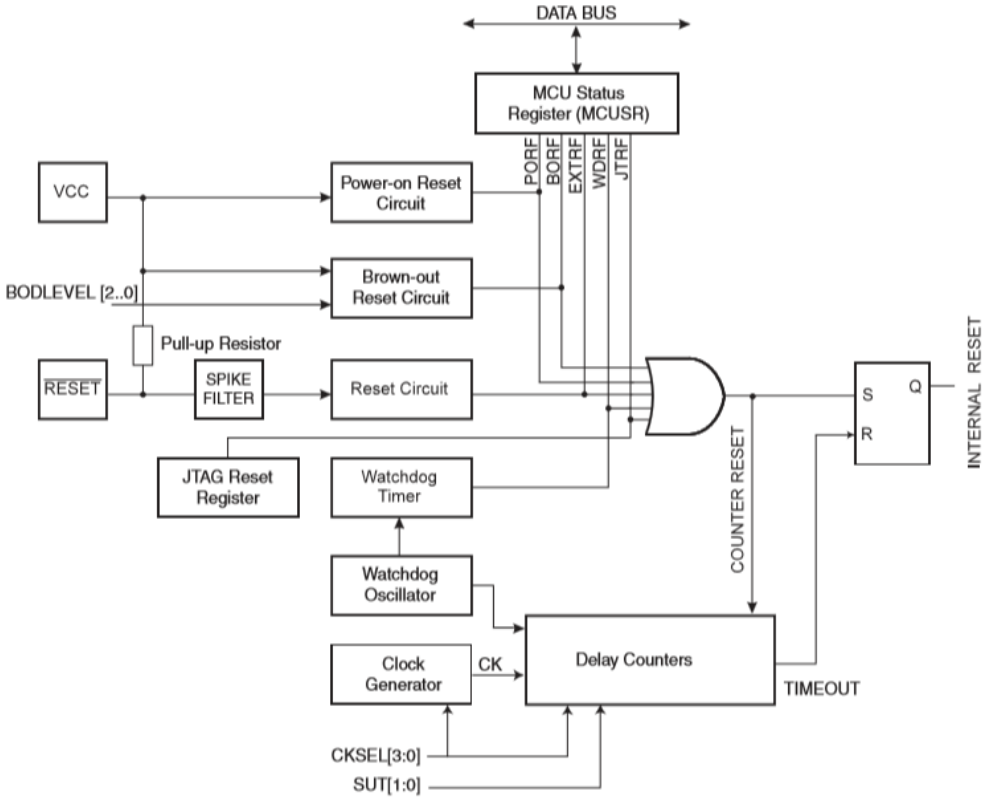
Układy zerujące

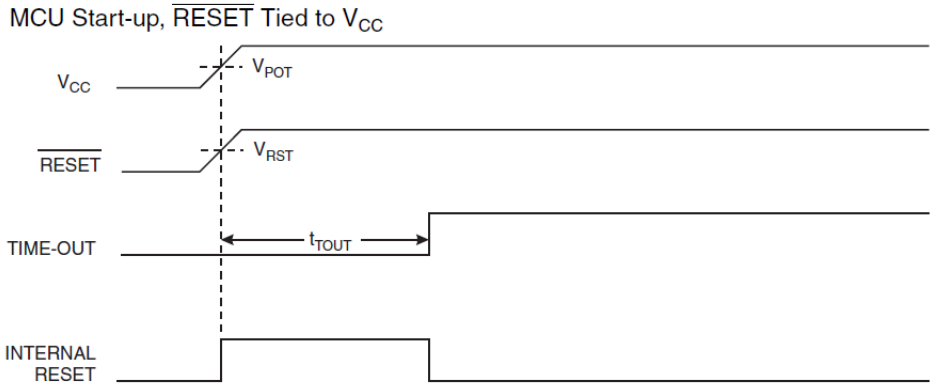
Mikrokontrolery AVR wyposażono w funkcję inicjalizacji wszelkich bloków wewnętrznych przed przystąpieniem do normalnej pracy. W trakcie tego procesu rejestry z przestrzeni wejścia-wyjścia ładowane są wartościami domyślnymi, wyprowadzenia portów ustawiane są w stan wysokiej impedancji, a licznik programu jest zerowany. Ponieważ pod zerowym adresem w pamięci programu powinien znajdować się rozkaz skoku do procedury inicjalizacyjnej, ostatnie z wymienionych działań jest jednoznaczne z obsługą przerwania, jakim jest zerowanie. Należy zwrócić uwagę na fakt, że zawartości rejestrów roboczych i pamięci SRAM nie są zerowane w trakcie inicjalizacji i mogą być przypadkowe.

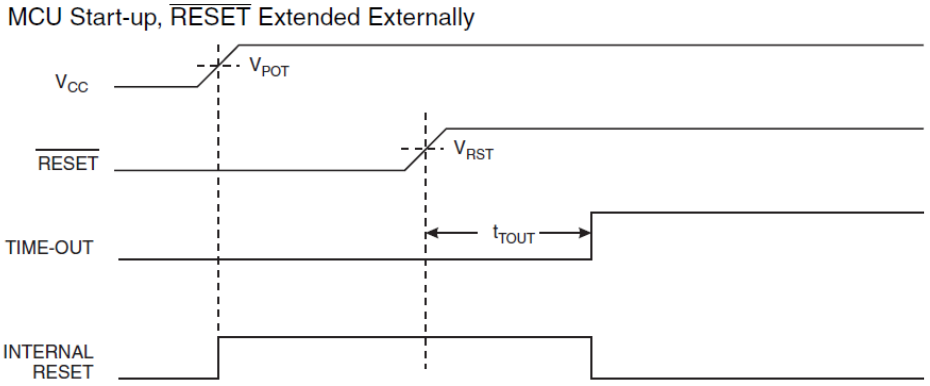


Logika układu zerowania

Bez względu na źródło i przyczynę wystąpienia impulsu zerującego, mikrokontroler rozpoczyna pracę po jego zaniku i odczekaniu czasu opóźnienia, oznaczonego zgodnie z dokumentacją jako t_{TOUT}. Opóźnienie wprowadzono w celu zapobieżenia ewentualnym błędom w działaniu układu spowodowanym stanami nieustalonymi zasilania i ustaleniem się częstotliwości zegara systemowego. Wartość t_{TOUT} jest programowana przez odpowiednie ustawienie bitów CKSEL i SUT.

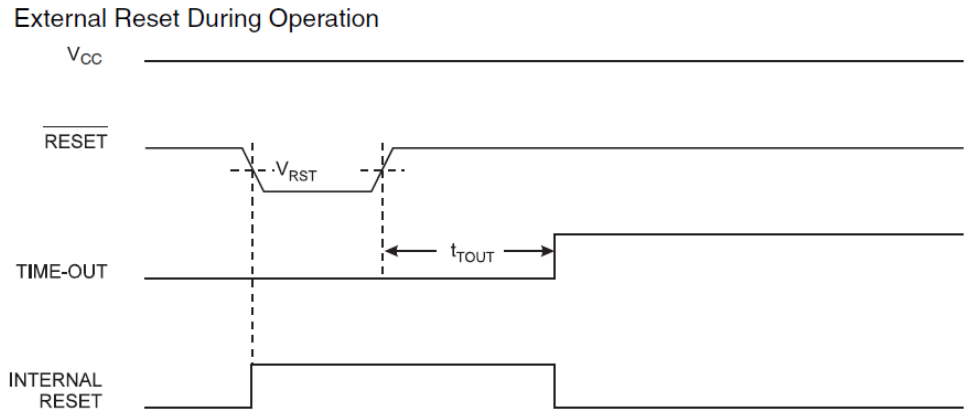
Najważniejszym z bloków układu zerującego jest POR (Power-on Reset). Moduł ten jest autonomiczny i nie mamy żadnego wpływu na jego działanie. Układ POR generuje impuls zerujący mikrokontroler dopóty, dopóki napięcie zasilania jest niższe od napięcia odniesienia ustalonego przez producenta (zwykle ok. 1,4V). Pozwala to na automatyczną inicjalizację mikrokontrolera po włączeniu zasilania.





Zewnętrzne źródło zerujące.

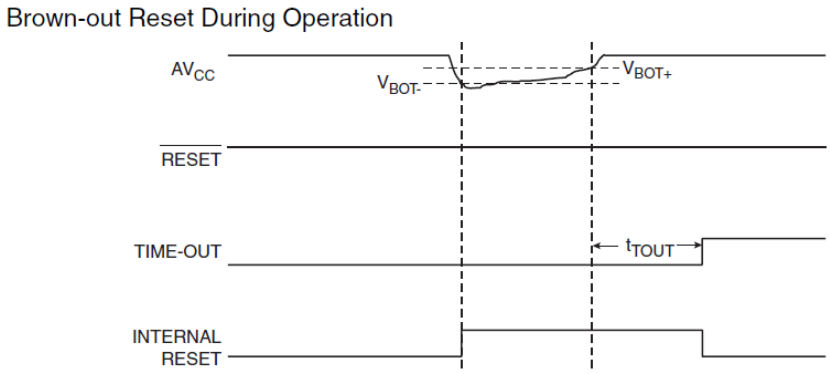
Zewnętrzny sygnał zerujący jest generowany prze podanie sygnału niskiego na wejście RESET przez czas co najmniej 2,5 ms. Sygnał ten spowoduje wyzerowanie mikrokontrolera nawet przy braku sygnału zegarowego. Zerowanie nastąpi po osiągnięciu przez sygnał wejściowy poziomu V_{RST} i odmierzeniu opóźnienia t_{TOUT}. Należy zauważyć, że końcówka RESET ma inne napięciowe progi przełączania niż porty wejścia-wyjścia. Ma ona również stale aktywny, wewnętrzny rezystor podciągający, którego wartość może wynosić 30…60kΩ.



Układ BOD

Poza układem POR, mikrokontrolery ATMega wyposażone są w dodatkowy moduł, który inicjalizuje układ w odpowiedzi na zbyt niskie napięcie zasilania. Działanie układu BOD

(Brown-out Detector) jest bardzo zbliżone do POR, jednakże ten pierwszy wyposażono w histerezę progu wyzwalania i możliwość wyboru jego poziomu

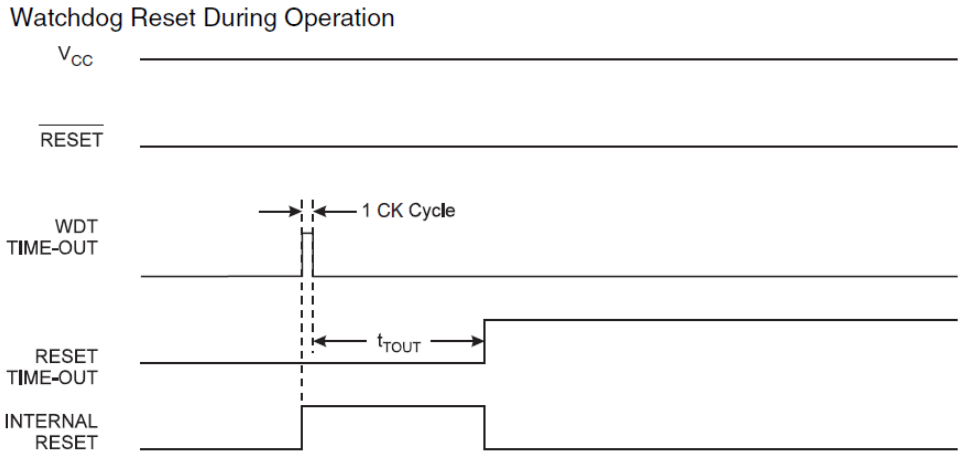


Moduł BOD uaktywnia się poprzez zaprogramowanie bitu konfiguracyjnego BODEN. Do określania progu wyzwalania tego układu służą bity BODLEVEL(2..0).

Włączenie układu BOD zwiększa pobór mocy przez mikrokontroler.

Układ WDT

Układ Watchdog Timer nadzoruje poprawną pracę mikrokontrolera. Ma postać licznika zliczającego impulsy autonomicznego generatora WDT. Przepełnienie tego licznika powoduje inicjalizację mikrokontrolera. Aby temu zapobiec w strategicznych punktach programu należy umieścić specjalne instrukcje zerujące (**wdr**). W przypadku zawieszenia pracy mikrokontrolera licznik watchdoga po ograniczonym czasie przepełni się, powodując zerowanie mikrokontrolera.



Układ WDT taktowany jest niezależnym generatorem o częstotliwości 128 kHz. Można zaprogramować czas opóźnienia jego działania w przedziale od 16 ms do 8 s.

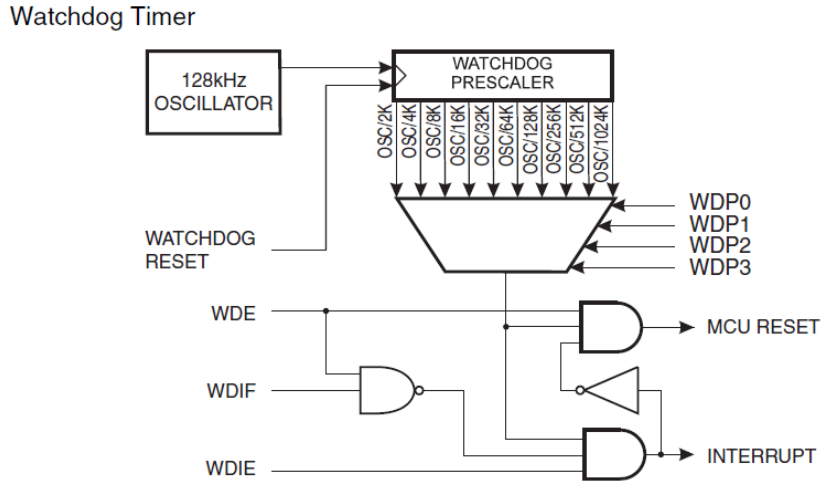
Może pracować w trzech trybach:

- przerwania,

- resetu systemowego,

- przerwania i resetu systemowego.

Układ WDT ma w przestrzeni adresowej wejścia-wyjścia dedykowany rejestr **WDTCSR**, który umożliwia włączanie/wyłączanie watchdoga, dokonanie nastawy czasu opóźnienia oraz ustawienie trybu pracy.



WDP3..0 nastawa preskalera generatora

WDE włączenie układu WDT w trybie zerowania mikrokontrolera

WDIE włączenie układu WDT w trybie przerwań

WDIF znacznik przerwania WDT (nastąpiło przepełnienie licznika)

Rejestr statusu i kontroli MCU – MCUCSR

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Każdy z modułów systemu zerującego mikrokontrolerów ATmega jest w stanie, jeśli spełniono odpowiednie kryteria, spowodować zerowanie układu. Rejestr statusowy MCUSR dostarcza informacji na temat, która z przyczyn resetu wystąpiła.

• Bit 4 – JTRF: JTAG Reset Flag

Ten bit jest ustawiony jeżeli przyczyną resetu było wpisanie jedynki logicznej do rejestru resetu JTAG wybranego przez rozkaz AVR_RESET. Bit ten jest resetowany przez reset przy włączeniu zasilania lub poprzez przypisanie mu logicznego zera.

• Bit 3 – WDRF: Watchdog Reset Flag

Ten bit jest ustawiany jeżeli zajdzie reset Watchdog'a. Jest on resetowany przez reset przy włączonym zasilaniu lub przez wpisanie zera do znacznika.

• Bit 2 – BORF: Brown-out Reset Flag

Bit ten jest ustawiony jeżeli zajdzie reset przy obniżonym napięciu zasilania. Jest on resetowany przez reset przy włączonym zasilaniu lub przez wpisanie zera do znacznika.

• Bit 1 – EXTRF: External Reset Flag

Bit ten jest ustawiony jeżeli zajdzie zewnętrzny reset. Jest on resetowany przez reset przy włączonym zasilaniu lub przez wpisanie zera do tego znacznika.

• Bit 0 – PORF: Power-on Reset Flag

Bit ten jest ustawiony jeżeli zajdzie reset po włączeniu zasilania. Jest on resetowany tylko przez wpisanie mu zera.

Sterowanie poborem mocy

Systemy wbudowane bardzo często pracują nieprzerwanie lub korzystają z zasilania bateryjnego. Stąd niezwykle ważną sprawą jest minimalizacja poboru mocy.

Nowoczesne mikrokontrolery posiadają możliwość pracy przy obniżonej częstotliwości zegara taktującego jak również korzystania z tzw. trybów wstrzymania (Sleep modes). Dotyczy to całego procesora lub jego układów peryferyjnych. W takim stanie mikrokontroler może oczekiwać na zajście jakiegoś zdarzenia i wtedy uruchamiać swoje możliwości i moc obliczeniową.

Wstrzymywanie systemu

Przełączenie mikrokontrolera w tryb wstrzymania następuje po wykonaniu rozkazu **sleep**. Uśpienie polega zawsze na wyłączeniu jednostki centralnej (CPU) i zaprzestaniu dalszego wykonywania programu. Tryb wstrzymania nie niszczy zawartości pamięci danych – zarówno rejestrów, jak i pamięci SRAM. Układ może zostać przywrócony do normalnej aktywności albo przez zdarzenie przerwania, albo na skutek zerowania mikrokontrolera.

Jeśli „pobudką" jest przerwanie, mikrokontroler po wyjściu z trybu wstrzymania wykonuje podprogram jego obsługi, a następnie przechodzi do wykonywania programu głównego od miejsca następującego po instrukcji sleep. Jeśli natomiast tryb wstrzymania przerywany jest żądaniem inicjalizacji (np. sygnałem Reset), przeprowadzona jest normalna procedura zerowania.

Istnieje do sześciu trybów pracy z ograniczonym poborem mocy, a ich wyborem steruje rejestr **SMCR**. Bity sterujące określają:

SE (Sleep Enable) – zezwolenie na tryby wstrzymania. Powinien być ustawiony przed rozkazem sleep.

SM2..0 (Sleep Mode Select) – wybór trybu wstrzymania

Możliwe jest również wstrzymywanie pracy poszczególnych modułów – sterowane jest przez ustawianie/zerowanie bitów w rejestrze **PRR**.

RESET i obsługa przerwań

AVR zapewnia wiele różnych źródeł przerwań. Przerwania i oddzielne wektory resetu mają oddzielny wektor programu w przestrzeni pamięci programu. Każde przerwanie ma przypisany bit przerwania, który musi być ustawiony w stan 1 łącznie z Globalnym bitem zezwolenia na przerwania w rejestrze stanu umożliwiają obsługiwanie zgłoszonych przerwań. Zależnie od wartości licznika programu (PC) przerwania mogą być automatycznie blokowane kiedy bity blokujące program ładujący BLB02 lub BLB12 są zaprogramowane. To rozwiązanie zapewnia bezpieczeństwo programowania.
Najniższe adresy w przestrzeni pamięci programu są domyślnie zdefiniowane jako wektory RESET'u i przerwań. Kompletna lista wektorów jest opisana w sekcji Przerwania (tabela 14.1 dokumentacji). Lista ta determinuje priorytet poszczególnych przerwań. Im niższy adres tym wyższy priorytet. RESET ma najwyższy priorytet, następnie INT0. Wektory przerwań mogą zostać przeniesione na początek ładowalnej sekcji pamięci Flash prze ustawienie bitu IVSEL w rejestrze kontroli MCU. Wektor RESET'u również może zostać przeniesiony do tej sekcji poprzez programowanie bezpiecznika BOOTRST.
W przypadku wystąpienia przerwania, I-bit rejestru globalnych przerwań jest zerowany i wszystkie inne przerwania zostają zablokowane. Oprogramowanie użytkownika może

wpisać do tego bitu logiczną jedynkę, aby odblokować zagnieżdżone przerwania. Wtedy wszystkie przerwania mogą przerywać procedurę obsługi przerwania. I-bit jest automatycznie ustawiany przy powrocie z przerwania - RETI.
Wyróżniamy dwa podstawowe rodzaje przerwań. Pierwszy typ jest uruchamiany poprzez zdarzenie, które ustawia flagę przerwań. Dla tych przerwań licznik programu (PC) jest ustawiany na aktualny wektor przerwania w celu wykonania procedury obsługi przerwania i sprzęt zeruje odpowiednią flagę przerwania. Flagi przerwań mogą być zerowane poprzez wpisywanie logicznej jedynki do bitu flagi. Jeśli zajdą wszystkie warunki przerwania podczas obsługi takiego samego przerwania zerującego bit przerwań, flaga zostanie ustawiona i zapamiętana do obsłużenia tego przerwania, lub jest ona wyzerowana przez oprogramowanie. Podobnie, jeżeli zostaną zgłoszone przerwania podczas, gdy bit globalnego przerwania jest wyzerowany, flagi tych przerwań zostaną zapamiętane do czasu ustawienia globalnego bitu przerwania i zostaną wykonane zgodnie z priorytetem.
Drugi typ przerwań będzie uruchamiany tak długo, jak warunki na zaistnienie przerwania będą obecne. Takie przerwanie nie muszą koniecznie mieć flag przerwań. Jeśli warunki powodujące przerwanie zakończą się zanim zostanie ono obsłużone, przerwanie to nie zostanie obsłużone.
Kiedy AVR wyjdzie z obsługi przerwania zawsze powróci do głównego programu i wykona następną instrukcję przed obsługą oczekującego przerwania.
Należy zauważyć, że rejestr stanu nie jest automatycznie odkładany na stos przy przejściu do obsługi przerwania, jak również nie jest z tego stosu pobierany po powrocie z obsługi przerwania. To musi zostać obsłużone programowo.
Korzystając z rozkazu CLI możemy natychmiastowo zamaskować przerwanie. Żadne przerwanie nie zostanie obsłużone po tym rozkazie, nawet jeżeli zostało zgłoszone równolegle z wykonywaniem tego rozkazu.

Czas odpowiedzi na przerwanie

Minimalnym czasem oczekiwania na wykonanie przerwania jest czas czterech cykli maszynowych. Po czterech cyklach maszynowych adres programu wskazywany wektorem dla aktualnego obsługi przerwania jest wykonywany. W tym czasie licznik programu jest odkładany na stos. Zwyczajnie wektor jest skokiem do podprogramu obsługi przerwania, i skok ten trwa zazwyczaj trzy cykle maszynowe. Jeżeli dojdzie do zgłoszenia przerwania podczas wykonywania wielocyklowej operacji, operacja jest kończona przed obsługą przerwania. Jeżeli zostanie zgłoszone przerwanie gdy MCU jest w trybie snu, odpowiedź na przerwanie jest wydłużona o cztery cykle maszynowe. To zwiększenie jest następstwem czasu potrzebnego na rozpoczęcie działania.
Powrót z procedury obsługi przerwania trwa cztery cykle maszynowe. W tym czasie ze stosu jest pobierany licznik programu, wskaźnik stosu jest zwiększany o dwa i jest ustawiany I-bit rejestru SREG.

Interrupt Vectors in ATMega 2560

http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B

Interrupt Vectors in ATMega 2560

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega2560 is:

Address	Labels	Code Comments
0x0000	jmp RESET	; Reset Handler
0x0002	jmp INT0	; IRQ0 Handler
0x0004	jmp INT1	; IRQ1 Handler
0x0006	jmp INT2	; IRQ2 Handler
0x0008	jmp INT3	; IRQ3 Handler
0x000A	jmp INT4	; IRQ4 Handler
0x000C	jmp INT5	; IRQ5 Handler
0x000E	jmp INT6	; IRQ6 Handler
0x0010	jmp INT7	; IRQ7 Handler
0x0012	jmp PCINT0	; PCINT0 Handler
0x0014	jmp PCINT1	; PCINT1 Handler
0x0016	jmp PCINT2	; PCINT2 Handler
0X0018	jmp WDT	; Watchdog Timeout Handler
0x001A	jmp TIM2_COMPA	; Timer2 CompareA Handler
0x001C	jmp TIM2_COMPB	; Timer2 CompareB Handler
0x001E	jmp TIM2_OVF	; Timer2 Overflow Handler
0x0020	jmp TIM1_CAPT	; Timer1 Capture Handler

External Interrupts

Example 6: C implementation using Arduino IDE generic functions

```
// Include the header for the avr interrupt system
#include "avr/interrupt.h"
volatile byte counter; // 8 bit counter - public variable
void setup(void)
{
// The 2 interrupt pins 21 and 20 declared as inputs with pull-up resistors activated
  pinMode(20 ,INPUT);
  pinMode(21 ,INPUT);
  digitalWrite(20, HIGH); // activate pull-up resistors
  digitalWrite(21, HIGH);

// Atach ISRs to the interrupts corresponding to pins 21 and 20 (INT0 and INT1)
  attachInterrupt(digitalPinToInterrupt(21), my_INT0, FALLING);
  attachInterrupt(digitalPinToInterrupt(20), my_INT1, FALLING);
}

void loop()
{
}

void my_INT0()// ISR for INT0
{
  counter = counter + 2;
  PORTA = counter; // Update the LEDs
}

void my_INT1() // ISR for INT1
{
  counter-- ; // Decrement the counter
  PORTA = counter; // Update the LEDs
}
```

Przerwania Arduino

Jeżeli w trakcie wykonywania programu wystąpi zdarzenie, na które kontroler musi zareagować natychmiast, kończy się wykonywać rozpoczęte polecenie, jego wynik zostaje zapisany w pamięci (na stosie), uruchamiany jest program obsługi przerwania, po czym kontroler wraca do wykonywania kolejnych kroków programu.

W zależności od typu arduino dostępna jest różna ilość przerwań. Standardowo są to dwa przerwania, o nazwach int.0 oraz int.1, które są wywoływane zmianą stanu na pinach, odpowiednio, 2 oraz 3.

Arduino Mega2560 oferuje 6 przerwań (nazwy od int.0 do int.5) na **pinach 2, 3, 21, 20, 19, 18**.

Arduino IDE pozwala na użycie czterech funkcji związanych z przerwaniami :

attachInterrupt()

detachInterrupt()

interrupts()

noIntrerrupts()

Funkcje **interrupts**() oraz **noInterrupts**() określają miejsca w programie, w których można używać przerwań.

Funkcja **interrupts()** włącza obsługę przerwań, funkcja **noInterrupts()** - wyłącza. Jeżeli zależy nam, żeby jakiś fragment programu został wykonany bez przerywania go, polecenia należy umieścić pomiędzy linijkami noInterrupts() oraz interrupts(), jak pokazano w przykładzie:

```
void setup() {}

void loop()
{
  noInterrupts();
  // polecenia, w trakcie wykonywania których przerwania nie będą obsługiwane
  interrupts();
  // pozostałe polecenia
}
```

Funkcja ***detachInterrupt()*** wyłącza wybrane przerwanie.

Składnia funkcji:

detachInterrupt(interrupt)

Parametry:

interrupt - numer przerwania do wyłączenia

Funkcja ***attachInterrupt()*** definiuje sposób obsługi przerwania. Po wystąpieniu zmiany stanu wejścia odpowiadającego za dane przerwanie (co określa parametr mode) uruchamiana jest funkcja, określona przez parametr function

Składnia funkcji:

attachInterrupt(interrupt, function, mode)

Parametry:

interrupt - numer przerwania

function - funkcja uruchamiana, gdy nastąpi przerwanie; funkcja ta nie może mieć parametrów i nie może zwracać żadnej wartości. Jest nazywana funkcją obsługi przerwania.

mode - określa, kiedy przerwanie ma być wyzwolone. Możliwe są cztery warianty:

LOW - przerwanie wywołane, gdy stan wejścia jest w stanie LOW (0V)

CHANGE - przerwanie wywołane przy zmianie stanu wejścia

RISING - przerwanie wywołane przy zmianie stanu z niskiego (0V) na wysoki (5V)

FALLING - przerwanie wywołane przy zmianie stanu wejścia z wysokiego (5V) na niski (0V).

Ważne jest, że w trakcie wykonywania programu obsługi przerwania nie działa funkcja delay(), a wartość funkcji millis() nie jest zmieniana (nie odliczany jest czas)

Przykład 1:

```
int pin = 13;
 volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);                    //pin 13 jest traktowany jako wyjście
  attachInterrupt(0, blink, CHANGE);       //deklaracja przerwania:
     // funkcja blink() zostaje wywołana przy zmianie stanu (CHANGE) wejścia 2
                                          //(bo do przerwania int.0 przypisany jest pin 2)
}
void loop()
{
  digitalWrite(pin, state);  //zapisanie wartości state (LOW) do wyjścia 13 (określonego jako
pin)
}
void blink()      //funkcja obsługi przerwania. Nie posiada parametrów i nie zwraca żadnej wartości
{
  state = !state;                          //zanegowanie zmiennej state
}
```

Przykład 2:

```
#define WY 13
volatile int stan = LOW;
int czas = 100;
void setup()
{
  pinMode(WY, OUTPUT);
  attachInterrupt(0, diodaOnOff, RISING);

}
void loop()
{
  if (stan)
  {
     digitalWrite(WY, HIGH);
     delay(czas);
     digitalWrite(WY, LOW);
     delay(czas);
  }
}
void diodaOnOff()
{
  static unsigned long lastMillis = 0;
  unsigned long newMillis = millis();
  if (newMillis - lastMillis < 50)
  {
  }
  else
  {
    stan = !stan;
    lastMillis = newMillis;
  }
}
```

Przykład 3:

```
#define LED 13
void setup()
{
   Serial.begin(9600);
   pinMode(LED, OUTPUT);
   attachInterrupt(0, displayMicros, RISING);
   attachInterrupt(1, displayMillis, RISING);
}
```

```
void displayMicros()
{
  Serial.write("micros() = ");
  Serial.println(micros());
}

void displayMillis()
{
  Serial.write("millis() = ");
  Serial.println(millis());
}

void loop()
{
  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);
}
```

Przykład 4

Przerwania od zegara wewnętrznego.

```
#include <TimerOne.h>

void Mruganie(void);

void setup()
{
 pinMode(13, OUTPUT);
 //Ustawienie przerwania co 1s
 Timer1.initialize(1000000);
 //Przyczepienie funkcji mruganie do timera1
 Timer1.attachInterrupt(Mruganie);
}

void loop()
{}

void Mruganie()
{
 digitalWrite(13, !digitalRead(13));
}
```