|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP2 | |  |
| Tytuł ćwiczenia: | | **Podstawy obsługi środowiska  – praca w projekcie** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Struktura wyboru - Case** | 3 |
| 2.2. | **Praca w projekcie** | 5 |
| 2.3. |  | 6 |
| 2.4. |  | 8 |
| 2.5. |  | 9 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 9 |
| 3.1. | **Obiekt badany** | 9 |
| 3.2. | **Urządzenia dodatkowe** | 10 |
| 3.3. | **Oprogramowanie** | 10 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 10 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA – Przygotowanie symulowanej karty DAQ** | 14 |
| 6. | **RAPORT** | 14 |
| 7. | **PYTANIA** | 14 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Podstawowa wersja struktury Case z selektorem typu boolean. | 3 |
| 2. | Struktura Case z selektorem typu integer. | 3 |
| 3. | Przedziały zakresów liczbowych dla struktury Case z warunkami z Rys. 2. | 4 |
| 4. | Struktura Case z selektorem typu string | 4 |
| 5. | Struktura Case z selektorem typu enum. | 4 |
| 6. | Węzeł Select – wygląd oraz opis w pomocy kontekstowej. | 5 |
| 7. |  |  |
| 8. |  |  |
| 9. |  |  |

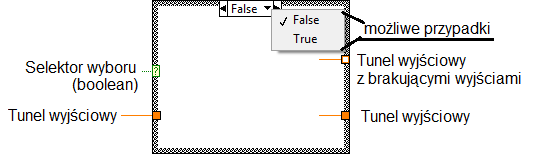
1. **CELE ĆWICZENIA**

Celem ćwiczenia jest zapoznanie się z podstawową obsługą środowiska LabVIEW:

* Poznanie właściwości i zastosowania struktury wyboru Case
* Praca w projekcie

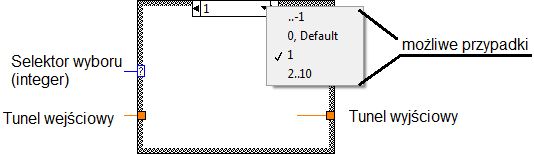
1. **WPROWADZENIE**
   1. **Struktura wyboru - Case**

Struktura wyboru **Case** służy do rozgałęzienia lub zróżnicowania wykonywanego kodu. Decyzja, który przypadek struktury ma zostać wykonany jest podejmowana na podstawie informacji wpływającej do selektora wyboru (**Case Selector**). Domyślnym typem danych dla selektora jest typ **boolean**, jednak może przyjmować także inne typy danych jak: **integer** (typ całkowity), **string**, **enum** tych typów umożliwia rozgałęzienie programu na więcej niż dwa sposoby. Na Rys. 1 przedstawiono strukturę **Case** z selektorem typu boolean oraz z tunelami wejściowymi i wyjściowymi. W przypadku tuneli wyjściowych WSZYSTKIE warunki muszą wpisywać dane. Na Rys. 1 przedstawiono przykład tunelu wyjściowego z wypełnionymi (pomarańczowy) oraz niewypełnionymi (biały z pomarańczową obwódką) warunkami.



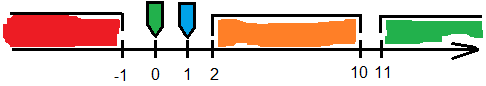
1. Podstawowa wersja struktury Case z selektorem typu boolean.

Przypadek użycia struktury Case z selektorem typu integer został przedstawiony na Rys. 2. Dla liczb całkowitych można tworzyć przypadki dla konkretnych wartości (0, 1), zamkniętych przedziałów wartości [2..10], oraz otwartych przedziałów wartości (-∞..-1].



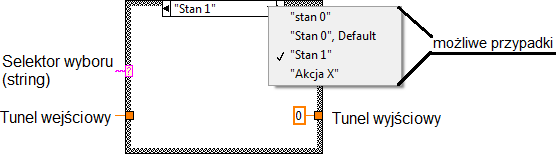
1. Struktura Case z selektorem typu integer.

Dla przypadków nieokreślonych wykonywany jest warunek oznaczony jako domyślny (**Default**). Przypadki takie na Rys. 2 to zakres liczb powyżej 10. Zakresy liczbowe dla poszczególnych przypadków zostały przedstawione na



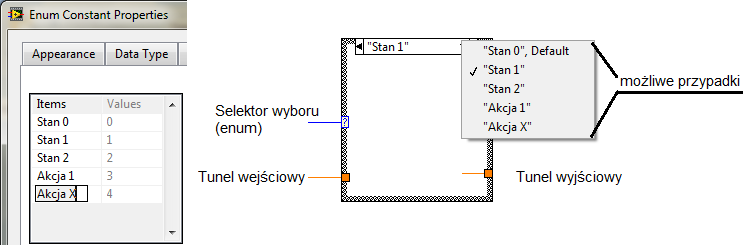
1. Przedziały zakresów liczbowych dla struktury Case z warunkami z Rys. 2.

Przypadek użycia selektora tekstowego (**string**) został przedstawiony na Rys. 4. LabVIEW rozróżnia wielkość liter więc przypadki „*stan 0*” oraz „*Stan 0*” to dwa różne stany. Należy zwrócić szczególną uwagę, jeżeli teksty podawane na selektor nie są zunifikowane (np. podawane przez użytkownika).



1. Struktura Case z selektorem typu string.

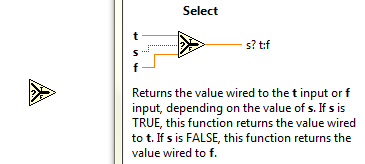
Przypadek użycia selektora **Enum**. Zmienna tego typu przypisuje kolejnym wartościom typu integer nazwy, jednak środowisko LabVIEW obsługuje go jako tryb numeryczny i umożliwia wykonywanie operacji arytmetycznych. Na Rys. 5 przedstawiono strukturę **Case** z selektorem typu enum, którego wartości także znajdują się na ryunku.



1. Struktura Case z selektorem typu enum.

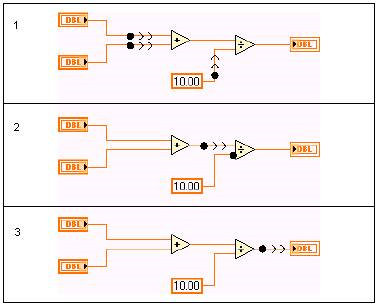
Tunele wyjściowe muszą mieć wprowadzone dane we wszystkich przypadkach. Jednak istnieje opcja na tunelu wyjściowym PPM 🡪 **Use Default If Unwired**, która wypełnia wartościami domyślnymi przypadki, nieobsłużone. W przypadku zmiennych liczbowych wartością domyślną jest 0. Nie jest to opcja zalecana, powoduje zmniejszenie czytelności kodu. Może także powodować niespodziewane błędy, domyślną wartością dla referencji do pliku jest pusta referencja.

W przypadku prostych wyborów dwustanowych warto zamiast struktury **Case** stosować węzeł wyboru **Select** (Rys. 6). Znajduje się on w zakładce  
Programming 🡪 Comparison.



1. Węzeł Select – wygląd oraz opis w pomocy kontekstowej.
   1. **Praca w projekcie**

Bardzo istotną sprawą jest zrozumienie mechanizmu kolejności wykonywania kodu w LabVIEW. W tekstowych językach strukturalnych kolejność wykonywania kodu jest determinowana kolejnymi liniami od góry do dołu. W LabVIEW kolejność wykonywana kodu jest określana przepływem danych **DATAFLOW**. Elementami wprowadzającymi dane są terminale wejściowe – są to kontrolki, stałe lub węzły nie posiadające tuneli wejściowych (np. węzeł Random), dalej dane przechodzą przez kolejne węzły zgodnie ze strukturą połączeń, ostatnimi elementami są wskaźniki prezentujące wynik działania kodu. Przykład działania dataflow z rozbiciem na kolejne kroki wykonywanego kodu został przedstawiony na Rys. 3.



1. Dataflow – kolejność wykonywania kodu w LabVIEW.

Kolejność wykonywania programu w LabVIEW jest określona następującymi zasadami:

* Przepływ danych odbywa się od lewej do prawej strony (kontrolki umieszczamy na lewo a indykatory na prawo kodu),
* Przepływ danych odbywa się od kontrolek do indykatorów,
* Węzeł wykona swoje zadanie jeżeli otrzyma WSZYSTKIE wymagane dane wejściowe,
* Wszystkie węzły, które otrzymują dane w tym samym kroku wykonują się „równolegle”,
* Niezależne kody programu wykonują się „równolegle”.

Kolejność wykonywania programu w uproszczony sposób prezentuje opcja **Highlight execution**, ikona z paska okna diagramu została przedstawiona na Rys. 4



1. Ikony highlight execution – opcja nieaktywna oraz aktywna.
   1. **Tworzenie podprogramów – SubVI**

Bardzo istotna w LabVIEW jest modułowość pisanego kodu. Jest ona nierozłączna z dobrymi praktykami programowania w LabVIEW. Do dobrych praktyk zaliczyć można: czytelność i skalowalność. Modułowość kodu bazuje na zamykaniu fragmentów kodu w podprogramach celem późniejszego ich zastosowania w różnych miejscach aplikacji, w LabVIEW podprogram nazywa się **subVI**. Każdy plik vi może być użyty w innym pliku jako subVI. Pamiętać należy, że przekazywanie danych do i z subVI odbywa się za pomocą panelu połączeń (patrz instrukcja do ćwiczenia 1). Tworzenie podprogramów można wykonać na trzy sposoby:

* Tworzyć nowy czysty plik vi i zaimplementować w nim strukturę subVI,
* Utworzyć nowy subVI z szablonu: SubVI with error handling (),
* Gotowy fragment kodu zaznaczyć i zamienić na subVI: Edit 🡪 Create SubVI,

Podstawowa oraz inne dostępne struktury panelu połączeń zostały przedstawione na Rys. 5. Pamiętać należy, że wszystkie komponenty pulpitu, które mają przenosić dane do nadrzędnego vi musza być podpięte do panelu terminali.

|  |  |
| --- | --- |
| a) | b) |
| 1. Struktury panelu połączeń (a) domyślna, (b) inne dostępne. | |

Wewnątrz struktury połączeń wybrane terminale zarezerwowane są dla sygnałów błędu oraz referencji – Rys. 6.



1. Domyślne terminale dla klastra błędu oraz referencji.

Terminale wejściowe posiadają różne priorytety podłączenia. Mogą przyjmować następujące:

* Recommended – domyślny priorytet, oznaczony cienką czarną linią,
* Required – wejście wymagane, jeżeli do tego terminalu nie jest podpięty przewód kompilator zgłasza błąd kompilacji,
* Optional – wejście opcjonalne – oznaczone kolorem szarym.

Typ wejścia zmienia się klikając:

PPM 🡪 This Connection Is 🡪 Required/Recommended/Optional

Ponadto dla wszystkich wejść można ustalić wartości domyślne dla danej kontrolki:

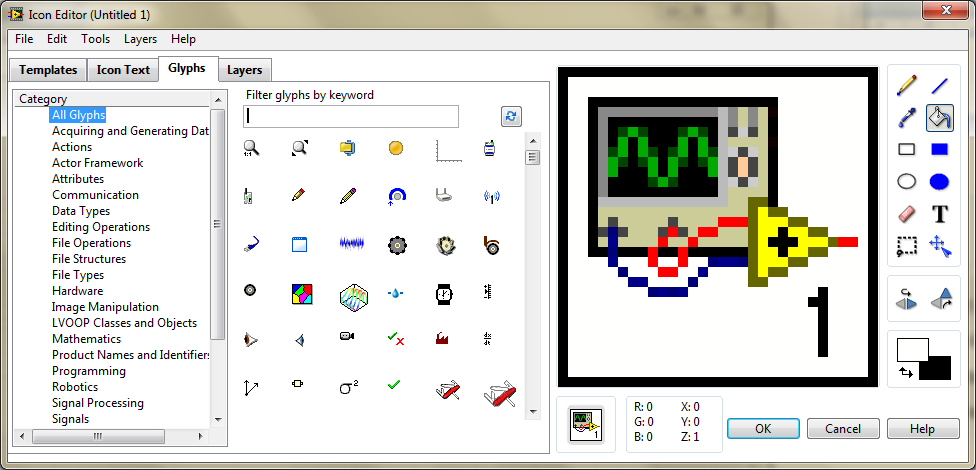
(na kontrolce wpisać wartość, która ma być domyślna a następnie)

PPM 🡪 Data Operations 🡪 Make Current Value Default

lub domyślnie dla wszystkich kontrolek na panelu:

Edit 🡪 Make Current Values Default

Każdy subVI w diagramie jest identyfikowany przede wszystkim za pomocą ikony, dlatego do dobrych praktyk należy tworząc plik, zwłaszcza taki, który intencjonalnie ma być jako subvi utworzyć ikonę. Ikona jest integralną częścią pliku vi. Ikonę modyfikuje się za pomocą edytora ikon (**icon editor**) – jest to prosty program graficzny zawierający gotowe szablony komponentów, wzorców oraz umożliwiający pracę z warstwami – Rys. 7.



1. Okno edytora ikon.

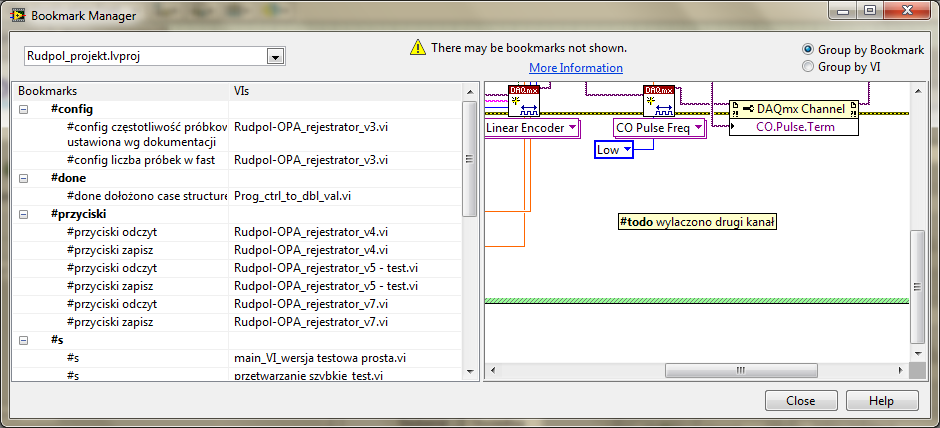
Najlepszą praktykę tworzenia ikon można zaobserwować w gotowych ikonach środowiska LabVIEW. Jednak wykonanie takiej ikony zajmuje relatywnie sporo czasu. Na Rys. 8 przedstawiono kilka przykładów ikon.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 1. Przykładowe ikony   (na pierwszym miejscu ikona domyślna) | | | | | | |

* 1. **Dokumentacja kodu**

Dokumentacja kodu jest kolejnym bardzo ważnym zagadnieniem, dbanie o jakość kodu w znaczy sposób przyspieszy znajdowanie potencjalnych błędów oraz rozwój aplikacji. Do zestawu działań związanych z poprawną dokumentacją kodu zaliczyć można:

* Nadawanie znaczących nazwy plików VI - nazwy powinny być w miarę krótkie ale dokładnie opisujące zadanie, które realizują,
* Tworzenie dedykowanej ikony w istotny sposób ułatwia wzrokową analizą kodu,
* Nadawanie znaczącej nazwy terminali we/wy dla subVI, dodatkowo w nazwie można zawrzeć wartość domyślną, która będzie przyjęta kiedy terminal nie jest podłączony, przykład nazwy terminalu: „Frequency (500Hz)”,
* Zadawanie wartości domyślnej dla kontrolek, w przypadku powyższego terminala, kontrolka podpięta do niego będzie miała wartość domyślną 500,
* Wypełnianie zakładki **Documentation** we właściwościach pliku File 🡪 VI properties
* Wypełnianie zakładki **Documentation** we właściwościach kontrolek PPM 🡪 Properties
* W celu umieszczenia komentarza wewnątrz diagramu lub panelu, nie związanego z żadnym komponentem można użyć wolne etykiety (**free labels**) wystarczy podwójne kliknięcie w pustą strefę,
* Dodatkowe etykiety można tworzyć na przewodach PPM 🡪 Visible Items 🡪 Label
* Rozszerzeniem funkcjonalności wolnych etykiet w LabVIEW 2013 są zakładki, tworząc etykietę zaczynając od znaku „#” pierwszy wyraz jest pogrubiony tworząc nazwę etykiety, listę zakładek można wyświetlić w managerze zakładek View 🡪 Bookmark Manager, zakładki tworzyć można wyłącznie w oknie diagramu, na rysunku RYS przedstawiono okno managera zakładek.



1. Okno managera zakładek z doinstalowanym dodatkiem prezentującym fragment kodu

* 1. **Debuggowanie i wyszukiwanie błędów w programie**

W LabVIEW istnieje szereg narzędzi umożliwiających sprawdzanie programu. Zaliczyć do nich można:

* **Highlight execution ** – opcja ta prezentuje kolejność wykonywania kodu, praca aplikacji jest zwalniana, żeby programista mógł śledzić kolejne wykonywane kroki i analizować przepływ danych, jest to najprostsza metoda debuggowania, nie sprawdza się dla fragmentów kodu, których wykonanie jest uzależnione czasowo np. wymiana danych z urządzeniem zewnętrznym,
* **Probe** – sonda śledząca wartość danych na przewodzie – pracuje w czasie rzeczywistym wyświetlając wartości, przy bardzo szybkiej pracy programista nie ma szans na odczytanie wszystkich wartości, jednak sonda pamięta ostatnie wartości po zatrzymaniu programu, kolejne sondy na przewodach oznaczone są numerami, numery te odpowiadają numerom w oknie **Probe Watch Window**,
* **Breakpoint** – punkty zatrzymania aplikacji – po zadziałaniu przerwania aplikacja przechodzi w stan zatrzymania Pause, przerwania sygnalizowane są na przewodach czerwonymi kółkami,
* **Single stepping**  – praca krokowa w dwóch opcjach **Step Into** oraz **Step Over** umożliwia analizę kodu z możliwością wejścia do subvi (step into) lub wykonanie kodu subvi bez wchodzenia do środka (step over), praca krokowa współpracuje break pointami, możliwe jest zatrzymanie kodu przed interesującym fragmentem za pomocą przerwania, następnie przeanalizowanie dalszej części pracą krokową.

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:

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

* Zapoznanie się z metodami wyszukiwania węzłów diagramu oraz komponentów panelu:

- zidentyfikować okno panelu oraz okno diagramu,

- przełączyć kilka razy między oknami za pomocą odpowiedniego skrótu klawiaturowego (patrz poprzednia instrukcja) (………………… można wpisać skrót)

- za pomocą palety funkcji (PPM) na panelu umieścić dowolną kontrolkę typu string

- skopiować kontrolkę używając Ctrl+LPM

- zmienić typ skopiowanego terminalu z kontrolki na terminal

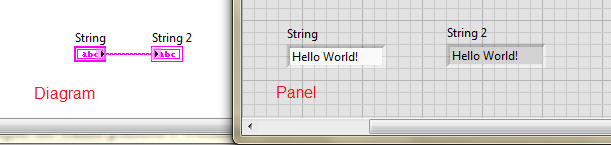
PPM 🡪 Change to Indicator

- sprawdzić reprezentacje terminali w oknie diagramu

- połączyć za pomocą przewodu kontrolkę ze wskaźnikiem

- wrócić do Panelu i w kontrolce wpisać wybrany tekst np. „**Hello World!**”

- uruchomić aplikację (Ctrl+R)



- przejść do okna panelu i uruchomić **Quick-Drop** (Ctrl+Space)

- wyszukać kontrolkę typu Numeric i umieścić na panelu

- zmienić nazwę kontrolki na „a”

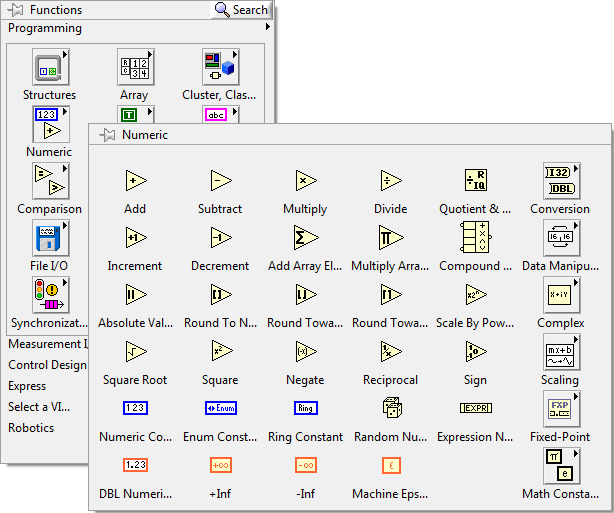
- wybraną metodą (kopiowanie, Quick-Drop, paleta kontrolek) umieścić dodatkowe 3 kontrolki typu Numeric

- pozmieniać nazwy: „a”, „b”, „x”, „y”,

- kontrolkę „y” zamienić a wskaźnik,

- przejść do okna diagramu

- w palecie funkcji znaleźć funkcje matematyczne



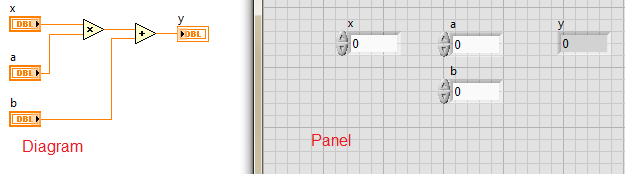
- sprawdzić i zapamiętać nazwę węzła mnożenia

- wybrać i umieścić na diagramie węzeł dodawania

- za pomocą Quick drop dodać węzeł mnożenia

- połączyć elementy diagramu aby uzyskać funkcję liniową „”

- uporządkować obiekty w obydwu oknach (np. jak na rysunku)



- zadać wartości w kontrolkach i uruchomić aplikację

- zapisać plik pod nazwą „f\_liniowa.vi”

* Zapoznanie się z ideą wykonywania kodu graficznego – „dataflow”:

- kilkakrotnie uruchomić aplikację z aktywną opcją highlight execution 

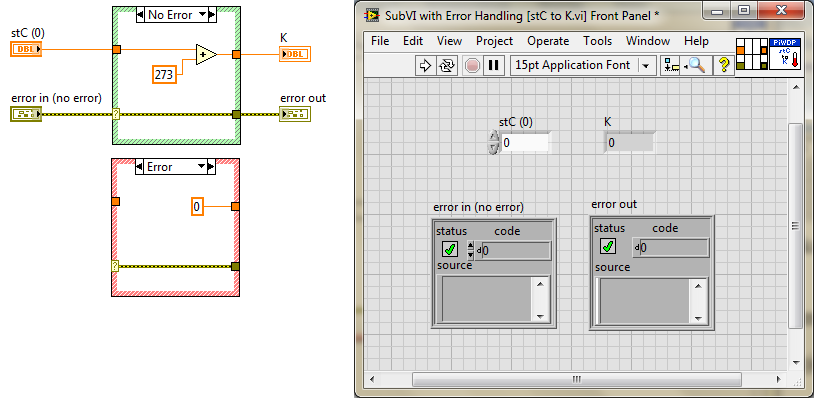
- zamknąć plik

* Tworzenie kodu subVI korzystając z szablonu, zgodnie z dobrymi praktykami:

- otworzyć nowy plik z szablonu SubVI with error handling

- zapisać plik pod nazwą „stC to K.vi”

- napisać program przeliczający temperaturę w stopniach Celcjusza na Kelwiny (zgodnie z Rys. 10)



1. Program przeliczający temperaturę między skalami Celcjusza a Kewina

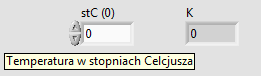
Realizując program pamiętać należy o układzie komponentów pulpitu, właściwych nazwach kontrolek i wskaźników, ikonie i panelu połączeń,

- nadać wejściu stC priorytet „wymagane” (**Required**)

- uzupełnić dokumentację we właściwościach pliku File 🡪 VI Properties 🡪 Documentation

- w zakładce **Window Apperance** zmienić pole **Window Title** na „stC to K”

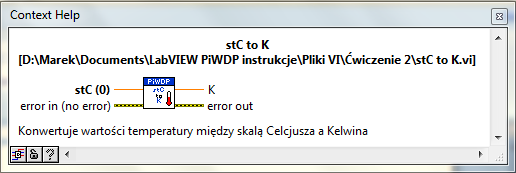
- we właściwości Documentation kontrolki stC uzupełnić pole Tip strip (np. tekstem „Temperatura w stopniach Celcjusza”) – efekt przedstawiony poniżej, po najechaniu kursorem na kontrolkę



- we właściwości **Documentation** wskaźnika K uzupełnić pole Tip strip

- uruchomić pomoc kontekstową Ctrl+H

- podejrzeć opis pliku w pomocy kontekstowej najeżdżając na ikonę – powinna wyglądać jak na



1. Pomoc kontekstowa utworzonego pliku

- zapisać plik

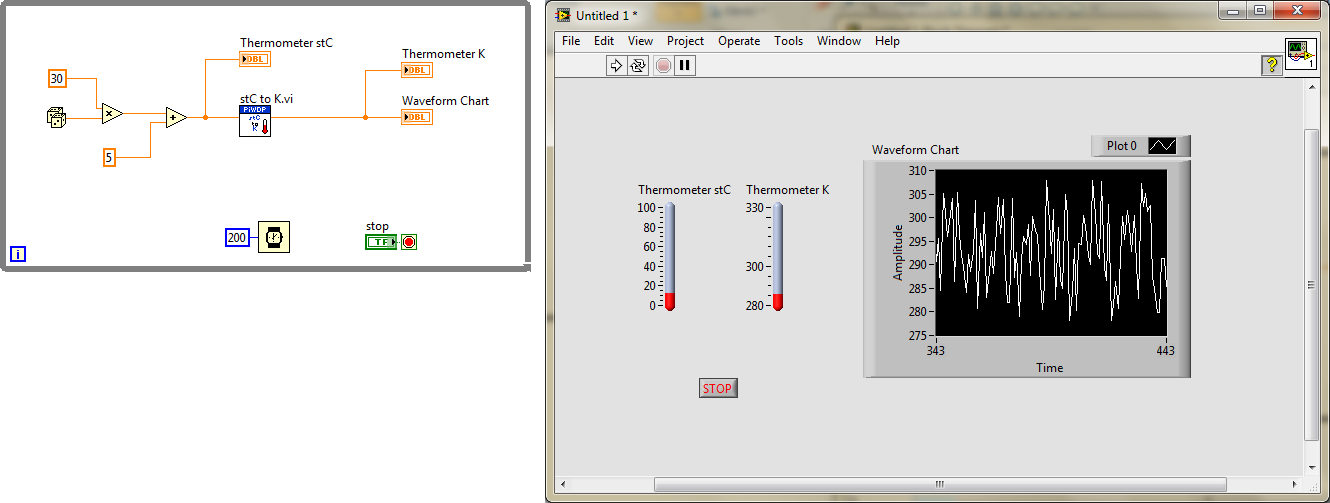
* Zapoznanie się z metodami debuggowania aplikacji i subVI,

- otworzyć nowy plik

- umieścić w diagramie plik „stC to K.vi”

- sprawdzić listę błędów 

- uzupełnić plik jak na



1. Aplikacja do debuggowania

- uruchomić aplikację

- umieścić na wybranych przewodach sondy PPM 🡪 Probe

- uruchomić aplikację

- zamknąć okno sondy

- usunąć: strukturę while, przycisk stop oraz wait (ms)

- włączyć opcję „highlight executinon”

- kilkakrotnie uruchomić aplikację

- uruchomić i sprawdzić działanie aplikacji przyciskami Start Single Stepping (Step Into oraz Step Over) 

- zapisać plik pod nazwą „debugg app.vi”

-zamknąć środowisko LabVIEW

1. **PRZYKŁAD REALIZACJI ZADANIA – Przygotowanie symulowanej karty DAQ**

brak.

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?

# 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 Kewina Celcjusza i Farenchaita

|  |  |  |  |
| --- | --- | --- | --- |
| Współczynniki (a,b) | Kelwin, K | Celcjusz oC | Ferenchait, F |
| Kelwin, K | 1, 0 |  |  |
| Celcjusz oC |  | 1, 0 |  |
| Ferenchait, F |  |  | 1, 0 |

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

- 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

# ZAŁĄCZNIKI

## Linkowanie tuneli wejściowych z wyjściowymi