INSTRUMENTATIE VIRTUALA

CURS 8

Arhitecturi de programare

Design Pattern - SMoRES

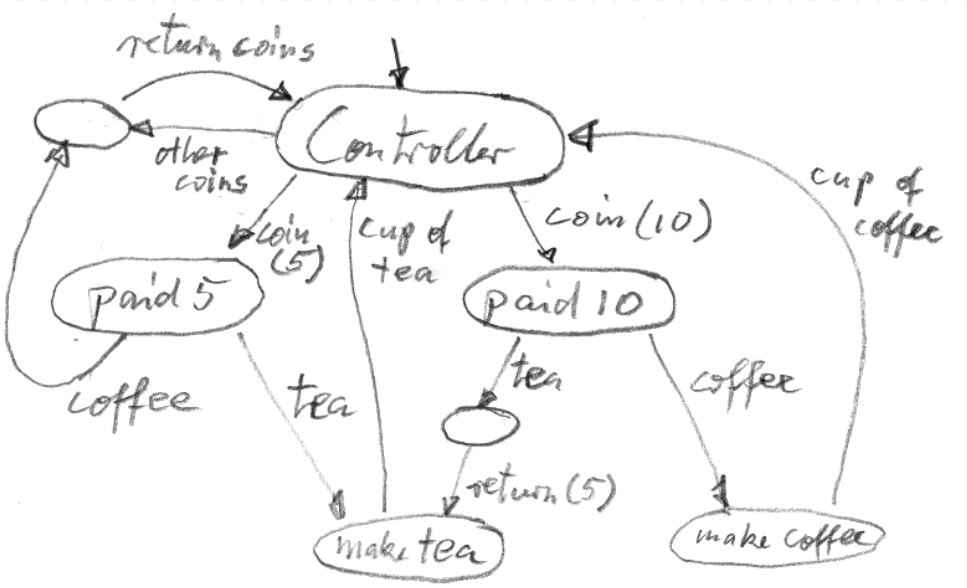
- **S**calable – cat de greu este de scalat aplicatia?
- **M**odular – cat de bine este modularizata aplicatia (modulele sunt de sine statatoare)?
- **R**eusable – codul si modulele pot fi utilizate si in alt proiect?
- **E**xtensible – cat de dificil este de adaugat facilitati in plus?
- **S**imple – care este solutia cea mai simpla care satisface toate cerintele impuse?

Design Pattern

- Se bazeaza pe
  - Utilizarea Functional Global Variable
  - Arhitectura State Machine
  - Interfete utilizator bazate pe evenimente
  - Arhitectura Producer / Consumer
  - Arhitectura Queued State Machine – Producer / Consumer

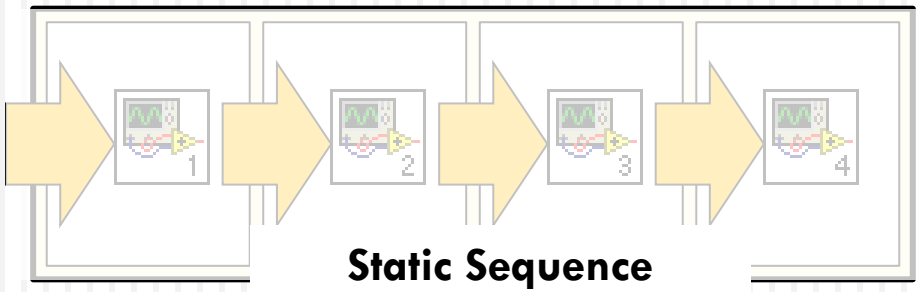
State Machine

- Este necesara executia unei secvente de evenimente, dar ordinea de executie este determinata de rezultatele/actiunile programului/utilizatorului

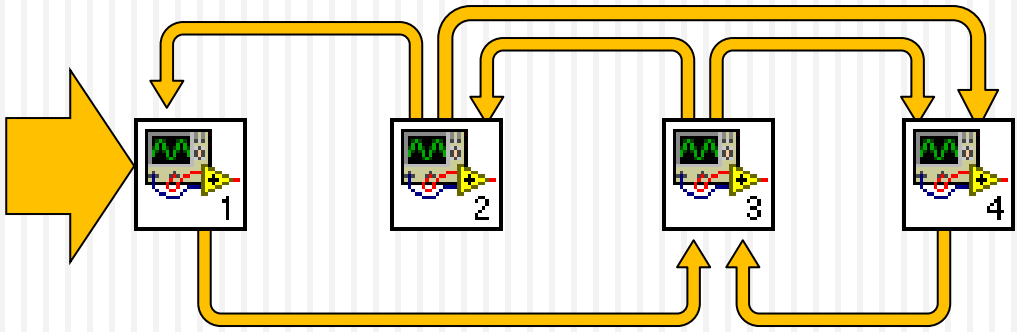


State Machine

- Implementare liniara

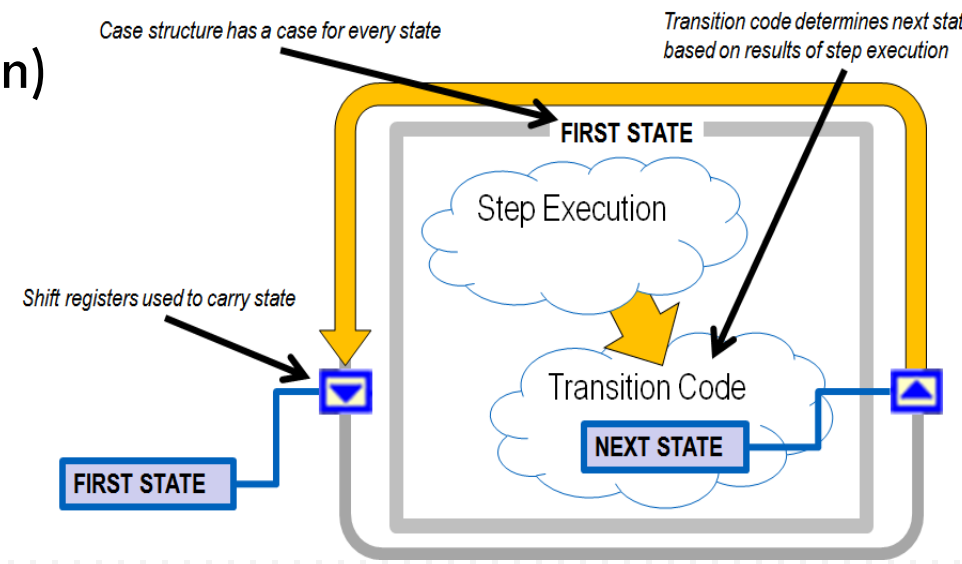


- Secventa dinamica – starile distincte pot fi executate intr-o secventa determinata programatic



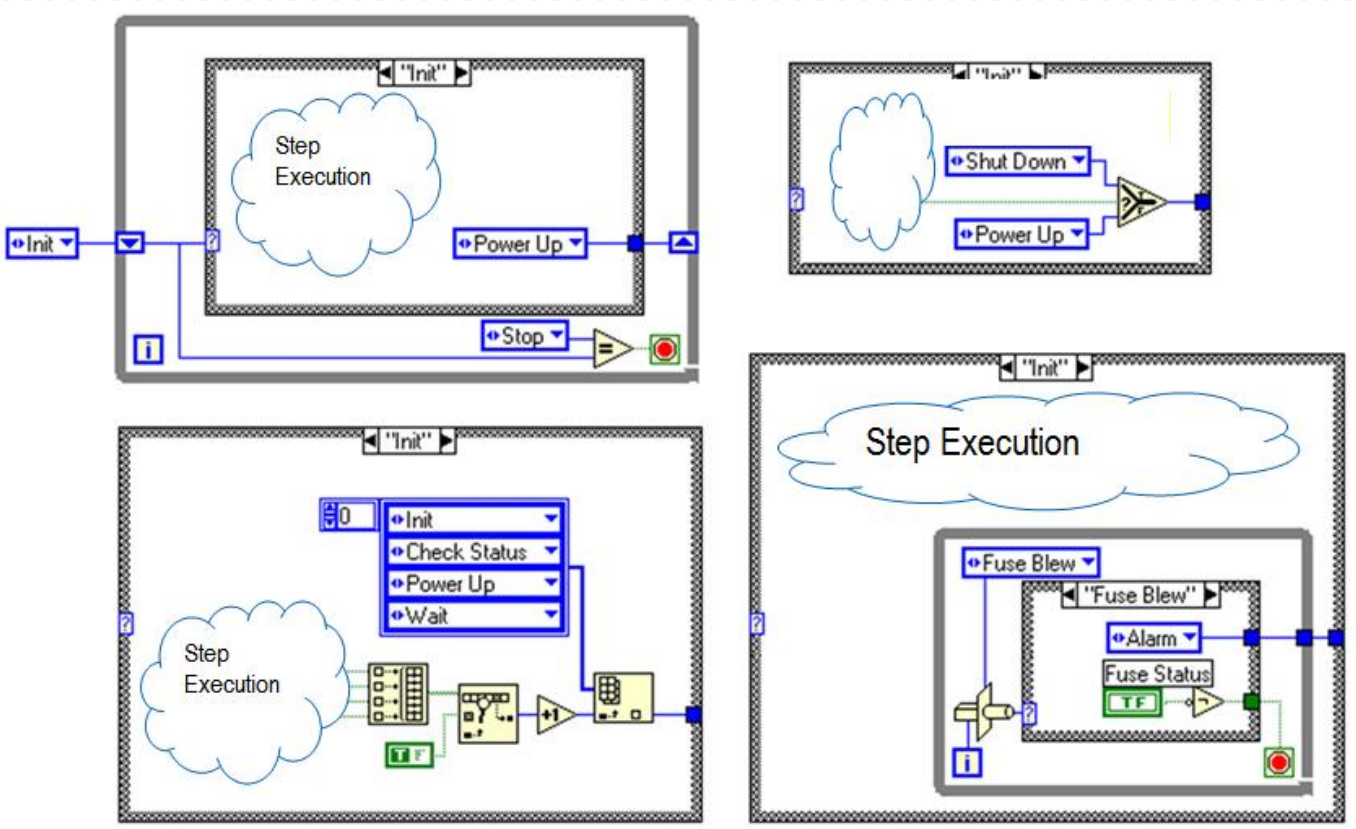
State Machine

- Concept
  - Utilizarea unei structuri Case intr-o bucla while
  - Fiecare frame a structurii Case este o stare
  - Starea curenta trebuie sa contina un cod ce determina in ce stare se va trece
  - Utilizati constante de tip enum (type definition) pentru a trimite starile ce trebuie executate (spre shift register)

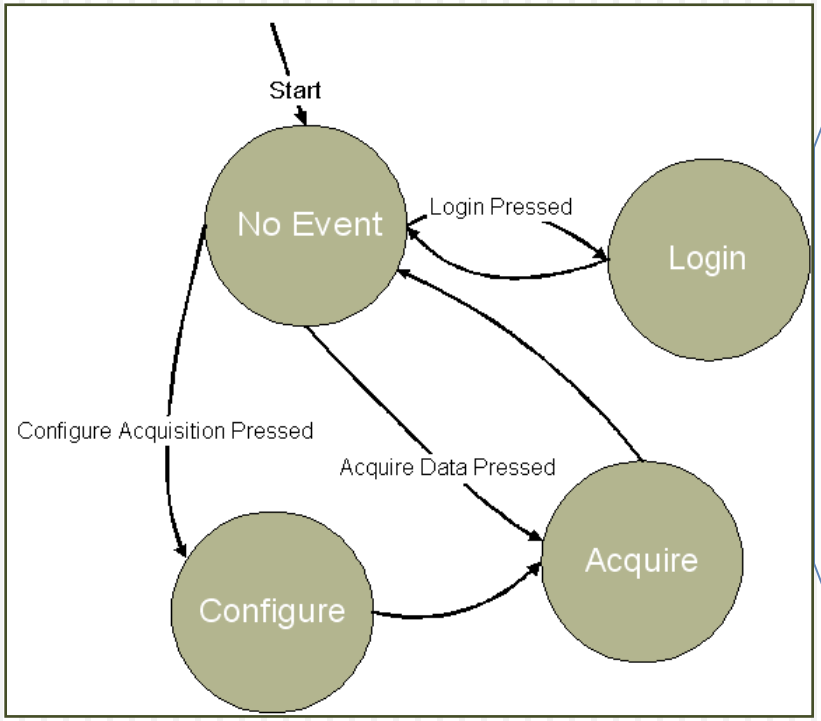


State Machine

- Optiuni de cod de tranzitie

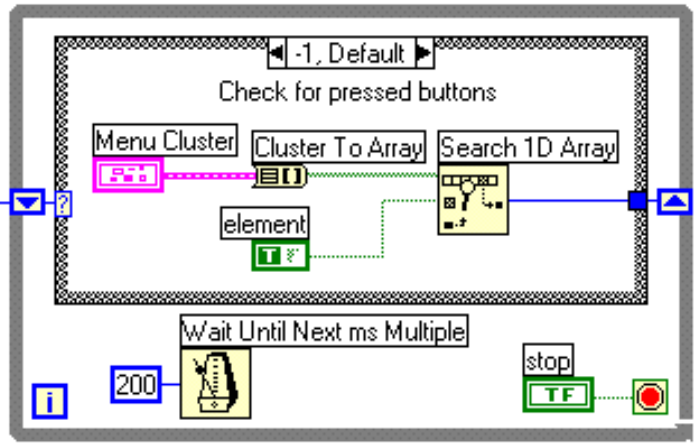


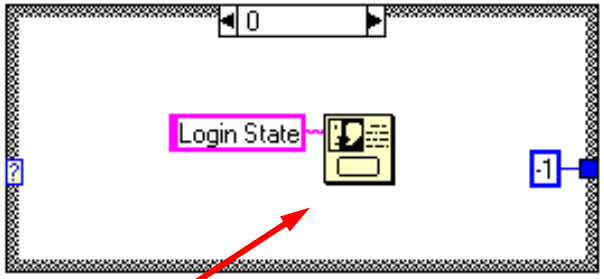
Programarea tip “State Machine”

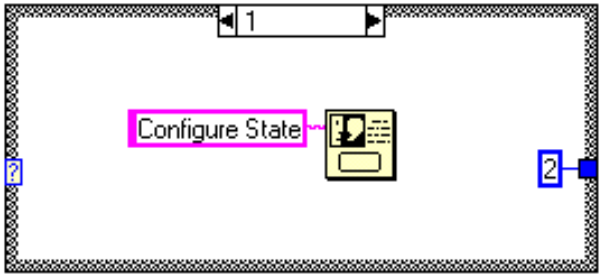


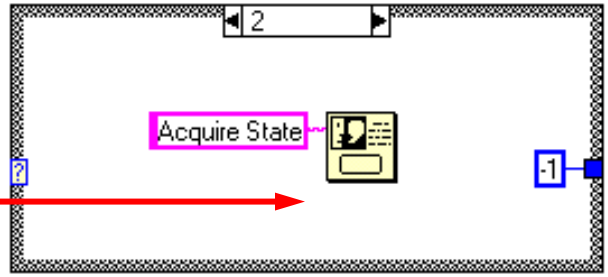
Value	State Name	Description	Next State
–1	**No Event**	Prezinta menu-ul cu butoane pentru a selecta unul din ele	Depinde de ce buton se apasa. Daca nu se apasa un buton, noua stare este: **No Event**
0	Login	Log in user	**No Event** (–1)
1	Configurare	Configurarea Achizitiei	Achizitioneaza (2)
2	Achizitie	Achizitia Datelor	**No Event** (–1)

Programarea tip “State Machine”

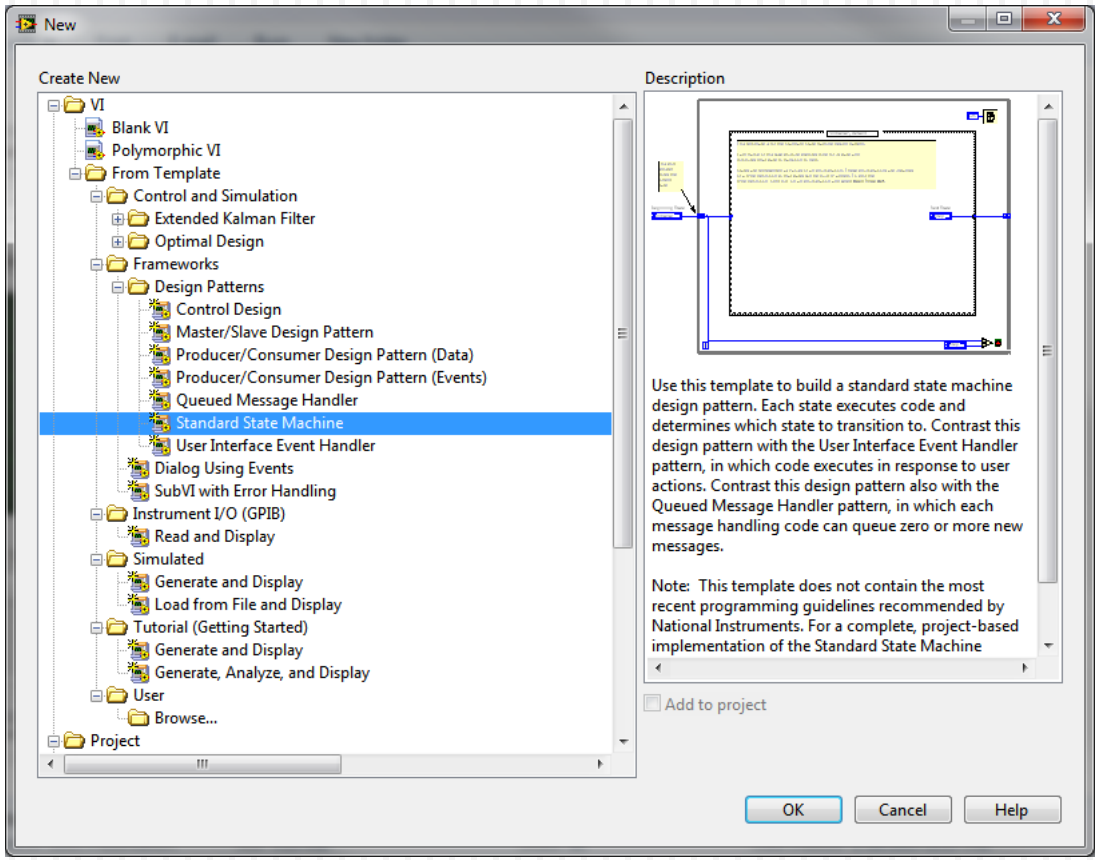


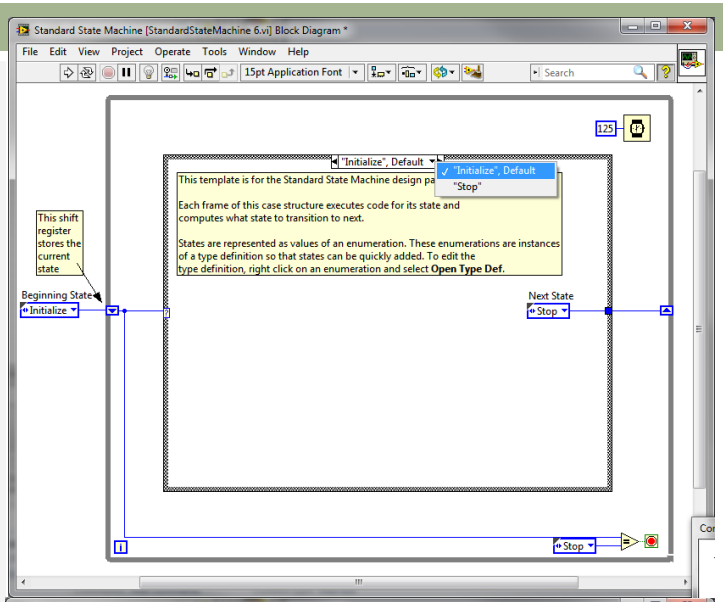


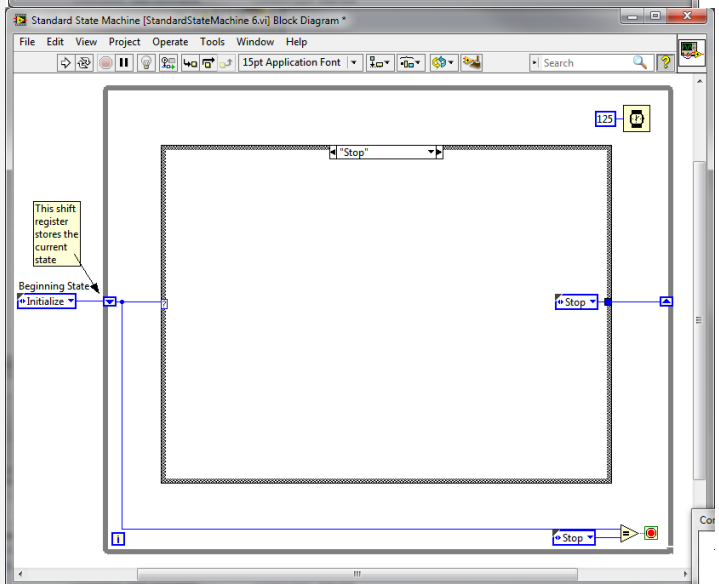




State Machine – from Template





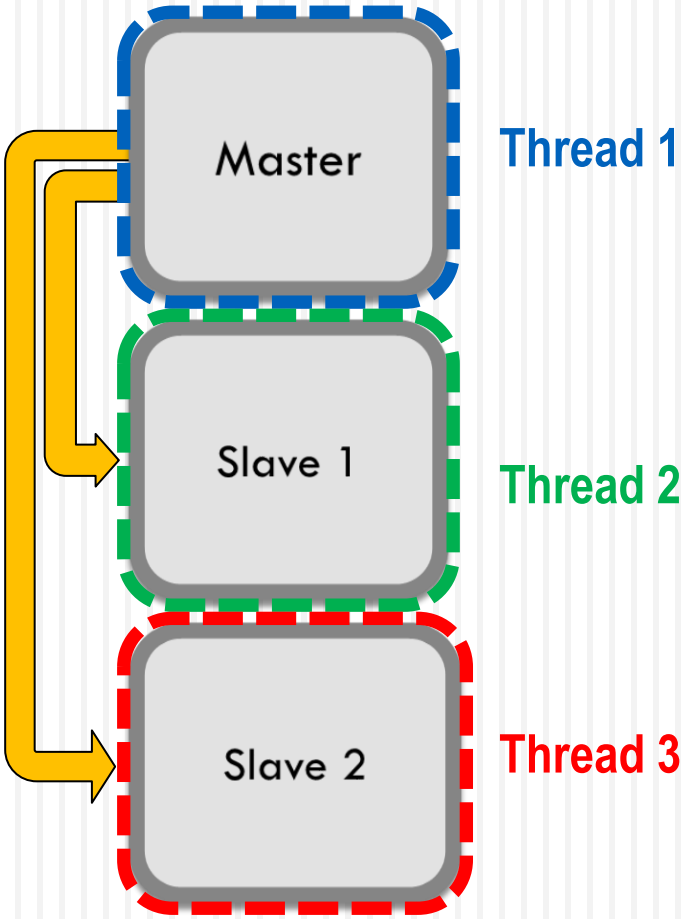


Producer/Consumer

- Utilizata in cazul in care exista doua procese ce trebuie sa se execute in paralel si nici unul dintre ele nu trebuie sa-l incetineasca/intrerupa pe celalalt
- Cele doua procese pot sa se execute asincron cu rate diferite
- Utilizati bucle independente
- Relatia dintre ele este de Master/Slave
- Trebuie realizata comunicarea si sincronizarea intre bucle

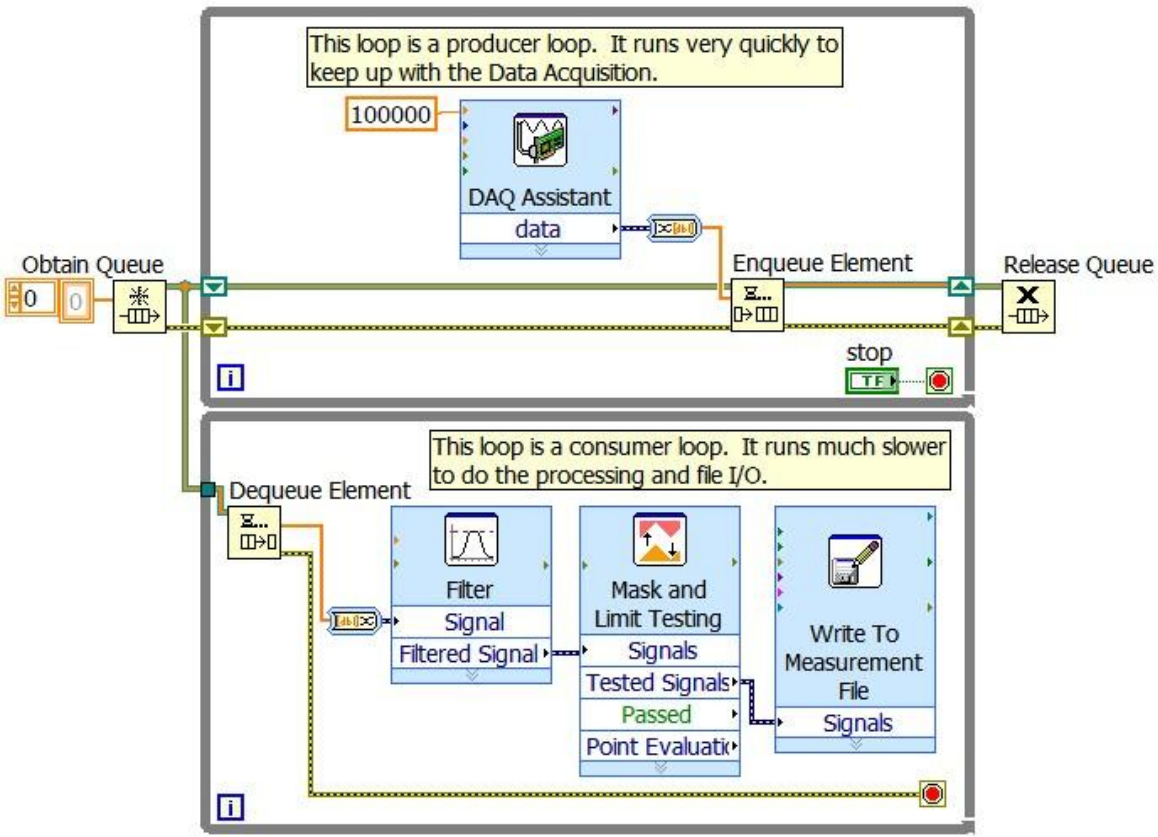
Producer/Consumer

- Bucla Master fixeaza executia uneia sau mai multor bucle Slave
- Permite executia asincrona
- Independenta buclelor intrerupe fluxul de date → permite executia multi-fir (multi-threading)
- Comunicarea se face prin:
  - Variabile
  - Cozi
  - Notificatori
- Pentru sincronizari se pot folosi si:
  - Semafoare
  - Occurrence
  - Rendezvous



Producer/Consumer

- Achizitia, analiza si salvarea datelor
- Executie multi-thread

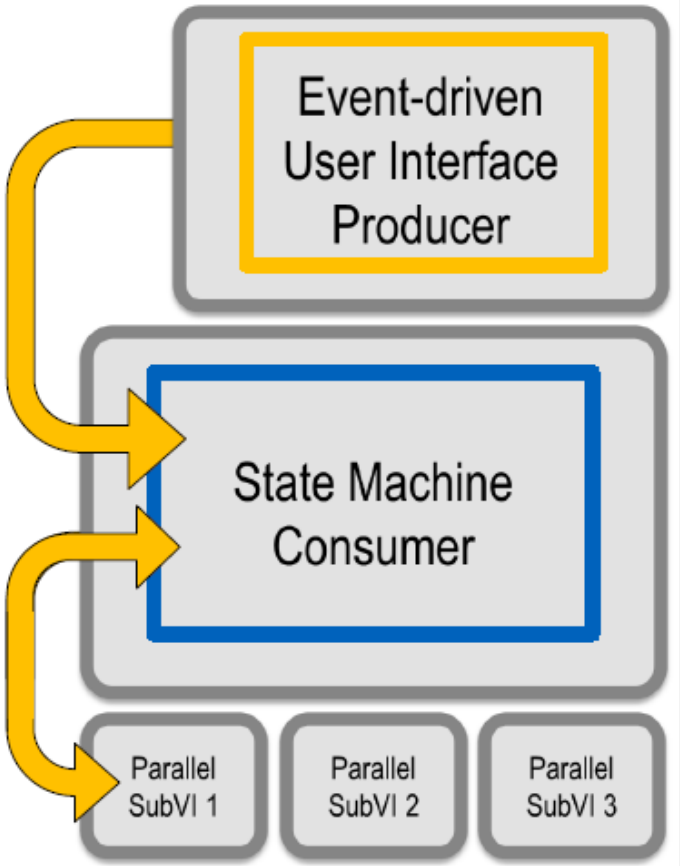


Queued State Machine & Event-Driven Producer/Consumer

- Structura complexa
- Daca este nevoie de a manipula mai multe evenimente generate de utilizator (stocate intr-o coada) ce vor controla secventa de evenimente (ordinea de executie) in bucla Consumer
- Se bazeaza pe:
  - Interfete utilizator bazate pe evenimente
  - Arhitectura State machine
  - Arhitectura Producer/Consumer
  - Comunicarea intre bucle pe baza de cozi

Queued State Machine & Event-Driven Producer/Consumer

- Mod de functionare
  - Evenimentele generate de utilizator sunt manipulate de bucla Producer
  - Producer-ul introduce datele intr-o coada
  - Din coada sunt citite datele in bucla Consumer ce are arhitectura de tip State Machine
  - Bucle sau subVI-uri ce se executa in paralel comunica prin cozi pe baza de referinte



Queued State Machine & Event-Driven Producer/Consumer