

Podstawy Systemów Mikroprocesorowych

Laboratorium

Laboratorium 2: Przerwania, timery i pomiar czasu

Autorzy: Dariusz Tefelski, Angelika Tefelska

Spis treści

1	Wprowadzenie	1
2	Wyposażenie stanowiska	1
3	Zadanie 1: Od pollingu do przerwania	2
4	Zadanie 2: Drgania styków i pomiar czasu	3
5	Zadanie 3: Timery	5
6	Zadanie 4: Wyświetlacz 4x7-segmentowy i multipleksowanie	6
7	Zadanie dodatkowe: Refleksomierz	9
8	Podsumowanie	10

1 Wprowadzenie

Cel ćwiczenia



Celem ćwiczenia jest poznanie mechanizmów pozwalających mikrokontrolerowi reagować na zdarzenia oraz wykonywać operacje w określonych chwilach czasu.

Na poprzednich zajęciach przyciski były sprawdzane cyklicznie wewnątrz głównej pętli programu. Takie rozwiązanie nazywa się *pollingiem*. Jest ono proste, ale wymaga ciągłego wykonywania instrukcji sprawdzających stan wejścia. W systemach mikroprocesorowych często korzystniejsza jest reakcja na zdarzenie dopiero wtedy, gdy ono rzeczywiście wystąpi.

Przerwanie i callback



Przerwanie powoduje chwilowe przerwanie zwykłego wykonywania programu i uruchomienie wskazanej funkcji obsługi. W MicroPythonie funkcję wywoływaną w odpowiedzi na zdarzenie często określa się nazwą *callback*. Po zakończeniu callbacku program kontynuuje zwykłą pracę.

Jak intuicyjnie zrozumieć przerwanie? Wyobraź sobie, że czytasz książkę i w pewnym momencie dzwoni telefon. Przestajesz czytać książkę, odbierasz telefon i po skończonej rozmowie wracasz do czytania książki. W tym przypadku przerwanie to dzwonek telefonu, rozmowa przez telefon to wywołanie funkcji *callback* a po jej zakończeniu dalsze czytanie to powrót do górnego nurtu wykonywania programu.

2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- płytką dydaktyczną z Raspberry Pi Pico 2 W,
- przewód USB i przewody połączeniowe,
- przyciski i diody LED dostępne na płycie dydaktycznej,
- wyświetlacz LED 4x7-segmentowy znajdujący się na płycie,
- Analog Discovery II do obserwacji drgań styków i przebiegów cyfrowych.

W przykładowej konfiguracji tego laboratorium przyjmujemy następujący podział GPIO:

Element	GPIO
Przycisk reakcji	GP14
Dioda sygnalizacyjna	GP15
Przycisk START	GP16
Segmenty A, B, C, D, E, F, G, DP	GP2–GP9
Cyfry DIGIT1–DIGIT4	GP10–GP13

Interpretacja



Elementy płytki dydaktycznej nie są na stałe przypisane do wymienionych GPIO. Połączenia wykonuje się przewodami. Jeżeli na stanowisku użyto innego zestawu pinów, należy odpowiednio zmienić numery w programach.

3 Zadanie 1: Od pollingu do przerwania

Na poprzednim laboratorium stan przycisku był sprawdzany wielokrotnie w pętli. W pierwszym zadaniu zastąpimy polling przerwaniem GPIO.

Przycisk jest połączony z rezystorem *pull-up*. W stanie spoczynku na wejściu występuje logiczna 1, a naciśnięcie zwiiera wejście do masy i powoduje zmianę $1 \rightarrow 0$. Taka zmiana jest nazywana **zbozcem opadającym**.

Zadanie 1.1 — Pierwsze przerwanie GPIO



Podłącz przycisk do GP14, a diodę LED do GP15. Uruchom poniższy program:

```

1 from machine import Pin
2 from utime import sleep
3
4 button = Pin(14, Pin.IN, Pin.PULL_UP)
5 led = Pin(15, Pin.OUT, value=1)
6
7 def button_irq(pin):
8     print("Przerwanie")
9     led.toggle() #Zmiana stanu na przeciwny 1->0 badz 0->1
10
11 button.irq(trigger=Pin.IRQ_FALLING, handler=button_irq)
12
13 while True:
14     print("Instrukcja 1")
15     sleep(1)
16     print("Instrukcja 2")
17     sleep(1)
18     print("Instrukcja 3")
19     sleep(1)

```

Uruchom program i obserwuj komunikaty w konsoli. Naciskaj przycisk w róż-

 nych momentach wykonywania programu. Sprawdź, czy obsługa przerwania wymaga oczekiwania na zakończenie funkcji `sleep()` w pętli głównej. Czy zawsze tylko raz wywoływało się przerwanie po naciśnięciu przycisku?

Co oznacza `button_irq(...)`?

 Metoda `irq()` konfiguruje zdarzenie generujące przerwanie:

- `trigger=Pin.IRQ_FALLING` — reakcja na zbocze opadające,
- `handler=button_irq` — funkcja, która zostanie wywołana po wystąpieniu zdarzenia.

Funkcja obsługi musi przyjmować jeden argument. W przypadku przerwania GPIO MicroPython przekazuje do niej obiekt pinu, który wywołał przerwanie. Dlatego funkcję zapisujemy jako `def button_irq(pin):`, nawet jeśli w danym programie nie korzystamy z argumentu `pin`.

Callback powinien być krótki

 W funkcji obsługi przerwania wykonuj możliwie mało operacji. Długie obliczenia, opóźnienia oraz rozbudowane operacje wejścia/wyjścia mogą opóźniać obsługę kolejnych zdarzeń. W większych programach callback często jedynie ustawia flagę lub zapisuje wynik pomiaru, a właściwe działania wykonuje główna część programu.

4 Zadanie 2: Drgania styków i pomiar czasu

Mechaniczny przycisk nie przechodzi zawsze idealnie i natychmiastowo pomiędzy stanem 0 i 1. Podczas naciskania i zwalniania styki mogą przez krótki czas wielokrotnie zwiierać się i rozwierać. Zjawisko to nazywamy **drzaniami styków** (ang. *contact bounce*).

Zadanie 2.1 — Licznik naciśnięć

 Zmodyfikuj program z poprzedniego zadania tak, aby każde przerwanie zwiększało zmienną `counter`. W głównej pętli wyświetlaj jej aktualną wartość. Naciśnij przycisk kilkanaście razy. Sprawdź, czy licznik zawsze zwiększa się dokładnie o 1 na jedno fizyczne naciśnięcie.

Pomiar 2.1 — Czy przycisk naprawdę zmienia stan tylko raz?

 Korzystając z Analog Discovery II, zarejestruj sygnał na wejściu przycisku podczas pojedynczego naciśnięcia i zwolnienia. Spróbuj zaobserwować krótkie, wielokrotne zmiany stanu.

Do odmierzania czasu w MicroPythonie można wykorzystać licznik milisekund:

```
1 from utime import ticks_ms, ticks_diff
```

```

2
3 start = ticks_ms()
4 # ...
5 elapsed = ticks_diff(ticks_ms(), start)

```

Dlaczego używamy ticks_diff()?



Licznik zwracany przez `ticks_ms()` po bardzo długim czasie przepełnia się i zaczyna ponownie od początku zakresu. Funkcja `ticks_diff()` poprawnie oblicza różnicę pomiędzy dwiema wartościami licznika również w pobliżu tego przepełnienia. Dlatego do pomiaru odstępów czasu jest bezpieczniejsza niż zwykłe odejmowanie.

Zadanie 2.2 — Programowe usuwanie drgań styków



Wprowadź minimalny odstęp pomiędzy zaakceptowanymi przerwaniem. Wykorzystaj poniższy schemat:

```

1 from machine import Pin
2 from utime import ticks_ms, ticks_diff, sleep
3
4 button = Pin(14, Pin.IN, Pin.PULL_UP)
5
6 counter = 0
7 last_press = 0
8
9 def button_irq(pin):
10     global counter, last_press
11
12     now = ticks_ms()
13     if ticks_diff(now, last_press) > 50:
14         counter += 1
15         last_press = now
16
17 button.irq(trigger=Pin.IRQ_FALLING, handler=button_irq)
18
19 while True:
20     print("Counter", counter)
21     sleep(1)
22

```

Podwójne sprawdzenie stanu przycisku z reguły pomaga wyeliminować efekt drgania styków. Sprawdź działanie dla kilku wartości czasu eliminacji drgań, np. 10 ms, 50 ms i 200 ms. Która wartość zapewnia poprawne działanie, a jednocześnie nie utrudnia szybkiego naciskania przycisku?

5 Zadanie 3: Timery

Timer umożliwia wykonanie funkcji po określonym czasie lub cyklicznie w zadanym odstępie. Na Raspberry Pi Pico 2 W w MicroPythonie korzystamy z klasy `Timer`.

Dwa podstawowe tryby pracy to:

- `Timer.PERIODIC` — callback wykonywany cyklicznie,
- `Timer.ONE_SHOT` — callback wykonywany tylko jeden raz po upływie zadanego czasu.

Zadanie 3.1 — Timer okresowy



Uruchom program, który zmienia stan diody co 250 ms bez użycia funkcji `sleep()` w głównej pętli:

```
1 from machine import Pin, Timer
2 from utime import sleep
3
4 led = Pin(15, Pin.OUT, value=1)
5
6 def timer_callback(timer):
7     led.toggle()
8
9 timer = Timer()
10 timer.init(period=250, mode=Timer.PERIODIC, callback=
    timer_callback)
11
12 while True:
13     print("Petla glowna moze wykonywac inne zadania")
14     sleep(1)
```

Pomiar 3.1 — Okres sygnału z timera



Podłącz oscyloskop do pinu sterującego diodą. Zmierz okres i częstotliwość przebiegu. Porównaj wynik z wartością oczekiwaną na podstawie programu. Następnie zmień `period` na 100 ms i ponownie wykonaj pomiar.

Zadanie 3.2 — Timer jednorazowy

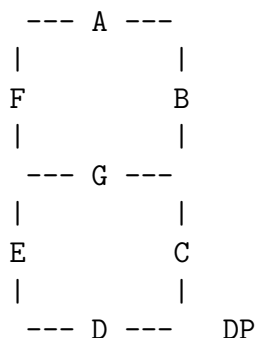


Zmodyfikuj program tak, aby po uruchomieniu odczekał 2 s, a następnie jednokrotnie zapalił diodę. Użyj trybu `Timer.ONE_SHOT`.

Po wykonaniu callbacku dioda ma pozostać zapalona. Nie używaj `sleep(2)` do realizacji opóźnienia.

6 Zadanie 4: Wyświetlacz 4x7-segmentowy i multipleksowanie

Płytką dydaktyczną jest wyposażona w czterocyfrowy wyświetlacz LED 7-segmentowy. Każda cyfra składa się z segmentów A–G oraz kropki dziesiętnej DP. Linie segmentów są wspólne dla wszystkich czterech cyfr, natomiast linie DIGIT1–DIGIT4 wybierają aktualnie aktywną cyfrę.



W proponowanej konfiguracji połącz:

Segment	A	B	C	D	E	F	G	DP
GPIO	2	3	4	5	6	7	8	9

Cyfra	DIGIT1	DIGIT2	DIGIT3	DIGIT4
GPIO	10	11	12	13

Na płytce linie sterujące wyświetlaczem pracują w logice aktywnej stanem niskim: dla aktywnego segmentu lub cyfry ustawiamy wartość 0.

Zadanie 4.1 — Wyświetlenie jednej cyfry



Podłącz wyświetlacz zgodnie z tabelami. Następnie uruchom program, który ustawi cyfrę 2 na pierwszym wyświetlaczu:

```
1 from machine import Pin
2
3 segments = [Pin(pin, Pin.OUT, value=1) for pin in range(2,
4               10)]
5
6 digits = [Pin(pin, Pin.OUT, value=1) for pin in range(10,
7               14)]
8
9 # Kolejność: A, B, C, D, E, F, G, DP
10 pattern_2 = (0, 0, 1, 0, 0, 1, 0, 1)
11
12 digits[0].value(0) #Włączenie cyfry nr 1
13
14 while True:
```



```
for i, seg in enumerate(segments):
    seg.value(pattern_2[i])
```

Jeżeli chcemy niezależnie wyświetlić cztery cyfry, nie możemy pozostawić wszystkich linii DIGIT aktywnych jednocześnie, ponieważ wszystkie pozycje otrzymałyby ten sam wzorec segmentów. Zamiast tego szybko przełączamy kolejne cyfry:

DIGIT1 → DIGIT2 → DIGIT3 → DIGIT4 → DIGIT1 → ...

Zjawisko bezwładności wzroku sprawia, że przy odpowiednio dużej częstotliwości człowiek widzi cztery świecące cyfry jednocześnie. Taki sposób sterowania nazywamy **multipleksowaniem**.

Tablica wzorców cyfr



W logice aktywnej stanem niskim można użyć następujących wzorców. Kolejność elementów to A, B, C, D, E, F, G, DP.

```
1 PATTERNS = (
2     (0,0,0,0,0,0,1,1), # 0
3     (1,0,0,1,1,1,1,1), # 1
4     (0,0,1,0,0,1,0,1), # 2
5     (0,0,0,0,1,1,0,1), # 3
6     (1,0,0,1,1,0,0,1), # 4
7     (0,1,0,0,1,0,0,1), # 5
8     (0,1,0,0,0,0,0,1), # 6
9     (0,0,0,1,1,1,1,1), # 7
10    (0,0,0,0,0,0,0,1), # 8
11    (0,0,0,0,1,0,0,1)  # 9
12 )
```

Zadanie 4.2 — Multipleksowanie z użyciem dwóch timerów



W tym zadaniu wykorzystamy **dwa niezależne timery okresowe**. Pierwszy będzie odpowiedzialny za szybkie odświeżanie kolejnych pozycji wyświetlacza, natomiast drugi będzie zmieniał liczbę, którą chcemy wyświetlić.

Przygotuj zmienne przechowujące aktualnie wyświetlaną wartość oraz numer aktualnie odświeżanej pozycji:

```
1 display_value = 0
2 current_digit = 0
```

Napisz funkcję pomocniczą, która na podstawie liczby z zakresu 0–9999 utworzy listę czterech cyfr. Przykładowo dla liczby 37 powinna powstać lista:

[0, 0, 3, 7]

Możesz wykorzystać dzielenie całkowite // oraz operator reszty z dzielenia %.



Następnie przygotuj dwa callbacki:

1. **callback odświeżania wyświetlacza** — przy każdym wywołaniu:
 - (a) wyłącza wszystkie cztery cyfry,
 - (b) wybiera odpowiednią cyfrę z aktualnej wartości,
 - (c) ustawia wzorzec segmentów,
 - (d) włącza jedną pozycję wyświetlacza,
 - (e) przechodzi do kolejnej pozycji;
2. **callback licznika** — zwiększa wyświetlaną wartość o 1. Po przekroczeniu 9999 licznik powinien rozpocząć zliczanie ponownie od 0.

Uruchom dwa timery okresowe:

```
1 refresh_timer = Timer()
2 counter_timer = Timer()
3
4 refresh_timer.init(period=3, mode=Timer.PERIODIC, callback
   =refresh_display)
5
6 counter_timer.init(period=1000, mode=Timer.PERIODIC,
   callback=increment_counter)
```

Pierwszy timer powinien działać z okresem około 2–4 ms, natomiast drugi początkowo z okresem 1000 ms. Dzięki temu wyświetlacz jest odświeżany kilkaset razy na sekundę, podczas gdy prezentowana liczba zwiększa się raz na sekundę. Sprawdź działanie programu, a następnie:

1. zmień okres timera `counter_timer` na 200 ms i obserwuj licznik,
2. zwiększaj okres timera `refresh_timer`, aż migotanie wyświetlacza stanie się zauważalne,
3. sprawdź, czy zmiana szybkości zliczania wpływa na częstotliwość odświeżania wyświetlacza.

Dwa timery — dwa niezależne zadania



W tym programie oba timery działają niezależnie od siebie. Timer `refresh_timer` realizuje zadanie wymagające dużej częstotliwości — multipleksowanie wyświetlacza. Timer `counter_timer` wykonuje znacznie wolniejsze zadanie — zmianę wyświetlanej wartości.

Jest to przykład sytuacji, w której mikrokontroler wykonuje kilka czynności o różnych skalach czasowych bez stosowania funkcji `sleep()` do odmierzania czasu w głównej części programu.

Callbacks timerów powinny być krótkie



Funkcje wywoływane przez timery powinny wykonywać tylko niezbędne operacje. W szczególności callback odpowiedzialny za odświeżanie wyświetlacza jest wywoływany bardzo często, dlatego nie należy umieszczać w nim funkcji



`print()`, opóźnień ani rozbudowanych obliczeń.

Dlaczego najpierw wyłączamy wszystkie cyfry?



Zmiana segmentów przy aktywnej cyfrze może na krótko powodować wyświetlenie niepożądanego wzorca. Wyłączenie wszystkich pozycji przed zmianą segmentów ogranicza efekt tzw. *ghostingu*.

7 Zadanie dodatkowe: Refleksomierz

Na zakończenie wykorzystamy przerwanie GPIO, timer jednorazowy, pomiar czasu i wyświetlacz 7-segmentowy w jednym urządzeniu.

Zadanie 5.1 — Pomiar czasu reakcji



Zbuduj refleksomierz spełniający następującą specyfikację:

1. naciśnięcie przycisku START rozpoczyna próbę,
2. dioda sygnalizacyjna pozostaje zgaszona przez losowy czas z zakresu około 1–3 s,
3. odliczanie losowego czasu zrealizuj za pomocą timera `Timer.ONE_SHOT`,
4. po upływie czasu timer zapala diodę i zapisuje chwilę rozpoczęcia pomiaru za pomocą `ticks_ms()`,
5. naciśnięcie przycisku reakcji jest obsługiwane przez przerwanie GPIO,
6. czas reakcji oblicz za pomocą `ticks_diff()`,
7. wynik w milisekundach wyświetl na wyświetlaczu 4x7-segmentowym,
8. kolejne naciśnięcie START rozpoczyna nową próbę.

Wynik powinien pozostać na wyświetlaczu do momentu rozpoczęcia kolejnej próby.

Podpowiedź organizacyjna — stany programu



W wygodnym rozwiązaniu program może przechowywać aktualny stan, np.:

IDLE → WAIT → GO → RESULT

Przerwanie od przycisku reakcji może wtedy zachowywać się inaczej zależnie od tego, czy system czeka na sygnał START, czy pomiar już się rozpoczął.

Rozszerzenie dla chętnych



Wykonaj 5 prób i zapamiętaj najlepszy wynik. Po każdej próbie pokaż aktualny czas, a po zakończeniu serii wyświetl najlepszy rezultat.

8 Podsumowanie

Po tych zajęciach powinieneś umieć



- skonfigurować przerwanie GPIO na wybrane zbocze,
- wyjaśnić, czym różni się polling od obsługi przerwaniowej,
- ograniczyć wpływ drgań styków przycisku,
- mierzyć odstępy czasu funkcjami `ticks_ms()` i `ticks_diff()`,
- uruchomić timer okresowy i jednorazowy,
- wyjaśnić zasadę multipleksowania wyświetlacza 4x7-segmentowego,
- wykorzystać dwa niezależne timery okresowe w jednym programie,
- wykorzystać timer do cyklicznego odświeżania wyświetlacza,
- połączyć kilka mechanizmów asynchronicznych w jeden program.

Uwaga



Przed opuszczeniem stanowiska zatrzymaj program, wyczyść pamięć (patrz niżej) i uporządkuj przewody oraz elementy używane podczas ćwiczenia.

Wyczyść mikrokontroler w ten sam sposób co na laboratoriach nr 0:

1. Pobierz plik `flash_nuke.uf2` z sekcji *Resetting flash memory* dokumentacji Raspberry Pi:
<https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html#reset-flash-memory>
2. Ustaw przełącznik zasilania używanego modułu Pico w pozycji **OFF**.
3. Naciśnij i przytrzymaj przycisk **BOOTSEL**, a następnie włącz zasilanie modułu przełącznikiem **ON**.
4. Po pojawieniu się dysku USB zwolnij przycisk **BOOTSEL**.
5. Skopiuj plik `flash_nuke.uf2` na wykryty dysk. Mikrokontroler automatycznie uruchomi program czyszczący i wyczyści pamięć flash.
6. **Nie wgrywaj ponownie MicroPythona.** Następna grupa sama go wgra przed rozpoczęciem laboratoriów.