|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP03 | |  |
| Tytuł ćwiczenia: | | **Podstawy obsługi środowiska  – Podstawowa struktura programistyczna** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Pętla While jako podstawowa struktura programistyczna** | 3 |
| 2.2. | **Opóźnienie programowe – execution timing** | 4 |
| 2.3. | **Działanie pętli równoległych** | 5 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 6 |
| 3.1. | **Obiekt badany** | 6 |
| 3.2. | **Urządzenia dodatkowe** | 6 |
| 3.3. | **Oprogramowanie** | 6 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 6 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA – „*Aplikacja Pomiarowa 2.vi*”** | 8 |
| 6. | **RAPORT** | 9 |
| 7. | **PYTANIA** | 9 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Widok struktury aplikacji Simple Measurement Structure. | 3 |
| 2. | Przykładowa struktura Simple Measurement Structure do akwizycji danych z karty DAQ. | 4 |
| 3. | Pomoc kontekstowa dotycząca węzłów opóźniających pracę programu. | 5 |
| 4. | Kod realizujący działanie dwóch niezależnych pętli równolegle. | 5 |
| 5. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. |  |  |
| 9. |  |  |
| 10. |  |  |
| 11. |  |  |
| 12. |  |  |
| 13. |  |  |

1. **CELE ĆWICZENIA**

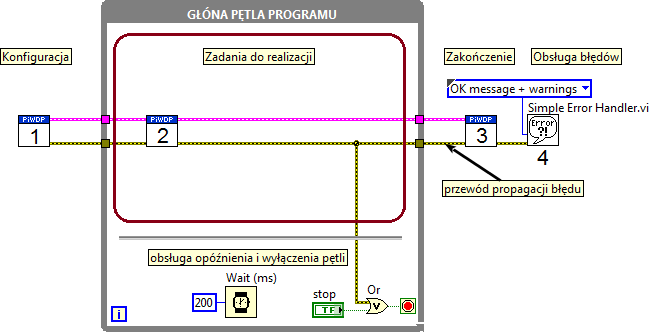
Celem ćwiczenia jest zapoznanie się z podstawową obsługą środowiska LabVIEW:

* Poznanie podstawowej struktury programistycznej – Simple Measurement Structure

1. **WPROWADZENIE**
   1. **Pętla While jako podstawowa struktura programistyczna**

Programy napisane w środowisku LabVIEW wykonują się zgodnie z przepływem danych czyli tzw. **DATAFLOW**. Podstawowe wiadomości zostały przedstawione w poprzedniej instrukcji.

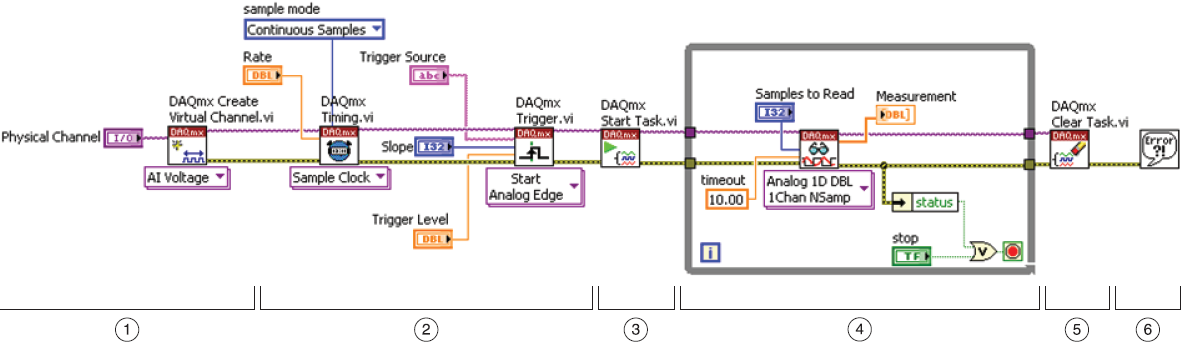
Uruchomiony program powinien działać aż do jego zatrzymania przez użytkownika (względnie do wystąpienia błędu, którego skutkiem ma być wymuszenie zamknięcia aplikacji). Podstawowym rozwiązaniem w środowisku LabVIEW jest tworzenie aplikacji w strukturze pętli **While**. Pętla **While** wykonuje zawarty w niej kod do czasu wystąpienia warunku zakończenia (wyjścia z pętli). Jest więc podstawową pętlą, na bazie której tworzy się różne struktury programistyczne. Najprostszą strukturą programistyczną jest **Simple Measurement Structure**. Składa się z: części konfiguracyjnej (1), właściwej części programu – pętli While (2), części kończącej działanie aplikacji (3) oraz części obsługi błędów (4). Struktura programu została przedstawiona na Rys. 1. Kolejne iteracje pętli wykonują się co ok. 200ms, czas ten jest określony za pomocą węzła **Wait (ms)**, wyjście z pętli następuje po naciśnięciu przycisku stop lub (węzeł **Or**) wystąpieniu błędu.



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 zamyka się połączenia otwarte w części konfiguracyjnej,
5. – obsługa błędów – w najprostszej postaci wysyła użytkownikowi komunikat o ewentualnym wystąpieniu błędu.

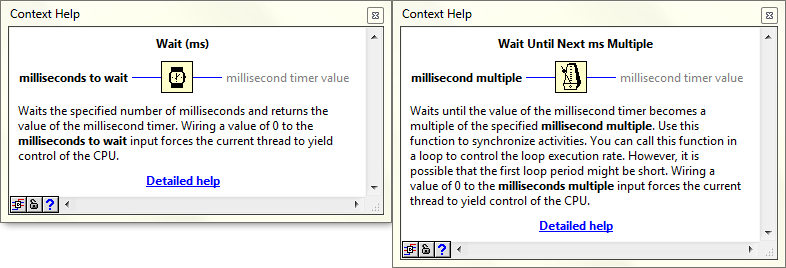
Pamiętać należy, żeby w tej strukturze stosować opóźnienie działania aplikacji (Execution Timing).

Przykładowy program realizujący akwizycję danych, napisany w strukturze **Simple Measurement Structure** jest przedstawiony na Rys. 2. Elementy 1-3 to konfiguracja, 4 – program główny, 5 – zakończenie, 6 – obsługa błędów.



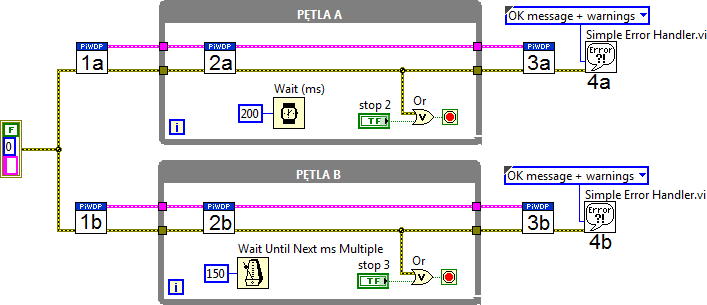
1. Przykładowa struktura Simple Measurement Structure do akwizycji danych z karty DAQ.
   1. **Opóźnienie programowe – execution timing**

Dobrym przyzwyczajeniem jest, aby w aplikacjach napisanych w środowisku LabVIEW uwzględnić czas przerwy, w którym system operacyjny będzie realizował inne zadania. Czas ten powinien być dobrany w taki sposób, aby użytkownik aplikacji nie zauważył tego opóźnienia. Dobiera się go w zakresie do ok. 300ms i realizuje za pomocą węzłów **Wait (ms)** lub **Wait Until Next ms Multiple**. Opis dostępny w pomocy kontekstowej został przedstawiony na Rys. 3. Węzeł drugi umożliwia także synchronizację działania pętli równoległych, między którymi nie zachodzi wymiana danych. Obsługa opóźnienia aplikacji związana z zapewnieniem czasu dla innych zadań systemowych nazywa się **Execution Timing** i jest bardzo istotna.



1. Pomoc kontekstowa dotycząca węzłów opóźniających pracę programu.
   1. **Działanie pętli równoległych**

Środowisko LabVIEW jest w sposób naturalny przystosowane do realizacji zadań w sposób równoległy. Z zasad dotyczących DATAFLOW przedstawionych w poprzedniej instrukcji fragmenty kodu nie połączone węzłami wykonują się równolegle. Zasada ta dotyczy także pętli. Na Rys. 4 przedstawiono dwie pętle działające równolegle. Pętla A realizuje zadania oznaczone jako a, wykonuje kolejne iteracje co ok. 200ms. Pętla B realizuje zadania oznaczone jako b, wykonuje kolejne iteracje co wielokrotność wartości 150 czasu systemowego pobieranego jako polecenie Tick (**Tick Count (ms)**). Obydwie pętle są zatrzymywane niezależnie przez dwa różne przyciski **stop 2** i **stop 3**. Zamknięcie jednej z nich nie powoduje zamknięcia całej aplikacji. Dopiero zamknięcie obydwu pętli powoduje zakończenie całego programu. Po zatrzymaniu jednej pętli nie ma również możliwości ponownego jej uruchomienia bez restartu aplikacji.



1. Kod realizujący działanie dwóch niezależnych pętli równolegle.

Bardziej szczegółowe informacje dotyczące pętli While oraz struktur programistycznych znajdują się w dalszych instrukcjach.

1. **LABORATORYJNE STANOWISKO BADAWCZE**
   1. **Obiekt badany**

- Środowisko programistyczne LabVIEW,

* 1. **Urządzenia dodatkowe**

- brak,

* 1. **Oprogramowanie**
* LabVIEW 2014 lub nowszy,

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

Kolejne kroki do wykonania podczas zajęć:

- utworzyć nowy plik vi (File 🡪 New VI lub Ctrl+N)

* Aplikacja odczytująca dane z wirtualnej karty DAQ:

- umieścić pętlę **While**, wybrane opóźnienie oraz przycisk stop,

- znaleźć bibliotekę „*SubVI\_Virrtual.llb*” i dodać na diagram znajdujące się wewnątrz pliki: „*Konfiguracja\_virtual.vi*”, „*Pomiar\_virtual.vi*”, „*Zwolnienie\_virtual.vi*”, dodanie pliku można zrealizować za pomocą **PPM** 🡪 **Select a VI…** 🡪 ***wyszukanie pliku w kreatorze***,

- umieścić pliki odpowiednio jako węzeł konfiguracyjny, węzeł realizujący zadanie oraz węzeł zwalniający zasoby,

- sprawdzić czy program zamknął się bez błędów umieszczając odpowiednio „*Simple error handler*” oraz stałą *type of dialog*: *OK message + warnings* (**PPM** na wejściu type of dialog 🡪 **Create** 🡪 **Constant**),

- na panelu użytkownika umieścić Waveform Chart,

- na diagramie połączyć wejście Waveform Chart z wyjściem out węzła „*Pomiar\_virtual.vi*”,

- uruchomić aplikację (RUN),

- sprawdzić działanie aplikacji (Highlight Execution),

- zapisać aplikację jako „*Prosta Aplikacja Pomiarowa.vi*”,

- zamknąć aplikację.

* Aplikacja zapisująca i odczytująca dane z wirtualnej karty DAQ:

- otworzyć aplikację „*Prosta Aplikacja Pomiarowa.vi*”,

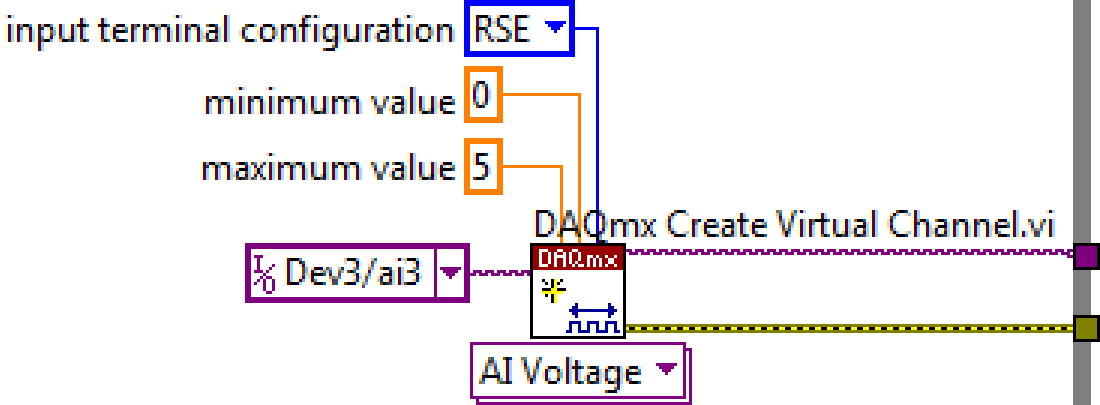
- zapisać plik: **Save As…** 🡪 **Substitute copy for original** 🡪 **Continue... 🡪** “*Program pomiarowy 2.vi*”,

- zastąpić wirtualne pliki obsługą rzeczywistej karty DAQ:

- z zakładki PPM 🡪 Measurement I/O 🡪 NI-DAQmx wybrać i odpowiednio połączyć następujące węzły: „*DAQmx Create Virtual Channel.vi*”, „*DAQmx Read.vi*”, „*DAQmx Clear Task.vi*”,

- zastąpić dotychczasowe węzły zachowując funkcjonalność,

- węzeł „*DAQmx Create Virtual Channel.vi*” skonfigurować (stałe wartości) w następujący sposób: input terminal configuration: = RSE, minimum value = 0, maximum value = 5, physical channels – kanał 3 podłączonej karty np. *Dev3/ai3*,



- zmienić nazwę pętli na „Akwizycja”

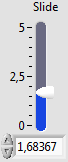
- sprawdzić działanie aplikacji,

- wprowadzić drugą pętlę While powyżej/poniżej pierwszej,

- uzupełnić przycisk stop oraz opóźnienie,

- umieścić na panelu użytkownika suwak pionowy **PPM** 🡪 **Numeric Control** 🡪 **Vertical** **Pointer Slide**,

- skonfigurować jak na rysunku poniżej (zakres wartości: 0 - 5):



- dodać i odpowiednio podłączyć węzły obsługujące generację sygnału na karcie DAQ:

„*DAQmx Create Virtual Channel.vi*”, „*DAQmx Write.vi*”, „*DAQmx Clear Task.vi*”, (UWAGA! Create Virtual Channel to ten sam węzeł co poprzednio, różnica polega na konfiguracji poniżej – należy wybrać **Analog Output** 🡪 **Voltage**),

- węzeł „*DAQmx Create Virtual Channel.vi*” skonfigurować (stałe wartości) w następujący sposób: input terminal configuration: = RSE, minimum value = 0, maximum value = 5, physical channels – kanał 3 podłączonej karty np. *Dev3/ao0*,

- uruchomić aplikację, **UWAGA! Jeżeli program współpracuje z kartą DAQ oraz generatorem sygnału należy zewrzeć trzecie wyprowadzenie goldpinów I3 z O0. W innym przypadku należy zewrzeć odpowiednie wyprowadzenia sygnałowe.**

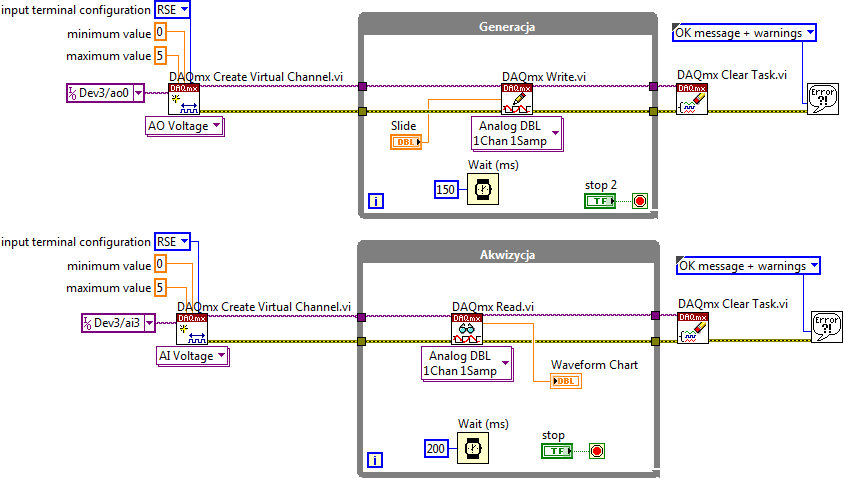
- sprawdzić działanie (Highlight Execution),

- zapisać plik,

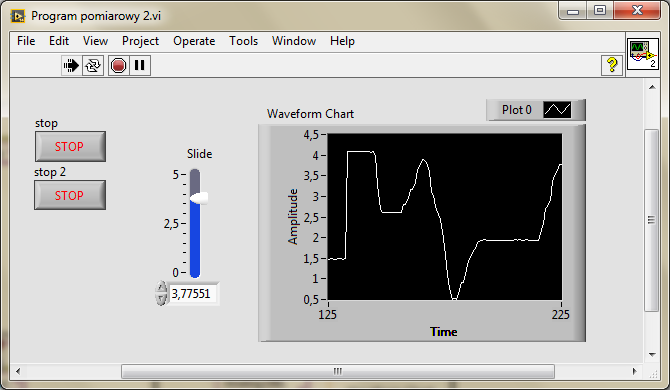
- zamknąć środowisko LabVIEW.

1. **PRZYKŁAD REALIZACJI ZADANIA – „*Aplikacja Pomiarowa 2.vi*”**

Druga część zadania została przedstawiona na Rys. 5. Program został podzielony na dwie niezależne pętle programowe, w których jednej (górnej) generowany jest sygnał wyjściowy karty, w drugiej dokonuje się akwizycja sygnału na wybranym wejściu. Jeżeli obydwa wyprowadzenia karty są ze sobą zwarte przebieg w *Waveform Chart* prezentuje wartości ustawiane suwakiem *Slide* - Rys. 6.



1. Diagram realizujący funkcjonalność drugiej części ćwiczenia.



1. Widok panelu użytkownika programu z pierwszej części ćwiczenia.
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. Jaka jest różnica między działaniem opóźnień Wait (ms) oraz Wait Until Next ms Multiple.
3. Co to jest Execution Timing i jak się go dobiera.
4. Cechy zastosowania pętli równoległych w LabVIEW.
5. Czy poniższe przykładowe kody (A i B) będą działały w sposób równoległy (podkreśl właściwą odpowiedź)?

|  |  |
| --- | --- |
|  |  |
| TAK/NIE | TAK/NIE |

# 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. Nota katalogowa LM35

Opracowanie: Marek Kciuk

# ZADANIA DO REALIZACJI PRZED PRZYSTĄPIENIEM DO ĆWICZENIA

- zrealizować wszystkie zadania z instrukcji nr 2.

- zapoznać się z właściwościami (PPM 🡪 Properties) kontrolek, zwłaszcza z zakładkami:

**Appearance**, **Data type**, **Data Entry**, **Display format**, **Documentation**,

w przypadku wskaźników typu graph dodatkowo:

**Plots**, **Scales**, **Cursos**.

- Jakie możliwości edycji mają właściwości?

Do niektórych właściwości jest szybszy dostęp bezpośrednio po naciśnięciu PPM (np. **PPM** 🡪 **Representation**),

- przerobić plik f\_liniowa.vi (z pierwszej części ćwiczenia) zgodnie z wszystkimi wytycznymi dla plików subVI

- na podstawie przerobionego pliku f\_liniowa.vi przygotować pliki konwertujące temperaturę między skalami we wszystkich możliwych kierunkach

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

-

# ZAŁĄCZNIKI

## Wait (ms) vs. Wait Until Next ms Multiple

Węzeł Wait (ms):

Realizuje opóźnienie programu o zadany czas od momentu wywołania tego węzła. Zastosowanie w pętli While spowoduje, że pętla po wykonaniu wszystkich zadań w niej zawartych będzie czekała do końca czasu podanego na wejście tego węzła. W przypadku, jeśli zadania w pętli wykonują się dłużej niż czas wpisany do węzła, pętla natychmiast po wykonaniu tych zadań przedzie do następnej iteracji nie wprowadzając dodatkowego opóźnienia. Czas przejścia między kolejnymi iteracjami dodaje się do ogólnego czasu działania pętli.

Węzeł Wait Until Next ms Multiple:

Realizuje opóźnienie programu do zadanego punktu czasu określonego jako Tick zegara systemowego podanego w ms. Zastosowanie w pętli While spowoduje, że kolejne iteracje będą kończyły się co wielokrotność wartości podanej na wejście tego węzła. Spowoduje to synchronizację działania pętli z czasem systemowym. W przypadku, jeżeli zadania do realizacji będą wymagały dłuższego czasu …(SPRAWDZIĆ)