Wykład 5.

Przerwanie – sygnał powodujący zmianę sterowania w systemie operacyjnym. Jego pojawienie się wstrzymuje wykonanie bieżącego programu na rzecz wykonania procedury obsługi przerwania. Z punktu widzenia systemów wbudowanych przerwanie daje następujące korzyści:

1. Jest to pewnego rodzaju forma komunikacji między programem a procesorem
2. Jednocześnie stanowi alternatywne rozwiązanie w stosunku do aktywnego czekania

Aktywne czekanie – świadomie określone przez programistę czekanie na odpowiedź działań spoza kodu programu w systemie lub urządzeń zewnętrznych (operacja UART, odczyt z ADC, transmisja SPI/I2C, itp.). Podczas aktywnego czekania procesor wykonuje jałowe iteracje (instrukcje nop)

Procesor posiada wbudowany mechanizm obsługi przerwań. W pamięci systemu operacyjnego istnieją specjalne lokacje, w której znajdują się adresy procedur obsługi przerwań. Te adresy to tzw. **wektory przerwań.**

Rodzaje przerwań

1. **Sprzętowe**
    1. Zewnętrzne – sygnał pochodzi z zewnętrznego układu obsługującego przerwanie sprzętowe (np. z klawiatury, z modułu DMA) – służy do komunikacji z urządzeniami zewnętrznymi za pomocą impulsów elektrycznych (np. naciśnięcie przycisku w układzie z mikrokontrolerem powoduje wysłanie impulsu elektrycznego, który może być obsłużony przez układ obsługi przerwań sprzętowych mikrokontrolera). Takie urządzenie posiada specjalne połączenie zwaną **linią przerwania.**
    W ATMEGA 328P są 2 piny z obsługą tego rodzaju przerwań: INT0, INT1. Takie przerwanie możemy konfigurować pod kątem czasu zadziałania oraz na jakim rodzaju zboczu zegara (narastającym/opadającym). Urządzenia zgłaszają przerwania zboczem zegarowym i stanem niskim (niskim ze względu na powszechność tranzystorów polowych N i NP w układach, w których łatwiej zejść napięciem w dół)
    2. Wewnętrzne, nazywane wyjątkami:
        1. Faults (niepowodzenia) – aktualnie wykonywana instrukcja programu powoduje błąd, lecz procesor powraca do wykonania przerwanego kodu.
        2. Pułapka – w debuggerach, gdy świadomie każemy zatrzymać bieg programu w określonym miejscu wykorzystując wyjątek.
        3. Aborts – błędy nienaprawialne, procesor kończy wykonanie programu tym samym zabija proces.

W przypadku mikrokontrolera ATMEGA 328P przerwaniem wewnętrznym określa się przerwanie tzw. *Change interrupt* – PCINT0, 1, 2, … (każdy port je ma). Uwaga: kilka pinów na jednym porcie może uruchomić przerwanie, nie wiemy który. Przykłady przerwań wewnętrznych:
    - *TIMER: gdy TIMER osiągnie wartość TOP*
    - *ADC: gdy ADC skończy konwersję*
2. **Programowe** – wykorzystywane do komunikacji z systemem operacyjnym, który wywołuje odpowiednie przerwania w zależności od stanu rejestrów skojarzonych z programem wywołującym. W sytuacji systemów wbudowanych takie przerwanie jest inicjowane z poziomi kodu programu (przykład: printf/scanf z UART)

Przebieg przerwania z klawiatury

1. Naciśnięty klawisz wywołuje impuls zwany żądaniem przerwania
2. Żądanie przerwania jest odebrane przez procesor – wie, z której linii przerwania on przyszedł, zatem wykorzystując jej identyfikator kieruje się do tablicy wektorów przerwań, w której znajduje adres odpowiadającej identyfikatorowi linii procedury przerwania
3. W procedurze obslugi przerwania procesor odczyta jaki klawisz został naciśnięty odczytując stan rejestrów kontrolnych klawiatury. **Klawiatura wyłącza sygnał przerwania na czas odczytu rejestrów kontrolnych.** Dzięki temu gwarantuje mu poprawność znajdujących się tam danych.
4. Ostatnią instrukcją w procedurze przerwania jest **reti** – return from interrupt czyli skok na adres powrotu – miejsca, w którym nastąpiło przerwanie.

Problemy związane z implementacją przerwań

1. Procesor musi wiedzieć, gdzie ma wrócić z procedury obsługi przerwania – adres powrotu do programu.

Rozwiązanie: Współdzielenie wybranych rejestrów/rejestru między pamięcią programu, a programem przerwania.

2. Synchronizacja – czy możemy przerwać wykonywane przerwanie innym przerwaniem?

Rozwiązanie: Maskowanie/Odmaskowywanie przerwań

Maskowanie przerwań
Każde przerwanie ma swoje miejsce w danej masce przerwań. **Kontroler przerwań** sprawdza, które flagi przerwań są ustawione. Tym samym obsłużenie przerwania przebiega w następujący sposób:

- Flaga przerwania jest zerowana (sygnał dla inicjatorów przerwań z zewnątrz, że nie mogą wywołać tego samego przerwania, gdy jest w trakcie wykonywania)
- Po zakończeniu przerwania flaga z powrotem jest ustawiana.
- W ATMEGA 328P flagę przerwań czyści się wpisując 1

Priorytety przerwań
Po wyznaczeniu zbioru możliwych do wykonania przerwań w danym momencie kontroler przerwań musi zadecydować w jakiej kolejności wykonać przychodzące przerwania (często jest ich więcej niż jedno naraz):

- Wykonuje je wg odwrotnych priorytetów – te o najniższym mają priorytet
- Kolejne przerwanie jest wykonywane tylko po zakończeniu bieżącego
- W ATMEGA 328P przycisk RESET powoduje przerwanie, które jest wykonywane zawsze (nie można go wymaskować)

Globalna obsługa przerwań

- sei(); - włącza globalną obsługę przerwań. Bez tego nie możemy używać przerwań w kodzie programu
- cli(); - wyłącza globalną obsługę przerwań. Możliwość zdecydowania przez programistę o tym, kiedy jest potrzeba wyłączenia obsługi przerwań w kontekście pisanego programu.
- Na czas wykonania procedury obsługi przerwań globalny bit przerwań jest wyłączony (po to, abyśmy nie mogli się wywłaszczyć z przerwania na kolejne przerwanie). Na końcu obsługi procedury ma Interrupt return, który znowu włącza globalny bit przerwań
- Jawne umieszczanie sei() w procedurze obsługi przerwania może mieć nieobliczalne efekty – lepiej tego nie robić

Tryby uśpienia procesora – ATMEGA 328P

- Im głębszy poziom uśpienia, tym większa oszczędność energii, lecz tym trudniej procesor wybudzić
- **ADC Noise Reduction** – to tryb działania ADC, w którym usypiany jest procesor na czas dokonywania pomiaru. Dzięki temu układ jest mniej zaszumiony i pomiar jest dokładniejszy.
- **Sleep Enable bit** – odblokowuje jakiekolwiek możliwości usypiania procesora z poziomu kodu programu

Volatile w języku C

- *volatile* to słowo kluczowe zmiennej umieszczane przez programistę będące informacją dla kompilatora, żeby przy **każdym** odwołaniu do tej zmiennej w trakcie programu sięgał po jej wartość z pamięci, a nie z uprzednio załadowanej kopii do wydzielonego rejestru (co kompilatory lubią robić w trybach optymalizacji, gdyż znacznie przyspiesza im czas działania)
- Ma to szczególne znaczenie przy zmiennych zmapowanych na fizyczne rejestry sprzętowe, w tym zewnętrznych urządzeń. Rejestr takiego urządzenia będzie zmieniał swoją zawartość, o czym program i kompilator nie mają pojęcia. Wówczas *volatile* załatwia sprawę, gdyż zmuszamy kompilator przy każdym odwołaniu takiej zmapowanej zmiennej do sięgnięcia do pamięci programu, w której jest zmapowany rejestr urządzenia za pomocą MMIO.

  MMIO – sposób ułatwienia obsługi dostępu do i wykonywania operacji na urządzeniach wejścia/wyjścia w systemach komputerowych. W metodzie tej rejestry urządzenia zostają odwzorowane w przestrzeni adresowej pod zadanym adresem.

Rejestr urządzenia → (MMIO) RAM →(volatile) Kompilator

Przykładowe procedury obsługi przerwań w ATMEGA 328P w C

```
# przerwanie sprzętowe
ISR(INT0_vect) {
  led = 1; # zmienne używane w przerwaniach deklarujemy globalnie
}

# przerwanie wewnętrzne timera – overflow TOP (przepełnienie)
ISR(TIMER2_OVF_vect) {
}
```

Wykład 6.

Magistrala – układ zbudowany ze zbioru linii przenoszących sygnały oraz układów wejścia-wyjścia do inicjowania/odbierania sygnałów (nadajniki/odbiorniki). Służy do komunikacji pomiędzy urządzeniami połączonymi w systemach mikroprocesorowych.

Rodzaje magistrali

1. Równoległa – sygnały są przesyłane równolegle jednocześnie wieloma kanałami. Przykład: zapisana na n-bitach liczba może zostać przesłana szeregowo za pomoca n-kanałów naraz. Przykłady: PCI, AGP, FSB. **Problem: synchronizacja między bitami**
2. Szeregowa – sygnał jest przesyłany szeregowo, tzn. bit po bicie. Ta sama liczba zapisana na n-bitach zostanie przesłana przez 1 kanał w n kawałkach. Przykłady: SPI, USB, PCI Express, I2C. **Problem: kiedy czytać poszczególne wartości bitów**

Czemu używa się magistrali szeregowych znacznie częściej niż równoległych? **Bo są dużo szybsze niż równoległe.**

Rodzaje magistrali ze względu na kierunek transmisji

1. Jednokierunkowa – **simplex** – dane przesyłane są w jednym kierunku
2. Dwukierunkowa – **duplex** – dane są przesyłane dwukierunkowo:
    1. Przepływają w obu kierunkach naraz – **full duplex**
    2. W danym momencie może przepływać tylko jeden sygnał – **half duplex**

Rejestr przesuwny - służy do zamiany sygnałów równoległych w szeregowe i odwrotnie. To rejestr, który posiada dodatkowo:

- wejście zegarowe – z niego wie, kiedy ma wykonać daną operację
- wejście i wyjście danych
- write enable – z niego wie, kiedy jest zapis, a kiedy odczyt

W momencie pojawienia się bitu na wejściu danych aktualny stan rejestru przesuwa się o jeden w prawo – przedostatni bit staje się ostatnim bitem, a bit wejścia jest teraz na 1 miejscu. Przykład:

Stan rejestru w danym momencie *t*:

- **IN → 0 1 1 1 0 0 1 1 → OUT**

Na wejściu pojawia się **1**:

- **IN → 1 0 1 1 1 0 0 1 → OUT (na wyjściu pojawia się 1)**

Gdybyśmy zbuforowali przychodzące dane równolegle w postaci n-bitowych pakietów w buforze o ilości n – bitów, to mamy konwerter sygnału równoległego w sygnał szeregowy. Odwracając proces, czyli buforując dane wyjściowe, otrzymalibyśmy konwerter szeregowy → równoległy.

Magistrala SPI – szeregowy interfejs do komunikacji pomiędzy systemami mikroprocesorowymi, a urządzeniami peryferyjnymi (np. ADC/ DAC, pamięci EEPROM, FLASH, karty MMC, SD). Dane przesyłane są typowo seriami po: bit startu + 1 bajt danych + bit stopu. Nadawca i odbiorca (a właściwie ich rejestry przesuwne) są rozróżnialni:

- Nadawca TX - *Master*:
  - pin 1 IN: MasterInputSlaveOutput (MISO)
  - pin 2 OUT: MasterOutputSlaveInput (MOSI)
  - zegar: SCK
- Odbiorca RX - *Slave*:
  - pin 1 IN: MOSI
  - pin 2 OUT: MISO
  - zegar: SCK

Operacje nadawania/odbierania są wykonywane na różnych zboczach zegara.

Odbiorców może być więcej niż 1 połączonych ze sobą równolegle. Wówczas jest potrzebny sygnał **Slave Select** do każdego odbiornika od mastera. Za pomocą Slave Select wybieramy odbiornik, do którego nadawana jest w danym momencie transmisja.
Czy master może posiadać pin Slave Select? Tak, w ten sposób możemy ustalać, czy master będzie nadawał czy nie (musi być ten pin ustawiony w trybie wyjścia, czyli stanem wysokim - 1)

Jeśli odbiorniki są połączone szeregowo, to wówczas robi się potok przesyłania danych – pierwszy w kolejności odbioru z linii transmisyjnej podłączony odbiornik przekazuje dane swoim wyjściem na wejścia

kolejnego podłączonego odbiornika. Za to pierwszy może odebrać już nowe dane, które potem znowu przekaże, itd.

Korzystając z magistrali SPI (konfigurując ją do własnych potrzeb) w praktyce ustawiamy wartości z następujących rejestrów:

Parametry rejestru kontrolnego SPI (Control Register)

- *włączenie/wyłączenie przerwań*
- *bit aktywacji transmisji (1 – działa)*
- *data order – czy transmitujemy od najmniej znaczącego czy najbardziej znaczącego bitu bajtu*
- *master slave select*
- *sterowanie częstotliwością zegara w trybie master (4 ustawienia preskalera)*

Parametry rejestru statusu SPI (Status Register)

- *flaga przerwań*
- *Write Collision flag – poinformuje o błędnych transmisjach, gdy np. nadajemy zbyt szybko*

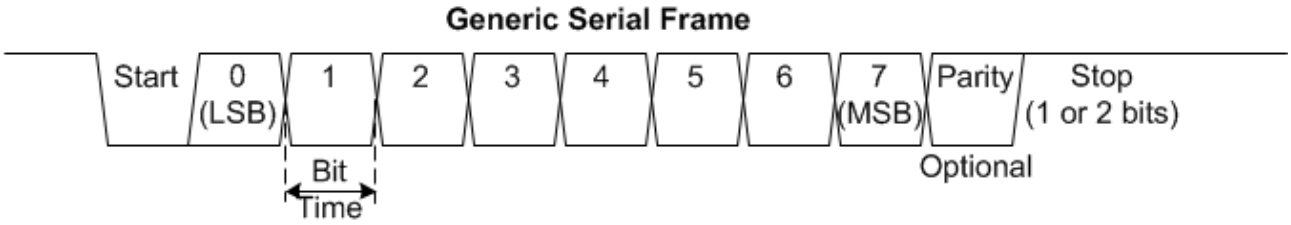
Parametry rejestru danych SPI (Data Register)

- *Wpisujemy dane, które chcemy przesłać*
- *Z tego samego rejestru odczytujemy dane, które nam zostały przesłane*

***Output enable** – służy do wyłączania/włączania wyjścia. Przykład: multipleksowanie diód - na czas załadowania bitów w transmisji chcemy wyłączyć wyświetlacz podczas przełączania.*

UART – uniwersalny asynchroniczny nadajnik-odbiornik – to układ scalony służący do asynchronicznego przekazywania i odbierania danych przez port szeregowy (port – inaczej interfejs, przez który przesyła się dane). Innymi słowy, implementuje ***transmisję szeregową asynchroniczną***.

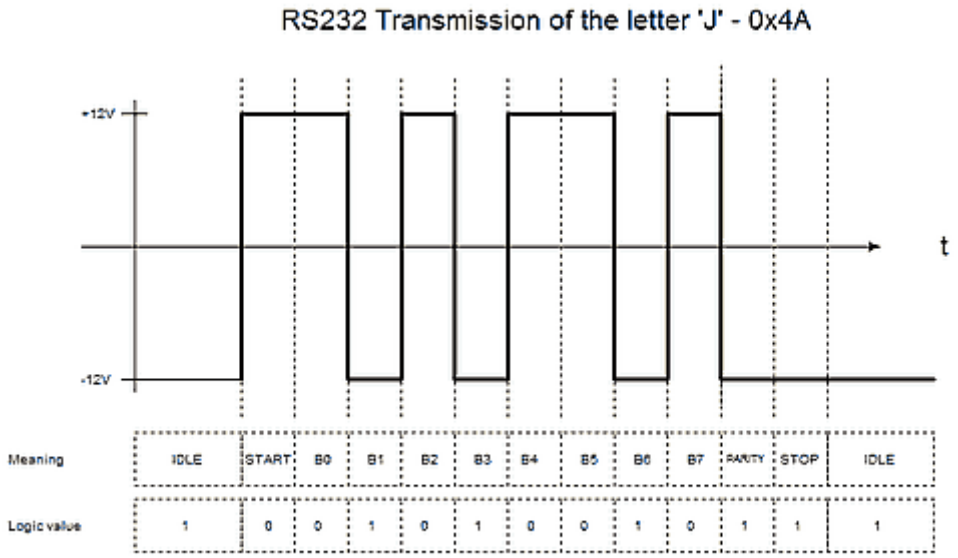
***USART –** uniwersalny synchroniczny i asynchroniczny nadajnik-odbiornik.*



Rys. 1 Transmisja danych przez UART

Transmisja szeregowa asynchroniczna (RS-232)

- Dane poprzedzone bitem startu o stanie logicznym 0
- Po nim przesyłane są bity danych, domyślnie od najmłodszego do najstarszego
- Następnie bit parzystości (opcjonalnie)
- Na koniec odstęp nazywany bitem stopu przed następnym znakiem stanem wysokim – długość tego odstępu może być różna. Stan logiczny 0 podczas bitu stopu jest identyfikowany z błędem ramki (*framing error)*
- Układ odbierający po wykryciu stanu logicznego 0 odmierza czas równy ½ czasu przesyłania bitu i sprawdza, czy stan jest nadal 0 – jeśli jest, to wtedy uznaje, że właśnie nastąpił bit startu i może odbierać transmisję – odmierza odcinki czasu równe czasowi 1 bitu i odczytuje stan linii otrzymując wartości bitów
- Podawanie stanu 0 przez dłuższy czas jest sygnałem przerwy w transmisji



Baud rate

- *baud rate* – określa częstotliwość transmisji (ile bitów informacji zostanie przesłanych w ciągu 1 sekundy). W praktyce oznacza też, jak częste będą zmiany napięcia w jednostce czasu i jak często odbiornik będzie odczytywać wartości napięcia.

Przebieg transmisji w UART

Nadawanie:

1. Czekamy na zapalenie bitu UDRE0 (USART Data Register Empty),
2. Po zapaleniu ładujemy nasz bajt (8 bitów) do UDR0

Odbiór:

1. Czekamy na RXC0 receive complete bit w UCSR0A