ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1980

Seria: AUTOMATYKA z. 53 Nr kol. 649

Adam PAWLAK, Janusz KRAWCZYK,
Andrzej KUPKA

JĘZYK FDL MODELOWANIA UKŁADÓW CYFROWYCH NA POZIOMIE FUNKCYJNYM

Streszczenie. W pracy przedstawiono opis języka modelowania układów cyfrowych na poziomie funkcyjnym. Zamieszczono definicję syntaktyki w notacji Backusa-Naura. Translator/symulator języka FDL umożliwia zarówno weryfikację logiczną projektu układu cyfrowego, jak i analizę jego dynamiki. W artykule zawarto opisy w języku FDL przerzutnika JK, konwertera oraz prostej maszyny cyfrowej.

1. Wprowadzenie

Modelowanie na m.c. dużych układów cyfrowych z uwagi na: ograniczoną dostępną dla użytkownika pamięć m.c., dopuszczalny ekonomicznie uzasadniony czas symulacji oraz niedostępność opisu bramkowego większości elementów LSI i VLSI natrafić może na trudności. Znaczne przyspieszenie symulacji oraz zmniejszenie wielkości niezbędnej wymaganej przez symulator pamięci operacyjnej można uzyskać stosując symulację funkcyjną [1-3]. Element cyfrowy traktowany jest wówczas jako "czarna skrzynka" [9], przy czym interesuje nas zależność sygnałów wyjściowych elementu od sygnałów wejściowych i stanu wewnętrznego (dla układów sekwencyjnych). Wprowadzenie do systemu modelowania procedur symulacji funkcyjnej implikuje opracowanie translatora/symulatora języka do opisu i symulacji układu cyfrowego na tym poziomie.

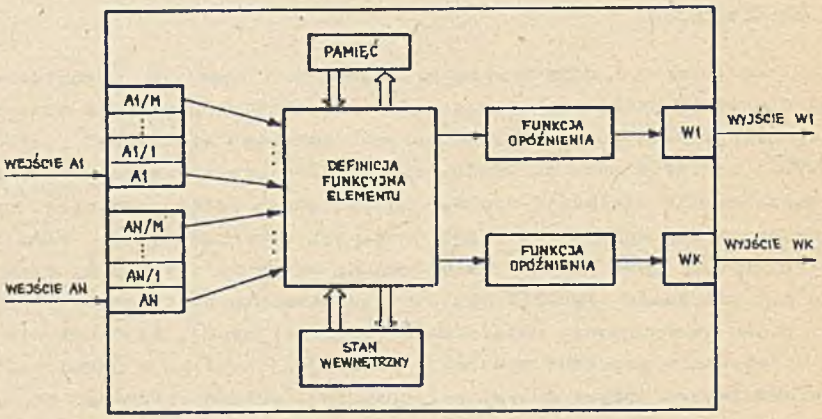
Język modelowania tego poziomu powinien umożliwiać użytkownikowi funkcyjny opis elementów MSI i LSI takich, jak np.: liczniki, rejestry, sumatory, dekodery, pamięci czy nawet procesory. Wynika z tego, że język powinien być bardzo uniwersalny, gdyż przy modelowaniu układów takich jak procesory i np. liczniki występują jakościowo różne problemy. I tak symulując układ złożony z bramek logicznych oraz elementów MSI interesować nas może oprócz weryfikacji logicznej projektu dokładna analiza jego dynamiki, z drugiej strony na poziomie procesorów, pamięci, urządzeń we/wy problemy czasowe są mniej istotne. Nacisk położony jest bowiem na weryfikację algorytmów działania systemu. Prezentowany język FDL (w aktualnej wersji) nie zawiera mechanizmów umożliwiających modelowanie układów na poziomie systemowym. Zasadniczym celem pracy było stworzenie narzędzia umożliwiającego symulację modułów funkcjonalnych wraz z bramkami logicznymi. Przyjęto mo-

del (bramki) zmiennych opóźnień z przedziałem dwuznaczności oraz z absorpcją. Zastosowana pięciowartościowa reprezentacja sygnału ma interpretację taką samą jak w translatorze/symulatorze języka SSM [6].

Znanych jest wiele języków umożliwiających opis i modelowanie układów na poziomie funkcyjnym [4,8]. W zdecydowanej większości przypadków umożliwiają one jednak bardzo uproszczoną analizę czasową układu. W prezentowanym języku wprowadzono między innymi: specjalne instrukcje warunkowe pozwalające na modelowanie reakcji układu cyfrowego na zbocze opadające lub narastające sygnału oraz możliwość uwzględnienia w opisie dynamiki elementu zarówno poprzednich jak i aktualnych stanów logicznych na końcówkach wejściowych modułu. Język FDL jest językiem nieproceduralnym.

2. Język FDL - modelowanie na poziomie funkcyjnym

Opis - program w języku FDL - elementu cyfrowego stanowi jego (rys. 1) programowy model. Użytkownik opisując moduł cyfrowy w języku powinien w



Rys. 1. Programowy model elementu cyfrowego w języku FDL

w segmencie definicji FDL-u umieścić opisy funkcyjne typów elementów wykorzystywanych w modelowanym układzie. Każdy ze zdefiniowanych typów należy zadeklarować, określając w ten sposób liczbę i nazwy stosowanych elementów. Struktura układu jest specyfikowana w bloku połączeń. Symulację można rozpocząć od dowolnych wartości zadanych w bloku warunków początkowych. Sekwencja pobudzeń wejściowych modelowanego układu określona jest za pomocą programowalnego generatora sygnałów. Instrukcja wydruku zadaje listę końcówek w układzie, których stany logiczne należy wyprowadzać na drukarkę.

Poniżej na rys. 2 przedstawiono definicję funkcyjną przerzutnika J-K. Jedną z podstawowych zalet prezentowanego narzędzia opisu jest jego prostota, która jest między innymi rezultatem abstrahowania od wewnętrznej struktury bramkowej elementu. Szczególnie uciążliwy jest przy opisie na poziomie bramkowym taki dobór opóźnień poszczególnych bramek [6], by moduł na końcówkach wejściowych i wyjściowych wykazywał parametry dynamiczne zgodne z danymi katalogowymi.

```
1            ELEM:JK;
2            WEJ:J,K,CL,S,R;
3            WYJ:Q,NQ;
4            STU:P1,P2,W1,W2;
5            HIST: 3;
6                P1=J*CL*NQ;
7                P2=K*CL*Q;
8                'IF' P1 'THEN' W1="1", W2="0";
9                'IF' P2 'THEN' W1="0", W2="1";
10               'IF' CL<1 'THEN'
11                           'IF' CL/3*CL/2*CL/1 'THEN'
12                                        Q<4,6>=W1,NQ<4,6>=W2
13                           'ELSE' Q<4,6>="X", NQ<4,6>="X";
14               'IF' -S 'THEN' Q<3,4>="1", NQ<3,4>="0";
15               'IF' -R 'THEN' Q<3,4>="0", NQ<3,4>="1";
16           END;
17           JK=JK;
18           WPC:   JK=1,0;
19                  #JK=0,0,1,0;  END;
20           JK.CL=6(8(0),8(1)),0,0,1,1;
21           JK.J=20(0),16(1),52(0),46(1);
22           JK.K=52(0),32(1),52(0);
23           JK.S=104(1),3(0);
24           JK.R=84(1),6(0),16(1);
25           DRK:JK=120;
26           STOP;
```

Rys. 2. Opis przerzutnika JK master-slave

Słowo kluczowe ELEM określa początek bloku definicji. Po nim następuje nazwa identyfikująca typ danego elementu. Kolejne instrukcje specyfikują: liczbę, nazwy i rodzaj końcówek wejściowych i wyjściowych oraz stanów wewnętrznych opisywanego elementu.
Zmienne zadeklarowane w instrukcji STP oznaczają 12-bitowe słowa pamięci. Instrukcja HIST określa liczbę poprzednich stanów logicznych na końcówkach wejściowych koniecznych do poprawnego zdefiniowania dynamiki elementu cyfrowego. Iloczyn logiczny

$$CL/1 * CL/2 * CL/3$$

umożliwia sprawdzenie stałości sygnałów na wejściu zegarowym przerzutnika w trzech ostatnich krokach symulacji.

FDL umożliwia opis funkcyjny elementu cyfrowego za pomocą równań boolowskich oraz instrukcji warunkowych IF ... THEN ... ELSE ... .
Zapis

```
'IF '- S 'THEN ' Q <3,4> = " 1 ",  NQ <3,4> = " 0 ";
```

oznacza, że jeśli zmienna logiczna -S będzie miała wartość logiczną TRUE to wtedy na końcówkach Q i NQ pojawią się z opóźnieniem (3 jednostki dla zbocza narastającego i 4 dla opadającego) stany logiczne odpowiednio "1" i "0".

W celu umożliwienia opisu elementów reagujących na zbocze sygnału, warunek instrukcji warunkowej może mieć również postać:

wyrażenie logiczne > 0 (lub < 1),

gdy element reaguje na zbocze narastające (opadające) sygnału.
Warunek:

`IF ` CL < 1 `THEN `....

uzależnia reakcję układu od zbocza opadającego na wejściu zegarowym (rys.2).

Opis konwertera liczb postaci szeregowej dwójkowej na równoległą BCD przedstawia rys. 3. W programie tym zastosowano definicję bramki, jako elementu podstawowego (wiersze 20-22).

Opóźnienie można przypisać każdej zmiennej wyjściowej, określając parametry opóźnienia w definicji funkcyjnej (rys. 3 wiersz 7) lub używając deklaracji opóźnienia (rys. 3, wiersze 14-17).

Modelowanie komunikacji elementu cyfrowego z pamięcią zapewniają instrukcje zapisu i odczytu z pamięci. Operują one na całych słowach pamięci zadeklarowanych za pomocą instrukcji STP.

Język FDL jakkolwiek ukierunkowany na funkcyjny opis modułów MSI i LSI ma pewne mechanizmy języka przesłań rejestrowych. Jego możliwości w tym zakresie prezentuje przykład prostej maszyny cyfrowej [5] (rys. 4). Rolę etykiet języka CDL (RTL) spełnia w FDL-u warunek instrukcji warunkowej IF ...THEN... Przykład ten ilustruje przypadek symulacji pojedynczego elementu. Cała bowiem prosta maszyna cyfrowa została opisana w jednym bloku. Zbędne jest wówczas specyfikowanie połączeń.

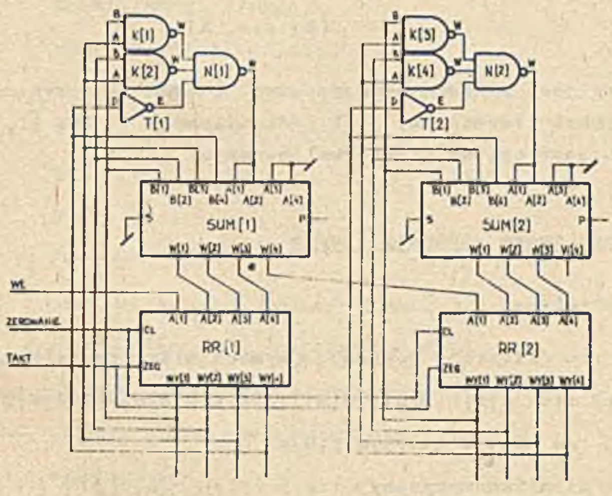
W języku FDL można również opisywać układ cyfrowy na poziomie bramkowym. Zasadnicza różnica pomiędzy opisami bramkowymi języków SSM [6] i FDL polega na tym, że ten drugi nie ma własności opisu hierarchicznego.

3. Definicja języka FDL

Do opisu formalnego syntaktyki języka została wykorzystana notacja Backusa-Naura. Ponadto przyjmujemy umowę:

<X-lista> ::= <X>|<X> , <X-lista>

<X-rząd> ::= <X>|<X> <X-rząd>




```
% KONWERTER POSTACI SZEREG. LICZBY BINARNEJ NA ROWNOLEGL. BCD
        PRZYJETO ZALOZENIE ZE DLUGOSC CYKLU WYNOSI 5[NS]

ELEM:RR;                     % DEFINICJA REJESTRU
WEJ:CL,ZEG,A[4];
WYJ:WY[4];
'IF' CL 'THEN' WY="0" 'ELSE' 'IF' ZEG>0 'THEN' WY<3>=A;
END;
ELEM:SUMATOR:                % DEFINICJA SUMATORA
WEJ:S,A[4],B[4];
WYJ:W[4],P;
P=S;
     W=A<+>B<+>P:   P=P*A+A*B+B*P;
     W[1]<5>;
     W[2]<5>;
     W[3]<5>;
     W[4]<5>;
END;
                             % DEFINICJE BRAMEK
NAND1(A,B,C,W<2>);
NAND2(A,B,W<2>);
NOT0(D,E<2>);
                             % DEKLARACJE ELEMENTOW UKLADU
RR-RR[2];
SUMATOR-SUM[2];
NAND1-N[2];
NAND2-K[4];
NOT0-T[2];
POL:                         % POLACZENIA
    RR.A[2-4]=SUM.W[1-3];
    SUM.B=RR.WY;
    SUM[1].A[1-2]=N[1];
    SUM[2].A[1-2]=N[2];
    N[1]=K[1-2],T[1];
    N[2]=K[3-4],T[2];
    T=RR.WY[4];
K[1-2]=RR[1].WY[3],RR[1].WY[1],RR[1].WY[3],RR[1].WY[2];
K[3-4]=RR[2].WY[3],RR[2].WY[1],RR[2].WY[3],RR[2].WY[2];
    RR[2].A[1]=SUM[1].W[4];
END;
WPC:                         % NADANIE WARUNKOW POCZATKOWYCH
     RR,SUM,N,K,T=20*0,6*1;  END;
                             % SEKWENCJA SYGNALOW WEJSCIOWYCH
SUM.S,SUM.A[3-4]=0;
RR.CL=7(1),153(0).
RR.ZEG-10(0),10(1);
RR[1] [illegible]15(0),20(1),20(0),40(1),20(0),40(1);
                             % DEKLARACJA WYDRUKU
D[illegible]:RR[1].CL,RR[1].ZEG,RR[1].A[1],
RR[2] [illegible],RR[2].WY[3],RR[2] WY[1],
RR[1] [illegible] WY[2],RR[1] WY[1]-160;
S[illegible];
```

Rys. 3. Modelowanie konwertera liczb w postaci szeregowej binarnej na równoległą BCD

a - schemat układu, b - opis konwertera

oraz nawiasy kwadratowe [ ] są zmiennymi metajęzykowymi o własności:

$$[X] ::= X|$$

Znak, który jest zmienną metajęzykową (< lub [) wykorzystywany w języku jako symbol terminalny jest podkreślony ($\underline{<}$ lub $\underline{[}$). Opis semantyki języka FDL jest opisem niesformalizowanym.

3.1. Podstawowe elementy 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-rząd>

<nazwa> ::= <litera> | <nazwa> <litera> | <nazwa> <cyfra>

<słowo kluczowe>::= ELEM|WEJ|WYJ|STW|STP|HIST|POL|WPC|DRK|END|STOP

3.1.2. Semantyka

Znaczenie nazw wynika z kontekstu. Spacje mogą występować w dowolnych miejscach opisu układu cyfrowego w języku.

3.2. Opis układu cyfrowego, definicja elementu

3.2.1. Syntaktyka

<program>::= <definicja-rząd>
<deklaracja-rząd>
[POL: <połączenia-rząd> END;]
[WPC: <warunki początkowe-rząd> END;]
<generator-rząd>
DRK: <wydruk> ; STOP;

<definicja> ::= <blok definicji funkcyjnej> | <definicja bramki>

<blok definicji funkcyjnej> ::= <nagłówek> <opis funkcyjny> END;

<nagłówek> ::= ELEM: <nazwa typu> ;
WEJ: <wejście - lista> ;
WYJ: <wyjście - lista> ;
[STW: <stan wewn. - lista> ;]
[STP: <stan pam. - lista> ;]
[HIST: <liczba> ;]

<nazwa typu> ::= <nazwa>

<wejście> ::= <nazwa wejścia> [[<wskaźnik max>]]

<wyjście> ::= <nazwa wyjścia> [[<wskaźnik max>]]

<stan wew.> ::= <nazwa stanu wew.> [[<wskaźnik max>]]

<stan pam.> ::= <nazwa stanu pam.> [[<wskaźnik max.>]]

<wskaźnik max> ::= <liczba>

<nazwa wejścia> , <nazwa wyjścia> , <nazwa stanu wew.> , <nazwa stanu pam.> :: = <nazwa>

<opis funkcyjny> ::= <typ opisu-rząd>

<typ opisu> ::= <opis> ; | <opis złożony> ;

<opis> :: = <równanie> | <przesłanie> | <licznik> | <deklaracja opóźnienia>

<równanie> ::= <zmienna 1> = <wyrażenie logiczne>

<zmienna 1> :: = <zmienna wyjściowa> | <zmienna stanu wew.>

<zmienna wyjściowa> ::=
<nazwa wyjścia> [[<param. pomoc.>]] [<parametry>]

<param. pomoc.> ::= <liczba> - <liczba> | <liczba>

<parametry> ::= << param. pomoc.> , <param. pomoc.>> |
<< param. pomoc.>>

<zmienna stanu wew.> ::= <nazwa stanu wew.> [[<param. pomoc.>]]

<wyrażenie logiczne> :: = <stan logiczny> | <wyrażenie>

<stan logiczny> ::= "<wartość logiczna>"

<wartość logiczna> ::= 0 | E | X | A | 1

<wyrażenie> ::= <zmienna 2> | <wyrażenie> <operator> <wyrażenie> |
(<wyrażenie>)

<zmienna 2> ::= <zmienna wejściowa> | <nazwa wyjścia> [[<param.pomoc.>]] |
<zmienna stanu wewn>

<zmienna wejściowa> ::= <nazwa wejścia> [[<param. pomoc.>]] [/ <liczba>]

<operator> ::= - | * | + | < + > | < = > | =

<przesłanie> ::= <zapis> | <odczyt> | <przesłanie proste>

<zapis> ::= <zmienna typu pamięć> ← <zmienna 2 - lista>

<zmienna typu pamięć> ::= <nazwa stanu pam.> [[<liczba>]] |
<nazwa stanu pam.> [[<zmienna 2 - lista>]]

<odczyt> ::= <zmienna 1 - lista> ← <zmienna typu pamięć>

<przesłanie proste> ::= <zmienna 1 - lista> ← <zmienna 2 - lista>

<licznik> ::= <przedział> +1 | <przedział> -1

<przedział> ::= <nazwa stanu wew.> [[<liczba> - <liczba>]] |
<nazwa wyjścia> [[<liczba> - <liczba>]]

<deklaracja opóźnienia> ::= <nazwa wyjścia> [[<param.pomoc.>]] <parametry>

<opis złożony> ::= 'IF' <warunek> 'THEN' <łańcuch opisu> |
'IF' <warunek> 'THEN' <łańcuch opisu> 'ELSE' <łańcuch opisu>

<warunek> ::= <wyrażenie> | <równanie> | <wyrażenie> ≥ 0 | <wyrażenie> ≤ 1

<łańcuch opisu> ::= <opis - lista> | <opis złożony>

<definicja bramki> ::=
<typ bramki> (<wejście - lista>, <nazwa wyjścia> [<parametry>]);

<typ bramki> ::= <rodzaj bramki> <wskaźnik typu>

<rodzaj bramki> ::= NOT | AND | NND | ORR | NOR | XOR

<wskaźnik typu> ::= <cyfra> | <litera>

3.2.2. Semantyka

Podstawowe mechanizmy bloku definicji języka zostały omówione w punkcie 2. Dodatkowo wprowadzono instrukcję licznika. Pozwala ona na zwarty zapis prostych liczników binarnych dzięki możliwości inkrementowania lub dekrementowania stanu licznika reprezentowanego przez <przedział>. Definiując bramkę, jako element podstawowy użytkownik podaje: typ bramki, liczbę i nazwy wejść, nazwę wyjścia oraz parametry opóźnień. Wskaźnik typu identyfikuje bramki w ramach tego samego rodzaju. W przypadku opóźnień jednostkowych można pominąć podawanie ich parametrów.

3.3. Deklaracja elementów

3.3.1. Syntaktyka

<deklaracja> ::= <typ elementu> - <deklaracja nazw - lista> ;

<typ elementu> ::= <nazwa typu> | <typ bramki>

<deklaracja nazw> ::= <nazwa elementu> [[<wskaźnik max>]]

<nazwa elementu> ::= <nazwa>

3.3.2. Semantyka

Deklaracja określa liczbę używanych w układzie elementów o tej samej definicji oraz nadaje im nazwy pod jakimi będą używane w opisie. Typ elementu musi być wcześniej w programie określony albo w bloku definicji, albo w definicji bramki.

3.4. Połączenia

3.4.1. Syntaktyka

<połączenia> ::= <końcówka wejściowa-lista> = <końcówka wyjściowa-lista>;

<końcówka wejściowa> ::= <końcówka>

<końcówka wyjściowa> ::= <końcówka> | <stała>

<końcówka> ::= <zmienna elementu> [. <nazwa końcówki> [[<param. pomoc.>]]]

<stała> ::= 0 | 1 | X

<zmienna elementu> ::= <nazwa elementu> [[<param. pomoc.>]]

<nazwa końcówki> ::= <nazwa wejścia> | <nazwa wyjścia>

3.4.2. Semantyka

Translator przyporządkowuje sobie końcówki wejściowe i wyjściowe znajdujące się na analogicznych pozycjach w listach wejść i wyjść.
W przypadku gdy jedna końcówka wyjściowa "zasila" kilka wejść, to po prawej stronie znaku równości występuje tylko jedna nazwa końcówki. Jeżeli użyta zostanie tylko nazwa elementu, to do listy zostaną dołączone nazwy wszystkich końcówek wejściowych lub wyjściowych w zależności od tego, na której liście ta nazwa wystąpiła.

3.5. Wprowadzenie warunków początkowych.

3.5.1. Syntaktyka

<warunki początkowe> ::= <końcówka wyjściowa-lista> = <wartość-lista> ; |
<zmienna elementu-lista> = <zawartość-lista>;

<wartość> ::= <stała> | <liczba> * <stała>

<zawartość> ::= <wartość-lista> | <program pamięci>
<wartość-lista> , <program pamięci>
<program pamięci> ::= <słowo-lista>
<słowo> ::= <słowo proste> | <liczba>*<słowo proste>
<słowo proste> ::= <liczba> D | (<liczba dwójkowa>)
<liczba dwójkowa> ::= <stała-rząd>

3.5.2. Semantyka

Instrukcja nadania warunków początkowych przyporządkowuje wartości logiczne końówkom wyjściowym lub stanom wewnętrznym elementów. Wprowadzenie początkowych wartości stanów wewnętrznych lub stanów pamięci rozpoczyna się znakiem #, po którym wolno podać tylko nazwy elementów, gdyż nazwy tych stanów są niedostępne poza blokiem definicji. W razie nie podania warunków początkowych symulacja rozpocznie się standardowo od wartości niezdeterminowanych X.

3.6. Generatory przebiegów wejściowych

3.6.1. Syntaktyka

<generator> ::= <końówka wejściowa - lista> = <sekwencja - lista> ;
<sekwencja> ::= <wartość logiczna> | <liczba> (<sekwencja - lista>)

3.6.2. Semantyka

Na każdą z końówek wejściowych z listy końówek będą wprowadzane wartości sygnałów logicznych kolejno, co krok symulacji, zgodnie z podaną sekwencją. Liczba sygnałów logicznych nie musi być zgodna z liczbą kroków symulacji. W przypadku wyczerpania się wartości z listy sygnałów, będą one pobierane powtórnie od początku listy.

3.7. Wyprowadzanie wyników - wydruk przebiegów sygnałów

3.7.1. Syntaktyka

<wydruk> ::= <końówka - lista> <postać> <liczba>
<postać> ::= = + | -

3.7.2. Semantyka

Dopuszcza się drukowanie wartości stanów końcówek wejściowych i wyjściowych w dwóch postaciach: znakowej (+) lub jako wykres czasowy (-). Zapis instrukcji jest kończony przez podanie liczby kroków symulacji.

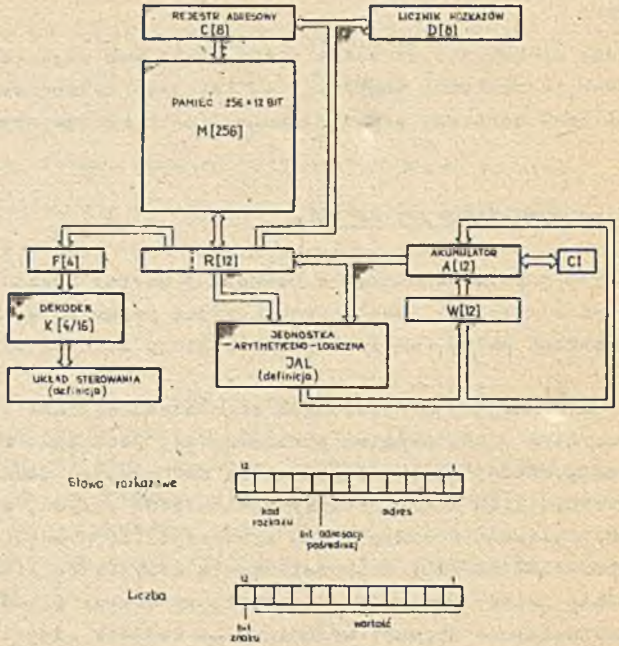
4. Translator/symulator języka FDL

Dla języka FDL zaprezentowanego w punkcie 3 został zrealizowany program translatora/symulatora. Część translacyjna programu tłumaczy zdania języka na wewnętrzny układ tablic (rys. 5), który stanowi bazę danych dla symulatora.

Podstawowe informacje o modelowanych elementach są zawarte w tablicach MDF i KAB, przy czym w tej drugiej przechowywany jest opis funkcyjny elementu w odwrotnej notacji polskiej. Tablica KDT zawiera wskaźniki selektywnego śladu wszystkich zadeklarowanych elementów układu, które kwalifikują element do obliczeń w danym kroku symulacji. Informacje o końcówkach elementu zawiera tablica KED. Po przeliczeniu wszystkich elementów (stosu w KAB) na wejścia układu wprowadza się następny wektor pobudzeń oraz przepropagowuje uaktualnione sygnały wyjściowe na wejścia odpowiednich elementów, zgodnie ze strukturą połączeń układu (KDV). Przeprowadza się również kwalifikację elementów do przeliczania w następnym kroku symulacji. W przypadku wykorzystania opóźnień w opisie modelowego układu jego funkcja opóźnień przeliczana jest zawsze, niezależnie od poprzedniej kwalifikacji. Okazuje się bowiem, że w przypadku przyjętego modelu opóźnień przeliczanie funkcji opóźnienia jest szybsze od kwalifikacji.

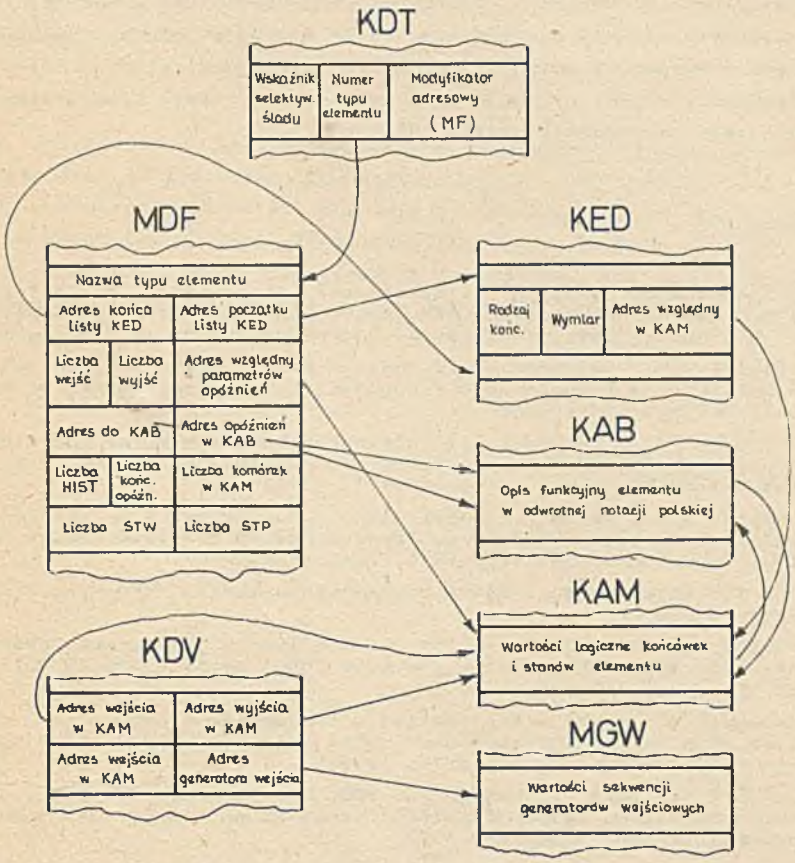
5. Zakończenie

Program translatora/symulatora języka FDL napisany został w FORTRAN-ie i był implementowany na m.c. ODRA 1325. Umożliwia on modelowanie układów cyfrowych złożonych maksymalnie z 500 elementów. Dla układów o średniej wielkości szybkość symulacji ograniczona jest prędkością pracy drukarki, na którą wyprowadzone są wyniki modelowania. Zaprezentowany język FDL nie stanowi wersji ostatecznej języka modelowania na poziomie funkcyjnym. Planuje się w oparciu o doświadczenia z eksploatacji systemu dalszą rozbudowę języka (translatora/symulatora). Z uwagi na brak w oprogramowaniu m.c. ODRA języka programowania wysokiego rzędu umożliwiającego między innymi opis modelowanego sprzętu przewidujemy nadanie językowi (w ograniczonym zakresie) cech języka proceduralnego. Przewiduje się również wykorzystanie pewnych mechanizmów języka FOL w translatorze/symulatorze języka SSM.




```
                                          % MASZYNA CHU
FLEM:MC;
WEJ:CL,STRT,STOP;                         % KLAWISZE
WYJ:A[12],                                % AKUMULATOR
    R[12],                                % REJESTR BUFOROWY
    C[8],                                 % REJESTR ADRESOWY
    F[4],                                 % REJESTR STEROWANIA
    G,                                    % REJESTR START/STOPU
    K[1,];                                % DEKODER ROZKAZU
STW:D[8],                                 % LICZNIK ROZKAZOW
    CI,CK,W[12];                          % ZMIENNE POMOCNICZE
STP:M[256]:                               % PAMIEC PROGRAMU
      'IF' STRT 'THEN' G="1";
      'IF' STOP 'THEN' G="0";
       C+D, R+M[C];                       % POBIERZ ROZKAZ
       D[1-8]+1;                          % ZWIEKSZ LICZNIK
       F[1-4]+R[9-12];                    % POBIERZ KOD ROZKAZU
       C[1-7]+R[1-7], C[8]="0";           % PRZESLIJ ADRES ARG.
       'IF' R[8] 'THEN' R+M[C], C[1-8]+R[1-8];
       K="0", K[F]="1";                   % DEKODUJ ROZKAZ
   'IF' G*K[2]  'THEN'                    % ROZKAZ ADD
                 R+M[C],     CI="0",
                 W=A<+>R<+>CI:  CI=CI*A+A*R+R*CI,  A=W;
   'IF' G*K[3]  'THEN'                    % ROZKAZ SUB
                 R+M[C], R[1-12]=-R[1-12], R[1-12]+1, CI="0",
                 W=A<+>R<+>CI:  CI=CI*A+A*R+R*CI,  A=W;
   'IF' G*K[4]  'THEN'                    % ROZKAZ STO
                 R+A,  M[C]+R;
   'IF' G*K[5]  'THEN'                    % ROZKAZ CLA
                 R+M[C],  A+R;
   'IF' G*K[6]  'THEN'                    % ROZKAZ JMP
                 D[1-8]+R[1-8];
   'IF' G*K[7]  'THEN'                    % ROZKAZ JOM
                 'IF'  A[12]  'THEN' D[1-8]+R[1-8];
   'IF' G*K[8]  'THEN'                    % ROZKAZ SHR
                 A[1-11]+A[2-12], A[12]="0";
   'IF' G*K[9]  'THEN'                    % ROZKAZ CIL
                 CI=A[12], A[2-12]+A[1-11], A[1]=CI;
   'IF' G*K[10] 'THEN'                    % ROZKAZ SHL
                 A[2-12]+A[1-11], A[1]="0";
   'IF' G*K[11] 'THEN'                    % ROZKAZ CIR
                 CI=A[1], A[1-11]+A[2-12], A[12]=CI;
   'IF' G*K[12] 'THEN'                    % ROZKAZ STP
                 G="0";
   'IF' G 'THEN' D[1-8]=C;
END;
MC-[?],
WPC: M[?]5*0;
   ØM[ ?*0,1,?,1,?*0,                     % PROGRAM
          ?*0,?D,2D,3D,4D,5D,6D,7D,8D,9D,10D,
           ?D,?*0D,?D,??0D,1029D,4116D,1585D,8010D,1056D,
           2650,80??D,??7D,1321D,6772D,2816D,205*0D;
END;
MC.CL=0,1; MC.STOP=100(0), MC.STRT=100(1);
DRK:MC-100(3(A1,2X),7HAKU,2X,12A1,2X,5HBUF:R,2X,12A1,2X,5HADR,
2X,8A1,2X,6HR.STER,2X,4A1,2X,2HG:,2X,A1,2X,3HDEK,2X,16A1);
STOP;
```

a – schemat blokowy, b – opis w języku FDL



Rys. 5. Ogólna struktura tablicowa translatora/symulatora

W Instytucie Elektroniki prowadzone są jednocześnie prace nad modelowaniem złożonych układów sterujących. Część sterująca takiego układu będzie opisywana w wyspecjalizowanym języku wykorzystującym grafy skierowane[7]. Do modelowania części wykonawczej urządzenia cyfrowego wykorzystana zostanie rozszerzona wersja języka FDL.

LITERATURA

[1] Szygenda S.A., Hemming C.: Functional Simulation, A Basis for a Systems Approach to Digital Simulation and Fault Diagnosis.Proceedings of the Annual Summer Simulation Conference, June 1972.

[2] Hemming C.W., Szygenda S.A.: Modular Requirements for Digital Logic Simulation at a Predefined Functional Level. Proceedings of the 1972 National ACM Conference.

[3] Szygenda S.A., Lekkos A.A.: Integrated Techniques for Functional and Gate-Level Digital Logic Simulation. Proceedings of the 10th Design Automation Workshop, 25-30.06.1973, Portland.

[4] Chappel S.G., Menon P.R., Pellegrin J.F., Schowe A.M.: Functional Simulation in the LAMP System. Proceedings of the 13th Design Automation Conference, 28-30.06.1976, San Francisco.

[5] Chu Y.: Organizacja i mikroprogramowanie maszyn cyfrowych. WNT, Warszawa 1979.

[6] Pawlak A., Jeżewski J., Skiba J.: Realizacja języka modelowania układów cyfrowych SSM na maszyny cyfrowe ODRA serii 1300. ZN Politechniki Śląskiej, Automatyka z. 53.

[7] Zachariades M.: MAS: Realisation d'un langage d'aide a la description et a la conception des systèmes logiques. These, Institut National Polytechnique de Grenoble, 1977.

[8] Anlauff H., Funk P., Meinen P.: PHPL - A language for logic design and simulation. Third EUROMICRO Symposium on Microprocessing and Microprogramming, Amsterdam 1977.

[9] Lipovski G.J.: On gray box descriptions of microprocessors. Proceedings of the 1975 International Symposium on Computer Hardware Description Languages and their Applications, New York 1975.

ЯЗЫК FDL МОДЕЛИРОВАНИЯ ЦИФРОВЫХ СХЕМ НА ФУНКЦИОННОМ УРОВНЕ

Р е з ю м е

В статье представлено описание языка моделирования цифровых схем на функционном уровне. Дано определение синтаксиса в записи Бакуса-Наура. Транслятор/симулятор языка FDL даёт возможность логической проверки проекта цифровой схемы и анализа его динамики. Статья содержит описание на языке FDL тригера JK, конвертера а также простой ЭВМ.

THE FDL LANGUAGE FOR THE FUNCTIONAL MODELLING
OF THE DIGITAL CIRCUITS

S u m m a r y

The description of the language for the functional modelling of the digital circuits is presented. The definition of the syntax in Backus-Naur's notation is inserted. The translator/simulator of the FDL language makes possible the logical verification of the digital circuit design as well as the analise of its dynamic. In the paper the description in the FDL language of the JK flip-flop, the converter and the simple computer are contained.