaju da izvršavaju svoje funkcije, [Kan10]. Postoji više faktora koji su doveli do razvoja i širenja koncepta FT RTS-a. Savremeni elektronski sistemi postaju sve složeniji, sa sve većim brojem sastavnih komponenti, što neminovno povećava verovatnoću pojave otkaza. Drugi bitan faktor je napredak VLSI tehnologije koja je mnoge tehnike otpornosti na otkaze učinila praktično izvodljivim. Konačno, mnogi sistemi koji su ranije bili mehanički sada su elektronski. Elektronski sistemi su po pravilu manje pouzdani od mehaničkih, te su zbog toga neophodne dodatne mere za povećanje njihove pouzdanosti, [Kor07].

U ovom poglavlju dat je pregled primene sistema otpornih na otkaze, obrađeni su osnovni pojmovi FT RTS-a kao i karakteristike i uzroci nastanka otkaza. Takođe su dati projektantski pristupi za prevazilaženje otkaza, objašnjen je pojam redundanse sa posebnim osvrtom na tehnike za prevazilaženje otkaza koje koriste vremensku redundansu.

3.1 Primena FT RTS-a

Postojeće primene sistema otpornih na otkaze mogu se svrstati u četiri primarne oblasti, [Joh89]:

- aplikacije sa dugim vremenom života - tipični primeri su svemirske letelice i sateliti bez ljudske posade. Radi se o sistemima koji duži vremenski period moraju raditi samostalno, bez mogućnosti direktne intervencije radi popravke eventualnog kvara, te stoga trebaju postajati ugrađeni mehanizmi koji će omogućiti da u slučaju pojave kvara, sistem samostalno preduzme odgovarajuće aktivnosti kako bi se predupredio otkaz. Tipični zahtevi koji se postavljaju za ovakve aplikacije su da imaju verovatnoću od najmanje 0.95 da će biti operativne posle perioda od deset godina.
- aplikacije sa kritičnim izračunavanjima - primeri su sistemi za kontrolu leta aviona, vojni sistemi, i određeni tipovi industrijskih kontrolera. U aplikacijama sa kritičnim izračunavanjima neispravan rad sistema može da dovede do katastrofalnih posledica. Tipičan zahtev koji se postavlja pred ovakve sisteme je da imaju pouzdanost od $0{,}9_7$ na kraju tročasovnog perioda rada. Glavni cilj u skoro svim aplikacijama sa kritičnim izračunavanjima je sprečiti da elektronika bude slaba tačka u sistemu.
- aplikacije sa odloženim održavanjem - najčešće se javljaju u slučajevima kada su operacije održavanja (servisiranja) ekstremno skupe, neprikladne, ili teške za izvođenje. Osnovni cilj je koristiti toleranciju otkaza ne bi li dozvolili odlaganje servisiranja do prikladnijeg i ekonomičnijeg vremena. Osoblje održavanja može obilaziti lokaciju mesečno i vršiti neophodne popravke. Između ovih servisnih obilazaka, sistem koristi toleranciju otkaza za neprekidno izvršavanje svojih zadataka.
- aplikacije visoke dostupnosti - primeri su bankarski i drugi sistemi zasnovani na *on-line* opsluživanju korisnika od kojih se zahteva visok nivo dostupnosti. Korisnici ovakvih sistema, nakon što su zatražili uslugu očekuju veliku verovatnoću da budu opsluženi u kratkom vremenu.

3.2 Osnovni pojmovi

Tri osnovna termina u oblasti projektovanja sistema otpornih na otkaze su, [Arl99]:

- otkaz (eng. *fault*) - se može definisati kao fizički defekt, nepotpunost, ili proboj koji se javlja unutar neke hardverske ili softverske komponente, ali i kao mana, slabost ili ostarelost određene hardverske ili softverske komponente. Primeri otkaza su kratki spojevi između električnih provodnika, prekidi provodnika i sl. Primer softverskog otkaza je programska petlja u koju kada se uđe, više se ne može izaći. Otkaz može biti unutrašnji ili spoljašnji. Unutrašnji otkaz je otkaz u sistemu koji je prvo bio prikriven

a zatim se aktivirao pod dejstvom nekog unutrašnjeg procesa, dok je spoljašnji otkaz posledica delovanja faktora izvan sistema.

- greška (eng. *error*) - je manifestacija otkaza. Konkretno, greška je odstupanje od tačnosti ili ispravnosti. Na primer, pretpostavimo da na vezi u nekom digitalnom kolu postoji spoj koji rezultuje time da je linija uvek na nivou logičke 1. Fizički spoj je otkaz u kolu. Ako se pojavi stanje koje zahteva da se linija prebaci na nivo logičke 0, vrednost na liniji će biti pogrešna i javiće se greška. S druge strane, ako je stanje kola takvo da na konkretnoj liniji treba biti nivo logičke 1, postojanje otkaza ne uzrokuje grešku. Dakle, čak iako postoji otkaz, greška ne mora da se ispoljava u svim situacijama. Greška može biti detektovana od strane algoritama, tj. mehanizama za detekciju greške, može se desiti i da ne bude otkrivena tj. prepoznata na vreme kao greška, a nekada greška nestane pre i nego bude detektovana. Najčešće je greška uzrok novih greška u sistemu.
- kvar (eng. *failure*) - nastaje kada greška rezultira time da sistem neku svoju funkciju izvršava pogrešno. Suštinski, kvar je neizvršavanje neke akcije kako bi trebalo, ili kako je očekivano. Kvar takođe predstavlja i izvršavanje neke funkcije u neodgovarajućem kvalitetu ili obimu.

Postoji uzročno posledična veza između ova tri termina[1]: otkaz je uzrok grešaka, dok je greška uzrok otkaza. Međutim sve ovo ne treba posmatrati suviše restriktivno, naime, nekoliko grešaka se mogu javiti pre nego se javi kvar. Takođe, RTS su obično velike složenosti sastavljeni od komponenata koje su takođe veoma složene. Zbog toga je moguće da se otkaz u jednom podsistemu manifestuje kao greška, a u drugom kao kvar, a to opet kao otkaz na višem nivou hijerarhije itd.

3.3 Uzroci nastanka otkaza

Otkazi mogu biti rezultat raznovrsnih pojava unutar same elektronske komponente, u okruženju komponente ili mogu biti uzrokovani još za vreme projektovanja ili proizvodnje komponente ili sistema. Mogući uzroci otkaza mogu se povezati sa problemima u četiri osnovne oblasti, [Joh89]:

[1] Što se tiče prevoda na srpski jezik pojmova *fault, error* i *failure*, mogu se naći nekoliko varijanti. Jedna od njih je kvar, greška i otkaz, [SRV], [OSO], [FTS], druga varijanta je otkaz, greška i kvar, [PES], [TES]. Još jedna mogućnost prevoda je otkaz, greška i pad, [Mil11]. Kako se izraz *fault-tolerant* prevodi u većini slučajeva sa otporan na otkaze, nadalje u disertaciji biće korišćen izraz otkaz kada se govori o pojmu *fault*.

- specifikacija - greške u specifikaciji gde spadaju netačni algoritmi, arhitekture, ili pogrešne specifikacije u procesu projektovanja hardvera i softvera.
- realizacija - greške u realizaciji koje mogu proizvesti otkaze zahvaljujući lošem projektu, lošem izboru komponenti, lošoj konstrukciji, ili greškama u kreiranju softvera.
- komponenta - defekti komponenti kao što su nesavršenosti u proizvodnji, slučajni otkazi uređaja, i istrošenost komponenti. Defekt može biti i rezultat preloma veza u kolu ili korozije metala.
- spoljašnji faktori - kao na primer, radijacija, elektromagnetna interferencija, greške operatera, i ekstremne promene u radnoj sredini. Elektronski sistemi su veoma osetljivi na temperaturske varijacije i elektrostatičke izvore kao što su munje ili drugi vremenski efekti. Takođe, greške operatera se smatraju za spoljašnje poremećaje jer je, gledano sa fizičke strane samog sistema, operater spoljašnji uticaj. Naime, operater može sistemu zadati pogrešne komande, koje, na kraju, dovode do kvara sistema.

3.4 Karakteristike otkaza

Osnovne karakteristike otkaza su, [Dub13]:

- priroda otkaza - opisuje tip otkaza tj. da li je on hardverski ili softverski.
- rasprostranjenost otkaza - opisuje da li je uticaj otkaza lokalizovan na konkretni hardverski ili softverski modulu, ili on globalno utiče na rad sistema.
- trajanje otkaza – odnosi se na dužinu vremenskog perioda u kom je otkaz aktivan. U zavisnosti od trajanja postoje tri tipa otkaza koji se mogu javiti kod RTS-a:
    - stalni (permanentni) otkazi (eng. *permanent faults*) - pojave se i traju sve dok se komponenta kod koje se javio otkaz ne popravi ili zameni drugom (npr. prekid električne veze ili greška u pisanju programa);
    - povremeni otkazi (eng. *imtermittent faults*) - javljaju se s vremena na vreme u nepravilnim vremenskim intervalima (npr. usled uticaja povišene temperature na komponentu koja je osetljiva na toplotu - komponenta prestane sa radom, a kada se ohladi nastavi);
    - prolazni (tranzijentni) otkazi (eng. *transient faults*) – pojave se, traju neko vreme, a zatim nestanu (npr. prolazna greška može nastati kao posledica

negativnog uticaja elektromagnetnog zračenja ili radioaktivnosti na hardver pri čemu greška nestaje čim ova smetnja nestane);

U mnogim radovima koji se bave proučavanjem prolaznih otkaza pokazano je da su prolazni otkazi najzastupljeniji [Sie78], [Cms82], [Iye86], [Cam92], [Sri04], [Bau05]. Konkretno, Siewiorek je u radu [Sie78] pokazao da se prolazni otkazi javljaju 30 puta češće nego stalni, a slični rezultati prikazani su i u [Cms82]. Rezultati eksperimenta prikazani u [Iye86] pokazuju da 83% svih otkaza čine prolazni ili povremeni otkazi. U ovoj disertaciji pažnja će biti usmerena upravo ka prevazilaženju grešaka nastalih zbog prolaznih otkaza kao i na uticaj koje one imaju u procesu projektovanja RTS-a.

3.5 Projektantski pristupi za prevazilaženje otkaza kod FT RTS-a

Postoje tri osnovna projektantska pristupa za poboljšavanje ili zadržavanje nominalnih performansi RTS-a u okruženju u kome su mogući otkazi: izbegavanje otkaza, maskiranje otkaza, i tolerancija otkaza, [Pra96].

Izbegavanje otkaza je bilo koja tehnika koja se koristi za sprečavanje pojave otkaza. Izbegavanje otkaza uključuje procese kao što su preispitivanja projekta, zaštita komponenti od dejstva visokog napona, testiranje, i druge metode kontrole kvaliteta. Naime, ako se pregled projekta izvrši korektno, mnoge greške nastale u specifikaciji se mogu otkloniti. Takođe, ako se sistem oklopi mogu se sprečiti spoljašnji poremećaji. A isto tako, ako se sistem pažljivo testira, mnogi otkazi se na vreme mogu otkriti i otkloniti pre nego što započne eksploatacija sistema.

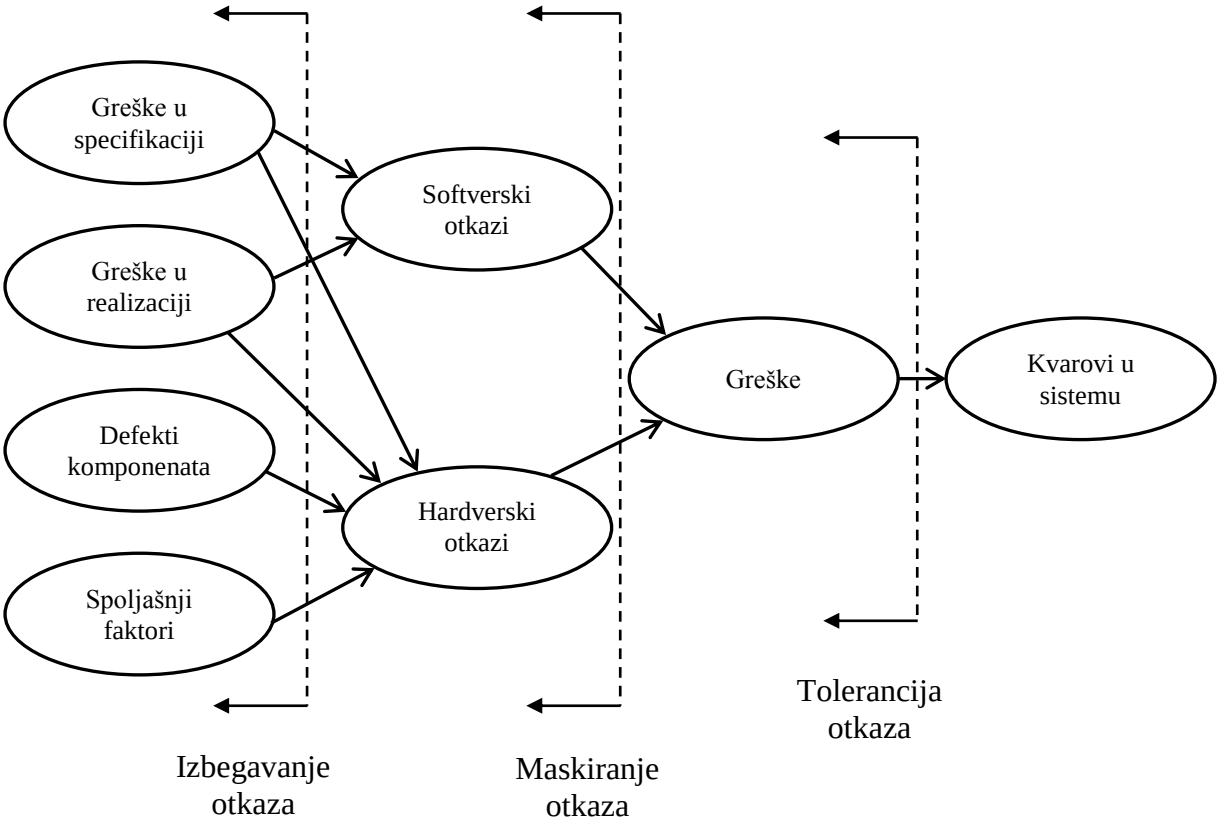
Maskiranje otkaza je bilo koja tehnika koji sprečava da otkaz u sistemu proizvede grešku unutar šire strukture sistema. Primer maskiranja otkaza su memorije sa mogućnošću korekcije grešaka, koje koriguju memorisane podatke pre nego ih sistem koristi. Na taj način sistem ne oseća otkaz koji se desio unutar memorije. Još jedan primer maskiranja otkaza je većinsko izglasavanje. Ono se koristi u sistemima gde postoje tri istovetna modula, i gde dva, na osnovu većine mogu maskirati treći neispravan modul.

Tolerancija otkaza je sposobnost sistema da nastavi sa izvršavanjem svojih zadataka i posle pojave otkaza. Osnovni cilj tolerancije otkaza je sprečavanje pojave kvara u sistemu usled pojave otkaza. Tolerancija otkaza se može postići mnogim tehnikama. Jedna od njih se sastoji u tome da se otkaz detektuje i locira, a da se zatim, sistem rekonfiguriše kako bi se eliminisao uticaj pokvarene komponente ili modula. Uopšteno govoreći, rekonfiguracija je

proces uklanjanja pokvarenog entiteta iz sistema i vraćanje sistema u operativno stanje. Rekonfiguracija uključuje sledeće aktivnosti:

- detekcija otkaza - proces otkrivanja tj. prepoznavanja otkaza.
- lokalizacija otkaza - proces određivanja gde se otkaz pojavio.
- izolacija otkaza - proces izolovanja komponente pogođene otkazom i sprečavanja da se efekti nastalog otkaza prošire kroz sistem.
- sanacija otkaza - proces koji obezbeđuje da se i u prisustvu otkaza, a uz pomoć rekonfiguracije, zadrži ili ponovo uspostavi operativni status sistema.

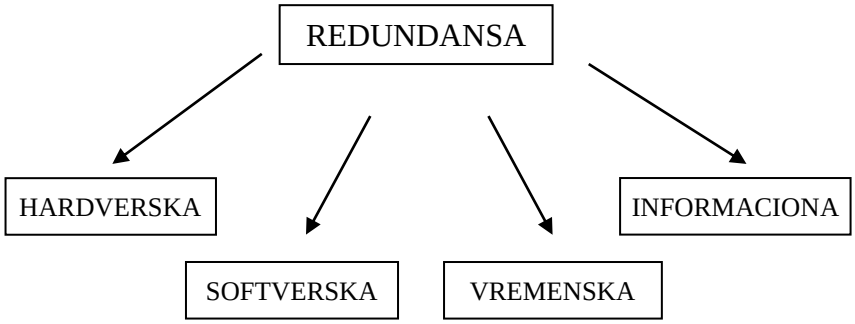
Na Sl. 3-1 prikazana je uzročno-posledična veza između otkaza, greške i kvara, Prikazana je i priroda otkaza, kao i osnovni uzroci njihovog nastanka otkaza. Date su i tehnike za eliminaciju otkaza, pri čemu je jasno naglašeno gde se koja tehnika može upotrebiti.



Sl. 3-1: *Uzroci otkaza i osnovne tehnike za njihovu eliminaciju*

3.6 Redundansa

Prevazilaženje greška kod FT RTS-a zasnovano je na korišćenju neke vrste redundanse, [Kor07], [Chr08], [Bar96]. Redundansa se može definisati kao dodatni resurs preko onog koji je potreban za normalno funkcionisanje sistema. Osnovni resursi RTS-a su računarski hardver, softver, vreme i informacija (podaci), pa se prema tome može i klasifikovati redundansa kao što je i prikazano na Sl. 3-2, [Jev04a].



Sl. 3-2: *Osnovni tipovi redundanse*

3.6.1 Hardverska redundansa

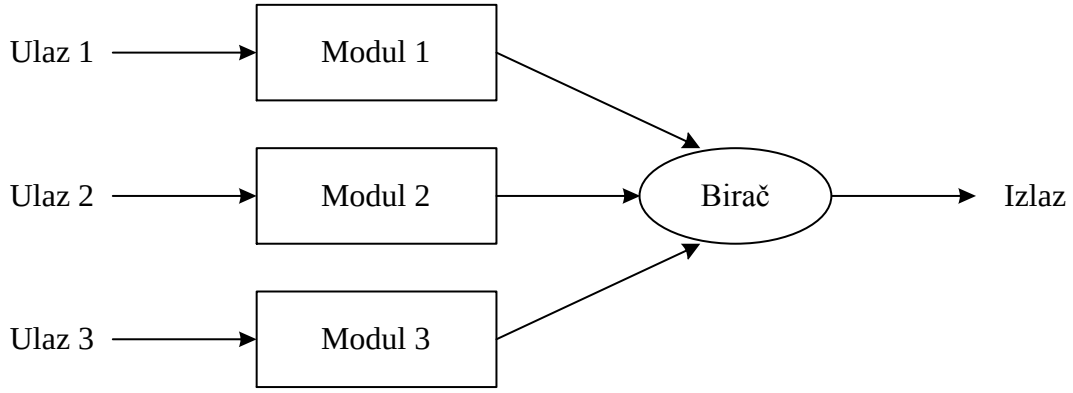
Hardverska redundansa podrazumeva postojanje dodatnog hardvera sa svrhom detektovanja i/ili tolerancije otkaza. Fizičko umnožavanje hardvera je najčešći oblik redundanse koji se danas koristi čemu značajno doprinosi i činjenica da poluprovodničke komponente postaju sve većeg stepena integracije i sve jeftinije. Prema načinu delovanja hardverske redundanse se dele na pasivnu, aktivnu i hibridnu.

3.6.1.1 Pasivna hardverska redundansa

Pasivne tehnike koriste princip maskiranja otkaza da sakriju pojavu otkaza i da spreče da otkaz uslovi pojavu greške. U ovom slučaju tolerancija grešaka se izvodi bez bilo kakve dodatne akcije oporavka sistema (npr. bez ponovnog izračunavanja, bez rekonfiguracije i sl.).

Osnovni koncept sistema sa pasivnom hardverskom redundansom predstavljen je na Sl. 3-3. Koncept se sastoji u utrostručavanju hardvera i primeni većinskog izglasavanja pri određivanju izlaza sistema.

Kao što se sa Sl. 3-3 može videti koriste se tri identična modula (trostruka modularna redundansa - TRM), koji sinhronizovano primaju isti signal kao ulaz i vrše istu obradu nad njima tj. izvršavaju iste operacije. Birač određuje izlaz na osnovu rezultata koje daju moduli. Šta će se naći na izlazu birača zavisi od primenjene strategije. Osnovna strategija određivanja izlaza je "većinsko izglasavanje" u okviru koje se porede binarne vrednosti izlaza modula i dva ista rezultata se prihvataju kao tačan. Ukoliko jedan modul otkaže a druga dva generišu korektne vrednosti, izlaz će i dalje imati korektnu vrednost, čime je otkaz maskiran. Druga strategija birača je izbor jedne od tri vrednosti i to ona koja je između preostale dve [Krs05]. To je neophodna strategija kada po prirodi informacije, ulazne vrednosti nisu jednake (recimo informacije od tri identična analognog senzora).



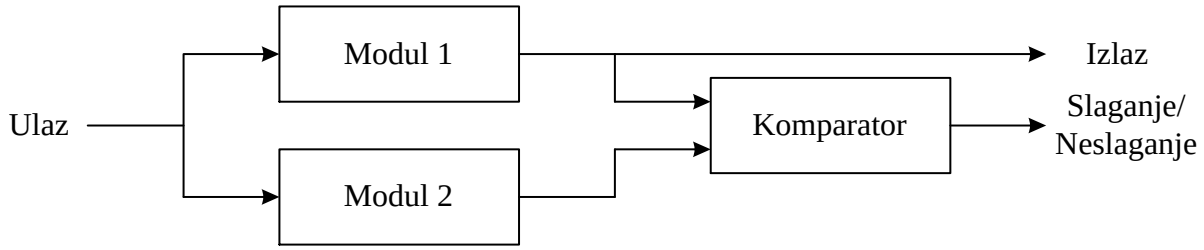
Sl. 3-3: *Osnovni koncept sistema sa pasivnom hardverskom redundansom*

3.6.1.2 Aktivna hardverska redundansa

Aktivne tehnike postižu toleranciju otkaza detektovanjem postojanja otkaza i izvođenjem akcija sa ciljem prevazilaženja greške i/ili uklanjanja dela hardvera sa otkazom iz sistema (rekonfiguracija). Rekonfiguaracija hardvera sistema nije uvek obavezna. U ovom slučaju nema maskiranja otkaza. Aktivnom hardverskom redundansom se mora obezbediti:

- detektovanje otkaza
- lociranje otkaza
- rekonfiguracija sistema ako je potrebno i oporavak na normalan status.

Klasične tehnike aktivne hardverske redundanse su udvostručavanje sistema sa kompariranjem rezultata čiji je osnovni princip prikazan na Sl. 3-4.



Sl. 3-4: *Tehnika udvostručavanja sistema sa kompariranjem rezultata*

Osnova ove tehnike je da se prilikom detekcije različitih rezultata inicira procedura samoprovere i prevazilaženja problema. Kod ove tehnike je problem što redundansa samo detektuje da u jednom od sistema - modula postoji greška tj. ne postoji način za određivanje koji je od dva modula je u kvaru. Osim ove tehnike postoji i veći broj tehnika koje se ne oslanjaju samo na hardver, a koje hardverskom redundansom omogućavaju detektovanje otkaza ili greške. Druga mogućnost je da udvostručena redundansa služi kao rezerva, pa pri detektovanju otkaza, naročito stalnih, rezervni sistem automatski preuzima dalji rad.

3.6.1.3 Hibridna hardverska redundansa

Hibridna tehnika predstavlja kombinaciju pasivne i aktivne tehnike. Od pasivne tehnike koristi maskiranje otkaza, za skrivanje pojave otkaza i za prevenciju generisanja pogrešnih rezultata. Onda kada maskiranje nije više u stanju da obezbedi ispravan rad sistema, primenjuje se neka aktivna tehnika kako bi se otkaz locirao i sanirao zamenom pokvarenog hardvera rezervnim. Ovim se osigurava veoma visoka pouzdanost funkcionisanja sistema.

3.6.2 Softverska redundansa

Softverska redundansa predstavlja dodavanje softvera, iznad onog neophodnog za obavljanje zadatih funkcija, a u cilju detekcije i tolerisanja grešaka.

Osnovna tehnika čiste softverske redundanse za detekciju i maskiranje otkaza u softveru je N verzija programa. Ova tehnika je slična pasivnoj hardverskoj redundansi i kod nje postoje najmanje tri verzije softvera (obično razvijene od tri nezavisna tima za razvoj softvera) koje rade paralelno. Ovo znači da je za svaku verziju softvera potreban poseban procesor. Ako se pak radi sa jednim procesorom onda je neophodno obezbediti veliku

vremensku redundansu. Nakon izvršene obrade svaka verzija softvera daje svoj rezultat, a jedan od procesora vrši poređenje i većinskim glasanjem daje konačan rezultat.

Zajedno sa redundansama u ostalim domenima, redundansa u softveru može biti i u vidu nekoliko dodatnih naredbi ili u vidu male programske procedure u kojima se realizuje neka tehnika za detekciju i prevazilaženje grešaka.

3.6.3 Informaciona redundansa

Informaciona redundansa predstavlja dodavanje informacija iznad zahtevanih za izvršavanje zadate funkcije. Dobri primeri informacione redundanse su kodovi za detekciju, kao i kodovi za detekciju i korekciju grešaka, koji se formiraju dodavanjem redundantnih bitova rečima, ili prevođenjem reči u neki novi oblik koji sadrži redundante informacije.

Generalno, kod može da ispravi do r pogrešnih bitova i detektuje do e dodatnih pogrešnih bitova ako i samo ako je

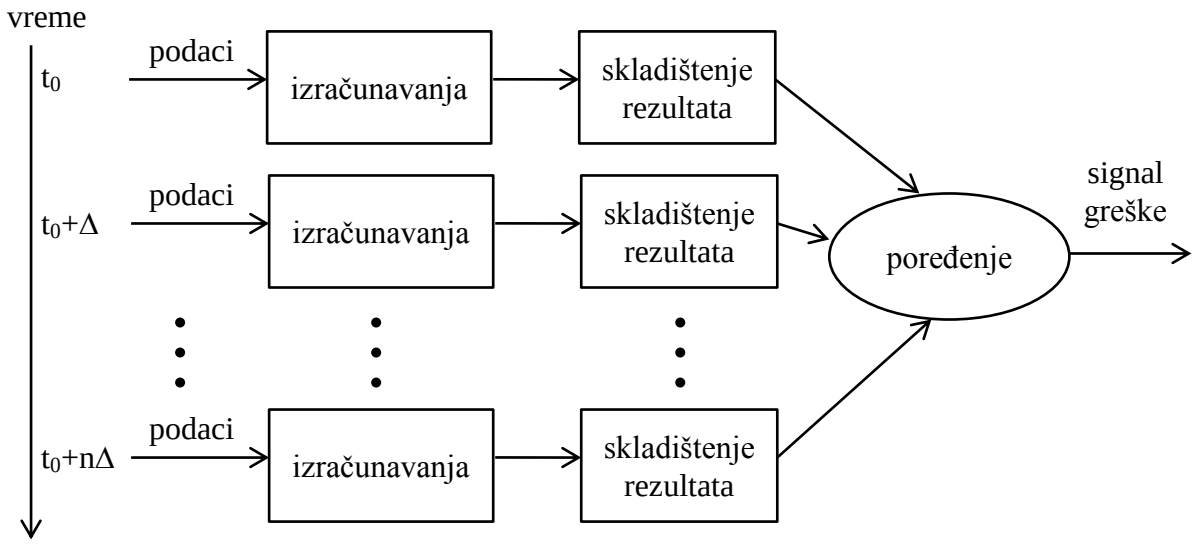
$$2r + e + 1 \leq H,$$

gde je H Hemingovo kodno rastojanje (tj. broj bitskih pozicija u kojima se bilo koje dve kodne reči razlikuju). Tako, prosti kodovi parnosti ($H = 2$) obezbeđuju detekciju bilo koje jednobitne greške i sve višestruke greške sa neparnim brojem bitskih pozicija na kojima je došlo do promene.

3.6.4 Vremenska redundansa

Vremenska redundansa podrazumeva korišćenje redundantnog vremena u radu sistema.Osnovni princip vremenske redundanse sastoji se u ponavljanju izračunavanja na način koji omogućava detekciju otkaza Sl. 3-5. Naime, ista izračunavanja izvršavaju se dva ili više puta i upoređuju rezultati u cilju određivanja eventualnog neslaganja. Ako se detektuje greška, možemo ponoviti izračunavanje da bi videli da li neslaganje ostaje, ili nestaje. Ova tehnika je posebno pogodna za detekciju grešaka nastalih usled prolaznog otkaza. Korišćenje vremenske redundanse na prethodno opisan način omogućava ne samo detekciju već i korekciju prolaznih grešaka. Međutim iznos vremenske redundanse koji je potreban za višestruko ponovljeno izračunavanje može biti neprihvatljivo veliki. Drugi pristup se sastoji u tome da se za detekciju greške koristi neki drugi mehanizam, a da se vremenska redundansa

koristi za oporavak sistema nakon detektovane greške. Oporavak RTS-a može biti ponovnim izvršavanjem zadatka, izvršavanjem alternativnog zadatka, ili čak restartovanjem aplikacije.



Sl. 3-5: *Osnovni princip vremenske redundanse*

Tehnike koje koriste vremensku redundansu ne zahtevaju dodatni hardver, relativno su jeftine i posebno pogodne za primenu kod aplikacija gde postoje ograničenja vezana za težinu, veličinu i potrošnju.

U ovoj disertaciji pažnja će prevashodno biti usmerena na tehnike koje koriste vremensku redundansu za izvršavanje funkcija oporavka sistema nakon nastanka otkaza. Ove tehnike se posebno primenjuju za prevazilaženje prolaznih otkaza, [Kan03], [Đoš11b].

3.7 Detekcija grešaka

Da bi se postigla pouzdanost u radu RTS-a neophodno je uspešno izvršiti proceduru tolerisanja otkaza, odnosno neophodno je prvo detektovati grešku koja predstavlja uzrok otkaza. U osnovi, detektori otkaza zasnovani su na korišćenju redundanse u hardveru, informaciji, softveru i/ili vremenu. Uzimajući u obzir mesto gde je u RTS-u detektor otkaza/greške primenjen, detektore grešaka možemo podeliti na tri nivoa [Jev04]:

- nivo kola (ili nivo zapisa informacija),
- nivo sistema (ili funkcionalni nivo), i

- nivo aplikacije.

Detekcija grešaka nije tema ove disertacije tako da će samo u najkraćim crtama biti objašnjene najčešće korišćene tehnike.

3.7.1 Nivo kola

Mehanizmi detekcije greške na nivou kola uvode se korišćenjem:

- kodne sekvence za detekciju greške,
- kola za samoproveru i
- kola sa komplementarnim udvostručavanjem.

Kodne sekvence za detekciju greške predstavljaju specifične prezentacije simbola koje omogućavaju detekciju grešaka unutar kodne reči. Jedna od tehnika karakteristična za nivo kola je tehnika nazvana *memory protection codes* odnosno kodovi za zaštitu memorije. Ova tehnika omogućava detekciju grešaka nastalih u RAM memoriji pod uticajem spoljašnjih smetnji upotrebom tzv. EDAC (*error detection and correction code*) kodova [Shi00]. U najjednostavnijem obliku, ova tehnika zahteva proširenje svake memorijske reči jednim bitom koji se koristi za čuvanje bita parnosti memorisanog podatka. Dodavanjem većeg broja redundantnih bita i primenom odgovarajućih EDAC kodova, moguća je ne samo detekcija već i korekcija grešaka na ograničenom broju bita memorisanog podatka. EDAC kodovi štite pojedinačno svaki memorijski blok kako bi se izbegla propagacija greške.

Kola za samoproveru (eng. *Self-Checking Circuits*) su digitalna kola (interne jedinice u čipu, čip, ploča, sistem) koja su u stanju da autonomno (bez korišćenja spoljašnjih test signala) detektuje grešku, odmah nakon njenog pojavljivanja, [Met00], [Djo05].

Kola sa komplementarnim udvostručavanjem sadrže module koji prilikom rada koriste pozitivnu logiku i module koji prilikom rada koriste negativnu logiku. Ako moduli ispravno funkcionišu izlazi će im biti komplementarni, u suprotnom detektuje se greška [Kar12].

Detektori na nivou kola su efikasni za neke sistemske blokove kao što su memorije ili neki komunikacioni podsistemi. Za ostale blokove odnos cena/pokrivanje grešaka može biti nezadovoljavajući za mnoge primene, tako da je dosta pažnje usmereno ka detektorima na nivou sistema [Jev09b].

3.7.2 Nivo sistema

Mehanizmi detekcije otkaza na nivou sistema proveravaju pogrešne aktivnosti ili informacije na višem nivou nego prethodni. Oni verifikuju rad sistema funkcionalnim potvrđivanjem kontrole toka programa, pristupa memoriji itd. Dakle, ovi detektori verifikuju ispravnost rada sistema proverom njegovih različitih opštih svojstava. Najefikasniji detektori na nivou sistema koriste hardversko uvišestručavanje i vremensku redundansu.

Jedna od tehnika karakteristična za nivo sistema je tehnika duplikacije (eng. d*uplication*) kod koje se detektuje greška ako su rezultati dobijeni od dupliranih entiteta različiti [Gom06]. Duplirani entiteti obavljaju istu operaciju nad istim operandima i u normalnim uslovima generišu isti rezultat. Nepodudarnost rezultata ukazuje na pojavu greške u jednom od dupliranih entiteta. Primeri dupliranih entiteta su duplirane hardverske jedinice, softverske instrukcije, funkcije, pozivi procedura, ili celi procesi. Tehnika duplikacije najčešće se primenjuje zajedno sa još nekom tehnikom za detekciju, kako bi se povećala verovatnoća otkrivanja grešaka.

Još jedna tehnika karakteristična za nivo sistema je *watchdog* tehnika koja omogućava periodičnu detekciju grešaka tj. proveru toka izvršenja programa ili podataka koji se prenose [Ben03]. Najjednostavnija *watchdog* šema je *watchdog timer* odnosno nadzorni tajmer koji nadzire vreme izvršenja segmenta programskog koda, RT zadatka ili neke aktivnosti i reaguje kada je ono van dozvoljenih granica [Mir95]. Kod najjednostavnijih mikroračunara nadzorne tajmere restartuju programske sekvence koje su ugrađene u aplikacionom programu, a koje periodično reinicijalizuju tajmere. Ako nisu reinicijalizirani (zbog greške u izvršavanju programa) unutar specificiranog vremena, nadzorni tajmeri generišu signale greške. Signal obično aktivira nemaskirani prekid mikroprocesora izazivajući restartovanje sistema. *Watchdog* može biti implementiran ili u hardveru kao poseban procesor [Mah88] ili u softveru kao poseban test program.

3.7.3 Nivo aplikacije

Mehanizmi detekcije grešaka na nivou aplikacije su prilagođeni algoritmima koji su realizovani u sistemu i svojstvima generisanih rezultata (prihvatljivi opseg promenljivih, razumljivost rezultata izračunavanja i sl.). Jedna od tehnika karakteristična za nivo primene je tehnika pod nazivom a*ssertions* [Pet05]. *Assertions* (tj. tvrđenja ili asercije) su logički iskazi

koji se u obliku „*if not* <asercija> *then* <greška>“ umeću na pojedinim tačkama u programu radi provere vrednosti promenljivih ili ispravnosti rezultata pojedinih operacija. Na primer, asercija <a = 1> znači da u datoj tački programa promenljiva a mora imati vrednost 1; svaka druga vrednost ukazuje na pojavu greške. U poređenju sa drugim tehnikama, asercije mogu biti veoma efikasno sredstvo za detekciju grešaka. Međutim, upotreba asercija pretpostavlja dubinsko poznavanje toka izračunavanja i karakteristika konkretne primene, a njihovo pravilno raspoređivanje u programu traži veštinu i iskustvo programera. Takođe, veliki broj asercija u programu dovodi do pada performansi i u najgorem slučaju do prekoračenja krajnjeg roka.

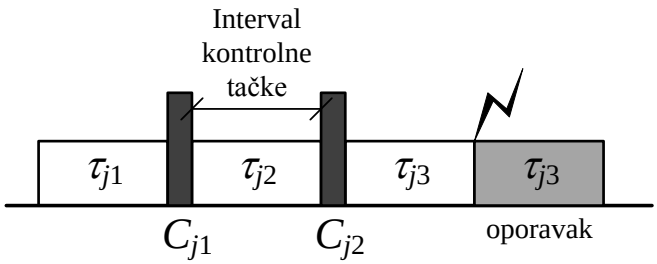
3.8 Tehnike za prevazilaženje otkaza koje koriste vremensku redundansu

Dve tehnike koje koriste vremensku redundansu za prevazilaženje prolaznih otkaza i koje se najčešće koriste kod FT RTS-a su:

- tehnika kontrolnih tačaka (eng. *checkpointing technique)*, i
- tehnika ponovljenog izvršenja (eng. *re-execution technique*).

3.8.1 Tehnika kontrolnih tačaka

Tehnika kontrolnih tačaka sastoji se u umetanju određenog broja kontrolnih tačaka (eng. *checkpoint*) u tok izvršenja RT zadatka, [Pun97]. Vremenski period između izvršenja dve kontrolne tačke naziva se interval kontrolnih tačaka (eng. *checkpoint interval*). Tokom izvršenja zadatka, svaki zadatak čuva informacije o svom statusu nakon svakog intervala kontrolnih tačaka, odnosno u svakoj kontrolnoj tački. Ako se greška javi tokom izvršenja zadatka, prevazilazi se tako što se deo zadatka ponovo izvršava počev od prethodne najskorije kontrolne tačke. Takođe, status zadatka se vraća na stanje zapamćeno u kontrolnoj tački od koje se ponavlja izvršenje. Na Sl. 3-6 ilustrovana je tehnika kontrolnih tačaka za slučaj RT zadatka τ_j koji ima dve kontrolne tačke. Kontrolne tačke C_{j1} i C_{j2} dele izvršenje zadatka τ_j na tri dela τ_{j1}, τ_{j2} i τ_{j3}. Ako pretpostavimo da se greška javila neposredno pred kraj izvršenja zadataka τ_j, u cilju njegovog oporavka ponavlja se izvršenje samo dela τ_{j3}, a ne celokupnog zadatka. Jasno je da je sa stanovišta brzine prevazilaženja grešaka poželjno da kontrolne tačke budu gušće raspoređene. Međutim, veliki broj kontrolnih tačaka dovodi do povećanog utroška procesorskog vremena i memorije.



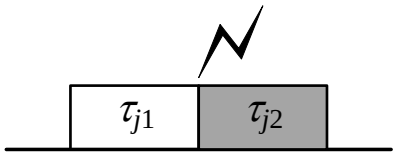
Sl. 3-6: *Tehnika kontrolnih tačaka*

Tehnika kontrolnih tačaka je često obrađivana u literaturi [Zhs03], [Ziv97], [Kwa01], [Shi87], [Bow93]. Shin i ostali predlažu dva analitička modela za projektovanje i procenu optimalnog broja kontrolnih tačaka u jednom RTS-u, [Shi87]. Kako se tehnika kontrolnih tačaka prilagođava karakteristikama pojedinih procesora tema je u [Bow93]. Ideja da se kontrolne tačke nalaze na mestima gde se troši najmanje vremena za pamćenje statusa zadatka razmatrana je u [Ziv97]. Model tehnike kontrolnih tačaka zasnovan na ideji da se kontrolne tačke rasporede tako da intervali kontrolnih tačaka budu jednaki, obrađen je u [Kwa01].

3.8.2 Tehnika ponovljenog izvršenja

Osnova tehnike ponovljenog izvršavanja zadatka (eng. *task re-executon technique*) sastoji se u ponovljenom izvršavanju celokupnog zadatka u toku čijeg izvršenja je detektovana greška sa ciljem njenog prevazilaženja. U okviru ove tehnike sistem nakon detekcije greške vraća sve parametre konkretnog zadatka na početnu vrednost tj. na vrednost pre njegovog izvršenja, a zatim ga ponovo izvršava. Treba napomenuti da detekcija može biti na kraju izvršenja zadatka ili u više tačaka tokom izvršenja zadatka, i da bez obzira u kojoj tački se detektuje greška, zadatak se uvek ponovo izvršava od početka. Na taj način sistem prevazilazi nastalu grešku.

Tehnika ponovljenog izvršavanja zadatka ilustrovana je na Sl. 3-7 gde je prikazan RT zadatak τ_{j1} u toku čijeg izvršenja je detektovana greška. Da bi se nastala greška prevazišla neophodno je ponovno izvršiti zadatka što je i predstavljeno zadatkom τ_{j2}. Prednosti tehnike ponovljenog izvršavanja zadatka u odnosu na tehniku kontrolnih tačaka su manji troškovi i jednostavnija realizacija, a mane što ne postoji kontrola nad vremenom oporavka, već zavisi od vremena izvršenja (obimnosti) zadataka.

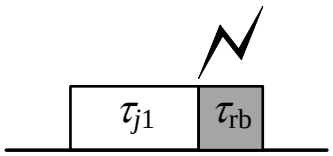


Sl. 3-7: *Ponovljeno izvršenje RT zadatka*

Ponovljeno izvršavanja zadatka je često korišćeno u literaturi kao osnova za razvoj tehnika za prevazilaženje prolaznih otkaza kod FT RTS-a, [Kan03], [Pop07], [Gho95], [Bur96], [Izo08]. Ghosh, Melhem i Mossé bavili su se istraživanjima vezanim za prevazilaženje prolaznih otkaza kod aperiodičnih RT zadataka i koristili su ovu tehniku za oporavak sistema, [Gho95]. Burns, Davis i Punnekkat analizirali su izvodljivost RT zadataka sa fiksnim prioritetom i koristili tehniku ponovljenog izvršavanja zadatka za prevazilaženje otkaza, [Bur96].

Varijanta tehnike ponovljenog izvršavanja zadataka je tehnika izvršenja alternativnog zadatka (eng. *recovery block*) koja se ogleda u izvršavanju nekog alternativnog zadatka nakon detekcije greške, a sa ciljem oporavka sistema. Najčešće se alternativni zadatak razlikuje od originalnog po vremenu izvršenja i može predstavljati drugačiju implementaciju zadatka u toku čijeg izvršenja je detektovana greška. Takođe, alternativni zadatka može biti i zadatak koji ima za cilj da sistem dovede u neko stabilno, odnosno sigurno stanje.

Tehnika izvršenja alternativnog zadatka ilustrovana je na Sl. 3-8 gde je prikazan RT zadatak τ_{j1} u toku čijeg izvršenja je detektovana greška. Da bi se nastala greška prevazišla neophodno je izvršiti alternativni zadatak τ_{rb} koji u prikazanom slučaju ima kraće vreme izvršenja.



Sl. 3-8: *Izvršenje alternativnog zadatka*

Tehnika izvršenja alternativnog zadatka je često korišćena u literaturi kao izbor za toleranciju prolaznih otkaza, [Bur96], [Han03]. Burns, Davis i Punnekkat analizirali su izvodljivost RT zadataka i upoređivali performanse kada se za prevazilaženje otkaza koristi tehnika ponovljenog izvršavanja zadatka i kada se koristi tehnika izvršenja alternativnog

zadatka, [Bur96]. Mogućnost da svaki RT zadatka ima samo po jedan alternativni pomoću koga se mogu prevazići prolazni otkazi, razmatrana je u [Han03].

U disertaciji će akcenat prevashodno biti na tehnikama ponovljenog izvršavanja zadataka sa ciljem prevazilaženja prolaznih otkaza.

4 Analiza vremena odziva

Analiza vremena odziva (eng. *Response Time Analysis* - RTA) predstavlja tehniku za analizu rada RTS-a sa stanovišta izvodljivosti zadataka. RTA omogućava izračunavanje vremena odziva RT zadataka kod jedno ili više procesorskih RTS-a koji koriste algoritme za raspoređivanje zadataka sa fiksnim prioritetom. Drugim rečima, koristeći RTA-a moguće je za date vremenske parametre zadataka RTS-a ispitati da li RTS može ispravno da funkcioniše ili ne.

Prva istraživanja vezana za RTA primenjivana su kod RTS-a kod kojih se ne uzima u obzir mogućnost pojave greške [Aud91], [Aud93], [Jos96]. U kasnijim istraživanjima se uzima u obzir činjenica da „*sistemi koji rade bez greške ne postoje, postoje samo sistemi kod kojih se greška još uvek nije pojavila"* [Lap92], što je uslovilo i modifikaciju analize vremena odziva, [Bur96], [Đoš09], [Lim03], [Bin04]. Ovako modifikovana RTA koristi vremensku redundansu za ponovljeno izvršenje zadatka u toku čijeg izvršenja se greška javila i na taj način obezbeđuje ispravno funkcionisanje RTS-a. Još jedna primena modifikovane RTA je i procena mogućnosti tolerisanja grešaka kod RTS-a, [Đoš11a].

RTA se često pominje i u radovima novijeg datuma vezanim za pouzdane RTS-e [Mub12], [Alt12], [Yun 12]. U [Mub12] je predstavljena integracija RTA u već postojeću metodu namenjenu razvoju distribuiranih RTS-a. Altmeyer i ostali u [Alt12] proširuju jednačinu RTA razmatrajući RT zadatke čije izvršenje može biti prekinuto usled pojave zahteva za izvršenje zadatka sa većim prioritetom. U [Yun 12] je osnovna RTA jednačina modifikovana za primenu kod višeprocesorskih RTS-a.

U ovom poglavlju predstavljena je RTA u osnovnom obliku i u obliku prilagođenom za slučaj RTS-a kod koga je moguća pojava grešaka u toku rada sistema. Glavni doprinosi ove disertacije, obuhvaćeni ovim poglavljem su razvijeni algoritam za verifikaciju ispravnog rada RTS-a za dati vremenski interval između moguće dve uzastopne greške, kao i modifikacije RTA za specijalne slučajeve *hard* RTS-a kod kojih je neophodno obezbediti visoku verovatnoću prevazilaženja grešaka. Takođe je predstavljena analiza rada RTS-a za različite planere i načine oporavka sistema.

4.1 Primena RTA kod RTS-a kod kojih se ne uzima u obzir mogućnost pojave greške

Pretpostavimo da postoji skup od n RT zadataka, $\Gamma=\{\tau_1,...,\tau_n\}$, pri čemu je za svaki zadatak τ_i poznat period pojavljivanja T_i, rok za izvršenje D_i ($D_i \leq T_i$) kao i vreme izvršenja zadatka C_i. Pretpostavimo još da postoji i n nivoa prioriteta (1,2,...,n), pri čemu je sa p_i označen prioritet zadatka τ_i gde 1 označava najviši prioritet. Zadaci pretpostavljenog RTS-a su takvi da je njihovo izvršenje moguće prekinuti radi izvršenja zadatka višeg prioriteta, [Doš05], [Doš10a].

Zadacima se dodeljuju prioriteti u skladu sa nekim od standardnih algoritama za raspoređivanje zadataka, kao što su RM algoritam, DM algoritam ili neki drugi algoritam za dodeljivanje prioriteta, [Cot02], [Doš05]. RTA ne zavisi od izabranog algoritma za raspoređivanje zadataka tako da nema ograničenja koja se odnose na izbor algoritama rada sistema što je čini veoma pogodnom za široku primenu.

Za pretpostavljeni RTS, kod koga se ne uzima u obzir mogućnost pojave greške, RTA se može predstaviti jednačinom (4-1):

$$R_i = C_i + I_i \tag{4-1}$$

gde je sa R_i označeno vreme odziva zadatka τ_i koje se definiše kao vreme koje protekne od trenutka kada se javi zahtev za izvršenje zadatka τ_i do trenutka kada se zadatak τ_i kompletno izvrši. Vreme odziva R_i predstavljeno je u jednačini (4-1) kao suma vremena izvršenja C_i zadatka τ_i i interferencije I_i koja je posledica pojave zahteva za izvršenje zadataka sa većim prioritetom od prioriteta zadatka τ_i. Jednačina (4-1) detaljnije se može napisati kao:

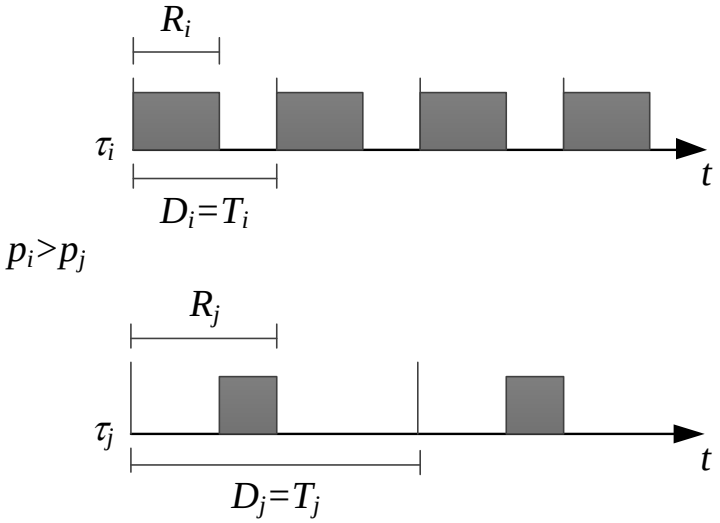
$$R_i = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{4-2}$$

gde je detaljnije prikazan drugi sabirak odnosno interferencija I_i. U drugom sabirku, član $\left\lceil \frac{R_i}{T_j} \right\rceil$ označava koliko se puta u toku R_i može javiti zadatak τ_j dok suma predstavlja ukupno vreme potrebno za izvršenje svih zadataka koji imaju prioritet veći od p_i, tj. zadataka iz skupa hp(i) = $\{\tau_j \in \Gamma \mid p_j > p_i\}$, [Aud91], [Aud93], [Jos96]. Ako je moguće naći vrednost R_i takvu da $R_i \in [0, D_i]$ i da zadovoljava jednačinu (4-2) onda je zadatak τ_i moguće rasporediti u skladu sa izabranim algoritmom. Najmanja vrednost R_i koja zadovoljava jednačinu (4-2) predstavlja vreme odziva zadatka τ_i u najgorem slučaju.

Kako se R_i nalazi sa obe strane jednačine (4-2), do rešenja se može doći iterativnim postupkom predstavljenim jednačinom (4-3):

$$R_i^{n+1} = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{4-3}$$

gde za početnu vrednost R_i treba uzeti da je $R_i^0 = C_i$. Ovaj iterativni proces se završava u iteraciji kad je $R_i^{n+1} = R_i^n$ (određeno je vreme odziva zadatka τ_i u najgorem slučaju) ili kada je $R_i^{n+1} > D_i$ (slučaj kada se ne može garantovati da će zadatak τ_i biti izvršen pre svog roka).



Sl. 4-1: *Vreme odziva dva RT zadatka*

Na Sl. 4-1 ilustrovana su vremena odziva R_i i R_j dva periodična RT zadatka τ_i i τ_j za slučaj da je period pojavljivanja zadatka jednak roku za njihovo izvršenje, tj. da je $D_i = T_i$ i $D_j = T_j$ i da je prioritet zadatka τ_i veći od prioriteta zadatka τ_j tj. da je $p_i > p_j$. Vremena odziva R_i i R_j zadataka τ_i i τ_j izračunata su u skladu sa jednačinom (4-3). Kao što se može videti, vreme odziva zadataka τ_i se podudara sa vremenom njegovog izvršenja. To je zato što je prioritet zadatka τ_i veći od prioriteta zadatka τ_j. Vreme odziva zadatka τ_j obuhvata ne samo vreme potrebno za njegovo izvršenje već i vreme potrebno za izvršenje zadatka τ_i kao zadatka sa većim prioritetom.

4.1.1 Primer RTA za slučaj RTS-a bez pojave grešaka

Kod RTS-a kod koga se ne uzima u obzir mogućnost pojave greške, RTA se može ilustrovati jednostavnim primerom za slučaj četiri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i, prioritet zadatka p_i) dati u Tab. 4-1. Na osnovu jednačine (4-3) dolazi se do vrednosti za vremena odziva zadataka R_i koja su takođe prikazana u Tab. 4-1. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms.

Tab. 4-1: *Primer I*

Zadaci	Parametri zadataka				
	C_i	T_i	D_i	p_i	R_i
τ_1	30	100	100	1	30
τ_2	35	175	175	2	65
τ_3	25	200	200	3	90
τ_4	30	300	300	4	150

Na osnovu dobijenih vrednosti za R_i, prikazanih u Tab. 4-1, može se zaključiti da će se svi zadaci izvršiti pre predviđenog roka jer sve vrednosti parametra R_i zadovoljavaju uslov $R_i \in [0, D_i]$. Predstavljeni RTS može uspešno da funkcioniše. Treba naglasiti da vrednosti R_i ne predstavljaju egzaktna vremena odziva, već vremena odziva u najnepovoljnijem slučaju. Realno vreme odziva zadatka τ_i može da varira iz periode u periodu, zavisno od trenutnog stanja aktivnosti ostalih zadataka, ali ono nikada neće biti duže od R_i.

Ilustracije radi, dati su koraci računanja vremena odziva R_4 zadatka τ_4 koji su dobijeni u skladu sa jednačinom (4-3):

$$R_4^0 = 30$$

$$R_4^1 = 30 + \left\lceil \frac{30}{100} \right\rceil 30 + \left\lceil \frac{30}{175} \right\rceil 35 + \left\lceil \frac{30}{200} \right\rceil 25 = 120$$

$$R_4^2 = 30 + \left\lceil \frac{120}{100} \right\rceil 30 + \left\lceil \frac{120}{175} \right\rceil 35 + \left\lceil \frac{120}{200} \right\rceil 25 = 150$$

$$R_4^3 = 30 + \left\lceil \frac{150}{100} \right\rceil 30 + \left\lceil \frac{150}{175} \right\rceil 35 + \left\lceil \frac{150}{200} \right\rceil 25 = 150$$

$$R_4^3 = R_4^2 \Rightarrow R_4 = 150$$

Kao što se može videti konkretna računica nije kompleksna, jer se do konačne vrednosti vremena odziva R_4 dolazi nakon četiri, gde svaka iteracija obuhvata po tri operacije deljenja i množenja i sabiranja. Međutim, za veći skup RT zadataka, broj iteracija kao i računica tokom svake iteracije može biti značajno složenija. Iz tog razloga, a i za potrebe ove disertacije, realizovan je program za izračunavanje vremena odziva u okviru softverskog paketa Matlab čiju osnovu predstavlja jednačina (4-3).

4.2 Primena RTA kod RTS-a sa pojavom grešaka koje se mogu prevazići

Pojava grešaka u toku rada RTS-a predstavlja realnu mogućnost. Prilikom projektovanja elektronskih sistema, a pogotovo RTS-a, imperativ je obezbediti da sistem prevaziđe grešku pri čemu oporavak od greške ne sme da remeti vremenska ograničenja sistema. Akcenat disertacije je na prolaznim otkazima tako da se u ovoj sekciji pretpostavlja pojava upravo ovog tipa otkaza odnosno grešaka kao njihovih posledica. Takođe, usvaja se da RTS koristi tehniku (baziranu na vremenskoj redundansi i karakterističnoj za prolazne otkaze) ponovljenog izvršenja zadatka kod koga je detektovana greška.

Budući da se za oporavak sistema troši izvesno vreme, za primenu RTA kod RTS-a kod kojih je moguća pojava grešaka, osnovna jednačina (4-1) se mora modifikovati i njen novi oblik predstavljen je jednačinom (4-4), [Bur96], [Đoš09], [Lim03]:

$$R_i = C_i + I_i + F_i \tag{4-4}$$

gde treći sabirak F_i predstavlja vreme potrebno za oporavak sistema usled moguće pojave greške. Detaljnije jednačina (4-4) može se predstaviti kao:

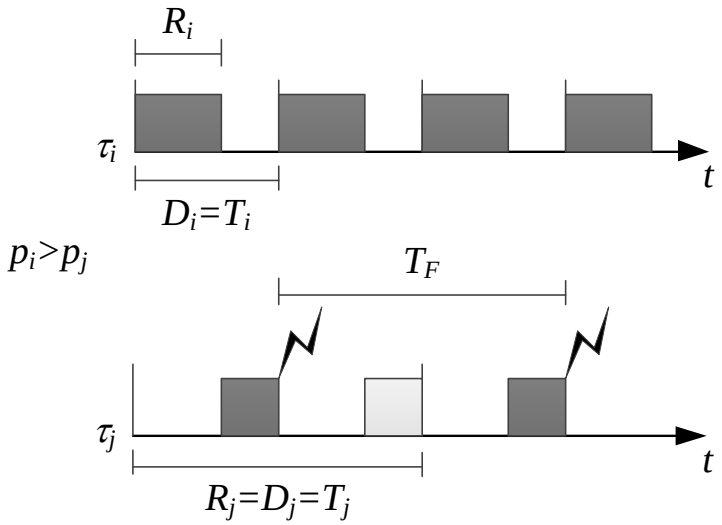
$$R_i = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_F} \right\rceil \max_{j \epsilon hp(i) \cup i} (C_j) \tag{4-5}$$

gde se vreme odziva R_i zadatka τ_i računa kao suma vremena izvršenja zadatka τ_i, vremena potrebnim za izvršenje svih zadataka koji imaju prioritet veći od p_i i vremena neophodnog za oporavak RTS-a usled pojave greške (treći sabirak).

U trećem sabirku novi parametar koji se javlja, a vezan je za mogućnost pojave greške, je parametar označen sa T_F. T_F predstavlja minimalno vreme koje može proteći između moguće pojave dve uzastopne greške. U okviru trećeg sabirka sa $\left\lceil \frac{R_i}{T_F} \right\rceil$ je označen maksimalni broj grešaka koje se mogu javiti u intervalu R_i. Kako se te greške mogu javiti u toku izvršenja zadatka τ_i ili nekog zadataka sa većim prioritetom od p_i, onda za svaku grešku u vremenu odziva R_i treba dodati vreme $\max_{j \epsilon hp(i) \cup i}(C_j)$. Kako se i u ovom slučaju R_i nalazi sa obe strane znaka jednakosti i ova jednačina se rešava iterativnim postupkom opisanim jednačinom (4-6):

$$R_i^{n+1} = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j + \left\lceil \frac{R_i^n}{T_F} \right\rceil \max_{j \epsilon hp(i) \cup i} (C_j) \tag{4-6}$$

Slično kao i kod jednačina (4-3) i ovde za početnu vrednost R_i treba uzeti da je $R_i^0 = C_i$ dok vrednost T_F pretpostavlja ulazni parametar sistema. Iterativni proces se završava ili kad je $R_i^{n+1} = R_i^n$ (određeno je vreme odziva zadatka τ_i u najgorem slučaju) ili kada je $R_i^{n+1} > D_i$ (slučaj kada se ne može garantovati da će zadatak τ_i biti izvršen pre svog roka).



Sl. 4-2: *Vreme odziva dva RT zadatka kada je moguća pojave greške u RTS-u*

Pretpostavljeno je da se prva greška javila nešto pre kraja izvršenja zadatka τ_j i da se sistem oporavio ponovljenim izvršenjem zadatka u toku čijeg izvršenja se greška javila. Takođe je pretpostavljeno i minimalno vreme između pojave dve uzastopne pojave greške, T_F, takvo da nakon oporavka sistema nijedan od zadataka nije prekoračio svoj rok. Usled pojave greške u toku izvršenja zadatka τ_j i njenog prevazilaženja, povećalo se i vreme odziva zadatka τ_j.

4.2.1 Primer RTA za slučaj RTS-a sa pojavom grešaka koje se mogu prevazići

RTA za RTS kod koga je moguća pojava grešaka može se ilustrovati jednostavnim primerom za slučaj četiri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i, prioritet zadatka p_i) dati u Tab. 4-2. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. Pretpostavljeno je da je minimalno vreme između pojave dve uzastopne greške T_F = 300 vremenskih jedinica. Na osnovu jednačine (4-6) dolazi se do vrednosti za vremena odziva zadataka R_i koja su takođe prikazana u Tab. 4-2.

Tab. 4-2: *Primer II*

Zadaci	Parametri zadataka				T_F = 300
	C_i	T_i	D_i	p_i	R_i
τ_1	30	100	100	1	60
τ_2	35	175	175	2	100
τ_3	25	200	200	3	155
τ_4	30	300	300	4	275

Ilustracije radi, dati su koraci računanja vremena odziva R_3 zadatka τ_3 koji su dobijeni u skladu sa jednačinom (4-6):

$$R_3^0 = 25$$

$$R_3^1 = 25 + \left\lceil \frac{25}{100} \right\rceil 30 + \left\lceil \frac{25}{175} \right\rceil 35 + \left\lceil \frac{25}{200} \right\rceil 35 = 125$$

$$R_3^2 = 25 + \left\lceil \frac{125}{100} \right\rceil 30 + \left\lceil \frac{125}{175} \right\rceil 35 + \left\lceil \frac{125}{200} \right\rceil 35 = 155$$

$$R_3^3 = 25 + \left\lceil \frac{155}{100} \right\rceil 30 + \left\lceil \frac{155}{175} \right\rceil 35 + \left\lceil \frac{155}{200} \right\rceil 35 = 155$$

$$R_3^3 = R_3^2 \Rightarrow R_3 = 155$$

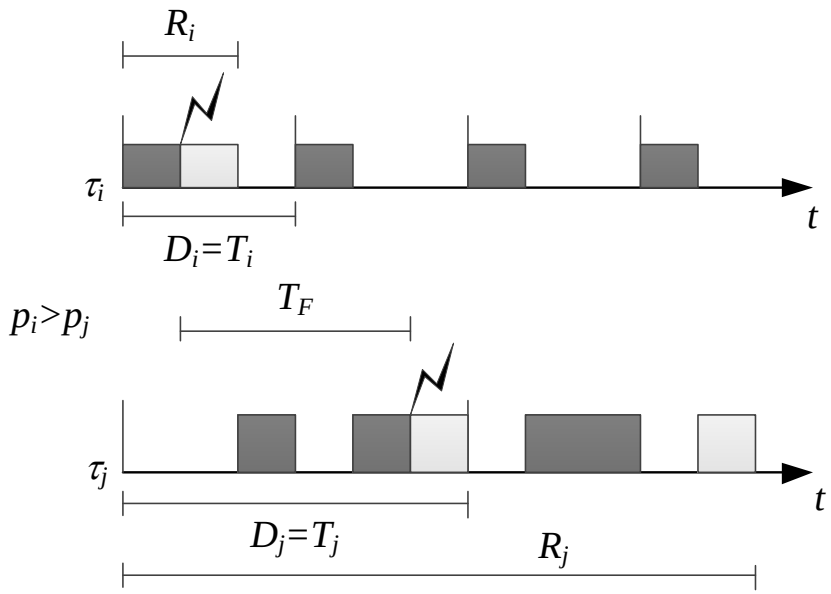
Na osnovu dobijenih vrednosti za R_i, prikazanih u Tab. 4-2, može se zaključiti da će se svi zadaci izvršiti pre predviđenog roka jer sve vrednosti parametra R_i zadovoljavaju uslov $R_i \in [0, D_i]$.

Kako su u Tab. 4-1 i Tab. 4-2 dati parametri RT zadataka identični mogu se prokomentarisati dobijene vrednosti za vremena odziva R_i. Povećanje vrednosti R_i u primeru II predstavljenom u Tab. 4-2 je posledica činjenice da je moguća pojava greške i da je neophodno obezbediti vremensku redundansu za oporavak RTS-a. U ovom slučaju do prekoračenja rokova zadataka neće doći sve dok je vreme između pojave dve greške duže od T_F. Konkretno, RTS definisan skupom zadataka prikazanim u Tab. 4-2 može uspešno da funkcioniše ukoliko je vreme između dve uzastopne moguće pojave greške 300 ili više vremenskih jedinica.

4.3 Primena RTA kod RTS-a sa pojavom grešaka koje se često javljaju

Problem koji se može javiti prilikom pretpostavke vrednosti vremena T_F jeste da je ta vrednost suviše mala tako da se nakon oporavka sistema javi prekoračenje roka za izvršenje jednog ili više zadataka.

Ovaj slučaj ilustrovan je na Sl. 4-3 za primer dva periodična RT zadatka τ_i i τ_j kod kojih je period pojavljivanja zadatka jednak roku za njihovo izvršenje tj. $D_i = T_i$ i $D_j = T_j$, prioritet zadatka τ_i veći od prioriteta zadatka τ_j tj. $p_i > p_j$ i kod kojih je moguća pojava greške u RTS-u. Pretpostavljeno je da se prva greška javila nešto pre kraja izvršenja zadatka τ_i i da se sistem oporavio ponovljenim izvršenjem zadatka τ_i u toku čijeg izvršenja se greška javila. Vreme odziva zadatka τ_i se usled oporavka sistema nakon pojave greške povećalo ali je i dalje ostalo u granicama $R_i \in [0, D_i]$ tako da nema prekoračenja roka za izvršenje zadatka τ_i. Nakon pretpostavljenog vremena T_F greška se opet javila i to neposredno pred kraj izvršenja zadatka τ_j što je uslovilo ponovljeno izvršenje tog zadatka u cilju prevazilaženja nastale greške u sistemu. Kao što se sa Sl. 4-3 može videti, zadatak τ_j prekoračuje svoj rok za izvršenje tj. $R_j > D_j$. Ovo je neprihvatljivo kod RTS-a i sistem u ovom slučaju nije uspeo da prevaziđe nastalu grešku.



Sl. 4-3: *Vreme odziva dva RT zadatka za slučaj RTS koji ne uspeva da se oporavi nakon pojave grešaka*

Ovo je ilustracija primera kada je pretpostavljeno vreme T_F suviše malo tj. greške se u sistemu javljaju suviše često i RTS ne uspeva da se oporavi i da nastavi sa normalnim funkcionisanjem.

4.3.1 Primer RTA za slučaj RTS-a sa pojavom grešaka koje se često javljaju

Problem sa Sl. 4-3, može se detaljnije analizirati na primeru četiri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i, prioritet zadatka p_i) dati u Tab. 4-3. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. Pretpostavljeno je da je minimalno vreme između pojave dve uzastopne greške T_F = 200 vremenskih jedinica. Na osnovu jednačine (4-6) dolazi se do vrednosti za vremena odziva zadataka R_i prikazanih u Tab. 4-3.

Tab. 4-3: *Primer III*

Zadaci	Parametri zadataka				T_F = 200
	C_i	T_i	D_i	p_i	R_i
τ_1	30	100	100	1	60
τ_2	35	175	175	2	100
τ_3	25	200	200	3	155
τ_4	30	300	300	4	**340**

Na osnovu dobijenih vrednosti za R_i, prikazanih u Tab. 4-3, može se videti da je $R_4 > D_4$ odnosno može se zaključiti da ne postoji garancija da zadatak τ_4 neće prekoračiti rok za izvršenje. Drugim rečima ako je vreme između dve uzastopne moguće pojave greške 200 vremenskih jedinica ne može se garantovati uspešno funkcionisanje RTS-a.

4.4 Algoritam za pronalaženje najmanje moguće vrednosti T_F za dati RTS

Treba istaći da se parametar T_F jednačine (4-6) može tretirati na dva načina.

Prvi, kada je T_F unapred zadati parametar sistema. U tom slučaju, T_F je konstanta u jednačini (4-6), a jednačina daje odgovor na pitanje da li RTS uspeva da ispuni sva vremenska ograničenja zadataka pod pretpostavkom da greške postoje, ali da vremenski

interval između dve uzastopne greške nikad nije kraći od T_F. U ovom slučaju T_F je karakteristika sistema.

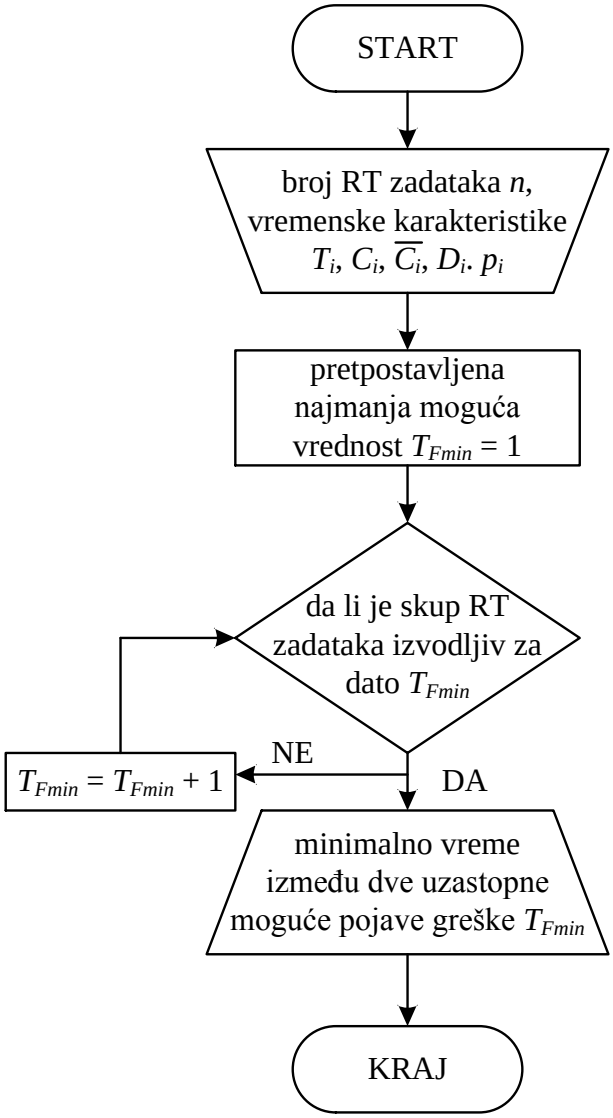
Drugi način je da se jednačina (4-6) koristi za određivanje minimalnog vremenskog perioda između dve uzastopne greške takvog da RTS može da prevaziđe moguće pojave grešaka i nastavi sa radom pod uslovom da sva vremenska ograničenja budu ispunjena. Ovaj parametar može se označiti sa T_{Fmin}. Ovako određeno T_{Fmin} takođe predstavlja karakteristiku sistema konkretnije meru otpornosti RTS-a na otkaze jer što je vrednost T_{Fmin} manja, to je sistem otporniji, i obrnuto.

Jedan od zadataka u okviru disertacije je posmatrati parametar T_F na ovaj drugi način tj. odrediti parametar T_{Fmin} odnosno odrediti najmanje vreme između moguće pojave dve uzastopne greške koje dati RTS može da prevaziđe i nastavi sa normalnim funkcionisanjem. Pretpostavljeno je da se greška može javiti u toku izvršenja bilo kog RT zadatka i da se RTS oporavlja nakon pojave grešaka ponovnim izvršavanjem tog istog zadatka ili izvršavanjem nekog alternativnog zadatka.

Na Sl. 4-4 prikazan je algoritam za izračunavanje vrednosti T_{Fmin}. Ulazni parametri prikazanog algoritma su broj RT zadataka n i njihove vremenske karakteristike: period pojavljivanja zadatka T_i, njegovo vreme izvršenja C_i, vreme izvršenja alternativnog zadatka $\overline{C_i}$, rok za izvršenje zadatka D_i i prioritet zadatka p_i.

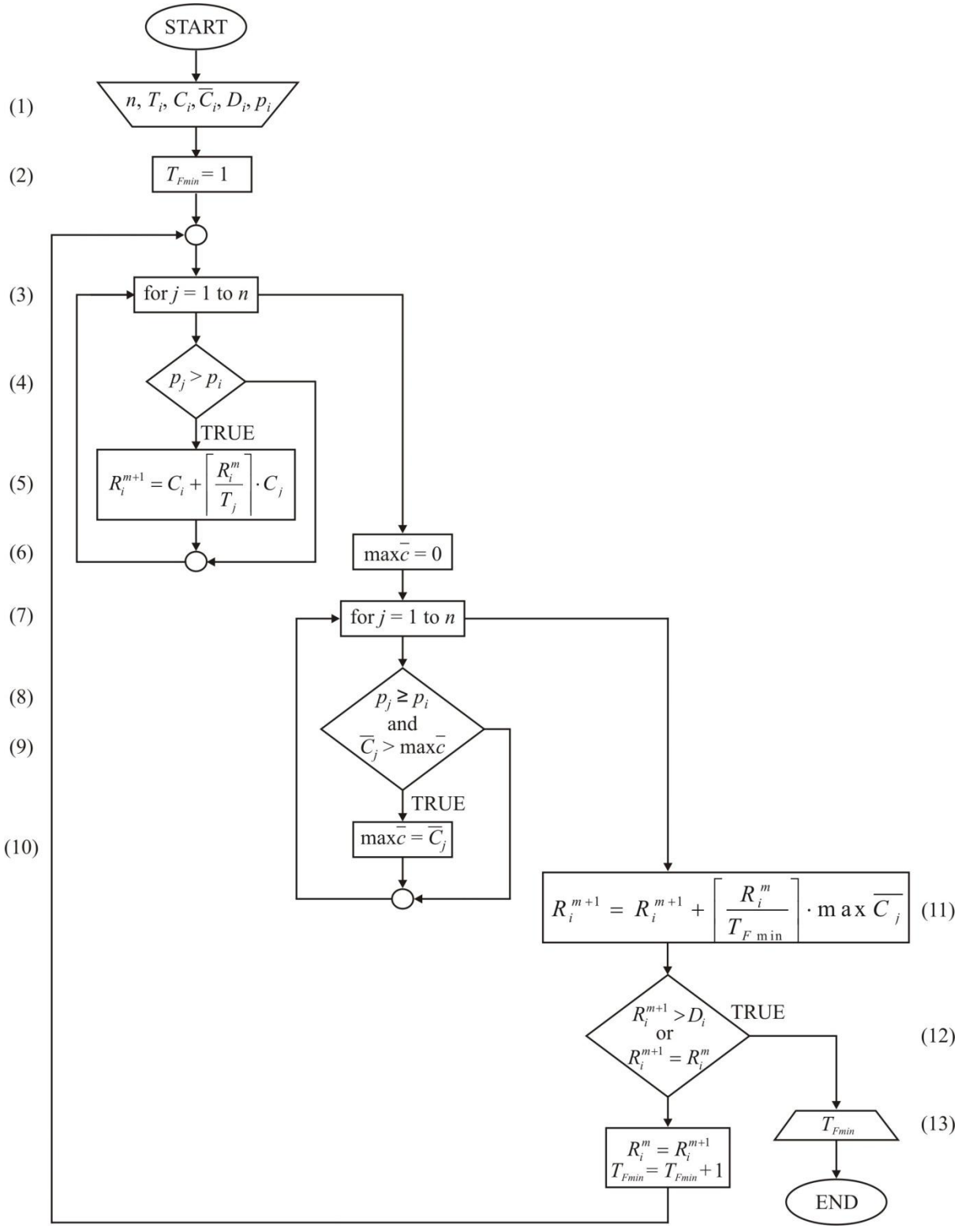
Algoritam započinje tako što dodeli najmanju moguću vrednost parametru T_{Fmin} a zatim proverava da li je dati RTS izvodljiv koristeći RTA. Ako RTS nije izvodljiv, povećava se vrednost parametra T_{Fmin} za jednu vremensku jedinicu i ponovo proverava izvodljivost RTS-a. Ova petlja se izvršava sve dok RTS ne postane izvodljiv za dato T_{Fmin}. Izlaskom iz petlja dobija se vrednost parametra T_{Fmin} tj. najmanji vremenski interval između dve uzastopne moguće pojave greške koje RTS može da toleriše.

Detaljan algoritma kojim se dolazi do vrednosti parametra T_{Fmin} prikazan je na Sl. 4-5. Korak (1) predstavlja ulazne parametre algoritma: broj RT zadataka n i njihove vremenske karakteristike T_i, C_i, $\overline{C_i}$, D_i i p_i. Na početku algoritma pretpostavlja se vrednost parametra T_{Fmin} od 1 vremenske jedinice tj. $T_{Fmin} = 1$, korak (2).



Sl. 4-4: *Algoritam za izračunavanje* T_{Fmin}

Koraci (3) do (5) predstavljaju prvu algoritamsku petlju gde se za svaki RT zadatak τ_i računa njegovo vreme odziva, konkretno računaju se prvi i drugi sabirak jednačine (4-6). Petlja ima ukupno n prolaza, korak (3), gde se u okviru svakog prolaza prvo proveravaju prioriteti zadatka τ_i i τ_j odnosno proverava se da li je ispunjen uslov $p_j > p_i$, korak (4). Ako je uslov ispunjen neophodno je na već izračunato vreme odziva R_i zadatka τ_i dodati vreme potrebno da se izvrši zadatak sa većim prioritetom τ_j, korak (5).



Sl. 4-5: *Detaljan algoritam za izračunavanje vrednosti parametra T_{Fmin}*

Druga algoritamska petlja, koraci (6) do (10), predstavlja postupak pronalaženja maksimalne vrednosti vremena izvršenja RT zadataka čiji su prioriteti jednaki ili veći od

prioriteta trenutno aktivnog zadatka τ_i. I ova petlja ima n prolaza, korak (7) gde se u okviru svakog prolaza prvo proveravaju prioriteti zadatka τ_i i τ_j odnosno proverava se da li je ispunjen uslov $p_j \geq p_i$, korak (8). Ako je uslov iz koraka (8) ispunjen ispituje se da li je vreme izvršenja C_j zadatka τ_j veće od trenutne maksimalne vrednosti vremena izvršenja, korak (9). Ako jeste onda vrednost C_j postaje trenutna maksimalna vrednost vremena izvršenja korak (10).

U koraku (11) računa se konačna vrednost vremena odziva R_i zadatka τ_i. Kako je, u skladu sa jednačinom (4-6), to iterativni proces računanje se završava ili kada je $R_i^{n+1} = R_i^n$ (određeno je vreme odziva zadatka τ_i u najgorem slučaju) ili kada je $R_i^{n+1} > D_i$ (slučaj kada se ne može garantovati da će zadatak τ_i biti izvršen pre svog roka), korak (12). Ako je uslov predstavljen korakom (12) ispunjen to znači da se došlo do izlazne veličine tj. vrednosti parametra T_{Fmin}, korak (13). U suprotnom, ako uslov predstavljen korakom (12) nije ispunjen, povećava se pretpostavljena vrednost parametra T_{Fmin} za jednu vremensku jedinicu i nastavlja se iterativni proces počevši od koraka (3).

Za potrebe disertacije, na osnovu algoritma sa Sl. 4-5 realizovan je program koji omogućava pronalaženje vrednosti parametra T_{Fmin}. Ovim programom omogućeno je brzo i efikasno izračunavanje parametra T_{Fmin} čak i za veće skupove RT zadataka. Sve ovo omogućava kvalitetnu analizu rada RTS-a sa stanovišta provere njegove izvodljivosti i ograničenja vezanih za toleranciju grešaka.

Tab. 4-4: *Primer IV*

Zadaci	Parametri zadataka				T_{Fmin} = 275
	C_i	T_i	D_i	p_i	R_i
τ_1	30	100	100	1	60
τ_2	35	175	175	2	100
τ_3	25	200	200	3	155
τ_4	30	300	300	4	275

Algoritam za izračunavanje vrednosti parametra T_{Fmin} ilustrovan je jednostavnim primerom četiri RT zadatka čiji su parametri (vreme izvršenja C_i, period pojavljivanja T_i, rok za izvršenje D_i, prioritet zadatka p_i) dati u Tab. 4-4. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. Izlazna vrednosti algoritma parametar T_{Fmin}, za konkretni

set RT zadataka, iznosi 275 vremenskih jedinica. Ovo znači da ako se dve uzastopne greške jave u intervalu manjim od 275 vremenskih jedinica ne može se garantovati uspešno funkcionisanje datog RTS-a.

4.5 Modifikacija RTA za slučaj kada treba obezbediti maksimalnu redundansu za zadatak najvećeg prioriteta

Tipično, u RTS-u postoji jedan ili nekoliko zadataka kritičnih po pitanju bezbednosti sistema. Pojava greške u izvršenju kritičnog zadatka zahteva momentalni oporavak, u smislu da ponovno izvršenje zadataka ili izvršenje procedure za oporavak mora početi neposredno nakon što je greška detektovana. Jedan način kako se može garantovati trenutna reakcija na grešku koja se javila prilikom izvršenja kritičnog zadatka jeste da se za kritični zadatak predvidi 100% vremenska redundansa, odnosno da se rezerviše vreme za oporavak odmah nakon završetka zadatka koje se neće koristiti za izvršenje drugih zadataka.

Treba uočiti da je povećanje bezbednosti sistema na ovaj način praćeno smanjenjem sposobnosti sistema da toleriše greške jer vremenska redundansa koja je fiksno vezana za jedan zadatak ostavlja manje vremena za oporavak preostalih zadataka.

Ako se uzme u obzir prethodno opisan model RTS-a, gde se za kritični zadatak predviđa 100% vremenska redundansa, RTA ne može direktno da se primeni. Razlog leži u činjenici da je vremenska redundansa kod RTA deljivi resurs između svih zadataka tako da je neophodno modifikovati za opisan model.

Jedan od zadataka ove disertacije bila je i upravo ova modifikacija RTA za slučaj kada treba obezbediti maksimalnu redundansu za kritični tj. zadatak najvećeg prioriteta. Sa tim ciljem pretpostavljen je RTS kod koga postoji jedan kritičan zadatak. Bez narušavanja opštosti pristupa pretpostavlja se da je to zadatak najvišeg prioriteta τ_1. Za ovaj zadatak obezbeđena je vremenska redundansa dovoljna za njegovo ponovljeno izvršenje. Uvođenjem ove pretpostavke i (u skladu sa njom) modifikacijom algoritma za izračunavanje vrednosti parametra T_{Fmin}, može se doći do preciznije analize mogućnosti prevazilaženja grešaka RTS-a. Cilj ovog dela disertacije bio je da se sagleda kako uvedena vremenska redundansa kritičnog zadatka utiče na frekvenciju pojavljivanja grešaka u toku izvršenja ostalih zadataka u jednom RTS-u.

Značaj modifikacije RTA može se ilustrovati jednim jednostavnim primerom RTS-a koji se sastoji od četiri zadatka čiji su parametri dati u Tab. 4-5. Brojne vrednosti u tabeli

predstavljaju vremenske jedinice izražene u ms. Ako se na dati RTS primeni algoritam sa Sl. 4-5 dobija se da je minimalno vreme između moguće pojave dve uzastopne greške koje sistem može da toleriše 60 vremenskih jedinica tj. T_{Fmin} = 60.

Tab. 4-5: *Primer V*

Zadaci	Parametri zadataka				T_{Fmin} = 60
	C_i	T_i	D_i	p_i	R_i
τ_1	20	100	100	1	40
τ_2	25	175	175	2	95
τ_3	20	200	200	3	160
τ_4	25	300	300	4	300

Pretpostavimo sada da je za zadatak najvećeg prioriteta, τ_1, obezbeđena vremenska redundansa od 100% dupliranjem njegovog vremena izvršenja. Sada je

$C_1 = C_1 + C_{1dod} = 20 + 20 = 40$

gde je C_1 vreme izvršenja zadatka τ_1, a C_{1dod} dodatno vreme potrebno za ponovljeno izvršenje zadatka τ_1 u slučaju da dolazi do greške. Ako se sa novim parametrom C_1 primeni algoritam sa Sl. 4-5 dobijaju se rezultati prikazani u pretposlednjoj koloni Tab. 4-6.

Tab. 4-6: *Primer VI*

Zadaci	Parametri zadataka				T_{Fmin} = 275	T_{Fmod} = 143
	C_i	T_i	D_i	p_i	R_i	R_{imod}
τ_1	40	100	100	1	80	40
τ_2	25	175	175	2	145	90
τ_3	20	200	200	3	165	175
τ_4	25	300	300	4	275	285

Sada je dobijena vrednost za T_{Fmin} znatno povećana i iznosi 275 vremenskih jedinica. Ovo povećanje vrednosti parametra T_{Fmin} znači da RTS može da toleriše mnogo manji broj grešaka. Pored toga vreme odziva prvog zadatka R_1 je 80 vremenskih jedinica. Kako je već pretpostavljeno dvostruko vreme izvršenje prvog zadatka, onda rezultat koji nam RTA daje

nije validan. Realno vreme odziva prvog zadatka treba da bude u najgorem slučaju (u slučaju da se greška javi pred sam kraj izvršenja zadatka) 40 vremenskih jedinica.

Kako bi se dobili korektni rezultati neophodno je izvršiti modifikaciju algoritma za izračunavanje minimalne vrednosti parametra T_{Fmin} koja se pre svega odnosi na jednačinu (4-6) RTA. Osnova modifikacije se sastoji u tome da se prilikom izračunavanja vremena odziva izbaci mogućnost pojave greške kod zadatka najvećeg prioriteta kod koga je obezbeđena 100%-tna vremenska redundansa odnosno zadatka τ_1. Kako treći sabirak jednačine (4-6) računa vreme neophodno za oporavak zadatka usled nastale greške modifikacije treba načiniti baš tu. Naime, prilikom računanja maksimalne vrednosti vremena izvršenja ne treba uzimati u obzir vreme izvršenja zadatka najvećeg prioriteta kod koga je obezbeđena 100%-tna vremenska redundansa. Modifikovana jednačina glasi:

$$R_{imod} = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_{imod}}{T_j} \right\rceil C_j + \left\lceil \frac{R_{imod}}{T_{Fmod}} \right\rceil \max_{j \epsilon hpmod(i)} (C_j) \tag{4-7}$$

gde je *hpmod*(i) = $\{\tau_j \in \Gamma_{\text{mod}} \mid p_j \geq p_i\}$ pri čemu je $\Gamma_{\text{mod}} = \{\tau_2, \ldots, \tau_n\}$. Kako se R_i nalazi sa obe strane znaka jednakosti jednačina (4-7) rešava se iterativnim postupkom opisanim jednačinom (4-8):

$$R_{imod}^{n+1} = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_{imod}^{n}}{T_j} \right\rceil C_j + \left\lceil \frac{R_{imod}^{n}}{T_{Fmod}} \right\rceil \max_{j \epsilon hpmod(i)} (C_j) \tag{4-8}$$

Kriterijumi vezani za početne vrednosti kao i za završetak iterativnog procesa su slični kao za jednačinu (4-6). Za početnu vrednost R_{imod} su treba uzeti da je $R_{imod}^{0} = C_i$ dok se iterativni proces završava kada je $R_{imod}^{n+1} = R_{imod}^{n}$ (trenutak kada se nalazi vreme odziva zadatka τ_i u najgorem slučaju) ili kada je $R_{imod}^{n+1} > D_i$ (slučaj kada zadatak τ_i ne može da zadovolji kriterijume trenutno aktivnog algoritma po kome se zadaci izvršavaju). Sa T_{Fmod} označeno je minimalno vreme između moguće pojave dve uzastopne greške koje RTS može da prevaziđe. Vrednost parametra T_{Fmod} u jednačini (4-8) je ulazna veličina koja se zadaje.

Ako se u algoritmu za izračunavanje vrednosti parametra T_{Fmin}, prikazanom na Sl. 4-5, umesto jednačine (4-6) koristi jednačina (4-8) moguće je izračunati minimalnu vrednost parametra T_{Fmod}. Ako se ovako modifikovani algoritam za izračunavanje minimalne vrednosti

parametra T_{Fmod} primeni na primer iz Tab. 4-6 dobijaju se rezultati dati u poslednjoj koloni iste tabele.

Ono što se može videti na osnovu rezultata datih u Tab. 4-6 jeste da je vreme odziva prvog zadatka R_{1mod} = 40 vremenskih jedinica što je u skladu sa polaznom pretpostavkom. Takođe se može zapaziti i promena vrednosti parametra T_{Fmin}. Primenom modifikovanog algoritma dobila se i manja vrednost za T_{Fmin} tj. sada je T_{Fmod} = 143. Time je pokazano da RTS može da toleriše mnogo veći broj grešaka nego što je to predviđeno primenom RTA u izvornom, nemodifikovanom obliku. Konkretno, smanjenje vrednosti parametra T_{Fmin} sa 275 na 143 predstavlja poboljšanje od čak 48% što je veoma bitna činjenica sa stanovišta projektovanja RTS-a. Modifikacijom RTA dobili smo optimističnije rezultate analize mogućnosti prevazilaženja grešaka u RTS-u.

4.6 Modifikacija RTA za slučaj kada treba izvršiti alternativni zadatak u cilju oporavka sistema usled pojave greške

RTA se može koristiti kod metoda tolerancije grešaka zasnovanih na tehnikama kod kojih se oporavak od greške postiže izvršavanjem alternativnog zadatka ili ponavljanjem zadatka u toku čijeg izvršenja je došlo do pojave greške. Osnovna verzija RTA definisana je za slučaj ponovljenog izvršenja zadatka kod koga se greška javila. Jedan od ciljeva disertacije bio je modifikovati RTA za slučaj kada se za oporavak od greške koriste alternativni zadaci. Osnova modifikacije leži i činjenici da alternativni zadaci imaju različito vreme izvršenja od vremena izvršenja zadataka RT sistema.

Pretpostavlja se RTS koji se sastoji od n primarnih RT zadataka, $\Gamma=\{\tau_1,...,\tau_n\}$, koje sistem izvršava u skladu sa nekim algoritmom rada planera. Za svaki zadatak, τ_i, u skupu Γ, poznat je period pojavljivanja T_i, rok za izvršenje zadatka D_i ($D_i \leq T_i$), kao i vreme izvršenja zadatka C_i.

Pored toga, svakom primarnom zadatku τ_i pridružen je alternativni zadatak, $\overline{\tau_i}$. Alternativni zadatak predstavlja određenu akciju tj. zadatak koji je neophodno izvršiti pre roka za izvršenje primarnog zadatka kako bi se sistem oporavio usled nastale greške. Za svaki alternativni zadatak poznato je još i njegovo vreme izvršenja, $\overline{C_i}$, koje se može nazvati i vreme oporavka zadatka τ_i.

Pretpostavimo još da postoji i n nivoa prioriteta primarnih i alternativnih zadataka (1,2,...,n), pri čemu je sa 1 označen najniži prioritet. Prioritet primarnog zadatka τ_i označen je sa p_i dok je prioritet alternativnog zadatka $\overline{\tau_i}$ označen sa $\overline{p_i}$.

Modifikacija RTA odnosi se na činjenicu da sada prilikom izračunavanja vremena odziva RT zadataka R_i treba uzeti u obzir vreme neophodno za izvršenje alternativnog zadatka. Najgori slučaj je kada greška prekida izvršenja zadatka u momentu neposredno pred kraj njegovog izvršenja i kada je vreme neophodno za oporavak tog zadatka najveće (tj. kada se greška javlja kod zadatka koji ima najduži alternativni zadatak). Modifikacija se odnose na treći sabirak gde sada treba uzeti u obzir vreme izvršenja najdužeg alternativnog zadatka. Modifikovana RTA jednačina sada ima oblik:

$$R_i = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_F} \right\rceil \max_{j \epsilon hpe(i)} (\overline{C_j}) \tag{4-9}$$

gde je $hp(i) = \{\tau_j \in \Gamma \mid p_j > p_i\}$, $hpe(i) = \{\tau_j \in \Gamma \mid p_j \geq p_i\}$ i $\overline{C_i}$ vreme izvršenja alternativnog zadatka. I ovde je T_F ulazna veličina i označava vremenski interval između moguće pojave dve uzastopne greške. Kako se R_i nalazi sa obe strane znaka jednakosti jednačina (4-9) rešava se iterativnim postupkom opisanim jednačinom (4-10):

$$R_i^{n+1} = C_i + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j + \left\lceil \frac{R_i^n}{T_F} \right\rceil \max_{j \epsilon hpe(i)} (\overline{C_j}) \tag{4-10}$$

gde za početnu vrednost R_i treba uzeti da je $R_i^0 = C_i$. Iterativni proces se završava ili kad je $R_i^{n+1} = R_i^n$ (određeno je vreme odziva zadatka τ_i u najgorem slučaju) ili kada je $R_i^{n+1} > D_i$ (slučaj kada se ne može garantovati da će zadatak τ_i biti izvršen pre svog roka).

4.7 Analiza rada RTS-a za različite planere i načine oporavka

U okviru ove sekcije biće analiziran rad RTS-a sa dva stanovišta: algoritmi planera i načini oporavka. Stanovište vezano za algoritme planera odnosi se na način dodele prioriteta zadacima RTS-a. Konkretno biće razmatrani RM i DM algoritmi za dodelu prioriteta detaljno objašnjeni u sekciji 2.5. Ova dva algoritma spadaju u grupu vremenski baziranih algoritama čiji je osnovni kriterijum prilikom određivanja rasporeda izvršenja zadataka period

pojavljivanja (za RM algoritam) odnosno rok za izvršenje zadatka (za DM algoritam) [Bra02a], [Bra02b] i [Đoš04].

Drugo stanovište vezano za način oporavka odnosi se na oporavak RTS-a izvršavanjem alternativnog zadatka ili ponavljanjem zadatka u toku čijeg izvršenja je došlo do pojave greške.

Pretpostavimo prvi slučaj da se RT zadaci izvršavaju u skladu sa RM algoritmom a da je oporavak sistema ponavljanjem zadatka kod koga se javila greška. Karakteristike zadataka (period pojavljivanja T_i, vreme izvršenja C_i, rok za izvršenje D_i, prioritet zadatka p_i) date su u Tab. 4-7. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms.

Tab. 4-7: *RTA za slučaj RM algoritma i oporavka ponavljanjem zadatka*

Zadaci	Parametri zadataka					
	T_i	C_i	D_i	p_i	$R_i(11)$	$R_i(10)$
τ_1	13	2	13	3	4	4
τ_2	25	3	25	2	8	8
τ_3	30	5	30	1	22	**37**

Na osnovu realizovanog algoritam za izračunavanje vrednosti parametra T_{Fmin} (sekcija 4.4) dobijeni su rezultati dati u Tab. 4-7. Na osnovu rezultata dolazi se do zaključka da se zadaci čiji su parametri dati u Tab. 4-7 mogu uspešno izvršiti za vrednost $T_{Fmin} = 11$ odnosno zadaci se uspešno mogu izvršavati ako je minimalno vreme između dve uzastopne moguće pojave grešaka 11 vremenskih jedinica. Za $T_F = 10$ nije moguće uspešno funkcionisanje datog RTS-a jer se zadatak τ_3 ne izvršava u roku tj. $R_3(10) > D_3$.

Analizirajmo sada drugi slučaj tri periodična zadatka koji se izvršavaju u skladu sa DM algoritmom (kod koga je $T_i \neq D_i$) i čije su karakteristike (period pojavljivanja T_i, vreme izvršenja C_i, rok za izvršenje D_i, prioritet zadatka p_i) date u Tab. 4-8. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. I u ovom slučaju je oporavak sistema ponavljanjem zadatka kod koga se greška javila.

Na osnovu rezultata RTA prikazanih u Tab. 4-8 može se zaključiti da se zadaci čiji su parametri prikazani u istoj tabeli mogu uspešno izvršiti za $T_{Fmin} = 17$ dok to nije moguće za $T_F = 16$ vremenskih jedinica jer u tom slučaju dolazi do prekoračenja roka za izvršenje zadatka τ_3 tj. (22 > 21).

Tab. 4-8: *RTA za slučaj DM algoritma i oporavka ponavljanjem zadatka*

Zadaci	Parametri zadataka					
	T_i	C_i	D_i	p_i	$R_i(17)$	$R_i(16)$
τ_1	13	2	9	3	4	4
τ_2	25	3	17	2	8	8
τ_3	30	5	21	1	17	**22**

Ako uporedimo prethodna dva slučaja, može se zaključiti da se smanjenjem roka za izvršenje zadataka povećava vrednost parametra T_{Fmin}, odnosno povećava se minimalno vreme između moguće pojave dve uzastopne greške. Samo povećanje parametra T_{Fmin}, u drugom primeru, je loše jer ukazuje na činjenicu da se otpornost sistema na greške smanjuje.

U naredna dva primera biće razmatrani slučajevi oporavka RTS-a, posle detekcije greške, izvršavanjem alternativnog zadatka koji ima kraće vreme izvršenja od primarnog zadatka.

U Tab. 4-9 date su karakteristike (period pojavljivanja T_i, vreme izvršenja C_i, vreme izvršenja alternativnog zadatka $\overline{C_i}$, rok za izvršenje D_i, prioritet zadatka p_i) tri periodična RT zadatka koji se izvršavaju u skladu sa RM algoritmom. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. Pretpostavka je da je vreme oporavka zadatka usled pojave greške kraće od njegovog vremena izvršenja tj. $C_i < \overline{C_i}$.

Tab. 4-9: *RTA za slučaj RM algoritma i oporavka izvršavanjem alternativnog zadatka*

Zadaci	Parametri zadataka					$R_i(6)$	$R_i(5)$
	T_i	C_i	$\overline{C_i}$	D_i	p_i		
τ_1	13	2	1	13	3	3	3
τ_2	25	3	2	25	2	9	9
τ_3	30	5	3	30	1	24	**35**

Na osnovu rezultati RTA datih u Tab. 4-9 sledi da se ovaj skup zadataka može uspešno izvršiti za T_{Fmin} = 6 dok to nije moguće za T_F = 5 vremenskih jedinica jer se u tom slučaju zadatak τ_3 ne izvršava u roku.

Sada se mogu uporediti rezultati RTA dati u Tab. 4-7 i Tab. 4-9. Ova dva primera RTS-a razlikuju se samo u jednom parametru a to je vreme oporavka zadatka. U prvom slučaju (Tab.

4-7) vreme oporavka zadatka jednako je vremenu njegovog izvršenja jer se podrazumeva način oporavka RTS-a ponovljenim izvršavanjem zadataka. U drugom slučaju (Tab. 4-9) oporavak sistema je izvršavanjem alternativnog zadatka. Upoređivanjem rezultata analize u ova dva slučaja može se zaključiti da se smanjenjem vremena izvršenja alternativnog zadataka smanjuje i minimalno vreme između moguće pojave dve uzastopne greške. Ovo smanjenje parametra T_{Fmin} ukazuje na povećanje otpornosti sistema na greške, što je dobro.

Ostao je još slučaj tri periodična RT zadatka koji se izvršavaju u skladu sa DM algoritmom, čije su karakteristike (period pojavljivanja T_i, vreme izvršenja C_i, vreme izvršenja alternativnog zadatka $\overline{C_i}$, rok za izvršenje D_i, prioritet zadatka p_i) date u Tab. 4-10. Brojne vrednosti u tabeli predstavljaju vremenske jedinice izražene u ms. I u ovom slučaju pretpostavlja se da je oporavak RTS-a usled pojave greške izvršavanjem alternativnog zadatka kraćeg trajanja od njegovog vremena izvršenja tj. $C_i < \overline{C_i}$.

Tab. 4-10: *RTA za slučaj DM algoritma i oporavka izvršavanjem alternativnog zadatka*

Zadaci	Parametri zadataka						
	T_i	C_i	$\overline{C_i}$	D_i	p_i	$R_i(7)$	$R_i(6)$
τ_1	13	2	1	9	3	3	3
τ_2	25	3	2	17	2	7	9
τ_3	30	5	3	21	1	21	**24**

Na osnovu rezultata RTA datih u Tab. 4-10 dolazi se do zaključka da se zadaci mogu uspešno izvršiti za $T_{Fmin} = 7$ dok to nije moguće za $T_F = 6$ zbog prekoračenja roka za izvršenje zadatka τ_3. U odnosu na prethodni slučaj (Tab. 4-9) može se uočiti povećanje minimalnog vremena između pojave dve uzastopne greške odnosno smanjenje otpornost sistema na greške.

Primeri skupova RT zadataka dati u Tab. 4-7, Tab. 4-8, Tab. 4-9 i Tab. 4-10, iako jednostavni, ukazuju na opšti tok analize RTS-a sa stanovišta otpornosti sistema na moguće greške koji bi, u zavisnosti od projektantskih zahteva, omogućio izbor adekvatnog algoritma za dodelu prioriteta i načina oporavka sistema.

Na primer, ako se uporede prva dva prikazana slučaja (Tab. 4-7 i Tab. 4-8), može se zaključiti da se smanjenjem roka za izvršenje zadataka povećava minimalno vreme između pojave dve uzastopne greške odnosno otpornost sistema na greške se smanjuje. Ovo znači da

ako se primeni prvi način oporavka sistema, odnosno oporavak sistema ponovnim izvršavanjem zadatka u toku čijeg izvršenja se greška javila, RM algoritam, u ovom konkretnom slučaju, daje bolje rezultate.

Do istog zaključka može se doći i upoređivanjem trećeg i četvrtog prikazanog slučaja (Tab. 4-9 i Tab. 4-10). Znači, i kada se za oporavak od greške koriste alternativni zadaci, RM algoritam daje bolje rezultate u odnosu na DM.

Moguće je vršiti i poređenje rezultata RTA za slučaj kada se zadržava algoritam planera a menja se način oporavka sistema. Odnosno, moguće je vršiti poređenje prvog i trećeg prikazanog slučaja, kada je aktivan RM, odnosno drugog i četvrtog slučaja, kada je aktivan DM algoritam. Kako, kod analiziranih RTS-a, alternativni zadatak ima kraće vreme izvršenja od primarnog, logično je da će i sistem biti otporniji na greške ako se primeni drugi način oporavka sistema, odnosno oporavak sistema izvršavanjem alternativnog zadatka, bez obzira na trenutno aktivan algoritam. Do istog zaključka se dolazi i na osnovu rezultata RTA primenjene na konkretne slučajeve. Analize ovog tipa neizostavne su u procesu projektovanja pouzdanih RTS-a.

5 Energetski efikasni RTS-i

Bitna karakteristika RTS-a jeste njihova energetska efikasnost. Upravljanje potrošnjom sistema (eng. *power management*) obuhvata tehnike kojima se povećava energetska efikasnost RTS-a odnosno kontroliše potrošnja sistema sa ciljem da ona bude minimalna moguća [Uns03], [Jev08].

Upravljanje potrošnjom sistema omogućava ispunjenje strogih ograničenja vezanih za potrošnju energije koja su naročito izražena kod RTS-a sa baterijskim napajanjem. RTS-i koji se primenjuju u avijaciji, robotici, svemirskim stanicama zahtevaju da budu malih dimenzija što automatski zahteva i malo zagrevanje komponenti. Posredno i u ovom slučaju je neophodno smanjiti potrošnju komponenata kako bi disipacija energije odnosno temperatura bila manja. Upravljanje potrošnjom sistema može regulisati i temperaturu a samim tim uticati i na pouzdanost ovakvih sistema.

Upravljanje potrošnjom je takođe od bitne važnosti kod FT RTS kod kojih se tolerancija grešaka obezbeđuje umnožavanjem hardverskih, odnosno softverskih komponenata. S obzirom da dodatne hardverske i/ili softverske komponente neminovno dovode do povećanja potrošnje, primena FT tehnika u RTS-ima često mora biti praćena adekvatnim tehnikama za upravljanjem potrošnjom na nivou sistema.

RTS-i su projektovani tako da sva vremenska ograničenja budu zadovoljena čak i u slučaju da se svi zadaci izvršavaju sa svojim najdužim vremenima izvršenja (WCET). U praksi su vremena izvršenja svih ili bar većine zadataka kraća od WCET-a tako da ostaje neko dodatno slobodno (neiskorišćeno) vreme. Slobodno vreme RTS-a, u suštini, predstavlja resurs koji se može koristiti za poboljšanje energetske efikasnosti. Jedan pristup je da se

tokom slobodnog vremena pojedini delovi sistema isključe ili uvedu u režim smanjene potrošnje. Drugi pristup je da se pojedini zadaci izvršavaju manjom brzinom, tj. pri sniženoj frekvenciji i smanjenim naponom napajanja procesora, čime se produžava njihovo izvršenje, ali i smanjuje utrošak energije.

U ovom poglavlju predstavljene su i klasifikovane tehnike upravljanja potrošnjom RTS-a. Posebna pažnja posvećena je DVS tehnikama, kao i njihovoj podeli na InterDVS i IntraDVS tehnike koje su posebno predstavljene. Takođe, objašnjen je koncept primene i specifičnosti DVS tehnika kod RTS-a.

5.1 Klasifikacija tehnika upravljanja potrošnjom sistema

U osnovi, tehnike kojima se smanjuje potrošnja sistema mogu se podeliti na statičke (eng. *static power management - SPM*) i dinamičke (eng. *dynamic power management - DPM*) [Wis02]. Osnovna razlika između SPM i DPM je u momentu kada se one primenjuju. Naime, SPM se primenjuje *off-line*, u procesu projektovanja sistema i može se primeniti kako na hardver tako i na softver dok se DPM primenjuje *on-line*, u toku rada sistema gde se koriste određene informacije o trenutnom stanju sistema (recimo trenutno manje opterećenje sistema ili stanje mirovanja) kako bi se smanjila potrošnja sistema. Pregled SPM i DPM tehnika prikazan je u Tab. 5-1.

Tab. 5-1: *Klasifikacija tehnika upravljanja potrošnjom sistema*

	CPU/Sistem	Metodologija
SPM	CPU	CL simulacija IL simulacija
	Sistem	SL model Merenja CS model
DPM	CPU	DVS
	Sistem	On/Off

Domen primene tehnika SPM i DPM može biti sistem, odnosno sve hardverske i softverske komponente sistema, ili može biti ograničen na CPU, kao komponentu koja tipično predstavlja najveći potrošač energije u RTS-u. Primena SPM tehnika pretpostavlja

mogućnost procene potrošnje još u fazi projektovanja sistema. Do ove procene se može doći na osnovu simulacionih modela komponenata ili merenjem potrošnje komponenata koje će biti ugrađene u sistem. DPM tehnike uglavnom su zasnovane na dinamičkoj promeni napona/frekvencije rada CPU-a.

5.1.1 SPM tehnike

U osnovi SPM tehnike mogu se podeliti u dve kategorije. Prvu čine tehnike koje se odnose na upravljanje potrošnjom procesora tzv. tehnike niskog nivoa (eng. *low-level approach*). Drugu grupu čine tehnike koje se odnose na upravljanje potrošnjom celog sistema tzv. tehnike visokog nivoa (eng. *high-level approach*).

Tehnike koje se odnose na upravljanje potrošnjom procesora mogu se svrstati u dve kategorije zavisno od nivoa apstrakcije:

- CL (eng. *cycle-level*) – tehnike karakteristične za RTL (eng. *register transfer level*) nivo apstrakcije. Potrošnja procesora se računa po ciklusima rada procesora, gde se u okviru svakog ciklusa na osnovu zadatih instrukcija aktiviraju određeni delovi procesora. Na osnovu modela potrošnje i aktivnosti delova procesora može se precizno proceniti potrošnja kako svake njegove unutrašnje jedinice, tako i procesora kao celine.
- IL (eng. *instruction-level*) – tehnike karakteristične za IL nivo apstrakcije koje izračunavaju potrošnju procesora tako što sumiraju potrošnje prilikom izvršavanja svake instrukcije u nizu instrukcija tog programa.

CL i IL tehnike mogu se koristiti za procenu količine energije potrebne za izvršenje aplikacionog programa (npr. RT zadataka), a zatim se ta procene može koristiti za optimizaciju programa u cilju smanjenja potrošnje procesora.

Tehnike koje se odnose na upravljanje potrošnjom celog sistema mogu se podeliti u zavisnosti od primene tj. da li su u pitanju hardverske, softverske ili hardver/softver komponente:

- SL (eng. *state level*) model – karakterističan za hardverske komponente sistema. Kod ovog modela poznato je kolika je potrošnja pojedinačnih delova sistema u određenim stanjima odnosno kolika je potrošnja prilikom tranzicija iz jednog stanja u drugo. Ukupna potrošnja sistema računa se kao suma potrošnje stanja svake jedinice sistema

pomnožena sa vremenom provedenim u tom stanju plus potrošnja dobijena prilikom svih tranzicija.

- merenja – karakterističan model za softverske komponente sistema. Merenjem se dolazi do podatka kolika je potrošnja celog sistema kao i do podatka koji delovi aplikacija i procedura operativnog sistema čine kritične tačke što se same potrošnje tiče.
- CS (eng. *complete system*) model – karakterističan je i za hardverske i za softverske komponente sistema. Ovim modelom simulira se rad celokupnog sistema i na osnovu energetskih modela komponenata sistema dobija informacija o ukupnoj potrošnji sistema. CS model vodi računa o uticaju korisnika i koda kernela na potrošnju sistema i omogućava identifikaciju sistemskih servisa koji imaju veliki uticaj na povećanje potrošnje.

Podaci dobijeni primenom tehnika koje se odnose na upravljanje potrošnjom celog sistema mogu se koristiti za kreiranje i procenu efekata plana uštede energije na sistemskom nivou.

5.1.2 DPM tehnike

DPM tehnike mogu se podeliti u dve kategorije. Prvu kategoriju čine tehnike koje se odnose na upravljanje potrošnjom procesora dok u drugu kategoriju spadaju tehnike koje se odnose na upravljanje potrošnjom celog sistema.

DVS (eng. *dynamic voltage scaling*) spada u tehnike koje se odnose na upravljanje potrošnjom procesora. Osnovna ideja DVS tehnike proistekla je iz činjenice da je potrošnja procesora srazmerna radnoj frekvenciji procesora i kvadratu napona napajanja. Ušteda u potrošnji procesora se može ostvariti smanjivanjem napona napajanja uporedo sa smanjenjem radne frekvencije procesora [Bur00]. DVS tehnika je veoma rasprostranjena tako da savremeni mikroprocesori, kao što su AMD Mobile Athlon [Mob01], Intel XScale [Int00] i Transmeta Crusoe [Fle01], proizvode se sa mogućnošću primene ove tehnike, [Nas07].

DPM tehnike koje se odnose na upravljanje potrošnjom celokupnog sistema baziraju se na ideji da se isključuju U/I uređaji koji se ne koriste kao i da se isključuju odnosno prelaze „*sleep*“ stanje hard diskovi odnosno displeji. Ponovno uključivanje odnosno buđenje uređaja dešava se kada se ponovo javi potreba za njima.

5.2 DVS tehnika i RTS-i

Specifičnost primene DVS tehnike kod RTS-a ogleda se u činjenici da se promenom napona napajanja odnosno radne frekvencije procesora menja i vreme potrebno za izvršenja zadatka. Drugim rečima, kao rezultat smanjenja potrošnje procesora povećava se vreme potrebno da procesor izvrši zadatke RTS-a pa se može reći da DVS tehnika koristi slobodno vreme procesora kako bi smanjila njegovu potrošnju ali samo do mere koja ne dovodi do prekoračenja vremenskih rokova [Ahm10], [San09].

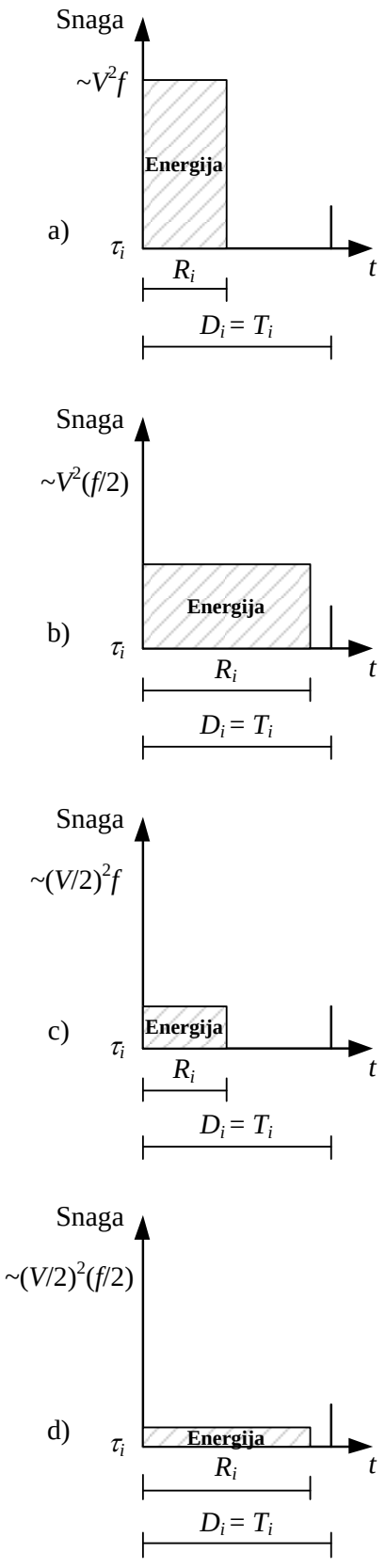
Veza DVS tehnike i RTS-a ilustrovana je na Sl. 5-1. Prvi deo, Sl. 5-1 a), predstavlja jedan RT zadatak, τ_i, i njegove parametre: vreme odziva R_i, period T_i i rok za izvršenje D_i. Pretpostavljeno je da je $T_i = D_i$ i da se zadatak τ_i izvršava pre roka D_i. Utrošak energije procesora da izvrši ovaj zadatak srazmerna je markiranoj površini pravougaonika prikazanog na Sl. 5-1 a) za slučaj da procesor izvršava zadatak radeći na frekvenciji f i naponu V.

Sl. 5-1 b) ilustruje kako bi se RT zadatak τ_i izvršavao na duplo manjoj radnoj frekvenciji $f/2$ i naponu V. Ono što se može videti sa slike jeste da je vreme odziva R_i zadatka τ_i duže jer procesor radi sporije tj. na nižoj frekvenciji, ali i u ovom slučaju zadatak τ_i izvršava se pre roka D_i. Treba napomenuti da u ovom slučaju nije došlo do promene energije.

Sl. 5-1 c) ilustruje kako bi se RT zadatak τ_i izvršavao na frekvenciji f i duplo manjem naponu $V/2$. U ovom slučaju energija bi bila 4 puta manja od polazne, jer snaga zavisi od kvadrata napona. Treba napomenuti da se tako nešto, najčešće ne može postići jer, smanjenjem napona dolazi do pada brzine rada digitalnog kola, i nemogućnosti da i dalje radi na frekvenciji f. To dovodi do toga da smanjenje napona napajanja mora biti praćeno odgovarajućim smanjenjem radne frekvencije što je prikazano na Sl. 5-1 d). Slučaj prikazan na Sl. 5-1 c) je najpovoljniji u smislu uštede energije jer za ovaj slučaj važi da je energija 8 puta manja od polazne.

Slučaj kada smanji napon napajanja procesora V i frekvencija f ilustrovan je na Sl. 5-1 d). Konkretno prikazano je kako bi se RT zadatak τ_i izvršavao na duplo manjoj radnoj frekvenciji $f/2$ i duplo manjem naponu $V/2$. U ovom slučaju usled smanjenja napona napajanja dolazi do uštede energije pri čemu je energija 4 puta manja od polazne.

Cilj primene DVS tehnike kod RTS-a jeste smanjiti potrošnju procesora koja se može definisati kao energija po jedinici vremena. Na dalje u disertaciji kada se bude pričalo o potrošnji procesora smatraće se utrošak energije procesora po jedinici vremena.



Sl. 5-1: *Utrošak energije za parametre procesora*

a) *frekvencija f i napon V;* b) *frekvencija f/2 i napon V;*

c) *frekvencija f i napon V/2*; d) *frekvencija f/2 i napon V/2*

Konkretno na Sl. 5-1 snaga je prikazana na y-osi i može se primetiti da se istovremenim smanjenjem napona napajanja i radne frekvencije procesora smanjuje njegova snaga odnosno potrošnja.

Jedna od aproksimacija koja se koristi kada je u pitanju primena DVS tehnike kod RTS-a odnosi se na procenu promene vrednosti vremena izvršenja C_i RT zadatka τ_i, prilikom promene radne frekvencije odnosno radnog napona procesora. U disertaciji biće prihvaćena aproksimacija da se sa promenom frekvencije rada procesora f_j linearno menja i vreme izvršenja zadatka C_i [Ahm10]. Odnosno da se skaliranjem frekvencije f_j za faktor α skalira C_i za faktor $1/\alpha$ tj. važi da je

$$C_i(f_i) = \frac{f_m}{f_j} C_i(f_m).$$

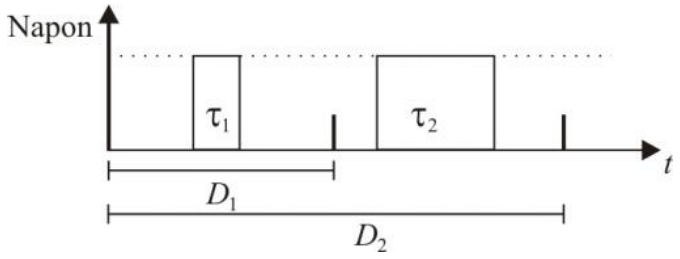
Parametar $C_i(f_m)$ odnosi se na vreme izvršenja zadatka τ_i kada procesor radi na maksimalnoj radnoj frekvenciji f_m. Ova aproksimacija je dosta korišćena i može se videti u [Fen08], [Ahm10], [Pin08].

5.3 Podela DVS tehnika

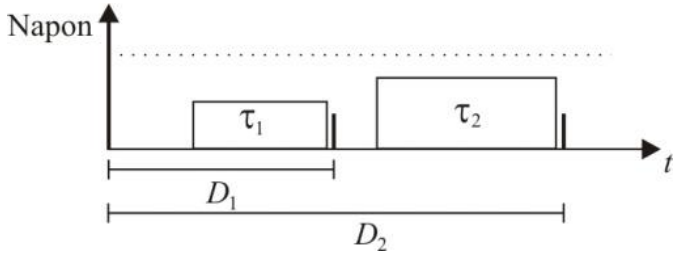
DVS tehnike se mogu podeliti u dve osnovne grupe: InterDVS i IntraDVS [Ses07a]. Karakteristično za InterDVS tehniku je da definiše napon napajanja odnosno frekvenciju procesora na kojoj će se izvršiti RT zadatak tako da ona bude nepromenjena tokom izvršenja zadatka. Suprotno, IntraDVS tehnika omogućava da se napon napajanja odnosno frekvencija procesora menja tokom izvršenja RT zadatka. Postoje još i HybridDVS tehnike su nastale kombinacijom InterDVS i IntraDVS tehnika [Woo02].

Na Sl. 5-2 prikazana su dva RT zadatka τ_1 i τ_2 koji se izvršavaju bez korišćenja DVS tehnika. Sl. 5-3 predstavlja izvršenje istih RT zadatka primenom InterDVS tehnike. Sa Sl. 5-3 može se videti da se za vreme izvršenja zadatka τ_1 ne menja frekvencija procesora na kojoj se zadatak izvršava. Isto važi i za zadatka τ_2 tj. frekvencija procesora na kojoj se zadatak τ_2 izvršava je stalna. Treba naglasiti da najčešće frekvencije procesora na kojima se izvršavaju zadaci, u konkretnom slučaju zadaci τ_1 i τ_2, nisu jednake. Sl. 5-4 ilustruje primenu IntraDVS tehnike. Frekvencije procesora na kojima se izvršava zadatak τ_1 (isto važi i za zadatak τ_2) nije

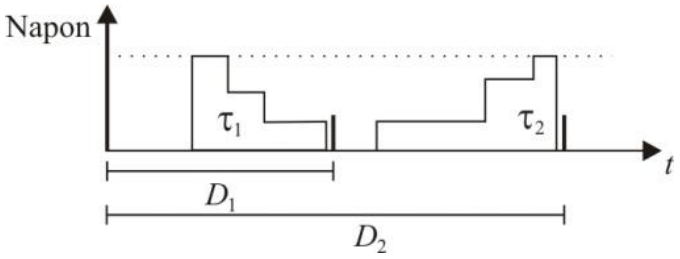
stalna tokom njegovog izvršenja. Zadatak τ_1 je podeljen na delove tako da je frekvencije procesora na kojima se izvršavaju delovi zadatka stalna, s tim što su frekvencije procesora na kojima se izvršavaju delovi zadatka različite. U sva tri slučaja zadaci se izvršavaju pre svog roka za izvršenje D_1, odnosno D_2.



Sl. 5-2: *Izvršenje RT zadataka bez korišćenja DVS tehnika*



Sl. 5-3: *Izvršenje RT zadataka korišćenjem InterDVS tehnika*



Sl. 5-4: *Izvršenje RT zadataka korišćenjem IntraDVS tehnika*

Osnovna razlika između ove dve tehnike je na koji način koriste slobodno vreme procesora sa ciljem da smanje potrošnju. Konkretno, ako se trenutno aktivan RT zadatak završi pre predviđenog roka za izvršenje, preostalo slobodno vreme procesora InterDVS tehnike dodeljuju narednom, po redu izvršenja, RT zadatku. IntraDVS tehnike koriste slobodno vreme procesora trenutno aktivnog zadatka za trenutno aktivan zadatak i omogućavaju da njegovo izvršenje traje duže tj. da se izvrši na nižoj radnoj frekvenciji procesora i ujedno smanji potrošnju.

5.3.1 InterDVS tehnike

InterDVS tehnike odnosno njihovi algoritmi sastoje se od nekoliko osnovnih koraka: (1) izvrši trenutno aktivan RT zadatak τ_i, (2) kada je zadatak τ_i izvršen, izračunaj maksimalno dozvoljeno vreme za izvršenje sledećeg RT zadatka τ_{i+1}, (3) odredi minimalnu frekvenciju izvršenja sledećeg RT zadatka τ_{i+1} tako da ne bude prekoračen rok za izvršenje D_{i+1}, (4) izvrši sledeći RT zadatak τ_{i+1}. Uglavnom se InterDVS algoritmi razlikuju u koraku (2) gde se računa maksimalno raspoloživo vreme za izvršenje narednog RT zadatka τ_{i+1}. Maksimalno raspoloživo vreme zadatka τ_{i+1} računa se kao suma WCET zadatka τ_{i+1} i dodatnog raspoloživog slobodnog vremena procesora nastalo eventualnim ranijim završetkom zadatka τ_i. Na ovaj način ušteda u vremenu ostvarena izvršenjem zadatka τ_i može se koristiti za uštedu energije prilikom izvršenja sledećeg zadatka τ_{i+1}.

InterDVS algoritmi ne moraju biti ograničeni samo na prvi sledeći RT zadatak, već prilikom korekcije radne frekvencije u obzir mogu da uzmu nekoliko narednih RT zadataka. U opštem slučaju, InterDVS algoritmi se sastoje od dva dela:

- prvog koji se odnosi na procenu raspoloživog slobodnog vremena procesora koje se može iskoristiti za izvršenje zadatka (eng. *slack estimation*), i
- drugog koji se odnosi na raspodelu tog vremena između zadataka (eng. *slack distribution*).

Cilj prvog dela jeste da proceni koliko je maksimalno slobodno vreme procesora koje zadaci mogu koristiti, dok je cilj drugog dela algoritma da izvrši raspodelu slobodnog vremena među zadacima tako da frekvencije procesora na kojima će se izvršavati zadaci budu što ujednačenije iz razloga što i sama promena frekvencije traži izvesno vreme i troši izvesnu količinu energije.

Slobodno vreme procesora predstavlja sumu statičkog (eng. *static slack time*) i dinamičkog (eng. *dynamic slack time*) vremena. Statički deo predstavlja slobodno vreme procesora tj. vreme kada procesor ne izvršava zadatke, proračunato na osnovu WCET zadataka. Dinamički deo odnosi se na dodatno slobodno vreme procesora koje nastaje kao posledica toga da se zadaci najčešće izvršavaju za kraće vreme nego što je to bilo predviđeno odnosno da je najčešće stvarno vreme izvršenja zadatka kraće od njegovog WCET.

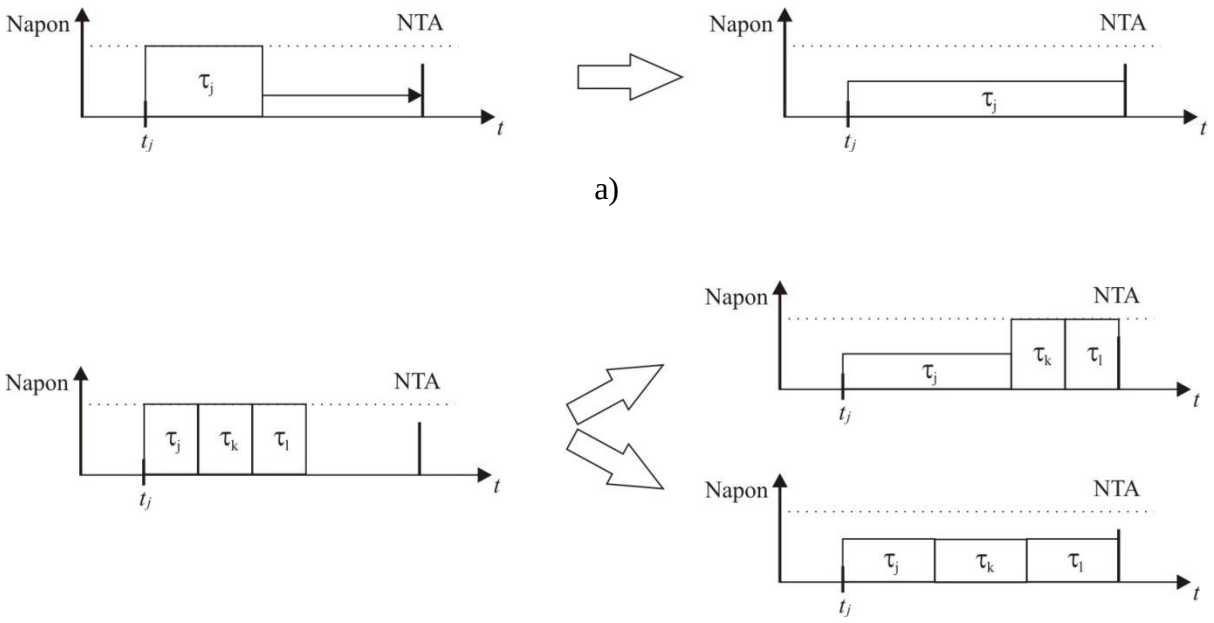
Kada se govori o proceni raspoloživog statičkog dela slobodnog vremena procesora (eng. *static slack estimation* uobičajeno se koristi tzv. metoda maksimalne konstante brzine (eng. *maximum constant speed*). U okviru ove metode traži se najmanja moguća frekvencija rada procesora na kojoj će se izvršavati zadaci, takva da se garantuje njihova izvodljivost

[Shin00]. Pretpostavimo da se RT skup zadataka izvršava u skladu sa EDF algoritmom za raspoređivanje zadataka. Ako pretpostavimo i da je iskorišćenost procesora *U* manja od 1 pri minimalnoj frekvenciji rada procesora f_{max}, onda u tom slučaju metoda maksimalne konstante brzine koriguje maksimalnu radnu frekvenciju procesora tako da bude $f'_{max} = U \cdot f_{max}$. Suština je da ako procesor radi na f'_{max} tada će iskorišćenje biti U = 1 pod pretpostavkom da se svi zadaci izvršavaju za WCET. Znači, tokom rada sistema frekvencija nikada neće biti viša od f'_{max}, a eventualni raniji završetak pojedinih zadataka se može iskoristiti za dodatno smanjenje frekvencije. Metoda maksimalne konstante brzine primenjena na RT skup zadataka koji se izvršava u skladu sa RM algoritmom detaljno je obrađena u [Shi00], [Gru01].

Jedna od metoda karakteristična za procenu raspoloživog dinamičkog dela slobodnog vremena procesora naziva se proširenje do dolaska narednog zadatka (eng. *Stretching to NTA - Next Task Arrival*). Ova metoda kao osnovni kriterijum koristi podatak vezan za vreme dolaska odnosno trenutak iniciranja narednog zadatka, skraćeno NTA. Osnova metode sastoji se u sledećem: ako se u trenutku *t* izvršava RT zadatak τ i ako važi da je NTA>*t*+WCET(τ) onda se zadatak τ može izvršiti na nižoj frekvenciji tako da završi sa izvršavanjem baš u trenutku NTA. Metoda je ilustrovana na Sl. 5-5 i to za slučaj jednog Sl. 5-5 a) odnosno tri RT zadatka Sl. 5-5 b).

Na Sl. 5-5 a) prikazan je RT zadatka τ_j koji počinje sa izvršenjem u trenutku t_j i koji se izvršava pre trenutka iniciranja narednog zadatka, NTA. Ako se primeni opisana metoda moguće je sniziti frekvenciju na kojoj se izvršava zadatak τ_j tako da se produži njegovo vreme izvršenja upravo do trenutka NTA.

Na Sl. 5-5 b) prikazano je kako je opisana metoda primenjena za slučaj tri RT zadatka. U ovom slučaju, odnosno uopšte za slučaj više zadataka, postoji nekoliko mogućnosti kako iskoristiti dinamički deo slobodnog vremena. Jedna je da se dinamički deo slobodnog vremena procesora dodeli samo jednom zadatku. Tada se taj zadatak izvršava sporije, na nižoj frekvenciji, a njegovo vreme izvršenja je toliko da popuni svo slobodno vreme. Druga mogućnost je da se dinamički deo slobodnog vremena procesora podeli svim zadacima, kada se svi izvršavaju sporije ali tako da se završe pre vremena NTA. Obe mogućnosti prikazane su na Sl. 5-5 b).



Sl. 5-5: *Metoda proširenje do dolaska narednog zadatka za slučaj*
a) jednog RT zadatka
b) tri RT zadatka

Treba naglasiti da RT zadatak τ_j sa Sl. 5-5 a), odnosno RT zadaci τ_j, τ_k i τ_l sa Sl. 5-5 b) su zadaci koji tek treba da se izvršavaju, odnosno inicirani su, ali čekaju da se završi neki njima prethodni zadatak koji se završava u trenutku t_j. Moguće je da se taj prethodni zadatak završio ranije nego što je očekivano i da se tako javilo novo dodatno slobodno vreme koje treba raspodeliti.

Kada se govori o delu InterDVS algoritma koji se odnosi na raspodelu slobodnog vremena procesora uglavnom se misli na tzv. *greedy* tehniku koja se najviše koristi. U osnovi ova tehnika se sastoji u tome da svo slobodno vreme procesora dodeljuje narednom RT zadatku koji treba izvršiti. Dodela celokupnog slobodnog vremena samo jednom, prvom narednom zadatku, sigurno nije najbolje rešenje u svim slučajevima, ali je veoma korišćeno baš zbog svoje jednostavnosti.

5.3.2 IntraDVS tehnike

Prilikom *off-line* određivanja rasporeda po kome će se izvršiti zadaci *hard* RTS-a, a sa ciljem garancije zadovoljenja svih zadatih vremenskih zahteva, najčešće se uzimaju vrednosti WCET kao njihova vremena izvršenja. Po pravilu, *on-line* vremena izvršenja RT zadataka su

kraća od pretpostavljenih vrednosti WCET, što kao rezultat daje neko dodatno slobodno vreme (eng. *slack time*). IntraDVS tehnike koriste to dodatno slobodno vreme dobijeno od trenutno aktivnog zadatka upravo za sam taj zadatak i omogućavaju da se on izvršava na nižoj frekvenciji procesora čime ostvaruju uštede u energiji. U zavisnosti od toga kako procenjuju koliko će to slobodno vreme biti i u zavisnosti od načina promene brzine rada procesora IntraDVS tehnike odnosno njihovi algoritmi dele se u dve grupe: *path-base* metod i stohastički metod.

Kod prve, *path-base* IntraDVS metode, napon napajanja odnosno frekvencija procesora određeni su na osnovu pretpostavljene putanje kroz programski kod kojom je zadatak krenuo da se izvršava, [Ins07]. Suština je da ova metoda slobodno vreme traži unapred, a ne unazad tj. ne oslanja se na vreme koje je dobijeno ranijim završetkom zadatka, već na kraće vreme izvršenja preostalog dela zadatka jer je u nekoj tački tokom svog izvršenja zadatak krenuo putanjom koja se izvršava za kraće vreme od pretpostavljenog. Ako stvarno vreme izvršenja odstupa od predviđenog frekvencija procesora se menja i to: smanjuje, ako je vreme izvršenja kraće od predviđenog, odnosno povećava ako je vreme izvršenja duže. Prilikom promene frekvencije rada procesora menja se vreme izvršenja zadatka tako da je neophodno voditi računa o svim zadatim vremenskim ograničenja. Cilj *path-base* IntraDVS metode jeste, kada god je to moguće, smanjiti frekvenciju procesora i uštedeti energiju. Postoje brojne tehnike koje određuju kod kojih zadataka i u koje vreme je moguće izvršiti promenu frekvencije a neke od njih su opisane u [Lee00], [Shi01].

Stohastički IntraDVS metod bazira se na ideji da je bolje započeti izvršenje zadatka na nižoj frekvenciji procesora a zatim povećati frekvenciju i ubrzati izvršenje ako bude bilo potrebe za tim, nego krenuti sa izvršenjem zadatka na višoj frekvenciji a zatim je smanjivati. U suštini, umesto da se zadatak izvršava konstantom brzinom koja je prilagođena njegovom WCET, zadatak se izvršava brzinom koja postepeno raste, ali tako da u najnepovoljnijem slučaju trenutak završetka zadataka bude isti kao da se izvršavao konstantnom brzinom. Ukoliko dođe do ranijeg završetka zadataka, po stohastičkoj IntraDVS metodi, nije neophodno povećavati frekvenciju rada procesora tj. biće izbegnut rad na višim frekvencijama. Teoretski, ako se unapred zna raspodela gustine verovatnoće vremena izvršenja zadataka onda se može izračunati optimalni zakon promene brzine izvršenja zadatka [Gru01]. Ono što je još karakteristično za stohastički IntraDVS metod je činjenica da ona ne koristi svo potencijalno slobodno vreme za razliku od *path-base* IntraDVS metode koja upravo ima za cilj da iskoristi što više slobodnog vremena [Woo02].

Može se zaključiti da je prednost IntraDVS tehnike u odnosu na InterDVS u tome što ne zahteva manipulaciju sa narednim zadacima. Razlika između ove dve tehnike je i u slučaju da u trenutku završetka nekog zadatka (koji se ranije završio) nema iniciranih narednih zadataka, tada kod InterDVS tehnike slobodno vreme do prve inicijacije ostaje neiskorišćeno.

5.3.3 HybridDVS tehnike

U osnovi HybridDVS tehnike nastaju kombinacijom InterDVS i IntraDVS tehnika. Tokom izvršenja trenutno aktivnog RT zadatka i procene eventualnog slobodnog vremena procesora u okviru HybridDVS tehnike odlučuje se da li će se primeniti InterDVS ili IntraDVS metod odlučivanja dodele slobodnog vremena. Ako je odluka korišćenje InterDVS moda onda se slobodno vreme procesora dodeljuju narednom, po redu izvršenja, RT zadatku. Ako je odluka korišćenje IntraDVS moda onda se slobodno vreme procesora koristi za izvršavanje trenutno aktivnog zadatka na nižoj radnoj frekvenciji procesora.

Najčešće se HybridDVS tehnike koriste u zavisnosti od njihove konkretne primene. Recimo [Woo02] razlikuje četiri tipa HybridDVS tehnika:

- H1 – koristi se InterDVS mod rada kao osnovni, a IntraDVS mod samo u slučaju kada u trenutku završetka nekog zadatka nema iniciranih narednih zadataka;
- H2 – koristi IntraDVS mod kao osnovni, a InterDVS mod samo kada je trenutno aktivan zadatak iskoristio svo unapred određeno slobodno vreme;
- H3 – koristi InterDVS mod rada kao osnovni i menja ga u IntraDVS mod kada je neiskorišćeno slobodno vreme procesora veće od unapred procenjenog slobodnog vremena;
- H4 – naizmenično radi u InterDVS i IntraDVS modu održavajući ravnotežu u ravnomernom utrošku slobodnog vremena procesora.

U literaturi se uglavnom mogu naći rezultati koji daju prednost HybridDVS tehnikama u odnosu na čisto InterDVS odnosno čisto IntraDVS tehnike [Woo02], [Rui07], [Ses07b].

6 FT-DVFS algoritam

Kao što je razmatrano u okviru 2. poglavlja, za prevazilaženje otkaza koriste se tehnike na bazi redundanse. Među njima tehnike koje koriste vremensku redundansu ne zahtevaju dodatni hardver, relativno su jeftine i posebno pogodne za primenu kod aplikacija gde postoje ograničenja vezana za težinu, veličinu i potrošnju. U ovom slučaju se za izvršavanje funkcija radi oporavka sistema pri pojavi grešaka koristi slobodno vreme procesora. Tehnike na bazi vremenske redundanse najčešće se koriste za prevazilaženje prolaznih otkaza koji su i tema ove disertacije. Slobodno vreme procesora koristi se za ponovljeno izvršavanje zadataka kod kojih se greška javila, vodeći računa o poštovanju svih vremenskih ograničenja, čime se obezbeđuje da RTS nastavi da funkcioniše. Druga mogućnost je da se slobodno vreme procesora koristi za izvršavanje određenih alternativnih zadataka, opet poštujući vremenska ograničenja, koji bi RTS odveli u određeno bezbedno stanje i time izbegli katastrofalne posledice usled pojave greške.

Sa druge strane, cilj savremenih RTS-a je što manja potrošnja. Kako je potrošnja procesora dominantna u odnosu na potrošnju ostalih delova RTS-a akcenat disertacije je upravo na smanjenju potrošnje procesora. Tehnika koja to omogućava je DVS. Ono što je karakteristično za primenu DVS tehnike kod RTS-a jeste činjenica da se za smanjenje potrošnje procesora takođe koristi njegovo slobodno vreme. Naime, ako je cilj smanjiti potrošnju procesora potrebno je promeniti, konkretno smanjiti napon napajanja procesora. Samim tim dolazi do promene, odnosno smanjenja njegove radne frekvencije što ima za posledicu promenu (povećanje) vremena potrebnog za izvršenja zadatka. Zadatak može da se izvršava duže samo pod uslovom da postoji slobodno vreme procesora koje se može

iskoristiti za to, pri tome vodeći stalno računa da ne dođe do prekoračenja vremenskih rokova.

Zaključak je da slobodno vreme procesora koriste i DVS tehnike i tehnike vezane za prevazilaženje otkaza korišćenjem vremenske redundanse, tako da se prirodno postavlja pitanje kojoj tehnici dodeliti koliko slobodnog vremena procesora. Kako je slobodno vreme procesora ograničeno, očigledno je da ako DVS tehnike potroše više slobodnog vremena manje će ostati za tehnike prevazilaženja otkaza i obrnuto.

Problem simbioze DVS tehnika i tolerancije grešaka kod RTS-a je tek odnedavno prisutan u literaturi [Ahm10], [San09], [Zhu08], [Qad03], [Ayd01], [Mel04]. Ranije su ovi problemi razmatrani posebno, odnosno problemi vezani za upravljanje potrošnjom koristeći DVS tehnike nisu uzimali u obzir toleranciju grešaka. U skorije vreme ova dilema počinje da bude zastupljena u literaturi. U [Zha04b] predstavljena je tehnika na bazi DVS-a modifikovana za toleranciju grešaka pri čemu se razmatra samo mogućnost prevazilaženja otkaza korišćenjem tehnika kontrolnih tačaka. Melham i saradnici [Mel04] su razvili tehniku na bazi DVS-a koja omogućava da se iskoristi slobodno vreme procesora za smanjenje njegove potrošnje. Međutim, ova tehnika polazi od pretpostavke da se može javiti samo jedna greška u toku izvršavanja zadatka kao i da se radna frekvencija procesora može kontinualno menjati. Realno, komercijalni procesori koji podržavaju DVS tehnike imaju striktno definisane frekventno/naponske nivoe [Fle01], [Int00], [Mob01].

Glavni cilj ove disertacije vezan je upravo za pitanje kako naći kompromis između zahteva vezanih za energetsku efikasnost RTS-a i zahteva vezanih za mogućnost prevazilaženja prolaznih otkaza. Osnova problema jeste naći odgovarajuću frekvenciju na kojoj će procesor izvršiti zadatke tako da njegova potrošnja bude najmanja moguća a da se zadovolje svi zahtevi vezani za prevazilaženje prolaznih otkaza.

Rešenje je predstavljeno u vidu novog heurističkog algoritma koji je razvijen upravo za namenu nalaženja kompromisa između energetske efikasnosti i sposobnosti RTS-a da prevaziđu prolazne otkaze [Djo12a], [Djo12b], [Đoš12a], [Đoš12b], [Djo13]. Algoritam je nazvan FT-DVFS (*fault tolerant dynamic voltage frequency scaling*) i on ima za cilj smanjenje potrošnje procesora uz poštovanje svih zahteva vezanih za prevazilaženje prolaznih otkaza i svih zahteva vezanih za vremenska ograničenja RTS-a. FT-DVFS algoritam je primenljiv na skup periodičnih RT zadataka sa fiksnim prioritetom i različitim periodama, koje izvršava procesor sa mogućnošću promene napona napajanja odnosno radne frekvencije. Usvojeno je da RTS prevazilazi prolazne otkaze koristeći tehniku ponovljenog

izvršavanja zadataka koja se bazira na vremenskoj redundansi, koja je detaljno predstavljena u sekciji 3.8.2. Za proveru izvodljivosti zadataka FT RTS-a iskorišćena je analiza vremena odziva (RTA) detaljno obrađena u 4. poglavlju. Treba naglasiti da do sada analiza vremena odziva nije korišćena zajedno sa DVS tehnikama, tako da je prema mojim saznanjima ovo prvo rešenje gde je izvršena integracija RTA u DVS.

U okviru ovog poglavlja dat je detaljan opis modela RTS-a na osnovu koga je predstavljeno rešenje u vidu heurističkog FT-DVFS algoritma koji omogućava pronalaženje kompromisa između energetske efikasnosti i prevazilaženja otkaza. Takođe, razmatrana je i složenost heurističkog FT-DVFS algoritma.

6.1 Model RTS-a

FT-DVFS algoritam razvijen je za skup n RT zadataka, pri čemu su poznate sledeće vremenske karakteristike RT zadataka: vreme izvršenja, period pojavljivanja, rok za izvršenje i prioritet. Takođe, FT-DVFS algoritam podrazumeva i procesor kod koga je moguća promena napona napajanja odnosno radne frekvencije što omogućava različite režime rada sa stanovišta njegove potrošnje. Pored toga, FT-DVFS algoritam uzima u obzir i određena ograničenja vezana za toleranciju grešaka odnosno ograničenja vezana za sposobnost da skup RT zadataka toleriše greške u određenoj meri.

Cilj razvijenog FT-DVFS algoritma jeste da svakom zadatku dodeli odgovarajuću frekvenciju na kojoj će se izvršavati tako da potrošnja procesora bude minimalna moguća a da skup RT zadataka ostane izvodljiv tj. da sva vremenska ograničenja RT zadataka budu zadovoljena. Istovremeno moraju da budu ispunjeni i svi zahtevi vezani za sposobnost RTS-a da toleriše greške.

6.1.1 Model RT zadatka

RTS modelovan je kao skup n RT zadataka, $\Gamma = \{\tau_1, ..., \tau_n\}$. Pretpostavljeno je da su RT zadaci nezavisni i da je jedini resurs koji zadaci dele procesor. Usvojeno je i da su u pitanju zadaci čije izvršenje može biti privremeno prekinuto radi izvršenja zadatka višeg prioriteta. Za svaki zadatak τ_i poznat je:

- period pojavljivanja T_i,
- vreme izvršenja zadatka u najgorem slučaju (WCET) C_i i

- rok za izvršenje D_i.

Pretpostavlja se da je svakom zadatku dodeljen jedinstveni fiksni prioritet p_i. Algoritam planera po kome se izvršavaju zadaci RTS-a može biti bilo koji prioritetni planer za periodične zadatke (algoritmi planera detaljno su obrađeni u okviru sekcije 2.5).

RT zadaci izvršavaju se na jednoprocesorskom RTS-a čiji procesor ima mogućnost promene napona napajanja odnosno radne frekvencije između m diskretnih nivoa. Naponski nivoi označeni su sa U_j (j = 1, ..., m), pri čemu važi da je $U_j < U_{j+1}$ i da U_m predstavlja maksimalni napon napajanja procesora. Frekventni nivoi označeni su sa f_j (j = 1, ..., m) kod kojih važi da je $f_j < f_{j+1}$ i da je f_m maksimalna radna frekvencija procesora. Broj naponskih i frekventnih nivoa je jednak i iznosi m i važi da naponu napajanja procesora U_j odgovara radna frekvencija f_j. Treba naglasiti da naponski i frekventni nivoi uvek idu u parovima (U_j, f_j) odnosno da se pri naponu napajanja U_j isključivo koristi frekvencija f_j. Nadalje u tekstu kada se koristi termin frekventni/naponski nivo j misli se na par (U_j, f_j).

Pretpostavka je da se promena napona/frekvencije ne obavlja u toku izvršenja zadataka već samo u trenucima kada procesor počinje izvršenje zadatka ili nastavlja izvršenje ranije istisnutog zadatka. Takođe treba napomenuti da se naponi/frekvencije dodeljuju zadacima statički i da se ne menjaju tokom rada sistema.

Kako postoji skup Γ i skup m diskretnih nivoa napona napajanja odnosno radnih frekvencija procesora, sledi da postoji m^n različitih preslikavanja skupa frekvencija na skup zadataka. Svako takvo preslikavanje predstavljeno je vektorom $F = (x_1, x_2, ..., x_n)$ nazvanim *vektor dodele frekvencija*, gde je $x_i \in \{1, ..., m\}$ frekventni/naponski nivo dodeljen zadatku τ_i.

Usvojena je aproksimacija koja se često koristi kada je u pitanju primena DVS tehnike kod RTS-a, a koja se odnosi na zavisnost vrednosti vremena izvršenja RT zadatka prilikom promene radne frekvencije procesora [Fen08], [Ahm10], [Pin08]. Naime, pretpostavlja se da se sa promenom frekvencije rada procesora linearno menja i vreme izvršenja zadatka. Drugim rečima, skaliranjem radne frekvencije za faktor α skalira se vreme izvršenja zadatka za faktor $1/\alpha$ tj. važi da je

$$C_i(f_j) = C_i(f_m) f_m / f_j$$

gde je $C_i(f_j)$ vreme izvršenja zadatka τ_i kada procesor radi na frekvenciji f_j, a $C_i(f_m)$ vreme izvršenja zadatka τ_i kada procesor radi na maksimalnoj frekvenciji f_m.

6.1.2 Model potrošnje

U okviru disertacije, usvojen je model potrošnje mikroračunarskih sistema predložen u [Zhu04a] koji se često koristi u radovima vezanim za DVS i RTS-e. Po ovom modelu najveći potrošači mikroračunarskog sistema su procesor, memorija, kola za generisanje i distribuciju taktnog signala kao i periferijske jedinice.

Tokom funkcionisanja RTS-a izmenjuju se periodi aktivnog rada tj. izvršenja RT zadataka (tzv. „*active*“ stanje) i periodi neaktivnosti (tj. periodi između završetka izvršenja zadataka i početka izvršenja sledećeg zadataka). Neaktivni period predstavljaju slobodno vreme procesora. Jedan način za smanjenje potrošnje RTS-a je da se tokom slobodnog vremena procesor prebaci u režim smanjene potrošnje (tzv. „*sleep*“ stanje). Radi smanjenja potrošnje u “*sleep*” stanju, taktovanje većeg dela sistema je obustavljeno, a napon napajanja memorijskih kola je smanjen do nivoa koji omogućava čuvanje informacija, ali ne upis/čitanje. Još veća ušteda u potrošnji sistema moguća je ako se isključe pojedini delovi RTS-a (tzv. „*off*“ stanje). Kada je sistem isključen (u „*off*“ stanju) nema potrošnje. Potrošnja sistema može se podeliti u dve kategorije [Zhu04b]:

- potrošnja u režimu smanjene potrošnje (eng. *sleep power*) i
- potrošnju u aktivnom režimu (eng. *active power*).

Potrošnja u režimu smanjene potrošnje je konstantna, u smislu da zavisi isključivo od karakteristika hardvera RTS-a i na nju se ne može uticati softverskim putem. Ova vrsta potrošnje obuhvata delom potrošnju periferijskih jedinica, kola za generisanje i distribuciju taktnog signala i memorije.

Potrošnja u aktivnom režimu odnosi se samo na potrošnju sistema kada je on aktivan. Ova vrsta potrošnje deli se na:

- frekventno nezavisnu aktivnu potrošnju (eng. *speed-independent active power*) koja ne zavisi od napajanja i frekvencije na kojoj sistem radi. Ona obuhvata delom potrošnju memorije, procesora i delova sistema čija se potrošnja može eliminisati prebacivanjem sistema u režim smanjene potrošnje;
- frekventno zavisnu aktivnu potrošnju (eng. *speed-dependent active power*) obuhvata dinamičku potrošnju procesora i delova sistema koja zavisi od napajanja i frekvencije rada sistema.

Prema tome potrošnja sistema može se modelovati sledećom jednačinom:

$$P = P_s + h(P_{ind} + P_d) \quad (6\text{-}1)$$

gde je

$$P_d = V^2(f)\, C_{ef}\, f \quad (6\text{-}2)$$

U jednačini (6-1) P_s predstavlja potrošnju u režimu smanjene potrošnje, P_{ind} je frekventno nezavisna aktivna potrošnja i P_d je frekventno zavisnu aktivnu potrošnju. Za jednačinu (6-1) važi da kada je sistem aktivan promenljiva $h = 1$ u suprotnom kada je sistem isključen ili u režimu smanjene potrošnje promenljiva $h = 0$.

U jednačini (6-2) C_{ef} je efektivna parazitna kapacitivnost koja je konstanta (dimenziono kapacitivnost) i koja zavisi od složenosti i tehnoloških karakteristika digitalnih kola u RTS-u. Sa f je obeležena frekvencija rada, dok je sa V označen frekventno zavistan napon napajanja.

Potrošnja procesora, kao najvećeg potrošača RTS-a, predstavljena je jednačinom (6-2). Kako je jedan od ciljeva FT-DVFS algoritma smanjiti potrošnju procesora, primenom DVS tehnike koja se bazira na promenama napona napajanja odnosno radne frekvencije procesora, od interesa za algoritam je upravo jednačina (6-2) odnosno frekventno zavisna aktivna komponenta potrošnje.

6.1.3 Model greške i oporavka sistema

U disertaciji fokus je na prolaznim greškama koje se mogu javiti tokom izvršenja bilo kog RT zadatka. Za analizu rada RTS-a kod kojih je moguća pojava grešaka korišćena je RTA (detaljno opisana u 4. poglavlju). Pretpostavlja se da je pojava greške slučajan proces sa ograničenjem T_F. T_F je parametar RTA koji odgovara minimalnom vremenu između dve uzastopne moguće pojave greške. Prema tome ako se greška desila u trenutku t_i, sledeća greška može se desiti samo u trenutku t_{i+1} za koji važi

$t_{i+1} > t_i + T_F$.

Pretpostavka je i da sistem za oporavak koristi vremensku redundansu, tj. slobodno vreme procesora tako što ponovo izvršava zadatak kod koga je detektovana greška. Pri tome ne menja prioritet zadatka niti frekventno/naponski nivo rada procesora. Ponovno izvršavanje zadatka mora biti takvo da ne naruši svoja i vremenska ograničenja ostalih zadataka iz skupa RT zadataka Γ.

Za proveru izvodljivosti FT RTS-a tj. proveru da li svi RT zadaci ispunjavaju vremenske zahteve, u FT-DVFS algoritmu koristi se analiza vremena odziva, konkretno jednačina (4-5) u okviru koje je pojava grešaka predstavljena parametrom T_F. Izvodljivost skupa RT zadataka može se proveriti koristeći iterativni postupak opisan jednačinom (4-6)(4-6). Pri tome se polazi od poznatih činjenica vezanih za skup RT zadataka:

- broj RT zadataka n,
- vremenske karakteristike svakog zadatka - period pojavljivanja T_i, vreme izvršenja C_i, rok za izvršenje D_i, prioritet p_i
- minimalno vreme između dve uzastopne moguće pojave greške T_F.

Kao što je i naglašeno u sekciji 4.2.1, zbog kompleksnosti računice za veći skup RT zadataka u okviru disertacije razvijen je postupak za efikasno izračunavanje vremena odziva R_i prema jednačini (4-6). Ovaj postupak je integrisan u FT-DVFS algoritam.

6.2 Postavka problema

Ulazni parametri FT-DVFS problema su:

- poznati potpuno definisani skup RT zadataka Γ, odnosno poznate karakteristike T_i, C_i, D_i i p_i svakog RT zadatka,
- poznatih m frekventnih/naponskih nivoa rada procesora, odnosno potrošnja procesora karakteristična za svaki od m nivo rada (ovi parametri zavise od tipa procesora),
- ograničenja vezana za tolerisanje grešaka koja se ogledaju kroz parametar koji odgovara minimalnom vremenu između dve uzastopne moguće pojave greške. Zahtevana ograničenja biće obeležena sa T_{Fgoal} i u ovom slučaju predstavljaju ciljno T_F koje se zadaje kao ulazna veličina.

Algoritam FT-DVFS na osnovu ulaznih parametara određuje vektor dodele frekvencija F koji daje informaciju koji zadatak treba izvršavati na kojoj radnoj frekvenciji. Sa stanovišta potrošnje procesora poželjno je da se zadaci izvršavaju na što nižoj frekvenciji. Međutim, rad pri niskim frekvencijama dovodi do povećanja vremena izvršenja zadataka, što može da dovede do prekoračenja krajnih rokova. Zbog toga, cilj algoritma FT-DVSF je da zadacima dodeli što niže radne frekvencije ali tako da ni jedan zadatak ne prekorači svoj krajnji rok.

Optimizacioni problem koji se rešava algoritmom FT-DVSF može se formalno definisati na sledeći način. Skupovi $\{C_1, ..., C_n\}$, $\{T_1, ..., T_n\}$, $\{D_1, ..., D_n\}$ i $\{p_1, ..., p_n\}$ predstavljaju skupove WCET-a, perioda izvršenja, rokova za izvršenje i prioriteta svakog zadatka iz skupa

RT zadataka $\Gamma = \{\tau_1, ..., \tau_n\}$, respektivno. Neka je sa $\{f_1, ..., f_m\}$ označen skup dozvoljenih radnih frekvencija procesora, a sa $F = (x_1, x_2, ..., x_n)$, vektor dodele frekvencija. Ako se sa T_{Fgoal} označi ulazna vrednost minimalnog vremena između pojave dve uzastopne greške, onda se FT-DVFS problem može definisati na sledeći način

$$\min P_{\Gamma}(F) = \sum_{i=1}^{n} P_{d,x_i} \cdot C_{i,x_i} / T_i \tag{6-3}$$

uz ograničenja da

$$R_i(F) \leq D_i,\ i=1, ..., n. \tag{6-4}$$

$P_{\Gamma}(F)$ u jednačini (6-3) predstavlja ukupnu potrošnju procesora koji izvršava skup RT zadataka Γ u skladu sa F. Sa P_{d,x_i} označena je potrošnja procesora koji radi na frekvenciji f_i dok je sa C_{i,x_i} označeno WCET zadatka τ_i koji se izvršava sa frekventnim indeksom x_i. Konkretno C_{i,x_i} definisano je kao

$$C_{i,x_i} = \frac{f_m}{f_i} \cdot C_{i,m}.$$

Jednačina (6-4) predstavlja uslov izvodljivosti skupa RT koji se ispituje koristeći RTA. Da bi se RTA koristila za problem FT-DVFS algoritma neophodno je modifikovati je na način prikazan jednačinom (5.4),

$$R_i^{n+1}(F) = C_{i,x_i} + \sum_{j \epsilon hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_{j,x_j} + \left\lceil \frac{R_i^n}{T_{Fgoal}} \right\rceil \max_{j \epsilon hp(i) \cup i} \left(C_{j,x_j} \right) \tag{6-5}$$

Jednačina (6-5) predstavlja modifikovanu jednačinu (4-6). Jedna modifikacija odnosi se na parametar vremena izvršenja RT zadataka. Umesto vremena izvršenja C_i zadatka τ_i uzima se vreme izvršenja C_{i,x_i} odnosno vreme izvršenja zadatka τ_i koji se izvršava sa frekventnim indeksom x_i. Dobijeno vreme odziva zadatka τ_i iz jednačine (6-5) označeno je sa $R_i(F)$. Druga modifikacija je u parametru T_F jednačine (4-6). Konkretno umesto promenljive T_F koristi se promenljiva T_{Fgoal} koja predstavlja ulazni parametar FT-DVFS problema.

6.3 FT-DVFS algoritam

Trivijalno, a uz to i optimalno rešenje problema FT-DVSF jeste da se generiše svih m^n vektora dodele frekvencija, da se eliminišu oni za koje RTS nije izvodljiv, a zatim da se među preostalim vektorima dodele frekvencija nađe onaj za koji je potrošnja procesora najmanja. Potrošnja procesora se izračunava na osnovu jednačine (6-3) dok se za proveru izvodljivosti RTS-a koristi analiza vremena odziva prema (6-5). Uz sve to treba obezbediti zahtevanu mogućnost tolerisanja grešaka datu kroz ulazni parametar algoritma T_{Fgoal} odnosno interval između moguće dve pojave greške. Opisani metod nalaženja rešenja nazvan je metod iscrpljivanja. Metod iscrpljivanja je koristan i predstavlja optimalno rešenje za skup sa malim broj RT zadataka i procesorom sa nekoliko frekventnih/naponskih nivoa. Glavni problem kod ove metode je da za veći skup RT zadataka i veći broj naponskih odnosno frekventnih nivoa postaje veoma kompleksna i dugotrajna kada je računica u pitanju. Razlog je veliki broj vektora dodele frekvencija F koji predstavlja eksponencijalnu zavisnost broja RT zadataka n i naponskih nivoa m za koje treba eksplicitno ispitati izvodljivost koristeći RTA.

U ovoj sekciji biće predstavljen heuristički FT-DVFS algoritam koji pronalazi skoro optimalno rešenje u zadovoljavajućem vremenu čak i za veoma velike skupove RT zadataka i veliki broj frekventnih/naponskih nivoa procesora.

6.3.1 Pseudo kôd FT-DVFS algoritma

FT-DVFS algoritam počinje tako što svim zadacima dodeli maksimalne frekvencije procesora na kojima će se izvršavati a zatim postepeno smanjuje frekvencije izabranih zadataka odnosno menja vektor dodele frekvencija sve dok se mogu postići poboljšanja vezana za potrošnju procesora a da pri tom uslov izvodljivosti skupa RT zadataka ostane ispunjen.

Na Sl. 6-1 predstavljen je pseudo kôd FT-DVFS algoritma. Ulazni parametri algoritma su:

- frekventni nivoi na kojima procesor može da radi f_j (j = 1, ..., m) pri čemu je $f_j < f_{j+1}$ a m je broj nivoa;
- potrošnja procesora koja odgovara njegovim frekventnim/naponskim nivoima P_j (j = 1, …, m) pri čemu je $P_j < P_{j+1}$ a m je broj nivoa;

- vremenske karakteristike svih n RT zadataka: period T_i, WCET C_i ako procesor radi na frekvenciji f_m, prioritet p_i i rok za izvršenje zadatka D_i, za i=1,..., n;
- zahtevano minimalno vreme između pojave dve uzastopne greške T_{Fgoal}.

Algoritam počinje sa vektorom dodele frekvencija F = (m, m, ..., m) što znači da se svim RT zadacima iz skupa Γ preliminarno dodeljuje maksimalnu frekvenciju izvršenja f_m, korak (1). Takođe su na početku svi zadaci "otključani" tj. imaju mogućnost da menjaju frekvenciju izvršenja, korak (2).

Input: operating frequency levels f_j (j = 1..m),
operating power levels P_j (j = 1..m),
characteristics for n real time tasks (C_i, D_i, T_i, p_i),
fault-tolerant constraint (T_{Fgoal})

Output: Γ with new set F

---

(1) set F = (m, m, ..., m)
(2) unlock all task in Γ
(3) until there are unlock task in Γ
(4) do
(5) for each unlocked task τ_i in Γ do
(6) change τ_i's frequency index from x_i to x_i-1
(7) run schedulability test
(8) if (new Γ is not schedulable) then
(9) lock τ_i
(10) else
(11) $\Delta P(\tau_i) = P_\Gamma(\tau_i, x_i) - P_\Gamma(\tau_i, x_i\text{-}1)$
(12) endif;
(13) return τ_i's frequency index to x_i
(14) end for;
(15) if (unlocked task exist) then
(16) find task τ_p with maximum $\Delta P(\tau_p)$ and set τ_p's frequency index to x_p-1;
(17) end do;

Sl. 6-1: *FT-DVFS algoritam*

Glavna petlja algoritma počinje sa korakom (3) i izvršava se sve dok ima otključanih zadataka, koraci (4 do 17). U svakoj iteraciji algoritma smanjuje se radna frekvencija po jednog "otključanog" zadatka za po jedan frekventni nivo. Svakom otključanom zadatak τ_i pojedinačno, korak (5), privremeno se smanjuje frekvencija izvršenja sa trenutne f_j na $f_{j\text{-}1}$, korak (6). Koristeći RTA tj. jednačinu (6-5) proverava se izvodljivost ovakvog skupa RT

zadataka, korak (7). Korak (7) daje informaciju da li su zahtevi vezani za tolerisanje grešaka (predstavljeni parametrom T_{Fgoal}) ispunjeni. Ako dobijeni skup zadataka nije izvodljiv, korak (8), zadatak τ_i se "zaključava", korak (9). U suprotnom, ako je dobijeni skup zadataka izvodljiv, izračunava se razlika u potrošnji procesora kada izvršava zadatak τ_i na nižoj (f_{j-1}) i na višoj (f_j) frekvenciji, korak (11), kao

$$\Delta P_i(\tau_i) = P_\Gamma(\tau_i, x_i) - P_\Gamma(\tau_i, x_i - 1)$$

gde je sa

$$P_\Gamma(\tau_i, x_i) = P_{d,x_i} \cdot C_{i,x_i}/T_i$$

označena potrošnja procesora kada izvršava zadatak τ_i na nižoj (f_{j-1}) frekvenciji, a sa

$$P_\Gamma(\tau_i, x_i - 1) = P_{d,x_i-1} \cdot C_{i,x_i-1}/T_i$$

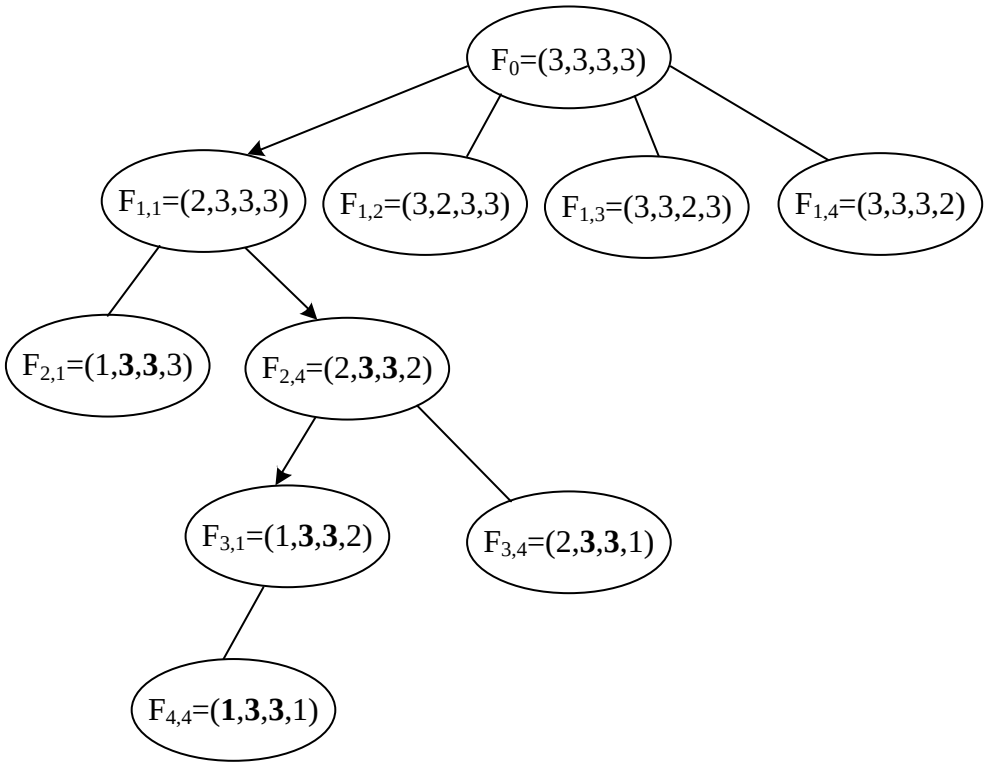
potrošnja procesora kada izvršava zadatak τ_i na višoj (f_j) frekvenciji. Nakon toga, frekventni indeks zadatka τ_i vraća se na x_i odnosno frekvencija zadatka τ_i ponovo vraća na f_j, korak (13). Ovaj postupak ponavlja se za sve "otključane" zadatke i među njima traži se onaj kod koga je smanjenje frekvencije dovelo do najveće uštede u potrošnji procesora. Tom zadatku dodeljuje se (u ovom slučaju za stalno) frekvencija izvršenja smanjena za jedan frekventni nivo, korak (16). U toku iteracije zadaci se "zaključavaju" i u slučaju kada im je dodeljen frekventni indeks jednak jedinici što znači da im je dodeljena frekvencija izvršenja f_1 koja predstavlja najniži frekventni nivo.

Algoritam završava u trenutku kada su svi zadaci iz skupa "zaključani". Izlaz algoritma je vektor dodele frekvencija F koji sadrži frekventne indekse svih RT zadataka odnosno koji sadrži frekvencije na kojima procesor treba da izvrši skup RT zadataka.

6.3.2 Primer rada FT-DVFS algoritma

FT-DVFS algoritam može se objasniti na jednostavnom primeru skupa od četiri potpuno definisana RT zadataka $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. Pretpostavimo da se zadaci izvršavaju na procesoru koji ima tri frekventna nivoa f_j (j = 1, 2, 3), pri čemu je $f_1 < f_2 < f_3$ i f_3 je maksimalna frekvencija. Kako imamo skup od četiri RT zadatka i procesor sa tri moguće frekvencija rada, sledi da postoji 81 različitih vektora dodele frekvencija F. Kako heuristički FT-DVFS algoritam dolazi do rešenja ilustrativno je prikazano na Sl. 6-2.

FT-DVFS algoritam počinje rad sa vektorom dodele frekvencija, F_0 = (3, 3, 3, 3) što znači da je svim zadacima inicijalno dodeljena maksimalna frekvencija izvršenja f_3. Takođe, na početku algoritma svi zadaci su "otključani".



Sl. 6-2: *Putanja kojom se dolazi do rešenja FT-DVFS algoritma za slučaj 4 RT zadatka i 3 frekventna nivoa rada procesora*

U prvoj iteraciji algoritma privremeno se svakom zadatku pojedinačno dodeljuju frekvencije niže za jedan frekventni nivo. To znači da se u prvoj iteraciji algoritma formiraju 4 vektora dodele frekvencija i to: $F_{1,1}$ = (2, 3, 3, 3), $F_{1,2}$ = (3, 2, 3, 3), $F_{1,3}$ = (3, 3, 2, 3) i $F_{1,4}$ = (3, 3, 3, 2). Između njih FT-DVFS algoritam prvo odbacuje one za koje RTS nije izvodljiv, proveravajući to koristeći RTA. Pretpostavimo da su to skupovi zadataka koji odgovaraju vektorima dodele frekvencija $F_{1,2}$ i $F_{1,3}$. Na osnovu prethodnog, algoritam zaključava zadatke τ_2 i τ_3, vraća njihove frekvencije na f_3 i ne dozvoljava im više promenu frekvencija. Nakon toga algoritam računa kolika je ušteda u potrošnji procesora ako se radna frekvencija smanji za jedan frekventni nivo. U konkretnom primeru algoritam proverava uštedu u potrošnji procesora koja se ostvaruje u slučaju skupova zadataka koji odgovaraju vektorima dodele frekvencija F_0 i $F_{1,1}$ i druge grupe skupova koji odgovaraju vektorima dodele frekvencija F_0 i $F_{1,4}$. Pretpostavimo da je ušteda u potrošnji procesora veća za slučaj skupova zadataka koji odgovaraju vektorima dodele frekvencija F_0 i $F_{1,1}$, odnosno da je ušteda u potrošnji procesora veća kada se frekvencija izvršenja zadatka τ_1 promeni sa f_3 na f_2, u odnosu na to kada se frekvencija izvršenja zadatka τ_4 promeni sa f_3 na f_2. Zbog toga algoritam eliminiše vektor

dodele frekvencija $F_{1,2}$, zadatku τ_1 smanjuje radnu frekvenciju "za stalno" na f_2 i počinje drugu iteraciju od vektora dodele frekvencija $F_{1,1}$ = (2, **3**, **3**, 3).

Kako su zadaci τ_2 i τ_3 zaključani, u drugoj iteraciji FT-DVFS algoritam ima mogućnost da privremeno smanji radne frekvencije samo zadacima τ_1 i τ_4. To dovodi do dva nova vektora dodele frekvencija $F_{2,1}$ = (1, **3**, **3**, 3) i $F_{2,4}$ = (2, **3**, **3**, 2). Pretpostavimo da su oba skupa zadataka koji odgovaraju vektorima dodele frekvencija $F_{2,1}$ i $F_{2,4}$ izvodljiva, što znači da za oba skupa zadataka treba računati uštedu u potrošnji procesora. Ušteda u potrošnji procesora računa se za slučaju skupova zadataka koji odgovaraju vektorima dodele frekvencija $F_{1,1}$ i $F_{2,1}$ i za slučaj skupova koji odgovaraju vektorima dodele frekvencija $F_{1,1}$ i $F_{2,4}$. Neka je ušteda u potrošnji procesora veća za slučaj druge grupe skupova koji odgovaraju vektorima dodele frekvencija $F_{1,1}$ i $F_{2,4}$, odnosno za slučaj kada se zadatku τ_4 smanji frekvencija procesora na kojoj se izvršava sa f_3 na f_2. Sve dovodi do toga da algoritam eliminiše vektor dodele frekvencija $F_{2,1}$, zadržava vektor dodele frekvencija $F_{2,4}$ i trajno dodeljuje frekvenciju izvršenja f_2 zadatku τ_4. U ovoj iteraciji oba zadatka, τ_1 i τ_4, ostaju otključana.

U trećoj iteraciji algoritam počinje od vektora dodele frekvencija $F_{2,4}$ = (2, **3**, **3**, 2). Sada je opet moguće smanjiti samo frekvencije na kojima se izvršavaju zadaci τ_1 i τ_4. Vektori dodele frekvencija karakteristični za treću iteraciju su $F_{3,1}$ = (1, **3**, **3**, 2) i $F_{3,4}$ = (2, **3**, **3**, 1). Pretpostavimo opet da su oba skupa zadataka koji odgovaraju vektorima dodele frekvencija $F_{3,1}$ i $F_{3,4}$ izvodljiva, što znači da za oba skupa zadataka treba računati uštedu u potrošnji procesora. Sada se ušteda u potrošnji procesora računa za slučaj skupova zadataka koji odgovaraju vektorima dodele frekvencija $F_{2,4}$ i $F_{3,1}$ i za slučaj skupova koji odgovaraju vektorima dodele frekvencija $F_{2,4}$ i $F_{3,4}$. Neka je sada ušteda u potrošnji procesora veća za slučaj prve grupe skupova koji odgovaraju vektorima dodele frekvencija $F_{2,4}$ i $F_{3,1}$, odnosno za slučaj kada se zadatku τ_1 smanji frekvencija procesora na kojoj se izvršava sa f_2 na f_1. Sada algoritam eliminiše vektor dodele frekvencija $F_{3,4}$, zadržava vektor dodele frekvencija $F_{3,1}$ i trajno dodeljuje frekvenciju izvršenja f_1 zadatku τ_1. Kako je zadatku τ_1 dodeljena najniža frekvencija f_1 FT-DVFS algoritam zaključava i ovaj zadatak.

U okviru četvrte iteracije ostaje samo jedan otključan zadatak τ_4 što omogućava da se generiše samo jedan vektor dodele frekvencija $F_{4,4}$ = (**1**, **3**, **3**, 1). Ako pretpostavimo da skup zadataka koji odgovara vektoru dodele frekvencija $F_{4,4}$ nije izvodljiv onda algoritam zaključava i zadatak τ_4 i vraća mu privremeno promenjenu frekvenciju f_2.

Sada su svi zadaci zaključani tako da FT-DVFS algoritam završava pretragu i kao izlaz daje vektor dodele frekvencija F = (**1**, **3**, **3**, **2**). Drugim rečima, heuristički FT-DVFS

algoritam kao rešenje daje frekvencije procesora na kojima treba da se izvršavaju zadaci τ_1, τ_2, τ_3 i τ_4 skupa Γ i to f_1, f_3, f_3 i f_2, respektivno. Pri tome, algoritam garantuje da će biti zadovoljena sva vremenska ograničenja RT zadataka skupa Γ, sva ograničenja vezana za tolerisanje grešaka i da će pri tome i potrošnja procesora biti minimalna moguća.

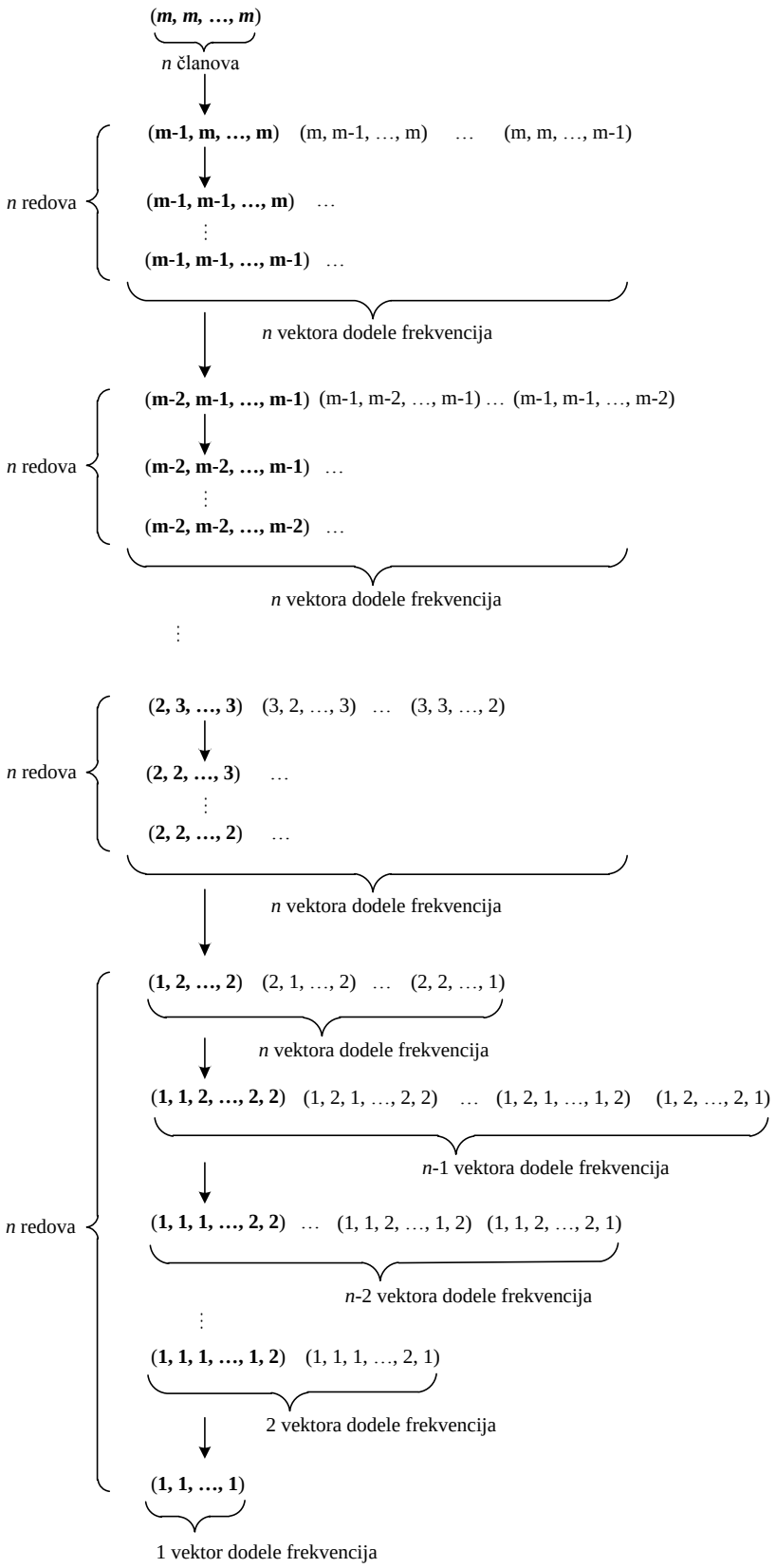
6.4 Složenost heurističkog FT-DVFS algoritma

Vremenski najzahtevniji deo heurističkog FT-DVFS algoritma jeste korak (7) u pseudo kôdu sa Sl. 6-1 u okviru koga se koristi RTA, konkretno jednačina (6-5), radi provere izvodljivosti skupa RT zadataka. Korak (7) daje informaciju da li su zahtevi vezani za tolerisanje grešaka (predstavljeni parametrom T_{Fgoal}) ispunjeni ili ne. Složenost FT-DVFS algoritma ogleda se u broju skupova RT zadataka za koje treba izvršiti RTA tokom pronalaženja rešenja.

Za najgori mogući slučaj sa stanovišta složenosti FT-DVFS algoritma treba pretpostaviti sledeće. Prvo, da su svi zadaci tokom celokupnog rada algoritma otključani, odnosno da su svi novodobijeni RT skupovi zadataka u toku rada algoritma izvodljivi. Drugo, da se tokom rada algoritma smanjuje uvek frekvencija zadatka koja je maksimalna u trenutnoj iteraciji. Ako su oba uslova ispunjena putanja kojom bi se došlo do rešenja FT-DVFS algoritma za slučaj n RT zadatka i m frekventnih nivoa rada procesora izgledala bi kao što je prikazano na Sl. 6-3.

Ako se pretpostavi skup Γ i skup m diskretnih frekventnih/naponskih nivoa algoritam polazi od vektora dodele frekvencija $F_0 = (m, m, \ldots, m)$ odnosno svim RT zadacima iz skupa Γ dodeljuje maksimalnu frekvenciju izvršenja f_m. Radi preglednijeg zapisa na Sl. 6-3 vektori dodele frekvencija označeni su bez odgovarajuće oznake F_{ij} već samo n-to članim skupovima, otuda je vektor dodele frekvencija $F_0 = (m, m, \ldots, m)$ označen samo sa $(m, m, \ldots, m)$.

Nakon vektora dodele frekvencija $(m, m, \ldots, m)$ algoritam u prvoj iteraciji formira n novih vektora dodele frekvencija kod kojih je samo po jednom zadatku snižena frekvencija za jedan frekventni nivo. Ti vektori dodele frekvencija označeni su sa $(m\text{-}1, m, \ldots, m)$, $(m, m\text{-}1, \ldots, m)$, …, $(m, m, \ldots, m\text{-}1)$. FT-DVFS algoritam među dobijenim vektorima dodele frekvencija bira jedan koji će biti polazni za narednu iteraciju. U ovom konkretnom slučaju je svejedno koji vektor dodele frekvencija će biti izabran. Sa Sl. 6-3 može se videti da je izabran vektor dodele frekvencija $(m\text{-}1, m, \ldots, m)$.



Sl. 6-3: *Putanja rešenja FT-DVFS algoritma za najgori slučaj*

Sa vektorom dodele frekvencija (*m*-1, *m*, …, *m*) počinje naredna iteracija algoritma gde se opet formira *n* novih vektora dodele frekvencija kod kojih je samo po jednom zadatku snižena frekvencija za jedan frekventni nivo među kojima se bira onaj vektor koji će biti polazni za narednu iteraciju algoritma. Sa Sl. 6-3 može se videti da je izabran vektor (*m*-1, *m*-1, …, *m*).

Da bi se došlo do vektora dodele frekvencija (*m*-1, *m*-1, …, *m*-1) kod koga su svim zadacima snižene frekvencija za jedan frekventni nivo u odnosu na maksimalnu frekvenciju, potrebno je *n* iteracija što je na Sl. 6-3 označeno sa *n* redova. Ova grupa on *n* iteracija tj. *n* redova, gde u okviru svakog reda ima *n* vektora dodele frekvencija tj. *n* RT skupova zadataka označava da je potrebno *n*·*n* puta izvršiti RTA.

Naredna grupa, odnosno njena prva iteracija počinje od vektora dodele frekvencija (*m*-1, *m*-1, …, *m*-1). U okviru prve iteracije formiraju se *n* novih vektora dodele frekvencija kod kojih je opet samo po jednom zadatku snižena frekvencija za samo jedan frekventni nivo i među njima bira se jedan vektor kao polazni za drugu iteraciju. Sa Sl. 6-3 može se videti da je među njima izabran vektor dodele frekvencija (*m*-2, *m*-1, …, *m*-1). U okviru druge iteracije dobijaju se *n* sledećih vektora dodele frekvencija (*m*-3, *m*-1, …, *m*-1), (*m*-2, *m*-2, …, *m*-1), …, (*m*-2, *m*-1, …, *m*-2) između kojih FT-DVFS algoritam bira vektor (*m*-2, *m*-2, …, *m*-1) koji je početni za treću iteraciju algoritma.

Nakon *n* iteracija dolazi se do vektora dodele frekvencija (*m*-2, *m*-2, …, *m*-2) kod koga su svim zadacima snižene frekvencija za dva frekventna nivo u odnosu na maksimalnu frekvenciju. To znači da i u okviru druge grupe imamo *n* redova sa po *n* vektora dodele frekvencija odnosno *n*·*n* puta treba izvršiti RTA.

Slično se nastavlja i za naredne grupe i dolazi do vektora dodele frekvencija (2, 2, …, 2) koji počinje grupa koja se razlikuje od prethodnih po broju vektora u okviru svake iteracije. Naime, u okviru prve iteracije, snižavanjem frekvencija za jedan frekventni nivo svakom zadatku pojedinačno, moguće je formirati *n* vektora dodele frekvencija (1, 2, …, 2), (2, 1, …, 2), …, (2, 2, …, 1). Sa Sl. 6-3 može se videti da FT-DVFS algoritam bira vektor (1, 2, …, 2) kao početni za drugu iteraciju.

U okviru druge iteracije opet treba smanjiti svakom zadatku pojedinačno frekvenciju za samo jedan frekventni nivo. Kako je prvom zadatku već dodeljena minimalna frekvencija (označena sa 1) nije moguće dalje smanjenje frekventnog nivoa. Ovo znači da će druga iteracija imati *n*-1 vektora dodele frekvencija, a to su (1, 1, 2, …, 2, 2), (1, 2, 1, …, 2, 2), …, (1, 2, 2, …, 1, 2) i (1, 2, 2, …, 2, 1). Pod pretpostavkama vezanim za najgori slučaj, FT-

DVFS algoritam među dobijenim vektorima dodele frekvencija bira vektor (1, 1, 2, …, 2, 2) kao polazni za treću iteraciju.

Kako u trećoj iteraciji polazimo od vektora dodele frekvencija (1, 1, 2, …, 2, 2) kod koga dva zadatka imaju već najniže moguće frekvencije moguće je smanjiti frekvencije samo preostalim n-2 zadacima. To znači da će treća iteracija imati *n*-2 vektora dodele frekvencija, a to su (1, 1, 1, …, 2, 2), (1, 1, 2, …, 1, 2), …, (1, 1, 2, …, 2, 1).

Sličnom logikom dolazimo do *n*-1 iteracije u okviru koje se, polazeći od vektora dodele frekvencija (1, 1, 1, …, 2, 2), mogu formirati samo dva nova vektora (1, 1, 1, …, 1, 2) i (1, 1, 1, …, 2, 1). Koji god vektor da se izabere kao polazni za *n*-tu iteraciju može se formirati samo jedan novi vektor i to je (1, 1, 1, …, 1, 1). Vektor dodele frekvencija (1, 1, 1, …, 1, 1) predstavlja završetak rada FT-DVFS algoritma.

Ono što je karakteristično za poslednju grupu jeste da imamo *n* redova tj. iteracija pri čemu je u svakom redu moguće formirati *n*, *n*-1, *n*-2, … , 2, 1 novih vektora dodele frekvencija. Ovo znači da u okviru poslednje grupe imamo ukupno *n*+(*n*-1)+(*n*-2)+(n-3)+…+2+1 vektora dodele frekvencija. Kako je suma ovog aritmetičkog reda *n*·(*n*+1)/2 to znači da upravo toliko puta treba izvršiti RTA u poslednjoj grupi.

Pod pretpostavkama vezanim za najgori slučaj u okviru FT-DVFS algoritma ukupan broj RTA, označen sa *N*, može se izračunati na sledeći način:

$$N = 1 + (m-2)n^2 + \frac{n(n+1)}{2}$$

Prvi sabirak tj. 1 predstavlja *n*-to člani polazni vektor dodele frekvencija (*m*, *m*, …, *m*). Drugi sabirak predstavlja grupe u kojima je potrebno *n*·*n* izvršiti RTA. Kao što se i sa Sl. 6-3 može videti takvih grupa ima ukupno *m*-2. Treći sabirak odnosi se na broj RTA u poslednjoj grupu koji predstavlja sumu aritmetičkog niza *n*+(*n*-1)+(*n*-2)+(n-3)+…+2+1.

Nakon sređivanja prethodne formule dobija se:

$$N = n^2\left(m - \frac{3}{2}\right) + \frac{n}{2} + 1$$

odakle se može zaključiti da je složenost heurističkog FT-DVFS algoritma polinomalna tj. $O(n^2m)$.

Složenost heurističkog FT-DVFS algoritma može se uporediti sa složenošću optimalnog algoritma tj. metode iscrpljivanja detaljno opisane u sekciji 6.3. U najgorem slučaju kod metode iscrpljivanja broj vektora dodele frekvencija koje treba formirati, odnosno broj RTA koje treba izvršiti jednak je

$$N = m^n$$

gde je n broja RT zadataka a m broj frekventnih/naponskih nivoa. Može se zaključiti da je složenost optimalnog algoritma tj. metode iscrpljivanja eksponencijalna tj. $O(m^n)$.

Ako se napravi poređenje složenosti optimalne metode iscrpljivanja i heurističkog FT-DVFS algoritma zaključuje se da se složenost sa eksponencijalne $O(m^n)$ redukuje na polinomalnu $O(n^2m)$.

Ako bi se konkretan primer RT skupa od 4 zadatka i 3 frekventna/naponska nivoa opisan u sekciji 6.3.2 rešio koristeći optimalnu metodu iscrpljivanja, bilo bi potrebno 81 put izvršiti RTA. Heuristički FT-DVFS algoritam u najgorem slučaju trebao bi da RTA izvrši 27 puta što je znatno manje. Realno sa Sl. 6-3 može se videti da je FT-DVFS algoritmu potrebno samo 9 puta da izvrši RTA kako bi došao do rešenja.

7 Analiza performansi

U okviru ovog poglavlja biće predstavljeni rezultati analize performansi FT-DVFS algoritma za različite modele procesora kao i za različite slučajeve RT skupova zadataka kako sintetičkih tako i onih iz "realnog" sveta. Posebno su analizirani slučajevi RTS-a bez mogućnosti tolerisanja grešaka i kada se greške javljaju. Takođe, izvršeno je i poređenje FT-DVFS metode sa optimalnom metodom iscrpljivanja.

7.1 Simulaciono okruženje

Na bazi FT-DVFS algoritma, opisanog u poglavlju 6, realizovan je simulator u programskom jeziku C++ sa ciljem analize performansi predloženog algoritma. Ulazni parametri simulatora su karakteristike RTS-a, tj. parametri procesora i skupa RT zadataka, kao i ograničenja vezana za mogućnost tolerisanja grešaka. Konkretno, ulaznim parametrima se definiše potrošnja procesora iskazana kroz frekventne/naponske nivoe, zatim broj RT zadataka kao i vremenske karakteristike zadataka (period, vreme izvršenja, prioritet, rok za izvršenje) dok su ograničenja vezana za tolerisanje grešaka predstavljena minimalnim vremenom između moguće pojave dve uzastopne greške. Na tako opisan RTS, simulator primenjuje FT-DVSF algoritam i generiše niz informacija, kao što su izvodljivost RT skupa zadataka, potrošnja procesora, informacije vezane za frekvencije rada procesora na kojima se pojedinačni zadaci izvršavaju i sl., na osnovu kojih je moguće izvršiti analizu rada RTS-a.

Kako je glavna namena FT-DVFS algoritma smanjiti potrošnju procesora tako su i rezultati simulacija analizirani sa stanovišta energetske efikasnosti procesora. Energetska

efikasnost procesora prikazana je kroz odnos njegove potrošnje kada izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji. Ovako izražena energetska efikasnost procesora u procentima predstavlja uštedu u njegovoj potrošnji ostvarenu primenom FT-DVFS algoritma.

7.2 Procesor i RT skup zadataka iz "realnog" sveta korišćen u simulaciji

U okviru ove sekcije biće prikazani rezultate simulacije RTS-a iz "realnog" sveta. Konkretno u pitanju je RT skup zadataka vezan za *Generic Avionics Platform* (GAP) [Loc91]. Ovaj RT skup zadataka sastoji se od deset zadataka čije su vremenske karakteristike prikazane u Tab. 7-1. Za svaki zadatak dati su sledeći parametri: prioritet (p_i), period (T_i) i vreme izvršenja (C_i). Prioriteti zadacima dodeljeni su na osnovu EDF algoritma za raspoređivanje zadataka. Takođe, za svaki zadatak važi da je $T_i = D_i$, odnosno da je rok za izvršenje zadataka jednak njihovoj periodi pojavljivanja. Vreme izvršenja zadataka C_i, dato u Tab. 7-1, odgovara vremenu izvršenja kada procesor radi na maksimalnoj frekvenciji.

Tab. 7-1: *Vremenske karakteristike Generic Avionics Platform skupa zadataka*

Zadatak τ_i	p_i	$T_i = D_i$ (ms)	C_i (ms)
Nav_Status	1	1000	1
BET_E_Status_Update	2	1000	1
Display_Stat_Update	3	200	3
Display_Keyset	4	200	1
Display_Stores_Update	5	200	1
Nav_Steering_Cmds	6	200	3
Tracking_Target_Upd	7	100	5
Display_Hook_Update	8	80	2
Display_Graphic	9	80	9
Nav_Update	10	59	8

Model procesora korišćen u simulaciji bazira se na *Transmeta Crusoe* procesoru koji u sebi već ima ugrađene DVS karakteristike [Zha04]. Karakteristike korišćenog *Transmeta*

Crusoe procesora, mogući nivou frekvencija, napona i snage koja odgovara potrošnji procesora, prikazani su u Tab. 7-2.

Kao što se iz Tab. 7-2 može videti radna frekvencija procesora može se kretati u opsegu od 300MHz do 667MHz. Napon napajanja procesora je u opsegu od 1.6V što odgovara maksimalnom frekventnom nivou do 1.2V za minimalni frekventni nivo. Ako se pogleda kolona Tab. 7-2 vezana za potrošnju *Transmeta Crusoe* procesora može se primetiti da se smanjenje radne frekvencija sa 667MHz na 300MHz tj. za 2.23 puta, potrošnja procesora smanji 4.07 puta.

Tab. 7-2: *Karakteristike Transmeta Crusoe procesora*

Frekvencija (MHz)	Napon (V)	Snaga (W)
300	1.200	1.3
400	1.225	1.9
533	1.350	3.0
600	1.500	4.2
667	1.600	5.3

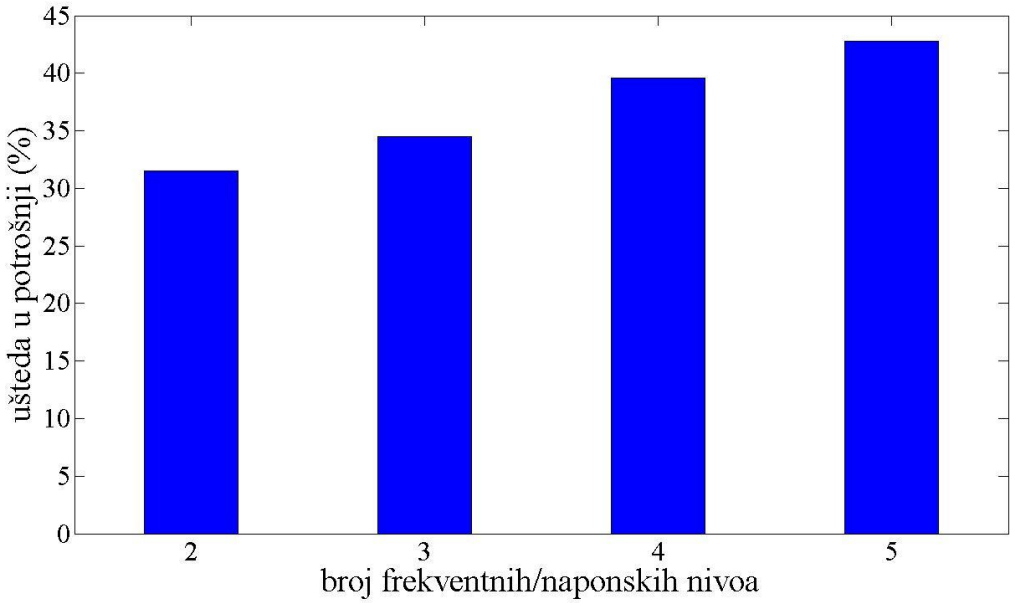
Osim podataka vezanih za karakteristike *Transmeta Crusoe* procesora prikazanih u Tab. 7-2 i vremenskih karakteristika RT skupa zadataka prikazanih u Tab. 7-2, ulazni podaci za simulator su ograničenja vezana za tolerisanje grešaka predstavljena parametrom T_{Fgoal}.

7.2.1 RTS bez mogućnosti tolerisanja grešaka

Prve simulacije rađene su za pretpostavku da RTS nema mogućnosti tolerisanja grešaka. Usvajajući tu pretpostavku, FT-DVFS algoritam primenjen je na GAP RT skup zadataka Tab. 7-1 i *Transmeta Crusoe* procesor Tab. 7-2, sa ciljem dobijanja frekvencija na kojima procesor treba da izvršava RT zadatke tako da potrošnja procesora bude najmanja moguća. Ovde se zapravo radi o slučaju kada je minimalni vremenski interval između dve moguće pojave grešaka beskonačno veliko tj. kada $T_{Fgoal} \rightarrow \infty$. Ovaj slučaj ostavlja mogućnost da se celokupno slobodno vreme procesora koristi za smanjenje njegove potrošnje.

Sl. 7-1 predstavlja rezultate simulacija za navedeni slučaj gde osenčeni pravougaonici označeni sa *i*, $i = 2, \ldots, 5$ predstavljaju uštedu u potrošnji procesora dobijenoj korišćenjem *i*

dostupnih frekventnih nivoa *Transmeta Crusoe* procesora datih u Tab. 7-2. Konkretno za i = 2 korišćene su frekvencije 667MHz i 300MHz, za i = 3 frekvencije 667MHz, 600MHz i 300MHz, za i = 4 frekvencije 667MHz, 600MHz, 533MHz i 300MHz i za i = 5 frekvencije 667MHz, 600MHz, 533MHz, 400MHz i 300MHz. Broj frekventnih nivoa prikazan je na x-osi, dok y-osa predstavlja odnos potrošnje procesora kada izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji izražena u procentima.

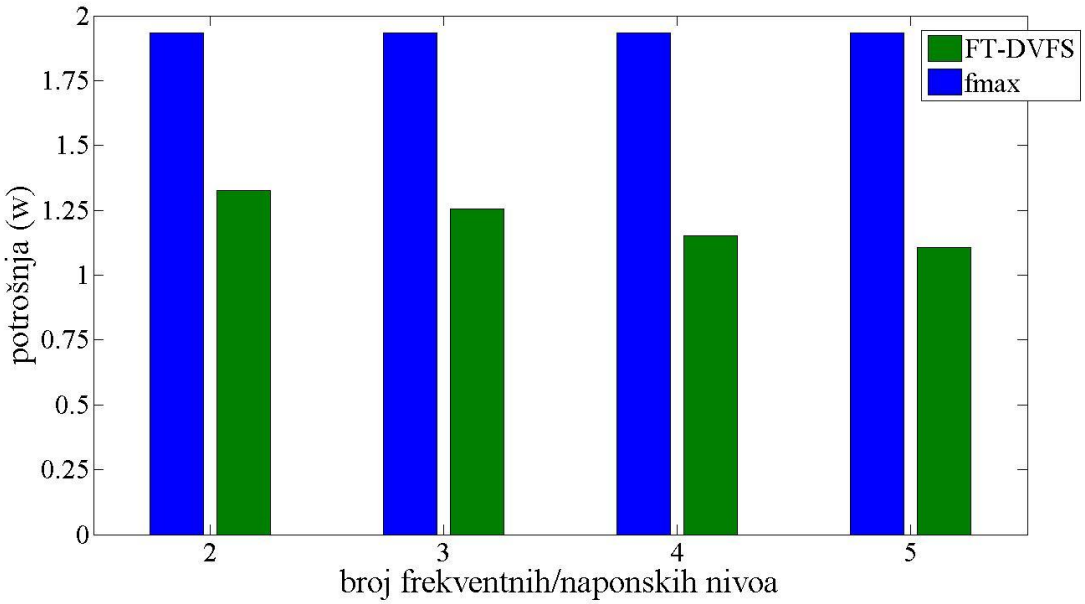


Sl. 7-1: *Ušteda u potrošnji u zavisnosti od broja frekventnih/naponskih nivoa za slučaj RTS bez mogućnosti tolerisanja grešaka*

Na osnovu rezultata prikazanih na Sl. 7-1 može se zaključiti da se primenom FT-DVFS algoritma ostvaruje znatna ušteda u potrošnji procesora čak i korišćenjem manjeg broja dostupnih frekventnih nivoa. Kao što se sa Sl. 7-1 može videti, korišćenjem samo dva frekventna nivoa ostvarena je ušteda u potrošnji procesora od 31.5%. Pod ovakvim uslovima svaki zadatak se izvršava ili pri najvišoj ili pri najnižoj frekvenciji. Ušteda u potrošnji procesora raste sa porastom broja dostupnih frekventnih nivoa, tako da se maksimalna ušteda u potrošnji od 42.8% dobija korišćenjem svih pet dostupnih frekventnih nivoa procesora.

Pošto se radi o GAP RT skupu zadataka i *Transmeta Crusoe* procesoru, odnosno o primeru iz stvarnog sveta, moguće je prikazati kolika je konkretna vrednost potrošnje procesora za dati skup zadataka (Sl. 7-2). I u ovom slučaju broj frekventnih nivoa i, i = 2, …,

5 prikazan je na *x*-osi. Konkretne frekvencije su za i = 2 667MHz i 300MHz, za i = 3 667MHz, 600MHz i 300MHz, za i = 4 667MHz, 600MHz, 533MHz i 300MHz i za i = 5 667MHz, 600MHz, 533MHz, 400MHz i 300MHz. U ovom slučaju *y*-osa predstavlja potrošnju procesora izraženu u vatima (W). Za svaku grupu radnih frekvencija procesora, na Sl. 7-2, prikazana su po dva osenčena pravougaonika. Jedan obojen zeleno koji odgovara potrošnji procesora za GAP RT skup zadataka kada se zadaci izvršavaju na frekvencijama dobijenim primenom FT-DVFS algoritma. Drugi, plavi pravougaonik odgovara potrošnji procesora za GAP RT skup zadataka kada se svi zadaci izvršavaju na maksimalnoj radnoj frekvenciji, odnosno bez primene FT-DVFS algoritma.



Sl. 7-2: *Potrošnja primenom i bez primene FT-DVFS algoritma u zavisnosti od broja frekventnih/naponskih nivoa za slučaj RTS-a bez mogućnosti tolerisanja grešaka*

Rezultati prikazani na Sl. 7-2 su u skladu sa rezultatima prikazanim na Sl. 7-1 i još slikovitije prikazuju koliko se štedi primenom FT-DVFS algoritma. Već sa samo dva frekventna nivoa ušteda je očigledna i kao što se sa Sl. 7-2 može videti ušteda u potrošnji raste sa porastom broja dostupnih frekventnih nivoa. Razlog za rast uštede u potrošnji povećanjem broja radnih frekvencija/napona leži u činjenici da se sa povećanjem broja frekventnih/naponskih nivoa povećava broj vektora dodele frekvencija i omogućava se "finiji" izbor frekvencija rada procesora.

7.2.2 RTS sa mogućnošću tolerisanja grešaka

Tolerisanje grešaka u usvojenom modelu FT RTS-a oslanja se na ponavljanje izvršenja zadatka kod koga je detektovana greška. Nivo tolerantnosti sistema na greške izražava se parametrom T_F koji predstavlja minimalno vreme između dve uzastopne greške koje sistem može da toleriše. T_F kao karakteristika RTS-a zavisi od iznosa vremenske rezerve (redundanse) prisutne u sistemu. U opštem slučaju, povećanje frekvencije na kojoj se zadaci izvršavaju dovodi do povećanja vremenske rezerve, a time i do poboljšanja tolerantnosti na greške što se ogleda kroz manju vrednost parametra T_F. Najviši nivo tolerantnosti se postiže ukoliko se svi zadaci izvršavaju na najvišoj frekvenciji. Vrednost parametra T_F koja važi pod ovakvim uslovima označena je sa T_{Fmax}.

7.2.2.1 Normalizovani nivo tolerancije grešaka

Za potrebe simulacije definisana je promenljiva nazvana normalizovani nivo tolerancije grešaka (eng. *normalized fault tolerance capability*), skraćeno NFTC. Normalizovana vrednost tolerancije grešaka definiše se kao odnos maksimalnog i ciljanog nivoa tolerancije grešaka, odnosno

$NFTC = T_{Fmax}/T_{Fgoal}$.

Treba naglasiti da je T_{Fmax} izračunata vrednost koja odgovara minimalnom vremenskom intervalu između dve uzastopne greške koje se mogu javiti kod RTS-a i koje sistem može da toleriše ali pod pretpostavkom da se svi RT zadaci datog sistema izvršavaju na maksimalnoj radnoj frekvenciji procesora. Sa druge strane T_{Fgoal} je ulazna veličina algoritma i odgovara minimalnom vremenskom intervalu između dve uzastopne greške koje se mogu javiti kod RTS-a i koje bi sistem trebao da toleriše. Znači, kroz T_{Fgoal} su dati zahtevi od strane korisnika vezani za FT ograničenja koja je neophodno ispuniti. Granične vrednosti za normalizovanu vrednost tolerancije grešaka tj. NFTC su 0 i 1.

Ako se pretpostavi da sistem nema mogućnost tolerisanja grešaka, odnosno da je minimalno vremensko vreme između dve moguće greške beskonačno veliko ($T_{Fgoal} \rightarrow \infty$) onda je NFTC = 0. U ovom slučaju FT aspekt se ne uzima u obzir tako da je pažnja usmerena samo na smanjenje potrošnje procesora, odnosno samo na energetsku efikasnost RTS-a.

Slučaj NFTC = 1 dobija se za $T_{Fgoal} = T_{Fmax}$ odnosno za slučaj kada se zahteva maksimalna mogućnost tolerisanja grešaka. U tom slučaju je apsekt energetske efikasnosti stavljen u drugi plana ili moguće i izostavljen.

Ako je zahtevana vrednost za T_{Fgoal} veća od T_{Fmax}, odnosno ako je NFTC između 0 i 1, onda je moguće vremensku redundansu iskoristiti kako za upravljanje potrošnjom procesora tako i za ispunjavanje FT ograničenja. Jasno je da je vremenska redundansa ograničen resurs tako da ako se veći deo redundanse iskoristi za smanjenje potrošnje procesora, manji deo ostaje za ispunjenje zahteva vezanih za FT ograničenja, i obrnuto. Uz pomoć FT-DVFS algoritma može se rešiti dilema koji deo vremenske redundanse koristiti za obezbeđivanje visoke tolerancije grešaka a koji za smanjenje potrošnje procesora. Primarno, FT-DVFS teži da dostigne zadati nivo tolerantnosti na greške, a eventualnu preostalu vremensku rezervu koristi radi smanjenja potrošnje procesora.

7.2.2.2 Ušteda u potrošnji u zavisnosti od NFTC-a

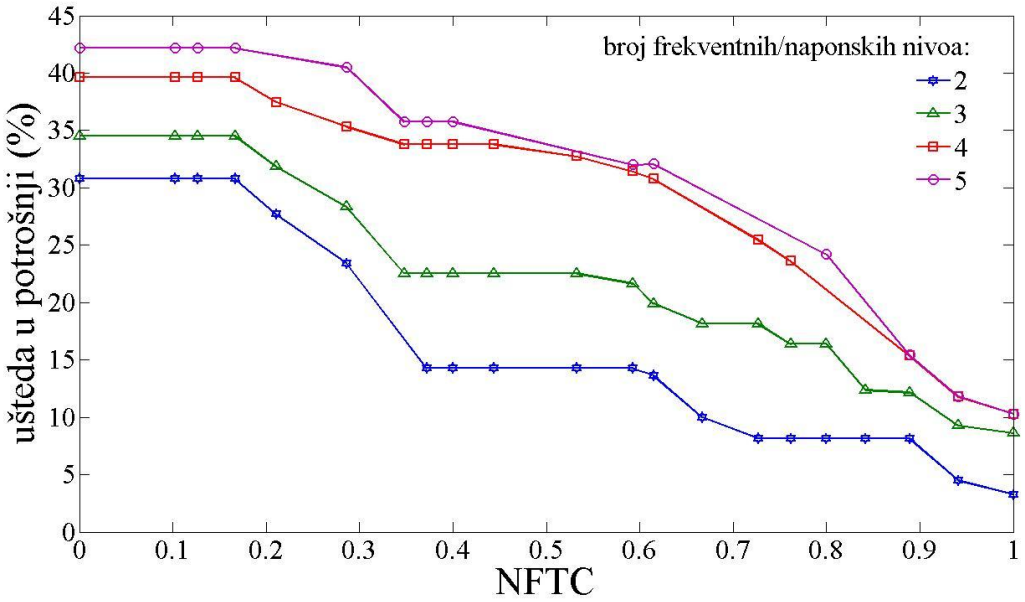
FT-DVFS algoritam primenjen je na GAP RT skup zadataka Tab. 7-1 i *Transmeta Crusoe* procesoru Tab. 7-2, sa ciljem dobijanja frekvencija na kojima procesor treba da izvršava RT zadatke tako da potrošnja procesora bude najmanja moguća i da se istovremeno ispoštuju FT ograničenja.

Na Sl. 7-3 predstavljeni su rezultati simulacije pri čemu je na *x*-osi dat odnos T_{Fmax} i T_{Fgoal} tj. data je vrednost veličine NFTC. Treba naglasiti da je NFTC proporcionalna sposobnosti sistema da prevaziđe otkaz, a kako je sposobnost sistema da prevaziđe otkaz proporcionalna vremenskoj redundansi sistema može se reći da *x*-osa predstavlja i slobodno (redundanto) vreme procesora.

Na *y*-osi data je ušteda u potrošnji procesora izražena u procentima predstavljena kroz odnos potrošnje procesora kada on izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji uz poštovanje zadatih FT ograničenja.

Simulacije su rađene za različit broj dostupnih frekventnih nivoa procesora pri čemu su za 2 frekventna nivoa korišćene radne frekvencije (667MHz, 300MHz), za 3 (667MHz, 600MHz, 300MHz), za 4 (667MHz, 600MHz, 400MHz, 300MHz) i za 5 nivoa (667MHz, 600MHz, 533MHz, 400MHz, 300MHz).

U skladu sa raspoloživim brojem frekventnih nivoa, na Sl. 7-3 mogu se videti četiri grafika. Ono što se prvo može zaključiti i što je zajedničko za sva četiri grafika je činjenica da što je veća vrednost NFTC manja je ušteda u potrošnji procesora, i obrnuto što je ušteda u potrošnji procesora veća to je manja mogućnost prevazilaženja otkaza.



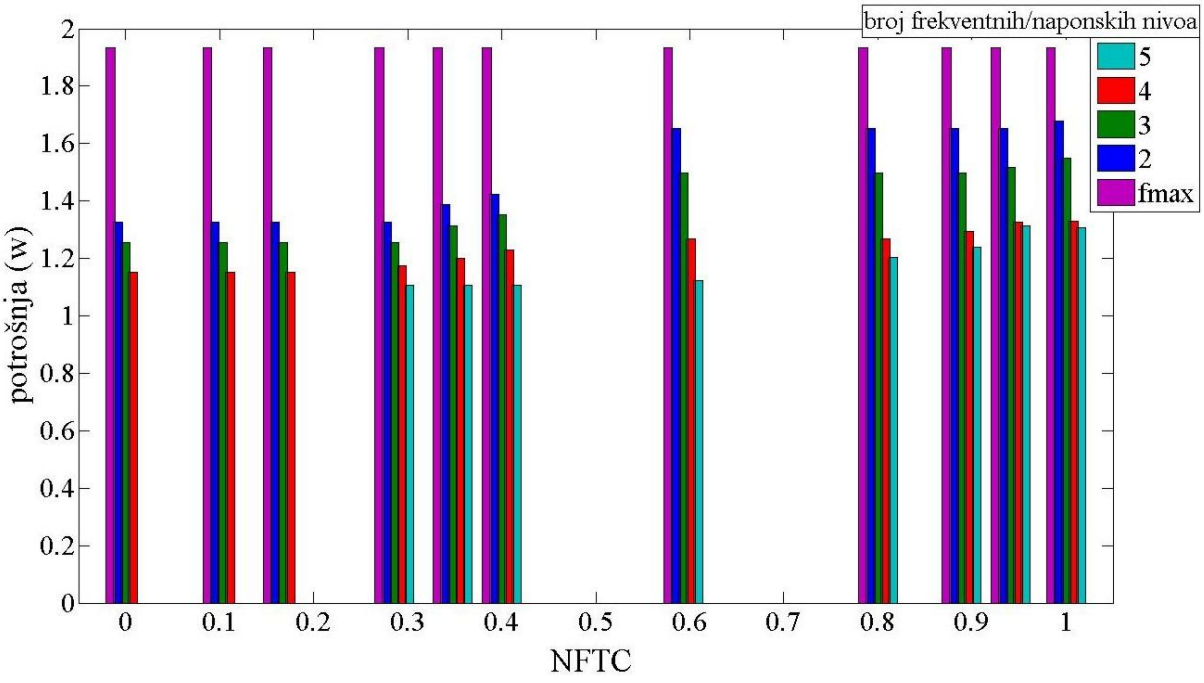
Sl. 7-3: *Ušteda u potrošnji procesora u zavisnosti od NFTC-a za različit broj frekventnih/naponskih nivoa za slučaj RTS-a kod koga se greške mogu javiti*

U skladu sa rezultatima simulacije prikazanim na Sl. 7-3 sada se može preciznije proceniti odnos potrošnje procesora i njegove mogućnosti prevazilaženja otkaza. Recimo, neka se zahteva da ušteda u potrošnji procesora bude između 40% i 45%. Sa Sl. 7-3 može se videti da procesor ako koristi 4 ili 5 frekventnih nivoa ispunjava date zahteve. Kako mogućnost prevazilaženja otkaza varira u okviru zahtevanog intervala smanjenja potrošnje, najbolje je onda izabrati slučaj kada je ta mogućnost maksimalna.

Sa Sl. 7-3 se može videti da je ušteda u potrošnji procesora veća kada se koristi procesor sa više radnih frekvencija. Recimo, za NFTC = 0.5 ušteda u potrošnji procesora može biti 35% ako se koriste sve dostupne frekvencije *Transmeta Crusoe* procesora Tab. 7-2. Ako su procesoru na raspolaganju dva frekventna nivoa (konkretno frekvencije 667MHz i 300MHz) tada je ušteda u potrošnji svega 15%.

Kako se i ovde radi o primeru iz stvarnog sveta, moguće je prikazati kolike su konkretne vrednosti potrošnje procesora (Sl. 7-4). Na Sl. 7-4 *x*-osa predstavlja vrednost veličine NFTC,

dok su na *y*-osi vrednosti potrošnje procesora izraženu u vatima (W). I u ovom slučaju simulacije su rađene za različit broj dostupnih frekventnih/naponskih nivoa procesora pri čemu su za 2 frekventna nivoa korišćene radne frekvencije (667MHz, 300MHz), za 3 (667MHz, 600MHz, 300MHz), za 4 (667MHz, 600MHz, 400MHz, 300MHz) i za 5 nivoa (667MHz, 600MHz, 533MHz, 400MHz, 300MHz). Svakoj grupi frekventnih/naponskih nivoa, na Sl. 7-4, odgovara osenčeni pravougaonik odgovarajuće boje i to: teget za 2 frekventna/naponska nivoa, zeleni za 3, crveni za 4 i svetlo plavi za 5 frekventnih/naponskih nivoa. Ljubičasti pravougaonik odgovara potrošnji procesora za GAP RT skup zadataka kada se svi zadaci izvršavaju na maksimalnoj radnoj frekvenciji, odnosno bez primene FT-DVFS algoritma.



Sl. 7-4: *Potrošnja primenom i bez primene FT-DVFS algoritma u zavisnosti od NFTC-a za različit frekventnih/naponskih nivoa za slučaj RTS-a kod koga se greške mogu javiti*

Rezultati prikazani na Sl. 7-4 su u skladu sa rezultatima prikazanim na Sl. 7-3 i još slikovitije prikazuju koliko se štedi primenom FT-DVFS algoritma. Ljubičasti pravougaonik je znatno viši od ostalih što govori da je potrošnja procesora znatno veća ako nema primene FT-DVFS algoritma. Takođe, može se primetiti da što je veća vrednost NFTC, to je razlika između ljubičastog i ostalih pravougaonika manja, odnosno manja je ušteda u potrošnji procesora. Naravno, važi i obrnuto - za manju vrednost veličine NFTC razlika između ljubičastog i ostalih pravougaonika je veća, tj. ušteda u potrošnji procesora je veća.

Sa Sl. 7-4 može se primetiti da je razlika između ljubičastog i plavog pravougaonika najveća što znači da je ušteda u potrošnji procesora veća kada se koristi procesor sa više radnih frekvencija. I u ovom slučaju razlog za to je činjenica da se sa povećanjem broja frekventnih/naponskih nivoa povećava broj vektora dodele frekvencija i omogućava se "finiji" izbor frekvencija rada procesora.

7.2.3 Poređenje karakteristika algoritma FT-DVFS i metode iscrpljivanja

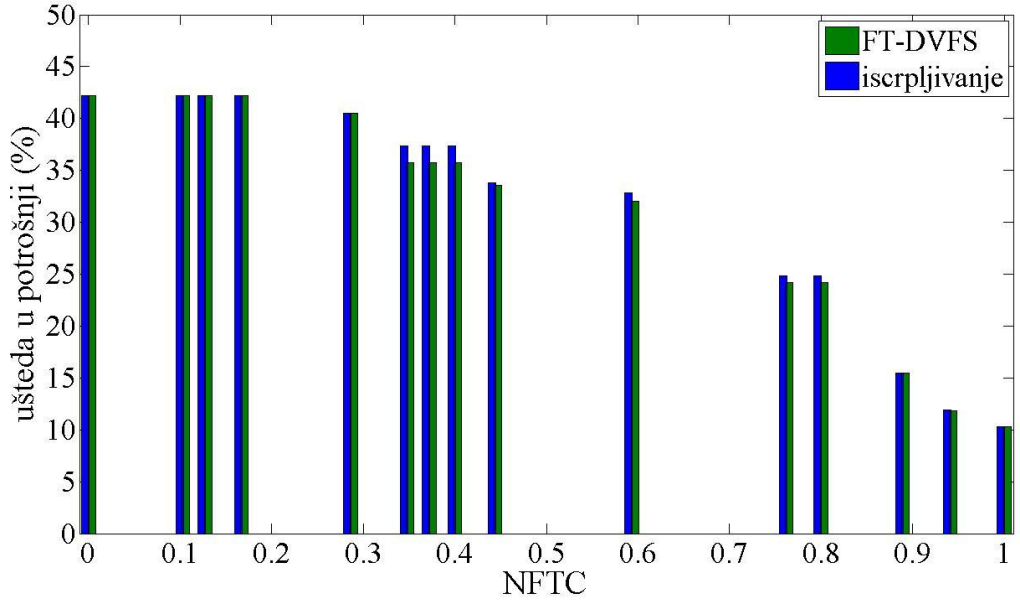
Kako bi se dokazao kvalitet FT-DVFS algoritma izvršeno je njegovo poređenje sa metodom iscrpljivanja opisanim u sekciji 6.3. Kao što je ranije naglašeno, metod iscrpljivanja predstavlja optimalno rešenje za skup sa malim broj RT zadataka. Sa povećanjem broja RT zadataka i broja frekventnih/naponskih nivoa procesora ovaj metod ne može dati rešenje u zadovoljavajućem vremenskom periodu.

Za potrebe poređenja karakteristika FT-DVFS algoritma i metode iscrpljivanja korišćen je GAP RT skup zadataka Tab. 7-1 i *Transmeta Crusoe* procesor Tab. 7-2. Primenjujući oba algoritma došlo se do rezultata vezanih za uštedu u potrošnji procesora prikazanih na Sl. 7-5.

Na Sl. 7-5 *x*-osa predstavlja odnos T_{Fmax} i T_{Fgoal} tj. data je vrednost veličine NFTC. Kao što je ranije pomenuto T_{Fmax} je izračunata vrednost koja odgovara minimalnom vremenskom intervalu između dve uzastopne greške koje se mogu javiti kod GAP RT skupa zadataka i koje GAP RTS može da toleriše, pod pretpostavkom da se svi RT zadaci datog sistema izvršavaju na maksimalnoj radnoj frekvenciji procesora. U okviru ove simulacije korišćeno je svih 5 raspoloživih frekventnih/naponskih nivoa *Transmeta Crusoe* procesora. Parametar T_F je ulazna veličina simulacije kroz koju se zadaju FT ograničenja. Simulacije su odrađene za različite vrednosti parametra T_F.

Na *y*-osi grafikona sa Sl. 7-5 data je ušteda u potrošnji procesora izražena u procentima. Ova osa predstavlja odnos potrošnje procesora kada on izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma odnosno metode iscrpljivanja i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji uz poštovanje zadatih FT ograničenja.

Zeleni pravougaonici predstavljaju rezultate simulacija FT-DVFS algoritma dok su teget pravougaonici rezultat primene metode iscrpljivanja. Ono što se može zaključiti na osnovu rezultata prikazanih na Sl. 7-5 jeste da FT-DVFS algoritam u većini slučajeva generiše skoro optimalne rezultate.



Sl. 7-5: *Poređenje karakteristika algoritma FT-DVFS i algoritma „metod iscrpljivanja“*

Rezultati su posebno dobri za slučaj T_{Fmax} / T_F < 0.3, kada je fokus na smanjenju potrošnje procesora odnosno kada se veći deo redundantnog vremena troši na poboljšanje energetske efikasnosti RTS-a. Takođe su rezultati dobri za slučaj T_{Fmax} / T_F > 0.8 kada je fokus na toleranciji grešaka odnosno kada se veći deo redundantnog vremena troši na poboljšanje FT karakteristika RTS-a. U ovim slučajevima rezultati su identični odnosno u ovim slučajevima FT-DVFS postiže optimalne rezultate.

Nešto lošiji rezultati se vezani za oblast 0.3< T_{Fmax} / T_F < 0.8 kada se već javlja dilema da li veći deo redundantnog vremena potrošiti na poboljšanje FT karakteristika ili na smanjenje potrošnje procesora. Iako u ovom slučaju rezultati nisu najbolji opet je odstupanje vezano za smanjenje potrošnje procesora kod FT-DVFS algoritma u odnosu na optimalni metod iscrpljivanja svega 5%.

Sa druge strane, ako se vrši poređenje vezano za vreme izvršenja onda je FT-DVFS algoritam u velikoj prednosti u odnosu na metod iscrpljivanja. Vreme koje je potrebno FT-DVFS algoritmu da dođe do izlaznih podataka za GAP RT skup zadataka i *Transmeta Crusoe* procesor je manje od 1s. Za isti RT skup zadataka i isti broj dostupnih frekventnih/naponskih nivoa metod iscrpljivanja zahteva vreme od nekoliko sati.

7.3 Sintetički procesor i RT skup zadataka korišćen u simulaciji

Model sintetičkog procesora korišćen u realizovanom simulatoru bazira se na procesoru koji u sebi ima ugrađene DVS karakteristike i mogućnost izbora broja frekventnih/naponskih nivoa kao i mogućnost izbora opsega radnih frekvencija tj. minimalne i maksimalne frekvencije rada.

Maksimalna frekvencija rada je normalizovana na 1 i ona je u simulacijama fiksna veličina, dok se minimalna frekvencija bira iz opsega (0, 1). Za *i* frekventnih/naponskih nivoa vrednosti radnih frekvencija procesora su frekvencije iz opsega $[f_{min}, f_{max}]$ sa korakom Δf definisan jednačinom (7-1):

$$\Delta f = \frac{f_{max} - f_{min}}{i - 1} \tag{7-1}$$

Za potrošnju procesora iskorišćen je model, koji se često koristi u radovima [Riz11a], [Zom12], [Riz11b], u okviru koga je potrošnja procesora srazmerna f^3 tj.

$P = \lambda f^3$.

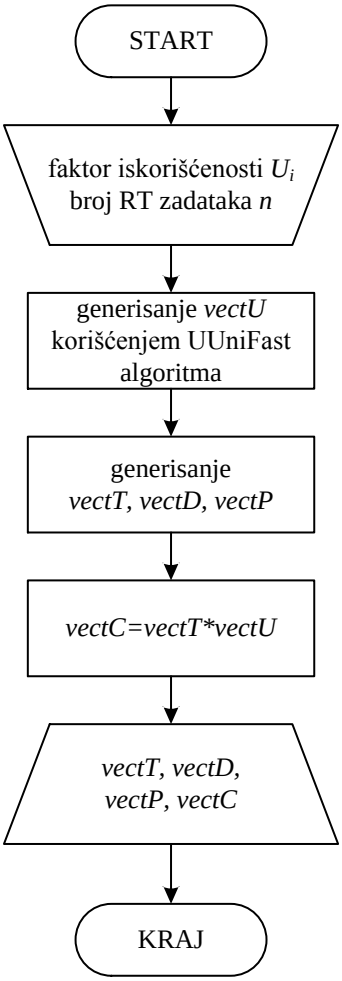
Sintetički skupovi RT zadataka korišćeni u ovoj sekciji kreirani su po uzoru na već korišćene sintetičke skupove koji se mogu naći u literaturi koja obrađuje teme vezane za izvodljivost RTS-a [Ays07], [Dav08], [Dob08].

Zadatak τ_i, i = 1, .., n u sintetičkom skupu zadataka predstavljen je uređenom trojkom (C_i, T_i, p_i) gde je C_i vreme izvršenja zadataka kada procesor radi na maksimalnoj radnoj frekvenciji, T_i je period a p_i prioritet RT zadatka. Vremenski parametri zadatka, C_i i T_i, biraju se u osnovi na slučajan način, ali tako da se ispune unapred postavljeni uslovi koji se tiču određene karakteristike skupa zadataka kao celine. Sa stanovišta analize izvodljivosti i procene performansi algoritma FT-DVSF, najvažnija karakteristika skupa zadataka je iskorišćenost procesora, *U*, koja se definiše kao

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{7-2}$$

Iskorišćenost procesora je manja ili jednaka 1 i pokazuje koliko deo vremena je procesor aktivno uposlen izvršavanjem RT zadataka. Posredno, veličina (1-*U*) ukazuje na iznos slobodnog vremena procesora, odnosno vremena koje se može koristiti za uštedu energije i/ili

poboljšanje sposobnosti tolerisanja grešaka. Veličina u_i predstavlja „faktor iskorišćenosti zadatka“ i ukazuje na to koliki deo procesorskog vremena treba rezervisati za izvršenje zadatka τ_i.



Sl. 7-6: *Algoritam generisanja skupa RT zadataka*

U okviru softverskog paketa Matlab realizovan je program za generisanje skupa RT zadataka prikazan na Sl. 7-6. Program kreće od dve poznate veličine: broja RT zadataka n i faktora iskorišćenosti procesora U. Prvi korak u proceduri generisanja sintetičkog skupa zadataka jeste određivanje vrednosti u_i tako da jednačina (7-2) bude zadovoljena. U okviru ovog koraka korišćen je *UUniFast* algoritam [Bin05], [Dav08], na osnovu koga se dolazi do vrednosti faktora iskorišćenosti svakog od n zadataka, u_i. Osnova *UUniFast* algoritma leži u Matlab kodu prikazanom na Sl. 7-7. Ukratko, *UUniFast* algoritam prvo, na određen način, generiše slučajnu veličinu, a zatim računa faktor iskorišćenosti pojedinačnog zadatka kao razliku trenutne vrednosti faktora iskorišćenosti procesora i generisane slučajne veličine. U

prvoj iteraciji trenutna vrednost faktora iskorišćenosti procesora je upravo veličina *U* dok u svakoj narednoj iteraciji ona uzima vrednost generisane slučajne veličine. Ova petlja izvršava se *n*-1 puta dok se za zadnju vrednost faktora iskorišćenosti pojedinačnog zadatka uzima poslednja vrednost generisane slučajne veličine. Na ovaj način generiše se *vectU* odnosno vrednosti faktora iskorišćenosti svakog od *n* zadataka u_i.

Nakon generisanja *vectU* sledi generisanje *vectT* tj. perioda RT zadataka, T_i. Periodi zadataka, T_i predstavljaju slučajno izabrane vrednosti iz intervala [1, 1000] koje imaju uniformnu raspodelu. Takođe, svakom zadatku dodeljuje se rok za izvršenje D_i tako da je period zadatka jednak njihovom roku za izvršenje

$T_i = D_i$.

Na taj način generiše se *vectD* dok se *vectP* odnosno prioriteti p_i RT zadataka dodeljuju na osnovu EDF algoritma za raspoređivanje zadataka.

```
function vectU = UUniFast(n, U)
sumU = U;
    for i=1:n−1,
    nextSumU = sumU.*rand^(1/(n−i));
    vectU(i) = sumU − nextSumU;
    sumU = nextSumU;
end
vectU(n) = sumU;
```

Sl. 7-7: *Matlab kod UUniFast algoritma*

Na osnovu jednačine (7-2) lako se mogu izračunati vrednosti vremena izvršenja RT zadataka C_i, odnosno *vectC* kao

$$C_i = T_i \cdot U_i.$$

Vreme izvršenja RT zadataka C_i odgovara vremenu izvršenja zadatka kada procesor radi na maksimalnoj frekvenciji.

Konkretno, u okviru simulacija korišćeni su skupovi koji se sastoje od 10 RT zadataka dok su sve simulacije rađene na bazi 100 RT skupova.

7.3.1 RTS bez mogućnosti tolerisanja grešaka

Prve simulacije rađene su za pretpostavku da RTS nema mogućnosti tolerisanja grešaka. Usvajajući tu pretpostavku, FT-DVFS algoritam primenjen je na model sintetičkog procesora i RT skup zadataka (generisan na osnovu algoritma Sl. 7-6) sa ciljem dobijanja frekvencija na kojima procesor treba da izvršava RT zadatke tako da potrošnja procesora bude najmanja moguća. Ovde se zapravo radi o slučaju kada je minimalno vreme između dve moguće pojave grešaka beskonačno veliko tj. kada $T_{Fgoal} \rightarrow \infty$. Ovaj slučaj ostavlja mogućnost da se celokupno slobodno vreme procesora koristi za smanjenje njegove potrošnje.

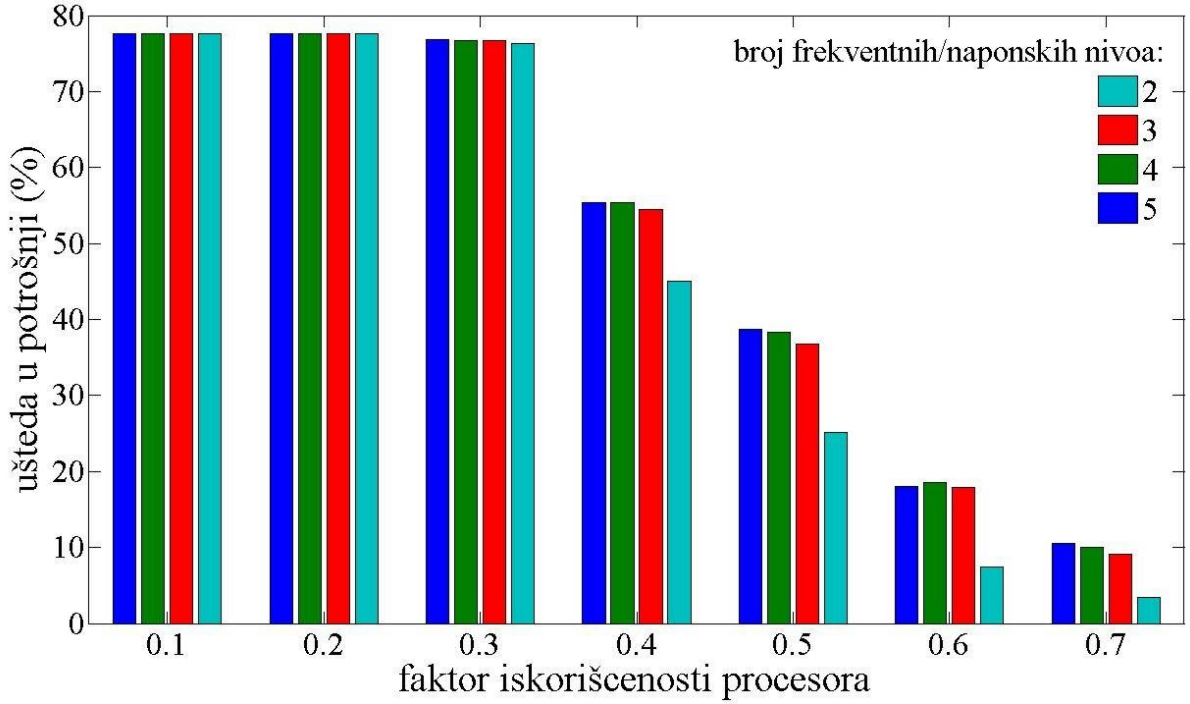
Ušteda u potrošnji u zavisnosti od broja frekventnih nivoa

Sl. 7-8 predstavlja rezultate simulacija prikazane u vidu uštede u potrošnji procesora u funkciji faktora iskorišćenosti procesora *U* za različit broj frekventnih/naponskih nivoa.

Na *x*-osi nalaze se vrednosti faktora iskorišćenosti procesora od $U = 0.1$ do $U = 0.7$, dok *y*-osa predstavlja odnos potrošnje procesora kada izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji izražena u procentima.

Vrednost maksimalne radne frekvencija procesora iznosi $f_{max} = 1$, dok je za minimalnu radnu frekvenciju, u ovom slučaju, uzeta vrednost $f_{min} = 0.45$. Upravo ove dve vrednosti korišćene su kao frekvencije rada procesora za 2 frekventna/naponska nivoa. Za $i = 3$, 4 i 5 frekventnih/naponskih nivoa korišćene su frekvencije iz opsega [0.45, 1] sa korakom Δf definisan jednačinom (7-1). Konkretno, za 3 nivoa korišćene su frekvencije 0.45, 0.725 i 1, za 4 nivoa frekvencije 0.45, 0.633, 0.816 i 1 i za 5 nivoa frekvencije 0.45, 0.5875, 0.725, 0.8625 i 1.

Na osnovu rezultata prikazanih na Sl. 7-8 može se zaključiti da se primenom FT-DVFS algoritma ostvaruje ušteda u potrošnji procesora čak i za velike vrednosti faktora iskorišćenosti procesora tj. za $U = 0.6$ i $U = 0.7$. Za male vrednosti faktora iskorišćenosti procesor ušteda u potrošnji je izrazito velika, između 70% i 80% što je i logično jer postoji velika vremenska redundansa koja se može iskoristiti za izvršenje zadataka na najnižim mogućim frekvencijama. Kako faktor iskorišćenosti procesora raste sve je manje slobodnog vremena procesora koje može iskoristiti FT-DVFS algoritam pa je i ušteda u potrošnji manja, između 20% i 60%. Za velike vrednosti faktora iskorišćenosti procesora ima najmanje redundantnog vremena tako da je ušteda u potrošnji najmanja, oko 10%.



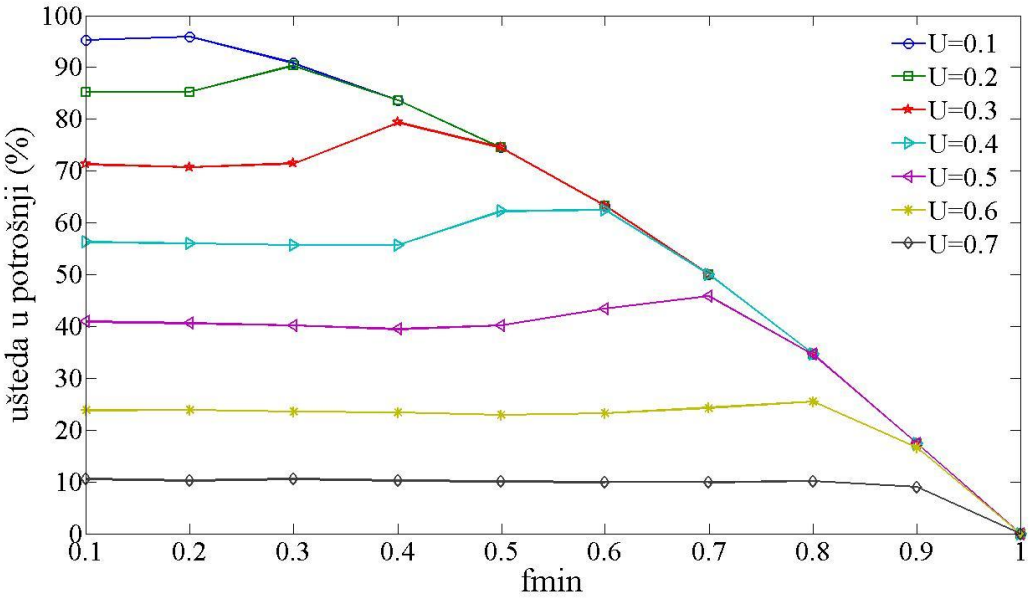
Sl. 7-8: *Ušteda u potrošnji u funkciji faktora iskorišćenosti za različit broj frekventnih/naponskih nivoa kod RTS-a bez mogućnosti tolerisanja grešaka*

Za svaku vrednost faktora iskorišćenosti procesora rađene su simulacije za različit broj frekventnih/naponskih nivoa. Na Sl. 7-8 teget pravougaonikom predstavljeni su rezultati za 5 frekventnih/naponskih nivoa, zelenim pravougaonikom za 4, crvenim za 3 i svetlo plavim za 2 frekventno/naponska nivoa. Sa Sl. 7-8 može se zaključiti da ušteda u potrošnji procesora raste sa porastom broja dostupnih frekventnih/naponskih nivoa. Razlog za rast uštede u potrošnji povećanjem broja dostupnih radnih frekvencija/napona leži u činjenici da se sa povećanjem broja frekventnih/naponskih nivoa povećava broj vektora dodele frekvencija i omogućava se "finiji" izbor frekvencija rada procesora.

Ušteda u potrošnji u zavisnosti od veličine frekventnog opsega

Simulacije su rađene i za slučaj uticaja širine frekventnog opsega na potrošnju pri čemu je maksimalna frekvencija fiksirana, dok se minimalna radna frekvencija menja. Na Sl. 7-9 prikazani su rezultati simulacija prikazani u vidu uštede u potrošnji procesora u funkciji njegove minimalne radne frekvencije f_{min} za različite vrednosti faktora iskorišćenosti procesora U.

Na Sl. 7-9 *y*-osa predstavlja odnos potrošnje procesora kada izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji izražena u procentima. Na *x*-osi nalaze se vrednosti minimalne radne frekvencije f_{min} iz opsega [0.1, 1] sa korakom 0.1. Simulacije su rađene za slučaj različite vrednosti faktora iskorišćenosti procesora *U*, konkretno za opseg [0.1, 0.7] sa korakom 0.1.



Sl. 7-9: *Ušteda u potrošnji u funkciji minimalne radne frekvencije za različite vrednosti faktora iskorišćenosti procesora kod RTS-a bez mogućnosti tolerisanja grešaka*

Ako se grafikon posmatra s desna na levo, odnosno od maksimalne frekvencije ka nižim frekvencijama, uočava se da ušteda raste, a zatim ulazi u zasićenje. Porast uštede je posledica mogućnosti izvršenja zadataka na nižim frekvencijama. Zasićenje nastaje kada se iskoristi svo raspoloživo slobodno vreme. Što je *U* veće, to je i iznos slobodnog vremena veći, što ostavlja više prostora za smanjivanje frekvencije izvršenja zadataka, a time i veću uštedu.

Takođe, treba naglasiti da širina upotrebljivog frekventnog opsega neposredno zavisi od faktora iskorišćenosti procesora. Na primer, sa slike se vidi da je za U=0.7, dovoljan frekventni opseg [0.9, 1], odnosno da dalje širenje ovog opsega ne doprinosi uštedi. S druge strane, za *U*=0.1, potreban je daleko širi frekventni opseg, [0.2, 1] da bi se u punoj meri iskoristilo slobodno vreme radi uštede.

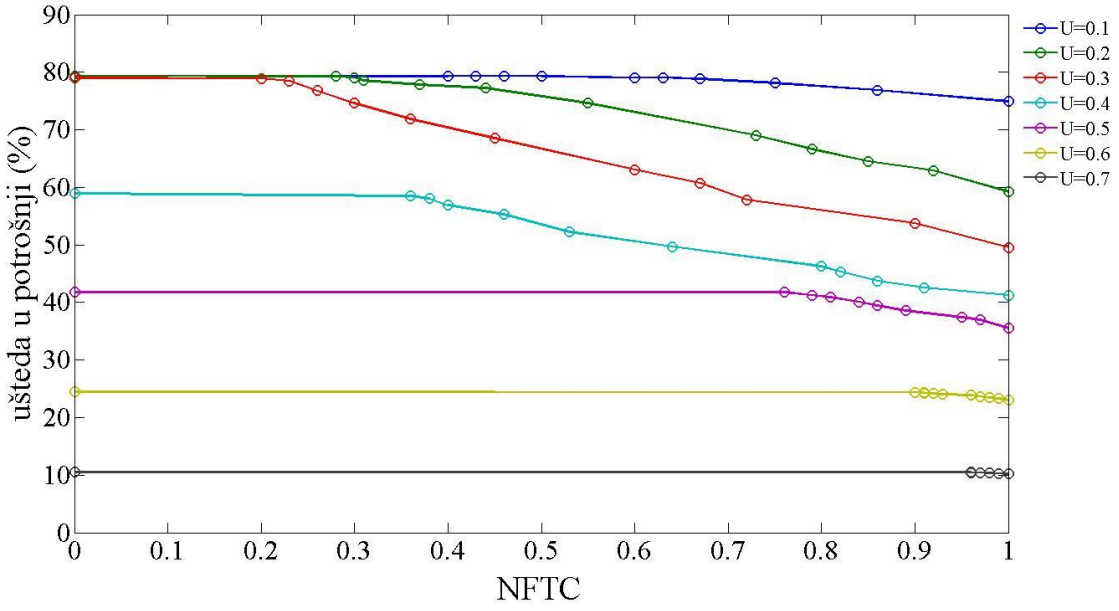
7.3.2 RTS sa mogućnošću tolerisanja grešaka

Druga grupa simulacija rađena za sintetički procesor i sintetički skup zadataka odnosi se na slučaj kada RTS ima mogućnost tolerisanja grešaka. I u ovom slučaju tolerisanje grešaka oslanja se na ponavljanje izvršenja zadatka kod koga je detektovana greška dok se nivo tolerantnosti sistema na greške izražava parametrom T_F. Takođe, u okviru ove grupe simulacija korišćena je promenljiva NFTC tj. normalizovani nivo tolerancije grešaka detaljno objašnjen u sekciji 7.1.2.1.

Ušteda u potrošnji u zavisnosti od NFTC

Na Sl. 7-10 prikazani su rezultati simulacija dobijenih primenom FT-DVFS algoritma na RT skupove zadataka sa različitim vrednostima faktora iskorišćenosti procesora. Konkretno prikazana je zavisnost uštede u potrošnji procesora u funkciji parametra NFTC.

Na Sl. 7-10 *x*-osa predstavlja odnos T_{Fmax} i T_{Fgoal} tj. vrednost veličine NFTC. Na *y*-osi data je ušteda u potrošnji procesora izražena u procentima koja, kao i u prethodnim simulacijama, predstavlja odnos potrošnje procesora kada on izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji uz poštovanje zadatih FT ograničenja.



Sl. 7-10: *Ušteda u potrošnji u funkciji parametra NFTC za različite vrednosti faktora iskorišćenosti procesora kod RTS-a sa mogućnošću tolerisanja grešaka*

Simulacije su rađene za 10 dostupnih frekventnih nivoa sintetičkog procesora iz opsega [0.45, 1] sa korakom Δf definisan jednačinom (7-1). U simulacijama su korišćeni RT skupovi zadataka sa različitim faktorom iskorišćenosti procesora U od 0.1 do 0.7 sa korakom 0.1.

Na Sl. 7-10 mogu se videti 7 grafika koji odgovaraju različitim vrednostima parametra U. Ono što se prvo može zaključiti i što je zajedničko za svih sedam grafika je činjenica da što je veća vrednost NFTC to je manja ušteda u potrošnji procesora, i obrnuto što je ušteda u potrošnji procesora veća to je manja mogućnost prevazilaženja otkaza. Rezultati su logični i očekivani.

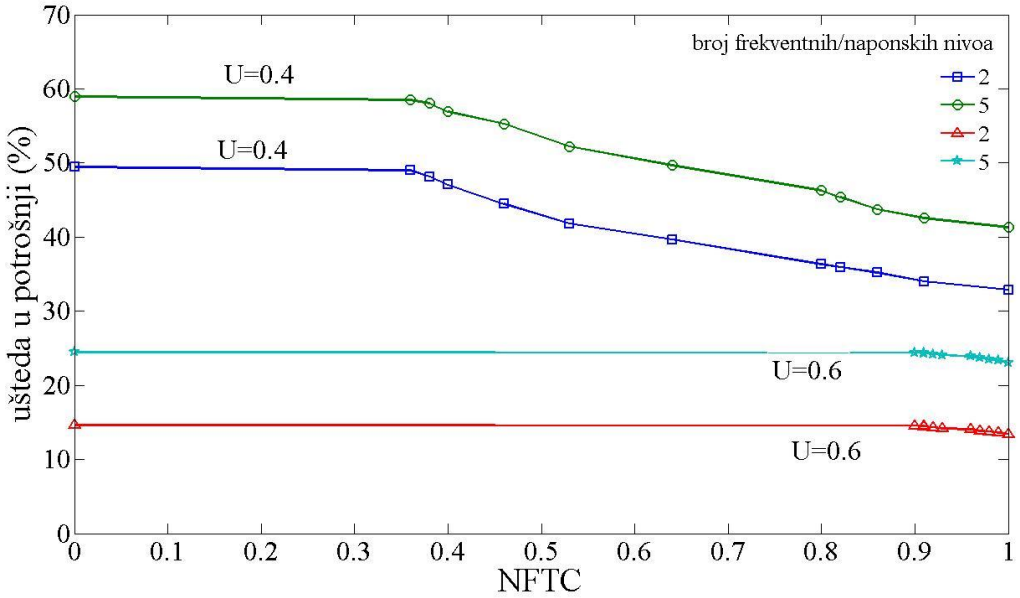
Recimo za NFTC = 0 (slučaj kada $T_{Fgoal} \rightarrow \infty$ tj. kada je minimalni vremenski interval između dve moguće greške beskonačno veliko) FT aspekt se ne uzima u obzir tako da je pažnja usmerena samo na smanjenje potrošnje procesora tako da je ušteda u potrošnji najveća. Sa povećanjem parametra NFTC FT aspekt se uzima u obzir i FT-DVFS algoritam odlučuje koji deo vremenske redundanse da bude iskorišćen za obezbeđivanje visoke tolerancije grešaka, a koji deo da bude iskorišćen za smanjenje potrošnje procesora. Zbog toga se na Sl. 7-10 može videti da se ušteda u potrošnji procesora smanjuje. Ušteda u potrošnji procesora je najmanja za NFTC = 1 (slučaj kada je $T_{Fgoal} = T_{Fmax}$) tj. kada se zahteva maksimalna mogućnost tolerisanja grešaka jer je u tom slučaju apsekt energetske efikasnosti stavljen u drugi plana ili moguće i izostavljen.

Sa Sl. 7-10 može se videti da se povećanjem faktora iskorišćenosti procesora smanjuje ušteda u potrošnji što je i očekivano jer sa porastom parametra U ostaje sve manje slobodnog vremena koje FT-DVFS algoritam može iskoristiti za uštedu u potrošnji. Istovremeno, povećanjem faktora iskorišćenosti procesora smanjuje se opseg vrednosti parametra NFTC za koje je RTS izvodljiv. Na graficima se ovo vidi kao različito grupisane tačke koje su za malo U (U = 0.1, U = 0.2 i U = 0.3) raspoređene duž celog grafika, dok su za veliko U (U = 0.6 i U = 0.7) grupisane na grafiku u intervalu parametra NFTC [0.9, 1].

Ušteda u zavisnosti od broja frekventnih nivoa

Na Sl. 7-11 prikazani su rezultati simulacija rađeni za slučaj kada imamo na raspolaganju procesor sa različitim brojem dostupnih frekventnih/naponskih nivoa. Konkretno, korišćene su frekvencije sintetičkog procesora 0.45 i 1 za 2 nivoa i frekvencije 0.45, 0.5875, 0.725, 0.8625 i 1 za 5 nivoa.

Na *x*-osa nalaze se vrednosti veličine NFTC dok je na *y*-osi data ušteda u potrošnji procesora izražena u procentima koja, kao i u prethodnim simulacijama, predstavlja odnos potrošnje procesora kada on izvršava svaki zadataka na frekvenciji dobijenoj primenom FT-DVFS algoritma i potrošnje procesora kada izvršava sve zadatke na maksimalnoj radnoj frekvenciji uz poštovanje zadatih FT ograničenja.



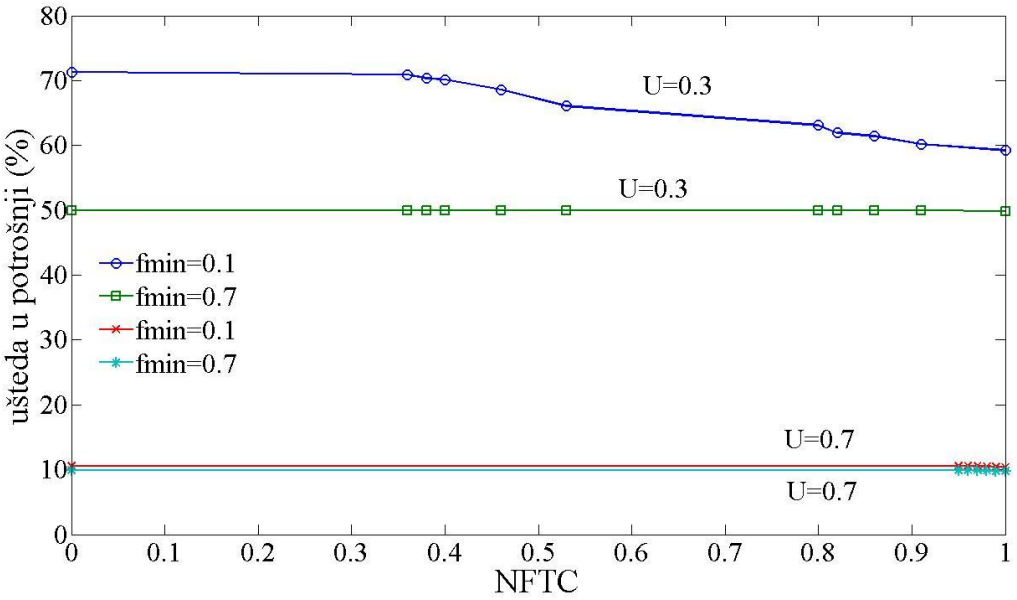
Sl. 7-11: *Ušteda u potrošnji u funkciji parametra NFTC za različit broj frekventnih/naponskih nivoa kod RTS-a sa mogućnošću tolerisanja grešaka*

Simulacije čiji su rezultati prikazani na Sl. 7-11 rađene su za 2 vrednosti faktora iskorišćenosti procesora i to za $U = 0.4$ i $U = 0.6$. U oba slučaja ušteda u potrošnji procesora je veća kada se koristi procesor sa više radnih frekvencija. Ušteda u potrošnji je između 50% i 60% za faktor iskorišćenosti procesora $U = 0.4$ odnosno između 15% i 25% za $U = 0.6$. I u ovom slučaju razlog za to je činjenica da se sa povećanjem broja frekventnih/naponskih nivoa povećava broj vektora dodele frekvencija i omogućava se "finiji" izbor frekvencija rada procesora.

Takođe, povećanjem faktora iskorišćenosti procesora povećava se minimalno NTFC za koje je skup zadataka izvodljiv što znači da je otpornost RTS-a na greške manja što je i očekivano jer ostaje manje slobodnog vremena koje je moguće iskoristiti za oporavak sistema nakon pojave greške.

Ušteda u zavisnosti od širine frekventnog opsega

Na Sl. 7-12 prikazani su rezultati simulacija rađeni za slučaj kada imamo na raspolaganju procesor sa različitim frekventnim opsezima i skupove RT zadataka sa različitim faktorima iskorišćenosti procesora. I na Sl. 7-12 *x*-osa predstavlja odnos T_{Fmax} i T_{Fgoal} tj. vrednost veličine NFTC. Na *y*-osi data je ušteda u potrošnji procesora izražena u procentima.



Sl. 7-12: *Ušteda u potrošnji u funkciji parametra NFTC za različite vrednosti minimalne radne frekvencije kod RTS-a sa mogućnošću tolerisanja grešaka*

Simulacije na Sl. 7-12 rađene su za dve vrednosti minimalne radne frekvencije f_{min} procsora i to za f_{min} = 0.1 i f_{min} = 0.7, dok je za vrednost maksimalne radne frekvencije uzeto f_{max} = 1. Procesor sa ovakvim karakteristikama iskorišćen je za izvršenje dve grupe od po 100 RT skupova zadataka koje se razlikuju po stepenu iskorišćenja procesora. Konkretno korišćeni su RT skupovi zadataka čije je U = 0.3 i U = 0.7.

Na Sl. 7-12 mogu se videti 4 grafika, teget i zeleni su za RT skup zadataka čije je U = 0.3 dok su crveni i svetlo plavi grafici za RT skup zadataka čije je U = 0.7. Za obe grupe grafika može se zaključiti da se primenom FT-DVFS algoritma ušteda u potrošnji procesora opada sa povećanjem njegove minimalne radne frekvencije f_{min}. Sa Sl. 7-12 može se videti da je taj gubitak u uštedi (od preko 20%) izraženiji kod RT skupova zadataka sa manjim faktorom iskorišćenosti procesora U = 0.3, dok je kod RT skupova zadataka čije je U = 0.7 gubitak u

uštedi svega par procenata. Rezultati su očekivani jer kod RT skupova zadataka sa malim faktorom iskorišćenosti procesora ima dovoljno slobodnog vremena tako da se zadaci mogu izvršavati na najnižim mogućim frekvencijama. Kod RT skupova zadataka sa velikim faktorom iskorišćenosti procesora promene vezane za uštedu u potrošnji nisu izražene jer se uglavnom svo slobodno vreme procesora troši prvenstveno na ispunjenje zahteva vezanih za toleranciju grešaka.

Takođe, ušteda u potrošnji procesora opada sa povećanjem faktora iskorišćenosti procesora. Istovremeno, povećanjem faktora iskorišćenosti procesora smanjuje se opseg vrednosti parametra NFTC za koje je RTS izvodljiv. Na graficima se ovo vidi kao različito grupisane tačke koje su za $U = 0.3$ raspoređene duž celog grafika, dok su za $U = 0.7$ grupisane u intervalu $[0.9, 1]$.

8 Zaključak

Koncept sistema za rad u realnom vremenu (RTS) prisutan je u računarstvu već nekoliko decenija unazad. U osnovi, RTS-i su računarski sistemi koji su u neposrednoj interakciji sa fizičkim okruženjem. Ono što razlikuje RTS-e od drugih elektronskih sistema jeste činjenica da je pored logičke podjednako bitna i vremenska ispravnost generisanih rezultata, tj. potrebno je da odziv RTS-a na spoljašnju pobudu bude pripremljen i dostavljen okruženju u pravom trenutku. U suprotnom, kašnjenje odziva, koje premašuje unapred definisanu graničnu vrednost, može da dovede do smanjenja kvaliteta, prestanka rada RTS-a ili čak do pojave grešaka katastrofalnih po okolinu i ljude.

Primena RTS-a evaluirali je od sistema koji su pretežno razvijani za specijalne namene (kontrola leta, upravljanje projektilima, nadzor industrijskih sistema i sl.), do sistema koji se danas, u najrazličitijim oblicima, prisutni svuda oko nas (igračke, kućni aparati, mobilni telefoni i sl.). S obzirom na ubrzani tempo razvoja i širenje primene RTS-a može se sa sigurnošću tvrditi da će se u budućnosti broj ovih sistema povećavati pa će i potreba za njihovo projektovanje i razvoj rasti. Više znanja o takvim sistemima je svakako potrebno inženjerima koji će se u takve projekte upuštati.

Nove raznorodne oblasti primene RTS-a dovele su do toga da su pojedini projektantski zahtevi koji su ranije zanemarivani ili smatrani sekundarnim danas izbili u prvi plan. To se pre svega odnosi na sve strožije zahteve koji se tiču pouzdanosti i potrošnje energije. Potreba za projektovanjem RTS-a sa garantovanim nivoom pouzdanosti i minimalnom potrošnjom energije zahteva modifikaciju postojećih i razvoj novih metodologija projektovanja. Ova disertacije je upravo odgovor na pitanje kako u okvirima strogih vremenskih ograničenja naći

kompromis između, često oprečnih, zahteva vezanih za energetsku efikasnost i zahteva vezanih za povećanje tolerantnosti na greške.

Problematika tolerisanja otkaza je u disertaciji obrađivana sa aspekta tipova otkaza i načina njihovog prevazilaženja prilagođeno specifičnostima RTS-a. Između trajnih, povremenih i prolaznih otkaza posebna pažnja posvećena je prolaznim otkazima koji se, kao posledica smetnji i interferencija iz fizičkog okruženja, danas smatraju najzastupljenijim tipom otkaza. Mnoga istraživanja pokazala su da je verovatnoća pojave prolaznih otkaza 10 do 50 puta veća od mogućnosti pojave stalnih otkaza. Kao metod za uspešno tolerisanje grešaka nastalih usled prolaznih otkaza u disertaciji je između hardverske, softverske, informacione i vremenske redundanse izabrana vremenska. Prednost vremenskoj redundansi je data iz više razloga, a prvenstveno zbog činjenice da ne zahteva dodatni hardver i da je relativno jeftino rešenje. U okviru disertacije posebno su razmatrane dve tehnike vremenske redundanse: tehnika kontrolnih tačaka i tehnika ponovljenog izvršavanja zadatka. Između ove dve tehnike prednost je data tehnici ponovljenog izvršavanja zadatka jer se relativno jednostavno implementira i ne otežava značajno analizu rada RTS-a.

U okviru aspekta vezanog za tolerisanje grešaka paralelno je razmatran i vremenski aspekt gde su proučavane tehnike za analizu rada RTS-a sa stanovišta izvodljivosti zadataka. Posebna pažnja u disertaciji posvećena je metodi koja je poznata pod nazivom analiza vremena odziva (RTA). Prednost ove metode je da ne zavisi od izabranog algoritma za dodelu prioriteta zadacima što je čini veoma pogodnom za široku primenu. U okviru disertacije predstavljeno je više modifikacija analize vremena odziva u cilju prilagođenja ove metode različitim tehnikama za oporavak sistema nakon detektovane greške. Takođe, zbog kompleksnosti računice analize vremena odziva za veći skup zadataka, realizovan je program za izračunavanje vremena odziva u okviru softverskog paketa Matlab. Programa omogućava da se za veće skupove zadataka brzo i efikasno izračuna najmanje vreme između moguće pojave dve uzastopne greške (T_{Fmin}) koje dati RTS može da prevaziđe i nastavi sa normalnim funkcionisanjem. Takođe je modifikovana osnovna verzija RTA za slučaj oporavka od greške izvršavanjem alternativnog zadatka. Rešenje ovog problema uzima u obzir činjenicu da alternativni zadaci, čija je funkcija da nakon detektovane greške postave sistem u bezbedno stanje, po pravilu imaju kraće vreme izvršenja od primarnih zadataka RTS-a. U disertaciji je predstavljena i modifikacija RTA za slučaj kada u RTS-u postoje zadaci kritični po pitanju bezbednosti sistema koji u slučaju pojave grešaka zahtevaju momentalni oporavak.

Jedna od bitnih karakteristika savremenih RTS-a jeste njihova energetska efikasnost koja se u okviru procesa projektovanja RTS-a manifestuje kroz zahteve vezane za smanjenje potrošnje. Kao veoma bitan, aspekt energetske efikasnosti RTS-a proučavan je u disertaciji kroz pregled i klasifikaciju tehnika upravljanja potrošnjom. Među svim tehnikama prednost u disertaciji data je DVS tehnici koja se u zadnje vreme veoma često koristi kod RTS-a kada se želi smanjiti potrošnja procesora a da se pri tome ne naruše zahtevi vezani za ostale performanse sistema. Osnova DVS tehnike je da se ušteda u potrošnji može ostvariti smanjivanjem napona napajanja uporedo sa smanjenjem radne frekvencije procesora. Da je DVS tehnika danas veoma rasprostranjena govori i činjenica da se savremeni mikroprocesori, kao što su *AMD Mobile Athlon*, *Intel XScale*, *Transmeta Crusoe* i mnogi drugi, proizvode sa mogućnošću primene ove tehnike.

Posebna pažnja u disertaciji posvećena je specifičnostima primene DVS tehnika kod RTS-a, pre svega činjenici da se promenom radne frekvencije odnosno napona napajanja procesora menja i vreme potrebno za izvršenja zadatka. Konkretno, kao posledica smanjenja potrošnje procesora povećava se vreme potrebno da procesor izvrši sve zadatke RTS-a. Ovo je moguće pod uslovom da postoji slobodno vreme procesora tj. vremenska redundansa, koje se za to može iskoristiti pri čemu je smanjenje potrošnje moguće samo do mere koja ne dovodi do prekoračenja vremenskih rokova.

Tradicionalno, u naučnoj literature, aspekt energetske efikasnosti i aspekt vezan za tolerisanje grešaka tretiraju se kao nezavisni problemi RTS-a. Naime, aspekt energetske efikasnosti RTS-a uglavnom je dominantan kod projektovanja uređaja široke potrošnje sa baterijskim napajanjem, dok je aspekt vezan za tolerisanje grešaka veoma bitan kod RTS-a sa strogim zahtevima u pogledu bezbednosti. Međutim, trend povećanja gustine pakovanja i smanjenja napona napajanja digitalnih integrisanih kola doveo je do povećanja učestanosti pojave prolaznih otkaza i potrebe objedinjenog sagledavanja ova dva aspekta. U okviru disertacije ova dva aspekta RTS-a razmatrana su zajedno pri čemu je predmet istraživanja bio razvoj i analiza tehnika koje će omogućiti korišćenje slobodnog vremena procesora radi simultane optimizacije RTS-a u pogledu potrošnje energije i sposobnosti tolerisanja grešaka. Glavni cilj disertacije bio je razvoj algoritama za dodelu radnih frekvencija zadacima RTS-a tako da se postigne željeni balans između potrošnje energije i pouzdanosti uz očuvanje vremenskih ograničenja zadataka. Kako slobodno vreme procesora koriste i tehnike energetske efikasnosti i tehnike vezane za tolerisanje grešaka korišćenjem vremenske redundanse neophodno je bilo obraditi pitanje kojoj tehnici dodeliti koliko slobodnog

vremena. Imajući u vidu da se doktorska disertacija bavi rešavanjem, u suštini, kombinatornog problema, čija složenost raste eksponencijalno sa povećanjem broja zadataka u RTS-u i broja raspoloživih frekventnih nivoa, istraživanja su bila usmerena ka iznalaženju vremenski-efikasnih, heurističkih metoda.

Glavni doprinos disertacije je novorazvijeni heuristički FT-DVFS algoritam koji ima za cilj smanjenje potrošnje procesora uz poštovanje svih vremenskih zahteva i zahteva vezanih za prevazilaženje otkaza kod RTS-a. Ono što je novo kod FT-DVFS algoritma, i po dosadašnjim saznanjima prvi put upotrebljeno, jeste integracija u DVS algoritam analize vremena odziva koja se koristi za proveru ispunjenja zahteva vezanih za vremenska ograničenja RTS-a.

Da bi se dokazala efikasnost FT-DVFS algoritma neophodno je bilo razviti simulator, koji je realizovan u programskom jeziku C++. Simulator je omogućio brzu i efikasnu analizu rada RTS-a sa različitih aspekata, kako vremenskog tako i aspekta vezanog za energetsku efikasnost i aspekta vezanog za tolerisanje grešaka. Analizirani su slučajevi RT skupova zadataka kako sintetičkih tako i onih iz "realnog" sveta, pri čemu su korišćeni različiti modeli procesora.

Rezultati simulacija pokazuju da se primenom FT-DVFS algoritma ostvaruje znatna ušteda u potrošnji procesora čak i kada se koriste samo dva frekventna/naponska nivoa kao i da ušteda u potrošnji procesora raste sa porastom broja dostupnih frekventnih/naponskih nivoa. Veći broj dostupnih frekventnih/naponskih nivoa omogućava povećanje uštede u potrošnji od oko 10%. Takođe simulacijama je pokazano da na rezultate FT-DVFS algoritma utiče faktor iskorišćenosti procesora i to tako da ušteda u potrošnji opada sa povećanjem faktora iskorišćenosti. Ušteda u potrošnji ide od čak 80% za vrednost faktora iskorišćenosti 0.1, odnosno do 10% za faktor iskorišćenosti procesora 0.7.

Simulacije vezane za analiza rada RTS-a sa aspekta vezanog za tolerisanje grešaka pokazale su da se primenom FT-DVFS algoritma može ostvariti manja ušteda u potrošnji ako su zahtevi vezani za prevazilaženje otkaza strožiji i obrnuto da što je ušteda u potrošnji procesora veća to je manja mogućnost prevazilaženja otkaza. Ovaj rezultat je očekivan budući da se slobodno vreme procesora, kao konačan resurs, koristi i za tolerisanje grešaka i ta smanjenje potrošnje procesora.

U okviru disertacije predstavljeni su rezultati simulacija koje vrše poređenje heurističkog FT-DVFS algoritma sa optimalnim algoritmom pri čemu su dobijeni vrlo optimistični rezultati. Naime, rezultati simulacija pokazali su da u većini slučajeva FT-DVFS algoritam

generiše skoro optimalne rezultate. Takođe, FT-DVFS algoritam dolazi do rezultata u mnogo kraćem vremenu. Primera radi, za analizu skupa od 10 zadataka, FT-DVFS algoritmu je potrebno manje od 1s, dok je za iste parametre optimalnom algoritmu potrebno nešto više od sat vremena.

Na kraju, treba istaći da ova doktorska disertacija predstavlja jedan od prvih pokušaja integracije analize vremena odziva i DVS tehnika u oblasti projektovanja energetski efikasnih RTS-a koji koriste vremensku redundansu za prevazilaženje otkaza. Rezultat ove integracije je razvijeni FT-DVFS algoritam čija je efikasnost dokazana simulacijama, što otvara mogućnost daljeg istraživanja u cilju usavršavanja predložene metode ili modifikacije algoritma na druge srodne probleme. Jedna od mogućnosti je proširenje FT-DVFS algoritma kako bi se obuhvatili i višeprocesorski *hard* FT RTS-a. U disertaciji je korišćen deterministički model grešaka koga je moguće zameniti probabalističkim modelom i time proširiti FT-DVFS algoritam i za taj slučaj. Navedene teme ostaju za budući rad.

9 Literatura

[Ahm10] A. S. Ahmadian, M. Hosseingholi, A. Ejlali, "A Control-Theoretic Energy Management for Fault-Tolerant Hard Real-Time Systems", *2010 IEEE International Conference on Computer Design*, pp 173-178, 2010.

[Alt12] S. Altmeyer, R. Davis, C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems", *Real-Time Systems*, vol 48, pp 499-526, 2012.

[Arl99] J. Arlat, Y. Crouzet, Y. Deswarte, J. Laprie, D. Powell, P. David, J. Dega, L. Rabéjac, H. Schindler, J. Soucailles, "Fault Tolerant Computing", *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[Aud91] N. C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach", *IFAC/IFIP Workshop, Atlanta, Georgia*, pp 127–132, 15-17 May 1991.

[Aud93] N. C. Audsley, A. Burns, M.F. Richardson, K. Tindell, A.J.Wellings "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling", *Software Engineering Journal*, vol 8, pp 284–292, September 1993.

[Ayd01] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez, "Determining optimal processor speeds for periodical real-time tasks with different power characteristics", *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pp 225–232, 2001.

[Ays07] H. Aysan, R. Dobrin, S. Punnekkat, "FT-Feasibility in Fixed Priority Real-Time Scheduling", *Report – MRTC*, March 2007.

[Bas10] A. Bastoni, B. Brandenburg, J. Anderson, "An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers", In *Proc. of RTSS*, pp 14–24, 2010.

[Bar96] B. W. Johnson, "An Introduction to the Design and Analysis of Fault-tolerant Systems", *Fault-tolerant Computer System Design*, pp 1-87, Prentice-Hall, 1996.

[Bau05] R. Baumann, "Soft errors in advanced computer systems", *IEEE Design and Test of Computers*, vol 22, pp 258–266, 2005.

[Ben03] A. Benso, "A Watchdog Processor to Detect Data and Control Flow Errors", *Proc. 9th IEEE On-Line Testing Symp.*, pp 144 - 148, 2003.

[Bin04] E. Bini, G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems", *IEEE Transactions on Computers*, vol 53, issue 11, pp 1462 – 1473, 2004.

[Bin05] E. Bini, G. C. Buttazzo, "Measuring the Performance of Schedulability Tests Real-Time Systems", vol 30, pp 129–154, 2005.

[Bow93] N. S. Bowen, D. K. Pradhan, "Processor and memory based checkpoint and rollback recovery", *IEEE Computer*, vol 26, pp 22–29, February 1993.

[Bra02a] S. Brankov, M. Jevtić, "Algoritmi planera HRTS-a sa tolerisanjem greške", *Zbornik XLVI konferencije za elektroniku, telekomunikacije, računarstvo, automatiku i nuklearnu tehniku, ETRAN 2002*, Banja Vrućica, pp I.70-I.73, juni 2002.

[Bra02b] S. Brankov, M. Jevtić, "Izbor algoritama vremenskog planiranja izvršavanja zadataka u sistemima za upravljanje i nadzor industrijskih procesa", *Zbornik radova IV simpozijuma industrijske elektronike, INDEL 2002*, Banja Luka, pp. 250-254, novembar 2002.

[Bru04] P. Brucker, "Scheduling Algorithms", *Springer Science & Business Media*, 2004.

[Bur96] A. Burns, R. Davis, S. Punnekkat, "Feasibility Analysis of Fault-Tolerant Real-Time Task Sets", *Proceedings of the EuroMicro Conference on Real-Time Systems*, pp 522–527, 1996.

[Bur97] A. Burns, A. Wellings, "Real-Time Systems and Programming Languages", *Addison Wesley Longman*, 1997.

[Bur00] T. D. Burd, T. A. Pering, A. J. Stratakos, R. W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System", *IEEE J. Solid-State Circuits*, vol 35, pp 1571-1580, 2000.

[Cam92] A. Campbell, P.McDonald, K. Ray, "Single Event Upset Rates in Space", *IEEE Trans. on Nuclear Sci.*, vol 39, pp 1828–1835, December 1992.

[Che89] S. Cheng, J. Stankovic, K. Ramamritham, "Scheduling Algorithms for Hard Real-Time Systems-A Brief Survey", *IEEE Computer Society*, 1989.

[Chr08] A. Christy Persya, T.R. Gopalakrishnan Nair, "Fault Tolerant Real Time Systems", *International Conference on Next Generation Software Application*, pp 177-180, 2008.

[Cms82] X. Castillo, S. R. McConnel, D. P. Siewiorek, "Derivation and Calibration of a Transient Error Reliability Model", *IEEE Transactions on Computers*, vol 31, pp 658–671, 1982.

[Cot02] F. Cottet, J. Delacroix, Z. Mammeri, "Scheduling in Real-Time Systems", *John Wiley & Sons*, 2002.

[Dai08] J. Daintith, T. Wright, "Oxford dictionary of computing", *Oxford University Press*, 2008.

[Dav08] R. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-time Systems", *IEEE Transactions on Computers*, vol 57, issue 9, pp 1261 – 1276, 2008.

[Djo05] G. Lj. Djordjevic, T. R. Stankovic, M. K. Stojcev, "Concurrent error detection in FSMs using transition checking technique", *7th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*, pp 61-64, 2005.

[Djo12a] S. Djosic, M. Jevtic, M. Damnjanovic, "Power consumption analysis of fault tolerant real-time systems", *XLVII International Scientific Conference on Information, Communication and energy Systems and technologies*, vol 1, pp 163- 166, Veliko Trnovo, Bulgaria, June 28 - 30, 2012.

[Djo12b] S. Djosic, M. Jevtic, "Dynamic voltage scaling for real-time systems under fault tolerance constraints", *Proceedings of the 28th International Conference on Microelectronics-MIEL 2012*, pp 375-378, Niš, Serbia, May 13-16, 2012.

[Djo13] S. Djosic, M. Jevtic, "Dynamic Voltage and Frequency Scaling Algorithm for Fault-Tolerant Real-Time Systems", *Microelectronics Reliability, Elsevier Ltd.*, vol 53, number 7, pp 1036-1042, 2013.

[Dob08] R. Dobrin, H. Aysan, S. Punnekkat, "Maximizing the Fault Tolerance Capability of Fixed Priority Schedules", *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[Dub13] E. Dubrova, "Fault-Tolerant Design", *Springer*, 2013.

[Đoš04] S. Đošić, M. Jevtić, "Planiranje zadataka u sistemu za rad u realnom vremenu sa redundansom u vremenu za prevazilaženje otkaza", *Zbornik radova V simpozijuma industrijske elektronike, INDEL 2004*, Banja Luka, pp. 146-149, novembar 2004.

[Đoš05] S. Đošić, "Projektovanje sistema za rad u realnom vremenu sa *on-line* nadzorom procesa i redundansom u vremenu", *Magistarska teza*, Niš, Novembar 2005.

[Đoš07] S. Đošić, M. Jevtić, "Opis planera RTS-a UML-om", *Zbornik radova XIII simpozijuma o računarskim naukama i informacionim tehnologijama YU INFO 2007*, Kopaonik, mart 2007.

[Đoš09] S. Đošić, M. Jevtić, "Analysis of transient fault tolerance in hard real-time systems with time redundancy", *Facta Universitatis, Series: Automatic control and robotics*, vol 8, no 1, pp 149-163, 2009.

[Đoš10a] S. Đošić, M. Jevtić, "Analysis of Real-Time Systems Timing Constrains", *SSSS2010, 3rd Small Systems Simulation Symposium*, pp 56-60, Niš, Serbia, February 12-14, 2010.

[Đoš10b] S. Đošić, M. Jevtić, "Analiza i prikaz vremenskog sleda događaja u RTS-u sa RM algoritmom planiranja", *Zbornik radova 54. konferencije za ETRAN*, EL 4.3-1-4, Donji Milanovac, 07-11 jun 2010.

[Đoš11a] S. Đošić, M. Jevtić, M. Damnjanović, "Analysis of possibilities to overcome the transient faults in real-time systems with time redundancy", *Proceedings of the XLVI International Scientific Conference on Information, Communication and Energy Systems and Technologies*, pp 417- 420, 2011.

[Đoš11b] S. Đošić, M. Jevtić, "Mogućnosti prevazilaženja prolaznih otkaza kod RTS-a sa redundansom u vremenu", *Zbornik radova 55. konferencije za ETRAN*, EL 2.1-1-4, Banja Vrućica, 6-9. juna 2011.

[Đoš12a] S. Đošić, M. Jevtić, "Energy efficiency and fault tolerance analysis of hard real-time systems", *Proceedings of 4rd Small Systems Simulation Symposium 2012*, pp 101-105, Niš, Serbia, February 12-14, 2012.

[Đoš12b] S. Đošić, M. Jevtić, "Povećanje energetske efikasnosti RTS-a sa redundansom u vremenu za prevazilaženje otkaza", *Zbornik radova 56. konferencije za ETRAN*, EL 2.1-1-4, Zlatibor, 11-14. juna 2012.

[Fen08] F. Xia, Y.C. Tian, Y. Sun, J. Dong, "Control-theoretic dynamic voltage scaling for embedded controllers", *IET Computers & Digital Techniques*, vol 2, issue 5, pp 377 – 385, September 2008.

[Fle01] M. Fleischmann, "Longrun power management: Dynamic power management for crusoe processors", *Advanced Micro Devices, Tech. Rep*., 2001.

[FTS] http://es.elfak.ni.ac.rs/mps/materijal/4-FTS.pdf

[Gho95] S. Ghosh, R. Melhem, D. Mossé, "Enhancing real-time schedules to tolerate transient faults", *Proceedings of the IEEE Real-Time Systems Symposium*, pp 120–129, 1995.

[Gom06] M. A. Gomaa, T. N. Vijaykumar, "Opportunistic Transient-Fault Detection", *IEEE Micro*, vol 26, pp 92-99, 2006.

[Gru01] F. Gruian, "Hard Real-Time Scheduling Using Stochastic Data and DVS Processors", *Proceedings of the International Symposium on Low Power Electronics and Design*, pp 46–51, August 2001.

[Gua11] N. Guan, P. Ekberg, M. Stigge, W. Yi, "Resource sharing protocols for real-time task graph systems", *Proc. of the ECRTS*, pps 272 –281, 2011.

[Han03] C.-C. Han, K. G. Shin, J. Wu, "A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults", *IEEE Transactions on Computers*, vol 52, pp 362–372, 2003.

[Ins07] I. Lee, J. Y-T. Leung, S. H. Son, "Handbook of Real-Time and Embedded Systems", *Chapman&Hall*, 2007.

[Int00] "The intel xscale micro architecture", *Intel Corporation, Tech. Rep.*, 2000.

[Iye86] R. K. Iyer, D. J. Rossetti, M. C. Hsueh, "Measurement and Modeling of Computer Reliability as Affected by System Activity", *ACM Trans. on Comp. Syst.*, vol 4, pp 214–237, 1986.

[Izo08] V. Izosimov, P. Pop, P. Eles, Z. Peng, "Scheduling of Fault-Tolerant Embedded Systems with Soft and Hard Timing Constraints", *Design, Automation and Test in Europe, DATE '08*, pp 915 – 920, 10-14 March 2008.

[Jev02] M. Jevtić, M. Cvetković, S. Brankov, "Task Execution in Real-Time Systems for Industrial Control and Monitoring", *Electronics*, Faculty of Electrical Engineering University of Banjaluka, vol 6, no 2, pp 56-61, december 2002.

[Jev04a] M. Jevtić, "*Projektovanje pouzdanih mikroračunarskih sistema*", *Edicija monografije*, Elektronski fakultet, Niš 2004.

[Jev04b] M. Jevtić, V. Zerbe, S. Brankov, "Multilevel Validation of Online Monitor for Hard Real-Time Systems", *Proceedings of the Conference MIEL 2004*, Niš, Serbia and Montenegro, pp 755-758, May 2004.

[Jev08] M. Jevtić, S. Đošić, B. Jovanović, "Programabilni system za energetski efikasno upravljanje sistemom individualnog grejanja u domaćinstvima", *Zbornik radova 52. konferencije za ETRAN*, EL 4.1-1-4, Palić, juni 2008.

[Jev09a] M. Jevtić, B. Jovanović, S. Brankov, M. Cvetković, "Jedna realizacija detektora kvara u upravljačkim sistemima robota" *INFOTEH-JAHORINA*, vol 8, ref E1-7, pp 814-818, March 2009.

[Jev09b] M. Jevtić, M. Damnjanović, "Design for testability of real-time systems for industrial process control", *FACTA UNIVERSITATIS, Series: Automatic Control and Robotics*, vol 8, no 1, pp 45 – 65, 2009.

[Joh89] B. W. Johnson, "*Design and Analysis of Fault-Tolerant Digital Systems*", *Addison-Wesley Publishing Company*, Reading, Massachusetts, 1989.

[Jos96] M. Joseph, P. Pandya, "Finding Response Times in a Real-Time System", *The Computer J. (British Computer Soc.)*, vol 29, no 5, pp 390-395, 1996.

[Kan03] N. Kandasamy, J.P. Hayes, B.T. Murray, "Transparent Recovery from Intermittent Faults in Time-Triggered Distributed Systems", *IEEE Trans. on Computers*, vol 52, pp 113–125, 2003.

[Kan10] N. Kanekawa, E. H. Ibe, T. Suga, Y. Uematsu, "Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances", *Springer*, 2010.

[Kar12] M. Karmani, C. Khedhiri, B. Hamdi, K. L. Man, E. G. Lim, C. U. Lei, "A Concurrent Error Detection Based Fault-Tolerant 32 nm XOR-XNOR Circuit Implementation", *Proceedings of the International Multi*

conference of Engineers and Computer Scientists, vol II, March 14-16, Hong Kong 2012.

[Kor07] I. Koren, C. M. Krishna, "Fault-Tolerant Systems", http://www.amazon.com/Fault-Tolerant-Systems-Israel-Koren/dp/0120885255 *Morgan Kaufmann*, 2007.

[Krs05] M.D. Krstic, M.K. Stojcev, G. Lj. Djordjevic, I.D. Andrejic, "A mid-value select voter, Microelectronics Reliability", vol 45, issues 3–4, pp 733–738, March–April 2005.

[Kwa01] S. W. Kwak, B. I. Choi, B. K. Kim, "An optimal checkpointing strategy for real-time control systems under transient faults", *IEEE Trans. Reliability*, vol 50, pp 293-301, September 2001.

[Lap92] J.C. Laprie, "Dependability: Basic Concepts and Terminology", *Dependable Computing and Fault-Tolerant Systems*, vol 5, Springer-Verlag, 1992.

[Lee00] S. Lee, T. Sakurai, "Run-time Voltage Hopping for Lowpower Real-Time Systems", *Proceedings of the 37th Design Automation Conference*, pp 806–809, June 2000.

[Lim03] M. G. Lima, A. Burns, "An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems", *IEEE Trans. Computers*, vol 52, no 10, pp 1332-1346, October 2003.

[Lin10] F. Lindh, T. Otnes, J. Wennerström, "Scheduling Algorithms for Real-Time Systems", *Mini-conference on Interesting Results in Computer Science and Engineering 2010*, Mälardalen University, Sweden, 2010.

[Loc91] C. D. Locke, D. R. Vogel, T. J. Mesler, "Building a Predictable Avionics Platform in Ada: A Case Study", *Proceedings of the IEEE Real-Time Systems Symposium*, pp 181–189, 1991.

[Mah88] A. Mahmood, E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors - A Survey", *IEEE Trans. on Computers*, vol 37, pp 160-174, 1988.

[Mel04] R. Melhem, D. Mosse, E. Elnozahy, "The interplay of power management and fault recovery in real-time systems", *IEEE Transactions on Computers*, vol 53, pp 217-231, 2004.

[Met00] C. Metra, M. Favalli, B. Ricco, "Signal Coding and CMOS Gates for Combinational Functional Blocks of Very Deep Submicron Self-checking Circuits", *VLSI DESIGN*, vol 11, no 1, pp 23- 34, 2000.

[Mil11] D. Milićev, B. Furlan, "Programiranje u realnom vremenu", *Univezitet u Beogradu –Elektrotehnički fakultet*, Beograd, 2011.

[Mir95] G. Miremadi, J. Torin, "Evaluating Processor-Behaviour and Three Error-Detection Mechanisms Using Physical Fault-Injection", *IEEE Trans. on Reliability*, vol 44, pp 441-454, 1995.

[Mob01] "Mobile AMD Athlon 4 processor model 6 cpga data sheet review", *Advanced Micro Devices, Tech. Rep*. 24319, 2001.

[Mub12] S. Mubeen, J. Maki-Turja, Mikael. Sjodin, "Support for Holistic Response-Time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study", *Proceedings of the 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pp 210-221, 2012.

[Nas07] N. Min-Allah, Y. J. Wang, J. S. Xing, W. Nisar, A. R. Kazmi, "Towards Dynamic Voltage Scaling in Real-Time Systems – A Survey", *IJCSES International Journal of Computer Sciences and Engineering System*s, vol 1, no 2, April 2007.

[Nis97] N. Nissanke, "Realtime Systems", *Prentice Hall*, 1997.

[Nol09] T. Nolte, I. Shin, M. Behnam, M. Sjodin, "A Synchronization Protocol for Temporal Isolation of Software Components in Vehicular Systems", *IEEE Transactions on Industrial Informatics*, vol 5, pp 375 –387, 2009.

[OSO] http://www.google.rs/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=r-ja&ved=0CDwQFjAD&url=http%3A%2F%2Fpredmet.singidunum.ac.rs%2Fpluginfile.php%2F3949%2Fmod_folder%2Fcontent%2F1%2FII_Odrzavanje_Struktura_Otkazi_PP.ppt%3Fforcedownload%3D1&ei=iyTqUrWYIoXGswaC24CABg&usg=AFQjCNEOKwrtavNiSe2zOvn_YfYuhfLgzg&bvm=bv.60444564,d.Yms

[PES] http://leda.elfak.ni.ac.rs/education/PES/predavanja/2011/02%20%28PES%2-9%20Projektovanje%20elektronskih%20sistema.pdf

[Pet05] P. Peti, R. Obermaisser, H. Kopetz, "Out-of-Norm Assertions", *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp*., pp 209-223, 2005.

[Pin08] P. Zhu, F. Yang, G. T., W. Luo, "Fault-Tolerant Scheduling for Periodic Tasks based on DVFS", *The 9th International Conference for Young Computer Scientists 2008*, pp 2186 – 2191, November 2008.

[Pop07] P. Pop, K. H. Poulsen, V. Lzosimov, P. Eles, "Scheduling and Voltage Scaling for Energy/Reliability Trade-offs in Fault-Tolerant Time-Triggered Embedded Systems", *IEEE/ACM International Conference on Hardware/Software Co-design and System Synthesis*, pp 233-238, 2007.

[Pra96] D. K. Pradhan, "Fault-Tolerant Computer System Design", *Prentice-Hall*, 1996.

[Pun97] S. Punnekkat, A. Burns, "Analysis of Checkpointing for Schedulability of Real-Time Systems", *Proc. Fourth Intl. Workshop on Real-Time Computing Systems Applications*, pp 198-205, 1997.

[Qad03] A. Qadi, S. Goddard, "A dynamic voltage scaling algorithm for sporadic tasks", *Proceedings of the 24th International Real-Time Systems Symposium*, pp 52–62, 2003.

[Riu07] X. Ruibin, D. Mosse, R. Melhem, "Minimizing Expected Energy Consumption in Real-Time Systems through Dynamic Voltage Scaling", *ACM Transactions on Computer Systems*, vol 25, no 4, article 9, December 2007.

[Riz11a] N. B. Rizvandi, "Multiple Frequency Selection in DVFS-Enabled Processors to Minimize Energy Consumption", Chapter 17: Energy Efficient Distributed Computing, *John Wiley Publisher* (In press), 2011.

[Riz11b] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, "Some Observations on Optimal Frequency Selection in DVFS–based Energy Consumption Minimization", *Journal of Parallel and Distributed Computing*, vol 71, issue 8, pp 1154-1164, August, 2011.

[San09] R. M. Santos, J. Santos, J. D. Orozco, "Power saving and fault-tolerance in real-time critical embedded system", *Journal of system Architecture*, vol 55, pp 90-101, 2009.

[Ses07a] R. Seshasayanan, S.K. Srivatsa, "Hardware Implementation of Hybrid Dynamic Voltage Scaling", *Information Technology Journal*, vol 6, pp 338-344, 2007.

[Ses07b] R. Seshasayanan, S. K. Srivatsa, "A novel architecture for VLIW processor", *Academic Open Internet Journal*, vol 21, 2007.

[Shi00] P. P. Shirvani, N. R. Saxena, E. J. McCluskey, "Software-Implemented EDAC Protection against SEUs", *IEEE Trans. on Reliability*, vol 49, pp 273-284, 2000.

[Shin00] Y. Shin, K. Choi, T. Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors", *Proceedings of the International Conference on Computer-Aided Design*, pp 365–368, November 2000.

[Shi87] K. G. Shin, T. Lin, Y. Lee, "Optimal checkpointing of real-time tasks", *IEEE Transactions on Computers*, vol 36, pp 1328–1341, November 1987.

[Shi01] D. Shin, J. Kim, S. Lee "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications", *IEEE Design and Test of Computers*, vol 18, pp 20–30, March 2001.

[Sie78] D.P. Siewiorek, V. Kini, H. Mashburn, S. McConnel, M. Tsao, "A case study of C.mmp, Cm*, and C.vmp: Part I–Experiences with fault tolerance in multiprocessor systems", *Proceedings of the IEEE*, vol 66, pp 1 –1199, 1978.

[Sri04] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers, "The Impact of Technology Scaling on Lifetime Reliability", *Proceedings of the International Conference on Dependable Systems and Networks*, pp 177–186, 2004.

[SRV] http://es.elfak.ni.ac.rs/rts/Materijal/NASTAVA_2012/PDF_PPT/4visoka%20-pouzdanost.pdf

[Sun96] S. Ghosh, "Guaranteeing Fault Tolerance Through Scheduling In Real-Time Systems", 1996.

[TES] http://leda.elfak.ni.ac.rs/education/PES/predavanja/2011/10%20%28PES%2-9%20Testiranje%20elektronskih%20sistema%20-%20deo%202.pdf

[Uns03] O.S. Unsal, I. Koren, "System-level power-aware design techniques in real-time systems", *Proceedings of the IEEE*, vol 91, issue 7, pp 1055 – 1069, July 2003.

[Wis02] W. Chedid, C. Yu, "Survey on Power Management Techniques for Energy Efficient Computer Systems", *Mobile Computing Research Lab., Cleveland State University, Cleveland*, OH, Laboratory Rep., 2002.

[Woo02] K. Woonseok, S. Dongkun, Y. Han-Saem, K. Jihong, M. Sang Lyul, "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems", *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pp 219 – 228, 2002.

[You82] S.J. Young, “Real time languages: Design and development”, *Chichester: Ellis Horwood*, 1982.

[Yun 12] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, L. Sha, “Memory Access Control in Multiprocessor for Real-time Systems with Mixed Criticality”, *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pp 299-308, 2012.

[Zha04a] Y. Zhang, K. Chakrabarty, “Task Feasibility Analysis and Dynamic Voltage Scaling in Fault-Tolerant Real-Time Embedded Systems”, *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp 1170 – 1175, 2004.

[Zha04b] Y. Zhang, K. Chakrabarty, “Dynamic Adaptation for Fault Tolerance and Power Management in Embedded Real-Time Systems”, *Trans. on Embedded Computing Sys*, vol 3, 2004.

[Zhs03] Y. Zhang, K. Chakrabarty, “Energy-aware adaptive checkpointing in embedded real-time systems”, *Proc. of the conference on Design,Automation and Test in Europe (DATE)*, pp 918-923, 2003.

[Zhu04a] D. Zhu, R. Melhem, D. Mosse, “The Effects of Energy Management on Reliability in Real-Time Embedded Systems”, *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pp 35-40, 2004.

[Zhu04b] D. Zhu, R. Melhem, D. Mosse, E. Elnozahy, “Analysis of an Energy Efficient Optimistic TMR Scheme”, *Proc. of the* 10*th International Conference on Parallel and Distributed Systems, Jul.* 2004, pp 559-568, 2004.

[Zhu08] P. Zhu, F. Yang, G. Tu, W. Luo, “Fault-Tolerant Scheduling for Periodic Tasks based on DVFS”, *Proceedings of the 9th International Conference for Young Computer Scientists*, pp 2186 – 2191, 2008.

[Ziv97] A. Ziv, J. Bruck, “An on-line algorithm for checkpoint placement”, *IEEE Transaction on Computer*, vol 46, pp 976-985, 1997.

[Zom12] Y. A. Zomaya, Y. C. Lee, “Energy Efficient Distributed Computing Systems”, *John Wiley & Sons*, 2012.

Kratka biografija kandidata

Kandidat mr Sandra (Brankov) Đošić je rođena 13. maja 1974. godine u Beogradu. Stalno je nastanjena u Nišu. Osnovnu školu „Učitelj Tasa“ i gimnaziju „Bora Stanković“ je završila u Nišu, sa odličnim uspehom. Nosilac je diplome „Vuk Karadžić” kao i većeg broja diploma sa učešća na brojnim takmičenjima u znanju.

Diplomirala je na Elektronskom fakultetu u Nišu na smeru Elektronika i telekomunikacije 2000. godine i stekla stručno zvanje diplomirani inženjer elektrotehnike. Magistrirala je 2006. godine na Elektronskom fakultetu u Nišu, smer Elektronika, odbranivši magistarsku tezu pod naslovom “Projektovanje sistema za rad u realnom vremenu sa *on-line* nadzorom procesa i redundansom u vremenu” i time ispunila uslov za sticanje akademskog naziva magistra tehničkih nauka.

Od 2000. godine uključena je u naučno-istraživački rad na Elektronskom fakultetu, na katedri za Elektroniku. 2001. godine radila je kao istraživač-pripravnik, u periodu od 2002. do 2010. godine kao asistent-pripravnik, a od marta 2010. kao asistent. U okviru nastavnog rada angažovana na izvođenju računskih i laboratorijskih vežbi iz predmeta: Digitalna elektronika, Objektno orijentisane tehnike projektovanja sistema, Sistemi za upravljanje i nadzor u industriji, Projektovanje elektronskih sistema, Digitalni procesni sistemi,

Elektronika 2, Digitalna integrisana kola, Procesni mikroračunarski sistemi, Digitalna elektronska kola.

U svom dosadašnjem radu aktivno je učestvovala u realizaciji osam projekata Ministarstva za nauku i tehnološki razvoj Republike Srbije. Autor je 2 tehnička rešenja. Do sada je publikovala 28 naučnih radova, od čega 1 rad u međunarodnom časopisu, 3 rada u domaćim časopisima, 10 u zbornicima sa međunarodnih konferencija i 14 radova u zbornicima sa domaćih konferencija.

Naučni radovi kandidata

Odbranjen magistarski rad

a.1. Sandra Đošić, "Projektovanje sistema za rad u realnom vremenu sa on-line nadzorom procesa i redundansom u vremenu", magistarska teza, Elektronski fakultet, Univerzitet u Nišu, September 2006.

Rad u časopisu međunarodnog značaja

b.1. Sandra Djosic, Milun Jevtic, "Dynamic Voltage and Frequency Scaling Algorithm for Fault-Tolerant Real-Time Systems", *Microelectronics Reliability, Elsevier Ltd.*, vol 53, no 7, pp 1036-1042, 2013.

Rad u časopisu nacionalnog značaja

c.1. Sandra Đošić, Milun Jevtić, "One realization of switched ethernet fault tolerant communication", *Facta Universitatis, Series: Automatic control and robotics*, vol 10, no 1, pp 115-123, 2011.

c.2. Sandra Đošić, Milun Jevtić, "Analysis of transient fault tolerance in hard real-time systems with time redundancy", *Facta Universitatis, Series: Automatic control and robotics*, vol 8, no 1, pp 149-163, 2009.

c.3. Milun Jevtić, Marko Cvetković, **Sandra Brankov**, "Task Execution in Real-Time

Systems for Industrial Control and Monitoring", *Electronics*, Faculty of Electrical Engineering University of Banjaluka, vol 6, n. 2, p. 56-61, december 2002.

Rad saopšten na skupu međunarodnog značaja štampan u celini

d.1. Igor Stojanovic, Milica Jovanovic, **Sandra Djosic**, Goran Djordjevic, "Improved deflection routing method for bufferless networks-on-chip", *Proceedings of the XLIX International Scientific Conference on Information, Communication and Energy Systems and Technologies ICEST 2014*, vol 1, pp 91-94, Serbia, Niš, June 25 - 27, 2014.

d.2. Sandra Djosic, Milun Jevtic, Milunka Damnjanovic, "Power consumption analysis of fault tolerant real-time systems", *Proceedings of the XLVII International Scientific Conference on Information, Communication and Energy Systems and technologies ICEST 2012*, vol 1, pp 163- 166, Veliko Trnovo, Bulgaria, June 28 - 30, 2012.

d.3. Sandra Djosic, Milun Jevtic, "Dynamic voltage scaling for real-time systems under fault tolerance constraints", *Proceedings of the 28th International Conference on Microelectronics-MIEL 2012*, pp 375-378, Niš, Serbia, May 13-16, 2012.

d.4. Sandra Đošić, Milun Jevtić, "Energy efficiency and fault tolerance analysis of hard real-time systems", Proceedings of 4rd Small Systems Simulation Symposium 2012, pp 101-105, Niš, Serbia, February 12-14, 2012.

d.5. Bojan Jovanović, Milun Jevtić, **Sandra Đošić**, "Using altera DE1 development board for educational purposes", *UNITECH '11 Gabrovo*, Proceedings vol I, pp I-258 – I-261, Bulgaria, 18-19 November 2011.

d.6. Sandra Đošić, Milun Jevtić, Milunka Damnjanović, "Analysis of possibilities to overcome the transient faults in real-time systems with time redundancy", *Proceedings of the XLVI International Scientific Conference on Information, Communication and Energy Systems and technologies ICEST 2011*, vol 2, pp 417- 420, Serbia, Niš, June 29 - July 1, 2011.

d.7. Sandra Đošić, Milun Jevtić, Milunka Damnjanović, "Real-time ethernet communication with transient faults tolerance", *X Triennial International SAUM Conference on Systems, Automatic Control and Measurements*, Niš, Serbia, pp 304-306, November 2010.

d.8. Sandra Đošić, Milun Jevtić, "Analysis of Real-Time Systems Timing Constrains", *SSSS2010, 3rd Small Systems Simulation Symposium*, pp 56-60, Niš, Serbia, February 12-14, 2010.

d.9. Sandra Đošić, Milun Jevtić, „Switched Ethernet fault tolerant communication", *UNITECH '09*, Gabrovo, Proceedings vol I, pp I-322 – I-325, Bulgaria, 20-21 November 2009.

d.10. Milun Jevtić, Volker Zerbe, **Sandra Brankov**, "Multilevel Validation of Online Monitor for Hard Real-Time Systems", *Proceedings of the Conference MIEL 2*004, pp 755-758, Niš, Serbia and Montenegro, May 2004.

Rad saopšten na skupu nacionalnog značaja štampan u celini

e.1. Sandra Đošić, Milun Jevtić, "Povećanje energetske efikasnosti RTS-a sa redundansom u vremenu za prevazilaženje otkaza", *ETRAN 2012, Zbornik radova 56. konferencije za ETRAN*, EL 2.1-1-4, Zlatibor, 11-14. juna 2012.

e.2. Sandra Đošić, Milun Jevtić, "Mogućnosti prevazilaženja prolaznih otkaza kod RTS-a sa redundansom u vremenu", *ETRAN 2011, Zbornik radova 55. konferencije za ETRAN*, EL 2.1-1-4, Banja Vrućica, 6-9. juna 2011.

e.3. Sandra Đošić, Milun Jevtić, "Analiza i prikaz vremenskog sleda događaja u RTS-u sa RM algoritmom planiranja", *ETRAN 2010, Zbornik radova 54. konferencije za ETRAN*, EL 4.3-1-4, Donji Milanovac, 07-11 jun 2010.

e.4. Sandra Đošić, Milun Jevtić, Milunka Damnjanović, "Jedna realizacija industrijske ethernet komunikacije sa tolerancijom prolaznih otkaza", *INFOTEH-JAHORINA*, vol 9, ref B-II-3, pp 185-188, Mart 2010.

e.5. Milun Jevtić, Bojan Jovanović, **Sandra Brankov**, Marko Cvetković, "Jedna realizacija detektora kvara u upravljačkim sistemima robota", *INFOTEH-JAHORINA*, vol 8, ref E1-7, pp 814-818, March 2009.

e.6. Milun Jevtić, **Sandra Đošić**, Bojan Jovanović, "Programabilni system za energetski efikasno upravljanje sistemom individualnog grejanja u domaćinstvima", *Zbornik radova 52. konferencije za ETRA*N, EL 4.1-1-4, Palić, juni 2008.

e.7. Sandra Đošić, Milun Jevtić, "Opis planera RTS-a UML-om", *Zbornik radova XIII simpozijuma o računarskim naukama i informacionim tehnologijama YU INFO 2007*, Kopaonik, mart 2007.

e.8. Borisav Jovanović, Milun Jevtić, **Sandra Đošić**, Miljana Sokolović, "Projektovanje BIST logike u DSP bloku integrisanog merača potrošnje električne energije", *Zbornik radova V simpozijuma industrijske elektronike INDEL*, pp 120-125, Banja Luka, novembar 2004.

e.9. Sandra Đošić, Milun Jevtić, "Planiranje zadataka u sistemu za rad u realnom vremenu sa redundansom u vremenu za prevazilaženje otkaza", *Zbornik radova V simpozijuma industrijske elektronike INDEL*, pp 146-149, Banja Luka, novembar 2004.

e.10. Milun Jevtić, Borisav Jovanović, **Sandra Brankov**, "Upravljačka jedinica sistema na čipu za re*gistrovanje potrošnje električne energije",* Zbornik radova 48. konferencije za ETRAN, sveka I, pp 75-78, Čačak, juni 2004.

e.11. Sandra Brankov, Milun Jevtić, "Realizacija modula za nadzor real-time procesa u VHDL-u", *Zbornik radova 47. konferencije za ETRAN*, sveka I, pp 80-83, Herceg Novi, juni 2003.

e.12. Sandra Brankov, Dragiša Krstić, "Podešavanje frekvencije ring oscilatora pomoću transkonduktanse i dodatne kapacitivnosti", *Zbornik radova X telekomunikacionog foruma TELFOR*, pp 617-619, Beograd, novembar 2002.

e.13. Sandra Brankov, Milun Jevtić, "Izbor algoritama vremenskog planiranja izvršavanja zadataka u sistemima za upravljanje i nadzor industrijskih procesa", *Zbornik radova IV simpozijuma industrijske elektronike INDEL*, pp 250-254, Banja Luka, novembar 2002.

e.14. Sandra Brankov, Milun Jevtić, "Algoritmi planera HRTS-a sa tolerisanjem greške", *Zbornik radova 46. konferencije ETRAN*, sveka I, pp I.70-I.73, Banja Vrućica, juni 2002.

Izjave autora

Prilog 1.

IZJAVA O AUTORSTVU

Izjavljujem da je doktorska disertacija, pod naslovom

Tolerisanje grešaka i energetska efikasnost kod sistema za rad u realnom vremenu sa vremenskom redundansom

- rezultat sopstvenog istraživačkog rada,
- da predložena disertacija, ni u celini, ni u delovima, nije bila predložena za dobijanje bilo koje diplome, prema studijskim programima drugih visokoškolskih ustanova,
- da su rezultati korektno navedeni i
- da nisam kršila autorska prava, niti zloupotrebila intelektualnu svojinu drugih lica.

U Nišu, 09.02.2015.

Autor disertacije:
mr Sandra Đošić, dipl. inž.

Potpis doktoranda

Prilog 2.

IZJAVA O ISTOVETNOSTI ŠTAMPANE I ELEKTRONSKE VERZIJE DOKTORSKE DISERTACIJE

Ime i prezime autora: **Sandra Đošić**

Studijski program: **Elektronika**

Naslov rada: **Tolerisanje grešaka i energetska efikasnost kod sistema za rad u realnom vremenu sa vremenskom redundansom**

Mentor: **prof. dr Goran Lj. Đorđević**

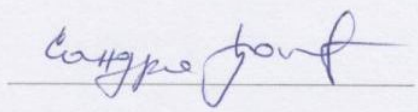
Izjavljujem da je štampana verzija moje doktorske disertacije istovetna elektronskoj verziji, koju sam predala za unošenje u **Digitalni repozitorijum Univerziteta u Nišu**.

Dozvoljavam da se objave moji lični podaci, koji su u vezi sa dobijanjem akademskog zvanja doktora nauka, kao što su ime i prezime, godina i mesto rođenja i datum odbrane rada, i to u katalogu Biblioteke, Digitalnom repozitorijumu Univerziteta u Nišu, kao i u publikacijama Univerziteta u Nišu.

U Nišu, 09.02.2015.

Autor disertacije:
mr Sandra Đošić, dipl. inž.

Potpis doktoranda



Prilog 3.

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku „Nikola Tesla“ da, u Digitalni repozitorijum Univerziteta u Nišu, unese moju doktorsku disertaciju, pod naslovom:

Tolerisanje grešaka i energetska efikasnost kod sistema za rad u realnom vremenu sa vremenskom redundansom

koja je moje autorsko delo.

Disertaciju sa svim prilozima predala sam u elektronskom formatu, pogodnom za trajno arhiviranje.

Moju doktorsku disertaciju, unetu u Digitalni repozitorijum Univerziteta u Nišu, mogu koristiti svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (Creative Commons), za koju sam se odlučila.

1. Autorstvo
2. Autorstvo – nekomercijalno
3. Autorstvo – nekomercijalno – bez prerade
4. Autorstvo – nekomercijalno – deliti pod istim uslovima
5. Autorstvo – bez prerade
6. Autorstvo – deliti pod istim uslovima

U Nišu, 09.02.2015.

Autor disertacije:

mr Sandra Đošić, dipl. inž.

Potpis doktoranda