

Elektronika

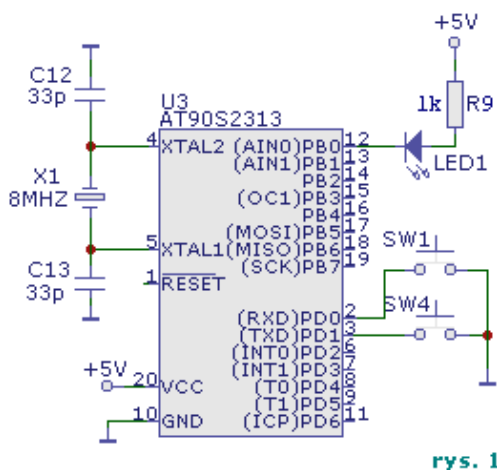


mikrokontrolery



Kurs BASCOM-AVR w przykładach

Pierwszy program



rys. 1



Od czego zacząć? - Co to jest mikrokontroler? - Kody i liczby - Podstawowe układy logiczne - Budowa i działanie AT90S2313 - Zestawy uruchomieniowe - Kurs BASCOM-AVR w przykładach - Kurs C dla AVR w przykładach - Kurs Asemblera dla AVR w przykładach - Gotowe projekty



Pierwszy program jaki wszyscy najczęściej piszą, to zapalanie diody LED podłączonej do któregoś z portów mikrokontrolera. Ja nie byłem więc oryginalny i również taki program napisałem, a poza tym to dobre ćwiczenie przy poznawaniu konfiguracji portów.

Dioda LED podłączona będzie do wyprowadzenia PB0 portu B. Do wyprowadzeń PD0 i PD1 portu D podłączone będą przełączniki, których zadaniem będzie odpowiednio włączanie i wyłączanie świecenia diody LED. Naciśnięcie przełącznika podłączonego do PD0 powinno spowodować zaświecenie diody, natomiast naciśnięcie przełącznika podłączonego do PD1 powinno spowodować zgaszenie diody.

Zapewne każdy kto używał różnego rodzaju przełączników wie, że podczas naciśnięcia, a następnie zwolnienia przycisku przełącznika występuje zjawisko drgania styków powodujące na przemian zwarcie i rozwarcie styków. W tym konkretnym przypadku nie będziemy zajmować się programową eliminacją wpływu drgania styków przełączników gdyż nie ma to znaczenia dla poprawnego działania układu.

Opisywane działanie programu ma być zrealizowane w układzie przedstawionym na rys. 1. Jest to fragment schematu ideowego zestawu ZL1AVR.

W tabelce poniżej są pliki do ściągnięcia z omawianym programem i pomoc dla BASCOM-AVR w języku polskim.

Pliki do pobrania

Wersja programu	Nazwa pliku do pobrania	
Pierwszy program - zestaw ZL1AVR	prog001.bas	PROG001.BIN
Pomoc BASCOM-AVR (j. polski)	bascavr.zip	-

Aby schemat widoczny na rys. 1 był zgodny z połączeniami w ZL1AVR, to w zestawie należy zewrzeć następujące złącza:

- JP1 - podłączenie +5V do zasilania LED,
- JP6 i JP7 - podłączają SW1 do PD0 i SW4 do PD1
- JP4 zewrzeć 2 z 3 - podłączy to masę do SW1 i SW4
- J4 - 2 z 3, J3 - 1 z 2 - zostanie dołączony kwarc X1
- ZW_PORTB - 1 z 2 - podłączenie LED1 do PB0
- do JP13 podłączyć zasilanie.

Dioda LED1 zaświeci się gdy PB0 (pin12 AT90S2313) będzie skonfigurowany jako wyjście i jego stan przyjmie wartość "0", to umożliwi przepływ prądu przez diodę LED1, prąd ten ograniczony jest rezystorem R9 do wartości ok. 3,5mA (zależy to również od wartości spadku napięcia na diodzie LED). Aby świecenie diody uzależnić od stanu przełączników SW1 i SW4 to PD0 (pin 2) i PD1 (pin 3) muszą być skonfigurowane jako wejściem typu *pull-up* wymuszającym początkowy stan 1. W takim przypadku naciśnięcie jednego z przełączników spowoduje, że na odpowiednim wejściu pojawi się stan "0". Pozostałe niewykorzystywane wyprowadzenia zarówno portu B jak i D mogą być skonfigurowane dowolnie, można je więc ustawić np. jako wyjścia.

W procesorach AVR na początku zawsze należy skonfigurować porty określając dla każdego wyprowadzenia dwa parametry:
 - funkcję jaką ma pełnić - wejścia czy wyjścia

- stan spoczynkowy jaki ma przyjąć - "0" czy "1"
Jeżeli porty nie zostaną skonfigurowane, to przy starcie mikrokontrolera (po wyzerowaniu) rejestry PORTx i DDRx (x w zależności od portu będzie zastąpiony literką A, B, C lub D) zostaną wyzerowane co oznacza, że wszystkie wyprowadzenia portów będą wejściami w stanie wysokiej impedancji (wejścia pływające)

Teraz przechodzimy do sedna sprawy, czyli jak to co opisałem wyżej zamienić na program zrozumiały dla naszego AVR-a.

W **helpie do BASCOM-AVR** (wersja w języku polskim) są dokładnie opisane wszystkie instrukcje i inne elementy języka BASCOM dla AVR, dlatego nie ma sensu abym w tym miejscu bardziej szczegółowo opisywał te elementy, umieszczał będę tylko niezbędne dla zrozumienia tematu informacje i wyjaśnienia. Ściągnij sobie tego helpa i często tam zaglądaj!

Zgodnie z tym co jest napisane powyżej (w ramce) należy najpierw skonfigurować porty AT90S2313. Do konfiguracji urządzeń sprzętowych w BASCOM-AVR służy instrukcja **CONFIG**, którą się używa razem z nazwą konfigurowanego sprzętu, czyli w przypadku konfiguracji portu nazwą będzie **PORTx**, a w przypadku pojedynczej końcówki nazwą będzie **PINx.y**, gdzie x to port B lub D (dla innych mikrokontrolerów może być jeszcze A lub C), y to numer końcówki z danego portu (0 ... 6 lub 7). Za stan portów w mikrokontrolerach AVR odpowiedzialne są trzy rejestry DDRx, PORTx i PINx (x to: A, B, C lub D). Instrukcja CONFIG ustawia cały port lub wybraną końcówkę portu w tryb pracy wejścia lub wyjścia. Inaczej mówiąc ustawia odpowiednio rejestr kierunku czyli DDRx. Jeżeli do każdego bitu rejestru wpiszemy "1" to wszystkie wyprowadzenia portu będą wyjściami, natomiast jeżeli do każdego bitu rejestru DDRx wpiszemy "0" to wszystkie wyprowadzenia będą wejściami. W przykładach poniżej stosuję kolorystykę składni poleceń taką jaką mam ustawioną w BASCOM-AVR. Kolorem zielonym i kursywą wyróżniam komentarze które zawsze muszą być poprzedzone znakiem apostrofu ' lub instrukcją **REM**. Warto przyzwyczaić się do opatrywania swoich programów komentarzami gdyż to po pewnym czasie ułatwia analizę wcześniej napisanych programów:

' - to jest komentarz
Rem - i to też jest komentarz

Konfiguracja całego portu B jako wyjście lub wejście:

```
Config Portb = Output 'cały port B jako wyjście  
Config Portb = Input 'cały port B jako wejście
```

można również skonfigurować każde wyprowadzenie (każdy bit) osobno:

```
Config Pinb.0 = Output 'wyprowadzenie PB0 portu B jako wyjście  
Config Pinb.1 = Output 'wyprowadzenie PB1 portu B jako wyjście  
Config Pinb.2 = Output 'wyprowadzenie PB2 portu B jako wyjście  
Config Pinb.3 = Output 'wyprowadzenie PB3 portu B jako wyjście  
Config Pinb.4 = Output 'wyprowadzenie PB4 portu B jako wyjście  
Config Pinb.5 = Output 'wyprowadzenie PB5 portu B jako wyjście  
Config Pinb.6 = Output 'wyprowadzenie PB6 portu B jako wyjście  
Config Pinb.7 = Output 'wyprowadzenie PB7 portu B jako wyjście  
Config Pinb.0 = Input 'wyprowadzenie PB0 portu B jako wejście  
Config Pinb.1 = Input 'wyprowadzenie PB1 portu B jako wejście  
Config Pinb.2 = Input 'wyprowadzenie PB2 portu B jako wejście  
Config Pinb.3 = Input 'wyprowadzenie PB3 portu B jako wejście  
Config Pinb.4 = Input 'wyprowadzenie PB4 portu B jako wejście  
Config Pinb.5 = Input 'wyprowadzenie PB5 portu B jako wejście  
Config Pinb.6 = Input 'wyprowadzenie PB6 portu B jako wejście  
Config Pinb.7 = Input 'wyprowadzenie PB7 portu B jako wejście
```

powyższego zapisu nie polecam z oczywistych powodów, lepiej przedstawić to samo korzystając z tego, że bajt można przedstawić jako 8 bitów i od razu będzie widać, który bit jest wyjściem, a który wejściem:

```
Config Portb = &B11111111 ' cały port B jako wyjście
Config Portb = &B00000000 ' cały port B jako wejście
```

w tym przypadku wszystkie wyprowadzenia są wyjściami lub wejściami, gdyby zapisać:

```
Config Portb = &B11111100 ' wyprowadzenia PB0 i PB1 to wejścia,
                          ' PB2 do PB7 to wyjścia
```

oznaczałoby to, że bit 0 i bit 1 są ustawione na 0 co oznacza, że wyprowadzenie PB0 i PB1 portu B są wejściami, a pozostałe wyjściami. Prefiks **&B** oznacza w BASCOM-AVR, że liczba występująca po tym znaku jest zapisana w postaci dwójkowej, prefiks **&H** oznacza liczbę zapisaną w kodzie szesnastkowym, bez prefiksu liczba w kodzie dziesiętnym.

Przypisanie funkcji dla portu to pierwsza rzecz, druga natomiast to określenie stanu spoczynkowego wyprowadzeń portów mikrokontrolera po starcie programu. W tym celu do każdego bitu rejestru PORTx należy wpisać "0" lub "1". Można tego dokonać dla każdego bitu osobno, lub dla całego rejestru od razu, podobnie jak to miało miejsce przy przypisaniu funkcji wejścia lub wyjścia.

Ustawienie stanu spoczynkowego portu B:

```
Portb = &B11111111 ' podciągnięcie wszystkich wyprowadzeń
                  ' portu B do "1"
```

dla portu B skonfigurowanego jako wyjście będzie to oznaczało, że stanem spoczynkowym wszystkich wyprowadzeń jest jedynka (stan wysoki), dla portu B skonfigurowanego jako wejście, to oznacza podciągnięcie wszystkich wejść do jedynki. Gdy zapiszemy:

```
Portb = &B00000000 ' dla portu B jako wyjście oznacza to
                  ' ustawienie na wszystkich wyprowadzeniach
                  ' stanu "0",
                  ' dla portu B jako wejście oznacza to
                  ' pozostawienie wszystkich wejść
                  ' pływających
```

będzie to oznaczało dla portu B skonfigurowanego jako wyjście ustawienie na wszystkich wyprowadzeniach stanu logicznego "0", natomiast dla portu B skonfigurowanego jako wejście oznacza to, że wszystkie wejścia pozostają w stanie wysokiej impedancji (wejścia pływające). Warto pamiętać, że dla wyprowadzenia portu pełniącego rolę wejścia, do którego podpięty jest przełącznik zwierający go do zera (po naciśnięciu), ważnym jest aby to wejście było podciągnięte do jedynki.

Podobnie jak konfigurowanie pojedynczego wyprowadzenia portu można zrobić ustawienie stanu spoczynkowego dla pojedynczego wyprowadzenia, co ilustruje poniższy zapis:

```
Portb.5 = 1 ' wyprowadzenie PB5 portu B podciągnięte do "1"
Portb.5 = 0 ' wejście PB5 portu B pływające lub
            ' wyjście PB5 portu B ustawione w stan "0"
```

Po tych wszystkich wyjaśnieniach wracamy do naszego układu i przygotowujemy porty mikrokontrolera do pracy zgodnie z założeniami.

```
Config Portb = &B11111111 ' cały port B jako wyjście
Portb = &B11111111 ' wszystkie wyjścia w stanie "1"
Config Portd = &B11111100 ' PD0 i PD1 - wejścia, pozostałe - wyjścia
Portd = &B11111111 ' PD0 do PD6 podciągnięte do "1"
```

Nasz mikrokontroler przygotowany jest do pracy, SW1 i SW4 podpięte są do wejść PD0 i PD1, które podciągnięte są do jedynki, pozostałe wyprowadzenia portu D są wyjściami (co dla działania układu jest obojętne), cały port B jest wyjściem i stan spoczynkowy jest "1", co w przypadku diody LED1 podpiętej

do PBO powoduje, że dioda nie świeci. Aby program działał zgodnie z założeniami, to należy sterować stanem wyjścia PBO poprzez zmianę najmłodszego bitu rejestru PORTB czyli Portb.0 w zależności od stanu wejść PD0 i PD1. Zmieniać stan bitu rejestru PORTB (i innych rejestrów PORTA, PORTD, PORTC również) można w BASCOM-AVR na dwa sposoby, aby dioda zaświeciła należy napisać:

```
Portb.0 = 0 ' PBO w stanie "0" - dioda świeci
```

lub

```
Reset Portb.0 ' PBO w stanie "0" - dioda świeci
```

w drugim sposobie użyłem instrukcji **RESET**, która ustawia określony bit w stan "0".

Aby dioda przestała świecić należy napisać:

```
Portb.0 = 1 ' PBO w stanie "1" - dioda nie świeci
```

lub

```
Set Portb.0 ' PBO w stanie "1" - dioda nie świeci
```

tutaj w drugim sposobie użyłem instrukcji **SET**, która Ustawia określony bit w stan "1".

Czas na najważniejsze, a więc co trzeba zrobić aby program ciągle sprawdzał stan wejść, do których są podpięte przełączniki SW1, SW4 i w zależności od ich stanów zmieniał stan wyjścia PBO powodując świecenie bądź gaszenie diody LED1. Aby program wykonywał w kółko jakąś operację należy zastosować w programie nieskończoną pętlę. W BASCOM-AVR może do tego celu posłużyć instrukcja **DO ... LOOP**, która powtarza blok programu dopóki warunek końcowy nie będzie spełniony. Gdy warunek nie zostanie podany (a tak będzie w naszym przypadku) pętla będzie się wykonywać w nieskończoność. Zapisać to więc można tak jak poniżej:

```
Do ' początek nieskończonej pętli
    ciąg wykonywanych instrukcji
Loop ' powrót do początku pętli
```

Teraz pozostało jeszcze spowodowanie sprawdzania stanów wejść PD0, PD1 i uzależnienia od nich stanu PBO, czyli musimy wykonać instrukcję w stylu: "wykonaj coś pod warunkiem, że jest spełnione coś innego". W BASCOM-AVR jest instrukcja **IF ... THEN ... ELSE ... END IF**, która znakomicie się do naszych celów nadaje. Tworzy ona tzw. blok decyzyjny. Instrukcja **IF ... THEN** oblicza logiczną wartość podanego wyrażenia. Jeśli będzie ono prawdziwe (wynikiem będzie logiczna prawda) wykonany zostanie blok instrukcji umieszczony po instrukcji **THEN**. Jeśli będzie ono fałszywe, to instrukcje po słowie **THEN** nie zostaną wykonane. Wykonane za to będą instrukcje po słowie **ELSE**, jeśli ono występuje. W naszym przypadku musimy więc sformułować następujący blok decyzyjny: "... jeśli PD0 jest w stanie zero, to ustaw PBO w stan "0" jeśli tak nie jest to przejdź do następnej instrukcji, w której będzie następna decyzja - jeśli PD1 jest w stanie zero, to ustaw PBO w stan "1", jeśli tak nie jest to wróć do początku ...". I tu mała niespodzianka - jak odczytać stan dowolnego wyprowadzenia portu? Umiemy wpisywać do dowolnego bitu portu jedynkę lub zero poleceniem **PORTx.y = 1** czy **PORTx.y = 0**, a jak jest z odczytem? I tu okazuje się przydatny trzeci rejestr PINx (to jest rejestr tylko do odczytu; x to: A, B, C lub D). Do odczytu stanu wyprowadzenia dowolnego portu służy polecenie **PINx.y**. Jeśli już wszystko jasne, to czas zapisać nasz blok decyzyjny:

```
Do ' początek nieskończonej pętli
    If Pind.0 = 0 Then Portb.0 = 0 ' jeśli na wejściu PD0 jest 0 to wyjście
    ' PBO przyjmuje stan "0"
    If Pind.1 = 0 Then Portb.0 = 1 ' jeśli na wejściu PD1 jest 0 to wyjście
    ' PBO przyjmuje stan "1"
Loop ' powrót do początku pętli
```

Teraz wystarczy połączyć wszystkie niezbędne elementy programu czyli "blok konfiguracyjny porty" oraz ostatnio napisany "blok decyzyjny", całość zakończyć instrukcją **END** i otrzymamy pierwszy program:

```

Config Portb = &B11111111 ' cały port B jako wyjście
          Portb = &B11111111 ' wszystkie wyjścia w stanie "1"
Config Portd = &B1111100  ' PD0 i PD1 - wejścia, pozostałe - wyjścia
          Portd = &B1111111  ' PD0 do PD6 podciągnięte do "1"
Do                                     ' początek nieskończonej pętli
  If Pind.0 = 0 Then Portb.0 = 0 ' jeśli na wejściu PD0 jest 0 to wyjście
                                     ' PB0 przyjmuje stan "0"

  If Pind.1 = 0 Then Portb.0 = 1 ' jeśli na wejściu PD1 jest 0 to wyjście
                                     ' PB0 przyjmuje stan "1"

Loop                                     ' powrót do początku pętli
End                                       ' koniec programu

```

Teraz wystarczy zaprogramować AT90S2313 i sprawdzić działanie programu 😊 . Powodzenia!

To nie wszystko - już wkrótce następne programy kursu BASCOM-AVR ...

Literatura: "Mikrokontrolery AVR w praktyce" - J. Doliński
 "Mikroprocesorowa ośła łączka" - P. Górecki (kurs w EDW)
 "Bascom-AVR - help w wersji polskiej" - Z. Gibek



••• [wstecz](#) • [menu](#) • [dalej](#) •••

[nowości na stronie](#) | [elektronika w stylu retro](#) | [mikrokontrolery](#) | [ciekawe rozwiązania układowe](#) | [katalogi](#) | [schematy](#) | [coś dla początkujących](#) | [teoria - wstęp](#) | [podstawowe prawa](#) | [elementy RLC](#) | [diody](#) | [tranzystory](#) | [źródła napięcia i prądu](#) | [wzmacniacze operacyjne](#) | [elementy optoelektroniczne](#) | [filtry](#) | [generatory](#) | [zasilacze](#) | [stabilizatory](#) | [zadania i przykłady](#) | [trochę matematyki](#) | [archiwum](#) |

© Copyright 2001-2004 <http://www.elektronika.qs.pl>