Układy programowalne, część 5

Czytelnicy, którzy cierpliwie przebrnęli przez wprowadzenie do języka CUPL, znajdą teraz nieco „praktycznej" satysfakcji. Przechodzimy bowiem (powoli) do prezentacji przykładowych projektów implementowanych w układzie ispGAL22V10, który jest „sercem" zestawu ewaluacyjnego AVT-559, opisanego w EP3/2004.

Każda próba systematyzacji nabywanej wiedzy budziła we mnie bunt: po co zaczynać prace od podstaw, skoro chciałbym od razu zająć się zagadnieniami poważnymi? Pewnie wśród Czytelników jest wiele osób podobnie podchodzących do tematu (tak przynajmniej wynika z listów, a przychodzi ich zadziwiająco – jak na PLD – dużo), ale teraz już wiem, że systematyczne pokonywanie etapów poznania jest (przeciętnie rzecz ujmując) lepszym wyjściem, niż porywanie się od razu na zbudowanie kontrolera sieci Ethernet (to oczywiście tylko przykład).

List. 1. Przykładowy opis ilustrujący strukturę pliku źródłowego **.pld*

```
Name   proba;
Partno 999;
Revision  01;
Date   21/05/2004;
Designer  PZb;
Company   EP;
Assembly  PCB01;
Location  U2;
Device G16V8;
Format j;

/***** Wejscia *****/
pin 1 = CLK;  /* Zegar */
pin 2 = CL;   /* Zerowanie */
pin 4 = CA;   /* Wejscie czujnika A */
pin 5 = CB;   /* Wejscie czujnika B */

/***** Wyjscia *****/
pin 12 = ERROR;   /* Wyjscie wskazujace blad */
pin 14 = C_PLUS;
/* Wyjscie zwiekszajace licznik */
pin 15 = C_MINUS;
/* Wyjscie zmniejszajace licznik */
pin [16..19] = [Q0..Q3];

/***** Deklaracje pomocnicze *****/
field NUMER_STANU = [Q2..0];
field WEJSCIA = [CB,CA,CL];

PAUZA = WEJSCIA:'b'000;
A = WEJSCIA:'b'010;
B = WEJSCIA:'b'100;
ERR = WEJSCIA:'b'110;
CLR = WEJSCIA:'b'XX1;

$define S0 'b'000
$define S1 'b'001
$define S2 'b'010
$define S3 'b'011
$define S4 'b'100
$define S5 'b'101
$define S6 'b'110

/***** Opis HDL *****/
sequence NUMER_STANU {
present S0  if A         next S1;
            if B         next S4;
            if PAUZA     next S0;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
present S1  if PAUZA     next S2;
            if A         next S1;
            if CLR       next S0;
            if ERR       next S0     out ERROR;

present S2 if B          next S3;
            if PAUZA     next S2;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
present S3 if PAUZA      next S0     out C_PLUS;
            if B         next S3;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
present S4 if PAUZA      next S5;
            if B         next S4;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
present S5 if A          next S6;
            if PAUZA     next S5;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
present S6 if PAUZA      next S0     out C_MINUS;
            if A         next S6;
            if CLR       next S0;
            if ERR       next S0     out ERROR;
}
```

Zaczniemy więc od przydatnych „banałów", stopniowo przechodząc do przykładów, które pokażą prawdziwe możliwości „małych" układów PLD.

Format pliku wejściowego

Standardowym rozszerzeniem nazwy pliku źródłowego dla kompilatorów CUPL-a jest **.pld*. Format pliku źródłowego i jego organizacja są takie same, niezależnie od tego, w jakim systemie będzie on kompilowany.

Plik źródłowy (przykład pokazano na **list. 1**) składa się z trzech części.

1. Nagłówek, w skład którego wchodzą następujące pola rozpoczynające się od słów kluczowych:

```
Name        proba;
Partno      999;
Revision    01;
Date        21/05/2004;
Designer    PZb;
Company     EP;
Assembly    PCB01;
Location    U2;
Device      G22V10;
Format      ij;
```

Każda linia musi być zakończona średnikiem. Znaczenie poszczególnych wpisów jest następujące:

`Name` – zawiera nazwę projektu, której maksymalna długość wynosi 32 znaki. Nazwa ta nie musi być taka sama jak nazwa pliku źródłowego (**.pld*), ale należy pamiętać, że pliki będące wynikiem kompilacji projektu będą nosiły nazwy takie same jak nazwa wpisana w to pole (będą się różniły tylko rozszerzeniami).

`Partno` – pole służące do wpisania firmowego oznaczenia projektowanego układu, co ma ułatwić identyfikację układu.

`Revision` – numer wersji projektu. Aktualizacja tego numeru w niektórych systemach CUPL odbywa się automatycznie po zmianie zawartości pliku źródłowego, w niektórych wersjach CUPL-a (m.in. w wersji atmelowskiej dla Windows) numer wersji nie jest aktualizowany automatycznie. Słowo kluczowe `revision` można zastąpić skrótem `rev`.

`Date` – w niektórych wersjach CUPL-a jest tu wstawiana data utworzenia pliku źródłowego, w niektórych wersjach jest automatycznie wprowadzana data ostatniej aktualizacji.

`Designer` – pole przeznaczone na wpisanie nazwiska projektanta.

`Company` – pole przeznaczone na wpisanie nazwy firmy, w której projekt jest realizowany.

`Assembly` – identyfikator płytki drukowanej, na której ma być montowany projektowany układ. Alternatywnie słowo kluczowe `assembly` można zastąpić skrótem `assy`.

`Location` – pole przeznaczone na współrzędne określające miejsce montażu projektowanego układu na płytce drukowanej. Alternatywnie słowo kluczowe `location` można zastąpić skrótem `loc`.

`Device` – w tym polu jest wpisywana mnemoniczna nazwa określająca docelowy układ PLD. W zależności od systemu, w którym zintegrowano kompilator CUPL-a, liczba dostępnych układów i odnoszące się do nich mnemoniki mogą być różne. W naszym przypadku (układ GAL22V10 w obudowie PLCC28, który zastosowano w zestawie AVT-559) będziemy stosować nazwę `g22v10lcc`. Jeżeli realizowany projekt nie będzie implementowany w konkretnym typie układu PLD, po słowie kluczowym `device` można wpisać `virtual`, co oznacza wirtualny układ PLD. W CUPL-u układ wirtualny ma architekturę PAL z opcjonalnymi rejestrami wejściowymi i nieograniczoną liczbą wejść bramek AND i OR ulokowanych w „matrycy programowanej".

`Format` – za pomocą dyrektyw wpisanych w tej linii projektant może określić, jakie pliki wynikowe są tworzone podczas kompilacji pliku **.pld*. Dostępne są następujące opcje:
h – powoduje utworzenie pliku

w formacie ASCII-hex

i – powoduje utworzenie pliku w formacie Signetics HL,

j – powoduje utworzenie pliku w formacie JEDEC (najczęściej stosowane w przypadku układów GAL).

Wpisanie dwóch lub trzech liter odpowiadających opisanym opcjom po słowie kluczowym `format` powoduje wygenerowanie przez kompilator odpowiedniej liczby plików wynikowych.

Jak widać, w nagłówku znajduje się wiele zbędnych informacji, których wprowadzanie można pominąć. W takim przypadku kompilator każdorazowo informuje użytkownika o braku oczekiwanej linii w pliku, ale poddaje go normalnej kompilacji. Aby uniknąć niepotrzebnego alarmowania, można w zbędne pola wpisać dowolne słowo (np. „brak") lub literę.

2. Deklaracje wejść, wyjść i węzłów, które służą do określania zewnętrznego interfejsu projektowanego układu oraz „ręcznego" przypisania sygnałów wewnętrznych do określonych (charakterystycznych) miejsc wewnątrz układu scalonego (węzłów wewnętrznych, często zwanych węzłami „zagrzebanymi").

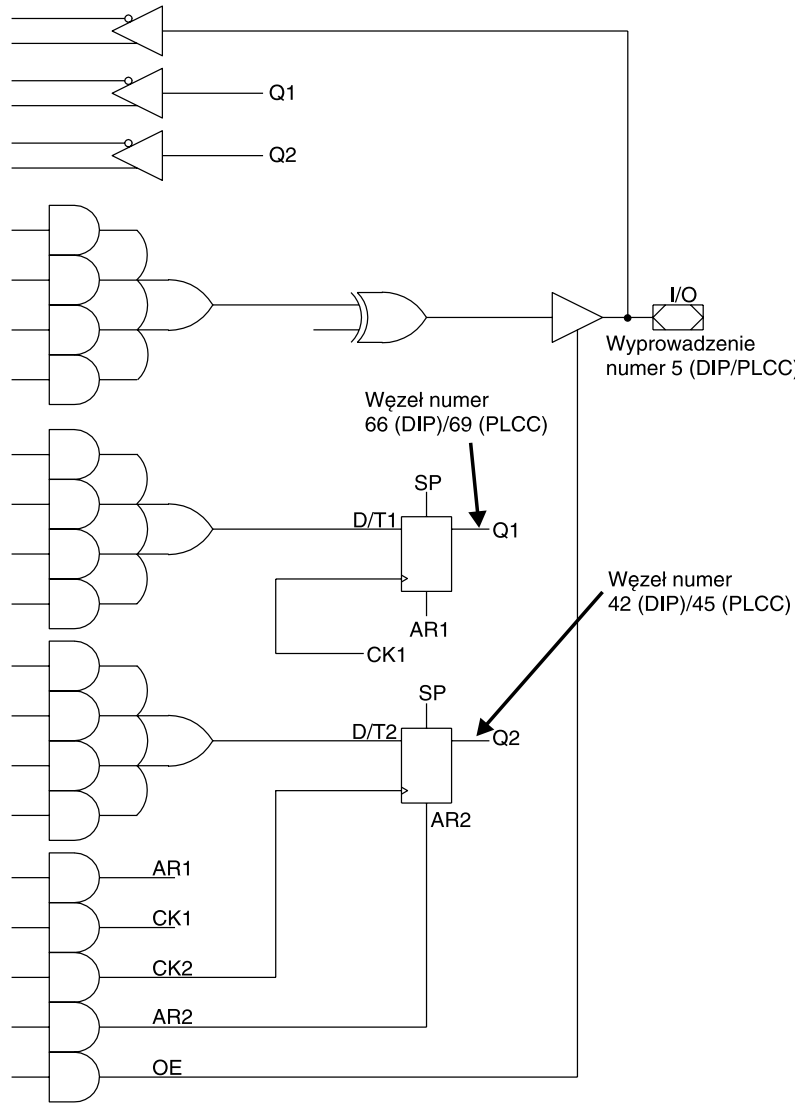
Przykładowe deklaracje przedstawiono poniżej:

```
/***** Wejscia *****/
PIN 1 = CLK;/* wejscie zegarowe */
PIN 2 = RES;/* wejscie zerowania */
PIN 3 = RxT;/* wejscie danych */

/***** Wyjscia *****/
PIN 19 = wy_clk_8; /* wyjscie preskalera 1:8 */
PIN 18 = wy_clk_4; /* wyjscie preskalera 1:4 */

/*****
Przypisanie sygnałów do węzłów wewnętrznych
*****/
PINNODE 213 = Q_X;
/* przypisanie sygnału Q_X do węzła 213 */
PINNODE 199 = RES_XTAL;
/* przypisanie sygnału RES_XTAL do węzła 199 */
```

Jak widać, w deklaracjach projektant nie określa w jawny sposób kierunku wyprowadzeń (wejście/wyjście), ustala to kompilator na podstawie opisu HDL. Deklarowanie węzłów zagrzebanych nie jest niezbędne i w praktyce (zwłaszcza w przypadku realizacji projektów na układy PLD o niewielkich zasobach) jest rzadko stosowane. W przypadku korzystania z bezpośrednich odwołań do węzłów zagrzebanych należy pamiętać, że liczby określające ich numery zależą od typu obudowy. Przykładowo, w przypadku układu ATV2500 firmy Atmel wyjście przerzutnika Q1 z komórki przypisanej

Węzły zagrzebane...

...są to miejsca w strukturze logicznej układów PLD określane w języku CUPL liczbą z zakresu 0...512. Informacje o lokalizacji takich węzłów były publikowane w notach katalogowych układów

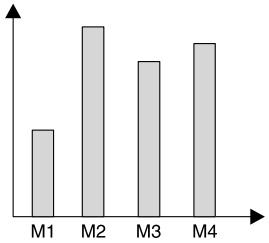


Rys. 25

Tab. 12. Dostępne sposoby minimalizacji funkcji logicznych w CUPL-u

n	Algorytm minimalizacji	Opis
0	Bez minimalizacji	Optymalizacja wyłączona, opcja zalecana podczas implementacji projektu w pamięci PROM/EPROM/EEPROM.
1	Quick	Metoda zrównoważona, zapewniająca krótki czas optymalizacji, wymagająca niewielkich zasobów pamięci, charakteryzująca się relatywnie słabą skutecznością optymalizacji.
2	Quine McCluskey	Najbardziej skuteczna minimalizacja, wymagająca dużych zasobów pamięci i - w przypadku dużej liczby zmiennych - wymagająca długotrwałych obliczeń.
3	Presto	Średnia skuteczność minimalizacji, nie wymaga pamięci o dużej pojemności. Szczególnie dobre wyniki daje podczas minimalizacji projektów implementowanych w układach IFL.
4	Expresso	Metoda o większej skuteczności minimalizacji niż Presto. Szczególnie dobre wyniki daje podczas minimalizacji projektów implementowanych w układach IFL.

Uwaga! Układy IFL (*Integrated Fuse Logic*) firmy Signetics nie są obecnie produkowane.



Rys. 26

wyprowadzeniu 5 (obudowa DIP) ma numer 66, a ten sam węzeł w przypadku układu w obudowie PLCC ma numer 69 (**rys. 25**).

W tej części opisu mogą się znaleźć (ale nie muszą, ma to znaczenie wyłącznie porządkowe) także deklaracje pomocnicze, jak na przykład:

```
field COUNT = [WY1..0];
field WYJSCIA = [wy1,acc_xa,re_out];
```

W przypadku odwoływania się w takich deklaracjach do zmiennych indeksowanych należy przestrzegać następujących zasad:
– nie należy w jednym polu odwoływać się do zmiennych indeksowanych i nieindeksowanych (jak np. `field [wy1..0,a,b,ext]`),
– nie należy w jednym polu używać dwóch lub więcej zmiennych o takim samym indeksie (jak np. `field [x1,y1,c2]`).

W obszarze deklaracji można (choć może to także nastąpić w dowolnym innym miejscu opisu HDL) określić sposób minimalizacji funkcji logicznej generującej określoną zmienną. Do tego celu służy słowo kluczowe MIN. Format deklaracji jest następujący:

```
MIN zmienna.rozszerzenie = n;
```

gdzie:

`zmienna` – jest to nazwa funkcji poddawanej minimalizacji,

`rozszerzenie` – opcjonalne rozszerzenie nazwy, stosowane na przykład w przypadku, gdy minimalizowana zmienna jest przypisana do wejścia D przerzutnika,

`n` – liczba z zakresu 0...4, która określa sposób (algorytm) minimalizacji (zgodnie z **tab. 12**).

Szacowane przez producenta wartości (uśrednione dla różnych projektów) współczynnika minimalizacji pokazano na **rys. 26** (podano za dokumentacją firmy Logical Devices).

3. Opis HDL, który może zostać przygotowany za pomocą:
– równań logicznych (Boole'a),
– tablic prawdy,
– opisu automatu (tekstowy odpowiednik grafu przejść).

Przykładowy fragment opisu projektowanego układu pokazano poniżej:

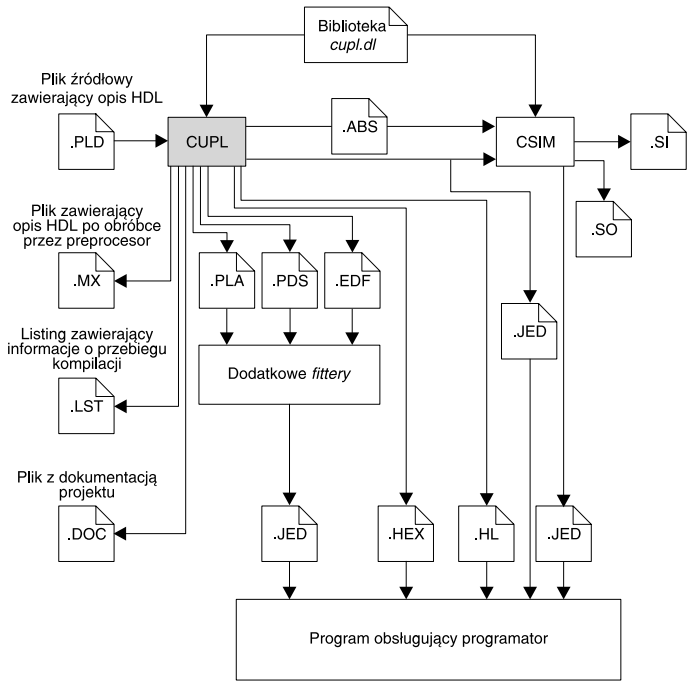
```
/***** Opis HDL *****/
OUTP = !ADR0 & !ADR1 & INP0
   # ADR0 & !ADR1 & INP1
   # !ADR0 & ADR1 & INP2
   # ADR0 & ADR1 & INP3;

CLK_OUT = !SELECT & SYNC
   # SELECT & ASYNC;

field COUNT = [WY1..0];
$define S0 'b'00
$define S1 'b'01
$define S2 'b'10
$define S3 'b'11

sequence COUNT {
present S0 next S1;
present S1 next S2;
present S2 next S3;
present S3 next S0;
}
```

Warto wiedzieć

Z wykresu pokazanego na rys. 26 wynika, że najskuteczniejszy jest algorytm minimalizacji Quine McCluskey. Warto jednak wziąć pod uwagę, że czas obliczeń rośnie wykładniczo (zgodnie ze wzorem $3^n/n$) wraz ze wzrostem liczby zmiennych wejściowych (n).

Pliki tworzone podczas kompilacji

Kompilator CUPL składa się z kilku programów (CUPLA – parser, CUPLB – fitter, CUPLC – generator plików wyjściowych, CUPLX – preprocesor, CUPLM – minimalizator), które wywoływane kolejno realizują



Rys. 27

etapy kompilacji. Ponieważ środowiska IDE, w które wbudowano CUPL-a (zajmiemy się ich przybliżeniem w kolejnych odcinkach cyklu), samodzielnie uruchamiają te programy i zarządzają obiegiem plików pomiędzy nimi, my skupimy się na przedstawieniu sposobu wymiany danych wyłącznie pomiędzy kompilatorem, symulatorem i dodatkowymi programami, jak na przykład edytory schematów, programy obsługujące programatory itp.

Na **rys. 27** pokazano obieg plików pomiędzy kompilatorem, symulatorem i opcjonalnymi programami dodatkowymi. Opis funkcji poszczególnych plików znajduje się w **tab. 13**.

Układy kombinacyjne

Po sporej dawce rozważań teoretycznych przechodzimy do pierwszych przykładów. W tej części kursu przedstawimy sposoby projektowania układów kombinacyjnych.

Układy kombinacyjne są to takie układy cyfrowe, których stany wyjściowe w danej chwili zależą jedynie od aktualnego stanów wejść (są one pozbawione pamięci historii).

Podstawowymi, powszechnie stosowanymi, elementami kombinacyjnymi są bramki logiczne i to od przedstawienia ich opisu zaczniemy opisywanie sprzętu w CUPL-u.

Na **list. 2** znajduje się przykładowy opis bramek logicznych, który wykonano za pomocą równań boole'owskich (z wykorzystaniem operatorów logicznych, które przedstawiono w EP5/2004). Taki sam efekt (czyli implementację w układzie PLD bramek logicznych) można uzyskać w nieco inny sposób, a mianowicie z wykorzy-

Tab. 13. Rozszerzenia nazw plików tworzonych przez kompilator CUPL (nazwy plików są takie, jak zadeklarowano w polu name pliku **.pld*)

Rozszerzenie	Tworzony przez	Funkcja pliku
PLD	Projektanta	Zawiera opis HDL projektowanego układu PLD.
Pliki dokumentacyjne		
DOC	Kompilator	Plik dokumentujący sposób zaimplementowania projektu w układzie docelowym, łącznie z mapą przepaleń i rozmieszczeniem sygnałów dołączonych do wyprowadzeń układu PLD.
ABS	Kompilator	Plik binarny zawierający informacje niezbędne dla poprawnej pracy symulatora.
MX	Kompilator	Plik zawierający „rozwinięty" opis projektu, czyli zawierający jawne opisy makrofunkcji, opisy generowane przez preprocesor, a także opisy dołączane do pliku źródłowego (pobierane z zewnętrznych plików bibliotecznych).
LST	Kompilator	Plik zawierający listing programu z ponumerowanymi liniami. Błędy wykryte podczas kompilacji są umieszczana na końcu pliku. Zawierają one odwołania do linii, w której wykryto błąd.
Pliki przejściowe		
PLA	Kompilator	Plik zawiera informacje umożliwiające implementację projektów w układach PLA.
PDS	Kompilator	Plik zawierający opis projektu w języku PALASM.
EDF	Kompilator	Plik w formacie EDIF (presyntezowany), który można wykorzystać do implementacji projektu w dowolnym układzie PLD.
Pliki zawierające informacje niezbędne do programowania układów		
JED	Kompilator	Plik zawierający informacje umożliwiające zaprogramowanie układu. Stosowany dla większości układów PLD. Format pliku został ustandaryzowany przez komitet JEDEC (dokument JESD-3) i zaaprobowany przez stowarzyszenie EIA.
HEX	Kompilator	Plik zawierający informacje umożliwiające zaprogramowanie układu. Stosowany do programowania pamięci.
HL	Kompilator	Plik zawierający informacje umożliwiające zaprogramowanie układów IFL firmy Signetics.
Pliki symulacyjne		
SI	Projektanta	Plik wejściowy dla symulatora funkcjonalnego. Zawiera wektory wejściowe (pobudzenia) i - opcjonalnie - wyjściowe (odpowiedzi).
SO	Kompilator	Plik zawierający wyniki symulacji prowadzonej przez CUPL-a. Zawiera także informacje o błędach wykrytych podczas symulacji.

List. 2. Opis bramek logicznych za pomocą równań logicznych

```
Name            bramki;
Partno          U1;
Revision        01;
Date            20/05/04;
Designer        PZb;
Company         EP;
Location        brak;
Assembly        brak;
Device          g22v10lcc;

/*stany na wejsciach a (b_0 na plytce) i b (b_1)*/
/*ustala sie za pomoca nastawnika SW1, pozycje: 0...3*/

/***** Wejscia *****/
Pin 11 =  a;
Pin 10 =  b;

/***** Wyjscia *****/
Pin 17 = inva; /* D1 */
Pin 18 = and; /* D2 */
Pin 19 = nand; /* D3 */
Pin 20 = or; /* D4 */
Pin 21 = nor; /* D5 */
Pin 23 = xor; /* D6 */
Pin 24 = xnor; /* D7 */

/***** Opis HDL *****/
inva = !a;       /* inwerter sygnalu z wejscia A*/
and  = a & b;    /* bramka AND */
nand = !(a & b); /* bramka NAND */
or   = a # b;    /* bramka OR */
nor  = !(a # b); /* bramka NOR */
xor  = a $ b;    /* bramka ExOR */
xnor = !(a $ b); /* bramka ExNOR */
```

staniem tablic prawdy (**list. 3**).

Ten drugi sposób jest nieco bardziej rozwlekły, ale miał za zadanie zilustrować możliwość uzyskania takiego samego efektu za pomocą różnych sposobów opisu. Przedstawiony sposób tablicowego opisu bramek nie jest jedynym możliwym. Przykładowo, zamiast korzystać z zadeklarowanego w pliku źródłowym (list. 3) pola `wejscia` tablicę można zbudować korzystając z jawnie podanych wejść a i b, przykładowo:

```
/* bramka AND */
table [a,b] => and {
   'b'00  => 0;
   'b'01  => 0;
   'b'10  => 0;
   'b'11  => 1;
}
```

Także wartości bitów wejściowych, można zapisać inaczej niż to pokazano w przedstawionych przykładach. Przykładowo, stany wejściowe można podawać jawnie (w tym przypadku zapisy <!'b'10 i [1,0] są równoważne). Taki zapis pokazano poniżej:

```
/* bramka AND */
table [a,b] => and {
   [0,0]  => 0;
   [0,1]  => 0;
   [1,0]  => 0;
   [1,1]  => 1;
}
```

Jeżeli z jakichś przyczyn wygodniejsze jest posługiwanie się sposobem zapisu liczb innym niż binarny, tę samą tablicę można zapisać na przykład w taki sposób:

```
/* bramka AND */
table [a,b] => and {
   'd'0 => 'b'0;     /* zapis dziesietny/
                        binarny */
   'o'1   => 'b'0;   /* zapis osemkowy/
                        binarny */
   'h'2 => 'o'0;     /* zapis szesnastkowy/
                        osemkowy */
   'b'11  => 'h'1;   /* zapis binarny/
                        szesnastkowy */
}
```

Piotr Zbysiński, EP
piotr.zbysinski@ep.com.pl

List. 3. Opis bramek logicznych za pomocą tablic prawdy

```
Name            bram_tab;
Partno          U1;
Revision        01;
Date            20/05/04;
Designer        PZb;
Company         EP;
Location        brak;
Assembly        brak;
Device          g22v10lcc;

/* stany na wejsciach a (b_0 na plytce)
      i b (b_1) */
/* ustala sie za pomoca nastawnika SW1, pozy-
      cje: 0...3 */

/***** Wejscia *****/
Pin 11 =  a;
Pin 10 =  b;

/***** Wyjscia *****/
Pin 17 = inva; /* D1 */
Pin 18 = and; /* D2 */
Pin 19 = nand; /* D3 */
Pin 20 = or; /* D4 */
Pin 21 = nor; /* D5 */
Pin 23 = xor; /* D6 */
Pin 24 = xnor; /* D7 */

/***** Deklaracje pomocnicze *****/
field wejscia = [a,b];

/***** Opis HDL *****/
table a => inva {    /* inwerter sygnalu
                        z wejscia A*/
   0   => 1;
   1   => 0;
}

table wejscia => and {  /* bramka AND */
    'b'00  => 0;
    'b'01  => 0;
    'b'10  => 0;
    'b'11  => 1;
}

table wejscia => nand   { /* bramka NAND */
    'b'00  => 1;
    'b'01  => 1;
    'b'10  => 1;
    'b'11  => 0;
}

table wejscia => or {   /* bramka OR */
    'b'00  => 0;
    'b'01  => 1;
    'b'10  => 1;
    'b'11  => 1;
}

table wejscia => nor {  /* bramka NOR */
    'b'00  => 1;
    'b'01  => 0;
    'b'10  => 0;
    'b'11  => 0;
}

table wejscia => xor {  /* bramka ExOR */
    'b'00  => 0;
    'b'01  => 1;
    'b'10  => 1;
    'b'11  => 0;
}

table wejscia => xnor { /* bramka ExNOR */
    'b'00  => 1;
    'b'01  => 0;
    'b'10  => 0;
    'b'11  => 1;
}
```