Miha Gjura, Red Pitaya
Andrej Trost, Fakulteta za elektrotehniko, UL

Uporabniški priročnik in eksperimenti z Red Pitayo

marec 2025
1. izdaja

CIP – Kataložni zapis o publikaciji

Kazalo

Kazalo slik

Kazalo tabel

Slovar kratic

A

- ADC – analogno-digitalni pretvornik (angl. Analog-to-Digital Converter)
- AIN – analogni vhod (angl. Analog Input)
- AMBA – arhitektura povezave na čipih (angl. Advanced Microcontroller Bus Architecture)
- AMD – ime podjetja (angl. Advanced Micro Devices)
- ARM – vrsta procesorja (angl. Advanced RISC Machines)
- ASG – generator poljubnih signalov (angl. /Arbitrary Signal Generator)
- AXI – protokol na ARM čipih (angl. Advanced eXtensible Interface)

B

- BRAM – bralno-pisalni pomnilnik (angl. Block Random Access Memory)

C

- CAN – vrsta serijske komunikacije (angl. Control Area Network)
- CH – kanal (angl. Channel)
- CMD – ukazni poziv (angl. Command prompt)
- CR – (angl. Carriage Return)
- CRLF – (angl. Carriage Return, Line Feed)

D

- DAC – digitalno-analogni pretvornik (angl. Digital-to-Analog Converter)
- DC – Enosmerna napetost/tok (angl. Direct Current)
- DDR – tip spomina (angl. Data Double Rate)
- DIO – digitalni vhod/izhod (angl. Digital Input/Output)
- DW – širina podatka (angl. Data Width)

E

- ECDSA – vrsta elektronskega podpisa (angl. Elliptic Curve Digital Signature Algorithm)

F

- FPGA – (angl. Field Programmable Gate Array)

G

- GCC – Linux prevajalnik kode (angl. GNU Compiler Collection)
- GNU – operacijski sistem (angl. Gnu's Not Unix)
- GPIO – splošni vhod/izhod (angl. General-Purpose Input/Output)
- GZIP – protokol za arhiviranje (angl. GNU ZIP)

H

- HDL – hardversko opisni jezik (angl. Hardware Description Language)
- HK – preostala funkcionalnost FPGA (angl. Housekeeping)
- HV – visoka napetost (angl. High Voltage)

I

- I2C – vrsta serijske komunikacije (angl. Inter-Integrated Circuit)
- ID – identifikacija (angl. Identification)
- IEEE – organizacija (angl. Institute of Electrical and Electronics Engineers)
- IN – okrajšava za vhod (angl. input)
- IO – vhod-izhod (angl. Input Output)
- IP – vrsta bloka v Vivadu (angl. Intellectual Property)

L

- LCR – vrsta inštrumenta (angl. Inductance (L), Capacitance (C), Resistance (R))
- LED – (angl. Light Emitting Diode)
- LF – (angl. Line Feed)
- LNIV – Laboratorij za Načrtovanje in Integracijo Vezij
- LV – nizka napetost (angl. Low Voltage)

M

- MAC – unikaten hardvareski naslov naprave (angl. Media Access Control)
- MIMO – več-vhodno in več-izhodno (vezje) (angl. Multiple-Input Multiple-Output)

O

- OS – operacijski sistem (angl. Operating System)
- OUT – okrajšava za izhod (angl. output)

P

- PID – (angl. Proportional-Integral-Derivative)

S

- Sa/s, Sps – vzorcev na sekundo (Samples per second)
- SCPI – (angl. Serial Commands for Programmable Instruments)
- SDR – softvaresko definiran radio (angl. Software-Defined Radio)
- SD – spominska kartica (angl. Secure Digital)
- SFTP – vrsta omrežnega protokola (angl. Secure File Transfer Protocol)
- SMA – vrsta priključka (angl. Surface Mount Assembly)
- SoC – sistem na čipu (angl. System-on-a-Chip)
- SPI – vrsta serijske komunikacije (angl. Serial Peripheral Interface)
- SSH – vrsta omrežnega protokola (angl. Secure Shell)
- STEM – (angl. Science, Technology, Engineering and Mathematics)

T

- TAR – protokol za arhiviranje (angl. Tape Archive)
- TCL – programski jezik za nadzor Vivada (angl. Tool Command Language)

U

- UART – vrsta serijske komunikacije (angl. Universal Asynchronous Receiver Transmitter)
- URL – naslov spletne strani (angl. Uniform Resource Locator)
- USB – univerzalno serijsko vodilo (angl. Universal Serial Bus)

V

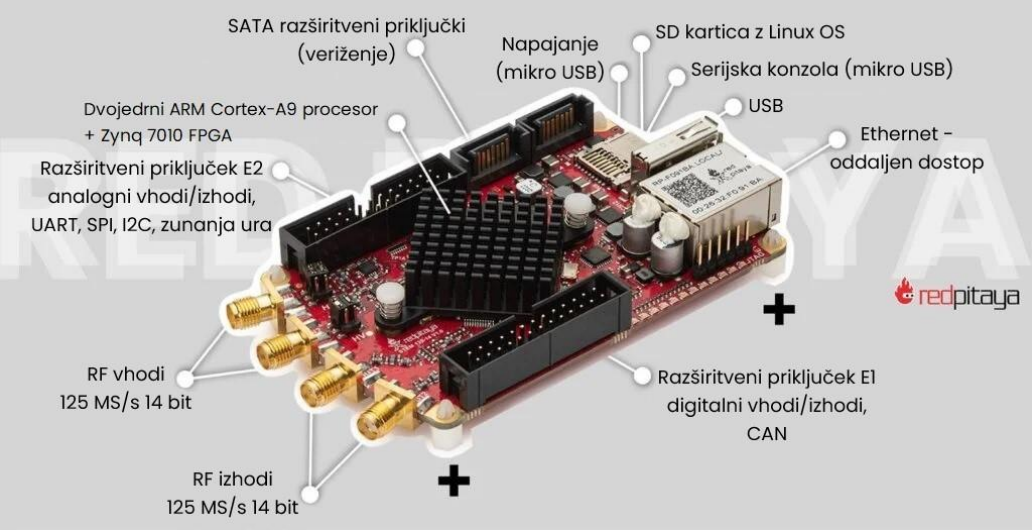
- VHDL – vrsta HDL (angl. VHSIC Hardware Description Language)
- VNA – omrežni analizator (angl. Vector Network Analyzer)
- V_{pp} – Medtemenska napetost (angl. peak-to-peak voltage)
- VSC/VS Code – program za urejanje kode (angl. Visual Studio Code)

Z

- ZIP – format za arhiviranje. (sopomenka angl. “speed”). Ni kratica.

1 Uvod v STEMlab 125-14

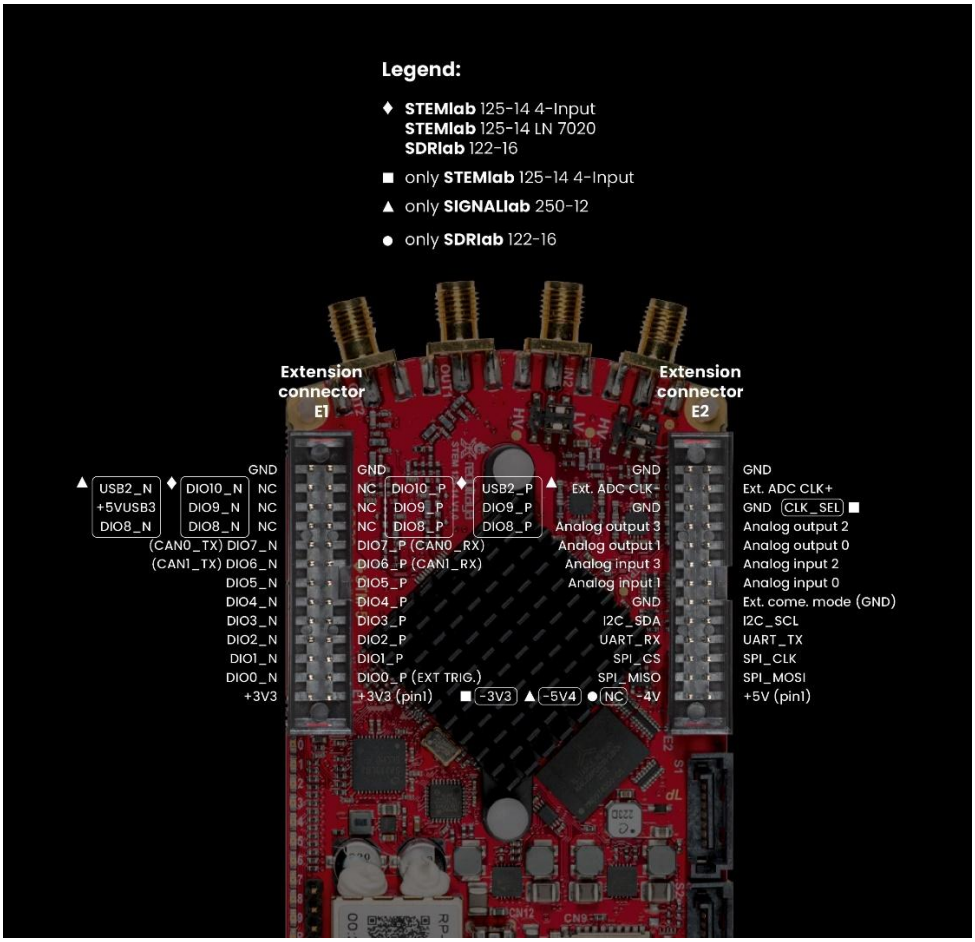
STEMlab 125-14 [Slika 1] je odprtokodna FPGA razvojna plošča, ki združuje 14-bitne 125 MHz analogne vhode in izhode z FPGA čipom (AMD Xilinx Zynq 7010 SoC), Ubuntu Linux operacijskim sistemom in možnostjo oddaljenega dostopa preko Etherneta. Na razširitvenih konektorjih razvojne plošče so digitalni vhodi in izhodi, ter priključki za digitalno komunikacijo preko UART, SPI, I2C, in CAN [Slika 2], kar uporabniku, skupaj s prej naštetimi lastnostmi, omogoča široko področje uporabe [1].



Slika 1: STEMlab 125-14 glavne značilnosti. Prilagojeno iz [1].

Osnovni namen uporabe je izvedba več digitalnih inštrumentov (osciloskop, spektralni analizator, Bodejev analizator, itd.), ki so na voljo kot aplikacije dostopne preko spletnega vmesnika. Naprednejši načini uporabe vključujejo oddaljeno kontroliranje inštrumentov v programskem okolju Python, MATLAB oziroma LabVIEW, pisanje lastnih programov v jeziku C ali Python, ki se izvajajo na Red Pitayi, ali popolnoma reprogramiranje FPGA [1], [2], [3], [4].

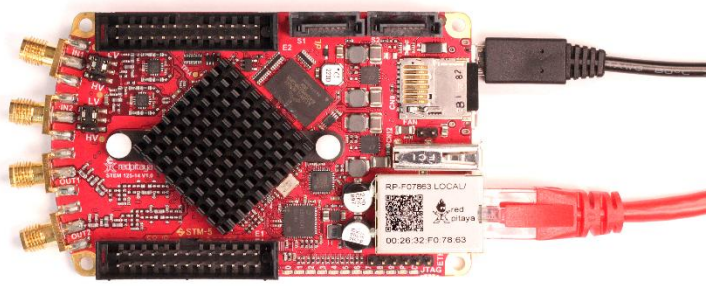
Njena glavna prednost pred konkurenčnimi produkti je odprta programska koda, kar uporabniku omogoča uporabo obstoječe kode za nadgradnjo ali popolno reprogramiranje. Zaradi lažjega razumevanja bomo v nadaljevaju STEMlab 125-14 [5] imenovali Red Pitaya, po nazivu slovenskega podjetja, ki razvojno ploščo proizvaja.



Slika 2: Shema digitalnih priključkov na Red Pitayi STEMlab 125-14 [5]

Preden se lahko uspešno povežemo na Red Pitayo moramo zagotoviti naslednje [6], [Slika 3]:

- Preverimo, da je SD kartica vstavljena v Red Pitayo.
- Priključimo napajanje v zunanji mikro USB konektor na spodnji strani Red Pitaye.
- Povežemo Red Pitayo v omrežje preko ethernet kabla.



Slika 3: Pravilna priključitev Red Pitaye [6]

1.1 Kako se povežemo na Red Pitayo?

Na Red Pitayo se lahko povežemo na dva načina:

- preko spletnega brskalnika [6],
- preko SSH (Secure Shell) povezave [7].

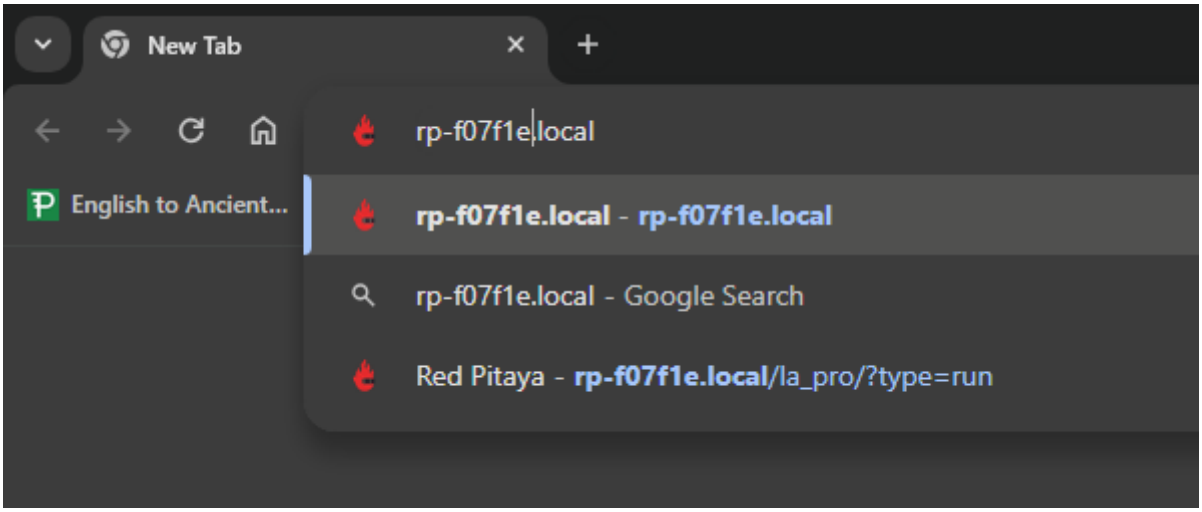
Ne glede na način povezave mora biti Red Pitaya na istem lokalnem omrežju kot naš računalnik. Napravo lahko povežemo neposredno na ethernet priključek računalnika ali na ethernet vtičnico, ki je povezana z brezžičnim usmerjevalnikom (router). Priporočljivo je, da je Red Pitaya, v primeru uporabe aplikacij, neposredno priključena v brezžični usmerjevalnik, saj to izboljša delovanje naprave. [8]

Kaj želimo narediti z Red Pitayo, močno vpliva na izbiro načina povezave.

1.1.1 Povezava preko brskalnika

Povezavo preko brskalnika uporabimo, ko želimo Red Pitayo uporabljati kot osnoven inštrument ali pa jo kontrolirati v okoljih Python, MATLAB, ali LabVIEW z našega računalnika. Google Chrome je priporočen brskalnik za povezavo [9].

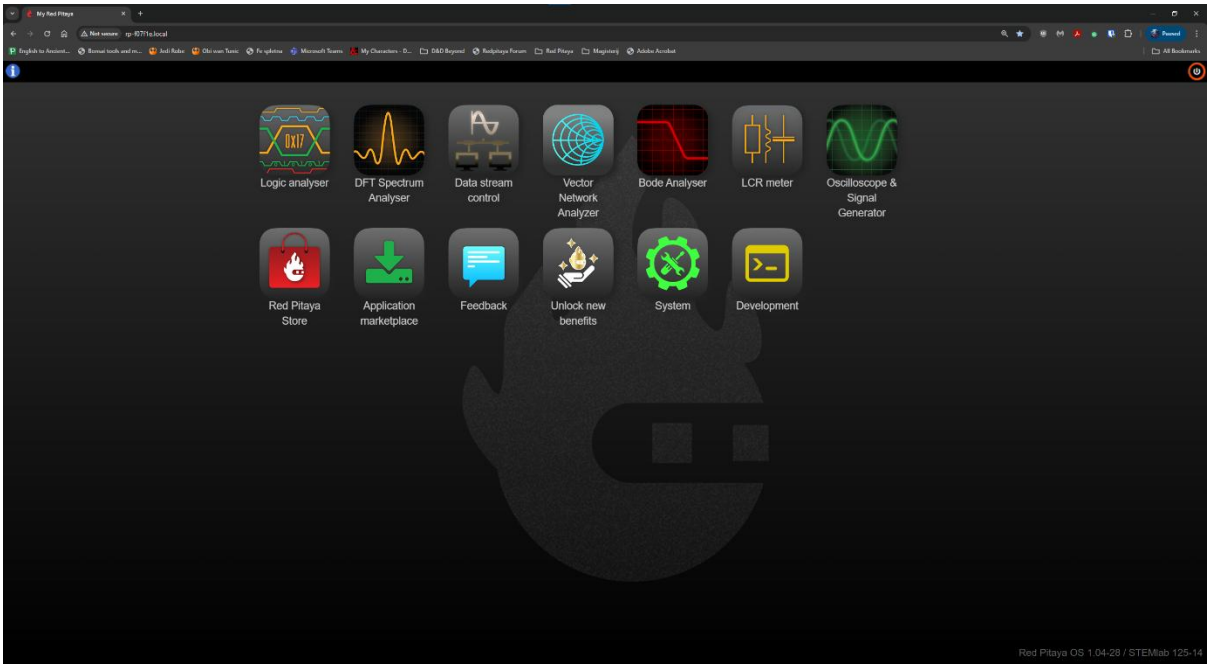
V splošnem se lahko na Red Pitayo povežemo z vpisom »***rp-xxxxxx.local***« v URL okno brskalnika, kjer »xxxxxx« predstavlja zadnjih šest znakov v MAC naslovu Red Pitaye [Slika 4]. MAC naslov je natisnjen na Ethernet konektorju. V ozadju se lokalni naslov prevede v IP naslov na katerem se trenutno nahaja Red Pitaya. [6]



Slika 4: *Povezava na Red Pitayo preko brskalnika*

V okviru teh vaj, pa so vse Red Pitaye že konfigurirane na statičen IP naslov, tako da povezava preko lokalnega naslova ne bo delovala. Za povezavo v URL okno brskalnika vpišemo statičen IP naslov »***192.168.1.15***«.

Ob uspešni povezavi se odpre internetni vmesnik Red Pitaye [Slika 5]:



Slika 5: Internetni vmesnik Red Pitaye

V primeru težav s povezavo

Preverite naslednje [9]:

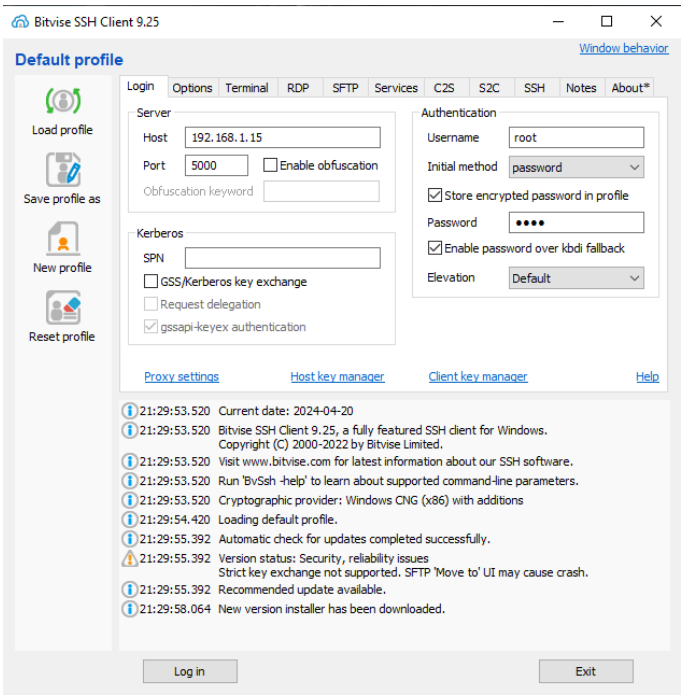
- Status LED indikatorjev. Zelena (napajanje) in modra (FPGA slika naložena) morata konstantno goreti, rdeča (stanje procesorja) utripa v ritmu srčnega utripa, oranžna pa vsake toliko časa oziroma, ko Red Pitaya dostopa do podatkov na SD kartici.
    - Modra LED ne gori – preverite če ste vklopili napajanje v pravi mikro USB konektor in če je SD kartica prisotna v napravi
- Preverite, če se Red Pitaya nahaja na istem lokalnem omrežju kot računalnik. To storite tako, da odprete Ukazni Poziv (CMD/Command Prompt/Terminal) in vanj vpišete »***ping 192.168.1.15***« oziroma »***ping rp-xxxxxx.local***«.
- Red Pitaya potrebuje približno 30 sekund po priključitvi napajanja, da se zažene in je vidna na omrežju.
- Izklopite blokiranje oglasov (adblocker-je) v brskalniku, saj lahko vplivajo na pravilno delovanje internetnega vmesnika.
- V primeru, da se aplikacije velikokrat ponovno zaganjajo in je posledično izvajanje meritev težko izvedljivo, poizkusite priklopiti Red Pitayo v brezžični usmerjevalnik, saj je spletni vmesnik v tem načinu bolj stabilen.

1.1.2 SSH povezava

SSH povezavo uporabimo, ko želimo izvajati programe v jeziku C oziroma Python na sami Red Pitayi, dostopati do podatkov in operacijskega sistema Linux ali reprogramirati FPGA.

V okviru vaj se bomo na operacijski sistem Red Pitaye povezali s programom ***Bitvise SSH Client*** [10], saj tako lažje prenašamo datoteke na in iz Red Pitaye. Enako lahko storimo tudi direktno preko Ukaznega poziva (terminala), kar bomo omenili na koncu tega poglavja.

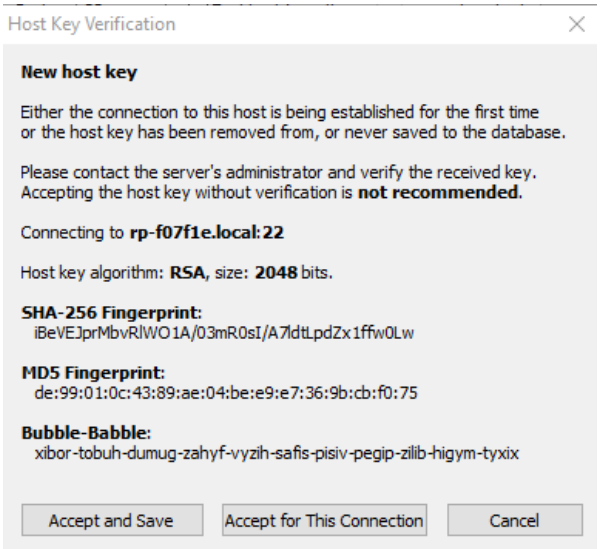
Pri zagonu programa se odpre okno na [Slika 6].



Slika 6: Program Bitvise SSH

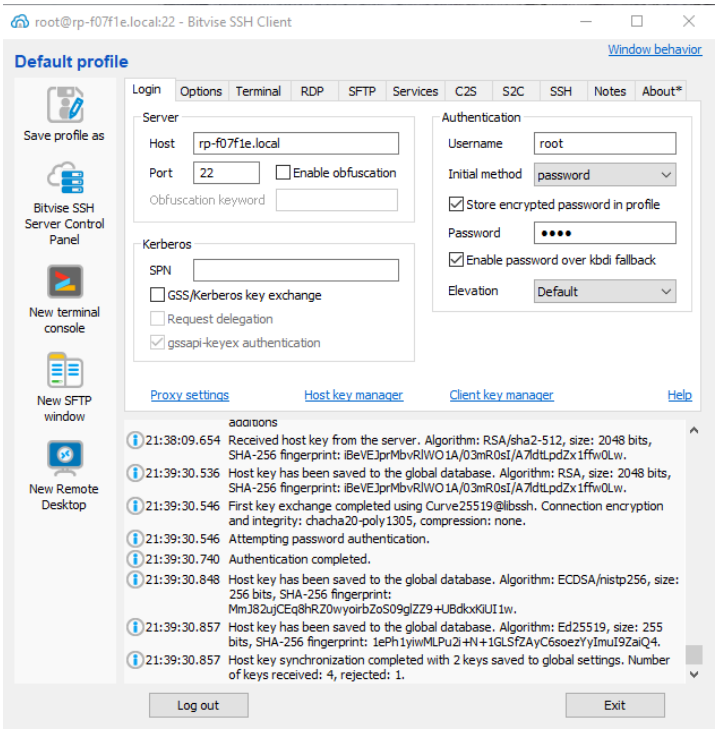
V polje »***Host***« vpišemo IP naslov Red Pitaye (***192.168.1.15***), preverimo polje »***Port***« (***22***), v polje »***Username***« vpišemo ***root***, ter v polje »***Password***« prav tako ***root***.

Ob kliku na gumb »***Log in***« se ob prvi povezavi pojavi pojavno okno na [Slika 7]:



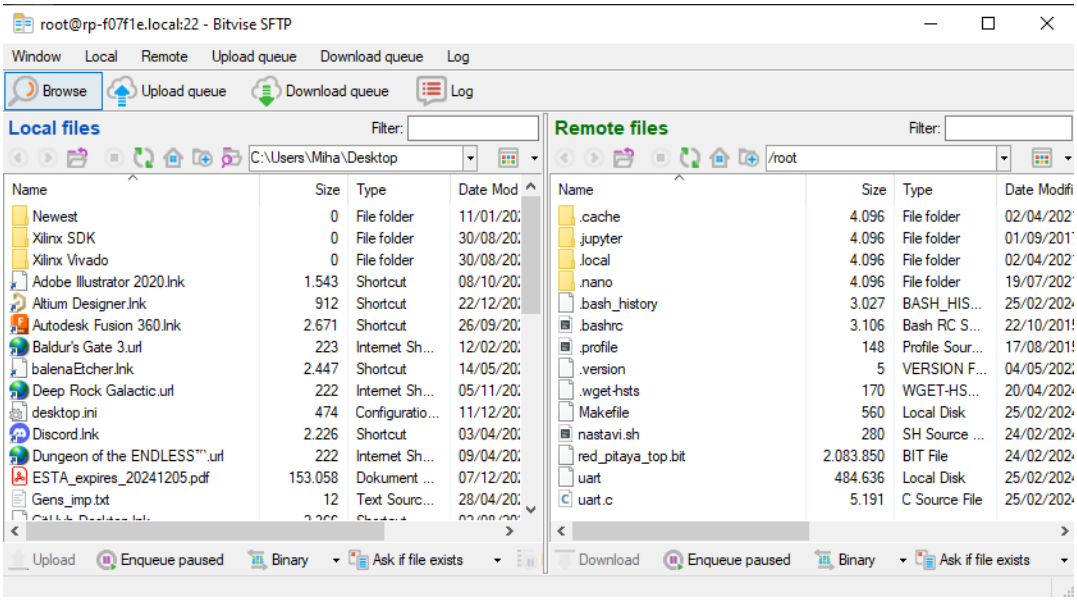
Slika 7: Prva povezava preko SSH

Klinemo »***Accept and Save***«. Opazimo lahko, da se okno programa sedaj spremeni [Slika 8]:



Slika 8: Bitvise SSH po vpisu

Sedaj lahko do terminala na Red Pitayi dostopamo s klikom na »***New terminal console***«. Za prenos datotek pa uporabimo »***New SFTP window***«, kjer lahko datoteke med našim računalnikom in Red Pitayo prenašamo tako, da jih povlečemo iz ene na drugo stran:

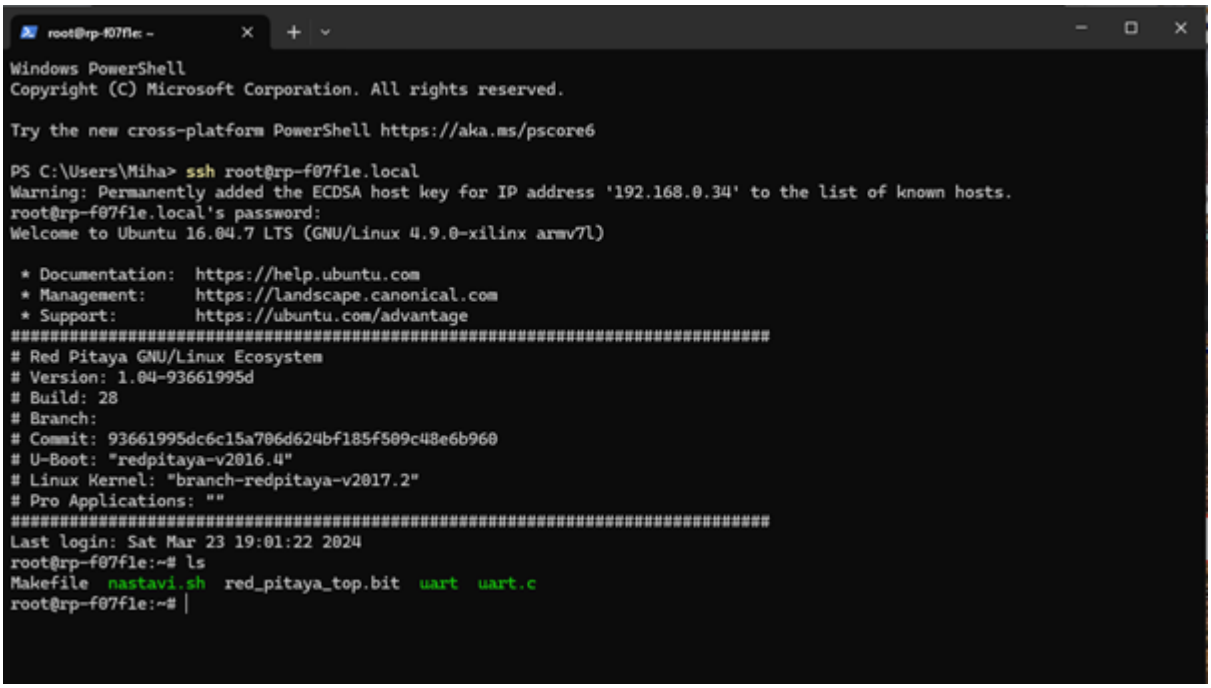


Slika 9: Prenos datotek

Na levi se nahaja datotečni sistem našega računalnika, na desni strani pa datotečni sistem Red Pitaye [Slika 9].

1.1.3 Povezava preko Ukaznega Poziva

Na Red Pitayo se lahko povežemo tudi direktno preko ukaznega poziva. To storimo tako, da v okno Ukaznega Poziva vpišemo »***ssh root@192.168.1.15***« [Slika 10]. Po vpisu bomo pozvani za vpis gesla (»***root***«) [7]. Če se na Red Pitayo s posameznega računalnika vpisujemo prvič, moramo potrditi povezavo z ECDSA ključem.



Slika 10: Povezava preko ukaznega poziva

Od tu naprej lahko za prenos datotek med računalnikom in Red Pitayo uporabljamo Linux ukaze.

Tukaj je hiter seznam najpomembnejših:

- cd – pomik med datotečnim sistemom
- ls – izpis datotek v trenutnem direktoriju
- nano – odprtje datoteke v urejevalniku besedila
- rm – izbris datotek
- pwd – pot trenutnega direktorija
- exit – zaprtje povezave

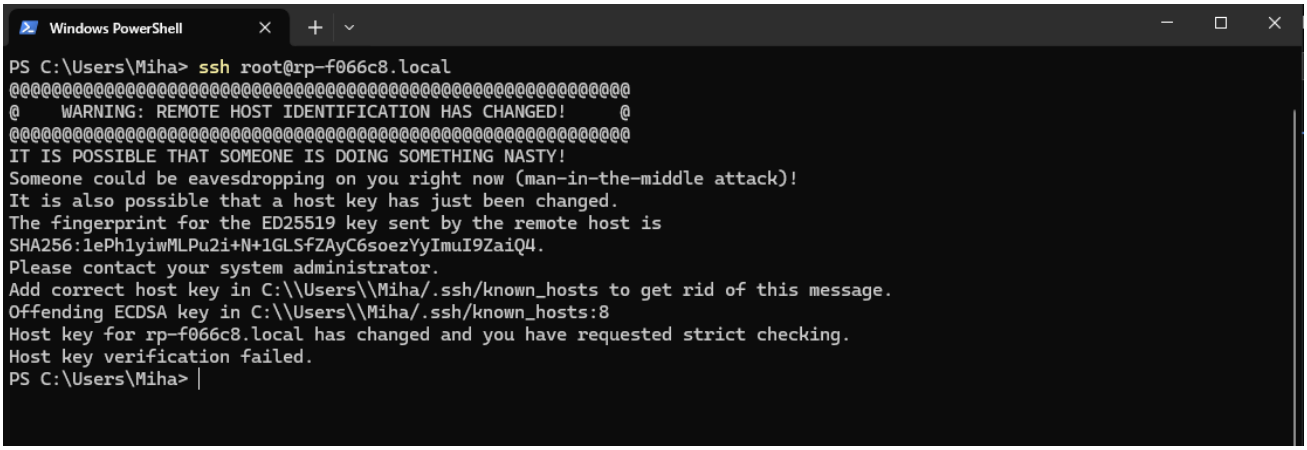
Za prenos datotek med Red Pitayo in računalnikom odprite še eno okno Ukaznega poziva, vendar se ne povežite na Red Pitayo. Uporabite ukaz »***scp***«, tukaj je primer:

»***scp -r pot/do/datoteke/na/računalniku root@192.168.1.15:/root***« za prenos datoteke iz računalnika na Red Pitayo

»***scp -r root@192.168.1.15:/root/pot/do/datoteke/na/RedPitayi pot/do/datoteke/na/računalniku***« za prenos datoteke iz Red Pitaye na računalnik

1.2 Težave s povezavo

V primeru, da se večkrat povežete na Red Pitayo, se lahko zgodi, da pri vzpostavitvi SSH povezave naletite na naslednje obvestilo [Slika 11]:



Slika 11: Obvestilo o neuspešni potrditvi ključa

Ne skrbite nič ni narobe z Red Pitayo ali z vašim računalnikom, vse kar se je zgodilo v ozadju je menjava IP naslova, ki ga brezžični vmesnik dodeli Red Pitayi. Za popravilo v Ukazni poziv napišite:

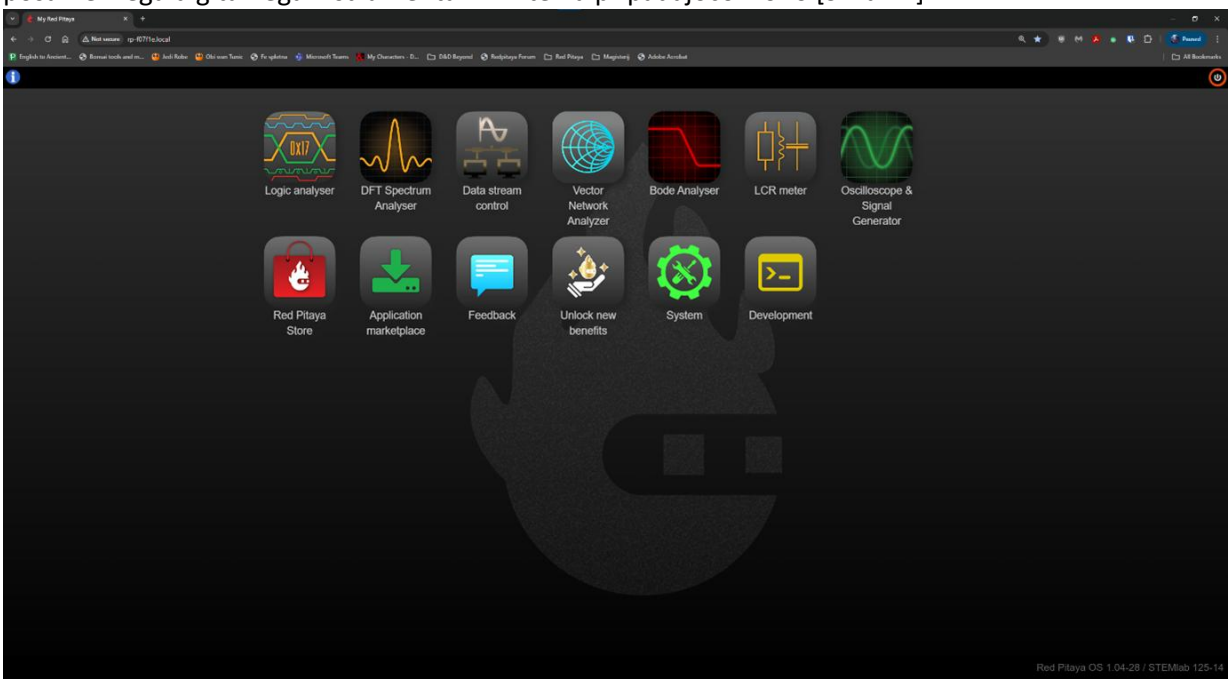
»ssh-keygen -R 192.168.1.15« oziroma »ssh-keygen -R rp-xxxxxx.local«

S čimer izbrišete trenutni ECDSA ključ in nato ponovno vzpostavite SSH povezavo.

1.3 Uvod v aplikacije na Red Pitayi

V prejšnjem poglavju smo se naučili kako se lahko povežemo na Red Pitayo, tokrat pa se bomo posvetili digitalnim inštrumentom na Red Pitayi kot sta osciloskop in signalni generator ter Bodejev analizator, ki jih bomo uporabljali tekom vaj.

Za dostop do digitalnih inštrumentov se moramo na Red Pitayo povezati preko spletnega brskalnika. Za zagon posameznega digitalnega inštrumenta kliknite na pripadajočo ikono [Slika 12].



Slika 12: Spletni vmesnik Red Pitaye

V uporabniškem priročniku sta podrobneje predstavljena osciloskop in Bodejev analizator. Večina ostalih digitalnih inštrumentov ima skoraj identične nastavitve in način upravljanja kot osciloskop, zato z njimi ne boste imeli večjih težav.

Za zagon vektorskega analizatorja omrežja (VNA – »Vector Network Analyzer«) in LCR metra je potrebna dodatna razširitvena plošča za Red Pitayo, ki omogoči funkcionalnost teh dveh inštrumentov.

Za dostop do tržnice aplikacij (»Application marketplace«), potrebuje Red Pitaya povezavo na internet, npr. preko brezžičnega usmerjevalnika.

Težave z Red Pitayo

Če imate težave z Red Pitayo, ki jih ne morete rešiti z katero izmed zgoraj navedenih opcij ali ponovno namestitvijo operacijskega sistema:

- najprej se posvetujte z profesorjem
- v skrajnem primeru pišite na support@redpitaya.com. V sporočilu podajte trenutni operacijski sistem Red Pitaye, model razvojne plošče, ter podroben opis napake.

2 Namestitev operacijskega sistema

V primeru, da ste reprogramirali Red Pitayo, ponesreči izbrisali ali modificirali del operacijskega sistema in ne veste, oziroma se ne morete vrniti v prejšnje stanje, ali se na Red Pitayo ne morete več povezati, je najenostavnejša rešitev težav izbris in ponovna namestitev operacijskega sistema.

Pred začetkom ponovne namestitve operacijskega sistem je priporočljivo, da naredite kopije vseh pomembnih datotek in jih shranite na vaš računalnik, saj bodo vse datoteke in spremembe v programski kodi, ki se nahajajo na Red Pitayi, izbrisane.

Za namestitev operacijskega sistema boste potrebovali čitalnik mikro SD kartic in program ***balenaEtcher***, ki ga lahko dobite tukaj [11].

2.1 Nalaganje slike operacijskega sistema

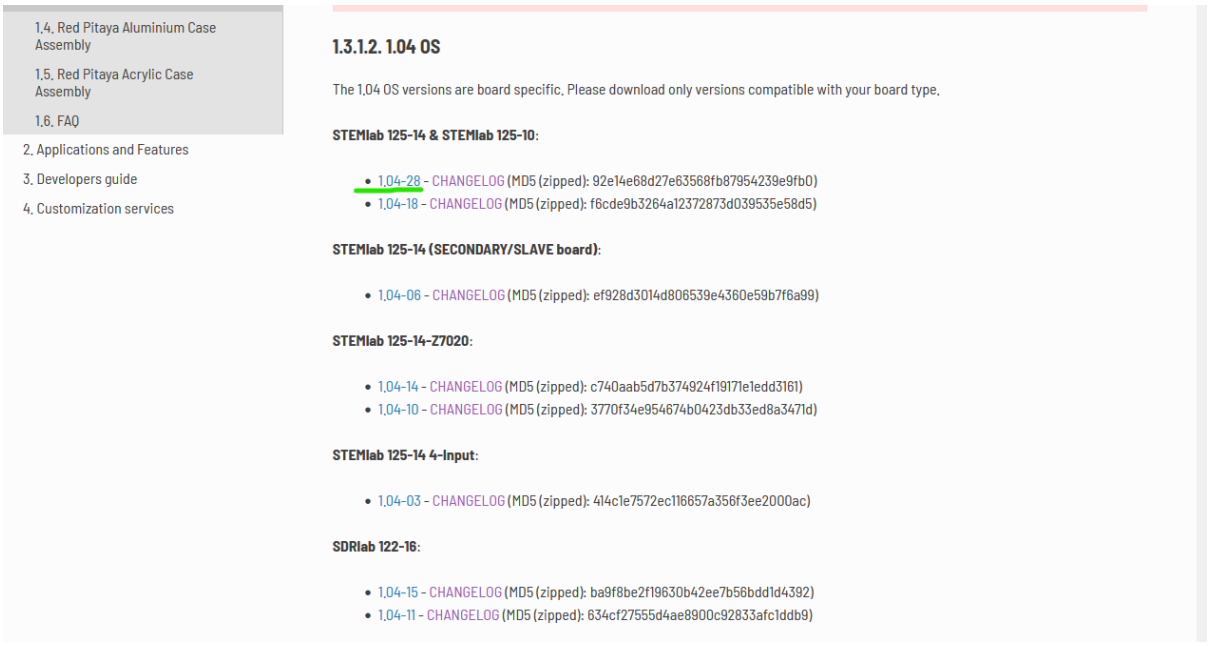
V sklopu vaj se uporablja Red Pitaya operacijski sistem 1.04-28. Verzijo operacijskega sistema in model razvojne plošče Red Pitaya lahko preverite v spodnjem desnem kotu spletnega vmesnika [Slika 13].

Slika 13: Verzija operacijskega sistema in model Red Pitaye na spletnem vmesniku

Slike Red Pitayinega operacijskega sistema so dostopne tukaj .

Operacijski sistemi so urejeni po poglavjih od najnovejših do najstarejših. Zelo pomembna je razlika med OS 2.00 in OS 1.04, saj je OS 2.00 enak za vse modele Red Pitaye (STEMlab 125-14, SDRlab 122-16, itd.), medtem ko ima OS 1.04 svojo različico za vsak posamezen model Red Pitaye [12].

Pri vajah uporabljamo OS 1.04-28, ki ga najdete v poglavju »1.04 OS« pod »STEMlab 125-14 & STEMlab 125-10«:

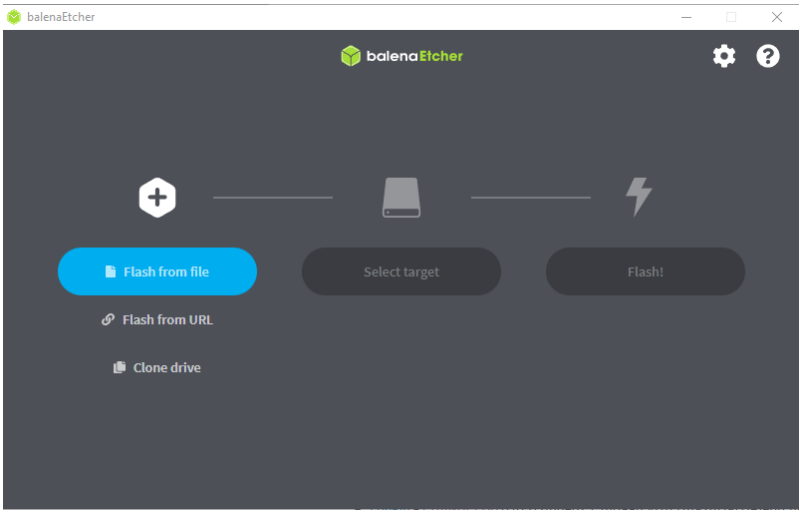


Slika 14: Lokacija OS 1.04-28

Operacijski sistem prenesete s klikom na verzijo operacijskega sistema (označeno z zeleno na [Slika 14]). Velikost stisnjene datoteke je približno 0,7 GB. Po prenosu stisnjene datoteke ni potrebno razširjati, saj program *balenaEtcher* to naredi avtomatsko tekom postopka namestitve.

2.2 Prenos operacijskega sistema na SD kartico

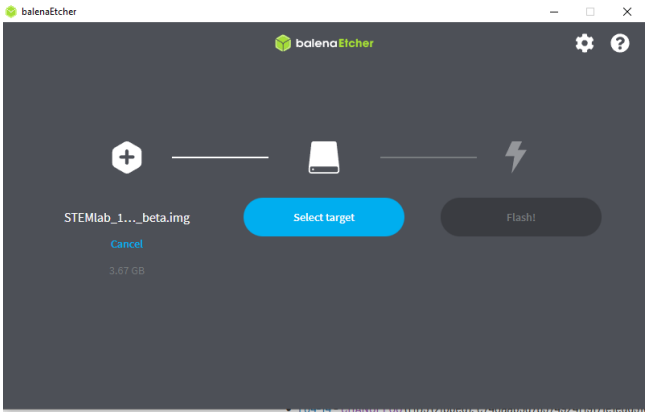
Vstavite mikro SD kartico v čitalnik SD kartic in ga priključite v računalnik. Odprite program *balenaEtcher* [Slika 15].



Slika 15: Program balenaEtcher

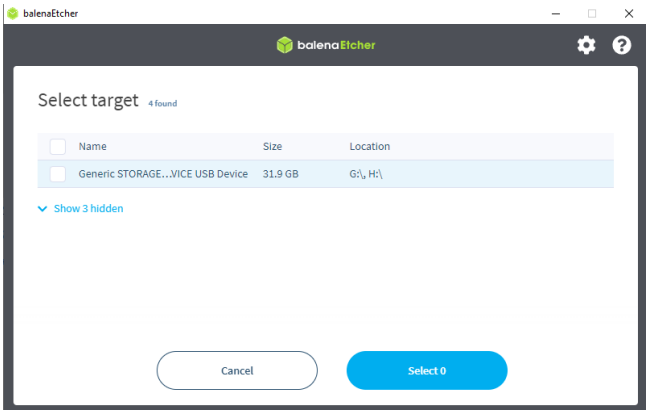
Postopek prenosa poteka v treh korakih:

1. **Izbira datoteke** – Klinknite na gumb »Preslikaj iz datoteke« (»Flash from file«) in poiščite preneseno stisnjeno datoteko z operacijskim sistemom [Slika 16].



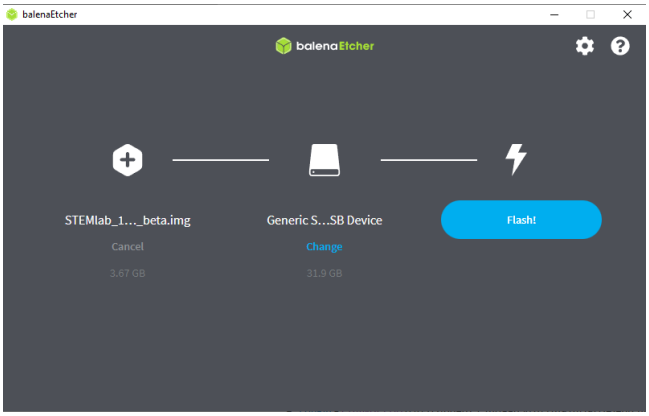
Slika 16: Izbira datoteke z operacijskim sistemom

2. **Izbira tarče** – Kliknite na gumb »Izberi tarčo« (»Select target«) in izberite disk z priključeno mikro SD kartico. Program avtomatsko skrije vse sistemske diske pod zavihek »Pokaži skrite« (»Show hidden«) in jih označi z rdečo, kar minimizira verjetnost preslikave na napačno mesto. Odkljukajte izbrani disk in pritisnite »Izberi« (»Select«) [Slika 17].



Slika 17: Izbira tarče preslikave

3. Preslikava – Pritisnite gumb »Preslikaj!« (»Flash!«) [Slika 18].

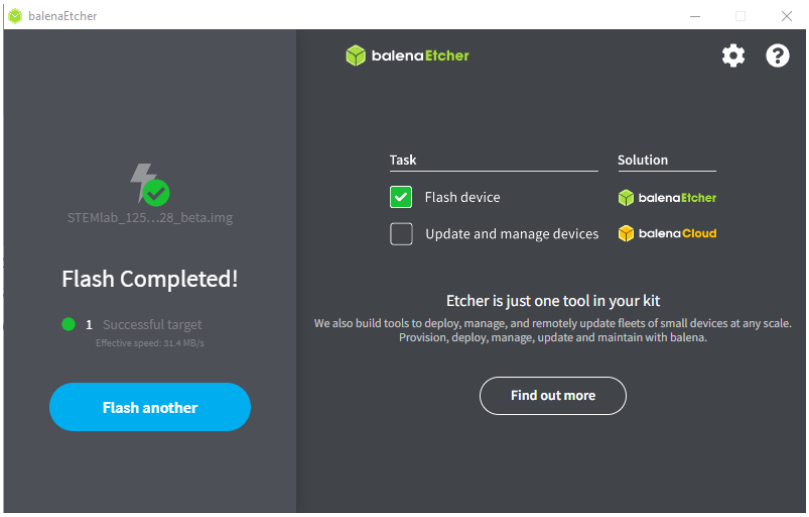


Slika 18: Preslikaj

Pojavilo se bo sistemsko pojavno okno za potrditev procesa, kjer izberite »Potrdi«. Nato, pa se bo preslikava izvedla v treh korakih:

- Razširitev slike operacijskega sistema (modra)
- Zapisovanje slike na mikro SD kartico (vijolična)
- Preverjanje zapisa (zelena)

Med postopkom se bodo pojavila različna pojavna okna datotečnega sistema, z zahtevami po formatiranju diska, neuspešno odprtimi datotekami, itd., saj bo Windows neuspešno poizkušal odpreti Linux datoteke na mikro SD kartici. Vsa pojavna okna lahko brez skrbi zaprete oziroma prekličete operacije. Ob končanem prenosu boste zagledali sledeče okno [Slika 19].



Slika 19: Prenos končan

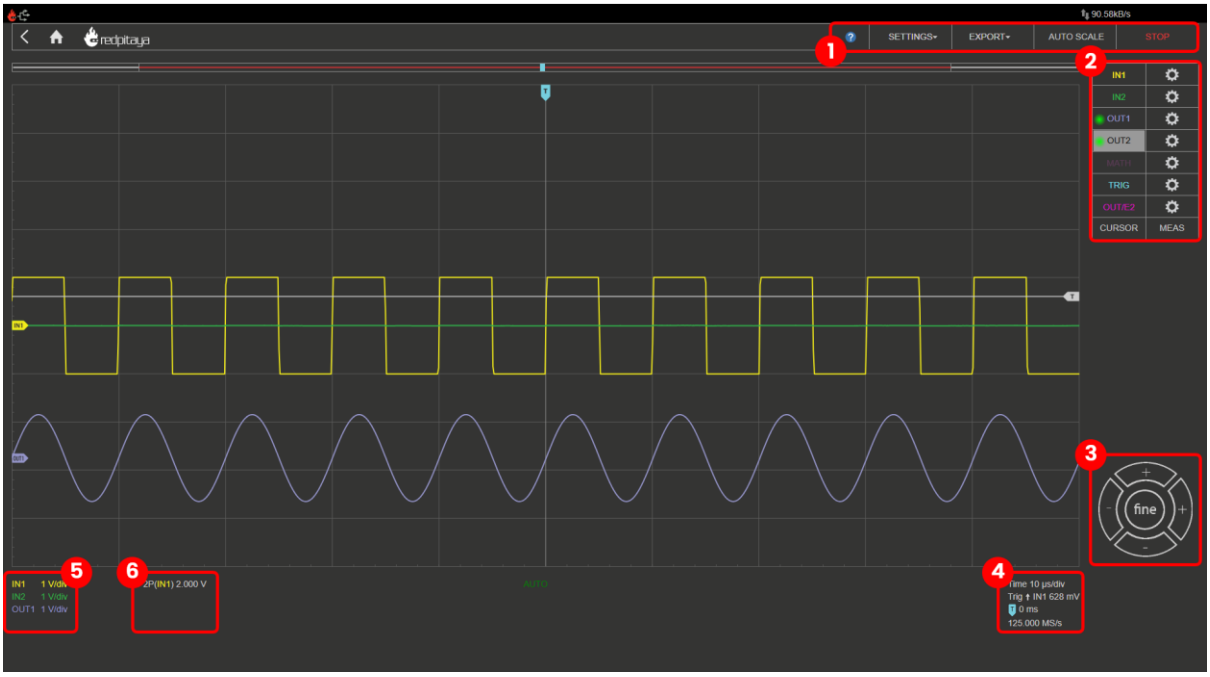
Lahko se zgodi, da bo program sporočil neuspešen prenos (značilno pri preslikavi 2.00 OS), kar pa ne vpliva na sam prenesen operacijski sistem. Težavo lahko odpravimo z izbrisom particij na SD kartici preko programa »Upravitelja računalnika« (angl. »Computer Manager«) in ponovno preslikavo OS.

Zadnji korak je odklopitev čitalnika kartic iz računalnika (*balenaEtcher* avtomatsko odstrani USB napravo) in vstavitev mikro SD kartice nazaj v Red Pitayo.

3 Osciloskop in signalni generator

Pri zagonu te aplikacije Red Pitayo spremenimo v dvokanalni osciloskop in signalni generator. Vsak kanal ima vzorčno frekvenco 125 MSa/s, 14-bitno resolucijo, ter je neodvisen od ostalih kanalov. Teoretična pasovna širina osciloskopa je 62 MHz (Nyquistova frekvenca), vendar v praksi dobro deluje do maksimalno 20 MHz. Programsko smo omeji na zajem in generacijo signalov do 50 MHz.

Nekatere nastavitve prikazane na slika v tem poglavju niso na voljo na naši različici operacijskega sistema (1.04-28), saj so slike vzete iz Red Pitayine spletne strani, ki prikazuje novejšo različico inštrumenta [13].



Slika 20: Osciloskop in signalni generator [13]

Območje grafa na [Slika 20] prikazuje vhodne in izhodne signale. Na robovih grafa lahko vidimo več kurzorjev, ki so povezani s posameznimi signali:

- **Časovni zamik** – moder navpičen kurzor označen s T na zgornjem robu grafa
- **Nivo proženja** – bel vodoraven kurzor označen s T na desnem robu grafa
- **Ničelni nivoji kanalov** – kurzorji na levem robu grafa v barvah posameznih kanalov

Kurzorje lahko premikamo z klikom in povlekom. Časovni zamik in nivo proženja skupaj določata prožilec oziroma točko proženja. Ničelni nivoji kanalov se večinoma uporabljajo za razporeditev signalov po grafu za lažjo primerjavo in izvedbo meritev [13].

Celoten graf je razdeljen na 10x10 pravokotnikov, kjer vsak pravokotnik predstavlja en razdelek [Slika 20]. Tako enote časovne kot napetostne skale so podane v sekundah oziroma voltih na razdelek (pravokotnik) s čimer dobimo predstavo o velikosti signalov.

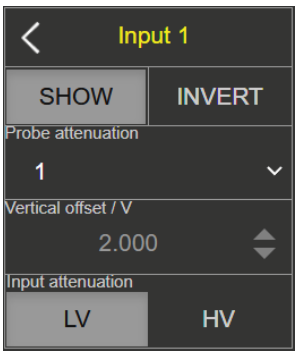
Ob prvem zagonu aplikacije se lahko zgodi, da nekateri izmed kanalov niso vidni, kar rešimo tako da označimo gumb »*prikaz*« (»*show*«) pod nastavitvami posameznih kanalov.

Spletni vmesnik osciloskopa in signalnega generatorja je, poleg območja glavnega grafa razdeljen na šest delov [Slika 20] [13]:

1. **Meni z glavnimi nastavitvami aplikacije** – Zaustavitev in zagon inštrumenta, izvoz grafa in avtomatsko skaliranje.
2. **Nastavitve kanalov** – Urejanje nastavitev posameznih vhodnih in izhodnih kanalov, prožilnika, kurzorjev in meritev.
3. **Skaliranje osi** – Spreminjanje skal za amplitudo in čas
4. **Podatki o časovni skali in proženju** – Prikaz trenutne nastavitve časovne skale, nastavitev prožilnika in vzorčne frekvence
5. **Amplitudna skala kanalov** – amplitudna skala vsakega izmed prikazanih kanalov
6. **Prikaz meritev** – prikaz trenutno nastavljenih meritev

3.1 Nastavitve osciloskopa in generatorja

Do nastavitev posameznih kanalov osciloskopa dostopamo s klikom na zobnik poleg želenega kanala.



V nastavitvah vhodnih kanalov [Slika 21] lahko nastavimo:

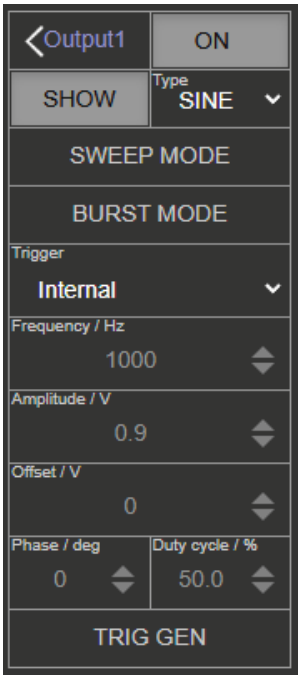
- Prikaz vhodnega kanala na grafu
- Invertiramo prikaz na grafu
- Konfiguriramo atenuacijo sond
- Spreminjamo vertikalni zamik kanala
- Nastavimo vhodno atenuacijo, ki naj bo enaka kot položaj vhodnih mostičnikov direktno za SMA priključki.

Slika 21: Nastavitve vhodov [13]

V nastavitvah izhodov [Slika 22] lahko nastavimo:

- Aktivnost izhoda (ON/OFF)
- Prikaz izhoda na grafu
- Obliko izhodnega signala (sinusni, trikoten, kvadratni, žagasti, itd.)
- Nastavimo preletni (»sweep«) ali pulzni (»burst«) način delovanja
- Izberemo tip prožilca (notranji ali zunaji)
- Frekvenco izhodnega signala (1 Hz – 50 MHz)
- Amplitudo (0-1 V)
- Odmik (»offset«) – vsota amplitude in odmika ne sme preseči maksimalne oziroma minimalne izhodne napetosti +-1 V.
- Fazo
- Obratovalni cikel (»Duty cycle«)

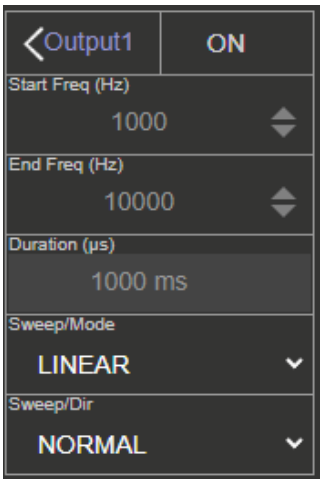
Signal se na izhodu začne generirati s pritiskom na gumb »ON«. S klikom na »TRIG GEN« lahko ponovno sprožimo generacijo signala, kar je koristno v pulznem načinu delovanja.



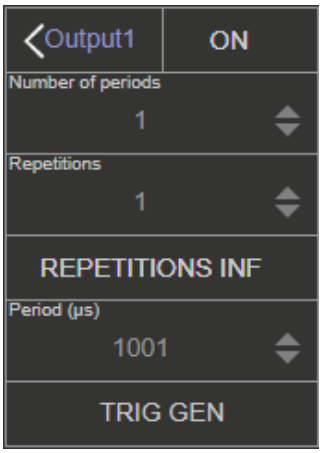
Slika 22: Nastavitve izhodov [13]

V nastavitvah preletnega (»sweep«) načina [*Slika 23*] lahko določimo:

- Začetno in končno frekvenco preleta
- Čas preleta
- Način preleta (linearni ali logaritmičen)
- Smer preleta (normalna/gor ali gor-dol)



Slika 23: Preletni način [13]



Slika 24: Pulzni način [13]

V nastavitvah pulznega (»burst«) načina [Slika 24] lahko določimo:

- Število period v enem pulzu
- Število ponovitev (število vseh pulzov)
- Neskončno pulzov (»repetitions inf«) – z označbo se pulzni signal ob sprožitvi ponavlja v neskončnost
- Perioda pulzov – čas med začetkom prvega in začetkom naslednjega pulza
- Sprožitev generatorja (»trig gen«)

POZOR: ob vklopu preletnega ali pulznega načina le-ta ostane vklopljen dokler ga ne izklopimo!
Vklop in izklop generatorja oziroma izhod iz aplikacije ne resetira nastavitev.

3.1.1 Nastavitve matematičnega kanala

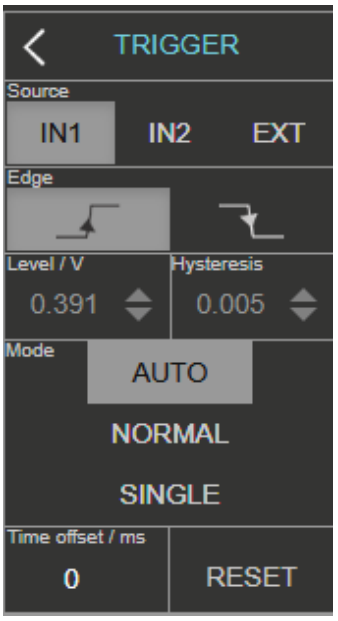
V matematičnem kanalu lahko izvajamo preproste matematične operacije nad obema vhodnima signaloma. Na primer, seštevanje, odštevanje, množenje, integracija, itd.

3.1.2 Nastavitve prožilnika

Prožilnik (»trigger«) je izjemno pomemben parameter za osciloskop, saj določa pogoj za zajem vhodnih signalov, ter v pravilni nastavitvi ohranja stabilno sliko grafa.

V nastavitvah prožilnika [Slika 25] lahko nastavimo:

- **Vir proženja** (vhod 1, vhod 2, ali zunanji digitalni vhod DIO0_P)
- **Fronta proženja** – pozitivna ali negativna
- Nivo proženja
- **Histereza proženja** – minimalna vrednost za katero mora biti presežen nivo proženja, da se prožilnik sproži. Hkrati pa preprečuje zaporedna proženja zaradi šuma.
- Način proženja:
    - *Avtomatski* – graf se osvežuje avtomatsko
    - *Normalen* – graf se osveži vsakič, ko so doseženi pogoji proženja
    - *Enkraten (»single«)* – graf se osveži ob prvem pogoju proženja, nato pa se ne osvežuje več (zajem podatkov se ustavi).
- **Časovni zamik** z ponastavitvijo (»reset«)



Slika 25: Meni proženja [13]

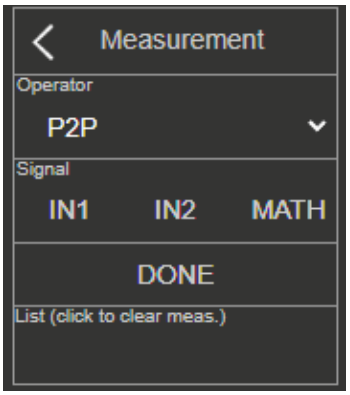
Časovni zamik, fronta proženja in nivo proženja skupaj določata prožilec oziroma točko proženja, ki jo na grafu določa presečišče kurzorja za časovni zamik in kurzorja za nivo proženja.

3.1.3 Nastavitve kurzorjev

Na vsako os lahko dodamo po dva kurzorja, ki olajšata meritve. Na vsakem kurzorju je zapisana vrednost osi, v primeru dveh kurzorjev na isti osi pa tudi razdalja med njima.

Vsi kurzorji so vezani na enega izmed vhodnih kanalov ali matematični kanal.

3.2 Meritve



Slika 26: Meni meritev [13]

V meniju za meritve [Slika 26] lahko nastavimo meritve, ki jih želimo izvajati na vhodnih signalih. Najprej izberemo želeno meritev iz spustnega menija. Izbiramo lahko med medtemensko, maksimalno, srednjo, minimalno in RMS napetostjo, ter frekvenco, periodo, ter obratovalnim ciklom. Nato izberemo kanal na katerem želimo izvajati meritve, ter kliknemo »zaključi« (»done«).

Vse izvajajoče se meritve so zapisane v razdelku 6 na glavnem grafu. Meritev lahko pobrišemo tako da kliknemo nanjo na listi meritev.

3.3 Splošna priporočila za osciloskop

1. Avtomatsko skaliranje grafa nam velikokrat naredi več dela, kot pri ročnem skaliranju osi. Predvsem, če na enem izmed vhodov nimamo signalov, saj takrat avtomatsko skaliranje v iskanju optimalnih nastavitev popolnoma prebliža amplitudno skalo.
2. Pri merjenju pulznih signalov, je izredno pomembno, da preklopimo osciloskop v normalen način proženja, saj so pulzni signali velikokrat izjemno kratki in jih v avtomatskem načinu zamudimo.
3. Če je osciloskop v enkratnem načinu proženja, ga je potrebno po vsaki meritvi ponovno zagnati s klikom na gumb »run« v zgornjem desnem kotu.
4. Če generator generira preletni (»sweep«) ali pulzirajoč (»burst«) signal, in ga želimo preklopiti nazaj v način neprekinjene generacije, moramo izklopiti pulzirajoč oziroma pulzni način v istoimenskem meniju.
5. Pri generaciji in merjenju visokofrekvenčnih signalov (nad 100 kHz) je priporočena uporaba 50-Ω zaključnih uporov na hitrih analognih izhodih, s čimer zaključimo izhodno impedanco generatorja. To velja tudi pri splošni uporabi Red Pitayinega generatorja, saj v nasprotnem primeru lahko pride do odbojev signala na prenosni liniji.
6. Nikoli ne priključite sond za osciloskop na Red Pitayin generator. Sonde so namenjene merjenju signalov. Z generacijo signala preko sond, le-tega popačite in posledično pridete do napačnih meritev.

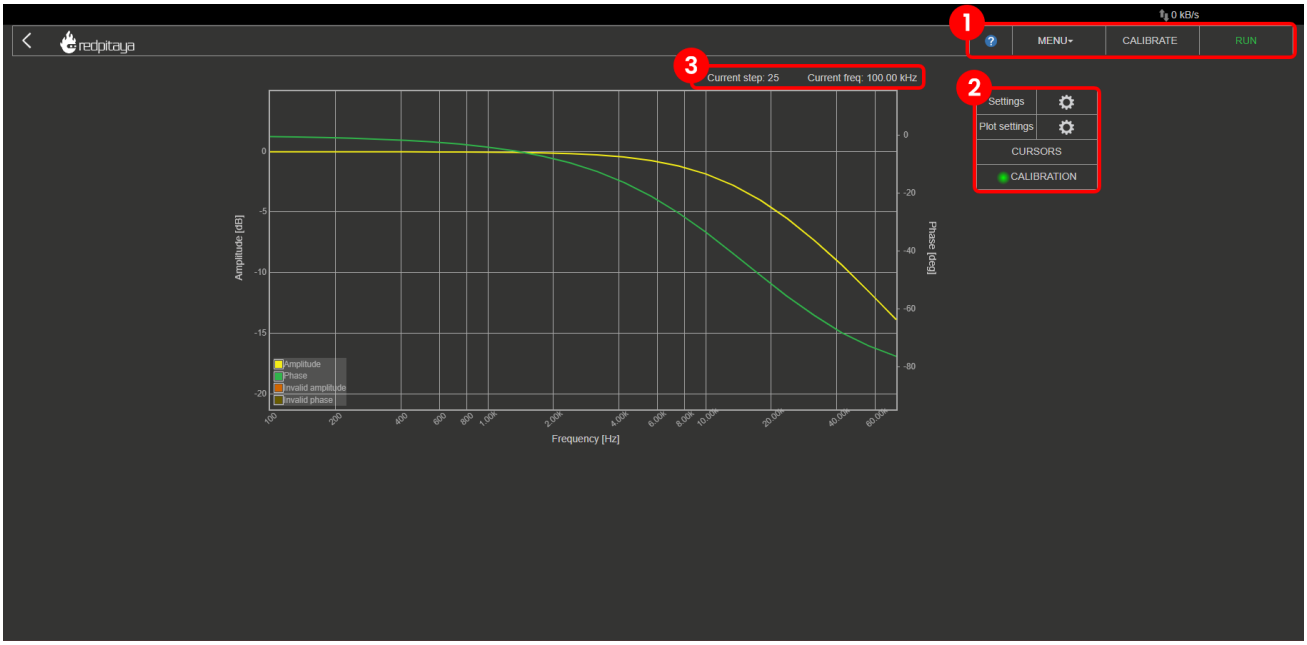
Reševanje težav z osciloskopom

- V primeru, da signal na grafu ni stabilen, oziroma se premika v levo ali v desno, je potrebno preveriti nastavitve proženja. Največkrat je proženje nastavljeno na napačen vhodni kanal ali pa je nivo proženja nad maksimalno vrednostjo signala.
- Če kurzor za nivo proženja ni viden na grafu, kar se lahko zgodi pri povečevanju amplitudne skale, pojdite v meni proženja in nastavite nivo proženja na 0. Alternativno, zmanjšajte amplitudno skalo in ga povlecite bližje osnovnemu nivoju signala.
- V primeru, da na analognih vhodih ne zaznavate signala, poizkusite odstraniti kratkostičnike za hitrimi analognimi vhodi in jih ponovno namestiti na nastavljeno napetostno območje (LV ali HV), saj se včasih zgodi, da imajo slab kontakt.
- V primeru zaznavanja nepričakovanih signalov povežite analogne izhode direktno na analogne vhode ter preverite ustreznost generacije (na primer z generacijo 1 kHz sinusa). Če še vedno zaznavate nepričakovane signale, je možno, da je nekdo pred vami modificiral FPGA sliko, ki se naloži ob zagonu aplikacije. Najlažja rešitev je ponovna namestitev operacijskega sistema ali sprememba v nastavitvah aplikacije (preverite katera slika se naloži ob zagonu).

4 Bodejev analizator

S pomočjo Bodejevega analizatorja lahko izmerimo amplitudni in fazni odziv vezij v odvisnosti od frekvence, kar je izredno uporabno pri analizi stabilnosti kontrolnih sistemov (na primer pri načrtovanju povratnih zank v napajalnikih), načrtovanju analognih filtrov, ipd. Frekvenčno območje Bodejvega analizatorja je med 1 Hz in 60 MHz [14].

Bodejev analizator na Red Pitayi, je tako kot osciloskop, zanesljiv do približno 20 MHz, saj z vzorčno frekvenco 125 MHz pri merjenju signalov višjih frekvenc lahko nezanesljive rezultate zaradi premajhnega števila točk na periodo.



Slika 27: Bodejev analizator na Red Pitayi [14]

Bodejev graf [*Slika* 27] je sestavljen iz dveh meritev, amplitudnega in faznega odziva merjenega vezja v odvisnosti od frekvence. Amplitudni odziv, prikazan z rumeno barvo, je merjen v decibelih. Pretvorbo med amplitudnim odzivom v voltih in amplitudnim odzivom v decibelih izvedemo z (1):

$$A[dB] = 20 \log\left(\frac{U_{izh}[V]}{U_{vh}[V]}\right) \qquad (1)$$

Fazni odziv, prikazan z zeleno barvo, je merjen v stopinjah in predstavlja fazni zamik med vhodnim in izhodnim signalom merjenega vezja. Omejen je med ±180 stopinj.

Amplitudna skala se nahaja na levi strani grafa, fazna pa na desni strani. Frekvenčna skala je v logaritmičnem merilu, kar omogoča lažji pregled odziva vezja pri različnih frekvencah [*Slika* 27].

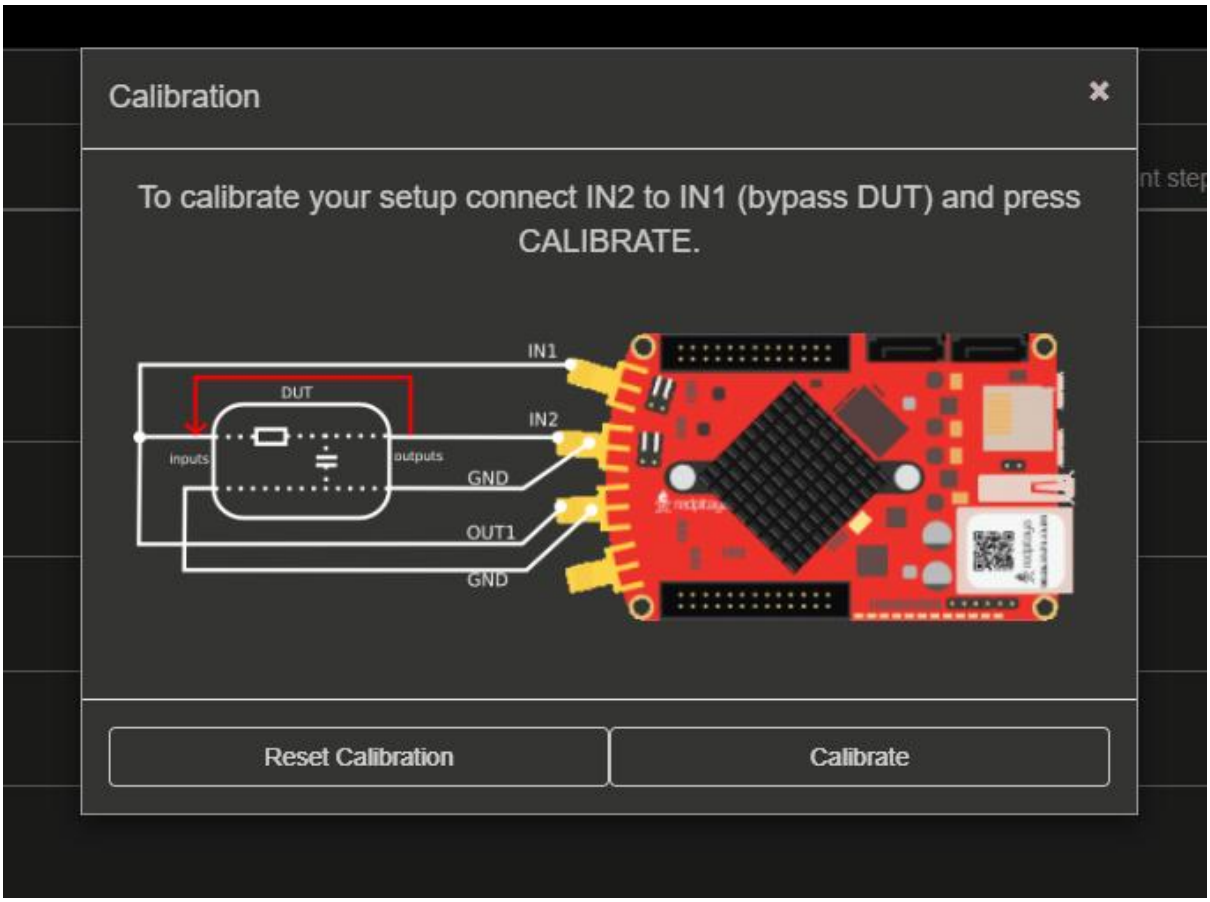
Poleg območja grafa je Bodejev analizator razdeljen še na tri druga območja [14]:

1. **Meni z nastavitvami aplikacije** – Zaustavitev in pogon inštrumenta, izvoz podatkov v obliki grafa ali CSV datoteke in kalibracija.
2. **Nastavitve merjenja in grafa** – konfiguracija frekvenčnega območja, števila korakov, mej grafa, postavitev kurzorjev, ipd.
3. **Podatki o trenutni meritvi** – prikaz trenutnega koraka in trenutne frekvence.

4.1 Kalibracija

Pred prvo meritvijo posameznega vezja je načeloma potrebno izvesti kalibracijo. Zaradi enostavnosti izvedbe in časovnih omejitev jo bomo na vajah izpustili. Posledično bo v naših meritvah prisoten tudi vpliv Red Pitaye, kabelskih povezav in preizkusne plošče, na kateri je sestavljeno vezje. Kljub temu je vpliv vseh treh na karakteristiko dovolj majhen, da ga lahko zanemarimo.

Kalibracijo izvedemo z klikom na gumb »kalibriraj« (»calibrate«) v nastavitvah aplikacije, pri čimer se pojavi naslednje okno [14], [Slika 28]:



Slika 28: Kalibracija Bodejevega analizatorja [14]

Na tej točki moramo poskrbeti za pravilno povezavo vezja in Red Pitaye:

- Kratkostičnike na Red Pitayi, ki se nahajajo za hitrimi analognimi vhodi IN1 in IN2, nastavimo v nizkonapetostni način (LV).
- Vhod 1 in izhod 1 Red Pitaye povežemo skupaj in ju priključimo na vhod testnega vezja. Ne pozabimo dodati 50-Ω zaključnega upora na izhod Red Pitaye.
- Vhod 2 vežemo na izhod testnega vezja
- Kratko sklenemo vhod in izhod testnega vezja.

Zgoraj opisana vezava je zelo pomembna, saj je v primeru napačne povezave kalibracija neustrezna.

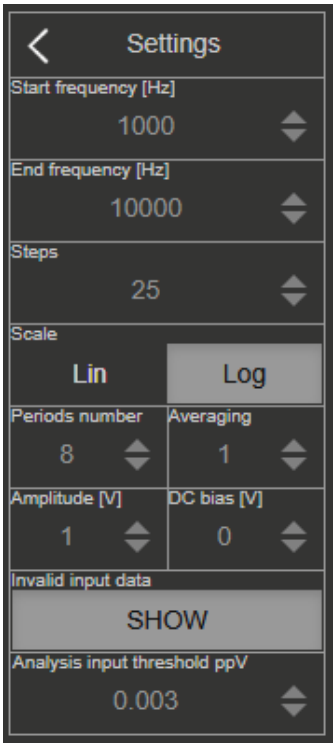
Potem, ko preverimo povezave, kliknemo na gumb »kalibriraj« (»calibrate«) [Slika 28]. Red Pitaya bo avtomatsko izvedla 500 točkovno meritev na frekvenčnem razponu med 100 Hz in 62.5 MHz. Kalibracija na operacijskem sistemu 1.04-28 traja približno 3 minute, na različicah OS nad 2.04-35 pa je veliko hitrejša. Zaradi dolžine kalibracije je stabilnost povezave izrednega pomena, saj se v primeru nestabilne povezave aplikacija lahko ponovno naloži, kar prekine postopek kalibracije.

V primeru, da med kalibracijo opazimo meritve amplitude in faze, prikazane z oranžno oziroma rjavo barvo, kar nakazuje neveljavne meritve, je potrebno postopek kalibracije prekiniti. Najpogostejši vzrok so slabe ali prekinjene povezave v testnem vezju. Še enkrat preverimo vse povezave in komponente ter ponovno začnemo postopek kalibracije.

Po zaključeni kalibraciji se bo indikator za kalibracijo v "nastavitvah merjenja in grafa" obarval zeleno. Po končani kalibraciji ne pozabite odstraniti kratkostičnika med vhodom in izhodom testnega vezja.

4.2 Nastavitve merjenja in grafa

V zavihku nastavitev (»settings«) [Slika 29] upravljamo z nastavitvami meritev. Bodejev analizator ob vsaki meritvi generira določeno število period sinusnega signala, ki jih po prehodu skozi vezje ponovno izmeri.



Spreminjamo lahko naslednje nastavitve [14]:

- **Začetna frekvenca** merjenja
- **Končna frekvenca** merjenja
- **Število korakov** – skupno število meritev, ki so enakomerno razporejene med začetno in končno frekvenco
- **Skala** – *linearna ali logaritmična*, vpliva na frekvenčni razpored meritev
- **Število period** – število period v eni meritvi
- **Amplituda** – amplituda sinusnega signala. V primeru, da naše testno vezje ojačuje signale, je priporočeno, da se amplituda generiranih pulzov zmanjša, saj lahko v nasprotnem primeru pridemo preko napetostnih mej vhodov (±1 V).
- **Povprečevanje (»averaging«)** – *vklopljeno ali izklopljeno*. Določa ali je končna meritev povprečje vseh poslanih period ali zadnja izvedena meritev.
- **DC odmik (»DC bias«)** – DC napetostni odmik signala od mase.
- **Vhodna napetostna meja analize (»analysis input threshold V_{PP}«)** – meritve periodičnih signalov, katerih medtemenska napetost je manjša od praga, se nadomestijo z minimalno vrednostjo praga (za namene izračuna).

Slika 29: Nastavitve merjenja [14]

V nastavitvah grafa lahko spreminjamo meje amplitudne in fazne osi. V osnovi je vklopljeno avtomatsko skaliranje osi, zato se z ročnim nastavljanjem po navadi ne ukvarjamo.

Na vsako os lahko, enako kot pri osciloskopu, za lažjo izvedbo meritev, dodamo dva kurzorja.

4.3 Nasveti za izvedbo meritev Bodejev analizator

1. Dvakrat preverite vse povezave med Red Pitayo in testnim vezjem, saj lahko ob napačni povezavi vhodov hitro pridemo do popolnoma drugačnih rezultatov. Primer, pri meritvah pasivnih filtrov je v primeru narobe obrnjenih vhodov (vhod 1 vezan na izhod testnega vezja, vhod 2 pa na vhod), celotna karakteristika zrcaljena.
2. V primeru, da med meritvijo opazite izris amplitude in faze z oranžno oziroma rjavo barvo, je potrebno meritev ponoviti, saj so meritve neveljavne. Najpogostejši vzrok so slabe oziroma prekinjene povezave v testnem vezju (preverite, da sonde niso iztaknile katere izmed komponente). Še enkrat preverite vse povezave in komponente in ponovno izvedite meritev.
3. Ne pozabite priključiti obeh konektorjev na sondi v vezje (črni krokodilček mora biti povezan na maso).
4. V veliki večini primerov ne potrebujete več kot 100 korakov. Meritve na vajah so namenjene spoznavanju z delovanjem vezij in ne zelo podrobni analizi. Z zmanjšanjem števila korakov tudi prihranite čas.
5. Nikoli ne priključite sond za osciloskop na Red Pitayin generator. Sonde so namenjene merjenju signalov. Z generacijo signala preko sond, le-tega popačite in posledično pridete do napačnih meritev.

Reševanje težav: Bodejev analizator

- Pri nekaterih novejših različicah operacijskega sistema je prisotna napaka v kodi, zaradi katere je potrebno zagnati meritev preden lahko dostopamo in spreminjamo nastavitve aplikacije.
- Po kalibraciji so nastavitve meritve nastavljene na takšne kot so bile uporabljene v kalibraciji (frekvenčni razpon 100 Hz – 62.5 MHz, 500 točk).
- Preverite, da je atenuacija sond nastavljena na 1x, saj trenutne različice spletnega vmesnika še ne omogočajo sprememb atenuacije.
- V primeru, da na analognih vhodih ne zaznavate signala, poizkusite odstraniti kratkostičnike za hitrimi analognimi vhodi in jih ponovno namestiti na nastavljeno napetostno območje (LV ali HV), saj se včasih zgodi, da imajo slab kontakt.
- Obstaja napaka v kodi Bodejevega analizatorja pri izračunu vrednosti amplitudnega odziva za majhne signale (opazna pri velikem razmerju vhodne in izhodne napetosti (npr. uporovni delilnik 1:100)).

5 Uvod v uporabo FPGA na Red Pitayi

V tem poglavju se bomo posvetili vezju FPGA na Red Pitayi, kaj vse vključuje, kako ga modificiramo, generiramo bitstream in na koncu reprogramiramo FPGA. Navodila so namenjena operacijskemu sistemu 1.04-28 za Red Pitayo STEMlab 125-14, vendar z nekaj modifikacijami delujejo tudi na ostalih OS in modelih Red Pitaye, ki jih bomo obravnavali na koncu tega poglavja.

V okviru vaj bomo uporabljali programski opremi Vivado 2020.1 [15] in ModelSim [16].

5.1 Namestitev razvojnih orodij

V okviru vaj bomo uporabljali dve razvojni okolji: Vivado 2020.1 za pisanje kode in generacijo datoteke za reprogramiranje FPGA, ter ModelSim za simulacijo ustvarjenih komponent in HDL kode. Za uspešno namestitev okolja potrebujemo okrog 100 GB prostora na našem računalniku (celotno okoje na koncu zavzema okrog 30 GB prostora na disku).

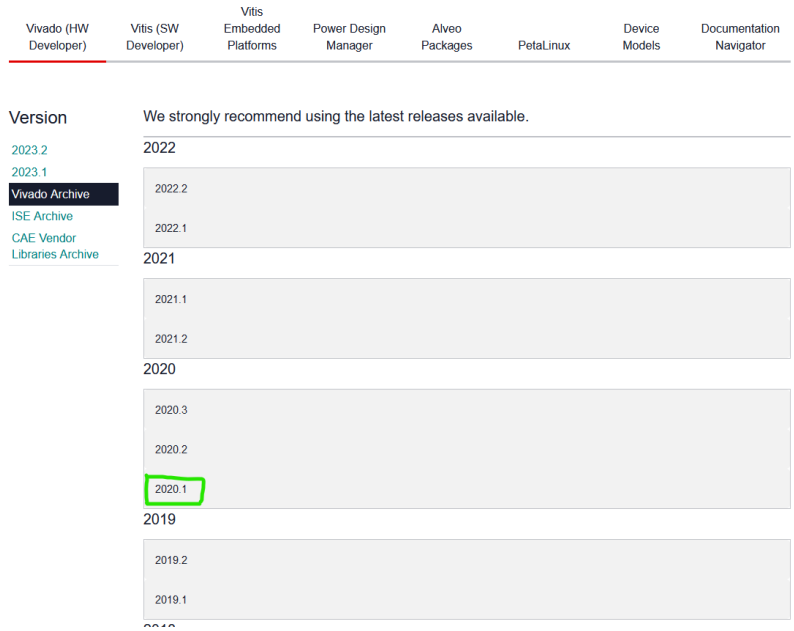
Obrazložitev uporabe Vivada 2020.1

Red Pitayini FPGA projekti so narejeni v orodju Vivado 2020.1 [17]. Nekaterim izmed nas se lahko poraja vprašanje, zakaj ni uporabljena najnovejša različica orodja Vivado (ali Vitis). Različne verzije Vivada se med seboj razlikujejo glede podprtih ukazov in že narejenih komponent, kar hitro povzroči nekompatibilnost starejših različic projektov z novejšimi, še posebej, ko so v igri skripte s stotinami vrstic ukazov, ki poskrbijo za avtomatsko vzpostavitev projekta. Predstavljajmo si, da smo ravnokar napisali 1000 stransko knjigo v programu Word, nato pa naložimo novejši urejevalnik in naenkrat oblika besedila ni več enaka (spremeni se velikost besedila, izbrani font ni več na voljo, črka ž naenkrat ne obstaja več…), zato moramo iti še enkrat čez celotno besedilo in popraviti napake.

Uporaba novejših različic je mogoča, vendar zahteva dobro poznavanje orodij in veliko časa za uspešno implementacijo, zato za učne namene priporočamo uporabo tiste različice v kateri je projekt prvotno zasnovan.

5.1.1 Vivado 2020.1

Za namestitev Vivada 2020.1 pojdimo na AMD Xilinx spletno stran s prenosi. Na levi strani je stolpec z različicami okolja, kjer kliknemo na »Vivado Arhiv«. V arhivu poiščemo različico 2020.1 [Slika 30].



Slika 30: Vivado arhiv

S klikom na »2020.1« se odpre spustni meni z različnimi opcijami. Poiščemo »Vivado Design Suite - HLx Editions - 2020.1 Full Product Installation« in prenesemo okolje s klikom na »Vivado HLx 2020.1: All OS installer Single-File Download (TAR/GZIP - 35.51 GB)«. Paziti moramo, da pomotoma ne izberemo »Vivado 2020.1 Update 1«, ki je v spustnem meniju prva in ima velikost 10,23 GB.

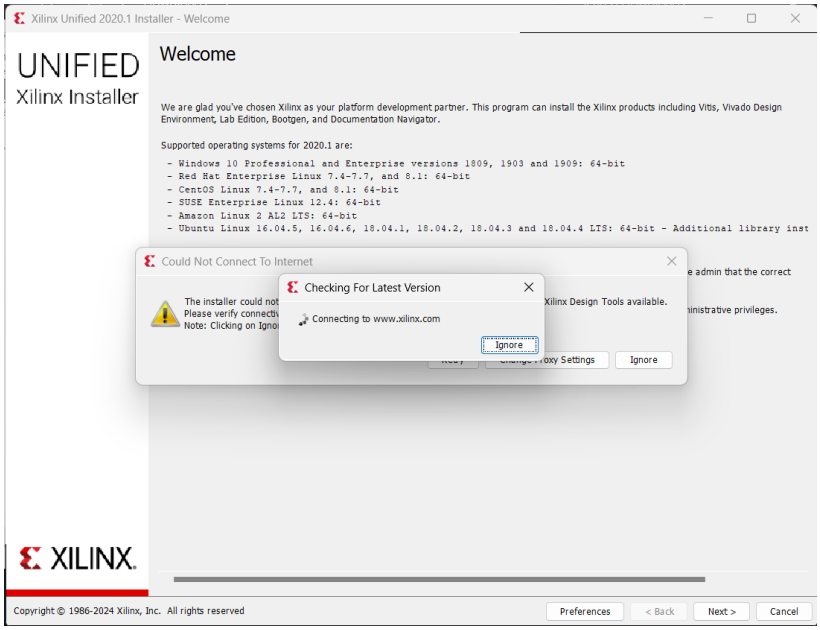
Datoteka za prenos je izredno velika (35,51 GB), kar bo, ne glede na hitrost internetne povezave, vzelo precej časa. Na spletni strani imamo tudi možnost izbire spletne namestitve (angl. »web installer«) za Linux in Windows, ki pa že nekaj časa **ne** deluje pravilno, zato smo primorani prenesti celoten programski paket.

S klikom na povezavo za prenos smo preusmerjeni na spletno stran za vpis z AMD uporabniškim računom. Če ga še nimamo, ga ustvarimo. Po vpisu z uporabniškim računom smo preusmerjeni v AMD prenosni center, kjer moramo izpolniti obrazec z osebnimi podatki. Z klikom na gumb »Prenesi« (angl. »Download«) se prenos datoteke začne.

Ko se prenos konča, imamo na računalniku stisnjeno datoteko velikosti 35,5 GB. Datoteke ne razširjamo ročno, saj bi to samo povečalo potreben prostor na disku. Najlažje je odpreti datoteko s programom za razširjanje, kot je na primer WinRAR [18]. Če uporabljamo Linux operacijski sistem, je potrebno datoteko razširiti (npr. z ukazom »tar«).

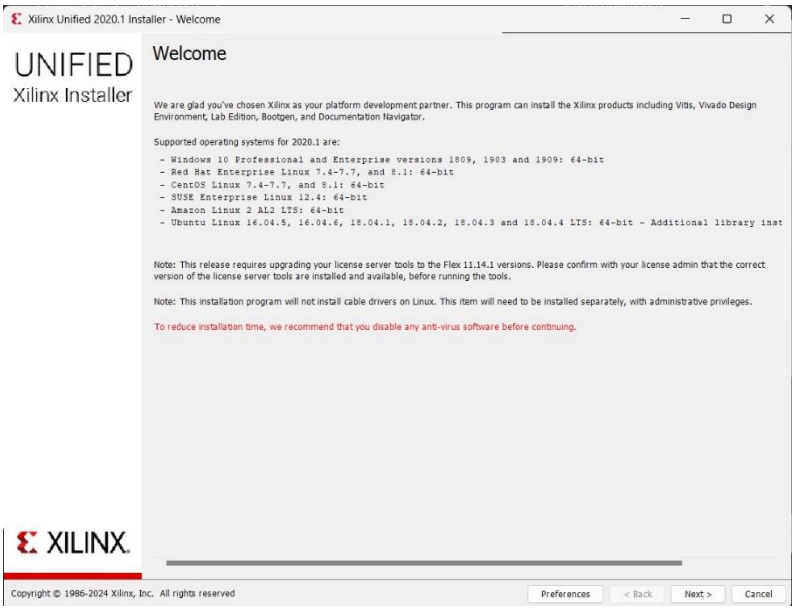
Na dnu vsebine datoteke vidimo namestitveni program za Windows (»xsetup.exe«) oziroma za Linux (»xsetup«), ki ga zaženemo z dvojnim klikom. WinRAR bo pred zagonom namestitvenega programa samodejno razširil celotno datoteko, kar bo trajalo približno 5 minut.

Ob zagonu namestitvenega programa nas operacijski sistem obvesti, da moramo najprej potrditi zagon programa Xilinx, nato še zagon programa Java. Po potrditvi obeh zahtev se prikaže začetno okno. Ob tem se pojavita dve opozorilni okni za neuspešno povezavo na internet, ki ju zapremo ali ignoriramo [Slika 31].



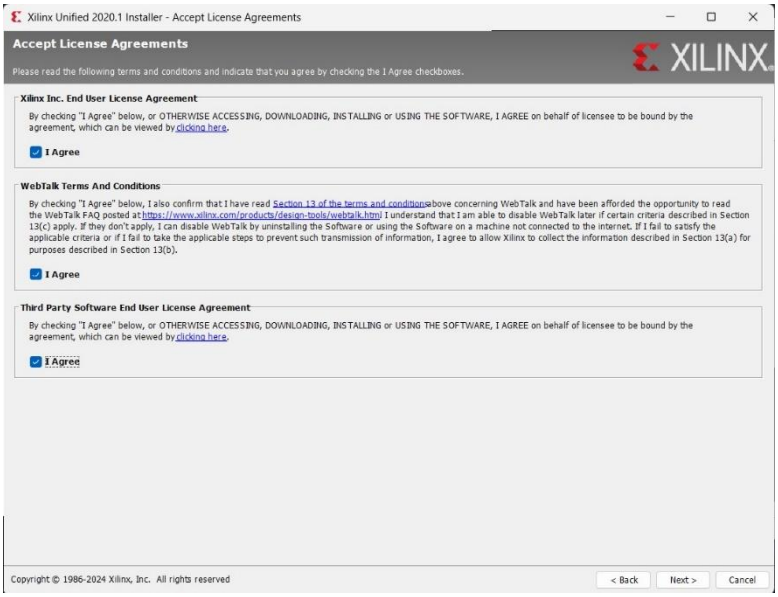
Slika 31: Namestitveni program Vivado 2020.1 - začetek

Izberemo »naprej« [Slika 32].



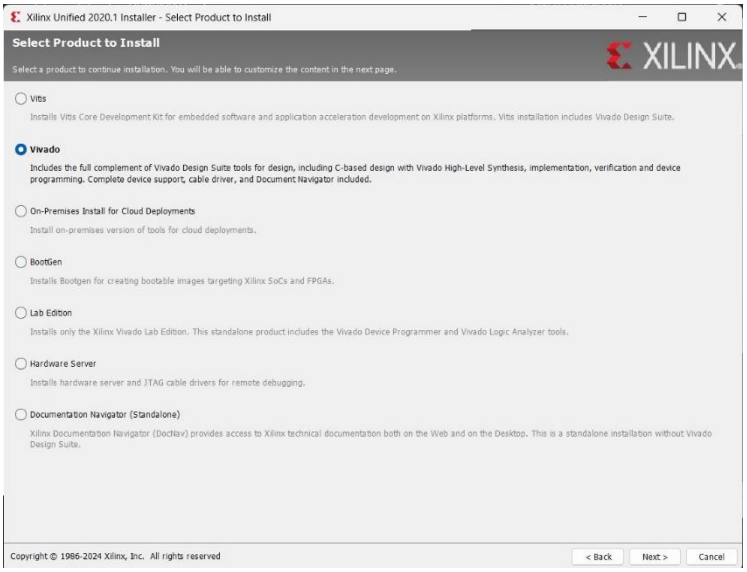
Slika 32: Začetno okno namestitvenega programa

Odkljukamo vsa tri soglasja in kliknemo »naprej« [*Slika 33*].



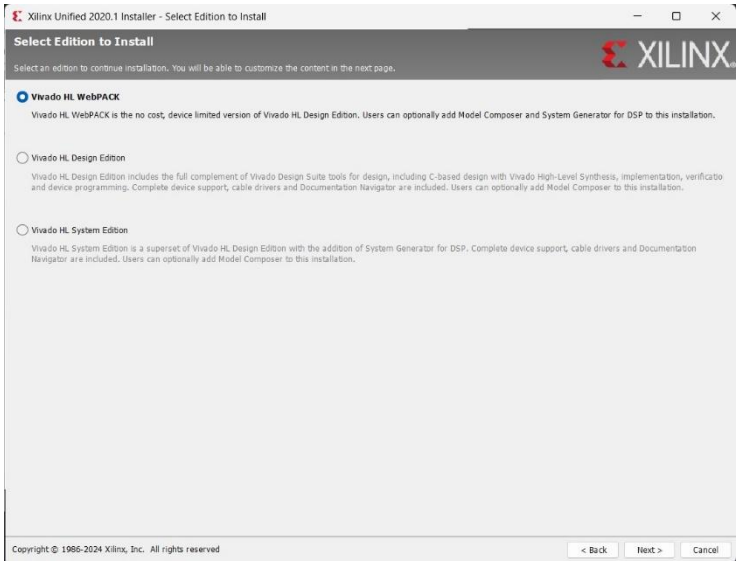
Slika 33: *Soglasja*

Pri izbiri namestitve programskega paketa izberemo »Vivado« [Slika 34].



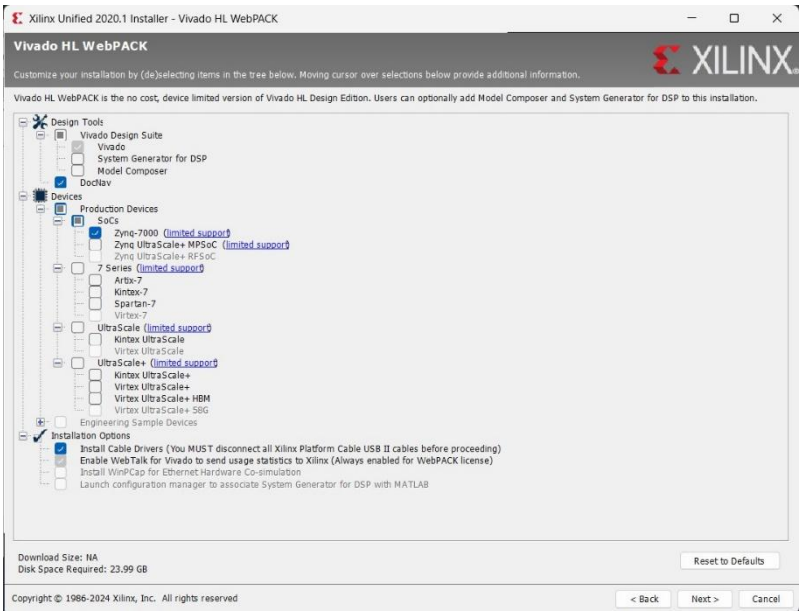
Slika 34: Izbira programskega paketa

Izberemo »Vivado HL WebPACK«, ki je vključen v brezplačno spletno licenco [Slika 35].



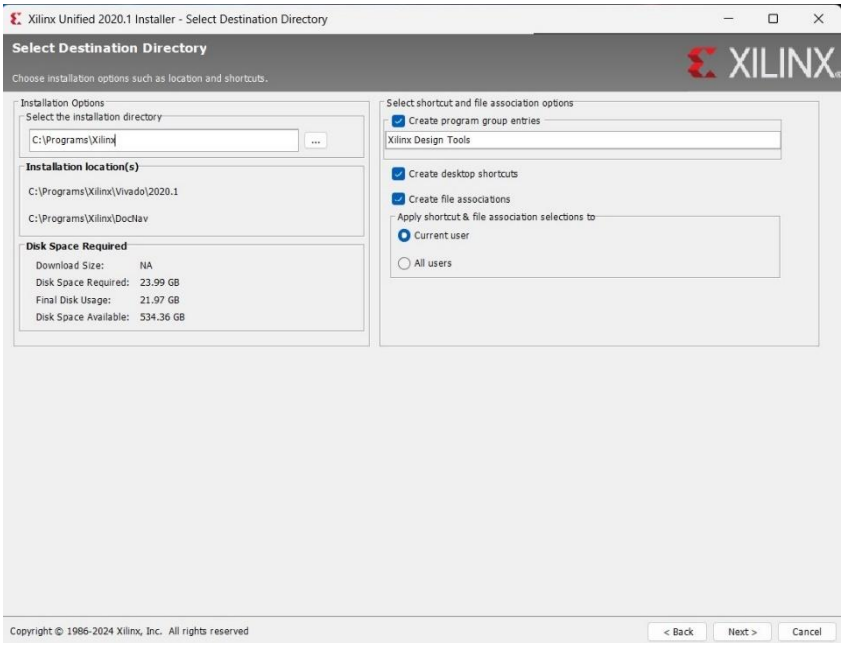
Slika 35: Izbira različice Vivada

Na naslednjem oknu odkljukamo opcije na [Slika 36]. Pri napravah izberemo samo Zynq 7000, saj Red Pitayin FPGA čip Zynq 7010 spada v to družino. Ostale možnosti samo povečajo količino prostora na disku, ki jo zavzema Vivado in so, razen če imamo v lasti kakšno drugo razvojno ploščo z različnim FPGA, popolnoma neuporabne.



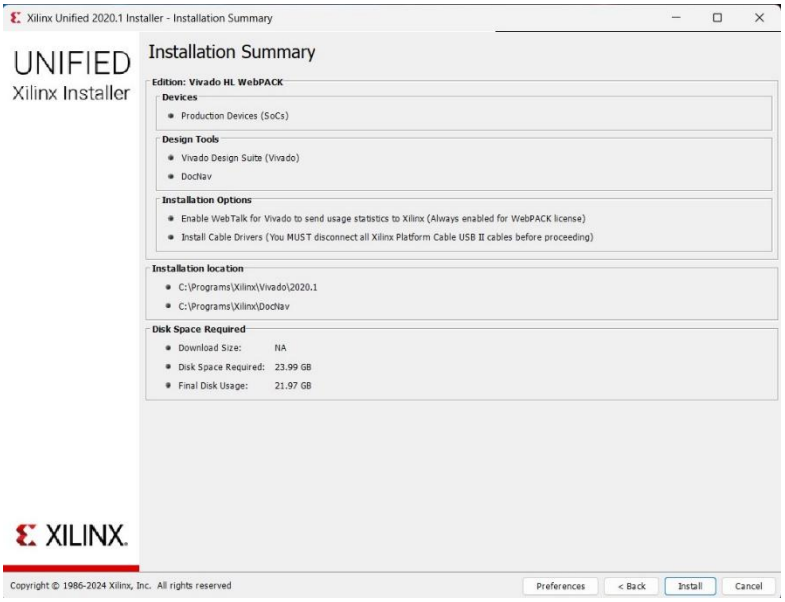
Slika 36: Izbira naprav in komponent programske opreme

Izberemo lokacijo namestitve na disku [Slika 37].



Slika 37: Izbira lokacije na disku

Še enkrat preverimo vse izbrane možnosti in kliknemo »namesti« [Slika 38].



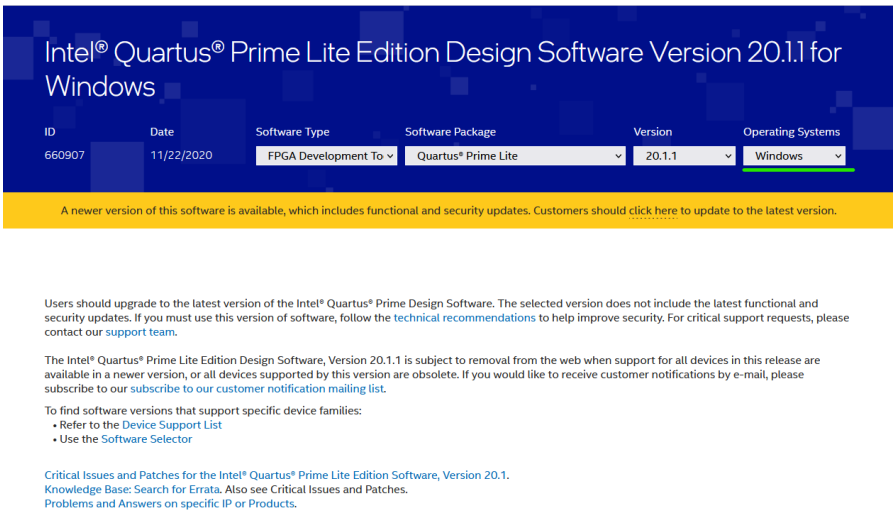
Slika 38: Povzetek namestitve

Zatem se začne postopek namestitve, ki bo trajal okrog 10 minut. Celoten proces namestitve je dokaj hiter, saj so vse datoteke že prisotne na našem računalniku. Po zaključeni namestitvi se prikaže pojavno okno, ki nas obvesti o uspešni namestitvi.

5.1.2 ModelSim

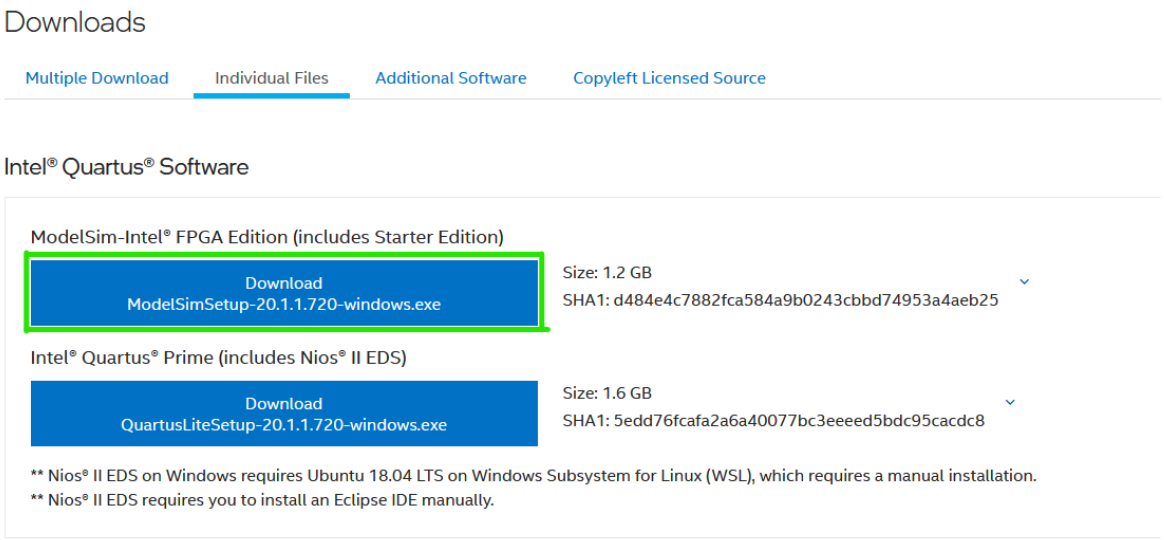
Na vajah bomo za simulacijo posameznih komponent v jeziku VHDL uporabili orodje ModelSim [16]. Vivado sicer vključuje lasten simulator, vendar vsebuje tudi celoten FPGA projekt Red Pitaye, ki je prezahteven za simulacijo. Zato bomo v ModelSim vključili le lastne komponente in jih simulirali, da se prepričamo o pravilnosti delovanja, nato pa jih vključili v okvir celotnega projekta in generirali FPGA sliko oziroma bitstream.

ModelSim lahko dobimo tukaj. V zadnjem spustnem meniju izberemo naš operacijski sistem (Windows ali Linux). Preverimo, da imamo izbrano različico 20.1 ali 20.1.1, saj je to zadnja različica, ki omogoča namestiti samo ModelSim simulator s čemer se izognemo namestitvi celotnega programa Intel Quartus [Slika 39].



Slika 39: ModelSim izbira OS

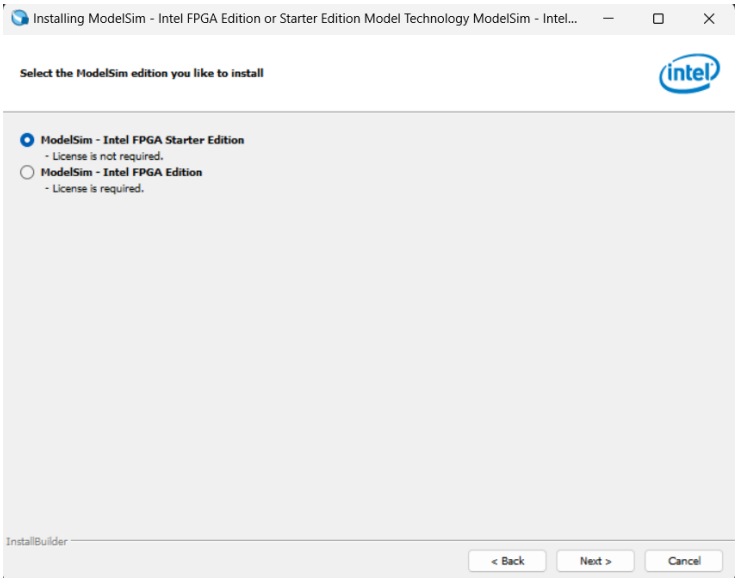
V zavihku »posamezne datoteke« (»individual files«) [Slika 40] izberemo »***ModelSim-Intel® FPGA Edition (includes Starter Edition)***«



Slika 40: Prenos ModelSim

Preusmerjeni smo na spletno stran z licenčno potrditvijo. Po potrditvi se program prenese.

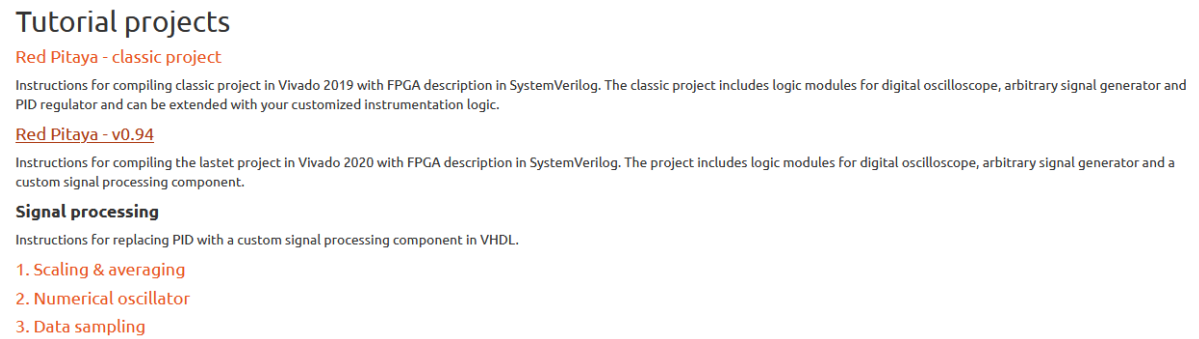
Pri namestitvi smo pozorni, da izberemo različico brez licence [Slika 41].



Slika 41: *Izbira različice ModelSim*

5.2 Prenos FPGA projekta za Red Pitayo

V okviru vaj bomo vedno uporabljali prirejen osnovni v0.94 projekt Red Pitaye, ki ga najdemo na spletni strani LNIV [19] v poglavju demonstracijski projekti (»tutorial projects«) [Slika 42].



Slika 42: LNIV Red Pitaya demonstracijski projekti

Kliknemo na »Red Pitaya – v0.94« s čemer odpremo novo spletno stran z navodili. V poglavju »Vivado 2020 projekt« poiščemo link s stisnjeno datoteko »*redpitaya94.zip*«. Prenesemo projekt in ga razširimo.

Zatem odpremo Vivado 2020.1 in se z uporabo Tcl konzole pomaknemo v razširjeni direktorij projekta. Projekt ustvarimo z ukazom:

```
source ./make_project.tcl
```

Celoten projekt se bo avtomatsko generiral.

5.2.1 Splošen projekt

Za splošen Red Pitaya projekt se uporablja osnovni projekt v0.94. Najdemo ga znotraj:

- Red Pitaya FPGA GitHub skladišča za operacijske sisteme 2.00 in novejše.
- Red Pitaya GitHub skladišča za operacijske sisteme 1.04 in starejše

Če želimo narediti projekt skladen z določeno različico Red Pitayinega operacijskega sistema, potem prenesemo točno določeno vejo skladišča kode. Povezave med operacijskimi sistemi in kodo lahko določimo s ključem v [Tabela 1]. [3]

OS	GitHub veja
2.05-37	2024.3
2.04-35	2024.2
2.00-30	2024.1
2.00-23	2023.3
2.00-18	2023.2
2.00-15	2023.1
1.04-28	2022.2
1.04-18	2022.1

Tabela 1: Povezava med Red Pitaya OS in GitHub vejami [3]

Kodo prenesemo s klikom na zeleni gumb »koda« in izbiro »prenesi zip« (»download zip«). Kodo razširimo v poljubno mapo na računalniku.

Na Windows operacijskem sistemu zatem odpremo »*Vivado*« in se s TCL konzolo pomaknemo v mapo z preneseno kodo. Nato v TCL konzoli zaženemo naslednji ukaz, ki avtomatsko odpre Vivado in generira celoten v0.94 FPGA projekt.

```
make project PRJ=v0.94 MODEL=Z10
```

V primeru, da je na našem računalniku nameščen ukaz »make«, lahko skripto zaženemo tudi preko terminala oziroma Ukaznega poziva.

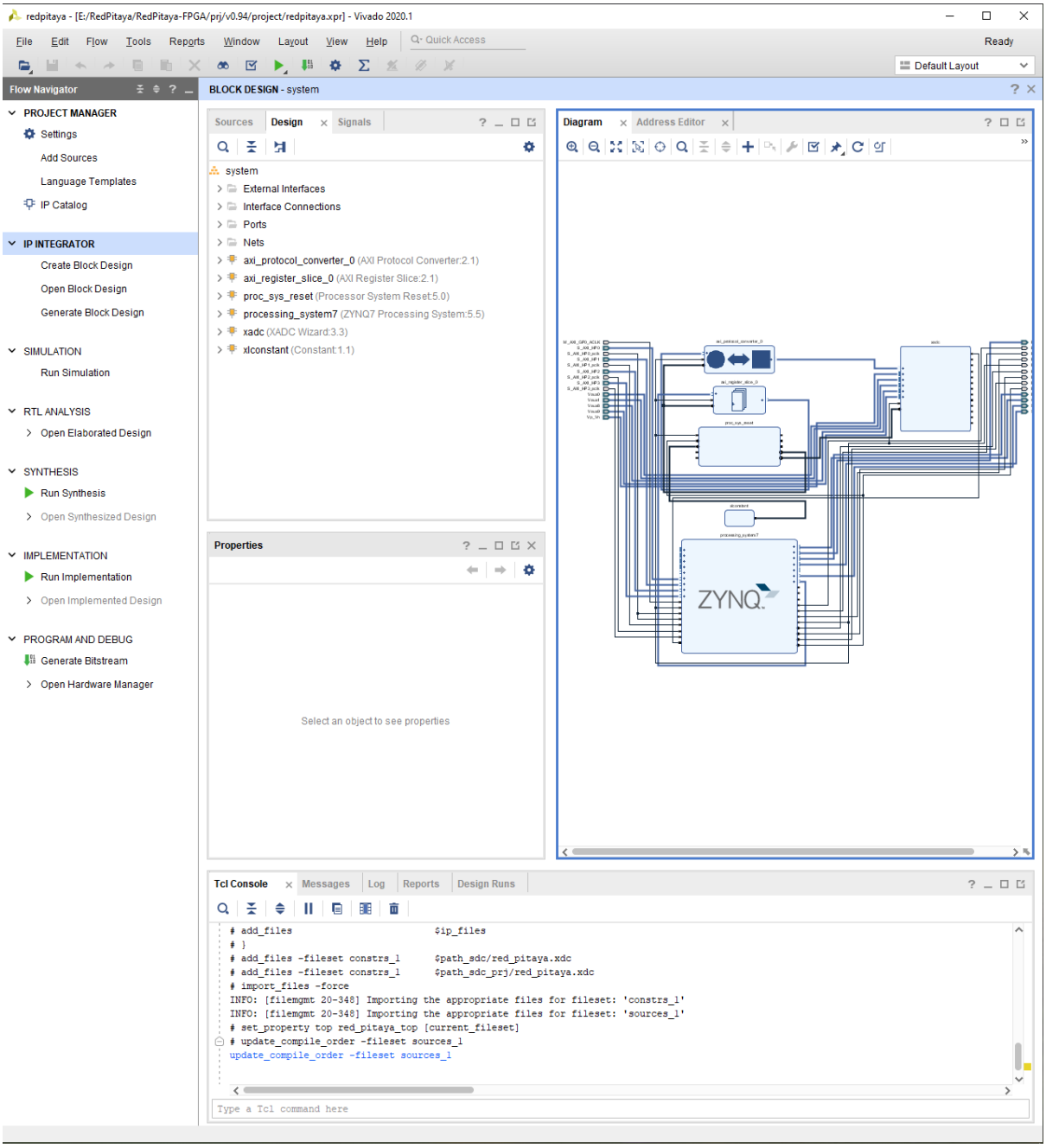
Na Linux operacijskem sistemu se moramo prepričati, da pred zagonom ukaza zaženemo datoteko z nastavitvami Vivada, ki se nahajaja v »*/opt/Xilinx/Vivado/2020.1/settings64.sh*«.

5.2.2 Ostali modeli Red Pitaya

V primeru, da imamo opravka z drugačnim modelom Red Pitaye, moramo pri zagonu ukaza podati drugačne parametre. Več si lahko preberete tukaj.

5.3 Vivado in osnovni projekt v0.94

Uporabniški vmesnik Vivada [Slika 43] je sestavljen iz opravilne vrstice in serije podoken, ki skupaj predstavljajo celoten program. Celoten uporabniški vmesnik je zelo kompleksen, zato bomo na hitro spoznali samo najpomembnejše nastavitve.



Slika 43: Programsko okolje Vivado

Opravilna vrstica

V opravilni vrstici se nahajajo vse nastavitve. Iz nje lahko odpremo katerokoli okno, če ga slučajno ponesreči zapremo. V zavihku:

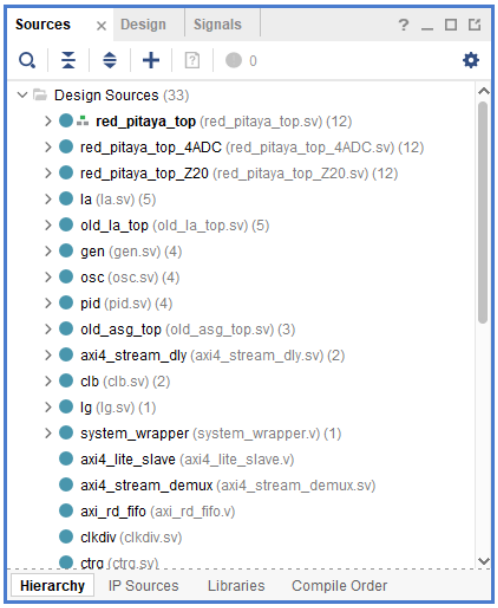
- »File« lahko odpremo in zapremo celoten projekt ter dodamo nove vire.
- »Flow« omogoča odprtje blokovnega diagrama, zagon sinteze, implementacije in generacije bitstreama.
- »Tools« omogoča generiranje novih IP blokov.
- »Window« omogoča odprtje večine podoken, ki so vidna na zaslonu.

Navigator toka (»Flow navigator«)

Navigator toka se nahaja v navpičnem oknu na skrajni levi strani. Vsebuje najpomembnejše možnosti, ki so na voljo uporabniku, kot so zagon simulacije, implementacije, generacije bitstreama ter ponovno odprtje blokovnega diagrama.

Viri, načrt in signali

Podokno z viri, načrt in signali [Slika 44] je sestavljeno iz treh zavihkov. Za nas je najbolj uporaben zavihek »viri« (angl. »sources«), saj prikazuje celotno hierarhijo Verilog in VHDL komponent. V zavihku »načrt« (»design«) pa najdemo komponente blokovnega diagrama našega projekta. Nov vir lahko vključimo s klikom na gumb »*+*« v zavihku »*viri*«, nato pa izberemo med vključitvijo novega ali obstoječega »načrtnega vira« (»design source«).



Slika 44: Podokno z viri, načrtom in signali

Celoten projekt Red Pitaye se na prvi pogled zdi izredno zapleten, vendar je treba upoštevati, da vključuje vse HDL datoteke za vse možne hardverske različice Red Pitaye. Za nas je relevantno samo drevo virov, ki se nahaja pod »*red_pitaya_top*«. Vse ostale vire lahko izključimo iz projekta brez kakršnihkoli posledic na končni rezultat.

Informacije o trenutno označenem viru

Pod oknom z viri se nahaja podokno z informacijami o trenutno označenem viru, komponenti, datoteki z izvorno kodo, IP bloku, naslovnem prostoru ali čem drugem. Izpisane so vse informacije, ki jih ima Vivado o posamezni komponenti, vključno z vsemi parametri, ki jih lahko spreminjamo.

Glavno okno

V glavnem oknu, ki se nahaja na skrajni desni strani lahko najdemo blokovni diagram projekta, shemo naslovnega prostora in pa kodo trenutno odprtih datotek.

Tcl konzola, obvestila, ipd.

V podoknu na dnu uporabniškega vmesnika je več zavihkov:

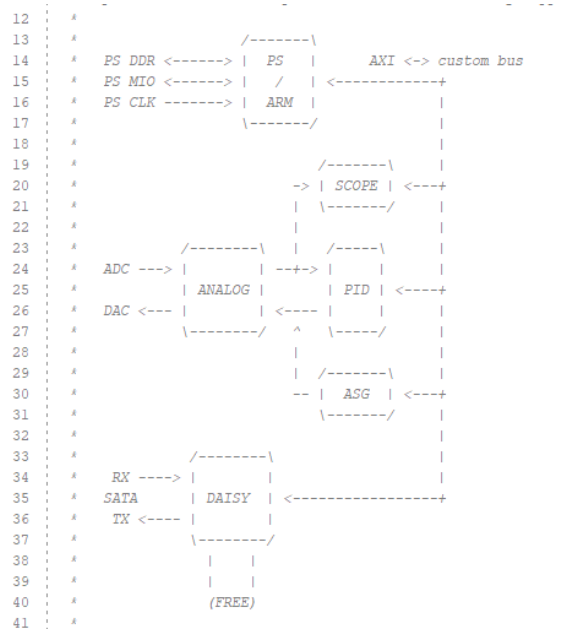
- V »*Tcl konzoli*« lahko poganjamo »tcl« datoteke in vanjo vpisujemo ukaze. Večina ukazov je podobna tistim v Linux terminalu.
- Pod »*sporočili*« (»*messages*«) najdemo obvestila o kodi. Tukaj nas Vivado obvešča o napakah v kodi in ter prikaže splošna obvestila o projektu in izvedenih korakih.
- Pod »*poročili*« lahko po zagonu sinteze, implementacije, itd. pregledamo zasedenost posameznih virov na FPGA čipu.

Potek programiranja

Med vajami bomo večinoma dodajali nove vire k že obstoječemu projektu, popravljali in pisali VHDL kodo, ter po simulaciji v orodju ModelSim v Vivadu zagnali sintezo, implementacijo in na koncu še generacijo Bitstream datoteke.

5.3.1 Opis projekta v0.94

Večina Red Pitayinega FPGA projekta je napisana v jezikih Verilog in System Verilog, vendar lahko v projekt vključimo tudi komponente, napisane v jeziku VHDL. Če odpremo spustni meni virov, skrit pod »red_pitaya_top«, vidimo 12 komponent, ki dejansko sestavljajo sistem. Celotna povezava med njimi je opisana v komentarjih na vrhu modula »red_pitaya_top« [Slika 45], kjer je na voljo tudi kratek opis pomembnejših komponent. [20]



Slika 45: Blokovna zgradba Red Pitaya v0.94 projekta [20]

Posamezne komponente so med seboj povezane s poenostavljeno obliko AXI vodila. AXI (»Advanced eXtensible Interface«) [21] je komunikacijski protokol namenjen mikrokrmilnikom in spada v sklop AMBA (»Advanced Microcontroller Bus Architecture«) protokolov [22]. AMBA je sklop različnih protokolov, ki definirajo način komunikacije med različnimi funkcionalnimi bloki na sistemih na čipu (SoC) .

Poenostavljeno AXI vodilo je sestavljeno iz naslednjih signalov:

- *clk* – (»clock«) urin signal
- *addr* – (»address«) naslovno vodilo
- *wdata* – (»write data«) piši podatke
- *rdata* – (»read data«) beri podatki
- *wen* – (»write enable«) omogoči pisanje
- *ren* – (»read enable«) omogoči branje
- *err* – (»error«) napaka
- *ack* – (»acknowledge«) potrditev

Naslovni prostor AXI vodila Red Pitaye je v projektu v0.94 razdeljen na 8 enot, kjer ima vsaka enota 20 bitni naslov [Tabela 2]. Začne se na heksadecimalnem naslovu 0x40000000 in konča na 0x407FFFFF. Na vsako izmed osmih enot je povezan določen modul [20], [23], [24].

Začetni naslov	Končni naslov	Ime modula
0x40000000	0x400FFFFF	Ostalo
0x40100000	0x401FFFFF	Osciloskop
0x40200000	0x402FFFFF	Generator poljubnih signalov
0x40300000	0x403FFFFF	PID krmilnik
0x40400000	0x404FFFFF	Mešani analogni signali
0x40500000	0x405FFFFF	Veriženje
0x40600000	0x406FFFFF	Prosto
0x40700000	0x407FFFFF	Test napajanja

Tabela 2: Razdelitev AXI naslovnega prostora na Red Pitayi [23]

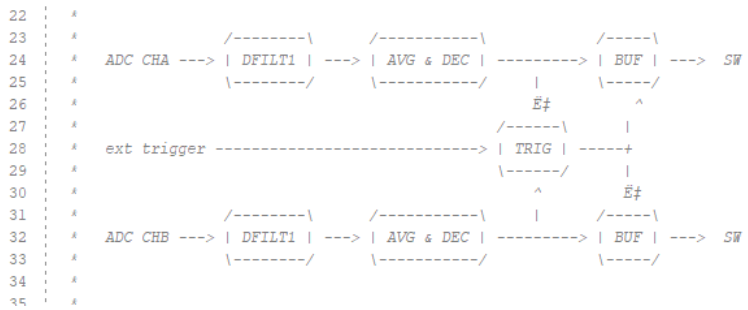
Arhitektura Red Pitaye je 32-bitna, kar pomeni, da posamezen register zaseda 32-bitov (4 bajte). Podrobnejšo razdelitev naslovnega prostora AXI vodila najdete tukaj.

Ostalo

Modul ostalo (»*housekeeping*«), ki je znotraj projekta v0.94 poimenovan »*red_pitaya_hk*« oziroma »*i_hk*«, skrbi za identifikacijsko številko sistema (ID), hkrati pa ima nadzor nad osmimi LED diodami in šestnajstimi digitalnimi vhodi in izhodi.

Osciloskop

Modul osciloskop, poimenovan »*red_pitaya_scope*« oziroma »*i_scope*«, je namenjen zajemanju podatkov v dva medpomnilnika realizirana v BRAM, po eden za vsak vhodni kanal. Med vhodnim signalom in medpomnilnikom sta prisotna še dva modula. Prvi je digitalni filter, ki poskrbi za pravilno frekvenčno kalibracijo signala, drugi modul pa lahko povpreči in decimira vhodni signal [Slika 46].



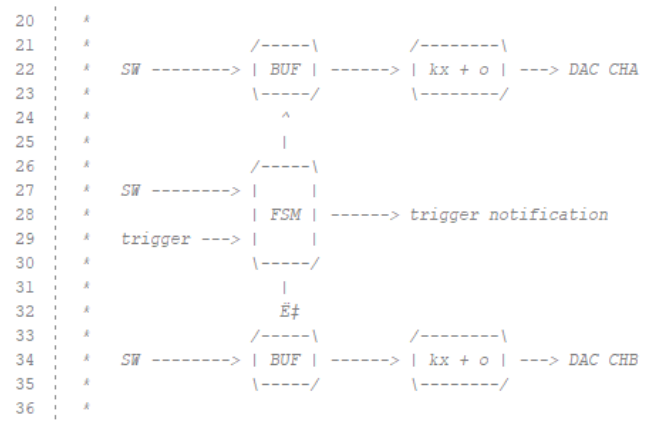
Slika 46: Blokovna zgradba osciloskopa [24]

Zajem signala se prične, ko je izpolnjen pogoj za sprožitev (»trigger«).

Generator poljubnih signalov

Generator poljubnih signalov (angl. »Arbitrary signal/waveform generator«) se nahaja na naslovu omogoča generacijo poljubnih signalov definiranih s strani uporabnika. V FPGA je predstavljen z modulom »red_pitaya_asg« oziroma »i_asg« [Slika 47]. Signal se generira iz medpomnilnika fiksne dolžine. Pri čemer je skok bralnega kazalnika odvisen od podane frekvence. Vsebina medpomnilnika je generirana softversko, ali

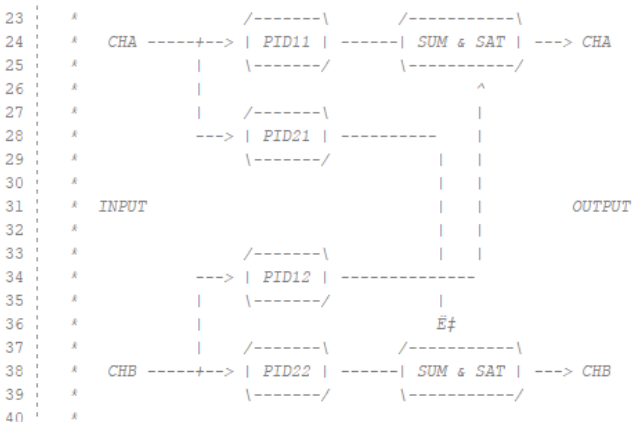
kot ena izmed vnaprej izbranih oblik ali pa podana s strani uporabnika. Signal iz medpomnilnika je pred izhodom še skaliran preko linearne funkcije določene s kalibracijo.



Slika 47: Blokovna zgradba generatorja poljubnih signalov [24]

PID krmilnik

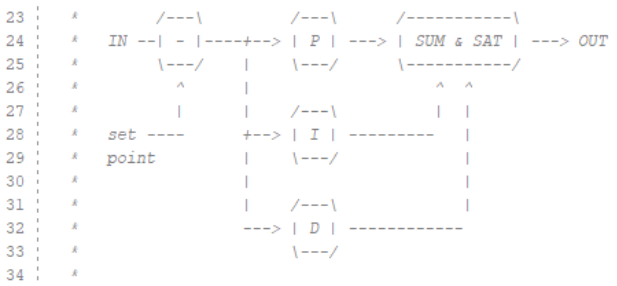
PID krmilnik (»Proportional-Integral-Derivative controller«) je v FPGA predstavljen z modulom »red_pitaya_pid« oziroma »i_pid« [Slika 48]. Celoten modul PID je sestavljen iz štirih PID blokov. Vsaka linija podatkov iz ADC-ja, ki predstavlja enega izmed hitrih analognih vhodov (CHA in CHB oziroma IN1 in IN2), se razcepi na dva dela. Vsak izmed delov gre skozi svoj PID blok in se nato sešteje z izhodom iz PID bloka drugega kanala in poda na hitri analogni izhod. [24]



Slika 48: Blokovna zgradba PID krmilnika [24]

S tem lahko enostavno dosežemo, da hitri analogni vhod postane funkcija obeh analognih vhodov, kar je zelo pomembno pri nadzorovanju različnih sistemov (npr. za stabilizacijo laserjev).

Vsak PID blok je zgrajen iz proporcionalnega, integrirnega in odvajalnega dela. Vhodni signal v PID blok se odšteje od »nastavljene točke«. Razlika predstavlja napako in se poda na vhod P, I, in D delov, nakar se izhodi vseh treh delov seštejejo in podajo na izhod [Slika 49].



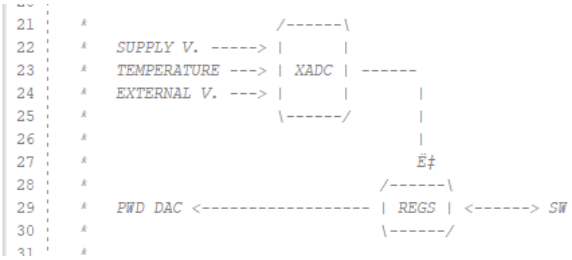
Slika 49: Blokovna zgradba PID bloka [24]

Integrirni del je mogoče resetirati s posebnim signalom.

Mešani analogni signali

Angl. »analog mixed signals«, predstavljajo »počasne« analogne vhode in izhode na razširitvenem konektorju E2. V FPGA so predstavljeni z modulom »red_pitaya_ams« oziroma »i_ams« [Slika 50].

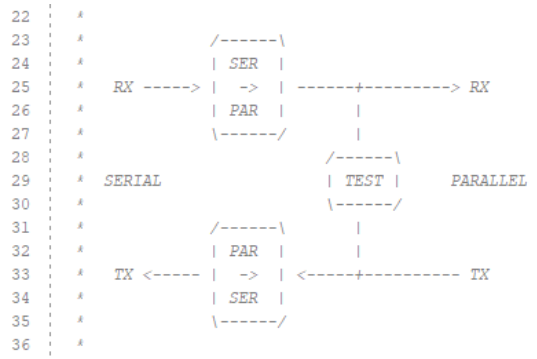
Na Red Pitayi so štirje počasni analogni vhodi in štirje počasni analogni izhodi. Počasni analogni vhodi (AIN0, AIN1, AIN2 in AIN3) so povezani na 12-bitni ADC, ki hkrati meri tudi notranje napetosti na Red Pitayi (pomembne za stabilnost sistema). Napetostno območje teh vhodov je med 0 in 3,5 V. Izmerjene vrednosti se shranijo v FPGA registre. Preko registrov pa lahko nastavimo vrednosti analognih izhodov, z izhodnim napetostnim območjem med 0 in 1,8 V. [5], [24]



Slika 50: Blokovna zgradba mešanih analognih signalov [24]

Veriženje

Modul za veriženje (»daisy chaining«) je namenjen povezavi in sinhronizaciji večih Red Pitay preko urinega signala glavne oziroma primarne enote. V FPGA je predstavljeni z modulom »red_pitaya_daisy« oziroma »i_daisy«. V verigi primarna enota deli svoj urin in prožilni signal vsem sekundarnim enotam, kar omogoča sinhrono generacijo in zajemanje podatkov [Slika 51].



Slika 51: Blokovna zgradba modula za veriženje [24]

Poleg deljenja urinega in prožilnega signala, modul omogoča tudi hitro komunikacijo in pošiljanje podatkov med posameznimi enotami.

Prosta vodila

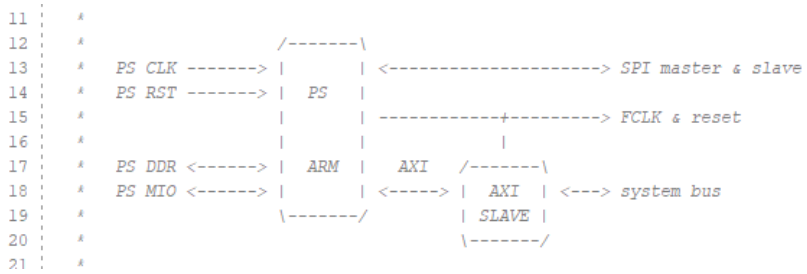
Kot smo omenili na začetku poglavja »Opis projekta v0.94« je naslovno vodilo v v0.94 projektu razdeljeno na 8 naslovnih prostorov. Od tega jih je 6 zasedenih z zgoraj naštetimi komponentami, dve pa sta prosti in povezani na vodilni štrcelj/končnik (»system stub«). Vodilni štrcelj poskrbi, da pri branju prostih, oziroma nepovezanih, naslovov ne pride do sistemske napake. [20]

Sistemski povezovalnik

Sistemski povezovalnik (angl. »system bus interconnect«), ki je v FPGA predstavljen z modulom »system_bus_interconnect«, poskrbi za razdelitev naslovnega vodila/prostora na več manjših. V projektu v0.94 razdeli naslovni prostor na osem različnih delov, kjer ima vsak del 20-bitni naslov. [20]

Procesni sistem

Procesni sistem se nahaja v modulu »red_pitaya_ps« oziroma »ps« in predstavlja dvojedrni ARM procesor, ki skupaj z FPGA vezjem sestavlja Zynq 7010 čip [Slika 52]. Procesni sistem je odvisen od zunanje ure in reseta, komunicira pa z DDR spominom, analognimi in digitalnimi konektorji na razširitvenih konektorjih E1 in E2, serijsko konzolo na mikro USB konektorju poleg napajanja, ter AXI vodilom. Znotraj modula je poseben AXI blok, ki poenostavi AXI vodilo, da je prijaznejše uporabniku, in ga poveže na sistemsko vodilo. [20]



Slika 52: Blokovna zgradba procesnega sistema [20]

5.4 Dodatek lastne komponente v sistem

Če želimo v sistem dodati lastno komponento, kateri želimo spreminjati parametre (oziroma jo upravljati z C programom), jo moramo povezati na sistemsko vodilo. Projekt v0.94 ima sistemsko vodilo razdeljeno na 8 razdelkov. Načeloma bi lahko priključili našo komponento na eno izmed prostih razdelkov, vendar bomo iz praktičnih razlogov raje odklopili eno izmed obstoječih komponent in na njeno mesto povezali našo komponento. Odklopili bomo komponento »PID« in na njeno mesto priključili komponento »red_pitaya_proc« (oziroma »i_proc«) s katero bomo lahko nadzorovali LED, digitalne priključke, ter hitre analogne vhode in izhode. Primer je povzet po nalogah iz LNIV [19] in Red Pitayine spletne strani [25].

Povezavo nove komponente moramo narediti na dveh nivojih. Najprej moramo odstraniti »red_pitaya_pid« komponento iz vrhnjega modula in na njeno mesto priključiti »red_pitaya_proc«. Nato moramo v projekt dodati nov vir, ki vsebuje podatke o naši komponenti in njeni funkcionalnosti.

5.4.1 Spremembe vrhnje komponente

Izključitev že obstoječe komponente je izredno preprosta, saj moramo iz vrhnje komponente, v našem primeru »red_pitaya_top«, le odstraniti kodo, ki povezuje obstoječo komponento in vrhnjo komponento. Kodo kompnente PID bomo zakomentirali, saj lahko pride prav v prihodnosti.

V »red_pitaya_top« poiščemo »red_pitaya_pid« komponento. Nahaja se pod komentarjem »MIMO PID controller« med vrsticami 526 in 548 [Slika 53].

```
526 ////////////////////////////////////////////////////////////////
527 //  MIMO PID controller
528 ////////////////////////////////////////////////////////////////
529
530 red_pitaya_pid i_pid (
531    // signals
532   .clk_i           (adc_clk   ),  // clock
533   .rstn_i          (adc_rstn  ),  // reset - active low
534   .dat_a_i         (adc_dat[0]),  // in 1
535   .dat_b_i         (adc_dat[1]),  // in 2
536   .dat_a_o         (pid_dat[0]),  // out 1
537   .dat_b_o         (pid_dat[1]),  // out 2
538   // System bus
539   .sys_addr        (sys[3].addr ),
540   .sys_wdata       (sys[3].wdata),
541   .sys_wen         (sys[3].wen  ),
542   .sys_ren         (sys[3].ren  ),
543   .sys_rdata       (sys[3].rdata),
544   .sys_err         (sys[3].err  ),
545   .sys_ack         (sys[3].ack  )
546 );
547
```

Slika 53: Priključitev MIMO PID komponente na vrhnjo komponento [20]

Pri podrobnem ogledu signalov, ki so povezani na »red_pitaya_pid«, že lahko sklepamo katere signale bomo potrebovali:

- Sistemsko vodilo, sestavljeno iz naslovnega vodila, urinega signala, ter reseta
- Povezave do hitrih analognih vhodov in izhodov
- Povezave do LED
- Povezave do digitalnih priključkov

Sedaj lahko komentiramo kodo PID komponente.

Zaradi preglednosti bomo priključitev komponente dodali povsem na konec »red_pitaya_top«. Za začetek lahko kar kopiramo povezavo PID komponente in jo prilepimo na konec »red_pitaya_top«.

LED in digitalni konektorji so priključeni na modul »red_pitaya_hk«, zato jih moramo najprej od tam odklopiti [Slika 54], [20].

```
450 ////////////////////////////////////////////////////////////////////////////////
451 // GPIO
452 ////////////////////////////////////////////////////////////////////////////////
453
454 assign trig_output_sel = daisy_mode[2] ? trig_asg_out : scope_trigo;
455 assign exp_p_otr = exp_p_out;
456 assign exp_n_otr = exp_n_out | ({DWE{daisy_mode[1]}} & {{DWE-1{1'b0}},trig_output_sel});
457
458 assign exp_p_dtr = exp_p_dir;
459 assign exp_n_dtr = exp_n_dir | ({DWE{daisy_mode[1]}} & {{DWE-1{1'b0}},1'b1});
460
461 IOBUF i_iobufp [DWE-1:0] (.O(exp_p_in), .IO(exp_p_io), .I(exp_p_otr), .T(~exp_p_dtr) );
462 IOBUF i_iobufn [DWE-1:0] (.O(exp_n_in), .IO(exp_n_io), .I(exp_n_otr), .T(~exp_n_dtr) );
463
464 assign gpio.i[2*GDW-1:  GDW] = exp_p_in[GDW-1:0];
465 assign gpio.i[3*GDW-1:2*GDW] = exp_n_in[GDW-1:0];
466
```

Slika 54: GPIO povezave [20]

GPIO signali so povezani na digitalne priključke na razširitvenem konektorju E1 [5]. Signali iz konektorja so povezani na vhodno-izhodni medpomnilnik, s katerim lahko določimo smer posameznih priključkov (torej izberemo ali se obnašajo kot vhod ali izhod). Medpomnilnik razdeli vhodno-izhodni signal »exp_io« (sestavljen iz »exp_n_io« in »exp_p_io«, kjer vsak predstavlja eno vrsto osmih digitalnih priključkov) na vhodni signal »exp_in«, izhodni signal »exp_out«, ter kontrolni smerni signal »exp_dtr« [Slika 55], [20]. Logike ne bomo spreminjali, zato bomo v lastno komponento vključili dodatne signale potrebne za kontolo vhodno-izhodne logike.

```
409 ////////////////////////////////////////////////////////////////////////////////
410 //  House Keeping
411 ////////////////////////////////////////////////////////////////////////////////
412
413 logic [DWE-1: 0] exp_p_in , exp_n_in ;
414 logic [DWE-1: 0] exp_p_out, exp_n_out;
415 logic [DWE-1: 0] exp_p_dir, exp_n_dir;
416 logic [DWE-1: 0] exp_p_otr, exp_n_otr;
417 logic [DWE-1: 0] exp_p_dtr, exp_n_dtr;
418
419 red_pitaya_hk i_hk (
420   // system signals
421   .clk_i           (adc_clk ),  // clock
422   .rstn_i          (adc_rstn),  // reset - active low
423   // LED
424   .led_o           (  led_o                    ),  // LED output
425   // global configuration
426   .digital_loop    (digital_loop),
427   .daisy_mode_o    (daisy_mode),
428   // Expansion connector
429   .exp_p_dat_i     (exp_p_in ),  // input data
430   .exp_p_dat_o     (exp_p_out),  // output data
431   .exp_p_dir_o     (exp_p_dir),  // 1-output enable
432   .exp_n_dat_i     (exp_n_in ),
433   .exp_n_dat_o     (exp_n_out),
434   .exp_n_dir_o     (exp_n_dir),
435   .diag_i          (locked_pll_cnt_r2),
436    // System bus
437   .sys_addr        (sys[0].addr ),
438   .sys_wdata       (sys[0].wdata),
439   .sys_wen         (sys[0].wen  ),
440   .sys_ren         (sys[0].ren  ),
441   .sys_rdata       (sys[0].rdata),
442   .sys_err         (sys[0].err  ),
443   .sys_ack         (sys[0].ack  )
444 );
445
```

Slika 55: Povezava komponente z ostalimi signali [20]

V »red_pitaya_hk« komentiramo naslednje signale:

- led_o
- exp_p_dat_i
- exp_p_dat_o
- exp_p_dir_o
- exp_n_dat_i
- exp_n_dat_o
- exp_n_dir_o

in jih prenesemo v povezave naše nove komponente [Slika 56]:

```
594 ////////////////////////////////////////////////
595 // Custom processing component
596 ////////////////////////////////////////////////
597
598 red_pitaya_proc i_proc (
599    // signals
600   .clk_i           (adc_clk   ),    // clock
601   .rstn_i          (adc_rstn  ),    // reset - active low
602   .dat_a_i         (),  // in 1
603   .dat_b_i         (),  // in 2
604   .dat_a_o         (),  // out 1
605   .dat_b_o         (),  // out 2
606   // GPIO + LED
607   .led_o           (led_o   ),      // LED output
608   .gpio_p_i        (exp_p_in ),     // input data
609   .gpio_p_o        (exp_p_out),     // output data
610   .gpio_p_dir      (exp_p_dir),     // 1-output enable
611   .gpio_n_i        (exp_n_in ),
612   .gpio_n_o        (exp_n_out),
613   .gpio_n_dir      (exp_n_dir),
614   // System bus
615   .sys_addr        (sys[3].addr ),
616   .sys_wdata       (sys[3].wdata),
617   .sys_wen         (sys[3].wen  ),
618   .sys_ren         (sys[3].ren  ),
619   .sys_rdata       (sys[3].rdata),
620   .sys_err         (sys[3].err  ),
621   .sys_ack         (sys[3].ack  )
622 );
623
```

Slika 56: »red_pitaya_proc« po dodatku novih signalov

Ostanejo le še povezave na hitre analogne vhode in izhode. Zaradi kompatibilnosti z ostalimi projekti na LNIV spletni strani, bomo dodali še možnost digitalne povratne povezave. Naslednji dve vrstici dodamo pred povezavo »red_pitaya_proc«:

```
assign proc_i[0] = digital_loop ? asg_dat[0] : {adc_dat_raw[0][14-1], ~adc_dat_raw[0][14-2:0]};
assign proc_i[1] = digital_loop ? asg_dat[1] : {adc_dat_raw[1][14-1], ~adc_dat_raw[1][14-2:0]};
```

Dodamo vodili za signalni generator in zajem podatkov »proc_i« in »proc_o« v vrstico 184:

```
// PID > proc
SBA_T [2-1:0]           proc_i;
SBG_T [2-1:0]           proc_o;
```

Na koncu preoblikujemo še ostale signal, ki uporabljajo PID signale. V razdelku DAC IO, ki se nahaja med vrsticama 380 in 390 [Slika 57], spremenimo signala:

```
387 ////////////////////////////////////////////////////////////////////////////////
388 // DAC IO
389 ////////////////////////////////////////////////////////////////////////////////
390
391 // Sumation of ASG and PID signal perform saturation before sending to DAC
392 //assign dac_a_sum = asg_dat[0] + pid_dat[0];
393 //assign dac_b_sum = asg_dat[1] + pid_dat[1];
394
395 assign dac_a_sum = asg_dat[0] + proc_o[0];
396 assign dac_b_sum = asg_dat[1] + proc_o[1];
```

Slika 57: Sprememba ostalih PID signalov

Komponenti »proc_i« dodamo še statičen parameter, s katerim lahko na drugih modelih Red Pitaye po potrebi spremenimo število GPIO priključkov. Na koncu celotna povezava v »red_pitaya_top« izgleda tako [Slika 58]:

```
601 ////////////////////////////////////////////////////
602 // Custom processing component
603 ////////////////////////////////////////////////////
604
605 assign proc_i[0] = digital_loop ? asg_dat[0] : {adc_dat_raw[0][14-1], ~adc_dat_raw[0][14-2:0]};
606 assign proc_i[1] = digital_loop ? asg_dat[1] : {adc_dat_raw[1][14-1], ~adc_dat_raw[1][14-2:0]};
607
608 red_pitaya_proc #(
609     // GPIO width
610     .DW             (DWE)
611 )
612 i_proc(
613    // signals
614   .clk_i           (adc_clk   ),    // clock
615   .rstn_i          (adc_rstn  ),    // reset - active low
616   .dat_a_i         (proc_i[0]),  // in 1 from ADC or loopback
617   .dat_b_i         (proc_i[1]),  // in 2 from ADC or loopback
618   .dat_a_o         (proc_o[0]),  // out 1 to DAC
619   .dat_b_o         (proc_o[1]),  // out 2 to DAC
620   // GPIO + LED
621   .led_o           (led_o   ),      // LED output
622   .gpio_p_i        (exp_p_in ),     // input data
623   .gpio_p_o        (exp_p_out),     // output data
624   .gpio_p_dir      (exp_p_dir),     // 1-output enable
625   .gpio_n_i        (exp_n_in ),
626   .gpio_n_o        (exp_n_out),
627   .gpio_n_dir      (exp_n_dir),
628   // System bus
629   .sys_addr        (sys[3].addr ),
630   .sys_wdata       (sys[3].wdata),
631   .sys_wen         (sys[3].wen  ),
632   .sys_ren         (sys[3].ren  ),
633   .sys_rdata       (sys[3].rdata),
634   .sys_err         (sys[3].err  ),
635   .sys_ack         (sys[3].ack  )
636 );
```

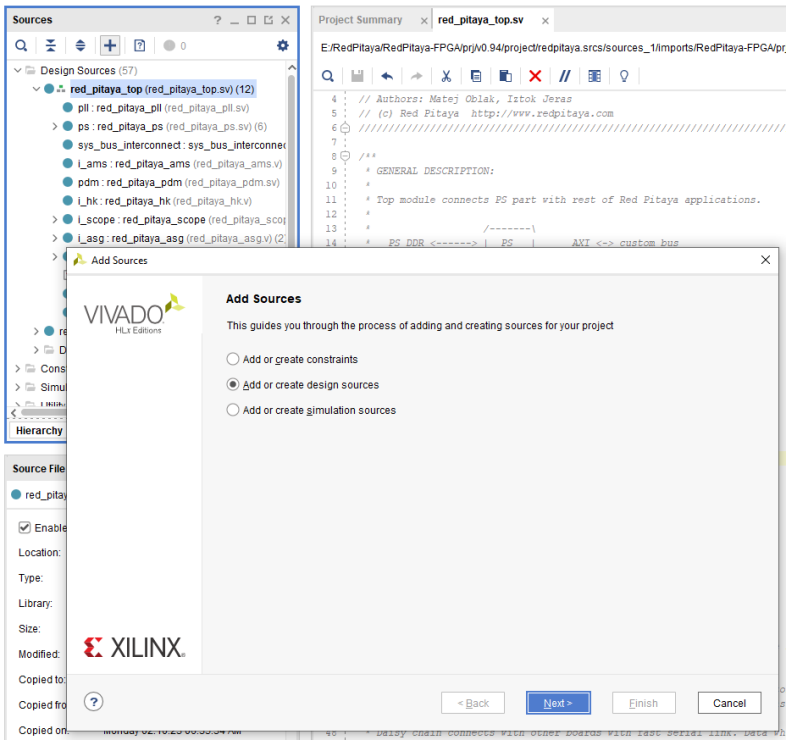
Slika 58: Končna povezava nove komponente

Ne smemo pozabiti, da smo s prevezavo digitalnih vhodov in izhodov, ter LED diod iz »red_pitaya_hk« onemogočili njihovo nadzorovanje preko C API ukazov [3]. Sedaj moramo narediti nove registre, ki nam bodo omogočali nadzor nad njimi.

5.4.2 Zasnova nove komponente

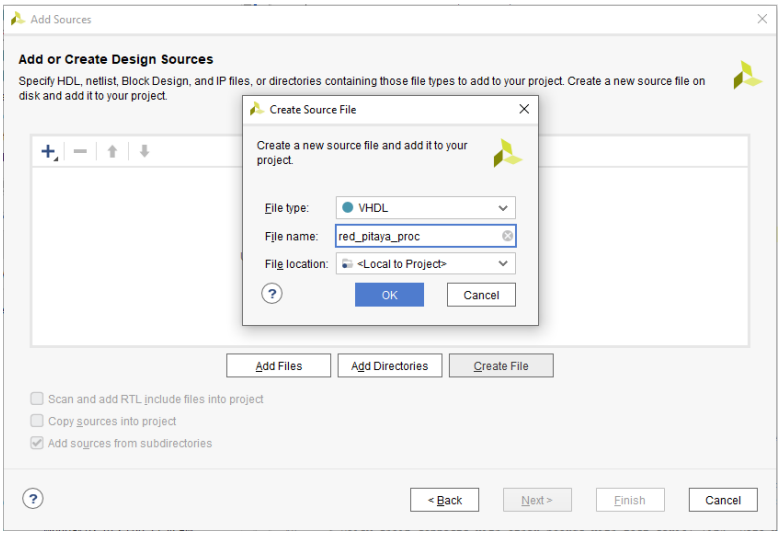
Komponento »red_pitaya_proc« bomo zaradi skladnosti z ostalimi vajami napisali v VHDL jeziku. Kombiniranje hardversko-opisnih jezikov VHDL in Verilog je mogoče zaradi njihove modularnosti. Vsak modul se obnaša kot črna škatla z vhodnimi in izhodnimi signali, zato pri priključitvi ni pomembno kaj se nahaja znotraj samega modula.

Nov modul dodamo s klikom na »*+*« v zavihku »*viri*« [Slika 59]:



Slika 59: Dodatek nove komponente

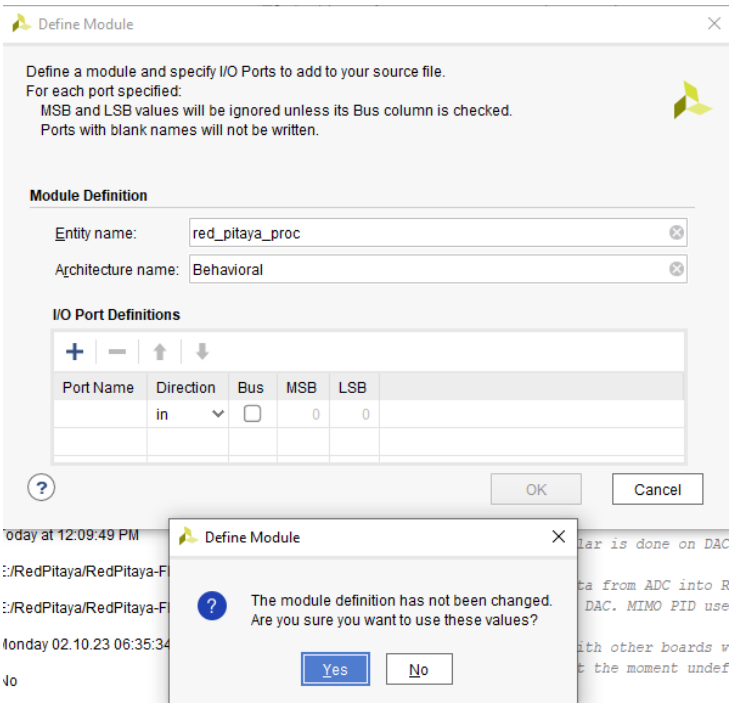
Izberemo možnost dodatka načrtovalskega vira (druga možnost). V naslednjem oknu izberemo »ustvari novo datoteko«. Za tip datoteke izberemo »VHDL« in nov modul poimenujemo »red_pitaya_proc«. Lokacijo lahko pustimo lokalno glede na projekt [Slika 60].



Slika 60: Poimenovanje nove komponente

Nato kliknemo »zaključi«.

Odpre se okno za določitev vhodno-izhodnih signalov modula. Ker bomo vse signale določili ročno, kliknemo naprej in potrdimo našo odločitev [Slika 61].



Slika 61: Določitev vhodno-izhodnih signalov

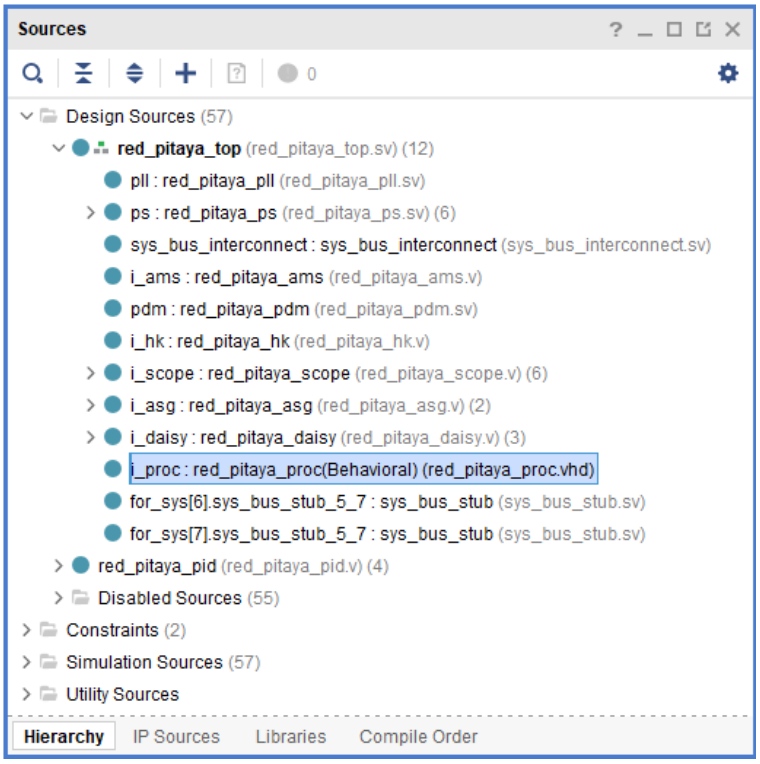
V nov modul skopirajmo spodnjo kodo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity red_pitaya_proc is
  generic(
    DW  : integer := 8
  );
  port (
    clk_i                 : in  std_logic;                      -- bus clock
    rstn_i                : in  std_logic;                      -- bus reset - active low
    dat_a_i, dat_b_i      : in  std_logic_vector(13 downto 0);  -- input
    dat_a_o, dat_b_o      : out std_logic_vector(13 downto 0);  -- output

    led_o                 : out std_logic_vector(7 downto 0);   -- LED output
    gpio_p_i, gpio_n_i    : in  std_logic_vector(DW-1 downto 0); -- GPIO input data
    gpio_p_o, gpio_n_o    : out std_logic_vector(DW-1 downto 0); -- GPIO output data
    gpio_p_dir, gpio_n_dir  : out std_logic_vector(DW-1 downto 0); -- GPIO direction

    sys_addr  : in  std_logic_vector(31 downto 0);            -- bus address
    sys_wdata : in  std_logic_vector(31 downto 0);            -- bus write data
    sys_wen   : in  std_logic;                                -- bus write enable
    sys_ren   : in  std_logic;                                -- bus read enable
    sys_rdata : out std_logic_vector(31 downto 0);            -- bus read data
    sys_err   : out std_logic;                                -- bus error indicator
    sys_ack   : out std_logic                                 -- bus acknowledge signal
    );
end red_pitaya_proc;

architecture Behavioral of red_pitaya_proc is

    constant ZERO               : std_logic_vector(32-1 downto 0) := (others => '0');

    signal diop_in, dion_in     : std_logic_vector(DW-1 downto 0);
    signal diop_out, dion_out   : std_logic_vector(DW-1 downto 0) := (others => '0');  -- output 0
    signal diop_dir, dion_dir   : std_logic_vector(DW-1 downto 0) := (others => '0');  -- direction in=0, out=1
```

```
    signal led : std_logic_vector(7 downto 0) := (others => '0');


    signal a, b: std_logic_vector(7 downto 0); -- amplitude registers
    signal mul_a, mul_b: signed(22 downto 0);

begin

    -- multiply signed inputs with 8-bit register, register values are unsigned
    mul_a <= signed(dat_a_i) * signed('0' & a);

    -- divide by 16 (multiplication format 4.4), possible output overflow
    dat_a_o <= std_logic_vector(mul_a(17 downto 4));

    pbus: process(clk_i)
    begin
        if rising_edge(clk_i) then
            if rstn_i = '0' then
                diop_dir <= (others => '0');
                dion_dir <= (others => '0');
                diop_out <= (others => '0');
                dion_out <= (others => '0');
                led <= (others => '0');

                a <= x"10";
            else
                sys_ack <= sys_wen or sys_ren;    -- acknowledge transactions

                if sys_wen='1' then                                 -- decode address & write registers
                    if sys_addr(19 downto 0)=X"00010" then
                        diop_dir <= sys_wdata(DW-1 downto 0);       -- Change direction P
                    elsif sys_addr(19 downto 0)=X"00014" then
                        dion_dir <= sys_wdata(DW-1 downto 0);       -- Change direction N
                    elsif sys_addr(19 downto 0)=X"00018" then
                        diop_out <= sys_wdata(DW-1 downto 0);       -- Change output P
                    elsif sys_addr(19 downto 0)=X"0001C" then
                        dion_out <= sys_wdata(DW-1 downto 0);       -- Change output N
                    elsif sys_addr(19 downto 0)=X"00030" then
                        led <= sys_wdata(7 downto 0);               -- Change LEDs
                    elsif sys_addr(19 downto 0)=X"00054" then
                        a <= sys_wdata(7 downto 0);                 -- 8-bit amplitude
                    end if;
                end if;
            end if;
        end if;
    end process pbus;

    -- Handling errors
    sys_err <= '0';

    -- Direct connections
    gpio_p_dir <= diop_dir;
    gpio_n_dir <= dion_dir;
    gpio_p_o <= diop_out;
    gpio_n_o <= dion_out;
    diop_in <= gpio_p_i;
    dion_in <= gpio_n_i;
    led_o <= led;

    -- Decode address & read data
    with sys_addr(19 downto 0) select
        sys_rdata <= X"FE240000"                        when x"00050",      -- ID
                     ZERO(32-1 downto DW) & diop_dir    when x"00010",      -- GPIO P direction
                     ZERO(32-1 downto DW) & dion_dir    when x"00014",      -- GPIO N direction
                     ZERO(32-1 downto DW) & diop_out    when x"00018",      -- GPIO P output
                     ZERO(32-1 downto DW) & diop_out    when x"0001C",      -- GPIO N output
                     ZERO(32-1 downto DW) & diop_in     when x"00020",      -- GPIO P inputs
                     ZERO(32-1 downto DW) & dion_in     when x"00024",      -- GPIO N inputs
                     ZERO(32-1 downto 8) & led          when x"00030",      -- LEDs
                     ZERO(32-1 downto 8) & a            when x"00054",      -- Amplitude
                     ZERO when others;

end Behavioral;
```

Koda novega modula je izredno preprosta. Omogoča interakcijo z LED in digitalnimi vhodi in izhodi preko pisanja in branja registrov. Poleg tega pa skalira signal iz hitrega analognega vhoda 1 (IN1) z vrednostjo 8-bitne amplitude »a«, ter rezultat pošlje na izhod 1 (OUT1). Koda prikazuje tudi osnovni koncept branja in pisanja v registre preko naslovov določenih v VHDL kodi. Naslovno vodilo Red Pitaya ima 32 bitov. Zgornjih 12 je določenih z razdelkom, kamor je priključena komponenta. Ker smo nadomestili komponento PID je prvi veljavni naslov komponente »red_pitaya_proc« 0x40300000. Spodnjih 20 bitov naslovnega prostora pa lahko prosto razdelimo med registre znotraj kode same komponente. Tako lahko na primer z branjem naslova 0x40300050 (torej osnova »0x40300000« plus »0x50« zamika) preberemo ID naše komponente in se prepričamo, da je FPGA pravilno konfiguriran.

Po shranitvi, se bo »*red_pitaya_proc*« avtomatsko vključil v projekt, saj smo povezavo na vrhnjo komponento naredili vnaprej [Slika 62].

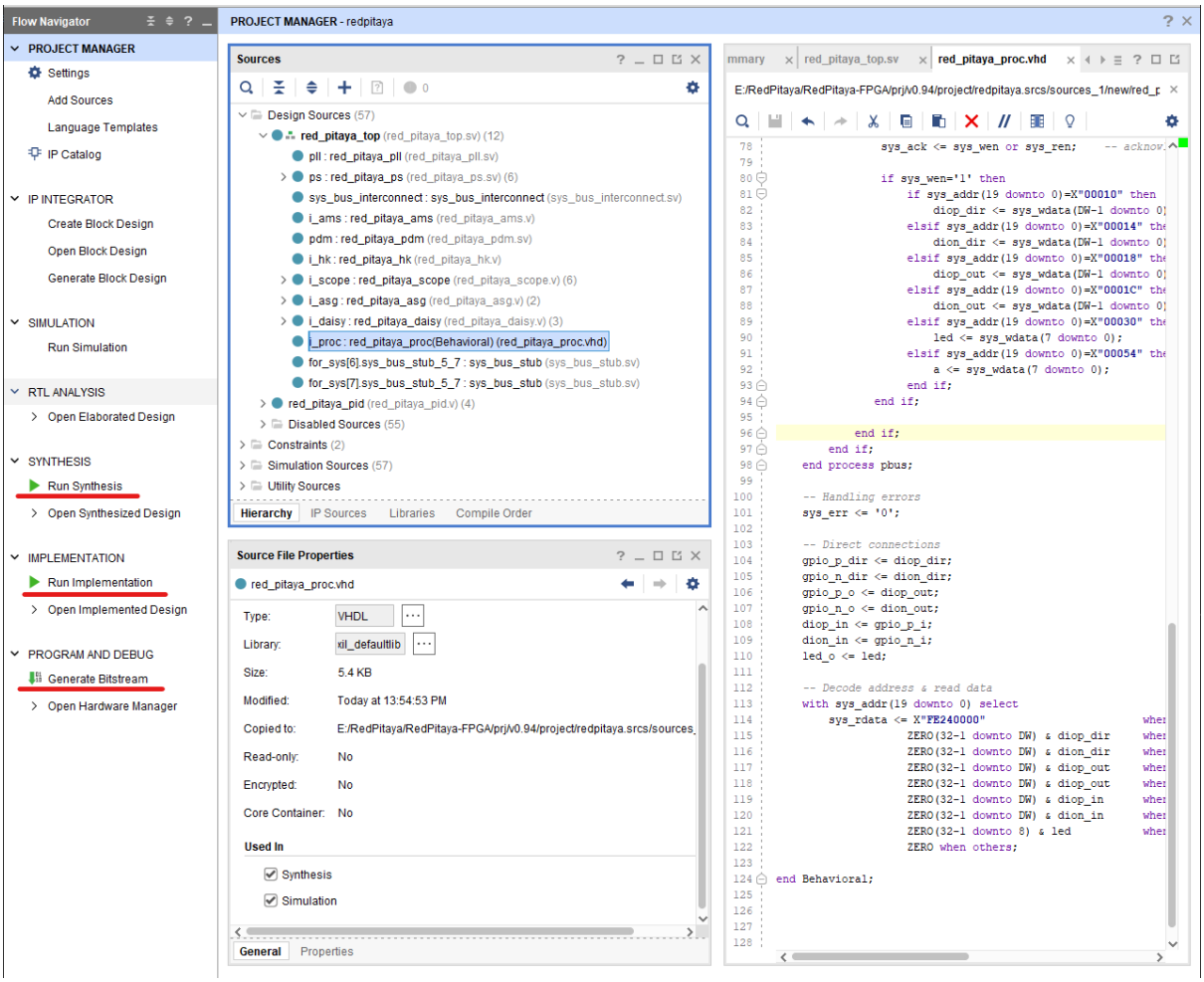


Slika 62: Spremembe v hierarhiji po dodatku nove komponente

Zaradi preglednosti so vsi nepotrebni moduli izključeni iz projekta. Vidimo lahko, da je modul »red_pitaya_proc« nadomestil modul »red_pitaya_pid«, saj je ta izključen iz spostnega menija vrhnje komponente »red_pitaya_top«. V primeru, da je komponenta »red_pitaya_proc« vključena v projekt, vendar se znotraj glavne hierarhije nahaja datoteka z vprašajem, moramo preveriti, če se ime entitete na nivoju »red_pitaya_top« ujema z imenom entitete v »red_pitaya_proc«.

5.4.3 Generacija bitstream datoteke

Sedaj, ko smo modificirali osnovni v0.94 projekt, je čas, da generiramo bitstream datoteko in reprogramiramo FPGA na Red Pitayi.



Slika 63: Generacija bitstreama

Konfiguracijsko datoteko oziroma bitstream generiramo z zaporednim zagonom sinteze, implementacije in generacije bitstreama, v naštetem vrstnem redu [Slika 63].

Med sintezo Vivado avtomatsko optimizira kodo, jo pretvori v dostopne FPGA komponente in določi povezave med njimi. Korak sinteze je zelo podoben izrisu shematike pri načrtovanju tiskanih vezij. Vivado preveri tudi vse pravila in javi morebitne napake v kodi ali povezavah.

Med implementacijo se sintetizirane povezave implementirajo na samem FPGA, podobno kot pri povezovanju komponent na tiskanini na podlagi sheme znotraj programa kot je npr. Altium ali KiCad.

V koraku generacije bitstreama se implementacija pretvori v binarna navodila za reprogramiranje FPGA vezja.

Za uspešno generacijo morajo biti vsi trije koraki izvedeni uspešno, brez večjih napak. O vseh odkritih napakah in obvestilih smo obveščeni preko zavihka »sporočila« v spodnjem podoknu programa.

Generirano bitstream datoteko najdemo v mapi »RedPitaya-FPGA\prj\v0.94\project\redpitaya.runs\impl_1« kot »red_pitaya_top.bit«.

5.5 Reprogramiranje FPGA na Red Pitayi

Reprogramiranje FPGA vezja na Red Pitayi je dokaj enostavno in ga lahko izvedemo tekom delovanja naprave. Najprej prenesemo bitstream datoteko na Red Pitayo in nato zaženemo ukaz za reprogramiranje. FPGA ohranja trenutno sliko do ponovnega reprogramiranja, ki se zgodi:

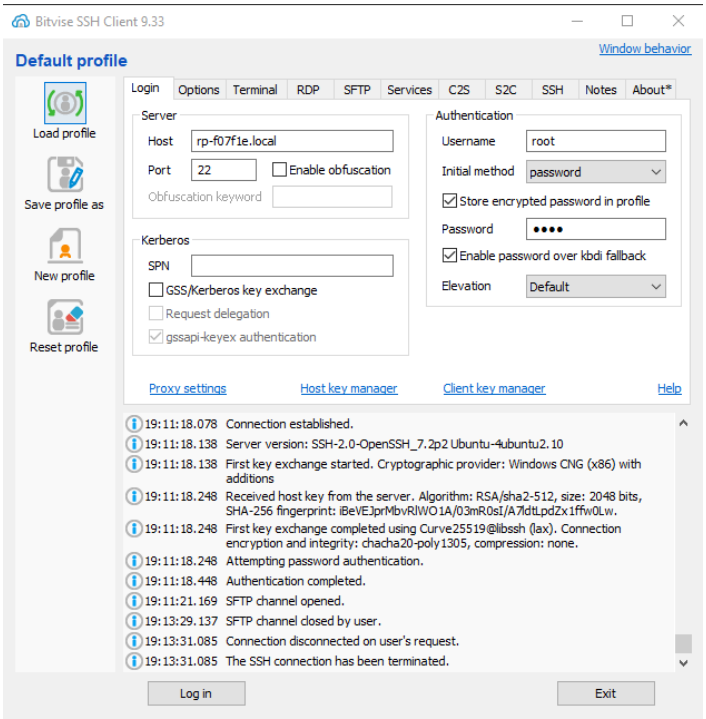
- Ob ponovnem zagonu naprave.
- Ob zagonu katerekoli aplikacije.
- Ob ročnemu reprogramiranju FPGA vezja.

V primeru, da želimo spremeniti FPGA sliko, ki se naloži ob zagonu določene aplikacije, oziroma reprogramirati FPGA na novejšem operacijskem sistemu, moramo izvesti še nekaj dodatnih korakov.

5.5.1 Prenos bitstreama na Red Pitayo

Ko v Vivadu uspešno ustvarimo bitstream oziroma ».bit« datoteko, jo najprej prenesemo na Red Pitayo. Datoteko najdemo v mapi »**RedPitaya-FPGA\prj\v0.94\project\redpitaya.runs\impl_1**«. Poimenovana je »**red_pitaya_top.bit**«. Pot do datoteke se lahko razlikuje glede na uporabljen projekt oziroma kombinacijo zastavic, s katerimi smo ustvarili projekt. Če uporabljamo drugačen model Red Pitaya, moramo na primer odpreti direktorij »v0.94_250« namesto »v0.94«. Preostala pot ostane nespremenjena. [26]

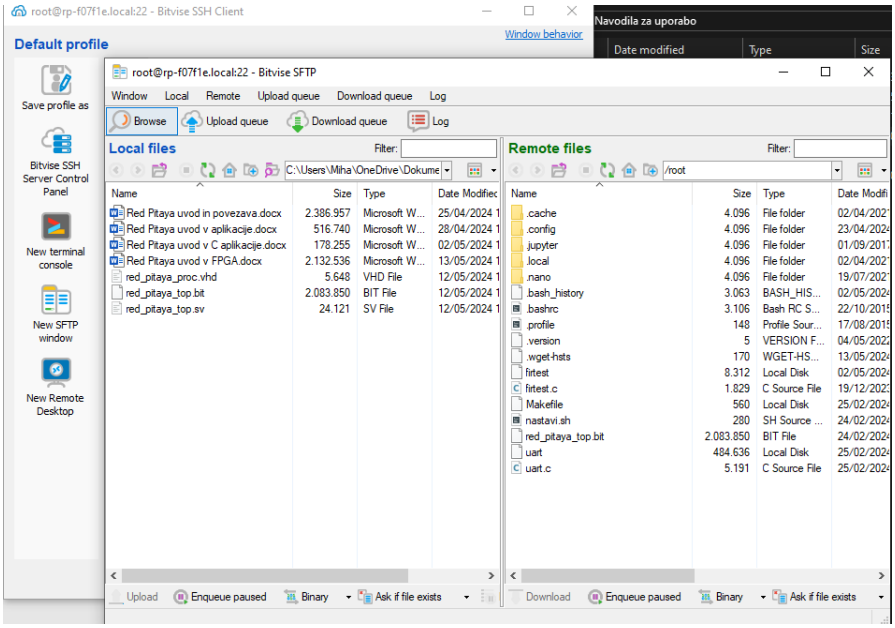
Datoteko lahko kopiramo na Red Pitayo s pomočjo programa kot je npr. »Bitvise SSH Client« [10] ali pa uporabimo Terminal oziroma Ukazni poziv.



Slika 64: Program Bitvise SSH Client [10]

V programu »Bitvise SSH Client« [Slika 64] vpišemo podatke za server. V polje »host« vpišemo *Red Pitayin IP naslov* (lahko pa uporabimo lokalni ».local« nasloa), v polje »port« vpišemo *22*, nato pa pod »avtorizacijo« izpolnimo »uporabniško ime« *root*, ter »geslo« *root*. Zatem kliknemo »Vpiši se«.

V levostranskem meniju se, po uspešni povezavi, pojavijo nove možnosti. Izberemo »novo SFTP okno« [Slika 65].



Slika 65: Novo SFTP okno

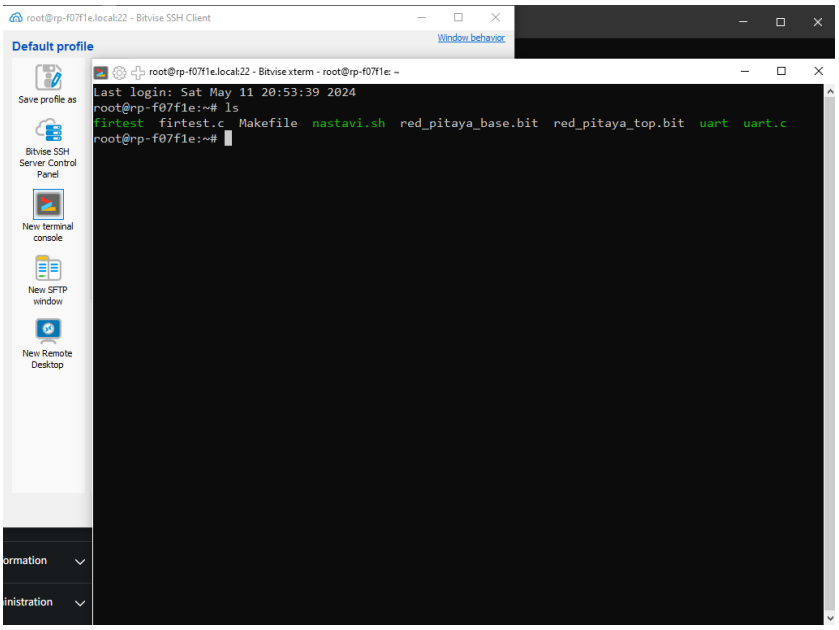
Na desni strani okna se nahaja datotečni sistem Red Pitaye, odprt v domači mapi »root«, na levi strani pa datotečni sistem našega računalnika. Na levi strani se navigiramo do mapo z ».bit« datoteko in jo prenesemo na Red Pitayo. Datoteke med datotečnima sistemoma prenašamo s povlekom iz ene na drugo stran.

Sicer pa lahko uporabimo Ukazni poziv in prenos datoteke izvedemo z ukazom »*scp*« (»*secure copy*«):

```
scp -r »/pot/do/red_pitaya_top.bit« »root@<IP Red Pitaye>:/root«
```

5.5.2 Reprogramiranje FPGA

Zatem se na Red Pitayo povežemo z SSH povezavo; Ali s klikom na »novo terminalno okno« (»new terminal console«) [Slika 66],



Slika 66: Novo terminalno okno

ali pa z ukazom:

```
ssh root@<IP Red Pitaye>
```

in vpisom gesla »root«.

FPGA konfiguracijo zamenjamo z ukazom:

```
cat red_pitaya_top.bit > /dev/xdevcfg
```

Na koncu preverimo identifikacijsko številko FPGA slike, ki smo jo določili znotraj VHDL kode:

```
monitor 0x40300050
```

Če so bili vsi koraki izvedeni pravilno, Red Pitaya izpiše ***0xfe240000***.

5.5.3 Menjava FPGA slike aplikacij

Pri izvedbi vaj bomo spremenili FPGA konfiguracijo, ki se naloži ob zagonu posamezne aplikacije. Za menjavo potrebujemo bash program »*nastavi.sh*«, katerega najdemo med datotekami za vaje:

```
#!/bin/bash
if [ $# -eq 0 ]
then
conf="/opt/redpitaya/fpga/fpga_0.94.bit"
else
conf=$(realpath $1)
#echo "test1/2" > file.conf
fi

mount -o rw,remount $PATH_REDPITAYA
echo $conf > /opt/redpitaya/www/apps/la_pro/fpga.conf
mount -o ro,remount $PATH_REDPITAYA
echo "set to "$conf
```

Program »*nastavi.sh*« konfigurira datoteko »*fpga.conf*«, ki se nahaja v mapi vsake izmed aplikacij in vsebuje pot do bistream datoteke, ki se naloži ob zagonu aplikacije. Vsebino »*fpga.conf*« zamenjamo s potjo do našega »*bitstreama*«. V primeru, da ne podamo poti do datoteke, se »*fpga.conf*« ponastavi v originalno različico.

Datoteko »nastavi.sh« najprej kopiramo na Red Pitayo in omogočimo njen zagon z ukazom:

```
chmod +x nastavi.sh
```

Ob zagonu programa podamo pot do bitstream datoteke:

```
./nastavi.sh red_pitaya_top.bit
```

5.5.4 Reprogramiranje na OS 2.00

Pri uporabi različice Red Pitaya operacijskega sistema 2.00 ali višje, se postopek spremembe FPGA spremeni. Pred pošiljanjem bitstream datoteke na Red Pitayo moramo izvesti dodaten korak, kjer spremenimo obliko datoteke. Za reprogramiranje pa uporabimo drugačen ukaz [26].

1. Odpremo Vivado 2020.1 in se preko »Tcl konzole« premaknemo v direktorij z bitstream datoteko. Izvedemo naslednja ukaza:

```
echo all:{ red_pitaya_top.bit } > red_pitaya_top.bif
bootgen -image red_pitaya_top.bif -arch zynq -process_bitstream bin -o
red_pitaya_top.bit.bin -w
```

Prvi ukaz ustvari datoteko s potjo do naše bitstream datoteke, nato pa drugi ukaz pretvori datoteko v binarno obliko s pomočjo orodja »bootgen«.

2. Datoteko »***red_pitaya_top.bit.bin***« kopiramo na Red Pitayo
3. Znotraj Red Pitayinega Linux terminala poženemo ukaz

```
fpgautil -b red_pitaya_top.bit.bin
```

Menjava FPGA aplikacij na OS 2.00

Za menjavo zagonske FPGA konfiguracija, ki se naloži ob zagonu Red Pitaye oziroma posamezne aplikacije na OS 2.00, potrebujemo skripto »*nastavi2.sh*«. Program zamenja originalno »fpga.bit.bin« datoteko, ki se nahaja v mapi »*/opt/redpitaya/fpga/z10_125/v0.94*« z podano bitstream datoteko. Hkrati pa poskrbi, da se originalna konfiguracija »*fpga.bit.bin*« ohrani.

```
#!/bin/bash
BITSTREAM=$1
MODEL=$(/opt/redpitaya/bin/monitor -f)
PROJ=v0.94

mount -o rw,remount $PATH_REDPITAYA
cp  -n  "/opt/redpitaya/fpga/$MODEL/$PROJ/fpga.bit.bin"  "/opt/redpitaya/fpga/$MODEL/$PROJ
/fpga_orig.bit.bin"

if [ $# -eq 0 ]
then
cp  -f  "/opt/redpitaya/fpga/$MODEL/$PROJ/fpga_orig.bit.bin"  "/opt/redpitaya/fpga/$MODEL
/$PROJ/fpga.bit.bin"
conf="Restored original fpga.bit.bin"
else
cp -f "$(realpath $1)" "/opt/redpitaya/fpga/$MODEL/$PROJ/fpga.bit.bin"
conf="fpga.bit.bin overwritten with $BITSTREAM"
fi

mount -o ro,remount $PATH_REDPITAYA
echo "$conf"
```

S spremembo »*fpga.bit.bin*« ne spremenimo samo bitstream datoteke za določeno aplikacijo, ampak zamenjamo FPGA konfiguracijo vseh aplikacij, ki za svoje delovanje uporabljajo projekt »v0.94«. Prav tako spremenimo Red Pitayino zagonsko FPGA konfiguracijo.

5.6 Razlike v postopku za druge modele Red Pitaye

V primeru, da želimo ustvariti FPGA projekt za drugo različico Red Pitaye, so vsi postopki pri reprogramiranju FPGA in kreaciji splošnega Red Pitaya projekta popolnoma enaki. Spremenijo se le uporabljene zastavice pri kreaciji projekta [4], [Tabela 3]:

Model	MODEL zastavica
STEMlab 125-10 STEMlab 125-14	Z10
STEMlab 125-14 Z7020	Z20_14
SDRlab 122-16	Z20
SIGNALlab 250-12	Z20_250
STEMlab 125-14 4-input	Z20_4

Tabela 3: MODEL zastavice pri kreaciji Red Pitaya FPGA projekta [4]

V primeru uporabe modela SIGNALlab 250-12, se spremeni tudi zastavica PRJ in sicer v v0.94_250.

Projekti različnih modelov Red Pitaye so na prvi pogled popolnoma enaki, vendar uporabljajo različne čipe (FPGA, ADC, …), ki spremenijo velikosti in število posameznih priključkov. Zato projekti med seboj niso kompatibilni in ne delujejo na drugih modelih Red Pitaye (npr., pri uporabi STEMlab 125-14 4-input nimamo dostopa do hitrih analognih izhodov).

Nasveti v primeru težav

Tukaj je nekaj uporabnih nasvetov za reševanje težav z projektom.

- Pred generacijo bitstreama je potrebno vse novo ustvarjene komponente simulirati. S tem se prepričam, da vse datoteke delujejo pravilno in hkrati hitro odpravimo morebitne napake. Generacija bitstreama, kopiranje datotek na Red Pitayo in testiranje je zelo zamuden način za odkritje in odpravo napak.
- Napake se lahko pojavijo na različnih mestih. Na primer, v kodi komponente, pri povezavi posameznih komponent, uporaba napačnega projekta, … . S simulacijo lahko hitro odkrijemo prisotnost napak v posameznih komponentah in zožimo področje iskanja.
- Pri generaciji projekta moramo paziti, da uporabimo pravilne zastavice za našo verzijo Red Pitaye, saj v nasprotnem primeru »bitstream« na razvojni plošči ne bo deloval, čeprav celotna simulacija deluje pravilno. Red Pitaya v takšnem primeru javi napako zaradi napačnih vhodno-izhodnih priključkov:

```
sh: 1: echo: echo: I/O error
BIN FILE loading through FPGA manager failed
```

- Pri konfiguriranju datotek v Linux operacijskem sistemu moramo biti izredno previdni, da ponesreči ne izbrišemo kakšne pomembne datoteke, saj imamo popolen nadzor nad celotnim sistemom.
- V najslabšem primeru lahko vedno izbrišemo in ponovno naložimo celoten operacijski sistem, kar odpravi vse programske napake.

6 Aplikacije v jeziku C in Python

V tem poglavju si bomo pogledali kako lahko napišemo in zaženemo C in Python aplikacije znotraj Red Pitayinega Linux operacijskega sistema.

Preden začnemo s samim programiranjem potrebujemo naslednje:

- Urejevalnik kode z ustreznimi razširitvami za lažje programiranje v Python oziroma C kodi (priporočam Visual Studio Code [27])
- Ustrezno različico Python prevajalnika [28]

6.1 Programiranje v C kodi

Za pisanje programov v C kodi ne potrebujemo nobenih dodatnih orodij, saj lahko programe prevajamo neposredno na Red Pitayi. Kodo sicer lahko s pomočjo ustreznih razširitev razhroščujemo in testiramo v urejevalnikna na samem računalniku, ki pa se lahko izkaže kot precej težavno v primeru komunikacije z FPGA registri ali uporabo funkcij iz Red Pitayinih knjižnic. Celoten postopek lahko razbijemo na naslednje korake:

- Pisanje C programa.
- Kopiranje programa na Red Pitayo.
- Zagon programa na Red Pitayi.

Za komuniciranje z Red Pitayo bomo uporabili SSH povezavo [7].

Ne bomo se spuščali v splošna pravila za pisanje C kode. Če potrebujete pomoč, je Google vaš najboljši prijatelj. Veliko uporabnih nasvetov pa boste našli na spletni strani stackoverflow. Uporabite znanje, ki ste ga pridobili pri drugih predmetih oziroma z lastnim trudom. Vsa C koda na vajah je dokaj enostavna.

V tej sekciji se bomo posvetili začetnemu osnutku kode iz vaj ter splošnim navodilom za komunikacijo z različnimi perifernimi enotami Red Pitaye.

6.1.1 Opis delovanja osnutka C kode

Osnutek C programske kode za vaje je namenjen prikazu izmenjave podatkov med FPGA registri in C programom. Pri načrtovanju FPGA kode je izredno pomembno, da si zabeležimo na katerih naslovih se nahajajo parametri (na primer, nastavitev amplitude, vpis koeficientov, ID, itd.), saj nam to močno olajša pisanje C programa.

```
9
10  void Out32(void *adr, int offset, int value)
11  {
12      *((uint32_t *)(adr+offset)) = value;
13  }
14
15  int In32(void *adr, int offset)
16  {
17      return *((uint32_t *)(adr+offset));
18  }
19
20  int In16(void *adr, int offset)
21  {
22      int r;
23      r= *((uint32_t *)(adr+offset));
24      if (r > 32767) return r-65536;
25      return r;
26  }
27
```

Slika 67: *Funkcije za komunikacijo z FPGA registri*

Po vključitvi knjižnic, a pred glavno zanko, se nahajajo tri funkcije [*Slika 67*]:

- Out32
- In32
- In16

Z njimi lahko beremo in pišemo podatke na naslove FPGA registrov. Vsi FPGA registri, ki vsebujejo s strani uporabnika nastavljive parametre, so povezani s procesorskim vodilom, zato jih lahko naslovimo z vpisom pravilnega naslova.

- Funkcija *Out32* sprejme kazalec na naslovni prostor »*adr*«, naslovni zamik »*offset*«, in vrednost »*value*«. Funkcija sešteje osnovni naslov in zamik in ju spremeni v nepredznačeni 32-bitni kazalec, kamor se zapiše podana vrednost.
- *In32* deluje v obratni smeri kot *Out32* in vrne vrednost registra na podanem naslovu.
- Funkcija *In16*, deluje enako kot *In32*, le da prebrano vrednost pretvori v 16-bitna predznačeno število.

```
28
29  int main(int argc, char **argv)
30  {
31    int fd;
32    int BASE = 0x40300000; // bazni naslov komponente proc
33    void *adr;
34    char *name = "/dev/mem";
35    int i,d,id;
36
37    int coef[6] = {93, 93, 93, 93, 93, 93};
38
39    // odpri in mapiraj pomnilnik za dostop do registrov
40    if((fd = open(name, O_RDWR)) < 0) {
41      perror("open");
42      return 1;
43    }
44    adr = mmap(NULL, sysconf(_SC_PAGESIZE), PROT_READ|PROT_WRITE, MAP_SHARED, fd, BASE);
45
46    // Beri ID
47    id = In32(adr, 0x00);
48    printf("ID = %x\n", id);
49
```

Slika 68: Začetni del glavne funkcije

Celoten program je namenjen komunikaciji in nastavljanju parametrov v modificiranem FPGA vezju [Slika 68]. Za začetek moramo ugotoviti začetni naslov komponente »red_pitaya_proc«, ki smo jo ustvarili v poglavju »Dodatek lastne komponente v sistem«.Ker smo v osnovnem projektu nadomestili PID krmilnik, je začetni naslov 0x40300000. Mapa registrov osnovnega projekta v0.94 je dostopna tukaj (med načrti registrov ne vidimo načrta za operacijski sistem 1.04-28, zato uporabimo najstarejšo možno različico načrtov 2.00-15) [4].

Po definiranju spremenljivk odpremo Red Pitayin spomin v načinu za branje in pisanje. Registri se nahajajo v direktoriju »*/dev/mem*«. Zatem izrišemo spominski načrt z ukazom »*mmap*«. Registre bi lahko spreminjali tudi

ročno z ukazom »*monitor*«, ki je na voljo znotraj Linux terminala, vendar je rešitev preko C programa enostavnejša in bolj uporabna.

Kot primer imamo podano branje identifikacijskega registra znotraj modificirane FPGA slike s pomočjo funkcije *In32.*

Zatem sledi prostor namenjen kodi za komunikacijo in nastavitve FPGA vezja.

Povsem na koncu najdemo še sledečo vrstico, ki zapre oziroma sprosti registrski načrt [Slika 69].

```
76
77      munmap(adr, sysconf(_SC_PAGESIZE));
78      return 0;
79  }
80
```

Slika 69: Zaključek programa

6.1.2 Programske knjižnice na Red Pitayi

C

Red Pitaya že vključuje različne C programske knjižnice, ki omogočajo nadzor nad vsemi perifernimi enotami. Med vajami jih ne bomo uporabljali, saj je cilj predmeta modifikacija in komunikacija z FPGA.

Med njimi sta najbolj uporabni:

- ***rp*** – vsebuje večino funkcij za nadzor nad digitalnimi vhodi in izhodi, LED, ter hitrimi analognimi vhodi in izhodi.
- ***rp_hw*** – nadzor nad funkcijami za digitalno komunikacijo.

Vključimo jih tako kot vsako drugo C knjižnico, edina razlika je, da delujejo le znotraj operacijskega sistema na Red Pitayi.

Vse C knjižnice in vključene funkcije lahko najdemo v razdelku rp-api na Red Pitaya GitHubu. Pazimo le, da gledamo pravo vejo GitHub kode (za 1.04-28 izberemo vejo »*release-2022.2*«) [29].

Opise posameznih funkcij lahko najdemo v »*.h*« datotekah na GitHub repozitoriju [29] in pa v seznamu ukazov (orientiramo se po stolpcu »ekosistem«) [30], kjer najdemo tudi tabelo kompatibilnosti operacijskih sistemov in GitHub vej [Tabela 2].

Python

Python razvojno okolje je na voljo na Red Pitaya operacijskih sistemih novejših od 2.00-30, kjer se trenutno uporablja različica Python 3.10. Vsaka Python funkcija ima istoimensko C funkcijo in z nekaj izjemami sprejema tudi enake parametre. Python knjižnice lahko najdemo v direktoriju *»/opt/redpitaya/lib/python*« znotraj Red Pitaya Linux operacijskega sistema [3].

Glavni knjižnici sta »*rp.py*« in »*rp_hw.py*«, ki imata enako funkcionalnost kot istoimenski C knjižnjici.

Vse Python funkcije v ozadju neposredno kličejo istoimenske C funkcije, zato kot enega izmed parametrov vedno vrnejo tudi uspešnost izvedbe, kar je za Python neobičajno. Prvi vrnjen parameter je vedno uspešnost izvedbe zagnane funkcije, vsi nadaljnji parametri pa so pričakovani podatki.

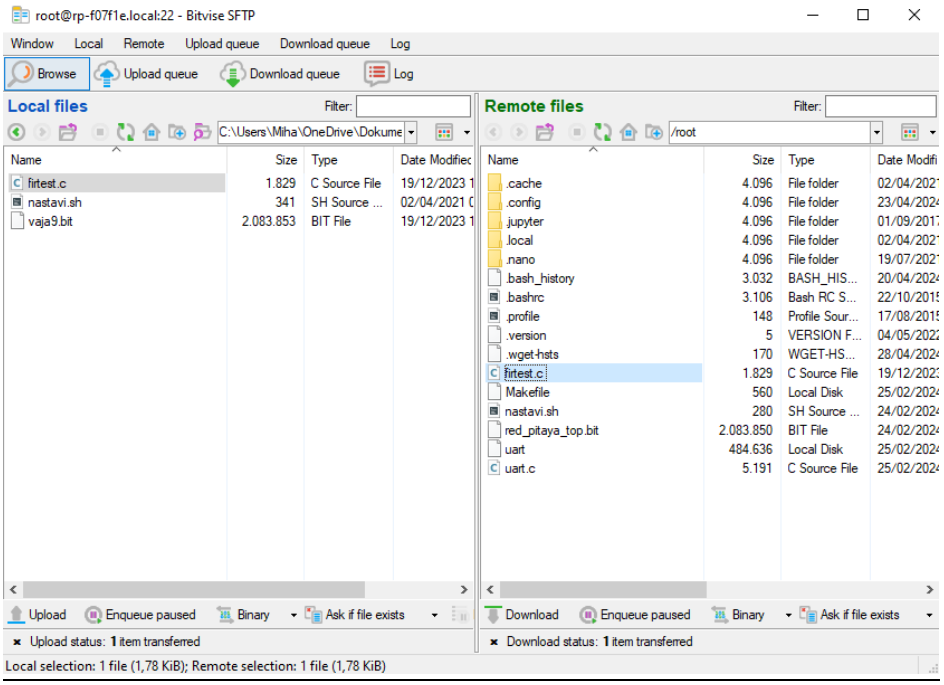
6.2 Prenos in prevajanje programske kode

Ko smo z našim programom zadovoljni ga lahko prenesemo in testiramo na Red Pitayi. Pred tem preverimo, da je »*končna vrstična sekvenca*« (angl. »*End of line sequence*«) nastavljen na LF (angl. »*Line Feed*«) in ne na CRLF (angl. »*Carrige Return Line Feed*«) oziroma CR (angl. »*Carrige Return*«) [Slika 70]. Ta nastavitev nam lahko povzroča precej preglavic pri zagonu C programa, saj prevajalnik javi napako zaradi nedefiniranega znaka oziroma ukaza, poda pa prvo vrstico programa. Po spremembi ne smemo pozabiti shraniti programa.

Slika 70: Pozicija nastavitve »končne vrstične sekvence« v programu VSCode

6.2.1 Prenos programa na Red Pitayo

Preko aplikacije BitWise SSH, oziroma preko terminala vzpostavimo SSH povezavo z Red Pitayo, kot je to opisano v poglavju »SSH povezava«. Nato program prenesemo iz direktorija na računalniku v domači direktorij (»*/root*«) na Red Pitayi [Slika 71].



Slika 71: Prenos programa na Red Pitayo s programom Bitvise SSH

6.2.2 Prevajanje programa v jeziku C

Preden lahko program zaženemo na Red Pitayi, ga moramo prevesti. V okviru vaj, oziroma ko ne uporabljamo Red Pitayinih knjižnic, to lahko storimo z ukazom »*gcc*«, ki zažene Linuxov prevajalnik GCC (»GNU Compiler Collection). Primer prevedbe programa:

```
gcc -o test test.c
```

GCC vzame program »*test.c*«, ga prevede, in izhod shrani kot »*test*«. Pri prevedbi nam prevajalnik javi morebitne napake, ki jih moramo pred uspešno prevedbo odpraviti.

Program moramo po vsaki posodobitvi kode prevesti, saj se sicer zažene starejša različica kode. Najbolje je prevajanje izvesti ob vsakem kopiranju programa iz računalnika na Red Pitayo ali urejanju programa znotraj Linux operacijskega sistema.

Program nato poženemo z vpisom »*./test*« v terminal.

Uporaba Red Pitaya programskih knjižnic

Postopek prevajanja je daljši, če smo v program vključili tudi Red Pitayine programske knjižnice, saj moramo med prevedbo ali kasnje programu podati povezavo do njih. Najlažje to storimo z uporabo datoteke z navodili, ki ji pravimo »Makefile«, nato pa program prevedemo z uporabo ukaza »make«. »make« je Linux orodje, ki omogoča enostavnejšo izvedbo vnaprej določenega zaporedja ukazov, ki ga določimo znotraj datoteke »Makefile«. Za manjše projekte je uporaba priporočena, pri večjih projektih, ki zavzemajo več programskih datotek pa skoraj nujna, saj prihrani veliko dela pri pisanju ukazov.

Datoteko »Makefile« za prevedbo C ukazov na Red Pitayi najdemo tukaj [3]. Pri izbiri moramo biti previdni, da izberemo datoteko, ki ustreza naši različici operacijskega sistema (1.04-28):

```
MODEL ?= Z10

CFLAGS  = -std=gnu11 -Wall ## -Werror
CFLAGS += -I/opt/redpitaya/include -D$(MODEL)
LDFLAGS = -L/opt/redpitaya/lib
LDLIBS = -static -lrp

ifeq ($(MODEL),Z20_250_12)
INCLUDE += -I/opt/redpitaya/include/api250-12
LDLIBS += -lrp-gpio -lrp-i2c
endif

LDLIBS += -lrp-hw -lm -lstdc++ -lpthread

# List of compiled object files (not yet linked to executable)

PRGS = digital_led_blink \
       api_example_2 \
       api_example_3 \
       api_example_x

OBJS := $(patsubst %,%.o,$(PRGS))
SRC := $(patsubst %,%.c,$(PRGS))

all: $(PRGS)

$(PRGS): %: %.c
        $(CC) $< $(CFLAGS) $(LDFLAGS) $(LDLIBS) -o $@

clean:
        $(RM) *.o
        $(RM) $(OBJS)
```

Pred kopiranjem datoteke na Red Pitayo, jo moramo prilagoditi naši kodi. Po shranitvi na računalnik, jo odpremo v urejevalniku kode, ter popravimo argument »*PRGS*«. Izbrišemo vse za enačajem in zatem zapišemo imena C programov, ki jih želimo prevesti. V imenih izpustimo končnico ».c«. Če želimo hkrati prevesti več programov vsako datoteko ločimo z znakom »\«.

»Makefile« shranimo in prenesemo na Red Pitayo v domači direktorij (»*/root*«).

Program lahko zatem enostavno prevedemo z ukazom:

```
make test
```

6.2.3 Zagon programa v jeziku Python

Pri zagonu Python programov ne potrebujemo dodatnih datotek, saj je se na Red Pitayi že nahaja Python prevajalnik. Poskrbeti moramo le, da program lahko zaženemo. To storimo z ukazom »*chmod*«.

```
chmod +x test.py
```

6.3 Dodatne informacije

Dodatne informacije lahko najdete na naslednjih spletnih straneh:

- Red Pitaya navodila za zagon C in Python aplikacij[30]
- Red Pitaya GitHub

V primeru težav

Tukaj lahko najdete rešitve za nekatere najpogostejše napake pri zagonu C in Python aplikacij.

Brez uporabe Red Pitaya knjižnic

- Na večino napak v programu vas opozori že sam prevajalnik, večino ostalih se da razrešiti z nekaj iskanja po spletu.
- Pri težavah s prevajanjem C in Python aplikacij preverite končno vrstično sekvenco »End of line sequence«, ki mora biti nastavljena na LF (»Line Feed«).

Z uporabo Red Pitaya knjižnic

- Poskrbite, da uporabljate pravilne funkcije, ki so na voljo v naloženem operacijskem sistemu. Hitro se lahko zgodi, da uporabite funkcije, ki so na voljo le v novejših različicah Red Pitaya OS. Vsaka funkcija na seznamu ukazov ima navedeno tudi različico operacijskega sistema, kjer je bila prvič na voljo [30].
- Preverite različico »Makefile« datoteke in poskrbite, da se ujema s tisto namenjeno za trenutni operacijski sistem [3]. Če vas program opozori, da je bil »Makefile« ustvarjen v prihodnosti, spremenite čas operacijskega sistema z ukazom »date«. Nastavite trenutni čas in datum. Primer:

```
date -s “19 Jan 2024 14:00:00”
```

- Pri zagonu Python programov na 2.00-30 Python knjižnice niso pravilno povezane. Uporabite navodila na Red Pitaya spletni strani [3].

7 Viri

[1] Red Pitaya d.o.o., „What is Red Pitaya?“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/intro.html

[2] Red Pitaya d.o.o., „Applications and Features“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/appsFeatures/appsFeatures.html

[3] Red Pitaya d.o.o., „C and Python Applications“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/appsFeatures/remoteControl/API_scripts.html

[4] Red Pitaya d.o.o., „Build FPGA image“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/fpga/fpga.html

[5] Red Pitaya d.o.o., „STEMlab 125-14“, Red Pitaya documentation. Pridobljeno: 15. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/top.html

[6] Red Pitaya d.o.o., „Connect to Red Pitaya“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/quickStart/first.html

[7] Red Pitaya d.o.o., „Establish remote SSH connection“, Red Pitaya documentation. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/developerGuide/software/console/ssh/ssh.html

[8] Red Pitaya d.o.o., „Connection types“, Red Pitaya documentation. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/quickStart/connect/connect.html

[9] Red Pitaya d.o.o., „FAQ“, Red Pitaya documentation. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/quickStart/troubleshooting/troubleshooting.html

[10] Bitvise Limited, *Bitvise SSH Client*. Bitvise Limited. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://bitvise.com/ssh-client-download

[11] balenaEcher.dev, *balenaEtcher*. balenaEcher.dev. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://etcher.balena.io/

[12] „Prepare SD card“, Red Pitaya. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/quickStart/SDcard/SDcard.html

[13] Red Pitaya d.o.o., „Oscilloscope & Signal Generator“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/appsFeatures/applications/oscSigGen/osc.html

[14] Red Pitaya d.o.o., „Bode Analyzer“, Red Pitaya documentation. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/appsFeatures/applications/bode/bode.html

[15] Advanced Micro Devices, Inc., *AMD Vivado™ Design Suite*. Advanced Micro Devices, Inc. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html

[16] Intel Corporation, *ModelSim-Intel® FPGAs Standard Edition*. Intel Corporation. Pridobljeno: 18. januar 2025. [Na spletu]. Dostopno na: https://www.intel.com/content/www/us/en/software-kit/750368/modelsim-intel-fpgas-standard-edition-software-version-18-1.html

[17] Red Pitaya d.o.o., „Installation of Vivado 2020.1“, Red Pitaya knowledge base. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://redpitaya-knowledge-base.readthedocs.io/en/latest/learn_fpga/3_vivado_env/tutorfpga1.html

[18] win.rar GmbH, *WinRAR*. win.rar GmbH. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://www.win-rar.com/start.html?&L=0

[19] LNIV, Faculty of Electrical Engineering, „LNIV Red Pitaya“, Laboratory for Integrated Circuit Design. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://lniv.fe.uni-lj.si/redpitaya/
[20] Red Pitaya d.o.o., *RedPitaya v0.94 top module*. System Verilog. Red Pitaya d.o.o. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://github.com/RedPitaya/RedPitaya-FPGA/blob/master/prj/v0.94/rtl/red_pitaya_top.sv
[21] Xilinx, Inc, „Vivado Design Suite: AXI Reference Guide (UG1037)“. 15. julij 2017. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://docs.amd.com/v/u/en-US/ug1037-vivado-axi-reference-guide
[22] Arm Ltd, „AMBA Specifications“, Arm | The Architecture for the Digital World. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://www.arm.com/architecture/system-architectures/amba/amba-specifications
[23] Red Pitaya d.o.o., „Project - v0.94 (2.05-37)“, Red Pitaya documentation. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/developerGuide/software/build/fpga/regset/2.05-37/v0.94.html
[24] Red Pitaya d.o.o., *RedPitaya-FPGA/prj/v0.94/rtl*. Verilog, System Verilog. Red Pitaya d.o.o. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://github.com/RedPitaya/RedPitaya-FPGA/blob/master/prj/v0.94/rtl/red_pitaya_top.sv
[25] Red Pitaya d.o.o., „LED Counter“, Red Pitaya knowledge base. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://redpitaya-knowledge-base.readthedocs.io/en/latest/learn_fpga/4_lessons/LedCounter.html
[26] Red Pitaya d.o.o., „Programming the FPGA“, Red Pitaya knowledge base. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://redpitaya-knowledge-base.readthedocs.io/en/latest/learn_fpga/3_vivado_env/tutorfpga2.html
[27] Microsoft, *Visual Studio Code*. Microsoft. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://code.visualstudio.com/
[28] Python Software Foundation, *Python*. (3. december 2024). Python. Python Software Foundation. Pridobljeno: 16. januar 2025. [Na spletu]. Dostopno na: https://www.python.org/
[29] Red Pitaya d.o.o., *Red Pitaya GitHub Repository*. Pridobljeno: 14. januar 2025. [Na spletu]. Dostopno na: https://github.com/RedPitaya
[30] „List of supported SCPI & API commands“, Red Pitaya documentation. Pridobljeno: 19. januar 2025. [Na spletu]. Dostopno na: https://redpitaya.readthedocs.io/en/latest/appsFeatures/remoteControl/command_list.html