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 2

Korišćenje struktura

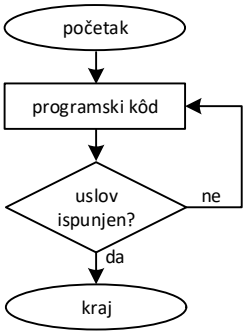
Cilj

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.

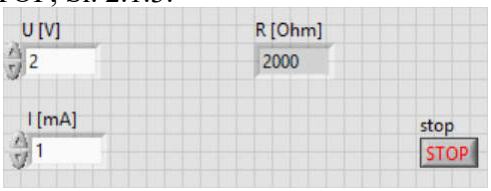
Uputstvo:

1. Pokrenuti ***NI LabVIEW*** softver izborom opcije: ***Start»National Instruments»LabVIEW 2018 (32-bit)*** ili klikom na ***NI LabVIEW*** prečicu na *Desktop*-u.
2. Otvoriti 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 pritiskom 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*).

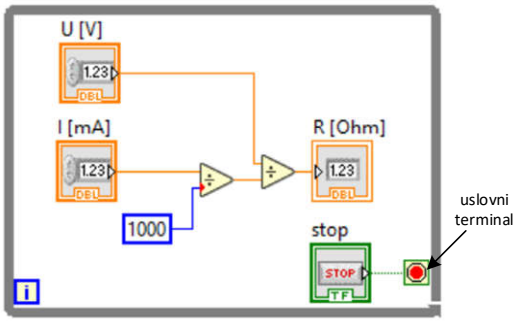
Sl. 2.1.2. *While* petlja

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 odgovarajuće natpise za labele 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.



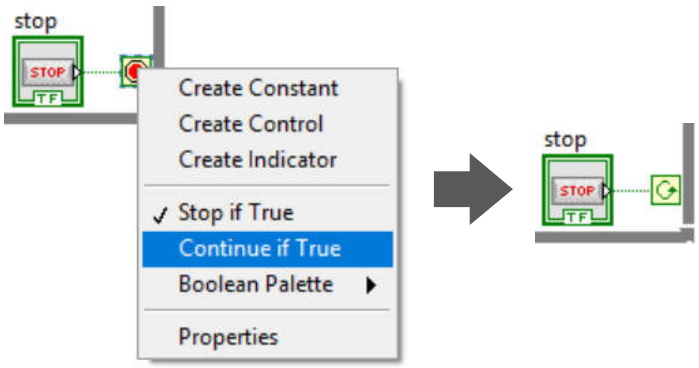
Sl. 2.1.4. *Block diagram* programa za izračunavanje vrednosti otpornosti

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***.

SAVET: 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 *0201_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.

Uputstvo:

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

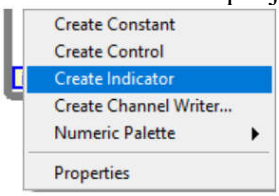
Sl. 2.2.1. *Flat Sequence* struktura

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.

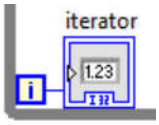
Sl. 2.2.2. *Flat Sequence* struktura sa tri sekvence

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

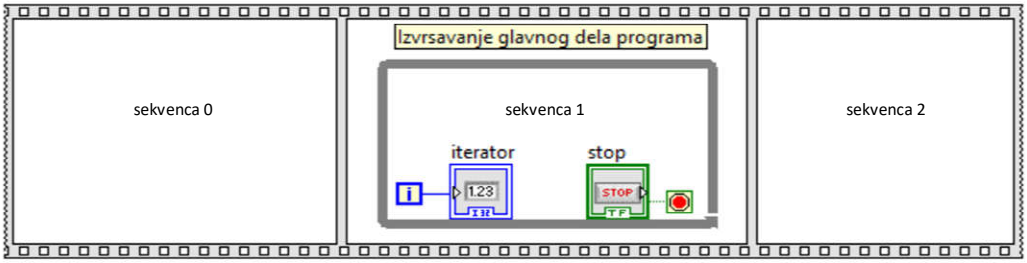
meniju izabere opcija ***Create Indicator*** kao na Sl. 2.2.3. Uočiti da je pri pravljenju indikatora „prečicom“ format podataka automatski podešen da bude I32 (*signed integer 32 bit*-a), tj. ne mora se podešavati u opciji *Properties* indikatora (podsetnik iz Lekcije 1: opcija *Properties* indikatora se otvara desnim klikom miša na terminal indikatora u *Block Diagram*-u). Labelu 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“

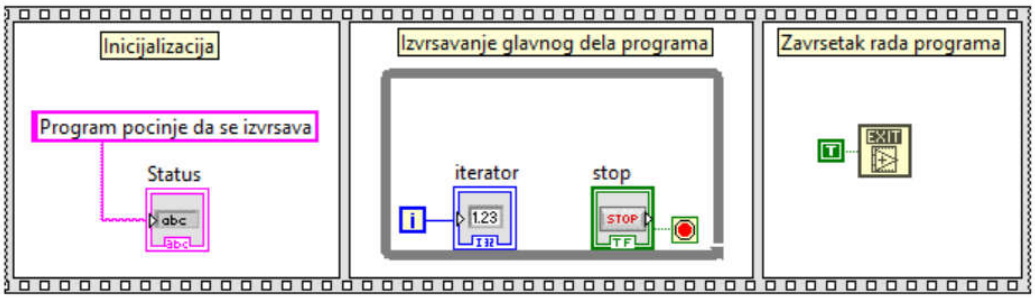


Sl. 2.2.4. Kreiranje numeričkog indikatora za prikaz vrednosti iteratora *While* petlje



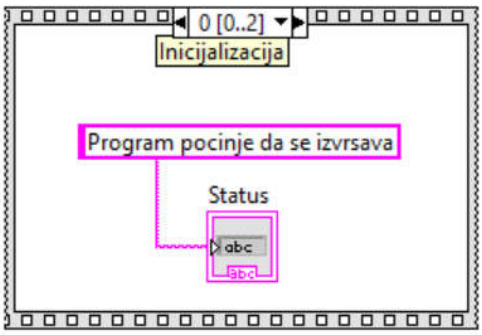
Sl. 2.2.5. Centralna sekvenca sa *While* petljom

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 labelom „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 *0202_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 *0202_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 leve/desne strelice (ili izborom iz padajućeg menija) može selektovati prikaz sekvence 0, 1 ili 2.



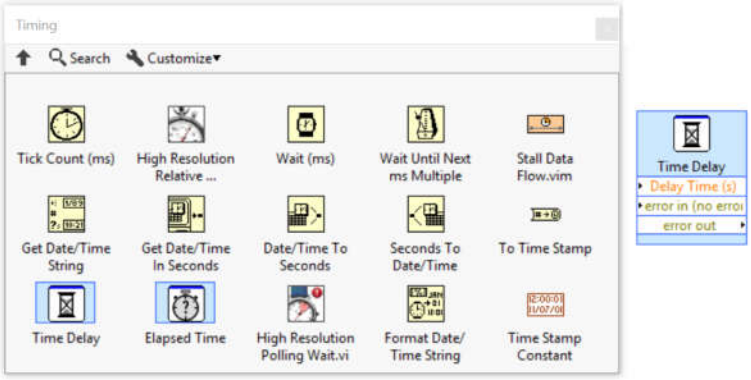
Sl. 2.2.7. *Block Diagram* programa sa *Stacked Sequence* strukturom

12. Zatvoriti progam bez čuvanja izmene iz tačke 11.

SAVET: 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

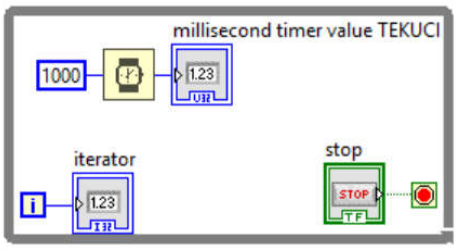
U primerima iz prethodna dva poglavlja, iteracije *While* petlje se izvršavaju jedna za drugom, bez pauze izmedju, a vreme izvršavanja jedne iteracije *While* petlje je reda veličine milisekunde. U prethodnim primerima, procesorsko vreme je zauzeto neprekidnim ponavljanjem iteracija. Izvršavanje iteracija *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.

Uputstvo:
1. Otvoriti 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

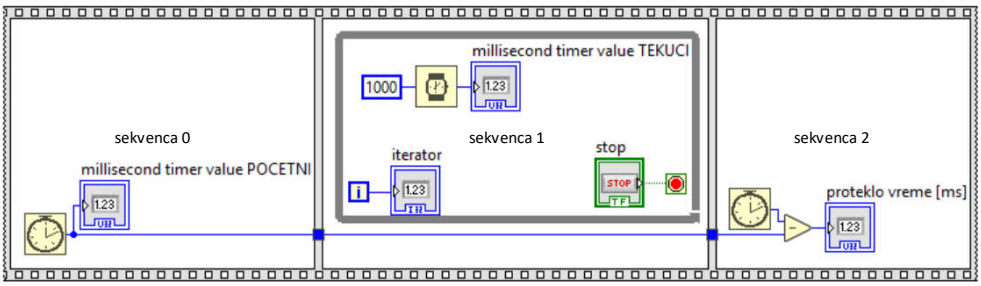
3. Pokrenuti izvršavanje programa. Uočiti da se vrednost iteratora menja svake sekunde.

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 paljenju 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),* u tom slučaju će vreme izvršavanja iteracije *While* petlje biti diktirano trajanjem programskog kôda.

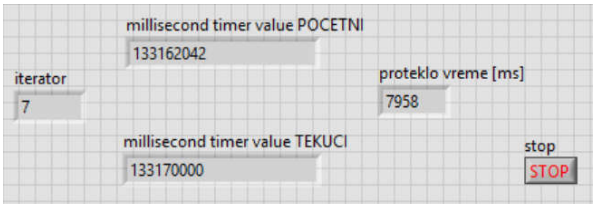
4. Oko *While* petlje razvući *Flat Sequence* strukturu, a potom kreirati jednu sekvencu pre i jednu sekvencu posle sekvence u kojoj je *While* petlja (prema instrukcijama u Poglavlju 2.2.), Sl. 2.3.3.



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 sekvenci 0 i 2.
7. Sačuvati program kao *0203_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 sekvenci 0 i 2.



Sl. 2.3.4.

10. Sačuvati program kao *0203_Wait_Until.vi*.
11. Zatvoriti progam.

SAVET: *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 odnosu na 1.01.1904. 12:00 a.m.

Zadatak 2.3.2.

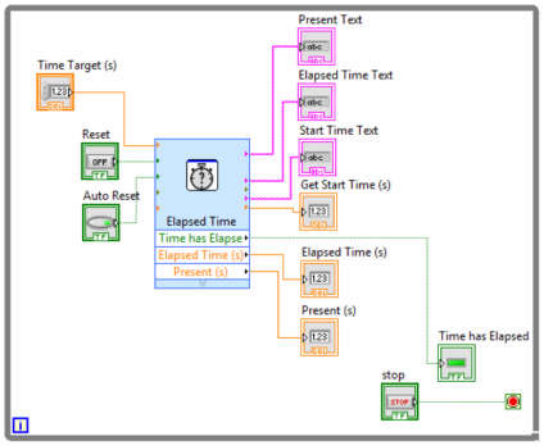
Koristeći *Express Elapsed Time* funkciju kreirati program koji meri proteklo vreme i:

- kome se može zadati vreme koje treba izmeriti
- koji može resetovati merenje vremena pritiskom na dugme na interfejsu
- koji može automatski resetovati merenje vremena kada prođe zadato vreme

Modifikovati program tako da završi njegovo izvršavanje kada istekne zadato vreme.

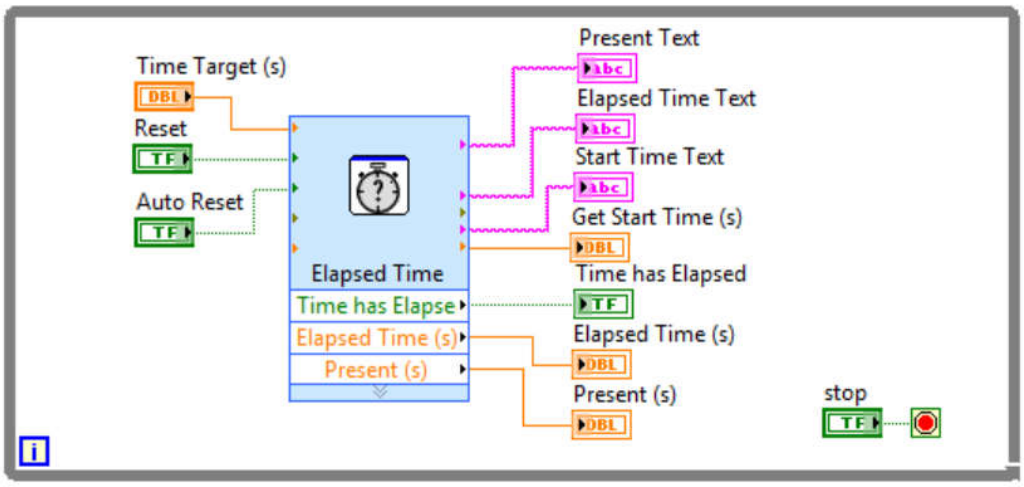
Uputstvo:

1. Otvoriti 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 desni klik miša na odgovarajući ulaz/izlaz, a potom se selektuje opcija Create Control/Indicator).



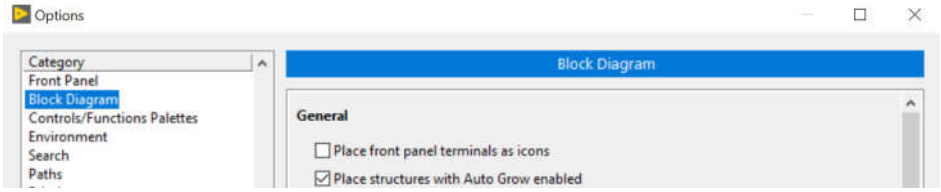
Sl. 2.3.5.

3. Sačuvati program kao *0203_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 odčekira 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.



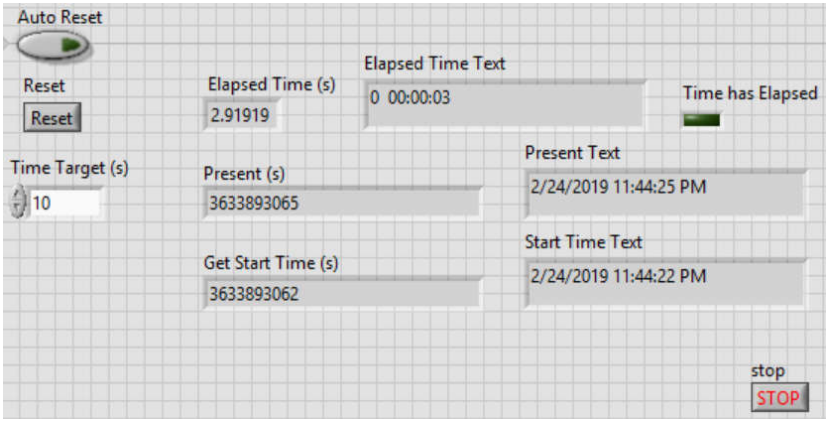
Sl. 2.3.6.

SAVET: prikaz terminala u obliku ikona može izazvati nepreglednost u *Block Diagram*-u. Zato je preporučljivo u meni liniji u Tools/Options/Block Diagram/ odčekirati opciju *Place front panel terminals as icons*, Sl. 2.3.7.



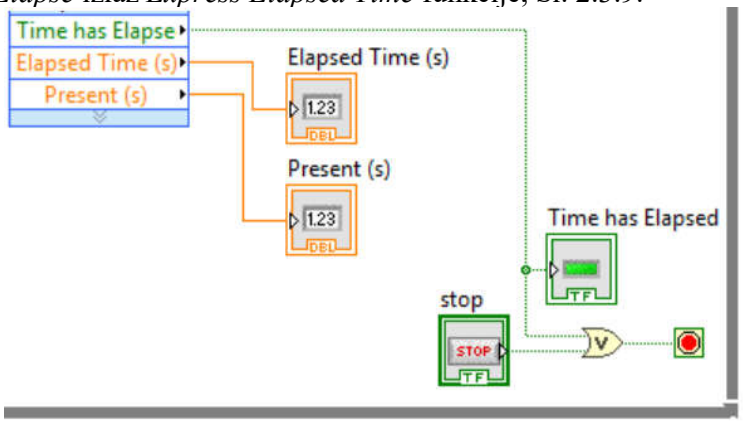
Sl. 2.3.7. Podešavanje u Tools/Options/Block Diagram/ da se terminali ne prikazuju kao ikone

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 (desni klik miša na indikatoru, 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.

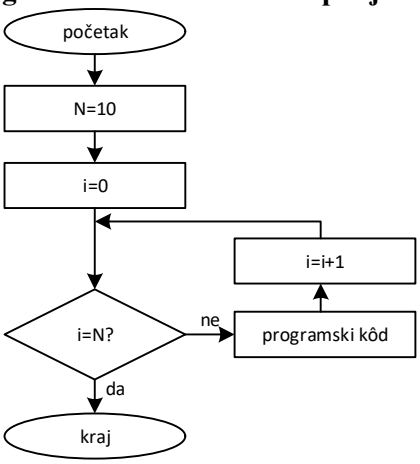


Sl. 2.3.9.

8. Pokrenuti izvršavanje modifikovanog programa.
9. Sačuvati program kao *0203_Merenje_proteklog_vremena_zaustavi_program.vi*.

2.4 For petlja

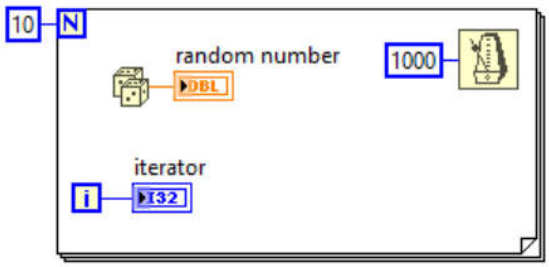
Tok izvršavanja programa sa *For* petljom (*For Loop*) je prikazan na Sl. 2.4.1 Programski kôd unutar *For* petlje počinje da se izvršava ako je „uslov“ izvršavanja ispunjen. U slučaju kada u prvoj iteraciji *For* petlje „uslov“ nije ispunjen, programski kôd se ne izvršava, tj. **programski kôd unutar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

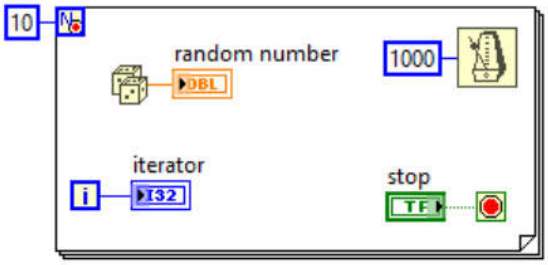
Uputstvo:
1. Otvoriti nov **.vi** program.
2. Izabrati *For Loop* u paleti ***Functions»Programming»Structures***, a potom u *Block Diagram*-u pritiskom levog tastera miša razvući *For* petlju kao na Sl. 2.4.2. Uočiti na Sl. 2.4.2. iterator i (*Iteration Terminal*) i brojač N (*Count Terminal*) *For* 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 N (*Count Terminal*) *For* petlje je tipa I32 (*signed integer 32-bit*-a). U slučaju da se drugačiji tip podataka poveže na brojač, taj tip podataka 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 čekirati 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.



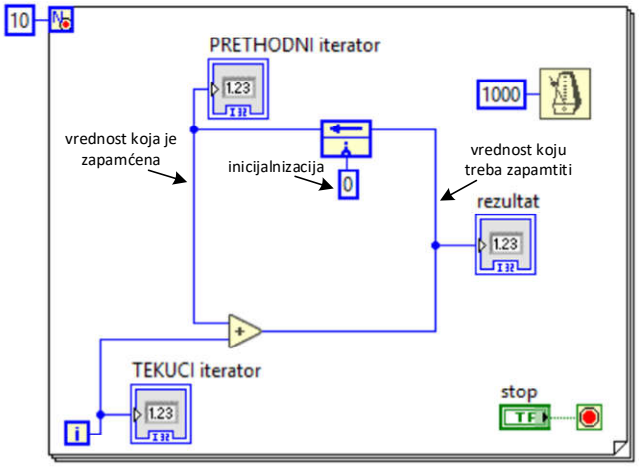
Sl. 2.4.3. *For* petlja sa *Conditional Terminal*-om

5. Pokrenuti izvršavanje programa. Pritiskom na dugme STOP se može zaustaviti izvršavanje *For* petlje, i pre završetka prikaza slučajnih brojeva.
6. Sačuvati program kao *0204_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.

Uputstvo:
1. Otvoriti program *0204_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 da se zapamti vrednost promenljive u prethodnoj iteraciji petlje i da se iskoristi za sabiranje u narednoj iteraciji.
3. U *Block Diagram*-u kreirati kôd kao na Sl. 2.5.1.



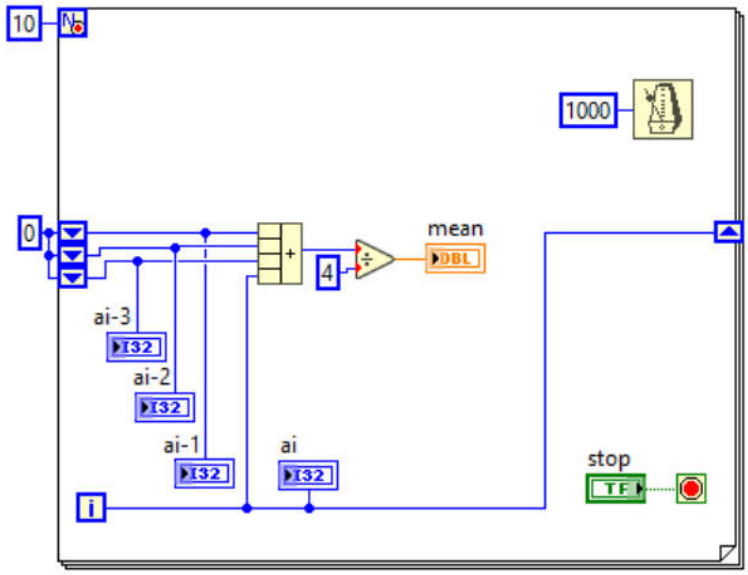
Sl. 2.5.1. *For* petlja sa *Feedback Node*-om

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 *0205_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.
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.

Uputstvo:
1. Otvoriti program *0204_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.

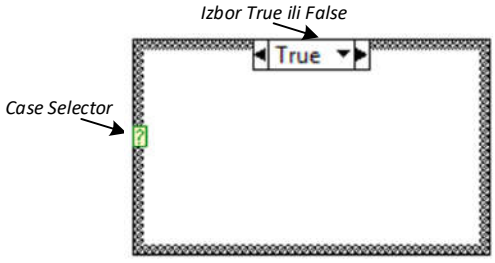
3. Pokrenuti izvršavanje programa. U početnoj iteraciji *For* petlje (i=0), *Shift registar* je inicijalizovan vrednošću 0. U sledećoj iteraciji je u njemu zapamćena vrednost poslednjeg iteratora, i tako redom za sve tri pozicije levog *Shift* registra Sl. 2.5.2.
4. Sačuvati program sa *File*/*Save As...* kao *0205_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

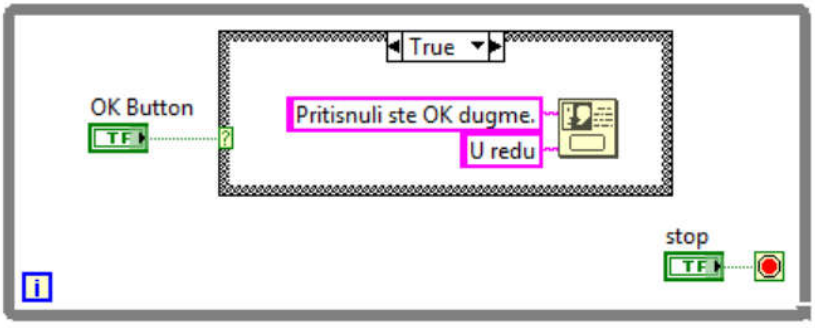
Zadatak 2.6.1.
Kreirati program koji nakon pritiska na dugme OK prikazuje prozor sa rečenicom „Pritisnuli ste OK dugme.“ Prozor se gasi pritiskom na dugme „U redu“ koje se nalazi u prozoru. Program se završava pritiskom na dugme STOP.

Uputstvo:
1. Otvoriti 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* gde se povezuje 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 nizovne 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 *0206_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 mišem 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.

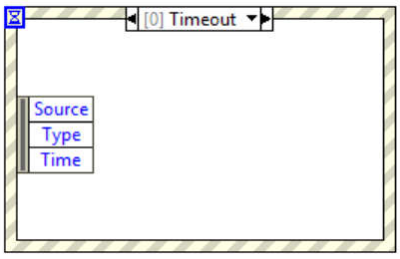
Zadatak 2.6.2.
U ***Help»Find Examples*** pronaći i otvoriti primer *Mechanical Action.vi*. Proučiti razlike u mehaničkim akcijama logičkih promenljivih.

2.7 Event struktura

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

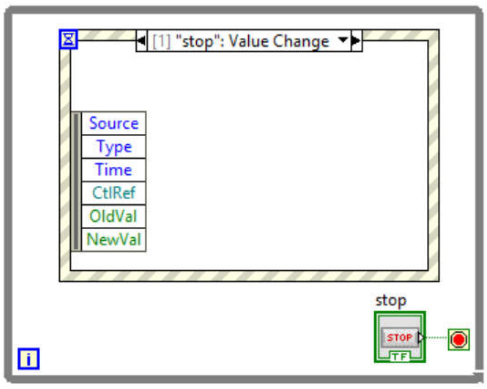
- zaustavlja se pritiskom na dugme STOP
- kada se prvi put klikne na dugme OK Button (mehanička akcija podešena na *Latch When Released*) se upali lampica na korisničkom interfejsu
- ako nikakva akcija nije preduzeta 5 sekundi prikazuje prozor sa natpisom „Niste nista uradili 5 sekundi“. Prozor se gasi pritiskom na dugme „U redu“ koje se nalazi u prozoru.
- 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.

Uputstvo:
1. Otvoriti 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.



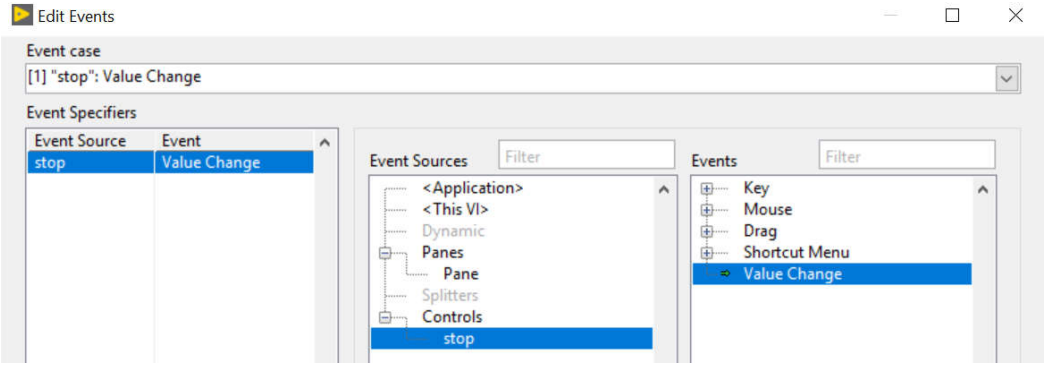
Sl. 2.7.1. *Event* struktura

3. Oko *Event* strukture razvući *While* petlju koja se završava pritiskom na dugme STOP, Sl. 2.7.2.

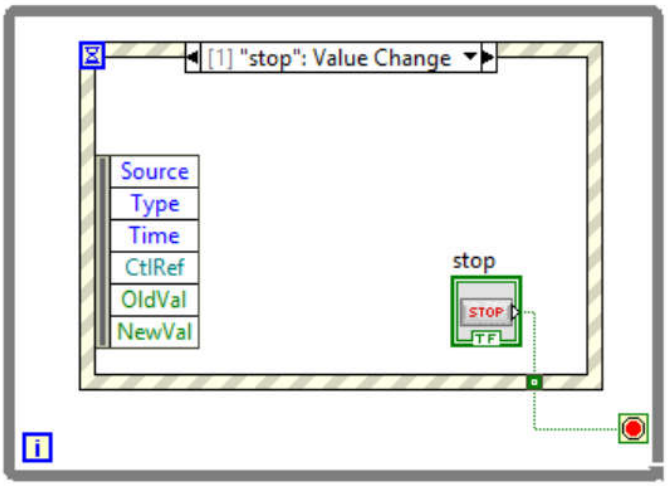


Sl. 2.7.2. Pogrešan način zaustavljanja *While* petlje koja ima *Event* strukturu

4. Uključiti dugme ***Execution Highlighting***  i pokrenuti izvršavanje programa. Zašto se tek nakon drugog pritiska na dugme STOP program zaustavlja?
5. Zaustaviti izvršavanje programa. Modifikovati program dodavanjem događaja *„stop“: Value Change* u *Event* strukturu. Dodavanje događaja se vrši tako što se klikne desnim tasterom miša na *Selector Label* u gornjem delu *Event* strukture i izabere se opcija *Add Event Case...* Potom se u *Event case* prozoru selektuje „*stop*“ među kontrolama i „*Value Change*“ među *event*-ima, Sl. 2.7.3. Obratiti pažnju na to šta sve od događaja može biti iskorišćeno za akciju na korisničkom interfejsu. 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. 2.7.4. Povezati STOP dugme sa uslovnim terminalom While petlje na način koji je prikazan na Sl. 2.7.4.

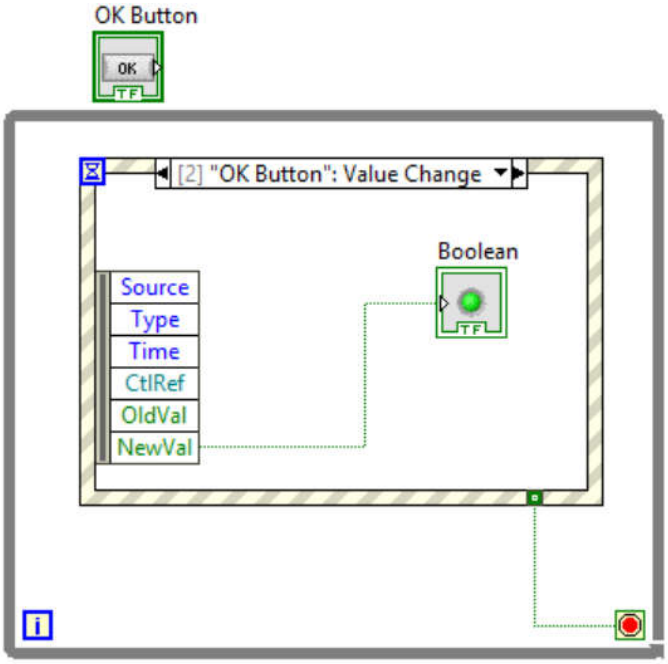


Sl. 2.7.3. Dodavanje događaja *„stop“: Value Change* u *Event* strukturu



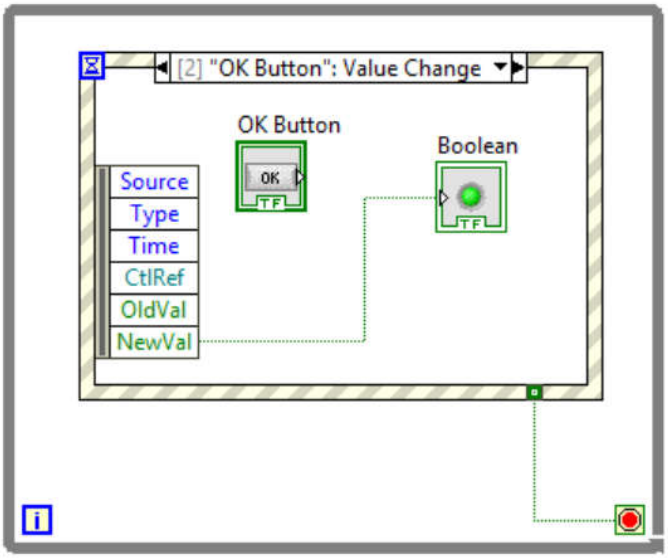
Sl. 2.7.4. Ispravan način zaustavljanja *While* petlje koja ima *Event* strukturu

6. Pokrenuti izvršavanje programa. Zaustaviti izvršavanje programa pritiskom na dugme STOP. Uočiti razliku u odnosu na tačku 4.
7. Dodati u program 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 5. 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 nakon pritiska *Latch OK Button* dugmeta se lampica upali, ali da se *OK Button* dugme ne vrati u *False* položaj.
9. Zaustaviti izvršavanje programa pritiskom na dugme STOP.



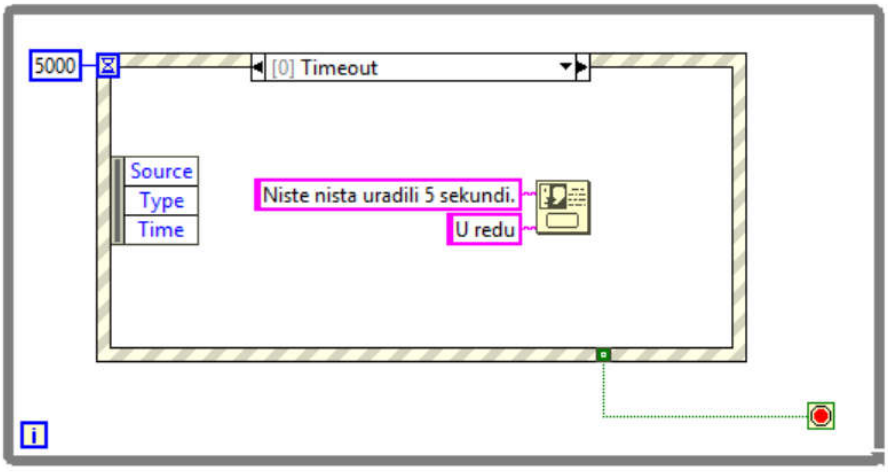
Sl. 2.7.5. Neispravan način upotrebe *Latch* dugmeta u *Event* strukturi

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 ćošak *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.
    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 *0207_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.