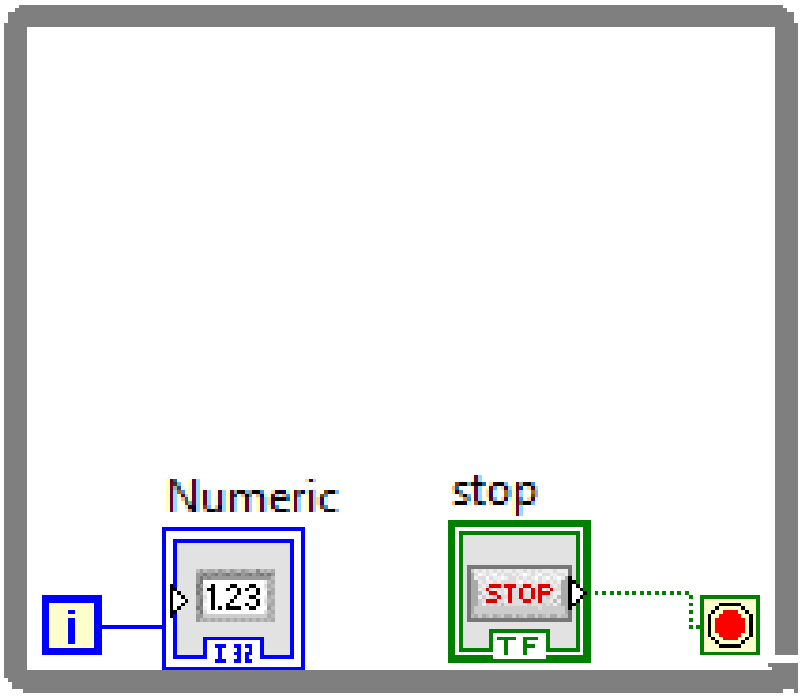
Producer – Consumer automati stanja

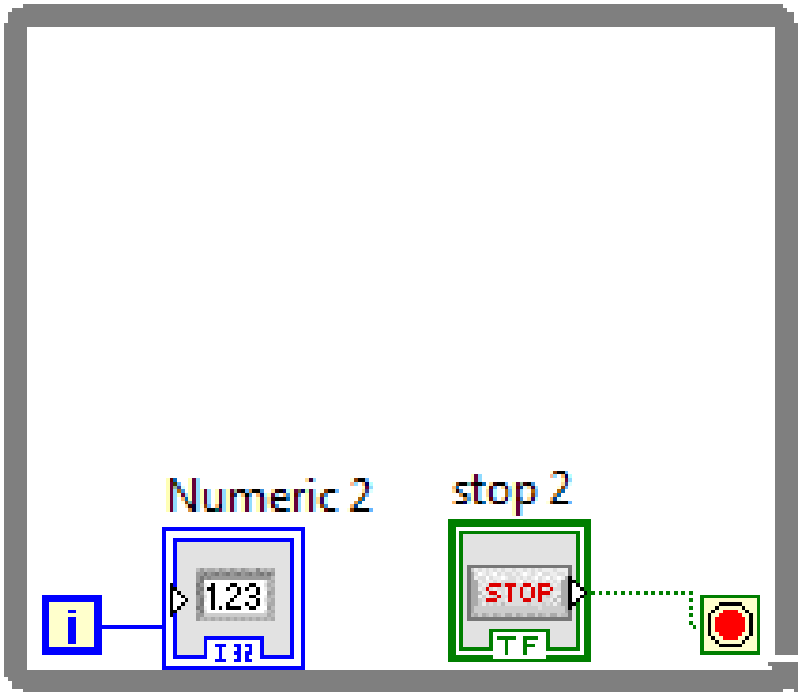
Uvod

- Do sada:
  - Klasični (case strukture)
  - Event-driven
    - Enum konstante
    - Redovi klastera i stringova
    - Argumenti
- Zajedničko – jedna petlja!
- Izvesna ograničenja (npr. reaguje samo u Wait stanju)
- Potrebno stalno praćenje korisničkog interfejsa u svim stanjima
- Rešenje – arhitektura sa paralelnim petljama!
- Producer – Consumer:
  - Podela uloga između petlji
  - Akvizicija podataka i obrada - Producer–Consumer (Data) arhitektura
  - Praćenje stanja i njihovo izvršavanje - Producer–Consumer (Events) arhitektura
    - Producer nadzire korisnički interfejs i prati događaje pomoću Event strukture
    - Consumer izvršava procedure za zadata stanja

Komunikacija

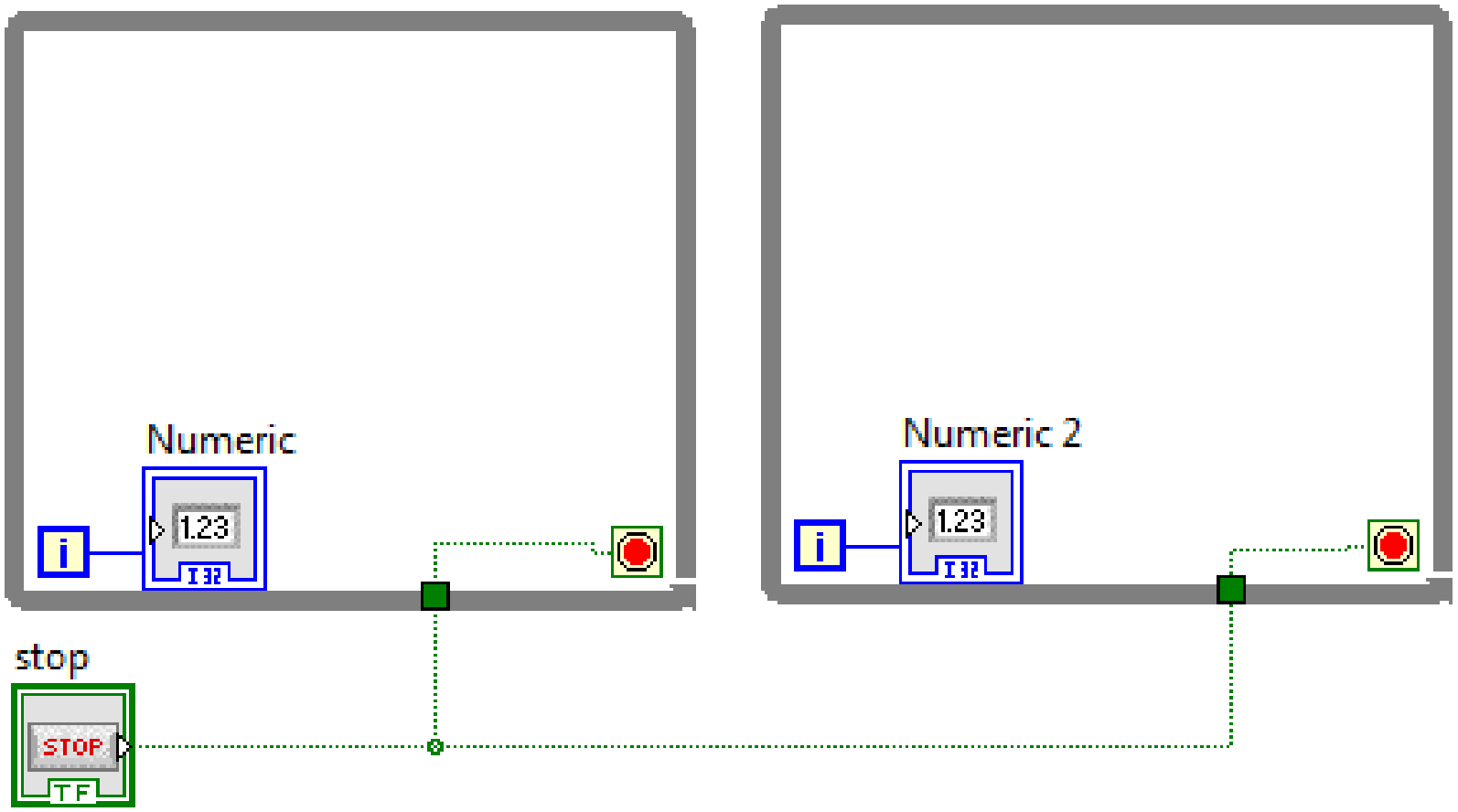
Zaustavljanje paralelnih petlji





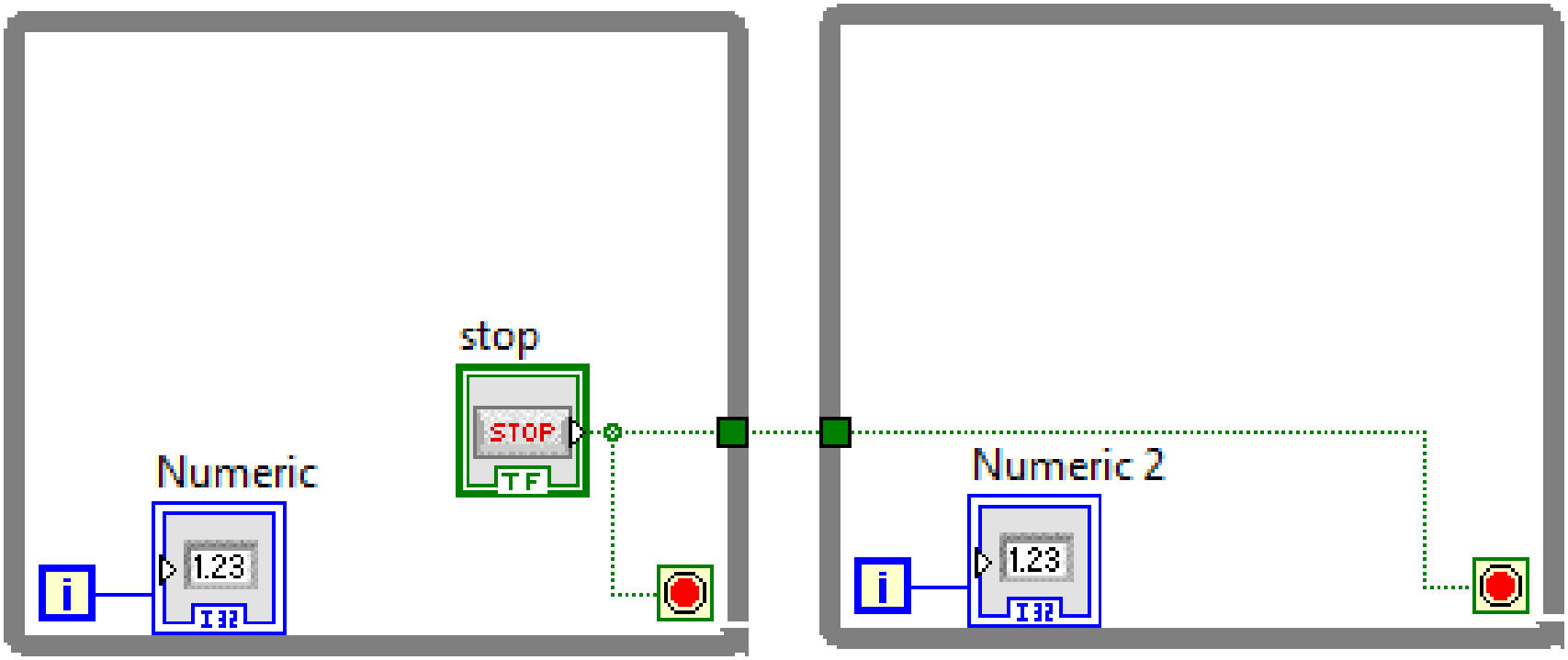
Komunikacija

Zaustavljanje paralelnih petlji



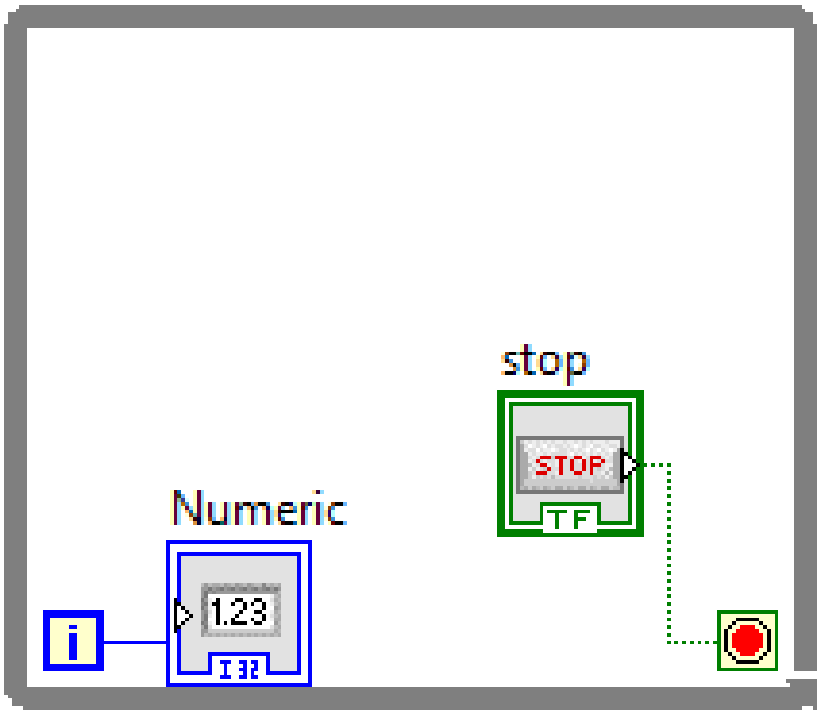
Komunikacija

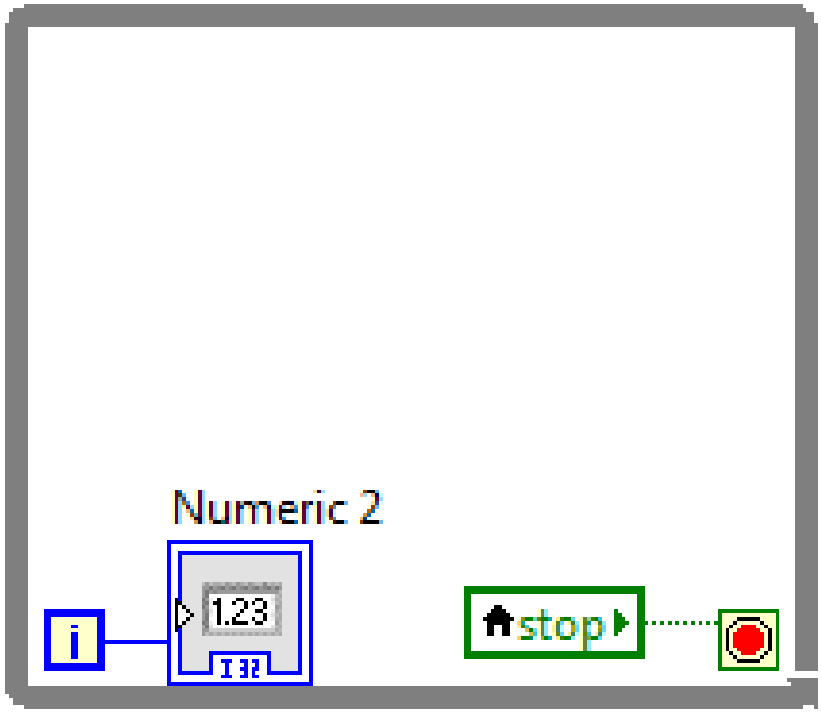
Zaustavljanje paralelnih petlji



Komunikacija

Zaustavljanje paralelnih petlji





Arhitektura

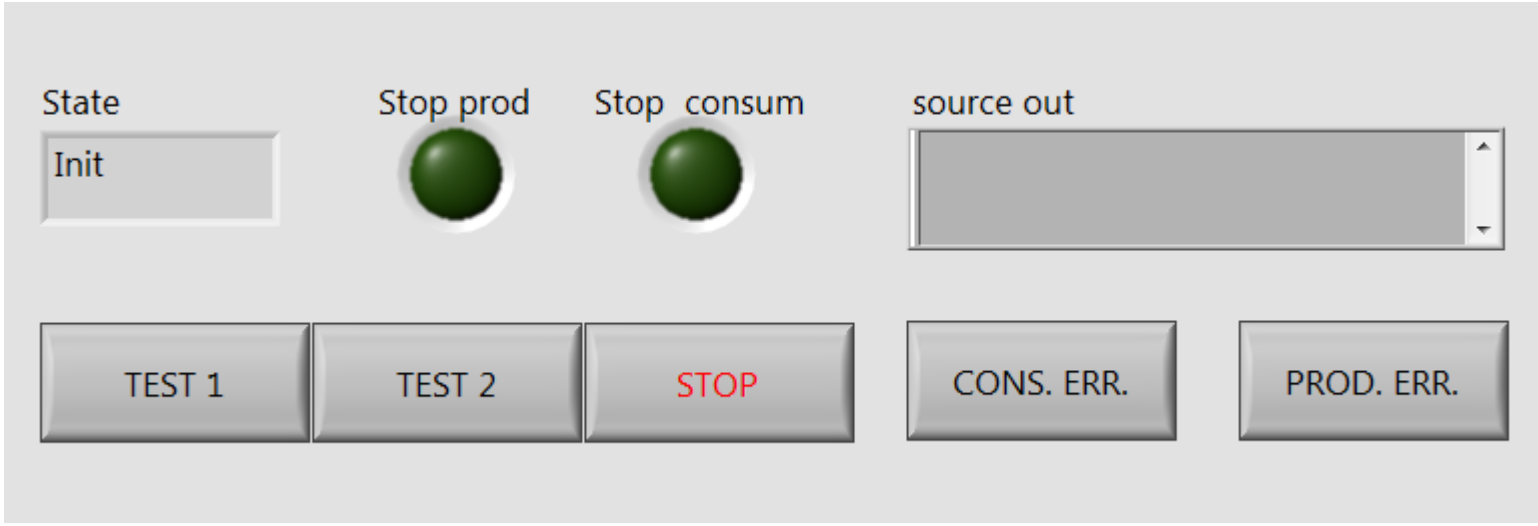
- Paralelne petlje koje komuniciraju preko redova
- Producer petlja kreira podatke i šalje ih preko redova do Consumer petlje; uglavnom jednosmerno
- U obrnutom smeru – kada se isključuje Producer petlja
- Podaci se “baferuju”, pa nema gubitaka
- Primer – akvizicija podataka
  - Producer prikuplja podatke jednom brzinom
  - Consumer obrađuje podatke, obično mnogo sporije
  - Komunikacija preko redova čuva sve podatke
  - Producer-Consumer (Data) arhitektura

Producer-Consumer (Events) arhitektura

- Češća upotreba kod automata stanja
- Producer petlja sadrži Event strukturu, nadzire prednji panel i promene na njemu i reaguje na događaje
- Consumer petlja sadrži automat stanja, najčešće u vidu Case structure
- Consumer procesira stanja koja dobija od Producera preko redova
- Interfejs je aktivan i dok se stanja izvršavaju

Producer-Consumer (Events) arhitektura

- Na primeru testova
- Dodati tasteri za simulaciju grešaka
- Važno je urediti upravljanje greškama, jer greška u bilo kojoj petlji mora zaustaviti ceo automat

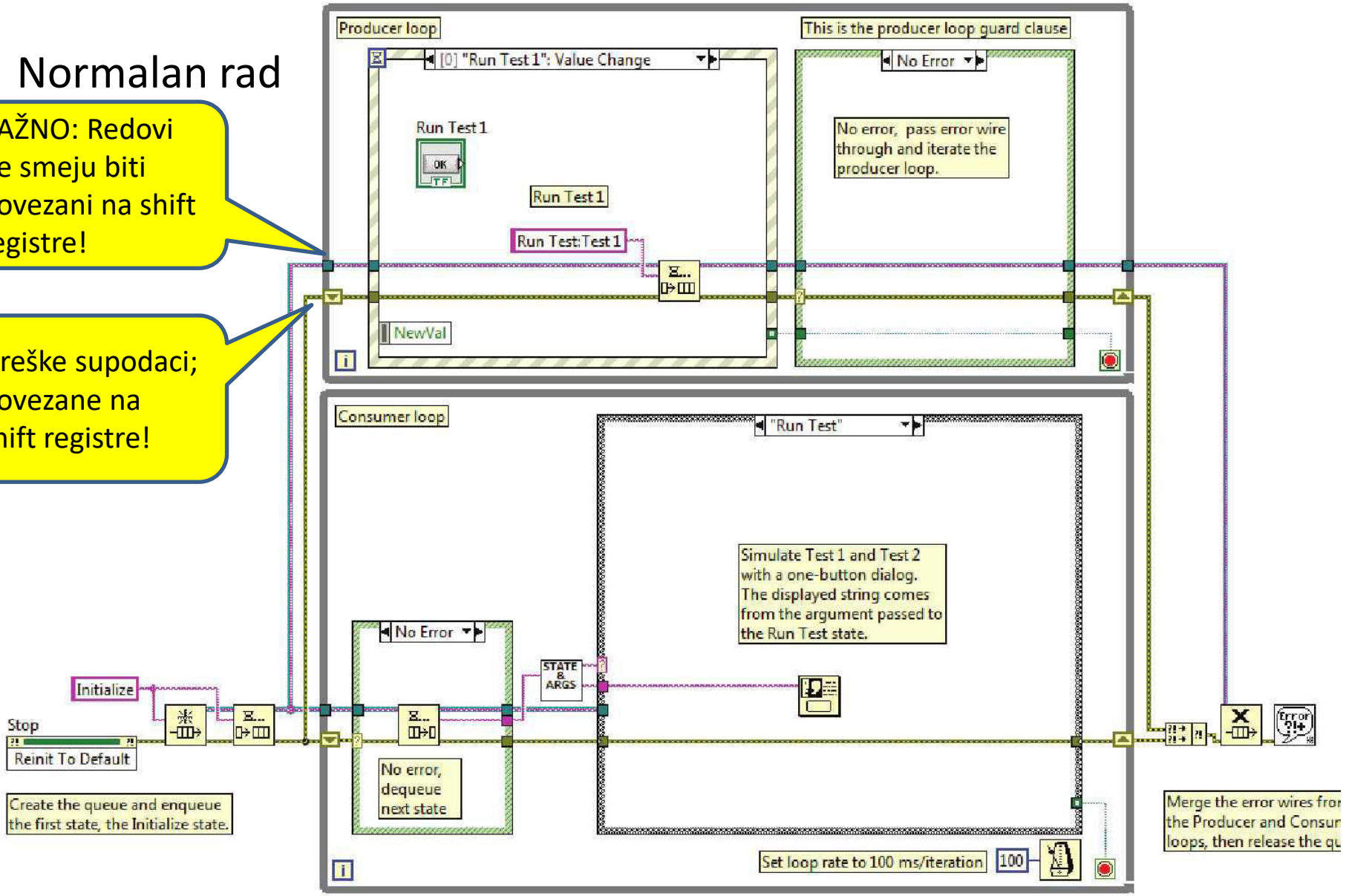


Producer-Consumer (Events) arhitektura

Normalan rad

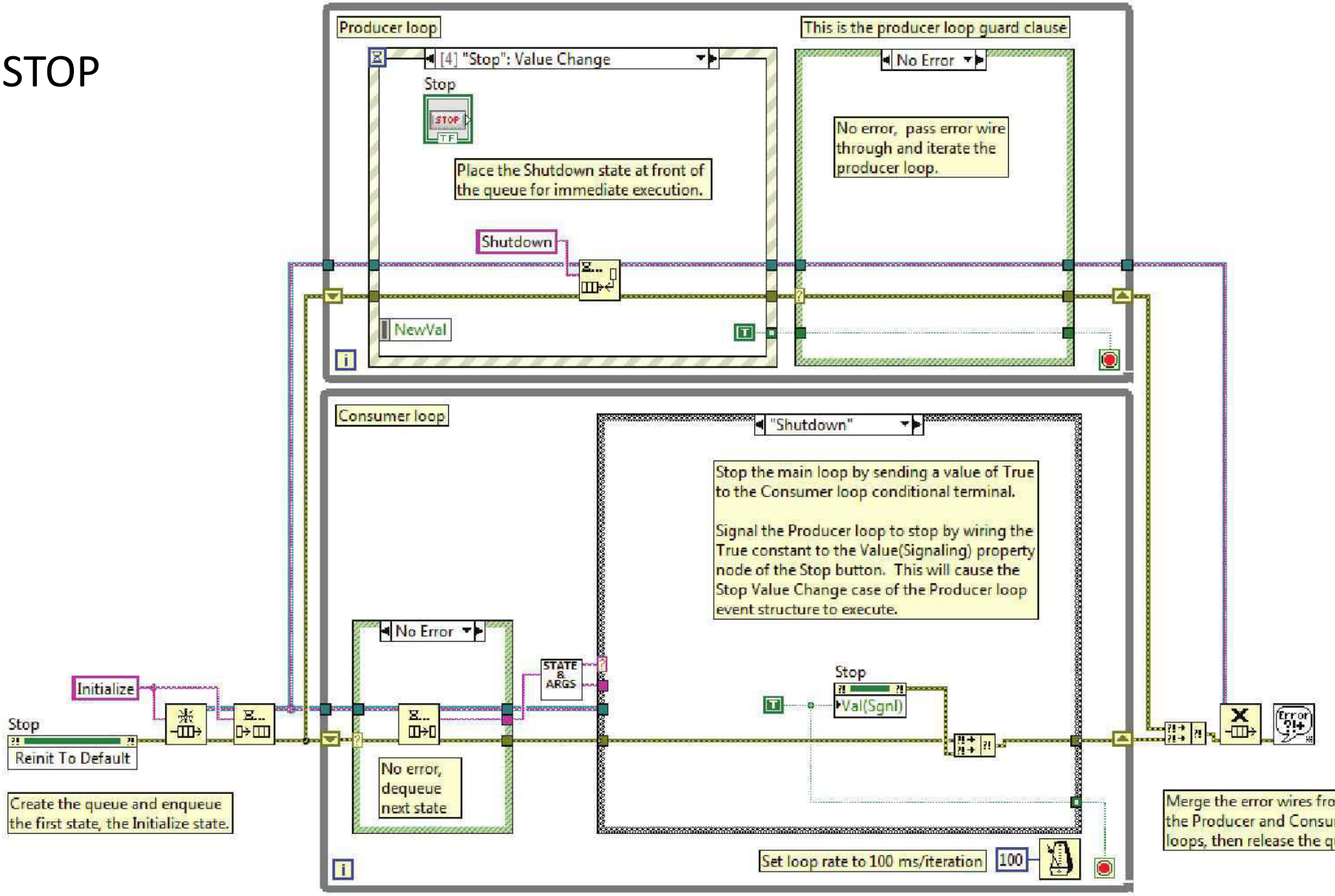
VAŽNO: Redovi ne smeju biti povezani na shift registre!

Greške supodaci; povezane na shift registre!



Producer-Consumer (Events) arhitektura

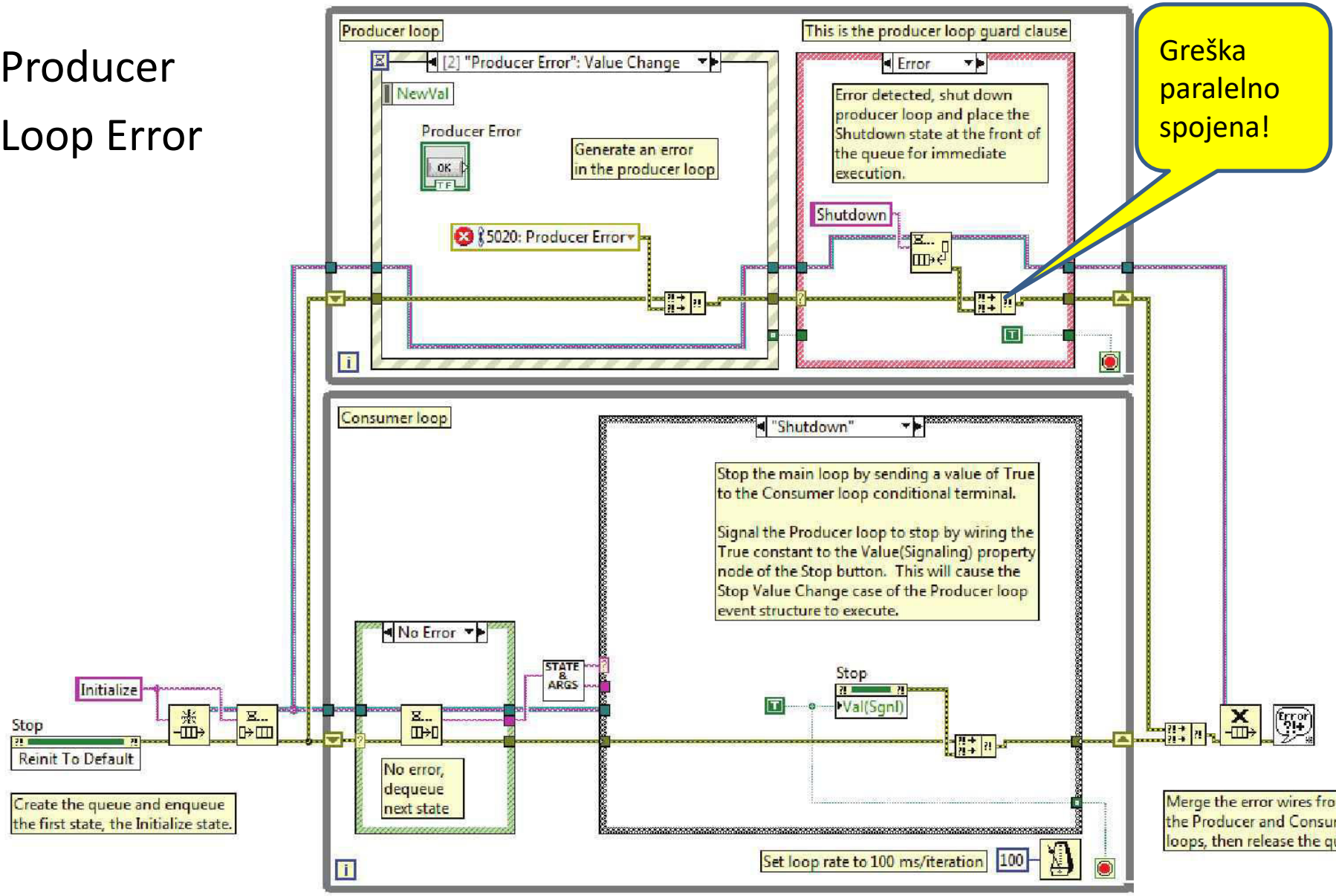
STOP



Producer-Consumer (Events) arhitektura

Producer

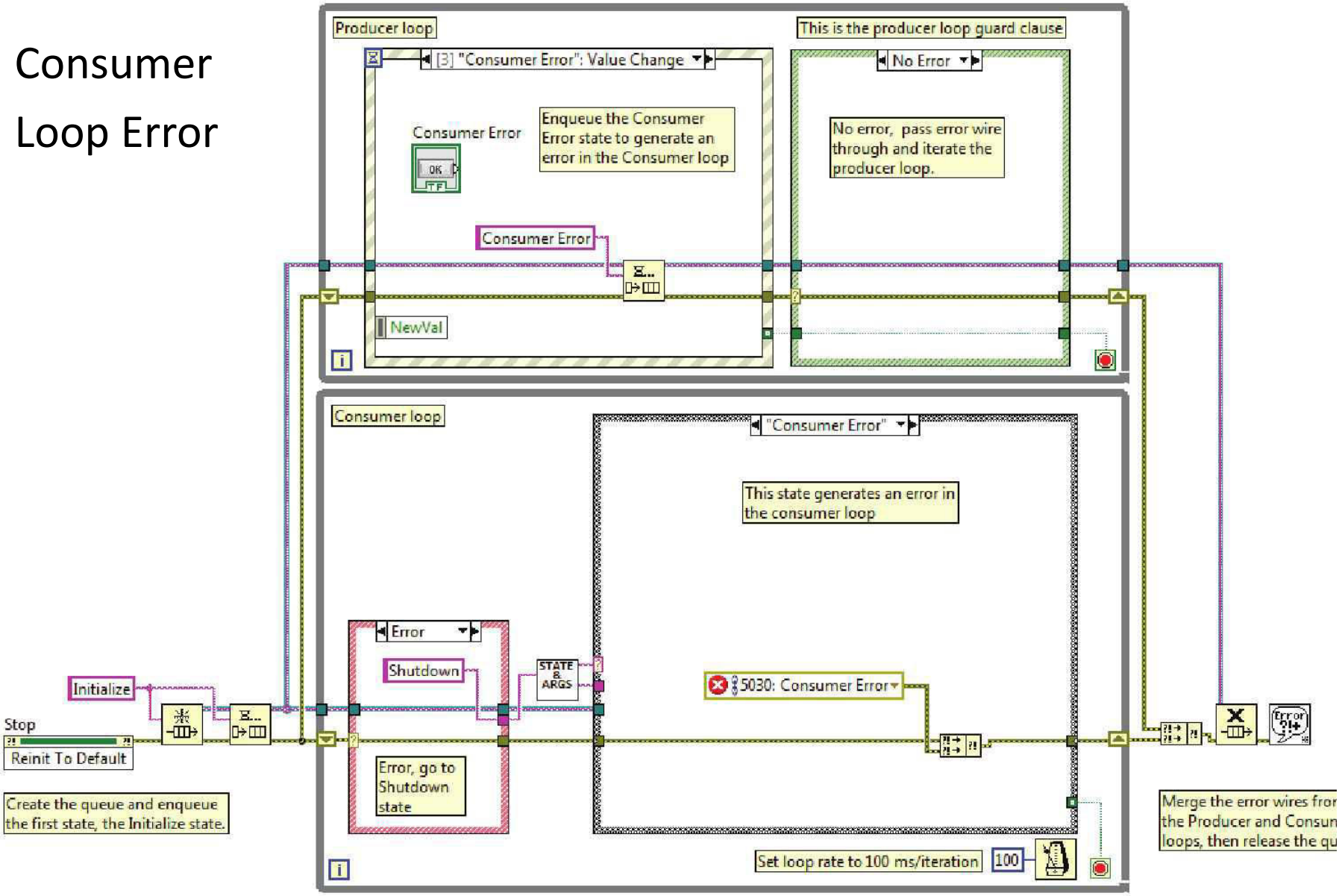
Loop Error



Producer-Consumer (Events) arhitektura

Consumer

Loop Error



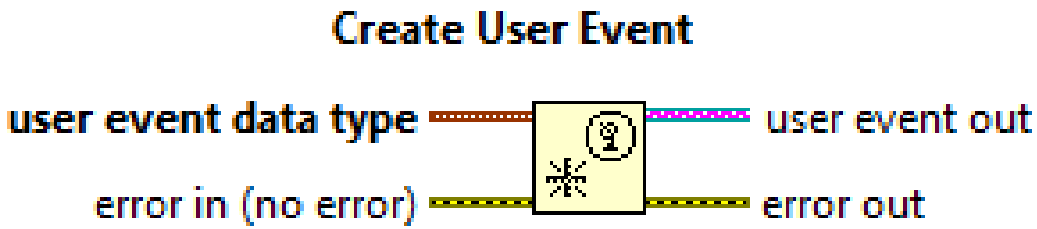
Korisnički događaji

- Do sada – Event strukture prate stanje komandi na prednjem panelu
- Moguće ih je aktivirati i programski, ali moraju biti fizički prisutne
- Korisnički događaji – generisani programski
- Više koraka:
  1. Kreiranje događaja
  2. Registrovanje – referenca za event strukturu
  3. Spajanje sa dinamičkim ulaznim terminalom
  4. Konfiguracija događaja
  5. Generisanje događaja
  6. Deregistracija (Unregister) nakon završetka petlje
  7. Brisanje događaja (Destroy) kada se završi

Korisnički događaji - funkcije

Kreiranje

- Konstanta ili kontrola određuju tip podatka
- Mora imati ime!



Returns a reference to a user event. LabVIEW uses the **user event data type** you wire to determine the event name and data type of the event. Wire the **user event out** output to a Register For Events function to register for the event. Wire the **user event out** output to a Generate User Event function to send the event and associated data to all Event structures registered for the event.

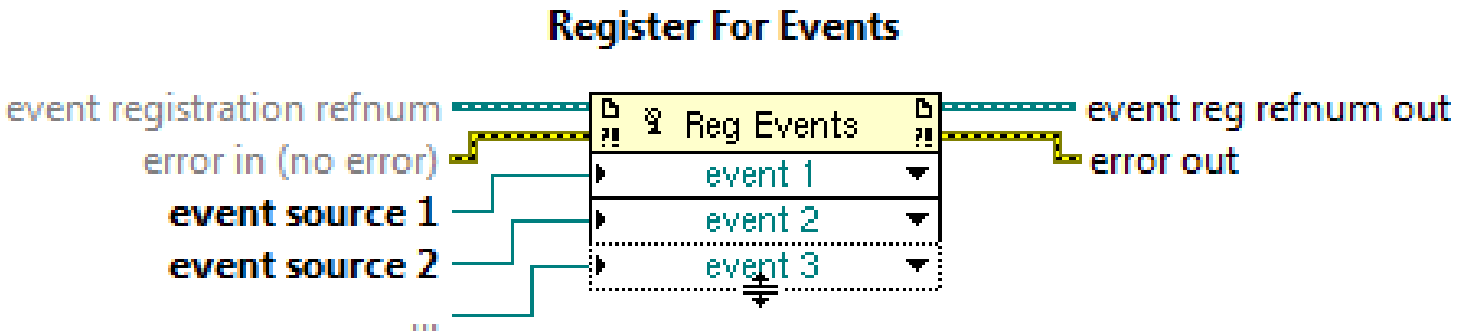
Terminal Data Type

user event data type (cluster of 0 elements)

Korisnički događaji - funkcije

Registracija

- Omogućava programsku kontrolu; koji događaji i kada će biti detektovani u Event strukturi
- Podrazumevano – detektuje sve događaje iz tekućeg VI-ja, sa njegovog prednjeg panela; Registracija dozvoljava detektovanje događaja iz drugog VI-ja
- Moguće je odrediti i pod kojim uslovima će se detektovati događaj
- Izlaz – referenca događaja; na dinamički terminal

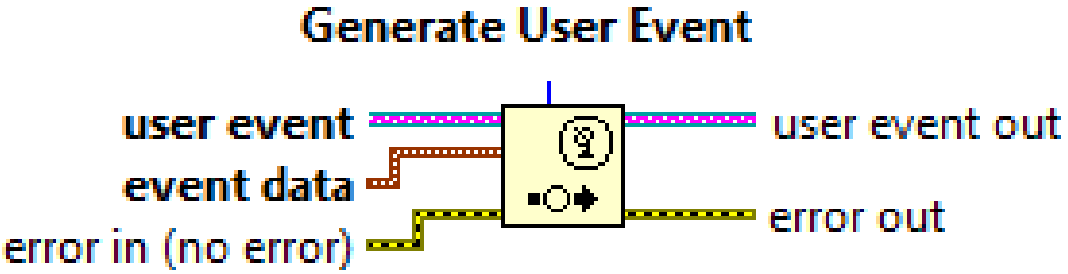


Dynamically registers events. The events for which you can register depend on the type of the reference you wire to each **event source** input. Wire the **event reg refnum out** output to an Event structure or to another Register For Events function.

Korisnički događaji - funkcije

Generisanje

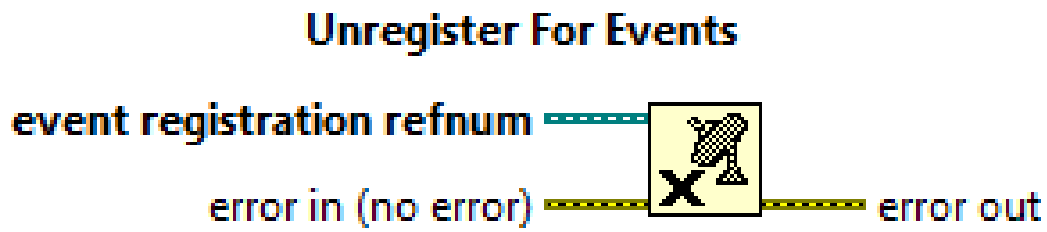
- Ulaz – referenca događaja i ulazna vrednost za definisani tip podatka
- Nakon generisanja, vrednosti događaja prenosi se zajedno sa događajem i dostupna je na Event data node unutar Event structure



Broadcasts the user event you wire to the **user event** input and sends the user event and associated event data to each Event structure registered to handle the event.

Korisnički događaji - funkcije

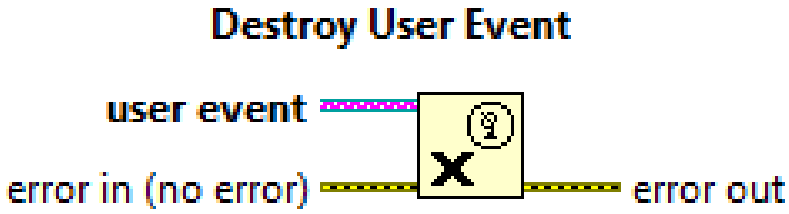
Deregistracija



Unregisters all events associated with an event registration refnum.

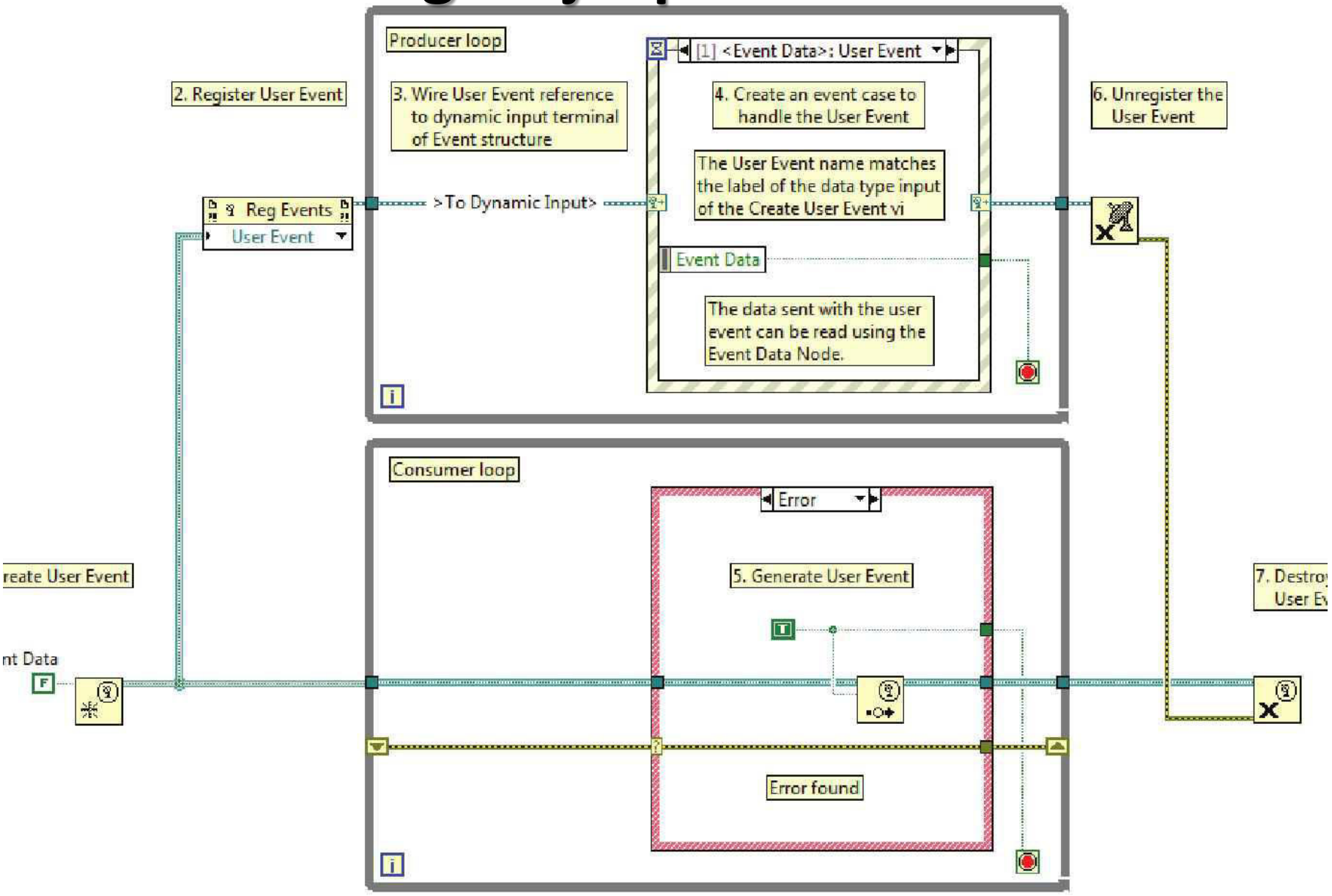
Brisanje

- Izvršava se čak i ako je greška prisutna!



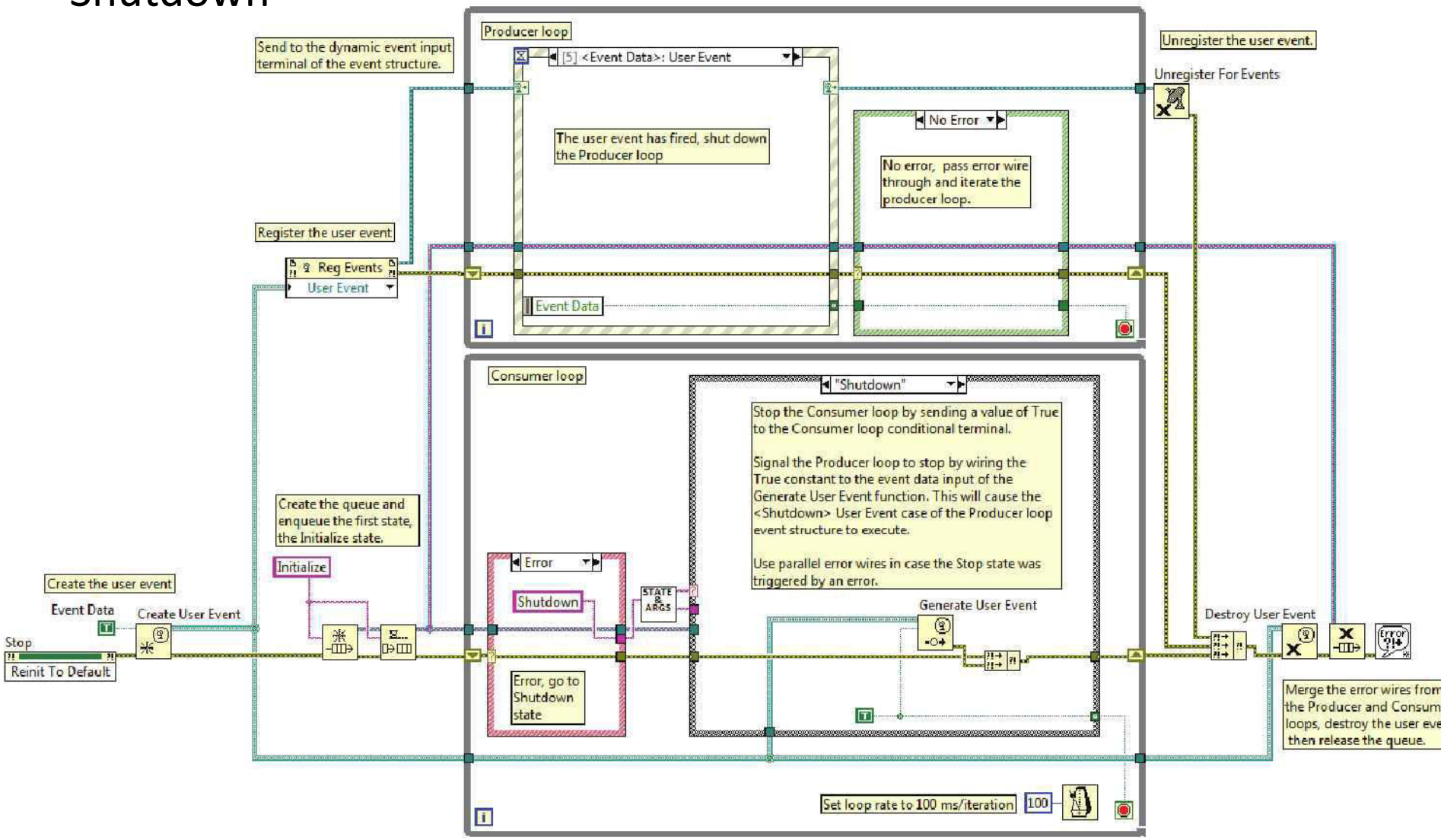
Releases a user event reference by destroying its associated user event refnum. Any Event structures registered for this user event no longer receive the event.

Korisnički događaji - primer

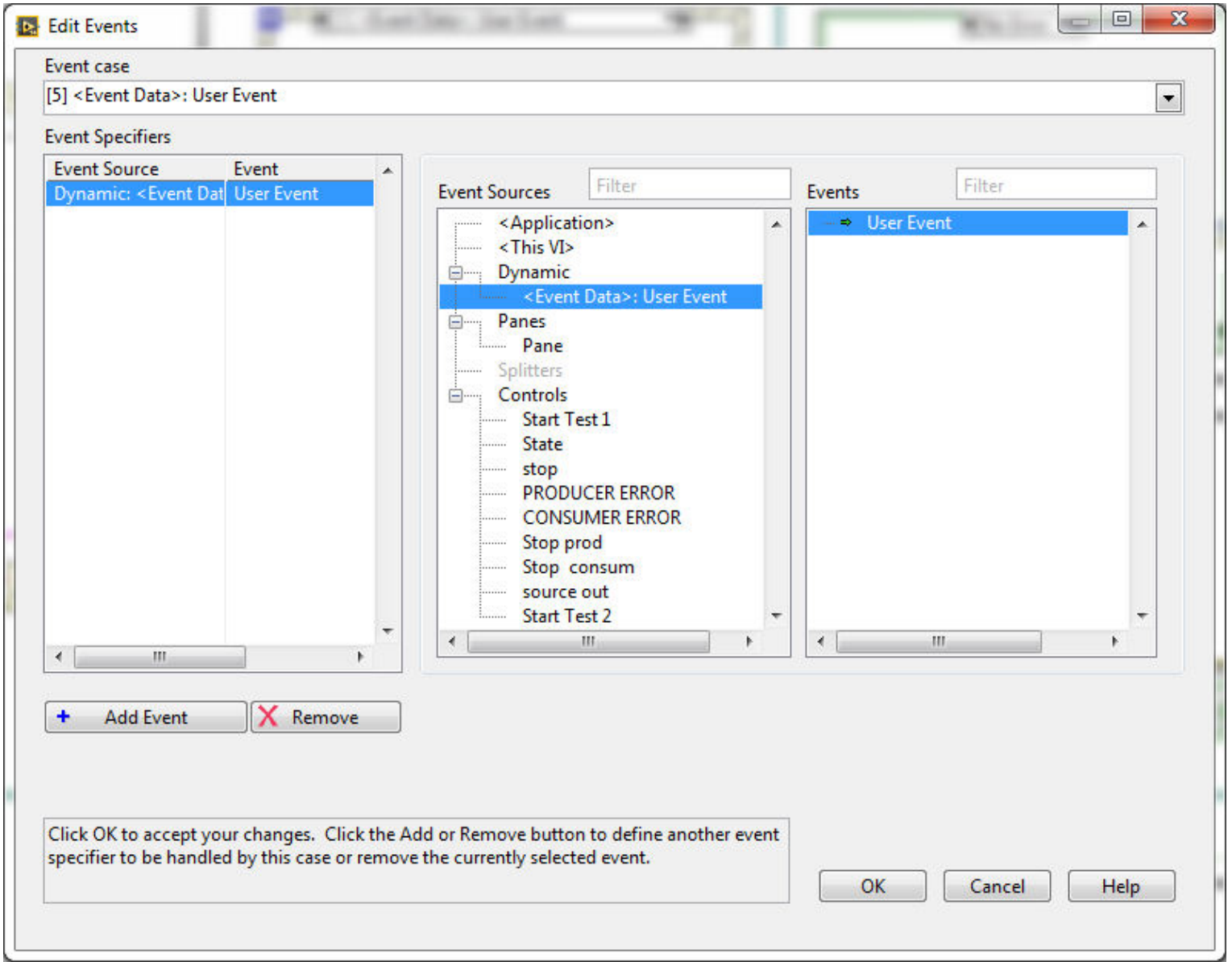


Producer-Consumer sa korisničkim događajima

Shutdown



Producer-Consumer sa korisničkim događajima



Value (Signalling) ili korisnički događaji?

- Value (Signalling):
  - Kompaktnije i jednostavnije
  - Moraju imati fizičku komponentu na front panelu; nije ih moguće primeniti ako je nema
  - Ne može se odrediti kako su aktivirane (sa panela ili programski)

- Događaji
  - Traže više truda i vremena
  - Nezavisni od prednjeg panela
  - Može se kreirati više događaja, za svaku situaciju
  - Mogu se isprogramirati različiti scenariji (normal stop, error, procedure zaustavljanja…)

Value (Signalling) ili korisnički događaji?

- Value (Signalling):
  - Kompaktnije i jednostavnije
  - Moraju imati fizičku komponentu na front panelu; nije ih moguće primeniti ako je nema
  - Ne može se odrediti kako su aktivirane (sa panela ili programski)

- Događaji
  - Traže više truda i vremena
  - Nezavisni od prednjeg panela
  - Može se kreirati više događaja, za svaku situaciju
  - Mogu se isprogramirati različiti scenariji (normal stop, error, procedure zaustavljanja…)

Procesuiranje grešaka

- Prioritet kod projektovanja VI
- Paralelne petlje su mnogo zahtevnije; mora se voditi računa o obe petlje
- Producer petlja
  - Čuvar uvek dolazi nakon Event structure
  - Pre strukture – ako Event struktura generiše grešku, ona ide na šift registar, Consumer petlja se izvršava; greška se detektuje u sledećoj iteraciji i aktivira Shutdown stanje; Event struktura čeka sledeći događaj (“spava”)
  - Posle strukture –greška je odmah detektovana i mašina se gasi; Producer ne “spava”
- Treba voditi računa o spajanju grešaka; sve moraju biti očitane na kraju!
- Merge errors – spaja informacije o greškama
- Treba ih iskoristiti i za pravilan Data flow
- Brisanje događaja i redova mora se uraditi nakon zaustavljanja svih petlji (“nizvodno” od Merge Error funkcije); tada su petlje sigurno završene!