Itasdi

Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems

Praktikum iz merno-akvizicionih sistema

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

Itasdi

Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems

Faculty of Technical Sciences

Ss. Cyril and Methodius University
Faculty of Electrical Engineering and Information Technologies

Zagreb University of Applied Sciences

School of Electrical Engineering
University of Belgrade

Faculty of Physics
Warsaw University of Technology

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

Lekcija 5

Sekvencijalno programiranje, Mašina stanja, Promenljive, Razmena podataka između više paralelnih petlji

deo

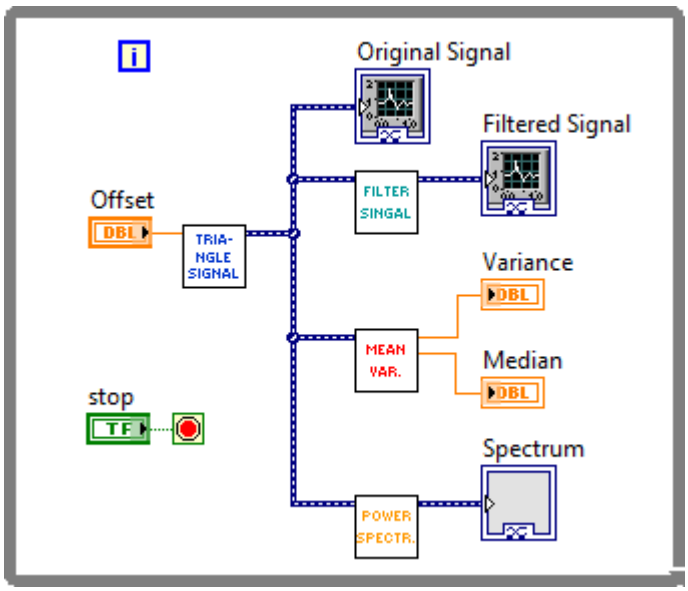
1. Cilj vežbe

Cilj vežbe je da studente upozna sa:

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

Zadatak 7.1.

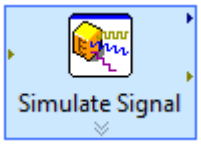
Na slici 7.1 prikazan je blok dijagram programa koji je predstavlja inicijalno napisani kod. Neophodno je realizovati kod za svaki od predstavljenih SubVI. Zatim potrebno je obezbediti da se nakon simulacije signala prvo izvršava određivanje srednja vrednost i standardno odstupanje signala, potom spektar snage signala i na kraju 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*.



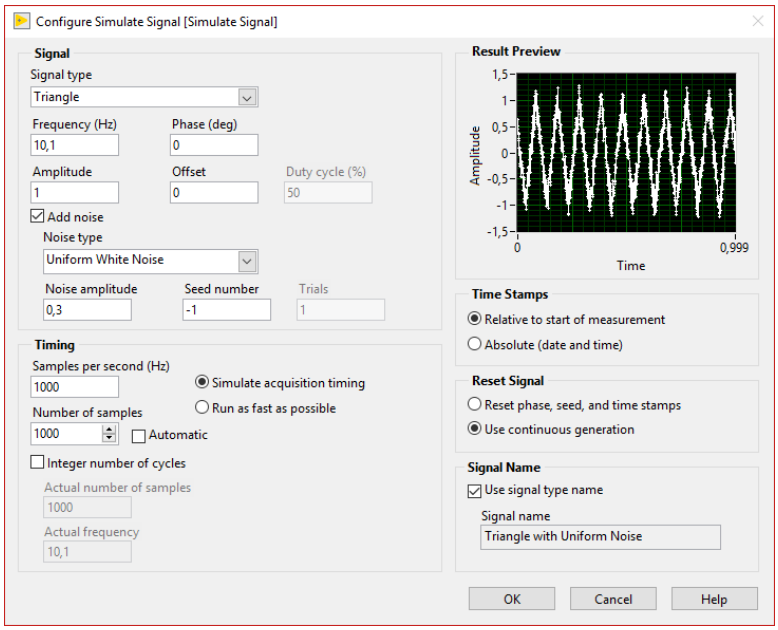
Sl. 7.1.

Uputstvo:

1. Otvoriti novi .vi program. Ubaciti vi *Simulate Signal*, slika 7.2a. Kada se otvori dijagram za podešavanje podesiti parametre kao na slici 7.2b

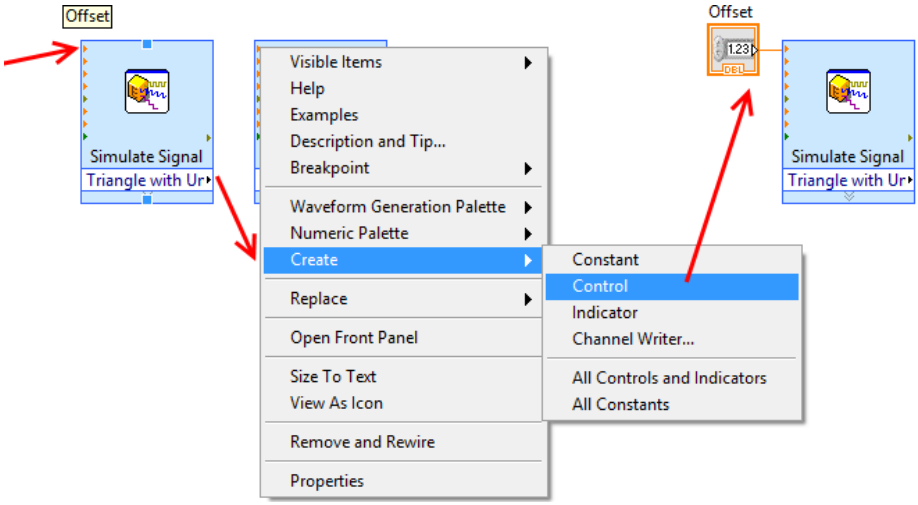


Sl. 7.2.a



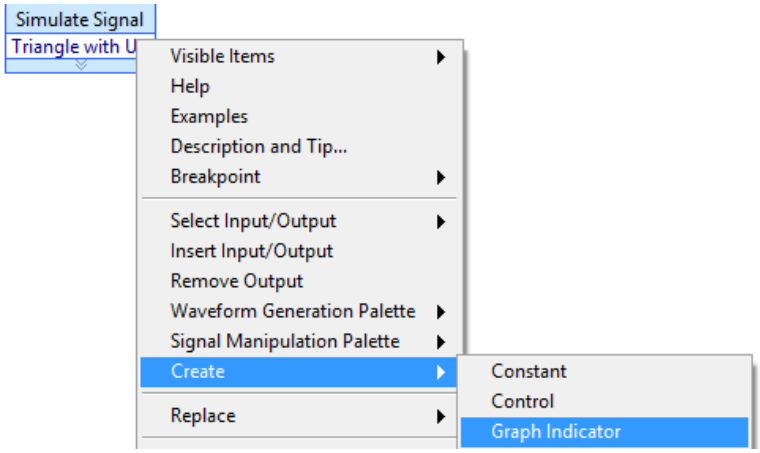
Sl. 7.2.b

2. Desnim klikom na polje *Offset* na ikoni *Express VI*-a *Simulate Signal*, zatim izabrati *Create»Control* dodati polje *Offset*, slika 7.3.



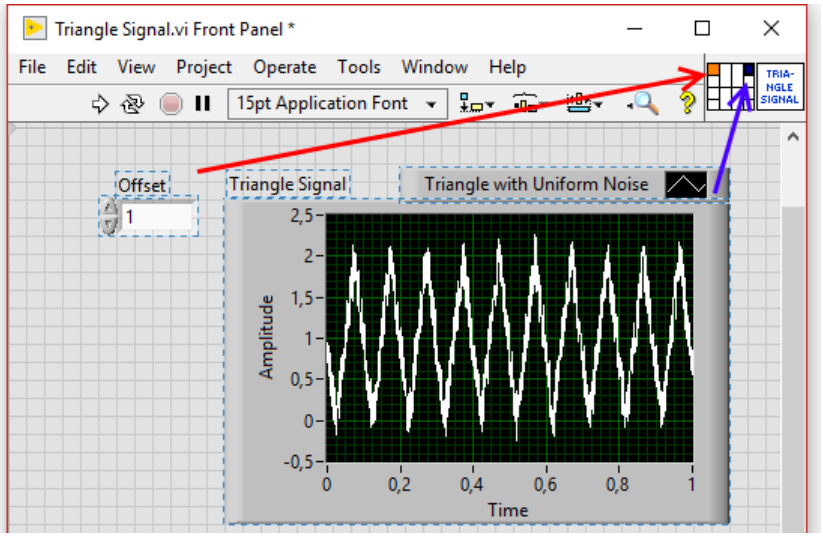
Sl. 7.3.

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



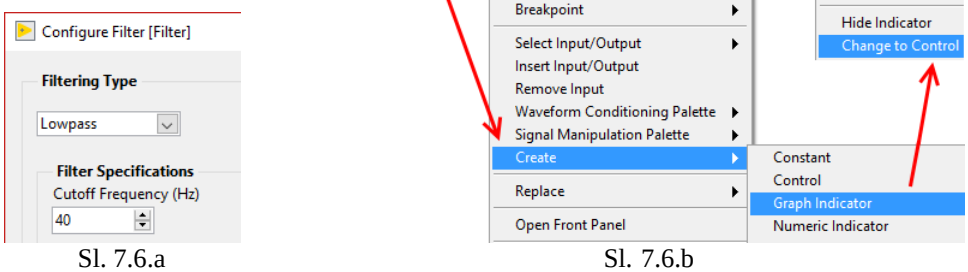
Sl. 7.4.

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



Sl. 7.5.

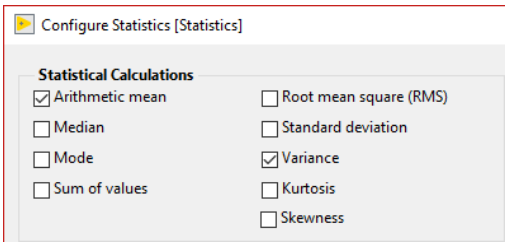
5. Slično, kao i za SubVI *Triangle Signal.vi* realizovati i ostala tri SubVI sa slike 7.1.
   a. Dodati *Express VI* za filtriranje signala *Filter*. Za tip filtra izabrati *Lowpass* i *Cutoff Frequency* postaviti na 40, slika 7.6.a. Izlaz funkcije *Filter* dodati slično kao pod tačkom 3, dok za ulaz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



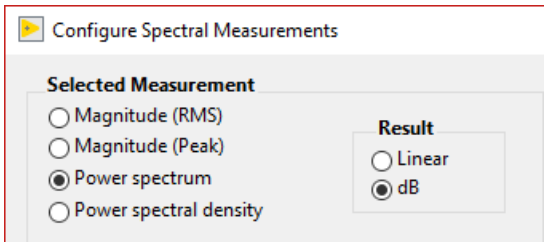
Sl. 7.6.a

Sl. 7.6.b

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

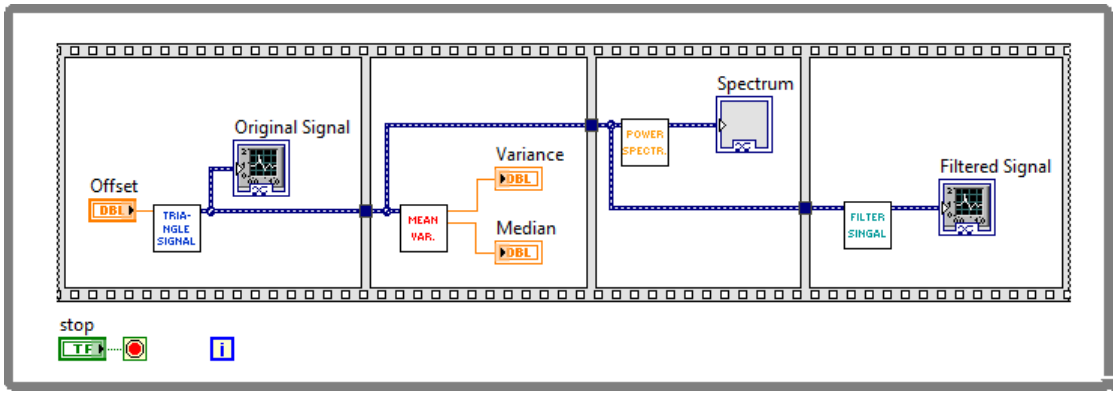


Sl. 7.7.a



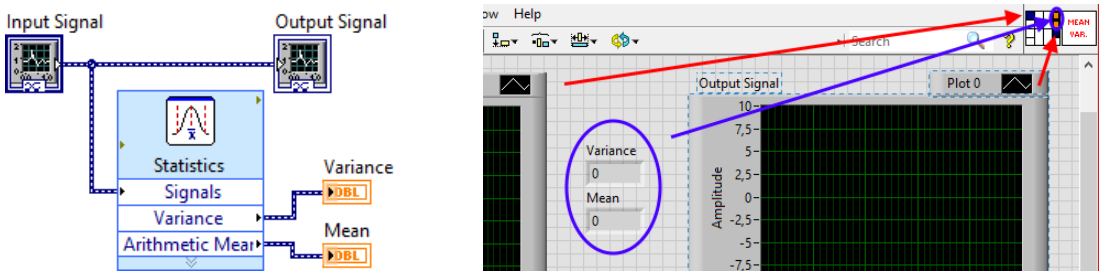
Sl. 7.7.b

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



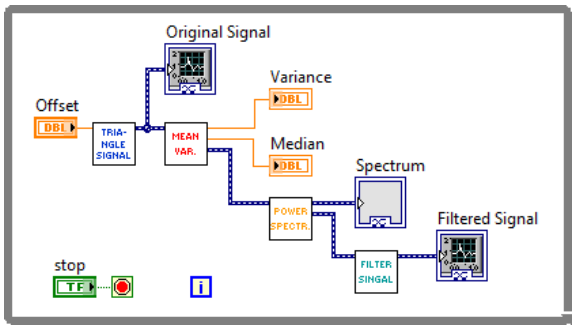
Sl. 7.8.

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



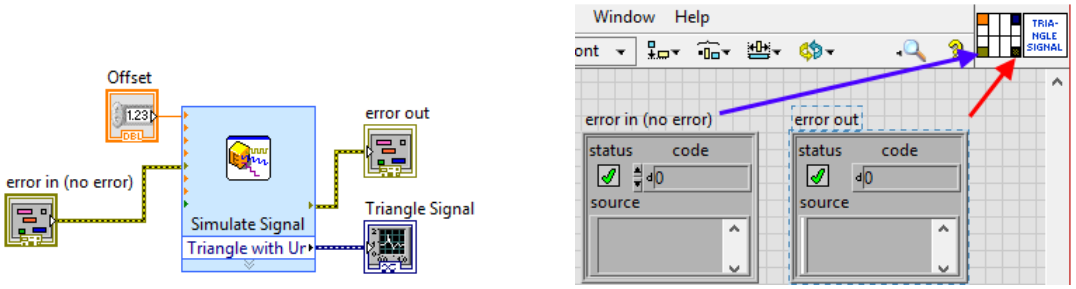
Sl. 7.9.

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



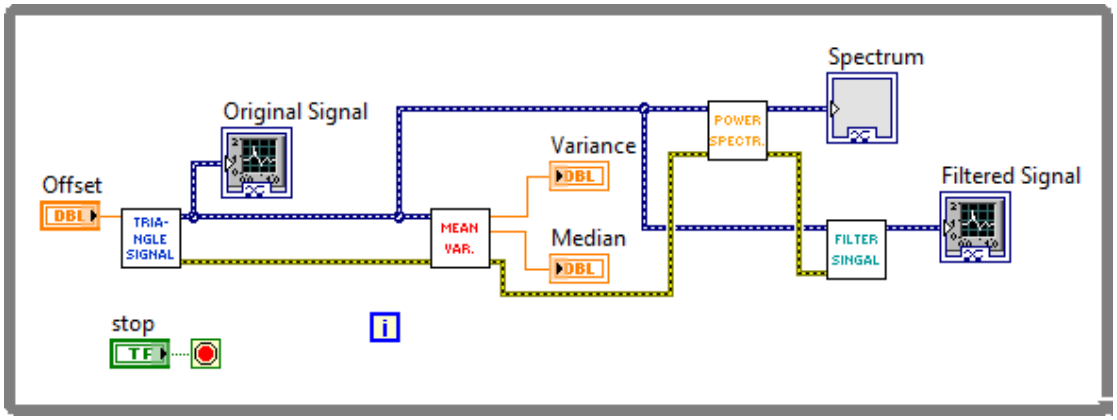
Sl. 7.10.

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 nju koristi za sekvencijalno izvršavanje programa. Ipak, razumljivi je dodati signal greške, što je potrebno uradi za svaki SubVI, slika 7.11. Napomena, većina ugrađenih VI u LabVIEW sadrži klaster greške.



Sl. 7.11.

11. Sada obezbediti sekvencijalno izvršavanje pomoću signala greške, slika 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.

Uputstvo:

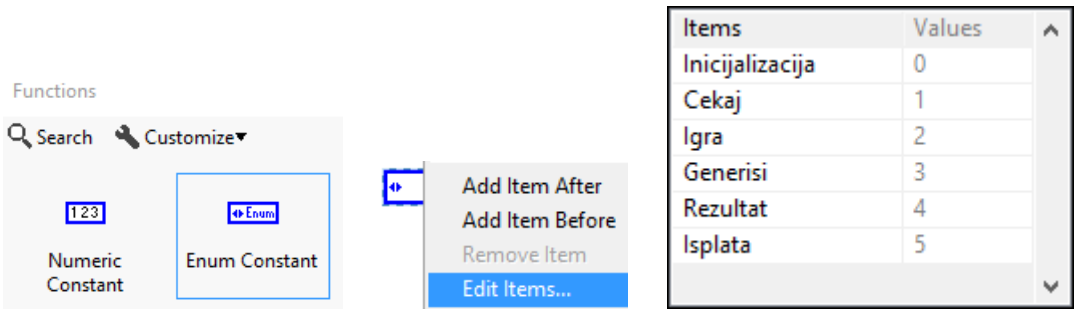
1. Pri realizaciji mašine stanja potrebno je dizajnirati dijagram toka programa, kako bi se olakšalo pisanje koda. Zato se definišu sva stanja:

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, slika 7.13. Niz od tri broja je realizovati kao tri zasebna indikatora.

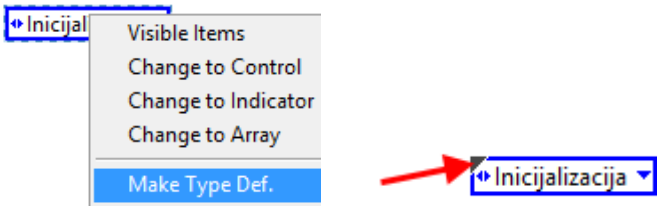
Sl. 7.13.

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



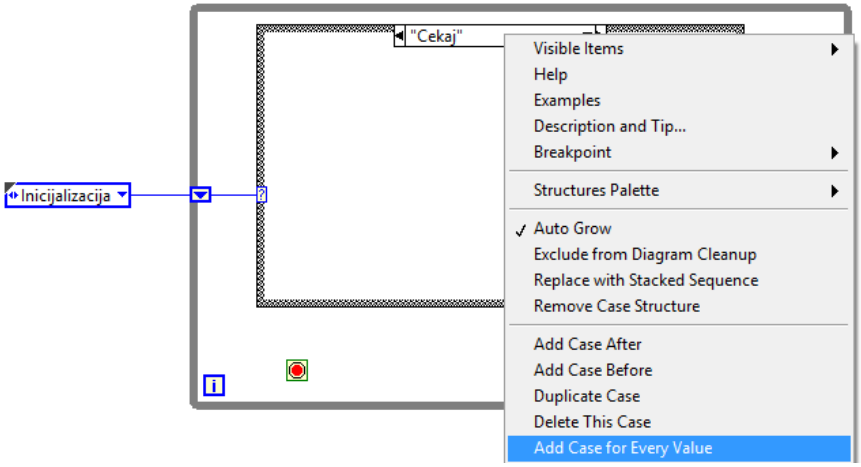
Sl. 7.14.

4. Kliknuti desnim na *enum* konstantu i izabrati *Make Type Def*. slika 7.15. Ovo omogućava da se da se svaka promena redosleda ili broja stanja menja samo na jednom mestu. Pri realizaciji koda 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*.



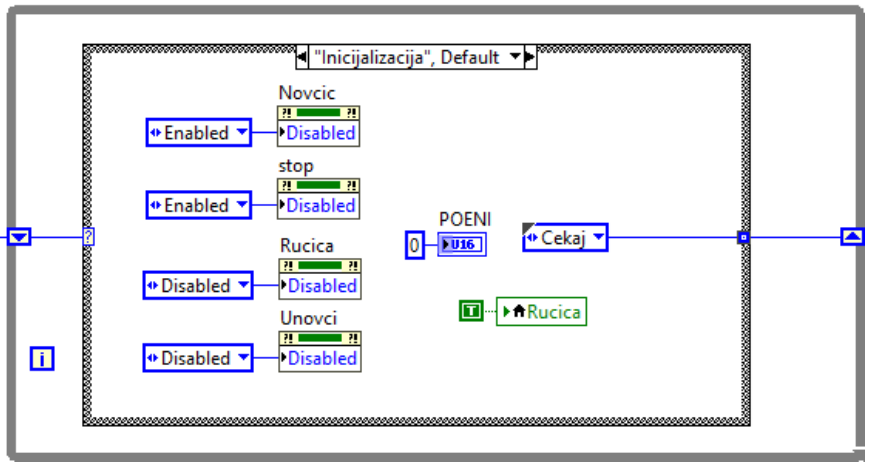
Sl. 7.15.

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



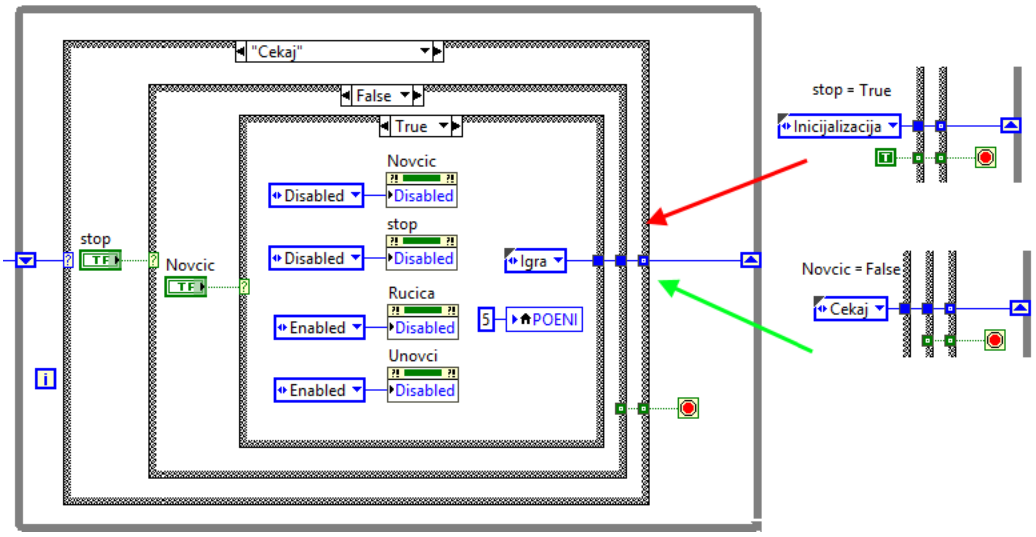
Sl. 7.16.

6. Realizovati kod za stanje *Inicijalizacija* kao na slici 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?).



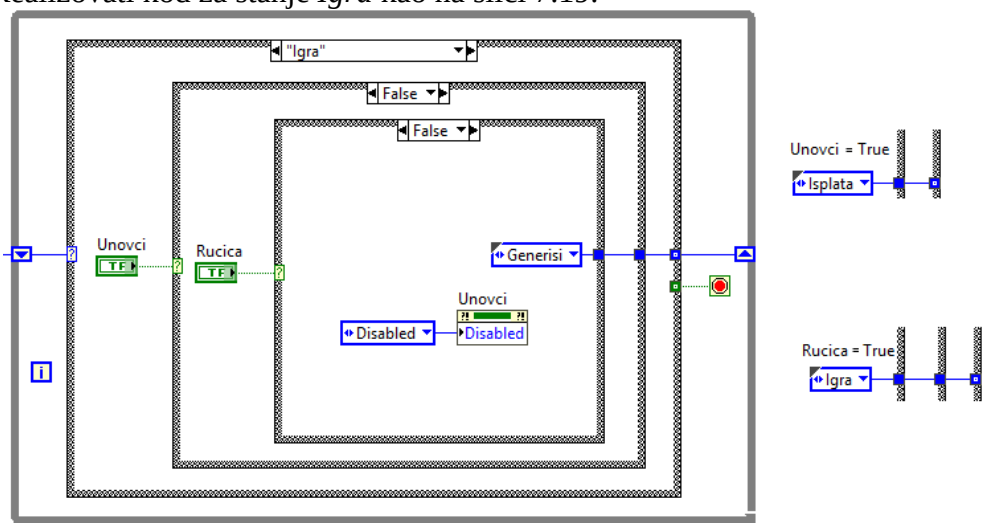
Sl. 7.17.

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



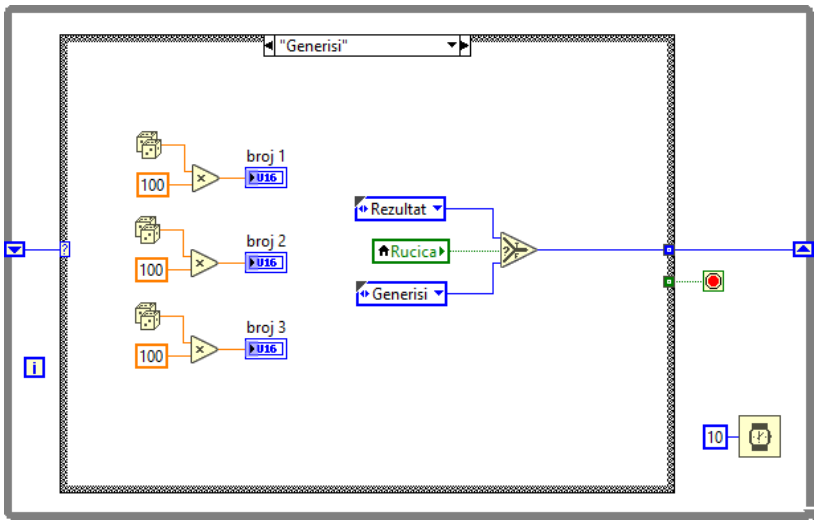
Sl. 7.18.

8. Realizovati kod za stanje *Igra* kao na slici 7.19.



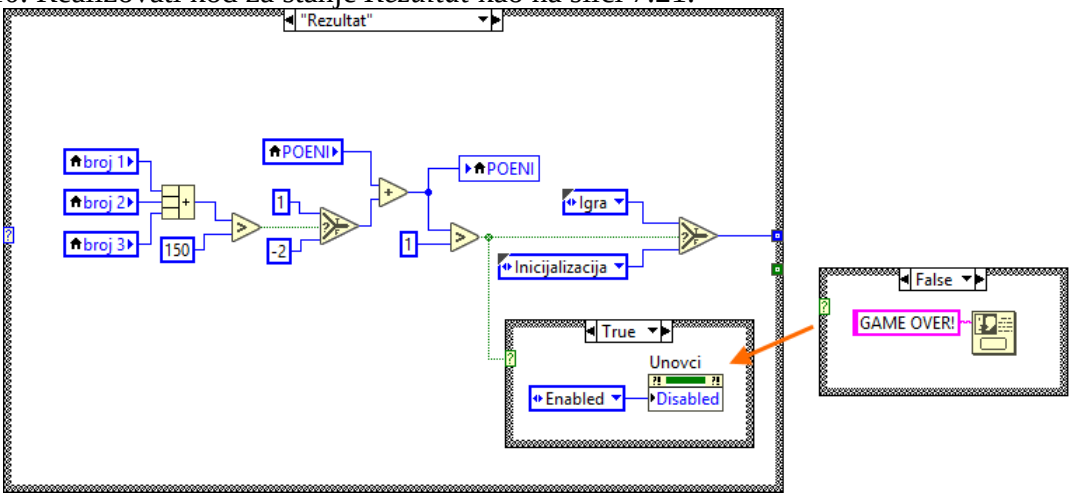
Sl. 7.19.

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



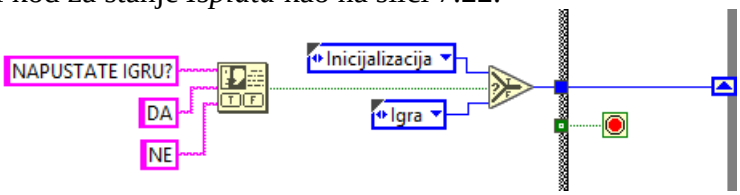
Sl. 7.20.

10. Realizovati kod za stanje *Rezultat* kao na slici 7.21.



Sl. 7.20.

11. Realizovati kod za stanje *Isplata* kao na slici 7.22.

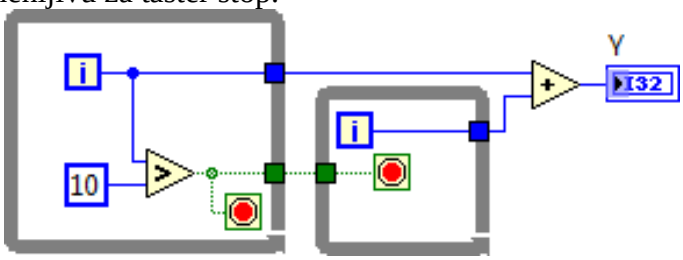


Sl. 7.21.

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 se LabVIEW javlja grešku? Kako je moguće zadržati kreiranu lokalnu promenljivu i izbeći grešku? Da li to utiče na funkcionalnost programa.

Zadatak 7.3.

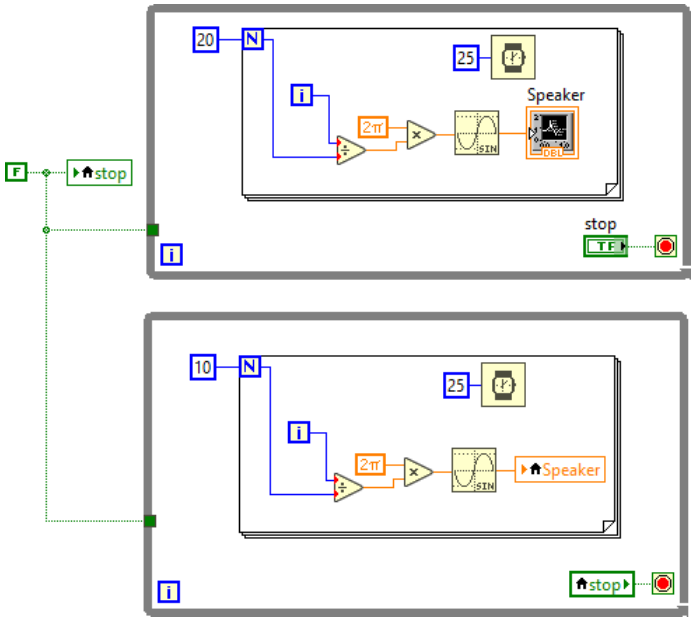
1. Realizovati kod sa slike 7.22., a zatim obezbediti da se petlje istovremeno isključuju, pri čemu se druga petlja izvršava nekoliko puta. Napomena: koristi lokalnu promenljivu za taster stop.



Sl. 7.23.

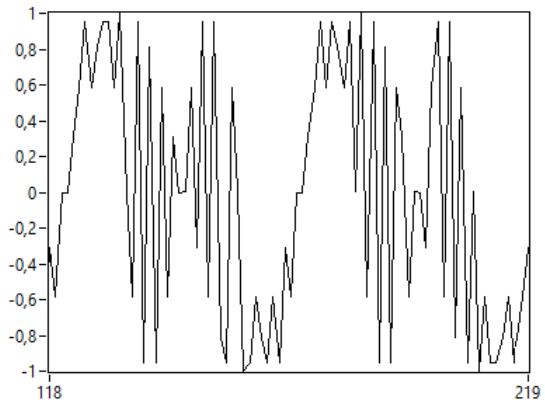
Zadatak 7.4.

1. Kreirati kod sa slike 7.24. Napomena: indikator *Speaker* je *Waveform Chart*.

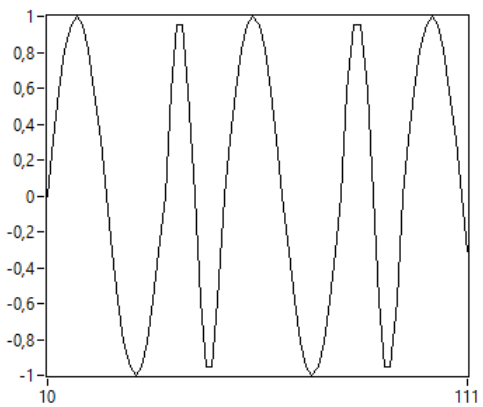


Sl. 7.24.

2. Pokrenuti dobije kod. Objasniti zašto se dobija rezultat kao na slici 7.25.a, a ne kao na slici 7.25.b. 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.



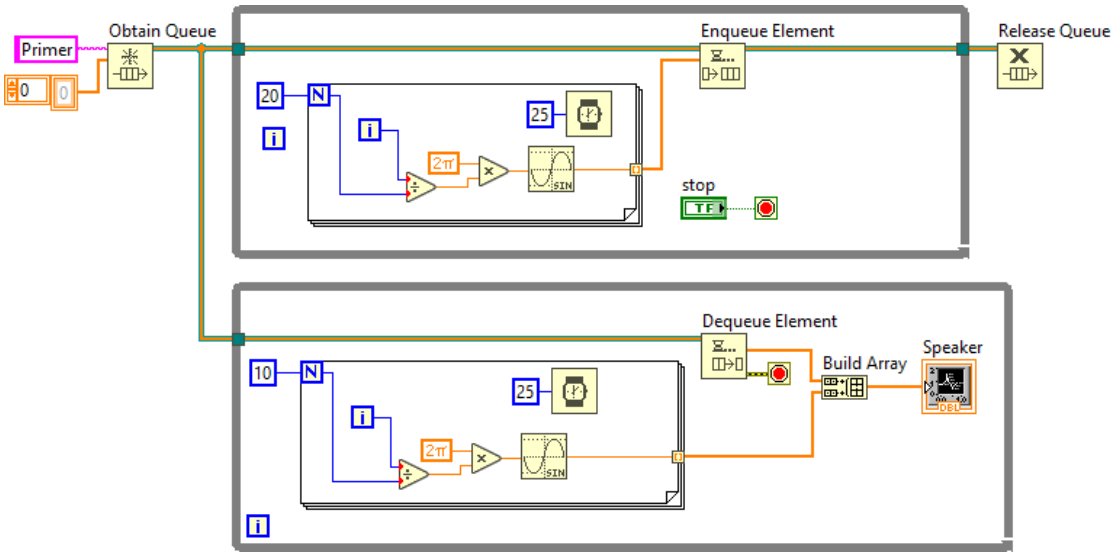
Sl. 7.25.a



Sl. 7.25.b

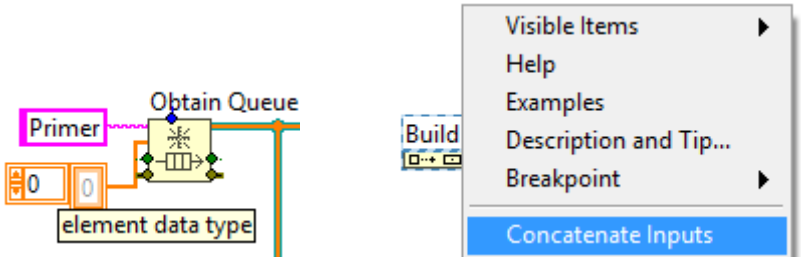
Zadatak 7.5.

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



Sl. 7.26.

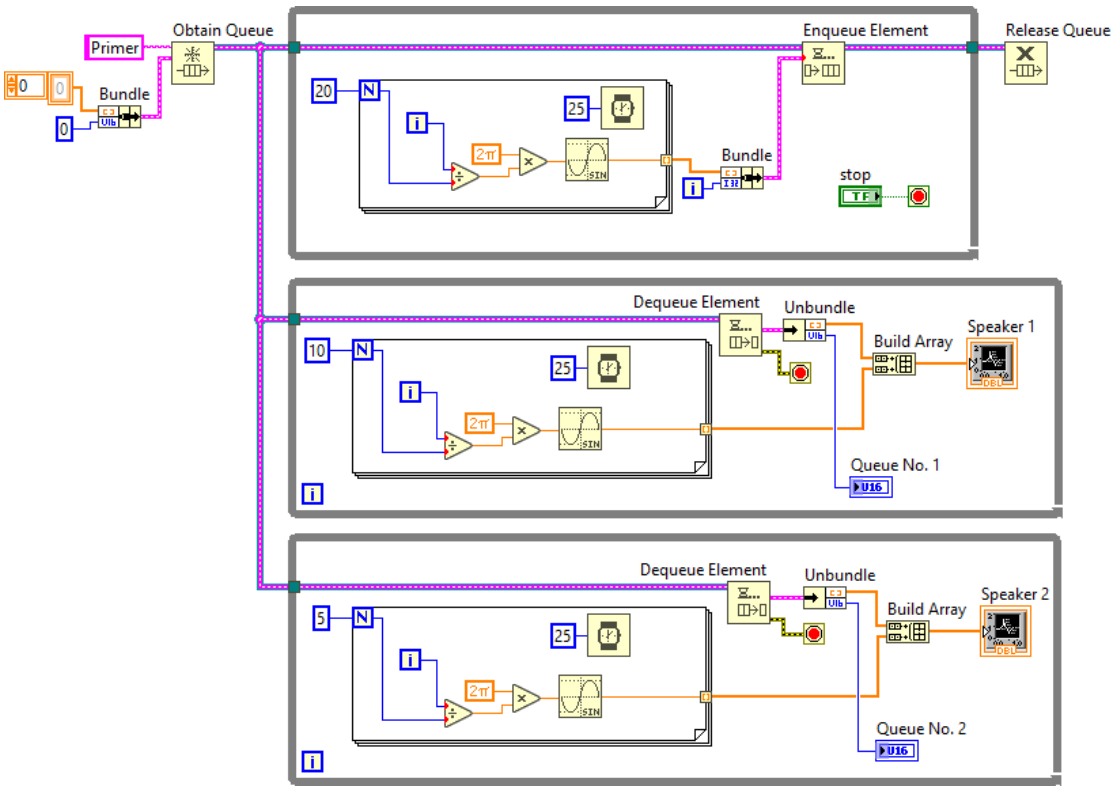
2. Prvo se indikator *Speaker* prebaci u donju *while* petlju, a zatim izlaz iz sinus funkcije dovede na izlazi tunel *for* petlje kako bi se dobio niz. To važi za obe *for* petlje. Obrisati sve lokalne promenljive za taster *stop*, a njega prebaci u stanje *Latch*. Na ulaz *element data type* (slika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



Sl. 7.28.

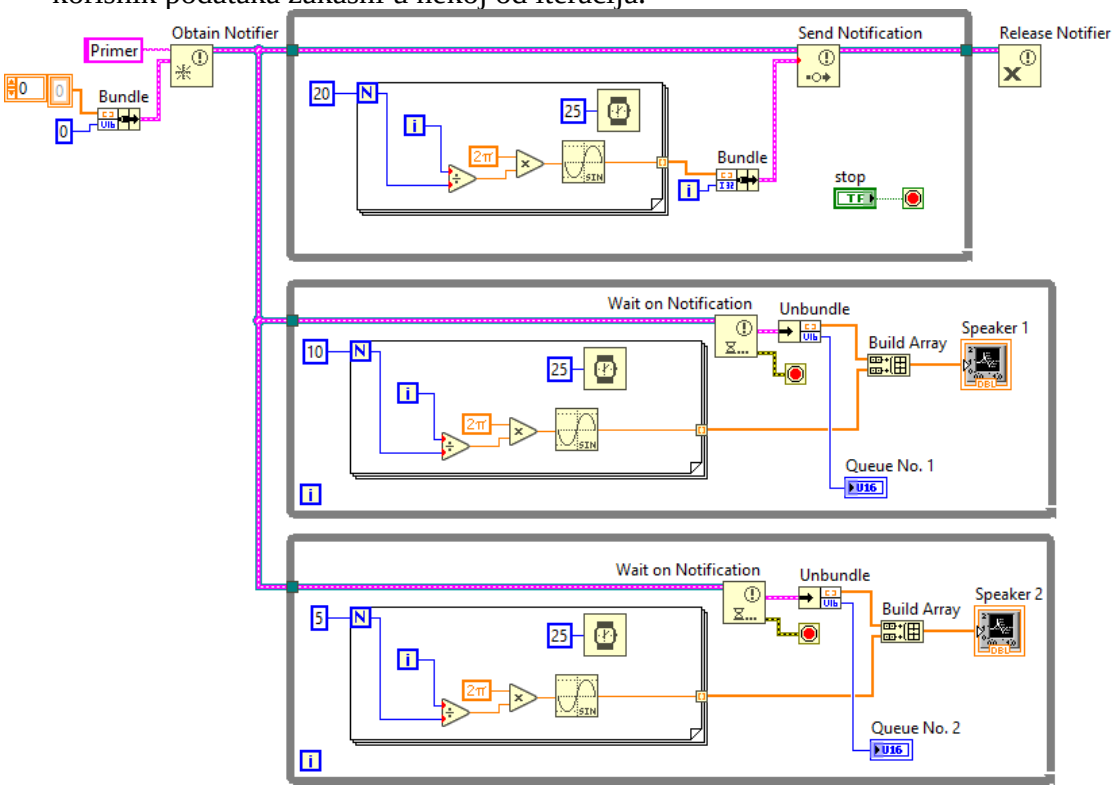
Zadatak 7.6.

1. Promeniti prethodno zadatak kako bi se dobio kod sa slike 7.28. Sada je element reda klaster koji sadrži niz i ceo broj.
2. Pokrenuti realizovani kod 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 jedno iteraciji element reda se pročita u srednjoj *while* petji, a u sledećoj se pročita u donjoj *while* petlji. Odnosno 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 kod kao na slici 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.