|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP02 | |  |
| Tytuł ćwiczenia: | | **Podstawy obsługi środowiska, tworzenia i debuggowania programów i podprogramów** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Rozpoczynanie pracy i pomoc w środowisku LabVIEW** | 3 |
| 2.2. | **Dataflow - idea programowania graficznego** | 5 |
| 2.3. | **Tworzenie podprogramów – SubVI** | 6 |
| 2.4. | **Dokumentacja kodu** | 8 |
| 2.5. | **Debuggowanie i wyszukiwanie błędów w programie** | 9 |
| 2.6. | **Paleta narzędzi** | 10 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 11 |
| 3.1. | **Obiekt badany** | 11 |
| 3.2. | **Urządzenia dodatkowe** | 11 |
| 3.3. | **Oprogramowanie** | 11 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 11 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA – sprawdzenie działania funkcji Highlight Execution** | 15 |
| 6. | **RAPORT** | 16 |
| 7. | **PYTANIA** | 16 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Widok okna wyboru szablonu z zaznaczonymi najważniejszymi strukturami. | 3 |
| 2. | Okno wyszukiwarki przykładów. | 4 |
| 3. | Dataflow – kolejność wykonywania kodu w LabVIEW. | 5 |
| 4. | Ikony highlight execution – opcja nieaktywna oraz aktywna. | 6 |
| 5. | Struktury panelu połączeń (a) domyślna, (b) inne dostępne. | 6 |
| 6. | Domyślne terminale dla klastra błędu oraz referencji. | 7 |
| 7. | Okno edytora ikon. | 8 |
| 8. | Przykładowe ikony. | 8 |
| 9. | Okno managera zakładek z doinstalowanym dodatkiem prezentującym fragment kodu. | 9 |
| 10. | Widok palety narzędzi z aktywną opcją automatycznego wyboru narzędzia. | 10 |
| 11. | Widok diagramu i panelu użytkownika pierwszego programu „*Hello World!*”. | 12 |
| 12. | Widok zakładki Numeric w palecie funkcji. | 12 |
| 13. | Widok diagramu i panelu użytkownika programu „*f\_liniowa.vi*”. | 13 |
| 14. | Program przeliczający temperaturę między skalami Celcjusza na Kewina. | 13 |
| 15. | Widok kontrolki z wypełnioną opcją TipStrip. | 14 |
| 16. | Pomoc kontekstowa utworzonego pliku. | 14 |
| 17. | Aplikacja do debuggowania. | 15 |
| 18. | Widok działania programu z włączoną opcją Highlight Execution. | 16 |

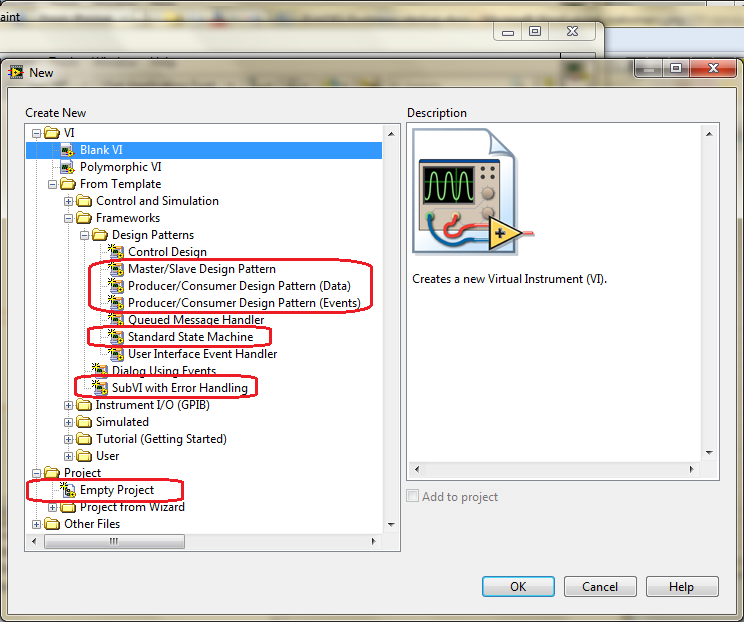
1. **CELE ĆWICZENIA**

Celem ćwiczenia jest zapoznanie się z podstawową obsługą środowiska LabVIEW:

* Zapoznanie się z metodami wyszukiwania węzłów diagramu oraz komponentów panelu,
* Zapoznanie się z funkcjami Panelu Czołowego oraz Diagramu kodu,
* Zapoznanie się z mechanizmami wspierającymi pracę programisty,
* Zapoznanie się z ideą wykonywania kodu graficznego – „dataflow”,
* Zapoznanie się z metodami debuggowania aplikacji i subVI,
* Zapoznanie się z podstawowymi technikami tworzenia i opisywania kodu.

1. **WPROWADZENIE**
   1. **Rozpoczynanie pracy i pomoc w środowisku LabVIEW**

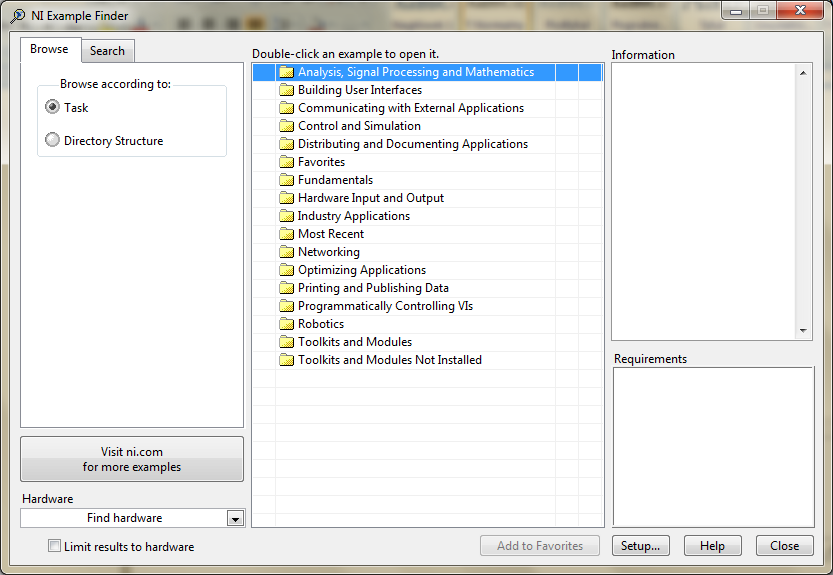
Pracę w środowisku LabVIEW można zacząć na kilka sposobów. Można utworzyć nowy pusty plik (**File** 🡪 **New VI**), jednak można w znaczy sposób przyspieszyć pracę korzystając z szablonów (**File** 🡪 **New…**). Szablony startowe w LabVIEW zawierają gotowe, najczęściej używane struktury programistyczne. Do najważniejszych z nich można zaliczyć: szablon projektu, strukturę podprogramu z funkcją obsługi błędów (*SubVI with error handling*), struktury wielopętlowe oraz strukturę maszyny stanów.



1. Widok okna wyboru szablonu z zaznaczonymi najważniejszymi strukturami.

Zaznaczone na Rys. 1 szablony będą omawiane w trakcie zajęć. Szablony są  
to gotowe fragmenty plików zawierające określoną strukturę programistyczną  
oraz najważniejszą funkcjonalność. Niektóre szablony mogą zostać uruchomione bez żadnych modyfikacji. Wszystkie szablony wyposażone są w dokładne opisy działania i instrukcje dla programisty.

Inną metodą przyspieszającą pracę w LabVIEW są przykłady. Wyszukać ich można  
za pomocą wyszukiwarki przykładów uruchamianej za pomocą Help 🡪 Find Examples… (Rys.2).



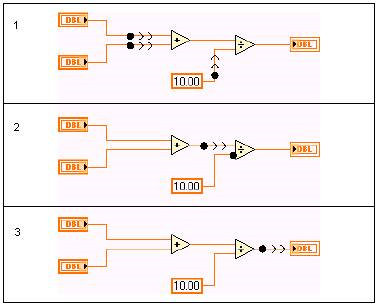
1. Okno wyszukiwarki przykładów.

W przeciwieństwie do szablonów wszystkie przykłady są gotowymi, uruchamialnymi aplikacjami LabVIEW. Zarówno wszystkie szablony jak i przykłady mogą być użyte bez dodatkowych opłat w pracy programisty w ramach licencji (***licencja akademicka nie umożliwia komercyjnego wykorzystania środowiska LabVIEW***).

Pomoc w środowisku LabVIEW podzielona jest na cztery podstawowe grupy:

* pomoc kontekstowa (Ctrl+H) – niewielkie okno pomocy prezentujące najważniejsze informacje dotyczące aktualnie zaznaczonego obiektu lub przewodu,
* okno pomocy (F1) – standardowe okno pomocy programu dla środowiska LabVIEW zawierające pełną pomoc dotyczącą zarówno środowiska jak i dokładny komponentów zarówno panelu jak i diagramu,
* List Errors – lista błędów zgłaszanych przez kompilator uniemożliwiająca uruchomienie aplikacji – jeżeli kompilator nie może uruchomić aplikacji z powodu błędów  
  w programie strzałka **RUN ** zmienia się w złamaną strzałkę **List Errors .** Kliknięcie powoduje otworzenie okna zawierającego listę plików, które nie mogą być otwarte, listę błędów i szczegóły zgłoszenia. Podwójne kliknięcie na wybranym błędzie spowoduje przeniesienie do odpowiedniego miejsca w kodzie, gdzie występuje błąd,
* Explain Error – pomoc środowiska LabVIEW dotycząca kodów błędów oraz ostrzeżeń wskazanych przez kompilator w trakcie działania aplikacji. Wybranie Help 🡪 Explain Error uruchamia okno, w którym można podać numer błędu zwracanego przez aplikację. Opis błędu, jego źródło lub prawdopodobne przyczyny zostanie wyświetlony w oknie.
  1. **Dataflow - idea programowania graficznego**

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. 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 muszą 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

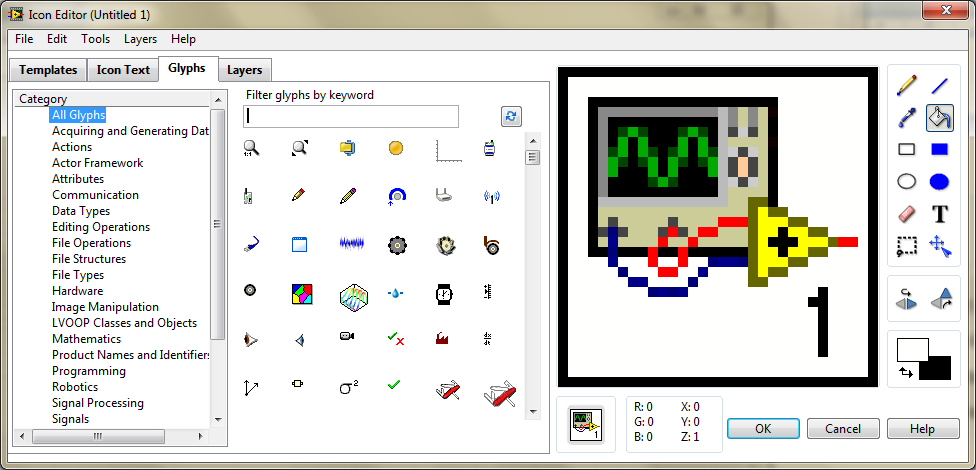
Ponadto dla wszystkich wejść można ustalić wartości domyślne dla danej kontrolki:

* na kontrolce wpisać wartość, która ma być wartością domyślną,
* kliknąć: 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 używany jako subvi, utworzyć jego unikalną 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.

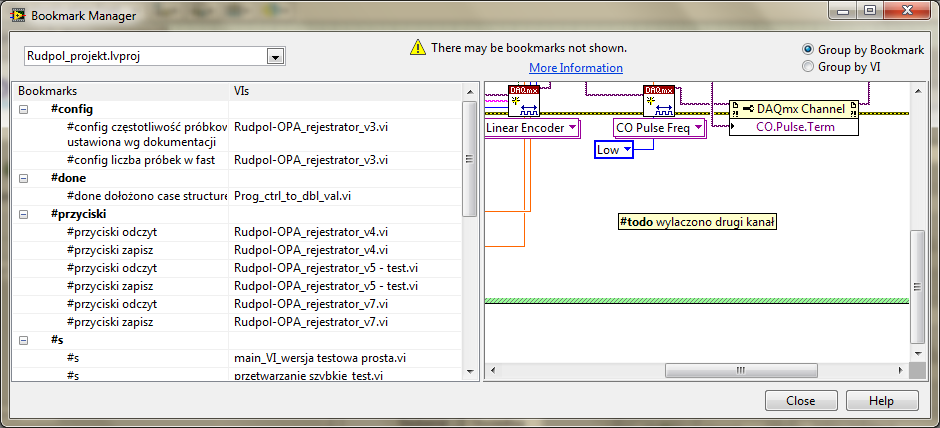
Najlepszą praktykę tworzenia ikon można zaobserwować w gotowych ikonach środowiska LabVIEW. Jednak wykonanie takiej ikony zajmuje relatywnie sporo czasu.  
Na Rys. 8 przedstawiono kilka przykładów ikon.

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

* 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 możliwie 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. Okno managera zakładek przedstawiono na Rys. 9.



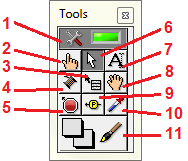
1. Okno managera zakładek z doinstalowanym dodatkiem prezentującym fragment kodu.

* 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 może odczytać 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. **Paleta narzędzi**

Podczas pracy ze środowiskiem LabVIEW kursor myszy automatycznie przełącza swoją funkcję w zależności nad jakim obiektem znajduje się. Czasem trzeba wymusić określone działanie kursora, inne niż domyślne. Funkcje kursora można ręcznie zmieniać za pomocą palety narzędzi: **View** 🡪 **Tools Palette**. Paleta narzędzi została przedstawiona na Rys. 10. Poniżej znajduje się opis wszystkich narzędzi.



1. Widok palety narzędzi z aktywną opcją automatycznego wyboru narzędzia.
2. **Automatic Tool Selection** – opcja aktywna powoduje automatyczny wybór narzędzia, aktualnie wybrane narzędzie (od 2 do 10) zaznacza się samoczynnie,
3. **Operate Value** – wprowadzanie wartości dla obiektów panelu frontowego, edycja
4. **Object Shortcut Menu** – wymuszenie opcji obiektu dostępnych po wciśnięciu **PPM**,
5. **Connect Wire** – połączenie przewodów,
6. **Set/Clear Breakpoint** – uaktywnia lub dezaktywuje przerwanie wykonywania kodu,
7. **Position /Size /Select** – narzędzie wyboru komponentu, zmiany jego pozycji i rozmiaru na panelu (zmiana rozmiaru dotyczy tylko obiektów znajdujących się na panelu frontowym),
8. **Edit Text** – edycja nazw komponentów, tworzenie wolnych etykiet,
9. **Scroll Window** – przesuwanie okna – odpowiednik scrolli umieszczonych po prawej i dolnej stronie okna,
10. **Probe Data** – wstawia sondę do podglądu danych na przewód,
11. **Get Color** – „kroplomierz” pobiera kolor pola, nad, którym właśnie znajduje się – funkcja nigdy nie uaktywnia się automatycznie,
12. **Set Color** – zmienia kolor warstwy górnej oraz dolnej wybranego komponentu lub tła, wybór koloru odbywa się przez kliknięcie na odpowiedni kwadracik, możliwe jest także ustawienie koloru przeźroczystego () – funkcja nigdy nie uaktywnia się automatycznie.
13. **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)

* 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:

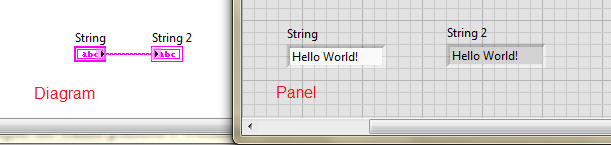
PPM 🡪 Change to Indicator

- 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),



1. Widok diagramu i panelu użytkownika pierwszego programu „*Hello World!*”.

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

- wyszukać kontrolkę typu Numeric i umieścić na panelu,

- zmienić nazwę kontrolki na „a”,

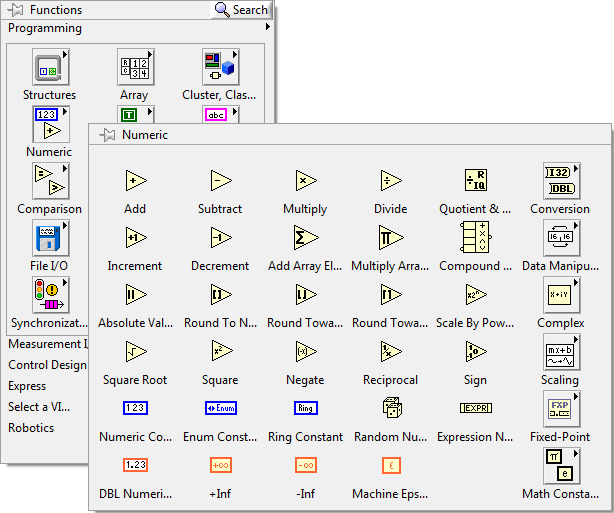
- wybraną metodą (kopiowanie, Quick-Drop, paleta kontrolek) umieścić dodatkowe 3 kontrolki typu Numeric,

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

- kontrolkę „y” zamienić na wskaźnik,

- przejść do okna diagramu,

- w palecie funkcji znaleźć funkcje matematyczne,



1. Widok zakładki Numeric w palecie funkcji.

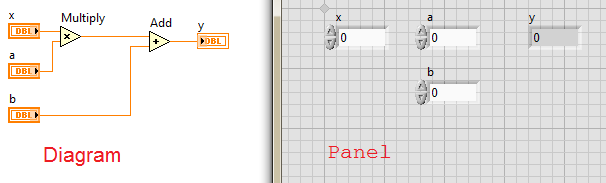
- sprawdzić i zapamiętać nazwę węzła mnożenia,

- wybrać i umieścić na diagramie węzeł dodawania,

- za pomocą Quick Drop dodać węzeł mnożenia (*Multiply*),

- połączyć elementy diagramu aby uzyskać funkcję liniową „”,

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



1. Widok diagramu i panelu użytkownika programu „*f\_liniowa.vi*”.

- 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 ,

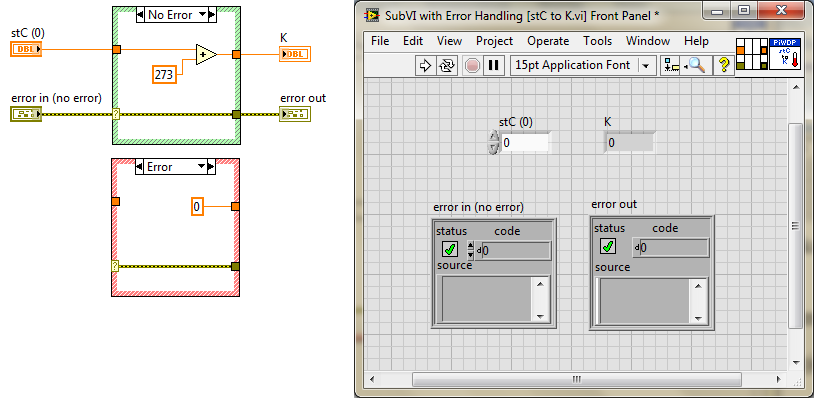
- zamknąć plik,

* 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 Celsjusza na Kelwiny (zgodnie z Rys. 14),



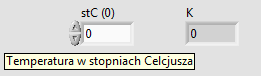
1. Program przeliczający temperaturę między skalami Celcjusza na Kewina.

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 Celsjusza”) – efekt przedstawiony poniżej, po najechaniu kursorem na kontrolkę,

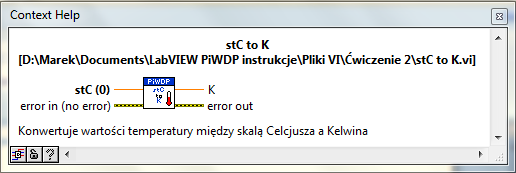


1. Widok kontrolki z wypełnioną opcją TipStrip.

- 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 Rys. 16.,



1. Pomoc kontekstowa utworzonego pliku.

- zapisać plik,

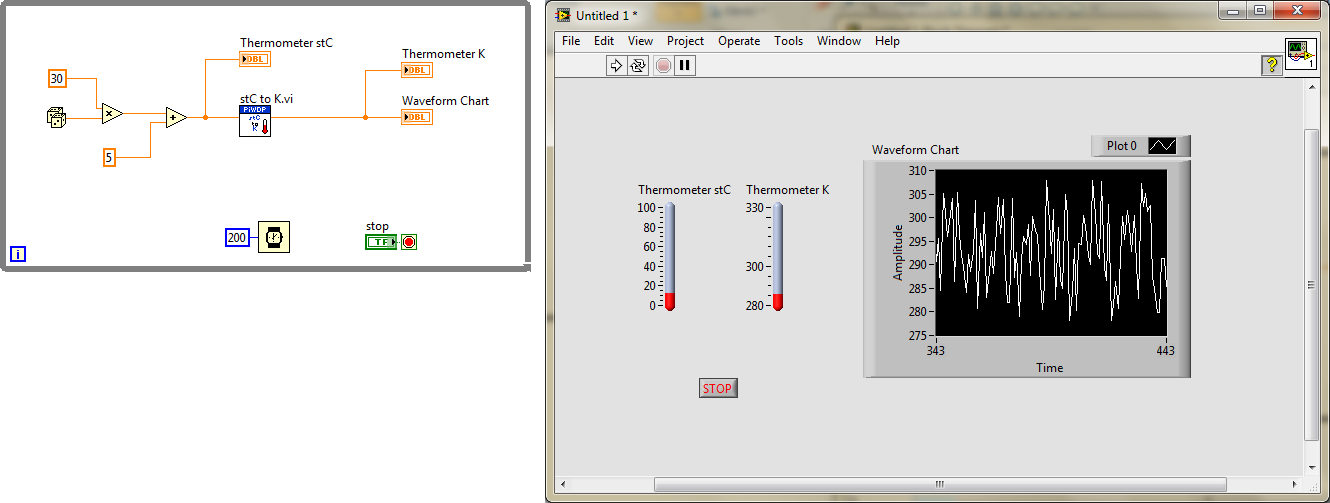
* Zapoznanie się z metodami debuggowania aplikacji i subVI:

- otworzyć nowy plik,

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

- sprawdzić listę błędów ,

- uzupełnić plik jak na Rys. 17.,



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) ,

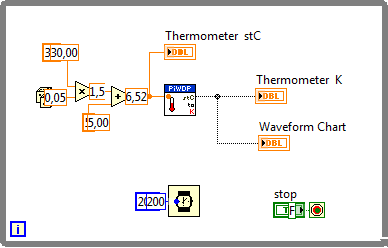
- zapisać plik pod nazwą „debugg app.vi”,

-zamknąć środowisko LabVIEW.

1. **PRZYKŁAD REALIZACJI ZADANIA – sprawdzenie działania funkcji Highlight Execution**

Uruchomienie opcji **Highligh Execution** (Rys. 4) powoduje zwolnienie działania aplikacji i wskazanie przepływu danych (dataflow). Rozwiązania tego nie można stosować w aplikacjach, w których istotne są zależności czasowe (np. komunikacja z urządzeniami np. za pomocą RS232).

Rys. 18 przedstawia widok diagramu w trakcie działania programu z uruchomioną funkcją Highlight Execution – widać wartości (lub rozmiar wektora) na przewodach za węzłami, które zostały już wykonane.



1. Widok działania programu z włączoną opcją Highlight Execution.
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. Opisać ideę programowania “dataflow”.
3. Jaka jest różnica między działaniem aplikacji podczas uruchamiania Step Over a Step Into?
4. Napisz wzór zapisany w postaci kodu LabVIEW



x = ………………………

# LITERATURA

1. LabVIEW Core 1 Participant Guide.
2. LabVIEW Core 2 Participant Guide.
3. 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

- 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**),

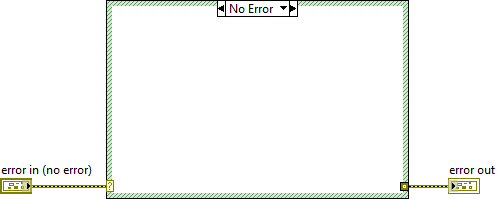
- przerobić plik f\_liniowa.vi (z pierwszej części ćwiczenia) zgodnie z wszystkimi wytycznymi dla plików subVI

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

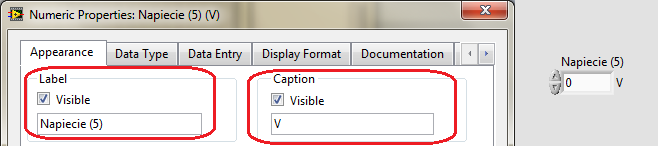
# ZAŁĄCZNIKI

## Dobre praktyki tworzenia podprogramów SubVI

1. Budowa struktury SubVI z obsługą klastra błędów, (proponowany szablon **File** 🡪 **New…** 🡪 **SubVI with error handling**),



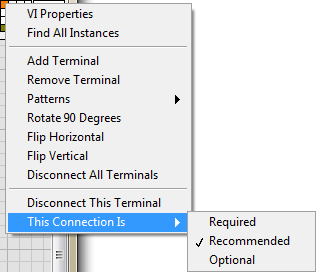
1. Nie zostawiać wartości domyślnych wskaźników w przypadku *Case ERROR*!
2. Nadać znaczące nazwy kontrolkom i wskaźnikom, z zaznaczeniem wartości domyślnych w nawiasach okrągłych (np. „*Napiecie (5)*”) – można nazwy na panelu czołowym podzielić na nazwę „Label” i opis „Caption” *Kontrolka* 🡪 **PPM** 🡪 **Properties** 🡪 **Apperance**



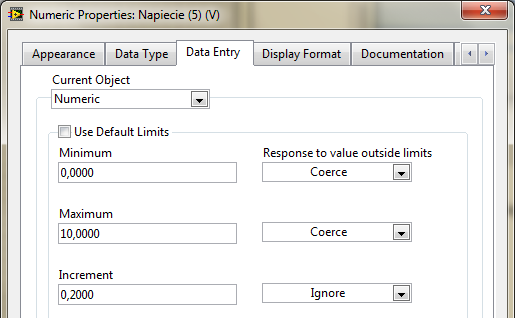
1. Nadanie wartości domyślnych kontrolkom: ustawić wartość domyślną, następnie: *kontrolka* 🡪 **PPM** 🡪 **Data Operations** 🡪 **Make Current Value default**,
2. Przypiąć wskaźniki i kontrolki do konektora połączeń (Conector Pane) tworząc tunele we/wy, pamiętając o zasadach przydzielania:



1. Przydzielić wejściom stopnień ważności (Wymagane, Rekomendowane, Opcjonalne), *tunel wejściowy w Conector Pane* 🡪 **PPM** 🡪 **This Connection Is…** 🡪 (*odpowiednia opcja*):



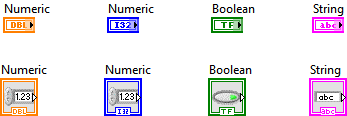
1. Uzupełnić dokumentację VI (*Conector Pane* 🡪 **PPM** 🡪 **VI Properties** 🡪 **Documentation**), dokumentacja powinna zawierać krótki opis, który będzie wyświetlany w pomocy kontekstowej – maksymalnie 3-5 zdań,
2. Umieścić w kodzie komentarze typu Free Label, ewentualnie z opcją # - zakładki możliwe do zobaczenia w **View** 🡪 **Bookmark Manager**,
3. \* Zadać kontrolkom zakresy możliwych wartości oraz krok:



1. \* Dla kontrolek i wskaźników uzupełnić krótkie podpowiedzi (tzw. Tip Strip – Kontrolka 🡪 **PPM** 🡪 **Properties** 🡪 **Documentation**),

## Wygląd wejściowych i wyjściowych komponentów pulpitu na diagramie

Komponenty pulpitu mają swoją reprezentację w oknie diagramu, niezależnie od typu oraz wyglądu na panelu w oknie diagramu może przyjąć jeden z dwóch widoków: widok ikony lub widok terminalu. Na rysunku przedstawiono obydwa widoki:



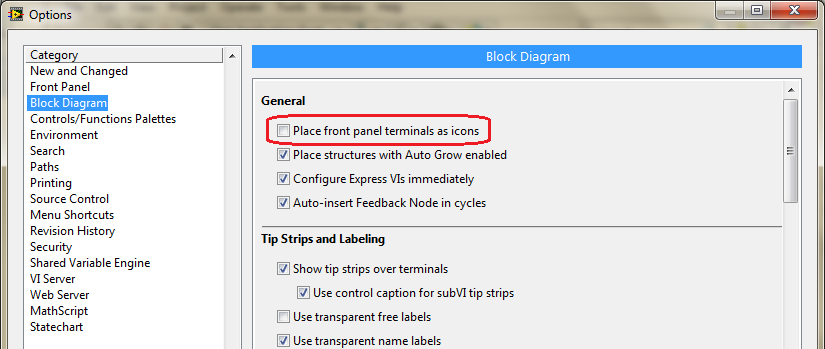
Zmianę widoku pojedynczej ikony można dokonać:

**PPM** 🡪 **View as Icon** (zaznaczyć lub odznaczyć).

Jeżeli chcemy zmienić domyślny wygląd terminalu w opcjach LabVIEW należy wybrać:

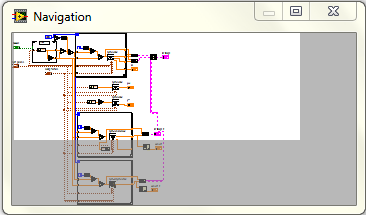
**Tools** 🡪 **Options…** 🡪 **Block Diagram**

(zaznaczyć lub odznaczyć opcję „**Place front panel terminals as icons**”)



## Okno Nawigacji

W przypadku, jeżeli w oknie diagramu/panelu nie mieści się cały kod programu/panel czołowy, czasem wygodnym narzędziem jest Okno Nawigacji: **View** 🡪 **Navigation Window**. W przypadku okna diagramu okno to zawiera cały kod programu. Białym obszarem zaznaczony jest aktualny widok okna diagramu, na szarym tle niewidoczna część kodu. W oknie nawigacji możliwe jest przełączanie widoku okna diagramu/panelu. Rozmiar okna nawigacji jest stały, im większy jest kod programu tym większa jest skala. Przyjętą zasadą w LabVIEW jest takie pisanie kodu, aby mieścił się w jednym oknie diagramu, jeżeli jest to niemożliwe ze względu na wielkość aplikacji, należy starać się tak pisać kod, aby do przesuwania okna korzystać tylko z jednego scrolla (poziomego lub pionowego).



## Zapisywanie plików wykonywalnych w postaci plików graficznych (snippet)

W LabWIEV istnieje możliwość tworzenia plików wykonywalnych ukrytych pod postacią pliku graficznego o rozszerzeniu PNG. Plik taki w swojej strukturze zawiera zrzut ekranu „printscreen” fragmentu kodu z diagramu oraz kod wykonywalny w postaci subVI. Próba podwójnego kliknięcia spowoduje uruchomienie domyślnego programu graficznego i wyświetlenie zawartości części graficznej pliku. Umieszczenie pliku na stronie internetowej także skutkować będzie wyświetleniem części graficznej (patrz link niżej). Uruchomienie tego typu pliku polega na przeciągnięciu go do okna diagramu metodą Drag & Drop.

Przygotowanie graficznego pliku wykonywalnego:

- zaznaczyć interesujący fragment kodu,

- **Edit** 🡪 **Create snippet from selection** 🡪 **zapisać plik z rozszerzeniem png**.

Zaletą tego typu rozwiązania w porównaniu do klasycznych printscreenów jest fakt, że w przypadku pętli warunkowych cały kod (z wszystkimi warunkami) zostanie zapisany w części uruchamialnej. Klasyczny printscreen wymaga wykonania zrzutu dla każdego warunku **Case** osobno.

Dodatkowe informacje oraz przykładowe snippety:

http://www.ni.com/tutorial/9330/en/

## Manager zakładek z podglądem kodu

Podstawowa wersja managera zakładek nie posiada okna podglądu kodu, w celu doinstalowania odpowiedniego dodatku należy wykonać następujące kroki:

- uruchomić LabVIEW Tools Network (VI Package Manager),

- przejść do okna JKI Package Manager,

- wyszukać paczkę: *NI Bookmark Manager with Block Diagram Preview*,

- zainstalować,

- uruchomić LabVIEW,

- uruchomić managera zakładek,

- wybrać nowo zainstalowanego managera z podglądem jako domyślnego\*.