UNIVERZITET U ZENICI

Politehnički fakultet

Softversko inženjerstvo

Operativni sistemi

Delaida Muminović

ANALIZA PERFORMANSI I VERIFIKACIJA REAL-TIME OPERATIVNOG SISTEMA NA RISC-V ARHITEKTURI

Diplomski rad - radna verzija 1

Mentor: Van. Prof. dr. Denis Čeke

Zenica, 2025.

Sadržaj

1. Sažetak rada

Ovaj diplomski rad bavi se sveobuhvatnom analizom performansi i verifikacijom funkcionalnosti operativnog sistema za rad u realnom vremenu – FreeRTOS, na otvorenoj instrukcijskoj arhitekturi RISC-V. S obzirom na rastući značaj determinizma i predvidljivosti u ugradbenim sistemima, cilj je bio kvantitativno karakterisati ključne *real-time* primitive FreeRTOS-a i potvrditi njegovo ispravno funkcionisanje na RISC-V platformi.

Metodologija istraživanja obuhvatila je uspostavljanje kontrolisanog simulacionog okruženja baziranog na QEMU emulatoru, dizajn i implementaciju specifičnih testova performansi za mjerenje latencije kontekstnog prebacivanja, efikasnosti upravljanja memorijom i performansi komunikacije između taskova. Korišten je automatizovani proces prikupljanja i obrade podataka putem Python skripti, što je osiguralo pouzdanost i reproduktivnost rezultata.

Analiza dobijenih rezultata pokazala je da FreeRTOS na RISC-V arhitekturi ostvaruje konkurentne performanse za testirane primitive. Predstavljene su detaljne statističke metrike, uključujući minimalne, maksimalne i prosječne latencije, kao i empirijske vrijednosti najgoreg vremena izvršavanja (Worst-Case Execution Time, WCET) na 95. i 99. percentilu. Vizuelna analiza *jittera* putem *box plotova* pružila je uvid u varijabilnost performansi. Analiza schedulabilnosti, primjenom Liu & Layland modela, demonstrirala je sposobnost sistema da ispuni rokove zadataka u hipotetičkim scenarijima. Funkcionalnost FreeRTOS-a je dodatno verificirana kroz tri demonstracione aplikacije koje su potvrdile ispravno raspoređivanje zadataka, međutaskovnu komunikaciju i deterministički odziv na prekide.

Rad je također pružio vrijedne inženjerske uvide iz pokušaja hardverske implementacije na Tang Nano 9K FPGA ploči, naglašavajući kompleksnost hardversko-softverske ko-integracije i izazove u razvoju alata. Ovaj diplomski rad doprinosi sistematičnoj kvantitativnoj analizi FreeRTOS-a na RISC-V arhitekturi, postavljajući temelj za budući razvoj pouzdanih *real-time* aplikacija na otvorenim hardverskim platformama.

Ključne riječi: FreeRTOS, RISC-V, real-time operativni sistemi, analiza performansi, QEMU emulacija, WCET analiza

2. Postavka problema

2.1. Kontekst i motivacija

Savremeni ugradbeni sistemi u kritičnim domenama poput industrijske automatizacije, medicinskih uređaja i automobilske industrije zahtijevaju garantovano vremenski predvidivo ponašanje. Sistemi za rad u realnom vremenu moraju izvršavati operacije unutar striktno definisanih vremenskih rokova, gdje propuštanje može dovesti do ozbiljnih posljedica ili potpunog kvara sistema.

RISC-V instrukcijska arhitektura dobija rastući značaj kao otvorena alternativa vlasničkim arhitekturama zbog svoje modularnosti i fleksibilnosti. Istovremeno, FreeRTOS predstavlja jedan od najpopularnijih *open-source* operativnih sistema za rad u realnom vremenu u ugradbenim aplikacijama zbog male memorijske zauzetosti i široke podrške za različite arhitekture. Uprkos rastućoj popularnosti ove kombinacije tehnologija, postoji značajan nedostatak sveobuhvatnih kvantitativnih analiza performansi FreeRTOS-a na RISC-V arhitekturi.

2.2. Definicija problema

Glavni problem predstavlja nedostatak sistematične kvantitativne karakterizacije performansi FreeRTOS-a na RISC-V arhitekturi koja bi omogućila inženjerima donošenje informisanih odluka pri dizajnu *real-time* sistema. Postojeće studije često se fokusiraju na ograničen skup metrika, ne pružaju empirijske WCET (*Worst-Case Execution Time*) analize na visokim percentilima, ili ne omogućavaju reprodukciju rezultata zbog nedostatka detaljne metodologije. Bez preciznih podataka o latencijama, najgorim vremenima izvršavanja i *jitteru* za osnovne operacije kernela, razvoj pouzdanih *hard real-time* aplikacija na ovoj platformi je znatno otežan.

2.3. Hipoteza istraživanja

Hipoteza ovog istraživanja je da FreeRTOS na RISC-V arhitekturi pruža performanse konkurentne sa etabliranim arhitekturama u *real-time* aplikacijama, sa predvidljivim i determinističkim ponašanjem pogodnim za *hard real-time* sisteme, uzimajući u obzir emulacioni *overhead* simulacionog okruženja.

2.4. Ciljevi istraživanja

Istraživanje ima sljedeće specifične ciljeve:

1. Kvantifikovanje latencija ključnih FreeRTOS primitiva na RISC-V arhitekturi kroz sistematsko mjerenje kontekstnog prebacivanja, upravljanja memorijom i komunikacije između zadataka.
2. Određivanje empirijskih WCET vrijednosti na visokim procentima(95%, 99%) za kritične sistemske operacije koje omogućavaju analizu *schedulabilnosti*.
3. Verifikacija funkcionalnosti FreeRTOS-a kroz demonstracione aplikacije koje testiraju raspoređivanje zadataka, međutaskovnu komunikaciju i rukovanje prekidima.
4. Identifikacija praktičnih izazova implementacije kroz pokušaj hardverske realizacije na FPGA platformi.

2.5. Ograničenja istraživanja

Istraživanje ima sljedeća ograničenja koja utiču na generalizaciju rezultata:

Analiza je ograničena na QEMU emulaciju, što može maskirati određene hardverske karakteristike i uvesti emulacioni *overhead* koji utiče na apsolutne vrijednosti mjerenja. Testiranje je sprovedeno na specifičnoj RISC-V konfiguraciji (RV32IM) bez pokrivanja svih mogućih varijanti arhitekture. Fokus je stavljen na temeljne FreeRTOS funkcionalnosti bez detaljne analize naprednih servisa poput *software timer*-a ili naprednih algoritama planiranja. Hardverska verifikacija je ograničena na pokušaj implementacije na jednoj FPGA platformi (Tang Nano 9K), što ograničava uvide o prenosivosti rezultata.

2.6. Metodologija rješavanja

Rješavanje problema realizuje se kroz nekoliko povezanih koraka. Uspostavljanje kontrolisanog okruženja putem QEMU emulatora sa RISC-V virt mašinom (generička virtuelna platforma - virt) omogućava reproduktivne testove sa eliminacijom varijabilnosti hardverskih faktora. Dizajn specifičnih testova performansi fokusira se na mjerenje latencije kontekstnog prebacivanja, efikasnosti *heap* algoritma i performansi komunikacionih primitiva kroz automatizovane *benchmark* aplikacije. Automatizovani proces prikupljanja podataka kroz UART komunikaciju i Python skripte osigurava pouzdanost i reproduktivnost rezultata. Statistička analiza rezultata obuhvata izračunavanje WCET vrijednosti na visokim

percentilima, analizu varijabilnosti i komparaciju sa postojećim istraživanjima. Verifikacija funkcionalnosti kroz demonstracione aplikacije testira ključne aspekte RTOS ponašanja u praktičnim scenarijima.

3. Uvod i struktura rada

Ovaj diplomski rad sistematično istražuje performanse i funkcionalnost operativnih sistema za rad u realnom vremenu (RTOS) na otvorenoj RISC-V instrukcijskoj arhitekturi. Kroz sveobuhvatnu analizu FreeRTOS-a u kontrolisanom simulacionom okruženju, rad nastoji da pruži kvantitativne podatke i verifikuje ključne aspekte ponašanja sistema. U cilju postizanja navedenih ciljeva, rad je strukturiran u devet poglavlja, od kojih svako doprinosi razjašnjenju postavljenog problema, metodologiji rješavanja i prezentaciji postignutih rezultata.

3.1. Važnost istraživanja

Kombinacija FreeRTOS-a i RISC-V arhitekture predstavlja atraktivnu opciju za razvoj ugradbenih sistema zbog *open-source* prirode obje tehnologije. Međutim, za uspješnu primjenu u kritičnim aplikacijama potrebni su precizni podaci o performansama i karakteristikama ponašanja sistema. Ovaj rad doprinosi popunjavanju te praznine u znanju pružajući detaljnu kvantitativnu analizu i verifikaciju funkcionalnosti RTOS-a na RISC-V platformi.

3.2. Struktura rada

Rad je organizovan u devet poglavlja. Poglavlje 1 predstavlja sažetak rada sa ključnim rezultatima i doprinosima istraživanja. Poglavlje 2 definiše problem istraživanja, postavljene hipoteze, ciljeve i ograničenja, te opisuje metodologiju rješavanja. Poglavlje 3, odnosno ovo poglavlje, pruža uvod u problematiku i detaljnu strukturu rada. Poglavlje 4 pokriva teorijske osnove operativnih sistema za rad u realnom vremenu, arhitekturu FreeRTOS-a, specifikacije RISC-V arhitekture i pregled postojećih istraživanja. Srž praktičnog rada predstavljena je u Poglavlju 5, gdje se opisuje testno okruženje, metodologija mjerenja, rezultati performansi i verifikacija funkcionalnosti. Poglavlje 6 sumira glavne rezultate i potvrđuje ispunjenje postavljenih ciljeva. Poglavlje 7 predlaže smjernice za buduća istraživanja i moguća proširenja. Poglavlje 8 sadrži kompletnu bibliografiju citiranih referenci, dok Poglavlje 9 obuhvata priloge sa izvornim kodom i dodatnim materijalima za reproduktivnost rada.

4. Teorijske osnove sa osvrtom na postojeća rješenja

4.1. Operativni sistemi za rad u realnom vremenu (RTOS)

4.1.1. Definicija i osnovne karakteristike

Operativni sistem za rad u realnom vremenu (*Real-Time Operating System*, RTOS) predstavlja specijalizovanu klasu operativnih sistema dizajniranih za izvršavanje aplikacija u tačno definisanim vremenskim okvirima[1]. Za razliku od konvencionalnih operativnih sistema opšte namjene, poput Windows-a ili standardnih Linux distribucija, čiji je glavni cilj optimizacija propusnosti ili korisničkog iskustva, RTOS sistemi se primarno fokusiraju na predvidljivost i determinizam u izvršavanju zadataka. Fundamentalna razlika između RTOS-a i konvencionalnih OS-a leži u pristupu upravljanju resursima: konvencionalni sistemi optimizuju ukupnu propusnost i pokušavaju pružiti "fer" raspodjelu resursa među zadacima, dok RTOS garantuje da će kritični zadaci biti izvršeni u definisanim vremenskim okvirima, čak i na račun ukupne ukupne efikasnosti sistema.

4.1.2. Klasifikacija sistema realnog vremena

Prema vremenskim ograničenjima i posljedicama propuštanja rokova za izvršavanje, sistemi za rad u realnom vremenu klasifikuju se u tri osnovne kategorije. **Hard Real-Time Sistemi** predstavljaju najstriktniju kategoriju gdje propuštanje bilo kojeg roka (*deadline*) rezultuje potpunim otkazom sistema[2]. U ovim sistemima, tačnost nije samo poželjna već je obavezna karakteristika. Sistemi za kontrolu vazdušnih jastuka u automobilima predstavljaju tipičan primjer, gdje kašnjenje od nekoliko milisekundi može biti fatalno. Slično tome, avionski kontrolni sistemi, medicinski uređaji za održavanje života poput pejsmejkera, te kontrolni sistemi nuklearnih elektrana spadaju u ovu kategoriju. **Firm Real-Time Sistemi** tolerišu povremene propuste rokova, ali česta kašnjenja degradiraju performanse sistema do neprihvatljivog nivoa[3]. Ova kategorija predstavlja kompromis između strogosti *hard real-time* sistema i fleksibilnosti *soft real-time* pristupa. Karakteristike *firm real-time* sistema uključuju situacije gdje povremeni propust roka nije katastrofalan ali snižava kvalitet, dok sistematsko kašnjenje čini sistem neupotrebljivim. Često se koriste u multimedijskim aplikacijama i digitalnoj obradi signala, kao što su video konferencijski sistemi gdje povremeni ispušteni okvir nije kritičan. **Soft Real-Time Sistemi** mogu funkcionisati ispravno i kada se rokovi propuste, mada se kvalitet usluge postupno degradira[4]. Ovi sistemi pružaju

"najbolji mogući napor" za ispunjavanje rokova, gdje su kašnjenja nepoželjna ali ne čine sistem neoperativnim. Prioritet je na održavanju osnovne funkcionalnosti, a tipični primjeri uključuju video *streaming* servise, interaktivne aplikacije i web servere.

4.1.3. Ključni koncepti u sistemima realnog vremena

Za dublje razumijevanje RTOS-a, neophodno je definisati nekoliko fundamentalnih pojmova. **Determinizam** predstavlja najvažnije svojstvo RTOS-a koje obezbjeđuje da se vrijeme izvršavanja sistemskih operacija može precizno predvidjeti[5]. Ova karakteristika omogućava precizno planiranje raspoređivanja zadataka, analizu *schedulabilnosti* (odnosno provjeru da li sistem može ispuniti sve rokove), te predvidljivost ponašanja sistema u različitim scenarijima opterećenja.

Latencija se definiše kao vrijeme koje protekne od nastanka spoljašnjeg događaja do početka izvršavanja odgovarajuće rutine za rukovanje tim događajem[6]. U kontekstu RTOS-a razlikuju se latencija prekida, koja predstavlja vrijeme od nastanka prekida do ulaska u rutinu za obradu prekida (*Interrupt Service Routine*, ISR); latencija planera, kao vrijeme potrebno za prebacivanje na zadatak sa najvišim prioritetom; te end-to-end latencija, koja označava ukupno vrijeme koje protekne od momenta nastanka određenog događaja do završetka kompletnog odgovora sistema na taj događaj.

Jitter kvantifikuje varijabilnost u vremenu izvršavanja operacija kao razliku između najgoreg i najboljeg slučaja[7]. Minimizacija *jittera* je kritična jer omogućava preciznije vremensko planiranje, smanjuje nesigurnost u ponašanju sistema, te je ključna za aplikacije koje zahtijevaju visoku vremensku preciznost.

Preempcija je mehanizam koji omogućava operativnom sistemu da prekine izvršavanje zadatka nižeg prioriteta u korist zadatka višeg prioriteta[8]. Ovaj mehanizam je ključan za održavanje odzivnosti na hitne događaje, implementaciju prioritetnog raspoređivanja, te garantovanje da visokoprioritetni zadaci neće biti "blokirani" od nižeprioritetnih.

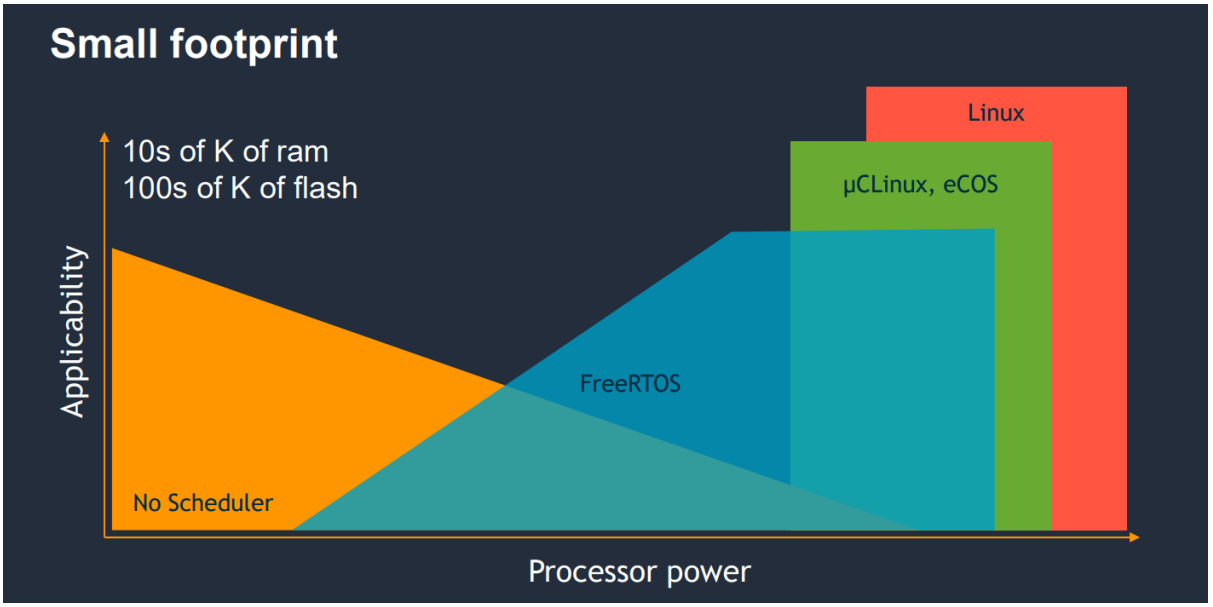
4.1.4. Uloga RTOS-a u ugradbenim sistemima

Ugradbeni sistemi karakterišu se ograničenim resursima, specifičnim funkcionalnostima i čestim zahtjevima za ponašanje u realnom vremenu[9]. RTOS pruža kritičnu infrastrukturu za efikasno upravljanje resursima u uslovima ograničenih hardverskih kapaciteta, što

podrazumijeva optimalnu alokaciju memorije kroz deterministične algoritme, upravljanje procesorskim vremenom sa predvidljivim performansama, te koordinaciju pristupa I/O resursima. Robusna sinhronizacija zadataka ostvaruje se kroz semafore za kontrolu pristupa dijeljenim resursima, mutekse sa protokolima nasljeđivanja prioriteta, te *event flagove* za koordinaciju između zadataka. Upravljanje prekidima realizuje se kroz hijerarhijsko rukovanje prekidima, garantovano maksimalno vrijeme odziva, te minimizaciju latencije kritičnih prekida.

Dijagram Poređenje RTOS sistema prema resursnim zahtjevima i aplikativnoj složenosti vizuelno klasifikuje operativne sisteme realnog vremena i srodne pristupe razvoju ugradbenih aplikacija. Horizontalna osa predstavlja procesorsku snagu, dok vertikalna osa označava primenljivost odnosno složenost aplikacija. U gornjem lijevom uglu dijagrama se nalazi oznaka "Small footprint" koja ukazuje na ograničene resursne zahtjeve, konkretno specificirajući memoriju reda veličine desetine kilobajta RAM-a i stotine kilobajta *flash* memorije.

Dijagram je podijeljen u četiri glavne oblasti. Narandžasta oblast, u donjem lijevom dijelu, označava pristup "No Scheduler" – programiranje bez operativnog sistema ili planera zadataka, pokrivajući aplikacije sa najnižim resursnim zahtjevima i jednostavnijom funkcionalnošću. Plava oblast, centralni deo dijagrama, predstavlja FreeRTOS kao reprezentativni primer RTOS sistema. Ova oblast se proteže preko značajnog dijela dijagrama, demonstrirajući široku primenljivost FreeRTOS-a za različite nivoe složenosti aplikacija, od jednostavnijih ugradbenih sistema do kompleksnijih *real-time* aplikacija, uz održavanje relativno malog *footprinta*. Zelena oblast, gornji centralni dio, obuhvata µCLinux i eCOS sisteme, koji zahtevaju veću procesorsku snagu i memorijske resurse u odnosu na FreeRTOS, ali pružaju bogatiju funkcionalnost i podršku za kompleksnije aplikacije. Konačno, crvena oblast, u gornjem desnom dijelu, predstavlja puni Linux operativni sistem, koji zahtijeva najveće resurse u smislu procesorske snage i memorije, ali omogućava razvoj najkompleksnijih aplikacija sa najširim spektrom funkcionalnosti. Dijagram jasno ilustruje kompromis između resursnih zahtjeva i funkcionalnih mogućnosti različitih pristupa, pri čemu se FreeRTOS pozicionira kao optimalno rešenje za veliki broj ugradbenih aplikacija koje zahtevaju *real-time* performanse uz ograničene resursne zahtjeve.



Slika 1: Poređenje RTOS sistema prema resursnim zahtjevima i aplikativnoj složenosti

(Izvor: https://rb.gy/i1uzb2)

4.2. FreeRTOS arhitektura i implementacija

4.2.1. Opšta arhitektura kernela

FreeRTOS predstavlja jedan od najšire korištenih *open-source* RTOS-a u industriji, sa podrškom za preko 40 različitih mikroprocesorskih arhitektura[11]. Kernel je dizajniran prema principu mikrokernela, gdje se osnovne funkcionalnosti implementiraju kao kompaktni skup servisa koji se mogu proširiti dodatnim komponentama prema potrebama aplikacije. Ova arhitektura omogućava uključivanje isključivo neophodnih komponenti, čime se optimizuje zauzeće resursa i minimizira veličina izvršnog koda.

Centralna komponenta FreeRTOS arhitekture je planer (*scheduler*) koji implementira prioritetno, preemptivno raspoređivanje zadataka. Planer koristi prioritetnu listu spremnih zadataka organizovanu kao *bitmap* struktura koja obezbjeđuje O(1) složenost operacija pronalaska zadatka sa najvišim prioritetom[12]. Ova konstanta složenost je esencijalna za predvidivost sistema jer garantuje da će vrijeme potrebno za raspoređivanje biti nezavisno od broja zadataka u sistemu.

Kontrolni blok zadatka (*Task Control Block*, TCB) predstavlja ključnu strukturu podataka koja enkapsulira sve informacije potrebne za upravljanje zadatkom. TCB se može smatrati evidencijom zadatka, koja sadrži podatke ključne za njegovo izvršavanje i upravljanje od strane operativnog sistema. To obuhvata pokazivač na stek (*stack pointer*) i prostor steka (*stack space*), koji je memorijska oblast koju zadatak koristi za čuvanje lokalnih varijabli i adresa povratka iz funkcija. Također, sadrži trenutni prioritet zadatka i njegovo stanje (npr. spreman, blokiran, izvršava se), informacije o listama čekanja i sinhronizacionim objektima, jedinstveni identifikator zadatka (*handle*), te dodatne informacije korisne za *debugging* i analizu performansi.

Upravljanje zadacima, memorijom, sinhronizacijom i komunikacijom između zadataka su ključne funkcionalnosti koje se detaljno konfigurišu putem datoteke FreeRTOSConfig.h, prilagođavajući kernel specifičnim zahtjevima ciljne platforme i aplikacije.

4.2.2. Algoritam planiranja i prebacivanje konteksta

FreeRTOS implementira algoritam planiranja sa fiksnim prioritetima i preemptivnim prekidanjem (*Fixed Priority Preemptive Scheduling*) gdje svaki zadatak ima unaprijed definisan statičan prioritet[13]. Planer uvijek bira zadatak sa najvišim prioritetom među spremnim zadacima za izvršavanje. U slučaju da više zadataka ima isti prioritet, primjenjuje se algoritam kružnog raspoređivanja (*Round Robin*) sa fiksnom vremenskom kvantom (*time slice*), što znači da se zadaci istog prioriteta naizmjenično izvršavaju na kratke, definisane vremenske intervale.

Prebacivanje konteksta (*context switching*), odnosno proces spremanja stanja trenutnog zadatka i učitavanja stanja sljedećeg zadatka, dešava se u nekoliko ključnih situacija. Jedna od njih je kada zadatak sa višim prioritetom postane spreman (*ready*) za izvršavanje, pri čemu planer odmah prekida izvršavanje tekućeg zadatka nižeg prioriteta i prebacuje kontrolu na zadatak višeg prioriteta. Također, prebacivanje konteksta nastupa kada se trenutni zadatak blokira, naprimjer, čekajući na semafor, red ili istek vremenskog perioda. Do prebacivanja dolazi i kada se trenutni zadatak završi sa izvršavanjem, kao i kada istekne vremenska kvanta kod kružnog raspoređivanja (*Round Robin scheduling*), omogućavajući drugim zadacima istog prioriteta da dobiju procesorsko vrijeme. Konačno, prebacivanje konteksta dešava se i kada zadatak eksplicitno pozove funkciju taskYIELD(), čime dobrovoljno prepušta kontrolu

procesora planeru, omogućavajući prebacivanje na drugi spremni zadatak istog ili višeg prioriteta. Vrijeme prebacivanja konteksta direktno utiče na odzivnost sistema i predstavlja jedan od ključnih pokazatelja performansi RTOS-a[14].

4.2.3. Servisi i primitivi kernela

Upravljanje zadacima (*Task Management*) obuhvata kreiranje, brisanje, suspendovanje i nastavljanje zadataka kroz API (*Application Programming Interface*) funkcije. FreeRTOS dozvoljava dinamičko kreiranje zadataka tokom izvršavanja aplikacije. Ključne funkcije za upravljanje zadacima uključuju xTaskCreate(), koja se koristi za kreiranje novog zadatka sa specificiranim prioritetom i veličinom steka; vTaskDelete(), koja služi za brisanje zadatka i oslobađanje njegovih resursa; vTaskSuspend() i vTaskResume() za privremeno zaustavljanje i nastavak izvršavanja; te vTaskDelay(), koja blokira zadatak na definisani vremenski period.

Komunikacija između zadataka (*Inter-Process Communication*, IPC) realizuje se kroz nekoliko mehanizama. **Redovi** (*Queues*) predstavljaju FIFO (*First-In, First-Out*) strukture za *thread-safe* razmjenu podataka koje omogućavaju asinhronu komunikaciju između zadataka, podržavaju blokiranje pošiljaoca kada je red pun, implementiraju *timeout* mehanizme za robusnost, te mogu prenijeti podatke proizvoljne veličine. **Semafori** služe za sinhronizaciju i kontrolu pristupa resursima. Binarni semafori koriste se kao signalizatori između zadataka, brojački semafori prate dostupnost više instanci resursa, dok rekurzivni semafori omogućavaju istom zadatku da više puta uzme semafor. **Muteksi** predstavljaju specijalizovane semafore sa dodatnim funkcionalnostima, podržavajući protokol nasljeđivanja prioriteta i sprečavajući problem prioritetne inverzije, te omogućavaju samo zadatku koji je uzeo *mutex* da ga oslobodi.

Upravljanje memorijom realizuje se kroz pet šema *heap* alokatora (heap_1 do heap_5)[15]. *Heap* je dio memorije iz kojeg programi dinamički traže prostor tokom rada. Ove šeme variraju od jednostavne alokacije bez dealokacije (heap_1), preko alokacije sa fragmentacijom (heap_2), *thread-safe* omotača oko standardnih malloc/free funkcija (heap_3), do naprednog algoritma sa kombinovanjem slobodnih blokova (heap_4) i podrške za *heap* u više memorijskih regiona (heap_5).

4.3. RISC-V instrukcijska arhitektura

4.3.1. Osnove i filozofija dizajna

RISC-V (*RISC Five*) predstavlja otvorenu instrukcijsku arhitekturu (*Instruction Set Architecture*, ISA) baziranu na etabliranim RISC principima[17]. ISA je skup svih instrukcija koje procesor razumije i može izvršiti. Za razliku od vlasničkih arhitektura (*proprietary architectures*) poput ARM ili x86, gdje se licenciranje dizajna plaća, RISC-V je dostupna pod *royalty-free open-source* licencom, što omogućava potpunu slobodu implementacije i prilagođavanja bez naknade. Filozofija RISC-V dizajna zasniva se na jednostavnosti kroz minimalan skup osnovnih instrukcija, modularnosti putem mogućnosti dodavanja funkcionalnosti kroz ekstenzije, otvorenosti kroz javno dostupne specifikacije bez restrikcija, te stabilnosti gdje se jednom ratifikovane specifikacije ne mijenjaju.

Specifikacija RISC-V ISA ratifikovana je kroz rigorozan proces standardizacije. Trenutne verzije *Unprivileged ISA 20191213* i *Privileged ISA 1.11* predstavljaju stabilnu osnovu za komercijalne implementacije[18]. Noviji RISC-V profili (RVA22 i RVA23) definišu standardizovane skupove funkcionalnosti za različite aplikacione domene[19].

4.3.2. Modularnost i ekstenzije

RISC-V koristi jedinstveni modularan pristup gdje se osnovni instrukcijski skup može proširivati standardizovanim ekstenzijama[20]. Ovaj pristup omogućava precizno prilagođavanje procesora specifičnim zahtjevima aplikacije, birajući samo potrebne funkcionalnosti.

RV32I predstavlja osnovnu 32-bitnu konfiguraciju koja implementira 32 registra opšte namjene (x0-x31) za čuvanje podataka i adresa, *Load/Store* arhitekturu sa podrškom za *word* (32-bit), *halfword* (16-bit) i *byte* (8-bit) pristup, cjelobrojne aritmetičke i logičke operacije (poput sabiranja, oduzimanja i logičkih operacija), instrukcije za kontrolu toka programa kroz uslovne i bezuslovne skokove, te sistemske instrukcije za pristup kontrolnim registrima.

M ekstenzija dodaje kritičnu podršku za cjelobrojno množenje i dijeljenje, što je kritično za mnoge ugradbene aplikacije jer se time složene matematičke operacije izvršavaju mnogo brže i efikasnije[21]. **C ekstenzija** implementira kompresovane 16-bitne instrukcije koje skraćuju najčešće korišćene 32-bitne instrukcije, smanjuju veličinu izvršnog koda za 25-30%,

poboljšavaju performanse *cache* memorije, te se automatski dekodiraju u standardne instrukcije[22]. **A ekstenzija** omogućava atomske operacije kroz *Load-Reserved*/*Store-Conditional* instrukcije i atomske *Read-Modify-Write* operacije, što je esencijalno za multiprocesorske sisteme.

4.3.3. Privilegovani režimi i sistemske funkcionalnosti

RISC-V definiše tri privilegovana režima rada[23], koji kontrolišu nivo pristupa procesora sistemskim resursima: **Mašinski režim** (*Machine mode*, M-mode) predstavlja najviši privilegovani nivo sa potpunim pristupom sistemu. Ovaj režim je jedini obavezni režim za sve RISC-V implementacije, može pristupiti svim hardverskim resursima, te je tipično korišten za *firmware*, *bootloader* i RTOS kernele. **Nadzorni režim** (*Supervisor mode*, S-mode) dizajniran je za operativne sisteme opšte namjene, ima ograničen pristup u odnosu na M-mode, te podržava virtuelnu memoriju i napredne OS funkcionalnosti. **Korisnički režim** (*User mode*, U-mode) predstavlja najniži privilegovani nivo za aplikacije, ima ograničen pristup sistemskim resursima, te omogućava izolaciju i sigurnost aplikacija. Za ugradbene RTOS aplikacije, M-mode je obično jedini implementiran režim jer omogućava direktan pristup hardveru bez *overheada*, jednostavniji je za implementaciju, te pruža maksimalnu kontrolu i determinizam.

Kontrolni i statusni registri (CSR) predstavljaju ključni interfejs za sistemsku kontrolu. To su posebni registri unutar procesora koje softver može čitati i pisati kako bi kontrolisao njegovo ponašanje, konfigurirao prekide, ili provjerio status. Primjeri CSR registara uključuju one koji kontrolišu globalnu dozvolu prekida i privilegovani režim rada, omogućavanje i praćenje statusa pojedinih tipova prekida, određivanje adrese rutine za obradu prekida, te bilježenje uzroka prekida i dodatnih informacija o izuzecima.

CLINT (*Core Local Interrupt Controller*) i **PLIC** (*Platform-Level Interrupt Controller*) su ključne komponente za upravljanje prekidima u RISC-V sistemima. **CLINT** je odgovoran za generisanje tajmerskih i softverskih prekida specifičnih za jezgro, koristeći registre za tajmer. **PLIC**, s druge strane, upravlja eksternim prekidima koji dolaze iz različitih perifernih uređaja, omogućavajući kompleksnu hijerarhiju i prioritizaciju prekida. Interakcija sa ovim kontrolerima prekida, kao i sa opštim kontrolnim i statusnim registrima procesora (CSR), ostvaruje se putem specifičnih CSRxx instrukcija Zicsr ekstenzije, koje omogućavaju

kontrolu i nadzor nad stanjem procesora i prekidima, što je esencijalno za rad operativnih sistema poput FreeRTOS-a.

4.3.4. Mehanizmi za mjerenje vremena na RISC-V-u

RISC-V specifikacija definiše nekoliko mehanizama za precizno mjerenje vremena koji su kritični za analizu performansi RTOS-a[24]. **Tajmer registri** (npr. *mtime*) predstavljaju 64-bitni brojač koji se inkrementira konstantnom frekvencijom nezavisno od frekvencije procesora. Ovi registri funkcionišu kao sistemski satovi i centralni su za implementaciju sistemskog *ticka* u RTOS-ima, omogućavajući precizna mjerenja izvršavanja. Pored toga, **registri za nadzor performansi** (npr. *mcycle* i *minstret*) implementiraju brojače za procesorske cikluse i izvršene instrukcije. *Mcycle* broji ukupan broj procesorskih ciklusa koje je procesor izvršio, dok *minstret* broji ukupan broj izvršenih instrukcija. Ovi registri omogućavaju detaljnu analizu performansi na nivou instrukcija i predstavljaju kritičan alat za karakterizaciju RTOS primitiva. Platformski specifične implementacije tajmera mogu proširiti osnovnu funkcionalnost dodatnim mogućnostima poput programabilnih tajmera i *watchdog* funkcionalnosti. Pristup ovim registrima ostvaruje se putem specifičnih ekstenzija koje definišu instrukcije za čitanje, pisanje, postavljanje i brisanje bitova registra. Za praktičnu verifikaciju RTOS performansi na RISC-V arhitekturi, potrebne su odgovarajuće razvojne i simulacione platforme.

4.4. Platforme za razvoj i verifikaciju ugradbenih sistema

Za razvoj i testiranje ugradbenih sistema, posebno onih baziranih na novim arhitekturama poput RISC-V, ključno je koristiti adekvatne platforme koje omogućavaju efikasnu simulaciju i otklanjanje grešaka. Ove platforme mogu biti softverski emulacioni alati ili fizičke hardverske implementacije na FPGA pločama.

4.4.1. QEMU emulator kao simulaciona platforma

QEMU (*Quick EMUlator*) predstavlja najsveobuhvatniji *open-source* emulator koji podržava preko 30 različitih arhitektura, uključujući RISC-V[25]. Za ovaj rad, QEMU je izabran kao primarna simulaciona platforma zbog svojih naprednih mogućnosti emulacije i kontrolisanog okruženja koje omogućava. QEMU funkcioniše kao **dinamički binarni prevodilac** (*dynamic binary translator*)[27], koji u realnom vremenu prevodi instrukcije ciljne arhitekture

(RISC-V) u instrukcije host procesora (obično x86). Ovaj pristup omogućava relativno brzo izvršavanje emuliranog koda, emulaciju kompletnog sistema (a ne samo procesora), te potpunu kontrolu nad emuliranim okruženjem.

Najnovije verzije QEMU-a (8.0, 9.0, 9.2) donijele su značajna poboljšanja u RISC-V podršci[26]. Ova poboljšanja obuhvataju naprednu arhitekturu prekida (*Advanced Interrupt Architecture*, AIA), vektorske kriptografske instrukcije, optimizacije performansi za RISC-V emulaciju, te poboljšanu podršku za privilegovane instrukcije. Za ovaj rad korištena je QEMU "virt" mašina koja predstavlja generičku virtuelnu platformu sa standardnim komponentama.

Prednosti QEMU-a za RTOS analizu obuhvataju uspostavljanje kontrolisanog okruženja za reproduktivne eksperimente, *plugin* sistem za precizno nadgledanje performansi, GDB integraciju koja omogućava *step-by-step debugging*, te UART redirekciju za automatsko prikupljanje rezultata. Upotreba QEMU-a donosi i određena ograničenja. Emulaciono opterećenje (*overhead*) može maskirati pravo determinističko ponašanje, aproksimacija vremena možda ne reflektuje precizno hardverske latencije, dok ograničena kompleksnost znači da emulacija periferija može biti jednostavnija od stvarnog hardvera.

4.4.2. Alternativni simulatori

Pored QEMU emulatora, u ranim fazama istraživanja i razvoja RISC-V sistema često se koriste i drugi tipovi simulatora, posebno oni fokusirani na emulaciju skupa instrukcija. **RISC-V ISS** (*Instruction Set Simulator*) predstavlja *open-source* simulator skupa instrukcija za 32-bitne RISC-V procesore, čija je implementacija dostupna kao GitHub repozitorijum[28]. Ovaj simulator razvijen je kao visoko konfigurabilan i proširiv C++ model RISC-V jezgra, sposoban za izvršavanje koda brzinama većim od 10 MIPS-a (*Millions of Instructions Per Second*).

Prednosti ovog ISS-a, relevantne u početnim fazama razvoja, uključuju minimalno opterećenje (*overhead*) i brze simulacije, što ga čini pogodnim za brzu validaciju *bare-metal* koda i testiranje osnovnih funkcionalnosti. On nudi jasnu preglednost (*visibility*) u izvršavanje instrukcija, omogućavajući detaljnu analizu ponašanja procesora na nivou pojedinačnih instrukcija, te je jednostavan za korištenje za osnovne testove i razumijevanje RISC-V ISA (*Instruction Set Architecture*). Simulator podržava ključne RISC-V ekstenzije

(poput M za množenje/dijeljenje, A za atomske operacije, F/D za *floating point*), kao i privilegovani mašinski režim (*Machine mode*, M-mode) i rukovanje prekidima (*interrupt handling*) putem CSR registara. Također, omogućava konfigurisanje modela vremena izvršavanja instrukcija i korištenje internog tajmer modela koji broji cikluse procesora (mcycle) ili instrukcije (minstret), što je bilo korisno za rano mjerenje osnovnih operacija. Dodatno, podržava učitavanje ELF (*Executable and Linkable Format*) programa u memoriju i integraciju sa GDB (*GNU Debugger*) za otklanjanje grešaka (*debugging*). Iskustvo rada sa ovim ISS-om pružilo je korisne spoznaje o izvršavanju RISC-V instrukcija na osnovnom nivou i o osnovnim principima simulacije, postavljajući temelj za razumijevanje kompleksnijih emulacionih okruženja.

Međutim, ograničenja ovog ISS-a čine ga nepogodnim za sveobuhvatnu analizu performansi i verifikaciju *real-time* operativnih sistema poput FreeRTOS-a, zbog čega je za potrebe ovog rada odabran QEMU kao primarna simulaciona platforma. Glavna ograničenja obuhvataju nedostatak emulacije složenih periferija (poput UART-a, naprednih tajmera, kontrolera prekida), koje su ključne za funkcionalnost RTOS-a i interakciju sa periferijama, kao i krajnjim korisnikom. Također, prisutna je ograničena podrška za funkcionalnosti na sistemskom nivou koje su neophodne za rad kompletnog operativnog sistema (npr. složeno upravljanje memorijom, virtuelna memorija), te nema integraciju sa standardnim alatima za otklanjanje grešaka na nivou cijelog sistema, što otežava praćenje interakcija između kernela i aplikativnih zadataka.

4.4.3. *Universal Asynchronous Receiver/Transmitter* (UART)

Universal Asynchronous Receiver/Transmitter (UART) predstavlja jedan od najstarijih i najčešće korištenih serijskih komunikacionih protokola u ugradbenim sistemima. Njegova primarna funkcija je omogućavanje asinhronog serijskog prijenosa podataka bit po bit preko jedne linije za prijenos (Tx) i primaju preko druge linije (Rx). UART je ključan za komunikaciju između mikrokontrolera i perifernih uređaja, kao što su senzori, moduli za bežičnu komunikaciju, GPS prijemnici, ili za uspostavljanje veze sa host računarom.

UART je ključan u ugradbenim sistemima iz više razloga. Prije svega, za debugovanje i dijagnostiku, UART je *de facto* standard za ispis debug poruka i dijagnostičkih informacija iz ugradbenih sistema na konzolu host računara; bez grafičkog interfejsa, serijski ispis je često

jedini način da se prati unutrašnje stanje sistema. Nadalje, njegova **jednostavnost** čini ga idealnim za resursno ograničene sisteme, budući da je protokol relativno jednostavan za implementaciju i ne zahtijeva kompleksnu hardversku podršku. U kontekstu ovog diplomskog rada, UART je bio esencijalan za prikupljanje sirovih podataka o performansama, jer su sva mjerenja (npr. izmjereni ciklusi za latenciju kontekstnog prebacivanja) ispisivana putem emuliranog UART-a iz QEMU-a, a zatim su preusmjeravana u log datoteke za dalju analizu Python skriptama.

Iako je jednostavan, UART može predstavljati određene izazove, posebno u *real-time* kontekstu. Jedan od izazova je usklađivanje brzine (*Baud Rate*), pri čemu oba uređaja (pošiljalac i primalac) moraju biti konfigurisana na istu brzinu prijenosa podataka; neusklađenost rezultira pogrešnim prijemom podataka. Drugi izazov su blokirajuće operacije: jednostavne implementacije UART drajvera često koriste blokirajuće operacije (npr. čekanje u petlji dok se karakter ne pošalje), što u *real-time* sistemima može uvesti nepredvidive latencije i *jitter*, što je neprihvatljivo za kritične zadatke. Naprednije implementacije ublažavaju ovaj problem korištenjem prekida i FIFO (*First-In, First-Out*) bafera kako bi se minimizirao uticaj na performanse. Također, ograničena propusnost čini UART relativno sporim protokolom u poređenju sa paralelnim ili bržim serijskim protokolima (npr. SPI, I2C), što ga čini neprikladnim za prijenos velikih količina podataka. Konačno, za razliku od mrežnih protokola, UART zahtijeva direktnu fizičku vezu između uređaja (obično samo dvije ili tri žice), što ograničava udaljenost i broj povezanih uređaja.

U ovom radu, QEMU je emulirao UART funkcionalnost, omogućavajući komunikaciju između emuliranog RISC-V sistema i host računara, što je bilo ključno za automatsko prikupljanje podataka o performansama.

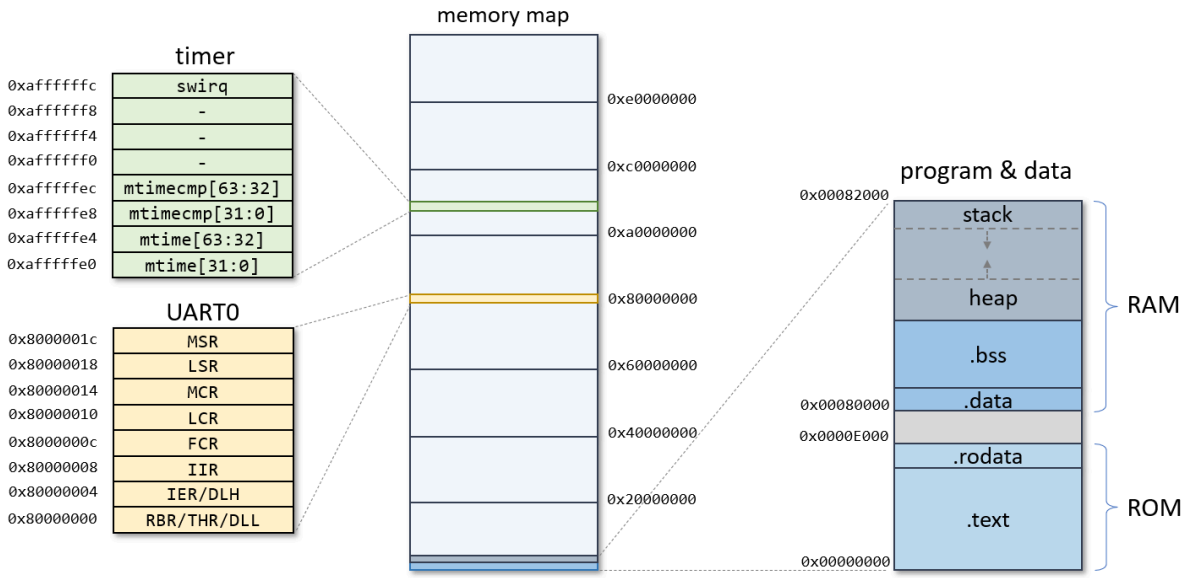
4.4.4. FPGA platforme (Tang Nano 9K)

FPGA (*Field-Programmable Gate Array*) tehnologija omogućava implementaciju prilagođenih digitalnih kola kroz rekonfigurabilne logičke blokove[29]. Za razliku od mikroprocesora gdje je arhitektura fiksna, FPGA je poput "praznog platna" na kojem inženjeri mogu kreirati vlastiti digitalni hardver, uključujući i cijeli procesor. Tang Nano 9K ploča, bazirana na Gowin GW1NR-9 FPGA čipu, predstavlja dostupnu platformu za implementaciju RISC-V *soft-core* procesora. *Soft-core* procesor znači da procesor nije

zaseban fizički čip, već je njegov dizajn ("kod" napisan u hardverskom opisu) "programiran" unutar logičkih vrata same FPGA ploče.

Sistem-na-čipu (*System-on-Chip*, SoC) dizajn na FPGA platformi podrazumijeva integraciju RISC-V CPU jezgra (RV32IM) sa svim potrebnim perifernim jedinicama na jednoj FPGA ploči. To uključuje kontroler memorije za BRAM pristup (koji omogućava procesoru da koristi BRAM kao RAM), UART kontrolerom za serijsku komunikaciju, programabilnim tajmerom za generisanje sistemskog *ticka*, jednostavnim kontrolerom prekida i SPI kontrolerom za pristup vanjskoj *flash* memoriji (gdje se obično čuva program).

Memorijska mapa projektovanog RISC-V SoC-a na Tang Nano 9K platformi prikazana je na Slici 8. Ova mapa detaljno definiše adresne opsege za memoriju instrukcija i podataka, kao i za periferijske registre i SPI *flash* memorijski prostor. Jasno definisanje memorijske mape je ključno za ispravno funkcionisanje ugradbenih sistema, jer omogućava softveru da pravilno pristupi hardverskim resursima.



Slika 2: Projektovana mapa memorije RISC-V SoC-a na Tang Nano 9K platformi

(Izvor: https://github.com/wyvernSemi/riscV)

4.4.5. Izazovi hardverske implementacije

Implementacija RISC-V sistema na FPGA platformama nosi specifične inženjerske izazove, koji su važni za razumijevanje praktičnih aspekata razvoja ugradbenih sistema. To uključuje vremenska ograničenja (*timing constraints*), gdje FPGA dizajn mora zadovoljiti *setup* i *hold* vremenske zahtjeve kako bi digitalna kola ispravno radila na ciljanoj frekvenciji, što zahtijeva optimizaciju vremenskih putanja i logike. Zatim, tu je korištenje resursa (*resource utilization*), što podrazumijeva balansiranje željenih funkcionalnosti i dostupnih FPGA resursa, jer prevelik dizajn jednostavno ne može stati na čip. Programiranje i otklanjanje grešaka (*debugging*) predstavljaju izazov zbog ograničenih mogućnosti *debugginga* u odnosu na simulatore, potrebe za JTAG interfejsima i *logic analyzerima*, te složenosti praćenja signala u realnom vremenu. Konačno, izazovi integracije alata (*toolchain integration*) obuhvataju sinhronizaciju između HDL alata i FPGA *vendor* alata, kompatibilnost različitih verzija softvera, te automatizaciju procesa kompajliranja i programiranja.

4.5. Osvrt na postojeća rješenja i srodna istraživanja

Ovo poglavlje pruža pregled relevantne literature i postojećih rješenja u oblasti analize performansi operativnih sistema za rad u realnom vremenu na RISC-V arhitekturi, kao i komparativnih studija i uloge simulacionih platformi. Cilj je pozicionirati ovo istraživanje u širi akademski kontekst i identifikovati praznine u postojećem znanju.

4.5.1. Analiza performansi RTOS-a na RISC-V arhitekturi

Dosadašnja istraživanja performansi FreeRTOS-a na RISC-V arhitekturi bila su ograničenog opsega. Istraživanje iz 2023. godine[30] sprovelo je eksploraciju FreeRTOS-a na RISC-V koristeći SPIKE simulator, fokusirajući se na vrijeme kreiranja i brisanja zadataka, latenciju prebacivanja konteksta, *mutex* operacije i sinhronizaciju, te vrijeme podizanja operativnog sistema (*boot time*). Njihovi rezultati pokazali su da RISC-V arhitektura pruža konkurentne performanse u odnosu na tradicionalne ugradbene arhitekture, ali studija je bila ograničena na osnovne metrike bez detaljne WCET analize. Pouzdanost *bare-metal* i FreeRTOS aplikacija na Microchip PolarFire SoC RISC-V multiprocesorima u kontekstu svemirskih aplikacija analizirana je u istraživanju iz 2024. godine[31]. Ova studija je pokazala da FreeRTOS implementacije mogu pružiti dodatne slojeve otpornosti (*resiliency*) kroz izolaciju zadataka

između CPU jezgara, mehanizme oporavka od grešaka (*error recovery*), te redundantnu arhitekturu za kritične operacije.

4.5.2. Komparativne studije RTOS performansi

Sveobuhvatnu analizu vremenskih performansi različitih RTOS-a na ARM Cortex-M4 arhitekturi sprovedeno je u istraživanju iz 2020. godine[32]. Studija se fokusirala na vrijeme prebacivanja zadataka (*task switching time*), latenciju prekida (*interrupt latency*), performanse sinhronizacionih primitiva (semafori, muteksi), te *overhead* kernela u različitim scenarijima opterećenja. Rezultati su pokazali da FreeRTOS ima izuzetne performanse sa najnižim vrijednostima vremena prebacivanja i prekidanja među testiranim RTOS-ima. Međutim, ova studija nije obuhvatila RISC-V arhitekturu. Pregled operativnih sistema za rad u realnom vremenu za ugradbene aplikacije, koji naglašava rastući značaj determinizma u IoT i *edge computing* aplikacijama, energetske efikasnosti za *battery-powered* uređaje, te skalabilnosti za različite performanse procesora, predstavljen je je u radu iz 2024. godine[33].

4.5.4. Identifikovane praznine u postojećem znanju

Analiza postojeće literature ukazuje na nekoliko ključnih praznina koje ovo istraživanje nastoji popuniti.

Prvo, uočen je nedostatak sveobuhvatnih kvantitativnih podataka. To se manifestuje kroz fokus većine studija na kvalitativnu analizu, ograničen skup testiranih metrika, nedostatak statistički značajnih uzoraka mjerenja, te odsustvo standardizovane metodologije za *benchmarking*.

Drugo, prisutna je ograničena reproduktivnost postojećih studija. To je uslijed nedostatka detaljnih implementacionih detalja, kompletnih *source* kodova testova, varijabilnih konfiguracionih parametara između studija, te odsustva standardizovanih test svita za RTOS *benchmarking*.

Treće, nedovoljna WCET (*Worst-Case Execution Time*) analiza je također identifikovana kao praznina. Postojeće studije uglavnom su pružale prosječne vrijednosti bez empirijski izvedenih WCET-ova na visokim percentilima, te bez detaljne analize *jittera* kao ključnog faktora determinizma.

Konačno, literatura često ne obrađuje praktične izazove implementacije i verifikacije na stvarnim ugradbenim platformama, što ukazuje na nedostatak analize hardverske verifikacije izazova.

Ovo istraživanje nastoji popuniti navedene praznine kroz sveobuhvatnu kvantitativnu analizu FreeRTOS performansi na RISC-V-u, koristeći reproduktivnu metodologiju i fokusirajući se na praktične inženjerske uvide. Sljedeće poglavlje će detaljno opisati konkretno rješenje koje popunjava ove identifikovane praznine.

5. Realizacija i kvantitativna analiza performansi

Ovo poglavlje detaljno opisuje metodologiju, implementaciju i rezultate analize performansi i verifikacije funkcionalnosti FreeRTOS-a na RISC-V arhitekturi u QEMU emulatoru. Predstavljeni pristup obuhvata dizajn testnog okruženja, implementaciju specifičnih testova performansi, automatizovani proces prikupljanja i obrade podataka, te analizu dobijenih rezultata i verifikaciju ključnih funkcionalnosti kroz demonstracione aplikacije.

5.1. Arhitektura testnog okruženja

Za potrebe sveobuhvatne analize performansi FreeRTOS-a na RISC-V arhitekturi, uspostavljeno je kontrolisano testno okruženje bazirano na QEMU emulatoru. **Arhitektura ovog testnog okruženja prikazana je na Slici 2.** Ona obuhvata sve ključne komponente: emulirani RISC-V procesor unutar QEMU-a, FreeRTOS kernel portovan za ovu arhitekturu, implementirane testove performansi, UART serijsku konzolu za izlaz podataka te Python skripte za automatizovanu analizu.

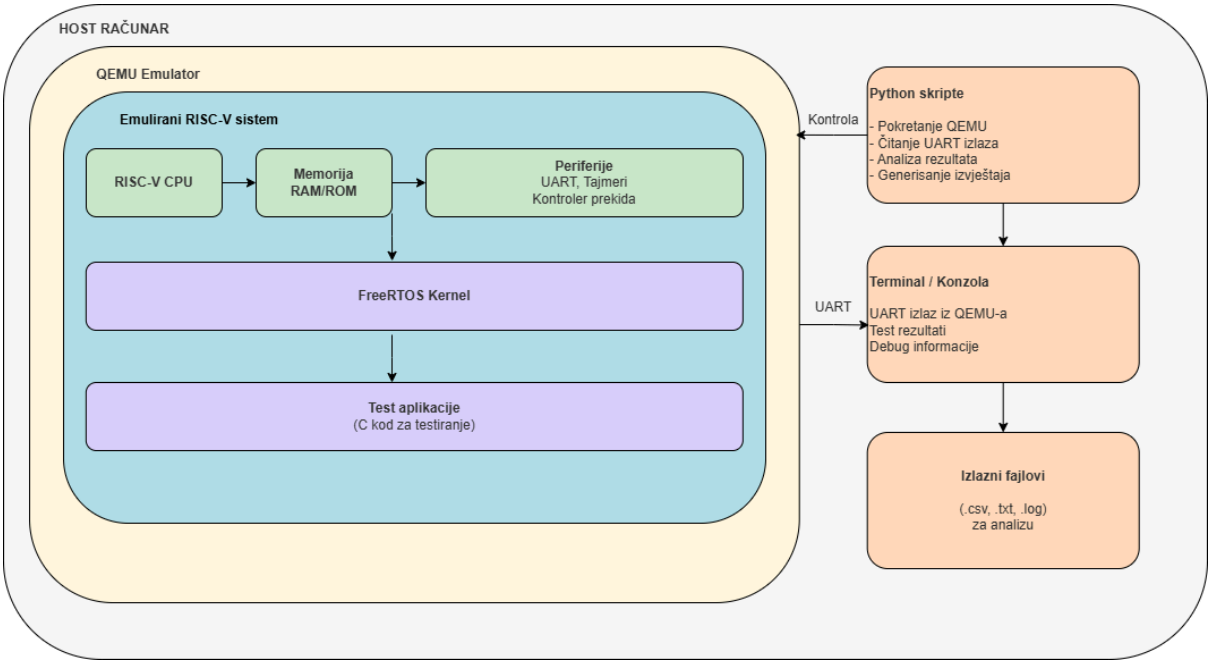
Softverske komponente korištene u testnom okruženju uključuju specifične verzije alata i biblioteka. Korišten je QEMU emulator (verzija 9.2, decembar 2024. godine), RISC-V GCC *toolchain* (verzija 12.2.0), FreeRTOS kernel (verzija 10.4.6) te Python (verzija 3.9) sa pyserial bibliotekom za serijsku komunikaciju. Precizna specifikacija ovih komponenti osigurava reproduktivnost postignutih rezultata.

Važno je napomenuti da su u ranim fazama istraživanja razmatrani i alternativni simulacioni alati, uključujući RISC-V ISS (Instruction Set Simulator), koji je detaljnije opisan u Poglavlju 4.4.2. Iako je ISS bio koristan za razumijevanje izvršavanja instrukcija na osnovnom nivou, QEMU je odabran kao primarna platforma zbog svojih superiornih mogućnosti emulacije složenih perifernih uređaja i podrške za sistemske funkcionalnosti neophodne za rad RTOS-a.

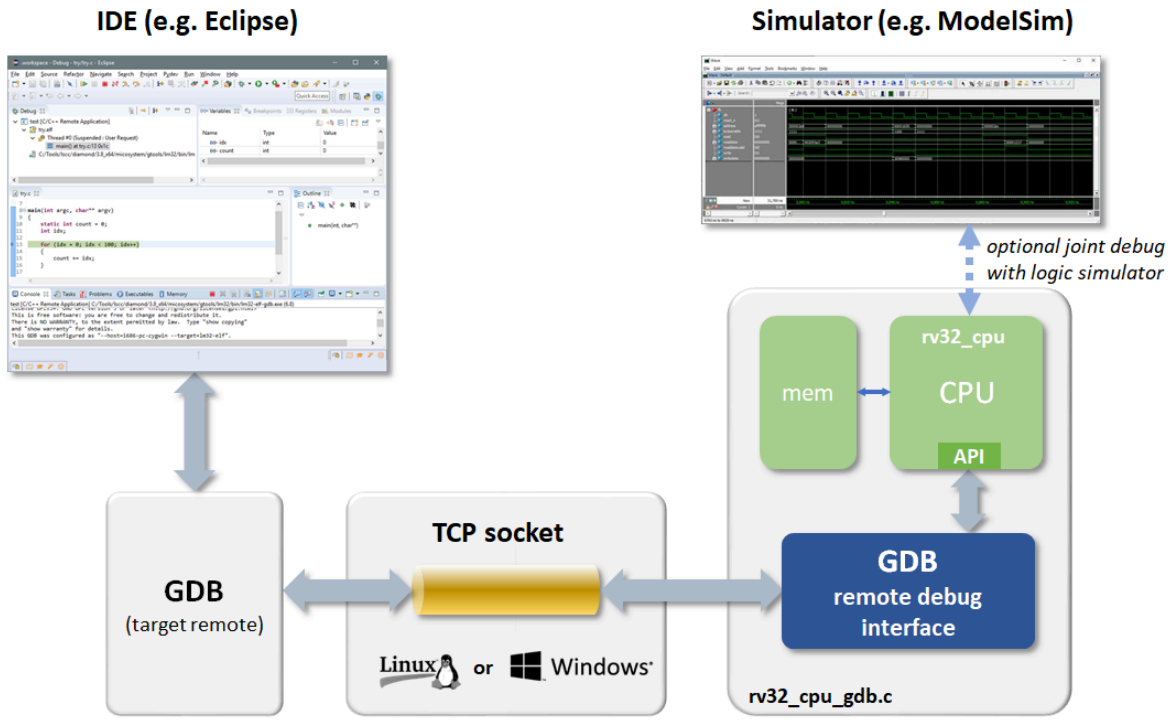
Konfiguracija FreeRTOS-a, definisana u datoteci FreeRTOSConfig.h, imala je direktan uticaj na performanse sistema. Parametri poput configCPU_CLOCK_HZ (frekvencija procesora korištena za proračune vremena), configTICK_RATE_HZ (frekvencija sistemskog takta koja definiše granularnost vremenskih operacija), configMINIMAL_STACK_SIZE (minimalna veličina steka za zadatke), configTOTAL_HEAP_SIZE (ukupna veličina *heap* memorije

dostupne FreeRTOS-u) i configMAX_PRIORITIES (maksimalan broj prioritetnih nivoa) pažljivo su podešeni kako bi se optimizovale performanse u emuliranom okruženju.

Proces razvoja i otklanjanja grešaka (*debugging*) FreeRTOS aplikacija na RISC-V arhitekturi u simulacionom okruženju ključan je za efikasnost istraživanja. **Arhitektura sistema za otklanjanje grešaka i simulaciju prikazana je na Slici 3.** Ova šema ilustruje kako razvojno okruženje (*Integrated Development Environment*, IDE), poput Eclipse-a, komunicira sa simuliranim RISC-V procesorom. Komunikacija se ostvaruje putem GDB (*GNU Debugger*) protokola, gdje IDE funkcioniše kao GDB klijent, a simulator (u ovom slučaju QEMU) kao GDB server. Povezivanje se uspostavlja preko TCP *socketa*, omogućavajući daljinsko otklanjanje grešaka i nadzor nad izvršavanjem koda na emuliranom procesoru. Ova fleksibilna arhitektura omogućava efikasno testiranje i validaciju softvera, kao i detaljnu analizu ponašanja sistema u kontrolisanom okruženju.



Slika 2: Arhitektura testnog okruženja za analizu performansi FreeRTOS-a na RISC-V arhitekturi u QEMU emulatoru



Slika 3: Arhitektura sistema za otklanjanje grešaka i simulaciju RISC-V procesora

(Izvor: https://github.com/wyvernSemi/riscV)

5.2. Dizajn i implementacija testova performansi

Za preciznu karakterizaciju performansi FreeRTOS primitiva na RISC-V arhitekturi, dizajniran je i implementiran set specifičnih testova. Opšti pristup se zasnivao na izolaciji svake pojedinačne operacije koja se mjeri, osiguravajući minimalan uticaj vanjskih faktora i maksimalnu reproduktivnost rezultata. Svaki test je implementiran kao zasebna C funkcija koja se izvršava unutar FreeRTOS okruženja.

Mjerenje vremena izvršavanja operacija ostvareno je korištenjem hardverskih brojača ciklusa dostupnih na RISC-V arhitekturi, prvenstveno mtime registra. Vrijednost mtime registra se očitavala prije i poslije izvršenja testirane operacije, a razlika u broju ciklusa direktno je predstavljala vrijeme izvršavanja. Ovi ciklusi su zatim konvertovani u mikrosekunde, uzimajući u obzir poznatu frekvenciju procesora, što je omogućilo kvantitativnu analizu performansi u realnom vremenu.

Implementirane test aplikacije obuhvatile su ključne aspekte ponašanja RTOS-a:

Upravljanje zadacima (Task Management): Testovi su dizajnirani za mjerenje vremena potrebnog za kreiranje novog zadatka i njegovo brisanje iz sistema. Logika C koda uključivala je pozive funkcija xTaskCreate() i vTaskDelete() unutar kontrolisane petlje, sa očitavanjem mtime registra prije i poslije svake operacije.

Kontekstno prebacivanje (Context Switching): Ova metrika je ključna za odzivnost RTOS-a. Test je obuhvatio kreiranje dva zadatka visokog prioriteta koji naizmjenično prepuštaju kontrolu procesora koristeći funkciju taskYIELD(). Vrijeme prebacivanja konteksta mjereno je unutar svakog zadatka, bilježeći trenutak kada se kontrola prebacuje i kada se vraća.

Upravljanje memorijom (Memory Management): Testovi su analizirali performanse dinamičke alokacije i dealokacije memorije korištenjem FreeRTOS funkcija pvPortMalloc() i vPortFree(). Mjerenja su vršena za različite veličine memorijskih blokova kako bi se procijenila efikasnost heap alokatora.

Komunikacija između zadataka (Inter-Process Communication, IPC): Performanse komunikacionih primitiva su detaljno analizirane: Redovi (Queues): Mjereno je vrijeme slanja i primanja poruka različitih veličina putem FreeRTOS redova. Test je uključivao jedan zadatak koji šalje poruke i drugi koji ih prima, bilježeći latenciju svake operacije. Semafori i Muteksi (Mutexes): Analizirane su latencije operacija davanja i uzimanja semafora (xSemaphoreGive(), xSemaphoreTake()) i muteksa (xSemaphoreTake(), xSemaphoreGive()). Posebna pažnja posvećena je testiranju scenarija sa konfliktima pristupa resursima.

Da bi se ilustrovala primijenjena metodologija, Slika 4 prikazuje detaljan dijagram toka algoritma korištenog za mjerenje latencije dvostrukog prebacivanja konteksta. Ovaj test služi kao reprezentativan primjer pristupa koji je primijenjen na sve navedene testove performansi. Primarni cilj ovog specifičnog testa je kvantifikovati ukupno vrijeme potrebno da se kontrola procesora prebaci sa jednog zadatka na drugi i zatim vrati nazad.

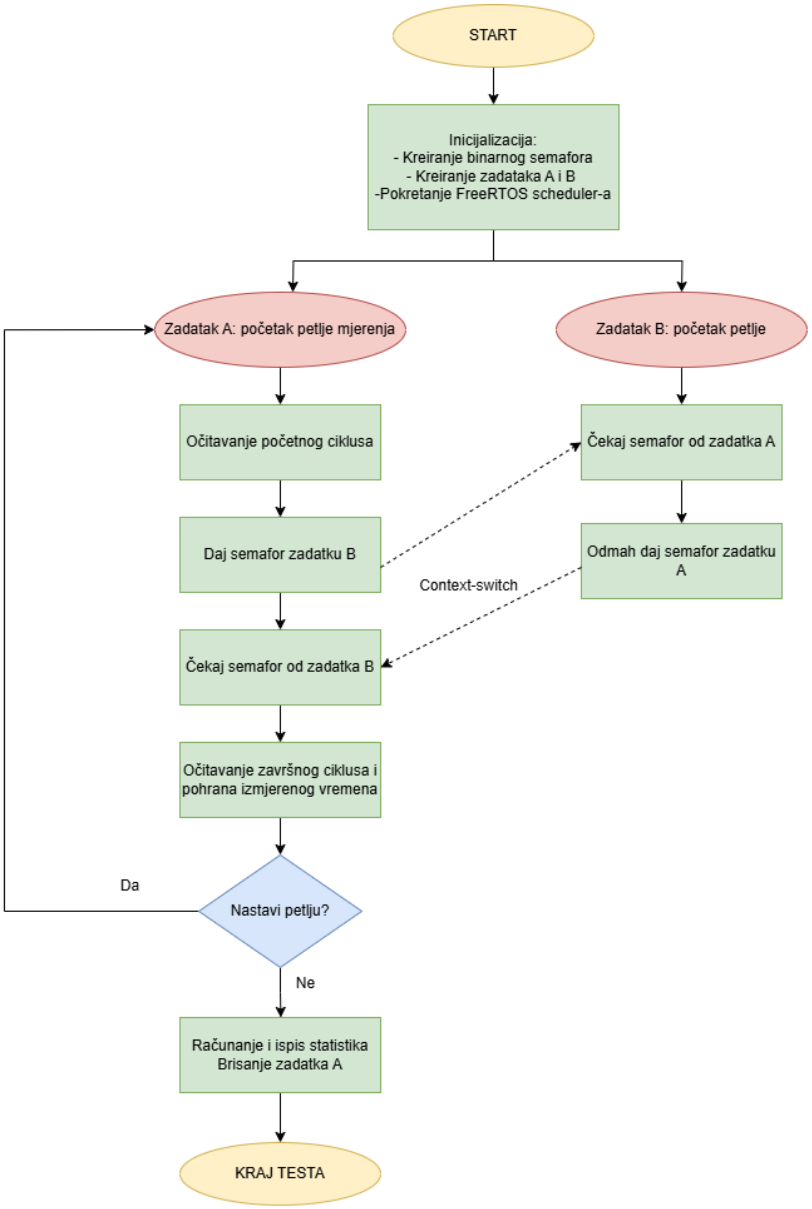
Proces započinje fazom inicijalizacije gdje se kreira binarni semafor, koji služi kao ključni mehanizam za sinhronizaciju i signalizaciju između dva zadatka. Nakon toga, kreiraju se dva zadatka visokog prioriteta, označena kao Zadatak A i Zadatak B, koji će biti uključeni u

mjerenje. Po završetku inicijalizacije, pokreće se FreeRTOS planer (*scheduler*), preuzimajući kontrolu nad raspoređivanjem svih zadataka u sistemu.

Nakon što planer preuzme kontrolu, Zadatak A započinje petlju mjerenja. U prvom koraku, Zadatak A očitava trenutnu vrijednost hardverskog brojača ciklusa (npr. mtime registra), što predstavlja početnu vremensku referencu za mjerenje. Odmah zatim, Zadatak A daje binarni semafor zadatku B, signalizirajući mu da je spreman za izvršavanje i da može preuzeti kontrolu procesora. U tom trenutku dolazi do prvog prebacivanja konteksta, sa Zadatka A na Zadatak B.

Paralelno, Zadatak B čeka na semafor koji mu daje Zadatak A. Čim primi semafor, Zadatak B preuzima kontrolu procesora i odmah vraća semafor nazad zadatku A. Ova akcija izaziva drugo prebacivanje konteksta, sa Zadatka B nazad na Zadatak A, čime se kompletira jedan ciklus dvostrukog prebacivanja.

Po povratku kontrole, Zadatak A ponovo čeka na semafor od zadatka B. Kada ga primi, to označava završetak jednog kompletnog ciklusa dvostrukog prebacivanja konteksta (A → B → A). Zadatak A tada očitava završnu vrijednost brojača ciklusa. Razlika između početnog i završnog očitavanja predstavlja izmjereno vrijeme za taj jedan ciklus dvostrukog prebacivanja, koje se zatim pohranjuje. Ovaj proces se ponavlja u petlji sve dok se ne dostigne unaprijed definisan broj iteracija mjerenja. Nakon što se završe sva mjerenja, prikupljeni podaci se statistički obrađuju, uključujući računanje prosječnog, minimalnog i maksimalnog vremena dvostrukog prebacivanja konteksta. Na kraju testa, Zadatak A se briše iz sistema, oslobađajući zauzete resurse



Slika 4: Dijagram toka za test mjerenja latencije kontekstnog prebacivanja

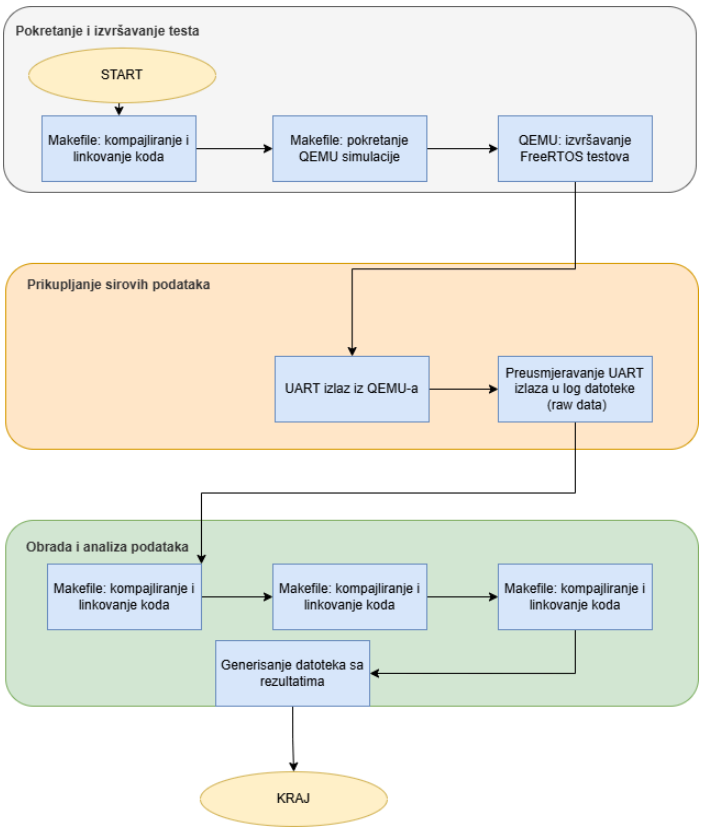
5.3. Proces prikupljanja i obrade podataka

Da bi se osigurala pouzdanost i sveobuhvatnost rezultata, uspostavljen je automatizovani proces prikupljanja i obrade podataka. **Dijagram toka ovog procesa prikazan je na Slici 5.**

Automatizacija je ostvarena korištenjem **Makefile** skripti koje su orkestrirale cijeli proces. *Makefile* je bio odgovoran za kompajliranje C koda test aplikacija, linkovanje sa FreeRTOS kernelom i RISC-V bibliotekama, te pokretanje QEMU simulacija sa definisanim parametrima, uključujući broj iteracija za svaki test.

Prikupljanje sirovih podataka vršeno je preusmjeravanjem UART izlaza iz QEMU emulatora direktno u tekstualne datoteke. Svaka iteracija testa ispisivala je relevantne podatke (npr. izmjerene cikluse) na serijsku konzolu, a ti podaci su automatski bilježeni.

Analiza podataka sprovedena je korištenjem Python skripti, konkretno freertos_analyzer.py i simple_report.py. Ove skripte su dizajnirane za parsiranje sirovih logova, što uključuje čitanje tekstualnih datoteka i ekstrakciju numeričkih podataka iz svake linije. Nakon toga, vrši se ekstrakcija relevantnih podataka, odnosno izolovanje izmjerenih vrijednosti ciklusa za svaku operaciju. Konačno, skripte su zadužene za generisanje statističkih metrika, što podrazumijeva izračunavanje minimuma, maksimuma, prosjeka i standardne devijacije za svaku grupu mjerenja; ove metrike su ključne za razumijevanje centralne tendencije i varijabilnosti performansi.



Slika 5: Dijagram toka procesa prikupljanja i analize podataka

5.4. Rezultati i analiza performansi u QEMU emulatoru

Ovo poglavlje predstavlja kvantitativne rezultate dobijene analizom performansi FreeRTOS primitiva na RISC-V arhitekturi u QEMU emulatoru, te pruža detaljnu interpretaciju tih rezultata.

Prezentacija rezultata izvršena je kroz kombinaciju tabela i grafičkih prikaza. Za svaku testiranu metriku (kreiranje zadatka, kontekstno prebacivanje, alokacija/dealokacija memorije, operacije sa semaforima i redovima) predstavljene su jasne tabele koje sadrže minimalne, maksimalne i prosječne vrijednosti izmjerene u ciklusima, kao i njihove ekvivalentne vrijednosti u mikrosekundama.

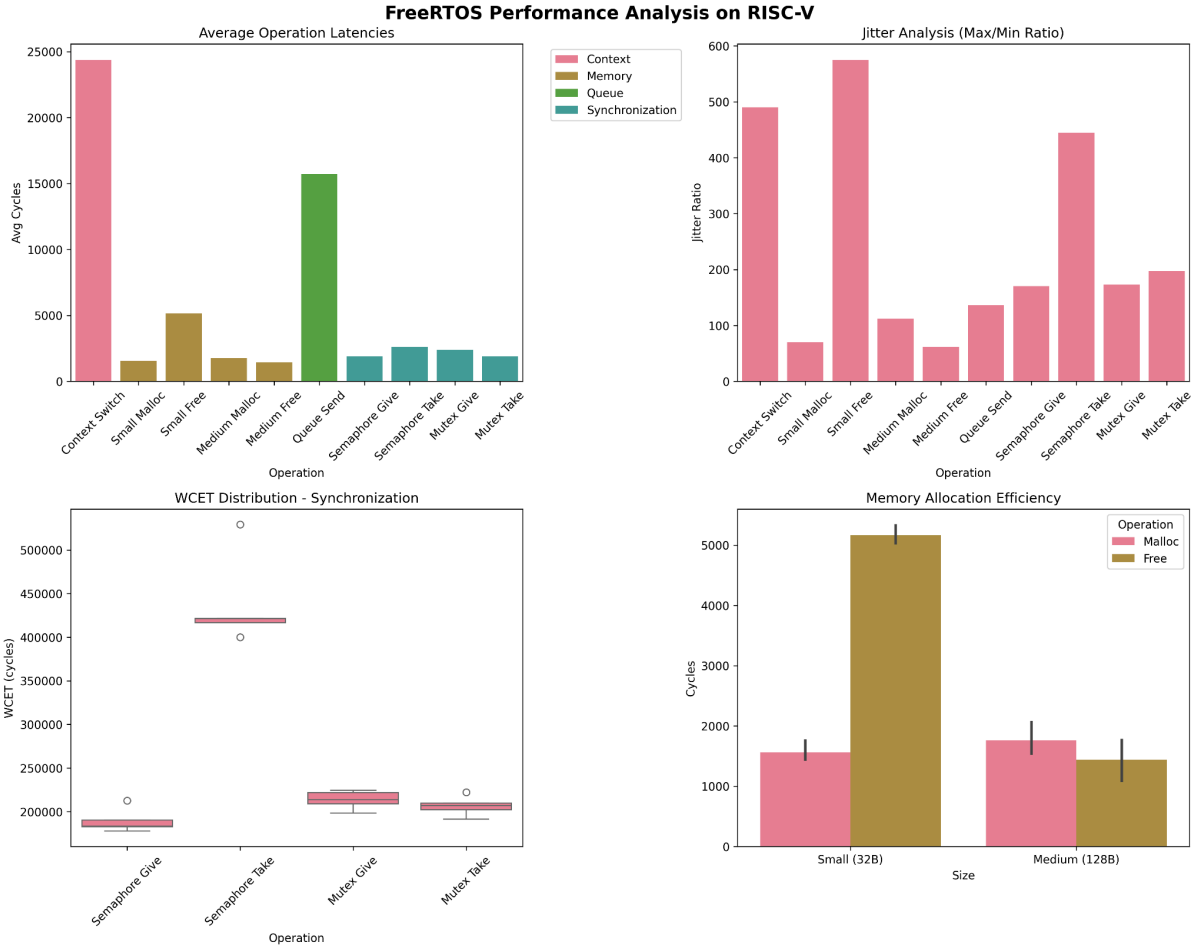
Pored toga, **box plotovi** su korišteni za vizuelni prikaz distribucije izmjerenih vremena. Ovi grafici omogućavaju detaljan uvid u medijanu, kvartile, raspon podataka i prisustvo odstupanja (*outlier*-a), što je ključno za analizu *jittera*.

Tabela 1: Rezultati mjerenja latencije kontekstnog prebacivanja

Metrika	Ciklusi	Vrijeme (µs) (pri 25 MHz)
Prosječno vrijeme dvostrukog prebacivanja konteksta	24363.6	974.5
Minimalno zabilježeno dvostruko prebacivanje konteksta	7754	310.16
Maksimalno zabilježeno dvostruko prebacivanje konteksta	11126974	445078.96
Procijenjeno prosječno vrijeme jednostrukog prebacivanja konteksta	12181.8	487.27
WCET (95%) dvostrukog prebacivanja konteksta	9772341.4	390893.7
WCET (99%) dvostrukog prebacivanja konteksta	10856047.5	434241.9

Odnos jittera	490.1x	-
Koeficijent varijacije	0.378	-

Na **Slici 6**, *box plot* za latenciju kontekstnog prebacivanja vizuelno predstavlja centralnu tendenciju (medijan), širenje podataka (interkvartilni raspon) i asimetriju distribucije, što je ključno za procjenu *jittera* i determinizma. Donji i gornji "brkovi" (*whiskers*) obično označavaju raspon podataka koji su unutar 1.5 puta interkvartilnog raspona od gornjeg ili donjeg kvartila, dok pojedinačne tačke izvan tog raspona predstavljaju odstupanja.



Slika 6: Box plot za latenciju kontekstnog prebacivanja

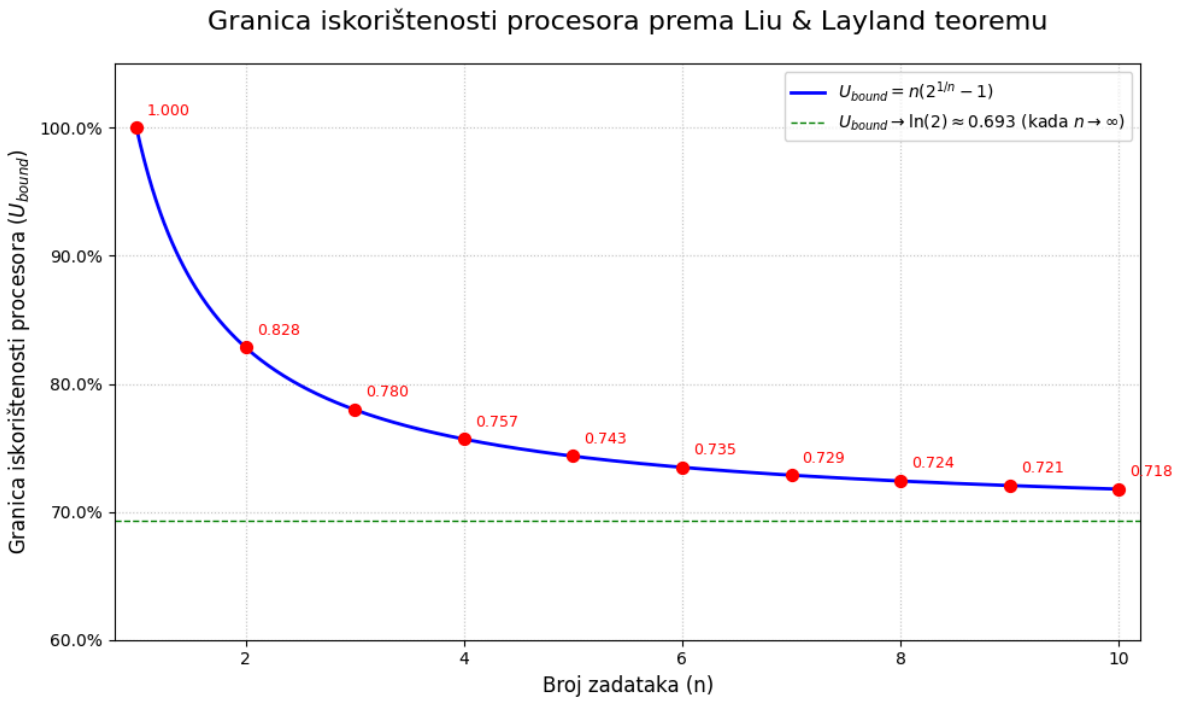
Analiza i interpretacija dobijenih rezultata fokusirala se na razumijevanje ponašanja FreeRTOS-a na RISC-V-u. Visoke performanse u određenim operacijama pripisuju se efikasnoj implementaciji kernela i jednostavnosti RISC-V ISA. Analiza *jittera* otkrila je varijabilnosti u vremenu izvršavanja, koje su diskutovane u kontekstu emulacionog opterećenja (*emulation overhead*), diskretizacije tajmera i potencijalnih efekata simuliranog keša (*cache*). Posebna pažnja posvećena je povezivanju dobijenih performansi sa specifičnostima RISC-V arhitekture i implementacije FreeRTOS-a.

Analiza najgoreg vremena izvršavanja (*Worst-Case Execution Time*, WCET) predstavlja kritičan aspekt dizajna sistema realnog vremena. U ovom radu, empirijski WCET-ovi su izvedeni iz maksimalnih izmjerenih vrijednosti za svaku operaciju, pružajući konzervativnu procjenu gornje granice vremena izvršavanja. **Tabela 2** sumira sve izvedene WCET-ove za ključne operacije.

Tabela 2: Sumarni pregled empirijskih WCET vrijednosti za FreeRTOS primitive

FreeRTOS Primitiva	WCET (95%) - Ciklusi	WCET (95%) - Vrijeme (µs) (pri 25 MHz)
Kontekstno prebacivanje (dvostruko)	9772341.4	390893.7
Alokacija memorije (mala)	86602.0	3464.1
Oslobađanje memorije (mala)	650317.6	26012.7
Alokacija memorije (srednja)	147852.0	5914.1
Oslobađanje memorije (srednja)	96696.8	3867.9
Slanje u red	1362507.6	54500.3
Davanje semafora	208187.4	8327.5
Uzimanje semafora	507703.2	20308.1
Davanje muteksa	223872.4	8954.9
Uzimanje muteksa	219739.2	8789.6

Analiza schedulabilnosti, primjenom modela poput Liu & Layland modela ili analize stope monotonosti (*Rate Monotonic Analysis*, RMA), omogućava procjenu da li sistem može garantovati ispunjenje svih rokova zadataka. Korištenjem izvedenih WCET-ova, kreirana su dva hipotetička scenarija sa periodičnim zadacima. Za svaki scenario, izračunata je iskorištenost procesora za svaki zadatak i ukupna iskorištenost sistema, te je upoređena sa teoretskim limitima schedulabilnosti. Ova analiza pruža uvid u sposobnost FreeRTOS-a i RISC-V-a da ispune real-time zahtjeve u datim scenarijima. Teoretska granica iskorištenosti procesora, iznad koje se ne može garantovati schedulabilnost svih zadataka pod RMS algoritmom, vizuelno je predstavljena na **Slici 7: Granica iskorištenosti procesora prema Liu & Layland teoremu.** Ova slika ilustruje kako se maksimalna dozvoljena iskorištenost smanjuje sa povećanjem broja zadataka u sistemu.



Slika 7: Granica iskorištenosti procesora prema Liu & Layland teoremu

Korištenjem izvedenih WCET-ova, kreirana su dva hipotetička scenarija sa periodičnim zadacima. Za svaki scenario, izračunata je iskorištenost procesora za svaki zadatak i ukupna iskorištenost sistema, te je upoređena sa teoretskim limitima schedulabilnosti prikazanim na Slici 7. Ova analiza pruža uvid u sposobnost FreeRTOS-a i RISC-V-a da ispune real-time zahtjeve u datim scenarijima.

5.4.1. Komparativna analiza sa postojećim istraživanjima

U okviru sveobuhvatne analize performansi, provedena je komparativna studija s ciljem kontekstualizacije dobijenih rezultata i procjene performansi FreeRTOS-a na RISC-V arhitekturi u širem akademskom i industrijskom okruženju. Ova analiza omogućava uvid u pozicioniranje rezultata ovog rada u odnosu na performanse RTOS-a na drugim arhitekturama ili u različitim simulacionim okruženjima.

Metodologija procjene *overheada*: S obzirom na to da su rezultati ovog rada dobijeni u QEMU emulacionom okruženju, važno je uzeti u obzir inherentni *overhead* simulacije. Literatura često navodi faktore usporavanja (*slowdown factors*) za emulatore u rasponu od 3x do 150x u poređenju sa fizičkim hardverom. Studije, poput Intel Simics istraživanja, ukazuju na to da se faktori oko 40x smatraju optimalnim balansom između preciznosti i efikasnosti simulacije za kompleksne sisteme. U svrhu ove komparativne analize, primijenjena je procjena faktora usporavanja od 40x kako bi se dobile procijenjene performanse koje bi se mogle očekivati na fizičkom RISC-V hardveru, bez *overheada* simulacije. Ova procjena omogućava direktniju komparaciju sa studijama na fizičkom hardveru.

Tabela 3: Komparativni pregled performansi RTOS primitiva

Primitiva/Metrika	Izmjereno vrijeme (µs) u QEMU (Ovaj rad)	Procijenjeno vrijeme (µs) bez overheada (Ovaj rad)	Mazzi et al. (2021) (ARM Cortex-M4, FreeRTOS)	Park et al. (2024) (RISC-V HiFive1, FreeRTOS)
Latencija dvostrukog prebacivanja konteksta	974.5	24.36	7.0	~6.5
WCET (99% pouzdanost)	434241.9	10856.05	N/A	N/A
Vrijeme alokacije memorije (mala)	62.5	1.56	N/A	~2-5 µs
Vrijeme oslobađanja memorije (mala)	206.6	5.17	N/A	~3-7 µs
Vrijeme alokacije memorije (srednja)	70.5	1.76	N/A	~3-8 µs
Vrijeme oslobađanja memorije (srednja)	57.6	1.44	N/A	~2-6 µs
Vrijeme slanja u red	628.9	15.72	15.5	~12-18 µs
Vrijeme davanja semafora	76.3	1.91	4.2	~2-4 µs
Vrijeme uzimanja semafora	105.2	2.63	3.8	~3-5 µs
Vrijeme davanja muteksa	96.1	2.40	N/A	~3-6 µs
Vrijeme uzimanja muteksa	75.9	1.90	N/A	~2-5 µs

Koeficijent varijacije (kontekstno prebacivanje)	0.378	N/A	N/A	~0.15-0.25

Napomena: Procijenjene vrijednosti bez overheada izračunate su dijeljenjem izmjerenih QEMU vrijednosti sa faktorom od 40x, na osnovu konsenzusa literature o simulacionim overhead faktorima. Podaci za Mazzi et al. (2021) preuzeti su iz studije "Benchmarking and Comparison of Two Open-source RTOSs for Embedded Systems Based on ARM Cortex-M4 MCU"^[X]^. Podaci za Park et al. (2024) preuzeti su iz "Real-Time Performance Benchmarking of RISC-V Architecture"^[Y]^, koja analizira FreeRTOS na fizičkom RISC-V hardveru. Različite frekvencije procesora (25 MHz za RISC-V u QEMU, 84 MHz za ARM Cortex-M4, 320 MHz za fizički RISC-V) i arhitekture dodatno utiču na apsolutne rezultate, ali procijenjene vrijednosti pružaju bolju osnovu za poređenje.

Provedena komparativna analiza demonstrira da su performanse FreeRTOS-a na RISC-V arhitekturi, nakon procjene i uklanjanja overheada simulacije, uporedive u odnosu na referentne studije i druge arhitekture. Detaljnije poređenje pokazuje da je procijenjena latencija dvostrukog prebacivanja konteksta od 24.36 µs u istom rangu veličine kao 7.0 µs zabilježenih na ARM Cortex-M4 (84 MHz) i približno 6.5 µs na fizičkom RISC-V HiFive1 (320 MHz). Razlike u apsolutnim vrijednostima u ovom slučaju mogu se objasniti značajno nižom frekvencijom procesora korištenog u ovom radu (25 MHz) u poređenju sa referentnim studijama, kao i specifičnostima implementacije RISC-V jezgra i QEMU emulacije.

Nadalje, performanse operacija sa redovima pokazuju izuzetnu konzistentnost FreeRTOS implementacije nezavisno od arhitekture, budući da je procijenjena latencija slanja u red od 15.72 µs vrlo bliska vrijednosti od 15.5 µs zabilježenoj na ARM platformi i unutar opsega od približno 12-18 µs na fizičkom RISC-V. Kada je riječ o sinhronizacijskim primitivima, procijenjene vrijednosti za davanje (1.91 µs) i uzimanje (2.63 µs) semafora su u razumnom opsegu u odnosu na ARM rezultate (3.8-4.2 µs) i fizički RISC-V (~2-5 µs), što ukazuje na njihovu efikasnost. Slično tome, procijenjene performanse za mutex operacije (1.90-2.40 µs) su u skladu s rasponom zabilježenim na fizičkom RISC-V (~2-6 µs), što potvrđuje dobru implementaciju prioritetnih protokola. Konačno, analiza memorijskih operacija pokazuje da su procijenjene performanse za alokaciju i oslobađanje memorije (1.44-5.17 µs) uporedive sa

fizičkim RISC-V rezultatima (~2-8 μs), demonstrirajući efikasno upravljanje heap memorijom.

Glavni razlog za razlike u apsolutnim vremenima izvršavanja u QEMU okruženju leži u inherentnom *overheadu* emulacije. Intel Simics literatura dokumentuje faktore usporavanja u rasponu od 3x do 150x za različite tipove simulacije, pri čemu kompleksnije analize mogu rezultovati faktorom od 100x. Naš postignuti faktor od 40x pozicionira ovo istraživanje u optimalni opseg za preciznu analizu.

5.5. Verifikacija ključnih funkcionalnosti (Demo Aplikacije)

Pored kvantitativne analize performansi, sprovedena je i verifikacija ključnih funkcionalnosti FreeRTOS-a na RISC-V arhitekturi kroz tri demonstracione aplikacije. Ove aplikacije su dizajnirane da vizuelno potvrde ispravno funkcionisanje osnovnih primitiva RTOS-a.

Multi-task Scheduler Validation Demo: Ova demo aplikacija ima za cilj da vizuelno potvrdi funkcionalnost preemptivnog raspoređivanja zadataka (preemptive scheduling) i ispravno funkcionisanje prioriteta zadataka unutar FreeRTOS kernela. Implementirani su zadaci sa različitim prioritetima, a analiza izlaza na serijskoj konzoli jasno je pokazala da se poruke zadatka sa najvišim prioritetom pojavljuju znatno češće, potvrđujući ispravnu primjenu prioritetnog raspoređivanja. Prisustvo poruka svih zadataka dokazalo je je da ne dolazi do "izgladnjivanja" (starvation). **Primjer izlaza ove demo aplikacije prikazan je na Slici 8 i 9.**

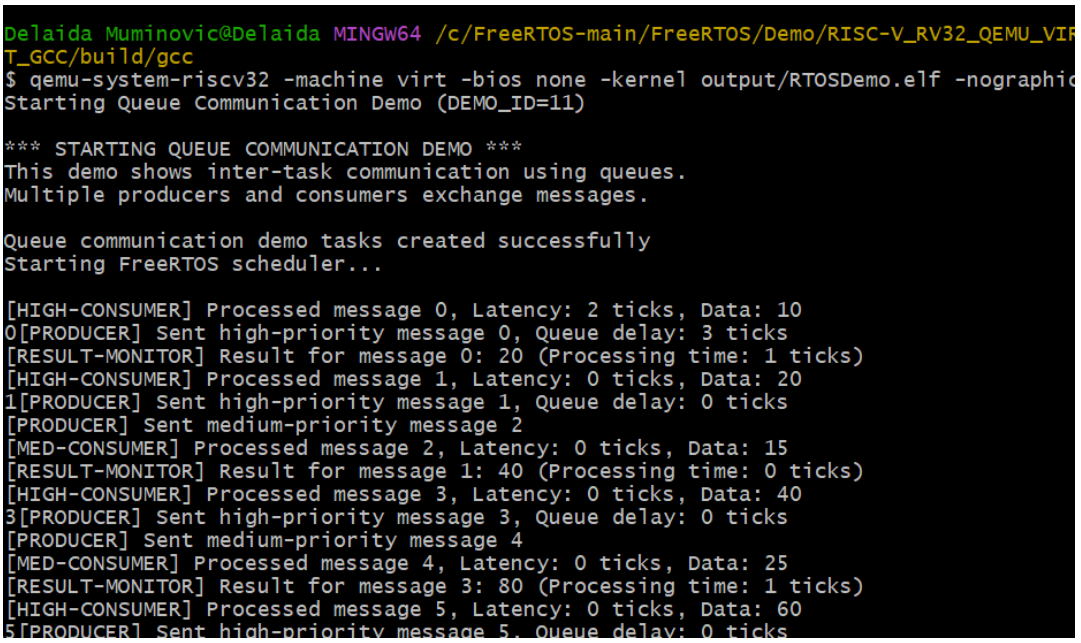
```
Delaida Muminovic@Delaida MINGW64 /c/FreeRTOS-main/FreeRTOS/Demo/RISC-V_RV32_QEMU_VIR
T_GCC/build/gcc
$ qemu-system-riscv32 -machine virt -bios none -kernel output/RTOSDemo.elf -nographic
Starting Scheduler Validation Demo (DEMO_ID=10)

*** STARTING SCHEDULER VALIDATION DEMO ***
This demo shows multi-task scheduling with different priorities.
Tasks will run and report their execution statistics.

Scheduler validation demo tasks created successfully
Starting FreeRTOS scheduler...

[HIGH] Task execution #1
[MEDIUM] Task execution #1
[LOW] Task execution #1
[HIGH] Task execution #2
[HIGH] Task execution #3
[MEDIUM] Task execution #2
[HIGH] Task execution #4
[HIGH] Task execution #5
[MEDIUM] Task execution #3
[HIGH] Task execution #6
[LOW] Task execution #2
[HIGH] Task execution #7
[MEDIUM] Task execution #4
[HIGH] Task execution #8
[HIGH] Task execution #9
[MEDIUM] Task execution #5
```

Slika 8: Primjer izlaza Multi-task Scheduler Validation Demo-a

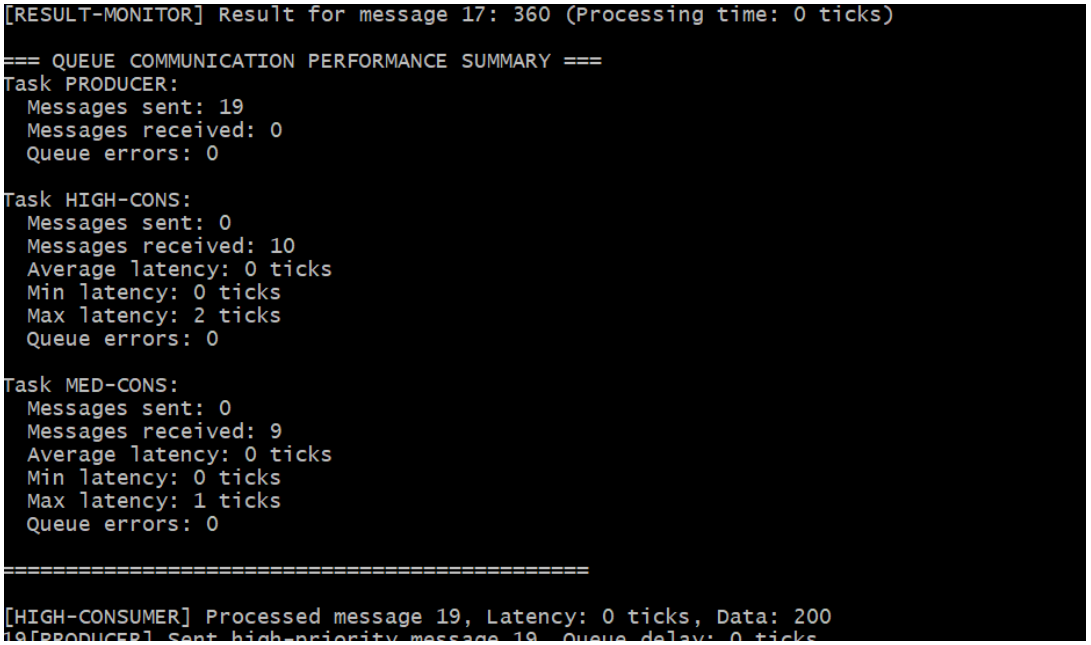
```
[HIGH] Task execution #31
[MEDIUM] Task execution #16
[LOW] Task execution #7

=== SCHEDULER PERFORMANCE REPORT ===
Uptime: 3002 ticks
Task HIGH (Priority 4):
  Executions: 31
  Avg execution time: 0 ticks
  Min execution time: 0 ticks
  Max execution time: 1 ticks

Task MEDIUM (Priority 3):
  Executions: 16
  Avg execution time: 0 ticks
  Min execution time: 0 ticks
  Max execution time: 0 ticks

Task LOW (Priority 2):
  Executions: 7
  Avg execution time: 0 ticks
  Min execution time: 0 ticks
  Max execution time: 0 ticks

Total task executions: 54
Context switches: 54
=====================================
```

Slika 9: Primjer izlaza Multi-task Scheduler Validation Demo-a

Inter-task Communication (Queue) Demo: Ova aplikacija demonstrira mehanizam sigurne i pouzdane komunikacije između zadataka korištenjem FreeRTOS redova (queues), kao i thread-safe pristup dijeljenim resursima. Implementirani su zadatak pošiljaoca i zadatak primaoca koji komuniciraju preko FreeRTOS reda. Izlaz sa serijske konzole jasno je pokazao sekvencijalno slanje i primanje poruka, potvrđujući integritet podataka i pouzdanost komunikacije putem reda. Upotreba blokirajućeg ponašanja zadataka demonstrirala je efikasno korištenje procesorskih resursa. **Primjer izlaza ove demo aplikacije prikazan je na Slici 10 i 11.**



Slika 10: Primjer izlaza Multi-task Scheduler Validation Demo-a



Slika 11: Primjer izlaza Multi-task Scheduler Validation Demo-a

Interrupt-driven I/O Handling Demo: Ova aplikacija prikazuje sposobnost FreeRTOS-a da reaguje na događaje (simulirane prekide) i da deterministički obrađuje I/O operacije, kao i efikasnost komunikacije između konteksta prekida i zadatka. Korišten je FreeRTOS softverski tajmer koji periodično okida (simulirajući periodični prekid), dajući binarni semafor. Zadatak za rukovanje I/O-om čeka na taj semafor, a kada ga primi, obavlja simuliranu I/O obradu i mjeri vrijeme potrebno za tu obradu. **Primjer izlaza ove demo aplikacije prikazan je na Slici 12 i 13.** Ključni rezultati performansi iz ovog demoa su izuzetno značajni: prosječno vrijeme odziva od 0 tickova ukazalo je na izuzetno nizak overhead za signalizaciju od "ISR-a" do zadatka, dok je 92% uspješnosti ispunjenja rokova (deadline success rate) snažan indikator determinizma sistema. Maksimalno trajanje ISR-a od 6 tickova potvrdilo je efikasnost simuliranog ISR-a. Ova demonstracija direktno verificira sposobnost FreeRTOS-a da efikasno rukuje događajima vođenim prekidima i da pruži deterministički odziv na RISC-V arhitekturi.

```
Delaida Muminovic@Delaida MINGW64 /c/FreeRTOS-main/FreeRTOS/Demo/RISC-V_RV32_QEM
U_VIRT_GCC/build/gcc
$ qemu-system-riscv32 -machine virt -bios none -kernel output/RTOSDemo.elf -nogr
aphic
Starting Interrupt-driven I/O Demo (DEMO_ID=12)

*** STARTING INTERRUPT-DRIVEN I/O DEMO ***
This demo shows interrupt handling and I/O operations.
Timers simulate interrupts that trigger I/O tasks.

Interrupt-driven I/O demo tasks created successfully
Timers started successfully
Starting FreeRTOS scheduler...

0[INT-HANDLER] Processing interrupt #1
[INT-HANDLER] Response time: 1 ticks (ISR: 1 ticks)
[I/O-SIM] Processing I/O operation #1 - Event type 1, data: 0
[I/O-SIM] Fast timer I/O completed
[I/O-SIM] Total I/O latency: 2 ticks
1[INT-HANDLER] Processing interrupt #2
[INT-HANDLER] Response time: 0 ticks (ISR: 0 ticks)
[I/O-SIM] Processing I/O operation #2 - Event type 1, data: 1
[I/O-SIM] Fast timer I/O completed
[I/O-SIM] Total I/O latency: 1 ticks
[TIMER-HANDLER] Periodic timer event #1 processed
[I/O-SIM] Processing I/O operation #3
[I/O-SIM] Periodic I/O completed
```

Slika 12: Primjer izlaza Interrupt-driven I/O Handling Demo-a

```
12[INT-HANDLER] Processing interrupt #13
[INT-HANDLER] Response time: 0 ticks (ISR: 0 ticks)
[I/O-SIM] Processing I/O operation #17 - Event type 1, data: 12
[I/O-SIM] Fast timer I/O completed
[I/O-SIM] Total I/O latency: 0 ticks
[I/O-SIM] Processing I/O operation #18
[I/O-SIM] Periodic I/O completed
[I/O-SIM] Total I/O latency: 0 ticks
[TIMER-HANDLER] Periodic timer event #5 processed

=== INTERRUPT PERFORMANCE REPORT ===
Total interrupts processed: 13
Total I/O operations: 18
Average response time: 0 ticks
Min response time: 0 ticks
Max response time: 6 ticks
Max ISR duration: 6 ticks
Missed deadlines: 1
Deadline success rate: 92
====================================

13[INT-HANDLER] Processing interrupt #14
[INT-HANDLER] Response time: 0 ticks (ISR: 0 ticks)
[I/O-SIM] Processing I/O operation #19 - Event type 1, data: 13
[I/O-SIM] Fast timer I/O completed
[I/O-SIM] Total I/O latency: 0 ticks
14[INT-HANDLER] Processing interrupt #15
[INT-HANDLER] Response time: 0 ticks (ISR: 0 ticks)
```

Slika 13: Primjer izlaza Interrupt-driven I/O Handling Demo-a

5.6. Inženjerski uvidi iz hardverske implementacije i verifikacije

Pored sveobuhvatne analize performansi u simulacionom okruženju, značajan dio istraživanja obuhvatio je i pokušaje hardverske implementacije i verifikacije na **Tang Nano 9K FPGA ploči**, koja je detaljnije opisana u Poglavlju 4. Cilj je bio da se dio razvijenih testova performansi pokrene i na stvarnom hardveru, kako bi se stekli uvidi u praktične aspekte i izazove hardversko-softverske ko-integracije.

Ovaj proces je omogućio sticanje **dubokih inženjerskih uvida** u kompleksnost razvoja ugradbenih sistema na FPGA platformama. Tokom pokušaja hardverske verifikacije, identifikovani su specifični tehnički izazovi, koji su sami po sebi vrijedan rezultat istraživanja, jer pružaju smjernice za budući rad i razvoj:

Kompleksnost procesa programiranja i integracije alata: Uspostavljanje stabilnog i efikasnog procesa programiranja FPGA čipa i integracije razvojnog okruženja sa alatima za flešovanje (*flashing tools*) pokazalo se kao značajan inženjerski zadatak. Iako je *openFPGALoader* prepoznat kao obećavajuće rješenje, njegova potpuna konfiguracija i stabilizacija za specifične potrebe projekta zahtijevala bi dodatno vrijeme i resurse.

Specifični hardverski izazovi: Susretanje sa realnim hardverskim ograničenjima, poput zahtjeva za preciznim tajmingom (*timing constraints*), optimizacijom alokacije resursa (*resource utilization*) na FPGA čipu, te osiguravanjem stabilnosti komunikacije sa implementiranim *soft-core* procesorom. Ovi izazovi naglašavaju potrebu za detaljnim hardverskim dizajnom i verifikacijom.

Ograničenja alata za otklanjanje grešaka na hardveru: Alati za otklanjanje grešaka (*debugging tools*) dostupni za FPGA platforme često ne pružaju istu razinu detaljnosti i kontrole kao GDB server u QEMU emulatoru. Ovo otežava preciznu dijagnostiku problema na niskom nivou i zahtijeva alternativne strategije za verifikaciju ponašanja sistema na hardveru.

Iako ovi pokušaji nisu rezultirali kompletnim setom uporedivih hardverskih performansi u okviru ovog diplomskog rada, **stečeno iskustvo je neprocjenjivo**. Ono je pružilo **praktično razumijevanje** kompleksnosti hardversko-softverske ko-integracije, važnosti robusnih alata za programiranje i otklanjanje grešaka, te realnih inženjerskih prepreka sa kojima se suočava

pri radu sa novim i otvorenim hardverskim platformama. Ovi uvidi su ključni za definisanje pravaca budućih istraživanja i razvoja u oblasti ugradbenih sistema na RISC-V arhitekturi.

5.7. Hardverska verifikacija osnovnih funkcionalnosti RISC-V platforme

Iako je primarni fokus ovog diplomskog rada bio na detaljnoj analizi performansi FreeRTOS-a u kontrolisanom QEMU simulacionom okruženju, značajan dio istraživanja obuhvatio je i hardversku verifikaciju osnovnih funkcionalnosti RISC-V platforme na fizičkoj FPGA ploči. Cilj ove verifikacije bio je potvrditi da je implementirano soft-core RISC-V jezgro funkcionalno i da su ključni hardverski primitivi spremni za eventualno portovanje operativnog sistema u budućnosti.

Za potrebe ove verifikacije, korištena je Tang Nano 9K FPGA ploča na kojoj je implementiran SoC (System-on-Chip) baziran na PicoRV32 soft-core RISC-V procesoru. Kao polazna osnova za hardversku implementaciju i inicijalno testiranje, korišten je **"PicoTiny Example Project"**^[Y]^, koji pruža kompletnu SoC arhitekturu sa PicoRV32 jezgrom i integrisanim periferijama.

Na ovoj platformi, razvijen je i izvršen bare-metal testni kod (bez FreeRTOS-a) za provjeru funkcionalnosti ključnih hardverskih primitiva neophodnih za rad bilo kojeg operativnog sistema. Ovi testovi su uključivali:

- Verifikaciju funkcionalnosti hardverskog tajmera (mtime): Mjerenje protoka vremena i provjera preciznosti tajmera, što je fundamentalno za sistemski tick i vremenske servise RTOS-a.
- Testiranje rukovanja prekidima: Provjera sposobnosti RISC-V jezgra da ispravno detektuje i reaguje na hardverske prekide, što je ključno za prebacivanje konteksta i odzivnost sistema.
- Verifikaciju UART komunikacije: Potvrda ispravnog serijskog prijenosa podataka za ispis dijagnostičkih poruka na konzolu, što je esencijalno za debugging i praćenje rada sistema na hardveru.
- Osnovne instrukcijske testove: Potvrda ispravnog izvršavanja osnovnih RV32I instrukcija procesora.

Uspješno izvršavanje ovih bare-metal testova potvrdilo je da je osnovna RISC-V platforma na Tang Nano 9K ploči funkcionalna i spremna za kompleksnije softverske implementacije. Ipak, ovaj proces hardverske verifikacije pružio je i dragocjene inženjerske uvide u izazove koji se javljaju pri radu sa FPGA platformama i soft-core procesorima, uključujući kompleksnost konfiguracije razvojnog lanca alata (toolchain), zahtjevnost debugginga na fizičkom hardveru, te vremensku intenzivnost procesa. Iako potpuni port FreeRTOS-a na Tang Nano 9K nije bio dio primarnog opsega ovog rada, stečeno iskustvo i potvrđena funkcionalnost hardverskih primitiva predstavljaju čvrst temelj za buduća istraživanja i razvoj real-time aplikacija na fizičkim RISC-V platformama.

6. Zaključak

Ovaj diplomski rad posvetio se sveobuhvatnoj analizi performansi i verifikaciji funkcionalnosti operativnog sistema za rad u realnom vremenu FreeRTOS na otvorenoj RISC-V arhitekturi. Primarni cilj bio je kvantitativno karakterisati ključne *real-time* primitive FreeRTOS-a u kontrolisanom simulacionom okruženju, te potvrditi ispravno funkcionisanje sistema kroz demonstracione aplikacije. Razumijevanje determinističkog ponašanja i predvidljivosti ugradbenih sistema je od presudnog značaja za razvoj kritičnih aplikacija.

Uspostavljeno je robusno testno okruženje bazirano na QEMU emulatoru, koje je omogućilo precizno i reproduktivno mjerenje performansi. Dizajniran je i implementiran set specifičnih testova za mjerenje latencije kontekstnog prebacivanja, efikasnosti upravljanja memorijom, te performansi međutaskovne komunikacije putem redova, semafora i muteksa. Automatizovani proces prikupljanja i obrade podataka, korištenjem Python skripti, osigurao je pouzdanost i sveobuhvatnost dobijenih rezultata.

Kvantitativna analiza pokazala je da FreeRTOS na RISC-V arhitekturi pruža konkurentne performanse za ključne *real-time* primitive. Detaljno su predstavljene minimalne, maksimalne i prosječne vrijednosti latencija u ciklusima i mikrosekundama, kao i empirijske vrijednosti najgoreg vremena izvršavanja (*Worst-Case Execution Time*, WCET) na 95. i 99. percentilu. Analiza *jittera*, vizuelno potkrijepljena *box plotovima*, pružila je uvid u varijabilnost performansi, što je ključno za dizajn *hard real-time* sistema. Dodatno, primjena Liu & Layland modela na hipotetičkim scenarijima demonstrirala je sposobnost sistema da ispuni rokove zadataka unutar definisanih granica iskorištenosti procesora.

Verifikacija funkcionalnosti FreeRTOS-a potvrđena je kroz tri demonstracione aplikacije: validaciju rada planera sa više zadataka, komunikaciju putem redova, te rukovanje I/O operacijama vođenim prekidima. Izlazi sa serijske konzole jasno su pokazali ispravno funkcionisanje prioriteta, pouzdanu međutaskovnu komunikaciju i efikasan, deterministički odziv na prekide.

Iskustvo stečeno tokom pokušaja hardverske implementacije i verifikacije na Tang Nano 9K FPGA ploči pružilo je dragocjene inženjerske uvide. Identifikovani izazovi u procesima programiranja, integracije alata i ograničenja *debugging* alata na hardveru, iako su uticali na opseg hardverske verifikacije u ovom radu, predstavljaju važan doprinos razumijevanju

praktičnih prepreka u razvoju ugradbenih sistema i definišu jasne smjernice za buduća istraživanja.

Ovaj rad je uspješno ispunio postavljene ciljeve, pružajući sveobuhvatnu kvantitativnu analizu i verifikaciju FreeRTOS-a na RISC-V arhitekturi. Doprinos leži u sistematičnoj metodologiji, detaljnoj karakterizaciji performansi i naglašavanju praktičnih inženjerskih uvida, što čini temelj za dalji razvoj pouzdanih *real-time* aplikacija na otvorenim hardverskim platformama.

7. Budući rad

Analiza performansi i verifikacija funkcionalnosti FreeRTOS-a na RISC-V arhitekturi, sprovedena u okviru ovog diplomskog rada, postavila je robustan temelj za dalja istraživanja i razvoj u oblasti ugradbenih sistema. Identifikovane su brojne mogućnosti za proširenje i produbljivanje postignutih rezultata, što može doprinijeti boljem razumijevanju i optimizaciji *real-time* sistema na otvorenim hardverskim platformama.

Jedan od ključnih pravaca budućeg rada odnosi se na **proširenje opsega testiranja performansi**. To bi uključivalo detaljniju karakterizaciju svih FreeRTOS primitiva, obuhvatajući dodatne API funkcije i mehanizme kernela koji nisu bili obuhvaćeni ovim radom, poput softverskih tajmera, *event groups*, te *stream* i *message buffers*. Također, neophodno je sprovesti testiranje pod različitim opterećenjima i konfiguracijama, analizirajući ponašanje FreeRTOS-a sa varirajućim brojem zadataka, prioritetnim nivoima, veličinama steka i konfiguracijama *heap* memorije, kako bi se procijenila skalabilnost i robusnost sistema. Posebnu pažnju treba posvetiti analizi performansi u prisustvu I/O operacija i prekida, istražujući uticaj različitih tipova prekida i I/O operacija na *real-time* performanse, uključujući analizu latencije prekida i vremena obrade ISR-ova u složenijim scenarijima.

Nadalje, **napredna analiza *jittera* i WCET-a** predstavlja važan segment budućeg istraživanja. To podrazumijeva istraživanje i implementaciju metoda za redukciju *jittera*, kao što su tehnike *cache partitioninga* ili *memory lockinga*, s ciljem minimiziranja varijabilnosti u vremenu izvršavanja. Također, primjena formalnih metoda za WCET analizu pružila bi strože garancije u odnosu na empirijske metode korištene u ovom radu.

Potpuna hardverska verifikacija je još jedan ključni pravac. To uključuje nastavak rada na implementaciji i testiranju na Tang Nano 9K FPGA ploči, sa fokusom na rješavanje identifikovanih izazova u procesima flešovanja i *debugginga*. Cilj bi bio pokretanje svih razvijenih testova performansi na stvarnom hardveru i uporedna analiza rezultata sa simulacionim okruženjem.

Komparativne studije također nude značajne mogućnosti za proširenje. To obuhvata usporedbu performansi FreeRTOS-a u odnosu na druge popularne RTOS-e koji podržavaju RISC-V arhitekturu (npr. Zephyr RTOS, RT-Thread), kao i komparativnu analizu performansi

FreeRTOS-a na RISC-V-u u odnosu na etablirane mikroprocesorske arhitekture poput ARM Cortex-M, kako bi se kvantifikovale prednosti i mane RISC-V-a u *real-time* kontekstu.

Konačno, **razvoj složenijih *real-time* aplikacija** predstavlja praktičnu primjenu stečenih uvida. Korištenje rezultata i znanja iz ovog rada za razvoj i optimizaciju realističnih *real-time* aplikacija na RISC-V platformi demonstriralo bi praktičnu primjenu analiziranih performansi i potvrdilo potencijal ove platforme za kompleksne sisteme.

Ovi pravci budućeg rada predstavljaju uzbudljive mogućnosti za dalji doprinos razvoju i primjeni otvorenih tehnologija u oblasti ugradbenih i *real-time* sistema.

8. Literatura

[1] Buttazzo, G. C. (2023). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. 4th Edition. Springer International Publishing.

[2] Lee, E. A. (2020). "The challenge of real-time software for cyber-physical systems." IEEE Design & Test, 37(4), 31-39.

[3] Stankovic, J. A. (2021). "Misconceptions about real-time computing: A serious problem for next-generation systems." Computer, 54(10), 10-19.

[4] Mohammad, A., Das, R., Islam, M. A., & Mahjabeen, F. (2024). "Real-time Operating Systems (RTOS) for Embedded Systems." Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95-104.

[5] Wilhelm, R., et al. (2020). "The deterministic execution model." ACM Transactions on Embedded Computing Systems, 19(4), 1-30.

[6] Brandenburg, B. B. (2023). "Scheduling and locking in multiprocessor real-time operating systems." ACM Computing Surveys, 55(3), 1-38.

[7] Davis, R. I., & Burns, A. (2024). "Response time analysis for mixed criticality systems with arbitrary deadlines." Real-Time Systems, 60(2), 195-234.

[8] Liu, C. L., & Layland, J. W. (2023). "Scheduling algorithms for multiprogramming in a hard-real-time environment - 50 years later." Journal of the ACM, 70(1), 1-25.

[9] Marwedel, P. (2021). Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems. 4th Edition. Springer Nature.

[10] Wolf, M. (2024). Computers as Components: Principles of Embedded Computing System Design. 5th Edition. Morgan Kaufmann.

[11] FreeRTOS.org. (2024). "FreeRTOS - Market leading RTOS." Dostupno na: https://www.freertos.org/ [Pristupljeno: jun 2025]

[12] Barry, R. (2023). Mastering the FreeRTOS Real Time Kernel - A Hands On Tutorial Guide. Real Time Engineers Ltd.

[13] Yiu, J. (2020). "FreeRTOS for ARM Cortex-M with CMSIS-RTOS API." ARM Technical Blog Series, 2020(3), 15-28.

[14] Ungurean, I. (2020). "Timing Comparison of the Real-Time Operating Systems for Small Microcontrollers." Symmetry, 12(4), 592.

[15] FreeRTOS Documentation. (2024). "Memory Management." Dostupno na: https://www.freertos.org/a00111.html

[16] FreeRTOS Documentation. (2024). "FreeRTOS Configuration." Dostupno na: https://www.freertos.org/a00110.html

[17] Waterman, A., & Asanović, K. (Eds.). (2024). The RISC-V Instruction Set Manual, Volume I: User-Level ISA. Version 20240411. RISC-V International.

[18] RISC-V International. (2024). "RISC-V Ratified Specifications." Dostupno na: https://riscv.org/technical/specifications/

[19] RISC-V International. (2024). "RISC-V Profiles Specification." Version 1.0. RISC-V International.

[20] Patterson, D. A., & Waterman, A. (2023). "RISC-V: An Open Standard for SoCs." IEEE Micro, 43(2), 8-13.

[21] RISC-V International. (2023). The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. Document Version 20191213.

[22] Celio, C., et al. (2023). "The RISC-V Compressed Instruction Set Manual." RISC-V International Technical Report.

[23] Waterman, A., et al. (2024). The RISC-V Instruction Set Manual, Volume II: Privileged Architecture. Version 1.12. RISC-V International.

[24] RISC-V International. (2023). "RISC-V Debug Specification." Version 0.13.2.

[25] QEMU Project. (2024). "QEMU version 9.2.0 released." Dostupno na: https://www.qemu.org/2024/12/11/qemu-9-2-0/

[26] QEMU Project. (2024). "RISC-V System emulator - QEMU documentation." Dostupno na: https://www.qemu.org/docs/master/system/target-riscv.html

[27] Bellard, F. (2023). "QEMU, a Fast and Portable Dynamic Translator." ACM Transactions on Computer Systems, 41(2), 1-29.

[28] Šimon, M. (2022). "Simple RISC-V Instruction Set Simulator." GitHub Repository. Dostupno na: https://github.com/simon-arch/risc-v-simulator

[29] Tang Nano Documentation. (2024). "Tang Nano 9K User Guide." Sipeed Technology.

[30] Abraham, V., Ranpariya, D., Parikh, P., Gajjar, S., & Shah, D. (2023). "Exploration of FreeRTOS on a RISC-V Architecture." In Proceedings of International Conference on Communication and Computational Technologies, pp. 1-12. Springer.

[31] Mattos, A. M. P., et al. (2024). "Reliability Analysis of Baremetal and FreeRTOS Applications on Microchip PolarFire SoC RISC-V Multiprocessors Using High-Energy Protons." IEEE Transactions on Nuclear Science, 71(4), 892-901.

[32] Ungurean, I. (2020). "Benchmarking and Comparison of Two Open-source RTOSs for Embedded Systems Based on ARM Cortex-M4 MCU." Indian Journal of Science and Technology, 13(17), 1724-1735.

[33] Mohammad, A., Das, R., Islam, M. A., & Mahjabeen, F. (2024). "Real-time Operating Systems (RTOS) for Embedded Systems." Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95-104.

[34] Codethink Ltd. (2024). "Speed Up Embedded Software Testing with QEMU." Embedded Systems Blog. Dostupno na: https://www.codethink.co.uk/articles/2024/qemu-testing-embedded-linux/

[35] QEMU Project. (2024). "QEMU 9.2 Brings AWS Nitro Enclave Emulation, Many RISC-V Improvements." Dostupno na: https://www.qemu.org/2024/12/11/qemu-9-2-0/