|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP9 | |  |
| Tytuł ćwiczenia: | | **Struktury programistyczne: pętle równoległe** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Zmienna lokalna** | 3 |
| 2.2. | **Zmienna globalna** | 4 |
| 2.3. | **Zmienna współdzielona** | 4 |
| 2.4. | **Zmienna funkcjonalna** | 5 |
| 2.5. | **Race condition** | 6 |
| 2.6. | **Metody zabezpieczenia przed efektem Race Condition** | 6 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 7 |
| 3.1. | **Obiekt badany** | 7 |
| 3.2. | **Urządzenia dodatkowe** | 7 |
| 3.3. | **Oprogramowanie** | 7 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 7 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA – zmienna funkcjonalna służąca do odliczania czasu** | 9 |
| 6. | **RAPORT** | 9 |
| 7. | **PYTANIA** | 9 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Tworzenie zmiennej lokalnej. | 3 |
| 2. | Widok zmiennej globalnej: powłoka, zmienna i panel zmiennej. | 4 |
| 3. | Widok szablonu zmiennej funkcjonalnej. | 5 |
| 4. | Zmienna funkcjonalna realizująca dwa zadania zapisu i odczytu. | 5 |
| 5. | Przykład pracy równoległej z występowaniem efektu Race Condition. | 6 |
| 6. | Zabezpieczenie przed efektem Race Condition z zastosowaniem zmiennej funkcjonalnej. | 7 |
| 7. | Zastosowanie zmiennej typu FVG do odmierzania przedziałów czasu. | 9 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. **CELE ĆWICZENIA**

Celem ćwiczenia jest poznanie typów zmiennych:

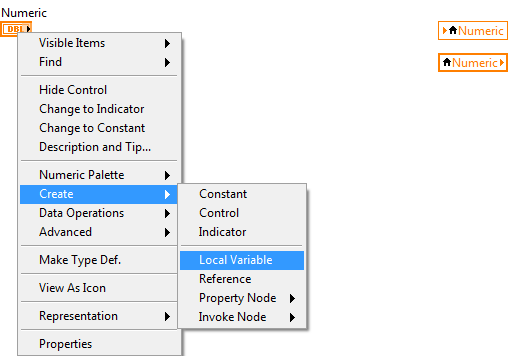
* Zmienna lokalna,
* Zmienna globalna,
* Zmienna współdzielona,
* Zmienna funkcjonalna.

1. **WPROWADZENIE**

Zmienne w środowisku LabVIEW nie są zalecanym rozwiązaniem w pisaniu kodu, powinny być używane z dużą rozwagą wtedy, kiedy inne mechanizmy i struktury nie mogą być zastosowane lub kiedy jest to uzasadnione. Jeżeli dane zadanie możliwe jest do realizacji w inny sposób powinno się zrezygnować ze zmiennych. Podstawową wadą zmiennych jest utrata ciągłości **dataflow**, co utrudnia analizę kodu oraz może powodować nieoczekiwane efekty pracy programu bardzo trudne do zdebugowania (np. tzw. wyścig – **Race Condition**).

* 1. **Zmienna lokalna**

Zmienne lokalne tworzą dodatkowe instancje obiektów panelu (kontrolki i wskaźniki). Tworzy się je klikając na terminalu kontrolki lub wskaźnika znajdującym się w oknie diagramu **PPM** 🡪 **Create** 🡪 **Local Variable**. Zmienna ma kierunek zapisu lub odczytu wartości, niezależnie od typu obiektu, z którego została utworzona. Kierunek można zmienić **PPM** 🡪 **Change to Write** lub **PPM** 🡪 **Change to Read**. Tworzenie zmiennych oraz ich wygląd zostały przedstawione na Rys. 1. Zmienna lokalna posiada zawiera ikonkę domu.



1. Tworzenie zmiennej lokalnej.

Zmienne lokalne dostępne są tylko w VI, w którym znajduje się wskaźnik lub kontrolka.

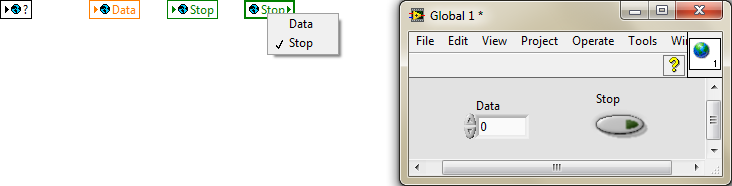
Zastosowania zmiennych lokalnych:

- wpisywanie wartości początkowych do wskaźników i kontrolek w trakcie inicjacji programu,

- Przenoszenie wartości między pętlami równoległymi – dane te nie mogą być krytyczne czasowo (np. kontrolka STOP zatrzymująca działanie aplikacji).

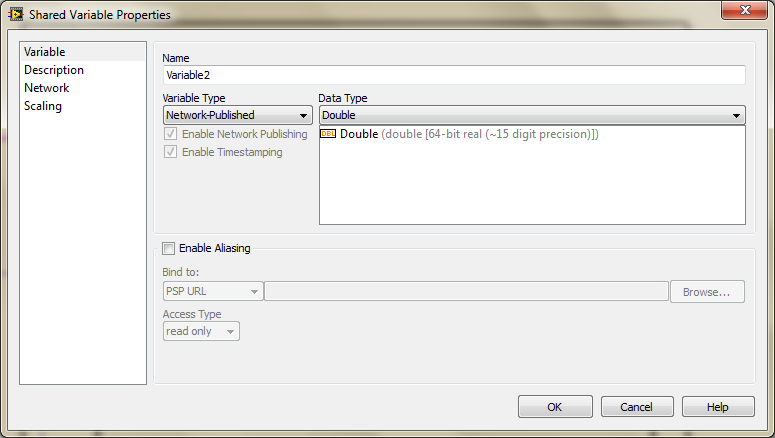
* 1. **Zmienna globalna**

Zmienne globalne są to pliki VI zawierające tylko panel, na którym umieszcza się zmienne. Umożliwiają one przenoszenie danych między równolegle działającymi plikami VI. Zmienna globalna zawiera ikonkę kuli ziemskiej. Mogą być utworzone na dwa sposoby: **New…** 🡪 **Global Variable** lub **PPM** 🡪 (paleta) **Programming** 🡪 **Structures** 🡪 **Global Variable** a następnie dwukrotnie kliknąć na powłoce węzła. Na Rys. 2 przedstawiono panel zmiennej globalnej zawierający dwie kontrolki: Data i Stop, powłokę zmiennej oraz zmienne powiązane z panelem. Wybór zmiennej ze zmiennej globalnej odbywa się poprzez kliknięcie **LPM**.



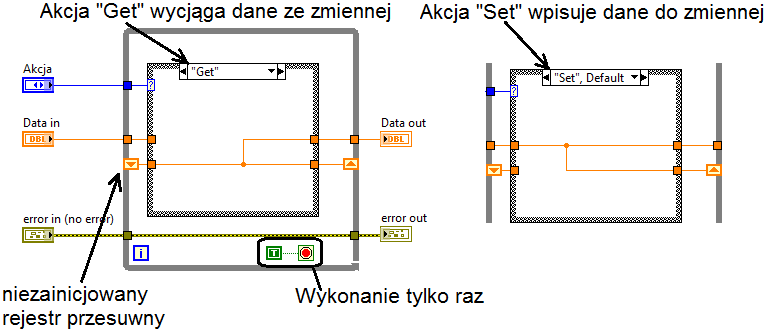
1. Widok zmiennej globalnej: powłoka, zmienna i panel zmiennej.
   1. **Zmienna współdzielona**

Zmienne współdzielone (**Shared Variable**) są tworzone w oknie projektu. Mogą być stosowane zarówno do przesyłania danych między plikami VI na jednym komputerze jak i przez sieć (**Network Shared Variable**). Węzeł zmiennej funkcjonalnej zawiera dodatkowo konektory klastra błędu. W celu konfiguracji zmiennej współdzielonej należy wypełnić szablon. Widok okna Variable szablonu został przedstawiony na Rys. 3. Typy zmiennych dostępne dla zmiennej funkcjonalnej wybiera się z listy Data Type, wśród nich można wybrać kontrolkę dedykowaną (**custom control**).



1. Widok szablonu zmiennej funkcjonalnej.
   1. **Zmienna funkcjonalna**

Zmienne funkcjonalne (**FVG** – **Functional Global Variable**) są pełnoprawnymi plikami VI. Zapamiętywanie danych jest realizowane za pomocą niezainicjowanego rejestru przesuwnego (s**hift register**). Struktura zmiennej funkcjonalnej zawiera pętlę While (wykonywaną tylko raz), strukturę Case (sterowaną za pomocą zmiennej Enum). Pętla While wymagana jest w celu zastosowania rejestru przesuwnego. Podstawowa struktura zmiennej funkcjonalnej przedstawiona jest na Rys. 4.

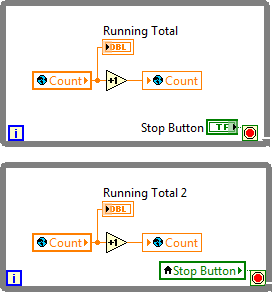


1. Zmienna funkcjonalna realizująca dwa zadania zapisu i odczytu.

Zmienna funkcjonalna umożliwia także realizację innych zadań, umożliwia zminimalizowanie efektu kolejki. Typowym zastosowaniem zmiennej funkcjonalnej jest odliczanie zadanego przedziału czasu (patrz rozdział 5). Przykładowe zastosowanie zmiennej funkcjonalnej to przesyłanie danych między pętlami np. między pętlą przetwarzania a pętlą GUI do odświeżenia wskaźników w pętli GUI.

* 1. **Race condition**

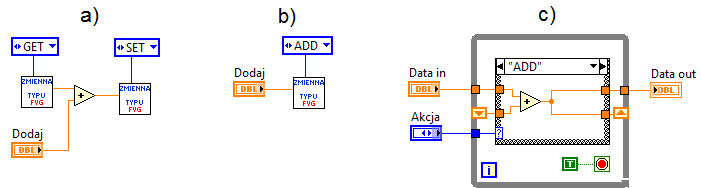
Efekt „wyścigu” pojawia się wtedy, gdy dwie części programu (lub dwa różne programy) chcą zapisać dane w tym samym czasie do tego samego zasobu – zmiennej. Efekt takiego działania jest trudny do przewidzenia. Jest to efekt bardzo trudny do wykrycia ponieważ jego występowanie jest losowe i może pojawiać się bardzo rzadko.



1. Przykład pracy równoległej z występowaniem efektu Race Condition.
   1. **Metody zabezpieczenia przed efektem Race Condition**

W celu zabezpieczenia kodu przed występowaniem efektu Race Condition należy tak projektować kod, żeby dostęp do zmiennych odbywał się z jednego miejsca na raz. W celu zablokowania krytycznych fragmentów kodu można zastosować strukturę semaforów: **PPM** 🡪 (paleta) **Programming** 🡪 **Synchronization** 🡪 **Semaphores**.

Inną metodą jest zastosowanie odpowiednio zaprojektowanych zmiennych funkcjonalnych. Na Rys. 6 przedstawiono przykład dodania liczby do zmiennej FVG. W przypadku a) kod nie jest zabezpieczony przed efektem Race Condition, w przypadku b) kod jest zabezpieczony przed efektem Race Condition, przypadek c) przedstawia realizację zabezpieczonego kodu w strukturze FVG.



1. Zabezpieczenie przed efektem Race Condition z zastosowaniem zmiennej funkcjonalnej.
2. **LABORATORYJNE STANOWISKO BADAWCZE**
   1. **Obiekt badany**

- Środowisko programistyczne LabVIEW,

* 1. **Urządzenia dodatkowe**

- brak,

* 1. **Oprogramowanie**
* LabVIEW 2013 lub nowszy

1. **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI**

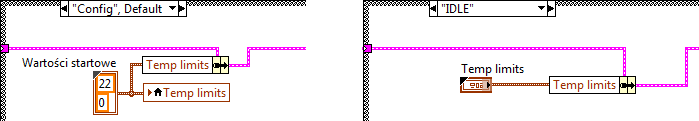
Kolejne kroki do wykonania podczas zajęć:

* Zastosowanie zmiennych lokalnych:

- otworzyć projekt „*System pomiarowy DAQ*”,

- otworzyć plik „*Main\_maszyna.vi*”,

- zmodyfikować stany *Config* i *Idle* programu „*Main\_Maszyna.vi*” w celu zastosowania zmiennej lokalnej wprowadzającej wartości początkowe (stan config) do struktury danych zgodnie z poniższym rysunkiem:



* Zastosowanie zmiennych funkcjonalnych FVG:

- utworzyć nowy plik z szablonu „*SubVI with Error handling*”,

- zapisać plik pod nazwą „S*rednia FVG.vi*”,

- przygotować kontrolkę Enum z następującymi stanami: *Config*, *Mean*, i nazwać „*Akcje*”,

- zapisać kontrolkę jako **type def** pod nazwą „*Akcje FVG.ctl*”,

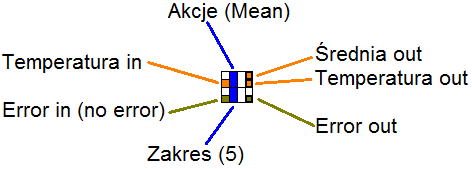
- na panelu ustawić stan *Mean* jako domyślny i zmienić nazwę na „*Akcje (Mean)*”,

- umieścić nową kontrolkę i nazwać ją „*Temperatura in*” i dwa wskaźniki „*Temperatura out*” i „*Średnia out*” – wszystkie typu **double**,

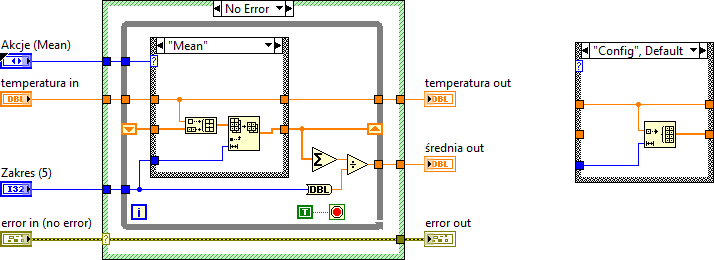
- umieścić nową kontrolkę typu **I32** na panelu i nazwać ją „*zakres*”,

- zadać wartość domyślną kontrolce *zakres* = 5,

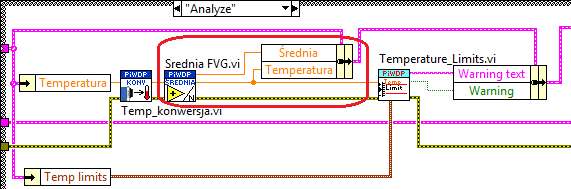
- uzupełnić panel połączeń zgodnie ze wzorem:



- uzupełnić plik kodem (stan Error uzupełnić zgodnie z uznaniem):



- uzupełnić stan *Analyze* zgodnie ze wzorem:



- utworzyć nowy plik z szablonu „*SubVI with Error handling*”,

- zapisać plik pod nazwą „Timer *FVG.vi*”,

- przygotować kontrolkę Enum z następującymi stanami: *Reset*, *Wait*, i nazwać „*Akcje*”,

- zapisać kontrolkę jako **type def** pod nazwą „*Timer FVG.ctl*”,

- na panelu ustawić stan *Wait* jako domyślny i zmienić nazwę na „*Akcje (Wait)*”,

- umieścić nową kontrolkę typu **double** i nazwać ją „*Czas (s)*” i dwa wskaźniki „*Elapsed?*” typu **boolean** i „*Elapsed Time (s)*” typu **double**,

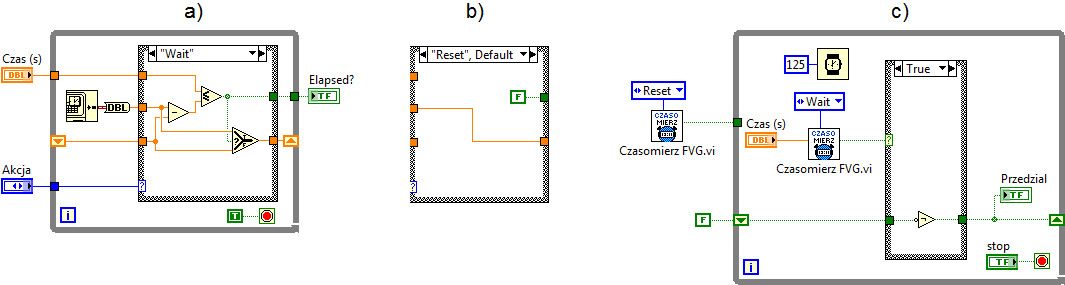
- wypełnić kodem zgodnie z przykładem,

- zmodyfikować decyzyjność stanu „*Display*” w pliku „*Main\_Maszyna.vi*” tak, aby przed upływem zadanego czasu program zapętlał się do stanu „*Measure*” a po upływie czasu wracał do stanu „*Idle*”, należy pamiętać także o dodatkowym – wcześniejszym opuszczeniu pętli za pomocą przycisku,

- uzupełnić stan „*Idle*” tak, aby zmienna „*Timer FVG.vi*” resetowała swój licznik.

1. **PRZYKŁAD REALIZACJI ZADANIA – zmienna funkcjonalna służąca do odliczania czasu**

Jednym z najpopularniejszych zastosowań FVG jest pomiar przedziałów czasu w konwencji Timing Execution. Przykład realizacji zadania został przedstawiony na Rys. 7. Rysunki a) i b) przedstawiają dwie akcje wykonywane przez zmienną FVG, rysunek c) przedstawia aplikację negującą wskaźnik Przedział co zadany czas.



1. Zastosowanie zmiennej typu FVG do odmierzania przedziałów czasu.
2. **RAPORT**

Raport z przeprowadzonego ćwiczenia laboratoryjnego powinien zawierać opis kolejnych czynności wykonywanych w trakcie realizacji ćwiczenia, zrzuty ekranu dokumentujące wykonane kroki oraz zanotowane parametry konfiguracyjne kart DAQ (mogą być zawarte  
w tabeli)

1. **PYTANIA**
2. Wymienić i omówić typy zmiennych w LabVIEW.
3. Omówić na przykładzie efekt Race Condition.
4. Wyjaśnić działanie wszystkich stanów w strukturze programu *Main\_Maszyna.vi*”.
5. Omówić przykładowe zastosowania zmiennych funkcjonalnych.

# LITERATURA

1. LabVIEW Core 1 Course manual.
2. LabVIEW Core 1 Exercise book.
3. LabVIEW Core 2 Course manual.
4. LabVIEW Core 2 Exercise book.
5. https://www.youtube.com/watch?v=iNm0zWY7o8g&list=PLUnVykytJXxPxm5u0vRKpPRVVg2u\_WgFg&index=3
6. https://www.youtube.com/watch?v=RuIN31rSO2k

Opracowanie: Marek Kciuk

# ZADANIA DO REALIZACJI PRZED PRZYSTĄPIENIEM DO ĆWICZENIA

- ukończyć zadania z instrukcji nr 7.

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

- uzupełnić strukturę danych oraz działanie programu „*Main\_Maszyna.vi*”, żeby były przechowywane wektory próbek wartości *Temperatura* i *Średnia*. – zadanie ma na celu przygotowanie programu do uzupełnienia stau „*SAVE*”.

- zmodyfikować działanie aplikacji tak, żeby sekwencja: Measure, Analyze, Display była zapętlona przez zadany okres czasu, z możliwością wcześniejszego wyjścia.

- zmodyfikować program tak, żeby przycisk „*Zakończ Pomiar*” znajdował się w strukturze kontrolki „Akcje”.

# ZAŁĄCZNIKI

## Opóźnienia i zarządzanie czasem działania aplikacji