LABORATORIUM 3. UKŁADY PERYFERYJNE MIKROKONTROLERA AVR. PORTY.

Cel laboratorium:

Zapoznanie studentów z budową i sposobem programowania układów we/wy mikrokontrolera z rdzeniem AVR. Budowa programu wbudowanego z elementami pętli licznikowych oraz instrukcjami warunkowmi. Operacje selektywne na rejestrach specjalnych.

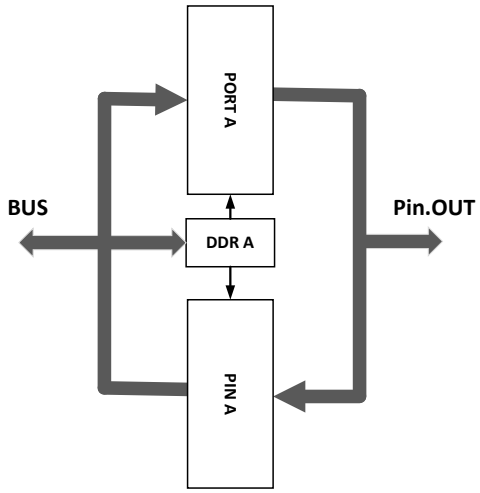
Zakres tematyczny zajęć:

zakres 1, Porty mikrokontrolera ATmega32.

Mikrokontroler ATmega32 zawiera 64 rejestry specjalne (rejestry I/O) i 32 rejestry uniwersalne (Rx). Nazwy rejestrów specjalnych są związane z funkcją jaką te rejestry pełnią, bądź ze sprzętem którego pracę konfigurują (np. rejestr statusu SREG, rejestry DDRx, PORTx, PINx portów I/O,). Natomiast nazwy rejestrów uniwersalnych składają się z numeru rejestru (od 0 do 31) poprzedzonych literą R (np. R16). Rejestry uniwersalne nie są dostępne z poziomu języka C. Dostęp do nich możliwy jest poprzez wstawki asemblerowe. Aby móc korzystać ze zdefiniowanych nazw rejestrów należy program rozpocząć dyrektywą:

```
#include <avr/io.h>
```

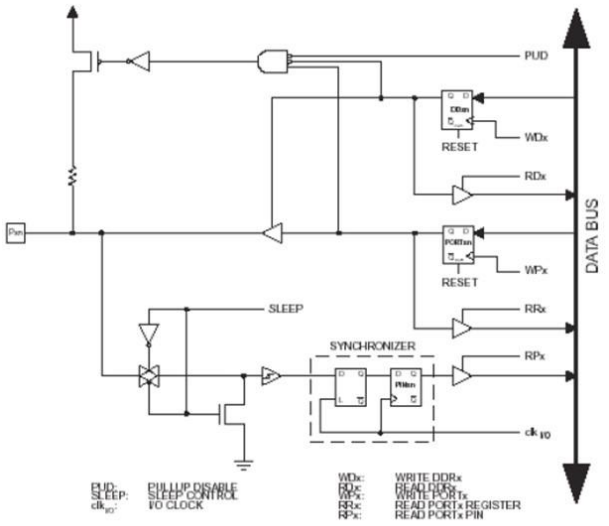
Rejestry specjalne służą do konfigurowania i sterowania pracą peryferii mikrokontrolera (np. portów, liczników, przetwornika analogowo-cyfrowego, itd.). Jednym z podstawowych peryferii każdego mikrokontrolera są porty wejścia/wyjścia. Służą one między innymi do wysyłania i odbierania danych przez MCU. Mikrokontroler ATmega32 wyposażony jest w 4 porty (A, B, C, D). Do konfiguracji kierunku portu (wejście/wyjście) służy rejestr specjalny DDRx (Data Direction Register, gdzie x to nazwa portu np. dla portu A rejestr będzie nosił nazwę DDRA).



Rys. 3.1. Struktura logiczna portu mikrokontrolera

Jak wynika z funkcji portu do odczytu stanu linii we/wy (Pin.OUT) służy rejestr PINx. Rejestr ten odczytuje stan logiczny będący funkcją wyjścia rejestr PORTx oraz stanów

logicznych na liniach we/wy uzależnionych od cyfrowych układów zewnętrznych dołączonych do tych linii.



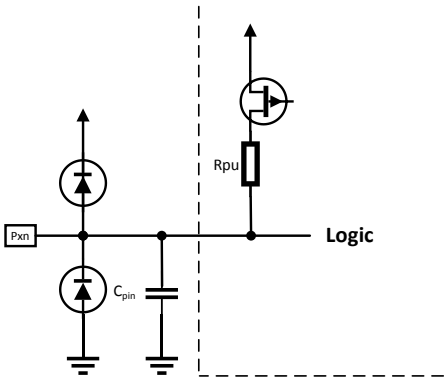
Rys. 3.2. Schemat ideowy portu mikrokontrolera

Tabela 2.1 Konfiguracja wyprowadzeń

DDRn	PORTn	PUD	I/O	pull-up	STAN
0	0	X	In	No	Tri - state
0	1	0	In	Yes	In - Low
0	1	1	In	No	Tri - state
1	0	X	Out	No	Out Low
1	1	X	Out	No	Out High

PUD – bit w rejestrze SFIOR. Umożliwia blokowanie funkcji pull-up dla wszystkich portów.

Każdemu z bitów w rejestrze DDRx odpowiada fizyczna nóżka (pin-out) wejścia/wyjścia mikrokontrolera, znajdująca się na porcie X. Np. bit nr 3 w rejestrze DDRC (czyli PC3) odpowiada nóżce podpisanej „Port C 3” na makiecie dydaktycznej.



Rys. 3.3. Struktura pin-outu portu mikrokontrolera

Końcówki są zabezpieczone przed przepięciami (diody ESD). Istnieje możliwość aktywacji rezystora podciągającego do VCC (pull-up).

Aby skonfigurować daną nóżkę jako wyjście należy wpisać w odpowiednie miejsce w rejestrze DDRC wartość logiczną ′1′. Jeżeli wpisana zostanie wartość logiczna ′0′ to dana linia portu będzie skonfigurowana jako wejście. Do wpisywania wartości do rejestrów służy między innymi instrukcja:

```
DDRA=0xFF;
```

Użyta powyżej instrukcja powoduje przepisanie wartości do rejestru DDRA. Wykonanie tego fragmentu kodu spowoduje wysterowanie wszystkich 8 nóżek portu A (PA0..7) jako wyjścia.

Możliwe jest skonfigurowanie części nóżek portu jako wejścia, części jako wyjścia, np.:

```
DDRA=0x0f;   //konfiguracja wyprowadzeń portu A jako wejścia i
wyjścia
```

Taki zapis spowoduje, że 4 najstarsze nóżki portu A (linie PA7..5) będą wejściami, natomiast pozostałe będą wyjściami (linie PA4..0).

Jeżeli dana nóżka (nóżki, cały port) pracuje jako wejście stany logiczne odpowiadające sygnałom dostarczonym z zewnątrz do linii portu są wpisywane przez mikrokontroler do rejestru PINx. Aby odczytać wartość z portu i zapisać ją do rejestru uniwersalnego należy użyć instrukcji:

```
char x=PINA;
```

Jeżeli dana nóżka (nóżki, cały port) pracuje jako wyjście to wysyłane nią dane należy wpisywać do rejestru PORTx. W wypadku wysyłania danych np. portem A należy użyć rejestru PORTA (uwaga: port A" oznacza jedno z urządzeń wewnątrz mikrokontrolera, „PORTA" oznacza jeden z rejestrów sterujących portem A):

```
PORTA=0b01010101;
```

Wysłanie informacji 0b01010101 przy użyciu portu A.

```
DDRA=0x00;
PORTA=0xFF;
```

Instrukcja ta przy skonfigurowaniu portu jako wejście powoduje podciągnięcie do „1" wejścia portu. W stanie gdy do linii nie są podłączone zewnętrzne sygnały, odczytywany stan portu będzie wynosił: PINA=0xFF;

Wpisanie powyższego programu do mikrokontrolera i podłączenie jego portu A z wejściami sterującymi pracą diod LED – Złącze JP22 spowoduje, że zaświeci się co druga dioda (D1, D3, D5, D7), co druga nie będzie świeciła (D2, D4, D6, D8).
Podsumowując:

- mikrokontroler ATmega32 posiada 4 urządzenia typu ‘PORT’: portA, portB, portC, portD;
- do sterowania każdym z tych urządzeń służą 3 rejestry specjalne: DDRx, PORTx i PINx;
- bity w tych rejestrach oznaczamy w jednolity sposób: PC0, PB2, PA7, PD5, itp.

zakres 2, Operacje selektywne
W języku C jest sześć operatorów bitowych:|,&,^,<<,>>,~. Operatory te są szczególnie użyteczne przy manipulacjach bitami rejestrów. Poniżej na przykładach wyjaśnienie ich użycia. Oczywiście w przykładach liczby przedstawione są w postaci dwójkowej.

operator "|" - bitowa alternatywa (OR)

```
    0 1 0 1 0 1 0 1
|
    0 0 1 1 0 0 1 1
=
    0 1 1 1 0 1 1 1
```

operator "&" - bitowa koniunkcja (AND)

```
    0 1 0 1 0 1 0 1
&
    0 0 1 1 0 0 1 1
=
    0 0 0 1 0 0 0 1
```

operator "^" - bitowa alternatywa wykluczająca (XOR)

```
    0 1 0 1 0 1 0 1
^
    0 0 1 1 0 0 1 1
=
    0 1 1 0 0 1 1 0
```

operator "<<" - przesunięcie w lewo

```
 1 0 0 1 1 0 0 1  << 3 =  1 1 0 0 1 0 0 0
```

operator ">>" - przesunięcie w prawo

```
 1 0 0 1 1 0 0 1  >> 5 =  0 0 0 0 0 1 0 0
```

operator "~" - dopełnienie jedynkowe

```
 ~1 0 0 1 1 0 0 1  =  0 1 1 0 0 1 1 0
```

Operacje selektywne polegają na użyciu funkcji logicznych celem modyfikacji wskazanej w masce numerów bitów w rejestrach mikrokontrolera.

Operacja selektywnego ustawienia:

```
PORTD |= 0xF0;                          /* ustawia bity nr. 4..7 */
PORTD |= _BV(7)|_BV(6)|_BV(5)|_BV(4);
```

Operacja selektywnego zerowania:

```
PORTD &= 0xAA;                        /* zeruje bity nr. 0,2,4,6 */
PORTD &=BV(7)|_BV(5)|_BV(3)|_BV(1);
```

Operacja selektywnego negowania:

```
PORTD ^= 0x0F;                         /* neguje bity nr. 0..3 */
PORTD ^= _BV(3)|_BV(2)|_BV(1)|_BV(0);
```

zakres 3. Pętle licznikowe i instrukcje warunkowe.

W naszych prostych, przykładowych programach będzie można wyróżnić dwie sekcje, pierwsza to instrukcje wykonywane raz jeden, natychmiast po starcie programu, a druga sekcja to blok instrukcji wykonywany wielokrotnie w nieskończonej pętli. Nieskończoną pętle w programie można zbudować używając instrukcji "for" lub "while", w taki sposób, jak w przykładzie poniżej. W obu przypadkach instrukcje we wnętrzu pętli objęte są parą nawiasów klamrowych "{","}", podobnie jak zawartość całej funkcji głównej "main". Instrukcje tworzące pętle to: "for" i "while". Obie instrukcje pozwalają na utworzenie pętli nieskończonej:

```
int main(void)
{
  /* Instrukcje wykonywane raz jeden po starcie programu */

  /* Pętla nieskończona utworzona instrukcją "for" */
  for(;;)
  {
  /* Instrukcje w nieskończonej pętli */
  }
}

/* Pętla nieskończona utworzona instrukcją "while" */
  while(1)
  {
  /* Instrukcje w nieskończonej pętli */
  }
```

Do podejmowania decyzji (sterowania pracą programu) wykorzystywana jest instrukcja "if-else":

```
if( wartość_logiczna PRAWDA/FAŁSZ )

/* Instrukcja wykonywana jeśli PRAWDA */
else
/* Instrukcja wykonywana jeśli FAŁSZ */
```

Część "else" , czyli instrukcje wykonywaną gdy FAŁSZ można pominąć.

```
if( wartość_logiczna PRAWDA/FAŁSZ )
/* Instrukcja wykonywana jeśli PRAWDA */
```

Obejmując fragment kodu parą klamrowych nawiasów "{","}" tworzymy tzw. blok instrukcji, blok w "if-else" jest traktowany jako pojedyncza instrukcja.

```
if( wartość_logiczna PRAWDA/FAŁSZ )
{
/* Blok instrukcji wykonywany jeśli PRAWDA */
}
else
{
/* Blok instrukcji wykonywany jeśli FAŁSZ */
}
```

Chcąc sprawdzić czy jeden lub grupa bitów jednocześnie w rejestrze I/0 ma wartość "1" można to zrobić z użyciem instrukcji "if" w następujący sposób:

```
if(PINA & MASKA)
{
  /* Blok instrukcji wykonywany jeśli warunek spełniony */
}
```

Gdzie "PINA" to nazwa rejestru, a "MASKA" to stała wartość z ustawionymi tymi bitami, które potrzeba testować

Przykłady:

```
/* Jeśli bit nr. 3 rejestru PINA ma wartość "1" */
if(PINA & 0x08){}

/* Jeśli bity 0 lub 1 w PIND mają wartość "1" */
if(PIND & 0x03){}
```

```
unsigned char a, b;

/* Jeśli wartość zmiennej 'a' jest równa wartości zmiennej
'b', w rejestrze PORTA zostanie zapisana wartość 0x01, w
przeciwnym razie w PORTB zostanie zapisana wartość 0x0F */

 if(a==b) PORTA = 0x01; else PORTB = 0x0F;
```

Pytania kontrolne:

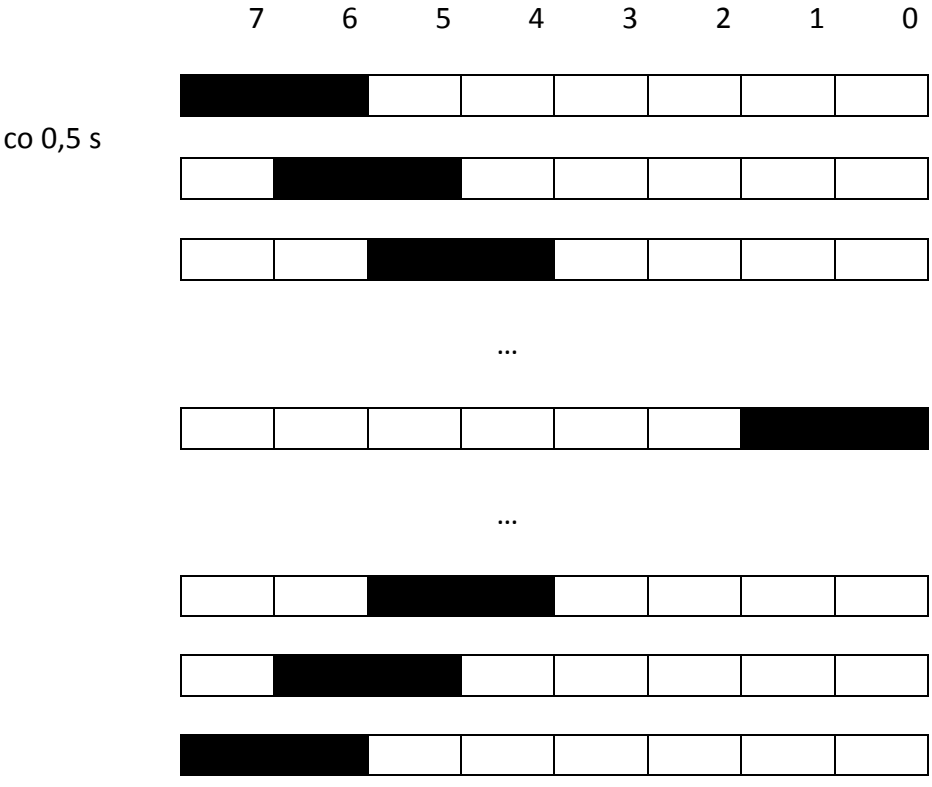
1. Pytanie 1. Co to są operacje selektywne? Selektywnie ustaw bit 0 rejestru DDRA
2. Pytanie 2. W jaki sposób odczytujemy stan portu A. Jaka jest wymagana konfiguracja portu, gdy chcemy odczytać tylko linie 1 i 2.
3. Pytanie 3. W jaki sposób dokonujemy testowania wybranych bitów w rejestrach specjalnych?

Zadanie 3.1. Programowanie portów mikrokontrolera. Operacje selektywne.

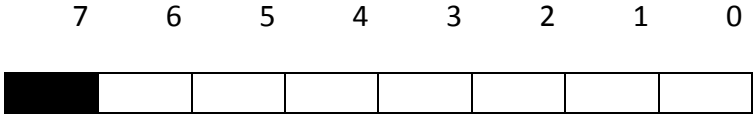
Treść zadania

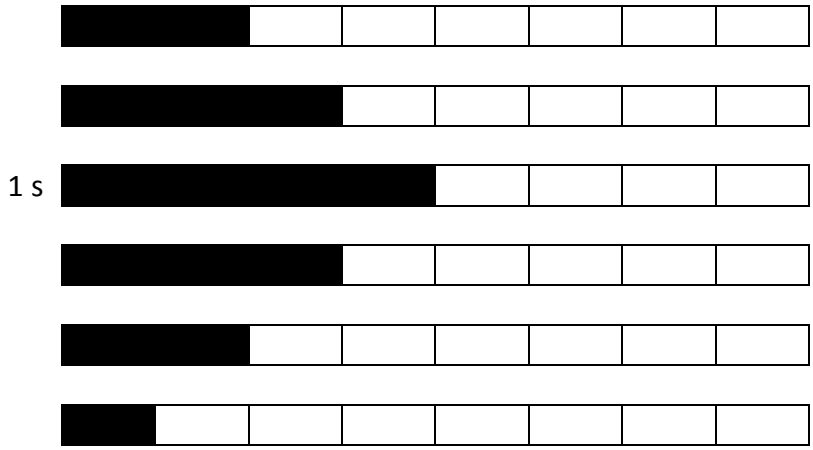
Polecenie 1. Napisz, uruchom i przetestuj program sterujący liniami portu A, do których dołączono linijkę diodową D0…D7. Korzystając wyłącznie z operacji selektywnego: ustawienia, zerowania i negowania napisz kolejne programy w języku C dla poniższych przykładów od 1 do 5 w oparciu o rejestry specjalne Portu A. Do opóźnień czasowych wykorzystaj dostępne w bibliotece: delay.h funkcje opóźnień - _dely_ms(xxx);.
Każdy program należy wykonać w trybie krokowym obserwując (rejestrując) wybrane stany Portu A. W sprawozdaniu należy umieścić: kod źródłowy wraz z opisem oraz kod modułu main() w asm (należy skorzystać z deasemblera dostępnego w AvrStudio).

Przykład1



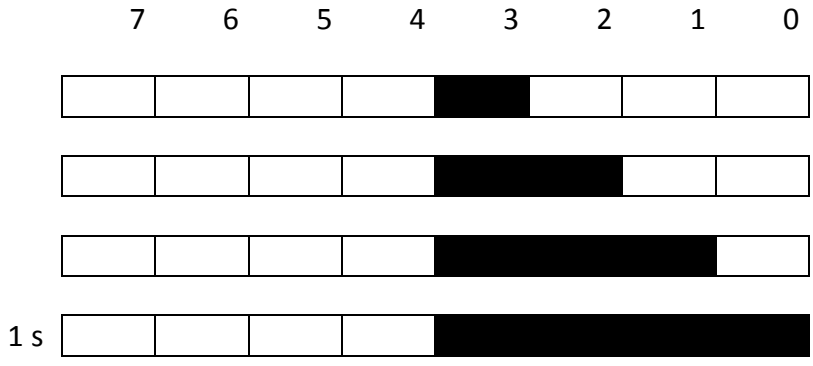
Przykład2

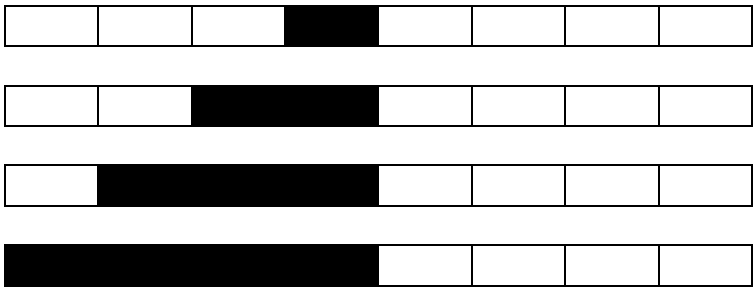




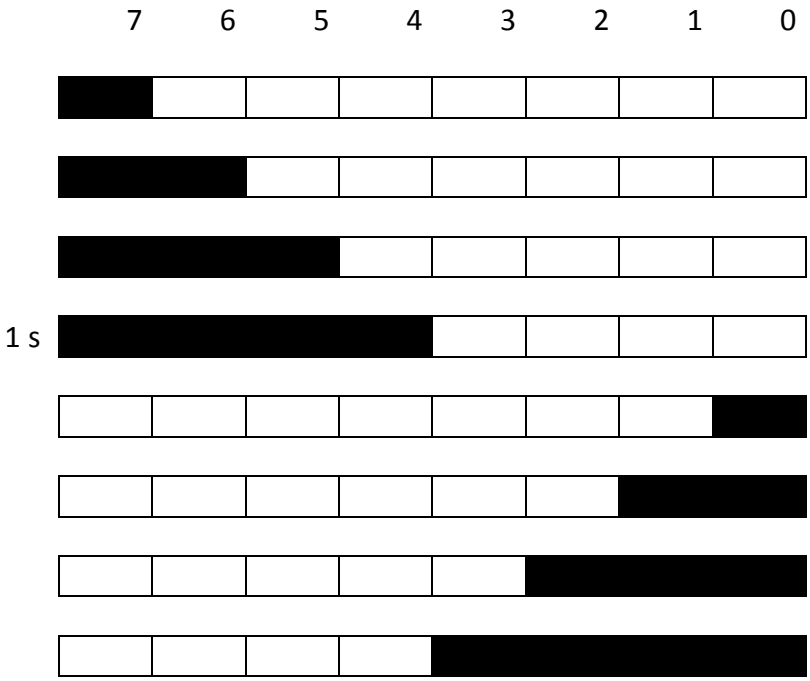
Przykład3

Przykład4





Przykład5



Polecenie 2. Na podstawie testów programu dokonaj rejestracji wybranych stanów portu A. Opracuj sprawozdanie na podstawie wskazówek prowadzącego laboratorium.