Wykład w ramach zajęć Akademia ETI

Programowanie Mikrokontrolerów rodziny AVR ATmega

Inż. Kamil Kujawski
Inż. Krzysztof Krefta

Metody programowania

- Assembler
- Język C
- BASCOM

Assembler – kod maszynowy

- Zalety:
  - Najbardziej efektywny
  - Intencje programisty są realizowane dokładnie
  - Najszybszy i najmniejszy
- Wady:
  - Bardzo długi i trudny czas tworzenia
  - Nieczytelny

Assembler – kod maszynowy

- Zalety:
  - Najbardzi
  - Intencje p ane dokładnie
  - Najszybsz
- Wady:
  - Bardzo dł enia
  - Nieczytel

```
; ============================
;    delay loop generator
;     8000000 cycles:
; ----------------------------
; delaying 7999992 cycles:
          ldi  R17, $48
WGLOOP0:  ldi  R18, $BC
WGLOOP1:  ldi  R19, $C4
WGLOOP2:  dec  R19
          brne WGLOOP2
          dec  R18
          brne WGLOOP1
          dec  R17
          brne WGLOOP0
; ----------------------------
; delaying 6 cycles:
          ldi  R17, $02
WGLOOP3:  dec  R17
          brne WGLOOP3
; ----------------------------
; delaying 2 cycles:
          nop
          nop
; ============================
```

BASCOM AVR

- Zalety:
  - Prosty (?) dla początkującego użytkownika
- Wady:
  - Bardzo nie optymalny (wymaga dużo pamięci FLASH i RAM)
  - Najdłuższy czas wykonywania
  - Niestandardowy (wyjątkowy dla AVR)
  - Programista ma niewielki wpływ na to jak kompilator wykona polecenie.

Język C

- Zalety:
  - Standard przemysłowy / dużo bibliotek
  - Optymalny stosunek szybkość tworzenia do wykonywania kodu
  - Pozwala pracować na danych abstrakcyjnych
- Wady
  - Programista nie ma pełnego wpływu na to jak kompilator zrealizuje / zoptymalizuje polecenie (czasami powoduje błędy)

Operacje bitowe na rejestrach i danych

Praca z danymi w języku C

Operacje bitowe

- Chcąc wpisać jedną ‘jedynkę’ na jednym z pól rejestru (bądź też dowolnej danej) musimy się posłużyć operacjami bitowymi (eng. Bitwise), nie ma gotowego polecenia „wpisz na pozycję 4tą jedynkę”, ale jest na to bardzo prosty sposób…

Operatory logiczne

- W warunkach stosuje się podwójne operatory „ if(a && b) ” lub „ if(a || b) ”

- W równaniach stosuje się pojedyncze operatory:
- X = (a|b)&c;

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

Lub w C zapisujemy jako „ | „

Przykładowo:

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Lub w C zapisujemy jako „ | „

Przykładowo:

0 | 0 = 0

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

Lub w C zapisujemy jako „ | „

Przykładowo:

0 | 1 = 1

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

Lub w C zapisujemy jako „ | „

Przykładowo:

1 | 0 = 1

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

Lub w C zapisujemy jako „ | „

Przykładowo:

0 | 0 = 0

Itd…

Logiczne lub (wstawianie ‘1’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(|| lub) maska	0	1	0	0	0	0	0	0
Dana wyjściowa	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Procesor przeprowadza operację „lub” logicznego dla każdego bitu.

Lub w C zapisujemy jako „ | „

Przykładowo:

0 | 0 = 0

Itd…

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

‘i’ w C zapisujemy jako „ & „

Przykładowo:

0 & 0 = 0

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

‘i’ w C zapisujemy jako „ & „

Przykładowo:

0 & 1 = 0

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

‘i’ w C zapisujemy jako „ & „

Przykładowo:

1 & 0 = 0

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

‘i’ w C zapisujemy jako „ & „

Przykładowo:

0 & 0 = 0

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

‘i’ w C zapisujemy jako „ & „

Przykładowo:

1 & 1 = 1

Logiczne ‘i’ (wstawianie ‘0’)

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
(&& and) maska	0	1	0	0	1	1	0	0
Dana wyjściowa	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

X	Y	OUT
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Procesor przeprowadza operację „i” logicznego dla każdego bitu.

‘i’ w C zapisujemy jako „ & „

Przykładowo:

1 & 0 = 0

Itd…

Logiczne XOR

Dana wejściowa	0	0	1	0	1	**1**	1	0
(^ XOR) maska	0	0	0	0	0	**1**	0	0
Dana wyjściowa	0	0	1	0	1	**0**	1	0
(^ XOR) maska	0	0	0	0	0	**1**	0	0
Dana wyjściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**

X	Y	OUT
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Procesor przeprowadza operację „wyłącznego lub (exclusice or)”, daje ona 1 tylko gdy stany są różne.

‘XOR’ w C zapisujemy jako „ ^ „

Logiczne przesunięcie w lewo

Dana wejściowa	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
Przesunięcie (shift)								3
Dana wyjściowa	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

Procesor przeprowadza przesunięcia bitowego w lewo będącego mnożeniem przez 2. Przesuwa Ona wartość w lewo.

W C zapisuje się tę operację jako <<

Przykładowo powyższe przesunięcie to:

Wyjście=(dana<<3)

Analogicznie działa to w drugą stronę dla przesunięcia w prawo (dzielenia przez dwa).

Ustawianie bitów w języku C

```
Uint8_t set_bit(uint8_t dana, uint8_t bit)
{
  uint8_t out;
  out = (dana | (1<<bit));
  return (out);
}
```

Zmiana stanu bitów w języku C

```
Uint8_t tog_bit(uint8_t dana, uint8_t bit)
{
  uint8_t out;
  out = (dana^(1<<bit));
  return (out);
}
```

GPIO – General Purpouse Input Output

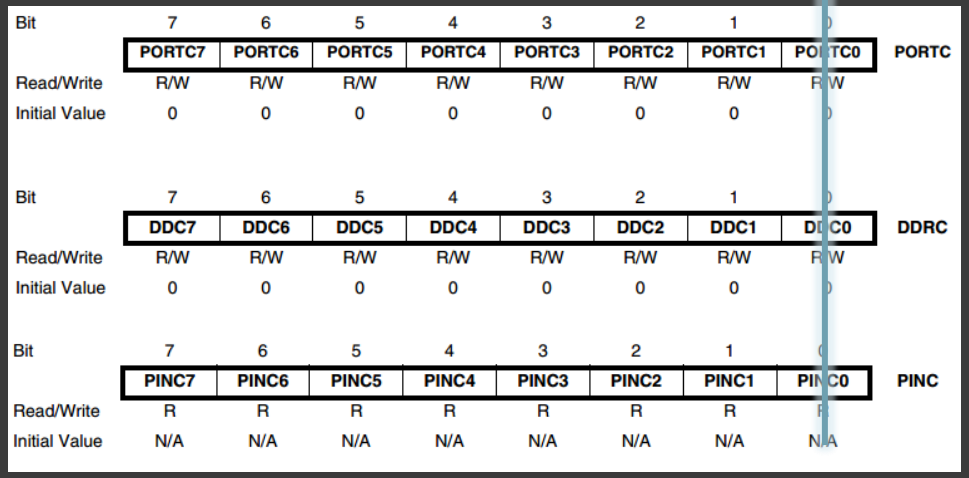
Sterowanie wyjściami mikrokontrolera serii AVR w języku C

Rejestry sterujące

- Jak każdym peryferium mikrokontrolera tak i modułem wyjść/wejść steruje się za pomocą rejestrów. W układach z rdzeniem AVR używane do tego są 3 rejestry DDRx, PORTx, PINx (gdzie X oznacza numer portu np. PORTA lub PORTC).

Rejestry sterujące

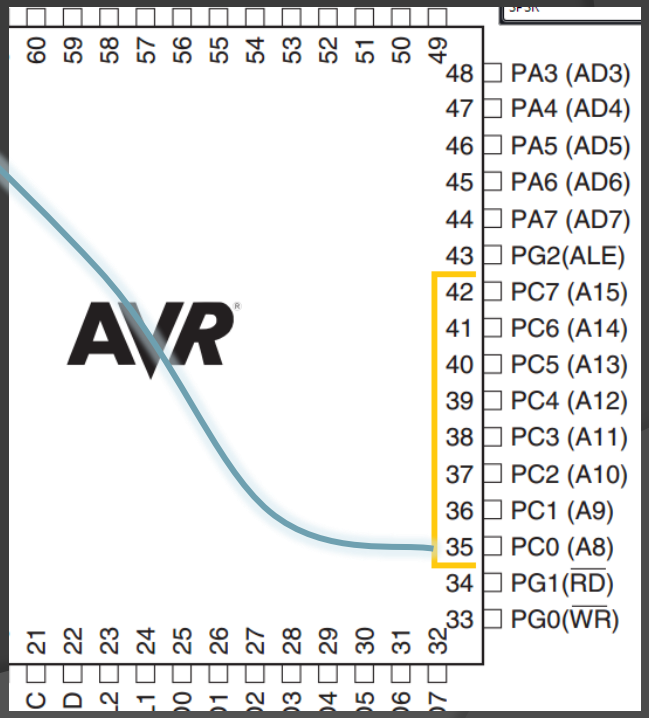
- Każdy bit odpowiada odpowiedniemu wyprowadzeniu (nóżce) układu.



Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
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	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
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	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
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	



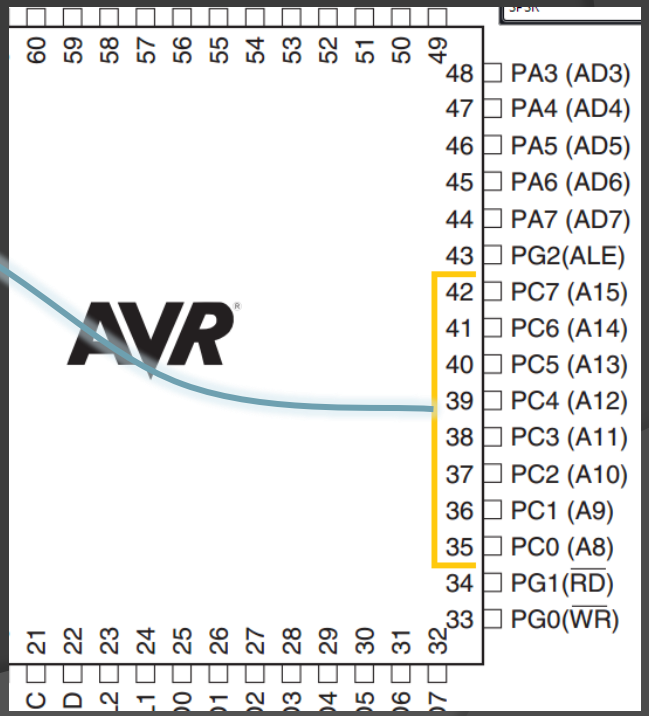
Rejestry sterujące

- Każdy bit odpowiada odpowiedniemu wyprowadzeniu (nóżce) układu.

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
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	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
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	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
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	



Rejestry sterujące

- DDRx – Data Direction Register
  - Steruje kierunkiem przepływu danych (wyjście/wejście). W stanie wejściowym ustawia pin w stan wysokiej impedancji tak aby jego sygnał był słaby.
- PORTx
  - Ustawia wyjścia portu (wysoki / niski)
- PINx
  - Czyta stan z portu (wysoki / niski)

Rejestry sterujące

- Przykłady konfiguracji:

  - DDRB =0x01=0b00000001
  - PORTB=0x03=0b00000011

  //pierwszy pin służy jaki silne wyjście, drugi jako wejście ze słabym podciągnięciem do masy. Obie końcówki łączymy z masą przez rezystor 1k Ohm.

  - PINB=0x01=0b00000001

Ustawienie wyjść i wejść układu

```
Void main()
{
   //chcemy aby piny 5 6 7 portu E były wejściami,
   //ale 0 1 2 był wyjściami (3 i 4 bez znaczenia).
   //5 Ma być ze słabym podciągnięciem do 5V, a 6 i
   //7 do masy.
   //Piny 0 1 i 2 mają mieć wartość 101.
   DDRE= 0x03;
   PORTE |=((1<<0)|(1<<2)|(1<<5));
   while(1);
   Return (0);

}
```

Czytanie wyjść i wejść układu

```
Uint8_t sprawdz_portb()
{
   //chcemy sprawdzić stan wejść
   //mikrokontrolera, nie ważne jak
   //ustawione są dane wyprowadzenia
   //(wejścia czy wyjścia) i tak możemy je
   //sprawdzić. W danym momencie '1' jest na
   //5 i 6 nóżce.
      char out = PINB;
      Return(out);
}
```

Przykładowy kod migający:

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

#define F_CPU 16000000

int main(void)
{

    DDRD=0x00;
    PORTD=0xFF;
    DDRB=0xFF;
    PORTB=0xFF;
     while(1)
     {
         _delay_ms(1000);
         PORTB^=0xFF;
    }
    return;
}
```