Univerza v Ljubljani

Fakulteta za elektrotehniko

Primož Perušek

Razvoj LabVIEW gonilnikov za inštrument Red Pitaya

Diplomsko delo

Mentor: doc. dr. Marko Jankovec

Ljubljana, 2015

Zahvala

Hvala mentorju doc. dr. Marku Jankovcu za vsa navodila pri pisanju diplomske naloge ter za hitre in korektne odgovore na moja vprašanja. Hvala tudi podjetju Red Pitaya d.d., kjer sem imel možnost sodelovati na projektu razvoja LabVIEW-gonilnikov. Posebno bi se zahvalil Markusu Salamonu za učenje programa LabVIEW ter nasvete pri delu.

Povzetek

Uporabniki programirljivih inštrumentov želijo čim hitreje in enostavneje povezati inštrument z računalnikom. Če ti za kontrolo inštrumenta uporabljajo program LabVIEW, lahko uporabijo gonilnik za njihov inštrument.

V diplomski nalogi je opisan razvoj LabVIEW-gonilnikov za inštrument Red Pitaya. Opisano je tudi delovanje omenjenega inštrumenta ter katere funkcije se upravljajo z ukazi SCPI, ki omogočajo kontrolo inštrumenta z LabVIEW-aplikacijami. Namen omenjenega gonilnika je olajšati in pospešiti sestavljanje blokovnih diagramov v programu LabVIEW za kontrolo inštrumenta Red Pitaya.

Glede na funkcionalne zahteve, katerim morajo gonilniki ustrezati, se lotimo se razvoja funkcionalne in intuitivne zbirke LabVIEW-datotek, ki morajo zadostovati funkcionalnim zahtevam standardnih LabVIEW-gonilnikov. Najprej so opisani LabVIEW-gonilniki ter postopek njihovega razvoja. V tem je predstavljeno načrtovanje konceptnih VI blokov za inštrument Red Pitaya, ki nam bodo kasneje pomagali pri odločitvi strukture omenjenih blokov. Proces razvoja vključuje razvoj knjižnice VI blokov in komunikacije med računalnikom in inštrumentom ter se zaključi z gradnjo javne datoteke.

Rezultat razvoja je knjižnica LabVIEW-blokov, kateri omogočajo enostavnejšo povezljivost z Red Pitaya ploščo ter lažjo uporabo komand za kontrolo omenjene plošče. Ko te bloke združimo v en projekt s smiselnimi povezavami med njimi, jim rečemo LabVIEW Instrument Driver.

Ključne besede: LabVIEW, gonilnik, merilni inštrument, SCPI

Abstract

Users of programming instruments want to connect their instrument to a PC as fast and as easy as possible. In LabVIEW they can use an instrument driver.

In the batchelor's thesis is described a development of LabVIEW instrument driver for Red Pitaya. Described is also an operation of the instrument and which functions are used to control Red Pitaya with SCPI commands. The main purpose of the driver is to make the application programming with the instrument easire and faster.

We start with a development of functional and intuitive library of VI-s considering the functional requirements of standard LabVIEW drivers. At first we describe drivers and their development process. In the process we present a planning of concept VI blocks fort he instrument. The process of development includes a development of VI library and a communication between a personal computer and Red Pitaya. It finishes with a build of public file.

The resoult of the development is a library of LabVIEW blocks, enabling easier connectivity with a Red Pitaya board and an easier use of commands controlling the board. All developed VI blocks united in one project are called LabVIEW Instrument driver.

Key words: LabVIEW, driver, measurment instrument, SCPI

Vsebina

Seznam slik

Seznam uporabljenih kratic

FPGA (Field programmable gate array) mreža polj programirljivih vrat

API (Application programming interface) aplikacijski programski vmesnik

LabVIEW (Laboratory virtual instrument engineering workbench) inženirsko delovno okolje za laboratorijske virtualne inštrumente

VI (Virtual instrument) virtualni inštrument

IP (Internet protocol) internetni protokol

CSV (Comma-separated values) z vejico ločene vrednosti

ASCII (American standard code for information interchange) ameriška standardna koda za izmenjavo informacij

1. Uvod

V današnjih časih se vedno bolj in bolj uvaja avtomatizacija v vseh možnih pogledih. V industriji so to avtomatizirane linije (Slika 1.1), v administraciji so informacijski sistemi, v raziskavah in testiranjih so to avtomatizirani merilni sistemi. Povsod poizkušamo predati del našega dela računalniku, da se sami osredotočimo na pomembnejše stvari.

Z avtomatizacijo se izognemo nepotrebnemu delu. Za primer bi dal znanstvenike, od katerih jih zagotovo precej vpisuje rezultate meritev v računalnik ročno. Če povežemo merilni inštrument ali sistem povezanih merilnih inštrumentov z računalnikom, znanstveniku ni potrebno več zapravljati časa z vpisovanjem rezultatov meritev.

Slika 1.1: Avtomatizirana proizvodnja (preslikano iz [8])

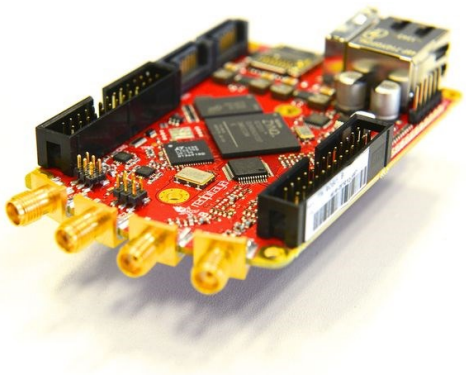
Za povezavo inštrumenta z računalnikom seveda najprej potrebujemo komunikacijsko povezavo. Ta je lahko žična ali brezžična. Le s komunikacijsko povezavo med inštrumentom in računalnikom še nimamo avtomatiziranega zapisovanja rezultatov v željeno tabelo na računalniku. Podatki meritev verjetno že prihajajo do računalnika, vendar se še ne vpišejo na željeno mesto. Za to potrebujemo programsko opremo, ki bo brala podatke na vhodu računalnika, jih po potrebi dešifrirala, preračunala ter vpisala na željeno mesto v namenski tabeli.

Orodij za izdelavo te programske opreme je ogromno. Izmed vseh bi izpostavil programsko orodje LabVIEW, katerega namen uporabe je tudi izdelava programov za kontrolo in branje podatkov iz inštrumentov ter obdelava teh podatkov. V diplomskem delu se bom osredotočil na izdelavo gonilnikov za programirljivi inštrument Red Pitaya v tem programskem orodju.

2. Inštrument in razvojna plošča Red Pitaya

V podjetju Red Pitaya d.d. sem 3 mesece opravljal praktično izobraževanje. V tem času sem sodeloval na projektu za razvoj LabVIEW-gonilnikov za Red Pitayo.

Izdelek (Slika 2.1) podjetja Red Pitaya d.d. z istim imenom je odprtokodna razvojna plošča, ki vsebuje dva hitra analogna vhoda, dva hitra analogna izhoda, set priključkov z dodatnimi počasnejšimi analognimi vhodi in izhodi, komunikacijami I2C, SPI in UART, vtičnike ethernet, USB ter dvakrat mikro USB (eden za napajanje in drugi za konzolo) ter režo za spominsko kartico. Kontrola celotne periferije se izvaja na Zinq sistemu integriranem v čip z dvojedrnim ARM procesorjem ter Xilinxovim FPGA blokom. FPGA je angleška kratica za Field-programmable gate array. Je integrirano vezje načrtovano tako, da ga uporabnik konfigurira po lastnih željah. Konfiguracija FPGA-ja je določena z opisnim jezikom za strojno opremo (ang. Hardware description language) Verilog ali VHDL. FPGA vsebuje zbirko programirljivih logičnih blokov ter hierarhijo nastavljivih povezav med njimi. Logični bloki so nastavljeni, da izvajajo kompleksne kombinacijske funkcije ali le enostavna logična vrata, kot so in, ali, ekskluzivni ali [2]. V večini FPGA vezij logični bloki vsebujejo tudi spominske elemente, kot je na primer bistabilni multivibrator (ang. Flip-flop) s kratico BMVB.



Slika 2.1: Merilni inštrument in razvojna plošča Red Pitaya (preslikano iz [1])

Program za kontrolo strojne opreme se lahko napiše v programskih jezikih C za kontrolo procesorja ali HDL za kontrolo FPGA čipa, poleg tega na plošči lahko poženemo tudi strežnik SCPI, ki se preko TCP protokola poveže na računalnik v istem omrežju in tako iz računalnika pošiljamo Ukaze SCPI na ploščo. V tem primeru lahko na našem računalniku napišemo program v kateremkoli programskem jeziku, ki podpira povezljivost s TCP protokolom in trenutno so že napisani programi v orodjih Matlab, Python, Scilab in LabVIEW.

3. Delovanje inštrumenta Red Pitaya ter podpora ukazov SCPI

Red Pitaya je v osnovi odprtokodna razvojna plošča, kjer lahko napišemo program za procesor in FPGA blok. S temi programi kontroliramo celotno periferijo na plošči. FPGA blok se večinoma uporablja za enostavna hitra opravila, kot je na primer branje vrednosti iz analogno digitalnega pretvornika. Procesor večinoma izvaja bolj kompleksna opravila.

Če se uporabnik ne želi ukvarjati z nizkonivojskim programiranjem Red Pitaye, uporabi ekosistem, na katerem so med drugim tudi že delujoči programi za zajem in generiranje signala. Poleg tega na ekosistemu deluje tudi aplikacijski programski vmesnik (ang. Aplication programming interface – API). Aplikacijski programski vmesnik je serija rutin, protokolov in orodij za izdelavo programskih aplikacij [3]. Ta na Red Pitayi vsebuje komande za konfiguracijo in delovanje priključkov in kanalov.

Na ekosistemu je tudi že naložena programska aplikacija SCPI server. Ta deluje kot strežnik, ki sprejema pošiljke. Ko dobi pošiljko, jo interpretira in glede na vsebino izvrši določeno rutino iz aplikacijskega programskega vmesnika.

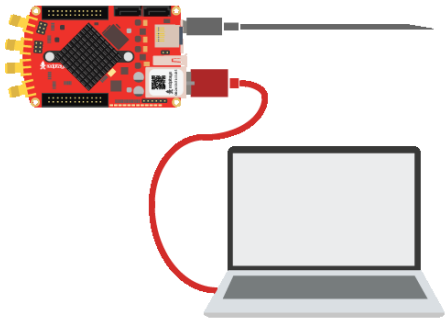
V primeru, ko uporabnik nima namena izvajati večjih konfiguracij na inštrumentu, lahko uporablja spletne aplikacije, ki so bazirane na Red Pitaya plošči in od tam spletni brskalnik prejema informacije za prikaz. Uporabnik lahko prenese spletne aplikacije iz Red Pitayine spletne trgovine. Aplikacije so lahko plačljive ali brezplačne. Nekatere aplikacije, ki so že dostopne na spletni trgovini, so:

- Osciloskop,
- Signalni generator,
- Spektralni analizator,
- LCR meter.

3.1. Povezljivost Red Pitaye

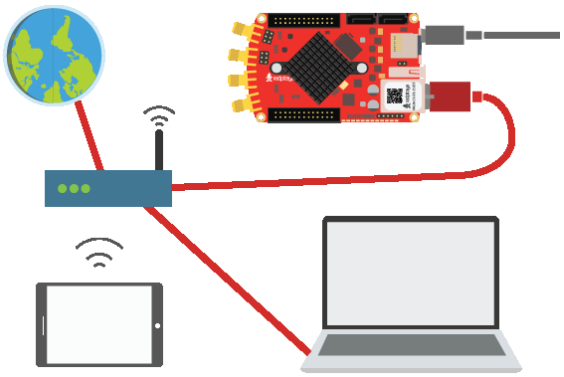
Red Pitaya se lahko poveže na računalnik preko:

- ethernet kabla direktno na računalnik in deluje plošča kot svoje omrežje (Slika 3.1),



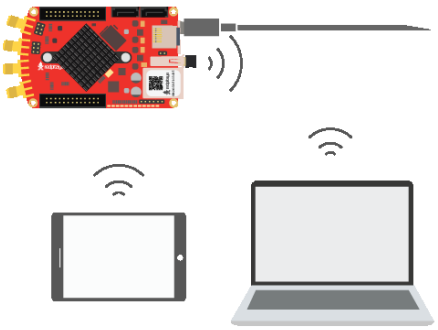
Slika 3.1: Direktna povezava preko ethernet kabla (preslikano iz [4])

- ethernet kabla, ki se priključi na usmerjevalnik in se tako poveže v omrežje le tega (Slika 3.2),



Slika 3.2: Povezava na usmerjevalnik (preslikano iz [5])

- WiFi kartice, ki jo vstavimo v USB vhod in deluje kot brezžična dostopna točka (Slika 3.3),



Slika 3.3: Povezava preko WiFi (preslikano iz [6])

- srednjega mikro USB vhoda, kjer se povežemo na konzolo.

Če uporabnik vzpostavi povezavo med računalnikom in ploščo preko omrežne povezave (ethernet kabel ali WiFi), lahko pošilja Ukaze SCPI preko TCP/IP protokola, vendar mora prej zagnati SCPI server na plošči.

3.2. SCPI

SCPI je kratica za Standard Commands for Programmable. Je standard za sintakso in komande za uporabo upravljanja z inštrumenti za meritve in testiranje. Omenjeni Standard definira sintakso, strukturo komand in format za pošiljanje podatkov, da je ta uporabljen v vseh inštrumentih. Struktura je sestavljena iz več generičnih komand (npr. CONFigure, MEASure,…), ki se uporabljajo v vseh inštrumentih. Standard tudi določa več različnih razredov inštrumentov (npr. DCSUPPLY, OSCilloscope, GENerator …), kateri kontrolirajo določeno funkcijo inštrumenta. Način povezave ni določen v standardu, zato se lahko kot fizični sloj uporablja vsaka komunikacija (npr. USB, Ethernet, GPIB …). Ukazi SCPI se pošiljajo v obliki ASCII nizov z velikimi črkami. Za ločevanje podskupin se uporablja dvopičje (primer: GEN:FREQ). Pri poizvedbah se na koncu komande pripne znak vprašaj (primer: GEN:FREQ?). Argumente določene komande se na koncu komande pripne za presledkom. Numerične vrednosti se pošlje kot niz ASCII znakov. Če ima komanda več argumentov, se ti ločijo z vejico (primer: ANALOG:PIN P10,1.8). Za koncentriranje komand, torej da se v enem nizu pošlje več komand, se med komandama doda podpičje ter za tem dvopičje (primer: GEN:FREQ 1000;:GEN:OFFSET 0.5) [9].

3.3. Podpora ukazov SCPI na Red Pitayi in njeno delovanje

Za kontrolo Red Pitaya plošče z ukazi SCPI je treba najprej zagnati SCPI server na plošči. SCPI server sprejema komande preko TCP/IP protokola ter jih ustrezno interpretira. Informacijo nato posreduje na API program, ki ima nadzor nad strojno opremo ter komunicira s FPGA čipom.

Omenjene komande na Red Pitaya plošči so razdeljene v 4 skupine, kjer vsak deluje posebej:

- LED diode in GPIO priključki (DIG)
- Počasni analogni vhodi in izhodi (ANALOG)
- Signalni generator (GEN)
- Zajem signala (ACQ)

3.3.1. LED diode in GPIO priključki

Ta razred (Tabela 1) zajema digitalne priključke (angl. GPIO – general purpose input/output) in LED diode. Skrajšano ime razreda, ki se uporablja v SCPI, je DIG. GPIO priključkom lahko najprej določimo smer na vhod ali izhod (privzeto so vsi nastavljeni na izhod). Če ima določen priključek smer nastavljeno na izhod, potem lahko pošljemo komando za nastavitev visokega ali nizkega stanja na omenjenem priključku. Isto komando lahko pošljemo tudi za vse LED diode na plošči. Na vse priključke in LED diode lahko pošljemo poizvedbo o stanju do plošče in ta nam vrne informacijo o stanju na željenem priključku.

3.3.2. Počasni analogni vhodi in izhodi

Ta razred (Tabela 2) zajema počasne analogne priključke. Red Pitaya ima štiri vhodne in štiri izhodne počasne analogne priključke. Ti lahko zajemajo do 100 tisoč vzorcev na sekundo. Vhodni priključki lahko sprejmejo napetosti od 0 V do 3.3 V, izhodni priključki od 0 V do 1.8 V. Na izhodne priključke lahko pošljemo komando za nastavitev napetosti. Na vse počasne analogne priključke lahko pošljemo poizvedbo o napetosti na posameznem priključku.

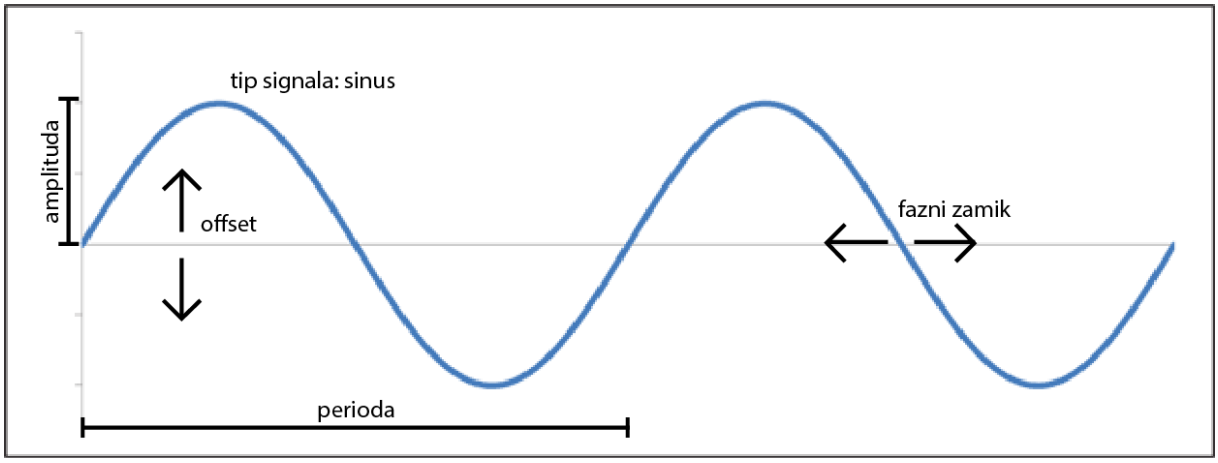
3.3.3. Signalni generator

Ta razred zajema dva hitra izhodna kanala, ki imata funkcijo signalnega generatorja. Vhodno območje je od –1 V do 1 V z največjo frekvenco signala pri 50 MHz.

Deluje tako, da je v FPGA bloku podatkovni medpomnilnik, iz katerega bere vrednosti in jih v obliki napetosti odda na izhodni kanal. Podatkovni medpomnilnik (ang. Data buffer) je pomnilniško področje, kjer so podatki shranjeni za krajši čas z namenom, da se prenesejo na drug kraj [11]. Za generiranje standardnega signala z določenimi parametri se podatki o signalu zapišejo v podatkovni medpomnilnik in nato iz njega ob ustreznem času premaknejo na izhodni kanal. Če želimo oddajati poljuben signal (»arbitrary waveform«), potem se v podatkovni medpomnilnik zapiše lista vrednosti, ki predstavlja signal.

Za zagon enega od kanalov signalnega generatorja se pošlje komando za odprtje kanala, vendar se v praksi najprej nastavi signal in šele nato odpre kanal. Z ukazi SCPI (Tabela 3) na Red Pitaya lahko signalu določimo (Slika 3.4):

- frekvenco, privzeto je ta nastavljena na 1000 Hz in se lahko nastavi na vrednosti od 0 Hz do 50 MHz,
- tip signala, privzeto je nastavljen na sinus, lahko se nastavi tudi na kvadratnega, trikotnega, pulzno širinska modulacija, ter žaga gor in žaga dol, poljuben signal (»arbitrary«),
- amplitudo, privzeto je nastavljena na 1 V in se lahko nastavi na vrednosti od 0 V do 1 V,
- enosmerno komponento signala (angleško »offset«), privzeto je nastavljen na 0 V in se lahko nastavi na vrednosti od –1 V do +1 V, tu je treba upoštevati, da vsota amplitude in absolutne vrednosti enosmerne komponente signala ne sme presegati 1V,
- fazni zamik med signaloma na obeh izhodnih kanalih, privzeto je nastavljen na 0 ° in se lahko nastavi na vrednosti od 0 ° do 360 °,
- čas trajanja visokega stanja, privzeto je nastavljen na 0 % in se lahko nastavi na vrednosti od 0 % do 100 %, ta vrednost se upošteva samo, ko imamo tip signala nastavljen na pulzno širinsko modulacijo.



Slika 3.4: Sinusni signal s prikazanimi osnovnimi parametri

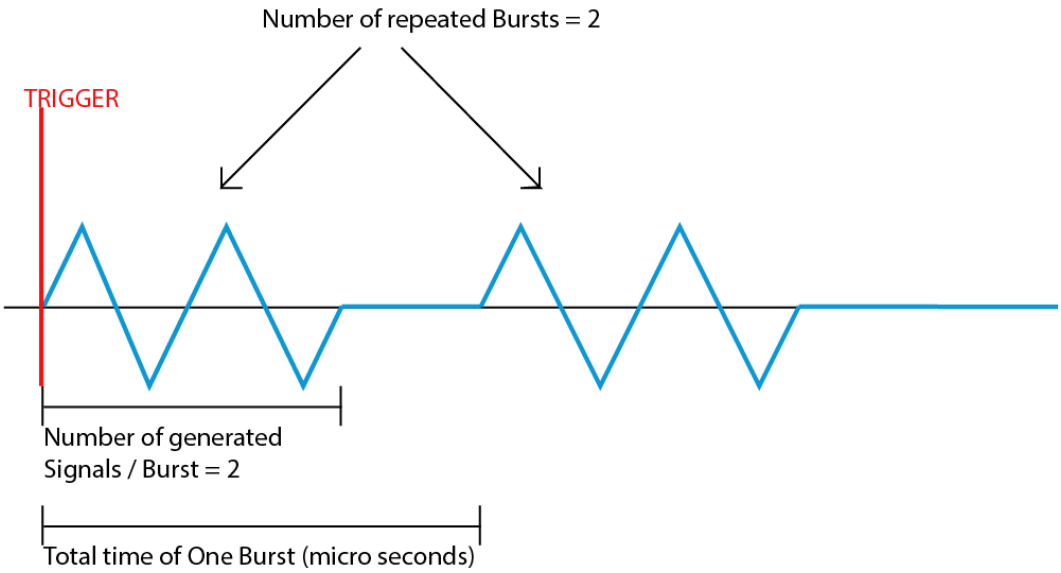
Signal, ki ga generiramo, je privzeto nastavljen na neprekinjenega (»continuous«), to pomeni, da se ves čas generira, torej, ko se ena perioda konča, se takoj začne naslednja. Lahko z ukazi SCPI ukažemo plošči za generiranje rafal signala (

Tabela 4). Ta je za razliko od neprekinjenega lahko prekinjen in ko odpremo kanal za rafal signala, se ta še ne bo generiral, saj mora najprej veljati pogoj za proženje. Red Pitaya ima več tipov za proženje rafal signala:

- zunanje proženje se sproži, ko na priključek, ki ima funkcijo zunanjega proženja, pripeljemo visoko ali nizko fronto, kar tudi prej določimo z ukazi SCPI,
- notranje proženje se sproži, ko na ploščo pošljemo komando za takojšnjo sprožitev določenega kanala ali obeh,
- ang. gated trigger se sproži, ko je na priključku, ki ima funkcijo zunanjega proženja, visoko stanje in rafal traja ves čas, ko je na omenjenem priključku visoko stanje.

Rafal signalu lahko nastavimo 3 dodatne parametre (Slika 3.5):

- število period v rafalu,
- število ponovljenih rafalov, to pomeni, da se lahko po proženju generira več zaporednih rafalov,
- celoten čas enega rafala.

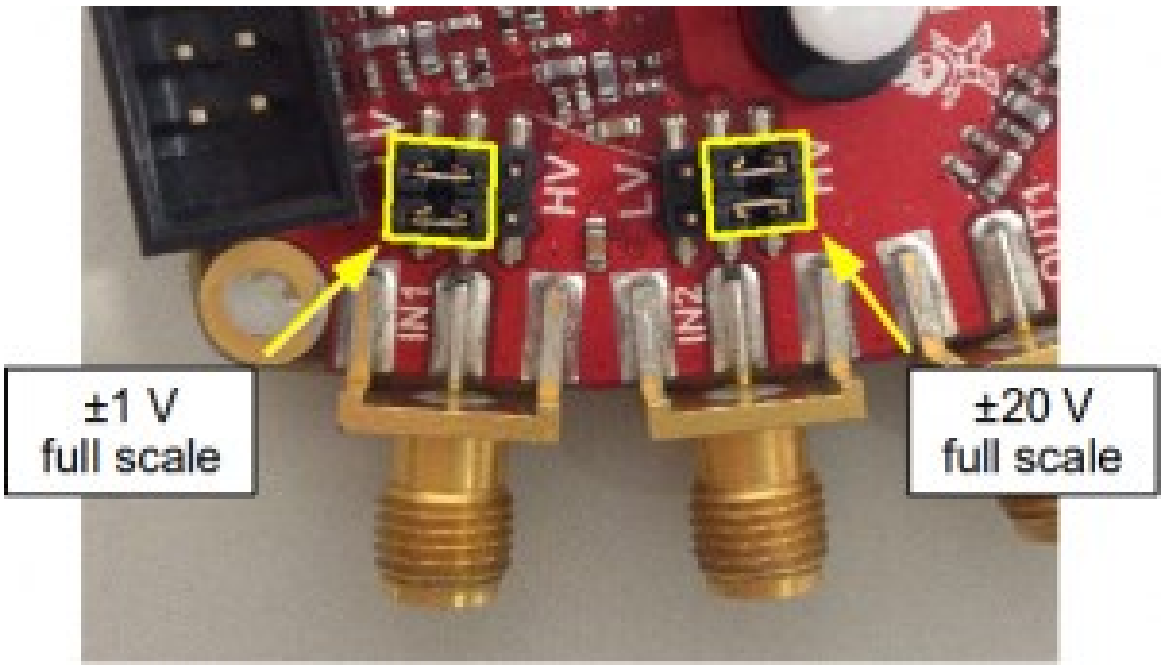


Slika 3.5: Prikaz generiranja rafal signala

Parametre signalnega generatorja, ki smo jih nastavili med delovanje Red Pitaye, lahko ponastavimo s komando »GEN:RST«. Seveda se parametri ponastavijo tudi od vsakem vnovičnem zagonu inštrumenta.

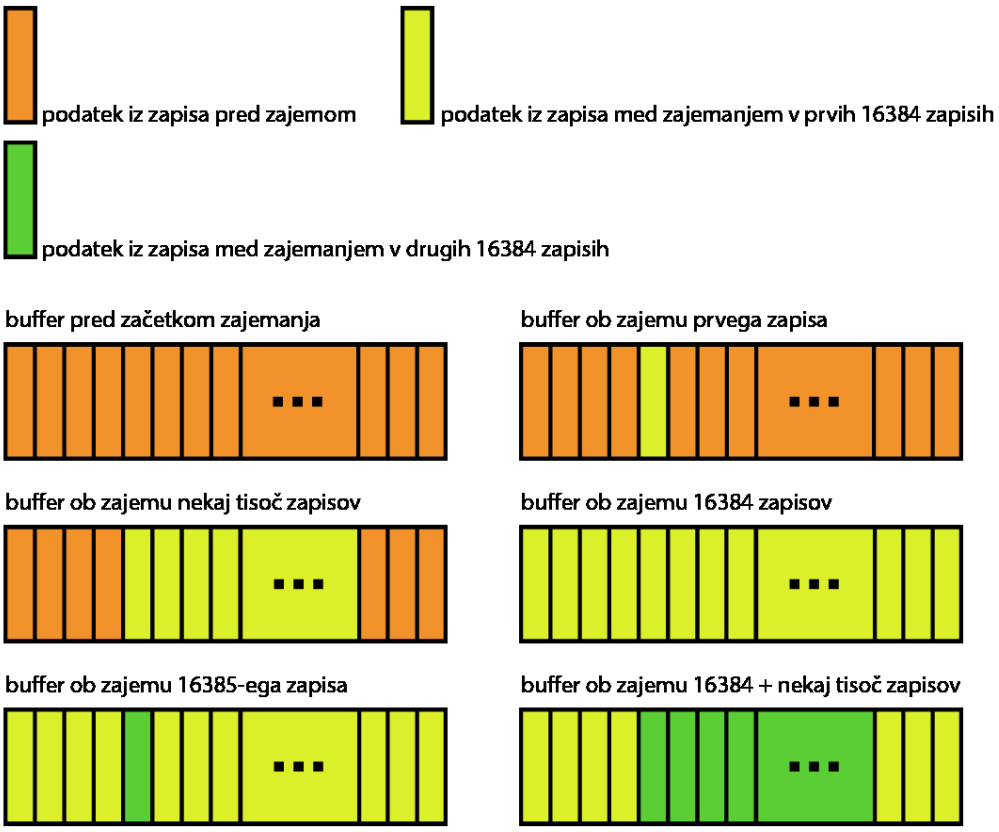
3.3.4. Zajem signala

Ta razred zajema dva hitra vhodna kanala, ki imata funkcijo signalnega generatorja. Vhodno območje je lahko od –1 V do 1 V ali od –20 V do 20 V s hitrostjo zajemanja do 125 vzorcev na sekundo. Možno je nastaviti željeno Vhodno območje izmed dveh možnosti s kratkospojnikom (Slika 3.6), ki se nahaja na plošči za željenim kanalom.



Slika 3.6: Dve možnosti vhodnega območja

Zajem signala deluje tako , da najprej zapiše vrednosti zajetih vzorcev iz vhoda v podatkovni medpomnilnik v FPGA čipu. Zapis vsakega vzorca v podatkovnem medpomnilniku se izvede tako, da se vrednost najprej prebere na vhodnem kanalu, kjer se preko analogno digitalnega pretvornika pretvori v digitalen signal, ta se nato zapiše v podatkovnem medpomnilniku na mesto, kjer je najstarejši podatek. Ko se podatkovni medpomnilnik napolni z novim signalom, se zajem zaključi. Po zaključenem zajemu se lahko pošlje vrednosti signala na računalnik. Sedaj podatkovni medpomnilnik shrani do 16384 vzorcev signala, vendar se istočasno že razvija zapisovanje vzorcev na pomnilnik, kar bo povečalo kapaciteto zapisa na nekaj milijonov vzorcev. Primer postopka zapisovanja vzorcev v podatkovni medpomnilnik je prikazan na Slika 3.7.



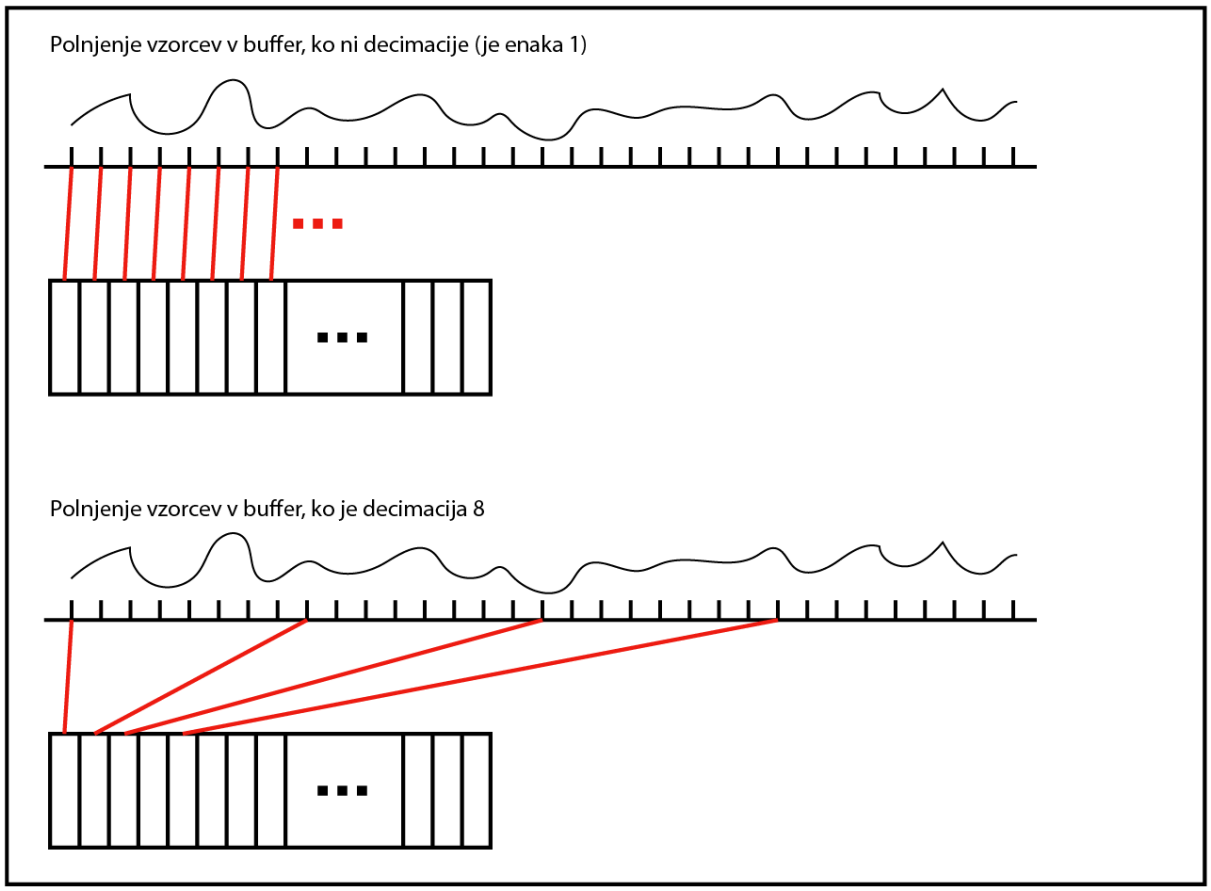
Slika 3.7: Postopek zapisovanja vrednosti v podatkovni medpomnilnik

Za zajem signala na enemu od kanalov se pošlje komando (Tabela 5) za začetek zajema na kanalu in za tem še komando za določitev proženja, vendar se v praksi pred tem nastavi parametre za zajem in proženje, da uporabnik dobi željen zajeti signal. Če za zajem signala pošljemo le komando za začetek zajema, ne tudi komande za določitev proženja, potem se zajemanje ne bo ustavilo, dokler ne pošljemo komande za ustavitev ali ponastavitev zajema. V času neprekinjenega zajemanja se novi vzorci ves čas zapisujejo v podatkovni medpomnilnik na mesta najstarejših vzorcev, torej se ob vsakem novem zapisu izbriše najstarejši vzorec in vpiše najnovejši. To se lahko zgodi tudi, ko pošljemo komando za začetek zajema in komando za določitev proženja, če signal, ki ga proženje pregleduje, nikoli ne zadosti vseh pogojev proženja. Torej z ukazom za začetek zajemanja se vrednosti začnejo zapisovati v podatkovni medpomnilnik, po določitvi proženja se bo zapisovanje ustavilo najhitreje ob času, ko bodo pogoji za proženje prvič zadoščeni. Tu je treba še upoštevati zakasnitev proženja.

Zajemanju lahko nastavimo naslednje parametre:

- decimacija,
- hitrost zajemana,
- povprečenje.

Decimacija in hitrost zajemanja določata isto spremenljivko. Ta spremenljivka določa hitrost zajemanja. Decimacija določa, na koliko zajetih vzorcev se zapiše vzorec v podatkovni medpomnilnik v primeru, ko nimamo vključenega povprečenja, torej z večjo decimacijo naredimo manjšo hitrost zajemanja, to je tudi prikazano na sliki 5.10. Če imamo vključeno povprečenje, decimacija deluje tako, da zajame določeno število vzorcev, izračuna povprečje in zapiše to vrednost v podatkovni medpomnilnik. Ukazi SCPI na plošči sedaj podpirajo le decimacije 1, 8, 64, 1024, 8192, 65536. Ker Red Pitaya deluje v ozadju na principu decimacije, je komanda za določitev hitrosti zajemanja uporabljena le za lažje upravljanje parametra. Primer zapisovanja v podatkovni medpomnilnik pri različnih decimacijah je prikazan na Slika 3.8



Slika 3.8: Zapisovanje vzorcev v podatkovni medpomnilnik ob različnih decimacijah

Red Pitaya ima v ukazih SCPI podporo za naslednja proženja:

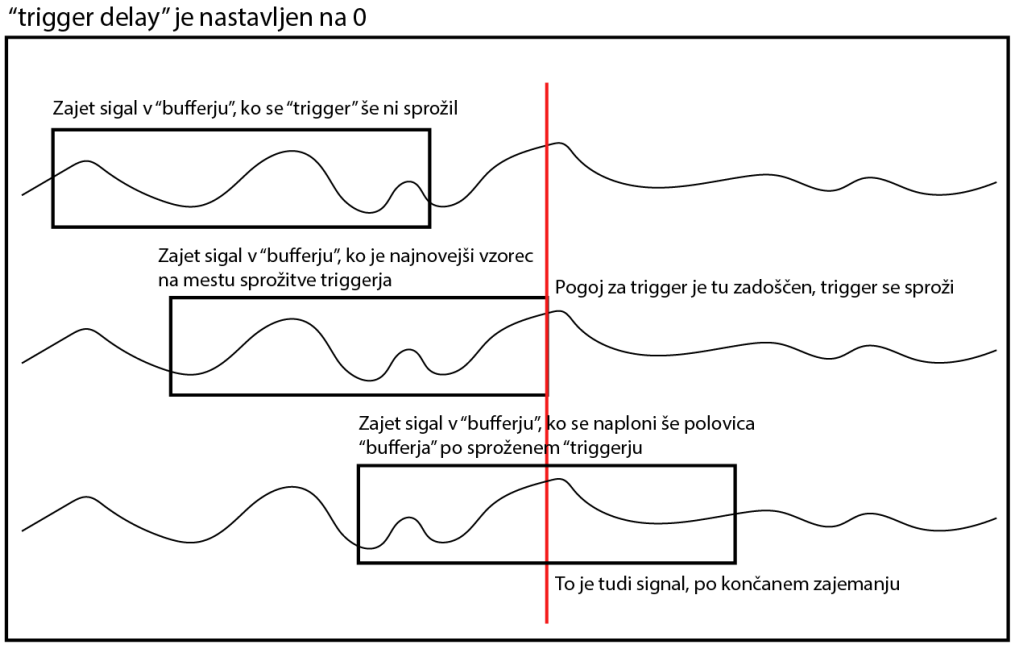
- takojšnji, kar sproži zajem, takoj ko je komanda prebrana na plošči,
- na vhodnih kanalih, kjer je treba še določiti smer prehoda vrednosti (pozitivni ali negativni prehod),
- zunanje proženje, ki je na priključku in je treba določiti smer prehoda signala, ki proži (pozitivni ali negativni prehod stanja),
- proženje z generatorjem poljubnega signala,
- onemogočen, kar je nastavljeno kot privzeto, ter ob tej nastavitvi se zajem nikoli ne zaključi, ker proženje ni nastavljeno.

Ukazi SCPI za namen proženja zajema so prikazani v Tabela 6. Da program, ki kontrolira ploščo, ve, kdaj se je proženje izvedlo in takrat lahko prebere podatke iz podatkovnega medpomnilnika, mora pošiljati poizvedbe na ploščo o stanju proženja. Ko se proženje izvede, preide v stanje onemogočen. Omenjeno preverjanje stanja proženja se po navadi izvede z »while« zanko.

Poleg tipa proženja lahko proženju določimo še naslednja parametra:

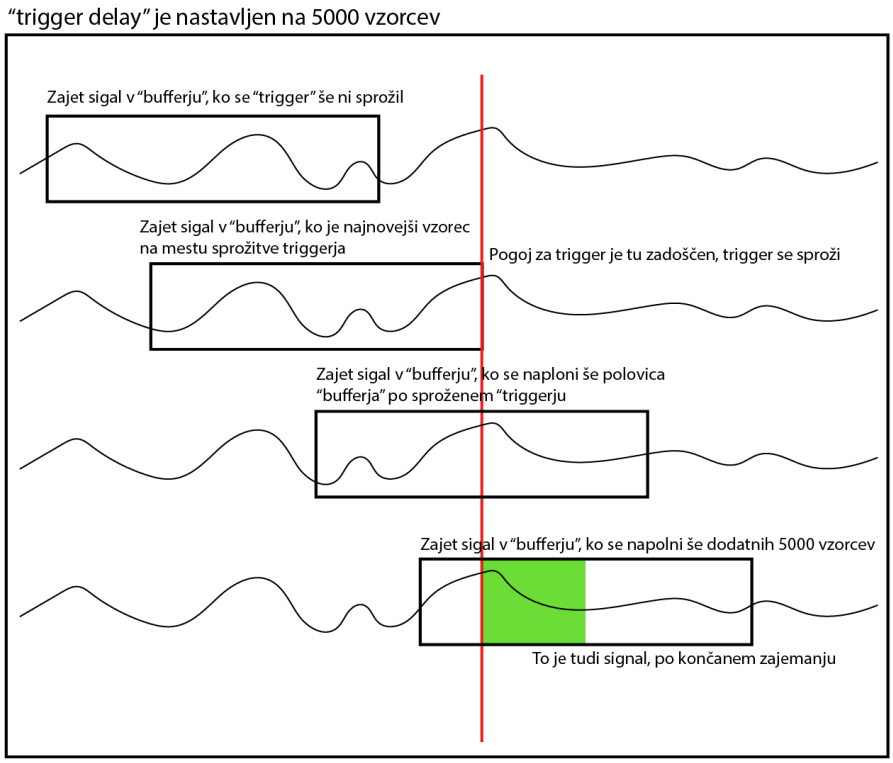
- zakasnitev proženja ali zakasnitev,
- nivo proženja ali nivo.

Zakasnitev proženja lahko določimo v številu zajetih vzorcev ali v nanosekundah in je privzeto nastavljena na 0. Zajemanje deluje tako, da ob zakasnitvi proženja enaki 0 in ob sprožitvi zajame še toliko vzorcev kot je polovica števila mest v podatkovnem medpomnilniku. Zato je ob zakasnitvi proženja enaki 0 dogodek, ko se proženje izvede, vedno na sredini zajetega signala. Če zakasnitev proženja povečamo ali zmanjšamo, se bo dogodek sprožitve premaknil tudi na zajetem signalu. Ob tem je treba upoštevati, da je lahko zakasnitev najmanj negativna vrednost polovice velikosti podatkovnega medpomnilnika. Na Slika 3.9 je prikazano polnjenje podatkovnega medpomnilnika, ko je zakasnitev nastavljena na 0.



Slika 3.9: Prikaz zapisovanja podatkovnega medpomnilnika, ko je zakasnitev proženja enaka 0

Na Slika 3.10 je prikazano zapisovanje v podatkovni medpomnilnik s signalom, ko imamo zakasnitev proženja nastavljeno na več kot 0 oziroma v primeru na omenjeni sliki približno + 5000 vzorcev.



Slika 3.10: Prikaz polnjenja podatkovnega medpomnilnika, ko je zakasnitev proženja približno + 5000 vzorcev

Zaradi kratkospojnikov za analognima vhodoma na plošči je možno tudi nastavljanje vhodnega območja na od –1 V do 1 V ter od –20 V do 20 V. Ker procesor ne dobi informacije o poziciji kratkospojnikov, moramo ročno nastaviti vhodno območje. Privzeto je zajemanje nastavljeno na –1 V do 1 V. Če to želimo spreminjati, moramo poslati komando za drugo vhodno območje.

Podatkovni medpomnilnik za zapisovanje vzorcev v pomnilniške celice uporablja kazalec. Kazalec se ob vsakem novem zapisu premakne za eno mesto višje. Ko je na zadnjem mestu, se premakne na začetek. Tudi ko se zapiše zadnji vzorec zajema v podatkovni medpomnilnik nekje na sredini pomnilniških celic, ta ne bo začel naslednji zapis na prvem naslovu v podatkovnem medpomnilniku, ampak na naslednjem naslovu, ki je za naslovom z zadnjim zapisom. Ukazi SCPI (Tabela 7) omogočajo poizvedbo na trenutno lokacijo kazalce, ter na lokacijo, kjer se je zgodilo proženje.

Preden začnemo z branjem zajetega signala iz podatkovnega medpomnilnika, moramo najprej določiti, s kakšno enoto želimo prejeti podatke (surovo ali v voltih) ter v kakšen formatu bo naš program na računalniku prebral podatke. Za oba omenjena parametra so podprti Ukazi SCPI na Red Pitayi.

Ko sta ta dva parametra nastavljena po naših željah, lahko preberemo podatke iz podatkovnega medpomnilnika na našem računalniku, za katere pošljemo poizvedbe na ploščo. Podprti Ukazi SCPI so navedeni v Tabela 8.

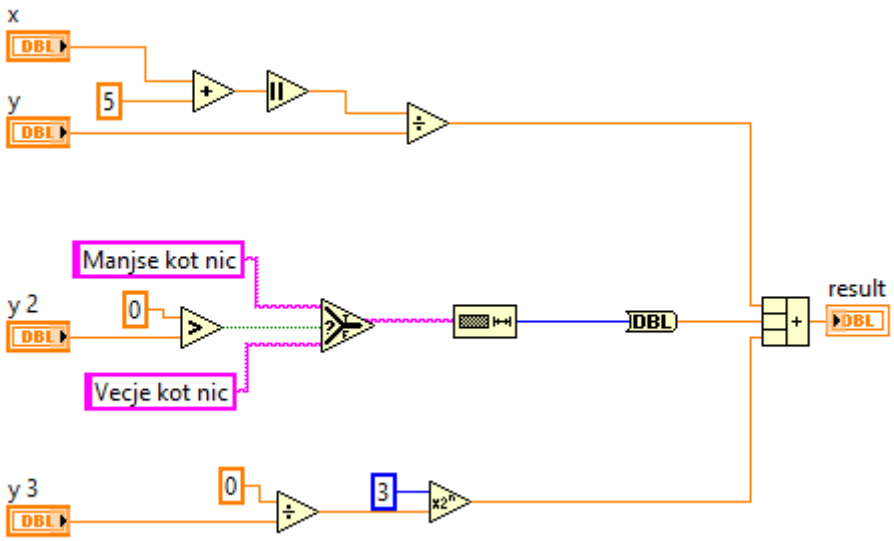
4. LabVIEW

LabVIEW (Slika 4.1) je okrajšava za inženirsko delovno okolje laboratorijskih virtualnih inštrumentov (ang. Laboratory Virtual Instrument Engineering Workbench). Razvilo ga je podjetje National instruments, ki je vodilno na področju meritev in testiranj. LabVIEW je na trgu že od leta 1986.

Slika 4.1: Logo programa LabVIEW (preslikano iz [13])

Je grafično programsko okolje z grafičnim programskim jezikom imenovanim »G«. LabVIEW se uporablja za zajem podatkov, kontrolo inštrumentov, industrijsko avtomatizacijo itd. Ena glavnih prednosti grafičnega programiranja je lažje razumevanje blokovnega diagrama, čemur bi se pri tekstovnem programiranju reklo koda. Poleg tega uporabnik ne potrebuje predznanja o sintaksi, saj ima celotno knjižnico ukazov ob pritisku na desni klik.

Zaradi izvajanja ukazov v blokovnem diagramu tudi zaporedje izvajanja teh ukazov deluje drugače kot pri tekstovnem programiranju, kjer se ukazi izvajajo od vrha navzdol, torej zaporedno, kot so ukazi zapisani. Izjeme so le funkcije in zanke. Pri grafičnem programiranju zaporedje izvajanja ukazov ni odvisno od pozicije blokov, temveč od povezav med njimi. Zgradba blokovnega diagrama je sestavljena iz povezav in vozlišč. Vozlišče se izvrši, ko prejme vrednosti na vseh povezavah. Vozlišče je v analogiji z izjavami, operatorji, funkcijami in podVI (ang. subVI). Zaradi tega se lahko več različnih vej blokovnega diagrama izvaja vzporedno. Tu bi le opozoril, da se dva ukaza dejansko ne moreta izvajati vzporedno, saj se ukazi s stališča jedra procesorja vedno izvajajo zaporedno [12]. Primer blokovnega diagrama s tremi vejami je prikazan na spodnji sliki (Slika 4.2).

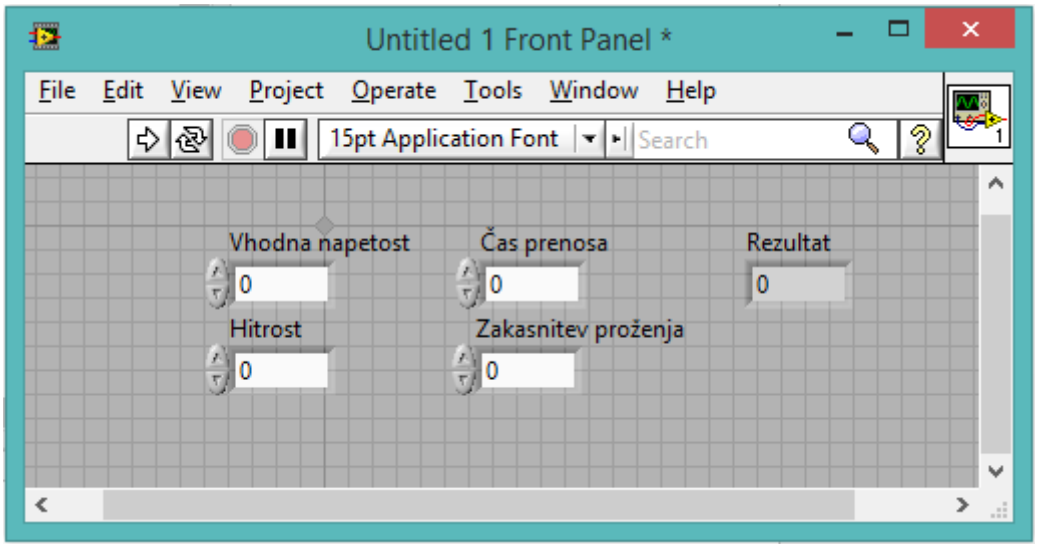


Slika 4.2: Prikaz blokovnega diagrama s tremi vejami.

Orodja za razhroščevanje v LabVIEW-u se razlikujejo od metod pri tekstovnem programiranju. Ene najbolj popularnih orodij za razhroščevanje je poudarjanje izvajanja (ang. Highlight execution), s čimer se animira potek izvajanja programa v blokovnem diagramu. To je prikazano tako, da je razvidno, kje vsak trenutek potuje vrednost po povezavah. Poleg tega so na voljo tudi klasične metode po korakih (ang. Step in, out, over), ustavitvena točka (ang. Breakpoint) ter prikaz vrednosti na povezavi (ang. Probe).

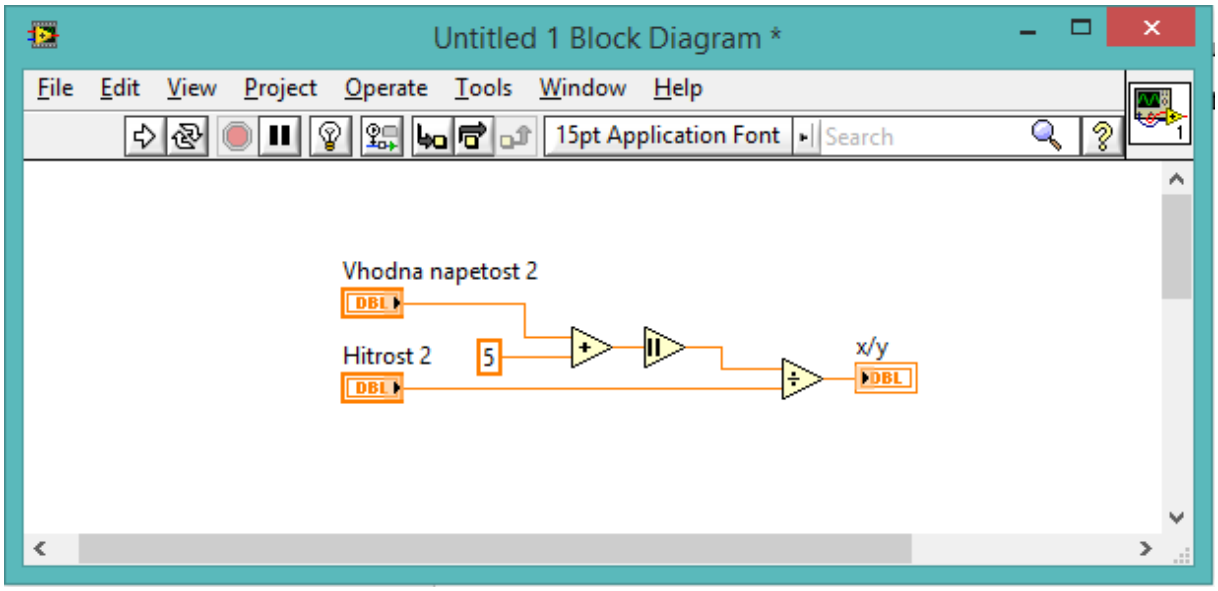
Osnovni gradnik za programe, napisane v LabVIEW-u, se imenuje virtualni inštrument s kratico VI. Vsak VI, ki ga odpremo, odpre dve okni, eno je čelna plošča (ang. Front panel) in drugo je blokovni diagram (ang. Block diagram). Čelna plošča (Slika 4.3) je namenjena prikazovanju in spreminjanju vrednosti. Ima nalogo uporabniškega vmesnika za isti VI. Grafični element za spreminjanje vrednosti se imenuje kontrolno polje (ang. Control). Grafični element za prikazovanje vrednosti se imenuje indikator (ang. Indicator). Na čelni plošči se lahko prikazuje in spreminja vrednosti različnih tipov, kot so na primer: niz znakov, celo število, vrednost s plavajočo vejico, boolean … Vse elemente kontrolnega polja dobimo z desnim klikom v paleti kontrol (ang. Control palette). Namen kontrolnega polja in indikatorja je tudi prenos vrednosti med VI-ji. Za to je najprej potrebno definirati vezne elemente (ang. Connector) VI-ja. Ko je to definirano, lahko uporabimo ta VI v drugem VI-ju. S stališča drugega VI-ja, ki vsebuje prvi VI, je ta prvi VI zanj podVI (ang. subVI). Poleg blokovnega diagrama in čelne plošče vsak VI vsebuje še priključitveno ploščo (ang. Connector panel), ki je uporabljena za prikazovanje VI-ja v drugih VI-jih, kjer lahko preko

priključitvene plošče vnesemo vrednosti v podVI in dobimo vrednosti iz njega. Katere vrednosti lahko vnesemo noter, ter katere lahko dobimo ven, določimo v priključitveni plošči.



Slika 4.3: Čelna plošča

Blokovni diagram (Slika 4.4) je namenjen kontroliranju čelne plošče. Vsebuje kodo, ki je grafično prikazana s povezavami med vozlišči. Vsebuje terminale, subVI-je, funkcije, konstante, strukture in povezave, ki prenašajo podatke med ostalimi objekti v blokovnem diagramu. Vse elemente blokovnega diagrama dobimo z desnim klikom v paleti funkcij (ang. Functions palette). Kontrolna polja in indikatorji so prikazani v blokovnem diagramu kot terminali.



Slika 4.4: Okno blokovnega diagrama

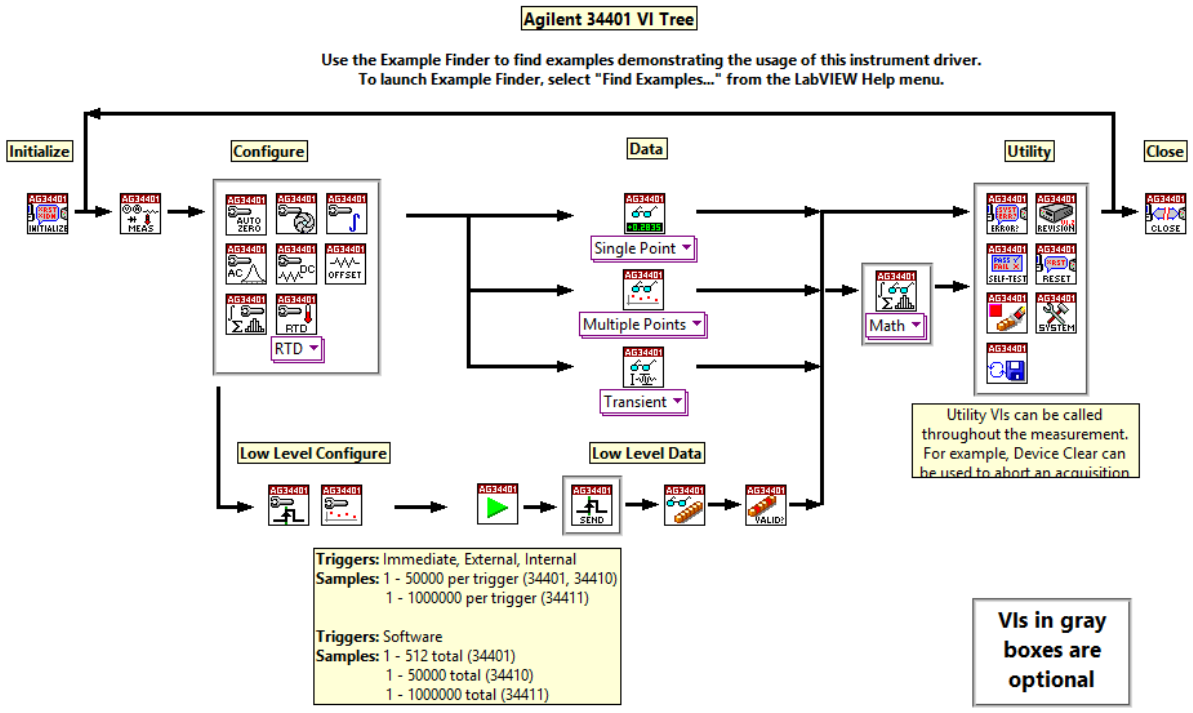
5. LabVIEW-gonilniki za inštrumente

LabVIEW-»vklopi in zaženi« gonilnik za inštrumente (ang. LabVIEW plug and play instrument driver) je serija VI-jev, ki kontrolirajo programirljiv inštrument. Vsak VI se odzove na določeno funkcijo inštrumenta, kot so proženje, nastavljanje ter branje podatkov iz inštrumenta. Gonilniki za inštrumente pomagajo uporabniku za lažji začetek uporabe inštrumenta z računalnikom in jim prihrani čas razvoja in denar, saj uporabniku ni potrebno znati protokola, po katerem inštrument komunicira z računalnikom. Z odprtokodnimi dobro dokumentiranimi gonilniki lahko uporabnik prilagodi gonilnike svojim potrebam in izboljša delovanje [14].

Z uporabo standardne arhitekture (Slika 5.1) za vse LabVIEW-gonilnike za inštrumente pridobimo naslednje koristi [14]:

- izboljša doslednost gonilnikov v korist končnemu uporabniku,
- izboljša kakovost gonilnikov,
- izboljšuje razvoj gonilnikov v korist razvijalcem.

Uporabnik inštrumenta, ki ima podporo LabVIEW-gonilnikov, lahko te najde na spletni strani LabVIEW Instrument Driver Network.

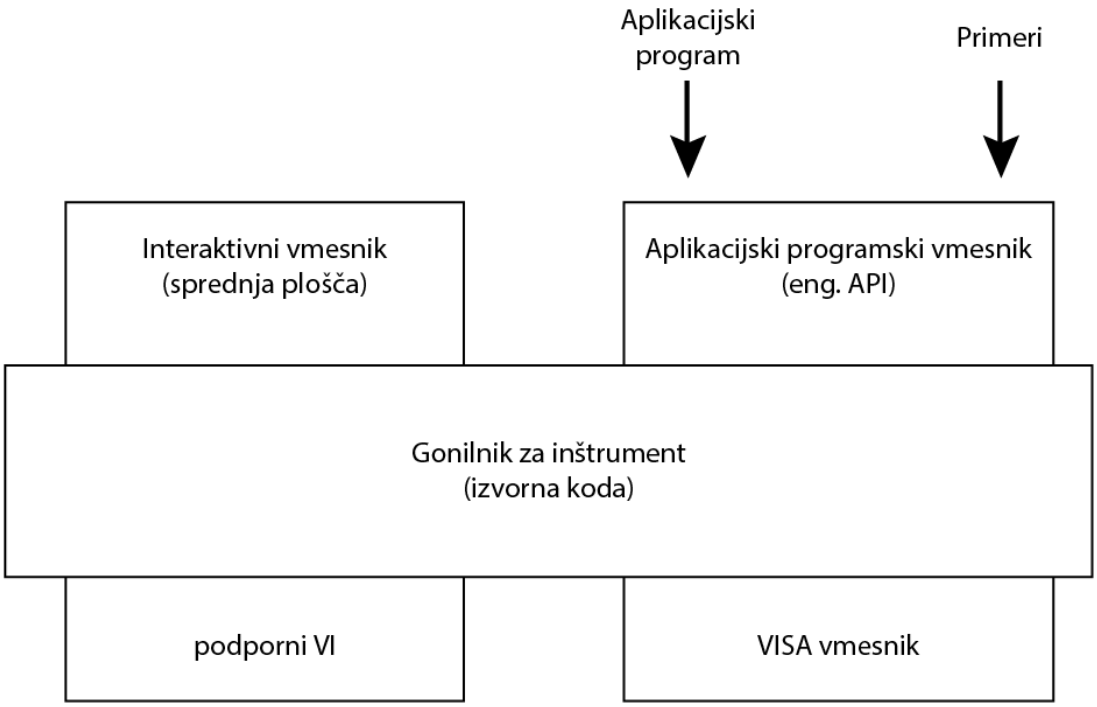


Slika 5.1: Primer strukture LabVIEW-gonilnikov za Agilent 34401

5.1. Model LabVIEW-gonilnikov za inštrumente

Ogromno programirljivih inštrumentov ima veliko število funkcij in načinov delovanja. Zato je pomembno, da se priskrbi usklajen model, ki pomaga razvijalcem gonilnikovin uporabnikom gonilnikov. LabVIEW-gonilniki za inštrumente vsebujejo smernice za notranjo in zunanjo strukturo.

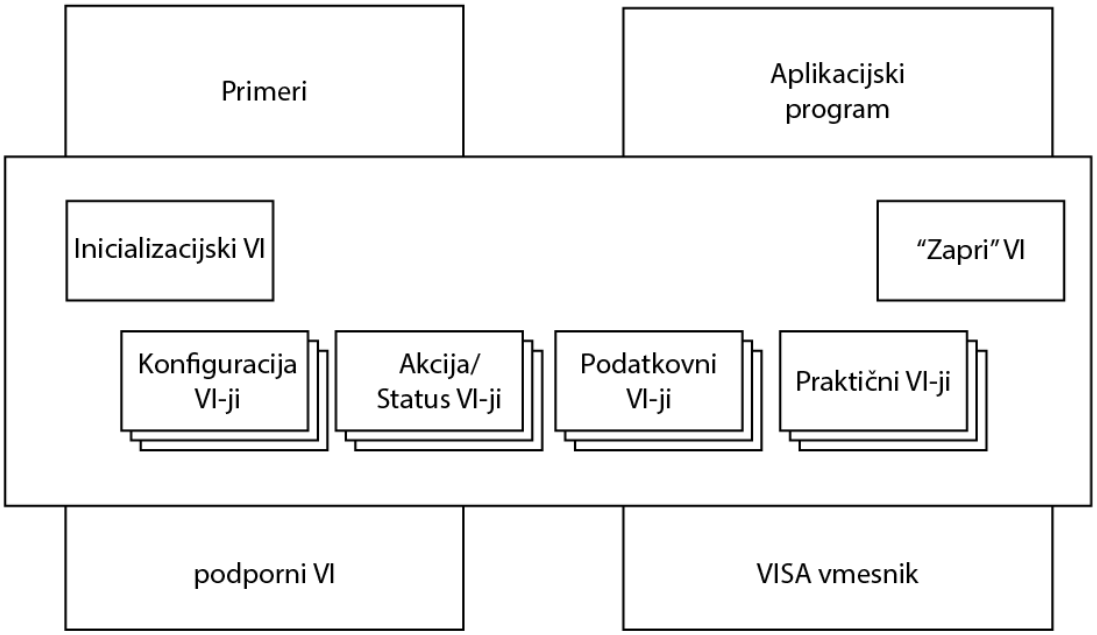
Zunanja struktura (Slika 5.2) vsebuje serijo gonilniških VI-jev, katere uporabnik kliče v višjem nivoju aplikacije. Za lažje in hitrejše razumevanje VI-jev lahko uporabnik preizkusi delovanje določenega VI-ja na njegovi čelni plošči, kjer lahko spremeni vrednosti in preveri, kaj se je ob tem spremenilo v delovanju inštrumenta. Programska arhitektura virtualnega instrumenta (ang. Virtual instrument software architecture – VISA) je serija LabVIEW-funkcij, katere gonilnik uporabi za komunikacijo z inštrumentom. Podporni VI-ji so vsi tisti VI-ji, do katerih naj uporabnik ne bi dostopal, saj je njihov namen izvajati metode, ki jih potrebuje več javnih VI-jev. Tako na primer podporni VI Post.vi uporabljajo vsi javni VI-ji, ki pošljejo komando na inštrument.



Slika 5.2: Zunanja struktura gonilnikov

Notranja struktura (Slika 5.3) definira organizacijo uporabniku dostopnih VI-jev za gonilnik, kateri so razvrščeni v modularni hierarhiji glede na funkcionalnost inštrumenta. Končnim

uporabnikom je zelo pomembno, da inštrument, ki ga kontrolirajo z LabVIEW-gonilnikom, kontrolirajo pravilno v svojih aplikacijah. Zato je zadolžena notranja struktura, ki razvrsti VI-je v skupine, glede na čas izvršitve VI-ja. Tako je VI za inicializacijo na začetku strukture, za njim so VI-ji za konfiguracijo in tako naprej do zadnjega »Zapri« VI, kateri preneha komunikacijo z inštrumentom.

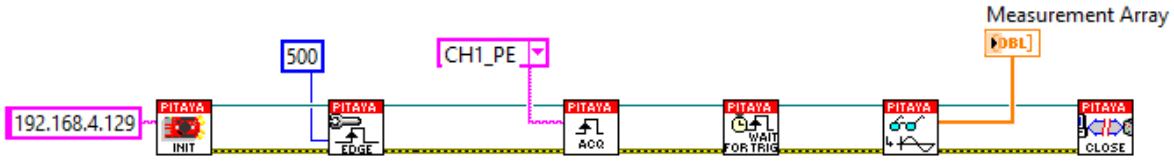


Slika 5.3: Notranja struktura gonilnikov

Inicializacijski VI (ang. Initialize VI) bi morali vsebovati vsi gonilniki za inštrumente, saj ta VI vzpostavi komunikacijo z inštrumentom in zato je tudi VI, ki je vedno na začetku blokovnega diagrama. Konfiguracijski VI-ji (ang. Configure VIs) so namenjeni za nastavitve parametrov na inštrumentu in te se po navadi nastavi, preden se zažene glavna operacija inštrumenta (na primer zajem signala pri osciloskopu). Akcijski VI-ji (ang. Action VIs) pošljejo inštrumentu, da glede na nastavitve izvede operacijo testiranja in merjenja. Pri tem se nobena nastavitev ne spremeni in v tem se akcijski VI-ji razlikujejo od konfiguracijskih. Statusni VI-ji (ang. Status VIs) so namenjeni za pridobivanje informacij o statusu določenega parametra. Podatkovni VI-ji (ang. Data VIs) so namenjeni prenosu večjih količin podatkov, kot je na primer zajet signal, ki vsebuje 8000 številskih vrednosti. Servisni VI-ji (ang. Utility VIs) no namenjeni izvajanju pomožnih operacij, kot je na primer ponastavitev. »Zapri« VI konča komunikacijo z inštrumentom.

Če na primer (blokovni diagram - Slika 5.4) želimo zajeti signal, ko gre signal mimo + 500 mV, uporabimo VI-je po naslednje vrstnem redu:

1. inicializacijski VI za vzpostavitev povezave,
2. konfiguracijska VI ja za nastavitev nivoja proženja zajema na + 500 mV,
3. akcijski VI za aktivacijo zajemanja,
4. statusni VI v zanki za preverjanje, če se je proženje že sprožilo,
5. podatkovni VI za prenos zajetih podatkov iz inštrumenta na računalnik,
6. »zapri« VI za konec komunikacije.



Slika 5.4: Blokovni diagram aplikacije, ki na Red Pitayi zajame signal pri proženju na 500 mV

Dodatni VI-ji, ki jih paket gonilnika vsebuje, so VI-ji primerov (ang. exampleVIs) in drevo VI-jev (ang. VI Tree).

VI-ji primerov prikazujejo primere kombinacij gonilniških VI-jev v blokovnem diagramu. Ti primeri so večinoma enostavnejše aplikacije inštrumenta na področju merjenj in testiranja. Prikažejo funkcionalnost inštrumenta z nalogami, kot so merjenje, proženje, generiranje itd. Uporabljeni so tudi za potrditev komunikacije med računalnikom in inštrumentom. Priporočeno je, da so vsi VI-ji primerov vključeni v iskalnik primerov (ang. Example finder), ki je zbirka vseh primerov VI-jev z vključenimi privzetimi VI-ji za učenje programa LabVIEW ter vsemi gonilniškimi VI-ji, kateri so naloženi.

Drevo VI-jev (Slika 5.1) je nedelujoči VI, ki v blokovni shemi vključuje vse gonilniške VI-je. Namenjen je prikazu celotne hierarhije teh VI-jev. Hierarhija je prikazana po načelu notranje strukture. Vseeno uporabniki večinoma dostopajo do gonilniških VI-jev preko palete funkcij, ker je to hitreje.

6. Razvoj LabVIEW-gonilnika za inštrument Red Pitaya

V tem poglavju je opisan postopek razvoja LabVIEW-gonilnika za inštrument Red Pitaya. Idealen LabVIEW-gonilnik omogoča končnemu uporabniku, da upravlja z vsemi funkcionalnostmi inštrumenta. Pri razvoju gonilnikov za Red Pitayo se je poizkušalo čim bolj približati tem merilom.

Postopek razvoja gonilnikov poteka v treh korakih. V prvem koraku se načrtuje struktura gonilnikov, kar se po navadi zapiše na papir in služi kot osnutek. V drugem koraku se ustvari LabVIEW-projekt z uporabo čarovnika za izdelavo projekta gonilnikov za inštrumente. V tretjem koraku se implementira v osnutku zastavljene VI-je in VI-je primerov.

6.1. Prvi korak: načrtovanje strukture gonilnika

S povečanjem števila komand se poveča tudi število gonilniških VI-jev. Pri tem seveda ni dobro, če je v celotnem projektu gonilnika le eden VI, ki vsebuje funkcionalnost vseh komand. Ter seveda obratno, da vsak VI vsebuje funkcionalnost ene komande. S tem bi se moral uporabnik vseeno naučiti, katere komande delujejo skupaj in seveda bi porabil več časa za postavljanje več VI-jev, ko bi lahko bilo več funkcionalnosti združenih v enem VI-ju.

Preden razvijalec začne z načrtovanjem strukture, mora poznati delovanje inštrumenta in vsakega ukaza SCPI, kateri je podprt na inštrumentu. Da se s tem prej spozna, je priporočeno, da:

1. prebere uporabniški priročnik delovanja inštrumenta; to naj bi opravil, preden sploh zažene inštrument,
2. preizkusi inštrument na delujočem primeru,
3. v podrobnost preuči programirljive komande,
4. pogleda, če so gonilniki že narejeni za kakšen inštrument s podobnimi funkcionalnostmi,
5. poišče komande, ki se izvajajo skupaj za isto funkcionalnost.

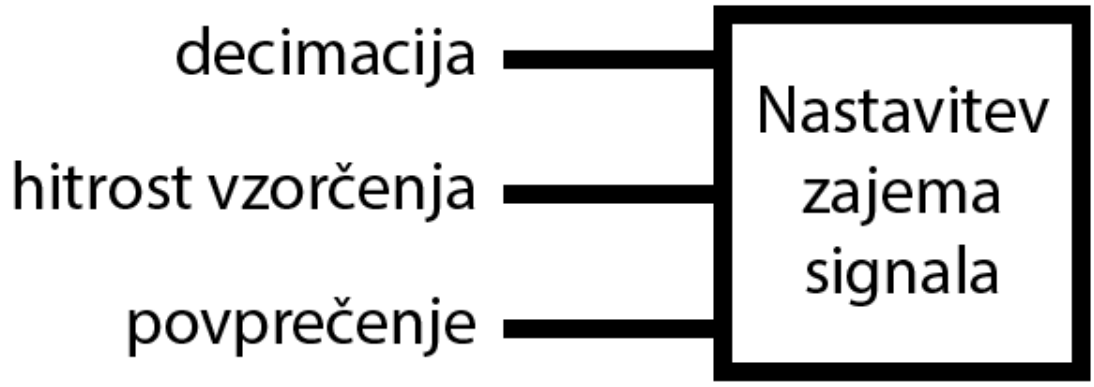
API VI-ji so lahko takšni, ki se uporabljajo na vseh inštrumentih (inicializacija), ali so posebej narejeni glede na funkcionalnost inštrumenta. Prve uporabniku ni potrebno načrtovati, ker imajo za vsak inštrument enako strukturo, druge uporabnik načrtuje glede na funkcionalnost inštrumenta.

Sedaj se razvrsti Ukaze SCPI najprej v skupine inicializacija, konfiguracija, akcija/status, podatkovni, servisni ter VI za zaprtje. Za primer bom predstavil skupino konfiguracija. V to skupino se po sekcijah vključi naslednje skupine komand:

- Iz sekcije zajem signala so to komande za: nastavitev zajema signala, nastavitev proženja zajema, nastavitev vhodnega območja, nastavitev branja podatkov.
- Iz sekcije generiranje signala so to komande za: nastavitev generiranega signala, nastavitev generiranega poljubnega signala, nastavitev generiranega rafala.
- Iz sekcije GPIO priključkov je to komanda za nastavitev smeri.

Kot je razvidno iz primera, so v konfiguraciji vse komande za nastavitev nečesa. Te komande se tudi v blokovnem diagramu skoraj vedno uporabijo na začetku, takoj za inicializacijo. Pri sestavljanju teh skupin komand bi le opozoril, da ni vedno najbolje dati vse komande s podobnim delovanjem v isto skupino. Tako bi na primer moral po pravilih komando za nastavitev vhodnega območja dodati v skupino komand za nastavitev zajema signala, saj ta določa vhodno območje signala. Vendar se v praksi ta komanda uporablja zelo malokrat, saj je potrebno za spremembo vhodnega območja spremeniti tudi pozicijo kratkospojnika. Zato je priporočeno, da se razvijalec gonilnikov posvetuje tudi z izkušenimi uporabniki inštrumenta.

Te skupine komand, ki jih sestavimo, nato združimo v VI-je. Tako na primer skupina komand za nastavitev zajema signala vsebuje komande za nastavitev povprečenja, decimacije in hitrosti vzorčenja. Za lažjo predstavo je tudi priporočeno risanje osnutkov z grafičnimi bloki z vhodi in izhodi, kot je razvidno s Slika 6.1.

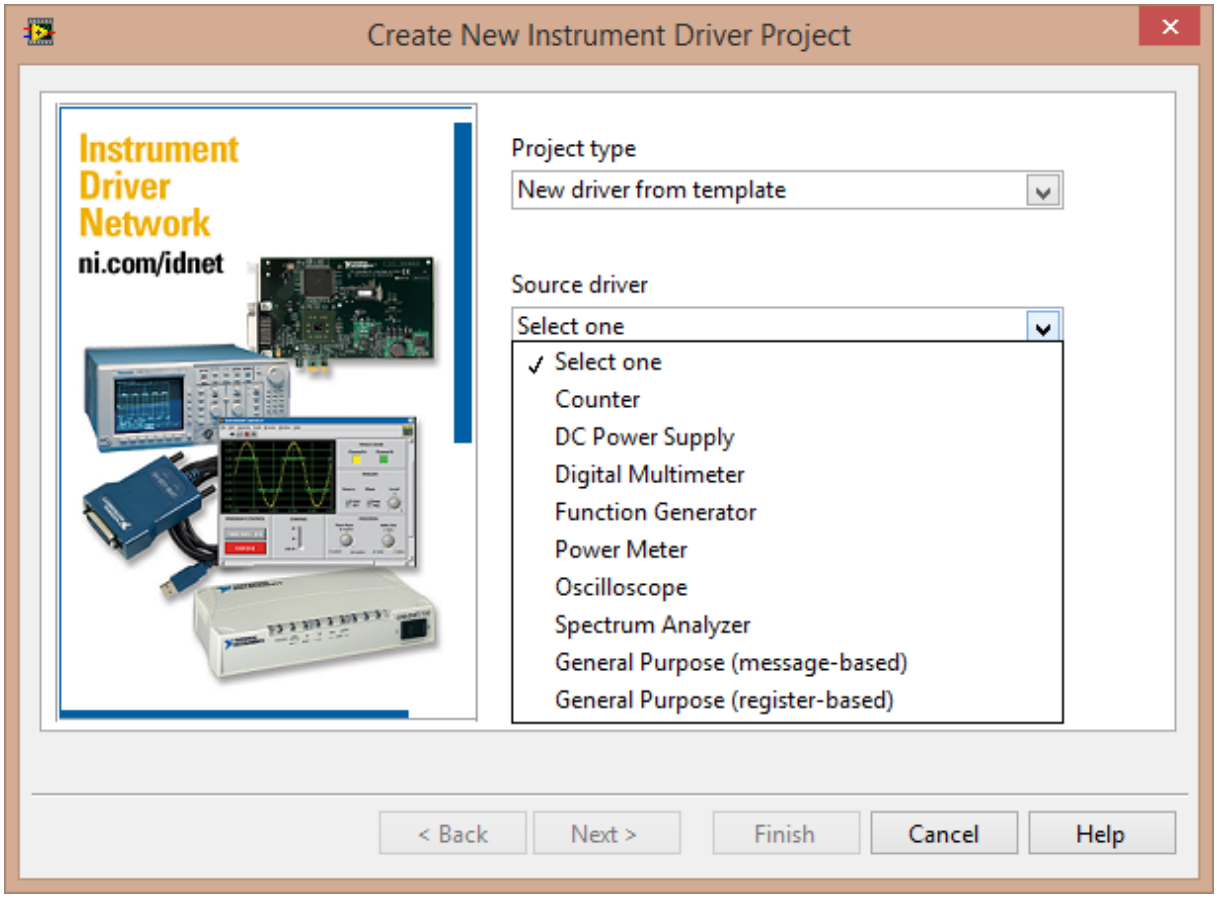


Slika 6.1: Osnutek bloka z vhodi

6.2. Drugi korak: Uporaba čarovnika za izdelavo projekta gonilnikov za inštrumente

Ko ima razvijalec za vsak VI blok določene Ukaze SCPI, začne z uporabo programa LabVIEW. V tem koraku izdela LabVIEW-projekt. Za izdelavo tega projekta uporabi čarovnik za izdelavo projekta gonilnikov za inštrumente (ang. Instrument driver project wizard), saj s tem izdela projekt zelo hitro in enostavno.

Do čarovnika se dostopi v začetnem oknu LabVIEW v orodni vrstici po korakih Orodja – Tools / Instrumentacija – Instrumentation / Izdelaj projekt gonilnika za inštrument – Create instrument driver project.

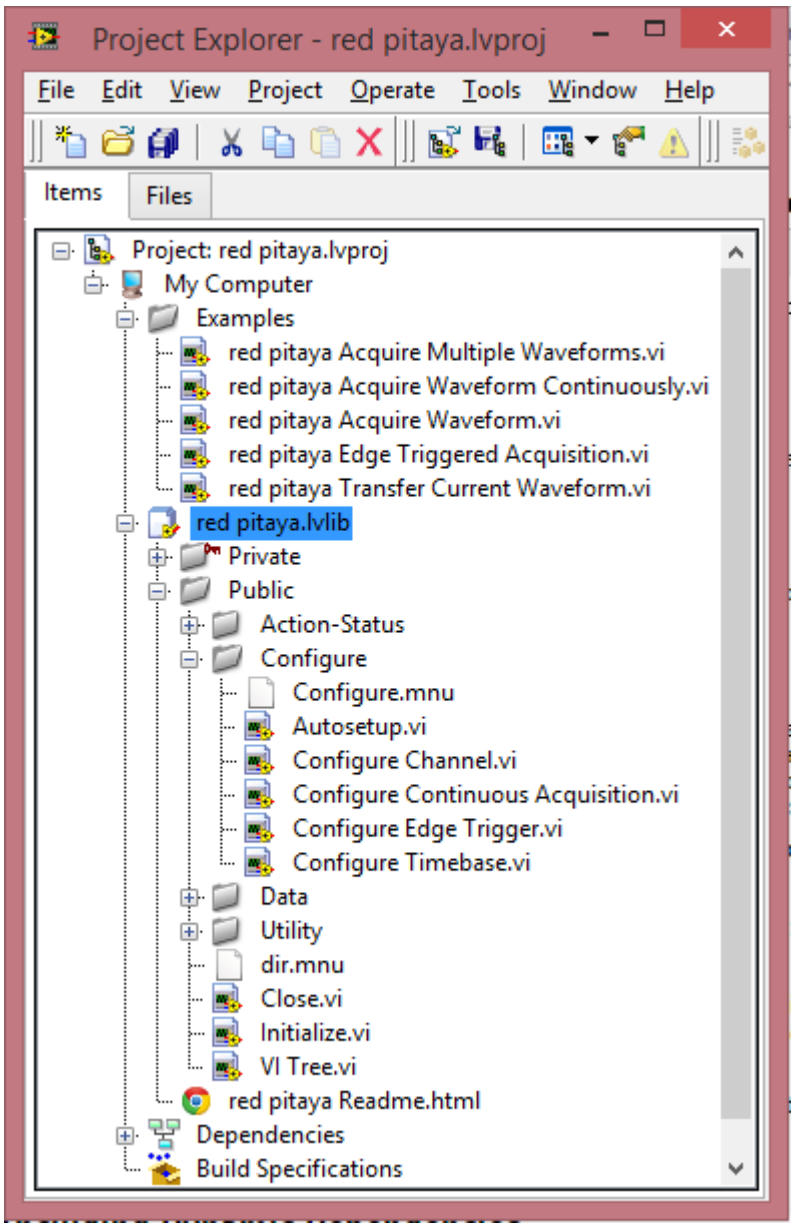


Slika 6.2: Čarovnik za izdelavo projekta gonilnikov za inštrument

V prvem koraku čarovnika se določi tip projekta in vir gonilnika. Tip projekta je lahko nov projekt s predlogo ali nov projekt z uvoženim obstoječim gonilnikom. Pri viru gonilnika se izbere tip inštrumenta, za katerega se bo izdelal gonilnik. Za Red Pitayo se je izbralo nov projekt s predlogo ter osciloskop kot vir gonilnika. Ker ima Red Pitaya tudi funkcijo signalnega generatorja, se je VI-je za generator naredilo brez predloge, čeprav se lahko vzame tudi iz drugih predlog VI-je. Ob kliku na gumb naprej – next, se v drugem koraku določi ime gonilnika in njegov opis.

V tretjem koraku se vnese dve ikoni. Prva je predloga za vse API VI-je v gonilniku, druga je za prikaz na paleti funkcij. Priporočeno je, da ima razvijalec te slike že pripravljene pri tem koraku, saj se kasneje v projektu to naredi težje.

V četrtem koraku se določi lokacijo projekta. Ker je za dostop do gonilnikov na paleti funkcij pogoj, da se mapa projekta nahaja v mapi LabVIEW 20xx/instr.lib, je kot privzeto ta lokacija nastavljena v čarovniku. S klikom na zaključek – finish se zapre okno čarovnika in se generira LabVIEW-projekt s predlogo za gonilnik.



Slika 6.3: LabVIEW-projekt s predlogo gonilnika za osciloskop

Ta projekt vsebuje že vse elemente, potrebne za gonilnik. Na Slika 6.3 je na vrhu najprej vidna mapa primeri – Examples, ki vsebuje 5 osnovnih primerov VI-jev LabVIEW-aplikacij

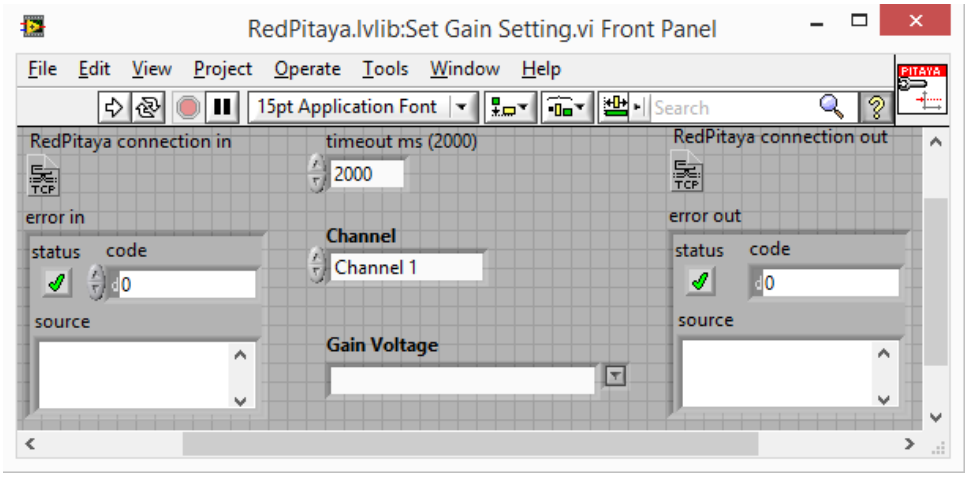
za osciloskop. Pod mapo primeri je knjižnica gonilniških VI-jev. Tu so vsebovani vsi osnovni VI-ji za osciloskop. Poleg tega je v vsaki virtualni mapi vsebovana tudi menu (mnu) datoteka, ki vsebuje informacije o lokaciji mape v paleti funkcij. V knjižnici VI-jev je tudi datoteka (red pitaya Readme.html) z vsemi informacijami o gonilniku. Pod knjižnico VI-jev je set vseh odvisnosti med VI-ji, imenovan odvisnosti – dependencies. Pod odvisnostmi je mapa datotek za gradnjo projekta (ang. Build specifications).

6.3. Tretji korak: Izdelava API VI-jev

V tem koraku se izdela VI-je, ki so specifični za ta inštrument, ter prilagodi Red Pitayi tiste, ki so že vsebovani v predlogi projekta.

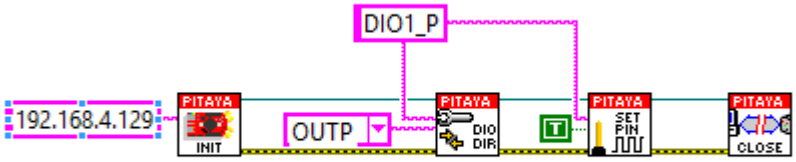
Preden razvijalec začne z izdelavo novih VI-jev, je potrebno, da se drži nastavitev, ki so aplicirane na VI-jih iz predloge:

- čelne plošče morajo imeti enako barvo ozadja,
- terminal na čelni plošči morajo biti postavljeni tako, da so kontrolna polja na levi, indikatorji na desni strani čelne plošče (primer prikazan na Slika 6.4),
- avtomatična obdelava napak mora biti onemogočena,
- nastavitev za postavitev terminalov na čelni plošči kot ikon mora biti onemogočena,
- okna ne smejo biti maksimirana,
- izogibati se je treba kombinaciji podVI-jev različnih skupin (konfiguracija, akcija/status, podatki) v en višje nivojski VI,
- vsi VI-ji morajo vsebovati opis za lažjo uporabo,
- izogibati se je potrebno ikonam VI-jev, ki vsebujejo samo tekst,
- označiti je potrebno priključke VI-jev kot obvezne, priporočene ali opcijske.



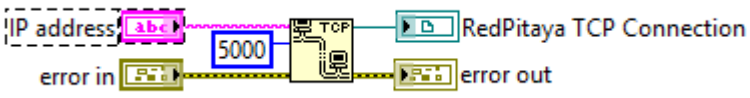
Slika 6.4: Čelna plošča VI-ja za nastavitev vhodne napetosti

Vsi API VI-ji, razen za inicializacijo in konec komunikacije, morajo vsebovati dva vhoda in dva izhoda za informacijo o omrežni povezavi in informacijo o napaki. Na Slika 6.5 sta razvidni dve povezavi, ki se zaporedno povežeta med vsemi bloki. Zelena povezava prenaša informacijo o omrežni povezavi protokola TCP. Rjava povezava z belo črtkano črto v sredini prenaša informacijo o napaki. Ta je tipa množica (ang. Cluster), ki vsebuje več vrednosti različnih tipov. Ta, ki prenaša informacijo o napaki, vsebuje boolean vrednost o statusu, celoštevilsko vrednost o identifikaciji napake ter niz znakov o izvoru napake, če se je ta zgodila. Obe povezavi se začneta pri VI-ju za inicializacijo in končata pri VI-ju za konec komunikacije. Zato vsebuje VI za inicializacijo samo indikatorska terminala za obe povezavi, VI za konec vsebuje samo kontrolna terminala za obe povezavi.



Slika 6.5: Primer uporabe gonilnišlih VI-jev za nastavitev pozitivnega stanja na priključku

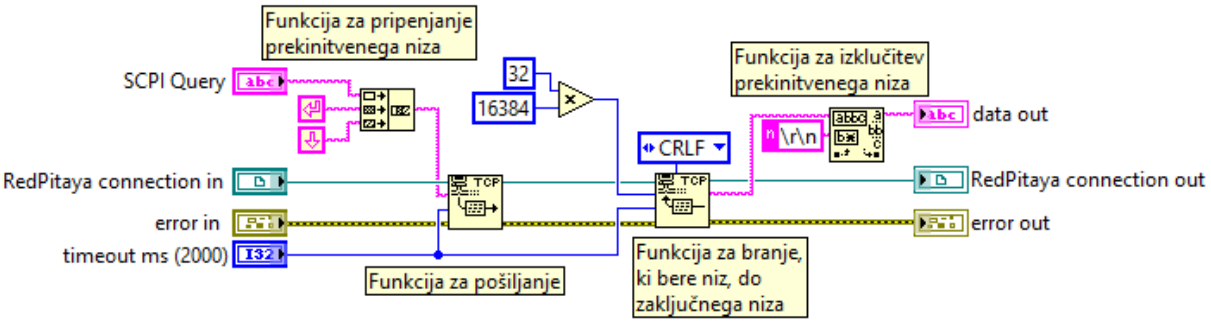
Ker na Red Pitayi ni VISA strežnika, tudi v gonilniku ne teče komunikacija preko VISA funkcij, temveč komunikacijo izvajajo navadne TCP funkcije. Zaradi tega smo najprej izdelali nova inicializacijski in »stop« VI, da je komunikacija tekla po navadnem TCP protokolu. Na Slika 6.6 je prikazan blokovni diagram inicializacijskega VI-ja. Ta vsebuje le eno funkcijo, in sicer za odprtje TCP povezave. Proti standardnemu VI-ju za inicializacijo ima ta mnogo manj funkcionalnosti. Predvsem mu manjka procedura za preverjanje inštrumenta, torej če gonilnik komunicira z napravo, ki jo podpira. Poleg tega standardni VI za inicializacijo vsebuje VISA funkcije za komunikacijo, ki vsebujejo dodatne parametre za konfiguracijo komunikacije, med tem ko TCP funkcije prejmejo le IP naslov in informacijo o napaki.



Slika 6.6: Blokovni diagram VI-ja inicializacija

SCPI strežnik na Red Pitayi bere ukaze, dokler ne pride do prekinitvenega niza ki vsebuje zaporedna znaka začetek vrste (ang. Carriage return) in za njim nova vrsta (ang. Line feed). V

VISA funkcijah se to nastavi in zapakira v VI za inicializacijo, pri TCP funkcijah tega ni. Pri branju prejetih nizov znakov je na koncu niza isti prekinitveni niz znakov, ki ga je potrebno izključiti. Zato smo za bolj pregleden blokovni diagram naredili dodatna VI-ja za pošiljanje ukazov in pošiljanje poizvedbe in ju kasneje uporabili pri vseh ostalih API Vi-jih. Na Slika 6.7 je prikazan blokovni diagram VI-ja za poizvedbo. Tu se izvede funkcija za pošiljanje ukazov, ki pošlje poizvedbo na določeno informacijo, in funkcija za branje prejetega niza znakov, ki prejme informacijo.



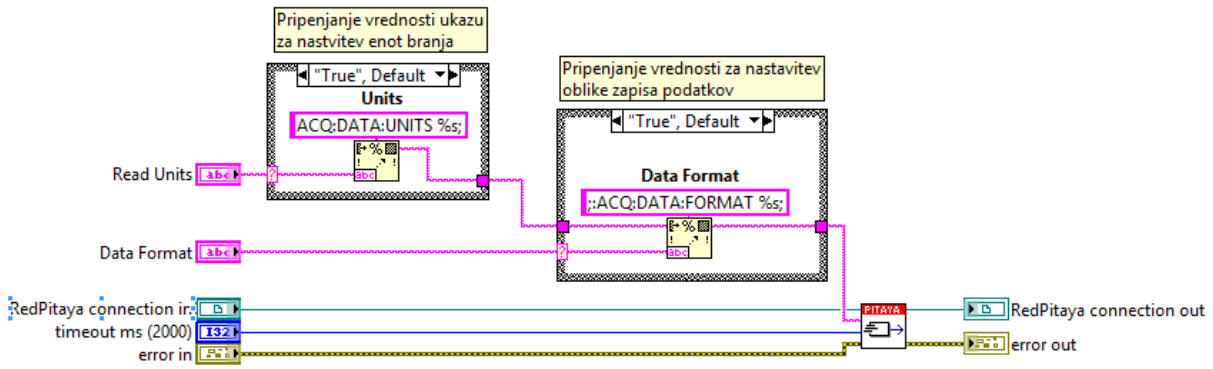
Slika 6.7: Blokovni diagram VI-ja za poizvedbo

Ko so narejeni vsi VI-ji za komunikacijo z inštrumentom, se začne izdelava VI-jev, ki so specifični za delovanje tega inštrumenta. To so vsi VI-ji za konfiguracijo, akcijo/status, podatkovni ter servisni.

6.3.1. VI-ji za konfiguracijo

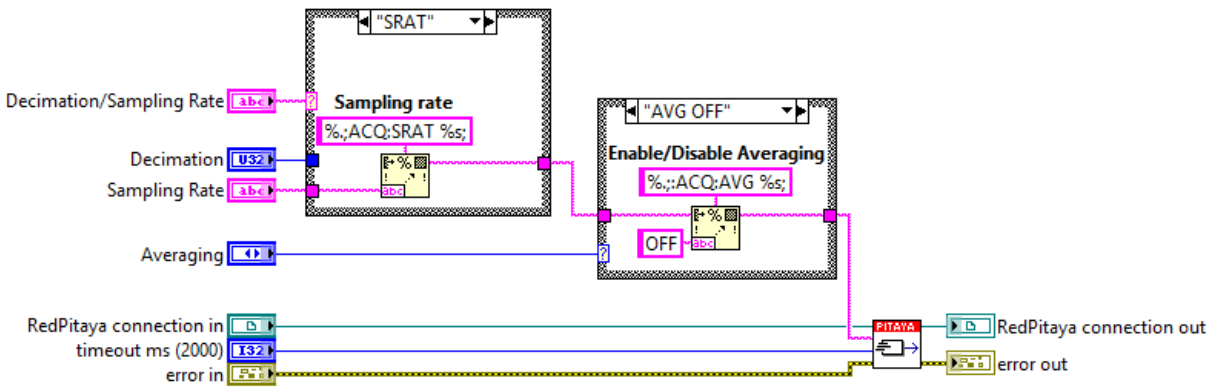
VI-ji iz te skupine imajo nalogo nastavljanja parametrov za željeno delovanje inštrumenta. Zato vsebujejo vsi VI-ji iz te skupine VI za pošiljanje ukazov. Vsakem nizu ukaza SCPI, ki vsebuje vrednosti, je potrebno pripeti te vrednosti v obliki niza znakov. Za spajanje dveh nizov ukazov se med nizoma vstavi znaka podpičje in za njim dvopičje (;:). To je prikazano na Slika 6.8 na zgornjem srednjem delu, kjer se dvema nizoma ukazov vstavita vrednosti s funkcijo za vstavljanje niza v drug niz. Pri standardnih LabVIEW-gonilnikih za inštrumente posamezen VI pošlje vse vsebovane ukaze SCPI, ko se ta izvrši. To se zgodi tudi v primeru, ko ima VI več ukazov SCPI, uporabnik želi nastaviti le vrednosti za en ukaz SCPI v tem VI-ju. Ta problem smo pri VI-jih v LabVIEW-gonilniku za Red Pitayo delno odpravili. To smo naredili tako, da smo na vseh veznih elementih nastavili privzeto vrednost takšno, ki ne velja za delovanje inštrumenta. Vsak ukaz smo dali v strukturo, ki preverja vrednost za ukaz. Če je vrednost enak privzeti, se ukaz ne pošlje. Ta rešitev ni popolna, saj obstaja verjetnost, da

uporabnik ne ve, kakšno je območje vrednosti za določen parameter in po pomoti vpiše vrednost, ki je nastavljena kot privzeta.



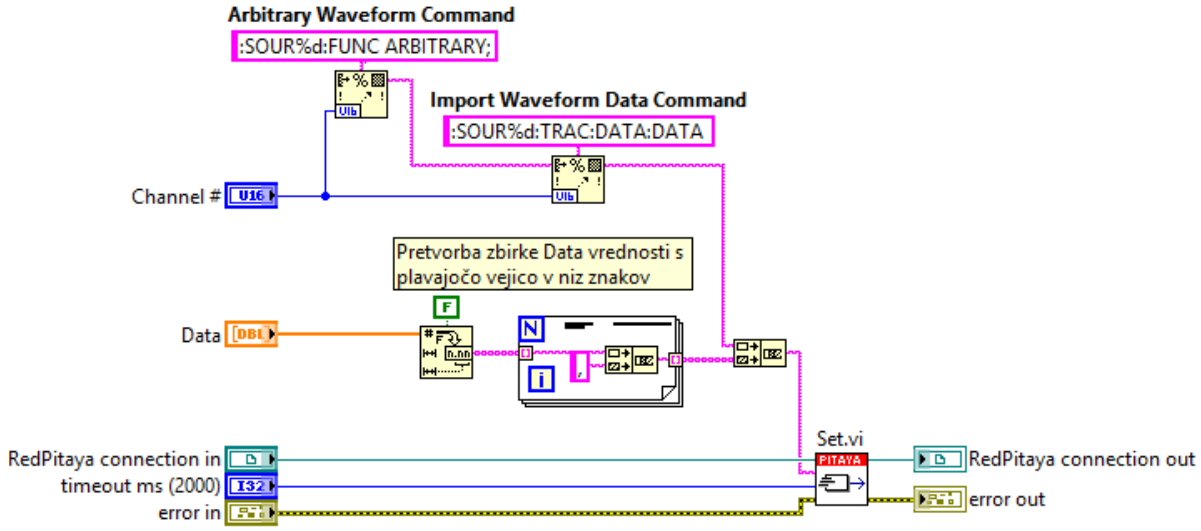
Slika 6.8: VI za nastavitev branja podatkov

VI za nastavitev parametrov zajema in VI za nastavitev parametrov proženja zajema vsebujeta vsak po dva ukaza, ki nastavita en pramater. Pri prvem je to decimacija zajema, ki se zapiše tudi kot hitrost vzorčenja. Pri drugem je to nastavitev zakasnitve proženja v številu vzorcev ali v nano sekundah. Če damo oba ukaza za parameter na VI, lahko pride do nejasnosti s strani uporabnika. Pri obeh Vi-ji smo ta problem rešili tako, da smo dodali vsakemu VI-ju po en vezni kontrolni element, ki določa, kateri ukaz se bo poslal. Na Slika 6.9 je prikazan blokovni diagram VI-ja za nastavitev parametrov zajema, kjer je terminal za določitev ukaza povezan na strukturo za preverjanje vrednosti. Struktura preverja vrednost omenjenega terminala in glede na to da na izhod pravilen niz znakov. Struktura vsebuje 3 primere. 2 primera vsebujeta blokovni diagram za generiranje niza ukaza, 1 primer vsebuje prazen niz znakov.



Slika 6.9: VI za nastavitev parametrov zajema

VI za nastavitev generiranja poljubnega signala smo tudi vključili v skupino konfiguracija, čeprav vsebuje ukaz za prenos podatkov o poljubnem signalu na Red Pitayo, kar sodi v skupino podatki. Vključili smo ga v skupini konfiguracija, ker se glede na zaporedje izvrševanja VI-jev ta izvede med konfiguracijo signalnega generatorja, torej preden se prične generiranje. VI kot vhodni parameter z informacijo o signalu vnese zbirko vrednosti s plavajočo vejico. V blokovnem diagramu (Slika 6.10) se vnesena zbirka pretvori v niz znakov po principu z vejico deljenih vrednosti (ang. Comma-sperated values – CSV), kar pomeni, da je med nizom znakov vsake vrednosti pripet znak vejica in ta se pripne na konec niza ukaza za pošiljanje podatkov o poljubnem signalu.



Slika 6.10: Blokovni diagram VI-ja za nastavitev generiranja poljubnega signala

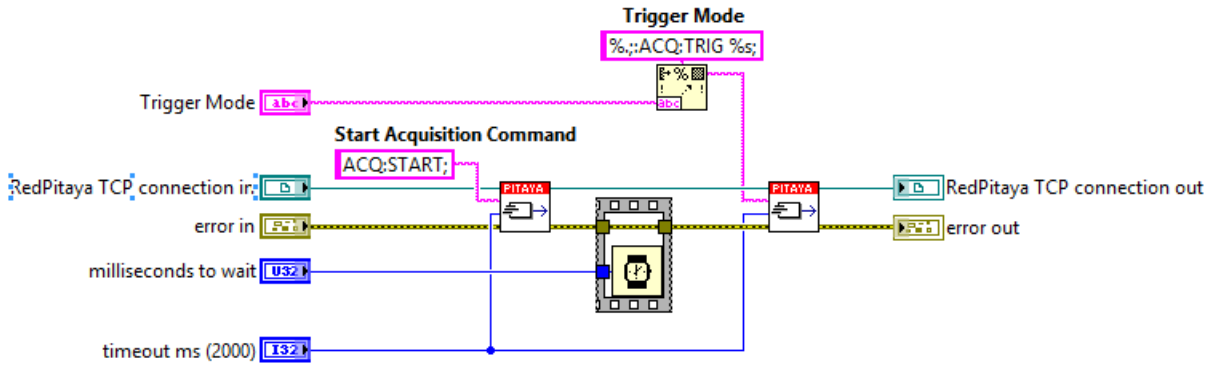
6.3.2. VI-ji za akcijo in status

VI-ji iz te skupine pošiljajo inštrumentu ukaze za izvajanje in prekinitev operacij iz meritev in testiranj. To so operacije za:

- ponastavitev parametrov zajema ter generiranja signala,
- aktivacijo izhodnega kanala ter rafala,
- nastavitev napetosti na digitalnem ter analognem izhodu,
- aktivacijo zajema,
- preverjanje stanja proženja.

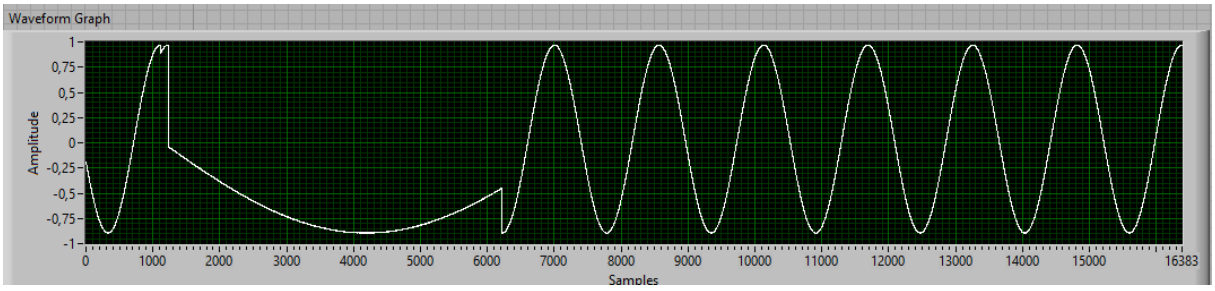
VI za aktivacijo zajema pošlje ukaz za nastavitev tipa proženja, kar sicer sodi v skupino konfiguracija. Ker se mora na Red Pitayi nastaviti tip proženja po aktivaciji zajema, smo to

komando vključili v ta VI. V blokovnem diagramu (Slika 6.11) se tako najprej pošlje ukaz za aktivacijo zajema in za tem še ukaz za nastavitev tipa proženja.



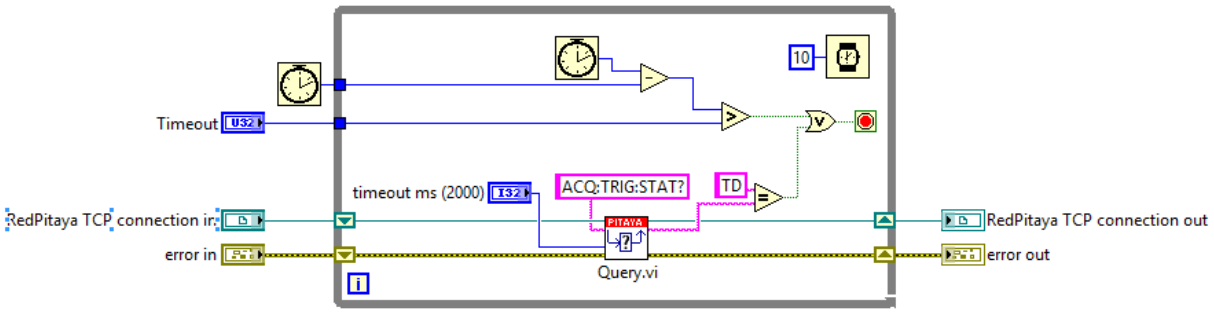
Slika 6.11: Blokovni diagram VI-ja za aktivacijo zajema

Pri zaporednem pošiljanju ukazov za aktivacijo zajema in nastavitev tipa proženja je možno, da se po aktivaciji zajema izvrši proženje, preden se v podatkovnem medpomnilniku prepišejo vsa pomnilniška mesta z novimi vzorci. Na Slika 6.12 je prikazan zajem takšnih podatkov, kjer so novi podatki prikazani po 6200. vzorcu, pred tem so podatki iz prejšnjih zajemov. Ta problem smo rešili z dodatnim vhodnim terminalom, ki kontrolira zakasnitev pošiljanja ukaza za nastavitev tipa proženja.



Slika 6.12: Zajeti podatki ob prehitri izvršitvi proženja, prikazani v grafu na čelni plošči

VI za preverjanje stanja proženja se uporablja za čakanje na sprožitev proženja. Glavni element blokovnega diagrama () je zanka med tem ko (ang. While loop). V iteraciji se izvede poizvedba na stanje proženja. V primeru, da se je proženje izvedlo, se zanka konča. VI ima en dodaten vhodni terminal za nastavitev najdaljšega možnega časa čakanja. Če se zanka izvaja toliko časa, kot je nastavljeno na omenjenem terminalu, se zanka konča. V vsaki iteraciji se izvede zakasnitev, dolga 10 ms. Namen tega je zagotoviti branje odziva na poizvedbo. Če te zakasnitve ni, je zelo velika verjetnost, da se podatki iz odgovora na poizvedbo ne vpišejo pravi čas na mesto za branje.

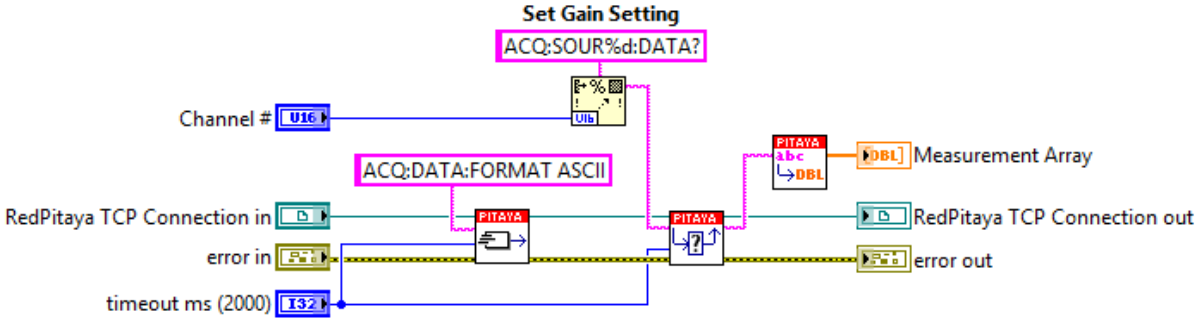


Slika 6.13: Blokovni diagram VI-ja za preverjanje stanja proženja

6.3.3. Podatkovni VI-ji

V skupini podatkovni VI-ji so VI-ji, ki izvajajo prenos podatkov. V LABVIEW-gonilnikih za Red Pitayo so to VI-ji za prenos zajetega signala iz Red Pitaye na računalnik in eden VI za poizvedbe na vse parametre z eno vrednostjo.

Vsi VI-ji za prenos zajetega signala iz Red Pitaye na računalnik imajo zelo podobno strukturo blokovnega diagrama (Slika 6.14). Med izvajanjem se najprej pošlje ukaz za nastavitev formata zapisa podatkov na ASCII. Za tem se pošlje ukaz za prejem podatkov iz Red Pitaye. Prejeti podatki so zapisani v nizu znakov v ASCII zapisu. Na prvem mestu niza znakov je zaviti oklepaj, na zadnjem mestu je zaviti zaklepaj. Vmes so z ASCII znaki zapisana števila po principu CSV. Niz znakov je povezan na VI, ki pretvori ASCII zapis v množico števil s plavajočo vejico. Ta množica je nato povezana na izhodni terminal.

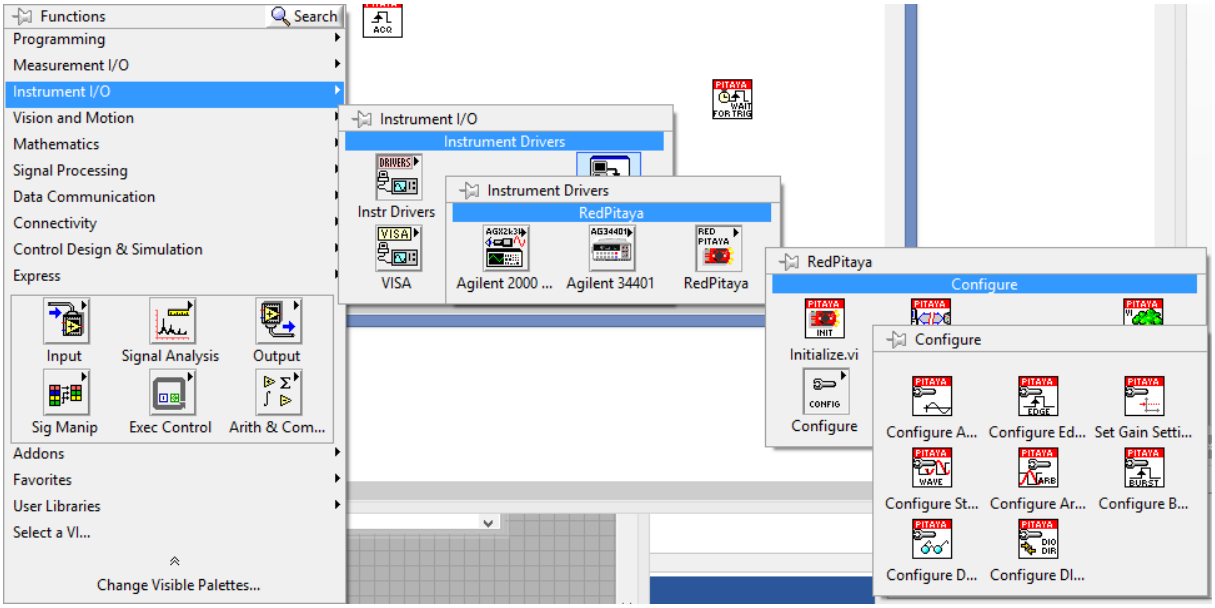


Slika 6.14: Blokovni diagram VI-ja za prenos celega zajetega signala

6.4. Četrti korak: izdelava menija za paleto funkcij

Ko so vsi VI-ji za gonilnik narejeni, se naredi meni gonilnikov za prikaz v paleti funkcij. Podatki o meniju so shranjeni v datotekah meni (ang. menu), ki so razporejene v projektni mapi po mapah skupaj z VI-ji. Vsaka datoteka meni vsebuje informacijo o prikazu VI-jev, ki so v isti mapi. VI-ji za gonilnik so prikazani v paleti funkcij (Slika 6.15) pod Inštrument izhod vhod – Instument IO / gonilniki za inštrumente – Instrument drivers / Ime gonilnika. Da so

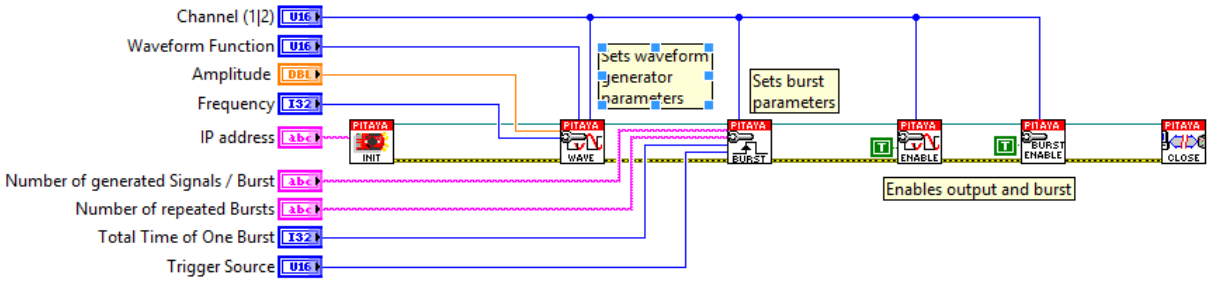
VI-ji prikazani v paleti funkcij, mora biti mapa projekta locirana v mapi instr.lib, ki se nahaja v mapi LabVIEW 20xx.



Slika 6.15: Meni gonilnikov Red Pitaya v paleti funkcij

6.5. Peti korak: izdelava VI-jev primerov

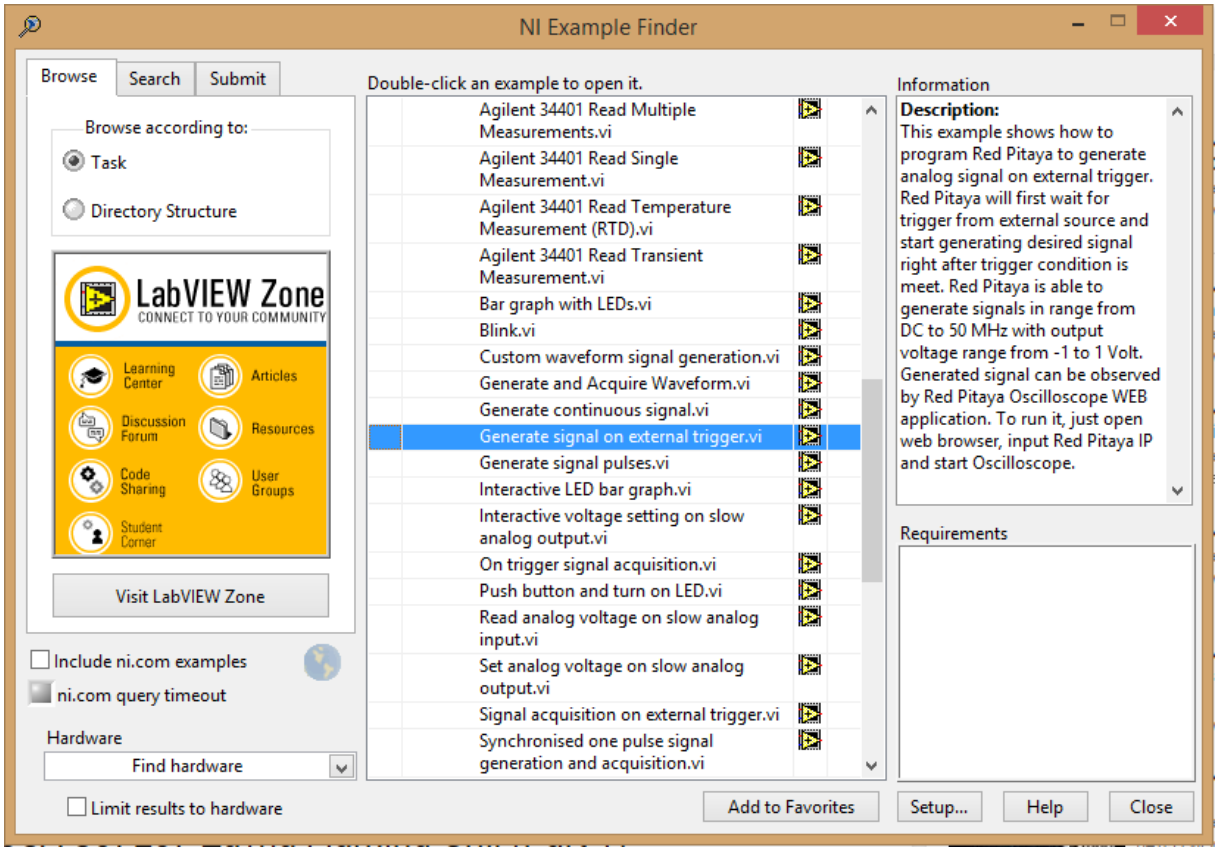
VI-ji primerov aplikacij inštrumenta so referenca delujoče aplikacije. S tem uporabnik vidi, kako mora biti sestavljen blokovni diagram z vključitvijo VI-jev gonilnika za inštrument. VI-ji primerov morajo vsebovati enostavne primere uporabe funkcionalnosti inštrumenta. LabVIEW-gonilniki za Red Pitayo vsebujejo VI-je primerov aplikacij za vse štiri izhodno-vhodne elemente (digitalni priključki, analogni priključki, kanala za signalni generator, kanala za zajem signala). VI-ji primerov za aplikacije z digitalnimi priključki vsebujejo primere nastavitev stanja na digitalnih izhodih in branje stanja na digitalnih vhodih. VI-ji primerov za aplikacije z analognimi priključki vsebujejo primere nastavitev napetosti na analognih izhodih in branje napetosti iz analognih vhodov. VI-ji primerov za aplikacije s signalnim generatorjem vsebujejo primere za generiranje neprekinjenih signalov, rafalov signalov (Slika 6.16), poljubnih signalov ter proženje rafalov signalov z zunanjim proženjem. VI-ji primerov za aplikacije z zajemom signala vsebujejo primere za zajem signala ob različnih tipih proženja, ter sinhrono generiranje rafala in zajem tega rafala.



Slika 6.16: Blokovni diagram VI-ja za primera za generiranje rafala

Ko so vsi VI-ji primerov narejeni, morajo biti nujno testirani, če delujejo. Ker se uporabljajo za učenje uporabe gonilnika mora vsak VI primera vsebovati vsaj en komentar, ki opisuje delovanje blokovnega diagrama.

Vsi VI-ji primerov morajo biti prikazani v iskalniku primerov (ang. NI Example finder), kot je razvidno na Slika 6.17. V projektu gonilnikov se to naredi v čarovniku za pripravo VI-jev primerov v iskalniku primerov. V čarovniku se vsakemu VI-ju nastavi ime, lokacijo, opis ter ključne besede. Ko so te nastavitve določene vsem VI-jem primerov, se klink na gumb narejeno – done. Za tem se generira datoteka s kratico bin3, ki vsebuje informacije o prikazu VI-jev primerov v iskalniku primerov.



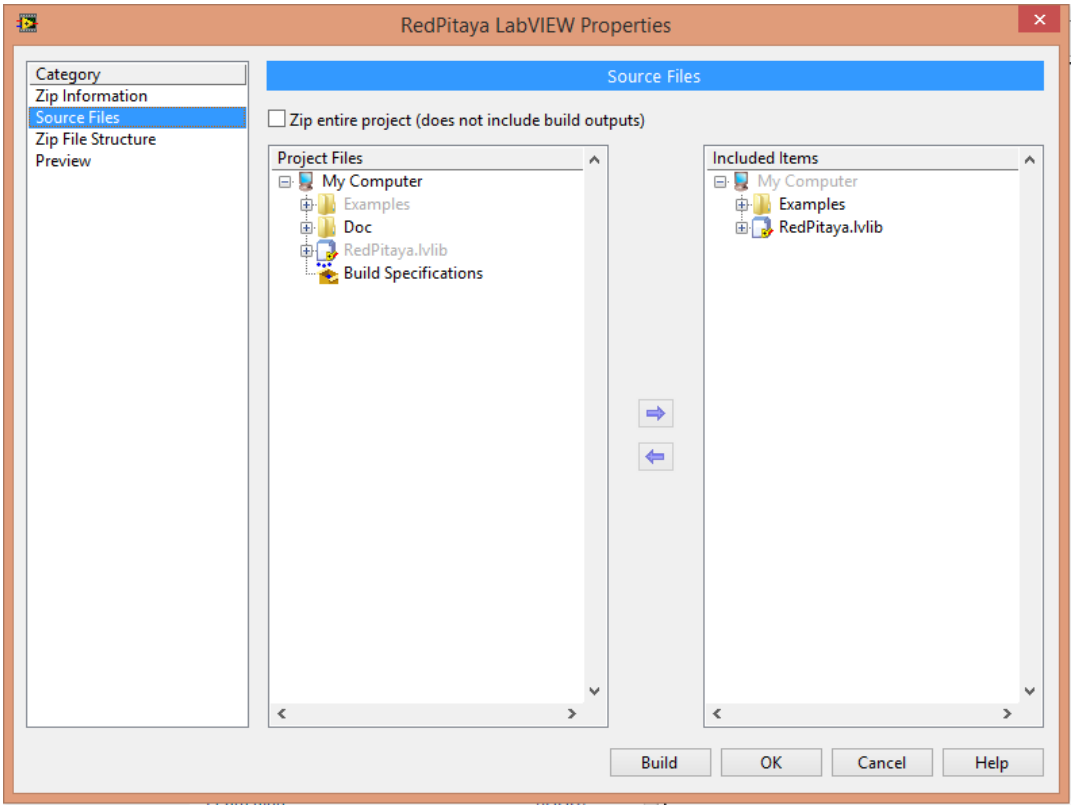
Slika 6.17: Iskalnik primerov

6.6. Šesti korak: informacije o gonilniku

Predloga projekta gonilnika za inštrumente vsebuje predlogo datoteko HTML (ang. HyperText Markup Language) za pisanje dokumentacije z imenom Readme. V datoteko se vpiše informacije o tehnologiji gonilnikov, proizvajalcu, podprtih programskih jezikih (LabVIEW), podprtih modelih inštrumentov, testiranjih inštrumentih z gonilniki, vmesniku za povezavo, programski podpori na inštrumentu (ang. firmware), potrditvi gonilnikov, podpori s strani podjetja National Instruments, dostopnosti izvorne kode, verziji gonilnikov, datumu prvega izida ter datumu trenutnega izida gonilnika. V datoteki so tudi navodila za namestitev gonilnika, lista vseh znanih napak gonilnika ter seznam popravkov.

6.7. Sedmi korak: nastavitev gradnje stisnjene datoteke projekta

Ko je projekt končan, se ga lahko objavi na spletni strani NI omrežje gonilnikov za inštrumente. Gonilnik mora biti za objavo v stisnjeni obliki. Zato se v čarovniku (Slika 6.18) izdela proceduro za generiranje stisnjene datoteke projekta. V čarovniku se določi mape, ki bodo vključene v stisnjeni datoteki, lokacijo stisnjene datoteke in ime stisnjene datoteke. Ko je vse določeno, se proceduro potrdi s klikom na gumb narejeno – done. Za generiranje stisnjene datoteke se nato uporabi narejeno proceduro. Ob vsakem zaključenem popravku se uporabi isto proceduro za generiranje stisnjene datoteke.



Slika 6.18: Čarovnik za načrtovanje procedure generiranja stisnjene datoteke projekta

7. Sklep

Razvojna plošča Red Pitaya vsebuje kanala za signalni generator in kanala za zajem signala, kar omogoča izdelavo ogromnega števila aplikacij na področju merjenja in testiranj.

Razvoj LabVIEW-gonilnikov za inštrumente zahteva veliko znanja o ciljnem inštrumentu, ki ga bo gonilnik podpiral, ter osnovno znanje programa LabVIEW. Stvari, ki so specifične za razvoj LabVIEW-gonilnikov, so razložene na spletni strani podjetja NI. V velikem številu korakov razvoja se uporablja namenske čarovnike, ki olajšajo delo. Največ težav pri razvoju gonilnikov sem imel pri izdelavi meni-datotek za prikaz VI-jev gonilnika v paleti funkcij, ker je tu najprej potrebno uvoziti projekt v mapo instr.lib, nato modificirati paleto funkcij ter po modifikaciji prekopirati meni-datoteke v originalen projekt.

Trenutna različica gonilnika prejme signal in ga pretvori v listo vrednosti. Za prikazovanje na grafu je v tem primeru na x-osi prikazano število vzorcev. Priporočeno bi bilo izboljšati VI-je za prenos podatkov signalov, ki bi na izhod dali podatke o signalu v formatu graf, kar je sedaj v formatu lista.

Zaradi konstantnih nadgradenj v ekosistemu Red Pitaye in s tem tudi sprememb v delovanju ukazov SCPI, bo v prihodnosti potrebnih prav toliko nadgradenj v LabVIEW-gonilnikih za Red Pitayo. Prva nadgradnja bo povečava prostora podatkovnega medpomnilnika na Red Pitayi. Za to bo potrebno prilagoditi VI za poizvedbo.

8. Priloge

Tabela 1: Ukazi SCPI za LED diode in GPIO priključke [10]

SCPI komanda	Opis
DIG:PIN:DIR <dir>,<pin>	Določi smer priključka
DIG:PIN <pin>,<state>	Določi stanje priključka
DIG:PIN? <pin>	Poizvedba na stanje priključka

Tabela 2: SCPI komande za počasne analogne vhode in izhode [10]

SCPI komanda	Opis
ANALOG:PIN <pin>,<value>	Določi napetost na priključku
ANALOG:PIN? <pin>	Poizvedba na napetost priključka

Tabela 3: SCPI komande za konfiguracijo signala signalnega generatorja [10]

SCPI komanda	Opis
OUTPUT<n>:STATE <par>	Določi odprtje ali zaprtje kanala
SOUR<n>:FREQ:FIX <value>	Določi frekvenco signala na kanalu
SOUR<n>:FUNC <par>	Določi tip signala na kanalu
SOUR<n>:VOLT <value>	Določi amplitudo signala na kanalu
SOUR<n>:VOLT:OFFS <value>	Določi enosmerno komponento signala (offste) signala na kanalu
SOUR<n>:PHAS <value>	Določi fazni zamik ned signaloma na obeh kanalih
SOUR<n>:DCYC <par>	Določi čas trajanja visokega stanja signala na kanalu
SOUR<n>:TRAC:DATA:DATA <array>	Pošlje listo vrednosti signala za določen kanal
GEN:RST	Ponastavitev parametrov generatorja

Tabela 4: SCPI komande za konfiguracijo in upravljanje rafal signalov [10]

SCPI komanda	Opis
SOUR<n>:BURS:STAT <par>	Omogoči ali onemogoči generiranje rafal signala na kanalu
SOUR<n>: BURS:NCYC <value>	Določi število period v rafalu na kanalu
SOUR<n>: BURS:NOR <value>	Določi število ponovljenih rafalov na kanalu
SOUR<n>: BURS:INT:PER <value>	Določi celoten čas enega rafala na kanalu
SOUR<n>:TRIG:SOUR <par>	Določi tip proženja za proženje rafal signala na kanalu
SOUR<n>: TRIG:IMM	Komanda za takojšnjo sprožitev rafala na kanalu
TRIG:IMM	Komanda za takojšnjo sprožitev rafala na obeh kanalih

Tabela 5: SCPI komande za konfiguracijo in kontroliranje zajemanja [10]

SCPI komanda	Opis
ACQ:START	Komanda za začetek zajemanja
ACQ:STOP	Komanda za ustavitev zajemanja
ACQ:RST	Ponastavi parametre za zajem signala
ACQ:DEC <par>	Določi decimacijo zajemanja vzorcev
ACQ:DEC?	Poizvedba na decimacijo zajemanja vzorcev
ACQ:SRAT <par>	Določi hitrost zajemanj
ACQ:SRAT?	Poizvedba na hitrost zajemanj, odgovori z okrajšano številko
ACQ:SRA:HZ?	Poizvedba na hitrost zajemanj, odgovori v Hz
ACQ:AVG <par>	Omogoči ali onemogoči povprečenje
ACQ:AVG?	Poizvedba na povprečenje
ACQ:SOUR<n>:GAIN <par>	Določi razpon zajete napetosti

Tabela 6: SCPI komande za konfiguracijo in kontroliranje proženja zajema [10]

SCPI komanda	Opis
ACQ:TRIG <par>	Določi tip proženja
ACQ:TRIG:STAT?	Poizvedba na stanje proženja
ACQ:TRIG:DLY <par>	Določi zakasnitev proženja
ACQ:TRIG:DLY?	Poizvedba na zakasnitev proženja, odgovori s številom vzorcev zakasnitve
ACQ:TRIG:DLY:NS?	Poizvedba na zakasnitev proženja, odgovori s časom v nanosekundah zakasnitve
ACQ:TRIG:LEV <par>	Določi nivo proženja
ACQ:TRIG:LEV?	Poizvedba na nivo proženja

Tabela 7: SCPI komade za poizvedbo o vmesnem pomnilniku [10]

SCPI komanda	Opis
ACQ:WPOS?	Poizvedba na trenutno lokacijo kazalca
ACQ:TPOS?	Poizvedba na lokacijo, kjer se je zgodilo proženje
ACQ:BUF:SIZE?	Poizvedba na velikost vmesnega pomnilnika

Tabela 8: SCPI komande za branje podatkov iz vmesnega pomnilnika [10]

SCPI komanda	Opis
ACQ:SOUR<n>:DATA:STA:END? <start_pos>,<end_pos>	Pošlji podatke od pozicije v vmesnem pomnilniku do pozicije v vmesnem pomnilniku na kanalu
ACQ:SOUR<n>:DATA:STA:N? <start_pos>,<m>	Pošlji m podatkov v vmesnem pomnilniku od določene pozicije na kanalu
ACQ:SOUR<n>:DATA?	Pošlji celoten vmesni pomnilnik z začetnim podatkom pri najstarejšem vzorcu na kanalu
ACQ:SOUR<n>:DATA:OLD:N? <m>	Pošlji m podatkov po zakasnitvi proženja na kanalu
ACQ:SOUR<n>:DATA:LAT:N? <m>	Pošlji m podatkov pred zakasnitvijo proženja na kanalu

8.1. Seznam tabel

9. Viri

[1] Slika inštrumenta Red Pitaya (22.8.2015)
http://www.samm.com/userfiles/productlargeimages/product_1770.jpg.

[2] R. Wisniewski, Synthesys of compositional microprogram control units for programmable devices, Zielona Góra: University of Zielona Góra, 2009, str. 17-21.

[3] Spletna stran Wikipedia, API (22.8.2015)
https://en.wikipedia.org/wiki/Application_programming_interface.

[4] Slika direktne povezave preko ethernet kabla z Red Pitayo in računalnikom (23.8.2015)
http://redpitaya.com/wp-content/uploads/2015/01/direct.png.

[5] Slika povezave Red Pitaya na usmerjevalnik (23.8.2015) http://redpitaya.com/wp-content/uploads/2015/01/Internet-con-red-pitaya-turniton1.png.

[6] Slika povezave preko WiFi (23.8.2015) http://redpitaya.com/wp-content/uploads/2015/01/wifi-con-red-pitaya-turniton.png.

[7] Slika priključkov na zadnji strani Red Pitaye (23.8.2015)
http://wiki.redpitaya.com/tmp/Bootup_procedure_side.png.

[8] Slika avtomatiziranje proizvodnje (23.8.2015)
http://www.autoalliance.org/images/dmImage/SourceImage/AdvancedTech-Cobots.png.

[9] SCPI Consortium, 1999 SCPI Syntax & Style, La Mesa: SCPI Consortium, 1999.

[10] Dokumentacija SCPI komand za ploščo Red Pitaya (17.8.2015)
https://dl.dropboxusercontent.com/s/eiihbzicmucjtlz/SCPI_commands_beta_release.pdf.

[11] Spletna stran Wikipedia, Data Buffer (1.6.2015)
https://en.wikipedia.org/wiki/Data_buffer.

[12] S. Sumathi in S. Surekha, LabVIEW based Advanced Instrumentation Systems, Springer-Verlag Berlin Heidelberg, 2007.

[13] Slika logotipa programa LabVIEW (24.8.2015)
http://www.inds.co.uk/engineering/images/NI_LabVIEW_Logo.jpg.

[14] N. Adorno. (maj 1998). Developing a LabVIEW™ Instrument Driver. Dosegljivo: http://www.thierry-lequeu.fr/data/AN006ni.pdf. [Dostopano: 28.8.2015].

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Marka Jankovca. Izkazano pomoč ostalih sodelavcev sem v celoti navedel v zahvali

Ljubljana, september 2015

Primož Perušek