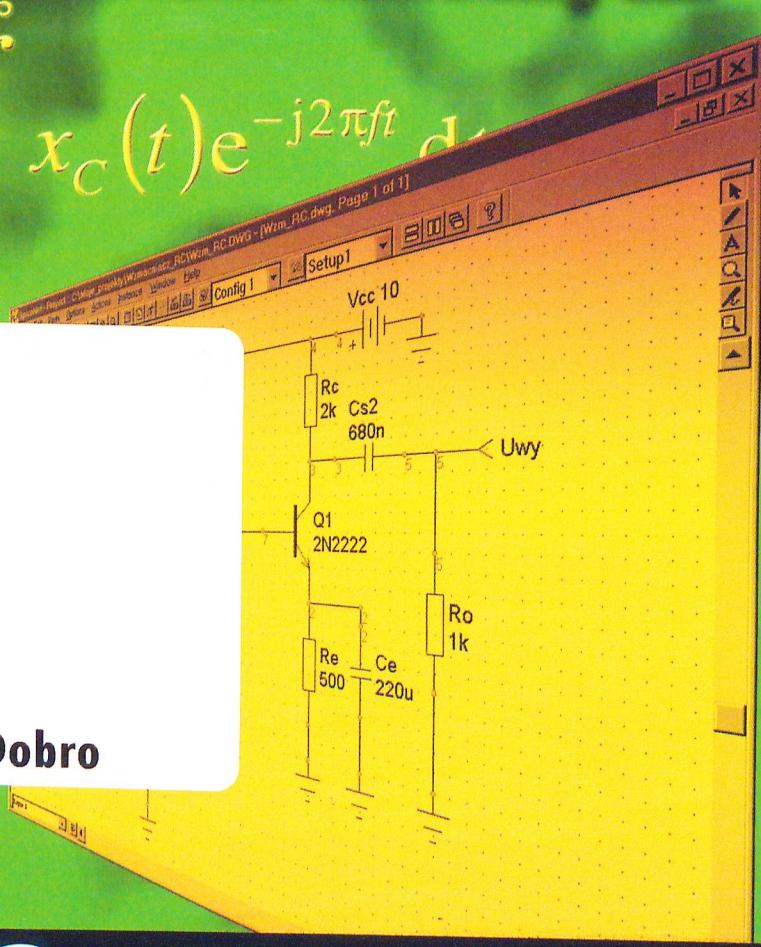


Andrzej Dobrowolski

Pod maską SPICE'a

Metody i algorytmy analizy układów elektronicznych

$$X_C(f) = \int_{-\infty}^{\infty} x_C(t) e^{-j2\pi ft} dt$$



Wolski, Andrzej.

Wską SPICE'a :

8075

Z.XV.3.Dobro

btc

W książce przedstawiono obszerny opis języka symulacyjnego SPICE, który jest jednym z podstawowych narzędzi komputerowej analizy obwodów elektronicznych, zarówno analogowych, jak i cyfrowych. Po wprowadzeniu, zawierającym teoretyczne podstawy algorytmów stosowanych w języku SPICE, zaprezentowano opis podstawowych rodzajów analiz dostępnych w programie i przykłady ich użycia.

Książka jest adresowana do studentów wydziałów elektronicznych i elektrycznych wyższych uczelni technicznych oraz do wszystkich elektroników zainteresowanych komputerową analizą układów elektronicznych.

Recenzent: *prof. dr hab. inż. Stanisław Osowski*

ISBN 83-921073-3-0

© Copyright by Wydawnictwo BTC
Warszawa 2004.



Wydawnictwo BTC
ul. Inowlodzka 5
03-237 Warszawa
fax: (22) 814-13-02
<http://www.btc.pl>
e-mail: redakcja@btc.pl

Wydanie I.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawnictwo BTC dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawnictwo BTC nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentów niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Druk i oprawa: Łódzka Drukarnia Dzielowa S.A.

Od autora	6
Przedmowa	7
Część 1. Podstawy	13
1. Ogólne uwagi o komputerowych metodach analizy układów elektronicznych.....	14
2. Węzłowe metody formułowania równań równowagi obwodów prądu stałego	22
2.1. Metoda węzłowa	22
2.2. Zmodyfikowana metoda węzłowa	24
2.3. Konstrukcja macierzy poprzez przeglądanie	26
3. Metody rozwiązywania układu liniowych równań algebraicznych	32
3.1. Metoda eliminacji Gaussa.....	32
3.2. Metoda dekompozycji LU	38
3.3. Uwagi o technikach macierzy rzadkich.....	39
4. Komputerowe modele elementów układu	41
4.1. Klasifikacja i hierarchia modeli.....	41
4.2. Modele wybranych elementów pasywnych i aktywnych.....	43
4.2.1. Model rezystora.....	43
4.2.2. Model tranzystora bipolarnego	44
4.3. Modele szumowe	55
Część 2. Analizy	61
5. Analiza stałoprądowa układów nieliniowych	62
5.1. Algorytm Newtona-Raphsona.....	62
5.2. Iterowane modele elementów nieliniowych	66
5.3. Modyfikacje algorytmu ułatwiające uzyskanie zbieżności	71
5.3.1. Problem zbieżności obliczeń	71
5.3.2. Modyfikacja warunków stopu	72
5.3.3. Ustalanie minimalnej i maksymalnej konduktancji układu	74
5.3.4. Wymuszanie potencjałów startowych.....	77
5.3.5. Parametryzacja źródeł	79
5.3.6. Blokowanie elementów nieliniowych.....	79
6. Malosygnalowe analizy częstotliwościowe	80
6.1. Analiza zmiennoprądowa w stanie ustalonym	80
6.2. Analiza zniekształceń nieliniowych.....	81
6.3. Analiza szumowa	87
7. Analiza czasowa układów dynamicznych	89
7.1. Metody całkowania numerycznego	89
7.1.1. Analiza czasowa jako problem numeryczny	89
7.1.2. Algorytmy Eulera.....	90

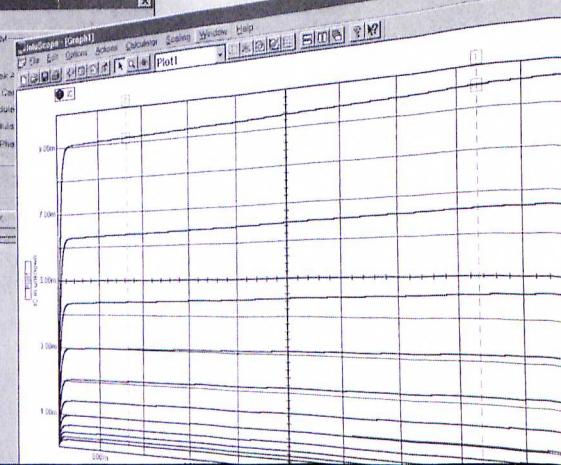
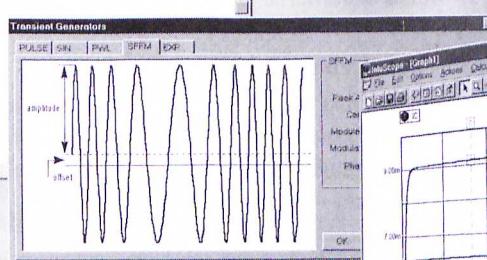
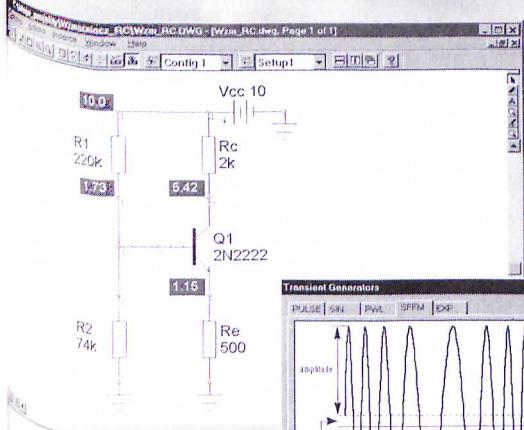
7.1.3.	Algorytm trapezów	92
7.1.4.	Algorytmy Geara.....	92
7.1.5.	Porównanie algorytmów całkowania numerycznego	93
7.2.	Modele stwarzyszone elementów zachowawczych.....	94
7.2.1.	Modele stwarzyszone idealnego kondensatora.....	94
7.2.2.	Modele stwarzyszone idealnej cewki	98
7.3.	Warunki początkowe.....	99
7.4.	Dynamiczna zmiana kroku	101
8.	Analiza wrażliwości.....	108
8.1.	Wrażliwość w analizie układów elektronicznych	108
8.2.	Analiza przyrostowa	110
8.3.	Analiza Monte Carlo.....	111
8.4.	Analiza najgorszego przypadku.....	112
Część 3. Zastosowania	113	
9.	Składnia języka opisu układów elektronicznych	114
9.1.	Zasady opisu topologii układu.....	114
9.2.	Zasady opisu behawioralnego.....	116
9.3.	Deklaracje elementów binarnych.....	120
9.3.1.	Rezystor.....	120
9.3.2.	Kondensator	120
9.3.3.	Cewka.....	121
9.3.4.	Indukcyjność wzajemna.....	121
9.3.5.	Klucz sterowany.....	121
9.4.	Deklaracje elementów półprzewodnikowych	122
9.4.1.	Dioda	122
9.4.2.	Tranzystor bipolarny	122
9.4.3.	Tranzystor polowy JFET.....	123
9.4.4.	Tranzystor polowy MOSFET	124
9.4.5.	Tranzystor polowy MESFET	124
9.5.	Deklaracje źródeł	125
9.5.1.	Źródła niezależne	125
9.5.2.	Źródło prądowe sterowane prądem	130
9.5.3.	Źródło prądowe sterowane napięciem	130
9.5.4.	Źródło napięciowe sterowane prądem	131
9.5.5.	Źródło napięciowe sterowane napięciem	131
9.5.6.	Źródło behawioralne	131
9.6.	Deklaracje analiz.....	132
9.6.1.	Stałoprądowa analiza punku pracy	132
9.6.2.	Krokowa analiza stałoprądowa	132
9.6.3.	Małosygnalowa analiza zmiennoprądowa	133
9.6.4.	Małosygnalowa analiza zniekształceń nieliniowych.....	133
9.6.5.	Analiza szumowa	134

9.6.6.	Analiza czasowa.....	134
9.6.7.	Analiza harmoniczna.....	135
9.6.8.	Małopryrostowa analiza wrażliwości.....	136
9.7.	Polecenia sterujące i pomocnicze	136
9.7.1.	Sterowanie zbieżnością obliczeń	136
9.7.2.	Definiowanie warunków początkowych.....	136
9.7.3.	Wyprowadzanie wyników	137
9.7.4.	Deklaracja i definicja modelu.....	137
9.7.5.	Dołączanie bibliotek	138
9.7.6.	Deklaracja i definicja podobwodu	138
9.8.	Interaktywny język sterowania symulacją ICL	140
9.8.1.	Operatory.....	141
9.8.2.	Funkcje	142
9.8.3.	Komendy języka ICL.....	142
10.	Analiza układów elektronicznych za pomocą symulatora ICAP/4.....	148
10.1.	Struktura symulatora ICAP/4.....	148
10.2.	Wprowadzanie schematu ideowego analizowanego układu	149
10.3.	Zastosowanie języka ICL.....	156
10.4.	Analiza punktu pracy	159
10.5.	Krokowa analiza stałoprądowa.....	162
10.6.	Analiza temperaturowa	168
10.7.	Małosygnałowe analizy częstotliwościowe	169
10.7.1.	Analiza zmiennoprądowa.....	169
10.7.2.	Analiza zniekształceń nieliniowych.....	178
10.7.3.	Analiza szumowa	182
10.8.	Analiza czasowa.....	186
10.9.	Analiza harmoniczna.....	194
10.10.	Małopryrostowa analiza wrażliwości.....	196
Dodatki.....	197	
Dodatek A. Opcje programu ISSPICE 4	198	
Dodatek B. Sposoby rozwiązywania problemów	203	
Dodatek C. Aktualizacja bibliotek w programie ICAP/4 Windows.....	205	
Dodatek D. Dyskretnie przekształcenie Fouriera.....	211	
Literatura.....	227	
Skorowidz.....	229	

3

Część 3

Zastosowania



9. Składnia języka opisu układów elektronicznych

W niniejszym rozdziale usystematyzowano najistotniejsze informacje dotyczące składni poleceń stosowanych w programie SPICE. Kolejno przedstawiono: zasady opisu topologii układu i programowania behawioralnego, deklaracje wybranych elementów biernych i półprzewodnikowych, deklaracje źródeł niezależnych i sterowanych oraz deklaracje analiz i deklaracje pomocnicze. Rozdział zakończono krótkim opisem skryptowego języka sterowania symulacją *ICL* (ang. *Interactive Command Language*). Pełne informacje związane z konkretną implementacją programu SPICE znajdzie Czytelnik w podręczniku użytkownika dołączonym do programu oraz w plikach pomocy dostępnych w Internecie na stronie producenta. Rolą niniejszego rozdziału jest uporządkowanie rozproszonych opisów i zaprezentowanie ich w formie wygodnej dla polskiego użytkownika. Zapoznanie się z treścią kolejnych podrozdziałów pozwoli w pełni zrozumieć pracę simulatora, często ukrytą za kurtyną tworzoną przez edytor schematów z kreatorami i scenariuszami symulacji.

9.1. Zasady opisu topologii układu

Program SPICE przeprowadza symulację na podstawie tekstuowego zbioru wejściowego, zawierający listę połączeń (ang. *netlist*) oraz wykaz analiz przechowywany w pliku dyskowym z rozszerzeniem **.cir* (ang. *circuits*). Plik ten może być utworzony „ręcznie” za pomocą dowolnego edytora tekstowego zapisującego w kodzie ASCII bądź, jak to się dzieje najczęściej, może być wygenerowany automatycznie przez graficzny edytor schematów.

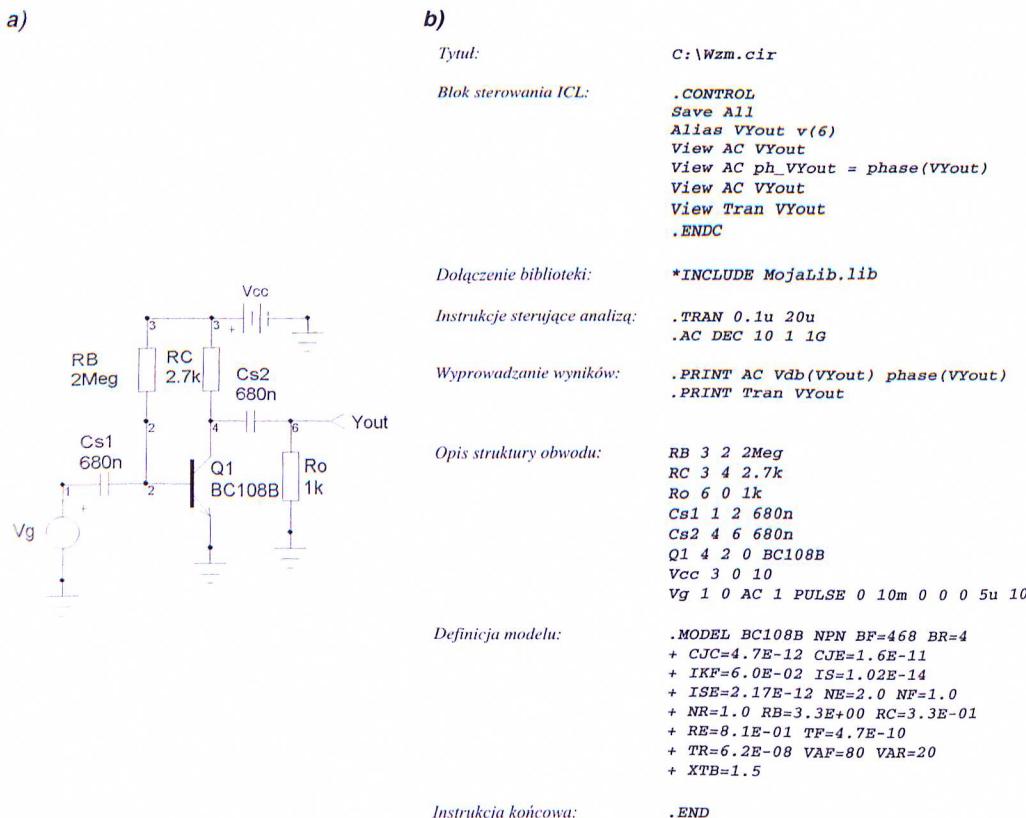
Analizowany obwód opisuje się, wymieniając wszystkie elementy wchodzące w jego skład z podaniem numerów węzłów, do których są przyłączone oraz wartości odpowiednich parametrów. Liczba węzłów i parametrów jest zależna od rodzaju wprowadzanego elementu. Każdy element musi być opisany w osobnej linii zbioru wejściowego. Elementy o złożonej strukturze, takie jak np. elementy półprzewodnikowe lub ferromagnetyczne, deklaruje się podając nazwę odpowiedniego modelu. Kolejność występowania deklaracji elementów i numeracji węzłów jest dowolna.

Przy tworzeniu pliku wejściowego należy pamiętać, że:

- układ musi zawierać węzeł o numerze „0” – najczęściej przyjmuje się, że jest to węzeł masy;
- numery węzłów muszą być liczbami naturalnymi;
- nazwa danego elementu powinna rozpoczynać się odpowiadającym mu symbolem^{9.1}, np. dla tranzystora bipolarnego musi to być *Q*. Wpisanie w tym przypadku np. nazwy *T2* spowoduje wygenerowanie błędu, gdyż symulator będzie oczekiwał struktury linii transmisyjnej – pierwsza litera „*T*”;

^{9.1} Program odczytuje osiem pierwszych znaków nazwy (nie biorąc, przy rozróżnianiu nazw elementów, dalszych znaków pod uwagę), ale nazwa może być oczywiście krótsza, kurecząc się w skrajnym przypadku do jednej, charakterystycznej litery.

- nie mogą wystąpić dwa elementy o takiej samej nazwie, np. dwa rezystory o nazwie $R1$;
- każda instrukcja sterująca, deklaracja elementu itp. niemieszcząca się w jednej linii pliku źródłowego, może być kontynuowana w linii następnej. Powinna się ona wtedy rozpoczynać znakiem kontynuacji linii „+” (patrz **rysunek 9.1**);
- linie kontynuacji (oznaczone „+”) muszą następować po liniach, których są kontynuującą;
- nie może wystąpić węzłów, do którego jest podłączony tylko jeden element^{9.2}. Ponadto należy mieć na uwadze ograniczenia wynikające z zastosowanej w programie SPICE zmodyfikowanej metody węzłowej (patrz rozdział 2), tj.:
 - układ nie może zawierać oczek składających się jedynie ze źródeł napięciowych i indukcyjności;
 - każdy węzeł musi mieć stałoprądowe połączenie z węzłem o numerze „0”, co jest równoważne warunkowi, że układ nie może zawierać rozcięć składających się jedynie ze źródeł napięciowych i indukcyjności;



Rys. 9.1. Przykład układu i pliku opisującego prosty wzmacniacz RC

9.2 Od tej zasady są wyjątki związane z linią transmisyjną i węzłem podłożem w tranzystorach bipolarnych i polowych z izolowaną bramką.

się jedynie ze źródeł prądowych i pojemności – program generuje wówczas błąd sygnalizujący brak stałoprądowego przejścia do masy.

Dozwolone jest natomiast:

- stosowanie numerów węzłów niebędących kolejnymi liczbami;
- umieszczanie w dowolnych liniach komentarza rozpoczynającego się od gwiazdki (*)^{9.3}.

Zbiór wejściowy do symulacji musi zawierać następujące elementy:

- linię tytułową (nawet pustą) jako pierwszą linię zbioru;
- opis topologii obwodu i instrukcje analiz;
- instrukcję .END jako końcową linię zbioru.

Zwykle w opisie obwodu, między linią tytułową i końcową, są umieszczone:

- blok komend *ICL* sterujących analizą;
- instrukcje dołączające wymagane biblioteki;
- instrukcje sterujące analizą i wyprowadzaniem wyników obliczeń;
- opis struktury obwodu;
- definicje modeli elementów i podobwodów, które występują w strukturze obwodu.

Przykładowy zbiór wejściowy, opisujący prosty wzmacniacz *RC* pokazany na rysunku 9.1a, z zaznaczonymi poszczególnymi blokami instrukcji, jest przedstawiony na rysunku 9.1b.

Plik *.cir generowany automatycznie przez edytor schematów i szablony symulacji może różnić się w szczegółach od przedstawionego powyżej, gdyż każdy producent dostosowuje fragmenty pliku *.cir do swojej implementacji programu SPICE.

W plikach opisu układu i innych plikach skryptowych duże i małe litery nie są rozróżniane. W przypadku bezpośredniej modyfikacji pliku *.cir należy więc stosować zapis charakteryzujący się jak najlepszą czytelnością, w miarę potrzeb opatrzony komentarzami.

9.2. Zasady opisu behawioralnego

Modelowanie behawioralne układów analogowych *ABM* (ang. *Analog Behavioral Modeling*) polega na zastosowaniu źródeł i elementów o parametrach (np. napięcie źródłowe, indukcyjność itp.) uzależnionych od zmiennych występujących w obwodzie (takich jak np.: różnica potencjałów, prąd, czas, częstotliwość, temperatura). Podstawą modelowania behawioralnego są wyrażenia algebraiczne, logiczne i warunkowe umożliwiające opisanie „zachowania się” układu. Zestaw najważniejszych operatorów i funkcji matematycznych stosowany w wyrażeniach *ABM* jest przedstawiony w **tablicy 9.1**.

W wyrażeniach matematycznych dostępne są trzy globalne zmienne związane z realizowanymi analizami, są to:

^{9.3} Gwiazdka jest często wykorzystywana w celu wywołania poleceń dodatkowych nierozpoczwalanych bezpośrednio przez jądro programu SPICE. Przykładowo instrukcja *Include powoduje, że zewnętrzny interpreter dołącza do pliku *.cir wymagane elementy biblioteczne (patrz rysunek 9.1 i rozdział 9.7.5). Sam symulator instrukcję *Include ignoruje.

- *Time* – bieżący czas symulacji w [s], możliwy do wykorzystania w trakcie analizy czasowej .Tran;
- *Freq* – bieżąca częstotliwość w [rad/s] – nie w [Hz] (!), możliwa do wykorzystania w trakcie małosygnałowej analizy częstotliwościowej .AC;
- *Temp* – bieżąca temperatura globalna analizowanego układu w [°C], dostępna w trakcie wszystkich analiz.

Przykłady:

```
V=2.0*V(1)^0.5+3*V(2)*Time+V(2)*Sqrt(Temp)
I=Freq*2e-9
R=100+2k*Int(Time)
C=0.1*Randc(5)
L=V(2)*Rand(1.25)
```

Tab. 9.1. Wybrane operatory i funkcje wykorzystywane do modelowania behawioralnego

Lp.	Nazwa operatora lub funkcji w programie SPICE	Opis
Operatory algebraiczne		
1	+	operator dodawania
2	-	operator odejmowania
3	*	operator mnożenia
4	/	operator dzielenia
5	^	operator potęgowania
Operatory logiczne		
1	&	operator And
2		operator Or
3	~	operator Not
Funkcje matematyczne		
1	ABS(x)	$ x $
2	SQRT(x)	\sqrt{x}
3	SGN(x)	$\text{sgn}(x), \pm 1$
4	EXP(x)	e^x
5	EXPL(x, L)	e^x z limitem L
6	LN(x)	$\ln(x)$; podstawa e
7	LOG(x)	$\log(x)$; podstawa 10
8	CEIL(x)	Najmniejsza liczba całkowita nie mniejsza niż x
9	FLOOR(x)	Największa liczba całkowita nie większa niż x
10	FRAC(x)	Część ułamkowa liczby x
11	INT(x)	Część całkowita liczby x
12	MAX(x, y)	Funkcja równa większemu z argumentów
13	MIN(x, y)	Funkcja równa mniejszemu z argumentów
14	PWR(x, y)	$ x ^y$
15	PWRS(x, y)	$\text{sgn}(x) x ^y$
16	RAND(X)	Liczba losowa o rozkładzie równomiernym z przedziału $<0..x>$
17	RANDC(X)	j. w. – tylko podstawa losowana jest jednokrotnie
18	SINC(X)	$\sin(x)/x$
19	STP(X)	$1(x)$

Tab. 9.1. cd.

Funkcje zmiennej zespolonej		
1	MAG (z)	z
2	PHS (z)	Arg (z) w [°]
3	REAL (z)	Re {z}
4	IMAG (z)	Im {z}

Funkcje trygonometryczne		
Argumenty i wartości kątów wyrażone są w radianach		
1	SIN (x)	sin (x)
2	ASIN (x)	arc sin (x)
3	SINH (x)	sinh (x)
4	ASINH (x)	ar sinh (x)
5	COS (x)	cos (x)
6	ACOS (x)	arc cos (x)
7	COSH (x)	cosh (x)
8	ACOSH (x)	ar cosh (x)
9	TAN (x)	tg (x)
10	ATAN (x)	arc tg (x)
11	TANH (x)	tgh (x)
12	ATANH (x)	ar tgh (x)

W zakresie instrukcji warunkowych dostępne są instrukcje: *If*, *Then* i *Else*. Struktura typowego wyrażenia, na przykładzie źródła behawioralnego (patrz rozdział 9.5.6), jest następująca

B1 n+ n- V=Warunek ? [Wart1] lub [Wyraż1] : [Wart2] lub [Wyraż2]

Tego typu instrukcję interpretuje się następująco: jeżeli warunek jest prawdziwy, to wydajność napięciowa źródła behawioralnego, czyli napięcie między węzłami $n+$ i $n-$, jest równa wartości *Wart1* lub wynikowi obliczeń wyrażenia *Wyraż1*, w przeciwnym przypadku jest równa *Wart2* (lub *Wyraż2*).

W powyższych instrukcjach:

- *Warunek* – jest wyrażeniem typu $x > y$ lub $x < y$ (niedopuszczalny jest znak „=”),
- *Wart1/2* – oznacza wartość liczbową,
- *Wyraż1/2* – jest wyrażeniem algebraicznym, logicznym lub mieszanym,
- ? – zastępuje słowo *Then*,
- : – zastępuje słowo *Else*.

Znaki „?” i „:” muszą być obustronnie oddzielone spacjami od sąsiadującego tekstu.

Przykład (ogranicznik napięcia):

B1 2 0 V=V(1) < .5 ? V(1)*.5+.25 : V(1) > 1.53 ? 1.54 : V(1)

W tym przykładzie źródło behawioralne *B1* pełni rolę źródła napięciowego o wydajności *V*, włączonego między węzeł 2 i masę.

Jeśli potencjał węzła 1 *V(1)* jest mniejszy niż 0,5 V, to

$$V = V(1) \cdot 0,5 + 0,25 \text{ V}$$

Jeśli potencjał węzła 1 $V(1)$ jest większy niż 1,53 V, to

$$V = 1,54 \text{ V}$$

w pozostałych przypadkach

$$V = V(1)$$

W zakresie boolowskich instrukcji logicznych można wykorzystywać operatory wymienione w drugiej sekcji tablicy 9.1. W tego typu wyrażeniach wykorzystuje się parametry ustawiane dyrektywą .OPTIONS, są to *Lone* (poziom jedynki, ang. *Level one*), *Lzero* (poziom zera, ang. *Level zero*), *Lthresh* (poziom progu, ang. *Level threshold*). Domyślnie:

$$\text{Lone} = 3,5$$

$$\text{Lzero} = 0,3$$

$$\text{Lthresh} = 1,5$$

W celu zasymulowania np. idealnych układów logicznych standardu TTL (ang. *Transistor-Transistor Logic*) należy zastosować dyrektywę

```
.OPTIONS LONE=5 LZERO=0 LTHRESH=2.5
```

Jeżeli wynik warunku jest większy niż próg (*Lthresh*), to wynik równania boolowskiego przyjmuje wartość odpowiadającą logicznej „jedynce” (*Lone*). W przeciwnym przypadku jest podstawiana wartość równa *Lzero*.

Przykład (prosty układ kombinacyjny – bramka NAND):

```
B1 3 0 V=~(V(1)&V(2))
```

Jeśli potencjały węzła 1 i 2 będą jednocześnie większe od napięcia progowego, to wydajność napięciowa źródła behawioralnego *B1* (w tym przypadku potencjał węzła 3) będzie równa wartości odpowiadającej logicznemu zeru. W każdym innym przypadku wyniesie *Lone*.

Przy wprowadzaniu wielkości liczbowych można skorzystać z trzech wariantów: zastosować tzw. *zapis zmiennoprzecinkowy* (ang. *Floating Point Notation*), np. „1234.5”, zapis w stałoprzecinkowym *formacie naukowym* (ang. *Scientific Notation*) – „1.2345e3” lub wykorzystać tzw. *zapis inżynierski* (ang. *Engineering Notation*) – „1.2345k”, oparty na przedrostkach skalujących przedstawionych w **tablicy 9.2**^{9.4}.

Tab. 9.2. Przedrostki skalujące stosowane w programie SPICE

Symbol	Nazwa	Krotność
T	tera	$\cdot 10^{12}$
G	giga	$\cdot 10^9$
Meg	mega	$\cdot 10^6$
k	kilo	$\cdot 10^3$
m	milli	$\cdot 10^{-3}$
u	mikro	$\cdot 10^{-6}$
n	nano	$\cdot 10^{-9}$
p	piko	$\cdot 10^{-12}$
f	femto	$\cdot 10^{-15}$

^{9.4} Należy pamiętać, że separatorem dziesiętnym – podobnie jak we wszystkich językach programowania – jest kropka.

Podawanie jednostek nie jest konieczne (ale jest dozwolone), gdyż rodzaj wprowadzanej wielkości jednoznacznie definiuje jednostkę^{9.5}. Gdyby przykładowa liczba podawana wyżej dotyczyła np. wartości rezystancji, system odczytałby ją jako 1,2345 kΩ.

9.3. Deklaracje elementów biernych

9.3.1. Rezystor

Deklaracja rezystora jest następująca^{9.6}:

```
Rnazwa n+ n- [Wartość]
              [R=Wyrażenie]
              [Model [L=Długość [W=Szerokość]]] [Temp=t]
```

gdzie:

n+, n- – numery węzłów,

Wartość – wartość rezystancji (musi być różna od zera),

Wyrażenie – wyrażenie zgodne z zasadami modelowania behawioralnego,

Model – nazwa modelu rezystora,

Długość, Szerokość – wymiary geometryczne rezystora półprzewodnikowego,

t – temperatura rezystora.

Deklaracja rezystora poprzez *Model* umożliwia opis rezystora półprzewodnikowego za pomocą gabarytów (*Długość, Szerokość*) lub modelowanie wpływu temperatury (patrz rozdział 4.2.1).

Przykłady:

```
R1 1 2 10k
Rg 3 4 R=1k+1k*.Sqrt(Time)+5*Temp ← wartość zależna od czasu i temperatury
Rp 5 6 100 RMODEL L=10u W=1u
```

9.3.2. Kondensator

Kondensator deklarowany jest wg składni:

```
Cnazwa n+ n- [Wartość]
              [C=Wyrażenie]
              [Model L=Długość [W=Szerokość]] [IC=V]
```

gdzie:

n+, n- – numery węzłów,

Wartość – wartość pojemności,

Wyrażenie – wyrażenie zgodne z zasadami modelowania behawioralnego,

Model – nazwa modelu kondensatora,

Długość, Szerokość – wymiary geometryczne kondensatora półprzewodnikowego,

^{9.5} Domyslnymi jednostkami są jednostki z układu SI (Międzynarodowy Układ Jednostek Miar – fr. *Système International d'Unités*).

^{9.6} We wszystkich deklaracjach nawiasy kwadratowe [] identyfikują parametry opcjonalne.

V – napięcie początkowe skierowane od $n-$ do $n+$, uwzględniane, gdy parametr UIC jest włączony w deklaracji analizy czasowej .Tran (patrz rozdział 7.3). Deklaracja kondensatora poprzez *Model* łącznie z jego wymiarami geometrycznymi umożliwia zamodelowanie kondensatora półprzewodnikowego.

Przykłady:

```
C1 1 2 1u
C2 3 4 CMODEL L=10u W=1u
Ci 5 6 .01uF IC=10V ← z ustalonimi warunkami początkowymi
Cz 7 8 C=1u+1p*Freq^2 ← pojemność zależna od częstotliwości
```

9.3.3. Cewka

Deklaracja cewki ma postać:

```
Lnazwa n+ n- [Wartość]
                [L=Wyrażenie] [IC=I]
```

gdzie:

$n+, n-$ – numery węzłów,

Wartość – wartość indukcyjności,

Wyrażenie – wyrażenie zgodne z zasadami modelowania behawioralnego,

I – prąd początkowy płynący przez cewkę od $n+$ do $n-$, uwzględniany, gdy parametr UIC jest włączony w deklaracji analizy czasowej .Tran (patrz rozdział 7.3).

Przykłady:

```
Lo 1 7 2u
L1 8 0 .01mH IC=10mA ← z ustalonimi warunkami początkowymi
Lm 1 0 L=1u+2*V(3)+I(R1) ← indukcyjność sterowana prądem i napięciem
```

9.3.4. Indukcyjność wzajemna

Indukcyjność wzajemną deklaruje się poprzez współczynnik sprzężenia między zadeklarowanymi wcześniej cewkami, wg składni:

```
Knazwa1 Knazwa2 Wartość
```

gdzie:

Knazwa1 i *Knazwa2* – nazwy sprzężonych indukcyjnie cewek,

Wartość – współczynnik sprzężenia należący do przedziału otwartego (0...1).

Indukcyjność wzajemna jest określona wzorem

$$M = \text{Wartość} \cdot \text{Knazwa1} \cdot \text{Knazwa2} \quad (9.1)$$

Przykład:

```
K12 L1 L2 .9999
```

9.3.5. Klucz sterowany

Deklaracje klucza sterowanego prądem i napięciem mają postać odpowiednio:

```
Vnazwa n+ n- Vnazwa Model [On] [Off]
Snazwa n+ n- nc+ nc- Model [On] [Off]
```

gdzie:

- $n+$, $n-$ – numery węzłów łączonych kluczem,
- $nc+$, $nc-$ – numery węzłów, do których jest przyłączone napięcie sterujące,
- V_{nazwa} – nazwa źródła napięciowego, przez które przepływa prąd sterujący,
- $Model$ – nazwa modelu klucza,
- On/Off – przełącznik określający stan klucza przy obliczaniu stałoprądowego punktu pracy.

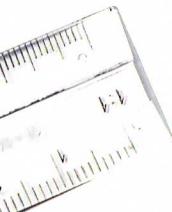
$Model$ umożliwia m.in. uwzględnienie napięć włączenia i wyłączenia klucza oraz rezystancji klucza w stanie zamkniętym i otwartym.

Przykłady:

```
W1 1 2 VCLOCK SWITCH
W2 3 0 VRAMP SM1 On
S1 1 2 3 4 Switch1 On
S2 5 6 7 8 SM2 Off
```

9.4. Deklaracje elementów półprzewodnikowych

9.4.1. Dioda



Deklaracja diody półprzewodnikowej jest następująca:

```
Dnazwa na nc Model [Area] [Off] [IC=Vd] [Temp=t]
```

gdzie:

- na , nc – numery węzłów odpowiednio anody i katody,
- $Model$ – nazwa modelu diody,
- $Area$ – współczynnik zwielokrotnienia, określający ile razy pole powierzchni zajmowanej przez element jest większe od pola przyjętego dla modelu. Pominiecie współczynnika $Area$ oznacza, że ma on wartość równą jeden,
- Off – opcjonalny warunek początkowy stosowany w celu poprawy zbieżności podczas analizy wyznaczającej stałoprądowy punkt pracy,
- Vd – warunki początkowe (napięcie anoda-katoda), uwzględniane, gdy parametr UIC jest włączony w deklaracji analizy czasowej .Tran (patrz rozdział 7.3),
- t – temperatura diody.

Przykłady:

```
D1 2 4 1N4001 Off
D2 3 7 DMODEL 3 IC=0.2V Temp=50
```

9.4.2. Tranzystor bipolarny

Deklaracja tranzystora bipolarnego (*BJT* – ang. *Bipolar Junction Transistor*) ma postać:

```
Qnazwa nc nb ne [ns] Model [Area] [Off] [IC=Vbe, Vce] [Temp=t]
```

gdzie:

nc, nb, ne, ns – numery węzłów kolektora, bazy, emitera i podłoża (ang. *collector, base, emitter, substrate*),

Model – nazwa modelu tranzystora,

Area – analogicznie jak dla diody,

Off – analogicznie jak dla diody,

Vbe, Vce – warunki początkowe (napięcia baza-emiter i kolektor-emiter), uwzględniane, gdy parametr *UIC* jest włączony w deklaracji analizy czasowej .Tran (patrz rozdział 7.3),

t – temperatura tranzystora.

Przykłady:

```
Q1 4 7 0 BC108B
Q2 1 2 4 QMOD IC=0.64, 5.0
```

Przykładowy model dostępny w bibliotece ma postać:

```
.MODEL BC108B NPN (IS=1.02E-14 NF=1.0 BF=468 VAF=80
+ IKF=6.0E-02 ISE=2.17E-12 NE=2.0 BR=4 NR=1.0 VAR=20
+ XTB=1.5 RE=8.1E-01 RB=3.3E+00 RC=3.3E-01
+ CJE=1.6E-11 CJC=4.7E-12 TF=4.7E-10 TR=6.2E-08)
```

Znaczenie poszczególnych parametrów modelu^{9.7} przedstawiono w tablicy 4.1. Jeśli w definicji modelu wyekstrahowanej z pliku bibliotecznego *.lib pewne parametry nie występują, oznacza to, że mają one wartości domyślne.

9.4.3. Tranzystor polowy JFET

Deklaracja złączowego tranzystora polowego *JFET* (ang. *Junction Field-Effect Transistor*) ma postać:

```
Jnazwa nd ng ns Model [Area] [Off] [IC=Vds, Vgs] [Temp=t]
```

gdzie:

nd, ng, ns – numery węzłów drenu, bramki, źródła (ang. *drain, gate, source*),

Model – nazwa modelu tranzystora,

Area – analogicznie jak dla diody,

Off – analogicznie jak dla diody,

Vds, Vgs – warunki początkowe (napięcia dren-źródło i bramka-źródło), uwzględniane, gdy parametr *UIC* jest włączony w deklaracji analizy czasowej .Tran (patrz rozdział 7.3),

t – temperatura tranzystora.

Przykłady:

```
J1 1 2 4 BF245A
J2 7 2 3 JM1 Off
```

Program SPICE oferuje dwa różne modele tranzystora polowego złączowego, są to: *model „Berkeley SPICE 3F”* oparty na klasycznym *modelu Shichmana-Hodgesa* oraz *model Parkera-Skellerna* opracowany na *Macquarie University* w Sydney

^{9.7} Tranzystor bipolarny jest opisany modelem Gummela-Poona.

(Australia), zawierający wiele dodatkowych parametrów wyraźnie poprawiających dokładność analizy.

9.4.4. Tranzystor polowy MOSFET

Deklaracja tranzystora polowego z izolowaną bramką *MOSFET* (ang. *Metal Oxide Semiconductor FET*) ma postać:

Mnazwa nd ng ns nb Model
 + [L=l] [W=w] [AD=a] [AS=b] [PD=c] [PS=d] [NRD=e] [NRS=f]
 + [Off] [IC=Vds, Vgs, Vbs] [Temp=t]

gdzie:

nd, ng, ns, nb – numery węzłów drenu, bramki, źródła i podłoża (ang. *base*),

Model – nazwa modelu tranzystora,

l, w, a, b, c, d, e, f – parametry geometryczne tranzystora,

Off – analogicznie jak dla diody,

Vds, Vgs, Vbs – warunki początkowe (napięcia dren-źródło, bramka-źródło i podłoże-źródło), uwzględniane, gdy parametr *UIC* jest włączony w deklaracji analizy czasowej *.Tran* (patrz rozdział 7.3),

t – temperatura tranzystora.

Przykłady:

M1 1 2 3 4 VN0603L

M2 5 6 7 8 MODM L=10u W=5u AD=100p AS=100p PD=40u PS=40u

W przypadku tranzystora *MOSFET* dysponujemy największą liczbą modeli. Są to modele określane za pomocą tzw. *poziomów* (ang. *levels*): od *Level 1* (klasyczny *model Shichmana-Hodgesa*) poprzez półempiryczny *Level 3* do *Level 8* (*BSIM3 v. 3.1*)^{9.8}. Stopień złożoności modelu jest związany bezpośrednio z dokładnością i czasem trwania symulacji. Producenci branży *IC* (ang. *Integrated Circuits*) podają z reguły kilka modeli dla każdego swojego elementu.

9.4.5. Tranzystor polowy MESFET

W programie SPICE deklaracja tranzystora polowego ze złączem metal-półprzewodnik *MESFET* (ang. *Metal-Semiconductor FET*) ma postać:

Znazwa nd ng ns Model [Area] [Off] [IC=Vds, Vgs]

gdzie:

nd, ng, ns – numery węzłów drenu, bramki, źródła,

Model – nazwa modelu tranzystora,

Area – analogicznie jak dla diody,

Off – analogicznie jak dla diody,

^{9.8} W wersji *IsSpice 4* zaimplementowano dodatkowo najnowszy i najdokładniejszy – opracowany w języku *AHDL* (ang. *Analog Hardware Description Language*) – model, o pełnej nazwie *Fully Depleted SOI MOSFET C Code Model*.

V_{ds} , V_{gs} – warunki początkowe (napięcia dren-źródło i bramka-źródło), uwzględniane, gdy włączony jest parametr UIC w deklaracji analizy czasowej .Tran (patrz rozdział 7.3).

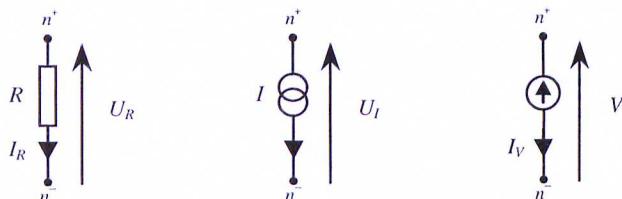
Przykłady:

```
Z1 1 2 3 NE760
Z2 4 5 6 ZM2 OFF
```

W przypadku tranzystora *MESFET* dostępne są cztery modele o rosnącym stopniu złożoności, kolejno od poziomu *Level 1 (model Statza)* do poziomu *Level 4 (HMET 2 Model)*.

9.5. Deklaracje źródeł

Jak wspomniałem w rozdziale 2.3, dla wszystkich elementów obowiązuje ogólna zasada, że **przez element prąd płynie od węzła n^+ do węzła n^- , strzałka napięcia wskazuje zaś węzeł n^+** . A zatem, w przypadku źródła napięciowego węzeł n^+ identyfikuje wyższy potencjał – strzałka napięcia skierowana jest od n^- do n^+ . W przypadku źródła prądowego prąd płynie od węzła n^+ do węzła n^- przez źródło (w obwodzie zewnętrznym od n^- do n^+). Przedstawioną zasadę ilustruje **rysunek 9.2**.



Rys. 9.2. Zasady strzałkowania prądów i napięć w programie SPICE

9.5.1. Źródła niezależne

Deklaracje obu niezależnych źródeł są identyczne, za wyjątkiem pierwszej, kluczowej, litery nazwy. W przypadku niezależnego źródła prądowego jest to I , a w przypadku źródła napięciowego V . Deklaracja źródła niezależnego, na przykładzie częściej stosowanego źródła napięciowego, ma postać

```
Vnazwa n+ n-
[ [DC] Wartość ] ← dla analizy staloprądowej
[ [AC Amplituda [Faza]] ] ← dla analizy zmiennoprądowej
[ DistoFl [AmplitudaFl [FazaF1]] ] ← dla analizy zniekształceń
+ [ DistoF2 [AmplitudaF2 [FazaF2]] ]
[ PULSE v1 v2 [td [tr [tf [pw [per [phase]]]]]] ] ← dla analizy czasowej
lub [SIN vo va [freq [td [kd [phase]]]]]
lub [PWL tl v1 t2 v2 ... tn vn]
lub [SFFM vo va fc [mdi [fm [phase]]]]
lub [EXP v1 v2 [td1 [t1 [td2 [t2]]]]]
```

Przykłady:

```
VCC 1 0 5
VG1 2 0 AC 1
I12 3 4 DC 2 AC 1
Vin 5 0 DistoFl 10m 90 DistoF2 5m 45
IWE 6 7 AC 1 SIN 1m 0.5m 10k
```

Dla danego źródła napięciowego (lub prądowego) można zadeklarować jednocześnie kilka rodzajów wymuszeń, należy jednak pamiętać, że dla konkretnej analizy jest dopuszczalna deklaracja tylko jednego źródła określonego typu.

Jeśli wartość napięcia źródłowego ma być zerowa (źródło takie można wykorzystać do pomiaru prądu płynącego w gałęzi, w której jest umieszczone^{9.9}), to wówczas deklaracja kończy się na podaniu numerów węzłów.

Po określeniu numerów węzłów podajemy, dla jakiej analizy deklarujemy źródło. Oznaczenie *DC* można pominąć, wystarczy podać jedynie wartość napięcia, np. w przypadku deklaracji źródła stałoprądowego o napięciu źródłowym 0,6 V, zapis

`Vcc 1 0 .6`

jest równoważny deklaracji

`Vcc 1 0 DC 0.6`

Podczas analizy zmiennoprądowej źródło *AC* musi mieć określona amplitudę. Jeśli nie zadeklarujemy fazy początkowej (w stopniach), to symulator domyślnie przyjmie wartość zerową.

Dla źródeł stosowanych podczas analizy zniekształceń musi być podana amplituda sygnału *DistoF1*. Jeśli nie podano amplitudy sygnału *DistoF2*, a w deklaracji analizy *.Disto* wystąpi stosunek f_2/f_1 (co sygnalizuje, że do wejścia układu podłączone są dwa sygnały), to symulator przyjmie, że amplituda sygnału *DistoF2* jest równa amplitudzie sygnału *DistoF1*. Fazy początkowe obu sygnałów *Disto* są domyślnie równe零.

Pozostałe wymuszenia dotyczą analizy czasowej^{9.10}. Są to:

- sygnał impulsowy,
- tłumiony sygnał harmoniczny,
- sygnał odcinkowo-liniowy,
- sygnał zmodulowany częstotliwościowo pojedynczym tonem
- sygnał eksponentjalny.

Tab. 9.3. Parametry sygnału impulsowego PULSE

Parametry		Jednostka
<i>v1</i>	Wartość początkowa (ang. <i>Initial Value</i>)	V, A
<i>v2</i>	Wartość „po skoku” (ang. <i>Pulsed Value</i>)	V, A
<i>td</i>	Czas opóźnienia (ang. <i>Delay Time</i>)	s
<i>tr</i>	Czas narastania (ang. <i>Rise Time</i>)	s
<i>tf</i>	Czas opadania (ang. <i>Fall Time</i>)	s
<i>pw</i>	Czas trwania impulsu (ang. <i>Pulse Width</i>)	s
<i>per</i>	Okres powtarzania (ang. <i>Period</i>)	s
<i>phase</i>	Opóźnienie fazowe (ang. <i>Phase Delay</i>)	°

^{9.9} Ponieważ, jak wcześniej wspomniałem, współczesne wersje programu SPICE zapamiętują wszystkie prądy i napięcia w analizowanym układzie, to do odczytania wartości prądu stosuje się bezpośrednio instrukcję *.Print* wstawianą automatycznie po umieszczeniu na schemacie sondy prądowej.

^{9.10} Wprowadzenie do programu SPICE źródeł behawioralnych daje praktycznie nieograniczone możliwości generowania dowolnych rodzajów wymuszeń dla analizy czasowej.

Tab. 9.4. Parametry tłumionego sygnału harmonicznego SIN

Parametry		Jednostka
<i>vo</i>	Wartość średnia (ang. Offset)	V, A
<i>va</i>	Maksymalna wartość amplitudy (ang. Peak Amplitude)	V, A
<i>freq</i>	Częstotliwość (ang. Frequency)	Hz
<i>td</i>	Czas opóźnienia (ang. Delay Time)	s
<i>kd</i>	Współczynnik tłumienia (ang. Damping Coefficient)	1/s
<i>phase</i>	Opóźnienie fazowe (ang. Phase Delay)	°

Współczesne wersje programu SPICE wyposażone są w graficzny interfejs użytkownika ułatwiający deklaracje sygnałów wymuszających dla analizy czasowej. Charakterystyka „źródeł czasowych” w programie *ICAP/4 Windows* przedstawiona jest poniżej.

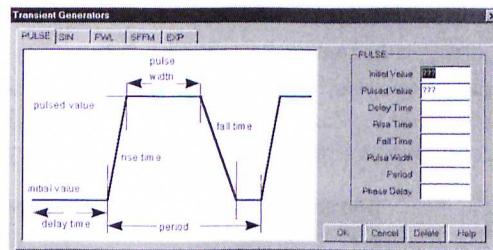
a) Sygnał impulsowy (*PULSE*)

```
PULSE v1 v2 [td [tr [tf [pw [per [phase]]]]]]]
```

Przykład:

```
v1 1 0 PULSE 0 1 100n 40n 90n 200n 390n
```

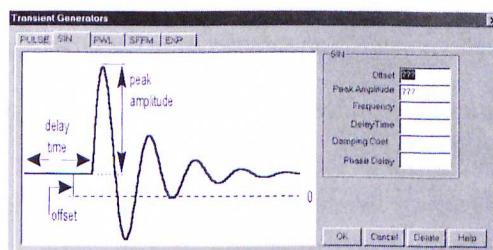
Parametry sygnału impulsowego *PULSE* podane są w tablicy 9.3, a jego kształt wraz z oknem dialogowym do ustawiania parametrów jest przedstawiony na rysunku 9.3.



Rys. 9.3. Graficzny interfejs użytkownika do wprowadzania sygnału impulsowego *PULSE*

b) Tłumiony sygnał harmoniczny (*SIN*)

```
SIN vo va [freq [td [kd [phase]]]]
```



Rys. 9.4. Graficzny interfejs użytkownika do wprowadzania tłumionego sygnału harmonicznego *SIN*

Przykład:

```
V1 1 0 SIN 5 4 10k 0.13m 1e3 45
```

Parametry tłumionego sygnału harmonicznego *SIN* są podane w **tablicy 9.4**, natomiast jego kształt wraz z oknem dialogowym do ustawiania parametrów jest przedstawiony na **rysunku 9.4**.

Zależność opisująca wartość chwilową tłumionego sygnału harmonicznego $v(t)$ ma postać

$$v(t) = vo + va \cdot \mathbf{1}(t - td) \cdot e^{-(t-td)kd} \cdot \sin[2\pi \cdot freq(t - td) + phase] \quad (9.2)$$

Jeżeli nie zadeklarujemy wartości średniej vo i współczynnika tłumienia kd ^{9.11}, to otrzymamy nietłumiony sygnał harmoniczny.

c) Sygnał odcinkowo-liniowy (PWL – ang. Piece-Wise Linear)

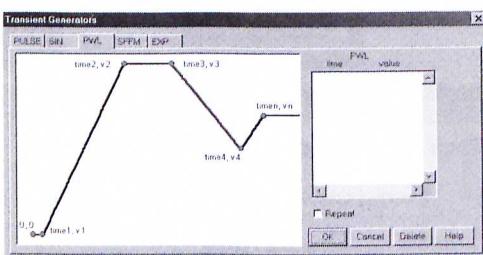
```
PWL t1 v1 t2 v2 ... tn vn [r]
```

Przykład:

```
V1 1 0 PWL 10n 0 100n 1 150n 1 225n .5 250n .7 r
```

Każda para (t_i, v_i) reprezentuje współrzędne punktów o wartościach napięć v_i w chwilach t_i . W deklaracji przebiegu można umieścić dowolną liczbę takich par. Przebieg sygnału w czasie między sąsiednimi wartościami t_i i t_{i+1} jest obliczany metodami interpolacji liniowej (punkty łączone są liniami prostymi). Przebieg czasowy sygnału odcinkowo-liniowego wraz z oknem dialogowym do ustawiania parametrów jest przedstawiony na **rysunku 9.5**.

Jeśli nie zaznaczono opcji *Repeat [r]*, to od chwili t_n do końca czasu analizy przebieg zachowuje wartość v_n . Zaznaczenie tej opcji umożliwia uzyskanie przebiegu okresowego o zdefiniowanym przez użytkownika – praktycznie dowolnym – kształcie^{9.12}.



Rys. 9.5. Graficzny interfejs użytkownika do wprowadzania sygnału odcinkowo-liniowego PWL

d) Sygnał zmodulowany częstotliwościowo pojedyńczym tonem (SFFM – ang. Single Frequency FM)

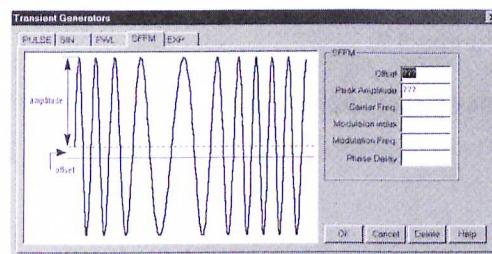
```
SFFM vo va fc [mdi [fm [phase]]]
```

^{9.11} Domyslnie $vo = 0$ i $kd = 0$.

^{9.12} Ścisłe rzecz ujmując, kształt sygnału PWL może być złożony jedynie z fragmentów linii prostych, lecz odpowiednio skracając odcinki składowe można praktycznie dowolną krzywiznę.

Tab. 9.5. Parametry sygnału zmodulowanego SFFM

Parametry		Jednostka
<i>vo</i>	Wartość średnia (ang. <i>Offset</i>)	V, A
<i>va</i>	Amplituda (ang. <i>Amplitude</i>) ^{9.13}	V, A
<i>fc</i>	Częstotliwość nośna (ang. <i>Carrier Frequency</i>)	Hz
<i>mdi</i>	Indeks modulacji (ang. <i>Modulation Index</i>)	—
<i>fm</i>	Częstotliwość modulująca (ang. <i>Modulation Frequency</i>)	Hz
<i>phase</i>	Opóźnienie fazowe (ang. <i>Phase Delay</i>)	°



Rys. 9.6. Graficzny interfejs użytkownika do wprowadzania sygnału zmodulowanego SFFM

Przykład:

```
v1 1 0 SFFM 0 10M 10.7MEG 3 1K 90
```

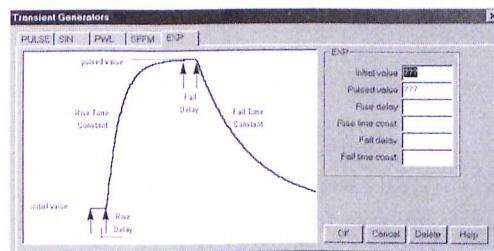
Parametry sygnału zmodulowanego *SFFM* są umieszczone w tablicy 9.5, a jego przebieg czasowy wraz z oknem dialogowym do ustawiania parametrów jest zaprezentowany na rysunku 9.6.

e) Sygnał eksponencjalny (EXP)

```
EXP v1 v2 [td1 [t1 [td2 [t2]]]]
```

Przykład:

```
v1 1 0 EXP 0 15m 0 25n 200n 25n
```



Rys. 9.7. Graficzny interfejs użytkownika do wprowadzania sygnału eksponencjalnego EXP

^{9.13}Na zrzucie ekranowym pokazanym na rys. 9.6 programista „z rozpudu” wprowadził błędą nazwę „Peak Amplitude” zamiast „Amplitude”.

Tab. 9.6. Parametry sygnału eksponencjalnego EXP

Parametry		Jednostka
v1	Wartość początkowa (ang. Initial Value)	V, A
v2	Wartość „po skoku” (ang. Pulsed Value)	V, A
td1	Czas opóźnienia zbocza narastającego (ang. Rise Delay)	s
t1	Stała czasowa zbocza narastającego (ang. Rise Time Constant)	s
td2	Czas opóźnienia zbocza opadającego (ang. Fall Delay)	s
t2	Stała czasowa zbocza opadającego (ang. Fall Time Constant)	s

Parametry sygnału eksponencjalnego EXP są umieszczone w **tablicy 9.6**, natomiast jego przebieg wraz z oknem dialogowym do ustawiania parametrów jest przedstawiony na **rysunku 9.7**.

9.5.2. Źródło prądowe sterowane prądem

Deklaracja źródła prądowego sterowanego prądem ma postać:

Fnazwa n+ n- Vnazwa k

gdzie:

n+, n- – węzły wyjściowe,

Vnazwa – źródło napięciowe dołączone do węzłów nc+ i nc-, przez które przepływa prąd sterujący,

k – transmitancja prądowa w [A/A].

Przykład:

F1 2 0 Vcc 100

Prąd wyjściowy jest obliczany z zależności (patrz rysunek 2.6)

$$I_{out} = k \cdot I_{in} \quad (9.3)$$

gdzie:

I_{out} – prąd płynący w obwodzie zewnętrznym od węzła n- do n+,

I_{in} – prąd płynący przez źródło Vnazwa od węzła nc+ do nc-.

9.5.3. Źródło prądowe sterowane napięciem

Deklaracja źródła prądowego sterowanego napięciem ma postać:

Gnazwa n+ n- nc+ nc- g

gdzie:

n+, n- – węzły wyjściowe,

nc+, nc- – węzły wejściowe,

g – transkonduktancja w [S].

Przykład:

G1 3 4 1 2 1ms

Prąd wyjściowy jest obliczany z zależności (patrz rysunek 2.7)

$$I_{out} = g \cdot U_{in} \quad (9.4)$$

gdzie:

- I_{out} – prąd płynący w obwodzie zewnętrznym od węzła $n-$ do $n+$,
- U_{in} – napięcie między węzłami $nc+$ i $nc-$.

9.5.4. Źródło napięciowe sterowane prądem

Deklaracja źródła napięciowego sterowanego prądem ma postać:

Hnazwa $n+$ $n-$ Vnazwa r

gdzie:

- $n+, n-$ – węzły wyjściowe,
- $Vnazwa$ – źródło napięciowe dołączone do węzłów $nc+$ i $nc-$, przez które przepływa prąd sterujący,
- r – transrezystancja w $[\Omega]$.

Przykład:

H1 2 0 Vg 1.5k

Napięcie wyjściowe jest obliczane z zależności (patrz rysunek 2.8)

$$U_{out} = r \cdot I_{in} \quad (9.5)$$

gdzie:

- U_{out} – napięcie między węzłami $n+$ i $n-$,
- I_{in} – prąd płynący przez źródło $Vnazwa$ od węzła $nc+$ do $nc-$.

9.5.5. Źródło napięciowe sterowane napięciem

Deklaracja źródła napięciowego sterowanego napięciem ma postać:

Enazwa $n+$ $n-$ $nc+$ $nc-$ k

gdzie:

- $n+, n-$ – węzły wyjściowe,
- $nc+, nc-$ – węzły wejściowe,
- k – transmitancja napięciowa w $[V/V]$.

Przykład:

E1 3 4 2 1 50

Napięcie wyjściowe jest obliczane z zależności (patrz rysunek 2.9)

$$U_{out} = k \cdot U_{in} \quad (9.6)$$

gdzie:

- U_{out} – napięcie między węzłami $n+$ i $n-$,
- U_{in} – napięcie między węzłami $nc+$ i $nc-$.

9.5.6. Źródło behawioralne

Deklaracja źródła behawioralnego ma postać:

Bnazwa $n+$ $n-$ [I=Wyrażenie] [V=Wyrażenie]

gdzie:

$n+$, $n-$ – węzły wyjściowe,

Wyrażenie – wyrażenie zgodne z zasadami modelowania behawioralnego.

Instrukcja ta służy do deklaracji zarówno źródeł prądowych, jak i napięciowych. O rodzaju źródła decyduje użyty parametr: I zdeterminuje typ źródła jako prądowe, a V jako napięciowe. Jednocześnie można użyć tylko jednego z tych parametrów.

Przykłady:

```
B1 1 0 V=ln(cos((V(1,2)^2))) ← źródło napięciowe
B2 2 0 I=cos(V(1))+sin(V(2)) ← źródło prądowe
```

9.6. Deklaracje analiz

9.6.1. Stałoprądowa analiza punku pracy

Statyczny punkt pracy jest automatycznie wyznaczany przed uruchomieniem wszystkich pozostałych analiz, jednakże w celu uzyskania szczegółowych informacji o punkcie pracy należy zastosować instrukcję

.OP

Umieszczenie tej instrukcji w pliku wejściowym powoduje wylistowanie parametrów małosygnałowych modeli elementów półprzewodnikowych i wydajności nienieliniowych źródeł sterowanych w punkcie pracy. Wyznaczone parametry umieszczone są w pliku wyjściowym *.out. W przypadku pominięcia instrukcji .OP nie jest możliwe odczytanie w inny sposób wartości tych parametrów, chociaż punkt pracy i tak zostanie wyznaczony. Ponadto instrukcja ta powoduje umieszczenie w standaryzowanym zbiorze wynikowym prądów i rozpraszanej mocy dla wszystkich występujących w układzie elementów.

Znacznie więcej informacji o punkcie pracy układu można uzyskać, korzystając z komend *ICL* (*op* i *show* lub *showmod*). Odpowiedni blok komend *ICL* może mieć postać (patrz rozdział 9.8):

```
.control
op
show all
.endc
```

9.6.2. Krokowa analiza stałoprądowa

Analiza stałoprądowa umożliwia uzyskanie rodzin charakterystyk statycznych badanego układu. Po wyznaczeniu stałoprądowego punktu pracy wydajności zadeklarowanych do analizy źródeł prądów i napięć stałych są odpowiednio zmieniane i każdorazowo jest przeprowadzana analiza stałoprądowa (patrz rozdział 5).

Instrukcja sterująca ma postać

```
.DC Źródło1 Start1 Stop1 Kroki1 [Źródło2 Start2 Stop2 Kroki2]
```

gdzie:

Źródło1/2 – niezależne źródła napięciowe lub prądowe, których wydajności są zmieniane w trakcie analizy,

StartI/2, StopI/2 – wartości początkowe i końcowe, określające zakresy zmian wydajności źródeł *Źródło1* i *Źródło2*,
KrokI/2 – przyrosty wydajności źródeł *Źródło1* i *Źródło2*.

Przykłady:

```
.DC VIN 2.25 5.0 0.25
.DC VDS 0 10 .5 VGS 0 5 1
.DC VCE 0 10 .25 IB 0 100 1U
```

W przypadku zastosowania drugiego źródła *Źródło2* deklaracja *.DC* umożliwia otrzymanie rodziny charakterystyk statycznych (np. wyjściowych tranzystora bipolarnego). W tym przypadku drugie źródło zmienia swoją wydajność w ustalonym zakresie, dla kolejnych wydajności źródła pierwszego.

Deklaracja *.DC* musi współlistnieć z deklaracją *.Print* zapisującą do pliku wyjściowego **.out* wyniki analizy w wyspecyfikowanych punktach układu (patrz rozdział 9.7.3).

9.6.3.

Małosygnalowa analiza zmiennoprądowa

Małosygnalowa analiza zmiennoprądowa polega na obliczeniu wszystkich prądów i napięć w stanie ustalonym w układzie, który jest pobudzany wymuszeniem zmiennoprądowym typu *AC* (patrz rozdział 9.5.1). W trakcie tej analizy obliczane są amplitudowe i fazowe charakterystyki układu.

Deklaracja analizy zmiennoprądowej ma postać

```
.AC Rodzaj_skali Ilość_punktów fStart fStop
```

Znaczenie wszystkich parametrów podano w rozdziale 6.1.

Przykłady:

```
.AC DEC 10 1 10K
.AC OCT 10 1K 100MEG
.AC LIN 100 1 100
```

Gdy układ zawiera tylko jedno wejściowe źródło zmiennoprądowe, dogodnie jest przyjąć dla niego jednostkową amplitudę i zerową fazę początkową. Wielkość wyjściowa wskazana w deklaracji *.Print* jest wówczas liczbowo równa odpowiedniej immitancji układu.

Deklaracja *.AC* musi współlistnieć z deklaracją *.Print* zapisującą wyniki analizy do pliku wyjściowego **.out* (patrz rozdział 9.7.3).

9.6.4.

Małosygnalowa analiza zniekształceń nieliniowych

Analiza zniekształceń nieliniowych jest przeprowadzana przy założeniu, że do wejścia układu doprowadza się pojedynczy sygnał harmoniczny (*DistoF1*) lub sumę dwóch sygnałów harmonicznych (*DistoF1 + DistoF2*) określonych w deklaracjach źródeł (patrz rozdział 9.5.1).

Wykorzystując uproszczone wielkosygnalowe modele elementów (patrz rozdział 6.2) i zmieniając częstotliwość podstawową *f₁* w zadanym zakresie i z założonym krokiem, program SPICE, z wykorzystaniem odpowiednich wyrażeń dla każdej gałęzi nieliniowej, wyznacza drugą i trzecią harmoniczną sygnału podstawowego, tj.

$2f_1$ i $3f_1$, jeśli zadeklarowano tylko częstotliwość f_1 lub składowe intermodulacyjne drugiego i trzeciego rzędu: f_1+f_2 , f_1-f_2 i $2f_1-f_2$, gdy zadeklarowano ponadto częstotliwość f_2 .

Deklaracja analizy zniekształceń nieliniowych ma postać

```
.Distro Rodzaj_skali Ilość_punktów fStart fStop [f2/f1]
```

Znaczenie wszystkich parametrów podano w rozdziale 6.2. Nie jest wskazane stosowanie wartości ilorazu f_2/f_1 równej dokładnie połowie, gdyż prowadzi to do niejednoznaczności – w takim przypadku $2f_1-f_2 = f_1+f_2$ i obie składowe sumują się.

Przykłady:

```
V1 1 2 AC=1 DistoF1=5m DistoF2=2m
.Distro DEC 10 1K 1MEG 0.49
```

Deklaracja .Distro musi współlistnieć z analizą .AC, w trakcie której wyznaczane są odpowiednie immitancje oraz z deklaracją .Print zapisującą wyniki analizy do pliku wyjściowego *.out (patrz rozdział 9.7.3).

9.6.5. Analiza szumowa

Deklaracja analizy szumowej ma postać^{9.14}

```
.Noise Wyjście Źródło Rodzaj_skali Ilość_punktów fStart fStop Krok
```

Znaczenie wszystkich parametrów podano w rozdziale 6.3, przy czym *Wyjściem* może być potencjał lub różnica potencjałów wybranych węzłów.

Przykłady:

```
.Noise V(1) I1 DEC 10 1 1K 1
.Noise V(2,3) V1 OCT 8 1.0 1.0E6 1
```

W celu otrzymania wyników w pliku wyjściowym *.out należy skorzystać z instrukcji

```
.Print Noise Inoise Onoise
```

9.6.6. Analiza czasowa

Podczas analizy czasowej jest obliczana odpowiedź układu w zadanym przedziale czasu. Przed wykonaniem analizy należy zadeklarować co najmniej jedno źródło (prądowe lub napięciowe) dostosowane do analizy czasowej (patrz rozdział 9.5.1). Deklaracja instrukcji sterującej analizą czasową jest następująca:

```
.Tran TStep TStop [TStart [TMax]] [UIC]
```

Znaczenie poszczególnych parametrów jest opisane w rozdziale 7.

Przykłady:

```
.Tran 10u 10m
.Tran 10n 10u 9u 1n UIC
```

^{9.14} W przeciwieństwie do starszych wersji programu SPICE w implementacji *IsSpice 4* w celu przeprowadzenia analizy szumowej nie jest wymagana wcześniejsza deklaracja analizy AC.

Analiza jest przeprowadzana w przedziale czasu $<0, TStop>$ z krokiem $TStep$, natomiast wyniki są wyświetlane w przedziale $<TStart, TStop>$. Wartość parametru $TStart$ nie może być przy tym mniejsza od zera. Jeśli parametr $TStart$ pominięto, to program przyjmuje dla niego wartość zero. Wyniki wyprowadzane są za pomocą instrukcji `.Print` z krokiem czasowym określonym przez parametr $TStep$. Wartość tego parametru nie może być ujemna. Parametr $TMax$ określa natomiast maksymalną wartość kroku w procesie całkowania numerycznego (patrz rozdział 7.4).

9.6.7. Analiza harmoniczna

Analiza harmoniczna opiera się na algorytmie obliczającym dyskretną transformatę Fouriera ciągu próbek otrzymanych w wyniku analizy czasowej. Podczas tej analizy, podobnie jak w trakcie analizy `Tran`, są wykorzystywane pełne nieliniowe modele elementów. Analiza ta umożliwia więc wielkosygnalową analizę zniekształceń nieliniowych. Jej deklaracja jest następująca

```
.Four Częstotliwość Wyjściel [Wyjście2 ...]
```

Przykład:

```
.Four 1K V(4, 5) I(VIN)
```

W ramach analizy są obliczane wartości składowej stałej i pierwszych 9 harmonicznych dla wyspecyfikowanych w deklaracji wyjść. Składowe widma obliczane są na podstawie ciągu próbek czasowych z przedziału od ($TStop - Okres$) do $TStop$, gdzie $Okres$ wynosi $1/Częstotliwość$, a parametr $TStop$ – deklarowany w ramach instrukcji `.Tran` – oznacza moment zakończenia analizy czasowej. W praktyce wartość parametru $TStop$ dobiera się tak, aby do chwili ($TStop - Okres$) zanikły w obwodzie wszystkie stany nieustalone. Parametr `Częstotliwość` oznacza częstotliwość podstawowej harmonicznej, względem której będą liczone pozostałe składowe widma. Jeśli $TStop < Okres$, to symulator wygeneruje komunikat typu: „*Wavelength longer than time span*”.

Należy podkreślić, że instrukcja ta musi być poprzedzona instrukcją analizy stanów nieustalonych `.Tran`, podczas której obliczone zostaną przebiegi czasowe wielkości określonych przez `Wyjściel, Wyjście2...`.

W pliku wyjściowym `*.out` program umieszcza automatycznie amplitudy i fazy dziesięciu pierwszych składowych szeregu Fouriera (począwszy od składowej stałej, a skończywszy na dziewiątej harmonicznej) sygnałów występujących na zadeklarowanych wyjściach. Ponadto program SPICE oblicza całkowite – z dokładnością do dziewiątej harmonicznej – zniekształcenia harmoniczne sygnału wyjściowego (THD – ang. *Total Harmonic Distortion*).

Stosując tę analizę należy umiejętnie dobrać parametry generatora sterującego układem i skorelować je z deklaracją częstotliwości podstawowej. Najbardziej użyteczne wyniki otrzymuje się, gdy generator wytwarza sygnał okresowy o częstotliwości podstawowej równej zadeklarowanej pierwszej harmonicznej (parametr `Częstotliwość`).

Ze względu na błędy numeryczne algorytmu *DFT* analiza .Four jest polecana do analizy układów wnoszących duże zniekształcenia nieliniowe. Przy małych zniekształceniach dokładniejsze wyniki daje analiza .Disto.

9.6.8. Małoprzyrostowa analiza wrażliwości

W sposób klasyczny program SPICE oblicza wrażliwość dla analiz stałoprądowej i zmiennoprądowej. Odpowiednie deklaracje mają postać:

```
Sens Wyjście
Sens Wyjście AC Rodzaj_skali Ilość_punktów fStart fStop
          *x
```

Znaczenie poszczególnych parametrów jest opisane w rozdziale 8.2. W programie *IsSpice 4* deklaracja analizy wrażliwości nie ma formy „z kropką”, lecz należy do bloku komend skryptowego języka sterowania symulacją *ICL* (patrz rozdział 9.8).

Przykład:

```
.Control
Sens V(2)
Sens V(2) AC LIN 100 1k 2k
Print all
.Endc
```

9.7. Polecenia sterujące i pomocnicze

9.7.1. Sterowanie zbieżnością obliczeń

W przypadku problemów ze zbieżnością obliczeń stałoprądowego punktu pracy można zastosować instrukcję:

```
.Nodeset V(Nr_węzła1)=Wartość1 [V(Nr_węzła2)=Wartość2 ...]
```

Przykład:

```
.Nodeset V(2)=3.4 V(10)=0
```

Instrukcja .Nodeset, opisana szczegółowo w rozdziale 5.3.4, pozwala na rozpoczęcie iteracji od wartości szacowanych przez użytkownika, czyli bliskich właściemu punktowi pracy układu^{9.15}. W przypadku nieuzyskania zbieżności program drukuje ostatnie wartości potencjałów węzłowych układu i przerywa obliczenia. Wydrukowane wartości nie muszą być w takim przypadku poprawne ani nawet zbliżone do poprawnych, ale niosą pewną pomocną informację (patrz rozdział 5.3.4).

9.7.2. Definiowanie warunków początkowych

Deklaracja instrukcji ustalającej warunki początkowe ma postać

```
.IC V(Nr_węzła1)=Wartość1 [V(Nr_węzła2)=Wartość2 ...]
```

Przykład:

```
.IC V(1)=0 V(2)=5 V(3)=.64
```

^{9.15}Należy złożyć, że użytkownik zna zasadę działania analizowanego układu.

Problematyka ustalania warunków początkowych symulacji jest przedstawiona szczegółowo w rozdziale 7.3.

9.7.3. Wyprowadzanie wyników

Instrukcja `.Print` powoduje generowanie do pliku wyjściowego `*.out` wyników analizy w postaci tablicy. Format instrukcji jest następujący

```
.Print Typ Wyjście1 [Wyjście2 ...]
```

Pole *Typ* określa, dla której z analiz drukowane będą wyniki – dozwolone są analizy *AC*, *Disto*, *Noise* i *Tran*. Parametry *Wyjście1*, *Wyjście2*, ... określają zmienne wyjściowe, których wartości będą umieszczone w tablicy. W przypadku analiz: *AC* i *Disto*, za literami *V* lub *I* (potencjał / napięcie lub prąd), można w celu uściślenia postaci wyniku dodać litery:

- *R* – część rzeczywista,
- *I* – część urojona,
- *P* – faza,
- *dB* – $20 \cdot \log(\text{amplituda})$.

Przykłady:

```
.Print AC VdB(5,7) VP(5,7) IR(Vin) II(Vin)
.Print Disto V(4)
.Print Noise Inoise Onoise
.Print Tran V(4) I(Vin)
```

9.7.4. Deklaracja i definicja modelu

Deklaracja modelu następuje przy wywoływaniu elementu nim określonego, np.

```
Q1 4 2 0 BC108B
```

Model definiuje się wg formatu

```
.Model Nazwa Typ (pn1=pv1 pn2=pv2 ...)
```

Parametr *Nazwa* jednoznacznie określa zadeklarowany model elementu. W polu *Typ* powinno znaleźć się jedno ze słów kluczowych, które identyfikują rodzaj elementu, którego model dotyczy. Wspomniane słowa kluczowe to m.in.:

- *C* – kondensator,
- *R* – rezystor,
- *D* – dioda,
- *NPN* – tranzystor bipolarny typu *n-p-n*,
- *PNP* – tranzystor bipolarny typu *p-n-p*,
- *NJF* – tranzystor polowy *JFET* z kanałem typu *n*,
- *PJF* – tranzystor polowy *JFET* z kanałem typu *p*,
- *NMOS* – tranzystor polowy *MOSFET* z kanałem typu *n*,
- *PMOS* – tranzystor polowy *MOSFET* z kanałem typu *p*,
- *NMF* – tranzystor polowy *MESFET* z kanałem typu *n*,
- *PMF* – tranzystor polowy *MESFET* z kanałem typu *p*.

Na końcu linii deklaracji modelu podaje się, po słowach kluczowych identyfikujących parametry modelu (*pn1*, *pn2* ...), wartości poszczególnych parametrów (*pv1*, *pv2* ...). Nie jest istotna kolejność deklarowania parametrów. Nie ma również konieczności podawania wszystkich wartości. Najprostsze modele mogą w ogóle nie zawierać listy parametrów i wówczas wszystkie wartości parametrów będą równe wartościami domyślnymi.

Przykład:

```
.MODEL BC108B NPN (IS=1.02E-14 NF=1.0 BF=468 VAF=80
+ IKF=6.0E-02 ISE=2.17E-12 NE=2.0 BR=4 NR=1.0 VAR=20
+ XTB=1.5 RE=8.1E-01 RB=3.3E+00 RC=3.3E-01
+ CJE=1.6E-11 CJC=4.7E-12 TF=4.7E-10 TR=6.2E-08)
```

9.7.5. Dołączanie bibliotek

Instrukcja pozwalająca na przeszukiwanie zbiorów bibliotecznych *.lib w celu znalezienia potrzebnych podobwodów i modeli urządzeń ma postać

```
*Include [d:] [ścieżka] nazwa.lib
```

Przykład:

```
*Include User.lib
```

Po odnalezieniu wymaganych podobwodów i/lub modeli program automatycznie dołącza je do zbioru wejściowego *.cir.

9.7.6. Deklaracja i definicja podobwodu

Podobwód jest deklarowany wg składni:

```
Xnazwa Nr_węzła1 [Nr_węzła2 [Nr_węzła3 ...]] Nazwa_podobwodu
```

Przykład:

```
X1 9 8 7 6 5 UA741
```

Definiowanie podobwodu należy rozpocząć od linii zawierającej instrukcję

```
.Subckt Nazwa_podobwodu Nr_węzła1 [Nr_węzła2 [Nr_węzła3 ...]]
```

Instrukcja .Subckt rozpoczyna deklarację struktury podobwodu o nazwie podanej przez parametr *Nazwa_podobwodu*. Parametry *Nr_węzła1*, *Nr_węzła2* ..., oznaczają numery węzłów wewnętrznych udostępnianych na zewnątrz podobwodu. Linie następujące po instrukcji .Subckt powinny zawierać instrukcje i deklaracje definiujące strukturę podobwodu. Zabronione jest używanie w tym miejscu instrukcji sterujących analizą obwodu. Dozwolone jest natomiast zdefiniowanie kolejnego podobwodu, wywołanie istniejącego już podobwodu, zadeklarowanie i wywołanie dowolnego modelu. Należy przy tym podkreślić, że wszelkie nazwy definiowane wewnątrz podobwodu mają znaczenie lokalne, tzn. nie są rozpoznawane przez program na zewnątrz podobwodu. Deklaracja struktury podobwodu kończy się instrukcją .Ends.

Wykorzystanie podobwodów pozwala zamknąć skomplikowane struktury układowe w jeden blok, co m.in. zwiększa czytelność schematu oraz pozwala wykorzystać dany podobwód w wielu projektach. Poza tym wiele modeli elementów jest tworzo-

nych jako podobwody. Doskonałymi przykładami są tranzystory w.cz. (ich modele zawierają klasyczne modele tranzystorów uzupełnione elementami L i C pozwalającymi precyzyjniej modelować zachowanie tranzystorów w zakresie w.cz. i b.w.cz.) i wzmacniacze operacyjne (w tym przypadku dla jednego typu wzmacniacza może istnieć kilka modeli o różnym stopniu złożoności).

Przykład 1 (tranzystor w.cz.):

```
.Subckt BFY90 1 2 3
LC 1 4 0.375E-9
LB 2 6 1.064E-9
LE 5 3 1.232E-9
CC 4 3 0.644E-12
CB 4 6 0.205E-12
Q1 4 6 5 QR04
.MODEL QR04 NPN ( BF=56 VAF=120 VAR=12.0 RC=4.6 RB=28.5
+RE=.824 IKF=0.35E-01 ISE=0.45E-14 TF=0.556E-10
+TR=0.40E-08 ITF=0.17E-01 VTF=0.50E+01 CJC=0.534E-12
+CJE=0.957E-12 XTI=3.0 NE=1.5 ISC=0.15E-14 EG=1.11
+XTB=1.5 BR=1.46 VJC=0.75 VJE=0.75 IS=0.50E-15
+MJC=0.33 MJE=0.33 XTF=4.0 IKR=0.35E-01 KF=0.1E-14
+NC=1.7 FC=0.50 RBM=20 IRB=0.50E-02 XCJC=0.482 )
.Ends
```

Przykład 2 (liniowy model wzmacniacza operacyjnego μA741):

```
.Subckt UA741 2 3 6 7 4
RP 4 7 10K
RXX 4 0 10MEG
IBP 3 0 80NA
RIP 3 0 10MEG
CIP 3 0 1.4PF
IBN 2 0 100NA
RIN 2 0 10MEG
CIN 2 0 1.4PF
VOFST 2 10 1MV
RID 10 3 200K
EA 11 0 10 3 1
R1 11 12 5K
R2 12 13 50K
C1 12 0 13PF
GA 0 14 0 13 2700
C2 13 14 2.7PF
RO 14 0 75
L 14 6 30UHY
RL 14 6 1000
CL 6 0 3PF
.Ends
```

Przykład 3 (nieliniowy model wzmacniacza operacyjnego μA741):

```
.Subckt UA741 2 3 6 7 4
*nodes: - IN + OUT VCC VEE
QNI1 10 2 13 QNI1
QNI2 12 3 13 QNI2
.MODEL QNI1 NPN(NF=1.5 BF=111 IS=8E-16 CJE=3PF)
.MODEL QNI2 NPN(NF=1.5 BF=144 IS=8.3E-16 CJE=3PF)
Q3 13 14 4 QN741
IEE 4 14 185NA
```

```

CCM 13 4 2.5PF
RCM 13 4 10MEG
RC1 11 10 1K
RC2 11 12 1K
CHF 10 12 55PF
D1 7 11 D741
RP 7 4 10K
GA 0 15 12 10 .9MMHO
GCM 0 15 13 0 6.3NMHOS
R2 15 0 100K
D2 15 0 D741 OFF
D3 0 15 D741 OFF
C2 15 16 30PF
GB 16 0 15 0 12.5
RO2 16 0 1000
D4 16 17 D741P OFF
EP 17 0 7 0 -1.8 1
D5 18 16 D741P OFF
EN 0 18 0 4 -2.3 1
.MODEL D741P D(RS=1M)
D6 19 16 D741
D7 16 20 D741
IRO 20 19 170UA
RR0 16 21 1MEG
Q4 7 19 21 QNO
Q5 4 20 21 QPO
.MODEL QNO NPN(BF=150 CJC=3P IS=1E-14)
.MODEL QPO PNP(BF=150 CJC=3P IS=1E-14)
L1 21 6 30UHY
RL1 21 6 1K
.MODEL D741 D(CJO=3PF)
.MODEL QN741 NPN
.Ends

```

9.8. Interaktywny język sterowania symulacją ICL

Interaktywny język sterowania symulacją *ICL* (ang. *Interactive Command Language*) stanowi rozszerzenie języka SPICE, zwiększające jego możliwości obliczeniowe i funkcjonalne, przede wszystkim w zakresie przetwarzania i analizy wyników. Dzięki bogatemu zestawowi nowych funkcji oraz komend możliwe jest sterowanie symulacją na bieżąco, bez konieczności dokonywania zmian z poziomu edytora schematów. W celu zastosowania komend języka *ICL* należy do pliku wejściowego **.cir* dodać blok sterujący rozpoczynający się od linii *.Control*, a kończący się na linii *.Endc.*

```

.Control
...
komendy języka ICL
...
.Endc

```

Blok sterujący jest umieszczany na początku pliku wejściowego, po linii tytułowej i przed instrukcjami rozpoczynającymi się od kropki. Zawiera on komendy *ICL* wraz z argumentami, którymi mogą być wyrażenia lub wektory.

Wyrażenia są używane jako podstawa do podejmowania decyzji lub do wyliczania wektorów wyjściowych, które nie są bezpośrednio dostępne w wyniku przeprowa-

dzenia symulacji. Wyrażenie może składać się z różnych kombinacji wektorów, wielkości skalarnych i funkcji połączonych z różnymi kombinacjami operatorów.

Wyniki symulacji zapisywane są w formie wektorów (uporządkowanych ciągów wartości), za pośrednictwem komend *print*^{9.16} lub *let*. Preferowana jest komenda *print* tworząca wektory, które mogą być wykorzystane jako argumenty komend *ICL* i są widoczne w postprocesorze graficznym. Wektory powstałe w wyniku działania komend *let* i *save* mogą być wykorzystane jedynie jako argumenty komend.

Nazwy wektorów mogą rozpoczynać się od symbolu „@”. Użycie tego symbolu oznacza, że nazwa odnosi się do wewnętrznego parametru urządzenia lub modelu. Nazwa takiego wektora ma postać: @nazwa[parametr], gdzie *nazwa* musi być nazwą urządzenia (modelu), a *parametr* nazwą odpowiedniego parametru, właściwego dla podanego wcześniej urządzenia.

W nazwie wektora możemy również odwoływać się do poszczególnych jego elementów. Odwołanie to może mieć postać: *wektor*[*n*] lub *wektor*[*m*, *n*]. Pierwszy zapis odwołuje się do *n*-tego elementu *wektora*, natomiast drugi – do wszystkich elementów *wektora* znajdujących się pomiędzy *m*-tym i *n*-tym elementem.

9.8.1. Operatory

Dostępne w programie operatory (arytmetyczne, logiczne i relacyjne) są zebrane w tablicy 9.7. Każdy operator relacyjny może być zastąpiony dwuliterowym syno-

Tab. 9.7. Operatory stosowane w komendach ICL

Lp.	Nazwa operatora w bloku ICL	Opis
Operatory arytmetyczne		
1	+	operator dodawania
2	-	operator odejmowania
3	*	operator mnożenia
4	/	operator dzielenia
5	^	operator potęgowania
6	%	operator reszty z dzielenia
Operatory logiczne		
1	&	operator <i>And</i>
2		operator <i>Or</i>
3	!	operator <i>Not</i>
Operatory relacyjne		
1	< (lt)	mniejszy niż (ang. <i>less than</i>)
2	> (gt)	większy niż (ang. <i>greater than</i>)
3	>= (ge)	większy niż lub równy (ang. <i>greater than or equal</i>)
4	<= (le)	mniejszy niż lub równy (ang. <i>less than or equal</i>)
5	= (eq)	równy (ang. <i>equal</i>)
6	<> (ne)	nie równy (ang. <i>not equal</i>)

^{9.16} Działanie komendy *print* występującej w ramach bloku ICL jest zbliżone do działania „oryginalnej” deklaracji *.Print*, wywodzącej się wprost ze standardu „Berkeley SPICE”.

nimem podanym w nawiasie. W przypadku operatorów logicznych można alternatywnie stosować nazwy tradycyjne, tj. *and*, *or* i *not*.

9.8.2. Funkcje

W języku *ICL* dostępne są funkcje użytkownika i funkcje standardowe. Funkcje użytkownika definiuje się, wykorzystując komendę *function* (patrz rozdział 9.8.3). Funkcje standardowe można podzielić na funkcje matematyczne, funkcje związane z pozycjami cursorów i funkcje wektorowe. Wykaz najważniejszych funkcji standardowych stosowanych w komendach *ICL* jest przedstawiony w **tablicy 9.8**.

W przypadku funkcji trygonometrycznych argument jest standardowo wyrażany w radianach. Zamiany na stopnie można dokonać poprzez zmianę parametru *units* za pomocą komendy *set* (patrz rozdział 9.8.3).

9.8.3. Komendy języka ICL

Z uwagi na fakt, że wiele komend języka *ICL* jest ze sobą powiązanych, powinny być one wprowadzane, a następnie wykonywane w odpowiedniej kolejności. Przeważnie zakłada się następującą kolejność:

- zapis wektorów *save*,
- deklaracje *alias*,
- deklaracje *view*,
- punkty kontrolne lub deklaracje pętli,
- instrukcje sterujące analizami,
- deklaracje *let*,
- deklaracje *alter*,
- deklaracje wprowadzania wyników *print*.

Do podstawowych komend *ICL* należą:

- ***alias***

Komenda *alias* służy do skracania nazw wektorów i tworzenia wyrażeń opisujących te wektory. Format komendy jest następujący

alias nazwa wyjście

Wszystkie komponenty występujące po parametrze *nazwa* muszą być wcześniej zapisane za pomocą komend *save* lub *print*. W programie *IsSpice 4* deklaracja *alias* jest dodawana automatycznie po dołączeniu sond pomiarowych.

Przykłady:

```
alias vwy v(4)
alias gus 20*log(v(9)/v(1))
```

- ***alter***

Komenda *alter* jest przeznaczona do zmiany parametrów modeli i elementów. Format komendy jest następujący:

```
alter @model[parametr] = [wartość] lub [wyrażenie]
```

lub

```
alter @element[parametr] = [wartość] lub [wyrażenie]
```

Tab. 9.8. Wybrane funkcje standardowe stosowane w komendach ICL

Lp.	Nazwa funkcji w bloku ICL	Opis
Funkcje matematyczne		
1	mag(z)	$ z $
2	ph(z)	$\arg(z) \text{ w } [^\circ]$
3	j(z)	$j \cdot z$
4	real(z)	$\operatorname{Re}\{z\}$
5	imag(z)	$\operatorname{Im}\{z\}$
6	db(x)	$20 \cdot \log_{10}(x)$
7	log(x)	$\log(x); \text{ podstawa } 10$
8	ln(x)	$\ln(x); \text{ podstawa e}$
9	exp(x)	e^x
10	abs(x)	$ x $
11	sqrt(x)	\sqrt{x}
12	sin(x)	$\sin(x)$
13	cos(x)	$\cos(x)$
14	tan(x)	$\operatorname{tg}(x)$
15	atan(x)	$\operatorname{arc tg}(x)$
16	rnd(x)	Liczba losowa o rozkładzie równomiernym z przedziału $<0...x>$
Funkcje związane z kurSORAMI ograniczającymi wektor danych		
1	rms	Wartość średniokwadratowa (ang. root mean square)
2	mean	Wartość średnia
3	max	Wartość maksymalna
4	min	Wartość minimalna
5	stddev	Wartość średniokwadratowa składowej zmiennej
6	pk_pk	Wartość międzyszczytowa
7	trise	Czas narastania (od 10% do 90% – względem kurSORów)
8	tfall	Czas opadania (od 90% do 10% – względem kurSORów)
9	getcursor0	Funkcja zwraca wartość wektora określona pozycją kurSora 0
10	getcursor1	Funkcja zwraca wartość wektora określona pozycją kurSora 1
11	getcursorx	Funkcja zwraca pozycję kurSora
12	getcursor(y, #)	Funkcja zwraca wartość wektora vec określona pozycją kurSora #
Funkcje wektorowe		
1	interpolate	Funkcja powoduje przeskalowanie wektora do bieżącej skali
2	initialvalue	Funkcja zwraca pierwszą wartość wektora
3	finalvalue	Funkcja zwraca ostatnią wartość wektora
4	norm	Funkcja dzieli wszystkie elementy wektora przez wartość elementu największego
5	pos	Funkcja zwraca wektor, którego wartości wynoszą 1 – gdy odpowiednie elementy argumentu mają niezerową część rzeczywistą i 0 – w przeciwnym przypadku
6	length	Funkcja zwraca długość wektora

Argumenty komendy *alter : model lub element* są nazwami modelu lub elementu, których parametry (*parametr*) mają być zmienione.

Przykłady:

```
alter @qn2222[bf] = 225
alter @rl[resistance] = @rl[resistance]+10
alter @l:nmos[vto] = 2
```

- **dowhile / while**

Komendy *dowhile* / *while* umożliwiają realizację obliczeń w pętli. Struktura pętli jest następująca:

```
dowhile / while warunek
komendy
end
```

Przykład:

```
dowhile rms(v(3))>400m
alter @q5[temp]=@q5[temp]+10
tran ln 100n
print @q5[temp] rms(v(3))
end
```

Działanie pętli *dowhile* jest funkcjonalnie identyczne z działaniem pętli *while*. Różnica polega na tym, że *komendy* wewnętrz pętli *dowhile* są wykonywane przed sprawdzeniem *warunku*. Instrukcja *while* wykonuje *komendy* wewnętrz pętli dopóki spełniany jest *warunek* – sprawdza więc warunek przed rozpoczęciem pętli.

- **function**

Komenda *function* umożliwia definiowanie własnych funkcji. Format komendy jest następujący

```
function nazwa_funkcji(argumenty) wyrażenie
```

Przykład:

```
function min3(a,b,c) min(min(a,b),c)
```

Zdefiniowane funkcje użytkownika mogą być następnie wykorzystywane w wyrażeniach innych komend, np.

```
let test = min3 (v(1),v(2),v(3))
```

- **if – then – else**

Struktura *if – then – else* umożliwia realizację instrukcji warunkowej. Składnia tej struktury jest następująca:

```
if warunek
komendy
else
komendy
end
```

Przykład:

```
repeat 10
tran ln 100n
if mean(v(9)) > 50m
alter @rl[resistance] = @rl[resistance] + 10
else
print mean(v(9)) @rl[resistance]
end
```

Instrukcja warunkowa jest stosowana w przypadku, gdy konieczne jest podjęcie różnych działań w zależności od spełnienia określonego *warunku*. Powyższy przykład ilustruje zastosowanie struktury *if – then – else* w celu znalezienia odpo-

wiedniej wartości rezystancji R_1 . Pętla jest realizowana za pomocą przedstawionej dalej instrukcji *repeat*.

- **let**

Komenda *let* umożliwia tworzenie nowych wektorów. Struktura instrukcji jest następująca:

```
let nwektor = wektor lub wyrażenie
```

Przykład:

```
let test=exp(v(8)*v(9))/((470/@d3[id])-1)
```

Nowy wektor (*nwektor*) może być wykorzystany w innych komendach lub wyświetlony bezpośrednio, np.

```
view tran test
```

- **repeat**

Działanie komendy *repeat* polega na powtarzaniu komend zawartych między słowami kluczowymi *repeat* i *end*. Liczba повторzeń jest określona przez parametr *liczba_kroków*. Struktura pętli jest następująca:

```
repeat liczba_kroków  
komendy  
end
```

Przykład:

```
save all allcur  
function rms(vec) sqrt(mean(vec*vec))  
view tran v(8)  
repeat 10  
tran ln 100n  
print @l1[inductance] rms(i(v2)) mean(v(8))  
alter @l1[inductance]=@l1[inductance]-5m  
end
```

- **save**

Komenda *save* umożliwia zapis wektora. Format komendy jest następujący

```
save [wektory] [all] [allcur] [allpow]
```

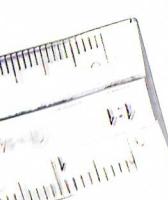
Przykłady:

```
save all allcur allpow  
save v(8) v(7)
```

Wszystkie wektory powinny być zapisane przed wydrukowaniem lub użyciem w innych komendach *ICL*. Parametr *all* powoduje, że zapisywane będą wszystkie napięcia i prądy źródeł napięciowych. Parametr *allcur* powoduje zapis prądów przepływających przez wszystkie elementy, a *allpow* – mocy wydzielanych na elementach. Można stosować dowolne kombinacje tych parametrów. Argumenty instrukcji *Print* są zapisywane automatycznie i nie trzeba ich ponownie zapisywać komendą *save*.

- **set**

Komenda *set* umożliwia ustawianie lub zmianę zmiennych globalnych programu. Format komendy jest następujący



```
set [zmienna] lub [zmienna = wartość]
```

Przykłady:

```
set temp=125
set units=degrees
```

Oprócz wszystkich parametrów instrukcji `.Options` komenda `set` oferuje wiele własnych *zmiennych* dotyczących m.in. transformaty Fouriera i wydruków w trybie tekstowym. W szczególności zmienna `units` umożliwia zmianę z radianów na stopnie domyślnej miary kąta stosowanej w funkcjach trygonometrycznych (jak w drugim przykładzie).

- **`show`**

Komenda `show` umożliwia wyświetlanie informacji o wartości parametrów w punkcie pracy. Format komendy jest następujący

```
show [element] [litera] [all] [:parametry]
```

Przykłady:

```
show q1
show c r q5 : p
show all : temp
show m:1:5 : id iss vto
```

Komenda `show` pozwala uzyskać informacje o wszystkich elementach obwodu w wyznaczonym punkcie pracy. Elementy do wylistowania możemy zadeklarować bezpośrednio, podając pełną nazwę konkretnego elementu lub podając literę kluczową, wskazującą programowi typ elementu. Słowo `all` powoduje wylistowanie wyników dla wszystkich użytych urządzeń lub modeli. Po dwukropku możemy zadeklarować parametry urządzeń/modeli, które nas interesują.

Informacje o urządzeniach/modelach znajdujących się wewnętrz podobwodów są również dostępne, ale wymagają nieco innej deklaracji. Deklarację należy uzupełnić o nazwę podobwodu objętą dwukropkami (`litera:nazwa:`). Przykładowo, deklaracja

```
show X:3:
```

wyprowadzi informacje o wszystkich urządzeniach wewnętrz podobwodu `X3`. Powyższy opis odnosi się również do komendy `showmod`, która umożliwia wyświetlanie parametrów modeli.

- **`view`**

Komenda `view` umożliwia tworzenie podglądu wykresu. Format komendy jest następujący

```
view analiza wektor lub nazwa [min] [max] [=wyrażenie]
```

Przykłady:

```
view dc v(8)
view tran v(8) 0 5
view tran gain 0 10 =20*log(v(8)/v(1))
view ac out =sqrt(real(v(8))^2+imag(v(8))^2)
```

Komenda *view* powoduje utworzenie wykresu bez zapisywania danych tabelarycznych w pliku wyjściowym. Po komendzie należy podać rodzaj *analizy* i *wektor*, który może być jednym z już istniejących lub może zostać utworzony za pomocą *wyrażenia* – będzie wówczas nosił nazwę *nazwa*. Wszystkie wektory pojawiające się w wyrażeniu powinny być wcześniej zapisane za pomocą komendy *save* lub instrukcji *.Print*. Parametry *min* i *max* określają skalę wykresu. Aby otrzymać wyniki w pliku wyjściowym, należy zadeklarować *wektor* w instrukcji *.Print*.