# Układy PLD część 2

## Oprogramowanie - CUPL

*Po sporej dawce teorii dotyczącej układów PLD, zamieszczonej w poprzedniej EP, proponujemy teraz zgłębienie strony praktycznej - programowania tych układów.*

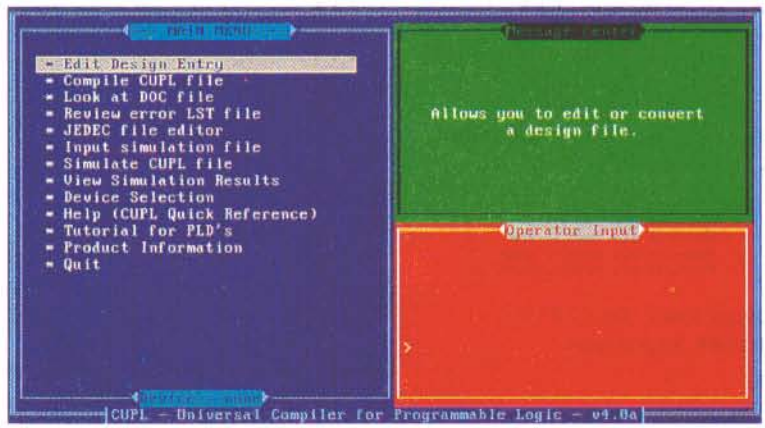
Spośród wielu języków programowania układów PLD wybraliśmy do dokładniejszego omówienia dwa:
- CUPL firmy DATA I/O;
- PALASM firmy Advanced Micro Devices.

W tym artykule omawiamy język CUPL, natomiast PALASM przedstawimy za miesiąc.

Języki te różnią się między sobą składnią, sposobem opisu układów kombinacyjnych i sekwencyjnych, a przede wszystkim uniwersalnością zastosowań.

Niezależnie od złożoności, szybkości, wykorzystanych algorytmów minimalizacji i jej efektywności, wszystkie programy kompilujące dla układów PLD mają za zadanie przetworzenie tekstowego opisu logicznego funkcji w plik standardu JEDEC, który zawiera wszystkie informacje niezbędne do poprawnego zaprogramowania układu. Struktura tego pliku zostanie omówiona w dalszej części artykułu.

Język CUPL został zaprojektowany jako uniwersalne narzędzie do projektowania praktycznie wszystkich dostępnych na świecie układów PLD, włącznie z pamięciami PROM, układami MACH, itp.




PALASM jest to niejako język „wewnętrzny" firmy AMD - jego biblioteki są ograniczone do układów oferowanych przez AMD.

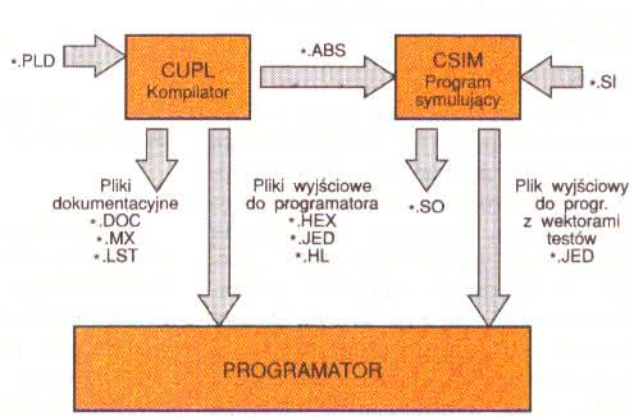
Chcąc ułatwić poznanie i zrozumienie podstawowych struktur tych języków, sposobu opisu równań, a także charakterystycznej struktury pliku tekstowego przeznaczonego do kompilacji, przedstawione zostaną po trzy przykłady o rosnącym poziomie trudności dla obydwu języków. Przykłady będą jednakowe, co umożliwi bezpośrednie spostrzeżenie różnic. Analizę podanych przykładów (w przypadku CUPL'a) znacznie ułatwia oprogramowanie freeware (wersja demo CUPL'a) oferowane przez AVT. Oprócz wersji DEMO oferujemy półprofesjonalną wersję CUPL PalStart, która posiada możliwości kompilacji projektów (programów) z możliwością wyboru układu docelowego.

Taka „żywa" edukacja, naszym zdaniem, daje o wiele większe efekty od standardowych wykładów teoretycznych, co powinno przyczynić się do powiększania grona zwolenników praktycznych zastosowań układów PLD.

Realizowanymi kolejno przykładami, będą:
- podstawowe bramki (funktory) logiczne (AND, NAND, OR, NOR, ExOR i ExNOR - sześć typów bramek w jednym układzie scalonym),
- dekoder kodu binarnego (NKB)

```
Name            Jakis_Uklad;
Partno          CA1010;
Revision        1.1;
Date            2/11/93;
Designer        Autor;
Company         Moja_Firma;
Assembly        PC Multi-Function;
Location        U33;
Format          J;
Device          P16P8;

/* ************************************************************** */
/* *          To jest blok tytulowy, powyzej znajduje sie naglowek * */
/* ************************************************************** */

/**            Inputs - deklaracja wejsc projektowanego ukladu      **/

pin [1..8] = [a9..2] ;                  /* system addresses a2 - a9    */
pin 9      = !memw ;                    /* memory write strobe         */

/**            Outputs - deklaracja wyjsc projektowanego ukladu     **/

pin 13     = !ioacc ;                   /* on-board I/O being accessed */
pin 14     = !parport ;                 /* parallel port chip select   */

/**          Declarations and Intermediate Variable Definitions     **/
/**                    Pole deklaracji zmiennych                    **/

field   ioaddr     = [a9..2] ;
serport1_eqn       = ioaddr:[2F8..2FF] ;

/**                         Logic Equations                         **/
/**                Pole definiowania rownan logicznych              **/

wrbuff             = memacc & memw # ioacc_in & iow ;
parport            = parport_eqn ;
```

List. 1. Format pliku wejściowego dla CUPL'a




Rys. 1. Obieg informacji w CUPL'u

```
Name     Bramki;
Partno   2;
Date     11/14/93;
Revision 1.0;
Designer P.Z.;
Company  BTC;
Location US1;
Device   G16V8;

/* ************************************************** */
/* * Projekt realizacji szesciu funktorow logicznych:* */
/* *        AND, NAND, OR, NOR, ExOR, ExNOR         * */
/* *              w ukladzie GAL16V8                * */
/* ************************************************** */

/*  Inputs  */
PIN [1,2] = [A,B];    /* Wejscia bramki AND     */
PIN [3,4] = [C,D];    /* Wejscia bramki NAND    */
PIN [5,6] = [E,F];    /* Wejscia bramki OR      */
PIN [7,8] = [G,H];    /* Wejscia bramki NOR     */
PIN [9,11] = [I,J];   /* Wejscia bramki ExOR    */
PIN [12,13] = [K,L];  /* Wejscia bramki ExNOR   */

/**  Outputs  **/
PIN 19 = Y1;          /* Wyjscie bramki AND     */
PIN 18 = Y2;          /* Wyjscie bramki NAND    */
PIN 17 = Y3;          /* Wyjscie bramki OR      */
PIN 16 = Y4;          /* Wyjscie bramki NOR     */
PIN 15 = Y5;          /* Wyjscie bramki ExOR    */
PIN 14 = Y6;          /* Wyjscie bramki ExNOR   */

/** Equations  **/
Y1 = A & B;       /* Rownanie dla funkcji AND   */
Y2 = !(C & D);    /* Rownanie dla funkcji NAND  */
Y3 = E # F;       /* Rownanie dla funkcji OR    */
Y4 = !(G # H);    /* Rownanie dla funkcji NOR   */
Y5 = I $ J;       /* Rownanie dla funkcji ExOR  */
Y6 = !(K $ L);    /* Rownanie dla funkcji ExNOR */
```

List. 2a. Plik wejściowy dla CUPL'a. Realizacja bramek logicznych.

na kod wyświetlacza 7-mio segmentowego (włącznie ze znakami A-F w przypadku dekodowania liczb heksadecymalnych); przyjęto iż wyświetlacz będzie miał wspólną anodę,

- licznik dziesiętny, synchroniczny, z wyjściem przeniesienia, wejściem zmiany kierunku zliczania i synchronicznym wejściem kasującym; jest to przykład układu sekwencyjnego, opisanego za pomocą kodowania stanów.

Opis programów kompilujących nie będzie dotyczył ich konkretnych wersji, ponieważ różnice pomiędzy starszymi a nowszymi polegają między innymi na bogatszej oprawie „shella", rozbudowie bibliotek, zwiększeniu skuteczności i szybkości minimalizacji, natomiast struktury języków i sposoby opisu logicznego ulegają stosunkowo nieznacznym przeobrażeniom.

## CUPL

CUPL jest skrótem nazwy Universal Compiler for Programmable Logic, która znakomicie odzwierciedla wszechstronność zastosowań tego kompilatora, stającego się obecnie standardem światowym (taki „Pascal" w dziedzinie układów programowalnych). Na **rys. 1** przedstawiona jest struktura oprogramowania wchodzącego w skład kompilatora CUPL wraz z zaznaczonym obiegiem informacji pomiędzy poszczególnymi programami.

```
*****************************************
                Bramki
*****************************************

CUPL            4.0a Serial# MD-40A-8209
Device          g16v8s  Library DLIB-h-26-9
Created         Thu Oct 28 13:13:27 1993
Name            Bramki
Partno          2
Revision        1.0
Date            11/14/93
Designer        P.Z.
Company         BTC
Assembly        xxxxxx
Location        US1

=========================================
      Expanded Product Terms
=========================================

Y1 =>
    A & B

Y2 =>
    C & D

Y3 =>
    E
  # F

Y4 =>
    G
  # H

Y5 =>
    I & !J
  # !I & J

Y6 =>
    K & !L
  # !K & L
```

```
=============================================================
                        Symbol Table
=============================================================

Pin Variable                                  Pterms   Max     Min
Pol   Name      Ext      Pin     Type         Used     Pterms  Level
---   ----      ---      ---     ----         -----    ------  -----

      A                   1       V          -        -       -
      B                   2       V          -        -       -
      C                   3       V          -        -       -
      D                   4       V          -        -       -
      E                   5       V          -        -       -
      F                   6       V          -        -       -
      G                   7       V          -        -       -
      H                   8       V          -        -       -
      I                   9       V          -        -       -
      J                   11      V          -        -       -
      K                   12      V          -        -       -
      L                   13      V          -        -       -
      Y1                  19      V          1        8       1
      Y2                  18      V          1        8       1
      Y3                  17      V          2        8       1
      Y4                  16      V          2        8       1
      Y5                  15      V          2        8       1
      Y6                  14      V          2        8       1

LEGEND    F : field          D : default variable        M : extended node
          N : node           I : intermediate variable   T : function
          V : variable       X : extended variable       U : undefined
=============================================
                Chip Diagram
=============================================

                  ____________
                 |   Bramki   |
              A x-|1        20|-x Vcc
              B x-|2        19|-x Y1
              C x-|3        18|-x Y2
              D x-|4        17|-x Y3
              E x-|5        16|-x Y4
              F x-|6        15|-x Y5
              G x-|7        14|-x Y6
              H x-|8        13|-x L
              I x-|9        12|-x K
            GND x-|10       11|-x J
                  |____________|
```

List. 2b. Plik dokumentacyjny (*.DOC) dla programu z list. 2a

Konstrukcja tego języka pozwala na bardzo elastyczną budowę opisu realizowanego projektu. Możliwe jest definiowanie funkcji logicznych realizowanych przez układ za pomocą:

- równań boolowskich z wykorzystaniem wszystkich podstawowych operacji logicznych - AND [&], OR [#], ExOR [$] oraz NOT [!].

Przykład: Y = A0 & A3 # !B1 # B0 & A2;

- operacji listowych, które czytelnym zapisem pozwalają tworzyć dość złożone równania.

Przykład: [A0, A1, A2, .... A10] OP St,

gdzie OP=& lub # lub ! lub $, natomiast St - dowolna stała. Zapis ten jest równoważny: [A0 OP St, A1 OP St, ..., A10 OP St];

- operacji przyrównania (operacja dwuargumentowa) - podobnie jak operacje listowe, ale dla dwóch list argumentów.

Przykład: zapis [I0..I3] : [A..C], gdzie A..C oznacza zakres liczb zapisany w kodzie heksadecymalnym, jest równoważny:

[I0..I3]:A # [I0..I3]:B # [I0..I3]:C, a po rozwinięciu:

(I0 & !I1 & I2 & !I3) #

(I0 & ! I1 & I2 & I3) #

(I0 & I1 & !I2 & !I3);

- operacji przypisania (tabela prawdy), jest narzędziem bardzo ułatwiającym bezpośrednie przeniesienie opisu urządzenia z tablicy prawdy z pominięciem etapu tworzenia równań i minimalizacji funkcji logicznych - czynności te wykonuje kompilator.

Przykład:

```
TABLE ListWe => ListWy
StWe0 => StWy0;
StWe1 => StWy1;
.....
.....
StWeN => StWyN;
,
```

gdzie TABLE jest słowem kluczo-

```
PARTNO US2;
NAME DeC_HEX;
REV 1.4;
DATE 12/10/93;
DESIGNER Piotr Zbysinski;
ASSEMBLY N/A;
COMPANY BTC;
LOCATION N/N;
DEVICE G16V8;
/* ********************************************** */
/* *    Dekoder wyswietlacza ze wspolna anoda   * */
/* *        dekodujacy znaki 0..F (HEX).        * */
/* *         W projekcie wykorzystano           * */
/* *              TABLICE PRAWDY                * */
/* ********************************************** */

/*   INPUTS                                       */
PIN [1..4] = [A3..A0];

/*   OUTPUTS
PIN[13..19] = [A,B,C,D,E,F,G];

/*DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS*/
Field Dana = [A3..A0];
Field Segment = [A,B,C,D,E,F,G];

/*   LOGIC EQUATIONS
/* Do zdefiniowania funkcji wyjsc */
/* zastosowano tablice prawdy */
Table Dana => Segment {

/*  Wejscia     Wyjscia segmentowe
/*  ____________                                 */
/*  AAAA          ABCDEFG
/*  3210

  'b'0000   =>  'b'0000001;
  'b'0001   =>  'b'1001111;
  'b'0010   =>  'b'0010010;
  'b'0011   =>  'b'0000110;
  'b'0100   =>  'b'1001100;
  'b'0101   =>  'b'0100100;
  'b'0110   =>  'b'0100000;
  'b'0111   =>  'b'0001111;
  'b'1000   =>  'b'0000000;
  'b'1001   =>  'b'0000100;
  'b'1010   =>  'b'0001000;
  'b'1011   =>  'b'1100000;
  'b'1100   =>  'b'0110000;
  'b'1101   =>  'b'1000010;
  'b'1110   =>  'b'0110000;
  'b'1111   =>  'b'0111000;
}
```

List. 3a. Plik wejściowy (*.PLD) do CUPL'a. Realizacja dekodera wyświetlacza.

wym kompilatora, StWeN - stała wejściowa (stany wejść) dla stanu wejść równym N, StWyN wartość stałej wyjściowej N.

Przykład zastosowania tej operacji jest przedstawiony w realizacji dekodera.

- sekwencji stanów w opisie układów sekwencyjnych (automatów).

Przykład:

```
SEQUENCE zmienne stanu
PRESENT Stan_0 PInst;
.....
.....
PRESENT Stan_N PInst;
```

, gdzie SEQUENCE oraz PRESENT są słowami kluczowymi, natomiast PInst oznacza podinstrukcję.

Możliwe są następujące rodzaje podinstrukcji:

- instrukcja zmiany stanu - NEXT Stan_M;

- instrukcja asynchronicznego wyprowadzenia sygnału - OUT St1..OUT StN;

- instrukcja synchronicznego wyprowadzenia sygnału - NEXT Stan_M OUT St1..OUT StN;

```
*******************************************
                  DeC_HEX
*******************************************
CUPL            4.0a Serial# MD-40A-8209
Device          g16v8s  Library DLIB-h-26-9
Created         Thu Oct 28 13:13:41 1993
Name            DeC_HEX
Partno          US2
Revision        1.4
Date            12/10/93
Designer        Piotr Zbysinski
Company         BTC
Assembly        N/A
Location        N/N
===========================================
                Expanded Product Terms
===========================================
A =>
    A0 & !A1 & A2 & A3
  # A0 & !A1 & !A2 & !A3
  # !A0 & !A1 & A2 & !A3
  # A0 & A1 & !A2 & A3

B =>
    !A0 & !A1 & A2 & A3
  # A0 & !A1 & A2 & !A3
  # !A0 & A1 & A2
  # A0 & A1 & A3

C =>
    A0 & A1 & A2 & A3
  # !A0 & A1 & !A2 & !A3
  # !A0 & A2 & A3
  #
D =>
    !A0 & A1 & !A2 & A3
  # A0 & !A1 & !A2 & !A3
  # !A0 & !A1 & A2 & !A3
  # A0 & A1 & A2

Dana =>
    A3 , A2 , A1 , A0

E =>
    A0 & !A1 & !A2 & A3
  # A0 & !A2 & !A3
  # !A1 & A2 & !A3
  # A0 & A1 & A2 & !A3

F =>
    A0 & !A1 & A2 & A3
  # A0 & !A2 & !A3
  # !A0 & A1 & !A2 & !A3
  # A0 & A1 & A2 & !A3

G =>
    !A1 & !A2 & !A3
  # A0 & A1 & A2 & !A3

Segment =>
    A , B , C , D , E , F , G

===========================================
                Chip Diagram
===========================================

                 ____________
                |  DeC_HEX   |
          A3 x-|1          20|-x Vcc
          A2 x-|2          19|-x G
          A1 x-|3          18|-x F
          A0 x-|4          17|-x E
             x-|5          16|-x D
             x-|6          15|-x C
             x-|7          14|-x B
             x-|8          13|-x A
             x-|9          12|-x
         GND x-|10         11|-x
                |____________|
```

List. 3b. Plik dokumentacyjny (*.DOC) do programu z list. 3a.

- stosować instrukcje warunkowego wyprowadzania (asynchronicznego i synchronicznego) sygnałów, a także warunkowej zmiany stanu.

- zdefiniowane funkcje, które mogą być wywoływane w dowolnym miejscu programu, przez funkcje ewentualnymi parametrami.

Przykład:

```
FUNCTION and (We1, We2) {
  and = We1 & We2;
}
```

Wywołanie takiej funkcji wygląda następująco:

Wy = and (ZmA, ZmB);, co oznacza Wy = ZmA & ZmB, gdzie

We1, We2 - parametry wejściowe;

Wy - funkcja realizowana na danym wyjściu;

ZmA, ZmB - zmienne wejściowe.

## Format pliku wejściowego kompilatora CUPL

Na **list. 1** przedstawiono podstawową strukturę pliku wejściowego (jest to plik z rozszerzeniem domyślnym *.PLD).

Jak widać plik składa się z sześciu podstawowych części:

- bloku nagłówkowego - zawiera on podstawowe informacje o projektowanym układzie, tzn. nazwę układu (NAME), numer elementu (PARTNO), wersję projektu (REVISION lub REV), datę wykonania (DATE), projektanta (DESIGNER), firmę projektującą (COMPANY), nazwę projektu, dla którego projektowany jest układ (ASSEMBLY lub ASSY), położenie układu na płytce (LOCATION lub LOC), typ układu, dla którego kompilator będzie projektował plik wyjściowy (DEVICE), format pliku wyjściowego (FORMAT). Nie jest konieczne wypełnianie tego „formularza", ale z praktyki autora wynika iż wprowadzanie poprawek do projektów z pominięciem uaktualnienia, np. numeru wersji, prowadzi często do ogromnego nieładu. Jeżeli kkompilator wykryje puste miejsce w formularzu, generuje ostrzeżenie ale kompilacja odbywa się bez przeszkód;

- bloku tytułowego - swobodny opis słowny projektu, dodatkowe uwagi, komentarze. Wypełnianie tego pola nie stwarza żadnych formalnych wymagań, ponieważ tekst jest ograniczony znakami komentarza (para znaków „/*"oraz"*/")

- blok deklaracji wejść i wyjść układu - deklaracje poprzedzone są słowem kluczowym PIN, następnie znajduje się numer wyprowadzenia układu scalonego, znak równości i nazwa sygnału wyjściowego.

Osobno deklarowane są wejścia („/* INPUTS */") i wyjścia („/* OUTPUTS */") układu, przy czym kompilator sam „wnioskuje", czy dany pin jest wejściem, czy wyjściem - poprzedzenie deklaracji takim komentarzem służy tylko wygodzie projektanta;

- blok deklaracji pozostałych zmiennych - nie zawsze konieczny,

```
Name     Licznik;
Partno   US3;
Date     28/10/93;
Revision 1.9;
Designer P.Z.;
Company  BTC;
Device   g16v8;

/*************************************************** */
/* *              Licznik dziesietny (dekada).       * */
/* *                                                 * */
/* * Wejscie RES jest synchronicznym wejsciem kasujacym - * */
/* *                stan aktywny "1".                * */
/* *        Wejscie DIR ustala kierunek zliczania:   * */
/* *          - dla DIR = 0 licznik zlicza w gore;   * */
/* *          - dla DIR = 1 licznik zlicza w dol.    * */
/* *************************************************** */

/** Inputs **/
pin 1 = CLK;        /* Wejscie zegarowe                */
pin 3 = RES;        /* Wejscie kasujace (synchroniczne)*/
pin 5 = DIR;        /* Wejscie kierunku zliczania      */

/** Outputs **/
pin [14..17] = [Q3..Q0];    /* Binarne wyjscia licznika */
pin 19 = CARRYO;            /* Wyjscie przeniesienia    */

/** Declarations and Intermediate Variable Definitions **/
field count = [Q3..0];      /* Deklaracja pola zliczania */
$define S0 'b'0000          /* Definicje stanow S0..S10  */
$define S1 'b'0001          /* - w kodzie binarnym       */
$define S2 'b'0010
$define S3 'b'0011
$define S4 'b'0100
$define S5 'b'0101
$define S6 'b'0110
$define S7 'b'0111
$define S8 'b'1000
$define S9 'b'1001

field mode = [RES,DIR];  /* Deklaracja pola sterowania      */
up = mode:0;             /* Deklaracja wartosci pola "mode" */
                         /* dla trybu zliczania w gore      */
down = mode:1;           /* Deklaracja wartosci pola "mode" */
                         /* dla trybu zliczania w dol       */
reset = mode:[2..3];     /* Deklaracja wartosci pola "mode" */
                         /* dla trybu synchronicznego kasow. */

/** Logic Equations **/

sequence count {

present S0      if up           next S1;
                if down         next S9;
                if reset        next S0;
                if down         out CARRYO;
present S1      if up           next S2;
                if down         next S0;
                if reset        next S0;
present S2      if up           next S3;
                if down         next S1;
                if reset        next S0;
present S3      if up           next S4;
                if down         next S2;
                if reset        next S0;
present S4      if up           next S5;
                if down         next S3;
                if reset        next S0;
present S5      if up           next S6;
                if down         next S4;
                if reset        next S0;
present S6      if up           next S7;
                if down         next S5;
                if reset        next S0;
present S7      if up           next S8;
                if down         next S6;
                if reset        next S0;
present S8      if up           next S9;
                if down         next S7;
                if reset        next S0;
present S9      if up           next S0;
                if down         next S8;
                if reset        next S0;
                if up           out CARRYO;
}
```

List. 4a. Plik wejściowy CUPL'a. Realizacja licznika dekadowego

```
*************************************************
                    Licznik
*************************************************

CUPL            4.0a Serial# MD-40A-8209
Device          g16v8ms  Library DLIB-h-26-11
Created         Thu Oct 28 13:13:50 1993
Name            Licznik
Partno          US3
Revision        1.9
Date            28/10/93
Designer        P.Z.
Company         BTC
Assembly        xxxxxx
Location        xxxxxx


================================================
              Expanded Product Terms
================================================

CARRYO =>
    DIR & !Q0 & !Q1 & !Q2 & !Q3 & !RES
  # !DIR & Q0 & !Q1 & !Q2 & Q3 & !RES

Q0.d  =>
    !Q0 & !Q1 & !Q2 & Q3 & !RES
  # !Q0 & !Q3 & !RES

Q1.d  =>
    DIR & !Q0 & !Q1 & !Q2 & Q3 & !RES
  # !DIR & Q0 & !Q1 & !Q3 & !RES
  # !DIR & !Q0 & Q1 & !Q3 & !RES
  # DIR & Q0 & Q1 & !Q3 & !RES
  # DIR & !Q0 & !Q1 & Q2 & !Q3 & !RES

Q2.d  =>
    DIR & !Q0 & !Q1 & !Q2 & Q3 & !RES
  # !DIR & Q0 & Q1 & !Q2 & !Q3 & !RES
  # !DIR & !Q1 & Q2 & !Q3 & !RES
  # DIR & Q0 & Q2 & !Q3 & !RES
  # !Q0 & Q1 & Q2 & !Q3 & !RES

Q3.d  =>
    DIR & Q0 & !Q1 & !Q2 & Q3 & !RES
  # DIR & !Q0 & !Q1 & !Q2 & !Q3 & !RES
  # !DIR & Q0 & Q1 & Q2 & !Q3 & !RES
  # !DIR & !Q0 & !Q1 & !Q2 & Q3 & !RES

count =>
    Q3 , Q2 , Q1 , Q0

down =>
    DIR & !RES

mode =>
    RES , DIR

reset =>
    RES

up =>
    !DIR & !RES

CARRYO.oe  =>
    1

========================================
              Chip Diagram
========================================

                 _____________
                |   Licznik   |
          CLK x-|1           20|-x Vcc
              x-|2           19|-x CARRYO
          RES x-|3           18|-x
              x-|4           17|-x Q0
          DIR x-|5           16|-x Q1
              x-|6           15|-x Q2
              x-|7           14|-x Q3
              x-|8           13|-x
              x-|9           12|-x
          GND x-|10          11|-x
                |_____________|
```

List. 4b. Plik dokumentacyjny (*.DOC) do list. 4a.

definiuje stany początkowe wewnętrznych węzłów, pól bitowych (ustalenie stanu początkowego wymaga użycia słowa kluczowego APPEND);

- blok równań logicznych - zawiera równania boolowskie, tablice prawdy i sekwencje stanów automatów.

Warto wspomnieć, iż stałe St, definiowane jako argumenty operacji przypisania, mogą być przedstawione jako liczby w dowolnym kodzie liczenia. Domyślnie przyjmowana wartość reprezentowana jest jako liczba szesnastkowa (nie wymaga żadnego prefiksu). Jeżeli liczba zapisywana jest w kodzie dziesiętnym to należy ją poprzedzić znacznikiem ", (np.: 'd'27), w przypadku liczby ósemkowej 'o' (np. 'o'256 lub też 'o'[100..230] przy deklaracji przedziału), dla liczby binarnej jest to znacznik 'b' (np. 'b'110110 lub 'b'[100..111]). W przypadku liczb dwójkowych możliwe jest zadeklarowanie wartości nieustalonej bitu, np. 'b'1XX0110X.

Proponujemy teraz analizę **listingu 2** - przykład realizacji funktorów logicznych (opis za pomocą równań boolowskich), **listingu 3** - dekoder wyświetlacza kodu heksadecymalnego (przykład zastosowania opisu za pomocą tabeli prawdy) i **listingu 4** - projekt licznika synchronicznego (przykład opisu za pomocą funkcji przejść), które zawierają przykładowe problemy wraz z implementacją do układów PLD. Aby nie powiększać zbytnio objętości artykułu, jedynie dla najprostszego (list.2) przypadku przedstawione są wszystkie pliki wyjściowe (oprócz symulacyjnych) „produkowane" przez kompilator. Dla pozostałych przypadków większą część (poza nagłówkiem i rozwinięciami funkcji przypisanych do poszczególnych wyjść) plików dokumentacyjnych usunięto.

W następnym numerze EP napiszemy o innym języku - PALASM.

**Piotr Zbysiński, AVT**