

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3052

Implementacija podrške za emulaciju PLC-a u alatu QEMU

Ardian Pantina

Zagreb, lipanj 2022.

DIPLOMSKI ZADATAK br. 3052

Pristupnik: **Ardian Pantina (0036502252)**

Studij: Računarstvo

Profil: Računarska znanost

Mentor: doc. dr. sc. Stjepan Groš

Zadatak: **Implementacija podrške za emulaciju PLC-a u alatu QEMU**

Opis zadatka:

Programirljivi logički kontroleri (engl. Programmable logic controller, PLC) su industrijska računala koja se koriste za automatizaciju industrijskih procesa. U industrijskim postrojenjima oni primjerice reguliraju temperaturu u nekoj smjesi ili brzinu vrtnje centrifuge. Njihov rad ima utjecaj na kvalitetu proizvoda te su posljedice neispravnog rada PLC-a financijske štete, a moguće su i ugroze ljudskih života. U novije vrijeme su se događali kibernetički napadi na industrijska postrojenja kojima je cilj zloupotrijebiti ranjivosti koje se nalaze u programskoj podršci industrijskih računala. Radi toga je bitno otkriti ranjivosti prije nego što mogu biti zloupotrebene, što je otežano činjenicom da većina programske podrške za PLC-ove nije javno dostupna. Za otkrivanje ranjivosti potrebno je istraživačima olakšati pristup i testiranje programske podrške PLC-ova. U sklopu ovog diplomskog rada potrebno je istražiti mogućnosti emulacije PLC-a u alatu QEMU. Konkretno, potrebno se je pozabaviti emulacijom periferije u alatu QEMU te dokumentirati postupak dodavanja novih uređaja i složiti jedan jednostavan primjer novog uređaja. Potrebno je također opisati način traženja grešaka tijekom razvoja novih perifernih uređaja korištenjem programa za tu namjenu (debuggers). Diplomskom radu priložiti napisani kod te citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 27. lipnja 2022.

SADRŽAJ

1. Uvod	1
2. SIMATIC S7-1200	2
2.1. Opće informacije	3
2.2. Karakteristike sklopovlja	4
2.3. Arhitektura procesora	4
3. Programska podrška PLC-a	6
3.1. Izvlačenje programske podrške	6
3.2. Reverzno inženjerstvo	7
4. QEMU	8
4.1. O alatu	8
4.2. <i>User-mode</i> emulacija	8
4.3. <i>System-mode</i> emulacija	9
4.4. Prevođenje instrukcija	9
4.5. QEMU uređaji	10
5. Emulacija <i>bootloadera</i>	11
5.1. Xilinxova inačica QEMU-a	11
5.2. Konfiguracija QEMU-a	11
5.3. Prilagodba <i>firmware</i> datoteke	12
5.4. Postavke QEMU-a za emulaciju PLC-a	14
6. Implementacija upravljačkih programa	16
6.1. QEMU Object Model (QOM)	16
6.2. Proces dodavanja novog uređaja	18
6.3. Implementacija sklopovskog brojača	21

7. Integracija <i>firmwarea</i> i <i>bootloadera</i>	25
7.1. Predaja kontrole <i>firmwareu</i>	25
7.2. Kopiranje <i>firmwarea</i> u radnu memoriju PLC-a	26
8. Zaključak	28
9. Literatura	29

1. Uvod

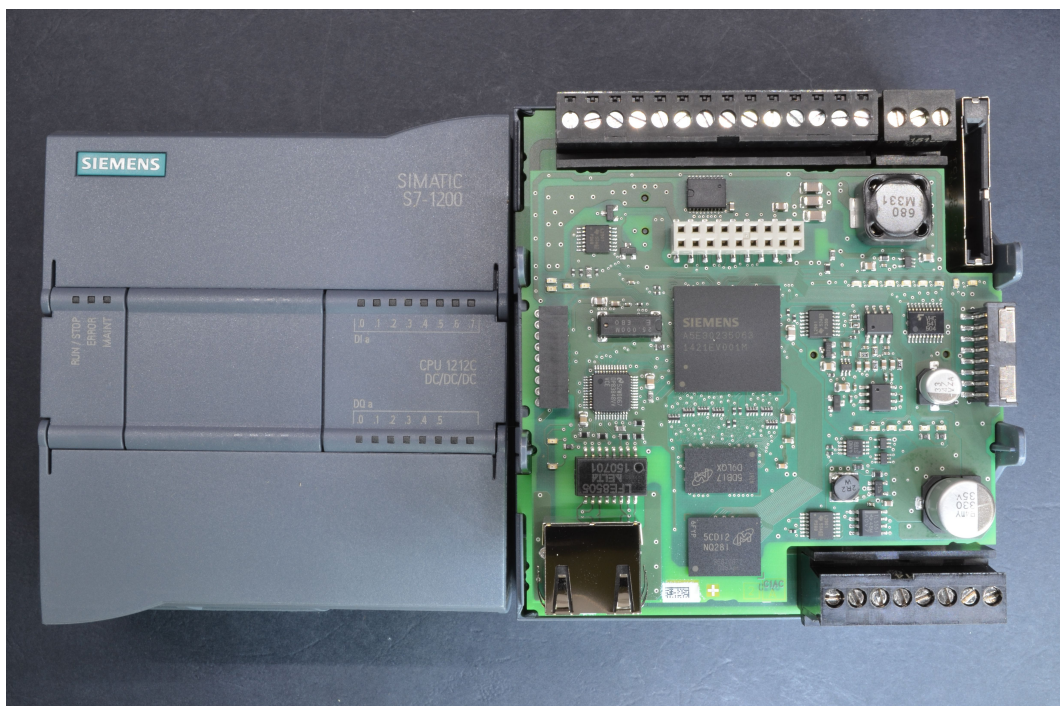
Programirajući logički kontroleri (engl. *Programmable logic controller*, PLC) su računala specijalizirana za upravljanje i automatizaciju proizvodnih procesa. Područja primjene ovih računala sežu od proizvodnih linija, robotskih uređaja pa sve do kritičnih dijelova infrastrukture poput nuklearnih elektrana. Zbog specifičnog područja primjene znatno različitog od osobnih računala, PLC-ovi imaju posebne zahtjeve za robusnost, pouzdanost i lakoću programiranja.

S obzirom na njihovu važnu ulogu u mnogim domenama, PLC-ovi su postali značajna meta kibernetičkih napada zloupotrebom ranjivosti u njihovoj programskoj podršci. Ovi napadi kao posljedicu mogu imati zaustavljanje proizvodnje čitavih pogona te čak ugrožavanje ljudskih života u ekstremnim okolnostima. Stoga je potrebno otkriti ranjivosti u takvim sustavima kako bi se od njih mogli obraniti u budućnosti. Otkrivanje istih donosi sa sobom niz izazova, koji su ujedno i motivacija iza ovog rada. Jedan od izazova je zatvorenost većine PLC-ova - naime, izvorni kôd programske podrške ovih uređaja je rijetko kad javno dostupan. Nadalje, programska podrška je usko vezana uz samo sklopovlje PLC-a što istraživačima otežava pristup ranjivostima i povećava troškove.

Ovaj rad se bavi emulacijom sustava PLC-a koristeći alat otvorenog kôda Quick Emulator (skraćeno QEMU). Nakon uvoda, u drugom poglavlju opisuje se sklopovlje i računalna arhitektura PLC-a koji se koristi za emulaciju. Treće poglavlje sadrži pregled programske podrške PLC-a te kako se do iste došlo reverznim inženjeringom. Zatim slijedi opis QEMU-a, prikazuju se njegove mogućnosti te princip rada. Nakon toga demonstrira se emulacija opisanog PLC-a koristeći QEMU i ostale alate otvorenog kôda. Posljednje poglavlje opisuje postupak dodavanja upravljačkog programa (engl. *driver*) u QEMU-ov izvorni kôd kako bi se omogućila komunikacija programske podrške PLC-a s perifernim uređajima poput brojača i memorijskih jedinica.

2. SIMATIC S7-1200

Kako bi se neki računalni sustav precizno emulirao, potrebno je imati uvid u njegovu arhitekturu i periferiju koju koristi. PLC kojim se ovaj rad bavi jest SIMATIC S7-1200 [1] tvrtke Siemens, prikazan na slici 2.1. Radi se o PLC-u opće namjene koji je optimalan za obavljanje računski jednostavnijih operacija u stvarnom vremenu. Sudeći po službenim referencama, ovaj PLC je svoju primjenu našao u širokom spektru industrija, od upravljanja postrojenjima za obradu drva pa sve do nadzora robota za varenje [2].





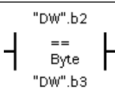

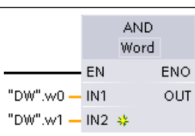
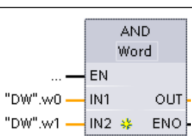
Slika 2.1: Prikaz glavnog modula PLC-a Simatic S7-1200 [3]

2.1. Opće informacije

SIMATIC je serija PLC-ova i sustava za automatizaciju tvrtke Siemens. Najnovija generacija serije zove se SIMATIC S7, te njoj pripada PLC S7-1200 kojim se bavi ovaj rad. Programi za automatizaciju se pokreću u posebnom programskom okruženju tvrtke Siemens naziva STEP 7. U ovom okruženju nudi se grafičko sučelje putem kojeg se može razvijati logika za upravljanje procesom automatizacije. Osim toga STEP 7 omogućuje praćenje i podešavanje svih uređaja u sustavu. Logika za automatizaciju se može programirati na tri različita načina:

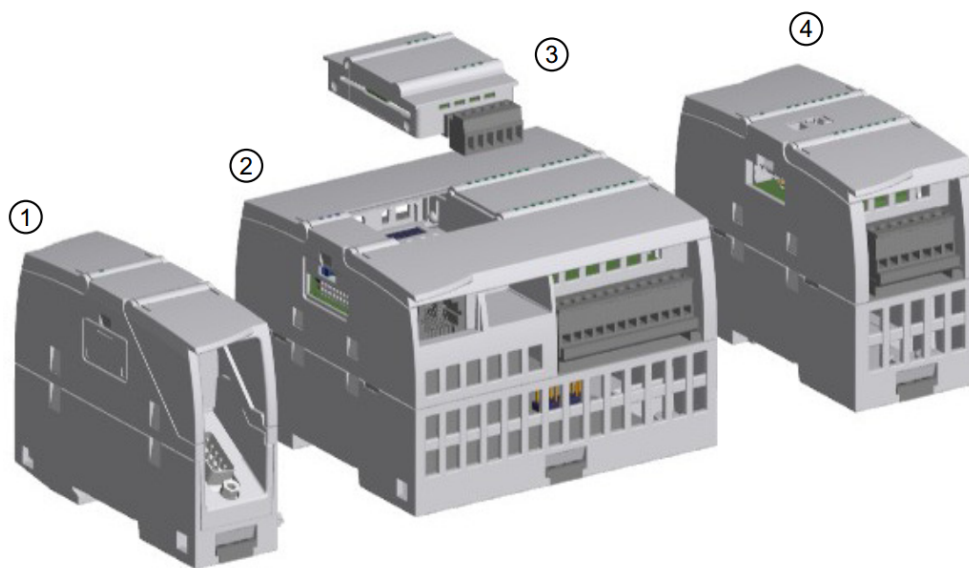
- LAD (*ladder logic*) - grafički programski jezik temeljen na dijagramima strujnih krugova,
- FBD (*functional block diagram*) - programski jezik temeljen na logičkim simbolima Booleove algebre po standardima DIN 40700/ DIN 40719,
- SCL (*structured control language*) - tekstualni programski jezik više razine koji podržava uvjetno grananje, petlje i skokove.

Detaljniji prikaz ovih načina programiranja je vidljiv na slici 2.2.

	LAD	FBD	SCL
Bitovna operacija			<pre>IF "DW".x11 THEN ... END_IF;</pre>
Oktetna operacija			<pre>IF "DW".b2 = "DW".b3 THEN ... END_IF;</pre>
Operacija nad riječima			<pre>out:= "DW".w0 AND "DW".w1;</pre>

Slika 2.2: Prikaz različitih načina programiranja u SIMATIC PLC-ovima.

S7-1200 je modularnog dizajna, te se može proširiti raznim modulima koje Siemens proizvodi. Glavni modul prikazan na slici 2.1 se u dokumentaciji zove *Central Processing Unit (CPU)* te osim mikroprocesora sadrži integrirano napajanje te ulazno-izlazna sučelja u kompaktnom kućištu. Ugradnjom dodatnih modula se PLC proširuje funkcionalnostima poput obrade analognih signala, komunikacije putem Profinet ili Ethernet protokola itd. Modularnost ovog PLC-a se može primjetiti i na slici 2.3.



Slika 2.3: Prikaz CPU-a modela S7-1200 s dodatnim modulima[4]. Legenda: 1) komunikacijski modul, 2) CPU inačica 1211C, 3) Modul za obradu signala, 4) rezervno napajanje

2.2. Karakteristike sklopovlja

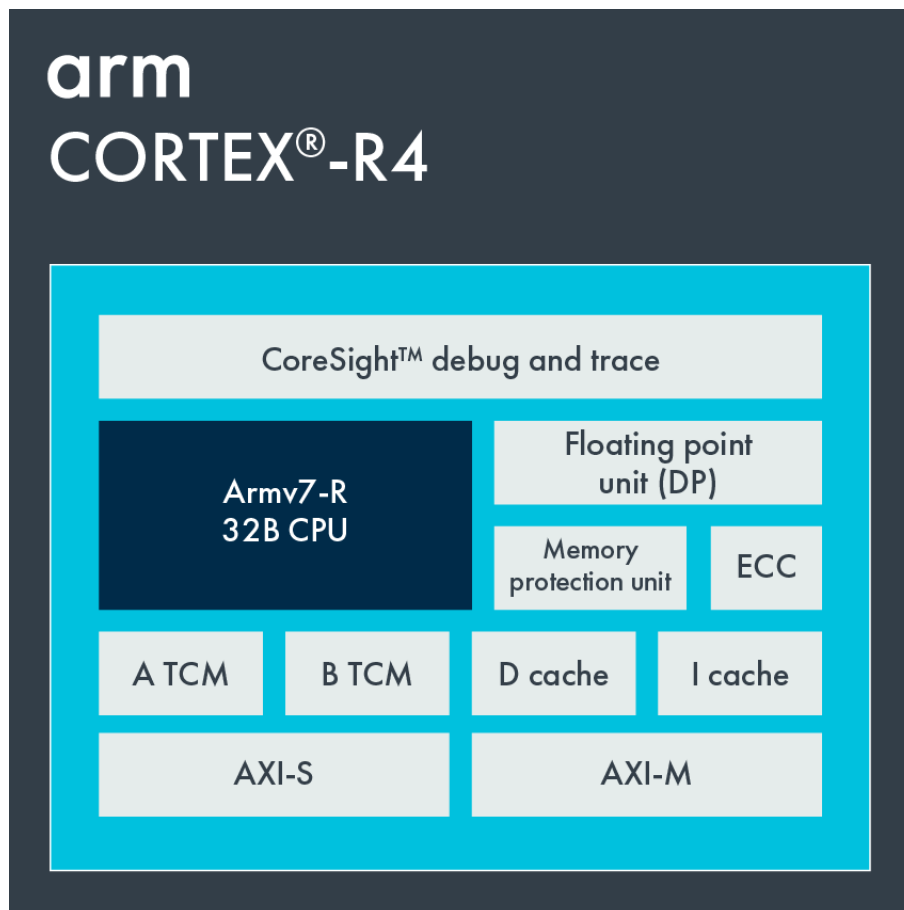
Iako su službeni podatkovni listovi (engl. *datasheet*) šturi po pitanju internog sklopovlja ovog PLC-a, postoji nekoliko radova koji su uspjeli izvući korisne informacije drugim metodama.

Thomas Weber je objavio zanimljiv rad [5] gdje je zbog propusta u lancu opskrbe uspio doći do većeg broja ploča korištenih S7-1200 PLC-a. Nakon njihovog otvaranja i analize otkriveni su stvarni modeli većine komponenta. Identificirana su sklopovska sučelja za JTAG, NAND flash i radnu memoriju. Detaljnijom analizom korištenjem metalurškog mikroskopa otkriven je i integrirani krug, a na njemu i sam procesor kojeg PLC koristi.

2.3. Arhitektura procesora

ARM Cortex-R je obitelj 32-bitnih i 64-bitnih RISC ARM procesora. Razlikuju se od Cortex-M (Microcontroller) i Cortex-A (Application) grupa po tome što su optimizirane za rad u stvarnom vremenu i u aplikacijama s naglaskom na visoku sigurnost[6], što ih čini prikladnim za uporabu u PLC-ovima.

SIMATIC S7-1200 PLC koristi ARM-ov Cortex-R4 procesor, u *big-endian* načinu rada. Oznaka verzije procesora je r1p3, što označava *revision 1, point 3*. Radi se o 32-bitnoj arhitekturi, što će biti ključno znati prilikom emulacije. ARM Cortex-R4 je ujedno i najmanji ARM-ov procesor za rad u stvarnom koji se temelji na ARMv7-R arhitekturi. Na slici 2.4 je detaljnije prikazana arhitektura Cortex-R4 inačice ARM-ovog procesora.



Slika 2.4: Blok dijagram arhitekture procesora ARM Cortex-R4

3. Programska podrška PLC-a

3.1. Izvlačenje programske podrške

U okviru projekta sklopovlje PLC-a nije bilo izravno dostupno, pa je do programske podrške bilo potrebno doći analiziranjem datoteka za nadogradnju. Te su datoteke dostupne na Siemensovim službenim stranicama uz prethodnu registraciju i autorizaciju[7]. Paket za nadogradnju se nalazi u jednoj datoteci ekstenzije ‘.upd’. Postoje dva moguća načina nadogradnje programske podrške, učitavanjem datoteke na web poslužitelj PLC-a ili koristeći SD karticu. Uočeno je kako veličina ovih datoteka monotono raste s novim inačicama, na temelju čega je zaključeno da se u svakoj datoteci nalazi čitav *firmware*. Analizom entropije datoteke utvrđeno je da je komprimirana, no nije bilo očito o kojem se algoritmu kompresije radi. Alati poput *binwalk*-a[8] su prepoznali kompresijske tokove, no sam algoritam kompresije nije bilo moguće razlučiti.

U konferencijskom radu znanstvenika tvrtke Quarkslab [9] točno je definiran algoritam kompresije. Radi se o relativno opskurnom algoritmu LZP3[10] koji se temelji na LZ skupini algoritama koji generiraju rječnik prilikom kompresije. U tom radu se također detaljno opisuje format same datoteke za nadogradnju *firmwareom*. Iako su ove informacije bile korisne, bez implementacije algoritma dekompresije nije bilo moguće doći do samog *firmwareom*. S obzirom na vremenska ograničenja te nedostatak dokumentacije, ručna je implementacija bila značajan izazov. Srećom, nakon kontaktiranja Jean-Baptiste Bedrunea, jednog od autora rada, objavljen je izvorni kod[11] programa kojim je bilo moguće dekomprimirati cijelu *.upd* datoteku.

3.2. Reverzno inženjerstvo

Analizom dekomprimirane datoteke može se uočiti tablica s prepoznatljivim segmentima poput *.text*, *.rodata*, *.data* sličnih ELF formatu. Na početku *.rodata* segmenta nalazi se polje unosa koji opisuju memorijski raspored PLC-a, prikazanih u tablici 3.1. Ovo je polje korisno jer potvrđuje kako PLC komunicira s perifernim uređajima metodom *memory-mapped input/output*.

Radi se o metodi gdje su memorija i registri ulazno-izlaznih uređaja preslikani na određene adresne lokacije. Ovo omogućuje da instrukcije strojnog kôda mogu pristupati ulazno-izlaznim uređajima na isti način kako bi se inače pristupalo radnoj memoriji. Svaki uređaj tada prati adresnu sabirnicu procesora i može odgovoriti na pristup memoriji tog uređaja, povezujući podatkovnu sabirnicu sa sklopovskim registrima uređaja. Kako bi ovo moglo funkcionirati, dijelovi adresnog prostora su posebno rezervirani za ulazno-izlazne uređaje i ne smiju biti dostupni radnoj memoriji.

Ime segmenta	Raspon	Veličina
itcm	0x00000000-0x00008000	0x00008000
ddram	0x00008000-0x04000000	0x03ff8000
configured_dtcm	0x10010000-0x10014000	0x00004000
internal_ram0	0x10030000-0x10040000	0x00010000
MAP3_TIMERS	0xffffbb00-0xffffbb15c	0x0000015c
MAP3_IOC	0xffffbc00-0xffffbc02c	0x0000002c
MAP3_FL_MEMCTL	0xffffbd00-0xffffbe000	0x00001000
MAP3_HSC4	0xffffb9300-0xffffb9380	0x00000080
MAP3_VIC	0xffffffc00-0xffffffe00	0x00000200

Tablica 3.1: Dio otkrivenih memorijskih mapiranja uređaja PLC-a.

Poznavanjem memorijskih mapiranja moći će se implementirati programska podrška za periferne uređaje, čime se u teoriji može omogućiti emulacija cijelog sustava PLC-a. Također je iz *.text* segmenta izvađen izvršni kôd *firmwarea* PLC-a koji će se podizati u emuliranom sklopovlju PLC-a putem QEMU-a, o čemu više slijedi u nastavku rada.

4. QEMU

4.1. O alatu

QEMU [12] je emulator otvorenog kôda čiji je originalni autor Fabrice Bellard. Omogućuje emulaciju programa ili čitavih sustava kako bi se mogli pokretati na arhitekturi ili operacijskom sustavu za kojeg nisu bili inicijalno predviđeni. Ovo se ostvaruje putem dinamičkog prevođenja, odnosno tako da se instrukcije izvorne binarne datoteke predviđene za drugu arhitekturu prevode na instrukcije koje odgovaraju arhitekturi na kojoj se QEMU izvodi. Licenciran je pod *GNU General Public License* (GPL) [13] licencom.



Slika 4.1: Službeni logo QEMU-a

QEMU podržava dva načina emulacije, tzv. *user-mode* i *system* emulacije koji se temeljno razlikuju po načinu rada.

4.2. *User-mode* emulacija

U ovom načinu rada QEMU pokreće pojedinačni Linux ili MacOS program umjesto cijelog sustava. Sustavski pozivi se također prevode, te ako se duljina riječi na računalu (32/64 bita) ne podudara s duljinom riječi emuliranog programa, QEMU može napraviti prilagodbu. Također je podržano upravljanje POSIX signalima u oba smjera. Drugim riječima, QEMU sve signale s računala može direktno preusmjeriti na emulirani program, te virtualne procesorske iznimke sintetizirati u signal (npr. ako program pokuša dijeliti s nulom, generira se *SIGFPE*)[14]. Inačice QEMU-a u *user-*

mode načinu rada pokreću se preko naredbenog retka koristeći `qemu-<arch>` program [argumenti...], gdje `arch` treba zamijeniti s arhitekturom koja se koristi, dok je program argument putanja do programa koji se pokreće. Također se nudi mnoštvo dodatnih argumenata za čije se korištenje preporučuje konzultirati službenu dokumentaciju[14].

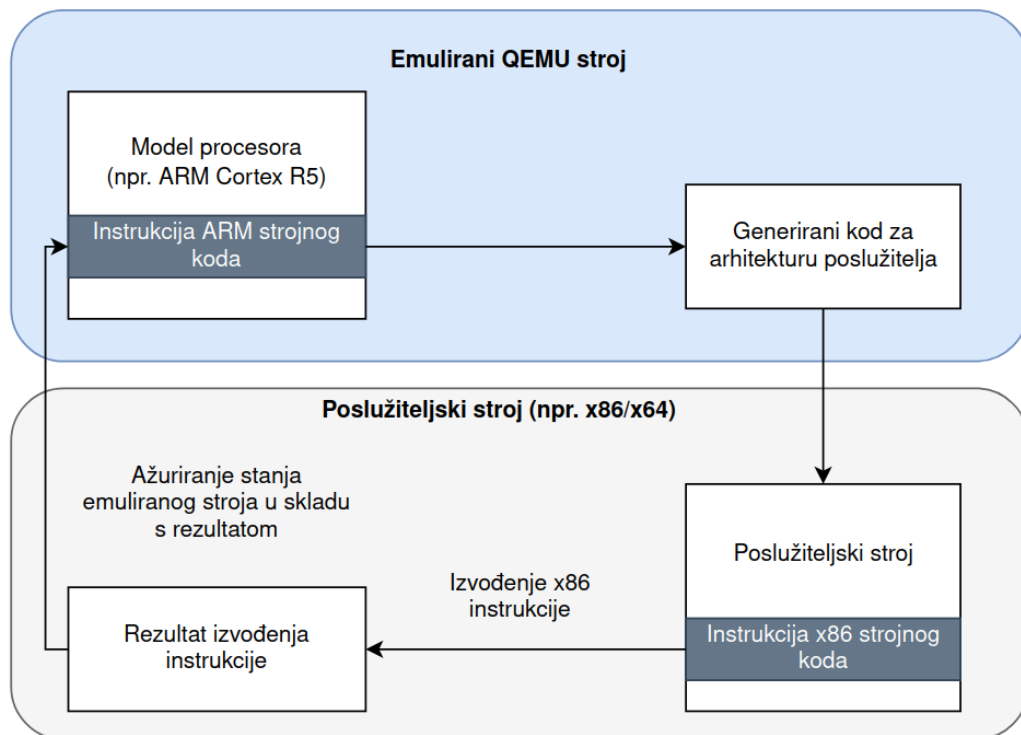
4.3. *System-mode* emulacija

S obzirom na to da se ovaj rad bavi emulacijom cijelog PLC-a, od operacijskog sustava, *bootloadera* pa sve do upravljačkih programa, ovaj način emulacije će nam biti u fokusu u odnosu na *user-mode*. U ovom načinu rada QEMU omogućuje emulaciju cijelog uređaja, operacijskog sustava i pripadne periferije bez potrebe za pristupom samom sklopovlju. Također se može koristiti za virtualno posluživanje više operacijskih sustava na jednom fizičkom računalu. Podržano je mnogo arhitektura, uključujući ARM, x86, RISC-V, PowerPC i ostalih[15].

Budući da svaka arhitektura ima svoje posebnosti prilikom izvođenja, tako se i *system* emulacija drukčije pokreće ovisno o sustavu kojeg se emulira. Emulacija programske podrške PLC-a kojim se bavi ovaj rad će se detaljnije opisati u sljedećim poglavljima.

4.4. Prevođenje instrukcija

QEMU koristi TCG (*Tiny Code Generator*) kako bi prevodio instrukcije s programa emuliranog stroja u instrukcije koje odgovaraju arhitekturi poslužiteljskog stroja. Princip rada je veoma sličan JIT (*just-in-time*) prevoditeljima. Glavna petlja izvođenja sastoji se od traženja unaprijed prevedenog bloka kôda na temelju trenutne vrijednosti programskog brojača (engl. *program counter*, PC). Ako se ne može pronaći prevedeni blok, potrebno je pozvati prevoditelja. Prevoditelj čita instrukcije emuliranog stroja i pretvara ih u tzv. *TCG Ops*, odnosno međuinstrukcije koje TCG može razumjeti. Nakon optimizacija, TCG generira niz odgovarajućih instrukcija poslužiteljske arhitekture koje QEMU zatim izvodi [16]. Vizualni prikaz ovog procesa dostupan je na slici 4.2.



Slika 4.2: Prikaz načina generiranja i izvođenja kôda. U ovom primjeru se QEMU pokreće na stroju arhitekture x86, dok se sustav koji se emulira temelji na ARM arhitekturi.

4.5. QEMU uređaji

Kod emulacije sustava važno je napomenuti kako nije dovoljno samo prevoditi instrukcije s jedne arhitekture na drugu. Osim izvođenja instrukcija, sustav komunicira s uređajima poput tvrdog diska ili mrežne kartice tako da pristupa njihovim memorijskim lokacijama, što za sobom donosi popratne pojave. Stoga QEMU nudi podršku za uređaje (engl. *device*) koji imaju ulogu oponašanja tih pojava. Krajnji cilj emulacije je korisničkom programu omogućiti upravljanje svim popratnim pojavama koje sustav očekuje, ali tako da samo mora čitati s određenih memorijskih lokacija gdje bi se ti uređaji inače nalazili na stvarnom sklopovlju.

QEMU u svom izvornom kodu već ima podršku za mnoštvo perifernih uređaja koji se mogu dodavati putem naredbenog retka ili integrirati modifikacijom samog kôda. Novi se uređaji mogu razviti pisanjem prikladnog C kôda i mapiranjem na određene memorijske lokacije putem konfiguracijskih datoteka. Ovo je detaljnije opisano u praktičnom dijelu rada gdje je prikazan postupak dodavanja novog uređaja za potrebe točnije emulacije ponašanja PLC-a.

5. Emulacija *bootloadera*

5.1. Xilinxova inačica QEMU-a

Kod emulacije cijelog sustava koristeći QEMU, potrebno je specificirati kakvo će se sklopovlje koristiti. U okviru ovog rada razmatra se ARM arhitektura pa će se koristiti *qemu-system-aarch64*. Valja napomenuti kako ova inačica podržava i 32-bitne ARM procesore unatoč imenu. QEMU za svaku od arhitektura podržava velik broj računalnih sustava poput pločica za razvoj, starijih modela mobitela i slično. Ipak, s obzirom na to da u trenutku pisanja ovog rada nijedan od podržanih uređaja ne koristi ARM-ov Cortex-R4 procesor, bilo je potrebno tražiti izvan okvira službene implementacije. Pronađena je inačica QEMU-a koju tvrtka Xilinx zasebno razvija[17], također otvorenog kôda. Zanimljiva je zato što nudi podršku za dodatne pločice, od kojih je jedna Xilinx ZynqMP ZCU1285 [18]. Ova je pločica odabrana zbog toga što na sebi ima ARM-ov Cortex-R5F procesor, koji je najbliži procesoru Cortex-R4 koji se nalazi na PLC-u kojeg se emulira. Sudeći po službenoj ARM dokumentaciji[19], Cortex-R5F se ne razlikuje značajno od procesora Cortex-R4 pa nije bilo brige oko vjerodostojnosti emulacije.

5.2. Konfiguracija QEMU-a

QEMU je pretežno napisan u programskom jeziku C, te koristi sustav Make za instalaciju. Nakon dohvaćanja QEMU-a sa Xilinxovog Git repozitorija[17], potrebno je instalirati dodatne softverske knjižnice o kojima QEMU ovisi. U konačnici treba odabrati arhitekturu za koju će se QEMU izgraditi koristeći make, što je vidljivo u isječku 5.1.


```
1 mkdir build
2 cd build
3 ../configure --target-list="aarch64-softmmu" --enable-fdt
  ↪ --disable-kvm --disable-xen --enable-gcrypt
4 make -j4
5 # -j opcija označava broj dretvi za izgradnju QEMU-a
```

Isječak 5.1: Konfiguracija i instalacija QEMU-a za ARM64 arhitekturu

U ovom slučaju se odabire `aarch64-softmmu` kao opcija za izgradnju zato što ćemo koristiti `qemu-system-aarch64` kao alat za emulaciju koji odgovara 32-bitnoj i 64-bitnoj ARM arhitekturi. Više o mogućnostima konfiguracije i instalacije može se pronaći u Xilinxovoj službenoj dokumentaciji za QEMU[20].

Xilinxova inačica QEMU-a ima svoje specifičnosti u odnosu na službenu inačicu, što se može očitati i u procesu izgradnje. Naime, osim prije navedenih alata potrebno je instalirati i tzv. *device tree binaries*[21]. Oni služe za vjernije modeliranje Xilinxovih pločica prilikom emulacije tako što definiraju memorijske regije, kontrolere za prekide, ulazno-izlazna sučelja itd. Njihova instalacija je preduvjet za korištenje Xilinxovih pločica za emulaciju, koje su u našem slučaju potrebne kako bi mogli pokrenuti programsku podršku S7-1200 PLC-a na ARM-ovom Cortex-R5F procesoru.

5.3. Prilagodba *firmware* datoteke

Inicijalni pokušaji emulacije fokusirali su se na pokretanje *firmwarea* PLC-a ekstrahiranog iz datoteke za nadogradnju opisane u poglavlju 4.1. Međutim, ovo se pokazalo neuspješnim budući da se prije samog *firmwarea* izvodi *bootloader* koji ima ulogu postavljanja inicijalizacijskih varijabli i perifernih uređaja. Stoga je bilo potrebno emulirati *bootloader* koji se originalno nalazi na SPI(engl. *Serial Peripheral Interface*) flash memoriji PLC-a, za razliku od *firmwarea* koji se učitava na NAND flash memoriju.

Kako bi se *bootloader* mogao pokrenuti na QEMU-u, bilo ga je potrebno pretvoriti s *big-endian* poretka okteta u *little-endian*. Razlog tome je što je kôd *bootloadera* predviđen za izvođenje u *big-endian* poretku okteta, kojeg QEMU nije podržavao u vrijeme pisanja ovog rada. Nakon toga je bilo moguće pokretati kôd PLC-ovog *bootloadera* na QEMU-u, te se namještanjem opcija za bilježenje izvođenih naredbi mogao vidjeti ispis, čiji je jedan dio vidljiv u isječku 5.2.

```

1 IN:
2 0x000000a0: e3e00911 mvn r0, #0x44000
3 -----
4 IN:
5 0x000000a4: e2800001 add r0, r0, #1
6 -----
7 IN:
8 0x000000a8: e5901000 ldr r1, [r0]
9 -----
10 IN:
11 0x000000ac: e2011c01 and r1, r1, #0x100
12 -----
13 IN:
14 0x000000b0: e3510000 cmp r1, #0
15 -----
16 IN:
17 0x000000b4: 0afffff9 beq #0xa0

```

Isječak 5.2: Ispis izvedenog strojnog kôda prilikom emulacije *bootloadera*.

Iz ispisa strojnog kôda se može zaključiti da PLC provjerava nalazi li se heksadekadska vrijednost $0x100$ na memorijskoj lokaciji $0xFFFBC000$. Ovo radi tako da se učita komplement vrijednosti $0x44000$, odnosno $0xFFFBFFF$ na koju se tada dodaje 1 čime dobivamo $0xFFFBC000$. Kombinacijom logičkog I te usporedbom rezultata s 0 izvodi se grananje, gdje se ovisno o vrijednosti registra izvođenje vraća na provjeru, ili nastavlja s procesom *bootloadera*. Pogledom na memorijska mapiranja PLC-a možemo zaključiti da se na $0xFFFBC000$ nalazi ulazno-izlazni kontroler, te se za isti treba implementirati upravljački program u sklopu QEMU-a.

5.4. Postavke QEMU-a za emulaciju PLC-a

U ovom dijelu prikazane su konačne postavke QEMU-a za pokretanje programske podrške PLC-a, te se pojedinačno opisuju konfiguracijski parametri i njihove uloge. Primjer pokretanja QEMU-a u svrhu emulacije PLC-a je prikazan na isječku 5.3.

```
1 ./build/aarch64-softmmu/qemu-system-aarch64 \  
2 -M arm-generic-fdt \  
3 -serial mon:stdio \  
4 -device loader,  
5 file=../../firmware-images/fw.stripped.rev,addr=0x40000,  
6 cpu-num=4,force-raw=true \  
7 -device loader,  
8 file=../../bootloader_images/Siemens_1211Cv4_bootloader.bin,addr=0x0  
9 -device loader,addr=0XFF5E023C,data=0x80088fde,data-len=4 \  
10 -device loader,addr=0xff9a0000,data=0x80000218,data-len=4 \  
11 -hw-dtb  
12 ↪ ../qemu-devicetrees/LATEST/SINGLE_ARCH/board-zynqmp-zcu1285.dtb \  
12 -m 4G -singlestep -d in_asm,nochain -s
```

Isječak 5.3: Konačne postavke za emulaciju

- **-serial mon:stdio:** Ovim parametrom preusmjeravamo serijski izlaz na tzv. QEMU monitor. Pomoću QEMU monitora možemo upravljati programskim tokom, ispisivati memorijske lokacije i slično.
- **-M arm-generic-fdt:** Ovom zastavicom se određuje koji će se stroj emulirati. Budući da se koristi Xilinx-ovu inačicu QEMU-a, ovdje se upotrebljava tzv. generički stroj, te se sama pločica koju ćemo koristiti specificira u sljedećoj uputi.
- **-device loader:** tzv. *loader* uređaj[22] služi za učitavanje određene memorijske vrijednosti u radnu memoriju emuliranog uređaja. Ako se specificira datoteka pomoću upute *-file* onda se učitava cijela datoteka na specificiranu adresnu lokaciju. Tako se u prva dva poziva učitavaju *firmware* te zatim *bootloader* u radnu memoriju. U druga dva poziva se postavljaju određene vrijednosti specifične za CPU koristeći *-data* parametar.
- **-hw-dtb board-zynqmp-zcu1285:** Ovime specificiramo *device tree binary*,

odnosno Xilinxov uređaj koji ćemo koristiti za emulaciju. U našem slučaju to je Zynq UltraScale+ RFSoc ZCU1285.

- **-m 4G**: Ovime određujemo da će naš emulirani uređaj imati 4GB radne memorije.
- **-singlestep**: Prevođenje ulaznih instrukcija jednu po jednu.
- **-d in_asm,nochain**: Debug opcija kojom QEMU ispisuje assembly na standardni izlaz. *nochain* opcijom kažemo QEMU-u da ne radi optimizacije, odnosno da ne ulančava prevedene naredbe već da ih izvodi jednu po jednu.
- **-s**: Skraćena zastavica kojom se stvara GDB (GNU Debugger) poslužitelj za *debugging*.

6. Implementacija upravljačkih programa

PLC je složen računalni sustav te kao takav ne funkcionira u vakuumu, već komunicira s više različitih uređaja. Pri inicijalnim pokušajima emulacije vrlo brzo se moglo zaključiti kako će biti potrebno implementirati dodatnu programsku podršku u obliku upravljačkih programa koji će "glumiti" periferne uređaje. Srećom QEMU ima modularnu programsku arhitekturu te se novi uređaji mogu relativno jednostavno dodavati na postojeće strojeve.

6.1. QEMU Object Model (QOM)

QEMU Object Model[23] (QOM) je objektno-orijentiran radni okvir unutar QEMU-a koji služi za modeliranje svih uređaja - CPU-ova, perifernih uređaja, memorijskih sučelja itd. koristeći pojam tipova. Pomoću njega se dodaje podrška za nove uređaje i sučelja u izvorni kôd QEMU-a, što se radi i u okviru ovog rada. QOM nudi sljedeće funkcionalnosti:

- Sustav za dinamičku registraciju novih tipova,
- podrška za jednostruko nasljeđivanje tipova,
- višestruko nasljeđivanje putem sučelja.

Svaki tip (uređaj, stroj, CPU itd.) se modelira koristeći `TypeInfo` strukturu. `TypeInfo` prilaže sljedeće informacije o svom tipu:

- `.name` - niz znakova koji označava ime tipa
- `.parent` - niz znakova koji specificira ime roditelja tipa
- `.instance_size` - veličina instance (objekta) danog tipa. QOM automatski obavlja stvaranje svakog novog tipa koristeći ovu informaciju
- `.class_init` - inicijalizacijska funkcija koja se poziva prilikom kreiranja tipa

U isječku 6.1. demonstrira se QOM na primjeru jednostavnog uređaja.

```
1 #include "qdev.h"
2
3 #define TYPE_MY_DEVICE "my-device"
4
5 typedef DeviceClass MyDeviceClass;
6 typedef struct MyDevice
7 {
8     DeviceState parent;
9     int reg0, reg1, reg2;
10 } MyDevice;
11
12 static const TypeInfo my_device_info = {
13     .name = TYPE_MY_DEVICE,
14     .parent = TYPE_DEVICE,
15     .instance_size = sizeof(MyDevice),
16     .instance_init = init,
17 };
18
19 static void init(void)
20 {
21     // inicijalizacijska logika
22 }
23
24 static void my_device_register_types(void)
25 {
26     type_register_static(&my_device_info);
27 }
28
29 type_init(my_device_register_types)
```

Isječak 6.1: Primjer jednostavnog uređaja za demonstraciju QOM-a.

Makro `type_init` se poziva prilikom pokretanja QEMU-a, te je nužan kako bi se izvršila inicijalizacija dodanog tipa. Funkcija `type_reigster_static` alokira memoriju i inicijalizira instancu danog tipa. Važno je napomenuti da za svaki tip postoji samo jedna instanca (drugim riječima, svaki tip je tzv. *singleton*).

6.2. Proces dodavanja novog uređaja

Prvi uređaj s kojim *bootloader* PLC-a komunicira je ulazno-izlazni kontroler. Iz strojnog kôda je zaključeno kako se on nalazi na adresi `0xFFFBC000` te se očekuje da će čitanje s te memorijske lokacije vratiti vrijednost `0x100`. U ovom se poglavlju opisuje proces dodavanja novog uređaja kroz primjer implementacije jednostavnog upravljačkog programa.

Prvi korak je dodavanje logike za uređaj koji se implementira. Potrebno je napraviti novu datoteku, te ju smjestiti u hw direktorij u izvornom kodu QEMU-a. Dobar dio kôda, prikazan u isječku 6.2, podudara se s prethodnim isječkom - definira se `TypeInfo` struktura te inicijalizacijska funkcija.

```
1 #include "qemu/osdep.h"
2 #include "qapi/error.h" /* provides error_fatal() handler */
3 #include "hw/sysbus.h" /* provides all sysbus registering func */
4 #include "hw/misc/plc_ioc.h"
5
6 #include "qemu/log.h"
7 #define TYPE_PLC_IOC "xlnx.plc_ioc"
8 typedef struct PLCIOCState PLCIOCState;
9 DECLARE_INSTANCE_CHECKER(PLCIOCState, PLC_IOC, TYPE_PLC_IOC)
10
11 struct PLCIOCState {
12     SysBusDevice parent_obj;
13     MemoryRegion iomem;
14 };
15
16 static uint64_t plc_ioc_read(void *opaque, hwaddr addr, unsigned int
17     ↪ size)
18 {
19     return 0x00000100;
20 }
21 static void plc_ioc_write(void *opaque, hwaddr addr, uint64_t val64,
22     ↪ unsigned int size)
23 {
24     // omogućeno pisanje, bez operacije
25 }
```

```

24
25 static const MemoryRegionOps plc_ioc_ops = {
26     .read = plc_ioc_read,
27     .write = plc_ioc_write,
28     .endianness = DEVICE_NATIVE_ENDIAN,
29 };
30
31 static void plc_ioc_instance_init(Object *obj)
32 {
33     PLCIOCState *s = PLC_IOC(obj);
34
35     /* alokacija segmenta memorijske mape */
36     memory_region_init_io(&s->iomem, obj, &plc_ioc_ops, s,
37     ↪ TYPE_PLC_IOC, 0x2c);
38     sysbus_init_mmio(SYS_BUS_DEVICE(obj), &s->iomem);
39 }
40
41 static const TypeInfo plc_ioc_info = {
42     .name = TYPE_PLC_IOC,
43     .parent = TYPE_SYS_BUS_DEVICE,
44     .instance_size = sizeof(PLCIOCState),
45     .instance_init = plc_ioc_instance_init,
46 };
47
48 static void plc_ioc_register_types(void)
49 {
50     type_register_static(&plc_ioc_info);
51 }
52
53 type_init(plc_ioc_register_types)

```

Isječak 6.2: Implementacija jednostavnog memorijski mapiranog uređaja.

S obzirom na to da se radi o uređaju mapiranom na određenoj memorijskoj lokaciji, bilo je potrebno i dodati `MemoryRegionOps` strukturu u implementaciju. U njoj se definiraju *callback* funkcije koje će se pozvati svaki put kad se čita ili piše na memorijsku lokaciju gdje se nalazi uređaj. U našem slučaju ne postoji funkcionalnost prilikom pisanja već je samo definirana kako bi pisanje bilo dozvoljeno. U slučaju čitanja jed-

nostavno se vraća vrijednost koju *bootloader* PLC-a inače očekuje u toku izvođenja. U inicijalizacijskoj funkciji se poziva *memory_region_init_io* gdje se inicijalizira memorijski prostor veličine $0x2c$, što se dojavljuje sustavskoj sabirnici QEMU-a pozivom *sysbus_init_mmio*.

Nakon dodavanja izvornog kôda, potrebno je podesiti memorijska mapiranja. U Xilinxovoj inačici to se radi u posebnom device-tree repozitoriju, koji pri prevođenju stvara prije spomenute *device tree binaries* datoteke. U njima se definiraju memorijska mapiranja, sučelja za uređaje i slično. U našem slučaju treba modificirati datoteku *zynqmp-iou.dtsi* te u njoj dodati dio kôda prikazan u isječku 6.3.

```
1 plc_ioc: plc_ioc@0xffffbc000 {
2     compatible = "xlnx,plc_ioc";
3     reg = <0x0 0xffffbc000 0x0 0x2c 0x0>;
4 };
```

Isječak 6.3: Definicija memorijskog mapiranja u *device tree* konfiguraciji.

Ovime naš Xilinx uređaj zauzima memorijski prostor veličine $0x2c$ na adresi $0xffffbc000$. Pomoću ključne riječi "xlnx,plc_ioc" povežujemo taj memorijski prostor s izvornim kodom u gornjem primjeru.

Konačno, potrebno je modificirati par konfiguracijskih datoteka kako bi QEMU dodao novi uređaj u izvršni paket pri prevođenju. Treba modificirati datoteke *hw/misc/Kconfig* i *hw/misc/meson.build* u skladu sa sadržajem isječka 6.4. Ovo odgovara *hw/misc/* direktoriju u kojeg je dodana datoteka *plc_ioc.c* gdje se nalazi logika upravljačkog programa iz isječka 6.2.

```
1 # hw/misc/Kconfig
2 config PLC_IOC
3     bool
4     ...
5 # hw/misc/meson.build
6 softmmu_ss.add(when: 'CONFIG_PLC_IOC', if_true: files('plc_ioc.c'))
```

Isječak 6.4: Izmjene u *Kconfig* i *Meson* konfiguracijskim datotekama.

Uz to, potrebno je dodati uređaj na odgovarajuću arhitekturu, u datoteci "hw/arm/Kconfig" u našem slučaju, što je pokazano u isječku 6.5.

```

1 config ZYNQ
2     bool
3     select A9MPCORE
4     select CADENCE # UART
5     ...
6     select ZYNQ_DEVCFG
7     select PLC_IOC

```

Isječak 6.5: Izmjene u Kconfig i Meson konfiguracijskim datotekama.

Sada je QEMU spreman za izgradnju s novim promjenama pokretanjem naredbe `make`. Onda se promjene mogu utvrditi pokretanjem QEMU-a s ulaznim parametrima iz isječka 5.3. Budući da se koristi `-serial mon:stdio` parametar, može se pokrenuti QEMU monitor pritiskom na CTRL+C te zatim na tipku A. Time se otvara sučelje naredbenog retka i možemo se uvjeriti da novi uređaj radi kako treba pogledom na ispis vidljiv u isječku 6.6.

```

1 QEMU 6.1.0 monitor - type 'help' for more information
2 (qemu) x /x 0xFFFFBC000
3 00000000ffffbc000: 0x00000100

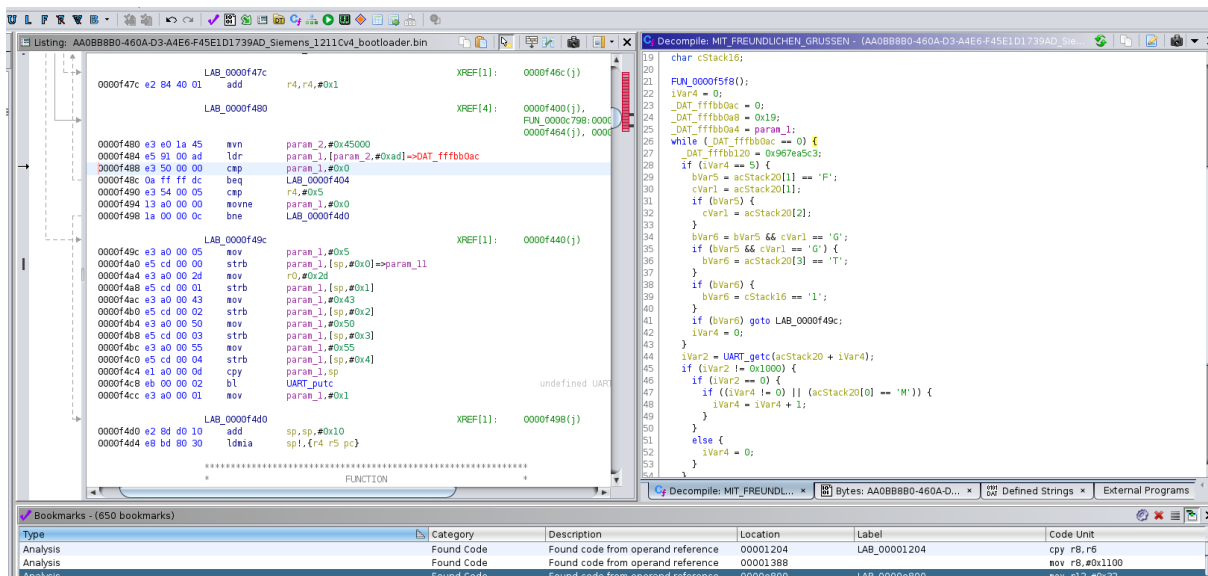
```

Isječak 6.6: Provjera memorijskog mapiranja uređaja putem QEMU monitora.

6.3. Implementacija sklopovskog brojača

Sljedeći izazov bila je implementacija sklopovskog brojača. Sklopovski brojači su ključna komponenta računalnih sustava zato što omogućuju praćenje vremena u pozadini dok CPU izvršava naredbe nekog procesa istovremeno. S obzirom na to da su PLC-ovi računala namijenjena za rad u stvarnom vremenu, praćenje vremena je još važnije nego što bi bilo u konvencionalnim računalima.

Prilikom emulacije *bootloadera* izvođenje je stalo na određenom dijelu kôda koji je čitao s memorijske lokacije `0xFFFFBB0AC` u petlji strojnog kôda. Detaljnijom analizom strojnog kôda *bootloadera* u alatu Ghidra[24], čije je sučelje prikazano na slici 6.1, pronađen je dio kôda koji čeka na istek intervala s jednog sklopovskog brojača za kojeg se očekuje da se nalazi na adresi `0xFFFFBB0AC`.



Slika 6.1: Prikaz dekompiranog strojnog kôda *bootloadera* u alatu za reverzno inženjerstvo Ghidra.

Fokus je na gornjem dijelu dekompiranog (obrnut proces od prevođenja izvornog kôda, engl. *decompile*) kôda, gdje se može vidjeti kako PLC čeka da se određeni dio memorije postavi na pozitivnu vrijednost prije nego se nastavi izvođenje. Iz rada Ali Abassija i ostalih [25] zaključeno je da se radi o dijelu *bootloadera* za specijalni pristup, gdje postoji prozor od 500ms gdje se čeka na unos posebnog niza znakova prije nego se nastavi s procesom. Cilj je bio modelirati ovo ponašanje sklopovskog brojača tako da se razvije prilagođen upravljački program.

Na memorijskoj lokaciji `0xffffbb0a0` je trenutni broj otkucaja. Sljedeća 4 okteta označavaju vršnu vrijednost otkucaja od kojih se broji, `0xffffbb0a4`, zatim `0xffffbb0a8` označava stanje (*stop* ili *start*), i u konačnici na `0xffffbb0ac` se gleda je li brojač uključen. Interakcija s ovim memorijskim lokacijama je vidljiva u isječku 6.7.

```

1 DAT_ffffbb0ac = 0; // postavljanje mem. lokacija brojača
2 _DAT_ffffbb0a8 = 0x19;
3 _DAT_ffffbb0a4 = param_1;
4 while (_DAT_ffffbb0ac == 0) {
5     _WATCHDOG_TIMER = 0x967ea5c3;
6     ...
7 }

```

Isječak 6.7: Dekompajlirani kôd funkcije koja čeka na sklopovski brojač.

U skladu s navedenim saznanjima napisan je upravljački program po QOM modelu. Ključan dio za demonstraciju jesu metode koje se pozivaju prilikom čitanja i pisanja na memorijske lokacije samog brojača, prikazane u isječku 6.8.

```

1 #define TICKS_PER_MS 100
2
3 int start_time_ms = 0;
4 int tick_count = 50;
5 int stop_start = 1;
6
7 static uint64_t plc_timer_ffffbb0ac_read(void *opaque, hwaddr addr,
↪ unsigned int size) {
8     if (start_time_ms == 0) {
9         start_time_ms = qemu_clock_get_ms(QEMU_CLOCK_VIRTUAL);
10    }
11
12    int curr_time = qemu_clock_get_ms(QEMU_CLOCK_VIRTUAL);
13    int ticks_left = tick_count - (curr_time - start_time_ms) /
↪ TICKS_PER_MS;
14    if (ticks_left < 0) {
15        ticks_left = 0; // one-shot timer
16    }
17
18    switch (addr) {
19        case 0x0:
20            return ticks_left;
21        case 0x4:
22            return tick_count;

```

```

23     case 0x8:
24         return stop_start;
25     case 0xc:
26         if (ticks_left > 0) return 0; // timer still running
27         return 1; // timer finished
28     default:
29         return curr_time - start_time_ms; // elapsed time in ms
30 }
31 }

```

Isječak 6.8: Dio upravljačkog programa odgovoran za ponašanje brojača prilikom čitanja. Za potrebe modeliranja protoka vremena koristi se QEMU-ov interni brojač, čije se trenutno vrijeme dohvaća pozivom funkcije `qemu_clock_get_ms(QEMU_CLOCK_VIRTUAL)`. Iz isječka 6.9 može se zaključiti da je logika prilikom pisanja na memorijsku lokaciju brojača znatno jednostavnija - ovdje je samo potrebno čuvati stanje memorije u internim varijablama upravljačkog programa.

```

1  static void plc_timer_fffbb0ac_write(void *opaque, hwaddr addr,
↪  uint64_t val64, unsigned int size) {
2      uint32_t value = val64;
3      switch (addr) {
4          case 0x4:
5              // setter - broj otkucaja koje brojač mora otkucati do 0
6              tick_count = value;
7              start_time_ms = qemu_clock_get_ms(QEMU_CLOCK_VIRTUAL);
8              break;
9          case 0x8:
10             // stop_start
11             stop_start = value;
12             break;
13     }
14 }

```

Isječak 6.9: Dio upravljačkog programa brojača koji modelira pisanje.

7. Integracija *firmwarea* i *bootloadera*

Prikladnom konfiguracijom QEMU-a i implementacijom potrebnih upravljačkih programa za periferiju uspješno je emuliran *bootloader* PLC-a. Sljedeći korak je emulacija *firmwarea*.

7.1. Predaja kontrole *firmwareu*

Završni dio *bootloadera* vrši konačne provjere prije nego što preda kontrolu *firmwareu*. Putem registra R7 provjerava se je li uspješno izvršeno kopiranje *firmwarea* u radnu memoriju PLC-a. Ako vrijednosti odgovaraju očekivanima, obavlja se skok na prvu instrukciju *firmwarea* što je prikazano u isječku 7.1.

```
1      cmp      r7,#0x1      ; provjera vrijednosti registra r7
2      beq      LAB_00014b60
3      b        LAB_00014b80
4      mov      r0,#0x40000  ; učitaj adresu firmwarea u r0
5      ldr      r0,[r0,#offset DAT_00040004]
6      cmp      r0,#0x0
7      beq      LAB_00014b80
8      mov      r1,#0x1
9
10     mcr      p15,0x0,r1,cr9,cr1,0x1
11     dsb      SY
12     cpy      pc,r0      ; skok na prvu instrukciju firmwarea
13     mov      r1,#0x1
```

Isječak 7.1: Završni dio *bootloadera* gdje se predaje kontrola *firmwareu*.

7.2. Kopiranje *firmwarea* u radnu memoriju PLC-a

Osim inicijalizacije perifernih uređaja i pokretanja sustava, *bootloader* ima ulogu kopiranja *firmwarea* s NAND flash memorije u radnu memoriju.

Analizom dijela *bootloadera* koji obavlja kopiranje *firmwarea* u radnu memoriju zaključeno je da se dio *firmwarea* kopira na adresu $0x40000$, a početni dio na sam početak, odnosno adresu $0x0$. S obzirom na to da se na početku memorije inicijalno nalazi *bootloader*, potrebno je pisati preko tog dijela memorije. Srećom, ovo ne predstavlja problem zato što se u tom trenutku *bootloader* nalazi na adresi većoj od $0x10000$.

Ovaj dio logike je emuliran implementacijom upravljačkog programa koji učitava dio *firmwarea* te ga kopira u adresni prostor radne memorije na mjesto $0x0$, što je prikazano u isječku 7.2. Taj dio posla se obavlja nakon što se početni dio *bootloadera* izvede kako bi se izbjeglo pisanje preko dijela *bootloadera* koji se još nije izvršio.

```
1 AddressSpace *as = &address_space_memory;
2 FILE *f;
3 char buf[MAX_FILE_SIZE];
4 f = fopen(EXEC_IN_LOMEM_FILENAME, "rb");
5 if (f) {
6     n = fread(buf, sizeof(char), MAX_FILE_SIZE, f);
7 } else {
8     error_printf_unless_qmp("[plc_fw_copy.c] [FAIL] failed loading file
9     ↪ %s\n", EXEC_IN_LOMEM_FILENAME);
10 }
11 if (n > 0) {
12     error_printf_unless_qmp("[SUCCESS] loaded file %s: %s\n",
13     ↪ EXEC_IN_LOMEM_FILENAME, buf);
14 }
15 res = address_space_rw(as, 0x0, attrs, buf, MAX_FILE_SIZE, true);
16 if (res == MEMTX_OK) {
17     error_printf_unless_qmp("[SUCCESS] wrote file to addr. space %s:
18     ↪ %s\n", as->name, buf);
19 }
```

Isječak 7.2: Završni dio *bootloadera* gdje se predaje kontrola *firmwareu*.

Na prvoj liniji isječka 7.2. specificira se adresni prostor u kojeg će se kopirati dio memorije. Ovo je bitno zato što QEMU podržava više adresnih prostora, no u ovom slučaju želimo da je isti unificiran. Konkretnije, želimo da se kôd *bootloadera* i *firmwarea* nalazi u istom adresnom prostoru kao i svi periferni uređaji s kojima PLC komunicira. Ovo je postignuto upravo eksternom varijablom *address_space_memory*. Uspješno razmještanje memorije potvrđeno je QEMU monitorom, što je prikazano u isječku 7.3.

```

1 QEMU 6.1.0 monitor - type 'help' for more information
2 (qemu) x /x 0x0
3 0000000000000000: 0xe59ff0b4
4 (qemu) c
5 (qemu) x /x 0x0
6 0000000000000000: 0xe59ff0e0

```

Isječak 7.3: Potvrda kopiranja dijela *firmwarea* na početak radne memorije nakon izvođenja *bootloadera*. QEMU je u ovom slučaju pokrenut tako da stane na prvoj instrukciji. Naredbom *c* se QEMU-u kaže da nastavi s izvođenjem te je tako utvrđeno da se kopiranje izvelo nakon izvršavanja *bootloadera*.

Sad se može pratiti izvođenje *firmwarea* i *bootloadera* ispisom naredbi strojnog koda koje su prikazane u isječku 7.4.

```

1 0x00014b78: f57ff04f dsb sy
2 0x00014b7c: e1a0f000 mov pc, r0
3 0x00040040: e3a00000 mov r0, #0
4 0x00040044: ee070f15 mcr p15, #0, r0, c7, c5, #0
5 0x00040048: f57ff04f dsb sy
6 0x0004004c: f57ff06f isb sy
7 0x00040050: ee110f10 mrc p15, #0, r0, c1, c0, #0
8 0x00040040: e3a00000 mov r0, #0
9 0x00040044: ee070f15 mcr p15, #0, r0, c7, c5, #0
10 0x00040048: f57ff04f dsb sy
11 0x0004004c: f57ff06f isb sy
12 0x00040050: ee110f10 mrc p15, #0, r0, c1, c0, #0

```

Isječak 7.4: Dio emuliranog strojnog kôda gdje se prebacuje kontrola *firmwareu*. Primijetiti skok na adresu 0x40040.

8. Zaključak

Emulacija cijelog računalnog sustava zahtjevan je zadatak koji očekuje duboko poznavanje emuliranog sustava. Osim programske podrške, potrebno je imati uvid u arhitekturu procesora, proces pokretanja sustava i periferiju s kojom sustav komunicira. Ako sustav kojeg emuliramo nema javno dostupnu dokumentaciju, potrebno je primijeniti tehnike reverznog inženjerstva i istraživanja javnih izvora (engl. *open-source intelligence*, OSINT) kako bi se došlo do potrebnih informacija.

U sklopu ovog rada istraživao se Siemensov PLC S7-1200 u svrhu emulacije kako bi se postavili temelji za jednostavnije istraživanje ranjivosti. Koristio se QEMU, alat izvornog kôda koji se pokazao veoma korisnim za emulaciju cijelog sustava, od *bootloadera* pa sve do programske podrške za periferiju. Njegova iznimna proširivost je bila ključna kako bi se mogla implementirati dodatna programska podrška za vanjske uređaje bez koje potpuna emulacija ne bi bila moguća.

Kao rezultat rada ostvarena je uspješna emulacija cijelog sustava PLC-a od *bootloadera* do *firmwarea* na Xilinxovoj javnoj inačici QEMU-a. Demonstrirane su prilagodbe u konfiguraciji koje su bile potrebne kako bi se programska podrška PLC-a mogla pokrenuti bez potrebe za sklopovljem na kojem se inače pokreće, što može znatno olakšati proces traženja sigurnosnih ranjivosti. Konačno, implementirana je programska podrška za periferne uređaje PLC-a u obliku upravljačkih programa na QEMU platformi.

9. LITERATURA

- [1] Siemens. *SIMATIC S7-1200*, 2022. URL <https://new.siemens.com/global/en/products/automation/systems/industrial/plc/s7-1200.html>. Pristupljeno 30.5.2022.
- [2] Siemens. *SIMATIC S7-1200 - službena web stranica*, 2022. URL <https://new.siemens.com/global/en/products/automation/systems/industrial/plc/s7-1200.html>. Pristupljeno 7.6.2022.
- [3] Nepoznat autor. *S7 Detalji*, 2022. URL http://s7detali.narod.ru/S7_1200/S7_1212C.html. Pristupljeno 31.5.2022.
- [4] Siemens. *SIMATIC S7, S7-1200 Programmable controller System Manual*, 2022. URL <https://support.industry.siemens.com/cs/document/109741593/simatic-s7-s7-1200-programmable-controller?lc=en-ww>. Inačica V4.5 05/2021.
- [5] Thomas Weber. *Reverse Engineering Architecture And Pinout of Custom Asics*, 2019. URL <https://sec-consult.com/blog/detail/reverse-engineering-architecture-pinout-plc/>. Pristupljeno 31.5.2022.
- [6] ARM. *ARM® Cortex®-R Series Programmer's Guide*, 2014. URL <https://documentation-service.arm.com/static/5f72ef531b758617cd9546ff>. Pristupljeno 13.6.2022.
- [7] Siemens. *Firmware update for CPU 1212C, DC/DC/DC, 8DI/6DO/2AI*, 2022. URL <https://support.industry.siemens.com/cs/document/107539748/firmware-update-for-cpu-1212c-dc-dc-dc-8di-6do-2ai?dti=0&lc=en-DE>. Pristupljeno 8.6.2022.
- [8] ReFirmLabs. *Binwalk*, 2022. URL <https://github.com/ReFirmLabs/binwalk>. Pristupljeno 31.5.2022.

- [9] Florent Monjalet Jean-Baptiste Bédrune, Alexandre Gazet. Supervising the supervisor: Reversing proprietary scada tech. <https://web.archive.org/web/20211120202855/https://conference.hitb.org/hitbsecconf2015ams/wp-content/uploads/2015/02/WHITEPAPER-Reversing-Proprietary-SCADA-Tech.pdf>, 2015. Pristupljeno 31.5.2022.
- [10] Charles Bloom. Lzp: A new data compression algorithm. In James A. Storer and Martin Cohn, editors, Data Compression Conference, page 425. IEEE Computer Society, 1996. Pristupljeno 31.5.2022.
- [11] Jean-Baptiste Bedrune. Unpacker for siemens s7 1200 plc firmware. <https://github.com/jibeee/s7unpack>, 2021. Pristupljeno 31.5.2022.
- [12] Fabrice Bellard. *QEMU (Quick Emulator)*, 2003. URL <https://www.qemu.org/>. Pristupljeno 30.5.2022.
- [13] Richard Stallman. *GNU General Public License*, 1989. URL <https://www.gnu.org/licenses/gpl-3.0.html>. Pristupljeno 30.5.2022.
- [14] QEMU. *QEMU Usermode documentation*, 2022. URL <https://www.qemu.org/docs/master/user/main.html>. Pristupljeno 30.5.2022.
- [15] QEMU. *QEMU System mode documentation*, 2022. URL <https://qemu.readthedocs.io/en/latest/system/targets.html>. Pristupljeno 30.5.2022.
- [16] Alex Bennée. *Multi-threaded emulation for QEMU*, 2022. URL <https://lwn.net/Articles/697265/>. Pristupljeno 30.5.2022.
- [17] Xilinx. Xilinx's fork of quick emulator (qemu) with improved support and modeling for the xilinx platforms. <https://github.com/Xilinx/qemu>, 2022. Pristupljeno 1.6.2022.
- [18] Xilinx. Zynq ultrascale+ rfsoc zcu1285 characterization kit. <https://www.xilinx.com/products/boards-and-kits/zcu1285.html>, 2022. Pristupljeno 1.6.2022.
- [19] ARM. *ARM Cortex-R5*, 2022. URL <https://www.arm.com/products/silicon-ip-cpu/cortex-r/cortex-r5>. Pristupljeno 1.6.2022.

- [20] Xilinx. *QEMU User Documentation*, 2022. URL <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/821395464/QEMU+User+Documentation>. Pristupljeno 1.6.2022.
- [21] Xilinx. *Device trees used by QEMU to describe the hardware*, 2022. URL <https://github.com/Xilinx/qemu-devicetrees>. Pristupljeno 1.6.2022.
- [22] QEMU. *QEMU Generic Loader*, 2022. URL <https://qemu.readthedocs.io/en/latest/system/generic-loader.html>. Pristupljeno 1.6.2022.
- [23] QEMU. *QEMU Object Model*, 2022. URL <https://qemu.readthedocs.io/en/latest/devel/qom.html>. Pristupljeno 2.6.2022.
- [24] NSA (National Security Agency). *Ghidra*, 2022. URL <https://ghidra-sre.org/>. Pristupljeno 2.6.2022.
- [25] Thorsten Holz Ali Abbasi, Tobias Scharnowski. *Doors of durin: The veiled gate to siemens s7 silicon*. <https://i.blackhat.com/eu-19/Wednesday/eu-19-Abbasi-Doors-Of-Durin-The-Veiled-Gate-To-Siemens-S7-Silicon.pdf>, 2019. Pristupljeno 31.5.2022.

Implementacija podrške za emulaciju PLC-a u alatu QEMU

Sažetak

Ovaj rad bavi se emulacijom programske podrške za PLC SIMATIC S7-1200. Rad započinje analizom sklopovlja PLC-a te programske podrške koja se na njemu izvodi. Reverzним inženjerstvom datoteke za nadogradnju PLC-a dolazi se do programske podrške koja će se emulirati. Zatim se opisuje programski alat otvorenog kôda QEMU koji služi za emulaciju cijelog sustava PLC-a. Slijedi detaljan opis potrebne konfiguracije QEMU-a za uspješnu emulaciju PLC-a bez potrebe za korištenjem hardvera. Budući da PLC komunicira s velikim brojem perifernih uređaja, potrebno je implementirati upravljačke programe za periferiju kako bi emulacija bila vjerodostojna stvarnom sustavu. Imajući to u vidu, pokazuje se kako se podrška za periferiju dodaje u izvorni kôd QEMU-a putem ugrađenog objektno-orijentiranog programskog modela. Konačno se demonstrira kako se *bootloader* povezuje s *firmwareom* PLC-a, rezultirajući kompletnijom emulacijom sustava.

Ključne riječi: QEMU, PLC, emulacija, SIMATIC S7-1200

Implementation of software support for QEMU emulation of PLCs

Abstract

This thesis is concerned with the emulation of software for the SIMATIC S7-1200 PLC. The thesis begins with an analysis of the PLC's hardware and software. By reverse engineering the PLC's firmware update file, the firmware of the PLC is extracted. What follows is an overview of QEMU, an open-source tool that is used for full system emulation of the PLC. The configuration necessary for successful execution of firmware on the emulated system is described in detail. Since the PLC also communicates with a large number of peripheral devices, it is necessary to implement device drivers for a more accurate emulation of the real system. Having that in mind, the implementation of peripheral device drivers is demonstrated via QEMU's object-oriented software model. Finally, the thesis shows how the bootloader is integrated into the PLC's firmware, resulting in a more complete emulation of the system.

Keywords: QEMU, PLC, emulation, SIMATIC S7-1200