

### 3. Układy scalone pamięciowe

W skład msc często mogą wchodzić układy pamięciowe. Szczególnie, gdy użyty mk nie posiada wewnętrznej pamięci danych i programu. Generalnie pamięci można podzielić na dwie grupy:

- **pamięci ulotne**, trzymają dane tylko wtedy, gdy są zasilane. Zanik napięcia zasilania powoduje utratę przechowywanych informacji,
- **pamięci nieulotne**, zachowane dane nie ulegają skasowaniu po wyłączeniu napięcia zasilania.

Pierwsza grupa to pamięci RAM (*Random Access Memory*). Są one pamięciami, w których dane mogą być zarówno zapisywane, jak z niej odczytywane. Istnieją dwa główne typy tych pamięci:

- pamięć **statyczna SRAM**,
- pamięć **dynamiczna DRAM**.

Pamięć SRAM przechowuje bity informacji w postaci stanów przerzutników bistabilnych. Każda komórka pamięci składa się z około od 4 do 6 tranzystorów. Układy SRAM mogą mieć względnie niski pobór mocy w trybie podtrzymywania zawartości pamięci, natomiast w trybie pracy pobierają dość duży prąd. Zaletą tych pamięci jest prostota sterowania oraz szybkość (pamięci te wykonane z arsenku galu GaAs mają czas dostępu poniżej 1ns). Dzięki prostemu sterowaniu są one najczęściej stosowane w msc.

W pamięci DRAM informacja przechowywana jest w postaci ładunków zgromadzonych w kondensatorach, jakie powstają pomiędzy emiterami tranzystorów MOS i podłożem układu. Dane w tych pamięciach mają tendencje do zanikania i wymagają ciągłego odnawiania. Proces taki nazywa się „**odświeżaniem**” i wymaga specjalnych układów do wytwarzania sygnałów odświeżających. Zaletą układów DRAM jest niski koszt produkcji, ze względu na niewielką powierzchnię zajmowanej powierzchni krzemu. Każda komórka pamięci DRAM ma tylko jeden tranzystor MOS. Z zalety tej wynika również inna cecha: pamięci DRAM posiadają znacznie większą pojemność od pamięci SRAM (obecnie dostępne są na rynku pamięci SRAM do 512kB). Wadą tych układów jest skomplikowane sterowanie, wynikające z multipleksowania adresów rzędów i kolumn (pamięci te adresuje się jak macierz) oraz wynikającego z tego dłuższego czasu dostępu, który jest dodatkowo zwiększany o czas potrzebny na odświeżanie.

Istnieje cały szereg pamięci nieulotnych (część z nich zestawiono w tabeli 2.1). Poniżej przedstawiono najczęściej spotykane typy tych pamięci:

- **Pamięć ROM** (*Read Only Memory*) służy tylko do odczytywania. Informacja w pamięci ROM jest zapisywana w czasie produkcji w formie maski, na podstawie dostarczonego wzorca. Dla zamawiającego taką pamięć oznacza to wysoki koszt jednostkowy. Dlatego pamięci ROM używa się jedynie w urządzeniach, które są produkowane w długich seriach.
- **Pamięć EPROM** jest pamięcią programowaną elektrycznie. Użytkownik sam programuje swój wzór informacji, jaką chce zapisać. Gdy układ posiada okienko ze szkła kwarcowego, można taką pamięć kasować i programować ponownie. Jest to duża zaleta w fazie uruchamiania msc. Kasowanie odbywa się na ogół przy pomocy światła ultrafioletowego. Pamięć EPROM umieszczona w obudowie plastikowej, znacznie tańszej od obudowy ceramicznej z okienkiem szklanym, nosi nazwę **OTP** (*One Time Programmable*). Komórki pamięci EPROM są tranzystorami MOS z pływającymi bramkami. Są one programowane przez wstrzykiwanie elektronów o wysokiej energii. Komórka taka zawiera „0”, podczas gdy nie zaprogramowane komórki zawierają „1”.

Zaprogramowanego w ten sposób tranzystora nie można wprowadzić w stan przewodzenia i usunąć zgromadzonego w nim ładunku. Można tego dokonać jedynie przez kasowanie światłem UV. Promienie UV dostarczają elektronom energii, wystarczającej do ponownego przejścia przez barierę energetyczną w warstwie izolacyjnej dwutlenku krzemu wokół pływającej bramki. Układy EPROM mogą być kasowane i ponownie programowane około 100 razy.

- **Pamięci EEPROM** są układami, które nie tylko są elektrycznie programowalne, ale również elektrycznie kasowane. Mają ograniczoną liczbę cykli programowania od około 10000 razy do 1000000 razy. Komórka pamięci EEPROM składa się z tranzystora NMOS (*Metal Nitride Oxide Semiconductor*). Do każdej komórki można wprowadzić ładunek elektryczny, który przechowywany jest w izolowanej warstwie azotku krzemu, znajdującej się między bramką tranzystora, a podłożem z tlenku krzemu. Ładowanie inicjuje się poprzez tzw. efekt tunelowy. Powstaje on po przyłożeniu wysokiego napięcia (20-40V) do bramki tranzystora. Przez zmianę polaryzacji tego napięcia, układ można programować, bądź też kasować.
- **Pamięć błyskowa (FLASH)** jest również pamięcią programowaną i kasowaną elektrycznie. Koszt jej produkcji jest niższy niż układu EEPROM. W porównaniu z pamięcią EEPROM cechuje się ona szybszymi czasami programowania i kasowania. Najczęściej pamięć ta składa się z sektorów, na których mogą być wykonywane odpowiednie operacje, posiada własną listę komend automatyzujących programowanie i kasowanie pamięci.

Poniżej zostaną omówione trzy przykładowe układy pamięciowe, które można spotkać w mse: najprostszy z możliwych układów – 8-bitowy rejestr zatraskujący 74HC574, pamięć SRAM 128kB i pamięć FLASH 128kB.

### 3.1. 8-bitowy rejestr zatraskujący 74HC574

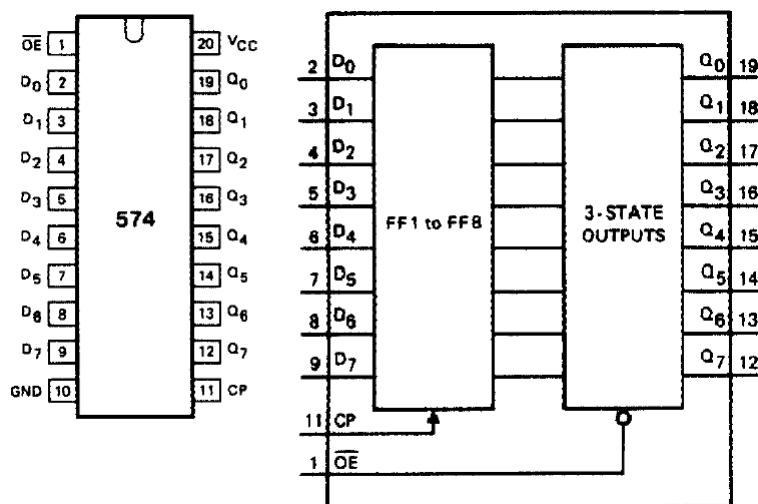
Często istnieje potrzeba zastosowania w mse takich prostych układów jak 8-bitowy rejestr zatraskujący. Np. układy te stosuje się w celu „zwiększenia” liczby linii wyjściowych mk, jako układy pamiętające młodszą część adresu przy adresowaniu zewnętrznej pamięci RAM i ROM przez mk 8051, itp.

Jednym z takich układów jest 8-bitowy rejestr zatraskujący 74HC/HCT574. Posiada on następujące właściwości:

- trzystanowe nieodwracające wyjścia, co pozwala na stosowanie układu w systemach magistralowych,
- zawiera w sobie 8-bitowy rejestr składający się z przerzutników typu D wyzwalanych zboczem narastającym,
- każdemu przerzutnikowi przyporządkowana jest linia wejściowa i wyjściowa,
- dla wszystkich przerzutników jest wspólny sygnał zegara CP (clock) i sygnał sterowania ich buforami wyjściowymi OE.

Gdy na linii CP pojawi się zbocze narastające, to stan sygnałów na liniach wejściowych układu (D0 – D7) zostaje zapamiętany w rejestrze. Kiedy sygnał OE jest w stanie wysokim wyjścia układu (Q0 – Q7) są w stanie wysokiej impedancji. Stan niski na linii OE spowoduje wystawienie zawartości rejestru na linie wyjściowe. Linia OE nie wpływa na zawartość rejestrów układu.

Na rys. 3.1 pokazano wyprowadzenia końcówek układu oraz jego schemat funkcjonalny.



Rys. 3.1. Schemat wyprowadzeń oraz schemat funkcjonalny układu 74HC574

Jak widać z rys. 3.1 układ składa się z rejestru zatrzymującego złożonego z ośmiu przerzutników typu D oznaczonych od FF1 do FF8 (*flip-flop*) i bloku ośmiu buforów trójstanowych. Rejestry są sterowane wyłącznie sygnałem CP, a bufony sygnałem OE.

Poniżej przedstawiono tabelę 3.1 opisującą sposób sterowania tym układem.

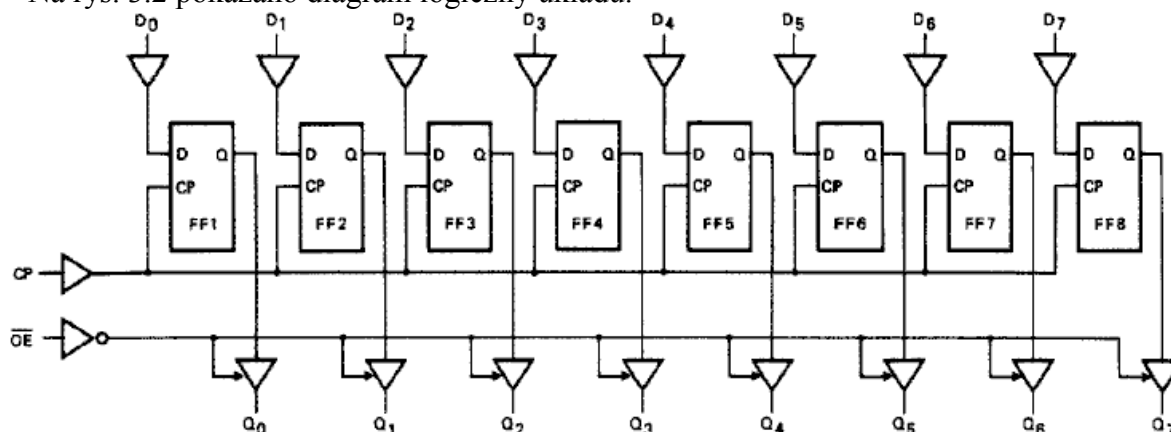
Tabela 3.1. Sterowanie układem 74HC574

OPERATING MODES	INPUTS			INTERNAL FLIP-FLOPS	OUTPUTS
	$\overline{OE}$	CP	$D_n$		$Q_0$ to $Q_7$
load and read register	L	$\uparrow$	L	L	L
	L	$\uparrow$	h	H	H
load register and disable outputs	H	$\uparrow$	L	L	Z
	H	$\uparrow$	h	H	Z

#### Notes

1. H = HIGH voltage level  
h = HIGH voltage level one set-up time prior to the LOW-to-HIGH CP transition  
L = LOW voltage level  
l = LOW voltage level on set-up time prior to the LOW-to-HIGH CP transition  
Z = HIGH impedance OFF-state  
 $\uparrow$  = LOW-to-HIGH clock transition

Na rys. 3.2 pokazano diagram logiczny układu.

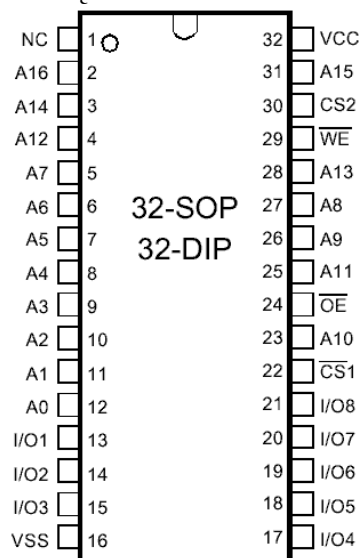


Rys. 3.2. Diagram logiczny układu 74HC574

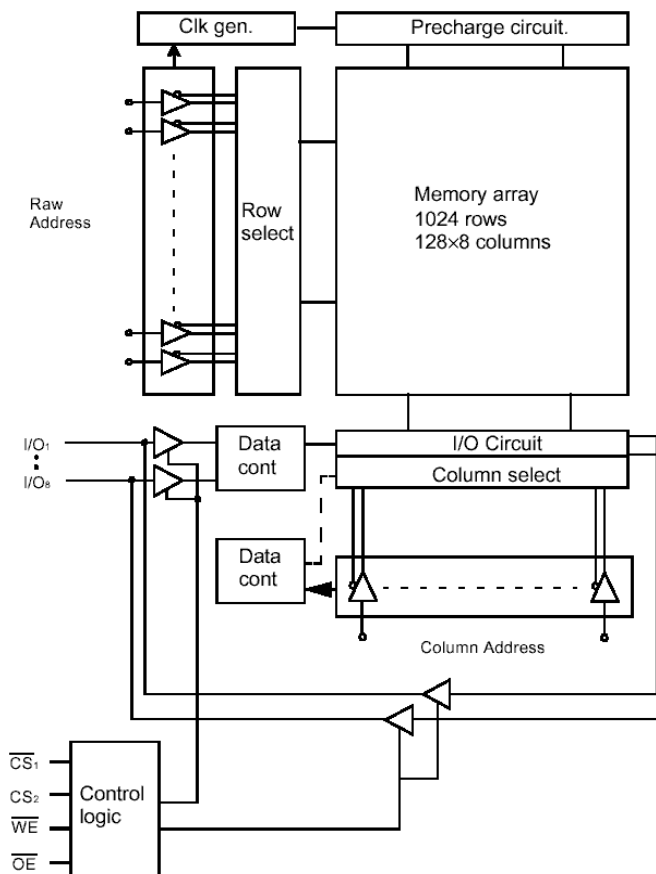
### 3.2. Pamięć SRAM 128kB

Pamięć SRAM zostanie przedstawiona na przykładzie układu K6T1008C2E firmy Samsung Electronics. Jest to pamięć SRAM o pojemności 128Kx8 bitów wykonana w technologii CMOS o małym poborze mocy.

Na rys. 3.3 pokazano wyprowadzenia układu. Są one zgodne ze standardem dla pamięci o tej pojemności. Również identyczne wyprowadzenia mają pamięci np. 512Kx8 bitów, przy czym pin 1 jest linią A17, a pin 30 linią A18.



Rys. 3.3. Wyprowadzenia układu K6T1008C2E



Rys. 3.4. Schemat blokowy układu K6T1008C2E

Na rys. 3.4 pokazano schemat blokowy układu pamięci SRAM. Składa się on z układu kontrolnego, do którego podłączone są sygnały CS<sub>1</sub>, CS<sub>2</sub>, WE i OE za pośrednictwem których steruje się pamięcią, układu we/wy służącego do wymiany danych pomiędzy matrycą pamięci składającej się z 1024 rzędów i 128x8 kolumn, a liniami danych I/O<sub>1</sub> do I/O<sub>8</sub>. Wybór komórki (bajtu) do zapisu lub odczytu odbywa się za pomocą układów wyboru wierszy i rzędów do których podłączone są linie adresowe A<sub>0</sub> – A<sub>16</sub>.

W tabeli 3.2 przedstawiono znaczenie poszczególnych sygnałów.

Tabela 3.2. Funkcje sygnałów dla układu K6T1008C2E

Name	Function
$\overline{\text{CS}}_1, \text{CS}_2$	Chip Select Input
$\overline{\text{OE}}$	Output Enable Input
$\overline{\text{WE}}$	Write Enable Input
I/O <sub>1</sub> ~I/O <sub>8</sub>	Data Inputs/Outputs
A <sub>0</sub> ~A <sub>16</sub>	Address Inputs
V <sub>cc</sub>	Power
V <sub>ss</sub>	Ground
NC	No Connection

Sygnały CS<sub>1</sub> i CS<sub>2</sub> służą do uaktywnienia układu. Najczęściej steruje się pierwszym sygnałem, drugi jest na stałe podłączony do „1”. Sygnał OE jest wykorzystywany w trakcie czytania danej, a sygnał WE w trakcie pisania danej do pamięci za pośrednictwem dwukierunkowych linii I/O<sub>1</sub> – I/O<sub>8</sub> pod adres wskazany przez linie A<sub>0</sub> – A<sub>16</sub>.

Zastosowanie tych sygnałów pokazano w następnej tabeli (tabela 3.3).

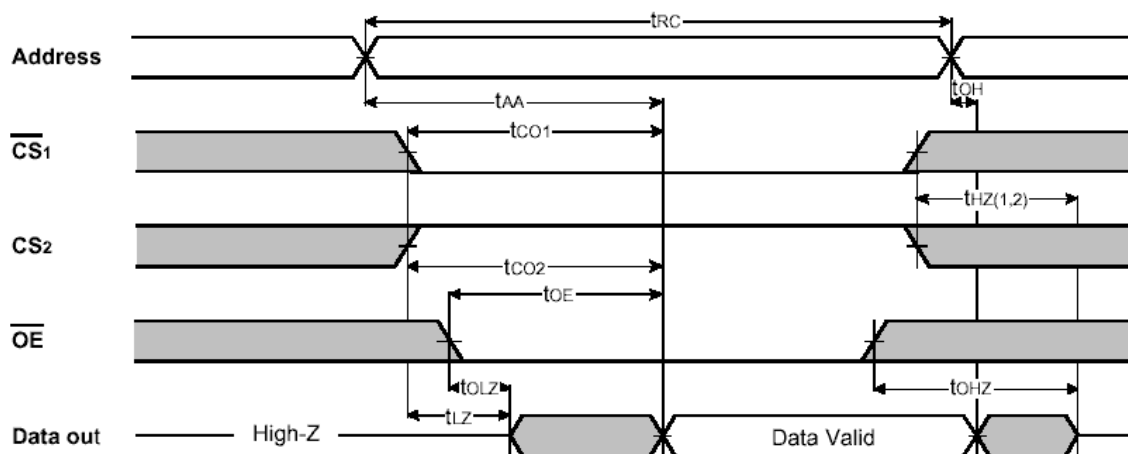
Tabela 3.3. Zastosowanie sygnałów sterujących dla układu K6T1008C2E

$\overline{\text{CS}}_1$	CS <sub>2</sub>	$\overline{\text{OE}}$	$\overline{\text{WE}}$	I/O	Mode	Power
H	X <sup>1)</sup>	X <sup>1)</sup>	X <sup>1)</sup>	High-Z	Deselected	Standby
X <sup>1)</sup>	L	X <sup>1)</sup>	X <sup>1)</sup>	High-Z	Deselected	Standby
L	H	H	H	High-Z	Output Disabled	Active
L	H	L	H	Dout	Read	Active
L	H	X <sup>1)</sup>	L	Din	Write	Active

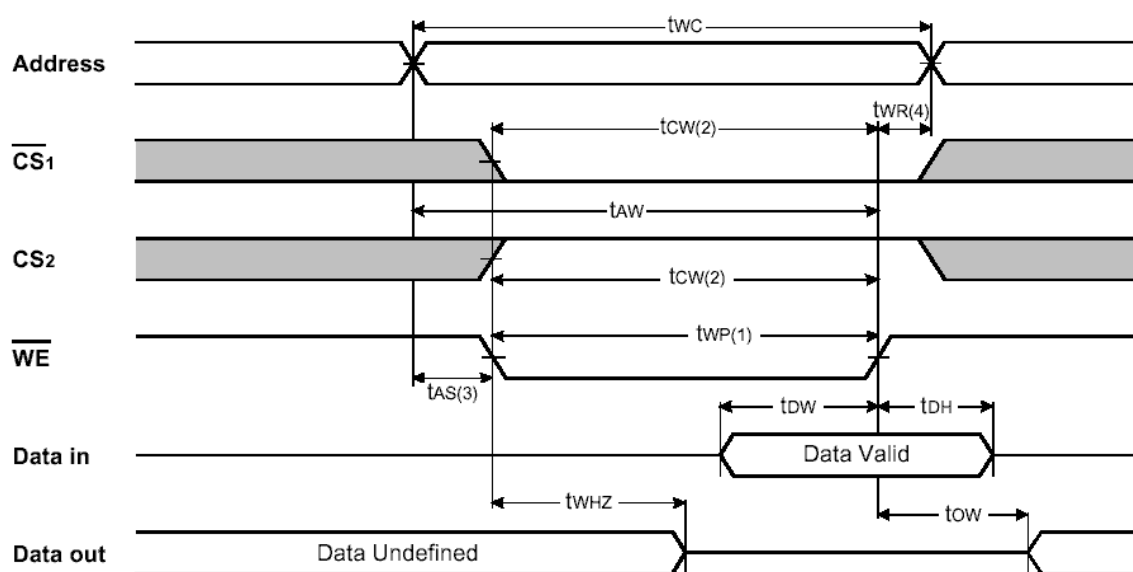
1. X means don't care (Must be in high or low states)

Jak widać z tabeli 3.3, aby móc zapisać bajt do pamięci musi być ona aktywna (CS<sub>1</sub> – stan niski (L), CS<sub>2</sub> – stan wysoki (H)) oraz na linii OE ma być stan niski (aktywny), a na linii WE stan wysoki (nieaktywny). Podobnie jest dla zapisu. W tym przypadku sygnał WE jest niski (warto aby OE było w stanie wysokim).

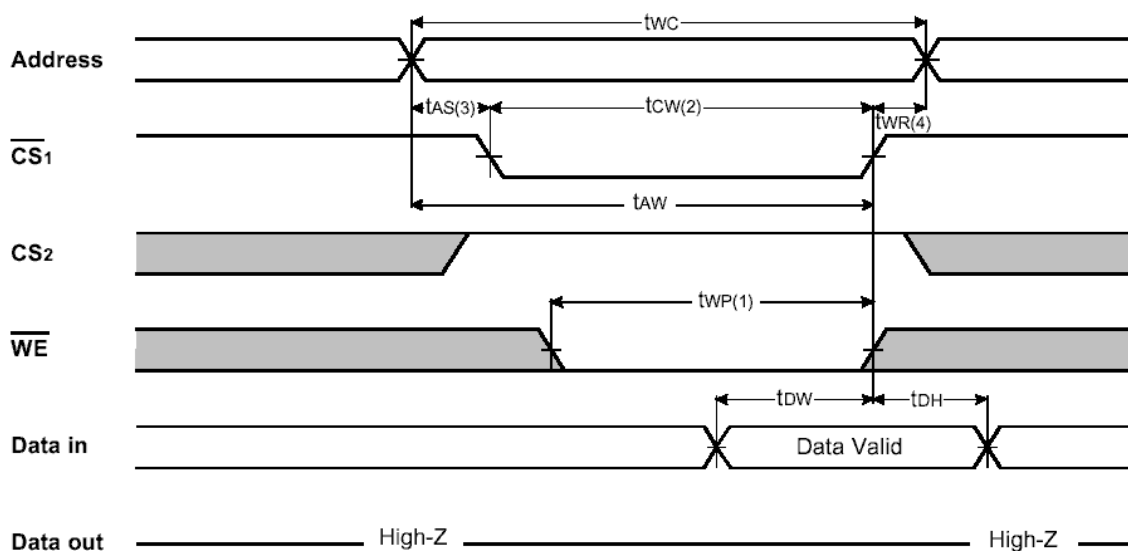
Na poniższych rysunkach pokazano przebiegi cyklu odczytu bajtu z pamięci SRAM (rys. 3.5) oraz zapisu do niej bajtu dwoma sposobami, przy czym zakłada się, że sygnał CS<sub>2</sub> jest nieaktywny (na zacisku jest stan wysoki). Pierwszy sposób oparty jest na sterowaniu zapisem danej sygnałem WE (rys. 3.6), natomiast drugi bazuje na sterowaniu zapisem za pomocą sygnału CS<sub>1</sub> (rys. 3.7).



Rys. 3.5. Przebiegi czasowe cyklu odczytu danych z pamięci (sygnał WE w stanie „Hi”)



Rys. 3.6. Przebiegi czasowe cyklu zapisu danych do pamięci (sterowanie sygnałem WE)



Rys. 3.7. Przebiegi czasowe cyklu zapisu danych do pamięci (sterowanie sygnałem CS1)

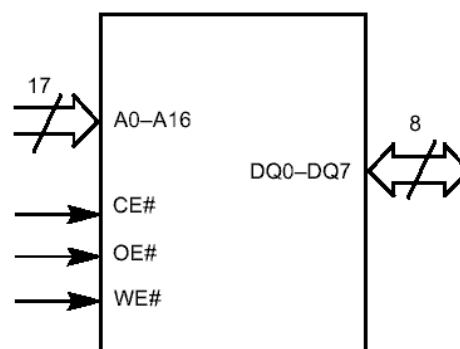
### 3.2. Pamięć FLASH 128kB

Szczególną zaletą pamięci błyskowych (FLASH) w stosunku do pamięci EEPROM jest krótki czas kasowania i zapisu. Do programowania i kasowania pamięci błyskowych potrzebne jest **podwyższone napięcie** 11,5 - 13,5V, które w zależności od typu układu może być podawane z zewnątrz lub wytwarzane przez **wbudowaną przetwornicę**.

Pamięci FLASH przyjmują oprócz danych polecenia, które umożliwiają realizację różnych rodzajów operacji.

Ten typ pamięci zostanie przedstawiony na przykładzie popularnej pamięci typu 29F010 firmy AMD (*Advanced Micro Devices*) o organizacji 128k x 8 bitów. Na rys. 3.8 pokazano symbol logiczny pamięci wraz z opisem linii.

A0–A16	=	17 Addresses
DQ0–DQ7	=	8 Data Inputs/Outputs
CE#	=	Chip Enable
OE#	=	Output Enable
WE#	=	Write Enable
V <sub>CC</sub>	=	+5.0 Volt Single Power Supply (See Product Selector Guide for speed options and voltage supply tolerances)
V <sub>SS</sub>	=	Device Ground
NC	=	Pin Not Connected Internally



Rys. 3.8. Symbol logiczny pamięci Am29F010

Pamięć ta posiada następujące właściwości:

- pojedyncze zasilanie, 5V +/-10% dla czytania, zapisu i operacji kasowania programu,
- maksymalny czas dostępu 45ns,
- niskie zużycie energii, maksymalnie 30mA na odczyt i 50mA na programowanie oraz czyszczenie, mniej niż 25µA podczas trybu standby,
- elastyczna architektura oparta na sektorach, 8 zunifikowanych sektorów, wszystkie kombinacje sektorów mogą być czyszczone,
- możliwość wymazania całego chipu,
- ochrona sektorów przed zapisem,
- wsparcie sprzętowe dla zablokowania i odblokowania programowania i czyszczenia dla wszystkich kombinacji sektorów,
- wbudowany algorytm czyszczenia automatycznie przeprogramuje i czyści chip lub wszystkie kombinacje wybranych sektorów,
- wbudowany algorytm programowania automatycznie programujący i sprawdzający dane z zadanego adresu,
- minimum 100000 gwarantowanych cykli programowania i czyszczenia,
- programowe metody detekcji końca cyklu programowania i czyszczenia.

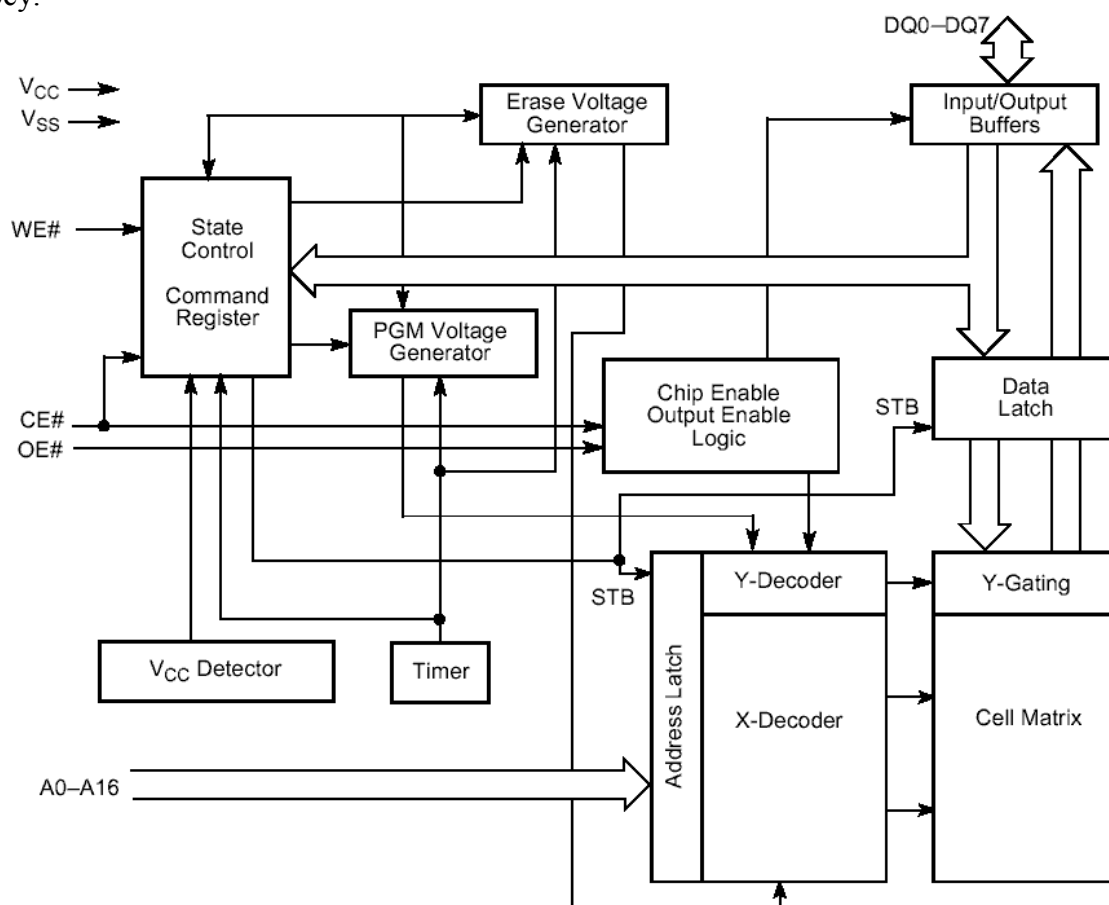
Pamięć Am29F010 jest 1Mbit, 5V pamięcią FLASH zorganizowaną jako 131072 bajtów. Produkowana jest w 32-pinowych obudowach typu PLCC, TSOP i PDIP. Zaprojektowano ją z możliwością programowania w systemie ze standardowym 5V zasilaniem. Ponadto może być programowana za pomocą standardowego programatora do pamięci EPROM.

Pamięć posiada wbudowany algorytm programowania (wewnętrzny algorytm, który automatycznie weryfikuje szerokość i czas trwania impulsów programowania). Zawiera ona

także wbudowany algorytm czyszczenia. Podczas czyszczenia urządzenie automatycznie kontroluje czas i szerokość sygnałów oraz weryfikuje właściwe komórki docelowe.

System nadrzędny może wykrywać zakończenia operacji czyszczenia/programowania poprzez czytanie bitów statusu : DQ7 (*DATA#Polling*) i DQ6 (*toggle*). Po zakończeniu cyklu programowania/czyszczenia pamięć jest gotowa do czytania z niej danych lub oczekuje na następną komendę.

Sektorowa architektura pamięci pozwala na czyszczenie, czy ponowne programowanie wybranych sektorów pamięci bez wpływu na pozostałe sektory. Sprzętowa ochrona danych wykrywa niski poziom napięcia zasilania i wstrzymuje proces zapisywania podczas zmiany napięcia. Pamięć można przełączyć w tryb standby, w celu znacznej redukcji pobieranej mocy.



Rys. 3.9. Schemat blokowy pamięci Am29F010

Na rys. 3.9 przedstawiono schemat blokowy pamięci Am29F010. Jak widać, składa się ona z wielu bloków, z których część służy do obsługi procesu zapisu do pamięci (blok *Cell Matrix*), są nimi generatory napięć *PGM Voltage Generator* i *Erase Voltage Generator*. Pracą całego układu zarządza blok *State Control / Command Register*. Komórki pamięci są wybierane za pomocą układu dekodera adresu. Sterowanie przepływem danych odbywa się za pośrednictwem dwukierunkowego bufora i rejestru zatrzymującego oraz bloku *Chip Enable / Output Enable Logic*.

Rejestr komend (*Command Register*) składa się z zatrząsków, które przechowują komendy z adresami i danymi potrzebnymi do wykonania komendy. Zawartość rejestru służy jako wejścia dla „wewnętrznej maszyny stanu”. Wyjścia maszyny stanu dyktują funkcje urządzenia. Funkcje te (operacje na szynie danych) zestawiono w tabeli 3.4.



Tabela 3.4. Operacje na szynie danych pamięci Am29F010

Operation	CE#	OE#	WE#	Addresses	DQ0–DQ7
Read	L	L	H	A <sub>IN</sub>	D <sub>OUT</sub>
Write	L	H	L	A <sub>IN</sub>	D <sub>IN</sub>
Standby	V <sub>CC</sub> ± 0.5 V	X	X	X	High-Z
Output Disable	L	H	H	X	High-Z
Hardware Reset	X	X	X	X	High-Z

**Legend:**

L = Logic Low = V<sub>IL</sub>, H = Logic High = V<sub>IH</sub>, V<sub>ID</sub> = 12.0 ± 0.5 V, X = Don't Care, A<sub>IN</sub> = Addresses In, D<sub>IN</sub> = Data In, D<sub>OUT</sub> = Data Out

Aby odczytać dane z pamięci system musi zapewnić niski stan linii CE# i OE#. Linia CE# kontroluje odczyt. Linia OE# steruje buforem wyjściowym. Sygnał na linii WE# powinien być na poziomie V<sub>IH</sub>. Żadna komenda nie jest wymagana w tym trybie. Jest on trybem domyślnym po włączeniu napięcia zasilania. Zatem mk może podawać na linie adresowe A0 – A16 adresy danych, które chce otrzymać na liniach danych DQ0 – DQ7. Urządzenie pozostaje dostępne do odczytu dopóki nie zmieni się zawartość rejestru komend.

Aby zapisać komendę lub sekwencję komend (które zawierają dane programujące czy czyszczące wybrane sektory pamięci), system musi zapewnić następujące stany: na liniach WE# i CE# stan niski V<sub>IL</sub> oraz na linii OE# stan wysoki V<sub>IH</sub>.

Operacja czyszczenia może skasować jeden sektor, wiele sektorów lub całe urządzenie. Tablica adresów sektorów wskazuje na adres w przestrzeni jaką każdy sektor zajmuje. „Adres sektora” to bity adresu potrzebne aby jednoznacznie zidentyfikować sektor (patrz tabela 3.5).

Tabela 3.5. Tablica adresów sektorów pamięci Am29F010

Sector	A16	A15	A14	Address Range
SA0	0	0	0	00000h-03FFFh
SA1	0	0	1	04000h-07FFFh
SA2	0	1	0	08000h-0BFFFh
SA3	0	1	1	0C000h-0FFFFh
SA4	1	0	0	10000h-13FFFh
SA5	1	0	1	14000h-17FFFh
SA6	1	1	0	18000h-1BFFFh
SA7	1	1	1	1C000h-1FFFFh

**Note:** All sectors are 16 Kbytes in size.

Podczas operacji czyszczenia lub programowania system może sprawdzić status operacji przez czytanie bitów statusu DQ7-DQ0.

Pamięć posiada tryb autoselect, który umożliwia identyfikację urządzenia oraz producenta, a także weryfikację chronionych sektorów przez kody identyfikacyjne podane na DQ7-DQ0. Ten tryb jest niezbędny dla urządzeń programujących, aby mogły wykorzystać odpowiedni algorytm programujący. Ten tryb może być także wykorzystany w systemie przez rejestr komend. Pamięć można wprowadzić w tryb autoselect wydając odpowiednią komendę (tabela 3.6).

Zapis adresu i danej rozkazu lub całej sekwencji rozkazowej do rejestru rozkazów inicjuje wykonanie operacji. Poprawne sekwencje rozkazowe przedstawia tabela rozkazów (tabela 3.6). Zapis niepoprawnych wartości adresu i danych lub zapis sekwencji w nieodpowiedniej kolejności przestawia pamięć w tryb odczytu danych.

Adres jest "zatraskiwany" w momencie pojawienia się opadającego zbocza na linii WE# lub CE#, w zależności od tego, które z tych zdarzeń wystąpi jako ostatnie.

Tabela 3.6. Formaty poleceń sterowania pamięcią Am29F010

Command Sequence		Cycles	Bus Cycles											
			First		Second		Third		Fourth		Fifth		Sixth	
			Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read		1	RA	RD										
Reset		1	XXXX	F0										
Reset		3	555	AA	2AA	55	555	F0						
Autoselect	Manufacturer ID	4	555	AA	2AA	55	555	90	X00	01				
	Device ID	4	555	AA	2AA	55	555	90	X01	20				
	Sector Protect Verify	4	555	AA	2AA	55	555	90	(SA) X02	00 01				
Program		4	555	AA	2AA	55	555	A0	PA	PD				
Chip Erase		6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
Sector Erase		6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30
Erase Suspend		1	XXX	B0										
Erase Resume		1	XXX	30										

**Legend:**

X = Don't care

RA = Address of the memory location to be read.

RD = Data read from location RA during read operation.

PA = Address of the memory location to be programmed.

PD = Data to be programmed at location PA. Data latches on the rising edge of WE# or CE# pulse, whichever happens first.

SA = Address of the sector to be verified (in autoselect mode) or erased. Address bits A16–A14 uniquely select any sector.

Po włączeniu zasilania domyślnie ustawiany jest tryb odczytu danych. Tryb ten jest ustawiany domyślnie także po wykonaniu wbudowanego algorytmu programowania (*Embedded Program Algorithm*) lub wbudowanego algorytmu kasowania (*Embedded Erase Algorithm*). Odczyt jest możliwy bez użycia jakichkolwiek rozkazów.

Rozkaz AUTOSELECT umożliwia uzyskanie dostępu do kodów producenta i pamięci, a także pozwala na ustalenie, czy dany sektor jest chroniony. Na sekwencję rozkazową składa się rozkaz AUTOSELECT poprzedzony zapisem 2 cykli odblokowujących. Następnie pamięć wchodzi w tryb autoselect i możliwy jest odczyt spod każdego adresu dowolną ilość razy bez inicjowania kolejnej sekwencji rozkazowej. Cykl odczytu spod adresu XX00h zwraca kod producenta, a spod adresu XX01h kod urządzenia. Cykl odczytu zawierający adres sektora (SA) i adres 02h zwraca 01h jeśli ten sektor jest chroniony, a 00h w przeciwnym wypadku. W celu wyjścia z trybu autoselect do pamięci trzeba przesłać rozkaz RESET.

Programowanie pamięci składa się z 4 cykli szyny (rozkaz PROGARM). Na sekwencję rozkazową składa się rozkaz ustawiający tryb programowania poprzedzony zapisem 2 cykli odblokowujących. Dane i adres są wysyłane w następnej kolejności. Operacja ta inicjuje wykonanie wbudowanego algorytmu programowania. System programujący nie musi dostarczać dodatkowych sygnałów sterujących czy taktujących. Urządzenie samo generuje wewnętrzne impulsy programujące i weryfikuje poprawność zaprogramowania komórki. Kiedy wbudowany algorytm programowania zakończy działanie urządzenie wraca do trybu odczytu danych i zwalniany jest przerzutnik "zatraskujący" adresy. System nadrzędny może ustalić status operacji programowania na podstawie stanu linii DQ6 i DQ7. Wszystkie rozkazy wysyłane do urządzenia podczas wykonywania wbudowanego algorytmu

programowania są ignorowane. Programowanie może odbywać się w dowolnej kolejności i może przekraczać granice sektorów.

Sekwencja kasowania pamięci składa się 6 cykli szyny. W jej skład wchodzi rozkaz ustawienia (*set-up*) poprzedzony zapisem 2 cykli odblokowujących. Po nich następują kolejne 2 cykle odblokowujące zapis i rozkaz kasowania, który uruchamia wbudowany algorytm kasowania. Wszystkie rozkazy wysyłane do urządzenia podczas wykonywania wbudowany algorytm kasowania są ignorowane. Kiedy ten algorytm zakończy działanie urządzenie wraca do trybu odczytu danych i zwalniany jest przerzutnik "zatrzasujący" adresy.

Na rozkaz kasowanie sektora również składa się 6 cykli szyny. Pierwszym jest rozkaz ustawienia (*set-up*) poprzedzony zapisem 2 cykli odblokowujących. W następnej kolejności wysyłane są 2 dodatkowe cykle odblokowujące. Potem zapisywane są: adres sektora do kasowania oraz rozkaz kasowania. Po zapisie sekwencji rozkazowej rozpoczyna się 50µs przerwy. W trakcie tej przerwy mogą zostać przesłane dodatkowe adresy i rozkazy kasowania sektorów. Załadowanie bufora kasowania sektorów może się odbyć w dowolnej kolejności. Dowolna może być także liczba sektorów przeznaczonych do skasowania. Odstęp czasu między tymi dodatkowymi cyklami, w których przesyłane są informacje o dodatkowych sektorach do skasowania musi być mniejszy niż 50µs. Jakikolwiek rozkaz wysłany podczas 50µs przerwy przedstawia urządzenie w tryb odczytu. System musi ponownie wysłać sekwencje rozkazową oraz adresy dodatkowych sektorów i rozkazy. System może nadzorować stan linii DQ3 w celu sprawdzenia, czy minął czas przerwy. Przerwa zaczyna się od narastającego zbocza ostatniego impulsu w sekwencji rozkazowej na linii WE#. Wszystkie rozkazy wysyłane do urządzenia podczas kasowania sektora są ignorowane. Kiedy wbudowany algorytm kasowania zakończy działanie urządzenie wraca do trybu odczytu danych i zwalniany jest przerzutnik "zatrzasujący" adresy. System nadrzędny może ustalić status operacji programowania przy wykorzystaniu linii DQ6 i DQ7.

Stan operacji zapisu można sprawdzić wykorzystując linie: DQ3, DQ5, DQ6 i DQ7. Za pomocą pinów DQ7 i DQ6 można sprawdzić, czy operacja programowania lub kasowania została zakończona (tabela 3.7).

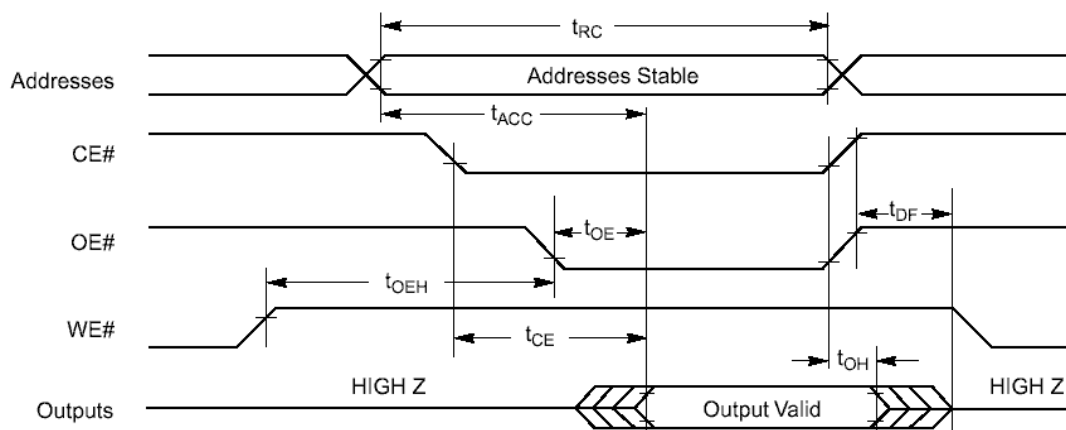
Tabela 3.7. Status operacji zapisu

Operation		DQ7 (Note 1)	DQ6	DQ5 (Note 2)	DQ3
Standard Mode	Embedded Program Algorithm	DQ7#	Toggle	0	N/A
	Embedded Erase Algorithm	0	Toggle	0	1
Erase Suspend Mode	Reading within Erase Suspended Sector	1	No toggle	0	N/A
	Reading within Non-Erase Suspended Sector	Data	Data	Data	Data

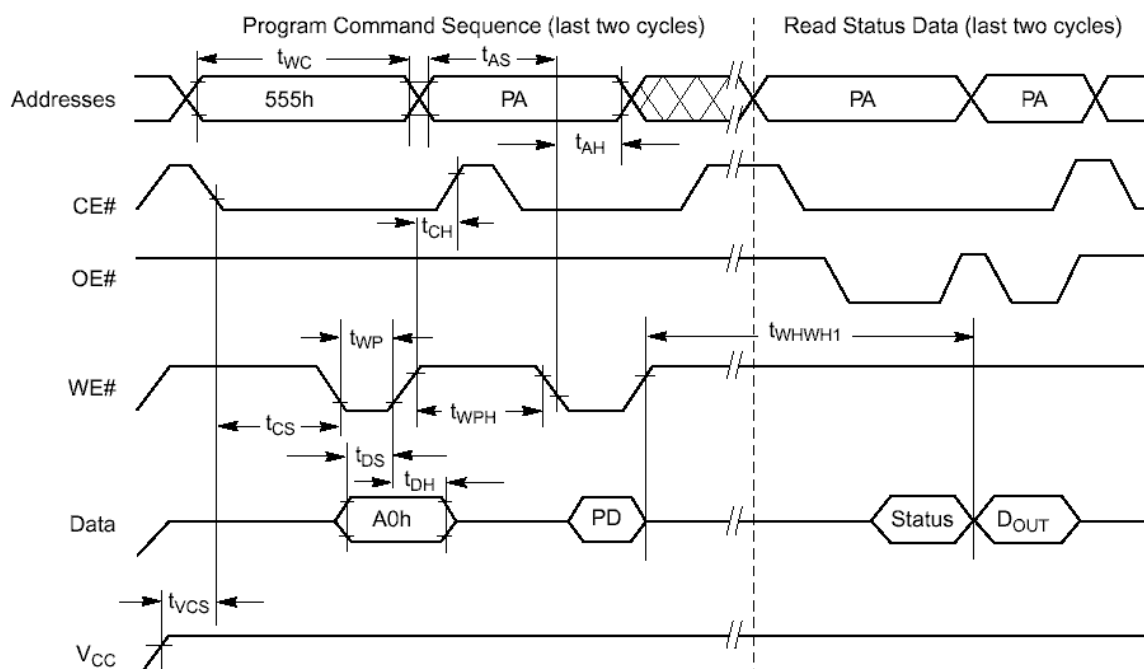
**Notes:**

1. DQ7 requires a valid address when reading status information.
2. DQ5 switches to '1' when an Embedded Program or Embedded Erase operation has exceeded the maximum timing limits.

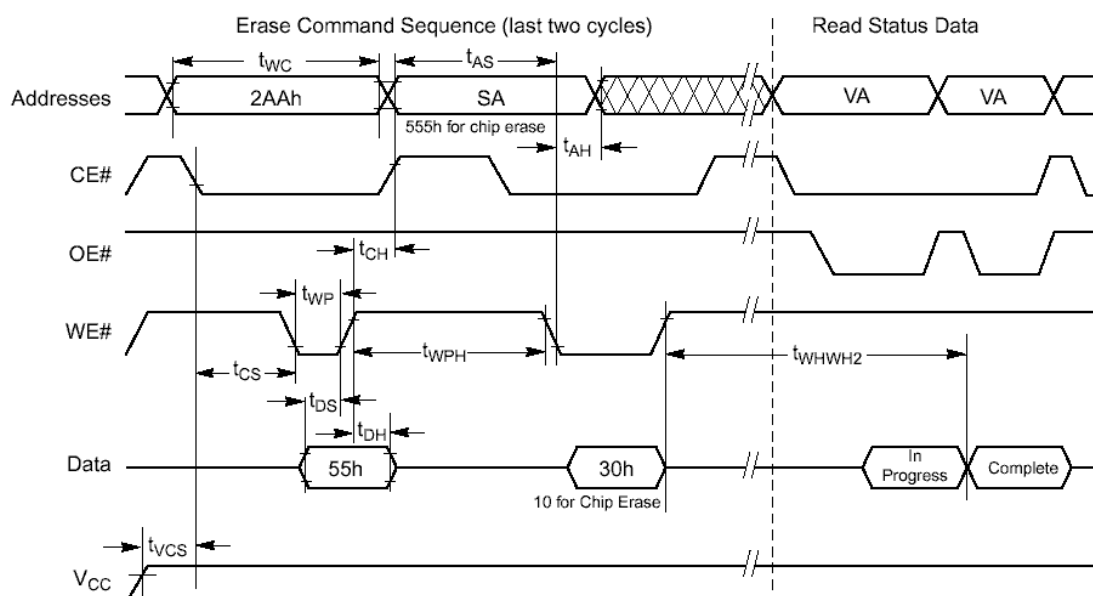
Na poniższych rysunkach pokazano przebiegi czasowe operacji odczytu (rys. 3.10), operacji zapisu danej do pamięci (rys. 3.11) oraz operacji kasowania pamięci (rys. 3.12).



Rys. 3.10. Przebieg czasowy operacji odczytu z pamięci Am29F010



Rys. 3.11. Przebieg czasowy operacji programowania pamięci Am29F010



Rys. 3.12. Przebieg czasowy operacji kasowania pamięci Am29F010

## 4. Programowalne układy logiczne

Termin logiczne układy programowalne PLD (*Programmable Logic Devices*) odnosi się do każdego cyfrowego układu scalonego, którego właściwości funkcjonalne mogą być **zdefiniowane (ustalane) przez końcowego użytkownika**, który może zaimplementować w jego strukturze opracowany przez siebie projekt jakiegoś wyspecjalizowanego układu cyfrowego. Najważniejszą cechą tych układów jest więc ich **konfigurowalność** przez użytkownika w jego własnym laboratorium.

Współczesne układy programowalne klasyfikuje się najczęściej w trzech grupach, przy czym kryterium klasyfikacji są głównie cechy ich architektury. Najczęściej przyjmuje się, że układy PLD dzieli się na:

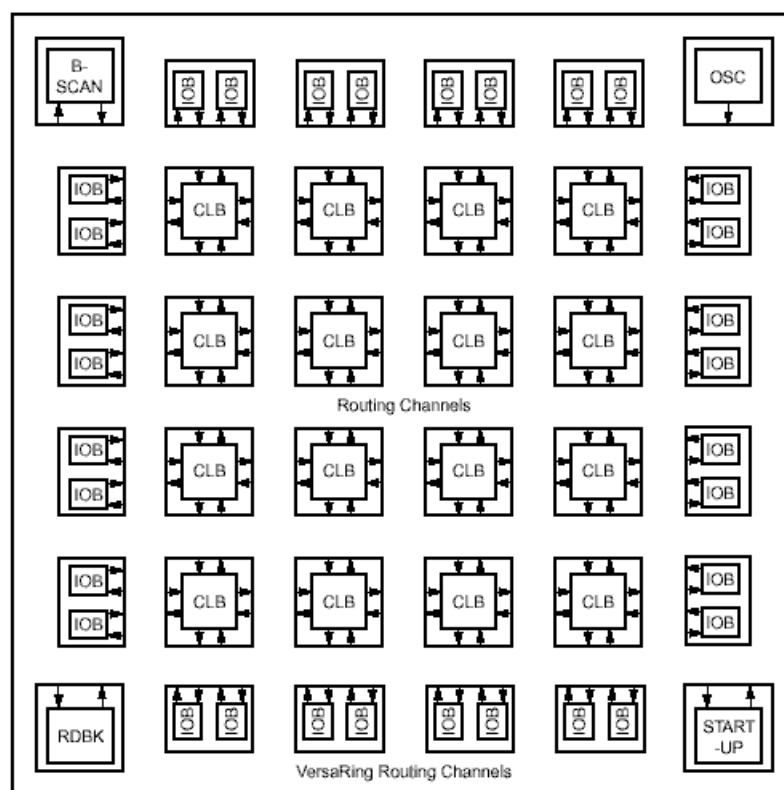
- **SPLD** (*Simple Programmable Logic Device*) – proste układy programowalne,
- **CPLD** (*Complex Programmable Logic Devices*) – złożone układy programowalne,
- **FPGA** (*Field Programmable Gate Array*) – programowalne matryce bramkowe.

Do układów **SPLD** zalicza się układy programowalne o architekturach **PLA** (*Programmable Logic Array*), **PAL** (*Programmable Array Logic*) i **GAL** (*Generic Array Logic*). Są to układy o najskromniejszych możliwościach logicznych, a więc i najtańsze ze wszystkich układów programowalnych. Niemniej jednak ich zasoby logiczne są spore, zawierają bowiem typowo od 4 do 22 makrokomórek logicznych o dwupoziomowej strukturze logicznej i mogą zwykle zastąpić kilka standardowych układów scalonych rodziny 74xx. Każda z komórek jest zwykle w pełni połączona z innymi komórkami w danym układzie scalonym. Do określenia funkcji realizowanych przez makrokomórki (ich skonfigurowania) stosowane są **łączniki** (klucze), którymi są w większości przypadków przepalane fragmenty ścieżek (łączniki rozwarciowe – dla układów PLA i PAL) lub tranzystory MOS (dla układów GAL).

Układy **CPLD** są koncepcyjnie podobne do układów SPLD, lecz są bardziej złożone: mają większe zasoby logiczne i możliwości funkcjonalne. Ich architektura ma strukturę **hierarchiczną** opartą na **makrokomórkach logicznych**, których zawierają od kilkudziesięciu do kilkuset. Typowo od czterech do szesnastu makrokomórek jest połączonych w większy **blok funkcjonalny**. Jedną z ważniejszych cech architektury układów CPLD jest liczba termów przypadających na pojedynczą makrokomórkę oraz możliwość pożyczki termów z sąsiednich makrokomórek. Jeśli układ zawiera wiele bloków funkcjonalnych, są one łączone między sobą za pomocą **matrycy połączeniowej kluczy**, której zdolność łączeniowa jest ważną cechą układów CPLD. Ta liczba połączeń wewnątrz matrycy określa bowiem, jak łatwo jest „wpasować” jakiś projekt w dany układ programowalny.

Architektura układów **FPGA** różni się od architektury układów CPLD. Na ogół układy FPGA zawierają **rozmieszczone matrycowo** boki logiczne CLB (rys. 4.1). Poszczególne bloki są łączone ze sobą za pośrednictwem **linii traktów połączeniowych** (*Routing Channels*) oraz programowalnych matryc kluczy połączeniowych umieszczonych w miejscu krzyżowania się traktów poziomych i pionowych. Na obrzeżach matrycy bloków logicznych znajdują się programowalne bloki IOB (wejściowo-wyjściowe). Struktury FPGA zawierają od 64 do dziesiątków tysięcy **bloków logicznych** o bardzo zróżnicowanej budowie. Bloki logiczne mogą być bardzo złożone, jest ich wówczas mniej w układzie, lub względnie proste i jest ich wówczas więcej. Zazwyczaj złożone bloki logiczne zawierają dwie lub więcej pamięci RAM umożliwiających tworzenie tablic wartości funkcji LUT (*Look-up Table*) i dwa lub więcej przerzutników. W większości układów są to tablice czterowejściowe (pamięć RAM o pojemności 16 bitów). W układach o prostszej budowie, bloki logiczne zawierają

zwykle dwuwejściowe układy generacji funkcji kombinacyjnych lub multipleksery czterowejściowe i ewentualnie przerzutniki.



Rys. 4.1. Schemat blokowy układu FPGA

Do grupy największych producentów układów programowalnych i narzędzi komputerowych do ich syntezy należą następujące firmy: Altera ([www.altera.com](http://www.altera.com)), Lattice ([www.latticesemi.com](http://www.latticesemi.com)), Xilinx ([www.xilinx.com](http://www.xilinx.com)), Cypress ([www.cypress.com](http://www.cypress.com)), Atmel ([www.atmel.com](http://www.atmel.com)) i QuickLogic ([www.quicklogic.com](http://www.quicklogic.com)).

W dalszej części wykładu zostaną omówione dwie pierwsze grupy układów PLD: SPLD na przykładzie układu GAL16V8 firmy Lattice i CPLD na przykładzie rodziny układów XC9500 firmy Xilinx.

#### 4.1. Układy logiczne SPLD na przykładzie układu GAL16V8

Obecnie najpopularniejszą grupą układów SPLD są układy GAL, w których wykorzystano architekturę PAL wzbogaconą o konfigurowalne makrokomórki wyjściowe zawierające przerzutniki typu D. Przerzutniki te spełniają rolę wyjściowych elementów pamięciowych umożliwiających budowanie układów synchronicznych.

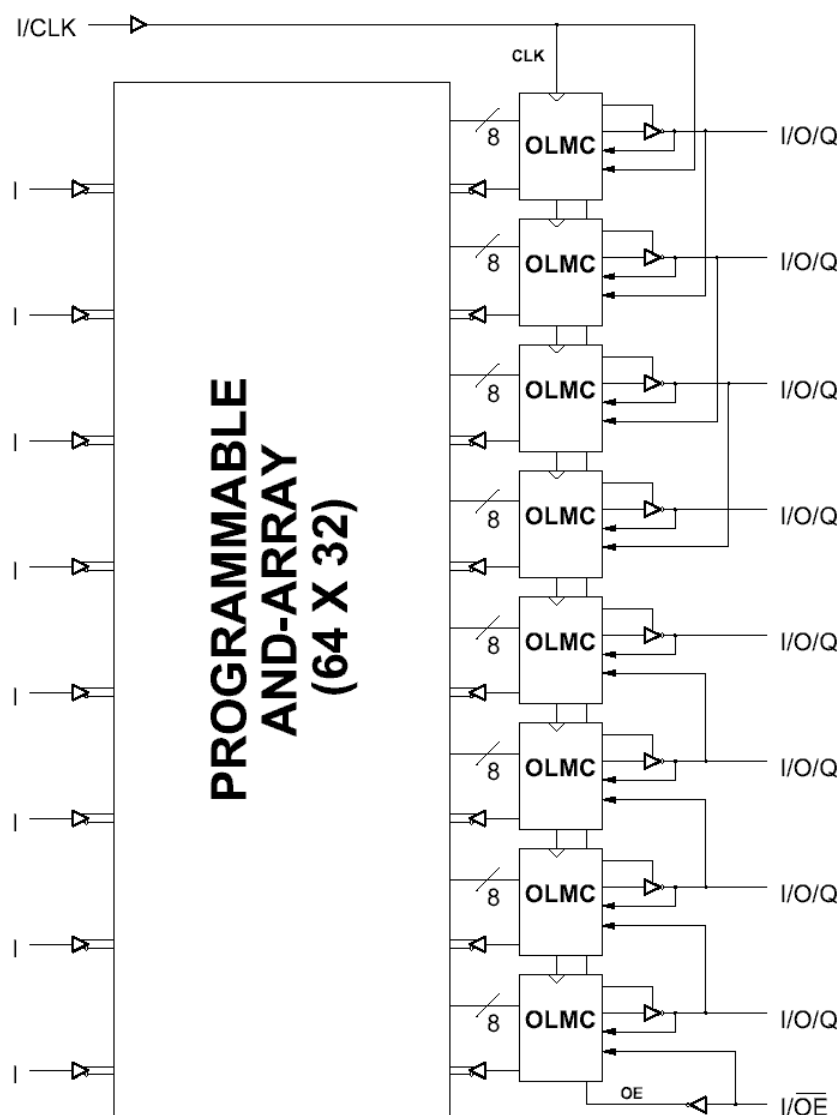
Układy programowalne GAL zostaną przedstawione na przykładzie układu GAL16V8 firmy Lattice. Jest on wykonany w technologii CMOS i zawiera elektrycznie reprogramowalne komórki pamięci typu E<sup>2</sup>CMOS. Układ ten charakteryzuje się następującymi cechami:

- Wysoko wydajna technologia E<sup>2</sup>CMOS (3,5ns maksymalny czas propagacji,  $F_{\max}=250\text{MHz}$ , 3ns od wejścia zegara do wyjścia danych).
- Rezystory podwieszone na każdej końcówce (*active pull-up*).
- Technologia E<sup>2</sup>CELL (rekonfigurowalna logika, reprogramowalne komórki, bardzo szybkie elektryczne kasowanie (poniżej 100ms), czas podtrzymywania danych 20 lat).

- Osiem wyjściowych makrokomórek (programowalna polaryzacja wyjść, emulacja 20-pinowych układów PAL).
- Wstępny zapis (*preload*) i reset po włączeniu zasilania (*power-on reset*) wszystkich rejestrów.
- Identyfikacyjny „elektroniczny podpis” zawarty w układzie.

Układ składa się z następujących bloków (rys. 4.2):

- z 9 **buforów wejściowych**,
- **matrycy połączeń logicznych PROGRAMMABLE AND-ARRAY**,
- z 8 programowalnych logicznych wyjściowych makrokomórek **OLMC** (*Output Logic MacroCell*),
- 8 **trójstanowych buforów wyjściowych** konfigurowanych przez użytkownika,
- **układu ochrony danych** przed odczytem.

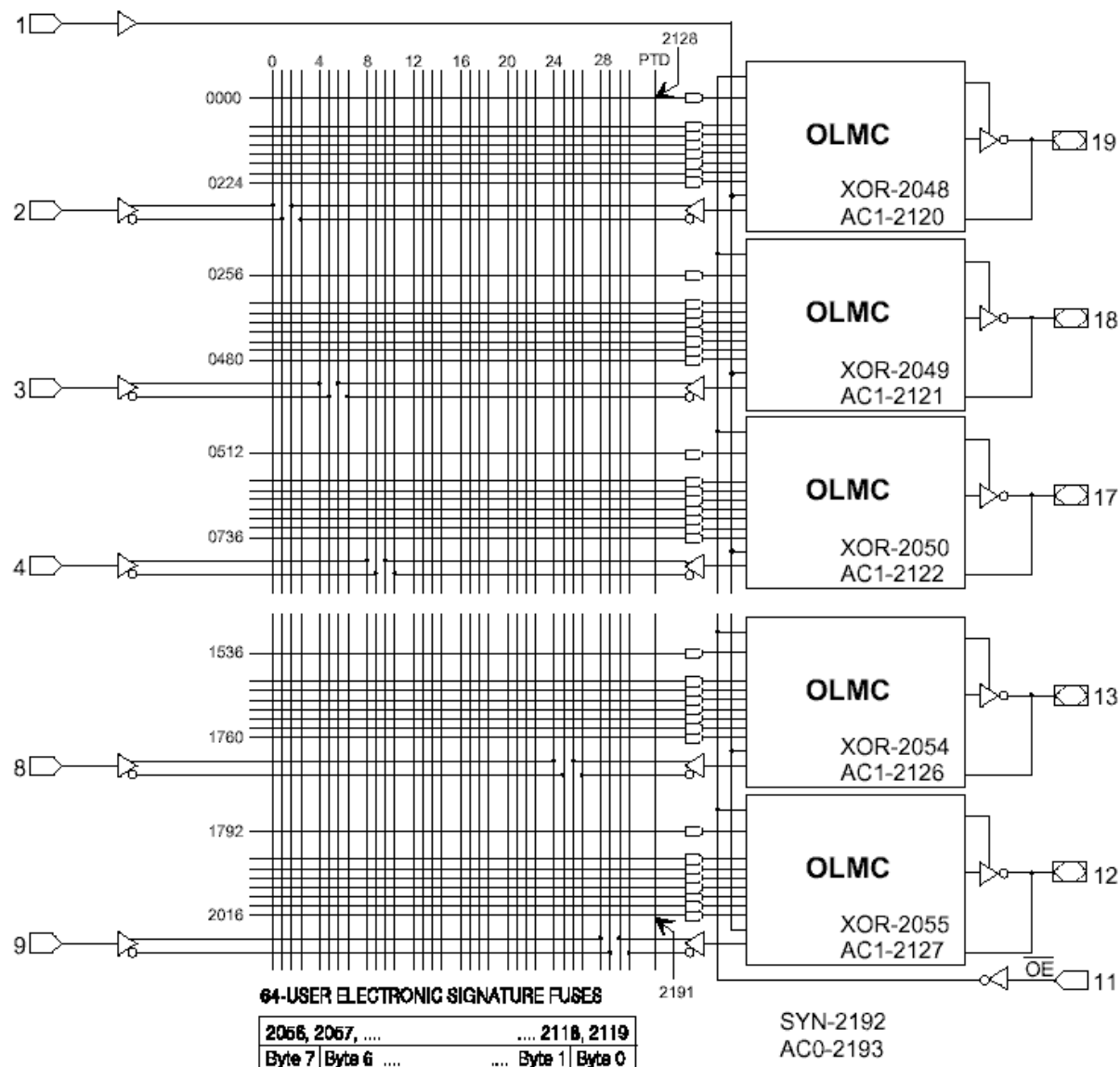


Rys. 4.2. Schemat blokowy układu GAL16V8

**Bufory wejściowe** układu GAL16V8 są kompatybilne ze standardowymi poziomami TTL. Bufory te charakteryzują się wysoką impedancją i reprezentują o wiele mniejsze obciążenie dla sterującej logiki niż bipolarne układy TTL.

Wejścia układu mają wbudowane aktywne rezystory podwieszające (*Active Pull-Up*), więc niepołączone będą w stanie „wysokim” (logiczna „1”). Producent zaleca aby wszystkie nieużywane wejścia układu i trzystanowe piny I/O były podłączone do zasilania układu lub masy. Zwiększa to odporność na zakłócenia i redukuje prąd pobierany przez układ.

**Matryca połączeń logicznych** składa się z programowalnej macierzy typu AND array z ustalonymi połączeniami do bramek typu OR. Pole logicznych połączeń jest zorganizowane jako 16 komplementarnych linii wejściowych (z sygnałami i ich negacjami) krzyżujących się z 64 liniami typu „product term”. Na każdym skrzyżowaniu linii znajduje się komórka typu E<sup>2</sup>PROM, która w zależności od zaprogramowania zwiera lub rozwiera linię poziomą od pionowej. Łącznie w matrycy znajduje się 2048 komórek (rys. 4.3).

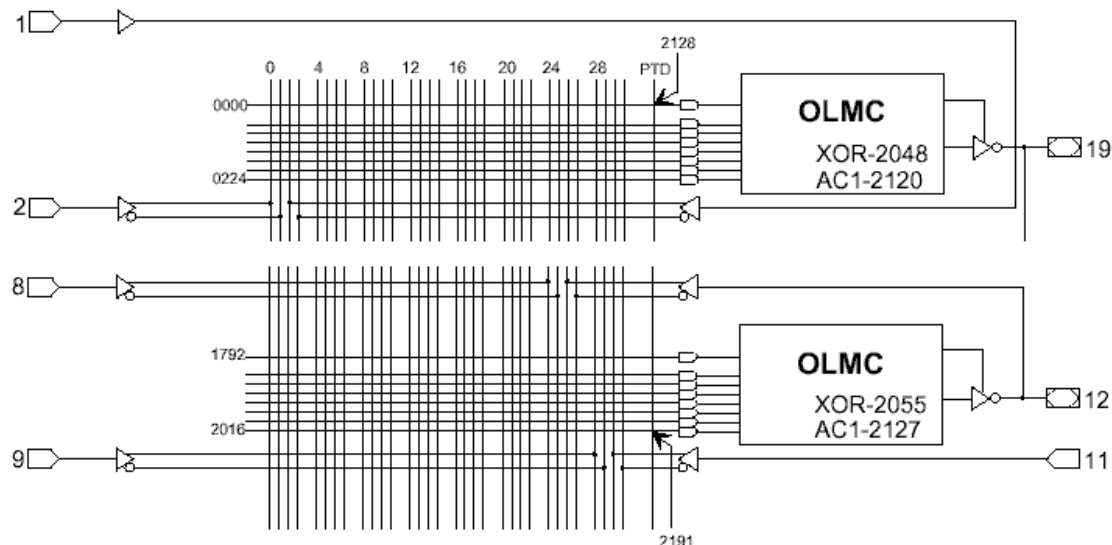


Rys. 4.3. Matryca połączeń logicznych w trybie rejestrowym układu GAL16V8

Każda komórka powinna zapewnić połączenie pomiędzy linią wejściową (sygnał normalny lub zanegowany) i końcówką wytwarzającą funkcje sygnału. Lina pozioma jest w stanie logicznym *true* jeżeli wszystkie linie „podłączane” do niej są w wysokim stanie. 64 linii *product term* jest zorganizowanych w osiem grup wyjściowych z ośmioma końcówkami (liniami) każda. Grupa jest przyporządkowana do danego układu OLMC. Siedem z ośmiu linii *product terms* w każdej grupie wyjściowej wchodzi do bramki OR układu OLMC, jedna służy do sterowania buforem trójstanowym.



Dwie linie wejściowe **pin 1** (CLK) i **pin 11** (OE) mają szczególne znaczenie. Pierwszy z nich w trybie rejestrowym (rys. 3.15) pełni funkcję sygnału zegarowego podawanego na wszystkie przerzutniki typu D układów OLMC. Sygnał OE w tym trybie steruje wyjściowymi buforami trójstanowymi. W pozostałych trybach pracy (rys. 4.4) te zaciski są normalnymi wejściami.



Rys. 4.4. Konfiguracja linii wejściowych CLK (pin 1) i OE (pin 2) w trybach prostym i złożonym układu GAL16V8

Konfiguracja **komórki OLMC** jest ustawiana przez programatory i oprogramowanie narzędziowe oraz jest całkowicie niewidoczna (przezroczysta) dla użytkownika. Komórkę OLMC można ustawić w trzech trybach:

- prostym (*simple*),
- złożonym (*complex*),
- rejestrowym (*registered*).

O konfiguracji wszystkich makrokomórek OLMC decydują dwa globalne bity SYN i AC0. Bit XOR każdej komórki ustawia polaryzację wyjścia w każdym z trzech trybów, natomiast bit AC1 ustala kierunek (wejście lub wyjście). Dwa globalne i 16 indywidualnych bitów definiuje wszystkie możliwe konfiguracje układu GAL16V8. Kompilatory ustawiają te bity automatycznie tak, iż użytkownik nie musi ich samodzielnie ustawiać. Niżej podana jest w tabeli 4.1 lista deklaracji układu GAL dla poszczególnych kompilatorów, za pomocą których można wybrać tryb pracy układu GAL (typ wyjścia komórki OLMC). W pierwszej kolumnie tabeli zawarto listę dostępnych kompilatorów.

Tabela 4.1. Lista deklaracji trybów pracy układu GAL16V8

	Registered	Complex	Simple	Auto Mode Select
ABEL	P16V8R	P16V8C	P16V8AS	P16V8
CUPL	G16V8MS	G16V8MA	G16V8AS	G16V8
LOG/iC	GAL16V8_R	GAL16V8_C7	GAL16V8_C8	GAL16V8
OrCAD-PLD	"Registered" <sup>1</sup>	"Complex" <sup>1</sup>	"Simple" <sup>1</sup>	GAL16V8A
PLDesigner	P16V8R <sup>2</sup>	P16V8C <sup>2</sup>	P16V8C <sup>2</sup>	P16V8A
TANGO-PLD	G16V8R	G16V8C	G16V8AS <sup>3</sup>	G16V8

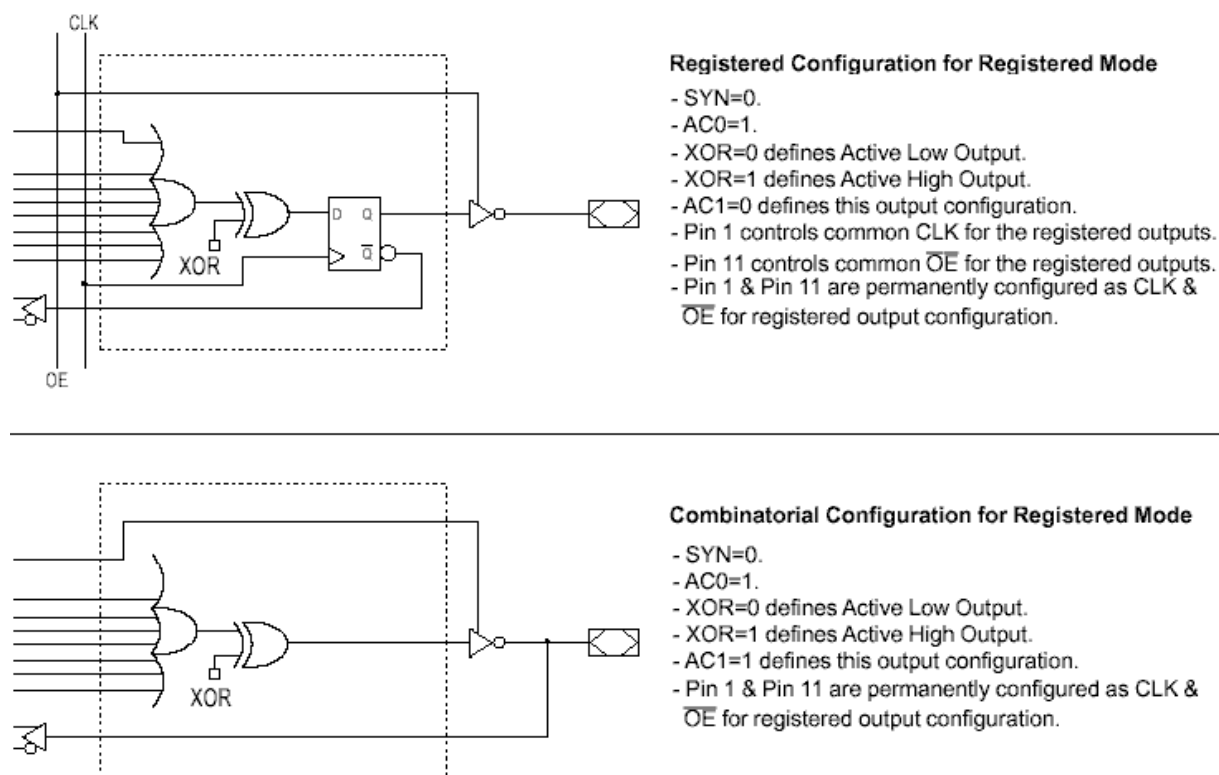
Większość kompilatorów ma możliwość automatycznego wyboru trybu pracy układu. Wybór jest dokonywany na podstawie kodu programu napisanego przez użytkownika. Jeżeli w kodzie zostanie użyty choćby jeden rejestr, to kompilator wybiera tryb rejestrowy. Gdy wyjścia kombinacyjne są kontrolowane przez linie *product term* to kompilator używa trybu

złożonego. Oprogramowanie wybierze tryb prosty jedynie wtedy, gdy wszystkie wyjścia są zadeklarowane jako kombinacyjne bez kontroli sygnałem OE.

Używając kompilatorów do konfiguracji układu należy zwrócić uwagę na następujące ograniczenia poszczególnych trybów:

- W trybie **rejestrowym** piny 1 i 11 są stale skonfigurowane odpowiednio jako zegar CLK i wejście OE. Nie mogą być one dedykowanymi wejściami.
- W trybie **złożonym** piny 1 i 11 stają się dedykowanymi wejściami i używają ścieżek sprzężenia zwrotnego (*feedback paths*) odpowiednio pinów 19 i 12. Z tego powodu piny 19 i 12 nie mają opcji sprzężenia w tym trybie, są wyłącznie wyjściami.
- W trybie **prostym** wszystkie ścieżki sprzężenia zwrotnego pinów wyjściowych są poprowadzone przez przyległe piny. Z tego powodu dwa wewnętrzne piny (15 i 16) nie mają opcji sprzężenia zwrotnego i są zawsze skonfigurowane jako dedykowane kombinacyjne wyjścia.

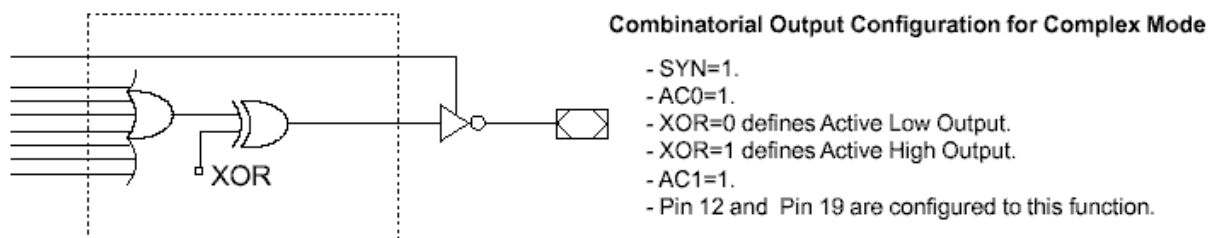
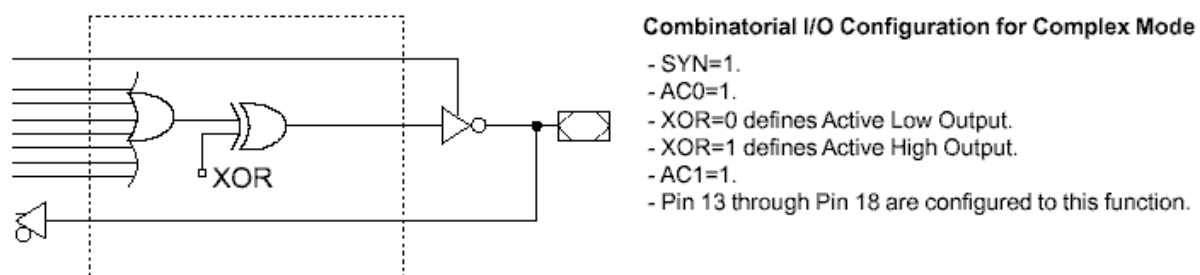
W trybie **rejestrowym** makrokomórki OLMC (rys. 4.5) są skonfigurowane jako dedykowane wyjścia rejestrowe lub jako funkcje wejściowo-wyjściowe (*I/O functions*). Wszystkie makrokomórki dzielą wspólny zegar CLK i pin kontroli OE. Każda makrokomórka może być skonfigurowana jako rejestrowa lub wejście albo wyjście kombinacyjne. Dedykowana funkcja wejścia lub wyjścia może być zaimplementowana jako podzestaw funkcji wejściowo-wyjściowej (*I/O function*).



Rys. 4.5. Konfiguracja makrokomórki OLMC dla trybu rejestrowego

W trybie **złożonym** makrokomórki są skonfigurowane tylko jako wyjścia lub jako funkcje wejściowo-wyjściowe (*I/O functions*) (rys. 4.6). W tym trybie można uzyskać maksymalnie sześć linii wejściowo-wyjściowych. Dwie zewnętrzne makrokomórki (piny 12 i 19) nie mają możliwości pracy jako wejście (*input*). Projekty wymagające ośmiu I/O mogą być zaimplementowane w trybie rejestrowym.

Do wszystkich makrokomórek dochodzi siedem linii *product term*. Ósma linia *product term* jest używana do sterowania buforem trójstanowym. Piny 1 i 11 są zawsze dostępne jako wejścia danych dla matrycy iloczynu logicznego.



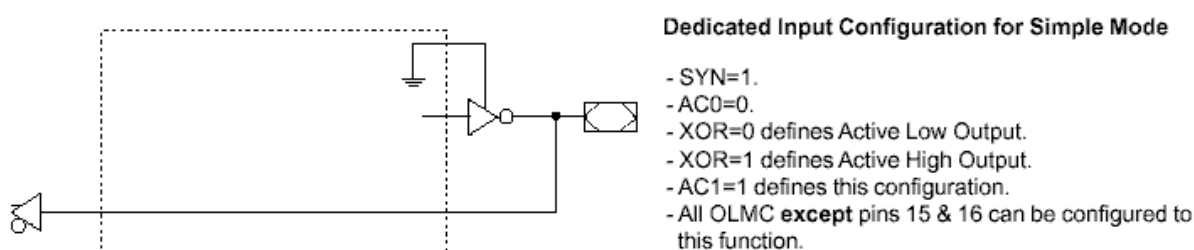
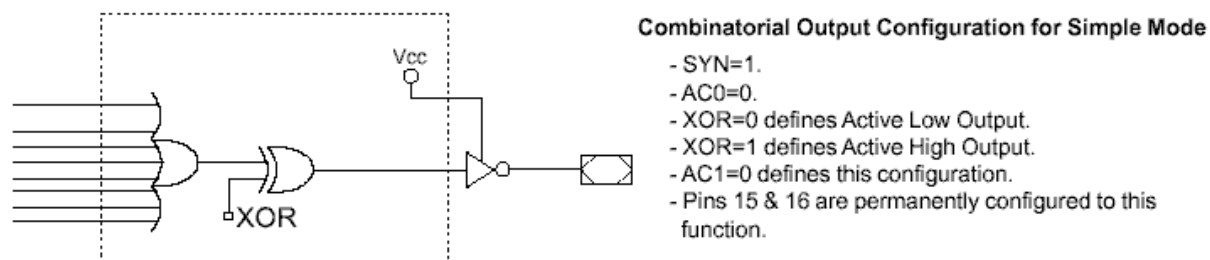
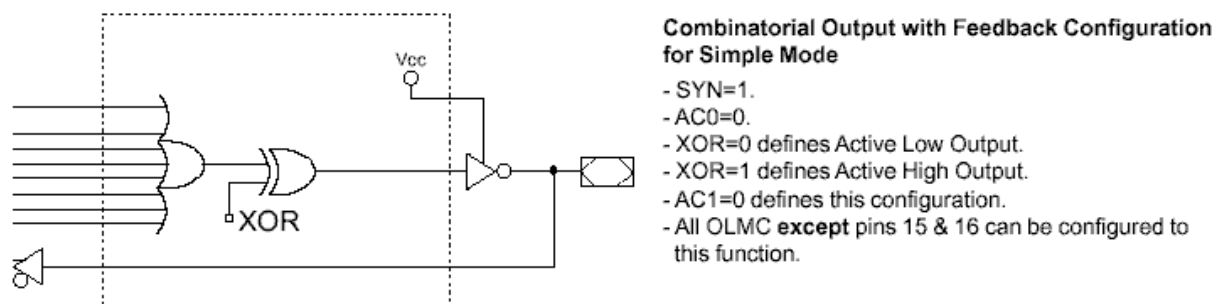
Rys. 4.6. Konfiguracja makrokomórki OLMC dla trybu złożonego

W **trybie prostym** makrokomórki są skonfigurowane jako dedykowane wejścia lub jako dedykowane, zawsze aktywne, kombinacyjne wyjścia (rys. 4.7). Do makrokomórki OLMC dochodzi osiem linii *product term* kontrolujących jej logikę. Dodatkowo każde wyjście ma programowalną polaryzację. Piny 1 i 11 są zawsze dostępne jako wejścia danych matrycy iloczynów logicznych. Dwie środkowe makrokomórki (piny 15 i 16) są zawsze dostępne jako dedykowane wyjścia.

Każdy układ GAL16V8 posiada **elektroniczny podpis**. Składa się on z 64 bitów zawartych w reprogramowalnej pamięci. Służy on do przechowywania danych użytkownika. Niektórzy użytkownicy zapisują w niej kody identyfikacyjne, numery wersji, numer kontrolne. Zawartość podpisu jest zawsze dostępna dla użytkownika niezależnie od stanu komórki ochrony danych (zawartość elektronicznego podpisu jest wliczana do obliczania sumy kontrolnej. Zmiana elektronicznego podpisu spowoduje zmianę sumy kontrolnej).

W układach GAL16V8 zaimplementowano **komórkę ochrony danych** (*Security Cell*), służącą do blokady odczytu zaprogramowanej struktury. Raz zaprogramowana komórka zapobiega odczytowi bitów konfiguracyjnych układu. Może być ona skasowana tylko przez przeprogramowanie całego układu, czyli oryginalna konfiguracja nigdy nie będzie odczytana. Elektroniczny podpis jest zawsze możliwy do odczytania niezależnie od stanu komórki ochrony danych.

Specjalne obwody wewnątrz układu GAL16V8 zapewniają po włączeniu zasilania sygnał reset wszystkim rejestrom. Rejestry ustawiają swoje wyjścia Q w stan niski po określonym czasie (maksymalnie 1µs), wynikiem tego stan pinów wyjść rejestrowych (jeśli są włączone) będzie zawsze „wysoki” po włączeniu zasilania, niezależnie od zaprogramowanej polaryzacji końcówek wyjściowych.



Rys. 4.7. Konfiguracja makrokomórki OLMC dla trybu prostego

## 4.2. Układy logiczne CPLD na przykładzie rodziny układów XC9500 firmy Xilinx

Układy logiczne CPLD zostaną przedstawione na przykładzie rodziny układów XC9500 firmy Xilinx. Są one programowalne i testowalne w docelowym mse. Rodzina ta charakteryzuje się następującymi właściwościami:

- duża szybkość działania (5ns opóźnienia pomiędzy pinami,  $f_{CNT}$  do 125 MHz),
- duża gęstość upakowania (od 36 do 288 makrokomórek z 800 do 6,400 użytecznymi bramkami),
- układy programowalne w systemie o napięciu zasilania 5V (możliwość wykonania 10000 cykli programowania/kasowania),
- układy składają się z „elastycznych” bloków funkcyjnych (odpowiadających układom GAL typu 36V18), 90 linii *product terms* sterujących pojedynczymi lub wszystkimi 18 makrokomórkami wewnątrz bloku funkcyjnego,
- posiadają interfejs standardu IEEE 1149.1 (JTAG),
- zapewniają programowalny tryb redukcji mocy dla każdej komórki,
- wyjścia układów przewodzą prąd do 24mA,
- piny wejściowo-wyjściowe mogą być ustawione na standard 3,3V lub 5V.

W tabeli 4.2 zestawiono układy rodziny XC9500. Jak widać gęstość upakowania używanych bramek logicznych mieści się w przedziale od 800 do ponad 6400, a rejestrów od 36 do 288.

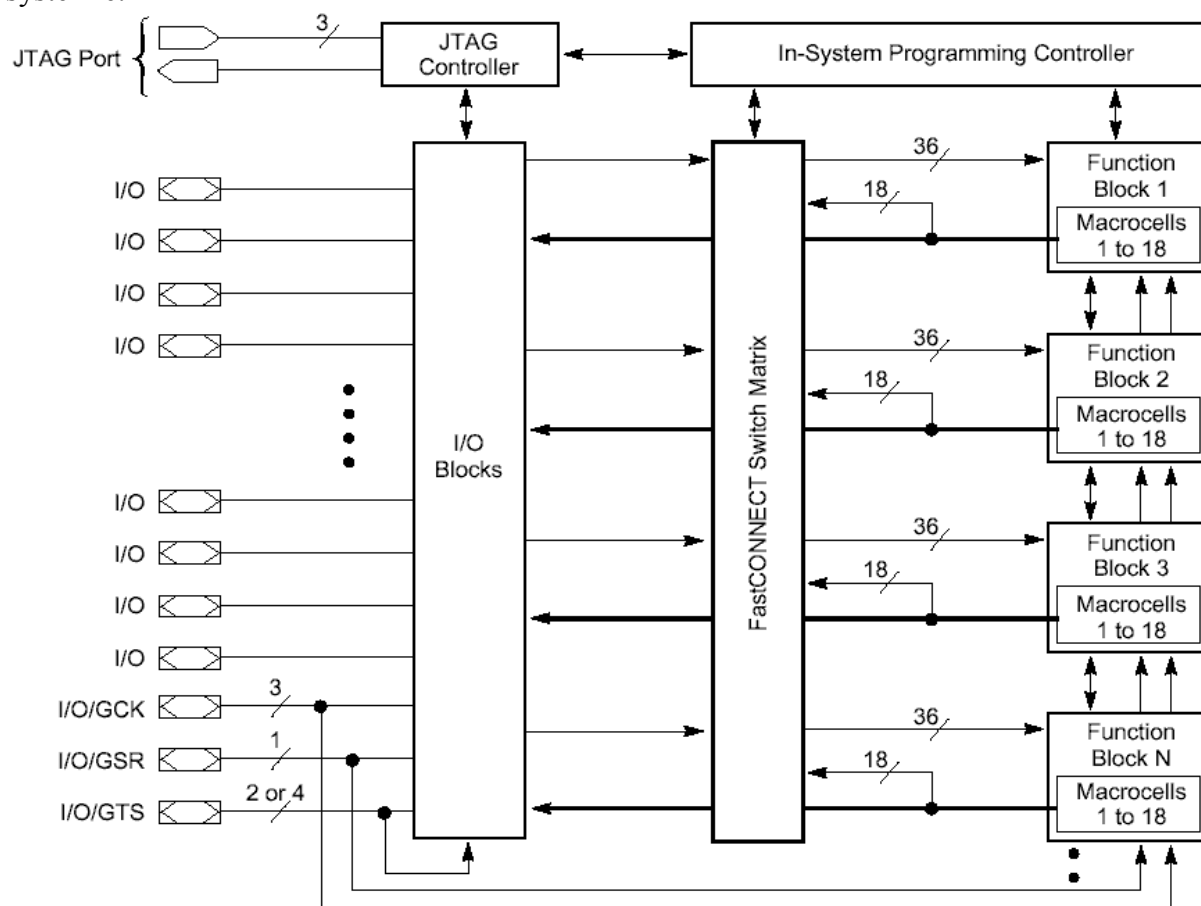
Tabela 4.2. Rodzina układów XC9500

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocells	36	72	108	144	216	288
Usable Gates	800	1,600	2,400	3,200	4,800	6,400
Registers	36	72	108	144	216	288
$t_{PD}$ (ns)	5	7.5	7.5	7.5	10	10
$t_{SU}$ (ns)	3.5	4.5	4.5	4.5	6.0	6.0
$t_{CO}$ (ns)	4.0	4.5	4.5	4.5	6.0	6.0
$f_{CNT}$ (MHz)	100	125	125	125	111.1	111.1
$f_{SYSTEM}$ (MHz)	100	83.3	83.3	83.3	66.7	66.7

**Note:**  $f_{CNT}$  = Operating frequency for 16-bit counters

$f_{SYSTEM}$  = Internal operating frequency for general purpose system designs spanning multiple FBs.

Na potrzeby programowania i testowania układów w systemie rozszerzono listę instrukcji sterujących interfejsem **JTAG**. Zatem interfejs ten pozwala nie tylko na testowanie zgodnie ze standardem IEEE 1149.1, ale również i programowanie układów zamontowanych już w systemie.



Rys. 4.8. Architektura układów rodziny XC9500

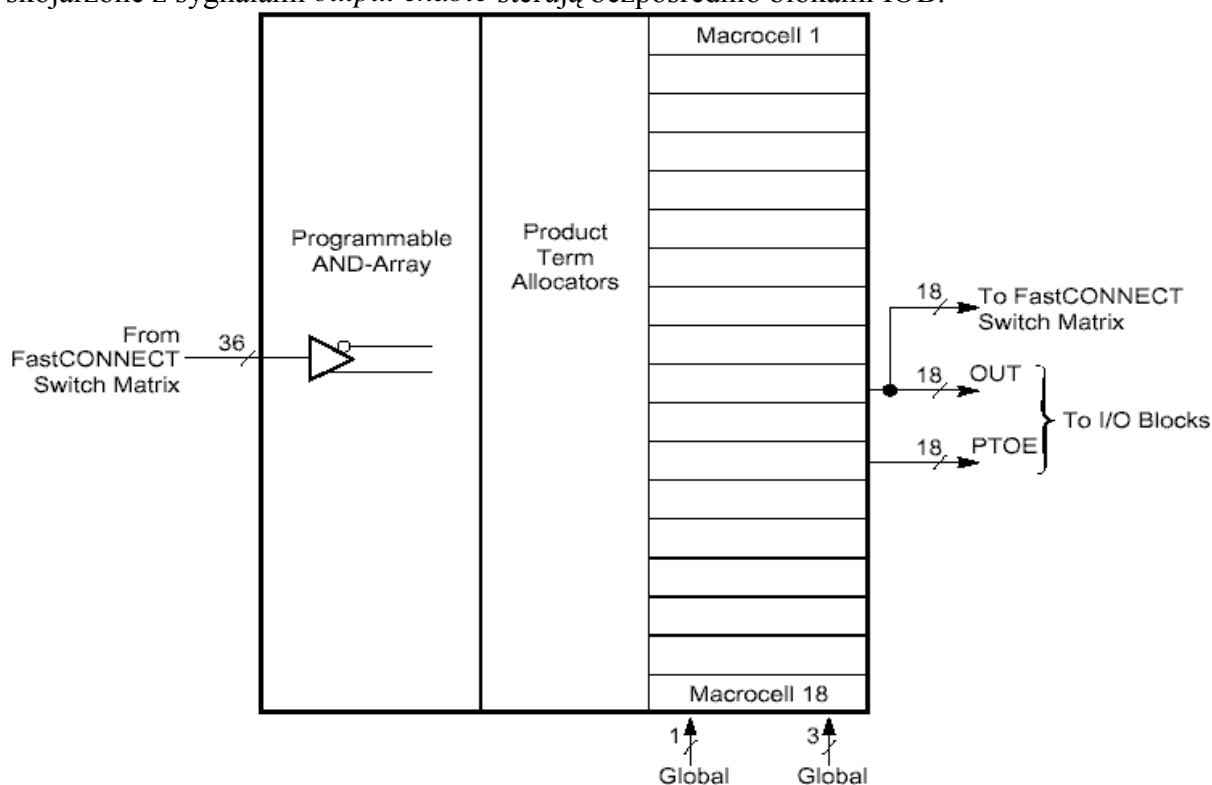
Możliwość przeprowadzania wielokrotnej liczby cykli programowania/kasowania zapewnia dużą swobodę w zmienianiu wewnętrznej konfiguracji układów, czy uaktualnianiu zawartych w nich projektów. Dodatkowo zapewniono kontrolę szybkości narastania napięć

wyjściowych oraz umożliwiono programowe uziemienie pinów w celu lepszej redukcji szumów. Urządzenia wejścia/wyjścia mogą być konfigurowane dla napięć 3,3V oraz 5V. Wszystkie wyjścia przewodzą prąd do 24mA.

Każdy układ XC9500 jest podsystemem zawierającym **bloki funkcyjne** (*FB – Function Block*) i **bloki wejścia/wyjścia** (*IOB – I/O Block*), które są łączone między sobą za pomocą **matrycy przełączającej** (*FastCONNECT switch matrix*) (rys. 4.8).

Bloki **IOB** między innymi buforują sygnały wejściowe i wyjściowe z układu oraz zapewniają odpowiednie parametry elektryczne zacisków.

Każdy blok funkcyjny **FB** daje możliwość zaprogramowania 36 wejść i 18 wyjść. Matryca przełączająca łączy wszystkie wyjścia bloku FB z wejściami innego bloku FB. Dla każdego bloku FB wyjścia w liczbie od 12 do 18 (w zależności od liczby wyprowadzeń obudowy) skojarzone z sygnałami *output enable* sterują bezpośrednio blokami IOB.

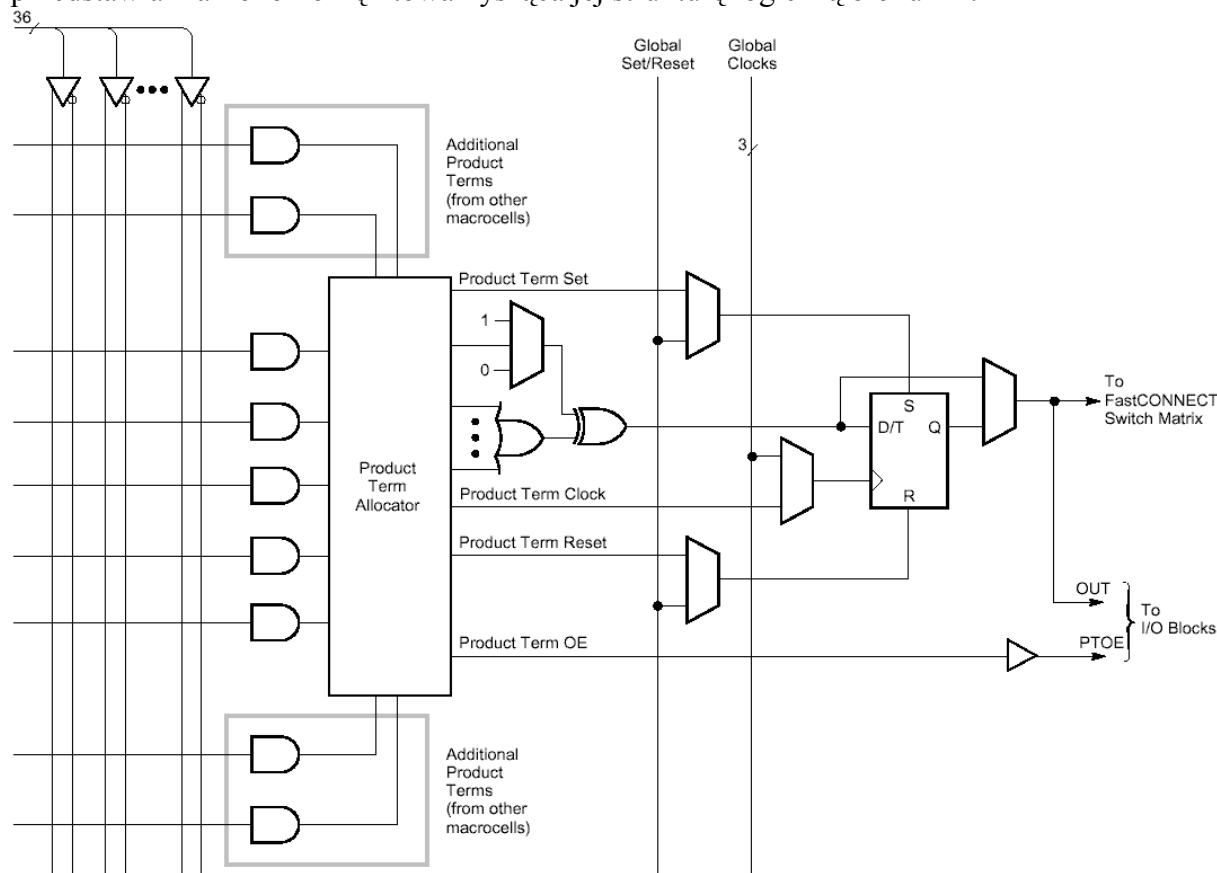


Rys. 4.9. Blok funkcyjny (FB) układów rodziny XC9500

Każdy **blok funkcyjny**, jak przedstawiono na rys. 4.9, składa się z 18 niezależnych makrokomórek, z których każda może realizować funkcję kombinacyjną bądź rejestrową. Do bloku funkcyjnego doprowadzony jest sygnał zegara oraz sygnały *set/reset*. Blok funkcyjny poprzez generację stanów na 18 wyjściach steruje matrycą przełączającą. Wyjścia te wraz z sygnałami *output enable* sterują blokami IOB.

Do programowalnej **matrycy iloczynów logicznych** dochodzą 72 sygnały (36 normalnych sygnałów i 36 ich negacji). Sygnały te są przez nią łączone do 90 linii *product term* dochodzących do 18 makrokomórek. Wewnątrz każdego bloku FB możliwe jest zrealizowanie połączeń (ścieżek) z wyjść makrokomórek do matrycy iloczynów logicznych, bez potrzeby wyprowadzania ich poza dany blok FB. Ścieżki te wykorzystywane są do tworzenia bardzo szybkich liczników oraz układów stanów, gdzie wszystkie rejestry stanu są wewnątrz tego samego bloku FB.

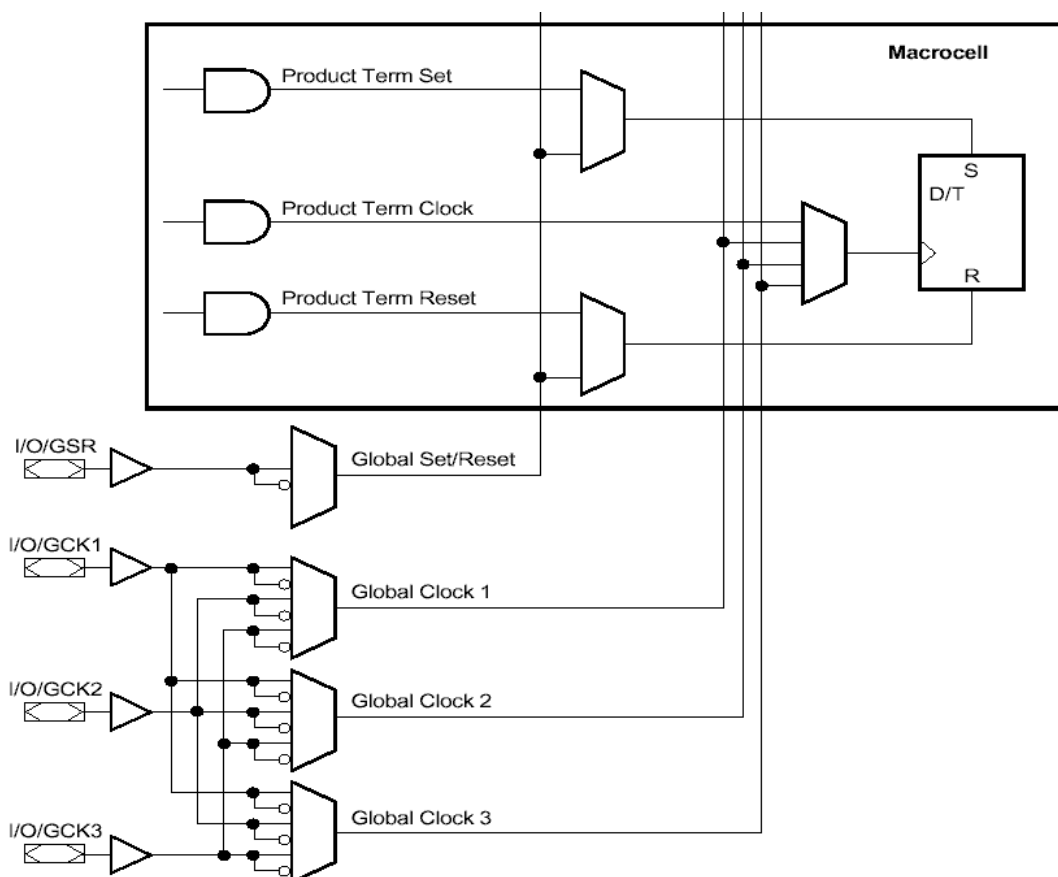
Każda **makrokomórka** w bloku FB układu XC9500 może być indywidualnie skonfigurowana tak, aby realizować funkcje kombinacyjne lub rejestrowe. Rys. 4.10 przedstawia makrokomórkę i towarzyszącą jej strukturę logiczną bloku FB.



Rys. 4.10. Makrokomórka bloku FB układów rodziny XC9500

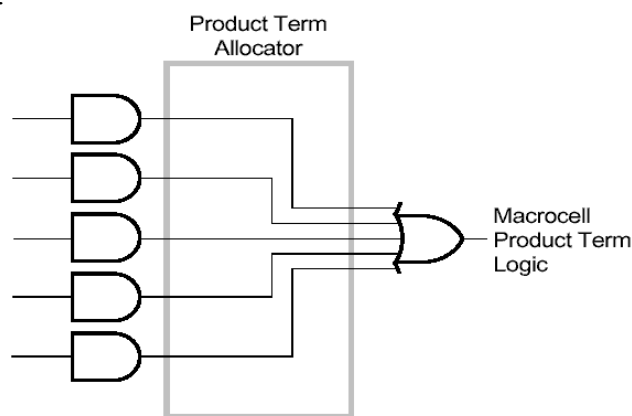
Pięciu linii *product terms* bezpośrednio wyprowadzonych z matrycy iloczynów logicznych można użyć jako podstawowych wejść danych (do bramek OR i XOR). Pozwalają one na implementację funkcji kombinacyjnej lub mogą posłużyć jako wejścia sterujące włączając w to sygnały zegara, *set/reset* i *output enable*. Blok **PTA** (*Product Term Allocator*), zawarty w każdej makrokomórce, służy do wyboru sposobu użycia tych pięciu linii. Rejestr makrokomórki może być skonfigurowany jako przerzutnik D lub T, lub też może być przeznaczony do operacji kombinacyjnych. Każdy rejestr wyposażony jest w dwie asynchroniczne operacje ustawiania (*set*) i zerowania (*reset*). Podczas włączania zasilania wszystkie wykorzystywane rejestry są inicjalizowane do stanu zdefiniowanego przez użytkownika (w przypadku braku specyfikacji domyślnie ustawiane jest 0).

Wszystkie globalne sygnały takie jak sygnały zegara, *set/reset* oraz *output enable* są dostępne dla każdej makrokomórki. Sygnał zegarowy dochodzący do rejestru makrokomórki może być jednym z trzech globalnych sygnałów zegara lub sygnałem zegara pochodzącym z linii *product term*, co przedstawiono na rys. 4.11. Dodatkowo istnieje możliwość wyboru aktywnego zbocza sygnału zegarowego. Poprzez wejście GSR możliwe jest ustawienie rejestru użytkownika do zdefiniowanego przez niego stanu.



Rys. 4.11. Sposób wyboru sygnałów zegarowych set/reset w makrokomórce bloku FB układów rodziny XC9500

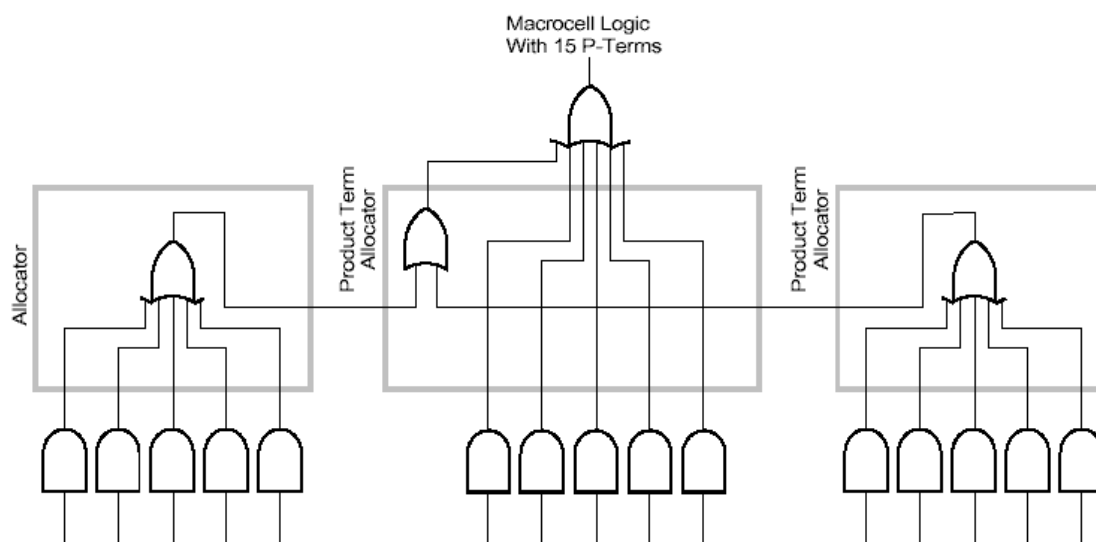
Blok **PTA** makrokomórki steruje wykorzystaniem pięciu bezpośrednich linii *product terms* dochodzących do niej. Np. wszystkie pięć linii może być dołączonych do bramki OR, jak pokazano na rys. 4.12.



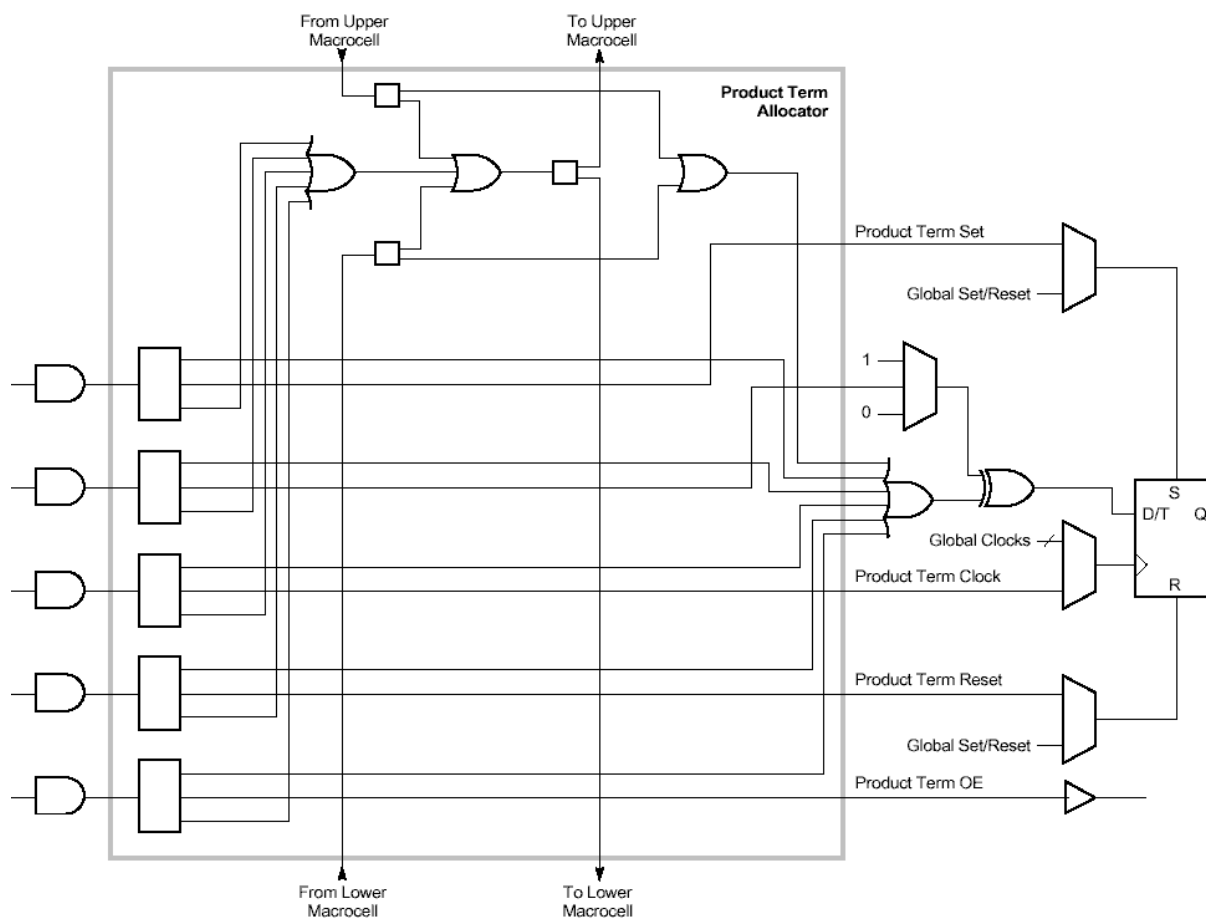
Rys. 4.12. Wykorzystanie bezpośrednich linii *product terms* w makrokomórce

Blok PTA może dokonywać zmian wewnątrz FB i przydzielać makrokomórce dodatkowe linie, oprócz pięciu podstawowych dochodzących już do niej. Każda makrokomórka wymagająca dodatkowej linii *product term* może skorzystać z niewykorzystanej linii innej makrokomórki wewnątrz bloku FB. Pojedyncza makrokomórka może użyć 15 linii *product terms*, przy czym należy uwzględnić dodatkowe opóźnienia sygnałów na dodanych liniach, wynikające z wydłużenia się ich drogi wewnątrz bloku FB (rys. 4.13).



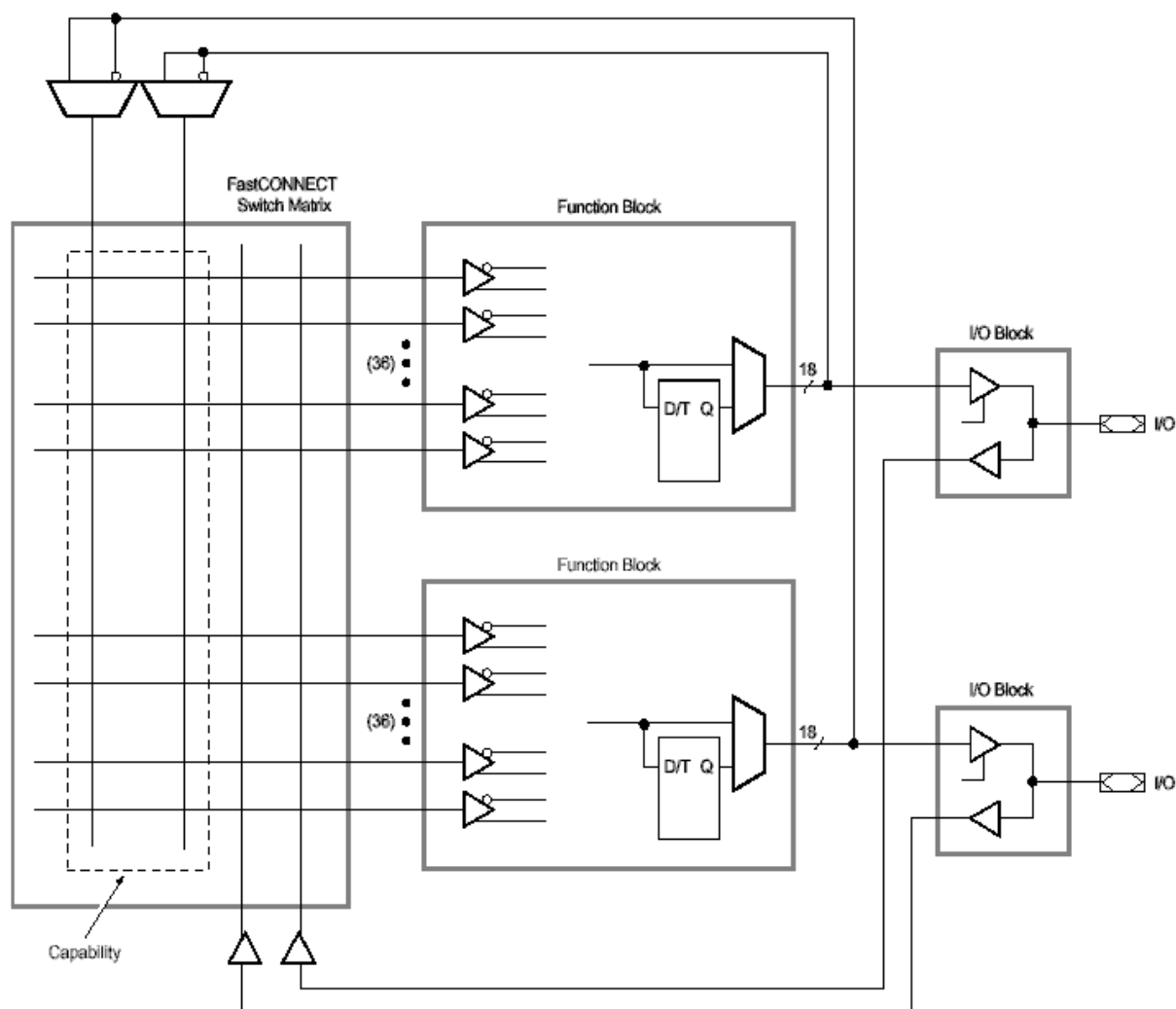
Rys. 4.13. Blok PTA z 15 liniami *product terms*

Elastyczność ta wynika z budowy bloku PTA pokazanej na rys. 4.14.



Rys. 4.14. Schemat logiczny bloku PTA

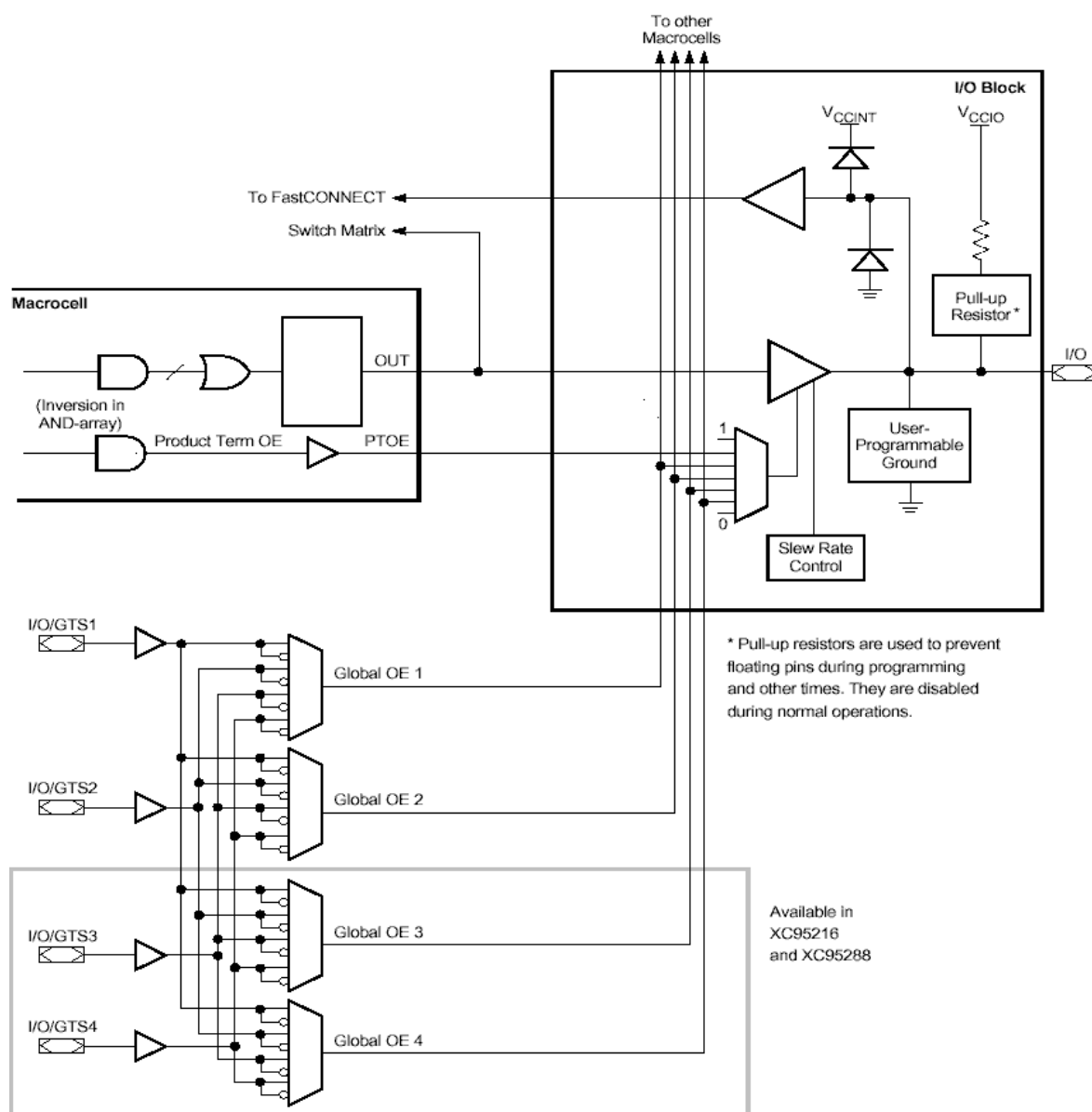
**Matryca przełączająca** dostarcza sygnały z bloków IOB i FB do wejść bloków FB (rys. 4.15). Wszystkie wyjścia bloków FB i wejścia/wyjścia bloków IOB są podawane na tę matrycę. Za pomocą matrycy użytkownik wybiera sygnały, które mają dochodzić do danego bloku FB. Dodatkowo realizuje ona iloczyn logiczny na drucie co zwiększa funkcjonalność całego układu.



Rys. 4.15. Schemat logiczny matrycy przełączającej (*FastCONNECT Switch Matrix*)

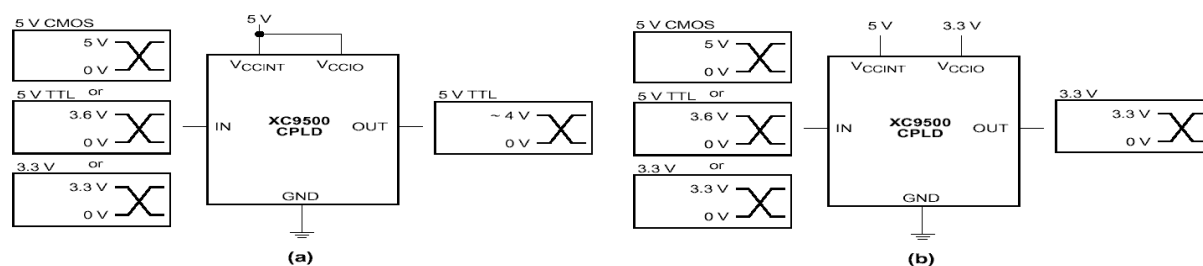
Bloki wejścia/wyjścia IOB stanowią interfejs pomiędzy wewnętrzną logiką, a pinami wejścia/wyjścia układu. Każdy blok zawiera bufor wejściowy, sterownik wyjściowy, multiplexer wyboru *output enable* oraz programowalne uziemienie (rys. 4.16). Bufor wejściowy jest kompatybilny ze standardami 5V CMOS, 5V TTL i poziomami sygnału 3,3V. Bufor wejściowy używa wewnętrznego 5V źródła zasilania ( $V_{CCINT}$ ) w celu zapewnienia stałych progów wejścia i eliminacji wahań napięcia  $V_{CCIO}$ . Sygnał *output enable* może być generowany w następujący sposób, jako: sygnał *product term* z makrokomórki, dowolny sygnał zegara globalnego, zawsze „1” lub zawsze „0”. Występują dwa globalne sygnały *output enable* dla układów do 144 makrokomórek oraz cztery sygnały globalne *output enable* dla układów z 180 lub większą liczbą makrokomórek.

Każde wyjście umożliwia kontrolę szybkości narastania napięcia wyjściowego. Szybkość narastania sygnału wyjściowego może być programowo spowalniana w celu redukcji szumów. Każdy blok IOB zapewnia programową obsługę pinu uziemienia. Pozwala to na dodatkową konfigurację pinów układów wejścia/wyjścia jako pinów uziemienia. Dzięki temu można znacznie zredukować poziom szumów. Rezystory podciągające (typowa wartość 10k $\Omega$ ) są dołączone do każdego układu wejścia/wyjścia w celu zabezpieczenia go przed stanem nieokreślonym, w przypadku niewykorzystania tego układu przez projekt użytkownika. Rezystory te są aktywne w trybie programowania oraz kasowania układu, a także w momencie włączania zasilania systemu.



Rys. 4.16. Schemat logiczny bloku IOB wraz z układami sterującymi sygnałami *output enable*

Rezystory są de aktywowane podczas normalnych operacji. Sterownik wyjściowy jest zdolny do przewodzenia prądu 24mA. Wszystkie sterowniki wyjściowe w układzie mogą być skonfigurowane dla poziomu 5V lub poziomu 3,3V przez połączenie wyjściowego pinu zasilania ( $V_{CCIO}$ ) do 5V lub do źródła zasilania 3,3V. Rys. 4.17 pokazuje jak układ XC9500 może być zasilany tylko napięciem 5V lub napięciem 5V i 3,3V.



Rys. 4.17. Sposoby zasilania układu rodziny XC9500: a) jedno źródło zasilania 5V, b) dwa źródła zasilania 5V i 3,3V

Programowanie i testowanie układów rodziny XC9500 jest realizowane za pomocą **interfejsu standardu IEEE 1149.1 (JTAG)**. Interfejs ten posiada następującą listę instrukcji: EXTES, SAMPLE/PRELOAD, BYPASS, USER-CODE, INTEST, IDCODE, HIGHZ. Dodatkowo na potrzeby programowania układów w systemie (ISP) dodano 5 instrukcji: ISPEN, FERAASE, FPGM, FVFY, ISPEX. Stanowią one rozszerzenie zestawu instrukcji 1149.1. Zgodnie ze specyfikacją standardu IEEE 1149.1 piny TMS i TCK układu interfejsu JTAG są podwieszone poprzez rezystory pull-up do zasilania.

Układy XC9500 zawierają zaawansowane sposoby ochrony danych, które całkowicie zabezpieczają program przed nieautoryzowanym czytaniem lub pomyłkowym skasowaniem, czy przeprogramowaniem. Użytkownik może ustawić odpowiednie bity w celu ochrony kodu programu zawartego w układzie przed jego przeczytaniem. Skasowanie w całości programu jest jedynym sposobem na wyzerowanie bitów zabezpieczających, tym samym na odblokowanie odczytu z układu.

Wszystkie układy XC9500 oferują możliwość ustawienia trybu niskiego poboru mocy dla każdej makrokomórki z osobna lub dla wszystkich jednocześnie. Zatem ważne dla danej aplikacji części układu mogą pozostać w standardowym trybie poboru mocy, podczas gdy pozostałe mogą zostać zaprogramowane na operacje przy niskim poborze mocy.

W czasie włączania zasilania (*power-up time*) układy XC9500 są w stanie uśpienia dopóki napięcie zasilające  $V_{CCINT}$  nie osiągnie bezpiecznego poziomu (około 3,8 V). Do tego czasu wszystkie piny układu oraz interfejsu JTAG są wyłączone i podwieszone do zasilania za pomocą rezystorów pull-up. Kiedy napięcie zasilania osiągnie bezpieczny poziom wszystkie rejestry użytkownika są inicjalizowane (typowo w czasie 100µs dla 9536 - 95144, 200µs dla 95216, 300µs dla 95288) po czym układ jest gotowy do pracy.

### 4.3. Język ABEL opisu układów realizowalnych w strukturach programowalnych

Podczas projektowania układu zazwyczaj wiemy, jakie są sygnały wejściowe do przetwarzania w układzie oraz jakie sygnały wyjściowe powinny wystąpić na wyjściach układu. W większości języków służących do opisu układów realizowalnych w strukturach programowalnych wejścia i wyjścia są nazywane **portami** układu. W każdym języku opis układu składa się z dwóch bloków: tzw. **bloku deklaracji** zawierającego przypisania sygnałów, identyfikowanych nazwami, do odpowiednich portów oraz z **bloku opisu budowy i działania** układu.

Język **ABEL** (*Advanced Boolean Expression Language*), to jeden z najstarszych języków opisu układów implementowanych w strukturach programowalnych. Opracowany został przez firmę DATA I/O. Opis układu w języku ABEL ma postać modułu tekstowego, lub modułów połączonych w strukturę hierarchiczną. Język ABEL daje możliwość opisu zachowania systemu na wiele sposobów, włączając równania logiczne, tablice prawdy i opis stanowy, podobny w zapisie do warunków języka C. Kompilatory do języka ABEL dają możliwość projektowania i programowania struktur PLD takich jak: SPLD, CPLD i FPGA.

#### 4.3.1. Podstawowa struktura pliku źródłowego w języku ABEL

Plik źródłowy w języku ABEL składa się z następujących części:

- nagłówek: załączone moduły, opcje i tytuł,
- opis logiczny: równania, tablica prawdy, opis stanowy,

- wektory testujące,
- koniec.

**Uwaga:** dla słów kluczowych języka ABEL wielkość liter nie jest istotna. Natomiast nazwy nadane przez użytkownika i etykiety są **identyfikowane również po wielkości liter** (np. input1 i Input1 są różnymi nazwami bądź etykietami).

W tabeli 4.3 pokazano listing opisu źródłowego w języku ABEL.

Tabela 4.3. Listing opisu źródłowego w języku ABEL

Komentarz	"Struktura pliku źródłowego " w języku ABEL
Nagłówek	<b>module</b> nazwa_modułu <b>title</b> 'tytuł projektu' deviceID <b>device</b> deviceType;
Deklaracje sygnałów wejściowych i wyjściowych, stałych zmiennych pomocniczych w postaci wektorów	<b>declarations</b> sygnały_we <b>pin</b> ; sygnały_wy <b>pin istype 'com'</b> ; nazwa = <b>.c.</b> ; nazwa = [nazwa1, nazwa2,...];
Opis logiczny układu	<b>equations</b> równania, lub/i <b>truth_table</b> tablice wartości, przejść lub/i <b>state_diagram</b> opis tekstowy grafu przejść
Wektory testowe	<b>test_vectors</b> wartości sygnałów wejściowych -> spodziewane odpowiedzi
Zakończenie opisu modułu	<b>end</b> nazwa_modułu

Plik źródłowy musi zaczynać się **słowem kluczowym module**, a **kończyć słowem end**. W nagłówku opisu źródłowego występuje nazwa modułu po słowie kluczowym module. Nazwa modułu określa również nazwę plików wyjściowych tworzonych przez oprogramowanie przetwarzające systemu syntezy (kompilator, program optymalizacji, symulator, fitter, itd.). Tytuł projektu wprowadzony po słowie **title** umożliwia łatwą identyfikację plików związanych z danym projektem.

Kolejnym blokiem jest **blok deklaracji**, w którym deklarowane są nazwy (identyfikatory) sygnałów wejściowych (**pin**) oraz wyjściowych (**pin istype 'typ\_końcówki'**). Można również wskazać (opcjonalnie) numery pinów wejściowych i wyjściowych układu programowalnego, który zamierzamy zastosować (zadeklarowanego słowem **device**). W tym bloku deklarowane są również nazwy sygnałów wewnętrznych, pomocnicze zmienne, stałe itp.

Kolejnym blokiem jest **blok opisu logicznego**, w którym definiuje się właściwości logiczne projektowanego układu. Możliwy jest opis działania układu za pomocą:

- **równań**, po słowie kluczowym **equations**, z operatorami logicznymi, ale także równań z operatorami arytmetycznymi i operacjami relacji,
- **tablicy wartości funkcji** (dla układów kombinacyjnych) lub **tablicy przejść** (dla układów sekwencyjnych) po słowie kluczowym **truth\_table**,

- tekstowego opisu **grafu przejść** po słowie kluczowym `state_diagram`

Opcjonalnym blokiem w pliku źródłowym jest **blok wektorów testowych** przygotowanych w postaci tablicy, w której wpisuje się arbitralnie wybrane kombinacje sygnałów wejściowych (wektory pobudzeń) i oczekiwane dla nich wartości sygnałów wyjściowych (odpowiedzi na wyjściu układu). Wektory testowe są wykorzystywane przez program symulacji funkcjonalnej (poprawności opisu logicznego) lub program symulacji układu po jego implementacji w wybranej strukturze programowalnej, a więc już po przygotowaniu wynikowego pliku konfiguracyjnego.

#### 4.3.2. Słowa kluczowe języka ABEL

**Module:** każdy plik źródłowy rozpoczyna się konstrukcją **module** a zaraz za nią występuje nazwa modułu (identyfikator). Duże źródła składają się często z wielu modułów z ich własnymi tytułami, równaniami, itd.

**Title:** jest opcjonalny i może być używany do identyfikacji projektu. Tytuł musi być umieszczony w pojedynczych cudzysłowach. Linia **Title** jest ignorowana przez kompilator, ale jest bardzo przydatna przy tworzeniu dokumentacji.

**String:** tekst jest łańcuchem znaków ASCII umieszczonym w pojedynczych cudzysłowach. Strings są używane do wpisania nazw Tytułu(Title), Opcji(Options), oraz nazw końcówek układu, węzłów i deklaracji atrybutów.

**device:** ta deklaracja jest opcjonalna i stowarzysza identyfikator układu z konkretną programowalną strukturą logiczną. Deklaracja **device** musi być zakończona średnikiem. Np. MY\_DECODER device 'XC4003E';

**comments:** komentarze mogą być umieszczane w dowolnym miejscu pliku. Rozpoczynają się podwójnym cudzysłowem, a kończą końcem linii, lub podwójnym cudzysłowem, w zależności od tego co będzie pierwsze.

**pin:** deklaracja ta służy to poinformowania kompilatora, które nazwy użytkownika są powiązane z końcówkami urządzenia. Zapis:

```
[!]pin_id pin [pin#] [istype 'attributes'];
```

Jedna deklaracja **pin** może opisywać więcej końcówek:

```
[!]pin_id , pin_id, pin_id pin [pin#, [pin#, [pin#]]] [istype 'attributes'];
```

Przykład 4.1:

```
IN1, IN2, A1 pin 2, 3, 4;
OUT1 pin 9 istype 'reg';
ENABLE pin;
!Chip_select pin 12 istype 'com';
!S0..!S6 pin istype 'com';
```

Nie jest konieczne do określenia końcówek układu. Numery końcówek mogą być określone podczas kompilacji używając np. Xilinx CAD. Jest to bardzo dobra własność dająca możliwość tworzenia projektów bardziej ogólnych i elastycznych.

Znak ! opisuje końcówkę jako aktywną stanem niskim (sygnał będzie negowany).

Atrybut `istype` jest powiązaniem dającym możliwość wybrania trybu pracy końcówki, takich jak 'com' wyjście wówczas jest kombinacyjne, lub 'reg' tryb rejestrowy taktowany zegarem. Ten atrybut jest tylko dla wyjścia.

**node:** węzeł jest deklaracją wewnętrznego sygnału, który nie ma połączenia z końcówką układu, ale ma taki sam format jak deklaracja **pin**. Np.:

```
tmp1 node [istype 'com'];
```

**Inne deklaracje:** pozwalają na definicje stałych, zmiennych makr i wyrażeń, które mogą uprościć program. Przykładowa deklaracja stałej ma postać: `id [, id],... = expr [, expr];`

Przykład 4.2:

```
A = 21;
C=2*7;
ADDR = [1,0,11];
LARGE = B & C;
D = [D3, D2, D1, D0];
D = [D3..D0];
```

Dwa ostatnie równania są sobie równoważne. Użycie „..” jest bardzo przydatne do określenia zakresu. Przedstawiają one zapis wektora, za każdym odwołaniem do D będzie to równoznaczne odwołaniu do wektora [D3, D2, D1, D0].

### 4.3.3. Liczby

Liczby mogą być wprowadzane w czterech różnych systemach: binarnym, ósemkowym, dziesiętnym i szesnastkowym. Systemem domyślnym jest dziesiętny. Istnieje możliwość zmiany bazy domyślnej przy użyciu dyrektywy **Radix**

Tabela 4.4. Systemy liczb w języku ABEL

Nazwa systemu	Podstawa	Symbol
Binarny	2	<sup>^</sup> b
Ósemkowy	8	<sup>^</sup> o
Dziesiętny	10	<sup>^</sup> d (domyślny)
Szesnastkowy	16	<sup>^</sup> h

### 4.3.4. Dyrektywy

Dyrektywy pozwalają na zaawansowane zarządzanie plikiem źródłowym i mogą być używane w każdym miejscu pliku, gdy tylko zajdzie taka potrzeba.

#### @ALTERNATE

Zapis: `@alternate`

`@ALTERNATE` włącza alternatywny zestaw operatorów. Użycie operatora `@alternate` wyklucza użycie operatorów ABEL-HDL dodawania(+), mnożenia(\*) i dzielenia(/), ponieważ reprezentują one operatory logiczne AND i NOT w trybie `@alternate`. Operatory standardowe działają podczas trybu `@alternate`.

Powrót do trybu normalnego odbywa się przez dyrektywę `@STANDARD`, lub koniec modułu.

#### @RADIX

Zapis: `@radix expr ;`

*Expr*: określa bazę systemu liczb jaki ma być domyślny : 2,8,10,16. Np.:

@radix 2; "zmień system domyślny na binarny

@radix 10; "zamień z powrotem system binarny na dziesiętny

### @STANDARD

Zapis: @standard

Dyrektywa @standard przywraca ustawienia ABEL-HDL do standardowych.

### 4.3.5. Zmienne zespolowe (SET) - wektory

Wektor (SET) jest zespołem sygnałów lub stałych używanych do opisu grupy sygnałów przez nazwę. Dyrektywa **Set** jest bardzo użyteczna przy upraszczaniu zapisu logicznego równań.

Wektor (SET) jest listą stałych lub sygnałów oddzielonych przecinkami lub operatorem zakresy „..”, umieszczonymi **koniecznie** w nawiasach kwadratowych.

Przykład 4.3:

```
[D0,D1,D2,D4,D5]
[D0..D6]           " zakres rosnący
[b6..b0]           " zakres malejący
[D7..D15]
[b1,b2,a0..a3]     " zakres z podzakresem
[!S7..!S0]         "zakres malejący z aktywnym
                    stanem niskim
```

Jednakże niedozwolony jest zapis: [D0, X];

gdzie X = [X3..X0];

Można zapisać takie wyrażenie w następujący sposób:

```
[D0, X3..X0];
```

#### 4.3.5.1. Indeksowanie i dostęp do wektora (SET)

Indeksowanie daje możliwość dostępu do elementów wewnątrz wektora. Do elementów wewnątrz wektora odwoływać się przez numer pozycji, który element zajmuje w wektorze.

Przykład 4.4:

```
D1 = [D15..D0]; "deklaracja wektora
X2 = [X3..X0];  "deklaracja wektora
X2 := D1[3..0]; "X2 równy [D3, D2, D1, D0]
X2 := D1[7..4]; "X2 równy [D7, D6, D5, D4]
```

Aby uzyskać dostęp do jednego z elementów wektora można użyć zapisu:

```
OUT = (X[2] == 1);
```

Operator(==) jest użyty to konwersji pojedynczego elementu X[2] do wartości bitowej jemu równoważnej, operator (==) daje wynik „1” lub „0” w zależności od tego czy porównanie



jest prawdą, czy fałszem. Operator (=) jest operatorem przypisania i przypisuje wynik komparacji do wyjścia OUT.

#### 4.3.5.2. Operacje na wektorze

Większość operatorów może być używana do wykonywania operacji na wektorach. Operacje te wykonywane są zgodnie z zasadami algebry boolowskiej, według ustalonych priorytetów.

Przykład 4.5:

```
Signal = [D2,D1,D0]; "deklaracja wektora
Signal = [1,0,1] & [0,1,1];" wynik dzia•ania [0,0,1]
```

Przykład 4.6:

```
[A,B] = C & D;
inny zapis:
A = C & D;
B = C & D;
```

Przykład 4.7:

```
[A1,B1] = [D1,D2] & [C3,C2];
równoważne z:
[A1,B1] = [D1 & C3, D2 & C2];
inny zapis:
A1 = D1 & C3;
B1 = D2 & C2;
```

Przykład 4.8:

```
X & [A,B,C];
równoważne z:
[X&A, X&B, X&C];

jednakże zapis:
2 & [A,B,C];
zostanie zinterpretowany jako:
[0 & A, 1 & B, 0 & C];
```

Przykład 4.9:

```
A=[A2,A1,A0]; "deklaracja wektora
B=[B2,B1,B0]; "deklaracja wektora
A # B; równoważne z [A2 # B2, A1 # B1, A0 # B0];
!A; równoważne z [!A2,!A1,!A0];
```

Przykład 4.10:

```
[b3,b2,b1,b0] = 2;"równowa•ne z b3=0,b2=0,b1=1,b0=0.
Wektory są bardzo przydatne w opisie równań logicznych:
```

## Przykład 4.11:

```

Chip_Sel = !A7 & A6 & A5;
Można to zrobić za pomocą wektora
Addr = [A7,A6,A5]; " tworzymy stały wektor
I używając równania wybrać adres
Chip_Sel = Addr == [0,1,1];
Co jest równoważne:
Chip_Sel = !A7 & A6 & A5;
Można również zapisać
Chip_Sel = Addr == 3; " dziesiętnie 3 to binarnie 011.

```

## Przykład 4.12:

Dla tych samych stałych co powyżej wyrażenie

```
3 & Addr;
```

co jest równoważne

```
[0,1,1] & [A7,A6,A5]
[0 & A7, 1 & A6, 1 & A5]
[0,A6,A5]
```

Jednakże zapis jest wówczas inny

```
3 & (Addr == 1);
```

co jest równoważne:

```
3 & (!A7 & !A6 & A5).
```

Operator(==) daje wynik bitowy, reszta wyrażenia jest też bitowa, a 3 jest obcinane do 1 i w rzeczywistości równanie jest równoważne:

```
1 & !A7 & !A6 & A5.
```

#### 4.3.6. Operatory

W języku ABEL istnieją cztery podstawowe typy operatorów: Logiczne, arytmetyczne, relacji i przypisania.

##### 4.3.6.1. Operatory logiczne

Tablica poniżej zawiera wszystkie dostępne operatory logiczne, operacje wykonywane są bit po bicie. Jeżeli aktywna jest dyrektywa @ALTERNATE można używać alternatywnego zapisu operatorów.

Tabela 4.5. Operatory logiczne w języku ABEL

Operator (domyślny)	Opis	Operator alternatywny
!	NOT (uzupełnienie do jednego)	/
&	AND	*
#	OR	+
\$	XOR: exclusive or	:+:
!\$	XNOR: exclusive nor	:*:

##### 4.3.6.2. Operatory arytmetyczne

Tabela poniżej zawiera wszystkie dostępne operatory arytmetyczne.

Tabela 4.6. Operatory arytmetyczne w języku ABEL

Operator	Przykład	Opis
-	-D1	Uzupełnienie do dwóch(negacja)
-	C1-C2	Odejmowanie
+	A+B	Dodawanie
<i>Poniższe nie są dozwolone z wektorami:</i>		
*	A*B	Mnożenie
/	A/B	Dzielenie całkowite
%	A%B	Reszta z dzielenia
<<	A<<B	Obróć A o B bitów w lewo
>>	A>>B	Obróć A o B botów w prawo

#### 4.3.6.3. Operatory relacji

Operatory te generują wartości Boolowskie prawda (-1) i fałsz (0). (W kodzie uzupełnień do dwóch na 16-bitach "-1" jest reprezentowany jako 1111 1111 1111 1111).

Tabela 4.7. Operatory relacji w języku ABEL

Operator	Przykład	Opis
==	A==B or 3==5 (fałsz)	równe
!=	A!=B or 3 != 5 (prawda)	Nie równe
<	A<B or 3 < 5 (prawda)	Mniej niż
<=	A<=B or 3 <= 5 (prawda)	Mniejszy lub równy
>	A>B or -1 > 5 (prawda)	Miekszy niż
>=	A>=B or !0 >= 5 (prawda)	Większy lub równy

**Operatory relacji działają bez znaku.** Zatem !0 jest dopełnieniem do 0 lub 11111111 (8 bitowa dana) co jest równe 255 w kodzie bez znaku. Dlatego !0 > 9 jest prawdą. Wyrażenie -1>5 jest prawdą z tego samego powodu. Np.:

$A = B \text{ !\$ } (C == D);$

A będzie równe B jeżeli C jest równe D, w innym przypadku A będzie równe dopełnieniu B (jeżeli C nie jest równe B (fałsz lub zero)).

#### 4.3.6.4. Operatory przypisania

Operatory te są stosowane do przypisywania wartości w równaniach do sygnałów i węzłów wyjściowych. Rozróżnia się dwa typy operatorów: **kombinacyjne** i **rejestrówce**. W kombinacyjnych przypisanie następuje natychmiast, w rejestrówce przypisanie następuje przy następnym takcie zegara.

```
Q1 pin istype 'reg';
Q1 := D;
```

Pierwsze wyrażenie definiuje Q1 jako przerzutnik przy użyciu 'reg' istype (wyjście rejestrowe), drugie wyrażenie dokonuje powiązania wyjścia Q1 z wejściem D. Jeżeli nastąpi zmiana D to przy następnym takcie zegara nastąpi zmiana na Q1.

Tabela 4.8. Operatory przypisania w języku ABEL

Operator	Opis
=	Przypisanie kombinacyjne
:=	Przypisanie rejestrowe

#### 4.3.6.5. Operatory priorytetu

Poniższa tabela przedstawia zestawienie priorytetów poszczególnych operatorów, gdzie 1 jest najwyższym 4 najniższym. W przypadku gdy dwa operatory mają priorytet tego samego poziomu, to wykonywane są zgodnie z regułą od lewej do prawej strony.

Tabela 4.9. Zestawienie priorytetów operatorów w języku ABEL

Priorytet	Operator	Opis
1	-	Negacja
1	!	NOT
2	&	AND
2	<<	Obrót w lewo
2	>>	Obrót w prawo
2	*	Mnożenie
2	/	Dzielenie bez znaku
2	%	Reszta z dzielenia
3	+	dodawanie
3	-	Odejmowania
3	#	OR
3	\$	XOR
3	!\$	XNOR
4	==	Równy
4	!=	Nie równy
4	<	Mniejszy niż
4	<=	Mniejszy równy
4	>	Większy niż
4	>=	Większy równy

#### 4.3.7. Opis logiki projektowanego układu

Projekt logiki projektowanego układu może być przedstawiony za pomocą:

- równań,
- tablicy prawdy,
- opisu stanów.

##### 4.3.7.1. Równania

Użycie słowa kluczowego **equations** rozpoczyna opis logiki za pomocą równań. Równania można konstruować używając operatorów jak powyżej, lub za pomocą wyrażenia WHEN-THEN-ELSE (uwaga: IF-THEN-ELSE używane jest przy opisie stanowym).

Sposób zapisu wyrażenia "When-Then-Else":

```
WHEN [warunek] THEN [element]=[wyra•enie];
ELSE [równanie];
lub
WHEN [warunek] THEN [równanie];
```

Przykład 4.13:

```
SUM = (A & !B) # (!A & B) ;
A0 := EN & !D1 & D3 & !D7;
WHEN (A == B) THEN D1_out = A1;
    ELSE WHEN (A == C) THEN D1_out = A0;
WHEN (A>B) THEN { X1 :=D1; X2 :=D2; }
Nawiasy klamrowe { } grupują sekcje w bloki.
```

#### 4.3.7.2. Tablica prawdy

Dla zapisu tablicy prawdy słowem kluczowym jest **truth-table** np.:

```
TRUTH_TABLE ( in_ids -> out_ids )
inputs -> outputs ;
lub
TRUTH_TABLE ( in_ids :> reg_ids )
inputs :> reg_outs ;
lub
TRUTH_TABLE
( in_ids :> reg_ids -> out_ids )
inputs :> reg_outs -> outputs ;
```

W powyższych zapisach „->” jest wyjściem kombinacyjnym, a „:>” wyjściem rejestrowym. Pierwsza linia tablicy prawdy definiuje wejścia i wyjścia sygnałów. Następne linie opisują wartości wejść i wyjść, każda linia musi być oddzielona średnikiem. Wejścia i wyjścia mogą być pojedyncze, lub wektorowe.

Przykład 4.14. (sumator połówkowy):

```
TRUTH_TABLE ( [ A, B ] -> [ Sum, Carry_out ] )
[ 0, 0 ] -> [ 0, 0 ];
[ 0, 1 ] -> [ 1, 0 ];
[ 1, 0 ] -> [ 1, 0 ];
[ 1, 1 ] -> [ 1, 1 ];
```

Jeżeli zdefiniujemy IN = [A,B] i OUT = [Sum,Carry\_out] to można zapisać tablice w postaci:

```
TRUTH_TABLE ( IN -> OUT )
0 -> 0;
1 -> 2;
2 -> 2;
```

```
3 -> 3;
```

Przykład 4.15. (EXOR z dwoma wejściami i wejściem enable (EN), .X. oznacza że stan nie jest istotny):

```
TRUTH_TABLE ([EN, A, B] -> OUT )
[ 0, .X., .X. ] -> .X.;
[ 1, 0 , 0 ] -> 0;
[ 1, 0 , 1 ] -> 1;
[ 1, 1 , 0 ] -> 1;
[ 1, 1 , 1 ] -> 0;
```

Przykład 4.16. (Trzybitowy licznik, zliczający do góry od 000 do 111, a następnie od 111 w dół do 000. Wyjścia typu rejestrowego nazwane kolejno QA, QB, QC. Na wyjściu OUT będą sygnalizowane stany 111 licznika. Licznik będzie zerowany, gdy na wejściu RESET będzie stan wysoki):

```
MODULE CNT3;
CLOCK pin; " wejście zegarowe
RESET pin; " wejście resetujące
OUT pin istype 'com'; " wyjście sygnału (kombinacyjne)
QC,QB,QA pin istype 'reg'; " wyjście sygnału (rejestrowe)

[QC,QB,QA].CLK = CLOCK; "FF taktowane na wejściu CLOCK
[QC,QB,QA].AR = RESET; "asynchroniczny reset przez wejście
                    " RESET

TRUTH_TABLE ( [QC, QB, QA] :> [QC,QB,QA] -> OUT)
[ 0 0 0 ] :> [ 0 0 1 ] -> 0;
[ 0 0 1 ] :> [ 0 1 0 ] -> 0;
[ 0 1 0 ] :> [ 0 1 1 ] -> 0;
[ 0 1 1 ] :> [ 1 0 0 ] -> 0;
[ 1 0 0 ] :> [ 1 0 1 ] -> 0;
[ 1 0 1 ] :> [ 1 1 0 ] -> 0;
[ 1 1 0 ] :> [ 1 1 1 ] -> 0;
[ 1 1 1 ] :> [ 0 0 0 ] -> 1;

END CNT3;
```

#### 4.3.7.3. Opis stanowy

Opis stanowy jest metodą analizy projektu logicznego na podstawie przejść stanów urządzenia. Opis stanowy zawiera wyrażenia „IF-THEN-ELSE”, „GOTO”, „WITH”. Zazwyczaj stany deklaruje się w tablicy nazw stanów, co czyni program łatwiejszym w czytaniu.

Deklaracja stanów (w części deklaracyjnej) posiada następującą formę:

```
state_id [, state_id ...] STATE;
```

Np.: SREG = [Q1, Q2]; powiązanie nazwy stanu SREG ze stanem zdefiniowanym przez Q1 i Q2.

Zapis diagramu stanowego:

```

State_diagram state_reg
STATE state_value : [equation;]
[equation;]
:
:
trans_stmt; ...

```

Słowo kluczowe **state\_diagram** wyznacza początek opisu maszyny stanowej. Natomiast słowo kluczowe **STATE** opisuje jeden stan włączając wartość stanu lub jego nazwę.

**state\_reg**: jest to identyfikator, który definiuje i determinuje stan urządzenia. Może to być symboliczny rejestr stanu, który został zadeklarowany w części deklaracyjnej.

**state\_value**: może być wyrażeniem, wartością lub symboliczną nazwą aktualnego stanu.

**equation**: równanie opisujące wyjścia maszyny stanowej.

**trans\_stmt**: wyrażenie definiujące stan następny. Mogą tu być użyte wyrażenia "If-Then-Else", CASE, GOTO, mogą być poprzedzone wyrażeniem obrębu WITH.

Wyrażenie **If-Then-Else** jest używane w opisie stanowym i służy do definiowania stanu następnego i do określenia wyjątków przejść stanowych. Zapis:

```

IF [wyrażenie] THEN state_exp
[ELSE state_exp] ;

```

W wyrażeniu „IF-THEN-ELSE” **state\_exp** może być wyrażeniem logicznym, lub symboliczną nazwą stanu. Warto zauważyć, że wyrażenie „IF-THEN-ELSE” może być używane tylko w opisie stanowym do opisu funkcji logicznych. ELSE jest opcjonalne, „IF-THEN-ELSE” może być poprzedzone wyrażeniem Goto, Case i With.

Przykład 4.17:

```

SREG = [Q1, Q0]; "definicja rejestru stanów
S0 = [0, 0];
S1 = [1, 1];
state_diagram SREG
state S0: OUT1 = 1;

    if A then S1
    else S0;
state S1: OUT2 = 1;

    if A then S0
    else S1;

```

Wyrażenie **WITH**: używa się następująco:

```

trans_stmt state_exp WITH [równanie]
[równanie] ... ;

```

W zapisie **trans\_stmt** może być wyrażeniem „If-then-else”, „Goto” lub „Case”. **state\_exp** jest następnym stanem, a **równanie** jest równaniem opisującym wyjścia maszyny stanowej. Wyrażenie „WITH” może być użyte z wyrażeniami: „If-Then-Else”, „Goto” lub „Case”, w

miejsce wyrażeń prostych. Wyrażenie „WITH” daje możliwość zapisu wyjść maszyny stanowej w postaci:

```
if X#Y==1 then S1 with Z=1 else S2;
```

W poniższym przykładzie wyjście Z będzie ustawione, jeśli tylko testowane wyrażenie logiczne przyjmie wartość jedynki logicznej (prawda „1”). Wyrażenie „WITH” może być równaniem, które będzie wykonane jeśli tylko warunek testowany w części „IF-THEN-ELSE” zostanie spełniony.

Przykład 4.18:

```
if X&!Y then S3 with Z=X#Y else S2 with Z=Y;
```

Wyrażenie „WITH” jest również przydatne do opisu zachowania rejestrów wyjściowych. Można na przykład określić konkretną zmianę stanu wyjścia, przy konkretnym warunku;

Przykład 4.19:

```
state S1:
  if RST then S2 with { OUT1 := 1; Error-Adrs := ADDRESS; }
  else if (ADDRESS <= ^hC101)
    then S4
    else S1;
```

Wyrażenie **Case**: posiada postać:

```
CASE expression : state_exp;
[ expression : state_exp; ]
:
ENDCASE ;
```

Wyrażenie *expression* jest jakimkolwiek dozwolonym w zapisie języka ABEL wyrażeniem logicznym, a *state\_exp* jest wyrażeniem opisującym następny stan (ewentualnie poprzedzony wyrażeniem „WITH”).

Przykład 4.20:

```
State S0:
  case ( A == 0 ) : S1;
  ( A == 1 ) : S0;
endcase;
```

#### 4.3.8. Rozszerzenie z kropką w deklaracji pinów

Rozszerzenia wyrażenia z kropką używa się w celu precyzyjniejszego opisu zachowania projektowanego układu, a ściślej do sposobu realizacji pewnych funkcji. Pozwala to na precyzyjną specyfikację sygnałów i węzłów powiązanych z rzeczywistym sygnałem. Zapis tego rozszerzenia jest następujący:

```
signal_name.ext
```



Niektóre z rozszerzeń z kropką opisano poniżej w tabeli. Rozszerzenia z kropką mogą być ogólne lub mogą dotyczyć konfiguracji pin-pin.

Tabela 4.10. Znaczenie rozszerzeń z kropką w języku ABEL

Rozszerzenie z kropką	Opis
<b>Do opisu niezależnej architektury, lub konfiguracja pin-pin</b>	
.ACLR	Asynchroniczne kasowanie rejestru
.ASET	Asynchroniczne ustawianie rejestru
.CLK	Połącz Zegar do wejść przerzutników wyzwanych zboczem
.CLR	Synchroniczne kasowanie rejestru
.COM	Kombinacyjne sprzężenie zwrotne z przerzutnika na wejście
.FG	Rejestr sprzężenia zwrotnego
.OE	Wyjścia dostępne
.PIN	Końcówka sprzężenia zwrotnego
.SET	Synchroniczne ustawianie rejestru
<b>Specyficzne rozszerzenia urządzenia(dla konkretnej architektury)</b>	
.D	Wejście danych przerzutnika D
.J	Wejście J przerzutnika JK
.K	Wejście K przerzutnika JK
.S	Wejście S przerzutnika SR
.R	Wejście R przerzutnika SR
.T	Wejście T przerzutnika T
.Q	Rejestr wyjścia(sprzężenie zwrotne)
.PR	Rejestr ustawiania
.RE	Rejestr kasowania
.AP	Asynchroniczny rejestr ustawiający
.AR	Asynchroniczny rejestr kasujący
.SP	Synchroniczny rejestr ustawiający
.SR	Synchroniczny rejestr kasujący

Przykład 4.21:

```
[S6..S0].OE = ACTIVE;
```

Jeżeli ACTIVE jest w stanie wysokim to sygnały składające się na wektor [S6..S0] będą miały załączone wyjścia, jeżeli natomiast ACTIVE będzie w stanie niskim, to wyjścia wektora [S6..S0] będą w stanie wysokiej impedancji.

Przykład 4.22:

```
Q.AR = reset;
[Z.ar, Q.ar] = reset;
```

Jeżeli reset jest w stanie wysokim, to następuje kasowanie przerzutników składających się na rejestr.

#### 4.3.9. Wektory testujące

Wektory testujące są opcjonalne i mają za zadanie zweryfikowanie poprawności działania maszyny stanowej. Wektory te określają oczekiwane wyniki pewnych operacji zapisywanych na wyjściach w funkcji zmiany wejść. Ich zapis przyjmuje postać:

```
Test_vectors [note]
(input [, input ].. -> output [, output ] .. )
[invalues -> outvalues ; ]
:
:
```

Przykład 4.23:

```
Test_vectors
( [A, B] -> [Sum, Carry] )
[ 0, 0 ] -> [0, 0];
[ 0, 1 ] -> [1, 0];
[ 1, 0 ] -> [1, 0];
[ 1, 1 ] -> [1, 1];
```

Dla wektorów można również określać wartości numeryczne określające zawartość rejestru.

```
Test_vectors
( [A, B] -> [Sum, Carry] )
0 -> 0;
1 -> 2;
2 -> 2;
3 -> 3;
```

Pola bez znaczenia (.X.), wejścia zegarowe (.C.), jak również zapis symboliczny stałych mogą być używane w wektorach testujących.

```
Test_vectors
( [CLK, RESET, A, B] -> [ Y0, Y1, Y3] )
[.X., 1, .X.,.X.]->[ S0, 0, 0];
[.C., 0, 0, 1 ] -> [ S0, 0, 0];
[.C., 1, 1, 0 ] -> [ S0, 0, 1];
```

#### 4.3.10. Deklaracje aktywności stanem niskim

Definicji sygnału aktywnego stanem niskim dokonuje się przy użyciu operatora „!”:

```
!OUT pin istype 'com' ;
```

Jeżeli ten sygnał będzie używany w rozbudowanym opisie projektu, to automatyczne będzie on uzupełniany do jedynki.

Przykład 4.24:

```
module EXAMPLE
```

```

A, B pin ;
!OUT pin istype 'com';
equations
OUT = A & !B # !A & B ;
end

```

W powyższym przykładzie sygnał OUT jest wynikiem operacji XOR sygnałów A i B. Wobec czego OUT (stan wysoki lub on) będzie miał miejsce dla sygnału A różnego od B. Jednakże OUT został zadeklarowany jako aktywny stanem niskim wobec czego wyniki operacji będą przeciwne. Wydawałoby się że można to samo osiągnąć przez negację OUT. Lecz to rozwiązanie nie będzie aktywne stanem niskim.

Aktywność stanem niskim można również zapisać w inny sposób. Weźmy wektory A, B i X:

```

A = [A2,A1,A0]; "deklaracja wektora
B = [B2,B1,B0]; "deklaracja wektora
X = [X2,X1,X0]; "deklaracja wektora

!X = A & !B # !A & B;

```

Powyższe równanie jest równoznaczne zapisowi:

```

!X0 = A0 & !B0 # !A0 & B0;
!X1 = A1 & !B1 # !A1 & B1;
!X2 = A2 & !B2 # !A2 & B2;

```

#### 4.3.11. Przykłady wykorzystania języka ABEL

Poniżej zostanie przedstawionych kilka przykładów wykorzystania języka ABEL opartych na układzie GAL16V8.

##### Przykład 4.25:

Proste bramki logiczne zaimplementowane w układzie GAL16V8:

- bramka AND o wejściach A i B oraz wyjściu C,
- bramka OR o wejściach D i E oraz wyjściu F,
- negator o wejściu G i wyjściu H,
- bramka NAND o wejściach A i B oraz wyjściu I,
- bramka NOR o wejściach D i E oraz wyjściu J,
- bramka EX-OR o wejściach K i L oraz wyjściu M.

Postać pliku źródłowego napisanego w języku ABEL dla powyższego projektu jest następująca:

```

Module proste_bramki
Title 'podstawowe bramki logiczne'
bramki device 'P16V8';

"Piny wejsciowe
A, B      pin 2, 3;  "wejscia bramek AND i NAND
D, E      pin 4, 5;  "wejscia bramek OR i NOR

```

```

G          pin 6;      "wejście bramki NOT
K, L       pin 7, 8;   "wejścia bramki EX-OR

"Piny wyjściowe
C          pin 12 istype 'com';    "wyjście bramki AND
F          pin 13 istype 'com';    "wyjście bramki OR
H          pin 14 istype 'com';    "wyjście bramki NOT
I          pin 15 istype 'com';    "wyjście bramki NAND
J          pin 12 istype 'com';    "wyjście bramki NOR
M          pin 12 istype 'com';    "wyjście bramki EX-OR

```

#### Equations

```

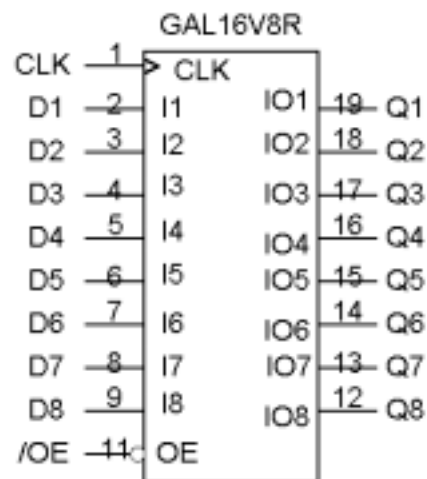
C = A & B;          " AND
F = D # E;          " OR
H = !G;             " NOT
I = !(A & B);        " NAND
J = !(D # E);        " NOR
M = K $ L;          " EX-OR

```

```
end proste_bramki
```

#### Przykład 4.26:

W przykładzie tym układ GAL16V8 ma za zadanie pracować jako 8-bitowy rejestr typu D wyzwany zboczem, czyli jako układ 74HCT374. Realizację tę pokazano na rys. 4.18.



Rys. 4.18. Realizacja za pomocą układu GAL16V8 8-bitowego rejestru 74HCT374

Plik źródłowy ma następującą postać

```

Module osmio_bitowy_rej
Title '8-bitowy rejestr typu 74HCT374'
74HCT374 device 'P16V8';

"wejścia
CLK, !OE          pin 1, 11;
D1, D2, D3, D4, D5, D6, D7, D8  pin 2, 3, 4, 5, 6, 7, 8, 9;
"wyjścia

```

```
Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8    pin 19, 18, 17, 16, 15, 14,
13, 12 istype 'reg';
```

```
"definicje
D = [D1, D2, D3, D4, D5, D6, D7, D8];
Q = [Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8];
```

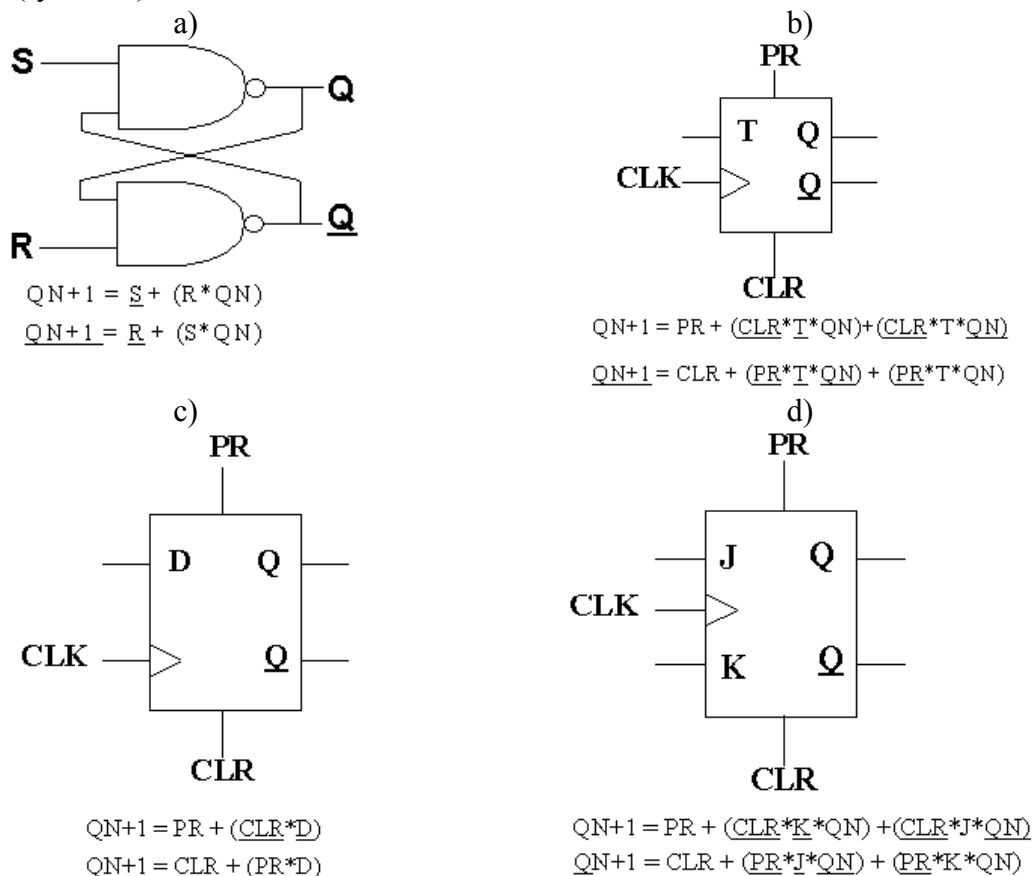
```
Equations
```

```
Q := D;
```

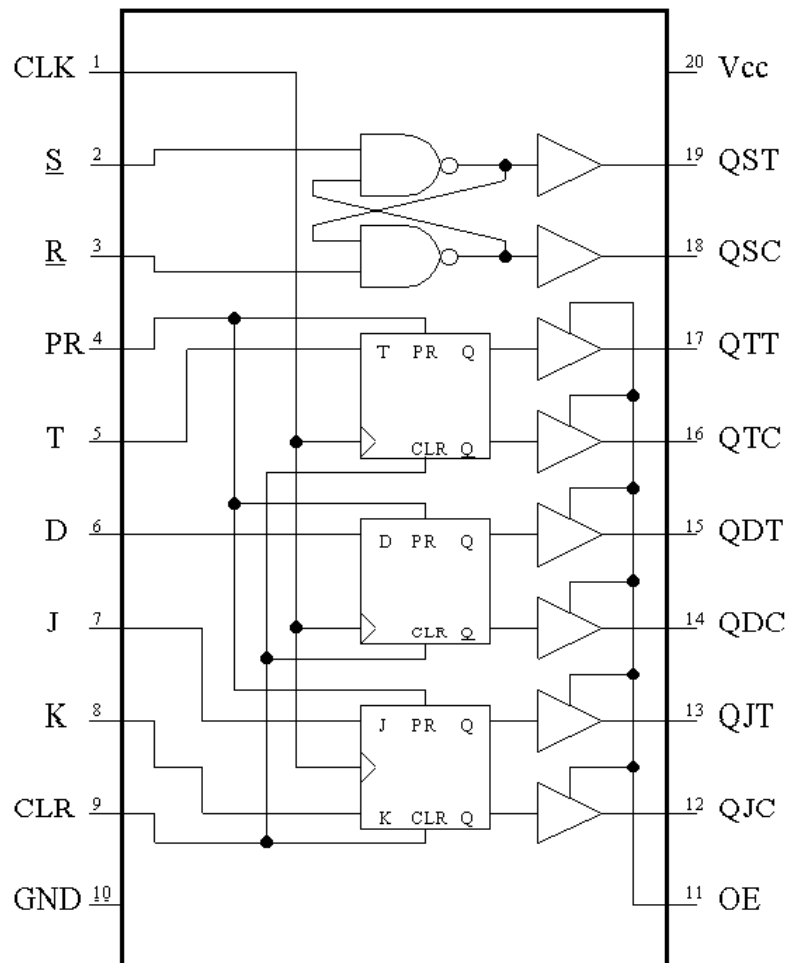
```
end osmio_bitowy_rej
```

#### Przykład 4.27:

W przykładzie tym przedstawiono implementację podstawowych typów przerzutników: RS, T, D i JK (rys. 4.19).



Rys. 4.19. Schematy i opis w postaci równań przerzutników: a) RS, b) T, c) D i d) JK  
 Sposób implementacji (umiejscowienia) tych przerzutników w układzie GAL16V8 pokazano na rys. 4.20.



Rys. 4.20. Sposób umiejscowienia przerzutników w układzie GAL16V8

Zatem plik źródłowy projektu jest następujący:

```
Module podstawowe_przerzutniki
Title 'Podstawowe przerzutniki: RS, T, D, JK'
przerzutniki device 'P16V8';

"wejścia
CLK,PR, CLR, !OE    pin 1, 4, 9, 11; " sygnały kontrolne
S, R                pin 2, 3;       " RS
T                   pin 5;          " T
D                   pin 6;          " D
J, K                pin 7, 8;       " JK

"wyjścia
QST, QSC            pin 19, 18 istype 'com'; " RS
QTT, QTC            pin 17, 16 istype 'reg'; " T
QDT, QDC            pin 15, 14 istype 'reg'; " D
QJT, QJC            pin 13, 12 istype 'reg'; " JK

Equations
" RS
QST = !S # (R & QST);
```

```
QSC = !R # (S & QSC);
" T
QTT := PR # (!CLR & !T & QTT) # (!CLR & T & QTC);
QTC := CLR # (!PR & !T & QTC) # (!PR & T & QTT);
" D
QDT := PR # (D & !CLR);
QDC := CLR # (!D & !PR);
" JK
QJT := PR # (J & QJC & !CLR) # (!K & QJT & !CLR);
QJC := CLR # (!J & QJC & !PR) # (!K & QJT & !PR);

end podstawowe_przerzutniki
```

Pliki źródłowe zawierające kod w języku ABEL (rozszerzenie **.abl**) poddaje się kompilacji przez odpowiednio do tego celu stworzone kompilatory. W wyniku kompilacji, dla układów typu SPLD, jak i również układów CPLD, uzyskuje się pliki w standardzie JEDEC zawierające mapę wewnętrznych połączeń w programowalnym układzie.

W kolejnym etapie przy pomocy odpowiednich programatorów i dostarczonego do nich oprogramowania wprowadza się ten kod do układów.

## 5. Układy peryferyjne z interfejsem szeregowym standardu SPI

W rozdziale tym zostaną omówione wyłącznie układy peryferyjne z interfejsem SPI, ponieważ coraz więcej układów jest wyposażanych w ten standard, a ponadto jest on znacznie prostszy i łatwiejszy w implementacji niż standard I<sup>2</sup>C.

Ponieważ opis interfejsu SPI został przedstawiony w podrozdziale 2.6.5.2 zatem tutaj zostaną omówione tylko jego właściwości wynikające ze współpracy mk z urządzeniami peryferyjnymi.

Do komunikacji pomiędzy mk a układami peryferyjnymi oraz do komunikacji pomiędzy samymi układami peryferyjnymi w mse stosuje się współcześnie interfejsy szeregowo **standardu SPI**, jak i standardu I<sup>2</sup>C. Transmisja równoległa pozwala na osiągnięcie większych szybkości, lecz wymaga stosowania magistrali o wielu (przynajmniej 8) liniach. Natomiast transmisja szeregowo w porównaniu z równoległą wymaga:

- mniej linii połączeniowych (przeważnie trzy lub dwie),
- mniej dodatkowych układów,
- charakteryzują ją łagodniejsze wymagania czasowe,
- układy transmisyjne zajmują mniej miejsca i zacisków we/wy układu scalonego.

W interfejsie SPI formaty danych układów peryferyjnych nie są określone. Bitowe ciągi danych transmitowane interfejsem SPI **nie mają ustalonej długości** (jest to zazwyczaj krotność ośmiu bitów) oraz **kolejność bitów nie jest określona** (najczęściej transmisja zaczyna się od MSB i kończy na LSB). Długość transmitowanych danych ulega zmianie, nawet przy transmisjach do tego samego układu scalonego. Stąd stosuje się przeważnie dwie metody zapisu/odczytu danych:

- W przypadku prostych układów (o jednym typie danych i niezależnym sygnale zapisu danych) możliwe jest **przepelnienie zawartości odbiorczego rejestru szeregowego**. Zatem w układzie po przesłaniu do niego dowolnej ilości bitów zostaje zapamiętane tylko N ostatnich bitów (N – długość danych dla danego układu) w rejestrze odbiorczym. Wysyłany po transmisji danych sygnał zapisu przepisze do wnętrza układu te N bitów. Właściwość ta pozwala na zaokrąglenie w górę długości wysyłanego słowa do wielokrotności liczby 8, co pozwala na korzystanie z interfejsu SPI mk, który jak wiadomo operuje na danych 8-bitowych. W tym przypadku pierwsze bity są nieistotne, gdyż nie są zapamiętywane. Jest ich  $8n - N$  (gdzie  $n$  – wielokrotność 8 bitów, tak aby zawsze  $8n > N$ ).
- Przy bardziej złożonych układach wprowadza się do formatu danych **bit startu**. Czyli układ pomija kolejno przychodzące bity dopóki nie pojawi się pierwsza jedynka będąca bitem startu. Po niej dane są ważne.
- Słowa sterujące są krotnością 8 bitów. Liczba wysyłanych bajtów zależy od typu rozkazu.

W interfejsie SPI stosuje się różne **techniki adresowania** układów peryferyjnych. Muszą one uwzględnić fakt różnorodnych wymagań wynikających np. z obecności wielu niezależnych układów korzystających z tego samego portu transmisyjnego, konieczności wysyłania rozkazów przed, w trakcie i po transmisji danych. Dwie najpopularniejsze to:

- Najczęściej spotykana w przypadku układów pojedynczych technika, polegająca na wydzieleniu **jednej linii adresującej układ** (CS- *chip select*). Wprowadzenie tej linii w stan aktywny (najczęściej niski) jest jednoznaczne z zaadresowaniem układu do udziału w transmisji.



- Kolejna technika polega na zawarciu **adresu w przesyłanym ciągu danych**. Gdy układ odbiorczy stwierdzi zgodność adresu, to odbiera dane zawarte po adresie. Np. stosuje się tę technikę w układach składających się z wielu niezależnie programowanych bloków, z których każdy ma swój własny adres.

Obecnie produkuje się coraz szerszą gamę układów peryferyjnych wyposażonych w interfejsy szeregowy standardu SPI oraz kompatybilne z nim. Wśród nich można wymienić między innymi następujące układy:

- mikrokontrolery,
- przetworniki A/C (np. AD7866 – 12-bitowy, LTC2400 – 24-bitowy),
- przetworniki C/A (np. AD7394 – 12-bitowy, LTC1650 – 16-bitowy),
- cyfrowe potencjometry (np. AD5260 – 256 pozycji, 20k, 59k, 200k, 12V; AD5235 – 1024 pozycji z pamięcią nieulotną, 25k, 250k; MCP42100 – 256 pozycji, 100k),
- multipleksery analogowe (np. ADG708 – multiplekser 8:1 typu RF/Video CMOS, LTC1391 – 8 kanałowy MUX),
- czujniki temperatury (np. ADT7320 – 10-bitowy),
- sterowniki interfejsów UART, CAN, itd. (np. MAX3100 – SPI na UART, MCP2510 SPI na CAN 2.0B),
- zegary czasu rzeczywistego (np. MAX6902),
- generatory sygnałów (np. DS1050 – 5-bitowy generator sygnału PWM o częstotliwości 1kHz),
- pamięci EEPROM (np. 25LC16 – 16Kbits (x8)),
- pamięci FLASH (np. AT45D081 – 1Mbits),
- sterowniki LCD (np. STE2001 – wymiar ekranu 65x128),
- sterowniki LED (np. MAX6950 – 5 cyfr, MAX6952 – macierz 5x7, MAX7221 – 8 cyfr, NLSF595 firmy ON Semiconductor – kontroler trzech diod czerwonej, zielonej i niebieskiej),

Obecnie coraz więcej specjalizowanych układów peryferyjnych jest wyposażanych w procesor (mk), zatem i one w naturalny sposób posiadają interfejs SPI (np. ADuC812 – mk oparty na mk 8052).

## 5.1. Szeregowe pamięci EEPROM

Pamięci nieulotne EEPROM i FLASH z szeregowym wejściem i wyjściem danych są coraz częściej stosowane w mse ze względu na możliwość programowania ich zawartości w układzie, w którym pracują (*in-circuit programming*). Przechowują one dane, które po zaniku napięcia zasilania nie mogą ulec skasowaniu. Dane te najczęściej są wykorzystywane do indywidualnej linearyzacji charakterystyk czujników, zbierania danych pomiarowych, parametryzacji wbudowanych algorytmów itd. Ich zaletą, oprócz niskiej ceny i małych rozmiarów obudowy, jest standardowe napięcie zasilania równe zwykle +5V lub +3V. Układy te mają zwykle **wbudowaną przetwornicę napięcia** i nie wymagają do programowania i kasowania zewnętrznego napięcia programującego. Ponadto posiadają własną automatykę, tzn. odczyt, zapis, kasowanie oraz funkcje związane z ochroną danych są obsługiwane za pomocą odpowiednich rozkazów wprowadzanych do układu i przez niego interpretowanych.

Spotykane pamięci szeregowo EEPROM mają pojemność: 1Kbit, 2Kbit, 4Kbit, 8Kbit, 16Kbit, 32Kbit i 64Kbit. Pamięć jest zorganizowana bajtowo (x8), dwubajtowo (x16) i bajtowo lub dwubajtowo (x8 lub x16 – wybór przez użytkownika).

W praktyce spotyka się trzy podstawowe odmiany pamięci EEPROM z szeregowym wejściem i wyjściem danych:

- układy z **interfejsem SPI**,
- układy pracujące z **niestandardowym protokołem wymiany danych** (przeważnie standard przemysłowy **Microwire**),
- układy z **interfejsem I<sup>2</sup>C**.

Do sterowania pamięci EEPROM z szeregowym dostępem stosuje się powszechnie dwie metody:

- sterowanie przy **użyciu linii portów we/wy** mk (dla pamięci z niestandardowym protokołem wymiany danych),
- sterowanie przez **interfejs komunikacji szeregowej** (dla pamięci z interfejsem SPI i I<sup>2</sup>C).

Zasadniczą wadą pamięci EEPROM jest **długi czas zapisu** informacji, jak i **kasowania** układu wynoszący typowo od 2ms do 5ms, a nawet 10ms.

### 5.1.1. Szeregowe pamięci EEPROM z interfejsem SPI

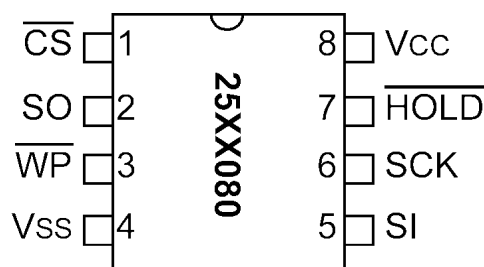
Zaletą pamięci szeregowych EEPROM z interfejsem SPI jest możliwość ich bezpośredniego podłączenia do interfejsu SPI mk. Zatem obsługa tych pamięci przez mk może odbywać się w trybie przerwaniowym. W tym trybie mk może realizować inne zadania, podczas gdy trwa transmisja. Jc jest angażowana tylko przy odczycie lub zapisie danych do rejestru szeregowego interfejsu SPI. Powiadomienie jc o zakończeniu transmisji bajtu odbywa się za pomocą przerwania od interfejsu SPI, który wpierw musi być właściwie skonfigurowany. Kolejną zaletą jest fakt, iż konwersja równoległo-szeregowa (wpis danej przez jc do bufora szeregowego i wysłanie jej szeregowo przez wyjście danych interfejsu) i szeregowo-równoległa (szeregowy odczyt danej z wejścia interfejsu do bufora i odczyt jej przez jc z tego bufora) danych odbywa się sprzętowo, a nie programowo jak ma to miejsce przy zastosowaniu popularnych pamięci EEPROM, umówionych w kolejnym podrozdziale, z niestandardowym interfejsem Microwire.

Pamięci szeregowo EEPROM i interfejsem SPI zostaną przedstawione na przykładzie układu 25C080 firmy Microchip. Jest to 8Kbitowa pamięć o architekturze bajtowej (1024x8 bitów). Dostęp do pamięci, jak i jej sterowanie odbywa się za pomocą interfejsu SPI, który składa się z linii: sygnał zegara SCK, sygnał danych wejściowych SI oraz sygnał danych wyjściowych SO. Dostęp do urządzenia jest kontrolowany poprzez sygnał wyboru układu CS (*Chip Select*). Istnieje możliwość wstrzymania komunikacji z pamięcią poprzez sygnał HOLD.

Układ ten ma następujące właściwości:

- Niski pobór mocy: prąd podczas zapisu maksymalnie 3mA, prąd podczas odczytu typowo 500μA, prąd spoczynkowy typowo 500nA.
- Organizacja pamięci: 1024x8 bity.
- 16 bajtowa strona.
- Czas cyklu zapisu maksymalnie 5ms.
- Wewnętrzne automatyczne cykle kasowania i zapisu.
- Ochrona zapisu bloku: ochrona żadnej, 1/4, 1/2 lub całej tablicy.
- Wbudowana ochrona zapisu: ochrona danych podczas włączania/wyłączania układu, zatrask umożliwiający zapis, pin ochrony przed zapisem.

- Sekwencyjny odczyt danych.
- Wysoka niezawodność: wytrzymałość 1M cykli kasowania/zapisu, czas przechowywania danych > 200 lat, ochrona wejść cyfrowych przed przepięciami typu ESD.
- Obudowa 8-pinowa PDIP oraz SOIC (rys. 5.1).

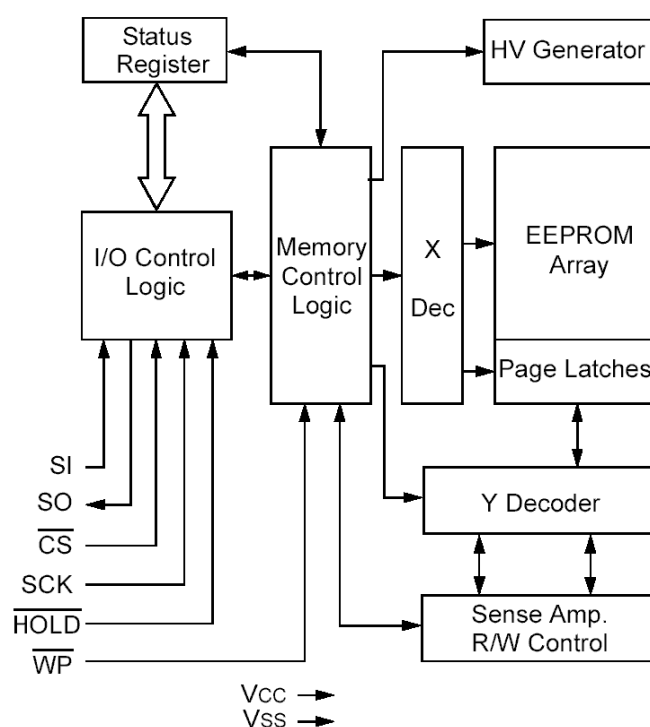


Rys. 5.1. Wyprowadzenia układu 25C080 firmy Microchop

Na rys. 5.1 pokazano wyprowadzenia pamięci EEPROM. Opis tych wyprowadzeń zebrano w tabeli 5.1.

Tabela. 5.1. Opis wyprowadzeń układu 25C080 firmy Microchip

Nazwa pinu	Obudowa PDIP	Obudowa SOIC	Funkcja
CS	1	1	Wejście wyboru układu
SO	2	2	Wyjście danych szeregowych
WP	3	3	Wyprowadzenie do ochrony zapisu
Vss	4	4	Uziemienie
SI	5	5	Wejście danych szeregowych
SCK	6	6	Wejście szeregowe zegara
HOLD	7	7	Wyprowadzenie pauzujące transmisję
Vcc	8	8	Zasilanie



Rys. 5.2. Schemat blokowy układu 25C080 firmy Microchip

Na rys. 5.2 pokazano schemat blokowy układu 25C080. Jak widać składa się on z dwóch części: pierwszą część stanowi zespół bloków związanych z interfejsem SPI i sterowaniem pamięcią. Są to bloki: we/wy układ sterujący (I/O Control Logic), rejestr statusu, układ sterujący pamięcią (*Memory Control Logic*), generator wysokiego napięcia (*HV Generator*), układ kontroli amplitudy przy odczycie/zapisie (*Sense Amp. R/W Control*). Druga część to pamięć "właściwa". Składa się ona z dekodera wierszy (*X Dec*) i dekodera kolumn (*Y Dec*), rejestru zatraskowego stron (*Page Latches*) i matrycy komórek EEPROM (*EEPROM Array*).

Zastosowania i funkcje pinów układu są następujące:

- **Linia CS** służy do uaktywnienia pamięci przez układ nadrzędny z nią współpracujący (np. mk). Uaktywnienie pamięci dokonuje się poprzez wystawienie na wejście CS niskiego poziomu sygnału. Wysoki poziom na tej linii dezaktywuje układ. Podczas inicjacji bądź trwania cyklu programowania wysoki stan sygnału CS nie powoduje przerwania procesu zapisu/odczytu. Zostaje on dokończony, po czym następuje dezaktywacja układu pamięci i przejście w stan czuwania. W tym trybie pracy wyjście SO przechodzi w stan wysokiej impedancji umożliwiając innym układom korzystanie z magistrali SPI. Zmiana stanu na wejściu CS z niskiego na wysoki po zakończonej sekwencji wprowadzania rozkazów do pamięci inicjuje wewnętrzny cykl zapisu.
- **Wyjście SO** jest używane w celu wysyłania danych z układu pamięci 25C080. Podczas cyklu odczytu dane są wystawiane na to wyjście na opadające zbocze sygnału zegarowego.
- **Wejście WP** wraz z bitem WPEN rejestru statusu służą do blokady zapisu rejestru statusu pamięci. Ochronę tę uzyskuje się, gdy na linii WP wystawiony jest niski poziom, zaś bitowi WPEN przypisana jest jedynka. W przypadku gdy bit WPEN jest ustawiony, to pojawienie się niskiego poziomu na linii WP podczas sekwencji zapisu rejestru statusu spowoduje przerwanie tej operacji. Jeżeli zaś wewnętrzny cykl zapisu został rozpoczęty wszelkie zmiany na linii WP nie wpływają na kontynuację wykonywania operacji zapisu. Gdy bit WPEN jest wyzerowany funkcja linii WP jest nieaktywna. Ustawienie bitu WPEN odblokowuje funkcję wyprowadzenia WP.
- **Wejście SI** jest używane do szeregowego wprowadzania instrukcji, adresów oraz danych z urządzenia nadrzędnego do pamięci. Dane na SI zatraskiwane są na narastające zbocze sygnału zegarowego.
- **Sygnał wejściowy SCK** służy do synchronizacji komunikacji pomiędzy urządzeniem nadrzędnym, a pamięcią. Narastające zbocze zegara zatraskuje (próbkuje) dane wprowadzane na wejściu SI, zaś opadające wystawia dane na wyjściu SO.
- **Wejście HOLD** służy do wstrzymania transmisji do pamięci, bez konieczności powtarzania całego cyklu transmisyjnego od nowa. W celu wstrzymania transmisji na pinie HOLD powinien być stan niski w momencie, gdy sygnał zegara SCK jest również w stanie niskim. W innym przypadku zawieszenie transmisji zostanie wykonane przy kolejnym przejściu sygnału SCK z poziomu wysokiego do niskiego. Istotne jest, aby układ pamięci pozostawał aktywny podczas wstrzymywania transmisji. Gdy transmisja zostaje zawieszona linie SI, SCK oraz SO przechodzą w stan wysokiej impedancji. Wymuszenie na linii HOLD wysokiego poziomu podczas niskiego poziomu sygnału zegara powoduje wznowienie transmisji.

Układ 25C080 zawiera 8 bitowy rejestr instrukcji. Aby układ był aktywny i przyjmował rozkazy/dane na linii CS musi panować stan niski, zaś na linii HOLD stan wysoki. Aby można było dokonywać operacji zapisu do tablicy pamięci na linii WP powinien być stan wysoki.

Tabela 5.2 zawiera listę i formaty instrukcji dostępnych dla układu 25C080. We wszystkich instrukcjach, adresach czy danych pierwszy jest przesyłany najbardziej znaczący bit MSB, a jako ostatni bit LSB. Wszelkie instrukcje, adresy, czy dane przesyłane są do pamięci paczkami 8-bitowymi, co pozwala na bezpośrednie stosowanie w tym celu interfejsu SPI mk.

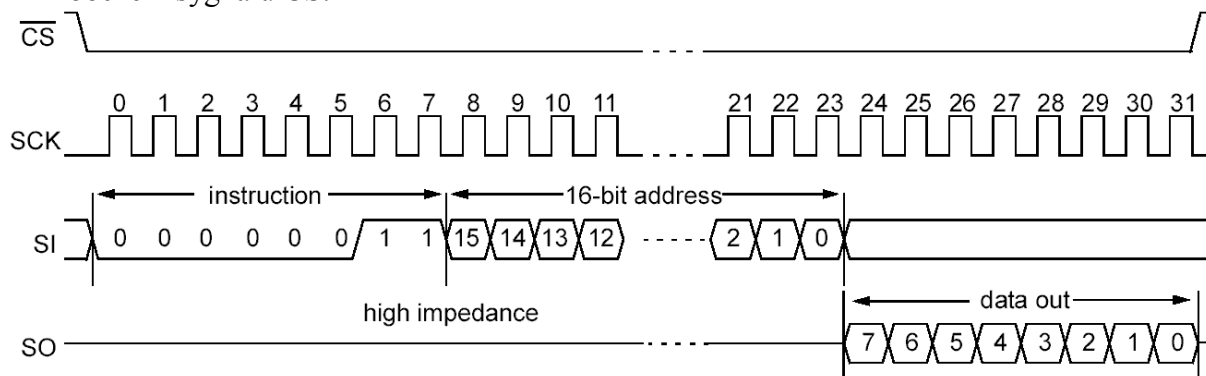
Tabela 5.2. Zestaw instrukcji sterujących układem 25C080

Nazwa instrukcji	Format instrukcji	Opis
READ	0000 0011	Odczyt danych z tablicy pamięci od wybranego adresu
WRITE	0000 0010	Zapis danych do tablicy pamięci od wybranego adresu
WRDI	0000 0100	Zerowanie zatrasku zezwolenia zapisu (niedostępne operacje zapisu)
WREN	0000 0110	Ustawienie zatrasku zezwolenia zapisu (dostępne operacje zapisu)
RDSR	0000 0101	Odczyt rejestru statusu
WRSR	0000 0001	Zapis rejestru statusu

Dane są próbkowane na pierwsze narastające zbocze sygnału zegara SCK po wymuszeniu na linii CS niskiego poziomu. W przypadku, gdy sygnał zegara na magistrali SPI jest współdzielony przez inne urządzenia peryferyjne, użytkownik może wprowadzić pamięć 25C080 w stan wstrzymania 'HOLD' poprzez wymuszenie niskiego stanu na linii HOLD. Po zmianie poziomu sygnału na tej linii z niskiego na wysoki następuje kontynuacja wstrzymanej transmisji.

Poniżej zestawiono opis rozkazów zawartych w tabeli 5.2:

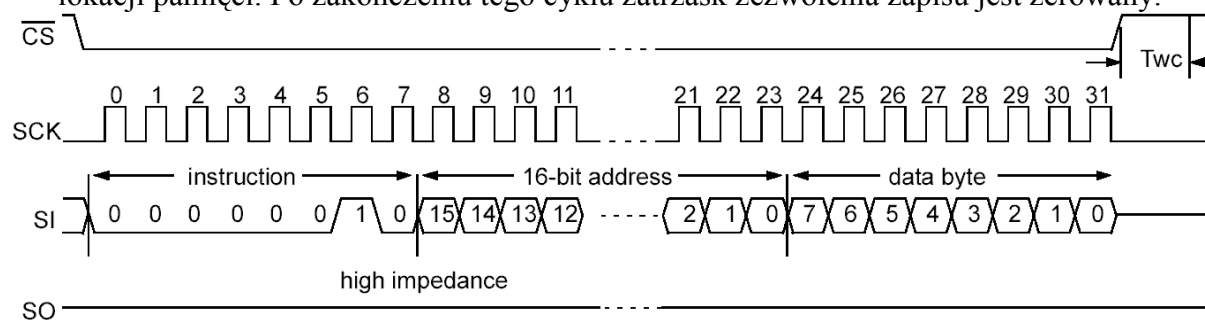
- Sekwencja odczytu **READ** rozpoczyna się od wyboru pamięci poprzez wymuszenie na linii CS stanu niskiego. Następnie wysyłana jest 8-bitowa instrukcja odczytu oraz 16-bitowy adres, gdzie sześć najbardziej znaczących bitów adresu nie jest brane pod uwagę. Po prawidłowej transmisji instrukcji i adresu, dane umieszczone pod wybranym adresem są wystawiane szeregowo na wyjściu SO. Po każdorazowym wysłaniu bajta danych wewnętrzny wskaźnik adresu jest automatycznie inkrementowany, aby wskazywać na kolejną daną w pamięci. Kiedy osiągnie adres 03FFH następuje zmiana jego wskazania na adres 0000H. Operacja transmisji sekwencji odczytu jest kończona na narastającym zboczem sygnału CS.



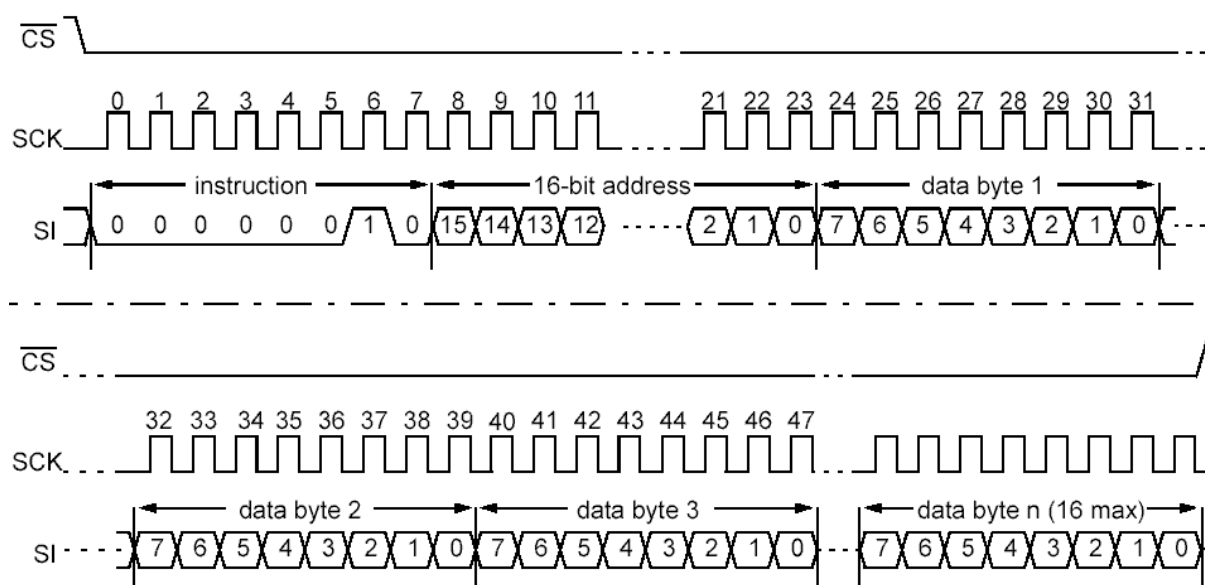
Rys. 5.3. Przebieg czasowy rozkazu READ

- Przed jakąkolwiek próbą zapisu danych do układu rozkazem WRITE, zatrask zezwolenia zapisu musi być ustawiony poprzez wykonanie instrukcji WREN. Sekwencja zapisu tej

instrukcji sprowadza się do wybrania układu niskim poziomem sygnału CS oraz wysłania kodu WREN do układu. Po transmisji 8 bitów tej instrukcji, poziom sygnału na linii CS musi zostać ustawiony z powrotem w stan wysoki, co w rezultacie powoduje ustawienie zatrasku zezwolenia zapisu. Próba rozpoczęcia transmisji zaraz po wysłaniu instrukcji WREN, bez zmiany stanu sygnału na linii CS pomiędzy tymi operacjami, nie powiedzie się ze względu na niewłaściwe ustawienia zatrasku zezwolenia zapisu. W momencie prawidłowego ustawienia tego zatrasku, użytkownik może rozpocząć operację zapisu danych do układu wcześniej wymusiwszy na pinie CS stan niski. Sekwencja zapisu sprowadza się do wysłania instrukcji zapisu WRITE, 16-bitowego adresu z sześcioma najbardziej znaczącymi bitami nie brany pod uwagę oraz danej przeznaczonej do zapisu pod przesłany adres. Przed wymaganym cyklem zapisu do pamięci może być wysłanych 16 bajtów danych. Jedynym ograniczeniem w tym przypadku jest to, iż wszystkie bajty muszą mieścić się w obszarze jednej strony pamięci. Adres strony zaczyna się od XXXX XXXX 0000 i kończy na XXXX XXXX XXXX 1111. W przypadku kiedy wewnętrzny licznik adresu osiągnie wartość graniczną, wskazuje on z powrotem na pierwszy adres strony i następuje nadpisywanie danych na już wcześniej zachowane. Aby zakończyć sekwencję zapisu danych do tablicy pamięci sygnał na wyprowadzeniu CS powinien zostać ustawiony w stan wysoki tuż po otrzymaniu ostatniego najmniej znaczącego bitu n-bajtowej danej. W przypadku gdy sygnał CS zostanie wcześniej ustawiony w stan wysoki operacja zapisu nie zostanie skompletowana. Podczas trwania sekwencji zapisu rejestr statusu może być odczytywany w celu sprawdzenia bitów WPEN, WIP, WEL, BP1 oraz BP0. Podczas trwania cyklu zapisu nie jest możliwy odczyt tablicy lokacji pamięci. Po zakończeniu tego cyklu zatrask zezwolenia zapisu jest zerowany.

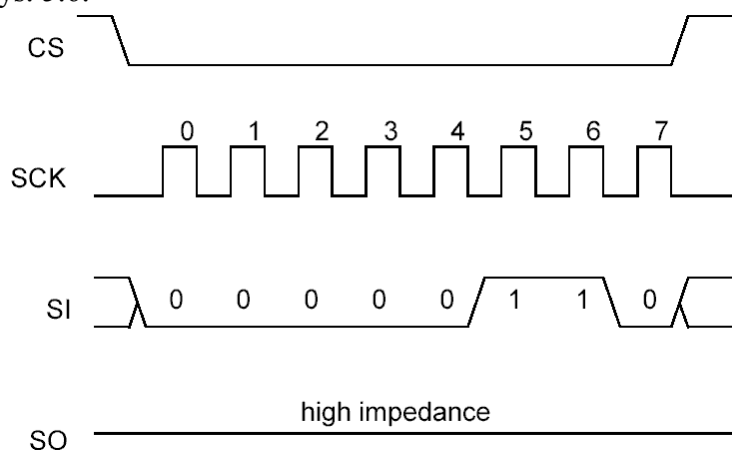


Rys. 5.4. Przebieg czasowy rozkazu WRITE dla zapisu pojedynczego bajta



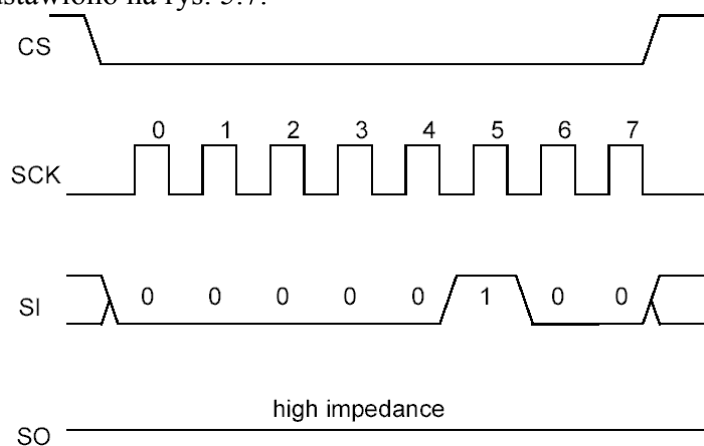
Rys. 5.5. Przebieg czasowy rozkazu WRITE dla zapisu wielu bajtów

- **Instrukcja WREN** (*Write Enable*) zezwala na zapis do pamięci. Jej przebieg czasowy pokazano na rys. 5.6.



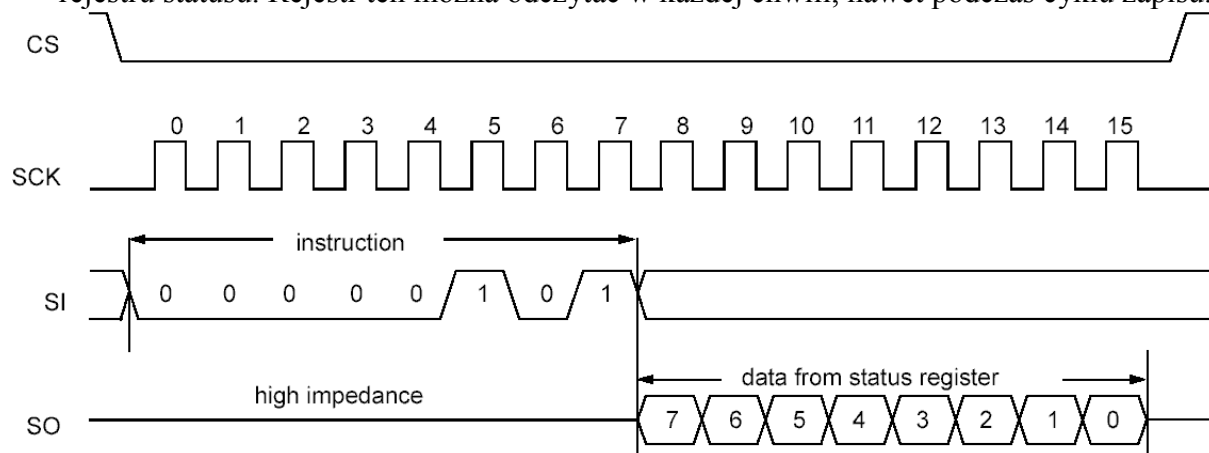
Rys. 5.6. Przebieg czasowy rozkazu WREN

- **Instrukcja WRDI** (*Write Disable*) blokuje zapis do pamięci. Przebiegi czasowe tej instrukcji przedstawiono na rys. 5.7.



Rys. 5.7. Przebieg czasowy rozkazu WRDI

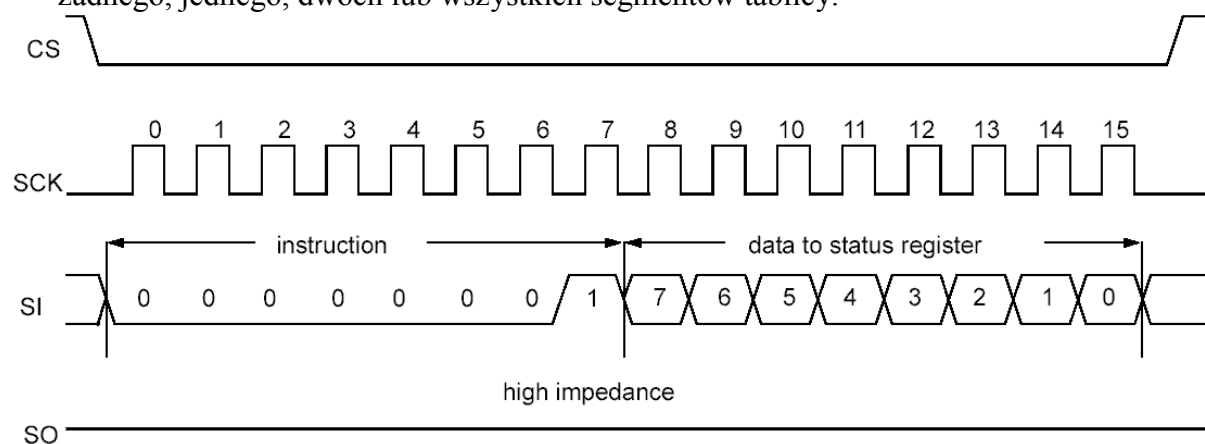
- **Instrukcja RDSR** odczytu rejestru statusu (*Read Status Register*) zapewnia dostęp do rejestru statusu. Rejestr ten można odczytać w każdej chwili, nawet podczas cyklu zapisu.



Rys. 5.8. Przebieg czasowy rozkazu RDSR

- **Instrukcja WRSR** zapisu rejestru statusu (*Write Status Register*) pozwala użytkownikowi na wybór jednego z czterech poziomów ochrony tablicy pamięci

EEPROM. Odbywa się to poprzez zapis bitów BP0 i BP1 w rejestrze statusu. Tablica pamięci podzielona jest na cztery segmenty. Użytkownik ma możliwość ochrony zapisu żadnego, jednego, dwóch lub wszystkich segmentów tablicy.



Rys. 5.9. Przebieg czasowy rozkazu WRSR

Format rejestru statusu jest następujący:

7	6	5	4	3	2	1	0
WPEN	X	X	X	BP1	BP0	WEL	WIP

gdzie:

Bit **WIP** (*Write-in-Process* – zapis w trakcie) wskazuje czy układ pamięci jest w trakcie operacji zapisu, czy nie. WIP =1 – pamięć w trakcie operacji zapisu, zaś WIP =0 – żadna operacja zapisu nie jest przeprowadzana. Bit WIP jest bitem przeznaczonym tylko do odczytu.

Bit **WEL** (*Write Enable Latch* – zatrask zezwolenia zapisu) wskazuje na status zatrasku zezwolenia zapisu. WEL =1 – zezwolenie zapisu do tablicy, WEL =0 – zakaz zapisu do tablicy. Stan tego bitu może być zawsze zmieniony poprzez polecenia WREN lub WRDI. Bit WEL jest bitem przeznaczonym tylko do odczytu.

Bity **BP0** i **BP1** (*Block Protection* – ochrona bloku) informują, który blok obecnie jest chroniony przed zapisem (patrz tabela 5.3). Stan tych bitów jest nieulotny i może być zmieniany przez użytkownika poprzez wykonanie instrukcji WRSR.

Bit **WPEN** (*Write Protect Enable* – zezwolenie ochrony zapisu) jest nieulotnym bitem dostępnym jako bit zezwalający dla wejścia WP.

Tabela 5.3. Znaczenie bitów BP0 i BP1

BP1	BP0	Adresy Tablicy Chronione przed Zapisem
0	0	brak
0	1	wyższa ćwiartka (0300H-03FFH)
1	0	wyższa połówka (0200H-03FFH)
1	1	wszystkie (0000H-03FFH)

Wejście WP (*Write Protect* – ochrona zapisu) i bit rejestru statusu WPEN służą do **sprzętowej ochrony pamięci przed zapisem**. Ochrona ta jest aktywna jeśli wyprowadzenie WP jest w stanie niskim zaś bit WPEN =1. Dotyczy ona ochrony nieulotnych bitów rejestru statusu.



Układy pamięci 25C080 zawierają **zatrask zezwolenia zapisu**. Zatrask musi być ustawiony przed tym, jak dowolna operacja zapisu zostanie „wewnętrznie” zakończona. Instrukcja WREN ustawia ten zatrask, natomiast instrukcja WRDI go zeruje. Poniżej została przedstawiona lista warunków, pod którymi nastąpi wyzerowanie zatrasku:

- włączenie zasilania,
- wykonanie instrukcji WRDI zakończone sukcesem,
- wykonanie instrukcji WRSR zakończone sukcesem,
- wykonanie instrukcji WRITE zakończone sukcesem.

Czyli przed każdym zapisem danych do pamięci należy ustawić ten zatrask instrukcją WREN.

Zastosowano następujące środki ochrony tablicy pamięci EEPROM przed przypadkowymi wpisami:

- zatrask zezwolenia zapisu jest zerowany natychmiast po włączeniu zasilania,
- w celu ustawienia zatrasku zezwolenia zapisu wymagana jest odpowiednia instrukcja,
- zatrask zezwolenia zapisu jest zerowany zaraz po zakończonym cyklu zapisu bajtu, strony lub statusu rejestru,
- sygnał na linii CS musi być ustawiony w stan wysoki po odpowiedniej liczbie cykli zegara w celu rozpoczęcia wewnętrznego cyklu zapisu,
- próba dostępu do tablicy podczas wewnętrznego cyklu zapisu jest odrzucana, zaś programowanie jest nadal kontynuowane.

W tabeli 5.4 zestawiono stan ochrony bloków pamięci i rejestru statusu w zależności od bitów WPEN i WEL oraz stanu na linii WP.

Tablica 5.4. Macierz Funkcjonalności Ochrony Zapisu

WPEN	WP	WEL	Bloki Chronione	Bloki Niechronione	Rejestr Statusu
x	x	0	chroniony	chroniony	chroniony
0	x	1	chroniony	zapisywalne	zapisywalny
1	low	1	chroniony	zapisywalne	chroniony
x	high	1	chroniony	zapisywalne	zapisywalny

Układ pamięci 25C080 po włączeniu zasilania przyjmuje następujący stan:

- układ jest w trybie czuwania, mały pobór mocy CS=1,
- zatrask zezwolenia zapisu jest wyzerowany,
- wyjście SO jest w stanie wysokiej impedancji,
- przejście ze stanu wysokiego do niskiego na linii CS powoduje wejście układu w stan aktywny.

### 5.1.2. Szeregowe pamięci EEPROM z interfejsem Microwire

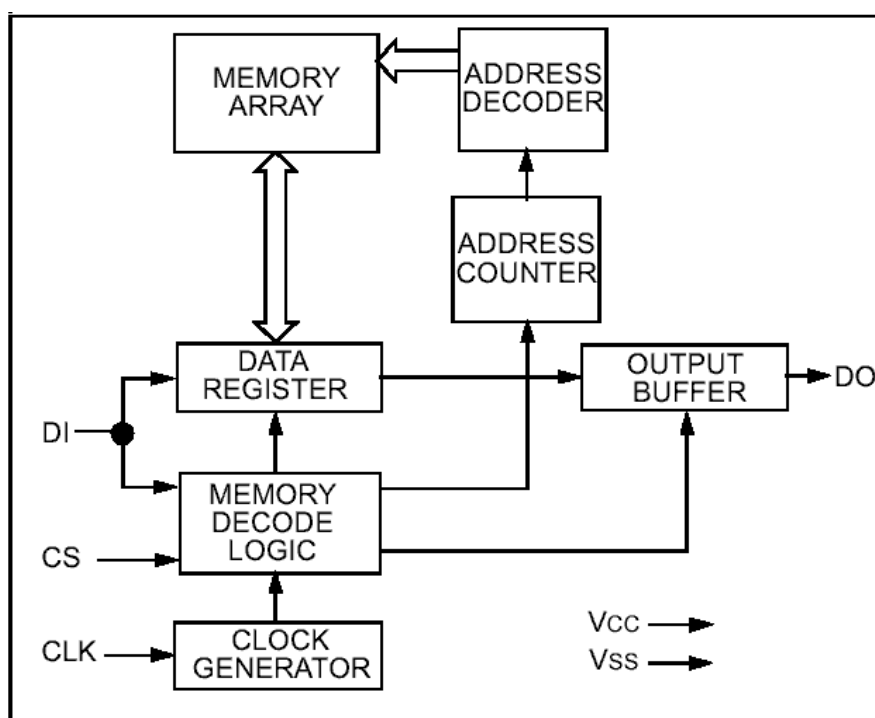
Przykładem pamięci EEPROM tego typu jest rodzina układów 93x4/6/7/8 produkowana między innymi przez takie firmy, jak *Microchip*, *National Semiconductors*, *Catalyst*, *STMicroelectronics*. Protokół wymiany danych bazuje na standardzie Microwire, który nie jest zgodny z żadnym standardem międzynarodowym. Jednak wszystkie pamięci tej rodziny wykorzystują identyczny zbiór poleceń dwójkowych potrzebnych do kasowania, programowania i odczytu ich zawartości. Zatem są układami popularnymi, często

stosowanymi w mse. Są dostępne w odmianach o pojemności 1Kbit (128x8 lub 64x16), 2Kbit (256x8 lub 128x16), 4Kbit (512x8, 256x16), 8Kbit (1024x8 lub 512x16) i 16Kbit (2048x8 lub 1024x16).

Pamięci tego typu zostaną przedstawione na przykładzie układu 93C46B firmy Microchip. Jest to pamięć o pojemności 1Kbit i organizacji 16-bitowej (64x16 bitów). Jest umieszczona w obudowie ośmionóżkowej.

Cechy układu:

- pojedyncze źródło zasilania + 5V,
- technologia Low Power CMOS: 1 mA prąd pracy (typowo), 1μA prąd czuwania (maksymalnie),
- organizacja 64x16 bitów,
- wewnętrzne taktowane dla cykli ERASE i WRITE,
- automatyczne ustawienie stanu ERAL przed cyklem WRAL,
- układ protekcji danych w trakcie włączenia/wyłączenia napięcia zasilania,
- standardowy 3-przewodowy szeregowy interfejs,
- wystawianie sygnału statusu urządzenia podczas cykli ERASE/WRITE,
- posiada sekwencyjną funkcję READ,
- zapewnione 1000000 cykli kasowania i zapisu,
- zachowanie danych w pamięci > 200 lat,
- obudowy: 8-pinowe PDIP/SOIC i 8-pinowa TSSOP.

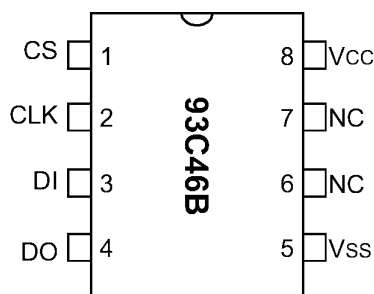


Rys. 5.10. Schemat blokowy układu 93C46B

Na rys. 5.10 pokazano schemat blokowy tej pamięci. Składa się ona z dwóch części. Pierwszą część stanowi interfejs szeregowy, w którego skład wchodzi układ kontrolny (*Memory Decode Logic*), rejestr danych (*Data Register*), bufor wyjściowy (*Output Buffer*) i generator sygnału zegarowego (*Clock Generator*). Drugą część to „właściwa” pamięć danych. Składa się ona z licznika adresu (*Address Counter*), dekodera adresu (*Address Decoder*) i pamięci EEPROM (*Memory Array*).

Tabela 5.5. Opis funkcji pinów układu 93C46B (rys. 5.11)

Nazwa	Funkcja
CS	Wybór urządzenia
CLK	Zegar dla danych szeregowych
DI	Wejście danych szeregowych
DO	Wyjście danych szeregowych
Vss	Masa
NC	Bez połączenia
Vcc	Napięcie zasilania



Rys. 5.11. Wyprowadzenia układu 93C46B

Zastosowania i funkcje pinów układu są następujące:

- Wysoki poziom sygnału na **linii CS** wybiera (uaktywnia) urządzenie, natomiast niski poziom dezaktywuje urządzenie i wymusza tryb oczekiwania. Dezaktywacja w trakcie cyklu programującego, nie spowoduje jego przerwania – cykl zostanie kontynuowany, aż do zakończenia, niezależnie od stanu linii wejściowej CS. Przejście linii CS w stan niski podczas cyklu programującego, spowoduje przejście urządzenia do trybu oczekiwania, zaraz po zakończeniu tego cyklu. Sygnał CS musi być w stanie niskim przynajmniej przez 250ns ( $T_{CSL}$ ) pomiędzy kolejnymi instrukcjami. Jeżeli CS jest w stanie niskim, wewnętrzny układ kontrolny jest trzymany w stanie RESET.
- **Zegar CLK** jest używany do synchronizowania komunikacji pomiędzy urządzeniem sterującym, a układem 93C46B. Bity kodu, adresu i danych wejściowych są wprowadzane, a bity danych wyjściowych wyprowadzane **narastającym zboczem** zegarowym. Sygnał CLK może być wstrzymany w każdym momencie transmisji, po czym może nastąpić jego kontynuacja. Właściwość ta pozwala na swobodne przygotowanie kodu programującego, adresu i danych przez mk. Stan linii CLK jest nieistotny, jeżeli CS jest w stanie niskim.
- **Wejście danych DI** jest używane do wprowadzania bitu START, kodów rozkazów, adresów i danych. Dane te są synchronizowane sygnałem zegarowym CLK.
- **Wyjście danych DO** używa się do wyprowadzania danych wyjściowych, zgodnie z taktami sygnału zegarowego CLK. Linie tę stosuje się do odczytu statusu READY/BUSY podczas cyklu ERASE i WRITE. Informacja ta jest dostępna na pinie DO, jeżeli linia CS została ustawiona w stan wysoki, po stanie niskim trwającym przez czas  $T_{CSL}$ , i zainicjowano operacje ERESE lub WRITE.

Tabela 5.6. Lista instrukcji sterujących układ 93C46B

Instrukcja	SB	Kod	Adres	Dane wejściowe	Dane wyjściowe	Cykle zegara CLK
<b>ERASE</b>	1	11	A5 A4 A3 A2 A1 A0	--	(RDY/BSY)	9
<b>ERAL</b>	1	00	1 0 X X X X	--	(RDY/BSY)	9
<b>EWDS</b>	1	00	0 0 X X X X	--	High-Z	9
<b>EWEN</b>	1	00	1 1 X X X X	--	High-Z	9
<b>READ</b>	1	10	A5 A4 A3 A2 A1 A0	--	D15-D0	25
<b>WRITE</b>	1	01	A5 A4 A3 A2 A1 A0	D15 – D0	(RDY/BSY)	25
<b>WRAL</b>	1	00	0 1 X X X X	D15 – D0	(RDY/BSY)	25

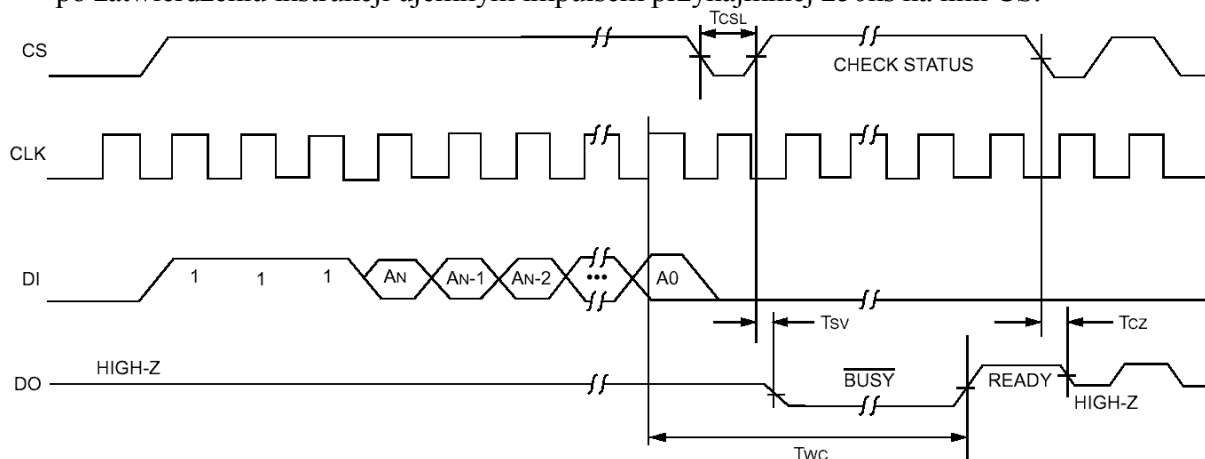
Instrukcje, adresy i dane są wprowadzane na pin DI na narastające zbocze sygnału zegara CLK dopiero jak zostanie wykryty bit START. Należy zawsze wprowadzić odpowiednią ilość impulsów zegarowych dla danego rozkazu (patrz tabela 5.6). Przed pojawieniem się tego bitu dane wejściowe na linii DI są ignorowane. Linia DO jest zazwyczaj w stanie wysokiej impedancji, z wyjątkiem odczytu danych z urządzenia, oraz sprawdzania statusu READY/BUSY podczas operacji programowania. Podczas operacji ERASE/WRITE status ten sprawdza się przez odpytywanie pinu DO. Stan niski na tym pinie oznacza, że programowanie jest w toku. Pojawienie się stanu wysokiego na linii DO świadczy o zakończeniu operacji kasowania/zapisu i gotowości układu do przyjęcia kolejnego rozkazu. Gdy na linii CS pojawi się opadające zbocze to pin DO wchodzi w stan wysokiej impedancji.

Podczas włączenia zasilania (*power-up*), wszystkie tryby programowania, są zablokowane dopóki Vcc nie osiągnie poziomu większego niż 3,8V. Podczas wyłączenia napięcia zasilania (*power-down*), obwód **ochrony danych** zatrzymuje wszystkie tryby, gdy Vcc spadnie poniżej 3,8V.

Po włączeniu zasilania, urządzenie automatycznie znajduje się w **trybie EWDS**. (blokada zapisu i kasowania pamięci). Stąd aby mogła być wykonana instrukcja ERASE lub WRITE, najpierw należy odblokować pamięć wysyłając do układu instrukcję EWEN. Odczyt z pamięci jest zawsze możliwy.

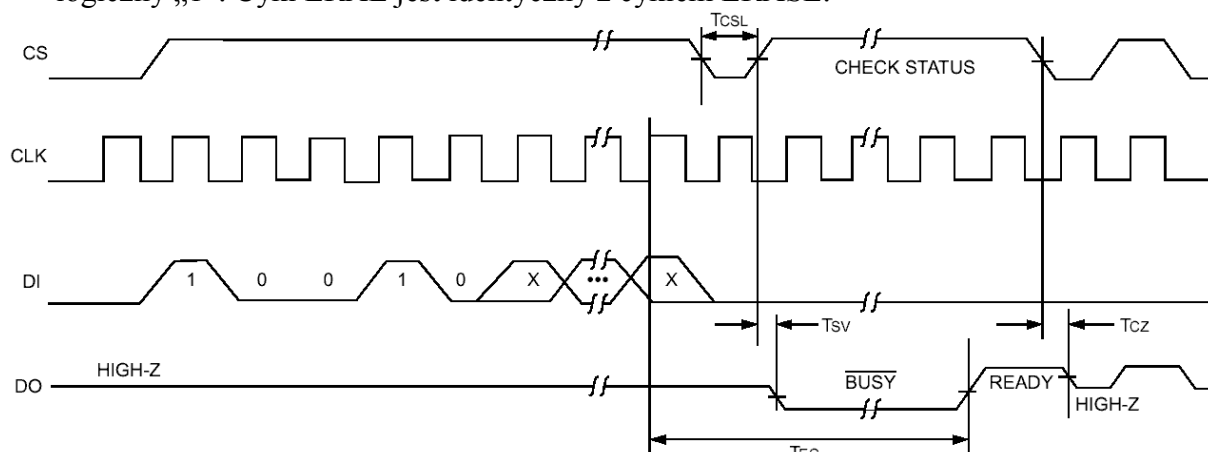
Zastosowanie instrukcji zawartych w tabeli 5.6 jest następujące:

- **Instrukcja ERASE** wymusza na wszystkich bitach danych o podanym adresie, logiczny stan "1". Cykl rozpoczyna się narastającym zboczem zegarowym, ostatniego bitu adresu. Linia DO wskazuje status READY/BUSY urządzenia, jeżeli na linii CS jest stan wysoki, po zatwierdzeniu instrukcji ujemnym impulsem przynajmniej 250ns na linii CS.



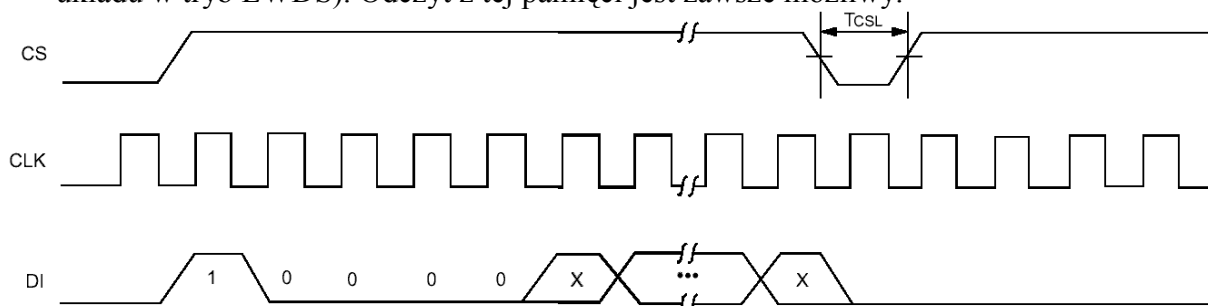
Rys. 5.12. Przebiegi czasowe dla instrukcji ERASE układu 93C46B

- **Instrukcja ERAL** (usuń wszystko) kasuje całą pamięć, ustawiając wszystkie bity w stan logiczny „1”. Cykl ERAL jest identyczny z cyklem ERASE.



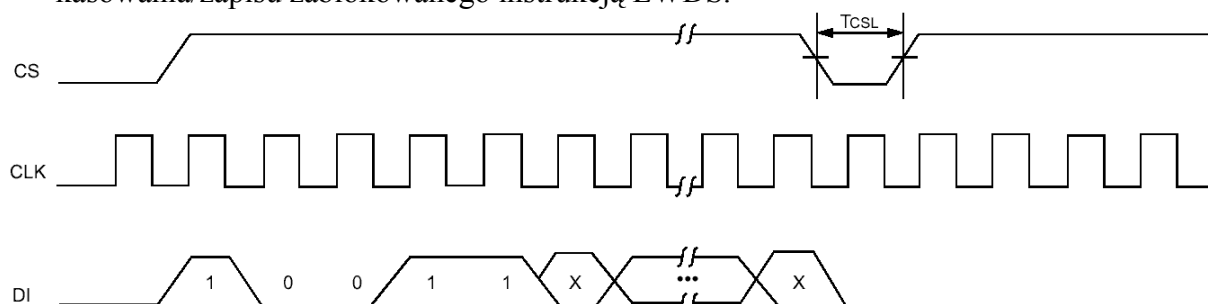
Rys. 5.13. Przebiegi czasowe dla instrukcji ERAL układu 93C46B

- **Instrukcja EWDS** powoduje zablokowanie kasowania/zapisu do pamięci (wprowadzenie układu w tryb EWDS). Odczyt z tej pamięci jest zawsze możliwy.



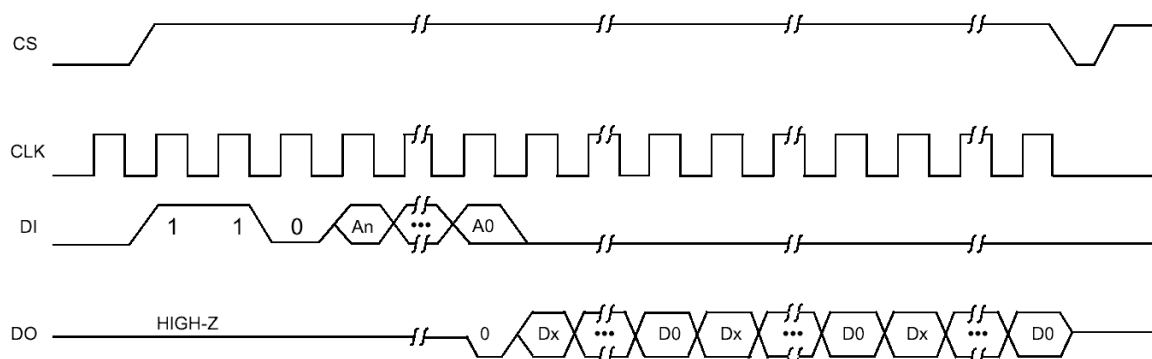
Rys. 5.14. Przebiegi czasowe dla instrukcji EWDS układu 93C46B

- **Instrukcja EWEN** odblokowuje możliwość kasowania/zapisu. Powinno ją się wykonać po włączeniu zasilania, aby wyjść z trybu EWDS lub w celu odblokowania kasowania/zapisu zablokowanego instrukcją EWDS.



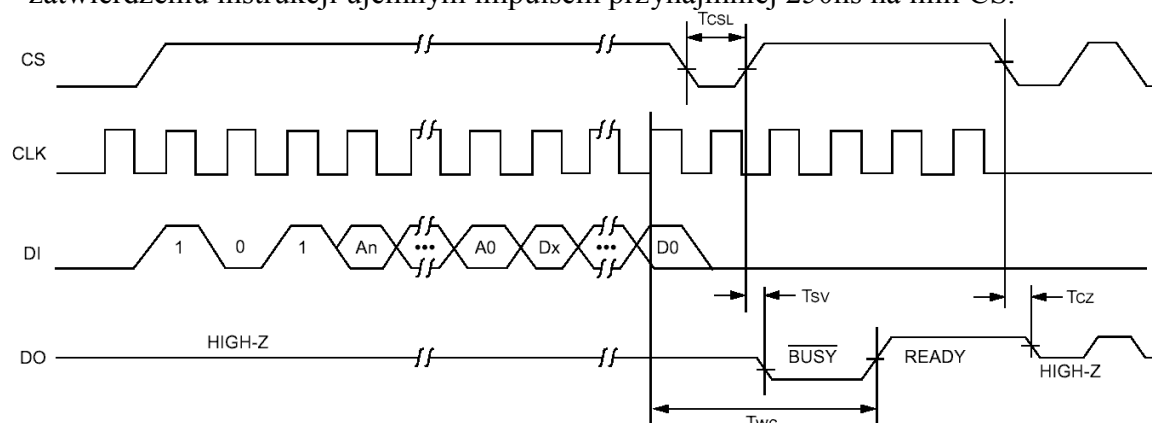
Rys. 5.15. Przebiegi czasowe dla instrukcji EWEN układu 93C46B

- **Instrukcja READ** (czytaj) wysyła szeregowo dane spod zaadresowanego obszaru pamięci, na pin DO. Bit „stucznego zera” rozpoczyna 16-bitowe słowo wyjściowe. Bity danych wyjściowych są wystawiane na narastające zbocze zegara CLK i są utrzymywane przez określony czas opóźnienia ( $T_{PD}$ ). Sekwencyjne czytanie jest możliwe jeżeli linia CS jest trzymana na wysokim poziomie.



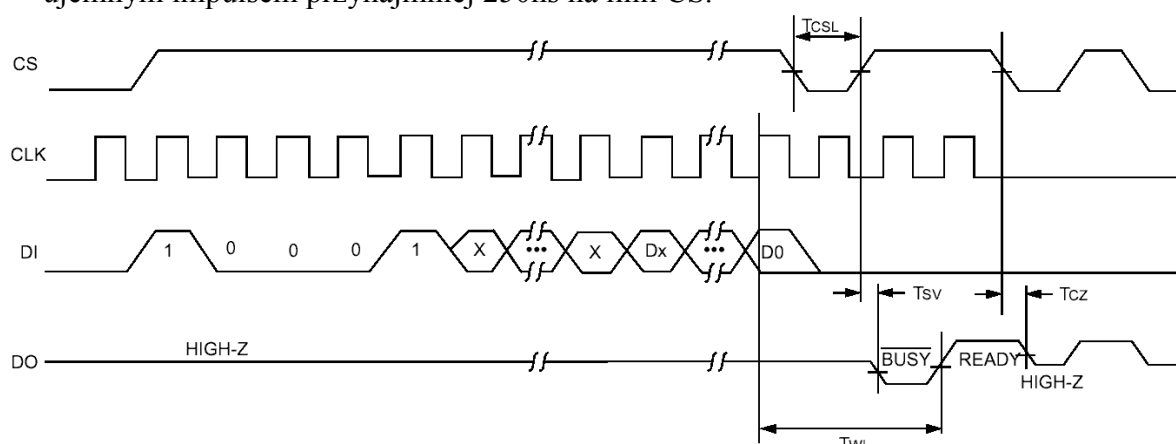
Rys. 5.16. Przebiegi czasowe dla instrukcji READ układu 93C46B

- Po instrukcji **WRITE** (wpisz) przesyłanych jest 16-bitów danych, które zostają wpisane pod określony adres. Po ostatnim bicie danych wprowadzonych na pin DI, rozpoczyna się tryb samo-taktowanego cyklu auto-usuwania (*auto-erase*) i programowania. Linia DO wskazuje status READY/BUSY urządzenia, jeżeli na linii CS jest stan wysoki, po zatwierdzeniu instrukcji ujemnym impulsem przynajmniej 250ns na linii CS.



Rys. 5.17. Przebiegi czasowe dla instrukcji WRITE układu 93C46B

- Instrukcja **WRAL** (WRITE ALL) wypełnia całą pamięć daną określoną w komendzie. Cykl WRAL jest całkowicie samo-taktowalny i zaczyna się wraz z narastającym zboczem ostatniego bitu danych. Komenda WRAL zawiera automatyczny cykl ERAL, dlatego instrukcja WRAL nie wymaga, aby wcześniej wykonano instrukcję ERAL. Kasowanie/zapis pamięci muszą być odblokowane. Linia DO wskazuje status READY/BUSY urządzenia, jeżeli na linii CS jest stan wysoki, po zatwierdzeniu instrukcji ujemnym impulsem przynajmniej 250ns na linii CS.



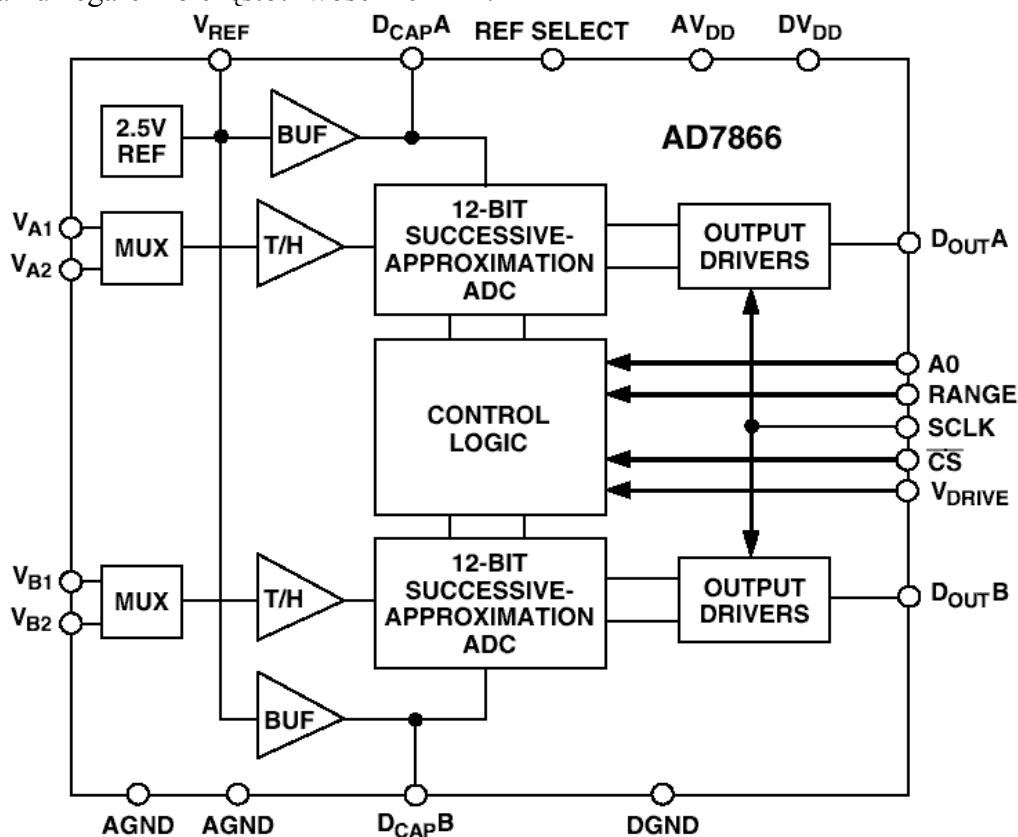
Rys. 5.18. Przebiegi czasowe dla instrukcji WRAL układu 93C46B

## 5.2. Zewnętrzne przetworniki analogowo-cyfrowe (A/C)

Jak wspomniano w podrozdziale 2.3.2 przetworniki umieszczone w strukturze mk nie posiadają rozdzielczości większej niż 10-bitowej oraz charakteryzują się stosunkowo dużym błędem całkowitym. Stąd, gdy zależy nam na dokładnych pomiarach napięcia musimy korzystać ze znacznie lepszych zewnętrznych przetworników A/C, które z reguły wyposażone są w interfejs SPI. Przetworniki te mają rozdzielczość od 8 bitów, aż do 24 bitów (typowe rozdzielczości to: 8, 10, 12, 14, 16, 24 bity). Przy czym przetworniki o większej rozdzielczości korzystają z metody sigma-delta, pozostałe przeważnie bazują na metodzie SAR.

Zewnętrzne przetworniki A/C sterowane interfejsem SPI zostaną przedstawione na przykładzie układu **AD7866** firmy Analog Devices.

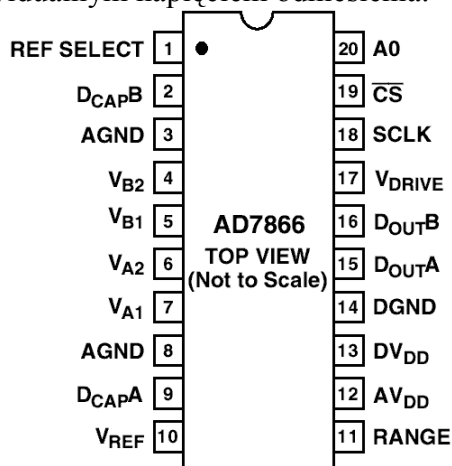
AD7866 jest szybkim, o niskim poborze mocy, podwójnym, 12-bitowym przetwornikiem analogowo-cyfrowym z pojedynczym zasilaniem, operującym w zakresie napięć zasilania od 2,7V do 5,25V. Działając z zasilaniem 5V lub 3V AD7866 jest zdolny do osiągnięcia prędkości transmisji danych (przepustowości) 1MSPS (milion próbki na sekundę) przy taktowaniu zegarem o częstotliwości 20MHz.



Rys. 5.19. Schemat blokowy układu AD7866 firmy Analog Devices

Układ AD7866 zawiera w sobie dwa śledząco-podtrzymujące wzmacniacze, dwa przetworniki analogowo-cyfrowe działające na zasadzie SAR i dwa szeregowe wyjścia oddzielne dla każdego przetwornika, przez które są wysyłane dane wyjściowe (rys. 5.19). Urządzenie znajduje się w 20-nóżkowej obudowie typu TSSOP (rys. 5.20). Szeregowe impulsy zegara wejściowego pobierają dane z urządzenia oraz zapewniają źródło impulsów zegarowych potrzebnych do konwersji dla każdego z przetworników analogowo-cyfrowych.

Zakres pomiarowy można ustawić w zakresie od 0V do  $V_{REF}$  lub do  $2 \cdot V_{REF}$ . Układ AD7866 posiada własne wewnętrzne napięcie odniesienia 2,5V, które może w razie potrzeby zostać zastąpione przez zewnętrzne napięcie odniesienia. Dodatkowo, każdy z przetworników może być zasilany osobnym, indywidualnym napięciem odniesienia.



Rys. 5.20. Rozmieszczenie pinów układu AD7866

Układ AD7866 ma możliwość zmniejszenia poboru mocy w stanie spoczynku poprzez wprowadzenie układu w stan uśpienia, co pozwala na znaczną oszczędność energii pomiędzy konwersjami. W trakcie konwersji przy zasilaniu 3V i przepustowości 1MSPS, układ pobiera maksymalnie 3,8mA. Przy zasilaniu 5V i przepustowości 1MSPS, pobór prądu wynosi do 4,8mA.

W tabeli 5.7 zawarto opis funkcji pełnionych przez piny układu AD7866.

Tabela 5.7. Opis funkcji pinów układu AD7866

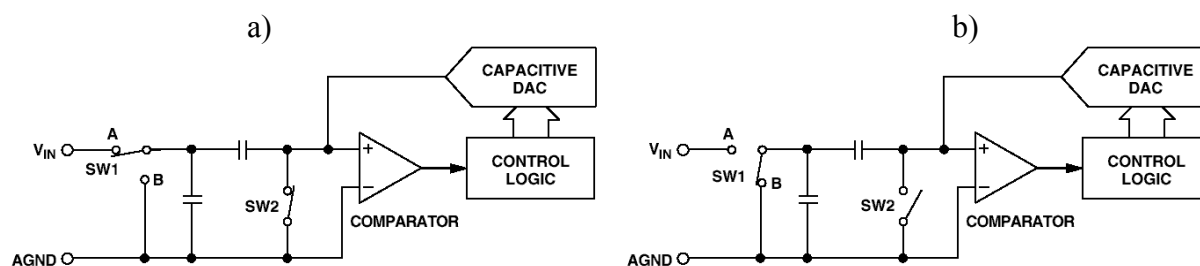
Numer pinu	Symbol	Funkcja
1	REF SELECT	Wybór wewnętrznego lub zewnętrznego napięcia odniesienia. Jeżeli nóżka jest przyłączona do GND, wówczas wewnętrzne napięcie 2,5V jest używane jako napięcie odniesienia dla obu przetworników (ADC A i ADC B). Dodatkowo, do nóżek $V_{REF}$ , $D_{CAPA}$ , $D_{CAPB}$ muszą być przyłączone w odpowiedni sposób kondensatory odsprężające. Jeżeli na nóżce REF SELECT ustawiona jest logiczna jedynka, wówczas używane jest zewnętrzne napięcie odniesienia na $V_{REF}$ . W takim przypadku do pinów $D_{CAPA}$ i $D_{CAPB}$ są podłączone te kondensatory. Gdy nóżka $V_{REF}$ jest zwarta do AGND i na REF SELECT ustawiony jest stan niski, to każdy z przetworników (ADC A i ADC B) może pracować z własnym napięciem odniesienia podawanym odpowiednio poprzez nóżki $D_{CAPA}$ , $D_{CAPB}$ .
2, 9	$D_{CAPA}$ , $D_{CAPB}$	Do tych nóżek przyłącza się kondensatory sprężające do masy. Z tych pinów można pobrać wewnętrzne napięcie odniesienia i przyłożyć do reszty systemu. W zależności od polaryzacji na REF SELECT i od konfiguracji $V_{REF}$ , nóżki mogą także być użyte do dostarczenia oddzielnych napięć odniesienia dla każdego z przetworników. Zakres zewnętrznego odniesienia jest zależny od wybranego zakresu analogowego wejścia.



3, 8	AGND	Analogowa masa dla wszelkich analogowych sygnałów wejściowych i zewnętrznych sygnałów odniesienia. Napięcia na AGND i DGND powinny być na tym samym potencjale, i nie mogą nawet na chwilę różnić się więcej niż 0,3V.
4, 5	$V_{B2}$ , $V_{B1}$	Analogowe wejścia przetwornika B (ADC B). Analogowe kanały wejściowe. Zakres na wejściu dla każdego kanału jest od 0V do $V_{REF}$ lub do $2 * V_{REF}$ w zależności od polaryzacji pinu RANGE w trakcie występowania opadającego zbocza sygnału na CS.
6, 7	$V_{A2}$ , $V_{A1}$	Analogowe wejścia przetwornika A (ADC A). Analogowe kanały wejściowe. Zakres na wejściu dla każdego kanału jest od 0V do $V_{REF}$ lub do $2 * V_{REF}$ w zależności od polaryzacji pinu RANGE w trakcie występowania opadającego zbocza sygnału na CS.
10	$V_{REF}$	Nóżka do której podłącza się napięcie odniesienia lub nóżka służąca do wyboru zewnętrznego odniesienia. Wymaga ona dołączenia kondensatora odprężającego. Nominalne napięcie odniesienia wynosi 2,5V i jest wystawione na tym pinie. Jeżeli wewnętrzne napięcie odniesienia ma zostać wykorzystane w systemie, to musi ono zostać pobrane z pinu $D_{CAPA}$ albo $D_{CAPB}$ . Nóżka ta jest także używana w powiązaniu z nóżką REF SELECT, gdy dostarczane jest zewnętrzne napięcie odniesienia do AD7866.
11	RANGE	Nóżka zakresu wejścia analogowego i wyboru kodowania danych na wyjściu. Polaryzacja na tej nóżce określa jaki będzie wejściowy zakres napięć analogowych kanałów wejściowych AD7866 i także określa wybór sposobu kodowania danych na wyjściu. Na opadającym zboczach CS sprawdzana jest polaryzacja na nóżce, aby określić zakres wejścia analogowego dla następnej konwersji. Jeżeli z nóżką związany jest stan niski, to zakres analogowego wejścia określony jest jako 0V do $V_{REF}$ i kodowanie na wyjściu ustawiane jest jako bezpośrednie binarne (dla następnej konwersji). Jeżeli z nóżką związany jest stan wysoki gdy CS przechodzi do stanu niskiego, to zakres analogowego wejścia wynosi $2 * V_{REF}$ i kodowanie na wyjściu ustawiane jest na kod z uzupełnieniem do 2. Jednakże, jeżeli po opadającym zboczach CS poziom na nóżce RANGE zmieni się na opadającym zboczach ósmego sygnału zegara (SCLK), wówczas kodowanie na wyjściu zmieni się na drugą opcję bez żadnej zmiany w zakresie analogowego wejścia.
12	$AV_{DD}$	Analogowe napięcie zasilania, 2,7V do 5,25V. Jest to jedyne napięcie zasilania dla wszystkich analogowych obwodów w AD7866. Idealnie, napięcia $AV_{DD}$ i $DV_{DD}$ powinny być na tym samym potencjale i nie mogą różnić się o więcej niż 0,3V nawet w przypadku wystąpienia stanów przejściowych. Zasilanie te powinno być odniesione do AGND.
13	$DV_{DD}$	Cyfrowe napięcie zasilania, 2,7V do 5,25V. Jest to napięcie zasilania dla wszystkich cyfrowych obwodów w AD7866. Idealnie, napięcia $AV_{DD}$ i $DV_{DD}$ powinny być na tym samym potencjale i nie mogą różnić się o więcej niż 0,3V nawet w

		przypadku wystąpienia stanów przejściowych. Zasilanie te powinno być odniesione do DGND.
14	DGND	Cyfrowa masa. Punkt odniesienia dla wszystkich cyfrowych obwodów w AD7866. Napięcia AGND i DGND powinny (idealnie) być na tym samym potencjale, i nie mogą nawet chwilowo różnić się więcej niż o 0.3V.
15,16	D <sub>OUTA</sub> , D <sub>OUTB</sub>	Szeregowe wyjście danych. Dane wyjściowe dostarczane są na tą nóżkę w sposób szeregowy. Bity są podawane na opadającym zboczu sygnału SCLK. Dane pojawiają się jednocześnie na obu nóżkach z obu przetworników. Strumień danych składa się z jednego prowadzącego zera, po którym następują trzy bity STATUS, a następnie 12 bitów danych konwersji. W danych najpierw występuje bit MSB. Jeżeli na CS jest stan niski przez następne 16 cykli SCLK po tym, gdy dane z konwersji zostały wyprowadzone przez D <sub>OUTA</sub> lub przez D <sub>OUTB</sub> , wówczas dane z innego przetwornika są podawane na nóżkę D <sub>OUT</sub> . Pozwala to, by dane uzyskane z jednoczesnych konwersji obu przetworników były zbierane szeregowo jedno po drugim tylko na D <sub>OUTA</sub> lub tylko na D <sub>OUTB</sub> .
17	V <sub>DRIVE</sub>	Wejście zasilania logiki. Napięcie przyłożone do tej nóżki determinuje (określa) na jakim napięciu będzie działać interfejs, czy na 3V, czy na 5V. To zasilanie powinno być odniesione do DGND.
18	SCLK	Zegar szeregowy. Szeregowe wejście zegara dostarcza sygnał SCLK taktujący transmisję danych. Zegar ten jest także używany jako źródło sygnałów zegarowych dla procesu konwersji.
19	CS	Stan niski na tym wejściu uaktywnia układ.
20	A0	Wejście służące do wyboru kanału multiplekserów. Wejście te używane jest do wyboru pary kanałów do jednoczesnej konwersji, np. kanał 1 obu przetworników (A i B) lub kanał 2 obu przetworników (A i B). Stan na tej nóżce sprawdzany jest na opadającym zboczu CS. Jeżeli jest stan niski, następna konwersja wykonana będzie na kanale 1 każdego przetwornika; jeżeli jest wysoki, następna konwersja wykonana będzie na kanale 2 każdego przetwornika.

AD7866 posiada dwa przetworniki analogowo-cyfrowe. Każdy przetwornik składa się z układu kontroli logiki, SAR i pojemnościowego przetwornika cyfrowo-analogowego C/A, które są używane do dodawania i odejmowania stałych wielkości ładunków z próbkującego kondensatora aby sprowadzić układ porównujący z powrotem do stanu równowagi. Rys. 5.21a pokazuje przetwornik analogowo-cyfrowy podczas fazy akwizycji (pobierania danych). SW2 jest zwarty i SW1 jest w pozycji A, układ porównujący jest utrzymywany w stanie równowagi, a próbkujący kondensator pobiera sygnał na V<sub>A1</sub>.



Rys. 5.21. Fazy pracy przetwornika A/C w układzie AD7866: a) faza akwizycji danych, b) faza konwersji

Gdy przetwornik analogowo-cyfrowy rozpoczyna konwersję (patrz rys. 5.21b), SW2 otwiera się, a SW1 przestawi się w pozycję B powodując, że komparator będzie niezrównoważony. Układ kontroli logiki i pojemnościowy przetwornik cyfrowo-analogowy są wykorzystywane do dodania lub odjęcia stałych wielkości ładunku z próbkującego kondensatora w celu sprowadzenia komparatora z powrotem do stanu równowagi. Gdy komparator jest ponownie zrównoważony, konwersja kończy się. Układ kontroli logiki generuje kod wyjściowy.

W tabeli 5.8 pokazano sposoby wyboru napięć odniesienia dla układu AD7866.

Tabela 5.8. Sposoby wyboru napięć odniesienia dla układu AD7866.

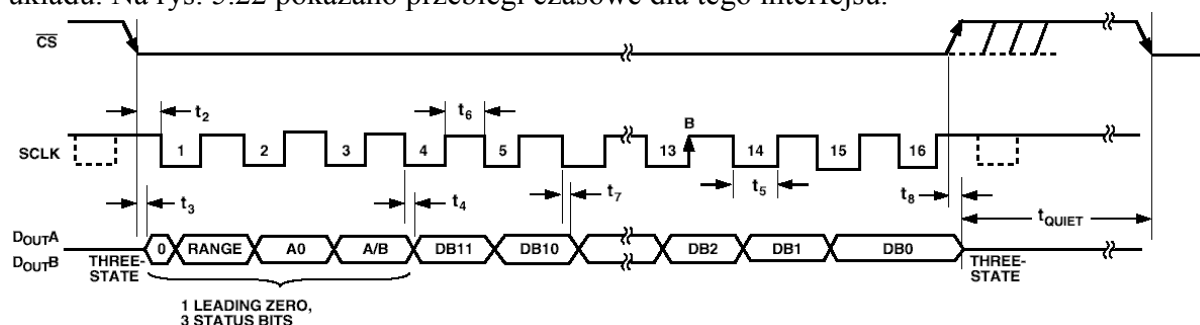
Opcja odniesienia	REF SELECT	$V_{REF}^1$	$D_{CAPA}$ i $D_{CAPB}^2$
wewnętrzne	niski	kondensator sprzęgający	kondensator sprzęgający
zewnętrzne przez $V_{REF}$	wysoki	zewnętrzne odniesienie	kondensator sprzęgający
zewnętrzne przez $D_{CAPA}$ i/lub $D_{CAPB}$	niski	AGND	zewnętrzne odniesienie A i/lub zewnętrzne odniesienie B

UWAGI:

<sup>1</sup> Zalecana wartość kondensatora sprzęgającego = 100 nF.

<sup>2</sup> Zalecana wartość kondensatora sprzęgającego = 470 nF.

Układ AD7866 jest sterowany za pomocą interfejsu SPI. **Sygnal zegara** tego interfejsu **dostarcza sygnał zegarowy potrzebny do konwersji**, jak i **steruje odbiorem danych** z układu. Na rys. 5.22 pokazano przebiegi czasowe dla tego interfejsu.



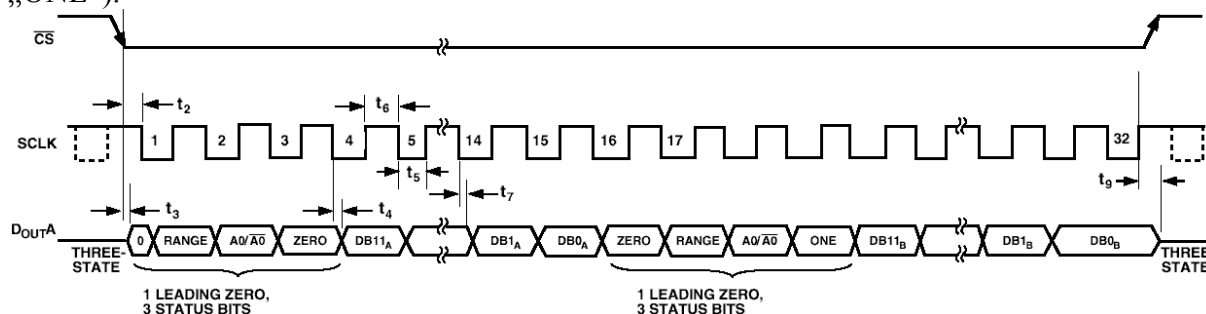
Rys. 5.22. Przebiegi czasowe interfejsu SPI

Sygnal CS inicjuje transfer danych i konwersję. Opadające zbocze tego sygnału rozpoczyna konwersję, która wymaga 16 impulsów zegarowych SCLK. Linie danych  $D_{OUTA}$  i  $D_{OUTB}$  wychodzą ze stanu wysokiej impedancji. Najpierw pojawia się na nich wiodące zero, po którym występują trzy bity statusu:

- RANGE „0” – konwersja dla zakresu od 0 do  $V_{REF}$ , „1” – konwersja dla zakresu  $2 * V_{REF}$ ,
- A0 „0” – dane z kanału 1, „1” – dane z kanału 2,
- A/B „0” – dane z przetwornika ADC A, „1” – dane z przetwornika ADC B.

Ostatni bit jest szczególnie użyteczny w przypadku transmitowania rezultatów konwersji obu przetworników przez jedno wyjście danych. Następnie następuje przesłanie wyniku konwersji od najstarszego bitu DB11 do najmłodszego DB0. Należy pamiętać, że wraz z przesyłaniem danych **trwa konwersja** napięcia wejściowego na wartość cyfrową zgodnie z prędkością sygnału zegarowego SCLK. Zatem zmieniając częstotliwość sygnału SCLK **sterujemy prędkością konwersji**. Dane są wysyłane z wyjścia danych na opadające zbocze sygnału zegarowego. Po 16 impulsach zegarowych proces konwersji i transferu danych jest kompletny i może zostać zakończony poprzez wprowadzenie linii CS w stan wysoki. Linie danych ponownie znajdują się w stanie wysokiej impedancji.

Jak wspomniano z jednego wyjścia danych przypisanego do danego przetwornika można odczytać wyniki konwersji z obu przetworników. W tym przypadku po kompletnej konwersji (16 impulsów zegarowych) sygnał CS nadal musi pozostać w stanie niskim. Następnie należy podać na wejście zegarowe kolejne 16 impulsów, co łącznie daje 32 impulsy zegarowe. Na rys. 5.23 pokazano przebiegi czasowe dla takiego przypadku. Zakłada się, że dane są czytane z wyjścia D<sub>OUTA</sub>, stąd w pierwszym 16-bitowym łańcuchu bit statusu A/B przyjmuje wartość „ZERO”, co oznacza iż dane zawarte w nim są z przetwornika ADC A. Drugi 16-bitowy łańcuch składa się z danych pochodzących od przetwornika ADC B (bit statusu A/D wynosi „ONE”).

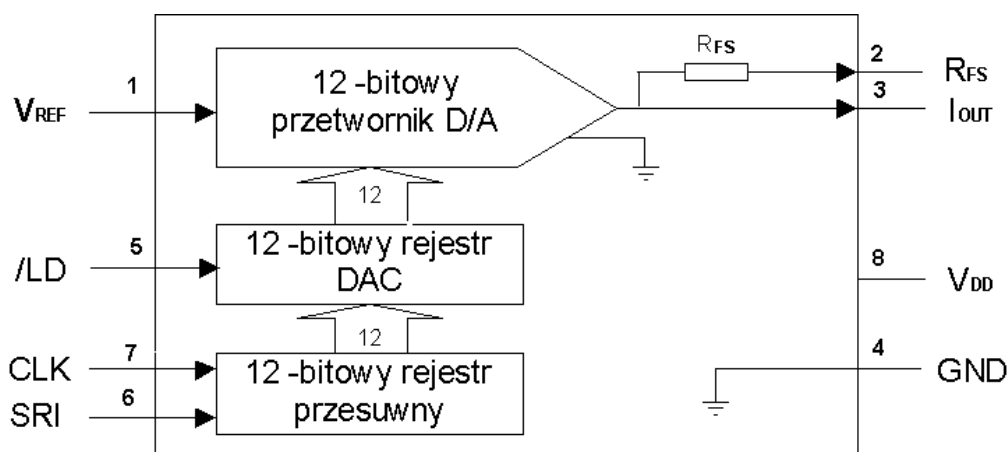


Rys. 5.23. Przebiegi czasowe interfejsu SPI dla przypadku odczytu danych z obu przetworników przez jedno wyjście danych

### 5.3. Zewnętrzne przetworniki cyfrowo- analogowe (C/A)

Również oferowana jest przez producentów szeroka gama przetworników C/A z interfejsem SPI. Dokonują one konwersji kodu cyfrowego na wartość napięcia (właściwie prądu) wyjściowego. Najczęściej konwersja odbywa się z wykorzystaniem drabinki R-2R. Spotykane rozdzielczości przetworników C/A to 8, 10, 12, 14 i 16 bitów.

Przetworniki te zostaną przedstawione na przykładzie układu **DAC8043** firmy Analog Devices (jego odpowiednikiem jest LCT8043 firmy Linear Technology). Układ ten jest 12-bitowym przetwornikiem C/A wykonanym w technologii CMOS, umieszczonym w 8-nóżkowej obudowie. Operuje on na pojedynczym napięciu zasilania +5V.



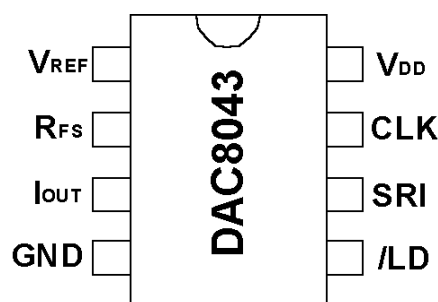
Rys. 5.24. Schemat blokowy przetwornika C/A DAC8043

Układ składa się z (rys. 5.24):

- 12-bitowego szeregowego rejestru z wejściem szeregowym i wyjściem równoległym,
- 12-bitowego rejestru zatraskowego DAC z wejściem i wyjściem równoległym,
- 12-bitowego przetwornika C/A (drabinki R-2R),
- układu kontrolnego.

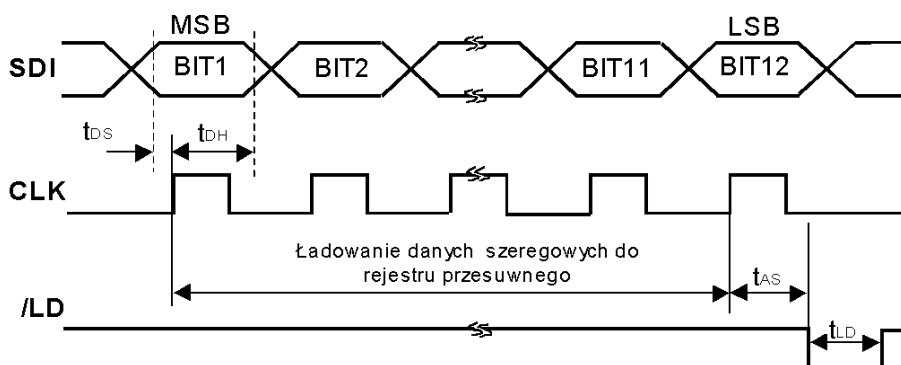
Układ DAC8043 posiada następujące cechy:

- 12-bitowy przetwornik C/A w 8-nóżkowej obudowie (rys. 5.25),
- szybki port szeregowy danych,
- podwójnie buforowane dane,
- niskie błęd nieliniowości i różnicowy błąd nieliniowości  $\pm \frac{1}{2}$  LSB,
- maksymalny błąd wzmocnienia  $\pm 1$  LSB,
- niska wartość zmian rezystancji względem zmian temperatury 5 ppm/°C,
- napięcie referencyjne (odniesienia)  $V_{REF}$  względem GND  $\pm 25V$ ,
- prąd zasilania w zakresie  $I_{DD}=100 - 500\mu A$ ,
- rezystancja ESD.



Rys. 5.25. Wyprowadzenia układu DAC8043

Pracą układu steruje interfejs szeregowy. Składa się on z trzech linii wejściowych SRI, LD, CLK. Wejścia te są kompatybilne ze standardem TTL. Obwody wejściowe tych linii posiadają rezystancję ESD, co zapewnia zwiększone bezpieczeństwo układu. Obwód zabezpieczający składa się z diod i rezystora szeregowego.

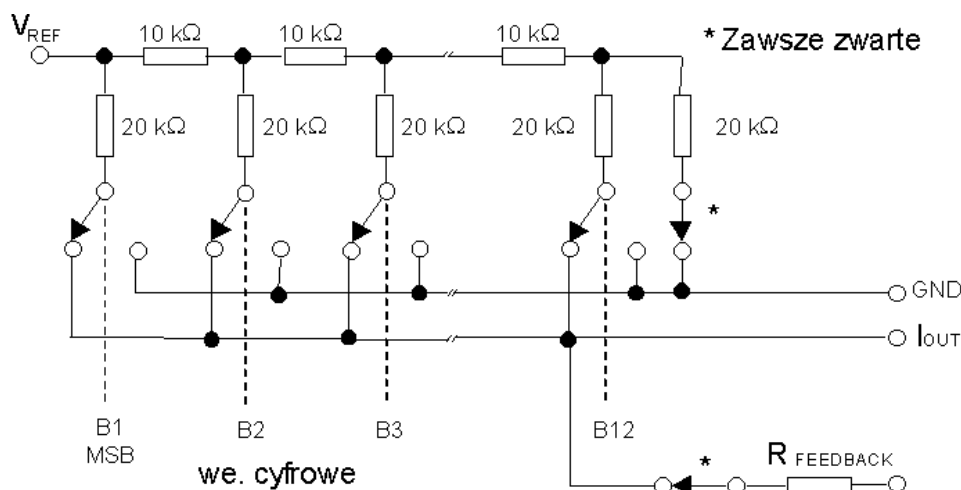


Rys. 5.26. Przebiegi czasowe interfejsu szeregowego układu DAC8043

Dane od najbardziej znaczącego bitu (MSB) do najmniej znaczącego (LSB) poprzez linię SDI są wprowadzane do rejestru szeregowego na narastające zbocze sygnału zegarowego na linii CLK (rys. 5.26). Jeżeli nowe 12-bitowe słowo programujące zostało wprowadzone do rejestru szeregowego, jest ono ładowane do rejestru zatrzymującego DAC poprzez podanie „ujemnego” impulsu na linii LD. Dane w tym rejestrze bezpośrednio sterują kluczami drabinki R-2R, tym samym są przetwarzane na wartość prądu wyjściowego. Maksymalna częstotliwość zegara wynosi 4MHz.

W rejestrze roboczym układu jest zapamiętywane 12 ostatnio przesłanych bitów. Zatem, w celu zaprogramowania określonej wartości prądu przez mk, musi on do tego przetwornika za pomocą interfejsu SPI przesyłać dwa bajty, przy czym starszy nibl pierwszego bajta jest nieistotny, gdyż i tak nie zostanie zachowany w rejestrze odbiorczym (metoda przepelnienia zawartości rejestru odbiorczego).

Przetwornik C/A składa się z drabinki rezystorowej R-2R. Drabinka ta jest wykonana z materiału *silikon-chrom*, cechującego się wysoką stabilnością temperaturową (+50 ppm/°C), oraz z dwunastu par kluczy typu NMOS (rys. 5.27).



Rys. 5.27. Schemat drabinki R-2R przetwornika C/A DAC8043

Klucze te dołączają nogę drabinki do GND lub  $I_{OUT}$ . Całkowity prąd wyjściowy zależy więc od kodu cyfrowego sterującego kluczami. Poważnym problemem jest zmiana rezystancji kluczy względem temperatury. Aby zniwelować ten wpływ, kolejne klucze mają rezystancje włączenia 10Ω, 20Ω, 40Ω itd., aby na każdym z nich było napięcie 5mV, oraz stosuje się rezystor  $R_{FEEDBACK}$ .

Układ DAC8043 może pracować w dwóch konfiguracjach, jako:

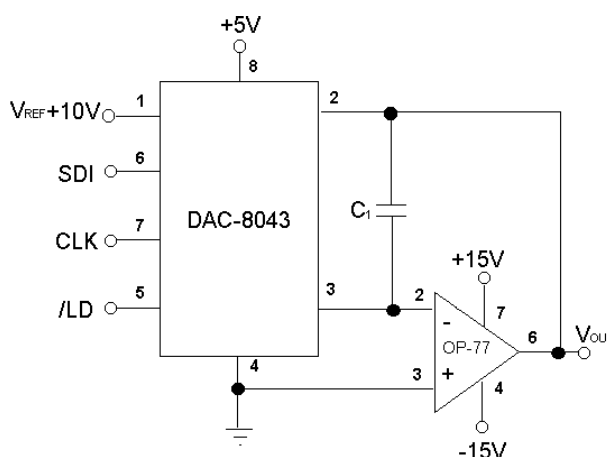
- układ **unipolarny**,
- układ **bipolarny**.

W pierwszym przypadku (rys. 2.28) może pełnić funkcje programowalnego źródła referencyjnego napięcia stałego DC lub zmiennego AC (w zależności od tego, jaki jest rodzaj napięcia referencyjnego). Napięcie wyjściowe układu można zmieniać w zakresie od 0V do  $-V_{REF} \cdot (4095/4096)$ . Zależność pomiędzy napięciem wyjściowym, a kodem jest przedstawiona w tabeli 5.9.

Tabela 5.9. Kody wprowadzane do przetwornika DAC8043 i odpowiadające im wartości napięć dla konfiguracji unipolarnej.

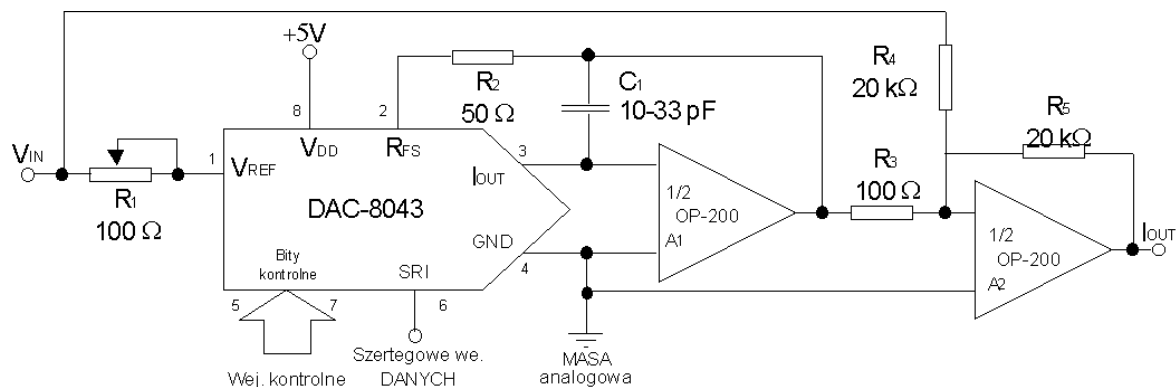
Digital Input MSB	LSB	Nominal Analog Output ( $V_{OUT}$ )
1111 1111 1111		$-V_{REF} \left( \frac{4095}{4096} \right)$
1000 0000 0001		$-V_{REF} \left( \frac{2049}{4096} \right)$
1000 0000 0000		$-V_{REF} \left( \frac{2048}{4096} \right) = -\frac{V_{REF}}{2}$
0111 1111 1111		$-V_{REF} \left( \frac{2047}{4096} \right)$
0000 0000 0001		$-V_{REF} \left( \frac{1}{4096} \right)$
0000 0000 0000		$-V_{REF} \left( \frac{0}{4096} \right) = 0$

Dla tej konfiguracji rozdzielczość napięcia określona jest zależnością  $LSB = V_{REF}(1/4096)$  lub  $LSB = V_{REF}(2^{-n})$ .



Rys. 5.28. Przetwornik DAC8043 w układzie unipolarnym

W drugiej konfiguracji przetwornik pracuje jako **układ bipolarny**. Na rys. 5.29 pokazano przykładowe rozwiązanie układu bipolarnego. Stan bitu MSB decyduje o kierunku napięcia. Rezystory  $R_3$ ,  $R_4$  i  $R_5$  muszą być wybrane z dokładnością 0,01% i muszą być tego samego typu. Niedokładność dopasowania wartości rezystorów  $R_3$  i  $R_4$  wpływa na offset i błąd skali, natomiast dokładność doboru rezystorów  $R_5$  w stosunku do  $R_4$  oraz  $R_3$  decyduje o błędzie skali.



Rys. 5.29. Przetwornik DAC8043 w układzie bipolarnym

W tabeli 5.10 zebrano kody i odpowiadające im wartości napięć wyjściowych dla układu bipolarnego.

Tabela 5.10. Kody wprowadzane do przetwornika DAC8043 i odpowiadające im wartości napięć dla konfiguracji bipolarnej.

Digital Input	Nominal Analog Output
MSB      LSB	( $V_{OUT}$ as Shown in Figure 7)
1111 1111 1111	$+V_{REF} \left( \frac{2047}{2048} \right)$
1000 0000 0001	$+V_{REF} \left( \frac{1}{2048} \right)$
1000 0000 0000	0
0111 1111 1111	$-V_{REF} \left( \frac{1}{2048} \right)$
0000 0000 0001	$-V_{REF} \left( \frac{2047}{2048} \right)$
0000 0000 0000	$-V_{REF} \left( \frac{2048}{2048} \right)$

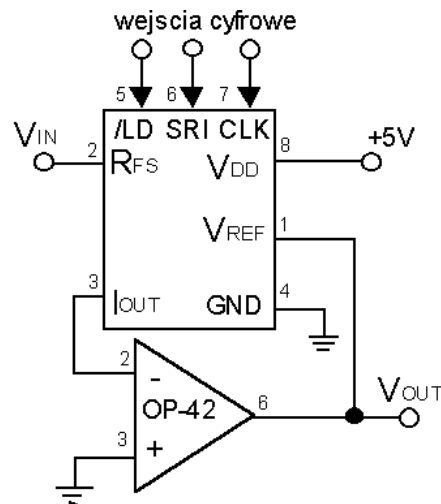
Dla tego układu napięcie wyjściowe wyrażone jest wzorem:

$$V_{OUT} = -V_{IN} \left( \frac{A_1}{2^1} + \frac{A_2}{2^2} + \dots + \frac{A_{12}}{2^{12}} \right) \text{ gdzie } A_X = "1" \text{ dla ON, "0" dla OFF.}$$

Warto zauważyć, że dla kodu 1000 0000 0000 napięcie wyjściowe powinno przyjmować wartość 0V. Fakt ten, może posłużyć do kalibracji całego układu. Dokonuje się tego regulując potencjometrem  $R_1$ , aż uzyska się napięcie 0V na wyjściu przy tym kodzie. Przy pomocy rezystorów  $R_3$  i  $R_4$  dobiera się wartość zakresową. W tym celu należy do przetwornika wysłać kod 1111 1111 1111.

Innym rozwiązaniem może być umieszczenie przetwornika C/A w pętli sprzężenia zwrotnego wzmacniacza operacyjnego, jak pokazano na rys. 5.30.





Rys. 5.30. Przetwornik DAC8043 w układzie bipolarnym umieszczony w sprzężeniu zwrotnym wzmacniacza operacyjnego

W tym przypadku napięcie wyjściowe jest równe ilorazowi wartości napięcia referencyjnego  $V_{IN}$  przez kod cyfrowy, zgodnie ze wzorem:

$$V_{OUT} = \left( \frac{-V_{IN}}{\left( \frac{A_1}{2^1} + \frac{A_2}{2^2} + \dots + \frac{A_{12}}{2^{12}} \right)} \right).$$

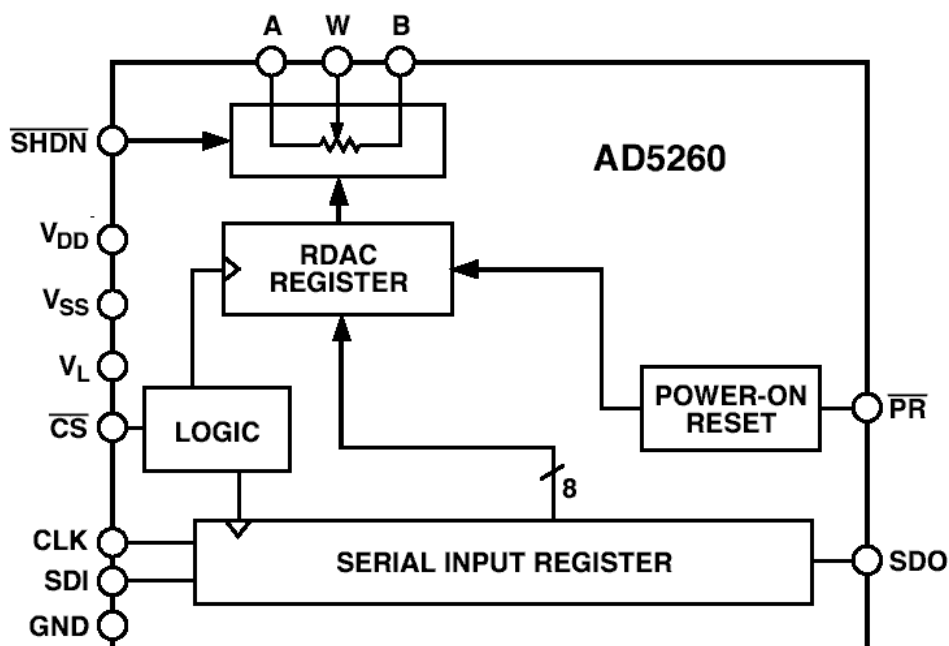
Należy uważać, aby kod cyfrowy nie składał się z samych zer, gdyż wzmacniacz będzie pracował w otwartej pętli sprzężenia zwrotnego. Gdy kod składa się z samych jedynek to uzyskuje się wzmocnienie równe 1. Największe wzmocnienie równe 4096 jest dla kodu 0000 0000 0001.

## 5.4. Cyfrowe potencjometry sterowane interfejsem SPI

Obecnie na rynku oferowana jest coraz większa gama potencjometrów, których wartość rezystancji można zmieniać programowo (np. układy firm *Analog Devices* lub *Linear Technology*). Grupę takich układów nazwano cyfrowymi potencjometrami. Najczęściej nastawę (wartość rezystancji jaką ma przyjąć potencjometr) wprowadza się do takiego układu za pośrednictwem interfejsu SPI.

Jednym z przedstawicieli potencjometrów cyfrowych jest układ AD5260 firmy Analog Devices. Posiada on następujące cechy:

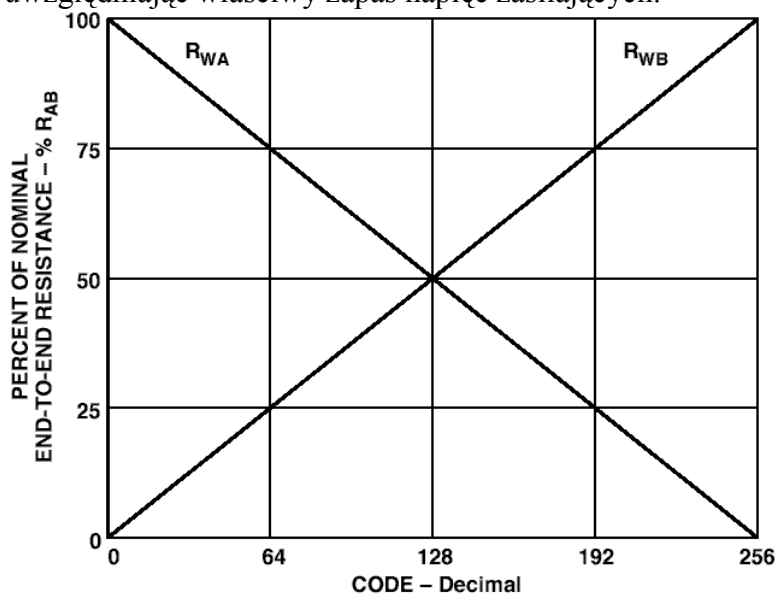
- 256 nastaw,
- jeden kanał,
- zamiennik potencjometrów mechanicznych 20kΩ, 50kΩ, 200kΩ,
- niski współczynnik temperaturowy zmian rezystancji 35ppm/C,
- cztero-liniowe wejście kompatybilne ze standardem SPI,
- pojedyncze zasilanie od 5 V do 15 V lub podwójne zasilanie +/-5.5 V,



Rys. 5.31. Schemat blokowy układu AD5260 firmy Analog Devices

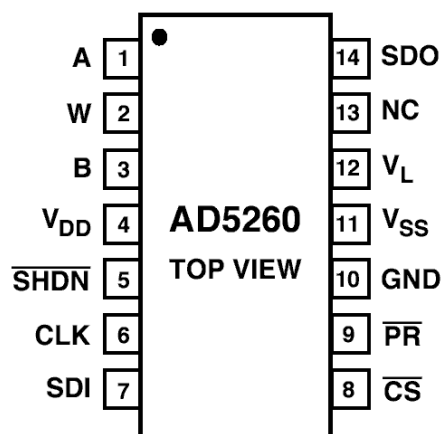
Układ AD5260 (rys. 5.31) posiada jeden kanał z 256-pozycyjnym, cyfrowo sterowanym rezystorem regulowanym (VR – *variable resistor*). Kanał ten można traktować jako oddzielny opornik z ruchomym kontaktem. Wartość rezystancji jest ustalana przez kod cyfrowy załadowany do rejestru RDAC. Kod ten wprowadzany jest najpierw do rejestru szeregowego wejściem SDI na narastające zbocze sygnału CLK, po czym po skompletowaniu ośmiu bitów przesyłany do rej. RDAC. Opór między suwakiem (linia W) i punktami przytwierdzonego opornika (linie A i B) zmienia się liniowo w zależności do kodu cyfrowego (rys. 5.32).

Normalna rezystancja pomiędzy pinami A i B ( $R_{AB}$ ) wynosi  $20k\Omega$ ,  $50k\Omega$ , lub  $200k\Omega$ , a nominalny współczynnik temperaturowy  $35ppm/C$ . W odróżnieniu od większości cyfrowych potencjometrów dostępnych na rynku, mogą one pracować z napięciami do 15V lub przy napięciach  $\pm 5V$ , uwzględniając właściwy zapas napięć zasilających.

Rys. 5.32. Wartości rezystancji  $R_{WA}$  i  $R_{WB}$  w zależności od kodu sterującego

Niski poziom na linii PR powoduje wpisanie do rejestru RDAC (inaczej: *VR latch*) wartości 80H (128), co powoduje ustawienie suwaka na „środku” potencjometru.

Na rysunku 5.33 przedstawiono obudowę układu AD5260. Opis pinów zawarto w tabeli 5.11.



Rys. 5.33. Wyprowadzenia układu AD5260

Tabela 5.11. Opis funkcji pinów układu AD5260

Numer pinu	Symbol pinu	Opis pinu
1	A	Końcówka A
2	W	Suwak
3	B	Końcówka B
4	VDD	Dodatni potencjał zasilania , zakres 5V lub 15V. (Suma $ V_{DD}  +  V_{SS}  \leq 15V$ )
5	SHDN	Aktywny stanem niskim. Rozwarcie końcówki A.
6	CLK	Wejście zegarowe aktywne zboczem narastającym.
7	SDI	Wejście szeregowe interfejsu SPI
8	CS	Wybór układu, aktywny stanem niskim. Kiedy stan na linii CS jest wysoki, dane mogą być ładowane do rejestru RDAC.
9	PR	Aktywny stanem niskim. Ustawia rezystor na połowę jego wartości, poprzez wpisanie do rej. RDAC wartości 80H.
10	GND	Masa
11	VSS	Ujemny potencjał zasilania, zakres napięć od 0V do 5V.
12	VL	Zasilanie „logiki” układu.
13	NC	Nie podłączone
14	SDO	Wyjście szeregowe typu otwarty dren, Wymaga zewnętrznego rezystora pull-up.

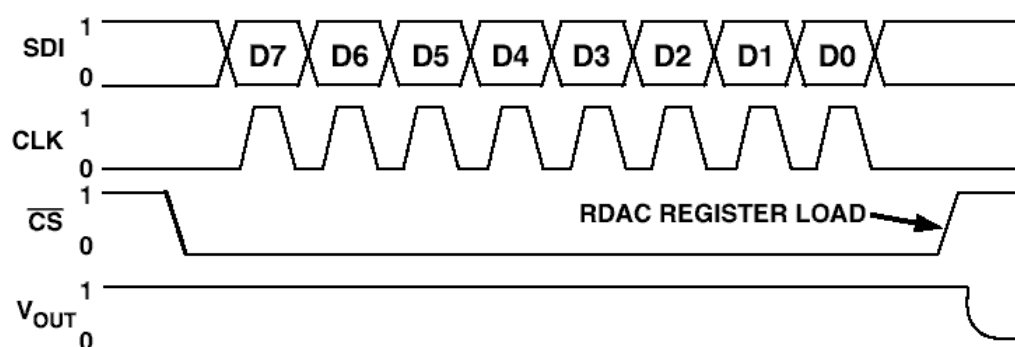
Po włączeniu zasilania potencjometr przyjmuje wartość środkową, co minimalizuje możliwość wystąpienia usterki. Wartość środkowa skali może być ustawiona w dowolnym momencie przez podanie niskiego poziomu na linii PR. Aktywny sygnał na linii wyłączającej SHDN powoduje, zminimalizowanie poboru mocy oraz rozwarcie końcówki A (końcówki W i B są nadal zwarte).

Kod nastawy potencjometru jest wprowadzany poprzez interfejs SPI składający się z linii: SDI, SDO, CS i CLK. 8-bitowe słowo ładowane jest do rejestru szeregowego od najstarszego bitu MSB (format tego słowa pokazano w tabeli 5.12). Kiedy linia CS jest w stanie niskim, dane pojawiające się na wejściu SDI są ładowane do rejestru szeregowego przy każdym narastającym zboczach sygnału zegara na linii CLK. Dane z tego rejestru są przesyłane do wewnętrznego rejestru RDAC, kiedy poziom na linii CS powraca do stanu wysokiego (rys.

5.34). Podczas wyłączania (niski sygnał na linii SHDN) wyjście SDO jest w stanie wysokiej impedancji.

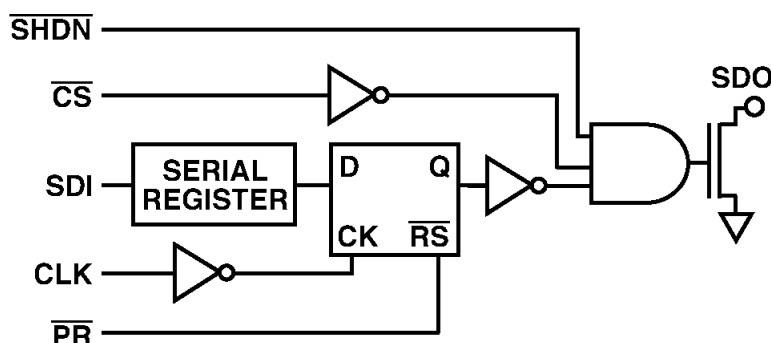
Tabela 5.12. Format słowa sterującego dla układu AD5260

DATA							
B7	B6	B5	B4	B3	B2	B1	B0
D7	D6	D5	D4	D3	D2	D1	D0
MSB				LSB			
$2^7$				$2^0$			



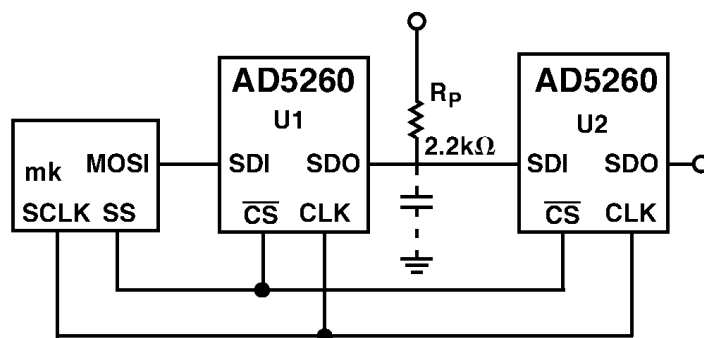
Rys.5.34. Przebiegi czasowe na interfejsie SPI dla układu AD5260

Szeregowe wyjście danych SDO jest typu otwarty dren (rys. 5.35). Wyjście to wymaga rezystora pull-up, aby móc przesyłać daną do następnego układu (musi być podłączone do jego wejścia SDI). Czyli służy ono do wyprowadzania danych, uprzednio wprowadzonych do układu przez wejście SDI (z opóźnieniem ośmiu impulsów zegarowych). Stąd z punktu widzenia funkcji interfejsowych, interfejs tego układu realizuje dwie funkcje: **funkcje odbiornika** (*listener*) i **funkcje pośrednika** (*repeater*).



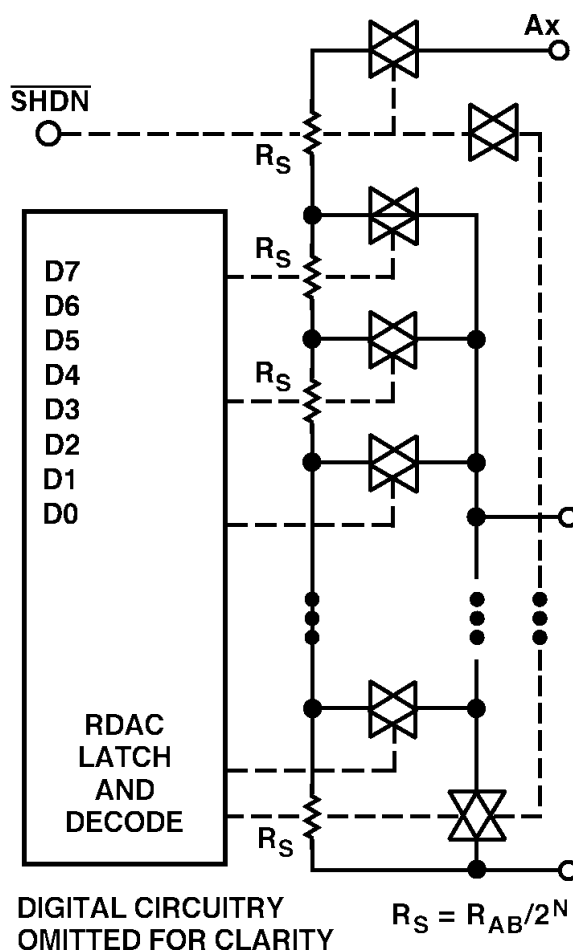
Rys. 5.35. Schemat wyjścia SDO układu AD5260

Taki mechanizm pozwala na łączenie wielu potencjometrów cyfrowych w szereg, przy czym linie CLK i CS tych układów są równolegle podłączone do mk. Napięcie, do którego podłączony jest rezystor pull-up może być wyższe, niż napięcie zasilania układu. Wskazane jest, aby zwiększać okres zegara, w przypadku podłączenia "łańcucha" takich potencjometrów, gdyż przesłanie danej pomiędzy kolejnymi układami wprowadza opóźnienie. Sygnał CS powinien być utrzymywany w stanie niskim do czasu, gdy do każdego układu zostanie przesłane 8 bitów. Np. w przypadku podłączenia dwóch potencjometrów, po 16 bitach można ustawić stan wysoki na linii CS (rys. 5.36).



Rys. 5.36. Podłączenie dwóch układów AD5260 do mk

**Blok RDAC** zawiera łańcuch rezystorów o tej samej wartości, odpowiednio podłączany do zacisku W przez macierz przełączników analogowych, które zachowują się jak suwak w tradycyjnym potencjometrze. W danej chwili, w zależności od kodu sterującego, włączony jest tylko jeden klucz. Sterowanie kluczami jest realizowane przez rejestr RDAC i układ dekodujący wartość cyfrową na numer klucza do włączenia. Układ AD5260 daje możliwość ustawienia 256 poziomów wartości rezystancji, a co za tym idzie dokładność nastawienia rezystancji wynosi 0.4%. Rys. 5.37 przedstawia równoważną strukturę połączeń pomiędzy trzema końcówkami potencjometru. Ponieważ przełączniki analogowe nie są idealne w stanie, gdy suwak znajduje się w skrajnych położeniach, wartość rezystancji pomiędzy suwakiem a końcówkami wynosi minimalnie 60Ω.



Rys. 5.37. Uproszczona architektura bloku RDAC

Zakres zmian rezystancji między końcówkami A i B, a suwakiem (końcówka W) opisane są poniższymi wzorami i tabelami.

Dla końcówki B wzór na rezystancje jest następujący:

$$R_{WB}(D) = \frac{D}{256} \times R_{AB} + R_w, \text{ gdzie: } D - \text{kod cyfrowy, } R_w - \text{rezystancja klucza (około } 60\Omega\text{)}.$$

Na podstawie tej zależności utworzono tabelę 5.13.

Tabela 5.13. Wartości rezystancji  $R_{WB}$  w zależności od kodu D dla  $R_{AB}=20k\Omega$

Kod cyfrowy D	Rezystancja $R_{WB}$ ( $\Omega$ )	Stan wyjścia
256	19982	Pełna skala ( $R_{AB} - 1 \text{ LSB} + R_w$ )
128	10060	Środek skali
1	138	1 LSB
0	60	minimum skali (rezystancja połączenia suwaka)

Natomiast dla końcówki A zależność na rezystancje przyjmuje postać:

$$R_{WA}(D) = \frac{256 - D}{256} \times R_{AB} + R_w.$$

Również na podstawie tej zależności utworzono tabelę (tabela 5.14).

Tabela 5.14. Wartości rezystancji  $R_{WA}$  w zależności od kodu D dla  $R_{AB}=20k\Omega$

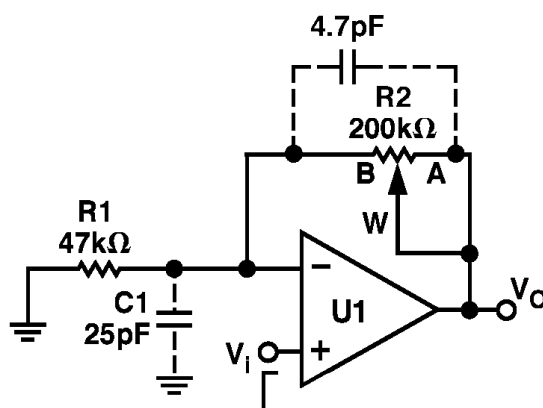
Kod cyfrowy D	Rezystancja $R_{WB}$ ( $\Omega$ )	Stan wyjścia
256	60	Pełna skala
128	10060	Środek skali
1	19982	1 LSB
0	20060	Minimum skali

Poniżej przedstawiono kilka przykładowych aplikacji potencjometru cyfrowego:

- Korzystając z układu AD5260 można zbudować w prosty sposób **programowalny dzielnik napięcia**, przykładowo podłączając końcówkę A do +5V, a końcówkę B do 0V. Zatem na suwaku można uzyskać napięcie w zakresie od 0V do 5V. Napięcie na końcówce suwaka W  $V_W(D)$  jest określone wzorem:

$$V_W(D) = \frac{D}{256} \times V_{AB} + V_B.$$

- Układ AD5260 można użyć do **kontroli wzmocnienia wzmacniacza**, w sposób pokazany na rys. 5.38.



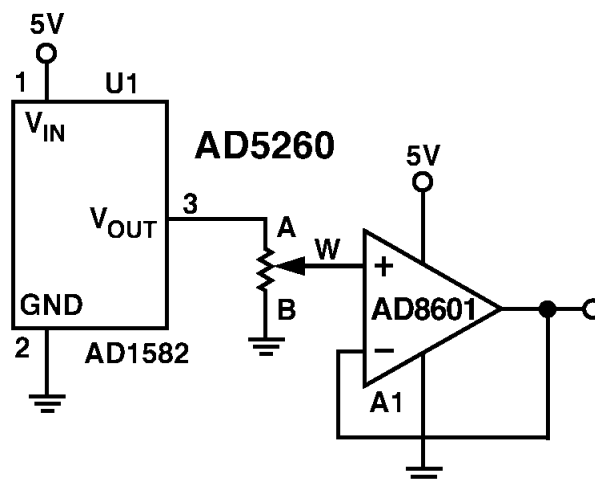
Rys. 5.38. Typowy nieodwracający wzmacniacz operacyjny

Wzmocnienie  $A(D)$  takiego układu można wyrazić zależnością:

$$A(D) = \frac{D}{256} \frac{R_{AB}}{R_1} + \frac{R_W}{R_1}.$$

Warto zauważyć, że w przypadku podłączenia potencjometru tak jak na rysunku 5.38 mamy do czynienia z dwiema pasożytniczymi pojemnościami. Wynikają one ze struktury potencjometru. Powoduje to wprowadzenie bieguna, który wpływa na zmianę charakterystyki częstotliwościowej. Wprowadza to przy pewnym zakresie częstotliwości dodatkowe tłumienie  $-20\text{dB/dek}$ . Dodatkowo może wystąpić wygarbienie rezonansowe charakterystyki, które w pewnym przypadku może spowodować generację drgań periodycznych, czyli układ stanie się niestabilny z powodu braku zapasu fazy. Zatem najlepiej by było gdyby  $R_1 \cdot C_1 = R_2 \cdot C_2$ . Jest to bardzo trudne do osiągnięcia, gdyż wartości pojemności  $C_1$  i  $C_2$  zmieniają się w funkcji zmiany rezystancji  $R_2$ . W przypadkach krytycznych, wartość pojemności  $C_2$  powinna być wyznaczana empirycznie (zmienia się ona w zakresie od pojedynczych pF do dziesiątek pF).

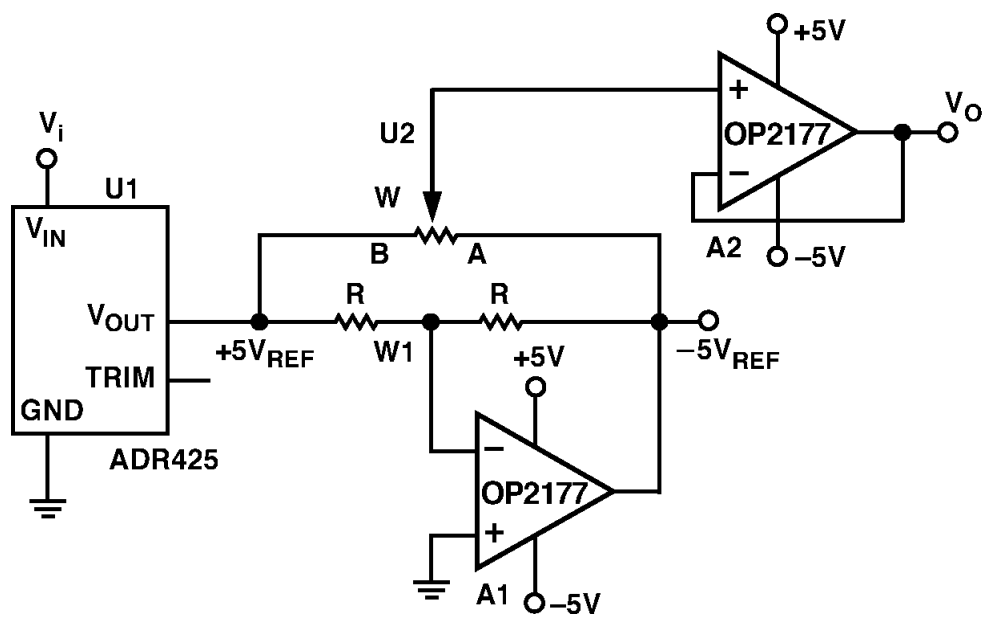
- **Układ programowalnego napięcia odniesienia** przedstawiono na rysunku 5.39. Dzięki zastosowaniu bufora wyjściowego możliwe jest podłączenie do niego większego obciążenia.



Rys. 5.39. Programowalne źródło napięcia odniesienia

- Na rys. 5.40 przedstawiono przykładowy **8-bitowy bipolarny przetwornik cyfrowo-analogowy** oparty na układzie AD5260. Przetwornik ten oferuje taką samą rozdzielczość jak 8-bitowe scalone przetworniki C/A, lecz w zakresie małych kodów nie zapewnia należytej dokładności. Wynika to z faktu, iż dla małych wartości nastawionej rezystancji dominującym czynnikiem staje się rezystancja suwaka nie mniejsza niż  $60\Omega$ . Napięcie na wyjściu przetwornika  $V_o$  określa się wzorem:

$$V_o = \left( \frac{2D}{256} - 1 \right) \times V_{ref}.$$



Rys. 5.40. 8-bitowy bipolarny przetwornik C/A



## 6. Bibliografia

### 6.1. Literatura podstawowa

1. Tłaga W.: „Procesory w systemach pomiarowych”, Mat. konf. Szkoła-Konferencja Metrologia Wspomagana Komputerowo, Tom 1, Zegrze k/Warszawy, 24-28 maj 1993, s. 129-164.
2. Dąca W.: „Mikrokontrolery od układów 8-bitowych do 32-bitowych”, Wyd. NIKOM, Warszawa, kwiecień 2000.
3. Pełka R.: „Mikrokontrolery, architektura, programowanie, zastosowania”, Wyd. WKŁ, Warszawa 1999.
4. Pasierbiński J., Zbysiński P.: „Układy programowalne w praktyce”, Wyd. WKŁ, Warszawa 2002.
5. Krysiak A.: „Mikrokontrolery rodziny AVR, AT 90S1200”, Wyd. Studio Wydawniczo-Typograficzne „Typoscript”, Wrocław 1999.
6. Gałka P., Gałka P.: „Podstawy programowania mikrokontrolera 8051”, Wyd. NIKOM, Warszawa, wrzesień 1995.
7. Jabłoński T.: „Mikrokontrolery PIC16F8x w praktyce”, Wyd. BTC, Warszawa 2002.
8. Rydzewski A.: „Mikrokomputery jednocukładowe rodziny MCS-51”, Wyd. Naukowo-Techniczne, Warszawa 1992.

### 6.2. Dokumentacja w postaci plików PDF

1. Dokumentacja mk AT90S8515 firmy Atmel (doc0841.pdf).
2. Dokumentacja rodziny 80C51 firmy Philips (89C51\_89C52\_89C54\_89C58\_10.pdf).
3. Dokumentacja mk PIC16F87X firmy Microchip (30292c.pdf).
4. Specyfikacja programowania EEPROM mk PIC16F87X firmy Microchip (39025f.pdf).
5. Dokumentacja mk ST72104G, ST72215G, ST72216G, ST72254G firmy STMicroelectronics (6815.pdf).
6. Dokumentacja układu 74HC/HCT574 firmy Philips (74HC\_HCT574\_CNV\_2.pdf).
7. Dokumentacja układów rodziny K6T1008C2E firmy Samsung (K6T1008C2E.PDF).
8. Dokumentacja pamięci Am29F010B firmy AMD (22336c1.pdf).
9. Dokumentacja układu GAL16V8 firmy Lattice (16V8.pdf).
10. Dokumentacja układów rodziny XC9500 firmy Xilinx (9500.pdf).
11. Dokumentacja układów 25AA080/25LC080/25C080 firmy Microchip (21230c.pdf).
12. Dokumentacja układu 93C46B firmy Microchip (21173e.pdf).
13. Dokumentacja układu AD7233 firmy Analog Devices (AD7233\_b.pdf).
14. Dokumentacja układu AD7866 firmy Analog Devices (AD7866\_0.pdf).
15. Dokumentacja układu DAC8043 firmy Analog Devices (dac8043.pdf).
16. Dokumentacja układu AD5260/AD5262 firmy Analog Devices (AD5260\_2\_0.pdf).
17. Dokumentacja układu MCP2510 firmy Microchip (21291e.pdf).