|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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. | **Struktura wyboru - Case** | 3 |
| 2.2. | **Praca w projekcie** | 5 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 6 |
| 3.1. | **Obiekt badany** | 6 |
| 3.2. | **Urządzenia dodatkowe** | 6 |
| 3.3. | **Oprogramowanie** | 7 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 7 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA – Plik „*temp limits.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. |  |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. |  |  |
| 9. |  |  |
| 10. |  |  |
| 11. |  |  |
| 12. |  |  |
| 13. |  |  |

**CELE ĆWICZENIA**

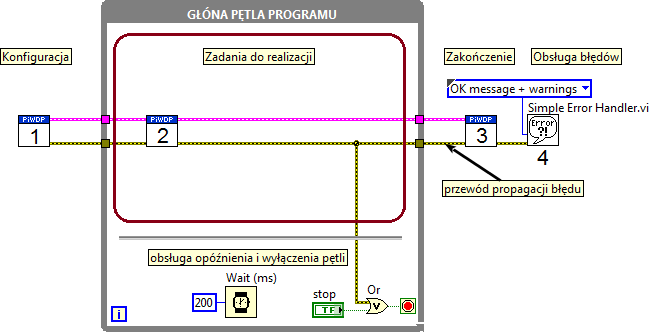
Celem ćwiczenia jest zapoznanie się z podstawową obsługą środowiska LabVIEW:

* 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.

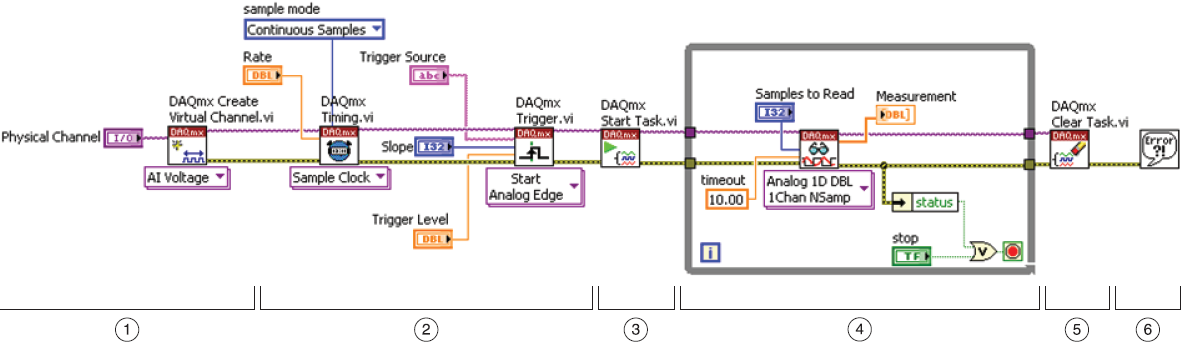
Uruchomiony program powinien działać aż do jego zatrzymania przez użytkownika (względnie do wystąpienia błędu, którego skutkiem ma być wymuszenie zamknięcia aplikacji). Podstawowym rozwiązaniem w środowisku LabVIEW jest tworzenie aplikacji w strukturze pętli **While**. Pętla **While** wykonuje zawarty 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 kończy się połączenia otwarte w części konfiguracyjnej,
5. – obsługa błędów – w najprostszej postaci wysyła użytkownikowi komunikat o ewentualnym wystąpieniu błędu.

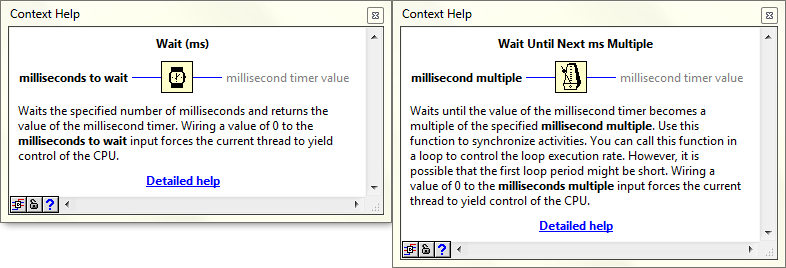
Pamiętać należy, żeby w tej strukturze stosować opóźnienie działania aplikacji (Execution Timing).

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.



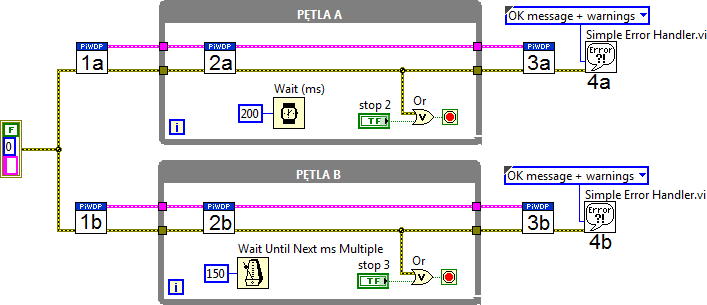
1. Przykładowa struktura Simple Measurement Structure do akwizycji danych z karty DAQ.
   1. **Opóźnienie programowe – execution timing**

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 najprostszą 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:

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

* Aplikacja odczytująca dane z karty DAQ:

- na podstawie tabeli konwersji temperatury przygotowanej po 1 ćwiczeniu przygotować plik subVI wykonujący konwersję wartości temperatur między wszystkimi skalami, uwzględnić również brak konwersji (jako wejście selektora wybrać typ **Enum**),

- ustawić domyślną konwersję ze skali Celsjusza na skalę Kelwina,

- sprawdzić działanie pliku,

- zapisać plik pod nazwą „*temp konwersja.vi*”,

- zamknąć plik,

* Praca w projekcie

- utworzyć nowy pusty projekt,

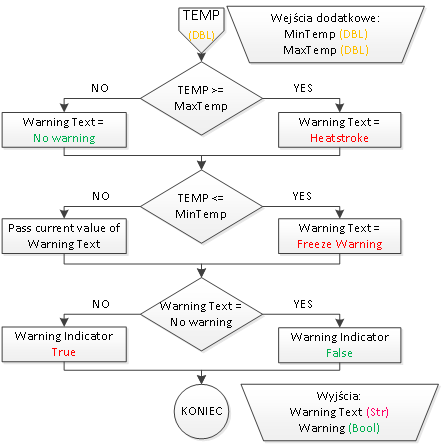
- zapisać projekt w nowym katalogu „*Stacja pogodowa/System pomiarowy DAQ.lvproj*”,

- dodać nowy wirtualny katalog „*subVI*”,

- umieścić pliki z biblioteki „*SubVI\_Virtual.llb*” w wirtualnym katalogu „*subVI*” (drag&drop),

- w katalogu „*subVI*” umieścić plik „*temp konwersja.vi*”

- przygotować plik realizujący decyzyjność działania aplikacji zgodnie z podanym niżej schematem blokowym



1. Schemat blokowy pliku „temp limits.vi”.

- dodać notatkę „***#todo*** *przekształcić w subVI*”,

- dodać notatkę „***#todo*** *zabezpieczyć przed limit min > limit max*”

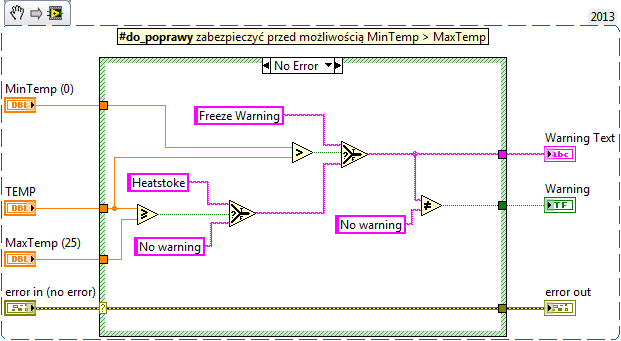
- sprawdzić działanie pliku,

- zapisać plik pod nazwą „*temp limits.vi*”,

- umieścić plik w katalogu „*subVI*”,

1. **PRZYKŁAD REALIZACJI ZADANIA – Plik „*temp limits.vi*”**

Przykładowa realizacja subVI „*temp limits.vi*” z zastosowaniem węzłów **selektor** została przedstawiona na Rys. 10.



1. Przykładowa realizacja pliku subVI - diagram.



1. Przykładowa realizacja pliku subVI - ikona i panel połączeń.

Kontrolki **Min Temp** oraz **Max Temp** ustawiono jako opcjonalne (**Optional**), wartości domyślne podano w nawiasach, kontrolka **TEMP** ustawiona jako wymagana (**Required**). Dla przypadku Error wskaźniki przyjmują następujące wartości: **Warning** – „***True***”, **Warning Text** – „***Error in***”.

1. **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. Jaki będzie stan operacji: „Stan 1” + „Akcja 1” dla zmiennej typu enum na Rys. 5?
3. Wymienić zalety stosowania projektu w LabVIEW.

# 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

- przygotować tabelę przeliczników między skalami Kelwina, Celsjusza i Fahrenheita

|  |  |  |  |
| --- | --- | --- | --- |
| Współczynniki (a,b) | Kelwin, K | Celsjusz, oC | Fahrenheit, oF |
| Kelwin, K | 1, 0 |  |  |
| Celsjusz, oC |  | 1, 0 |  |
| Fahrenheit, oF |  |  | 1, 0 |

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

- uzupełnić plik „*temp limits.vi*” zgodnie z konwencją plików subVI

- uzupełnić plik „*temp limits.vi*” aby zabezpieczyć przed odwrotnym podpięciem wejść ograniczających zakres dopuszczalnej temperatury **Temp min** oraz **Temp max**.

- przygotować plik polimorficzny sterowany za pomocą selektora pod ikoną umożliwiający wybór skalowania wartości temperatury (patrz załaczniki).

# ZAŁĄCZNIKI

## Linkowanie tuneli wejściowych z wyjściowymi

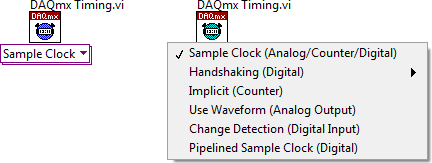
Jak napisano w rozdziale 2.1. opcja wprowadzenia domyślnej wartości do tunelu wyjściowego (**PPM** 🡪 **Use Default If Unwired**) nie jest zalecana. Często znacznie wygodniejsza jest opcja linkowania wybranego wejścia z wyjściem. Ma to zastosowanie w przypadku struktur wyboru z wieloma wewnętrznymi diagramami. W takim przypadku zdarza się, że często nie korzystamy z wybranych sygnałów i musimy ręcznie łączyć wejścia z wyjściami (korzystamy z nich tylko w kilku wybranych przypadkach). Połączenia wyjścia z wejściem (tego samego typu!) realizuje się **PPM** (na tunelu wyjściowym) 🡪 **Linked Input Tunel** 🡪 **Create & Wire Unwired Cases**.

|  |  |
| --- | --- |
| a) | b) |
|  |  |

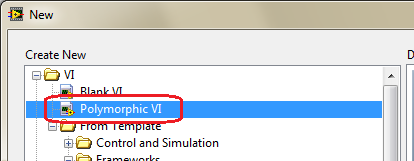
Operację linkowania przedstawia rysunek a), natomiast efekt połączenia prezentowany jest w postaci małych białych trójkątów na tunelach wejściowym i wyjściowym widoczny na rysunku b).

## Pliki polimorficzne

Pliki polimorficzne składają się z wielu pojedynczych plików składowych. Wybór właściwego pliku może odbywać się za pomocą wartości wprowadzanych do terminali wejściowych lub ręcznie za pomocą selektora umieszczonego pod ikoną. Przykładem pliku polimorficznego z opcją ręcznego wyboru jest *DAQmx Timing.vi*.



W celu przygotowania pliku polimorficznego należy przygotować wszystkie pliki składowe, a następnie utworzyć nowy plik **New…**:



A następnie uzupełnić kreatora plikami składowymi oraz wybrać odpowiednie opcje (więcej informacji w dokumentacji LabVIEW oraz <https://zone.ni.com/reference/en-XX/help/371361H-01/lvhowto/using_polymorphic_vis/> ).

