УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Лукач

Системска програмска подршка за DOC400 контролер заснована на OpenWRT систему

МАСТЕР РАД

Нови Сад, 2016.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Дипломски – мастер рад	
Аутор, **АУ**:		**Никола Лукач**	
Ментор, **МН**:		**Доц. др Иштван Пап**	
Наслов рада, **НР**:		**Системска програмска подршка за DOC400 контролер заснована на OpenWRT систему**	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		**2016**	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**8/55/11/9/31/0/2**	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		**ИоТ, уграђени системи, системска програмска подршка, кућна аутоматизација**	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		**У раду је описана реализација системске програмске подршке за главни контролер једног система за аутоматизацију куће. Под појмом системске програмске подршке се подразумева низ модификација радног програмског окружења, као и интеграција постојећих програмских решења и алата.**	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	**Доц. др Милан Лукић**	
	Члан:	**Доц. др Иван Каштелан**	Потпис ментора
	Члан, ментор:	**Доц. др Иштван Пап**	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		**Nikola Lukač**	
Mentor, **MN**:		**Ištvan Pap, PhD**	
Title, **TI**:		**System software for DOC400 controller based on OpenWRT system**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2016**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**8/55/11/9/31/0/2**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**IoT, embedded systems, system software, home automation**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper covers implementation of system software for the main controller in one home automation system. System software includes modifications of existing SDK and integration of other programs and tools.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Milan Lukić, PhD**	
	Member:	**Ivan Kaštelan, PhD**	Menthor's sign
	Member, Mentor:	**Ištvan Pap, PhD**	

Zahvalnost

Na prvom mestu neizmernu zahvalnost dugujem porodici na pomoći i strpljenju tokom godina mog studiranja, svim prijateljima, nastavnom osoblju, kao i svim onim ljudima koji su na direktan ili indirektan način doprineli da se ovaj rad realizuje.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

DOC400 - *Desktop OBLO Controller 400*, Stoni OBLO kontroler 400

LED - *Light-emiting Diode*, Svetlosna dioda

SDK - *Software Development Kit*, Skup alata za razvoj softvera

CPU - *Central Processor Unit*, Centralni procesor

GND - *Ground*, Oznaka za signal na nultom potencijalu

DDR - *Double Data Rate*, Dvostruka brzina prenosa

RAM - *Random-Access Memory*, Memorija sa proizvoljnim pristupom

SPI - *Serial Peripheral Interface*, Sprega za serijsku komunikaciju

I2C - *Inter-Integrated Circuit*, Magistrala za povezivanje mikrokontrolera

SDA - *Serial Data Line*, Linija podataka

SCL - *Serial Clock Line*, Linija takta

NFC - *Near-Field Communication*, Komunikacija kratkog polja

USB - *Universal Serial Bus*, Univerzalna serijska magistrala

GPIO - *General-Purpose Input/Output*, Prolazi opšte namene

QSDK - *QCA Software Development Kit*, QCA programsko okruženje

MTD - *Memory Technology Device*, Podsistem za memorijske uređaje

JFFS2 - *Journalling Flash File System version 2*, Tip sistema datoteka

YAFFS2 - *Yet Another Flash File System*, Tip sistema datoteka

UBIFS - *Unsorted Block Image File System*, naslednik *JFFS2*

UBI - *Unsorted Block Images*, Memorijski podsistem namenjen za *UBIFS*

ECC - *Error-Correcting Code*, Kod za ispravljanje grešaka

LEB - *Logical Erase Block*, Logički blok

PEB - *Physical Erase Block*, Fizički blok

1. Uvod

Ovaj rad obuhvata tematiku koja se tiče oblasti automatizacije kuća. Tačnije, proces integracije glavnog kontrolera koji ima ulogu centralnog uređaja u sistemu koji se sastoji od grupacija različitih uređaja, u formi senzora i aktuatora, dalje u tekstu čvor (engl. *Node).* DOC400 (engl. *Desktop OBLO Controller*) kontroler je namenski projektovan da podrži upravljanje i koordinaciju čvorovima koji implementiraju određene protokole bežične komunikacije, kao što su *ZigBee* i *Z-Wave*. Pored toga, kontroler pruža minimalnu korisničku interakciju i signalizaciju putem tastera i svetlosne *RGB (*engl. *Red Green Blue*) indikacije, dok se preostali set naprednog korišćenja i kontrole omogućava putem mobilnih aplikacija.

DOC400 je baziran na MIPS arhitekturi, pa se kao polazno radno programsko okruženje (engl. *Software Development Kit - SDK*) koristi verzija okruženja pod nazivom *QSDK* čiji se mehanizmi zasnivaju na *The Buildroot*-u [1], alata koji automatizuje i pojednostavljuje proces prevođenja kompletnog sistema za ciljanu platformu i arhitekturu. Pod pojmom sistemske programske podrške za DOC400 se podrazumeva modifikacija postojećeg radnog okruženja. U glavnim crtama potrebno je uraditi sledeće:

- Integracija biblioteka neophodnih za rad ostalih softverskih komponenti
- Dodavanje postojećih programa/aplikacija u sistem u standardnom formatu paketa
- Modifikacija konfiguracionih datoteka za dati kontroler
- Definisanje nove mape fleš (engl. *Flash*) memorije
- Implementacija mehanizma sigurnog ažuriranja kompletne programske podrške (engl. *Firmware*), kao i zaštitu od mogućeg prestanka rada kontrolera (engl. *Brick a device*)
- Integracija alata za generisanje izvršnih datoteka neophodnih za rad ovako projektovanog sistema

Realizacija ovog rada pre svega je zahtevala upoznavanje sa postojećim radnim programskim okruženjem, razumevanje različitih mehanizama sprezanja hardverskih modula u sistemu kao i tehnike otklanjanja i ispitivanja problema nastalih tokom samog razvoja na platformi.

Izazov u radu sa ovakvom platformom jeste pre svega u prevazilaženju hardverskih ograničenja u pogledu veličine radne memorije, mogućnosti centralne jedinice kao i dostupne veličine fleš memorije. To zahteva vođenje računa o načinu projektovanja i realizacije sistemske programske podrške, počev od stvari bliskih samoj platformi i dostupnom hardveru, preko konfiguracije operativnog sistema pa sve do pisanja programa. Posle čega dolaze optimizacije opet na različitim nivoima, što sve za cilj ima dobijanje jedne izvršne datoteke koja se skladno i u najboljoj mogućoj sprezi izvršava na datom kontroleru, u okviru jednog sistema za automatizaciju kuća.

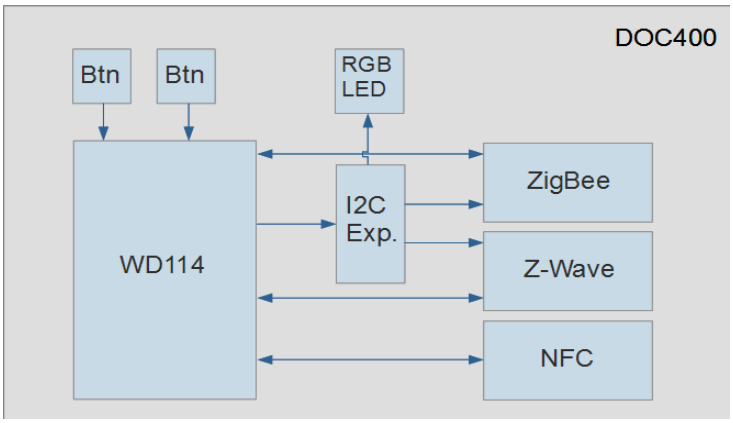
Rad sadrži neophodne teorijske osnove za razumevanje realizacije datog zadatka, opis samog rešenja kao i poglavlje koje obuhvata testiranje, u kojem se provera validnost svih komponenti (modula) sistema kao i merenja performansi samog kontrolera.

2. Teorijske osnove

Poglavlje koje sledi ima namenu da pre svega upozna čitaoca sa arhitekturom date platforme, svim nezavisnim modulima u okviru nje kao i načinom njihovog sprezanja u jedinstven uređaj, kontroler DOC400. Takođe, osnove potrebne za razumevanje rada jednog sistema/alata koji pojednostavljuje i automatizuje proces generisanja svih potrebnih segmenata softvera namenjenih za ugrađene sisteme (engl. *Embedded System*). U nastavku su date informacija o nekim specifičnostima vezanih za vrstu fleš memorija koje se koriste u ovakvim sistema kao i sistem datoteka (engl. *File System*) koji se koriste u tim slučajevima.

2.1 Arhitektura DOC400 kontrolera

Ovaj kontroler predstavlja uređaj koji se sastoji od više nezavisnih hardverskih modula, koji su u sprezi sa glavnim modulom. Dakle, kontroler je projektovan kao centralizovan sistem, gde postoji jedan glavni modul, u našem slučaju to je WD114 [2] modul i više manjih modula (*ZigBee*, *Z-Wave* itd.) koji su u direktnoj sprezi sa glavnim. Slika 2.1. daje pojednostavljen prikaz organizacije kontrolera na nivou modula, opisanih zasebno u narednim naslovima.

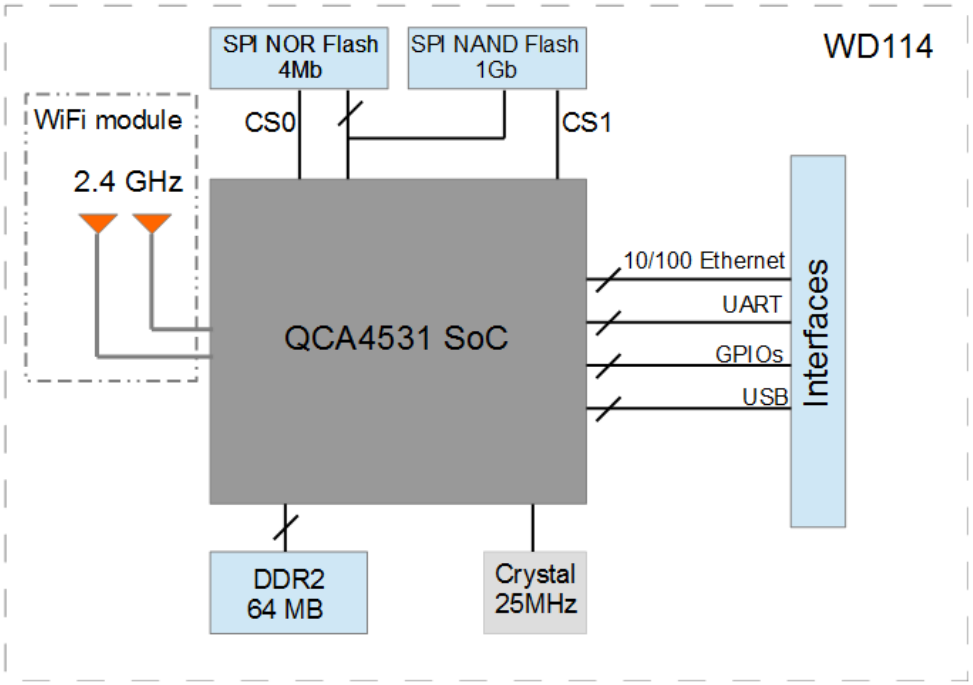


Slika 2.1: DOC400 organizacija modula

2.1.1 Modul WD114

WD114 je modul u čijem središtu se nalazi *Qualcomm Atheros* QCA4531 *SoC* (engl. *System on a Chip*) [3]. Pripada grupi ruterskih (engl. *Router*) čipova visokih performansi koji pronalazi upotrebu u *IoE* (*Internet of Everything*) granama, pa samim tim i u sistemima za automatizaciju kuća, kao i upotrebi u *WiFi* mrežama kao *repeater*/*range extender*.

Modul WD114 se sastoji od sledećih komponenti: QCA4531, 64 MB DDR2 (engl. *Double Data Rate*) memorije, 128Kb NOR SPI (engl. *Serial Peripheral Interface*) fleša, 1Gbit NAND SPI fleša. Slika 2.2 daje pojednostavljen prikaz organizacije modula.

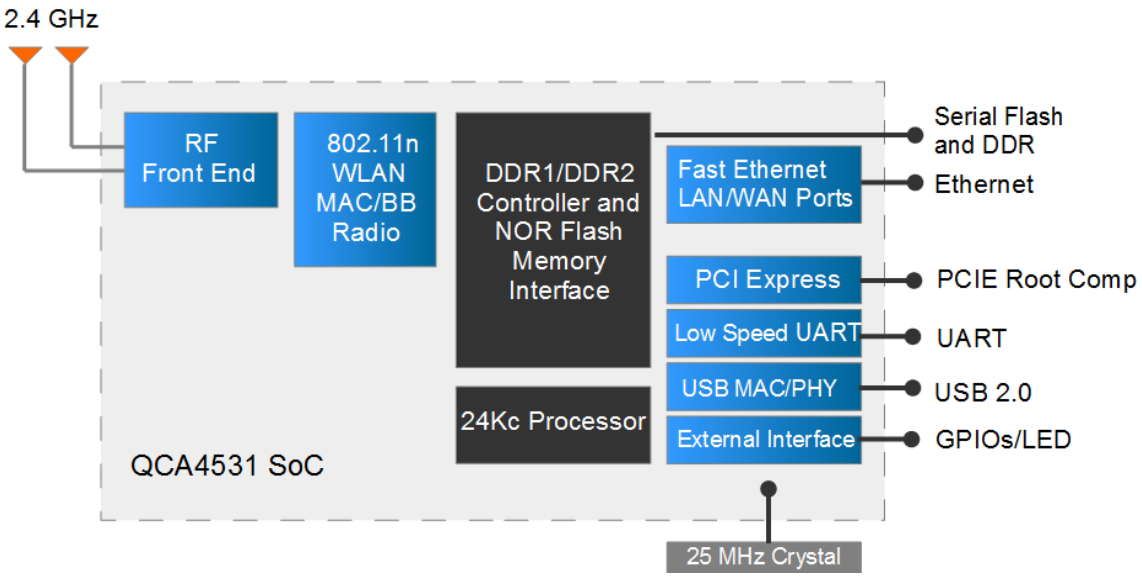


Slika 2.2: Periferije modula WD114

QCA4531 je baziran na *MIPS* arhitekturi iz familije 24Kc procesora. Osnovne karakteristike: *RF* (engl. *Radio Frequency*) prijemno/predajni modul na 2.4GHz; prolazi sa brzim komutiranim (engl. *Switch*) *Ethernet*-om; *DDR* i *SPI* kontroleri; izlaz za spori *UART* (engl. *Universal Asynchronous receiver/transmitter*); *USB* 2.0 (engl. *Universal Serial Bus*) magistrala; 12 konfiguracionih *GPIO* (engl. *General Purpose Input/Output*) prolaza. Slika 2.3 daje uvid u arhitekturu QCA4531 SoC.

Platforma je zasnovana na *Linux* operativnom sistemu, tačnije distribuciji pod nazivom OpenWrt, a koja je namenjena za ruterske čipove. Razvoj softvera i programske podrške za ovu platformu je obezbeđen skupom alata integrisanih u okruženje, ili drugačije rečeno *Build System* poznato kao *The Buildroot*. Tačnije, razvoj za QCA4531 je obezbeđen okruženjem pod nazivom

QSDK (engl. *QCA Software Development Kit*), a koje se sastoji od određenih proširenja i izmena u odnosu na prethodno pomenuti *The Buildroot*. U jednom od narednih poglavlja biće do određenih detalja opisan *QSDK*.



Slika 2.3: Arhitektura QCA4531 *SoC*

2.1.2 I2C modul za proširenje broja *GPIO* prolaza

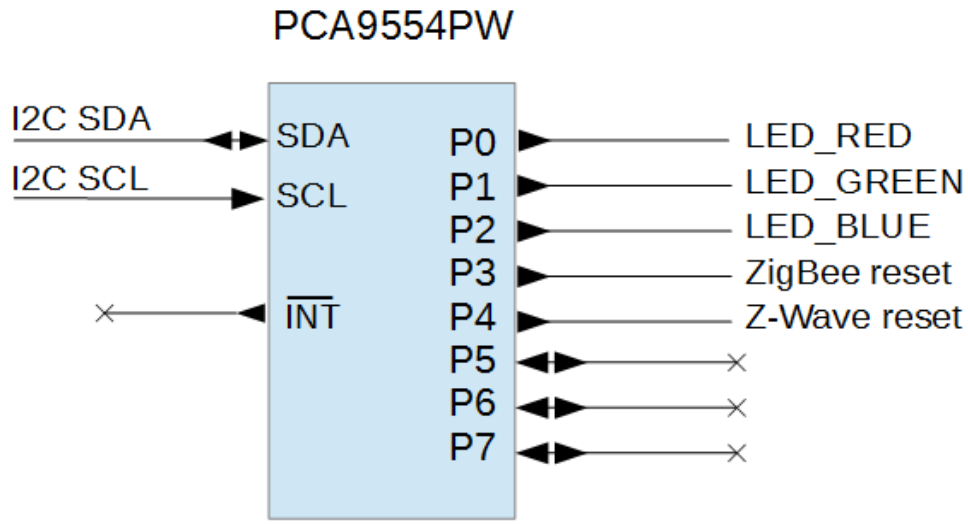
PCA9554 modul pruža proširenje broja softverski konfiguracionih *GPIO* prolaza u okviru kontrolera. Spreže se sa glavnim modulom preko *I2C* (engl. *Inter-Integrated Circuit*) magistrale. Kako QCA4531 ne poseduje *I2C* kontroler, tj. namensku hardversku realizaciju datog protokola, pristupa se korišćenju mehanizma pod nazivom *Bit-Banging*, odnosno softverskoj implementaciji datog protokola. Mehanizam zahteva dva *GPIO* prolaza koji će biti namenjeni za prenos podataka i takta, a koji će zameniti stvarne linije *I2C* protokola, *SDL* (engl. *Serial Data Line*) i *SCL* (engl. *Serial Clock Line)* linije.

Ponašanje i smer izlaznih *GPIO* prolaza se postavlja pomoću 4 8-bitna registra:

- Konfiguracioni registar ulaza/izlaza (engl. *Configuration Register*)
- Registar ulaznih *GPIO* signala (engl. *Input Port Register)*
- Registar izlaznih *GPIO* signala (engl. *Output Port Register)*
- Registar za inverziju polariteta signala (engl. *Polarity Inversion Register*)

Najmanje značajan bit svakog od registara odgovara pinu P0 na izlazu iz modula, dok bit najveće težine odgovara pinu P7. Postavljanjem bita u okviru konfiguracionog registra na 1 (što je ujedno i podrazumevana vrednost) određuje dati prolaz da bude ulazni, u suprotnom, prolaz je

izlazni. Registar ulaznih signala ima samo dozvolu čitanja, gde vrednost svakog od bita u okviru registra određuje stanje prolaza njemu pridruženog signala. Postavljanje vrednosti bita registra izlaznih *GPIO* signala na 0 ili 1 daje na izlaznom prolazu logičku jedinicu odnosno nulu, respektivno. Ukoliko je potrebna invertovana logiku signala koji su definisani kao ulazni, koristi se registar za inverziju polariteta, gde se postavljanjem bita registra na 1 obrće logika prolaza njemu dodeljenog signala. Prikaz pridruženih signala spoljnih modula na *GPIO Expander* dat je na slici 2.4.



Slika 2.4: Signali pridruženi *I2C GPIO Expander* prolazima

2.1.3 Moduli za bežičnu komunikaciju sa čvorovima

DOC400 kontroler kao glavni koordinator u okviru jednog sistema za automatizaciju kuća, komunicira sa velikim skupom različitih senzora/aktuatora. Osnovna razlika između ovakvih uređaja potiče od vrste bežične komunikacije na koju se oslanja dati uređaj. Trenutno, postoji velika raznovrsnost ovakvih protokola, koji se koriste u domenu automatizacija kuća, interneta stvari (engl. *Internet of Things*), itd. U okviru ovog sistema, podrška je obezbeđena za tri različita protokola za bežičnu komunikaciju, kroz modul za *ZigBee*, *Z-Wave* i *NFC* (engl. *Near Field Communication*). Za svaki od protokola postoji namensko integrisano kolo, pa tako postoje tri modula za svaki od protokola ponaosob.

ZigBee modul se spreže sa glavnim modulom putem serijskih linija za komunikaciju, dok je reset signal povezan na *I2C GPIO Expander* na jedan od prolaza, što isto važi i za reset signal Z-Wave modula, pri čemu se glavna komunikacija u ovom slučaju obezbeđuje linijama *USB* magistrale. Na kraju, *NFC* modul je povezan direktno sa glavnim modulom koristeći *I2C* magistralu.

2.1.4 Sprega za indikaciju stanja i korisničku kontrolu kontrolera

U osnovi, kontrolerom i njemu pridruženim uređajima se upravlja putem mobilnih klijentskih aplikacija. Međutim, jedan određeni skup operacija se može obaviti putem tastera na kontroleru. Pritiskom jednog od dva tastera ili kombinacijom oba, različitim brojem kratkih pritisaka ili dugim pritiskom tastera od nekoliko sekundi, izvršiće neku od predefinisanih operacija. Kontrola u oba slučaja upravljanja, koja menja stanje kontrolera se manifestuje menjanjem boja i periode paljenja i gašenja *RGB LED* diode. Izvodi sve tri komponente RGB diode su priključeni na *I2C GPIO Expander*, gde se kombinacijom stanja ta tri izvoda dobija set od 7 boja.

2.2 Operativni sistem *OpenWrt*

OpenWrt [4] je vrsta operativnog sistema namenjena za ugrađene platforme i uređaje sa ograničenim resursima, kao što je takt centralne procesorske jedinice (engl. *Central Processing Unit - CPU*), veličina *RAM*-a (engl. *Random Access Memory*), veličina fleš memorije itd. Poznat je dakle kao posebna distribucija *Linux* operativnog sistema.

Umesto pokušavati napraviti svaki put novi, jedinstveni i statičan firmver (engl. *Firmware*), *OpenWrt* pruža sistem datoteka potpuno podložan izmenama i konfiguraciji pomoću menadžera paketa (engl. *Package Management*), gde je paket bilo koja sistemska ili korisnička aplikacija, deljena i statička biblioteka itd. Ovim se postiže sloboda u odabiru paketa na aplikativnom nivou bez ulaženja u detalje konfiguracije vezane za samu platformu datog proizvođača. Trenutno, *OpenWrt* sistem ima zvaničnu podršku za preko hiljadu platforma različitih proizvođača. Međutim, podrška za QCA4531 se ne nalazi u listi podržanih uređaja *OpenWrt* distribucije, pa se za potrebe razvoja aplikacija i kompletne sistemske podrške koristi *QSDK*.

2.2.1 Qualcomm Software Development Kit

QSDK [5] predstavlja modifikovanu verziju *OpenWrt* sistema, tačnije proširenja koja se tiču specifičnosti vezanih za *QCA – Qualcomm* platforme. Najbitnija poboljšanja i razlike se odnose uglavnom na rukovaoce (engl. *Driver*) za mrežne adaptere iz familije *Atheros* čipova, kao i promene vezane za *SPI* magistralu preko koje se upravlja sa *NAND* memorijom. *QSDK* omogućava generisanje izvršnih datoteka baziranih na *OpenWrt* distribuciji sa dodatnim poboljšanjima za *Qualcomm Atheros* skup čipova (engl. *Chipset*), koji nisu još ušli u zvaničan *OpenWrt* repozitorijum podržanih uređaja. U potpunosti su nasleđeni mehanizmi konfiguracije, organizacije i menadžmenta paketa, kao i drugih alata direktno iz *OpenWrt Build* sistema.

Izvorni kod programa za učitavanje operativnog sistema (engl. *Bootloader*), konkretno *U-Boot* (*Universal Bootloader*) [6] je takođe integrisan u *QSDK*.

2.2.1.1 Struktura direktorijuma unutar *QSDK*-a

Poglavlje daje prikaz jednog dela direktorijuma *QSDK* okruženja kao i kratak opis i funkciju direktorijuma i datoteka od većeg značaja za razvoj aplikacija i sistemske programske podrške za kontroler DOC400. Opis direktorijuma:

- **bin/** - sve izvršne datoteke za datu platformu (.bin), datoteke za svaki odabrani paket (.ipk), kao i sliku sistema datoteka (.jffs, .ubi)
- **build_dir/** - unutar ovog direktorijuma se nalaze tri poddirektorijuma host/, target/ i toolchain/ i predstavljaju lokacije gde se vrši prevođenje alata za računar domaćin (engl. *Host*), programskog koda paketa i prevođenje međuplatformskih alata, respektivno. Svakom paketu se dodeljuje zasebna lokacija u okviru target/ direktorijuma u kom se nalazi kopija izvornog i konfiguracionog koda, kao i svaki drugi oblik sadržaja u vidu video, tekst datoteka koje se kopiraju na datu lokaciju u cilju prevođenja
- **files/** - predstavlja strukturu sistema datoteka koji treba da se nađe na datoj platformi prilikom prvog pokretanja. Na primer, ukoliko je potrebno da neka konfiguraciona datoteka treba da bude na sistemu umesto podrazumevane. To se postiže pravljenjem kopije sistema datoteka u okviru files/ direktorijuma i menjanjem samo onih datoteka koje su od značaja
- **packages/** - struktura direktorijuma svih paketa od kojih će neki završiti na platformi u zavisnosti od konfiguracije. Svaki paket je zaseban direktorijum u kojem se nalazi poddirektorijum **files/** gde se nalazi izvorni kod kao i datoteka **Makefile**, koja definiše pravila konfigurisanja, prevođenja i instaliranja datog paketa. O strukturi paketa, primeru pisanja novog biće reči u narednom poglavlju
- **qca/** - u okviru ovog direktorijuma se nalaze specifičnosti vezane za *Qualcomm Atheros* čipove, izvorni kod modifikovanog *U-boot*-a, kao i predefinisane konfiguracije za platformu
- **staging_dir/** - prevedeni alati za *Host* mašinu, međuplatformski alati, kao i prevedene statičke i dinamičke biblioteke koje koriste drugi paketi tokom prevođenja

2.2.1.2 Paketi u okviru *QSDK* okruženja

Paket je softverska gradivna jedinica jednog *OpenWrt* sistema. U osnovi predstavlja jednu korisničku ili sistemsku aplikaciju, statičku ili dinamičku biblioteku, sistemski pozadinski proces ili na kraju krajeva samo jedan objedinjeni skup dokumenta ili resursa koji treba da završe na ciljanom sistemu. Kako je *OpenWrt* jedna *Linux* distribucija, postoji i u ovom slučaju mehanizam za rad sa paketima pod nazivom *OPKG*. To je uslužni program (engl. *Utility*), u funkciji menadžera paketa koji se koristi za preuzimanje i instaliranje paketa sa udaljenih internet repozitorijuma ili repozitorijuma koji se nalaze u lokalu. Ovaj alat takođe razrešava moguće međuzavisnosti paketa i ukazuje na moguće probleme prilikom instalacije novih.

Kroz sistem paketa se nastoji da se na jednostavan način integriše novi softver za datu platformu. To je postignuto pre svega uvođenjem šablona u izgledu i strukturi svakog paketa. Svaki paket se sastoji od tri segmenta:

- **package/Makefile**
- **package/files/**
- **package/patches/**

Važno napomene, prvi je obavezan, dok se poslednja dva mogu izostaviti. *Makefile* datoteka je od važnosti jer pre svega sadrži konkretne korake preuzimanja sadržaja ukoliko se ne nalazi u lokalu, tj, u **files/** direktorijumu. U okviru direktorijuma **patches/** se nalaze prepravke programskih grešaka (engl. *Bugs*) ili koje se tiču optimizacija koda.

Jednostavnost pisanje potiče od već definisanih direktiva i promenljivih unutar *rules.mk i package.mk* datoteka koje se uključuju unutar *make* datoteke svakog paketa. Najznačajnije promenljive koje se koriste prilikom opisa paketa:

- PKG_NAME - naziv paketa kako ga vidi alat *menuconfig* i *opkg*
- PKG_VERSION - verzija paketa koja figuriše u nazivu i prilikom preuzimanja izvornog koda sa udaljenog repozitorijuma
- PKG_RELEASE - verzija *make* datoteke datog paketa
- PKG_BUILD_DIR - lokacija na kojoj se kreira poseban direktorijum za dati paket i odvija prevođenje istog
- PKG_SOURCE - naziv izvornog koda paketa
- PKG_SOURCE_URL - adresa repozitorijuma sa kog se preuzima izvorni kod paketa
- PKG_SOURCE_PROTO - protokol koji se koristi za preuzimanje sadržaja
- PKG_MD5SUM - kontrolni zbir (engl *Checksum*) za proveru preuzetog sadržaja
- PKG_CAT - indikacija algoritma za dekompresiju sadržaja (zcat, bzcat, unzip)

- PKG_INSTALL - postavljanjem na ‘1' poziva se “make install” datog paketa
- PKG_INSTALL_DIR - lokacija gde “make install” kopira prevedene datoteke

Blokovi definicija služe da se navedeni argumenti proslede direktivama koje će zatim izvršiti kopiranje, konfigurisanje, prevođenje i na kraju zapisati u *.ipk* arhivu pravila instalacije koje koristi *opkg* menadžer za dati paket na ciljanoj platformi. Neizostavni blokovi su:

- **Package/naziv_paketa** - osnovne informacije o paketu opisane kroz skup argumenata:
  - CATEGORY - kojoj grupi/kategoriji pripada dati paket
  - TITLE - kratak opis namene paketa
  - URL - na kom repozitorijumu se može pronaći paket
  - DEPENDS - međuzavisnost prema drugim paketima
- **Package/naziv_paketa/description** - slobodan tekstualni opis datog paketa
- **Package/install** - skup komandi za kopiranje datoteka koje treba da završe u .ipk arhivi, tj. da budu kopirane na definisane lokacije ciljane platforme prilikom instalacije paketa

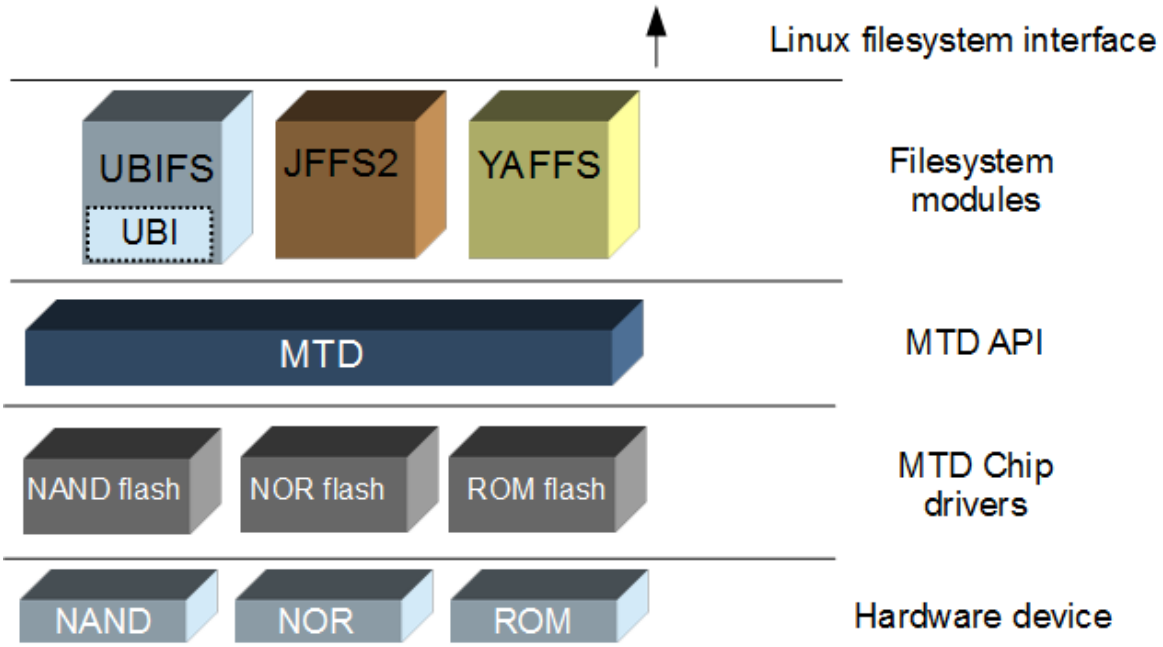
Neki opcioni blokovi su **Build/Prepare**, **Build/Compile**, **Build/Install**, **Build/InstallDev**, **Package/preinst**, **Package/postinst**, itd. Detaljan opis svakog korišćenog bloka sa njemu pridruženim argumentima će biti na primeru jednog paketa u okviru poglavlja programskog rešenja.

2.3 Sistemi datoteka na ugrađenim platformama

Kada se radi o platformama ovog tipa, onda se obično podrazumeva ograničen sistem u vidu procesorskih i memorijskih resursa koje je posledica kako potrebe za uređajem malih dimenzija tako i mnogo bitnije, zahteva za malom potrošnjom energije. Iz tih razloga, prilikom izbora trajnih (engl. *Non-volatile*) memorija, najčešće se vrši odabir iz grupe „sirovih“ (engl. Raw) memorija, kojoj pripadaju NAND i NOR fleš memorije. Kako ovaj tip memorije ne pripada niti blokovski niti karakterno orijentisanim memorijama, potreban je drugačiji pristup korišćenja u odnosu na postojeće mehanizme unutar *Linux* sistema, tj. potrebno je koristiti drugačiji mehanizam upravljanje memorijom.

MTD (engl. *Memory Technology Devices*) [7] podsistem pruža sloj apstrakcije za ovaj tip memorije i čini mogućim da se koristi jedinstven *API* (engl. *Application Programming Interface*) kada se radi sa različitim tipovima memorijskih tehnologija, kao što su NAND, OneNAND, NOR, ECC'd NOR, itd. Tek se nad ovim podsistemom može govoriti o odabiru različitih sistema datoteka.

Trenutno postoji tri najzastupljenija tipa sistema datoteka koje se biraju u zavisnosti od nekoliko parametara, kao što su brzina pristupa i veličina particije koja se koristi. To su *JFFS2* (engl. *Journaling Flash File System*), *YAFFS* (engl. *Yet Another Flash File System*) i *UBIFS* (engl. *Unsorted Block Image File System*). Sva tri poseduju mehanizam praćenja loših blokova (gradivne jedinice NAND memorija), sistem uniformnog raspoređivanja trošenja ćelija bloka (engl. *Wear Leveling*) i mehanizam ispravljanje grešaka prilikom upisa i čitanja podataka (engl. *Error Correction*), te zbog toga predstavljaju pouzdane sisteme datoteka za industrijsku namenu. U okviru ovog sistema kontrolera DOC400, koristi se *JFFS2* i *UBIFS*, o kojima će biti dato malo više detalja. Softverski moduli nad jednom fleš memorijom dat je na Slici 2.5.



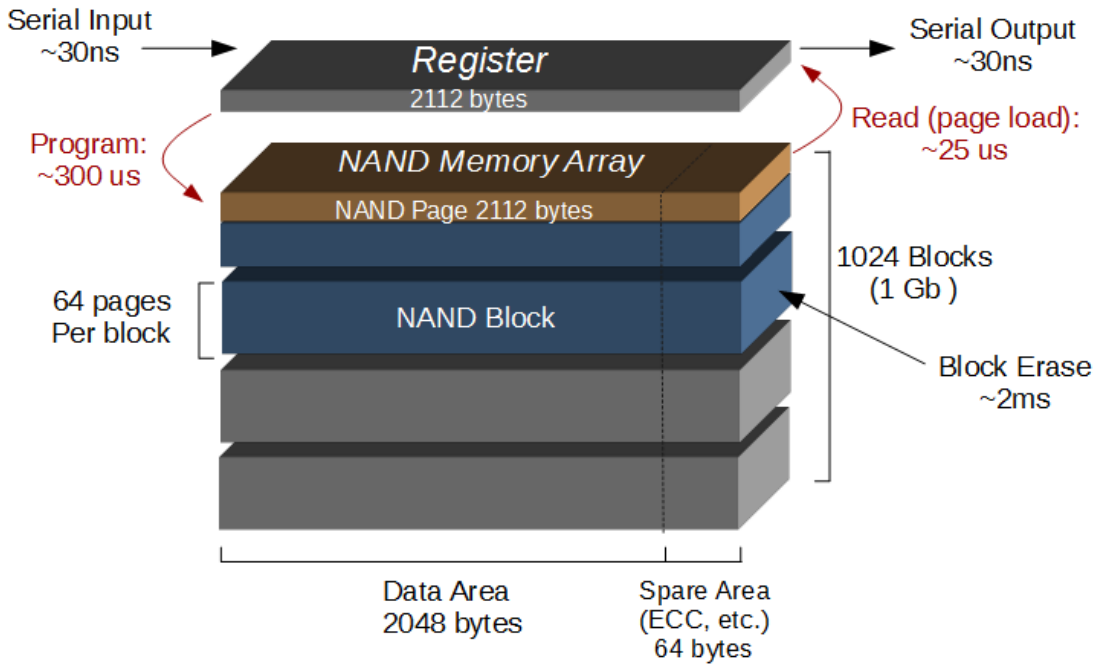
Slika 2.5: *MTD* podsistem kao sprega između memorije i sistema datoteka

2.3.1 Tehničke informacije *NAND* fleš memorije

NAND memorija je uređena kao niz sukcesivnih stranica (engl. *Page*). Jedna stranica se sastoji od 256/512/1024/2048 bajtova (engl. *Byte*) podataka i dodatnih 8/16/64/128 bajtova rezervnog regiona za svaku od stranica. Uglavnom su novije memorije u formatu 2048/64, od kojih neke poseduju deljenje jedne stranice od 2048 bajtova na podstranice od 512 bajtova, što nije slučaj kod WD114 modula. N susednih stranica formiraju jedan fizički blok (engl. *Block*). Osnovna jedinica pisanja/čitanja memorije je upravo stranica, dok se brisanje odvija na nivou bloka.

Rezervni region, poznat kao i *Out-of-Band* region ima funkciju u čuvanju dodatnih meta podataka vezanih za stranicu kojoj je pridružen. Jedan deo ovog regiona se koristi od strane

rukovaoca *NAND* memorije za skladištenje *ECC* (engl. *Error Correction Code*) informacija koje se generišu ukoliko je došlo do greške prilikom upisa korisničkih podataka. Tako generisani kodovi se koriste prilikom čitanja podataka sa date stranice kako bi se ispravile greške nastale tokom upisa i dostavili validni, ispravni podaci. Drugi deo regiona se koristi od strane sistema datoteka koji smešta specifične informacije, kao što je indikacija iskorišćenja stranice, u vidu broja upisa, a koja se koristi u procesu ravnomernog raspoređivanja upisa stranica. Prikaz organizacije *NAND fleš* memorije koja se koristi u okviru kontrolera DOC400, dat je na Slici 2.6.



Slika 2.6: Struktura 1Gbit SLC *NAND* fleš memorije

Činjenica da se upis vrši na nivou stranice, sistemski zahtevi upisa na memoriju nisu sinhroni ukoliko se upisuje količina podataka manja od veličine stranice, to ujedno predstavlja problem i izazov kojim se bave sistemi datoteka nad ovim tipom memorija. Naime, da bi se ostvario što duži životni vek memorijskih ćelija, ideja je raditi upise ređe, izbegavati upisivanje delimično popunjenih stranica. U tu svrhu se koriste bafer veličine jedne stranice u okviru *RAM* memorije, koji se prazne i upisuju svoj sadržaj na stranicu u tri slučaja: kada je bafer pun; kada se eksplicitno izda sistemski zahtev pražnjenja svih bafera; i u slučaju periodičnog pražnjenje bafera sa konstantnom periodom koju zadaje *Linux* sistem. U zavisnosti od prirode sistema, količine podataka koja protiče, učestalosti promene podataka, njihove važnosti u slučaju gubljenja, vrši se odabir sistema datoteka, podešavanje parametara rada i na kraju pisanje softverskog koda imajući i vidu sinhronizaciju podataka.

2.3.2 *UBI* i *UBIFS*

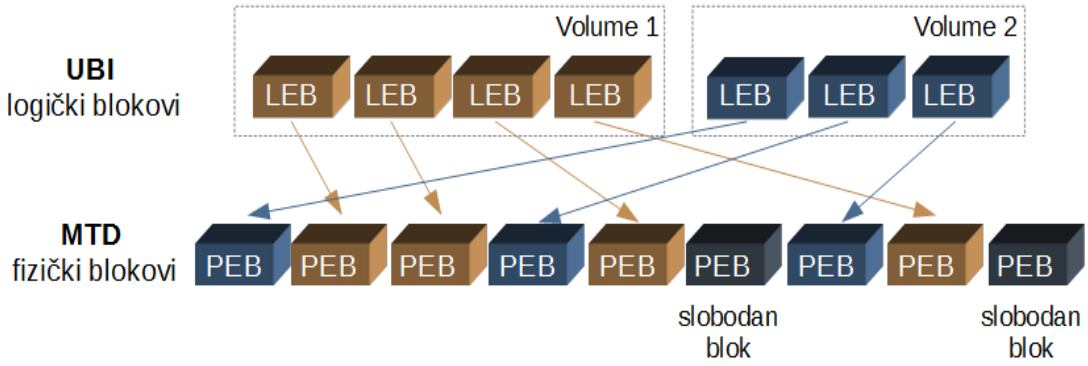
Za razliku od *JFFS2* sistema datoteka koji je prilično jednostavnije implementiran i direktno se oslanja na *MTD* particije, *UBIFS* ima jedan stepen indirekcije više, i malo usložnjava stvari. Drugačije rečeno, postoje 3 podsistema u ovom slučaju :

- *MTD* podsistem, koji daje uniformnu prezentaciju fleša u vidu MTD particije (primer.., /dev/mtd0), koji se direktno koristi u slučaju *JFFS2*
- *UBI* (engl. *Unorted Block Images*), pruža sistem ravnomernog raspoređivanja upisa i pojam logičkih jedinica (engl. *Logical Volume*)
- *UBIFS* sistem datoteka, koji se oslanja na podsistem logičkih blokova

UBI predstavlja upravljačku strukturu nad logičkim blokovima jedne fleš memorije. Osnovna funkcija ovog podsistema je upravo u povezivanju logičkih blokova (engl. *LEB - Logical Eraseblock*) sa fizičkim (engl. *PEB - Physical Eraseblock*). Ta je osnova mehanizma za globalno ravnomerno raspoređivanja upisa nad fizičkim blokovima memorije. Time se takođe obezbeđuje transparentnost u procesu zamene dotrajalih (engl. *Worn-out) PEB* segmenata sa validnim. Sprega logičkih sa fizičkim blokovima je prikazana na slici 2.7.

UBI logička jedinica se sastoji od sukcesivnog niza logičkih blokova, pri čemu je svaki logički blok spregnut sa jednim fizičkim. Veličina logičke jedinice se specificira prilikom kreiranja i može se menjati kasnije u toku rada. Postoje dva tipa *UBI* logičkih jedinica: statičke i dinamičke. Prve su samo za čitanje (engl. *Read only*) i njihov sadržaj se štiti CRC-32 (engl. *Cyclic Redundacy Check*) sumom, dok dinamičke imaju dozvolu pisanja i čitanja i njihov integritet se štiti gornjim slojevima *UBIFS*.

UBI podsistem je u potpunosti svestan loših fizičkih blokova, kako onih koji su nastali tokom proizvodnje u fabrici tako i onih koji se pojavljuju tokom života jedne fleš memorije. Poseduje skup rezervnih, dobrih fizičkih blokova i prilikom nastanka lošeg se vrši zamena tako da se ne primeti od strane gornjih slojeva. Takođe, pojava “bit-flips” nastaje prilikom upisa podataka, pri čemu se jedan (ili više) bit ne promeni sa ‘1‘ na ‘0‘ , a trebao je, koriguje *ECC* sumama, koje obezbeđuju dostavljanje validnih vrednosti prilikom čitanja podataka sa memorije. Vremenom se greške ovog tipa akumuliraju i dovode do nepovratnog gubljenja podataka. Da bi se to izbeglo, *UBI* prenosi sadržaj sa fizičkog bloka koji je pred granicom da izgubi podatke na validan fizički blok, tj. radi interno transparentno pomeranje sadržaja blokova. Proces koji je poznat pod nazivom *scrubbing,* takođe transparentno od gornjih slojeva.



Slika 2.7: Sprezanje *LEB* i *PEB* blokova *UBI* podsistema

UBIFS je izuzetno skalabilan sistem datoteka u odnosu na veličinu memorije . Konkretno, vreme priključivanja i brzina pristupa zadržavaju konstantnu vrednost, ne zavise od veličine memorije niti od nivoa iskorišćenosti iste [8]. Međutim, to ne važi za *UBI* podsistem, čije se performanse skaliraju linearno sa promenom veličine memorije. U kombinaciji *UBI*/*UBIFS* garantuje se mnogo bolje skaliranje od konkurentskog *JFFS2*, ali iako *UBI* postane usko grlo, moguće je implementirati *UBI2*, koji navodno rešava probleme njegovog prethodnika. Ovaj sistem datoteka je veoma pouzdan po pitanju čuvanja integriteta podataka prilikom naglog gubitka napajanja kao i u slučaju oštećenja podataka. Vrši se takođe računanje i provera kontrolnih suma prilikom svakog upisa podataka.

Pored veoma dobrih osobina ovog sistema datoteka, jedini nedostatak se javlja pri korišćenju istog nad malim patricijama, reda veličine do 16MByte. Uzrok potiče usled uvođenja zaglavlja za svaki *PEB* kao i potreba dodatnih za podršku različitih mehanizama *UBIFS*-a, a što za posledicu ima smanjenje jednog dela korisničke memorije. Konkretno, u pitanju su sledeći segmenti:

- 2 *PEB*-a se koriste za čuvanje tabele logičkih jedinica (engl. *Volume Table*)
- 1 *PEB* se koristi za *wear-leveling* mehanizam
- 1 *PEB* za atomične operacije menjanja *LEB* sadržaja
- jedan deo *PEB*-ova se čuva za rukovanje pojave loših blokova; taj broj je promenljiv, dok je podrazumevana vrednost 20 na 1024 blokova
- zaglavlja ispred svakog *PEB*-a, čija se veličina računa na dole opisan način

Opis simbola koji figurišu unutar formule za računanje veličine zaglavlja:

- W – ukupan broj svih fizičkih blokova memorije
- P – ukupan broj fizičkih blokova particije za koju se računa *overhead*
- S_P – veličina jednog fizičkog bloka (engl. *physical eraseblock size*)
- S_L – veličina jednog logičkog bloka (engl. *logical eraseblock size*)

- B_B – broj loših blokova date particije
- B_R - broj rezervisanih *PEB*-ova za rukovanje pojave loših blokova (20 * W/1024)
- B – max(B_B, B_R)
- O – dodatna memorija za zaglavlje koja se računa O = S_P - S_L (4KiB u slučaju 2KiB *NAND* stranica)

UBI overhead se računa kao (B - B_B + 4) * S_P + O * (P – B – 4). Ova metoda služi za precizno određivanje odnosa korisnog dela memorije prema delu za meta podatke i time ustanoviti da li se tako projektovan sistem datoteka uklapa u postojeće memorijske okvire sistema za koji je projektovan.

3. Koncept rešenja

Namena ovog poglavlja je da pruži širu sliku zadataka koje potrebno uraditi, pod tim se misli na implementaciju neophodnih, integraciju novih i modifikaciju postojećih softverskih delova jednog okruženja, kako bi se obezbedila potrebna sistemska programska podrška, a dobila funkcionalna jedinica pod nazivom DOC400 koja ispunjava sve zahteve jednog glavnog kontrolera u sistemu za automatizaciju kuće. Opisi su konceptualne prirode, bez detalja implementacije, grupisani u smislene celine kroz podnaslove koji slede.

3.1 Integracija novih paketa

Kako je već rečeno, organizacija svih aplikacija i biblioteka u okviru *QSDK* okruženja je u vidu tzv. paketa. Paketi kao nezavisne celine mogu da se oslanjaju na druge pakete, i time se formira međuzavisnost paketa. U okviru ovog zadatka, potrebno je integrisati četiri aplikacije i jednu C++ biblioteku, u postojeći sistem, ne narušavajući strukturu. Za svih pet paketa, postoji već izvorni kod, potrebno je napisati odgovarajuće *OpenWrt make* datoteke, kako bi se izvršila integracija u okruženje.

3.1.1 POCO C++ biblioteka kao novi paket

Ova biblioteka se koristi kao apstrakcija platforme u sistemskom softveru DOC400 i predstavlja osnovu za mrežne i internet aplikacije. Kako se sve aplikacije koje je potrebno integrisati oslanjaju na *POCO*, zbog praktičnog razloga je biblioteka izdvojena kao zaseban paket, i neće se nalaziti u okviru svake aplikacije kao dok su to bili zasebni projekti van *QSDK* okruženja. Time se smanjuje zauzeće fizičkog prostora diska kao i vreme prevođenja svakog od paketa, usled referenciranja uvek na isti paket. Lokacija za ovaj paket je predviđena u postojećem poddirektorijumu paketa *package/lib*, u kom se već nalaze paketi namenskih biblioteka koje se koriste u procesu prevođenja.

3.1.2 Aplikacije OSM, OMB, OHM u formi paketa

Ove tri međusobno spregnute aplikacije čine softverski deo kontrolera koji vrši upravljanje resursima kontrolera, sadrži logiku kontrole i razmene poruka sa modulima (ZigBee i Z-Wave), mehanizam razmene informacija između aplikacija na kontroleru kao i prema mobilnim aplikacijama i servisima baziranih na *cloud* tehnologijama. Bez ulaženja u detalje i opisa funkcije koje svaka od aplikacija ima, navodi se samo osnovna uloga u okviru DOC400 kontrolera:

- OSM (OBLO System Manager) [9] – aplikacija koja je usko povezana sa resursima (tasterima i *RGB* diodom), u nadležnosti je upravljanja mrežnim delom, (Ethernet i WiFi), vodi računa o pokretanju i zaustavljanju drugih aplikacija u zavisnosti od datog stanja
- OMB (OBLO Message Broker) [10] – aplikacija koja je u nadležnosti kako interne razmene poruka između aplikacija, tako i koordinaciji poruka između kontrolera i mobilnih aplikacija kao i kontrolera i OBLO *cloud* servera
- OHM (OBLO Home Manager) – kompletna logika sistema za upravljanje senzorima, aktuatorima i uređajima u okviru sistema za automatizaciju kuća

Važno napomene je da su ove softverske komponente već implementirane, pa je potrebno izvršiti samo integraciju istih u QSDK okruženje. Svaka od njih je bazirana na *Cmake* alatu koji obezbeđuje međuplatformsku nezavisnost koda. Sistem pisanja *make* datoteka za *OpenWrt* pakete je lepo spregnut sa *Cmake* baziranim programima, pa će proces integraciji biti sličan za sve tri uz minimalne razlike.

Ideja je uvesti poddirektorijum **oblo/** u okviru direktorijuma svih paketa, gde će se nalaziti sve komponente koje pripadaju datoj kategoriji. To bi važilo i za sve buduće aplikacije koje bi se integrisale u sistem. Aplikacije, a sada paketi, će nastaviti da budu nezavisni projekti, koji se održavaju na sopstvenim repozitorijumima, kao i do sada.

3.1.3 Uslužna aplikacija za ažuriranje softvera (UPDMNGR)

Update Manager ili UPDMNGR, je uslužna aplikacija koja ima dve funkcije. Prva, da omogući dobavljanje *serial* i *security* ključeva svim drugim aplikacija kojima su neophodni i druga, da izvrši pripremu nove programske podrške koja je preuzeta sa *cloud* servera na zahtev korisnika, i omogući nastavak procesa ažuriranja iste koji se odvija u domenu programa *U-boot*.

Lokacija za ovaj paket će biti kao i za prethodne tri, u okviru **package/oblo/** poddirektorijuma, sa nazivom paketa **updmngr.** Aplikacije će takođe biti bazirana na *cmake* alatu, pa će integracija biti slična kao i do sada.

3.2 Modifikacije konfiguracionih datoteka

Pod ovim se misli na dve stvari koje je potrebno uraditi:

- Kako ne postoji konkretna konfiguracija za platformu kontrolera DOC400 unutar *QSDK*, preciznije za takvu organizaciju hardverskih komponenti, potrebno je napisati novu, namensku za ovaj sistem
- Usled zahteva za mehanizmima korisničkog ažuriranja programske podrške na kontroleru, kao i mehanizam za vraćanje na fabričku verziju, za dati kontroler je potrebno izvršiti reorganizaciju particija *NAND* memorije, tj. napisati novu memorijsku mapu.

3.2.1 Podrška hardverskih komponenti kontrolera

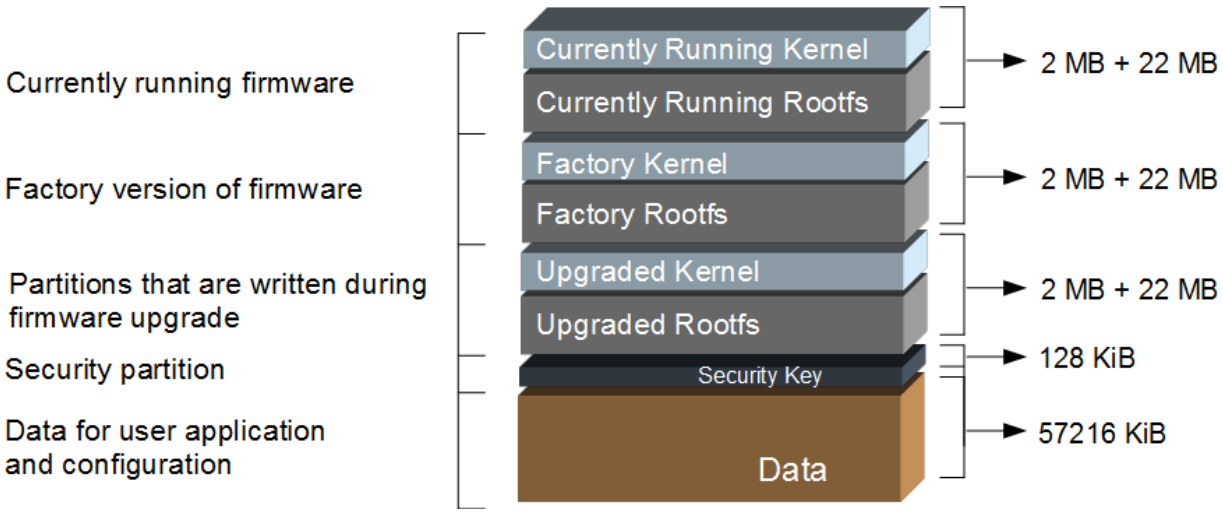
Konfiguraciona datoteka sadrži blokove koda za inicijalizaciju određenih segmenata hardvera. Tu se misli na konfigurisanje *GPIO* pinova za različite funkcije, kao što je kontrola tastera, *RGB* dioda, magistrala tipa *I2C* i *SPI*, kao i podešavanje parametra mreže.

Kao što je rečeno, ne postoji konfiguraciona datoteka za platformu kontrolera DOC400, međutim postoje one koje se tiču QCA4531 *SoC*-a, koji je korišćen na različitim platformama do sada. Kako je sprega ista i u slučaju DOC400 kontrolera, najjednostavnije rešenje je naslediti sve zajedničke korake iz postojećih konfiguracija, a modifikovati i dodati nove segmente kako bi se podržala ova platforma.

3.2.2 Memorijska mapa *NAND* fleš memorije

Potrebno je organizovati memoriju, podeliti je u posebne nezavisne regione (particije) kako bi se obezbedio mehanizam sigurnog ažuriranja programske podrške, kao i podrška za vraćanje na fabrički u slučaju pojave nefunkcionalnog kontrolera. Takođe, određeni delovi memorije će se koristiti za korisničke podatke koji nastaju u toku rada sistema, kao i za različite konfiguracione podatke koje koriste paketi.

Analizom memorijskih zahteva svakog od segmenata , memorija je izdeljena na optimalan način kako bi se svaki deo uklopio u zahtevane okvire. Potrebno je ispoštovati globalno ograničenje poravnanja particija na veličinu od 128KiB, tj. veličinu jednog fizičkog bloka. Grafički prikaz na slici 3.1 daje globalnu sliku organizacije *NAND* memorije na kontroleru DOC400.



Slika 3.1: Memorijska mapa *NAND* fleš memorije

Kao što se vidi sa slike, postoje tri puta po dve particije koje su identične po veličini (2MiB + 22MiB) i predstavljaju tri para *Kernel – Rootfs* (*Root Filesystem*), od kojih se svaki par koristi u različite svrhe. "Trenutna", podrazumeva onu verziju programske podrške koja se izvršava i koristi u redovnom režimu rada, "fabrička" se koristi u mehanizmu vraćanja programske podrške na fabričku verziju ukoliko je stanje kontrolera neupotrebljivo u funkcionalnom smislu i "nova" služi kao trenutna lokacija koja čuva programsku podršku koja treba putem mehanizma ažuriranja da bude prekopirana na lokaciju trenutne. *Security* particija sadrži ključ koji se koristi u kombinaciji sa proizvodnim ključem za jednoznačnu identifikaciju svakog kontrolera. Poslednja u nizu je particija za smeštanje korisničkih podataka i konfiguracionih datoteka vezanih za aplikacije / pakete.

3.3 Mehanizam sigurnog ažuriranja programske podrške

Potreba da se trenutna programska podrška ažurira na novu ili vrati na staru verziju zahteva postojanje sigurnog mehanizma za takvu operaciju. Sam momenat zamene je izuzetno kritičan, jer gubitak napajanja u određenim trenucima samog procesa može dovesti do potpuno nefunkcionalnog uređaja, koji je nemoguće više koristiti. Iz tog razloga postoje dva pravca u ovom mehanizmu. Prvi, koji podrazumeva ažuriranje trenutne novom, koji je posledica korisničkog zahteva, i drugi, koji predstavlja proces vraćanja trenutne na fabričku verziju koja se paralelno upisuje prilikom proizvodnje na posebnu particiju, gde i ostaje bez izmena ceo radni vek uređaja.

Proces ažuriranja programske podrške se u osnovi zasniva na kopiranju sadržaja sa jednog para particija koje sadrže *Kernel* i *Rootfs* izvršne datoteke na trenutno par particija koji se trenutno izvršava.

Osnovni koraci u okviru ažuriranja programske podrške:

- Brisanje sadržaja particija trenutno izvršavane programske podrške
- Kopiranja sadržaja sa particija fabričke ili nove verzije programske podrške u zavisnosti od zahteva na particije trenutno izvršavane
- Brisanje sadržaja particija sa kojih je kopiran sadržaj (Napomena: samo u slučaju ažuriranja na novu verziju!)
- Provera ispravnosti novog sadržaja i pokušaj učitavanja u radnu memoriju

Kompletan proces ažuriranja u oba slučaja se odvija u okviru prvog programa kontrolera, *U-boot*-a. Potrebno je dodati podršku za operacije kopiranja sadržaja i brisanja sadržaja sa specificiranih particija, odnosno na ciljane particije, učitavanja sadržaja u radnu memoriju kao i proveru validnosti date programske podrške pomoću kontrolnih suma. Ceo algoritam provere zahteva za ažuriranjem treba ugnezditi u postojeću liniju redovnog učitavanja glavne izvršne datoteke programske podrške, pre nego što se učita u radnu memoriju. Slika 7.1 u okviru dodatka na kraju radu daje tok potprograma *U-boot*-a, koji ima funkciju sigurnog ažuriranja programske podrške na kontroleru DOC400.

4. Programsko rešenje

Poglavlje programskog rešenja daje uvid u implementaciju sistemske programske podrške za kontroler DOC400. Podnaslovi su približno istog redosleda i strukture kao kroz poglavlje koncepta rešenja, samo se sadržaj svakog upotpunjuje sa konkretnim detaljima realizacije. Naslovi se tiču integracije paketa, modifikacije konfiguracionih datoteka za platformu, reorganizacije *NAND* memorijske mape, mehanizma za sigurno ažuriranje programske podrške i na kraju opisa korišćenog sistema datoteka.

4.1 Integracija paketa

Izvorni kod svakog od paketa već postoji na zasebnim repozitorijumima. Implementirana je setenv.sh *shell* skripta, koja izvorni kod svakog od paketa preuzima sa repozitorijuma i smešta na predefinisanu lokaciju **package/oblo/naziv_paketa/files/** unutar *QSDK* okruženja. Potrebno je za svaki od paketa napisati samo *OpenWrt make* datoteku kako bi se predstavio ostatku sistema kroz sistem paketa. Primer koda skripte za preuzimanje izvornog koda paketa sa udaljenog repozitorijuma:

```
1.  ## Clone projects and put them in dedicated directories
2.  if [ ! -d $TOP/qsdk/package/oblo/ohm/files  ]; then
3.      print_msg "Clone of OHM repo.."
4.      cd qsdk/package/oblo/ohm
5.      git clone ssh://$USER@gerrit.rt-rk.com:29418/oblo_ohm files
6.      cd files
7.      scp -p -P 29418 $USER@gerrit.rt-rk.com:hooks/commit-msg .git/hooks/
8.      chmod u+x .git/hooks/commit-msg
9.      git clone ssh://$USER@gerrit.rt-rk.com:29418/oblo_ohm_rtrk csm
10.     cd csm
11.     scp -p -P 29418 $USER@gerrit.rt-rk.com:hooks/commit-msg .git/hooks/
12.     chmod u+x .git/hooks/commit-msg
13. fi
```

Slika 4.1: Primer preuzimanja izvornog koda paketa sa udaljenog repozitorijuma

4.1.1 Paket POCO C++ biblioteke

Izvorni kod POCO biblioteka se smešta u direktorijum **package/libs/poco**. Deklarativan opis POCO C++ biblioteke je prikazan kroz segment *make* datoteke za paket libpoco i ukazuje na lokaciju prevođenja izvornog koda, verziju biblioteke, kojoj kategoriji pripada dati paket kao i međuzavisnost prema drugim paketima. Deklarativan deo paketa *libpoco*:

```
1.  ## OpenWrt Makefile for POCO C++ Library
2.
3.  include $(TOPDIR)/rules.mk
4.
5.  PKG_NAME:=libpoco
6.  PKG_VERSION:=1.6.0
7.  PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)
8.  PKG_INSTALL:=1
9.
10. include $(INCLUDE_DIR)/package.mk
11.
12. define Package/libpoco
13.   NAME:=libpoco
14.   SECTION:=libs
15.   CATEGORY:=Libraries
16.   TITLE:=Open source C++ class libraries
17.   URL:=http://www.pocoproject.org/
18.   DEPENDS:=+libstdcpp +libpthread +librt +libopenssl
19. endef
```

Slika 4.2: Deklarativan deo libpoco *Makefile* datoteke

Pravila prevođenja POCO biblioteke se podešavaju pomoću *configure* alata, pri čemu se argumenti kroz OpenWrt *make* datoteku prosleđuju na način prikazan ispod. Promenljiva *CONFIGURE_ARGS* se nalazi u okviru datoteke *package.mk*, potrebno je samo izvršiti redefinisanje vrednosti.

```
1.  CONFIGURE_ARGS = \
2.      --config=MIPS-Linux \
3.      --shared \
4.      --no-samples \
5.      --no-tests \
6.      --omit=Data/ODBC,Data/MySQL,PageCompiler \
7.      --prefix=/_INSTALL \
8.      --include-path=$(STAGING_DIR)/usr/include \
9.      --library-path=$(STAGING_DIR)/usr/lib
```

Slika 4.3: Konfigurisanje POCO biblioteke kroz *OpenWrt Makefile*

Definicije **Build/InstallDev** i **Package/libpoco/install** se tiču pravila koja se odnose na to kako se postupa sa izlaznim datotekama procesa prevođenja ovog paketa, u ovom slučaju statičkim i dinamičkim bibliotekama. Prva definicija ukazuje na proces kopiranja biblioteka na

predefinisanu lokaciju u okviru *QSDK* okruženja, kako bi se paketi koji se referenciraju na ovu biblioteku (libpoco), mogli povezati sa njom. Druga definicija se koristi pri formiranju libpoco.ipk arhive, i definiše koje datoteke će se naći u datoj arhivi, i na kojim lokacijama će se raspakovati prilikom manuelne instalacije datog paketa na ciljanoj platformi putem *OPKG* menadžera. Takođe, u slučaju da paket treba da završi inicijalno u okviru programske podrške, data definicija ima istu namenu. Koriste se tri makro izraza *INSTALL_DIR*, *CP* i *1*, sa funkcijama kreiranja date lokacije u slučaju da ne postoji, rekurzivnog kopiranja sadržaja i oznake /root direktorijuma platforme, respektivno. Prikaz definicija u nastavku.

```
1.  define Build/InstallDev
2.      $(INSTALL_DIR) $(1)/usr/include
3.      $(INSTALL_DIR) $(1)/usr/lib
4.
5.      $(CP) $(PKG_INSTALL_DIR)/_INSTALL/include/Poco $(1)/usr/include/
6.      $(CP) $(PKG_INSTALL_DIR)/_INSTALL/lib/libPoco{Crypto,Data,DataSQLite,\
7.              Foundation,JSON,MongoDB,Net,NetSSL,Util,XML,Zip}.so* $(1)/usr/lib/
8.  endef
9.
10. define Package/libpoco/install
11.     $(INSTALL_DIR) $(1)/usr/lib
12.     $(CP) $(PKG_INSTALL_DIR)/_INSTALL/lib/libPoco{Crypto,Data,DataSQLite, \
13.             Foundation,JSON,MongoDB,Net,NetSSL,Util,XML,Zip}.so* \ $(1)/usr/lib/
14. endef
```

Slika 4.4: Definisanje pravila kopiranja prevedenog sadržaja libpoco paketa

4.1.2 Paketi OSM, OMB i OHM i uslužni paket UPDMNGR

Deklarativni deo *make* datoteka sva tri paketa ove kategorije imaju istu strukturu kao i slučaju *libpoco* paketa, sa konkretnim vrednostima za svaki iz kategorije. U dodatku na kraju rada se nalazi primer jedne kompletne *OpenWrt make* datoteke, dok se ovde samo ističu specifične stvari.

UPDMNGR predstavlja uslužnu aplikaciju prevedenu u format paketa. Ima dvostruku ulogu na sistemu ciljane platforme. Prva je vezana za mehanizam ažuriranja programske podrške dok se druga odnosi na uslugu dobavljanja *serial* i *security* ključeva koji se koriste interno od strane drugih aplikacija na sistemu kao i eksterno radi jednoznačne identifikacije samog kontrolera.

4.1.2.1 OBLO System Manager (OSM)

Ova aplikacije ima dvostruku ulogu u sistemu ciljane platforme. Prva i osnovna funkcija koju ima je uloga nadgledanja rada drugih aplikacija i upravljanja resursima platforme. Međutim, zbog slojevite i dobro projektovane arhitekture aplikacije, uspešno je iskorišćena kao osnova za aplikaciju u domenu testiranja kontrolera prilikom fabričkog programiranja kontrolera. Iz tog razloga se uvodi mehanizam različitog konfigurisanja aplikacije u zavisnosti od namene.

To se postiže uvođenjem podmenija za dati paket, i pomoćne datoteke *Config.in* koja nosi informacije o konfiguracionim parametrima, čija stanja/vrednosti mogu da se iskoriste da bi se određeni delovi izvornog koda isključili i uključili u proces prevođenja. Primer kako se jednim logičkim parametrom bira odabir u kom režimu će se paket prevoditi dat je u delu koda ispod na slici 4.5.

```
1.  if PACKAGE_osm
2.
3.  config FTA
4.      bool "Factory Test Application"
5.      default n
6.      help
7.          This option is used for building OSM extended with
8.          support for Factory testing of DOC400.
9.  endif
```

Slika 4.5: Podmeni za podršku dodatnih opcija OSM paketa

U okviru podmenija paketa *osm* se uključivanjem FTA parametra u praznu listu argumenata *Cmake*-a dodaje nova definicija –DFTA=ON. Ta lista se prilikom poziva cmake alata prosleđuje ulaznoj *cmake* datoteci koja može na osnovu prosleđene definicije, selektivno uključiti odnosno isključiti delove izvornog koda.

```
1.  CMAKE_FLAG_LIST:=
2.
3.  ## Configuration for Factory Test Application (FTA)
4.  define Package/osm/config
5.      source "$(SOURCE)/Config.in"
6.  endef
7.
8.  ifeq ($(CONFIG_FTA),y)
9.      CMAKE_FLAG_LIST+="-DFTA=ON"
10. endif
11.
12. define Build/Configure
13.      cmake $(PKG_BUILD_DIR)/CMakeLists.txt -DCMAKE_BUILD_TYPE=Release \
14.            -DCMAKE_TOOLCHAIN_FILE=Toolchain-doc_400.cmake $(CMAKE_FLAG_LIST)
15. endef
```

Slika 4.6: Poziv *Cmake* alata kroz *OpenWrt Makefile*

4.1.2.2OBLO Message Broker (OMB)

U okviru ovog paketa su objedinjena dva paketa, *omb* koji je centralna aplikacija za razmenu poruka u okviru sistema i *liboblomessenger*, pridružena biblioteka koju koriste druge aplikacije prilikom komunikacije sa glavnom. Dva deklarativna segmenta u okviru iste *OpenWrt make* datoteke je na slici 4.7.

```
define Package/liboblomessenger
  NAME:=liboblomessenger
  SECTION:=libs
  CATEGORY:=Libraries
  TITLE:=Oblo Message Broker library for clients
  DEPENDS:=+libstdcpp +libpthread +libdl +libpoco +libuuid
endef

define Package/omb
  NAME:=omb
  SECTION:=Home Automation
  CATEGORY:=Oblo
  TITLE:=Oblo Message Broker
  DEPENDS:=+libstdcpp +libpthread +libdl +libpoco +libuuid +liboblomessenger
endef
```

Slika 4.7: Dve deklaracije paketa u okviru jedne *OpenWrt Makefile* datoteke

4.2 Konfiguraciona datoteka za platformu DOC400

Kako se DOC400 kontroler kao i *CUS531* razvojna ploča zasniva na QCA4531 *SoC*, moguće je preuzeti deo konfiguracionih podataka i koristeći istu spregu dodati podršku za dodatne hardverske komponente.

Prvo je urađeno podešavanje GPIO konfiguracije. Dodata je podrška za dva tastera koristeći strukturu *gpio_keys_button* kojom se daje opis parametara kao što je perioda provere promene stanja, tekstualni opis namene tastera kao i dodela numeričkog broja istom pomoću kojeg se dobija obaveštenje o promeni stanja konkretnog tastera na nivou korisničkih aplikacija. Sprega je omogućena kroz mehanizam poznat kao *Hotplug*, koji za sve definisane tastere vrši periodičnu proveru njihovog stanja, i na svaku promenu šalje u sistem poruku o datom događaju, na koje će korisnička aplikacija ukoliko se registrovala dobiti obaveštenje. Slika 4.8 daje prikaz koda za definisanje dva tastera kontrolera.

```
static struct gpio_keys_button doc400_gpio_keys[] __initdata = {
    {
        .desc       = "Front button",
        .type       = EV_KEY,
        .code       = BTN_0,
        .debounce_interval = DOC400_KEYS_DEBOUNCE_INTERVAL,
        .gpio       = DOC400_GPIO_BTN_FRONT,
        .active_low = 1,
    },
    {
        .desc       = "Reset button",
        .type       = EV_KEY,
        .code       = BTN_1,
        .debounce_interval = DOC400_KEYS_DEBOUNCE_INTERVAL,
        .gpio       = DOC400_GPIO_BTN_FRONT,
        .active_low = 1,
    }
};
```

Slika 4.8: Konfiguracija tastera kontrolera

Potrebno je osloboditi *GPIO* prolaze koji su namenjeni za ova dva tastera. JTAG sprega kao jedne od podrazumevanih prolaza koristi one pod rednim brojem 2 i 3. Podešavanje funkcija nad određenim prolazima kao i registracija prethodno definisane strukture tastera se nalazi u kodu ispod, na slici 4.9.

```
1.  static void __init doc400_gpio_setup(void)
2.  {
3.      /* Release GPIO pins (1,2,3,4) asigned to JTAG */
4.      ath79_gpio_function_enable(DOC400_JTAG_DISABLE);
5.
6.      /* Set GPIO direction of Buttons to IN */
7.      ath79_gpio_output_select(DOC400_GPIO_BTN_FRONT, 0x1);
8.      ath79_gpio_output_select(DOC400_GPIO_BTN_FRONT, 0x1);
9.
10.
11.     ath79_register_gpio_keys_polled(-1, DOC400_KEYS_POLL_INTERVAL,
12.                     ARRAY_SIZE(doc400_gpio_keys),
13.                     doc400_gpio_keys);
14. }
```

Slika 4.9: Podešavanje *GPIO* prolaza

Takođe je dodata podrška za dva tipa fleš memorija, NOR i NAND. To je omogućeno prvo hardverskim sposobnostima *QC4531 SoC*, koji poseduje dva *chip select* signala. Struktura koja se koristi u inicijalizacionoj sekvenci i opisuje organizaciju fleš memorija u sistemu data je na slici 4.10. Neki od parametara su maksimalna učestanost na kojoj radi memorija kao i kojem rukovaocu je data memorija pridružena.

```
1.  static struct spi_board_info doc400_spi_info[] = {
2.      {
3.          .bus_num    = 0,
4.          .chip_select    = 0,
5.          .max_speed_hz   = 25000000,
6.          .modalias   = "m25p80",
7.          .controller_data = &doc400_spi0_cdata,
8.          .platform_data  = NULL,
9.      },
10.     {
11.         .bus_num    = 0,
12.         .chip_select    = 1,
13.         .max_speed_hz   = 25000000,
14.         .modalias   = "ath79-spinand",
15.         .controller_data = &doc400_spi1_cdata,
16.         .platform_data  = NULL,
17.     }
18. };
```

Slika 4.10: Izgled definicija *NAND* i *NOR* fleš memorije

Ostatak konfiguracije vezan za mrežna podešavanje se u većini preklapa sa konfiguracijom za *CUS531* razvojnu ploču.

4.3 Memorijska mapa *NAND* fleš memorije

Izmena strukture memorijskih regiona *NAND* fleš memorije, tačnije menjanje broja i veličine particija, ima za cilj da se podrži mehanizam sigurnog ažuriranja programske podrške kao i mogućnost vraćanja na fabričku, podrazumevanu.

Kako bi postojala sinhronizacija između *U-boot*-a i *Linux*-u, potrebno je da obe strane imaju istu sliku o memorijskoj mapi. To se postiže čuvanjem informacije u okviru particije *NOR* memorije pod imenom *u-boot-env*, kojoj *u-boot* može uvek da pristupi, a *Linux*-u se dostavlja kroz argumente (engl. *Bootargs*) prilikom učitavanja i pokretanja. Na taj način se održava uvek sinhronizacija između dve strane.

Tekstualna datoteka sadrži pored argumenata koji se prosleđuju *Linux* operativnom sistemu prilikom pokretanja i informacije koje koristi sam *U-boot*, kao što su parametri serijske komunikacije i predefinisane komande koje se koriste prilikom programiranja particija kontrolera. Deo tekstualne datoteke koji se tiče samo memorijske mape, a koja je deo promenljive koja predstavlja ulazne argumente *Linux*-a data je na slici ispod.

```
1. bootargs=board=CUS531-NAND console=ttyS0,115200 ubi.mtd=5,2048 ubi.mtd=11,2048
2. root=/dev/mtdblock13
3. mtdparts=spi0.0:256k(u-boot)ro,64k(u-boot-env),128k(reserved),64k(art);
4. spi0.1:2m(kernel),22m(rootfs),2m(gold-kernel),22m(gold-rootfs),
5. 2m(upgrade-kernel),22m(upgrade-rootfs),128k(security),57216k(data),
6. 28m@0x0(firmware)
7. rootfstype=squashfs,jffs2 noinitrd
```

Slika 4.11: Definicija memorijska mape u okviru *u-boot-env* datoteke

Na *u-boot-env* particiju je upisana binarna datoteka koja sadrži parove formata ime_promenljive = vrednost, a koja se generiše pomoću alate *mkenvimage*, koja prevodi tekstualnu datoteku u format adekvatan za upisivanje na particiju. Alat smo integrisali u okruženje i prevodi se zajedno sa svim *host* alatima. Jednostavna skripta za generisanje binarne datoteke sa parametrima koji označavaju da će format biti binarna datoteka, poravnanje se radi sa nulama i veličina datoteke je 64KiB kolika je i sama particija.

```
1. #Generates image for u-boot environment partition
2. $QSDK_HOST_BIN/mkenvimage -b -p 0x00 -s 0x10000 -o ./uboot-env.bin ./uboot-
   env.txt
```

Slika 4.12: Skripta za generisanje uboot-env binarne datoteke

4.4 Optimizacija upisa na particiju podataka (*Data*)

Sistem datoteka koji je odabran da se koristi u okviru ove particije je *UBIFS*. Razlozi su izuzetna skalabilnost po pitanju konstantnog vremena povezivanja (engl. *Mount*) i pristupa nezavisno od popunjenosti ili veličine date particije.

Međutim, bez obzira na odabir sistema datoteka, u svakom slučaju se mora voditi računa o broju upisa na fleš memoriju. Uobičajeno je da se upisi vrše na veličinu jedne stranice fizičkog bloka fleš memorije, pa se koristi bafer u radnoj memoriji koji akumulira podatke koje je potrebno upisati, a upis i pražnjenje sa radi tek kad se popuni. Problem koji se javlja pri ovakvoj implementaciji jeste pojava nekonzistentnog stanja sistema usled iznenadnih gubitaka napajanja.

Analizirajući potrebe programa koji koriste ovu particiju da se izvrši deljenje na dve logičke jedinice (engl. *Volume)*. Prva je jedinicu za bitne konfiguracione promene, koja se povezuje u sinhronom režimu rada i time podaci direktno upisuju na memoriju bez obzira na veličinu, i druga koja ima ulogu dnevničke datoteke (engl. *Log*), u koju se upisuje pri dovoljnoj količini podataka u baferu. Konfiguraciona datoteka *UBIFS*-a za *Data* particiju je na slici 4.13.

```
1.  [config_volume]
2.  mode=ubi
3.  image=config.ubifs
4.  vol_id=1
5.  vol_type=dynamic
6.  vol_name=configuration
7.  vol_flags=autoresize
8.
9.  [log_volume]
10. mode=ubi
11. image=log.ubifs
12. vol_id=2
13. vol_type=dynamic
14. vol_name=log
15. vol_size=5MiB
```

Slika 4.13: *UBIFS* konfiguraciona datoteka za *Data* particiju

4.5 Mehanizam ažuriranja programske podrške

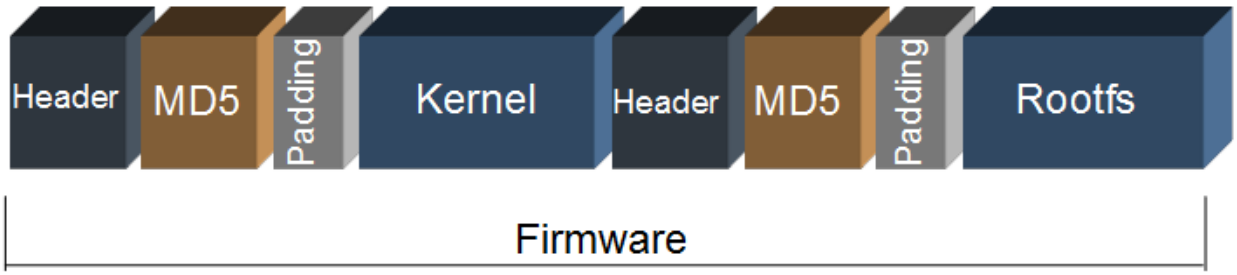
Ažuriranje programske podrške je procedura kojom se trenutno izvršavana zamenjuje novom ili fabričkom. To su ujedno i dva moguća pravca zamene, pri čemu se u prvom slučaju radi o promeni na verziju koja treba da donese neke novu funkcionalnost, dok se u drugom vrši vraćanje na ispravnu i proverenu fabričku programsku podršku. Pod pojmom programske podrške su objedinjene dve izvršne datoteke, ***Kernel*** i ***Root Filesystem***.

Mehanizam sigurnog ažuriranja je integrisan u postojeću implementaciju *U-boot*-a u formi biblioteke koja je nazvana lib_golden, a koja se prevodi i povezuje statički. Biblioteka implementira funkcionalnosti vezane za rad sa *NAND* memorijom, mehanizme dešifrovanja sadržaja kao i sistem provere validnosti dešifrovanih i tako upisanih izvršnih datoteka na memorijske particije. Takođe, dodata je podrška za rad sa uređajima vezanih putem *I2C* magistrale. U ovom slučaju, koristi se *I2C GPIO Expander*, preko kojeg se vrši upravljanje *RGB* indikacijom u cilju oslikavanja trenutnog stanja kontrolera tokom procesa ažuriranja.

4.5.1 Šifrovanje izvršnih datoteka programske podrške i izgled zaglavlja

Uloga šifrovanja u ovakvom procesu je višeznačna, pre svega u zaštiti krajnjeg korisnik od zloupotrebe kontrolera od treće strane. Nova programska podrška se na *cloud* serveru šifruje sa ključem koji je jedinstven za svaki kontroler i nalazi se na samom uređaju, a koji se dostavlja na server prilikom prve registracije. Fabrička programska podrška se šifruje ključem koji je zajednički za jednu proizvodnu seriju kontrolera, i za svaku novu seriju se generiše novi ključ.

Program koji radi spajanje *Kernel* i *Rootfs* izvršnih datoteka u jedinstvenu datoteku programske podrške, dodaje i neophodna poravnanja (engl. *Padding*), računanje MD5 sume kompletnog sadržaja, šifrovanje i na kraju dodavanje zaglavlja koje je formirano od dve celobrojne vrednosti i nosi informaciju o veličini izvršne datoteke i količini dodatog poravnanja. Programska podrška sa svim segmentima predstavljena je grafički na slici 4.14.



Slika 4.14: Struktura datoteke programske podrške

U slučaju ažuriranja na noviju verziju programske podrške, nakon što je preuzeta datoteka sa servera, uslužna aplikacija vrši deljenje iste na dva dela koji sadrže po četiri segmenta (*Header*, *MD5*, *Padding*, *Kernel*/*Rootfs*) i upisuje ih na predefinisane *MTD* particije sa kojih se vrši čitanje i nastavak procesa u okviru *U-boot*-a.

4.5.2 Podrška za kontrolu tastera i svetlosnu indikaciju

Prvo je potrebno korišćenjem namenskih *GPIO* registara QCA4531 *SoC* i sprege za pristup istim podesiti pravilnu konfiguraciju *GPIO* prolaza dodeljenih tasterima i *I2C Expander*-u koji upravlja RGB indikacijom. Koriste se dve funkcije za menjanje stanja registara na osnovu maske. Vrednost ’1’ na mestu određenog bit-a maske u slučaju *set* funkcije postavlja dati bit registra na logičku jedinicu, dok se u slučaju poziva *clear* funkcije dati bit registra postavlja na logičku nulu.

- `ath_reg_rmw_set (unsigned char _register, unsigned int _mask)`
- `ath_reg_rmw_clear (unsigned char _register, unsigned int _mask`

Redosled bita u okviru registra *GPIO_OE_ADDRESS* se direktno preslikava na brojeve prolaza. Postavljanjem bita na ’1’ određuje smer njemu pridruženog prolaza da bude „ulaz“ (engl. *Input*). U slučaju konfiguracije prolaza namenjenih signalima *SDA* i *SCL*, mora se prvo osloboditi registar funkcije prolaza (*GPIO_OUT_FUNCTION4_ADDRESS*) u slučaju da je neka interna logika vezana za dati prolaz, koja bi eventualno pravila konflikte pri upravljanju pomenutim signalima *I2C* komunikacije. Na kraju je potrebno podesiti *I2C* magistralu da operira na određenoj frekvenciji. Funkcija u okviru lib_golden biblioteke za podešavanje registara kako bi se obezbedila sprega sa tasterima i RGB indikacijom, kao i funkcija za vraćanje registara na početno stanje nakon što kontrola više nije date su na slikama 4.15 i 4.16, respektivno.

```
1.  void gdm_setup_registers(void)
2.  {
3.      /* Set GPIO 2 and GPIO 3 as input for detection of Golden procedure */
4.      ath_reg_rmw_set(GPIO_OE_ADDRESS, (1 << 2)); /* Back/Reset Button */
5.      ath_reg_rmw_set(GPIO_OE_ADDRESS, (1 << 3)); /* Front Button */
6.
7.      /* Set SDA and SCL pins as standard GPIO in/out */
8.      ath_reg_rmw_clear(GPIO_OE_ADDRESS, (1 << 17)); /* SDA pin for I2C */
9.      ath_reg_rmw_clear(GPIO_OUT_FUNCTION4_ADDRESS, \
10.                       GPIO_OUT_FUNCTION4_ENABLE_GPIO_17_MASK);
11.     ath_reg_rmw_set(GPIO_OUT_FUNCTION4_ADDRESS, \
12.                     GPIO_OUT_FUNCTION4_ENABLE_GPIO_17_SET(0x80));
13.
14.     uchar val = PCA_OUTPUT_REG_MASK;
15.     i2c_init(CFG_I2C_SPEED, CFG_I2C_SLAVE);
16.     gdm_change_led_state(TURNED_OFF_RGB);
17.     i2c_write(PCA_SLAVE_ADDRRESS, PCA_IO_DIRECTION_REG, I2C_ADDR_NUM_OF_BYTES, \
18.               &val, I2C_RW_NUM_OF_BYTES);
19. }
```

Slika 4.15: Funkcija podešavanja *GPIO* registara

```
1.  void gdm_restore_registers(void)
2.  {
3.      /* Clear GPIO 2 and GPIO 3 used as input for detection of Golden procedure */
4.      ath_reg_rmw_clear(GPIO_OE_ADDRESS, (1 << 2)); /* Back/Reset Button */
5.      ath_reg_rmw_clear(GPIO_OE_ADDRESS, (1 << 3)); /* Front Button */
6.
7.      /* Set SDA GPIO pin, need to be input */
8.      ath_reg_rmw_set(GPIO_OE_ADDRESS, (1 << 17)); /* SDA pin for I2C */
9.      /* Clear function register for SDA pin */
10.     ath_reg_rmw_clear(GPIO_OUT_FUNCTION4_ADDRESS, \
11.                       GPIO_OUT_FUNCTION4_ENABLE_GPIO_17_MASK);
12.     ath_reg_rmw_set(GPIO_OUT_FUNCTION4_ADDRESS, \
13.                     GPIO_OUT_FUNCTION4_ENABLE_GPIO_17_SET(0x01));
14. }
```

Slika 4.16: Funkcija za vraćanje stanja *GPIO* registara

Iz razloga što ne postoji hardverska podrška za *I2C* magistralu, realizovana je softverska implementacija komunikacije, poznata pod nazivom *Bit-Banging*. Korišćen je pseudo C kod koji

je u potpunosti baziran na makro funkcijama, a iza kojih treba da se nalazi specifična realizacija upravljanja GPIO prolaza za datu platformu. Na primer, SCL linija treba da simulira takt komunikacije, a makro funkcija I2C_SCL(bit) postavlja vrednost ’0’ ili ’1’ na izlaz GPIO prolaza. Prikaz redefinisanih makro funkcija dat je na slici 4.17.

```
1.  #define I2C_TRISTATE    ath_reg_rmw_set(GPIO_OE_ADDRESS, SDA_GPIO_MASK)
2.
3.  #define I2C_ACTIVE      ath_reg_rmw_clear(GPIO_OE_ADDRESS, SDA_GPIO_MASK)
4.
5.  #define I2C_READ        ((ath_reg_rd(GPIO_IN_ADDRESS) & SDA_GPIO_MASK) != 0)
6.
7.  #define I2C_SDA(bit)    if(bit) ath_reg_rmw_set(GPIO_OUT_ADDRESS, SDA_GPIO_MASK);\
8.                          else ath_reg_rmw_clear(GPIO_OUT_ADDRESS, SDA_GPIO_MASK)
9.
10. #define I2C_SCL(bit)    if(bit) ath_reg_rmw_set(GPIO_OUT_ADDRESS, SCL_GPIO_MASK);\
11.                         else ath_reg_rmw_clear(GPIO_OUT_ADDRESS, SCL_GPIO_MASK)
12.
13. #define I2C_DELAY       udelay(2)
```

Slika 4.17: Redefinisanje *I2C* makro funkcija

Zatim se provera stanja tastera svodi na čitanje određenog bita u okviru registra ulaznih prolaza (*GPIO_IN_ADDRESS*), koji se odnosi na dati taster, dok se promena *RGB* indikacije jednostavno kontroliše upisom u izlazni registar *I2C GPIO Expander*-a. Čitanje stanja jednog od tastera i funkcija za menjanje boje *RGB* indikacije su prikazani na slikama 4.18 i 4.19.

```
1.  int gdm_check_btn_state()
2.  {
3.      if( (ath_reg_rd((GPIO_IN_ADDRESS)) & RESET_BTN) != 0x0 )
4.      {
5.          printf("%s:reset button is not pressed\n", __func__);
6.          return 1;
7.      }
8.
9.      /* Wait for user confirmation */
10.     printf("%s:reset button is pressed\n", __func__);
11.     gdm_wait_user_conf();
12.     return 0;
13. }
```

Slika 4.18: Provera stanja tastera

```
1.  inline void gdm_change_led_state(uchar color)
2.  {
3.      uchar value = color | PCA_OUTPUT_REG_MASK;
4.      i2c_write(PCA_SLAVE_ADDRRESS, PCA_OUTPUT_REG,
5.                  I2C_ADDR_NUM_OF_BYTES, &value, I2C_RW_NUM_OF_BYTES);
6.  }
```

Slika 4.19: Menjanje *RGB* svetlosne indikacije

4.5.3 Implementacija sigurnog mehanizma ažuriranja programske podrške

Algoritam odlučivanja o eventualnom ažuriranju programske podrške je integrisan pre učitavanja trenutnog u radnu memoriju, kako bi se izvršila zamena ukoliko za takvu operaciju postoji zahtev. Uključivanje datog algoritma u tok redovnog izvršavanja *u-boot* programa se kontroliše definisanjem zastavice (engl. *Flag*) CONFIG_BOARD_DOC400, što za posledicu ima prevođenje biblioteke lib_golden i integraciju datog algoritma. Javne funkcije biblioteke su date ispod u tabeli 4.1 sa kratkim opisom namene.

Ime funkcije	**Opis**	**Argumenti**	**Povratna vrednost**
gdm_setup_registers	Podešavanje registara u cilju podrške za kontrolu RGB indikacije i tastera	-	-
gdm_restore_registers	Vraćanje stanja registara na zatečeno stanje pre ulaska u proceduru ažuriranja	-	-
gdm_check_btn_state	Provera stanja tastera za uslov početka ažuriranja PP[1]	-	0 u slučaju potvrde uslova, suprotno 1
gdm_check_for_user_upgrade	Provera da li postoji nova PP	-	0 u slučaju potvrde uslova, suprotno 1
gdm_check_factory_request	Provera da li je postoji zahtev za vraćanje PP na fabričku verziju	-	0 u slučaju potvrde uslova, suprotno 1
gdm_start_user_upgrade	Pokretanje procedure ažuriranja PP na noviju verziju	-	0 u slučaju uspešnog ažuriranja, suprotno 1

[1] PP – Programska podrška kontrolera (engl. *Firmware*)

gdm_start_factory_upgrade	Pokretanje procedure vraćanja PP na fabričku verziju	-	0 u slučaju uspešnog ažuriranja, suprotno 1
gdm_change_led_state	Promena stanja RGB indikacije u zavisnosti od ulaznog argumenta	Boja RGB indikacije	-
gdm_invalid_image_state	Neispravna programska podrška, daje svetlosnu indikaciju o stanju	-	-

Tabela 4.1: Javne funkcije biblioteke lib_golden

Algoritam započinje postavljanjem *GPIO* registara u adekvatno stanje kako bi se obezbedila adekvatna kontrola *RGB* indikacije i mogućnost dobavljanja trenutnog stanja tastera. Prvo se proverava da li postoji korisnički zahtev vraćanja na fabričku verziju, zatim i da li stanje tastera ukazuje na isto. Ukoliko postoji makar jedan zahtev, prelazi se na proceduru vraćanja na fabričku verziju programske podrške. U suprotnom se prelazi na proceduru ažuriranja na noviju verziju, ukoliko zahtev za to postoji. Nepostojanjem zahteva i u ovom slučaju se prelazi na učitavanje trenutne programske podrške u radnu memoriju. Poslednji korak se izvršava i u slučaju da je postojalo ažuriranje da li na noviju ili fabričku verziju. U slučaju da je datoteka neispravna ulazi se u beskonačnu petlju koja svetlosnom indikacijom ukazuje na trenutno stanje kontrolera. To je znak da je potrebno ponovo pokrenuti kontroler i predefinisanom kombinacijom pritisaka tastera izvršiti vraćanje na fabričku verziju. Programski kod algoritma je dat u dodatku, Slika 7.2.

4.5.3.1 Sprega *Linux* i *U-boot* prostora

Zahtev za fabričko ažuriranje koji je potekao iz *Linux*-a se mora na neki način proslediti do samog mehanizma koji se nalazi u *U-boot* prostoru. To je moguće uraditi posredstvom radne memorije zato što se prilikom ponovnog pokretanja kontrolera, misleći na „*soft reset*“, ne gubi napajanje pa i sadržaj upisan u radnu memoriju takođe ostaje nepromenjen.

Potrebno je izdvojiti region radne memorije koji neće biti dostupan memorijskom kontroleru u okviru *Linux*-a kako ne bi mogao da dodeli taj deo memorije nekom procesu. Uzeto je 4KiB, što odgovara poravnanju od jedne stranice *RAM*-a, sa vrha radne memorije kao deljeni region. Pristup fizičkim adresama radne memorije iz *Linux*-a je moguć putem */dev/mem* sprege [11] koji predstavlja kompletan prostor radne memorije. Način korišćenja i primer upisa zaglavlja koji nosi zahtev za vraćanje na fabričku programsku podršku dat je na slici 4.20.

```
{
    int fd;
    char *reserved_memory;
    factory_upgrade_hdr_t hdr;

    hdr.check_tag = CHECK_TAG;
    hdr.version = HEADER_VERSION;
    hdr.factory = SET_FACTORY_FLAG;

    fd = open("/dev/mem",O_RDWR);
    if(fd < 0)
        return 1;

    printf("Open /dev/mem.\n");
    reserved_memory = (char *) mmap(0, 4*1024, PROT_READ|PROT_WRITE,
              MAP_SHARED, fd, START_RESERVED_DDR_REGION);

    memcpy(reserved_memory, &hdr, sizeof(factory_upgrade_hdr_t));

    return 0;
}
```

Slika 4.20: Pristup radnoj memoriji putem */dev/mem* sprege

Definisan je zatim izgled zaglavlja koji poštuju obe strane kako bi se održala sinhronizacija. Zaglavlje je jednostavno, sadrži tri polja sa celobrojnim vrednostima. Prvo služi za proveru validnosti ostalog sadržaja zaglavlja, ima predefinisanu vrednost i upisuje se prilikom upisa zahteva, u svim ostalim slučajevima ima nasumičnu vrednost; drugo polje je informativnog karaktera i nosi trenutno verziju zaglavlja; na kraju, indikator samog zahteva za ažuriranje.

Funkcije koja radi proveru zahteva za vraćanje na fabričku verziju programske podrške je prikazana na slici 4.21. Sa predefinisane lokacije *START_RESERVED_DDR_REGION* u okviru deljenog segmenta radne memorije se učitava sadržaj zaglavlja, vrši se provera polja validnosti i nakon toga i polje indikacije zahteva.

```
int gdm_check_factory_request()
{
    factory_upgrade_hdr_t hdr;
    memcpy((void*)&hdr, START_RESERVED_DDR_REGION, sizeof(hdr));

    if ( hdr.check_tag == CHECK_TAG)
    {
        if(hdr.factory == SET_FACTORY_FLAG)
        {
            printf("%s:Factory header:\nVersion : 0x%08x\tFactory : 0x%08x\n",
                    hdr.version, hdr.factory);
            return 0;
        }
        else
            printf("%s : No Factory request!\n", __func__);
    }
    else
    {
        printf("%s : Invalid factory header check tag.\n", __func__);
    }
    return 1;
}
```

Slika 4.21: Provera zahteva za fabričko ažuriranje programske podrške

5. Testovi i merenja

Poglavlje se tiče kvalifikacije rada sistema, što obuhvata merenja opterećenja kontrolera, vremena podizanja (engl. *Booting*) sistema u zavisnosti od načina prevođenja izvršnih datoteka kao uticaj odabira tipa sistema datoteka. Testiranje se isključivo odnosi na uticaj gubitka napajanja kontrolera u različitim momentima tokom ažuriranja programske podrške. Podrazumevano je da se testiranje i merenje sistema sprovelo nakon potvrđene ispravnosti kontrolera u celosti.

5.1 Upotreba radne i fleš memorije

Osnovno ograničenje sistema je u dostupnom delu radne memorije. To je ujedno i nedostatak sistema, jer prilikom punog opterećenja resursi memorije se svode na minimum, reda veličine ~2MB. Konkretno, prilikom procesa rada uslužne aplikacije *UPDMNGR* za izvršenje dela ažuriranja programske podrške aplikacija zaustavlja od strane sistema usled nedostatka radne memorije. Aplikacija je optimizovana da radi u dostupnim okvirima slobodne memorije, što za posledicu ima povećanje ukupnog vremena izvršenja jedne iteracije ažuriranja programske podrške. Tabela 5.1 daje uvid u stanje radne memorije pod opterećenjem tri memorijski najzahtevnija programa na sistemu.

Opterećenje	**OHM**	**OSM**	**OMB**	**Slobodna memorija**	**Zauzeta memorija**
Puno opterećenje	18333KB	7457KB	5906KB	2628KB	58816KB
Bez opterećenja	0KB	0KB	0KB	34324KB	27120KB

Tabela 5.1: Upotreba radne memorije od strane glavnih programa kontrolera

Što se tiče prostora slobodnog na fleš memoriji, situacija je približno na granicama iskorišćenja. *Rootfs* particija se sastoji iz dva dela. Prvog koji je moguće samo čitati (*/rom*), i drugog (/) koji je u suštini prostor za eventualna proširenja. Kao što se vidi iz Tabele 5.2, eventualno proširenje sistema datoteka novim paketima se mora uklopiti u prostor od ~3,8MB.

Dve logičke jedinice (*/mnt/config* i */mnt/log*) objedinjene kroz *Data* particiju su relativno prazne, ali ne treba smetnuti sa uma da se prilikom procesa ažuriranja programske podrške, datoteka nove programske podrške smešta upravo na */mnt/config* jedinicu pre upisa na stvarne lokacije u okviru particija namenjenih za dati proces. To znači sledeće: kada je datoteka nove programske podrške maksimalne veličine od 24MB (22MB+2MB), 58% logičke jedinice mora biti u datom momentu slobodno kako bi se mogao nesmetano izvršiti proces ažuriranja.

Sistem datoteka	**Veličina**	**Iskorišćeno**	**Dostupno**	**Iskorišćeno (%)**	**Tačka povezivanja**
/dev/root	14,3MB	14,3MB	0	100%	/rom
overlayfs:/overlay	4,2MB	444,0KB	3,8MB	10%	/
ubi1:/config	40,9MB	6,1MB	34,8MB	15%	/mnt/config
ubi1:/log	2,7MB	56,0KB	2,6MB	2%	/mnt/log

Tabela 5.2: Zauzeće *NAND* fleš memorije

U suštini, sistem dostiže svoj limit po pitanju integracije novih paketa (programa) ako je sudeći po slobodnom prostoru za proširenja takve vrste. Takođe, o na izgled slobodnom prostoru *Data* particije se mora voditi računa, i uvek imati u vidu da je potrebno 58% prostora rezervisati za potrebe mehanizma za ažuriranje programske podrške, što ostavlja približno 17MB za korišćenje od strane drugih aplikacija. Na kraju, što se tiče zauzeća radne memorije, pod punim opterećenjem sistem se nalazi na samim granicama.

5.2 Vreme učitavanja sistema (engl. *Boot time*)

Više faktora utiče na ukupno vreme potrebno za dobijanje korisnički upravljivog kontrolera od trenutka dovođenja napajanja na isti. Osnovni parametri koji imaju uticaja su:

- Način prevođenja operativnog sistema (*Debug/Release/Retail)*
- Količina programa koji moraju da se učitaju tokom podizanja *Linux* operativnog sistema
- Odabir sistema datoteka, koji direktno utiče na vreme pristupa
- Broj korisničkih/sistemskih programa koji se pokreću na samom početku

Konkretno vreme potrebno za učitavanje samo operativnog sistema, kao i vreme do kompletno funkcionalnog i korisnički upravljivog kontrolera u zavisnosti od nekih gore navedenih parametara data je u tabeli 5.1. Postoji vremenska razlika pri prvom pokretanju kontrolera, gde se vreme produžava zbog procesa kopiranja određenih datoteka sa *Rootfs* particije na particiju za podatke (*Data*). Merenje je obavljeno kombinujući tri parametra:

- Debug/Retail – način prevođenja, za potrebe testiranja i komercijalne upotrebe
- JFFS2/UBIFS – tip sistema datoteka na *Data* particiji
- Puna/Delimična podrška paketa – pod ovim se misli na stepen integrisane funkcionalnosti (*SSH*, *Telnet*, *Busybox* sa raznim dodatnim alatima itd.) u sistemu

Tip konfiguracije	**Učitan OS**	**Funkcionalan sistem**	**Funkcionalan sistem prvo pokretanje**
Debug / JFFS2/ Puna podrška	53	71	160
Debug / JFFS2 / Delimična podrška	50	65	158
Debug / UBIFS / Delimična podrška	49	60	95
Retail / UBIFS / Puna podrška	40	53	85
Retail / UBIFS / Delimična podrška	37	48	80

Tabela 5.3: Vremena učitavanja sistema

Vrednosti u tabeli su vremena izražena u sekundama. Kao što se vidi iz priloženog, najveći uticaj na vreme učitavanja operativnog sistema jeste način prevođenja, dok se vreme do funkcionalnog sistema prilikom prvog pokretanja znatno smanjuje pri promeni tipa sistema datoteke koji se koristi na particiji za podatke (*Data*). Važno napomene je i to, da je merenje sprovedeno nad *Data* particiji koja je bila 15% iskorišćena, pa bi sigurno vreme druge kolone raslo sa porastom iskorišćenja, ali samo u slučaju *JFFS2*-a. To je posledica zbog same prirode ovog tipa sistema datoteka, gde je vreme povezivanja (engl. *Mount*) linearno proporcionalno iskorišćenosti particije. Vreme povezivanja u slučaju *UBIFS*-a je konstantno, jedan od ključnih razloga zbog kojih je odabran.

5.3 Testiranje mehanizma za ažuriranje programske podrške

Mehanizam je potvrđen i proverena je njegova ispravnost u normalnom režimu rada, bez prekida napajanja u toku samog ažuriranja. Međutim, to nije dovoljno, pa je potrebno izvesti testove naglog gubitka napajanja u različitim momentima kako bi se proverila sigurnost mehanizma, i potvrdila činjenica da se u bilo kom slučaju trenutno neispravnog i nefunkcionalnog kontrolera može uraditi ažuriranja na fabričku verziju programske podrške i time uspostavio ponovo normalan režim rada kontrolera.

5.3.1 Prekidi u vidu gubitka napajanja

Ime testa:	**Prekid prilikom brisanja *Kernel/Rootfs* particije trenutno izvršavane programske podrške**
Opis testa:	Ovim testom treba pokazati da se usled ovakvog prekida, kontroler dovodi u neispravno stanje
Koraci:	1. Pokrenuti proces vraćanja na fabričku verziju programske podrške 2. Pratiti ispis programa na serijskoj konzoli 3. Prekinuti napajanje u trenutku brisanja trenutne *Kernel/Rootfs* particije
Očekivani rezultat:	Očekivano stanje je da je se RGB indikacijom prikaže neispravno stanje kontrolera
Rezultat testa:	**Uspešan**

Tabela 5.4: Prekid prilikom brisanja particija trenutne programske podrške

Ime testa:	**Prekid prilikom kopiranja particija nove programske podrške**
Opis testa:	Ovim testom treba pokazati da se usled ovakvog prekida, kontroler dovodi u neispravno stanje
Koraci:	1. Pokrenuti proces ažuriranja programske podrške 2. Pratiti ispis programa na serijskoj konzoli 3. Prekinuti napajanje u trenutku kopiranja nove *Kernel/Rootfs* particije
Očekivani rezultat:	Ukoliko se prekine u toku kopiranja *Kernel* particije, kontroler se nalazi u neispravnom stanju, što se oslikava *RGB* indikacijom, dok se u slučaju prekida kopiranja *Rootfs* particije kontroler dovodi u nekonzistentno stanje, gde RGB indikacija ukazuje na uspešan početak učitavanja programske podrške koji se neće nikada završiti
Rezultat testa:	**Uspešan**

Tabela 5.5: Prekid prilikom kopiranja particija nove programske podrške

5.3.2 Podmetanje programske podrške šifrovane pogrešnim ključem

Ime testa:	**Podmetanje programske podrške šifrovane pogrešnim ključem**
Opis testa:	Ovim testom treba pokazati da usled pokušaja podmetanja programske podrške koja nije namenjena za dati kontroler dolazi do odbijanja
Koraci:	1. Pokrenuti proces ažuriranja programske podrške
Očekivani rezultat:	Proces ažuriranja treba da bude prekinut usled neuspešne dešifrovanja datoteka programske podrške
Rezultat testa:	**Uspešan**

Tabela 5.6: Podmetanje programske podrške šifrovane pogrešnim ključem

5.3.3 Testiranje oštećene datoteke programske podrške

Ime testa:	**Provera usled pojave oštećene *Kernel* izvršne datoteka**
Opis testa:	Ovim testom treba pokazati da je moguće detektovati oštećenje *Kernel* datoteke programske podrške
Koraci:	1. Napraviti oštećenu *Kernel* datoteku u okviru programske podrške 2. Postaviti je na *cloud* server 3. Pokrenuti ažuriranje programske podrške kontrolera
Očekivani rezultat:	Nakon uspešnog procesa ažuriranja programske podrške, vrši se CRC32 provera *Kernel* izvršne datoteke pre učitavanja u radnu memoriju, koja treba da prekine dalje učitavanje usled različitih kontrolnih suma
Rezultat testa:	**Uspešan**

Tabela 5.7. Provera usled pojave oštećene *Kernel* izvršne datoteka

Ime testa:	**Provera usled pojave oštećene *Rootfs* izvršne datoteka**
Opis testa:	Ovim testom treba pokazati da je nemoguće detektovati oštećenje *Rootfs* datoteke programske podrške
Koraci:	1. Napraviti oštećenu *Rootfs* datoteku u okviru programske podrške 2. Postaviti je na *cloud* server 3. Pokrenuti ažuriranje programske podrške kontrolera
Očekivani rezultat:	Nakon uspešnog procesa ažuriranja programske podrške , u slučaju oštećenja *Rootfs* datoteke, dolazi do nekonzistentnog stanja zato što se nad ovom datotekom ne radi provera kontrolne sume
Rezultat testa:	**Uspešan**

Tabela 5.8: Provera usled pojave oštećene *Rootfs* izvršne datoteka

6. Zaključak

Zadatak ovog rada je bio da se realizuje sistemska programska podrška za kontroler DOC400, centralnog uređaja u okviru jednog sistema za automatizaciju kuća. Potrebno je bilo pre svega upoznati se sa arhitekturom platforme na kojoj se zasniva dati kontroler, *QSDK* okruženjem kao i različitim mehanizmima ispitivanja ispravnosti određenih segmenata rešenja. Takođe, bilo je potrebno upoznati principe rada jednog operativnog sistema, *OpenWrt*, namenjenog za ugrađene sisteme, i pronaći moguće delove podložne optimizaciji kako bi se na što bolji način iskoristili resursi platforme.

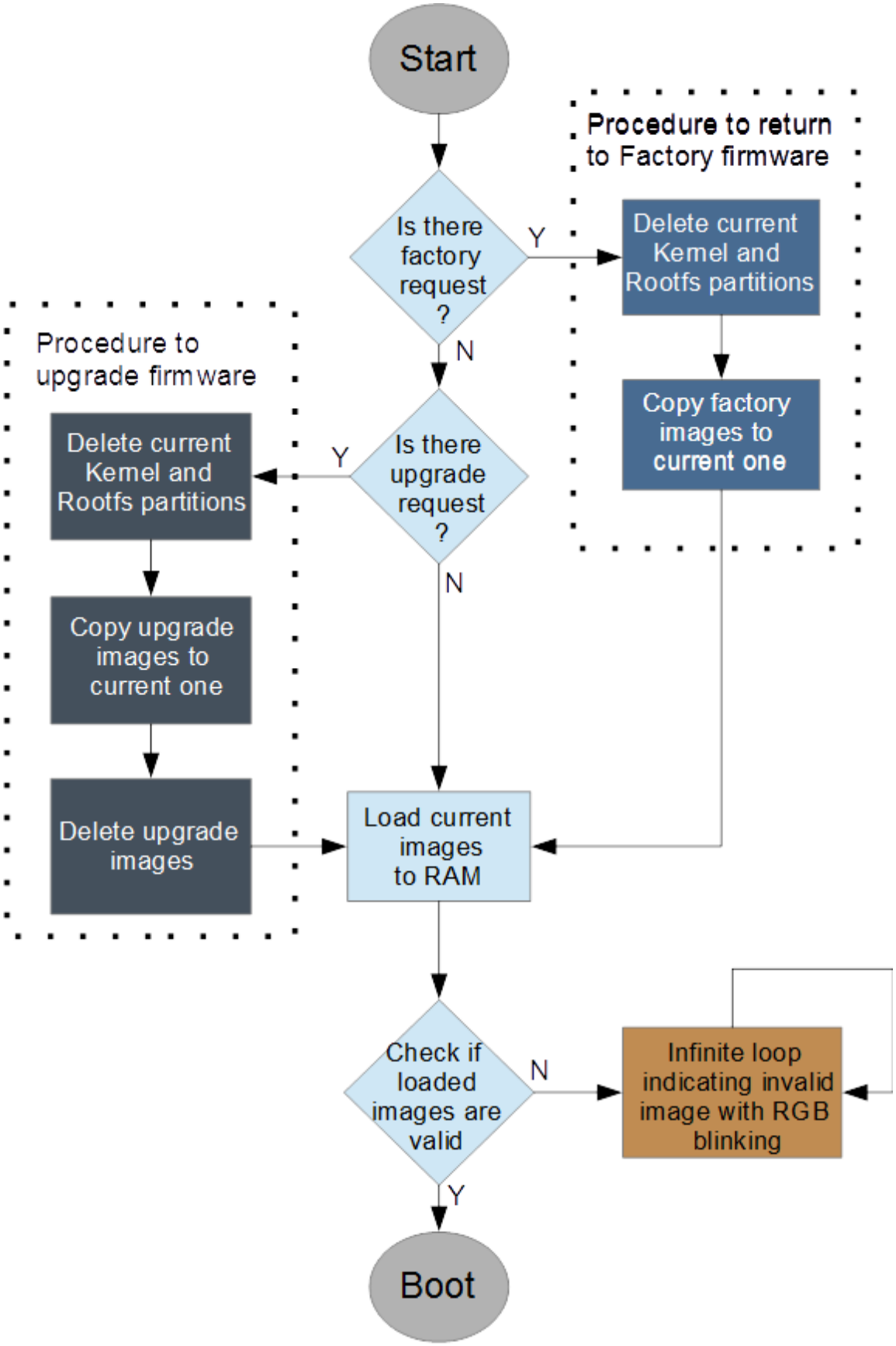
Predloženo i opisano rešenje obuhvata integraciju postojećih programa u sistem paketa, generisanje namenskih konfiguracionih datoteka u cilju podrške hardverskih komponenata kontrolera, definisanje nove memorijske mape za potrebe mehanizma za sigurno ažuriranje programske podrške kao i implementaciju samog mehanizma.

Skupom testova dokazana je potpuna funkcionalna ispravnost i stabilnost kontrolera u jednom kompletnom sistemu za automatizaciju kuća. Merenjem se pokazao stepen opterećenja i iskorišćenosti resursa, u pogledu radne memorije, slobodnog prostora na fleš memoriji kao i zauzeću centralne procesorske jedinice. Urađena je optimizacija po pitanju sistema datoteka, na osnovu vremenskih rezultata vezanih za dovođenje sistema do potpuno dostupnog za korisničku kontrolu od momenta dobijanja napajanja, gde se dobilo znatno poboljšanje performansi.

Izrada zadatka je predstavljala poseban izazov po pitanju ograničenih resursa u čije okvire je bilo potrebno ugraditi kompletnu programsku podršku određene kompleksnosti. Na kraju, kao dokaz i potvrda truda, imamo kontroler DOC400 kao centrali uređaj OBLO sistema za automatizaciju kuća. Dalji razvoj podrazumeva eventualnu integraciju novog kontrolera, hardverski jačeg, korišćenjem ovog rešenja uz moguće minimalne izmene.

7. Dodatak

U nastavku je dat koncept rada algoritma za ažuriranje programske podrške, kao i konkretno programsko rešenje datog algoritma.



Slika 7.1: Dijagram toka programa za ažuriranje programske podrške

```
#ifdef CONFIG_BOARD_DOC400
    #include "../lib_golden/gold_mechanism.h"

    gdm_setup_registers();
    /* check if user initiated factory request
     * from Linux or by pressing the reset button
     */
    if( gdm_check_factory_request() && gdm_check_btn_state() )
    {
        /* check if there is new user upgrade image */
        if( gdm_check_for_user_upgrade() )
        {
            /* no new images for upgrade, trying regular boot */
            debug("No new upgrade images\n");
            debug("Proceding to regular boot\n");
        }
        else
        {
            /* there is upgrade image, trying upgrade */
            gdm_change_led_state(UPGRADE_RGB_COLOR);
            gdm_start_user_upgrade();
            gdm_change_led_state(TURNED_OFF_RGB);
        }
    }
    else
    {
        /* restore factory image */
        gdm_change_led_state(FACTORY_RGB_COLOR);
        debug("Restoring factory image\n");
        gdm_start_factory_upgrade();
        gdm_change_led_state(TURNED_OFF_RGB);
    }

        gdm_change_led_state(BOOT_RGB_COLOR);
        gdm_restore_registers();

        /* execute predefined boot command */
        parse_string_outer(s, FLAG_PARSE_SEMICOLON |
                    FLAG_EXIT_FROM_LOOP);

        gdm_setup_registers();
        gdm_invalid_image_state();

        /* never should be reached */
        gdm_restore_registers();

#else
        /* execute predefined boot command */
        parse_string_outer(s, FLAG_PARSE_SEMICOLON |
                    FLAG_EXIT_FROM_LOOP);
#endif
```

Slika 7.2: Algoritam mehanizma ažuriranja programske podrške

8. Literatura

[1] The Buildroot user manual, https://buildroot.org/docs.html, pristupano 11.06.2016

[2] Qualcomm Atheros, Inc: IoT Client Overview: Control and Automation, 2015

[3] QCA4531 - A low-power Linux connectivity hub for the Internet of Everything, https://www.qualcomm.com/documents/low-power-wi-fi-qca4531, pristupano 11.06.2016

[4] OpenWrt dokumentacija, https://wiki.openwrt.org/doc/start, pristupano 11.06.2016

[5] QCA Software Development Kit (QSDK), https://wiki.codeaurora.org/xwiki/bin/QSDK/WebHome, pristupano 11.06.2016

[6] The Universal Boot Loader („Das U-Boot“), http://www.denx.de/wiki/publish/U-Bootdoc/U-Bootdoc.pdf, pristupano 11.06.2016

[7] General MTD documentation, http://www.linux-mtd.infradead.org/doc/general.html, pristupano 11.06.2016

[8] Homma Toru: *Evaluation of flash file systems for large NAND flash memory*, CELF Embedded Linux Conference, 2009.

[9] Milan Pandurov: *Programska podrška za apstrakciju i upravljanje sistemskim resursima namenskih sistema zasnovanih na Linux-u*, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2016

[10] Milan Tucić: *Protokol za razmenu poruka u sistemima pametnih kuća*, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2016

[11] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman: *Linux Device Drivers*, Third Edition, O'Reilly, Februar 2005