POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

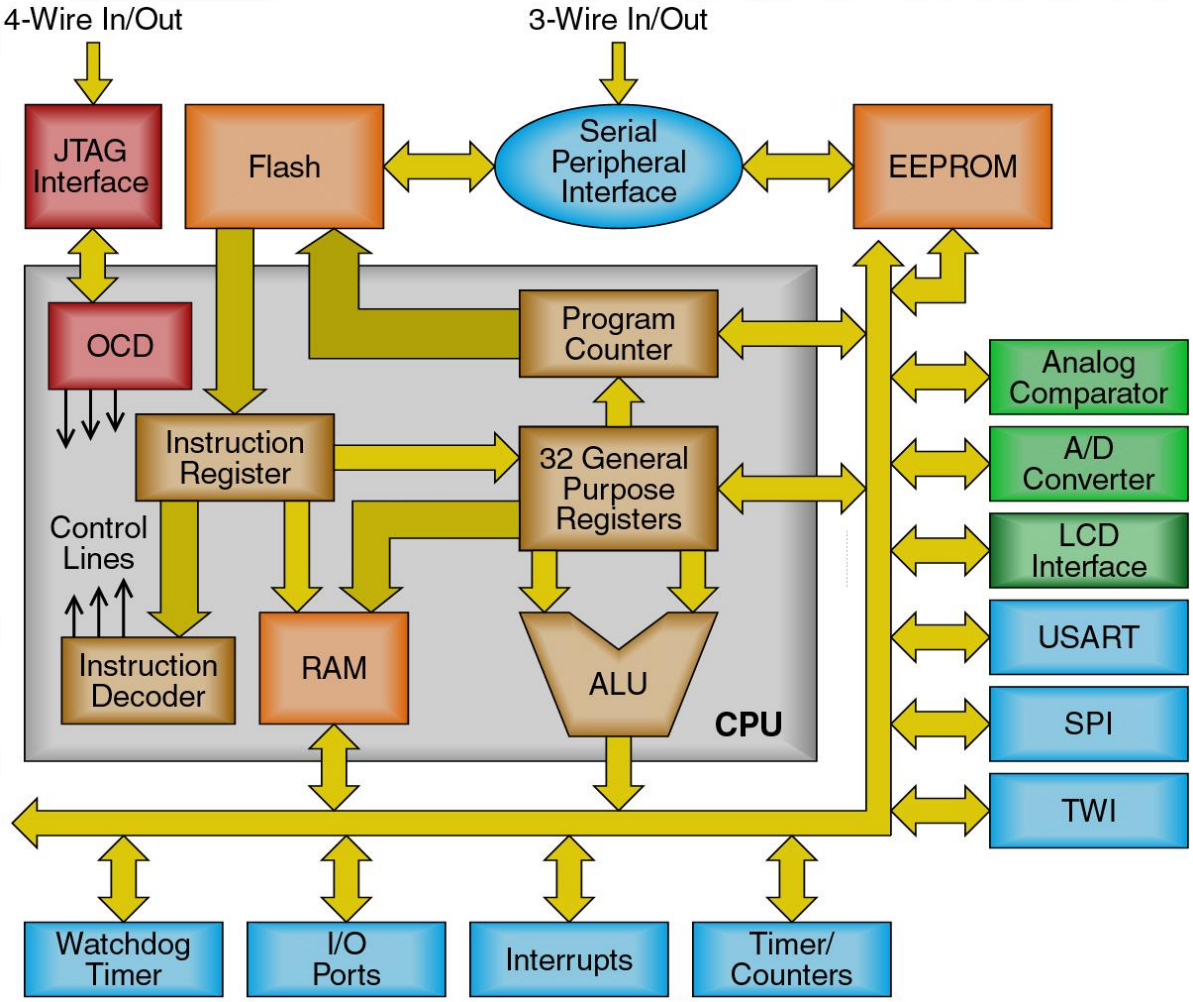
Systemy wbudowane

Układy peryferyjne mikrokontrolera AVR – porty, timery, system przerwań, zarządzanie energią, układy transmisji szeregowej

Dr inż. Wojciech Surtel

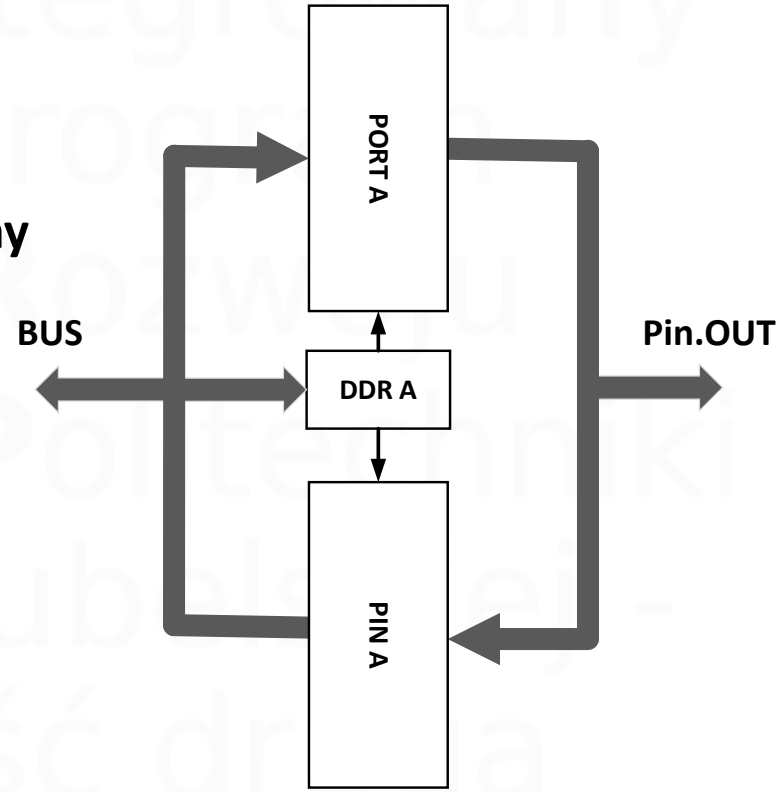
Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Struktura portu – AVR



Struktura portu – AVR

- Wszystkie porty AVR mają funkcjonalność : Read/Modify/Write
- Każdy pin w porcie może być modyfikowany selektywnie
- Rejestrom przydzielono trzy adresy w pamięci We/Wy
- dla każdego portu
  - rejestr danych — PORTx (odczyt/zapis)
  - rejestr kierunku danych — DDRx (odczyt/zapis)
  - piny wejściowe portu – PINx (odczyt)



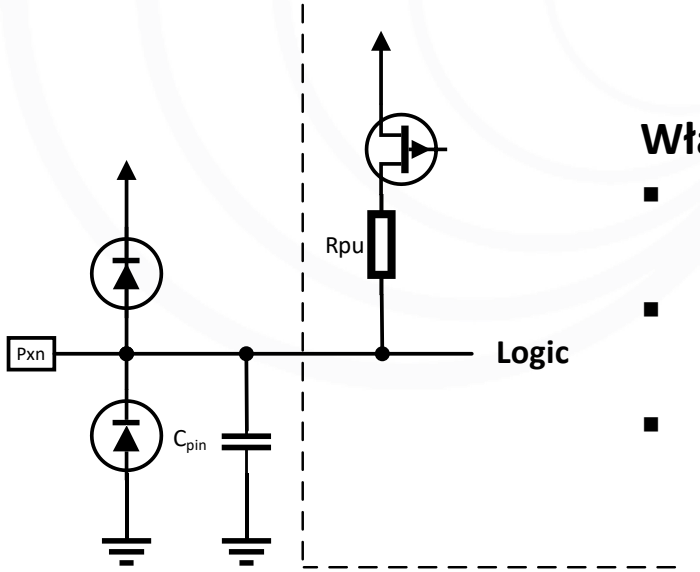
Układy peryferyjne mikrokontrolera AVR

Struktura portu – AVR

Atmega16 (32) posiada cztery 8-bitowe porty: PORTA – PORTD
Każdy port posiada trzy rejestry specjalne o adresach odwzorowanych w pamięci We/Wy:

- DDRn – ustalenie kierunku sygnału (1-wyjście, 0 – wejście)
- PORTn – rejestr wyjściowy, określający stan na końcówce lub aktywacja rezystora podciągającego Rpu
- PINn – rejestr służacy do odczytu stanu portu

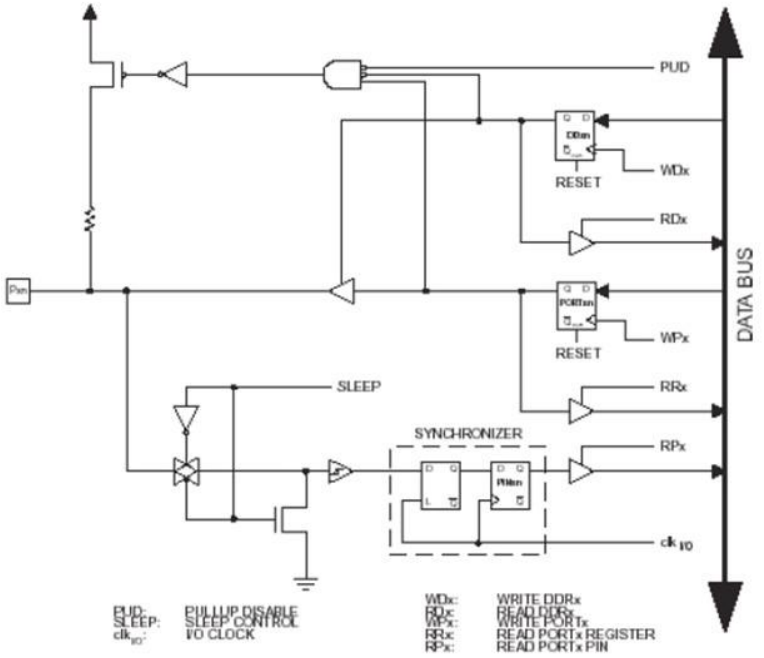
gdzie: n – oznaczenie portu (A,B,C,D)



Właściwości:

- Każda z końcówek portu może być indywidualnie ustawiona jako „We” lub „Wy”
- Końcówki są zabezpieczone przed przepięciami (diody ESD)
- Możliwość aktywacji rezystora podciągającego do V_{CC} (pull-up)

Struktura portu – AVR



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.

Cechy portu – AVR

- **Port jako wejście:**
  - **histereza** (około 50 mV), pozwalająca na eliminację błędów przy sygnałach wolnozmiennych i zaszumionych,
  - przy odczycie portu po jego zapisaniu należy odczekać około 1 takt zegara (wewnętrzny układ synchronizujący).
- Port jako wyjście:
  - stan pinu może się pojawiać z opóźnieniem jednego taktu zegara przy zmianie PORTxn,
  - **typowe obciążenie** linii portu wynosi 20mA, maksymalnie 40mA.

Timery/Liczniki - AVR

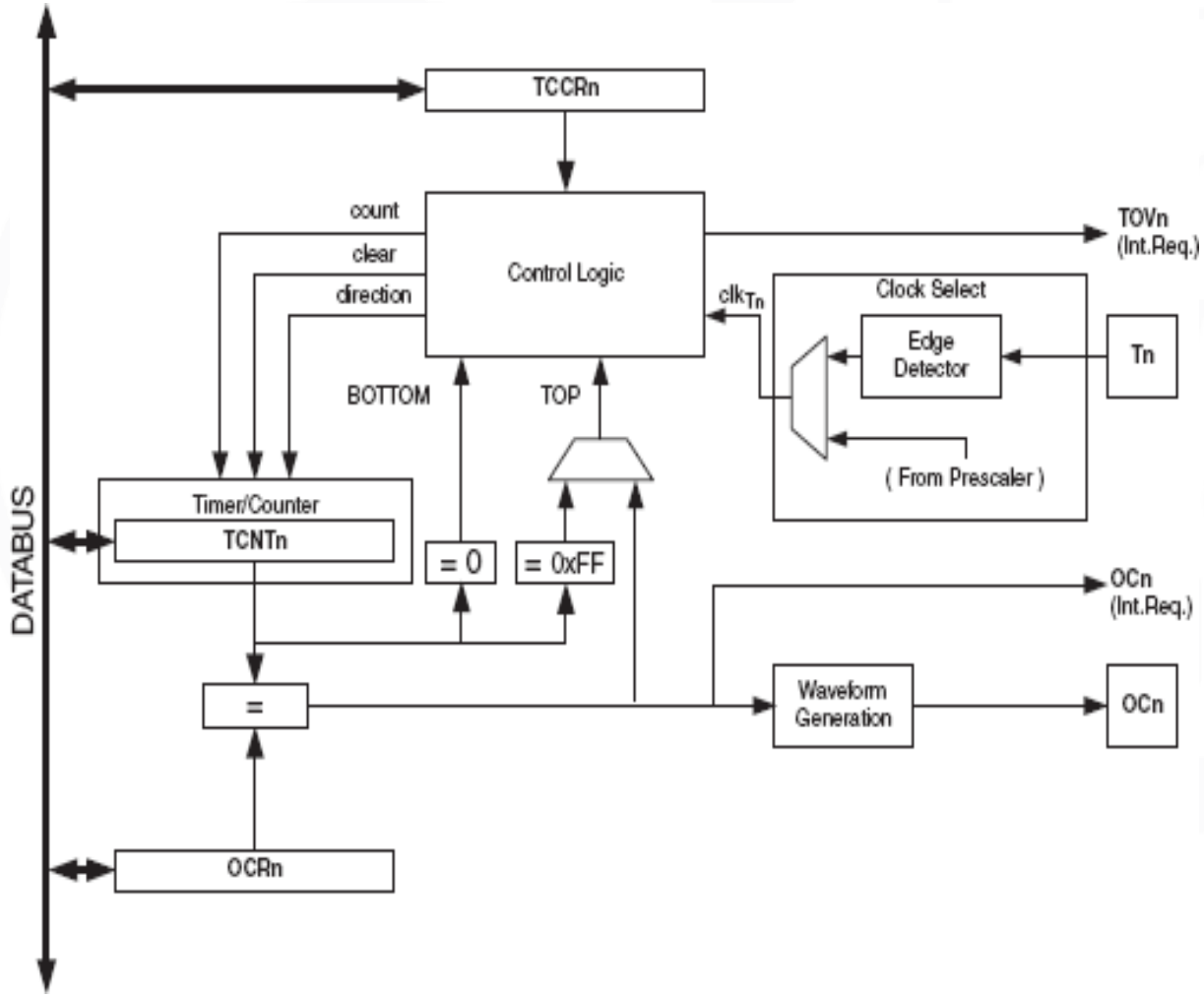
- 2 liczniki 8-bitowe
- liczanie impulsów wewnętrznych i zewnętrznych, odczytanie i zapisanie wartości licznika
- ustalenie zakresu licznika
- porównanie bieżącej wartości licznika z zadaną wartością, przełączenie napięcia portu binarnego
- 1 licznik 16-bitowy - dodatkowo
- porównanie wartości licznika z dwoma zadanymi wartościami
- rejestracja wartości licznika w momencie pojawienia się impulsu zewnętrznego

Timery/Liczniki – AVR: 8-mio bitowy licznik czasomierz T0 z funkcją PWM

Cechy

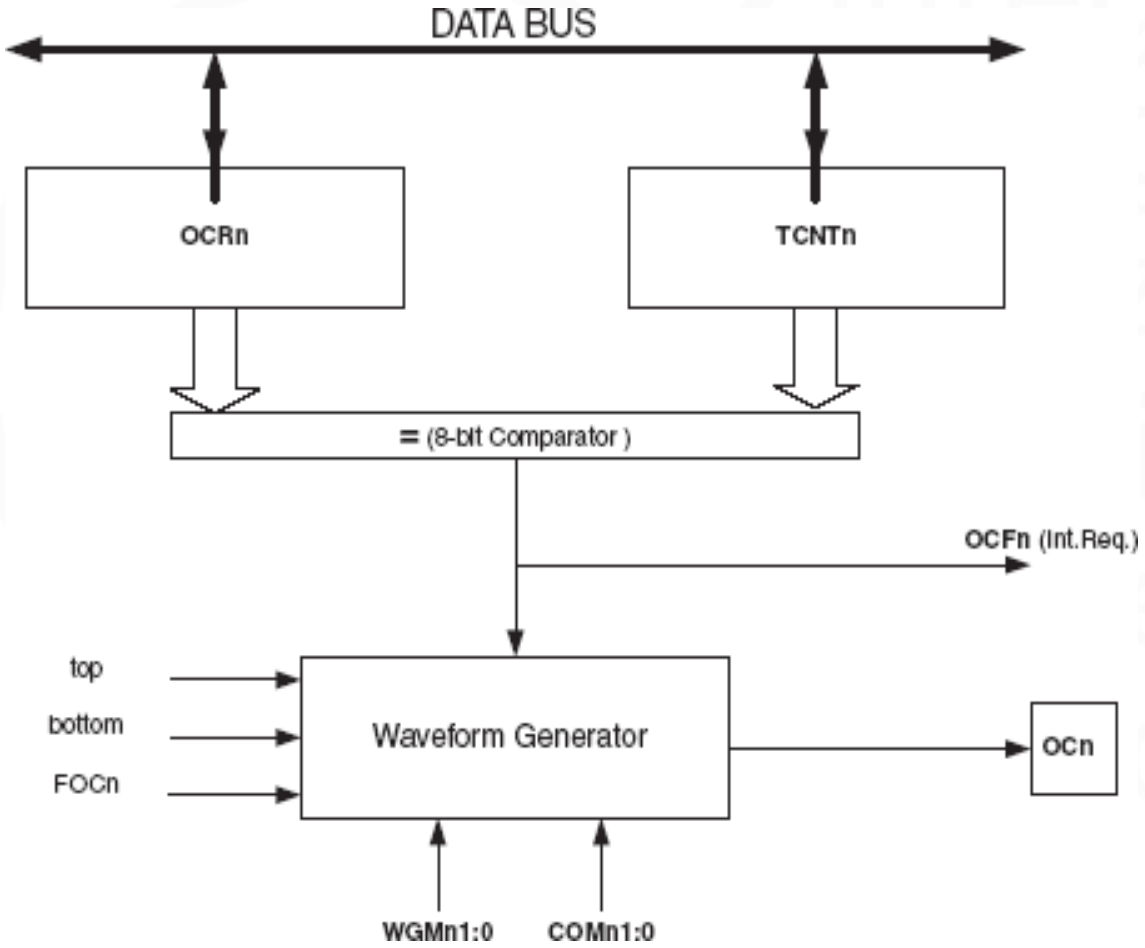
- Licznik z pojedynczym komparatorem
- Tryb CTC (Clear Timer on Compare)
- Tryb PWM z korekcją fazy
- Tryb PWM (fast PWM)
- Zewnętrzny licznik zdarzeń
- 10-bitowy Prescaler (1, 8, 64, 256, 1024)
- Flagi przerwań TOV0 – flaga przepełnienia; OCF0 – flaga porównania (komparatora)

Timery/Liczniki – AVR: 8-mio bitowy licznik czasomierz T0 z funkcją PWM



Schemat blokowy licznika T0

Timery/Liczniki - AVR



Praca licznika T0 z funkcją Output Compare

Timery/Liczniki – AVR: Rejestry układu czasowo-licznikowego T0 Rejestr kontrolny licznika T0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

Bity ustawiające źródło sygnału taktującego i podział prescalera:

CS02	CS01	CS00	Opis
0	0	0	Licznik zatrzymany
0	0	1	Taktowanie CK
0	1	0	Taktowanie CK/8
0	1	1	Taktowanie CK/64
1	0	0	Taktowanie CK/256
1	0	1	Taktowanie CK/1024
1	1	0	Zewnętrzny sygnał T0 (opadające zbocze)
1	1	1	Zewnętrzny sygnał T0 (narastające zbocze)

Timery/Liczniki - AVR

Tryb generacji sygnału

WGM01	WGM00	Opis
0	0	Zwykłą praca licznika
0	1	PWM z korekcją fazy
1	0	Porównanie z zerowaniem licznika
1	1	Szybki PWM

Tryb funkcji Output Compare gdy tryb PWM wyłączony

COM01	COM00	Opis
0	0	Pin odłączony
0	1	Zmiana stanu logicznego na pinie
1	0	Zerowanie pinu
1	1	Ustawienie pinu

Timery/Liczniki - AVR

Rejestr licznika T0- TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

Rejestr porównawczy- OCR0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
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	

Rejestr maski przerwań od liczników czasomierzy-TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

Rejestr znaczników przerwań od liczników-czasomierzy- TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

Timery/Liczniki – AVR: Przykład. Programowa obsługa Timera0 – tryb NORMAL

```
#include <avr/io.h>            //biblioteka zawierająca definicje i rejestry procesora
#include <util/delay.h>        //biblioteka pozwalająca na używanie funkcji opóźnienia

void timer()                   //procedura sprawdzająca przepełnienie flagi i zerująca zegar
{   if(TIFR & (1 << TOV0))     //sprawdzenie stanu flagi przepełnienia
   {                           //jeśli flaga przepełnienia ustawiona, to:
   PORTA ^= _BV(0);            //negacja linii 0 portu A - zaświecenie pierwszej diody
   TCNT0 = 155;                //ustawienie aktualnego stanu zliczonych impulsów licznika w
                                  rejestrze TCNT0
   TIFR |= _BV(TOV0);          //wyzerowanie bitu flagi przepełnienia
   }
}
```

Jak będzie wyglądać procedura, gdy zastosujemy pętlę warunkową *while* ?

Timery/Liczniki – AVR: Przykład. Programowa obsługa Timera0 – tryb NORMAL

```
int main(void)
{
DDRA = 0xFF;                          //ustawienie pinów sterujących portu A w stan -
                                        wyjście
PORTA = 0x00;                         //ustawienie wartości na porcie A - wszystkie diody
                                        zgaszone
TCCR0 |= (1 << CS02) | (1 << CS00);   //ustawienie PRESKALERA - 1024, co odpowiada 1ms
TCNT0 = 155;                          //ustawienie wartości początkowej rejestru TCNT0 –
                                        100 cykli - end
while(1)                              //pętla nieskończona
   { timer();                         //wywołanie procedury timer()
    }
return 0;
}
```

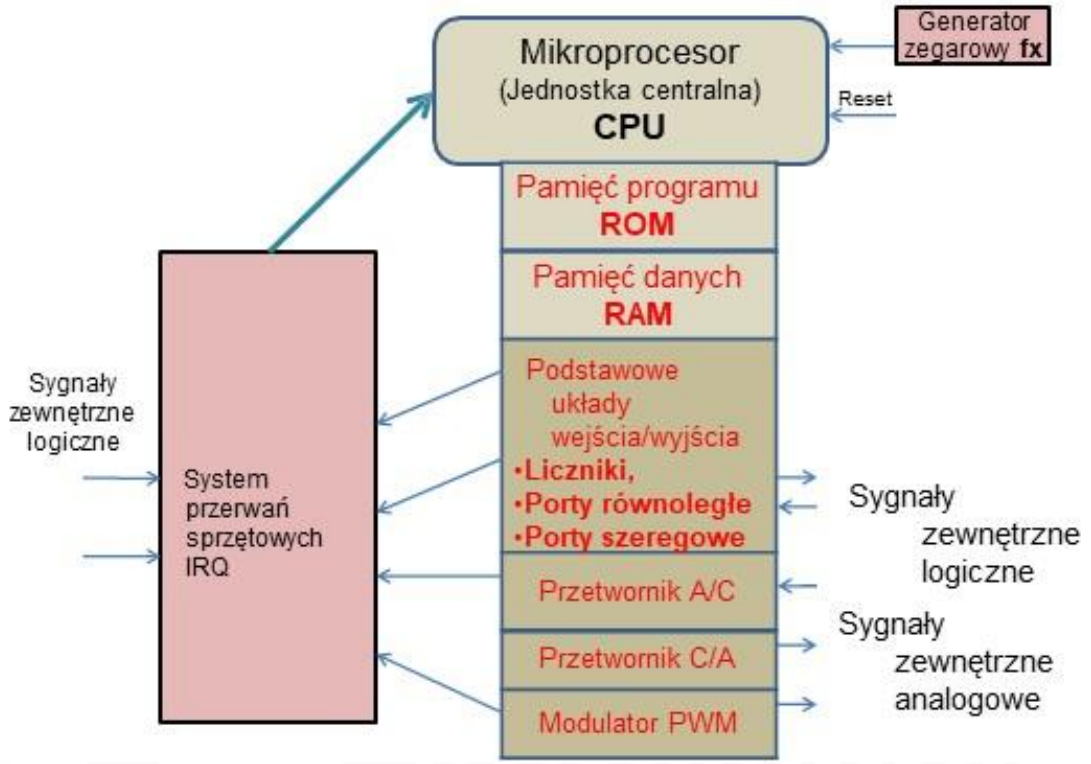
Timery/Liczniki – AVR: Przykład. Programowa obsługa Timera0 – tryb CTC

```
#include <avr/io.h>              //biblioteka zawierająca definicje i rejestry procesora
#include <util/delay.h>          //biblioteka pozwalająca na używanie funkcji opóźnienia

void timer()                     //procedura sprawdzająca przepełnienie flagi i zerująca
                                   zegar
{
   if(TIFR & (1 << OCF0))        //jeśli flaga przepełnienia ustawiona, to wykonaj następne
                                   instrukcje
   {
      PORTA ^= _BV(0);           //zaświecenie pierwszej diody
      TIFR |= _BV(OCF0);         //wyzerowanie bitu flagi przepełnienia, czyli ustawienie 1
   }
}
```

Timery/Liczniki – AVR: Przykład. Programowa obsługa Timera0 – tryb CTC

```
int main(void)
{
DDRA = 0x01;                    //ustawienie pinu portu A sterującego pierwszą
                                  diodą
PORTA = 0x00;                   //ustawienie wartości na porcie A - wszystkie
                                  diody zgaszone
OCR0 =141;                      //ustawienie wartości rejestru OCR
TCCR0 |= (1 << WGM01);          //przestawienie timera w tryb CTC
TCCR0 |= (1 << CS02) | (1 << CS00);   //ustawienie dzielnika częstotliwości

while(1)
{
        timer();                //wywołanie funkcji timer() }
        return 0;
}
```

System przerwań - AVR



Zdarzenia:

- zmiana napięcia podłączonego do wyprowadzenia mikrokontrolera
- przekroczenie zakresu licznika
- odebranie bajtu danych
- zakończenie pomiaru napięcia
- itd.

System przerwań - AVR

- Zmiana stanu pewnych wejść (przerwania INT0, INT1, INT2, ICP1).
- Określony stan pewnych wejść (przerwania INT0, INT1).
- Przepełnienie licznika (TIMER0 OVF, TIMER1 OVF, TIMER2 OVF).
- Osiągnięcie przez licznik zadanej wartości (TIMER0 COMP, TIMER1 COMPA, TIMER1 COMPB, TIMER2 COMP).
- Zakończenie przetwarzania analogowo-cyfrowego (ADC).
- Zakończenie transmisji przez interfejs szeregowy (SPI STC, USART TXC, TWI).
- Odebranie danych przez interfejs szeregowy (USART RXC).
- Gotowość pamięci EEPROM (EERDY).
- Zmiana stanu wyjścia komparatora (ANACOMP).

System przerwań – AVR: Tablica przerwań

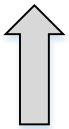
- W standardowej konfiguracji mikrokontrolera 42 najniższe adresy w pamięci programu zajmuje tablica przerwań.
- Im mniejszy adres, tym wyższy priorytet przerwania.
- Najwyższy priorytet ma RESET, potem przerwanie zewnętrzne INT0 itd.
- Przerwania są numerowane od 1 do 21.
- Obsługując przerwanie o numerze x, mikrokontroler ładuje do licznika programu wartość 2(x – 1)
- Powoduje to wykonanie rozkazu spod tego adresu w pamięci programu.
- Najczęściej jest to skok (rjmp lub jmp) do właściwej procedury obsługi przerwania.
- Uwaga: , ATmega16 i ATmega32 obsługują ten sam zbiór przerwań, ale są one inaczej ponumerowane.

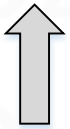
System przerwań - AVR

Przerwania - automatyczne wywołanie funkcji w odpowiedzi na zdarzenie

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

ISR(INT0_vect)
{
  // ...
}
```

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

ISR(INT0_vect, ISR_NOBLOCK)
{
  // ...
}
```





Wykonanie funkcji nie zostanie przerwane przez inne zdarzenie

Wykonanie funkcji może zostać przerwane przez inne zdarzenie, w tym ponowne wywołanie tej samej funkcji

- sei() – uaktywnia obsługę przerwań
- cli() – wyłącza obsługę przerwań (domyślnie po sygnale RESET)

System przerwań - AVR

Makra do zapisywania funkcji obsługi przerwań

```
#define  ISR(wektor, atrybuty)
#define  SIGNAL(wektor)
#define  EMPTY_INTERRUPT(wektor)
#define  ISR_ALIAS(wektor, target_vector)
#define  reti()
#define  BADISR_vect
```

Atrybuty ISR

```
#define  ISR_BLOCK
#define  ISR_NOBLOCK
#define  ISR_NAKED
#define  ISR_ALIASOF(target_vector)
```

System przerwań - AVR

Wektor przerwania Catch-all

Jeśli wystąpi nieoczekiwane przerwanie (przerwanie jest włączone i nie jest zainstalowany program obsługi, co zwykle wskazuje na błąd), domyślną akcją jest zresetowanie urządzenia przez przejście do wektora resetowania.

Można to zastąpić, dostarczając funkcję o nazwie, która powinna być zdefiniowana za pomocą ISR() - nazwa BADISR_vect jest w rzeczywistości aliasem __vector_default.

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

ISR(BADISR_vect)
{
    kod użytkownika tutaj
}
```

System przerwań - AVR

Dwa wektory dzielące ten sam kod

W niektórych okolicznościach działania, które należy podjąć w przypadku dwóch różnych przerw, mogą być całkowicie identyczne, więc wystarczyłoby jedno wdrożenie ISR.
Na przykład: przerwania zmiany pinów przychodzące z dwóch różnych portów mogą logicznie sygnalizować zdarzenie, które jest niezależne od rzeczywistego portu (a tym samym wektora przerwania), w którym się wydarzyło.
Udostępnianie kodu wektora przerwania można wykonać za pomocą atrybutu ISR_ALIASOF() do makra ISR:

```
ISR(PCINT0_vect)
{
  ...
  Kod do obsługi zdarzenia.
}
ISR(PCINT1_vect, ISR_ALIASOF(PCINT0_vect));
```

System przerwań - AVR

Procedury obsługi pustych przerwań

W rzadkich przypadkach wektor przerwania nie wymaga żadnej insujmowania kodu. Wektor i tak musi być zadeklarowany, więc gdy przerwanie zostanie wyzwolone, nie wykona kodu BADISR_vect (który domyślnie uruchamia ponownie aplikację).

Może tak być na przykład w przypadku przerwań, które są włączone wyłącznie w celu usunięcia kontrolera z sleep_mode().

Program obsługi takiego wektora przerwania można zadeklarować za pomocą makra EMPTY_INTERRUPT():

```
EMPTY_INTERRUPT(ADC_vect);
```

`Uwaga:` Nie ma ciała do tego makro.

System przerwań - AVR

W niektórych okolicznościach wygenerowany przez kompilator prolog i epilog ISR (odpowiedzialny za zapisywanie i przywracanie stanów) może nie być optymalny dla zadania. Być może rejestry stanu nie muszą być zapisywane i przywracane przez ISR.

Można rozważyć ręcznie zdefiniowanie makra ISR, w szczególności w celu przyspieszenia obsługi przerwań. Można to zrobić w następujący sposób.

```
ISR(TIMER1_OVF_vect, ISR_NAKED)
{ PORTB |= _BV(0);  // results in SBI which
          // does not affect SREG
reti(); }
```

Makro ISR() nie może tak naprawdę sprawdzić pisowni przekazanego im argumentu. W ten sposób, błędnie pisząc jedną z poniższych nazw w wywołaniu ISR(), zostanie utworzona funkcja, która, choć może być użyteczna jako funkcja przerwania, nie jest w rzeczywistości podłączona do tabeli wektorów przerwań. Kompilator wygeneruje ostrzeżenie, jeśli wykryje podejrzanie wyglądającą nazwę funkcji ISR() (tj. takiej, która po wymianie makr nie zaczyna się od "__vector_").

System przerwań – AVR: przerwania zewnętrzne

Trzy źródła przerwań zewnętrznych- wyprowadzenia: INT0, INT1 INT2

Bity sterujące przerwaniami INT0 i INT1 w rejestrze MCUCR:

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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 sterowania przerwaniem INT2 w rejestrze MCUCSR

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

System przerwań – AVR: Sposób zgłaszania przerwania INT0

Sposób zgłaszania przerwania INT0:

ISC01	ISC00	Sposób zgłaszania przerwania
0	0	Zgłaszanie niskim poziomem logicznym
0	1	Zgłaszanie negacją stanu logicznego
1	0	Zgłaszanie opadającym zboczem
1	1	Zgłaszanie narastającym zboczem

Sposób zgłaszania przerwania INT1:

ISC11	ISC10	Sposób zgłaszania przerwania
0	0	Zgłaszanie niskim poziomem logicznym
0	1	Zgłaszanie negacją stanu logicznego
1	0	Zgłaszanie opadającym zboczem
1	1	Zgłaszanie narastającym zboczem

System przerwań – AVR: sterowanie przerwaniami zewnętrznymi

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **INT1**- bit maski przerwania INT1 (INT1=”1” i bit I=„1” przerwanie INT1 odmaskowane, INT1=„0”- zamaskowane).
- **INT0**- bit maski przerwania INT0 (INT0=”1” i bit I=„1” przerwanie INT0 odmaskowane, INT0=„0”- zamaskowane).
- **INT2**- bit maski przerwania INT2 (INT2=”1” i bit I=„1” przerwanie INT2 odmaskowane, INT2=„0”- zamaskowane).

System przerwań – AVR: sterowanie przerwaniami zewnętrznymi

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **INTF1**: bit zgłoszenia przerwania na wejściu INT1, ustawiany gdy przerwanie jest odmaskowane i zgłoszone, kasowany po wejściu do procedury obsługi lub poprzez zapis jedynki logicznej. Gdy przerwanie jest aktywne poziomem bit ten nie jest ustawiany.
- **INTF0**: bit zgłoszenia przerwania na wejściu INT0, ustawiany gdy przerwanie jest odmaskowane i zgłoszone, kasowany po wejściu do procedury obsługi lub poprzez zapis jedynki logicznej. Gdy przerwanie jest aktywne poziomem bit ten nie jest ustawiany.
- **INTF2**: bit zgłoszenia przerwania na wejściu INT2, ustawiany gdy przerwanie jest odmaskowane i zgłoszone, kasowany po wejściu do procedury obsługi lub poprzez zapis jedynki logicznej.

System przerwań – AVR: Przykład

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

ISR(INT0_vect) {
  volatile uint16_t t;
  for (t = 0; t < 200; ++t);
  if ((PIND & (1 << PD2)) == 0)
  PORTA ^= 1 << PA0;
}
 ISR(INT2_vect) {
  PORTA ^= 1 << PA2;
}
```

System przerwań – AVR: Przykład

```
int main(void) {

   uint8_t tmp;
    DDRA = 1 << PA0 | 1 << PA2;
   tmp = MCUCR;
   tmp &= ~(1 << ISC00);
   tmp |= 1 << ISC01;
   MCUCR = tmp;
   MCUCSR &= ~(1 << ISC2);
   GICR |= 1 << INT0 | 1 << INT2;

   sei();

while(1) {}
}
```

System przerwań – AVR: Przykład

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

volatile int licz = 0;                  //Licznik wykonanych przerwań
volatile char przerwanie =0;            //Zmienna sygnalizująca wystąpienie przerwania

ISR(TIMER0_OVF_vect){                   //Przerwanie wywołane przepełnieniem Timera0
        przerwanie = 1;                 //Sygnalizacja, że przerwanie zostało wywołane
        TCNT0 = 155;                    //Przywrócenie wartości Timera0 na 155
}

int main(void){

DDRA = 0b00000001;                      //Ustawienie bitu zero w porcie A na wyjście
PORTA = 0x00;                           //Wyzerowanie portu A
TCCR0 |= (1<<CS00)|(1<<CS02);           //Ustawienie preskalera na dzielnik 1024
TIMSK |= 1<<TOIE0;                      //Zezwolenie na przerwanie przy ustawieniu flagi
                                        //TOV0
TCNT0 = 155;                            //Wartość początkowa licznika ustawiona tak, aby
                                        //Po odliczeniu do stu licznik ustawiał flagę TOV0
sei();                                  //Włączenie zezwolenia na przerwanie
```

System przerwań – AVR: Przykład

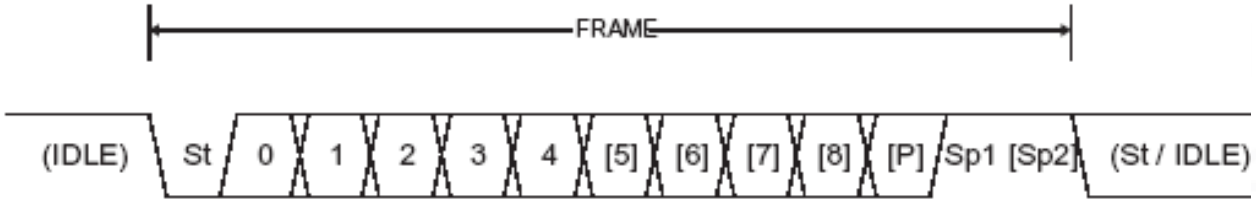
```
while(1){

    if(przerwanie == 1                  //Jeżeli wystąpiło przerwanie
        licz++;                         //Zwiększ licznik przerwań o jeden
        if(licz == 20){                 //Jeżeli licznik przerwań będzie równy 20
        licz =0;                        //Wyzeruj licznik przerwań
        PORTA ^= 0b00000001;            //I zmień stan portu A na zerowym bicie.
                        }
        przerwanie = 0;                 //Wyzeruj wystąpienie przerwania
                            }
}}
```

System przerwań – AVR: Przykład

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

volatile int licz = 0;                  //Licznik wykonanych przerwań
volatile char przerwanie =0;            //Zmienna sygnalizująca wystąpienie przerwania

ISR(TIMER0_OVF_vect){                   //Przerwanie wywołane przepełnieniem Timera0
        OCR0++;                         //Zwiększenie OCR0 o 1 (dioda świeci jaśniej)
                                }

int main(void){

DDRB = 0b00001000;                       //Ustawienie bitu trzeciego w porcie A na wyjście

//Konfiguracja Timera0, tryb FastPWM, preskaler 1024:

TCCR0 |= (1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<COM00)|(1<<CS01)|(1<<CS02);
TIMSK |= 1<<TOIE0;                      //Zezwolenie na przerwanie przy ustawieniu flagi
                                        //TOV0
TCNT0 = 0;                              //Wyzerowanie TCNT

sei();                                  //Włączenie zezwolenia na przerwania
OCR0 = 0;                               //Początkowe ustawienie OCR0 na 0 (dioda zgaszona)

        while(1){}

}
```

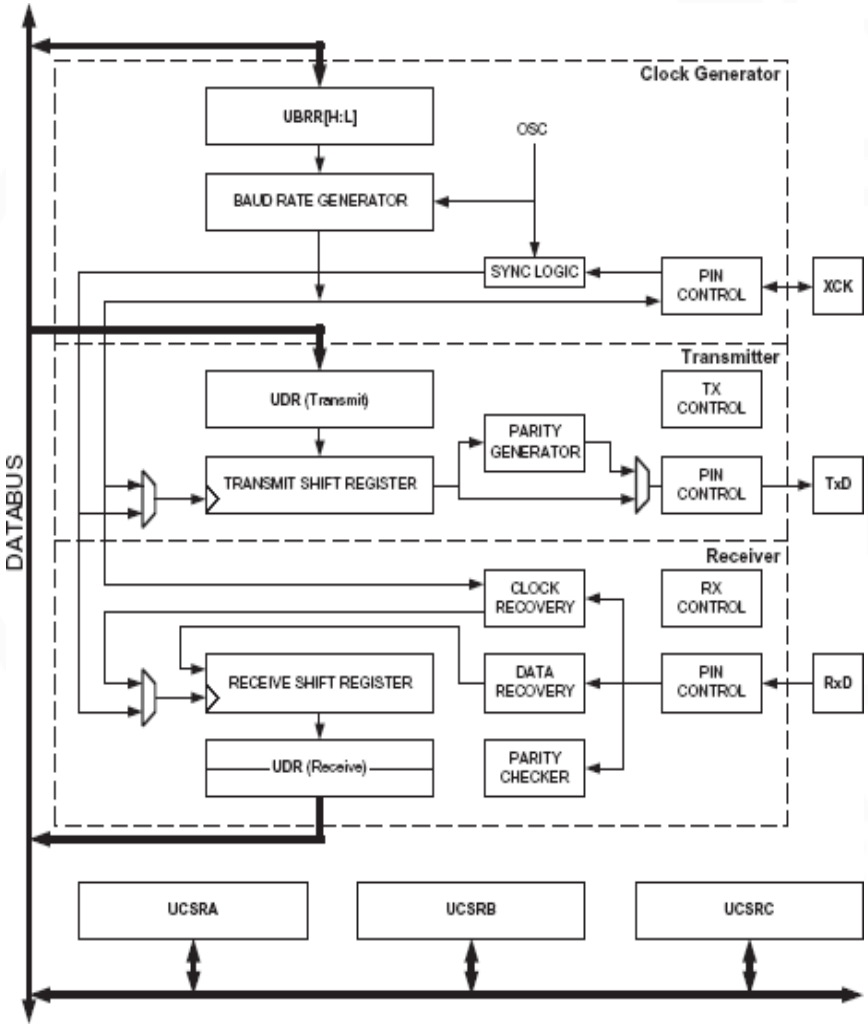
Port szeregowy synchroniczny-asynchroniczny USART

- Możliwość pracy synchronicznej i synchronicznej
- Ramka od 5 do 9 bitów
- 1 lub 2 bity stopu
- Dwa rodzaje kontroli parzystości
- Wykrywanie błędu ramki
- Eliminacja szumów
- Możliwość współpracy wieloprocesorowej



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Port szeregowy synchroniczny-asynchroniczny USART



Schemat blokowy portu USART

Port szeregowy synchroniczny-asynchroniczny USART

Rejestry: nadawczy i odbiorczy portu USART- UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
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	

Rejestr nadawczy i odbiorczy znajdują się pod tym samym adresem dostęp do rejestru jest rozpoznawany kierunkiem transferu danych (rejestr odbiorczy-odczyt, nadawczy-zapis)

Rejestr kontrolno-sterujący portu USART: A- UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Port szeregowy synchroniczny-asynchroniczny USART

RXC - bit informujący o skompletowaniu danej odbieranej

TXC - bit informujący o wysłaniu całej danej

UDRE - bit informujący, że bufor nadawania jest gotowy do przyjęcia nowej danej

FE - bit zgłoszenia błędu ramki (ustawiany gdy w oczekiwanym czasie nie pojawił się bit stopu)

DOR - bit zgłoszenia błędu nadpisania (dana odbierana jest skompletowana, a wykryto bit startu nowej danej odbieranej)

PE - błąd parzystości

U2X - podwojenie prędkości transmisji w trybie asynchronicznym

MPCM - bit współpracy wieloprocesorowej

Port szeregowy synchroniczny-asynchroniczny USART

Rejestr kontrolno-sterujący portu USART: B- UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXCIE - bit maski przerwania od skompletowania danej odbieranej

TXCIE - bit maski przerwania od wysłania danej

UDRIE - bit maski przerwania od pustego rejestru danych

RXEN - włączanie odbiornika portu USART (zapis „1”)

TXEN - włączanie nadajnika portu USART (zapis „1”)

UCSZ2 - jeden z bitów określających rozmiar danej

RXB8 - 9-ty bit odbierany

TXB8 - 9-ty bit nadawany

Port szeregowy synchroniczny-asynchroniczny USART

Rejestr kontrolno-sterujący portu USART: C- UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

URSEL - bit dostępu do rejestru UCSRC i UBRRH, ustawienie na 1 zapewnia dostęp do rejestru UCSRC. Oba rejestry mają ten sam adres.

UMSEL - wybór trybu synchronicznego lub asynchronicznego, „0” tryb asynchroniczny.

UPM1, UPM0 - wybór rodzaju kontroli parzystości: wyłączona, parzystość parzysta, parzystość nieparzysta

USBS - ilość bitów stopu: „0”- 1bit stopu, „1”- 2 bity stopu

UCSZ1, UCSZ0 - wybór ilości bitów danej

Port szeregowy synchroniczny-asynchroniczny USART

UCPOL - tylko w modzie synchronicznym określa polaryzację sygnału zegarowego

Rejestr prędkości bodowej

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

URSEL - bit określający dostęp do starszej lub młodszej części rejestru. Przy zapisie do UBRRH powinien być ustawiony w stan niski

Port szeregowy synchroniczny-asynchroniczny USART

```
#include <avr/io.h>
#include <avr/interrupt.h>
//ustawiona częstotliwość - 18432000

//pomocnicze stałe
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void usart_init(void)                    //funkcja inicjalizująca usart
{
    UBRRH = (BAUD_PRESCALE >> 8);        //wpisanie starszego bajtu
    UBRRL = BAUD_PRESCALE;               //wpisanie mlodszego bajtu

    //UCSRA bez zmian - 0x00
    UCSRB = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN);    //aktywne przerwanie od
odbioru oraz zmiana trybu działania pinów D0 i D1

    UCSRC = (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);     //praca
asynchroniczna, brak kontroli parzystości, 1 bit stopu, 8 bitów danych

}
```

Port szeregowy synchroniczny-asynchroniczny USART

```
ISR(USART_RXC_vect)          //przerwanie od odbioru danej
{
    static char a;           //zmienna pomocnicza
    a = UDR;                 //zapis odebranej danej
    a ^= 0xFF;               //operacja bitowa XOR
    UDR = a;                 //wysłanie danej zwrotnej
}


int main(void)
{
    PORTD = 0x02;            //pullup na TXC
    usart_init();
    sei();                   //aktywacja przerwań
    while(1) {}
}
```

Port szeregowy synchroniczny-asynchroniczny USART

Program przesyłający dane do komputera

```
#include <avr/io.h>
#include <avr/interrupt.h>
//ustawiona częstotliwość - 18432000

//pomocnicze stałe
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void usart_init(void)        //funkcja inicjalizująca usart
{
    UBRRH = (BAUD_PRESCALE >> 8);        //wpisanie starszego bajtu
    UBRRL = BAUD_PRESCALE;               //wpisanie mlodszego bajtu

    //UCSRA bez zmian - 0x00
    UCSRB = (1<<TXEN);            //zmiana trybu działania tylko dla D1

    UCSRC = (1<< URSEL) | (1<< UCSZ0) | (1<< UCSZ1);          //praca
synchroniczna, brak kontroli parzystości, 1 bit stopu, 8 bitów danych

}
```

Port szeregowy synchroniczny-asynchroniczny USART

Program przesyłający dane do komputera

```
void timer0_init(void)
{
    //praca w przerwaniu od przepelnienia, preskaler 256, wysylanie
danych co 3,5ms
    TCCR0 |= (1<<CS02);
    TIMSK |= (1<<TOIE0);
}

 ISR(TIMER0_OVF_vect)
{
    UDR = PINB;                    //Port B jako źródło danych
}

int main(void) {
    PORTD = 0x02;                  //pullup na TXC
    PORTB = 0xFF;                  //port B -wejscia z pullupem
    usart_init();
    timer0_init();
    sei();                         //aktywacja przerwań
    while(1) {}
}
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
//ustawiona częstotliwość - 18432000

//pomocnicze stałe
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
#define ROZMIAR 16

//deklaracja i inicjalizacja bufora
typedef struct {
    unsigned char tablica[ROZMIAR];
    unsigned char poczatek;
    unsigned char koniec;
} bufor_cykliczny;

volatile bufor_cykliczny bufor = {.poczatek=0, .koniec=0};  //poczatkowe wartosci dla
wskaznikow

//dodatkowe funkcje bufora
void bufor_dopisz(unsigned char data)
{
    if(bufor.koniec+1 < ROZMIAR) //standardowy przypadek
    {
        if(bufor.koniec+1 != bufor.poczatek) {bufor.tablica[bufor.koniec++]=data;} //jeżeli
jest miejsce w buforze to zapisz, aktualizuj koniec, jeżeli nie ma miejsca to nic nie rób
    }
    else //przypadek gdy trzeba wrócić do 0
    {
        if(bufor.poczatek != 0) {bufor.tablica[15]=data; bufor.koniec=0;}
    }
}
```

```
void bufor_wyslij(void)
{
    if(bufor.poczatek != bufor.koniec)       //jeżeli bufor nie jest pusty
    {
      UDR = bufor.tablica[bufor.poczatek++]; //wyślij daną z początku i zwiększ
wskaźnik
    }
}

void usart_init(void)        //funkcja inicjalizująca usart
{
    UBRRH = (BAUD_PRESCALE >> 8);         //wpisanie starszego bajtu
    UBRRL = BAUD_PRESCALE;                //wpisanie mlodszego bajtu

    //UCSRA bez zmian - 0x00
    UCSRB = (1<<TXEN)|(1<<UDRE);          //zmiana trybu działania pinu D1, przerwanie
gdy rejestr nadawczy jest pusty (USART data register empty)
    UCSRC = (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);         //praca synchroniczna,
brak kontroli parzystości, 1 bit stopu, 8 bitów danych

}

void timer0_init(void)
{
    //praca w przerwaniu od przepelnienia, preskaler 256, wysylanie danych co 3,5ms
    TCCR0 |= (1<<CS02);
    TIMSK |= (1<<TOIE0);
}
```

```
ISR(TIMER0_OVF_vect)
{
    static unsigned char b;
    b = PINB;
    bufor_dopisz(b);
    if(bufor.poczatek+1==bufor.koniec) bufor_wyslij();    //jeżeli przed
dopisaniem bufor był pusty, wyślij wiadomość z bufora
}

ISR(USART_UDRE_vect)
{
    bufor_wyslij();
}


int main(void)
{

    PORTD = 0x02;          //pullup na TX
    PORTB = 0xFF;          //port B  -wejscia z pullupem
    usart_init();
    timer0_init();
    sei();                  //aktywacja przerwań
    while(1) {}            //nieskonczona petla
}
```

Programowe sposoby zarządzania energią

- zredukowanie częstotliwości zegara,
- wykorzystania trybów wstrzymania (sleep modes).

Wstrzymywanie systemu możliwe przy użyciu instrukcji *SLEEP*

Interakcje w mikrokontrolerze:

- nie pracuje jednostka centralna,
- pamięć danych zachowywana,
- rejestry wejścia wyjścia zachowywane,
- układy peryferyjne pracują w zależności od trybu uśpienia.

Programowe sposoby zarządzania energią

Mikrokontrolery AVR dysponują odpowiednimi systemami dzięki którym możemy programowo zarządzać poborem energii. Dzięki temu mamy możliwość wyłączenia części z podsystemów czy przejścia w dedykowane tryby oszczędzania energii.

Sterowanie poborem energii odbywa się za pomocą rejestru SMCR (Sleep Mode Control Register) - bity SM0 - SM2.

Bit	7	6	5	4	3	2	1	0
					SM[2:0]			SE
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

- Po wybraniu odpowiedniego trybu należy go uruchomić ustawiając bit SE.

Programowe sposoby zarządzania energią - Tryby uśpienia układu

SLEEP_MODE_IDLE	Najprostszy tryb. Wszystkie układy peryferyjne działają normalnie, wyłączany jest tylko rdzeń procesora (brak generowania sygnału taktowania CPU i FLASH).
SLEEP_MODE_PWR_DOWN	W tym trybie wyłączany jest zewnętrzny oscylator i większość podsystemów procesora. Działa tylko BOD, Watchdog, TWI, możliwe są także przerwania zewnętrzne (INT). W przypadku przerwań wyzwalanych zboczem sygnał musi trwać odpowiednio długo.
SLEEP_MODE_PWR_SAVE	W tym trybie działają tylko timery. Źródło zegara, z którego nie korzysta dany timer, też jest wyłączane.
SLEEP_MODE_ADC	Wszystkie podsystemy działają normalnie, wyłączany jest tylko rdzeń procesora, pamięć FLASH i układy portów IO, co zmniejsza zakłócenia, ułatwiając pomiar ADC. Po wejściu w ten tryb automatycznie ustawiana jest flaga ADSC rejestru ADCSRA (rozpoczynany jest pomiar ADC).
SLEEP_MODE_STANDBY	W przypadku wykorzystania zewnętrznego oscylatora tryb ten różni się od trybu SLEEP_MODE_POWER_DOWN tylko tym, że oscylator pozostaje włączony, co przyśpiesza wybudzenie procesora (tylko 6 cykli zegara).
SLEEP_MODE_EXT_STANDBY	Tryb podobny do trybu SLEEP_MODE_PWR_SAVE, z tym że zewnętrzny oscylator pozostaje włączony, co przyśpiesza wybudzanie procesora.

Wstrzymywanie pracy poszczególnych modułów -bity sterujące rejestru PRR

7	6	5	4	3	2	1	0
PRTWI	PRTIM2	PRTIMO	PRLCD	PRTIM1	PRSPI	PRUSART0	PRADC

PRTWI- wstrzymywanie portu I2C

PRTIM2- wstrzymywanie licznika T2

PRTIM0- wstrzymywanie licznika T0

PRLCD- wyłączanie interfejsu LCD

PRTIM1- wstrzymywanie licznika T1

PRSPI- wstrzymywanie portu SPI

PRUSART0- wstrzymywanie interfejsu USART0

PRADC- wstrzymywanie przetwornika AC

Układy peryferyjne mikrokontrolera AVR – zarządzanie energią

Przykład 1

```
#include <avr/wdt.h>
#include <avr/sleep.h>

Test_sleep void(){
wdt_reset();
set_sleep_mode(SLEEP_MODE_PWR_DOWN);      //wybierz tryb wyłączenia zasilania
set_sleep_mode ust. SLEEP_MODE_PWR_SAVE); //wybierz tryb oszczędzania energii
set_sleep_mode ust. SLEEP_MODE_STANDBY);    //wybierz zewnętrzny tryb zasilania w trybie
gotowości
set_sleep_mode ust. SLEEP_MODE_EXT_STANDBY); //wybierz zewnętrzny tryb zasilania w
trybie gotowości
set_sleep_mode ust. SLEEP_MODE_IDLE;        //nie działał tylko CPU i pamięc flash
set_sleep_mode ust. SLEEP_MODE_ADC);        // wybierz tryb redukcji szumów ADC
sleep_bod_disable();                        // opcjonalny wyłącznik wykrywania niskiego
zasilania CPU
sleep_mode( );                              // śpij teraz!
}
```

Układy peryferyjne mikrokontrolera AVR – zarządzanie energią

Przykład 1

```
void watchdogSetup(void){
cli();
wdt_reset();
WDTCSR | = (1<<WDCE) | (1<<WDE);
WDTCSR = (1<<WDIE) | (0<<WDE) | (1<<WDP3) | (1<<WDP0);
// 8s / przerwanie, brak resetowania systemu

sei();
}

ISR(WDT_vect){

//umieść tutaj dodatkowy kod
}
```

```
sleep_mode();  // Funkcja odpowiada sekwencji poleceń:

sleep_enable();
sleep_cpu();
sleep_disable();
```

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

Materiały zostały opracowane w ramach projektu
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,
umowa nr **POWR.03.05.00-00-Z060/18-00**
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020
współfinansowanego ze środków Europejskiego Funduszu Społecznego

Fundusze Europejskie Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska Europejski Fundusz Społeczny