# Układy programowalne, część 4

***W tej części cyklu dokończymy omówienie poleceń preprocesora kompilatora CUPL, omówimy także rozszerzenia nazw sygnałów przydatne podczas realizacji projektów na układach GAL16V8, 18V8, 20V8 i 22V10.***

**Polecenie $ENDIF**

Jak wspomniano w poprzednim odcinku cyklu, polecenie to służy do zaznaczania końca opisu HDL kompilowanego warunkowo, którego początek wskazuje jedno z poleceń: $IFDEF lub $IFNDEF.

Format polecenia jest następujący:

```
$ENDIF
```

Polecenia warunkowej kompilacji mogą być zagnieżdżane, przy czym należy pamiętać o tym, żeby każdy poziom zagnieżdżenia zaznaczony poleceniem $IFDEF lub $IFNDEF został „zamknięty” poleceniem $ENDIF.

Przykład:

```
$IFDEF argument_1
pin 1 = we1;
pin 2 = we2;
$IFDEF argument_2
pin 3 = ramka_a;
pin 4 = rom_sel;
$ENDIF
pin 5 = in_rcs;
pin 6 = rs_dek;
$ENDIF
```

**Polecenie $ELSE**

Jest to polecenie, za pomocą którego można tworzyć lokalne rozgałęzienia kompilacji warunkowej, której obszar zaznaczono jednym z poleceń: $IFDEF lub $IFNDEF.

Działanie polecenia jest następujące: jeżeli warunek testowany przez polecenia $IFDEF i $IFNDEF jest spełniony (czyli opis znajdujący się za nimi jest kompilowany), to opis po poleceniu $ELSE jest ignorowany. Jeżeli natomiast testowany przez polecenia $IFDEF i $IFNDEF warunek nie jest spełniony, opis HDL znajdujący się po nich jest ignorowany, a kompilacji jest poddawany opis po poleceniu $ELSE.

Format polecenia jest następujący:

```
$ELSE
```

Przykład:

```
$DEFINE Wersja 1
$IFDEF Wersja
pin 1 = mem_req;
pin 2 = io_req;
$ELSE
pin 1 = io_req;
pin 2 = mem_req;
$ENDIF
```

W przedstawionym przykładzie można zdecydować o sposobie przypisania sygnałów do wyprowadzeń układu zmieniając nazwę zdefiniowanej stałej lub przez usunięcie linii zawierającej jej definicję ($DEFINE Wersja 1).

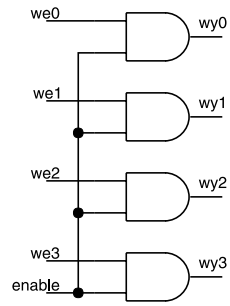
**Polecenie $REPEAT**

Za pomocą tego polecenia można automatyzować tworzenie opisów HDL, które składają się z wielu takich samych bloków funkcjonalnych, różniących się jedynie indeksem. Zakres wartości indeksu musi się mieścić w zakresie 0...1023.

Format tego polecenia jest następujący:

```
$REPEAT index=[liczba_0, liczba_1,...liczba_n]
    powielany opis
    z elementami
    indeksowanymi
$REPEND
```

Preprocesor przed kompilacją projektu „rozwija” opis, tworząc odpowiednią (wynikającą z zakresu indeksu) liczbę bloków funkcjonalnych. Indeksowanie nie musi przebiegać kolejno od liczby 0 do liczby *n*, ale w takim przypadku konieczne jest jawne podanie kolejnych wartości indeksów




Rys. 19

([liczba_0, liczba_1,...liczba_n]). Jeżeli indeksowanie ma przebiegać kolejno w podanym przedziale, użytkownik może podać tylko najmniejszą i największą wartość z przedziału indeksowania ([liczba_1..liczba_n]).

Przykład:

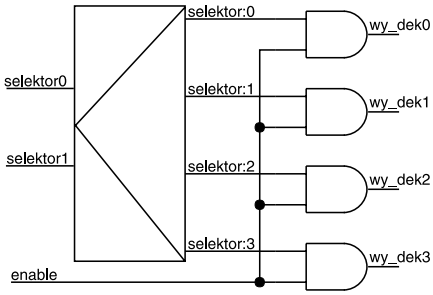
```
$REPEAT i = [0..3]
  wy{i} = we{i} & enable;
$REPEND
```

Wynikiem działania preprocesora jest następujący opis HDL, odpowiadający układowi pokazanemu na **rys. 19**:

```
wy0 = we0 & enable;
wy1 = we1 & enable;
wy2 = we2 & enable;
wy3 = we3 & enable;
```

Oczywiście, za pomocą polecenia $REPEAT można „rozwijać” znacznie bardziej skomplikowane bloki funkcjonalne, niż pokazany w przykładzie. Podczas „rozwijania” opisu można wykorzystywać także operatory arytmetyczne, czego przykład pokazano poniżej:

```
FIELD licznik = [wy2..wy0];
```

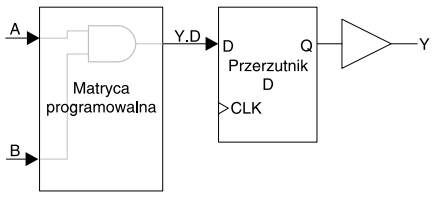



Rys. 20

```
SEQUENCE licznik {
$REPEAT i = [0..7]
  PRESENT {i}
  IF zliczaj & reset NEXT {(i+1)%(8)};
  IF !reset NEXT {0};
  DEFAULT NEXT {i};
$REPEND
}
```

Przedstawiony przykład po działaniu preprocesora wygląda następująco:

```
FIELD licznik = [wy2..wy0];
SEQUENCE licznik {
PRESENT 0
  IF zliczaj & reset NEXT 1;
  IF !reset NEXT 0;
  DEFAULT NEXT 0;
PRESENT 1
  IF zliczaj & reset NEXT 2;
  IF !reset NEXT 0;
  DEFAULT NEXT 1;
PRESENT 2
  IF zliczaj & reset NEXT 3;
  IF !reset NEXT 0;
  DEFAULT NEXT 2;
PRESENT 3
  IF zliczaj & reset NEXT 4;
  IF !reset NEXT 0;
  DEFAULT NEXT 3;
PRESENT 4
  IF zliczaj & reset NEXT 5;
  IF !reset NEXT 0;
  DEFAULT NEXT 4;
PRESENT 5
  IF zliczaj & reset NEXT 6;
  IF !reset NEXT 0;
  DEFAULT NEXT 5;
PRESENT 6
  IF zliczaj & reset NEXT 7;
  IF !reset NEXT 0;
  DEFAULT NEXT 6;
PRESENT 7
  IF zliczaj & reset NEXT 0;
  IF !reset NEXT 0;
  DEFAULT NEXT 7;
}
```




Rys. 21

## Polecenie $REPEND

Uważni Czytelnicy zauważyli z pewnością, że polecenie $REPEND jest używane do zaznaczania końca fragmentu opisu HDL, który jest „rozwijany" zgodnie z ustalonym przez użytkownika indeksowaniem. Każde polecenie $REPEAT musi zostać odwołane przez $REPEND, nie jest możliwe zagnieżdżanie poleceń $REPEAT.

Format tego polecenia jest następujący:

```
$REPEND
```

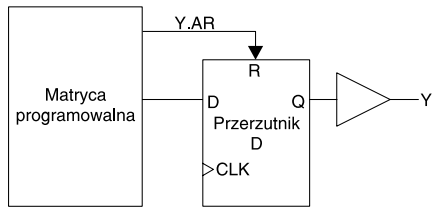
## Polecenie $MACRO

Za pomocą tego polecenia użytkownik może tworzyć własne makrofunkcje wywoływane w opisie HDL nadaną im nazwą i odpowiednimi parametrami. Jeżeli w projekcie makrofunkcja nie jest wykorzystywana, to jej opis HDL nie jest kompilowany.

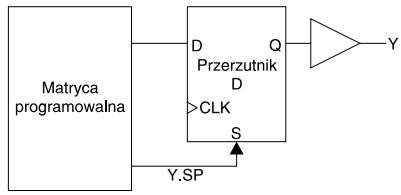
W opisie makrofunkcji można używać operatorów arytmetycznych, przy czym – podobnie jak ma to miejsce w przypadku polecenia $REPEAT – wszystkie działania muszą być ujęte w nawiasy półokrągłe.

Format polecenia:

```
$MACRO nazwa argument_1 argument_
2...argument_n
      opis HDL makofunkcji
$MEND
```




Rys. 22

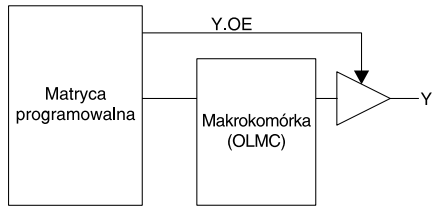



Rys. 23

Za pomocą makrofunkcji można na przykład tworzyć biblioteki gotowych bloków logicznych, na przykład będących odpowiednikami standardowych układów TTL.

Przykład (odpowiednik układu TTL 7474 – podwójny przerzutnik D):

```
$MACRO TTL7474 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
P11 P12 P13 P14
P5.d = P2;
P5.ck = !P3;
P5.ar = !P1;
P5.ap = !P4;
P6 = !P2;
P9.d = P12;
```




Rys. 24

**W przypadku, gdy nie wykorzystujemy wejść lub wyjść makrofunkcji, w chwili jej wywołania w miejsca parametrów odpowiadającym liniom niewykorzystanym należy wstawić słowo kluczowe NC.**

```
P9.ck = !P11;
P9.ar = !P13;
P9.ap = !P10;
P8 = !P12;
$MEND
```

Przykład (odpowiednik układu TTL 7402 – czterech dwuwejściowych bramek NOR):

```
$MACRO TTL7402 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
P11 P12 P13 P14
P3 = !(P1 # P2);
P6 = !(P4 # P5);
P8 = !(P9 # P10);
P11 = !(P12 # P13);
$MEND
```

Tworzone za pomocą makrofunkcji bloki funkcjonalne mogą być parametryzowane, co oznacza, że odpowiednio przygotowany opis licznika lub dekodera może być wykorzystywany w opisie projektu dla dowolnej liczby bitów (czyli np. jako licznik 4- i 12-bitowy lub jako dekoder 2->4 i 3->8 linii).

Przykład:

```
$MACRO dekoder l_bitow sel wy inh;
  FIELD dek = [sel{l_bitow-1}..0];
  $REPEAT i = [{(2**l_bitow)-1}..0]
     wy{i} = dek:'h'{i} & inh;
  $REPEND
$MEND
```

Wywołanie tej makrofunkcji (w wyniku rozwinięcia której powstaje dekoder z wejściem zezwalającym) wygląda na przykład następująco:

```
dekoder (2, selektor, wy_dek, inh_ext);
```

Po takim wywołaniu preprocesor generuje następujący opis HDL (odpowiadający mu, uproszczony schemat logiczny pokazano na **rys. 20**):

```
FIELD dek = [selektor1..0];
  wy_dek3 = dek:'h'3 & inh_ext;
  wy_dek2 = dek:'h'2 & inh_ext;
  wy_dek1 = dek:'h'1 & inh_ext;
  wy_dek0 = dek:'h'0 & inh_ext;
```

Dobrym zwyczajem jest przechowywanie makrodefinicji w zewnętrznym pliku. W CUPL-u przyjęto, że rozszerzeniem nazw plików zawierających makrofunkcje jest litera „m" (*.m). Korzystanie z tych plików umożliwia polecenie $INCLUDE, które opisano w poprzednim numerze EP.

### Polecenie $MEND

Jest to polecenie, za pomocą którego jest zaznaczany koniec każdej makrofunkcji (czyli polecenia $MACRO i $MEND muszą tworzyć pary i nie mogą być zagnieżdżane).

Format:

```
$MEND
```

### Rozszerzenia nazw zmiennych

Rozszerzenia nazw zmiennych są wykorzystywane w celu określenia specjalnej funkcji sygnału, źródła jego pochodzenia lub celu. Dzięki rozszerzeniom użytkownik może bezpośrednio operować na sygnałach niedostępnych na zewnątrz układu scalonego.

**Tab. 11. Rozszerzenia nazw zmiennych przydatne podczas korzystania z układów GAL22V10**

| Nazwa | Położenie względem znaku równości w równaniach logicznych | Opis |
|---|---|---|
| .AR | L | Asynchronous Reset – asynchroniczne zerowanie przerzutnika |
| .SP | L | Synchronous Preset – synchroniczne ustawianie przerzutnika |
| .OE | L | Output Enable – sygnał sterujący pracą bufora trójstanowego |
| .D | L | Wejście danych przerzutnika D |

Przykład:

```
pin 1 = A;
pin 2 = B;
pin 19 = Y;
Y.D = A & B;
```

Wynikiem przedstawionego zapisu jest przypisanie do wejścia D przerzutnika iloczynu logicznego sygnałów A i B, jak pokazano to na **rys. 21**.

CUPL obsługuje 42 rodzaje rozszerzeń, z których dla początkujących (przy założeniu, że dalsza część kursu będzie poświęcona głównie projektom realizowanym na układach GAL22V10) najistotniejsze są te, które przedstawiono w **tab. 11**. Na **rys. 22...24** pokazano graficzną interpretację znaczenia przedstawionych rozszerzeń.

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