Sadržaj

1 Uvod

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) je razvojno okruženje za razvoj aplikacija u grafičkom programskom jeziku G. Najčešće se koristi za akviziciju podataka, automatizaciju industrijskih procesa, kontrolu instrumenta i analizu dobijenih podataka, mada se mogu razvijati aplikacije opšte namene. LabVIEW je podržan na različitim softverskim platformama kao što su Microsoft Windows, Linux, različite verzije UNIX-a i Mac OS X. Osim navedenih operativnih sistema, realizovane aplikacije se mogu kompajlirati za operativne sisteme za rad u realnom vremenu (PharLap, VXWorks) a podržane su i hardverske platforme bazirane na FPGA ili ARM mikrokontrolerima.

Virtuelna instrumentacija je pojam koji označava integraciju hardvera i softvera, obuhvatajući procese merenja, akvizicije, obrade i prezentacije dobijenih podataka. Virtuelna instrumentacija je trend u tehnici i nauci zbog rastuće kompleksnosti inženjerskih poslova, potrebe za velikim brojem specijalizovanih i skupih instrumenata i softvera, potrebe za visokokvalifikovanim ljudima za rad sa tim instrumentima kao i organizovanja podele rada sa instrumentima i softverom. Osnovna paradigma ovog koncepta je *virtuelni instrument*.

Virtuelni instrument je višeslojni hardversko-softverski sistem. Njegova namena je identična funkciji klasičnog instrumenta. Razlika je u fleksibilnosti i funkcionalnosti koju je moguće menjati, za razliku od klasičnog instrumenta koji ima nepromenljivu funkcionalnost, određenu od strane proizvođača. Prednosti virtuelnih instrumenata nad klasičnim su i veća mogućnost obrade i prezentacije rezultata merenja. Fizički, odnosno hardverski sloj virtuelnog instrumenta čini računar sa odgovarajućom mernom, akvizicionom i dodatnom opremom. Ekspanzija proizvodnje personalnih računara, naročito intenzivna u prethodnih dvadeset godina, podstakla je razvoj uređaja za akviziciju i obradu podataka sa odgovarajućim interfejsima za računar. Veliki broj klasičnih mernih uređaja takođe poseduje interfejs za povezivanje sa računarom. Ovi uređaji se mogu povezati eksterno preko serijskog RS232/RS422/RS485 interfejsa, GPIB, paralelnog interfejsa, ili preko bržih USB 1.1/2.0/3.0 i Ethernet interfejsa. Najnoviji trendovi u povezivanju su bežične senzorske mreže (*Wireless Sensor Networks*), koje koriste različite implementacije bežičnih protokola (WLAN, *Bluetooth, ZigBee*) i autonomne metode napajanja. Ethernet interfejs je naročito pogodan, jer pruža mogućnost povezivanja velikog broja različitih uređaja i računara u heterogenu mrežu. Komunikacija sa uređajima se realizuje preko standardnih OSI i TCP/IP modela. Ovakva realizacija osim skalabilnosti omogućava i povećanje fizičke distance između uređaja i korisnika. Ukoliko se primenjuje TCP/IP model, razdaljina je praktično neograničena.

Softverski sloj virtuelnog instrumenta čine drajveri za rad sa uređajem i softver višeg nivoa za akviziciju i obradu podataka – aplikacija virtuelnog instrumenta. Često u literaturi termin virtuelni instrument predstavlja sinonim za aplikaciju virtuelnog instrumenta. Funkcija aplikacije je definisanje signala koji se generišu, obrada podataka dobijenih akvizicijom i njihovo prezentovanje. Obrada podataka obuhvata različite funkcije kao što je analiza u vremenskom i frekventnom domenu, različita izračunavanja fizičkih veličina koja se posredno mere pretvaranjem u električne, itd. Aplikacije virtuelnih instrumenata imaju iste osobine kao i druge aplikacije. Mogu se realizovati kao multithread aplikacije, klijent/server ili web aplikacije, postoji i mogućnost njihovog izvršavanja u obliku sistemskog servisa. Omogućena je i jednostavna prezentacija i razmena podataka u obliku standardnih i široko rasprostranjenih formata.

Predstavljanje podataka je najčešće u grafičkom obliku, funkcijama i kontrolama grafičkog interfejsa koje svojim izgledom podsećaju na realne merne instrumente. Interfejs je interaktivan i ima dvosmernu funkciju: osim predstavljanja pruža mogućnost kontrole procesa merenja, definisanjem parametara signala koji se generišu (talasni oblik, amplituda, frekvencija, faza, modulacija,...) ili čak definisanjem topologije mernog kola primenom matrica prekidača.

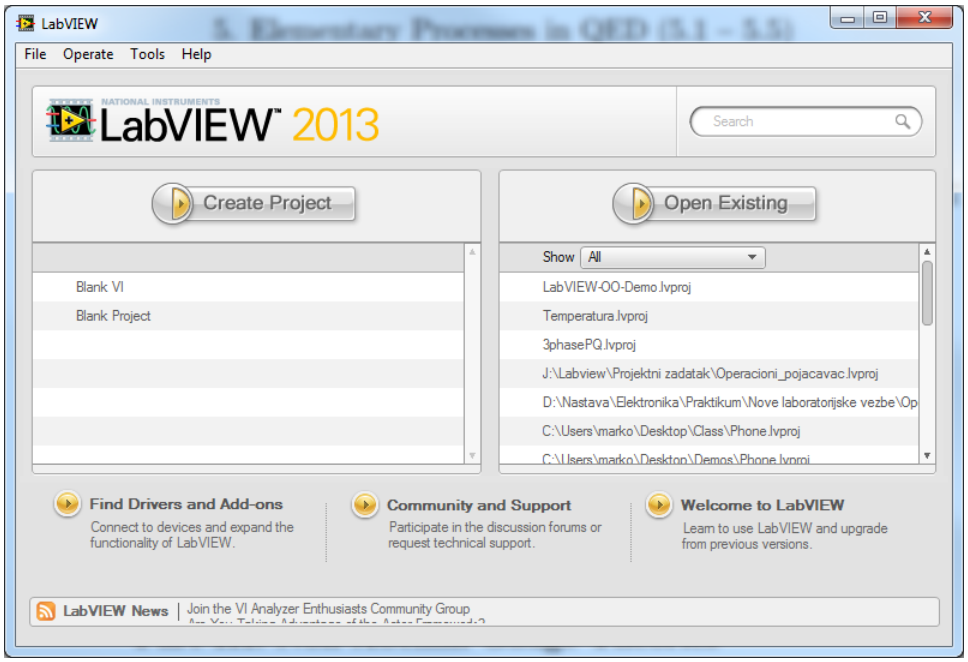
Postoje različiti razvojni alati i okruženja za realizaciju aplikacija virtuelnih instrumenta. Ovi alati mogu biti dodaci za već postojeća okruženja za razvoj softvera (uglavnom C ili C++) ili kao posebni alati za ovu namenu (LabVIEW). Posebni alati za razvoj aplikacija virtuelnih instrumenata omogućuju intuitivan i jednostavan razvoj bez potrebe za programerskim znanjem. Razvoj aplikacija je na taj način omogućen izvođaču merenja, pružajući posebnu fleksibilnost. Virtuelni instrumenti realizovani na taj način mogu imati opštu namenu ili biti specijalizovani, namenjeni za *ad hoc* merenja.

Pomenuti koncept virtuelne instrumentacije pruža mnogobrojne prednosti u odnosu na klasične metode merenja. Dodatna obrada podataka, povezivanje sa bazama podataka, izvođenje procesa merenja sa velike udaljenosti kroz web interfejs su samo neke mogućnosti. Najznačajnija prednost je vizuelizacija podataka. Vizuelizacija podataka, odnosno mogućnost njihove vizuelne analize i prezentacije predstavlja najbrži put da se sagleda celokupna situacija, uoče problemi i pronađu odgovarajuća rešenja.

2 LabVIEW okruženje

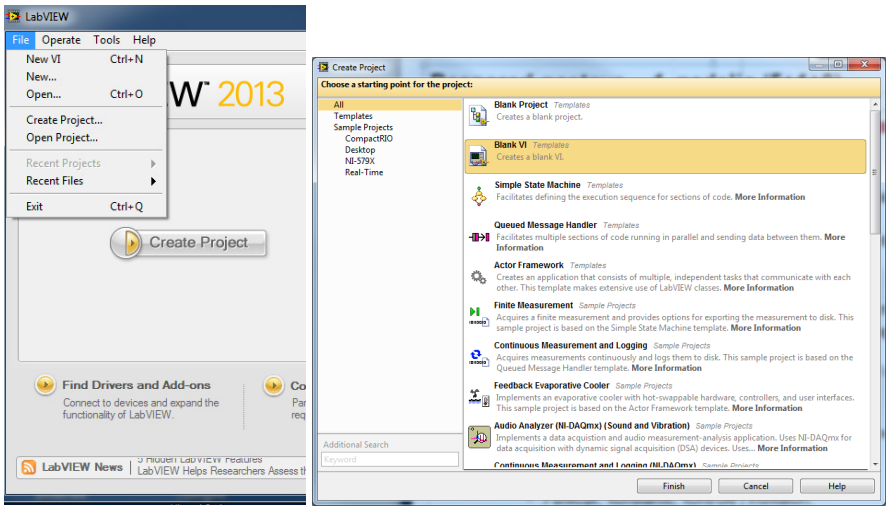
2.1 Startup dijalog

Okruženje se sastoji od više elemenata koji su aktivni u različitim fazama realizacije virtuelnog instrumenta ili projekta. Prilikom pokretanja razvojnog okruženja, aktivan je *startup dijalog* prikazan na slici 2.1. Osnovne funkcije dijaloga su započinjanje novog virtuelnog instrumenta/projekta ili otvaranje postojećeg. Pored osnovnih funkcija, dijalog pruža mogućnost pristupa *online* resursima, pretraživanje primera, interaktivnu pomoć i pretragu drajvera i opcionih dodataka. Iz dijaloga se može pristupiti drugim alatima za konfiguraciju okruženja, debagiranje i kompajliranje virtuelnih instrumenata.



Slika 2.1 Startup dijalog

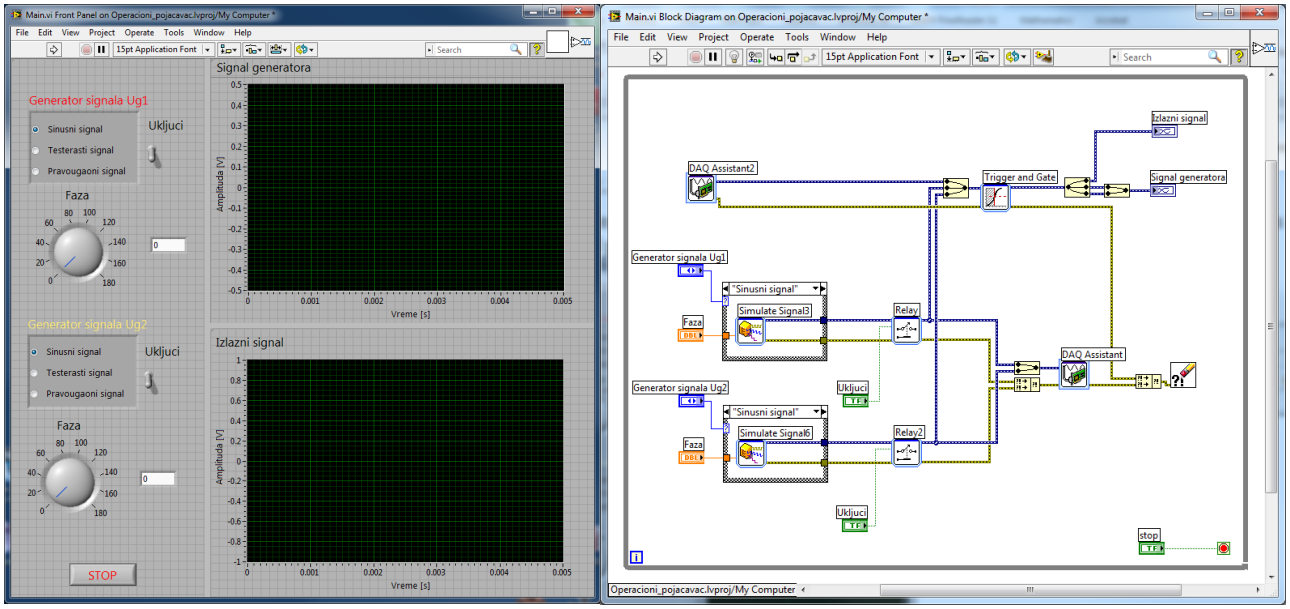
Novi virtuelni instrument se može kreirati na osnovu datih šablona ili kao prazan (Slika 2.2).



Slika 2.2 Kreiranje novog virtuelnog instrumenta – meni (levo) i dijalog sa postojećim šablonima (desno)

2.2 Front panel i Blok dijagram prozori

Nakon otvaranja/kreiranja novog virtuelnog instrumenta, aktivni su blok dijagram i front panel prozori (Slika 2.3)

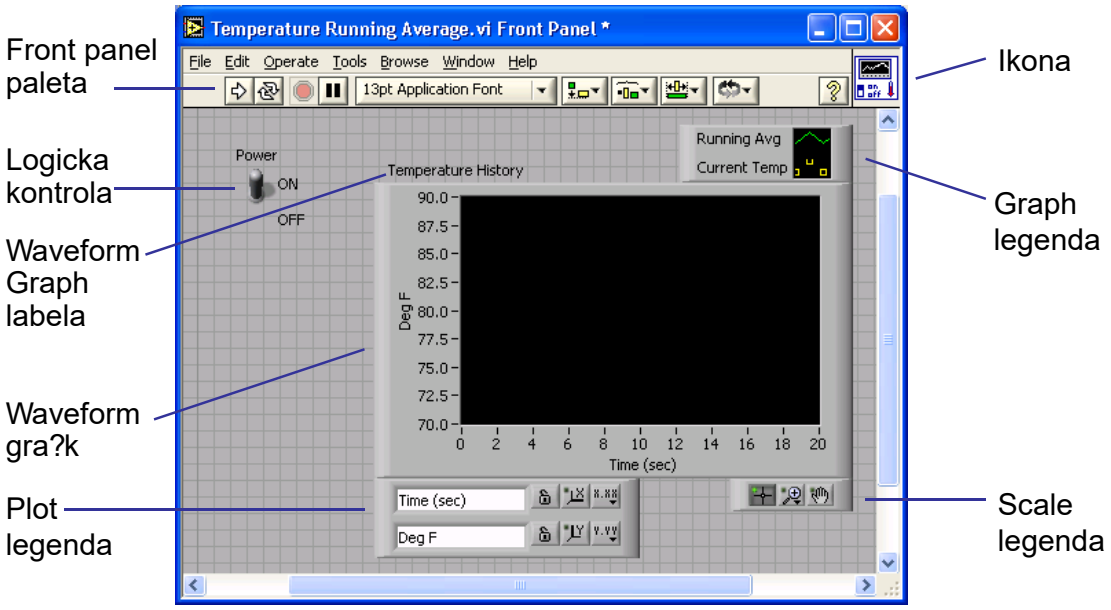


Slika 2.3 Front panel (levo) i blok dijagram (desno)

Prozori sadrže elemente virtuelnog instrumenta i alate za manipulaciju.

2.2.1 Front panel prozor

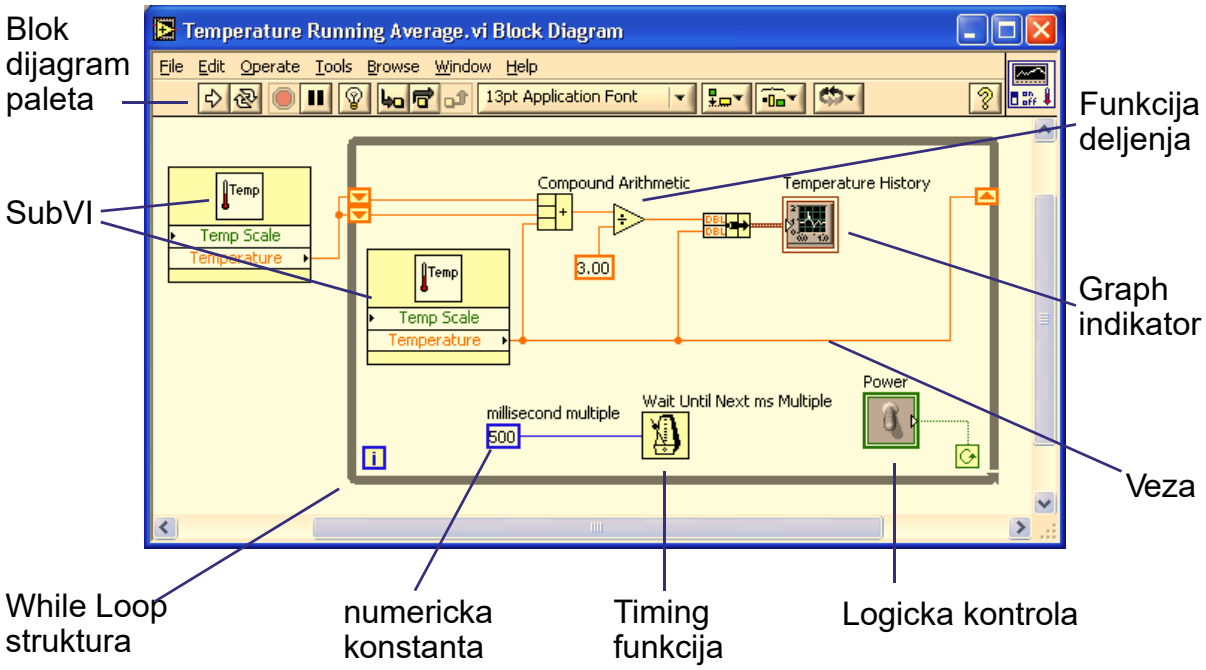
Front panel prozor je prikazan na slici 2.4. U gornjem delu prozora se nalaze meni i osnovna paleta sa alatima. Donji deo prozora sadrži elemente korisničkog interfejsa virtuelnog instrumenta (front panel). Ovi elementi se mogu dodati sa **Controls** palete i predstavljaju terminale (kontrole i indikatore) virtuelnog instrumenta, koji omogućuju interakciju korisnika i aplikacije.



Slika 2.4 Prozor Front panela

2.2.2 *Blok dijagram prozor*

Blok dijagram prozor je prikazan na slici 2.5.



Slika 2.5 Blok dijagram prozor

U gornjem delu prozora se nalaze meni i paleta sa alatima. Donji deo prozora sadrži elemente blok panela, koji predstavljaju kôd aplikacije. Elementi blok dijagrama se mogu proizvoljno dodavati sa palete **Functions** i povezivati odgovarajućim vezama.

2.2.3 *Meniji*

Na vrhu prozora se nalaze meni sa uobičajenim funkcijama. Glavni meni sadrži sledeće menije:

- Meni **File** sadrži opcije za osnovne operacije sa fajlovima, kao što su otvaranje, zatvaranje, snimanje, štampanje i slično.
- **Edit** meni objedinjuje opcije za rad sa virtuelnim instrumentima i njegovim elementima.
- **View** meni sadrži opcije vezane za izgled radnog okruženja, aktiviranje paleta, hijerarhiju virtuelnih instrumenata, klasa, eksternih komponenti i slično. U meniju
- **Project** se nalaze opcije za otvaranje, kreiranje i rad sa LabVIEW projektima.
- **Operate** meni objedinjuje kontrole za pokretanje, kontrolu toka izvršavanja i debagiranje virtuelnih instrumenata.
- **Tools** meni sadrži kontrole za konfigurisanje okruženja, virtuelnog instrumenta, projekta, drajvera ili drugih komponenti.
- **Window** služi za podešavanje izgleda radnog okruženja, prozora i paleta.
- **Help** je skup opcija za pomoć u realizaciji virtuelnih instrumenata, pristup dokumentaciji, online resursima i primerima, kao i tehničkoj podršci.

2.2.4 Paleta sa alatima

Ispod menija se nalazi paleta sa alatima (**Toolbar**), prikazana na slici 2.6.

Slika 2.6 Toolbar

Paleta sa alatima Front panela sadrži sledeće komande (s leva na desno):

- dugme **Run**, za izvršavanje virtuelnog instrumenta,
- dugme **Continuous Run**, za višestruko pokretanje virtuelnog instrumenta – nakon izvršavanja VI, LabVIEW će ponovo pokrenuti izvršavanje aktivnog VI,
- dugme **Abort**, za zaustavljanje izvršavanja VI,
- dugme **Pause/Continue** za pauziranje i nastavljanje izvršenja VI,
- Drop-down meni za izbor fonta i veličine,
- Opcije za ravnanje, raspodelu i dimenzionisanje elemenata,
- Dugme za podešavanje redosleda elemenata,
- Dugme kojim se aktivira kontekstni help.

U prozoru blok dijagrama, paleta ima dodatne komande, prikazane na slici 2.7,

Slika 2.7 Dodatne komande Blok dijagram palete

koje služe za debagiranje virtuelnog instrumenta. Prva opcije je **Execution Highlight**, koja naglašava čvorove blok dijagrama koji se trenutno izvršavaju, kao i tok podataka na vezama između čvorova. Sledeće opcije služe za izvršavanje virtuelnog instrumenta korak-po-korak.

U slučaju da virtuelni instrument ima grešaka, **Run** dugme će promeniti izgled u **Broken Run**  , što ukazuje na postojanje grešaka koje se moraju izmeniti. Klikom na **Broken Run**, LabVIEW će prikazati listu identifikovanih grešaka.

2.2.5 Tools paleta

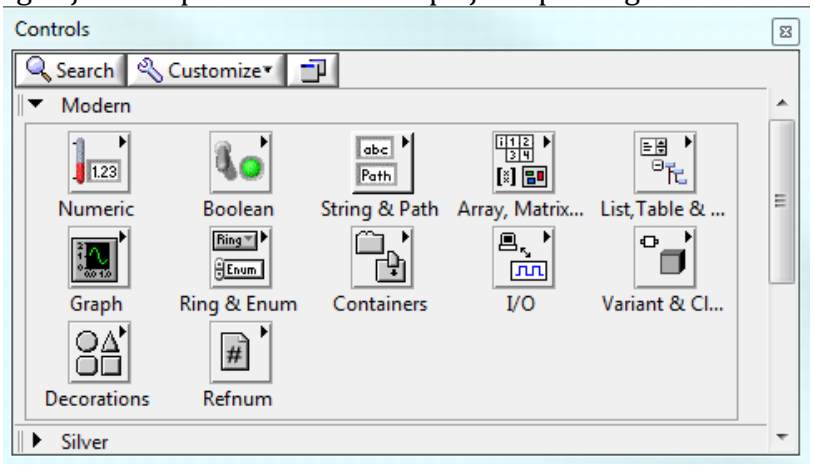
Tools paleta (slika 2.8) je dostupna u Front panel i Blok dijagram prozorima. Aktiviranjem opcija iz palete se menja môd rada sa objektima, što je indicirano promenom izgleda pointera. Tools paleta se može aktivirati dostupna izborom iz menija **VIEW » Tools Palette**.

Slika 2.8 Tools paleta

Ukoliko je selektovana opcija **Automatic Tool Selection,** koja se nalazi na vrhu palete, priliko, pomeranja kursora preko objekta, LabVIEW automatski selektuje odgovarajući alat iz palete.

2.2.6 Paleta sa kontrolama

Paleta sa kontrolama (**Control palette**) je dostupna na Front panel prozoru (slika 2.9). Paleta se može aktivirati desnim klikom ili iz menija **VIEW » Control Palette**. Paleta sadrži kontrole i indikatore, elemente korisničkog interfejsa virtuelnog instrumenta. Kontrole i indikatori su grupisani prema kategorijama i tipu. Paleta sadrži opcije za pretragu kontrola i indikatora po imenu.

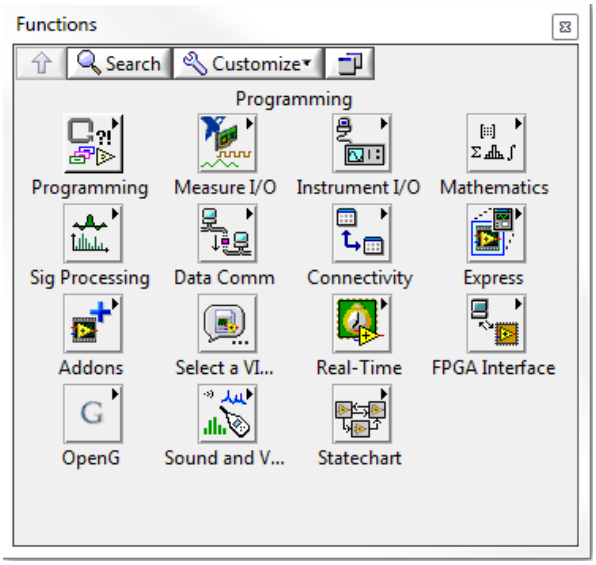


Slika 2.9 Paleta sa kontrolama – najčešće korišćene kontrole

Elementi front panela se postavljaju prevlačenjem odgovarajuće kontrole ili indikatora sa palete na prozor front panela.

2.2.7 Paleta sa funkcijama

Paleta sa funkcijama je dostupna na blok dijagram panelu (Slika 2.10). Paleta sadrži sve funkcije, programske strukture i module dostupne u LabVIEW. Paleta se može aktivirati desnim klikom ili iz menija **VIEW » Functions Palette**.



Slika 2.10 Paleta sa svim funkcijama – prikazane su sve raspoložive funkcije i programske strukture

Elementi blok dijagrama se postavljaju prevlačenjem odgovarajućih elemenata – čvorova, programskih struktura ili konstanti sa palete na blok dijagram. Elementi su na paleti grupisani prema tipu i nameni.

Functions paleta može biti fiksirana ili plivajuća. Subpalete – delovi glavne palete – se mogu posebno izdvojiti, u cilju lakše manipulacije elementima koji se često koriste. Paleta sadrži **Search** – opciju za pretragu funkcija po imenu. **Customize** opcija omogućava podešavanje prikaza elemenata na paleti.

2.3 Help

U okviru LabVIEW razvojnog okruženja korisnicima je na raspolaganju više opcija za pružanje pomoći. Na raspolaganju su **LabVIEW Help, Context Help, Explain Error, Find Examples, Web Resources** i drugi.

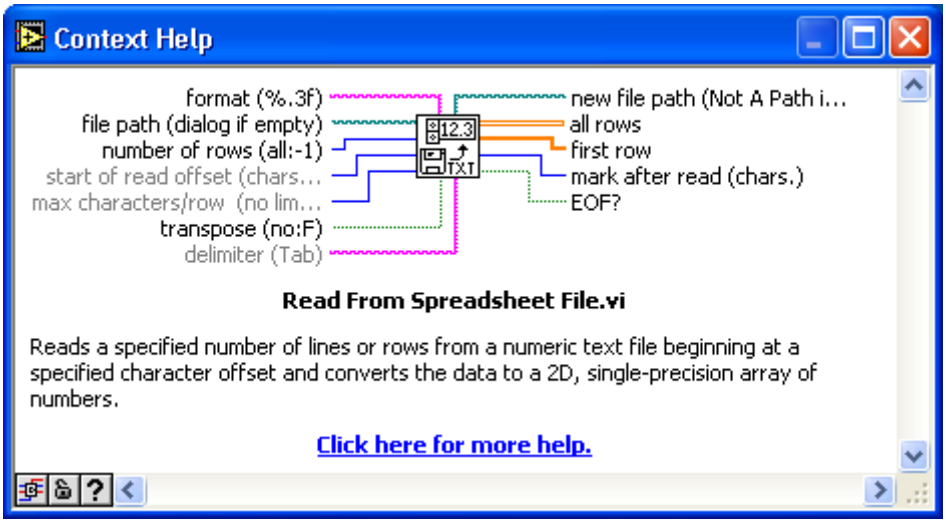
2.3.1 LabVIEW Help

LabVIEW Help je osnovni help razvojnog okruženja. Sadrži različite objašnjenja kontrola, indikatora, funkcija, programskih struktura i modula. Osnovni help je offline tipa. Help omogućuje pretragu prema kategorijama ili po nazivu elementa. Može se pristupiti direktno iz menija **Help » LabVIEW Help**, ili iz kontekstnog helpa.

Help sadrži detaljne opise većine paleta, menija, alata, VI, funkcija, instrukcija za korišćenje LabVIEW opcija, linkove ka online resursima, tutorijalima, PDF verzijama priručnika i tehničkoj podrški.

2.3.2 Context Help

Context Help omogućuje interaktivnu pomoć pri realizaciji virtuelnog instrumenta. Poziva se opcijom **Help » Show Context Help**, pritiskom <Ctrl-H> tastere, ili aktiviranjem **Show Context Help Window** dugmeta na paleti sa alatima. Kada je aktivan, u prozoru **Context Help**-a je prikazan selektovani čvor sa nazivom, pripadajućim ulazima i izlazima i kratkim objašnjenjem funkcije (Slika 2.11).

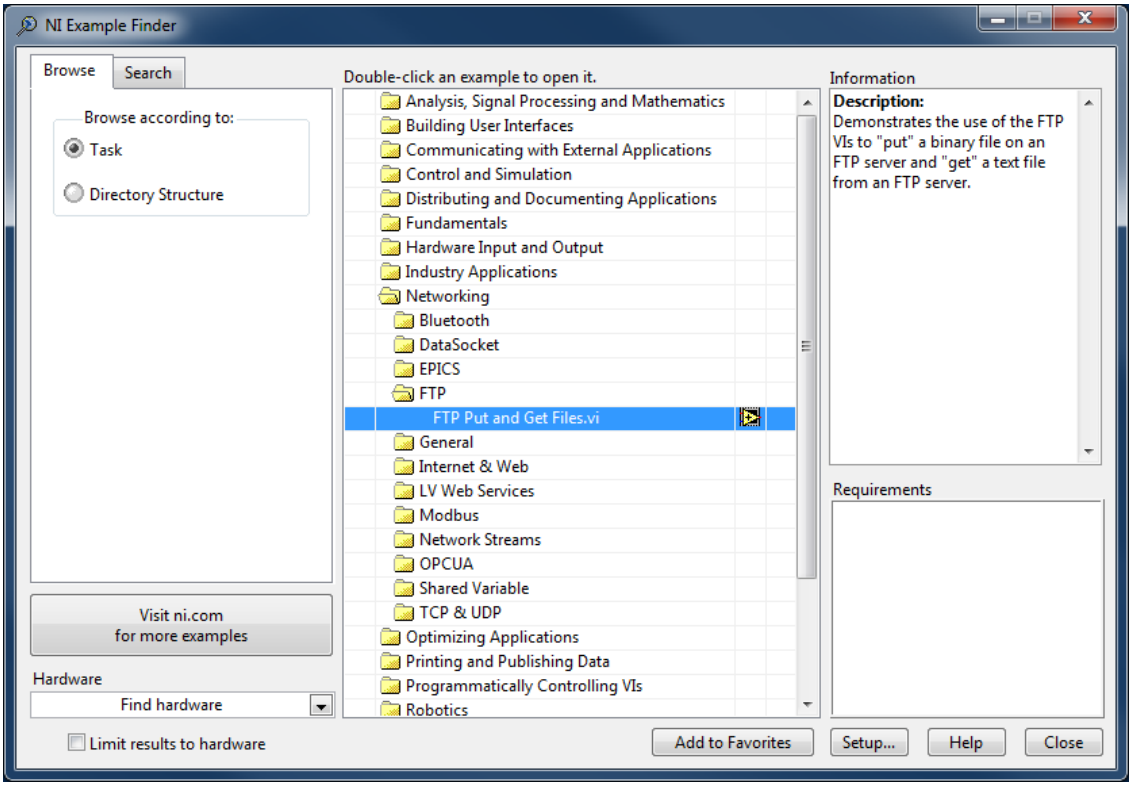


Slika 2.11 Kontekstni help

Iz **Context Help**-a je moguće pozvati detaljni help.

2.3.3 Online resursi

Iz LabVIEW okruženja je moguće pristupiti *online* resursima pomoću opcije **Help » Web Resources**. Opcija **Help » Find Examples** pokreće posebnu aplikaciju, **NI Example Finder**, (slika 2.12) koja omogućuje pretragu primera aplikacija i rešenja u različitim online i offline resursima.



Slika 2.12 NI Example Finder

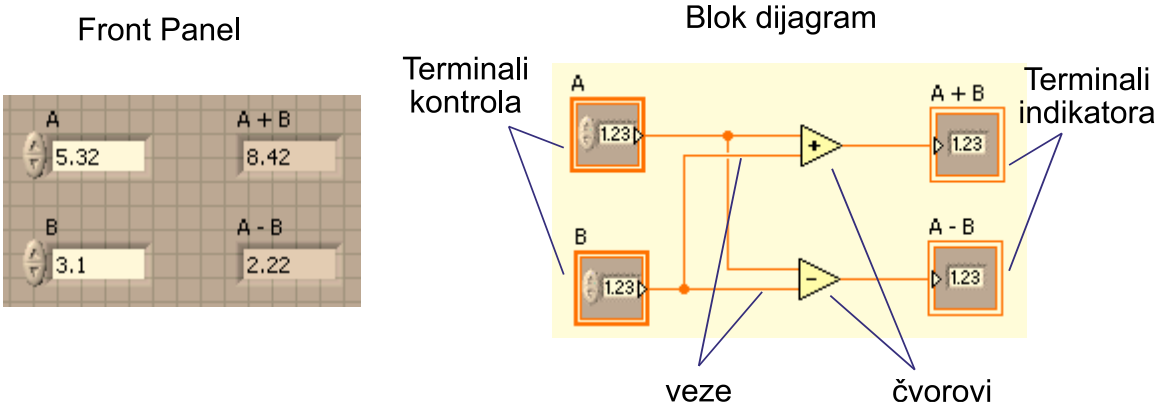
3 Elementi virtuelnog instrumenta

Aplikacije realizovane u LabVIEW okruženju se nazivaju *virtuelni instrumenti* (skraćeno VI). Za razliku od tradicionalnih programskih jezika sa tekstualnim kôdom kod kojih je redosled izvršavanja tekstualnih programskih jezika određen redosledom unetih dekleracija, programiranje u LabVIEW okruženju se izvodi povezivanjem čvorova koji predstavljaju određene funkcije. Redosled izvršavanja je definisan tokom podataka – *data flow*.

Virtuelni instrument se sastoji od nekoliko komponenti, od kojih su najznačajnije *front panel* i *blok dijagram*.

3.1 Front panel

Front panel predstavlja korisnički interfejs aplikacije i sadrži elemente koji omogućuju interakciju između korisnika i aplikacije (slika 3.1). Ulazni elementi front panela se nazivaju kontrole, izlazni indikatori. Front panel može sadržati i druge neinteraktivne i interaktivne elemente – dekoracije, menije, tabove, itd. Kontrole i indikatori imaju grafički izgled sličan realnim instrumentima i kontrolama. U zavisnosti od tipa podatka koji reprezentuju, elementi mogu biti logički, numerički, string, nizovi ili složeniji.



Slika 3.1 Front panel (levo) i blok dijagram (desno) virtuelnog instrumenta

Elementi front panela se postavljaju prevlačenjem iz **Controls** palete. Osobine elementa se mogu modifikovati kontekstnim menijem i dijalogom sa opcijama: izgled, tip i reprezentacija podatka, vidljive komponente elementa, opis itd.

Kontrola se može pretvoriti u indikator istog tipa, i obratno, izborom opcije **Change to indicator/control** iz kontekstnog menija.

3.2 Blok dijagram

Blok dijagram sadrži grafički kôd aplikacije, sačinjen od terminala, čvorova, veza i programskih struktura (slika 3.1).

Terminali blok dijagrama (slika 3.2) predstavljaju ulaze i izlaze blok dijagrama i definišu tok podataka u virtuelnom instrumentu – od ulaznih ka izlaznim terminalima. Mogu korespondirati kontrolama (terminali ulaznog tipa) i indikatorima (terminali izlaznog tipa) postavljenim na front panel. U terminale spadaju i konstante – terminali ulaznog tipa, kao i labele, koje mogu imati funkcije ulaza i izlaza.

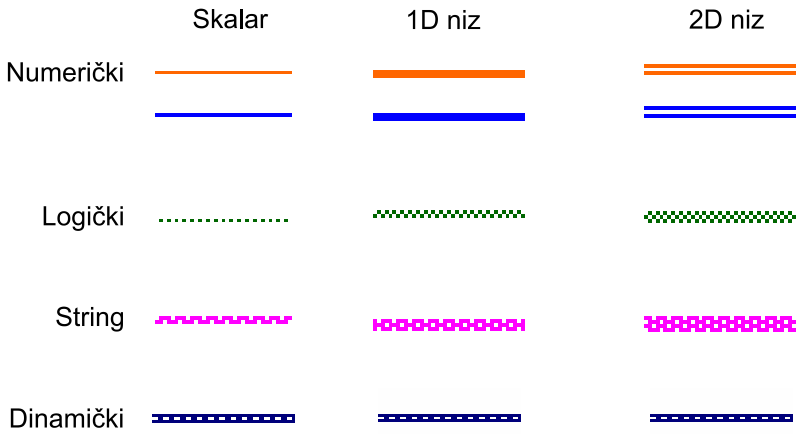
Slika 3.2 Terminali

Čvorovi blok dijagrama su elementi koji procesiraju podatke, analogni su deklaracijama i funkcijama tekstualnih programskih jezika. Čvorovi mogu biti elementarne funkcije ili drugi virtuelni instrumenti.

Funkcije su fundamentalni elementi u LabVIEW i predstavljaju osnovne gradivne blokove svakog virtuelnog instrumenta, koje je nemoguće razložiti.

Virtuelni instrumenti se mogu instancirati u drugim virtuelnim instrumentima, čime se postiže modularnost i preglednost grafičkog kôda i definiše hijerarhija u lancu virtuelnih instrumenata. Instancirani virtuelni instrumenti (SubVI) mogu biti standardni i tzv. Express VI, koji imaju definisanu funkciju i mogu se modifikovati kroz unapred definisan korisnički dijalog.

Elementi blok dijagrama se povezuju **vezama**. Boja i tekstura veze označavaju tip i dimenzionalnost podatka. Primer nekih tipova je dat na slici 3.3.



Slika 3.3 Tipovi podataka

Čvorovi blok dijagrama se izvršavaju zavisno od toka podataka – *data flow* koncept programiranja. Izvršavanje počinje kada su na svim ulaznim terminalima čvora dostupni podaci. Rezultati izvršavanja su dostupni na izlaznim terminalima nakon izvršenja čvora. Redosled izvršavanja i prosleđivanja podataka između čvorova se može pratiti aktiviranjem **Execution Highlighting** (odeljak 2.2.4) opcije sa palete sa alatima blok dijagram prozora.

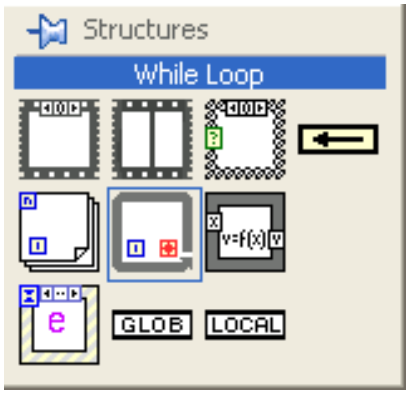
U cilju otklanjanja grešaka, izvršavanje virtuelnog instrumenta se može zaustaviti na određenom mestu pomoću prekida (**Breakpoints**). Prekid se postavlja otvaranjem kontekstnog menija (desni klik) veze i izborom opcije **Breakpoint/Set Breakpoint**. Izvršenje će biti zaustavljeno u tački prekida.

Vrednost podatka na vezi se može pratiti pomoću sonde (**Probe**). Sonda se može postaviti otvaranjem kontekstnog menija (desni klik) veze i izborom opcije **Probe**, čime se otvara prozor sa prikazom trenutne vrednosti podatka na vezi.

Uslovna sonda (**Conditional Probe**) predstavlja hibrid sonde i prekida. Postavlja se aktiviranjem kontekstnog menija (desni klik) veze i izborom opcije **Custom probe/Conditional Double Probe**. Uslovna sonda proverava podatak prema unapred zadatom kriterijumu i prekida izvršavanje ukoliko je ispunjen.

3.3 Programske petlje

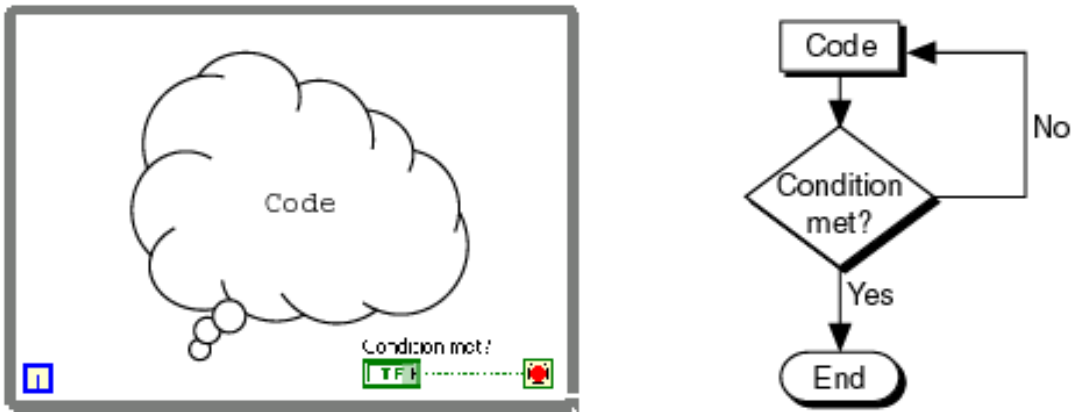
U realizaciji aplikacija je često neophodno određeni deo kôda izvršiti više puta. Standardni programski jezici imaju različite mehanizme kojima je ovakvo izvršavanje omogućeno – programske petlje. LabVIEW podržava nekoliko tipa programskih petlji: **While** petlja, **For** petlja, uslovna **For** petlja i **Timed loop** petlja. Programske petlje u LabVIEW okruženju predstavljaju skup programskih struktura, i mogu se postaviti na blok dijagram virtuelnog instrumenta iz palete **Programming/Structures** (slika 3.4).



Slika 3.4 Structures paleta

3.3.1 While petlja

While petlja je programska petlja koja izvršava sekvencijalno grafički kôd koji se nalazi unutar sve dok se ne ispuni neki logički uslov. Izgled strukture i odgovarajući algoritam su prikazani na slici 3.5.

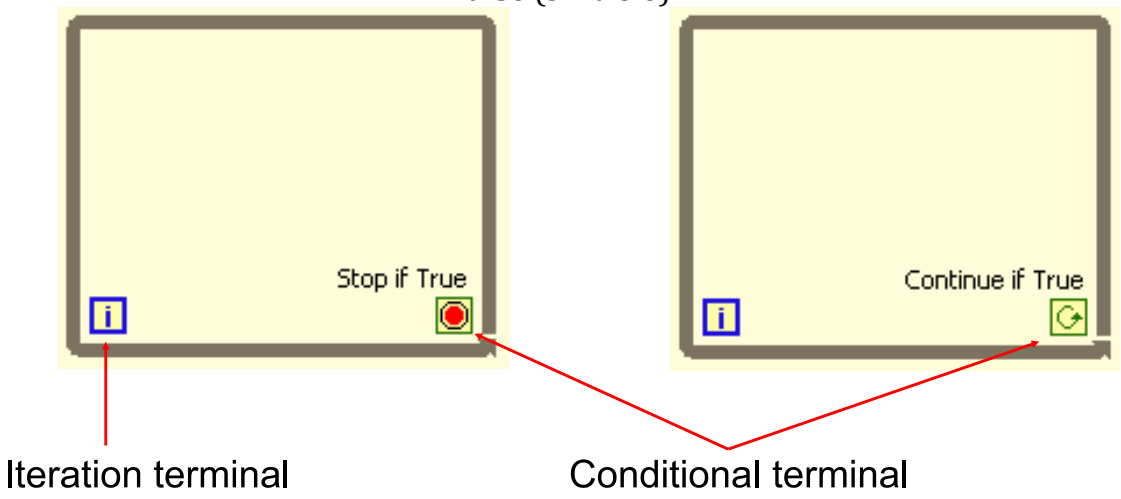


Slika 3.5 While loop struktura (levo) i algoritam (desno)

Elementi petlje su okvir, **iteration (i)** terminal i **condition** terminal. Okvir strukture određuje granice njenog domena, odnosno uokviruje kôd koji se izvršava. Iteration terminal je inicijalno postavljen sa leve strane strukture, predstavlja celobrojni numerički terminal izlaznog tipa čija vrednost se inkrementira nakon svakog izvršenja petlje. Inicijalna vrednost terminala je nula.

Na desnoj strani se nalazi condition terminal, ulaznog boolean (logičkog) tipa, kojim se kontroliše izvršavanje petlje. Podrazumevani način izvršavanja je **Stop if True**, koje zaustavlja petlju kada se na terminal dovede logička vrednost **True**. Način izvršavanja petlje se može promeniti otvaranjem kontekstnog menija terminala (desni klik na terminal) i izborom opcije **Continue if True**, čime petlja prekida izvršavanje kada se na terminal dovede logička vrednost

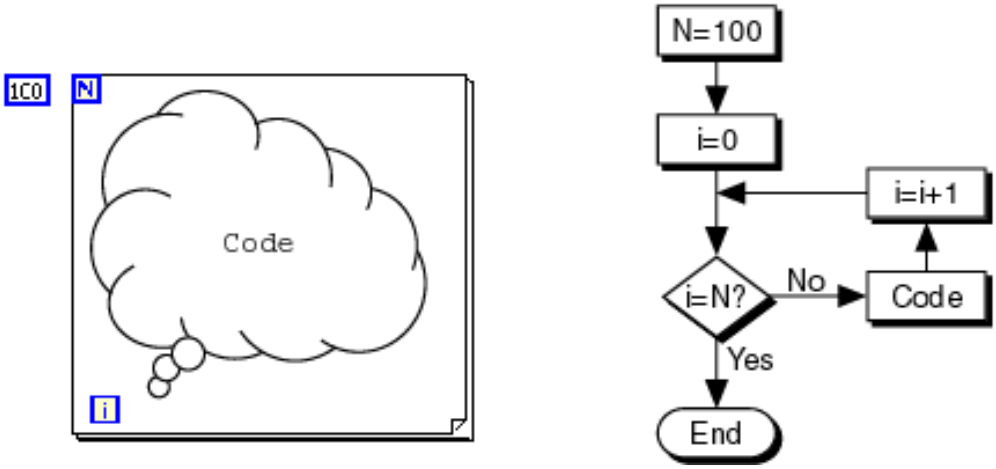
False (slika 3.6).



Slika 3.6 Načini izvršavanja petlje

3.3.2 For petlja

For petlja je programska petlja koja izvršava deo kôda unapred određen broj puta. Izgled strukture je prikazan na slici 3.7. Petlja se može postaviti na blok dijagram iz palete **Programming/Structures**.



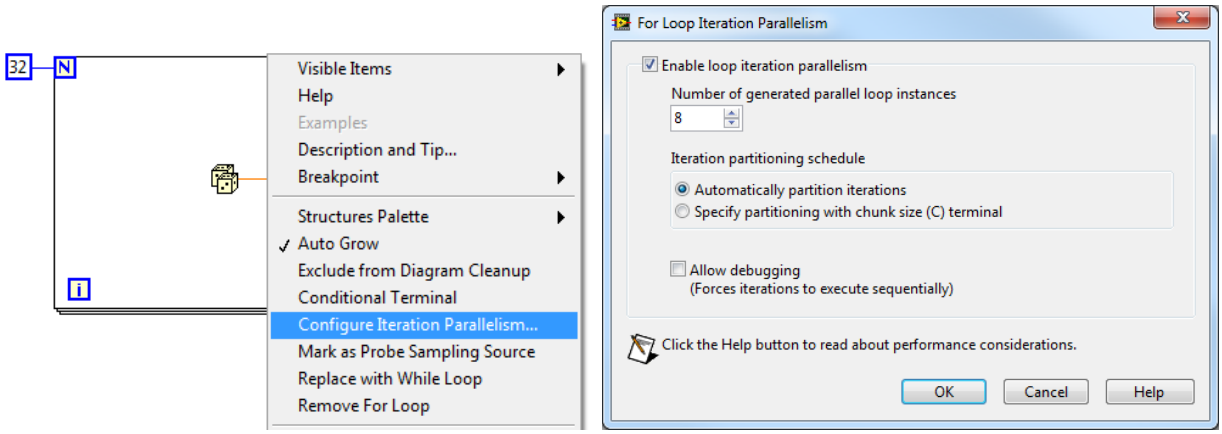
Slika 3.7 For petlja (levo) i algoritam (desno)

Elementi For petlje su okvir, **iteration (i)** terminal i **count (N)** terminal. Okvir petlje određuje granice njenog domena, odnosno uokviruje kôd koji se izvršava. Iteration terminal, kao i kod While petlje, je inicijalno postavljen sa leve strane strukture, predstavlja celobrojni numerički terminal izlaznog tipa čija vrednost se inkrementira nakon svakog izvršenja petlje. Inicijalna vrednost terminala je nula.

Count terminal se nalazi u gornjem levom uglu strukture. Terminal je ulazni, celobrojnog numeričkog tipa. Broj koji se dovodi na terminal određuje broj ponavljanja petlje.

For petlja se može konfigurisati za paralelno izvršavanje, pri čemu se kôd izvršava u različitim *thread*-ovima. Ovim se postižu bolje performanse, pogotovo na višeprocesorskim platformama. Prilikom korišćenja paralelizma, treba imati u vidu broj procesorskih jezgara kako bi se postigle optimalne performanse. Takođe treba uvažiti činjenicu da u većini slučajeva nije moguće paralelizovati kôd kod koga se u iteraciji petlje koriste podaci koji su rezultat izvršavanja prethodnih iteracija.

Konfiguracija paralelizma se ostvaruje pomoću dijaloga koji se pokreće iz kontekstnog menija For petlje, izborom opcije **Configure Iteration Parallelism**, slika 3.8.



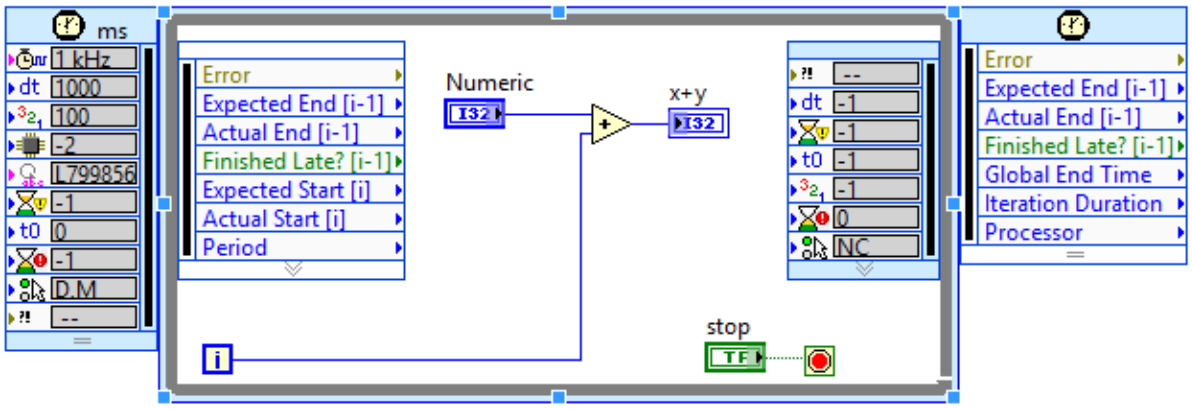
Slika 3.8 Kontekstni meni For petlje i dijalog za konfiguraciju

Nakon konfigurisanja i zatvranja dijaloga, ispod count terminala će biti dostupan **Parallelism (P)** terminal, koji se može programski menjati. Petlja će biti paralelizovana u onoliki broj thread-ova, koliki je minimum između broja dovedenog na parallelism terminal i broja unetog u dijalod (**Number of generated parallel loop instances**).

Paralelizam petlje je moguće precizno kontrolisati opcijom **Specify partition with chunk size (C) terminal**, čijim izborom postaje dostupan **Chunk (C)** terminal. Povezivanjem numeričkog niza na navedeni terminal precizno se može odrediti broj iteracija petlji koje izvršava svaki thread-a. Povezivanjem numeričke konstante na terminal, definiše se najmanji broj iteracija koje preuzima jedan thread.

3.3.3 Timed loop petlja

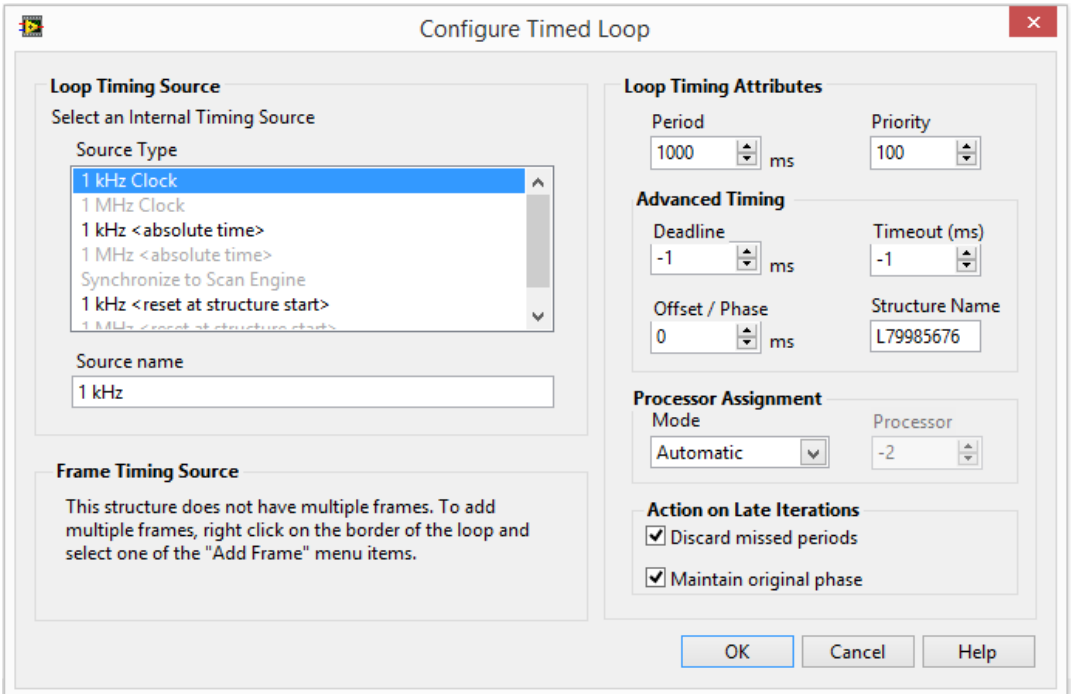
Vremenski definisane petlje **(Timed Loop)** omogućavaju precizno vremensko izvršavanje kôda, sinhronizaciju i multithreading, izbor CPU-a na kome će se thread izvršiti (slika 3.9).



Slika 3.9 Timed Loop petlja

Vremenski definisana petlja je slična While petlji, način izvršavanja kôda i algoritam su identični. Iteration i condition terminal imaju istu funkciju i ponašaju se na identičan način. Vremenski definisana petlja se može konfigurisati pomoću dijaloga koji se aktivira iz kontekstnog menija (slika 3.10). Prilikom konfiguracije, moguće je izabrati izvor sinhronizacije

(source type), pomoću koga se postiže sinhronizacija više različitih petlji u okviru virtuelnog instrumenta.



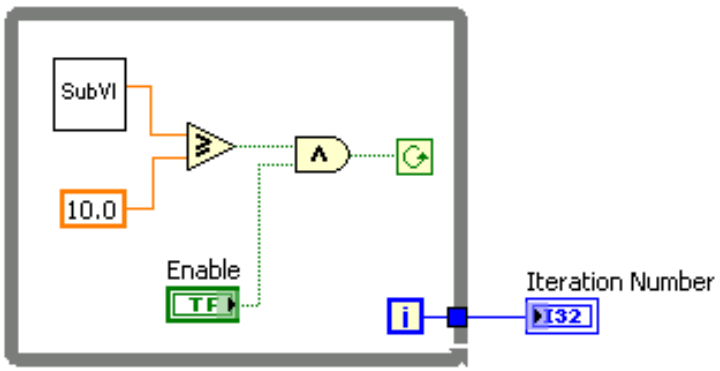
Slika 3.10 Dijalog za konfiguraciju vremenski definisane petlje

U dijalogu se zadaju parametri koji određuju period izvršavanja, kašnjenje u odnosu na izvor sinhronizacije (offset), vremensko prekoračenje (timeout), maksimalno vreme izvršavanja (deadline) i prioritet izvršavanja petlje. Dodeljivanje thread-ova može biti automatsko ili eksplicitno određeno. Svaka timed loop petlja ima jedinstveno ime.

Vremenski definisane petlje su neophodne kod procesa koji zahtevaju preciznu sinhronizaciju i vremenska ograničenja, kao kod *real-time* procesa.

3.3.4 Tuneli, šift registri i povratne petlje

Programske petlje obrađuju podatke koji su prethodno generisani ili obrađivani u aplikaciji, a rezultati obrade u petlji se prosleđuju aplikaciji nakon njenog izvršenja. Za prosleđivanje podataka unutar i van petlje služe **tuneli**. Tunel je prikazan kao kvadrat na ivici strukture, čija boja odgovara boji veze vezanoj za tunel, odnosno tipu podatka koji se prosleđuje (slika 3.11). Tunel će biti automatski generisan kada se veza dovede do ivice strukture.



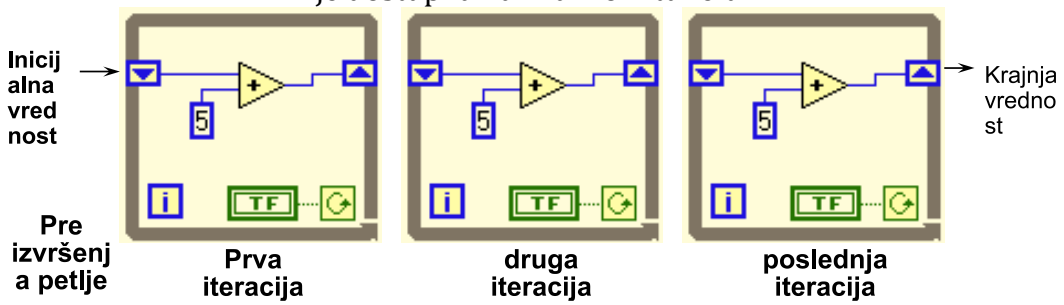
Slika 3.11 Tunel na izlazu While petlje

Tuneli mogu biti ulaznog i izlaznog tipa, u zavisnosti od toka podatka. Kod tunela ulaznog tipa, petlja počinje sa izvršavanjem tek nakon što je podatak postavljen na ulaz. Kod tunela izlaznog tipa, podatak je dostupan nakon izvršenja petlje.

Tuneli su univerzalan mehanizam razmene podataka između programskih struktura u LabVIEW grafičkom kôdu.

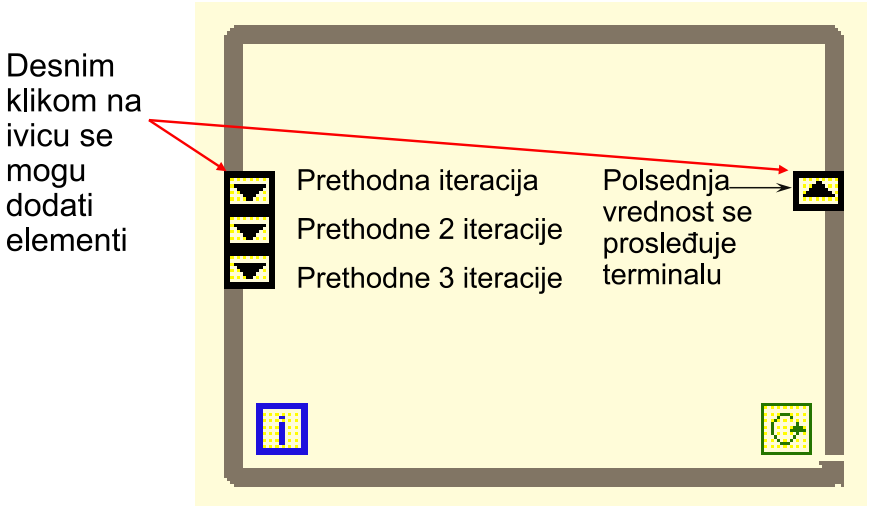
Ukoliko je u iteraciji programske petlje potrebno koristiti podatak dobijen obradom prethodne iteracije, koriste se šift registri (**Shift Register**) i povratne petlje (**Feedback Loop**).

Šift registar je ulazno/izlazni tunel koji omogućuje prenos podataka između iteracija petlje. Razmotrimo petlju koja u svakoj iteraciji dodaje konstantnu vrednost sumi, prikazanoj na slici 3.12. Šift registar je predstavljen kao kvadrat sa strelicom nadole (ulazni tunel), odnosno strelicom nagore (izlazni tunel). Boja šift registra odgovara tipu podatka, u konkretnom slučaju celobrojnom numeričkom tipu. Prilikom prve iteracije, vrednosti ulaznog tunela šift registra se dodaje konstantna vrednost (5), koja se prosleđuje ka izlaznom tunelu. U narednoj iteraciji se vrednosti prethodne iteracije dovedenoj sa ulaznog tunela dodaje konstantna vrednost i rezultat prosleđuje ka izlaznom tunelu. Ovaj proces se ponavlja dok logička vrednost dovedena na condition terminal ima vrednost True. Nakon završetka izvršavanja petlje, krajnja vrednost je dostupna na izlaznom tunelu.



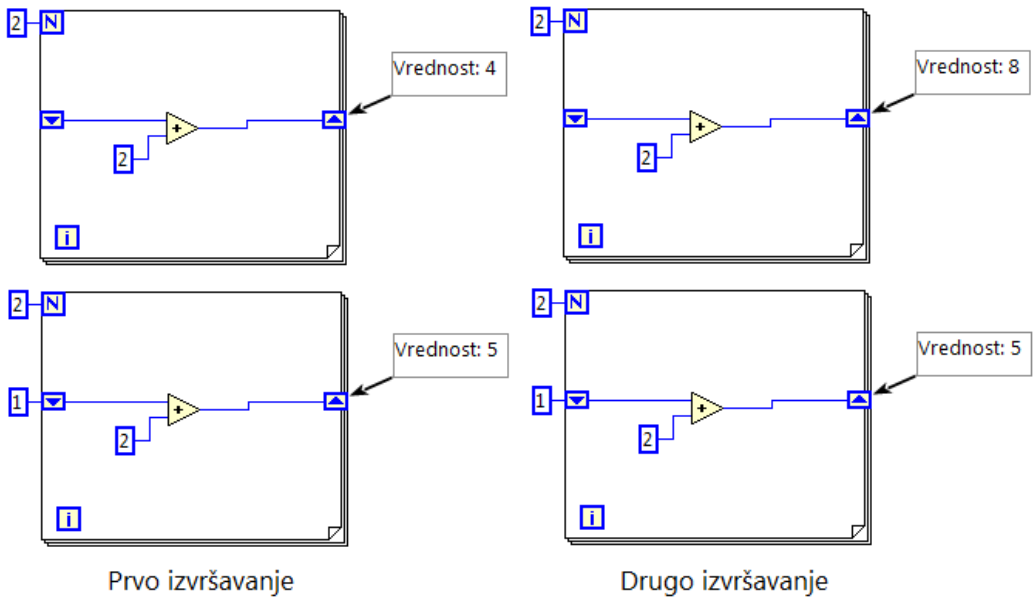
Slika 3.12 Petlja sa šift registrom

Šift registar se može postaviti opcijom **Add Shift Register** iz kontekstnog menija otvorenog na levoj ili desnoj ivici petlje. Ukoliko je registar već postavljen, pomoću kontekstnog menija šift registra se mogu dodati elementi koji predstavljaju vrednosti iz proizvoljnog broja prethodnih iteracija (slika 3.13).



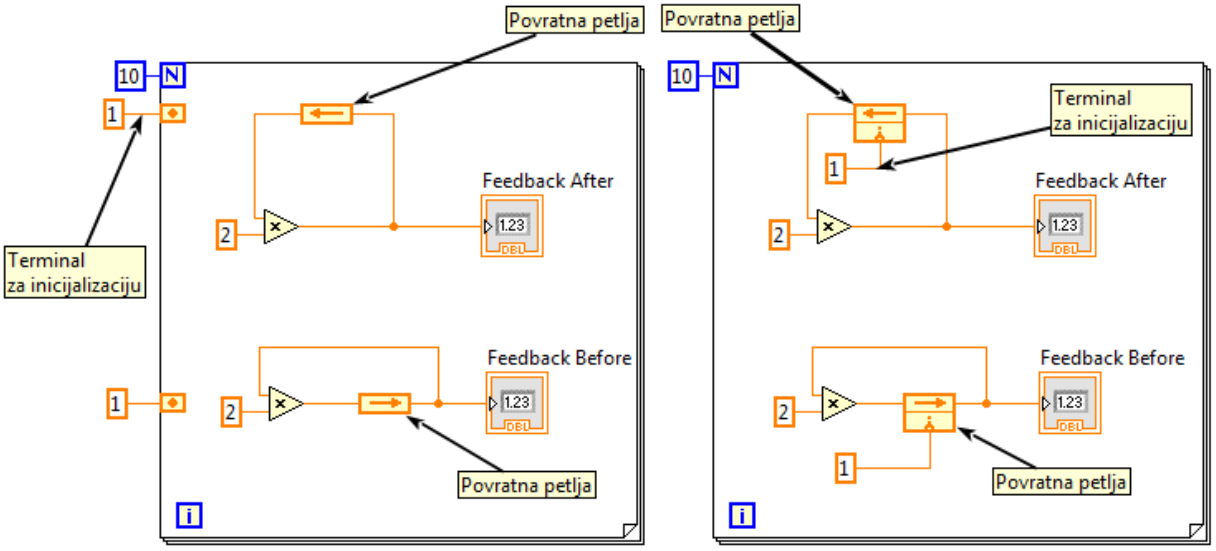
Slika 3.13 Dodavanje elemenata prethodnih petlji

Ukoliko na ulazni terminal nije vezan podatak, podrazumeva se vrednost prethodno izvršene petlje ili osnovna (default) vrednost podatka ukoliko petlja nije prethodno izvršavana (slika 3.14).



Slika 3.14 Inicijalne vrednosti šift registra

Povratne petlje će se automatski javiti ukoliko se u petlji poveže izlaz čvora ili grupe čvorova na ulaz istog čvora, odnosno grupe čvorova. Povratna petlja čuva vrednost podatka do okončanja izvršenja iteracije petlje, i prosleđuje podatke sledećoj iteraciji. Može da prosledi podatak bilo kog tipa, pri čemu boja povratne petlje zavisi od tipa podatka. Povratna petlja se može postaviti i iz palete **Programming/Structures**.



Slika 3.15 Primer povratne petlje

Primer povratne petlje prikazan je na slici 3.14. Obe petlje imaju identičnu funkciju. Sastavni deo petlje je terminal za inicijalizaciju, koji se može postaviti na ivicu strukture radi bolje preglednosti opcijom **Move Initializer One Loop Out**. Ovom opcijom je moguće precizno definisati u odnosu na koju programsku petlju povratna petlja čuva vrednosti. Vezivanjem

podatka za terminal za inicijalizaciju, određuje se početna vrednost povratne petlje. Ukoliko terminal nije vezan, podrazumeva se vrednost prethodno izvršene petlje ili osnovna (default) vrednost podatka ukoliko petlja nije prethodno izvršavana, slično kao kod šift registra.

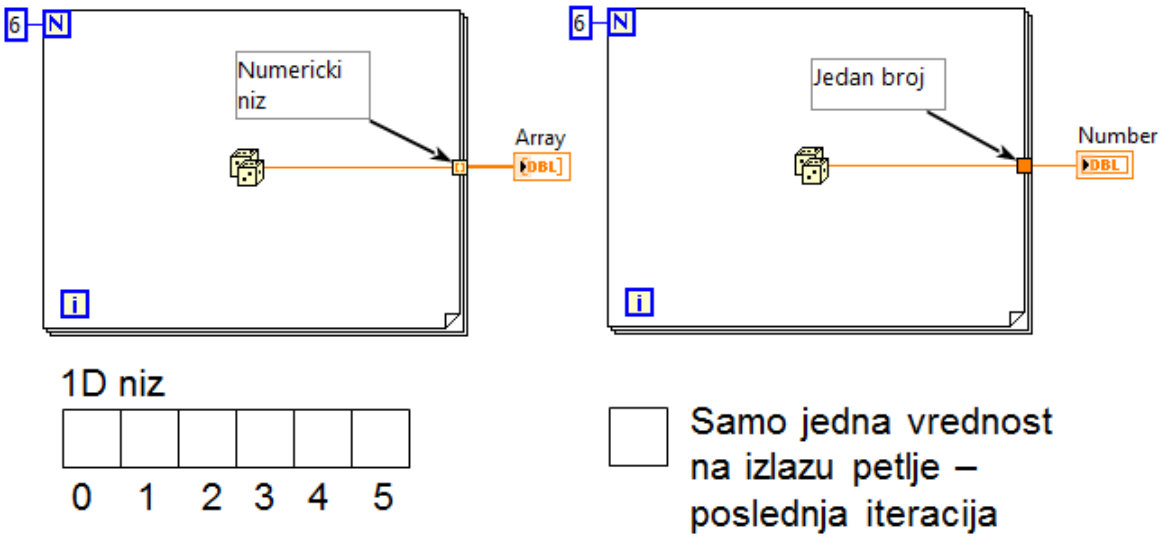
3.4 Nizovi

Niz je skup pobrojanih elemenata istog tipa. Određen je tipom podatka, brojem elemenata, dimenzionalnošću i samim elementima. Niz može biti jednodimenzionalan ili višedimenzionalan. Svakom elementu niza se pristupa preko indeksa, pri čemu prvi element niza ima indeks nula, poslednji element niza ima indeks N-1, gde je N broj elemenata niza. Tip elemenata niza je proizvoljan, a broj elemenata niza je ograničen memorijom.

U LabVIEW grafičkom jeziku, nizovi se tretiraju kao i ostali tipovi podataka, što implicira postojanje odgovarajućih kontrola, indikatora i konstanti nizova. Kontrole i indikatori nizova se mogu kreirati na front panelu izborom **Array** kontrole iz **Controls/Array, Matrix & Cluster** palete i postavljanjem elementa (kontrole ili indikatora) konkretnog tipa. Kreirani niz će biti istog tipa podatka kao i umetnuti element, kontrola ili indikator u zavisnosti od samog umetnutog elementa. Dimenzionalnost niza se može povećati opcijom **Add Dimension** iz kontekstnog menija. Na blok dijagramu je moguće realizovati konstantni niz izborom **Array Constant** iz **Programming/Array** palete i postavljanjem konstante elementa čiji tip odgovara željenom tipu niza.

3.4.1 Autoindeksiranje

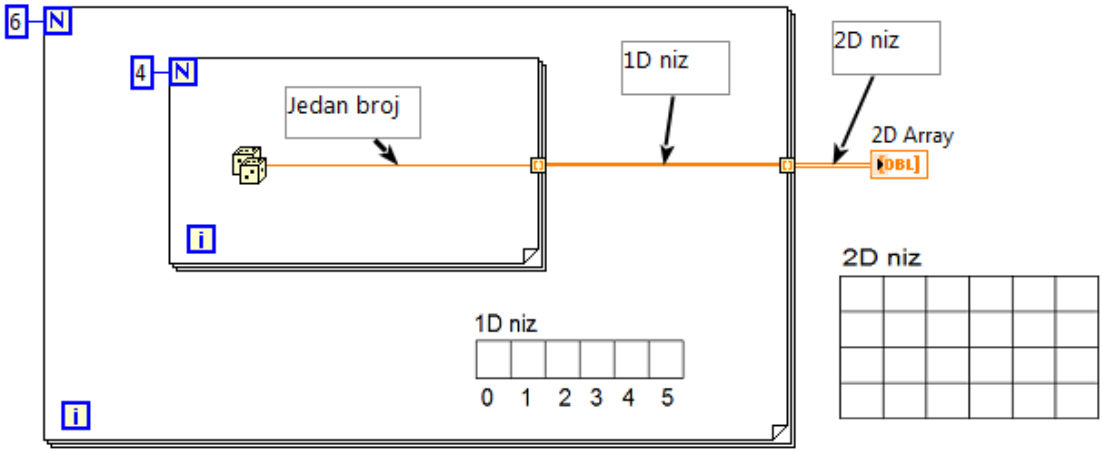
Za manipulaciju i obradu elemenata nizova se najčešće koriste programske petlje, mehanizmom koji se zove *autorindeksiranje*. Najjednostavniji primer autoindeksiranja je generisanje elemenata niza, prikazan na slici 3.16.



Slika 3.16 Autoindeksiranje

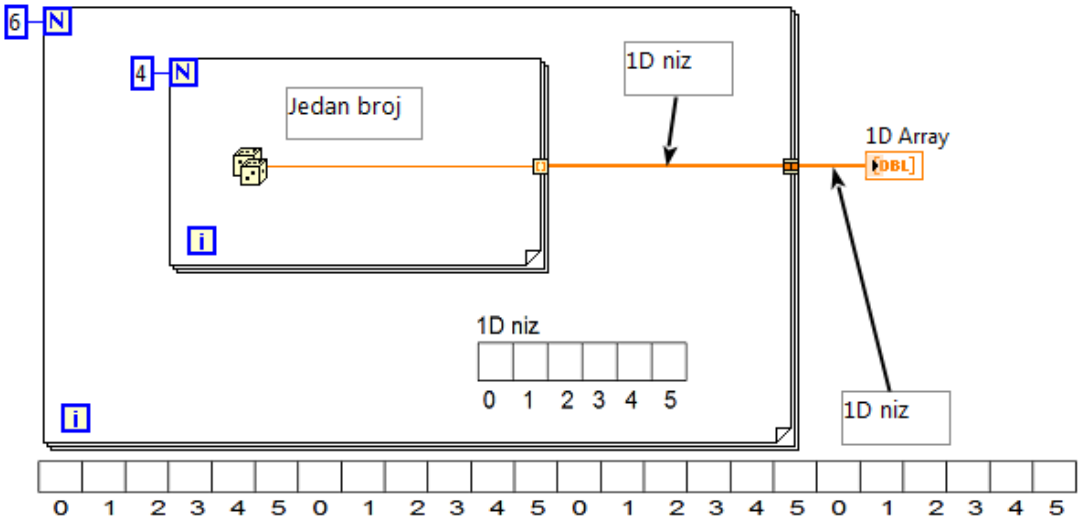
Kada se veza dovede do ivice For petlje, biće kreiran autoindeksni tunel, koji akumulira vrednosti iz svih iteracija u niz. U slučaju While petlje, biće kreiran klasičan tunel koji prosleđuje vrednost poslednje iteracije. Ponašanje tunela se može regulisati kontekstnim menijem, izborom podmenija **Tunnel Mode**.

Višedimenzionalni nizovi se mogu realizovati višestrukim petljama. Svaka petlja sa autoindeksnim tunelom dodaje jednu dimenziju nizu (slika 3.17).



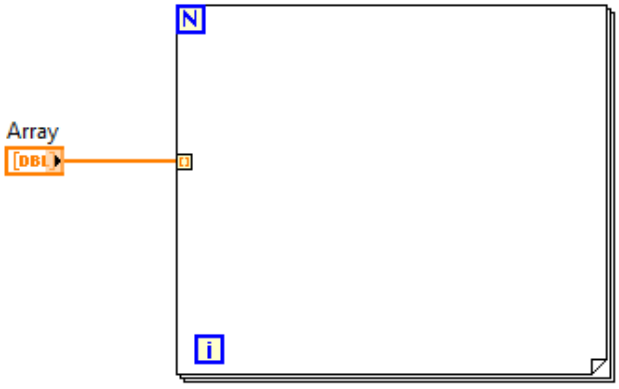
Slika 3.17 2D niz kreiran For petljama

Osim kreiranja višedimenzionalnih nizova, izborom opcije **Tunnel Mode/Concatenating** moguće je za svaku iteraciju spoljne petlje nastaviti jednodimenzionalni niz (slika 3.18).



Slika 3.18 Nastavljanje (konkatenacija) nizova

Mehanizam autoindeksiranja se može iskoristiti i kod obrade pojedinačnih elemenata niza, ukoliko se niz dovede na ulaz For petlje (slika 3.19).



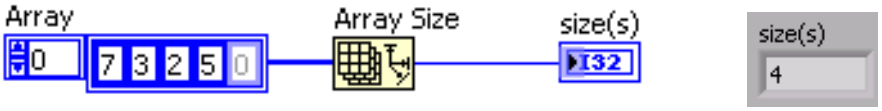
Slika 3.19 For petlja sa ulaznim nizom

U ovom slučaju, nije potrebno odrediti vrednost count terminala, već će se petlja izvršiti onoliko puta koliko ima elemenata niza.

3.4.2 Najčešće korišćene funkcije nizova

3.4.2.1 Array Size

Funkcija **Array Size** se poziva iz palete Programming/Array. Ulazni podatak je niz proizvoljnog tipa i dimenzionalnosti. Ukoliko je ulazni niz jednodimenzionalan, izlazni podatak je celobrojna numerička vrednost koja predstavlja broj elemenata niza (slika 3.20).



Slika 3.20 Array Size funkcija

U slučaju višedimenzionalnih nizova, izlazni podatak je numerički niz čije su vrednosti jednake veličini svake dimenzije niza. U tom slučaju, broj elemenata ulaznog niza je jednak proizvodu elemenata izlaznog numeričkog niza.

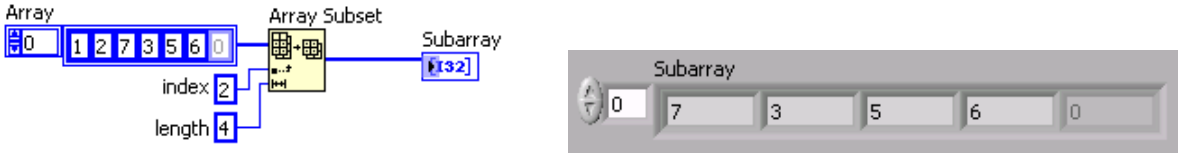
3.4.2.2 Initialize Array

Initialize Array funkcija kreira niz čiji su elementi jednaki. Ulazni parametri su vrednost elementa i broj elemenata (slika 3.21). Funkcija Initialize Array je *polimorfna*, tako da je moguće dodavanjem većeg broja ulaza (razvlačenjem funkcije na dole) kreirati višedimenzionalni niz. U ovom slučaju, osim vrednosti elementa, ulazne parametre čine veličine svake dimenzije niza.

Slika 3.21 Initialize Array funkcija

3.4.2.3 Array Subset

Array Subset funkcija izdvaja podniz iz datog niza (slika 3.22). Ulazni parametri su niz, indeks elementa od koga počinje izdvajanje i broj elementa koji se izdvajaju. Funkcija je polimorfna, tako da dovođenjem višedimenzionalnog niza na ulaz menja oblik i zahteva indeks elementa i broj elementa za svaku dimenziju.

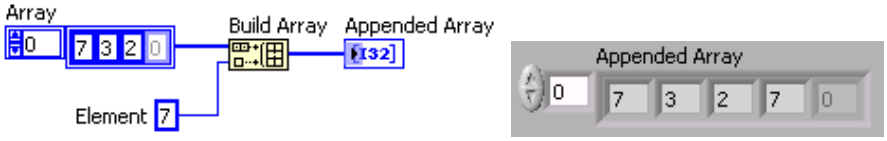


Slika 3.22 Array Subset

3.4.2.4 Build Array

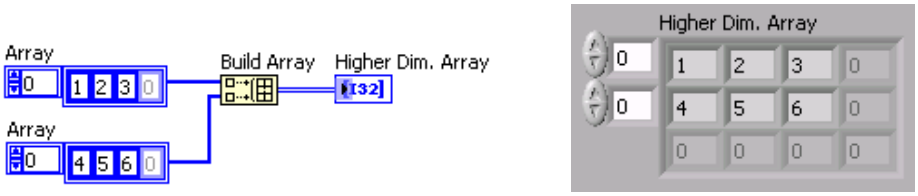
Build Array takođe spada u polimorfne funkcije. Ova funkcija može poslužiti za kreiranje niza od elemenata, dodavanje elemenata nizu, spajanje nizova ili dodavanje dimenzije nizu. Funkcija može prihvatiti proizvoljan broj ulaznih parametra, razvlačenjem funkcije na dole, koji mogu biti nizovi ili skalari (pojedinačni elementi).

Ukoliko su ulazni parametri niz i pojedinačni elementi, build array će kreirati niz koji se sastoji od elemenata niza i dodatih elemenata, sa redosledom elemenata koji odgovara redosledu ulaznih parametara odozgo nadole (slika 3.23). Ukoliko su ulazni parametri pojedinačne vrednosti, biće kreiran niz čiji će elementi odgovarati vrednostima parametara.



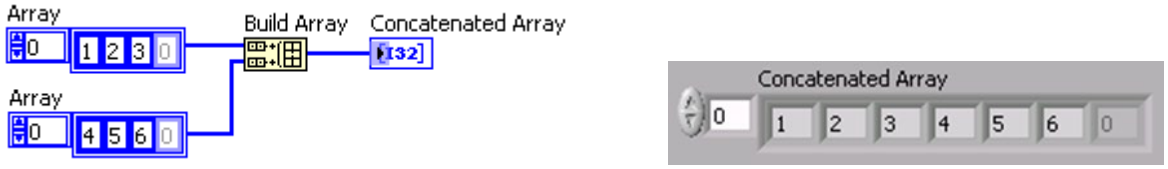
Slika 3.23 Dodavanje elemenata nizu

Ukoliko su parametri funkcije nizovi, Build Array će kreirati višedimenzionalni niz (slika 3.24). Dimenzionalnost izlaznog niza je jednaka broju ulaznih parametara, tj. nizova.



Slika 3.24 Dodavanje dimenzije nizu

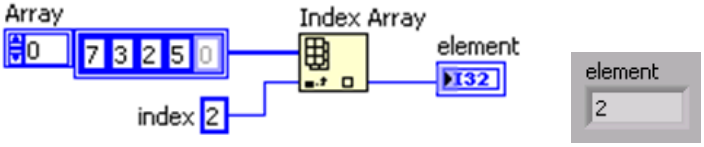
Ponašanje Build Array funkcije se može promeniti izborom opcije **Concatenate Inputs** iz kontekstnog menija, pri čemu će na izlazu biti kreiran niz iste dimenzionalnosti, sačinjen od elemenata ulaznih nizova prema redosledu odozgo nadole (slika 3.25).



Slika 3.25 Spajanje nizova

3.4.2.5 Index Array

Index Array funkcija služi za ekstrakciju elemenata iz niza. Ulazni parametri funkcije su niz i indeks elementa koji se izdvaja (slika 3.26).

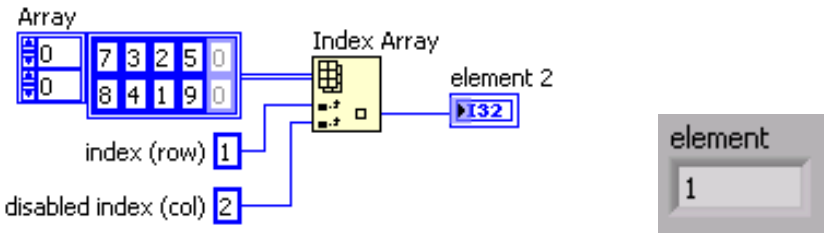


Slika 3.26 Izdvajanje elementa iz jednodimenzionalnog niza

Ukoliko je niz višedimenzionalan, moguće je izdvojiti jednu dimenziju (red) iz niza (slika 3.27).

Slika 3.27 Izdvajanje reda

Prilikom izdvajanja elementa kod višedimenzionalnog niza, ulazne parametre čine indeksi elementa koji se izdvaja po svim dimenzijama ().

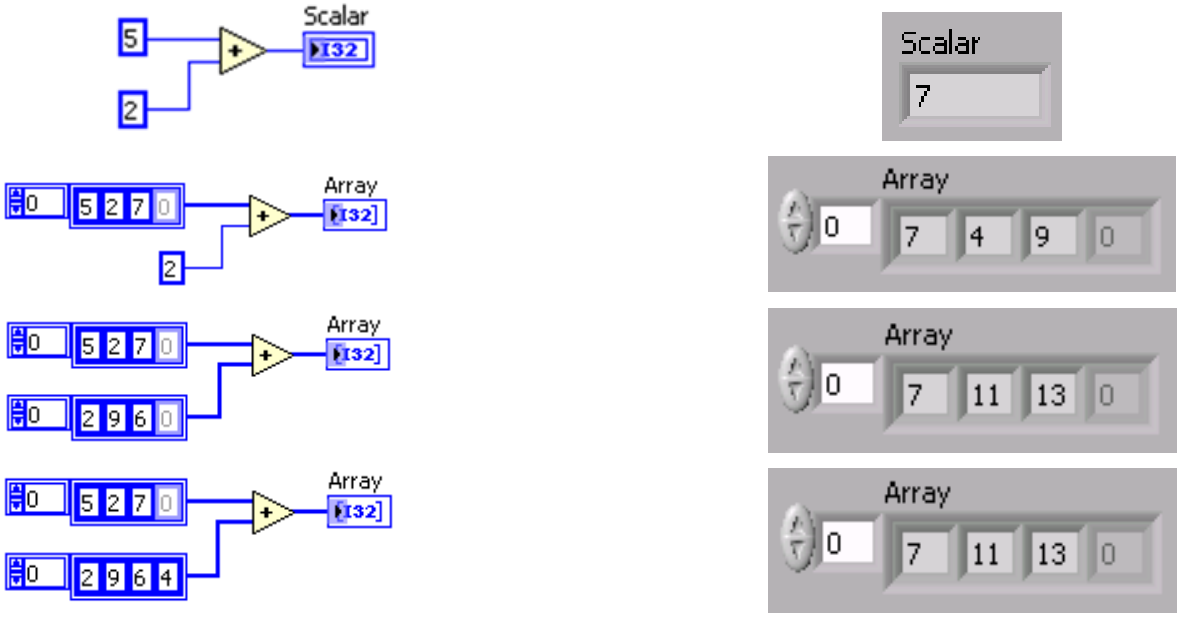


Slika 3.28 Izdvajanje elementa iz dvodimenzionalnog niza

Funkcija indeks array se može proširiti razvlačenjem nadole, pri čemu je moguće izdvojiti više od jednog elementa niza. Ukoliko se dovedu ulazni parametri za indekse elemenata koji se izdvajaju, biće izdvojeni elementi po redosledu od poslednjeg navedenog indeksa.

3.4.3 Polimorfizam

Većina elementarnih funkcija u LabVIEW kôdu je polimorfna, tj. prihvata različite tipove ulaznih podataka. Najznačajnije polimorfne funkcije su aritmetičke funkcije, koje kao ulazne parametre mogu prihvatiti različite numeričke tipove, kao i nizove numeričkih vrednosti. Primer funkcije sabiranja je dat na slici 3.29.



Slika 3.29 Funkcija sabiranja sa različitim ulaznim parametrima: sabiranje dva skalara, niza i skalara, dva niza istog broja elemenata i dva niza različitog broja elemenata

3.5 Programske strukture

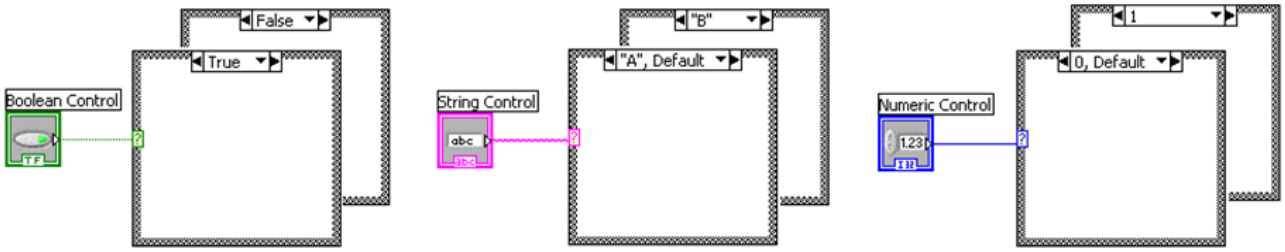
Programske strukture služe za kontrolu toka aplikacije, programska grananja i precizno definisanje redosleda izvršavanja pojedinih funkcija, odnosno delova kôda.

3.5.1 Case struktura

Case struktura omogućava izvršavanje kôda u zavisnosti od nekog logičkog uslova. U tekstualnim programskim jezicima sličnu namenu imaju Case i If dekleracije. Struktura se može postaviti na blok dijagram izborom iz **Programming/Structures** palete. Slično kao kod programskih petlji, Case strukturom se može obuhvatiti postojeći kôd. Case struktura se sastoji od okvira, selektorskog terminala i labele. Okvirom strukture je oivičen kôd raspoređen u *subdijagramima*, kojih može biti dva ili više. Subdijagrami su poređani kao špil karata, tako da se u jednom trenutku vidi samo jedan. *Labela subdijagrama* označava logički uslov pod kojim

će prikazani subdijagram biti izvršen. Klikom na strelice na vrhu strukture bira se prethodni ili naredni subdijagram, respektivno. Samo se jedan subdijagram izvršava, zavisno od logičke, numeričke ili znakovne vrednosti, koja je povezana za *selektorski terminal* (označen sa ?).

Ukoliko je **Boolean** vrednost povezana za selektorski terminal struktura ima dva subdijagrama: **FALSE** i **TRUE** (slika 3.30).



Slika 3.30 Case strukture sa logičkom (Boolean), znakovnom i numeričkom selektorskom varijablom

U slučaju da je numerički ili znakovni tip podatka povezan na selektorski terminal, struktura može imati beskonačan[1] broj subdijagrama. Prilikom postavljanja strukture, dostupna su samo dva subdijagrama, ali se taj broj može povećati dodavanjem novih. Opcijama **Add Case After** i **Add Case Before** iz kontekstnog menija strukture dodajemo nove subdijagrame. Opcijom **Duplicate Case** se kopira, a opcijom **Delete Case** se briše trenutno aktivan subdijagram.

Moguće je odrediti više od jedne vrednosti dodavanjem varijabli u labelu subdijagrama, pri čemu će se on izvršiti ukoliko je bar jedna dovedena na selektorski terminal. Case struktura mora imati definisan subdijagram koji će se izvršiti ukoliko se na selektorski terminal dovede vrednost koja nije obuhvaćena nijednim drugim subdijagramom. Ovaj subdijagram je označen kao **Default** subdijagram. Opcijom **Make This the Default Case** iz kontekstnog menija bira se Default subdijagram. Ukoliko je selektorska varijabla numerička, labela subdijagrama može biti

- jedan celi broj (0 ili 1, kao na slici 3.30),
- lista brojeva odvojenih zarezom (na primer 1,2,5,15,35),
- opseg brojeva (**10..20** – opseg od 10 do 20, **..10** – brojevi manji ili jednaki 10, **10..** – brojevi veći ili jednaki 10).

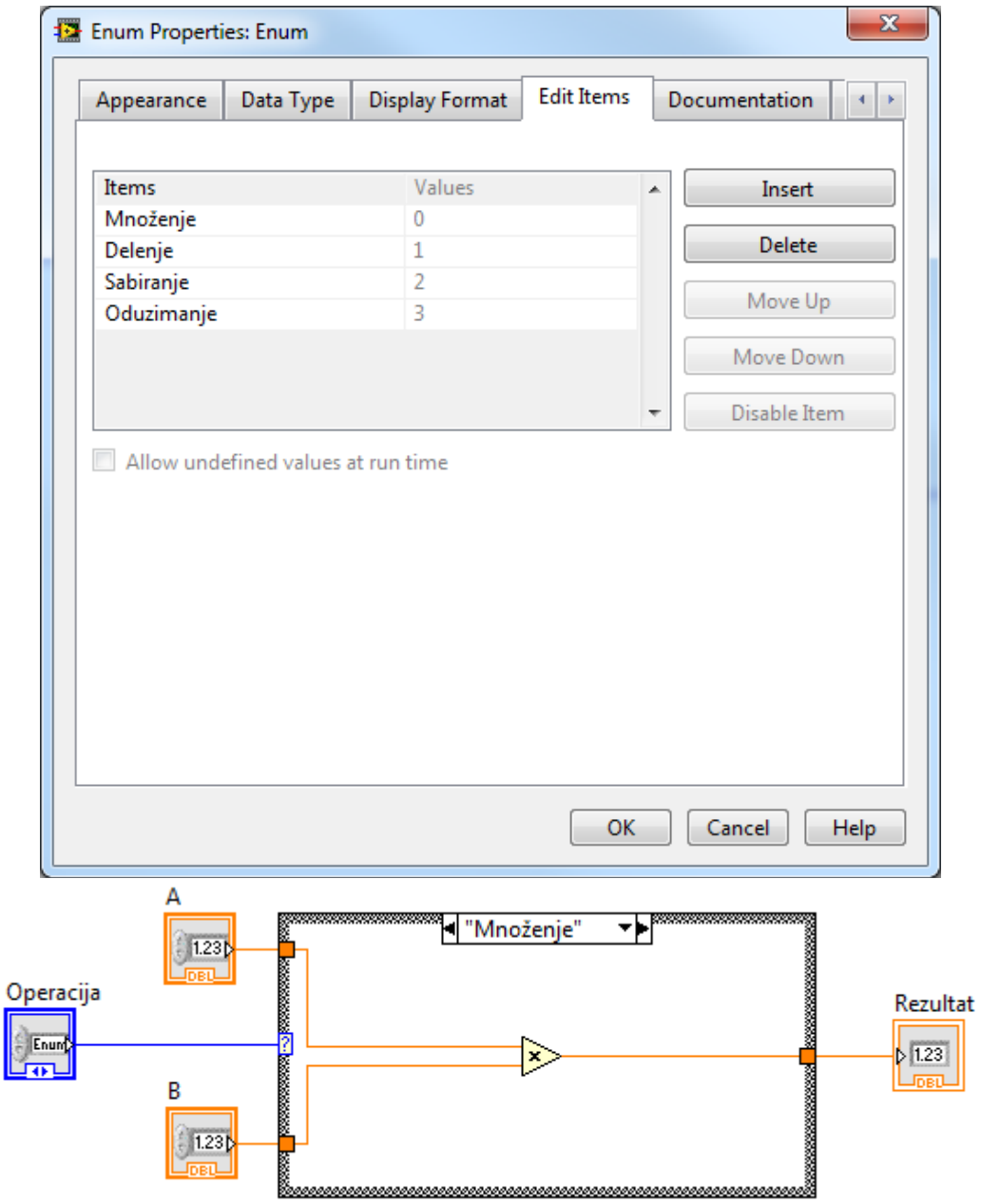
Labele subdijagrama moraju biti celobrojne.

3.5.1.1 Enumeracije i prstenovi

Selektorske varijable mogu biti i *enumeracije*. Enumeracije (Enum) su tip celobrojnih promenljivih u određenom, konačnom opsegu. Mogu se dodati kao kontrole na front panel iz palete **Ring & Enum**, ili na blok dijagramu kao konstanta izborom **Programming /Numeric/Enum Constant.** Svakoj vrednosti se dodeljuje celi broj sekvencijalno. Članovi enumeracije se mogu dodati u **Properties/Edit Items** dijalogu (slika 3.31). Prilikom vezivanja enumeracije za selektorski terminal Case strukture, dobija se mogućnost selekcije subdijagrama na osnovu članova enumeracije.

[1] Ovo ne treba shvatiti doslovno. Broj subdijagrama je ograničen opsegom koji određeni tip podataka ima i pre svega, memorijom.

Prsten (**Ring**) je tip podataka sličan enumeraciji, s tom razlikom što je moguće imati različite (nesekvencijalne) vrednosti asocirane članovima. Prsten imati necelobrojnu vrednost ili znakovni podatak.



Slika 3.31 Enumeracija i Case struktura kontrolisana enumeracijom

3.5.1.2 Tuneli

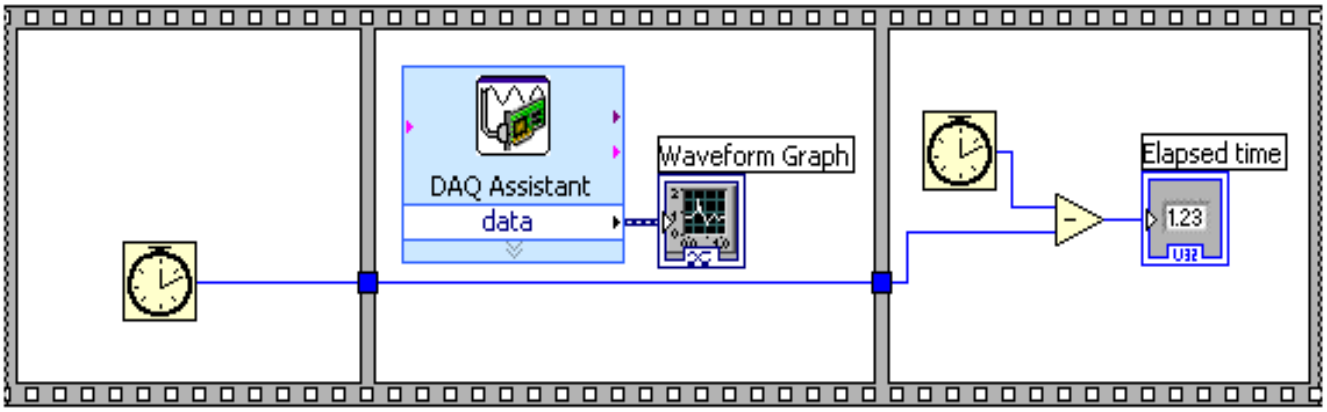
Tuneli kod Case strukture se ponašaju slično kao kod petlji. Pojaviće se automatski ukoliko se dovede veza do ivice strukture. Ukoliko je tok podataka od spolja ka strukturi, tunel je ulaznog tipa. Podaci na svim ulaznim terminalima Case strukture (tuneli i selektorski terminal) su dostupni za sve subdijagramne. Nije neophodno da svi subdijagrami koriste ulazne podatke. Ukoliko je tok podataka iz strukture ka spolja, tunel je izlaznog tipa. Ukoliko bar jedan subdijagram daje vrednost izlaznom tunelu, i ostali moraju dati izlaznu vrednost. Kada povežemo izlaz u jednom subdijagramu, tunel se pojavljuje na istoj lokaciji u svim slučajevima u obliku kvadrata sa belom unutrašnjošću. **Broken Run** dugme će biti prisutno dok se izlazni

tunel ne poveže u svakom subdijagramu. Desnim klikom na tunel se može odrediti podrazumevana vrednost opcijom **Use Default If Unwired**, čime se eliminiše potreba za povezivanjem u svakom subdijagramu.

3.5.2 Flat Sequence struktura

U tekstualnim programskim jezicima redosled dekleracija određuje redosled izvršavanja kôda. U LabVIEW, redosled nije toliko trivijalan. **Flat Sequence** strukture se mogu iskoristiti za precizniju kontrolu toka programa. Kôd se može rasporediti u specificirane sekvence i time odrediti redosled izvršavanja. **Flat Sequence** struktura se može postaviti izborom iz **Programming/Structures** palete.

Prilikom postavljanja, struktura ima jedan subdijagram – *frejm*. Kôd koji se izvršava sekvencijalno je raspoređen po subdijagramima koji se nazivaju *frejmovi*. Cela struktura liči na filmsku traku, gde se frejmovi prikazuju jedan za drugim (slika 3.32). Izvršavanje frejmova je s leva na desno.



Slika 3.32 Flat Sequence struktura sa tri frejma

Frejmovi se mogu dodavati Sequence strukturi desnim klikom na ivicu strukture i izborom opcije **Add Frame Before/After**. Frejm se može izbrisati desnim klikom na ivicu frejma i izborom opcije **Delete This Frame**.

Virtuelni instrument prikazan na slici 3.32 koristi **Tick Count** funkciju, koja vraća broj milisekundi očitan sa internog tajmera. Interni tajmer se resetuje prilikom butovanja računara. Prethodni primer može poslužiti za određivanje potrebnog vremena za izvršavanje nekog kôda, u konkretnom primeru meri se vreme akvizicije neke veličine. **Tick Count** funkcija se nalazi u paleti **Programming/Timing**.

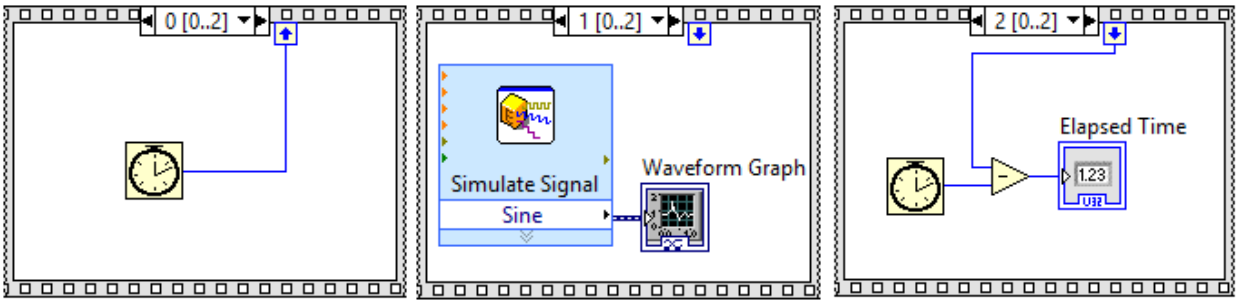
3.5.3 Stacked Sequence struktura

Stacked Sequence strukture se mogu iskoristiti za precizniju kontrolu toka programa. Kôd se može rasporediti u specificirane subdijagrame i time odrediti redosled izvršavanja. **Stacked Sequence** struktura se može postaviti izborom iz **Programming/Structures**. Prilikom postavljanja, struktura ima jedan subdijagram.

Stacked Sequence struktura ima sličnu funkciju kao i **Flat Sequence** struktura, pri čemu su subdijagrami postavljeni kao kod **Case** strukture, jedan iznad drugog. Kôd koji se izvršava sekvencijalno je raspoređen po subdijagramima. Cela struktura liči Case strukturu, pri čemu se izvršavaju svi subdijagrami prema redosledu. Na vrhu strukture se nalazi labela koja označava redni broj subdijagrama koji je prikazan, kao i ukupan broj subdijagrama. Samo je jedan subdijagram vidljiv, može se izabrati iz padajućeg menija sa vrha strukture.

Subdijagrami se mogu dodavati strukturi izborom opcije **Add Frame Before/After** iz kontekstnog menija strukture. Subdijagram se može izbrisati izborom opcije **Delete This Frame**. Subdijagram se može iskopirati u novi izborom opcije **Duplicate Frame**.

Podaci se prosleđuju preko posebnih tunela, takozvanih lokala, čija strelica ukazuje da li podaci ulaze ili izlaze iz subdijagrama. Lokal se može dodati izborom opcije **Add Sequence Local** iz kontekstnog menija strukture. Strelica lokala prikazuje smer toka podataka, i određuje se vezivanjem za ulaz/izlaz funkcija, kontrola ili indikatora unutar subdijagrama. Lokal koji je vezan kao izlazni lokal nekog subdijagrama, dostupan je kao ulazni lokal samo subdijagramima koji se kasnije izvršavaju.



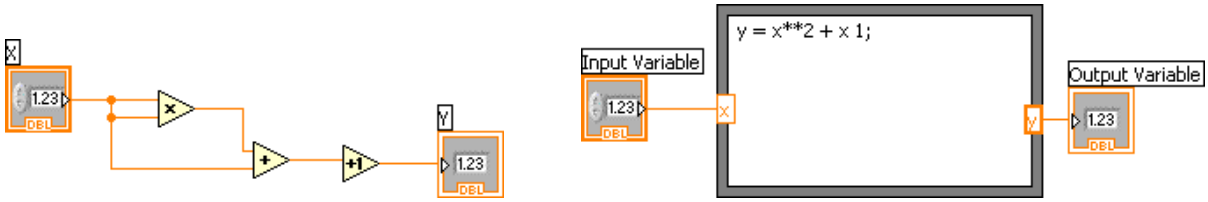
Slika 3.33 Stacked Sequence struktura. Na gornjoj ivici strukture se vide lokali celobrojnog numeričkog tipa

Na slici 3.33 je prikazan virtuelni instrument realizovan pomoću **Stacked Sequence** strukture sa identičnom funkcijom kao virtuelni instrument na slici 3.32.

3.5.4 Formula Node struktura

Formula Node struktura služi za iplementiranje komplikovanijih aritmetičkih formula, pri čemu se dobija na preglednosti. Struktura se nalazi u **Programming/Structures** paleti. Promenljive se kreiraju na ivicama strukture. Promenljive su *case-sensitive*. Svaka dekleracija se završava tačka-zarezom (;). Formula Node raspolaže velikim skupom funkcija, koje se mogu pronaći u LabVIEW Help-u (odeljak 2.3.1).

Primer virtuelnog instrumenta realizovanog elementarnim funkcijama i Formula Node strukturom je prikazana na slici 3.34.



Slika 3.34 Virtuelni instrument realizovan elementarnim funkcijama i Formula Node strukturom

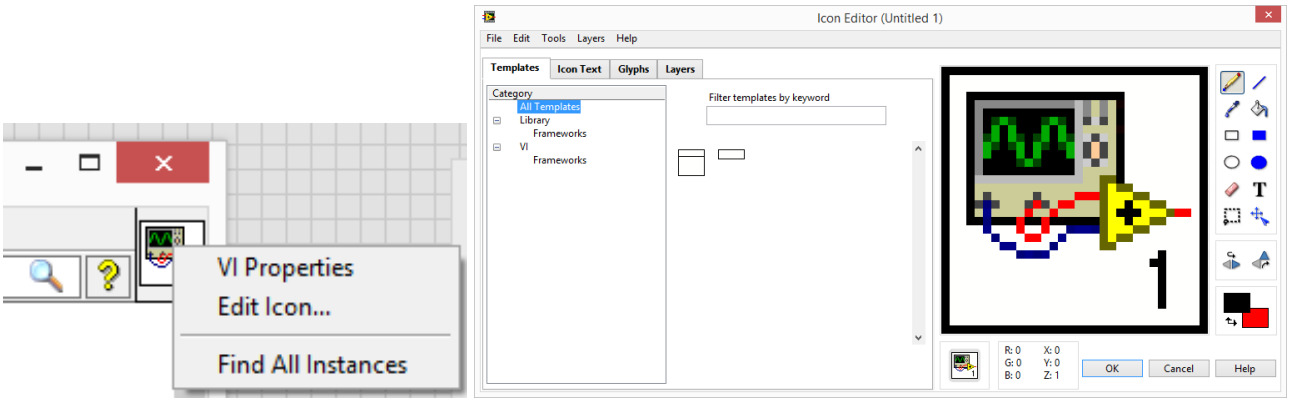
3.6 Modularno programiranje

Prilikom realizacije virtuelnog instrumenta, često je neophodno odgovarajuću funkciju upotrebiti u različitim delovima grafičkog kôda. Jednostavno kopiranje kôda koji predstavlja funkciju bi dovelo do nepreglednog i konfuznog blok dijagrama, kao i veće mogućnosti da se načini greška. U ovom slučaju, izmena funkcije zahteva promenu svake njene instance, što povećava vreme za razvoj i mogućnost da se načini greška. U svim programskim jezicima, ovaj problem je rešen postojanjem potprograma, procedura i funkcija. Sličan koncept postoji i u LabVIEW grafičkom programskom jeziku i naziva se **subVI**.

Kao što je već naglašeno u odeljku 3.2, čvorovi blok dijagrama mogu biti elementarne funkcije i drugi virtuelni instrumenti – **subVI**. Većina subVI ima otvoren kôd i dvostrukim klikom na ikonu na blok dijagramu se mogu otvoriti u posebnom prozoru. U kôd subVI virtuelnog instrumenta mogu biti uključeni drugi subVI virtuelni instrumenti, čime je definisana njihova hijerarhija. Svaki subVI je virtuelni instrument koji ima svoj front panel, blok dijagram, ikonu i konektorski panel.

Ikona virtuelnog instrumenta predstavlja njegovu reprezentaciju u drugim virtuelnim instrumentima. Njena funkcija je slična *prototipu* ili *deklaraciji* funkcije u tekstualnim programskim jezicima.

Svaki virtuelni instrument ima ikonu koja se može videti u gornjem desnom uglu prozora front panela ili blok dijagrama, u ravni sa menijem i paletom sa alatima. Prilikom kreiranja novog virtuelnog instrumenta, biće određena osnovna ikona koja se može izmeniti. Izborom opcije **Edit Icon** iz kontekstnog menija ikone ili dvostrukim klikom na ikonu (aktivna i na front panelu i blok dijagramu) se otvara **Icon Editor** (slika 3.35). Editor ikona sadrži alate za elementarnu grafičku obradu i omogućava izmenu grafičkog izgleda ikone virtuelnog instrumenta.

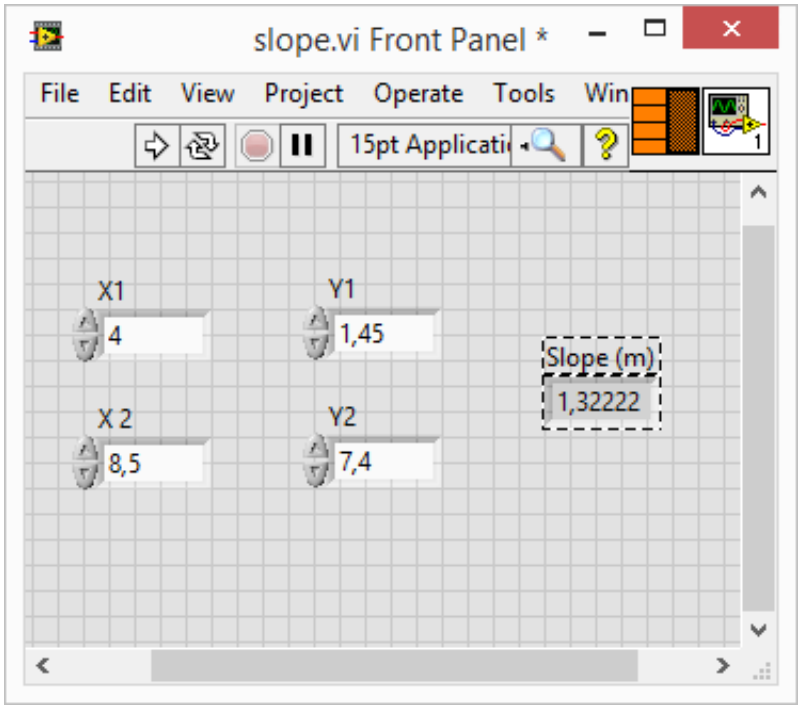


Slika 3.35 Icon Editor

Konektorski panel je skup terminala virtuelnog instrumenta, koji definišu ulazne parametre i rezultate izvršavanja. Predstavlja svojevrstan interfejs između subVI i virtuelnog instrumenta u koji je umetnut, koji služi za razmenu podataka.

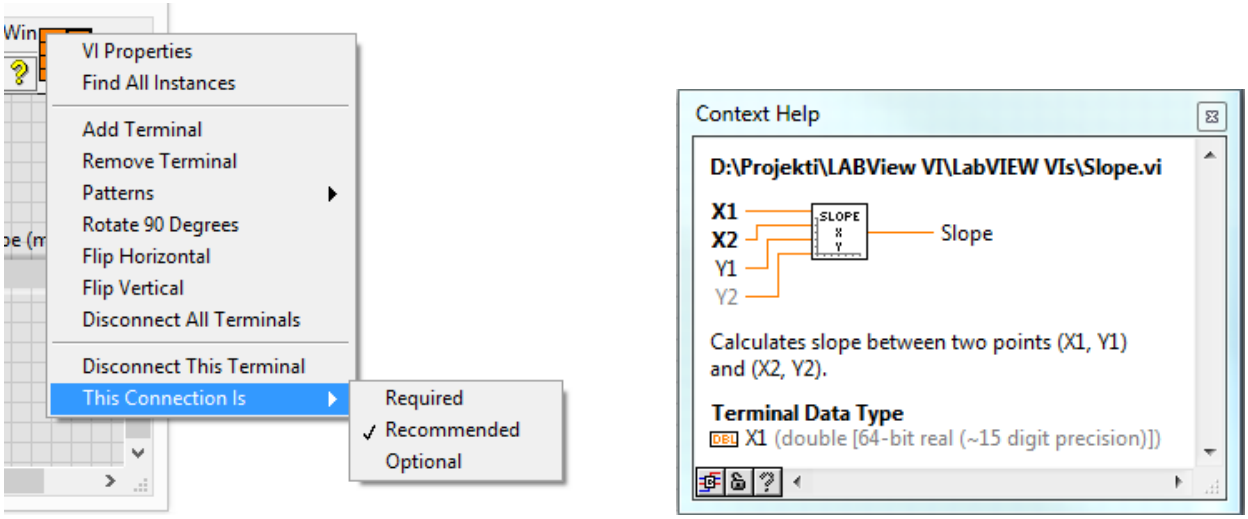
Panel se nalazi levo od ikone virtuelnog instrumenta na prozoru front panela. Broj polja na panelu i njihov raspored se mogu menjati izborom opcije **Patterns** iz kontekstnog menija konektorskog panela. Opcijama **Add Terminal/Remove Terminal** se može menjati broj polja, a opcijama **Rotate 90 Degrees**, **Flip Horizontal** i **Flip Vertical** njihova orijentacija. Kursor doveden na konektorski panel dobija oblik kalema žice – **connect wire tool**. Sukcesivnim klikom na polje koje se nalazi unutar konektorskog panela i kontrolu ili indikator na front panelu, definiše se ulazni, odnosno izlazni terminal, respektivno. Terminal će imati boju tipa kontrole ili indikatora za koje je vezan (slika 3.36).

Opcijom **Disconnect This Terminal** se raskida veza između polja u konektorskom panelu i kontrole, odnosno indikatora. Opcijom **Disconnect This Terminal** se raskidaju sve veze. Selektovanjem terminala u panelu selektuje se kontrola/indikator koji je za njega vezan.



Slika 3.36 Konektorski panel i kontrole/indikatori vezani za terminale

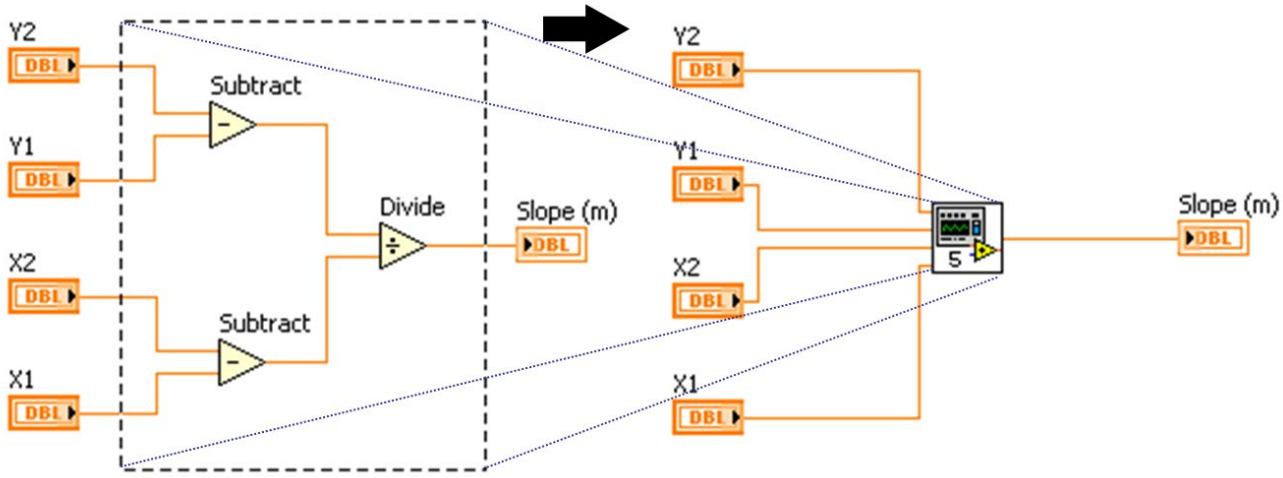
Izborom opcije **This Connection** Is iz kontekstnog menija konektorskog panela (slika 3.37) određuje se tip terminala: obavezan (**Required**), preporučen (**Recommended**) i opcion (**Optional**). Obavezni terminali se moraju povezati, u suprotnom virtuelni instrument neće biti moguće pokrenuti, greška će biti signalizirana **Broken Run** dugmetom. Preporučeni terminali se ne moraju vezati, u tom slučaju će biti signalizirano upozorenje (**Warning**) prilikom izvršavanja virtuelnog instrumenta u koji je inkorporiran subVI. Opcioni terminali se ne moraju vezivati i prilikom izvršavanja virtuelnog instrumenta višeg u hijerarhiji neće biti signalizirano upozorenje.



Slika 3.37 Kontekstni meni konektorskog panela sa izborom tipa terminala i kontekstni help SubVI sa objašnjenjem i tipovima terminala

SubVI se može postaviti u drugi virtuelni instrument opcijom **All Functions/Select a VI**, pri čemu se otvara dijalog sa stablom direktorijuma. Drugi način je drag & drop tehnikom, pri čemu se ikona virtuelnog instrumenta koji se importuje prevlači na blok dijagram ciljnog virtuelnog instrumenta.

Ukoliko je potrebno da se određeni deo kôda virtuelnog instrumenta instancira na više mesta na blok dijagramu, najefikasniji način je da se snimi kao poseban subVI. Selektovanjem kôda i izborom opcije **Create SubVI** iz **Edit** menija, od selektovanog kôda će biti kreiran subVI (slika 3.38).



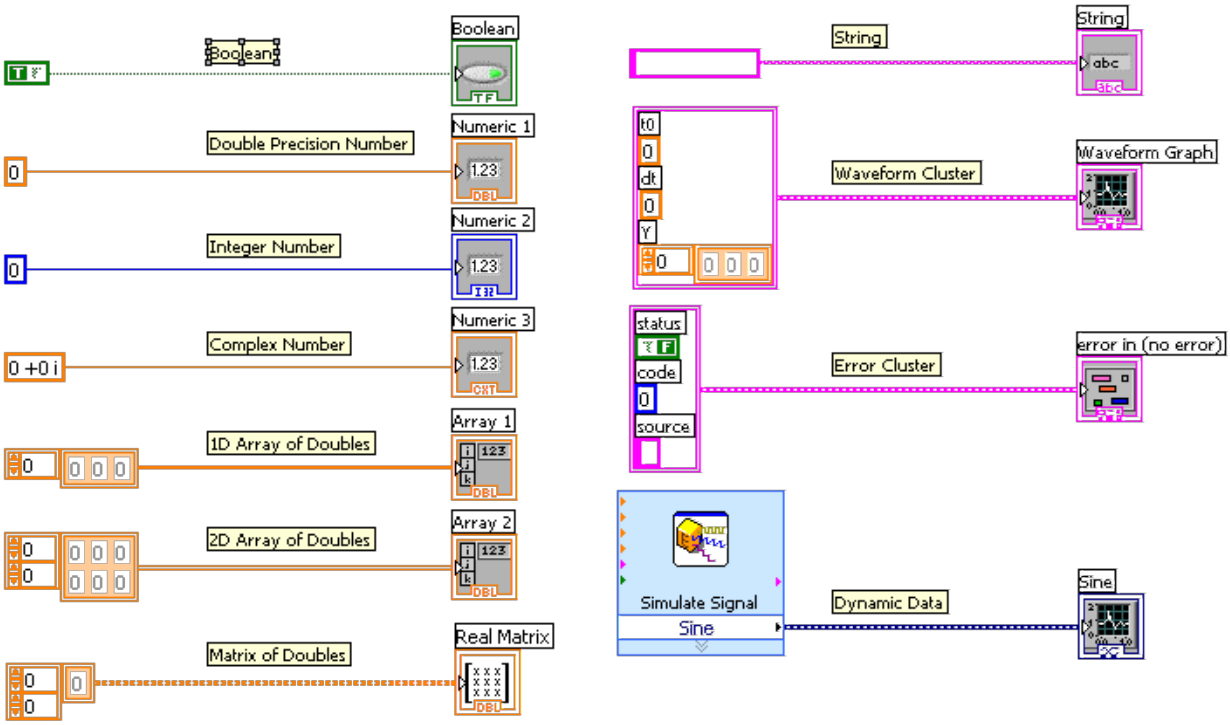
Slika 3.38 Kreiranje subVI od selektovanog kôda

Terminali će biti automatski kreirani, a ikona će biti podrazumevanog izgleda. Dvostrukim klikom na ikonu, otvara se blok dijagram i front panel kreiranog virtuelnog instrumenta, čiji se elementi mogu modifikovati.

4 Tipovi podataka

LabVIEW podržava veliki broj različitih tipova podataka. Veze koje predstavljaju različite tipove podataka su označene različitim bojama i tekstrurama. Elementarni tipovi u LabVIEW su brojni podaci (celobrojni, u pokretnom zarezu, kompleksni ili u fiksnom zarezu), znakovni podaci (karakteri i stringovi) i logički tip podataka. LabVIEW podržava složene strukture podataka, sačinjene od elementarnih, koje se nazivaju *klasteri*. Moguće je realizovati niz bilo kog tipa podatka, ili klastera.

Reprezentacije pojedinih tipova podataka su prikazane na slici 4.1.



Slika 4.1 Tipovi podataka su označeni različitim bojama i teksturama veza

4.1 Numerički tipovi

4.1.1 Celobrojni tipovi podataka

Celi brojevi (integer) su prikazani plavom bojom. Mogu biti označeni (*assigned*, označeni sa I) i neoznačeni (*unassigned*, označeni sa U). Razlikuju se po opsegu vrednosti potrebnoj memoriji. Celobrojni tip podataka može imati različite reprezentacije jednobajtni (I8 i U8), dvobajtni ili *word* (I16 i U16), dugi celobroni – *long integer* (I32 ili U32) i četvorostruki celobrojni (I64 i U64). Reprezentacija brojnog podatka se može promeniti opcijom **Representation** iz kontekstnog menija kontrole, indikatora ili konstante.

Pregled celobrojnih tipova podataka su dati u tabeli 4.1.

Opseg celih brojeva zavisi od reprezentacije, kao i prostora neophodnog za memorisanje jednog podatka. Neoznačeni celobrojni podaci mogu predstavljati samo pozitivne cele brojeve.

Tabela 4.1 Reprezentacije celobrojnih tipova podataka

Ikona	Numerički tip podataka	Dužina u bitovima	Decimalni eksponent	Opseg
	jednobajtni označeni	8	2	–128 do 127
	dvobajtni označeni	16	4	–32768 do 32767
	dugi označeni	32	9	–2147483648 do 2147483647
	četvorostruki označeni	64	18	Približno od $-1\cdot10^{19}$ do $1\cdot10^{19}$
	jednobajtni neoznačeni	8	2	0 do 255
	dvobajtni neoznačeni	16	4	0 do 65535
	dugi neoznačeni	32	9	0 do 4294967295
	četvorostruki neoznačeni	64	19	0 do približno $2\cdot10^{19}$

4.1.2 Brojevi u pokretnom zarezu

Brojevi u pokretnom zarezu (*floating point*) su prikazani vezama i terminalima narandžaste boje. Brojevi u pokretnom zarezu mogu biti jednostruke, dvostruke i proširene preciznosti. Oznaka **SGL** na ikoni označava jednostruku preciznost, **DBL** označava dvostruku preciznost, **EXT** četvorostruku numeričku preciznost broja.

Brojevi u pokretnom zarezu su u LabVIEW implementirani prema ANSI/IEEE 754-1985 standardu. Realnih brojeva ima beskonačno mnogo, tako da je moguće predstaviti samo ograničen broj imajući u vidu ograničenu memoriju računara. Iz tog razloga, prilikom izračunavanja koja uključuju brojeve u pokretnom zarezu, neizbežne su greške zaokurživanja. Brojevi u su binarnom obliku predstavljeni znakom (jedan bit), mantisom i eksponentom. Od dužine eksponenta i mantise zavise opseg i preciznost broja u pokretnom zarezu, respektivno.

Osim regularnih realnih brojeva koji se mogu približno predstaviti, ovim numeričkim tipom se mogu predstaviti i određeni specijalni kôdovi: **NaN**, **–inf**,**+inf** i **Machine epsilon**. **NaN** kôd, skraćenica od *Not a Number*, se javlja kao rezultat operacije koja nije matematički regularna (poznate i kao *floating-point exception*), kao na primer deljenje sa nulom ili kvadratni koren negativnog broja. Kodovi **–inf** i **+inf** predstavljaju minus i plus beskonačno. **Machine epsilon** je greška zaokruživanja broja u pokretnom zarezu date preciznosti.

Opsezi i preciznosti brojeva u pokretnom zarezu su predtsavljeni u tabeli 4.2. Reprezentacija brojeva u pokretnom zarezu se može promeniti opciom Represenation iz kontekstnog menija kontrole, indikatora ili konstante.

Tabela 4.2 Brojevi u pokretnom zarezu

Ikona	Numerički tip podataka	Dužina u bitovima	Broj decimalnih mesta	Približni opseg
	Jednostruka tačnost	32	6	Najmanji pozitivni broj: $1.4 \cdot 10^{-45}$
				Najveći pozitivni broj: $3.4 \cdot 10^{38}$
				Najveći negativni broj: $-1.40 \cdot 10^{-45}$
				Najmanji negativni broj: $-3.40 \cdot 10^{38}$
	Dvostruka tačnost	64	15	Najmanji pozitivni broj: $4.94 \cdot 10^{-324}$
				Najveći pozitivni broj: $1.79 \cdot 10^{308}$
				Najveći negativni broj: $-4.94 \cdot 10^{-324}$
				Najmanji negativni broj: $-1.79 \cdot 10^{308}$
	Proširena tačnost	128	15-20	Najmanji pozitivni broj: $6.48 \cdot 10^{-4966}$
				Najveći pozitivni broj: $1.19 \cdot 10^{4932}$
				Najveći negativni broj: $-6.48 \cdot 10^{-4966}$
				Najmanji negativni broj: $-1.19 \cdot 10^{4932}$

4.1.3 Kompleksni brojevi

Kompleksni brojevi imaju realni i imaginarni deo, koji ponaosob imaju osobine ekvivalentnih tipova u pokretnim zarezu. Predstavljeni su terminalima i vezama narandžaste boje. Oznaka **CSG** na ikoni označava jednostruku preciznost, **CDB** označava dvostruku preciznost, **CXT** četvorostruku numeričku preciznost kompleksnog broja (tabela 4.3).

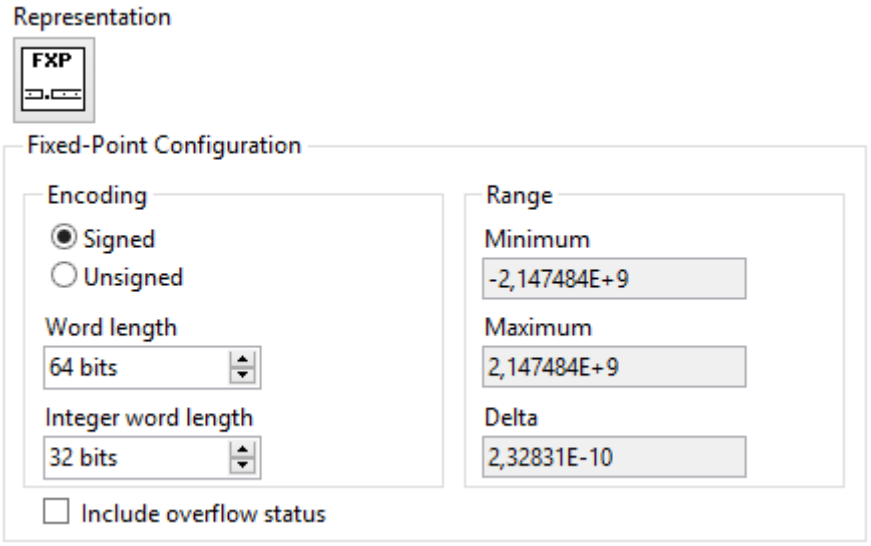
Tabela 4.3 Kompleksni brojevi

Ikona	Numerički tip podataka	Dužina u bitovima	Broj decimalnih mesta	Približni opseg
	Kompleksni jednostruke preciznosti	64	6	Kao kod broja u pokretnom zarezu sa jednostrukom preciznošću za realni i imaginarni deo
	Kompleksni dvostruke preciznosti	128	15	Kao kod broja u pokretnom zarezu sa dvostrukom preciznošću za realni i imaginarni deo
	Kompleksni višestruke preciznosti	256	15-20	Kao kod broja u pokretnom zarezu sa dvostrukom preciznošću za realni i imaginarni deo

4.1.4 Brojevi sa fiksnim zarezom i Timestamp

Brojevi sa fiksnom tačkom (*fixed point*) su numerički tip koji ima definisanu i nepromenljivu dužinu mantise i eksponenta. Obeleženi su sivom bojom. Dužine mantise i eksponenta se mogu definisati prilikom dekleracije broja. Najčešće se koriste u jezicima za opis hardvera (HDL). U LabVIEW okruženju se upotrebljavaju aplikacijama za FPGA platforme.

Konfiguracija broja sa fiksnim zarezom se može menjati dijalogom koji se aktivira opcijom **Properties** iz kontekstnog menija kontrole, indikatora ili konstante i izborom taba **Data Type** (slika 4.2).



Slika 4.2 Dijalog Properties/Data Type

U dijalogu je moguće specificirati ukupan broj bitova kojim je broj predstavljen (Word lenght), broj bitova celobrojnog dela (Integer lenght), označavanje (Signed/Unsigned) i status prekoračenja. Za zadate parametre je prikazan opseg i preciznost brojnog podatka.

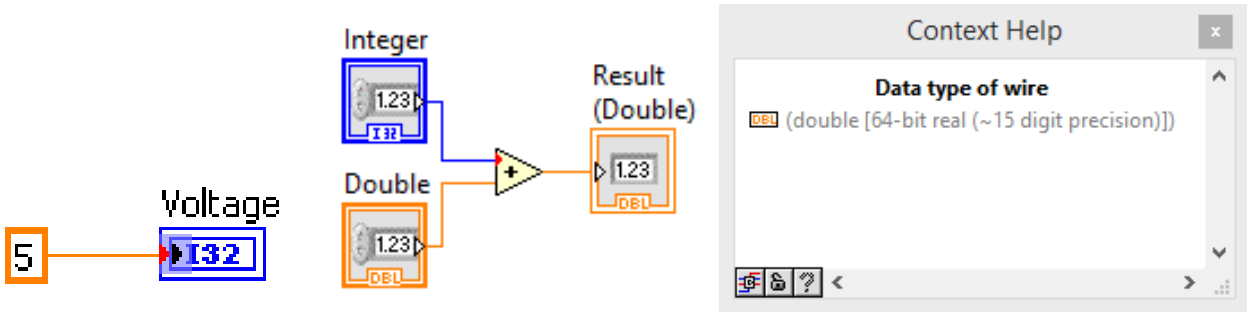
Timestamp je brojni podatak koji se koristi za predstavljanje apsolutnog vremena u virtuelnim instrumentima. Označen je braon bojom i predtsavlja 128-bitni podatak (tabela 4.4).

Tabela 4.4 Broj u fiksnom zarezu i Timestamp

Ikona	Numerički tip podataka	Dužina u bitovima	Broj decimalnih mesta	Približni opseg
FXP	Broj u fiksnom zarezu	Do 1024 (celobrojni) + 64 (razlomljeni) + 8 overflow status	varijabilan	U zavisnosti od konfiguracije
	Timestamp	128	19	Početno vreme: 01/01/1600 00:00:00 Krajnje vreme (UTC): 01/01/3001 00:00:00 UTC

4.1.5 Konverzija brojnih tipova

Ukoliko se povežu dva terminala sa različitim numeričkim reprezentacijama, LabVIEW konvertuje broj iz jedne reprezentacije u drugu. U ovom slučaju će se pojaviti crvena strelica – **coercion dot** na terminalu gde je podatak konvertovan (slika 4.3). **Coercion dot** predstavlja bafer koji je automatski postavljen između veza, funkcija ili terminala, čija funkcija je konvertovanje izvornog tip u odredišni.

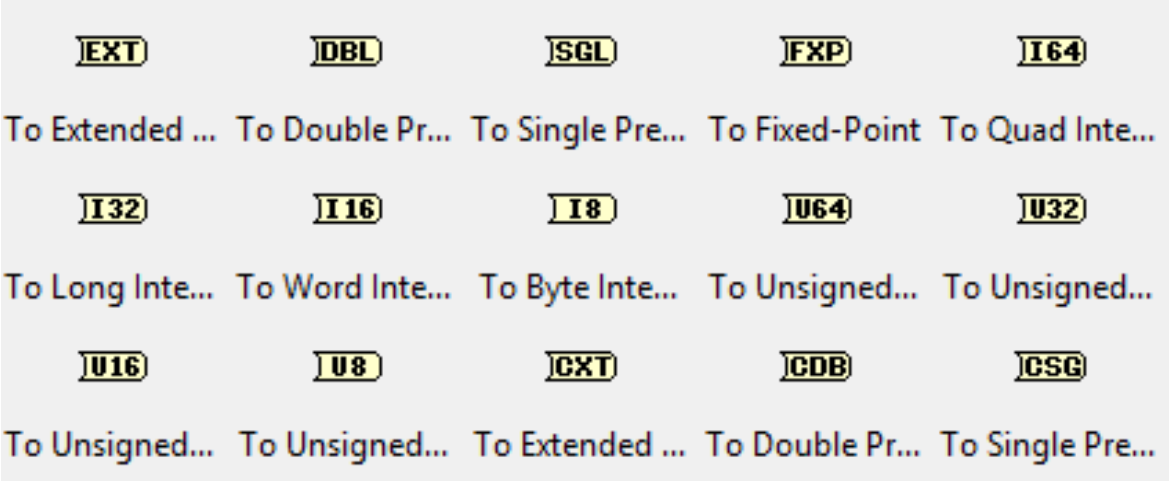


Slika 4.3 Coercion dot na terminalu prilikom konvertovanja broja u pokretnom zarezu u celobrojni tip (levo), konverzija celobrojnog u broj sa pokretnim zarezom (sredina) i Context Help prozor (desno)

U slučaju operacija nad različitim reprezentacijama, LabVIEW automatski konvertuje u reprezentaciju koji ima više bitova. Ukoliko je broj bitova isti, daje se prioritet neobeleženom (*unsigned*) nad obeleženim (*signed*) tipom. Prilikom konverzije brojeva u pokretnom zarezu u celobrojni tip, zaokurživanje se vrši na najbliži celi broj. x.5 se zaokružuje na najbliži paran celi broj. Na primer, 2.5 na 2 i 3.5 na 4. Reprezentacija terminala ili veze se može videti u **Context Help** prozoru, ukoliko se kursor dovede do terminala, konstante ili veze (slika 4.3 desno).

Automatska konvertovanja treba izbegavati, jer mogu dovesti do nekontrolisanog utroška memorije i grešaka koje je teško otkriti. Jedan od načina je uparivanje tipova, tako da se indikator ili funkcija na kojoj se pojavljuje **coercion dot** prilagodi ulaznom tipu. Prilagođavanje tipova se ostvaruje izborom opcije **Adapt to Source**, iz kontekstnog menija indikatora.

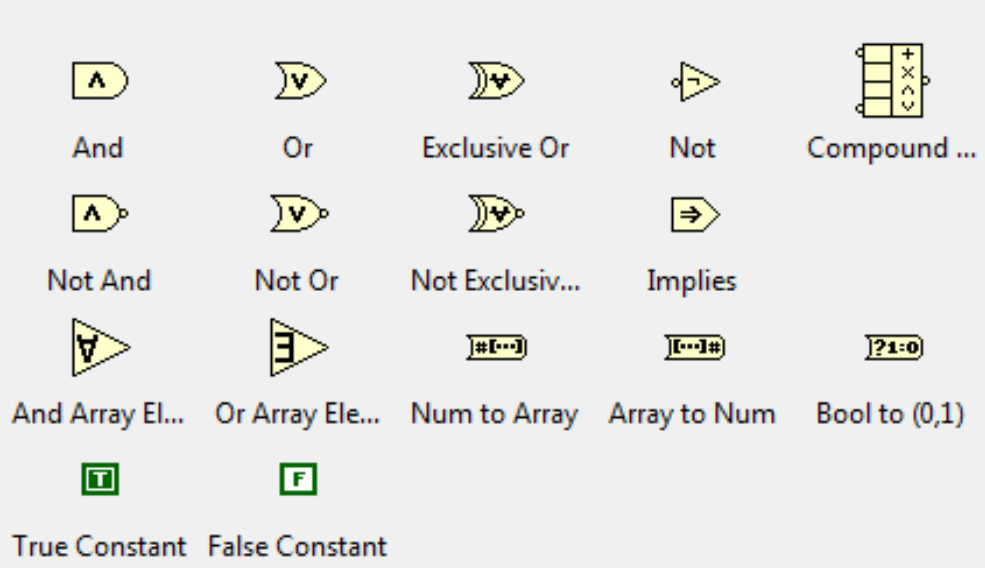
Drugi način je eksplicitno konvertovanje, izborom odgovarajuće funkcije iz palete **Functions/Numeric/Conversion** (slika 4.4).



Slika 4.4 Paleta Functions/Numeric/Conversion

4.2 Logički (Boolean) tip

Promenljive logičkog (**Boolean**) tipa mogu imati dve diskretne vrednosti **True** ili **False**. Promenljive logičkog tipa su najčešće rezultati logičkih operacija, poređenja brojnih promenljivih ili predstavljaju određene statuse. Koriste se kao promenljive za kontrolu programskih petlji (**While, Timed Loop, Conditional For**), struktura (**Case**) i signalizaciju određenih statusa (status greške). Veze i terminali logičkog tipa su predstavljeni zelenom bojim. Logičke funkcije i konstante su grupisane u **Programming/Boolean** paleti.



Slika 4.5 Programming/Boolean paleta

4.3 Znakovni podaci

Znakovni podaci su karakteri i stringovi. String je niz karaktera koji se mogu prikazati ili odštampati. Postoje različiti načini kodiranja znakovnih podataka, od kojih su najčešće upotrebljavani ASCII i UNICODE. ASCII (*American Standard Code for Information Interchange*) je sedmobitna znakovna kôdna stranica kojim su kodirana mala i velika slova engleske latinice, cifre dekadnog brojnog sistema, specijalni karakteri i kontrolne sekvence. UNICODE je šesnestobitna kodna stranica, koja podržava veliki broj pisama, alternativnih karaktera, dijakritika, znakova, itd.

Stringovi se u LabVIEW programiranju koriste za različite namene, kao što je prikazivanje poruka, kontrolu instrumenata, upis/čitanje podataka u fajlova, mrežnu komunikaciju, pristup bazama podataka, i slično. Kontrole i indikatori znakovnog tipa se mogu postaviti na front panel izborom iz **Controls»String & Path** palete. Terminali i veze znakovnog tipa su prikazani ružičastom bojom.

LabVIEW podržava nekoliko načina ispisivanja stringova: normalni način ispisivanja (**Normal**), način ispisivanja kôda (**Backslash \ Codes**), način ispisivanja lozinke (**Password**) i heksadecimalno ispisivanje (**Hexadecimal**). Način ispisivanja se može zadati izborom iz kontekstnog menija kontrole/indikatora ili iz taba **Appearance** dijaloga koji se aktivira opcijom **Properties**.

Svaki od izabranih načina ima karakterističan prikaz teksta. Način ispisivanja kôda podržava takozvane *escape* kôdove, koji su prikazani kao kombinacija *backslash* karaktera \ koga prati određeni kôd. Spisak podržanih kodova dat je u tabeli 4.5.

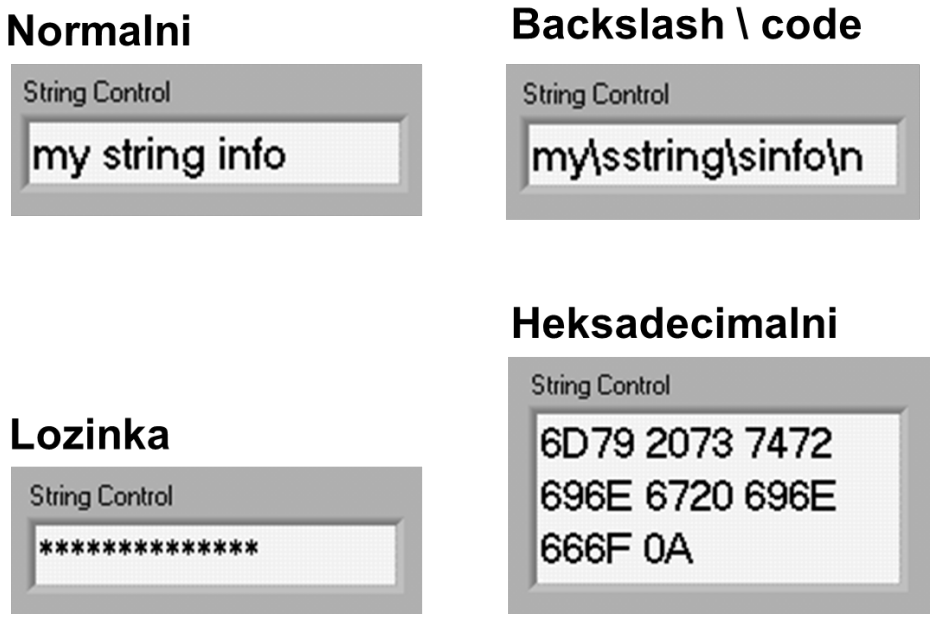
Način ispisivanja lozinke maskira tekst ispisan u kontroli ili indikatoru zvezdicom (*asterisk* karakter, *). Ovaj način ispisa je pogodan za skrivanje korisničkih šifri prilikom autentifikacije.

Heksadecimalni ispis teksta će zameniti svaki karakter njegovom heksadecimalnom vrednošću u ASCII kodnoj tabeli. Ovakav način prikaza se može iskoristiti prilikom serijske ili GPIB komunikacije.

Različiti načini ispisivanja stringa su prikazani na slici 4.6.

Tabela 4.5 Backslash kodovi podržani u LabVIEW

Kôd	Interpretacija
\00 – \FF	Heksadecimalna vrednost 8-bitnog karaktera; velika slova
\b	Backspace (ASCII BS, ekvivalentno **\08**)
\f	Form feed (ASCII FF, ekvivalentno **\0C**)
\n	Linefeed (ASCII LF, ekvivalentno **\0A**)
\r	Carriage return (ASCII CR, ekvivalentno **\0D**)
\t	Tab (ASCII HT, ekvivalentno **\09**)
\s	Space (ekvivalentno **\20**)
****	Backslash (ASCII \, ekvivalentno **\5C**)



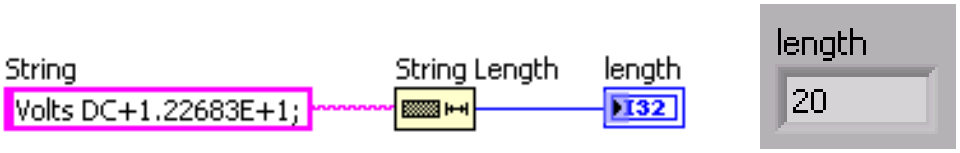
Slika 4.6 Ispisi teksta u string indikatoru

4.3.1 Najčešće korišćene funkcije sa stringovima

Funkcije za rad sa stringovima se nalaze u **Programming/Strings** paleti.

4.3.1.1 String Length

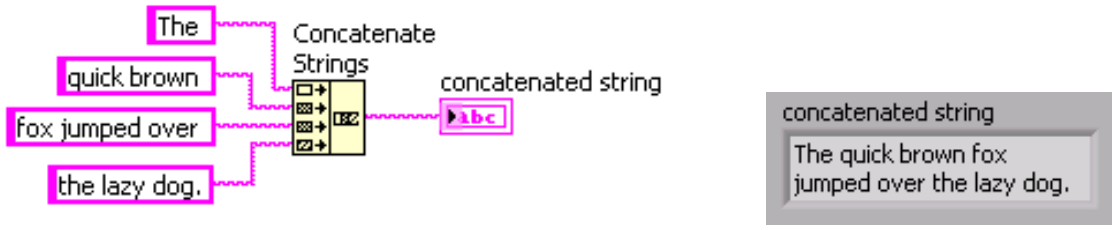
String Length funkcija određuje dužinu stringa (slika 4.7). Ulazni parametar je string, izlazni celobrojni podatak koji predstavlja dužinu stringa u karakterima, pri čemu su uračunati i neprintabilni karakteri.



Slika 4.7 String Length funkcija

4.3.1.2 Concatenate Strings

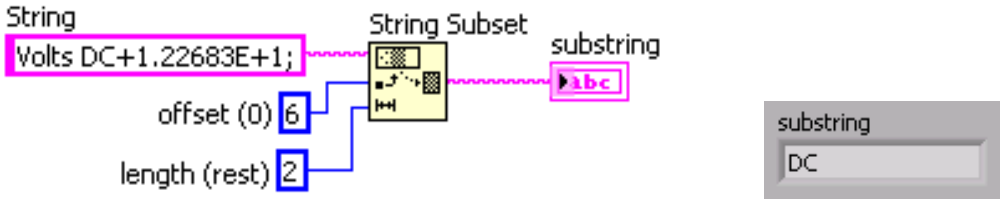
Concatenate Strings je polimorfna funkcija, koja može imati promenljiv broj ulaznih parametara. Broj parametara se može menjati povlačenjem donje ivice ikone na dole. Ulazni parametri su stringovi; funkcija spaja ulazne stringove u jedan izlazni string. String koji se dovodi na najviši ulazni terminal je na početku izlaznog stringa (slika 4.8).



Slika 4.8 Concatenate Strings funkcija, ulazni stringovi imaju blanko karakter na kraju

4.3.1.3 String Subset

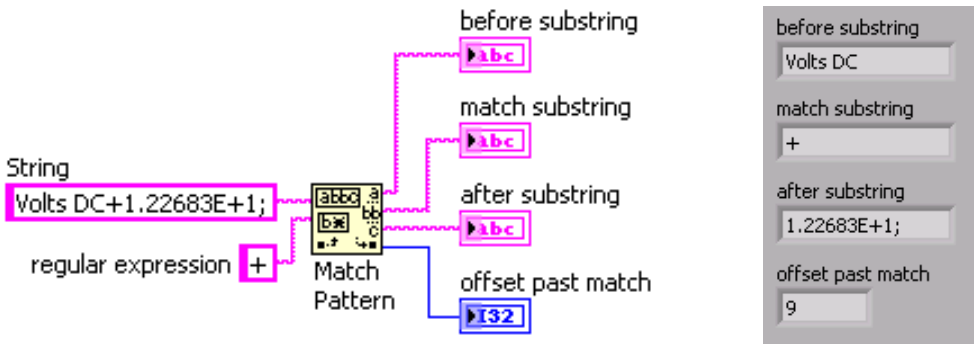
String Subset izdvaja deo stringa – substring (slika 4.9). Ulazni parametri su **string** iz koga se izdvaja deo, **offset** – pozicija karaktera od koga počinje izdvajanje (celobrojni podatak) i **length** – dužina u karakterima substringa koji se izdvaja (celobrojni podatak). Prvi karakter u stringu ima poziciju 0. Prilikom određivanja parametara String Subset funkcije, treba uzeti u obzir i neprintabilne karaktere.



Slika 4.9 String Subset funkcija

4.3.1.4 Match Pattern

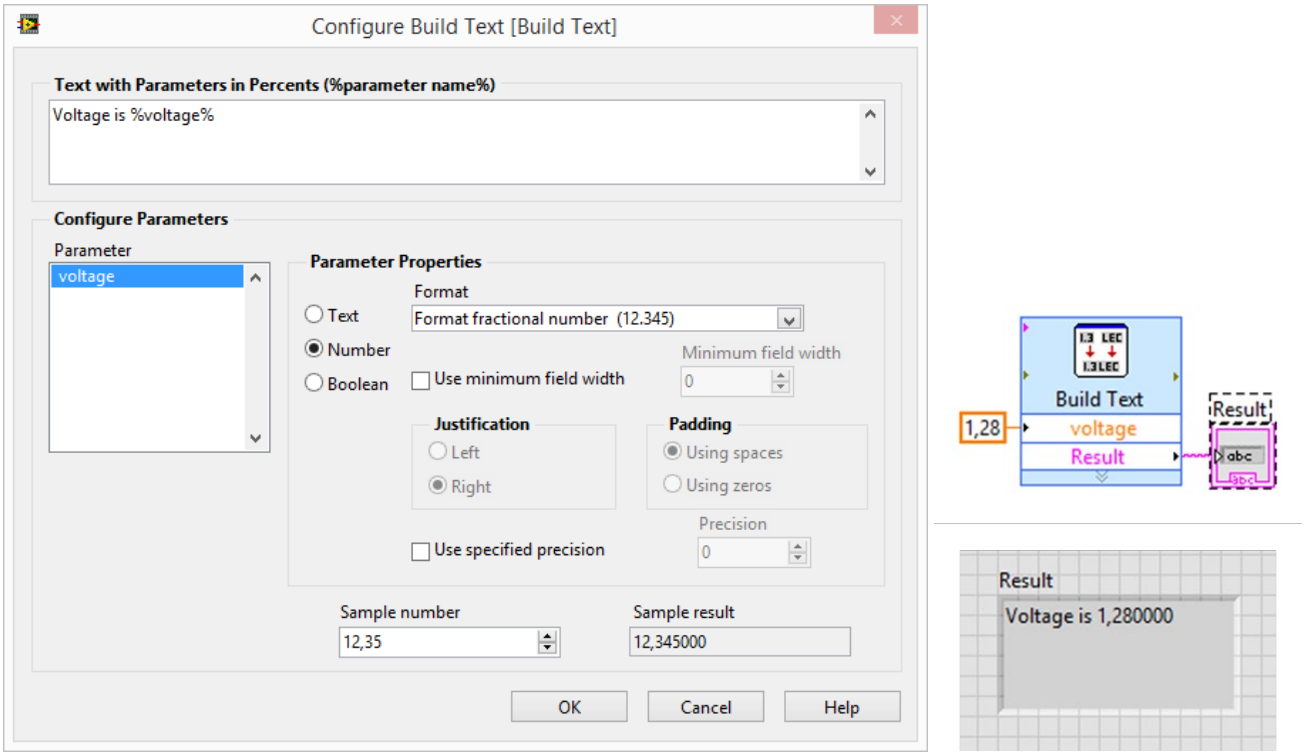
Match Pattern funvija pronalazi odgovarajući uzorak (*pattern*) u ulaznom stringu. Ulazni parametri su string, regularni izraz (*regular expression*) koji definiše uzorak za pretragu i offset – pozicija karaktera u ulaznom stringu od koje počinje pretraga. Funkcija daje četiri izlazna parametra: substring koji prethodi pronađenom uzorku, substring koji predstavlja pronađeni uzorak, substring koji sledi nakon pronađenog uzorak i **offset past match** – poziciju substringa koji sledi nakon uzorka. Ukoliko uzorak nije pronađen, poslednji parametar ima vrednost –1. Ukoliko u stringu ima više uzoraka, funkcija će dati parametre koji se odnose na prvi pronađeni uzorak.



Slika 4.10 Match Pattern funkcija

4.3.1.5 Build Text

Build Text je **ExpressVI** funkcija, čija se funkcionalnost konfiguriše u dijalogu koji se otvara dvostrukim klikom na ikonu virtuelnog instrumenta, ili izborom opcije **Properties** iz kontekstnog menija. Build Text može prihvatiti proizvoljan broj ulaznih parametara koji mogu biti numeričkog, tekstualnog ili logičkog (boolean) tipa. Tekst se formira unošenjem u polje **Text with Parameters in Percents**, u gornjem delu dijaloga (slika 4.11). Parametri se unose uokvireni znakom za procenat (**%voltage%** će biti identifikovano kao parametar **voltage**), pri čemu će svaki parametar biti prikazan u listi parametara. Tip svakog parametra se određuje u **Parameter Properties** delu dijaloga, tako da može biti numerički, tekstualni ili logički.



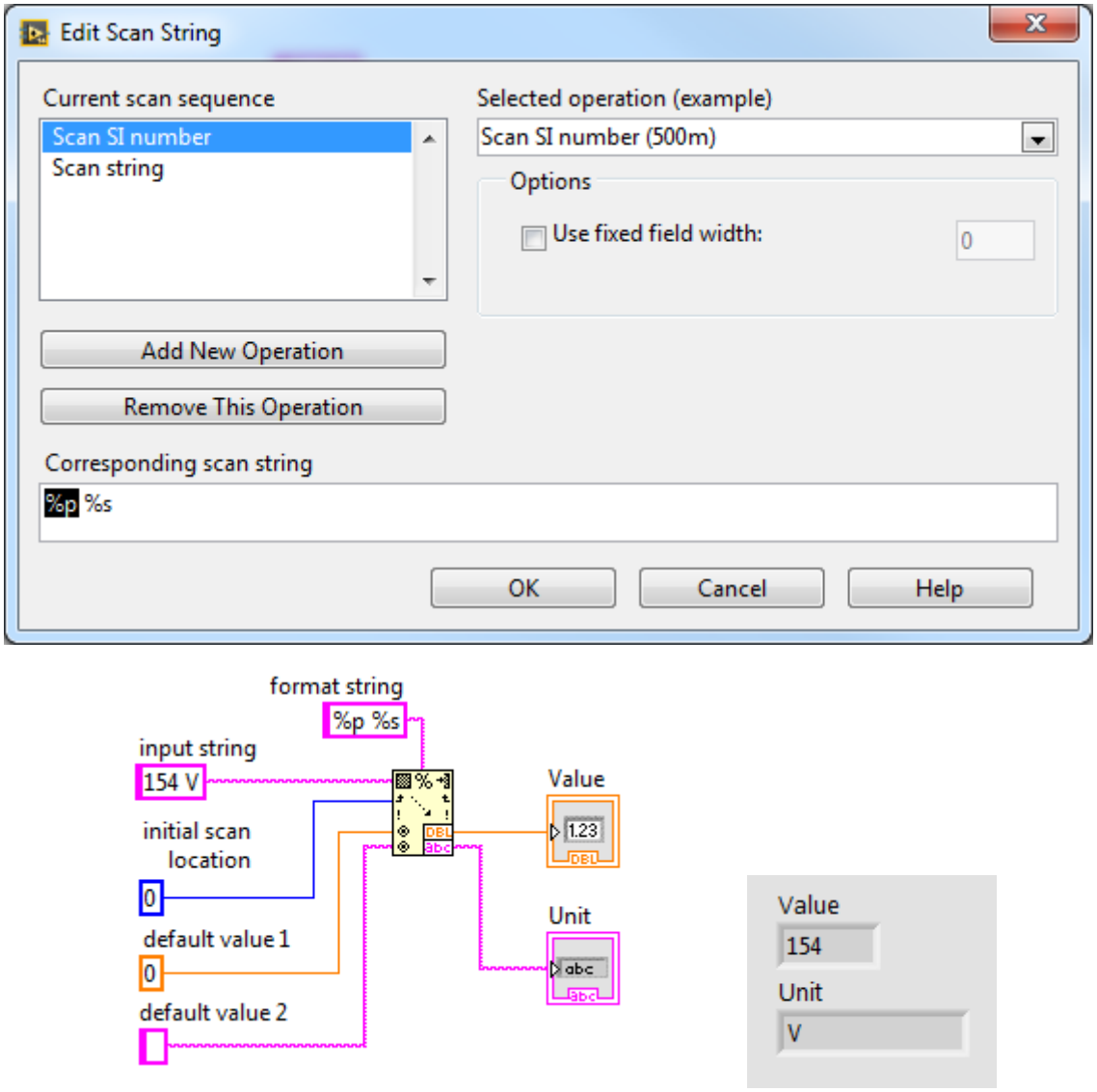
Slika 4.11 Build Text funkcija – dijalog (levo), blok dijagram virtuelnog instrumenta (desno) i front panel sa rezultatom (dole desno)

4.3.1.6 Izdvajanje parametara iz stringa – Scan From String

Scan From String funkcija konvertuje ulazni string u skup numeričkih, logičkih ili string parametara prema kriterijumu koji je zadat format stringom. Pored ulaznog stringa i format stringa, funkcija prihvata kao ulazne parametre poziciju karaktera od koga se počinje sa pretragom, klaster greške (više u poglavlju o greškama) i podrazumevane vrednosti izlaznih parametara.

Format string je ulazni parametar koji definiše konverziju ulaznog stringa. Format string se može dovesti programski na terminal funkcije ili zadati izborom opcije **Edit Scan String** iz kontekstnog menija funkcije, pri čemu se otvara dijalog za zadavanje format stringa (slika 4.12). Broj i tip izlaznih parametara funkcije je određen brojem parametara u sekvenci format stringa.

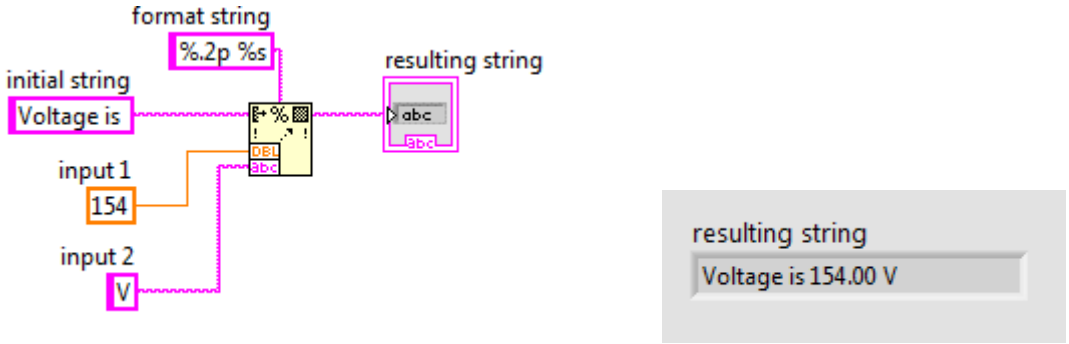
Funkcijom Scan From String se može izdvojiti parametar iz stringa ukoliko je poznat njegov format – prilikom parsiranja poruka koje su deo poznatog komunikacionog protokola. Primer funkcije je dat na 4.12.



Slika 4.12 Edit Scan String dijalog (gore), primer Scan From String funkcije (dole levo) i rezultat (dole desno)

4.3.1.7 Formatiranje stringa – Format Into String

Funkcija **Format Into String** formira string od ulaznih parametara različitih tipova. Funkcija je polimorfna, broj i tip ulaza zavise od format stringa, koji ima isti oblik kao kod funkcije Scan From String. Pored format stringa i parametara koji od njega zavise, ulazni parametri su inicijalni string i klaster greške.

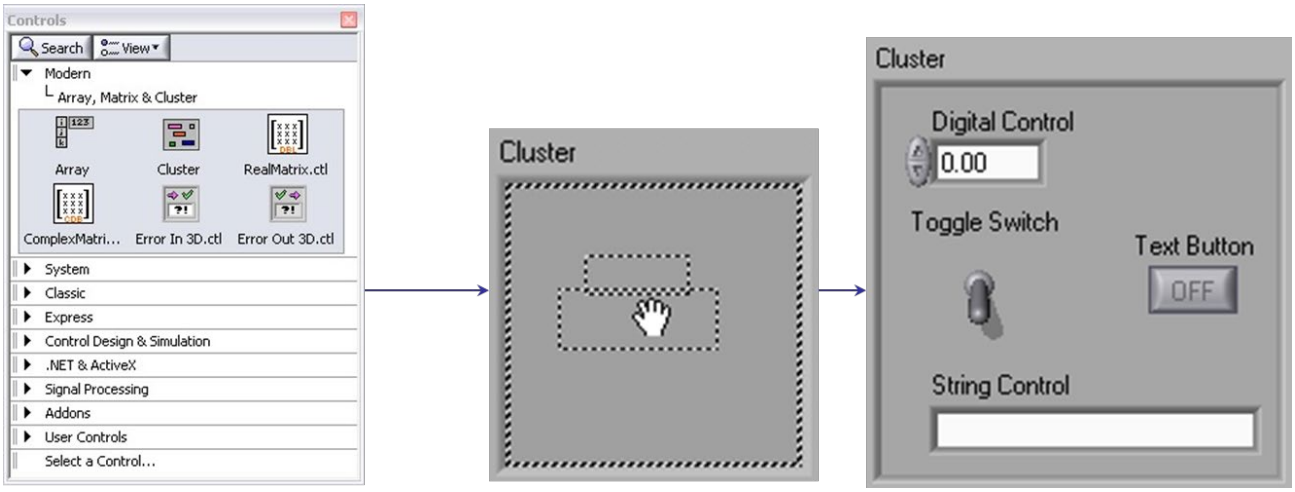


Slika 4.13 Format Into String funkcija

Izlazni string će biti formiran spajanjem ulaznog stringa i parametara konvertovanih u string na osnovu format stringa (slika 4.13). Format string se može dovesti programski na terminal funkcije ili zadati izborom opcije **Edit Format String** iz kontekstnog menija funkcije, pri čemu se otvara dijalog za zadavanje format stringa, sličan kao dijalog Scan From String funkcije (slika 4.12).

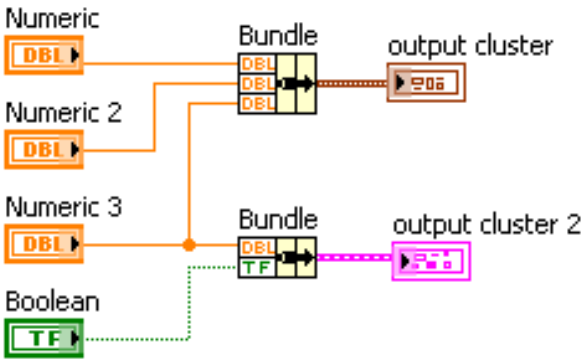
4.4 Klasteri

Klasteri predstavljaju složeni tip podataka, sastavljen od više elemenata koji pripadaju različitim tipovima. Elementi klastera mogu biti osim elementarnih tipova drugi klasteri i nizovi. U tekstualnim programskim jezicima, strukture (*struct*) i zapisi (*record*) imaju istu funkciju. Elementi klastera se identifikuju imenom, tipom i rednim brojem.



Slika 4.14 Formiranje klaster kontrole koja sadrži numeričku (Digital Control), dve logičke (Toggle Switch i Text Button) i string (String Control) kontrole

Kao i ostali tipovi podataka, klaster može biti predstavljen kontrolom, indikatorom ili konstantom. Svi elementi kontrole, indikatora ili konstante klastera moraju biti kontrole, indikatori ili konstante, respektivno. Svi terminali elemenata klastera moraju biti istog tipa. Klaster se može realizovati postavljanjem **Cluster** terminala iz **Array, Matrix & Cluster** palete na front panel virtuelnog instrumenta, i postavljanjem pojedinačnih elemenata unutar **Cluster** okvira (slika 4.14). Slično, konstanta klaster tipa se može formirati postavljanjem **Cluster Constant** iz **Programming / Cluster, Class & Variant** palete na blok dijagram virtuelnog instrumenta i dodavanjem elemenata (konstanti) unutar okvira klaster konstante. Redosled postavljanja elemenata određuje redni broj elemenata u klasteru.

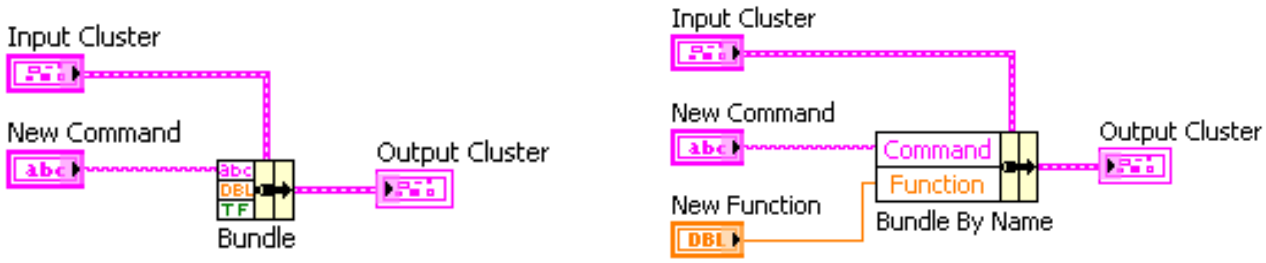


Slika 4.15 Bundle funkcije (levo), klaster sačinjen od tri numerička (DBL) elementa (braon bije) i klaster sačinjen od numeričkog i logičkog elementa (ružičaste boje)

Klaster se može formirati i funkcijom **Bundle** iz **Array, Matrix & Cluster** palete. Funkcija **Bundle** je polimorfna funkcija, sa proizvoljnim brojem i tipom ulaza. Terminali i veze klaster tipa su prikazani braon bojom ukoliko su svi elementi numerički, a ružičastom bojom ukoliko su različitih ili nenumeričkih tipova (slika 4.15). U LabVIEW postoje nekoliko klastera sa specifičnim funkcijama kao što su **Waveform** i **Error** klaster za kontrolu greške.

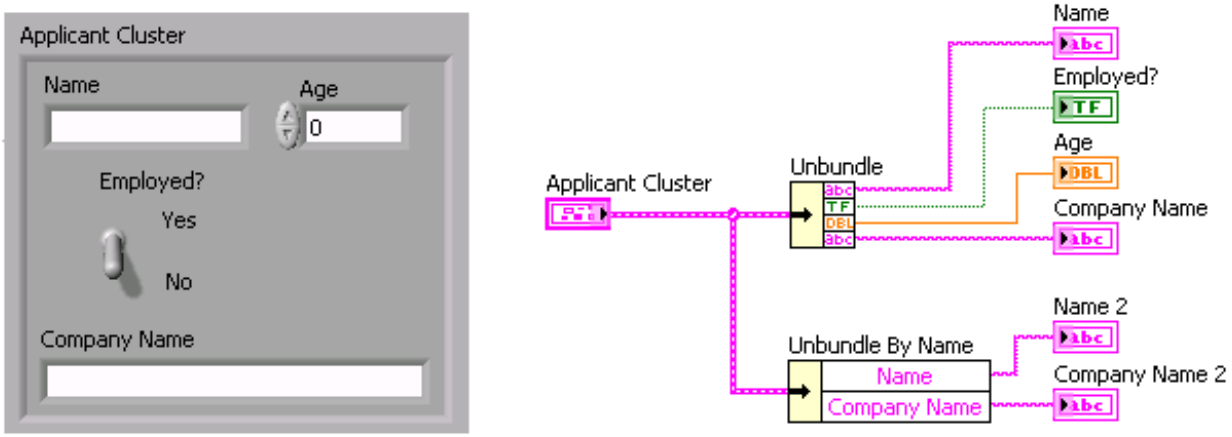
Funkcija **Bundle** formira klaster prema redosledu elemenata dovedenih na njene ulazne terminale – prvi element klastera je doveden na najviši terminal, poslednji na najniži. Ukoliko se **Bundle** funkciji prosledi klaster na **input cluster** terminal, koji se nalazi na gornjoj strani ikone, moguće je izmeniti pojedinačne elemente klastera (slika 4.16).

Izmena elemenata klastera je moguća i funkcijom **Bundle By Name**, koja se nalazi u **Array, Matrix & Cluster** paleti. Funkcija pristupa elementima klastera na osnovu njihovog imena (slika 4.16). Za razliku od **Bundle** funkcije, **input cluster** je neophodan ulazni parametar **Bundle By Name** funkcije. Izlazi **Bundle By Name** funkcije se mogu birati razvlačenjem ikone i izborom elementa iz liste.



Slika 4.16 Izmena klastera Bundle (levo) i Bundle By Name (desno) funkcijama

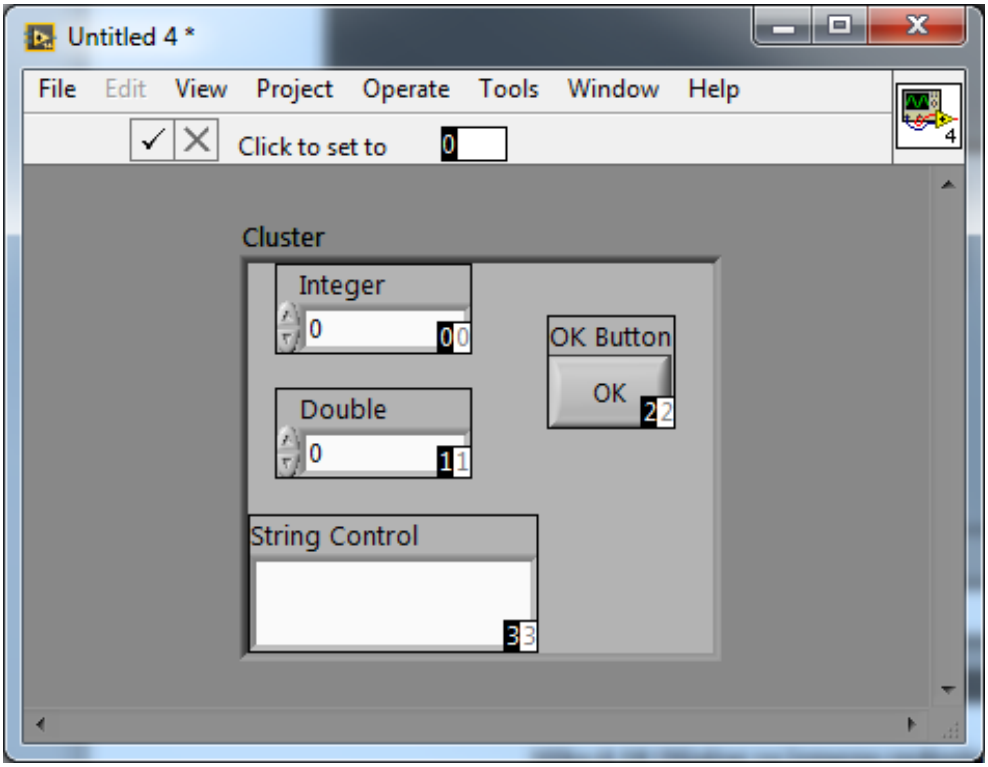
Pojedinačni elementi klastera se mogu izdvojiti **Unbundle** i **Unbundle By Name** funkcijama koje se nalaze u **Array, Matrix & Cluster** paleti. Slično kao kod odgovarajućih funkcija za formiranje klastera, **Unbundle** funkcija pristupa elementima prema redosledu, a **Unbundle By Name** funkcija prema imenima. Funkcije imaju jedan ulazni parametar – ulazni klaster – i promenljiv broj izlaza, čiji tip i broj zavisi od elemenata ulaznog klastera. Broj izlaza **Unbundle** funkcije je uvek jednak broju elemenata klastera, pri čemu tipovi odgovaraju tipovima elemenata. Izlazi **Unbundle By Name** funkcije se mogu birati razvlačenjem ikone i izborom elementa iz liste. Primer pomenutih funkcija je prikazan na slici 4.17.



Slika 4.17 Ulazni klaster (levo) i Unbundle i Unbundle By Name funkcije (desno)

Redosled elemenata u klasteru je određen prilikom njegovog formiranja, redosledom postavljanja pojedinačnih kontrola (indikatora ili konstanti) u okvir klastera ili redosledom

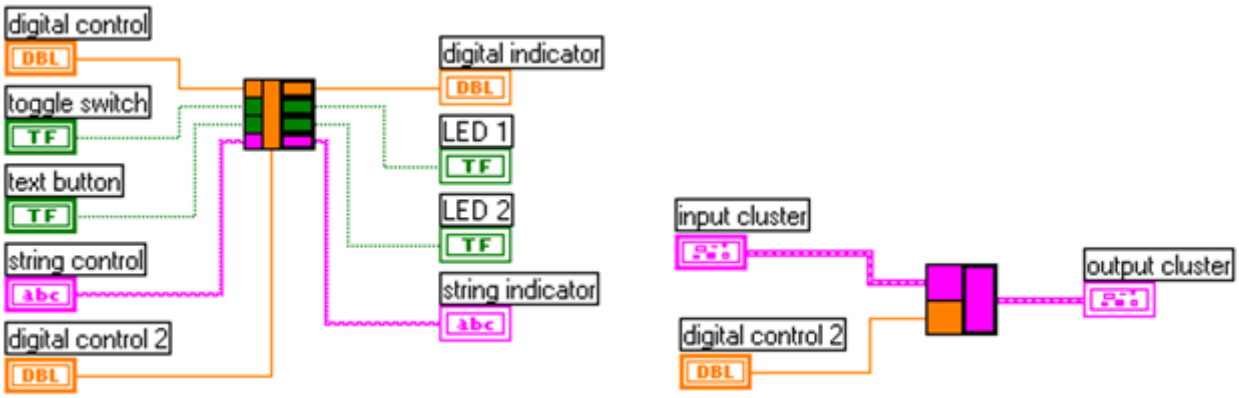
ulaznih parametara **Bundle** funkcije. Prvi element u klasteru je element 0, drugi element 1, i tako dalje. Redosled elemenata kontrole, indikatora ili konstante je moguće izmeniti izborom opcije **Reorder Controls in Cluster** iz kontekstnog menija, čime se menja izgled aktivnog prozora (slika 4.18).



Slika 4.18 Dijalog za izmenu redosleda elemenata u klasteru

Sukcesivnim obeležavanjem elemenata klastera određuje se novi redosled elemenata. U belom polju elementa prikazan je stari redni broj, a u belom novi. Redni broj elementa se može zadati poljem **Click to set to**.

Upotrebom klastera postiže se bolja preglednost blok dijagrama virtuelnog instrumenta, podaci su bolje struktuirani i grupisani po kategorijama. Klastere je pogodno koristiti i kada postoji veliki broj podataka različitog tipa koje je potrebno proslediti i obraditi u jednom subVI-u. Prilikom definisanja konektorskog panela, maksimalan broj terminala je 28. Povezivanje velikog broja parametra može biti zamoran posao sklon greškama. Prevazilaženje limita u broju terminala i lakše povezivanje se može postići definisanjem ulaznog ili izlaznog koji klastera objedinjuje sve ulazne, odnosno sve izlazne elemente (slika 4.19).



Slika 4.19 Grupsanje ulaznih i izlaznih parametara subVI u klastere