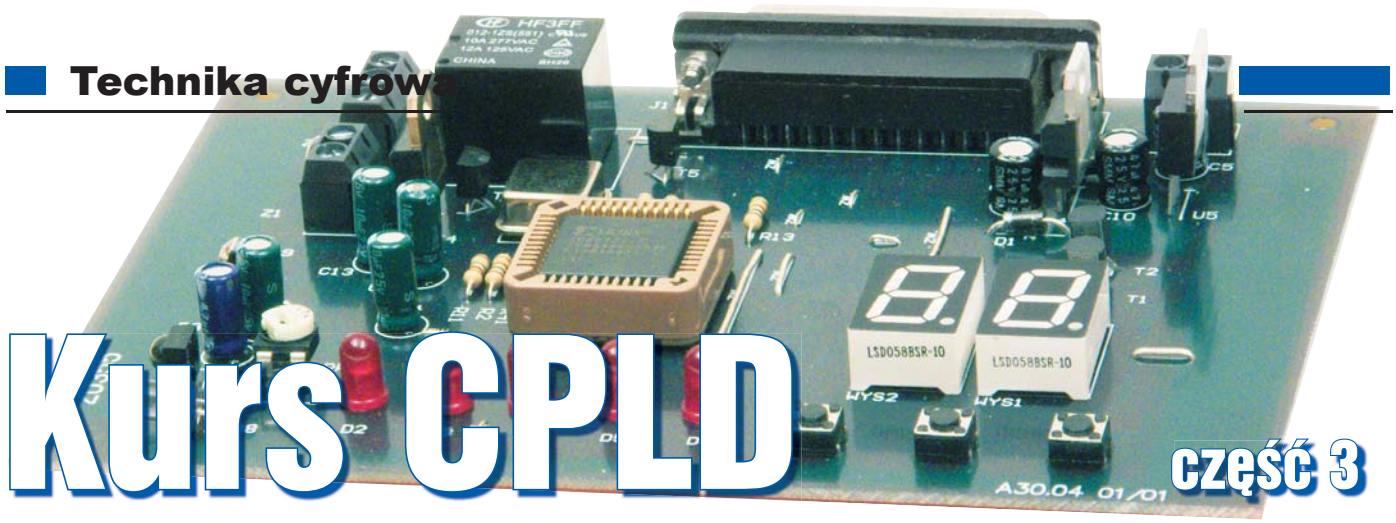


Kurs CPLD

część 3

Kilka słów wyjaśnienia

Tematem dzisiejszej lekcji będą układy kombinacyjne, do których zastosować można następującą definicję (przytoczona z książki *Układy cyfrowe* W. Głocki): *w układzie kombinacyjnym każda kombinacja sygnałów wejściowych określa jednoznacznie kombinację sygnałów wyjściowych. Kombinacja sygnałów wejściowych jest nazywana stanem wejść układu (lub słowem wejściowym), natomiast kombinacja sygnałów wyjściowych – stanem wyjść układu (słowem wyjściowym).*

Definicja ta nie jest trudna w interpretacji, ale mając na uwadze, że kurs czytają również młodzi Konstruktorzy, pozwolę sobie na parę zdań komentarza.

Słowem wejściowym jest w tym przypadku kombinacja zer i jedynek podanych na porty wejściowe układu. W przypadku bramki AND jest to para dwóch stanów logicznych podanych na jej wejścia. Mówiąc o słowie wyjściowym, mamy na myśli wszystkie stany logiczne występujące na wyjściu układu, np. stan wyjścia wspomnianej bramki AND.

Trudności może sprawić także określenie, co NIE jest układem kombinacyjnym. Układ kombinacyjny zawsze daje takie samo słowo wyjściowe dla danego słowa wejściowego – dobrym przykładem są tu wszelkiego rodzaju bramki i dekodery kodów (np. kody 1 z n). Układem kombinacyjnym NIE JEST np. komórka pamięci. Ma ona co prawda wejścia i wyjścia, ale dla zadanego adresu stan wyjść jest nieokreślony – wszystko zależy od tego, co było zapisane uprzednio w tej komórce. Tak samo przerzutnik D nie spełnia tej zależności – jego stan wyjściowy jest niezależny od wejścia, a zależy jedynie od tego, co zostało zapamiętane podczas wystąpienia aktywnego zbocza.

Ogólnie rzecz ujmując, układy kombinacyjne nie posiadają pamięci – nie są zdolne do gromadzenia informacji. Jednym z obszarów zastosowania układów kombinacyjnych są wszelkiego rodzaju przetworniki kodów. My zainteresujemy się obecnie takim przetwornikiem, w literaturze spotykanym pod określeniem *dekodera kodu BCD na kod wyświetlacza 7-segmentowego*. Czytelnicy na pewno zauważyli, że w dwóch poprzednich odcinkach unikaliśmy stosowania wyświetlacza 7-segmentowego, przynajmniej do wyświetlania liczb. Wynikało to z braku wiedzy o tym, jak zamienić liczbę w kodzie binarnym na odpowiednie sygnały niezbędne do wysterowania takiego wyświetlacza. W bibliotece nie ma gotowego układu, który zamieniłby *słowo wejściowe* podane w naturalnym systemie binarnym na *słowo wyjściowe* nadające się do wysterowania segmentów wyświetlacza. Do tego potrzebny jest właśnie układ kombinacyjny. Warto zauważyć, że taki dekoder *musi* dla danych sygnałów wejściowych dawać na wyjściu zawsze to samo. Niedopuszczalna jest sytuacja, w której podajemy na wejście wartość jeden i otrzymujemy czasami jeden, czasami dwa i niekiedy zero.

Dekoder 7-segmentowy

Przygotowanie stosownego układu kombinacyjnego wymaga przede wszystkim określenia, jaki rezultat chcemy otrzymać. Niewątpliwie chcemy mieć „coś", co wyświetli liczbę na wyświetlaczu 7-segmentowym. Optymalny wydaje się być blok funkcyjny (element w postaci symbolu), który ma cztery wejścia umożliwiające zadanie interesującej nas liczby w sposób binarny, i siedem wyjść, które będą podłączone do poszczególnych segmentów wyświetlacza. Skupimy się na zaprojektowaniu bloku, który będzie można wstawić do schematu jak każdy inny element, np. licznik. Nie jest to oczywiście konieczne, ale wiążą się z tym dwa udogodnienia:

– ukryjemy „wnętrzności" układu, co zdecydowanie podniesie czytelność schematu,

– stworzymy element, który będzie można wielokrotnie wykorzystywać również w innych projektach, bez potrzeby ponownego jego projektowania. Korzyść w postaci oszczędności czasu jest oczywista.

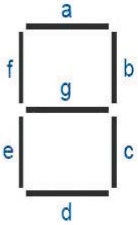
Każdy segment wyświetlacza 7-segmentowego można rozpatrywać jako osobną funkcję, której wartość (zero lub jeden, odpowiednio – segment włączony lub wyłączony) zależy od czterech parametrów – słowa wejściowego. Segment LED jest włączany zerem, gdyż tak została zaprojektowana płytka testowa CPLD.

Funkcji logicznych będzie w tym przypadku siedem, po jednej dla każdego segmentu. Warto wspomnieć, że są one wzajemnie niezależne, bo łączy je jedynie fakt, że pobierają słowo wejściowe z tego samego wejścia. Projektowanie układu kombinacyjnego dobrze jest rozpocząć od narysowania tabelki, w której wyszczególnione będą wszystkie stany wejściowe i wyjściowe. Wyświetlacz 7-segmentowy umożliwia prezentację wszystkich 10 cyfr, a do ich zapisania potrzebne są cztery bity. W związku z tym zapisujemy w tabelce wszystkich 16 (2^4) kombinacji jakie mogą wystąpić na wejściu. Obok, w kolumnie określającej stan wyjściowy, zapisujemy stan, jaki ma wystąpić dla danej kombinacji bitów na wejściu. Wartość funkcji jest określana przez wyświetlacz – jeżeli dla wyświetlenia danej liczby dany segment ma być włączony, to zapisujemy zero, a jeżeli nie – zapisujemy jeden. Tworząc taką tabelkę dla segmentu *a* wyświetlacza (górna kreska), chcemy, aby była ona włączona dla liczb: 0, 2, 3, 5, 6, 7, 8, 9, a wyłączona dla liczb 1 oraz 4. Przy stanach niedozwolonych, czyli od 10 do 15, zapisujemy kreskę. Poprawnie wypełniona tabelka dla segmentu *a* widoczna jest na **rysunku 1**. Zmienne wejściowe zostały oznaczone jako x3...x0 (kolor niebieski), gdzie x3 jest bitem najbardziej znaczącym. Wartość funkcji została zaznaczona kolorem zielonym. Kolor żółty reprezentuje kolejne liczby w systemie dziesiętnym, kolumna ta została dodana dla czytelności. Przy liczbach z zakresu od 10 do 15 postawiona została kreska, gdyż nie da się (na jednym wyświetlaczu) ich wyświetlić w postaci dziesiętnej.

Dla pozostałych segmentów należy przygotować tabelki w sposób analogiczny. Rysowanie siedmiu różnych tabelek nie jest wygodne, więc nic nie stoi na przeszkodzie, aby wszystko umieścić w jednej – **rysunek 2**.

Mając tak przygotowaną tabelkę, można przejść do kolejnego etapu – wyznaczenia funkcji

	wejścia				wyjście
liczba	x3	x2	x1	x0	a
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	-
11	1	0	1	1	-
12	1	1	0	0	-
13	1	1	0	1	-
14	1	1	1	0	-
15	1	1	1	1	-



Rys. 1

boole'owskich, które posłużą następnie do zaimplementowania dekodera w układzie CPLD. Wykorzystamy metodę określaną w literaturze mianem tablicy Karnaugha, która wymaga wykonania trzech kroków: narysowania następnej tabelki, wypełnienia jej i wykonania tzw. minimalizacji.

Konstrukcja tabeli oparta jest na kodzie Graya, a jej rozmiar jest zależny od liczby zmiennych występujących w funkcji. Kilka przykładów takiej tabelki pokazano na **rysunku 3**. W naszym przykładzie wykorzystamy niebieską tablicę przeznaczoną dla czterech zmiennych – tyle jest sygnałów na wejściu projektowanego dekodera. Zdziwienie może budzić „chaotyczna" numeracja poszczególnych komórek, ale wynika to z zastosowanego kodu Graya. W przypadku pierwszej kolumny pierwsza cyfra dotyczy zmiennej x3, a druga cyfra zmiennej x2. Podobnie jest w przypadku pierwszego wiersza – pierwsza cyfra odnosi się do zmiennej x1, a druga do x0. Wartości poszczególnych komórek (te zapisane w dolnym, prawym rogu) mają wartość wynikającą z zawartości pierwszej kolumny (dwa starsze bity) i pierwszego wiersza (dwa młodsze bity).

Mając przygotowaną tabelkę, można przystąpić do jej wypełniania. Czynność ta wymaga jedynie przepisania wartości funkcji (kolumna *wyjście* z rysunku 1), trzymając się zasady, że liczba z żółtej kolumny musi być równa numerowi komórki tablica Karnaugha. Kreski (wartości *don't care*) również przepisujemy. Wypełniona tabelka widoczna jest na **rysunku 4**.

Pozostał ostatni krok do wykonania – minimalizacja. Pierwszym etapem jest tzw. grupowanie jedynek (można grupować również zera, ale to pozostawiam dociekliwym do samodzielnego doczytania). Proces ten polega na zakreśleniu jedynek i ewentualnie znaków *don't care*. Wszystkie jedynki należy zakreślić obowiązkowo, natomiast znaki *don't care* tylko wtedy, gdy prowadzi to do zaznaczenia większego obszaru. Wyrażenie *don't care* oznacza w wolnym tłumaczeniu *bez znaczenia*, czyli wstawiając do tablicy taki znak nie ma znaczenia, czy będzie tam jedynka, czy zero. Łatwo zaobserwować to na przykładzie – czy po podaniu liczby 10 na wejście projektowanego obecnie dekodera segment *a* ma być włączony, czy wyłączony? Co za różnica, skoro i tak nie zamierzamy takiej liczby podawać?

Zawsze zakreślamy możliwie największe obszary, bo zmniejsza to liczbę zmiennych w poszczególnych składnikach wyrażenia. Efektem tego jest zmniejszenie liczby wejść bramek użytych do implementacji wyznaczonych funkcji i oszczędność zasobów układu CPLD.

Każda jedynka lub znak *don't care* mogą być zakreślone wielokrotnie. Oprócz tego trzeba pamiętać, że można zakreślić tylko obszary zawierające 1, 2, 4, 8, 16, itd. komórek. Zakreślane obszary mogą być jedynie prostokątami (kwadrat też jest prostokątem:)). Wolno również (wręcz należy) zakreślać obszary na skraju tablicy, np. gdyby pola 0, 2, 8 i 10 zawierały jedynki, to można je zakreślić razem. Inny podobny przypadek wystąpiłby, gdyby pola 12 i 14 zawierały jedynki – je również można razem zakreślić. Warto spojrzeć na tablicę Karnaugha jak na „kwadratową kulę" – górny i dolny kraniec tabeli jest połączony, podobnie jak lewy z prawym.

W przypadku tablicy dla 5 zmiennych należy pamiętać o dodatkowej zasadzie. Mianowicie, *jeżeli zakreślone pola znajdują się w obu połówkach tablicy, to w wyniku złożenia tej tablicy względem osi dzielącej ją na dwie symetryczne części zakreślony obszar powinien się dwukrotnie zmniejszyć i spełniać zasadę, że połączone pola muszą mieć kształt symetryczny względem swych osi (kwadraty lub prostokąty)*. (cytat: W. Głocki *Układy cyfrowe*). Na **rysunku 5** pokazano DOZWOLONE zakreślenia, a na **rysunku 6** ZABRONIONE.

Przykład zakreślenia dla segmentu *a* wyświetlacza pokazany został przed chwilą na rysunku 6. Widać tutaj, że wystąpił jeden prostokąt, jeden kwadrat i ŻADNA jedynka nie pozostała bez zakreślenia. Warto zauważyć, że nie było możliwości zakreślenia obszaru większego niż dwa pola.

Każde naniesione na tabeli zakreślenie tworzy jeden składnik funkcji. W przypadku tabeli z rysunku 4 będą dwa takie składniki – pierwszy wynika z zakreślenia kwadratem, a drugi z zakreślenia prostokątem. Przy tworzeniu takich składników obowiązują trzy zasady:

– ignorujemy zmienne, których wartość zmienia się w obrębie zakreślenia,

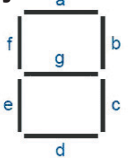
– zmienne, które się nie zmieniają i mają wartość 0 negujemy,

– na zmiennych wykonujemy iloczyn logiczny (bramka AND).

Zacznijmy od prostokąta. W jego obrębie zmienna x3 zmienia wartość, bo w drugim wierszu ma 0, a w trzecim 1, więc ją pomijamy. Pozostały zatem trzy zmienne: x2, x1 oraz x0. Zmienna x1 i x0 ma wartość zero, więc ulegną one zanegowaniu. Pozostało już wymnożyć zmienną x2 przez zanegowane zmienne x1 i x0. Ostatecznie pierwszy składnik ma postać: x2 * !x1 * !x0.

	wejścia				wyjścia						
liczba	x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0	0
10	1	0	1	0	-	-	-	-	-	-	-
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

Rys. 2a



Rys. 4

segment „a"

x3,x2 \ x1,x0	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	-	-	-	-
10	0	0	-	-

Rys. 3

2 zmienne

x1 \ x0	0	1
0	0	1
1	2	3

4 zmienne

x3,x2 \ x1,x0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

5 zmiennych

x4,x3,x2 \ x1,x0	00	01	11	10
000	0	1	3	2
001	4	5	7	6
011	12	13	15	14
010	8	9	11	10
110	24	25	27	26
111	28	29	31	30
101	20	21	23	22
100	16	17	19	18

3 zmienne

x2,x1 \ x0	0	1
00	0	1
01	2	3
11	6	7
10	4	5

Wykrzyknik będzie oznaczał symbol negacji.

Zajmijmy się teraz kwadratem. W tym wypadku zmienna x3, x2, x1 oraz x0 pozostają bez zmian, więc wszystkie cztery utworzą drugi składnik. Potwierdza to oczywiście wcześniejszą tezę, że zakreślenie większego obszaru oszczędza bramki – poprzedni składnik będzie wymagał bramki 3-wejściowej, a ten 4-wejściowej. Wracając do przykładu – tylko zmienna x0 ma wartość jeden, zatem wszystkie pozostałe będą zanegowane, co da składnik o postaci:

!x3 * !x2 * !x1 * x0.

Otrzymane w ten sposób składniki następnie sumujemy (bramka OR), zatem ostateczna postać funkcji dla segmentu *a* to:

fa(x3,x2,x1,x0) = x2 * !x1 * !x0 + !x3 * !x2 * !x1 * x0

Wyznaczenie pozostałych sześciu funkcji przebiega analogicznie. W ramach przykładu na **rysunku 7** zostały pokazane tabelki dla pozostałych segmentów, a poniżej wypisano wszystkie funkcje, aby umożliwić Czytelnikom sprawdzenie własnych wyników (nawiasy dla czytelności).

$f_a(x3, x2, x1, x0) = (x2 * !x1 * !x0) + (!x3 * !x2 * !x1 * x0)$

$f_b(x3, x2, x1, x0) = (x2 * !x1 * x0) + (x2 * x1 * !x0)$

$f_c(x3, x2, x1, x0) = (!x2 * x1 * !x0)$

$f_d(x3, x2, x1, x0) = (x2 * !x1 * !x0) + (!x3 * !x2 * !x1 * x0) + (x2 * x1 * x0)$

$f_e(x3, x2, x1, x0) = (x2 * !x1) + x0$

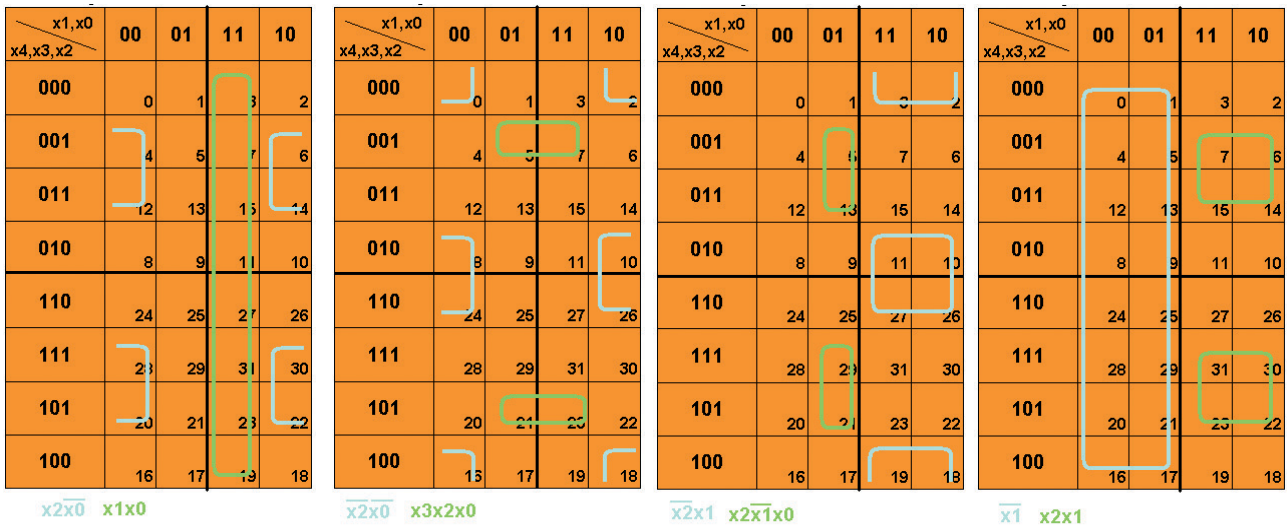
$f_f(x3, x2, x1, x0) = (!x3 * !x2 * x0) + (x1 * x0) + (!x2 * x1)$

$f_g(x3, x2, x1, x0) = (!x3 * !x2 * !x1) + (x2 + x1 + x0)$

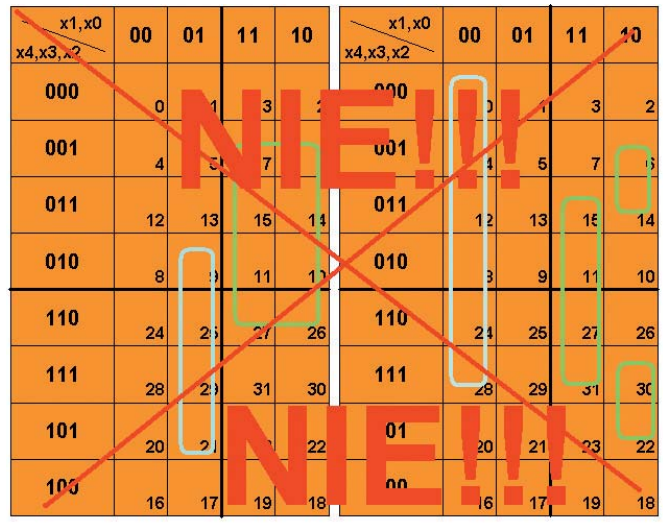
Komputerowe wspomaganie pracy z tablicami Karnaugha

Warto przeliczyć kilka takich tabelek na piechotę, aby nabrać lepszej orientacji w tym zagadnieniu. Do dyspozycji pozostają jeszcze dedykowane programy, które mają tę zaletę, że nie wymagają ręcznego rysowania tabelki i dokonują minimalizacji automatycznie. Przykładem jest program pokazany na **rysunku 8** – Karnaugh Minimizer (dostępny np. tu: http://www.brothersoft.com/karnaugh-minimizer-13861.html). Jego obsługa nie jest trudna, z tym że wersja trial ograniczona jest do tablicy o wymiarach 4x4 (cztery zmienne). Jest ona jednakże wystarczająca do wyznaczenia funkcji dla omawianego dekodera 7-segmentowego. Można także pracować na mniejszych tablicach, zmieniając położenie suwaka zaznaczonego czerwoną obwódką na rysunku 8. Po uruchomieniu warto zmienić opcje językowe, tak aby program przemawiał do nas w ojczystym języku. Sprowadza się to do wybrania menu *Tools*, następnie *Options*, *Language* i zaznaczenia pozycji *Polish* na liście i kliknięcia *OK*.

Nazwy zmiennych w programie również można dowolnie zmieniać po wybraniu polecenia *Opcje* z menu *Narzędzia*



Rys. 5



Rys. 6

Rys. 7

segment „b”

x1,x0 / x3,x2	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	-	-	-	-
10	0	0	-	-

segment „c”

x1,x0 / x3,x2	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	-	-	-	-
10	0	0	-	-

segment „d”

x1,x0 / x3,x2	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	-	-	-	-
10	0	0	-	-

segment „e”

x1,x0 / x3,x2	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	-	-	-	-
10	0	1	-	-

segment „f”

x1,x0 / x3,x2	00	01	11	10
00	0	1	1	1
01	0	0	1	0
11	-	-	-	-
10	0	0	-	-

segment „g”

x1,x0 / x3,x2	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	-	-	-	-
10	0	0	-	-

– w zakładce *Tablica*. Wspomnę, że sąsiednia zakładka, *Oznaczenia*, pozwala określić, jak ma wyglądać symbol negacji oraz sumy modulo 2 (operacja XOR).

Praca z programem może odbywać się na dwa sposoby. Po pierwsze, można wypełnić omówioną powyżej tablicę Karnaugha. Dwukrotne kliknięcie myszką w wybranym polu tabeli sprawi, że pojawi się w nim jedynka logiczna. Ponowne kliknięcie zamieni ją na logiczne zero, a jeszcze jedno – na symbol gwiazdki odpowiadającej symbolowi *don't care*. Po wypełnieniu całej tabeli wybiera się polecenie *Analizuj*, dostępne na pasku menu i w lewym oknie otrzymujemy zminimalizowane wyrażenie. Przykład pokazano na **rysunku 9** – widoczna jest tu funkcja wyznaczona dla segmentu *a* wyświetlacza.

Drugi sposób przygotowania tabelki wymaga kliknięcia na pasku menu ikony *Tabelka prawdy*, co spowoduje wyświetlenie tabeli analogicznej do tej z rysunku 1. Są w niej podane trzy reprezentacje kolejnych liczb (binarna, heksadecymalna oraz dziesiętna) i znajduje się tam kolumna *Fn*, do której wpisuje się wartość funkcji dla danego słowa wejściowego. W świetle naszego dekodera 7-segmentowego do kolumny *Fn* wpisujemy zero, gdy segment ma być włączony dla danej liczby, a jedynkę, gdy ma być wyłączony. Dla liczb większych od 9 możemy wpisać gwiazdkę – symbol *don't care*. Po kliknięciu *OK* wartości z tak wypełnionej tabelki zostaną przeniesione automatycznie do tablicy Karnaugha, co pozwoli skorzystać z polecenia *Analizuj*. Na **rysunku 10** widoczny jest przykład wypełnienia „zwykłej" tabelki dla segmentu *a* wyświetlacza.

Podpowiem jeszcze, że program pokazuje w jaki sposób narysować, za pomocą bramek, funkcję zapisaną w tabelce. Po wydaniu polecenia *Analizuj* należy nacisnąć klawisz F9 i w otwartym okienku kliknąć przycisk *Rysuj*.

Implementacja dekodera 7-segmentowego

Mamy już wyznaczone funkcje boole'owskie dla wszystkich segmentów. Pozostaje pytanie: jak sprawdzić, czy zostały one wyznaczone w sposób prawidłowy? Można je oczywiście porównać z tymi wyżej wypisanymi, ale co zrobić, gdy nie będzie takiej możliwości? Oczywiście zaimplementować! Do zadawania wszystkich dziesięciu dopuszczalnych słów wejściowych wykorzystamy licznik modulo 10. Jego wyjścia zostaną podłączone do dekodera, a wyjścia dekodera do wyświetlacza. Spodziewamy się otrzymać kolejne cyfry na wyświetlaczu.

Przyjrzyjmy się pierwszej funkcji przeznaczonej dla segmentu *a*. Dla czytelności poszczególne składniki zostały umieszczone w nawiasach. Zawartość tych nawiasów stanowi iloczyn logiczny, z tym że niektóre składniki są zanegowane. Jak wiemy z poprzednich części kursu, do schematu można wstawić bramkę z zanegowanymi wejściami. Możemy zatem stwierdzić, że dla przyjętej konwencji zapisu każdy nawias stanowi jedną taką bramkę. Nawiasów jest 15, więc tyle potrzebnych będzie bramek AND (gdybyśmy mieli do dyspozycji tylko bramki dwuwejściowe, potrzebowalibyśmy ich więcej). Bardziej ogólnym stwierdzeniem jest, że liczba bramek AND jest równa liczbie zakreśleń w tablicach Karnau-gha. Po dodaniu do schematu wszystkich bramek AND i doprowadzeniu do ich wejść odpowiednich sygnałów ze słowa wejściowego, konieczne są jeszcze sumowania za pomocą bramki OR. Przykład implementacji wyznaczonych funkcji z dodanym licznikiem modulo-10 pokazano na **rysunku 11**.

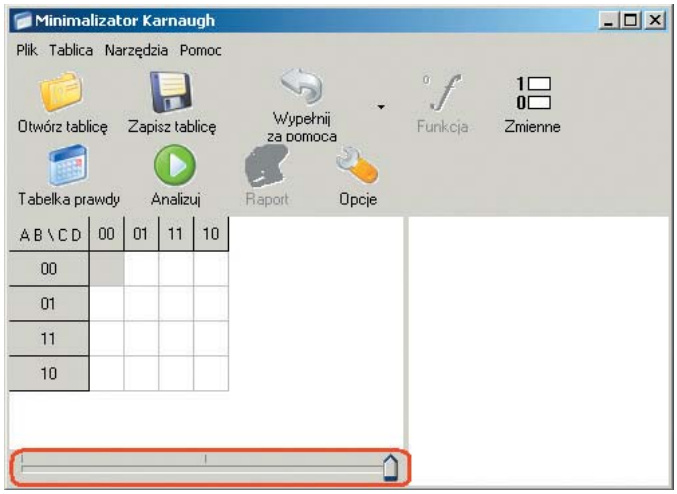
Wykorzystaliśmy tutaj różne typy bramek logicznych, które nie są spotykane w serii układów TTL czy CMOS (np. poczwórna bramka z trzema zanegowanymi wejściami). Nic nie stoi na przeszkodzie, oprócz zmniejszenia czytelności schematu, aby ten sam projekt zaimplementować za pomocą podstawowych bramek 2-wejściowych i negatorów. Chcąc zwielokrotnić liczbę wejść bramek AND czy OR, należy połączyć je w sposób kaskadowy. Na **rysunku 12** pokazano realizację bramek 3-wejściowych i 4-wejściowych (AND i OR). Takie połączenie pozwala na budowę omawianego dekodera z bramek serii np. 74XX.

Zachęcam do uruchomienia przykładu pokazanego na rysunku 11 i zapoznania się z rezultatami jego pracy. Przypominam o konieczności włączenia przynajmniej jednego tranzystora, aby wyświetlacz mógł świecić (na rysunku 11 włączone są oba).

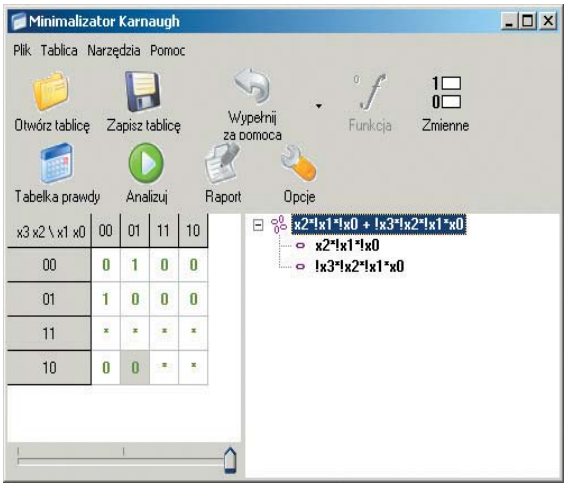
Tworzenie własnego symbolu bibliotecznego

Na zaimplementowanie dekodera 7-segmentowego poświęciliśmy trochę czasu i warto uczynić go gotowym, uniwersalnym elementem, przygotowanym do użycia w innych projektach. Oszczędzi to ponownego wyznaczania funkcji boole'owskich i ich rysowania z wykorzystaniem bramek. Co więcej, własny symbol biblioteczny znacznie zwiększa czytelność projektu. Wystarczy spojrzeć na rysunek 11 – ogarnięcie wszystkich połączeń nie jest wcale takie proste.

Utworzenie własnego symbolu nie jest rzeczą trudną i wymaga wykonania kilku prostych kroków. Pierwszym z nich jest upewnienie się, że narysowany schemat jest poprawny. Zostało to już zrobione – kolejne stany licznika modulo 10 z rysunku 11 są prezentowane poprawnie.



Rys. 8

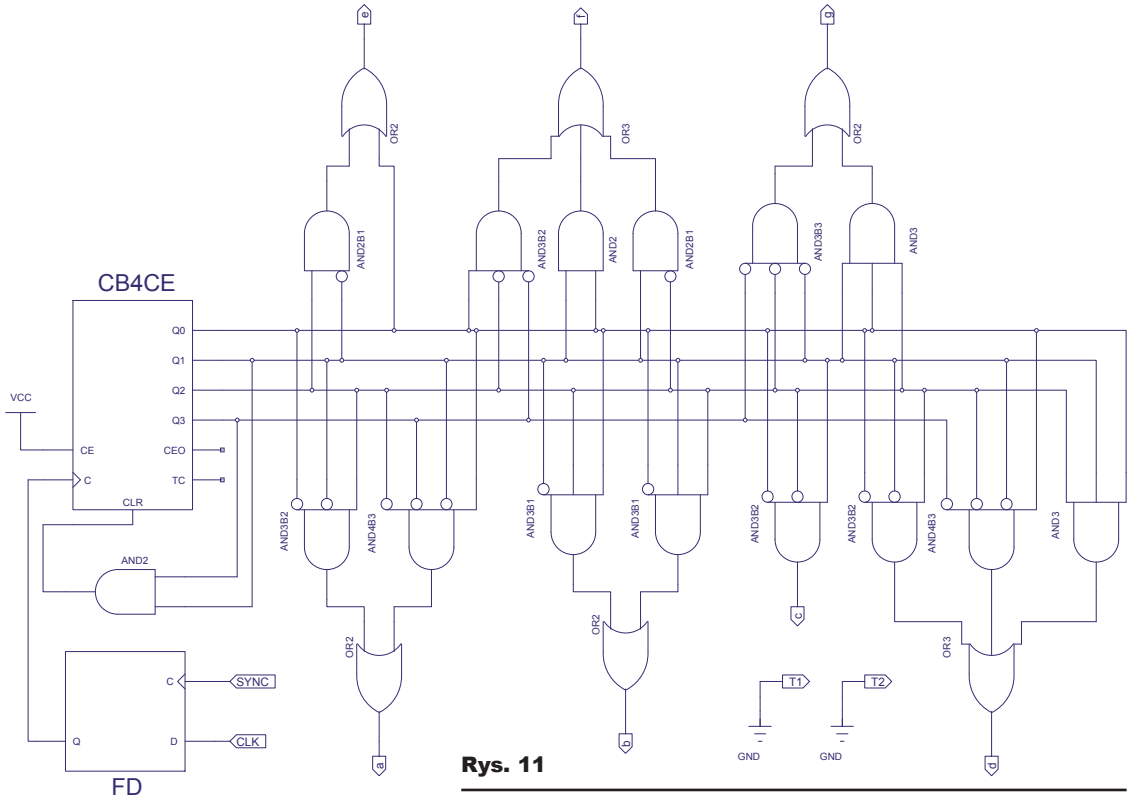


Rys. 9

Rys. 10

Drugim krokiem jest określenie wejść i wyjść. Wejściem w tym przypadku jest liczba zapisana w sposób binarny, czyli sygnały x0... x3 podłączone obecnie do wyjść licznika. Wyjściami są poszczególne sygnały sterujące pracą segmentów wyświetlacza. Chcemy otrzymać element posiadający wejścia x0..

x3 służące do zadawania liczby binarnej oraz mający wyjścia *a...g*, które podłączone do segmentów zagwarantują poprawne wyświetlenie cyfry. Tu pojawia się dobra wiadomość: połowę pracy mamy z głowy. Zdefiniowanie wejść i wyjść sprowadza się do narysowania odpowiednich markerów (I/O Marker). Wyjściowe porty zostały już narysowane, brakuje jedynie wejść. W tym celu należy usunąć ze schematu zbędne elementy (licznik, przerzutnik oraz porty tranzystora) i dorysować cztery markery wejściowe w miejsce wyjść licznika. Tak zmodyfikowany schemat widoczny jest na **rysunku 13**. Nie ma potrzeby tworzenia pliku **.ucf* zawierającego przypisania wyprowadzeń układu CPLD do poszczególnych markerów. Schemat zapisujemy pod dowolną nazwą, niech to będzie *dek7seg_symbol.sch*.

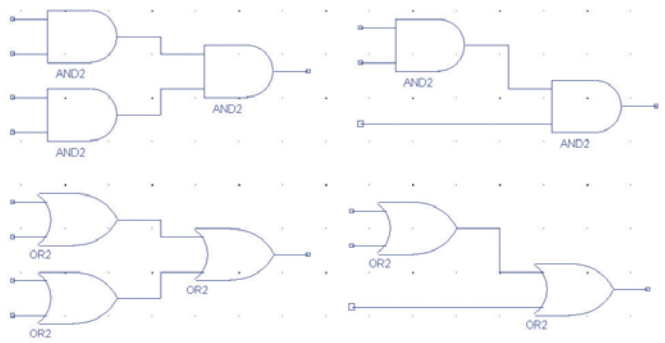


Rys. 11

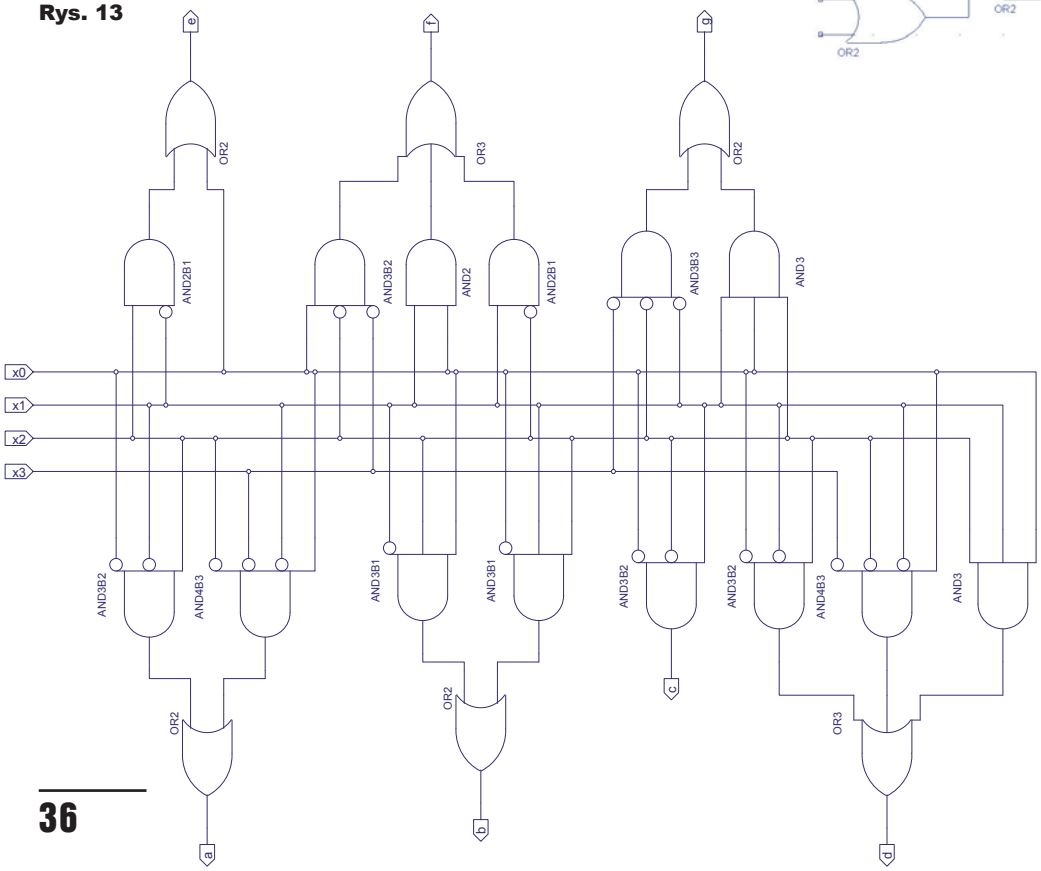
Ostatnim krokiem jest uruchomienie kreatora. Zasadniczo nie jest on obowiązkowy, gdyż umożliwia jedynie określenie, jak ma wyglądać nasz symbol: rozłożenie i kolejność wejść oraz wyjść, symbole negacji, kształt obudowy, etc. Niemniej pokażę w jaki sposób się go obsługuje, gdyż pozwala poprawić czytelność tworzonych symboli.

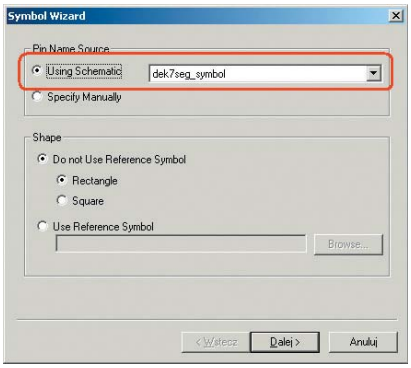
Pracę kreatora rozpoczyna się od wybrania z menu T*ools* pozycji *Symbol Wizard* (w edytorze schematów, w którym znajduje się przygotowana implementacja symbolu, ta z rysunku 13). Ukazuje się pierwsze okienko, w którym zaznaczamy pozycję *Using Schematic* i sprawdzamy, czy obok znajduje się właściwa nazwa schematu (**rysunek 14**). W kolejnym okienku

Rys. 12



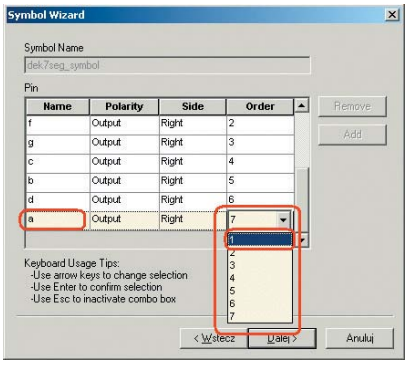
Rys. 13





Rys. 14

Rys. 15



wyszczególnione są wszystkie sygnały, jakie dostępne będą w nowym symbolu. Warto dokonać w tym miejscu pewnej reorganizacji. Przy symbolu *a* powinien znajdować się numer 1, przy *b* numer 2, itd. Podobnie przy sygnałach wejściowych x0...x3. Zmiany numeru dokonuje się poprzez kliknięcie ostatniego pola i wybranie stosownej pozycji – **rysunek 15**. Decyduje to o rozmieszczeniu sygnałów w elemencie – jeżeli liczby będą przypisane stosownie do sygnałów, to wyprowadzenia zostaną ułożone alfabetycznie. Ma to znaczenie przede wszystkim estetyczne.

Następne pole (**rysunek 16**) umożliwia określenie wymiarów „obudowy" elementu, odległości pomiędzy wyprowadzeniami, etc.

Ostatnie okienko (**rysunek 17**) pokazuje ,jak będzie wyglądał utworzony symbol. Nie da się tu już nic zmienić, więc klikamy *Zakończ*. Można w razie potrzeby skorzystać z opcji *Wstecz*, aby nanieść poprawki – przeorganizować rozłożenie pinów elementu czy zmienić rozmiar obudowy. Po zakończeniu pracy kreatora ukazuje się nowo utworzony element – **rysunek 18**. W tym miejscu można zapisać postępy pracy (*File->Save*), jednakże dokonamy jeszcze jednej, kosmetycznej modyfikacji. Włączenie segmentu wyświetlacza odbywa się stanem niskim, więc sygnałem aktywnym jest poziom L. Na różnych schematach takiej konwencji towarzyszy kółko znajdujące się przy danym wyprowadzeniu. Można je dodać także w tym miejscu poprzez wybranie z menu *Add* pozycji *Bubble* (CTRL+B). Dodajemy do schematu siedem takich okręgów, zaznaczamy wszystkie wyprowadzenia wyjściowe, przesuwamy je i wstawiamy między „obudowę" elementu a wyprowadzenia. Efekt powinien być porównywalny z tym z **rysunku 19**. Teraz można zapisać zmiany i zamknąć edytor.

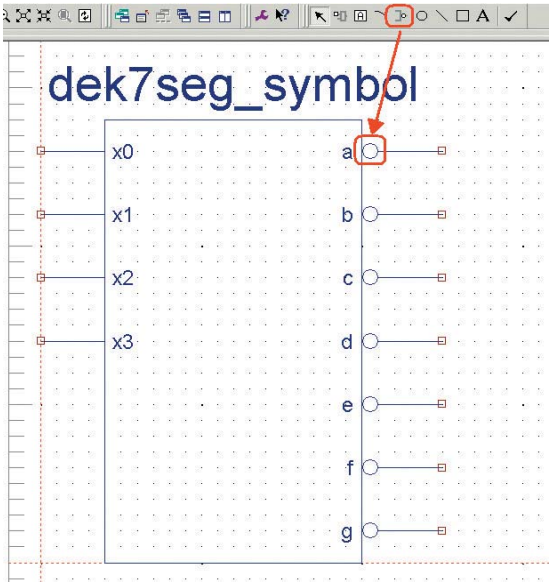
Licznik modulo 100

Mając opracowany dekoder 7-segmentowy, spróbujmy przygotować bardziej rozbudowany licznik – liczący do 100.

Proponuję utworzyć nowy projekt w *Project Navigator* (zgodnie ze wskazówkami z drugiej części kursu). Najpierw należy dodać nasz nowy element do biblioteki. Zadanie jest bardzo proste i sprowadza się do paru kliknięć myszką. Po pierwsze, należy wybrać z menu *Project* pozycję *Add Source*. W otwartym oknie WRACAMY do folderu, w którym znajduje się schemat elementu – *dek7seg_symbol.sch*, zaznaczamy go i dodajemy do projektu. Jest on teraz widoczny w lewej części okna, więc RAZ na niego klikamy (nie otwierając), a następnie rozwijamy opcję *Design Entry Utilities* i klikamy *Create Schematic* Symbol (**rysunek 20**). To wszystko. Nowy symbol jest dostępny w edytorze schematów w obecnym projekcie – można się o tym przekonać, klikając dwukrotnie na pliku schematu, otwierając w ten sposób edytor i zaglądając do zakładki. Tak utworzony element można dodać do schematu jak każdy inny symbol z biblioteki – **rysunek 21**.

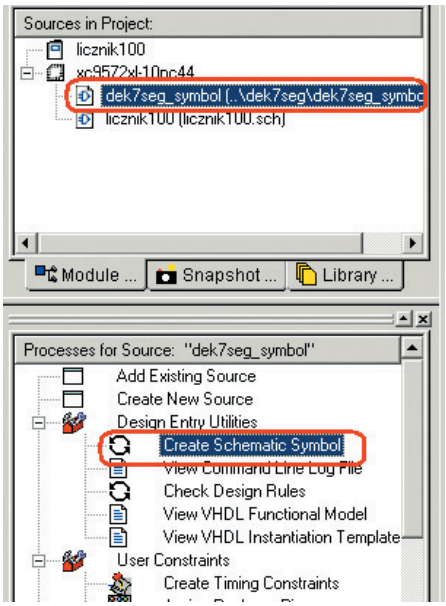
Pozostaje odpowiedzieć na pytanie: jak zbudować licznik liczący do stu? Zasadniczo można by wstawić tradycyjny licznik 8-bitowy i zliczać za jego pomocą. Pojawia się jednak problem wykonania stosownego dekodera. Zamiast 4 zmiennych mamy ich osiem, w najlepszym wypadku siedem. Stwarza to duże wyzwanie dla projektowania metodą tablicy Karnaugha. Nasuwa się jednak wniosek, że oba wyświetlacze pracują niejako niezależnie – jeden liczy jedności, a drugi dziesiątki. Do takiego zliczania można wykorzystać dwa liczniki modulo: 10 – pierwszy, taktowany sygnałem z generatora, będzie zliczał jedności, a drugi, taktowany przebiegiem wyjściowym z pierwszego licznika (częstotliwość generatora ulegnie podzieleniu przez dziesięć) będzie liczył dziesiątki. Takie rozwiązanie umożliwia zastosowanie już opracowanego dekodera, bo posiadamy dwie liczby 4-bitowe na wyjściach liczników, które mogą zostać dołączone do znanego już dekodera. Należy zwrócić uwagę, że liczniki reagują na zbocze narastające. W związku z tym niezbędna jest bramka NOT przed wejściem sygnału zegarowego licznika dziesiątek. Na pytanie „dlaczego" nie odpowiem – pozostawiam to Czytelnikom do ewentualnego sprawdzenia:). W razie problemów ze znalezieniem wyjaśnienia dlaczego „tak" się dzieje, warto narysować przebiegi na wyjściu licznika jedności i przyjrzeć się przebiegowi na wyjściu Q3, pamiętając, że licznik zwiększa swój stan podczas zbocza narastającego.

Pojawia się jednak kolejny problem – w jaki sposób podłączyć utworzony przez nas dekoder 7-segmentowy? Oba wyświetlacze 7-segmentowe mają wspólną magistralę, więc nie można ich dołączyć bezpośrednio. Rozwiązaniem może być multipleksowany sposób sterowania wyświetlaczami – najpierw włączamy pierwszy wyświetlacz i wyświetlamy jedności, a potem drugi, żeby wyświetlić dziesiątki. Powtarzając te dwie czynności cyklicznie i odpowiednio szybko (powyżej 30 razy na sekundę), uzyskamy złudzenie jednoczesnego prezentowania całej liczby (bezwładność ludzkiego oka). Mamy zatem dwa liczniki i dekoder, potrzebny jest element

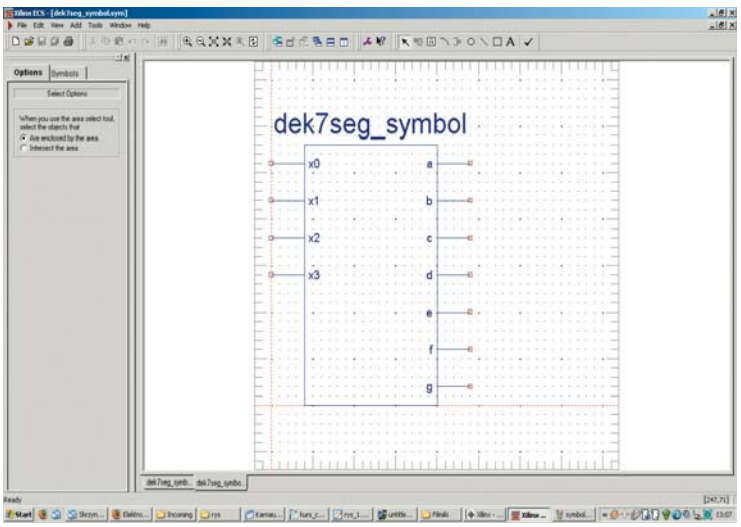


Rys. 19

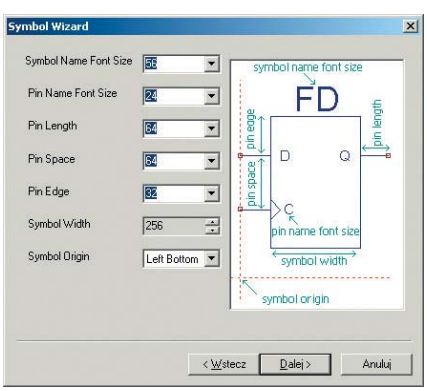
Rys. 20



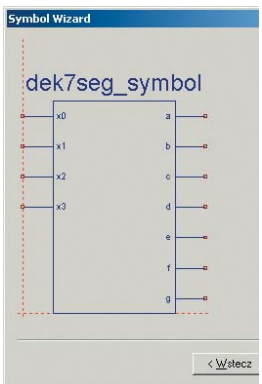
Rys. 18

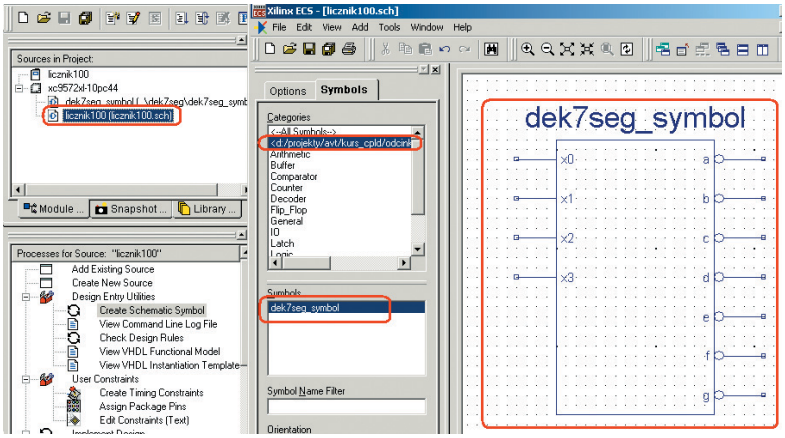


Rys. 16



Rys. 17





Rys. 21

przełączający wejścia dekodera raz do licznika jedności, a raz do licznika dziesiątek. Być może pamiętasz z drugiej części kursu, że takie zadanie pełni multiplekser. Potrzebujemy przełączenia pomiędzy dwoma sygnałami, więc odpowiedni będzie multiplekser 1-bitowy – ma on dwa wejścia danych i jedno wejście sterujące określające, z którego wejścia danych pobrać sygnał wyjściowy. Potrzebne będą cztery takie elementy, bo liczniki mają po cztery wyjścia. Pozostało ostatnie pytanie: w jaki sposób pozyskać sygnał przełączający multipleksery? Posiadają one jedno wejście sterujące, zatem potrzebne jest zero albo jeden – na zmianę. Taki sygnał sterujący zapewnia każdy generator przebiegu prostokątnego. Tu wykorzystamy obecny na płytce generator kwarcowy. Niestety nie można użyć sygnału pobranego bezpośrednio z tego generatora, gdyż ma on za dużą częstotliwość, która przekracza możliwości pracy wyświetlacza. Konieczne jest zastosowanie prostego dzielnika częstotliwości, czyli jeszcze jednego licznika, z którego pobierany będzie sygnał sterujący. Optymalny będzie sygnał pobrany z wyjścia Q(15) licznika 16-bitowego. Czytelnicy mogą również sprawdzić jak wygląda praca wyświetlacza przy zbyt dużej częstotliwości – wystarczy usunąć wspomniany licznik.

Mając opracowaną koncepcję układu, można przystąpić do jego implementacji. Przykład takowej przedstawiono na **rysunku 22**.

wyjście y

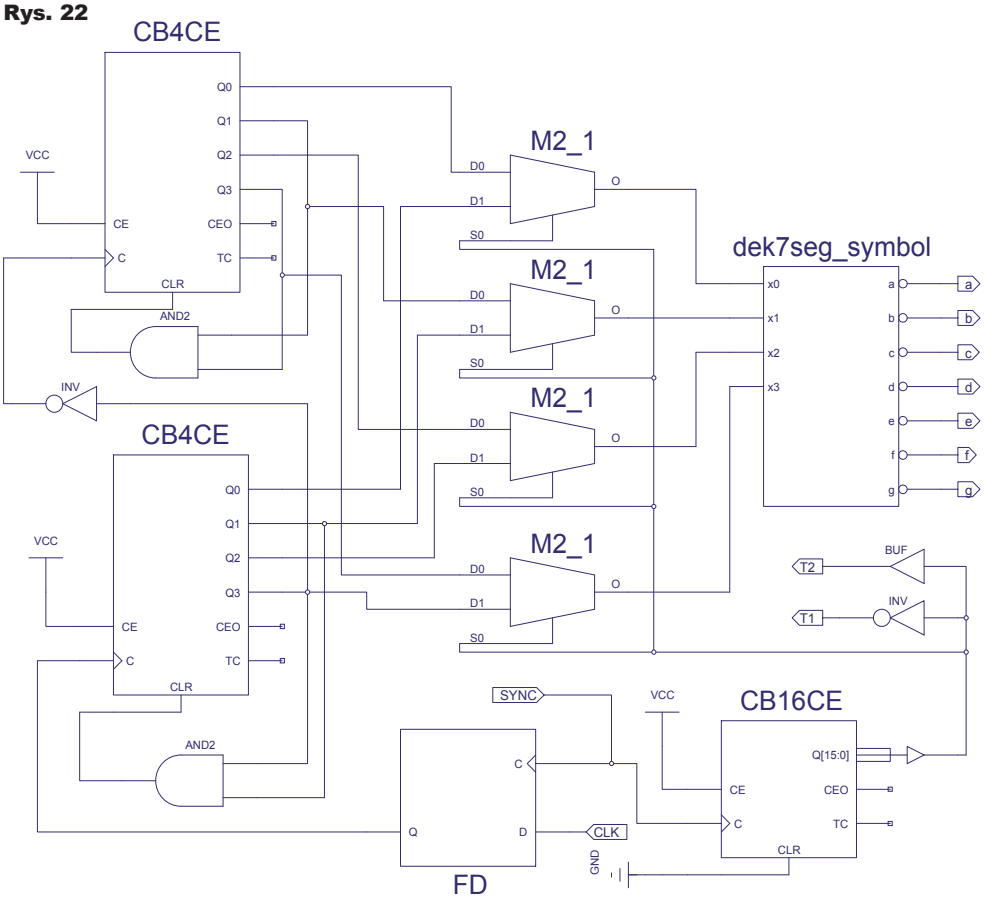
a	b	d	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

wyjście przeniesienia c

a	b	d	c
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Rys. 23

Rys. 22



Sumator 1-bitowy

Ostatnim zagadnieniem, które dziś zostanie poruszone, to prosty sumator umożliwiający dodawanie liczb jednobitowych. Jego przydatność jest w zasadzie zerowa, bo kto chciałby dodawać 1+1? Pewnie nikt, dlatego wykażemy więcej finezji i zaprojektujemy układ tak, aby można go... łączyć kaskadowo! W takiej sytuacji w bardzo prosty sposób będzie można wykonać sumatory n-bitowe. Tym razem również będziemy dążyć do uzyskania zgrabnego elementu, który będzie można przenieść do biblioteki elementów i łączyć w kaskadę.

A jakie powinny być zdefiniowane wejścia oraz wyjścia takiego elementu? Zacznijmy od wyjść. Niezbędne jest wyjście reprezentujące wynik, dajmy na to *y*. No niby się zgadza, ale jeżeli są dwa wejścia jednobitowe, to wszystko jest w porządku, gdy dodajemy 0+0 lub 1+0 – wynik zajmuje jeden bit. Co w sytuacji, gdy przychodzi dodać 1+1? Rezultat wynosi 2 i nie można go zapisać w jednym bicie. Konieczne jest jeszcze jedno wyjście informujące o przepełnieniu – nazwijmy je *c* (analogiczne rozwiązanie spotyka się w jednostkach ALU mikrokontrolerów i nie tylko). Sens tego wyjścia jest podobny jak przy dodawaniu pisemnym, bo dodając 6+5, zapisuje się 1, a dziesiątka jest przenoszona do następnej kolumny i uwzględniana w dalszych obliczeniach.

Zajmijmy się teraz wejściami. Niezbędne będą dwa wejścia danych, dajmy na to *a* oraz *b*. Należy pamiętać, że dodaliśmy również wyjście informujące o przeniesieniu, więc konieczne jest jego uwzględnienie przy dodawaniu (ma to szczególne znaczenie przy kaskadowym łączeniu sumatorów). Z tego względu potrzebne jest stosowne wejście informujące o przepełnieniu w poprzednim sumatorze. Niech będzie ono oznaczone literą *d*.

Teraz można przystąpić do wyznaczania funkcji boole'owskich. Zadanie to nie będzie trudne, bo są trzy wejścia i dwa wyjścia. Wyjścia i tym razem można potraktować jako niezależne – jako osobne funkcje. Każda z nich będzie zależała od wartości trzech parametrów – wejść. Możemy przystąpić do narysowania tabelki, w której zgromadzimy wszystkie dopuszczalne kombinacje słów wejściowych i odpowiadające im stany wyjść. Przykład takich tabelek pokazano na **rysunku 23**. Do przeprowadzenia minimalizacji wykorzystamy zieloną tablicę Karnaugha z rysunku 3 przeznaczoną dla trzech zmiennych. Umieszczenie w niej danych przebiega analogicznie do omówionego wcześniej – **rysunek 24**.

Minimalizacja pomarańczowej tablicy Karnaugha nie nastręcza problemów. Co jednak począć z zieloną tablicą? Przypadek wygląda na beznadziejny, gdyż jedynek nijak nie da się pogrupować. Na szczęście można wykorzystać pewną sztuczkę – gdy jedynki układają się po przekątnych (po dwie, cztery, etc.), to najprawdopodobniej da się zastosować bramkę Ex-OR lub Ex-NOR. Przypominam, że bramka Ex-OR (w bibliotece występuje jako XOR) daje na wyjściu jedynkę, gdy liczba jedynek na wejściu jest nieparzysta. Tak jest w przypadku zielonej tabelki z rysunku 23 – jedynka pojawia się na wyjściu tylko dla nieparzystej liczby jedynek na trzech wejściach. Okazuje się, że jedna, 3-wejściowa bramka XOR rozwiązu-

je problem minimalizacji. Ostatecznie funkcje boole'owskie sumatora są następujące:

$f_y(a,b,d) = a$ XOR b XOR d

$f_c(a,b,d) = (b * d) + (a * b) + (a * d)$.

Zaimplementowany sumator widoczny jest na **rysunku 25**. Na **rysunku 26** natomiast przedstawiono wersję trochę bardziej rozbudowaną, gdyż obecne są tu dwa sumatory. Pierwszy dokonuje właściwego dodawania, natomiast drugi uwzględnia tylko przeniesienie z pierwszego. Wyjścia zawierające wynik zostały dołączone do dobrze znanego dekodera 7-segmentowego. Przyciskami S1 oraz S2 można dokonywać dodawania. Obecność bramek NOT wymusza podawanie na wejścia sumatora stanu niskiego w momencie, gdy puszczony jest dany przycisk.

Mógłbym oczywiście uprościć schemat z rysunku 26 poprzez wstawienie symboli sumatorów. Nie zrobiłem tego jednak, aby „zmusić" Czytelnika do samodzielnej modyfikacji schematu.

Zadnie domowe :)

Trzeci odcinek kursu zbliża się do końca, więc przygotowałem dla Czytelników dwa zadanie do przemyślenia. Pozwolą ono trochę pogłówkować we własnym zakresie i utrwalić zdobyte w tej części kursu wiadomości.

Problem jest następujący: za pomocą trzech przycisków obecnych na płytce (S1, S2, S3) zadajemy dowolną liczbę z zakresu 0..7 w kodzie binarnym. Do tak zadanej liczby chcemy dodać wartość stałą – dwa. Rezultat dodawania powinien być wyświetlony na wyświetlaczu 7-segmentowym. Sugeruję tutaj wykorzystanie omówionego sumatora jednobitowego. Po dodaniu jego symbolu do biblioteki możliwe będzie szeregowe łączenie kilku takich sumatorów i dodawanie większych liczb niż 1-bitowe. Resztę rozważań pozostawiam Czytelnikom :).

Drugi problem jest następujący: jak zaimplementować sekundnik? Układ taki musiałby liczyć modulo 60, czyli od 0 do 59, zmieniając swój stan dokładnie co sekundę. Odradzam użycie przestrajanego generatora, bo jest zbyt mało dokładny.

W czasie wykonywania tych ćwiczeń postaraj się NIE modyfikować schematów utworzonych poprzednio. Zacznij nowy projekt i zrób to samodzielnie, w miarę możliwości nie zaglądając już do tego numeru EdW. Jeżeli czegoś nie pamiętasz, postaraj się wymyślić rozwiązanie samodzielnie.

Podsumowanie

W niniejszej części kursu skupiliśmy się na układach kombinacyjnych i związanych z nimi tablicami Karnaugha. Okazuje się, że można zrealizować różnorodne układy, mając daną tabelkę z wyszczególnionymi stanami wejściowymi oraz wyjściowymi. Opracowany element można szybko zamienić na element dostępny w bibliotece, co zwiększa czytelność schematu i pozwala wykorzystywać go wielokrotnie.

Niniejsza część kursu pokazała również, w jaki sposób pracować z wyświetlaczem 7-segmentowym. Nie wykorzystaliśmy tutaj najbardziej intuicyjnej drogi i nie użyliśmy licznika 8-bitowego. Spowodowane to było trudnością z przygotowaniem stosownego dekodera. Zamiast tego wykorzystany został trik z dwoma licznikami modulo-10, który koncepcyjnie jest trudniejszy, ale jego realizacja nie nastręcza tylu trudności. Z tej lekcji można wyciągnąć pewien wniosek: nie zawsze pierwszy i zdawałoby się najłatwiejszy pomysł na realizację jakiegoś zadania jest taki w rzeczywistości. W implementacji urządzeń cyfrowych najtrudniejsze jest właśnie opracowanie sensownej koncepcji. Znajomość narzędzi, takich jak tablice Karnaugha, bramki, układy programowalne, jest trochę mniej ważna, ale potrzebna, gdyż daje rozeznanie w ograniczeniach opracowywanych koncepcji. Młodszym i mniej doświadczonym Czytelnikom chciałbym powiedzieć, że eksperymentowanie i samodzielne myślenie pozwala poznać własne słabe strony i pracować nad nimi. Oprócz tego warto podpatrywać, jak to robią inni, gdyż cudze koncepcje można adoptować do własnych pomysłów i szybko zorientować się, co jest możliwe do zrobienia za pomocą danych narzędzi.

W tej lekcji pojawiły się propozycje zadań do samodzielnego przemyślenia. Warto się im przyjrzeć i postarać rozwiązać. Obawiam się, że nauka ograniczona do odtwarzania podanych przykładów nie jest zbyt efektywna. W związku z tym nie poprzestawaj na tym – do każdego przykładu podanego w ramach kursu spróbuj dodać coś samodzielnie. Kiedy przeczytasz dany odcinek, zapoznasz się z przykładami, postaraj się zrozumieć, jak funkcjonują, następnie zamknij EdW, usuń plik i zrób zadanie jeszcze raz – bez podpierania się przykładami. Zawsze zastanów się, czy nie da się danego problemu rozwiązać inaczej. Jeśli to możliwe, zawsze szukaj innej, lepszej drogi.

Jakub Borzdyński
jakub.borzdynski@elportal.pl

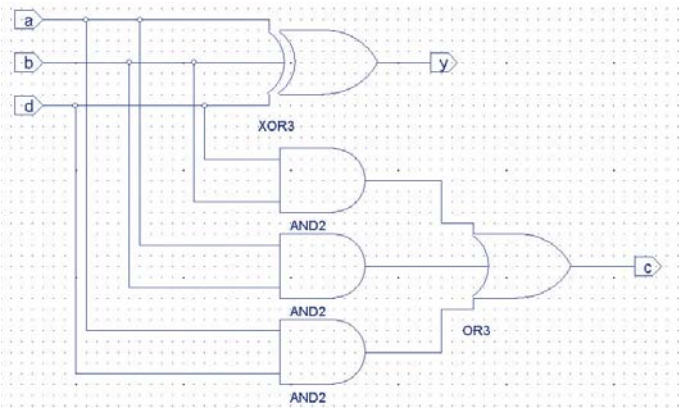
Rys. 24

wyjście y

a,b \ d	0	1
00	0	1
01	1	0
11	0	1
10	1	0

wyjście przeniesienia c

a,b \ d	0	1
00	0	0
01	0	1
11	1	1
10	0	1



Rys. 25

Rys. 26