Mikrokontrolery AVR – Wprowadzenie

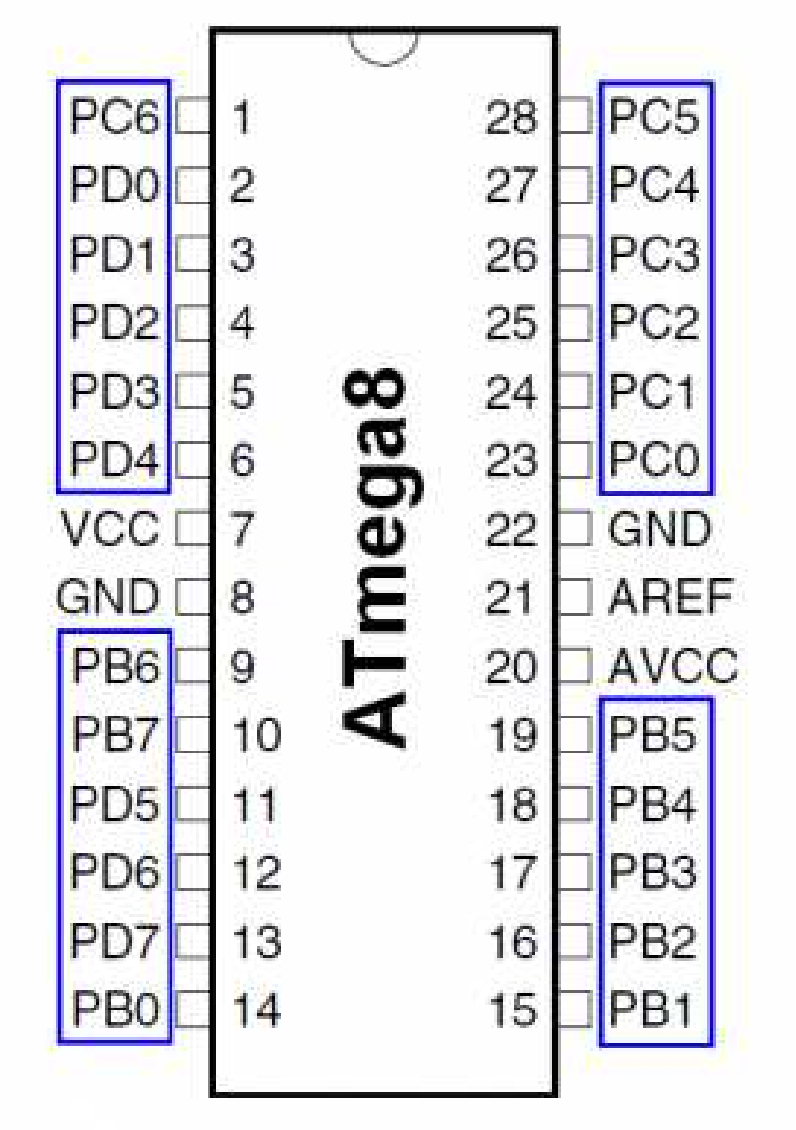
Komunikacja z otoczeniem mikrokontrolera

Każdy z mikrokontrolerów posiada pewna liczbę wyprowadzeń cyfrowych które służą do wprowadzania i odbierania informacji z mikrokontrolera. Wyprowadzenia te umożliwiają komunikację z układami peryferyjnymi, innymi mikroprocesorami oraz służą do programowania układów.

Wyprowadzenia nazywane są I/O (ang. In/Out) (wejścia/wyjścia).

Dla przykładu układ ATmega8 ma 23 programowalne wyprowadzenia I/O, a mikrokontroler ATmega16 posiada aż 32 I/O.

Linie wejść/wyjść podzielone zostały na porty.



- Wszystkie linie programowalne I/O zostały pogrupowane w porty zawierające do 8 linii.
- Firma Atmel produkująca mikroprocesory AVR przyjęła następujące oznaczanie wyprowadzeń: pierwsza litera P to skrót od nazwy PORT, druga litera oznacza nazwę portu.
- Dla ATmega8 są to litery „B", „C" i „D".
- Cyfra występująca po literze portu oznacza bit wyprowadzenia tego portu i może zawierać się w granicach od 0 do 7.

Wyprowadzenie ma oznaczenie PC5 – oznacza to, że jest to 5 bit portu C.

Nie każdy port ma 8 wyprowadzeń.

- Przykładowo port C posiada tylko 7 wyprowadzeń, a pozostałe pory B i D po 8.
- Dodatkowo linia PC6 nie jest normalnie dostępna jako I/O. Domyślnie znajduje się tam pin resetu.

- Mniejsze układy mają mniejsza ilość pinów

Programowanie

Podczas programowania podaje się literę portu i cyfrę dla określenia konkretnego wyprowadzenia.

W programie można odwoływać się również do całego portu i wpisać do niego pewną liczbę lub odczytać jego stan i zapisać go do zmiennej.

Za wszystkie ustawienia w mikrokontrolerze odpowiadają rejestry.

W rejestrach znajdują się bity które mogą przyjmować stan 0 lub 1 (rejestr to tablica w której każdy z bitów to oddzielna komórka). Tak jest w przypadku linii I/O.

Port jest rejestrem a konkretne linie są bitami tego rejestru. Poniżej przykład 3 rejestrów odpowiedzialnych za ustawienia fizycznych wyprowadzeń mikrokontrolera.

DDRB	DDRC	DDRD	rejestry kierunku
PORTB	PORTC	PORTD	rejestry wyjściowe
PINB	PINC	PIND	rejestry wejściowe

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bit – numer bitu
Read/Write – Litera R (read) oznacza możliwość odczytu bitu, litera W (write) oznacza możliwość zapisu bitu
Initial Value – wartość domyślna bitu

Przykład programowania – wyprowadzanie liczb na zewnątrz

Pierwszy z rejestrów PORTx (gdzie x to litera portu) odpowiedzialny jest za dane portu x.

Do portu B wpisujemy wartość dziesiętną 94.

- Heksalne 5E
- Binarne 01011110

```
PORTB = 0x5E;
```

Jak można zauważyć dodano przedrostek „0x” określa on, że wartość podana jest w systemie heksalnym. Równoznaczny z poprzednią instrukcją będzie zapis:

```
PORTB = 01011110;
```

Ustawienie bitów na stan wysoki (które mają mieć wartość 1)

```
PORTB = _BV(6)|_BV(4)|_BV(3)|_BV(2)|_BV(1); // ustaw bity 6,4,3,2,1 w rejestrze PORTB
```

Bit	7	6	5	4	3	2	1	0	
	0	1	0	1	1	1	1	0	PORTB

Przykład programowania – kierunek wejście/wyjście

Rejestr DDRx (gdzie x to litera portu) określa czy wyprowadzenie I/O ma być wejściem lub wyjściem.

Odpowiednie ustawienie bitów określa kierunek danych:

- 0 – wejście (In)
- 1 – wyjście (Out)

Przykładowo by wyprowadzenia PB4, PB5 i PB7 miały służyć jako wyjścia a pozostałe jako wejścia

rejestr DDRB w sposób podany poniżej

```
DDRB = 10110000; // zapisz wartość 10110000 do rejestru DDRB
```

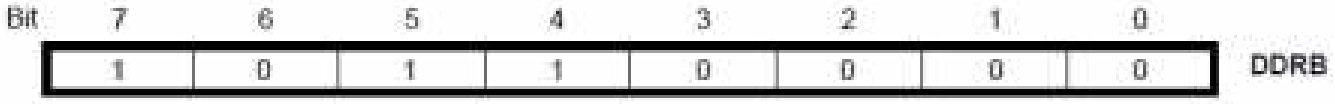
Identyczną wartość można zapisać również w systemie szesnastkowym (heksalnym):

```
DDRB = 0xB0;
```

Można również użyć poleceń ustawiających poszczególne bity rejestru:

```
DDRB = _BV(7)|_BV(5)|_BV(4);
```

Efekt widać poniżej



Przykład programowania – wczytywanie liczby

Do odczytu stanu któregoś z wejść mikrokontrolera korzysta się z rejestru PINx, (gdzie x to litera portu).

W rejestrze PINx zapisane są binarnie fizyczne stany wybranego portu (1 - stan wysoki, 0 - stan niski).

Wcześniej odpowiednio ustawiamy rejestr DDRx by wyprowadzenia na których sprawdzamy stan były wejściami (wartość 0).

Poniżej przykład pobierający dane z portu B i zapisujący stan całego portu do zmiennej „dane”.

```
dane = PINB; // zapisz stan rejestru PINB do zmiennej dane
```

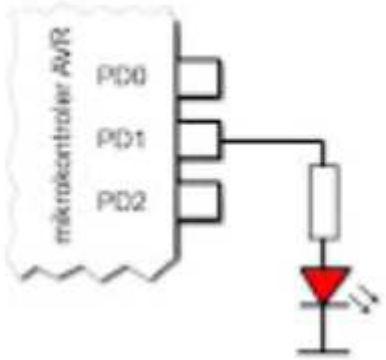
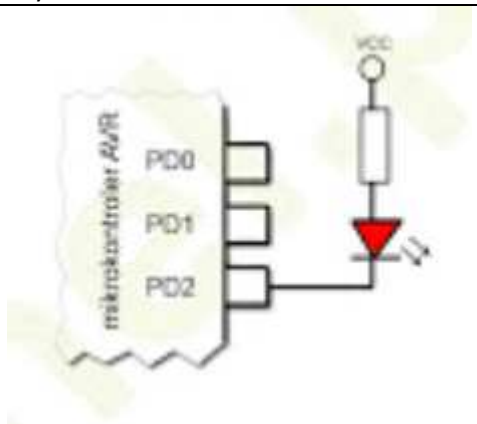
Konfiguracja portów

DDRx.n	PORTx.n	Px.n
0	0	wejście
0	1	wejście z podciągnięciem do VCC
1	X	wyjście

Konfiguracja równoległych portów we/wy mikrokontrolerów AVR

Program wyświetlanie diody LED

W ramach wyświetlania możemy ustalić jaki sposób chcemy użyć. Czy wyświetlamy korzystając z wysokiego lub niskiego stanu.

Wyświetlanie diod stanem wysokim	Wyświetlanie diod stanem niskim
	
Ustawienie wyjścia w stan wysoki	Ustawienie wyjścia w stan niski

Cały sposób, styl programowania jest jednakowy. Różnica polega na zapisaniu innego stanu na wyjściu.

1. Musimy ustalić kierunek danych.

   Ponieważ chcemy wyświetlić, to trzeba skierować je na zewnątrz.

Trzeba użyć rejestru DDR. Wykorzystujemy nóżki rejestru D, więc będzie to DDRD.

Wyjście to stan wysoki, więc na odpowiedni pin trzeba wysłać 1.

```
DDRD |= (1<<PD1);
```

Dotyczy to obydwu wyjść

```
DDRD |= (1<<PD2);
```

2. Trzeba ustalić odpowiednią wartość na wyjściu 1 dla PD1 i 0 dla PD2.

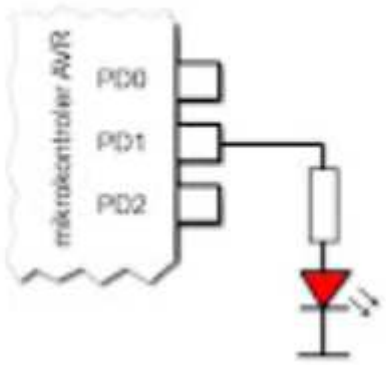
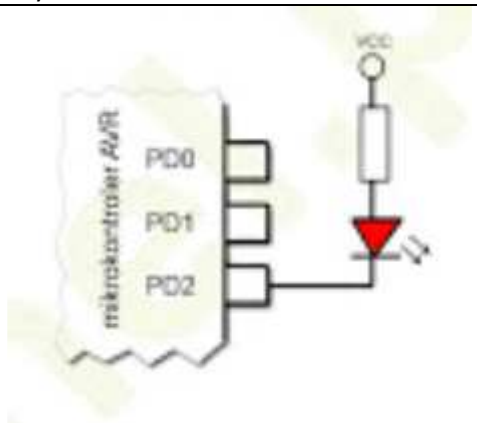
   Wykorzystujemy tu wartości rejestru PORT, który odpowiada za wysłanie danych na zewnątrz

   Na PD1 wysyłamy stan wysoki

```
PORTD |= (1<<PD1);
```

   Na PD2 wysyłamy stan niski

```
PORTD &= ~(1<<PD2);
```

Wyświetlanie diod stanem wysokim	Wyświetlanie diod stanem niskim
	
Ustawienie wyjścia w stan wysoki	Ustawienie wyjścia w stan niski
DDRD |= (1<<PD1);	DDRD |= (1<<PD2);
PORTD |= (1<<PD1);	PORTD &= ~(1<<PD2);

Efektem jest świecąca się stale dioda LED.

```
#define F_CPU 1000000L
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    /* Początek nieskończonej pętli */
    for(;;)
    {
        DDRD |= (1<<PD1);  //Wyjście stanu wysokiego
       PORTD |= (1<<PD1);
       DDRD |= (1<<PD2);    //Wyjście stanu niskiego
      PORTD &= ~(1<<PD2);
    }
}
```

```
#define F_CPU 1000000L
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    /* Początek nieskończonej pętli */
   // for(;;)
    {
        DDRD |= (1<<PD1);  //Wyjście stanu wysokiego
       PORTD |= (1<<PD1);
       DDRD |= (1<<PD2);    //Wyjście stanu niskiego
      PORTD |= (1<<PD2);
       _Delay_ms(500ms);        //Czas opóźnienia
      PORTD &= ~(1<<PD2);
      PORTD &= ~(1<<PD1);
        _Delay_ms(500ms);
    }
}
```