|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP7 | |  |
| Tytuł ćwiczenia: | | **Struktury programistyczne: prosty VI, maszyna stanów** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Błąd! Nie można odnaleźć źródła odwołania.** | **Błąd! Nie zdefiniowano zakładki.** |
| 2.2. | **Błąd! Nie można odnaleźć źródła odwołania.** | **Błąd! Nie zdefiniowano zakładki.** |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 5 |
| 3.1. | **Obiekt badany** | 5 |
| 3.2. | **Urządzenia dodatkowe** | 5 |
| 3.3. | **Oprogramowanie** | 6 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 6 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA** | 7 |
| 6. | **RAPORT** | 8 |
| 7. | **PYTANIA** | 8 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Widok struktury aplikacji Simple Measurement Structure. | 3 |
| 2. |  | **Błąd! Nie zdefiniowano zakładki.** |
| 3. |  | 3 |
| 4. |  | **Błąd! Nie zdefiniowano zakładki.** |
| 5. |  | **Błąd! Nie zdefiniowano zakładki.** |
| 6. |  | **Błąd! Nie zdefiniowano zakładki.** |
| 7. |  | **Błąd! Nie zdefiniowano zakładki.** |
| 8. |  | 7 |
| 9. |  | 8 |
|  |  |  |
|  |  |  |

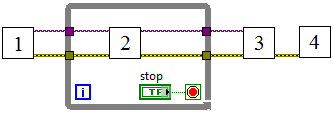
1. **CELE ĆWICZENIA**

Celem ćwiczenia jest zapoznanie się z następującymi strukturami programistycznymi:

* Prosty VI,
* Maszyna stanów.

1. **WPROWADZENIE**
   1. **Prosty VI**

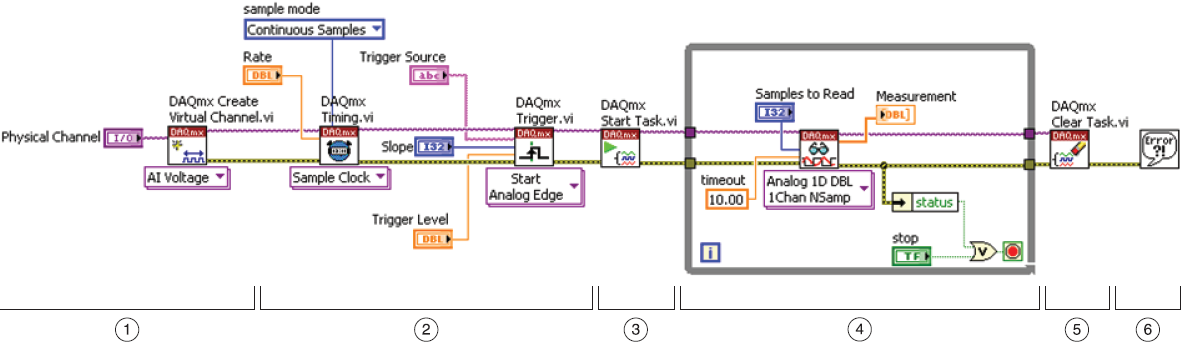
Podstawową strukturą głównego pliku aplikacji jest prosty VI, nazywany **Simple Measurement Structure**. Składa się z części konfiguracyjnej (1), właściwej programu – pętli While (2), części kończącej działanie aplikacji (3) oraz części obsługi błędów (4). Struktura programu została przedstawiona na Rys. 1



1. Widok struktury aplikacji Simple Measurement Structure.
2. – konfiguracja – w tej części konfiguruje się wszystkie kanały komunikacyjne takie jak: konfiguracja urządzenia DAQ, otwarcie referencji do pliku itp.
3. – program właściwy – w tej części znajduje się cała funkcjonalność programu, program kończy swoje działanie w wyniku zamknięcia aplikacji przez użytkownika, wystąpienia błędu, po zakończeniu wszystkich zadań,
4. – zakończenie aplikacji – w tej części kończy się połączenia otwarte w części konfiguracyjnej,
5. – obsługa błędów – w najprostszej postaci wysyła użytkownikowi komunikat o ewentualnym wystąpieniu błędu.

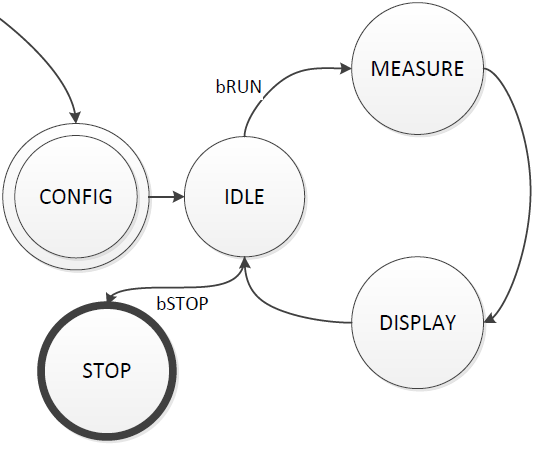
Pamiętać należy, żeby w tej strukturze stosować opóźnienie działania aplikacji (Execution Timing) – informacje w załączniku.

Przykładowy program realizujący akwizycję danych, napisany w strukturze Simple Measurement Structure jest przedstawiony na Rys. 2. Elementy 1-3 to konfiguracja



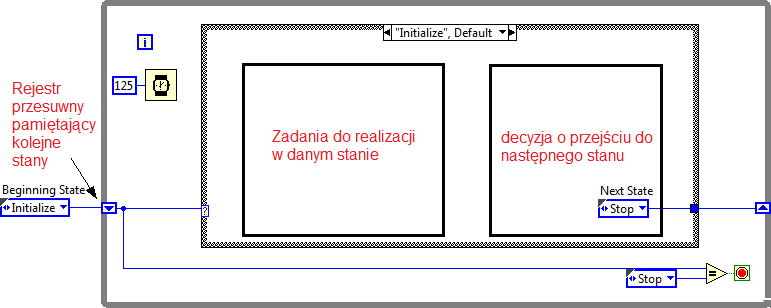
1. Przykładowa struktura Simple Measurement Structure do akwizycji danych z karty DAQ.
   1. **Maszyna stanów**

Maszyna stanów jest bardzo popularną strukturą programistyczną. Jej działanie można zaprezentować za pomocą diagramu. Diagram składa się z kółek oznaczających stany i strzałek oznaczających przejścia między stanami. Przykładowy diagram aplikacji do akwizycji i wyświetlania danych został przedstawiony na Rys. 3.



1. Diagram aplikacji akwizycji i wyświetlania danych.

Struktura maszyny stanów składa się z pętli While z rejestrem przesuwnym oraz struktury Case. W strukturze Case programowane są stany oraz definiowane warunki przejść. Kolejne stany są pamiętane w rejestrze przesuwnym. Strukturę maszyny stanów przedstawiono na Rys. 4. Z jednego stanu możliwe jest przejście do innego stanu, do dwóch stanów, lub większej ilości stanów. W przypadku wyboru między dwoma stanami można skorzystać z węzła **Select**. Jeżeli istnieje możliwość przejścia do jednego z większej niż dwóch stanów można zastosować strukturę case lub kaskadowo węzły Select, jednak lepsze i bardziej skalowalne rozwiązania zostały przedstawione na Rys. 5.



1. Struktura maszyny stanów.

|  |  |
| --- | --- |
| a)D:\PiWDP 2014\State_machine_przejscia.png | b)D:\PiWDP 2014\State_machine_przejscia.png |
| c)D:\PiWDP 2014\State_machine_przejscia.png | d)D:\PiWDP 2014\State_machine_przejscia.png |
| 1. Metody wyboru kolejnych stanów | |

Przykładem maszyny stanów jest struktura sterowana zdarzeniami (Event Driven Statemachine [5]).

Pamiętać należy, żeby w tej strukturze stosować opóźnienie działania aplikacji (Execution Timing) – informacje w załączniku.

1. **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 klastrów:

- otworzyć projekt „*System pomiarowy DAQ*”,

- otworzyć plik „*temp konwersa.vi*”,

- przenieść powłokę klastra na pulpit,

- nazwać klaster *Temp limits*,

- w klastrze umieścić dwie kontrolki typu double o nazwach *Min* i *Max*,

- zmodyfikować kod zastępując dwie kontrolki zakresu temperatury jedną kontrolką typu klaster,

- zapisać klaster jako definicję typu **PPM** 🡪 **Advanced** 🡪 **Customize…**,

* zmienić typ kontrolki na *Type def.*,
* zapisać plik pod nazwą *Params.ctl*,
* założyć nowy wirtualny katalog w oknie projektu *Kontrolki*,
* przenieść plik *Params.ctl* do katalogu *Kontrolki*,

- umieścić klaster na panelu połączeń (**Connector Pane**),

- zadać wartości domyślne takie same jakie miały wcześniej kontrolki,

- uzupełnić informacje w dokumentacji pliku,

- zapisać plik,

- poprawić kod w pliku *main.vi*,

- przygotować kontrolkę typu **Type def.** zawierającą następujące komponenty:

(można skorzystać z **File** 🡪 **New…** 🡪 **Other Files/Custom Control**)

* Temperatura (double),
* Średnia (double),
* Warning (boolean),
* Warning text (String),
* Params.ctl,

- zapisać nową kontrolkę pod nazwą *Data.ctl*,

- sprawdzić kolejność elementów w kontrolce *Data.ctl* **-** **PPM** 🡪 **Reorder Controls In Cluster**,

- umieścić kontrolkę w wirtualnym katalogu *Kontrolki*,

* Zastosowanie wektorów:

- z głównej pętli programu (**While**) wyprowadzić dane w postaci wektora danych (wartości zmierzone i wartości średnie),

- wprowadzić dane do następnej pętli **While**,

- napisać funkcję umożliwiającą użytkownikowi wybranie dowolnego fragmentu danych i obejrzenie ich na wykresie,

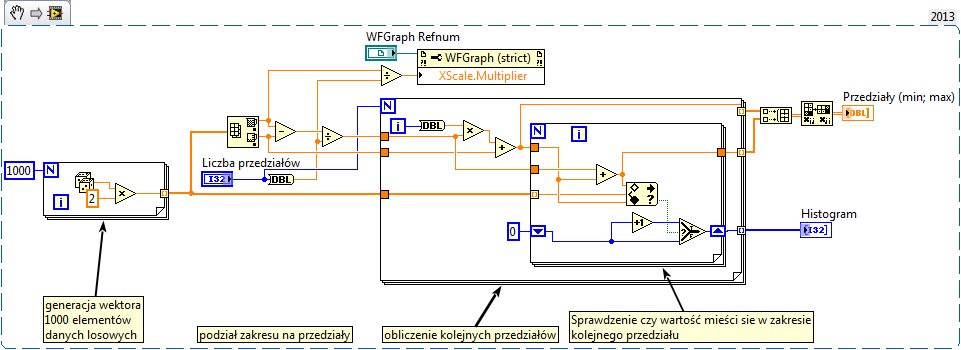
- napisać funkcję umożliwiającą użytkownikowi przeskalowania danych,

- napisać funkcję umożliwiającą użytkownikowi zmianę offsetu danych,

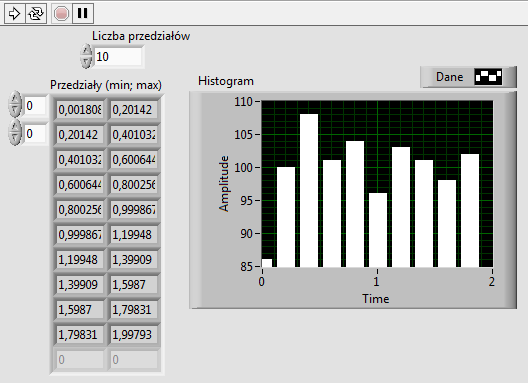
- napisać funkcję liczącą histogram danych z zadaną rozdzielczością,

1. **PRZYKŁAD REALIZACJI ZADANIA – obliczanie histogramu danych**

Przykładowa realizacja obliczania histogramu danych została przedstawiona na Rys. 5. Algorytm został podzielony na trzy części: obliczenie zakresów, przejście przez kolejne przedziały (zewnętrzna pętla **For**), sprawdzenie czy wartość mieści się w aktualnym zakresie (wewnętrzna pętla **For**). Wadą tego rozwiązania jest konieczność przeszukiwania całego zbioru danych tyle razy, ile wynosi liczba przedziałów.



1. Przykładowa realizacja zadania obliczania histogramu – kod programu.



1. Przykładowa realizacja zadania obliczania histogramu – panel programu.
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 podobieństwa i różnice pętli **While** oraz **FOR**.

# 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.

Opracowanie: Marek Kciuk

# ZADANIA DO REALIZACJI PRZED PRZYSTĄPIENIEM DO ĆWICZENIA

- ukończyć zadania z instrukcji nr 6.

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

- napisać program realizujący opóźnienie Software Timing za pomocą

# ZAŁĄCZNIKI

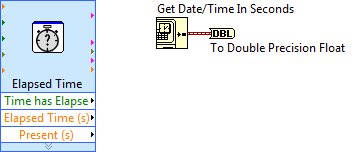
## Opóźnienia i zarządzanie czasem działania aplikacji

W LabVIEW stosuje się dwie podstawowe metody zarządzania czasem. Są to **Execution Timing** i **Software Timing**.

- **Execution Timing** – jest to zarządzanie szybkością działania i odpowiedzi (tzw. responsywność) aplikacji. Celem jest zwolnienie aplikacji, aby umożliwić procesorowi wykonanie innych działań związanych z obsługą zadań systemu operacyjnego, jednakże czas ten nie może być zbyt długi, żeby użytkownik aplikacji nie odczuwał skutków opóźnienia. Przeciętny czas opóźnienia w execution timing wynosi ok. 200ms. Funkcję tą realizuje się za pomocą węzłów **Wait (ms)** oraz **Wait Until Next ms Multiple**, przy czym drugi węzeł umożliwia dodatkowo prostą synchronizację pętli równoległych.



- **Software Timing** – jest to zarządzanie czasem oraz częstotliwością wykonywania określonych funkcji w programie np. pomiar temperatury co określony czas (np. co 10 min). W trakcie oczekiwania na kolejne zdarzenie program nie może zostać wstrzymany, co spowodowałoby zablokowanie GUI. Funkcję tę realizuje się za pomocą węzłów **Elapsed Time** (SuperVI) lub **Get Date/Time In Seconds**. Nie zaleca się w tym celu stosowania węzła **Tick Count (ms)**.



Przykład realizacji obydwu funkcji został przedstawiony na rysunku. Sekcje Execution i Software zostały oddzielone. Czas Execution Timing został ustawiony za pomocą stałej na wartość 250ms, czas Software Timing jest regulowany za pomocą kontrolki **Time Target (s)**.

Inną metodą stosowaną do pomiaru czasu Software Timing jest zastosowanie zmiennej funkcjonalnej (**FVG** – **Functional Global Variable**).

