PODSTAWY TECHNIKI CYFROWEJ

Pseudoskrypt w 24 stronach.

Spis treści:

1. Rejestry.

Nazwą rejestru określa się blok złożony z pewnej liczby n przerzutników, który zawiera pojedyncze słowo n-bitowe, przy czym wprowadzanie (wpisywanie) i wyprowadzanie (odczyt) tego słowa może następować równolegle lub szeregowo.

Podstawowe funkcje pełnione przez rejestry to:

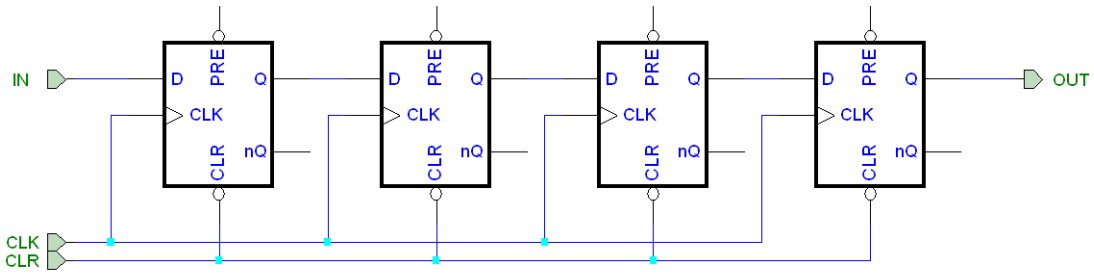
- rola pamięci pomocnicza
- buforowanie danych (np. między urządzeniami o różnej prędkości)
- zamiana postaci danych z szeregowej na równoległą i odwrotnie

Podstawowe cechy rejestrów:

- długość rejestru – liczba przerzutników użytych do budowy rejestru
- szybkość pracy rejestru
- czas propagacji
- maksymalna częstotliwość taktowania

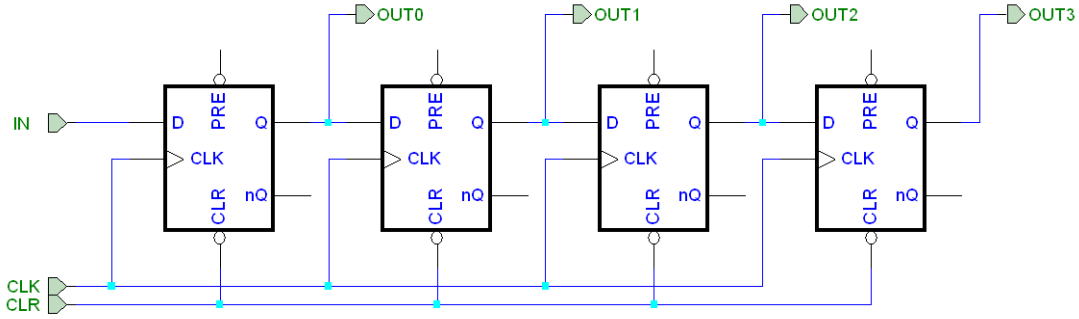
Rejestry o równoległym zapisie i odczycie nazywane są *równoległymi* (storage registers), a rejestry, w których choć jedna z tych operacji jest wykonywana szeregowo – rejestrami *przesuwającymi* (shift registers).

1.1. Rejestry szeregowo-szeregowe (SISO).



Rejestry szeregowo-szeregowe, jak sama nazwa wskazuje, charakteryzują się zarówno szeregowym wprowadzaniem, jak i szeregowym wyprowadzaniem danych, posiadają zatem pojedyncze wejście oraz pojedyncze wyjście danych. Działanie takiego rejestru sprowadza się do opóźnienia sygnału o n cykli zegara – sygnał z wejścia pojawia się na wyjściu po przejściu przez wszystkie przerzutniki.

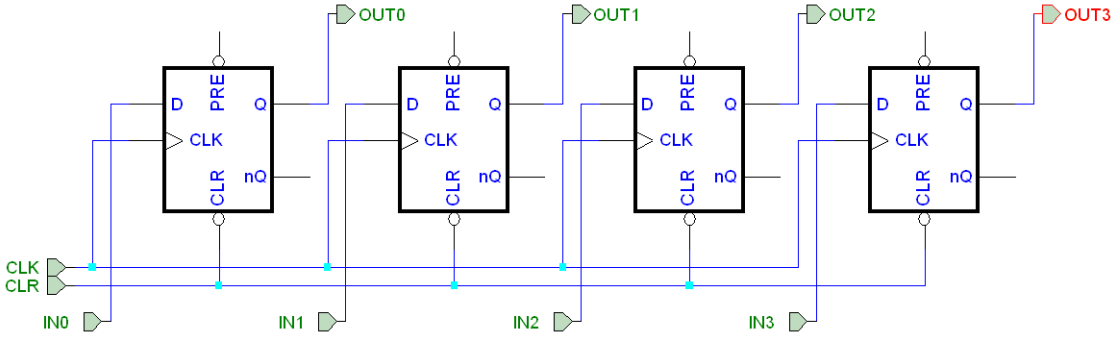
1.2 Rejestry szeregowo-równoległe (SIPO).



Rejestry szeregowo-równoległe służą do zamiany sygnału z postaci szeregowej na równoległą, stąd posiadają jedno wejście danych oraz n wyjść. Uzyskujemy dzięki temu jednoczesny dostęp do całego n-bitowego słowa przechowywanego w rejestrze.

Należy pamiętać, że na każdy takt sygnału zegarowego zawartość rejestru przesuwana jest tylko o jedną pozycję w prawo (wprowadzany jest kolejny bit do rejestru), stąd odczyt n-bitowej wartości w postaci równoległej musi następować co n taktów zegara, gdyż tyle zajmuje załadowanie kolejnych n bitów z wejścia szeregowego do rejestru. Najprostszym rozwiązaniem jest dodanie licznika modulo n, który będzie odmierzał czas między kolejnymi operacjami odczytu oraz układu zatrzaskującego odczytaną wartość (np. przerzutniki D taktowane wspomnianym licznikiem), aby była ona dostępna od jednej operacji odczytu aż do wystąpienia następnej.

1.3 Rejestry równoległo-równoległe (PIPO).

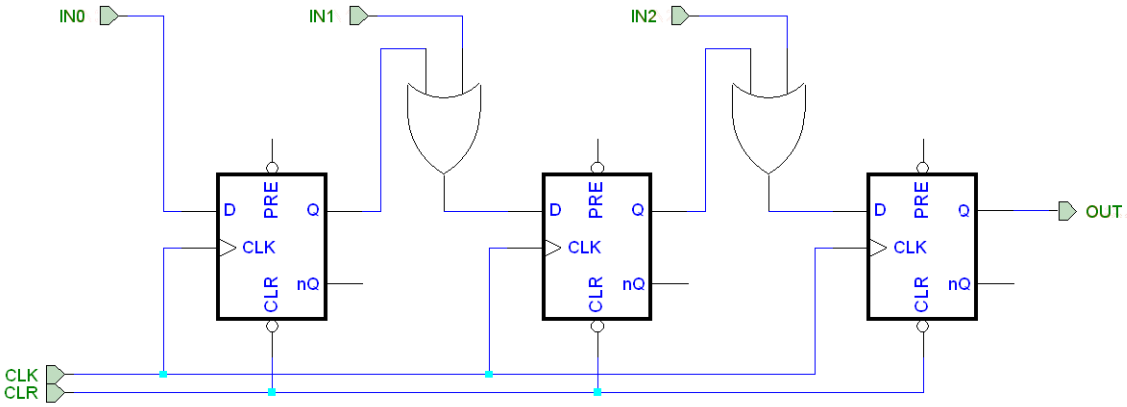


Rejestr równoległo-równoległy zwany jest często rejestrem buforowym. Zarówno dane wejściowe, jak i wyjściowe dane są w postaci równoległej, ładujemy zatem całe n-bitowe słowo, by później móc je w całości odczytać.

Podstawową zaletą tego układu jest fakt, że w momencie aktywnego zbocza zegara dane na wyjściu są zatrzaskiwane aż do pojawienia się następnego impulsu zegarowego i nie zależą w tym czasie od wejść (co tłumaczy nazwę *buforowy*). Pozwala nam to odczytywać dane co określony kwant czasu, odmierzany np. przez licznik, co było już nam potrzebne w punkcie 1.2 do zamiany postaci sygnału.

Wejście zegarowe można również rozumieć jako linię LOAD, która zboczem aktywnym zapisuje wartość z wejść danych do bufora, zatrzaskuje ją w nim i wystawia na wyjściach aż do kolejnego załadowania bądź zresetowania bufora.

1.4 Rejestry równoległo-szeregowe (PISO).

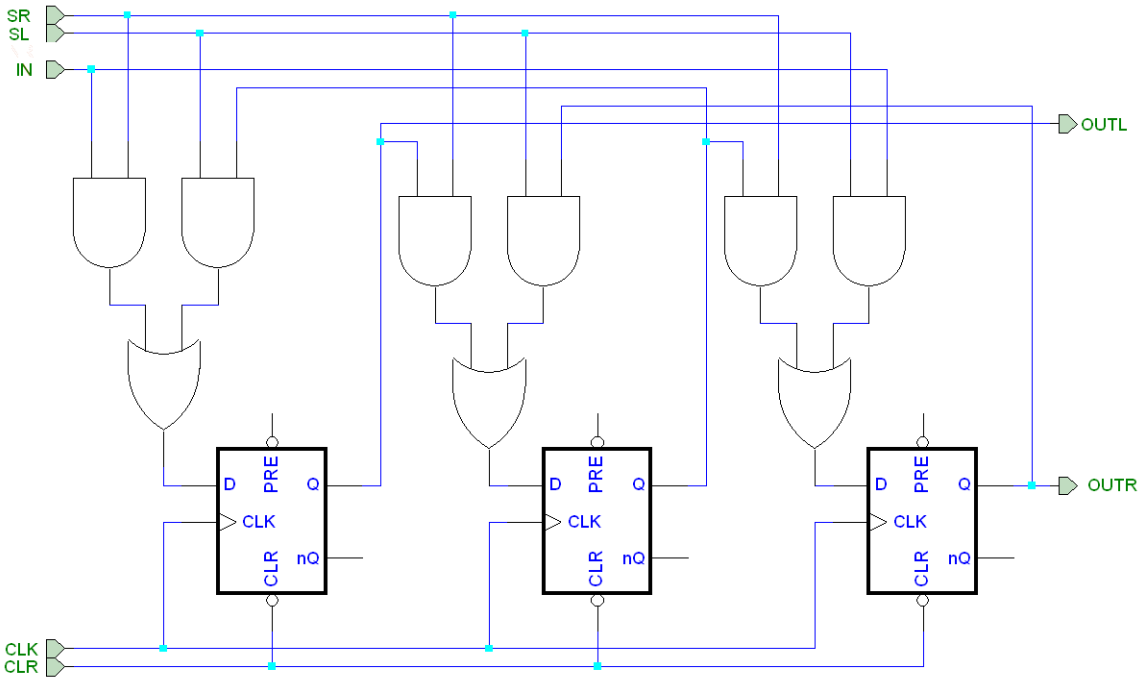


Analogicznie do rejestrów SIPO, rejestry równoległo-szeregowe służą do zamiany postaci sygnału, ale w przeciwnym kierunku – z n-bitowej postaci równoległej na szeregową. W związku z tym posiadają n wejść danych oraz pojedyncze wyjście, na którym wraz z kolejnymi taktami zegara pojawia się podane n-bitowe słowo w postaci szeregowej.

Przedstawiony powyżej schemat jest tylko poglądowy, gdyż pojawia się podobny problem jak w przypadku SIPO, z tą różnicą, że tym razem to zapis danych do rejestru musi następować co n taktów zegara. O ile w poprzednim przypadku różnica prędkości odczytu i zapisu nie powodowała większych komplikacji, o tyle tutaj przed zapisem należy wszystkie przerzutniki zresetować sygnałem CLR, a po zapisaniu danych do rejestru ustawić wszystkie wejścia w stan niski, co umożliwi poprawne przesuwanie zawartości rejestru w prawo (pozostawienie stanu wysokiego na dowolnym wejściu spowodowałoby błędy podczas przesuwania).

Alternatywnie zapis danych do rejestru można zrealizować korzystając z wejść PRE i CLR oraz dodatkowego wejścia LOAD – wówczas w zależności od sygnału na wejściu, podczas aktywnego poziomu na linii LOAD pojawiałby się niski poziom na wejściu PRE oraz wysoki na CLR (gdy na wejściu IN jest stan wysoki) lub odwrotnie (gdy na wejściu IN jest stan niski).

1.5 Rejestry rewersyjne.



Rejestry rewersyjne posiadają możliwość przesuwania danych zarówno w prawo, jak i w lewo. Wyborem kierunku steruje się poprzez parę wejść SR i SL (10: przesuwanie w prawo; 01: w lewo; 00: synchroniczne resetowanie rejestru - po aktywnym zboczu sygnału zegarowego przerzutniki przejdą do stanu 0; 11 jest stanem zabronionym).

Przedstawiony rejestr rewersyjny posiada pojedyncze wejście oraz zdublowane wyjścia (po jednym dla każdego z kierunków przesuwania) - podczas przesuwania w konkretnym kierunku jedno z wyjść należy po prostu pominąć. Można oczywiście skonstruować rejestr z pojedynczym wyjściem bramkując go odpowiednio z sygnałami SR oraz SL, jednak zwiększyłoby to stopień złożoności układu o kilka bramek, których i tak już jest w nim od cholery (wyobraźcie sobie, jak może wyglądać rewersyjny rejestr 8 czy 16 bitowy...).

Można także zrobić odwrotnie – zdublować wejścia (tzn. wyposażyć układ w osobne wejście szeregowe dla przesuwania w prawo oraz osobne dla przesuwania w lewo), a odczyt zorganizować w postaci równoległej. Tak skonstruowany jest np. 4-bitowy rejestr scalony 74194.

2. Liczniki.

Czym jest licznik chyba każdy wie, niemniej jednak definicja książkowa brzmi następująco:
„Jako liczniki określa się układy cyfrowe rejestrujące liczbę impulsów, podanych w określonym przedziale czasu na ich wejście. Są to układy sekwencyjne zawierające pewną liczbę przerzutników synchronicznych, odpowiednio ze sobą połączonych."
Podstawową funkcją licznika jest zatem, jak łatwo się domyślić, liczenie w określonym kodzie. Najczęściej spotykane są liczniki binarne (liczące po kolei w systemie dwójkowym) oraz liczniki BCD (liczące w kodzie dziesiętnym, wykorzystując 4 bity do reprezentacji jednej cyfry kodu dziesiętnego, czyli wartości od 0 do 9) jak i również kilka innych (pierścieniowe, liczące w kodzie Johnsona itp.), ale skonstruować można również liczniki, w których kodem jest dowolny ciąg różnych wartości (mniej lub bardziej bezsensownych).

Ogólnie liczniki można podzielić na grupy według następujących kryteriów:

a) ze względu na kod:
   - binarne
   - BCD
   - inne
b) ze względu na sposób taktowania przerzutników:
   - synchroniczne (gdy wszystkie przerzutniki są taktowane tym samym sygnałem)
   - asynchroniczne (gdy chociaż jeden jest taktowany innym sygnałem niż pozostałe)
c) ze względu na kierunek zliczania:
   - liczące w górę
   - liczące w dół
   - liczące w górę i w dół

Każdy licznik charakteryzują dwie wartości:

- liczba przerzutników **n**
- długość cyklu (pojemność) $\mathbf{m} \leq 2^n$

Generalnie, jeśli licznik ma m różnych stanów, przez które przechodzi cyklicznie, to określa się go jako *licznik modulo m*.

Projektowanie licznika liczącego w odpowiednim kodzie bądź o odpowiedniej długości cyklu zliczania może być wykonana na kilka sposobów. Stosują się następujące metody:

- poprzez składanie z gotowych modułów
- poprzez skracanie
- poprzez syntezę

2.1 Łączenie liczników.

Liczniki można połączyć na dwa sposoby – szeregowo, podłączając wyjścia jednego z liczników do wejścia zegarowego następnego w kolejności, lub równolegle, taktując każdy z liczników wspólnym zegarem i włączając odpowiednie liczniki tylko wtedy, gdy powinny zmienić swój stan.

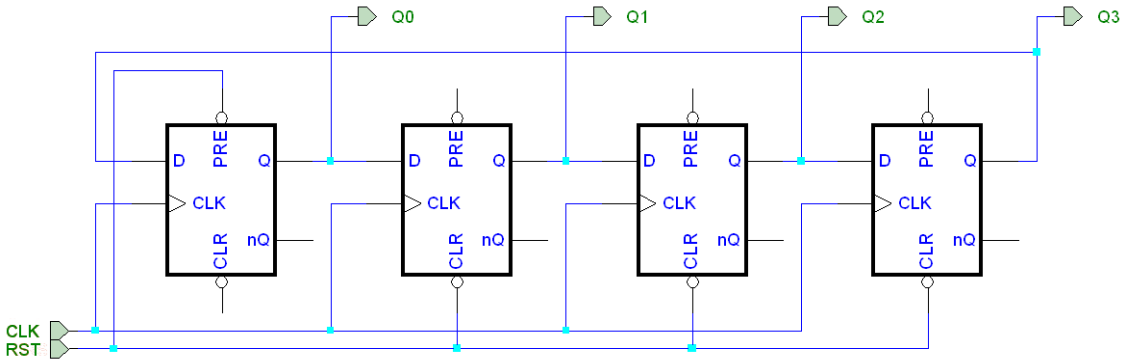
Liczniki scalone, czy to liczące w górę, czy w dół, najczęściej wyposażone są w wyjście przeniesienia CO (Carry Output). Połączenie kilku takich układów w jeden (np. czterech liczników BCD w celu uzyskania licznika mod 10000) polega na połączeniu linii CO poprzedniego licznika z wejściem zegarowym licznika następnego – w momencie wystąpienia przepełnienia pierwszego licznika otrzymujemy pojedynczy takt na liczniku drugim, który w wyniku tego zwiększa swoją wartość o 1. Jeśli na nim również wystąpi przepełnienie, to zwiększona zostanie wartość licznika trzeciego itp.
Zaletą łączenia szeregowego (również w przypadku łączenia pojedynczych przerzutników => synteza liczników asynchronicznych) jest prostota i niska złożoność otrzymywanego w ten sposób układu końcowego, posiada ono jednak dużą wadę – w najgorszym przypadku czas propagacji przez układ jest równy sumie czasów propagacji przez wszystkie liczniki po kolei i powoduje to znaczne obniżenie maksymalnej częstotliwości, z jaką nasz licznik może być taktowany.

Alternatywą jest łączenie równoległe – tzn. podłączenie jednego sygnału zegarowego na wszystkie liczniki oraz włączanie i wyłączanie konkretnych liczników poprzez podanie sygnału na linii CS.
W takim wypadku wyjście RCO (Ripple Carry Output) poprzedniego licznika podłączamy pod wejście CS licznika następnego. Aktywny poziom na linii RCO pojawia się przed taktem zegara powodującym przepełnienie oraz znika chwile po nim, zatem na czas tego taktu licznik następny będzie włączony (aktywne wejście CS) i zwiększy swój stan o 1.
Rozwiązanie takie zapewnia krótszy czas propagacji przez układ - pomijając zsumowane czasy propagacji do wyjść RCO kolejnych liczników (aby w momencie aktywnego zbocza zegara odpowiednie liczniki były włączone), zmiany na wszystkich licznikach wywoływane są jednocześnie, stąd czas propagacji całego układu jest równy czasowi propagacji pojedynczego licznika.

To, jak dane liczniki scalone można łączyć, zależy od samych liczników. Niektóre w ogóle nie posiadają wyjścia przeniesienia – wówczas należy najstarszy bit wyjścia poprzedniego licznika, który zmienia się dwukrotnie podczas całego jego cyklu zliczania, potraktować jako sygnał taktujący licznika następnego i podłączyć pod jego wejście zegarowe.

2.2 Licznik pierścieniowy z autokorekcją.

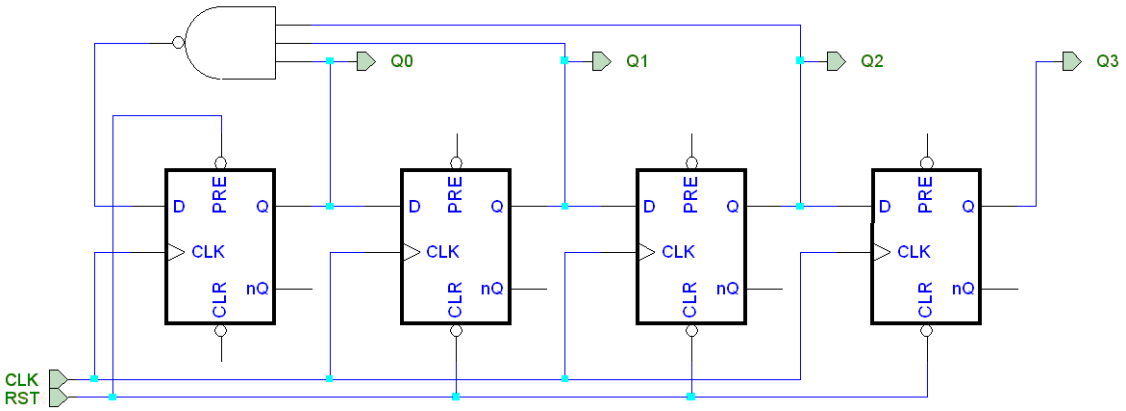
Licznik pierścieniowy jest to taki rejestr przesuwający, w którym wyjście zostało połączone z wejściem oraz w którym krąży tylko jedna jedynka lub tylko jedno zero.



Taki licznik dla n przerzutników posiada n stanów, które przedstawiają się następująco:

Sn	Q_3	Q_2	Q_1	Q_0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0
0	0	0	0	1

Dodatkowo do licznika można dołożyć układ autokorekcji, który sam wróci do poprawnego kodu w przypadku wystąpienia zakłóceń (tzn. np. gdy pojawią się dwie jedynki).
Schemat licznika pierścieniowego z autokorekcją z krążącym pojedynczym zerem wygląda tak:



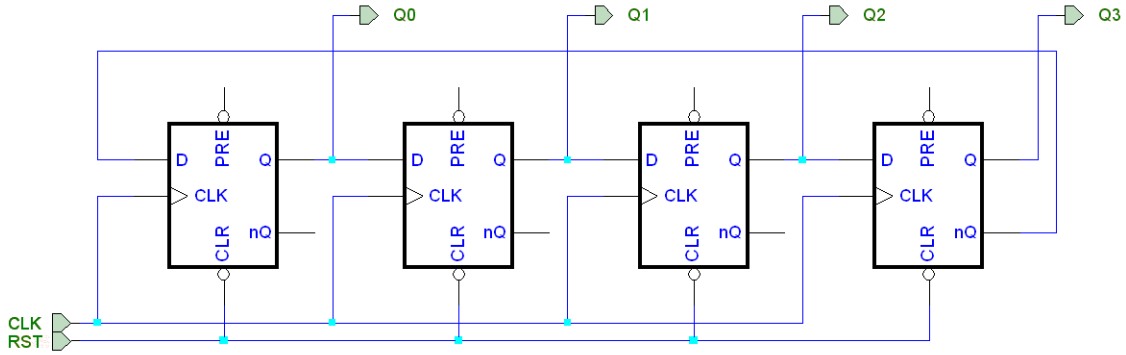
Na wejście pierwszego przerzutnika podawane jest zero tylko wtedy, gdy na pierwszych trzech mamy stan wysoki (czwarty przerzutnik można pominąć, gdyż jego wartość „wyjdzie" z rejestru przy najbliższym takcie zegara). Jeśli w wyniku zakłóceń pojawiłoby się dodatkowe zero (albo nawet same zera), to i tak po 4 cyklach zegara ponownie krążyć będzie tylko jedno. Analogicznie, jeśli na wszystkich przerzutnikach pojawi się w wyniku błędu stan wysoki, to na wejściu pierwszego przerzutnika podane zostanie od razu zero i po takcie zegara zacznie krążyć w liczniku.

2.3 Licznik w kodzie Johnsona.

Kolejne stany licznika 4-bitowego liczącego w kodzie Johnsona przedstawiają się następująco:

Sn	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0
0	0	0	0	0

Jak widać, kod Johnsona polega na wypełnieniu wszystkich bitów po kolei jedynkami, a następnie zerami. Realizacja licznika liczącego w takim kodzie na przerzutnikach D jest zatem bardzo prosta – wystarczy złożyć z nich rejestr przesuwający (szeregowo-równoległy), a zanegowane wyjście ostatniego przerzutnika połączyć z wejściem pierwszego:



Po zresetowaniu licznika, które ustawi wyjścia wszystkich przerzutników na zero, na wejściu pierwszego z nich pojawi się jedynka. Sygnał na tym wejściu zmieni się dopiero, gdy owa jedynka dotrze do ostatniego przerzutnika (wówczas jego zanegowany stan będzie wynosił zero), czyli w praktyce dopiero w momencie, kiedy cały rejestr zostanie wypełniony jedynkami. Kolejna zmiana stanu wejścia pierwszego przerzutnika będzie miała miejsce, gdy pierwsze zero trafi do ostatniego przerzutnika, co z kolei nastąpi dopiero po wypełnieniu całego rejestru zerami.
Licznik będzie liczył zatem w kodzie Johnsona.

Warto zauważyć, iż nie ma tutaj żadnego układu autokorekcji – jeśli w wyniku zakłóceń licznik wejdzie w jakiś stan, którego w kodzie Johnsona nie ma (np. 1010), to jedyną możliwością wymuszenia poprawnego liczenia będzie ręczne zresetowanie licznika.

2.4 Skracanie liczników.

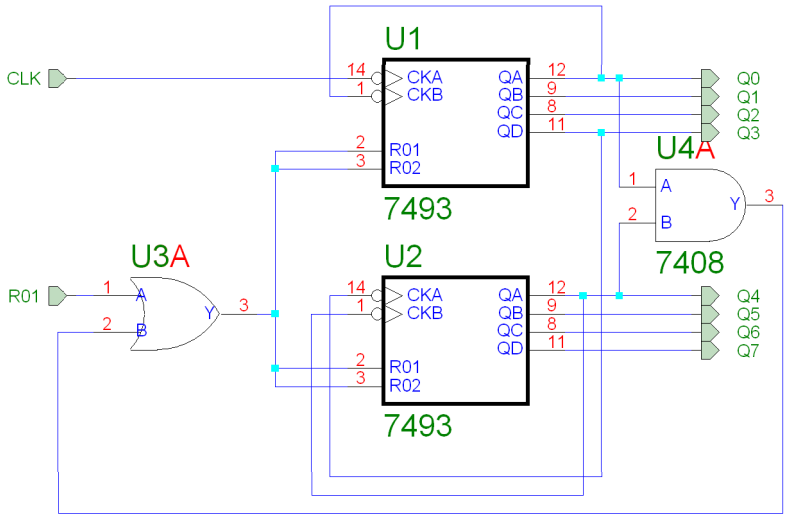
Często, np. przy projektowaniu układów sterowania, potrzebny jest nam licznik o odpowiednio długim cyklu zliczania (albo po prostu dzielnik częstotliwości modulo n). I jakoś tak się dzieje, że nie można kupić scalonego licznika np. modulo 666, ani takich, z których można by takowy licznik wprost złożyć. Ale są metody, żeby temu zaradzić.

Najprostszą jest technika zwana szumnie skracaniem liczników. Polega ona na tym, iż najpierw łączymy kilka liczników scalonych, aby uzyskać wystarczająco duży zakres wartości, a następnie za pomocą odpowiednich bramek powodujemy autoresetowanie się zmontowanego przez nas licznika, gdy znajdzie się on w konkretnym stanie (tzn. gdy sygnały na wyjściach mają konkretne wartości), czyli skracamy cykl zliczania licznika do potrzebnej wartości (stąd nazwa).
Liczniki rozpoczynają liczenie od 0, zatem w celu uzyskania licznika *modulo m* należy tak dobrać i połączyć bramki, aby resetował się on po osiągnięciu stanu *m* (innymi słowy, aby wykrywany był stan *m* na wyjściach i reagował poprzez zaktywowanie linii CLR). Efekt będzie taki, że po zmianie licznika ze stanu *m-1* na *m* (i czasie propagacji przez bramki oraz czasie propagacji sygnału CLR do wyjść) nastąpi natychmiastowe przejście do stanu 0. Kolejne aktywne zbocze zegara spowoduje już przejście do stanu 1, dlatego właśnie trzeba wykrywać stan *m*, a nie *m-1*; stany *m* oraz 0 niejako dzielą się między sobą jednym okresem zegara (w praktyce stan *m* zaobserwujemy na wyjściach bardzo krótko).
Rozważania te dotyczą tych liczników, które posiadają asynchroniczny CLR (tzn. ustawienie zer na wyjściach następuje po czasie propagacji z wejścia resetującego do wyjść), czyli np. układów 7490, 7493, 74160 czy 74192.

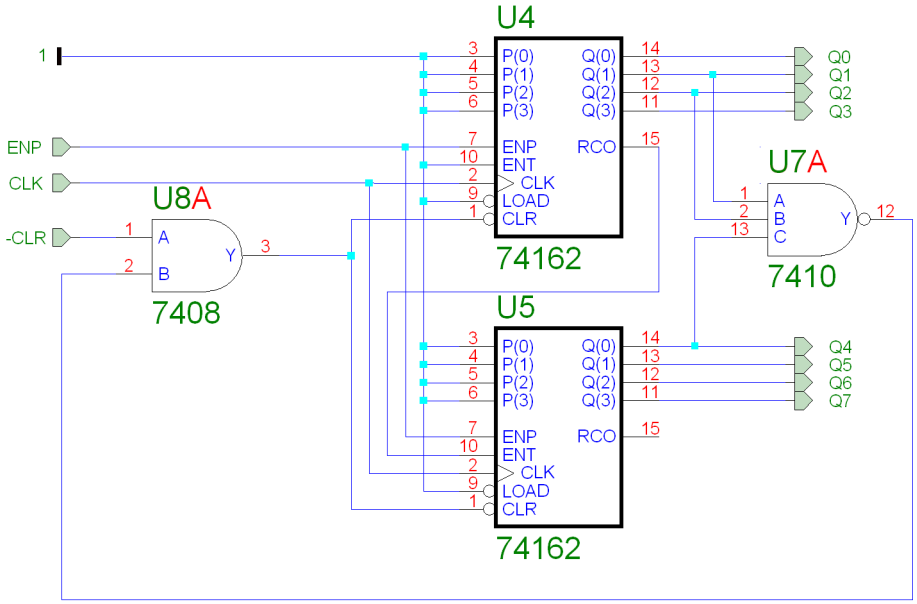
W przypadku liczników wyposażonych w synchroniczny CLR sprawa ma się nieco inaczej. Synchroniczność tego sygnału oznacza, iż zera na wyjściach pojawiają się po aktywnym zboczu zegara, jeśli linia CLR jest aktywna, czyli w celu zresetowania licznika musimy aktywować CLR i poczekać na takt zegara. Wówczas zamiast kolejnego stanu na liczniku pojawią się same zera.
Jeśli wykrywalibyśmy stan *m*, to stan 0 pojawiałby się zamiast stanu *m+1* (kolejnego po *m*), czyli nasz licznik liczyłby *modulo m+1*. Zatem przy synchronicznym CLR wykrywamy stan *m-1*, gdyż nie następuje tutaj dzielenie taktu zegara pomiędzy stan 0 oraz stan, który wywołuje resetowanie – ten drugi będzie trwał aż do aktywnego zbocza zegara. W synchroniczny CLR wyposażone są np. układy 74162 i 74163.

Oczywiście zamiast resetować licznik wejściem CLR możemy skorzystać z wejścia LOAD oraz wejść programujących (o ile licznik scalony takowe posiada) i spowodować przeskok do stanu innego niż 0. Można w ten sposób zrobić licznik, który rozpoczyna liczenie od dowolnego stanu i wraca do niego za każdym razem, gdy osiągnie określony przez nas stan końcowy.

Przykładowy licznik modulo 17 zrealizowany na układach scalonych 7493 (4-bitowy licznik binarny z asynchronicznym sygnałem CLR) – wykrywamy wartość 17 na wyjściach:



I ten sam licznik modulo 17 zrealizowany na układach scalonych 74162 (4-bitowy licznik BCD z synchronicznym sygnałem CLR) – wykrywamy wartość 16:



2.4 Synteza liczników synchronicznych.

Wyższą szkołą jazdy jest zaprojektowanie potrzebnego nam licznika metodą syntezy. Wymaga to wykonania kilku czynności, które najlepiej będzie przedstawić na przykładzie.

Zadanie: zaprojektować synchroniczny licznik 6-stanowy liczący w kodzie 1-3-5-0-2-4 metodą syntezy.

1. Krok pierwszy – zapisanie tablicy przejść.

Aby złożyć licznik na przerzutnikach, czy to D, czy JK, musimy dokładnie wiedzieć, co podać na wejście (w przypadku JK – wejścia) każdego z nich. Projektowanie rozpoczynamy zatem od zapisania tablicy przejść, w której umieszczamy stan każdego z przerzutników:

s_n	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

s_{n+1}	Q_2	Q_1	Q_0
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
1	0	0	1
0	0	0	0

Wartości w stanie n warto zapisywać w normalnej kolejności rosnącej, a nie w kolejności ich występowania w kodzie licznika – znacznie ułatwi to później wpisywanie danych do siatek Karnaugh.

W wierszach powyższej tabeli mamy zatem kolejne wartości licznika (s_n, po lewej) i stany następujące po nich (s_{n+1}, po prawej) rozpisane w postaci binarnej. Dowiadujemy się w ten sposób, jaki stan wejść ma spowodować konkretny stan wyjść:

- stan 000 (wszystkie przerzutniki w stanie niskim) ma wymusić stan 010 (czyli przejście przerzutnika 1 do stanu wysokiego)
- stan 001 ma wymusić stan 011
- stan 010 ma wymusić stan 100 itp.

W oparciu o tę tablicę możemy teraz stworzyć siatki Karnaugh i wyznaczyć na ich podstawie funkcję na wejściach przerzutników.

2. Krok drugi – siatki Karnaugh.

Każde wejście może być określone przez inną funkcję, zatem dla każdego z nich trzeba przygotować osobną siatkę Karnaugh. W przypadku przerzutników D będzie to oznaczało po jednej siatce na przerzutnik, czyli w naszym liczniku trzy. Przy syntezie licznika na przerzutnikach JK siatek tych potrzeba po dwie na przerzutnik (osobna dla wejścia J i osobna dla wejścia K), czyli dla naszego licznika sześć.

Syntezę naszego licznika przeprowadzimy na przerzutnikach D.

Najpierw szkicujemy puste siatki, które będziemy następnie uzupełniać:

D_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

D_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

D_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

Po przeliczeniu na kod dziesiętny adresy komórek są następujące:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	*0*	*1*	*3*	*2*
1	*4*	*5*	*7*	*6*

Uzupełnianie rozpoczniemy od siatki dla przerzutnika D_0.

Przerzutnik D przepisuje wejście na wyjście podczas aktywnego zbocza zegara (jego stan następny nie zależy od poprzedniego), zatem aby uzyskać na wyjściu 0 musimy podać na wejście stan niski, a aby uzyskać 1 – podać stan wysoki.

Wracamy zatem do naszej tablicy przejść i zwracamy uwagę tylko na prawą kolumnę dotyczącą przerzutnika D_0 oraz wartość licznika w każdym wierszu:

s_n	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

s_{n+1}	Q_2	Q_1	Q_0
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
1	0	0	1
0	0	0	0

Stan s_0 (czyli $Q_2Q_1Q_0 = 000$) ma spowodować przejście licznika do stanu s_2 ($Q_2Q_1Q_0 = 010$). Analizowany przez nas w chwili obecnej przerzutnik D_0 ma w stanie s_2 na wyjściu 0, stąd w stanie s_0 należy podać mu na wejście również 0 (wówczas po takcie zegara wartość ta zostanie przepisana na wyjście). Wpisujemy zatem do siatki Karnaugh w komórce 000 (lewy-górny róg) zero.

Analogicznie analizujemy drugi wiersz – w stanie s_1 (001) należy podać na wejście przerzutnika 1, gdyż taką wartość chcemy na nim mieć w stanie następnym (czyli po takcie zegara). Do komórki o adresie 001 wpisujemy zatem jedynkę.

Powoli zaczyna być widać, dlaczego warto było zapisać stany w kolejności rosnącej. Posługując się przedstawioną powyżej tablicą z wpisanymi wartościami dziesiętnymi adresów poszczególnych komórek nie musimy wcale szukać w tablicy komórki, do której należy wpisać otrzymaną wartość. Przy losowej kolejności wartości musielibyśmy każdorazowo odczytać stan licznika i zastanowić się, która komórka siatki posiada właśnie taki adres, co może znacznie wydłużyć proces projektowania układu.

Co więcej, zauważyć można, że tak naprawdę przepisujemy wartości z jednej podświetlonej kolumny (Q_0) do komórek o adresach z drugiej (s_n). Skoro adresy te mamy podane kolejno (w kolejnych wierszach), to wypełnienie siatki staje się banalnie proste i zajmuje niewiele czasu. Wystarczy po kolei przepisać analizowaną kolumnę stanów na wyjściu przerzutnika do siatki określającej jego wejście, zachowując przy tym wspomnianą kolejność rosnących adresów komórek w siatce.
Gotowa siatka wygląda tak:

D_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	1	1	0
1	1	0	X	X

gdzie X oznacza stan dowolny (wartości binarne 110 i 111 nie występują w kodzie licznika).

Analogicznie postępujemy dla przerzutnika D_1 oraz D_2, analizując odpowiednio drugą (Q_1) i trzecią (Q_2) kolumnę od prawej. Uzyskamy wówczas następujące siatki:

D_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	1	1	0	0
1	0	0	X	X

D_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	0	1	1
1	0	0	X	X

Następnym krokiem do wykonania jest minimalizacja i uzyskanie funkcji.

3. Krok trzeci – minimalizacja.

Przeprowadzamy standardową minimalizację uzyskanych siatek:

D_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	1	1	0
1	1	0	X	X

D_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	1	1	0	0
1	0	0	X	X

D_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	0	1	1
1	0	0	X	X

Uzyskane funkcje są zatem następujące:

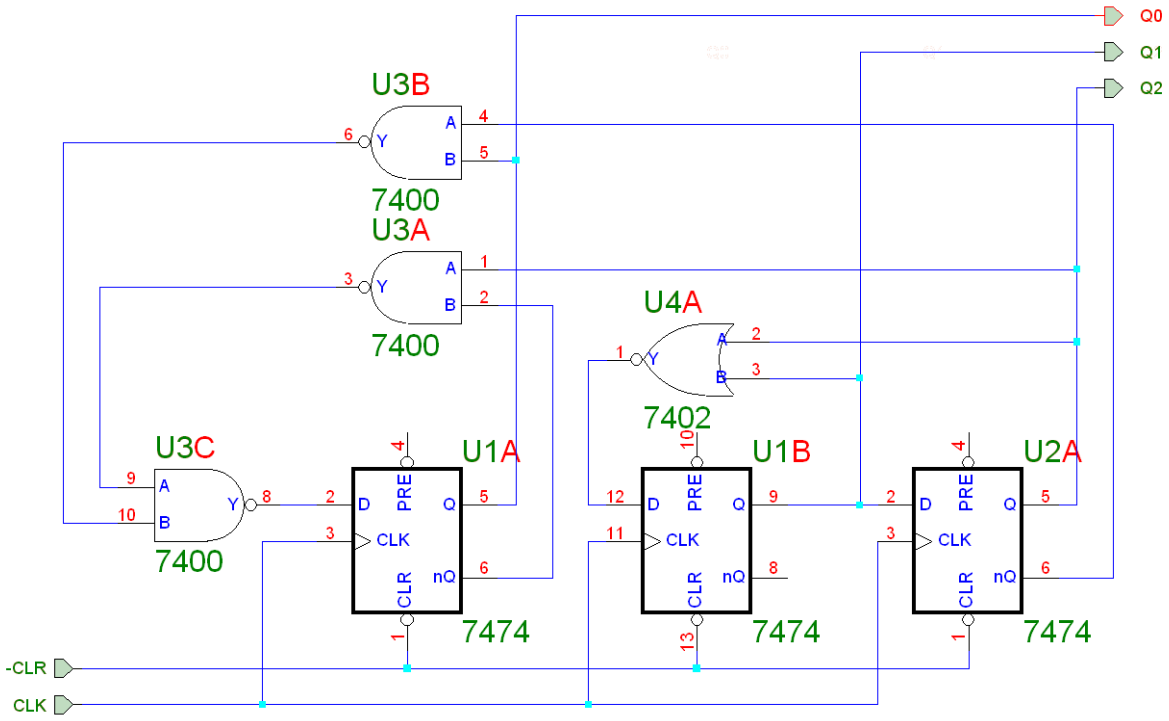
$$D_0 = \overline{Q_2}Q_0 + Q_2\overline{Q_0} = \overline{\overline{Q_2}Q_0 + Q_2\overline{Q_0}} = \overline{\overline{Q_2}Q_0}\,\overline{Q_2\overline{Q_0}}$$

$$D_1 = \overline{Q_2}\,\overline{Q_1} = \overline{\overline{Q_2}\,\overline{Q_1}} = \overline{Q_2 + Q_1}$$

$$D_2 = Q_1$$

4. Krok czwarty – realizacja funkcji na bramkach i schemat układu.

Uzyskane funkcje, po odpowiednim przekształceniu, można zrealizować na trzech bramkach NAND i jednej bramce NOR. Schemat naszego licznika wygląda tak:



Co kończy projektowanie licznika.

Może się jednak zdarzyć, że zadanie będzie precyzowało typ zastosowanych przerzutników.

Zadanie: zaprojektować synchroniczny licznik 6-stanowy liczący w kodzie 1-3-5-0-2-4 metodą syntezy oparty na przerzutnikach JK.

Synteza liczników synchronicznych na przerzutnikach JK przebiega bardzo podobnie do opisanej powyżej syntezy liczników na przerzutnikach D. Znaczącą różnicą jest fakt, iż każdy przerzutnik JK posiada dwa wejścia – J i K – zatem należy sporządzić dwa razy więcej siatek Karnaugh.

1. Krok pierwszy – zapisanie tablicy przejść.

Punkt jest identyczny jak dla przerzutników D.

s_n	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

s_{n+1}	Q_2	Q_1	Q_0
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
1	0	0	1
0	0	0	0

2. Krok drugi – siatki Karnaugh.

Jak już wspomniałem, na każdy przerzutnik JK potrzebujemy dwie siatki – osobną dla wejścia J oraz osobną dla wejścia K – co w przypadku naszego licznika oznacza przygotowanie w sumie 6 siatek. Naszkicowane siatki przed uzupełnieniem wyglądają tak:

J_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

K_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

J_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

K_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

J_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

K_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0				
1				

Uzupełniać je będziemy parami, tzn. równolegle zapisując wartości dla wejść J i K analizowanego przerzutnika.

W tym miejscu przyda się kilka słów przypomnienia działania przerzutnika JK. W odróżnieniu do przerzutnika D, stan na wyjściu JK nie zależy tylko od wejść, ale również od stanu poprzedniego. Tablica wzbudzeń przedstawia się następująco:

$Q_n \rightarrow Q_{n+1}$	J	K
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	X
$1 \rightarrow 0$	X	1
$1 \rightarrow 1$	X	0

Jak widać, przy każdym przejściu wymagany jest określony stan tylko na jednym z wejść (drugie może mieć wtedy dowolną wartość). Wartości wpisanych do pary siatek będzie zatem tyle, ile w pojedynczej siatce dla przerzutnika D, jednak rozłożymy je na dwie siatki, wypełniając puste pola stanami dowolnymi. Jaki z tego pożytek? Okaże się przy minimalizacji. Do wykonania tego konkretnego zadania nie będzie nam potrzebna ani jedna bramka.

Wracamy zatem do tablicy. Na początek weźmiemy przerzutnik JK_0 i o ile w przypadku D interesowały nas tylko dwie kolumny (adres i wartość), tak tym razem kolumn tych będzie trzy – dodatkowo potrzebny będzie nam stan badanego przerzutnika przed zmianą:

s_n	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

s_{n+1}	Q_2	Q_1	Q_0
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
1	0	0	1
0	0	0	0

Adres komórki, do której mamy wpisać wartość, wyznaczamy analogicznie (numer stanu przed zmianą) jak w przerzutnikach D. Różnić będzie się wyznaczenie samej wartości, gdyż w tym celu musimy porównać sygnał na wyjściu przerzutnika w stanie s_n oraz s_{n+1}.

W pierwszym wierszu jest to przejście $0\rightarrow0$, co oznacza, że wejście J_0 musi mieć stan niski, z kolei K_0 może mieć dowolny stan. Wpisujemy zatem zero w komórce o adresie 000 w siatce J_0 oraz X w komórce o tym samym adresie w siatce K_0.

W drugim wierszu widzimy przejście $1\rightarrow1$. Wpisujemy X w komórce o adresie 001 w siatce J_0 oraz zero w komórce w siatce K_0.

Analogicznie postępujemy dla pozostałych wierszy, a pozostałe na końcu puste pola wypełniamy stanami dowolnymi X, uzyskując w efekcie uzupełnioną parę siatek:

J_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	X	X	0
1	1	X	X	X

K_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	0	0	X
1	X	1	X	X

Tak samo postępujemy w przypadku dwóch pozostałych przerzutników (analizując odpowiadające im kolumny) i otrzymujemy siatki postaci:

J_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	1	1	X	X
1	0	0	X	X

K_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	X	1	1
1	X	X	X	X

J_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	0	1	1
1	X	X	X	X

K_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	X	X	X
1	1	1	X	X

Pozostałe kroki są identyczne jak dla licznika na przerzutnikach D.

3. Krok trzeci – minimalizacja.

J_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	X	X	0
1	1	X	X	X

K_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	0	0	X
1	X	1	X	X

J_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	1	1	X	X
1	0	0	X	X

K_1:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	X	1	1
1	X	X	X	X

J_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	0	1	1
1	X	X	X	X

K_2:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	X	X	X	X
1	1	1	X	X

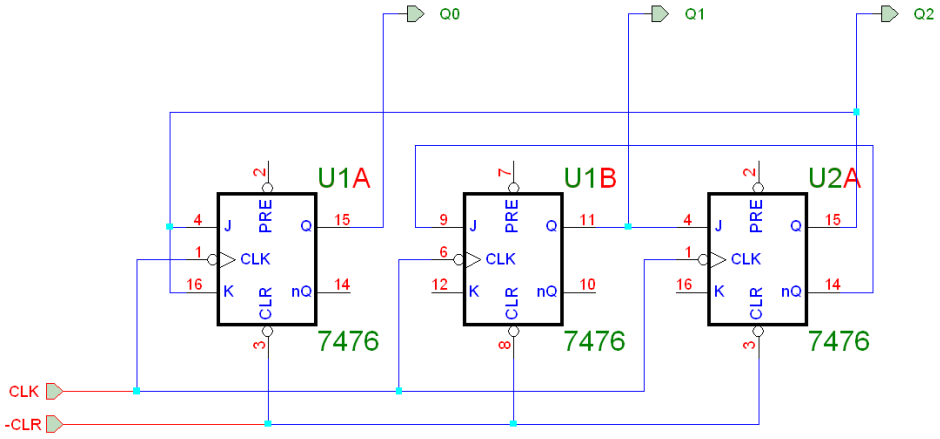
Uzyskane funkcje:

$J_0 = Q_2$ $\qquad$ $K_0 = Q_2$

$J_1 = \overline{Q_2}$ $\qquad$ $K_1 = 1$

$J_2 = Q_1$ $\qquad$ $K_2 = 1$

4. Krok czwarty – realizacja i schemat układu.



Zgodnie z obietnicą ani jednej bramki.

Widać zatem, że o ile synteza liczników synchronicznych opartych o przerzutniki JK jest bardziej czasochłonna, o tyle złożoność uzyskanego w wyniku niej licznika jest nieporównywalnie mniejsza.

3. Multipleksery i demultipleksery.

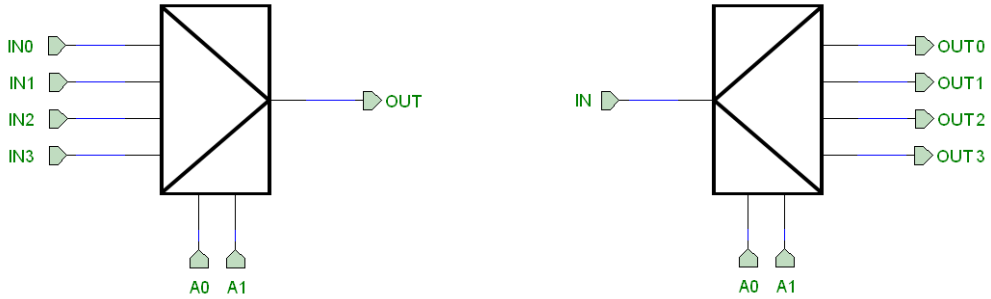
Układy zwane multiplekserami i demultiplekserami pełnią funkcję selektorów sygnału. Wyposażone są w dwa rodzaje wejść:

- informacyjne (danych)
- sterujące (adresowe)

Działanie multipleksera sprowadza się do przepisania stanu (zera bądź jedynki) z konkretnego wejścia informacyjnego wybranego przy pomocy wejść sterujących na wyjście układu. Multiplekser służy zatem do wyboru źródła sygnału.

Demultiplekser działa odwrotnie – posiada jedno wejście informacyjne, z którego sygnał podawany jest na wybrane (za pośrednictwem wejść sterujących) wyjście. Pozwala w ten sposób wybrać cel sygnału, czyli dokąd ma on polecieć.

Symbole multipleksera oraz demultipleksera 4-wejściowego wyglądają następująco:



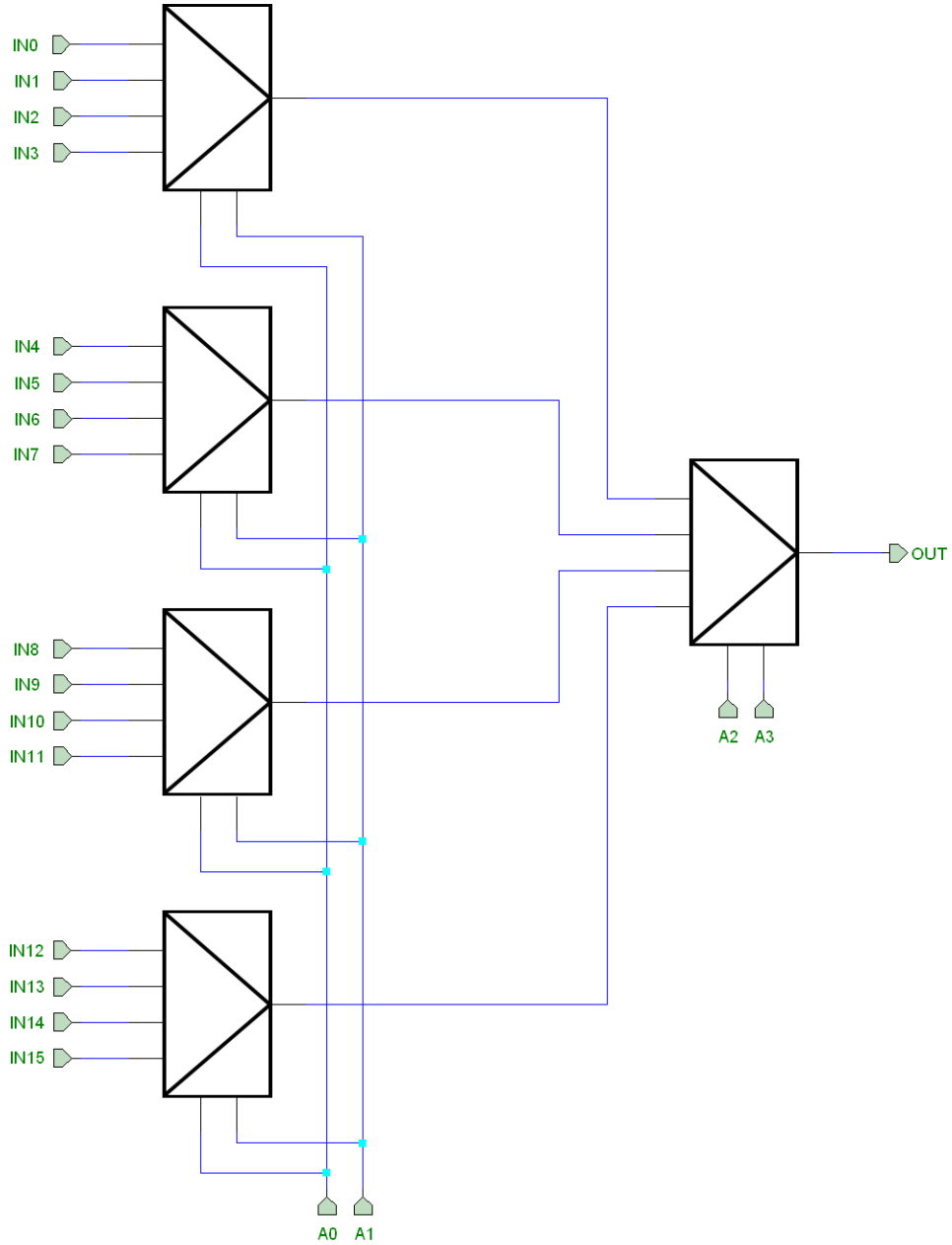
Na raz wybrane w multiplekserze jest tylko jedno wejście. Stan na pozostałych wejściach (niewybranych w danej chwili) nie ma wówczas żadnego znaczenia na stan wyjścia.

Adres demultipleksera wskazuje na jedno wyjście, na które przepisywany jest sygnał wejściowy, a stan na pozostałych zależy wtedy od wewnętrznej konstrukcji układu. Najczęściej jest to stan niski, czyli zero, jednak konstruuje się również demultipleksery o zanegowanych wejściach i wyjściach. Sygnał przepisywany na wybrane wyjście jest wówczas bez zmian (zgodnie z prawem podwójnej negacji), ale na pozostałych wyjściach otrzymujemy stan wysoki. Ma to kluczowe znaczenie przy stosowaniu demultiplekserów np. do rozszerzania przestrzeni adresowej pamięci, gdyż linie CS często są w tych układach aktywne niskim poziomem.

3.1 Łączenie multiplekserów.

Może się zdarzyć, że mamy do dyspozycji np. multipleksery 4-bitowe, a do efektywnej realizacji funkcji potrzebujemy powiedzmy 16 wejść. Innymi słowy, mamy dane 16 źródeł sygnału i musimy mieć możliwość zaadresowania (wyboru) każdego z nich.

W tym celu łączymy multipleksery w następujący sposób:



Zasada działania układu jest prosta. Do zaadresowania 16 wejść potrzebujemy 4 wejścia adresowe (2^4=16), stąd dwa mniej znaczące bity adresu przeznaczamy na pierwszy poziom multiplekserów (cztery układy po lewej stronie), a dwa bardziej znaczące na drugi (pojedynczy układ po prawej).

Bity A_0 i A_1 definiują wówczas, które z wejść multiplekserów pierwszego poziomu są przekazywane na ich wyjścia i podawane jako wejścia multipleksera drugiego poziomu. Bity A_2 i A_3 stwierdzają, które z tych wejść jest następnie przepisane na wyjście całego układu.

Przykład – załóżmy, że na wejścia adresowe podano wartość 0011.

- bity A_1A_0 mają wartości 11, co oznacza, że każdy z multiplekserów pierwszego poziomu przepisze na wyjście ostatnie ze swoich wejść. Zatem na wejściach multipleksera drugiego poziomu pojawią się sygnały IN_3, IN_7, IN_{11} oraz IN_{15}.
- bity A_3A_2 mają wartości 00, czyli zaadresowane jest zerowe wejście multipleksera drugiego poziomu. Na wejściu tym podano sygnał IN_3 i to właśnie on zostaje przepisany na wyjście.

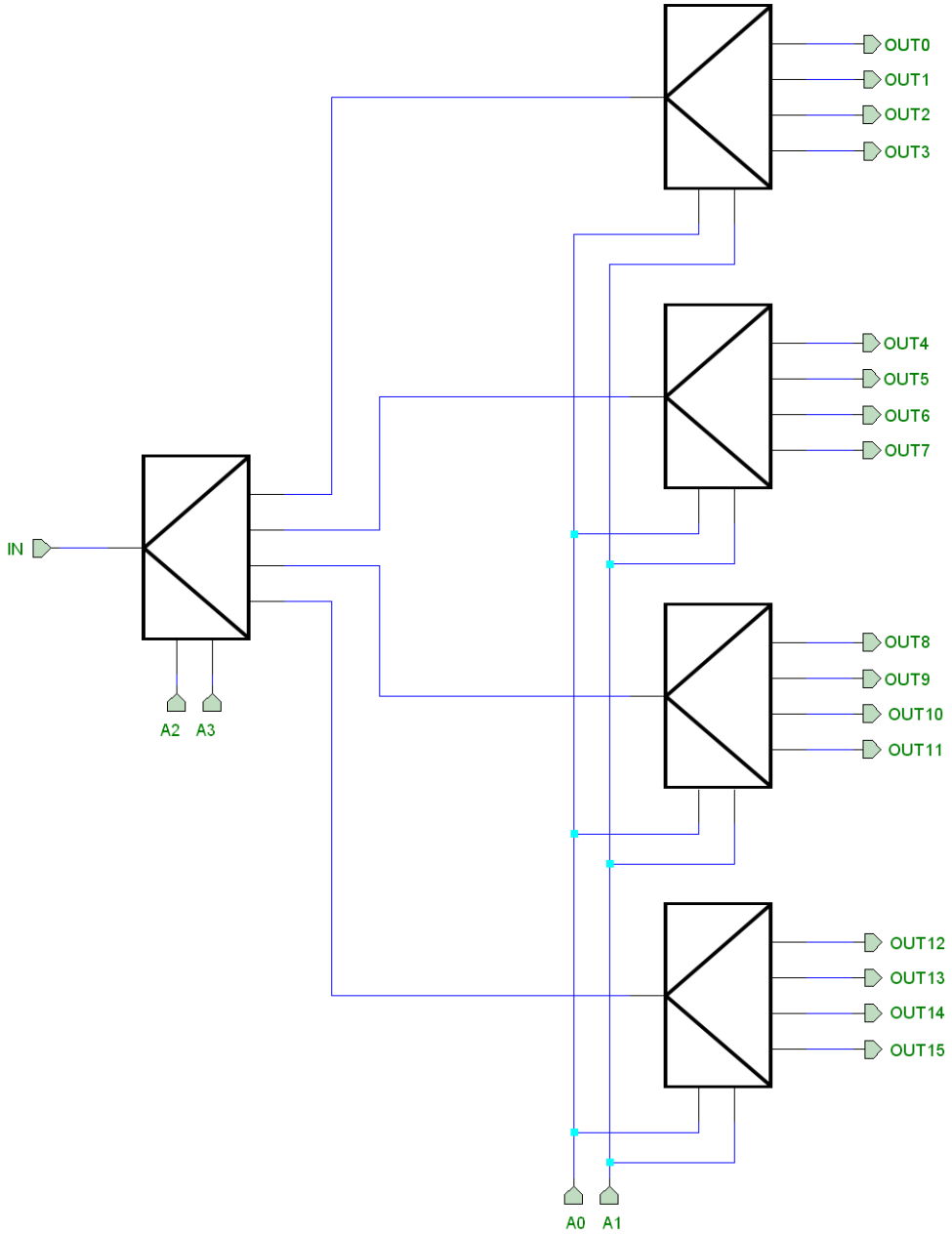
Wejścia multiplekserów numerowane są od zera i na wyjściu pojawiło się sygnał z wejścia nr trzy, co zgadza się z adresem - $(0011)_2 = (3)_{10}$.

Jeśli na wejścia adresowe podamy adres 0111, to na wyjściu otrzymamy sygnał IN_7. Adres 1011 da na wyjściu sygnał IN_{11}, adres 1101 sygnał IN_{13}, adres 0101 sygnał IN_5 itd.

3.2 Łączenie demultiplekserów.

Analogicznie wygląda sprawa, gdy posiadając np. demultipleksery 4-wyjściowe chcemy móc wybrać jedną z 16 docelowych linii sygnałowych.

Schemat połączonych demultiplekserów wygląda tak:



Występuje jeden demultiplekser pierwszego poziomu (adresowany bardziej znaczącymi bitami adresu) oraz cztery demultipleksery drugiego poziomu (adresowane mniej znaczącymi bitami). Adresowanie wyjść jest zatem analogiczne do adresowania wejść multiplekserów - dwa starsze bity określają, na który z demultiplekserów drugiego poziomu zostanie przekazany sygnał z wejścia układu, a młodsze bity stwierdzają, na które ze swoich wyjść następnie przepisze ten sygnał wskazany demultiplekser.

Na wszystkich pozostałych wyjściach układu pojawi się stan niski.

3.3 Realizacja funkcji na multiplekserach.

W siatkach Karnaugh, podczas syntezy liczników, potraktowaliśmy wartości zmiennych $Q_2Q_1Q_0$ jako adres komórki, w której znajdowała się jakaś określona wartość.

Wejścia adresowe w multiplekserze pozwalają na przepisanie sygnału z konkretnego wejścia na wyjście. Nietrudno zatem zauważyć, że podając zera na wejścia o adresach odpowiadających komórkom siatki zawierającym zero oraz jedynki na pozostałe otrzymamy gotową realizację szukanej funkcji. Przykładowo, siatkę dla wejścia przerzutnika D_0 syntezowanego przez nas licznika:

D_0:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	1	1	0
1	1	0	X	X

można zrealizować na 8 wejściowym multiplekserze podając odpowiednio na wejścia danych:

$IN_0=0$

$IN_1=1$

$IN_2=0$

$IN_3=1$

$IN_4=1$

$IN_5=0$

oraz na wejścia adresowe:

$A_0=Q_0$

$A_1=Q_1$

$A_2=Q_2$

Zamiast wykorzystywać 3 bramki wystarczy skorzystać z pojedynczego 8-wejściowego multipleksera.

Ponadto, funkcję tę można również zrealizować na układzie o mniejszej ilości wejść. W tym celu należy wyeliminować jedną zmienną z siatki – w naszym przypadku niech to będzie Q_1.

Eliminowanie zmiennych sprowadza się do łączenia pól siatki w odpowiedni sposób. Jeśli zaznaczymy różnymi kolorami pola, których adres różni się tylko wartością zmiennej Q_1, otrzymamy:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	1	1	0
1	1	0	X	X

Pola te należy połączyć parami w całość, otrzymując w efekcie siatkę 2x2:

$Q_2 \setminus Q_0$	0	1
0	0	1
1	1	0

Do realizacji takiej funkcji wystarczy nam multiplekser 4-wejściowy:

$IN_0=0$ $A_0=Q_0$

$IN_1=1$ $A_1=Q_2$

$IN_2=1$

$IN_3=0$

Przykład ten był dość trywialny, gdyż w obu komórkach każdej pary występowała ta sama wartość (no dobra, w przypadku par zielonej i czerwonej w jednej z komórek występował stan dowolny, co jednak nie komplikowało w żaden sposób zadania).

Załóżmy jednak, że nasza siatka wyglądałaby tak:

$Q_2 \setminus Q_1 Q_0$	00	01	11	10
0	0	1	1	0
1	X	0	1	X

Wówczas dla pary niebieskiej i szarej sprawa nadal jest prosta. Dla pary zielonej również – komórka o takim adresie w nowej siatce może mieć stan dowolny. Problem pojawia się jednak przy parze czerwonej, gdyż w obu komórkach mamy różną wartość. Należy tu przywołać nasze kryterium kolorowania komórek – jednym kolorem zaznaczaliśmy te komórki, których adresy różniły się tylko wartością eliminowanej zmiennej. Czerwona komórka zawierająca wartość zero ma adres 001, czyli zmienna Q_1 posiada wtedy również wartość zero. Druga komórka ma adres 011, zmienna Q_1 posiada wartość 1, co odpowiada jedynce znajdującej się w tej komórce. Do czerwonej komórki w docelowej siatce wpisujemy zatem po prostu Q_1:

$Q_2 \setminus Q_0$	0	1
0	0	1
1	X	Q_1

Realizacja na multiplekserze 4-wejściowym przedstawia się następująco:

$IN_0=0$ $A_0=Q_0$

$IN_1=1$ $A_1=Q_2$

$IN_3=Q_1$

Widzimy zatem, że realizując funkcję na multiplekserach możemy do siatki wpisywać nie tylko wartości 0 lub 1 (i ewentualnie stan dowolny X), ale także wartości zmiennych, a nawet całych wyrażeń i innych funkcji – trzeba potem po prostu podać taki sygnał na odpowiednie wejście multipleksera.