Wyższa Szkoła Handlowa w Radomiu

Systemy Wbudowane

Laboratorium 3

1 Cel zajęć

Niekiedy podczas pracy układu zachodzi potrzeba (a nawet konieczność) natychmiastowej reakcji układu na zaistniałe zdarzenie. Do tego typ zastosowań służą przerwania sprzętowe (rzadziej programowe). Celem niniejszego materiału jest zapoznanie się z ich obsługą.

2 Teoria

Niemal wszystkie układy elektroniczne posiadają obsługę przerwań. Bez przerwań układy działały by nieprzerwanie swoim cyklem, bez względu na otaczające je środowisko, w którym pracują.

Jako przykład przydatności przerwań niech posłuży zegarek komputera. Gdyby komputer nie odświeżał zegara bez względu na to, co aktualnie wykonuje, to już po kilku instrukcjach mógłby on późnić się o co najmniej 2-3 sekundy. Innym przykładem jest system alarmowy, który wykonując procedury sprawdzająco-konserwacyjne mógłby nie zareagować w porę np. o uszkodzonej rurze z gazem czy niepokojącym dymie w pomieszczeniu z materiałami łatwopalnymi (bo byłby w trakcie sprawdzania temperatury w pomieszczeniu obok).

Przerwania i ich obsługa w głównej mierze zależą od projektu programowanego kontrolera i/lub układów z nim współpracujących. Generalnie rozróżniamy następujące przerwania:

zewnętrzne – emitowane są one najczęściej przez inne urządzenia (np. mysz, klawiatura, układ graficzny) bądź do innych urządzeń (układ graficzny, pamięć masowa, ekran). W przypadku naszych układów przerwania mogą być generowane przez termometry, przyciski, potencjometry, fotodiody, podczerwień itp.

wewnętrzne – tego typu przerwania generuje sam mikrokontroler (nie mamy na nie zbyt dużego wpływu). Tego typu operacje (wyjątki) pozwalają na zgłaszanie błędów np. dzielenia przez zero, wystąpienie niewłaściwych/błędnych instrukcji w kodzie, obsługę odpluskwiacza (ustawienie tzw. punktu przerwania, który pozwala na analizę wykonywanego kodu przez programistę) oraz tzw. uszkodzonych wyjątków, których nie można obsłużyć (np. przez naruszenie ochrony pamięci i/lub odwołanie do uszkodzonego miejsca w pamięci ulotnej)

programowe – w tym przypadku wykorzystuje się nieprzypisane pozycje wektorów przerwań z tablicy przerwań. Programiści mogą, w reakcji na instrukcje warunkowe, wykorzystać skok do wskazanego wektora przerwań, który z kolei wykona operacje wskazanej procedury. Tego typu przerwania są szczególnie przydatne w przypadku obsługi wyjątków w naszym programie, odwołaniu się do wyjścia (np. monitora) itp.

Przerwania można także podzielić na tzw. maskowalne i niemaskowalne. Jako programiści częściej będziemy mieli do czynienia z przerwaniami maskowalnymi, tj. takimi, które można pominąć (zamaskować ich wystąpienie – stąd nazwa). Do tego typu przerwań należy spora część przerwań wewnętrznych oraz wszystkie programowalne. Aby zamaskować przerwania (wszystkie bądź wybrane) należy ustawić odpowiednie flagi w rejestrze stanu. Przerwania niemaskowalne to z kolei takie, których wystąpienia nie można pominąć. Przerwań tych nie dotyczą flagi maskowania. Do tego typu przerwań należą wszystkie wewnętrzne. Przerwania tego typu zawsze mają najwyższy priorytet i mogą przerywać inne wykonywane przerwania o niższym priorytecie (decyduje tablica wektorów przerwań). W przypadku użytkowanych Arduino Leonardo niemaskowalnym przerwaniem o najwyższym priorytecie jest RESET.

Na koniec warto zaznaczyć, że wszystkie przerwania muszą być obsługiwane wedle określonego porządku. Należy bowiem pamiętać, że układ może naraz (bądź w podobnym czasie) otrzymać do wykonania kilka przerwań (np. wygenerowane przez wciśnięcie przycisku, nadesłane z czujnika pomiarowego oraz zgłoszone przez kontroler). Dlatego przy wykonywaniu przerwań stosowany jest następujący porządek:

priorytet przerwania – pierwszym, niepodważalnym kryterium jest priorytet w tabli przerwań. Im numer w tabeli jest niższy, tym dane przerwanie jest ważniejsze. Jeżeli wystąpi przerwanie z wyższym priorytetem to

wszystkie przerwania z niższymi priorytetami zostaną natychmiast przerwane, a procesor wykona obsługę nadchodzącego przerwania.

sekwencja działań – jeżeli układ nie posiada priorytetowych wywołań przerwań bądź kontroler otrzyma do wykonania dwa przerwania o identycznych priorytetach (np. to samo przerwanie aktywowane z dwóch innych wejść) to następuje sekwencyjne obsłużenie przerwań (tj. wedle ich pojawienia się). Przerwanie, które pojawiło się jako pierwsze musi w pełni zakończyć się by obsłużyć przerwanie kolejne.

zagnieżdżenie – rozwiązanie to zakłada obsługę nowego przerwania kosztem aktualnie wykonywanego. Obecnie wykonywane przerwanie (jeżeli takowe jest obsługiwane) zostanie zawieszone (a jego stan zapisany), a po zakończeniu obsługi najnowszego przerwania układ wraca do obsługi poprzednio zawieszonego.

Każde z powyższych rozwiązań ma zarówno wady jak i zalety. Niektóre rozwiązania opierają się na sekwencji porządkowej, tj. najpierw priorytet, później zagnieżdżenie lub priorytet i sekwencja.

W przypadku Adruino Leonardo mamy do czynienia z priorytetem przerwań (tablica wektorowa). Ponadto przerwania zewnętrzne generowane są jedynie w chwili ich wystąpienia. Jeżeli układ nie zdąży obsłużyć przerwania w danym momencie wystąpienia, to po jego minięciu nie nastąpi obsługa przerwania z opóźnieniem.

2.1 Liczniki i czasomierze

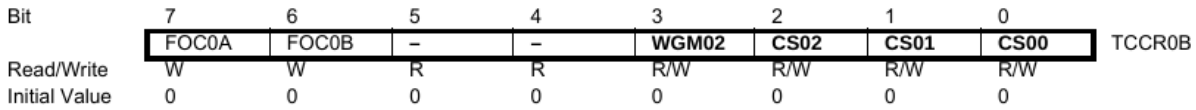
Jednym z często stosowanych typów przerwań jest cykliczna obsługa fragmentu kodu, np. stopera czy zegara. W standardowo pisanej aplikacji nie mielibyśmy możliwości precyzyjnego wykonywania kodu co określony czas (w tym wypadku – liczenia sekund lub odstępów czasu pomiędzy kolejnym wykonaniem kodu).

Do takich zadań wykorzystuje się właśnie liczniki i czasomierze wbudowane w układ. Każdy z nich posiada pewną precyzję (8 , 10 bądź 16 bitową), która stanowi podstawę jego liczenia (co każdy takt zwiększa się bądź zmniejsza zapisana liczba). Po ustawieniu wszystkich bitów na 1/zmniejszenia do 0 następuje tzw. przepełnienie czyli wywołanie przerwania wewnętrznego (i ponowne przeliczanie od zera/górnego zakresu liczbowego). Jako programiści mamy możliwość dowolnej zmiany precyzji (ustawienia głównego pułapu liczbowego w liczniku – wtedy następuje odliczanie do zera) oraz przeskalowania szybkości działania modułu. Domyślnie działa on z szybkością naszego kontrolera (czyli 16 MHz – 16 milionów operacji na sekundę). Oznacza to, że licznik posiadający precyzję 8 bit (mogący osiągnąć maksymalnie wartość 255) przepełni się po ok 16 mikrosekund (i co tyle będzie wyzwalane przerwanie). W najlepszym przypadku, przy użyciu 16 bitowego licznika przerwania występować będą co 4 milisekundy.

Aby uzyskać bardziej satysfakcjonujący wynik można posłużyć się wbudowanym preskalerem. Domyślnie układ pozwala na podzielenie aktualnej częstotliwości zegara kontrolera przez wartość 8, 64, 128 oraz 256. Dodatkowo układ posiada wyprowadzenia do zewnętrznych układów taktowania (możliwe podpięcie zegara taktującego opadającym bądź wzrastającym zboczem).

Poniższa tabela wskazuje ustawienia 3 młodszych bitów w rejestrze danego licznika, które decydują o jego trybie działania.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.



Rysunek 1: Wizualizacja rejestru kontrolnego licznika (np. dla Timer0 to TCCR0B)

W związku z tym częstotliwość wywołania przerwania poprzez zegar można wyliczyć z następującego wzoru:

$$czestotliwosc\_przerwan(Hz) = (16,000,000Hz)/(preskaler * (wartosc\_rejestru\_porownania + 1))$$

gdzie $wartosc\_rejestru\_porownania$ to ustawiona wartość licznika, od której ma liczyć do zera. Dodanie wartości 1 jest konieczne ponieważ osiągnięcie wartości 0 jest dla licznika także wartością (przerwanie wykonuje się dopiero w kolejnym cyklu po ponownym osiągnięciu maksymalnej wartości w liczniku).

W związku z tym jeżeli chcemy wykonać przerwanie np. co sekundę możemy obliczyć wartość odliczania z następującego wyprowadzenia:

$a = 16000000/(p * (c + 1))$

$p * (c + 1) = 16000000/a$

$c + 1 = 16000000/(a * p)$

$c = (16000000/(a * p))\check{}1$

Dla wartości preskalera 8

$c = (16000000/(8 * 1))\check{}1 = (16000000/8)\check{}1 = 2000000\check{}1 = 1999999$

Ponieważ żaden z liczników nie jest w stanie przyjąć takiej wartości, przerwanie musiałoby zliczać ilość jego wystąpienia. Tutaj należy wziąć pod uwagę czas obsługi przerwania (minimum 5 taktów + dokończenie wykonywania nieprzerywalnej operacji) oraz powrotu (kolejne 5 taktów). W związku z tym lepiej jest poszukać lepszej wartości preskalera (np. 1024)

$c = (16000000/(1024 * 1))\check{}1 = (16000000/1024)\check{}1 = 15624$

Wynik ten wskazuje, że najlepiej będzie wykorzystać licznik1 (Timer1), którego maksymalna wartość liczenia wynosi 2^{16} (65535).

2.1.1 Przykład kodu zmieniającego stan diody świecącej na złączu 13

```
bool toggle1 = true;

void setup() {
    //wylaczenie przerwan na czas ustawienia zegara Timer1
    cli();
    //ustawienie normalnego trybu pracy zegara -
    //zerowanie bitow wskazanego ponizej rejestru
    TCCR1A = 0;
    //zerowanie ustawien drugie rejestru trybu pracy
    //dzieki zerowaniu zegar bedzie pracowal jedynie
    //w trybie wewnetrznym; w dalszej czesci kodu
    //bedziemy modyfikowac niektore bity tego rejestru
    TCCR1B = 0;
    //ustawienie wstepnej wartosci zegara na 0
    TCNT1  = 0;
    //ustwienie wartosci do porownywania przez zegar do aktualnej wartosci
    //w TCNT1; jezeli wartosci beda rowne nastepuje przerwanie, a wartosc
    //TCNT1 jest zerowana
    OCR1A = 15624;
    //nie aktywujemy pinu komparatora (standardowe dzialanie zegara;
    //aby aktywowac tryb porownawczy dla PIN OC0A nalezy zamienic
    //0 na 1)
    TCCR1B |= (0 << WGM12);
    //aktywujemy tryb preskalera czestotliwosci zegara na 1024 (wedle
    //zalaczonej do instrukcji tabeli)
    TCCR1B |= (1 << CS12) | (1 << CS10);
    //uruchamiamy przerwanie dla zegara 1
    TIMSK1 |= (1 << OCIE1A);
    //uruchomienie przerwan
    sei();
}

//w przypadku podlaczenia LED do portu 13 bedzie ona sie rozswietlac i gasnac co
    ok. 1 sekunde
//przerwanie wywolywane jest niezaleznie od kodu znajdujacego sie w petli loop
ISR(TIMER1_COMPA_vect){
    if (toggle1)
    digitalWrite(13,HIGH);
    else
    digitalWrite(13,LOW);
    toggle1 != toggle1;
}

void loop() {
    //tutaj mozna dolaczyc wlasny kod
    //wykonujacy sie niezaleznie od przerwania by zobaczyc
    //ze przerwanie wykonuje sie niezalenie od ilosci
    //instrukcji w glownej petli programu
}
```

2.2 Przerwania wywoływane przez użytkownika

Kolejnym sposobem wywołania przerwań w Arduino Leonardo jest reakcja na zmianę stanów na określonych wyprowadzeniach układu. Do dyspozycji programisty oddane są następujące wyprowadzenia układu: 0, 1, 2, 3, 7. Wykorzystanie przerwań jest niezwykle proste i sprowadza się do ustawienia poniższej funkcji:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

Na stronie dokumentacji znajdziemy jeszcze inne funkcje odpowiadające za dodanie przerwań, jednak ta jest najpewniejsza (i polecana do użytku ze wszystkimi układami Arduino). Poszczególne parametry funkcji:

`digitalPinToInterrupt`**(pin)** - określa numer przerwania, które ma być wywoływane. Użyta tutaj funkcja pozwala na pobranie numer przerwania z tablicy przerwań dla wskazanego w niej wyprowadzenia (pin). Można, w wyjątkowych sytuacjach, użyć bezpośredniego numeru przerwania (o ile dokładnie wiemy jaki jest jego numer w tablicy przerwań)

ISR – pod ten parametr wstawiamy nazwę funkcji (która nie może przyjmować żadnych parametrów ani nic zwracać), której kod wykona się w przypadku nadejścia przerwania

mode – określa w jakiś sposób przerwanie ma zostać obsłużone. Dostępnymi wartościami są:

1. **LOW** – wyzwalanie gdy na wyprowadzeniu pojawi się logiczne 0
2. **CHANGE** – wyzwalanie w przypadku zmiany wartości na wyprowadzeniu
3. **RISING** – wyzwalanie w przypadku gdy wartość wyprowadzenia zmienia się z niskiej na wysoką
4. **FALLING** – wyzwalanie w przypadku zmiany wartości z wysokiej na niską

Jeżeli podczas wykonywania programu zajdzie potrzeba odwołania obsługi przerwania to umożliwia to następująca funkcja:

```
detachInterrupt(digitalPinToInterrupt(pin));
```

funkcja przyjmuje tylko jeden parametr – numer przerwannia, dla którego wyłączamy obsługę przerwań. Podobnie jak przy ustawianiu, numer ustalany jest przez funkcję na podstawie wyprowadzenia.

2.2.1 Przykład kodu zewnętrznego przerwania

```
1  volatile int state = LOW;
2  volatile unsigned int counter = 0;
3
4  void setup() {
5      pinMode(10, OUTPUT);
6      pinMode(7, INPUT_PULLUP);
7      attachInterrupt(digitalPinToInterrupt(7), blink, CHANGE);
8  }
9
10 void loop() {
11     if (counter >= 10)
12     detachInterrupt(digitalPinToInterrupt(7));
13 }
14
15 void blink() {
16     state = !state;
17     digitalWrite(10, state);
18     counter++;
19 }
```

UWAGI:

- funkcje przerwań powinny być jak najkrótsze, natomiast zawarty w nich kod powinien być możliwie najszybszy do wykonania.
- zmienne wykorzystywane w przerwaniach powinny być opatrzone dodatkowym słowem `volatile` (ulotny). Zastrzega ono, że zmienne te nie będą optymalizowane przez kompilator (podobnie jak to miało miejsce przy wstawkach assembler)

- Arduino wykonują tylko przerwania wedle priorytetów; kolejne przerwanie może zostać obsłużone jedynie wtedy, kiedy skończy się obsługa przerwania aktualnie trwającego
- większość funkcji liczących/opóźniających **NIE DZIAŁA** w funkcjach obsługi przerwań – dotyczy to `millis`(), `delay`() oraz częściowo `micros`(), które zaczyna działać błędnie po 1-2 ms od startu przerwania. Jedyną działającą funkcją czasową bezpośrednio w wątku jest `delayMicroseconds`().
- przerwania nie mogą przyjmować ani zwracać żadnych parametrów. Większość zmiennych wykorzystywanych w przerwaniach ma zatem charakter globalny (przynajmniej tych, których wartości mają być widoczne po zakończeniu przerwania).

3 Zadania do wykonania

1. Należy uruchomić i sprawdzić działanie pozostałych liczników (Timers) układu.
2. Sprawdzić działanie biblioteki Timer1 dla Arduino Leonardo (ostatni adres w materiałach).
3. Na podstawie wyżej wymienionej biblioteki napisać własne funkcje pozwalające na obsługę zegarów; jedna funkcja powinna dawać możliwość uruchamiania dowolnego z zegarów, zmiany wartości porównawczej bądź wyłączenia danego zegara.
4. Dodać funkcję czasowego zawieszenia przerwań generowanych przez zegary (maskowanie).
5. Napisać program czekający na wciśnięcie przycisku przez użytkownika. Jeżeli przycisk zostanie wciśnięty program ma odliczać czas do 60 sekund. Program można czasowo zawiesić poprzez naciśnięcie drugiego przycisk, bądź zresetować poprzez przytrzymanie pierwszego przycisku ok 5 sekund. Aktualny status liczenia należy wyświetlać poprzez konsolę (poprzednie zajęcia).
   DLA STUDENTÓW STUDIÓW NIESTACJONARNYCH:
6. Należy sporządzić sprawozdanie z przebiegu ćwiczenia według podanego szablonu (dostępny na stronie).

UWAGA: Układy badawcze budujemy w oparciu o doświadczenie z poprzednich laboratoriów!

4 Materiały wykorzystane przy opracowaniu

http://gammon.com.au/interrupts

https://pl.wikipedia.org/wiki/Przerwanie

https://oscarliang.com/arduino-timer-and-interrupt-tutorial/

http://www.robotshop.com/letsmakerobots/arduino-101-timers-and-interrupts

https://www.allaboutcircuits.com/technical-articles/using-interrupts-on-arduino/

https://www.arduino.cc/en/Reference/attachInterrupt

https://gonium.net/md/2006/12/27/i-will-think-before-i-code/

https://gonium.net/md/2007/04/18/tweaking-the-code/

http://playground.arduino.cc/Code/Interrupts

http://www.instructables.com/id/Arduino-Timer-Interrupts/

http://playground.arduino.cc/Code/Timer1