LabView Podprogramy.
Jakub Gliński

Programowanie proceduralne; Podprogram.

- Programowanie proceduralne to paradygmat programowania zalecająca dzielenie kodu na procedury, czyli fragmenty wykonujące ściśle określone operacje.
- Podprogram (inaczej funkcja lub procedura) to wydzielona część programu wykonująca określone operacje.
- Podprogramy stosuje się, aby uprościć program główny i zwiększyć czytelność kodu.
- Definicja podprogramu oprócz jego kodu obejmuje też deklarację argumentów jego wywołania. Podprogram dysponuje argumentami wejściowymi i wyjściowymi.
- Argumenty wejściowe dostarczają dane z miejsca wywołania do podprogramu.

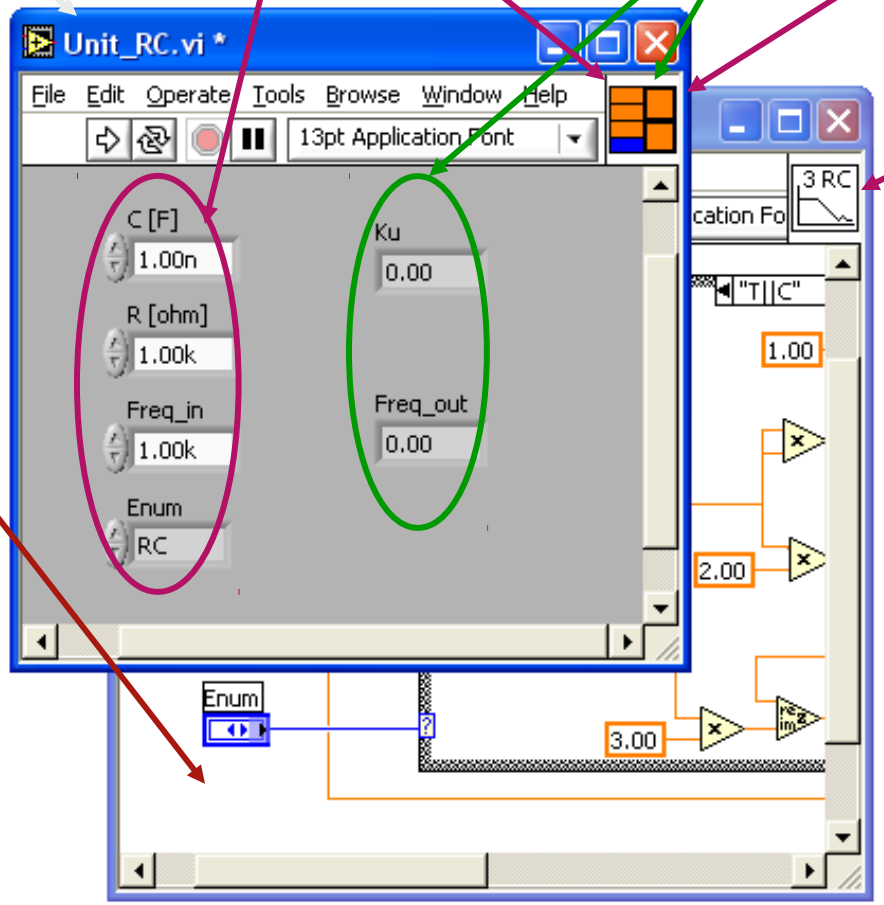
```
Void   my_function ( int arg1,  float arg2,  double *ptr1,  int *ptr2 )
```

- Argumenty wyjściowe zapewniają zwrócenie danych wynikowych do procedury wywołującej dany podprogram.

Podprogram w LabView (analogia do języka C)

Definicja funkcji w języku C:

```
Void  Unit_RC (SUIT Enum, float Freq_in, float C, float R,
                                        double &Ku,  double &Freq_out )
{ . . .
  . . .
}
```



Złącze – wraz z elementami kontrolnymi pulpitu definiuje wejścia i wyjścia subVI.

Ikona – unikalny obrazek identyfikujący węzeł podprogramu w diagramie wykorzystującym ten subVI.

Unikalność graficzna tylko dla zapewnienia rozróżnialności i opisu przeznaczenia węzła podczas edycji kodu diagramu lub jego analizy.

Technika kreowania podprogramu LabView

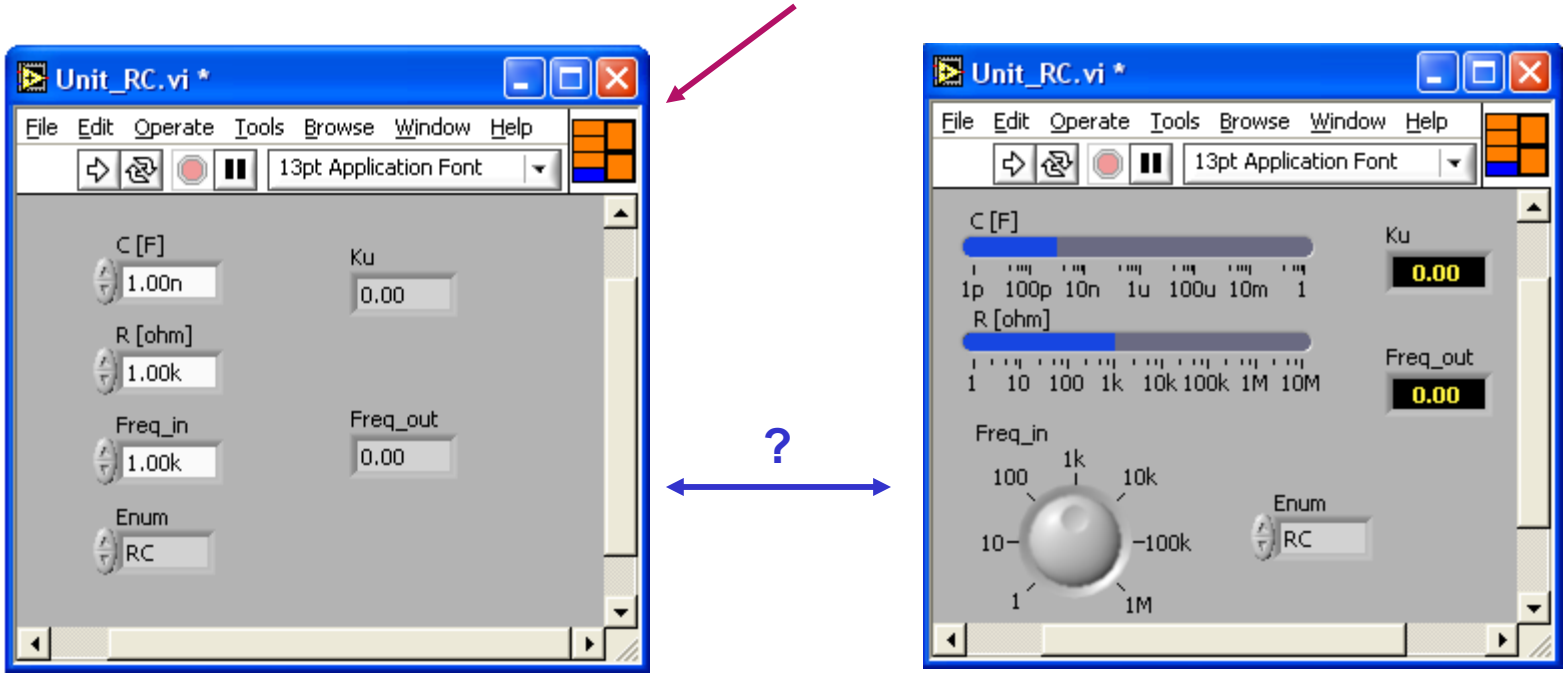
1. Zbudowanie programu VI realizującego algorytm podprogramu. Powstaje samodzielny program wykonujący zadania przewidziane dla podprogramu, złożony z pulpitu i diagramu. Program zostaje zachowany w postaci pliku VI o unikalnej nazwie. Każdy program VI jest potencjalnie podprogramem.
2. Edycja ikony reprezentującej podprogram w diagramach innych programów. Obrazek identyfikujący węzeł podprogramu w innych diagramach powinien być unikalny, aby zapewnić jednoznaczność odczytu kodu programu. Środowisko wykonawcze do rozpoznania subVI wykorzystuje nazwę pliku VI a nie ikonę.
3. Definicja argumentów podprogramu. Obejmuje:
   - Wytypowanie wejść i wyjść danych (elementy kontrolne i wskaźnikowe pulpitu podprogramu).
   - Określenie budowy złącza i przypisanie konkretnych wejść i wyjść do końcówek złącza.
   - Określenie statusu wejść.
4. Zdefiniowanie własności podprogramu.
5. Utworzenie informacji pomocniczych (wspomagających programistę).
6. Zapisanie dokonanych zmian projektu (plik o nazwie ustalonej w pkt 1).

Program VI z przeznaczeniem na podprogram.

Buduje się go tak jak każdy program VI. Różnica może dotyczyć wyłącznie projektu pulpitu, głównie w zakresie formy wizualnej.

Najczęściej użytkownik aplikacji widzi i posługuje się jej pulpitem. Pulpity zastosowanych podprogramów zwykle nie są uwidaczniane.

Jeśli podprogram ma spełniać typowe funkcje podprogramów forma jego pulpitu powinna być możliwie prosta.



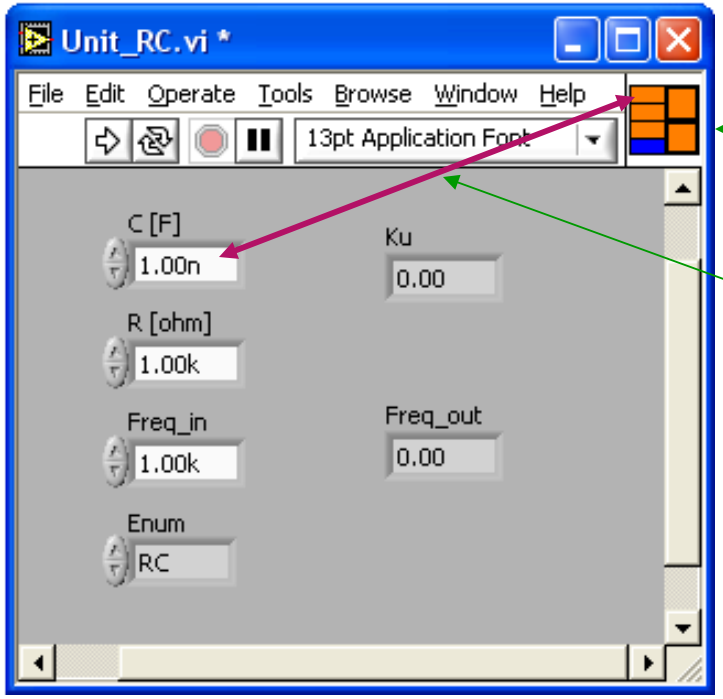
Można określić własności subVI tak, aby jego pulpit był widoczny. Wtedy forma graficzna pulpitu oraz własności obiektów kontrolnych są istotne (np. format wyświetlanych wartości numerycznych, kolory itp.).

Rola elementów pulpitu w podprogramie.

Każdy obiekt nastawczy pulpitu stanowi wejście danych i może być użyty jako argument wejściowy podprogramu, czyli wejście danych subVI.

Każdy obiekt wskaźnikowy pulpitu stanowi wyjście danych i może być użyty jako argument wyjściowy podprogramu, czyli wyjście danych subVI.

Nadanie obiektom pulpitu statusu wejść i wyjść następuje poprzez zdefiniowanie złącza podprogramu (subVI connector), które polega na powiązaniu poszczególnych pól złącza z obiektami pulpitu.



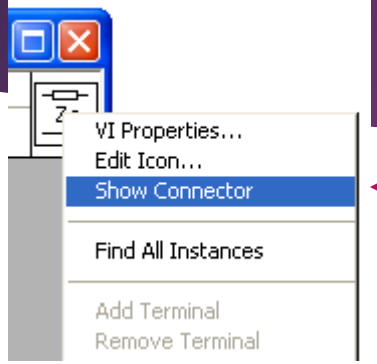
Złącze subVI.

Powiązanie wykonuje się za pomocą narzędzia ”szpulki”, zaznaczając wybrane pole złącza a następnie obiekt pulpitu.

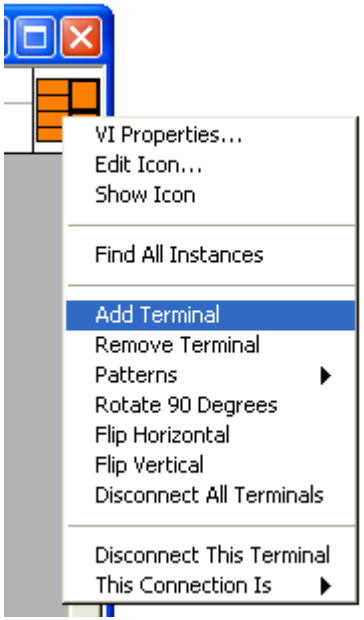
Pola bez powiązania z obiektem pulpitu mają kolor biały. Powiązane przyjmują kolory odpowiadające typom danych.

Edycja złącza subVI.

Edycję złącza można wykonać wyłącznie w oknie edytora pulpitu wykorzystując menu podręczne pola Ikona/Złącze.

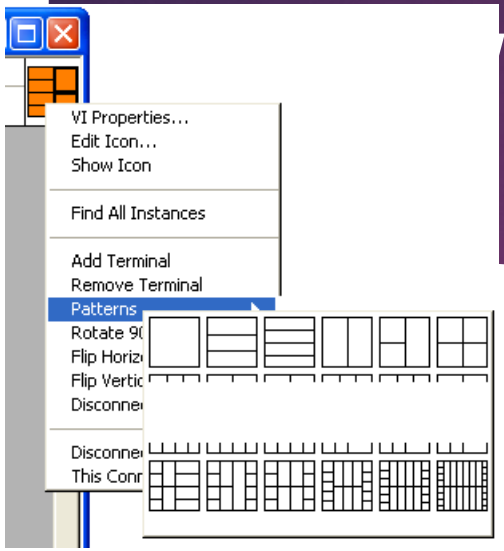


Wejście do edycji złącza podprogramu.



Edycja postaci złącza - liczba, rozkład pól, wzorce,.... itd.

ejść i wyjść
ramu.



Maksymalna liczba wejść i wyjść podprogramu jest podyktowana dopuszczalną liczbą pól złącza (28).

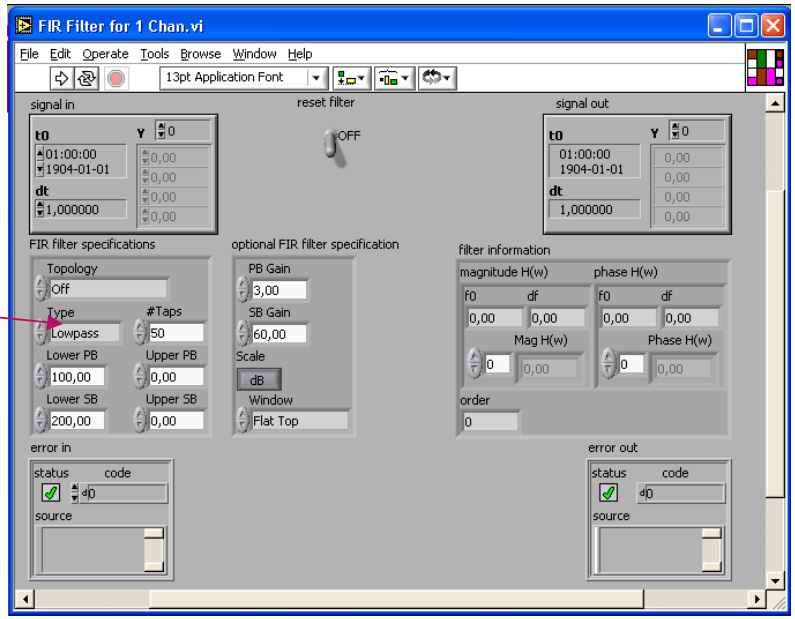
Przy dużej liczbie prostych danych wejściowych i wyjściowych można je grupować w typy złożone - struktury.

Powiązane logicznie dane można w ten sposób grupować zmniejszając liczbę osobnych wejść/wyjść podprogramu lub zapewnić dostarczenie więcej niż 28 prostych danych.

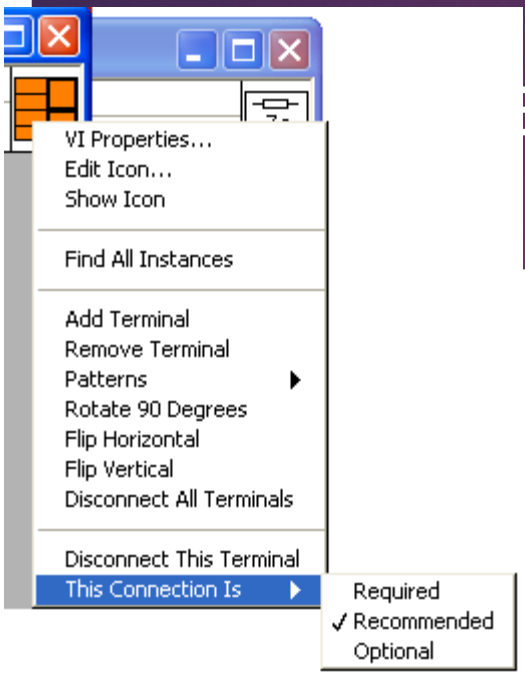
Przykład:

30 danych zgrupowanych w 7 strukturach, np. wejście specyfikacji filtru FIR jest klastrem złożonym z siedmiu prostych danych.

Liczba osobnych wejść i wyjść a także sposób grupowania prostych danych decyduje o wygodzie użycia podprogramu.



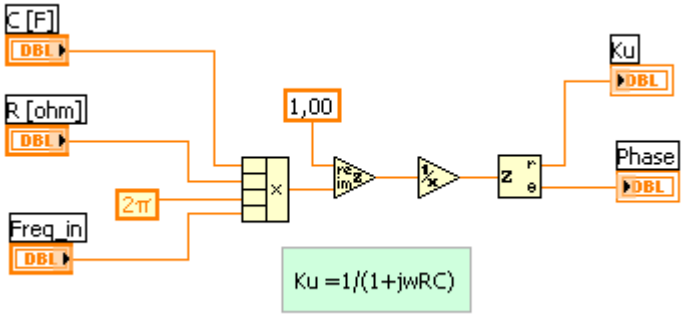
ejścia podprogramu.



Status doprowadzenia danej do wejścia:

- Wymagane - zawsze wymaga doprowadzenia danej;
- Polecane - wejście może pozostać nie podłączone do źródła danych (jest ostrzeżenie);
- Opcjonalne - wejście może pozostać nie podłączone do źródła danych (brak ostrzeżenia);

Wejścia ze statusem innym niż Required (Wymagane) potrzebują zdefiniowania dla nich wartości domyślnych wykorzystywanych przy braku danych zewnętrznych. Analogia do funkcji z argumentami domyślnymi.



Przykład:

Podprogram oblicza Ku dla zadanych wartości R, C i Freq_in.

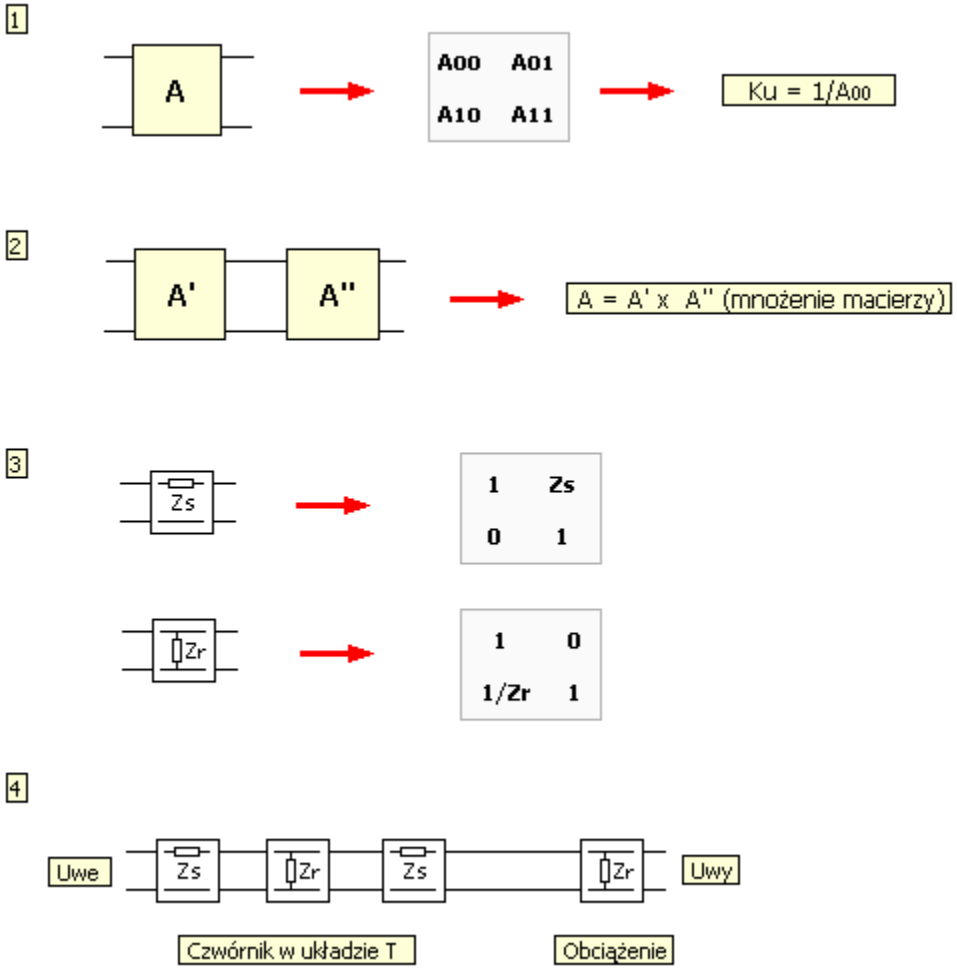
Wejścia R, C i Freq_in powinny zawsze korzystać z doprowadzonych danych.

Nie ma sensownego uzasadnienia dla wartości domyślnych.

Wejściom R, C i Freq_in należy przypisać status Required.

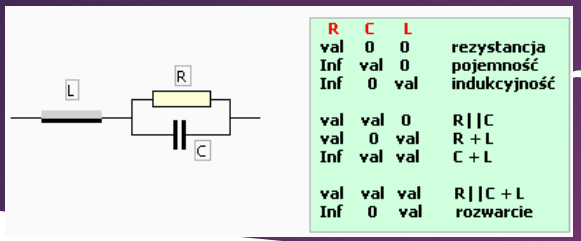
Obliczenie transmitancji kaskadowego układu czwórników z wykorzystaniem macierzy A.

Przykład projektu subVI :



mpedancji Zs :

Układ modelowania dowolnej impedancji.



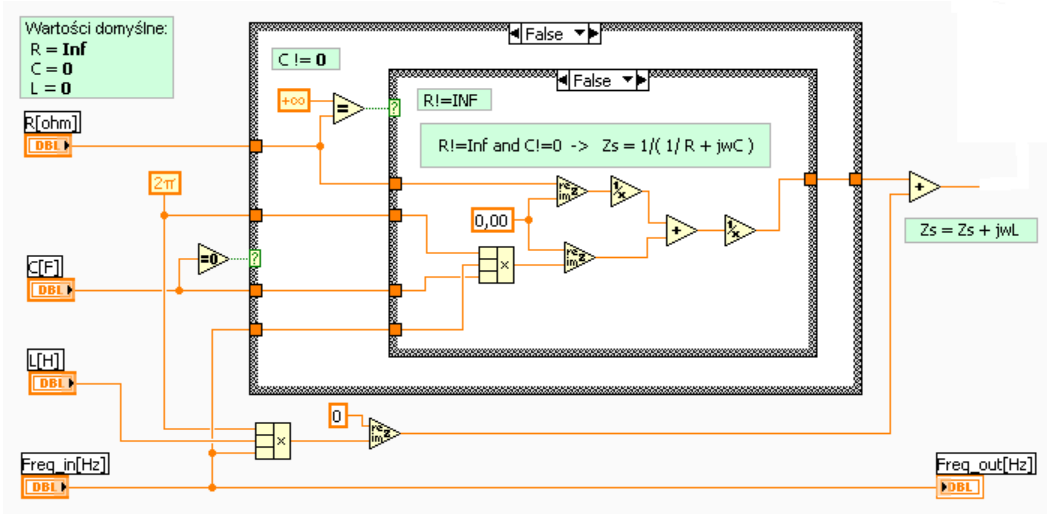
R	C	L	
val	0	0	rezystancja
Inf	val	0	pojemność
Inf	0	val	indukcyjność
val	val	0	R||C
val	0	val	R + L
Inf	val	val	C + L
val	val	val	R||C + L
Inf	0	val	rozwarcie

R	C	
val	val	-> Zs = 1/(1/ R + jwC)
Inf	val	-> Zs = 1/ jwC
val	0	-> Zs = R
Inf	0	-> Zs = 1e+15 (rozwarcie)

Obliczenie impedancji układu równoległego RC.

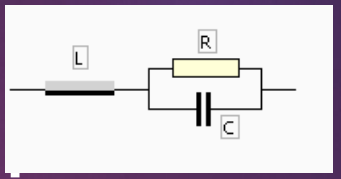
Zs = Zs + jwL **oraz dołączenie składowej indukcyjnej.**

Realizacja diagramu obliczającego impedancję Zs:



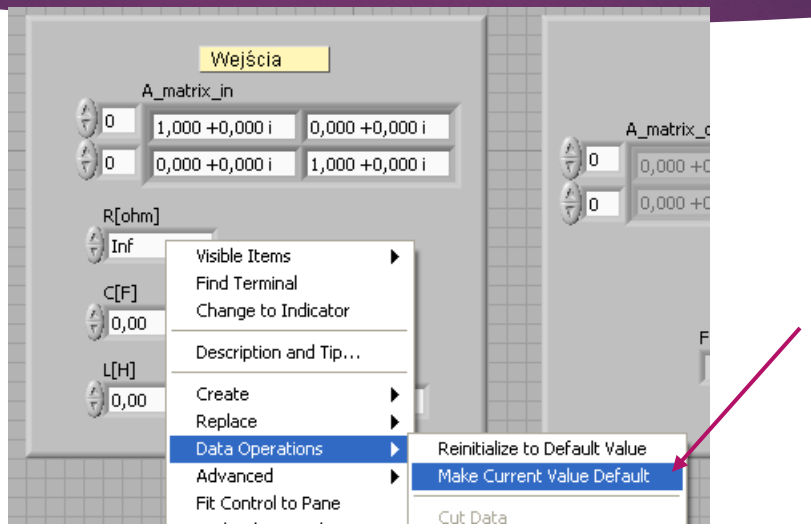
Określenie wartości domyślnej dla wejścia

Układ modelujący impedancję i wartości domyślne jego elementów:



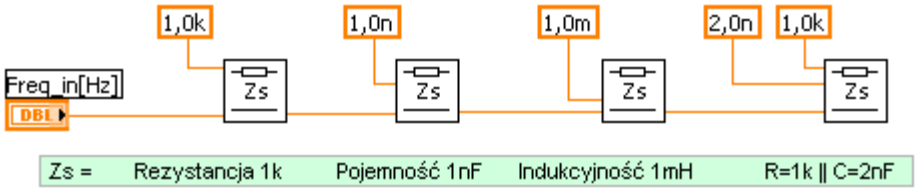
Wartości domyślne:
R = **Inf**
C = **0**
L = **0**

Przypisanie wartości domyślnych do elementów kontrolnych (wejść subVI):



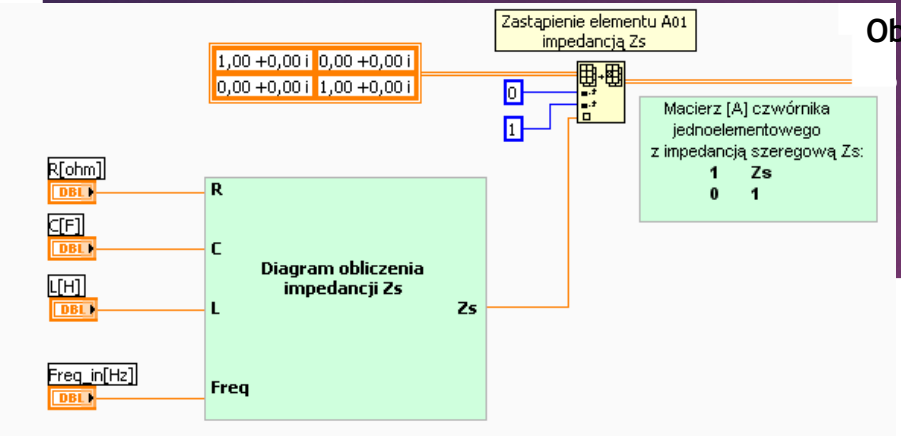
Poprzez dane z wejść R, C i L można praktycznie modelować dowolny rzeczywisty dwójnik.

Jeśli wejścia R, C i L otrzymają status Recommended oraz podane wartości domyślne to zapewni się łatwy sposób modelowania różnych impedancji Zs.

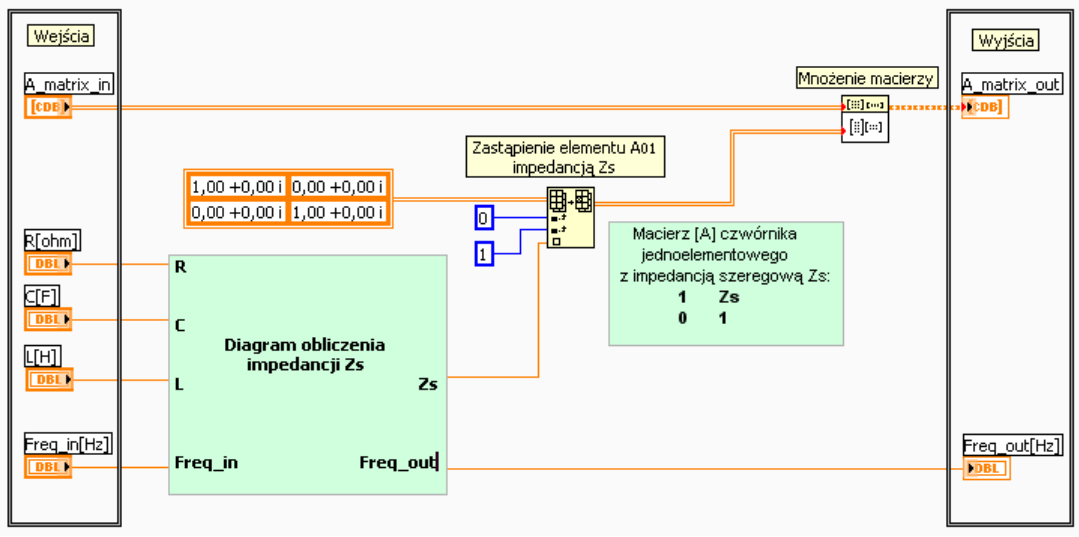


Obliczenie macierzy A czwórnika z impedancją szeregową Zs :

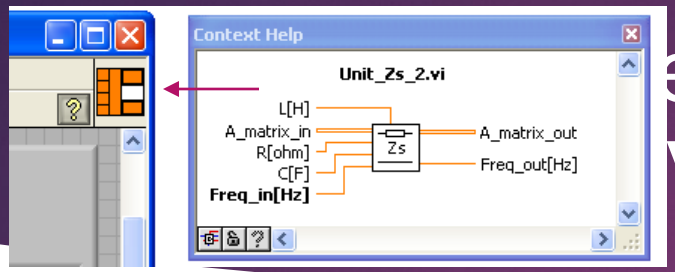
oraz kaskadowego



Rozszerzenie diagramu w celu obliczenia macierzy A połączonych kaskadowo czwórników:



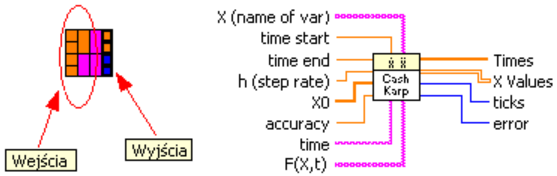
Przykład: subvi/test_matrix_A.vi



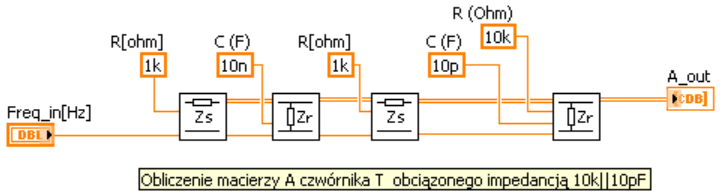
e położenia
VI:

Rozłożenie końcówek powinno zapewnić możliwie proste i czytelne połączenia pomiędzy węzłami, i tak:

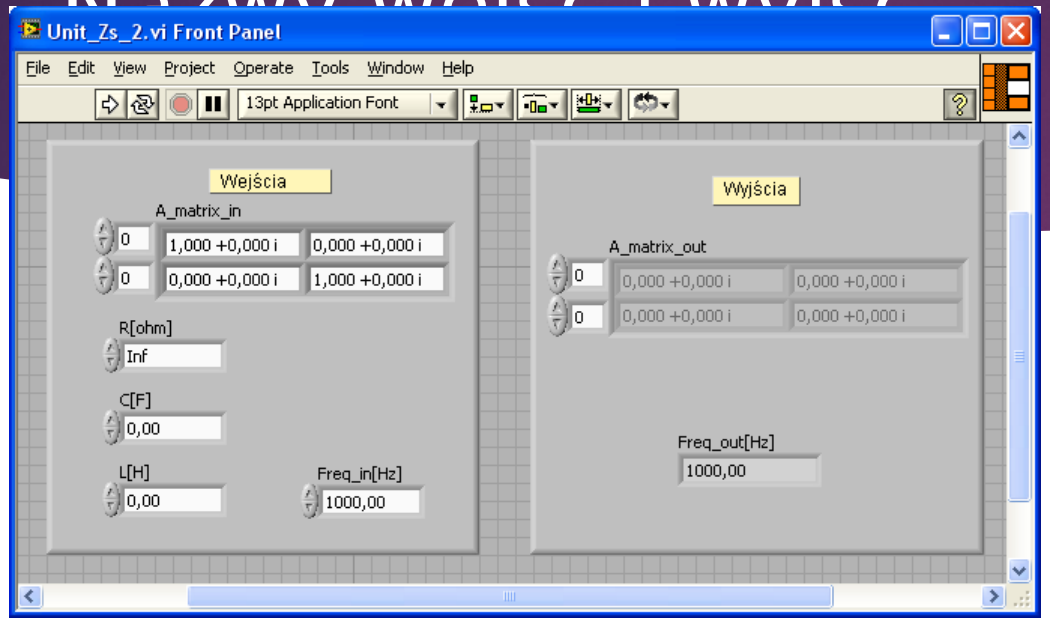
- **Końcówki wejściowe powinny znajdować się przy lewej krawędzi obrazu węzła. Jeśli jest ich więcej to należy umieścić je w lewej części węzła.**
- **Końcówki wyjściowe powinny znajdować się przy prawej krawędzi obrazu węzła. Jeśli jest ich więcej to należy umieścić je w prawej części węzła.**

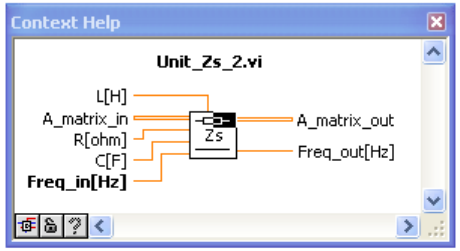


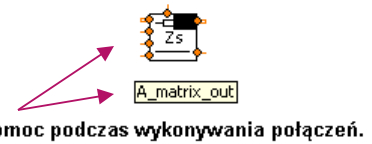
- **Węzły o podobnym przeznaczeniu umożliwiające kaskadowe przetwarzanie danych (tak jak np.: Unit_Zs i Unit_Zr) powinny stosować identyczne układy końcówek (połączenia na wprost);**
- **Wspólne dane wejściowe dla takich węzłów (tak jak np.: Freq w Unit_Zs) powinny mieć odpowiadające im wyjścia co pozwala prowadzić kaskadowe połączenie zamiast gwiazdowego, zapewniając jednocześnie kolejność wykonania wynikającą wprost z przepływu tej danej przez kolejne węzły.**



Nazwy wejść i wyjść





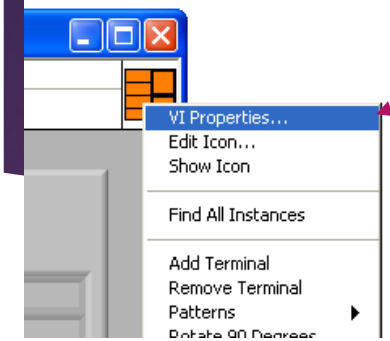


Obiekty pulpitu stanowiące wejścia i wyjścia subVI powinny otrzymać sensowne nazwy (własność Label obiektów). Powody:

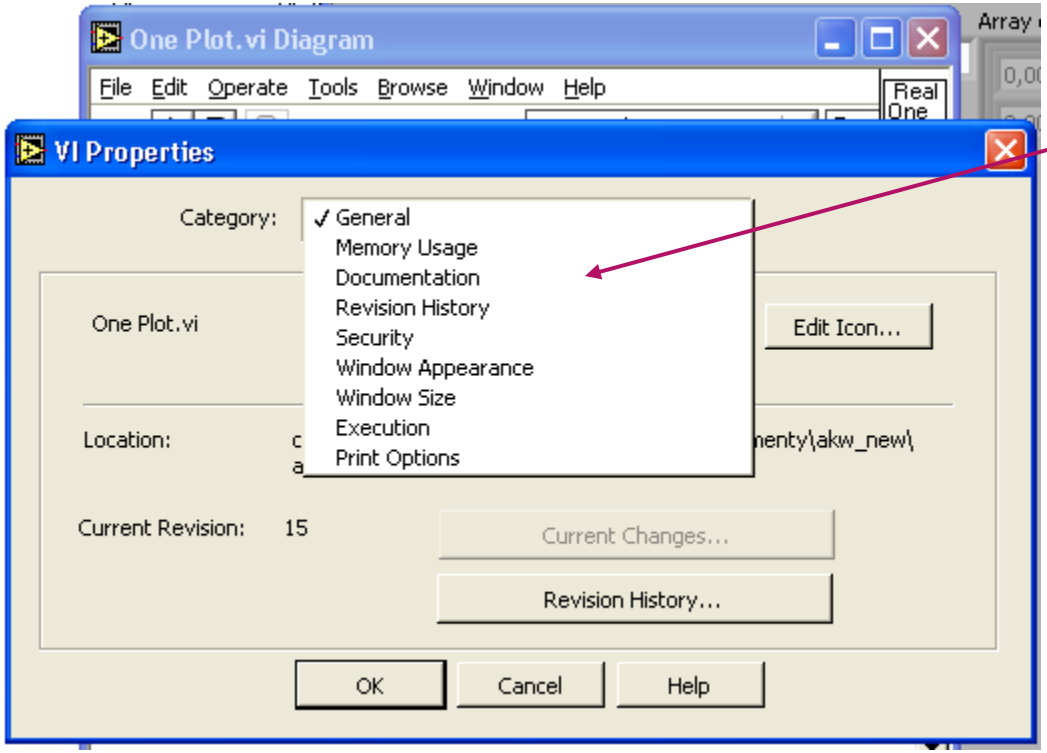
- **Czytelność projektu podprogramu;**
- **Help kontekstowy podaje automatycznie opis węzła w zakresie definicji wejść i wyjść.**
- **Przy aktywności narzędzia realizacji połączeń pokazywane są ich nazwy.**

Zastosowanie sensownych opisów będzie istotną pomocą dla programisty. Prosta auto-dokumentacja wskazuje przeznaczenie wejść i wyjść węzła.

Konfigurowanie własności programu.

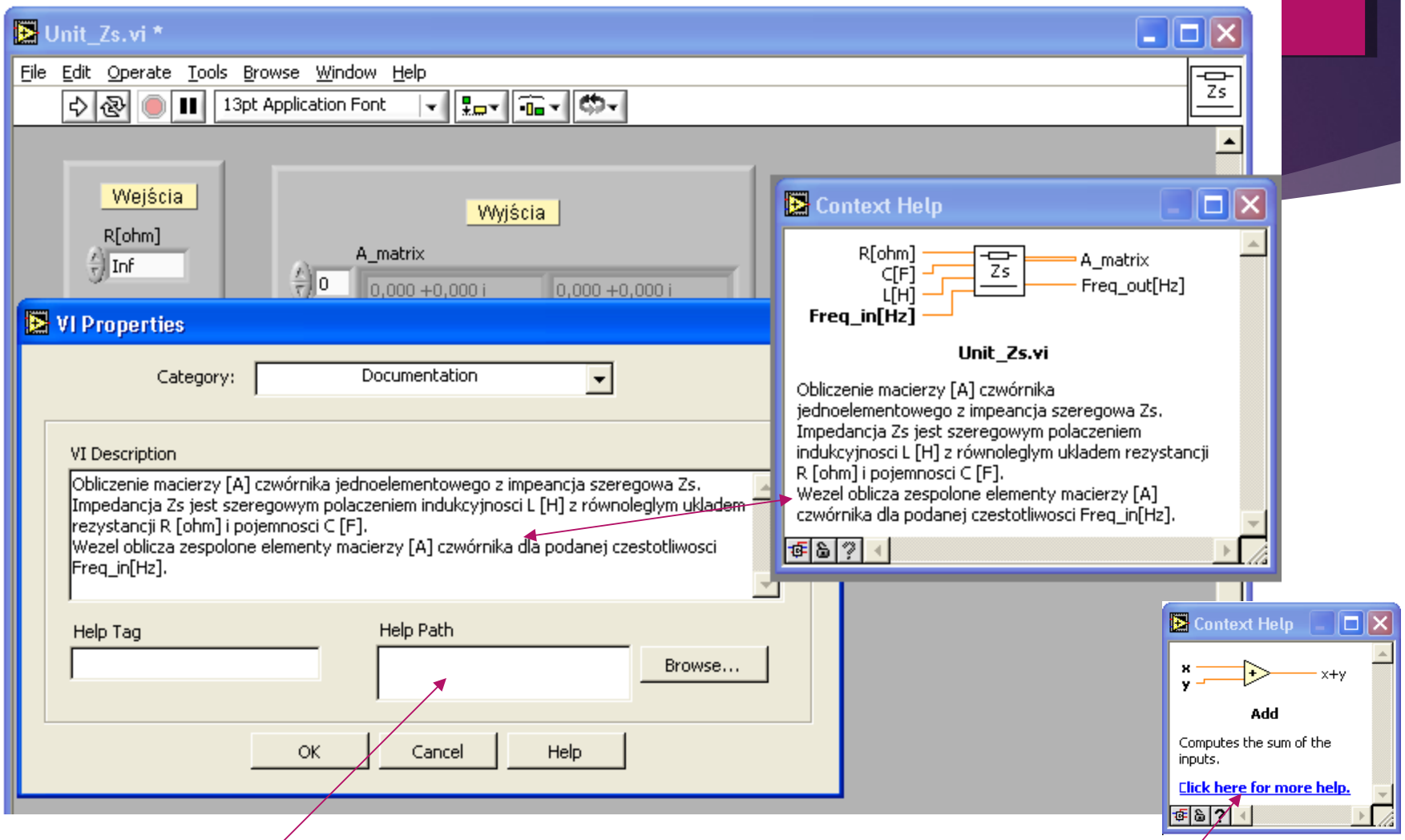


Wejście do definiowania własności programu VI; także podprogramu.

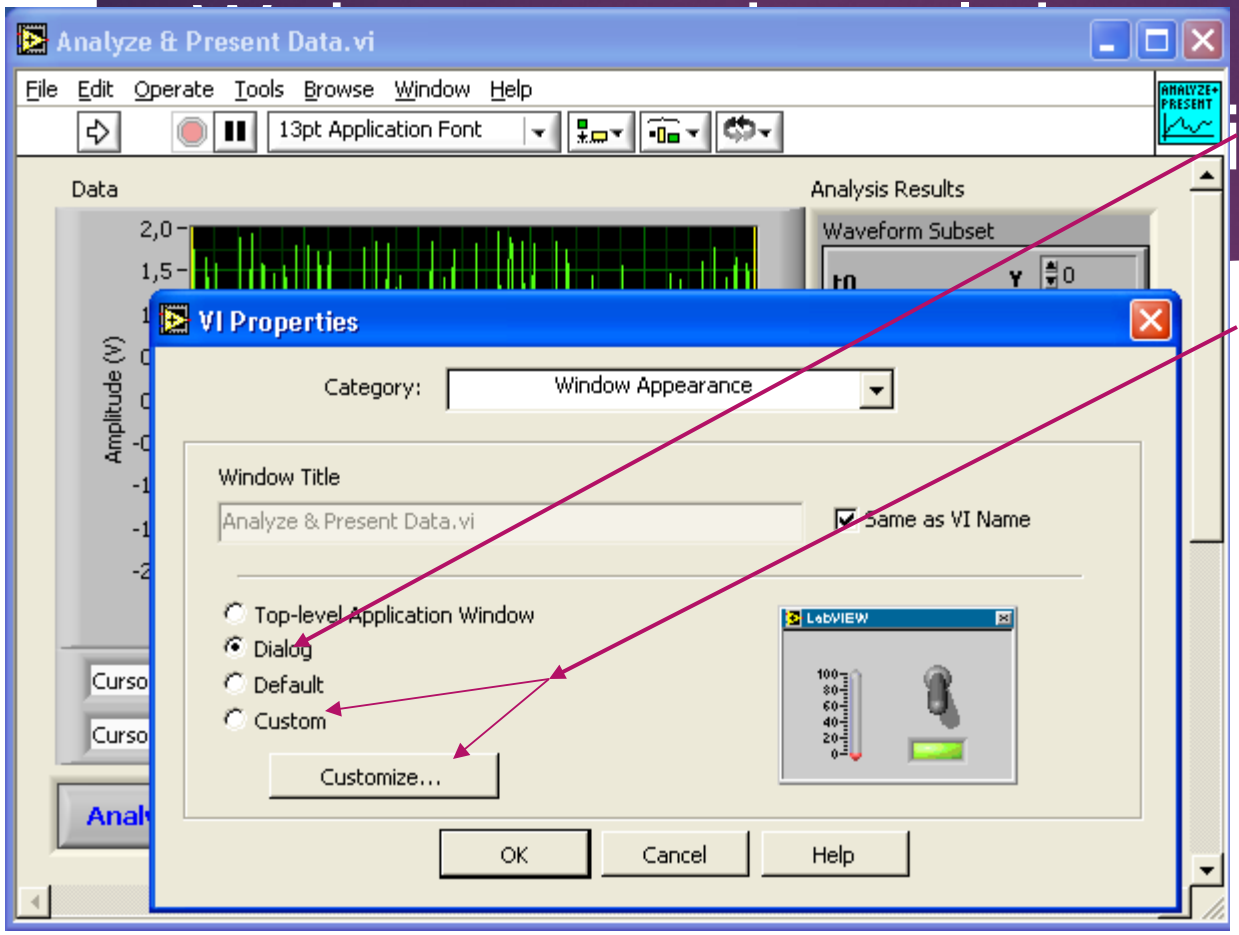


Kategorie konfigurowanych własności programu VI.

Najistotniejsze:
- Tryb wykonania
- Tryb użycia pulpitu
- Rozmiar i skalowalność pulpitu
- Dokumentacja pomocnicza
- Zabezpieczenie i prywatność projektów.



Pełny opis pomocniczy można zawrzeć w pliku z tekstem w formacie HLP lub HTML.
Odwołanie do niego umieścić w pozycji Help Path.
Help kontekstowy automatycznie podaje wówczas odnośnik do pełnej dokumentacji.



Zaznaczenie opcji Dialog oznacza, że pulpit podprogramu pojawi się na ekranie i pozostanie na nim do zakończenia działania podprogramu.

Podobne własności można uzyskać zaznaczając opcję Custom i w odpowiedni sposób konfigurując własności opcji stosownie do życzeń projektanta.

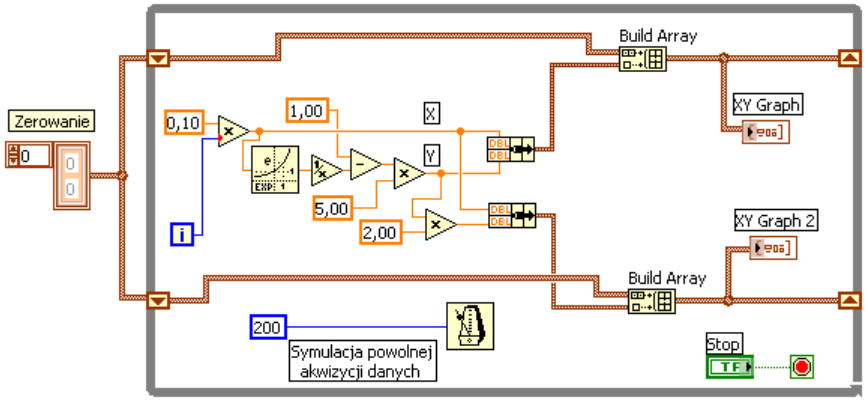
Opcja Dialog posiada szereg specyficznych, ustalonych własności.

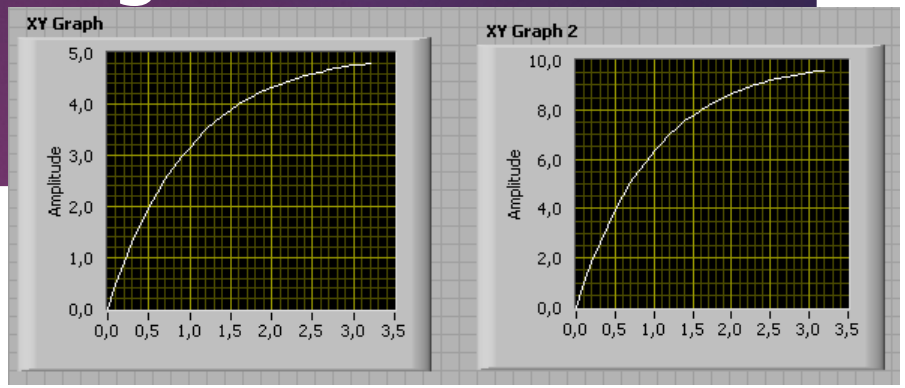
W opcji Dialog subVI zachowuje się jak okno dialogowe w systemie operacyjnym tak, że w czasie jego aktywności użytkownik nie może współpracować z innymi oknami LabView . Po wywołaniu podprogramu :

- Pulpit staje się widoczny ;
- Nie ma belki menu oraz narzędziowej;
- Użytkownik nie może modyfikować rozmiarów okna, ale może zamknąć okno.

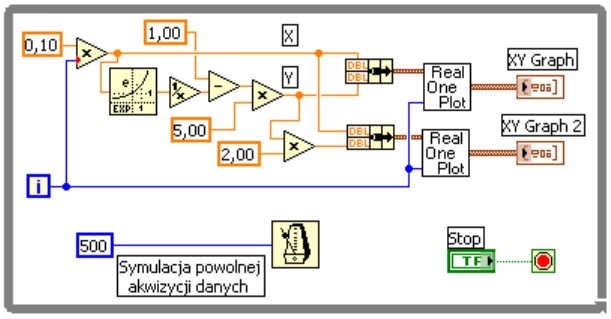
Wielokrotne, równoległe d

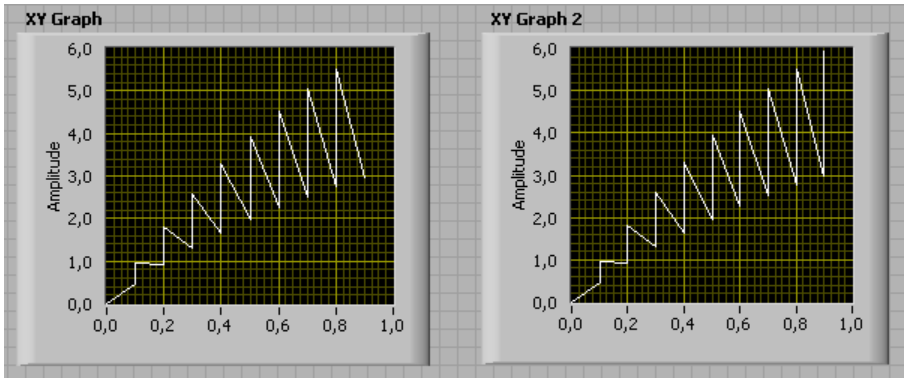
Kreślenie wykresu na bieżąco (w czasie pracy pętli) wymaga budowania tablicy oraz rejestru przesuwnego:





Można uprościć konstrukcję diagramu pętli przez zastosowanie subVI gromadzącego dane – ale wystąpi problem, jeśli kilka takich podprogramów będzie pracowało równolegle !

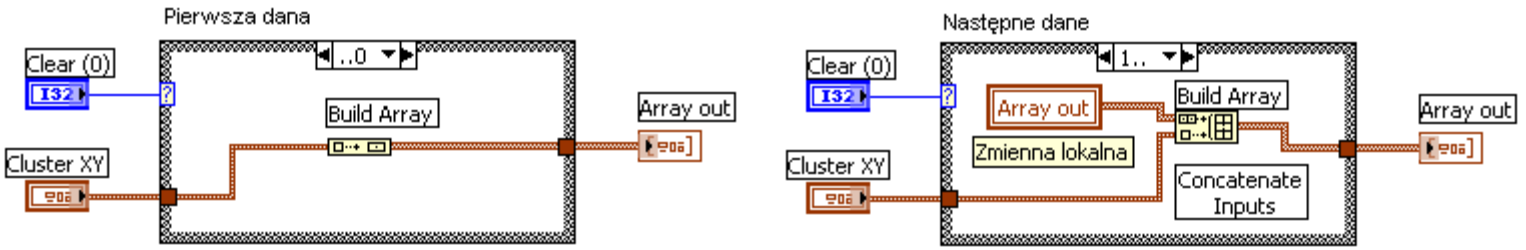




Przykład: Reentrant/wykres.vi oraz Reentrant/Reentrant1.vi

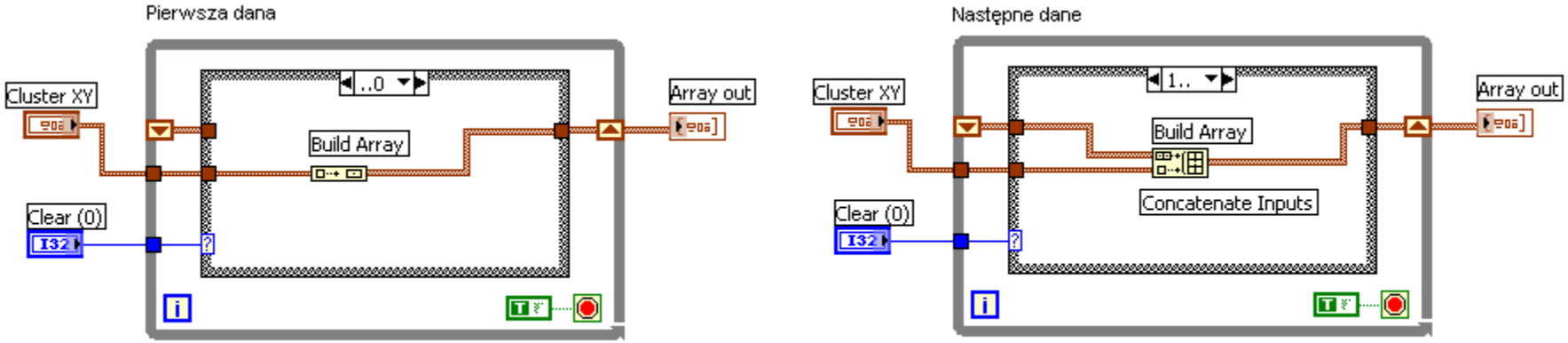
Podprogram gromadzący dane:

1. Rozwiązanie z zastosowaniem zmiennej lokalnej:
   - Podprogram gromadzi dostarczane struktury Cluster XY w tablicy Array out.
   - Gdy dana indeksująca Clear ma wartość 0, tworzona jest nawa tablica z jednym elementem XY.
   - Dla wartości indeksujących > 0 elementy XY są dołączane do istniejącej tablicy. Zrealizowano to za pomocą zmiennej lokalnej do tablicy i węzła budującego tablicę przez dołączanie.



Dwie ramki konstrukcji CASE podprogramu gromadzącego dane.

2. Rozwiązanie z zastosowaniem rejestru przesuwnego:
   - **Pętla wykonuje jedną iterację w każdym wywołaniu podprogramu.**
   - **Rejestr zachowuje dane pomiędzy kolejnymi wywołaniami podprogramu.**
   - **Jest zerowany automatycznie dla pierwszej danej (wejście Clear – wartość 0).**



Dwie ramki konstrukcji CASE podprogramu gromadzącego dane.

Sprawność działania programu.

Podczas operowania na dużych zbiorach danych są one udostępniane:

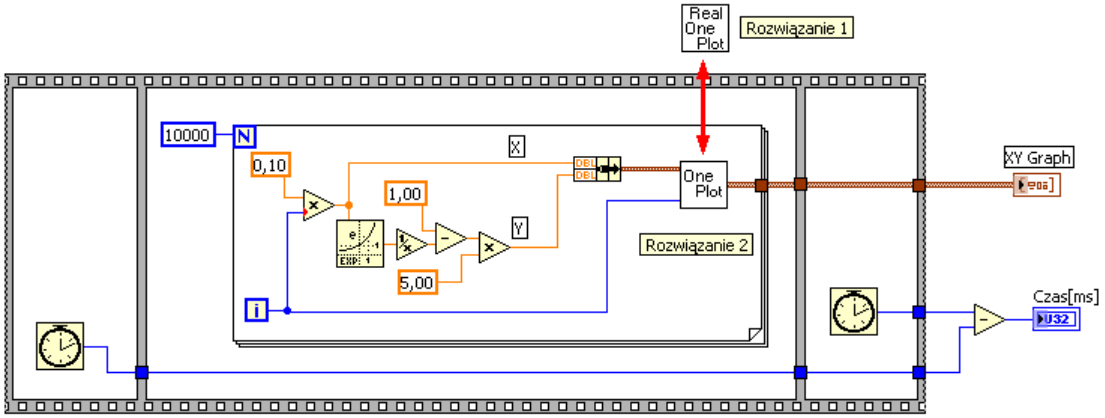
- Przez wykonanie kopii danych
- Przez przekazanie wskaźnika do miejsca ich zgromadzenia.

Operowanie na zmiennych lokalnych to tworzenie nowych kopii zgromadzonych danych.

Rejestr przesuwny działa na zaalokowanym obszarze pamięci i w swoim działaniu nie potrzebuje tworzenia kopii swojej zawartości.

Tworzenie kopii dużych zbiorów danych jest czasochłonne, dlatego druga wersja podprogramu (z rejestrem przesuwnym) jest lepszym rozwiązaniem.

Pomiar czasu wykonania programu wykorzystującego obie wersje podprogramu gromadzenia danych:



Czas wykonania programu z wersją 1 poprogramu - t1= 9493 ms

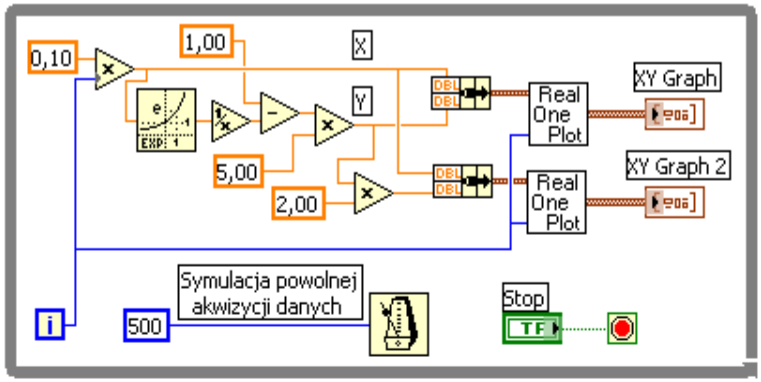
Czas wykonania programu z wersją 2 poprogramu - t2= 2216 ms

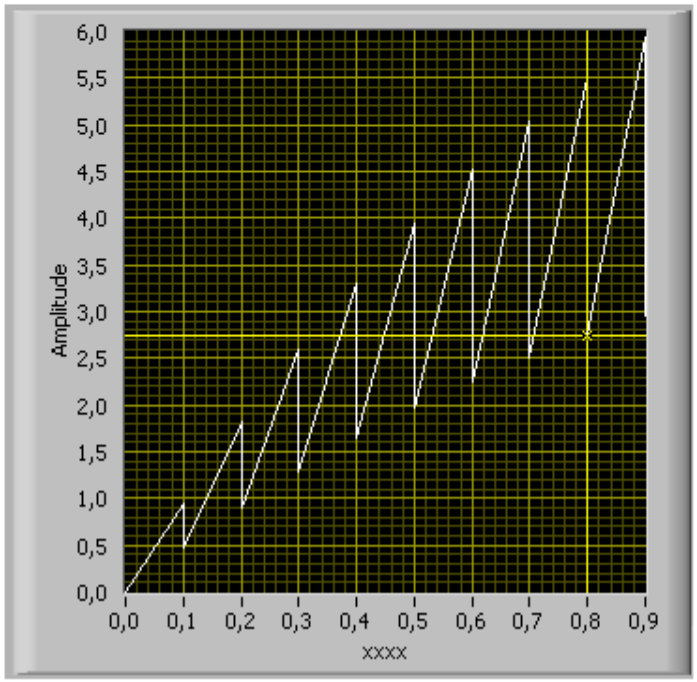
t1/t2 = 4.3

Wersja 2 jest około 4 razy szybsza od pierwszej.

Przykład: Reentrant/speed.vi

dla danych:





Dwa wystąpienia subVI w programie korzystają z tego samego zaalokowanego obszaru pamięci.

Ponieważ działają współbieżnie, składają na przemian dane z obu torów obliczeniowych w zaalokowanym obszarze.

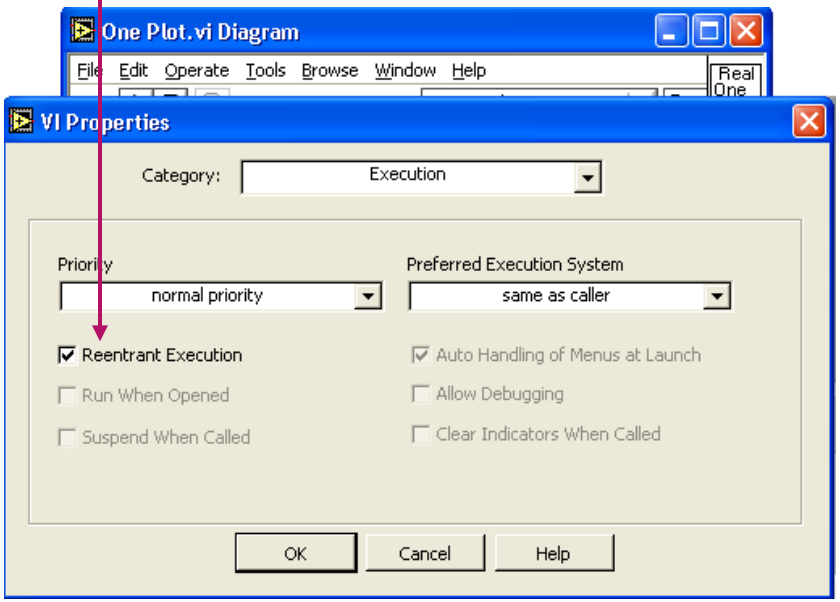
Skutkiem tego jest błędna prezentacja danych. Oba wykresy dostają te same dane.

Poprawnie, XY Graph powinien otrzymać punkty odpowiadające dolnej obwiedni wykreślonego kształtu a XY Graph2 punkty górnej obwiedni.

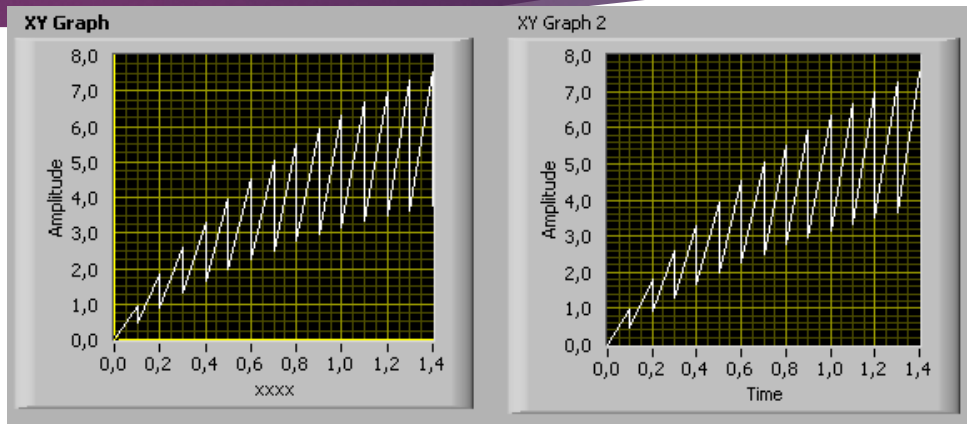
SubVI z możliwością wykorzystania w wielu instancjach.

Każdy podprogram, który zapamiętuje dane musi pracować w trybie Reentrant (wielobieżny), aby mógł występować w wielu instancjach.

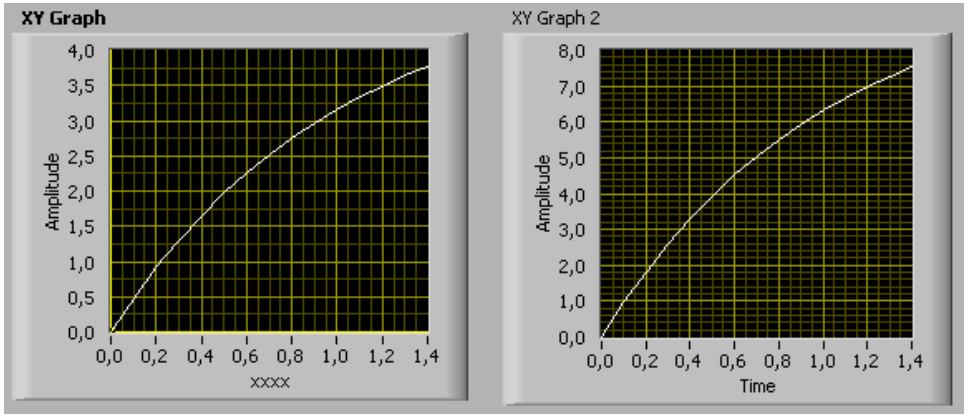
Wtedy każda jego instancja tworzy własny obszar dla danych.



Tryb Reentrant wyłączony :



Tryb Reentrant włączony :



Przykład: Reentrant/Reentrant2.vi