ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1980

Seria: AUTOMATYKA z. 53 Nr kol. 649

Adam PAWLAK, Janusz JEŻEWSKI,
Jan SKIBA

# REALIZACJA JĘZYKA MODELOWANIA UKŁADÓW CYFROWYCH SSM NA MASZYNY CYFROWE ODRA SERII 1300

Streszczenie. W pracy niniejszej zaprezentowano realizację języka modelowania układów cyfrowych SSM. Język ten służy do opisu i symulacji struktur cyfrowych na poziomie bramkowym. Szereg mechanizmów języka wyjaśniono na przykładach. W punkcie 3 podano formalny opis zrealizowanej wersji języka. W podsumowaniu ustosunkowano się do szeregu problemów związanych z wykorzystaniem translatora/symulatora oraz wytyczono kierunki dalszych prac nad tym narzędziem.

## 1. Wprowadzenie

Symulacja układów cyfrowych może w istotny sposób wpłynąć na przyspieszenie projektowania, jak i na jakość projektu układu cyfrowego. Do najważniejszych funkcji spełnianych przez symulatory układów cyfrowych [4,8] należą: logiczna weryfikacja projektowanego układu, analiza jego dynamiki (czasowa weryfikacja projektu),modelowanie błędów w układzie oraz generowanie testów diagnostycznych.

Istotny wpływ na użyteczność systemu modelowania ma język do opisu i symulacji struktur cyfrowych. Języki tego typu [12-15], tzw. języki CHDL (Computer Hardware Description Languages), są zwykle związane z poziomem symulacji. Język SSM [1-3], którego implementacja jest przedmiotem niniejszej pracy służy do modelowania układów cyfrowych na poziomie bramkowym. Charakteryzuje się on cechami, które na tym poziomie modelowania są bardzo użyteczne dla projektanta:

- zwięzłością i przejrzystością notacji,
- możliwością definiowania nowych struktur (elementów),
- dokładnym modelowaniem dynamiki układu,
- blokową strukturą języka umożliwiającą hierarchiczny opis układu.

Wykorzystanie w symulatorze logiki pięciowartościowej [5-6] pozwala na detekcję wyścigów oraz statycznych i dynamicznych hazardów. Decydujący wpływ na dokładność symulacji ma rodzaj stosowanego modelu bramki logicznej. W zrealizowanej wersji symulatora przyjęto model zmiennych opóźnień z przedziałem dwuznaczności oraz z absorpcją [7].

## 2. Opis układu w języku SSM

Wszelkie użyte do opisu układu elementy muszą zostać uprzednio określone. Jeżeli są to elementy z zestawu elementów podstawowych języka, to wystarcza podanie nazwy identyfikującej ich typ, lecz jeżeli są to elementy nowe, to zachodzi konieczność opisu ich "bramkowego" wnętrza - zdefiniowania.

Deklaracje służą do ilościowej specyfikacji zdefiniowanych wcześniej typów elementów. Instrukcje transferu są wykorzystywane do opisu połączeń międzyelementowych, a instrukcje symulacji precyzują dodatkowo warunki konieczne dla prawidłowej symulacji.

### 2.1. Elementy podstawowe a elementy definiowane

Przez elementy podstawowe rozumiemy takie elementy, których nie trzeba opisywać przed ich użyciem. Ponieważ język SSM pozwala opisywać układ cyfrowy tylko na poziomie bramkowym, stąd właśnie bramki logiczne traktowane są, jako główne elementy podstawowe (elementy obiektu symulacji). Rozróżniamy 16 typów elementów podstawowych, a mianowicie:

- bramki AND2, AND4, NAND2, NAND3, NAND4, NAND8, NOR2, NOR3, NOR4,NOR8,OR2 OR4, XOR, INV,
- element opóźnienia DELAY,
- generator wymuszeń zewnętrznych FUNCTION.

Właściwie mając do dyspozycji wyżej wymienione 16 typów elementów podstawowych, można opisać dowolny układ bez konieczności definiowania dodatkowych struktur. Byłoby to zajęcie bardzo żmudne i pracochłonne dla projektanta, a poza tym proces pisania programu stwarzałby wiele okazji do popełniania błędów, zwłaszcza przy opisie dużych układów. Z uwagi na to język posiada możliwość opisu wybranych struktur przy pomocy tzw. makroopisów. Otóż jeżeli pewne partie logiki obwodu globalnego o identycznej konfiguracji wewnętrznej są rozmieszczone w różnych częściach obwodu, to wtedy daną partię logiki opisujemy (definiujemy) tylko raz, nadając jej pewną odrębną nazwę (typ). W efekcie przy każdorazowym wykorzystaniu tej części nie powielamy już jej opisu wnętrza, a jedynie odwołujemy się do niej poprzez jej typ.

### 2.2. Deklaracje elementów składowych symulowanej struktury

Zadaniem deklaracji jest przydzielenie translatorowi określonej liczby elementów wybranych typów, których pragniemy użyć do opisu symulowanej struktury. Obowiązuje zasada, że deklarujemy tylko te elementy, których struktura wewnętrzna jest znana, tzn. albo elementy podstawowe języka,al-

bo elementy wcześniej zdefiniowane. Dodatkowych wyjaśnień wymagają deklaracje elementów opóźnień.

2.2.1. Opóźnienia

Standardowym opóźnieniem w symulatorze jest tzw. opóźnienie jednostkowe, które jest związane z każdą bramką jak i z każdym elementem definiowanym. Polega ono na tym, że wyliczona w i-tym cyklu wartość na wyjściu elementu pojawia się dopiero w cyklu (i+1)-szym. Przyłączenie do wyjścia bramki lub elementu definiowanego elementu opóźnienia DELAY umożliwia uzyskiwanie dowolnych opóźnień przedziałowych i to w zależności od potrzeb różnych dla narastania i opadania sygnału. Przykładowo deklaracja:

DELAY (1-4,2-5) D1, D2;

określa dwa elementy opóźnienia, których parametry czasowe są następujące:

$min_{LH}$ = 1 cykl, $max_{LH}$ = 4 cykle, $min_{HL}$ = 2 cykle, $max_{HL}$ = 5 cykli.

Jednostką, której wielokrotność stanowią powyższe parametry jest długość cyklu obowiązująca w bloku, w którym zadeklarowano dany element DELAY.

## 2.3. Połączenia elementów

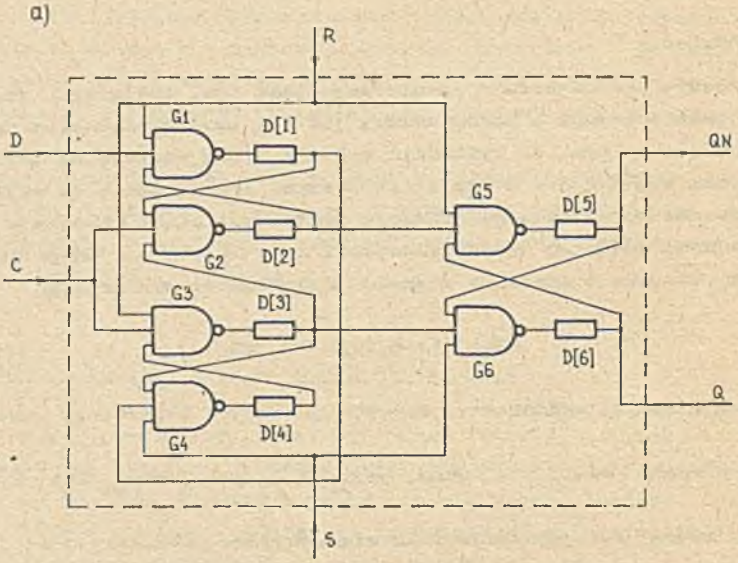
Kolejność w jakiej pisane są poszczególne transfery w danym bloku nie ma żadnego wpływu na wyniki symulacji. Na zewnątrz bowiem układ pracuje synchronicznie niezależnie od tego, w jakiej kolejności została opisana jego struktura. Instrukcje opisu połączeń elementów stanowią najbardziej skomplikowane wyrażenia języka ze względu na możliwość stosowania różnych sposobów opisu identycznych połączeń oraz różnorodnych ułatwień i uproszczeń przy zapisie wyrażeń transferu (np. brak konieczności specyfikacji końcówek elementów). Poniżej przedstawiono pełny opis przerzutnika typu D którego schemat oraz opis w języku przedstawia rys. 1.

## 2.4. Instrukcje sterujące symulacją

Działanie instrukcji EXECUTE zostanie przedstawione poniżej. Przykładowo chcąc zamodelować element typu AND OR INVERT serii standardowej o parametrach dynamicznych wynoszących: 5,25,5,15[ns], możemy dokonać tego na różne sposoby (rys. 2).

W wersji przedstawionej na rys. 2a element AOI1 reprezentuje wyłącznie logikę rzeczywistego elementu typu AND OR INVERT, aby dodać do opisu dynamikę wystarczy przed wykorzystaniem tego elementu zadeklarować opóź-

a)




b)

```
DEFINE   DFF(C,D,S,R,Q,QN);
  BEGIN
  NAND3  G1,G2,G3,G4,G5,G6;
  DELAY  /1-5,1-3/ D[6];
  G1.E1 := R;  G1.E2 := D;  G1.E3 := D[2];
  G2.E1 := D[1];  G2.E2 := C;  G2.E3 := D[3];
  G3.E1 := R;  G3.E2 := C;  G3.E3 := D[4];
  G4.E1 := D[3];  G4.E2 := D[1];  G4.E3 := S;
  G5.E1 := R;  G5.E2 := D[2];  G5.E3 := D[6];
  G6.E1 := D[5];  G6.E2 := D[3];  G6.E3 := S;
  D[1].E1 := G1;  D[2].E1 := G2;  D[3].E1 := G3;
  D[4].E1 := G4;  D[5].E1 := G5;  D[6].E1 := G6;
  Q := D[6];  QN := D[5];
  END   DFF Z PELNYM OPISEM TRANSFERU;
```

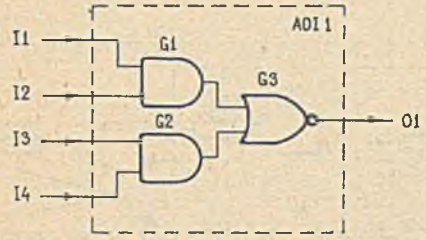
c)

```
DEFINE  DFF(C,D,S,R,Q,QN);
  BEGIN
  NAND  G1,G2,G3,G4,G5,G6;
  DELAY  /1-5,1-3/ D[6];
  G1,G2,G3 := R,D,D[2-1],C,D[3-4],R,C;
  G4,G5,G6 := D[3],S,D[1-2],R,D[6-5],S,D[3];
  D[1-6],Q,QN := G1,G2,G3,G4,G5,G6,D[6-5];
  END  DFF ZE SKROCONYM OPISEM TRANSFERU;
```

**Rys. 1. Opis przerzutnika typu D:**

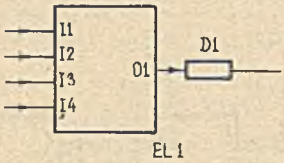
**a - schemat elementu, b - definicja przerzutnika DFF z pełnym opisem transferu, c - definicja przerzutnika DFF ze skróconym opisem transferu**

a)



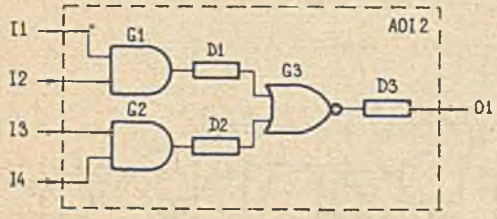

```
DEFINE  AOI1 (I1,I2,I3,I4,O1);
  BEGIN
  AND2 G1,G2;  NOR2 G3;
  G1,G2 := I1,I2,I3,I4;
  G3,O1 := G1,G2,G3;
  EXECUTE 2;
  END;
```

b)




```
AOI1 - EL1;
DELAY /1-5, 1-3/ D1;
D1 := EL1.O1;
```

c)



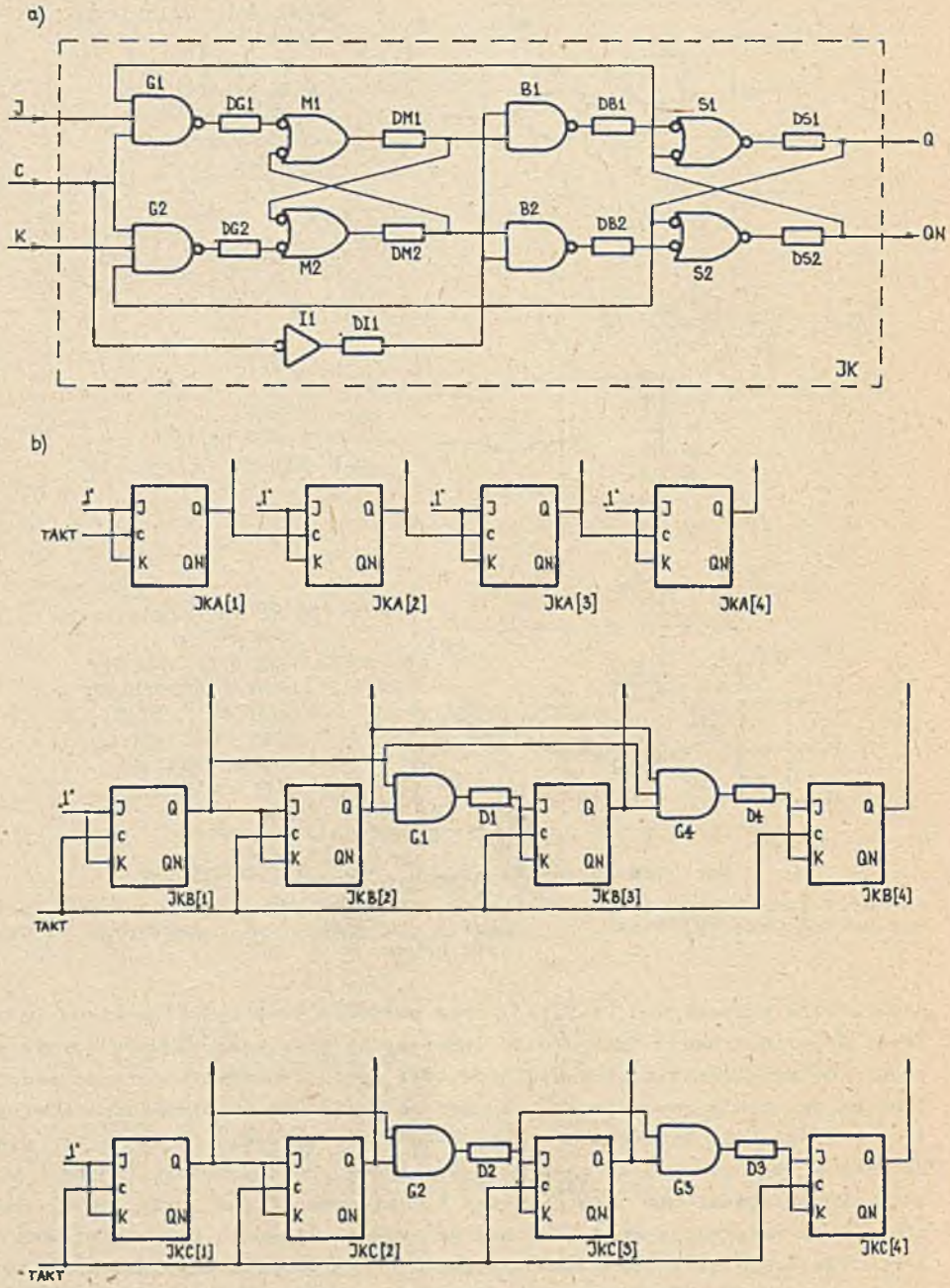

```
DEFINE  AOI2 (I1,I2,I3,I4,O1);
  BEGIN
  AND2 G1,G2;  NOR2 G3;
  DELAY /2-10, 2-5/ D1, D2;
  DELAY /3-15, 3-10/ D3;
  G1,G2 := I1,I2,I3,I4;
  D1,D2,D3 := G1,G2,G3;
  G3,O1 := D1,D2,D3;
  END;
```

Rys. 2. Modelowanie elementu typu AND-OR-INVERT

a - opis logiki elementu, b - element AND-OR-INVERT z opóźnieniem załączonym na końcówkę wyjściową, c - element z opóźnieniami załączonymi na wyjścia bramek

nienie D1 o parametrach (1-5,1-3) oraz połączyć je z wyjściem elementu AOI1 (rys. 2b). Instrukcja "EXECUTE 2" oznacza, że przy każdorazowym przystąpieniu do przeliczania elementu typu AOI1 cykl zewnętrzny zostanie podzielony na dwa cykle wewnętrzne, tzn. proces przeliczania elementów wnętrza AOI1 odbędzie się dwukrotnie. Zastosowano podział cyklu na dwa cykle wewnętrzne z tego powodu, że wnętrze zawiera dwa poziomy bramek (każdy poziom wnosi opóźnienie jednego kroku i konieczne są dwa cykle przeliczeniowe, aby informacja mogła dotrzeć na wyjście elementu D1). W elemencie z rys. 2c dynamika została już opisana w samym wnętrzu elementu AOI2. Zachodziła konieczność rozdziału wartości opóźnienia rzeczywistego elementu AND OR INVERT na dwa poziomy bramek i co się z tym wiąże ustalenia długo-

ści cyklu zewnętrznego na [1] ns. Przykład opisu układu bardziej złożonego przedstawia rys. 3.




Rys. 3. Modelowanie liczników
a - struktura wewnętrzna przerzutnika JK, b - schematy liczników,

```
0001          %                  BADANIE  LICZNIKOW
0002          % PRZYJETO ZAŁOŻENIE ŻE DŁUGOŚĆ CYKLU WYNOSI 5[NS]
0003
0004                BEGIN
0005                DEFINE  JK(C,J,K,Q,QN);
0006                      BEGIN
0007                      NAND2 M1,M2,S1,S2,B1,B2;
0008                      NAND3 G1,G2;
0009                      INV I1;
0010                      DELAY /1-2,1-1/ DG1,DG2,DM1,DM2,DB1,DB2,DS1,DS2,DI1;
0011                      G1,G2:=DS2,J,2(C),K,DS1;
0012                      DG1,DG2:=G1,G2;
0013                      M1,M2:=DG1,DM2,DM1,DG2;
0014                      DM1,DM2:=M1,M2;
0015                      B1,B2:=DM1,2(DI1),DM2;
0016                      DB1,DB2:=B1,B2;
0017                      S1,S2:=DB1,DS2,DB2,DS1;
0018                      DS1,DS2:=S1,S2;
0019                      I1,DI1:=C,I1;
0020                      Q,QN:=DS1,DS2;
0021
0022          % USTAWIENIE WARTOSCI POCZATKOWYCH BRAMEK PRZERZUTNIKA
0023
0024                       INITIAL M2,B1,S2:=0;
0025                      INITIAL G1,G2,I1,M1,B2,S1:=1;
0026                      END;
0027                AND2 G1,G2,G3;  AND4 G4;
0028                JK=JKA[4];
0029                JK=JKB[4];
0030                JK=JKC[4];
0031                DELAY /1-5,1-3/ D1,D2,D3,D4;
0032                FUNCTION TAKT=(10(0),10(1));
0033
0034          % OPIS POŁĄCZEŃ LICZNIKA SZEREGOWEGO
0035
0036                JKA[1-4].J:=4(1);
0037                JKA[1-4].K:=4(1);
0038                JKA[1-4].C:=TAKT,JKA[1-3].Q;
0039
0040          % OPIS POŁĄCZEŃ LICZNIKA Z PRZENIESIENIEM RÓWNOLEGŁYM
0041
0042                JKB[1-4].C:=4(TAKT);
0043                JKB[1].J,JKB[1].K:=2(1);
0044                JKB[2].J,JKB[2].K:=2(JKB[1].Q);
0045                G1,D1:=JKB[1-2].Q,G1;
0046                JKB[3].J,JKB[3].K:=2(D1);
0047                G4,D4:=JKB[1-3].Q,1,G4;
0048                JKB[4].J,JKB[4].K:=2(D4);
0049
0050          % OPIS POŁĄCZEŃ LICZNIKA Z PRZENIESIENIEM SKROSNYM
0051
0052                JKC[1-4].C:=4(TAKT);
0053                JKC[1].J,JKC[1].K:=2(1);
0054                JKC[2].J,JKC[2].K:=2(JKC[1].Q);
0055                G2,D2:=JKC[1-2].Q,G2;
0056                JKC[3].J,JKC[3].K:=2(D2);
0057                G3,D3:=G2,JKC[3].Q,G3;
0058                JKC[4].J,JKC[4].K:=2(D3);
0059                EXECUTE 500;
0060                WRITE (W) TAKT,JKA[1-4].Q,JKB[1-4].Q,JKC[1-4].Q;
0061                END;
```

Rys. 3. Modelowanie liczników:
c – opis struktury w języku SSM

## 3. Syntaktyka i semantyka języka SSM

Formalny opis syntaktyki języka przedstawiono w postaci notacji Backusa-Naura. Do notacji tej wprowadzono następujące dodatkowe oznaczenie:

$\{X\}$ oznacza listę złożoną z n symboli X, gdzie $n \geq 0$

### 3.1. Elementy podstawowe języka

#### 3.1.1. Syntaktyka

<ZNAK> :: = <LITERA> | <CYFRA> | <SYMBOL NIEALFANUMERYCZNY>

<LITERA> :: = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<CYFRA> :: = 0|1|2|3|4|5|6|7|8|9

<SYMBOL NIEALFANUMERYCZNY> :: = ,|;|.|:|<|=|>|"|%|'|(|)|@|
#|↑|*|+|-|/|[|]|←|↑|$|¢|?

<LICZBA> :: = <CYFRA> { <CYFRA> }

<NAZWA> :: = <LITERA> { <LITERA LUB CYFRA> }

<LITERA LUB CYFRA> :: = <LITERA> | <CYFRA>

<SŁOWO KLUCZOWE> :: = AND2|AND4|BEGIN|DEFINE|DELAY|END|
EXECUTE|E1|E2|E3|E4|E5|E6|E7|E8|
FUNCTION|INITIAL|INV|NAND2|NAND3|
NAND4|NAND8|NOR2|NOR3|NOR4|NOR8|
OR2|OR4|WRITE|XOR

#### 3.1.2. Semantyka

Główne przeznaczenie nazw, to identyfikacja elementów (obiektów) użytych do opisu modelowanego układu. Powiązanie nazwy z oznaczonym przez nią obiektem jest jednoznaczne w obszarze stanowiącym zakres działania nazwy.

Liczby służą do określania indeksów, jak również do wyznaczania parametrów dynamicznych i oznaczania liczby kroków symulacji.

Znaczenie słów kluczowych wynika z kontekstu. Są to bądź typy elementów podstawowych, czy nazwy wejść bramek, bądź też nazwy instrukcji. Słowa kluczowe nie mogą być używane jako nazwy. Spacje mogą występować w dowolnych miejscach programu mają bowiem znaczenie typograficzne.

## 3.2. Ogólna struktura programu w języku SSM

### 3.2.1. Syntaktyka

<PROGRAM> :: = <BLOK> <KOMENTARZ> ;

<BLOK> :: = BEGIN
{<DEFINICJA>}
<DEKLARACJA> {<DEKLARACJA>}
<TRANSFER> {<TRANSFER>}
<INSTR. SYMULACJI>
END

<KOMENTARZ> :: = <sekwencja dowolnych znaków oprócz;> | <PUSTY>
<PUSTY>:: =

### 3.2.2. Semantyka

Opis układu w języku SSM (program) ma strukturę blokową. Kolejność instrukcji w bloku jest następująca: definicje, deklaracje, transfer (opis połączeń) i instrukcje symulacji.

Komentarze mogą być umieszczane w programie na dwa sposoby:

- po słowie kluczowym END, aż do znaku ;
- po znaku % aż do końca karty stanowiącej dany zapis (rekord) programu źródłowego.

## 3.3. Definicja nowego typu elementu

### 3.3.1. Syntaktyka

<DEFINICJA> :: = DEFINE <NAGŁÓWEK DEFINICJI> ;
<BLOK> <KOMENTARZ> ;

<NAGŁÓWEK DEFINICJI> :: = <NAZWA TYPU> ( <NAZWA KOŃCÓWKI>
{, <NAZWA KOŃCÓWKI>})

<NAZWA TYPU> , <NAZWA KOŃCÓWKI> :: = <NAZWA>

### 3.3.2. Semantyka

Definicja służy do opisu elementu różnego od dostępnych w języku elementów podstawowych. Po słowie kluczowym DEFINE następuje nazwa identyfikująca typ aktualnie definiowanego elementu. Lista nazw końcówek specyfi-

kuje końcówki wejściowe i wyjściowe definiowanego elementu. Po nagłówku danej definicji następuje blok opisu wnętrza definiowanego typu elementu.

### 3.4. Deklaracje elementów

#### 3.4.1. Syntaktyka

<DEKLARACJA> ::= <TYP PODST.> <SEGM. DEKLARACJI>{, <SEGM. DEKLARACJI>};|
<NAZWA TYPU> - <SEGM.DEKLARACJI>{,<SEGM. DEKLARACJI>};|
<DEKL.EL.DELAY> | <DEKL.EL.FUNCTION>

<TYP PODST.> :: = AND2|AND4|INV|NAND2|NAND3|NAND4|
NAND8|NOR2|NOR3|NOR4|NOR8|OR2|OR4|XOR

<SEGM.DEKLARACJI> ::= <NAZWA EL.>{, <NAZWA EL.>} <CZĘŚĆ INDEKSOWA>|
<NAZWA EL.>{, <NAZWA EL.>}

<NAZWA EL.> :: = <NAZWA>

<CZĘŚĆ INDEKSOWA> :: = [<INDEKS>{, <INDEKS>}]

<INDEKS> :: = <LICZBA>

<DEKL.EL.DELAY> :: = DELAY <PARAMETRY> <SEGM.DEKLARACJI>
{, <SEGM. DEKLARACJI>};

<PARAMETRY> :: = / <PRZEDZIAŁ> , <PRZEDZIAŁ>/|
/<PRZEDZIAŁ> /

<PRZEDZIAŁ> :: = <MIN> - <MAX>|
<MINMAX>

<MIN> , <MAX> , <MINMAK> :: = <LICZBA>

<DEKL.EL.FUNCTION> :: = FUNCTION <GENERATOR> {, <GENERATOR>} ;

<GENERATOR> :: = <NAZWA EL.> = (<SEKW.POBUDZEŃ>{, <SEKW.POBUDZEŃ>})

<SEKW.POBUDZEŃ> ::= <STAŁA>|<WSPÓŁCZYNNIK> ( <SEKW.POBUDZEŃ>
{, <SEKW.POBUDZEŃ>})

<STAŁA> :: = Ø |1|A|E|?

<WSPÓŁCZYNNIK> :: = <LICZBA>

#### 3.4.2. Semantyka

Deklaracja elementu określa jego nazwę za pomocą,której będziemy się do niego odwoływać oraz typu tego elementu. W języku SSM istnieje możliwość deklarowania elementów podstawowych i definiowanych. Typ podstawowy służy do identyfikacji bramki logicznej (określa jej funkcję logiczną oraz li-

ozbę wejść). W segmencie deklaracji obok nazw pojedynczych elementów (zmienne proste) mogą występować nazwy tablic elementów (zmienne indeksowane). Indeksem zmieniającym się najszybciej jest indeks pierwszy z lewej w części indeksowej.

W deklaracjach elementów definiowanych nazwa typu identyfikuje typ deklarowanych elementów (jest to nazwa z odpowiedniego nagłówka instrukcji DEFINE).

Element podstawowy DELAY reprezentuje element opóźnienia. Na pełny opis jego parametrów czasowych składają się dwie pary minimalnych i maksymalnych czasów propagacji odpowiednio dla narastania i opadania sygnału wyjściowego.

Element o nazwie typu FUNCTION pełni rolę generatora wymuszeń zewnętrznych. Na opis generatora składa się nazwa identyfikująca dany generator oraz specyfikacja sekwencji wyjściowej generatora. W celu uproszczenia zapisu długich cyklicznych sekwencji umożliwiono stosowanie powtórzeń.

## 3.5. Opis połączeń międzyelementowych

### 3.5.1. Syntaktyka

<TRANSFER> ::= <WEJŚCIE> {, <WEJŚCIE>} := <WYJŚCIE> {, <WYJŚCIE>}

<WEJŚCIE> ::= <PRZYŁĄCZE> |
<PRZEDZIAŁ PUNKTOWY> . <WEJ.EL.PODST.> |
<NAZWA EL.> . <WEJ.EL.PODST.> |
<NAZWA KOŃCÓWKI>

<PRZYŁĄCZE> ::= <PRZEDZIAŁ PUNKTOWY> . <NAZWA KOŃCÓWKI> |
<PRZEDZIAŁ PUNKTOWY> |
<NAZWA EL.> . <NAZWA KOŃCÓWKI> |
<NAZWA EL.>

<PRZEDZIAŁ PUNKTOWY> ::= <NAZWA EL.> | <GRANICA> {, <GRANICA>}

<GRANICA> ::= <INDEKS DOLNY> - <INDEKS GÓRNY> |
<INDEKS>

<INDEKS DOLNY> , <INDEKS GÓRNY> ::= <LICZBA>

<WEJ.EL.PODST.> ::= E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8

<WYJŚCIE> ::= <PRZYŁĄCZE> |
<NAZWA KOŃCÓWKI> |
<WARTOŚĆ> |
<WSPÓŁCZYNNIK> ( <WYJŚCIE> {, <WYJŚCIE>})

<WARTOŚĆ> ::= ø | 1 | ?

3.5.2. Semantyka

Przyjęty w języku sposób opisu połączeń międzyelementowych jednoznacznie określa kierunek strumienia danych - sygnały z wyjść elementów podawane są na wejścia elementów. Po lewej stronie linii transferu (stronie wejść) mogą wystąpić nazwy elementów z towarzyszącymi im nazwami odpowiednich końcówek wejściowych lub też wyłącznie nazwy elementów. W drugim przypadku translator języka automatycznie generuje odpowiadającą danemu elementowi listę końcówek wejściowych w sekwencji identycznej, jak w nagłówku odpowiedniej definicji.

Pojęcie przedziału punktowego elementów związane jest ściśle z tablicą elementów. Pozwala ona na prostą i szybką deklarację większej liczby elementów tego samego typu, zaś przedział punktowy umożliwia prosty opis połączeń wybranej grupy elementów z odpowiadającej im w deklaracjach tablicy. Indeksy dolne poszczególnych wymiarów przedziału punktowego określają położenie pierwszego elementu wyodrębnionej grupy w odpowiedniej tablicy, indeksy górne położenie ostatniego. Nazwy E1, E2,...,E8 są zastrzeżone dla nazw końcówek wejściowych elementów podstawowych.

Po prawej stronie wyrażenia transferu (stronie wyjść) może dodatkowo wystąpić stała wartość sygnału oraz zamknięta w nawiasy () grupa wyjść z poprzedzającym ją współczynnikiem określającym liczbę powtórzeń tejże grupy.

Końcówka wyjściowa danego elementu podstawowego jest identyfikowana z nazwą tegoż elementu. Jest to przyporządkowanie wzajemnie jednoznaczne z tego względu, że elementy podstawowe są jednowyjściowe.

Nazwa końcówki wyjściowej (wejściowej) może wystąpić po stronie wejść (wyjść) tylko wtedy; jeżeli identyfikuje ona wyjście (wejście) elementu aktualnie definiowanego i sytuacja ta występuje w bloku opisującym wnętrze tego elementu.

## 3.6. Instrukcje sterujące symulacją

3.6.1. Syntaktyka

<INSTR. INITIAL> ::= INITIAL <USTAW. BRAMKA> { , <USTAW. BRAMKA> }
:= <WARTOŚĆ> ;

<USTAW. BRAMKA> ::= <NAZWA EL.> { . <NAZWA EL.> } . <NAZWA EL.> |
<NAZWA EL.> { . <NAZWA EL.> } . <PRZEDZIAŁ PUNKT.> |
<PRZEDZIAŁ PUNKTOWY> |
<NAZWA EL.>

<INSTR. EXECUTE> ::= EXECUTE <LICZBA CYKLI> ;

<LICZBA CYKLI> ::= <LICZBA>

<INSTR. WRITE> ::= <DRUK TEKSTU> |
<DRUK WYKRESU> |
<DRUK ZNAKOWY>

<DRUK TEKSTU> ::= WRITE (T) <TEKST> {, <TEKST>};

<TEKST> ::= <DŁUGOŚĆ> X |
<DŁUGOŚĆ> H <NAPIS>

<NAPIS> ::= <dowolna sekwencja znaków>

<DŁUGOŚĆ> ::= <LICZBA>

<DRUK WYKRESU> ::= WRITE (W) <PRZYŁĄCZE> {, <PRZYŁĄCZE>};

<DRUK ZNAKOWY> ::= WRITE (Z) <SPECYFIKACJA> {, <SPECYFIKACJA>};

<SPECYFIKACJA> ::= <PRZYŁĄCZE> |
<DŁUGOŚĆ> X

### 3.6.2. Semantyka

Instrukcja nadawania wartości początkowych (INITIAL) ustala wartości wyjściowe elementów występujące przed rozpoczęciem właściwej symulacji. W zależności od jej umiejscowienia w programie wyróżniamy dwa jej warianty:

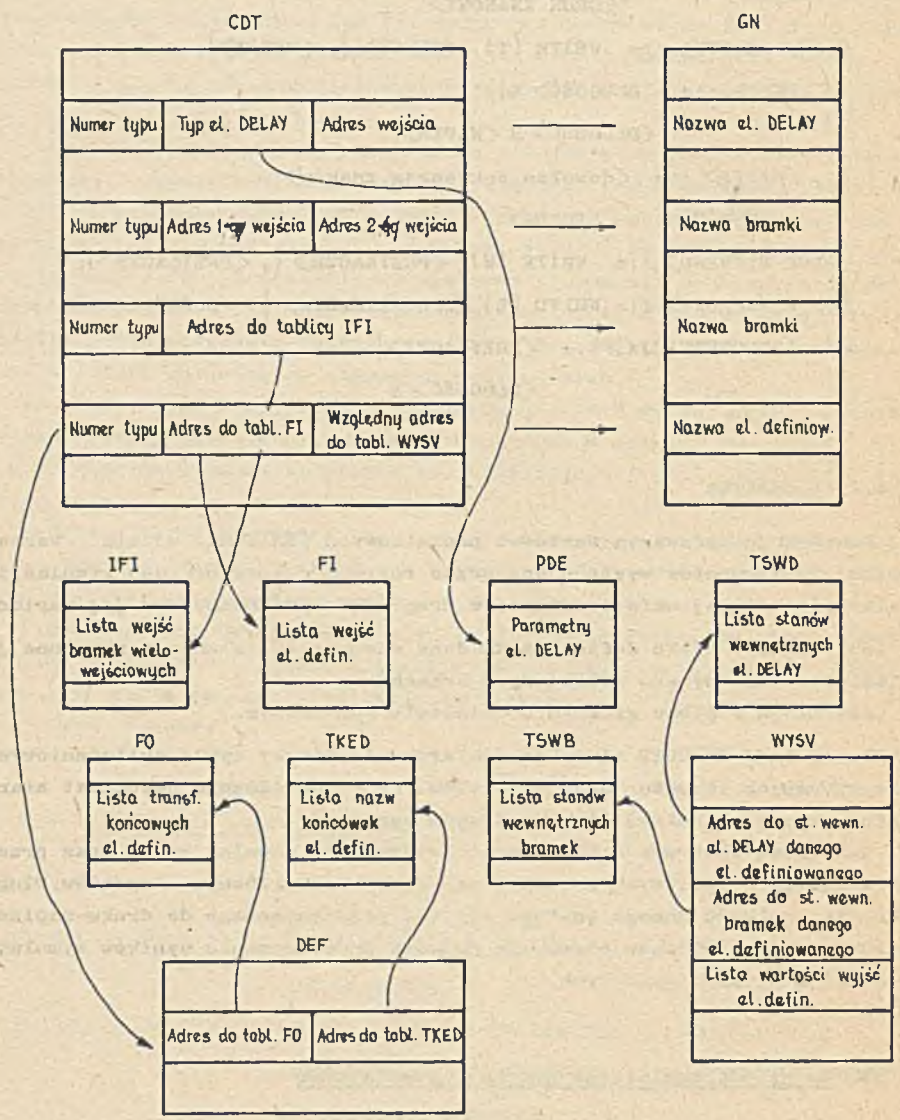
- instrukcja w bloku definiującym dany element (działanie pozbawione jest selektywnego wyboru ustawianych bramek)
- instrukcja w bloku głównym o działaniu wybiórczym.

Instrukcja EXECUTE służy do deklarowania liczby cykli obliczeniowych dla wybranych struktur układu (bloków), w bloku głównym natomiast stanowi informację o globalnej liczbie kroków symulacji.

Za pomocą instrukcji WRITE można wydrukować dowolny tekst oraz przebiegi sygnałów wyjściowych zadeklarowanych w bloku głównym elementów. Długość określa wielkość danego odstępu czy też przeznaczonego do druku napisu. Instrukcja druku wykresu czasowego pozwala na otrzymanie wyników symulacji w postaci diagramów czasowych.

## 4. Uwagi o translatorze języka i symulatorze

Symulator stanowi typ symulatora interpretacyjnego sterowanego tablicowo [9-11]. Oznacza to, że cały opis symulowanego układu zawarty jest w zbiorze tablic wygenerowanych w trakcie procesu translacji poszczególnych zdań programu źródłowego. Podstawową konfigurację zastosowanej struktury

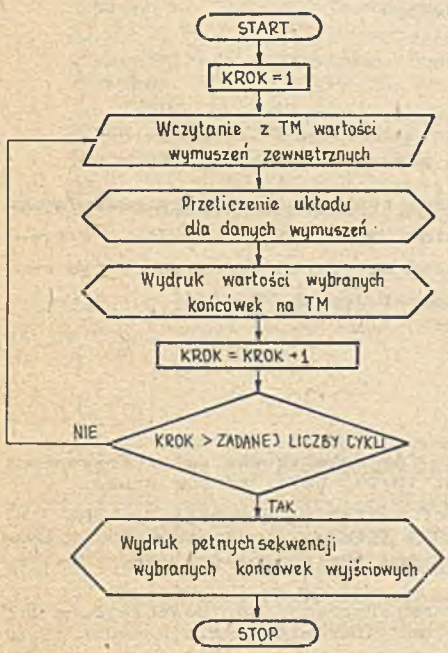



Rys. 4. Podstawowa konfiguracja struktury tablicowej

tablicowej przedstawia rys. 4. Zapełnione tablice zostają wysłane na taśmę magnetyczną, a program translatora kasując się z pamięci operacyjnej automatycznie ładuje program symulatora. Program ten po wczytaniu wygenerowanych przez translator tablic rozpoczyna właściwy proces symulacji modelowanego układu, bazujący na odpowiedniej interpretacji struktury tablicowej.

Ogólna postać przebiegu symulacji przedstawia rys. 5.

Program translatora został napisany prawie całkowicie w FORTRAN-ie, jednakże w programie symulatora najistotniejszą jego część realizującą właściwą symulację napisano w całości w PLAN-ie, w celu osiągnięcia maksymalnej szybkości symulacji.




Rys. 5. Ogólny algorytm symulacji

## 5. Podsumowanie

Translator/symulator języka SSM umożliwia symulację układu cyfrowego o wielkości około 1200 bramek podstawowych (wersja programu na m.c. ODRA 1325). Organizacja programu w postaci nakładek umożliwi modelowanie znacznie większych układów (4000 bramek). Przy symulacji dużych układów podstawowym problemem staje się czas symulacji. Mamy nadzieję, że problem ten przynajmniej częściowo zostanie rozwiązany dzięki:

- zastosowaniu w symulatorze techniki selektywnego śladu,
- wprowadzeniu do języka instrukcji wydruku selektywnego.

Implementowany w symulatorze model bramki prowadzi w szeregu sytuacji do zbyt pesymistycznych wniosków. Ma to miejsce głównie przy modelowaniu przebiegów w pętli sprzężenia zwrotnego. Istnieją propozycje rozwiązania tego problemu [19]. Wymagane jest jednakże zastosowanie bardziej złożonej algebry do modelowania układu cyfrowego. Prostszym rozwiązaniem jest wykorzystanie w symulatorze modelu opóźnień nominalnych. Symulacja funkcyjna [17-18] umożliwia częściowe rozwiązanie tego problemu. Rozszerzenie

asortymentu elementów podstawowych symulatora przynajmniej o przerzutniki i pamięci stałe oraz założenie biblioteki typowych elementów wpłynie na zwiększenie użyteczności translatora/symulatora języka SSM dla projektanta układu cyfrowego.

Zrealizowana wersja języka SSM wykazuje pewne odstępstwa w stosunku do języka prezentowanego w [1-2]; zastosowano bowiem:

- dokładniejszy model czasowy bramki,
- bogatszy repertuar bramek podstawowych,
- wydruk sygnałów w postaci wykresu czasowego

oraz jednocześnie zrezygnowano z:

- instrukcji warunkowego transferu,
- instrukcji definicji typu BLOCK,
- zestawu instrukcji dotyczących poziomu przesłań międzyrejestrowych.

Przewiduje się w najbliższej przyszłości rozbudowanie translatora/symulatora języka SSM o możliwość definiowania i symulowania układów cyfrowych na poziomach: funkcyjnym oraz przesłań międzyrejestrowych. W tym celu między innymi wykorzystany zostanie aparat języka FDL [16].

LITERATURA

[1] Hoffman H.J.: SSM-Simulationssprache für Schaltwerke unter Verwendung von mehrwertger Logik, Inter Bericht 18/74, Univ. of Karlsruhe.

[2] Görke W., Hoffman H.J.: Simulation of switching circuits by SSM- a new hardware simulation language. 1975 International Symposium on Computer Hardware Description Languages and their applications, Proceedings, pp. 125-133.

[3] Pawlak A.: Oberträger der SSM-Programmiersprache zur Beschreibung und Simulierung... Bauelementekonferenz der Mikroelektronik, Dresden 1978.

[4] Teets J.J.: The role of simulation in LSI design, Spring Joint Computer Conference 1972.

[5] Lewis D.W.: Hazard detection by quinary simulation of logic devices with bounded propagation delays. Proceedings of the 9th Design Automation Workshop, 1972.

[6] Fantauzzi G.: An algebraic model for the analysis of logical circuits, IEEE Trans. on Comp. vol. c-23, No. 6, June 1974.

[7] Chappel S., Yau S.: Simulation of large asynchronous logic circuits using an ambigous gate model. Proceedings FJCC 1971.

[8] Popiel J., Rosiński A.: Zarys koncepcji wielopoziomego symulatora struktur cyfrowych klasy mikroprocesorów, Prace IPI PAN Nr 322.

[9] Szygenda S.A.,: TEGAS2 - Anatomy of a general purpose test generation and simulation system for digital logic. Proceedings of the 9th Design Automation Workshop, 1972.

[10] Szygenda S.A: Modeling and digital simulation for design verification and diagnosis, IEEE Trans. on Comp., vol. c-25, No 12, December 1976.

[11] Szygenda S.A.: The next step in testing and modeling. Proceedings of the 13th Design Automation Workshop. 1976.

[12] Chu Y., Dietmeyer D.L., Hill F., Siewiorek D.: Introducing Computer Hardware Description Languages, Computer, vol. 7, No 12, Dec. 1974, pp. 27-44.

[13] Barbacci M.: A comparison of Register Transfer Languages for Describing Computers and Digital Systems, IEEE Trans. on Comp., vol. c-24, No 2, February 1975.

[14] Marczyński R., Pulczyn W., Sochacki J.: Język OSM- aspekty wyboru języka do symulacji sprzętu, Prace IPI PAN Nr 160.

[15] Marczyński R., Pulczyn W., Sochacki J.: OSM Microprogrammed Hardware Structure Description Language, jak 2 pp. 172-178.

[16] Pawlak A., Krawczyk J., Kupka A.: Język FDL modelowania układów cyfrowych na poziomie funkcyjnym (w przygotowaniu).

[17] Szygenda S.A.: Integrated techniques for functional and gate level digital logic simulation. Proceedings of the 10th Design Automation Workshop, 1973.

[18] Chappell S.G., Menon P.R., Pellegrin J.F., Schowe A.M.: Functional Simulation in the Lamp System. Proceedings of the 13th Design Automation Workshop.

[19] Hławiczka A., Badura D.: Multi-valued algebra to diagnostic of critical hazards, Komunikat nr 25 Instytutu Systemów Sterowania.

РЕАЛИЗАЦИЯ ЯЗЫКА МОДЕЛИРОВАНИЯ ЦИФРОВЫХ СХЕМ SSM
НА ЭВМ ODRA СЕРИИ 1300

Р е з ю м е

В статье показано реализацию языка моделирования цифровых схем SSM, язык предназначен для описания и симулирования цифровых структур на уровне вентилей. Ряд языковых механизмов поясняют приведенные примеры. В главе 3 подано формальное описание реализованного вида языка. Заключение содержит замечения по поводу ряда проблем связанных с использованием транслятора/симулятора а также намечено направление дальнейших исследований этого инструмента.

THE REALIZATION OF THE DIGITAL CIRCUIT MODELLING LANGUAGE
FOR THE ODRA 1300 SERIES OF COMPUTERS

S u m m a r y

In the paper the realization of the digital circuit modelling language SSM is presented. This language is of service to the description and simulation of the digital structures on the logic gates' level. A number of the ways of working of the realised version of the language is given in the point 3. In the concluding part a number of problems connected with the use of the translator/simulator are discussed, and the further ways of activities are marked out.