Зборник радова Факултета техничких наука, Нови Сад

UDK: 621.3
DOI: https://doi.org/10.24867/33BE36Nesic

РАЗВОЈ СОФТВЕРА НА *RISC-V* АРХИТЕКТУРИ СА ФОКУСОМ НА *RPC* ИМПЛЕМЕНТАЦИЈУ

RISC-V SOFTWARE DEVELOPMENT WITH FOCUS ON RPC IMPLEMENTATION

Марија Нешић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Овај рад представља детаљан опис софтверске архитектуре и имплементације механизма удаљених позива процедура (Remote Procedure Call – RPC) у оквиру уграђеног система заснованог на RISC-V архитектури. Рад документује комплетан процес подизања софтвера на новом чипу, од првобитне иницијализације хардвера до успостављања функционалне комуникације са спољашњим системима.*

Кључне речи: *Развој софтверског окружења, RISC-V, FreeRTOS, RPC*

Abstract – *This thesis presents a detailed description of the software architecture and implementation of the Remote Procedure Call (RPC) mechanism within an embedded system based on the RISC-V architecture. It documents the complete process of bringing up software on a new chip, from the initial hardware initialization to establishing functional communication with external systems.*

Keywords: *Software Bring-Up, RISC-V, FreeRTOS, RPC*

1. УВОД

Идеја удаљеног позива процедуре (*RPC*), коју су први пут увели *Birrell* и *Nelson* 1984. године, представљала је велики корак напред у дистрибуираној обради. Она омогућава да се процедуре у процесима на удаљеним рачунарима позивају као да су процедуре у локалном адресном простору. *RPC* систем управља основним механизмима — кодирањем и декодирањем података, слањем порука и обезбеђивањем да се позив понаша као регуларан позив функције [1].

RPC модел комуникације изабран је као основни механизам интеракције између *host*-а и *Wi-Fi HaLow* чипа заснованог на *SweRV EH1* архитектури.

Основна мотивација за овај избор лежи у потреби да се сложене операције са стране *host*-а иницирају једноставним функцијским позивима, а да се при томе изолују детаљи *low-level* комуникације, синхронизације и управљања меморијом.Имплементирани *RPC* механизам припада категорији *Hardware-Level RPC* (*Chip-to-Chip*, *On-Board*), конкретно *SPI/I2C RPC* типу. Иако чип интерно користи *AXI* протокол, класификација се одређује према спољашњем интерфејсу и комуникационим карактеристикама. Из перспективе спољашњег *host*-та, комуникација се реализује кроз стандардне *SPI* трансакције где *host* (*master*) шаље команде које функционишу као удаљени позиви процедура ка *Wi-Fi HaLow* чипу (*slave*). Што се тиче комуникационог модела нуди комбинацију синхроне и асинхроне комуникације.

Циљ рада је приказ конкретног решења развијеног у индустријском контексту за *Wi-Fi HaLow* чип, које омогућава комуникацију између чипа и спољашњег *host* уређаја.

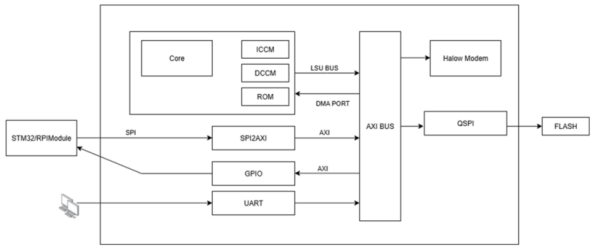
Главни проблеми решавани у оквиру овог пројекта обухватају иницијализацију и *bring-up* софтвера на *SweRV EH1* процесору заснованом на *RISC-V* архитектури, портовање оперативног система *FreeRTOS* на циљну платформу, развој *RPC* механизма за транспарентну комуникацију између уграђеног чипа и екстерног *host*-а, имплементацију *SPI* комуникационог слоја са синхронизацијом и управљањем прекидима, развој *host* стране *RPC* комуникације, као и оптимизацију ресурса и меморијских захтева система.

2. СОФТВЕРСКА И ХАРДВЕРСКА ПЛАТФОРМА

Цео систем се састоји из два процесора, од којих је један *HaLow modem* процесор који нуди *Wi-Fi* функционалности. Да би та функционалност остала изолована, остале функционалности су део интерног *host* процесора, *SweRV EH1*. У овом раду је описан његов развој. Чип представља специјализовану *Wi-Fi HaLow* станицу која прима команде од спољашњих *host*-ова преко *SPI* интерфејса.

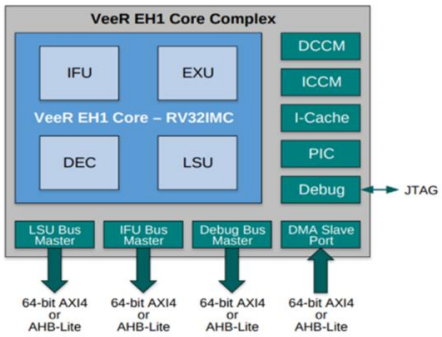
НАПОМЕНА:

Овај рад проистекао је из мастер рада чији ментор је био др Предраг Теодоровић, ванр. проф.



Слика 1. *Дијаграм система*

На слици 1 је приказан цео систем заједно са процесором.



Слика 2. *Архитектура SweRV EH1 Core Complex-a*

Као што је приказано на слици Слика 2, *SweRV EH1 Core Complex* садржи *RV32IMC* језгро са пратећим компонентама као што су *DCCM*, *ICCM*, *I-Cache*, *PIC* и *Debug* интерфејс, као и више *AXI4*/*AHB-Lite* интерфејса за спољашње конекције.

Систем осим интерног *host* и *HaLow modem* процесора садржи још и *SPI-to-AXI*, *UART* и *QSPI* модуле. Комуникација од екстерног *host-a* до чипа се одвија преко *SPI-to-AXI* модула који омогућава комуникацију тако што *SPI* сигнали од спољашњег контролера (нпр. *STM32*) бивају конвертовани у *AXI* протокол, прослеђени на *AXI* магистралу, а затим преко *DMA* порта приступају *DCCM* меморији за размену података. Комуникација од чипа ка екстерном *host-y* функционише у супротном смеру - сигнал који процесор емитује преко *LSU BUS-a* се рутира кроз *AXI* магистралу и појављује се на *GPIO* портовима за даље прослеђивање екстерном *host-y*.

FreeRTOS је изабран као оперативни систем за *Wi-Fi HaLow* чип, што представља логичан избор за ову врсту уграђених апликација. У питању је оперативни систем у реалном времену отвореног кода, дизајниран посебно за микроконтролере и уграђене процесоре са ограниченим хардверским ресурсима.

Избор шеме *heap_4* заснован је на њеном идеалном односу између детерминисаног понашања, отпорности на фрагментацију и компатибилности са *RISC-V (SweRV EH1)* окружењем.

Подешавање окружења укључује инсталацију *RISC-V GNU Toolchain*-а. *RISC-V GNU Toolchain* представља комплетно окружење за развој, које обухвата *GCC* компајлер, *Binutils*, *GDB debugger* и *Newlib C* стандардну библиотеку [2]. За подршку *SweRV EH1* језгру, алат је конфигурисан са *rv32imc* архитектуром и *ilp32 ABI*-јем.

Приликом успешног компајлирања генеришу се следећи изворни фајлови *firmware.elf*, *firmware.disasm* и *firmware.map*. За дебаговање кода коришћен је *J-Link* адаптер преко *JTAG* интерфејса.

3. ПОРТОВАЊЕ *FREERTOS*-А НА *SWERV EH1*

Портовање оперативног система *FreeRTOS* на процесор *SweRV EH1* архитектуре *RISC-V* обухвата адаптацију његових основних механизама, као што су замена контекста, руковање прекидима и конфигурација системског тајмера, у складу са специфичностима циљне хардверске платформе.

Портовање *FreeRTOS*-а на *SweRV EH1* почиње укључивањем изворних фајлова који садрже основне *RTOS* функционалности и специфичног *RISC-V* адаптационог слоја. Затим се конфигурише *trap handler*, иницијализује се систем прекида и ради се интеграција екстерних прекида.

FreeRTOS кернел се ослања на периодичне прекиде тајмера (*RTOS tick*) за пребацивање контекста између *task*-ова, праћење временских интервала и управљање *timeout* механизмима.

У функцији *vPortSetupTimerInterrupt()* се врши конфигурација тајмера. У портовању *FreeRTOS*-а за *SweRV EH1* језгро, *vPortSetupTimerInterrupt()* користи *Low Power Controller* (*LPC*) као извор системског тајмера. *LPC* има сопствени интерни тајмер који је у овом случају конфигурисан да активира прекид на сваких 1000 микросекунди.

Такође се у *linker* скрипти дефинише *IRQ stack* који се користи за управљање прекидима.

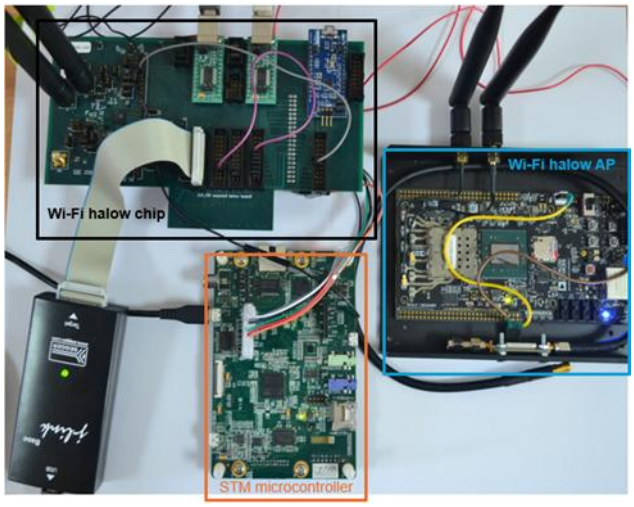
Након успешно извршене иницијализације система, конфигурације прекидне логике и покретања *scheduler*-а, извршена је провера рада *FreeRTOS* окружења кроз креирање и извршавање више *task*-ова са различитим приоритетима и функционалностима. Главна функција апликације иницијализује основни апликативни *task* и покреће *task* за комуникацију, након чега се позива *vTaskStartScheduler()*, чиме се управљање системом предаје *RTOS*-у.

4. *RPC* ДИЗАЈН

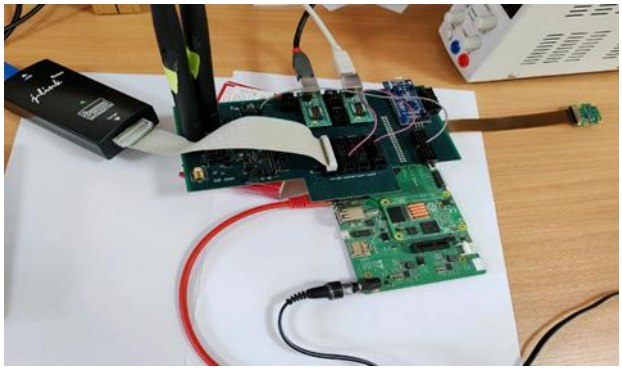
Дијаграм система који видимо на слици 1 је *Wi-Fi HaLow* са слике 3. Он преставља *Wi-Fi HaLow* станицу. Затим је направљена подршка да се станица преко *SPI* протокола може користити преко *STM* микроконтролера или преко *RPi Compute* модула. Слика 3 приказује чип повезан са *STM* микроконтролером. *Wi-Fi HaLow* приступна тачка је постављена поред њих. Слика 4 приказује чип повезан са *RPi Compute Module 4 IO* плочом.

Систем користи посвећен меморијски регион који служи као дељена меморија између *host* процесора и локалног микроконтролера. Кључни аспект система је хардверска транслација адреса. Екстерни *host* увек пише на адресе почевши од 0x00000000 из своје перспективе, али *SPI slave* контролер аутоматски транслира те адресе у стварне физичке адресе локалне меморије. Ово омогућава *host* процесору да види

дељену меморију као континуиран блок почевши од нуле, док се заправо приступа било ком меморијском региону микроконтролера.

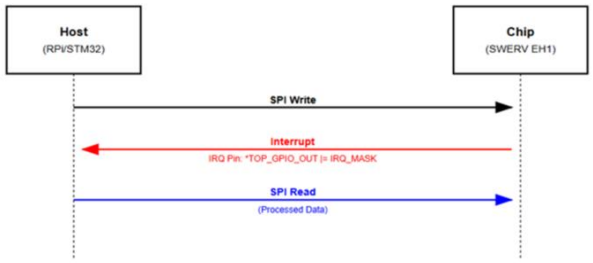


Слика 3. *Конфигурација са STM микроконтролером*



Слика 4. *Конфигурација са Raspberry Pi Compute Module*

На слици 5 приказан је поједностављен *RPC* механизам система.



Слика 5. *Поједностављен дијаграм RPC механизма*

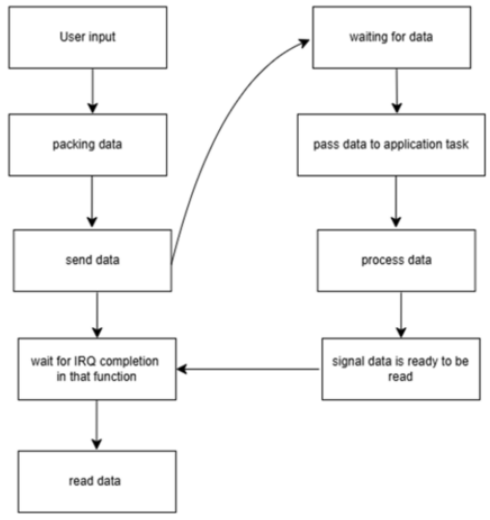
Комуникациони протокол користи флексибилан механизам паковања података. На слици 6 се види формат пакета.

Слика 6. *Формат пакета*

При упису података од стране *host* процесора аутоматски се генерише прекид који обавештава локални систем о пристиглим подацима.

По пријему прекида, систем врши анализу садржаја *RX buffer*-а, идентификује врсту захтева и позива одговарајући механизам обраде. Након обраде, резултати се уписују у *TX buffer* или *async TX buffer*. Затим се у *interrupt_status* структури бележи да је резултат доступан за читање и сетује се бит *GPIO* регистра да сигнализира да је податак спреман за читање.

Да би комуникација између екстерног *host*-а и *RISC-V* чипа била поуздана и ефикасна, неопходан је механизам синхронизације који обе стране информише када су подаци спремни за пријем или обраду. Овај систем користи комбинацију хардверског *SPI* прекида, *FreeRTOS* нотификација, *GPIO* сигнала и семафора на страни *host*-а.



Слика 7. *Дијаграм тока података*

RPC систем на чипу реализован је кроз два *FreeRTOS task*-а:

- *RPC task*, који је одговоран за пријем података преко *SPI* интерфејса, парсирање улазне поруке и прослеђивање података ка апликационом слоју
- *APP task*, који је посвећен извршавању логике на основу примљене команде

Овакво раздвајање одговорности осигурава да је *RPC task* увек спреман да реагује на нови прекид. Након што прими податке и упакује их у структуру *data_packet_t*, он их прослеђује *APP task-y* преко *message queue*-а. Обрада команде се врши независно, омогућавајући паралелну комуникацију и извршавање.

На страни екстерног *host*-а такође постоје 2 *task*-а. *CLI* интерфејс омогућава корисницима да иницирају *RPC* позиве преко командне линије. Ове функције шаљу податке ка чипу и затим чекају резултат. У исто време други *task* чека одговор од стране чипа. Функције се након слања блокирају семафором док не стигне одговор након чега се исти прочита. Свака функција има свој семафор.

5. РАЗВОЈ НА *HOST* СТРАНИ

Први део развоја система обухватао је креирање *SPI Linux* драјвера. Ово је омогућило директну и ефикасну контролу над *SPI* комуникацијом. *RPC* систем је први пут имплементиран на *Raspberry Pi Compute Module 4 IO Board-y*, због подршке за *spidev* интерфејс у *Linux* окружењу, што је значајно поједноставило имплементацију прототипа.

Због *AXI* протокола, адресе на које се шаљу или читају подаци морају бити умножак 4. А због тога како је имплементиран *SPI*, број података који се шаље мора бити дељив са 4, због тога се додаје *padding*. У *cmd_parser* модулу може се видети да је потребно потпуно 32-битно писање за све трансакције, као и да се строго захтева поравнање адреса на 4 бајта [3,4].

Као сигнал за завршетак трансцакције имплеметиран је *interrupt* механизам на *Raspberry Pi* страни, где сетовање бита *GPIO* регистра означава крај обраде података.

6. УНАКРСНА ПЛАТФОРМА И ПРОШИРЕЊА

Иако обе платформе користе исти *SPI* протокол, имплементације на *STM32* и на *Linux*-у се суштински разликују у погледу начина организације и руковања *SPI* трансакцијама.

У имплементацији *Linux SPI* драјвера користе се кернел сервиси. Синхронизација између позива функција и одговора хардвера се управља кроз *completion mechanism*. Док се за *STM32* платформу користе *RTOS* кернел сервиси. Конкретно синхронизациони механизми које он нуди.

У оквиру развоја *RPC* система, успостављена је *Host* библиотека реализована као *Git* submodule, која омогућава апстраховање заједничких *RPC* функционалности и њихово лако дељење између различитих пројеката и платформи. Овај механизам омогућава да се у више независних пројеката користи исти интерфејс, док се платформи специфичне имплементације могу прилагодити или заменити без утицаја на остатак система.

7. ИСКОРИШЋЕЊЕ РЕСУРСА

За анализу искоришћења меморије примењени су алати *riscv64-unknown-elf-nm* и *elf-size-analyze*, који омогућавају идентификацију највећих потрошача меморије и прецизно профилисање *ELF* фајла.

Утврђено је да секција *rodata* заузима значајан део *DCCM*-а, што је узроковало прекорачење расположиве меморије током компилације.

Проблем је решен релокацијом *rodata* секције у *ICCM*, што је подржано архитектуром *SweRV EH1*, и смањењем величине *stack*-а у складу са реалним потребама.

Овим оптимизацијама постигнуто је ефикасније коришћење ресурса и очувана је стабилност система.

8. ЗАКЉУЧАК

У овом раду успешно је реализован *RPC* систем који омогућава стабилну комуникацију између уграђеног *Wi-Fi HaLow* уређаја и спољашњег *host*-а, уз поуздано функционисање у реалном времену. Систем је прилагођен ограничењима уграђених окружења и подржава проширивост на више хардверских платформи.

Иако безбедност није била приоритет у овој фази, потенцијал за даље унапређење постоји увођењем механизама за аутентикацију и енкрипцију података. Уколико будуће апликације буду захтевале поуздану заштиту комуникације, систем се лако може проширити тим функционалностима. *RPC* може бити проширен механизмима за аутентикацију и шифровање, чиме може испунити критеријуме за *SESIP Level 3*, што се често захтева у индустријским апликацијама [5].

9. ЛИТЕРАТУРА

[1] “Distributed Systems : Concepts and Design ”, *by George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair.*

[2] https://github.com/riscv-collab/riscv-gnu-toolchain (приступљено у априлу 2025.)

[3] https://github.com/pulp-platform/axi_spi_slave/blob/master/spi_slave_regs.sv (приступљено у априлу 2025.)

[4] https://github.com/pulp-platform/axi_spi_slave/blob/master/spi_slave_cmd_parser.sv(приступљено у априлу 2025.)

[5] https://globalplatform.org/sesip/ (приступљено у јулу 2025.)

Кратка биографија:

Марија Нешић рођена је у Кикинди 1999. год. Дипломски рад на Факултету техничких наука из области Електротехнике и рачунарства – Ембедед системи и алгоритми одбранила је 2022. године.
контакт: nesicmarija123@gmail.com