Wykład

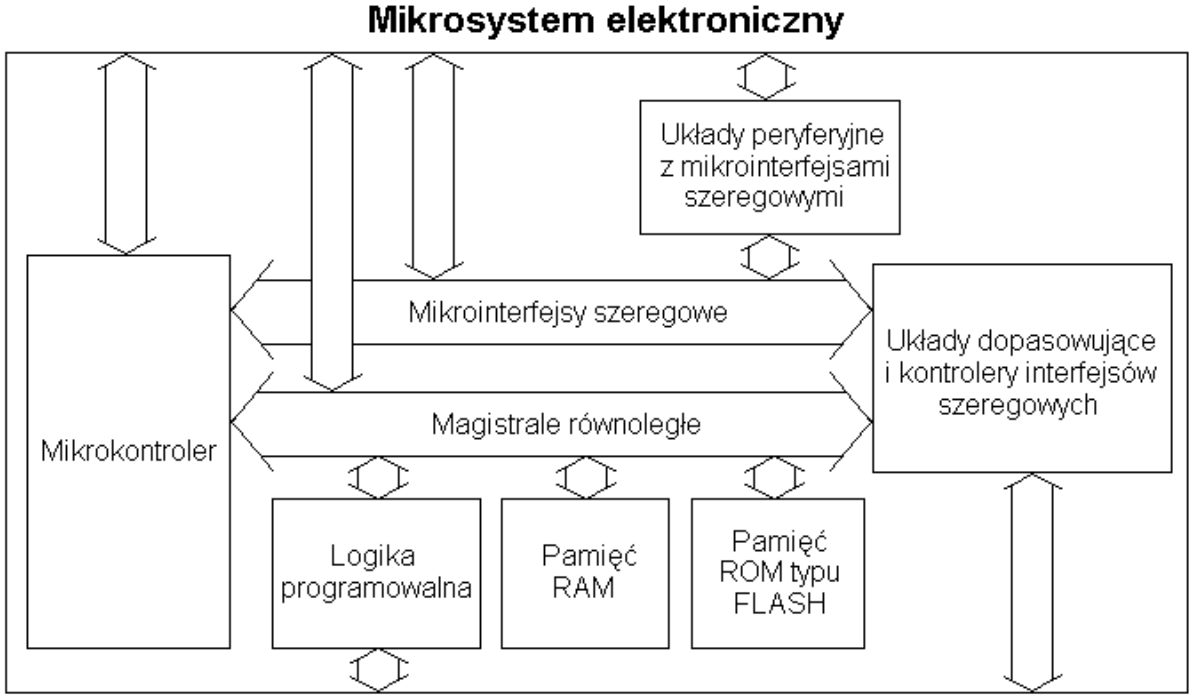
Mikrokontrolery i Mikrosystemy

autor: dr inż. Zbigniew Czaja

Gdańsk 2005

Spis treści

1. Wprowadzenie



Rys. 1.1. Przykładowy schemat blokowy mikrosystemu elektronicznego

Sercem mikrosystemu elektronicznego **(w skrócie: mse)** (rys. 1.1) jest **mikrokontroler** nadzorujący pracą wszystkich układów wchodzących w jego skład. Mikrokontroler steruje pracą mse za pośrednictwem **magistral równoległych** oraz **interfejsów** (**mikrointerfejsów**) szeregowych. Niektóre wersje mikrokontrolerów potrzebują do pracy zewnętrznych pamięci RAM i ROM (np. typu FLASH), stąd zostały one pokazane na schemacie blokowym. W skład mse mogą również wchodzić: **logika programowalna** (SPLD, CPLD lub FPGA), **układy peryferyjne** (przetworniki A/C, C/A, pamięci szeregowe EEPROM, itd.), wśród których można wyróżnić **układy dopasowujące sygnały elektryczne** do danego standardu interfejsu (np. MAX232AC) i **kontrolery interfejsów** szeregowych (np. Ethernetu, CAN, USB, IrDA), jak i równoległych.

Komunikacja wewnątrz mse odbywa się za pomocą magistral równoległych lub mikrointerfejsów szeregowych. Mse komunikuje się z otoczeniem przeważnie za pomocą interfejsów szeregowych.

W skrajnym przypadku mse może składać się wyłącznie z mikrokontrolera **(w skrócie: mk)**.

Podsumowując można wymienić następujące składowe mikrosystemu:

- mikrokontroler(y) – najważniejszy element mse,
- pamięć RAM – opcjonalnie w zależności od typu mk,
- pamięć ROM – opcjonalnie w zależności od typu mk,
- logika programowalna – zastępująca konwencjonalne układy TTL,
- układy peryferyjne z mikrointerfejsem szeregowym **(w skrócie: mis)** – np. przetworniki A/C, C/A, EEPROM,
- układy peryferyjne z interfejsem równoległym,
- układy dopasowujące i kontrolery interfejsów szeregowych – RS232, RS485, Ethernet, CAN, USB i bezprzewodowych IrDA, Bluetooth, Wi-Fi, Zig Bee.

- magistrale równoległe – równoległe porty we/wy, opcjonalnie magistrale danych i adresowe,
- mikrointerfejsy szeregowe – SPI, I^2C, CAN, UART, ISP, JTAG, 1-wire.

Mse można zdefiniować następująco:

Mikrosystem elektroniczny – zestaw układów elektronicznych stanowiących funkcjonalną, integralną część urządzenia lub samo urządzenie, które mogą komunikować się między sobą za pomocą dedykowanych mis. Przeważnie komunikacja mse z otoczeniem odbywa się również za pomocą dedykowanych interfejsów szeregowych przewodowych, jak również bezprzewodowych radiowych i na podczerwień.

W mse możemy wyróżnić dwie warstwy **sprzętową** i **programową**. Pierwszą warstwę stanowi „**sprzęt**", tj. płytka drukowana (*printed board*) z umieszczonymi na niej układami scalonymi. Drugą warstwą jest **oprogramowanie** dla mk realizujące zadane przez nas algorytmy, zapisane w pamięci FLASH ROM mk lub zewnętrznej pamięci ROM oraz zapis konfiguracji połączeń i realizowanych funkcji w programowalnych układach logicznych (PLD).

Mse sterowane mk, bądź procesorami sygnałowymi, mogą być częścią większego urządzenia, np.: są zrealizowane w postaci karty komputerowej lub pakietu montowanego do danego urządzenia lub mogą być samodzielnym urządzeniem np.: telefon komórkowy, inteligentny czujnik. Stąd nazywa się je **wbudowanymi systemami elektronicznymi**.

Gdyż, zgodnie z definicją **system wbudowany** jest to jakikolwiek produkt elektroniczny zawierający procesor lub procesory, nie dający się zakwalifikować jako komputer biurowy (ogólnego przeznaczenia). Uwzględniając najistotniejsze właściwości i cechy systemów wbudowanych można powiedzieć, że system wbudowany:

- to mikrokomputer (lub urządzenie), będący częścią jakiegoś większego systemu macierzystego (wbudowany w niego) w celach innych niż dostarczanie obliczeń i działań o ogólnym zastosowaniu,
- posiada typowo dedykowane oprogramowanie,
- często bez klawiatury, wyświetlacza, monitora. Typowy system wbudowany składa się z procesora (mikrokontrolera), odpowiednich urządzeń wejścia /wyjścia, pamięci ROM, RAM i oprogramowania. Oczywiście system może być tak rozbudowany, ze będzie potrzebował kilku procesorów na jednej lub kilku płytach głównych,
- służy do wykonywania zaimplementowanych w niego funkcji takich jak, np. sterowanie ABS (Anti-lock Braking System) i ESP (Electronic Stability Program) w samochodzie, sterowanie linią produkcyjna, trajektorią lotu pocisku, kompresją i dekompresja dźwięku, obrazu i danych w systemach multimedialnych, itd.,
- zwykle ma pracować przez cały czas życia systemu macierzystego, w którym jest on wbudowany; może to być parę lat (nieskomplikowany podzespół dźwiękowy), lub całe dekady (np. system sterowania lotem),
- nieraz zdarza się tak, ze system jest niszczony po wykonaniu zadania (system wbudowany w pocisk Cruise jest niszczony wraz z detonacja ładunku),
- musi nadążać za wymaganiami czasowymi dyktowanymi przez środowisko zdefiniowane przez system macierzysty. Wymagania te określają maksymalny czas reakcji na zaistniałe w systemie zdarzenie oraz mogą dotyczyć np. zapewnienia minimalnej częstotliwości próbkowania. Systemy spełniające powyższe wymagania określane są jako **systemy czasu rzeczywistego**. Można stwierdzić, że prawie wszystkie systemy wbudowane są systemami czasu rzeczywistego.

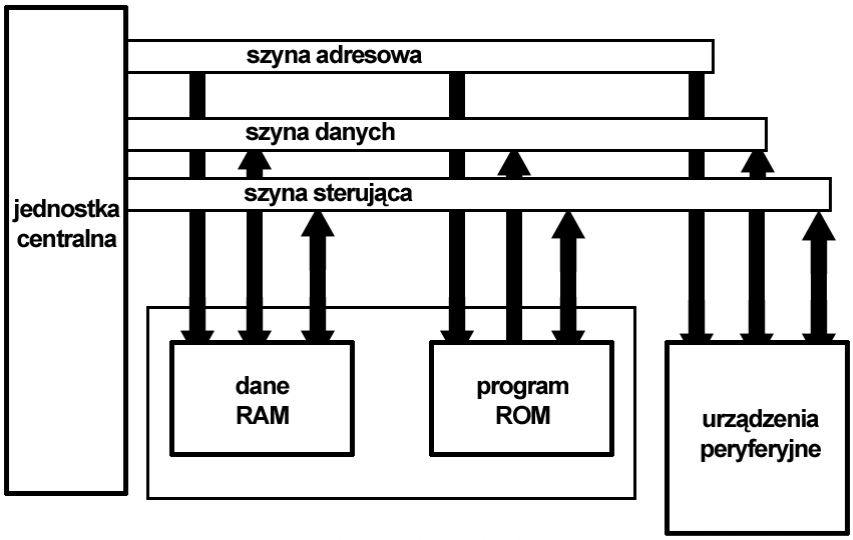
2. Mikrokontrolery

Najważniejszym elementem mse jest mikrokontroler (**mk**). Jak wspomniano we wstępie, w skrajnym przypadku mse może składać się wyłącznie z mk.

Mk można zdefiniować następująco:

Jest to układ cyfrowy z wyspecjalizowanym mikroprocesorem i niezbędnymi urządzeniami zawartymi w jednym układzie scalonym, czyniącymi go układem autonomicznym (do pracy nie są wymagane urządzenia zewnętrzne, takie jak np. kontrolery magistral, przerwań, generatory sygnałów taktujących mikroprocesor, itp.). Zatem:

- jest zdolny do **autonomicznej pracy**, tzn. w najprostszych zastosowaniach nie wymaga przyłączenia zewnętrznych układów pomocniczych (peryferyjnych),
- został zaprojektowany do pracy w **systemach kontrolno-pomiarowych oraz komunikacyjnych**, stąd posiada rozbudowany system komunikacji z otoczeniem,
- z reguły pracuje w **czasie rzeczywistym**.



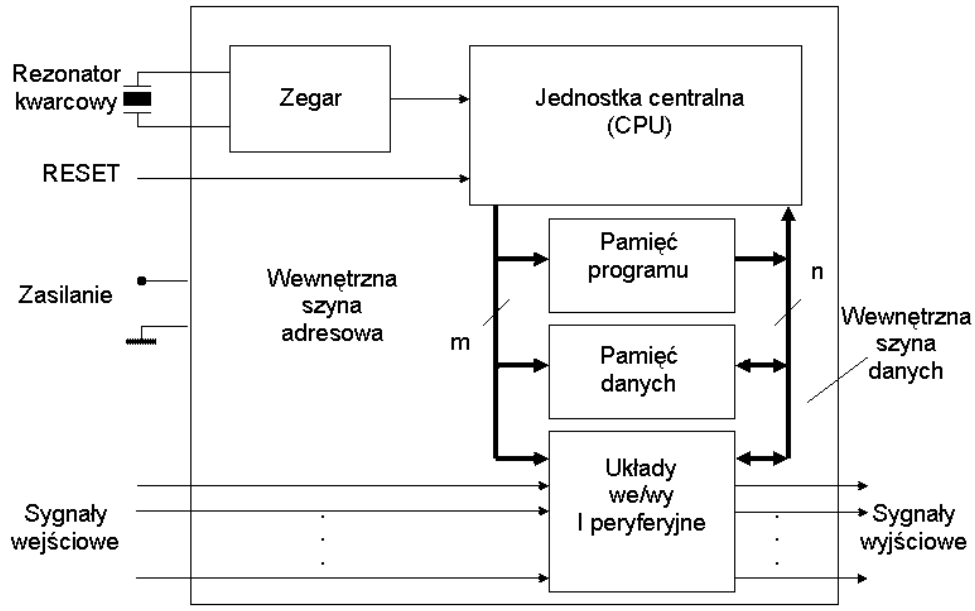
Rys. 2.1. Uproszczona budowa mk

Na rys. 2.1. pokazano uproszczoną budowę mk. Widać z niego, że mk jest układem scalonym, w którego strukturze zintegrowane są wszystkie elementy kompletnego komputera: **jednostka centralna**, **pamięci** oraz **urządzenia peryferyjne**. Mk komunikuje się z otoczeniem za pośrednictwem wewnętrznych urządzeń peryferyjnych.

Jedną z najważniejszych cech jednostki centralnej mk jest **szerokość szyny danych**. Może być ona 8-bitowa, 16-bitowa lub 32-bitowa. Na podstawie tej szerokości określa się typ mk, np. mk 8-bitowy.

Kolejną cechą jest **częstotliwość sygnału taktującego (zegarowego)**. Mk ukierunkowane są na zastosowania w układach kontrolno pomiarowych i komunikacyjnych. W wielu tego typu zastosowaniach daje się określić wymaganą minimalną prędkość przetwarzania danych przez jednostkę centralną. Nie obowiązuje przy tym zasada wzięta z mikroprocesorów, że im większa prędkość przetwarzania danych tym lepiej. Ze wzrostem częstotliwości sygnału zegarowego rośnie pobór mocy, co jest niekorzystne między innymi w systemach bateryjnych. Ponadto, najistotniejszym w mse jest czas reakcji mk na sygnały zewnętrzne. Zatem mk mają rozbudowane **układy przerwań**. Dodatkowo w wielu przypadkach zadania, które wykonuje

mk stosowany w układach sterowania, są zazwyczaj niezbyt złożone pod względem obliczeniowym.



Rys. 2.2. Budowa mk

Na rys. 2.2. przedstawiono uszczegółowioną budowę mk. **Jednostka centralna** realizuje program zawarty w **pamięci programu (ROM lub FLASH)**. Zmienne programu przechowywane są w **pamięci danych RAM**. Jednostka centralna jest taktowana **zegarem** (*clock*), którego częstotliwość jest stabilizowana **oscylatorem kwarcowym**. W stan początkowy mk wprowadzany jest **sygnałem RESET**. **Układy peryferyjne** umożliwiają odczyt sygnałów wejściowych cyfrowych, jak i również analogowych (przetworniki A/C) oraz generację sygnałów wyjściowych stosowanych do sterowania układami mse.

Mk posiada następujące cechy:

1. zamknięcie magistrali danych i adresowej wewnątrz układu scalonego,
2. stała struktura pamięci ROM/RAM,
3. stałość programu sterującego,
4. dostęp do rejestrów procesora i układów we/wy poprzez mechanizm adresowania pamięci RAM (*memory mapped registers and I/O*),
5. rejestrowa struktura jednostki centralnej,
6. procesory boolowskie wykonujące operacje na pojedynczych bitach w pamięci, rejestrach i układach we/wy,
7. bogaty zestaw urządzeń we/wy,
8. rozbudowane i szybkie układy przerwań,
9. różnorodne tryby i środki redukcji mocy pobieranej,
10. rozbudowane mechanizmy kontroli i detekcji nieprawidłowych stanów mk,
11. zawarcie w jednej strukturze układów cyfrowych (sterujących) i analogowych (pomiarowych).

Trzy pierwsze cechy dotyczą mk zamkniętych, które będą omówione dalej.

2.1. Jednostka centralna

Najważniejszą częścią mk jest jednostka centralna (**w skrócie jc**). Często określa się ją jako **procesor rdzeniowy** lub w skrócie procesor. Posiada ona właściwości:

- należy do grupy układów cyfrowych, określanych jako układy **synchroniczne i sekwencyjne**,
- synchroniczność oznacza, że wszystkie operacje wykonywane przez jc odbywają się w rytm sygnału zegarowego,
- sekwencyjność oznacza, że stan wyjść jc zależy nie tylko od stanu jej wejść, ale i od poprzednich stanów tego układu,
- posiada własną pamięć (rejestry) potrzebną np. do przechowywania argumentów rozkazów niezbędnych do wykonania na nich określonej operacji.

Działanie jc polega na cyklicznym wykonaniu instrukcji zawartych w programie użytkownika przechowywanym w pamięci programu mk. Lista instrukcji jest z góry określona dla danego mk. Zakłada się, że **instrukcja** składa się z kodu operacji nazywanego kodem rozkazowym lub w skrócie **rozkazem** i **argumentu** lub argumentów.

Cykl wykonania instrukcji rozpoczyna się zawsze od wczytania do wewnętrznych rejestrów jc kolejnego rozkazu. Gdy niezbędne do wykonania operacji są argumenty, to zawarte są one lub informacja o ich miejscu przechowywania w dalszej części instrukcji. W kolejnej fazie następuje pobranie tych argumentów i umieszczenie ich w odpowiednich wewnętrznych rejestrach jc. Po tym następuje wykonanie instrukcji. Jednocześnie jest inkrementowany **licznik rozkazów** wskazujący adres spod którego pobierane są instrukcje.

Sposób dostępu jc do argumentów zależy od **trybu adresowania**. Przez tryb adresowania rozumie się sposób wskazywania na argumenty wykorzystywane w trakcie wykonania instrukcji. Do najważniejszych trybów adresowania można zaliczyć:

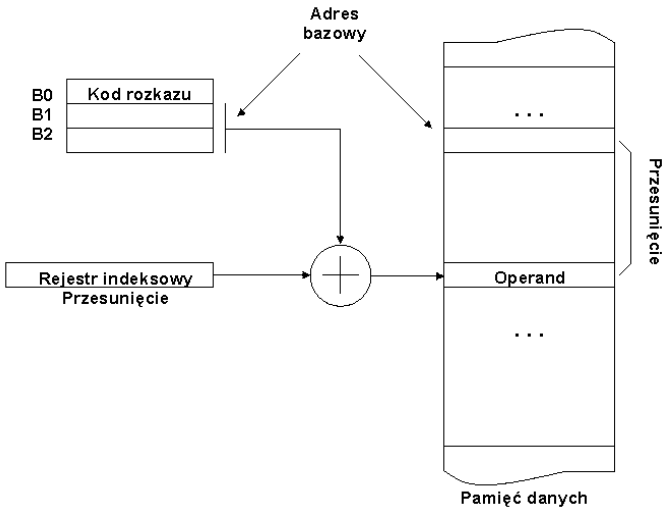
- implikowane, zwane też wewnętrznym lub rejestrowym (*inherent, register*),
- natychmiastowe (*immediate*),
- bezpośrednie (*direct*),
- indeksowe (*indirect*),
- względne (*relative*).

Adresowanie implikowane dotyczy jednobajtowych instrukcji, dla których zarówno operand jak i miejsce przeznaczenia wyniku są określone przez słowo rozkazowe. Tryb ten używany jest wyłącznie w odniesieniu do wewnętrznych rejestrów jc, w tym przede wszystkim do akumulatora. Np. ADD A,R1, INC A, CPL A, NOP.

Adresowanie natychmiastowe – operand (argument) jest podany w jawnej postaci w kodzie instrukcji. Tryb ten może być używany wyłącznie w odniesieniu do stałych zapisanych w kodzie programu. Np. ADD A,#dana, MOV A,#0F3h.

Adresowanie bezpośrednie odnosi się do instrukcji wielobajtowych, w których po kodzie rozkazu następuje adres argumentu umieszczonego w pamięci danych (komórki pamięci RAM). Np. ADD A,adres, MOV A,0F8h.

Adresowanie indeksowe polega na obliczeniu adresu przez sumowanie zawartości specjalnie przeznaczonego do indeksowania rejestru, nazywanego **rejestrem indeksowym**, z adresem bezpośrednim, zapisanym w instrukcji (lub odwrotnie – wówczas mówi się o **adresowaniu bazowym**). Obliczony w ten sposób adres fizyczny pamięci bywa nazywany **adresem efektywnym** (rys. 2.3). Adresowanie to jest szczególnie użyteczne przy operowaniu na blokach danych. Umieszczając w instrukcji adres początku bloku danych można uzyskać wygodny dostęp do kolejnych bajtów danych przez tylko samą zmianę zawartości rejestru indeksowego.



Rys. 2.3. Adresowanie indeksowe

Adresowanie pośrednie ma miejsce, gdy część adresowa instrukcji wskazuje na komórkę pamięci zawierającą adres efektywny. Odmianą tego adresowania jest **adresowanie zawartością rejestrów** (*pointer addressing*), w którym adres efektywny jest zawarty w przeznaczonym do tego celu rejestrze lub parze rejestrów mk. W tym przypadku identyfikacja tych rejestrów odbywa się na podstawie słowa rozkazowego. Np. MOV A, @R1, ADD A,@R0, MOVX A, @DPTR.

Adresowanie względne służy do adresowania pamięci względem adresu aktualnie wykonywanej instrukcji w pamięci programu. Adres ten jest przechowywany w specjalnie do tego celu przeznaczonym rejestrze, nazywanym **licznikiem programu PC** (*program counter*), **licznikiem rozkazów** lub **wskaźnikiem instrukcji IP** (*instruction pointer*). Adres efektywny jest obliczany jako suma zawartości licznika programu i adresu względnego, gdzie **adres względny** (*relative address*), będący argumentem instrukcji, np. zawarty jest w zakresie [-128,+127]. Np. JNB adresbitu,etykieta - JNB P0.1,LOOP.

2.1.1. Architektury procesorów rdzeniowych mk

Architektury procesorów rdzeniowych można sklasyfikować według typu **mapy pamięci** oraz według typu **listy instrukcji**.

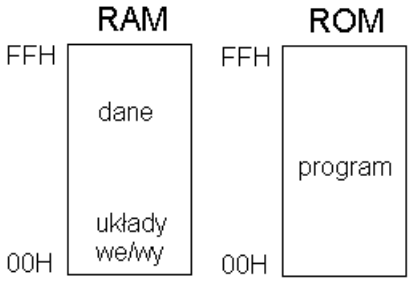
Mapa pamięci (*memory map*) w sposób graficzny przedstawia rozmieszczenie poszczególnych pamięci w **przestrzeni adresowej** jc. Oprócz adresów obszarów RAM, ROM i innych rodzajów pamięci, mapa ta podaje usytuowanie rejestrów uniwersalnych, adresów procedur obsługi przerwań, rejestrów układów we/wy (dostępne przez adresowanie pamięci RAM).

W zależności od typu struktury mapy pamięci, procesory rdzeniowe mogą mieć następującą architekturę:

- architekturę harwardzką,
- zmodyfikowaną architekturę harwardzką,
- architekturę Von-Neumanna.

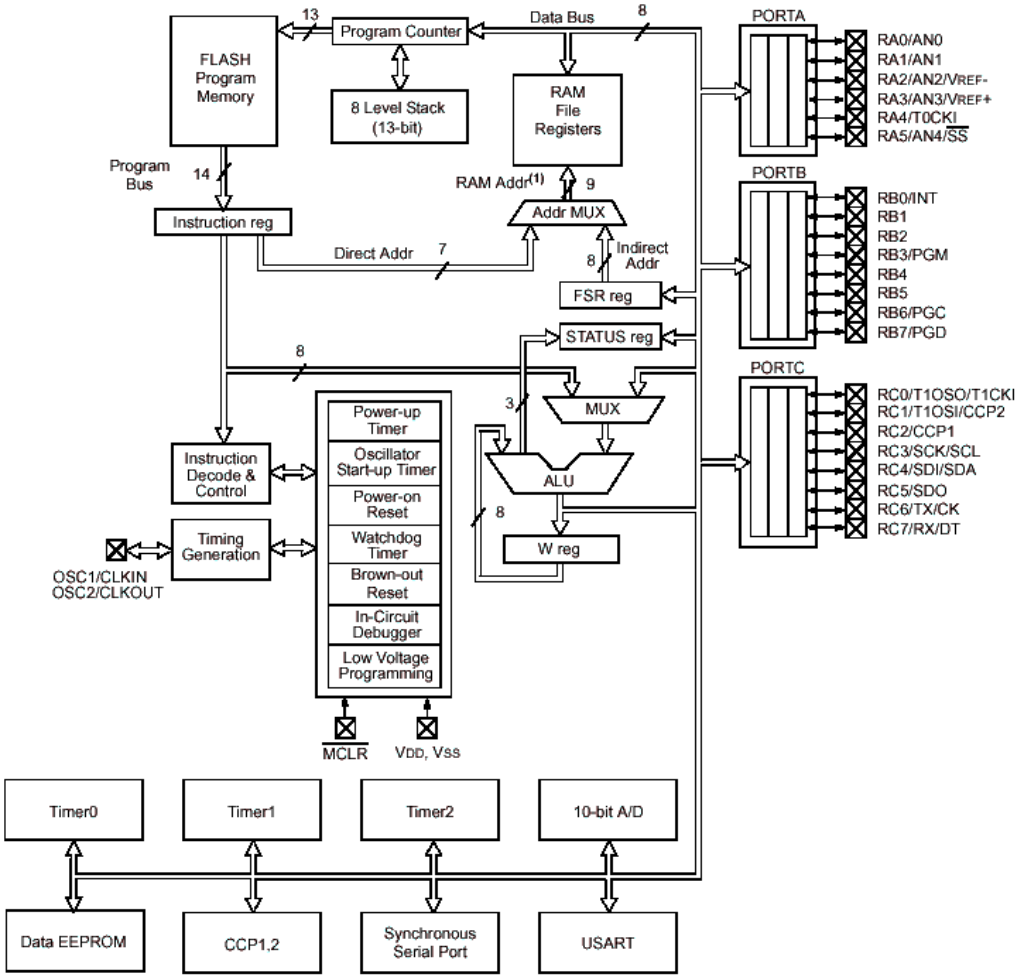
Architektura harwardzka opiera się na użyciu dwóch oddzielnych szyn dla danych i rozkazów, dzięki czemu w trakcie pobierania argumentów wykonywanej właśnie instrukcji można równocześnie zacząć pobieranie następnego słowa rozkazowego (*pre-fetch*). Skraca to

cykl rozkazowy i zwiększa szybkość pracy. Obszary adresowe pamięci danych i programu (wewnętrznych i czasami zewnętrznych) są **rozdzielone.** Pociąga to za sobą niejednoznaczność adresów, ponieważ pod tym samym adresem jc widzi pamięć RAM i ROM (rys. 2.4). W tym przypadku stosuje się inne rozkazy dla pamięci programu i inne dla pamięci danych. Ponadto magistrala danych i rozkazów mają **różną szerokość (długość słowa)**, np. PIC16F87x – magistrala danych 8-bitowa, magistrala rozkazów 14-bitowa (rys. 2.5).



Rys. 2.4. Mapa pamięci dla architektury harwardzkiej

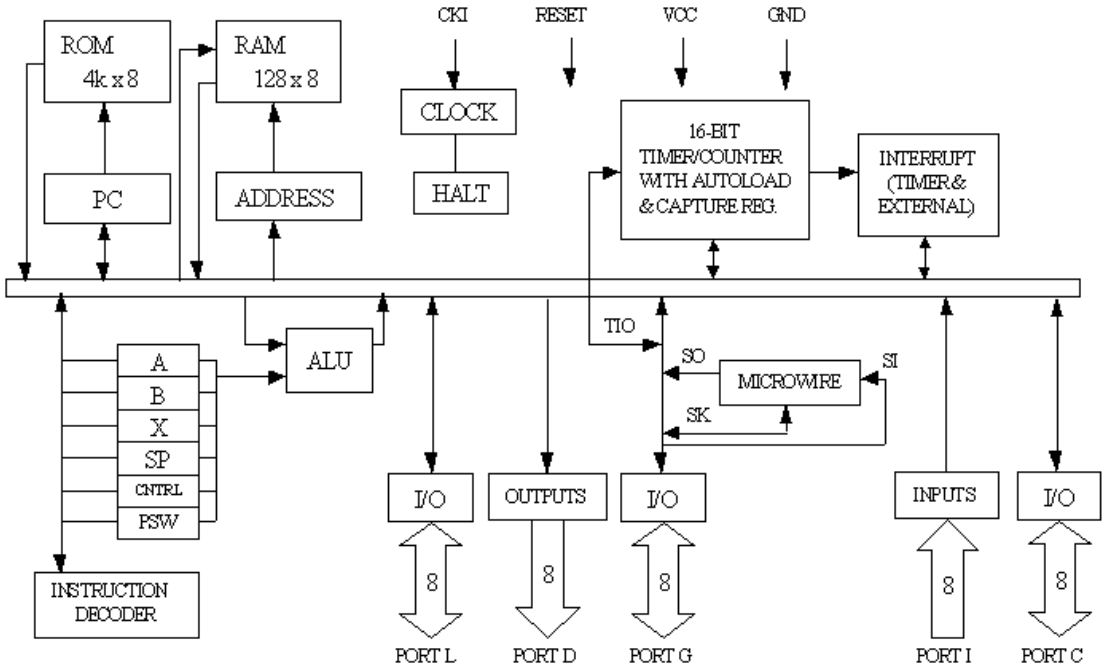
Wadą tego rozwiązania jest utrudniony przepływ danych z pamięci programu do obszaru pamięci operacyjnej, co uniemożliwia stosowanie jednej z podstawowych technik programistycznych (*look-up tables*). Innymi słowy nie jest możliwe indeksowane przesłanie danych z pamięci ROM do RAM, co oznacza np. brak możliwości budowy tabel współczynników stałych w pamięci ROM. Jedynym sposobem wbudowania stałych w program jest ukrycie ich w kodach rozkazów. Np. ADDLW k, MOVLW k,



Rys. 2.5. Schemat blokowy mk PIC16F873/6 firmy Microchip

Zmodyfikowana architektura harwardzka jest rozwiązaniem pośrednim, starającym się połączyć zalety architektury harwardzkiej i Von-Neumanna. Obszary pamięci ROM i RAM są rozdzielone, ale charakteryzują się **taką samą długością słowa**. Dzięki multiplekserom MUX i odpowiedniej organizacji magistrali pamięci ROM i RAM możliwe jest z pewnymi ograniczeniami przesyłanie stałych z pamięci ROM do rejestrów i pamięci RAM. Jedynym rejestrem niewidocznym jako komórka pamięci RAM jest rejestr akumulatora A. Np. dla mk COP880 (rys. 2.6):

LAID ; ładuj A zawartością ROM (PU,A)



Rys. 2.6. Schemat blokowy mk COP880 firmy National Semiconductor

Możliwe jest przesłanie stałej zapisanej w obszarze kodu do obszaru danych tylko w granicach bieżącej strony pamięci ROM określonej przez starszą część licznika rozkazów PU. Pierwotna zawartość akumulatora, służy jako indeks. Dodatkowo procesor wyposażono w możliwość budowy stałych tablic wektorów w pamięci ROM. Właściwość ta pozwala na budowę szybkich algorytmów dekodowania złożonych struktur danych. Oparta jest ona na skoku pośrednim, w którym zawartość akumulatora modyfikuje wartość finalną adresu w pamięci ROM. Działanie tego mechanizmu ograniczone jest do 1 strony ROM (256 bajtów):

JID ; skocz do adresu A na stronie PU.

Architektura Von-Neumanna cechuje się **jednolitą przestrzenią adresową**, w której wszystkie pamięci, rejestry i układy we/wy są umieszczone w jednej, wspólnej przestrzeni adresowej. W architekturze tej zakłada się, że podział przestrzeni adresowej na pamięć programu, pamięć danych oraz obszar we/wy jest czysto umowny i zależy wyłącznie od rozmieszczenia tych elementów w obszarze adresowym podczas projektowania systemu. Mk ma jedną szynę danych wspólną dla danych i programu (rys. 2.7).

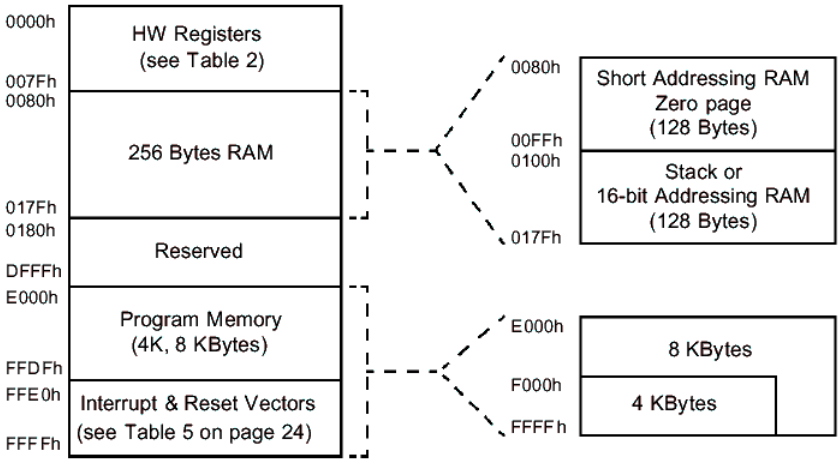
Dzięki temu programowanie jest ułatwione, gdyż dostęp do danych, programu i urządzeń we/wy odbywa się przy użyciu zunifikowanych rozkazów wykorzystujących te same tryby adresowania. Zatem nie istnieje tu potrzeba wprowadzania specjalnych rozkazów pozwalających na przepływ danych pomiędzy pamięcią ROM i RAM. Do tego celu może być

użyty typowy rozkaz adresowy. Tym samym tworzenie tablic stałych, tablicy wektorów, itp. w pamięci ROM nie stanowi problemu. Np.:

```
.send_prog_in_eeprom      ;
       CLR    Y           ; Reset pointer
       LD     A,(prog,Y)  ; Read byte

       segment 'rom2'

.prog  DC.B   $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
       DC.B   $46,$4C,$41,$53,$48,$20,$50,$52,$4F,$47,$52,$41,$4D,$49,$4E,$47
       DC.B   $20,$20,$20,$53,$54,$37,$20,$4D,$69,$63,$72,$6F,$73,$20,$20,$20
```



Rys. 2.7. Mapa pamięci mk ST72215G firmy STMicroelectronics

Kolejny podział architektur procesorów mk można uzyskać korzystając z kryterium typu listy instrukcji. Pozwala ono na wyróżnienie procesorów o:

- architekturze **RISC** (*reduced instruction set computer*),
- architekturze **CISC** (*complex instruction set computer*).

Architektura RISC, czyli o zredukowanej liście instrukcji, odznacza się następującymi cechami:

- procesor jest zbudowany zgodnie z architekturą harwardzką,
- procesor wykorzystuje **przetwarzanie potokowe** (*pipeling*) w celu zwiększenia szybkości wykonywania programu,
- zbiór realizowanych instrukcji jest **ograniczony** i spełnia warunki **ortogonalności** (symetrii).

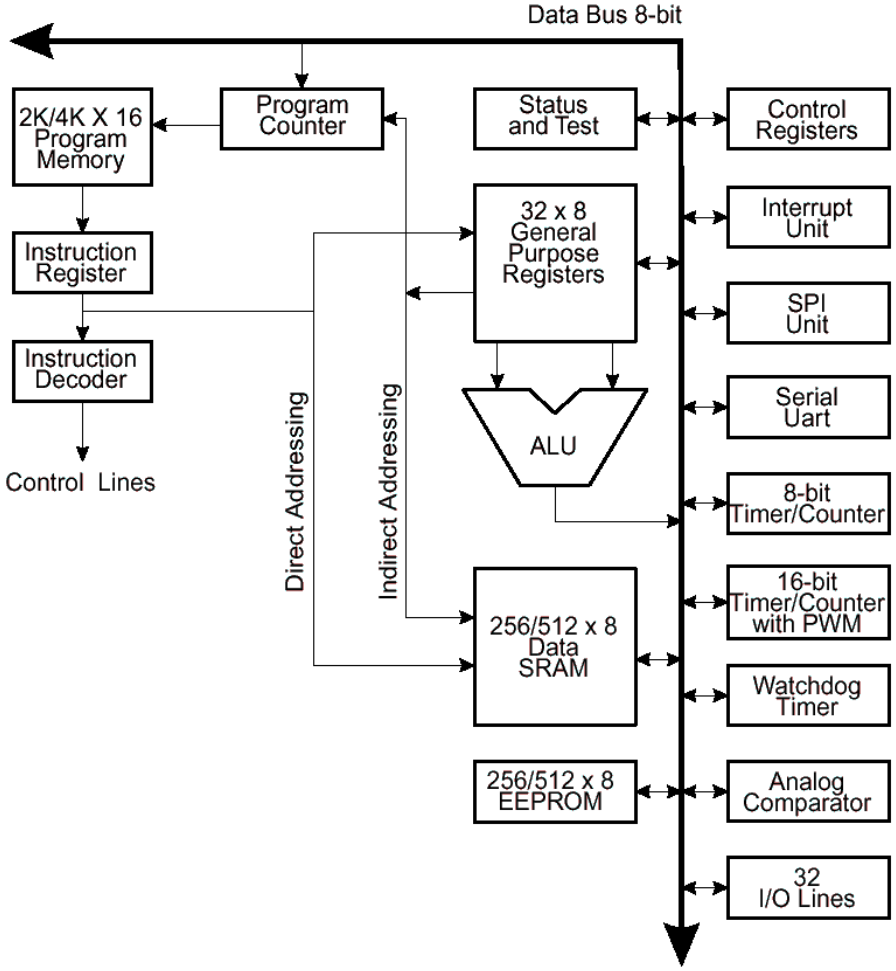
W przetwarzaniu potokowym jc dysponuje pobranymi na zapas instrukcjami, które będą kierowane do współbieżnego wykonania w jej poszczególnych jednostkach wykonawczych. W procesorze tego typu zamiast prostego rejestru instrukcji stosuje się **pamięć FIFO** (*first-in first-out*), która gromadzi kolejkę instrukcji. Instrukcje pobierane z pamięci programu do kolejki w cyklu *pre-fetch* opuszczają ją w takiej samej kolejności i są kierowane do poszczególnych układów wykonawczych.

Pojęcie ortogonalności oznacza unifikację instrukcji według następujących zasad:

- każda instrukcja może operować na **dowolnym rejestrze roboczym**. Procesor nie ma więc wyróżnionych rejestrów, które są specjalizowane do wykonywania określonych rodzajów operacji,
- każda instrukcja może wykorzystywać **dowolny tryb adresowania** argumentów,

- nie ma ukrytych związków między instrukcjami (efektów ubocznych), które powodowałyby nieprzewidziane reakcje systemu w zależności od kontekstu użycia rozkazów w programie,
- kody rozkazów i formaty instrukcji są **zunifikowane**. W szczególności wszystkie instrukcje zajmują w pamięci programu taką samą liczbę bajtów.

Ortogonalność zbioru instrukcji radykalnie upraszcza budowę układu sterowania, który może realizować cykl wykonania każdego rozkazu według identycznego algorytmu. Stąd prostszy układ sterowania może pracować znacznie szybciej, dlatego cykl rozkazowy ulega skróceniu. Np. mk AT90S8515 firmy Atmel (rys. 2.8).



Rys. 2.8. Schemat blokowy procesora rdzeniowego mk AT90S8515 firmy Atmel

Klasyczna architektura RISC jest stosowana w mk rzadko. Najczęściej można znaleźć elementy tej architektury, ale ortogonalność instrukcji nie jest pełna.

Architektura CISC charakteryzuje się **rozbudowaną liczbą instrukcji** (często powyżej 100). Przeciwstawia się ją architekturze RISC. Cecha ortogonalności nie jest zachowana. Instrukcje są wąsko specjalizowane, współpracują na ogół tylko z określonymi rejestrami i wymagają stosowania określonych trybów adresowania.

2.2. Pamięci

W mk można wyróżnić następujące typy pamięci:

- **pamięć programu** (zawierająca kod programu, tablice stałych, wektor resetu i przerwań),
- **pamięć danych** (przechowująca zmienne),
- **stos sprzętowy** (obsługi przerwań i wywołań funkcji odkładają na niego bieżącą wartość licznika rozkazów i po zakończeniu działania „zdejmują" ją),
- **pamięć EEPROM** (przechowuje zmienne lub tablice stałych, które po wyłączeniu zasilania nie mogą ulec skasowaniu).

Pamięć programu jest wykonana w technologii ROM, EPROM, OTP lub FLASH. Natomiast pamięć danych najczęściej jest typu SRAM.

Przegląd podstawowych typów pamięci zestawiono w tabeli 2.1.

Tabela 2.1. Przegląd podstawowych układów pamięci

Własności pamięci	Stosowane typy pamięci
Zawartość pamięci nie zanika wraz z wyłączeniem napięcia zasilania. Z pamięci można czytać, lecz nie można do niej wpisywać danych. Umieszczanie danych w pamięci wymaga specjalnego procesu, zwanego programowaniem.	**ROM (*Read Only Memory*)** – programowanie zawartości pamięci następuje w procesie produkcyjnym i nie może być przeprowadzone przez użytkownika. **EPROM (*Erasable Programmable ROM*)** – pamięci z możliwością kasowanie dotychczasowej zawartości promieniami ultrafioletowymi i prowadzania nowej zawartości za pomocą zewnętrznego programatora. Umieszczane są w obudowach z okienkiem kwarcowym w celu umożliwienia kasowania. **OTP (*One Time Programmable*)** – pamięci typu EPROM umieszczane w obudowach bez okienka kwarcowego. Dlatego możliwe jest tylko jednokrotne zaprogramowanie pamięci bez możliwości skasowanie jej zawartości. **FLASH (*Bulk Erasable Non-Volatile Memory*)** – pamięci z możliwością kasowanie zawartości i programowania bezpośrednio w systemie mikroprocesorowym.
Zawartość pamięci zanika wraz z wyłączeniem napięcia zasilania. Z pamięci można zarówno czytać, jak i wpisywać do niej dane.	**SRAM (*Static Random Access Memory*)** – pamięci RAM statyczne. Są to pamięci o krótkich czasach dostępu, prostsze w obsłudze przez jednostkę centralną, ale droższe. **DRAM (*Dynamic Random Access Memory*)** – pamięci RAM dynamiczne. Są to pamięci tańsze, ale ich obsługa przez jednostkę centralną jest bardziej skomplikowana. Polega to na konieczności wykonywania w krótkich odstępach czasu określonych operacji na pamięci (tzw. odświeżaniu). W przeciwnym razie dane zawarte w pamięci dynamicznej zanikają.

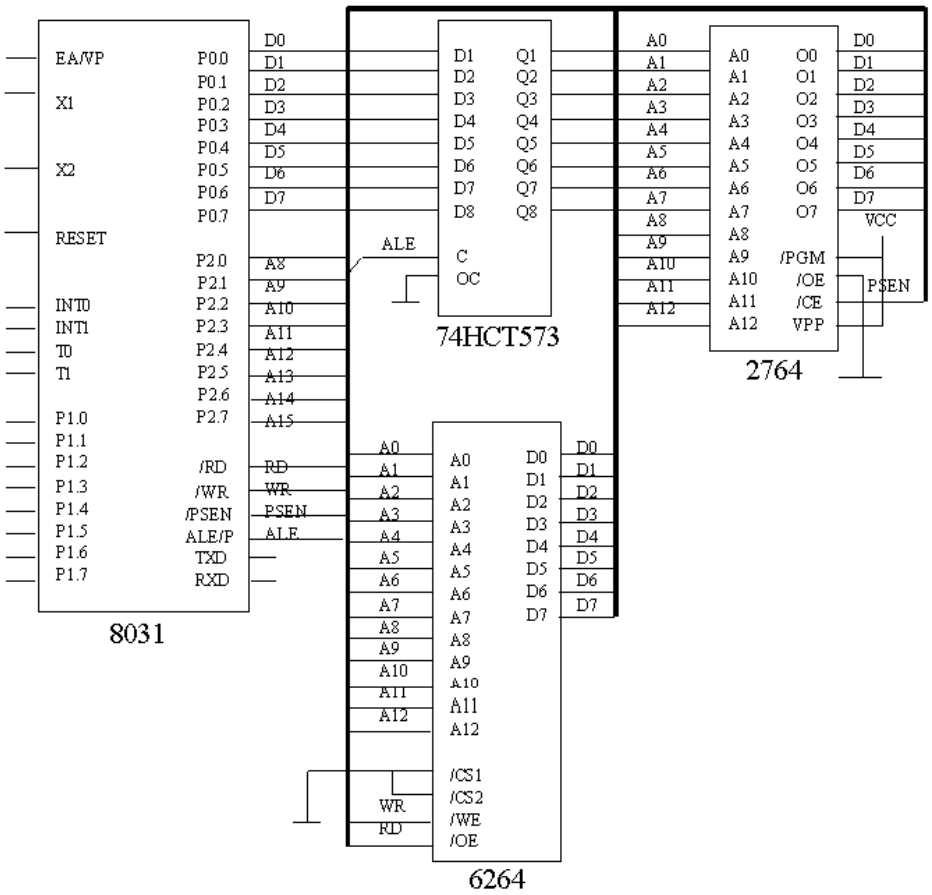
2.3. Struktury mikrokontrolerów

Zgodnie z definicją mk, w jego strukturze zawarte są zarówno układy peryferyjne, jak i układy pamięci. Jednak w praktyce czasami zachodzi potrzeba wykorzystania zarówno pamięci zewnętrznej, jak i zewnętrznych układów peryferyjnych. Stąd niektóre mk umożliwiają przyłączenie zewnętrznych pamięci programu i danych.

Ze względu na sposób korzystania z zewnętrznych pamięci możemy wyróżnić:

- mk udostępniające szyny systemowe (szyna danych, adresowa i sterująca) poprzez **wyprowadzenia portów**,
- mk udostępniające **bezpośrednio** szyny systemowe,
- mk **zamknięte** (*embedded*).

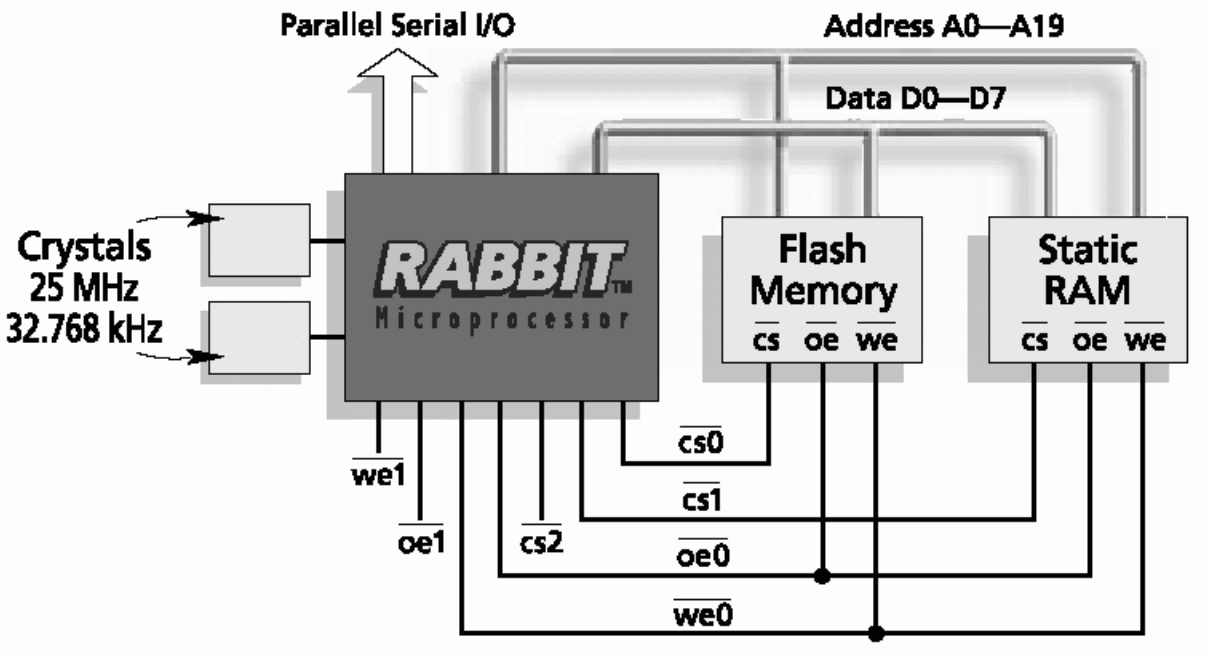
Mk udostępniające szyny systemowe poprzez wyprowadzenia portów – w tej strukturze systemu szyny systemowe są dostępne dla użytkownika jako alternatywne funkcje wyprowadzeń portów. Mk daje zatem możliwość podłączenia układów pamięci zewnętrznej, którą podłącza się do odpowiednich portów mk. Wadą tego rozwiązania jest rezygnacja z części portów. Np. na rys. 2.9 widać, iż należy skorzystać z portów P0 i P2 oraz części wyprowadzeń portu P3. Zatem nie można korzystać z linii tych portów jako we/wy sygnałów sterujących.



Rys. 2.9. Budowa mse z mk udostępniającym szyny systemowe jako alternatywne wyprowadzenia portów, na przykładzie mk 8031

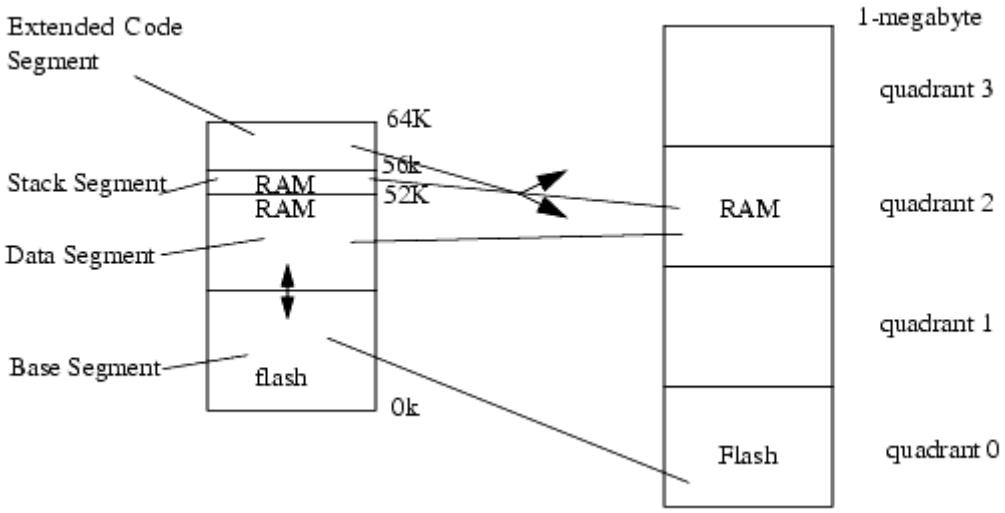
Mk udostępniające bezpośrednio szyny systemowe – w tej strukturze szyny systemowe mk są dostępne dla użytkownika bezpośrednio jako wyprowadzenia układu scalonego bez funkcji alternatywnych przypisanych tym wyprowadzeniom. Mk daje zatem możliwość przyłączenia układów pamięci zewnętrznych, bez potrzeby rezygnacji z niektórych portów

mk. Najczęściej mk te nie posiadają wewnętrznych pamięci programu i danych. Np. mk Rabbit 2000 (rys. 2.10).



Rys. 2.10. Budowa mse z mk udostępniającym szyny systemowe bezpośrednio na wyprowadzeniach układu scalonego

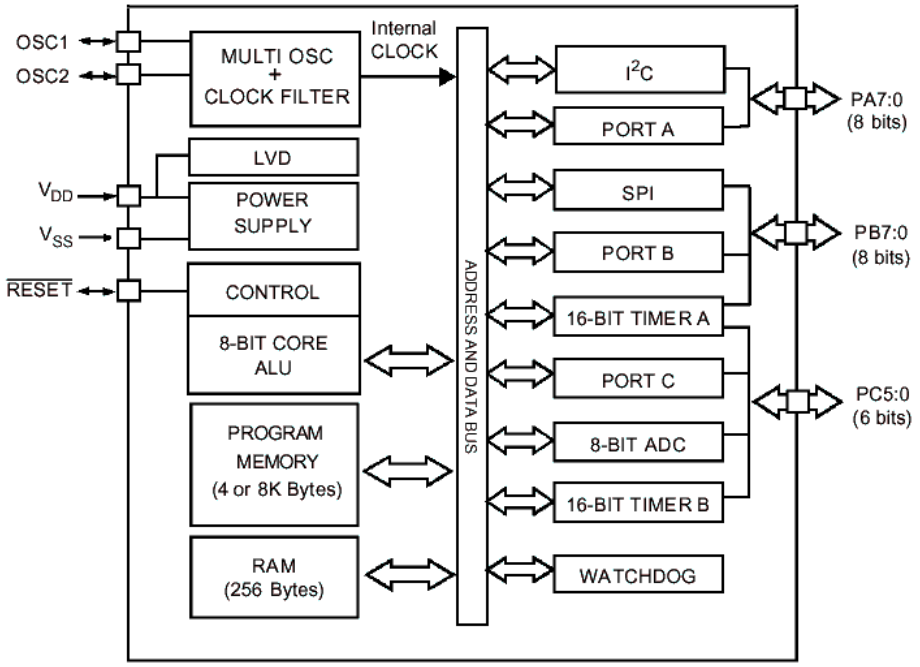
Podejście takie umożliwia korzystanie ze znacznie większych pamięci. Stąd tego typu struktura jest stosowana przeważnie dla mk 16 i 32-bitowych. Np. mk Rabbit 2000 pozwala na zaadresowanie 1MB pamięci. Sposób mapowa i przyporządkowanie funkcji poszczególnym obszarom pamięci pokazano na rys. 2.11.



Rys. 2.11. Sposób mapowania pamięci dla mk Rabbit 2000

Mk zamknięte – w tej strukturze systemu szyny systemowe nie są w ogóle dostępne dla użytkownika. Mk korzysta wyłącznie z wewnętrznej pamięci. Oczywiście można do niego podłączać zewnętrzne urządzenia peryferyjne poprzez jego porty. Tego typu struktura jest stosowana w przypadku prostych mk 8-bitowych (które są głównym tematem wykładu).

Na rys. 2.12 pokazano schemat takiego mk (mk ST72215G firmy STMicroelectronics).



Rys. 2.12. Budowa mse z mk nie udostępniającym szyn systemowych – zamkniętym, na przykładzie mk ST72215G

Jakkolwiek brak dostępu do magistrali może być postrzegany jako wada, jednak po dokładnej analizie można stwierdzić, że rozwiązanie takie ma następujące zalety:

1. **Zwiększenie niezawodności mikrosterownika.**

Magistrala jest jednym z najczulszych na zakłócenia systemem interfejsowym. Zamknięcie magistrali wewnątrz układu scalonego, odseparowanie jej od zacisków we-wy powoduje znaczący wzrost niezawodności systemu. Jądro systemu w postaci procesora rdzeniowego uzyskało w ten sposób bardzo dobrą izolację od pozostałej części systemu. Nastąpiło zmniejszenie i ograniczenie długości magistrali co ma znaczenie ze względu na rosnące wymagania na kompatybilność elektromagnetyczną układu.

2. Zmniejszenie poboru mocy.

Magistrala uniwersalna jest jednym z najbardziej energochłonnych systemów komunikacyjnych. Bez względu na ilość aktualnie odbierających informację odbiorników (zwykle 1) nadajnik musi ją nadać w takiej formie fizycznej, aby mogły ją odbierać wszystkie możliwe do podłączenia odbiorniki. Ograniczenie dostępu do magistrali do znanej liczby urządzeń o określonych parametrach pozwala na przeprowadzenie optymalizacji energo-czasowej, czego efektem jest znaczący spadek poboru mocy przez mk zamknięte przy zachowaniu bardzo dobrych parametrów czasowych.

3. Zwiększenie szybkości pracy.

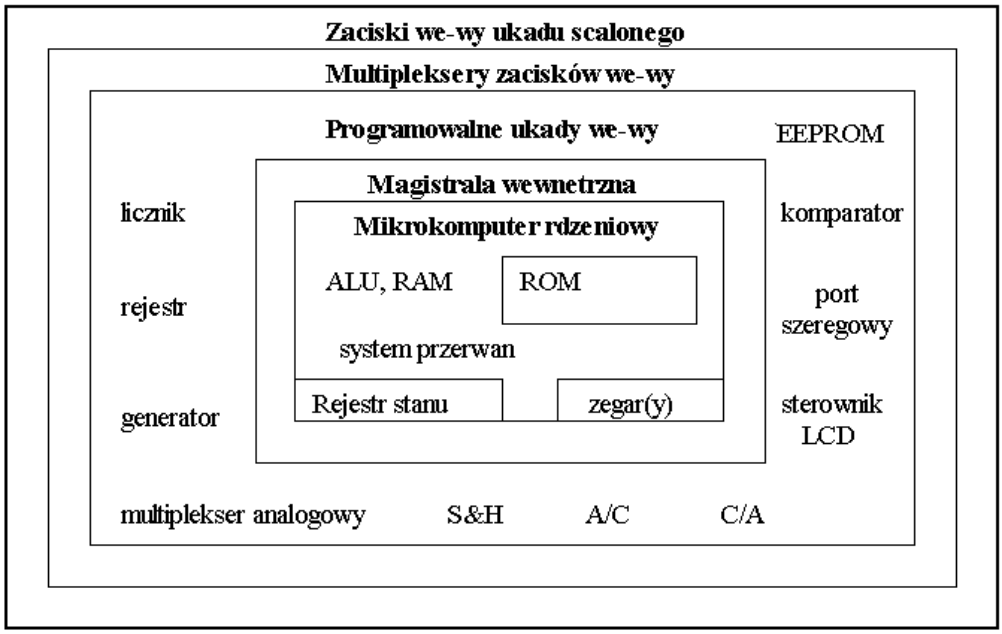
Precyzyjnie zdefiniowana i niezmienna magistrala wewnętrzna mk pozwala na optymalizację jego parametrów czasowych. mk wyposażone są w rozległe i programowalne układy dystrybucji sygnałów zegarowych wewnątrz układu scalonego, pozwalające na osiągnięcie dużych szybkości wykonania programu bez znaczącego wzrostu poboru mocy.

4. **Zmniejszenie ilości zacisków zewnętrznych i zwiększenie ich elastyczności.**

Zadaniem zacisków zewnętrznych jest zapewnienie przepływu sygnałów pomiędzy otoczeniem a układem we-wy wbudowanymi w mk. Ponieważ układy te w większości zastosowań wykorzystywane są w sposób sekwencyjny, możliwe jest dołączenie do jednego zacisku fizycznego kilku wewnętrznych układów we-wy. Pozwala to na dynamiczna rekonfigurację funkcji zacisków w takich mk.

Brak dostępu do magistrali wewnętrznej jest często rekompensowany wyposażeniem mk w specjalizowane porty do wymiany informacji z układami zewnętrznymi. Najczęściej stosuje się transmisję szeregową: SPI, I^2C, RS232 itp.

Na rys. 2.13 pokazano warstwowy model mk zamkniętego.



Rys. 2.13. Warstwowy model mk zamkniętego

Mk zamknięty składa się z kilku warstw:

1. Centralną część zajmuje **mikroprocesor rdzeniowy**. Jest to typowy mikroprocesor o architekturze magistralowej. Zawiera on wszystkie, niezbędne do pracy zasoby:
   a) jednostka ALU;
   b) pamięć danych (RAM), rejestry;
   c) pamięć programu (ROM);
   d) układ sterujący;
   e) generatory sygnałów zegarowych, systemy przerwań itp.

Jego cechą charakterystyczną jest **nierozszerzalność zasobów** oraz **duża izolacja** od świata zewnętrznego (zapewnia to niezawodność).

2. **Magistrala wewnętrzna** służy do wymiany informacji pomiędzy różnymi wewnętrznymi blokami składowymi w mk. Głównym zadaniem jest organizacja pracy mikrosystemu znajdującego się wewnątrz układu scalonego.
3. Warstwa **programowalnych układów we/wy** charakteryzuje dany typ mk. Zestawy urządzeń we/wy dostosowuje się do przewidywanych zastosowań danego mk. Cechą charakterystyczną układów we/wy spotykanych w mk jest ich duża autonomiczność. Układy te mogą wykonywać swoje funkcje niezależnie i współbieżnie z komputerem rdzeniowym i innymi układami. Możliwa jest często praca układu we/wy przy zablokowanym (np. dla oszczędności energii) procesorze rdzeniowym.
4. **Multipleksery zacisków** niezbędne są w przypadku, gdy sumaryczna liczba wejść i wyjść wewnętrznych układu jest większa od ilości zacisków fizycznych. Zachodzi to w większości mk. Umożliwiają ograniczenie ilości zacisków fizycznych przy zachowaniu elastyczności mk, dając możliwość dynamicznej zmiany funkcji danego zacisku fizycznego. Sumaryczny stosunek zaciski logiczne / fizyczne waha się od około 1.5 - 2 przy prostych mk, do 4 w mk klasy średniej i wyższej.

5. Warstwa **zacisków fizycznych** łączy mk ze światem zewnętrznym. Jej parametry są istotne dla zapewnienia prawidłowej współpracy mk z układami otaczającymi go. Parametry elektryczne tej warstwy (obciążalność, max. napięcie, tryb pracy nadajnika itp.) mogą być zmieniane statycznie dla serii mk poprzez odpowiednie uzgodnienia z producentem. Odbywa się to poprzez wybór odpowiednich opcji w czasie przystępowania do produkcji masek opisujących sposób wykonania pamięci programu. Niektóre mk pozwalają na dynamiczną zmianę części parametrów układów współpracujących z zaciskami.

2.4. Elementy składowe procesora rdzeniowego

W mk, w skład procesora rdzeniowego, poza samym procesorem przetwarzającym instrukcje i pamięciami, znajdują się również inne urządzenia niezbędne do funkcjonowania mk. Są to między innymi:

- **oscylator** i **układ dystrybucji sygnałów zegarowych**,
- **układ resetu** (często bardzo rozbudowany),
- **układy nadzorujące** (często traktuje je się jako układy peryferyjne mk),
- **system przerwań**.

2.4.1. Oscylator i dystrybucja sygnałów zegarowych

W mk można wyróżnić dwa źródła sygnału zegarowego:

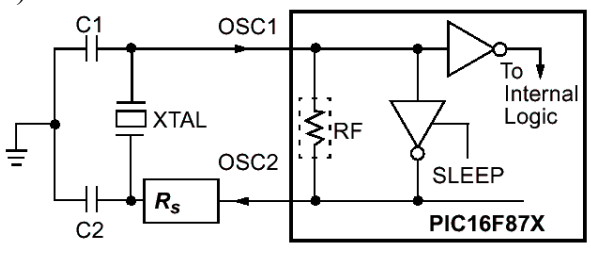
- sygnał zegarowy generowany „**wewnątrz**" mk (wewnętrzny układ RC),
- sygnał zegarowy generowany z **wykorzystaniem zewnętrznych elementów**.

Każdy mk posiada **dedykowane dwie końcówki** służące wyłącznie do podłączenia zewnętrznych elementów wykorzystywanych do generacji (stabilizacji) sygnału zegarowego. Jedna z nich jest wejściem (najczęściej oznaczana OSC1 lub XTAL1), druga wyjściem (OSC2 lub XTAL2) wewnętrznego układu generatora. Można do nich podłączyć oscylator kwarcowy, ceramiczny, układ RC lub podać na końcówkę wejściową zewnętrzny sygnał prostokątny.

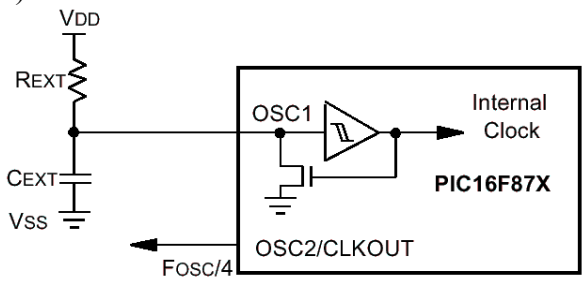
Stąd możemy wyróżnić następujące metody generacji sygnału zegarowego:

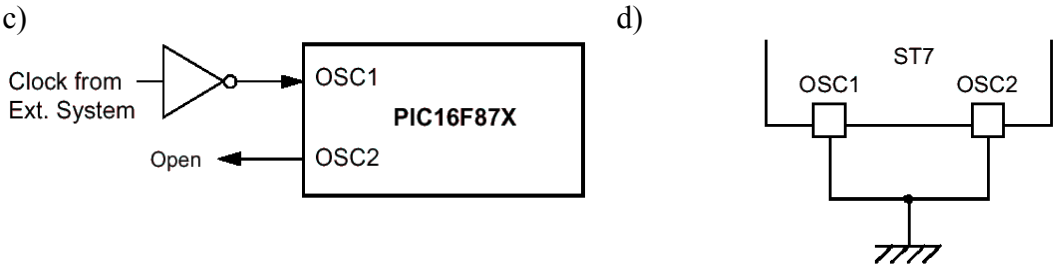
- z rezonatorem **kwarcowym/ceramicznym** (rys. 2.14a),
- z **rezonatorem RC** (najtańszy ale najgorsze parametry generowanego przebiegu) (rys. 2.14b),
- z podaniem **zewnętrznego sygnału prostokątnego** na wejście OSC1 (rys. 2.14c),
- z **wewnętrznym rezonatorem RC** (rys. 2.14d).

a)



b)





Rys. 2.14. Sposoby generowania sygnału zegarowego

Ponadto możemy wyróżnić następujące tryby pracy oscylatorów kwarcowych/ceramicznych:

- LP – generowanie niskich częstotliwości dla zapewnienia minimalnego poboru mocy (*Low Power Crystal*),
- XT – generowanie średnich częstotliwości (*Crystal/Resonator*),
- HS – generowanie wysokich częstotliwości (*Hight Speed Crystal/Resonator*).

Producenci zawierają w dokumentacji mk tabele wartości kondensatorów dla poszczególnych trybów pracy oscylatora/rezonatora. Dla przykładu podano tabelę wartości kondensatorów C1 i C2 dla wszystkich trybów pracy oscylatora kwarcowego mk PIC16F873.

Tabela 2.2. Wartości kondensatorów w zależności od trybu pracy oscylatora kwarcowego

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

Metodę generacji sygnału zegarowego jak i tryb pracy oscylatora wybiera się na **etapie programowania** mk, czyli wprowadzania kodu programu do jego pamięci FLASH. Służą do tego dwa bity zawarte w rejestrze konfiguracyjnym mk (rys. 2.15). Rejestr ten znajduje się w pamięci FLASH pod adresem 2007H i jest dostępny (można do niego pisać i z niego czytać) wyłącznie w trybie programowania wykorzystując w tym celu programator i dołączone do niego oprogramowanie np. MPLAB.

Rejestr konfiguracyjny jest 14-bitowy. Na rys. 2.15 pokazano jego fragment związany z omawianymi zagadnieniami. Z rys. wynika, iż niezbędne jest skonfigurowanie oscylatora mk zgodnie z tym, co zostało zamontowane na płytce drukowanej. W przeciwnym razie mk nie będzie działał!

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
						bit 0

BODEN: Brown-out Reset Enable bit
1 = BOR enabled
0 = BOR disabled

$\overline{\text{PWRTE}}$: Power-up Timer Enable bit
1 = PWRT disabled
0 = PWRT enabled

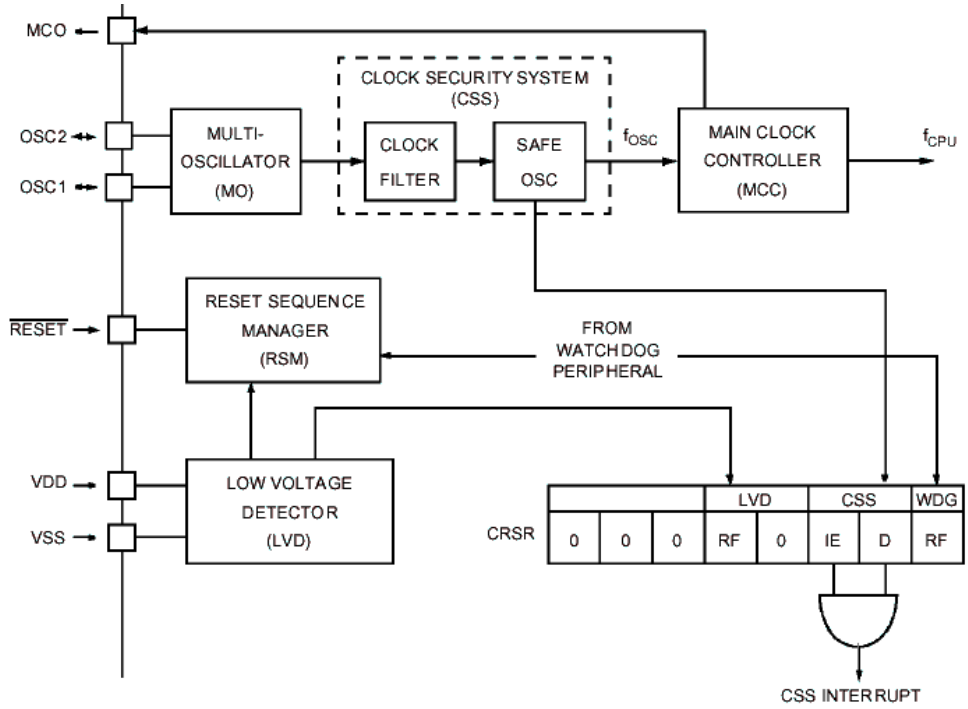
WDTE: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

FOSC1:FOSC0: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Rys. 2.15. Fragment rejestru konfiguracyjnego mk PIC16F873

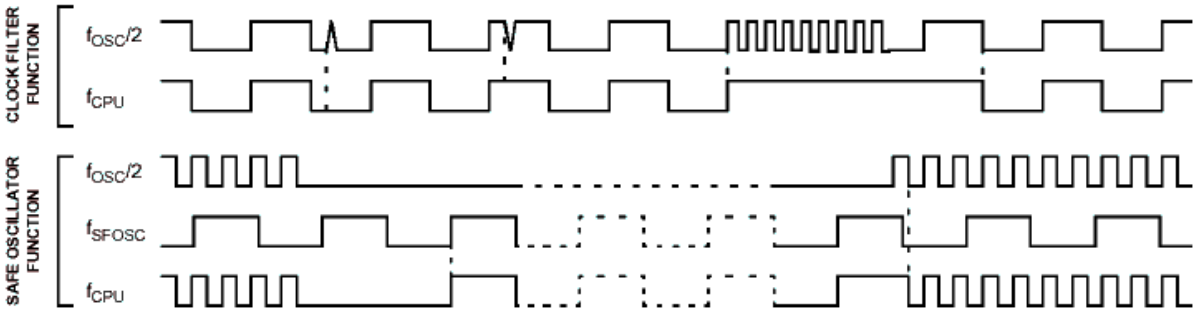
Ponieważ od mk wymaga się niezawodności niektóre z nich wyposażone są w specjalne generatory zapewniające ciągłą, bezawaryjną generację sygnału zegarowego oraz jego filtrację, np. mk ST72215G. Ponadto cechą charakterystyczną współczesnych mk są rozbudowane **układy generacji i dystrybucji sygnałów zegarowych**. Potrzeba ta wynika z dużej ilości pracujących autonomicznie modułów we/wy wymagających różnych sygnałów zegarowych. Same procesory rdzeniowe często są wyposażone w 2 generatory zegarowe, jeden o częstotliwości zapewniającej maksymalną szybkość pracy i drugi pozwalający na pracę z częstotliwością 32,768 kHz, np. Rabbit 2000. Sieć dystrybucji sygnałów zegarowych pozwala na programowanie częstotliwości sygnałów dostarczonych do różnych bloków. Możliwe jest też całkowite odcięcie sygnałów zegarowych od wybranych bloków. Jest to jedna z technik redukcji prądu zasilania pobieranego przez mk.

Parametry dystrybucji sygnałów zegarowych określają graniczne możliwości czasowe danego mk.



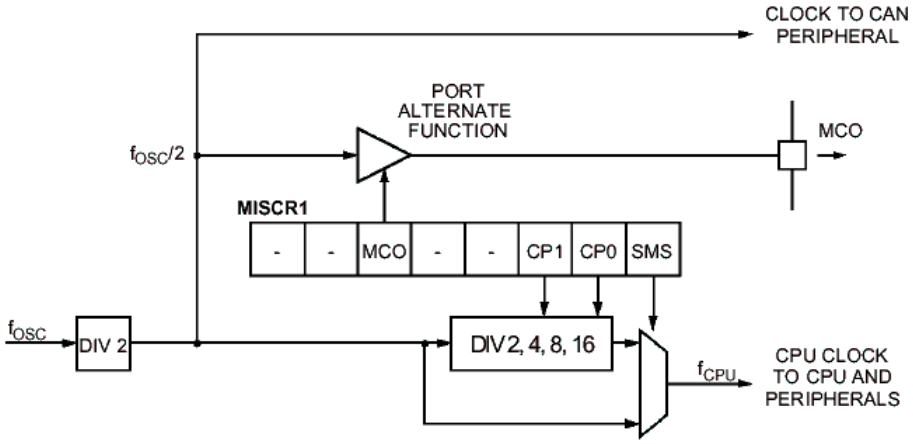
Rys. 2.16. Układ generacji i dystrybucji sygnałów zegarowych mk ST72215G

Np. mk ST72215G posiada układ ochrony zegara CSS (*Clock Security System*), który składa się z układu filtrującego sygnał zegarowy i układu „bezpieczny oscylator" SO (*Save Oscilator*). Pierwszy układ filtruje pojawiające się na wejściu impulsy zakłócające oraz sygnały o większej częstotliwości niż założono. Drugi układ w przypadku zaniku sygnału zegarowego, np. uszkodzenie oscylatora kwarcowego, po okresie sygnału generowanego przez siebie, załącza swój sygnał jako sygnał zegarowy do czasu powrotu właściwego sygnału zegarowego (rys. 2.17). Oprogramowanie na ten mk powinno być tak napisane, aby mk wykonał odpowiednie zadania związane z trybem awaryjnym w jakim się znalazł.



Rys. 2.17. Funkcje spełniane przez CSS i SO

Na rys. 2.18. pokazano układ dystrybucji sygnałów zegarowych MCC (*Main Clock Controller*) mk ST72215G. Dostarcza on sygnał zegarowy do jc i wewnętrznych urządzeń peryferyjnych. Składa się z programowalnego **preskalera** i układu dostarczającego sygnał do zewnętrznych urządzeń. Preskaler pozwala na ustawienie odpowiedniej częstotliwości sygnału zegarowego. Dzięki temu możliwa jest redukcja pobieranej mocy przez mk, bo jak wiadomo moc pobierana przez układy CMOS jest proporcjonalna do ich częstotliwości pracy.



Rys. 2.18. Układ dystrybucji sygnału zegarowego mk ST72215G

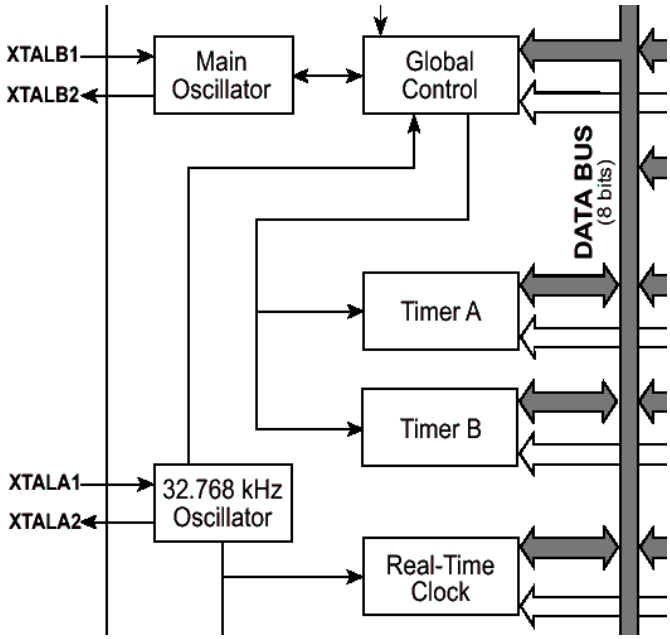
2.4.2. Techniki redukcji mocy i tryby specjalne mk

Bezpośrednio z układami dystrybucji sygnałów zegarowych związane są techniki redukcji mocy i tryby specjalne mk.

Często stawia się mk rygorystyczne wymagania co do ilości pobieranej energii w czasie ich pracy. Stosuje się je często w urządzeniach zasilanych bateryjnie i pracujących w trybie ciągłym tj. 24 godz. na dobę.

Jak wspomniano wcześniej, zależność poboru mocy układów CMOS zależy od częstotliwości pracy, stąd podstawową metodą redukcji mocy jest **elastyczne sterowanie szybkością pracy** w zależności od aktualnych potrzeb. Służy do tego sieć dystrybucji sygnałów zegarowych w mk.

Kolejnym sposobem redukcji poboru mocy jest zastosowanie **dwóch oscylatorów**. Jeden pracuje z maksymalną częstotliwością – główny oscylator – (np. 10MHz lub 40MHz), drugi z częstotliwością niską np. 32,768 kHz. Przejście z wysokiej do niskiej częstotliwości powoduje spadek pobieranej mocy o około trzy rzędy. Np. mk Rabbit 2000 (rys. 2.19).



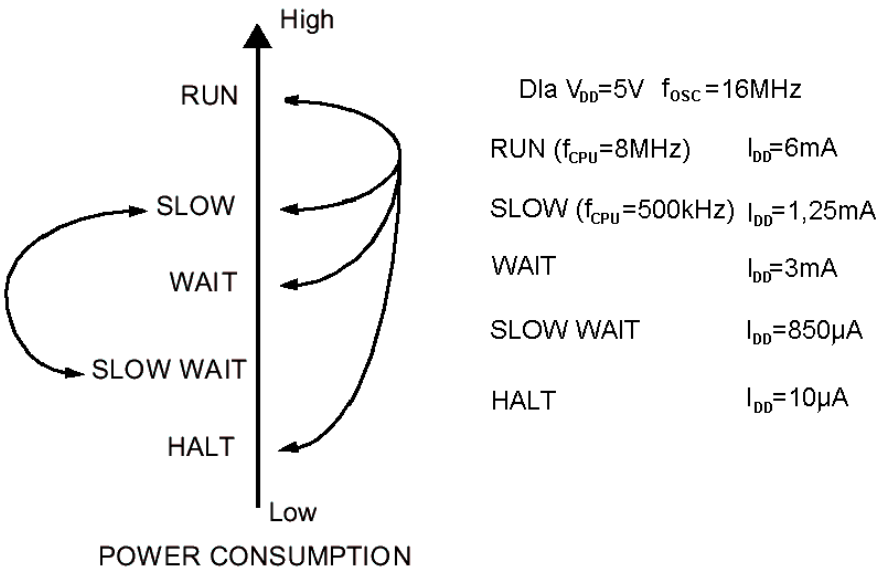
Rys. 2.19. Fragment schematu blokowego mk Rabbit 2000

Sieć dystrybucji sygnałów zegarowych umożliwia również odłączanie od sygnału zegarowego jc, jak i urządzeń peryferyjnych lub wręcz wstrzymanie pracy oscylatora. Stąd mk może znaleźć się w następujących specjalnych **trybach pracy**:

- tryb pełnej aktywności (*RUN*),
- tryb, w którym **nie pracuje procesor**, a **pracują wszystkie urządzenia peryferyjne** (*WAIT* lub *SLEEP*) – inaczej **tryb uśpienia**,
- tryb, w którym **nie pracuje procesor**, a **pracują niektóre urządzenia peryferyjne** (układ przerwań, porty równoległe, przetworniki A/C zasilane zewnętrznym sygnałem zegarowym),
- tryb pełnego wstrzymania (zamrożenia) pracy mk – **zatrzymany układ oscylatora**, zatem żadne urządzenie nie pracuje, stan rejestrów i pamięci RAM jest „zamrożony” (*STOP* lub *HALT*).

Dla niektórych mk np. COP880, przy korzystaniu z zewnętrznego źródła sygnału zegarowego możliwe jest wstrzymanie tego sygnału, co powoduje „zamrożenie” mk. Ponowne pojawienie się tego sygnału „odmraża” mk i kontynuowany jest przerwany cykl rozkazowy. W tym przypadku układ oscylatora i dystrybucji sygnałów zegarowych musi spełniać wysokie wymagania czasowe. Musi nadzorować jakością generowanych sygnałów i synchronizować zmiany stanu procesora.

Przy wchodzeniu w tryby specjalne mk istotne jest aby przez zaciski we/wy **nie płynął prąd**, gdyż wartość tego prądu może być np. o kilka rzędów większa niż jego pobór przez sam mk w stanie STOP.



Rys. 2.20. Specjalne tryby pracy dla mk ST72215G

Na rys. 2.20 pokazano graf przejść do poszczególnych trybów pracy mk ST72215G. W ryby WAIT i HALT wchodzi się wykonując odpowiednie instrukcje, natomiast tryb SLOW uzyskuje się ustawiając odpowiednie bity sterujące preskalerem (rys. 2.18). Aby przejść do trybu SLOW WAIT należy najpierw przejść w tryb SLOW a następnie wydać komendę WFI.

Wejście w specjalny tryb pracy mk można uzyskać w następujący sposób:
- wykonując odpowiednią, przeznaczoną do tego celu **instrukcję** (np. SLEEP w AT90S8515 i PIC16F873, WFI, HALT w ST72215G),
- ustawiając odpowiedni **bit** (np. dla AT89C51 w rejestrze PCON: bit IDL – stan wyłączenia jc, bit PD – stan zatrzymania mk).

Są następujące sposoby wyjścia z trybu uśpienia:
- pojawienie się **przerwania** zewnętrznego lub przerwania od urządzeń peryferyjnych (przerwania te muszą być wcześniej odblokowane),
- **reset zewnętrzny** mk,
- **reset wywołany przez** niezablokowany, aktualnie pracujący **układ watchdog**.

Często jest tak, że mk „obudzony” przez przerwanie obsługe je, a następnie ponownie przechodzi w tryb uśpienia.

Po resecie mk jest w stanie początkowym i pracuje zgodnie z programem zawartym w pamięci programu.

Ze stanu zatrzymania mk można wyjść wyłącznie na dwa sposoby:
- **reset zewnętrzny** mk,
- **reset wywołany przez** niezablokowany, aktualnie pracujący **układ watchdog**.

W tym przypadku układ watchdog musi być taktowany własnym generatorem (najczęściej jest to generator RC wbudowany w strukturę mk).

2.4.3. Reset mk

Sygnał RESET służy do inicjalizacji pracy mk, czyli wprowadzenia go w **stan początkowy**. Inicjalizacja polega najczęściej na wyzerowaniu licznika rozkazów, ustawienia go na początek wykonywania programu. Urządzenia we/wy i rejestry sterujące są ustawiane w tryb standardowy (spoczynku). Uniwersalne końcówki we/wy są ustawione jako wejścia o wysokiej impedancji, aby minimalnie wpływać na otoczenie mk (w dokumentacji każdego mk znajduje się informacja o stanie wszystkich rejestrów i portów po resecie mk).

Dla niektórych mk po resecie może być czytany rejestr stanu określający tryb pracy mk, stany (poziomy) sygnałów na niektórych końcówkach, a następnie czytany adres początku programu znajdujący się w obszarze obsługi resetu.

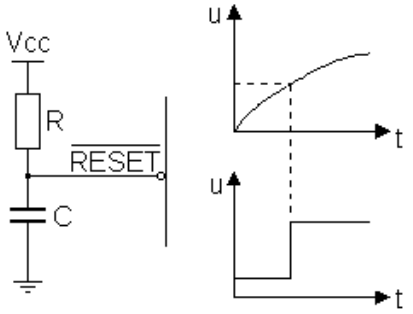
Dla mk o architekturze harwardzkiej po resecie wykonywana jest instrukcja najczęściej spod adresu 00h (np. 80C51, AT90S8515, PIC16F873, COP880). Natomiast dla mk bazujących na architekturze Von-Neumanna adres ten znajduje się zazwyczaj na końcu obszaru adresowania pamięci, gdyż początkowe adresy najczęściej przypisane są urządzeniom we/wy i pamięci RAM (np. ST72215G – po resecie wykonywana jest instrukcja spod adresu FFFEh-FFFFh).

Możemy wyróżnić następujące źródła resetu:

- reset po **włączeniu zasilania**,
- reset wywołany **zewnętrznym sygnałem RESET**,
- reset **programowy** wywołany przez **ustawienie odpowiedniego bitu**,
- reset wywołany przez **układ watchdog**,
- reset wywołany przez **układy nadzorujące** poprawność pracy mk (np. od układu wykrywającego spadek napięcia zasilania LVD (*Low Voltage Detector*) w ST72215G, BOR (*Brown-out Reset*) w PIC16F873).

Każdy mk posiada dedykowaną końcówkę (pin) do której przypisany jest zewnętrzny sygnał resetu. Sygnał ten umożliwia asynchroniczny reset mk. Najczęściej jest on aktywny poziomem niskim (z wyjątkiem mk 80C51).

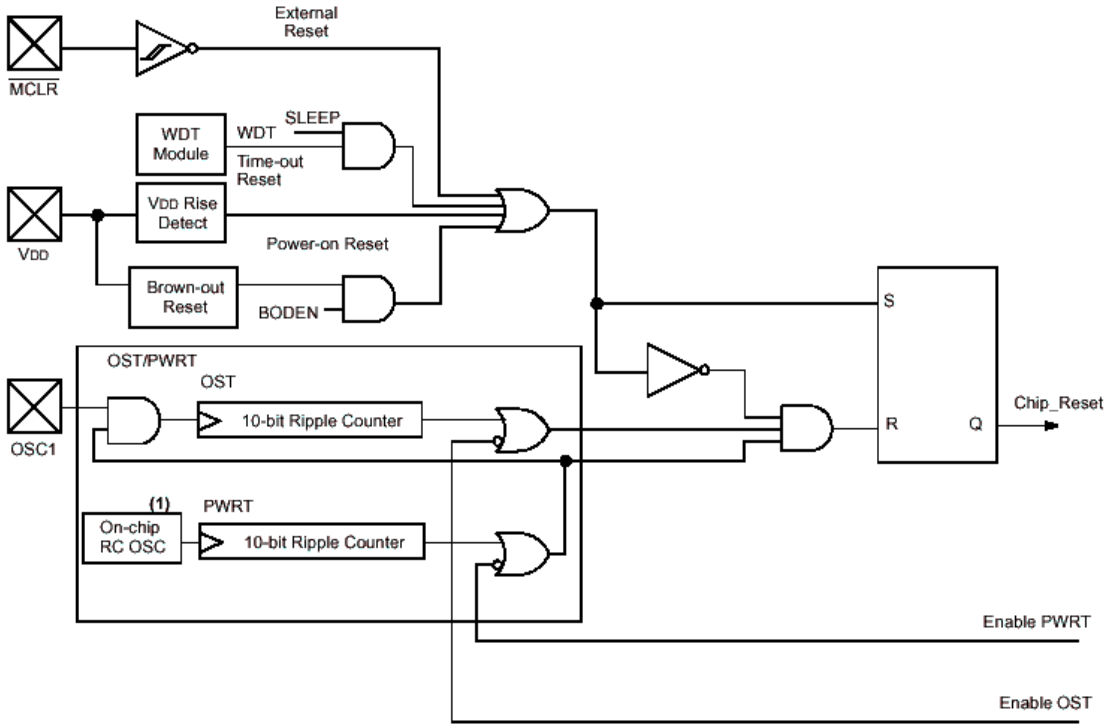
Najprostszy i najczęściej stosowany układ przedstawiono na rys. 2.21. Zapewnia on niski poziom sygnału bezpośrednio po włączeniu zasilania i trwający przez określony czas. Jest to układ RC. Zatem czas trwania resetu zależy od stałej czasowej RC.



Rys. 2.21. Prosty układ resetu

Obecnie mk posiadają rozbudowane układy resetu, które poza resetem mk również monitorują poprawność pracy systemu i reagują na sytuacje awaryjne.

Na rys. 2.22 pokazano taki układ.



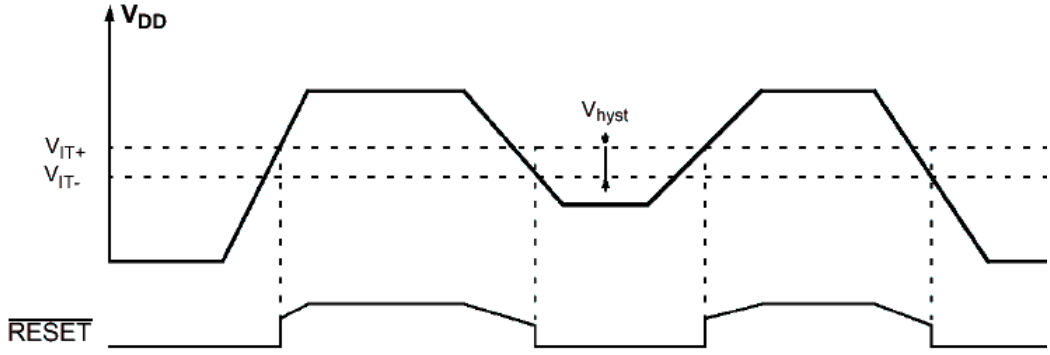
Rys. 2.22. Układ resetu mk PIC16F873

Reset po włączeniu zasilania jest generowany przez specjalny blok (V_{DD} Rise Detect), w przypadku wykrycia przez niego narastającego zbocza sygnału zasilającego. Dzięki temu układ z rys. 2.21 jest zbędny.

Sygnał resetu może być również wywołany przez bloki nadzorujące poprawność pracy mk takie jak BOR, czy LVD.

BOR sprawdza napięcie zasilające. Jeżeli spadnie ono poniżej V_{BOR} i jego spadek trwać będzie przynajmniej przez T_{BOR} to układ wysyła sygnał resetu.

Działanie układu LVD jest bardziej złożone. Można powiedzieć, iż spełnia on funkcje detektora narastania napięcia zasilania jak i układu BOR. Na rys. 2.23 pokazano na przebiegach działanie takiego układu.

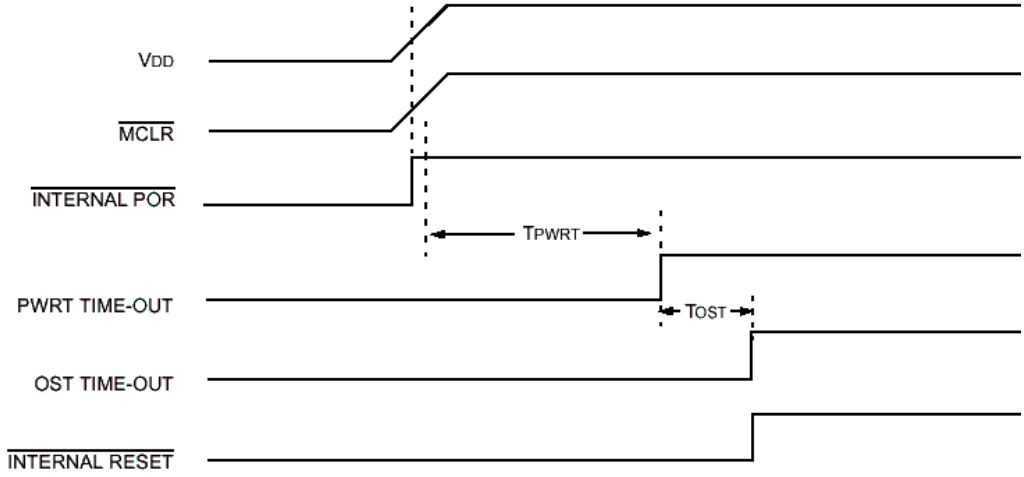


Rys. 2.23. Przebiegi sygnałów V_{DD} i RESET dla układu LVD mk ST72215G

Układ generuje sygnał resetu dopóki V_{DD} podczas narastania nie osiągnie założonego napięcia V_{IT+} oraz gdy to napięcie spadnie poniżej V_{IT-} podczas opadania.

Kolejnym źródłem resetu jest układ watchdog. Generuje on sygnał resetu, gdy nastąpi zawieszenie programu w mk.

Układy resetu posiadają również dodatkowe układy wydłużające sygnał reset (rys. 2.22). Ma to zapewnić pracę mk z już ustalonym napięciem zasilania i ustalonym sygnałem z oscylatora. W PIC16F873 licznik PWRT, z własnym wewnętrznym oscylatorem RC, wydłuża sygnał resetu o T_{PWRT}=72ms, a licznik OST o dodatkowe 1024 cykle sygnału z oscylatora kwarcowego (T_{OSC}), oczywiście o ile są włączone.



Rys. 2.24. Przebiegi sygnałów w układzie resetu mk PIC16F873

Dla mk ST72215G wydłużenie tego sygnału jest stałe i wynosi 4096 cykli sygnału zegarowego.

Do konfiguracji układów resetu są przeznaczone specjalne rejestry. Znajduje się w nich również informacja o przyczynie resetu, np. czy reset nastąpił sygnałem zewnętrznym, czy po włączeniu zasilania, czy też został wywołany przez układ watchdog.

Przy pomocy sygnału reset każdy mk można wprowadzić w **tryb testowy**. Służy on do testowania końcówek mk na etapie produkcyjnym. Podczas pracy mk w systemie należy zapewnić taką konfigurację sygnałów aby nigdy nie wszedł on w stan testowania (służy on tylko do testowania w fazie produkcji).

2.4.4. Układy nadzorujące – układ watchdog

Do układów nadzorujących poprawność pracy mk można zaliczyć:

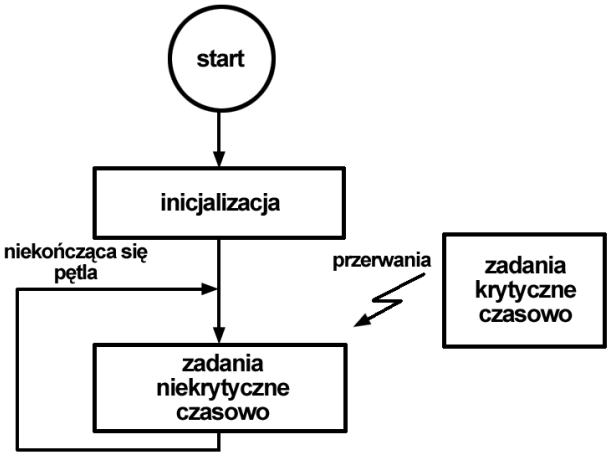
- autonomiczne liczniki **watchdog** (licznik nadzorcy),
- **monitory sygnału zegarowego**,
- **układy detekcji zaniku i powrotu napięć zasilających**,
- **układy detekcji przyczyn awarii systemowych**.

Część z tych układów została już przedstawiona przy omawianiu układu oscylatora i dystrybucji sygnału zegarowego oraz układu resetu.

Najczęściej stosowanym układem nadzorującym pracę mk jest licznik **watchdog**.

Poprawnie pracujący mk charakteryzuje się pewną sygnaturą częstotliwościową lub czasową zawierającą się w ściśle określonych granicach. Wynika to z zasady pisania oprogramowania na mk (rys 2.25). Program użytkownika, po części inicjalizacyjnej, jest wykonywany w niekończącej się pętli od czasu do czasu przerywanej przez obsługę przerwań. Program ten jest jedynym programem jaki znajduje się w mk – w przeciwieństwie do komputerów PC, gdzie program użytkownika jest uruchamiany przez pracujący na okrągło system operacyjny. W mk nie ma systemu operacyjnego. Zatem mk wykonuje tylko to co napiszemy. Stąd aby mk działał prawidłowo nasz program musi pracować w nieskończonej

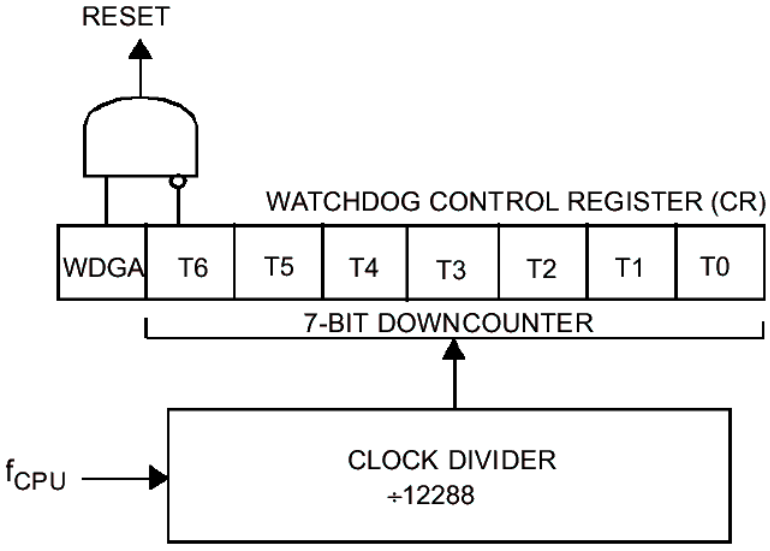
pętli. Pętla ta jest wykonywana z określoną częstotliwością możliwą do obliczenia przez programistę (znamy częstotliwość oscylatora, z dokumentacji wiemy ile cykli zegarowych potrzebuje każda instrukcja i oczywiście wiemy ile jest tych instrukcji w pętli).



Rys. 2.25. Struktura programu użytkownika w mk

Jeżeli w jakimś momencie pętla ta zawiesi się (np. błąd w programie) lub będzie wykonywana zdecydowanie wolniej niż powinna (np. nieprawidłowa praca urządzeń peryferyjnych) jest to równoważne z wykryciem sytuacji alarmowej. Układem, który to wykrywa jest **autonomiczny licznik watchdog**. Gdy wykryje on tę sytuację wysyła sygnał reset zerujący mk oraz ustawia odpowiednie bity w rejestrze przechowującym informację o przyczynach wyzerowania procesora.

Zasada jego działania jest następująca. Układ ten jest licznikiem zasilanym sygnałem z własnego oscylatora RC (np. dla PIC16F873 i AT90S8515) lub oscylatora kwarcowego (rys. 2.26). Jeśli nastąpi jego przepełnienie to wysyła on sygnał reset. Czyli trzeba ten licznik co jakiś czas zerować. Służą do tego specjalne instrukcje np. CLRWDT dla PIC16F873, które należy umieścić w nieskończonej pętli programu (lub inaczej – głównej pętli programu). Czyli jak program pracuje prawidłowo to rozkaz zerujący watchdog jest wykonywany cyklicznie z założoną częstotliwością. Natomiast gdy program się zawiesi to instrukcja ta nie zostanie wykonana na czas i licznik watchdoga przepełni się wywołując reset mk.



Rys. 2.26. Układ watchdog mk ST72215G

W przypadku mk ST72215G należy cyklicznie wpisywać do rejestru CR watchdoga bajt o wartości od FFh do C0h w zależności od tego ile zliczeń 12288 impulsów zegara założyliśmy. Licznik zlicza w dół (dekrementuje). Bit WDGA służy do włączenia/wyłączenia watchdoga. Natomiast wyzerowanie bitu T6 generuje sygnał resetu. Bit ten można również programowo wyzerować wywołując w ten sposób reset programowy.

Po uruchomieniu (inicjalizacji) mk, analiza śladu programu (np. stosu, zawartości rejestrów, itp.) oraz znajomość przyczyny zerowania daje podstawy do tworzenia programów auto-korekcyjnych, które po wykryciu błędu oprogramowania sygnalizują i zapamiętują numer błędu i mogą dokonywać blokady odpowiedniego fragmentu kodu zabezpieczając system przed ciągłym blokowaniem się w sytuacjach alarmowych.

2.4.5. System przerwań

Współpraca jc z urządzeniami peryferyjnymi i zewnętrznymi jest realizowana na dwa sposoby:

- przez **programowe testowanie (odpytywanie, przeglądanie)** stanu urządzeń (*polling*),
- w **systemie przerwań**.

Metoda z **przeglądaniem programowym** polega na okresowym sprawdzaniu zawartości rejestru kontrolno-sterującego lub odpowiednich bitów (flag przerwań) związanych z każdym urządzeniem peryferyjnym i na podstawie tego określenie, czy urządzenie wymaga obsługi, czy też nie. **Obsługa priorytetowa** (kolejność obsługi) jest tu realizowana przez ustalenie odpowiedniej kolejności testowania urządzeń. Urządzenie o największym priorytecie jest testowane w każdym cyklu sprawdzania jako pierwsze, natomiast urządzenie o najmniejszym priorytecie jako ostatnie. Wadą takiego rozwiązania jest długi czas oczekiwania urządzeń na obsługę, ponieważ jc musi w każdym cyklu sprawdzić wszystkie urządzenia, nawet jeśli nie wymagają one obsługi.

Znacznie bardziej efektywny jest **system (układ) przerwań**. Umożliwia on **natychmiastowe, czasowe** wstrzymanie aktualnie wykonywanej sekwencji programu użytkownika (rys. 2.25) i przejście do innej sekwencji programu związanej z obsługą określonego zdarzenia (przerwania), zaistniałego w urządzeniach peryferyjnych.

Generalnie, schemat obsługi przerwania jest następujący:

- urządzenie peryferyjne wykrywa wystąpienie określonego zdarzenia wymagającego reakcji mk,
- urządzenie peryferyjne zgłasza jc, za pośrednictwem układu przerwań, żądanie przerwania,
- jc przerywa aktualnie wykonywaną sekwencję programu i zachowuje w specjalnym obszarze pamięci – **stosie** (*stack*), wszystkie dane związane z aktualnie wykonywanym programem,
- jc przechodzi do wykonywania programu napisanego specjalnie przez użytkownika do obsługi danego zdarzenia,
- po wykonaniu tego programu jc odtwarza ze stosu dane związane z przerwanym programem i kontynuuje wykonanie przerwanego programu.

Możemy wyróżnić następujące źródła przerwań:

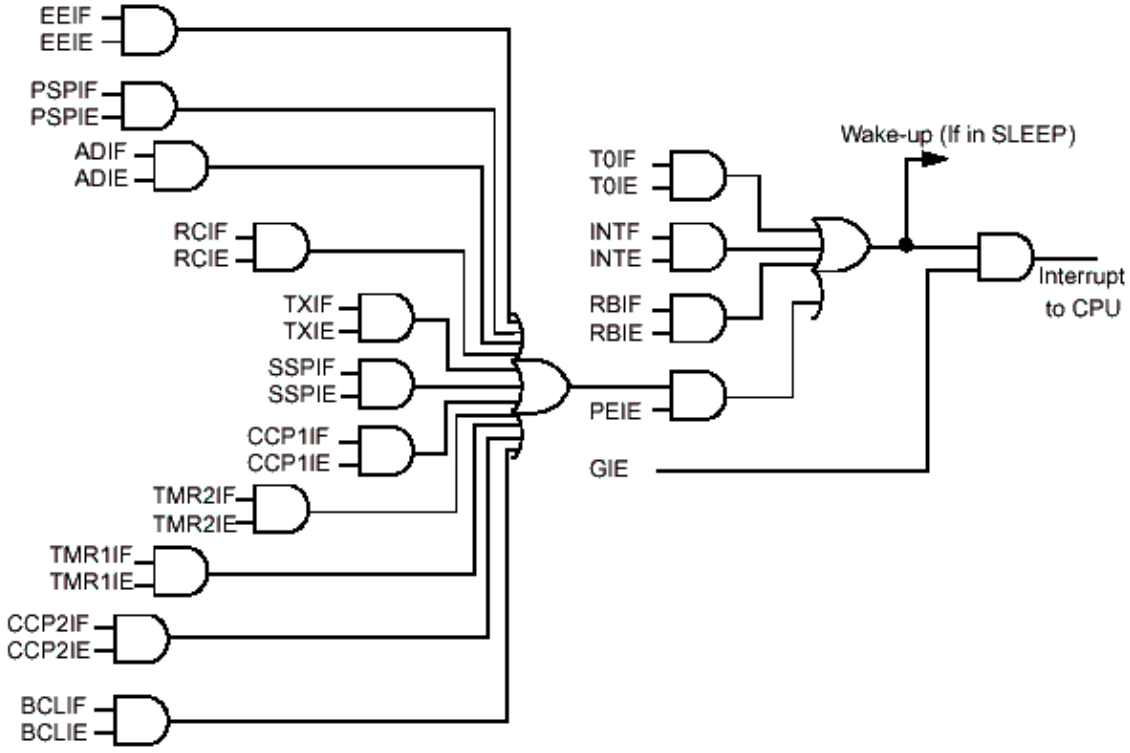
- przerwanie **zewnętrzne** (wywołane sygnałem o odpowiednim zboczu: narastającym lub opadającym lub poziomiem (najczęściej niskim) na dedykowanym pinie),

- przerwanie od **urządzeń wewnętrznych** (np. portów, liczników, komparatorów, przetworników A/C),
- przerwanie **programowe** (ustawienie odpowiedniego bitu, wykonanie rozkazu np. TRAP w mk ST72215G lub INTR w mk COP880).

Ponadto, przerwania mogą być **maskowalne**, tzn. można je zablokować zerując przypisane im bity maskujące lub **niemaskowalne** (najczęściej jest nim przerwanie programowe).

Można wyróżnić następujące systemy przerwań:
- system przerwań z **programowym przeglądaniem urządzeń**,
- system przerwań z **automatyczną detekcją źródła przerwania** – system przerwań **wektoryzowany**.

Przykładowy system przerwań z **programowym przeglądaniem urządzeń** pokazano na rys. 2.27.



Rys. 2.27. Schemat układu przerwań mk PIC16F873

Każde urządzenie wewnętrzne jest wyposażone w przerzutnik służący do zapamiętania stanu urządzenia – zawiera flagę przerwania (np. INTF – flaga przerwania zewnętrznego). Żądanie przerwania powoduje ustawienie przerzutnika, czyli tej flagi. Każdemu przerwaniu może być przypisana maska (np. INTE). Gdy jest ona ustawiona i nastąpi zgłoszenie przerwania, ustawienie flagi przerwania, to do jc jest przekazywany sygnał przerwania, o ile globalna maska przerwań (GIE) jest ustawiona. W tym systemie jc nie zna źródła (przyczyny) przerwania. Najczęściej obsługa wszystkich przerwań jest pod jednym wspólnym adresem (np. 04h). Zatem jc musi programowo przejrzeć (przepytać) wszystkie urządzenia tak, jak przy programowym testowaniu stanu urządzeń, z tą różnicą, iż wykonuje ona tę czynność tylko w trakcie obsługi przerwania. Również i tutaj kolejność priorytetu przerwań zależy od przyjętej kolejności odpytywania.

W tym systemie przerwań flagi poszczególnych przerwań **nie są kasowane sprzętowo** przy wejściu w obsługę przerwań, zatem należy je kasować programowo w trakcie ich obsługi.

Zaletą tego systemu przerwań jest prostota struktury sprzętowej potrzebnej do jego realizacji. Jego główną wadą jest długi czas potrzebny na identyfikację źródła przerwania.

Najbardziej zaawansowanym i często stosowanym w mk **jest system przerwań wektoryzowanych**. W systemie tym na sygnał potwierdzenia przyjęcia przerwania przez jc urządzenie, które zgłosiło przerwanie, podaje na szynę danych kod identyfikacyjny, który jest traktowany jako numer elementu w wektorze przerwań. Zatem każdemu przerwaniu przypisany jest adres obsługi przerwania w pamięci programu (np. tabela 2.3).

Tabela 2.3. Zestawienie wektorów przerwań dla mk AT89C51

Lp.	**źródło przerwania**	**priorytet**	**adres obsługi**
1.	IE0	najwyższy	03H
2.	TF0		0BH
3.	IE1		13H
4.	TF1		1BH
5.	RI + TI		23H
6.	TF2 + EXF2 (tylko AT89C52)	najniższy	2BH

W tym systemie każdemu przerwaniu jest **przypisany na stałe priorytet**. Jeżeli pojawi się w tym samym czasie kilka przerwań to najpierw obsługiwane jest to o najwyższym priorytecie, a następnie według ważności priorytetów kolejne przerwania. Często przerwanie o wyższym priorytecie może przerwać obsługę przerwania o niższym (nigdy nie odwrotnie).

Istnieje również możliwość zmiany kolejności priorytetów. Służy do tego celu rejestr poziomu priorytetów, np. rejestr IP dla mk AT89C51:

IP	X	X	PT2	PS	PT1	PX1	PT0	PX0	adres 0B8H

Gdzie:

X - zarezerwowane

PTn - licznik n (n=0,1,2)

PS - port szeregowy

PXn - przerwanie zewnętrzne z wejścia INTn (n=0,1)

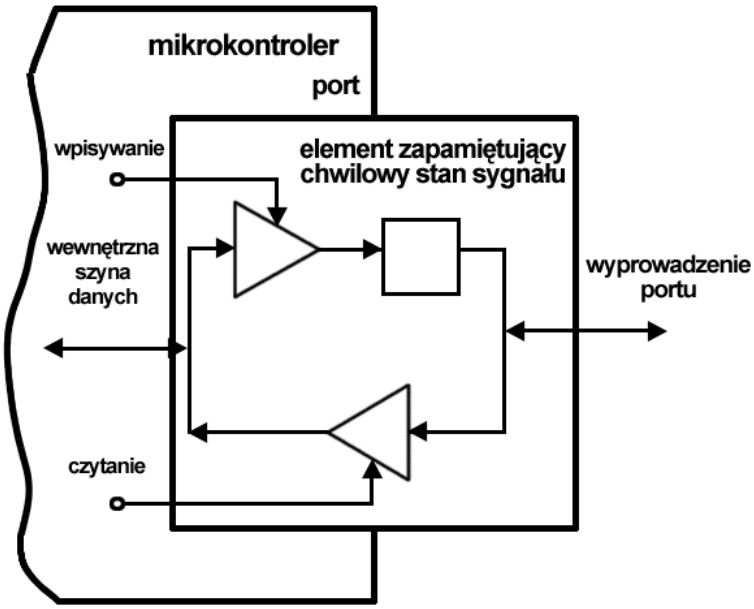
Ustawienie w rejestrze IP znacznika dla danego źródła przerwania powoduje, że przerwanie to osiąga wyższy priorytet od przerwań, dla których znaczniki mają stan 0. Natomiast wzajemna relacja pomiędzy źródłami przerwań, których znaczniki mają ten sam stan, nie ulega zmianie.

Zatem do obsługi przerwań przeważnie używane są trzy rejestry: rejestr z flagami przerwań, z maskami poszczególnych przerwań i maską globalną oraz rejestr poziomu priorytetów.

W tym systemie flagi przerwań są **zerowane sprzętowo** przez obsługę przerwań.

2.5. Porty równoległe mk – warstwa multiplekserów i zacisków we/wy

Do równoległej transmisji danych cyfrowych służą w mk zespoły linii we/wy zwane portami równoległymi (rys. 2.28) (*parallel ports* lub *I/O ports*). W przeważającej liczbie przypadków porty zawierają po 8 linii, mogą zatem transmitować dane bajtami. Liczba portów i ich wyprowadzeń (pinów) jest ograniczona przez liczbę wyprowadzeń obudowy mk. Są one podstawowymi i najważniejszymi urządzeniami peryferyjnymi (ponieważ zawierają w sobie warstwę multiplekserów i zacisków we/wy są omawiane w tym podrozdziale, a nie razem z urządzeniami peryferyjnymi). Wszystkie pozostałe urządzenia peryferyjne wykorzystują funkcje portów.



Rys. 2.28. Schematyczna budowa portu

Za transport informacji wewnątrz mk odpowiedzialna jest wewnętrzna szyna danych. Zatem czytanie danych przez port polega na doprowadzeniu chwilowych stanów wyprowadzeń portu (tzn. istniejących w momencie operacji czytania) do wewnętrznej szyny danych układu. Natomiast operacja wpisania do portu powoduje, że chwilowy stan wewnętrznej szyny danych zostaje zapamiętany na wyprowadzeniach portu. Stan wyprowadzeń portu pozostaje niezmienny, dopóki nie nastąpi kolejna operacja wpisania do portu.

Możemy wyróżnić następujące **typy portów**:

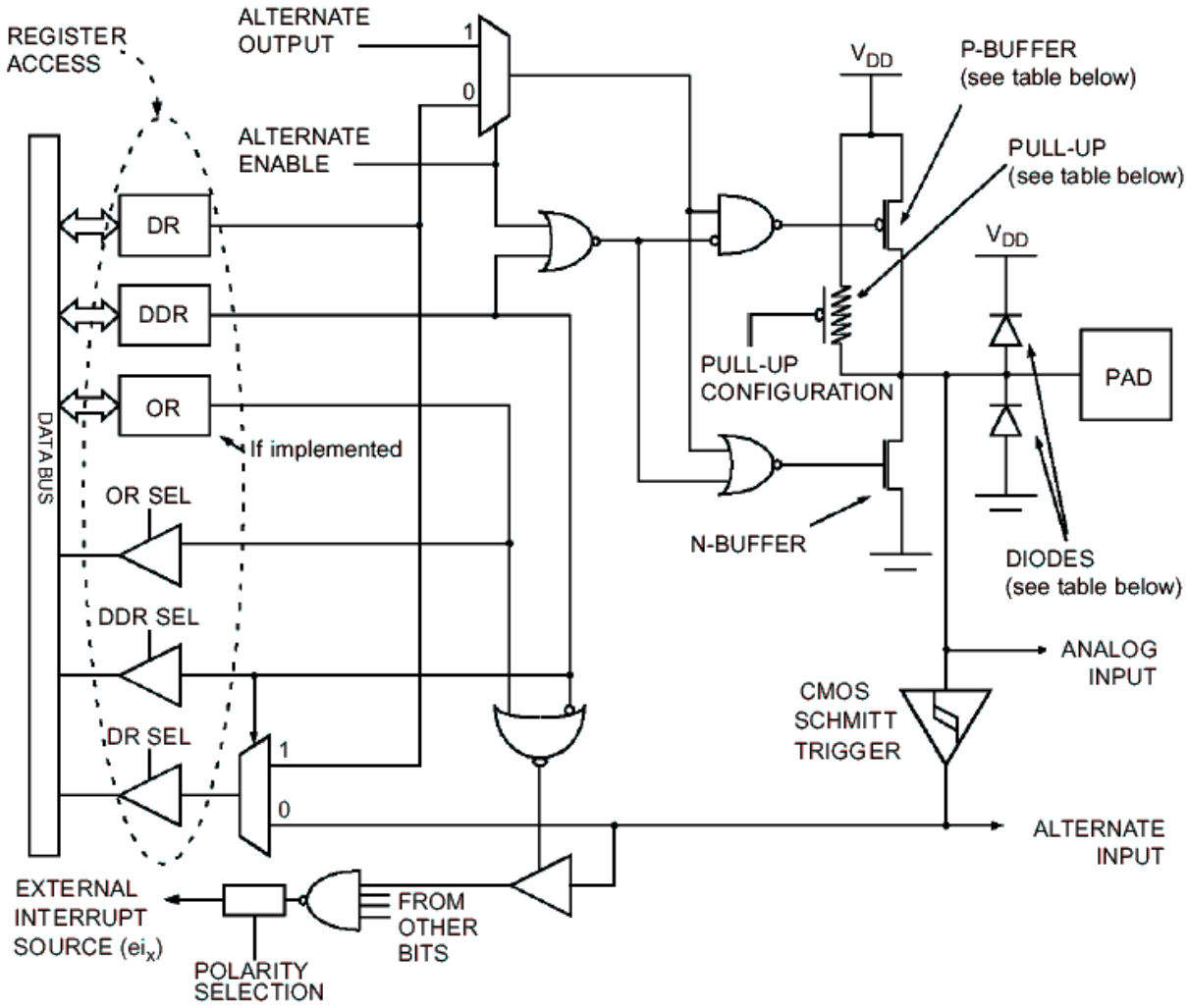
- porty dwukierunkowe,
- porty jednokierunkowe wejściowe,
- porty jednokierunkowe wyjściowe.

Wyróżnia się trzy **rodzaje linii portów (zacisków we/wy)**:

- dwukierunkowe (*quasi-bidirectional*),
- z otwartym obwodem drenu tranzystora (*open dren*),
- o zwiększonej obciążalności (*push-pull output*).

Jak wspomniano, do portów równoległych można podłączyć wewnętrzne urządzenia peryferyjne. Jest to możliwe, dzięki zawartych w nich multiplekserom (stanowią one warstwę multiplekserów, jak pokazano na rys. 2.13).

Zadaniem tej warstwy jest zwiększenie ilości funkcji oraz elastyczności mk bez zwiększania liczby końcówek obudowy. Warstwa ta jest programowalna i pozwala na dynamiczną zmianę konfiguracji końcówek we/wy mk w czasie pracy. Czyli raz końcówka może być we/wy portu równoległego lub wejściem (np. przetwornika A/C, licznika) lub wyjściem urządzenia wewnętrznego (generatora). Czyli zacisk maksymalnie może przyjmować 4 wyżej wymienione funkcje.



Configuration Mode		Pull-Up	P-Buffer	Diodes	
				to V_{DD}	to V_{SS}
Input	Floating with/without Interrupt	Off	Off	On	On
	Pull-up with/without Interrupt	On			
Output	Push-pull	Off	On		
	Open Drain (logic level)		Off		
	True Open Drain	NI	NI	NI (see note)	

Legend: NI - not implemented
Off - implemented not activated
On - implemented and activated

Note: The diode to V_{DD} is not implemented in the true open drain pads. A local protection between the pad and V_{SS} is implemented to protect the device against positive stress.

Rys. 2.29. Pełny schemat blokowy portu we/wy mk ST72215G

Zazwyczaj z dwukierunkowym portem we/wy związane są trzy rejestry:

- rejestr danych (DR dla mk ST72215G),
- rejestr kierunku (DDR),

- rejestr opcji (OR).

Na rys. 2.29 ALTERNATE OUTPUT/INPUT jest wy/we urządzenia peryferyjnego podłączanego do portu równoległego. O tym czy dana końcówka jest dedykowana urządzeniu decyduje sygnał ALTERNATE ENABLE. Wejście ANALOG INPUT jest wykorzystywane przez przetwornik A/C.

W tabeli 2.4 przedstawiono sposób konfiguracji pinów poszczególnych portów mk ST72215G. Jak widać, przy pomocy rejestru DDR ustala się kierunek portu, natomiast rejestr OR służy do ustawiania parametrów elektrycznych pinów.

Tabela 2.4. Konfiguracja portów we/wy mk ST72215G

Port	Pin name	Input (DDR = 0)		Output (DDR = 1)		
		OR = 0	OR = 1	OR = 0	OR = 1	High-Sink
Port A	PA7	floating	pull-up interrupt	open drain	push-pull	Yes
	PA6	floating	floating interrupt	true open-drain		
	PA5	floating	pull-up interrupt	open drain	push-pull	
	PA4	floating	floating interrupt	true open-drain		
	PA3:0	floating	pull-up interrupt	open drain	push-pull	
Port B	PB7:0	floating	pull-up interrupt	open drain	push-pull	No
Port C	PC7:0	floating	pull-up interrupt	open drain	push-pull	

Porty równoległe mogą być również inaczej konfigurowane. Zależy to od typu konkretnego mk. Np. dla mk COP880 mamy rejestr pinów (służy on wyłącznie do czytania poziomów logicznych na pinach portu), rejestr danych (wyłącznie do wystawiania danych na liniach portu) i rejestr kierunku. Mk 80C51 posiada tylko jeden rejestr stowarzyszony z portem równoległym. Zapis do niego ustawia poziomy na liniach portu, a odczyt czyta stan linii portu.

Ważna uwaga: każda linia układu cyfrowego będąca wyjściem, zatem i mk, jest w stanie przejąć znacznie większy prąd w stanie niskim niż wysokim. Wynika to z faktu, iż moc wydzielająca się na końcówce jest równa iloczynowi prądu płynącego przez nią i napięcia na niej. Dlatego tak projektuje się mse, aby sygnały w stanie niskim były sygnałami uaktywniającymi urządzenia pobierające duży (w sensie mse) prąd, np. aby zaświecić diody bezpośrednio podłączone do linii portów mk na tych liniach musi być stan niski.

Dla ST72215G pojedynczy pin może przejąć prąd 25mA, a pin typu Hight-Sink 50mA.

2.6. Wewnętrzne urządzenia peryferyjne mk

Podstawową cechą mk jest zawarcie w ich strukturze dodatkowych urządzeń, zwanych **urządzeniami peryferyjnymi**. Stanowią one warstwę programowalnych urządzeń we/wy w modelu warstwowym mk zamkniętego (rys. 2.13).

Warstwa ta charakteryzuje typ danego mk. Ilość urządzeń oraz ich typ decyduje o profilu zastosowań danego układu. Warstwę tę należy postrzegać jako rodzaj biblioteki funkcji sprzętowych dostępnych w danym typie mk. Wszystkie układy są **programowalne** i charakteryzują się **dużym stopniem autonomii w stosunku do procesora rdzeniowego**. Zadania przekazane im do wykonania odpowiednimi rozkazami, wykonywane są **samodzielnie**, **bez zaangażowania** czasu procesora. O ich zakończeniu procesor informowany jest wysłaniem **przerwania** lub ustawieniem odpowiedniego **bitu** w rejestrze stanu urządzenia.

Produkowane obecnie mk są wyposażone w bardzo różne zestawy rozbudowanych układów peryferyjnych. Można je podzielić na następujące grupy funkcyjne (zachowano oryginalne nazwy angielskie):

1. **Układy nadzorujące** (właściwie przypisane jc, ale wymienione tutaj):
- watchdog,
- LVD, (POR), RSM, MO, CSS, MCC, BOR.

2. **Układy czasowe**:
- Timers/Counters,
- PWM,
- RTC,
- Captures.

3. **Sterowniki komunikacji szeregowej**:
- UART, IrDA,
- SPI, QSPI,
- 1-Wire,
- I^2C,
- CAN,
- USB,
- MII i Ethernet PHY.

4. **Łącza równoległe**:
- Ports,
- Slave port.

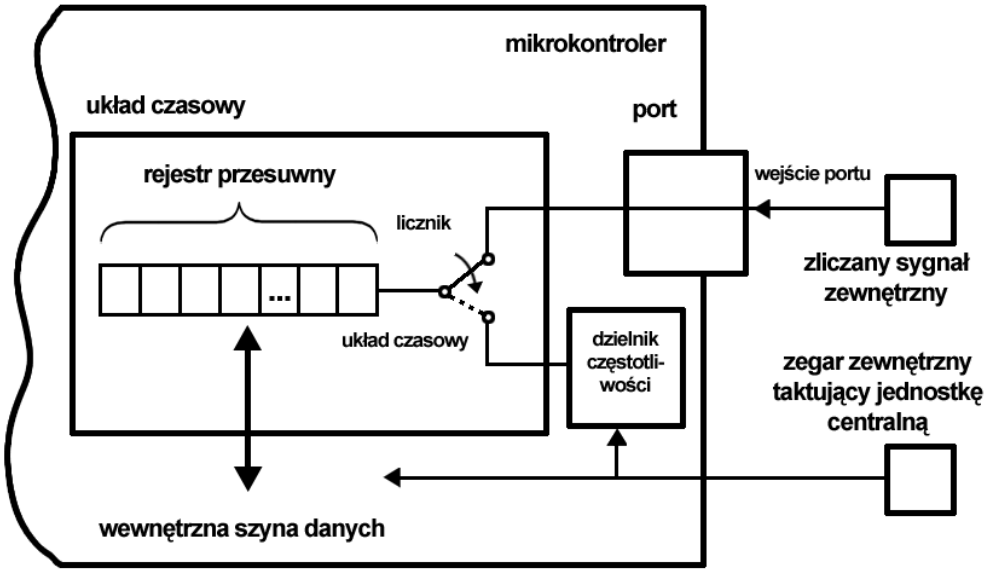
6. **Układy specjalizowane**:
- A/D Comparators,
- Comparators,
- C/A Converters,
- Buzzers,
- EEPROMs (zaliczane również do jc).

Od strony procesora układy we/wy programowane są za pośrednictwem **magistrali wewnętrznej**. Układom tym zazwyczaj przypisane są rejestry robocze (zawierające wyniki ich działania), rejestry konfiguracyjne/sterujące i rejestry statusu umieszczone w obszarze adresowania pamięci danych. Użycie funkcji realizowanej przez układy we/wy wymaga w większości przypadków połączenia go ze światem zewnętrznym przy pomocy warstwy multiplekserów i zacisków we/wy.

2.6.1. Układy licznikowe/czasowe

Współpraca mk z otoczeniem w czasie rzeczywistym wymaga odliczania czasu lub generowania złożonych sekwencji binarnych. Jest to realizowane przez specjalizowane bloki nazywane **licznikami** (*counters*) lub **układami czasowymi** (*timers*) z dokładnością oscylatora kwarcowego mk. Liczba liczników stosowanych w mk i ich długość wyrażona w bitach, różnią się dla konkretnych typów mk. Większość mk ma przynajmniej dwa liczniki 16-bitowe, każdy mk posiada choćby jeden licznik 8-bitowy.

Najprostszą strukturę takiego układu pokazano na rys. 2.30. Składa się on z **rejestru** o określonej długości i niezaznaczonych na rys. rejestru zawierającego np. bity przepełnienia, ustawiające tryb pracy, bit startu zliczania licznika itd. Rejestr przesuwny zliczający liczbę zmian poziomów doprowadzonych do niego sygnałów jest dostępny z poziomu programu użytkownika, jako rejestr (dla liczników 8-bitowych) lub para rejestrów (dla liczników 16-bitowych) o określonych adresach w obszarze pamięci danych.



Rys. 2.30. Schematyczna budowa układu czasowego

Z rys. 2.30 widać, że najprostsze układy czasowe mogą pracować w dwóch podstawowych konfiguracjach:

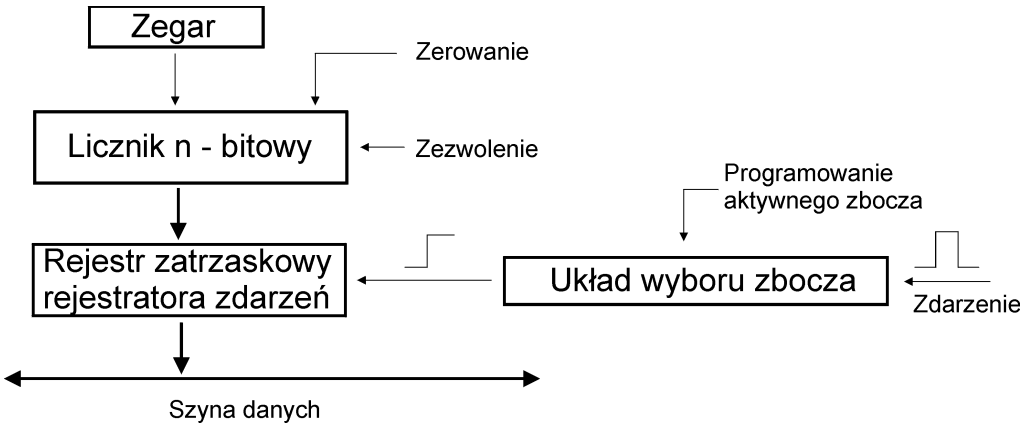
- jako właściwe **układy czasowe** (*timers*) – Są wtedy taktowane wewnętrznym sygnałem zegarowym przeznaczonym do taktowania jc. Timery wykorzystywane są w programie użytkownika jako wzorce czasu. W celu generowania wzorców czasu o różnej długości wewnętrzny sygnał zegarowy, przed doprowadzeniem do układu czasowego, przechodzi przez programowalny dzielnik częstotliwości.
- jako **liczniki** (*counters*) – Są one wtedy taktowane zewnętrznymi sygnałami doprowadzanymi poprzez linie wejściowe portów i wykorzystywane w programie użytkownika np. jako liczniki zmian poziomów sygnałów zewnętrznych

Kolejna właściwość liczników to sposób zliczania impulsów. Liczniki mogą zliczać impulsy w górę, tzn. **inkrementować** lub w dół czyli **dekrementować**.

W praktyce mk są wyposażone w bardziej rozbudowane układy licznikowe. Liczniki te są używane zazwyczaj do realizacji następujących funkcji:

- określenia (mierzenia) odstępów czasu między zdarzeniami zachodzącymi na zewnątrz mk, sygnalizowanymi przez impulsy elektryczne doprowadzone do mk. Funkcja ta bywa nazywana **rejestracją zdarzeń** (*input event capture*),
- **generowania impulsów** (sekwencji impulsów) w odstępach czasu o zaprogramowanej wartości (*output compare*) lub **przebiegu okresowego** o zadanej częstotliwości,
- **generowania sygnałów impulsowych** o określonym czasie trwania lub sygnałów okresowych o zadanym współczynniku wypełnienia (**PWM** – *puls width modulation*),
- sterowania szybkością transmisji w portach szeregowych, zarówno w trybie synchronicznym, jak i asynchronicznym (*baut rate generator*),
- realizacji zadań licznika nadzorcy – watchdoga, omówionych w poprzednim podrozdziale.

Podstawową konfigurację licznika jako **rejestratora zdarzeń** pokazano na rys. 2.31.



Rys. 2.31. Licznik w konfiguracji rejestratora zdarzeń

Jego zadaniem jest określenie **czasu wystąpienia zdarzenia zewnętrznego** sygnalizowanego przez impuls elektryczny podany na wejście. Czas ten jest mierzony w sposób względny, tzn. w stosunku do chwili zezwolenia na rozpoczęcie zliczenia w *n*-bitowym liczniku sterowanym impulsami zegara, ewentualnie **preskalera**. Przed uruchomieniem zliczania program zeruje licznik i określa zbocze sygnału zewnętrznego, które ma spowodować rejestrację zdarzenia. Po wystąpieniu tego zbocza **zawartość licznika jest przepisywana do rejestru zatrzaskowego** rejestratora. Układ może równocześnie wygenerować przerwanie informujące jc o zarejestrowaniu zdarzenia, jeśli wcześniej ustawiono zezwolenie przerwania. W tym układzie odczyt rejestru zatrzaskowego musi nastąpić przed wystąpieniem kolejnego zdarzenia, ponieważ pracujący cały czas licznik przy następnym sygnale zewnętrznym zmieni zawartość rejestru zatrzaskowego.

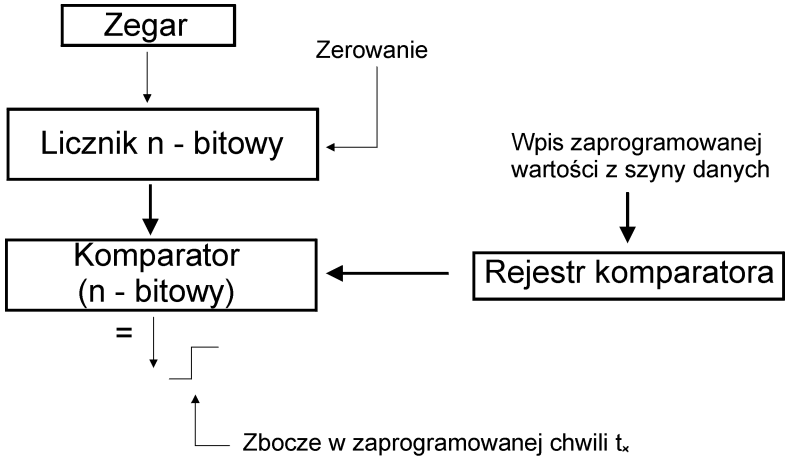
Rejestrując czas wystąpienia kolejnych zboczy można łatwo określić częstotliwość sygnału okresowego lub szerokość impulsów. W pierwszym przypadku rejestruje się dwa następujące po sobie zbocza o tej samej polaryzacji, natomiast w drugim – dwa zbocza o różnej polaryzacji.

Szczególne zastosowanie tej konfiguracji polega na generowaniu impulsów o programowalnym czasie opóźnienia w stosunku do zewnętrznych impulsów odniesienia. W tym przypadku układ pracuje w mieszanym trybie rejestrator – generator. Po zarejestrowaniu zdarzenia licznik odmierza zaprogramowaną liczbę impulsów i generuje sygnał wyjściowy.

W konfiguracji **programowanego generatora impulsów** licznik pracuje zgodnie z zasadą pokazaną na rys. 2.32. Jc wpisuje do rejestru komparatora liczbę określającą chwilę wygenerowania impulsu, po czym uruchamia licznik. Po upływie zaprogramowanej liczby cykli zegara, **komparator wykrywa zrównanie się zawartości licznika i rejestru**, po czym

generuje sygnał wyjściowy. Jednocześnie licznik może wysyłać do jc przerwanie, o ile zostało odblokowane.

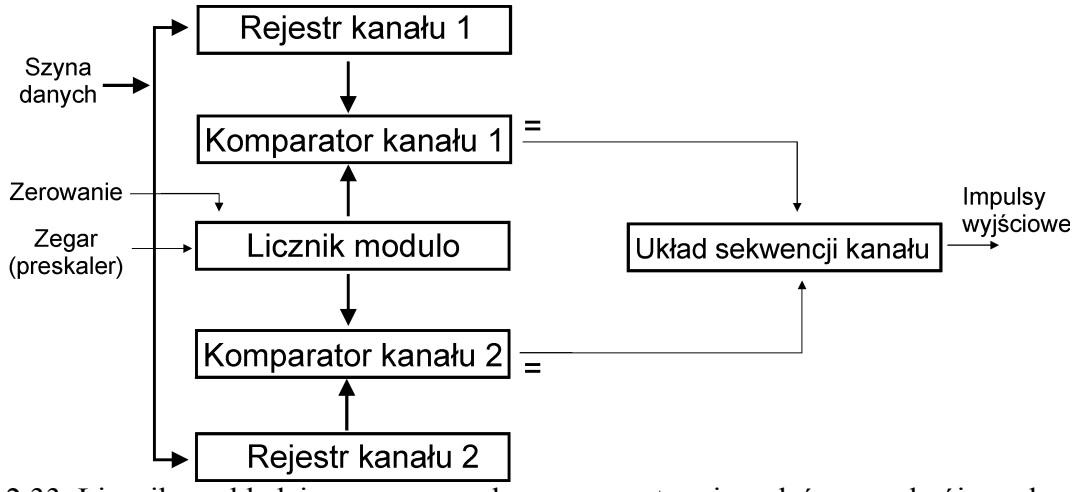
Większość liczników/generatorów wytwarza impulsy o zadanej polaryzacji. Ponadto po wygenerowaniu impulsu przez układ licznika jc może zlecić nowy cykl generacji z innym początkowym ustawieniem rejestru komparatora. W ten sposób mk jest w stanie generować przebieg o dowolnie ustalanych czasach trwania kolejnych impulsów.



Rys. 2.32. Licznik w układzie programowalnego generatora impulsów

Wadą przedstawionej prostej konfiguracji generatora jest to, że przy generacji kolejnych impulsów powstaje błąd wynikający z czasu działania jc (zwykle dwa okresy sygnału zegarowego), ponieważ jc musi po każdej operacji wpisywać nową zawartość do rejestru komparatora.

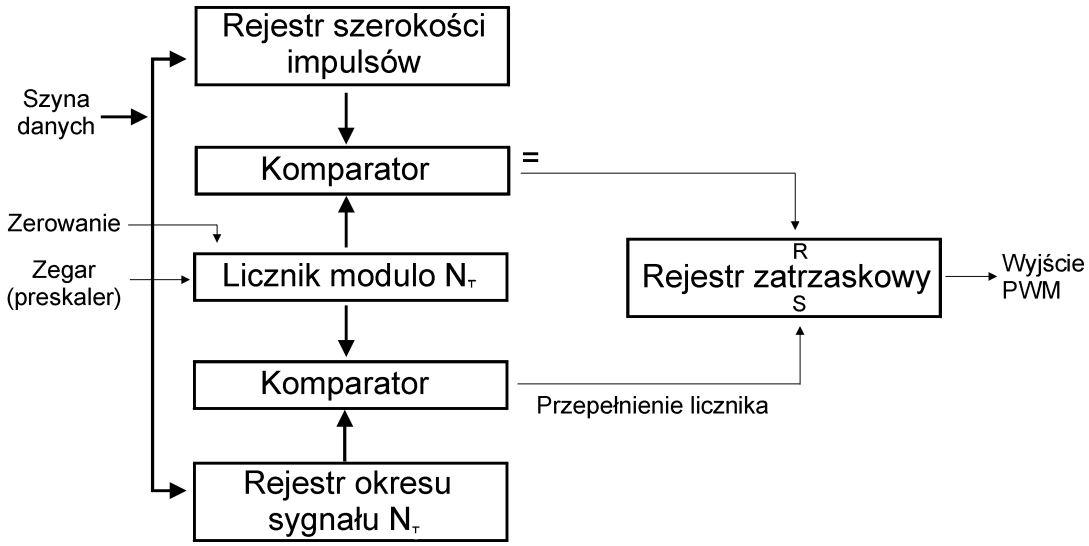
W celu wyeliminowania tej wady stosuje się ulepszone układy liczników z podwójnymi rejestrami i komparatorami, nazywane **licznikami z buforowaniem** (*buffered output compares*). W tych układach aktualnie zaprogramowana wartość opóźnienia jest przechowywana w rejestrze jednego z kanałów, natomiast wartość kolejnego opóźnienia w rejestrze drugiego kanału (rys. 2.33). Blok licznika autonomicznie steruje układem selekcji kanału. W najprostszym przypadku kanały pracują naprzemiennie. Należy pamiętać aby nie wpisywać danych do rejestru aktualnie pracującego kanału.



Rys. 2.33. Licznik w układzie programowalnego generatora impulsów z podwójnym kanałem

W **konfiguracji PWM** licznik pracuje jako generator fali prostokątnej o programowanym współczynniku wypełnienia. Układ ten pokazano na rys. 2.34. Zasada jego pracy jest

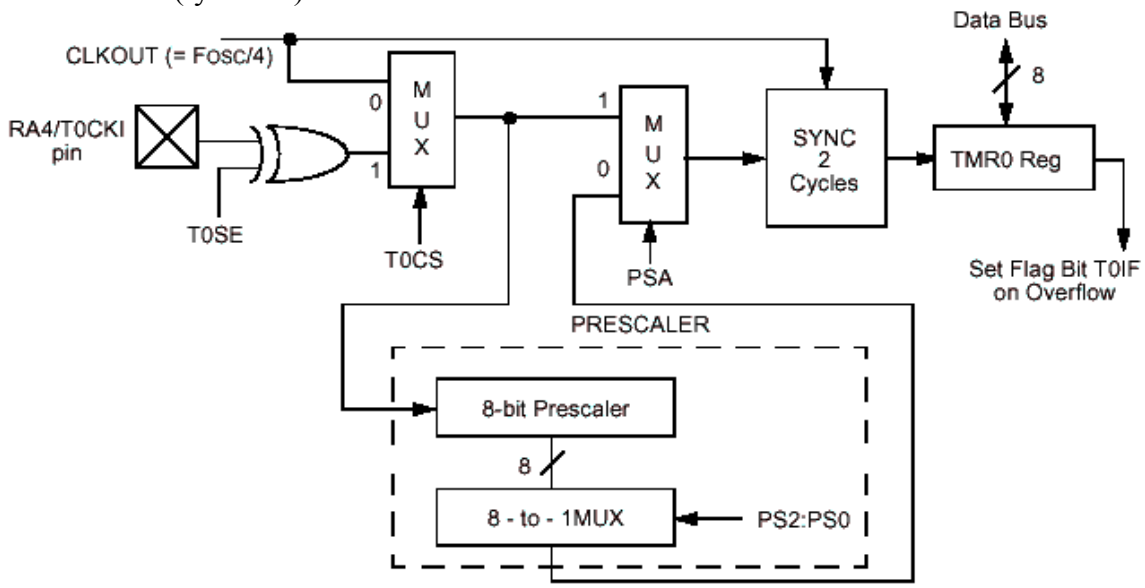
następująca. Jeśli **zawartość licznika osiągnie zaprogramowaną liczbę** N_T określającą okres impulsów, **komparator ustawia przerzutnik wyjściowy**. Sygnał przepełnienia licznika określa zatem początek okresu generowanego przebiegu. W czasie zliczania kolejnych impulsów zegara zawartość licznika jest porównywana z zawartością rejestru szerokości impulsów, a po zrównaniu się z nią generowany jest sygnał „=", który zeruje przerzutnik wyjściowy. Tym samym impuls wytwarzany przez PWM kończ się.



Rys. 2.34. Licznik w trybie PWM

Rozdzielczość wyjściowa układu PWM jest określona przez częstotliwość sygnału zegara i długość słowa licznika. Rejestr sterujący PWM zawiera dodatkowo wydzielone bity, które pozwalają na wyjściu uzyskać sygnał stały na poziomie „0" lub „1".

Jako przykład prostego układu licznikowego można przedstawić 8-bitowy licznik TIMER0 mk PIC16F873 (rys. 2.35).

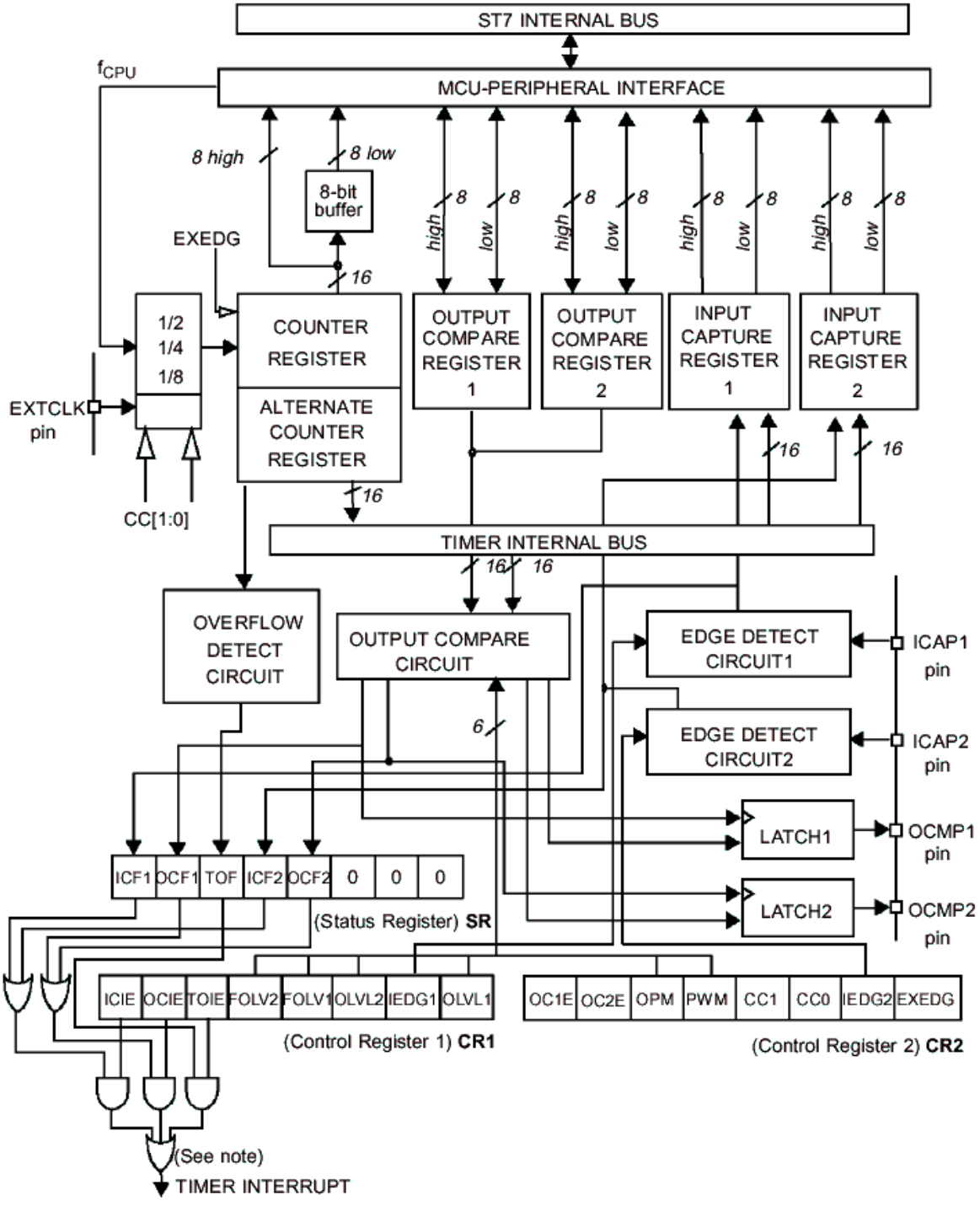


Rys. 2.35. Schemat blokowy licznika TIMER0 mk PIC16F873

Rejestrem przesuwnym (zliczającym) jest rejestr TMR0. Sterowanie licznikiem odbywa się za pomocą bitów zawartych w rejestrze OPRION_REG. Przy pomocy bitu T0SC wybiera się tryb pracy: czy timer „0", czy też licznik „1". Gdy taktujemy układ impulsami zewnętrznymi

bit T0SE pozwala nam wybrać aktywne zbocze: „0” narastające, „1” opadające. Przy pomocy bitu PSA wybieramy drogę sygnału: „0” przez preskaler z podziałem zgodnym z ustawieniem bitów PS2:PS0 lub „1” bezpośrednio, wówczas preskaler podłączony jest do układu watchdog. Następnie sygnał przechodzi przez układ synchronizujący przebieg zewnętrzny z wewnętrznym sygnałem taktującym jednocześnie opóźniając go o dwa cykle zegarowe. Sygnał ten inkrementuje licznik i jak nastąpi jego przepełnienie, tzn. przejście licznika z FFh do 00h to ustawiana jest flaga przepełnienia oraz wywoływane przerwanie, o ile nie zostało zablokowane.

Przykładem rozbudowanego układu licznikowego może być 16-bitowy Timer A (lub Timer B) mk ST72215G, którego schemat blokowy pokazano na rys. 2.36.



Rys. 2.36. Schemat blokowy układu Timer A mk ST72215G

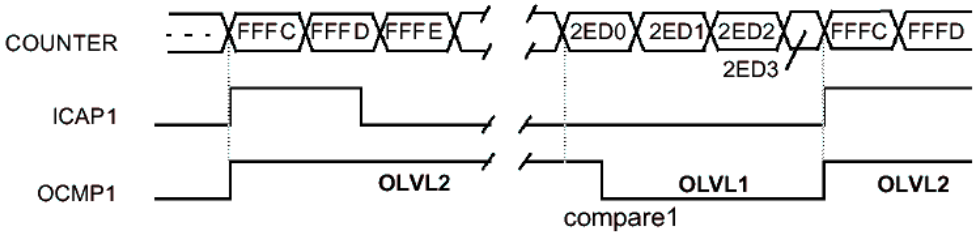
Układ ten składa się z:

- programowalnego preskalera, który zapewnia podział wewnętrznego sygnału zegarowego przez 2, 4 lub 8,
- posiada on flagę przepełnienia umożliwiającą generację przerwania,
- pinu, na który można zadać wejściowy sygnał zegarowy (przynajmniej 4-krotnie wolniejszy od sygnału zegarowego jc) z możliwością wyboru aktywnego zbocza,
- bloku generacji sygnałów wyjściowych (*output compare block*) złożonego z: dwóch rejestrów 16-bitowych, dwóch dedykowanych wyjściowych linii sygnałowych, dedykowanych dwóch flag i maskowalnego przerwania,
- bloku rejestratora zdarzeń (*input capture block*), w którego skład wchodzą: dwa 16-bitowe rejestry, dwie dedykowane wejściowe linie sygnałowe z możliwością wyboru aktywnego zbocza, dedykowane dwie flag i maskowalne przerwania,
- jednego rejestru statusu SR i dwóch rejestrów sterujących CR1 i CR2,
- ponadto układ wykorzystuje pięć linii portów we/wy.

Układ może pracować w następujących trybach:

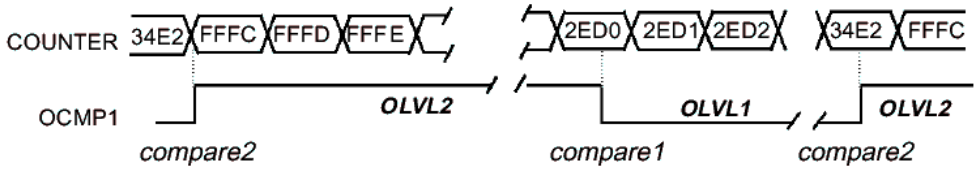
- **16-bitowy counter/timer** sterowany bitami CC0 i CC1 rejestru CR2. Przy pomocy tych bitów wybiera się tryb czasomierza (*timer*) z odpowiednio ustawionym preskalerem lub licznika (*counter*) z możliwością wyboru aktywnego zbocza (bit EXEDG rej. CR2). Do zliczania służą dwie pracujące równolegle (zliczające w górę) para rejestrów CHR i CLR stanowiące 16-bitowy licznik CR i para alternatywnych rejestrów ACHR i ACLR stanowiące 16-bitowy licznik ACR. Jedyna różnica między tymi rejestrami polega na tym, iż odczyt z rej. ACR nie zeruje flagi przepełnienia TOF zawartej w rej. statusu SR. Po resecie zawartość licznika wynosi FFFCh.
- **Tryb rejestratora zdarzeń (Input Capture)**. W tym trybie wykorzystuje się dwa wejścia ICAP1 i ICAP2, na których aktywny impuls powoduje przechwycenie bieżącej wartości pracującego licznika i zatrzaśnięcie jej odpowiednio w rejestrze IC1R, gdy ten impuls pojawi się na pierwszym wejściu lub w IC2R, gdy na drugim wejściu. Jednocześnie ustawiane są odpowiednio bity ICF1 lub ICF2, które pozwalają na wygenerowane przerwania. Rejestry IC1R i IC2R służą wyłącznie do odczytu. Aktywne zbocze sygnału pojawiającego się na wejściach ICAP1 i ICAP2 wybiera się odpowiednio za pomocą bitów: IEDG1 i IEDG2. Rozdzielczość pomiaru czasu wynosi f_{CPU}/CC[1:0].
- **Tryb Output Compare**. Ten tryb może być wykorzystany do kontroli sygnału wyjściowego na pinach OCMP1 i OCMP2 lub do wyznaczania odcinków czasu. Gdy zawartość pracującego licznika zrówna się z zawartością wpisaną do rej. OCR*i* (*i*=1,2), to na odpowiednim pinie wyjściowym jest ustawiana wartość OC*i*E, jeżeli bit OCIE jest ustawiony, ustawiane są flagi w rej. SR: OCF*i* oraz następuje generacja przerwania, o ile je odblokowano. Po resecie w rej. OCR*i* znajduje się wartość 8000h.
- **Tryb One Pulse**. W tym trybie pojawienie się zdarzenia (aktywnego impulsu) na pinie ICAP1 powoduje wygenerowanie pojedynczego impulsu na pinie OCMP1. W ten tryb wchodzi się przez ustawienie bitu OPM w rej. CR2. Korzysta on z funkcji trybów Input Capture 1 i Output Compare 1. Zasada pracy: kiedy aktywny impuls pojawi się na wejściu ICAP1, licznik jest inicjalizowany do FFFCh (i zaczyna zliczać wewnętrzne impulsy) oraz bit OLVL2 jest wystawiany na pin OCMP1, bit ICF1 jest ustawiany (można wywołać przerwanie), jak i wartość FFFDh jest wprowadzana do rej. ICR1. Następnie, kiedy zawartość licznika zrówna się z ustawioną przez nas zawartością rej. OC1R to bit OLVL1 jest wyprowadzony na pin OCMP1 (rys. 2.37).
- **Tryb Pulse With Modulation** umożliwia generację sygnału na wyjściu OCMP1 o częstotliwości i czasie trwania zależnym od zawartości rejestrów OC1R i OC2R. Tryb ten korzysta z pełnej funkcji trybu Output Compare 1 i rej. OC2R. Jeżeli bity OLVL1=1 i OLVL2=0, czas trwania impulsu jest równy różnicy pomiędzy zawartością rej. OC2R, a

rej. OC1R. Gdy nastąpi zrównanie zawartości licznika z rej. OC2R, to licznik przyjmuje wartość FFFCh (rys. 2.38).



IEDG1=1, OC1R=2ED0h, OLVL1=0, OLVL2=1

Rys. 2.37. Przebiegi czasowe dla trybu One Pulse



OC1R=2ED0h, OC2R=34E2, OLVL1=0, OLVL2= 1

Rys. 2.38. Przebiegi czasowe dla trybu Pulse With Modulation

2.6.2. Przetworniki analogowo-cyfrowe (A/C)

Przetworniki A/C stosowane w mk wykorzystują najczęściej dwie metody przetwarzania napięcia na odpowiadającą mu miarę liczbową:

- metodę **sukcesywnej aproksymacji** SAR (*successive approximation*),
- metodę **jednozboczowego ładowania pojemności** SS (*single-slope A/D*), która wymaga dołączenia zewnętrznego kondensatora.

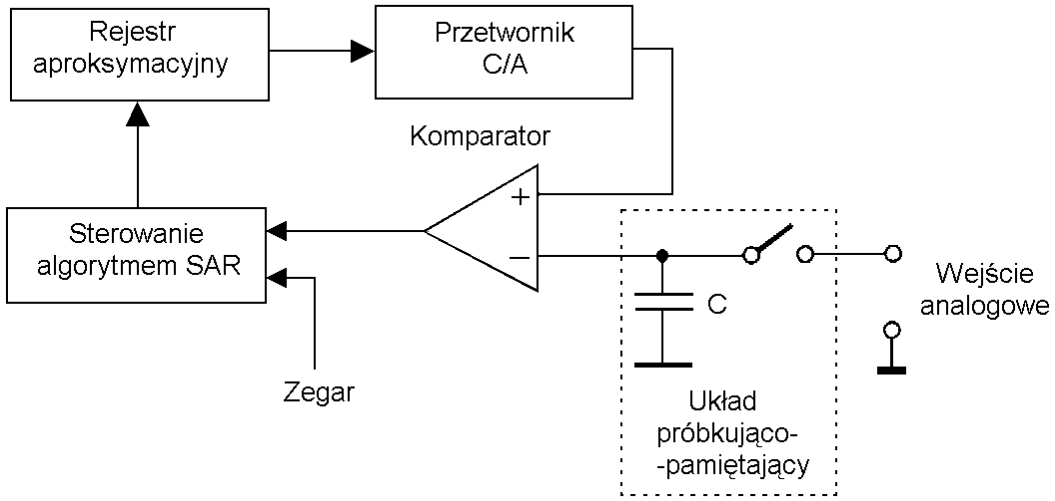
Częściej stosowana jest metoda SAR, która nie wymaga dołączenia do mk zewnętrznych elementów dyskretnych i jest łatwiejsza przy oprogramowaniu przetwornika. Zaletą tej metody jest również krótki czas konwersji, równy zwykle od 10 do 16 cykli zegara w przypadku przetworników 8-bitwych. W praktyce tego typu przetworniki A/C stosowane w mk nie przekraczają rozdzielczości 12 bitów. Wynika to z faktu, iż umieszczone w jednej strukturze z rozbudowanym układem cyfrowym mk nie zapewniałyby należytej dokładności przetwarzania. Gdy wymagana jest większa precyzja pomiaru napięć, używa się zewnętrznych przetworników A/C.

Przetworniki te konstruowane są jako bloki w pełni **autonomiczne**, mogące pracować niezależnie od procesora rdzeniowego. Posiadają własne układy sterowania, generatory (lub niezależne dzielniki) sygnałów zegarowych, z reguły współpracują z 4 - 8 kanałowymi multiplekserami i układami próbkująco-pamiętającymi S&H (*sample-and-hold*).

Przetwornik A/C oparty na metodzie SAR składa się z (rys. 2.39):

- z **układu próbkująco-pamiętającego**,
- **komparatora**,

- **rejestru aproksymacyjnego** (zawierającego wynik konwersji),
- **układu sterowania** realizującego algorytm metody SAR (zawiera on rejestry sterujące),
- **przetwornika C/A** (najczęściej drabinka R-2R),
- **multiplekserów analogowych**,
- źródła napięcia odniesienia,
- niekiedy oddzielnego układu zasilania części analogowej (w celu wyeliminowania zakłóceń).



Rys. 2.39. Uproszczony schemat blokowy przetwornika A/C pracującego na zasadzie SAR

Cykl konwersji w tym układzie zaczyna się od pobrania próbki mierzonego napięcia wejściowego i zapamiętania jej w pojemności C. Rejestr aproksymacyjny jest zazwyczaj inicjowany w taki sposób, że ma ustawiony najbardziej znaczący bit MSB, a pozostałe bity wyzerowane. Zawartość rejestru reprezentuje zatem napięcie równe połowie maksymalnego napięcia (zakresu pomiarowego). Wartość ta jest zmieniana przez przetwornik C/A na napięcie i porównywana z napięciem mierzonym. W zależności od wyniku porównania układ sterowania pozostawia najbardziej znaczący bit rejestru niezmieniony albo go neguje. Pierwszy przypadek ma miejsce, gdy napięcie mierzone jest większe od połowy napięcia zakresowego, drugi gdy jest mniejsze. Po określeniu wartości najbardziej znaczącego bitu rejestru aproksymacyjnego przetwornik ustawia w rejestrze kolejny bit i powtarza opisaną wyżej procedurę, aż do ostatniego najmniej znaczącego bitu LSB.

Ze względu na problemy natury technologicznej wewnętrzne przetworniki A/C nie osiągają zwykle rozdzielczości większej niż 10 bitów. W wielu przypadkach jest to wystarczające np. do wykrywania zmian napięcia na wejściu, sprawdzeniu stanu naładowania akumulatora lub baterii. W takim przypadku wystarcza, by charakterystyka przetwornika spełniała wymagania:

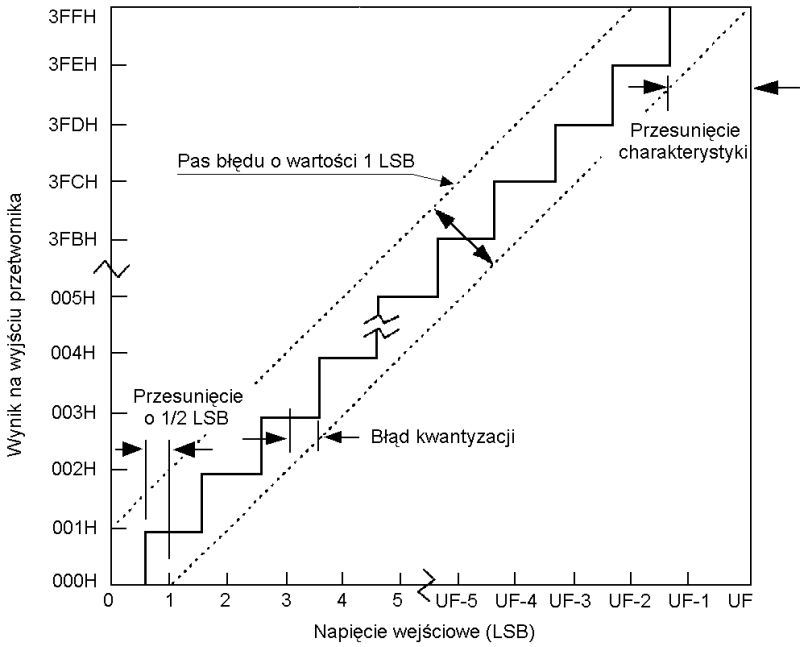
- **monotoniczności** (*monotonicity*)
- niewystępowanie efektów **brakujących kodów**– wraz ze wzrostem napięcia wejściowego od zera do napięcia maksymalnego przetwornik musi dawać na wyjściu wszystkie kolejne liczby od 0 do 2^{n-1}, gdzie n – rozdzielczość przetwornika. Przy zmniejszaniu napięcia przetwornik generuje natomiast liczby malejące o jedność od wartości 2^{n-1} do zera.

W praktyce parametry przetwornika A/C określone są przez:

- **długość słowa** (najczęściej 10 bitów),
- **czas konwersji** (typowo od kilku do kilkunastu μs),
- **rozdzielczość** (zwykle 1 LSB ~ 20 mV przy zakresie 5V dla długości 8 bitów),
- **błąd całkowity** (*absolute error*).

Natomiast w skład błędu całkowitego wchodzą następujące składniki:

- błąd kwantyzacji (*quantization error*),
- błąd przesunięcia zera (*zero-offset error*),
- błąd skalowania przetwornika (*full-scale error*),
- błąd nieliniowości całkowitej (*integral nonlinearity error*),
- błąd nieliniowości różniczkowej (*differential nonlinearity error*).



Rys. 2.40. Charakterystyka przejściowa idealnego, 10-bitowego przetwornika A/C

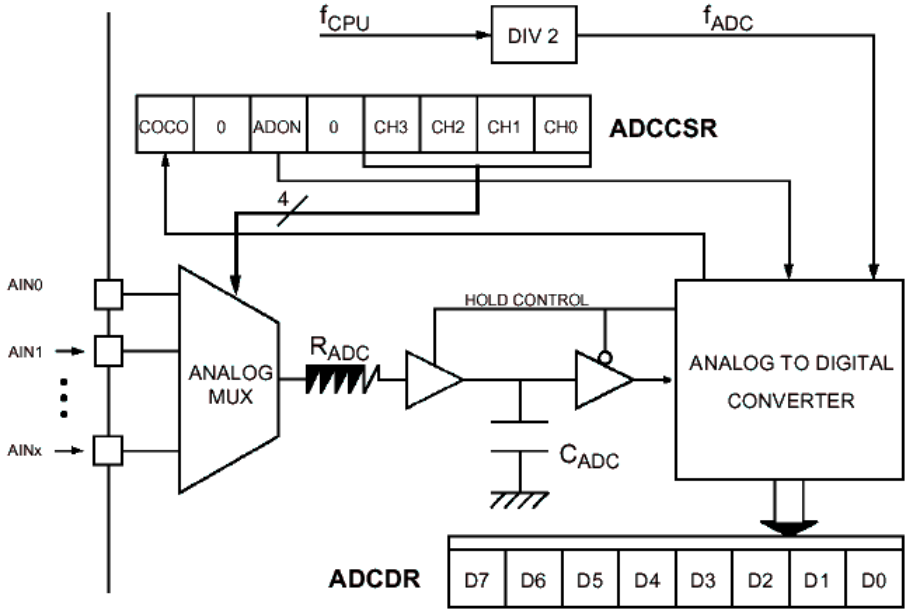
Na rys. 2.40 pokazano idealną charakterystykę 10-bitowego przetwornika A/C. Zaznaczono na niej błąd kwantyzacji wynikający z przesunięcia charakterystyki o ½ LSB oraz pas błędu, w którym ta charakterystyka się mieści.

Np. mk ST72215G posiada 8-bitowy przetwornik A/C z sukcesywną aproksymacją (rys. 2.41) . Układ ten posiada 16 multipleksowanych kanałów analogowych (patrz schemat pinu rys.2.29) umożliwiających pomiar napięcia na jednym z 16 pinów. Wynik konwersji jest trzymany w 8-bitowym rejestrze danych ADCDR. Przetwornik jest sterowany za pomocą rejestru kontrolno/statusowego ADCCSR.

Aby dokonać pomiaru napięcia należy wykonać następujące kroki:

- wybrać kanał konwersji bity CH[3:0] w rej. ADCCSR,
- pin, z którego będzie brany sygnał analogowy musi być ustawiony jako wejście bez pull-up i bez przerwania,
- ustawić bit ADON w rej ADCCSR aby rozpocząć konwersję,
- konwersja kończy się ustawieniem bitu COCO w rej. ADCCSR, wynik konwersji znajduje się w rej. ADCDR, nie jest generowane przerwanie.

Zapis do rej. ADCCSR (z ustawionym bitem ADON) przerywa bieżącą konwersję, resetuje bit COCO i rozpoczyna nową konwersję.

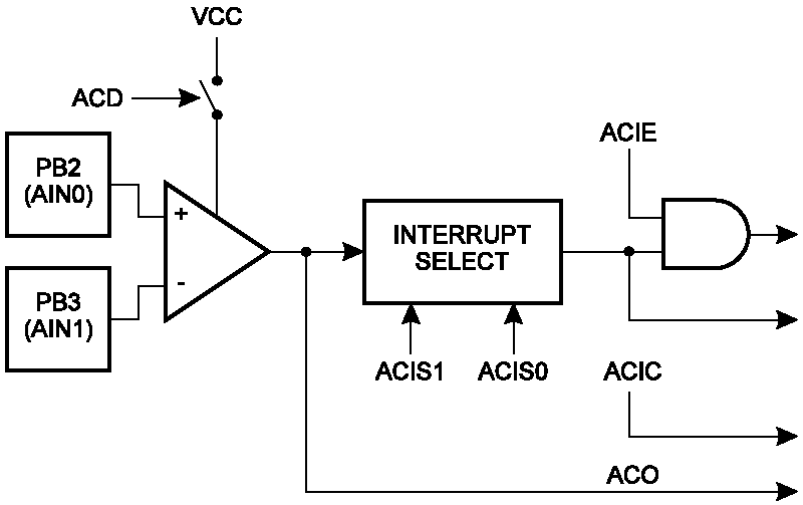


Rys. 2.41. Schemat blokowy przetwornika A/C mk ST72215G

2.6.3. Komparatory analogowe

Szczególnym przypadkiem przetworników A/C są **komparatory analogowe**. Są one najprostszym blokiem pozwalającym na dokonanie **1-bitowej konwersji** sygnału z postaci analogowej na cyfrową. Często w tych układach można ustawić **próg komparacji** (poziom komparacji) oraz **warunek komparacji**, tzn. co się stanie w przypadku, gdy napięcie wejściowe wzrośnie powyżej progu komparacji (gdy ustawiliśmy taki warunek) lub zmaleje poniżej jego. Najczęściej ustawiana jest odpowiednia flaga i generowane jest przerwanie, o ile nie zostało zablokowane.

Dla przykładu na rys. 2.42 pokazano schemat blokowy komparatora analogowego mk AT90S8515.



Rys. 2.42. Schemat blokowy komparatora analogowego mk AT90S8515

Komparator ten dokonuje porównania napięć na wejściu „+” pin PB2 (AINO) i „-” pin PB3 (AIN1). Jeśli napięcie na pinie „+” jest wyższe niż na pinie „-”, wyjście komparatora jest ustawiane na „1” (pin AC0). Wyjście to można podłączyć do wejścia licznika. W tym

przypadku może ono realizować funkcję licznika „zatrzaśnięcie wejścia". Wyjście komparatora może również wyzwalać przerwanie, zboczem narastającym lub opadającym oraz w trybie przełączania.

Układ ten jest sterowany przez rejestr ACSR umieszczony pod adresem $08 ($28), dla którego znaczenie poszczególnych bitów przedstawiono poniżej:

Bit	7	6	5	4	3	2	1	0
	ACD	-	AC0	ACI	ACIE	ACIC	ACIS1	ACIS0

7: ACD – Włączenie / wyłączenie komparatora.
Gdy bit ma wartość 1, komparator nie działa. Redukuje to pobór mocy. Bit ten wolno zmieniać tylko przy wyłączonym przerwaniu komparatora (gdy ACIE jest wyzerowany).

6 : Zarezerwowane.

5 : ACO – Wyjście komparatora.

4 : ACI – Flaga przerwania komparatora. Jest ustawiana, gdy komparator generuje przerwania w trybie przełączania (patrz bity ACIS1 i ACIS0).

3 : ACIE – Przerwanie komparatora włączone. Gdy bit ten jest ustawiony, aktywowane jest przerwanie komparatora. Oczywiście musi być także odblokowany bit globalnego zezwolenia na przerwania (bit I w rej. SREG).

2 : ACIC – Włączenie trybu zatrzaśnięcia wejścia. Gdy bit ten jest ustawiony (1), włączone jest wyzwalanie funkcji „zatrzaśnięcia wejścia" licznika 1.

1,0 : ACIS1, ACIS0 – Tryb wyzwalania przerwania komparatora. Możliwe kombinacje:

0 0 –tryb przełączania, przerwanie przy przełączeniu,
0 1 – zarezerwowane,
1 0 – zbocze opadające,
1 1 – zbocze narastające.

Bity te wolno zmieniać tylko przy wyłączonym przerwaniu komparatora (gdy bit ACIE = 0).

2.6.4. Wewnętrzna pamięć EEPROM

W niektórych mk stosuje się **nieulotną pamięć danych typu EEPROM**. Można w niej trzymać np. dane pomiarowe, zmienne konfiguracyjne, które nie mogą ulec skasowaniu po wyłączeniu zasilania lub w trakcie resetu mk. Zaletą tej pamięci jest możliwość kasowania jej zawartości i wpisu nowych danych przez program użytkownika zawarty w mk. Najczęściej pamięć ta ma wymiar 64, 128, 256 lub 512 bajtów.

Zapis, jak i odczyt z tej pamięci odbywa się za pośrednictwem kilku rejestrów: **rejestru sterującego**, **rejestru danych** i jednego lub dwóch **rejestrów adresu**.

Generalnie odczyt z pamięci EEPROM przebiega według następującej procedury:

- do rejestru(ów) adresu wpisuje się adres bajtu w pamięci EEPROM, spod którego chcemy pobrać daną,
- ustawiamy odpowiedni bit uruchamiający proces odczytu w rejestrze sterującym,
- czekamy, aż ustawi się odpowiednia flaga informująca o zakończeniu odczytu,
- w rejestrze danych znajduje się już nasza dana.

Natomiast zapis danej do pamięci najczęściej przebiega według następującej procedury:

- do rejestru(ów) adresu wpisuje się adres bajtu w pamięci EEPROM, do którego chcemy wpisać daną,
- do rejestru danych wprowadzamy naszą daną,
- ustawiamy odpowiedni bit uruchamiający proces zapisu do EEPROM (często należy również ustawić bity odblokowujące zapis),

- czekamy, aż ustawi się odpowiednia flaga informująca o zakończeniu zapisu (zapis trwa około od 2ms do 4ms).

Najczęściej procedury zapisu danych do pamięci EEPROM najpierw kasują zawartość docelowej komórki (bajtu), a następnie wpisują do niej naszą daną. Gdy tak nie jest, to należy pamiętać, aby przed zapisem do EEPROM skasować zawartość tej komórki (poprzez ustawienie odpowiednich bitów w rejestrze sterującym). Kasowanie pamięci EEPROM, FLASH, EPROM polega na ustawieniu w ich poszczególnych komórkach wartości FFh (samych jedynek).

Np. dla mk AT90S8515 posiada pamięć EEPROM o rozmiarze 512 bajtów (producent gwarantuje 100 000 cykli poprawnego zapisu / kasowania).

Rejestry dostępu do EEPROM'u znajdują się w przestrzeni adresowej we/wy. Czas dostępu przy zapisie zależy od napięcia zasilania i wynosi 2,5 – 4 ms. Kiedy następuje dostęp do pamięci EEPROM, jc jest zatrzymywana przez dwa takty zegara, a następnie kontynuuje wykonywanie programu. Operacje dotyczące dostępu do pamięci EEPROM wykonuje się korzystając z rejestrów:

- adresowych – EEARH, EEARL,
- danych – EEDR,
- sterującego – EECR.

Rejestr adresowy pamięci EEPROM zawiera adres komórki do której realizowany będzie dostęp. EEARL zawiera młodsze 8 bitów adresu, a EEARH jeden najstarszy bit adresu (do zaadresowania 512 bajtów). Rejestr danych EEDR zawiera daną, która ma być zapisana lub która została odczytana z pamięci EEPROM. Znaczenie bitów rejestru sterującego EECR pokazano poniżej:

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	EEMWE	EEWE	EERE

7..3: Zarezerwowane.

2 : EEMWE – Ogólny dostęp do zapisu, jedynka oznacza, że wpisanie jedynki do EEWE spowoduje zapis do pamięci EEPROM. Bit EEMWE zeruje się automatycznie po upływie 4 cykli od ustawienia.

1 : EEWE – Wpis do pamięci EEPROM. Wpisanie jedynki przy ustawionym EEMWE dokonuje wpisu zawartości rejestru EEDR pod adres EEAR. Bit EEWE zeruje się po czasie 2,5 do 4 ms od ustawienia. Przez 4 cykle od ustawienia CPU jest zatrzymany. Procedura wpisu danej do pamięci EEPROM:

1. Czekaj aż EEWE = 0.
2. Wpisz nowy adres do EEARL i EEARH.
3. Wpisz daną do EEDR.
4. Wpisz „1" do EEMWE.
5. W ciągu 4 cykli wpisz „1" do EEWE.

Zwraca się uwagę na konieczność wyłączenia przerwań na czas wykonywania wpisu, gdyż obsłużenie przerwania może zakłócić powyższą procedurę.

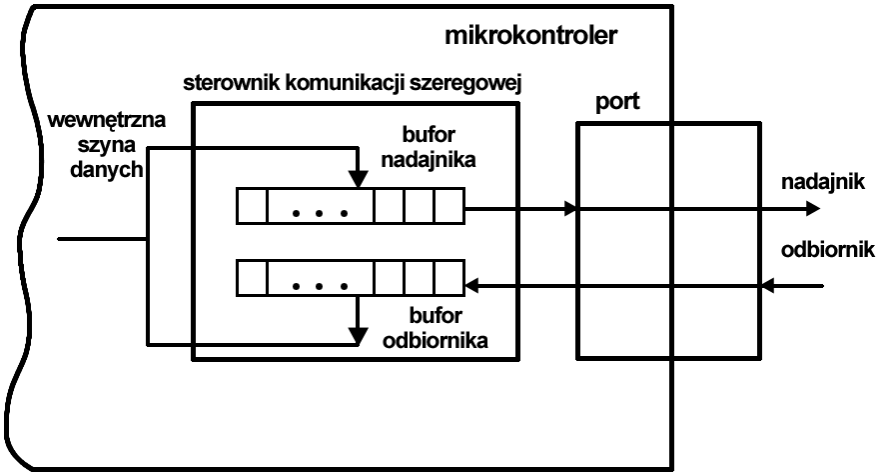
0 : EERE – Odczyt z EEPROM'u. Wpisanie „1" powoduje zatrzymanie CPU na dwa cykle zegarowe i przesłanie danej spod adresu EEAR do rejestru EEDR. Zakończenie odczytu jest sygnalizowane sprzętowym wyczyszczeniem bitu EERE. Przed odczytem należy testować ten bit czy na pewno zawiera „0".

Mk PIC16F873 również posiada wewnętrzną pamięć EEPROM (128 bajtów). Stowarzyszono z nią sześć rejestrów: EEDATA, EEDATAH – rejestry danych, EEADR, EEADRH – rejestry adresu oraz EECON1, EECON2 – rejestry sterujące. Rejestry te służą nie tylko do zapisu/odczytu danych z tej pamięci, ale również do zapisu i odczytu danych z

pamięci FLASH mk, w której to znajduje się kod programu. Cecha ta pozwala na odczyt danych z pamieci FLASH, jak i nawet na **zmianę kodu programu przez sam program zawarty wewnątrz mk**.

2.6.5. Sterowniki komunikacji szeregowej

Sterowniki komunikacji szeregowej służą do wymiany informacji pomiędzy mk, a jego otoczeniem. Przesyłanie danych odbywa się w **sposób szeregowy**. Zasadę działania tych układów pokazano na rys. 2.43.



Rys. 2.43. Schematyczna budowa sterownika komunikacji szeregowej

Urządzenie to umożliwia wysyłanie zawartości określonego rejestru, tzw. **bufora nadajnika**, w postaci szeregowej poprzez określone wyprowadzenia portu. Oznacza to, że na wyjściu linii portu pojawia się ciąg binarny odpowiadający zawartości wysyłanego rejestru (funkcja nadajnika – ***transmiter***). W funkcji odbiornika (***receiver***) sterownik komunikacji szeregowej potrafi przetworzyć ciąg binarny doprowadzony do wejścia określonej linii portu na zawartość rejestru, zwanego **buforem odbiornika**. Bufory nadajnika i odbiornika są dostępne z poziomu programu użytkownika.

Wyróżnia się dwa rodzaje transmisji szeregowej:

- **asynchroniczną**,
- **synchroniczną**.

Dane przesyłane **asynchronicznie nie są związane z żadnym sygnałem synchronizującym**, w szczególności nie towarzyszy im sygnał zegara. Transmisja przebiega zwykle bajtami przesyłanymi w postaci ciągów bitów. Oprócz ośmiu bitów danych przesyła się dodatkowo bit startu oraz opcjonalnie bit parzystości do detekcji błędów i bit stopu. Pomiędzy nadajnikiem, a odbiornikami **musi być ustalona częstotliwość przesyłania danych**.

Przy **transmisji synchronicznej** równolegle z ciągiem bitów danych przesyła się **sygnał synchronizujący** (zegarowy), który określa chwile, w których stan linii danych odpowiada ważnym wartościom kolejnych bitów. Po każdym bajcie może być dodatkowo przesłany bit parzystości.

Sterownik komunikacji szeregowej (w skrócie: **port szeregowy**) nazywa się **dwukierunkowym** (dupleksowym), jeśli może równocześnie odbierać i nadawać dane. Jest to możliwe, gdy jest wyposażony w dwie oddzielne linie do nadawania i do odbioru.

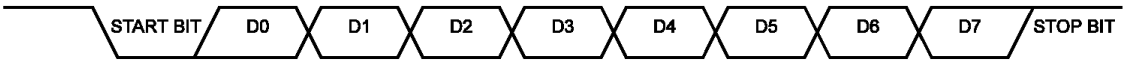
Można wyróżnić między innymi następujące interfejsy szeregowe stosowane w mk:

- UART (*Universal Asynchronous Receiver/Transmitter*),
- SPI (*Serial Peripherial Interface*),
- 1-Wire,
- I^2C (*Inter-Integrated Circuit*),
- CAN (*Cotroller Area Network*),
- USB (*Universal Serial Bus*).

2.6.5.1. Interfejs UART

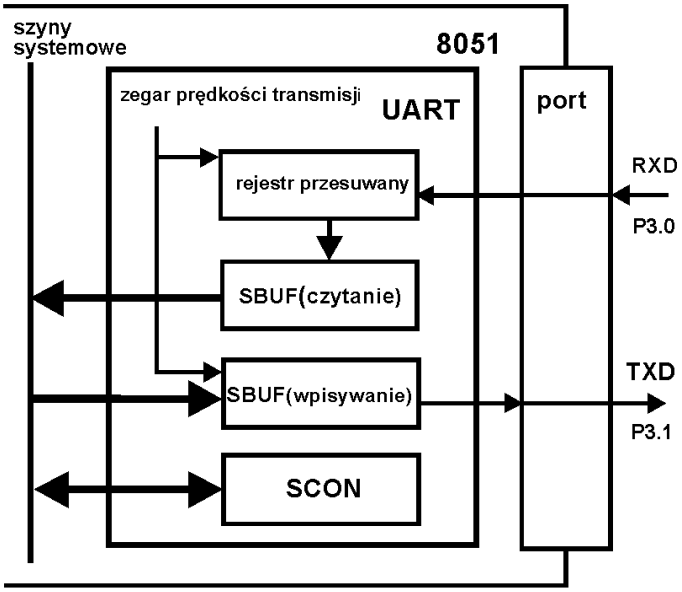
Interfejs ten przewidziany jest zazwyczaj do zapewnienia łączności mk z komputerem PC lub innym mk. Mk posiadają często rozbudowane układy UART posiadające własne generatory sygnałów odniesienia, bufory, dekodery kodów sterujących, mogące pracować w trybie dwukierunkowym z protokołem potwierdzeń sprzętowych (Xon/Xoff). W praktyce jednak nie spotyka się tak często zewnętrznych układów scalonych wyposażonych w ten system interfejsowy. Nie może więc on być traktowany jako platforma rozszerzenia mse, lecz jedynie jako interfejs dedykowany budowie specjalizowanego łącza, np. do komunikacji z komputerem PC.

Port szeregowy UART kontaktuje się ze światem zewnętrznym za pomocą dwóch wyprowadzeń: **wejścia odbiornika** (oznaczanego najczęściej RxD) i **wyjścia odbiornika** (TxD). Interfejs UART jest interfejsem **asynchronicznym**, najczęściej posiada swój własny **generator taktujący**.



Rys. 2.44. Format danych dla standardu UART

Format danych w tym standardzie pokazano na rys. 2.44. Jak widać, transmisja zaczyna się od **bitu startu**, po którym następuje **osiem bitów danej** (czasami dziewięć, gdzie dziewiąty bit jest najczęściej bitem parzystości) i jednego **bitu stopu** (w skrócie: 8N1).



Rys. 2.45. Schematyczna budowa urządzenia UART mk 80C51

Jako przykład takiego urządzenia przedstawiono urządzenie UART mk 80C51 (rys. 2.45). Do rej. SBUF użytkownik wpisuje dane przeznaczone do wysłania wyjściem TxD i odczytuje z niego dane, które przyszły wejściem RxD. Fizycznie są to dwa rejestry pod wspólną nazwą, ponieważ operacja zapisu dotyczy innego rejestru niż operacja czytania. Zapis danej do rej.

SBUF automatycznie uruchamia procedurę wysyłania danej przez interfejs. Rejestr SCON służy do sterowania pracą interfejsu oraz zawiera informację o jego stanie. Jego postać jest następująca:

SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0, SM1 – wybór trybu pracy:
- 0 0 tryb 0 – rejestr przesuwny (8 bitów), $f_{osc}/12$,
- 0 1 tryb 1 – UART (8 bitów), częstotliwość zmienna,
- 1 0 tryb 2 – UART (9 bitów), $f_{osc}/64$ lub $f_{osc}/32$,
- 1 1 tryb 3 – UART (9 bitów), częstotliwość zmienna.

SM2 - maskowanie odbioru znaku:
- tryb 0 – powinno być SM2 = 0,
- tryb 1 – przy SM2 = 1 znacznik przerwania RI nie jest uaktywniany przy braku bitu stopu w odbieranym słowie,
- tryb 2 i 3 – przy SM2 = 1 znacznik przerwania RI nie jest uaktywniany jeżyli 9-ty bit odbieranego słowa jest równy 0; w tych trybach znacznik ten jest wykorzystywany przy komunikacji wieloprocesorowej.

REN - znacznik uaktywnienia odbiornika:
- 0 odbiór zablokowany,
- 1 odbiór dozwolony.

TB8 - znacznik używany w transmisji dziewięciobitowej:
- tryb 0 i 1 – nie używany,
- tryb 2 i 3 – dziewiąty bit w nadawanym słowie.

RB8 - znacznik używany w transmisji dziewięciobitowej:
- tryb 0 – nie używany,
- tryb 1 – gdy SM2, do RB8 jest wpisywany bit stopu odbieranego słowa,
- tryb 2 i 3 – do RB8 jest wpisywany dziewiąty bit odbieranego słowa.

TI - znacznik przerwania nadajnika. Ustawiany w stan 1 przez mikrokontroler tylnym zboczem 9-tego bitu wysyłanego słowa w trybie 0 lub przednim zboczem bitu stopu w pozostałych trybach. Może być zerowany tylko programowo.

RI - znacznik przerwania odbiornika. Ustawiany w stan 1 przez mikrokontroler tylnym zboczem 9-tego bitu przyjmowanego słowa w trybie 0 lub w połowie bitu stopu w pozostałych trybach z wyjątkiem gdy w trybach 2 i 3 bit SM2 = 1, a dziewiąty bit odbieranego słowa jest równy 0.

Jednym z elementów konfiguracji urządzenia UART jest ustawienie prędkości transmisji. Dla trybu 1 i 3 interfejs taktuje się za pomocą liczników mk. W celu uzyskania jednej ze standardowych prędkości transmisji np. 9600 bitów na sekundę niezbędne jest zastosowanie oscylatora np. o częstotliwości 11,059MHz. „Okrągłe" częstotliwości pożądane w układach pomiarowych np. 12MHz nie zapewniają prawidłowej prędkości transmisji, która jest obarczona błędem. Gdy błąd ten nie jest duży (poniżej jednego procenta), to nie wpływa on na poprawność wymiany danych.

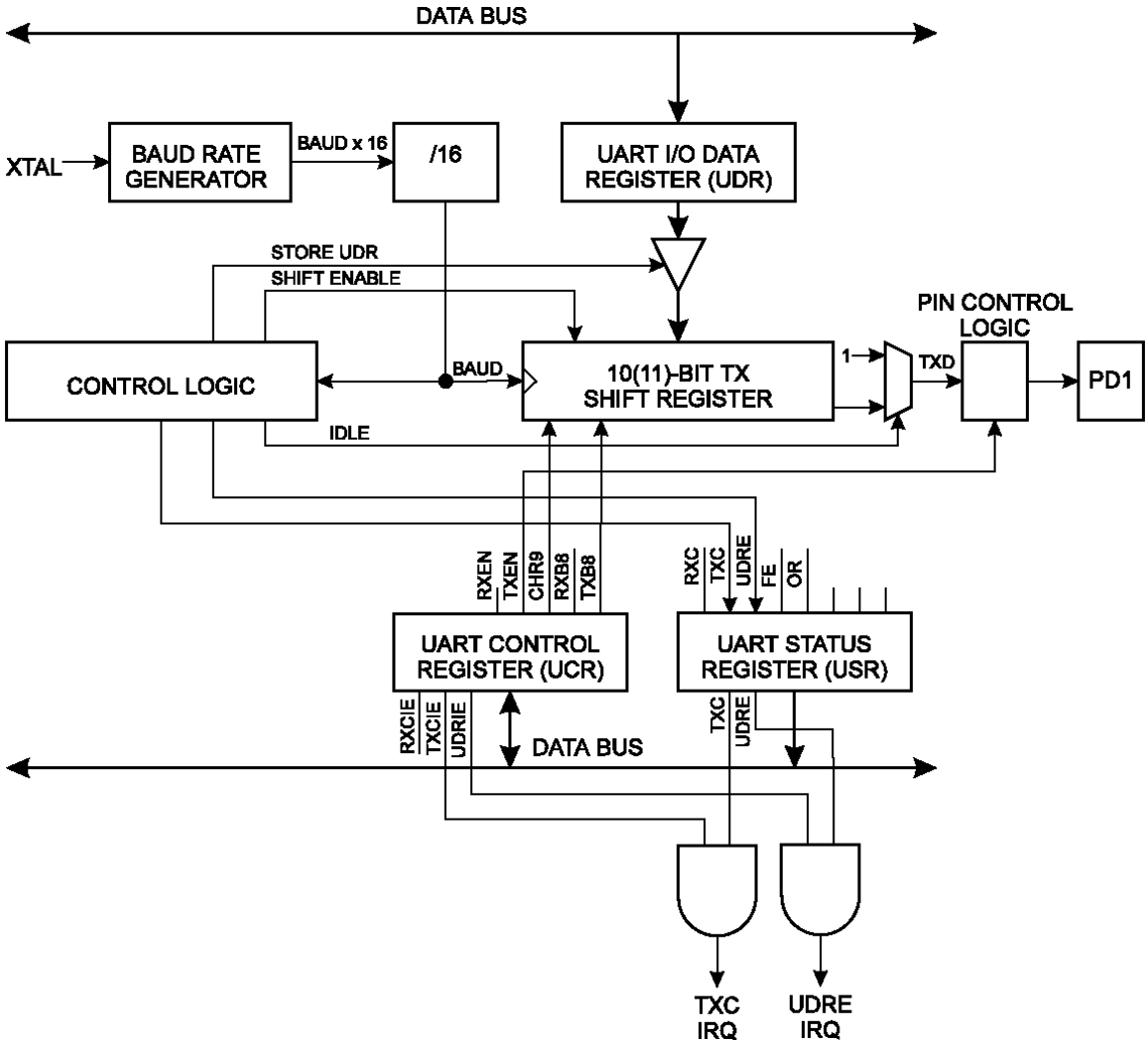
Drugim przykładem jest interfejs UART mk AT90S8515. Umożliwia on transmisję szeregową asynchroniczną z pełnym dupleksem. Posiada następujące cechy :
- full duplex (posiada oddzielne rejestry do odbioru i nadawania),
- można ustawić wiele prędkości transmisji,

- wysyłanie/odbieranie 8 lub 9 bitów danej,
- zapewnia filtrację szumów,
- ma detekcje błędu kolizji i błędu ramki,
- zapewnia detekcję błędu nieprawidłowego startu,
- generuje 3 oddzielne przerwania (Tx zakończone, Tx rejestr danej pusty, Rx zakończone).

Układ ten składa się z trzech oddzielnych bloków: **układu do nadawania** danej (nadajnika), **układu do odbioru** danej (odbiornika) oraz **generatora taktującego**.

W układzie **nadajnika** (rys. 2.46) transmisja jest inicjowana przez wpis danej do rejestru danej układu UART (rej. UDR). Dana jest transferowana z rej. UDR do rejestru przesuwnego gdy :

- Nowa dana została wpisana do rej. UDR po tym jak wysłany został ostatni bit danej poprzedniej. Rejestr przesuwny jest ładowany natychmiast.
- Nowa dana została wpisana do rej. UDR zanim zakończono wysyłanie poprzedniej. Rejestr przesuwny jest ładowany dopiero po wysłaniu bitu stopu danej poprzedniej.



Rys. 2.46. Schemat blokowy układu nadajnika UART w mk AT90S8515

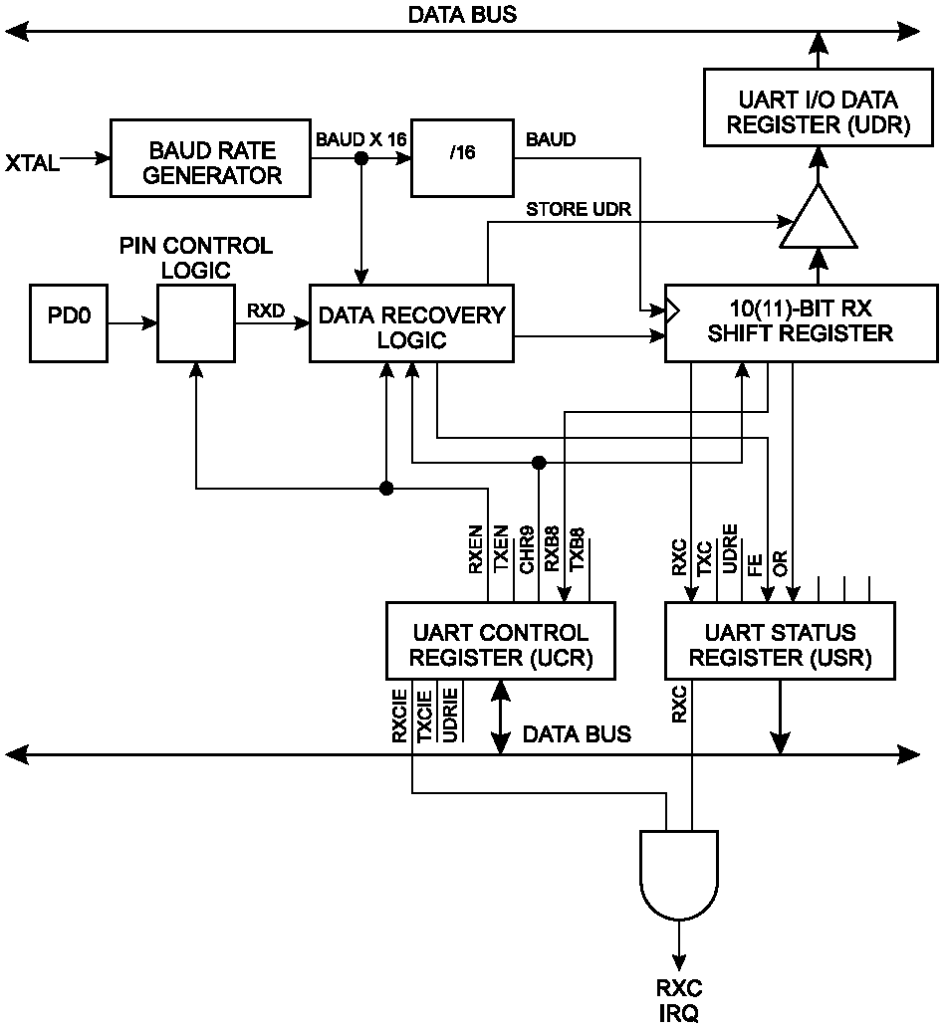
Przesłanie danej z rej. UDR do rejestru przesuwnego powoduje ustawienie flagi UDRE w rejestrze statusowym USR. Flaga ta sygnalizuje gotowość przyjęcia następnej danej do nadawania. Zaraz po załadowaniu rejestru przesuwnego czyszczony jest jego najmłodszy bit (bit startu), a bit 9-ty lub 10-ty jest ustawiany (bit stopu). W przypadku wysyłania danej

dziewięciobitowej (patrz bit CHR9 w rejestrze sterującym UCR), 9-ty bit danej jest przepisywany na 9-ty bit rejestru przesuwnego.

Z prędkością ustawianą za pomocą rej. UBRR, następuje przesyłanie bitów, rozpoczynając od bitu startu, następnie od bitu LSB, kończąc na bicie stopu. Bity są wystawiane na pin TxD. Po wysłaniu bitu stopu, następuje przesłanie nowej danej z rej. UDR do rejestru przesuwnego (o ile nowa dana została tam wpisana w czasie transmisji). Podczas przesyłania ustawiana jest flaga UDRE. W przypadku braku nowej danej, bit UDRE pozostaje ustawiony, aż do momentu wpisania danej do rej. UDR. Jeśli nie ma już danych do przesłania, a bit stopu pozostaje na TxD przez czas potrzebny na przesłanie 1 bitu, ustawiana jest flaga TxC w rej. USR.

Bit TXEN w rej. UCR włącza tryb nadawania UART. Jeśli jest wyczyszczony, pin PD1 może być użyty jako końcówka ogólnego przeznaczenia. Jeśli jest ustawiony, pin PD1 będzie ustawiony jako wyjście i to niezależnie od stanu rejestru kierunku danych portu D.

Drugim układem w interfejsie UART jest **odbiornik**, którego schemat blokowy pokazano na rys. 2.47.



Rys. 2.47. Schemat blokowy układu odbiornika UART w mk AT90S8515

Układ odbioru próbkuje sygnał wejściowy (RxD) z częstotliwością 16 * BAUD, gdzie BAUD – częstotliwość układu taktującego. Gdy linia jest w stanie spoczynkowym, pojedyncza próbka o stanie „0” jest interpretowana jako zbocze opadające bitu startu i

rozpoczyna się procedura sprawdzająca. Testuje ona, czy rzeczywiście stan „0” na linii jest bitem startu czy też zakłóceniem, i w tym drugim przypadku ignoruje zbocze.

Jeśli bit startu jest stwierdzony poprawnie (3 próbki z rzędu to „0”), następuje odczyt kolejnych bitów danej, rozpoczynając od LSB. Każdy bit próbkowany jest 3 razy i jako prawdziwa wartość interpretowana jest ta, która pojawia się w przynajmniej 2 próbkach. Pozwala to na eliminację zakłóceń.

Kiedy do odbiornika trafia bit stopu, przechodzi on procedurę sprawdzenia poprawności i jeśli z 3 próbek 2 lub 3 są zerami, ustawiana jest flaga FE (błąd ramki). Przed odczytaniem rej. UDR należy więc zawsze sprawdzać stan flagi FE.

Niezależnie od ewentualnego błędu ramki, dana jest przesyłana do rej. UDR i ustawiana jest flaga RxC.

W rzeczywistości rej. UDR nie jest jednym rejestrem, lecz dwoma oddzielnymi dla wysyłania i odbioru. W przypadku zapisu do rej. UDR dana jest wpisywana do rejestru wysłania, a w przypadku odczytu podstawiana jest zawartość rejestru odbioru. Jeśli ustawiony jest tryb odbioru danej 9–bitowej (patrz bit CHR9 w rej. UCR), bit RXB8 w rej. UCR jest ładowany wartością 9-tego bitu danej odebranej.

Jeśli po odebraniu danej stwierdzi się brak pobrania z rej. UDR danej poprzedniej, ustawiana jest flaga kolizji OR w rej. UCR. Oznacza ona, że ostatnio odbierana dana nie mogła zostać przesłana do rej. UDR i została utracona. Flaga OR jest czyszczona przy odczycie prawidłowej danej z rej. UDR. Przed odczytem rej. UDR należy więc zawsze sprawdzić flagę OR.

Kiedy bit RXEN w rej. UCR jest wyczyszczony, odbiornik jest wyłączony. Pin PD0 może być użyty jako linia portu równoległego. Gdy RXEN = „1” do linii PD0 podłączone jest wejście odbiornika, co oznacza, że staje się on pinem wejściowym, niezależnie od ustawień w rejestrze kierunku pinów DDRD portu D.

Kiedy ustawiony jest bit CHR9 w rej. UCR, dane odbierane i nadawane mają długość 9 bitów plus bit startu i stopu. 9-ty bit danej do nadania należy wpisać pod bit TXB8 w rej. UCR, a odebrany 9-ty bit danej znajduje się w RXB8 rej. UCR. Wpis bitu do TXB8 musi nastąpić wcześniej niż wpis reszty danej do rej. UDR, co rozpoczyna transmisję.

Do sterowania układem UART mk AT90S8515 wykorzystuje się cztery rejestry:

- Danej, rej. UDR (dana odbierana lub nadawana),
- Statusowy, rej. USR (flagi błędów, zakończenie transmisji, etc.),
- Sterowania, rej. UCR (włączenie przerwań, włączenie / wyłączenie UART),
- Prędkości, rej. UBRR (BAUD = fck / (16 (UBRR+1))).

Rejestr statusowy – rej. USR o adresie $0B ($2B) posiada następujące flagi:

Bit	7	6	5	4	3	2	1	0
	RXC	TXC	UDRE	FE	OR	-	-	-

7 : RXC – Odbiór zakończony. Gdy flaga jest ustawiona, odebrana dana została przesłana do rej. UDR. Flaga ta jest ustawiana niezależnie od błędów transmisji. W przypadku włączenia przerwania (RXCIE w rej. UCR), ustawienie RXC oznacza wywołanie obsługi przerwania zakończenia odbioru. RXC czyszczona jest przez odczyt rej. UDR. Obsługa przerwania zakończonego odbioru musi odczytać rej. UDR i wyczyścić flagę RXC.

6 : TXC – Nadawanie zakończone. Zakończono nadawanie bitu stopu, a w rej. UDR nie ma nowej danej. Flaga ta jest użyteczna w trybie *half-duplex*, kiedy urządzenie po nadaniu danych musi zwolnić linię i ustawić się w tryb odbioru.

Jeśli ustawiona jest flaga TXCIE w rej. UCR, wykona się obsługa przerwania zakończonego nadawania. TXC czyści się wtedy automatycznie. W przypadku wyłączonego przerwania TXC można wyczyścić poprzez wpisanie logicznej „1”.

5 : UDRE – Rejestr danej pusty. Flaga ta ustawia się przy przesłaniu danej z rej. UDR do rejestru przesuwnego celem nadania i oznacza, że do rej. UDR można wpisać kolejną daną.

4 : FE – Błąd ramki. Ustawia się gdy zamiast bitu stopu stwierdza się „0”. Czyści się automatycznie po prawidłowym odbiorze bitu stopu.

3 : OR – Błąd kolizji. Ustawia się, gdy nie można przesłać odebranej danej do rej. UDR, ponieważ nie odczytano z tego rejestru danej poprzedniej. Zeruje się przy prawidłowym przepisaniu danej do rej. UDR.

2..0 : – Zarezerwowane.

Do sterowania pracą portu UART wykorzystuje się rej. UCR spod adresu $0A ($2A). Znaczenie bitów jest następujące:

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8

7 : RXCIE – Włączenie przerwania zakończonego odbioru. Gdy flaga jest ustawiona, ustawienie flagi RXC w rej. USR powoduje wywołanie obsługi przerwania (oczywiście przy odblokowanej globalnej masce przerwań).

6 : TXCIE – Włączenie przerwania zakończonego nadawania. Gdy flaga jest ustawiona, ustawienie flagi TXC w rej. USR powoduje wywołanie obsługi przerwania (oczywiście przy odblokowanej globalnej masce przerwań).

5 : UDRIE – Włączenie przerwania „rejestr danej pusty”. Gdy flaga jest ustawiona, ustawienie flagi UDRE w rej. USR powoduje wywołanie obsługi przerwania (oczywiście przy odblokowanej globalnej masce przerwań).

4 : RXEN – Bit ten włącza tryb odbioru (gdy RXEN=”1”). Gdy RXEN=”0” flagi TXC, OR i FE nie mogą mieć wartości „1”. Jeśli flagi te są ustawione, wyłączenie RXEN nie powoduje ich czyszczenia.

3 : TXEN – Bit ten włącza tryb nadawania (gdy TXEN=”1”). Wyłączenie tego bitu w czasie transmisji ma efekt dopiero po zakończeniu nadawania zarówno aktualnej danej, jak i ewentualnie następnej już wpisanej do rej. UDR.

2 : CHR9 – Włączenie trybu transmisji danej 9–bitowej. Patrz opis bitów RXB8 i TXB8. Dziewiąty bit może być użyty jako dodatkowy bit stopu lub do kontroli parzystości.

1 : RXB8 – Dziewiąty bit odebranej danej. Czyta się go, jeśli bit CHR9 jest ustawiony.

0 : TXB8 – Dziewiąty bit danej do nadania. Należy go modyfikować, jeśli dana przesłana ma być 9–bitowa (czyli gdy CHR9=”1”).

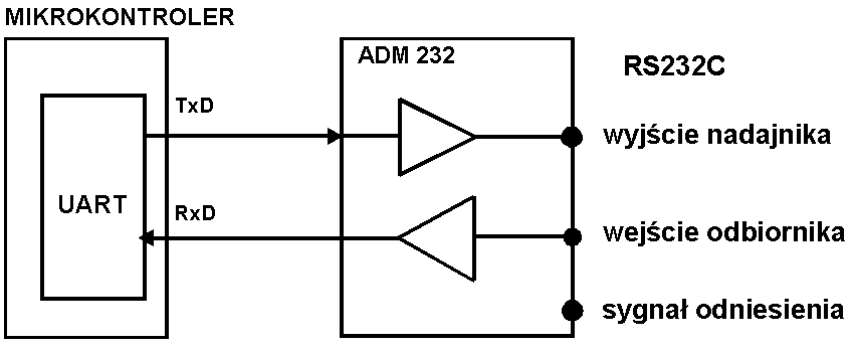
Ostatnim układem interfejsu UART jest **generator taktujący**. Jest on dzielnikiem częstotliwości, który generuje sygnał zegarowy o częstotliwości BAUD = fck / (16 (UBRR+1)), gdzie: rej UBRR jest rejestrem prędkości interfejsu UART (rejestr o adresie $09 ($29)), a fck częstotliwością oscylatora kwarcowego.

Przykładowe prędkości transmisji dla czterech częstotliwości oscylatora przedstawiono w tabeli 2.5. Zawiera ona wartości jakie należy wpisać do rej UBRR, aby uzyskać daną prędkość transmisji. Dodatkowo umieszczono w niej procentowy błąd rzeczywistej prędkości transmisji w stosunku do standardowej. Zaleca się stosować takie prędkości transmisji, aby ten błąd był mniejszy niż 1 %.

Tabela 2.5. Przykładowe prędkości transmisji układu UART i odpowiadające im wartości rej. UBRR

Body (bit/s)	fosc = 3.2768MHz	% błąd	fosc = 3.6864MHz	% błąd	fosc = 4 MHz	% błąd	fosc = 4.608 MHz	% błąd
2400	UBRR=84	0.4	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0
4800	UBRR=42	0.8	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0
9600	UBRR=20	1.6	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0
14400	UBRR=13	1.6	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0
19200	UBRR=10	3.1	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0
28800	UBRR=6	1.6	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0
38400	UBRR=4	6.3	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7
57600	UBRR=3	12.5	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0
76800	UBRR=2	12.5	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	6.7
115200	UBRR=1	12.5	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	20.0

Jak wspomniano, interfejs UART przeważnie służy do komunikacji mk z komputerem PC. Odbywa się to najczęściej za pomocą **standardu RS232**. Sposób realizacji sprzętowej standardu RS232C pokazano na rys. 2.48.

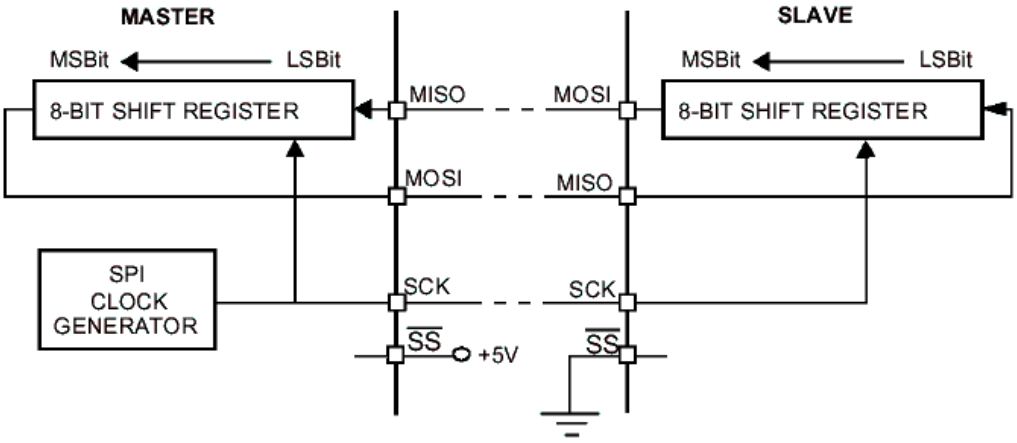


Rys. 2.48. Realizacja sprzętowa standardu RS232C

Zadaniem układu ADM232 jest konwersja sygnałów między poziomem TTL na wyprowadzeniach mk, a poziomem sygnału +/- 12V wykorzystywanym podczas transmisji łączem RS232C. Łącze to wymaga co najmniej trzech przewodów: **linii nadajnika**, **odbiornika** oraz **linii odniesienia**, czyli tzw. masy. Jest to zatem łącze **asymetryczne**. Umożliwia ono komunikację ze sobą tylko dwóch jednostek przy zastosowaniu transmisji *full duplex*.

2.6.5.2. Interfejs SPI

Interfejs SPI służy do **dwukierunkowej** (*full-duplex*), **synchronicznej**, szeregowej transmisji danych pomiędzy mk, a zewnętrznymi układami peryferyjnymi, np. przetwornikami A/C i C/A, szeregowymi pamięciami EEPROM, innymi mk, układami scalonymi z regulowanymi cyfrowo potencjometrami, dekadami pojemnościowymi, itd. Jest interfejsem **trójżyłowym**: składa się z **dwóch linii synchronicznie przesyłających dane** w przeciwnych kierunkach i **linii z sygnałem zegarowym** synchronizującym ten transfer.



Rys. 2.49. Przykład połączenia interfejsu ISP pomiędzy układem master i slave

Na rys. 2.49 pokazano sposób połączenia interfejsu SPI pomiędzy **układem nadrzędnym** (*master*) i **układem podrzędnym** (*slave*), gdzie pin MISO jest wejściem danych, pin MOSI wyjściem danych, a pin SCK wyjściem zegara dla układu master i wejściem zegara dla układu slave. Z rysunku widać, że protokół transmisji (w najprostszej postaci) wymaga dwóch rejestrów przesuwnych (*shift register*) połączonych w **licznik pierścieniowy** (wejście jest połączone z wyjściem). Transmisja jest synchroniczna - jeden z układów dostarcza sygnału zegara, odseparowanego od sygnału danych, zgodnie ze zboczami zegara taktującego. **Układ generujący sygnał zegara jest określony jako nadrzędny**, bez względu na to czy dane są przez niego nadawane czy odbierane. Wszystkie pozostałe układy na magistrali są określone jako **podrzędne**. Sygnały danych i zegara są przesyłane oddzielnymi jednokierunkowymi liniami. Sygnał zegara nie jest ciągły, nadawany jest jedynie w czasie trwania transmisji. Umożliwia to odbiór poszczególnych bitów danych bez konieczności testowania bitów startu i stopu, charakterystycznego dla transmisji asynchronicznej.

Kiedy układ nadrzędny nadaje dane na linii MOSI do układu podrzędnego, to układ podrzędny odpowiada, wysyłając do układu nadrzędnego dane na linii MISO. Implikuje to dwukierunkową transmisję z jednoczesnym wysyłaniem i odbieraniem danych, synchronizowanym tym samym sygnałem zegarowym, przez oba układy. Zatem bajt transmitowany jest od razu zastępowany bajtem odbieranym. Dzięki temu nie ma „pustych" transmisji: raz aby wysłać bajt, drugi aby go odebrać. Do wysłania bajtu i jego odebrania wystarczy osiem impulsów zegarowych na linii SCK.

W interfejsie tym dane są wpisywane (odbierane) do rejestru szeregowego jednym zboczem zegarowym, a przesuwane (wyprowadzane na zewnątrz – nadawane) drugim. Zwiększa to czas podtrzymania danych odbiornika do $\frac{1}{2}T_{CLK} - t_d$, gdzie T_{CLK} - okres zegara, t_d - opóźnienie magistrali. Zatem wypełnienie sygnału zegarowego wynosi około ½. Najczęściej szybkość transmisji nie przekracza 1-2 Mbit/s.

Wymiana danych za pomocą interfejsu SPI mk pracującego w trybie master pomiędzy mk, a zewnętrznym urządzeniem peryferyjnym generalnie przebiega według następującej procedury:

- najpierw należy odpowiednio skonfigurować interfejs SPI mk: tryb master, poprawnie skonfigurowane piny interfejsu (pin SCK ma być wyjściem), ustawić częstotliwość sygnału zegarowego,
- urządzenie peryferyjne musi być uaktywnione (przygotowane na odbiór danych) – najczęściej służy do tego dodatkowa linia mk podłączona do wejścia CS (*Chip Select*) urządzenia peryferyjnego,

- aby rozpocząć transmisję ustawia się odpowiedni bit w rejestrze sterującym SPI lub wpisuje się daną do rejestru przesuwnego,
- po czym czeka się na zakończenie transmisji, testując flagę informującą o zakończeniu transmisji lub czeka się na przerwanie od układu SPI, o ile jest odblokowane,
- na zakończenie z rejestru przesuwnego można odczytać daną odebraną.

Interfejs SPI mk można skonfigurować do pracy w trybie slave. W tym przypadku procedura postępowania jest następująca:

- najpierw należy odpowiednio skonfigurować interfejs SPI mk: tryb slave, poprawnie skonfigurowane piny interfejsu (pin SCK ma być wejściem),
- następnie wpisuje się daną, którą chcemy wysłać do rejestru przesuwnego,
- ustawia się odpowiedni bit w rejestrze sterującym SPI uruchamiającym interfejs,
- urządzenie peryferyjne musi być aktywne i pracować w trybie master,
- po czym czeka się na zakończenie transmisji, które wywołuje przerwanie, o ile jest odblokowane,
- na zakończenie z rejestru przesuwnego można odczytać daną odebraną.

W trybie slave metoda odpytywania flagi zakończenia transmisji jest nieefektywna, ponieważ transmisja z układu peryferyjnego może nastąpić w dowolnej chwili nieznanej mk. Zatem musi on w pętli głównej programu ciągle odpytywać tę flagę, co niepotrzebnie zajmuje czas procesora. Stąd wiele układów SPI posiada dodatkowy sygnał wejściowy (pin SS), który pozwala na wymuszenie trybu slave interfejsu SPI mk przez urządzenie peryferyjne.

Podsumowując, właściwości układu SPI z punktu widzenia interfejsu mogą być opisane przy pomocy 3 funkcji interfejsowych:

- **Talker** (nadajnik) – wprowadzenie danej do rejestru szeregowego, przesuwanie i wysyłanie danej z rejestru szeregowego.
- **Listener** (odbiornik) - wczytanie danej odbieranej z rejestru szeregowego po zebraniu całego słowa
- **Repeater** (pośredniczenie w transmisji) - dane wejściowe są wpisywane do rejestru szeregowego z linii wejściowej, przesuwane i od razu wysyłane na linię wyjściową.

Maksymalna konfiguracja zawiera z reguły wszystkie trzy wymienione funkcje, ale w praktyce spotyka się najczęściej dwie pierwsze.

Aby transmisja pomiędzy mk, a urządzeniem peryferyjnym przebiegała prawidłowo muszą być spełnione następujące warunki:

- zachowanie **jednakowej długość** danej (najczęściej 8 bitów lub wielokrotność tej liczby),
- **taka sama kolejność wysyłania bitów** (najczęściej od MSB do LSB, niektóre mk mają możliwość programowej zmiany tej kolejności),
- zgodna **polaryzacja i faza sygnału zegarowego**.

Częstotliwość sygnału zegarowego nie jest istotna, ponieważ dane są synchronizowane tym sygnałem i są aktywowane (wpisywane lub wyprowadzane z rejestru szeregowego) wyłącznie przez zbocza tego sygnału. Zatem prędkość transmisji możemy zmienić nawet w trakcie przesyłania danych. Można również wstrzymać transmisję w dowolnej chwili, aby następnie ją dalej kontynuować. Właściwość ta jest szczególnie przydatna, w przypadku przesyłania danych 8*n*-bitowych, gdzie *n* – liczna naturalna, za pomocą interfejsu SPI mk, który wyłącznie może przesyłać dane 8-bitowe. Po wysłaniu 8 bitów następuje przerwa – zatrzymanie transmisji, w czasie której do interfejsu jest wprowadzana/czytana kolejna dana 8-bitowa, po tej przerwie ponownie uruchamia się interfejs, itd.

Polaryzacja i faza sygnału zegarowego muszą być zgodne zarówno dla układu master, jak i układów slave, aby transmisja pomiędzy nimi przebiegała prawidłowo.

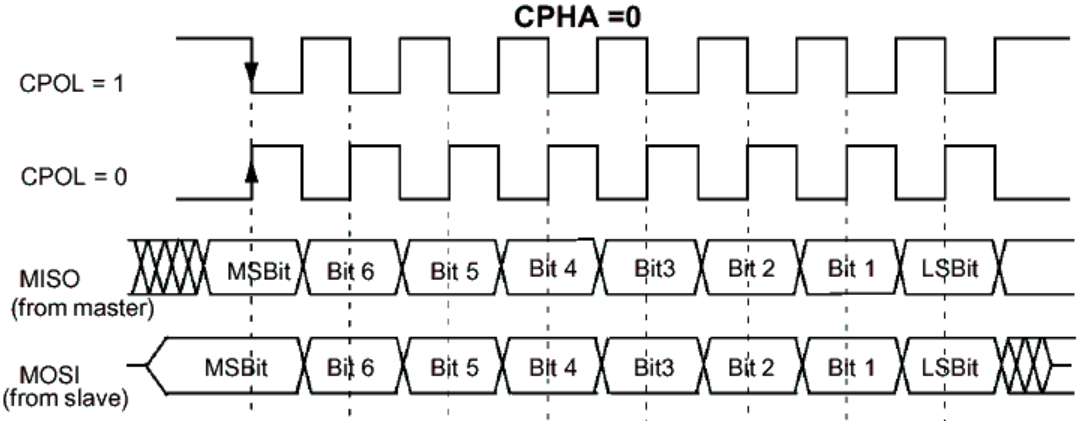
Polaryzacja sygnału zegarowego jest określona przez wartość logiczną sygnału zegara w stanie spoczynkowym (poza czasem transmisji):

- **CPOL=0** – sygnał zegara w stanie spoczynku jest w **stanie niskim** „Lo”,
- **CPOL=1** – sygnał zegara w stanie spoczynku jest w **stanie wysokim** „Hi”.

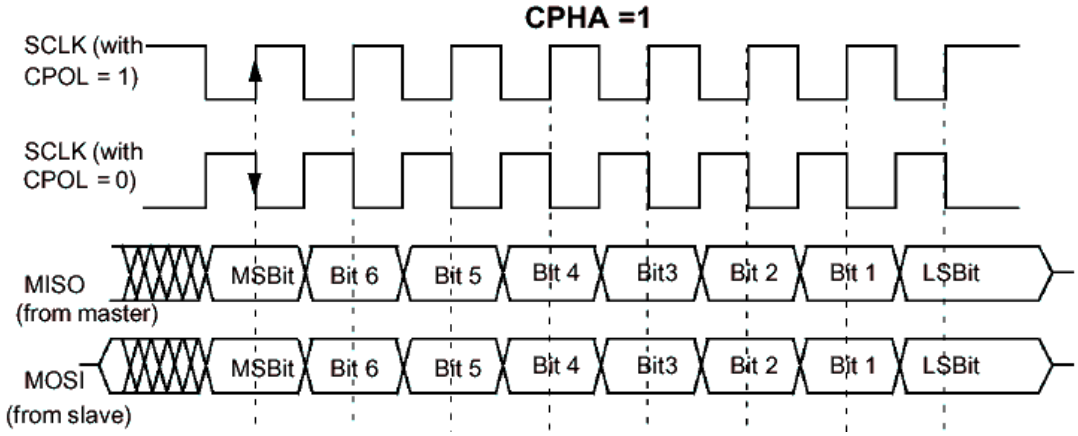
Faza sygnału zegarowego definiuje zależność pomiędzy zboczami sygnału zegarowego, a momentami próbkowania danych wejściowych; przesuwania zawartości rejestru (wysłania danych wyjściowych):

- **CPHA=0** – **pierwsze zbocze** sygnału zegarowego **próbkuje dane wejściowe**, drugie zbocze przesuwa dane w rejestrze – wyprowadza je z rejestru (dane są próbkowane, a następnie przesuwane i wysyłane),
- **CPHA=1** – **pierwsze zbocze** sygnału zegarowego **przesuwa dane** w rejestrze – wyprowadza je z rejestru, drugie zbocze próbkuje dane wejściowe (dane są przesuwane i wysyłane, a następnie próbkowane i wpisywane do rejestru).

Stąd można wyróżnić cztery warianty pracy interfejsu SPI (strobowania/przesuwania informacji): (1): CPHA=0 i CPOL=0, (2): CPHA=0 i CPOL=1, (3): CPHA=1 i CPOL=0, (4): CPHA=1 i CPOL=1. Najczęściej spotyka się pierwszy wariant, lecz pozostałe również są stosowane, dlatego we współczesnych mk istnieje możliwość programowego ustawienia polaryzacji i fazy sygnału zegarowego.



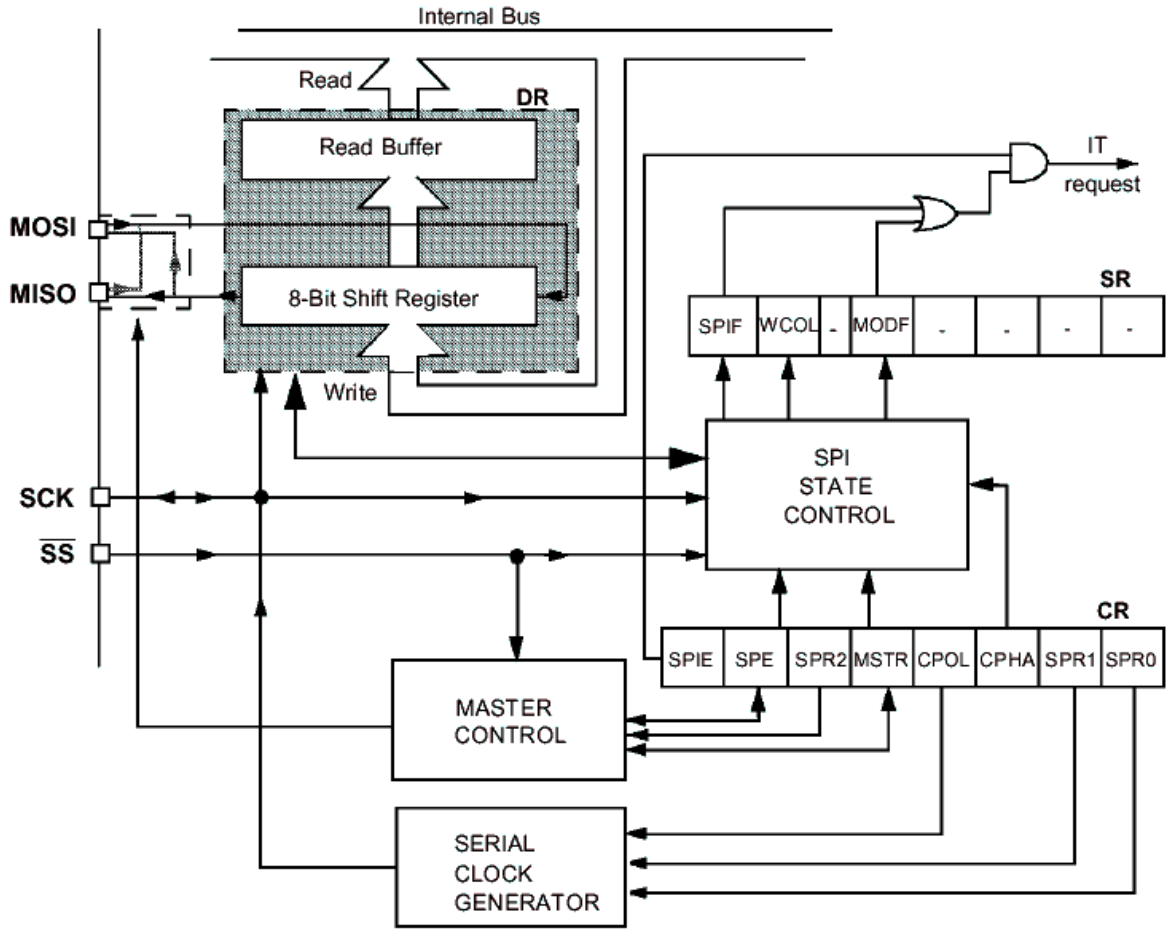
Rys. 2.50. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=0



Rys. 2.51. Przebiegi czasowe interfejsu SPI dla sygnału zegarowego o CPHA=1

Na rys. 2.50 przedstawiono (1) i (2) wariant pracy interfejsu SPI, dla sygnału zegarowego o fazie CPHA=0, natomiast na rys. 2.51 warianty (3) i (4) dla sygnału zegarowego o fazie CPHA=1.

Jako przykład interfejsu SPI wybrano układ z mk ST72215G, którego schemat blokowy pokazano na rys. 2.52.



Rys. 2.52. Schemat blokowy układu interfejsu SPI mk ST72215G

Z tym układem stowarzyszone są trzy rejestry:

- rejestr kontrolny, rej. CR,
- rejestr statusu, rej. SR,
- rejestr danych, rej. DR.

Rejestr danych DR służy do wpisywania danej, która ma zostać wysłana i do odczytu odebranej danej.

Rejestr statusu SR zawiera trzy flagi:

SPIF – informacja o zakończeniu transferu danej z rej. DR,

WCOL – bit kolizji ustawiany, gdy w trakcie wysyłania danej, wprowadzono nową daną do rej. DR,

MODF – jest ustawiany sprzętowo, gdy na linii SS pojawi się stan niski w trakcie pracy układu jako master. Bity te są zerowane programowo. Pierwsza i trzecia flaga mogą wywołać przerwanie, jeśli bit SPIE w rej. CR jest ustawiony.

Rejestr sterujący CR służy do konfigurowania interfejsu i sterowania jego pracą. Znaczenie bitów jest następujące:

SPIE – maska przerwania od interfejsu SPI,

SPE – podłączenie urządzenia do pinów portu równoległego, zerowany przez niski poziom sygnału na linii SS w trybie master,

SPR2, SPR1, SPR0 – określają częstotliwość sygnału taktującego interfejs w trybie master,

MSTR – ustawienie tego bitu wybiera tryb master (w przeciwnym przypadku układ jest w trybie slave), zerowany przez niski poziom sygnału na linii SS,

CPOL – wybór polaryzacji sygnału zegarowego, CPOL=0 lub CPOL=1,

CPHA – wybór fazy próbkowania, CPHA=0 lub CPHA=1.

Aby skonfigurować interfejs w trybie master należy wykonać następujące kroki:
- ustawić bity SPR1,SPR0 w rej. CR w celu wyboru prędkości transmisji,
- ustawić CPOL i CPHA w rej. CR aby wybrać właściwy wariant strobowania danych,
- na linii SS musimy zapewnić stan wysoki w czasie całej transmisji,
- bity MSTR i SPE muszą być ustawione.

Transmisja zaczyna się, gdy wpiszemy bajt do rej. DR. Jest on równolegle ładowany do rejestru szeregowego i następnie szeregowo wyprowadzany począwszy od najstarszego bitu. Gdy transfer jest kompletny, bit SPIF jest ustawiany sprzętowo i jeśli bit SPIE w rej. CR jest ustawiony oraz bit I w rejestrze kontrolnym przerwań CCR został wyzerowany, to następuje generacja przerwania.

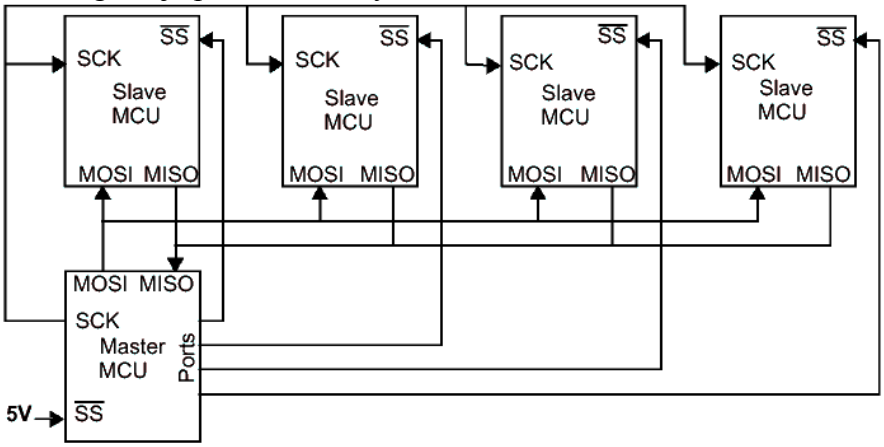
Kiedy ustawiany jest bit SPIF, to następuje jednoczesne przepisanie zawartości rejestru szeregowego do bufora. Bit ten jest zerowany poprzez odczyt/zapis rej. SR lub odczyt z rej. DR. Do rej. DR można wprowadzić kolejną daną wyłącznie w przypadku, gdy bit SPIF=0.

Dla trybu slave konieczne są następujące czynności:
- urządzenie peryferyjne musi być w tym samym trybie co mk (właściwie ustawiona CPOL i CPHA),
- na linii SS musimy zapewnić stan niski w czasie całej transmisji,
- należy wyzerować bit MSTR i ustawić bit SPE.

Transmisja zaczyna się, gdy urządzenie peryferyjne zaczyna generować sygnał zegarowy. Układ slave (mk) w takt tych impulsów wysyła z rejestru szeregowego bajt od MSB do LSB i jednocześnie kompletuje daną odbieraną. Dalej procedura przebiega według uprzedniej procedury.

Generalnie za pomocą interfejsu SPI można utworzyć dwie konfiguracje magistralowe:
- system z pojedynczym układem master (*single master system*),
- system z wieloma układami master (*multimaster system*).

Najczęściej spotykanym i najprostszym w konfiguracji jest pierwszy system, którego przykładową konfigurację pokazano na rys. 2.53.



Rys. 2.53. System z pojedynczym układem master oparty na interfejsie SPI

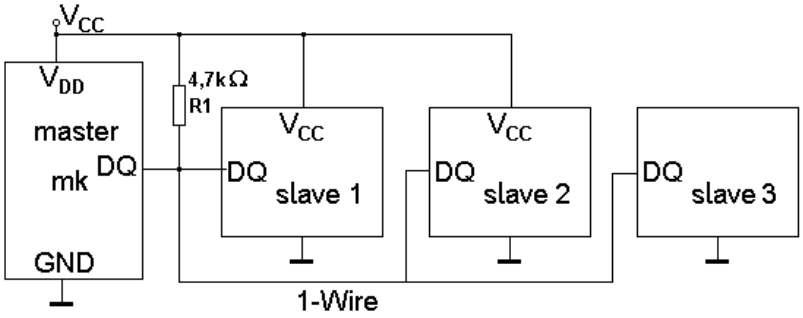
Układ master wybiera poszczególny układ slave korzystając z czterech linii portu równoległego. Dane urządzenie slave jest wybrane, gdy na jego wejściu SS pojawi się stan niski. W czasie transmisji, w celu uniknięcia kolizji na liniach interfejsu, tylko jeden układ slave może być aktywny.

2.6.5.3. Interfejs 1-Wire

Interfejs **1-Wire** został opracowany przez firmę Dallas Semiconductor (obecnie Maxim). jest on przeznaczony do przesyłania informacji pomiędzy układem nadrzędnym **master** (mk) i

układami podrzędnymi **slave** (np. termometry, układy identyfikacji, pamięci SRAM i EEPROM, programowalne klucze). Transmisja odbywa się w obu kierunkach z wykorzystaniem jednego przewodu sygnałowego (oprócz masy), który jednocześnie może być wykorzystany do zasilania układów do niego podłączonych. Dane przesyłane są z prędkością od bliskiej 0 do 16,3 kbps w trybie standard oraz do 115 kbps w trybie overdrive.

Każde z urządzeń podłączonych do magistrali musi mieć wyjście typu otwarty dren, a linia sygnałowa DQ jest połączona do zasilania przez rezystor podciągający o wartości około 5kΩ. Zatem w stanie bezczynności linia DQ jest w stanie wysokim. Magistrala nie ma ustalonego formatu danych. Sposób przesyłania informacji zależy od konfiguracji układów podrzędnych. Zawsze jednak jako pierwszy jest przesyłany bit najmniej znaczący.

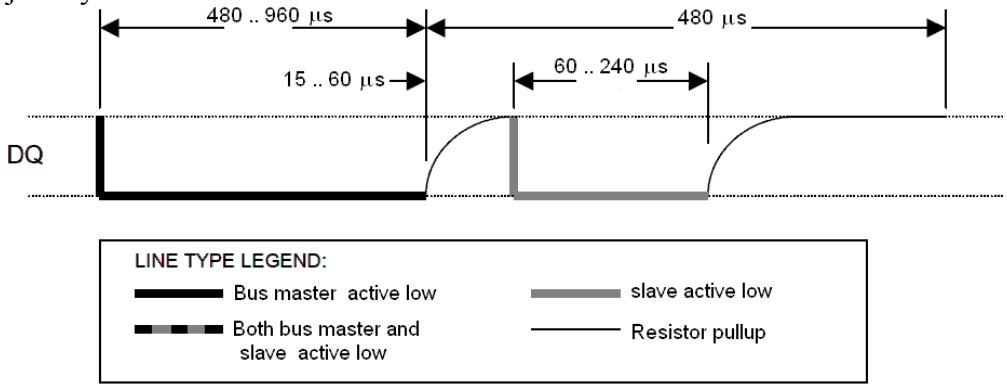


Rys. 2.54. Schemat połączeń interfejsem 1-Wire

Protokół wymiany danych poprzez magistralę 1-Wire składa się z czterech podstawowych sekwencji:

- inicjalizacji (zerowanie magistrali),
- wysłanie (zapis) zera,
- wysłanie (zapis) jedynki,
- odczyt bita.

Sekwencja inicjalizacji (rys. 2.55) rozpoczyna się wysłaniem na magistralę przez układ master (mk) impulsu zerującego o czasie trwania 480 .. 960µs. Po tym czasie układ master przechodzi w stan odbioru i na magistrali pojawia się stan wysoki. Po zidentyfikowaniu końca impulsu zerującego układ slave odczekuje 15 .. 60µs i wystawia na magistralę impuls obecności (*presence pulse*) o czasie trwania 60 .. 240µs. Sekwencja inicjalizacji umożliwia układowi master wykrycie podłączonych do niej układów slave. Po czasie koniecznym do pełnej inicjalizacji układów podrzędnych możliwe staje się przeprowadzenie normalnej transmisji danych.



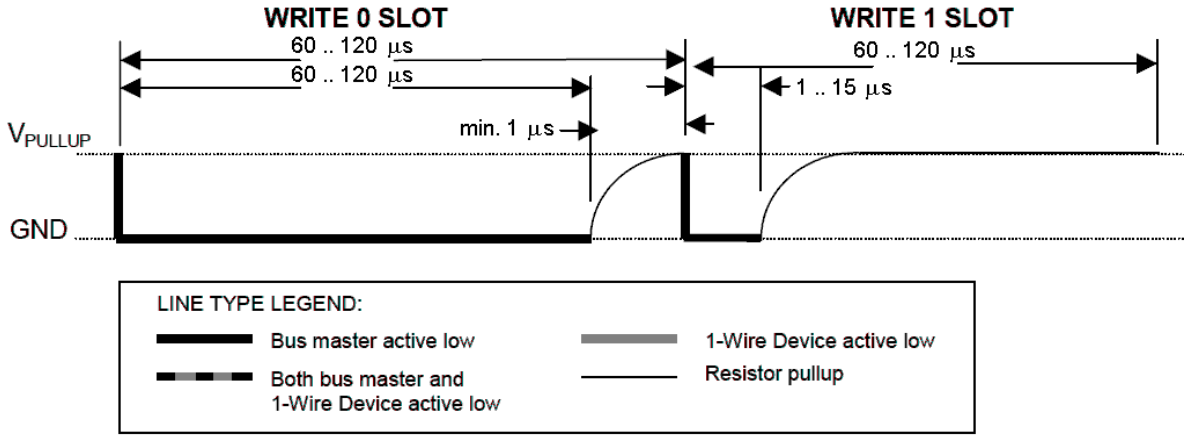
Rys. 2.55. Przebiegi sekwencji inicjalizacji dla magistrali 1-Wire

Transmisja danych do urządzenia podrzędnego polega na generowaniu serii impulsów o poziomie niskim o odpowiedniej długości, która definiuje stany logiczne „0” lub „1”. Długość sekwencji zapisu lub odczytu musi się zmieścić w szczelinie czasowej (*time slot*), która

wynosi 60 .. 120µs i jest inicjowana wymuszeniem przez układ master stanu niskiego na magistrali.

Nadanie logicznego „0” polega na wygenerowaniu impulsu o czasie trwania 60 .. 120µs, a następnie na zwolnieniu magistrali i odczekaniu minimum 1µs przed nadaniem następnego bita (rys. 2.56).

Dla logicznej „1” generowany impuls trwa 1 .. 15µs, natomiast wymagany czas bezczynności wynosi 60µs (rys. 2.56).



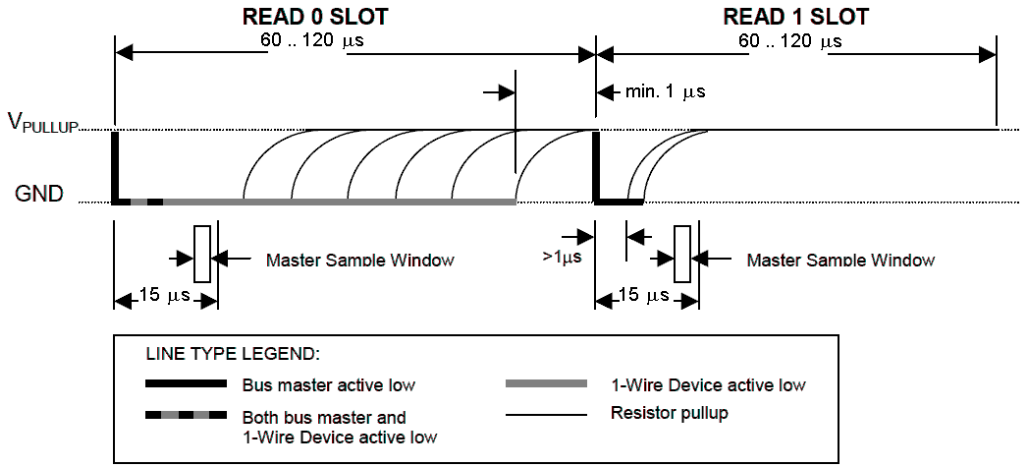
Rys. 2.56. Transmisja logicznych „0” i „1” na magistrali 1-Wire

Przy emulacji programowej interfejsu 1-Wire, na czas generowania impulsów (zwłaszcza dla „1”) warto wyłączać wszystkie przerwania mk, gdyż wystąpienie przerwania w trakcie generacji impulsu może doprowadzić do jego wydłużenia poza dopuszczalne granice, a co za tym idzie do zafałszowania transmisji.

Odczyt wartości bita przesyłanego przez układ podrzędny polega na generacji przez układ master impulsu o czasie trwania minimum 1µs (zazwyczaj stosuje się impulsy 3 .. 5µs), a następnie na zwolnieniu linii DQ i sprawdzeniu jej stanu logicznego przed upływem 15µs od rozpoczęcia sekwencji odczytu (rys. 2.57).

Jeżeli urządzenie podrzędne będzie transmitować logiczne „0”, wówczas generowany przez mk impuls zostanie przedłużony do czasu trwania co najmniej 15µs. Odczyt stanu napięcia na magistrali przed upływem tego czasu da w efekcie poziom niski, czyli transmitowane „0”. Ponieważ przedłużenie impulsu może trwać dłużej, to po odczytaniu stanu bita należy odczekać jeszcze nie krócej niż 46µs.

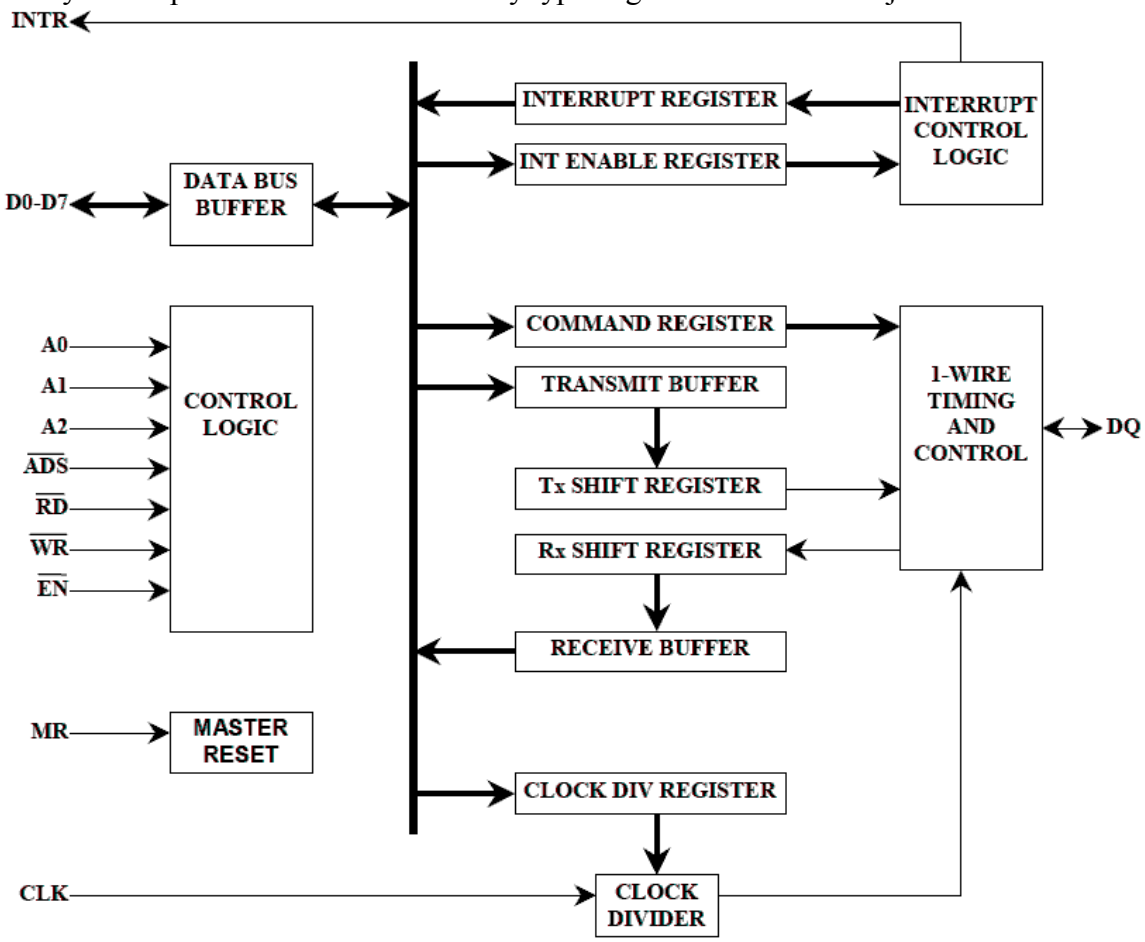
Jeżeli transmitowany będzie bit „1”, to wówczas urządzenie podrzędne nie przedłuży impulsu generowanego przez mk i odczyt stanu w tym samym czasie co przy transmisji „0” da w efekcie stan wysoki, czyli transmitowaną logiczną „1”.



Rys. 2.57. Odczyt bita przez układ master na magistrali 1-Wire

W większości przypadków interfejs 1-Wire implementuje się w mk w sposób programowy. Jednak, ze względu na rosnącą popularność interfejsu wynikającą z faktu, iż składa się on tylko z jednej linii sygnałowej, która może zasilać urządzenia podrzędne (o niewielkiej mocy pobieranej pojawiają się mikrokontrolery z kontrolerami tego interfejsu (np. DS80C400 firmy Dallas Semiconductor).

Na rys. 2.58 pokazano schemat blokowy typowego kontrolera interfejsu 1-Wire.



Rys. 2.58. Schemat blokowy kontrolera interfejsu 1-Wire

Linia DQ jest wejściem/wyjściem interfejsu 1-Wire. Pozostałe linie przenoszą sygnały wewnętrzne. Magistrala D0-D7 dostarcza dane do poszczególnych rejestrów kontrolera,

wybieranych za pomocą sygnałów A0-A2, które są zatrzaskiwane w układzie kontrolnym przez sygnał /ADS. Aktywny stan na linii /RD powoduje odczyt z wybranego rejestru, a aktywny sygnał /WR zapis do tego rejestru. Stan niski na linii /EN uaktywnia kontroler. Stan wysoki na linii MR resetuje kontroler zerując zawartości wszystkich rejestrów i flagi przerwań. Sygnał CLK dzielony przez dzielnik sterowany rejestrem Clock Div Register taktuje transmisją interfejsu 1-Wire. Na linii INTR pojawia się sygnał przerwania od kontrolera. Odczytując rejestr Interrupt Register identyfikowana jest przyczyna przerwania.

Interfejs steruje się za pomocą rejestru Command Register. Wysyłanie danej rozpoczyna się po wpisaniu jej do rejestru Transmit Buffer. Odebraną daną czyta się z rejestru Receive Buffer.

2.6.5.4. Interfejs I²C

Standard ten został opracowany przez firmę Philips. Jest stosowany w urządzeniach powszechnego użytku, zwłaszcza audio-video, telekomunikacji i systemach elektroniki przemysłowej. Transmisja danych odbywa się **szeregowo**, w **dwóch kierunkach**, przy użyciu **dwóch linii**. Jedną z nich **SCL** (*serial clock line*) przesyła się **impulsy zegarowe** synchronizujące transmisję, natomiast drugą **SDA** (*serial data line*) transmituje się w dwóch kierunkach **dane**.

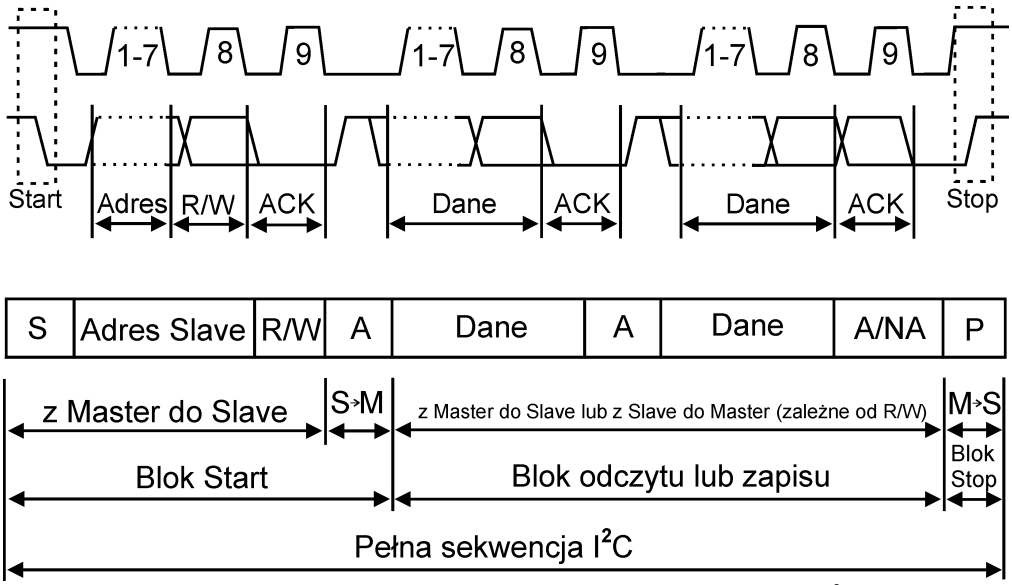
W transmisji interfejsem I²C uczestniczy układ **nadrzędny** (*master*) oraz jeden lub więcej układów **podrzędnych** (*slave*). Transmisja bloku danych jest inicjowana wówczas, gdy stan linii SDA zmieni się z wysokiego na niski, podczas gdy linia zegarowa SCL znajduje się w stanie wysokim. Sytuacja taka jest nazywana **warunkiem startowym (Start)**. Koniec transmisji ma miejsce wtedy, gdy linia SDA zmieni stan z niskiego na wysoki, podczas gdy linia zegara jest w stanie wysokim. Jest to warunek nazywany **Stop**. Stan linii danych może być zmieniany wówczas, gdy linia zegarowa znajduje się w stanie niskim. Od chwili wystąpienia warunku Start do chwili odległej o określony odstęp czasu od wystąpienia warunku Stop szyna jest zajęta.

Informacje są przesyłane bajtami, w postaci szeregowej, przy czym jako pierwszy wysyłany jest najbardziej znaczący bit. Po przesłaniu danych wysyłany jest **sygnał potwierdzenia** oznaczony symbolem A lub ACK (*acknowledge*). Dziewiąty impuls zegara towarzyszący potwierdzeniu jest generowany przez układ master. Na ten sygnał urządzenie nadające musi zwolnić linię SDA, wprowadzając swoje wyjście SDA w stan wysoki lub w stan wysokiej impedancji, natomiast odbiornik potwierdza odbiór sygnałem SDA=0. Liczba bajtów przesłanych w jednej **sesji, to jest między stanem Start i Stop**, jest nieograniczona.

Wysłanie bitu potwierdzenia przez odbiornik po odebraniu każdego bajtu danych jest obligatoryjne. Jeśli potwierdzenie nie zostaje wysłane, linia SDA musi znaleźć się w stanie wysokim lub w stanie wysokiej impedancji, by umożliwić układowi master wygenerowanie warunku Stop i zakończenie transmisji. Brak potwierdzenia może nastąpić na przykład wtedy, gdy odbiornik pracuje w czasie rzeczywistym i jest zajęty obsługa jakiegoś pilnego zadania. Jeśli odbiornikiem jest układ master, to po odebraniu ostatniego bajtu w sekwencji nie wysyła sygnału potwierdzenia NA (*not acknowledge*). Układ slave musi wtedy zwolnić linię SDA, by umożliwić układowi master wygenerowanie warunku Stop kończącego transmisję.

Na rys. 2.59 przedstawiono kompletny cykl transmisji w formacie standardu I²C. Po wygenerowaniu warunku Start układ master wysyła na szynę **adres układu slave**, z którym chce współpracować. **Adres składa się z siedmiu bitów**, po którym następuje **ósmy bit R/W** określający **kierunek transmisji**. Jeśli R/W=0, dane będą przesyłane z układu master do układu slave, natomiast gdy R/W=1, kierunek transmisji będzie odwrotny. Początkowa

sekwencja bitów przesyłanych w trakcie każdej sesji, złożona z bitu Startu, adresu urządzenia slave i bitu kierunku jest nazywana **blokiem Start**. Dalsza część transmisji polega na przesyłaniu kolejnych bajtów danych, zgodnie z protokołem pokazanym na rys. 2.59. Ostatnią fazą sesji jest wysłanie przez układ master warunku Stop.

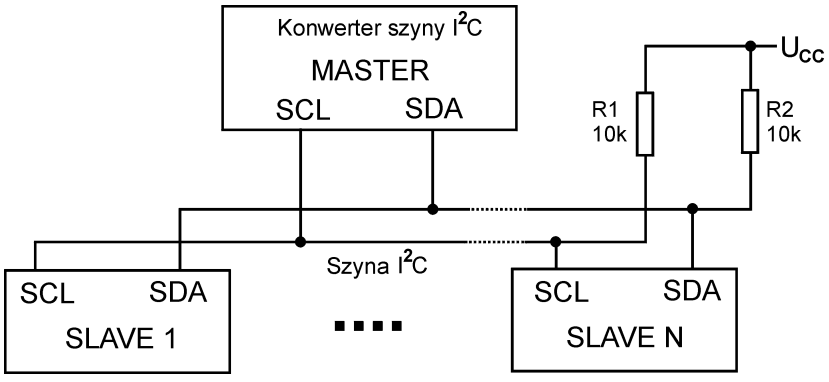


Rys. 2.59. Protokół transmisji szeregowej standardu I^2C

Pełna sesja składa się zatem **z bloku Start, przynajmniej jednego cyklu przesłania bajtu danych i bitu Stop**. Jeśli układ master zamierza w jednej sesji wysyłać informacje w kilku blokach do tego samego lub do różnych urządzeń, może nie kończyć sesji warunkiem Stop, lecz po przesłaniu bloku danych powtórzyć warunek Start z tym samym lub nowym adresem urządzenia slave.

Przemysłowa norma magistrali I^2C zakłada możliwość taktowania transmisji sygnałem zegarowym SCL o częstotliwości od **0 do 100kHz**. Wymaga się przy tym, by stan niski impulsów zegarowych trwał przynajmniej 4,7μs, natomiast stan wysoki był nie krótszy niż 4 μs. W razie potrzeby układ slave może spowolnić transmisję, utrzymując linię zegara SCL w stanie niskim po odebraniu bajtu danych od mk. Stan taki jest nazywany **stanem oczekiwania** i oznaczany symbolem **WAIT**. Realizacja stanów oczekiwania na magistrali wymaga, by układ master przed wysłaniem kolejnego bajtu przełączył linię SCL w stan wysoki lub w stan wysokiej impedancji, po czym odczytał stan na SCL.

Układy współpracujące z magistralą I^2C muszą być wyposażone w wyjścia z otwartym drenem dla linii SCL i SDA. Każda z tych linii jest połączona ze źródłem napięcia zasilającego U_{CC} przez rezystor 10kΩ.



Rys. 2.60. Połączenie urządzeń z magistralą I^2C

Do jednego układu master można przyłączyć dowolną liczbę układów slave, jednak pod warunkiem, że pojemność połączeń nie przekroczy maksymalnej wartości równej 400pF. Schemat połączeń przedstawiono na rys. 2.60.

Jako przykład układu interfejsu I^2C przytoczono układ MSSP (*Master Synchronous Serial Port*) mk PIC16F873, który może pracować jako interfejs SPI oraz interfejs I^2C. Układ ten w pełni implementuje wszystkie funkcje master i slave interfejsu I^2C. Zapewnia on sprzętową realizację przerwań na pojawienie się sekwencji Start i Stop. Umożliwia adresowanie 7-mio i 10-bitowe oraz może pracować przy dwóch prędkościach 100kHz i 400kHz.

Do obsługi interfejsu wykorzystuje się sześć rejestrów:

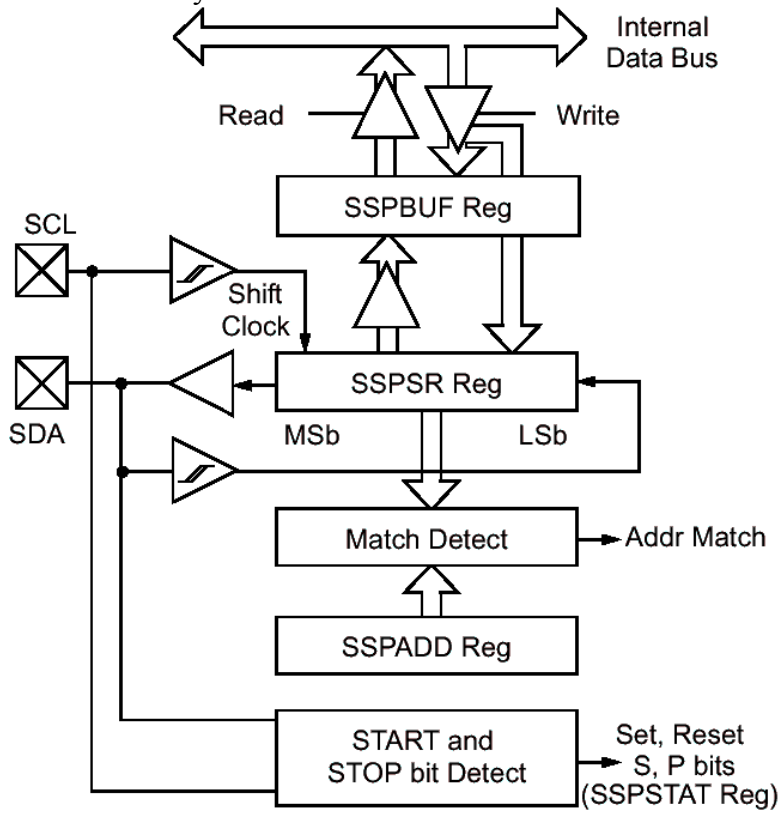
- rejestr sterujący SSPCON (*SSP Contorl Register*),
- rejestr sterujący drugi SSPCON2 (*SSP Contorl Register2*),
- rejestr statusowy SSPSTAT (*SSP Status Register*),
- bufor odbioru/wysyłania danej SSPBUF (*Serial Receive/Transmit Buffer*),
- rejestr przesuwny SSPSR (*SSP Shift Register*) –bezpośrednio nie dostępny,
- rejestr adresu SSPADD (*SSP Address Register*).

Rej. SSPCON służy do kontroli pracy interfejsu. Pozwala na wybór jednego z czterech trybów pracy interfejsu:

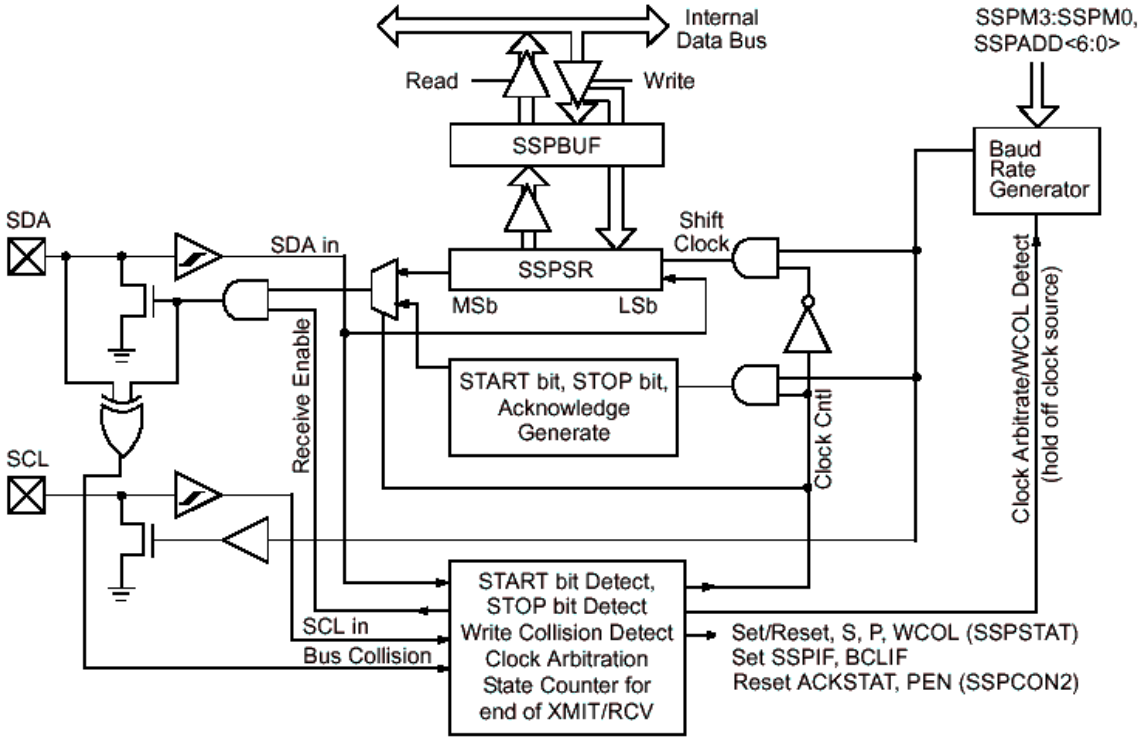
- tryb slave (adres 7-bitowy),
- tryb slave (adres 10-bitowy),
- tryb master, prędkość transmisji jest równa $f_{OSC}/(4 * (\text{adres SSPADD}+1))$,
- tryby firmware (aby zapewnić kompatybilność z innymi produktami średniej klasy).

Przed podłączeniem interfejsu I^2C do portu mk, właściwe piny muszą być skonfigurowane jako wejścia. Linii SCL i SDA muszą być poprzez zewnętrzne rezystory podwieszone do napięcia zasilania.

Schemat blokowy układu MSSP pracującego jako interfejs I^2C w trybie slave pokazano na rys. 2.61, a w trybie master na rys. 2.62.



Rys. 2.61. Schemat blokowy układu MSSP pracującego jako interfejs I^2C w trybie slave



Rys. 2.62. Schemat blokowy układu MSSP pracującego jako interfejs I²C w trybie master

Przedstawione układy w sposób sprzętowy w pełni realizują protokół interfejsu I²C zarówno dla trybu slave, jak i trybu master.

2.6.5.5. Interfejs CAN

Interfejs CAN stosuje się głównie w motoryzacji, w lotnictwie, a także w sterownikach przemysłowych. Założenia projektowe CAN koncentrują się na trzech właściwościach systemu transmisyjnego:

- **szybkości przesyłania danych** łączem szeregowym,
- **odporności transmisji** na zakłócenia elektromagnetyczne,
- możliwości **dzielenia wspólnych danych** przez rozproszone mk realizujące cząstkowe funkcje sterowania.

Szyna CAN jest **asynchroniczną** szyną transmisji szeregowej z tylko jedną **linią transmisyjną**. Ma strukturę **otwartą**, tzn. może być rozszerzana o nowe **węzły** (odbiorniki/nadajniki), oraz **liniową**, czyli nie zawiera pętli. Minimalna konfiguracja CAN składa się z dwóch węzłów. Liczba węzłów w trakcie pracy szyny może się zmieniać bez wpływu na działanie szyny. Właściwość ta oznacza możliwość dynamicznego konfigurowania węzłów, na przykład wyłączenia węzłów, które w wyniku działania procedur diagnostycznych zostały uznane za niesprawne.

Poszczególne węzły są przyłączone do szyny na zasadzie funkcji „**zwarty iloczyn**” (*wired-AND*). Przy braku sterowania szyna znajduje się w stanie nazywanym **recesywnym** R (recessive), któremu odpowiada stan logiczny „1”. Stan ten może być zmieniony na stan logiczny „0”, jeśli choć jeden węzeł wymusi na szynie poziom **dominujący** D (dominant).

Najpopularniejszy sposób sprzętowej realizacji szyny jest użycie **pary skręconych przewodów** oznaczonych symbolami **CAN_H** i **CAN_L**. Linie te przyłącza się do węzłów bezpośrednio lub za pośrednictwem buforów/nadajników/odbiorników. Po obu stronach linii

dołącza się rezystory. Maksymalna szybkość transmisji szyną CAN wynosi 1Mbps przy długości linii nie przekraczającej 40m. Im dłuższa linia tym prędkość transmisji mniejsza, np. dla długości przewodów 1km prędkość transmisji wynosi już 40kbps.

Dane przesyłane szyną są transmitowane metodą NRZ (*Non Return to Zero*). Adresy odbiorników (**identyfikatory**) są przesyłane jako integralna część przekazu. Ponieważ interfejs składa się wyłącznie z jednej linii konieczny jest arbitraż szyny. Wykonuje się go metodą CSMA/CD z opcją NDA (*carrier sense multiple access / collision detection with non-destructive arbitration*).

Specyfika standardu CAN przewiduje **11-bitowe identyfikatory**, czyli umożliwia współpracę 2048 węzłów. Najnowsza specyfikacja 2.0B (*extended CAN*) posiada identyfikator 29-bitowy, co daję teoretycznie możliwość adresowania 536 milionów współpracujących ze sobą urządzeń.

2.6.5.6. Interfejs USB

Interfejs ten został opracowany jako **uniwersalny interfejs szeregowy** wbudowany w architekturę komputerów PC. Jego przeznaczeniem jest współpraca komputerów z urządzeniami: przemysłowymi, urządzeniami powszechnego użytku oraz integracja z sieciami telekomunikacyjnymi. Prędkość transferu danych wynosi 1,5 lub 12 Mbit/s dla standardu USB 1.1, natomiast dla standardu USB 2.0 jest ona równa 480 Mbit/s.

Przyjęto następującą topologię systemu z interfejsem USB, jeden **host** (komputer zarządzający) połączony z pierwszym **Hubem**, do którego można podłączyć na zasadzie drzewa kolejne Huby lub urządzenia. Taka topologia zapewnia możliwość współpracy różnych urządzeń poprzez wyspecjalizowane Huby, które z kolei współpracują ze sobą tworząc jednolity system, wygodny do użycia przez końcowego użytkownika dzięki :

- prostemu i znormalizowanemu okablowaniu,
- izolacji elektrycznej,
- samoidentyfikacji urządzeń,
- automatycznej konfiguracji ich sterowników,
- automatycznej rejestracji w systemie,
- niskich kosztów sprzętu i okablowania.

Podstawowym elementem systemu są **Huby**. Hub posiada port do połączenia z hostem (**Upstream Port**) oraz do siedmiu portów do podłączenia urządzeń lub innych Hubów (**Downstream Port**), z których każdy może być indywidualnie konfigurowany i kontrolowany poprzez komputer host. Huby składają się z dwóch części:

- Hub Controller,
- Hub Repeater.

Hub Repeater jest protokołem kontrolującym połączenie pomiędzy Upstrem Port i Downstream Port. Dostarcza on rejestr służący do komunikacji z hostem.

Wszystkie elementy systemu połączone są magistralą składającą się z czterech linii:

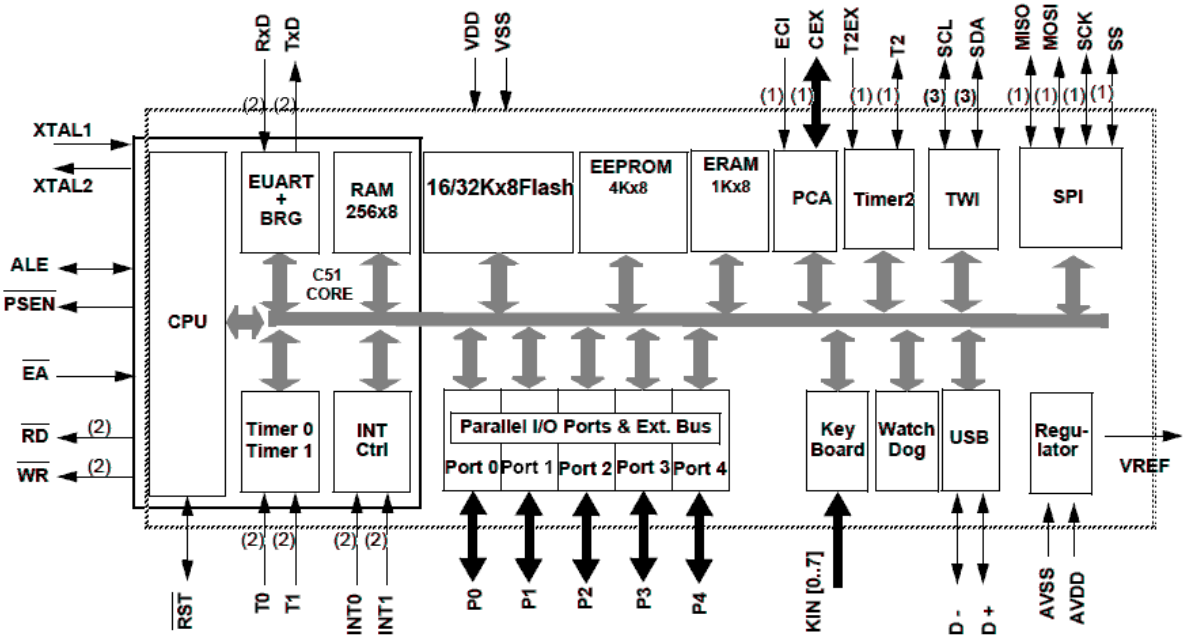
- **Vbus**, przewód zasilania +5[V]
- **D+**, przewód symetrycznej skrętki sygnałowej
- **D-**, przewód symetrycznej skrętki sygnałowej
- **GND**, przewód masy zasilania.

Typowym kablem sygnałowym jest skrętka ekranowana o impedancji charakterystycznej 90Ω +-15[%] i maksymalnej długości 5m. Maksymalne opóźnienie sygnału między punktami końcowymi, tzn. hostem i urządzeniem musi być mniejsze od 70ns.

Do transmisji danych interfejsem USB wykorzystuje się kodowanie danych **NRZI** (*Non-Return to Zero Inver on ones*). Kodowanie to polega na braku zmiany poziomu dla jedynki logicznej i zmianie poziomu dla zera logicznego.

Komunikacja między urządzeniami przebiega w następujący sposób: urządzenie fizyczne zostaje zidentyfikowane przez oprogramowanie typu klient, zainstalowane na komputerze (host). Następnie zostaje zainicjowane połączenie pomiędzy urządzeniem a hostem, umożliwiające przesyłanie komunikatów i danych. Dane odebrane z urządzenia zostają sformatowane zgodnie z wymaganiami protokołu USB i przekazane do hosta dzięki współpracy oprogramowania systemu USB, oprogramowania hosta oraz współpracującego sprzętu.

Przykładem mk wyposażonego w interfejs USB może być mk AT89C5131A-M firmy Atmel, którego schemat blokowy przedstawiono na rys. 2.63.



Rys. 2.63. Schemat blokowy mk AT89C5131A-M firmy Atmel z interfejsem USB 2.0

Mk ten zbudowany jest na bazie dwóch rdzeni 8051 (sześć cykli zegarowych na instrukcję). Interfejs USB może pracować w standardzie 1.1 i 2.0 z pełną prędkością.

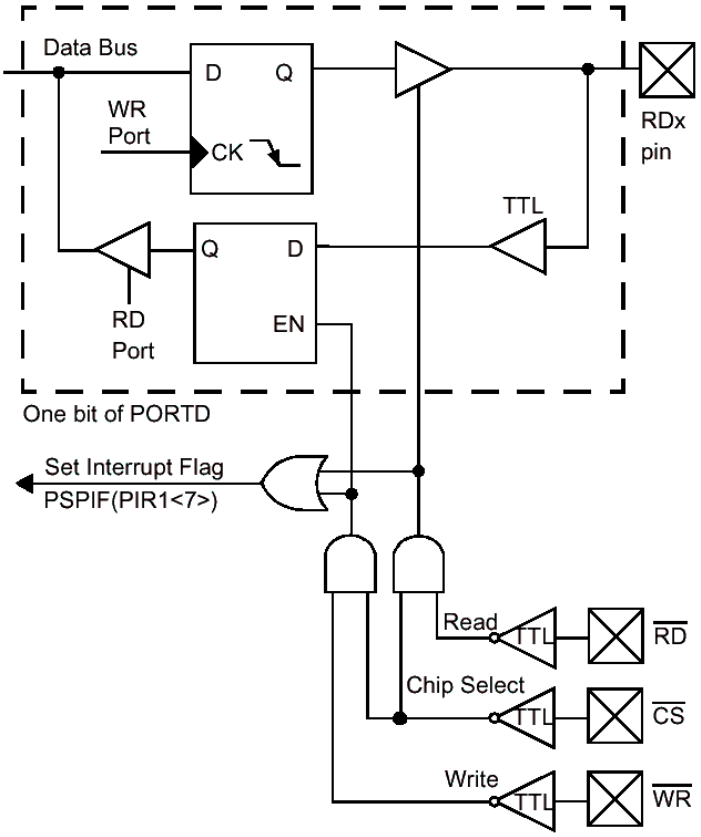
2.6.6. Interfejs równoległy typu paraller slave port

Interfejs typu **paraller slave port (PSP)** pozwala na **bezpośrednie podłączenie** portu mk do **magistrali danych**, sterowanej przez inny mk (nadrzędny) i na **asynchroniczny zapis** (za pomocą sygnału WR - *write*) lub **odczyt** (za pomocą sygnału RD - *read*) bajtu z tego portu. Czyli port równoległy w trybie PSP pracuje jak 8-bitowy rejestr, do którego wpisuje się bajt za pomocą sygnału WR (wpis ten może wygenerować przerwanie w mk), lub można z niego czytać bajt uaktywniając sygnał RD.

Przykładem takiego układu może być PSP w mk PIC16F877. W trybie tym pracuje port D, jeśli ustawi się bit PSPMODE w rej. TRISE. Wówczas trzy linie portu E mają następujące znaczenie: DR – RE0, WR- RE1, CS – RE2. Muszą być one ustawione jako wejścia cyfrowe.

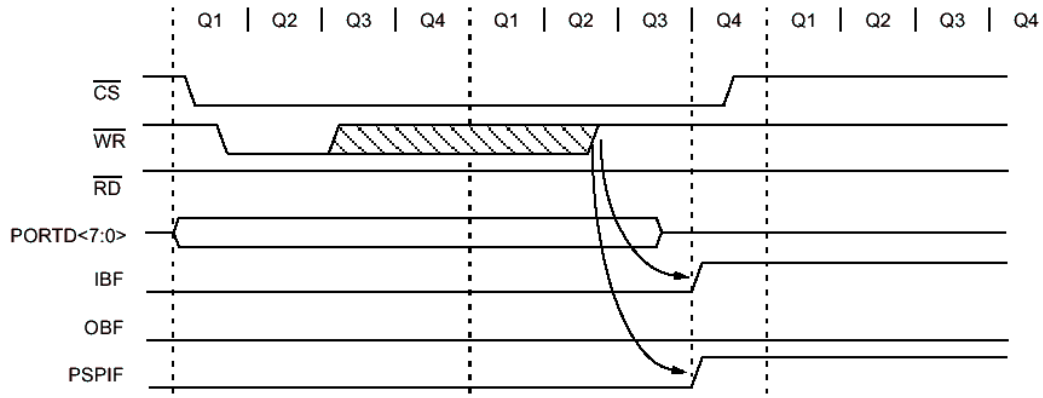
Interfejs PSP składa się z dwóch 8-bitowych rejestrów zatrzaskowych, jednego dla danych wejściowych, drugiego dla danych wyjściowych (rys. 2.64). Użytkownik wpisuje dane do

rejestru zatrzaskowego danych portu D i czyta dane rejestru zatrzaskowego pinów portu D (oba te rejestry są pod wspólnym adresem – adres portu danych rejestru D). W tym trybie, rejestr kierunku portu D jest ignorowany, ponieważ kierunkiem przepływu danych steruje zewnętrzne urządzenie.



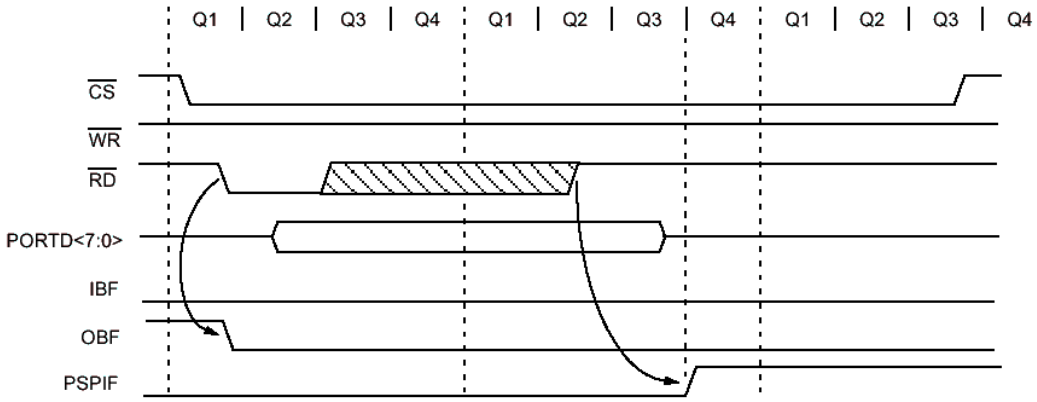
Rys. 2.64. Schemat blokowy interfejsu PSP mk PIC16F877

Zapis do interfejsu PSP jest możliwy, gdy na obu liniach WR i CS pojawi się stan niski. Jeśli te linie przejdą w stan wysoki bit IBF (*Input Buffer Full*) w rej. TRISE jest ustawiany w cyklu Q4 zegara, aby zasygnalizować, że zapis do rej. danych został dokonany (rys. 2.65). W tym samym cyklu zegara Q4 jest również ustawiana flaga przerwania PSPIF. Bit IBF można wyzerować wyłącznie przez odczyt zawartości rejestru danych portu D. Bit IBOV (*Input Buffer Overflow*) w rej. TRISE zostaje ustawiony, gdy nastąpi próba zapisu bajtu do rejestru danych, który zawiera nie odczytaną jeszcze daną.



Rys. 2.65. Zapis bajtu do rejestru danych interfejsu PSP

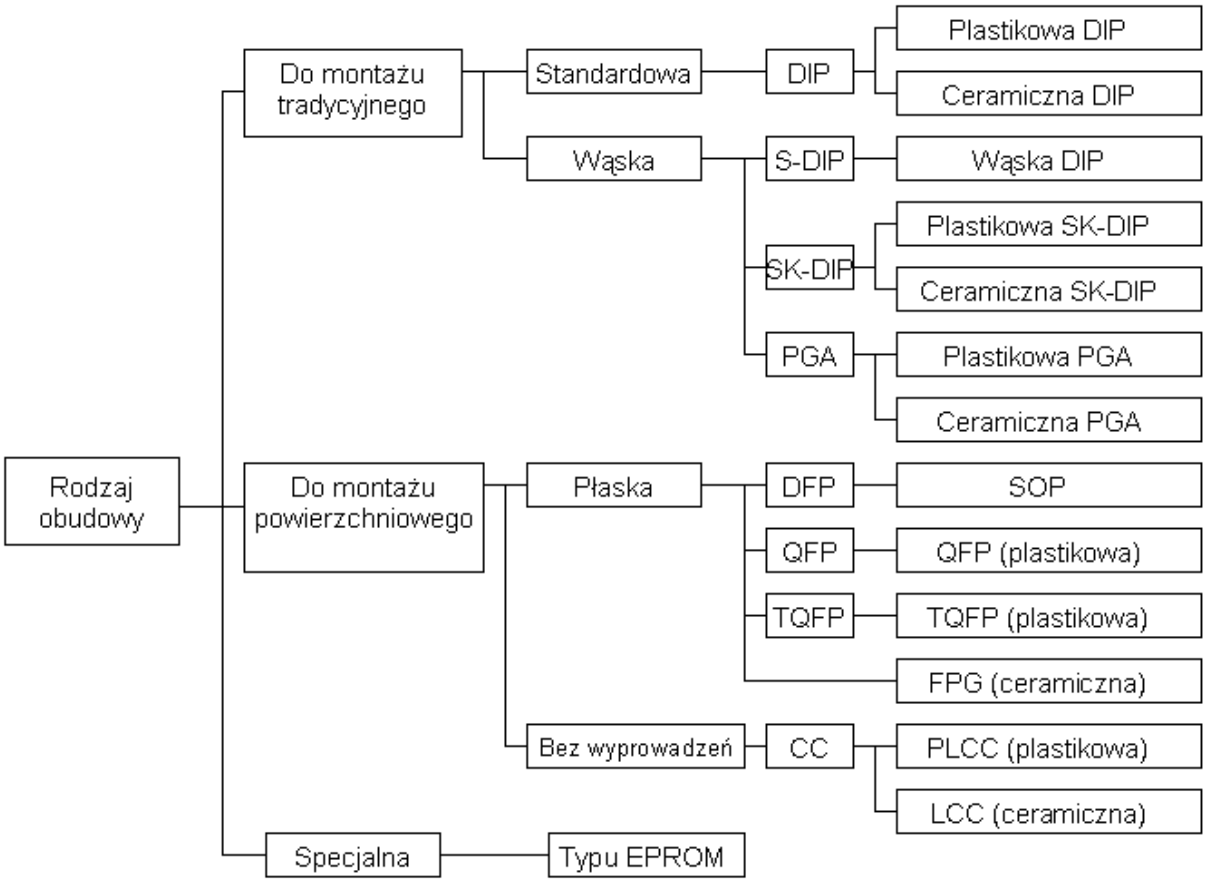
Odczyt z interfejsu PSP nastąpi, gdy na obu liniach RD i CS pojawi się stan niski. Flaga OBF (*Output Buffer Full*) w rej. TRISE jest natychmiast zerowana, aby wskazać, iż następuje odczyt bajtu z rejestru danych portu D przez zewnętrzne urządzenie (rys. 2.66). Kiedy na obu liniach pojawi się stan wysoki, w czwartym cyklu zegara Q4 zostaje ustawiona flaga przerwania PSPIF, informująca o zakończeniu odczytu. Bit OBF jest równy zeru dopóki nie nastąpi wpis bajtu do rejestru danych przez oprogramowanie użytkownika.



Rys. 2.66. Odczyt bajtu z rejestru danych interfejsu PSP

2.7. Typy obudów mk

Scalone struktury mk są zamykane w obudowach rozmaitych typów, zarówno plastikowych, jak i ceramicznych. Ich klasyfikację podano na rys. 2.67.



Rys. 2.67. Klasyfikacja obudów mk

Wyróżniono następujące typy obudów:

DIP – (*dual in-line package*) dwurzędowe płaskie; najstarszy, tradycyjny typ obudowy zarówno do wlutowywania jak i montażu w podstawkach; wytwarzane jako plastikowe i ceramiczne,

S-DIP – (*shrink dual in-line package*) dwurzędowe płaskie; o zmniejszonych gabarytach, plastikowe,

SK-DIP - (*skinny dual in-line package*) dwurzędowe płaskie, miniaturowe,

PGA – (*pin grid array*), ze szpilkowymi wyprowadzeniami rozmieszczonymi na obwodzie prostokątnej (lub kwadratowej) matrycy,

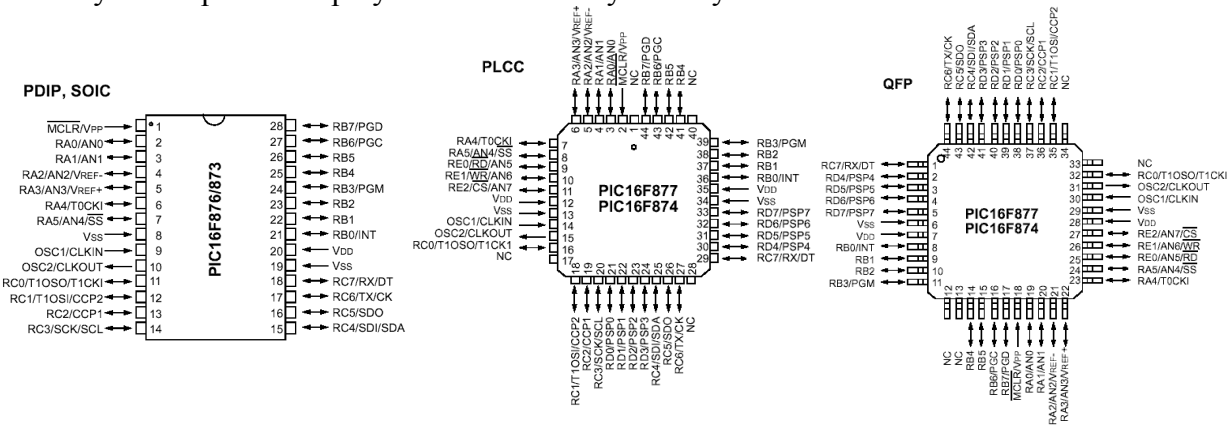
DFP – (*dual flat package*) do montażu powierzchniowego, z wyprowadzeniami po dwóch stronach obudowy,

QFP – (*qual flat package*) do montażu powierzchniowego, z wyprowadzeniami z czterech stron obudowy,

TQFP – (*thin qad flat package*) do montażu powierzchniowego, miniaturowa wersja QFP,

CC – (*chip carier*) obudowy z kontaktami na krawędzi obudowy, produkowane w dwóch podstawowych odmianach: PLCC (*plastic leaded chip carier*) oraz LCC (*leadless chip carier*).

Na rys. 2.68 pokazano przykładowe obudowy rodziny mk PIC16F87x.



Rys. 2.68. Przykładowe typy obudów mk PIC16F87x

2.8. Programowanie mk

Programowanie mk jest ściśle związane z architekturą samego procesora i jego sprzętowego otoczenia. Tworzenie oprogramowania dla mse opartych na mk (i nie tylko) i ukierunkowanych na zadania pomiarowo-sterujące oraz komunikacyjne określa się w literaturze mianem **programowania zagnieżdżonego** (*embeded programming*). Można wyróżnić następujące cechy programów zagnieżdżonych:

- Program **jednoznacznie ustala funkcję mse**, tzn. użytkownik ma możliwość zmiany funkcji systemu zazwyczaj tylko w niewielkim zakresie przewidzianym przez program użytkowy. Ta właśnie cecha określana jest jako „**zagnieżdżenie**" programu.
- Działanie programu **musi spełniać określone wymagania czasowe** dotyczące przekraczania maksymalnego czasu reakcji na określone zdarzenia zewnętrzne oraz realizacji określonych zadań programowych w nieprzekraczalnym czasie. Ta cecha określana jest jako praca programu w „**czasie rzeczywistym**".
- Są to programy **działające na specyficznych zasobach sprzętowych** warunkowanych ukierunkowaniem budowy sprzętowej mse na konkretne zadanie.

Cykl tworzenia oprogramowania dla mk jest zbliżony do cyklu tworzenia oprogramowania dla komputerów osobistych. Cykl ten obejmuje trzy fazy:

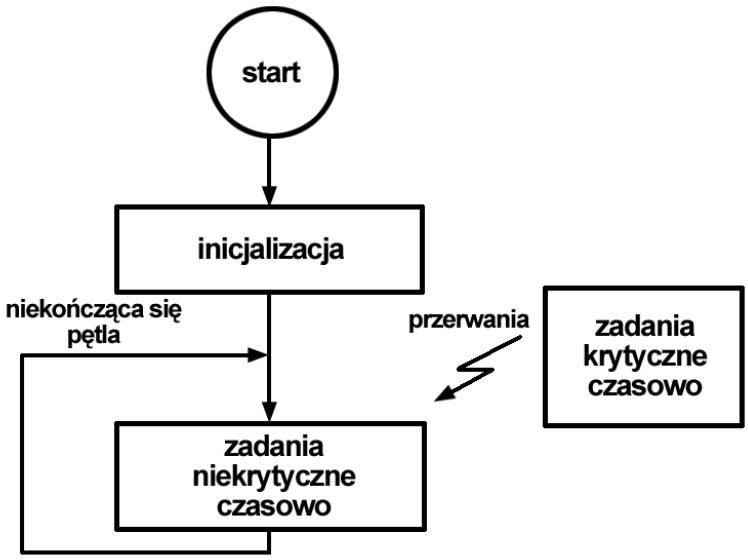
- **napisanie kodu źródłowego**,
- **przetłumaczenie kodu źródłowego na kod maszynowy danego mk**,
- **uruchomienie programu w systemie docelowym (mse)**.

Faza pierwsza i druga wykonywana jest najczęściej przy zastosowaniu standardowego komputera osobistego. Komputer osobisty w tej funkcji określany jest jako **system rozwojowy** (*development system*). Faza trzecia realizowana jest na rzeczywistym mse zawierającym mk. System ten określany jest jako **system docelowy** (*target system*).

Tworzenie oprogramowania mk wymaga od programisty dysponowania odpowiednimi programami, tzw. programami narzędziowymi (*software tools*). Obecnie programy te łączone są w jeden pakiet. Umożliwia to realizację wszystkich etapów cyklu rozwoju oprogramowania dla mk w ramach jednego programu. Środowisko takie określa się skrótem **IDE** (*Integrated Development Enviroment*). Zawiera ono między innymi specjalizowany edytor tekstowy, kompilator, linker, symulator programowy, debuger, oprogramowanie sterujące programatorem sprzętowym służące do programowania mk.

Przed rozpoczęciem pisania kodu źródłowego (w edytorze tekstowym) należy wybrać określony język programowania. W praktyce stosuje się dwa rodzaje języków programowania mk**: język asemblera** lub jeden z języków wysokiego poziomu – **język C**/C++ albo Java.

Ważna uwaga: poprawnie napisany program użytkownika musi działać w niekończącej się pętli. Jc po włączeniu napięcia zasilania bez przerwy pobiera z pamięci i wykonuje kolejne instrukcje. Nie może się zatem skończyć po wykonaniu ostatniego rozkazu. Zatem program użytkownika ma strukturę jak pokazano na rys. 2.69.



Rys. 2.69. Struktura programu użytkownika na mk

Po włączeniu napięcia zasilania w programie użytkownika musi się odbyć jednorazowa inicjalizacja, np. ustalenie trybu pracy urządzeń peryferyjnych. Właściwy program użytkownika jest realizowany częściowo w niekończącej się pętli głównej, a częściowo w obsługach przerwań.

2.8.1. Programowanie w języku asemblera

Asembler jest **językiem niższego poziomu**. Operuje on bezpośrednio na liście rozkazów (instrukcji) danej jc. Każdy rozkaz z tej listy ma przyporządkowaną krótką nazwę zwaną **mnemonikiem**. Za ich pomocą programista tworzy kod źródłowy. Następnie program tłumaczący tłumaczy mnemoniki na odpowiadający im kod maszynowy. Stąd określenie „język niższego poziomu". Program tłumaczący z języka asemblera określany jest jako **asembler** (*assembler*).

Programowanie w języku asemblera stosuje się w trzech przypadkach:

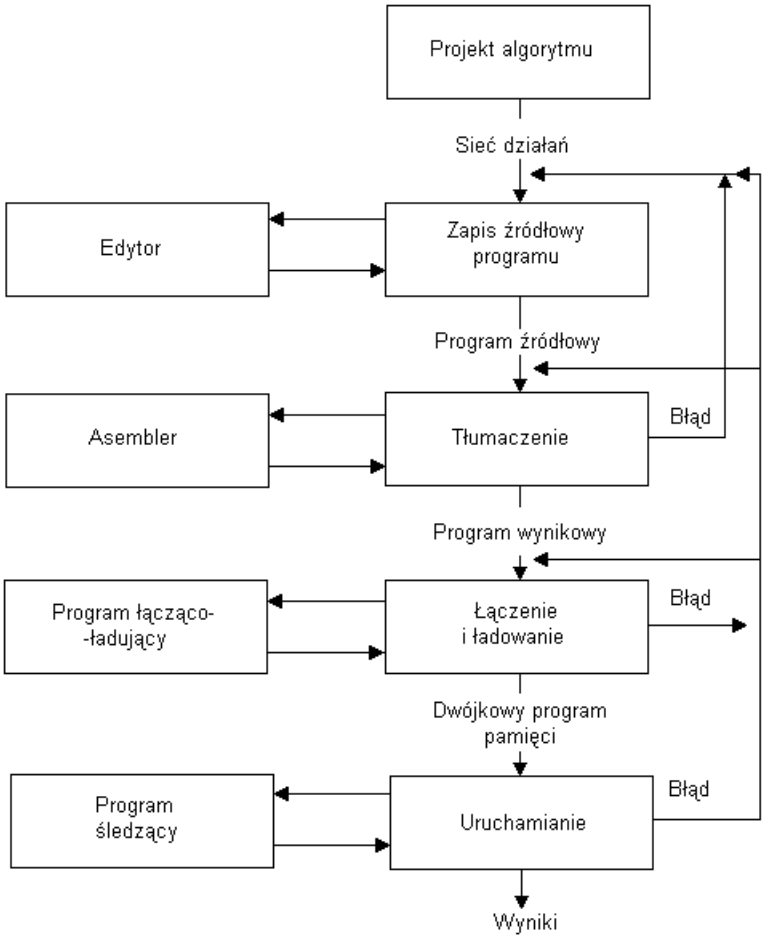
- gdy tworzona aplikacja jest bardzo prosta i zapis w języku asemblera nie sprawia kłopotu,
- gdy nie dysponujemy kompilatorem C/C++ lub jego koszt byłby niewspółmierny do skali projektu,
- gdy wymagania wobec szybkości działania programu i minimalizacji zajmowanej pamięci są szczególnie ostre.

Do zalet programowania w języku asemblera należy zaliczyć:

- Możliwość pełnego panowania nad zasobami systemu. Programista ma nieograniczony dostęp do wszystkich bloków na poziomie rejestrów i pojedynczych bitów. Żadna z funkcji systemu nie jest ukryta, w szczególności możliwe jest dowolne, nawet nietypowe oddziaływanie na obszar stosu i mechanizm przerwań.
- Swobodne dysponowanie obszarem pamięci.
- Efektywny program wynikowy, szybszy i zajmujący na ogół znacznie mniej pamięci niż równoważny program zapisany w języku wysokiego poziomu.
- Możliwość swobodnego wyboru formatu danych i precyzji obliczeń. Programista może samodzielnie definiować wielobajtowe struktury danych do obliczeń o praktycznie dowolnej dokładności.
- Możliwość dopasowania algorytmu do indywidualnych cech architektury mk oraz optymalizacji programu wynikowego.

Jednak zastosowanie asemblera ma też wady: programowanie jest żmudne i zajmuje więcej czasu niż przy użyciu języków wysokiego poziomu. Program jest zatem droższy, trudniejszy do modyfikowania, bardziej podatny na błędy i mniej czytelny, nawet jeśli zastosowano obszerne komentarze.

Projektowanie oprogramowania w języku asemblera przebiega według schematu przedstawionego na rys. 2.70. Do edycji programów źródłowych (standardowe rozszerzenie ASM) zapisanych w plikach tekstowych ASCII można użyć dowolnego, prostego edytora tekstowego. Następnie program źródłowy jest poddawany tłumaczeniu (czyli translacji lub asemblacji). W rezultacie najczęściej powstają dwa pliki. Pierwszy z nich to program wynikowy (*object code*) zapisany w pliku **OBJ**. Jest on **przemieszczalny (relokowalny** – *relocatable code*), tzn. może być przypisany do ustalonego później obszaru adresowego pamięci. Drugim jest plik tekstowy z raportem translacji i wydrukiem programu źródłowego z towarzyszącym mu kodem wynikowym (zazwyczaj w postaci szesnastkowej). Raport z translacji (*listing file* – stąd najczęstsze rozszerzenie LST) podaje ponadto numery i adresy kolejnych linii (instrukcji) programu, adresy zadeklarowanych zmiennych, zestawienie użytych nazw symbolicznych oraz ewentualne komunikaty o błędach.



Rys. 2.70. Cykl projektowania w języku asemblera

Po uzyskaniu programu wynikowego wolnego od błędów translacji następuje faza łączenia programu (konsolidacji). W fazie tej wykonuje się połączenie modułów wynikowych (jeśli program został napisany w kilku plikach OBJ), dołączenie modułów bibliotecznych (jeśli zostały użyte w programie) oraz uzgodnienie adresów modułów. W rezultacie powstaje **nie przemieszczalny** program binarny (*non-relocatable code*), który można załadować do pamięci mk i uruchomić. Format tego pliku zależy od typu procesora i użytego środowiska IDE. Jednym z najpopularniejszych formatów jest format **HEX** opracowany przez firmę Intel (Intel-HEX). Przechowuje on kody instrukcji i dane (stałe) w postaci ciągu rekordów zapisanych w kodzie ASCII. Każdy z rekordów ma stały format:

```
:llaaaatt[dd....]cc
```

przy czym:

: - znak dwukropka, identyfikator początku każdego nowego rekordu,

`ll` – długość pola danych (`dd....`) wyrażona w bajtach,

`aaaa` – 16-bitowy adres miejsca pamięci, do którego mają być załadowane dane zapisane w rekordzie,

`tt` – kod typu rekordu: 00 oznacza rekord danych, 01 oznacza rekord terminalny (kończący plik HEX),

`dd` – ciąg bajtów danych,

`cc` – suma kontrolna obliczana jako różnica: 00H-(suma wszystkich bajtów rekordu) mod 256.

Podczas ładowania programu z pliku HEX do pamięci systemu wykonywane jest rozpakowanie kolejnych rekordów, sprawdzanie sumy kontrolnej oraz przesłanie pola danych pod wskazany adres. Po załadowaniu program gotowy jest do uruchomienia. Testowanie programu jest wykonywane pod nadzorem **programu śledzącego** (symulator software'owy). W początkowej fazie symulacji korzysta się z trybu pracy krokowej lub systemu **pułapek** (punktów zatrzymań) ulokowanych w zadanych miejscach programu. Śledzenie pracy programu polega na sprawdzeniu zawartości rejestrów i wybranych komórek pamięci w kolejnych fazach wykonywania programu. Jeśli zawartości te zmieniają się zgodnie z założeniami projektowymi, a przyłączone do systemu urządzenia peryferyjne pracują również zgodnie z oczekiwaniami, uruchamianie programu jest zakończone.

W ostatnim kroku program zostaje załadowany do pamięci mk znajdującego się w mse za pomocą odpowiedniego programatora (zakłada się, że mk posiada pamięć typu FLASH). Na tym etapie prac mse jest traktowany jako prototyp. Następnie odbywa się testowanie prototypu: sprawdzanie jego parametrów elektrycznych i czasowych. Gdy mse zachowuje się prawidłowo, to można uznać projekt za zakończony.

2.8.2. Programowanie w językach wyższego poziomu

Języki programowania wyższego poziomu są w zasadzie niezależne od listy instrukcji konkretnej jc. Programista operuje rozkazami wykonującymi bardziej złożone operacje niż język asemblera. Program tłumaczący z języka programowania wyższego rzędu na kod maszynowy nosi nazwę **kompilatora** (*compiler*).

W języku asemblera każda linia programu tłumaczona jest na odpowiedni rozkaz jc danego mk. W języku wyższego poziomu każda linia programu tłumaczona jest na kilka, kilkanaście, a czasami nawet na kilkadziesiąt rozkazów asemblera. Ten ostatni przypadek ma miejsce np. wówczas, gdy w programie zastosowane są działania na liczbach zmiennoprzecinkowych, zaś jc nie wspiera operacji zmiennoprzecinkowych.

Różne jc mają różne listy rozkazów, zatem program napisany w języku asemblera jest związany na stałe z daną jc. Program napisany w języku programowania wyższego poziomu jest w niewielkim stopniu związany z daną jc. W celu uruchomienia tego programu w mse zawierającym inny mk wystarczą na ogół niewielkie modyfikacje kodu źródłowego, a następnie przetłumaczenie tego kodu na kod maszynowy przy zastosowaniu narzędzi programowych właściwych dla danego mk. Zmiany w kodzie muszą być znacznie większe, jeżeli obydwa mk wykorzystują inne układy peryferyjne.

Dominującym językiem wyższego poziomu jest **język C**. Obecnie obowiązuje jego **standard ANSI C**. Jednak większość kompilatorów zawiera dodatkowe rozszerzenia tego języka związane ze specyficznymi wymaganiami przy programowaniu mk. Najczęściej są to elementy:

- wprowadzone są nowe typy zmiennych, np. **zmienne bitowe**,
- przy deklaracji zmiennych wprowadzane są mechanizmy umożliwiające umiejscowienie (**alokację**) zmiennej w określonym miejscu przestrzeni adresowej,
- wprowadzone są nowe słowa kluczowe języka C (np. słowo ***interrupt*** jest używane przy deklaracji funkcji wywoływanych przy wystąpieniu przerwania).

Zalecane jest aby pisząc program w języku C na mk stosować następujące rozwiązania:

- fragmenty kodu źródłowego związane z obsługą specyficznych układów peryferyjnych umieszczać w wydzielonych modułach,
- typy zmiennych deklarować nie bezpośrednio przy deklaracji zmiennych, ale poprzez słowo kluczowe *typdef*,

- alokacji zmiennych nie dokonuje się bezpośrednio przy deklaracji zmiennych, lecz za pomocą dyrektyw preprocesora, głównie przez użycie tzw. modelu pamięci,
- alokacja funkcji służących do obsługi przerwań (tzw. rozprowadzanie przerwań) odbywa się w wydzielonym zbiorze, często napisanym w języku asemblera.

Tworzenie programu w języku C przebiega według tego samego schematu, co cykl projektowy programu w języku asemblera przedstawiony na rys. 2.70. Pliki źródłowe programu zwykle oznacza się rozszerzeniem C.

2.8.3. Uruchamianie programu mk

Uruchamianie programów jest procesem eliminacji błędów i wprowadzania zmian do programu użytkowego w celu uzyskania bezbłędnej i zgodnej z założeniami pracy systemu w danym zastosowaniu.

Uruchamianie prototypowych mse z mk jest trudnym procesem. Wynika to z dwóch podstawowych przyczyn:

- mk mają wiele wbudowanych bloków funkcjonalnych (pamięci i układów peryferyjnych), do których dostęp w czasie pracy systemu jest utrudniony lub niemożliwy,
- sterowniki z wbudowanymi mk są przeznaczone do pracy w czasie rzeczywistym. Testowanie systemu w czasie rzeczywistym, we współpracy z rzeczywistym otoczeniem, bez naruszenia relacji czasowych sygnałów, wymaga narzędzi diagnostycznych, które nie zakłócają pracy systemu.

Zatem opracowano wiele metod stosowanych w tym celu. Metody te dzielą się na dwie grupy:

- **bez wykorzystania systemu docelowego**,
- **z wykorzystaniem systemu docelowego**.

W ramach pierwszej grupy stosowana jest jedna metoda. Jest to wykorzystanie **symulatora** programowego mk (*simularot*). Symulatory umożliwiają symulację pracy jc, a bardziej rozbudowane także układów peryferyjnych mk. Zaletą tej metody jest to, iż nie trzeba dysponować fizycznym docelowym systemem do testowania oprogramowania oraz to, że na podstawie wyników testów (np. czasu wykonywania programu) można wprowadzać odpowiednie korekty do koncepcji budowy sprzętowej systemu docelowego jeszcze przed jego wykonaniem. Natomiast wadą tej metody jest to, że poprawne działanie programu z symulatorem nie daje całkowitej gwarancji bezbłędnej pracy w rzeczywistym systemie. Zatem ostateczna weryfikacja poprawności napisanego programu może nastąpić tylko podczas jego wykonywania w systemie docelowym.

W skład drugiej grupy metod wchodzą:

- **metoda prób i błędów** (polega na obserwacji działania programu w mse i jego korekcji na podstawie tych obserwacji, aż do uzyskania prawidłowego działania mk),
- zastosowanie **emulatora pamięci programu** (w miejsce zewnętrznej pamięci programu umieszcza się emulator pamięci podłączony najczęściej poprzez RS232 z komputerem PC) – jest to nadal metoda prób i błędów z tym, iż programowanie pamięci programu jest znacznie szybsze,
- zastosowanie **monitorów programowych** (*monitors*) i **programów śledzących** (*debuggers*) – są najczęściej stosowane. Monitory są instalowane w pamięci programu mk i kontrolują wykonywanie właściwego programu użytkowego oraz komunikują się z systemem rozwojowym poprzez złącze szeregowe. Natomiast debugery pracują w przyłączonych do systemu komputerach PC,

- zastosowanie **emulatora sprzętowego** mk (ICE – *in-circuit emulators*) – polega to na umieszczeniu, na czas uruchamiania programu, w podstawce na mk sondy połączonej ze specjalnym układem sprzętowym, który emuluje działanie mk. Emulator wiernie odtwarza wszystkie właściwości mk łącznie z jego wszystkimi układami peryferyjnymi oraz pamięcią,
- wykorzystanie **specjalnych zasobów wewnętrznych mk** – niektóre mk zwłaszcza 32-bitowe posiadają specjalne zasoby sprzętowe przeznaczone do wspierania procesu uruchomiania programu. Zasoby te oferują w przybliżeniu wszystkie te możliwości co emulator sprzętowy, między innymi ustawienie pułapek oraz pracę krokową. Zasoby te komunikują się przez dedykowane wyprowadzenia mk. Są one dostępne wyłącznie na etapie uruchamiania programu, zatem nie są wykorzystywane przez normalny program użytkowy.

2.8.4. Sposoby programowania mk z pamiecią FLASH

Obecnie prawie wszystkie produkowane mk są dostępne w wersji z pamięcią programu typu FLASH. Mk z pamięcią EPROM oraz typu OTP są grupą wymierającą. Zaletą pamięci FLASH (pamięć błyskowa) jest możliwość wielokrotnego jej programowania bezpośrednio w systemie docelowym (bez konieczności wyciągania mk z podstawki) ISP (*In-System Programmable*).

Możemy wyróżnić dwa typy programowania mk:
- tryb równoległy,
- tryb szeregowy.

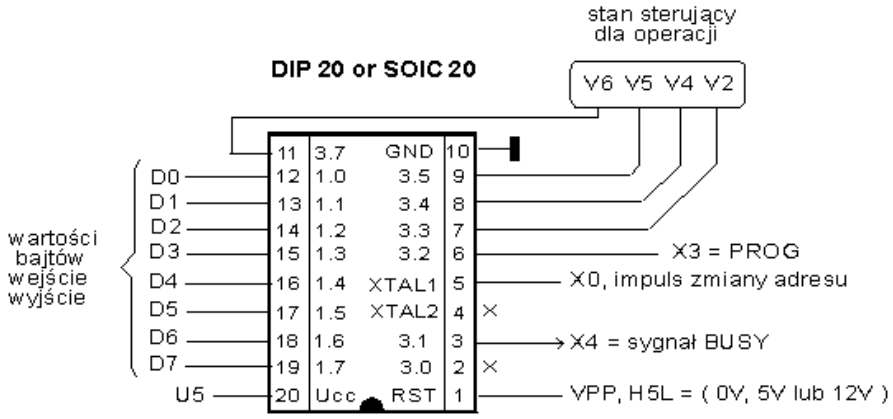
2.8.4.1. Tryb programowania równoległego

Wadą trybu równoległego programowania jest konieczność wyciagnięcia mk z mse i umieszczenie go w programatorze. Zaletą, zwłaszcza dla mk firmy Atmel, jest to, iż mk nie dające się z jakichkolwiek przyczyn zaprogramować w trybie ISP, dadzą się programować w trybie równoległym.

Tryb ten zostanie przedstawiony na przykładzie mk rodziny AT89Cx051. Bazuje on na algorytmie równoległego programowania AT051.

Charakterystyka algorytmu AT051:
- U5=5V tylko, Vpp=5V lub Vpp=12V.
- Adres dla odczytu sygnatury to: 0, 1, 2.
- Algorytm zawiera:
  - operację odczytu pamięci,
  - operację kasowania układu,
  - operację programowania i weryfikacji pamięci,
  - programowania bitów security, dwa bity tylko programowanie - bez odczytu.

Na rys. 2.71 pokazano konfigurację połączeń przy użyciu algorytmu AT051. Z rys. widać, iż dane programujące pamięć FLASH ładowane i, podczas weryfikacji lub odczytu, odczytywane są za pośrednictwem portu P1. Sygnały sterujące podłączone są do portu P3.



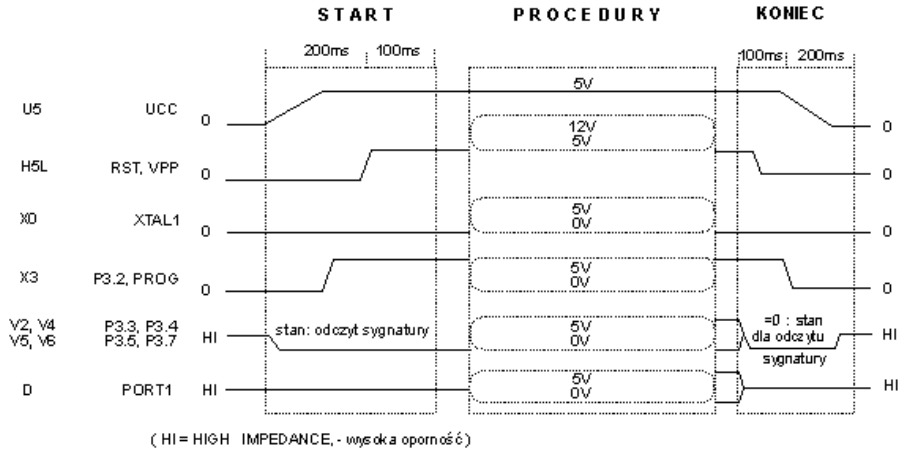
Rys. 2.71. Konfiguracja połączeń przy użyciu algorytmu AT051

Sygnały V6, V5, V4, V2 (linie 7, 5, 4 i 3 portu P3) służą do wyboru trybu pracy, co pokazano w tabeli 2.6. Linia PROG (P3.2) powoduje zatrzaśniecie (wprowadzenie/wyprowadzenie) danych na porcie P1. Impuls na linii X0 inkrementuje adres wskazujący na komórkę pamięci FLASH. Gotowość mikrokontrolera do przyjęcia kolejnego rozkazu jest wskazywana przez linię BUSY (P3.1).

Tabela 2.6. Tryby pracy mikrokontrolera dla algorytmu AT051

Mode		RST/VPP	P3.2/$\overline{\text{PROG}}$	P3.3	P3.4	P3.5	P3.7
Write Code Data		12V		L	H	H	H
Read Code Data		H	H	L	L	H	H
Write Lock	Bit - 1	12V		H	H	H	H
	Bit - 2	12V		H	H	L	L
Chip Erase		12V	(2)	H	L	L	L
Read Signature Byte		H	H	L	L	L	L

Na rys. 2.72 pokazano schemat ogólny algorytmu. Składa się on z trzech cykli START, PROCEDURY i KONIEC. W pierwszym cyklu następuje odczyt sygnatury z linii V6, V5, V4, V2 i wejście w dany tryb pracy (tabela 2.6). Następnie (cykl PROCEDURY) wykonywana jest właściwa operacja, po czym, o ile chcemy skończyć programować, następuje zakończenie programowania mikrokontrolera.



Rys. 2.72. Schemat ogólny algorytmu AT051

2.8.4.2. Tryb programowania szeregowego (ISP)

Jak wspomniano, zaletą programowania szeregowego jest możliwość programowania mk bezpośrednio w systemie docelowym (ISP).

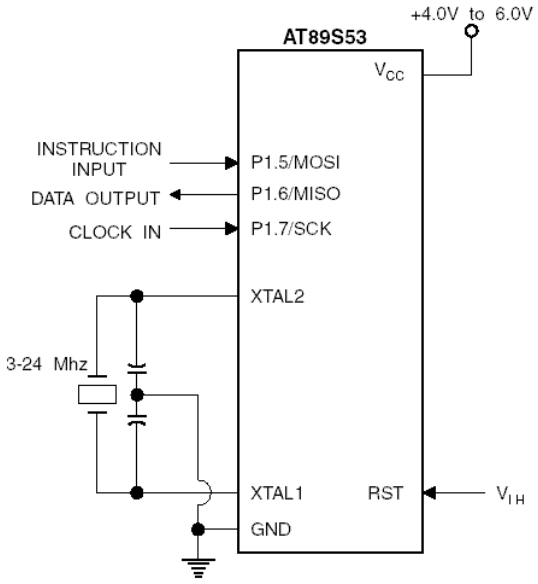
Generalnie procedura programowania szeregowego mk jest następująca:

- Wprowadzenie mk w tryb programowania. Odbywa się to podczas resetu mk poprzez podanie na odpowiednie końcówki określonej sekwencji sygnałów lub odpowiednich poziomów napięć.
- Następnie poprzez dedykowany do programowania interfejs szeregowy wprowadza się szeregowo odpowiednie rozkazy sterujące procesem programowania pamięci. Część rozkazów dotyczy zapisu i odczytu danych do/ze specjalnych rejestrów konfiguracyjnych zawartych w pamięci FLASH odpowiedzialnych za konfigurację mk (tryb pracy oscylatora, zabezpieczenia przed odczytem, itd.). Rejestry te dostępne są wyłącznie w trybie programowania mk. Pozostałe rozkazy są związane z zapisem i odczytem danych z pamięci FLASH oraz z jej kasowaniem.
- Na zakończenie programowania następuje odpowiednia sekwencja sygnałów lub zdjęcie odpowiednich napięć powodujące wyjście z trybu programowania.

Możemy wyróżnić następujące sposoby programowania ISP:

- z wykorzystaniem interfejsu SPI (mk rodziny AVR i ST7),
- korzystając z interfejsu I^2C (mk rodziny ST5),
- opierając się na interfejsie UART i wbudowanym programie ładującym zawartym w pamięci mk (ADuC8xx).
- bazując na dedykowanym interfejsie szeregowym (PIC16F8xx),

Pierwszy sposób, oparty na interfejsie SPI, zostanie przedstawiony na przykładzie mk AT89S53 firmy Atmel. Konfigurację pinów dla trybu programowania szeregowego pokazano na rys. 2.73.



Rys. 2.73. Sygnały w trybie programowania szeregowego

Jak widać z rys. do programowania wykorzystywany jest interfejs SPI (linie MOSI, MISO, SCK) oraz sygnał RST, który wraz z komendą umożliwiającą programowanie uprowadza mk w tryb programowania. W tym trybie mk musi być taktowany zewnętrznym sygnałem zegarowym podłączonym do linii XTAL1 o częstotliwości od 3 do 24MHz lub musi być do

mk podłączony oscylator kwarcowy. Sygnał zegarowy SCK taktujący transmisję musi być przynajmniej 40 razy wolniejszy niż częstotliwość na XTAL1.

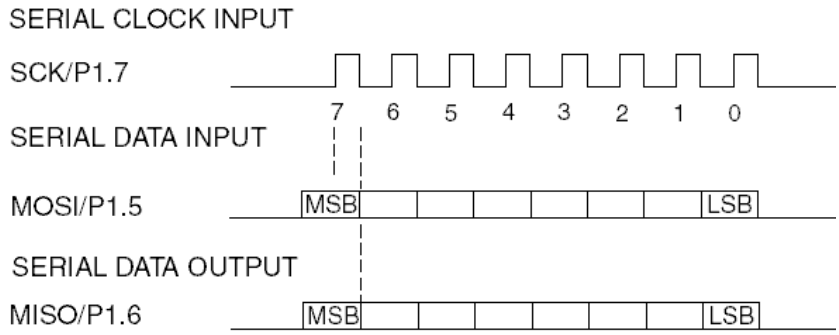
Aby zaprogramować lub zweryfikować układ należy wykonać następującą sekwencję:

1. Włączyć napięcie zasilania i na linię RST podać stan „Hi".
2. Umożliwić szeregowe programowanie przez wysłanie komendy Programing Enable za pomocą interfejsu SPI.
3. Wysłać odpowiednią komendę programującą i odczekać czas potrzebny na zaprogramowanie lub komendę związaną z odczytem (lista wszystkich komend zawarta jest w tabeli 2.7).
4. Na zakończenie programowania lub weryfikacji układu ustawić sygnał RST w stan niski, aby przejść w normalny tryb pracy.

Tabela 2.7. Lista instrukcji programowania mk AT89S53

Instruction	Input Format			Operation
	Byte 1	Byte 2	Byte 3	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	Enable serial programming interface after RST goes high.
Chip Erase	1010 1100	xxxx x100	xxxx xxxx	Chip erase the 12K memory array.
Read Code Memory	A12 A11 A10 A9 A8 A13 01	low addr	xxxx xxxx	Read data from Code memory array at the selected address. The 6 MSBs of the first byte are the high order address bits. The low order address bits are in the second byte. Data are available at pin MISO during the third byte.
Write Code Memory	A12 A11 A10 A9 A8 A13 10	low addr	data in	Write data to Code memory location at selected address. The address bits are the 6 MSBs of the first byte together with the second byte.
Write Lock Bits	1010 1100	LB1 LB2 LB3 x x111	xxxx xxxx	Write lock bits. Set LB1, LB2 or LB3 = "0" to program lock bits.

Notes: 1. $\overline{\text{DATA}}$ polling is used to indicate the end of a write cycle which typically takes less than 10 ms at 2.7V.
2. "x" = don't care.



Rys. 2.74. Przebiegi czasowe dla interfejsu SPI w trybie programowania

Kolejny sposób bazujący na na interfejsie UART i wbudowanym programie ładującym zawartym w pamięci mk zostanie zilustrowany na przykładzie mk ADuC812 firmy Analog Devices. Jest to mk z rdzeniem 80C52.

Posiada on wbudowany (umieszczony w pamięci ROM) program ładujący (loader) pozwalający na programowanie mk przez interfejs szeregowy UART przy prędkości 9600 bodów. Np. w celu zaprogramowania mikrokontrolera ADuC812 należy wykonać następujące kroki:

- na liniach portu P0 nie powinno być poziomów niskich,
- sygnały RX i TX powinny być podłączone do interfejsu UART mk,
- na linię EA mk podać stan niski,

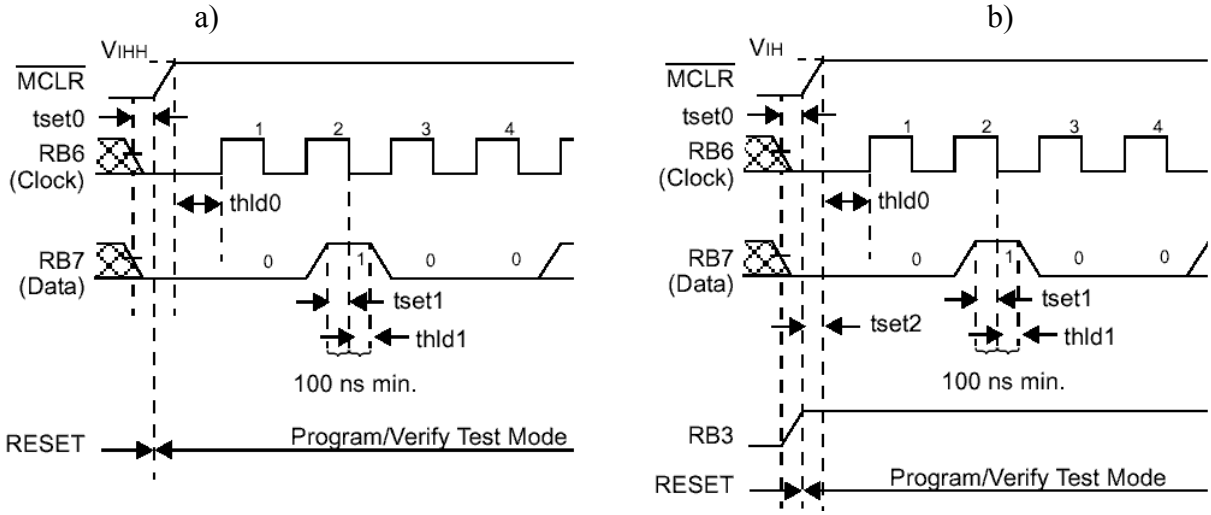
- zewrzeć linię PSEN przez rezystor 1kΩ do masy (poziom zero na tym pinie w trakcie resetu wprowadza mk w tryb programowania),
- zresetować mk,
- uruchomić program wsd.exe (program bezpłatnego środowiska IDE Devices MicroConventer QuickStart Development System firmy Analog Devices). Powinien pojawić się komunikat w okienku programu wsd.exe informujący, iż nawiązano połączenie z mk ADuC812,
- załadować plik *.hex do pamięci FLASH mk,
- po poprawnym załadowaniu pliku, co zostaje potwierdzone odpowiednim komunikatem, na linii EA ustawić stan wysoki – mk wykonuje instrukcje z wewnętrznej pamięci programu,
- nacisnąć klawisz RESET na płytce laboratoryjnej,
- program wsd.exe można już zamknąć,
- po kolejnym resecie mk wykonuje już program zawarty w pamięci FLASH.

Ostatni sposób programowania szeregowego zostanie pokazany na przykładzie mk PIC16F873. Tryb programowania tego mk pozwala na wpisanie do pamięci programu danych oraz programu użytkownika, danych pod specjalne lokacje typu ID i do rejestru konfiguracyjnego. Istnieją dwie metody programowania mk:

- normalne programowanie – Algorytm 1 (dla napięć zasilania od 2,2V do 5,5V i napięcia programowania V_{PP} wynoszącego 13V ±0,5V),
- programowanie niskonapięciowe – Algorytm 2 (dla napięć zasilania od 4,5V do 5,5V), nazywane w skrócie LVP (*Low Voltage ICSP*, gdzie *ICSP – In-Circuit Serial Programmable*).

W trybie programowania wykorzystuje się trzy linie portu B: RB3 (PGM) linia wejściowa wykorzystywana w trybie LVP, RB6 (CLOCK) wejście zegara, RB7 (DATA) wejście/wyjście danych oraz pin resetu MCLR służący do wyboru trybu programowania, V_{DD} zasilanie i V_{SS} masa.

W tym trybie obszar programu rozszerza się z 8K (0000h do 1FFFh) do 16K (0000h do 3FFFh), przy czym pierwsza część służy do trzymania kodu programu i stałych, i tylko ona jest dostępna w normalnym trybie pracy. Druga część obszaru zwana pamięcią konfiguracyjną zawiera cztery rejestry indentyfikacyjne ID (adresy od 2000h do 2003h i 2006h) oraz rejestr konfiguracyjny (adres 2007h). Pamięć EEPROM jest mapowana od adresu 2100h, zatem zawartość, która ma się w niej znaleźć może być wprowadzona w trakcie programowania z pliku HEX.



Rys. 2.75. Sposoby wejścia w tryb programowania: a) tryb normalny, b) tryb niskonapięciowy

Jak pokazano na rys. 2.75a w normalny tryb programowania można wprowadzić mk, jeżeli linie RB6 i RB7 są trzymane w stanie niskim podczas gdy napięcie na linii MCLR rośnie z V_{IL} (0V) do V_{IHH} (około 13V). W tym przypadku nie jest wykorzystywany pin RB3.

Natomiast tryb programowania LVP (rys. 2.75b) uzyskuje się przez podanie na RB3 sygnału narastającego podczas gdy MCLR=0, a następnie zwiększenie na linii MCLR napięcia z V_{IL} do V_{IH} (V_{DD} równe około 5V). Linie RB6 i RB7 również muszą być w stanie niskim.

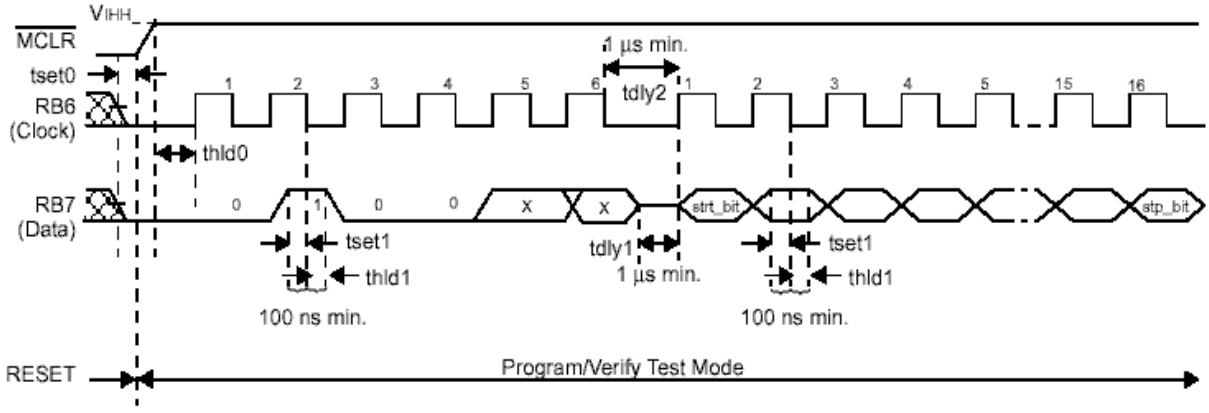
W obu przypadkach pozostałe linie mk są w stanie resetu, tzn. są liniami wejściowymi o wysokiej impedancji. Ponadto w trybie programowania linie RB6 i RB7 są wejściami Schmitta.

Po wprowadzeniu w ten tryb licznik rozkazów PC jest ustawiony na 0. Rozkaz inkrementacji (*increment address*) będzie zatem zwiększał licznik o jeden. Aby przejść do pamięci konfiguracyjnej należy wykonać instrukcję ładowania konfiguracji (*load configuration*), która ustawia licznik PC na 2000h. Listę komend zestawiono w tabeli 2.8.

Tabela 2.8. Zestawienie komend obsługi trybu programowania mk PIC16F873

Command	Mapping (MSB ... LSB)						Data	Voltage Range
Load Configuration	X	X	0	0	0	0	0, data (14), 0	2.2V - 5.5V
Load Data for Program Memory	X	X	0	0	1	0	0, data (14), 0	2.2V - 5.5V
Read Data from Program Memory	X	X	0	1	0	0	0, data (14), 0	2.2V - 5.5V
Increment Address	X	X	0	1	1	0		2.2V - 5.5V
Begin Erase Programming Cycle	0	0	1	0	0	0		2.2V - 5.5V
Begin Programming Only Cycle	0	1	1	0	0	0		4.5V - 5.5V
Load Data for Data Memory	X	X	0	0	1	1	0, data (14), 0	2.2V - 5.5V
Read Data from Data Memory	X	X	0	1	0	1	0, data (14), 0	2.2V - 5.5V
Bulk Erase Setup1	0	0	0	0	0	1		4.5V - 5.5V
Bulk Erase Setup2	0	0	0	1	1	1		4.5V - 5.5V

Pin RB6 służy jako linia zegarowa, natomiast na linii RB7 szeregowo wprowadza się rozkaz oraz dane do zapisu lub wyprowadza się dane w przypadku odczytu. Dane na linii RB7 są wprowadzane na narastające zbocze sygnału zegarowego, a próbkowane opadającym zboczem. Aby wprowadzić komendę potrzeba sześć impulsów zegarowych, po których jak mają być przesyłane dane powinno być opóźnienie minimum 1μs. Następnie można wprowadzić dane (14 bitów). Zaczynają się one od bitu START („0") i kończą bitem STOP („0"), co łącznie daje 16 bitów do przesłania. Rozkazy oraz dane są przesyłane od najmłodszego do najstarszego bitu. Na rys. 2.76 pokazano przykładowy diagram czasowy dla rozkazu *Load Configuration*.



Rys. 2.76. Diagram czasowy rozkazu *Load Configuration*

Zatem programowanie mk polega na wydawaniu odpowiednich rozkazów zgodnie z algorytmami zawartymi w dokumentacji mk. Np. aby wykasować pamięć programu należy wykonać kroki:

1. Wykonać komendę *Load Data for Program Memory* z daną 3FFFh,
2. Wykonać komendę *Bulk Erase Setup1*,
3. Wykonać komendę *Bulk Erase Setup2*,
4. Wykonać komendę *Begin Erase Programming Cycle*,
5. Odczekać 8ms,
6. Wykonać komendę *Bulk Erase Setup1*,
7. Wykonać komendę *Bulk Erase Setup2*.

2.9. Rodziny mk

Konsekwencją rezygnacji z otwartości konstrukcji procesora rdzeniowego i współpracujących z nim układów we/wy stała się konieczność produkowania różnych typów mk. Oparte o ten sam procesor rdzeniowy zawierają one różne zestawy urządzeń peryferyjnych, pozwalające na optymalne dokonanie wyboru mk w zależności od rodzaju aplikacji. Powstaje w ten sposób **rodzina mk**, w skład której wchodzą mk o różnych parametrach, ale zachowujące między sobą kompatybilność programową, tzn. posiadają tą samą jc, czyli tą samą listę instrukcji.

Modyfikacja członków danej rodziny odbywa się na kilku poziomach:

- na poziomie jc, dotyczy ona zmiany:
  - rozmiaru pamięci programu,
  - rozmiaru pamięci danych RAM,
  - maksymalnej szybkości pracy.
- na poziomie urządzeń peryferyjnych. Modyfikacje w tej warstwie stanowią podstawowy wyróżnik danego typu mk. Użytkownik otrzymuje do dyspozycji całą, niekiedy dość liczną rodzinę mk różniących się kombinacjami wbudowanych w układ scalony układami we/wy.
- na poziomie warstwy zacisków zewnętrznych i typu obudowy. Dotyczy głównie parametrów elektrycznych zacisków mk oraz stosowanego typu obudowy mk.

Na rynku istnieje wiele rodzin mk produkowanych przez różnych producentów. Informacje o najpopularniejszych rodzinach mk są dostępne na stronie http://www.microcontroller.com.

Najważniejszą rodziną mk 8-bitowych jest rodzina 8051 (nazywana przez jej twórcę firmę Intel – MCS51). Jest ona produkowana na zasadzie licencji przez szereg firm, pod różnymi nazwami, między innymi przez: Philips, Siemens, Atmel, Texas Instruments, Analog Devices.

Przykładowo w tabeli 2.9 przedstawiono część rodziny mk 8051 produkowaną przez firmę Atmel.

Tabela 2.9. Zestawienie mk rodziny 8051 firmy Atmel

	Program Memory (KB)	EEPROM (KB)	RAM (Bytes)	I/O Pins	UART	Timers	Other Features	Vcc (V)	(5V/ 3V) Max Speed (MHz)	Packages
ISP FLASH										
AT89S51	4		128	32	1	2	WDT	4.0 - 5.5	33	PDIP40, PLCC44, TQFP44
AT89LS51	4		128	32	1	2	WDT	2.7 - 5.5	16	PDIP40, PLCC44, TQFP44
AT89S52	8		256	32	1	3	WDT	4.0 - 5.5	33	PDIP40, PLCC44, TQFP44
AT89LS52	8		256	32	1	3	WDT	2.7 - 5.5	33	PDIP40, PLCC44, TQFP44
AT89S8252	8	2	256	32	1	3	WDT	3.0 - 6.0	12	PDIP40, PLCC44, TQFP44
AT89S53	12		256	32	1	3	WDT	3.0 - 6.0	24	PDIP40, PLCC44, TQFP44
T89C5115	16	2	512	20	1	3+PCA	WDT, A/D	4.5 - 5.5	40	SOIC28, PLCC28, VQFP32, SOIC24*
T89C5115	16	2	512	20	1	3+PCA	WDT, A/D	2.7 - 3.6	24	SOIC28, PLCC28, VQFP32, SOIC24*
T89C51RB2	16		1280	32	1	3+PCA	WDT, SPI, BRG, KBD	2.7 - 5.5	40	PDIP40, PLCC44, VQFP44
T89C51RB2	16		1280	32	1	3+PCA	WDT, SPI, BRG, KBD	2.7 - 3.6	30	PDIP40, PLCC44, VQFP44
T89C51RC2	32		1280	32	1	3+PCA	WDT, SPI, BRG, KBD	2.7 - 5.5	40	PDIP40, PLCC44, VQFP44
T89C51RC2	32		1280	32	1	3+PCA	WDT, SPI, BRG, KBD	2.7 - 3.6	30	PDIP40, PLCC44, VQFP44
T89C51IC2	32		1280	34	1	3+PCA	WDT, SPI, TWI, RTC, BRG, KBD	2.7 - 5.5	40	PLCC44, VQFP44
T89C51IC2	32		1280	34	1	3+PCA	WDT, SPI, TWI, RTC, BRG, KBD	2.7 - 3.6	30	PLCC44, VQFP44
T89C51AC2	32	2	1280	34	1	3+PCA	WDT, A/D	2.7 - 5.5	40	PLCC44, VQFP44, CA-BGA64*
T89C51AC2	32	2	1280	34	1	3+PCA	WDT, A/D	2.7 - 3.6	20	PLCC44, VQFP44, CA-BGA64*
T89C51RD2	64	2	1280	32/48	1	3+PCA	WDT	3.0 - 5.5	40	PDIP40, PLCC44, VQFP 44, PLCC68, VQFP64
T89C51RD2	64	2	1280	32/48	1	3+PCA	WDT	2.7 - 3.6	25	PDIP40, PLCC44, VQFP 44, PLCC68, VQFP64
FLASH										
AT89C1051U	1		64	15	1	2	Analog Comparator	2.7 - 6.0	24/12	PDIP20, SOIC20
AT89C2051	2		128	15	1	2	Analog Comparator	2.7 - 6.0	24/12	PDIP20, SOIC20
AT89C4051	4		128	15	1	2	Analog Comparator	2.7 - 6.0	24/12	PDIP20, SOIC20
AT89C51 (1)	4		128	32	1	2		4.0 - 6.0	33	PDIP40, PLCC44, TQFP44
AT89LV51 (2)	4		128	32	1	2		2.7 - 6.0	16	PDIP40, PLCC44, TQFP44
AT89C52 (3)	8		256	32	1	3		4.0 - 6.0	33	PDIP40, PLCC44, TQFP44
AT89LV52 (4)	8		256	32	1	3		2.7 - 6.0	16	PDIP40, PLCC44, TQFP44
AT89C55WD	20		256	32	1	3	WDT	4.0 - 5.5	33	PDIP40, PLCC44, TQFP44
AT89LV55WD	20		256	32	1	3	WDT	2,7 - 5.5	12	PDIP40, PLCC44, TQFP44
AT89LV55	20		256	32	1	3		2.7 - 5.5	12	PDIP40, PLCC44, TQFP44
AT89C51RC	32		512	32	1	3	WDT	4.0 - 5.5	33	PDIP40, PLCC44, TQFP44
AT89LV51RC	32		512	32	1	3	WDT	2.7 - 5.5	12	PDIP40, PLCC44, TQFP44

Kolejną popularną rodziną mk produkowanych przez firmę Atmel jest rodzina AVR 8-Bit RISC (w skrócie rodzina AVR). Fragment tej rodziny pokazano w tabeli 2.10.

Tabela 2.10. Zestawienie mk rodziny AVR firmy Atmel

	FLASH (KB)	EEPROM(Bytes)	RAM(Bytes) with 32 Register	Instructions	I/O Pins	Interrupts	Ext. Interrupts [1]	SPI	UART	TWI [2]	Hardware Multiplier	8-bit Timer	16-bit timer	PWM	Watchdog Timer	RTC Timer	Analog Comp	10-bit A/D Channels	On Chip RC-Oscillator	Brown Out Detector	In System Programming	Self Program Memory	Vcc(V)	Clock speed(MHz)	Packages
ATtiny11L	1	-	32	90	6	4	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	2.7-5.5	0-2	8-Pin DIP, SOIC
ATtiny11	1	-	32	90	6	4	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y[3]	-	4.0-5.5	0-6	8-Pin DIP, SOIC
ATtiny12V	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[3]	Y	Y	-	1.8-5.5	0-1	8-Pin DIP, SOIC
ATtiny12L	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[3]	Y	Y	-	2.7-5.5	0-4	8-Pin DIP, SOIC
ATtiny12	1	64	32	90	6	5	1(+5)	-	-	-	-	1	-	-	Y	-	Y	-	Y[3]	Y	Y	-	4.0-5.5	0-8	8-Pin DIP, SOIC
ATtiny15L	1	64	32	90	6	8	1(+5)	-	-	-	-	2	-	1	Y	-	Y	4	Y[3]	Y	Y	-	2.7-5.5	1.6	8-Pin DIP, SOIC
ATtiny26L	2	128	128+32	118	16	11	1(+8)	1[5]	1[6]	1[4]	-	2	-	4	Y	-	Y	11	Y[3]	Y	Y	-	2.7-5.5	0-8	20-Pin DIP, SOIC 32-Pin MLF
ATtiny26	2	128	128+32	118	16	11	1(+8)	1[5]	1[6]	1[4]	-	2	-	4	Y	-	Y	11	Y[3]	Y	Y	-	4.5-5.5	0-16	20-Pin DIP, SOIC 32-Pin MLF
ATtiny28V	2	-	32	90	20	5	1(+8)	-	-	-	-	1	-	-	Y	-	Y	-	Y[3]	-	Y[3]	-	1.8-5.5	0-1	28-Pin DIP 32-Pin TQFP, MLF
ATtiny28L	2	-	32	90	20	5	1(+8)	-	-	-	-	1	-	-	Y	-	Y	-	Y[3]	-	Y[3]	-	2.7-5.5	0-4	28-Pin DIP 32-Pin TQFP, MLF
AT90S1200	1	64	32	89	15	3	1	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y	-	2.7-6.0	0-4	20-Pin DIP, SOIC, SSOP
AT90S1200	1	64	32	89	15	3	1	-	-	-	-	1	-	-	Y	-	Y	-	Y	-	Y	-	4.0-6.0	0-12	20-Pin DIP, SOIC, SSOP
AT90S2313	2	128	128+32	120	15	10	2	-	1	-	-	1	1	1	Y	-	Y	-	-	-	Y	-	2.7-6.0	0-4	20-Pin DIP, SOIC
AT90S2313	2	128	128+32	120	15	10	2	-	1	-	-	1	1	1	Y	-	Y	-	-	-	Y	-	4.0-6.0	0-10	20-Pin DIP, SOIC
AT90LS2323	2	128	128+32	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	2.7-6.0	0-4	8-Pin DIP, SOIC
AT90S2323	2	128	128+32	120	3	2	1	-	-	-	-	1	-	-	Y	-	-	-	-	-	Y	-	4.0-6.0	0-10	8-Pin DIP, SOIC
AT90LS2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7-6.0	0-1	8-Pin DIP, SOIC
AT90LS2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	2.7-6.0	0-4	8-Pin DIP, SOIC
AT90S2343	2	128	128+32	120	5	2	1	-	-	-	-	1	-	-	Y	-	-	-	Y	-	Y	-	4.0-6.0	0-10	8-Pin DIP, SOIC
AT90LS4433	4	256	128+32	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	2.7-6.0	0-4	28-Pin DIP 32-Pin TQFP
AT90S4433	4	256	128+32	120	20	14	2	1	1	-	-	1	1	1	Y	-	Y	6	-	Y	Y	-	4.0-6.0	0-8	28-Pin DIP 32-Pin TQFP
AT90S8515	8	512	512+32	120	32	12	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	2.7-6.0	0-4	40-Pin DIP 44-Pin PLCC, TQFP
AT90S8515	8	512	512+32	120	32	12	2	1	1	-	-	1	1	2	Y	-	Y	-	-	-	Y	-	4.0-6.0	0-8	40-Pin DIP 44-Pin PLCC, TQFP
AT90LS8535	8	512	512+32	120	32	16	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	2.7-6.0	0-4	40-Pin DIP 44-Pin PLCC, TQFP
AT90S8535	8	512	512+32	120	32	16	2	1	1	-	-	2	1	3	Y	Y	Y	8	-	-	Y	-	4.0-6.0	0-8	40-Pin DIP 44-Pin PLCC, TQFP

Firma Microchip (http://www.microchip.com) produkuje szereg rodzin mk łącznie określanych PICmicro. Wszystkie one oparte są na jc o architekturze RISC. Oferowanych jest pięć rodzin mk, różniących się między innymi długością rozkazów i ich ilością: PIC12, PIC14, PIC16, PIC17 i PIC18.

Dla przykładu w tabeli 2.11 zestawiono mały fragment mk rodziny PIC16.

Tabela 2.11. Fragment listy mk rodziny PIC16

Device	Data RAM	ADCbits	Words	Serial I/O	Speed	Timers	Low Voltage Device
PIC16C54C	25	-	512	-	40	1+WDT	PIC16LC54C
PIC16CR54C	25	-	512	-	20	1+WDT	PIC16LCR54C
PIC16C745	256	5/8-bit	8192	USB, USART	24	3+WDT	-
PIC16C765	256	8/8-bit	8192	USB, USART	24	3+WDT	-
PIC16C781	128	8/8-bit	1024	-	20	2+WDT	PIC16LC781
PIC16C782	128	8/8-bit	2048	-	20	2+WDT	PIC16LC782
PIC16F84A	68	-	1024	-	20	1+WDT	PIC16LF84A
PIC16F870	128	5/10-bit	2048	USART	20	3+WDT	PIC16LF870
PIC16F871	128	8/10-bit	2048	USART	20	3+WDT	PIC16LF871
PIC16F872	128	5/10-bit	2048	I²C, SPI	20	3+WDT	PIC16LF872
PIC16F873	192	5/10-bit	4096	USART, I²C, SPI	20	3+WDT	PIC16LF873
PIC16F873A	192	5/10-bit	4096	USART, I²C, SPI	20	3+WDT	PIC16LF873A
PIC16F874	192	8/10-bit	4096	USART, I²C, SPI	20	3+WDT	PIC16LF874
PIC16F874A	192	8/10-bit	4096	USART, I²C, SPI	20	3+WDT	PIC16LF874A
PIC16F876	368	5/10-bit	8192	USART, I²C, SPI	20	3+WDT	PIC16LF876
PIC16F876A	368	5/10-bit	8192	USART, I²C, SPI	20	3+WDT	PIC16LF876A
PIC16F877	368	8/10-bit	8192	USART, I²C, SPI	20	3+WDT	PIC16LF877
PIC16F877A	368	8/10-bit	8192	USART, I²C, SPI	20	3+WDT	PIC16LF877A

Firma STMicroelectronics (http://www.st.com) jest znaczącym europejskim producentem mk. Produkuje ona między innymi następujące rodziny: ST6, ST7, ST9, ST10, ST40, Super10, ST100 i ST52. Dla przykładu w skład bardzo licznej rodziny ST7 wchodzą mk posiadające interfejs CAN np. ST72561xx i interfejs USB np. ST7262xx. Poniżej pokazano fragment tabeli z „podrodziną", z której pochodzi często przytaczany mk ST72215G (tabela 2.12).

Tabela 2.12. Fragment rodziny mk ST7

	ROM Device	Program Memory Type			Memory Size (bytes)			Peripherals				Package	Additional Features
		EPROM	OTP/ FLASH	FAST ROM	Prog Mem	RAM	Data EEP ROM	ADC resol (Inputs)	Timers	Serial Interface	I/Os (High Sink)		
BASIC	ST72101G1	ST72E251G2	OTP	Yes	4K	256	-	-	1x7-Bit WDG 1x16-Bit	SPI	22(8)	SDIP32/ SO28	POR
	ST72101G2	ST72E251G2	OTP	Yes	8K	256	-	-	1x7-Bit WDG 1x16-Bit	SPI	22(8)	SDIP32/ SO28	
	ST72104G1	-	FLASH	Yes	4K	256	-	-	1x7-Bit WDG 1x16-Bit	SPI	22(8)	SDIP32/ SO28	3 level LVD, RC Oscillator, CSS, ISP, ROP
	ST72104G2	-	FLASH	Yes	8K	256	-	-	1x7-Bit WDG 1x16-Bit	SPI	22(8)	SDIP32/ SO28	
	ST72121J2[1)]	ST72E311J4	OTP	Yes	8K	384	-	-	1x7-Bit WDG 2x16-Bit	SPI/ SCI	32(4)	SDIP42/ TQFP44	POR
	ST72121J4[1)]	ST72E311J4	OTP	Yes	16K	512	-	-	1x7-Bit WDG 2x16-Bit	SPI/ SCI	32(4)	SDIP42/ TQFP44	
	ST72124J2	ST72E311J4	FLASH	Yes	8K	384	-	-	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	32(4)	SDIP42/ TQFP44	3 level LVD, Active Halt, RC Oscillator, CSS, Beep, ISP, ROP
	ST72124J4	-	FLASH	Yes	16K	512	-	-	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	32(4)	SDIP42/ TQFP44	
ADC	ST72212G2	ST72E251G2	OTP	Yes	8K	256	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit	SPI	22(8)	SDIP32/ SO28	POR
	ST72213G1	ST72E251G2	OTP	Yes	4K	256	-	8-Bit (6)	1x7-Bit WDG 1x16-Bit	SPI	22(8)	SDIP32/ SO28	
	ST72215G2	-	FLASH	Yes	8K	256	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit	SPI	22(8)	SDIP32/ SO28	3 level LVD, RC Oscillator, CSS, ISP, ROP
	ST72216G1	-	FLASH	Yes	4K	256	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit	SPI	22(8)	SDIP32/ SO28	
	ST72311J2[2)]	ST72E311J4	OTP	Yes	8K	384	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit	SPI/ SCI	32(4)	SDIP42/ TQFP44	LVD
	ST72311J4[2)]	ST72E311J4	OTP	Yes	16K	512	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit	SPI/ SCI	32(4)	SDIP42/ TQFP44	
	ST72314J2	-	FLASH	Yes	8K	384	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	32(4)	SDIP32/ TQFP44	3 level LVD, Active Halt, RC Oscillator, CSS, Beep, ISP, ROP
	ST72314J4	-	FLASH	Yes	16K	512	-	8-Bit (6)	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	32(4)	SDIP32/ TQFP44	
	ST72311N2[2)]	ST72E311N4	OTP	Yes	8K	384	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit	SPI/ SCI	44(8)	SDIP56/ TQFP64	LVD
	ST72311N4[2)]	ST72E311N4	OTP	Yes	16K	512	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit	SPI/ SCI	44(8)	SDIP56/ TQFP64	
	ST72314N2	-	FLASH	Yes	8K	384	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	44(8)	SDIP56/ TQFP64	3 level LVD, Active Halt, RC Oscillator, CSS, Beep, ISP, ROP
	ST72314N4	-	FLASH	Yes	16K	512	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit - RTB	SPI/ SCI	44(8)	SDIP56/ TQFP64	
	ST72311R6	ST72E511R9	OTP	Yes	32K	1024	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit 1x8-Bit - RTB	SPI/ SCI	48(12)	TQFP64	4 PWM, Nested Interrupts, LVD, Active Halt, CSS, Beep, ROP
	ST72311R7	ST72E511R9	OTP	Yes	48K	1536	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit 1x8-Bit - RTB	SPI/ SCI	48(12)	TQFP64	
	ST72311R9	ST72E511R9	OTP	Yes	60K	2048	-	8-Bit (8)	1x7-Bit WDG 2x16-Bit 1x8-Bit - RTB	SPI/ SCI	48(12)	TQFP64	