UDŽBENIK ELEKTROTEHNIČKOG FAKULTETA U BEOGRADU

Milica Janković, Marko Barjaktarović,
Marija Novičić, Petar Atanasijević

PRAKTIKUM IZ MERNO-AKVIZICIONIH SISTEMA

Beograd, 2019.

dr Milica Janković, docent
dr Marko Barjaktarović, docent
Marija Novičić, master inž., saradnik u nastavi
Petar Atanasijević, master inž., asistent
Univerzitet u Beogradu – Elektrotehnički fakultet

PRAKTIKUM IZ MERNO-AKVIZICIONIH SISTEMA

elektronski udžbenik

Recenzenti:
dr Mirjana Simić-Pejović, vanredni profesor
dr Nenad Miljić, vanredni profesor

Nastavno-naučnog veće Elektrotehničkog fakulteta Univerziteta u Beogradu odobrilo je objavljivanje ovog udžbenika odlukom broj 846/3 od 12.11.2019. godine.

Izdavač:
Univerzitet u Beogradu – Elektrotehnički fakultet
Bulevar kralja Aleksandra 73, 11120 Beograd, Srbija

ISBN: 978-86-7225-073-2

Itasdi

Co-funded by the Erasmus+ Programme of the European Union

Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems, Erasmus+ KA2 2018-1-RS01-KA203-000432

Našim studentima

„Znanje, to su zlatne lestvice preko kojih se ide u nebesa; znanje je svetlost koja osvetljava naš put kroz život i vodi nas u budućnost punu večne slave“

Mihajlo Pupin

Sadržaj

Deo I – Primena LabVIEW softverskog alata u merno-akvizicionim sistemima

Deo II – Primena *Arduino* platforme i *Python* softverskog alata u merno-akvizicionim sistemima

Predgovor

„Praktikum iz merno-akvizicionih sistema“ je udžbenik nastao u okviru rada na projektu Erasmus+ KA2 2018-1-RS01-KA203-000432 strateška partnerstva u okviru visokog obrazovanja: „**Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems (ITASDI)**“, 2018-2019, http://itasdi.uns.ac.rs/. Udžbenik je prevashodno namenjen studentima druge godine Elektrotehničkog fakulteta Univerziteta u Beogradu koji prate kurs pod istoimenim nazivom „Praktikum iz merno-akvizicionih sistema“ (šifra predmeta 13E053PMS), ali i drugima koji žele da steknu osnovna znanja i praktična iskustva u dizajniranju merno-akvizicionih sistema.

U uvodnom delu knjige je objašnjen koncept sofverski dizajnirane instrumentacije, a potom se čitaoci uvode u konkretne realizacije i primene ovakvih sistema kroz detaljno („korak po korak“) objašnjene primere. Rešenja svih primera su dostupna na sajtu ITASDI projekta. Osim elementarnih zadataka, knjiga sadrži i tzv. „zadatke za samostalni rad“ koji imaju za cilj povezivanje stečenog osnovnog znanja i razvoj veština dizajniranja složenijih aplikacija. Predlozi rešenja autora „zadataka za samostalni rad“ su takođe dostupna na sajtu ITASDI projekta.

U prvom delu knjige (Lekcije 1-9) autori su ilustrovali specifičnosti i primenu komercijalnog *LabVIEW* (*National Instruments*, SAD) softverskog okruženja u akviziciji signala i slike. U ovom delu su objašnjene osnove *data flow* principa programiranja, načini debagovanja aplikacija, primene različitih vrsta petlji i različitih struktura podataka, ilustrovane su mogućnosti modularnog programiranja i različiti pristupi upisa podataka u datoteke. Osim osnova *LabVIEW* programiranja, u ovom delu su objašnjene i napredne tehnike paralelnog programiranja i događajem vođenog programiranja kao i način kreiranja distribucije *LabVIEW* aplikacija.

U drugom delu knjige (Lekcije 10-14) su objašnjeni primeri korišćenja „otvorenog“ harvera (eng. *open source hardware*) i softvera „otvorenog“ kôda (eng. *open source software*) za sofverski dizajniranu instrumentaciju. U ovom delu su ilustrovane primene *Arduino* mikrokontrolera i kompatibilnih senzora kroz primere sistema koji koriste njegove analogne i digitalne portove kao i serijsku komunikaciju. Za kreiranje grafičkog korisničkog interfejsa u zadacima je odabrana *PyQt* biblioteka *Python* programskog jezika. Među pratećim datotekama knjige, u direktorijumu „Dodatak“, priloženi su i studentski posteri prezentovani na *Seminaru iz merno-akvizicionih sistema* održanog u zgradi Rektorata Univerziteta Beogradu, 25.10.2019. u okviru diseminacione aktivnosti ITASDI projekta.

Na ovakav koncept knjige koja se jednim delom odnosi na primene komercijalno dostupnog sotvera, a drugim delom na primenu „otvorenih“ platformi su u velikoj meri uticali Akademik Dejan Popović, redovni profesor u penziji Elektrotehničkog fakulteta Univerziteta u Beogradu (na prvi deo) i dr Predrag Pejović, redovni profesor Elektrotehničkog fakulteta Univerziteta u Beogradu (na drugi deo).

Autori se zahvaljuju univerzitetskim partnerskim institucijama koje su učestvovale na ITASDI projektu (Fakultetu tehničkih nauka - Univerzitet u Novom Sadu, Tehničkom veleučilištu u Zagrebu – Univerzitet za primenjene nauke, Fakultetu

za elektrotehniku i informacione tehnologije – Univerzitet Sv. Kiril i Metodije u Skoplju, Fakultetu za fiziku Univerziteta za tehnologije u Varšavi) na saradnji tokom pripreme ovog udžbenika, kao i na saradnji tokom organizacije takmičenja *Balkan Open Competition in Software-designed Instrumentation*, http://blsc.etf.rs/.

Autori se posebno zahvaljuju recenzentima udžbenika, dr Mirjani Simić-Pejović, vanrednom profesoru Elektrotehničkog fakulteta Univerziteta u Beogradu i dr Nenadu Miljiću, vanrednom profesoru na Mašinskom fakultetu Univerziteta u Beogradu na strpljenju i podršci pri pisanju ovog udžbenika.

Beograd, oktobar 2019. *Autori*

Softverski dizajnirana instrumentacija

Razvoj računarskih tehnologija u poslednjih pola veka značajno je uticao na promenu uloge softvera u sistemima za testiranje i merenje. Tradicionalni instrumenti, namenski dizajnirani za merenje određenih fizičkih veličina, su 70-ih godina prošlog veka bili potisnuti programabilnim instrumentima sa GPIB (*General Purpose Interface Bus*) komunikacijom. Desetak godina kasnije su se pojavili i prvi merni računarski sistemi koji su bili bazirani na standardnom personalnom kompjuteru i imali u sebe ugrađene akvizicione kartice sa GPIB interfejsom pri čemu je način rada računarski-baziranog mernog sistema u potpunosti bio definisan softverom na računaru. Ovakvim konceptom je omogućeno da se paleta tradicionalnih instrumenata, fizički prisutnih na laboratorijskom stolu, zameni jednim personalnim računarom sa akvizicionom karticom i paletom softvera dizajniranih za specifične namene. Dalji razvoj pre svega u oblasti integrisanih tehnologija, uticao je na poboljšanje performansi akvizicionih kartica i njihovih interfejsa (*PCI, PCMCIA, RS 232, RS 485, USB, PXI, FireWire, Bluetooth, Ethernet, LAN, CAN,* I^2C itd.), ali je koncept „softverom definisane instrumentacije“, „softverski dizajnirana instrumentacija“ ili „virtuelne instrumentacije“ u kojoj je softver krucijalan deo merne instrumentacije ostao nepromenjen do danas.

Standardna arhitektura modernog mernog sistema obuhvata:

1. senzore koji mernu veličinu pretvaraju u električni signal (npr. termistori za merenje temperature, fotootpornici za merenje nivoa osvetljenosti, kapacitivni pretvarači za merenje pomeraja i sl.)
2. kola za kondicioniranje (pojačavači, filtri, kola za korekciju i sl.) koja omogućavaju da se električni signal sa izlaza senzora prilagodi tako da se performanse akvizicione kartice maksimalno iskoriste
3. akviziciona kartica za diskretizaciju signala u skladu sa teoremom odabiranja (frekvencija odabiranja može biti hardverski ili softverski kontrolisana)
4. memorija računara gde se po FIFO (*First In First Out*) principu smeštaju i čitaju odbirci signala
5. računar sa instaliranim drajverima za akvizicionu karticu i softverom koji omogućava:
    a. interfejs ka korisniku mernog sistema,
    b. obradu i vizuelizaciju podataka,
    c. snimanje, tzv. logovanje podataka,
    d. dalji prenos podataka (lokalna mreža, Internet).

Pri projektovanju savremenih merno-akvizicionih sistema, a u skladu sa prirodom merne veličine i namenom dizajniranog sistema, potrebno je voditi računa o sledećim koracima:

1. izbor senzora prema sledećim kriterijumima:
    a. karakteristikama senzora (osetljivost, ulazni opseg, stabilnost, ponovljivost i reproducibilnost, rezolucija, linearnost, histerezis, vreme odziva, frekvencijski odziv)

   b. ekonomskim faktorima (cena, dostupnost, vreme života)
   c. faktorima okruženja (opseg temperature, vlažnost, korozija, zaštita od prekoračenja opsega, osetljivost na elektromagnetnu interferenciju, čvrstina, veličina, potrošnja energije, mogućnost samokalibracije)
2. projektovanje kondicioniranja signala prema sledećim kriterijumima:
   a. prilagođenje propusnog opsega i pojačanja/slabljenja frekvencijskom i amplitudskom opsegu signala koji se meri i karakteristikama akvizicione kartice
   b. eliminacija artefakta (usled drifta u električnim kolima, interferencije, magnetske indukcije)
   c. niskošumni zahtevi
   d. mala merna nesigurnost
   e. mala potrošnja kola
3. izbor akvizicione kartice ili mikrokontrolera sa analogno-digitalnim (A/D) konvertorom prema sledećim kriterijumima:
   a. broj ulaza/izlaza i tip ulaza/izlaza (analogni/digitalni)
   b. paralelni A/D konvertori (po jedan za svaki od analognih ulaza) ili multipleksiranje analognih kanala na jednom A/D konvertoru
   c. referenciranje ulaznih kanala (diferencijalno merenje, merenje u odnosu na masu ili u odnosu neku drugu referentnu tačku)
   d. brzina i rezolucija analogno-digitalne konverzije
4. izbor računara u skladu sa zahtevima koja se odnose na merenja u realnom vremenu
5. izbor softverskog okruženja za dizajniranje instrumenta prema sledećim kriterijumima:
   a. isplativost cene softverskog okruženja (primena softvera otvorenog kôda, primena komercijalnih softvera)
   b. pouzdanost okruženja
   c. potreba korisnika da instrument radi pod određenim operativnim sistemom
   d. cena održavanja i nadgradnje aplikacije
   e. cena distribucije aplikacije.

U narednim poglavljima će biti detaljno opisani primeri jednostavnih merno-akvizicionih sistema koji koriste: 1) NI USB A/D karticu i komercijalno *LabVIEW* softversko okruženje i 2) hardver otvorenog kôda (*Arduino* mikrokontroler) i *Python* softver otvorenog kôda.

DEO I

Primena LabVIEW softverskog alata u merno-akvizicionim sistemima

Lekcija 1 – *LabVIEW* okruženje

Cilj

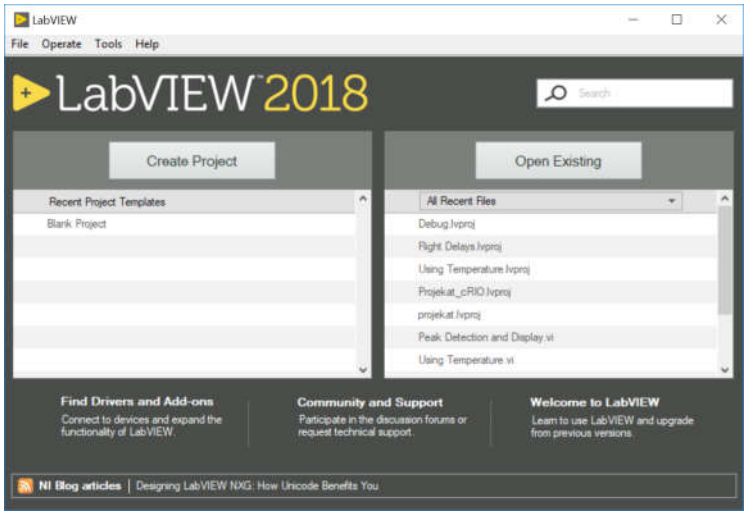
Cilj lekcije je da studente upozna sa:

- pokretanjem programa u *LabVIEW* okruženju
- osnovnim komponentama i alatima *LabVIEW* okruženja
- tehnikama debagovanja u *LabVIEW* okruženju
- kreiranjem projektne datoteke.

1.1 Pokretanje LabVIEW programa

LabVIEW programi imaju ekstenziju **.vi** i nazivaju se **V**irtuelni **I**nstrumenti (VI) jer svojim interfejsom i mogućnostima primene podsećaju na klasične instrumente (osciloskope, multimetre, kontrolere itd.).

LabVIEW softver se pokreće izborom opcije: ***Start»All programs»National Instruments»LabVIEW 2018 (32-bit)*** ili klikom na ***LabVIEW*** prečicu na *Desktop*-u. Inicijalni NI LabVIEW prozor je prikazan na Sl. 1.1.1.

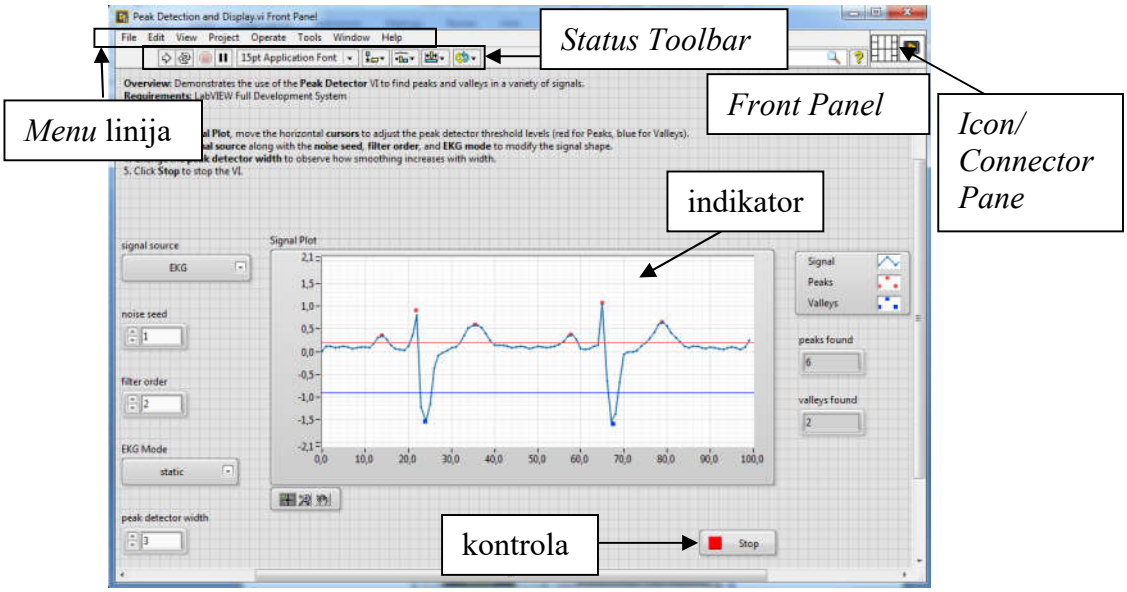


Sl. 1.1.1. *LabVIEW* inicijalni prozor

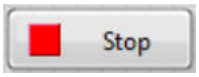
Pokrenuti primer *Peak Detection and Display.vi* na sledeći način:

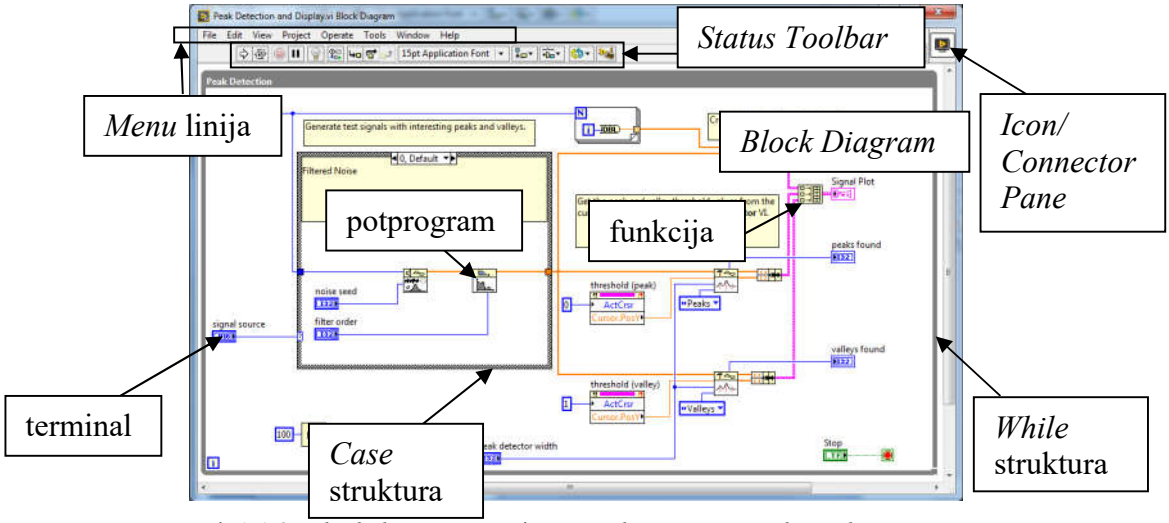
1. U prozoru na Sl. 1.1. selektovati opciju ***Help... »Find Examples***.
2. Selektovati ***Analysis, Signal Processing and Mathematics***, potom ***Signal Processing***, i na kraju ***Peak Detection and Display.vi.*** Pojaviće se prozor kao na Sl. 1.2. Ovaj prozor predstavlja *Front panel*, tj. korisnički interfejs za pozvani program. Na korisničkom interfejsu se nalaze: 1) **kontrole**

(promenljive koje omogućavaju korisniku da zada vrednosti na korisničkom interfejsu) i 2) **indikatori** (promenljive na kojima se prikazuju informacije i koje korisnik ne može da menja). Na *Front Panel*-u uočiti gde se nalaze *Menu* linija i *Status Toolbar*, Sl. 1.1.2.



Sl. 1.1.2 Front panel za primer *Peak Detection and Display.vi*

3. Pritiskom na ***Run*** dugme  pokrenuti program. Menjati vrednosti **kontrola** na interfejsu (*signal source*, *noise seed*, *filter order*, *EKG Mode*, *peak detector width*) i posmatrati promene na grafičkom **indikatoru** (*Signal Plot*, *peaks found*, *valleys found*).
4. Pritiskom na dugme ***Stop***  na *Front Panel*-u zaustaviti program.
5. Pokrenuti ponovo izvršavanje programa pritiskom na ***Run*** dugme  . Potom nasilno prekinuti izvršavanje programa pritiskom na dugme ***Abort Execution***  .
6. Otvoriti prozor *Block diagram* izborom ***Window»Show Block Diagram***, Sl. 1.1.3. Na *Block Diagram*-u uočiti gde se nalaze *Menu* linija i *Status Toolbar*, Sl. 1.1.3.



Sl. 1.1.3. *Block diagram* za primer *Peak Detection and Display.vi*

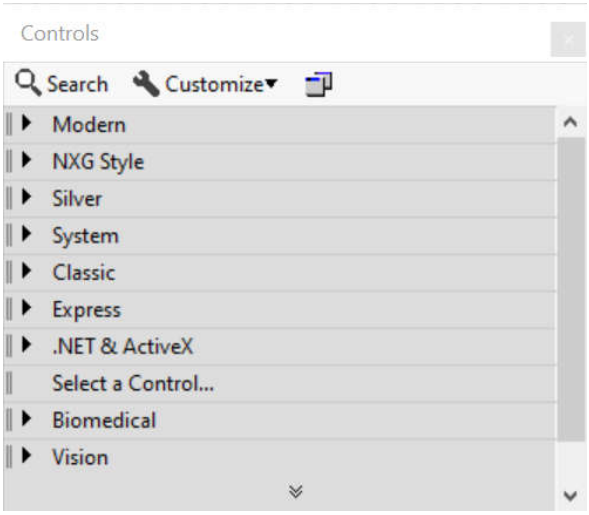
7. Uočiti da *Block diagram* sadrži terminale (tj. promenljive), funkcije, potprograme, strukture prikazane na Sl. 1.3. Uočiti da su terminali i žice različitih boja (npr. plave boje su celobrojne promenljive, roze boje su nizovne promenljive, zelene boje su logičke promenljive itd.). Uočiti i da se debljina žica razlikuje (tanke linije odgovaraju skalarnim promenljivama, deblje linije odgovaraju nizovima i matricama).
8. Zatvoriti program bez čuvanja eventualnih izmena.

NAPOMENA: Svaki *LabVIEW* VI sadrži tri osnovna dela: 1) ***Front panel*** (korisnički interfejs), 2) ***Block diagram*** (programski kôd) i 3) ***Icon/Connector pane*** (ikona i konektor) – gornji desni ugao *Front Panel*-a i *Block Diagram*-a - omogućava da se program koristi kao potprogram u nekom drugom VI-u (primena je detaljno objašnjena u Lekciji 5).

1.2 Kreiranje i debagovanje programa

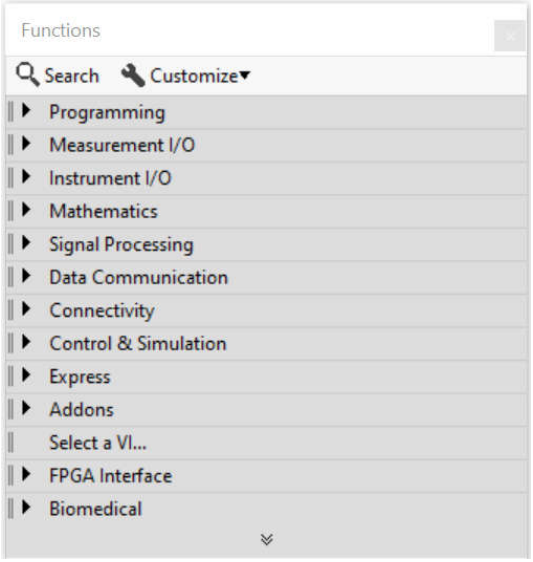
Zadatak 1.2.1.
Kreirati program koji vrednost numeričke kontrole prikazuje na grafičkom indikatoru. *Front panel* (korisnički interfejs) programa obojiti po želji. Omogućiti da se pri postavljanju miša iznad numeričke kontrole pojavi natpis „Temperatura“, kao i da se u dokumentacije numeričke kontrole pojavi objašnjenje „Ova promenljiva oznacava temperaturu“.

1. Kreirati nov **.vi** program izborom opcije ***File»New VI***. Pojaviće se dva prozora: *Front Panel* i *Block Diagram*. Ova dva prozora se mogu istovremeno posmatrati izborom opcije ***Window»Tile Up and Down*** ili izborom opcije ***Window»Tile Left and Right***.
2. Otvoriti paletu ***Controls*** selekcijom ***View»Controls Palette*** na *Front Panel*-u, Sl. 1.2.1.



Sl. 1.2.1. Paleta *Controls*

3. Otvoriti paletu ***Functions*** selekcijom ***View»Functions Palette*** u *Block Diagram*-u, Sl. 1.2.2.



Sl. 1.2.2. Paleta *Functions*

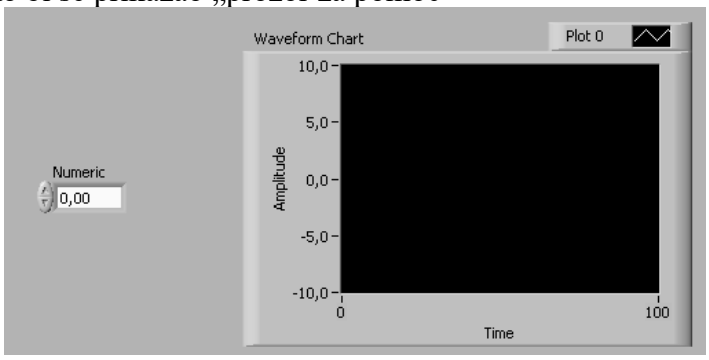
4. Otvoriti paletu ***Tools*** selekcijom ***View»Tools Palette*** ili na *Front Panel*-u ili u *Block Diagram*-u, Sl. 1.2.3. Ova paleta služi za manipulaciju elementima u oba prozora.

Sl. 1.2.3. Paleta *Tools*

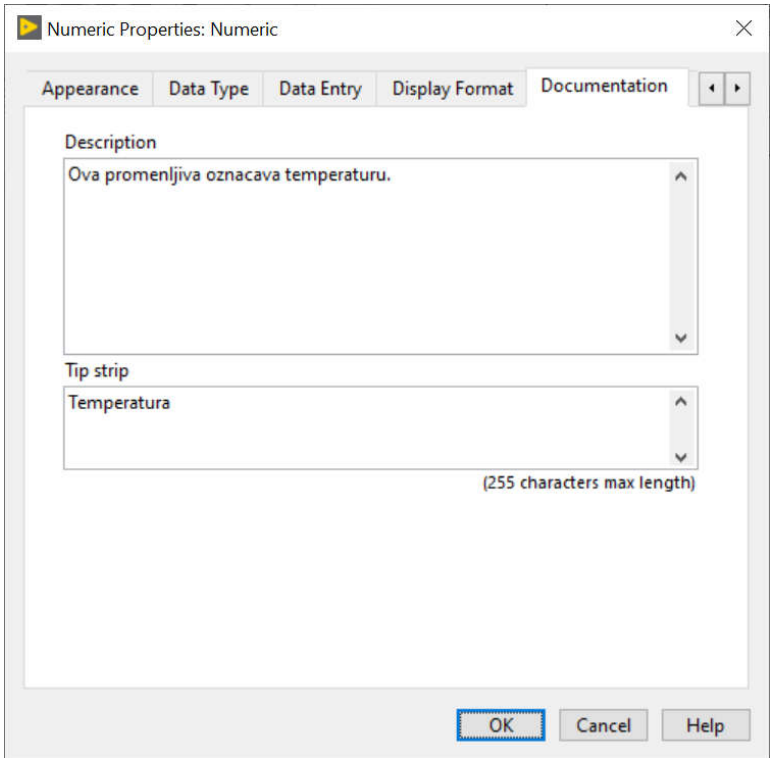
NAPOMENA: Paleta ***Tools*** je dostupna i na *Front Panel*-u i u *Block Diagram*-u za razliku od paleta ***Controls*** i ***Functions*** koje su dostupne samo u jednom od ova dva prozora.

5. Iz ***Controls»Modern*** palete izabrati kontrolu ***Numeric»Numeric Control*** i postaviti je na *Front panel*. U *Block Diagram*-u će se pojaviti odgovarajući terminal novopostavljene kontrole.
6. Iz ***Controls»Modern*** palete izabrati indikator ***Graph»Waveform Chart*** i postaviti ga na *Front panel*, Sl. 1.2.4. U *Block Diagram*-u će se pojaviti odgovarajući terminal novopostavljenog indikatora.
7. Klikom desnog tastera miša na *Waveform Chart* u *Font panel*-u ili *Block Diagram*-u otvoriti padajući meni. Izabrati opciju *Properties* i uočiti opcije (*Appearance*, *Data Type*, *Data Entry*, *Display Format*, *Documentation*) koje okruženje nudi za podešavanje izgleda ovog indikatora (prikaz, tip podataka, ograničenja promenljive, prikaz formata, dokumentacije, respektivno). U sekciji

Documentation uneti natpise kao na Sl. 1.2.5. Nakon unosa natpisa pritisnuti CTRL+H kako bi se prikazao „prozor za pomoć“



Sl. 1.2.4 Izgled *Front Panel*-a sa numeričkom kontrolom *Numeric* i grafičkim indikatorom *Waveform Chart*



Sl. 1.2.5. Podešavanje natpisa numeričke kontrole

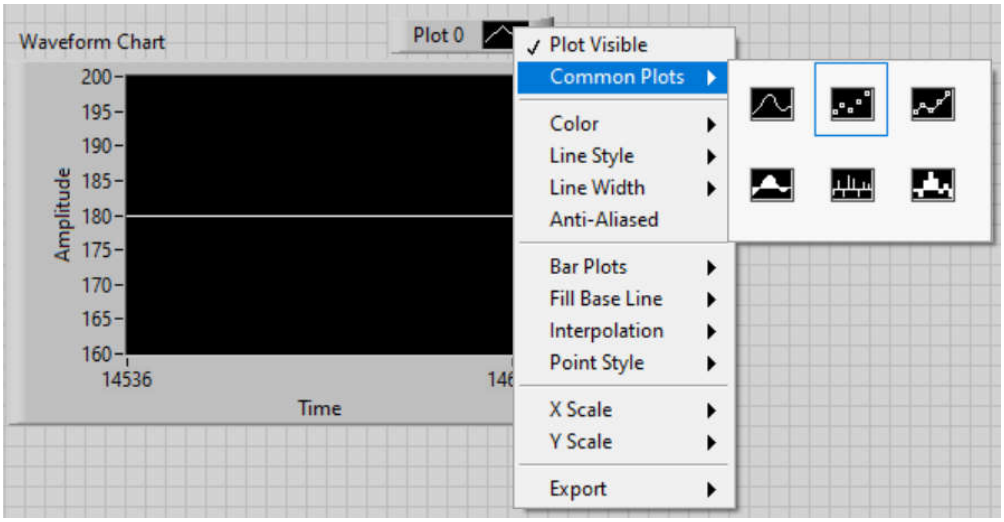
NAPOMENA1: Svaka kontrola i svaki indikator poseduju opcije podešavanja koje se pozivaju klikom na desni taster miša u *Front panel*-u, tj. *Block Diagram*-u. Programsko podešavanje opcija kontrola i indikatora će biti detaljno objašnjeno u Lekciji 3.4.

NAPOMENA2: Svaka promenljiva, funkcija i potprogram ima mogućnost prikaza „prozora za pomoć“ u kome se nalazi opis. Za funkcije i potprograme se u ovom prozoru nalazi i raspored ulaznih i izlaznih promenljivih.

8. Iz *Tools* palete izabrati „alat za povezivanje“ tj. ***Wiring Tool***  i povezati postavljenu kontrolu i indikator, Sl. 1.2.6. Uočiti da kontrola ima malu crnu strelicu sa desne strane terminala, a da indikator ima malu crnu strelicu sa leve strane terminala koje ukazuju na tok podataka odnosno njihovo „čitanje“ i „pisanje“, respektivno. Boja terminala i žica za numeričku kontrolu i indikator je narandžasta (narandžasta boja je rezervisana za *floating-point* tip podataka).

Sl. 1.2.6. Izgled *Block Diagram*-a sa numeričkom kontrolom *Numeric* i grafičkim indikatorom *Waveform Chart*

7. Pritiskom na ***Continuous Run Button*** , program će ući u kontinualan môd izvršavanja, sve dok se ne pritisne dugme ***Abort Execution*** . Dok se program izvršava, menjati vrednost kontrole pomoću ***Operating Tool***-a i pratiti promenu na indikatoru. Promeniti način prikaza tačaka na grafiku sa „linije“ na „tačkice“ klikom desnog tastera miša na *Plot* opciju *Waveform Chart*-a kao na Sl. 1.2.7. **NAPOMENA**: pri „linijskom“ prikazivanju vrednosti na grafiku, program spaja linijom diskretne vrednosti.

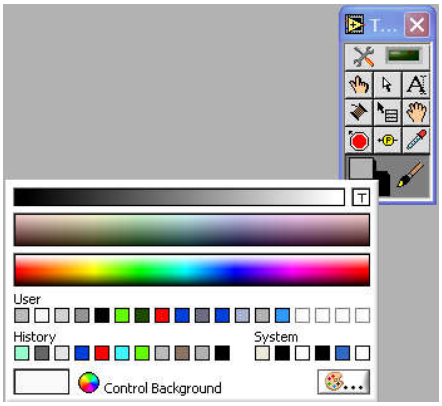


Sl. 1.2.7. Promena načina prikaza vrednosti na grafiku sa „linije“ na „tačkice“

8. Iz *Tools* palete izabrati ***Color Copy Tool*** i kliknuti crnu površinu *Waveform Chart* indikatora na *Front Panel*-u. Odgovarajuće boje (siva za okvir *Waveform Chart*-a i crna za površinu grafika) će se pojaviti u ***Coloring Tool***-u palete *Tools*.
9. Iz *Tools* palete izabrati ***Coloring Tool*** i po želji izmeniti boje interfejsa na sledeći način: 1) klikom levog tastera miša izabrati jedno od dva polja ***Coloring Tool***-a , 2) iz ponuđene palete boja izabrati željenu, Sl. 1.2.8, 3) klikom levog tastera miša izabrati površinu koju želite drugačije da obojite.
10. Sačuvati program kao *kontrola_indikator.vi*.

NAPOMENA 1: Korišćenjem opcije u paleti boja *Coloring Tool*-a, Sl. 1.2.8. je moguće objekat učiniti potpuno transparentnim.

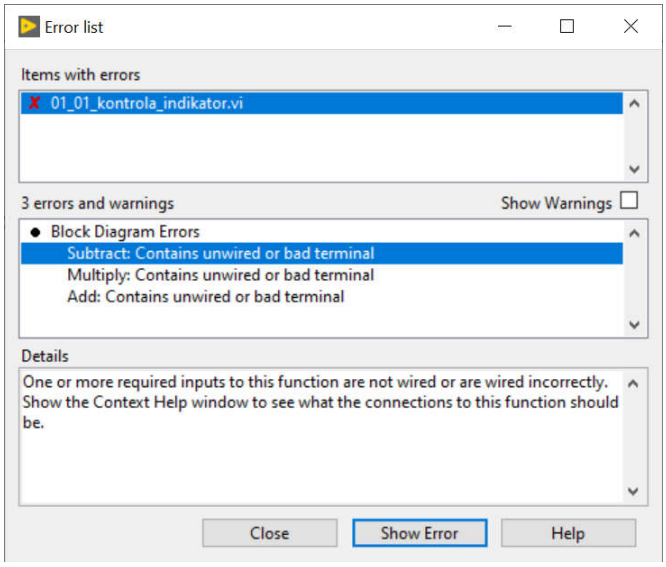
NAPOMENA 2: Selekcija i promena veličine elemenata se vrši pomoću alatke *Positioning/Resizing Tool* . Brisanje elemenata ili žice se vrši tako što se element ili žica selektuju, a potom pritisne taster *Delete* na tastaturi. Automatska selekcija alatki u *Tools* paleti se podešava klikom na opciju *Automatic Selection Tool* u *Tools* paleti.



Sl. 1.2.8. Selekcija boje u *Tools* paleti

Zadatak 1.2.2.
Modifikovati program iz prethodnog zadatka tako da na osnovu zadate temperature T i vrednosti konstanti R_0, T_0 i k izračunava vrednost $R=R_0(1+k(T-T_0))$, a rezultat prikazuje na grafičkom indikatoru.

1. Obrisati žicu koja povezuje *Numeric* kontrolu i *Waveform Chart* indikator tako što se najpre pomoću alatke *Positioning/Resizing Tool* selektuje žica, a potom pritisne dugme *Delete*.
2. U paleti ***Functions»Programming»Numeric*** pronaći funkcije *Add*, *Subtract* i *Multiply*, kao i numeričku konstantu *Numeric Constant*. Postaviti ih na *Block Diagram*. Uočiti da se u *Status Toolbar*-u pojavila tzv. slomljena strelica koja ukazuje na grešku u kôdu. Kliknuti levim tasterom miša na slomljenu strelicu. Pojaviće se prozor sa listom grešaka kao na Sl. 1.2.9.

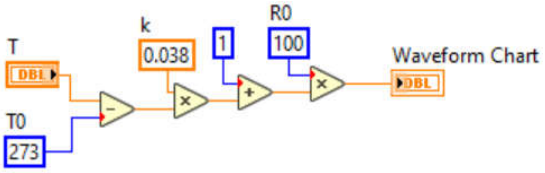


Sl. 1.2.9. Lista grešaka

3. Pogledati „prozore za pomoć“ za navedene elemente pomoću CTRL+H opcije. Povezati programski kôd kao na Sl. 1.2.10. Pomoću alatke *Labeling Tool* uneti vrednosti konstanti u *Block Diagram*-u kao i nazive oznaka (labela) *T*, *T0*, *k*, *R0*.

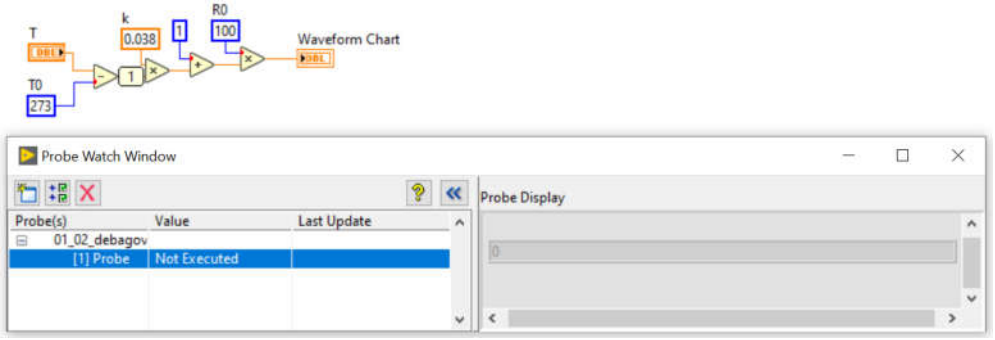
NAPOMENA 1: Kada se klikne desnim tasterom miša na kontrolu/indikator na *Front Panel*-u i potom izabere opcija *Visible Items*, moguće je selektovati vidljivost oznake („*Label*“) i natpisa („*Caption*“) kontrole/indikatora. Oznaka može biti vidljiva i u *Front Panel*-u i u *Block Diagram*-u, a natpis može biti vidljiv samo u *Front Panel*-u.

NAPOMENA 2: Pritiskom na *CTRL+Space* je moguće pozvati meni za brzo traženje funkcije prema njenom imenu (*Quick Drop* tj. brzo dodavanje).



Sl. 1.2.10.

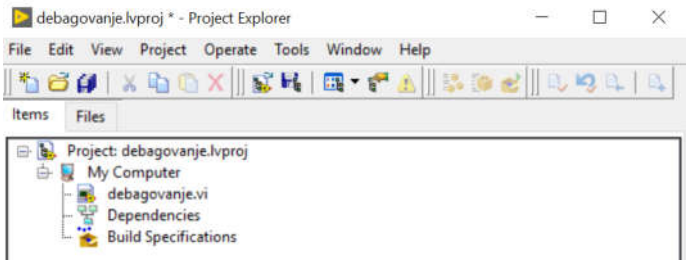
4. Pritiskom na opciju *Clean Up Diagram* u *Status Toolbar*-u *Block Diagram*-a obezbediti automatsko uređivanje programskog kôda. **NAPOMENA**: moguće je selektovati deo programskog kôda i samo na njega primeniti opciju „sređivanja“.
5. Pokrenuti izvršavanje programa pritiskom na dugme ***Continuous Run Button***. Uključiti opciju *Highlight Execution* i uočiti kako se u *Block Diagram*-u sada duž žica pojavljuju međurezultati. **NAPOMENA**: *LabVIEW* programiranje spada u kategoriju *dataflow* programiranja, tj. *LabVIEW* funkcije i potprogrami počinju da se izvršavaju tek kada su sve ulazne promenljive funkcije ili potprograma dostupne.
6. Iz *Tools* palete izabrati alatku *Probe Data* i postaviti je na žicu iza oduzimanja vrednosti *T* i *T0*. Pojaviće se *Probe Watch* prozor kao na Sl.1.2.11. Posmatrati kako se vrednost na izlazu funkcije za oduzimanje menja dok se na *Front Panel*-u pomoću ***Operating Tool***-a menja vrednost za *T*.



Sl. 1.2.11. *Probe Watch* prozor

7. Zaustaviti izvršavanje programa pritiskom na dugme ***Abort Execution***.
8. Uključiti opciju *Retain Wire Values* u *Status Toolbar*-u *Block Diagram*-a. Pokrenuti izvršavanje programa pritiskom na dugme ***Continuous Run Button***. Promeniti vrednost numeričke kontrole *T*. Zaustaviti izvršavanje programa pritiskom na dugme ***Abort Execution***. Iz *Tools* palete izabrati alatku *Probe Data* i postaviti po jednu na izlazu svake od funkcija u programu. Uočiti da je zahvaljući uključenju opcije *Retain Wire Values* sada moguće pratiti poslednje izračunate vrednosti na izlazu svake od funkcija i nakon završetka rada programa.

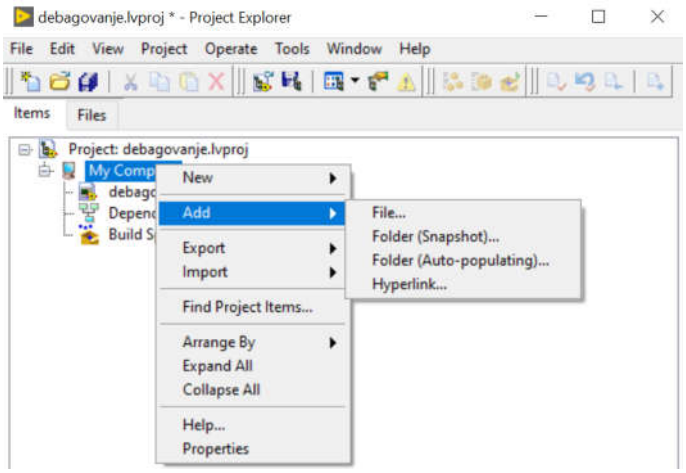
9. Iz *Tools* palete izabrati alatku *Breakpoint* . Postaviti je na proizvoljnu žicu u *Block Diagram*-u. Pokrenuti izvršavanje programa pritiskom na dugme ***Continuous Run Button*** . Primetiti kako će izvršavanje kôda biti pauzirano na mestu gde je „tačka prekida“ postavljena. Pritiskom na opciju *Continue* u *Status Toolbar*-u *Block Diagram*-a nastaviće se izvršavanje programa.
10. Okruženje omogućava tzv. „korak po korak“ debagovanje. Za pokretanje ove vrste testiranja izabrati opciju *Start Single Stepping* u *Status Toolbar*-u *Block Diagram*-a. Pomoću *Step Into* i *Step Over* opcija je moguće „ući u“ funkciju i „završiti“ izvršavanje pojedinih funkcija respektivno, a pomoću opcije *Finish Block Diagram* se završava izvršavanje programa. Isprobati navedene opcije i završiti izvršavanje programa.
11. Sačuvati program kao novi program *debagovanje.vi*.
12. U praksi, rešenja obično sadrže, osim glavnog programa, i niz potprograma i prateću dokumentaciju. Zbog toga većina softverskih okruženja, uključujući i *LabVIEW*, nudi čuvanje rešenja u vidu tzv. projektnih datoteka (eng. *project file*). Da bi se trenutni program *debagovanje.vi* sačuvao u okviru projektne datoteke izabrati opciju ***File»Create Project...»Blank Project»Finish*** i potvrditi da se trenutno otvoreni *debagovanje.vi* doda („Add“) u kreirani projekat. Pojaviće se projektna datoteka kao na Sl. 1.2.12.



Sl. 1.2.12. Projektna datoteka

13. Sačuvati projektu datoteku kao *debagovanje.lvproj*.

NAPOMENA: U projektnu datoteku je moguće dodati datoteke, hiperlinkove ili je u njoj moguće kreirati direktorijume izborom opcije ***MyComputer»Add***, Sl. 1.2.13. Direktorijumi u okviru projekta mogu biti virtuelni (*Snapshot*) ili stvarni (*Auto-populating*), tj. mogu da budu samo logička odrednica ili da zaista postoje na hard disku, respektivno.



Sl. 1.2.13. Dodavanje elemenata u projektnu datoteku

Lekcija 2 – Korišćenje struktura

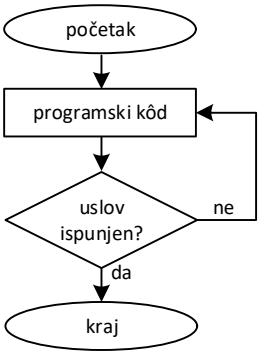
Cilj

Cilj lekcije je da studenti savladaju:

- korišćenje *While* petlje
- primenu sekvencijalnog programiranja
- kontrolisanje i merenje vremena
- korišćenje *For* petlje
- memorisanje promenljivih u petlji
- korišćenje *Case* strukture, tj. strukture za odlučivanje
- korišćenje koncepta „događajem“ vođenog programiranja.

2.1 While petlja

Tok izvršavanja programa sa *While* petljom (*While Loop*) je prikazan na Sl. 2.1.1 Programski kôd unutar *While* petlje se izvršava sve dok je „uslov“ izvršavanja ispunjen. U slučaju kada u prvoj iteraciji *While* petlje „uslov“ nije ispunjen, programski kôd se izvršava jedanput, tj. **programski kôd unutar *While* petlje se izvršava NAJMANJE JEDNOM.**



Sl. 2.1.11 Tok izvršavanja *While* petlje

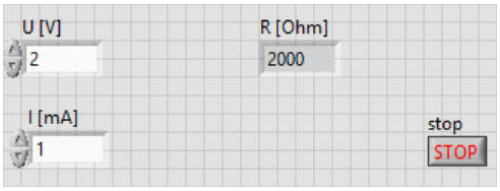
Zadatak 2.1.1.
Kreirati program koji na osnovu unetih vrednosti napona U [V] i struje I [mA] na otporniku izračunava otpornost R [Ω] i koja se zaustavlja pritiskom na dugme STOP.

1. Pokrenuti ***NI LabVIEW*** softver izborom opcije: ***Start»National Instruments»LabVIEW 2018 (32-bit)*** ili klikom na ***NI LabVIEW*** prečicu na *Desktop*-u.
2. Kreirati novi **.vi** program selekcijom opcije ***File»New VI***. Radi preglednosti, podesiti da se *Front panel* i *Block diagram* prikazuju u gornjem i donjem delu ekrana, respektivno, pomoću opcije ***Window»Tile Up and Down***.

3. Otvoriti palete ***Tools***, ***Controls*** i ***Functions*** selekcijom ***View»Tools Palette***, ***View»Controls Palette*** u *Front panel*-u, tj. ***View»Functions Palette*** u *Block diagram*-u.
4. Izabrati *While Loop* u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u klikom levog tastera miša razvući *While* petlju kao na Sl. 2.1.2. Uočiti na Sl. 2.1.2. brojač iteracija *While* petlje (*Iterator*) i uslovni terminal (*Conditional terminal*).

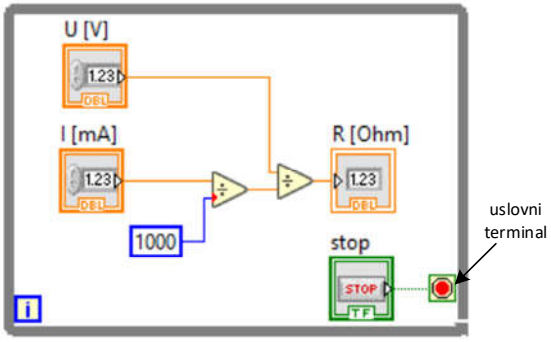
Sl. 2.1.2. *While* petlja

5. Na *Front Panel*-u, iz palete ***Controls»Modern»Numeric*** kreirati numeričke kontrole (*Numeric Control*) za unos vrednosti napona i struje, kao i numerički indikator (*Numeric Indicator*) za prikaz vrednosti otpornosti. Pomoću alatke *Labeling Tool* u *Tools* paleti uneti odgovarajući tekst za oznake kontrola i indikatora kao na Sl. 2.1.3.
   Na *Front Panel*-u, iz palete ***Controls»Modern»Boolean»Stop Button*** kreirati logičku kontrolu STOP, Sl. 2.1.3.



Sl. 2.1.3. *Front panel* programa za izračunavanje vrednosti otpornosti

6. U *Block Diagram*-u kreirati programski kôd koji izračunava vrednost otpornosti na osnovu vrednosti napona i struje koje korisnik unosi na *Front Panel*-u kao što je prikazano na Sl. 2.1.4. Funkciju deljenja (*Divide*) i numeričku konstantu (*Numeric Constant*) uneti u *Block Diagram* iz palete ***Functions»Programming»Numeric***. Logičku kontrolu STOP povezati na uslovni terminal.



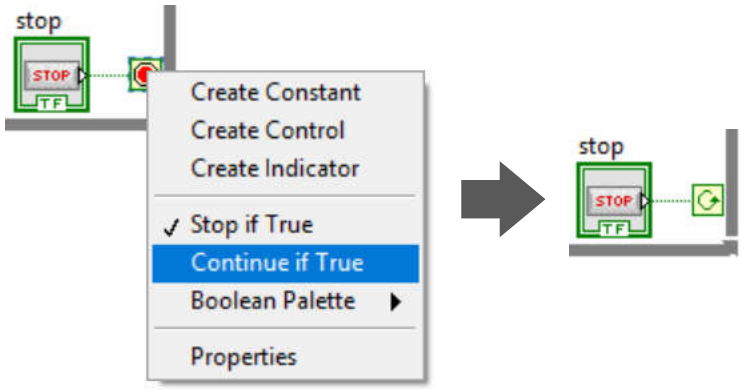
Sl. 2.1.4. *Block diagram* programa za izračunavanje vrednosti otpornosti

Uočiti da se na ulazu funkcije deljenja, na mestu gde je povezana numerička konstanta 1000, nalazi crvena tačka (*Coercion dot*). Ova tačka označava automatsku implicitnu konverziju usled različitih tipova podataka koji su dovedeni na ulaz numeričke funkcije (u ovom slučaju je celobrojni podatak konvertovan u realan broj).

7. Pokrenuti izvršavanje programa pritiskom na dugme ***Run*** u *Status Toolbar*-u. Menjati vrednosti numeričkih kontrola i pratiti promenu vrednosti numeričkog indikatora. Završiti izvršavanje programa pritiskom na dugme STOP. Kada dugme STOP nije pritisnuto, tj. kada je vrednost logičke kontrole STOP jednako *False*, iteracije *While* petlje se izvršavaju. Kada korisnik pritisne dugme STOP, vrednost te logičke promenljive postaje *True*, ova vrednost se prosleđuje uslovnom terminalu i izvršavanje *While* petlje se prekida.
8. Pokrenuti izvršavanje programa pritiskom na dugme ***Continous Run*** u *Status Toolbar*-u. Pokušati da se zaustavi izvršavanje programa pritiskom na dugme STOP. Zašto ne uspeva? Uključiti dugme ***Execution Highlighting*** i zaključiti zašto program ne može da se zaustavi pritiskom na dugme STOP. Zaustaviti izvršavanje programa pritiskom na dugme ***Abort Execution***.

NAPOMENA: Nije dobra praksa koristiti opciju ***Continous Run*** za ponavljanje izvršavanja programskog kôda. Kontinualnost izvršavanja treba omogućiti primenom *While* petlje, a za pokretanje programa koristiti opciju ***Run***.

9. Sačuvati program kao *Program_za_izracunavanje_vrednosti_otpornosti.vi*.
10. Pozicionirati miša iznad uslovnog terminala i kliknuti desnim tasterom. Izabrati opciju *Continue if True* kao na Sl. 2.1.5. Pokrenuti izvršavanje programa pritiskom na dugme ***Run***. Koliko se puta sada izvršava program i zašto?



Sl. 2.1.5. Promena tipa „uslova“ završetka *While* petlje

11. Zatvoriti progam bez čuvanja izmene iz tačke 10.

2.2 Sekvencijalno programiranje

Zadatak 2.2.1.
Kreirati program koji se sastoji iz tri sekvence:

1) inicijalizacije u kojoj se na indikatoru prikazuje natpis "Program pocinje da se izvrsava"
2) glavnog dela programa u kome se nalazi *While* petlja koja se završava pritiskom na dugme STOP i koja u toku izvršavanja prikazuje tekuću vrednost brojača iteracija na numeričkom indikatoru
3) zatvaranja *LabVIEW* okruženja.

1. Kreirati nov **.vi** program.
2. Izabrati *Flat Sequence* strukturu u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u klikom levog tastera miša razvući *Flat Sequence* strukturu kao na Sl. 2.2.1.

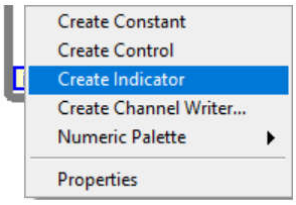
Sl. 2.2.1. *Flat Sequence* struktura

3. *Flat Sequence* struktura sa Sl. 2.2.1 ima samo jednu sekvencu. Dodati jednu sekvencu ispred nje i jednu iza nje tako što se miš pozicionira na gornju ivicu strukture, potom se klikne desnim tasterom miša i izaberu se opcije *Add Frame Before* i *Add Frame After*, respektivno. Na Sl. 2.2.2. je prikazana *Flat Sequence* struktura sa tri sekvence.

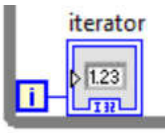
Sl. 2.2.2. *Flat Sequence* struktura sa tri sekvence

4. U centralnoj sekvenci kreirati *While* petlju koja se završava pritiskom na dugme STOP kao što je objašnjeno u Poglavlju 2.1.
5. Kreirati numerički indikator koji će prikazivati redni broj tekuće iteracije *While* petlje. Numerički indikator se može kreirati ili pomoću selekcije u ***Functions»Programming»Numeric*** paleti ili tzv. prečicom tako što se miš postavi iznad brojača iteracija, klikne se desnim tasterom miša i u padajućem

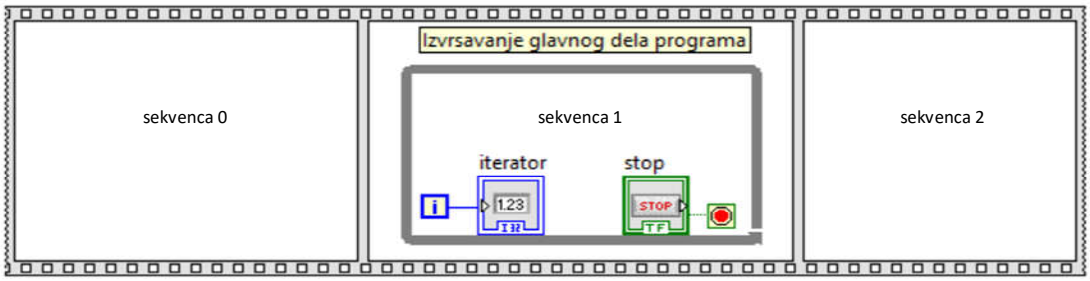
meniju izabere opcija ***Create Indicator*** kao na Sl. 2.2.3. Uočiti da je pri pravljenju indikatora „prečicom“ format podataka automatski podešen da bude I32 (*signed integer 32 bit*-a), tj. ne mora se podešavati u opciji *Properties* indikatora (podsetnik iz Lekcije 1: opcija *Properties* indikatora se otvara klikom na desni taster miša na terminal indikatora u *Block Diagram*-u). Oznaku indikatora nazvati „iterator“ kao što je to prikazano na Sl. 2.2.4 pomoću alatke *Labeling Tool* u *Tools* paleti. Izgled centralne sekvence sa *While* petljom je prikazan na Sl. 2.2.5.



Sl. 2.2.3. Kreiranje indikatora „prečicom“

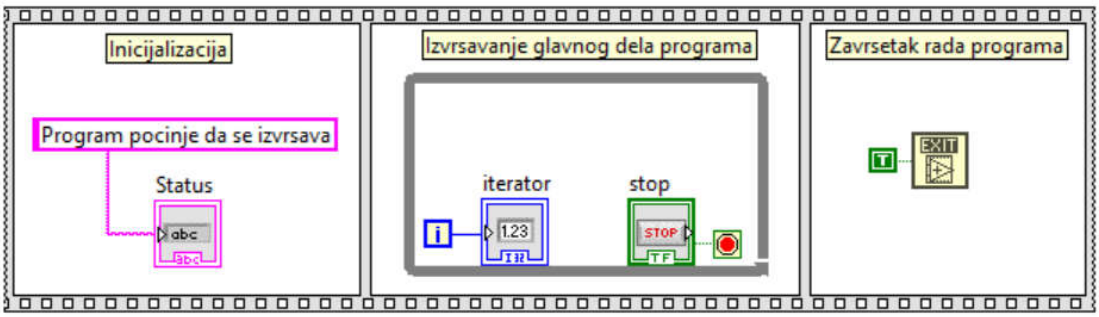


Sl. 2.2.4. Kreiranje numeričkog indikatora za prikaz vrednosti iteratora *While* petlje



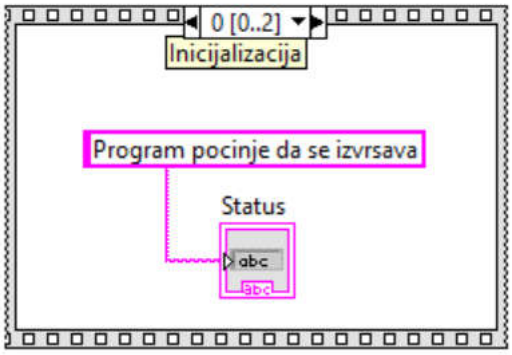
Sl. 2.2.5. Centralna sekvenca sa *While* petljom

6. Pokrenuti izvršavanje programa. Uočiti da se vrednost indikatora „iterator“ menja velikom brzinom (jedna iteracija kreirane *While* petlje kratko traje, ~1 ms).
7. U „sekvenci 0“ kreirati indikator sa oznakom „Status“ (***Controls»Modern»String & Path»String indicator***) koji prikazuje vrednost znakovne konstante “Program pocinje da se izvrsava” (***Functions»Programming»String»String Constant***), Sl. 2.2.6.
8. U „sekvenci 2” uneti funkciju za zatvaranje *LabVIEW* okruženja (***Functions»Programming»Application Control»Quit LabVIEW***), Sl. 2.2.6. Komentari “Inicijalizacija”, “Izvrsavanje glavnog dela programa” i “Zavrsetak rada programa” se unose pomoću alatke *Labeling Tool* u *Tools* paleti.



Sl. 2.2.6. *Block Diagram* programa koji ilustruje primenu sekvencijalnog programiranja

9. Sačuvati program kao *Sekvencijalno programiranje.vi*.
10. Pokrenuti izvršavanje programa. Najpre će se izvršiti prva sekvenca koja ispisuje na indikatoru „Program pocinje da se izvrsava“, a potom će se izvršavati druga sekvenca u kojoj radi *While* petlja. Nakon pritiska na dugme STOP program ulazi u treću sekvencu koja zatvara *LabVIEW* okruženje.
11. Otvoriti sačuvani program *Sekvencijalno programiranje.vi*. U *Block Diagram*-u pozicionirati miša na gornju ivicu *Flat Sequence* strukture, kliknuti desnim tasterom miša i iz padajućeg menija izabrati *Replace with Stacked Sequence*. Rezultat zamene je *Stacked Sequence* struktura kao na Sl. 2.2.7. Uočiti da se na gornjoj ivici sekvence nalazi meni u kome se klikom miša na levu/desnu strelicu (ili izborom iz padajućeg menija) može selektovati prikaz sekvence 0, 1 ili 2.



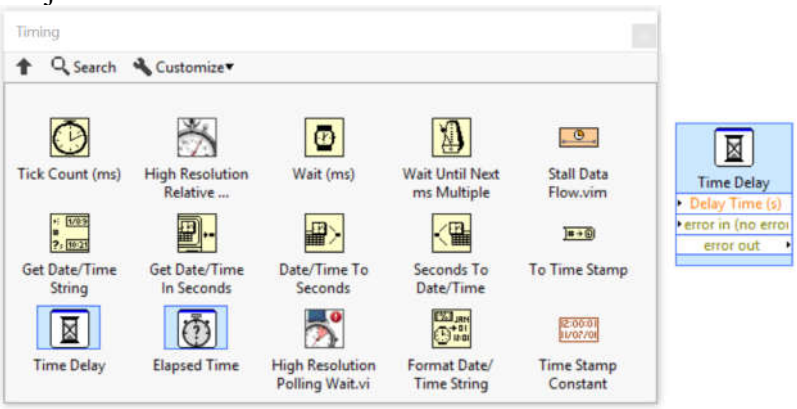
Sl. 2.2.7. *Block Diagram* programa sa *Stacked Sequence* strukturom

12. Zatvoriti progam bez čuvanja izmene iz tačke 11.

NAPOMENA: Zbog preglednosti, dobra praksa je da ceo *LabVIEW* kôd zauzima JEDAN ekran u *Block Diagram*-u. *Stacked Sequence* zauzima manje mesta od *Flat Sequence* strukture, pa se preferira njeno korišćenje u situacijama kada kôd zauzima puno mesta. Takođe, dobra praksa je da programi imaju tri sekvence (inicijalizaciju, glavni kôd i završnu sekvencu). U završnoj sekvenci se obično dealocira zauzet memorijski prostor, prikazuju greške, gasi *LabVIEW* okruženje i sl. o čemu će biti reči i u narednim lekcijama.

2.3 Kontrolisanje i merenje vremena

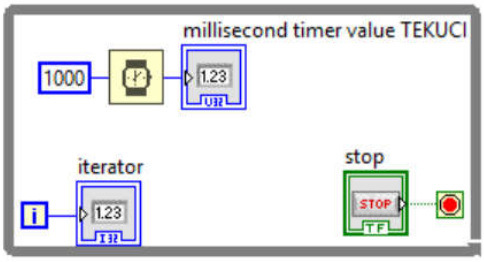
U primerima iz prethodna dva poglavlja, iteracije *While* petlje se izvršavaju jedna za drugom, bez pauze izmedju, a vreme izvršavanja jedne iteracije *While* petlje je reda veličine milisekunde. U prethodnim primerima, procesorsko vreme je zauzeto neprekidnim ponavljanjem iteracija. Izvršavanje iteracija *While* petlje se može usporiti, a procesorsko vreme osloboditi za rad na drugim zadacima pomoću korišćenja *Timing* funkcija (*Functions»Programming»Timing*): *Wait (ms)*, *Wait Until Next ms Multiple* i *Time Delay*, Sl. 2.3.1. *Express Time Delay* funkcija je ekvivalentna *Wait (ms)* funkciji s tom razlikom što omogućava propagaciju greške jer ima ulazne i izlazne priključke za klaster greške (*error in*, *error out*, Sl. 2.3.1.). O klasterima će biti reči u Lekciji 3.



Sl. 2.3.1. *Timing* funkcije (levo) i *Express Time Delay* funkcija (desno)

Zadatak 2.3.1.
Kreirati program koji omogućava da se programski kôd unutar *While* petlje izvršava svakih 1000 ms (pod uslovom da izvršavanje samog programskog kôda traje manje od 1000 ms). Izmeriti proteklo vreme od pokretanja programa do završetka rada *While* petlje.

1. Kreirati nov **.vi** program.
2. Kreirati *While* petlju koja se završava pritiskom na dugme STOP. Kreirati numerički indikator za prikaz tekuće vrednosti brojača iteracija. Unutar *While* petlje postaviti funkciju *Wait (ms)* (***Functions»Programming»Timing***). Na ulazu *milliseconds to wait* funkcije *Wait (ms)* postaviti numeričku konstantu 1000 (jer je uslov zadatka da jedna iteracija traje 1000 ms). Izlaz *millisecond timer value* funkcije *Wait (ms)* povezati na numerički indikator. *Block Diagram* je prikazan na Sl. 2.3.2.



Sl. 2.3.2. Primena *Wait (ms)* funkcije za diktiranje vremena trajanja iteracije *While* petlje

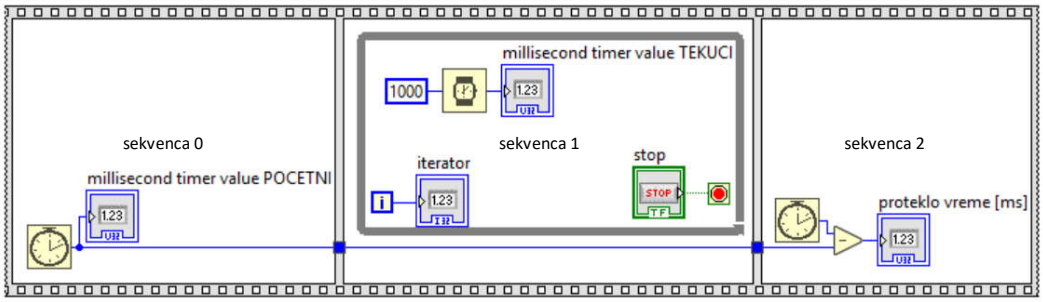
3. Pokrenuti izvršavanje programa. Uočiti da se vrednost iteratora menja svake sekunde.

Operativni sistem ima sistemski sat i vrednost sistemskog sata se prikazuje u milisekundama na indikatoru *millisecond timer value TEKUCI*. Sistemski sat počinje da broji od 0 pri startu operativnog sistema i može da broji sve do 2^{32}-1 nakon čega se resetuje i nastavlja da broji od 0.

U primeru sa Sl. 2.3.2. svaka iteracija *While* petlje traje ukupno 1000 ms i obuhvata: trajanje izvršavanja programskog kôda unutar *While* petlje i „čekanje“ do 1000 ms.

NAPOMENA: Ukoliko je samo trajanje izvršavanja programskog kôda duže od „čekanja“ koje diktira funkcija *Wait (ms),* vreme izvršavanja iteracije *While* petlje biće diktirano trajanjem izvršavanja programskog kôda.

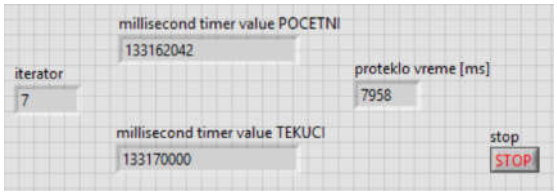
4. Oko *While* petlje razvući *Flat Sequence* strukturu, a potom kreirati jednu sekvencu pre i jednu sekvencu posle sekvence u kojoj je *While* petlja (prema instrukcijama u Poglavlju 2.2.), Sl. 2.3.3.



Sl. 2.3.3.

5. U sekvenci 0 i sekvenci 2 postaviti po jednu funkciju *Tick Count (ms)* koja na svom izlazu daje vreme sistemskog sata u milisekundama. U sekvenci 2 oduzeti vrednosti sa izlaza funkcija *Tick Count (ms)* kao na Sl. 2.3.3. Razlika će predstavljati proteklo vreme u milisekundama. Postaviti numeričke indikatore vremena sistemskog sata u sekvenci 0 i proteklog vremena u sekvenci 2 kao na Sl. 2.3.3. Uočiti da se podatak iz sekvence 0 može koristiti i u sekvenci 2.
6. Pokrenuti izvršavanje programa. Nakon zaustavljanja *While* petlje na indikatoru proteklog vremena će se prikazati koliko je vremena proteklo u milisekundama. Praktično je proteklo vreme jednako: (iterator+1)*1000 ms+vreme izvršavanje sekvenci 0 i 2.
7. Sačuvati program kao *Wait.vi*.
8. Umesto funkcije *Wait (ms)* postaviti funkciju *Wait Until Next ms Multiple***.** Najjednostavniji način da se ova zamena uradi je da se stane mišem iznad funkcije *Wait (ms)*, klikne desnim tasterom miša i iz padajućeg menija izabere opcija ***Replace»Timing Palette»Wait Until Next ms Multiple***.
9. Pokrenuti izvršavanje programa. Nakon zaustavljanja *While* petlje na indikatoru proteklog vremena će se prikazati koliko je vremena proteklo u milisekundama. Funkcija *Wait Until Next ms Multiple* sinhronizuje rad *While* petlje sa sistemskim satom, tj. „čeka“ da sistemski sat bude jednak celobrojnom umnošku zadane ulazne vrednosti (u ovom slučaju 1000 ms) i onda nastavlja sa sledećom iteracijom. Primer izgleda *Front Panela* i vrednosti indikatora je prikazan na Sl. 2.3.4. Početna vrednosti sistemskog sata je 133162042 ms. U prvoj iteraciji *While* petlje (iterator=0) „čeka“ se 958 ms tj. čeka se da sistemski sat ne odbroji do 133163000 ms što je prvi broj posle početne vrednosti koji je celobrojni

umnožak od 1000 ms. Nakon toga se ulazi u sledeću iteraciju (iterator=1), čeka se 1000 ms i tako redom. Proteklo vreme će u ovom slučaju biti 958 ms+iterator*1000 ms+vreme izvršavanja sekvenci 0 i 2.



Sl. 2.3.4.

10. Sačuvati program kao *Wait_Until.vi*.
11. Zatvoriti progam.

NAPOMENA: *Tick Count (ms)* funkcija broji od 0 do 2^{32}-1, a potom se resetuje i broji ponovo od 0. Zbog toga se za brojanje koje dugo traje (duže od dva meseca) ne koristi ova funkcija već se umesto nje koristi funkcija *Get Data/Time in Seconds* koja na svom izlazu daje proteklo vreme u sekundama u odnosu na 01.01.1904. 12:00 a.m.

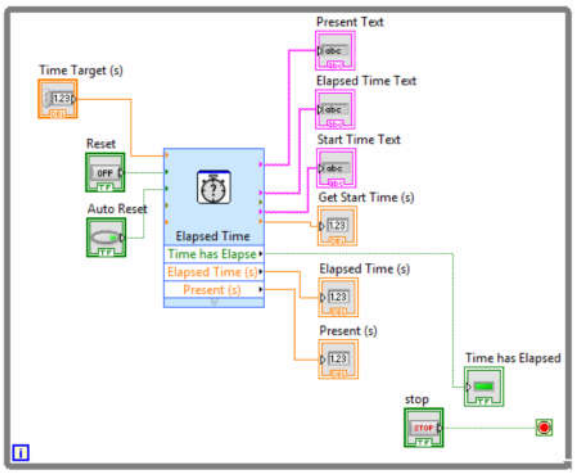
Zadatak 2.3.2.

Koristeći *Express Elapsed Time* funkciju kreirati program koji meri proteklo vreme i:

- kome se može zadati vreme koje treba izmeriti
- koji može resetovati merenje vremena pritiskom na dugme na interfejsu
- koji može automatski resetovati merenje vremena nakon isteka zadatog vremena

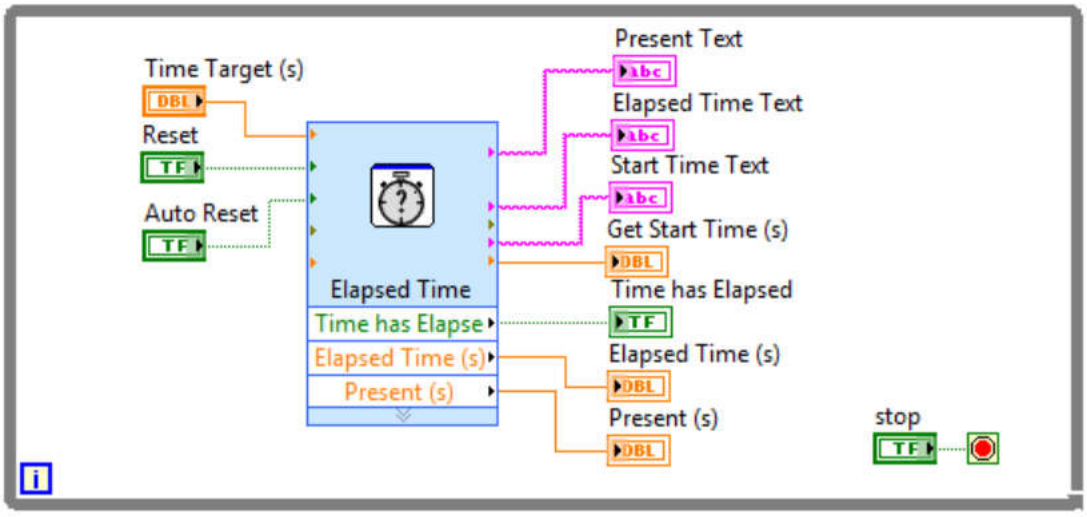
Modifikovati program tako da samostalno završi izvršavanje nakon isteka zadatog vremena.

1. Kreirati nov **.vi** program.
2. Kreirati *While* petlju koja se završava pritiskom na dugme STOP. Unutar *While* petlje postaviti funkciju *Elapsed Time* (***Functions»Programming»Timing***). Za ulaze/izlaze ove funkcije kreirati odgovarajuće kontrole/indikatore kao što je to prikazano na Sl. 2.3.5. (najjednostavniji način je da se uradi klik desnim tasterom miša na odgovarajući ulaz/izlaz funkcija, a potom se selektuje opcija *Create Control/Indicator*).



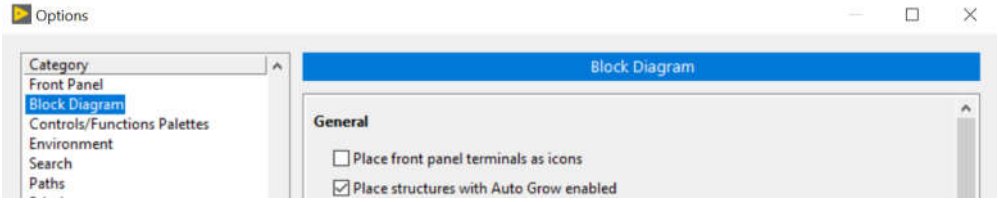
Sl. 2.3.5.

3. Sačuvati program kao *Merenje_proteklog_vremena.vi*.
4. Uočiti da su svi terminali prikazani kao ikone na *Block Diagram*-u. Kada se na terminal klikne desnim tasterom miša i deaktivira opcija *View as Icon*, terminali postaju manji, a *Block Diagram* pregledniji. Na Sl. 2.3.6. je prikazan *Block Diagram* ekvivalentan onome sa Sl. 2.3.5. pri čemu terminali nisu prikazani kao ikone.



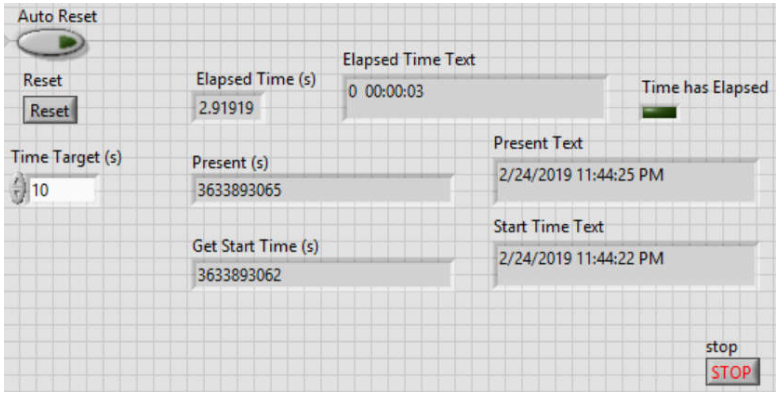
Sl. 2.3.6.

NAPOMENA: prikaz terminala u obliku ikona može učiniti prikaz kôda na *Block Diagram*-u prilično nepreglednim. Zato je preporučljivo u meni liniji u *Tools/Options/Block Diagram/* deaktivirati opciju *Place front panel terminals as icons*, Sl. 2.3.7.



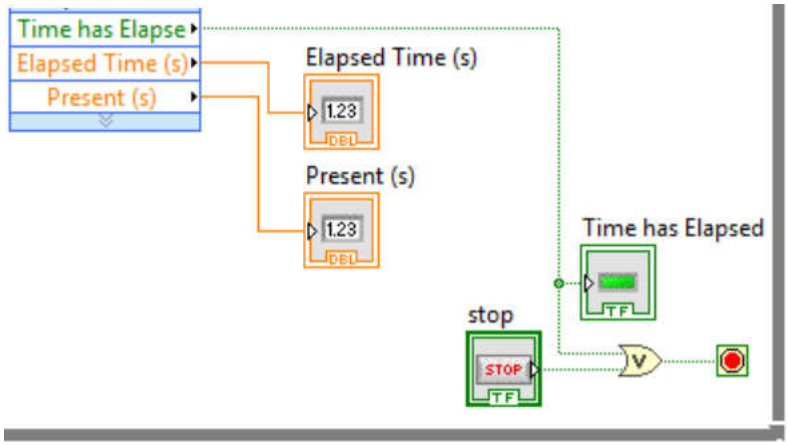
Sl. 2.3.7. Podešavanje u Tools/Options/Block Diagram/ da se terminali ne prikazuju kao ikone

5. *Front Panel* programa je prikazan na Sl. 2.3.8. Voditi računa da se za indikatore koji prikazuju vremena u sekundama omogući prikaz dovoljnog broja značajnih cifara (klik desnim tasterom miša na indikator, opcija ***Properties»Display Format***).



Sl. 2.3.8. Izgled *Front Panel*-a programa za merenje proteklog vremena

6. Uneti vreme koje treba da se izmeri u kontrolu *Time Target (s)* (npr. 10 s) Pokrenuti izvršavanje programa. Pritiskom na dugme *Reset* merenje vremena se resetuje. Ako je pritisnuto dugme *Auto Reset*, kada se završi merenje vremena zadatog kontrolom *Time Target (s)*, program počinje da ponovo meri od 0 s.
7. Konačno, treba modifikovati program tako da se završi njegovo izvršavanje kada istekne zadato vreme. Modifikacija se postiže tako što se uslovni terminal *While* petlje poveže na izlaz logičkog ILI kola kome su ulazni priključci STOP dugme i *Time has Elapse* izlaz *Express Elapsed Time* funkcije, Sl. 2.3.9.

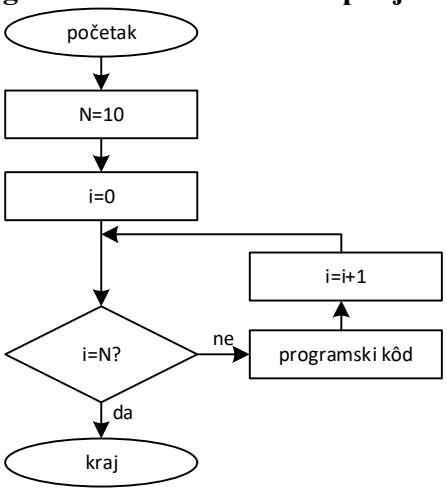


Sl. 2.3.9.

8. Pokrenuti izvršavanje modifikovanog programa.
9. Sačuvati program kao *Merenje_proteklog_vremena_zaustavi_program.vi*.

2.4 For petlja

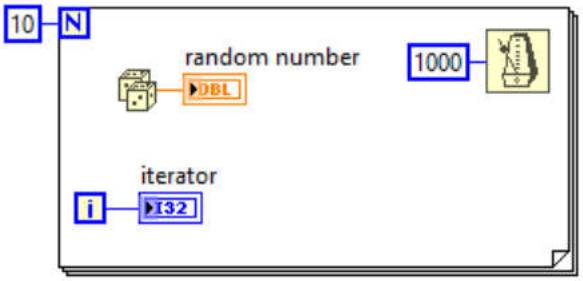
Tok izvršavanja programa sa *For* petljom (*For Loop*) je prikazan na Sl. 2.4.1 Programski kôd unutar *For* petlje počinje da se izvršava ako je „uslov“ izvršavanja ispunjen. U slučaju kada u prvoj iteraciji *For* petlje „uslov“ nije ispunjen, programski kôd se ne izvršava, tj. **programski kôd unutar *For* petlje može i da se NE IZVRŠI.**



Sl. 2.4.1. Tok izvršavanja *For* petlje

Zadatak 2.4.1.
Kreirati program koji prikazuje deset slučajnih brojeva u vremenskim intervalima od 1 s. Modifikovati program tako da se može završiti pritiskom na dugme STOP i pre završetka prikaza slučajnih brojeva.

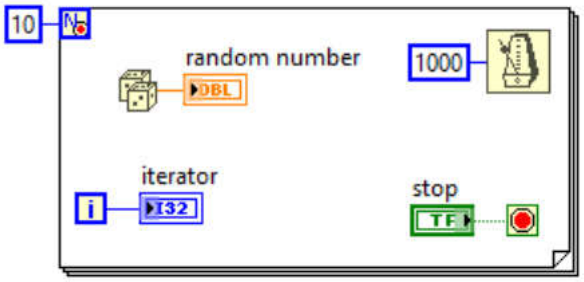
1. Kreirati nov **.vi** program.
2. Izabrati *For Loop* u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u klikom levog tastera miša razvući *For* petlju kao na Sl. 2.4.2. Uočiti na Sl. 2.4.2. iterator i (*Iteration Terminal*) i brojač N (*Count Terminal*) *For* petlje. Kreirati numeričku konstantu koja ima vrednost 10 (broj zadatih iteracija) i povezati je na brojač N *For* petlje. Kreirati numerički indikator za prikaz iteratora *For* petlje, a pomoću funkcije *Wait Until Next ms Multiple* omogućiti sinhronizaciju sa sistemskim satom i izvršavanje iteracija na 1 s. Iz palete ***Functions»Mathematics*** izabrati funkciju *Random Number 0-1* za generisanje slučajnih brojeva, uneti je u *For* petlju i povezati je na numerički indikator kao na Sl. 2.4.2.



Sl. 2.4.2. *For* petlja

NAPOMENA: Brojač (*Count Terminal*) *For* petlje je tipa I32 (*signed integer 32-bit*-a). U slučaju da se drugačiji tip podataka poveže na brojač, taj tip podatka će biti konvertovan u I32.

3. Pokrenuti izvršavanje programa.
4. Kada se završi izvršavanje programa, modifikovati ga tako da može da se završi pritiskom na dugme STOP i pre nego što iterator dostigne zadati broj iteracija. Kliknuti desnim tasterom miša na ivicu *For* petlje i aktivirati opciju *Condition Terminal*. Slično *While* petlji, u donjem desnom uglu petlje se pojavljuje „uslovni" terminal za koji treba napraviti logičku kontrolu STOP, Sl. 2.4.3. Primetiti da je uvođenjem uslovnog terminala brojač *For* petlje promenio svoj izgled (na brojaču se pojavila crvena tačka) ukazujući na to da broj realizovanih iteracija više neće eksplicitno zavisiti od brojne vrednosti koja je brojaču dodeljena.



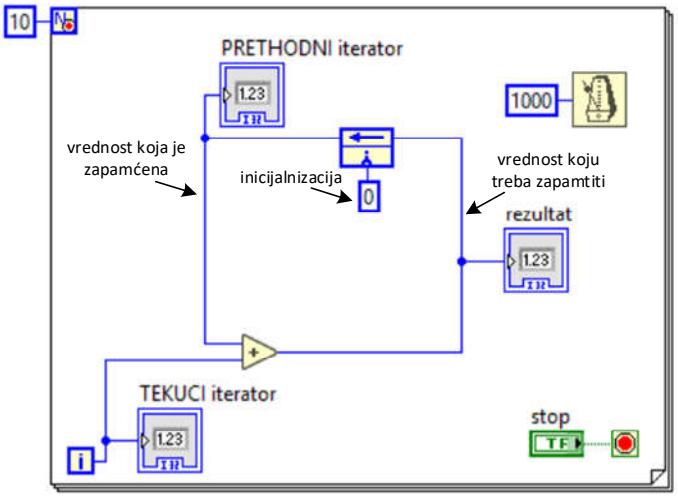
Sl. 2.4.3. *For* petlja sa *Conditional Terminal*-om

5. Pokrenuti izvršavanje programa. Pritiskom na dugme STOP se može zaustaviti izvršavanje *For* petlje, i pre završetka prikaza slučajnih brojeva.
6. Sačuvati program kao *For_petlja.*

2.5 Memorisanje promenljivih u petlji

Zadatak 2.5.1.
Kreirati program koji privremeno memoriše poslednju vrednost promenljive unutar *While* ili *For* petlje.

1. Otvoriti program *For_petlja*. Izbrisati deo programskog kôda koji generiše slučajne brojeve i prikazuje ih na indikatoru (pomoću *Position/Size/Select* alatke  selektovati kôd i kliknuti na *Delete* taster na tastaturi).
2. Iz palete ***Functions»Programming»Structures*** izabrati strukturu *Feedback Node*. Ova struktura omogućava memorisanje vrednosti promenljive iz prethodne iteracije kako bi ona mogla da se upotrebi za evaluaciju (konkretno u ovom primeru za sabiranje) u narednoj iteraciji.
3. U *Block Diagram*-u kreirati kôd kao na Sl. 2.5.1.



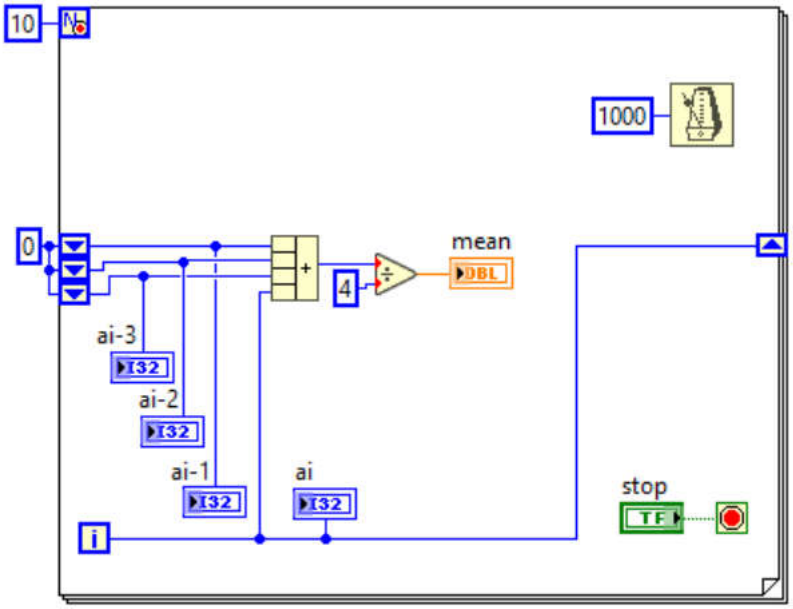
Sl. 2.5.1. *For* petlja sa *Feedback Node*-om

4. Pokrenuti izvršavanje programa. U početnoj iteraciji *For* petlje (i=0), *Feedback Node* je inicijalizovan vrednošću 0. U sledećoj iteraciji je u njemu zapamćena vrednost rezultata sabiranja, Sl. 2.5.1.
5. Sačuvati program sa *File/Save As...* kao *Feedback_node.vi*.
6. Zameniti *For* petlju *While* petljom tako što se miš pozicionira na ivicu *For* petlje, a potom se klikne desnim tasterom i iz padajućeg menija izabere opcija *Replace with While Loop*. Pokrenuti izvršavanje programa. Primetiti da predloženi programski kôd može da se modifikuje tako da računa sumu prvih *N* prirodnih brojeva (kako treba modifikovati kôd?).
7. Zatvoriti progam bez čuvanja izmene iz tačke 6.

Zadatak 2.5.2.
Kreirati program koji privremeno memoriše četiri poslednje vrednosti promenljive unutar *While* ili *For* petlje i računa srednju vrednost.

1. Otvoriti program *For_petlja*. Izbrisati deo programskog kôda koji generiše slučajne brojeve i prikazuje ih na indikatoru (pomoću *Position/Size/Select* alatke  selektovati kôd i kliknuti na *Delete* taster na tastaturi).

2. Kliknuti desnim tasterom miša na desnu ili levu ivicu *For* petlje i izabrati opciju *Add Shift Register*. *Shift* registar sa leve strane se može „razvući“ u proizvoljan broj mesta (klikne se levim tasterom miša na donju ivicu levog *Shift* registra, i povuče se nadole), Sl. 2.5.2. Povezati na desni *Shift* registar vrednost koju treba zapamtiti u sledećoj iteraciji. Vrednosti iz levog niza *Shift* registara povezati *Compound Arithmetic* funkcije kako bi se izračunao zbir, a potom i srednja vrednost, Sl. 2.5.2. Vrednosti levog *Shift* registra inicijalizovati vrednostima 0.



Sl. 2.5.2.

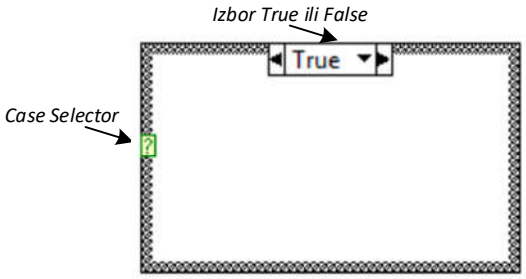
3. Pokrenuti izvršavanje programa. U početnoj iteraciji *For* petlje (i=0), *Shift registar* je inicijalizovan vrednošću 0. U sledećoj iteraciji je u njemu zapamćena vrednost poslednjeg iteratora, i tako redom za sve tri pozicije levog *Shift* registra Sl. 2.5.2.
4. Sačuvati program sa *File*/*Save As*... kao *Shift_registar.vi*.
5. Zameniti *For* petlju *While* petljom tako što se miš pozicionira na ivicu *For* petlje, a potom se klikne desnim tasterom i iz padajućeg menija izabere opcija *Replace with While Loop*. Pokrenuti izvršavanje programa.
6. Zatvoriti progam bez čuvanja izmene iz tačke 5.

NAPOMENA: Ako se *Shift* registar ne inicijalizuje, tj. ne poveže mu se početna vrednost, onda će početna vrednost pri prvom pokretanju programa biti 0, a pri sledećem pokretanju programa početna vrednost će biti jednaka poslednjoj vrednosti koja je bila u *Shift* registru na kraju prvog pokretanja. Samostalno testirati navedenu tvrdnju na jednostavnom primeru.

2.6 Case struktura

Zadatak 2.6.1.
Kreirati program koji nakon pritiska na dugme OK prikazuje prozor sa rečenicom „Pritisnuli ste OK dugme.“ Prozor se gasi pritiskom na dugme „U redu“ koje se nalazi u prozoru. Program se završava pritiskom na dugme STOP.

1. Kreirati nov **.vi** program.
2. Izabrati *Case* strukturu u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u pritiskom levog tastera miša razvući *Case* strukturu kao na Sl. 2.6.1. Sa leve strane *Case* strukture se nalazi *Case selector* na koji se vezuje uslov *Case* strukture. U gornjem delu *Case* strukture (*Selector label*) se može izabrati *True* tj. *False* opcija *Case* strukture.



Sl. 2.6.1. *Case* struktura

3. Na *Front Panel*-u, u paleti ***Controls»Modern»Boolean*** izabrati *OK Button* kontrolu i postaviti je na korisnički interfejs. Povezati *OK Button* sa *Case Selector*-om *Case* strukture. Potom oko *Case* strukture i *OK Button*-a razvući *While* petlju koja se završava pritiskom na dugme STOP, Sl. 2.6.2. U *True* delu *Case* strukture postaviti *One Button Dialog* (***Functions»Programming»Dialog&User Interfaces***) kome su znakovne konstante za „message“ ulaz i „button name“ ulaz „Pritisnuli ste OK dugme.“ i „U redu“, respektivno.

Sl. 2.6.2.

4. Pokrenuti izvršavanje programa. Svaki put kada se klikne na OK Button pojaviće se prozor sa porukom „Pritisnuli ste OK dugme.“ Prozor se gasi pritiskom na dugme prozora „U redu“.
5. Sačuvati program kao *Case.vi*.

NAPOMENA: Uočiti da je „mehanička akcija“ dugmeta *OK button* bila *Latch When Released* (pogledati koja je mehanička akcija podešena tako što se desnim tasterom miša klikne na *OK button*, i izabere se opcija *Mechanical Action* u padajućem meniju), što znači da pri pritisku dugme prelazi iz *False* stanja u *True* stanje na kratko, a potom se vraća u *False* stanje. U narednom zadatku će biti razmotrene sve vrste mehaničkih akcija logičkih kontrola.

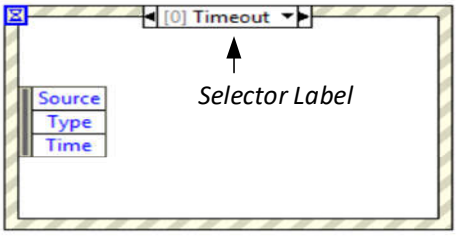
Zadatak 2.6.2.
U ***Help»Find Examples*** pronaći i otvoriti primer *Mechanical Action.vi*. Proučiti razlike u mehaničkim akcijama logičkih promenljivih.

2.7 Event struktura

Zadatak 2.7.1.
Kreirati program koji prati „događaje“ (*event*-e) koje korisnik pokreće na korisničkom interfejsu:

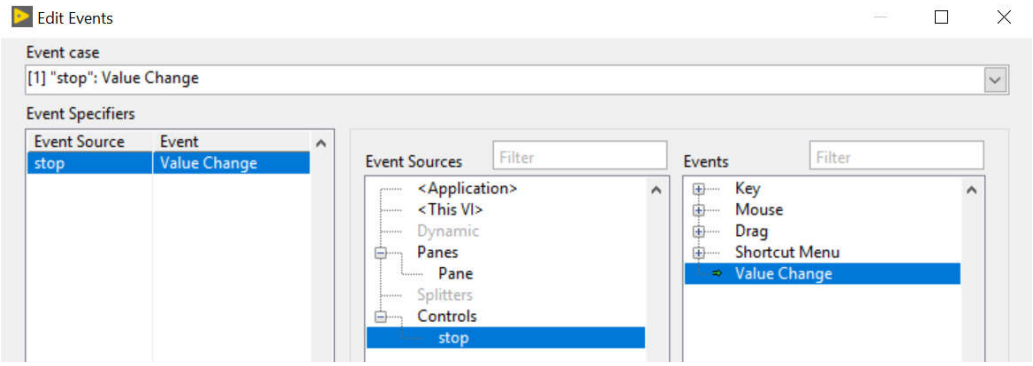
- zaustavlja se pritiskom na dugme STOP
- kada se prvi put klikne na dugme OK Button (mehanička akcija podešena na *Latch When Released*) pali se lampica na korisničkom interfejsu
- ako nikakva akcija nije preduzeta 5 sekundi prikazuje prozor sa natpisom „Niste nista uradili 5 sekundi“. Prozor se gasi pritiskom na dugme „U redu“ koje se nalazi unutar prozora.
- kada se miš pozicionira iznad dugmeta STOP, indikator *Color Box* postane zelen.
- kada korisnik pokuša da zatvori korisnički interfejs (događaj *Panel Close?*) program pita korisnika da li zaista želi „Kraj izvršavanja?“ – pritiskom na dugme „Prihvati“ korisnički interfejs se zatvara, a u slučaju pritiska na dugme „Odustani“ program nastavlja sa radom.

1. Kreirati nov **.vi** program.
2. Izabrati *Event* strukturu u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u pritiskom levog tastera miša razvući *Event* strukturu kao na Sl. 2.7.1.



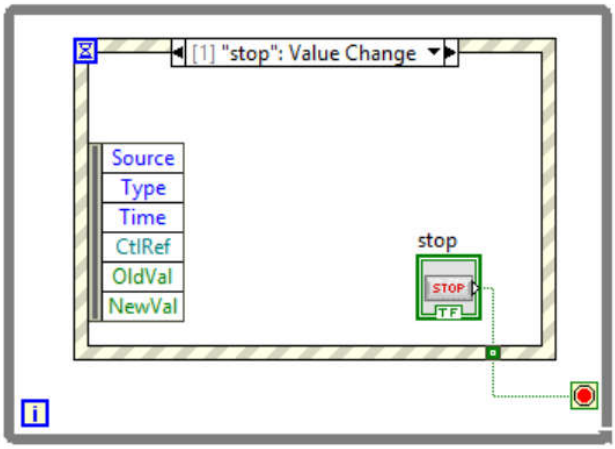
Sl. 2.7.1. *Event* struktura

3. Oko *Event* strukture razvući *While* petlju i kreirati dugme sa oznakom STOP. Dodati događaj *„stop“: Value Change* u *Event* strukturu na sledeći način: 1) kliknuti desnim tasterom miša na *Selector Label* u gornjem delu *Event* strukture i izabrati opciju *Add Event Case...* 2) u *Edit Events* prozoru selektovati „*stop*“ među izvorima događaja (*Event Sources*) i „*Value Change*“ među *event*-ima, Sl. 2.7.2. Obratiti pažnju na to šta sve od događaja može biti iskorišćeno za akciju na korisničkom interfejsu.

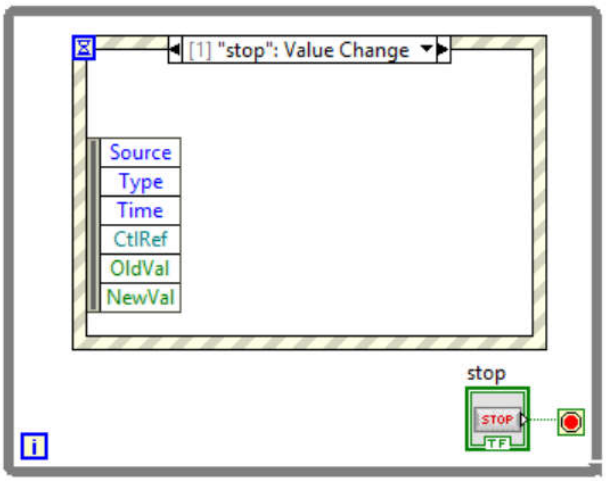


Sl. 2.7.2. Dodavanje događaja *„stop“: Value Change* u *Event* strukturu

4. Pritisnuti dugme OK na *Event case* prozoru i nakon toga će se u *Selector Label* polju *Event* strukture pojaviti natpis *„stop“: Value Change* kao na Sl. 2.7.3. Povezati dugme STOP sa uslovnim terminalom *While* petlje na način prikazan na Sl. 2.7.3.
5. Pokrenuti izvršavanje programa sa Sl. 2.7.3 i zaustaviti ga pritiskom na dugme STOP.

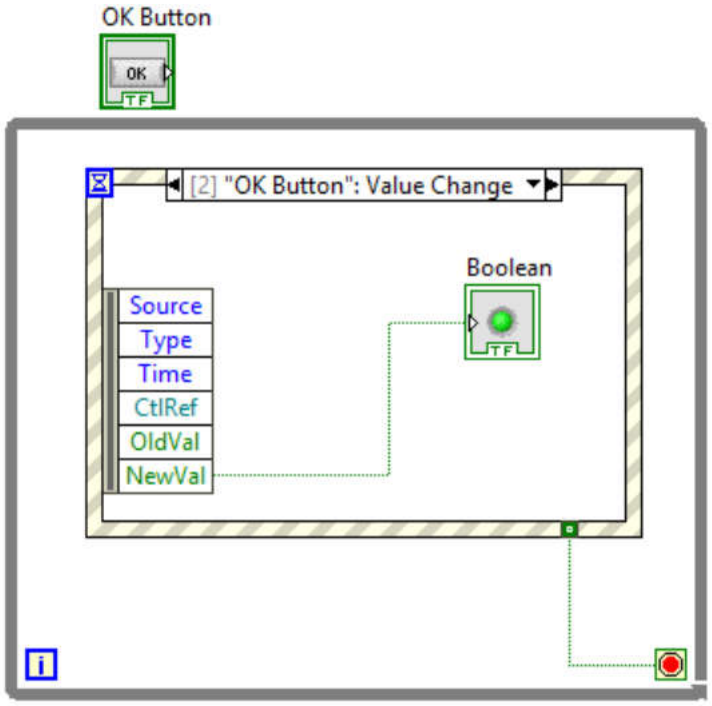


Sl. 2.7.3. Ispravan način zaustavljanja *While* petlje koja ima *Event* strukturu



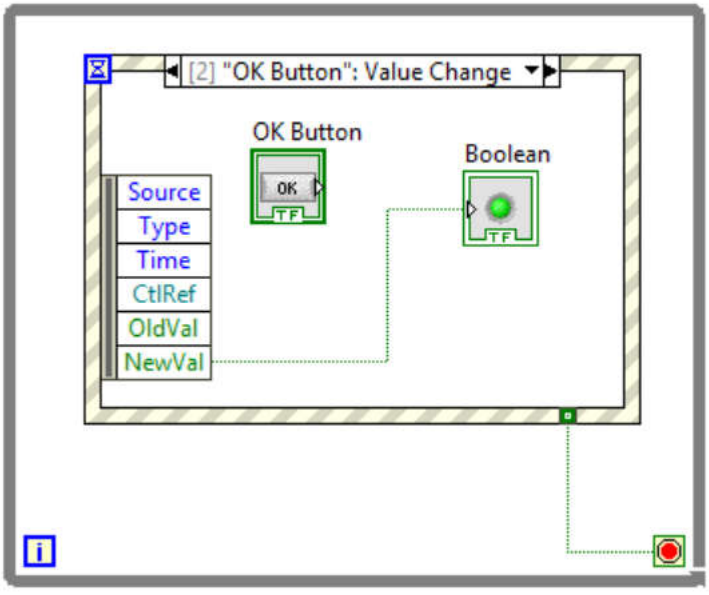
Sl. 2.7.4. Neispravan način zaustavljanja *While* petlje koja ima *Event* strukturu

6. Isprobati šta bi se desilo sa izvršavanjem programa kada bi dugme STOP bilo postavljeno izvan *Event* strukture, kao što je to prikazano na Sl. 2.7.4. Uključiti dugme ***Execution Highlighting*** i pokrenuti izvršavanje programa sa Sl. 2.7.4. Zašto se tek nakon drugog pritiska na dugme STOP program zaustavlja?
7. U program sa Sl. 2.7.3. dodati dugme *OK Button* sa mehaničkom akcijom *Latch When Released* van *While* petlje. Dodati događaj “OK Button”*: Value Change* kao što je to urađeno u tački 3. U okviru ovog događaja omogućiti da se nova vrednost dugmeta *OK Button* prikaže i na logičkom indikatoru (lampici) kao na Sl. 2.7.5.
8. Pokrenuti izvršavanje programa. Uočiti da se nakon pritiska *OK Button* dugmeta lampica upali, ali da se *OK Button* dugme ne vrati u *False* položaj iako je *latch* tipa mehaničke akcije.
9. Zaustaviti izvršavanje programa pritiskom na dugme STOP.



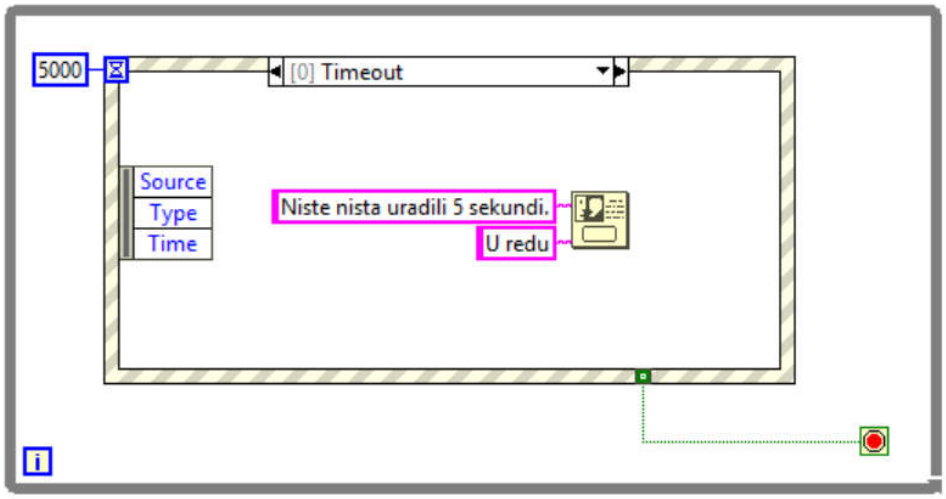
Sl. 2.7.5. Neispravan način upotrebe *Latch* dugmeta u *Event* strukturi

10. Premestiti dugme *OK Button* u *Event case* odgovarajućeg događaja, kao na Sl. 2.7.6.
11. Pokrenuti izvršavanje programa. Uočiti da se nakon pritiska *OK Button* vraća u *False* stanje kao što je i predviđeno za *latch* dugme. Zaustaviti izvršavanje programa pritiskom na dugme STOP.



Sl. 2.7.6. Ispravan način upotrebe *Latch* dugmeta u *Event* strukturi

12. Omogućiti da, ukoliko korisnik ne uradi ništa 5 sekundi na korisničkom interfejsu, se o tome pojavi obaveštenje. Na Sl. 2.7.7. je prikazan *Timeout case* koji omogućava pojavu *One Button Dialog* prozora sa konstatacijom "Niste nista uradili 5 sekundi." Primetiti da je na *Timeout* ulazu (gornji levi ugao *Event* strukture) povezana konstanta od 5000 ms.



Sl. 2.7.7. Primena *Timeout case*-a u *Event* strukturi

13. Testirati izvršavanje programa sa unetom modifikacijom za *Timeout*. Zaustaviti izvršavanje programa pritiskom na dugme STOP.
14. Samostalno modifikovati program tako da:
    a. kada se miš pozicionira iznad dugmeta STOP, indikator *Color Box* postane zelen (*Mouse Enter* događaj).
    b. kada korisnik pokuša da zatvori korisnički interfejs (*Filter* događaj *Panel Close?*) program pita korisnika da li zaista želi „Kraj izvršavanja?“ – pritiskom na dugme „Prihvati“ korisnički interfejs se zatvara, a u slučaju pritiska na dugme „Odustani“ program nastavlja sa radom.
15. Sačuvati program kao *Event_Filter.vi*.

NAPOMENA: Razlika između *Notify* i *Filter* događaja (npr. *Panel Close* i *Panel Close?*) je u tome što za *Filter* događaje postoji mogućnost da korisnik uradi *Discard*, tj. da ne prihvati izvršavanje događaja.

Lekcija 3 – Strukture podataka

Cilj

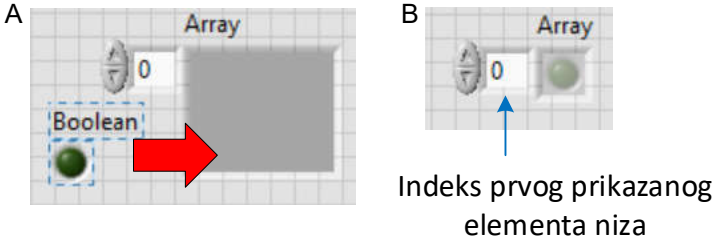
Cilj lekcije je da studenti savladaju:

- manupulaciju nizovima
- pojam dinamičkog tipa podataka
- kreiranje lokalnih promenljivih
- programsko kontrolisanje osobina kontrola/indikatora
- mogućnosti selekcije sekvenci *Case* strukture različitim tipovima podataka
- manupulaciju klasterima
- striktno definisanje izgleda kontrola i indikatora.

3.1 Nizovi

Zadatak 3.1.1.
Kreirati program koji konvertuje vrednosti logičkog niza u numerički niz, tj. koji TRUE/FALSE vrednosti konvertuje u 0/1 vrednosti.

1. Kreirati nov **.vi** program.
2. U paleti ***Controls»Modern»Boolean*** izabrati logičku kontrolu (*Round LED*) i postaviti je na *Front Panel*.
3. U paleti ***Controls»Modern»Array, Matrix & Cluster*** izabrati *Array* i postaviti na *Front Panel*.
4. Pomoću *Position/Size/Select* alatke  selektovati *Round LED* kontrolu i držeći pritisnut levi taster miša „prevući“ je u *Array* kao što je prikazano na Sl. 3.1.1A. Rezultat „prevlačenja“ je prikazan na Sl.3.1.1B.

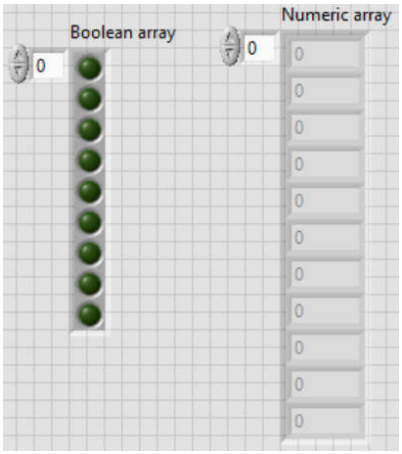


Sl. 3.1.1. A) Postupak kreiranja niza logičkih kontrola, B) Kreiran niz logičkih kontrola

NAPOMENA: Broj dimenzija niza se može uvećati tako što se klikne desnim tasterom miša na indeks (Sl. 3.1.1) i izabere opcija *Add Dimension*. Broj dimenzija se redukuje izborom opcije *Remove Dimension* (minimalna broj dimenzija je 1).

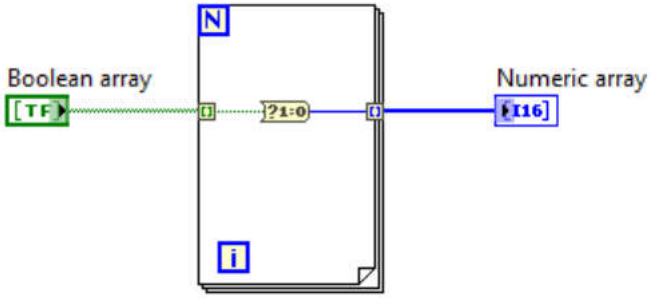
5. Postupak kreiranja niza ponoviti za niz numeričkih indikatora (*Numeric Indicator* u paleti ***Controls»Modern»Numeric***).
6. Razvući niz logičkih kontrola i niz numeričkih indikatora tako da se vidi više članova niza, Sl. 3.1.2. Pomoću alatke *Labeling Tool* u *Tools* paleti uneti odgovarajući tekst za oznake niza kontrola i indikatora.

NAPOMENA: Nije moguće napraviti niz nizova.



Sl. 3.1.2. Niz logičkih kontrola i niz numeričkih indikatora

7. U *Block Diagram*-u kreirati programski kôd koji primenom *For* petlje selektuje elemente logičkog niza i konvertuje ih pomoću funkcije ***Functions»Programming»Boolean» Boolean To (0,1)*** u broj 0 ili 1. Programski kôd je prikazan na Sl. 3.1.3. Povezivanje *Boolean array* kontrole sa *For* petljom ostvariti pomoću alatke *Wire Tool*. Primetiti da je terminal na ivici *For* petlje dobio prikaz sa indikacijom „uglastih zagrada“ na ivici *For* petlje. Ovaj prikaz ukazuje na aktivaciju tzv. auto-indeksiranja, tj. *For* petlja izdvaja element po element niza koji dalje može da se koristi za obradu unutar kôda *For* petlje. Takođe, i na „izlazu“ *For* petlje može se primetiti isti prikaz auto-indeksiranja pri povezivanju sa *Numeric array* indikatorom, tj. obrađeni elementi (u ovom slučaju elementi nastali nakon konverzije TRUE/FALSE u 0/1) se slažu u izlazni niz. Uočiti da brojač *For* petlje nije definisan eksplicitno već da je definisan implicitno zahvaljujući auto-indeksiranju.

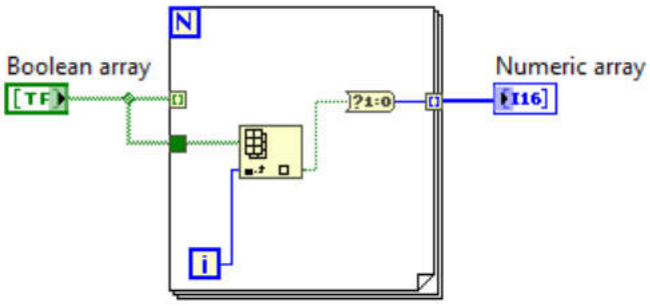


Sl. 3.1.3. Auto-indeksiranje u *For* petlji

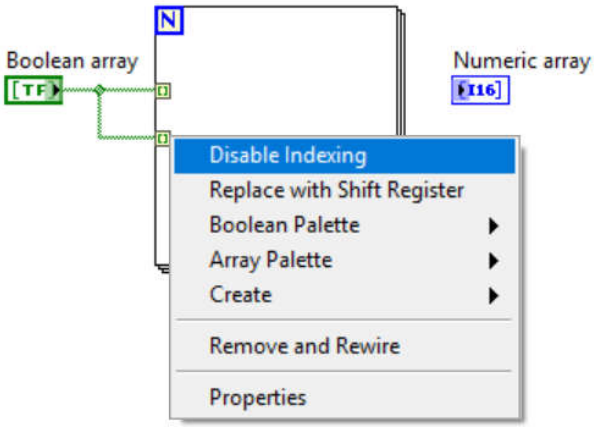
8. Sačuvati program kao *Konverzija_logickog_niza_u_numericki.vi*.
9. Na *Boolean array* kontroli upaliti nekoliko lampica, a potom pokrenuti izvršavanje programa. Uočiti da će se odgovarajuće vrednosti „1“ pojaviti i na *Numeric array* indikatoru. Indeks prvog prikazanog elementa niza (Sl. 3.1.1B) se

može menjati, tj. elementi se mogu „listati“ - uočiti kako se pri njegovoj promeni pomeraju i elementi niza.

10. Programski kôd sa Sl. 3.1.3 je ekvivalentan kôdu na Sl. 3.1.4 gde je umesto izlaza auto-indeksiranja iskorišćen izlaz funkcije ***Functions»Programming»Array »Index Array*** za dalju konverziju. Auto-indeksiranje se isključuje tako što se klikne desnim tasterom miša na „uglaste zagrade“ i izabere opcija *Disable Indexing,* Sl. 3.1.5.

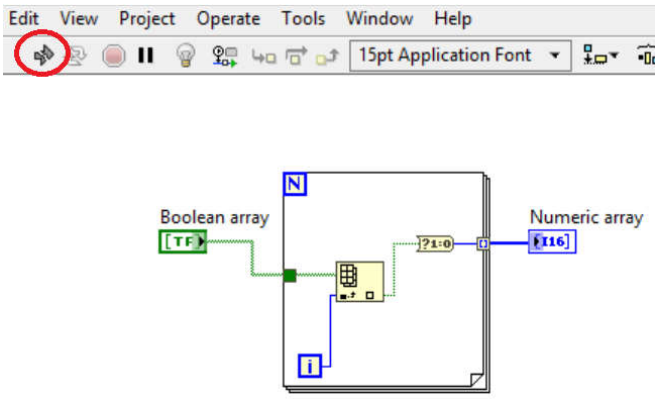


Sl. 3.1.4. Primena *Index Array* funkcije



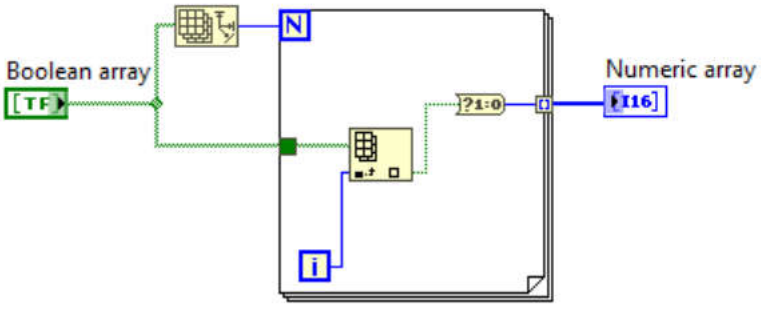
Sl. 3.1.5. Isključivanje auto-indeksiranja

11. Kreirati programski kôd sa Sl. 3.1.4. Uočiti da su od *Boolean array*-a na *For* petlju priključene obe žica: i sa i bez auto-indeksiranja. Pokrenuti izvršavanje programskog kôd-a.
12. Zaustaviti izvršavanje programa. Izbrisati žicu za auto-indeksiranje kao na Sl. 3.1.6. Uočiti da će se pojaviti slomljena strelica u *Status Toolbar*-u. Zašto? (Pomoć: na šta je vezan brojač?)



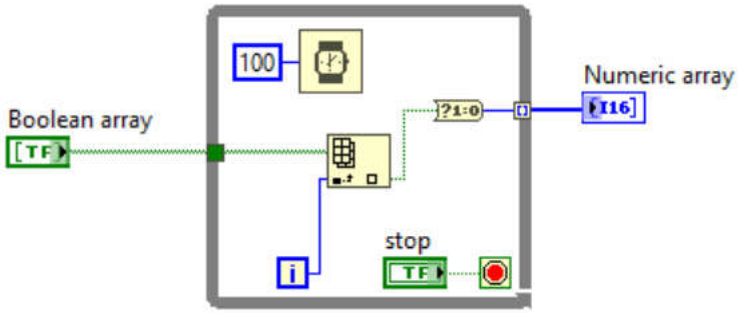
Sl. 3.1.6.

13. Kreirati programski kôd kao na Sl. 3.1.7 koji koristi funkciju ***Functions»Programming»Array»Array Size*** za određivanje dimenzije niza. Pokrenuti program.



Sl. 3.1.7. Primena *Array Size* funkcije

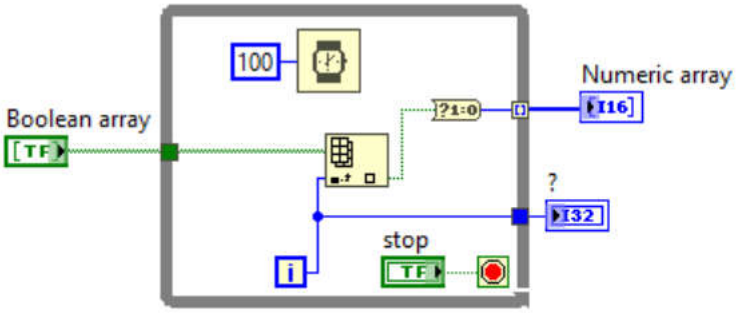
14. Zaustaviti program. Zameniti *For* petlju *While* petljom, Sl. 3.1.8. Nasumično uključiti lampice i pokrenuti program. Uočiti da će se vrednosti na *Numeric array* pojaviti tek nakon pritiska na dugme STOP.



Sl. 3.1.8. Auto-indeksiranje u *While* petlji

15. Zaustaviti program. Modifikovati ga kao na Sl. 3.1.9. Prenos podataka kroz petlju, bilo u nju ili van nje, naziva se „tunelovanje“. Pokrenuti program i zaustaviti ga. Šta će da pokazuje indikator označen sa „?“?

NAPOMENA: Mogućnosti auto-indeksiranja i „tunelovanja“ (tj. prenosa podataka u petlju ili iz petlje) važe podjednako i za *For* petlju i za *While* petlju.

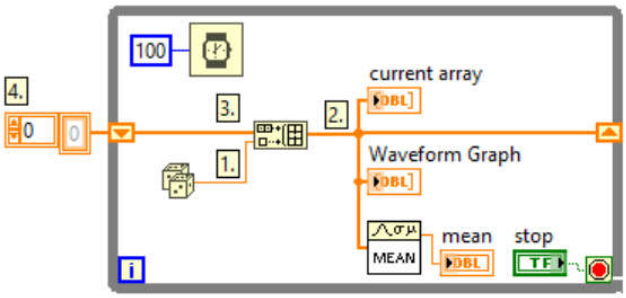


Sl. 3.1.9. „Tunelovanje“ tj. prenos podataka kroz petlju

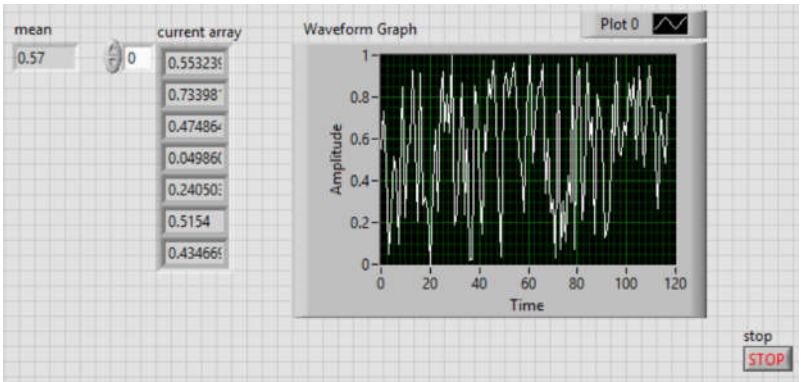
16. Sačuvati program kao *Konverzija_logickog_niza_u_numericki_While.vi*.
17. Proučiti pomoću *Context Help*-a šta rade i druge funkcije za manipulaciju nad nizovima a koje se nalaze u paleti ***Functions» Programming»Array***.

Zadatak 3.1.2.
Kreirati program koji koristeći *Shift* registar u *While* petlji dinamički gradi niz slučajnih podataka. U toku izvršavanja programa prikazivati srednju vrednost niza na numeričkom indikatoru, a niz prikazivati na indikatoru ***Controls»Modern»Waveform Graph***. Obezbediti da procesor ima slobodno vreme (trajanje jedne iteracije *While* petlje je podešeno na 100 ms).

1. Kreirati nov **.vi** program.
2. Kreirati *While* petlju koja se zaustavlja pritiskom na dugme STOP. Funkcijom *Wait (ms)* obezbediti da procesor ima slobodno vreme i da svaka iteracija traje 100 ms (Lekcija 2.3).
3. Iz palete ***Functions»Mathematics*** izabrati funkciju *Random Number 0-1* za generisanje slučajnih brojeva i uneti je u *While* petlju.
4. Funkcija za gradjenje niza je ***Build Array*** u paleti ***Functions» Programming»Array***. Razvlačenjem proširiti ovu funkciju tako da ima dva ulaza. Dodati *Shift* registar na *While* petlju na način objašnjen u Lekciji 2.5.2. *Block diagram* celog programa je prikazan na Sl. 3.1.10. Brojevima 1, 2, 3 i 4 je označen redosled povezivanja žica za *Build Array* funkciju i *Shift* registar. Uočiti da je najpre potrebno da se poveže izvor slučajnih podatka na donji priključak *Build Array* funkcije (1.), potom izlaz *Build Array* funkcije na desnu stranu *Shift* registra (tj. „napuniti“ *Shift* registar tipom podatka) (2.), a potom levu stranu *Shift* registra vratiti na gornji ulaz *Build* funkcije (3.). Inicijalizacija vrednosti *Shift* registra se postiže tako što se miš pozicionira iznad levog *Shift* registra, uradi klik desnim tasterom miša i izabere opcija *Create Constant* (4.).
5. *Front Panel* programa je prikazan na Sl. 3.1.11. Za određivanje srednje vrednosti niza izabrati funkciju ***Mean*** iz palete ***Functions»Mathematics»Probability &Statistics***.



Sl. 3.1.10. Dinamičko građenje niza. Redosled povezivanja priključaka *Build Array* funkcije na *Shift* registar je dat brojevima 1., 2., 3., 4.



Sl. 3.1.11. *Front Panel* programa za dinamičko građenje niza

6. Sačuvati program kao *Dinamicko_gradjenje_niza*.
7. Pokrenuti izvršavanje programa. Kojoj brojnoj vrednosti teži srednja vrednost niza koji se gradi?
8. Zaustaviti progam i zatvoriti ga.

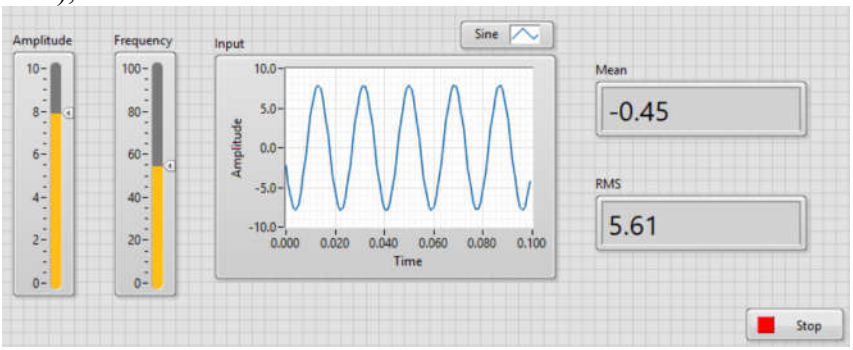
Zadatak 3.1.3. - za samostalni rad
Kreirati program koji:
1. sabira niz [1,2,3,4] sa brojem 4.
2. sabira niz [1,2,3,4] sa nizom [4,3,2,1]
3. sabira niz [1,2,3,4] sa nizom [4,3,2].

Diskutovati rezultate sabiranja.

NAPOMENA: *LabVIEW* funkcije imaju osobinu **polimorfizma**, tj. omogućavaju da se na njihove ulaze dovode različiti tipovi promenljivih. Na primer, funkcija sabiranja niza sa brojem i niza sa nizom je identična po svom izgledu.

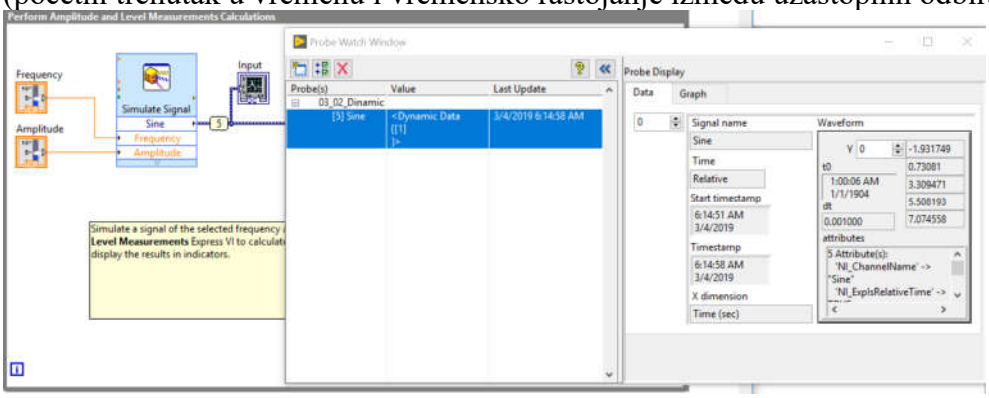
3.2 Dinamički tip podataka

1. Otvoriti primer *Express VI - Amplitude and Level Measurements.vi* koji se nalazi u ***Help»Find Examples*** (radi lakšeg nalaženja ovog primera u *Search* sekciji uneti „simulate“), Sl. 3.2.1.



Sl. 3.2.1. *Front Panel* primera *Express VI - Amplitude and Level Measurements.vi*

2. Proučiti *Block Diagram* primer-a. Pokrenuti program i menjati amplitudu i frekvenciju signala prateći promene na ekranu. Pomoću alatke *Probe* posmatrati sadržaj izlaza iz funkcije *Simulate Signal*, Sl. 3.2.2. Ova *Express* funkcija daje tzv. dinamički tip podataka koji sadrži informacije ne samo o vrednostima odbiraka signala već i o njihovoj vezi sa vremenskim domenom (početni trenutak u vremenu i vremensko rastojanje između uzastopnih odbiraka).

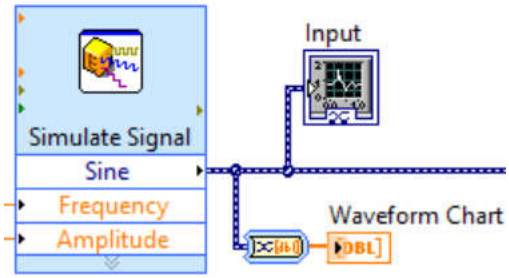


Sl. 3.2.2. Struktura dinamičkog tipa podataka

3. Zaustaviti program. Sačuvati ga pomoću *Save As* opcije kao *Dinamicki_tip_podataka.vi*.

Zadatak 3.2.1.
Polazeći od programa *Dinamicki_tip_podataka.vi*. kreirati program koji konvertuje dinamički tip podataka u numerički niz (koristeći funkciju ***Functions»Express»Convert from Dynamic Data***) i prikazuje na *Waveform Chart*-u.

1. Otvoriti program *Dinamicki_tip_podataka.vi* sačuvan u prethodnoj tački.
2. Modifikovati program kao što je prikazano na Sl. 3.2.3.



Sl. 3.2.3. Konverzija dinamičkog tipa podataka u numerički niz

3. Pokrenuti program. Po čemu se razlikuje prikaz na grafičkom indikatoru *Input* u odnosu na prikaz na kreiranom *Waveform Chart*-u?
4. Zaustaviti program i **sačuvati izmene**.

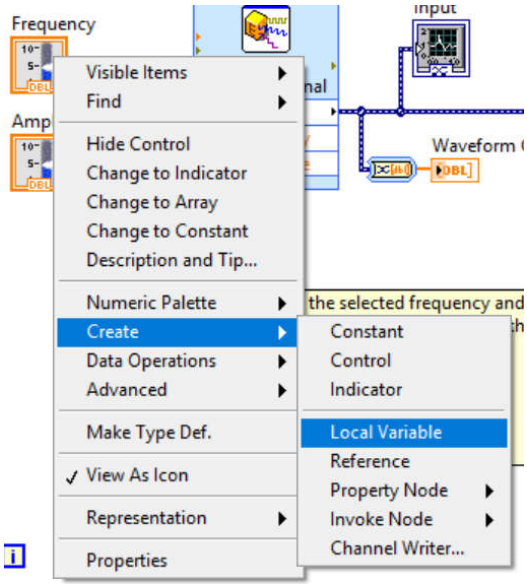
3.3 Lokalne promenljive – primena u inicijalizaciji

Zadatak 3.3.1.
Polazeći od sačuvanog programa *Dinamicki_tip_podataka.vi*. kreirati program koji na početku izvršavanja radi INICIJALIZACIJU koja podrazumeva zadavanje početnih vrednosti kontrola *Amplitude* i *Frequency*, a potom nastavlja sa regularnim radom.

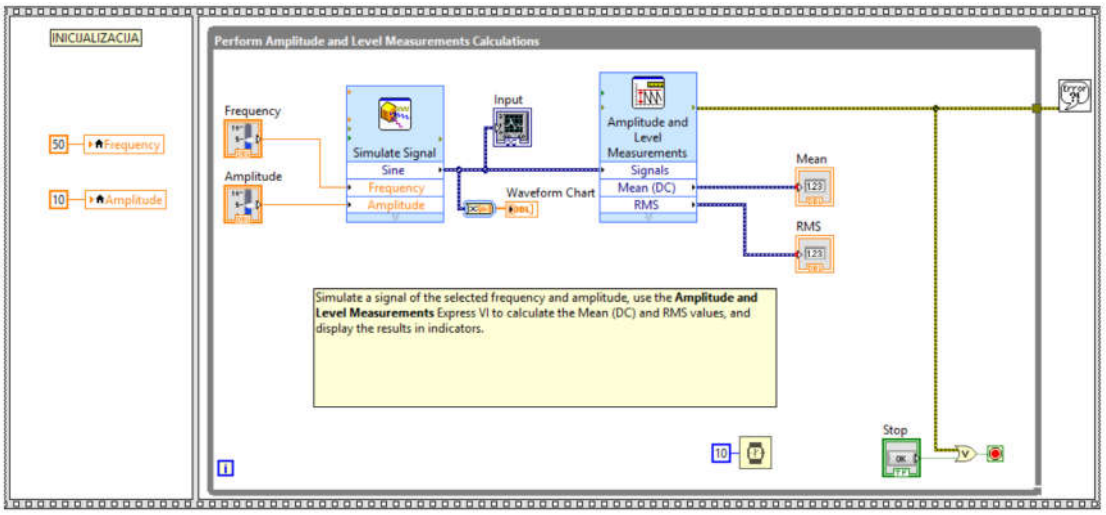
1. Otvoriti program *Dinamicki_tip_podataka.vi* sačuvan u prethodnoj tački.
2. Ceo program postaviti u *Sequence* strukturu (Lekcija 2.2). Potom dodati jedan frejm *Sequence* strukture pre tog i u njemu napraviti lokalne promenljive kontrola *Amplitude* i *Frequency*. Lokalne promenljive se kreiraju tako što se miš pozicionira iznad kontrole, potom se klikne desnim tasterom miša i izabere opcija ***Create»Local Variable***, Sl. 3.3.1. Kreirane lokalne promenljive pomoću *Position/Size/Select* alatke  selektovati i postaviti u nultu sekvencu *Sequence* structure kao na Sl.3.3.2.
3. Pokrenuti program. Uočiti da je početna vrednost kontrola sada kao što je zadato lokalnim promenljivama.
4. Zaustaviti program i **sačuvati izmene**.

NAPOMENA: Lokalne promenljive se na identičan način prave i za kontrole i za indikatore. U njih se može upisivati, ali i čitati. Lokalna promenljiva u koju se upisuje

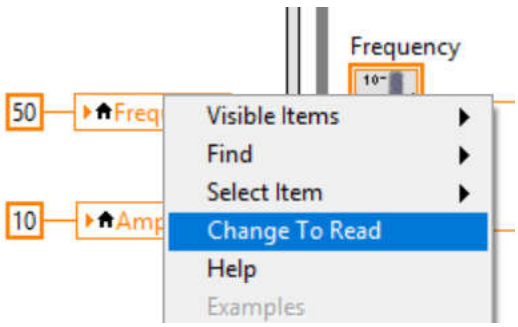
se može pretvoriti u promenljivu iz koje se čita tako što se miš pozicionira iznad nje, klikne se desnim tasterom miša i izabere opcija ***Change to Read***, Sl. 3.3.3. Sličan redosled operacija se primenjuje i kada treba promeniti namenu na ***Change to Write.***



Sl. 3.3.1. Kreiranje lokalne promenljive



Sl. 3.3.2. Primena lokalnih promenljivih za inicijalizaciju

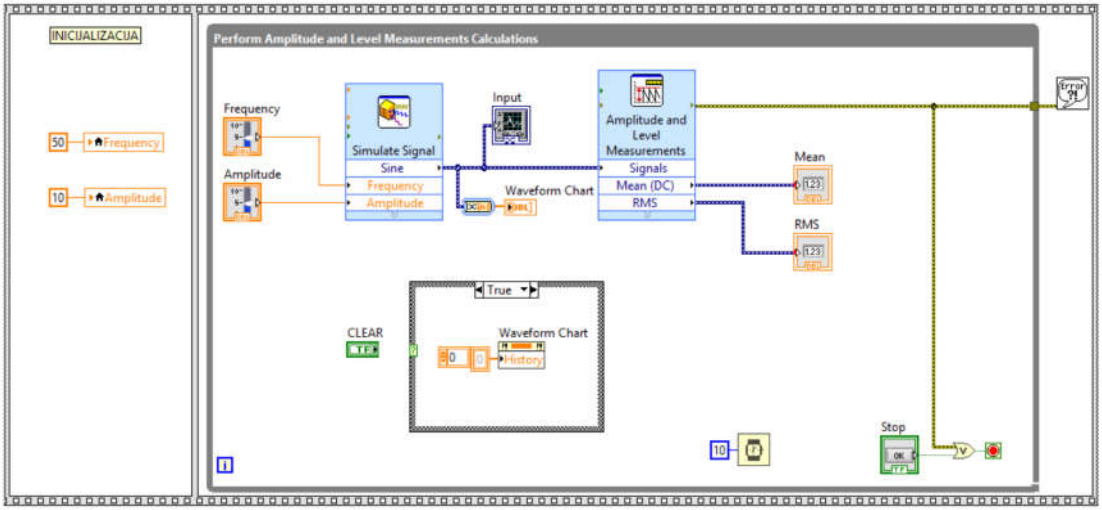


Sl. 3.3.3. Promena lokalne promenljive iz môda upisa u môd čitanja

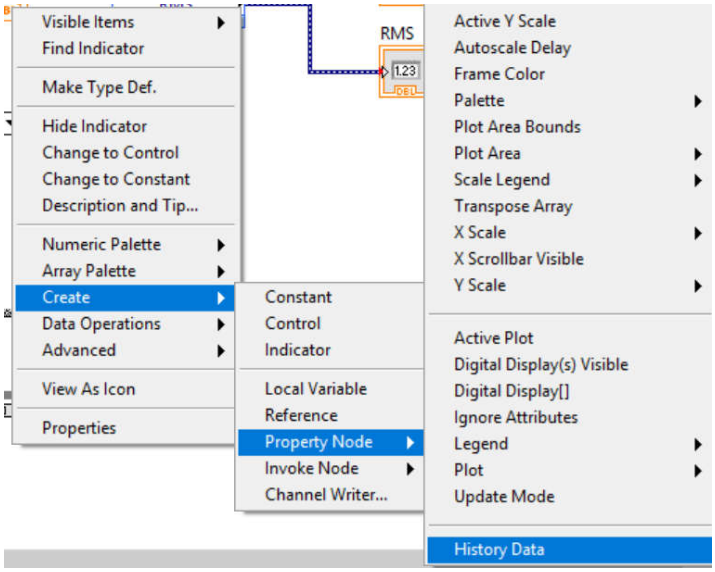
3.4 Programsko kontrolisanje osobina kontrola/indikatora

Zadatak 3.4.1.
Polazeći od sačuvanog programa *Dinamicki_tip_podataka.vi*. kreirati program koji pri pritisku na dugme CLEAR briše sadržaj *Waveform Chart*-a.

1. Otvoriti program *Dinamicki_tip_podataka.vi* sačuvan u prethodnoj tački.
2. Unutar *While* petlje kreirati *Case* strukturu kao i dugme CLEAR koje treba povezati na *Case selector* ulaz *Case* strukture, Sl. 3.4.1. Unutar TRUE sekvence *Case* strukture kreirati ***Property Node»History Data*** indikatora *Waveform Chart* tako što se miš pozicionira iznad indikatora, potom se klikne desnim tasterom miša i izabere opcija ***Create» Property Node»History Data***, Sl. 3.4.2.



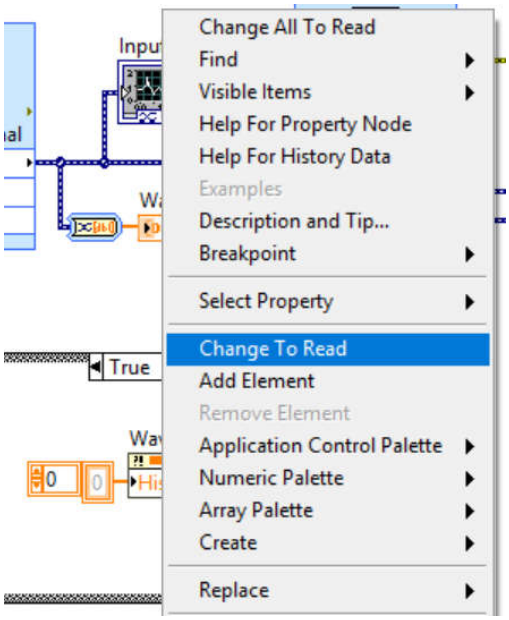
Sl. 3.4.1. Programsko podešavanje osobine kontrole/indikatora



Sl. 3.4.2. Kreiranje *Propery Node*-a kontrole/indikatora

3. Pokrenuti program. Uočiti da se klikom na dugme CLEAR briše sadržaj *Waveform Chart*-a.
4. Zaustaviti program i **sačuvati izmene**.

NAPOMENA: *Property Node* se na identičan način prave i za kontrole i za indikatore. U njih se može upisivati, ali i čitati. *Property Node* u koji se upisuje se može pretvoriti u promenljivu iz koje se čita tako što se miš pozicionira iznad nje, klikne se desnim tasterom miša i izabere opcija ***Change to Read***, Sl. 3.4.3. Sličan redosled operacija se primenjuje i kada treba promeniti namenu na ***Change to Write.***



Sl. 3.4.3. Promena Property Node-a iz môda upisa u môd čitanja

Zadatak 3.4.2. - za samostalan rad

Modifikovati Zadatak 3.4.1. tako se umesto pritiskom na dugme CLEAR, sadržaj *Waveform Chart*-a menja:

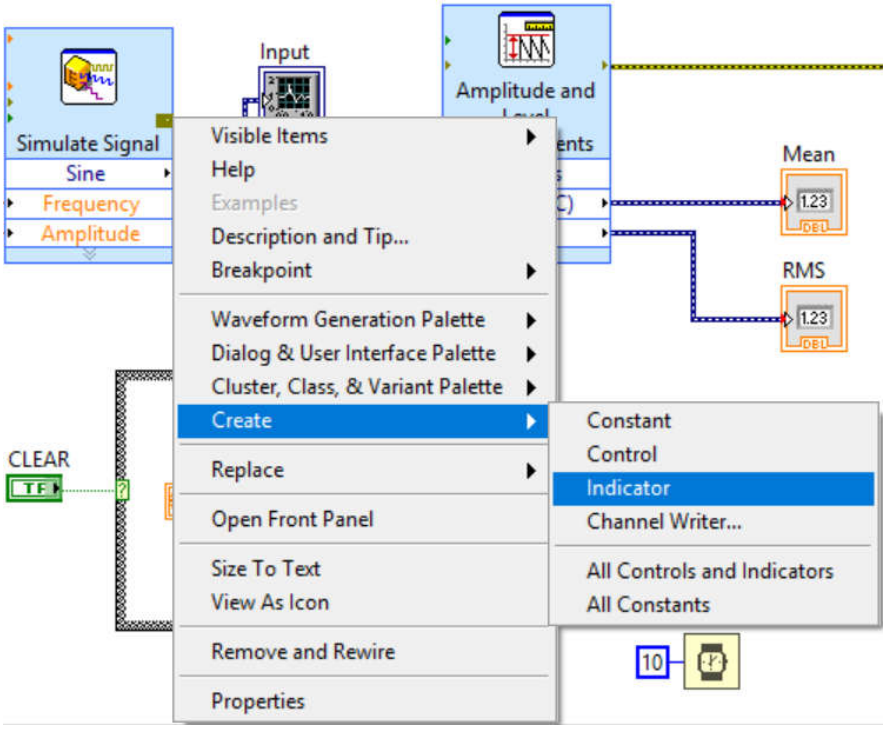
1) selekcijom odgovarajuće vrednosti *String* kontrole
2) selekcijom odgovarajuće vrednosti *Enum* kontrole.

3.5 Klasteri

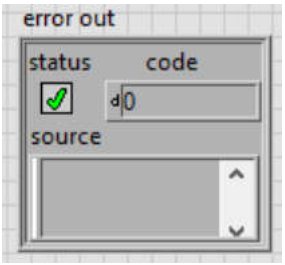
Klasteri predstavljaju strukturu podataka koja se sastoji iz više različitih tipova podataka.

Zadatak 3.5.1.
Modifikovati program iz prethodne tačke tako da se završava ili pritiskom na dugme STOP ili u slučaju da *status* greške postane TRUE.

1. Otvoriti program *Dinamicki_tip_podataka.vi* sačuvan u prethodnoj tački.
2. Pozicionirati alatku *Wire Tool* iznad „*error out*“ izlaza *Express* funkcije *Simulate Signal*. Kliknuti desnim tasterom miša i izabrati opciju ***Create»Indicator***, Sl. 3.5.1.
3. *Error* klaster se sastoji iz tri podatka: *status* (logička promenljiva koja pokazuje da li je došlo do greške), *code* (numerička promenljiva I32 koja je kôd greške) i *source* (znakovna tj. *string* promenljiva koja opisuje izvor greške).



Sl. 3.5.1. Kreiranje „*error*“ klastera

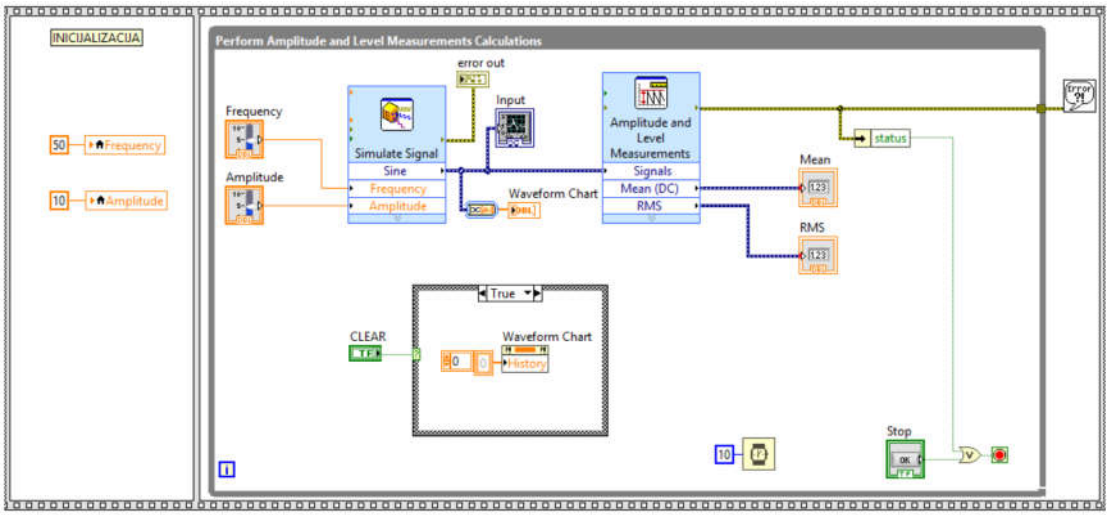


Sl. 3.5.2. „*Error*“ klaster, izgled na *Front Panel*-u

4. Klasteri se mogu „raspakovati“ pomoću funkcije ***Functions»Programming» Cluster, Class & Variant»Unbundle by Name.*** Funkcija *Unbundle by Name* se

može „razvući“ tako da ima onoliko izlaza koliko klaster ima tipova podataka u sebi.

5. Modifikovati program u skladu sa Sl. 3.5.3. tako da *status* promenljiva *error* klastera bude preko ILI kola dovedena na uslovni terminal *While* petlje.
6. **Sačuvati** izmene.

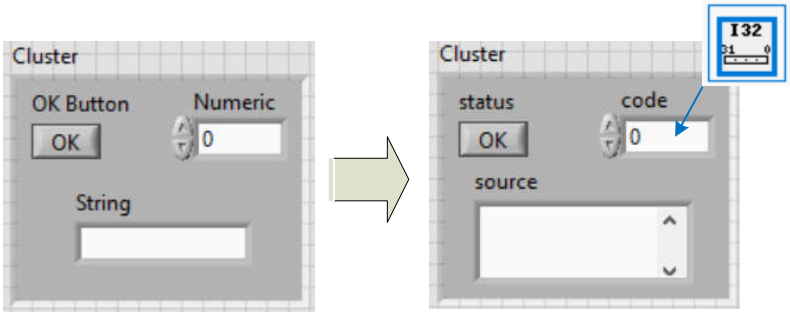


Sl. 3.5.3. „Raspakivanje“ *error* klastera i gašenje programa kada *status* greške postane TRUE

Zadatak 3.5.2.

Kreirati *error* klaster kontrolu i *error* klaster indikator. Omogućiti da se samo vrednost *status* promenljive *error* klaster indikatora menja pomoću logičke kontrole sa *Front Panel*-a, a da se *code* i *source* promenljive *error* klaster kontrole „preslikavaju“ u *code* i *source* promenljive *error* klaster indikatora.

1. Kreirati nov **.vi** program.
2. Iz palete ***Controls»Modern»Array, Matrix & Cluster*** selektovati kontrolu *Cluster* i postaviti je na *Front Panel*. Potom u *Cluster* kontrolu na *Front Panel*-u postaviti sledeće tri kontrole PREMA DATOM REDOSLEDU: 1) *String Control* (***Controls»Modern»String & Path***), 2) *Numeric Control* (***Controls»Modern» Numeric***) i 3) *OK Button* (***Controls»Modern»Boolean***). Za *Numeric Control* podesiti da oznaka bude *code*, a da tip podataka bude *integer* 32-bita (klik desnim tasterom miša na kontrolu, opcija *Representation, I32*) jer se tako standardno definiše u *error* klasteru. Takođe, podesiti da oznake za *String Control*-u i *OK Button* budu *source* i *status*, respektivno, Sl. 3.5.4.

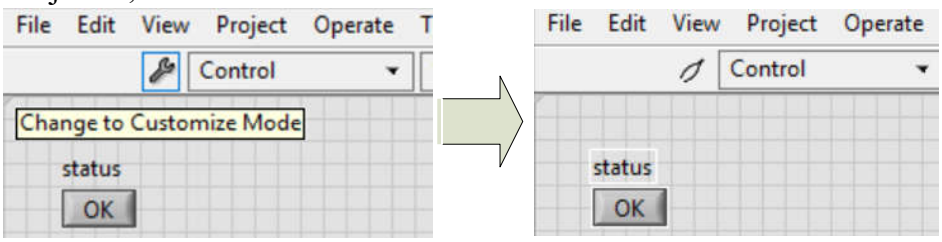


Sl. 3.5.4. Manuelno napravljen *error* klaster sastavljen iz tri promenljive različitog tipa

NAPOMENA: Vertikalni *scrollbar* kontrole *string* je onemogućen (*disabled*) sve dok se *string* kontrola ne proširi bar na dva reda. Kada se *string* kontrola proširi po vertikali tako da zahvata bar dva reda, onda postaje omogućena opcija *Vertical Scrollbar* (klik desnog tastera miša, *Visible Items*).

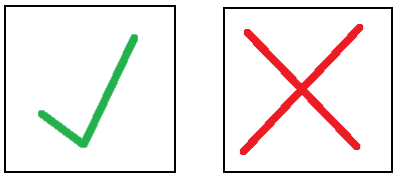
NAPOMENA: *Status* promenljiva u *error* klasteru trenutno nema standardni izgled (,). Promena izgleda kontrole/indikatora se može uraditi tako što se na kontrolu/indikator klikne desnim tasterom miša i izabere opcija *Advanced/Customize*.

3. Kliknuti desnim tasterom miša na *OK Button error* klastera i izabrati opciju *Advanced/Customize*. Otvoriće se novi prozor u kome treba izabrati môd za editovanje , Sl. 3.5.5.



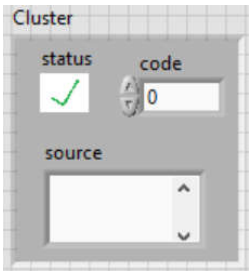
Sl. 3.5.5. Selekcija môda za editovanje kontrole/indikatora

4. U programu za kreiranje slika (npr. *Paint*-u) napraviti dve slike istih dimenzija kao na Sl.3.5.6.



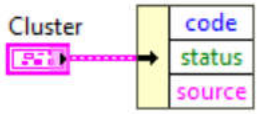
Sl. 3.5.6. Slike za TRUE i FALSE stanja *status* promenljive *error* klastera

5. Kliknuti desnim tasterom miša na kontrolu *status* u prozoru *Customize mode*. Izabrati opciju *Picture Item* i selektovati najpre izgled kontrole za TRUE stanje . Kliknuti desnim tasterom miša na kontrolu i izabrati opciju *Import from file...* Selektovati datoteku sa novom slikom za TRUE stanje . Sličan postupak za pomenu izgleda u ponoviti za FALSE stanje. S obzirom na to da standardna *status* promenljiva u *error* klasteru nema natpis OK na samom dugmetu, izbrisati natpis OK korišćenjem *Labeling Tool*-a .
6. Sačuvati editovanu kontrolu izborom opcije ***File»Save*** kao *status_kontrola.ctl* datoteku. Zatvoriti prozor za editovanje kontrola, a za pitanje „*Replace the original control „status“ with „status_kontrola.ctl?*“ kliknuti na opciju *Yes*. Novi izgled *error* klastera je prikazan na Sl. 3.5.7.



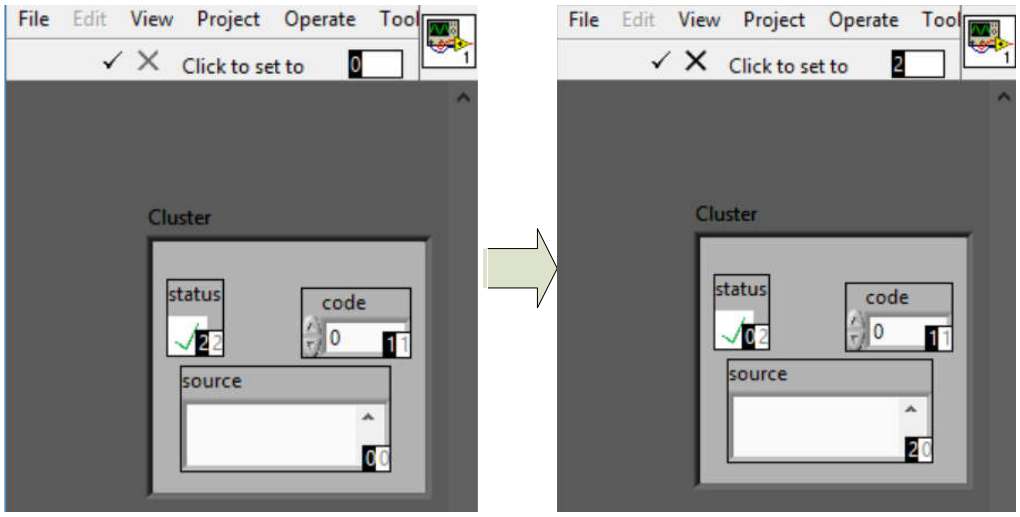
Sl. 3.5.7. Izgled editovanog *error* klastera

7. U *Block Diagram*-u izabrati funkciju ***Functions»Programming» Cluster, Class & Variant»Unbundle by Name*** i povezati je na *error* klaster. Proširiti funkciju *Unbundle by Name* tako da ima tri izlaza (pomoću *Position/Size/Select* alatke ), Sl. 3.5.8. Uočiti da je redosled promenljivih nestandardni, tj. da je prva promenljiva *code*, druga *status*, treća *source* (standardni redosled je: *status, code, source*). Ovakav redosled je nastao zbog redosleda postavljanja kontrola u klaster u tački 2.



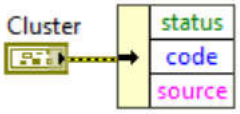
Sl. 3.5.8. „Raspakivanje“ editovanog error klastera – nestandardni redosled promenljivih u njemu

8. Promeniti redosled promenljivih u standardni. Najpre izabrati *Position/Size/Select* alatka  , potom kliknuti desnim tasterom miša na *error* klaster kontrolu i izabrati opciju *Reorder controls in cluster*. Otvoriće se prozor za editovanje redosleda u kome se redosled promenljivih u klasteru zadaje redosledom kliktanja mišem po kontrolama, Sl. 3.5.9. Kliknuti sledećim redom na: 1) *status* kontrolu, 2) *code* kontrolu i 3) *source* kontrolu. Prihvatiti novi redosled promenljivih klikom na opciju *Confirm*  u *Status Toolbar*-u.



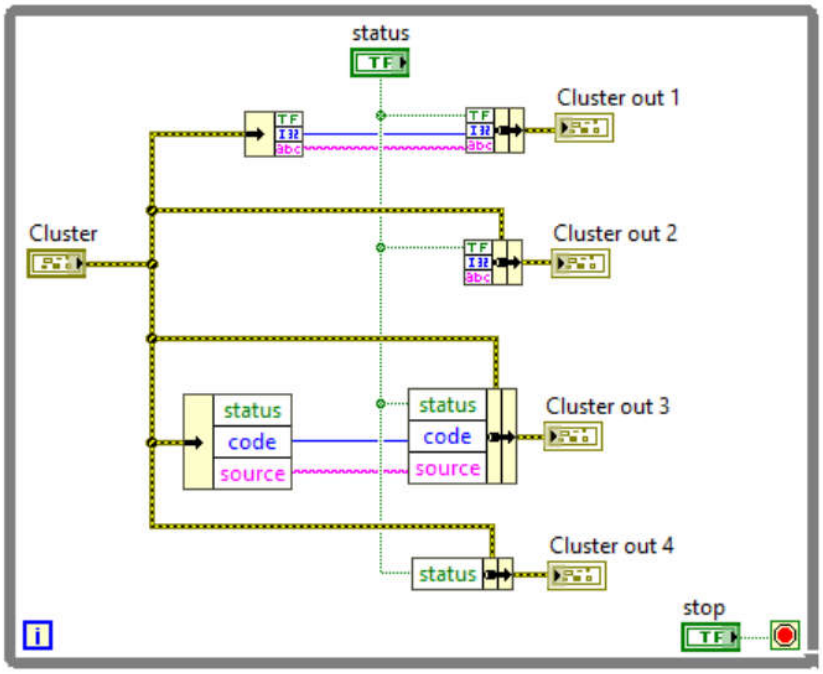
Sl. 3.5.9. Promena redosleda promenljivih u *error* klasteru

9. Izbrisati *Unbundle by name* funkciju iz tačke 7, potom uneti novu funkciju *Unbundle by name* i povezati je sa *error* klasterom koji sada ima novi redosled promenljivih. Uočiti da je sada redosled promenljivih na izlazu *Unbundle by name* funkcije u skladu sa novim redosledom promenljivih u klasteru, Sl. 3.5.10.

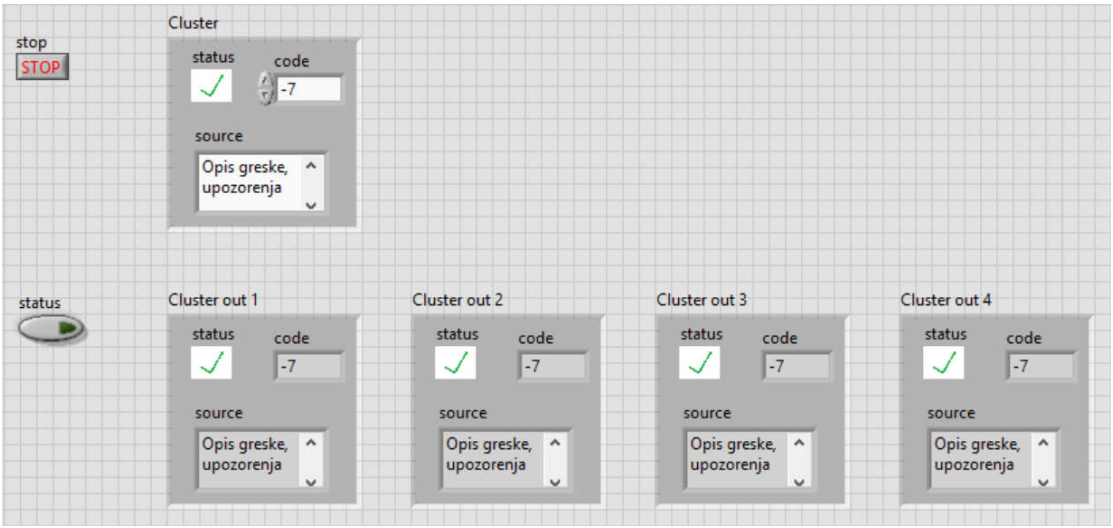


Sl. 3.5.10.

10. Iskopirati *error* klaster kontrolu (selektovati je, pritisnuti CTRL+C na tastaturi, a potom CTRL+V). Iskopiranu kontrolu „pretvoriti“ u *error* klaster indikator (selektovati kontrolu, kliknuti desnim tasterom miša, opcija *Change to Indicator*).
11. Kreirati *Block Diagram* kao na Sl. 3.5.11. i *Front Panel* kao na Sl. 3.5.12. koristeći *error* klaster kontrolu, 4 *error* klaster indikatora, *Unbundle* funkciju, *Unbundle by name* funkciju i logičke kontrole *status* i *stop*.



Sl. 3.5.11.



Sl. 3.5.12.

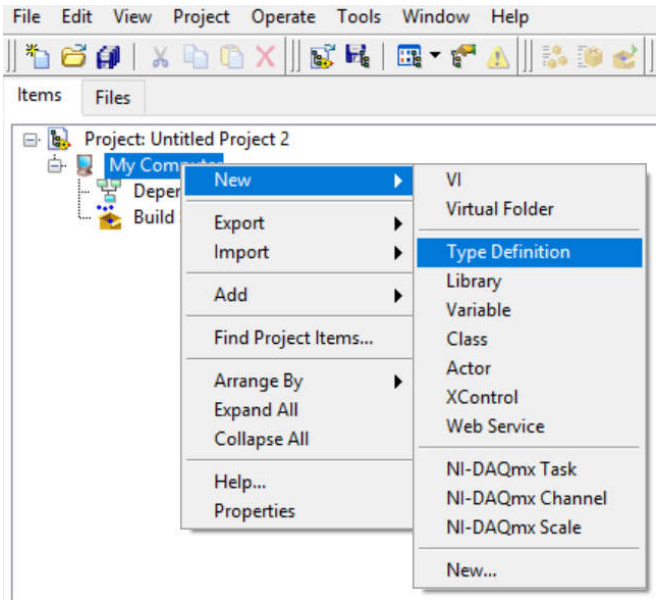
12. Pokrenuti program i uočiti kako se promenom stanja logičke kontrole *status* 

menja stanje promenljive *status* u *error* klaster indikatorima. Uočiti i da se vrednosti promenljivih *code* i *source* iz *error* klaster kontrole preslikavaju u odgovarajuće vrednosti *error* klaster indikatora.
13. Zaustaviti program i sačuvati ga kao *Error_klaster.vi*.

3.6 Striktno definisanje kontrola/indikatora

Ukoliko u programskom kôdu na više mesta postoje kontrole/indikatori koji treba da budu identičnih osobina, preporuka je da se najpre napravi njihov „templejt“, a da se onda na na sva potrebna mesta postavljaju instance tog templejta. „Templejt“ može biti manje striktan (*type definition*), tj. može da omogući da samo tip podataka bude zajednički svima, a može biti i u potpunosti striktan (*strict type definition*), tj. može da zahteva na svim instancama kontrola/indikatora izgled bude identičan (da osim istog tipa podataka imaju i istu boju, veličinu i stil).

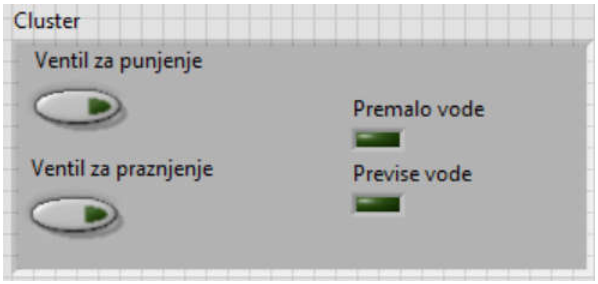
1. Kreirati novu projektnu datoteku, ***File»Create Project»Blank Project***.
2. Pozicionirati miša iznad opcije *My Computer*, uraditi klik desnim tasterom miša i odabrati opciju ***New»Type definition***. Otvoriće se prozor za definisanje kontrole/indikatora.
3. Postaviti u novootvorenom prozoru *Enum* kontrolu. Dizajnirati je po želji (boja, veličina, stil) i obavezno definisati i nekoliko vrednosti u padajućoj listi *Enum* kontrole (klik desnim tasterom miša, *Edit Items*). Na *Status Toolbar*-u padajućem meniju izabrati „*type definition*“.
4. Sačuvati kontrolu kao *Enum.ctl* datoteku.
5. Kreirati novi .vi u okviru tekućeg projekta. Na njegov *Front Panel* „prevući“ *Enum.ctl* datoteku iz stabla projekta. Uočiti da Enum kontrola koja je nastala na ovakav način više nema opciju *Edit Items*, ali da se veličina, boja i stil kontrole mogu menjati.
6. Otvoriti ponovo *Enum.ctl* datoteku iz stabla projekta. Na *Status Toolbar*-u padajućem meniju izabrati ovaj put „*strict type definition*“ i sačuvati *Enum.ctl* datoteku.
7. *Enum* kontrola koja je izvučena na novom .vi i dalje nema *Edit Items* opciju, ali su joj takođe zaključane i opcije za promenu veličine, boje i stila.



Sl. 3.6.1.

3.7 Projektni zadaci

Zadatak 3.7.1. – za samostalni rad
Napraviti program koji omogućava kontrolu nivoa vode u rezervoaru zapremine 1000 litara. Rezervoar poseduje ventil za punjenje i ventil za pražnjenje koji se mogu kontrolisati sa korisničkog interfejsa. Kada je otvoren ventil za punjenje, u rezervoar ulazi litar tečnosti na svakih 100 ms (pri svakoj iteraciji *While* petlje koja traje 100 ms povećati vrednost u rezervoaru za 1). Kada je otvoren ventil za pražnjenje, na svakih 100 ms iz rezervoara izlazi 2 litra tečnosti (pri svakoj iteraciji *While* petlje smanjiti vrednost u rezervoaru za 2). Ukoliko nivo vode pređe 990 litara, uključiti lampicu koja označava opasno visok nivo vode i ugasiti ventil za punjenje. Ukoliko nivo vode padne ispod 10 litara, uključiti lampicu koja označava opasno nizak nivo vode i ugasiti ventil za pražnjenje. Kontrole ventila i indikatore lampica grupisati zajedno u klaster kako bi se obezbedilo olakšano uključivanje još rezervoara u budućnosti. Pri pokretanju programa, nivo vode je 100 litara i ventili su zatvoreni. Pri zaustavljanju programa obezbediti da su oba ventila zatvorena. Koristiti klaster definisan kao na Sl. 3.7.1.



Sl. 3.7.1.

Zadatak 3.7.2. – za samostalni rad
Napraviti program za igru na sreću. U ovoj igri na sreću povlačenjem ručice na korisničkom interfejsu na dole u članove tročlanog niza brojeva počinju da se upisuju slučajne celobrojne vrednosti od 0 do 100. Obezbediti da se vrednosti menjaju na 2 ms. Vraćanjem ručice u početni položaj se upisivanje vrednosti prekida i trenutno zatečeni brojevi se sabiraju. Ukoliko je rezultat sabiranja veći ili jednak 150, igrač dobija poen. U suprotnom igrač gubi dva poena. Igra se završava kada igrač odluči da unovči svoje poene ili kada izgubi sve poene. Početni broj poena pri pokretanju igre je 5.
Dodatni zahtevi:

- Obavestiti igrača da je izgubio sve poene pre zaustavljanja programa.
- Ukoliko igrač pritisne dugme “Unovči”, dati mu mogućnost da se predomisli.

Lekcija 4 – Akvizicija i generisanje signala

Cilj

Cilj vežbe je da studente:

- upozna sa testiranjem ***N**ational **I**nstruments **D**ata **AcQ**uisition* (*NI DAQ*) uređaja pomoću *NI Measurement & Automation Explorer (NI MAX)* softvera
- upozna sa mogućnostima simulacije *NI DAQ* uređaja
- osposobi ih za samostalno kreiranje *LabVIEW* aplikacija za analognu akviziciju podataka pomoću *Express VIs* (*DAQ Assistant*)
- upozna sa osnovama akvizicije u *LabVIEW* paketu pomoću *DAQmx VIs*.
- osposobi ih za samostalno kreiranje *LabVIEW* aplikacija za analognu i digitalnu akviziciju i generisanje signala pomoću *Standard VIs* (*DAQmx VIs*)
- upozna sa *LabVIEW* funkcijama za generisanje zvučnih signala
- upozna sa konceptom trigerovane akvizicije u *LabVIEW* paketu pomoću DAQmx VIs
- osposobi ih za samostalno kreiranje *LabVIEW* aplikacije koja omogućava generisanje digitalnog upravljanja kada signal prikupljen na analognom ulazu zadovoljava zadati uslov.

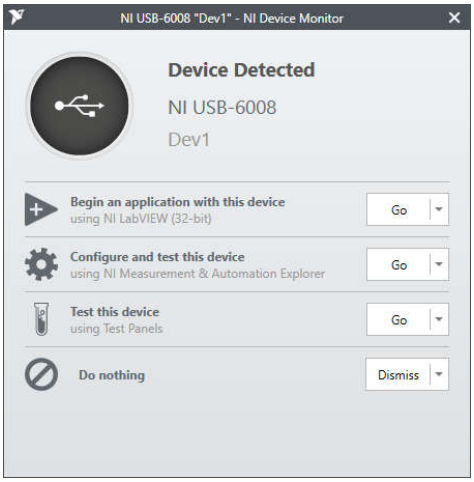
Oprema

- Računar sa instaliranim ***LabVIEW*** softverskim paketom i ***NI DAQ-mx*** drajverima.
- NI USB A/D konvertor
- Otpornici, termistor, dioda, fotootpornik, mikrofon, slušalice.

NAPOMENA: Nakon instalacije *LabVIEW Development* ili *Professional* okruženja, treba dodati i *NI DAQ-mx* drajvere. Odgovarajuću verziju drajvera je moguće download-ovati sa sajta http://www.ni.com/.

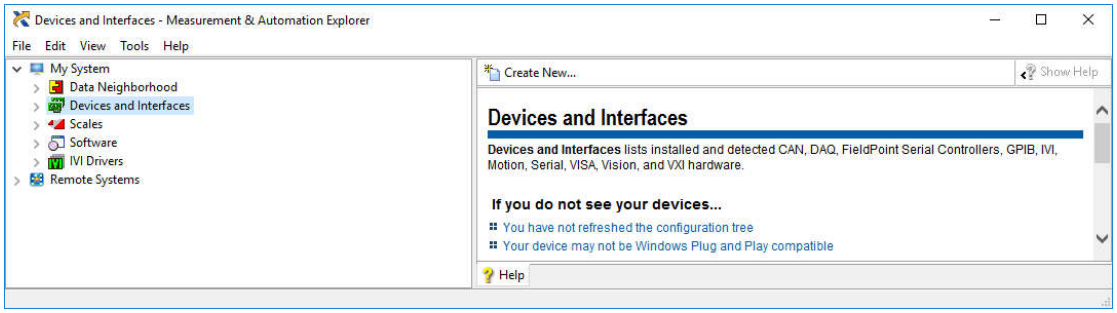
4.1 NI MAX *testiranje NI A/D kartice*

1. NI USB A/D karticu preko USB kabla priključiti na računar. Ako se pojavi *NI Device Monitor* kao na Sl. 4.1.1. izabrati *Configure and test this device* opciju. *NI Device Monitor* služi, nakon detekcije priključenja NI hardvera, kao prečica do:
   - pokretanja *LabVIEW* okruženja (*Begin an application with this device*),
   - pokretanja *NI MAX* softvera (*Configure and test this device*)
   - otvaranje panela za testiranje *NI DAQ* uređaja (*Test this device*).



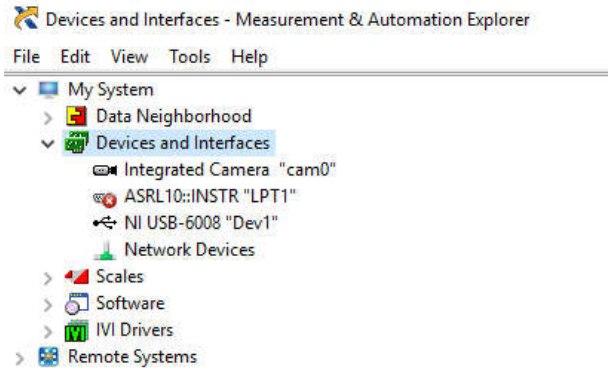
Sl. 4.1.1. *NI Device Monitor*

Ukoliko se pri priključenju *NI DAQ* uređaja ne pojavi prozor *NI Device Monitor* prikazan na Sl. 4.1.1, pokrenuti *NI MAX* softver (***NI M**easurement & **A**utomation **Explorer***) iz *Windows Start* menija. *NI MAX* softver služi za konfiguraciju, simulaciju i testiranje svih uređaja firme ***N**ational **I**nstruments*. *NI MAX* prozor je prikazan na Sl. 4.1.2.



Sl. 4.1.2. *NI MAX* softver

2. U meniju *My System* sa leve strane uočiti sekciju *Devices and Interfaces*. Klikom levog tastera miša na strelicu levo od natpisa *Devices and Interfaces*, proširiti tu sekciju. Pojaviće se lista svih *NI DAQ* uređaja priključenih na računar, Sl. 4.1.3.

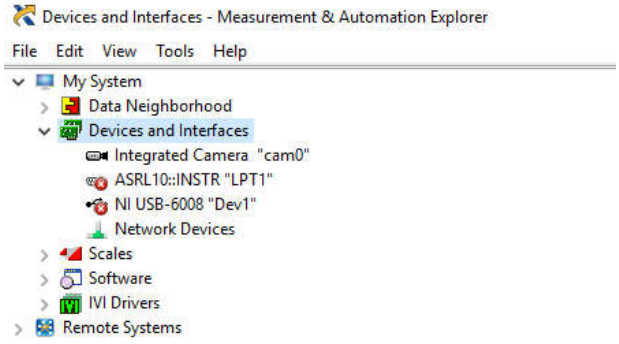


Sl. 4.1.3. Lista svih *NI DAQ* uređaja priključenih na računar

3. U listi *NI DAQ* uređaja uočiti ime NI USB A/D kartice. U zavisnosti od modela kartice, pojaviće se ime NI USB-6008 ili NI USB-6009 i slično. Pored imena A/D kartice nalazi se i natpis "DevN" gde broj N predstavlja tzv. *device number* tj. referentni broj *NI DAQ* uređaja. Na Sl. 4.1.3 je referentni broj za NI USB-6008 uređaj jednak 1.

NI MAX čita informacije iz zapisa *Device Manager*-a u *Windows Registry*-u i pridružuje svakom instaliranom *NI DAQ* uređaju jedan referentni broj. *NI MAX* čuva referentne brojeve uređaja i konfiguracione parametre u *Windows Registry*-u.

4. Isključiti NI USB A/D karticu iz računara. Na listi *NI DAQ* uređaja se može primetiti da kartica više nije priključena, Sl. 4.1.4. Na laptopu koji ima integrisanu kameru će i kamera biti prijavljena kao priključeni uređaj sa odgovarajućim referentnim brojem "camN" (npr. *Integrated Camera "cam0"* na Sl. 4.1.3. i Sl. 4.1.4). Serijski port će takođe biti prijavljen u listi detektovanih uređaja, sa oznakom koja na početku ima karaktere "ASRL".

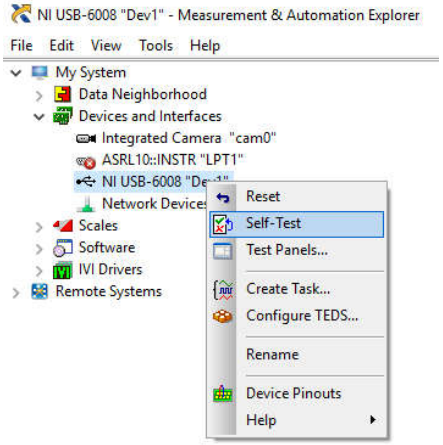


Sl. 4.1.4. Nema priključenih NI USB uređaja na računar

5. Priključiti NI USB A/D karticu na računar i uočiti da će biti ponovo identifikovana sa istim *device number*, čak i ako je promenjen USB port na koji je priključena.

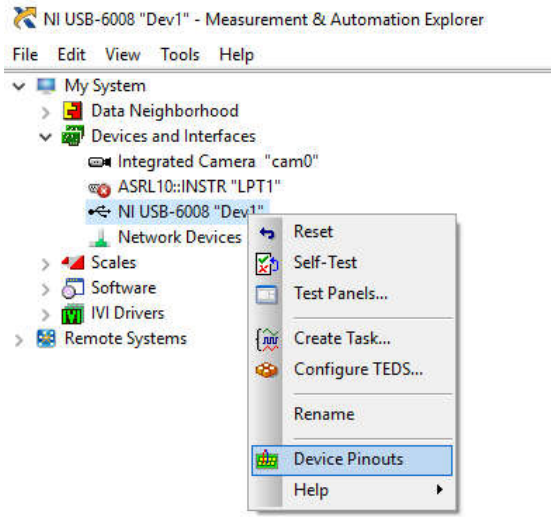
6. Na *Internet*-u pronaći specifikaciju NI USB A/D kartice koji se koristi u vežbi i proveriti koje su njegove osnovne karakteristike: broj analognih ulaza i izlaza, broj digitalnih ulaza i izlaza, rezolucija, maksimalna frekvencija odabiranja, naponski opseg ulaznih signala, strujno ograničenje itd. Neki od ovih podataka su naznačeni i na samom kućištu A/D kartice. *Pitanja: Kolika je maksimalna frekvencija odabiranja A/D konvertora ako se koristi 1 analogni kanal? Kolika je maksimalna frekvencija odabiranja A/D konvertora ako se koriste 2 analogna kanala (poznato je da se višekanalna akvizicija obavlja multipleksiranjem signala na jednom A/D konvertoru, tj. da ne postoje posebni A/D konvertori za svaki od kanala)?*

7. Pokrenuti *self-test* NI USB A/D kartice: klik desnim tasterom miša na ime *NI DAQ* uređaja, a potom izabrati opciju ***Self-Test*** u padajućem meniju, Sl. 4.1.5. Ovim postupkom se testira komunikacija *NI DAQ* uređaja sa resursima računara.



Sl. 4.1.5. *Self-Test* opcija *NI DAQ* uređaja

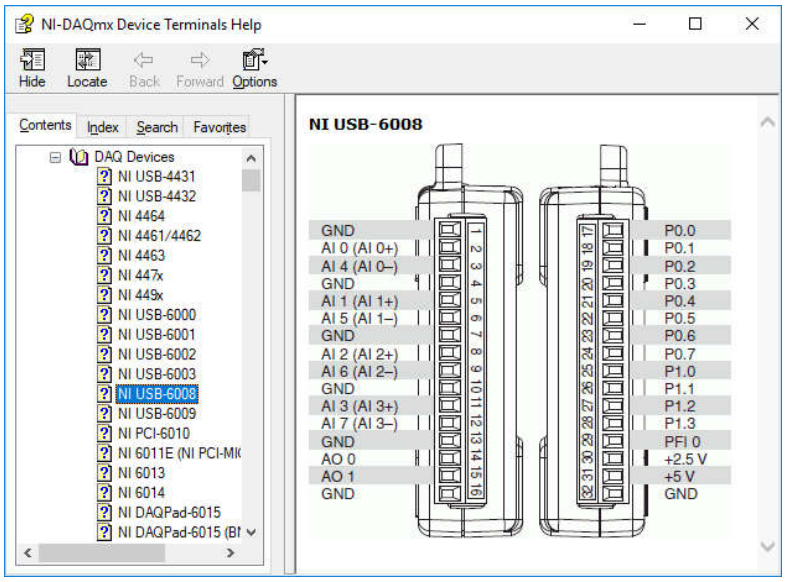
8. Proveriti raspored pinova (“*pinout*”) za *NI DAQ* uređaj: klik desnim tasterom miša na ime NI USB A/D kartice, a potom izabrati opciju ***Device Pinouts***, Sl. 4.1.6.



Sl. 4.1.6. *Device Pinouts* opcija *NI DAQ* uređaja

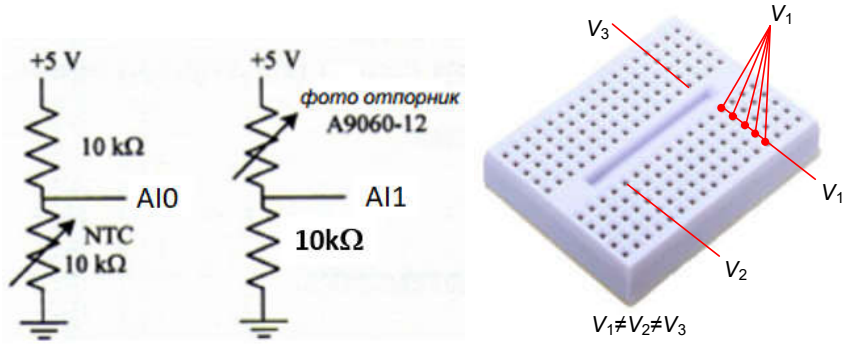
Raspored pinova *NI DAQ* uređaja za NI USB-6008 karticu je prikazan na Sl. 4.1.7:

- sa AI su označeni analogni ulazi 0, 1, 2, ... 7 (***a**nalog **i**nputs*),
- sa AO su označeni analogni izlazi 0 i 1 (***a**nalog **o**utputs*),
- GND je masa kartice (svi GND priključci su na istom potencijalu)
- sa Px.y su označeni digitalni ulazi/izlazi pri čemu je „x“ redni broj digitalnog porta (kanala 0 ili 1), a „y“ redni broj digitalne linije 0, 1, 2, ... 7. Isti pin može biti ulazni ili izlazni u zavisnosti od toga kako mu se softverski zada funkcija
- sa PFI su označeni digitalni ulazi višestruke namene (*Programmable Function Inputs*) koji se obično koriste kao ulazi „triger“ signala
- +5V je pin koji generiše 5V (može da se koristi kao izvor napona, ali obratiti pažnju na maksimalnu struju koji taj izvor može da dâ)
- +2.5V je pin koji generiše 2.5V (može da se koristi kao izvor napona, ali obratiti pažnju na maksimalnu struju koji taj izvor može da dâ).



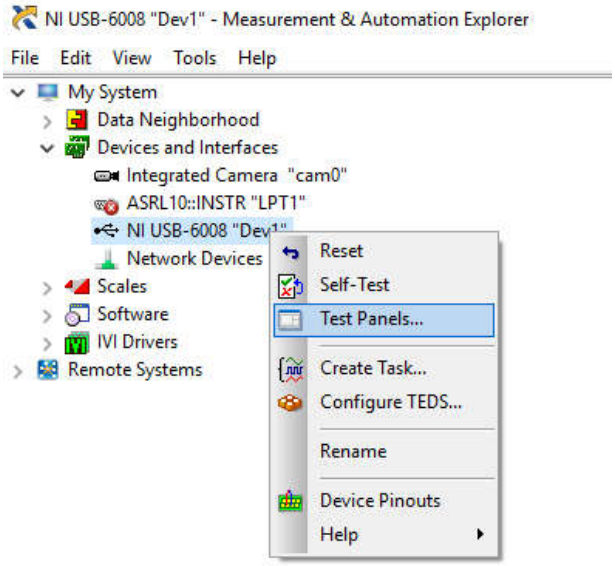
Sl. 4.1.7. *Device Pinouts* za NI USB-6008

9. **Isključiti NI USB A/D karticu iz računara i povezati je** prema šemi na Sl. 4.1.8. Nakon povezivanja ponovo priključiti NI USB A/D karticu na računar. NTC je otpornik sa negativnim temperaturnim koeficijentom, tj. njegova otpornost se pri povećanju temperature smanjuje. Fotootpornik je otpornik čija se otpornost menja pod uticajem svetlosti koja pada na njega.



Sl. 4.1.8: Redna veza otpornika od 10 kΩ i NTC otpornika od 10 kΩ (levo) i redna veza otpornika od 10 kΩ i fotootpornika (sredina), ekvipotencijalne tačke na protobordu

10. Otvoriti panel za testiranje NI USB A/D kartice (analognih ulaza/izlaza, digitalnih ulaza/izlaza i sl.): klik desnim tasterom miša na ime *NI DAQ* uređaja, a potom izabrati opciju ***Test Panels...***, Sl. 4.1.9.

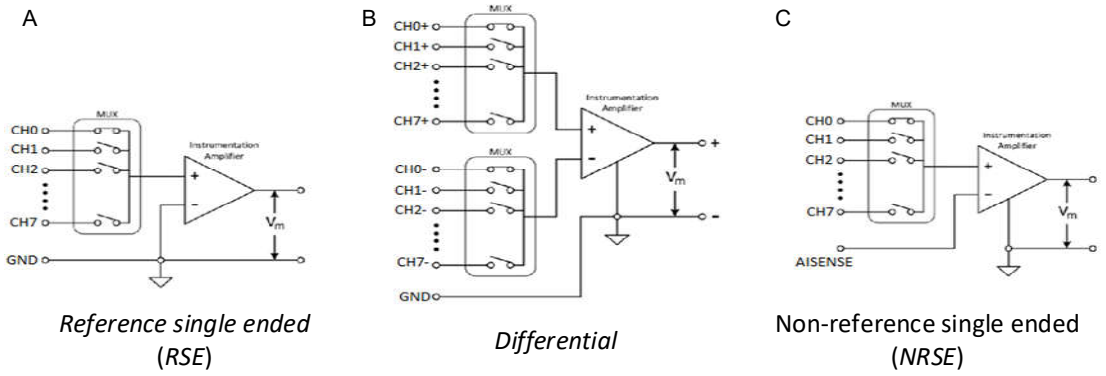


Sl. 4.1.9. Pokretanje opcije za testiranje NI USB A/D kartice

11. Na ***Analog Input*** listu test panela, izabrati sledeće parametre testiranja za analogni kanal AI0 na koji je povezan izlaz kola sa Sl. 4.1.8:
    a. kontinualna akvizicija signala pri testiranju, ***Mode***="Continuous"
    b. frekvencija odabiranja 100 Hz, ***Rate***=100 Hz
    c. broj odbiraka koji se odjednom prikazuje na ekranu 100, ***Samples To Read***=100)
    d. maksimalna vrednost ulaznog signala +3 V
    e. minimalna vrednost ulaznog signala +1 V
    f. konfiguracija ulaznih priključaka je ***RSE*** (*Reference Single Ended*).

NAPOMENA: obratiti pažnju na to da u *RSE* konfiguraciji A/D kartice meri signal koji je povezan između analognog ulaza i mase (u šemi sa Sl. 4.1.8 se meri signal

između analognog ulaza 0 (AI0) i GND). *RSE* konfiguracija merenja je prikazana na Sl. 4.1.10A. Ostale moguće konfiguracije povezivanja ulaza su prikazane na Sl. 4.1.10B (*differential*, diferencijalno merenje, tj. merenje signala između dva analogna ulaza) i Sl. 4.1.10C (*NRSE, non-reference single ended*, tj. merenje signala između analognog ulaza i AISENSE priključka koji nije povezan na GND).



Sl. 4.10. Različite konfiguracije za povezivanje ulaznih signala na A/D karticu

12. Pokrenuti testiranje *NI DAQ* uređaja pomoću *NI MAX* softvera pritiskom na dugme ***Start***. Menjati napon na analognom kanalu "zagrevanjem" NTC otpornika.
13. Završiti testiranje *NI DAQ* uređaja pritiskom na dugme ***Stop***.
14. Promeniti konfiguraciju ulaznih priključaka na ***differential*** dok su sva ostala podešavanja kao u tački 11. Koliko sada iznosi merenje na Test Panel-u i zašto?

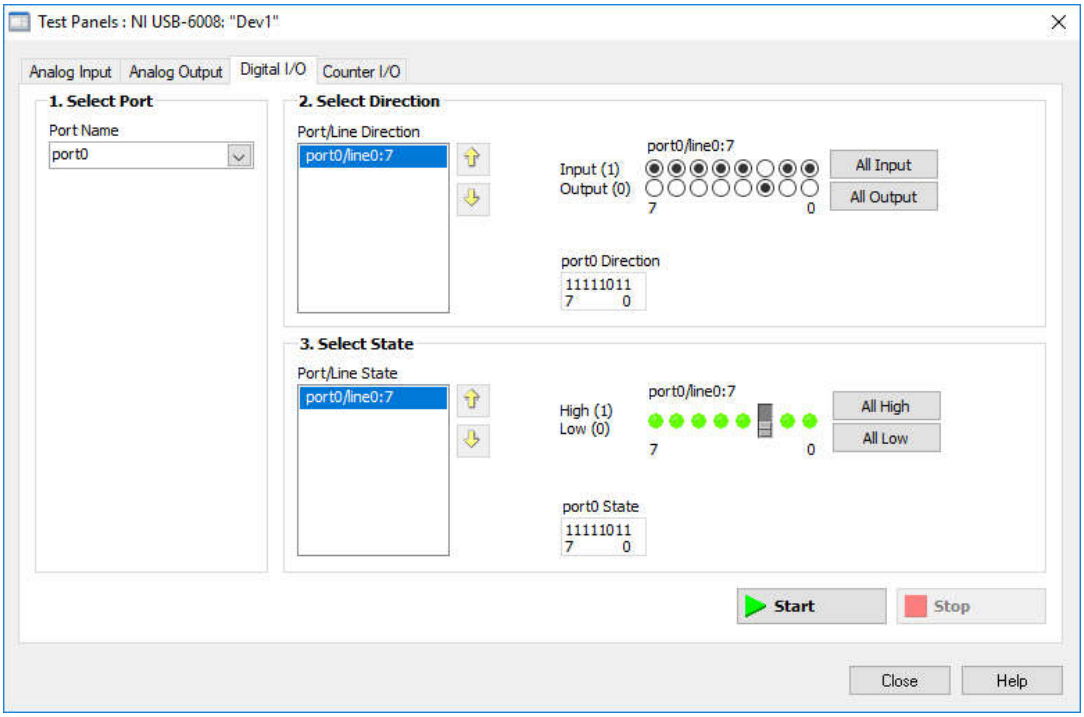
NAPOMENA: Pri merenju signala na analognom kanalu 0 u diferencijalnom môdu, merenje se vrši između priključaka AI0+ i AI0-, Sl. Sl. 4.1.10. Uočiti da AI0+ odgovara AI0 u *RSE* môdu, a AI0- odgovara AI4 u *RSE* môdu. AI0- pin nije nigde povezan, tj. potencijal na tom pinu odgovara potencijalu okoline.

15. Podesiti podešavanja za analogni ulaz AI1 tako da se pri menjaju osvetljenosti fotootpornika vidi odgovarajuća naponska promena na Test Panel-u.
16. **Isključiti NI USB A/D karticu iz računara i povezati je** prema šemi na Sl. 4.1.11. **Ostaviti povezane i analogne kanale sa Sl. 4.1.11 jer će koristiti u nastavku vežbanja.** Nakon povezivanja ponovo priključiti NI USB A/D karticu na računar.

Sl. 4.1.11. Veza linije 2 digitalnog kanala 0 (P0.2) i LED diode

2. Na **Digital I/O** listu test panel-a konfigurisati port da digitalna linija 2 na 0-tom kanalu bude ***Output***, Sl. 4.1.12. Ovim je softverski podešeno da ta linija bude izlazna dok su sve ostale ulazne. Pritisnuti dugme ***Start*** i u sekciji ***Select State*** prekidačima podešavati trenutno logičko stanje digitalnog izlaza 2: "0" ili "1", tj.

0 V ili 5 V. Primetiti da se dioda pali i gasi pri softverski zadavanim promenama napona na izlazu P0.2.

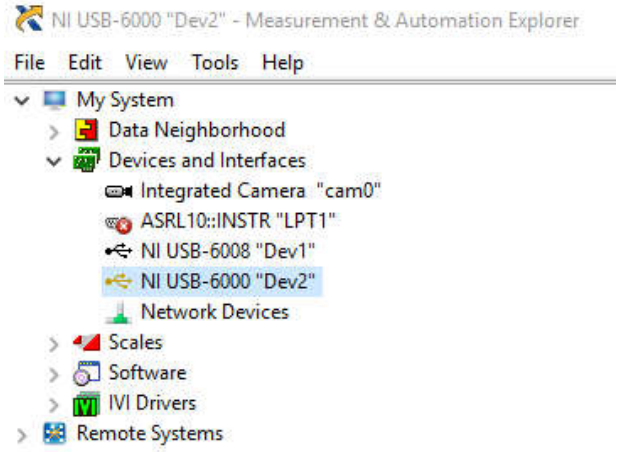


Sl. 4.1.12. Podešavanje digitalne linije 2 na 0-tom kanalu da bude ***Output***, tj. da bude izlazni pin

3. **Ostaviti povezana kola sa Sl. 4.1.11** jer će se koristiti u delu *4.2.1 Akvizicija podataka primenom Express funkcija.*

4.1.1 Simuliranje *NI DAQ* uređaja pomoću *NI MAX* softvera

1. Kliknuti desnim tasterom miša na ***Devices and Interfaces*** i izabrati opciju ***Create New»Simulated NI-DAQmx Device and Modular Instrument***.
2. Izabrati proizvoljan *NI DAQ* uređaj. Prihvatiti izbor pritiskom na dugme ***OK***. Uočiti da će se u sekciji ***Devices and Interfaces*** pojaviti i simulirana kartica sa pridruženim *device number* kao i kod realnog uređaja, Sl. 4.1.13.



Sl. 4.1.13. Simulirani *NI DAQ* uređaj se pojavljuje na listi *Devices and Interfaces*

Simulirani *NI DAQ* uređaj se može koristiti pri kreiranju *LabVIEW* aplikacije za akviziciju i/ili generisanje signala na istovetan način kao i realan uređaj, s tim što će ulazni analogni signali biti sinusoidalnog oblika.

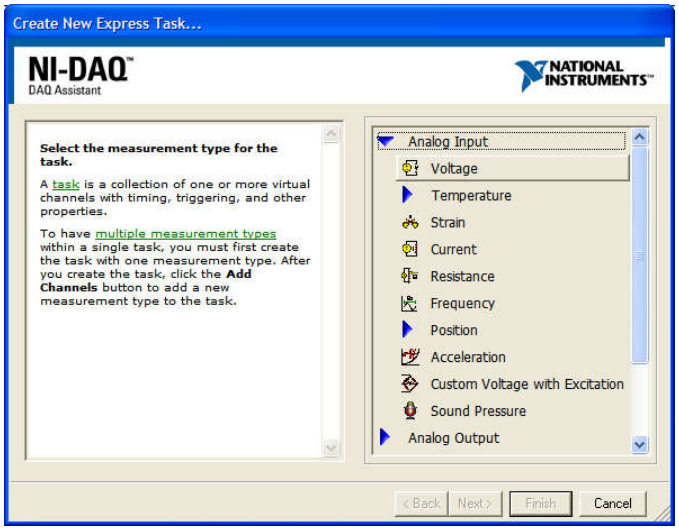
3. Isprobati korišćenje Test Panel-a za simuliranu karticu.

4.2 Akvizicija podataka

4.2.1 Akvizicija podataka primenom *Express* funkcija

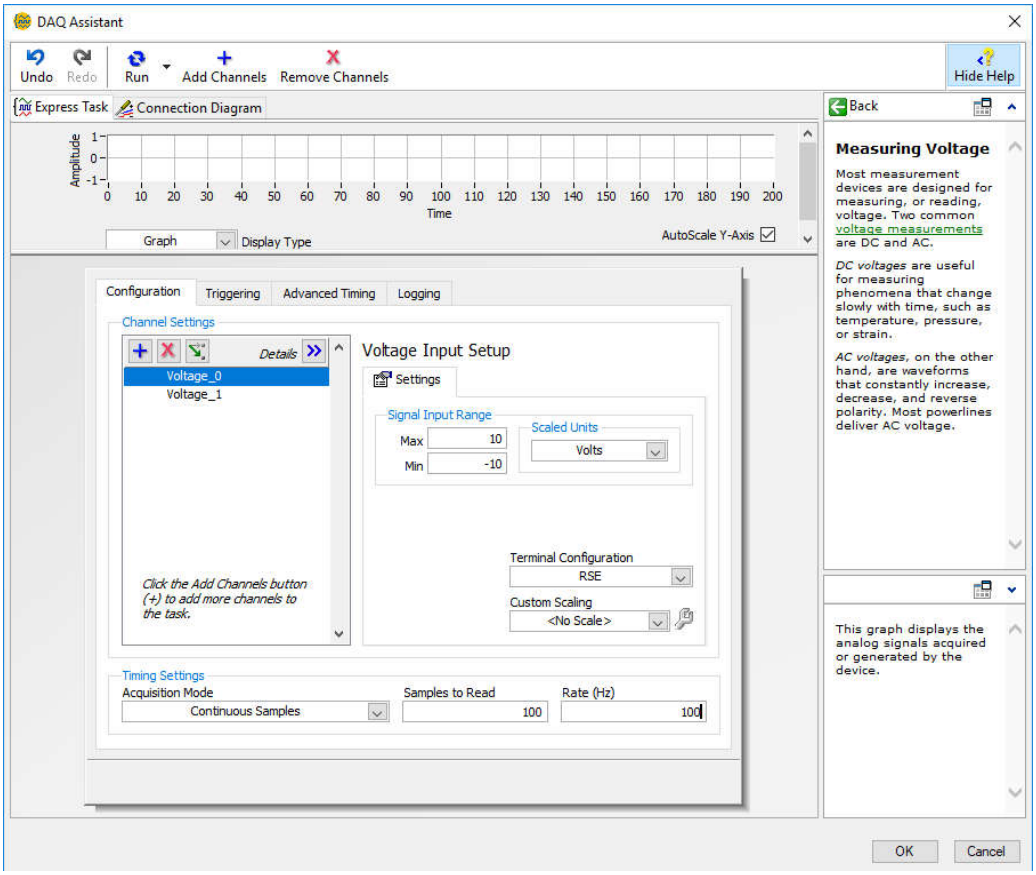
DAQ Assistant je funkcija koja konfiguriše "data acquisition task" (*DAQ task*) korak po korak, kroz biranje ponuđenih akvizicionih opcija u nekoliko konfiguracionih prozora. *DAQ Assistant* omogućava brzo i efikasno kreiranje akvizicionih aplikacija i posebno je pogodan za početnike u *LabVIEW* programiranju.

1. Pokrenuti *LabVIEW* okruženje i kreirati nov **.vi** program. *DAQ Assistant* funkcija se nalazi u paleti sa *Express VI* funkcijama ***Functions» Programming»Express»Input*** i u ***Functions» Measurement I/O»NI DAQmx***.
2. Kada se u Block dijagram unese funkcija *DAQ Assistant*, pojaviće se prozor *Create New Express Task* za konfiguraciju akvizicije, Sl. 4.2.1.



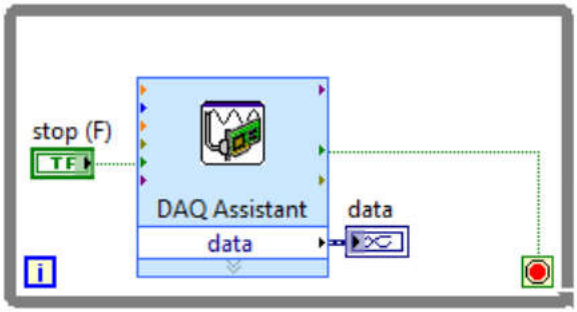
Sl. 4.2.1. Prozor *Create New Express Task*

3. Izabrati ***Analog Input»Voltage***, a potom selektovati analogne ulaze AI0 i AI1 instaliranog *NI DAQ* uređaja i pritisnuti dugme ***Finish***. Pojaviće se prozor kao na Sl. 4.2.2.
4. Definisati sledeće parametre analogne akvizicije:
   a. kontinualna akvizicija signala pri testiranju, ***Mode***="Continuous"
   b. frekvencija odabiranja 100 Hz, ***Rate***=100 Hz
   c. broj odbiraka koji se odjednom prikazuje na ekranu 100, ***Samples To Read***=100)
   d. maksimalna vrednost ulaznog signala +10 V
   e. minimalna vrednost ulaznog signala -10 V
   f. konfiguracija ulaznih priključaka je ***RSE*** (Reference Single Ended).
5. Prihvatiti parametre pritiskom na dugme ***OK***. Pojaviće se prozor sa pitanjem da želite prihvatiti opciju za automatsko iscrtavanje loop petlje (zato što je u prethodnoj tački izabran ***Mode***="Continuous") – prihvatiti pritiskom na ***OK*** dugme.



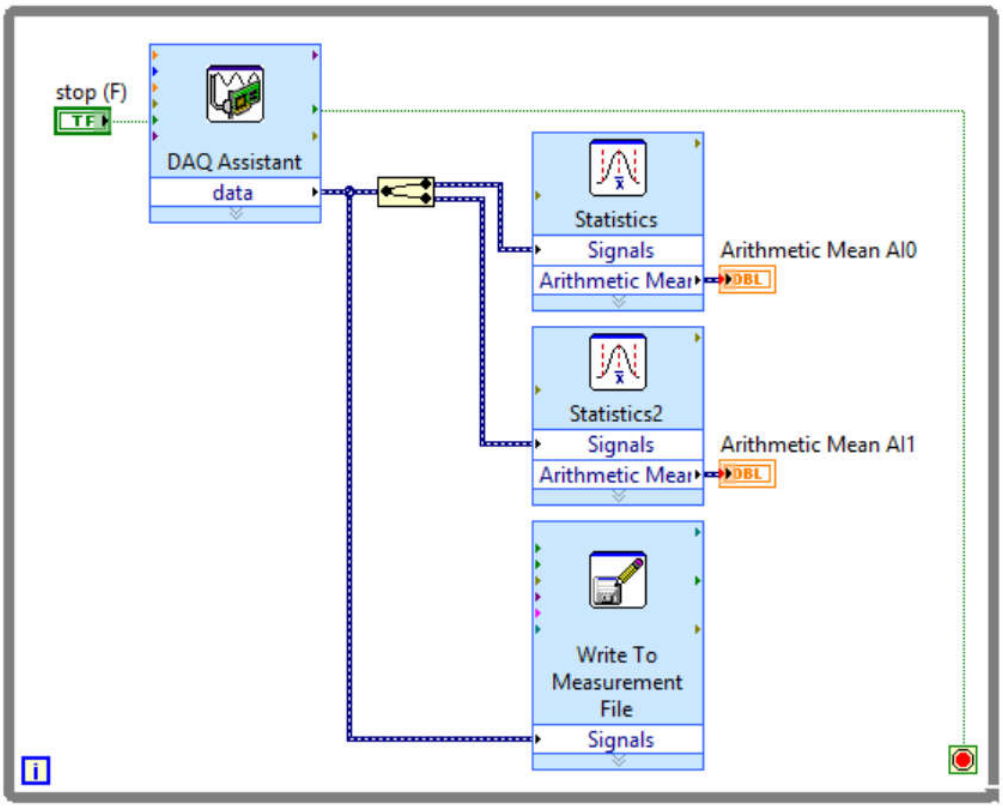
Sl. 4.2.2. Prozor za konfigurisanje akvizicije pomoću *DAQ Assistant*-a

6. U *Block Diagram*-u na izlazu ***data*** funkcije *DAQ Assistant* kliknuti desnim tasterom miša i izabrati opciju ***Create»Graph Indicator***, Sl. 4.2.3.
   a. Pokrenuti izvršavanje programa.
   b. Menjati napon na analognim kanalima i pratiti promene napona na grafiku interfejsa.



Sl. 4.2.3. Blok dijagram programa za analognu akviziciju pomoću *Express VI*-a (*DAQ Assistant*)

7. Pomoću *Express* funkcije *Statistics* koja se nalazi u ***Functions»Mathematics» Probability & Statistics*** omogućiti određivanje srednje vrednosti podataka (*Arithmetic Mean*) za svaki od analognih kanala. *Data* promenljivu razdvojiti na dva signala koristeći Split Signals funkciju, Sl. 4.2.4.
8. Omogućiti i upis podataka u datoteku pomoću funkcije *Write to Measurement File*, Sl. 4.2.4. Pokrenuti izvršavanje programa, a potom ga zaustaviti i pomoću *Notepad*-a otvoriti i pogledati sadržaj datoteke.



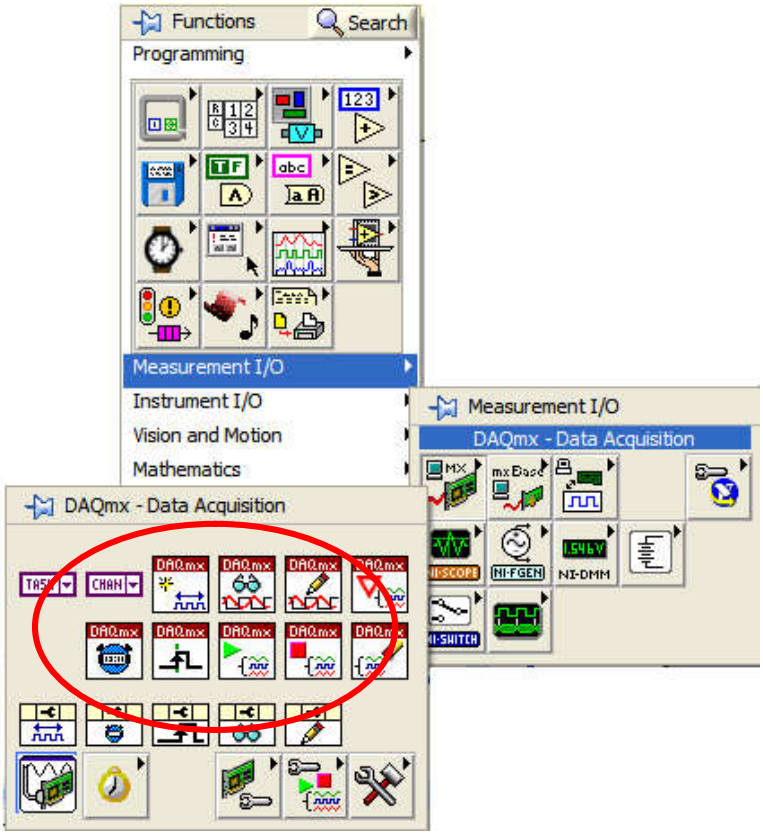
Sl. 4.2.4. Blok dijagram programa za analognu akviziciju, obradu i upis u datoteku pomoću *Express VI*-a (*DAQ Assistant*)

9. Sačuvati program kao *akvizicija_DAQAssistant.vi*.

4.2.2 Akvizicija podataka primenom DAQmx VIs

DAQ Assistant je jednostavna funkcija za korišćenje i konfigurisanje akvizicije, ali izvršavanje *LabVIEW* aplikacija kreiranih pomoću ovog *Express VI*-a angažuje više računarskih resursa nego kada se koriste *DAQmx VIs* (ugradjene u *Functions* paletu nakon instalacije *NI DAQmx drajvera:* ***Functions»Measurement I/O»DAQmx – Data Acquisition***, Sl. 4.2.5). Kada je brzina izvršavanja kritična, preporučuje se korišćenje *DAQmx VIs* za akviziciju podataka. *DAQmx VIs* obuhvataju VI-ove za konfigurisanje, započinjanje i zaustavljanje akvizicije, upis akvizicionih podataka u bafer i njihovo čitanje iz bafera.

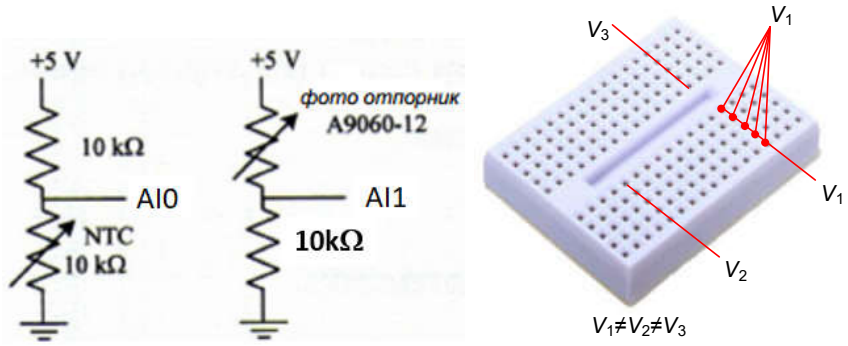
DAQmx VIs su polimorfni, tj. isti *VI*-ovi (*DAQmx Start Task, DAQmx Stop Task, DAQmx Read, DAQmx Write itd.*) se koriste bez obzira na tip akvizicije, ali njihovo konfigurisanje zavisi od tipa akvizicije.



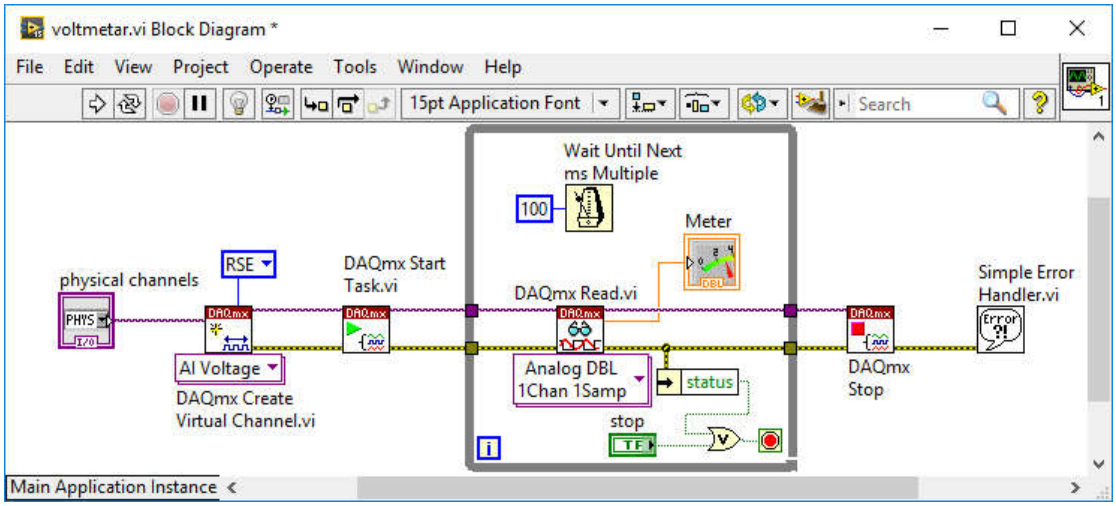
Sl. 4.2.5. DAQmx Vis: Functions»Measurement I/O»DAQmx – Data Acquisition

1. **Isključiti *NI DAQ* uređaj iz računara i povezati ga** prema šemi na Sl. 4.2.6. Nakon povezivanja ponovo priključiti *NI DAQ* uređaj na računar.
2. Pokrenuti *LabVIEW* okruženje i kreirati novi prazan *VI*.
3. Napraviti *Block Diagram* kao na Sl. 4.2.7. (i *Front Panel* kao na Sl. 4.2.8) za aplikaciju *voltmetar.vi* koristeći sledeće funkcije:
    a. *Functions»Measurement I/O»DAQmx – Data Acquisition»**DAQmx Create Virtual Channel*** – za konfigurisanje „virtuelnog“ kanala za akviziciju podataka. Pomoću *CTRL+H* pogledati kratak *Help* ove funkcije i raspored ulaznih/izlaznih priključaka. **NAPOMENA**: Ako se na ulazne priključke ne “dovedu“ vrednosti, biće korišćene *default*-ne vrednosti ***DAQmx***

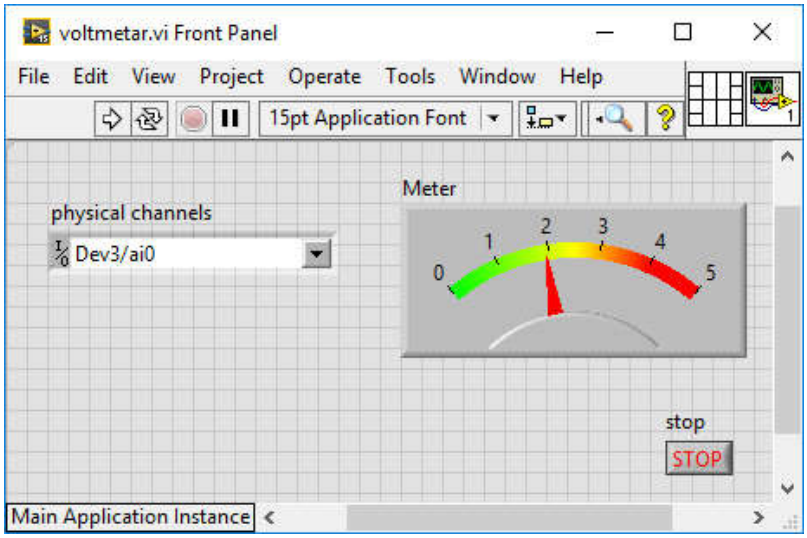
Create Virtual Channel.vi funkcije koje se mogu pogledati na njenom *Front Panel*-u: dva puta kliknuti na ikonicu funkcije i pojaviće se *Front Panel* kao na Sl. 4.2.9 gde se vidi da je *maximum* ulaznog signala podešen na +5 V, *minimum* ulaznog signala na -5 V, a *input terminal configuration* na *default*. **Nakon provere navedenih vrednosti zatvoriti *Front Panel* funkcije *DAQmx Create Virtual Channel.vi***. S obzirom na konfiguraciju kola na Sl. 4.2.6, gde su oba senzora vezana između mase i analognih ulaza, *input terminal configuration* treba podesiti na RSE (*Reference Single Ended*). To se postiže programski, tako što se na odgovarajući ulazni priključak funkcije ***DAQmx Create Virtual Channel.vi*** u *Block Diagram*-u aplikacije *voltmetar.vi* postavi ulazna konstanta *RSE*: iz palete *Tools* izabrati  , pa stati mišem iznad priključka *input terminal configuration* funkcije ***DAQmx Create Virtual Channel.vi***, pomoću klika desnog tastera miša otvoriti padajući meni u kome treba izabrati *Create»Constant*. Na sličan način treba napraviti i kontrolu ***physical channels***.



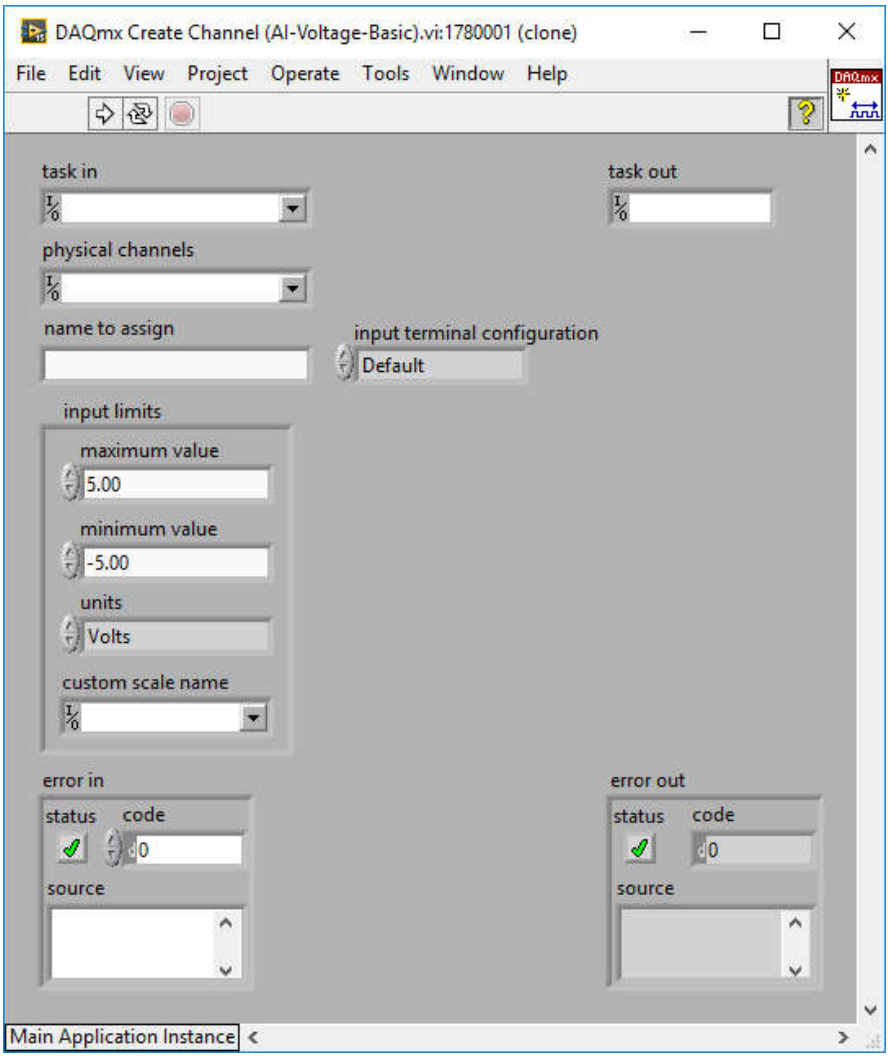
Sl. 4.2.6: Redna veza otpornika od 10 kΩ i NTC otpornika od 10 kΩ (levo) i redna veza otpornika od 10 kΩ i fotootpornika (sredina), ekvipotencijalne tačke na protobordu



Sl. 4.2.7. *Block Diagram* aplikacije *voltmetar.vi*



Sl. 4.2.8. *Front Panel* aplikacije *voltmetar.vi*



Sl. 4.2.9. *Front Panel* funkije ***DAQmx Create Virtual Channel.vi***

b. *Functions»Measurement I/O»DAQmx – Data Acquisition»**DAQmx Start Task*** – za startovanje akvizicionog zadatka (eng. *task*). Nakon izvršavanja ove funkcije, počinje odabiranje signala sa konfigurisanog analognog ulaza *NI DAQ*-a, a odbirci se smeštaju u bafer računara.

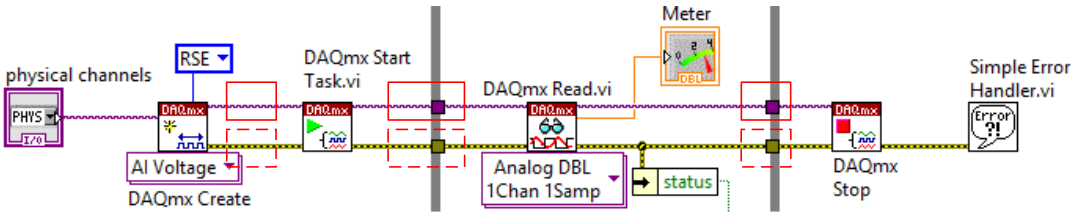
c. *Functions»Measurement I/O»DAQmx – Data Acquisition»**DAQmx Read*** – za čitanje odbiraka iz bafera računara. Uočiti da se ova funkcija nalazi unutar *While* petlje čime je omogućeno kontinualno čitanje odbiraka iz bafera sve dok korisnik ne pritisne dugme STOP na *Front Panel*-u ili dok se ne pojavi greška u procesu akvizicije (*status*=TRUE u izlaznom *Error Cluster*-u funkcije ***DAQmx Read***).

NAPOMENA 1: Da bi se omogućilo da računar osim čitanja odbiraka iz bafera ima vremena da radi i druge procese, unutar *While* petlje je postavljena i funkcija *Functions»Programming»Timing»**Wait Until Next ms Multiple.*** Ova funkcija omogućava čitanje odbiraka iz bafera na svakih *N* milisekundi (konkretno u ovom primeru 100 ms), a između dva uzastopna čitanja procesor je u mogućnosti da radi neke druge procese.

NAPOMENA 2: Da bi se omogućio regularan završetak rada aplikacije u slučaju greške u akviziciji (npr. *NI DAQ* uređaj nije priključen preko USB-a na računar, ili je konfigurisan pogrešan virtuelni kanal i sl.), za *Loop Condition* u *While* petlji se postavlja vrednost koja je rezultat **ILI** logičke operacije vrednosti promeljnive STOP i promenljive *status* (u *Error Cluster*-u funkcije ***DAQmx Read***).

d. *Functions»Measurement I/O»DAQmx – Data Acquisition»**DAQmx Stop*** – za zaustavljanje akvizicionog zadatka, tj. *task*-a.
e. *Functions»Programming»Dialog & User Interface»**Simple Error Handler*** – za prikaz obaveštenja o greški ukoliko je do nje došlo u procesu konfiguracije ili akvizicije.

4. Primetiti da su sve DAQmx funkcije međusobno povezane sa dve linije: *task* linijom i *error* linijom, Sl. 4.2.10. Ovim linijama se prosleđuju između *DAQmx* funkcija informacije o akvizicionom zadatku i greški u akviziciji, respektivno.

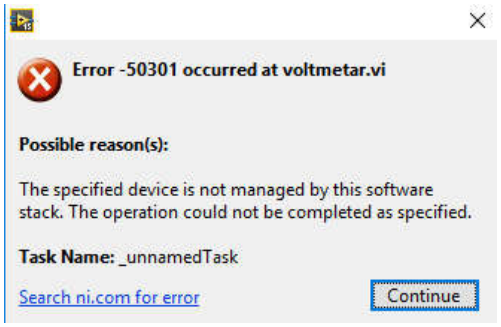


Sl. 4.2.10. Punom linijom su zaokružene *task* linije, a isprekidanom *error* linije

5. Proveriti u *NI MAX*-u koji je *device number NI DAQ* uređaja koji je priključen na računar.
6. Kontrolu *physical channels* na *Front Panel*-u podesiti da snima analogni ulaz *ai0* (selektovati vrednost *DevN/ai0*, gde je *device number NI DAQ* uređaja).
7. Pomoću *Edit Text* alatke u paleti *Tools* podesiti na indikatoru *Meter* na *Front Panel*-u da minimalna vrednost bude 0 V, a maksimalna 5 V.
8. Pokrenuti izvršavanje aplikacije *voltmetar.vi*. Zagrevati NTC termistor kako bi se videla promena napona na indikatoru *Meter* na *Front Panel*-u.
9. Zaustaviti izvršavanje programa pritiskom na dugme STOP.
10. Selektovati na kontroli *physical channels* na *Front Panel*-u analogni kanal *DevN/ai1*. Pokrenuti izvršavanje aplikacije *voltmetar.vi*. Menjati osvetljenost fotootpornika kako bi se videla promena napona na indikatoru *Meter* na *Front Panel*-u.

NAPOMENA: Ako se u toku izvršavanja aplikacije promeni vrednost kontrole *physical channels* na *Front Panel*-u, ta promena neće biti registrovana zato što aplikacija samo JEDNOM, na početku izvršavanja programa izvrši funkcije ***DAQmx Create Virtual Channel.vi*** i ***DAQmx Start Task,*** a potom počinje da izvršava programski kôd unutar *While* petlje.

11. Prilikom izvršavanja aplikacije *voltmetar.vi*, isključiti *NI DAQ* uređaj iz računara. Zahvaljući ILI operaciji nad vrednosti STOP promenljive i promenljive *status,* izvršavanje *While* petlje će se nakon isključivanja *NI DAQ* uređaja završiti. Potom će se izvršiti funkcija ***DAQmx Stop***, a funkcija ***Simple Error Handler*** će prikazati prozor kao na Sl. 4.2.11. Nakon pritiska na opciju *Continue* aplikacija *voltmetar.vi* će se završiti.

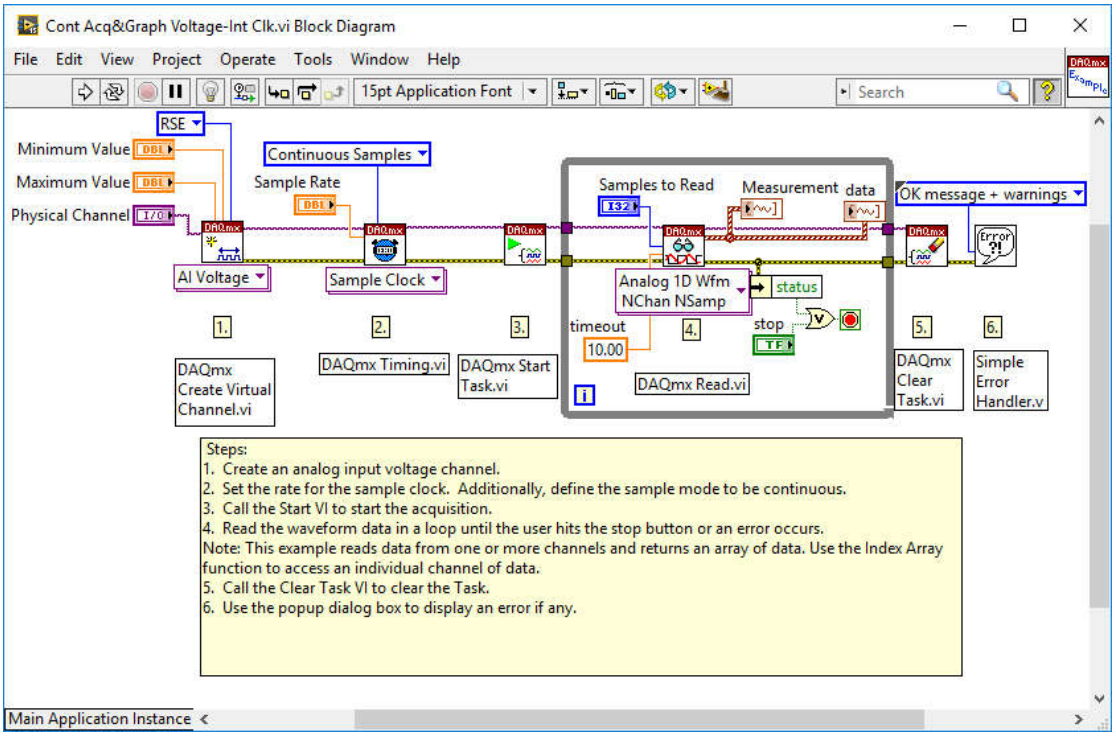


Sl. 4.2.11. Prozor o greški pri akviziciji

12. Sačuvati aplikaciju kao *voltmetar.vi.* **Ne treba razvezati kolo.**

4.3 Aplikacija za kontinualnu akviziciju analognih kanala

1. Otvoriti primer *Cont Acq&Graph Voltage-Int Clk.vi*. Pročitati objašnjenje rada kôda u *Block Diagram*-u, Sl. 4.3.1. **Uočiti da je za *input terminal configuration* dovedena ulazna promenljiva čija je vrednost *RSE*.**



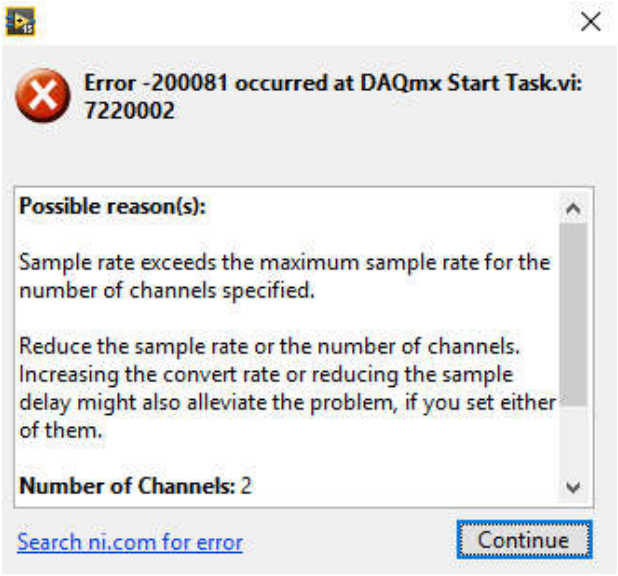
Sl. 4.3.1. *Block Diagram* primera *Cont Acq&Graph Voltage-Int Clk.vi*

NAPOMENA: Funkcijom ***DAQmx Timimg.vi*** se u programskom kôdu zadaje da akvizicija bude kontinualna, sa frekvencijom odabiranja zadatom pomoću kontrole *Sample Rate*.

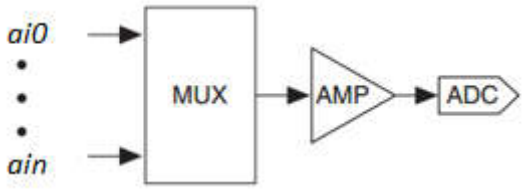
2. Kontrolu *Physical Channel* na *Front Panel*-u podesiti tako da se snima analogni ulaz *ai0* (selektovati vrednost *DevN/ai0*, gde je *device number NI DAQ* uređaja).
3. Pokrenuti izvršavanje aplikacije ***Cont Acq&Graph Voltage-Int Clk.vi***. Zagrevati NTC termistor kako bi se videla promena napona na indikatoru *Measurement* na *Front Panel*-u. Šta bi se videlo da nije *input terminal configuration* setovan na *RSE***?**
4. Kontrolu *Physical Channel* na *Front Panel*-u podesiti za snimanje DVA analogna ulaza *ai0* i *ai1* (nakon pritiska na strelicu kontrole odabrati opciju *Browse*, pa pri selekciji kanala držati pritisnuto CTRL ili SHIFT).
5. Pokrenuti izvršavanje aplikacije. Ako je *NI DAQ* uređaj NI USB 6008, pojaviće se greška kao na Sl. 4.3.2. Obratiti pažnju na to da je na *Front Panel*-u za frekvenciju odabiranja podešena vrednost *Sample Rate* = 10 000 Hz, što je prema specifikaciji maksimalna frekvencija odabiranja jednog kanala za NI USB 6008 (za NI USB 6009 je maksimalna frekvencija odabiranja 48 000 Hz, pa se za nju neće pojaviti prozor o greški). S obzirom na to da *NI DAQ* uređaji multipleksiraju analogne kanale pre analogno-digitalne konverzije, Sl. 4.3.3, pri akviziciji dva kanala je maksimalna frekvencija odabiranja 10 000 Hz/2=5 000 Hz. Podesiti na *Front Panel*-u za frekvenciju odabiranja vrednost *Sample Rate* = 5 000 Hz i

pokrenuti izvršavanje aplikacije. Zagrevati NTC termistor i menjati osvetljenost fotootpornika kako bi se videla promena napona na indikatoru *Measurement* na *Front Panel*-u.

6. Pomoću *Edit Text* alatke  u paleti *Tools* podesiti na indikatoru *Measurement* na *Front Panel*-u da minimalna vrednost bude 0 V, a maksimalna 5 V. Isprobati i *Autoscale Y* opciju indikatora (klik desnog tastera miša na grafik).

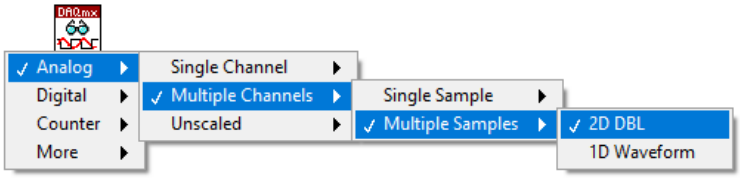


Sl. 4.3.2. Prozor greške usled neprilagođene frekvencije odabiranja



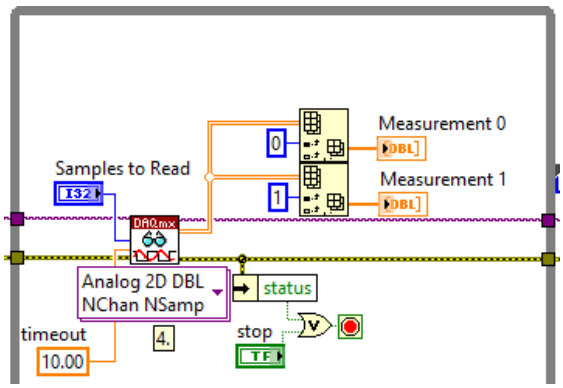
Sl. 4.3.3. Multipleksiranja analognih kanala – AMP je pojačavač, ADC je analogno-digitalni konvertor

7. Uočiti da je podešeno da se podaci čitaju pomoću ***DAQmx Read.vi*** funkcije u vidu *Waveform* tipa podataka (*data* indikator) koji u strukturi sadrži: početno vreme *t0*, vreme odabiranja *dt* i niz odbiraka *Y*.
8. Podesiti polimorfnu funkciju ***DAQmx Read.vi*** za čitanje prikupljenih odbiraka signala u DBL formatu, Sl. 4.3.4. Javiće se greška zbog povezivanja različitih tipova podataka, pa treba obrisati indikator *data*. DBL format sadrži samo matricu *Y* odbiraka signala na koju se mogu primenjivati funkcije za matrice, a ne sadrži početno vreme *t0*, vreme odabiranja *dt*. U svakom redu matrice su smešteni odbirci po jednog analognog kanala matrice: ako matrica ima dimenzije *m*x*n*, to znači da ima po *n* odbiraka za *m* analognih kanala. Pokrenuti izvršavanje aplikacije.



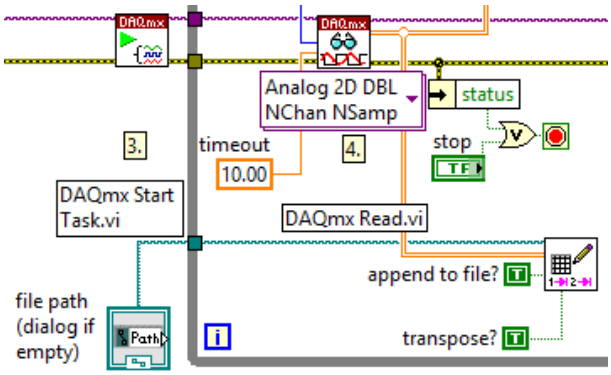
Sl. 4.3.4. Polimorfni *DAQmx Read.vi* za čitanje prikupljenih odbiraka signala

9. Koristeći *Index Array* funkciju prikazati odbirke kanala *ai0* i *ai1* na dva odvojena grafika, Sl. 4.3.5.



Sl. 4.3.5. Prikaz *ai0* i *ai1* kanala na dva odvojena grafika

10. Obezbediti da se prikupljeni odbirci oba analogna kanala kontinualno čuvaju u datoteku (pri svakoj iteraciji *While* petlje treba da se dodaju u datoteku novi podaci bez brisanja prethodnih). Svaka kolona u datoteci treba da sadrži odbirke jednog analognog ulaza.
Pomoć: Koristiti funkciju *Functions»Programming»File I/O»**Write Delimited Spreadsheet File*** kojoj su ulazne promenljive *append to file* (dodavanje u datoteku) i *transpose* (transponovanje) podešene na TRUE, Sl. 4.3.6.

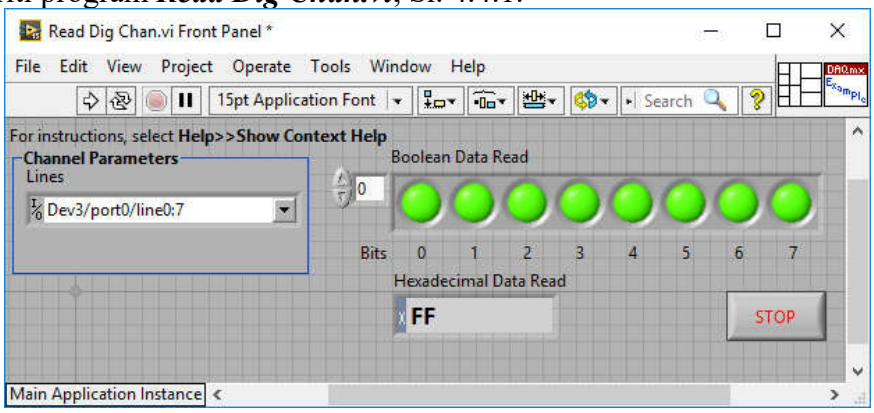


Sl. 4.3.6. Čuvanje odbiraka sa analognih ulaza u *spreadsheet* datoteku

11. Napraviti kontrolu *file path*. Na *Front Panel*-u kliknuti na nju desnim tasterom miša i izabrati opciju *Browse Options...* Selektovati opcije *Files* i *New or Existing*. U kontrolu na *Front Panel*-u uneti putanju datoteke za snimanje (npr. C:\proba.txt ili neka druga lokacija gde je dozvoljeno pisanje). Pokrenuti izvršavanje programa. Pregledati sadržaj datoteke pomoću *Notepad*-a. Uočiti da datoteka ima dve kolone, pri čemu 0-ta kolona sadrži odbirke sa analognog kanala *ai0*, a prva kolona sadrži odbirke sa analognog kanala *ai1*.
12. Napraviti posebne datoteke kada su ulazne promenljive funkcije ***Write Delimited Spreadsheet File*** *append to file* (dodavanje u datoteku) i *transpose* (transponovanje) podešene na FALSE. Koja je razlika u zapisu u datoteci?
13. Sačuvati aplikaciju kao *analogna_akvizicija.vi*.
14. **Razvezati kolo na protobordu**.

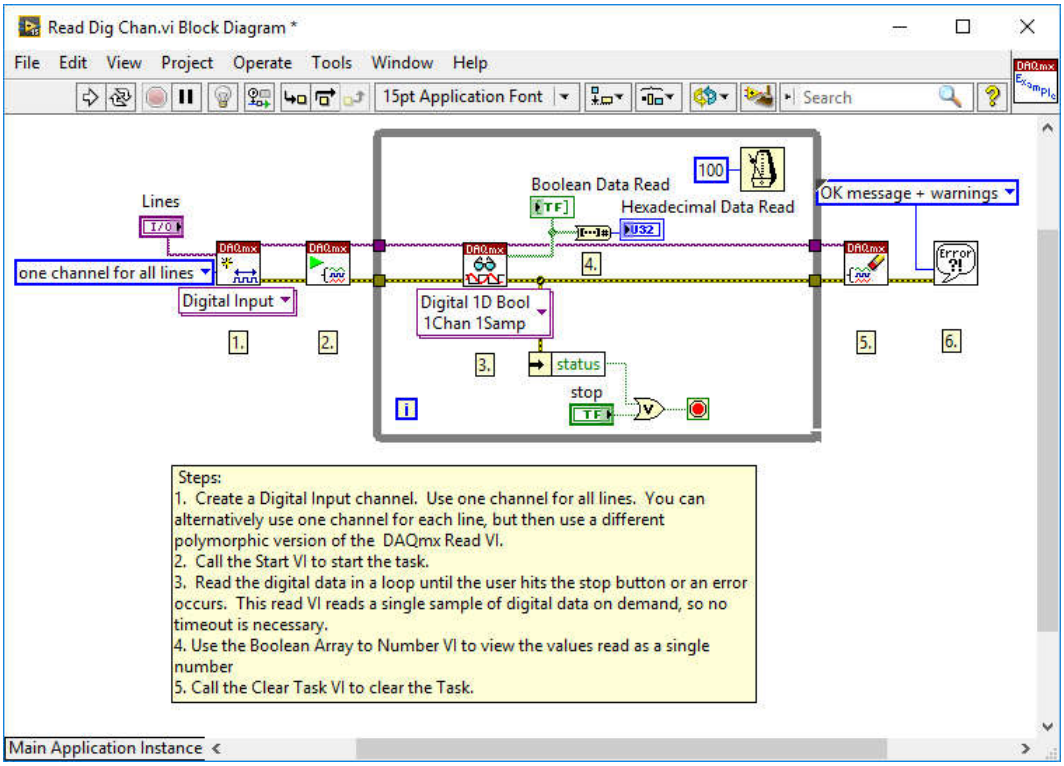
4.4 Korišćenje digitalnog ulaza

1. Otvoriti program ***Read Dig Chan.vi***, Sl. 4.4.1.



Sl. 4.4.1. *Front Panel* aplikacije ***Read Dig Chan.vi***

2. Kontrolu *Lines* podesiti na 8 digitalnih linija (0:7) porta 0 (port 0) za *NI DAQ* čiji je *device number DevN*: ***DevN/port0/line0:7***.
3. Pokrenuti izvršavanje aplikacije. Kada digitalne linije nisu nigde priključene, na njima je stanje „1“, tj. napon na njima je 5 V.
4. Fizički, pomoću žice, povezati jednu od selektovanih digitalnih linija sa masom (pogledati *pinout NI DAQ* uređaja u *NI MAX*-u (prethodna vežba) ili oznake konektora na samoj kutiji). LED indikator na *Front Panel*-u za odgovarajuću digitalnu liniju će promeniti stanje sa „TRUE“ na „FALSE“.
5. Zaustaviti izvršavanje aplikacije.
6. Proučiti objašnjenje programskog kôda u *Block Diagram*-u, Sl. 4.4.2.

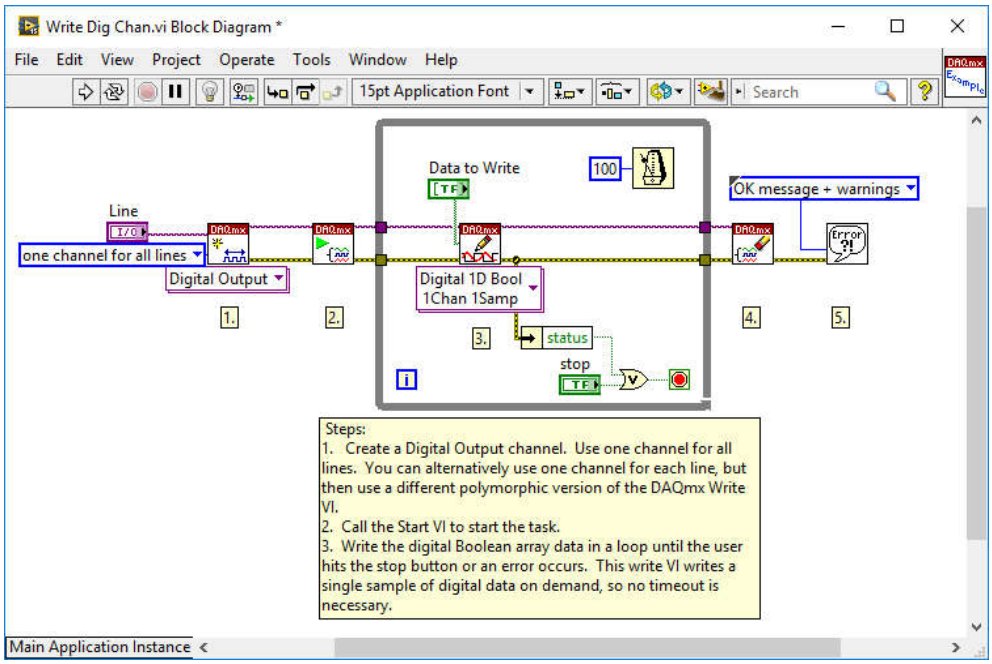


Sl. 4.4.2. *Block Diagram* aplikacije ***Read Dig Chan.vi***

4.5 Generisanje podataka pomoću DAQmx VIs

4.5.1 Korišćenje digitalnog izlaza

1. Otvoriti program ***Write Dig Chan.vi***.
2. Proučiti objašnjenje programskog kôda u *Block Diagram*-u, Sl. 4.5.1. Uočiti razlike u programskom kôdu u odnosu na primer iz prethodne glave *Read Dig Chan.vi*.

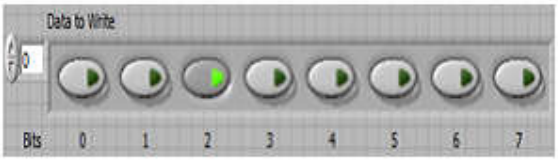


Sl. 4.5.1. *Block Diagram* aplikacije *Write Dig Chan.vi*

3. **Isključiti *NI DAQ* uređaj iz računara i povezati ga** prema šemi na Sl. 4.5.2. Nakon povezivanja ponovo priključiti *NI DAQ* uređaj na računar.

Sl. 4.5.2. Veza linije 2 digitalnog kanala 0 (P0.2) i LED diode

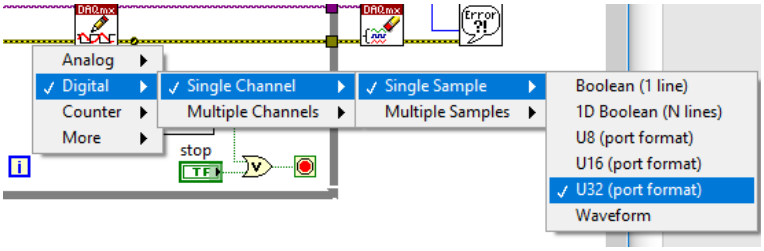
4. Kontrolu *Lines* podesiti na 8 digitalnih linija (0:7) porta 0 (port 0) za *NI DAQ* čiji je *device number DevN*: ***DevN/port0/line0:7***.
5. Pokrenuti izvršavanje aplikacije. Pomoću logičke kontrole ***Data to write***, Sl. 4.5.3, zadavati podatak koji će se upisati na digitalni port. Uočiti da je na liniju 2 (P0.2) digitalnog kanala 0 vezana LED dioda kao na Sl. 4.5.2. Digitalni izlazi P0.0..7 će biti +5 V ili 0 V u zavisnosti od toga da li je pomoću prekidačkih kontrola na *Front Panel*-u podešena vrednost *TRUE* (tj. "logička jedinica") ili *FALSE* (tj. "logička nula") za odgovarajuću digitalnu liniju P0.x (x=0,1,2,…7). Promeniti logički nivo digitalne linije 2 sa FALSE na TRUE - uočiti da se LED dioda na protobordu upalila.



logički nivo	0	0	1	0	0	0	0	0
naponski nivo	0 V	0 V	5 V	0 V	0 V	0 V	0 V	0 V
decimalni broj	4							

Sl. 4.5.3. Logička kontrola ***Data to write***

6. Modifikovati program tako se umesto niza logičkih vrednosti kontrole *Data to write* na digitalni port upisuje decimalni reprezent ovog niza:
   a. Obrisati niz logičkih promenljivih *Data to write*.
   b. Za polimorfnu funkciju *DAQmx Write* izabrati tip kao na Sl. 4.5.4.
   c. Pozicionirati se mišem na odgovarajući *data* ulaz funkcije *DAQmx Write*.
   d. Kliknutim desnim tasterom miša i izabrati opciju *Create Control*.
7. Pokrenuti izvršavanje programa. Uneti decimalni broj 4 u *data* kontrolu na *Front Panel*-u (binarni reprezent je 00000100) da bi se LED dioda upalila, tj. broj 0 (binarni reprezent je 00000000) da bi se ugasila.



Sl. 4.5.4. Izbor polimorfne funkcije DAQmx Write za upis decimalnog broja na port

8. Sačuvati program kao *Write Dig Chan_decimalni_broj.vi.*

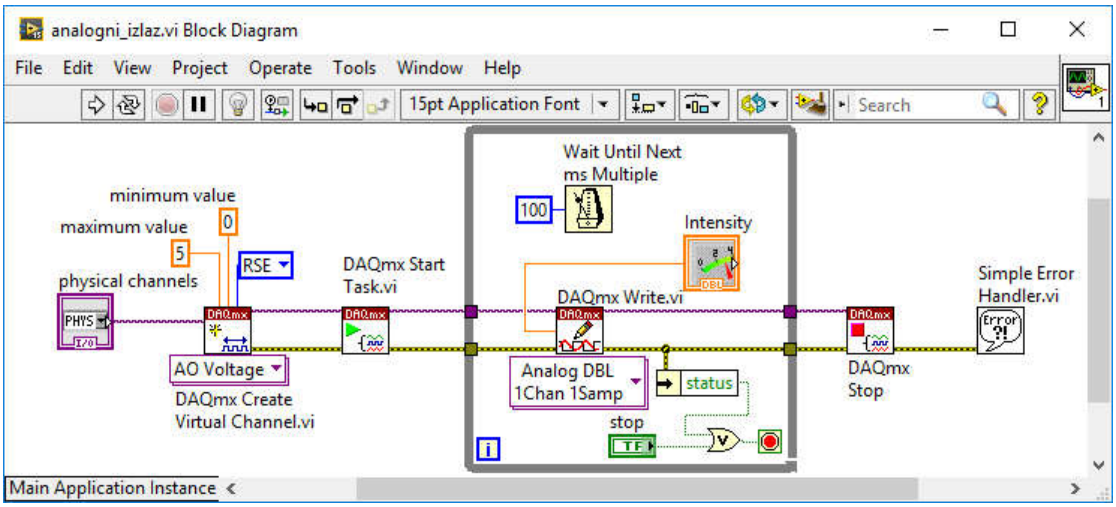
4.5.2 Korišćenje analognog izlaza

1. **Isključiti *NI DAQ* uređaj iz računara i povezati ga** prema šemi na Sl. 4.5.5. Nakon povezivanja ponovo priključiti *NI DAQ* uređaj na računar.

Sl. 4.5.5. Veza analognog izlaza *AO0* i LED diode

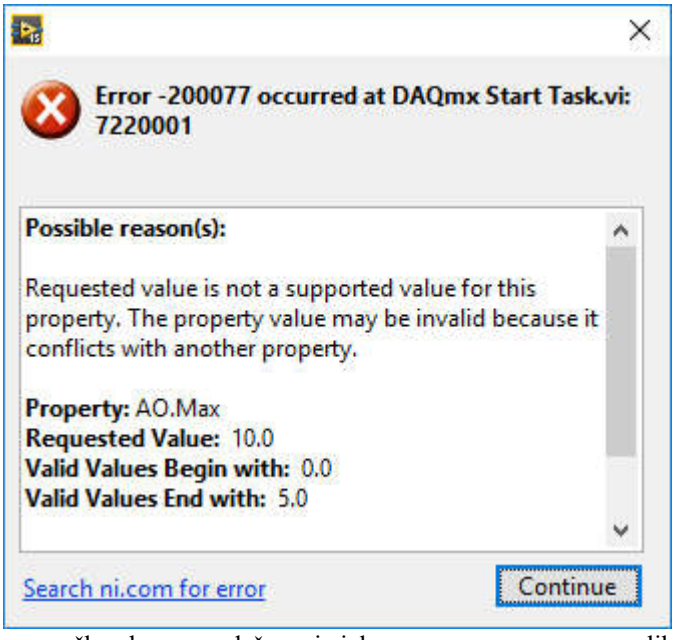
2. Otvoriti primer *voltmetar.vi*. Modifikovati ga tako da na analognom izlazu *ao0* može da generiše promenljivi napon koji će napajati LED diodu sa Sl. 4.5.5. U zavisnosti od nivoa napona, intenzitet svetljenja diode treba da se menja.
3. *Block Diagram* aplikacije za upravljanje intenzitetom svetljenja diode je prikazana na Sl. 4.5.6.
   NAPOMENA: NI USB 6008/6009 mogu na pinu *ao0* da generišu napon koji je u opsegu 0-5 V. Zbog toga je potrebno podesiti *minimum* i *maxmum* ulazne

promenljive funkcije *DAQmx Create Virtual Channel.vi* na 0 V i 5 V, Sl. 4.5.6. Ako se ne unese ovakvo podešavanje, ostaće podrazumevane vrednosti za *minimum (-10 V)* i *maxmum (10 V)* što će uzrokovati grešku kao na Sl. 4.5.7.



Sl. 4.5.6. *Block Diagram* aplikacije za upravljanje intenzitetom svetljenja diode

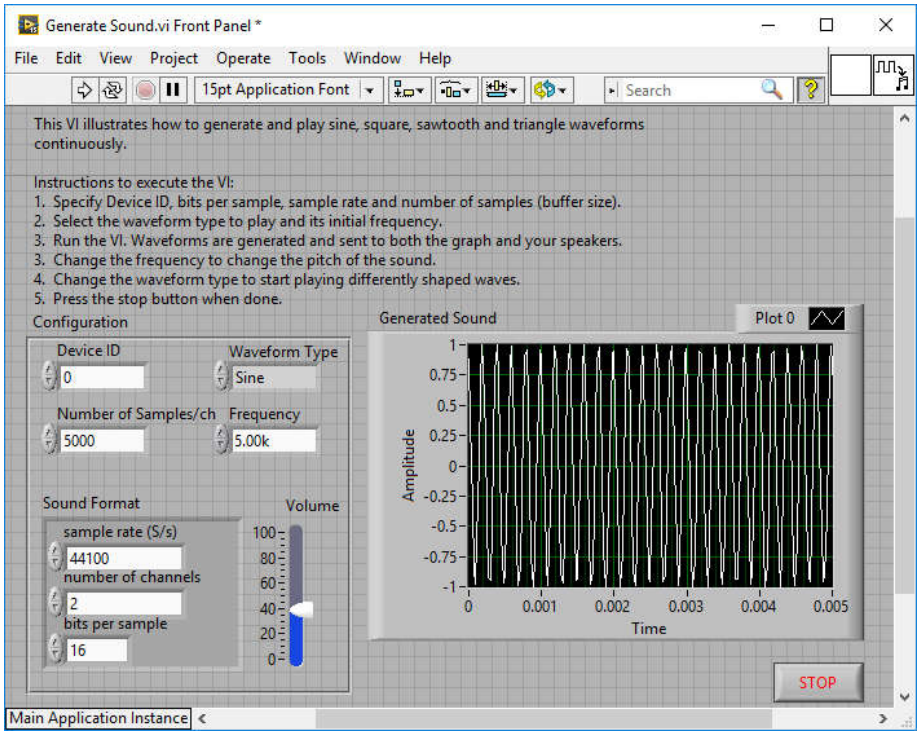
4. Kontrolu *physical channels* na *Front Panel*-u podesiti da generiše analogni izlaz *ao0* (uneti vrednost *DevN/ao0*, gde je *device number NI DAQ* uređaja).
   NAPOMENA: Nije moguće u padajućem meniju kontrole *physical channels* na *Front Panel*-u izabrati *ao0*, već ga treba ručno uneti sa tastature.
5. Pokrenuti izvršavanje programa. Promenom vrednosti kontrole *Intensity* menjati intenzitet svetljenja LED diode.
6. Sačuvati aplikaciju kao *analogni_izlaz.vi*.



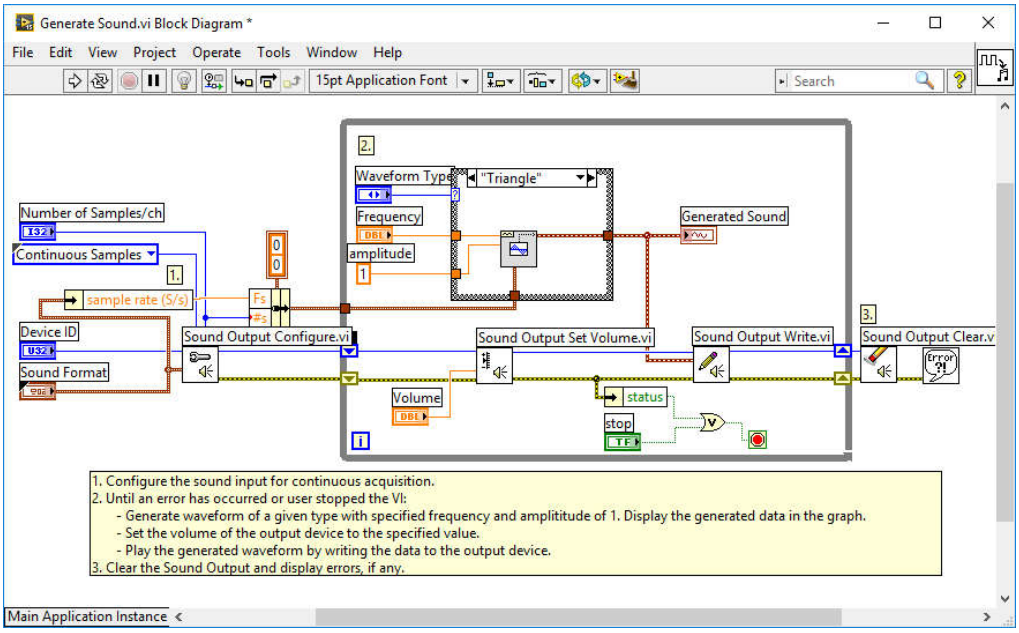
Sl. 4.5.7: Prozor za grešku zbog nepodešavanja izlaznog opsega napona u aplikaciji u skladu sa mogućnostima NI DAQ uređaja

4.6 Generisanje zvučnog signala

1. Otvoriti primer *Generate Sound.vi* čiji su *Front Panel* i *Block Diagram* prikazani na Sl. 4.6.1 i Sl. 4.6.2, respektivno. Proučiti objašnjenje rada programa u Block *Diagram*-u. Otvoriti i program *Cont Acq&Graph Voltage-Int Clk_DBL.vi* i uočiti sličnosti i razlike.
   Uočiti da je redni broj uredjaja, tj. zvučne kartice 0 po *default*-u (kontrola *Device ID* na *Front Panel*-u).
2. Pokrenuti program i „slušati“ različite frekvencije i oblike signala (menjati vrednosti kontrola *Frequency* i *Waveform Type*).



Sl. 4.6.1. *Front Panel* programa *Generate Sound.vi*

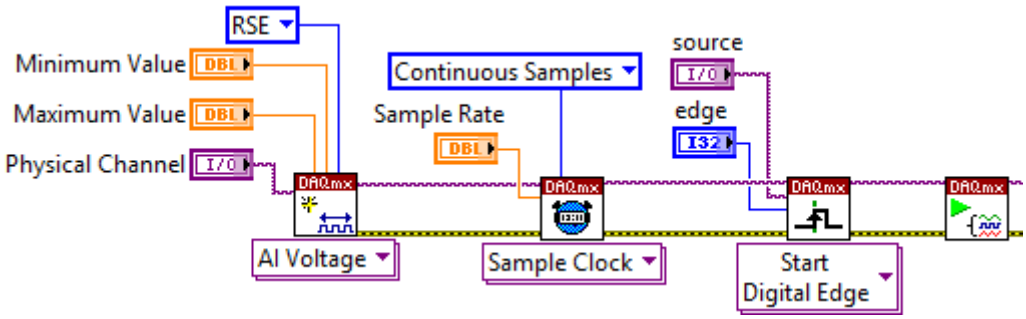


Sl. 4.6.2. *Block Diagram* programa *Generate Sound.vi*

4.7 Trigerovana kontinualna akvizicija

Trigerovana akvizicija je tip akvizicije u kome prikupljanje odbiraka u bafer započinje ispunjenjem odredjenog uslova (tzv. „trigera“): npr. kada nivo ili nagib analognog ulaznog signala dostigne zadatu vrednost, ili kada je detektovana uzlazna ili silazna ivica digitalnog ulaznog signala i sl. Pri ovom tipu akvizicije, moguće je zadati broj odbiraka koji će se prikupiti pre ili posle „triger“ signala, a moguće je i zadati kontinualnu akviziciju počev od pojave „triger“ signala. U ovom vežbanju će biti prikazana trigerovana kontinualna akvizicija u kojoj je „triger“ pojava silazne ivice digitalnog kanala PFI0.

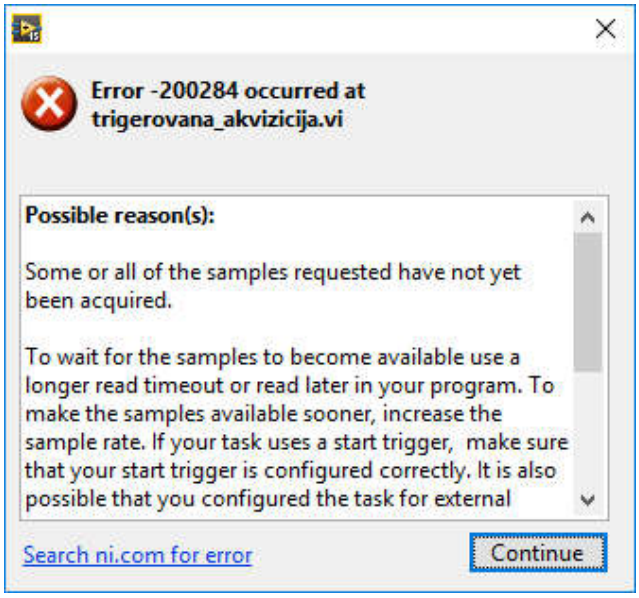
1. Priključiti *NI DAQ* uređaj preko USB-a na računar.
2. Kontrolu *Physical Channel* na *Front Panel*-u primera *Cont Acq&Graph Voltage-Int Clk_DBL.vi* podesiti da snima analogni ulaz *ai0* (selektovati vrednost *DevN/ai0*, gde je *device number NI DAQ* uređaja).
3. Pokrenuti primer *Cont Acq&Graph Voltage-Int Clk_DBL.vi.* Na grafiku će se pojaviti signal koji postoji u okolini pošto kanal *ai0* nije nigde povezan. Zaustaviti izvršavanje programa.
4. Uočiti u *Block Diagram*-u primera *Cont Acq&Graph Voltage-Int Clk_DBL.vi* da je polimorfna funkcija *DAQmx Read* podešena na *Analog 1D DBL 1Chan Nsamp,* tj. podešena je da pri svakoj iteraciji *While* petlje čita iz bafera *Nsamp* odbiraka sa jednog analognog kanala (u ovom konkretnom slučaju je *Samples to Read*=1000).
5. Žicom povezati PFI0 pin sa +5 V pinom.
6. Primer *Cont Acq&Graph Voltage-Int Clk_DBL.vi* dopuniti funkcijom *Measurement I/O»NI DAQmx»DAQmx Start Trigger* kao na Sl. 4.7.1. U padajućem meniju ove polimorfne funkcije izabrati *Start»Digital Edge*. Na odgovarajućim ulazima ove funkcije napraviti dve kontrole *source* i *edge*.



Sl. 4.7.1. Dopuna programa ***Cont Acq&Graph Voltage-Int Clk_DBL.vi*** za konfigurisanje trigerovane akvizicije

7. Podesiti vrednost kontrole *source* na *Front Panel*-u na /DevN/PFI0 (gde je N redni broj NI DAQ uređaja). Podesiti vrednost kontrole *edge* na *Front Panel*-u na „Falling“.
8. Pokrenuti izvršavanje modifikovanog programa. Uočiti da se na grafiku ne pojavljuju odbirci. Nakon 10 s će se pojaviti poruka o greški kao na Sl. 4.7.2. Program će završiti izvršavanje nakon pritiska na dugme *Continue*.

NAPOMENA: vreme čekanja „triger“ signala je podešeno na 10 s pomoću konstante *timeout* u *While* petlji. Menjati vreme čekanja „trigera“ i uočiti razliku.

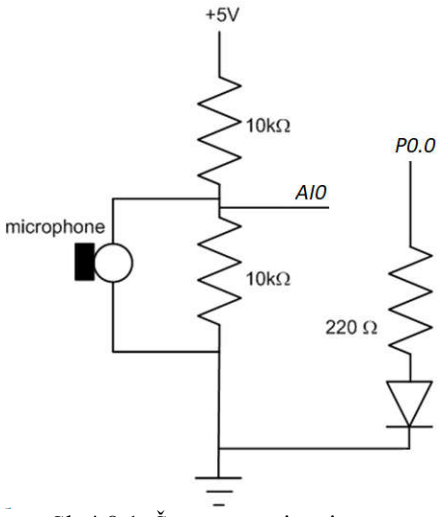


Sl. 4.7.2. Poruka o greški pri trigerovanog akviziciji kada „triger“ signal nije detektovan

9. Ponovo pokrenuti izvršavanje programa, ali sada žicu (koja je priključena s jedne strane na pin PFI0, a s druge strane na +5 V) prebaciti sa +5 V na GND (ovo se mora uraditi u okviru *timeout* vremena). Uočiti da će nakon ovog prelaska sa visokog na niski nivo napona, tj. nakon detekcije padajuće ivice digitalnog signala, početi kontinualna analogna akvizicija.
10. Sačuvati modifikovani primer kao *trigerovana_akvizicija.vi*.

4.8 Primer aplikacije s povratnom spregom

1. **Isključiti *NI DAQ* uređaj iz računara.** Povezati kolo kao na Sl. 4.8.1 na *NI DAQ* uređaj. Mikrofon povezati direktno u konektore *AI0* i *GND*, a ne preko protoborda (zbog širma koji ne može da se lako priključi na protobord). Za ostale komponente koristiti i protobord.

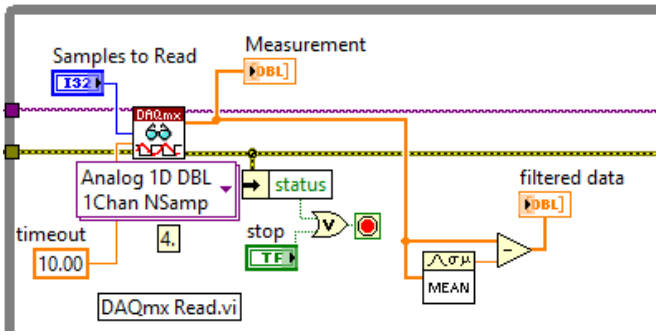


Sl. 4.8.1. Šema povezivanja

2. Žicom povezati PFI0 pin sa +5 V pinom. **Uključiti *NI DAQ* uređaj u računar.**
3. Pokrenuti aplikaciju *trigerovana_akvizicija.vi* (kreiranu u prethodnom poglavlju) i posmatrati izgled analognog signala *ai0* na grafiku kada se govori u mikrofon.

Uočiti promene u amplitudi i frekvenciji signala pri promeni jačine i frekvencije glasa. Uočiti da postoji jednosmerna komponenta signala.

4. Odkloniti jednosmernu komponentu analognog signala tako što će se od njega oduzeti njegova srednja vrednost. Srednju vrednost niza odbiraka pročitanih u jednoj iteraciji *While* petlje izračunati pomoću funkcije *Mathematics»Probability & Statistics»Mean*, Sl. 4.8.2.



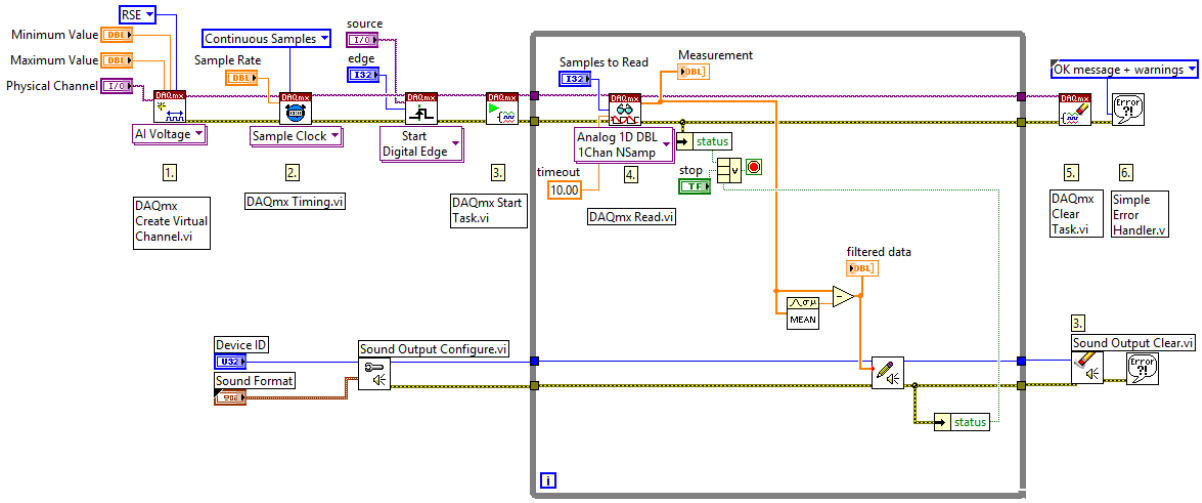
Sl. 4.8.2. Otklanjanje jednosmerne komponente

Zadatak 4.8.1.

Polazeći od programa *trigerovana_akvizicija.vi* (iz sekcije 4.7), dizajnirati aplikaciju koja omogućava:

a. čitanje analognog signala sa mikrofona *ai0* (mikrofon je priključen u kolo sa Sl. 4.8.1)
b. prikupljeni analogni signal *ai0* sa mikrofona nakon elementarne obrade (otklanjanje jednosmerne komponente) emituje na slušalicama
c. da se upali LED (sa Sl. 4.8.1) ako anvelopa analognog signala sa mikrofona *ai0* pređe zadatu vrednost praga.

1. Kombinovanjem programa *trigerovana_akvizicija.vi* i programa *Generate Sound.vi* napraviti aplikaciju koja omogućava da se pročitani analogni signal sa kanala *ai0* emituje na zvučniku, Sl. 4.8.3.

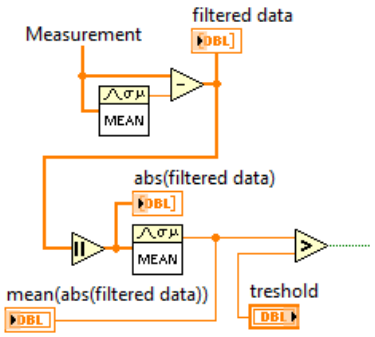


Sl. 4.8.3. Aplikacija koja omogućava trigerovanu kontinualnu akviziciju i emitovanje analognog signala na zvučnik

11. U *Sound Format* kontroli podesiti da frekvencija generisanja zvuka bude ista kao za akviziciju: 10 000 Hz. Pokrenuti izvršavanje aplikacije. Pričati u mikrofon, a slušalice će emitovati glas.

NAPOMENA: Pokušati da se na slušalice direktno, bez korekcije jednosmerne komponente, prosledi signal sa *Measurement* indikatora. Šta će se sada čuti na slušalicama i zašto?

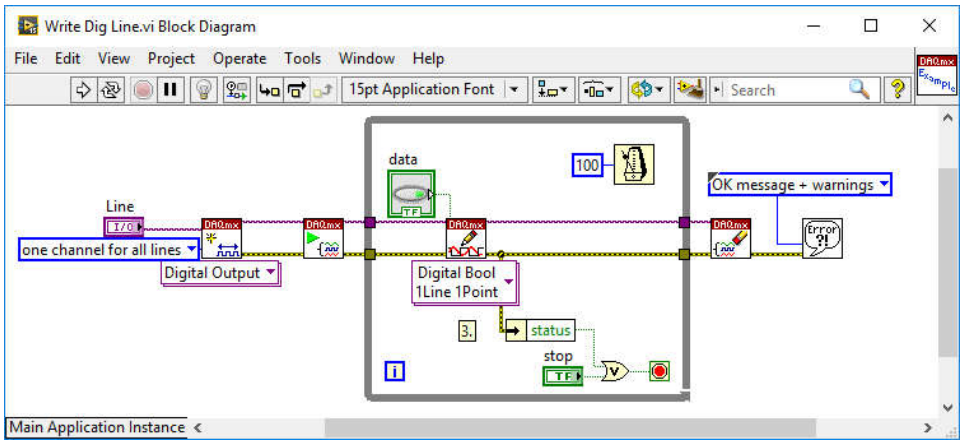
12. Proceniti anvelopu analognog signala sa mikrofona *ai0* na sledeći način, Sl. 4.8.4:
    a. korigovati jednosmernu komponentu
    b. ispraviti signal korišćenjem funkcije za određivanje absolutne vrednosti *Programming»Numeric»Absolute Value*
    c. odrediti srednju vrednost ispravljenog signala pomoću *Mathematics»Probability & Statistics»Mean*.
13. Uporediti procenjenu anvelopu signala za pragom koji će se zadavati pomoću kontrole *treshold*, Sl. 4.8.4, na *Front Panel*-u. Rezultat poređenja vrednosti anvelope sa pragom je logička promenljiva koja se može iskoristiti kao upravljački signal za paljenje/gašenje LED-a.



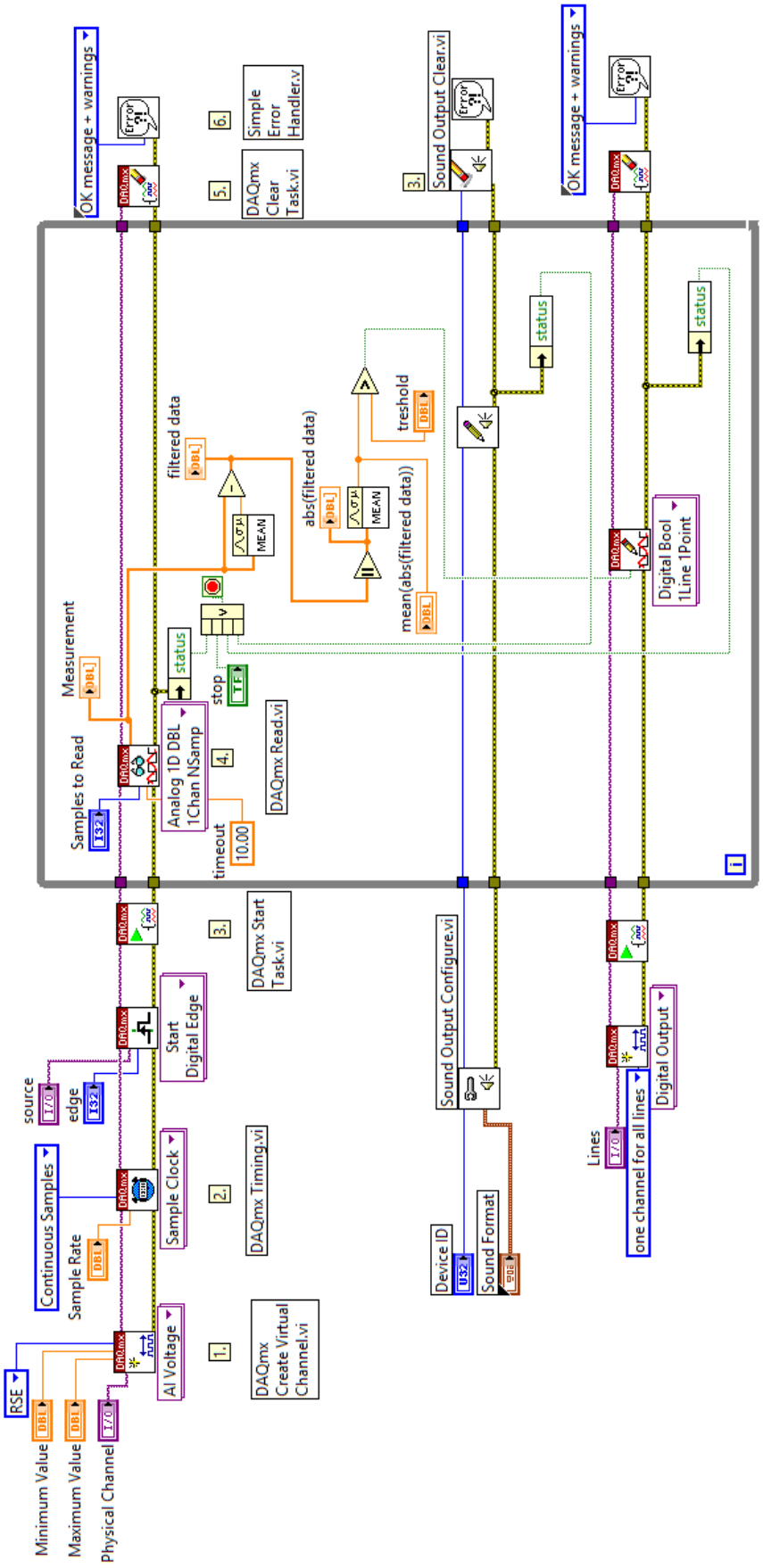
Sl. 4.8.4: Procena anvelope analognog signala sa mikrofona *ai0*

Pitanja: Da li bi mogao analogni signal sa mikrofona ai0 odmah nakon korekcije jednosmerne komponente da se upotrebi kao upravljački signal za paljenje/gašenje LED-a? (Pomoć: LED ne može da se beskonačno brzo pali/gasi) Zašto je neophodno ispraviti signal? (Pomoć: kolika je srednja vrednost neispravljenog signala kome je prethodno otklonjena jednosmerna komponenta?)

14. Otvoriti program *Write Dig Line.vi*, Sl. 4.8.5. Kombinovanjem programa sa Sl. 4.8.3, 4.8.4 i 4.8.5, napraviti aplikaciju koja pali/gasi LED u zavisnosti od nivoa signala na mikrofonu, tzv. *feedback* aplikaciju, Sl. 4.8.6.



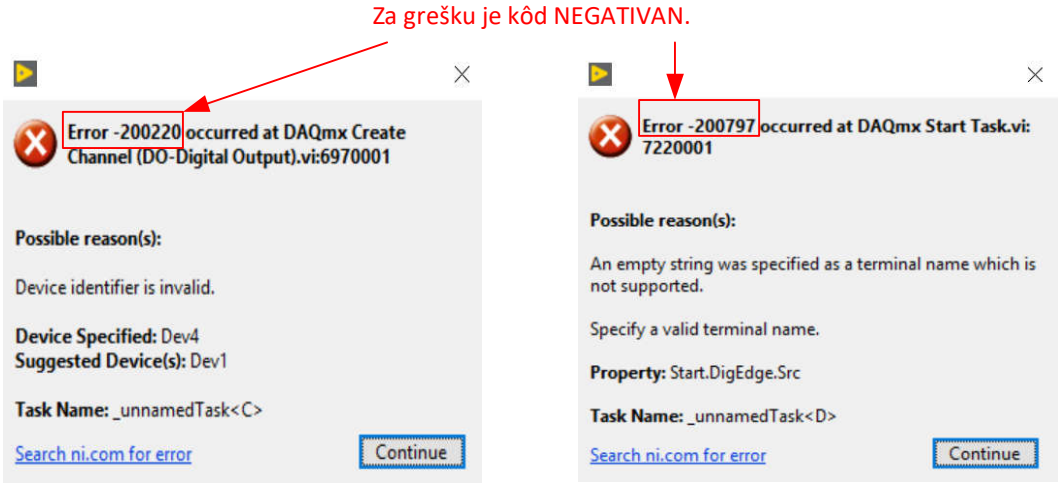
Sl. 4.8.5: *Block Diagram* programa *Write Dig Line.vi*



Sl. 4.8.6: Primer realizacije aplikacije sa povratnom spregom

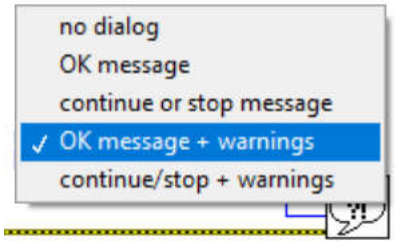
15. Sačuvati primer kao *primer_povratne_sprege.vi*.
16. Isključiti *NI DAQ* uređaj iz računara. Pokrenuti program. Koliko puta će se prozor sa greškama pojaviti i zašto? Koje opcije će biti ponuđene na prozoru: nastavak izvršavanja (*Continue*) ili/momentalno zaustavljanje (*Stop*)?

NAPOMENA: Na Sl. 4.8.7. su prikazani prozori sa greškama koji se pojavljuju u tački 16. Obratiti pažnju da su kôdovi greški (***Errors***) **negativni** brojevi. U slučaju samo upozorenja (***Warnings***) kôdovi su **pozitivni** brojevi. Logička vrednost **status** promenljive u *error* klasteru je za greške **TRUE**, a za upozorenja **FALSE**.



Sl. 4.8.7. Kodovi greški (***Errors***) su **negativni** brojevi

17. Za funkciju *Simple Error Handler*  na ulazu „*type of dialog*“ se može povezati jedna od sledećih vrednosti *enum* konstante (ili kontrole):
    a. *no dialog* – u slučaju greške ili upozorenja se neće prikazati prozor
    b. *OK message* – u slučaju greške će se prikazati prozor sa *Continue* dugmetom
    c. *continue or stop message* – u slučaju greške će se prikazati prozor sa *Continue* i *Stop* dugmetom
    d. *OK message + warnings* – u slučaju greške ili upozorenja će se prikazati prozor sa *Continue* dugmetom
    e. *continue/stop + warnings* – u slučaju greške ili upozorenja će se prikazati prozor sa *Continue* i *Stop* dugmetom

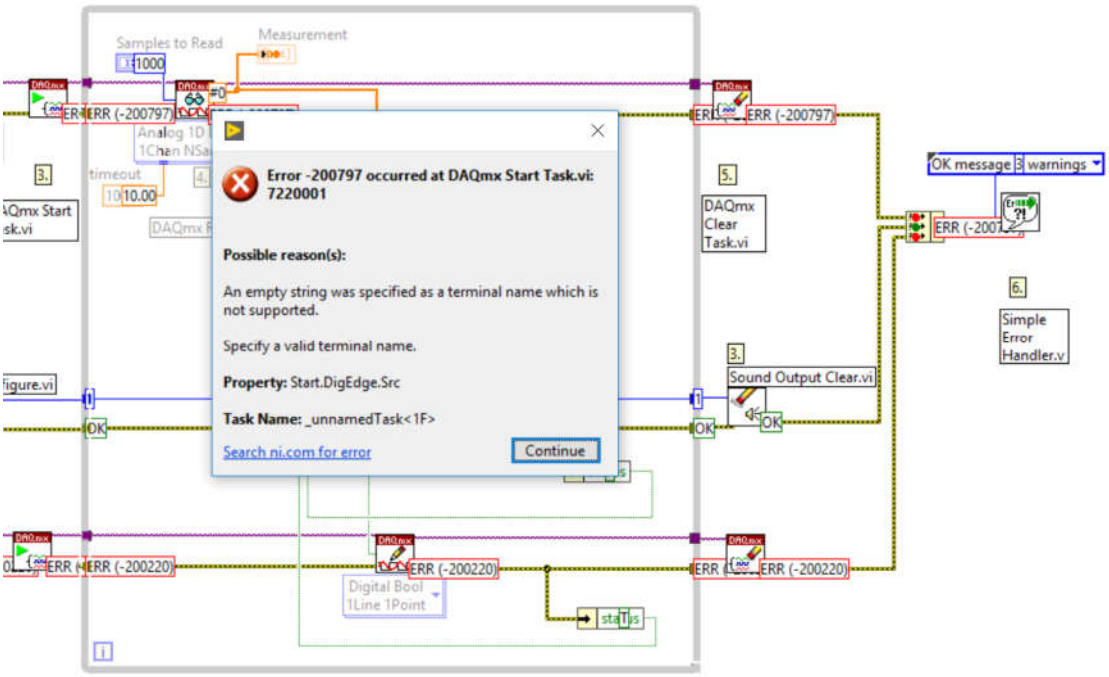


Sl. 4.8.8. Opcije ulaza „*type of dialog*“ funkcije *Simple Error Handler*

Isprobati sve navedene opcije *Simple Error Handler* funkcije kada je *NI DAQ* uređaj isključen iz računara. Pritiskom na dugme *Continue* u prozoru greške, izvršavanje programa se nastavlja, a pritiskom na dugme *Stop* u prozoru greške, izvršavanje programa se odmah zaustavlja.

18. Funkcija *Merge Errors*  iz palete ***Functions»Programming»Dialog & User Interfaces*** omogućava da se objedine rezultati svih *error* klastera u programu u jedan izlazni klaster koji se potom može dovesti na odgovarajući ulaz *Simple Error Handler* funkcije. Modifikovati primer *primer_povratne_sprege.vi*. tako da se svi error klasteri objedine pomoću funkcije *Merge Errors*. Pomoću opcije *Highlight Execution*  posmatrati izvršavanje kôda kada je *NI DAQ* uređaj isključen iz računara, Sl. 4.8.9. Uočiti da ne postoji greška ili upozorenje zbog generisanja zvuka (zeleni ulaz u *Merge Errors* funkciju u môdu za debagovanje) već da postoje samo greške usled neprikljućenosti *NI DAQ* uređaja (dva crvena ulaza u *Merge Errors* funkciju u môdu za debagovanje). Koji je kôd greške na izlazu *Merge Errors* funkcije i zašto?

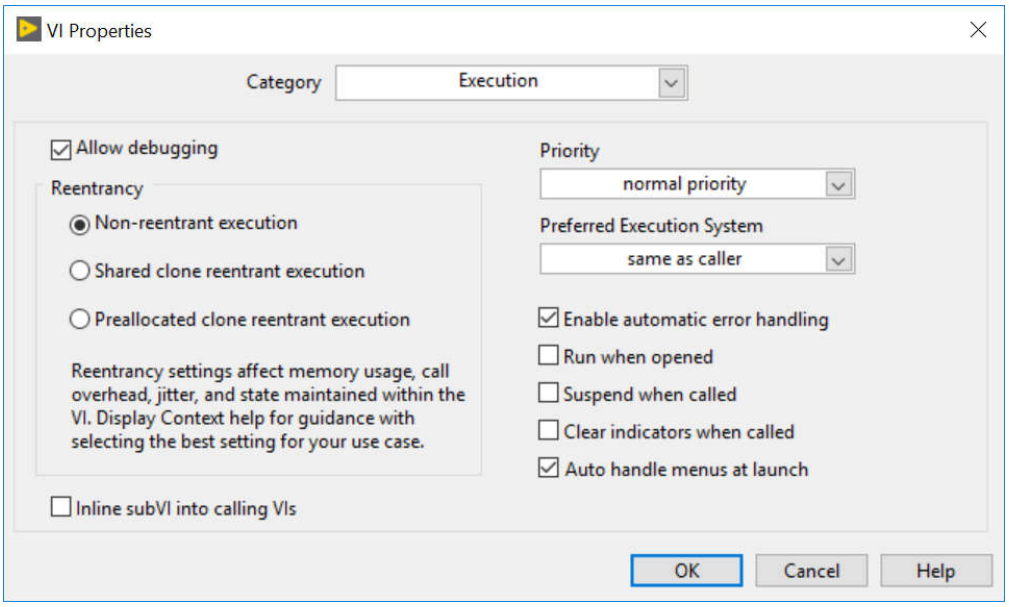
NAPOMENA: Ako više *error* klastera „predaje grešku" *Merge Errors* funkciji, ona će na svom izlazu dati kôd prve greške (prema redosledu povezivanja ulaznih priključaka), tj. NEĆE SPAJATI greške. U slučaju da nema „predaje grešaka", *Merge Errors* funkcija će na svom izlazu imati kôd prvog upozorenja (prema redosledu povezivanja ulaznih priključaka).



Sl. 4.8.9. Debagovanje kôda kada je *NI DAQ* uređaj isključen iz računara, a primenjena funkcija za objedinjavanje grešaka i/ili upozorenja

19. Sačuvati primer kao *primer_povratna_sprega_spajanje_gresaka.vi*.
20. U tačkama 16-19 je demonstriran manuelni način za prikaz greški/upozorenja (*Manual Error Handling*). Izbrisati sada funkciju *Simple Error Handler* iz programskog kôda primera *primer_ povratna_sprega_spajanje_gresaka.vi* i u opciji ***File»VI Properties»Execution*** aktivirati opciju *Enable automatic error handling*, Sl. 4.8.10. Ovim je podešeno automatsko prikazivanje greški/upozorenja (*Automatic Error Handling*) u tekućem programu. Takođe, proveriti i da li je u opciji ***Tools»Options»Block Diagram»Error Handling*** selektovana opcija koja omogućava automatsku podršku.

21. Pokrenuti izvršavanje programa. Uočiti da iako ne postoji u kôdu funkcija *Simple Error Handler*, program će prikazati prozor za grešku i blinkanjem će ukazati na čvor u program u kome je greška (u ovom primeru će to biti čvor *Merge Errors*).



Sl. 4.8.10. Podešavanje za automatsko prikazivanje grešaka/upozorenja

NAPOMENA: Ako program ne koristi neku od funkcija za prikaz greški/upozorenja (*Error Handler* funkcije) moguće je podesiti da okruženje automatski prikazuje prozor sa greškom/upozorenjem.

Lekcija 5 – Modularnost

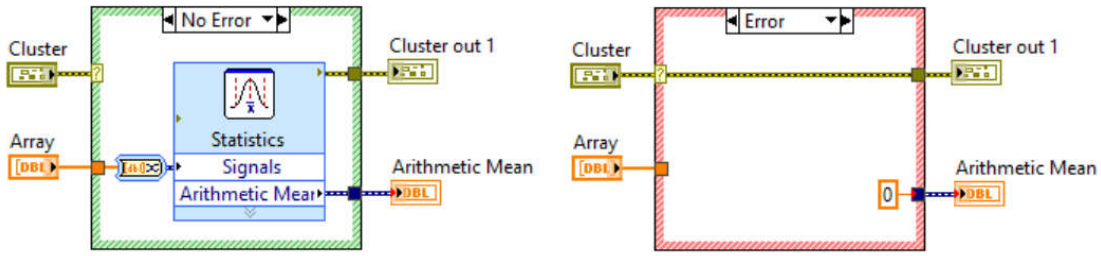
Cilj

Cilj vežbe je da studente upozna sa:

- načinima kreiranja potprograma
- pisanjem dokumentacije.

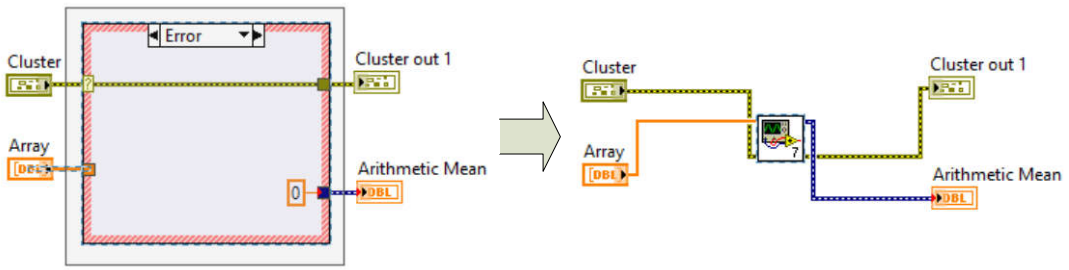
Zadatak 5.1.
Kreirati potprogram koji u slučaju greške u glavnom programu „propagira“ grešku, a u slučaju da nema greške u glavnom programu određuje srednju vrednost ulaznog niza. Glavni program se završava pritiskom na dugme STOP.

1. Kreirati novi .vi program. Iz programa *Error_klaster.vi* (Zadatak 3.5.2.) iskopirati *error* klaster kontrolu i *error* klaster indikator u novi program.
2. Kreirati *Block Diagram* kao na Sl. 3.5.1.



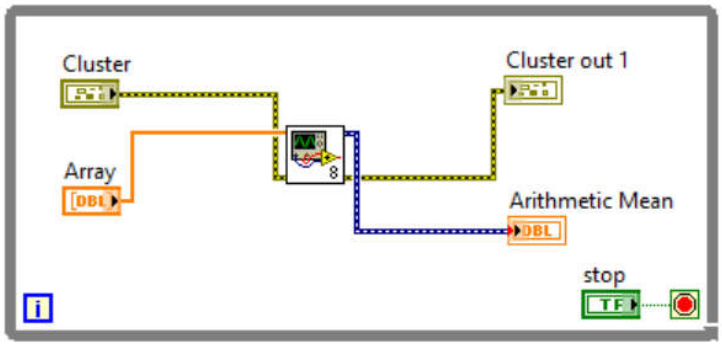
Sl. 3.5.1.

3. Pomoću *Position/Size/Select* alatke selektovati region oko *Case* strukture, Sl. 3.5.2. Izabrati opciju ***Edit»Create SubVI***. Od selektovanog dela programa će se formirati potprogram .



Sl. 3.5.2. Kreiranje potprograma (SubVI) „prečicom“

4. Dopuniti programski kôd *While* petljom kao na Sl. 3.5.3.



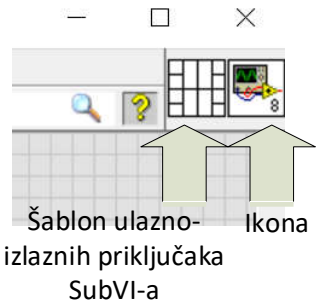
Sl. 3.5.3. Glavni program sa jednim potprogramom

5. Pokrenuti izvršavanje programa. Menjati vrednosti elemenata niza *Array* u glavnom programu i posmatrati kako se menja vrednost na indikatoru *Arithmetic Mean*. Promeniti vrednost *status* promenljive *error* klaster kontrole u glavnom programu na TRUE. Šta se sada dobija na indikatoru *Arithmetic Mean* i zašto? Zaustaviti izvršavanje glavnog programa pritiskom na dugme STOP.
6. Sačuvati glavni program kao *glavni_program_prečica.vi*. Okruženje će tražiti i čuvanje „prečicom" napravljenog potprograma – sačuvati ga kao *srednja_vrednost_prečica.vi.*
7. Zatvoriti programe *srednja_vrednost_prečica.vi* i *glavni_program_precica.vi*.

NAPOMENA: Potprogram u *LabVIEW* okruženju se naziva SubVI.

Zadatak 5.2.
Ponoviti Zadatak 5.1. bez korišćenja „prečice" u kreiranju SubVI-a. Dokumentovati rad SubVI-a. Omogućiti da se pri prelasku miša iznad STOP dugmeta pojavi natpis „Zaustavi aplikaciju" dok traje izvršavanje programa.

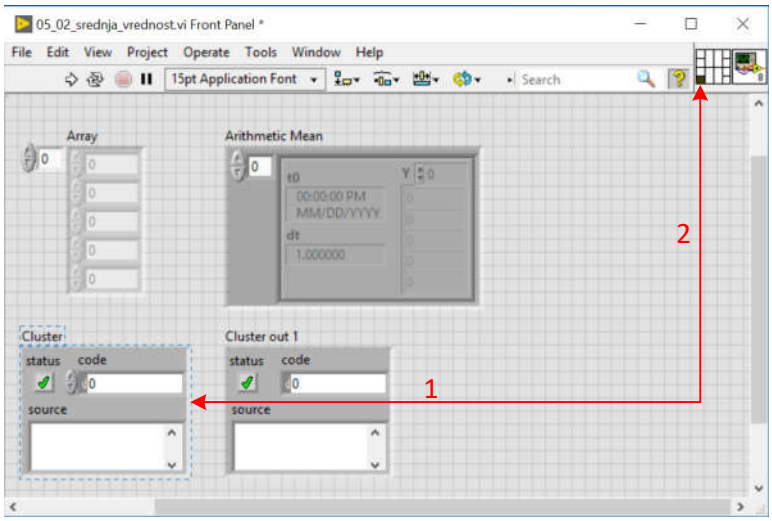
1. Kreirati novi .vi program. Kreirati *Block Diagram* kao na Sl. 3.5.1. Sačuvati program kao *srednja_vrednost.vi.*
2. Na Sl. 3.5.4. je prikazan šablon (*Pattern*) ulazno-izlaznih priključaka potprograma i ikona potprograma (ikona je slika potprograma koja će biti vidljiva u glavnom programu).
3. Kliknuti desnim tasterom miša na šablon potprograma i u opciji *Patterns* izabrati najčešće korišćen šablon  .



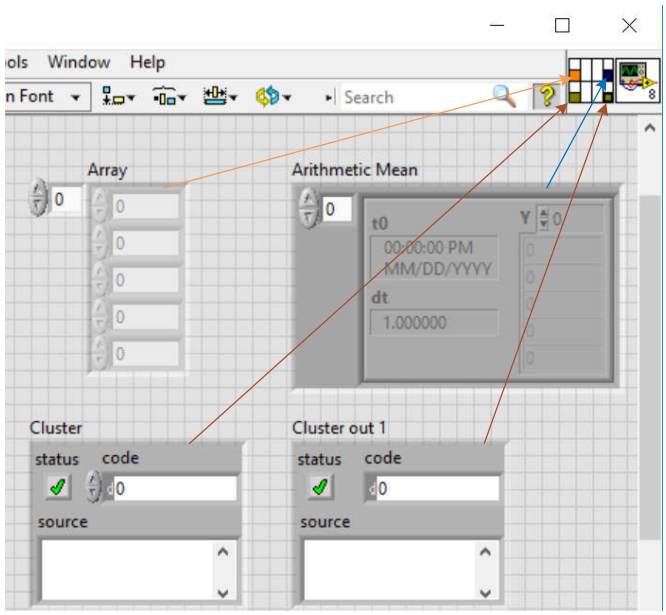
Sl. 3.5.5. Šablon i ikona SubVI-a

4. Selektovati *Wire Tool*  i kliknuti na *Front Panel*-u najpre na *Cluster* kontrolu (1), a potom na donje levo polje šablona (2), Sl. 3.5.5. Proceduru ponoviti za

svaku od kontrola i indikatora koje će biti ulazne ili izlazne promenljive potprograma. Konačni izgled šablona je prikazan na Sl. 3.5.6.



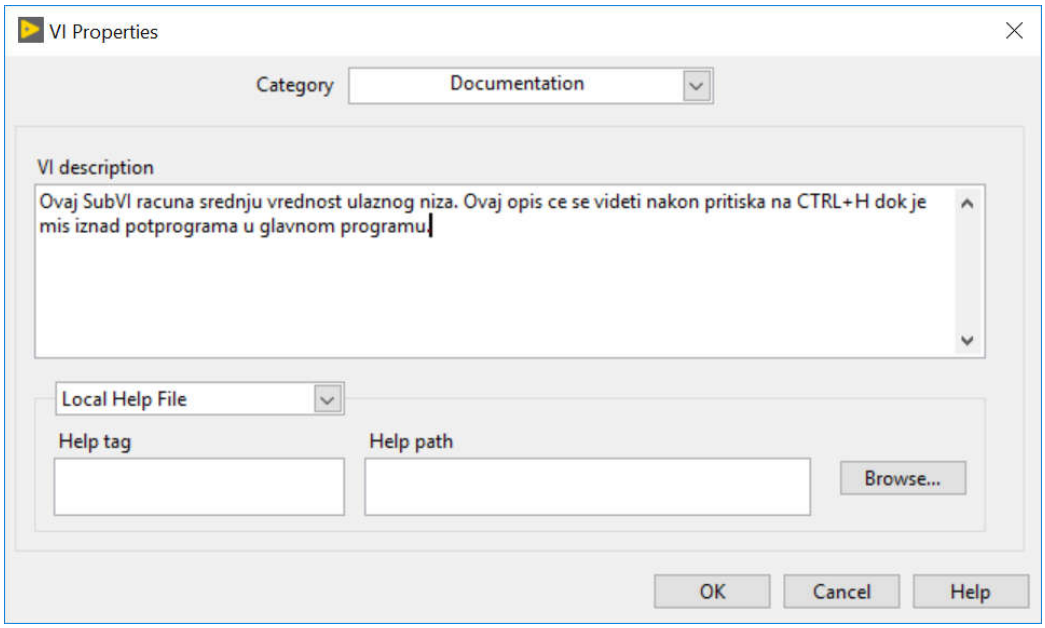
Sl. 3.5.5. Dodeljivanje ulaznog priključka šablona promenljivoj



Sl. 3.5.6.

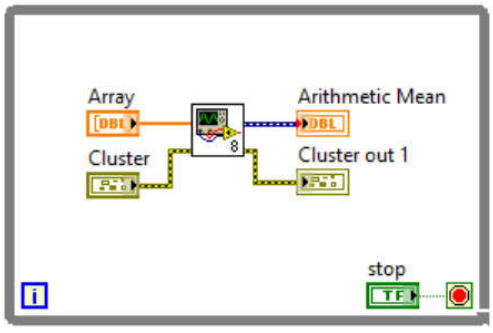
5. Potprogram *srednja_vrednost.vi* može da obavlja svoju funkciju samo ako mu je dostavljen niz za koji treba da računa srednju vrednost. Zbog toga je taj priključak „zahtevani“ priključak (*required*) potprograma. Svi ostali priključci nisu neophodni ali mogu biti preporučeni (*recommended*). Kliknuti desnim tasterom miša na šablon na mestu priključka *Array* kontrole i izabrati opciju ***This Connection Is»Required***. Za ostale priključke izabrati opciju ***This Connection Is»Recommended***.
6. Otvoriti opciju ***File»VI Properties»Documentation*** i upisati opis rada potprograma, Sl. 3.5.7.
7. Da bi se dokumentovao rad programskom kôda unutar *Case* strukture, kliknuti desnim tasterom miša na *Case* strukturu i izabrati opciju ***Visible Items»Subdiagram Label.*** U oznaku uneti opis rada *Case* strukture.

NAPOMENA: Bilo koja od žica u programu može da „nosi“ i komentar (*owned label*). Postavljanje komentara „na žici“ se postiže selekcijom žice, klikom desnog tastera miša i izborom opcije ***Visible Items»Label***. Slobodni komentari (*free label*) se mogu pisati bilo gde u programskom kôdu pomoću *Labeling Tool*-a  .



Sl. 3.5.7. Opis rada potprograma – dokumentovanje VI-a

8. Uneti bar jedan slobodan komentar i bar jedan „komentar na žici“ u potprogram.
9. Sačuvati izmene u SubVI *srednja_vrednost.vi* i zatvoriti ga.
10. Kreirati novi .vi program. Kreirati *Block Diagram* kao na Sl. 3.5.8. pri čemu je potprogram postavljen tako što je u paleti ***Functions»Select a VI*** izabrana datoteka *srednja_vrednost.vi.* Odgovarajuće kontrole/indikatori u glavnom program se najjednostavnije kreiraju postavljanjem *Wire Tool*-a  iznad svakog od ulaza/izlaza potprograma, a nakon klika desnog tastera miša se selektuju opcije *Create Control/Indicator*.



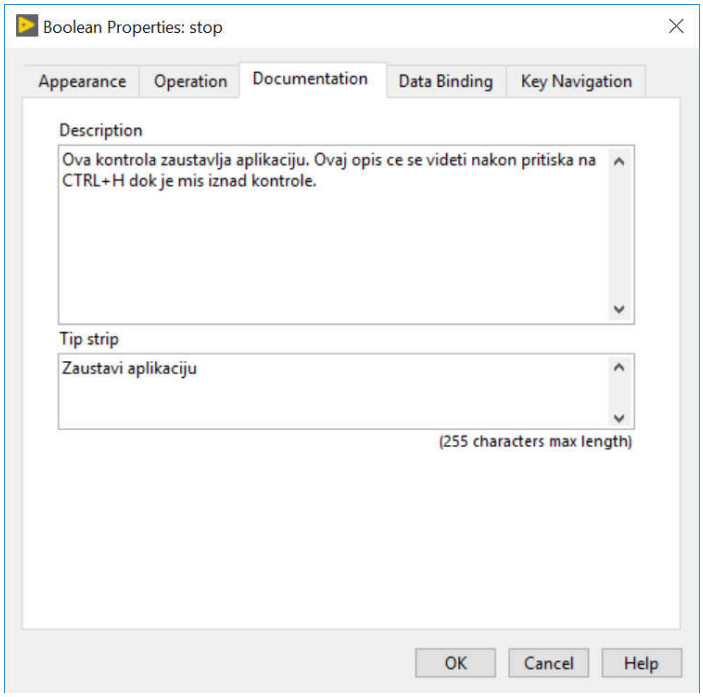
Sl. 3.5.8. Glavni program sa jednim potprogramom

11. Uočiti da kada se miš pozicionira iznad potprograma i pritisne CTRL+H, pojavljuje se pomoćni prozor sa opisom unetim u dokumentaciji (tačka 6).
12. Pokrenuti izvršavanje glavnog programa i proveriti njegovu funkcionalnost.
13. Zaustaviti izvršavanje programa i sačuvati ga kao *glavni_program.vi*.

14. Pokušajte da u glavnom programu obrišete vezu između *Cluster* kontrole i odgovarajućeg ulaznog priključka potprograma. Zašto se nije pojavila greška u programu? (Pomoć: *recommended* ulaz.) Pokušajte da obrišete vezu između *Array* kontrole i odgovarajućeg ulaznog priključka potprograma. Zašto se pojavila greška u programu ? (Pomoć: *recquired* ulaz.) Vratiti sve izbrisane veze.

NAPOMENA: Potprogrami u glavnom programu se prikazuju preko svoje ikone. Loša praksa je da svi potprogrami u glavnom programu imaju iste ili slične ikone jer se u tom slučaju teško razlikuju međusobno. Ikona svakog od potprograma treba da simbolizuje šta je zadatak potprograma.

15. Dva puta kliknuti levim tasterom miša na ikonu potprograma u *While* petlji glavnog programa. Otvoriće se potprogram *srednja_vrednost.vi.*
16. Dva puta kliknuti levim tasterom miša na ikonu . Otvoriće se prozor za editovanje ikone. U skladu sa zadatkom potprograma, editovati ikonu. Nakon čuvanja izmene potprograma, u glavnom programu će se pojaviti nova ikona.
17. Sačuvati izmene u glavnom programu.
18. Kliknuti desnim tasterom miša na STOP dugme. Izabrati opciju ***Properties»Documentation***. Uneti opise kao na Sl. 3.5.9. i prihvatiti ih klikom na dugme OK.



Sl. 3.5.9. Dokumentovanje kontrola/indikatora

Description polje omogućava da se opis funkcije kontrole (ili indikatora) pojavi kada se miš nađe iznad kontrole (ili indikatora) i pritisne se dugme CTRL+H. Natpis u *Tip strip* polju se pojavljuje u toku izvršavanja programa, pri prelasku miša iznad kontrole/indikatora. Testirati navedena navedena svojstva dokumentacije integrisane u VI.

19. Sačuvati izmene u glavnom programu.

Lekcija 6 – Prikaz podataka na graficima. Upis u datoteku

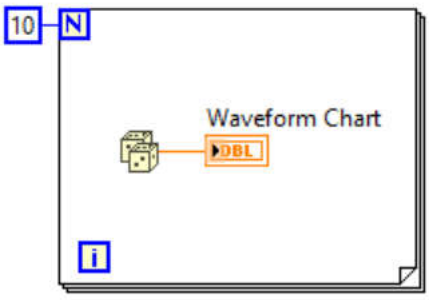
Cilj

Cilj vežbe je da studente upozna sa:

- načinima prikaza podataka na različitim grafičkim indikatorima
- načinima kreiranja različitih tipova datoteka (tekstualni, TDMS, binarni)
- programskim kreiranjem putanje datoteka
- upisom podataka u datoteku tokom akvizicije (*data streaming*).

Zadatak 6.1.
Kreirati program koji na grafičkom indikatoru prikazuje 10 slučajno generisanih brojeva.

1. Kreirati novi .vi program.
2. Kreirati *Block Diagram* kao na Sl. 3.6.1. Za grafički indikator izabrati *Waveform Chart* (***Controls»Modern»Graph***).



Sl. 3.6.1.

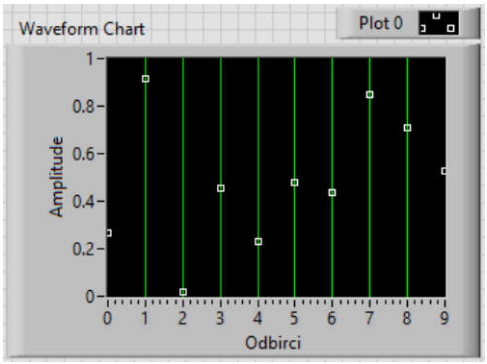
3. Pokrenuti program u *Highlight Execution* môd-u  . Primetiti da je na *Waveform Chart*-u maksimalan broj odbiraka koji će se prikazati na *x*-osi 100. Pokrenuti još jednom izvršavanje programa u normalnom môd-u (bez uključenog *Highlight Execution-a*). Primetiti da će na *Waveform Chart*-u biti dodato još deset slučajnih brojeva.
4. Na *Front Panel*-u kliknuti desnim tasterom miša na 

Waveform Chart-a. Izabrati opciju *Common Plots*, pa prikaz samo tačaka  . Pokrenuti izvršavanje programa i uočiti da sada tačke na grafiku neće više biti međusobno spojene linijama.
5. Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***X Scale»AutoScaleX***. Pokrenuti izvršavanje programa. Koliki će sada biti

maksimum na *x*-osi? Pokrenuti ponovo program i uočiti kako se maksimum *x*-ose automatski povećava.

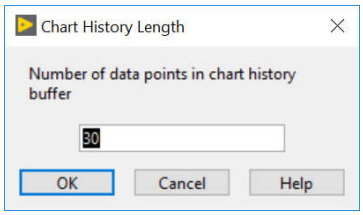
6. Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***X Scale»Style.*** Otvoriće se paleta sa izborom opcija za izgled podele i natpisa *x*-ose. Izabrati prikaz 

. Pokrenuti izvršavanje programa i uočiti kako će se podela i natpisi na *x*-osi promeniti.
7. Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***Data Operation»Clear Chart***. Uočiti da će se istorija grafika obrisati. Pokrenuti izvršavanje programa. Na koji način bi bilo moguće programski obrisati istoriju grafika? (podsetnik: *Lekcija 3.4, Programsko kontrolisanje osobine kontrola/indikatora*).
8. Uočiti da oznaka *x*-ose ima tekst „*Time*“. Da li x-osa zaista prikazuje vreme? Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju *Properties*. Proučiti šta sve od osobina *Waveform Chart*-a je moguće izmeniti. Potom podesiti osobine tako da se prikaže mreža (*grid*) za *x*-osu i da oznaka *x*-ose bude „*Odbirci*“ umesto „*Time*“, Sl. 3.6.2.



Sl. 3.6.2.

9. Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***Data Operation»Chart History Length...*** Podesiti da istorija grafika bude 30 odbiraka, Sl. 3.6.3. Pokrenuti izvršavanje programa 5 puta i uočiti kako se samo poslednjih 30 odbiraka prikazuje na grafiku pri čemu se *x*-osa menja u skladu sa rednim brojevima odbiraka.

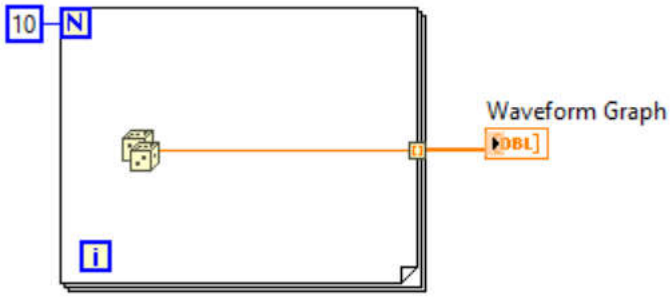


Sl. 3.6.3.

10. Sačuvati program kao *graficki_indikator_odbirak.vi*.
11. Zameniti *Waveform Chart* drugim tipom grafičkog indikatora: *Waveform Graph*-om (na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***Replace» Modern»Graph»Waveform Graph***). Uočiti da će se pojaviti greška u kôdu.

NAPOMENA: Na *Waveform Graph*-u se može prikazati samo niz odbiraka, a nije moguće prikazati pojedinačne odbirke kao što je to moguće na *Waveform Chart*-u.

12. Modifikovati kôd prema Sl. 3.6.4.



Sl. 3.6.4.

13. Pokrenuti izvršavanje programa nekoliko puta. Uočiti da se na *Waveform Graph*-u uvek prikazuje samo 10 odbiraka.
14. Na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Graph*. Uočiti da ne postoji opcija podešavanja istorije. Uočiti da postoji opcija ***Visible Items»Cursor Legend***. Uključiti ovu opciju i kreirati jedan kursor (klik desnog tastera miša na ***Cursor Legend***, opcija ***Create Cursor»Free***). Pokrenuti izvršavanje programa. Pomerati mišem kursor pomoću *Selection Tool*-a  i uočiti kako se u legendi kursora menjaju vrednosti pozicije kursora.
15. Sačuvati program kao *graficki_indikator_niz.vi*.
16. Zameniti *Waveform Graph* pomoću *Waveform Chart-a* (na *Front Panel*-u kliknuti desnim tasterom miša na *Waveform Chart* i izabrati opciju ***Replace» Modern»Graph»Waveform Chart***). Pokrenuti izvršavanje programa.
17. Zatvoriti program bez čuvanja izmena iz tačke 16.

NAPOMENA: Na *Waveform Graph*-u nije moguće pamtiti istoriju odbiraka za razliku od *Waveform Chart*-a, ali je moguće kreiranje kursora. Na *Waveform Chart*-u je moguće prikazivati pojedinačne odbirke ali i niz podataka.

Zadatak 6.2.

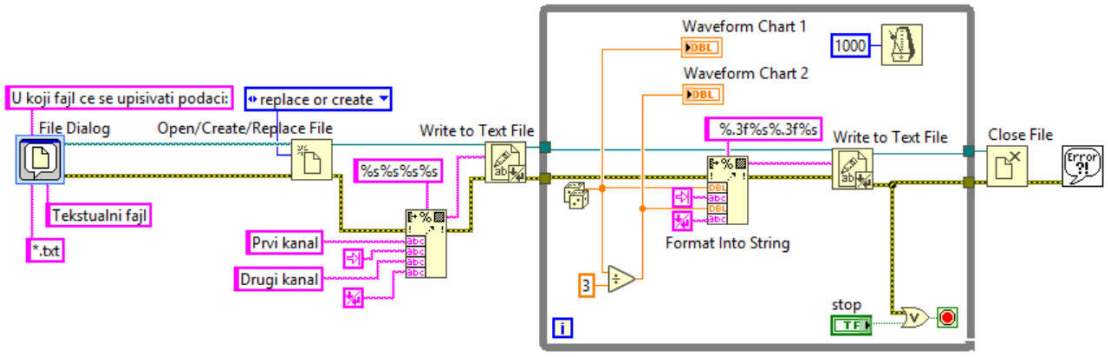
U ***Help»Find Examples*** pronaći (pomoću *Search* opcije) i proučiti programe koji koriste različite tipove grafičkih indikatora:

- *Waveform Chart Data Types and Update Modes.vi* (môd osvežavanja *Waveform Chart*-a se menja klikom desnog tastera miša na Front Panel-u i izborom opcije ***Advanced»Update Mode***)
- *XY Graph Data Types.vi*
- *Mixed Signal Graph.vi*
- *Intensity Graph and Chart Data Types.vi.*

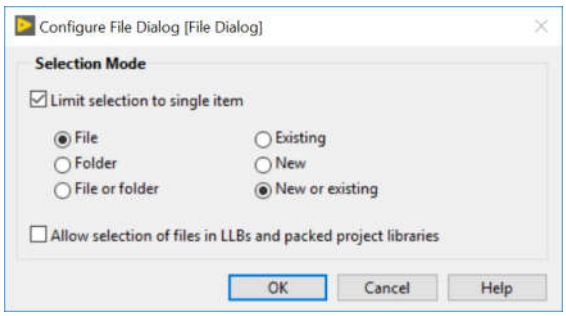
Zadatak 6.3.

Kreirati program koji kontinualno svake sekunde generiše dva broja (prvi broj je slučajan broj, a drugi broj je jednak trećini prvog broja) i prikazuje ih pojedinačno na dva grafička indikatora. Omogućiti kontinualni upis odbiraka sa tri decimale u **tekstualnu** datoteku (**ASCII** datoteku), pri čemu će prva kolona u datoteci odgovarati prvom broju, a druga kolona u datoteci odgovarati drugom broju. Omogućiti da korisnik izabere datoteka za upis na početku izvršavanja programa. U datoteku najpre upisati zaglavlje datoteke koje sadrži natpise „Prvi kanal“ i „Drugi kanal“ razdvojene *TAB*-om, a nakon toga nastaviti kontinualni upis brojeva u datoteku. Program se završava pritiskom na dugme STOP.

1. Kreirati novi .vi program.
2. Kreirati *Block Diagram* kao na Sl. 3.6.5. Selekcija imena datoteke je omogućena funkcijom *File Dialog* (***Programming»File I/O»Advanced File Functions***). Pri unosu ove funkcije u *Block Diagram* se podešavaju opcije *File* i *New or Existing*, Sl. 3.6.6. Ova *Express* funkcija se može prikazati u obliku ikone (kao što je i prikazana na Sl. 3.6.5) tako što se na klikne na funkciju desnim tasterom miša i odabere opcija *View As Icon*. Odgovarajuće ulaze funkcije *File Dialog* podesiti tako da se pri pokretanju programa korisniku prikaže prozor sa mogućnošću izbora tekstualnih datoteka i natpisom „*U koju datoteku ce se upisivati podaci:*“.



Sl. 3.6.5. Upis u tekstualnu datoteku



Sl. 3.6.6. Prozor *File Dialog* funkcije

Funkcije *Open/Create/Replace File, Write to Text File, Close File* se nalaze u paleti ***Programming»File I/O*** i omogućavaju otvaranje/kreiranje/zamenu datoteke, upis u tekstualnu datoteku i zatvaranje datoteke, respektivno. Ove funkcije predstavljaju *Low Level* funkcije za rad sa datotekama, tj. one ne sadrže druge funkcije (kada se dva puta klikne mišem na njih u *Block Diagram*-u neće se pojaviti njihov *Front Panel*). Uočiti da su ove funkcije međusobno spojene *error* linijama i *reference* linijama (*reference* linija nosi informaciju o referentnom broju datoteke koja je otvorena). Uočiti i da se zaglavlje datoteke upisuje samo jednom, pre ulaska u *While* petlju, a da se tokom izvršavanja *While* petlje upisuju samo podaci.

NAPOMENA: Otvaranje i zatvaranje datoteke je optimalno uraditi samo jednom u programu. Otvaranje datoteke treba uraditi u fazi inicijalizacije (pre *While* petlje), a zatvaranje datoteke u fazi delokacije memorijskog prostora (posle *While* petlje).

Funkcija *Format Into String* (***Programming»String***) formira nizovni podatak za upis u datoteku. Uočiti da ova funkcija ima ulaz *Format String* na koji se povezuje

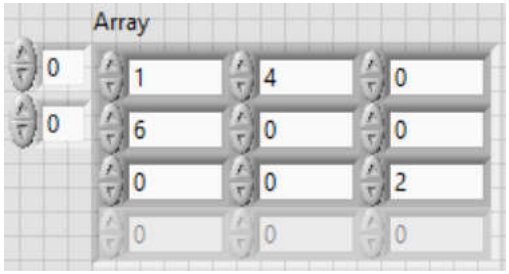
string koji objašnjava strukturu podataka. Na primer ako se na ulaz *Format String* dovede "%.3f%s%.3f%s", to znači da će izlaz funkcije sadržati: string koji odgovara broju sa tri decimalna mesta (%.3f), string (%s) i još jednom oba takva stringa. Uočiti na Sl. 3.6.5. da se koriste i dve specifične nizovne konstante (*Tab Constant*, za ubacivanje TAB razmaka) i (*End of Line Constant*, za prelazak u novi red) koje se mogu naći u paleti ***Programming»String***.

3. Pokrenuti izvršavanje programa. Nakon zaustavljanja programa pritiskom na dugme STOP, otvoriti u *Notepad*-u kreiranu datoteku i pregledati da li je njen sadržaj u skladu sa zahtevima zadatka. Potom je zatvoriti i ponovo otvoriti, ali korišćenjem *Microsoft Excel*-a.
4. Sačuvati program kao *upis_u_tekstualnu_datoteku.vi*.

Zadatak 6.4.

Kreirati matricu brojeva i upisati je u tekstualnu datoteku korišćenjem *High Level* funkcije *Write Delimited Spreadsheet* (***Programming»FileI/O***) za upis u tekstualnu datoteku.

1. Kreirati novi .vi program.
2. Na *Front Panel*-u kreirati matricu kao numeričku kontrolu (Lekcija 3) i uneti njene vrednosti, Sl. 3.6.7.



Sl. 3.6.7. Matrica – numerička kontrola

3. Kreirati *Block Diagram* kao na Sl. 3.6.8.

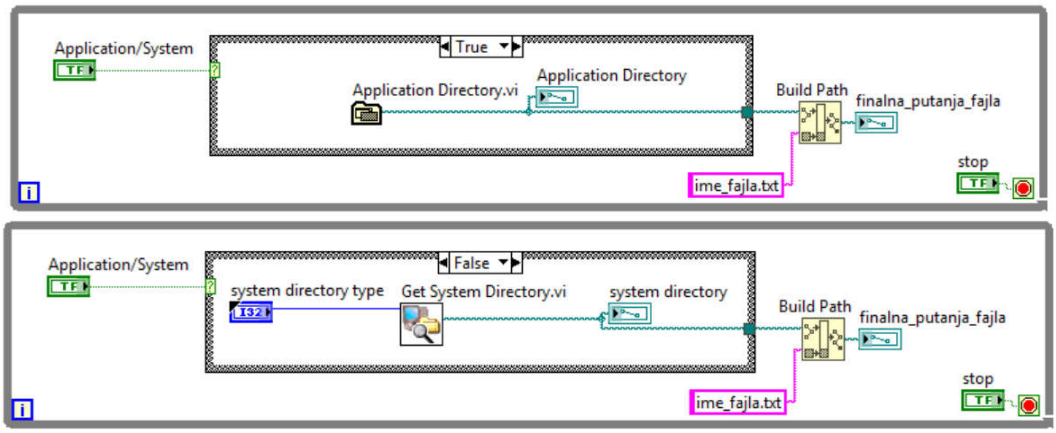
Sl. 3.6.8.

4. Pokrenuti izvršavanje programa. Pojaviće se prozor za izbor datoteke za upis. Uneti ime datoteke. Nakon završetka rada programa pogledati da li je datoteka kreirana i da li sadrži matricu koja je zadata numeričkom kontrolom.
5. Sačuvati program kao *high_level_tekstualna datoteka.vi*.
6. Pomoću *Context Help* opcije (CTRL+H dok je miš iznad funkcije) pregledati šta postoji od ulaznih priključaka funkcije *Write Delimited Spreadsheet* (mogućnost transponovanja – *transpose?*, mogućnost dodavanja sadržaja u postojeću datoteku – *append to file?*, upis *1D* ili *2D* niza i sl.).
7. Dva puta kliknuti mišem na *High Level* funkciju *Write Delimited Spreadsheet*. Pojaviće se *Front Panel* ove funkcije (*SubVI*-a). Pogledati i sadržaj Block Diagram-a ove funkcije. Uočiti sličnosti i razlike u odnosu na *Block Diagram* kreiran u Zadatku 6.3.

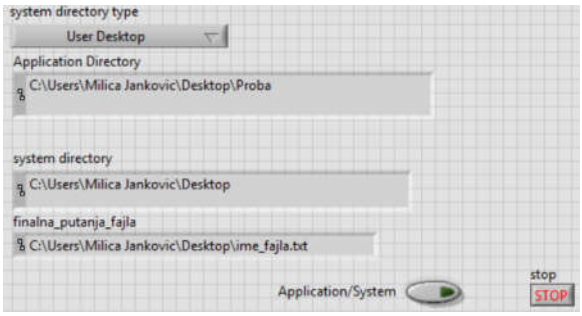
8. Zatvoriti program.

Zadatak 6.5.
Kreirati putanju datoteke koja se nalazi ili na *Desktop*-u ili u direktorijum u kome se nalazi *.vi* program. Omogućiti korisniku da bira na interfejsu način kreiranja putanje. Program se završava pritiskom na dugme STOP.

1. Kreirati novi .vi program.
2. Kreirati *Block Diagram* kao na Sl. 3.6.9. Pomoću funkcije *Application Directory* (**bilo koji** operativni sistem) je moguće generisati putanju do direktorijuma u kome je sačuvan .vi (ako nije sačuvan vrednost će biti <Not A Path>). Pomoću funkcije *Get System Directory* (**samo za Windows** operativni sistem) je moguće generisati putanju do nekih od sistemskih direktorijuma. Funkcija *Build Path* (***Programming»FileI/O***) omogućava spajanje putanja. *Front Panel* programa je prikazan na Sl. 3.6.10. Logičku kontrolu *Application/System* podesiti tako da ima *Switch* mehaničku akciju.
3. Pokrenuti izvršavanje programa i posmatrati kako se menja finalna putanja datoteke u zavisnosti od vrednosti logičke kontrole *Application/System.* Zaustaviti izvršavanje programa pritiskom na dugme STOP.



Sl. 3.6.9.



Sl. 3.6.10.

4. Sačuvati program kao *programsko_kreiranje_putanje_datoteke*.

Zadatak 6.6.
U ***Help»Find Examples*** pronaći (pomoću *Search* opcije) i proučiti programe koji koriste različite načine upisa u datoteku:

- Binarna datoteka: *Simple Binary File.lvproj*

- TDSM (*Technical Data Management Streaming*) datoteka: *TDMS Write Events Data.vi* i *TDMS Read Events Data.vi*.

NAPOMENA: Poređenje formata datoteka je dato tabelom 6.1.

Tabela 6.1

	ASCII (tekstualni)	TDMS	Binarni
Numerička preciznost	+	+++	+++
Jednostavnost deljenja podataka	+++	++	+
Efikasnost	+	+++	+++
Preporuka za korišćenje	Ako numerička preciznost i veličina datoteke nije bitna	Brzi *streaming* podataka bez gubitka numeričke preciznosti	Male datoteke Direktan pristup bilo kom podatku u datoteci (ne moraju da se čitaju svi podaci redom)

Zadatak 6.7. – za samostalni rad

Kreirati program koji će vršiti akviziciju napona sa NTC termistora (ai_0) i fotootpornika (ai_1) povezanih u zasebne naponske razdelnike (kao u Lekciji 4). U formi potprograma (SubVI-a) izvršiti konverziju otpornosti NTC termistora u temperaturu prema formuli:

$$t[°C] = \frac{4300}{\ln\left(\frac{R_{NTC}}{0.0054117}\right)} - 273.$$

Omogućiti grafički prikaz dobijenog napona sa fotootpornika i temperature NTC termistora. Programom takođe obezbediti mogućnost uključivanja dve svetleće diode. Realizovati potprogram koji će obezbediti uključivanje odgovarajuće svetleće diode ukoliko srednja vrednost uzoraka dobijenih akvizicijom prelazi prag definisan od strane korisnika. Potprogram za uključivanje dioda primeniti na akviziciju *napona* fotootpornika (LED 1), kao i na proračunate vrednosti *temperature* sa NTC termistora (LED 2). Obezbediti kontinualno upisivanje *napona* fotootpornika i *temperature* sa NTC termistora u datoteku.

NAPOMENA: Obezbediti inicijalizaciju promenljivih kao i potrebnih kanala na početku programa, kao i *clean up* po njegovom završetku (podsetnik: koristiti *Sequence* strukturu). Obezbediti zaustavljanje programa ukoliko se javi greška pri akviziciji ili pri kontroli paljenja dioda ili se pritisne dugme STOP.

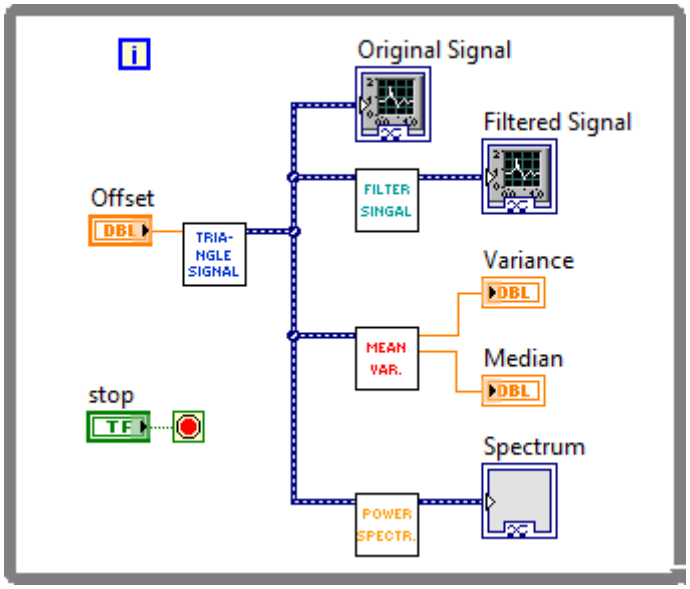
Lekcija 7 – Sekvencijalno programiranje. Mašina stanja. Paralelne petlje

Cilj

Cilj vežbe je da studente upozna sa:

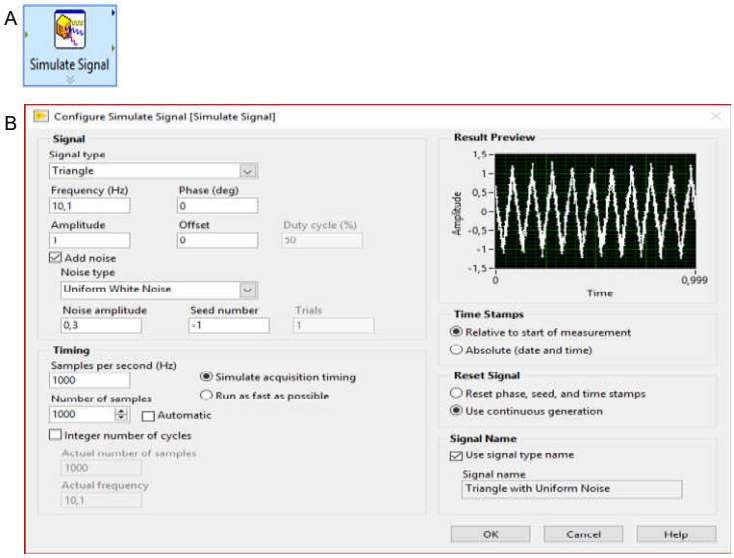
- Sekvencijalnim programiranjem
- Mašinom stanja
- Promenljivama
- Razmenom podataka između paralelnih petlji.

Zadatak 7.1.
Na Sl. 7.1 prikazan je blok dijagram programa koji je predstavlja inicijalno napisani kôd. Neophodno je realizovati kôd za svaki od predstavljenih SubVI. Zatim, potrebno je obezbediti da se nakon simulacije signala prvo izvršava određivanje srednje vrednosti i standardnog odstupanja signala, potom spektra snage signala i na kraju realizuje filtriranje signala. Zadatak realizovati na dva načina, korišćenjem *Sequence Structure* i pomoću toka signala ili signala greške. Preporuka je da se uvek koristi drugi način kada je to moguće, odnosno izbegavati *Sequence Structure*.

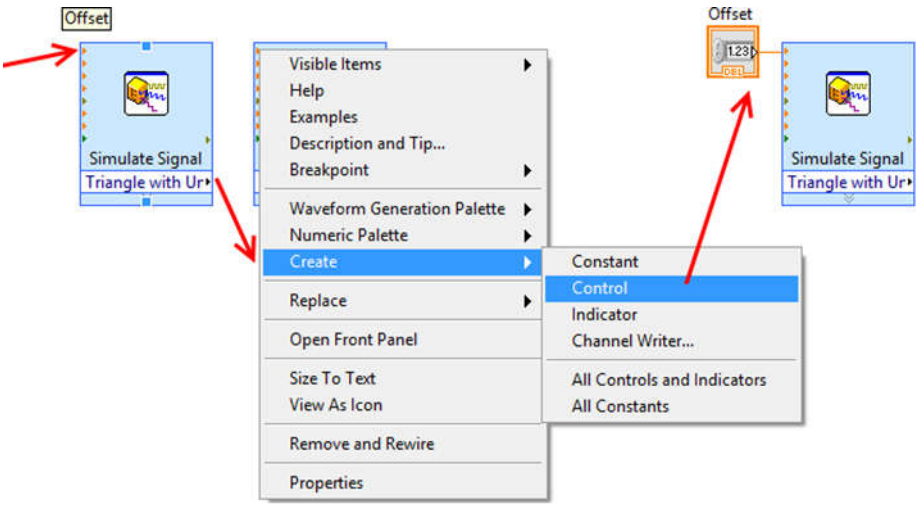


Sl. 7.1.

1. Kreirati novi .vi program. Ubaciti vi *Simulate Signal*, Sl. 7.2A. Kada se otvori dijagram za podešavanje podesiti parametre kao na Sl. 7.2B.
2. Klikom desnog tastera miša na polje *Offset* na ikoni *Express VI*-a *Simulate Signal*, zatim izabrati *Create»Control* dodati polje *Offset*, Sl. 7.3.

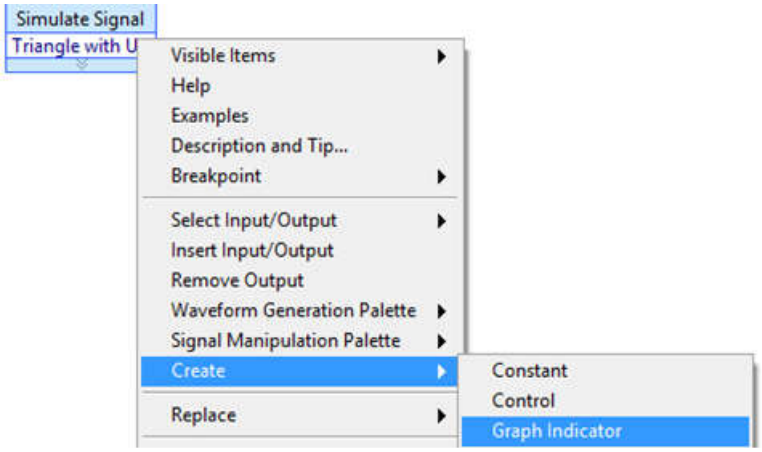


Sl. 7.2.



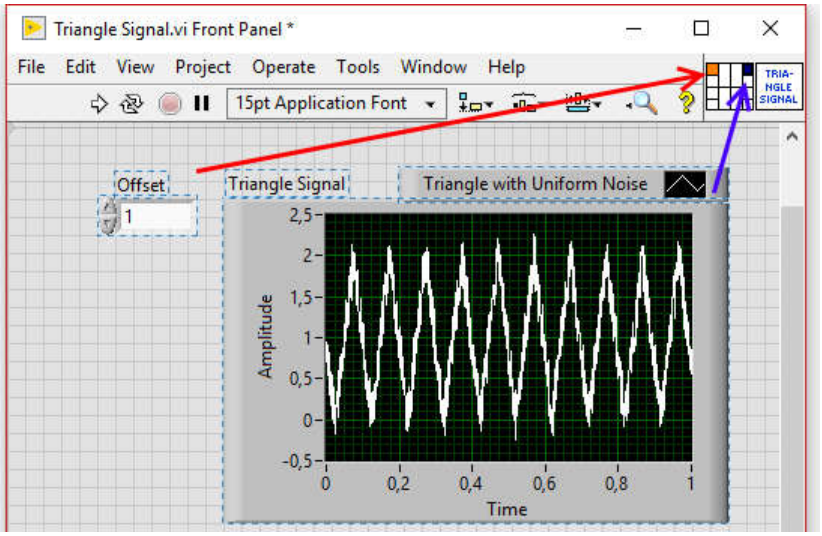
Sl. 7.3.

3. Sličnim postupkom dodati izlaz iz *Simulate Signal* i nazvati ga *Triangle Signal*, Sl. 7.4.



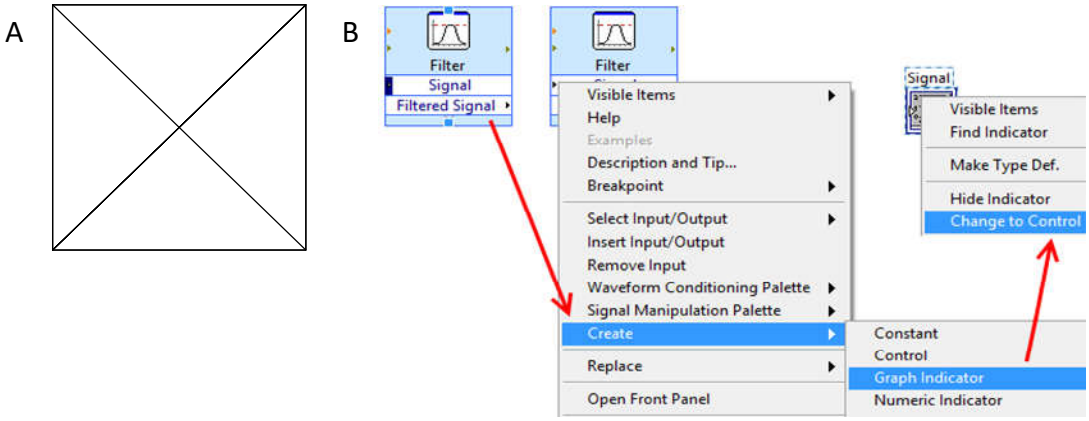
Sl. 7.4.

4. Promeniti ikonu i dodati ulaz i izlaz funkcije, Sl. 7.5, a zatim sačuvati SubVI pod imenom *Triangle Signal.vi*.



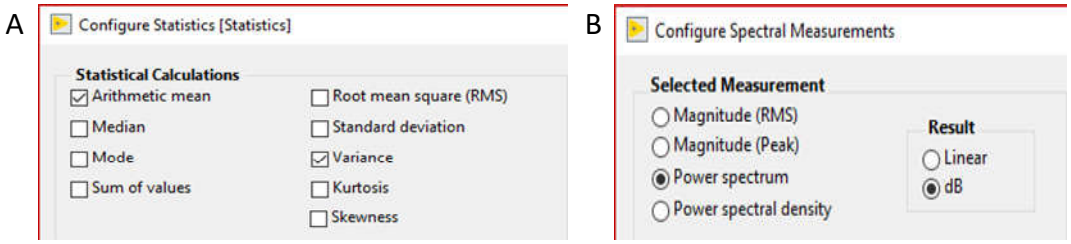
Sl. 7.5.

5. Slično, kao i za SubVI *Triangle Signal.vi* realizovati i ostala tri SubVI sa Sl. 7.1.
   a. Dodati *Express VI* za filtriranje signala *Filter*. Za tip filtra izabrati *Lowpass* i *Cutoff Frequency* postaviti na 40, Sl. 7.6A. Izlaz funkcije *Filter* dodati slično kao pod tačkom 3, dok za ulaz prvo izabrati *Graph Indicator*, a potom promeniti u kontrolu klikom na opciju *Change to Control*, Sl. 7.6B.



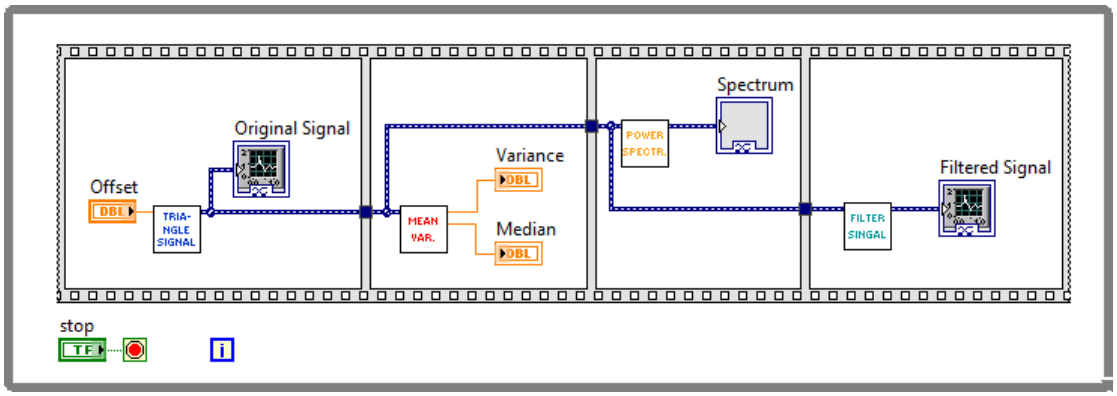
Sl. 7.6.

   b. Za SubVI *Mean and Variance.vi* izabrati *Express VI* "*Statistics*" i u samom čarobnjaku izabrati *Arithmetic mean* i *Variance*, Sl. 7.7A.
   c. SubVi *Power spectrum.vi* realizovati pomoću *Spectral Measurements* koji je takođe *Express VI*, a iz automatskog menija izabrati *Power spectrum*, Sl. 7.7B.



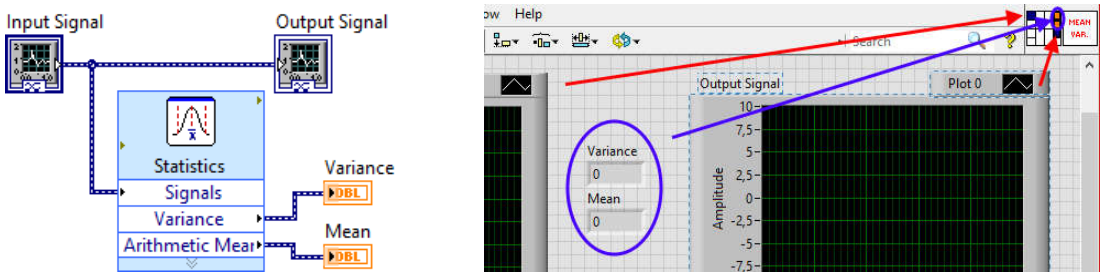
Sl. 7.7

6. Formirati blok dijagram dodavanjem svih realizovani SubVI unutar *while* petlje kako bi se dobio program prikazan na Sl. 7.1.
7. Redosled izvršavanja se može forsirati korišćenjem *Flat Sequence*, Sl. 7.8.



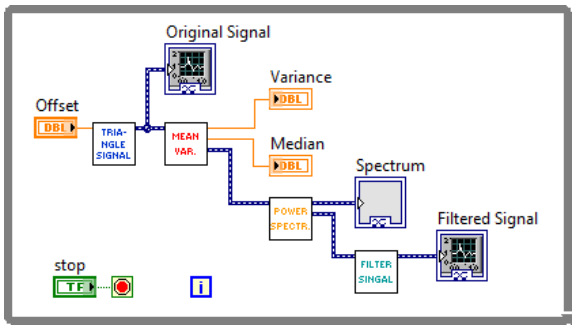
Sl. 7.8.

8. U SubVI *Mean and Variance.vi* dodati *Output Signal* koji predstavlja isti signal kao i na ulazu, Sl. 7.9. Slično uraditi i za *Power spectrum.vi*.



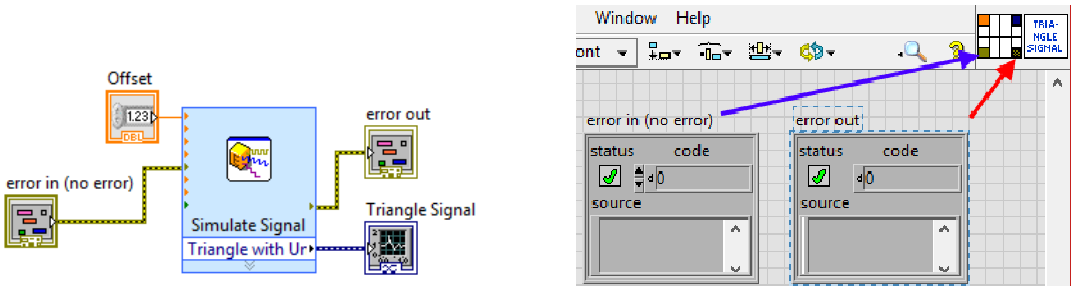
Sl. 7.9.

9. Umesto *Flat Sequence* povezati SubVI-eve jedan za drugim pomoću Input Signal i Output Signal konekcija, Sl. 7.10.



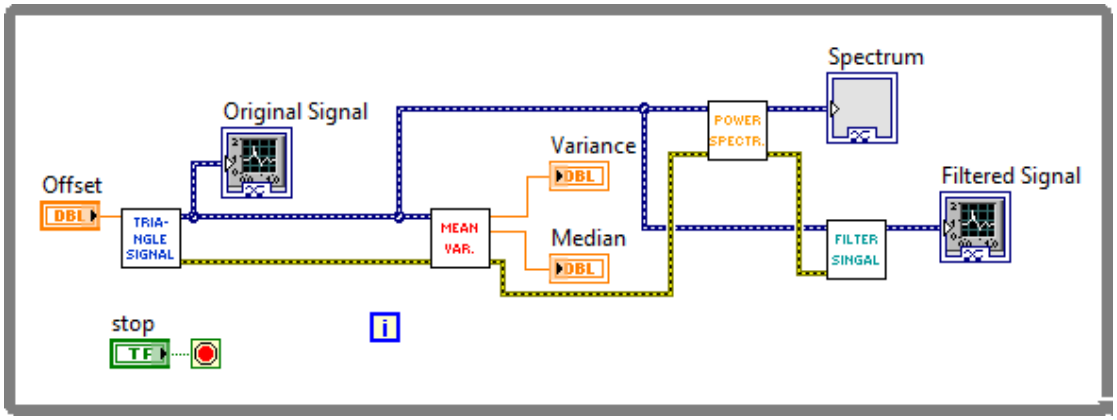
Sl. 7.10.

10. U pojedinim situacijama moguće je da SubVI nema potrebe da ima ulaz ili izlaz. Naravno, tada se može dodati promenljiva koja nema nikakvu funkcionalnost i dalje je koristiti za sekvencijalno izvršavanje programa. U takvim slučajevima, pogodno je da ta dodatna promenljiva bude baš klaster greške, odnosno, potrebno je klaster greške dodati na svaki od SubVI-jeva kako bi se upravo posredstvom ovog signala izvršilo njihovo ulančavanje i definisao određeni redosled izvršavanja., Sl. 7.11. Većina ugrađenih VI u *LabVIEW* sadrži klaster greške.



Sl. 7.11.

11. Sada obezbediti sekvencijalno izvršavanje pomoću signala greške, Sl. 7.12.



Sl. 7.12.

Zadatak 7.2.

Ponoviti Zadatak 3.7.2. uz manje izmene, koristeći mašinu stanja (*State Machine*): Napraviti program za igru na sreću. U ovoj igri na sreću povlačenjem ručice na korisničkom interfejsu na dole u članove tročlanog niza brojeva počinju da se upisuju slučajne celobrojne vrednosti od 0 do 100. Obezbediti da se vrednosti menjaju na 10 ms. Vraćanjem ručice u početni položaj se upisivanje vrednosti prekida i trenutno zatečeni brojevi se sabiraju. Ukoliko je rezultat sabiranja veći ili jednak 150, igrač dobija poen. U suprotnom igrač gubi dva poena. Igra se završava kada igrač odluči da unovči svoje poene ili kada izgubi sve poene. Početni broj poena pri pokretanju igre je 5. Obavestiti igrača da je izgubio sve poene i tada se igra vraća u početno stanje gde se očekuje da igrač (novi ili stari) ubaci novčić, pritiskom na taster **novčić**. Do ubacivanja novčića ručica za povlačenje i taster **unovči** su u stanju *disable*. Kada igrač ubaci novčić, taster **unovči** i ručica za povlačenje prelaze u stanje *enable*, a taster **novčić** u stanje *disable.* Ukoliko igrač pritisne dugme **unovči**, dati mu mogućnost da se predomisli. Program se može zaustaviti pritiskom na taster **stop**, ali samo dok je broj poena 0. Ne sme se koristiti *Sequence* struktura.

1. Pri realizaciji mašine stanja potrebno je dizajnirati dijagram toka programa, kako bi se olakšalo pisanje kôda. Zato se definišu sledeća stanja:
    a. *Inicijalizacija* – u ovom stanju u polje **poeni** se upisuje 0, taster **novčić** i taster **stop** su u stanju *enable*, dok su taster **unovči** i prekidač **povuci** u stanju *disable.* Prelazi se u stanje *Čekaj*.
    b. *Čekaj* – Proverava se da li je korisnik aktivirao taster **stop** i ukoliko nije prelazi se na proveru da li korisnik pritisnu taster **novčić**. Ukoliko nije ostaje se u stanju *Čekaj*, a ukoliko jeste, taster **novčić** i taster **stop** prelaze u stanje *disable*, a taster **unovči** i prekidač **povuci** u stanju *enable*, u polje

poeni se upisu vrednost 5 i prelazi se u stanje *Igra*. Ukoliko je korisnik kliknuo na taster **stop** program se zaustavlja.

c. *Igra* – Proverava se da li korisnik aktivirao taster **unovči** i ukoliko jeste prelazi se u stanje *Isplata*. Ukoliko nije proverava se da li je korisnik prebacio prekidač **povuci** na dole i ako jeste taster **unovči** prelazi u stanje *disable* i prelazi se u stanje *Generiši*. Ako korisnik nije preduzeo nijednu od prethodno dve opisane radnje ostaje se u stanju *Igra*.

d. *Generiši* – slučajno se generišu tri broja od nula do sto, a zatim se proverava da li korisnik prebacio prekidač **povuci** na gore. Ako nije program se pauzira na 10 ms, a ako jeste prelazi se u stanje *Rezultat*.

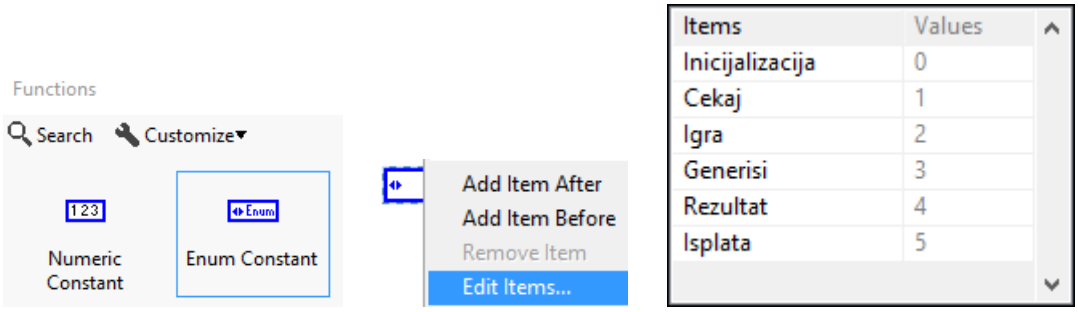
e. *Rezultat* – Provera se suma članova niza i ako je veća od 150 broj poena se uvećava za 1, u suprotnom se umanjuje za 2. Dobijen broj poena se proverava i ukoliko je manji od 1 korisnik se obaveštava da je završio igru i prelazi se u stanje *Inicijalizacija.* Ukoliko je broj poena veći od 0, taster **unovči** prelazi u stanje *enable* i program se vraća u stanje *Igra.*

f. *Isplata* – Proveriti da li je korisnik siguran da želi da mu se isplati dobitak. Ukoliko jeste prelazi se u stanje *Inicijalizacija*, a ako nije vraća se u stanje *Igra*.

2. Sada se prelazi na realizaciju prethodnog dijagrama toka. Preporuka je da se prethodni dijagram toka i skicira. Prvo, dodati sve kontrole i indikatore na front panel, Sl. 7.13. Niz od tri broja je realizovati kao tri zasebna indikatora.

Sl. 7.13.

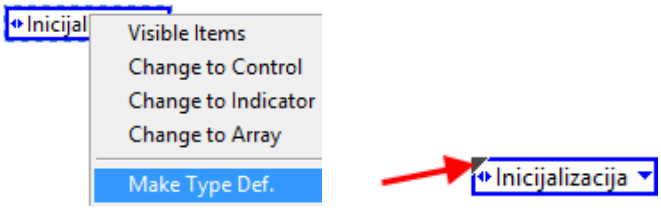
3. Formirati promenljivu tipa *enum* koja sadrži sva stanja programa. Nalazi se na paleti *Programming»Numeric*, Sl. 7.14. Zatim kliknuti desnim tasterom miša na unetu *enum* konstantu i izabrati *Edit Items..*, a potom uneti sve stanja mašine stanja, Sl. 7.14.



Sl. 7.14.

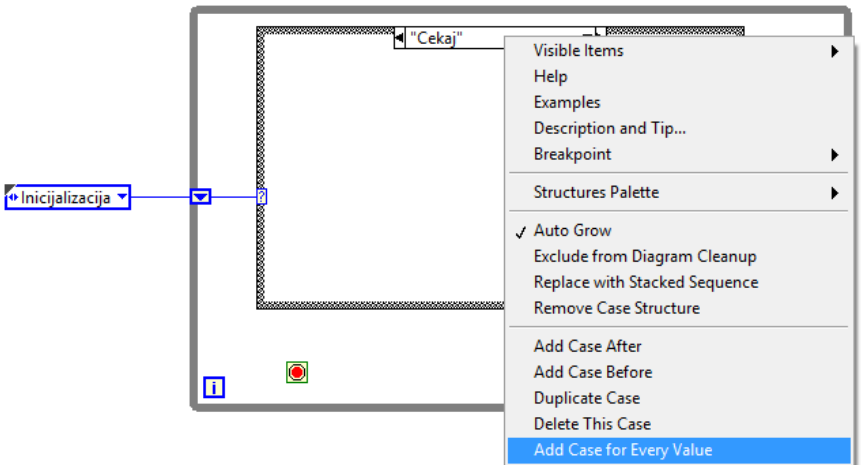
4. Kliknuti desnim tasterom miša na *enum* konstantu i izabrati *Make Type Def.* Sl. 7.15. Ovo omogućava da se da se svaka promena redosleda ili broja stanja menja samo na jednom mestu. Pri realizaciji kôda za mašinu stanja dobijeni *enum* će se često koristiti i ako ne bi bio definisan kao *Type Def.* na svakoj lokaciji morala bi da se

vršiti izmene svih stanja. Crni trouglić u gornjem levom uglu označava da je *enum* zadat kao *Type Def*.



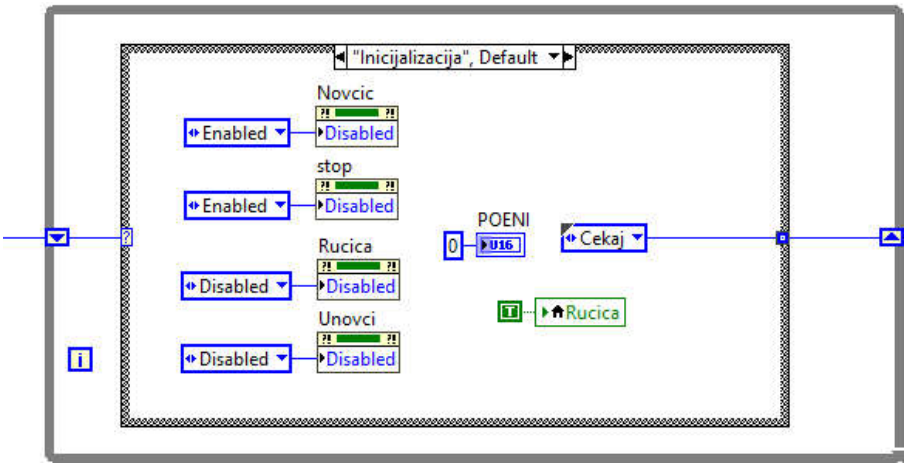
Sl. 7.15.

5. Na blok dijagram postaviti *while* petlju i u nju ubaciti *Case* strukturu. Dodati *Shift* registar na čiji ulaz je potrebno dovesti *enum* konstantu kôd koje je izabrana vrednost *Inicijalizaicja*, a izlaz iz *Shift* registra dovesti na ulaz *Case* strukture, Sl. 7.16. Time je obezbeđeno da je prvo stanje sa kojim počine izvršavanje programa *Inicijalizacija.* Potom, na vrhu *Case* strukture kliknuti desnim tasterom miša i izabrati *Add Case for Every Value*, čime se postiže da *Case* struktura može da obrađuje svako stanje mašine stanja, Sl. 7.16.



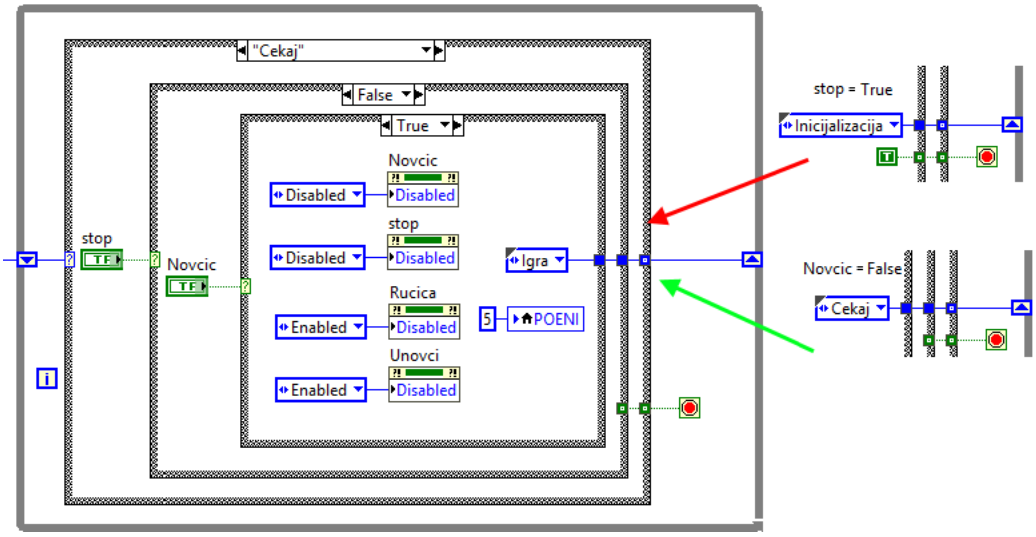
Sl. 7.16.

6. Realizovati kôd za stanje *Inicijalizacija* kao na Sl. 7.17. *Shift* registar obezbeđuje stanje za sledeću iteraciju, u ovom slučaju je to stanje *Cekaj*. Ručica se prebacuje u stanje *True*, što predstavlja položaj gore, jer je to prekidač i može se naći i u stanju dole pri pokretanju programa (kako?).



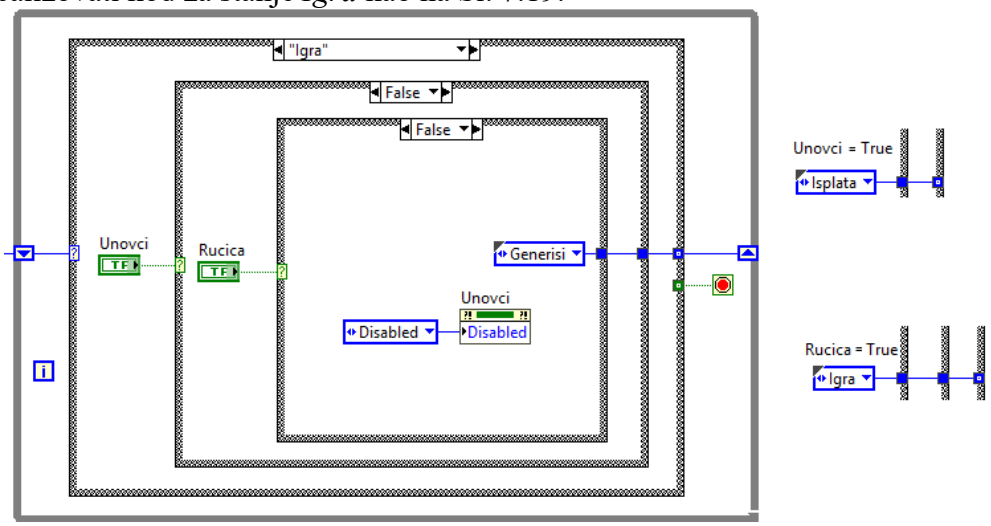
Sl. 7.17.

7. Realizovati kôd za stanje *Cekaj* kao na Sl. 7.18. Prikazani su delovi kôda kada je aktiviran taster **stop** i kada nije aktiviran ni jedan od tastera **stop** i **novcic.**



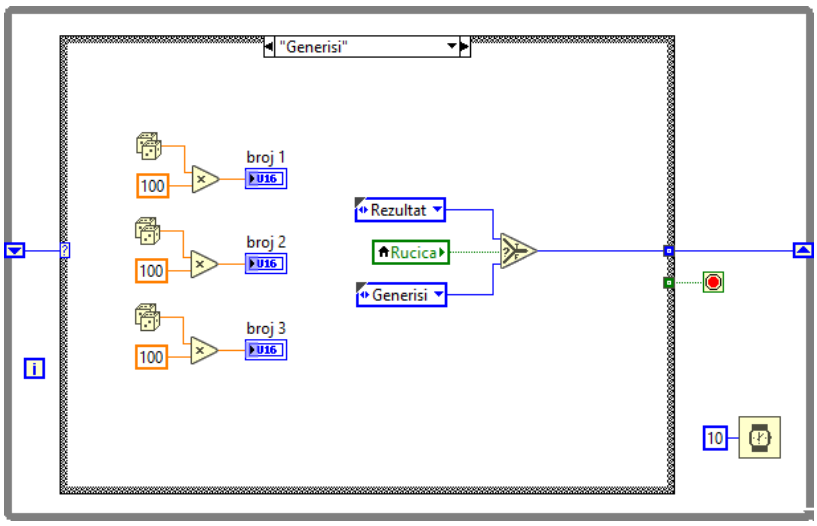
Sl. 7.18.

8. Realizovati kôd za stanje *Igra* kao na Sl. 7.19.



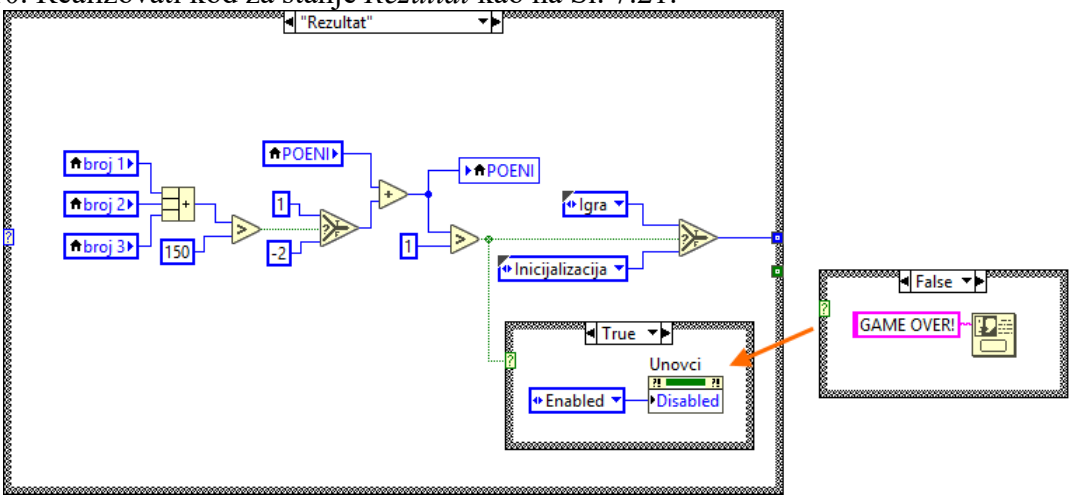
Sl. 7.19.

9. Realizovati kôd za stanje *Generisi* kao na Sl. 7.20. Funkcija *Wait* postavljena je van *while* petlje kako bi se obezbedilo da program ne uzima 100 % procesorskog vremena.



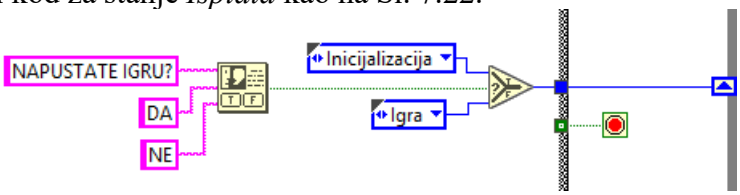
Sl. 7.20.

10. Realizovati kôd za stanje *Rezultat* kao na Sl. 7.21.



Sl. 7.21.

11. Realizovati kôd za stanje *Isplata* kao na Sl. 7.22.

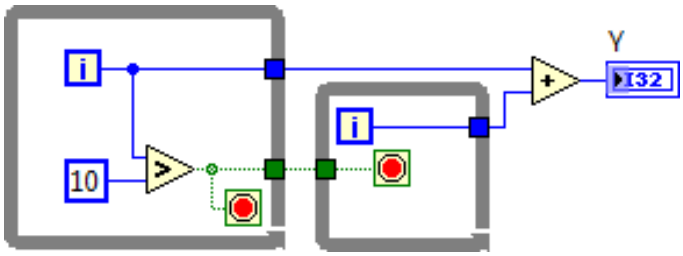


Sl. 7.22.

12. Testirati realizovani program. Pronaći sve lokacije gde se koriste lokalne promenljive. Kreirati lokalnu promenljivu za taster **novcic** i njenu vrednost dovesti na logički indikator. Zašto *LabVIEW* javlja grešku? Kako je moguće zadržati kreiranu lokalnu promenljivu i izbeći grešku? Da li to utiče na funkcionalnost programa.

Zadatak 7.3.

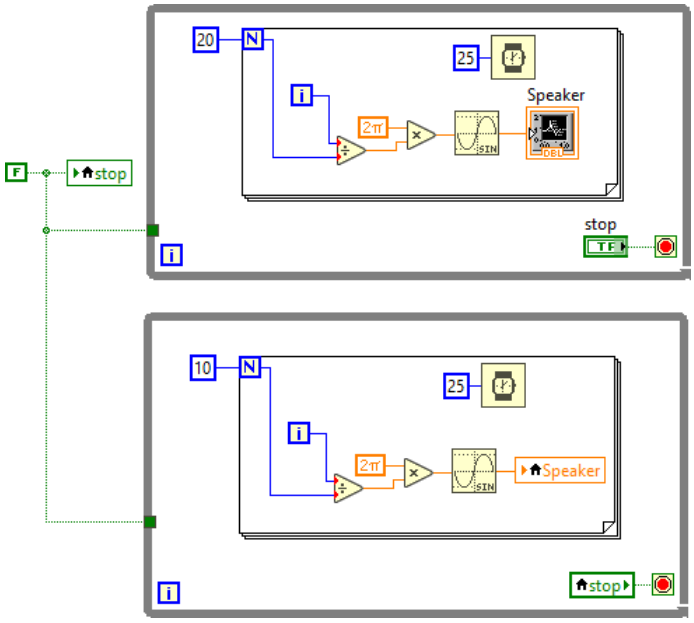
1. Realizovati kôd sa Sl. 7.23., a zatim obezbediti da se petlje istovremeno isključuju, pri čemu se druga petlja izvršava nekoliko puta. Koristiti lokalnu promenljivu za taster stop.



Sl. 7.23.

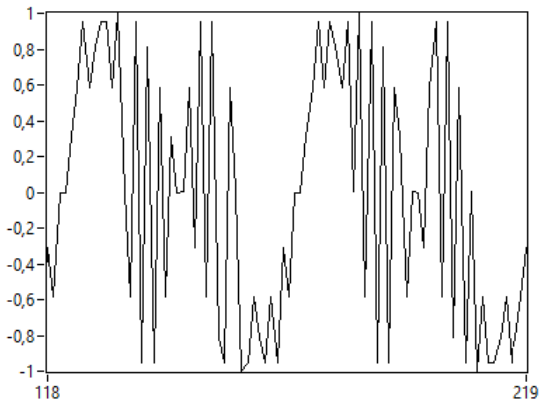
Zadatak 7.4.

1. Kreirati kôd sa Sl. 7.24. Indikator *Speaker* je *Waveform Chart*.

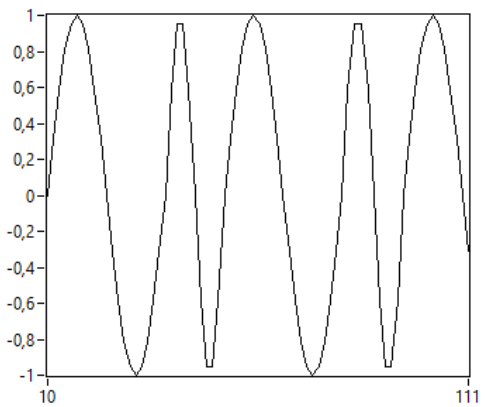


Sl. 7.24.

2. Pokrenuti napravljeni kôd. Objasniti zašto se dobija rezultat kao na Sl. 7.25A, a ne kao na Sl. 7.25. Pojava se naziva *Race condition* i nastaje kada dve petlje istovremeno upisuju u isti resurs preko lokalne promenljive. **Za samostalni rad**: realizovati lokalne promenljive kao globalne. Prednost globalnih je da se mogu koristi za razmenu podataka između dva VI.



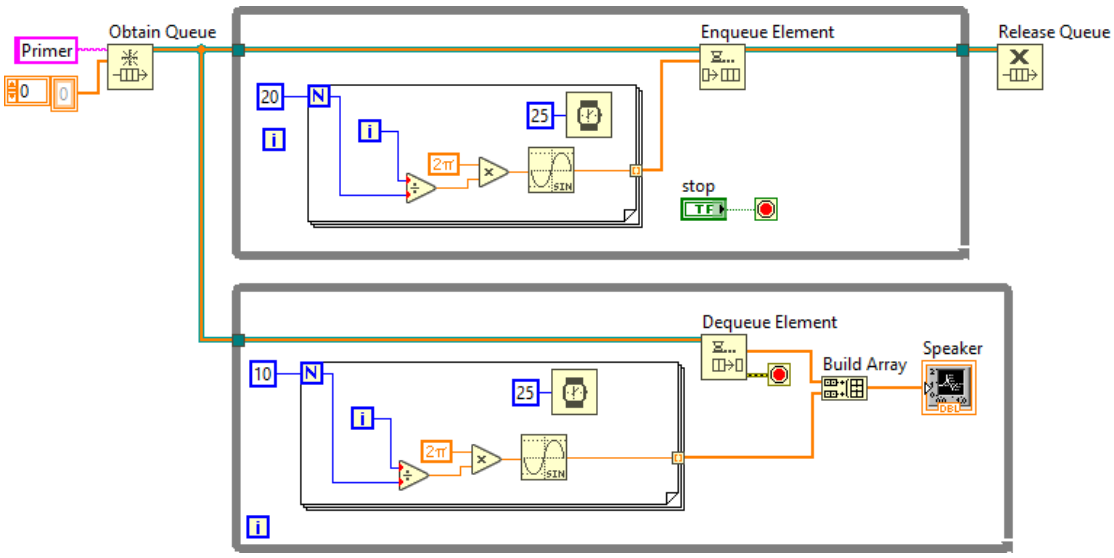
Sl. 7.25.A



Sl. 7.25.B

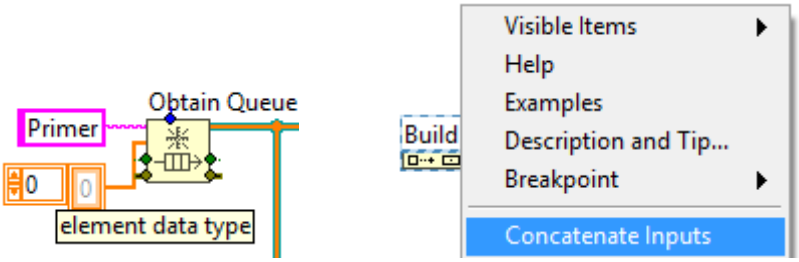
Zadatak 7.5.

1. Za prethodni zadatak potrebno je obezbediti da ne dolazi do pojave *Race Condition*. Da bi se to ostvarilo potrebno je realizovati kôd kao sa Sl. 7.26.



Sl. 7.26.

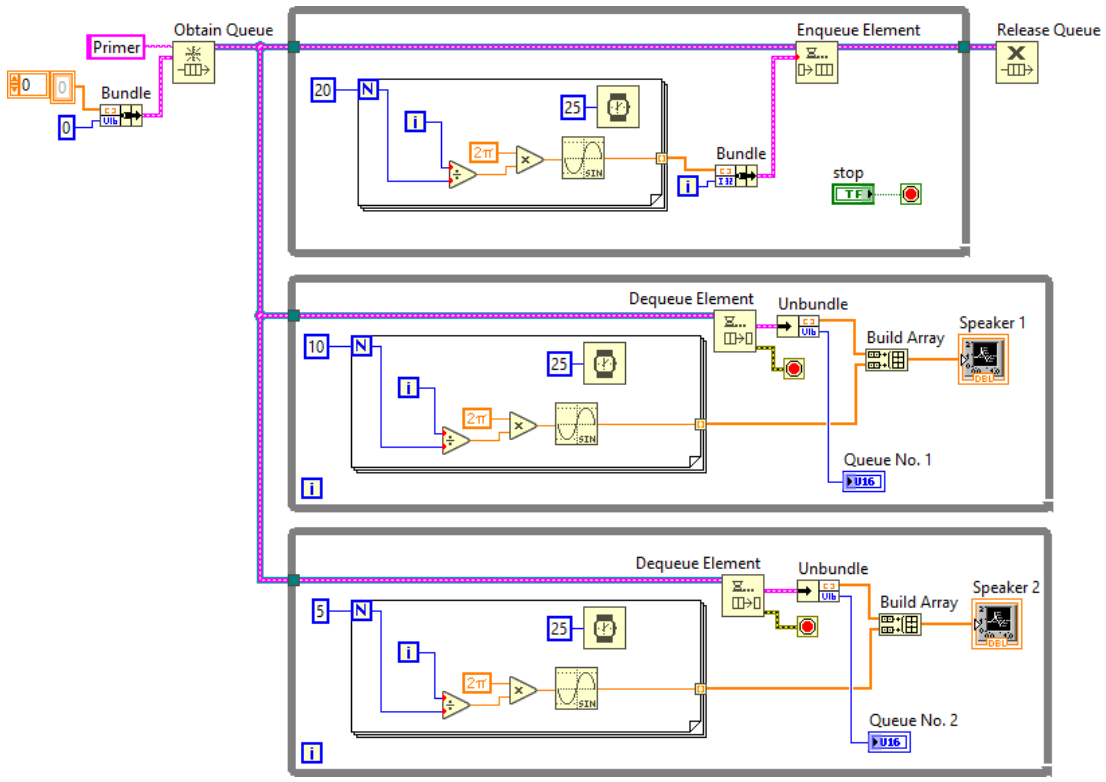
2. Prvo se indikator *Speaker* prebaci u donju *while* petlju, a zatim izlaz iz sinus funkcije dovede na izlazni tunel *for* petlje kako bi se dobio niz. To važi za obe *for* petlje. Obrisati sve lokalne promenljive za taster *stop*, a njega prebaciti u stanje *Latch*. Na ulaz *element data type* (Sl. 7.27) funkcije *Obtain Queue* dovesti konstantu tipa numerički niz. Ovim se obezbeđuje da je svaki element reda (*Queue*) tipa niz. Sve funkcije za rad sa redovima nalaze se na paleti *Programming » Synchronization » Queue Operations*. Kada se na blok dijagram unese funkcija *Build Array* kliknuti desnim tasterom miša na nju i izabrati opciju *Concatenate Inputs*, Sl. 7.27.



Sl. 7.28.

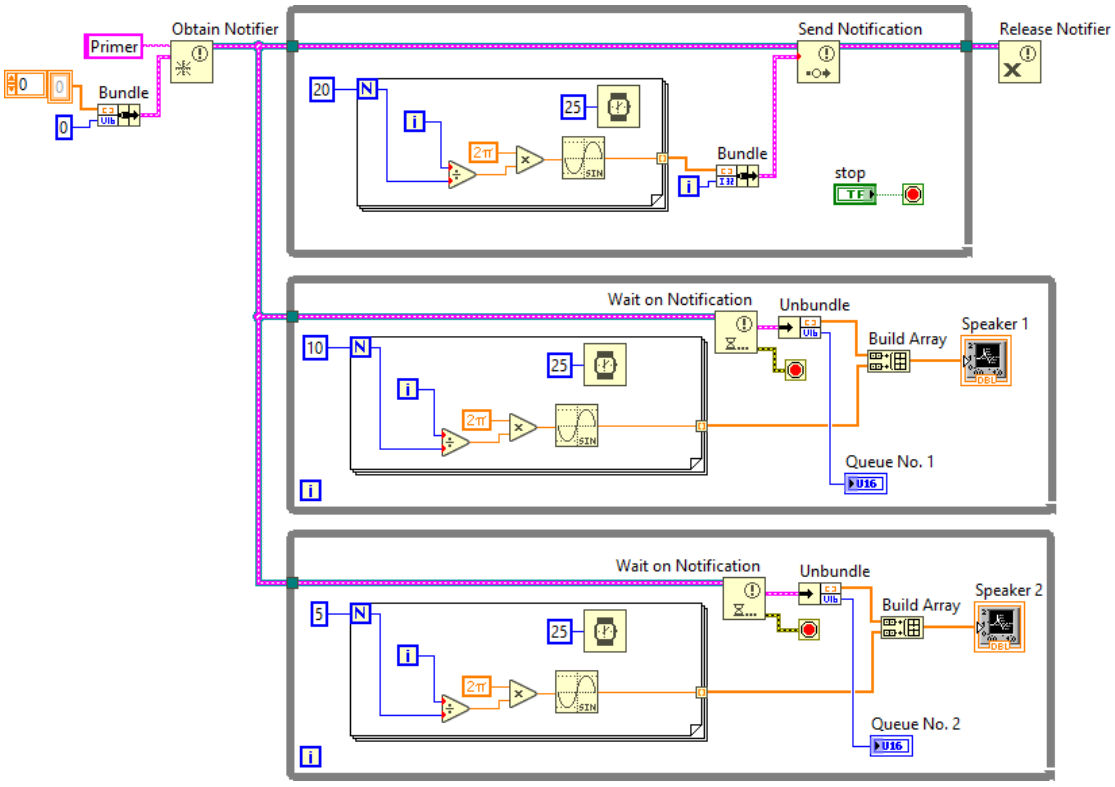
Zadatak 7.6.

1. Izmeniti kôd prethodnog zadatka kako bi se dobio kôd sa Sl. 7.29. Sada je element reda klaster koji sadrži niz i ceo broj.
2. Pokrenuti realizovani kôd i posmatrati šta se dešava sa poljima *Queue No. 1* i *Queue No. 2*. Funkcija *Enqueue Element* čita i briše element reda, zbog čega se svaki put indikatori *Queue No. 1* i *Queue No. 2* menjaju za dva, tj. u jednoj iteraciji element reda se pročita u srednjoj *while* petji, a u sledećoj se pročita u donjoj *while* petlji. Nemoguće je da obe petlje pročitaju isti element reda, tj. *Queue* nema mehanizam *Broadcasting*-a.



Sl. 7.29.

3. Da bi se prethodni problem rešio, umesto mehanizma *Queue* koristi se mehanizam *Notifier*, paleta *Programming » Synchronization » Notifier Operations*. Sada, realizovati kôd kao na Sl. 7.30. i testirati ga. **NAPOMENA**: *Notifer* za razliku od reda nema mogućnost skladištenja informacija pa se podatak može izgubiti ako korisnik podataka zakasni u nekoj od iteracija.



Sl. 7.30.

Lekcija 8 – Napredne metode programiranja. Distribucija aplikacije

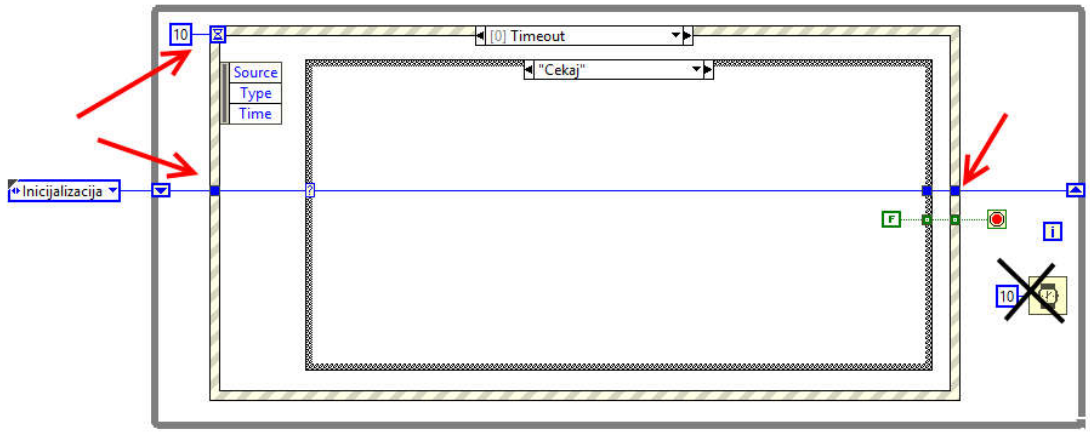
Cilj

Cilj vežbe je da studente upozna sa:

- Mašinom stanja sa *Event* strukturom.
- Funkcionalnom globalnom promenljivom.
- Programskim upravljanjem korisničkim interfejsom.
- Unapređenjenjem postojećih VI.
- Pripremom aplikacije za distribucije.
- Kreiranjem izvršne datoteke (EXE) i *installer*-a.

Zadatak 8.1.

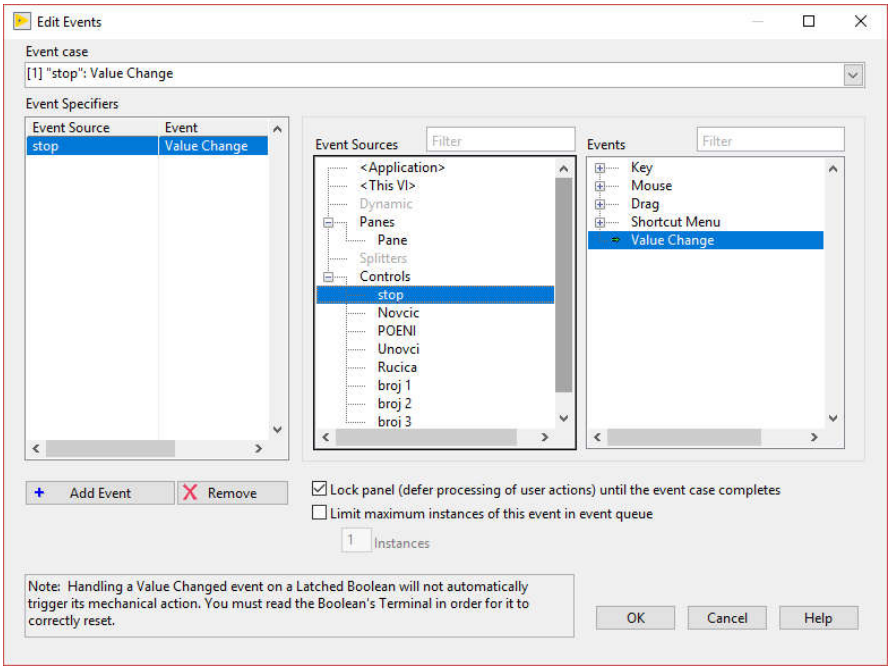
Polazeći od zadatka 7.2. realizovati Mašinu stanja sa *Event* strukturom: Napraviti program za igru na sreću. U ovoj igri na sreću povlačenjem ručice na korisničkom interfejsu na dole u članove tročlanog niza brojeva počinju da se upisuju slučajne celobrojne vrednosti od 0 do 100. Obezbediti da se vrednosti menjaju na 10 ms. Vraćanjem ručice u početni položaj se upisivanje vrednosti prekida i trenutno zatečeni brojevi se sabiraju. Ukoliko je rezultat sabiranja veći ili jednak 150, igrač dobija poen. U suprotnom igrač gubi dva poena. Igra se završava kada igrač odluči da unovči svoje poene ili kada izgubi sve poene. Početni broj poena pri pokretanju igre je 5. Obavestiti igrača da je izgubio sve poene i tada se igra vraća u početno stanje gde se očekuje da igrač (novi ili stari) ubaci novčić, pritiskom na taster **novčić**. Do ubacivanja novčića ručica za povlačenje i taster **unovči** su u stanju *disable*. Kada igrač ubaci novčić, taster **unovči** i ručica za povlačenje prelaze u stanje *enable*, a taster **novčić** u stanje *disable.* Ukoliko igrač pritisne dugme **unovči**, dati mu mogućnost da se predomisli. Program se može zaustaviti pritiskom na taster **stop**, ali samo dok je broj poena 0.



Sl. 8.1.

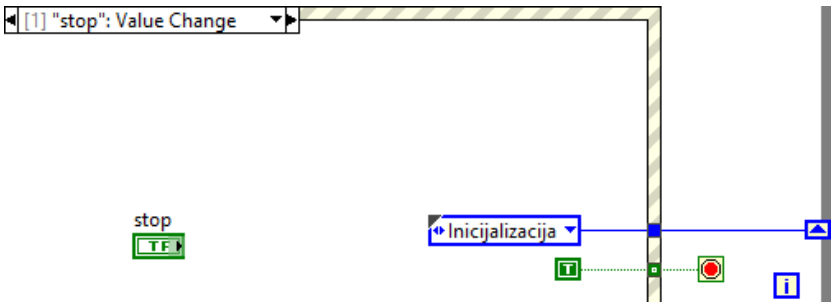
1. Otvoriti program *Igra na srecu - pocetak.vi*.

2. Oko *Case* strukture postaviti *Event Structure*, definisati ulaz *timeout* na 10 ms (ne može se ostaviti -1 jer bi se u tom slučaju beskonačno dugo čekalo na korisnički *Event*) i unutar stanja *timeout* povezati ulazno stanje na izlazno za Mašinu stanja (*shift register*). Obrisati *Wait* funkciju, sada je njenu ulogu preuzeo *timeout* ulaz *Event* strukture.
3. Dodati *Event Case* za taster *stop*, Sl. 8.2



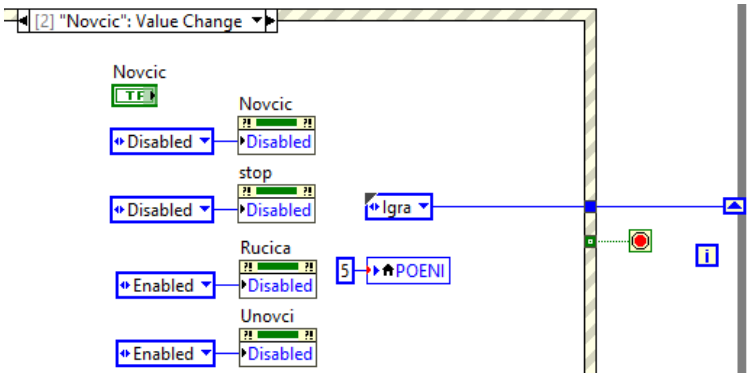
Sl. 8.2

4. Dodati kôd kao na Sl. 8.3 (koristiti već postojeći kôd, ako nije obrisan).



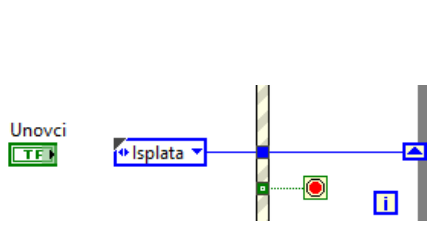
Sl. 8.3

5. Dodati *case* koji se izvršava prilikom aktivacije tastera *Novcic*, Sl. 8.4.

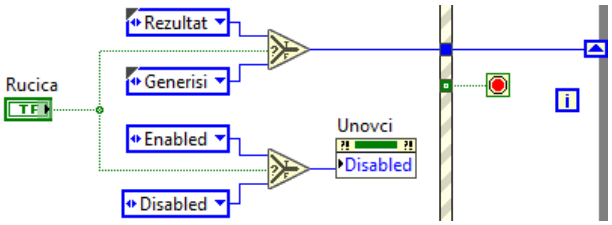


Sl. 8.4.

6. Slično je potrebno uraditi i pri aktiviranju tastera *Unovci* i tastera *Rucica*.

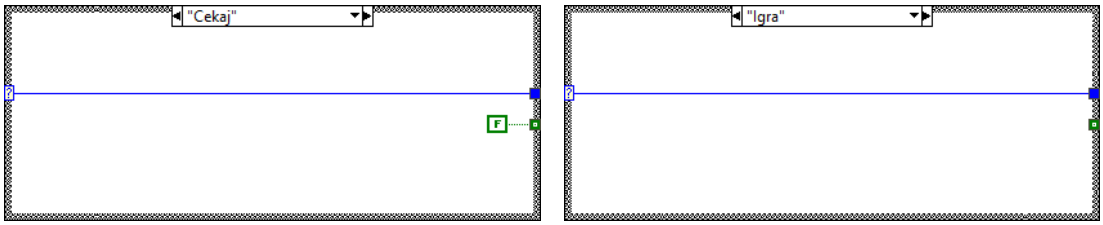


Sl. 8.5A



Sl. 8.5B.

7. Potom, potrebno je prilagoditi i kôd unutar *Case* strukture stanja za *Cekaj* i *Igra*, Sl. 8.6.



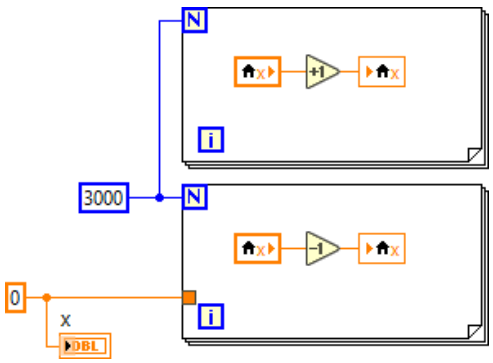
Sl. 8.6.

8. Sačuvati datoteku kao *Igra na srecu - kraj.vi*.
9. Testirati funkcionalost početne i krajne aplikacije. Realizacija sa *Event* strukturom se preporučuje, jer početni način provere da li je korisnik aktivirao određeni taster (naziva se i *pooling*) troši resurse, za razliku od *Event* strukture (*Event driven programming*).

Zadatak 8.2.

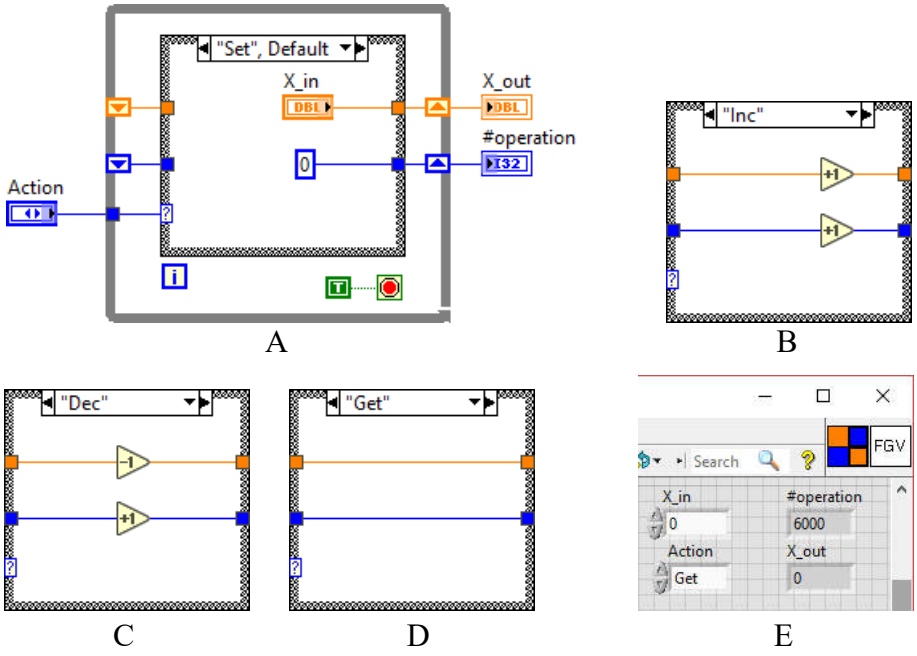
Race Condition se može rešiti i mehanizmom funkcionalne globalne promenljive kojom se vrši komunikacija između dve promenljive.

1. Realizovati kôd sa Sl. 8.7, a potom ga testirati i proveriti koju vrednost sadrži promenljiva *x*.



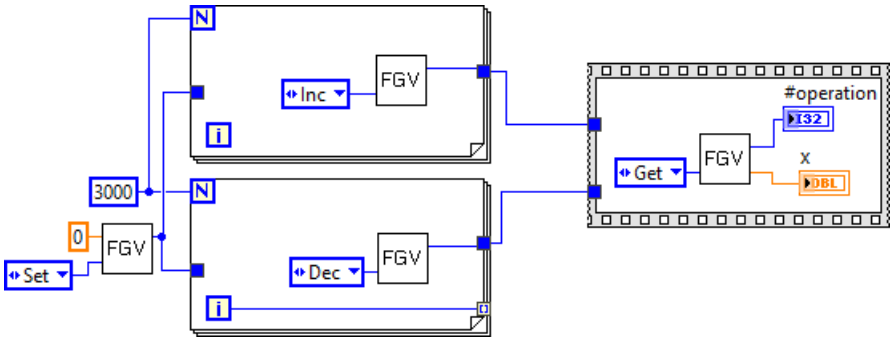
Sl. 8.7.

2. Zatim realizovati funkcionalnu globalnu promenljivu uvođenjem SubVI čija je struktura prikazana na Sl. 8.8.



Sl. 8.8.

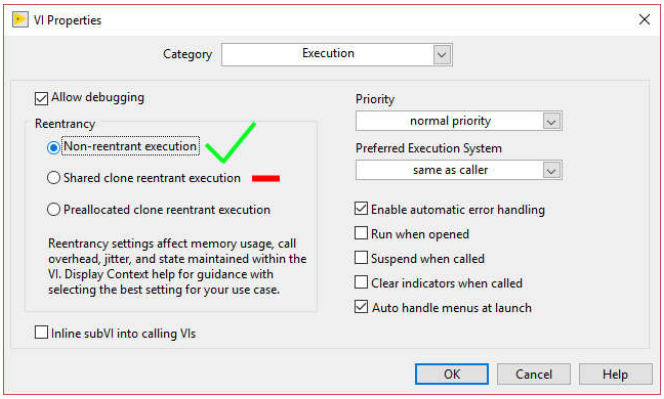
3. Sačuvati realizovani kôd pod nazivom *FGV.vi*. Kôd sa Sl. 8.7, modifikovati tako da izgleda kao na Sl. 8.9.



Sl. 8.9.

4. Testirati aplikaciju. Ukoliko se ne dobija očekivani rezultat promeniti u *VI Properties* u delu *Executioin*, tako da bude izabrano *Non-reentrant execution*, a ne *Shared clone reentrant execution*, Sl. 8.10. Ukoliko kôd radi ispravno, promeniti na *Shared clone reentratn execution* i testirati.

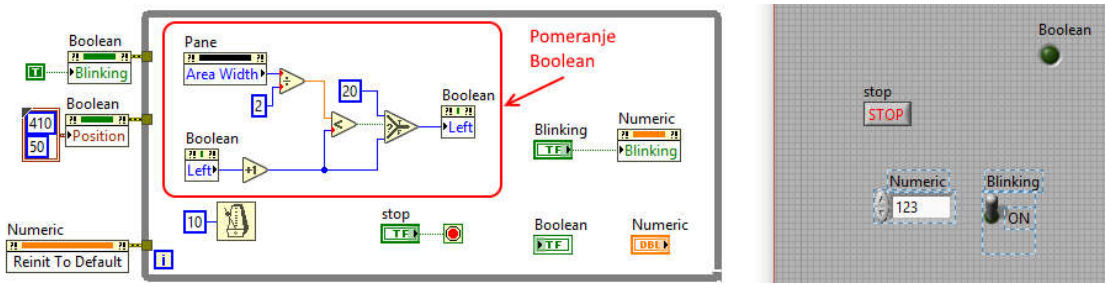
NAPOMENA: funkcionalna globalna promenljiva koristi *non-reentrant* osobinu funkcije, koja za svaki poziv funkcije koristi isti memorijski prostor, tada se trenutni poziv funkcije ne može započeti sve dok se ne završi prethodni poziv iste funkcije. Sam kôd *FGV.vi* sadrži *while* petlju koja se pri svakom pozivu izršava samo jednom i to onaj *Case* koji je zadat pri pozivu funkcije. Neinicijalizovan *Shift* registar omogućuva da se njegova vrednost pamti iz prethodnog poziva.



Sl. 8.10.

Zadatak 8.3.

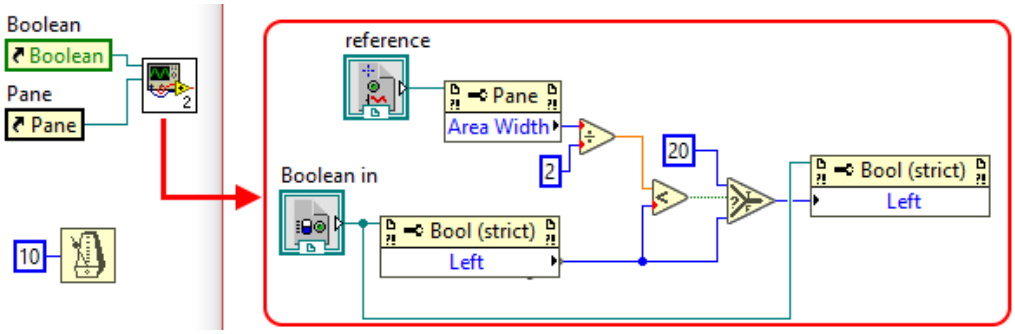
1. Realizovati *Block Diagram* i *Front Panel* kao na Sl. 8.11.



Sl. 8.11.

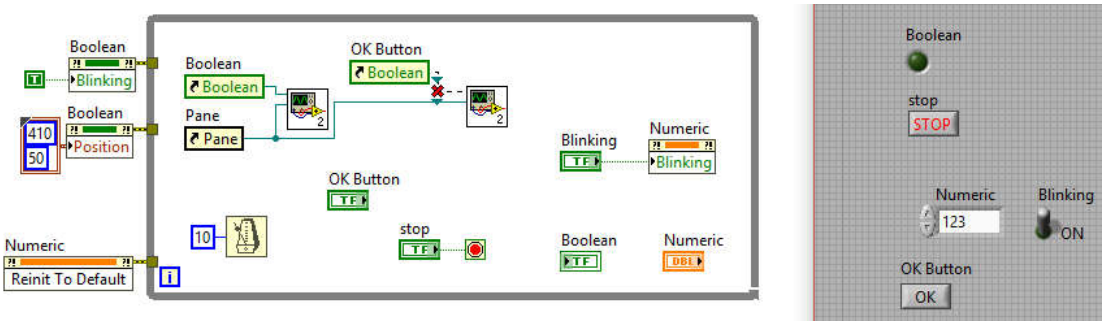
2. Testirati dobijenu aplikaciju. **Objašnjenje:** Pri pokretanju logički indikator *Boolean* pozicionira na koordinatu $x = 410$ i $y = 50$, a istovremeno se aktivira i osobina *Blinking* za isti indikator. *Property*-i logičkog indikatora *Boolean* se dobija klikom desnog tastera miša na terminal *Boolean* i izborom *Create/Propety Node/Blinking*. Pre pokretanja *while* petlje izvršava se i *Invoke Node* za numeričku kontrolu *Numeric* koji je vraća na *default* vrednost (*Reinit to Default*). U samoj *for* petlji aktiviranjem tastera *Blinking* osobina *Blinkning* za numeričku kontrolu se pali ili gasi. Levi deo kôda pomera logički indikator do polovine širine front panela. Opisani način povezivanja *Property node* (*Invoke node*) sa odgovarajućim terminalom naziva se implicitni.

3. Deo kôda označen crvenim pravougaonikom na Sl. 8.11. selektovati i od njega napraviti SubVI (ići na *Edit/Create SubVI*). Generisaće se kôd kao na Sl. 8.12. Unutar SubVI-a se poziva *Property Node* eksplicitno, prosleđivanjem reference (*control reference*) na određeni objekat u radnoj memoriji. Naime, svakom objektu kao što su logički indikator *Boolean* ili sam front panel (*Pane*) se dodeljuje jedinstveni broj (referenca). Sada *Property node* nije vezan za određeni objekat, već mu se može proslediti referenca na bilo koji objekat koji je tipa *Boolean*.



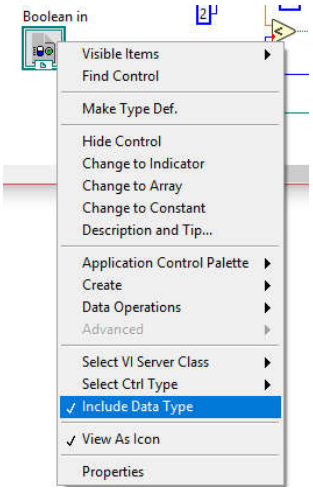
Sl. 8.12.

4. Dodati jednu logičku kontrolu na *Front Panel* glavnog VI, a zatim dodati još jedan poziv istog SubVI, Sl. 8.13. Međutim, javlja se greška.



Sl. 8.13.

5. Greška je posledica toga što se zahteva da *Property Node* unutar *SubVI* mora biti povezan sa referencom na objekat istog tipa (*strict*). Da bi se uklonila greška, potrebno je deselektovati polje *Include Data Type*, Sl. 8.14. Sada ne dolazi do greške u glavnom VI. Testirati dobijeni kôd. Može se uočiti prednost korišćenja kontrolne reference.

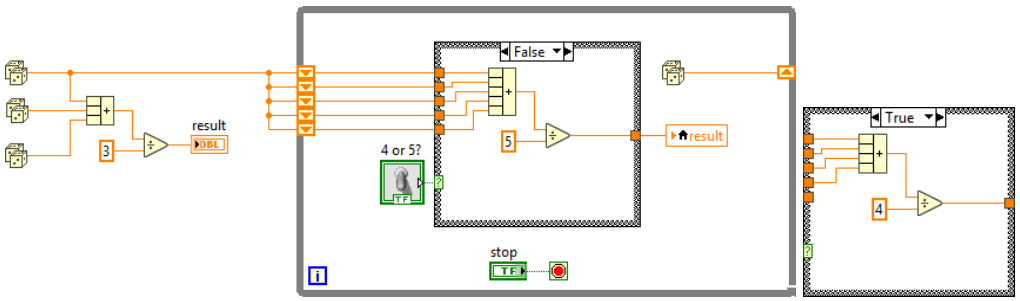


Sl. 8.14.

Zadatak 8.4.

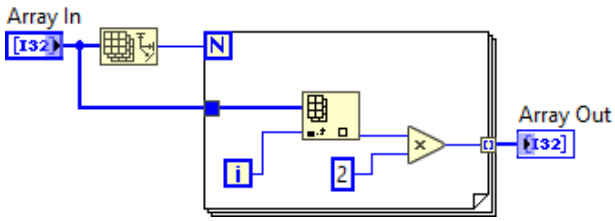
Poboljšanje postojećeg VI.

1. Uklanjanje dupliranog kôda. Identičan/sličan kôd je potrebno zameniti sa SubVI, jer se tada kôd menja samo na jednom mestu, pa je upravljanje kôdom jednostavnije. Realizovati kôd sa Sl. 8.15. Zatim obezbediti da se sličan kôd ne ponavlja (računanje srednje vrednosti).



Sl. 8.15.

2. Uklanjanje komplikovane logike. Realizovati kôd sa Sl. 8.16, a potom identifikovati i ukloniti komplikovanu logiku.

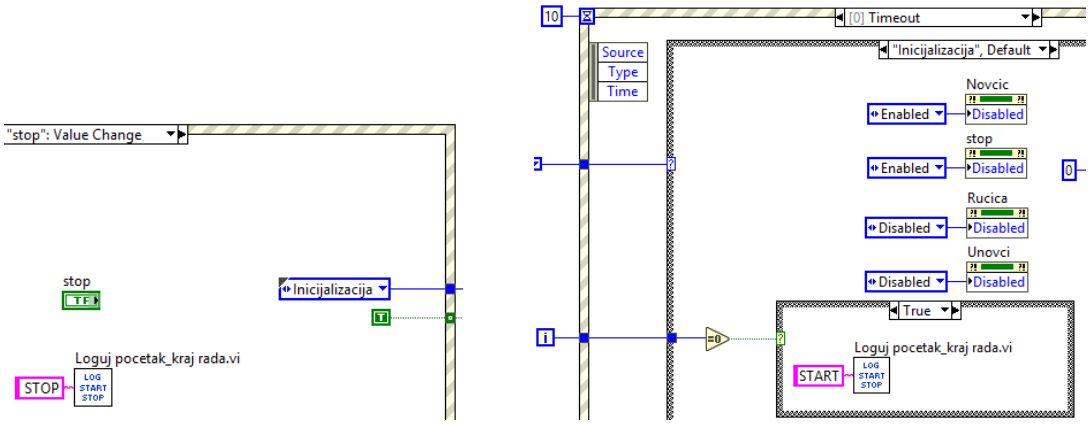


Sl. 8.16.

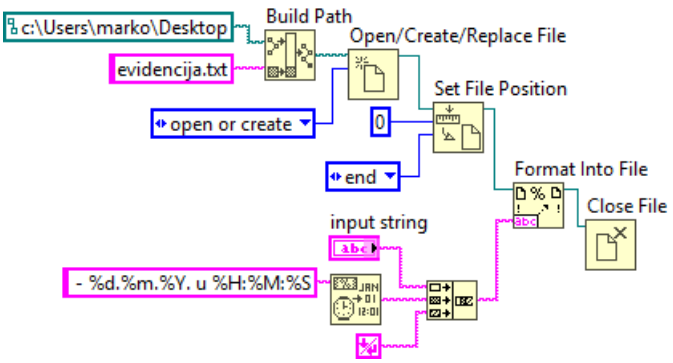
Zadatak 8.5.

Priprema VI za distribuciju kao samostalne aplikacije (*standalone*).

1. Pokrenuti snimljeni program *Igra na srecu - kraj.vi*. Potrebno je obezbediti da se u direktorijumu "c:\Users\marko\Desktop" vodi evidencija u vidu log datoteke u kojoj se upisuje vremenski trenutak (datum i vreme) za svako pokretanje i svaki izlazak iz aplikacije. Umesto "marko" koristiti ime trenutnog korisnika. Realizovati kôd kao na Sl. 8.17. i Sl. 8.18.

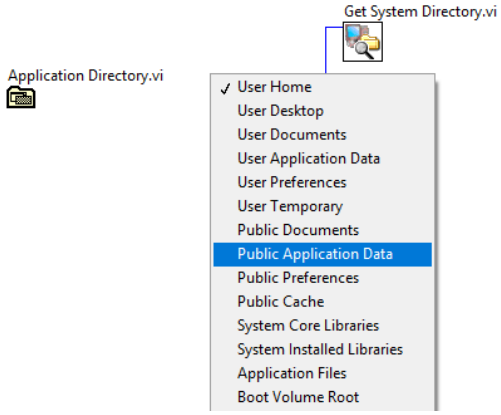


Sl. 8.17. Izmena dela kôda.



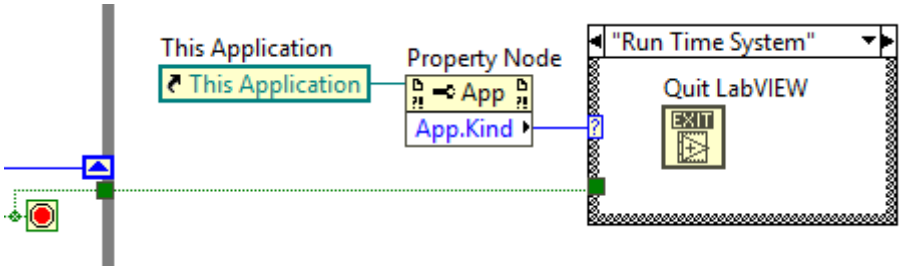
Sl. 8.18. Kôd za VI "Loguj pocetak_kraj rada".

2. Pokrenuti kôd i testirati. Međutim, pri distribuciji ove aplikacije verovatno će se javiti greška, jer je moguće da na računaru krajnjeg korisnika ne postoji zadati direktorijum "c:\Users\marko\Desktop". Zato je potrebno zadati drugi direktorijum koji sigurno postoji i dve opcije su prikazane na Sl. 8.18. Zameniti sa *Applicaton Directory.vi* postojeću putanju direktorijuma (Sl. 8.18).



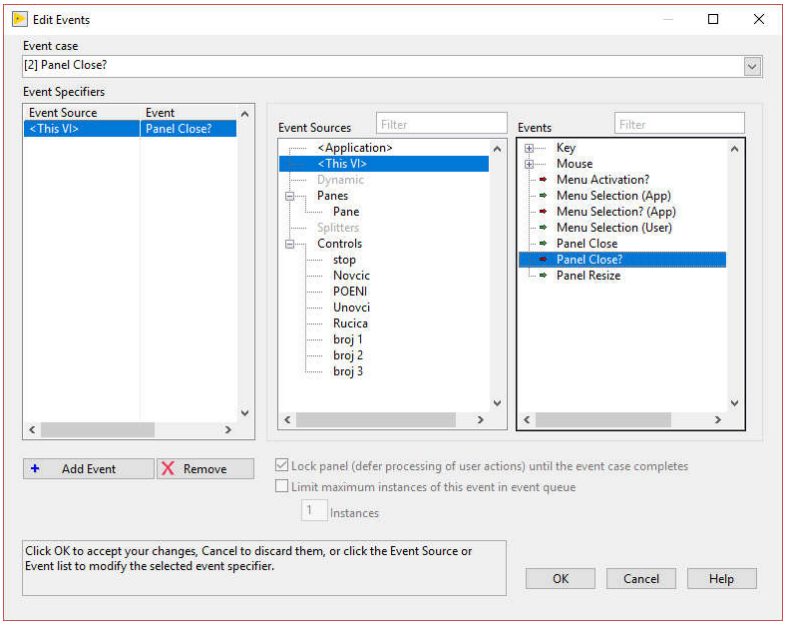
Sl. 8.19.

3. Sledeća izmena je neophodna kako bi se osiguralo da se napusti *LabVIEW* okruženje, prilikom izlaska iz aplikacije. U suprotnom ostao bi front panel koji više ne funkcioniše i korisnik bi morao sam da ga isključi. Potrebno je pozvati funkciju *Exit* kada se aktivira taster *stop*. Međutim, ako se bi se dodala funkcija *Exit* bez dodatnih provera (da li je *standalone*) aplikacija, *LabVIEW* okruženje bi se isključivalo i u toku razvoja aplikacije, što bi imalo vrlo frustrirajući efekat na korisnika. Da bi se to sprečilo potrebno je dodati kôd kao na Sl. 8.20. Referenca "This Application" dobija se izborom "Vi server reference" sa subpalete *Application Control*. Izlaz iz *Property Node* se vodi na *Case* strukturu. Međutim, *default* vrednost ne sadrži *case* "Run Time System" pa je potrebno kliknuti desnim tasterom miša na *Case* strukturu i izabrati *Add Case for Every Value*. Stop iz *while* petlje se dovodi na ulaz *Case* strukture da bi se obezbedilo izvršavanje *Case* strukture nakon *while* petlje.



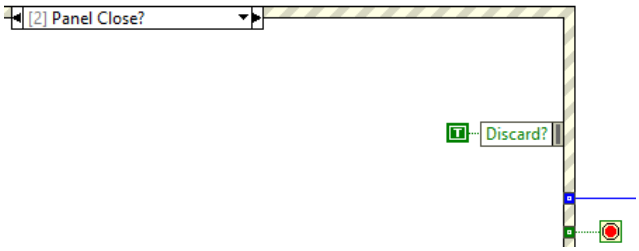
Sl. 8.20.

4. Preporučuju se i zabrana zatvaranja aplikacije klikom na X u desnom gornjem uglu, što se može obaviti dodavanjem još jednog *Event*-a u *Event* strukturi, Sl. 8.21. Potrebno je izabrati događaj *Panel Close?*. Crvena strelica označava da nije u pitanju *Notify Event* već *Filter Event*. Prvi tip događaja obaveštava aplikaciju da se aktivnost već dogodila, dok drugi tip omogućava da se aktivnost i spreči.



Sl. 8.21.

5. Zatim, potrebno je dodati i kôd kao na Sl. 8.22. u odgovaraju *case*.



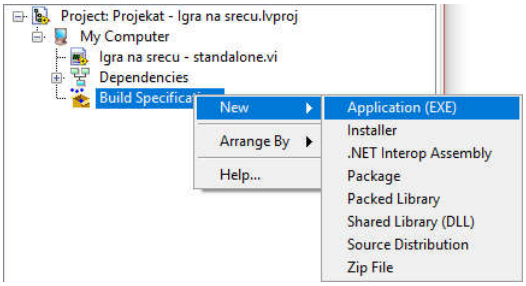
Sl. 8.22.

6. Sačuvati dobijeni program kao *Igra na srecu – standalone.vi*. Kreirati novi projekat "Projekat - Igra na srecu" i dodati prethodno sačuvani VI u njega.

Zadatak 8.6.

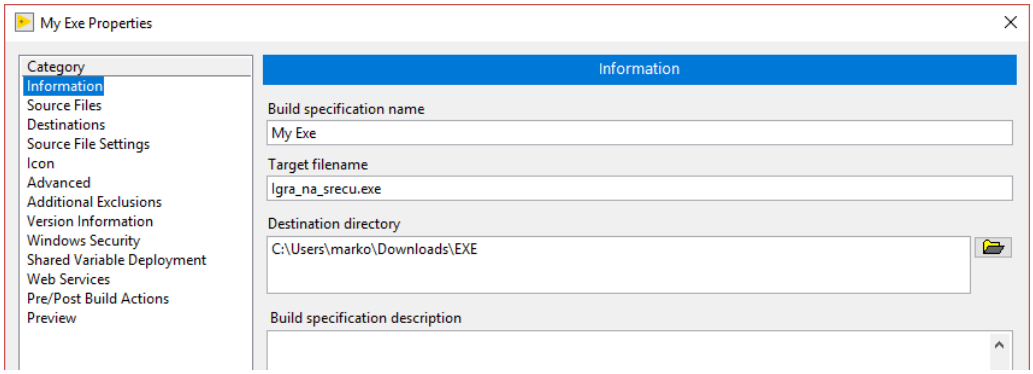
Kreiranje izvršne datoteke (EXE).

1. Kliknuti desnim tasterom miša na *Build Specification – New – Application (EXE)*, Sl. 8.23.

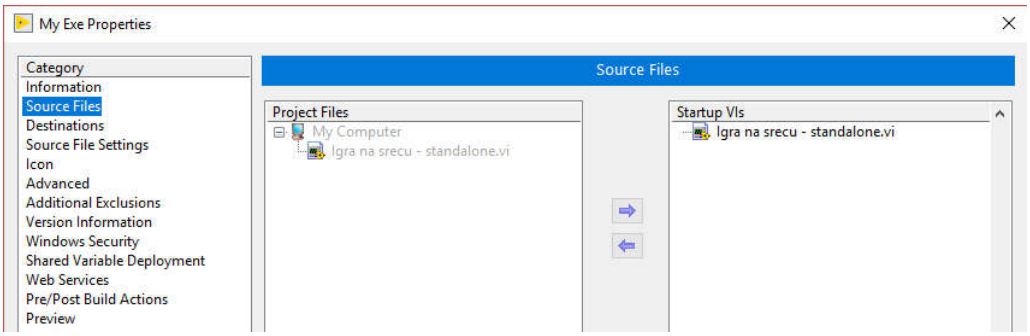


Sl. 8.23.

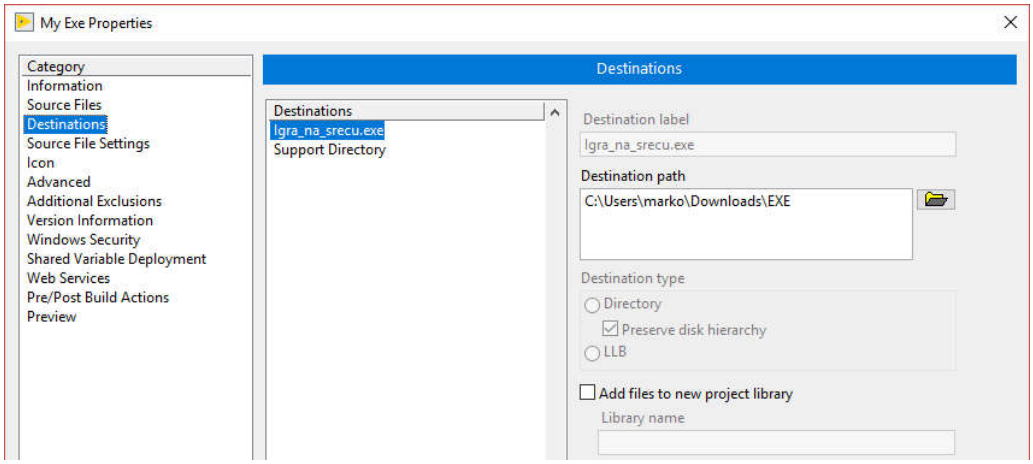
2. Podesiti polja slično kao na slikama od Sl. 8.24. do Sl. 8.26.



Sl. 8.24.



Sl. 8.25.



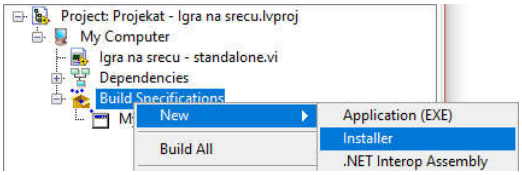
Sl. 8.26.

3. Aktivirati taster *Build*. Izaći iz *LabVIEW* okruženja. Pokrenuti dobijenu izvršnu datoteku (EXE) i proveriti da li se u istom direktorijumu gde i .exe datoteka nalazi i datoteka *evidencija.txt*.

Zadatak 8.7.

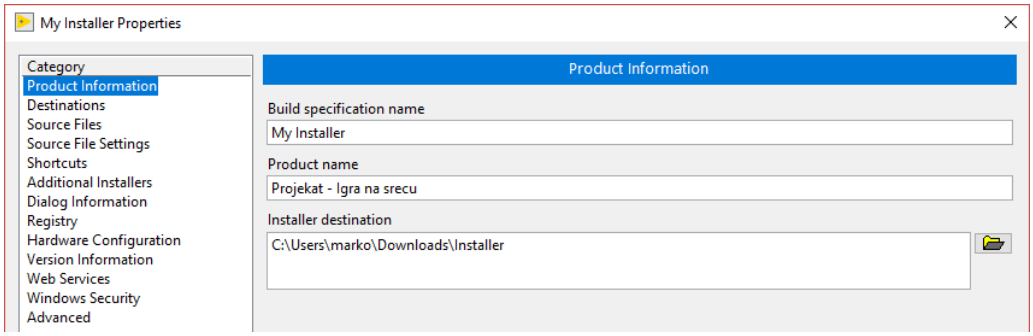
Kreiranje *installer*-a.

1. Kliknuti desnim tasterom miša na *Build Specification – New – Installer*, Sl. 8.27.

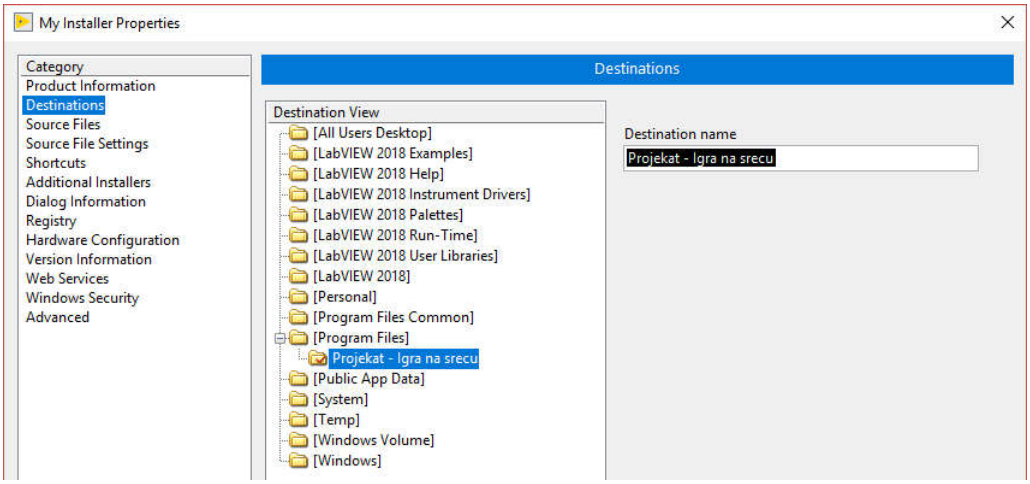


Sl. 8.27.

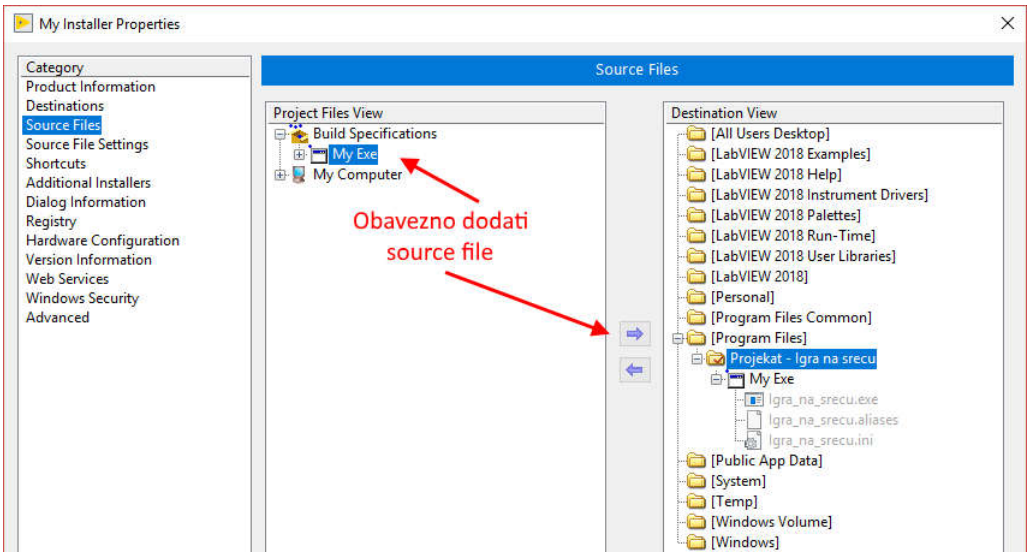
2. Podesiti polja slično kao na slikama od Sl. 8.28. do Sl. 8.30.



Sl. 8.28.

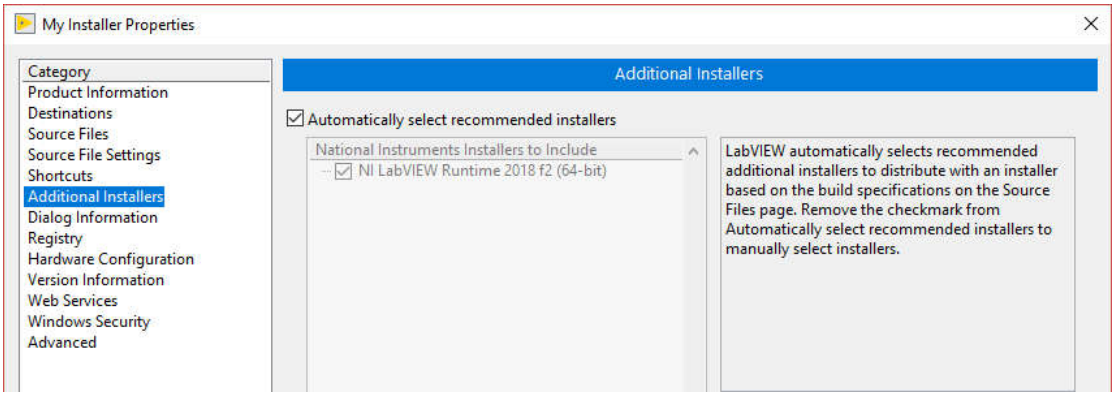


Sl. 8.29.



Sl. 8.30.

3. Ukoliko se koristi još neki modul za *LabVIEW*, kao na primer, *NI Vision Development*, potrebno je uključiti i *Run-Time* podršku (uključenje neophodnih izvršnih biblioteka), Sl. 8.31.



Sl. 8.31.

4. Na kraju potrebno je aktivirati taster *Build* kako bi započelo kreiranje *Installer-a*. Po okončanju postupka proveriti kolika je veličina dobijene datoteke. Ta veličina je posledica ugradnje svih potrebnih delova koje *LabVIEW* koristi. Bez njih izvršna datoteka (EXE) ne bi mogla da radi na računaru koji nema instaliran *LabVIEW*.

Lekcija 9 – Akvizicija slike i osnovne obrade slike

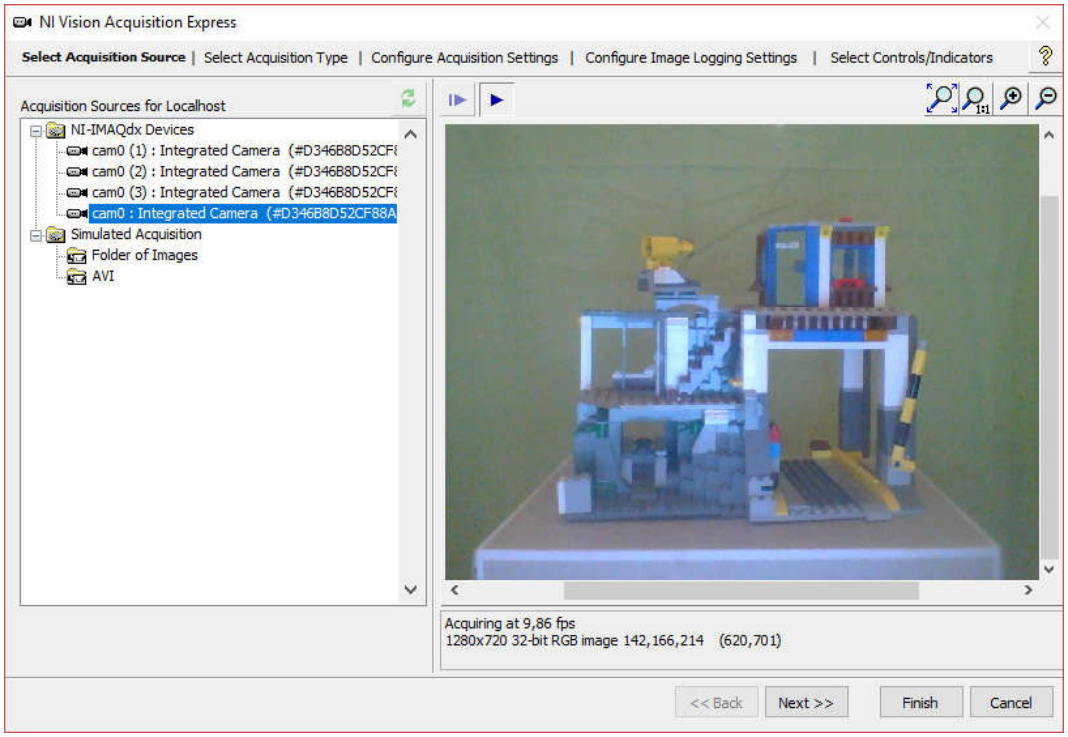
Cilj

Cilj vežbe je da studente upozna sa:

- Akvizicijom slike.
- Osnovnim koracima pri obradi slike

Zadatak 9.1.
Povezati USB kameru i kreirati kôd za akviziciju slike.

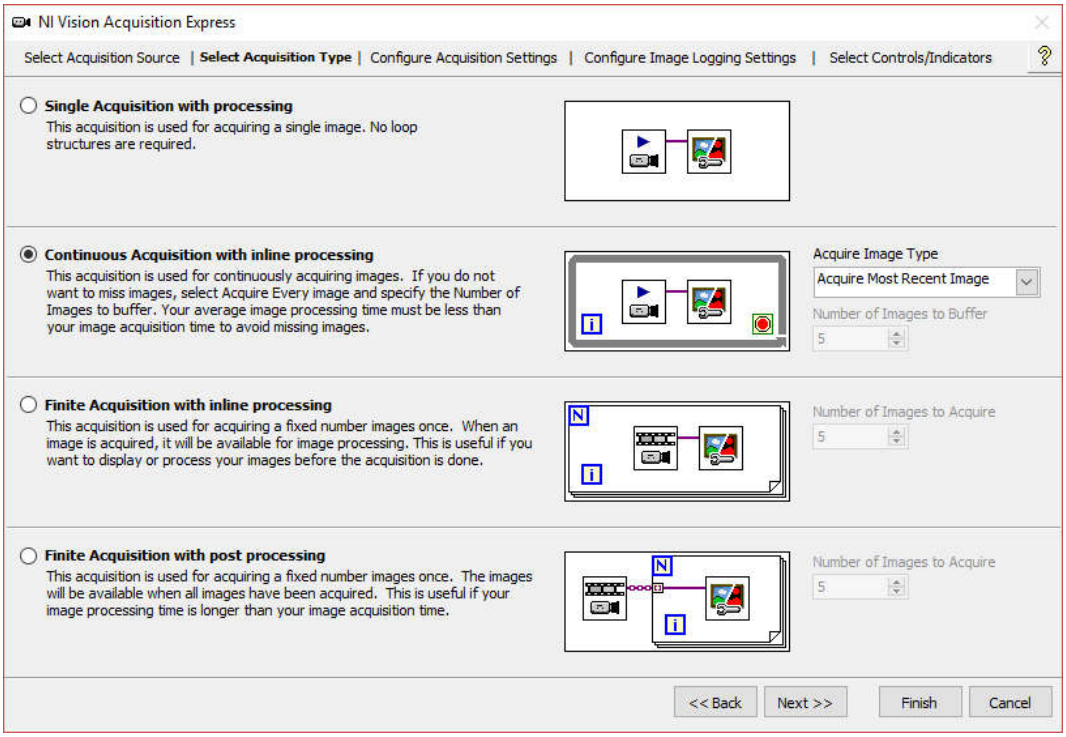
1. Kreirati novi VI. Dodati Vision Acquisition Express VI. Pokreće se čarobnjak koji sadrži 5 koraka. U prvom koraku bira se izvor slika (*Select Acquisition Source*), Sl. 9.1. Osim kamere izvor slika može biti direktorijum ili datoteka sa video sadržajem. Pri razvoju algoritma za obradu, uobičajeno je da se slike prvo snime pa da se nakon toga kreira algoritam za njihovu obradu. U tom slučaju se ponuđene opcije „izvora“ vezuju za direktorijum ili konkretnu datoteku.
2. Izabrati željeni izvor, u ovom slučaju cam0. Sa desne strane moguće je dobiti trenutnu sliku sa kamere aktivacijom taster *Play*.



Sl. 9.1.

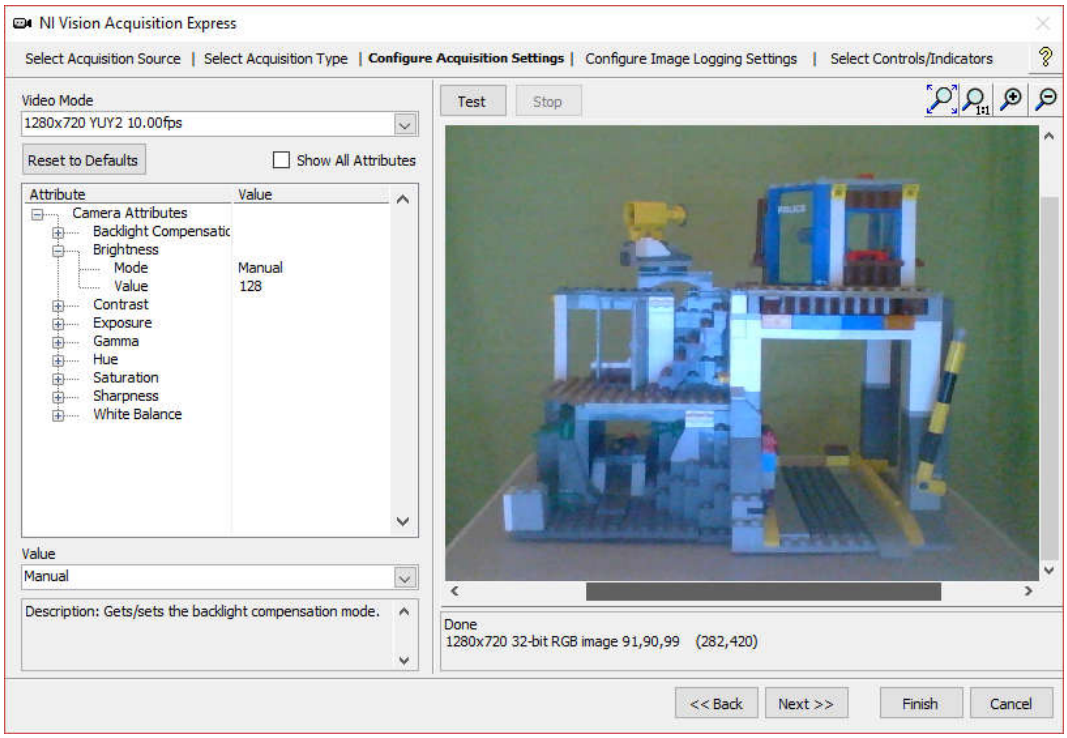
3. U sledećem koraku vrši se izbor tipa akvizicije (*Select Acquisition Type*), Sl. 9.2. Potrebno je izabrati drugu opciju *Continuous Acquisition with inline processing*.

To podrazumeva da se nakon akvizicije svake slike vrši i njena obrada i to predstavlja najčešći slučaj u većini realnih aplikacija.



Sl. 9.2.

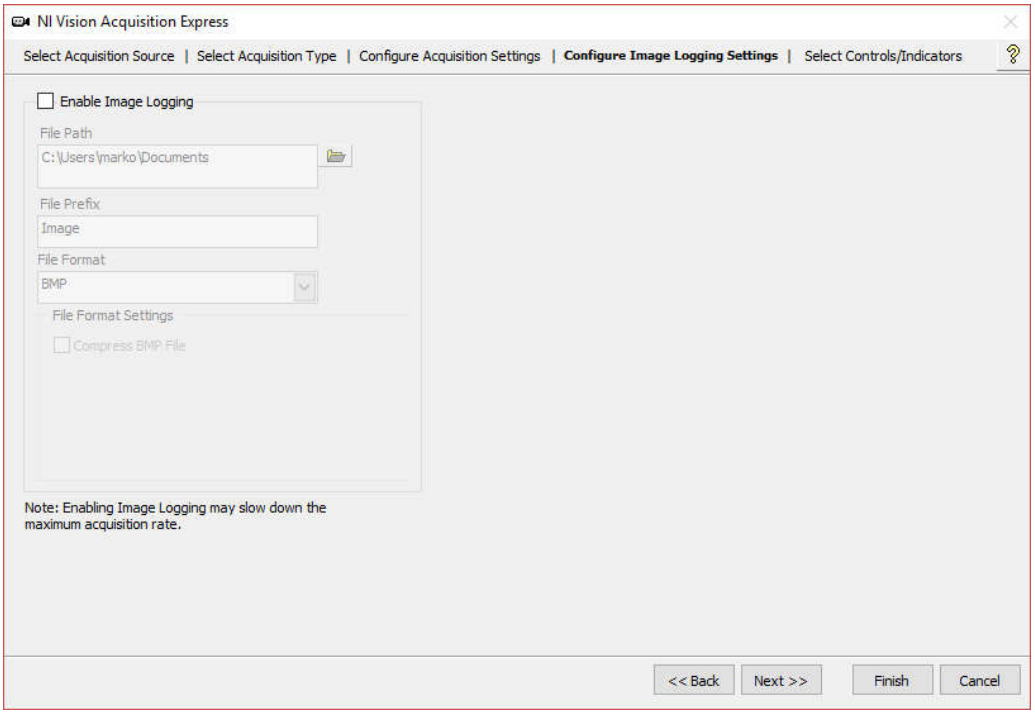
4. Sledeći korak je *Configure Acquisition Settings*, odnosno podešavanja parametara kamere, slično kao u MAX-u, Sl. 9.3. Kako se u primeru koristi web kamera, izbor parametara je ograničen i većina njih je u automatskom režimu podešavanja.



Sl. 9.3.

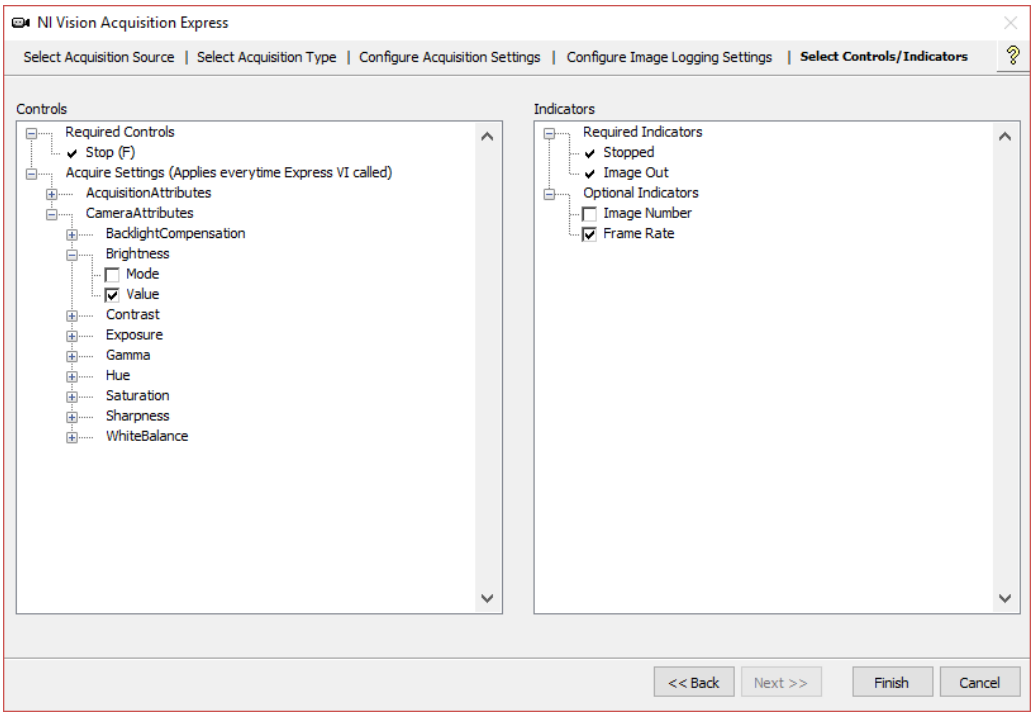
Za potrebe testiranja izabrati da je parametar *Brightness* u ručnom režimu i proveriti kako utiče njegova promena na osvetljenost slike. Potrebno je aktivirati akviziciju sa kamere pomoću tastera *Test*. Potom preći na sledeći korak.

5. U pretposlednjem koraku, Sl. 9.4, omogućava se snimanje svake slike ukoliko za to postoji potreba, u suprotnom prelazi se na poslednji korak, što je ovde i slučaj.



Sl. 9.4.

6. Poslednji korak omogućava izbor kontrola i indikatora. Kontrolama se može uticati na parametre akvizicije. U ovom slučaju, osim podrazumevanih (*Image Out* i *stop*) dodati kontrolu *Brightness Value* i indikator *Frame Rate*, Sl. 9.5.

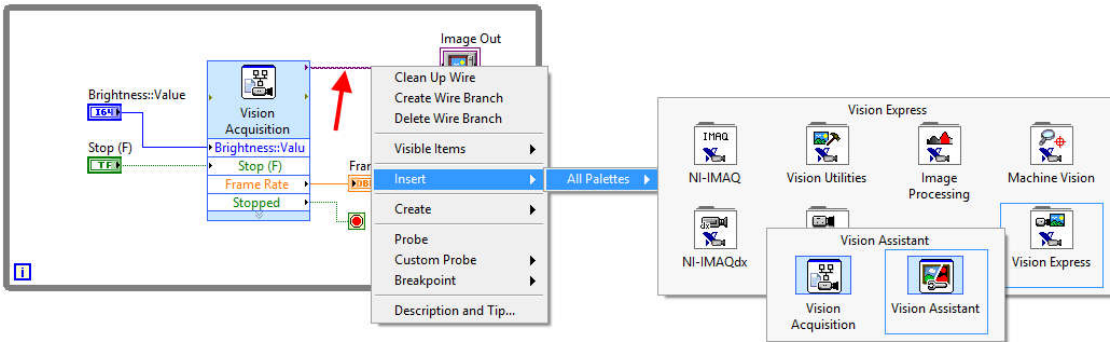


Sl. 9.5.

7. Na kraju aktivirati taster *Finish* i sačekati da se generiše kôd. Testirati dobijeni kôd i sačuvati aplikaciju pod imenom *Image Acq – start.VI*.

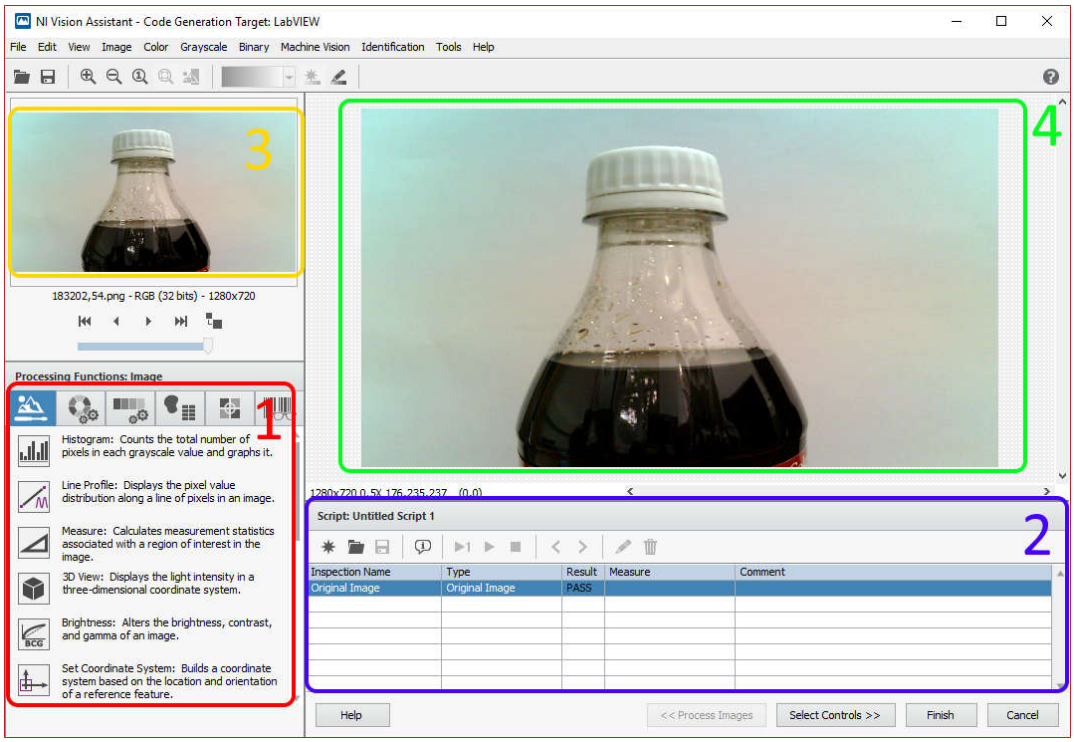
Zadatak 9.2.
Doraditi prethodni primer tako da se nakon akvizicije slike doda obrada slike koja se sastoji iz tri koraka: izdvajanje jedne kolor ravni, primerna Sobelovog filtra (formira se slika u nijansama sive u kojoj ivice imaju visoku vrednost, tj. svetlu boju) i množenje slike sa zadatim brojem.

1. Otvoriti VI "Image Acq - start". Pozicionirati kursor na žicu koja povezuje izlaz iz funkcije *Vision Acquisition* i displej *Image Out*, zatim kliknuti desnim tasterom miša i potom izabrati Express VI *Vision Assistant*, Sl. 9.6.



Sl. 9.6.

2. Pokreće se interfejs *Vision Assistant* u kome korisnik može da kreira algoritam za obradu slike, Sl. 9.7.

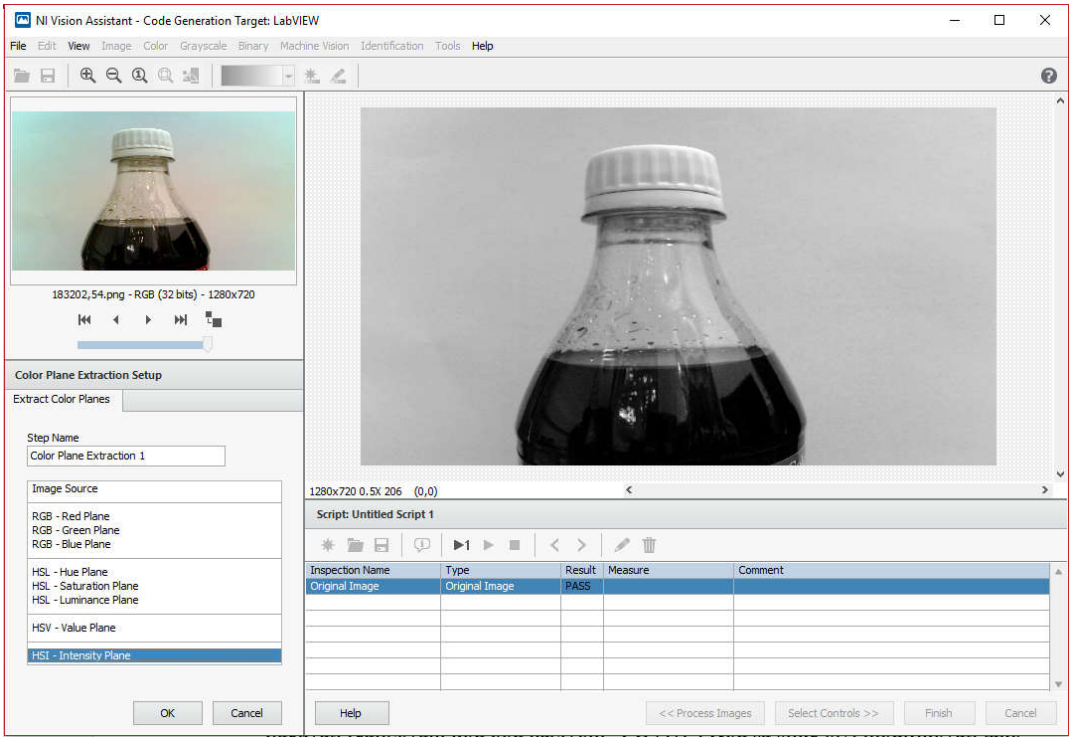


Sl. 9.7.

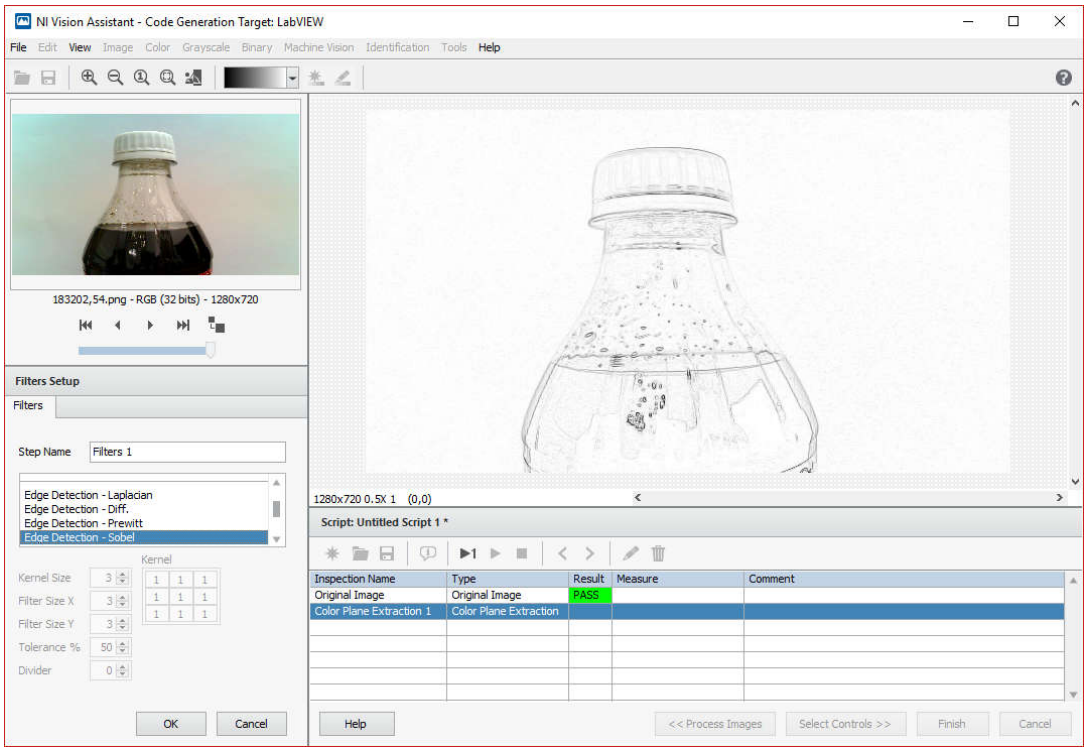
Ukratko objašnjenje delova *Vision Assistant*-a:
Pravougaonik 1 – nalazi se 6 tabova sa različitim funkcijama.
Pravougaonik 2 – koraci algoritma.

Pravougaonik 3 – rezultat trenutnog koraka algoritma za obradu slike.
Pravougaonik 4 – Omogućava pregled direktorijuma u kome se nalaze slike, a koji je određen za izvor slika.

3. Iz taba *Processing Functions: Color* izabrati funkciju *Color Plane Extraction* i tada će se otvoriti meni kao na slici, a zatim izabrati *Intensity Plane*, Sl. 9.8. Funkcija od kolor slike formira sliku u nijansama sive.

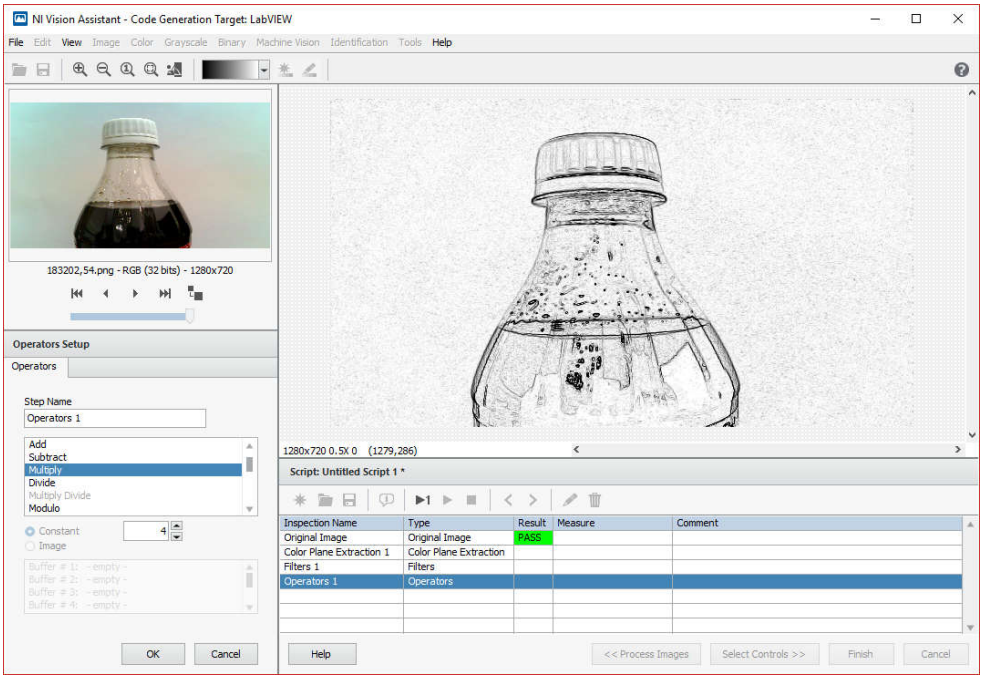


Sl. 9.8.

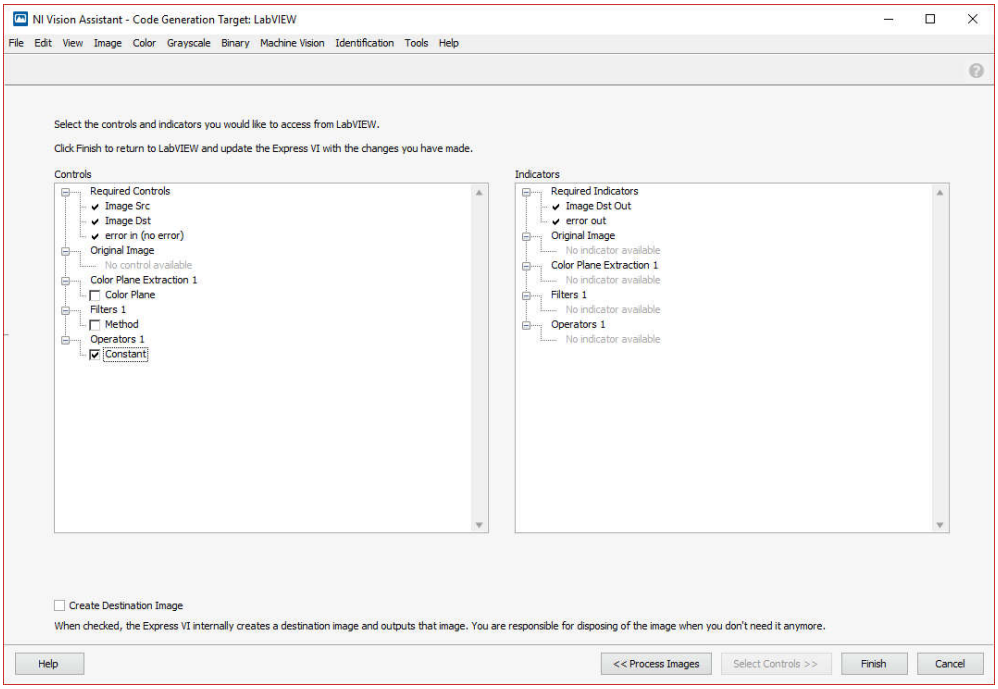


Sl. 9.10.

4. U sledećem koraku se formira gradijenta slika. Gradijenta slika predstavlja sliku koja nastaje primenom nekog oblika diferenciranja u x i y pravcu, a zatim se određuje kvaradtni koren sume kvadrata tog diferenciranja. Potrebno je sa trećeg taba *Processing Functions: Grayscale* izabrati funkciju *Filters* i u okviru nje algoritam *Edge detection – Sobel*, Sl. 9.9. Da bi lakše uočio rezultat, slika je invertovana i pojačan je kontrast, za razliku od stvarnog rezultata unutar *Vision Assistant*-a.
5. Da bi se lakše uočile ivice u gradijenoj slici, potrebno ju je pomnožiti brojem veći od jedan, odnosno dodaje se funkcija *Operators* sa trećeg taba *Processing Functions: Grayscale*, Sl. 9.11.

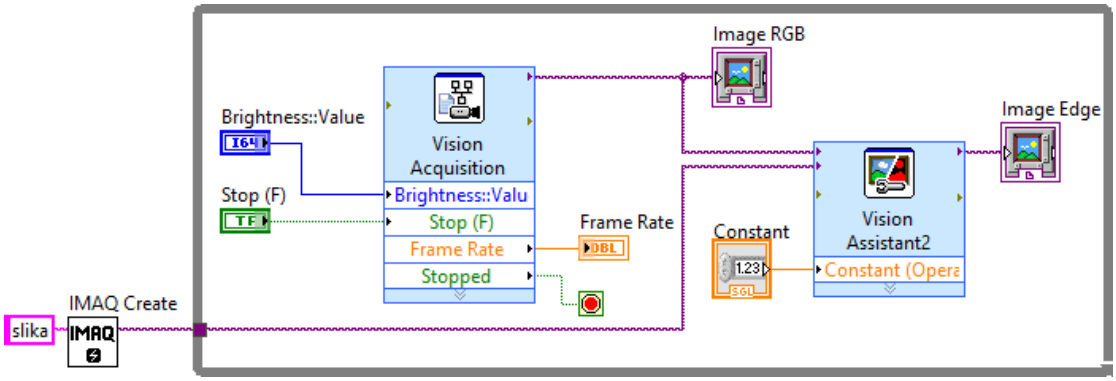


Sl. 9.11.



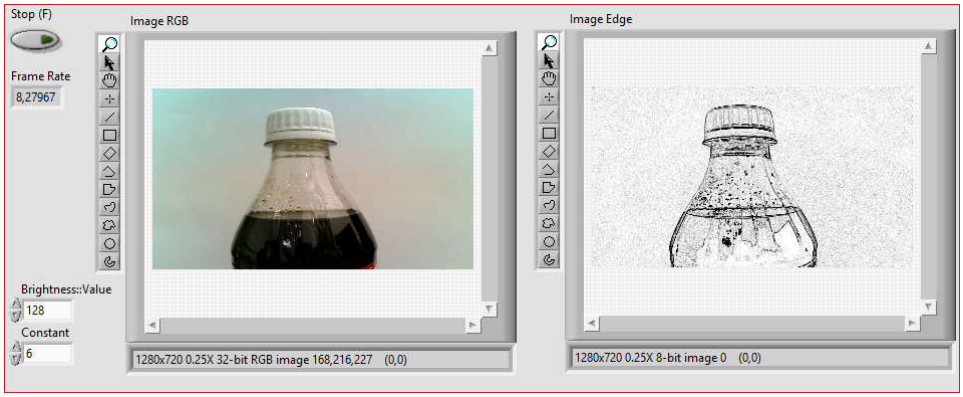
Sl. 9.12.

6. U finalnom koraku, pre aktiviranja tastera *Finish*, potrebno je izabrati kontrole i indikatore što se postiže pritiskom na taster *Select Controls*, kada se otvara meni kao Sl. 9.12. Potrebno je selektovati *Constant* za funkciju *Operators 1*.
7. Dobijeni blok dijagram predstavljen je na Sl. 9.13. Potrebno je dodati i funkciju *IMAQ Create* koja obezbeđuje dodatni bufer za sliku u koju se smešta rezultat funkcije *Vision Assistant2*.



Sl. 9.13.

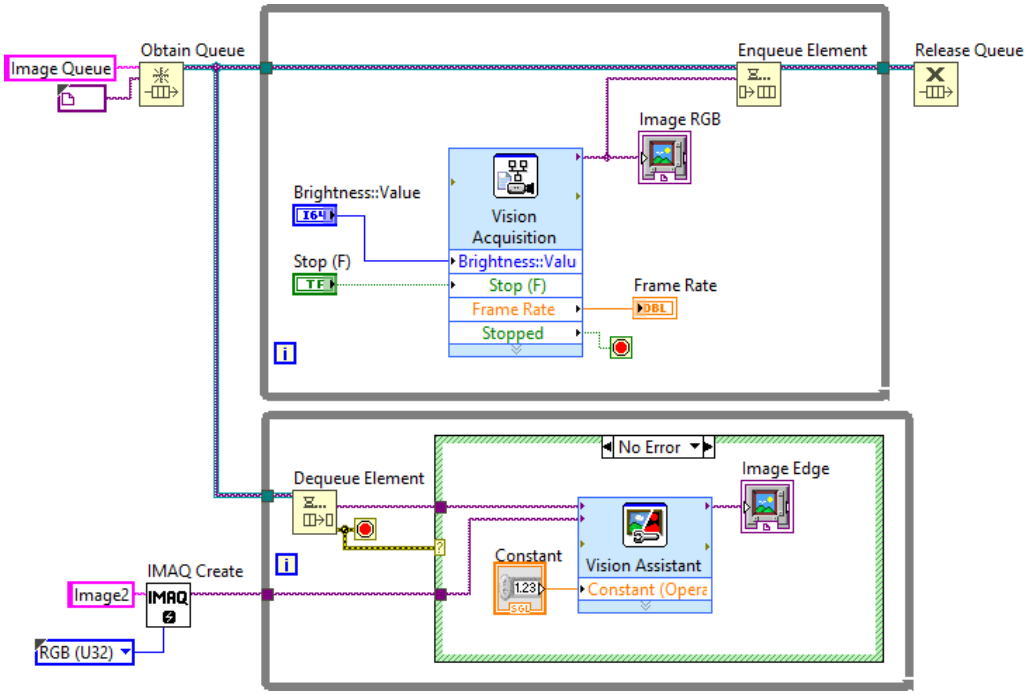
8. Realizovani *Front Panel* izgleda kao na Sl. 9.14. Snimiti VI pod nazivom *Image Acq - Edge in line processing.vi*.



Sl. 9.14.

Zadatak 9.3.
Prethodni zadatak prepraviti tako se akvizicija slika vrši u jednoj, obrada u drugoj petlji. Koristiti funkcije za rada sa redovima (*queue*) i *Producer/Consumer* šablon programiranja.

1. Otvoriti program *Image Acq - Edge in line processing.vi* i prepraviti da izgleda kao na Sl. 9.15 i sačuvati ga pod nazivom *Image Acq - Edge Producer-Consumer.vi*.

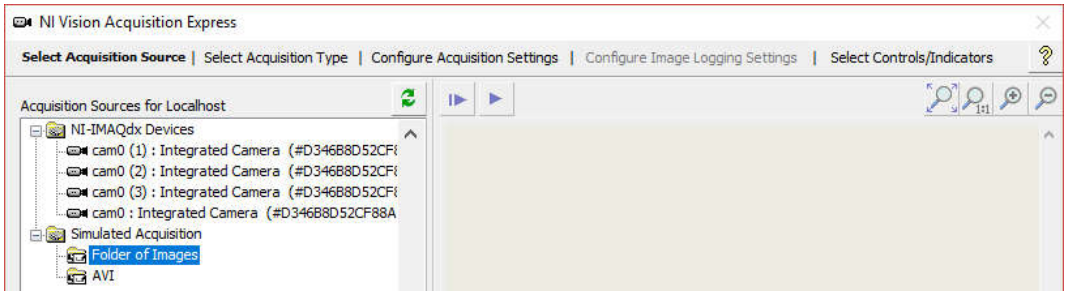


Sl. 9.15.

Zadatak 9.4.
Realizovati aplikaciju koja vrši ispitivanje da li je određeni automobilski osigurač prisutan. Zadovoljiti sledeće:

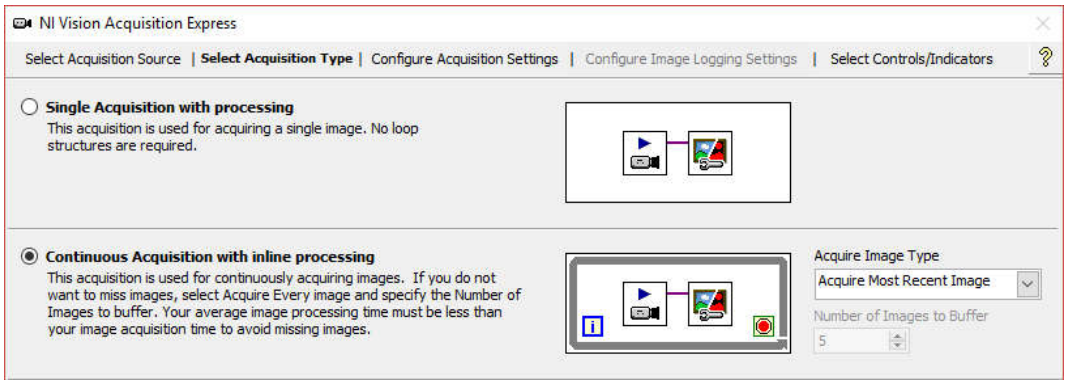
- Slike se nalaze u direktorijumu "Fusebox\Slike".
- Koristi se *color matching* funkcija da bi se izvršilo poređenje kolor informacije za poziciju osigurača sa očekivanom bojom osigurača za tu poziciju.
- Za posmatrani osigurač definiše se region u kome se očekuje da će biti pronađen.
- U ovom primeru posmatra se osigurač vrednosti 20 koji se nalazi u donjem redu na središnoj poziciji.

1. Kreirati novi VI i dodati *Vision Assistant Express* funkciju. Kada se pokrene meni za podešavanje kao izvor slika izabrati *Folder of Images*, Sl. 9.16, pa preći na sledeći korak.



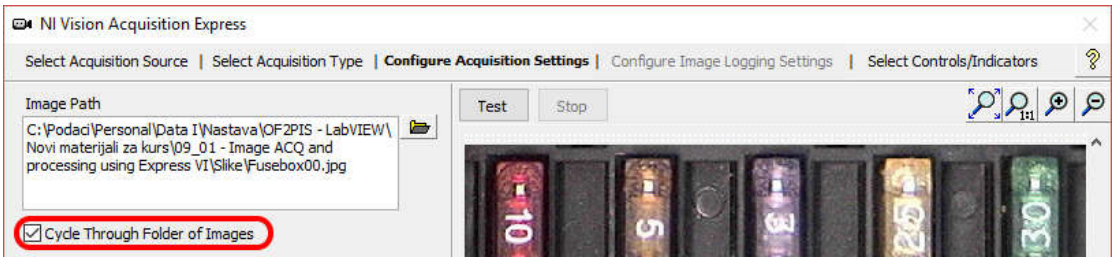
Sl. 9.16.

2. Za tip akvizicije izabrati *Continuous Acquisition with inline processing*, Sl. 9.17. Preći na sledeći korak.



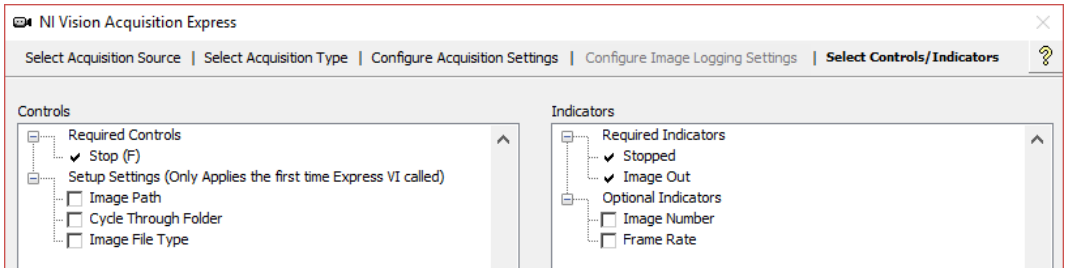
Sl. 9.17.

3. Sada je potrebno odrediti putanju gde se nalaze slike, a zatim aktivirati polje *Cycle Through Folder of Images*, kako bi omogućili da se slike ciklično menjaju, Sl. 9.18.



Sl. 9.18.

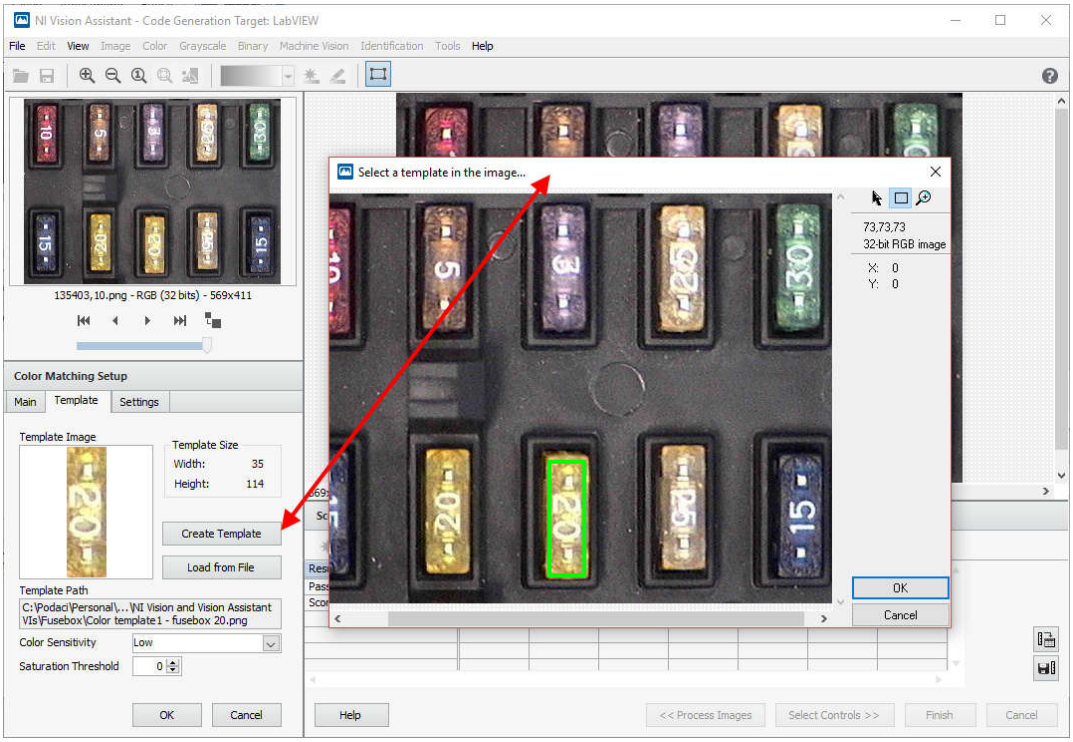
4. U poslednjem koraku mogu se izabrati kontrole i indikatori, ali u ovom slučaju nije potrebno ništa dodati, podrazumevane opcije su dovoljne, Sl. 9.19.



Sl. 9.19.

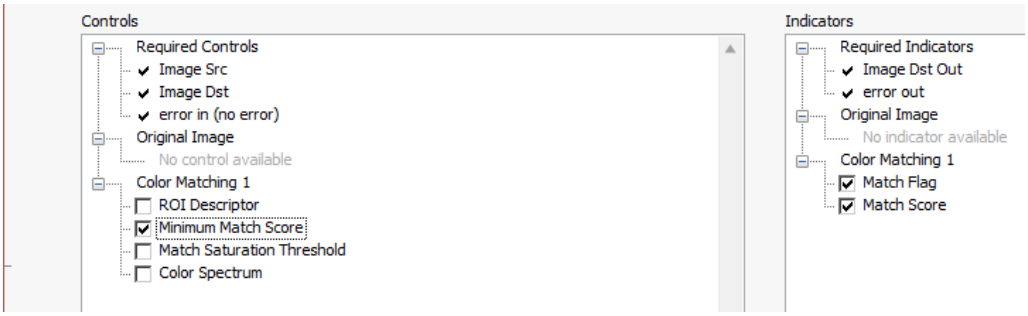
5. Sada se prelazi na deo za obradu slike načinom objašnjenim na Sl. 9.6, odnosno dodaje se *Vision Assistant Express VI*. Za ovaj primer potrebno je sa palete *Processing Functions: Color* samo izabrati funkciju *Color Matching*, Sl. 9.20.

Aktivirati taster *Create Template* i kada se otvori dodatni prozor za izbor dela slike koja treba da predstavlja šablon selektovati osigurač sa brojem 20 koji se nalazi u sredini donjeg reda, oivičen zelenom bolja. Nakon izbora i klika na taster OK otvara se dijalog za snimanje šablona, koji je potrebno snimiti pod nazivom *Color template1 - fusebox 20.png*.



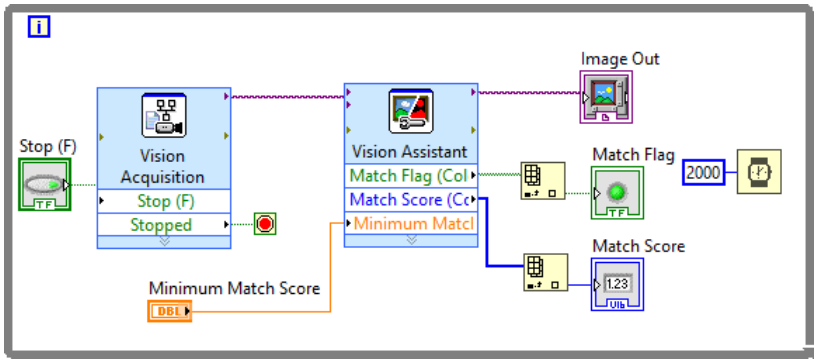
Sl. 9.20.

6. Preći na deo za izbor kontrola i indikatora, Sl. 9.21.



Sl. 9.21.

7. Izabrati kontrolu *Minimum Match Score* i indikatore *Match Flag* i *Match Score*.
8. Dodati elemente kao na Sl. 9.22. Snimiti aplikaciju pod nazivom *Vision Assistant Primer 1 – Color Matching.vi* i testirati je. Probati vrednost 950 za *Minimum Match Score*.



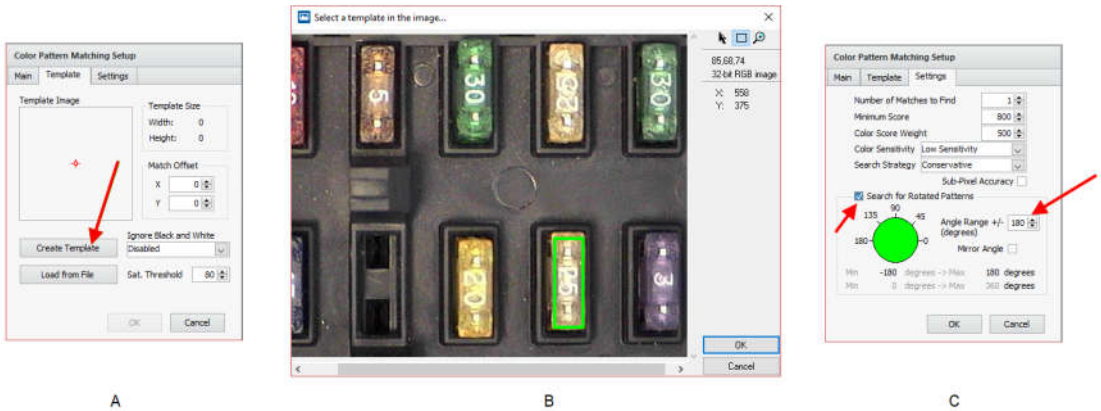
Sl. 9.22.

9. Dodatni zadaci: Omogućiti da se na sledeću sliku pređe kada korisnik aktivira taster *Next*. Obezbediti da se u drugoj petlji vrši obrada slika, tj. realizovati *Producer/Consumer Design Pattern*.

Zadatak 9.5.

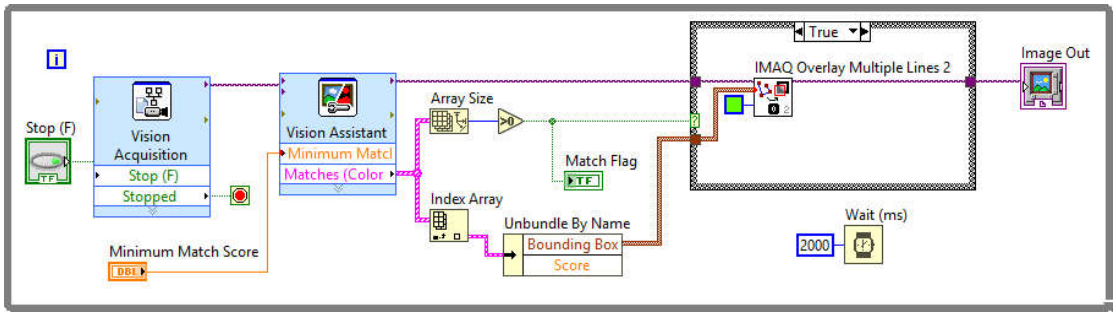
Poboljšati prethodni primer primenom *pattern matching* pristupa. U slučaju pomeranja postolja sa osiguračima ili promene orjentacije, funkcija *Color Matching* ne može da odrediti prisustvo osigurača. Napraviti aplikaciju koja određuje broj prisutnih osigurača vrednosti 25. Koristiti funkciju *Color Pattern Matching*. Slike se nalaze u direktorijumu "Fusebox\Slike dodatno".

1. Otvoriti primer *Vision Assistant Primer 1 – Color Matching.vi* i sačuvati ga pod nazivom *Vision Assistant Primer 1 – Color Pattern Matching.vi*. U *Vision Acquisition Express* zameniti putanje direktorijuma sa slikama. Otvoriti *Vision Assistant Express*. Obrisati funkciju *Color Matching* i zameniti je funkcijom *Color Pattern Matching*. Na drugom tabu funkcije izabrati *Create Template* (Sl. 9.23A) i izabrati deo slike koji predstavlja šablon – osigurač 25 u sredini donjeg dela slike, Sl. 9.23B. Aktivirati taster OK i sačuvati šablon pod nazivom "Color pattern template1 - fusebox 25.png". Na trećem tabu *Settings* aktivirati *Search for Rotated Patterns*, a za *Angle Range* uneti 180, Sl. 9.23C.



Sl. 9.23.

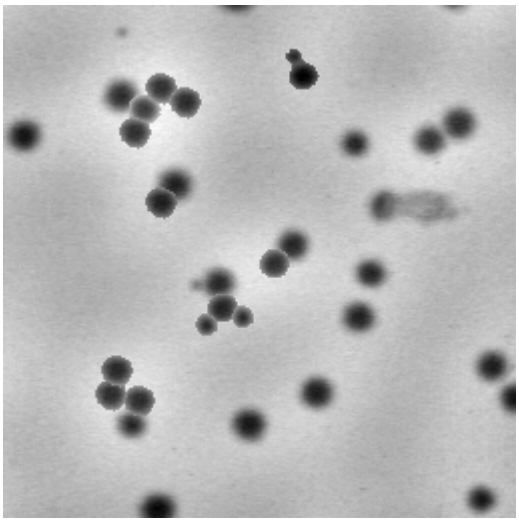
2. Dodati elemente kao na Sl. 9.24.



Sl. 9.24.

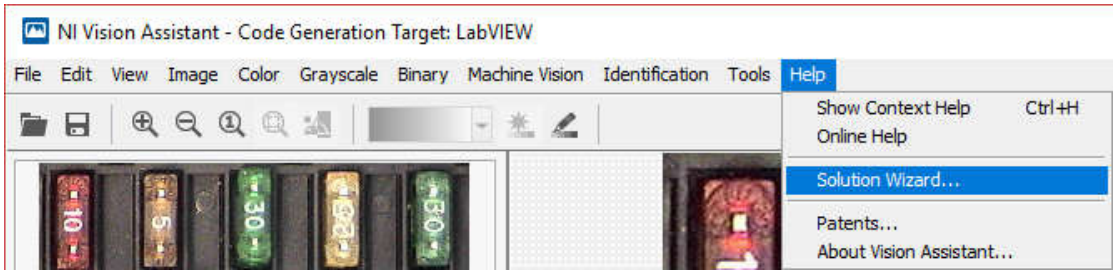
3. Dodatni zadatak: Omogućiti da se pronađe više od jednog šablona.

Zadatak 9.6.
Na Sl. 9.25 iscrtati konture oko svakog objekta pri čemu se objekti preklapaju. Slika se nalazi u direktorijumu *Culture* i postoji samo jedna slika.



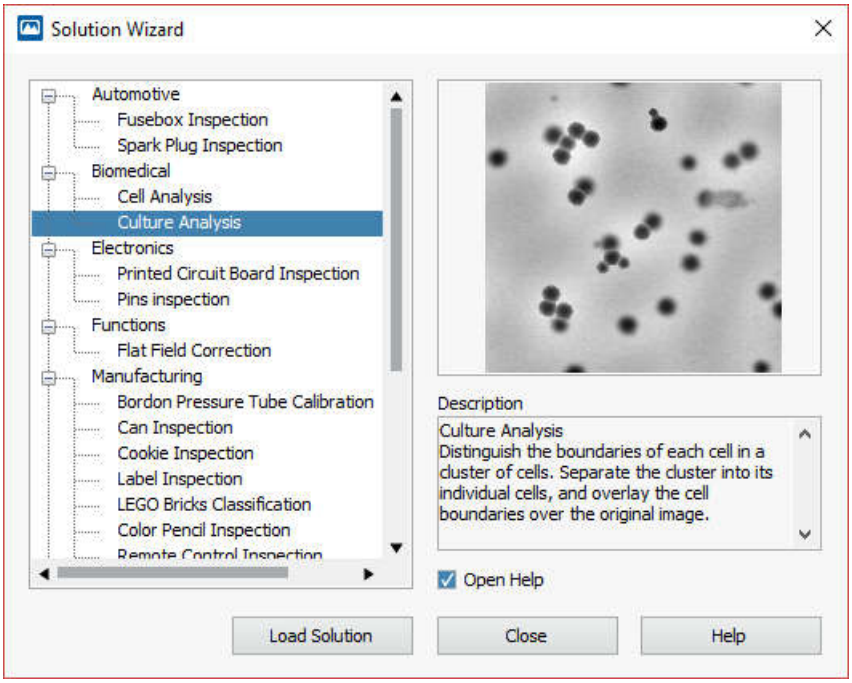
Sl. 9.25.

1. Otvoriti primer *Vision Assistant Primer 1 – Color Matching.vi* i sačuvati ga pod nazivom *Vision Assistant Primer 3 – Culture.vi*. U *Vision Acquisition Express* zameniti putanje direktorijuma sa slikama. Otvoriti *Vision Assistant Express*. Izabrati *Help/Solution Wizard…*, Sl. 9.26.



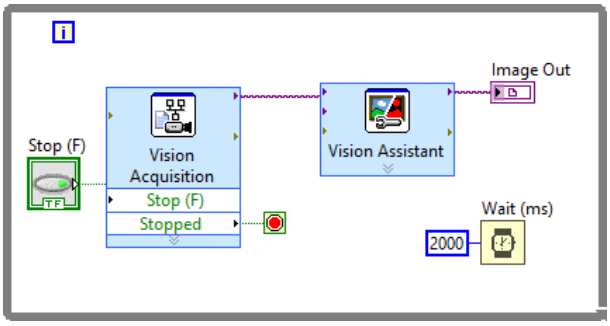
Sl. 9.26.

2. Izabrati *Biomedical – Culture Analysis*, Sl. 9.27.



Sl. 9.27.

3. Kada se skripta učita, proći kroz sve koraka algoritma, a potom aktivirati taster *Finish*.
4. Kreirati kôd kao na Sl. 9.28.



Sl. 9.28.

DEO II

Primena Arduino platforme i Python softverskog alata u merno-akvizicionim sistemima

Lekcija 10 – *Arduino* platforma. Akvizicija analognog napona i korišćenje digitalnih portova

Cilj

Cilj vežbe je da studente:

- upozna sa osnovnim funkcionalnostima *Arduino* platforme, prednostima koje ona pruža, kao i ograničenjima koja postavlja.
- osposobi za samostalno kreiranje osnovnih *Arduino* aplikacija koje uključuju akviziciju analognog napona sa senzora i komunikaciju platforme sa korisnikom korišćenjem računara ili eksternih tastera i displeja.
- uputi u korišćenje otvorenog hardvera, podižući na taj način svest o *Open Source* rešenjima u modernoj nauci i inženjerstvu.

Oprema

- Računar sa instaliranim ***Arduino IDE*** softverskim okruženjem.
- ***Arduino UNO R3*** ploča.
- Protobord, kratkospojnici, otpornici (150 Ω, 220 Ω, 10 KΩ, 100 KΩ), NTC termistor, svetleća dioda, TCRT5000 reflektivni optički senzor, kapacitivni senzor dodira, četvorocifreni sedmosegmentni displej sa TM1637 čipom.

NAPOMENA: Najnoviju verziju *Arduino IDE* okruženja, kao i dokumentaciju vezanu za platformu je moguće naći na zvaničnom *Arduino* sajtu:
https://www.arduino.cc/

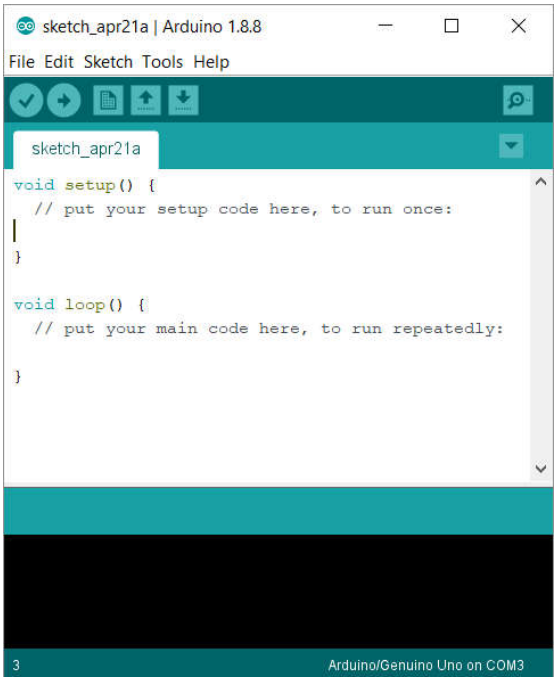
10.1 Arduino IDE okruženje i *Arduino UNO R3*

Arduino IDE okruženje se pokreće izborom opcije: ***Start»All programs»Arduino*** ili klikom na ***Arduino*** prečicu na *Desktop*-u. Inicijalni *Arduino* prozor je prikazan na Sl. 10.1.1.

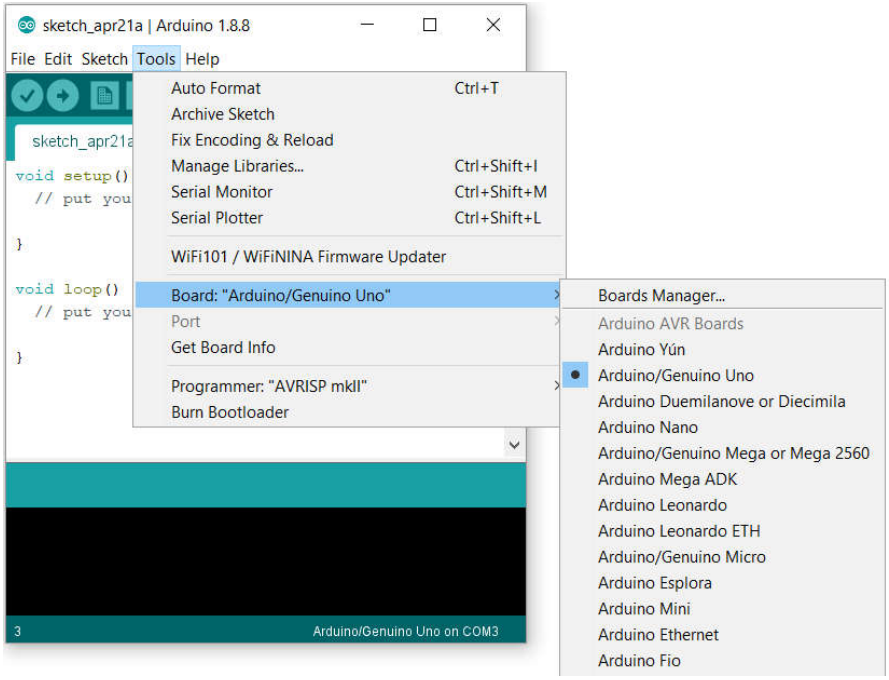
NAPOMENA*:* Uočiti da svaki *Arduino* program sadrži dve celine:

- setup: celina u kojoj se upisuje kôd za inicijalizaciju;
- loop: glavni deo kôda se piše u ovoj beskonačnoj petlji.

1. Povezati *Arduino UNO* sa računarom korišćenjem USB kabla. Kako *UNO* nije jedina ploča iz *Arduino* porodice, potrebno je u programu definisati za koju od dostupnih ploča se piše kôd. Izbor ploče je omogućen unutar *Tools* padajućeg menija kao na Sl. 10.1.2 gde je potrebno definisati da se kôd spušta na *Arduino/Genuino UNO* ploču.

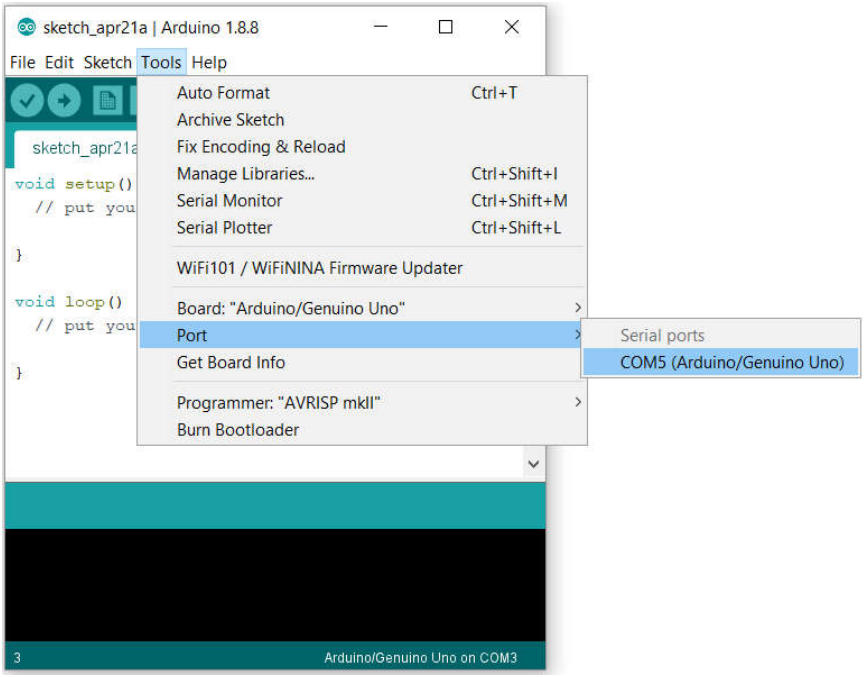


Sl. 10.1.1. Početni ekran *Arudino IDE* okruženja



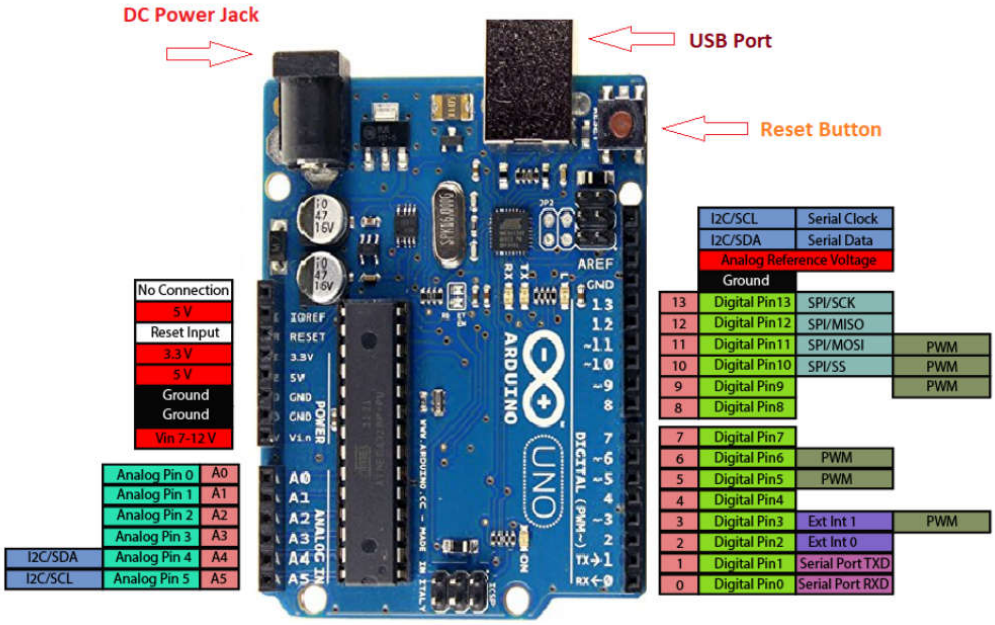
Sl. 10.1.2. Izbor ploče na koju se spušta kôd

2. Takođe je potrebno definisati na kom COM portu je priključen *Arduino*. Najpre je potrebno u *Device manager*-u proveriti broj COM porta na koji je *Arduino* priključen, nakon čega je potrebno izvršiti selekciju porta u *Arduino IDE* programu kao na Sl. 10.1.3 čime se završava inicijalno podešavanje programa.



Sl. 10.1.3. Izbor COM porta računara na koji je povezan *Arduino*

NAPOMENA: Pinovi *Arduino UNO* ploče se mogu razvrstati u tri sekcije: POWER, ANALOG IN i DIGITAL prema funkciji koju obavljaju. *Pinout Arduino UNO R3* ploče je prikazan na Sl. 10.1.4.



Sl. 10.1.4. *Pinout* dijagram *Arduino UNO R3* ploče

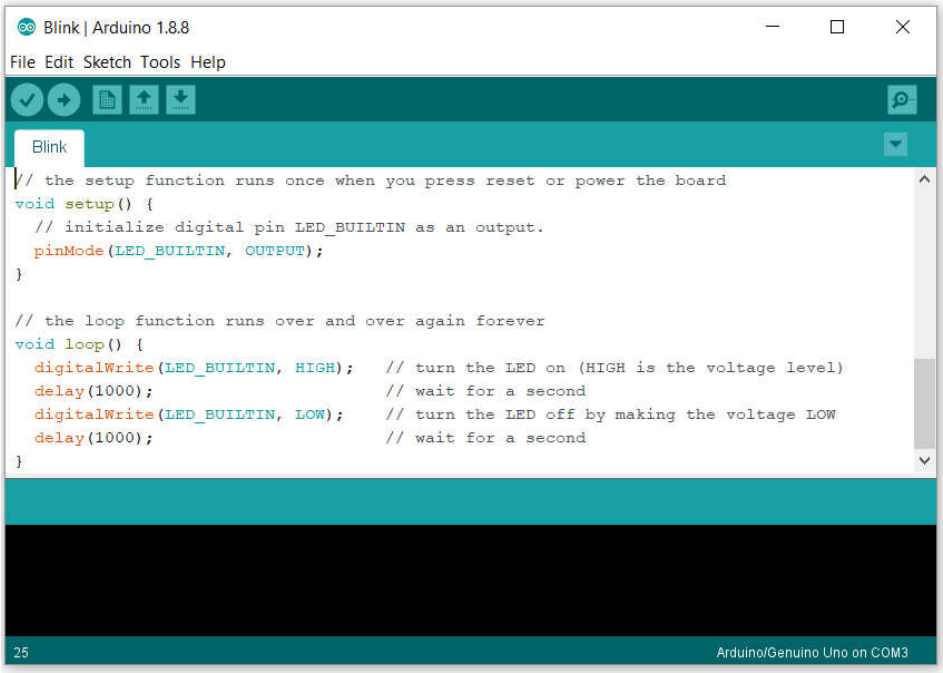
POWER sekciju čine pinovi koji se koriste za napajanje projekata (GND, 5 V i 3.3 V), kao i za dovođenje eksternog napajanja u opsegu 7-12 V (VIN + GND ili DC Power Jack).

ANALOG IN sekciju čine pinovi koji su povezani na A/D konvertor koji se nalazi u samom mikrokontroleru (naziv mikrokontrolera: *Atmel Atmega 328p*) na kojima je omogućeno čitanje analognog napona.

DIGITAL sekciju čine pinovi koji se mogu ponašati kao digitalni ulazi ili izlazi.

10.2 Digitalni ulazi i izlazi (Digital I/O)

1. *Arduino IDE* okruženje dolazi sa velikim brojem ugrađenih primera kojima se pristupa pritiskom na *File»Examples*. Pokrenuti *Blink* primer pritiskom na *File»Examples»01.Basics»Blink*. Otvoriće se prozor sa kôdom kao na Sl. 10.2.1. Ovaj primer demonstrira kontrolu svetleće diode koja je ugrađena na ploči i po potrebi se može koristiti kao indikator. Ova svetleća dioda je povezana na digitalni pin D13 *Arduino UNO* ploče koji se može pozvati i korišćenjem definisane konstante LED_BUILTIN.



Sl. 10.2.1. *Blink* primer za kontrolu ugrađene svetleće diode

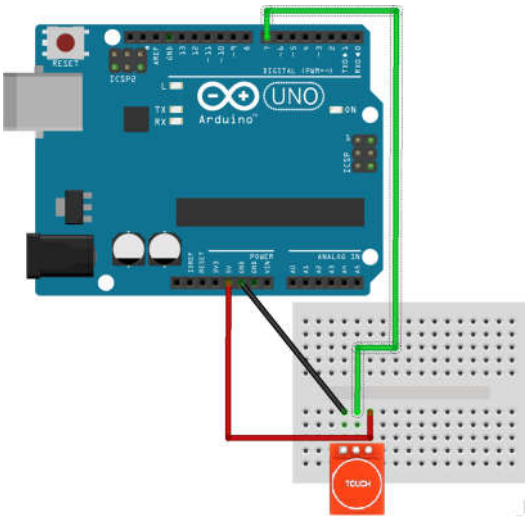
---

pinMode(pin,mode): služi za podešavanje digitalnog pina (u ovom primeru je pin podešen kao izlazni - OUTPUT).
digitalWrite(pin,value): služi za ispisivanje visoke (HIGH tj. 5V) ili niske (LOW tj. 0V) vrednosti na digitalnom izlazu.
delay(ms): zaustavlja izvršavanje programa za definisani broj milisekundi.

---

2. Spustiti kôd na ploču pritiskom na dugme *Upload*  . Posmatrati ploču i uočiti ponašanje svetleće diode obeležene sa L.
   NAPOMENA: Nekada je potrebno izvršti *Compiling* kôda bez spuštanja kôda na ploču. Ova opcija je omogućena pritiskom na dugme *Verify*  .
3. Sačuvati prethodni program pritiskom na *File»Save as* pod nazivom *Blink_test.ino*, a zatim taj, novosačuvani program modifikovati tako svetleća dioda svake sekunde bude uključena 200 ms.
4. **Isključiti** *Arduino* iz računara i povezati kapacitivni senzor dodira korišćenjem kratkospojnika i protoborda kao na Sl. 10.2.2. Zadatak je napraviti program koji će detektovati dodir na senzoru i dok je senzor dodirnut držati uključenu ugrađenu svetleću diodu na ploči. **Voditi računa** o

povezivanju napajanja kako se senzor ne bi bio uništen inverznom polarizacijom.



Sl. 10.2.2. Šema povezivanja kapacitivnog senzora dodira

5. Kreirati novi *Sketch* pritiskom na *File»New* i sačuvati ga pod zazivom *TouchLED.ino*. U programu najpre definisati dve nove promenljive *int* tipa pod nazivom touchPin i touchVal pre *setup* sekcije kôda. Njihove vrednosti inicijalizovati na 7 i 0 respektivno. Promenljiva touchPin označava broj digitalnog pina na koji je povezan senzor dodira, dok će promenljiva touchVal služiti za upisivanje pročitane vrednosti na touchPin digitalnom pinu.
   Definisanje promenljive: tip naziv = vrednost; (int touchPin = 7;)
6. U okviru setup sekcije korišćenjem funkcije pinMode() inicijalizovati pin LED_BUILTIN na kome se nalazi ugrađena svetleća dioda kao izlazni (OUTPUT) i touchPin kao ulazni pin (INPUT).
7. Unutar beskonačne petlje *loop* je najpre potrebno pročitati vrednost na digitalnom pinu touchPin i upisati je u promenljivu touchVal. Nakon toga je potrebno, korišćenjem *if* funkcije proveriti da li je vrednost jednaka 1 (senzor je dodirnut) ili 0 (senzor nije dodirnut). Potrebno je obezbediti da ugrađena svetleća dioda svetli samo ukoliko je senzor dodirnut. Korišćenjem delay() funkcije se obezbeđuje da se provera vrednosti pina odvija na ***približno*** 50 ms. Gotov kôd je prikazan na Sl. 10.2.3.

```
int touchPin = 7;
int touchVal = 0;

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(touchPin, INPUT);
}
void loop() {
  touchVal = digitalRead(touchPin);
  if (touchVal == 1) {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else {
    digitalWrite(LED_BUILTIN, LOW);
  }
  delay(50);
}
```

Sl. 10.2.3. Program za čitanje sa kapacitivnog senzora dodira i uključivanje svetleće diode

digitalRead(pin): služi za čitanje sa digitalnog ulaza. Može vratiti vrednosti 1 ili 0 u zavisnosti od toga da li je ulazu pročitana visoka ili niska vrednost napona respektivno.

Povezati *Arduino* sa računarom i prebaciti kôd na ploču pritiskom na dugme *Upload*. Dodirnuti kapacitivni senzor dodira i posmatrati svetleću diodu L na *Arduino UNO* ploči. Zatim bez dodirivanja približiti prst blizu senzora. Objasniti pojavu koja se uočava!

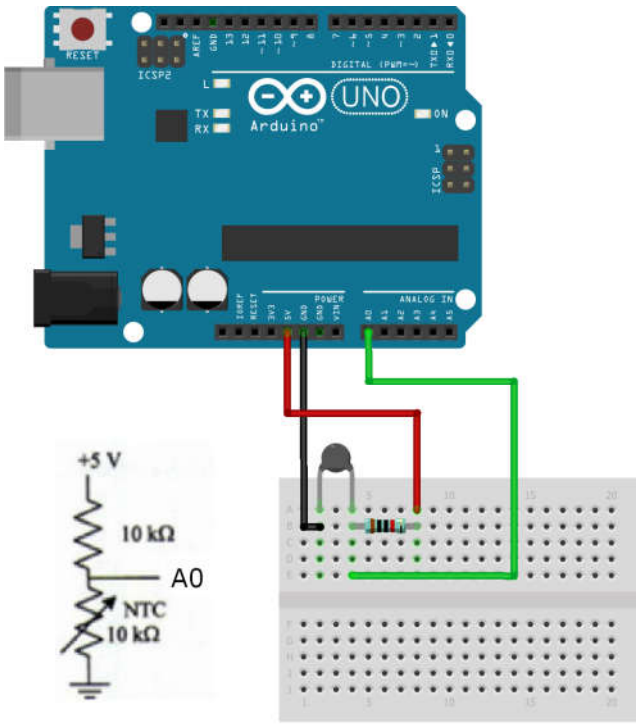
8. **Isključiti** *Arduino* iz računara i razvezati kolo.

10.3 Analogna akvizicija napona i serijska komunikacija sa računarom

1. Povezati razdelnik napona sa NTC termistorom na protobordu prema šemi sa Sl. 10.3.1.
2. Otvoriti primer *AnalogReadSerial* koji se nalazi u sekciji *File»Examples»01.Basics* i sačuvati ga kao novi *Sketch* pod nazivom *AnalogReadSerial_NTC.ino*.

Serial.begin(): inicijalizuje komunikaciju između *Arduino* ploče i računara (ili nekog drugog uređaja koji je sposoban za serijsku komunikaciju sa *Arduino* pločom). Argument funkcije je tzv. *baud rate* – broj bita po sekundi (najčešće 9600).
analogRead(pin): vraća rezultat A/D konverzije sa preciziranog analognog ulaznog pina. Kako A/D konvertor na *Arduino* UNO R3 ploči ima rezoluciju od 10 bit-a ova funkcija vraća *integer* vrednosti od 0 do 1023 koje predstavljaju nivoe celokupnog ulaznog naponskog opsega. U opštem slučaju,1024 nivoa se raspoređuju na opseg od 0 do 5V. Kolika je tada rezolucija A/D konverzije? Analogni pin nije potrebno inicijalizovati već je njegovo podrazumevano stanje da je ulazni.
Serial.print(): ispisuje zadatu vrednost na serijskom portu kako bi je računar pročitao. Dodavanjem sufiksa ln se obezbeđuje i prelazak u novi red nakon poslate vrednosti.

Za one koji žele da znaju više: Opseg napona funkcije analogRead() je moguće i smanjiti kako bi se povećala rezolucija u manjem opsegu ulaznog napona korišćenjem funkcije analogReference(type).



Sl. 10.3.1. Povezivanje naponskog razdelnika sa NTC termistorom na *Arduino*

3. Modifikovati kôd tako da *delay()* funkcija zaustavlja izvršavanje programa na 10 ms.
4. U petlji dodati i promenljivu voltage tipa *float* u kojoj će se upisivati vrednost sa A/D konvertora skalirana na opseg od 0 do 5V na taj način obezbeđujući ispisivanje realnih vrednosti napona, a ne naponskih nivoa od 0 do 1023.

5. Umesto ispisivanja promenljive sensorValue, za argument funkcije *Serial.println()* uzeti promenljivu voltage. Konačan izgled programskog kôda je prikazan na Sl. 10.3.2.

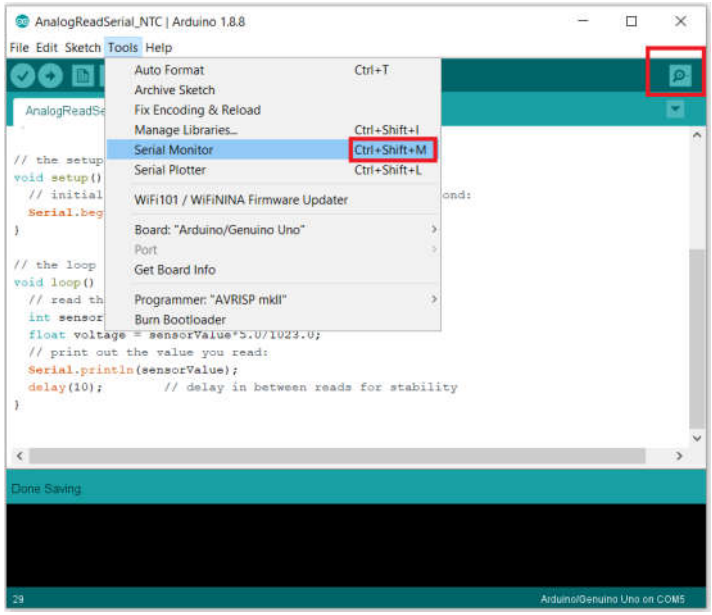
```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  float voltage = sensorValue*5.0/1023.0;
  // print out the value you read:
  Serial.println(voltage);
  delay(10);        // delay in between reads for stability
}
```

Sl. 10.3.2. Kôd za analognu akviziciju i ispisivanje vrednosti na serijskom portu

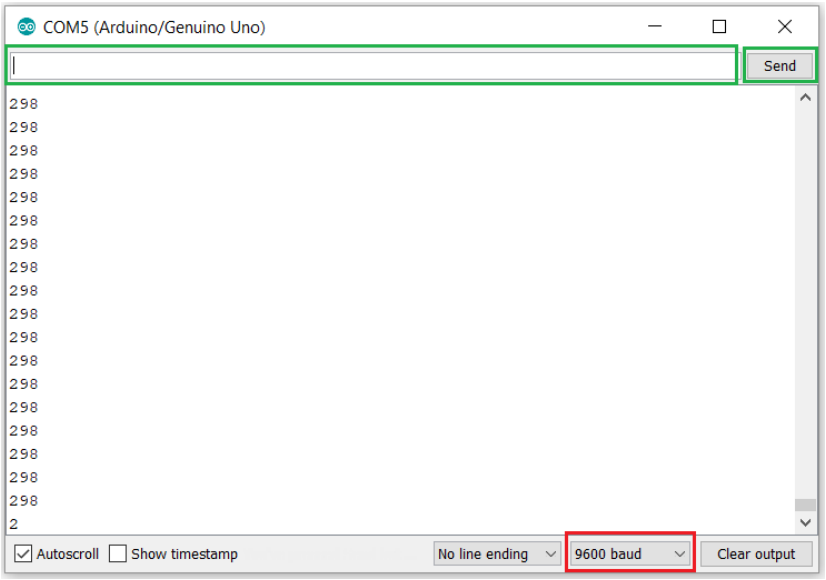
6. Sačuvati promene u programu.
7. Povezati *Arduino* sa računarom i spustiti kôd na ploču. Na ploči obratiti pažnju na svetleće diode obeležene sa RX i TX koje signaliziraju serijsku komunikaciju *Arduino* ploče sa računarom. Objasniti njihovo trenutno stanje!

NAPOMENA: Različiti programski paketi kao što su *LabVIEW*, *Python* ili *MATLAB* poseduju set funkcija koje omogućavaju računaru da prikuplja nadolazeće podatke na serijskom portu i prikazuje ih. *Arduino IDE* takođe omogućava ispisivanje podataka poslatih sa *Arduino* ploče korišćenjem *Serial Monitor* opcije.

8. U prozoru *Arduino IDE* okruženja pokrenuti *Serial Monitor* izborom *Tools»Serial Monitor*, prečicom Ctrl+Shift+M ili pritiskom na ikonicu u gornjem desnom uglu prozora, kao na Sl. 10.3.3. Otvoriće se prozor u kome će se ispisivati vrednosti napona koje se šalju sa *Arduino* ploče kao na Sl. 10.3.4.



Sl. 10.3.3. Uključivanje *Serial Monitor* prozora



Sl. 10.3.4. Izgled Serial Monitor prozora. Opcije za slanje poruka su uokvirene zelenim pravougaonikom. *Baud rate* podešavanje je uokviireno crvenim pravougaonikom.

U ovom prozoru je takođe moguće slati podatke *Arduino* ploči korišćenjem *text box*-a u vrhu prozora i pritiskom na dugme *Send*. Kako trenutni kôd koji se nalazi na ploči ne vrši čitanje na serijskom portu, slanje poruka ploči nije od interesa za rad programa. Ipak, interesantno je poslati poruku proizvoljne sadržine *Arduino* ploči i tokom slanja posmatrati RX i TX svetleće diode. Šta je sada moguće uočiti?

NAPOMENA: Obratiti pažnju na *Baud rate* podešavanje u okviru *Serial Monitor* prozora. Kako bi poruke bile protumačene ispravno, izabrana vrednost se mora poklopiti sa vrednošću koja je pozvana kao argument funkcije *Serial.begin*().

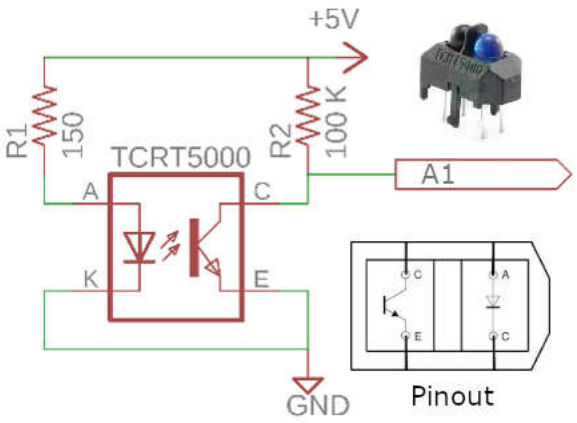
9. Promeniti vrednost za *Baud rate* u prozoru *Serial Monitor*-a i uočiti šta se dešava sa sadržajem poruke. Vratiti vrednost na 9600.

Često se *Arduino* koristi kao uređaj za akviziciju naponskih signala te je od interesa iscrtati te signale u realnom vremenu. U okviru *Arduino IDE* okruženja je iscrtavanje vrednosti koje *Arduino* šalje na serijski port omogućeno korišćenjem *Serial Plotter* opcije.

10. Zatvoriti *Serial Monitor* prozor. Otvoriti *Serial Plotter* izborom *Tools»Serial Plotter* ili prečicom Ctrl+Shift+L. Obratiti pažnju da *Serial Monitor* i *Serial Plotter* ne mogu biti otvoreni u isto vreme!
11. Posmatrati promenu dobijenog signala na ekranu sa promenom temperature NTC termistora.
12. **Isključiti** *Arduino* iz računara. Kolo **ostaviti** sastavljeno!

10.4 Analogna akvizicija i prikazivanje signala sa više senzora

1. Na protobordu **sastaviti** šemu sa Sl. 10.4.1.



Sl. 10.4.1. Šema za povezivanje TCRT5000 refleksivnog optičkog senzora

TCRT5000 je jednostavan primer refleksivnog optičkog senzora rastojanja. Čine ga svetleća dioda koja emituje i fototranzistor koji detektuje svetlost u IC delu spektra (svetlost se ne vidi golim okom). Ukoliko se prepreka nalazi ispred senzora, deo svetlosti koji se emituje sa svetleće diode se reflektuje nazad ka fototranzistoru gde biva detektovan. Što je prepreka bliža, više svetlosti se reflektuje ka fototranzistoru, i njegova struja raste!

2. Kolektor fototranzistora povezati na A1 analogni ulaz *Arduino* ploče.
3. Modifikovati program iz prethodne tačke ponavljanjem odgovarajućih funkcija kako bi se obezbedilo čitanje dve naponske veličine sa analognih ulaza A0 i A1. Vrednosti sa analognih ulaza ispisivati u istom redu, razdvojene *TAB* konstantom.

Modifikovani program je prikazan na Sl. 10.4.2.

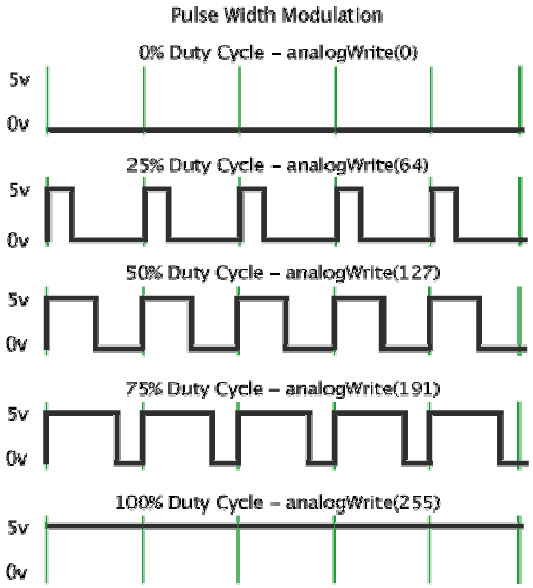
```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  float voltage = sensorValue*5.0/1023.0;

  // read the input on analog pin 1:
  int sensorValue1 = analogRead(A1);
  float voltage1 = sensorValue1*5.0/1023.0;

  // print out the values you read:
  Serial.print(voltage);
  Serial.print("  ");
  Serial.println(voltage1);
  delay(10);        // delay in between reads for stability
}
```

Sl. 10.4.2. Kôd koji omogućava iscrtavanje vrednosti sa dva analogna ulaza istovremeno

4. Sačuvati ovaj kôd pod novim imenom *AnalogRead_NTC_TCRT5000.ino*
5. Povezati *Arduino* sa računarom i spustiti kôd na ploču. Pokrenuti *Serial Monitor* kako bi se uočio format u kome se šalju podaci.
6. Zatvoriti *Serial Monitor* i pokrenuti *Serial Plotter*. Postaviti neki vid prepreke i ispred TCRT5000 senzora i posmatrati kako se dobijeni signal menja sa promenom rastojanja prepreke od senzora. Odgovoriti na sledeća pitanja:
    - Koliki je opseg rastojanja koje ovaj senzor može detektovati?
    - Ukoliko se uočava izražen uticaj šuma kada nema predmeta ispred senzora objasniti njegovo poreklo!
    - Ukoliko bi se izvršla kalibracija ovog senzora u trenutnom okruženju, da li bi se isti rezultati mogli očekivati i ukoliko se senzor iznese napolje?
7. **Isključiti** *Arduino* iz računara. Rastaviti kolo.

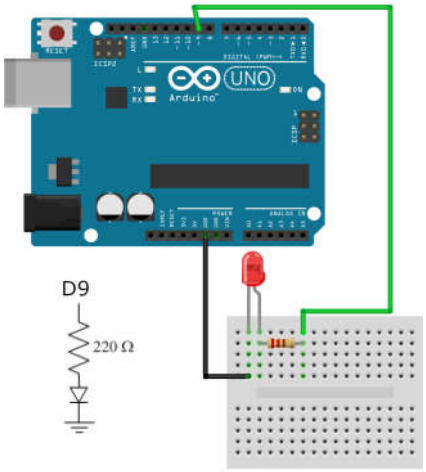
10.5 Impulsno širinska modulacija

Za razliku od *Arduino DUE* ploče, *UNO* ne poseduje D/A konvertor te nije moguće ispisivati analogne vrednosti napona na nekom od njegovih pinova. Međutim, *Arduino UNO* poseduje mogućnost generisanja Impulsno Širinski Modulisanih impulsa (*Pulse Width Modulation - PWM*) na pinovima koji pored svog broja poseduju oznaku ~. Impulsno širinski modulisani pulsevi predstavljaju povorku četvrtki definisane frekvencije kod kojih je moguće menjati procenat vremena jedne periode tokom koga izlazni napon ima visoku vrednost. Ovaj procenat se naziva faktorom ispune (*Duty Cycle - DC*). Impulsno širinska modulacija je neretko dovoljna i poželjna za kontrolu određenih uređaja kao što su neki tipovi motora, promenu intenziteta svetlosti svetleće diode itd. Izgled PWM signala sa različitim faktorom ispune je prikazan na Sl. 10.5.1.



Sl. 10.5.1. Ilustracija impulsno širinske modulacije

1. U *Arduino IDE* okruženju otvoriti primer *Fading* izborom *File»Examples»03.Analog»Fading*.
2. Sačuvati program pod novim imenom *Fading_test.ino*.
3. Povezati kolo prema Sl. 10.5.2. Kôd iz primera je prikazan na Sl. 10.5.3.



Sl. 10.5.2. Povezivanje eksterne svetleće diode na pin 9 *Arduino UNO* ploče

4. Povezati *Arduino* sa računarom i spustiti kôd na ploču. Uočiti šta se dešava sa svetlećom diodom. Objasniti zašto, iako je svetleća dioda uvek u samo dva stanja (uključenom ili isključenom), naše oko registruje kontinualnu promenu intenziteta svetlosti od minimalnog do maksimalnog!
5. **Isključiti** *Arduino* iz računara.

```
void setup() {
  // nothing happens in setup
}

void loop() {
  // fade in from min to max in increments of 5 points:
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of 5 points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

Sl. 10.5.3. Kôd iz primera *Fading* dostupnog u *Arduino IDE* okruženju

---

analogWrite(pin, value): omogućava generisanje PWM signala na odgovarajućim pinovima obeleženim sa ~ na ploči. Vrednost koju ova funckija prima kao drugi argument može biti u opsegu od 0 do 255 (8 bita je na raspolaganju) gde vrednosti iz ovog opsega odgovaraju ovrednostima od 0 do 100% faktora ispune. Ukoliko se za PWM pošalje vrednost 0, vrednost napona će biti uvek 0V, dok u slučaju slanja vrednosti 255 napon kontinualno zadržava vrednostod 5V. Kako *Arduino DUE* ploča poseduje i D/A konvertor povezan na pinove DAC0 i DAC1, funkcija analogWrite() za ove pinove na *DUE* ploči omogućava generisanje pravog analognog napona.

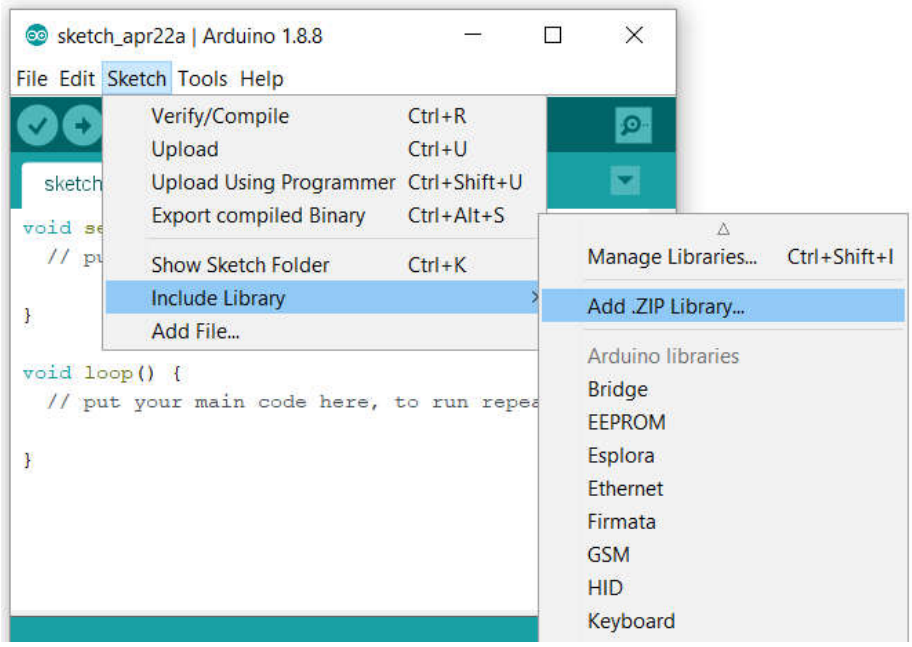
Za razmišljanje: Ukoliko je ipak neophodno dobiti analogni naponski izlaz, a na raspolaganju je *Arduino UNO* ploča, razmisliti koji jednostavan elektronski sklop bi mogao poslužiti kao konvertor PWM signala u jednosmerni napon!

Zadatak 10.5.1. – za samostalni rad
Korišćenjem znanja iz prethodnih poglavlja realizovati *Arduino* program koji u zavisnosti od blizine prepreke TCRT5000 senzoru menja osvetljaj eksterne crvene svetleće diode. TCRT5000 senzor i crvenu svetleću diodu povezati u kolo u skladu sa Sl. 10.4.1 i Sl. 10.5.2. Obezbediti da se provera blizine prepreke izvršava na približno 50 ms.

10.6 Primena biblioteka i prikazivanje vrednosti na displeju

U praktičnim primenama, često je neophodno obezbediti komunikaciju između *Arduino* ploče i korisnika, tj. slanje poruka korisniku, bez posredstva računara. Vizualne informacije složenije prirode, kao što je pročitana vrednost napona na analognom ulazu, nije moguće na jednostavan način predočiti korisniku korišćenjem jednostavnih indikatora poput jednostavne svetleće diode. Rešenje ovog problema se postiže korišćenjem displeja. Jedan od najčešćih tipova displeja danas je tzv. sedmosegmentni displej – displej sa sedam segmenata koji mogu biti uključivani u različitim kombinacijama kako bi prikazivali brojeve od 0 do 9, kao i određena slova. U zavisnosti od realizacije, pored osnovnih sedam segmenata se mogu naći i displeji koji poseduju tačku u donjem desnom uglu.

1. Da bi se omogućila kontrola četvorocifrenog sedmosegmentnog displeja sa *Arduino UNO* ploče, neophodno je najpre izvršiti ubacivanje, tj. *import* eksterne biblioteke u *IDE*. Biblioteka za kontrolu displeja je preuzeta sa linka: https://github.com/avishorp/TM1637 u formi .zip datoteke. Import biblioteke u *Arduino IDE* okruženju je moguć izborom opcije *Sketch»Include Library»Add .ZIP Library…* kao na Sl. 10.6.1. Pored nove biblioteke, *Arduino IDE* okruženje dolazi sa velikim brojem već dostupnih biblioteka o kojima se više informacija može naći na njihovoj zvaničnoj *web* stranici.



Sl. 10.6.1. Postupak *Import*-a biblioteke u *Arduino IDE* okruženje

2. Nakon uspešnog *Import*-ovanja biblioteke u okruženje, napisati program prema Sl. 10.6.2. Ovaj program demonstrira korišćenje TM1637Display biblioteke na jednostavnom primeru u kome se unutar for petlje brojač NumStep inkrementira na svakih 500 ms, nakon što se njegova vrednost ispiše na displeju.

```
#include <TM1637Display.h>

const int CLK = 9; //Set the CLK pin connection to the display
const int DIO = 8; //Set the DIO pin connection to the display


int NumStep = 0;  //Variable to interate

TM1637Display display(CLK, DIO);  //set up the 4-Digit Display.

void setup()
{
  display.setBrightness(7);  //set the diplay to maximum brightness
}


void loop()
{
  for(NumStep = 0; NumStep < 9999; NumStep++)  //Interrate NumStep
  {
    display.showNumberDec(NumStep); //Display the Variable value;
    delay(500);  //A half second delay between steps.
  }
}
```

Sl. 10.6.2. Kôd za kontrolu četvorocifrenog sedmosegmentnog displeja

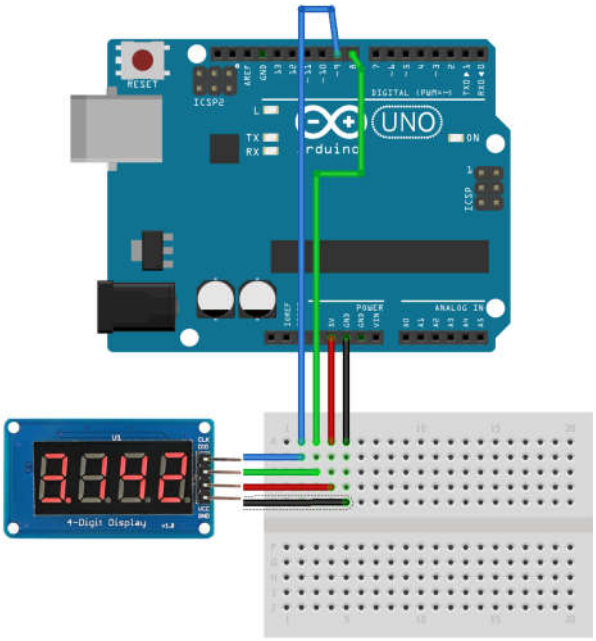
---

#include <TM1637Display.h> : obezbeđuje uključivanje biblioteke u program.
TM1637Display display(CLK, DIO): inicijalizacija displeja.
display.setBrightness(): podešava intenzitet svetlosti kojim svetli displej. Argument funkcije može uzeti vrednosti od 0 do 7 gde je sa 0 označen najslabiji intenzitet, dok u slučaju argumenta 7 displej najintenzivnije svetli.
display.showNumberDec(): prikazuje argument funkcije na displeju.

Sačuvati kôd pod nazivom *tm1637Display.ino*. Povezati samo *Arduino* sa računarom i spustiti kôd na ploču. **Isključiti** *Arduino* iz računara.

3. Povezati modul sa četvorocifrenim sedmosegmentnim displejem sa **isključenom** *Arduino* pločom prema Sl. 10.6.3.
4. Ponovo povezati *Arduino* sa računarom i uočiti promenu brojeva na displeju.
5. Promeniti argument funkcije display.setBrightness() na 0 i spustiti kôd na ploču. Uočiti razliku u intenzitetu svetlosti sa displeja.
6. Isključiti *Arduino* iz računara.

Zadatak 10.6.1 – za samostalni rad
Realizovati program koji će svake sekunde na četvorocifrenom sedmosegmentnom displeju prikazivati vrednost napona termistora povezanog u naponski razdelnik prema Sl. 10.3.1. Prikazani napon je potrebno da bude izražen u mV.



Sl. 10.6.3. Šema prema kojoj je potrebno povezati displej sa *Arduino* pločom

Lekcija 11 – Serijski port. Korišćenje prekida. Primena senzora i aktuatora.

Cilj

Cilj vežbe je da studente:

- upozna sa prednostima i ograničenjima standardnih metoda čitanja sa serijskog porta.
- uvede u principe primene prekida (eng. *interrupt*) uz naglašavanje značaja ovog tipa programiranja u situacijama kada je neophodno obezbediti brzu reakciju mikrokontrolera na spoljašnje događaje ili prosto precizno kontrolisati frekvenciju odabiranja pri akviziciji.
- uputi u uobičajenu problematiku povezivanja spoljašnjih senzora i aktuatora na *Arduino* ploču.

Oprema

- Računar sa instaliranim ***Arduino IDE*** softverskim okruženjem.
- ***Arduino UNO R3*** ploča.
- Protobord, kratkospojnici, otpornici (220 Ω, 10 KΩ), NTC termistor, svetleća dioda, kapacitivni senzor dodira, četvorocifreni sedmosegmentni displej sa TM1637 čipom, HC-SR04 ultrazvučni modul, pasivni *piezo buzzer*, *Servo* motor.

NAPOMENA: Najnoviju verziju *Arduino IDE* okruženja, kao i dokumentaciju vezanu za platformu je moguće naći na zvaničnom *Arduino* sajtu: https://www.arduino.cc/

11.1 Slanje poruka Arduino ploči preko serijskog porta

Ukoliko je potrebno kontrolisati izvršavanje programa na *Arduino* ploči putem računara, tj. omogućiti korisniku interakciju sa programom slanjem definisanih poruka, koristiće se skup funkcija za prijem i čitanje poruka poslatih *Arduino* ploči. Kako poruke obično dolaze na ploču povremeno i u nedefinisanim vremenskim intervalima (kada korisnik odluči da ih pošalje), potrebno je obezbediti da se čitanje sa porta izvršava samo onda kada postoji poruka koju je potrebno pročitati. U tom slučaju je *Arduino* slobodan da nesmetano obavlja druge zadatke između poruka. Kroz naredni primer će biti objašnjeni ključni koncepti u realizaciji programa koji omogućava korisniku da slanjem poruke „*start*“ ili „*stop*“ započne, odnosno zaustavi akviziciju analognog napona sa učestanošću od 1 Hz. Indikacija da *Arduino* čeka na start akvizicije će biti uključivanje i isključivanje ugrađene svetleće diode na ploči sa učestanošću od 1 Hz. Na ovaj način će biti ilustrovana sposobnost programa da

nesmetano izvršava druge zadatke (analognu akviziciju ili uključivanje svetleće diode) u periodu između poruka.

3. Pokrenuti *Arduino IDE* okruženje i kreirati promenljive **inputString**, **operation** i **voltage** tipa String, String i int respektivno. Podesiti inicijalne vrednosti promenljivih **inputString** i **operation** na prazan string ("") i promenljive **voltage** na 0.
4. U setup delu programa započeti serijsku komunikaciju sa računarom korišćenjem funkcije Serial.begin(9600) i podesiti pin na kome se nalazi ugrađena svetleća dioda kao izlazni korišćenjem funkcije pinMode(LED_BUILTIN, OUTPUT).
5. Za sada ostaviti praznu loop funkciju i kreirati novu funkciju, naziva serialEvent(). Ova funkcija ne vraća ništa kao rezultat, a biće pozvana automatski svaki put kada se detektuju dolazeći podaci. U telu funkcije serialEvent je potrebno upisati kôd koji će omogućiti čitanje dostupnih podataka na serijskom portu. Kôd koji je potrebno kreirati unutar funkcije serialEvent je prikazan na Sl. 11.1.1.

```
void serialEvent() {
  while (Serial.available()) {

    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      operation = inputString;
      inputString = "";
      Serial.println(operation);
    }
    else {
      inputString += inChar;
    }


  }
}
```

Sl. 11.1.1. Funkcija serialEvent()

Ovaj kôd omoućava čitanje karaktera sa serijskog porta sve dok postoje karakteri u baferu koje je moguće pročitati. Nakon što je karakter pročitan proverava se da li je "*newline*" karakter. Ukoliko nije, dodaje se na postojeći **inputString**. Ukoilko je ipak poslati karakter "*newline*", tj. "\n", smatra se da je cela poruka poslata i vrednost iz promenljive **inputString** se upisuje u promenljivu **operation** koja definiše trenutni zadatak programa ("start" ili "stop"). Radi provere rada programa se podaci iz promenljive **operation** šalju na računar gde se ispisuju, a promenljiva **inputString** se vraća na prazan string kako bi bila spremna za čitanje sledeće poruke.

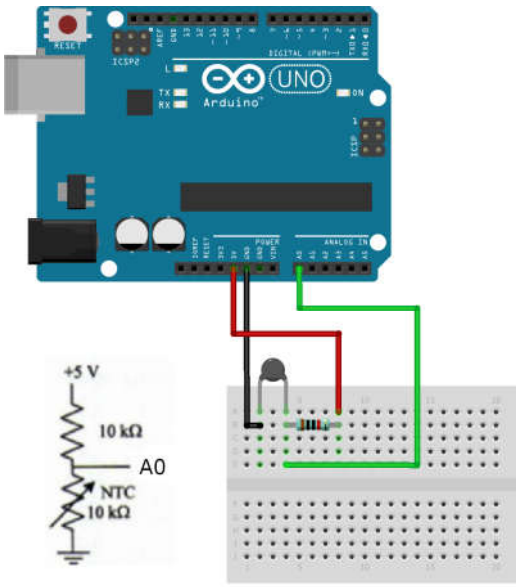
Serial.read(): funkcija za čitanje iz bafera sa serijskog porta.
Serial.available(): funkcija koja vraća broj bajtova (karaktera) koji su pristigli i nalaze se u baferu na serijskom portu. Maksimalan broj nepročitanih karaktera u baferu je 64.

6. Unutar loop funkcije je potrebno obezbediti izvršavanje različitih zadataka u zavisnosti od trenutne vrednosti promenljive **operation**. Ukoliko je u promenljivu **operation** upisano “start”, potrebno je da program sa učestanošću od 1 Hz izvršava akviziciju analognog napona i slanje dobijene vrednosti ka računaru. Ukoliko je u promenljivu **operation** upisano “stop”, potrebno je obezbediti da se sa učestanošću od 1 Hz uključuje i isključuje svetleća dioda ugrađena na *Arduino* ploči. Ukoliko vrednost promenljive **operation** nije ni “start” ni “stop”, korsniku se prijavljuje greška i u promenljivu **operation** se upisuje “stop”. Kôd koji unutar loop sekcije obebeđuje željenu funkcionalnost je prikazan na Sl. 11.1.2.

```
void loop() {

  if (operation == "start") {
    voltage = analogRead(A0);
    Serial.println(voltage);
    delay(1000);
  }

  else if (operation == "stop") {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500);
    digitalWrite(LED_BUILTIN, LOW);
    delay(500);
  }

  else {
    Serial.println("Greska pri unosu! Program se vraca u stanje cekanja.");
    operation = "stop";
    delay(1000);
  }
}
```

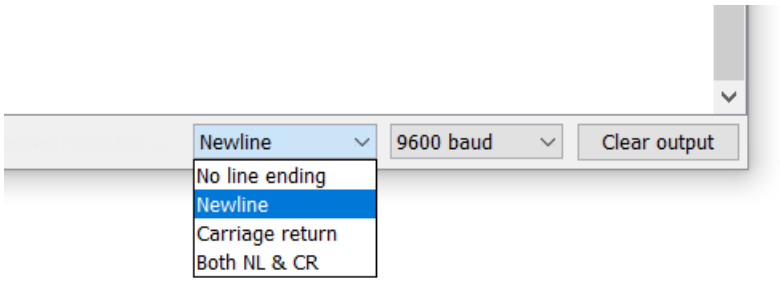
Sl. 11.1.2. Kôd unutar loop funkcije programa

7. Povezati kolo prema šemi sa S. 11.1.3. Povezati *Arduino* sa računarom, sačuvati program kao *SerialEvent_start_stop.ino* i spustiti ga na ploču.



Sl. 11.1.3. Povezivanje naponskog razdelnika sa NTC termistorom na *Arduino*

8. Testirati rad programa u slučaju izbora različitih opcija iz padajućeg menija na Sl. 11.1.4.
9. **Isključiti** *Arduino* iz računara.



Sl. 11.1.4. Definisanje karaktera koji še šalju na kraju poruke

Zadatak 11.1.1. – za samostalni rad

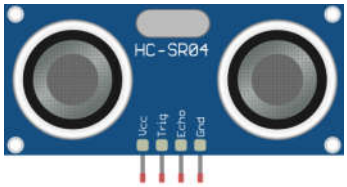
Realizovati program koji omogućava korisniku da slanjem vrednosti od 0 do 255 preko serijskog porta podešava faktor ispune impulsno širinski modulisanih pulseva koji kontrolišu svetleću diodu povezanu u kolo prema Sl. 10.5.2. Za konverziju iz *string*-a u *int* koristiti sekciju kôda sa Sl. 11.1.5.

```
if (inChar >= '0' && inChar <= '9') // is inChar a number?
{
  inputString = inputString * 10 + inChar - '0'; // yes, accumulate the value
}
```

Sl. 11.1.5. Konverzija iz stringa

11.2 Ultrazvučni senzor rastojanja

Modul HC-SR04 koji predstavlja ultrazvučni senzor rastojanja se sastoji iz dva osnovna elementa, emitera ultrazvuka koji generiše ultrazvučne talase frekvencije 40 KHz i detektora ultrazvuka koji detektuje ultrazvučne talase koji se reflektuju od prepreke i vraćaju ka modulu. Na taj način je, poznajući brzinu zvuka u vazduhu, jednostavno moguće doći do informacije o udaljenosti prepreke od modula ukoliko se izmeri vreme proteklo od generisanja ultrazvučnog talasa do njegovog povratka na detektor. *Pinout* modula HC-SR04 je prikazan na Sl. 11.2.1.



Sl. 11.2.1. *Pinout* HC-SR04 modula

Dok je funkcija pinova Vcc i Gnd jasna (pinovi za napajanje modula), funkcija pinova Trig i Echo mora biti dodatno objašnjena. Naime, Trig pin služi za generaciju ultrazvučnih pulseva. Ukoliko se na Trig pin dovede napon od 5 V u trajanju od 10 µs doći će do generacije 8 ultrazvučnih pulseva, dok Echo pin pri detekciji ultrazvuka menja vrednost na LOW.

1. Kreirati novi program i kreirati promenljive **trajanje** i **rastojanje** tipa long, i int respektivno.
2. Kreirati konstante **trigPin** i **echoPin** tipa int koje će označavati pinove 9 i 10 *Arduino* ploče respektivno.
3. U setup sekciji kôda definisati **trigPin** digitalni pin kao izlazni, a **echoPin** digitalni pin kao ulazni. Započeti serijsku komunikaciju sa računarom.
4. U loop sekciji kôda pri svakom ulasku najpre obezbediti da je stanje pina **trigPin** podešeno na LOW. Zatim zaustaviti izvršavanje programa na 2 µs korišćenjem funkcije delayMicroseconds(2).
5. Sledeći korak loop sekcije zahteva postavljanje **trigPin** digitalnog pina u stanje HIGH na 10 µs kako bi se generisali ultrazvučni pulsevi. Ovaj zadatak je moguće realizovati uzastopnim korišćenjem funkcija digitalWrite() i delayMicroseconds() kao na Sl. 11.2.2.

```
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
```

Sl. 11.2.2. Kôd koji obezbeđuje slanje napona od 5 V u trajanju od 10 µs na trigPin

6. Vreme koje je proteklo od generisanja ultrazvučnih impulsa do njihove detekcije na **echoPin** pinu nakon refleksije je moguće izmeriti korišćenjem funkcije pulseIn(echoPin, HIGH).

delayMicroseconds(µs): zaustavlja izvršavanje programa za definisani broj mikrosekundi.
pulseIn(pin, value): za pin koji predstavlja prvi argument funckije meri vreme u mikrosekundama koje taj pin provede HIGH ili LOW stanju, gde se željeno stanje definiše drugim argumentom funkcije - "value". Funkcija kao rezultat vraća izmereni broj mikrosekundi. Po potrebi je moguće dodati i treći argument funkcije "timeout" koji definiše koliko dugo će se čekati na završetak trajanja impulsa.

Naime, pri generisanju ultrazvučnih impulsa senzor automatski podešava stanje **echoPin** pina na HIGH. Kada detektor ultrazvuka detektuje reflektovani ultrazvučni talas, stanje **echoPin** pina se vraća na LOW. Rezultat funkcije pulseIn(echoPin, HIGH) je potrebno upisati u prethodno definisanu promenljivu **trajanje**.

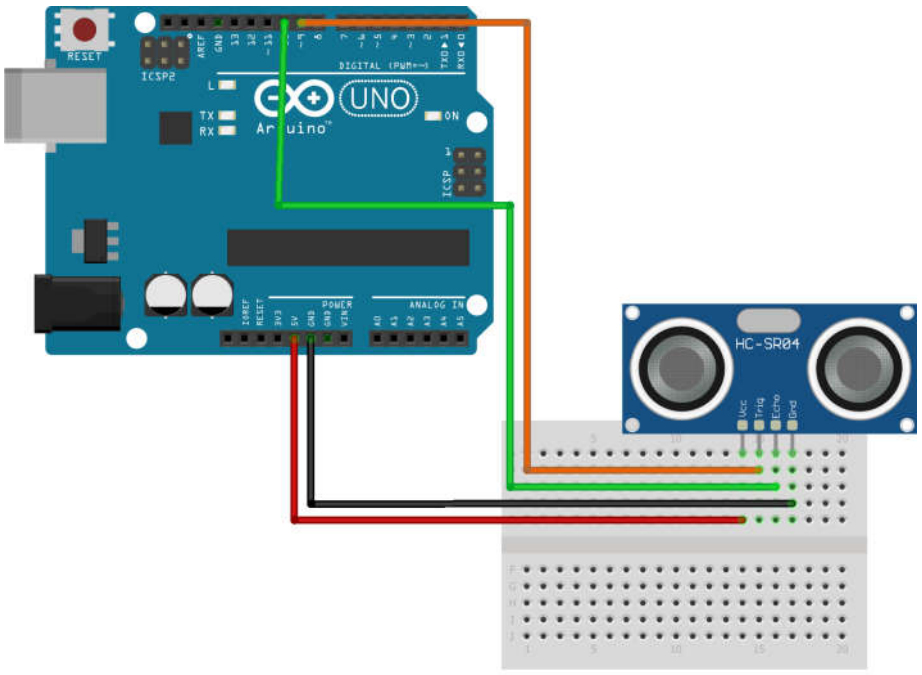
7. Pretpostavljajući da je brzina zvuka u vazduhu za trenutne parametre vazduha približno 340 m/s napisati jednačinu koja na osnovu vrednosti promenljive **trajanje** dolazi do rastojanja izraženog u centimetrima i rezultat jednačine upisati u promenljivu **rastojanje**.

NAPOMENA: Obratiti pažnju da je vreme u μs koje je proteklo od generisanja pulseva do njihovog povratka do modula nakon refleksije o prepreku zapravo vreme koje potrebno ultrazvučnom talasu da pređe **dva** rastojanja od modula do prepreke!

8. Ispisati rezultat za rastojanje izraženo u centimetrima na računaru u formi prikazanoj na Sl. 11.2.3. korišćenjem funkcija Serial.print() i Serial.println().

Sl. 11.2.3. Forma u kojoj je potrebno ispisati rezultate

9. Obezbezbediti da se merenje izvršava dva puta u sekundi dodavanjem funkcije delay(500).
10. Povezati HC-SR04 modul sa *Arduino* pločom prema Sl. 11.2.4.



Sl. 11.2.4. Šema prema kojoj je potrebno povezati *Arduino* i HC-SR04 modul

11. Sačuvati program kao *HC-SR04.ino*, povezati *Arduino* sa računarom i spustiti kôd na ploču.
12. Otvoriti *Serial Monitor* i testirati rad programa postavljanjem prepreke ispred ultrazvučnog senzora rastojanja. Do koje maksimalne udaljenosti senzor pokazuje adekvatno očitavanje? Gde sve nalaze primene senzori koji koriste isti ili sličan princip merenja?
13. **Isključiti** *Arduino* iz računara.

Zadatak 11.2.1. – za samostalni rad

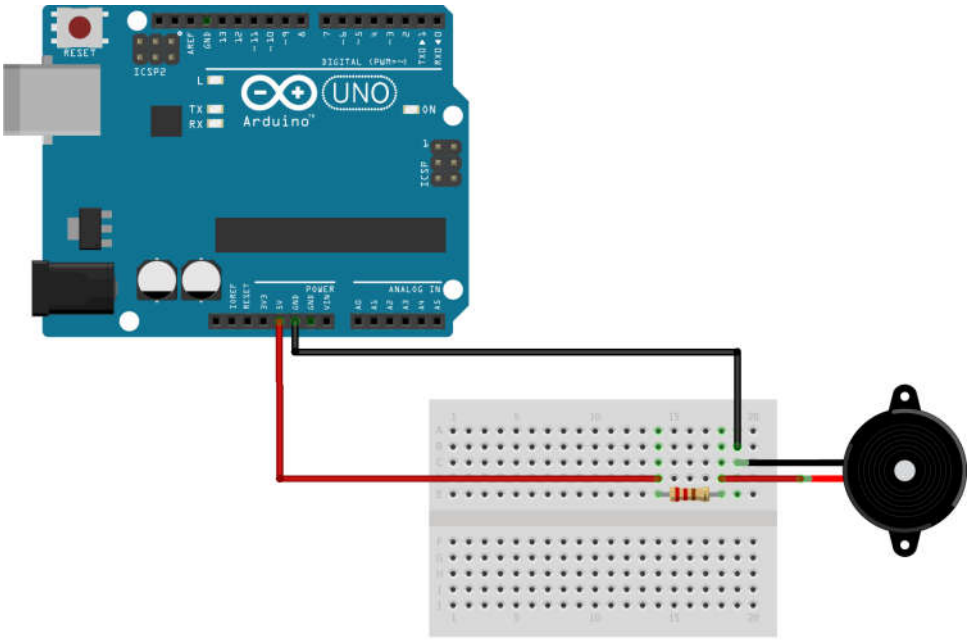
Sačuvati kopiju prethodnog programa pod novim imenom i modifikovati je tako da se vrednosti izmerenog rastojanja prikazuju na četvorocifrenom sedmosegmentnom displeju. Displej povezati prema Sl. 10.6.3, a **trigPin** promeniti na pin 11.

11.3 Arduino i zvučna signalizacija

Pored svetlosne signalizacije koja je već obrađena u okviru prethodnih poglavlja, projektima je često potrebno dodati i neki vid zvučne signalizacije. U najjednostavnijem slučaju, od interesa je korišćenje *piezo buzzer*-a, komponente koja u sebi sadrži materijal sa piezoelektričnim svojstvima. Piezoelektrični efekat materijala rezultuje deformacijom materijala kada se on nađe u električnom polju. Ovaj efekat je poznat kao inverzni piezoelektrični efekat i omogućava stvaranje zvuka i ultrazvuka usled periodičnih deformacija materijala koje su posledica promenljivog napona na *buzzer*-u. Direktni piezoelektrični efekat predstavlja generisanje naelektrisanja usled deformacije materijala i on se može koristiti pri detekciji ultrazvuka kao npr. kod modula HC-SR04.

U praksi je moguće naići na dve vrste *piezo buzzer*-a: aktivne i pasivne. Aktivni u sebi sadrže elektroniku koja automatski generiše periodičnu pobudu, te pobuđuje piezoelektrični materijal. Za njihov rad je stoga dovoljno priključiti ih na jednosmerni napon. Pasivni *piezo buzzer* zahteva da se pobuđuje periodičnim signalom kako bi se generisao zvuk.

1. Povezati *piezo buzzer* u šemu sa Sl. 11.3.1.



Sl. 11.3.1. Šema za povezivanje *piezo buzzer*-a sa *Arduino* pločom

NAPOMENA 1: Voditi računa o polarizaciji *piezo buzzer*-a. Kraj komponente koji je potrebno spojiti na pozitivan napon je često označen dužim metalnim kontaktom i/ili plusom na kućištu same komponente.

NAPOMENA 2: *Piezo buzzer* se obično povezuje na digitalni izlaz mikrokontrolera. U zavisnosti od same komponente, može se desiti da *piezo buzzer* poseduje veoma malu otpornost, te u slučaju direktnog povezivanja na digitalni izlaz povlači veću struju od maksimalne izlazne struje na digitalnom pinu *Arduino* ploče. Iz tog razloga, kako ne bi došlo do oštećenja digitalnog izlaza *Arduina*, poželjno je *piezo buzzer* povezivati na red sa otpornikom otpornosti 100-400 Ω, te na taj način ograničiti struju koju komponenta vuče.

2. Uključiti *Arduino* u računar i na osnovu ponašanja utvrditi da li *piezo buzzer* koji se koristi u zadatku spada u aktivne ili pasivne!
3. **Isključiti** *Arduino* iz računara.

U narednom zadatku će program iz sekcije 11.2. biti modifikovan tako da se u zavisnosti od izmerenog rastojanja od prepreke generišu različiti tonovi na *piezo buzzer*-u.

1. Otvoriti program koji je napisan u sekciji 11.2. i sačuvati njegovu kopiju pod novim nazivom. U kopiji programa dodati novu konstantu **tonePin** tipa int. Ovaj pin će biti korišćen za kontrolu *piezo buzzer*-a.
2. Na samom kraju loop sekcije, pre pozivanja delay(500) funkcije, potrebno je dodati sekciju kôda koja će obezbediti da *piezo buzzer* svira: ton frekvencije 100 Hz ukoliko se prepreka nalazi na rastojanju između 15 i 20 cm, ton frekvencije 250 Hz ukoliko se prepreka nalazi na rastojanju između 10 i 15 cm, i ton frekvencije 400 Hz ukoliko se prepreka nalazi na rastojanju manjem od 10 cm. Generisanje pulseva podesive frekvencije je u *Arduino IDE* okruženju omogućeno korišćenjem funkcije tone(). Deo kôda koji je potrebno dodati je prikazan na Sl. 11.3.2.
3. Otvoriti *Serial Monitor* i testirati rad kreiranog sistema menjanjem rastojanja prepreke od HC-SR04 modula.
4. **Isključiti** *Arduino* iz računara i ukloniti HC-SR04 modul sa šeme, a *piezo buzzer* ostaviti povezan.

```
if (rastojanje <= 20 && rastojanje > 15) {
  tone(tonePin, 100);
}
else if (rastojanje <= 15 && rastojanje > 10) {
  tone(tonePin, 250);
}
else if (rastojanje <= 10) {
  tone(tonePin, 400);
}
else {
  noTone(tonePin);
}
```

Sl. 11.3.2. Sekcija kôda koja obezbeđuje sviranje različitih tonova na *piezo buzzer*-u u zavisnosti od rastojanja prepreke

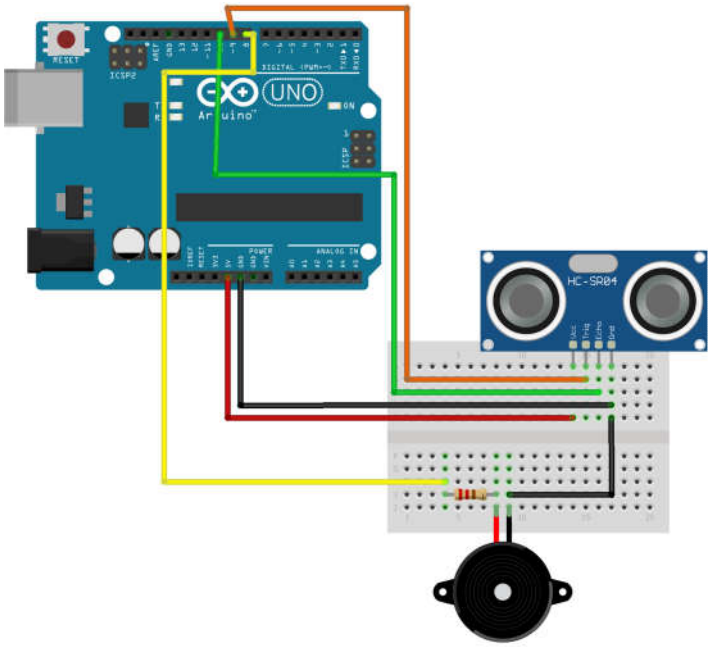
---

tone(pin, frequency, duration): generiše povorku četvrtki definisane frekvencije izražene u Hz na željenom pinu. Poslednji argunemt funkcije nije obavezan, a definiše trajanje tona u milisekundama.
noTone(pin): prekida generisanje povorke četvrtki ukoliko ton nije već prekinut u slučaju isteka definisanog trajanja (treći argument funkcije tone()).

Napomene:
- Funkcija tone() onemogućava korišćenje funkcije analogWrite() na pinovima 3 i 11.
- Nije moguće koristiti funkciju tone() za frekvencije manje od 31 Hz.
- Istovremeno generisanje različitih tonova na vše pinova nije moguće, već je potrebno pozvati funkciju noTone() pre poziva funkcije tone () na sledećem pinu.

---

5. Sačuvati program, povezati šemu prema Sl. 11.3.3, povezati *Arduino* sa računarom i spustiti kôd na ploču.



Sl. 11.3.3. Šema prema kojoj se povezuju *buzzer* i HC-SR04 modul sa *Arduino* pločom

6. Otvoriti primer *toneMelody* koji se nalazi u sekciji *File»Examples»02.Digital* i sačuvati ga kao novi *Sketch* pod nazivom *toneMelody_test.ino*. U okviru ovog primera se poziva "pitches.h" koja sadrži definisane konstante za frekvencije nota od B0 do D#8.
7. Proučiti prikazani program.
8. Povezati *Arduino* sa računarom, spustiti kôd na ploču i testirati rad sistema.
9. **Isključiti** *Arduino* iz računara.

Zadatak 11.3.1. – za samostalni rad

U dokumentu SuperMario.txt se nalaze note i trajanje nota za melodiju iz igrice Super Mario. Modifikovati kopiju prethodnog programa tako da *piezo buzzer* svira ovu melodiju **repetitivno**.

11.4 Korišćenje prekida

Nekada je od presudne važnosti detektovati određeni događaj i bez odlaganja izvršiti zadatu sekciju kôda nevezano od toga gde se trenutno nalazimo u programu. Primer za to bi bilo praćenje važnih signala sa senzora na čije promene je potrebno brzo odreagovati ili pak izvršavanje određenih zadataka u tačno definisanim vremenskim intervalima. Nekada nije poželjno opterećivati procesor konstantnim proveravanjem vrednosti na nekom digitalnom ulazu repetitivnim korišćenjem funkcije digitalRead() u loop sekciji programa, već je poželjno rešiti to na drugi način, a osloboditi procesor kako bi mogao da izvršava druge zadatke unutar petlje. Rešenje svih prethodno navedenih problema je moguće korišćenjem prekida (eng. *interrupt*). Prekid obezbeđuje da se na određeni definisani događaj odreaguje brzo i izvrši se određena sekcija kôda koja se nalazi unutar željene funkcije koja se naziva *Interrupt Service Routine (ISR)*. Pri detekciji posmatranog događaja procesor prekida ono što je u tom trenutku radio, izvršava kôd unutar ISR i nakon toga se vraća prethodno započetim zadacima u glavnom delu kôda.

U zadatku koji sledi će biti realizovan program koji će obavljati istu funkciju kao i program sa Sl. 10.2.3, tj. funkciju uključivanja ugrađene svetleće diode sve dok je kapacitivni senzor dodira dodirnut. Međutim, ovaj program će koristiti prekid za praćenje stanja kapacitivnog senzora dodira, te će procesor biti slobodan da radi druge zadatke unutar loop dela kôda!

1. Kreirati novi *Sketch* pritiskom na *File»New* i sačuvati ga pod nazivom *TouchLED_interrupt.ino*. U programu najpre definisati konstantu tipa int pod nazivom **touchPin** i njenu vrednost podesiti na 2 (ovo će biti pin na koji će biti povezan izlaz kapacitivnog senzora dodira). Zatim kreirati promenljivu volatile byte pod nazivom **state** i inicijalizovati je na LOW stanje.
2. U setup sekciji kôda inicijalizovati LED_BUILTIN pin kao izlazni, i **touchPin** kao INPUT_PULLUP. Zatim pozvati funkciju sa argumentima attachInterrupt(digitalPinToInterrupt(touchPin), blink, CHANGE).

---

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode): funkcija koja uključuje prekid. Prvi argument ove funkcije je broj, 0 ili 1 koji predstavlja broj prekida, a ne broj pina čije se stanje posmatra! Za *Arduino UNO R3* ploču brojevi 0 i 1 se odnose na pinove 2 i 3 respektivno. Ipak, nije preporučljivo koristiti vrednosti 0 i 1 direktno u funkciji, već je bolje kosrititi funkciju digitalPinToInterrupt(pin) koja kao argument prima željeni pin (2 ili 3 za UNO) i prevodi ga u odgovarajući broj prekida. Drugi argument funkcije attachInterrupt() predstavlja naziv funkcije koja će se izvršiti kada do prekida dođe. Treći i poslednji argument funkcije predstavlja mod koji definiše kakva akcija na željenom pinu će dovesti do prekida. Dozvoljeni modovi su: LOW, CHANGE, RISING i FALLING. U slučaju moda CHANGE koji se koristi u zadatku, potrebno je inicijalizovati *Interrupt* pin kao INPUT_PULLUP.
detachInterrupt(digitalPinToInterrupt(pin)): isključuje željeni prekid.

---

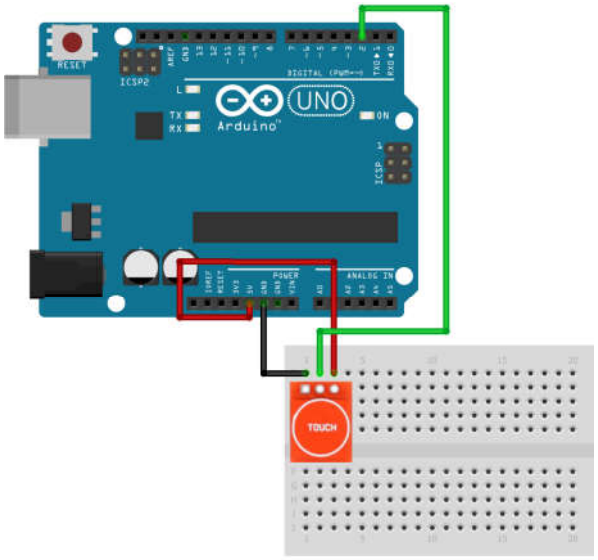
3. Glavnu petlju ostaviti praznu, a definisati novu funkciju naziva **blink** koja ne prima argumente i ne vraća rezultate - void blink(). Ova funkcija će biti ISR i izvršavaće se svaki put kada dođe do prekida.
4. U telu ove funkcije obezbediti da se pri svakom ulasku u ISR **blink** vrednost promenljive **state** menja - state = !state, a zatim upisati novu vrednost promenljive **state** na digitalnom izlazu ugrađne svetleće diode korišćenjem funkcije digitalWrite().

Konačan izgled programa je prikazan na Sl. 11.4.1. Obratiti pažnju da unutar *loop* dela kôda ne postoje instrukcije, tj. da je sada u loop delu kôda moguće izvršavati druge stvari po potrebi!

5. Sačuvati promene u programu.
6. Povezati kolo prema Sl. 11.4.2, povezati *Arduino* sa računarom i spustiti kôd na ploču.
7. Testirati rad programa dodirivanjem kapacitivnog senzora dodira i posmatranjem reakcije ugrađene svetleće diode na ploči. Ukoliko je sve urađeno prema instrrukcijama, ne bi trebalo da bude razlike u radu programa sa Sl. 10.2.3. i programa sa Sl. 11.4.1. uz komentar da novi program izvršava promenu stanja ugrađene svetleće diode samo kada dođe do promene stanja na pinu 2 *Arduino* ploče, a nema ništa u loop sekciji kôda!
8. **Isključiti** *Arduino* iz računara, i rastaviti kolo sa protoborda.

```
const int touchPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(touchPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(touchPin), blink, CHANGE);
}

void loop() {
}

void blink() {
  state = !state;
  digitalWrite(LED_BUILTIN, state);
}
```

Sl. 11.4.1. Program za čitanje sa kapacitivnog senzora dodira i uključivanje svetleće diode korišćenjem prekida



Sl. 11.4.2. Šema povezivanja kapacitivnog senzora dodira sa *Arduino* pločom u slučaju korišćenja prekida

NAPOMENA: Pri pisanju ISR funkcija treba poštovati sledećih nekoliko veoma važniih pravila:

- ISR mora biti veoma kratkog trajanja! Pri prekidu se prekida izvršavanje glavne petlje što ne bi trebalo dugo da traje!
- ISR nema ni ulazne ni izlazne promenljive.
- Unutar ISR ne koristiti funkcije delay(), Serial.println(), Serial.read().
- Promenljive koje se koriste za razmenu podataka između ISR i glavnog programa obavezno deklarisati kao volatile kako bi kompajler znao da ove promenljive mogu promeniti vrednost u svakom trenutku i kako ih ne bi eliminisao kao "nekorišćene" radi čuvanja memorije.

Zadatak 11.4.1. – za samostalni rad

Realizovati program koji omogućava trigerovanu akviziciju na *Arduino* ploči korišćenjem prekida. Potrebno je da akvzicija počne kada se detektuje silazna ivica na pinu 2 *Arduino* ploče. Za detekciju silazne ivice podesiti stanje pina 2 kao INPUT. Silaznu ivicu generisati prebacivanjem kraja kratkospojnika sa 5 V pina na GND.

11.5 Biblioteka TimerOne *za kontrolu prvog hardverskog tajmera*

Mikrokontroler na *Arduino* ploči poseduje tri hardverska tajmera, *Timer0*, *Timer1* i *Timer2*. Tajmeri su odgovorini za neke od ključnih funkcionalnosti koje su korišćene u programima koji su prethodno demonstrirani. Tako je, na primer, osmobitni tajmer *Timer0* direktno odgovoran za rad funkcija za merenje vremena delay(), millis() (funkcija koja vraća trenutno vreme u miliskundama od početka izvršavanja programa) i micros() (funkcija koja vraća trenutno vreme u mikrosekundama od početka izvršavanja programa sa korakom od 4 μs). Ovaj tajmer je povezan na pinove 5 i 6 *Arduino UNO R3* ploče i takođe je odgovoran za PWM na ovim pinovima. Osmobitni tajmer *Timer2* obezbeđuje precizno podešavanje frekvencije pri korišćenju funkcije tone() i odgovoran je za PWM na pinovima 3 i 11 *Arduino UNO R3* ploče. Tajmer sa najvećom rezolucijom od čak šesnaest bita je *Timer1* i on je odgovoran za PWM na pinovima 9 i 10 *Arduino UNO R3* ploče. *Timer1* se koristi unutar biblioteke za upravljanje *Servo* motorom, a inače je slobodan za posebne primene. Biblioteka koja olakšava korišćenje ovog tajmera je TimerOne biblioteka koju je moguće preuzeti sa sledećeg linka: https://github.com/PaulStoffregen/TimerOne. Ova biblioteka omogućava dve ključne funkcionalnosti:

- potpunu kontrolu nad generacijom PWM signala na pinovima 9 i 10 sa velikom rezolucijom faktora ispune i frekvencije; i
- periodično generisanje prekida, svaki put kada tajmer izbroji određeni broj mikrosekundi. Ova funckionalnost je posebno važna kada je potrebno veoma precizno definisati frekvenciju odabiranja pri analognoj akviziciji.

Program koji će biti obrađen treba da obezbedi akviziciju sa precizno definisanom frekvencijom odabiranja korišćenjem prekida.

1. Kreirati novi program i u njega uključiti biblioteku *TimerOne.h* izborom opcije *Sketch»Include Library»Add .ZIP Library*.
2. Deklarisati dve promenljive volatile int **voltage** i volatile byte **state** i inicijalizovati ih na 0.
3. U setup sekciji kôda započeti serijsku komunikaciju, inicijalizovati prvi tajmer na 100 ms korišćenjem funkcije Timer1.initialize(1000000) i aktivirati prekid za ovaj tajmer korišćenjem funkcije Timer1.attachInterrupt(timerIsr).

Timer1.initialize(value): inicijalizuje prvi hardverski tajmer. Argument funkcije je perioda tajmera izražena u mikrosekundama.
Timer1.attachInterrupt(function, period): poziva ISR čiji je naziv prvi argument funkcije na određeni broj mikrosekundi definisan drugim argumentum funkcije. Ukoliko se drugi argument funkcije izostavi, ISR se poziva na osnovu argumenta funkcije Timer1.initialize().

4. Zatim kreirati ISR pod nazivom **timerIsr**. Unutar **timerIsr** najpre obezbediti da pri svakom ulasku u taj deo kôda promenljiva **state** uzme vrednost 1 čime se označava da je ISR izvršena. Zatim pročitati vrednost na analognom ulazu A0 i upisati je u promenljivu **voltage**.

5. Unutar glavne petlje pri svakoj iteraciji proveravati vrednost promenljive **state**. Samo ukoliko je **state** jednako 1, na serijskom portu ispisati pročitanu vrednost sa A/D konvertora i vratiti stanje promenljive **state** na 0.

NAPOMENA: Na ovaj način se obezbeđuje da se čitanje analognog napona izvršava u definisanim vremenskim inetrvalima, dok se ispisivanje vrednosti izvršava u pauzi između čitanja samo onda kada postoji pročitana vrednost. Ovako definisan kôd, prikazan na Sl. 11.5.1, poseduje značajno precizniju frekvenciju odabiranja od kôda koji koristi funkciju delay() unutar glavne petlje, jer tada vreme između dva uzorkovanja zavisi od vremena izvršavanja svih ostalih funkcija u glavnoj petlji! Ipak, perioda sa kojom se poziva ISR ne sme biti isuvše mala kako bi bili sigurni da će kôd unutar ISR stići da se izvrši.

```
#include <TimerOne.h>

volatile int voltage = 0;
volatile byte state = 0;

void setup()
{
  Serial.begin(9600);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(timerIsr);
}

void loop()
{
  if(state){
    Serial.println(voltage);
    state = 0;
    }
}

void timerIsr()
{
    state = 1;
    voltage = analogRead(A0);
}
```

Sl. 11.5.1. Akvizicija sa precizno definisanom frekvencijom odabiranja

6. Sačuvati program kao *Timer_Acq.ino*, povezati *Arduino* sa računarom i testirati program. Rezultate čitanja analognog napona analizirati korišćenjem *Serial Monitor*-a. Spustiti kôd na ploču nekoliko puta sa različitim verdnostima za periodu sa kojom se izvršava čitanje sa A/D konvertora.
7. **Isključiti** *Arduino* iz računara.

11.6 Upravljanje Servo motorom

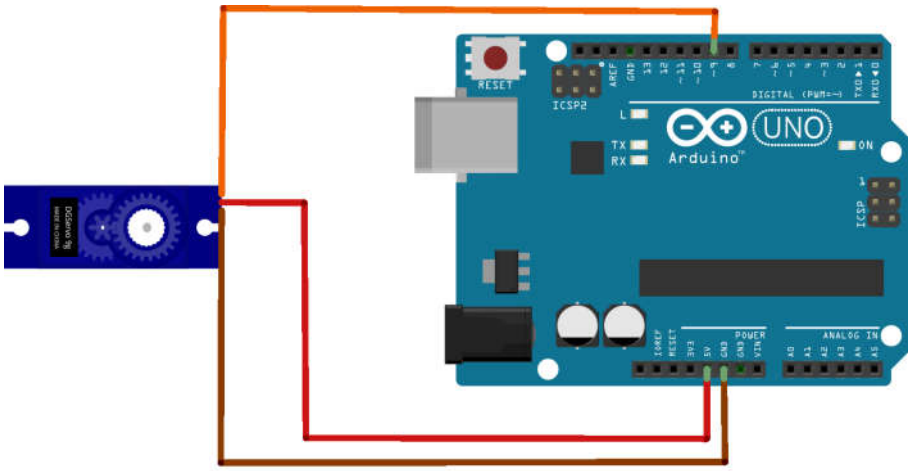
Servo motori su motori kod kojih je moguće precizno kontrolisati položaj vratila slanjem PWM signala sa *Arduino* ploče. Precizna kontrola položaja je posledica povratne sprege koju motor poseduje. Pulsevi koji služe za kontrolu *Servo* motora ne traju dugo, obično od oko 0.5 do oko 2 ms. Definisani opseg trajanja za konkretnu komponentu se stoga skalira na opseg uglova od 0 do 180 stepeni. Pored pina za kontrolu pozicije često žute ili narandžaste boje, *Servo* motor poseduje i pinove za napajanje Vcc i Gnd, najčešće crvene i crne(ili braon) boje respektivno. Jednostavna kontrola *Servo* motora je omogućena korišćenjem funkcija iz *Servo.h* biblioteke koja dolazi već inkorporirana u *Arduino IDE*.

1. Otvoriti primer *Sweep* izborom opcije *File»Examples»Servo* analizirati strukturu programa.

Servo myservo: kreira *Servo* objekat naziva **myservo** koji omogućava kontrolu *Servo* motora.
myservo.attach(pin, min, max): prvi argument ove funkcije predstavlja pin za kontrolu na koji je *Servo* povezan. Preostala dva argumenta nisu obavezna, a predstavljaju minimalno i maksimalno trajanje poslatih pulseva koji odgovaraju rotaciji od 0 i 180 stepeni respektivno. Ukoliko se ovi argumenti izostave, šalju se standardne vrednosti od 544 µs trajanja za 0 stepeni i 2400 µs trajanja za 180 stepeni.
myservo.write(angle): funkcija koja služi za kontrolu ugla *Servo* motora. Argument funkcije je ugao izražen u stepenima koji funkcija prevodi u odgovarajuće trajanje poslatog impulsa.

---

2. Povezati *Servo* na *Arduino* prema šemi sa Sl. 11.6.1.
3. Povezati *Arduino* sa računarom i spustiti kôd na ploču.



Sl. 11.6.1. Šema povezivanja *Servo* motora sa *Arduino* pločom

4. **Isključiti** *Arduino* iz računara, ali šemu ostaviti sastavljenu.
5. Kreirati novi program u kome će prema principu rada programa iz *Zadatka 11.1.1.* biti realizovan program koji će omogućiti kontrolu pozicije *Servo* motora slanjem poruka sa računara. Najpre u program uključiti biblioteku *Servo.h* i definisati promenljive **inputAngle** i **angle** tipa int koje će služiti za čuvanje trenutno pročitane vrednosti i krajnje pročitane vrednosti respektivno, Sl. 11.6.2. Zatim kreirati *Servo* objekat naziva **myservo**.

```
#include <Servo.h>

int inputAngle = 0;
int angle = 0;

Servo myservo;
```

Sl. 11.6.2.

6. U okviru setup sekcije programa započeti serijsku komunikaciju sa računarom i definisati pin 9 kao pin za kontrolu *Servo* motora, Sl. 11.6.3.

```
void setup() {

  Serial.begin(9600);
  myservo.attach(9);


}
```

Sl. 11.6.3.

7. Glavnu petlju ostaviti praznu, a u serialEvent() funkciji obezbediti čitanje nadolazećih karaktera sa serijskog porta i njihovu konverziju u brojeve od 0 do 9, Sl. 11.6.4.
8. Nakon što je čitanje završeno, obezbediti slanje vrenosti ugla *Servo* motoru i ispisivanje iste vrednosti na računaru radi provere.

```
void loop() {

}


void serialEvent() {
  while (Serial.available()) {

    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      angle = inputAngle;
      inputAngle = 0;
      Serial.println(angle);
      myservo.write(angle);
    }
    else if (inChar >= '0' && inChar <= '9') // is inChar a number?
    {
      inputAngle = inputAngle * 10 + inChar - '0'; // yes, accumulate the value
    }

  }
}
```

Sl. 11.6.4.

9. Sačuvati program kao *servoSerial.ino*, povezati *Arduino* na računar i spustiti kôd na ploču.
10. Korišćenjem *Serial Monitor*-a testirati rad programa.
11. **Isključiti** *Arduino* iz računara i rastaviti šemu.

NAPOMENA: Korišćenje biblioteke *Servo.h* onemogućava PWM na pinovima 9 i 10 *Arduino* ploče!

Lekcija 12 – Uvod u *Python* programski jezik

Cilj

Cilj lekcije je da se studenti upoznaju sa:

- *Python* programskim jezikom i programskim okruženjem *Spyder*
- Tipovima podataka koji se koriste u *Python* programskom jeziku
- Petljama i strukturama
- Učitavanjem biblioteka
- Radom sa funkcijama
- Radom sa različitim tipovima datoteka.

12.1 Python i programsko okruženje Spyder

Postoje dve verzije *Python* programskog jezika (*Python* 2.x i 3.x). Između ove dve verzije postoji mali broj razlika (kao npr. funkcija *print*, deljenje celih brojeva…). Veliki broj funkcija koje se koriste za rad sa podacima u *Python*-u se nalazi u bibliotekama (modulima) koji ne dolaze uz osnovnu instalaciju. Postoje različita okruženja za *Python* programski jezik (*Spyder, PyCharm, Canopy…*).

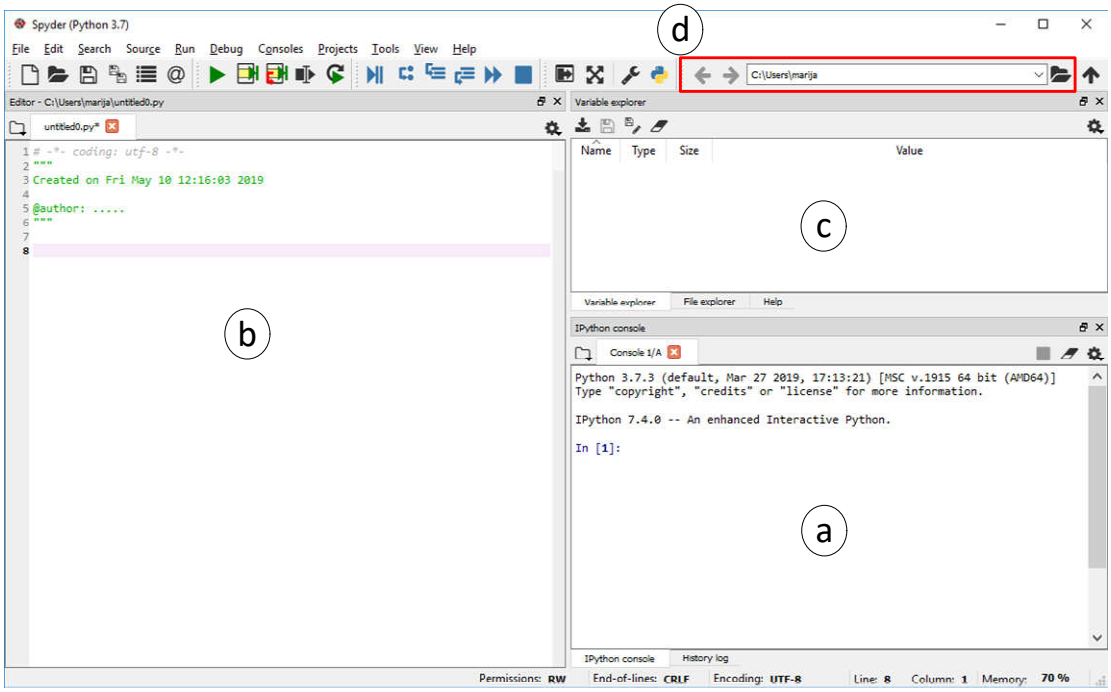
NAPOMENA: Na ovim časovima će se koristiti *Spyder* okruženje i verziju Python 3.7. Anaconda distribuciju zajedno sa *Spyder* okruženjem možete preuzeti na sledećem linku: https://www.anaconda.com/distribution/.

Spyder softver se pokreće izborom opcije: ***Start»All programs» Anaconda3(64-bit)»Spyder(Anaconda3)*** ili klikom na ***Spyder*** prečicu na *Desktop*-u. Inicijalni *Spyder* prozor je prikazan na Sl. 12.1.1.

Osnovni prozor *Spyder* okruženja se sastoji od četiri glavna dela:

a) Konzola – služi za prikaz rezultata izvršavanja *.py* skripte, kao i izvršavanje komandi jednu po jednu. Kucanjem naredbi u konzoli se automatski dobija rezultat izvršavanja.
b) *Editor* – služi za pisanje *.py* skripti sa programskim kôdom. Za razliku od pisanja kôda u konzoli, pisanjem kôda u skripti su lakše izmene kôda, kao i dodavanje delova kôda. Veza između korisnika i skripte se vrši preko konzole, o čemu će biti reči kasnije.
c) *Variable explorer* – služi za pregled svih varijabli koje su se pojavile nakon izvršavanja kôda; *File explorer* – služi za pregled svih datoteka koje se nalaze u trenutnom direktorijumu; *Help* – služi za objašnjenja funkcija. Informacije o funkciji se dobijaju prilikom kucanja komande *help(funkcija)* u konzoli.
d) Putanja ka trenutnom direktorijumu u kome se nalazimo. Ukoliko želimo da pozivamo neke spoljašnje datoteke (.txt, .xslx, .csv) u kôdu, potrebno je da se

putanja ka toj datoteci slaže sa putanjom ka trenutnom direktorijumu. U suprotnom će program prijaviti grešku kako ne može da nađe zadatu datoteku.



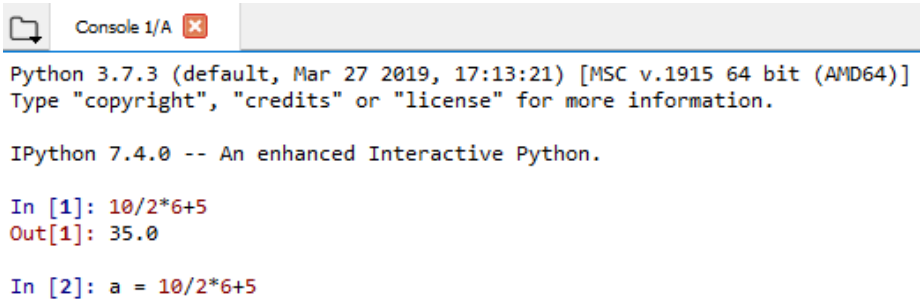
Sl. 12.1.1. *Spyder* programsko okruženje

12.2 Tipovi podataka u Python-u

U *Python* programskom jeziku nije potrebno navoditi tip promenljive, već on ima mogućnost da sam prepozna tip podatka. Osnovni tipovi podataka su:

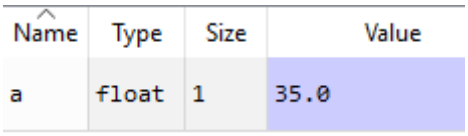
- Logički tip promenljivih (*boolean*)
- Brojevi – dele se u tri grupe: celi brojevi (*integer*), realni brojevi (*float*) i kompleksni brojevi (*complex*). **NAPOMENA**: svi brojevi koji imaju u sebi decimalnu tačku se smatraju realnim brojevima, uključujući i slučaj da se nakon decimalne tačke nalazi 0 (npr. 15.0).
- Sekvence – konačni uređeni skupovi koji mogu da se indeksiraju celim brojevima. U ovu grupu spadaju stringovi, liste i torke.

U *Python* konzoli se direktno može upisati matematički izraz ili izvršiti poziv funkcije. Rezultat izvršavanja komande se može, ali ne mora dodeliti nekoj promenljivoj. Ukoliko se rezultat dodeli promenljivoj, on se neće ispisati u konzoli, Sl. 12.2.1.



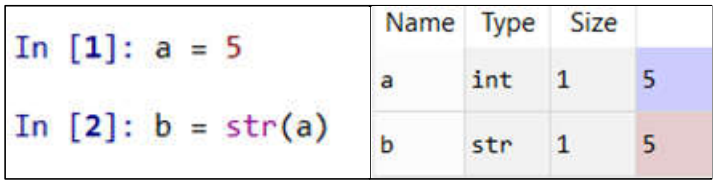
Sl. 12.2.1. Rezultat izvršavanja matematičkog izraza u konzoli programskog okruženja *Spyder*

Otvoriti *Variable explorer*. Primetiti da se u njemu nalazi promenljiva ***a*** na spisku svih promenljivih (Sl. 12.2.2) i da joj je klasifikovana kao realni tip promenljive.



Sl. 12.2.2. Primer prikaza promenljive u *Variable explorer* prozoru

Kao rezultat izvršavanja funkcije *type(x)* se dobija tip promenljive *x*. Moguće je i menjanje tipa promenljive. Funkcijom *cast(x)* promenljiva *x* prelazi u tip *cast,* Sl. 12.2.3.



Sl. 12.2.3. Primer promene tipa promenljive *a*

Zadatak 12.2.1.
Učitati promenljive *x* i *y* u konzoli. Vrednost promenljive *x* neka bude *'zadatak'*, a promenljiva *y* je jednaka 17. U *Variable explorer-u* proveriti tip promenljivih *x* i *y*.

Logičke operacije
Logičke operacije, Tabela 12.2.1 su definisane na operandima koji su tipa *boolean* (mogu da imaju vrednost *True* ili *False*). Rezultat izvršavanja je takođe *boolean*.

Tabela 12.2.1.

Tip operacije	Simbol
Logičko *i*	and
Logičko *ili*	or
Logička negacija	not

Operacije nad brojevima
Nad brojevima je moguće primeniti klasične matematičke operacije, Tabela 12.2.2.

Tabela 12.2.2.

Tip operacije	Simbol
Sabiranje	+
Oduzimanje	-
Množenje	*
Deljenje	/
Celobrojno deljenje	//
Ostatak pri deljenju	%
Stepenovanje	**

U *Python* verziji 2.x ukoliko su oba operanda celi brojevi prilikom deljenja dolazi do celobrojnog deljenja. U *Python* verziji 3.x to nije slučaj, već je potrebno eksplicitno definisati potrebu za celobrojnim deljenjem pomoću odgovarajućeg operatora.

Ostale matematičke funkcije (*sin, cos, sqrt…*) ne pripadaju osnovnom paketu, već je potrebno učitati dodatnu biblioteku.

Nad brojevima je moguće primeniti i operacije poređenja, Tabela 12.2.3.

Tabela 12.2.3.

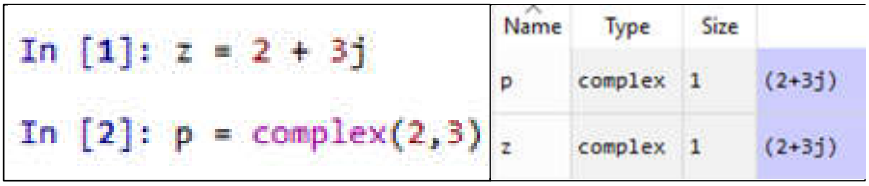
Tip operacije	Simbol
Jednako	==
Nije jednako	<>, !=
Veće od	>
Manje od	<
Veće ili jednako od	>=
Manje ili jednako od	<=

Zadatak 12.2.2.

U konzoli učitati promenljive *b1 = 92*, a *b2 = 13*. Izračunati koliko puta se broj *b2* sadrži u broju *b1* i ostatak pri deljenju broja *b1* sa brojem *b2*.

Kompleksni brojevi
Kompleksni brojevi u *Python* programskom jeziku se mogu definisati na dva načina, Sl. 12.2.4:

i. Eksplicitno pomoću imaginarne jedinice *j*
ii. Pomoću funkcije *complex(realni_deo, imaginarni_deo)*



Sl. 12.2.4. Primeri definisanja kompleksnih brojeva

Osnovne funkcije koje se primenjuju nad kompleksnim brojevima su prikazane u Tabeli 12.2.4.

Tabela 12.2.4.

Tip operacije	**Simbol**
Realni deo	p.real
Imaginarni deo	p.imag
Moduo	abs(p)
Konjugovano-kompleksni broj	p.conjugate()

Zadatak 12.2.3.
U konzoli definisati kompleksni broj $z = 11 + j7$. Izračunati moduo broja *z1* na dva načina: 1) pomoću funkcije *abs()* i 2) pomoću formule $\sqrt{real^2 + imag^2}$. Kvadratni koren izračunati kao stepenovanje sa brojem 0.5.

Liste
Liste predstavljaju niz elemenata odvojenih zarezom koji se nalaze unutar uglastih zagrada, Sl. 12.2.5. Elementi liste mogu biti različitog tipa.

```
In [14]: lista = [2, 'etf', True, 3.5]

In [15]: type(lista)
Out[15]: list
```

Sl. 12.2.5. Primer kreiranja jedne liste

Elementima liste je moguće pristupiti indeksiranjem pomoću uglastih zagrada [].

NAPOMENA: Indeksiranje u *Python*-u počinje od 0!

Moguće je vršiti dodelu vrednosti nekom elementu u listi. Pristupanju elementu liste koji nije još definisan dovodi do greške. Način pristupa i lista operacija nad listama su prikazani u Tabeli 12.2.5 i Tabeli 12.2.6, respektivno.

Liste u *Python*-u se posmatraju kao objekti koje sadrže svoje podatke, metode i funkcije. Metodama i funkcijama se pristupa pomoći operatora „.“.

Tabela 12.2.5.

Pristup elementima liste	**Simbol**
Prvi element liste	lista[0]
i-ti element liste	lista[i]
Poslednji element liste	lista[-1]
Deo liste od *i*-tog do *(k-1)*-tog elementa	lista[i:k]
Deo liste od *i*-tog do poslednjeg elementa	Lista[i:]

Tabela 12.2.6.

Tip operacije	Simbol
Dužina liste	len(lista)
Dodavanje elementa na kraj liste	lista.append(element)
Uklanjanje elementa liste na određenom indeksu	del(lista[indeks])
Uklanjanje određenog elementa iz liste	lista.remove(element)
Sortiranje liste pri čemu se originalna lista menja	lista.sort()
Sortiranje liste pri čemu se originalna lista ne menja	sorted(lista)

Dve liste se mogu međusobno sabrati pomoću operatora +, pri čemu se tada jedna lista nadodaje na drugu. Takođe, jedna lista se može umnožiti nekoliko puta pomoću operatora *, Sl. 12.2.6.

```
In [23]: lista1 = [1, 'etf', True, 3.5]

In [24]: lista2 = [False, 14.65]

In [25]: lista1 + lista2
Out[25]: [1, 'etf', True, 3.5, False, 14.65]

In [26]: lista2*3
Out[26]: [False, 14.65, False, 14.65, False, 14.65]
```

Sl. 12.2.6. Primena operacija sabiranja i množenja nad listama

Zadatak 12.2.4.
U konzoli kreirati tri liste koje sadrže po dva elementa. Svaka lista predstavlja jedno teme trougla, a elementi liste su *x* i *y* koordinata temena. Izračunati stranice trougla pomoću formule $s = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Izračunati površinu trougla pomoću formule $P = \sqrt{s(s-a)(s-b)(s-c)}$, gde je *s* poluobim. Kvadratni koren izračunati kao stepenovanje sa brojem 0.5.

U *Python*-u ime liste pokazuje na objekat te liste. Ukoliko se kreira nova lista na osnovu već postojeće liste (*lista2 = lista1*), tada se samo kreira novi pokazivač ka istom objektu. To znači da sve pomene nad novom listom, utiču na promene stare liste. Korišćenjem naredbe *lista2 = lista1[:]* se kreira nov objekat, tj. vrši se kloniranje liste.

Zadatak 12.2.5.
U konzoli kreirati listu *temperatura* koja sadrži vrednosti *hladno, toplo i vruce*. Kreirati listu *prognoza=temperatura.* Promeniti drugi element liste *prognoza* u *suncano*. Proveriti elemente liste *temperatra*. Kreirati listu *vreme=temperatura[:]*. Promeniti prvi element liste *vreme* u *oblacno*. Pogledati elemente liste *temperatura*.

Stringovi
Stringovi predstavljaju niz simbola koji se nalazi između navodnika (“) ili apostrofa (’). Dva stringa se mogu međusobno spojiti pomoću operatora +. Takođe, string se može umnožiti nekoliko puta pomoću operatora *, Sl. 12.2.7.

```
In [2]: string = 'Zdravo svete!'

In [5]: s = 'ba' + 2*'na'   In [3]: pozdrav = string + ' :)'

In [6]: s                   In [4]: pozdrav
Out[6]: 'banana'            Out[4]: 'Zdravo svete! :)'
```

Sl. 12.2.7. Primer manipulacija sa stringovima

Pomoću funkcije *input(tekst_poruke)* je moguće iz konzole pročitati poruku korisnika. Najpre se korisniku ispisuje *tekst_poruke*, a zatim korisnik treba da upiše njegovu poruku. Korisnikova poruka se čuva u odgovarajućoj promenljivoj kao tip *string,* Sl. 12.2.8. Funkcija *input* blokira izvršavanja ostatka kôda sve dok korisnik ne unese neku poruku. Moguće je ispisati rezultat izvršavanja kôda korisniku pomoću funkcije *print(poruka)*.

```
In [32]: poruka = input('Unesite zeljeni tekst. ')

Unesite zeljeni tekst. praktikum

In [33]: print(poruka)
praktikum
```

Sl. 12.2.8. Primer korišćenja *input* funkcije

Zadatak 12.2.6.
Zatražiti korisniku da unese željenu vrednost površine trougla. Poveriti da li je željena površina veća od površine izračunate u zadatku 12.2.4.

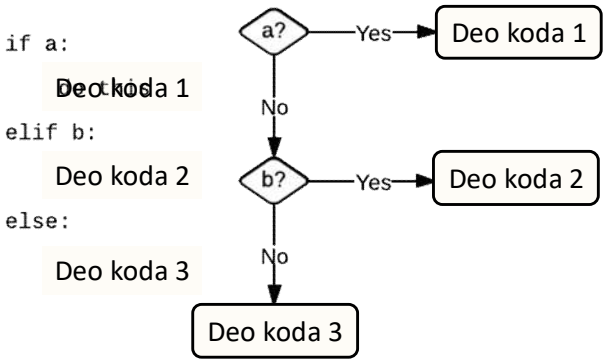
NAPOMENA: obratiti pažnju da funkcija *input* vraća string i da je potrebno izvršiti konverziju tipa podatka.

12.3 Strukture i petlje

Python programski jezik razdvaja blokove pomoću uvlačenja teksta, za razliku od drugih programskih jezika koji koriste ključne reči ili vitičaste zagrade. Na taj način se povećava čitljivost samog kôda. Uvlačenje teksta se primenjuje kod *while* i *for* petlje, *if* strukture, definisanja funkcija…

IF struktura

IF struktura određuje koji deo kôda se izvršava u zavisnosti od uslova, Sl. 12.3.1. Uslov je tipa *boolean*.



Sl. 12.3.1. Princip funkcionisanja *if* strukture

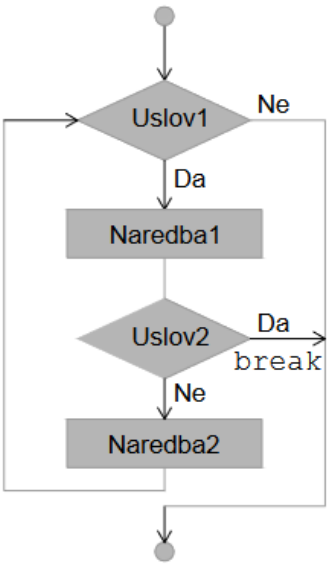
Zadatak 12.3.1.

Napisati program koji na osnovu kategorije avionske karte određuje da li korisnik ima prtljag koji je teži od dozvoljenog.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Korisnik preko konzole treba da unese kategoriju svoje karte (funkcije *input*): 1) K1, dozvoljen prtljag do 8 kg, 2) K2, dozvoljen prtljag do 15 kg i 3) K3, dozvoljen prtljag do 20 kg.
3. Nakon unosa kategorije, od korisnika treba da se traži da unese i težinu njegovog prtljaga.
4. Na osnovu kategorije avionske karte, ispitati da li je težina prtljaga veća od dozvoljene. Ukoliko jeste, ispisati korisniku poruku u konzoli (funkcija *print*).
5. Sačuvati skriptu pod imenom *avion.py*.
6. Pokrenuti i testirati program. *Python* skripta se pokreće iz padajućeg menija *Run»Run* ili prečicom F5 na tastaturi. Moguće je pokretanje i klikom na  .

WHILE petlja

WHILE petlja služi za ponavljanje dela kôda sve dok je ispunjen uslov, Sl. 12.3.2. Uslov je tipa *boolean*. Petlja može da se prekine naredbom *break*, čak i ako je i dalje ispunjen uslov.



Sl. 12.3.2. Tok izvršavanja *while* petlje. Petlja će se završiti ili ukoliko je nije ispunjen *uslov1* ili ukoliko dođe do *break*-a.

Zadatak 12.3.2.
Napisati program u kome korisnik preko tastature unosi niz pozitivnih brojeva. Korisnik treba da unosi broj po broj, sve dok ne unese karakter 'x'. Uneti brojevi se čuvaju u listi. Na kraju izvršavanja programa potrebno je ispisati najveći broj u listi.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Dodati promenljive **broj** sa početnom vrednošću '0' i **lista** sa početnom vrednošću []. Pomoću praznih uglastih zagrada je kreirana prazna lista.
3. Kreirati *while* petlju koja će se ponavljati sve dok promenljiva *broj* ne postane 'x'. Unutar petlje treba konvertovati broj u tip *float*, a zatim ga treba dodati u listu (funkcija *append*) i pitati korisnika da unese novi broj (pomoću funkcije *input*).
4. Nakon završetka *while* petlje potrebno je pronaći najveći broj iz liste. Jedan od načina je pomoću funkcije *max()*.
5. Pomoću funkcije *print* ispisati korisniku koji je najveći broj uneo.
6. Sačuvati skriptu pod imenom *lista.py*.
7. Pokrenuti i testirati program.

FOR petlja
FOR petlja se za razliku od *while* petlje ne izvršava sve dok je ispunjen zadati uslov, već dok god ima elemenata u nekoj datoj sekvenci, Sl. 12.3.3. *For* petlja se takođe može zaustaviti pomoću naredbe *break*.

```
for n in range(5):
    print(n)
```

Sl. 12.3.3. Primer korišćenja *for* petlje

For petlja može da vrši iteracije na više načina. Najčešće se koristi funkcija *range()* koja generiše niz brojeva u zadatom opsegu, Tabela 12.3.1. Sekvenca može biti i u rastućem i u opadajućem poretku.

Tabela. 12.3.1.

Poziv funkcije	Generisana sekvenca
range(n)	0, 1, 2,…, n-1
range(m, n)	m, m+1, m+2,…, n-1
range(m, n, step)	m, m+step, m+2*step,…, n-step

Zadatak 12.3.3.

Napisati program koji ispisuje *n* parnih brojeva.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Od korisnika tražiti da unese broj *n*.
3. Pomoću *for* petlje ispisati sve parne brojeve u konzoli (funkcija *print*). Vršiti iteracije kroz petlju pomoću funkcije *range()*.
4. Sačuvati program pod nazivom *parni_brojevi.py*.
5. Pokrenuti i testirati program.

For petlja može vršiti i iteracije kroz elemente liste, stringa ili niza, Sl. 12.3.4.

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

total = 0
for i in L:
    total += i

print(total)
```

Sl. 12.3.4. Primer *for* petlje koja prolazi kroz sve elemente liste *L* i računa zbir elemenata liste.

12.4 Biblioteke

U *Python* programskom paketu se nalaze samo osnovne funkcije za rad sa podacima. Ukoliko je potrebno vršiti naprednije manipulacije nad promenljivima potrebno je učitati odgovarajuću biblioteku. *Python* poseduje veliki broj biblioteka koje su već oformljene za različite probleme. Najčešće se koriste biblioteke *numpy, matplotlib, scipy, pandas…*

Biblioteke se učitavaju pomoću ključne reči *import*. Postoje različiti načini učitavanja biblioteke, Tabela 12.4.1.

Tabela 12.4.1.

Način uključivanja	Pozivanje funkcija iz biblioteke	Objašnjenje
import numpy	numpy.sin(numpy.pi)	Potrebno je uneti celo ime modula
import numpy as np	np.sin(np.pi)	Funkcije se pozivaju korišćenjem skraćenice np
from numpy import *	sin(pi)	Funkcije postaju deo okruženja pa nije potrebno koristiti ime modula

Dodavanje svih funkcija neke biblioteke kao deo podrazumevanog prostora se ne preporučuje, jer može doći do mešanja funkcija sa istim nazivom iz različitih biblioteka.

Ukoliko biblioteka nije instalirana u okruženju može se instalirati iz komandnog prozora. Iz Start menija otvoriti *Anaconda prompt*. Kucanjem komande *pip install ime_biblioteke* i pritiskom tastera *enter* će se instalirati potrebna biblioteka.

Zadatak 12.4.1.
Napisati program koji ispisuje vrednost funkcije $y = e^x$ za $x \in [a, b]$.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Učitati biblioteku *math* pomoću naredbe *import*
3. Pomoću *for* petlje proći kroz sve vrednosti između *a* i *b*. Samostalno izabrati *a*, *b* i korak (funkcija *range*).
4. Izračunati vrednost funkcije *y* za trenutno *x* pomoću funkcije *exp* iz biblioteke *math*. U svakoj iteraciji petlje prikazati rezultat korisniku.
5. Sačuvati program pod nazivom *eksponencijalna_funkcija.py*.
6. Pokrenuti i testirati program.

Numpy biblioteka
Numpy biblioteka je kreirana za tzv. *Scientific computing*. Omogućava vrlo efikasan rad sa višedimenzionalnim nizovima. Nizovi u *Python*-u podsećaju na liste, ali svi elementi niza moraju biti istog tipa i indeksiranje je moguće samo nenegativnim brojevima.

Nizovi se mogu kreirati na više načina prikazanih u Tabeli 12.4.2.

Tabela 12.4.1.

Tip operacije	Simbol
Kreiranje niza prvog reda	np.array([1, 2, 3])
Kreiranje niza ispunjenog nulama dimenzija *mxn*	np.zeros((m, n))
Kreiranje niza ispunjenog jedinicama dimenzija *mxn*	np.ones((m, n))
Kreiranje niza ispunjenog brojem *b* dimenzija *mxn*	np.full((m, n), b)
Kreira niz brojeva u opsegu *[a, b]*, sa korakom *k*	np.arange(a, b, k)
Kreira *n* brojeva u opsegu *[a, b]*	np.linspace(a, b, n)

Indeksiranje elemenata niza se vrši pomoću uglastih zagrada, Sl. 12.4.1. U primeru je kreirana matrica *a* kao dvodimenzionalni niz.

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

element = a[0, 1]   # Izdvajanje pojedniačnog elemnta
                    # iz prvog reda i druge kolone

# Izdvajanje podniza koji sadrži prva dva reda
# i drugu i treću kolonu; rezultat ima oblik (2, 2)
podniz = a[:2, 1:3]

red = a[1, :]       # Izdvaja samo drugi red matrice a
kolona = a[:, 2]    # Izdvaja samo treću kolonu matrice a
```

Sl. 12.4.1. Primer izdvajanja elemenata niza

Numpy biblioteka koristi osnovne matematičke operacije nad nizovima ili matricama *elementwise*. To znači da primenjuje neku matematičku operaciju na svaki član posebno.

Zadatak 12.4.2.
Napisati program koji računa zbir svih elemenata matrice, kao i srednju vrednost svake kolone.

$$M = \begin{bmatrix} 10 & 5 & 18 & 2 \\ 9 & 17 & 3 & 6 \end{bmatrix}$$

Sl. 12.4.1.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Učitati biblioteku *numpy* pomoću naredbe *import*
3. Kreirati matricu M kao na Sl. 12.4.1.
4. Pomoću funkcije *sum* iz biblioteke *numpy* izračunati sumu svih elemenata matrice M.
5. Pomoću funkcije *mean* iz biblioteke *numpy* izračunati srednju vrednost svake kolone. Funkcija *mean* ima obavezan argument (promenljivu čija se srednja vrednost računa) i opcioni argument (osu po kojoj računa, 0 – uzima svaku kolonu posebnu, 1 – uzima svaki red posebno).
6. Sačuvati program pod nazivom *matrica.py*.
7. Pokrenuti i testirati program.

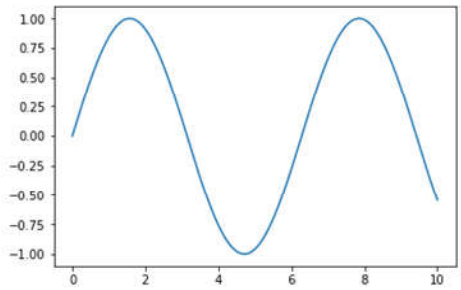
Matplotlib biblioteka

Matplotlib biblioteka se najčešće koristi za grafički prikaz podataka. Da bi podaci mogli da se grafički prikažu potrebno je definisati vektor x-koordinata i vektor

y-koordinata (za 2D slučaj). Prilikom prikaza grafika vektori *x* i *y* moraju biti istih dimenzija.

Naredba *plot* se koristi za crtanje kontinualnih signala. Prvi argument ove funkcije je raspodela *x* ose, a drugi predstavlja vrednosti na *y* osi. Prostor između dve poznate tačke se na grafiku aproksimira pravom linijom. To znači da što je „finija" podela x-ose to će grafik izgledati „glađe". Primer izgleda grafika dobijenog pomoću *plot* funkcije je prikazan na Sl. 12.4.2.

```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.plot(t, np.sin(t))
```



Sl. 12.4.2. Primer generisanja grafika pomoću naredbe *plot* (levo) i odgovarajući grafik (desno)

Naredba *figure* otvara novi prostor za crtanje grafika. Ukoliko se ne pozove ova naredba signali na grafiku će se prikazivati jedan preko drugog.

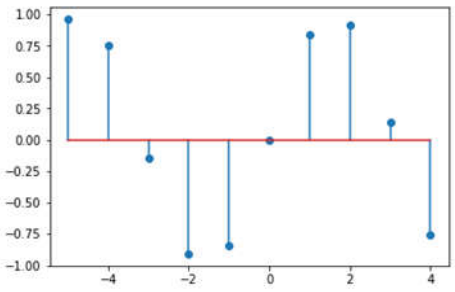
Zadatak 12.4.3

Nacrtati grafik funkcije $x(t) = cos(2\pi f_1 t) + 5sin(2\pi f_2 t)$, ako je f_1=10 *Hz* i f_2=7 *Hz*, a t je u rasponu od 0 do 1 s.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Učitati biblioteke *numpy* i *matplotlib.pyplot* pomoću naredbe *import*
3. Kreirati vremensku osu *t* u opsegu od 0 do 1 s sa odgovarajućem brojem tačaka (koristiti funkciju *linspace*).
4. Formirati signal *x(t)* i prikazati ga na grafiku pomoću funkcije *plot*.
5. Sačuvati program pod nazivom *kosinus_sinus.py*.
6. Pokrenuti i testirati program.

Naredba *stem* se koristi za crtanje diskretnih signala. Pozivanje funkcije je isto kao i kod naredbe *plot*, samo što se susedne tačke ne spajaju. Primer izgleda grafika dobijenog pomoću *stem* funkcije je prikazan na Sl. 12.4.3.

```
import matplotlib.pylab as plt
import numpy as np

n = np.arange(-5, 5, 1)

plt.figure()
plt.stem(n, np.sin(n))
```



Sl. 12.4.3. Primer generisanja grafika pomoću naredbe *stem* (levo) i odgovarajući grafik (desno)

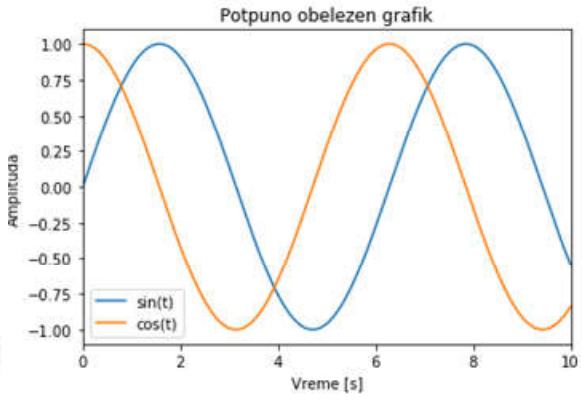
Zadatak 12.4.4.
Nacrtati funkciju $y(n)$ pomoću naredbe *stem*. Promenljive *n* i *y* definisati kao *numpy* nizove.

$$y(n) = \begin{cases} 1, & n = 1 \\ 5, & n = 2 \\ 6.2, & n = 3 \\ -9.5 & n = 4 \\ 11, & n = 5 \\ -3, & n = 6 \\ 0, & ostalo \end{cases}$$

1. U skripti formirati signale *y* i *n*.
2. Kreirati novu figuru. Pomoću naredbe *stem* prikazati signal *y* u zavisnosti od *n*.
3. Pokrenuti i testirati program. Sačuvati program kao *odbirci.py.*

Ova biblioteka omogućava obeležavanje grafika definisanjem naziva osa (*plt.xlabel, plt.ylabel*), dodavanjem naziva grafika (*plt.title*), definisanjem granica *x* i *y* ose u kojima će se prikazivati signal (*plt.xlim, plt.ylim*), dodavanjem mreže (*plt.grid*)…Ukoliko se na istom grafiku prikazuje više signala potrebno je dodati i legendu. Funkcija *legend* treba da ima onoliko argumenata koliko ima različitih signala na grafiku. Primer potpuno definisanog grafika je dat na Sl. 12.4.4.

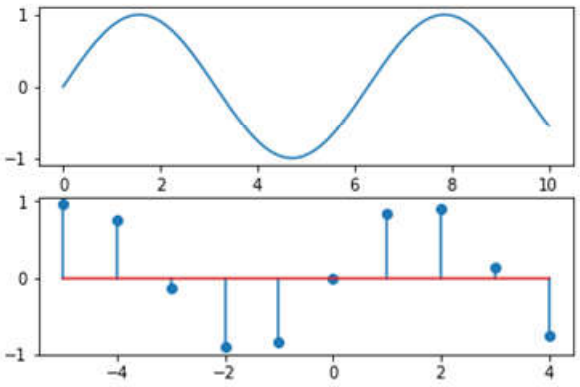
```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.plot(t, np.sin(t))
plt.plot(t, np.cos(t))
plt.legend(('sin(t)','cos(t)'))
plt.xlabel('Vreme [s]')
plt.ylabel('Amplituda')
plt.title('Potpuno obelezen grafik')
plt.xlim((0,10))
```



Sl. 12.4.4. Primer generisanja obeleženog grafika (levo) i odgovarajući grafik (desno)

Pomoću naredbe *subplot* moguće je prikazati više signala na posebnim graficima u okviru jedne figure. Naredba *plt.subplot(m, n, g)* generiše matricu grafika u jednoj figuri dimenzija *mxn* i vraća pokazivač na polje *g*. Primer pozivanja funkcije *subplot* je prikazan na Sl. 12.4.5. Korišćenjem funkcije *subplot* potrebno je zasebno obeležiti svaki grafik.

```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.subplot(2,1,1)
plt.plot(t, np.sin(t))

n = np.arange(-5, 5, 1)

plt.subplot(2,1,2)
plt.stem(n, np.sin(n))
```



Sl. 12.4.4. Primer generisanja grafika pomoću funkcije *subplot* (levo) i odgovarajući grafik (desno)

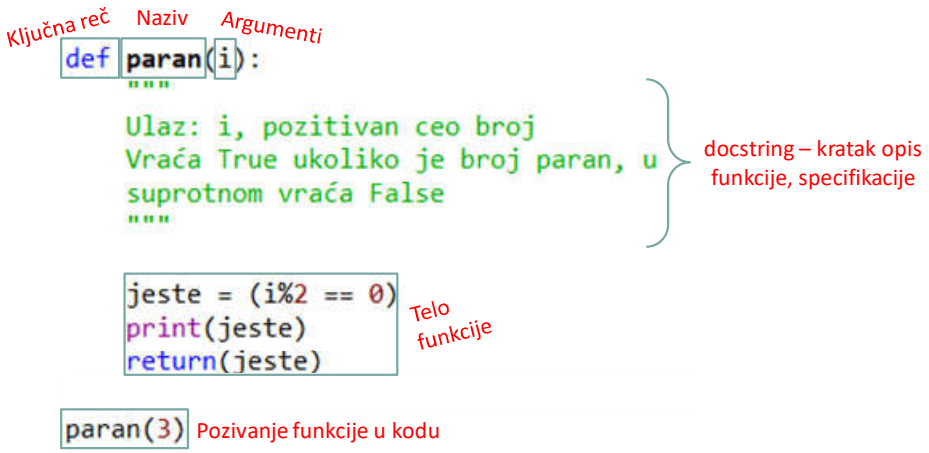
Zadatak 12.4.5. - za samostalni rad

Nacrtati grafike funkcija *sin(t), cos(t), sin(3t)* i *cos(3t)*

4. U editoru kreirati novu *.py* skriptu (*File»New File*).
5. Učitati biblioteke *numpy* i *matplotlib.pyplot* pomoću naredbe *import*
6. Kreirati vremensku osu *t* u opsegu od 0 do 15 s sa odgovarajućem brojem tačaka (koristiti funkciju *linspace*).
7. Pomoću funkcije *subplot* u jednoj figuri prikazati signale *sin(t)* i *sin(3t)* na jednom grafiku, a signale *cos(t)* i *cos(3t)* na drugom grafiku.
8. Obeležiti ose, dodati legendu, dodati naslov grafika.
9. Sačuvati program pod nazivom *crtanje_grafika.py*.
10. Pokrenuti i testirati program.

12.5 Funkcije

Funkcije sadrže delove kôda koji se više puta koriste u glavnom kôdu. Funkcija započinje sa ključnom reči **def** nakon koje se definiše naziv funkcije. Svaka funkcija može, ali ne mora imati ulazne argumente. Zatim, može se definisati *docstring* koji sadrži kratak opis funkcije. Na kraju, nalazi se telo funkcije. Prikaz definisanja funkcije je dat na Sl. 12.5.1.



Sl. 12.5.1. Definisanje funkcije *paran* koja proverava da li je dati broj paran.

Pozivanje funkcije u kôdu se vrši navođenjem njenog imena i potrebnih argumenata.

Zadatak 12.5.1.
Napisati funkciju koja računa elemente Fibonačijevog niza. Fibonačijev niz počinje brojevima 0 i 1, a svaki sledeći član se računa po formuli: $f_n = f_{n-1} + f_{n-2}, n = 3,4,5 \ldots$.

Korisnik treba da zada broj elemenata Fibonačijevog niza koji treba da se izračunaju.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Definisati funkciju *fibonacci* sa jedanim ulaznim argumentom *n* koji označava broj elemenata. U telu funkcije je potrebno definisati promenljivu *niz* tipa liste sa početnom vrednošću [0, 1]. U svakoj iteracije *for* petlje izračunati novi član niza i dodati ga u listu *niz* (funkcija *append*). Nakon završetka *for* petlje vratiti listu *niz* u glavni program (ključna reč *return*).
3. U glavnom delu programa zatražiti od korisnika da unese broj elemenata niza (funkcija *input*), a zatim iskoristiti tu vrednost pri pozivu funkcije.
4. Sačuvati program pod nazivom *fibonacijev_niz.py*.
5. Pokrenuti i testirati program

Argumenti funkcije se dele na obavezne i opcione argumente. Obavezne argumente je potrebno definisati prilikom poziva funkcije, tj. dodeliti im neku vrednost. Ukoliko se prilikom definisanja funkcije nekom argumentu dodeli podrazumevana vrednost tada on postaje opcioni argument. To znači da korisnik može da pozove funkciju bez dodele vrednosti tom argumentu, i u telu funkcije će se koristiti podrazumevana vrednost.

Posmatrajmo primer funkcije koja ima zadatak da izračuna PDV na zadatu cenu. Ulazni parametri funkcije su zadata cena *x* i stopa PDV-a *stopa*. Argumentu *stopa* je dodeljena podrazumevana vrednost 20 (Sl.12.5.2).

```
def pdv(x, stopa = 20):
    """
    x je obavezan argument
    stopa je opcioni argument
    """

    return x*(1 + stopa/100)
```

Sl. 12.5.1. Definisanje funkcije *pdv*

Prilikom pozivanja funkcije može se iskoristi ime argumenta. Na taj način ukoliko postoji veliki broj opcionih argumenata ne moraju se navoditi vrednosti za sve argumente, već samo za one čiju podrazumevanu vrednost želimo da promenimo. Ovo je prednosti u odnosu na druge programske jezike. Sam poziv funkcije se može izvršiti na više načina prikazanih u Tabeli 12.5.1.

Tabela 12.5.1.

Poziv funkcije	**Objašnjenje**
pdv(200)	Početna cena *x* je 200, a *stopa* uzima podrazumevanu vrednost
pdv(200, 10)	Početna cena *x* je 200, a *stopa* je 10
pdv(x = 200, stopa = 10)	Funkcija se poziva preko imena argumenata, redosled navođenja argumenata ne mora biti kao pri definiciji funkcije
pdv(200, stopa = 10)	Ukoliko se neka vrednost argumenata navede bez imena, odmah se dodeljuje obaveznom argumentu. Ukoliko ima više obaveznih argumenata, vrednosti se dodeljuju redom, kao u definiciji funkcije
pdv(200, x = 5)	Dolazi do greške zato što se argumentu *x* dva puta dodeljuje neka vrednost

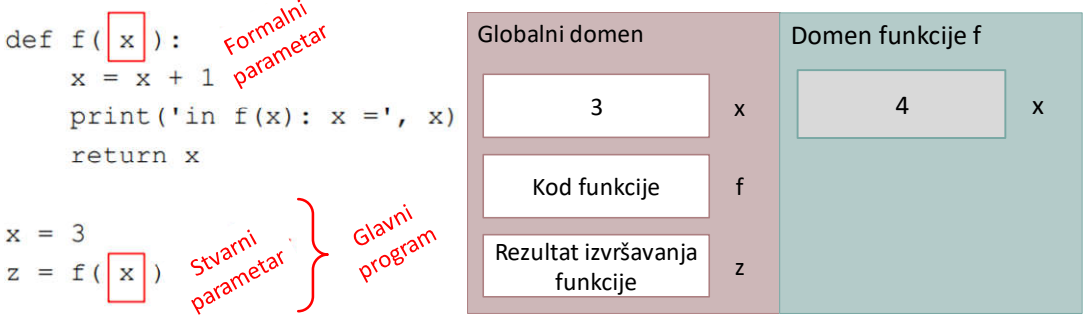
Funkcija može da vraća i više vrednosti tako što se one grupišu pomoću običnih zagrada: *return (a, b)*. Prilikom poziva funkcije potrebno je smestiti rezultat u onoliko promenljivih koliko se vraća iz funkcije, Sl. 12.5.2.

```
import numpy as np

def osobine(niz):
    srednja_vrednost = np.mean(niz)
    suma = np.sum(niz)

    return (srednja_vrednost, suma)

n = np.array([1,5,4,3,2,9,8])
srvr, s = osobine(n)
```

Sl. 12.5.2. Vraćanje više vrednosti iz funkcije

Prilikom pozivanja funkcije javljaju se stvarni i formalni parametri. Stvarni parametri pripadaju globalnom prostoru i oni se nalaze u glavnom programu. Formalni

parametri se vezuju za vrednost stvarnih parametara prilikom poziva funkcije. Unutar funkcije se može pristupiti promenljivama koje su definisane u glavnom delu programa. Prilikom svakog pozivanja funkcije definiše se novi domen, Sl. 12.5.3. Obratiti pažnju da promenljiva *x* postoji i u glavnom kôdu i u funkciji, ali ima različite vrednosti. Ukoliko dođe do preklapanja imena, funkcija ima mogućnost da utiče samo na izmenu promenljive *x* koja je definisana unutar nje.



Sl. 12.5.3. Domen važenja varijabli

Zadatak 12.5.2.
Napisati funkciju koja računa površinu geometrijskog oblika: kvadrat, pravougaonik ili pravougli trougao. Funkcija treba da ima tri argumenta: x (*float*), y (*float*) i tip oblika (*string*). Tip oblika treba da ima podrazumevanu vrednost 'k' za kvadrat, a y ima podrazumevanu vrednost 0. U zavisnosti od tipa geometrijske figure, potrebno je primeniti drugačiji izraz za izračunavanje površine, kao što je to prikazano u Tabeli 12.5.2.

Tabela 12.5.2.

Kvadrat	Pravougaonik	Pravougli trougao
$P = x^2$	$P = x \cdot y$	$P = \frac{x \cdot y}{2}$

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Definisati funkciju *povrsina* sa tri ulazna argumenta kao što je opisao u postavci zadatka.
3. Telo funkcije treba da sadrži računanje površine u zavisnosti od tipa geometrijskog oblika. U glavni program se vraća izračunata površina.
4. U glavnom programu pozvati funkciju i ispisati rezultati izvršavanja korisniku.
5. Sačuvati program pod nazivom *povrsina.py*.
6. Pokrenuti i testirati program za različite kombinacije ulaznih parametara funkcije

12.6 Rad sa datotekama

U zavisnosti od tipa datoteke koju želimo da učitamo u naš program postoje različite biblioteke sa funkcijama za rad sa datotekama. Podaci za obradu su najčešće smešteni u tekstualnim datotekama ili *.csv* datotekama (*comma separated values*).

Prilikom učitavanja datoteka treba obratiti pažnju na to da trenutni direktorijum koji *Spyder* „otvara“ mora da se poklopi sa direktorijumom u kom se nalazi datoteka.

Prilikom definisanja imena datoteke iz koje čitamo podatke potrebno je definisati i ekstenziju, npr. *ime_datoteke = 'proba.txt'*.

Rad sa tekstualnim datotekama

Funkcije za manupulaciju nad tekstualnim datotekama se nalaze u osnovnom paketu instaliranih biblioteka, Tabela 12.5.3.

Tabela 12.5.3.

Poziv funkcije	Objašnjenje
d = open('ime_datoteke', tip)	Otvaranje datoteke. Parametar *tip* može imati vrednosti 'r' (read), 'w' (write) ili 'a' (add). U zavisnosti od tipa, iz datoteke može da se čita sadržaj, da se upisuje sadržaj u nju ili da se doda sadržaj u nju. Podrazumevana vrednost je 'r'.
d.read()	Čitanje celog sadržaja datoteke. Kad se jednom pročita sadržaj, pri ponovnom pozivanju ove funkcije vraća se prazna datoteka. Rezultat ove funkcije je tipa *string*.
d.readline()	Čitanje samo jednog reda tekstualne datoteke. Kada se opet pozove ova funkcija vraća se sledeći red teksta, sve dok se ne pročitaju svi redovi. Rezultat koji vraća ova funkcija je tipa *string*.
d.readlines()	Čitanje svih redova tekstualne datoteke. Rezultat koji vraća ova funkcija je lista stringova, pri čemu svaki element liste odgovara jednom redu teksta.
d.write()	Upisivanje sadržaja u datoteku.
d.close()	Zatvaranje datoteke. Nakon zatvaranja datoteke nije moguće vršiti nikakve manipulacije sa njom.

Kroz datoteku je moguće proći pomoću *for* petlje. U svakoj iteraciji *for* petlje se čita novi red u datoteci, sve dok se ne dođe do kraja datoteke, Sl.12.6.1.

```
In [45]: d = open('proba.txt', 'r')

In [46]: for line in d:
    ...:     print(line)
    ...:
What do you want from me?

Why don't you run from me?

What are you wondering?

What do you know?
```

Sl. 12.6.1. Čitanje teksta iz datoteke

Zadatak 12.6.1.
Napisati program koji kopira sadržaj liniju po liniju iz jedne tekstualne datoteke u drugu.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Podesiti putanju trenutnog direktorijuma u *Spyder* okruženju tako da se slaže sa putanjom ka datotekama.
3. Otvoriti datoteku iz koje će se čitati sadržaj. Otvoriti datoteku u koju će se upisivati sadržaj.
4. Svaki red iz ulazne datoteke kopirati u jedan red u izlaznoj datoteci (koristiti *for* petlju).
5. Po završetku kopiranja, zatvoriti obe datoteke.
6. Sačuvati program pod nazivom *kopiranje_datoteke.py*.
7. Pokrenuti i testirati program za ulaznu tekstualnu datoteku koju samostalno kreirate.

Ukoliko .txt datoteka sadrži samo numeričke podatke može se učitati na lakši način pomoću funkcije *loadtxt* iz biblioteke *numpy* (Sl.12.6.2). Rezultat izvršavanja ove funkcije je tipa *float* ili *numpy* niz *float* vrednosti.

```
In [5]: import numpy as np

In [6]: a = np.loadtxt('datoteka.txt')

In [7]: a
Out[7]:
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]])

In [8]: type(a)
Out[8]: numpy.ndarray
```

Sl. 12.6.2. Primer čitanja numeričkih podataka iz .txt datoteke pomoću funkcije *loadtxt*

Zadatak 12.6.2.
Napisati program koji učitava signal iz datoteke EKG.txt i prikazuje ga na grafiku. U datoteci se nalaze odbirci EKG signala, koji su odabirani sa frekvencijom odabiranja od 100 Hz.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Podesiti putanju trenutnog direktorijuma u *Spyder* okruženju tako da se slaže sa putanjom ka datoteci.
3. Učitati biblioteke *numpy* i *matplotlib.pyplot*
4. Učitati sadržaj datoteke pomoću funkcije *loadtxt*. Kreirati promenljivu *fs* = 100. Kreirati vremensku osu pomoću funkcije *arange*. Vremenska osa treba da ima vrednosti od 0 do *len(signal)/fs* sa korakom 1/fs.
5. Na grafiku prikazati EKG signal u zavisnosti od vremena. Obeležiti ose grafika i dodati naslov grafika.
6. Sačuvati program pod nazivom *EKG.py*.
7. Pokrenuti i testirati program.

Rad sa .csv datotekama

Ponekad se podaci osim u .txt datoteci čuvaju u vidu *Excel* tabele sa ekstenzijom .csv. Funkcije za učitavanje .csv datoteka se nalaze u biblioteci *pandas*.

Datoteke sa ekstenzijom *.csv* u sebi sadrže podatke koji su najčešće razdvojene zarezom. Funkcija koja omogućava čitanje podataka se zove *read_csv*. Ova funkcija ima veliki broj argumenata. Najznačajniji argumenti ove funkcije su *filepath* (putanja do datoteke iz koje želimo da čitamo podatke) i *sep* (separator između kolona. Podrazumevana vrednost je ',').

Kao rezultat, ova funkcija vraća *data frame*. *Data frame* tip podataka je sličan matrici, samo što različite kolone mogu imati različite tipove promenljivih, ali unutar jedne kolone svi podaci moraju biti istog tipa. Takođe, kolone *data frame*-a mogu imati naziv preko koga je moguće pristupiti članovima te kolone pomoću operatora **.**.

Primer korišćenja ove funkcije je dat na Sl. 15.6.3.

```
import pandas

df = pandas.read_csv('analiza.csv', sep=',')
print(df[['ID']])
pol = df.Pol
```

Sl. 12.6.2. Primer čitanja podataka iz .csv datoteke pomoću funkcije *read_csv*

Zadatak 12.6.3.

Napisati program koji učitava signale iz datoteke GSR.csv. Izračunati srednju vrednost i standardnu devijaciju svakog od četiri GSR signala i prikazati podatke na grafiku u vidu *error bar*-a.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Podesiti putanju trenutnog direktorijuma u *Spyder* okruženju tako da se slaže sa putanjom ka datoteci.
3. Učitati biblioteke *numpy, matplotlib.pyplot* i *pandas*
4. Učitati sadržaj datoteke pomoću funkcije *read_csv*. Nakon učitavanja dobijeni *data frame* ima 4 kolone koje predstavljaju različite GSR signale.
5. Pomoću funkcije *mean* iz biblioteke *numpy* izračunati srednju vrednost svake kolone *data frame*-a. Smestiti rezultat u promenljivu *m*.
6. Pomoću funkcije *std* iz biblioteke *numpy* izračunati standardnu devijaciju svake kolone *data frame*-a. Smestiti rezultat u promenljivu *sd*.
7. Kreirati novu figuru. Iz biblioteke *matplotlib.pyplot* iskoristiti funkciju *errorbar* za prikaz greške. *Error bar* je pogodan za vizualizaciju podataka i rezultata u smislu odstupanja od srednje vrednosti. Ova funkcija kao ulazne argumente prima podelu x-ose, vrednosti na y-osi i grešku za svaki odbirak po y-osi. Formirati x-osu tako da ima vrednosti od 1 do 4, za svaki od četiri GSR signala. Argument *y* treba da bude jednak *m*, a argument *yerr* je jednak *sd*. Poziv funkcije je plt.errorbar(x, m, yerr=sd). Moguće je dodati opcioni argument *fmt* sa vrednošću '.' kako se ne bi spojile tačke na grafiku.
8. Obeležiti ose na grafiku i dodati naslov.
9. Sačuvati program pod nazivom *GSR.py*.
10. Pokrenuti i testirati program.

Zadatak 12.6.4. – za samostalni rad

Datoteka *temperatura.csv* sadrži srednju temperaturu u toku godine za vremenski period od 1880. do 2016. godine za dva izvora. Na istom grafiku prikazati zavisnost srednje temperature od godine za oba izvora. Obeležiti ose grafika na adekvatan način i dodati legendu. Izračunati srednju vrednost i standardnu devijaciju za oba izvora posebno. Prikazati rezultat pomoću *error bar*-a.

Lekcija 13 – Kreiranje grafičkog interfejsa u *Python*-u.

Cilj

Cilj lekcije je da se studenti upoznaju sa:

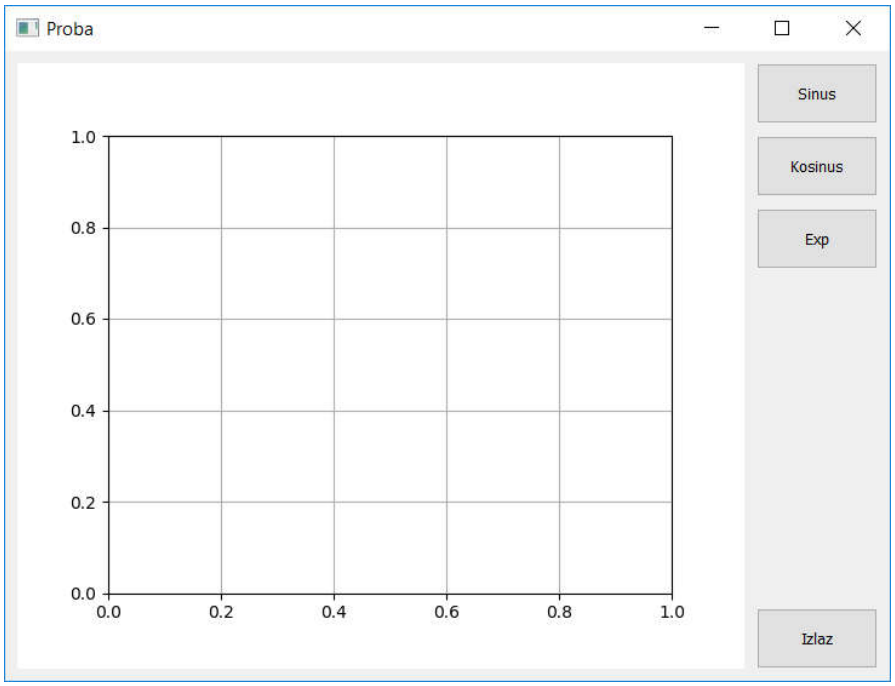
- Kreiranjem grafičkog interfejsa – dodavanje tastera, polja za unos i prikaz teksta, prikaz grafika.

Oprema

- Računar sa instaliranim ***Spyder*** softverskim okruženjem.

13.1 Grafički interfejs u Python-u

U *Python* programskom jeziku se može kreirati grafički interfejs koji omogućava komunikaciju između korisnika i programa. Funkcije za kreiranje grafičkog interfejsa se nalaze u biblioteci **PyQt5**. Primer grafičkog interfejsa je dat na Sl. 13.1.1.

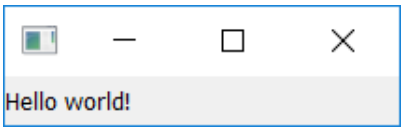


Sl. 13.1.1. Primer grafičkog interfejsa kreiranog u Python programskom jeziku

Zadatak 13.1.1.
Kreiranje jednostavne aplikacije.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).

2. Potrebno je dodati biblioteku *PyQt5* u okruženje. Iz biblioteke je moguće učitati samo funkcije koje su potrebne za program koji se kreira. Iz biblioteke *PyQt5.QtWidgets* učitati funkcije *QApplication* i *Qlabel*.
   `from PyQt5.QtWidgets import QApplication, QLabel`
3. Kreirati aplikaciju komandom `app=QApplication([])`. Svaki *Python* GUI mora da ima tačno jednu instancu aplikacije. Većina delova aplikacije neće raditi ukoliko se ne izvrši ova linija kôda. Ukoliko aplikacija koristi parametre, potrebno je unutar uglastih zagrada ubaciti parametre. Ako ne postoje parametri koji se prosleđuju aplikaciji, ostaviti prazne ove zagrade.
4. Zatim, kreirati tekstualni indikator u aplikaciji. Pomoću funkcije *QLabel* se definiše tekstualna oznaka (labela) na grafičkom interfejsu. Kao argument funkcije se prosleđuje string koji sadrži željeni tekst.
   `label = QLabel('tekst')`
   Da bi se tekstualni indikator pojavio na ekranu potrebno je reći `label.show()`.
5. Ukoliko želimo da se aplikacija izvršava sve dok je korisnik ne zatvori potrebno je dodati komandu `app.exec_()` na kraju programa.
6. Sačuvati skriptu pod imenom *Hello_world.py*.
7. Pokrenuti i testirati program. Aplikacija treba da izgleda kao na Sl. 13.1.2.



Sl. 13.1.2. Rezultat izvršavanja programa iz zadatka 13.1.1.

Zadatak 13.1.2.
Kreirati aplikaciju koja ispisuje poruku u konzoli kada se pritisne taster **Start**.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Iz biblioteke *PyQt5.QtWidgets* učitati funkcije *QApplication* i *QPushButton*.
3. Kreirati *PyQt* aplikaciju: `app = QApplication([])`
4. Dodati taster u aplikaciju. Kao argument funkcije *QPushButton* se prosleđuje tekst koji želimo da piše na samom tasteru.
   `button = QPushButton('Start')`
5. Nakon kreiranja samog tastera, potrebno je dodati i akciju koja se izvršava prilikom pritiska tastera. Akcija se definiše u posebnoj funkciji koju je potrebno kreirati. Kreirati funkciju **button_fcn** koja nema ulaznih argumenata, a u telu funkcije se nalazi ispis poruke korisniku (funkcija *print*).

NAPOMENA: funkcije u Python programskom jeziku se mogu definisati bilo gde u programu, pre pozivanja same funckije. Praksa je da se sve funkcije definišu na početku programa.

6. Pomoću naredbe `button.clicked.connect(button_fcn)` se akcija definisana u funkciji **button_fcn** povezuje sa tasterom **button**.
7. Da bi taster mogao da se vidi u aplikaciji potrebno je dodati komandu `button.show()`.

8. Dodati komandu `app.exec_()` na kraju programa.
9. Sačuvati skriptu pod imenom *ispis_poruke.py*.
10. Pokrenuti i testirati program.

Zadatak 13.1.3.
Kreirati aplikaciju koja ispisuje tekst koji je uneo korisnik kada se pritisne taster **Prikazi**.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Iz biblioteke *PyQt5.QtWidgets* učitati funkcije *QApplication, QPushButton* i *QEditLine*.
3. Kreirati *PyQt* aplikaciju: `app = QApplication([])`
4. Dodati polje za unos teksta u aplikaciju. Kao argument funkcije *QLineEdit* se prosleđuje tekst koji želimo da piše u polju za unos teksta prilikom pokretanja aplikacije. Ova funkcija se može pozvati i bez argumenta.
   `text_input = QLineEdit ('Tekst poruke')`
5. Dodati taster **Prikazi** u aplikaciju.
6. Dodati funkciju koja će se izvršiti prilikom pritiska tastera. Potrebno je ispisati tekst koji je korisnik uneo u polje za upis teksta. Tom tekstu se može pristupiti pomoću naredbe `poruka = text_input.text()`. Kreirati funkciju **button_fcn** koja nema ulaznih argumenata, a u telu funkcije se nalazi ispis **poruke** korisniku (funkcija *print*). Povezati definisanu funkciju sa dugmetom.
7. Dodati komande `button.show()` i `text_input.show()` kako bi se oni prikazali u aplikaciji.
11. Dodati komandu `app.exec_()` na kraju programa.
12. Sačuvati skriptu pod imenom *ispis_teksta.py*.
13. Pokrenuti i testirati program. Izmeniti sadržaj polja za unos teksta i pritisnuti opet taster **Prikazi**.

Primetiti da su se otvorila dva prozora. U jednom se nalazi taster, a u durom polje za unos teksta. Kako *Python* podržava objektno orijentisano programiranje, mnogo bolji način za kreiranje aplikacije je preko klase koja definiše samu aplikaciju.

Zadatak 13.1.4.
Izmeniti zadatak 13.1.3. tako da se aplikacije definiše kao jedna klasa koja sadrži taster, polje za unos teksta i polje za prikaz poruke koju je uneo korisnik, Sl. 13.1.3.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Iz biblioteke *PyQt5.QtWidgets* učitati funkcije *QApplication, QPushButton, QEditLine* i *QWidget*. Dodati i biblioteku *system* kako bi se pristupilo sistemskim podacima.
3. Program počinje od kreiranja aplikacije kojoj se prosleđuju sistemski podaci `app = QApplication(sys.argv)`. Zatim potrebno je kreirati klasu koja sadrži elemente aplikacije. Program se završava kada korisnik izađe iz aplikacije, što je omogućeno pomoću funkcije `app.exec_()`.

4. Definisati klasu **App** kojoj se prosleđuje *QWidget* klasa koja u sebi sadrži funkcije za manipulaciju elementima u aplikaciji. U samoj klasi su definisane metode za definisanje aplikacije.
5. Metoda **__init__** predstavlja konstruktor klase **App**. On se poziva prilikom kreiranja objekta klase u glavnom programu. Ovde se podešava pozicija aplikacije na ekranu, veličina prozora i naziv aplikacije. Veličina i pozicija su izraženi u pikselima, a pozicija se definiše pozicijom gornjeg levog ugla aplikacije. Da bi se definisali ostali elementi, u konstruktoru se poziva funkcija **initUI** za inicijalizaciju korisničkog interfejsa. Različitim metodama i promenljivim unutar klase se pristupa preko ključne reči **self**.
6. U metodi **initUI** se pomoću funkcije `setGeometry` postavlja geometrija aplikacije (veličina i pozicija). Pomoću funkcije `setWindowTitle` dodaje se naziv aplikacije. Unutar ove funkcije je potrebno definisati sve elemente koje aplikacija treba da sadrži.
7. Svi elementi koji se nalaze u aplikaciji se definišu kao i ranije. Moguće je promeniti poziciju i veličinu svakog elementa pomoću funkcija `move()` i `resize()`, respektivno.
8. Funkcija koja se pridružuje tasteru se definiše kao zasebna metoda klase na isti način kao i što je ranije opisano.

14. Da bi se aplikacija prikazala na ekranu potrebno je unutar metode **initUI** napisati naredbu `self.show()`.
15. Sačuvati skriptu pod imenom *ispis_teksta_modif.py*. Pokrenuti i testirati program.

```
from PyQt5.QtWidgets import QApplication, QPushButton, QLineEdit, QLabel, QWidget
import sys

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.title = 'Zadatak 13.1.4'
        self.left = 200
        self.top = 200
        self.width = 120
        self.height = 190
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.left, self.top, self.width, self.height)

        self.text_input = QLineEdit('Tekst', self)
        self.text_input.resize(100, 50)
        self.text_input.move(10,10)

        self.button = QPushButton('Start', self)
        self.button.resize(100,50)
        self.button.move(10,70)
        self.button.clicked.connect(self.button_fcn)

        self.text_output = QLabel(' ', self)
        self.text_output.resize(100, 50)
        self.text_output.move(10,130)

        self.show()

    def button_fcn(self):
        poruka = self.text_input.text()
        self.text_output.setText(poruka)

app = QApplication(sys.argv)
ex = App()
app.exec_()
```

Sl. 13.1.3. Kôd koji realizuje aplikaciju definisanu zadatkom 13.1.4.

Zadatak 13.1.5.
Kreirati aplikaciju koja omogućava da se pritiskom na taster **Prikazi signal**, na grafiku prikazuje 1000 slučajno generisanih odbiraka sa srednjom vrednošću koju unosi korisnik.

1. U editoru kreirati novu *.py* skriptu (*File»New File*).
2. Iz biblioteke *PyQt5.QtWidgets* učitati funkcije *QApplication, QPushButton, QEditLine* i *QWidget*. Dodati biblioteku *system*. Biblioteka *numpy.random* služi za generisanje slučajnih odbiraka (`import numpy.random as r`). Za prikaz grafika u aplikaciji su potrebne funkcije iz biblioteke *matplotlib*, Sl.13.1.4.

```
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
```

Sl. 13.1.4. Učitavanje funkcija za prikaz grafika u aplikaciji

3. U glavnom delu programa se kreira aplikacija i objekat klase **App**. Na kraju glavnog programa se nalazi naredba `app.exec_()`.
4. Potrebno je definisati klasu **PlotCanvas** čiji su objekti grafici koji će biti prikazani u aplikaciji, Sl. 13.1.5.

```
class PlotCanvas(FigureCanvas):

    def __init__(self, parent = None, width = 2, height = 2, dpi = 100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        self.axes.grid()

        FigureCanvas.__init__(self, fig)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self,
                QSizePolicy.Expanding,
                QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def plot(self, x):
        self.axes.cla()
        self.axes.grid()
        self.axes.plot(x, 'o')
        self.draw()
```

Sl. 13.1.4. Učitavanje funkcija za prikaz grafika u aplikaciji

Klasa se sastoji od konstruktora i jedne metode. Slično kao i pri kreiranju objekta aplikacije, u konstruktoru grafika se postavljaju dimenzije samog grafika i definiše se sama figura u kojoj će biti prikazan signal. Promenljiva **axes** definiše osu u odnosu na koju se crta grafik. Unutar jedne figure može biti više osa (*subplot*).
Metoda plot je vezana za objekat klase i služi za prikaz signala na grafiku. Na početku se izbriše sve što se nalazi na grafiku (*axes.cla*), a zatim se pomoću naredbe *axes.plot* definiše signal koji će biti prikazan na grafiku. Naredba *axes.draw* služi za crtanje signala.

5. Definisati klasu **App** kojoj se prosleđuje *QWidget* klasa.

6. U konstruktoru klase **App** (**__init__**) definisati da veličina prozora bude 550x500 piksela, a da se gornji levi ugao prozora nalazi na poziciji (150, 150). Nazvati aplikaciju **Zadatak 13.1.5**.
7. U metodi **initUI** postaviti geometriju aplikacije i njen naziv.
8. Kreirati polje za unos teksta dimenzija 200x75, sa početnom pozicijom u tački (10, 10). Početna vrednost u polju treba da bude 1.
9. Kreirati taster **Prikazi signal** koji ima dimenzije 300x75 piksela, sa početnom pozicijom u tački (240, 10).
10. Kreirati objekat **graphic** klase **PlotCanvas**, Sl. 13.1.5. Grafik treba da ima dimenzije 5.3x3.9, sa početnom pozicijom u tački (10, 100).

```
self.graphic = PlotCanvas(self, width = 5.3, height = 3.9)
self.graphic.move(10, 100)
```

Sl. 13.1.5. Kreiranje objekta klase PlotCanvas

11. Kreirati funkciju **prikaz_odbiraka,** Sl. 13.1.6 koju treba povezati sa tasterom **Prikazi signal**. Prvo treba pročitati koju vrednost je uzeo korisnik. Obratiti pažnju da funkcija *text()* vraća podatak tipa *string* tako da je potrebno izvršiti konverziju u *float*. Slučajno generisanje odbiraka sa normalnom raspodelom se vrši pomoću funkcije *normal* iz biblioteke *numpy.random*. Argumenti funkcije *normal* su srednja vrednost, standardna devijacija i broj odbiraka. `odbirci = r.normal(srednja_vrednost, 1, N)`, pri čemu je N = 1000. Prikazati odbirke na grafiku pozivanjem metode **plot** objekta **graphic**.

```
def prikaz_odbiraka(self):
    srednja_vrednost = float(self.text_input.text())
    N = 1000
    odbirci = r.normal(srednja_vrednost, 1, N)
    self.graphic.plot(x = odbirci)
```

Sl. 13.1.6. Funkcija prikaz_odbiraka koja služi za prikaz slučajno generisanih odbiraka

16. Da bi se aplikacija prikazala na ekranu potrebno je unutar metode initUI napisati naredbu `self.show()`.
17. Sačuvati skriptu pod imenom *prikaz_grafika.py*.
18. Pokrenuti i testirati program.

Zadatak 13.1.6. - za samostalan rad

Kreirati aplikaciju koja će imati četiri tastera, jedno polje za unos teksta, jedno polje za prikaz teksta i grafik. Korisnik treba prvo da unese vrednost u polje za unos teksta. Početna vrednost treba da bude 1. Ta vrednost predstavlja amplitudu signala koji će biti prikazan na grafiku. U zavisnosti od tastera koji je pritisnut prikazuje se jedna od četiri funkcije: $\sin(t)$, $\cos(t)$, $\sin(3t)$ i $\cos(3t)$. U polju za prikaz teksta potrebno je ispisati izabrani signal pomnožen sa zadatom amplitudom. Npr. ako korisnik unese vrednost 2 i izabere prikaz signala $\sin(3t)$ u polju za prikaz teksta će pisati „2*sin(3t)“. Kreiranje vremenske ose je opisano u Lekciji 12.

Lekcija 14 – Serijska komunikacija sa *Arduinom*

Cilj

Cilj lekcije je da se studenti upoznaju sa:

- Serijskom komunikacijom sa *Arduino* mirkokontrolerom
- Obradom izuzetaka.

Oprema

- Računar sa instaliranim ***Spyder*** softverskim okruženjem.
- Računar sa instaliranim ***Arduino IDE*** softverskim okruženjem.
- ***Arduino UNO R3*** ploča.
- Protobord, kratkospojnici, otpornici, NTC termistor, svetleća dioda, RFID, akcelerometar.

14.1 Slanje podataka sa Arduina ka Python okruženju

Programsko okruženje *Arduino IDE* nije veoma zgodno za prikaz signala, i neku kompleksniju obradu signala. Preko serijske komunikacije *Arduino* mikrokontroler može komunicirati sa različitim programskim jezicima (*LabVIEW*, *MATLAB*, *Python*…). U okviru ovog poglavlja će biti prikazana komunikacija između *Arduino* mikrokontrolera i *Python* programskog jezika.

Zadatak 14.1.1.
Kreirati *Arduino* program koji šalje vreme rada mikrokontrolera u sekundama ka serijskom izlazu.

1. Otvoriti *Arduino IDE* programsko okruženje.
2. Uključiti biblioteku *TimerOne.h* izborom opcije *Sketch»Include Library»Add .ZIP Library*. Ukoliko biblioteka nije instalirana na raučnaru pogledati lekciju 11 koja opisuje dodavanje biblioteke u *Arduino IDE*.
3. Definisati promenljive **int brojac** i **bool stanje** koje imaju početne vrednosti 0 i false, respektivno.
4. U **setup** sekciji kôda započeti serijsku komunikaciju. Inicijalizovati prvi tajmer na 1 s korišćenjem funkcije Timer1.initialize(1000000) i aktivirati prekid za ovaj tajmer korišćenjem funkcije Timer1.attachInterrupt(timerIsr).
5. Kreirati ISR pod nazivom **timerIsr**. Unutar **timerIsr** najpre obezbediti da se pri svakom ulasku u taj deo kôda promenljiva **brojac** uveća za 1 i promenljiva **stanje** postavi na true.
6. Unutar glavne petlje pri svakoj iteraciji proveravati vrednost promenljive **stanje**. Samo ukoliko je **stanje** jednako true, na serijskom portu ispisati trenutnu vrednost **brojaca** i vratiti stanje promenljive **stanje** na false.

7. Sačuvati program pod nazivom *vreme.ino*, povezati *Arduino* sa računarom i spustiti kôd na ploču. Proveriti rad programa korišćenjem *Serial Monitor*-a. Zatvoriti *Serial Monitor*.

Zadatak 14.1.2.
Kreirati *Python* program koji čita 30 vrednosti sa serijskog porta i ispisuje ih u konzoli.

1. U editoru *Spyder* programskog okruženja kreirati novu *.py* skriptu (*File»New File*).
2. Funkcije koje služe za serijsku komunikaciju se nalaze u biblioteci *serial*. Dodati biblioteku u program.

NAPOMENA*:* ukoliko biblioteka nije instalirana na računaru, otvoriti Anaconda Command Prompt, ukucati komadu ***pip install pyserial*** i pritisnuti *Enter*. Nakon nekoliko sekundi biblioteka će se instalirati. Osnovne funkcije ove biblioteke su prikazane u Tabeli 14.1.1.

Tabela 14.1.1.

Funkcija	**Objašnjenje**
import serial	Uključivanje biblioteke za serijsku komunikaciju
ser = serial.Serial(port, baudrate)	Otvaranje serijske komunikacije. Argument port označava na koji ulaz je povezan *Arduino*, a argument baudrate označava brzinu serijske komunikacije i mora da se slaže sa brojem koji je definisan u *Arduino IDE*.
ser.read()	Čitanje jednog karaktera sa serijskog ulaza
ser.readline()	Čitanje niza karaktera, do '\n', sa serijskog ulaza
ser.close()	Zatvaranje serijske komunikacije i oslobađanje resursa

3. Otvoriti serijsku komunikaciju. U *Device Manager*-u se može proveriti na kom portu je povezan *Arduino*. Ukoliko je povezan na portu 13, a brzina serijske komunikacije je 9600, otvaranje serijske komunikacije je definisano naredbom: ser = serial.Serial("COM13", 9600).
4. Kreirati promenljivu *i* koja ima vrednost 0.
5. Kreirati *while* petlju u kojoj će se izvršavati kôd sve dok vrednost promenljive *i* ne bude veća od 30. Unutar petlje pročitati vrednost sa serijskog ulaza (*ser.readline()*) i ispisati je u konzoli(*print()*). Na kraju svake iteracije petlje uvećati vrednost promenljive *i* za 1.
6. Nakon završetka *while* petlje zatvoriti serijsku komunikaciju.
7. Sačuvati program pod imenom *citanje_serijskog_porta.py*.
8. Pokrenuti program.

Primetiti da se u konzoli ne ispisuje samo vrednost **brojaca** koja se šalje sa *Arduino* mikrokontrolera, već postoje dodatni karakteri. Kako bi vrednost brojača mogla da se iskoristi u programu, potrebno ju je izdvojiti unutar *Python* programa.

Zadatak 14.1.3.
Dopuniti zadatak 14.1.2. tako da ispisuje samo trenutnu vrednost brojača. Kreirati funkciju **izdvajanje** koja vraća celobrojnu vrednost brojača.

1. Nakon čitanja vrednost sa serijskog ulaza, dekodirati tu vrednost pomoću funkcije *decode()*. Nakon dekodiranje se dobija promenljiva **value** tipa *string*.
   ```
   arduinoValue = s.readline()
   value = arduinoValue.decode()
   ```
2. Izvršiti konverziju promenljive **value** u ceo broj (*int()*) i ispisati tu vrednost u konzoli (*print()*).
3. Sačuvati program kao *citanje_serijskog_porta_brojac.py* i pokrenuti ga.

Primetiti da se sada ispisuje samo vrednost brojača, i da se promenljiva **value** pojavila u *Variable Explorer*-u.

14.2 Obrada izuzetaka

Svaki *Python* program se može nasilno prekinuti pritiskom tastera Ctrl+C na tastaturi tokom izvršavanja programa.

Zadatak 14.2.1.
Pokrenuti program *citanje_serijskog_porta.py*. U toku izvršavanja programa nasilno zaustaviti izvršavanje programa (Ctrl+C). Pokrenuti opet program. Primeti da se pojavila greška koja kaže da je traženi port zauzet. Pošto se aplikacija nije izvršila do kraja, serijski port je ostao otvoren. Ukucati u konzoli naredbu *ser.close()*. Kako je bitno izbeći ovakve situacije, potrebno je uvesti rukovanje izuzecima, Sl. 14.2.1.

```
import time

i = 0
try:
    while (i < 1000):
        print(i)
        i += 1
        time.sleep(1)

except KeyboardInterrupt:
    print('Prekid')
```

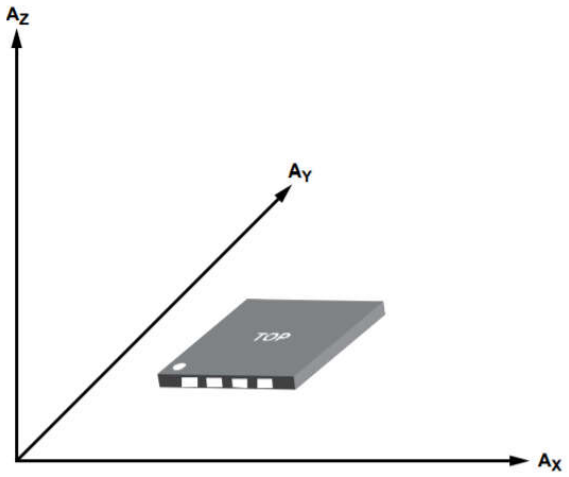
Sl. 14.2.1. Primer rukovanja izuzecima

Program sa Sl. 14.2.1 se izvršava sve dok vrednost brojača *i* ne postane jednaka 1000. U svakoj iteraciji *while* petlje se ispiše trenutna vrednost brojača, uveća za 1, i uvede kašnjenje od 1s. Taj deo kôda se nalazi u *try* grani. Ukoliko dođe do nasilnog prekida programa preko tastature, ispisaće se poruka u konzoli da je došlo do prekida.

Zadatak 14.2.2.
Napisati program koji će ispisivati celobrojnu vrednost koja se čita sa serijskog ulaza. U slučaju prekida programa preko tastature treba da se zatvori serijska komunikacija i ispiše poruka korisniku.

1. U editoru *Spyder* programskog okruženja kreirati novu *.py* skriptu (*File»New File*).
2. Uključiti biblioteku *serial*.
3. Otvoriti serijsku komunikaciju. Obratiti pažnju na definisanje potrebnih parametara – moraju da se slažu sa parametrima iz *Arduino* programa.
4. U *try* grani kreirati *while* petlju čiji je uslov uvek ispunjen (*while True*). Unutar *while* petlje pročitati vrednost koja je stigla na serijski ulaz dekodirati je i konvertovati u celi broj. Ispisati dobijenu celobrojnu vrednost u konzoli.
5. Ukoliko se pojavi izuzetak sa tastature (*except KeyboardInterrupt*) zatvoriti serijsku komunikaciju. Ispisati korisniku poruku da je došlo do prekida programa.
6. Kreirati i treću granu ukoliko se pojavi bilo kakva druga greška (*except*). Program treba da zatvori port i da ispiše korisniku poruku da je došlo do greške.
7. Sačuvati program pod imenom *citanje_i_ispis.py*.
8. Pokrenuti program i prekinuti ga posle nekog vremena preko tastature.

Akcelerometar predstavlja senzor ubrzanja. U najjednostavnijoj izvedbi sadrži inercijalni element čiji pomeraj usled ubrzanja daje informaciju o inercijalnoj sili koja na njega deluje. Danas se akcelerometri mogu realizovati u MEMS (MicroElectroMechanical System) tehnologiji i omogućavaju merenje komponenata ubrzanja duž sve tri ose a_x, a_y i a_z. Na Sl. 14.2.2 je prikazan ADXL335 3-osni akcelerometarski čip sa naznačenim osama duž kojih meri ubrzanje. Oko ovog čipa je izgrađen ceo GY-61 akcelerometarski modul.



Sl. 14.2.2 ADXL335 3-osni akcelerometar sa prikazanim mernim osama u odnosu na kućište čipa

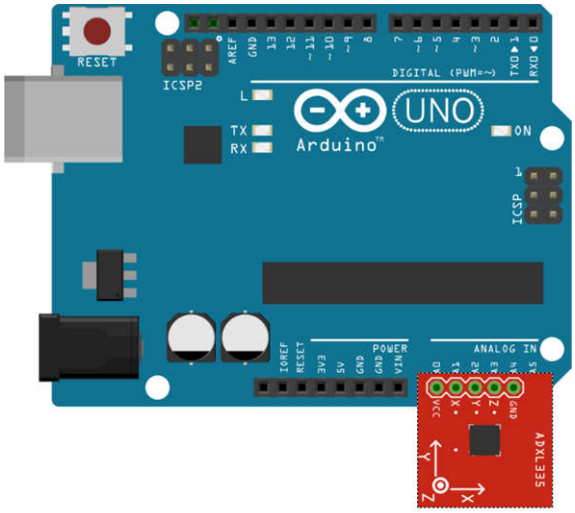
Pri korišćenju akcelerometra je potrebno imati u vidu da će na senzor u svakom trenutku, pored inercijalnog ubrzanja koje je posledica kretanja, delovati i ubrzanje Zemljine teže. Ova činjenica pruža osnov jednom od tipova kalibracije akcelerometara u kome se akcelerometar kontrolisano rotira čime se menja komponenta ubrzanja Zemljine teže koja deluje duž željene ose. Ukoliko poznajemo ugao rotacije, lako možemo povezati ubrzanje koje deluje duž ose sa izlaznim naponom akcelerometra čime se uređaj kalibriše.

Zadatak 14.2.3.
Kreirati *Arduino* program koji čita vrednost x komponente ubrzanja sa GY-61 akcelerometra na svakih 100 ms i šalje je preko serijske komunikacije ka *Python* programu.

1. Otvoriti *Arduino IDE* programsko okruženje.
2. Uključiti biblioteku *TimerOne.h*.
3. Definisati promenljive tipa volatile int **x** i volatile byte **stanje** koje imaju početne vrednosti 0.
4. Definisati konstante VCCPin, xPin, yPin, zPin i GNDPin tipa int koje uzimaju redom vrednosti A0, A1, A2, A3 i A4.
5. U setup sekciji kôda započeti serijsku komunikaciju inicijalizovati prvi tajmer na 100 ms korišćenjem funkcije Timer1.initialize(100000) i aktivirati prekid za ovaj tajmer korišćenjem funkcije Timer1.attachInterrupt(timerIsr).
6. Zatim u ostatku setup sekcije podesiti pinove A0 i A4 kao **IZLAZNE** korišćenjem funkcije pinMode() i njihove vrednosti inicijalizovati redom na HIGH i LOW. Ovaj potez pinovima koji su uobičajeno analogni ulazi menja funkciju, te postaju digitalni izlazi. Vrednost digitalnog pina A0 se zatim postavlja na 5V, a digitalnog pina A4 na GND. Razlog pribegavanju ovakvog

vida rešenja leži u olakšavanju povezivanja GY-61 akcelerometra sa *Arduino* pločom.

7. Kreirati ISR pod nazivom **timerIsr**. Unutar **timerIsr** postaviti promenljivu **stanje** na 1 i pročitati vrednost sa analognog ulaza A1 (x osa akcelerometra).
8. Unutar glavne petlje pri svakoj iteraciji proveravati vrednost promenljive **stanje**. Samo ukoliko je **stanje** jednako 1, ispisati vrednost promenljive **x** na serijskom portu i vratiti stanje promenljive **stanje** na 0.
9. Sačuvati program pod nazivom *akcelerometar_timer.ino*.
10. Povezati kolo kao na Sl. 14.2.3 ubacivanjem pinova akcelerometra u odgovarajuće ulaze *Arduino* ploče.



Sl. 14.2.3 Šema povezivanja GY-61 akcelerometra sa *Arduino* pločom

11. Povezati *Arduino* sa računarom i spustiti kôd na ploču. Proveriti rad programa korišćenjem *Serial Monitor*-a i *Serial Plotter*-a. Rotirati akcelerometar i posmatrati šta se događa sa signalom x ose akcelerometra. Zatvoriti *Serial Monitor* i *Plotter*.
12. **Isključiti** *Arduino* iz računara.

Zadatak 14.2.4.

Napisati *Python* program koji čita podatke koji mu stižu sa serijskog porta. Zatim skalira tu vrednost u opseg od 0 do 5 V i dodaje je u listu. Kada korisnik prekine izvršavanje programa preko tastature prikazati skaliran signal na grafiku.

1. U editoru *Spyder* programskog okruženja kreirati novu *.py* skriptu (*File»New File*).
2. Uključiti biblioteke *serial, numpy* i *matplotlib.pyplot*.
3. Otvoriti serijsku komunikaciju. Obratiti pažnju na definisanje potrebnih parametara – moraju da se slažu sa parametrima iz *Arduino* programa.
4. Kreirati promenljivu **lista** i postaviti joj početnu vrednost na [].
5. U *try* grani kreirati *while* petlju čiji je uslov uvek ispunjen (*while True*). Unutar *while* petlje pročitati vrednost koja je stigla na serijski ulaz dekodirati je i konvertovati u celi broj. Celobrojnu vrednost smestiti u promenljivu **vrednost**.

6. Kako *Arduino* nakon A/D konverzije vraća digitalizovane vrednosti u opsegu od 0 – 1023, potrebno ih je skalirati na realan opseg napona od 0 – 5 V. Promenljivu **vrednost** skalirati na zadati opseg i dodati je u listu (funkcija *append*).
7. Ukoliko se pojavi izuzetak sa tastature (*except KeyboardInterrupt*) zatvoriti serijsku komunikaciju. Kreirati promenljivu *fs* = 10. Kreirati vremensku osu pomoću funkcije *arange*. Vremenska osa treba da ima vrednosti od 0 do *len(signal)/fs* sa korakom 1/fs. Prikazati odbirke smeštene u promenljivoj **lista** u zavisnosti od vremena na grafiku.
8. Kreirati i treću granu ukoliko se pojavi bilo kakva druga greška (*except*). Program treba da zatvori port i da ispiše korisniku poruku da je došlo do greške.
9. Sačuvati program pod imenom *citanje_skaliranje.py*.
10. Pokrenuti i testirati program.

14.3 Slanje podataka iz Python okruženja ka Arduinu

Zadatak 14.3.1.
Kreirati *Arduino* program koji čita informacije sa serijskog porta. U zavisnosti da li pročita ON ili OFF treba da uključi, odnosno isključi svetleću diodu.

1. Otvoriti *Arduino IDE* programsko okruženje.
2. Definisati promenljive **inputString** i **operation** tipa String i String respektivno. Podesiti inicijalne vrednosti promenljivih **inputString** i **operation** na prazan string ("").
3. U setup delu programa započeti serijsku komunikaciju sa računarom korišćenjem funkcije Serial.begin(9600) i podesiti pin na kome se nalazi ugrađena svetleća dioda kao izlazni korišćenjem funkcije pinMode(LED_BUILTIN, OUTPUT).
4. Kreirati novu funkciju serialEvent() kao na Sl. 11.1 (lekcija 11).
5. Unutar glavne petlje pri svakoj iteraciji proveravati vrednost promenljive **operation**. Ukoliko je vrednost promenljive jednaka "ON" uključiti diodu, a ako je "OFF" isključiti diodu.
6. Sačuvati program pod nazivom *ON_OFF.ino*.
7. Povezati *Arduino* sa računarom i spustiti kôd na ploču.

Zadatak 14.3.2.
Napisati *Python* program koji čita šta korisnik unese preko konzole i šalje pročitanu vrednost preko serijske komunikacije ka *Arduino* mikrokontroleru.

1. U editoru *Spyder* programskog okruženja kreirati novu *.py* skriptu (*File»New File*).
2. Uključiti biblioteke *serial* i *time*.
3. Otvoriti serijsku komunikaciju. Obratiti pažnju na definisanje potrebnih parametara – moraju da se slažu sa parametrima iz *Arduino* programa.
4. Kreirati *while* petlju čiji je uslov uvek ispunjen (*while True*). Unutar *while* petlje pitati korisnika da unese poruku (funkcija *input*). Korisnik treba da unese ON da bi uključio diodu i OFF da bi isključio diodu. Ukoliko korisnik unese STOP treba zatvoriti serijsku komunikaciju i prekinuti petlju (naredba *break*). Na poruku koju je uneo korisnik dodati karakter za novi red '\n' (spajanje stringova je moguće pomoću operatora +). Preko funkcije *ser.write(poruka.encode())* poslati informaciju ka serijskom portu.
5. Dodati kašnjenje unutar *while* petlje pomoću funkcije *sleep* iz biblioteke *time*. time.sleep(1) – kašnjenje od 1 s
6. Sačuvati program pod imenom *upis.py*.
7. Pokrenuti i testirati program.

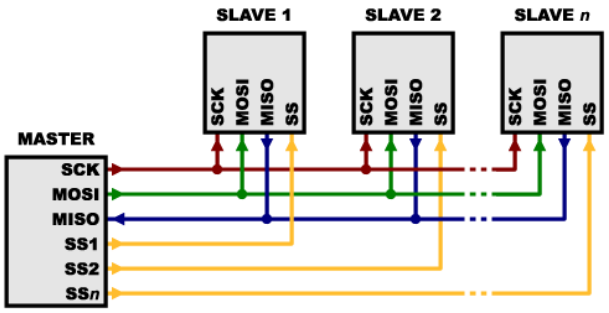
Zadatak 14.3.3 - za samostalan rad
Kreirati aplikaciju koja kontinualno čita vrednost napona sa NTC termistora i ako ta vrednost pređe neki prag koji korisnik unese treba da se uključi svetleća dioda. *Arduino* program treba da čita napona na analognog ulaza i šalje ga ka serijskom portu. Takođe, u zavisnosti od poruke koju primi sa serijskog porta treba da uključi/isključi diodu. *Python* program treba da pita korisnika da unese prag napona.

Zatim, treba da kontinualno (unutar *while* petlje) čita vrednosti sa serijskog porta, skalira dobijenu vrednost i poredi je sa pragom. Ako je vrednost napona veća od praga, treba poslati komandu ON ka *Arduino* mikrokontroleru, a ako je manja onda treba poslati komandu OFF. Program se zaustavlja preko tastature. Obezbediti adekvatno zatvaranje resursa.

14.4 Komunikacija Arduino ploče sa RFID – RC522 modulom

RFID je skraćenica engleskog termina *Radio-Frequency IDentification*, sistema za razmenu podataka bez direktnog kontakta na malim rastojanjima. Ono što ovaj vid komunikacije čini posebno interesantnim je činjenica da ostvarivanje komunikacije ne zahteva dva aktivna uređaja, već je dovoljno da samo jedan uređaj poseduje izvor energije, dok drugi uređaj može biti potpuno pasivan (kartice za plaćanje ili identifikaciju, Bus-Plus i mnogo drugih primera.). Aktivan uređaj svojom antenom emituje RF signale definisane učestanosti. Ovi signali imaju dvostruku ulogu u ostvarivanju RFID komunikacije sa pasivnim uređajem, tj. služe i za komunikaciju, ali i kao izvor energije pasivnoj kartici na kojoj se nalaze podaci. Jedna od prednosti koje RFID pruža leži upravo u tome što kartica na kojoj se nalaze podaci ne zahteva sopstveno baterijsko napajanje, te ne postoji potreba za zamenom baterija, a životni vek kartice može biti veoma dug.

RFID modul koji se koristi u ovoj vežbi **radi na 3.3 V**. Komunikacija između modula i *Arduino* ploče se ostvaruje putem SPI (engl. *Serial Peripheral Interface*) interfejsa. SPI predstava sinhroni (za razliku od klasičnog serijskog porta) serijski protokol za razmenu podataka između mikrokontrolera i perifernih uređaja ili više mikrokontrolera. SPI komunikacija zahteva postojanje jednog *Master* uređaja (obično mikrokontroler, kod nas *Arduino*) i bar jednog (a može i više) *Slave* uređaja (periferije, u ovom poglavlju RFID modul). Sinhronizacija komunikacije se obezbeđuje postojanjem *Clock* signala na zasebnom pinu SCK. Slanje podataka sa *Master* uređaja na *Slave* uređaj se vrši preko konekcije MOSI (*Master Out Slave In*), dok se podaci sa *Slave* uređaja na *Master* uređaj šalju putem MISO (*Master In Slave Out*) konekcije. Pored tri navedene linije (MOSI, MISO i SCK), postoji i četvrta linije *Slave Select* (SS) koja omogućava izbor *Slave* uređaja, tj. signalizira *Slave* uređaju da *Master* sa njim komunicira. Ovo omogućava kontrolu više periferija preko istog SPI porta. Tipičan primer takve konfiguracije je prikazan na Sl. 14.4.1.



Sl. 14.4.1 Povezivanje više *Slave* uređaja na jedan *Master* uređaj putem SPI interfejsa. Slika je preuzeta sa sajta *learn.sparkfun.com*.

Pinovi za SPI komunikaciju se razlikuju u zavisnosti od tipa ploče. *Arduino UNO R3* ploča poseduje pinout prikazan u Tabeli 14.4.1.

Tabela 14.4.1.

Ime linije	Broj pina
MOSI	11
MISO	12
SCK	13

Prednosti SPI interfejsa uključuju brzinu prenosa, jednostavnost prijemnog hardvera, kao i mogućnost slanja podataka na više uređaja preko istog porta, dok se mane ovog vida komunikacije uglavnom odnose na broj potrebnih linija za komunikaciju, posebno u slučaju više *Slave*-ova, i na činjenicu da *Master* mora posredovati svakoj komunikaciji između *Slave*-ova.
Modul koji se koristi za RFID komunikaciju u ovom poglavlju komunicira na učestanosti od 13.56 MHz. Kartica i privezak za ključeve koji dolaze uz modul poseduju sopstveni identifikacioni broj, kao i memoriju od 1024 bajta podeljenih u 16 sektora, koja je dostupna za skladištenje podataka na kartici.

Zadatak 14.4.1.
Kreirati program koji očitava identifikacioni broj sa kartice i ispisuje ga na računaru.

1. Pokrenuti *Arduino IDE* okruženje i u *Sketch* uključiti biblioteku SPI.h koja dolazi zajedno sa okruženjem.
2. Sa linka https://github.com/miguelbalboa/rfid preuzeti biblioteku u formi .ZIP datoteke, a zatim je uključiti u *Sketch*.
3. Korišćenjem #define definisati SS_PIN i RST_PIN kao pinove 10 i 9 respektivno.
4. Kreirati MFRC522 instancu, Sl. 14.4.2. Pinovi MOSI i MISO su automatski podešeni.

```
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

Sl. 14.4.2

5. Unutar setup dela kôda obezbediti inicijalizaciju serijske komunikacije kao i u ranijim primerima, kao i inicijalizaciju SPI komunikacije pozivanjem SPI.begin().
6. Inicijalizovati MFRC522 modul pozivanjem mfrc522.PCD_Init().
7. Poslednje što je potrebno uraditi u ovom delu kôda je obezbediti slanje poruke korisniku da je inicijalizacija prošla i da je potrebno približi karticu čitaču.
8. Unutar loop dela programa je potrebno najpre proveriti da li je nova kartica prislonjena čitaču i proveriti da li je moguće dobiti njen ID. Ukoliko nije, potrebno je prekinuti datu iteraciju petlje, Sl. 14.4.3.

```
// Da li ima novih kartica?
if ( ! mfrc522.PICC_IsNewCardPresent())
{
  return;
}
// Proverava da li je moguće pročitati ID kartice
if ( ! mfrc522.PICC_ReadCardSerial())
{
  return;
}
```

Sl. 14.4.3

9. Ukoliko je prethodna sekcija kôda uspešno izvršena i iteracija nije prekinuta potrebno je ispisati korisniku ID prislonjene kartice u formi „ID kartice je:“, zatim ID kartice i nakon toga preći u novi red. Kako bi se ispisao ID kartice,

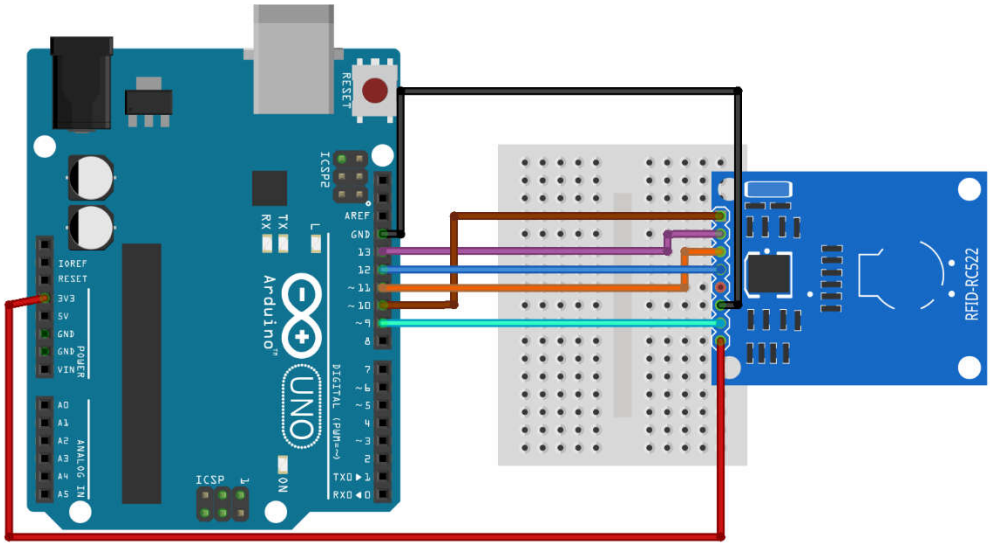
najpre je potrebno kreirati promenljivu String tipa inicijalizovanu kao prazan string koju možemo nazvati **content**.

10. Zatim je potrebno redom pročitati sva četiri bajta identifikacionog broja korišćenjem funkcije mfrc522.uid.uidByte[], pretvoriti ih u String i nadovezati ih korišćenjem funkcije concat, Sl. 4.4.4. Ispisivanje brojeva u heksadecimalnom obliku se obezbeđuje drugim argumentom funkcije String() prikazanog kôda. Brojevi su razdvojeni razmakom kako bi se lakše pročitali.

```
for (byte i = 0; i < mfrc522.uid.size; i++)
{
  content.concat(String(mfrc522.uid.uidByte[i], HEX));
  content.concat(" ");
}
```

Sl. 14.4.4

11. Dobijeni String **content** ispisati na računaru.
12. Povezati **SAMO** *Arduino* ploču sa računarom i spustiti kôd koji je prethodno napisan i sačuvan. (Ova stavka je bitna pre povezivanja RFID modula sa pločom kako se pre spuštanja kôda na ploču RFID modul ne bi spalio usled nekog ranijeg kôda koji se već nalazi na ploči).
13. **Isključiti** *Arduino* iz računara i zatim ga povezati sa RFID modulom prema Sl. 14.4.5.



Sl. 14.4.5. Šema prema kojoj je potrebno povezati RFID modul sa *Arduino* pločom

Voditi računa da modul radi na **3.3V** te mu je napajanje potrebno dovesti sa 3.3V pina na *Arduino* ploči! Ukoliko se pogreši pri povezivanju i modul se poveže na 5V postoji realna šansa da će modul biti trajno oštećen! Radi provere, u Tabeli 14.4.2 su izlistani nazivi pinova sa odgovarajućim brojevima.

Tabela 14.4.2.

SDA	SCK	MOSI	MISO	IRQ	GND	RST	3.3V
10	13	11	12	N.C.	GND	9	3.3V

14. Povezati *Arduino* sa računarom i otvoriti *Serial Monitor*. Testirati rad programa prinošenjem kartice i/ili priveska čitaču i zapisati njihove identifikacione brojeve na papiru jer će biti korišćeni u narednom zadatku.
15. Sačuvati program kao *citanje_ID_kartice.ino*.
16. **Isključiti** *Arduino* iz računara.

Zadatak 14.4.2.
Kreirati *Python* program koji čita podatke sa serijskog porta. Na serijski port stiže ID kartice. Poveriti da li se pročitan ID poklapa sa predefinisanom vrednošću. Ispisati poruku korisniku da li je došlo do poklapanja.

1. U editoru *Spyder* programskog okruženja kreirati novu *.py* skriptu (*File»New File*).
2. Uključiti biblioteke *serial*.
3. Otvoriti serijsku komunikaciju. Obratiti pažnju na definisanje potrebnih parametara – moraju da se slažu sa parametrima iz Arduino programa.
4. Kreirati promenljivu **ID** i postaviti joj početnu vrednost na željeni ID kartice (string koji može, a i ne mora da se slaže sa ID vaše kartice).
5. U *try* grani kreirati *while* petlju čiji je uslov uvek ispunjen (*while True*). Unutar *while* petlje pročitati vrednost koja je stigla na serijski ulaz, dekodirati je i smestiti u promenljivu **ID_tren**.
6. Porediti string **ID** sa **ID_tren**. Ukoliko se vrednost poklapaju ispisati korisniku poruku da je operacija uspešna, a ukoliko se ne poklapaju obavestiti ga da operacija nije uspešna.
7. Ukoliko se pojavi izuzetak sa tastature (*except KeyboardInterrupt*) ili bilo kakva druga greška zatvoriti serijsku komunikaciju.
8. Sačuvati program pod imenom *ID_provera_predefinisano.py*.
9. Pokrenuti i testirati program.

14.5 Kreiranje grafičkog interfejsa - multithreading

Kreiranje grafičkog interfejsa podrazumeva da se klasa vezana za aplikaciju kontinualno izvršava sve dok je korisnik ne zatvori. To znači, da izvršavanje kôda koji nije definisano komandama na interfejsu nije moguće. Pošto komunikacija sa serijskim portom mora da se izvršava kontinualno, uz izvršavanje aplikacije, potrebno je uvesti niti. Niti omogućavaju paralelno izvršavanje programa.

Funkcije za manipulaciju sa nitima se nalaze u biblioteci *threading*. Nit se kreira pomoću funkcije *Thread*. Kao argument ove funkcije se prosleđuje funkcija koju želimo da izvršavamo paralelno:

```
nit = threading.Thread(funkcija)
```

Prilikom kreiranja niti se kreira objekat klase. Nit se pokreće pomoću metode *start*:

```
nit.start()
```

Prilikom pokretanja niti, kôd napisan u telu funkcije izvršava se samo jedanput. Ukoliko je potrebno kontinualno izvršavanje kôda, u telu funkcije se mora koristiti *while* petlja.

Zadatak 14.5.1

Dopuniti dati *Python* kôd koji proverava da li se ID očitane kartice slaže sa tekstom koju je korisnik uneo preko aplikacije.

1. U *Spyder* okruženju otvoriti *ID_kartice_GUI.py* skriptu. U skripti se nalazi kôd koji definiše izgled grafičkog interfejsa.
2. U glavnom delu programa otvoriti serijsku komunikaciju. Obratiti pažnju na definisanje potrebnih parametara – moraju da se slažu sa parametrima iz *Arduino* programa.
3. Definisati dve funkcije **fcn1** i **fcn2**.
4. U telu **fcn1** kreirati globalnu promenljivu **data**. Pročitati vrednost sa serijskog ulaza, dekodirati je i smestiti u promenljivu **data,** Sl. 14.5.1.

```
def fun1():
    global data
    while(True):
        data = ser.readline().decode()
```

Sl. 14.5.1 Funkcija **fcn1**

5. U telu **fcn2** kreirati aplikaciju, Sl. 14.5.2.

```
def fun2():
    app = QApplication(sys.argv)
    ex = App()
    app.exec_()
```

Sl. 14.5.2 Funkcija **fcn2**

6. U funkciji **initUI** klase **App** kreirati globalnu promenljivu data (*global data)*. U funkciji koja je vezana za pritisak tastera **button_fcn** dodati poređenje promenljive **data** sa tekstom koji je korisnik uneo u polje za unos teksta (**poruka**).

7. Ukoliko se poklapaju ove dve vrednosti ispisati poruku u polje za ispis teksta „OK“. Ukoliko se ne poklapaju ispisati „FAIL“.
8. Nakon otvaranja serijskog porta, i definisanja potrebnih funkcija, kreirati i startovati dve niti.
9. Sačuvati program kao *ID_provera_tekst.py*.

```
t1 = Thread(target=fun1)
t2 = Thread(target=fun2)

t1.start()
t2.start()
```

Sl. 14.5.3 Kreiranje i pokretanje niti

Zadatak 14.5.2. - za samostalan rad

Kreirati *Python* aplikaciju koja proverava da li se ID očitane kartice nalazi u tekstualnoj datoteci. Ukoliko se nalazi ispisati korisniku poruku da mu je ulazak odobren i šalje se poruka ka *Arduino* mikrokontroleru da uključi svetleću diodu (šalje se poruka „ON“). Ukoliko se ne nalazi, korisniku se ispisuje poruka da mu nije odobren ulazak i dioda treba da se treperi sa učestanošću 1 Hz (šalje se poruka „ALARM“). Napisati i odgovarajući *Arduino* kôd.

Literatura

[1] S. Tumanski, *Principles of Electrical Measurement*, CRC Press, 2006.

[2] B. Ehsani, *Data Acquisition using LabVIEW*, Packt Publishing, 2016.

[3] Thomas Bress, *Effective LabVIEW programming*, National Technology and Science, 2013.

[4] M. Nawrocki Rick Bitter, T. Mohiuddin. *LabVIEW Advanced Programming Techniques*, CRC Press, Taylor and Francis Group, second edition, 2007.

[5] *Labview™ Core 1&2 Participant Guide*, National Instruments Corporation, 2015.

[6] C. Steger, M. Ulrich, C. Wiedemann, *Machine Vision Algorithms and Applications*, Wiley-VCH, 2018.

[7] C. Demant, B. Streicher-Abel, C. Garnica, *Industrial Image Processing - Visual Quality Control in Manufacturing*, Springer, 2013.

[8] K.S. Kwon, S. Ready, *Practical Guide to Machine Vision Software: An Introduction with LabVIEW*, Wiley-VCH, 2015.

[9] R. Gonzales, R. Woods, *Digital Image Processing*, 4th Ed., Pearson, 2018.

[10] A. Kaehler, G. Bradski, *Learning OpenCV 3*, O'Reilly Media, 2017.

[11] J. Blum, *Exploring Arduino: Tools and Techniques for Engineering Wizardry*, John Wiley & Sons, 2013.

[12] M. Margolis, *Arduino Cookbook*, 2nd Edition, O'Reilly Media, Inc., 2011.

[13] S. Knight, *Arduino for Beginners: Step-By-Step Guide to Arduino (Arduino Hardware & Software)*, Amazon Digital Services LLC - Kdp Print Us, 2018.

[14] S. Monk, *Programming Arduino: Getting Started with Sketches*, Second Edition, McGraw-Hill Education, 2016.

[15] S. Monk, *Programming Arduino Next Steps: Going Further with Sketches*, McGraw Hill Professional, 2013.

[16] M. Lutz, *Learning Python*, 4th Edition, O'Reilly Media, 2009.

[17] A. Downey, *Think Python - How to Think Like a Computer Scientist*, Green Tea Press, 2012.

[18] M. Summerfield, *Rapid GUI Programming with Python and Qt*", Prentice Hall, 2007.

[19] P. Desai, *Python Programming for Arduino*, Packt Publishing, 2015.

[20] J. T. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*, O'Reilly Media, 2016.