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

Podstawy Projektowania Przyrządów Wirtualnych

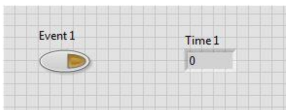
Wykład 7: Synchronizacja, zaawansowane wzorce projektowe

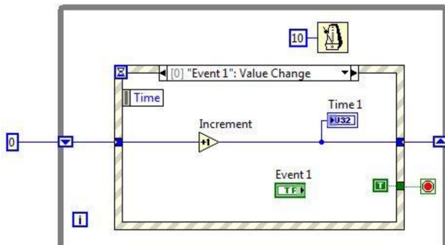
Co-funded by the
Erasmus+ Programme
of the European Union

Powtórka z wykładu 6

1/6 Po uruchomieniu VI użytkownik kliknął przycisk "Event1" dwa razy. Co zostanie wyświetlone w indykatorze "Time1" po zakończeniu działania programu?

a) 0
b) 1
c) 2
d) Nie wiadomo

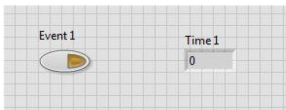


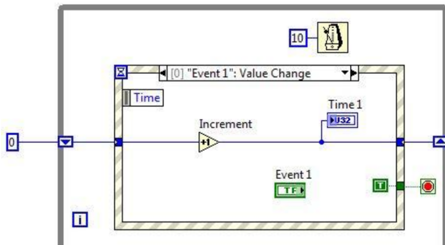


Powtórka z wykładu 6

1/6 Po uruchomieniu VI użytkownik kliknął przycisk "Event1" dwa razy. Co zostanie wyświetlone w indykatorze "Time1" po zakończeniu działania programu?

a) 0
b) **1**
c) 2
d) Nie wiadomo





Powtórka z wykładu 6

2/6 Które ze zdarzeń umożliwia wykonanie Twojego kodu przed domyślną odpowiedzą LabVIEW na zdarzenie?

a) Mouse Down
b) Panel Resize
c) Panel Close?
d) Value Change

Powtórka z wykładu 6

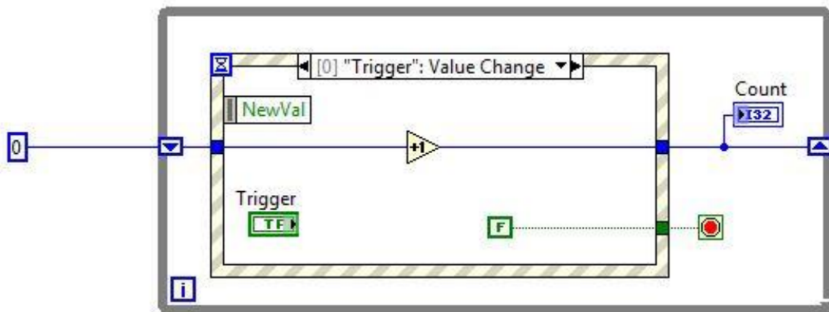
2/6 Które ze zdarzeń umożliwia wykonanie Twojego kodu przed domyślną odpowiedzą LabVIEW na zdarzenie?

a) Mouse Down
b) Panel Resize
c) **Panel Close?**
d) Value Change

Powtórka z wykładu 6

3/6 Kontrolka "Trigger" jest ustawiona w trybie switch. VI ma zwracać ile razy kontrolka "Trigger" miała wartość True podczas wykonywania się zdarzenia "Value Change". Który kod będzie wykonywał opisane założenie?

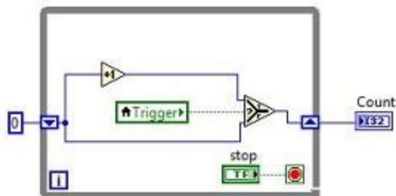
A

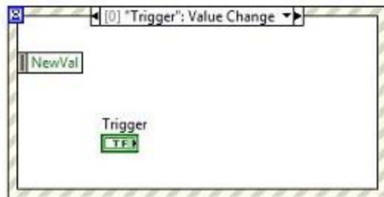


Powtórka z wykładu 6

3/6 Kontrolka "Trigger" jest ustawiona w trybie switch. VI ma zwracać ile razy kontrolka "Trigger" miała wartość True podczas wykonywania się zdarzenia "Value Change". Który kod będzie wykonywał opisane założenie?

B

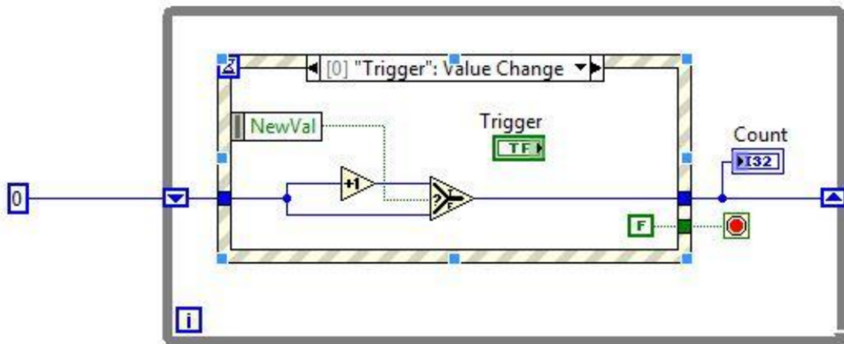




Powtórka z wykładu 6

3/6 Kontrolka "Trigger" jest ustawiona w trybie switch. VI ma zwracać ile razy kontrolka "Trigger" miała wartość True podczas wykonywania się zdarzenia "Value Change". Który kod będzie wykonywał opisane założenie?

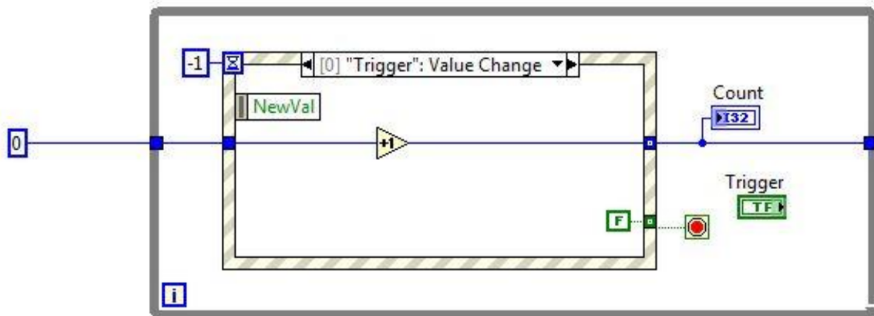
C



Powtórka z wykładu 6

3/6 Kontrolka "Trigger" jest ustawiona w trybie switch. VI ma zwracać ile razy kontrolka "Trigger" miała wartość True podczas wykonywania się zdarzenia "Value Change". Który kod będzie wykonywał opisane założenie?

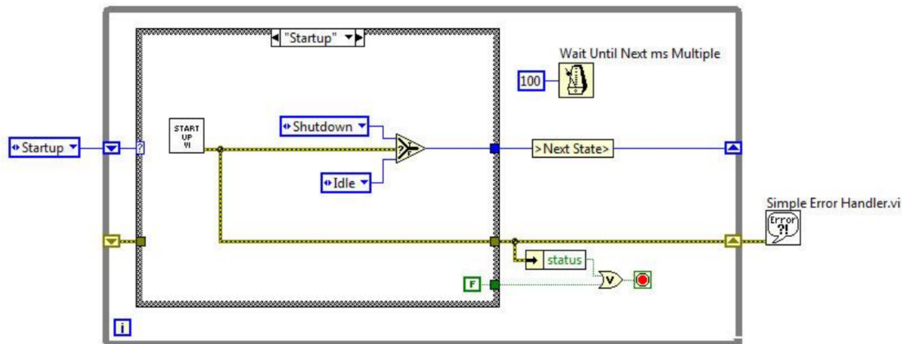
D



Powtórka z wykładu 6

4/6 Jaką architekturę reprezentuje poniższy kod programu?

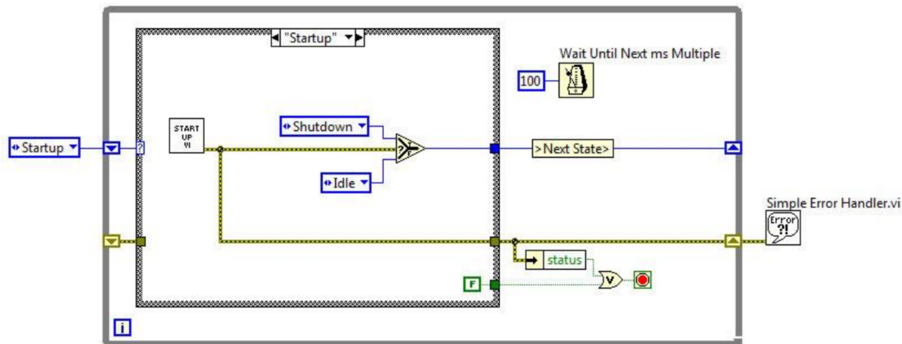
a) Wielokrotną strukturę Case (Multiple Case Structure VI)
b) Ogólny VI (General VI)
c) Maszyna Stanów(State Machine VI)
d) Równolegle pętle (Parallel Loop VI)



Powtórka z wykładu 6

4/6 Jaką architekturę reprezentuje poniższy kod programu?

a) Wielokrotną strukturę Case (Multiple Case Structure VI)
b) Ogólny VI (General VI)
c) **Maszyna Stanów(State Machine VI)**
d) Równolegle pętle (Parallel Loop VI)



Powtórka z wykładu 6

5/6 Który komponent nie jest wymagany przy tworzeniu maszyny stanów?

a) Case structure
b) While loop
c) Enum
d) Shift Register

Powtórka z wykładu 6

5/6 Który komponent nie jest wymagany przy tworzeniu maszyny stanów?

a) Case structure
b) While loop
c) **Enum - możemy użyć w zamian np. Ring**
d) Shift Register

Powtórka z wykładu 6

6/6 Jaką wadę ma architektura maszyny stanów?

a) Maszyna stanów może przekazywać stan tylko na żądanie.

b) Jeśli nastąpi podwójna zmiana stanu, tylko pierwsza z nich zostanie obsłużona. Druga zmiana stanu zostanie pominięta.

c) Kod programu znacznie się zwiększy przy przejściu z architektury ogólnej na maszynę stanów.

d) Maszyna stanów nie może przechowywać danych lub używać funkcji DAQ.

Powtórka z wykładu 6

6/6 Jaką wadę ma architektura maszyny stanów?

a) Maszyna stanów może przekazywać stan tylko na żądanie.

b) **Jeśli nastąpi podwójna zmiana stanu, tylko pierwsza z nich zostanie obsłużona. Druga zmiana stanu zostanie pominięta.**

c) Kod programu znacznie się zwiększy przy przejściu z architektury ogólnej na maszynę stanów.

d) Maszyna stanów nie może przechowywać danych lub używać funkcji DAQ.

Synchronizacja

- Do komunikacji między równoległymi procesami możemy wykorzystać zmienne (lokalne, globalne itd.).
- W niektórych przypadkach synchronizacja oparta na zmiennych może być trudna do wykonania w szczególności z powodu *Race conditions*.
- Alternatywą dla zmiennych są: *notifier* (zgłaszający) oraz queues (kolejki). Te elementy zapewniają synchronizacje pomiędzy dwoma procesami np. dwiema pętlami *While*.

Notifier (Zgłaszający)

- Służy do komunikacji między dwoma procesami.
- *Notifier* wysyła dane wraz z powiadomieniem, że dane są dostępne.
- *Notifier* może przekazywać dane każdego typu.
- Często wykorzystuje się go w wielo-pętlowych wzorcach projektowych do synchronizacji kilku pętli.
- Ważne: trzeba pamiętać, że *notifier* nie posiada bufora danych. Może przekazać pojedyncze dane.

Notifier - ważne funkcje

- **Obtain Notifier** - służy do inicjalizacji notifier-a i zwraca referencje do niego.
- **Send Notification** - wysyła wiadomość o dostępności danych do odczytu do funkcji czekających na *notifier*
- **Cancel Notification** - usuwa wszystkie dane znajdujące się aktualnie w *notifier* i zwraca wiadomość informującą o zamknięciu *notifier-a*
- **Get Notifier Status** - zwraca informacje o ostatniej wysłanej informacji do *notifier-a*.
- **Release Notifier** - uwolnienie referencji do *notifier*.
- **Wait on Notification** - oczekuje do momentu pojawienia się informacji w *notifier*.

example51.vi

File Edit View Project Operate Tools Window Help

Sine 1 Value 0,01681: Plot 0

Amplitude

Time

Sine 2 Plot 0

Sine 3 Plot 0

STOP

example51.vi Block Diagram *

File Edit View Project Operate Tools Window Help

600

Sine 1

Value

10

1. Obtain Notifier

2. Send Notification

4. Release Notifier

stop

400

Sine 2

Value

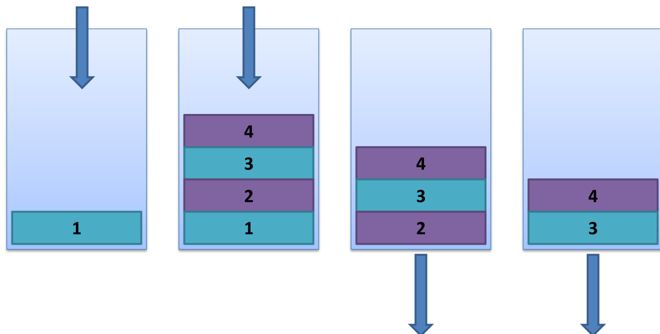
stop

3. Wait on Notification

Sine 3

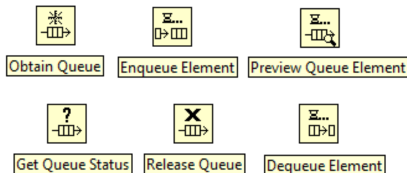
Queues (Kolejki)

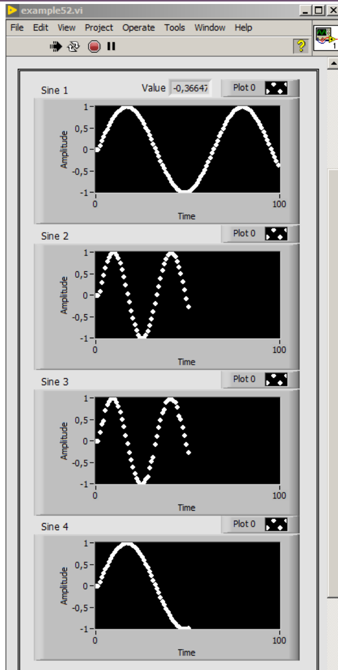
- Kolejki są wykorzystywane do komunikacji pomiędzy różnymi procesami.
- Kolejki pracują jako bufor FIFO (first in, first out).
- Mogą przechowywać dowolny typ danych.
- Powszechnie wykorzystywany do zaawansowanych wzorców projektowych.



Kolejki - ważne funkcje

- **Obtain Queue** - inicjalizuje kolejkę oraz zwraca referencje do kolejki.
- **Enqueue Element** - dodaje nowy element do końca kolejki.
- **Preview Queue Element** - zwraca następny element kolejki.
- **Get Queue Status** - zwraca liczbę elementów znajdujących się aktualnie w kolejce.
- **Release Queue** - uwalnia referencje do kolejki.
- **Dequeue Element** - usuwa aktualnie odczytany element z kolejki.





example52.vi Block Diagram *
File Edit View Project Operate Tools Window Help
200
Sine 1
Value
10
1. Obtain Notifier
4. Release Notifier
2. Send Notification
Obtain Queue
Enqueue Element
stop
Release Queue
400
Sine 2
Value
stop
3. Wait on Notification
Sine 3
Dequeue Element
Sine 4

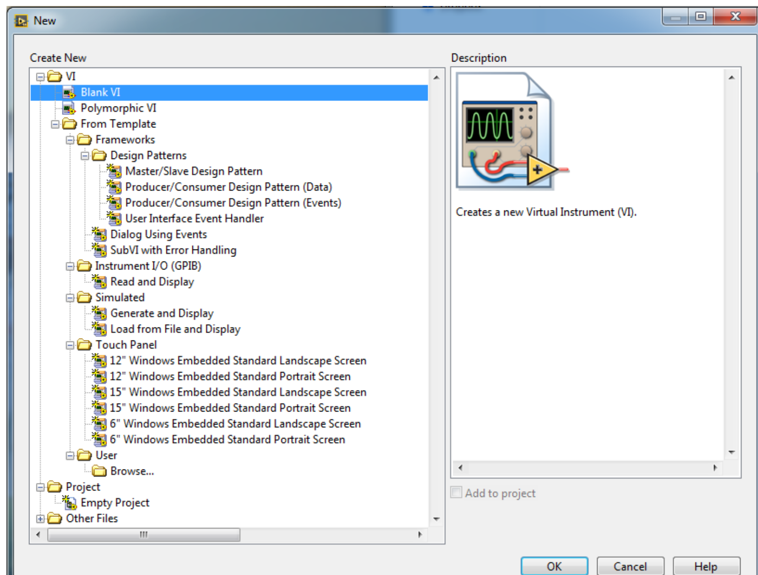
Notifier vs. Queue

	Zawiesza wykonywanie pętli do odczytu?	Bufor danych?	Dane mogą być odczytywane przez wiele pętli?
Local/global variables	Nie	Nie	Tak
Queue	Tak	Tak	Nie
Notifiers	Tak	Nie	Tak

Wzorce projektowe

Single Loop Design Patterns	Multiple Loop Design Pattern
Simple VI Design Pattern	Master/Slave Design Pattern
General VI Design Pattern	Producent/Consumer Design Pattern
State Machine Design Pattern	Functional Global Variable Design Pattern

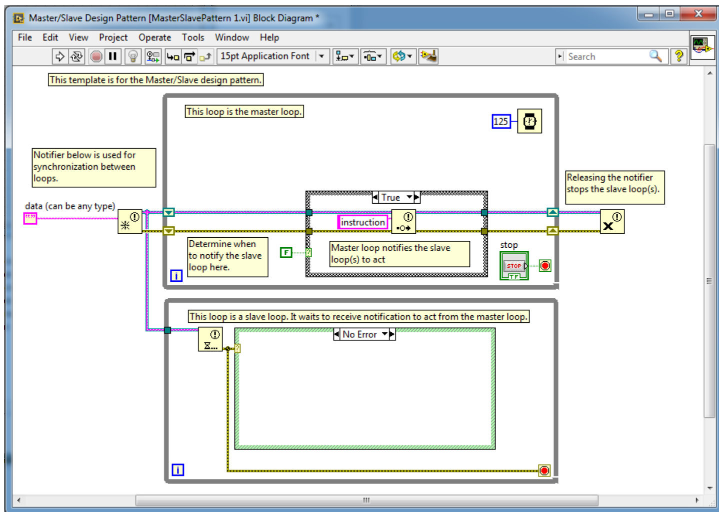
Szablony wzorców projektowych

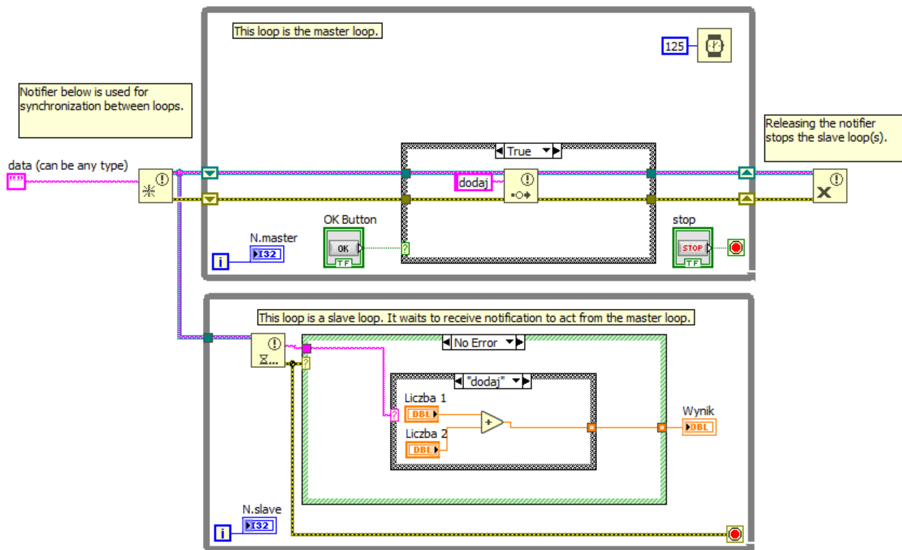


Master/Slave Design Pattern

- Master/Slave Design Pattern składa się z wielu pętli While. Każda z tych pętli może wykonywać zadania z różnymi prędkościami.
- Jedna z pętli pełni funkcje **master**, która kontroluje wszystkie inne pętle **slave** i kontaktuje się z nimi poprzez wiadomości np. przy użyciu *notifier*.
- Zastosowanie: program, w którym dane mają być zbierane równolegle do realizacji wolniejszych zadań np. obsługi interfejsu użytkownika.
- Upewnij się, że zadania realizowane przez pętle slave zajmują mniej czasu niż zadania realizowane przez master. W przeciwnym wypadku nowe wiadomości mogą nie zostać obsłużone.

Master/Slave Design Pattern

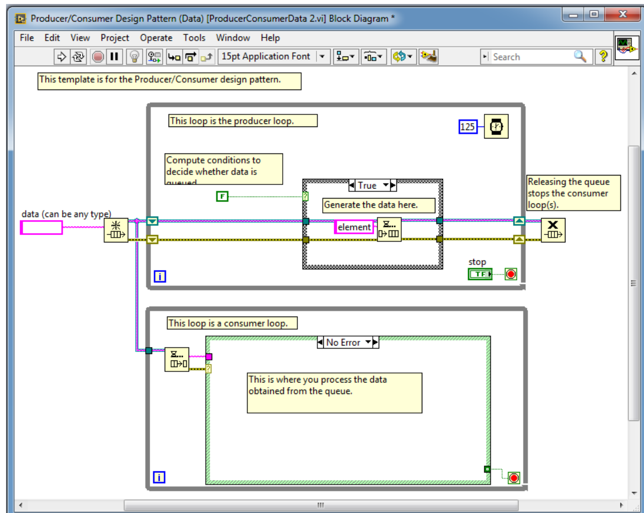


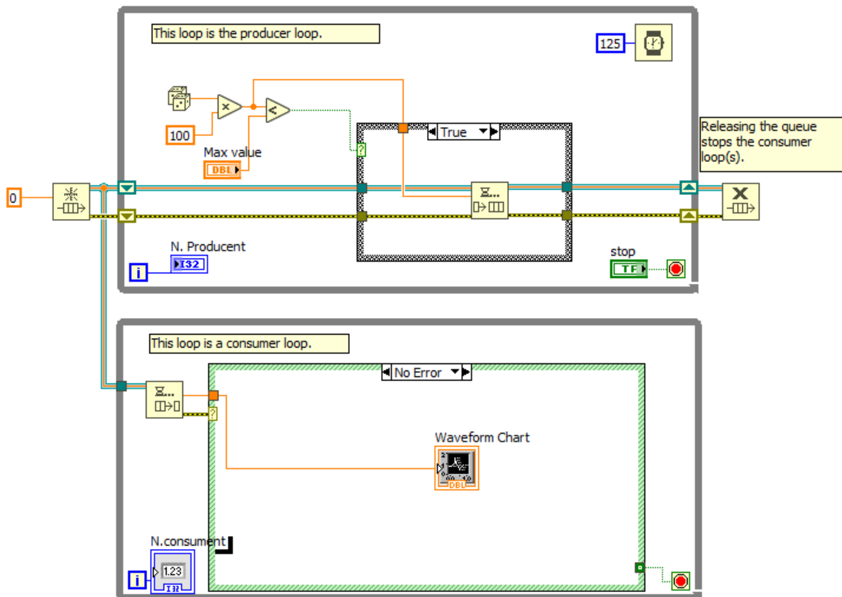

This template is for the Master/Slave design pattern.
This loop is the master loop.
125
Notifier below is used for synchronization between loops.
Releasing the notifier stops the slave loop(s).
data (can be any type)
True
dodaj
OK Button
N.master
stop
This loop is a slave loop. It waits to receive notification to act from the master loop.
No Error
"dodaj"
Liczba 1
Liczba 2
Wynik
N.slave

Producent/Consumer Design Pattern

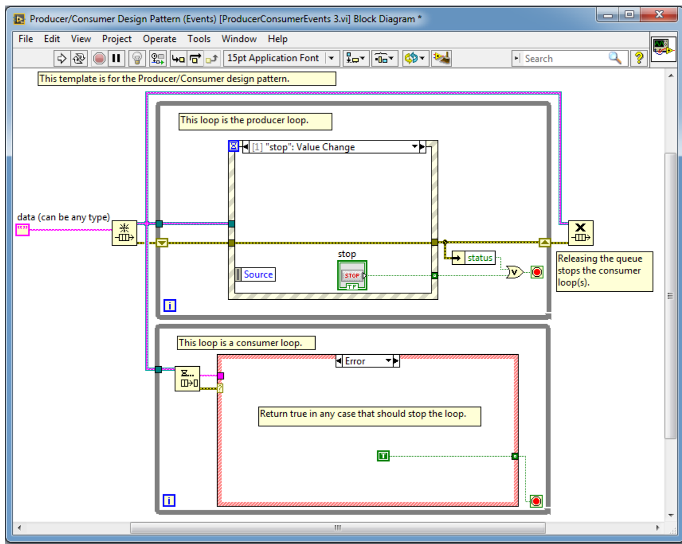
- Producent/Consumer Design Pattern jest oparty na architekturze Master/Slave.
- Udoskonalono przesyłanie danych pomiędzy pętlami wykonywającymi się z różnymi prędkościami poprzez zastąpienie notifier-a kolejką.
- Podobnie do architektury Master/Slave mamy rozdzielenie dwóch głównych zadań na dwie pętle: produkcje danych (Producer Loop) oraz konsumpcje danych (Consumer Loop).
- Ze względu na zastosowanie kolejki, możemy stworzyć jedynie jedną pętle producenta i jedną pętle konsumenta.
- Zastosowanie: w programach gdzie wiele różnych danych musi być przetwarzanych na żądanie.

Producent/Consumer Design Pattern




This template is for the Producer/Consumer design pattern.
This loop is the producer loop.
125
100
Max value
DBL
True
Releasing the queue stops the consumer loop(s).
0
N. Producent
I32
stop
TF
This loop is a consumer loop.
No Error
Waveform Chart
N.consument
1.23

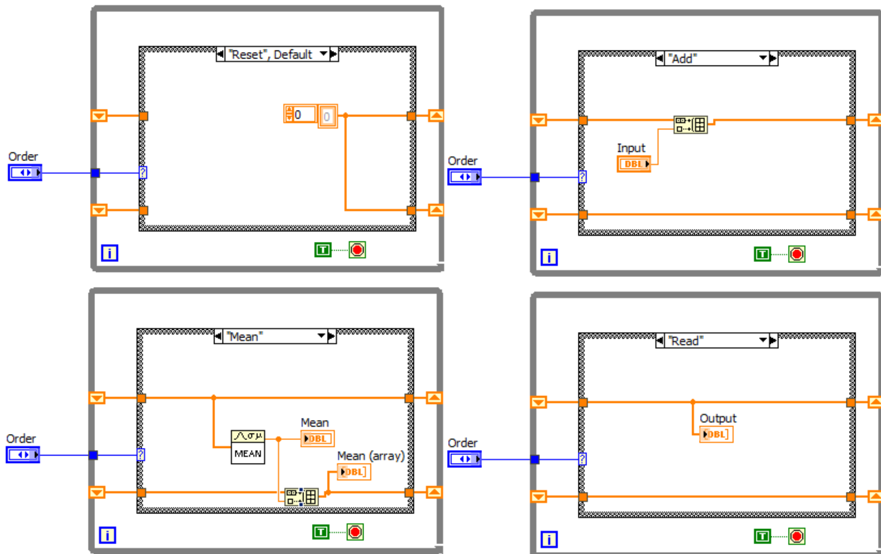
Producent/Consumer (Events) Design Pattern



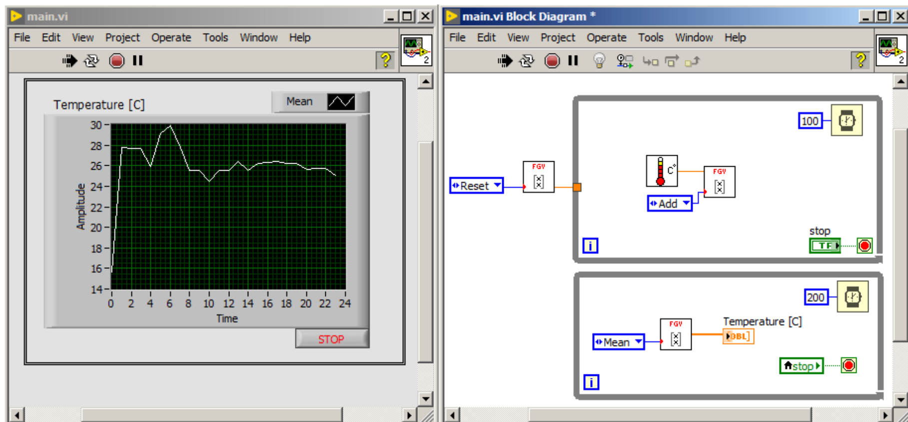
Functional Global Variable Design Pattern

- Functional Global Variable Design Pattern jest używany wtedy gdy potrzebujemy wykonać równolegle różne operacje na danych w zależności od potrzeby.
- Składa się ze zmiennej funkcjonalnej (subVI przechowujący dane w rejestrze przesuwnym) oraz równoległych pętel wykonujących operacje na danych korzystając ze zmiennej funkcjonalnej.
- Zapewnia ona idealną architekturę programu, w którym nie występuje zjawisko *race condition*.

Functional Global Variable Design Pattern - przykład



Functional Global Variable Design Pattern



Porównanie wzorców projektowych

Wzorzec	Użycie	Zalety	Wady
Simple	Obliczenia, modułowe przetwarzanie danych	Umożliwia tworzenie modułowych aplikacji	Nie odpowiedni do programów służących do interakcji
General	Dobry do tworzenia prototypów	Składa się z 3 faz: start, run i stop	Nie ma możliwości powrotu do ostatniego stadium
State Machine	Do programów sekwencyjnych	Kontroluje sekwencje, Łatwy w rozbudowie.	Brak możliwości wykonywania zadań równolegle
State Machine Event-Based	Do programów sekwencyjnych opartych na interakcji	Kontroluje sekwencje, Łatwy w rozbudowie, interaktywny.	Brak możliwości wykonywania zadań równolegle

Porównanie wzorców projektowych

Wzorzec	Użycie	Zalety	Wady
Producent/ Consumer (Data)	Do równoległych zadań	Bufor danych, równoległe procesy	Ograniczenie do przetwarzania jednego typu danych
Producent/ Consumer (Events)	Do równoległych zadań opartych na interakcji	Oddzielenie interfejsu użytkownika od zadań	Nie ma zbyt dobrej integracji nieinteraktywnych zadań
Functional Global Variable	Do projektów mających kilka VI	Przechowuje dane w pamięci	Problematyczne przy dużej ilości kopii

Kontrola czasu w wzorcach projektowych

- **Execution timing** - implementowany aby dać czas procesorowi na skończenie innych zadań. Wykonuje się go przy użyciu *timing functions* - *Wait, Wait unit Next*.
- **Software control timing** - do kontroli częstotliwości z jaką wykonuje się pętla. Wykonuje się go korzystając z funkcji *Elapsed Time*.

6_kontrola_czasu1.vi

File Edit View Project Operate Tools Window Help

Temperature

Plot 0

Temperature [C]: 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50

Time [s]: 0, 2, 4, 6, 8, 10, 12, 14

STOP

6_kontrola_czasu1.vi Block Diagram

File Edit View Project Operate Tools Window Help

"Read", Default

Temperature

DBL

0

Read

Wait

Elapsed Time

Time has Elapsed

Reset

stop

6_kontrola_czasu1.vi Block Diagram

File Edit View Project Operate Tools Window Help

13pt Application Font

"Wait"

Temperatura jest mierzona co 1s gdyż co 1s funkcja Elapsed Time zwraca True i wybierany jest kolejny stan Read

0

Read

Read

Wait

Elapsed Time

Time has Elapsed

Reset

stop

- W przypadku zastosowania wzorca z strukturą Event. Nie trzeba stosować funkcji *Elapsed Time* gdyż ta struktura zapewnia możliwość wyboru opóźnienia korzystając z wejścia *Timeout*.

Producer/Consumer Design Pattern (Events) [ProducerConsumerEvents 1.vi] Block Diagram *

File Edit View Project Operate Tools Window Help

13pt Application Font

Search

This template is for the Producer/Consumer design pattern.

This loop is the producer loop.

[1] "stop": Value Change

data (can be any type)

Source

stop

status

Releasing the queue stops the consumer loop(s).

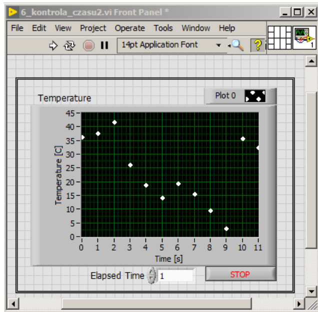
Domyślnie czeka w nieskończoność (-1) aż wystąpi zdarzenie. Te wartości można zmodyfikować!

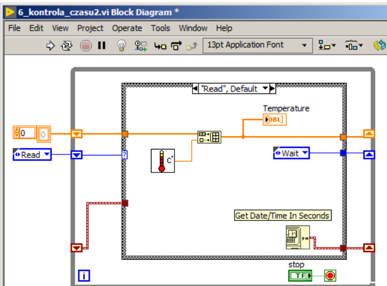
This loop is a consumer loop.

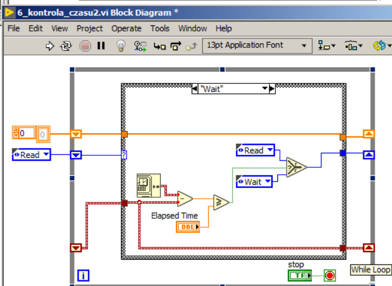
Error

Return true in any case that should stop the loop.

- W przypadku gdy nie możemy sobie pozwolić na zatrzymanie wykonywania się VI. Dobrym rozwiązaniem jest użycie funkcji **Get Date/Time.**







Dziękuję za uwagę!

Wsparcie Komisji Europejskiej dla produkcji tej publikacji nie stanowi poparcia dla treści, które odzwierciedlają jedynie poglądy autorów, a Komisja nie może zostać pociagnięta do odpowiedzialności za jakiekolwiek wykorzystanie informacji w niej zawartych.

Wykład został opracowany w oparciu o materiały: "LabVIEW Core 1 Course Manual", "LabVIEW Core 2 Course Manual", pierwotną wersje wykładu: mgr. inż. Marcina Biedy oraz przykładowe egzaminy CLAD opublikowane na stronie www.ni.com .