Akademia Górniczo-Hutnicza
w Krakowie
Katedra Elektroniki
WIET

Laboratorium TM2

Ćwiczenie 3

Przerwania i licznik PIT

Autor: Paweł Russek

wer. 21.10.2020

1 Wprowadzenie

1.1 Cel ćwiczenia

W toku realizacji poniższego ćwiczenia, student zapozna się z podstawowym mechanizmem pracy mikrokontrolerów jakim są wyjątki i przerwania. Działanie mechanizmu przerwań zostanie wyjaśnione na przykładzie mikrokontrolerów firmy Freescale, z rodziny Kinetis L, zbudowanych na bazie rdzenia procesora ARM Cortex-M0+. Instrukcja przedstawia w skrócie podstawy działania przerwań i podaje sposoby w jaki przerwania są programowane w układach Kinetis L. Poruszane są również ważne zagadnienia dot. zarządzania zadaniami aplikacji przy pomocy przerwań i programowania aplikacji energooszczędnych. Dodatkowo, student zapozna się z licznikiem PIT (ang. Periodic Interrupt Timer), w który wyposażone są mikrokontrolery z rodziny Kinetis L. Zostanie zrealizowany przykładowy kod programu, wykorzystujący przerwania od układów GPIO i licznika PIT.

1.2 Wymagania wstępne

- Komputer PC z zainstalowanym środowiskiem Keil uVision
- Zestaw uruchomieniowy FRDM-KL05Z
- Znajomość podstaw programowania układu FRDM-KL05Z w języku C

1.3 Literatura

- Joseph Yiu, The Definitive Guide to the ARM Cortex-M0, Elsevier, 2011
- KL05 Sub-Family Reference Manual, Freescale Semiconductor
- Kinetis L Peripheral Module Quick Reference, Freescale Semiconductor
- Power Management for Kinetis and ColdFire+ MCUs. , Freescale Semiconductor

2 Podstawy teoretyczne

2.1 Wyjątki i przerwania

Wyjątki to zdarzenia w systemie procesorowym powodujące zmianę kolejności realizowanych przez CPU instrukcji. Procesor wstrzymuje normalną pracę związaną z wykonywaniem kolejnych rozkazów - zgodnie z zawartością licznika rozkazów (ang. program counter – PC) i rozpoczyna wykonywanie instrukcji zdefiniowanych w procedurze obsługi wyjątku (ang. exception handler). Po zakończeniu procedury obsługi, procesor wraca do wykonywania instrukcji przerwanego fragmentu programu. Zdarzenia powodujące przerwanie wykonywania programu głównego mogą być wewnętrzne (związane z CPU i wykonywanym przez niego programem) lub zewnętrzne (związane z urządzeniami peryferyjnymi). Kiedy zdarzenie ma zewnętrzne źródło jest potocznie nazywany przerwaniem lub żądaniem przerwania (ang, interrupt request – IRQ). W przypadku, kiedy zdarzenie związane jest z wykonywanym przez procesor kodem (np. dzielenie przez zero), to jest nazywane wyjątkiem. Procedury obsługi przerwań lub wyjątków są określane jako ISR (ang. Interrupt Service Routine).

2.2 Wyjątki w procesorze ARM Cortex-M0

Procesory ARM serii Cortex-M są wyposażone we wbudowany blok kontrolera przerwań (ang. Nested Vector Interrupt Controller – NVIC). Kiedy NVIC przyjmie przerwanie, to komunikuje o tym fakcie blok CPU, aby mógł on rozpocząć wykonywanie właściwej procedury obsługi przerwania ISR. Procesor Cortex-M rozróżnia 32 różne źródła przerwań zewnętrznych. Dodatkowo obsługiwane są również wyjątki. Źródła przerwań i wyjątków są ponumerowane: wyjątki mające źródło w CPU mają numery od 1 do 15, a przerwania zgłaszane przez urządzenia peryferyjne i porty I/O mają numery od 16 do 47. Są to tzw. wektory wyjątków (wektory 1-47).

Przykładowo CPU może generować wyjątek zegara systemowego (ang. System Tick Timer). Wyjątek ten określany jest jako SysTick i jest generowany wewnątrz NVIC. Numer wektora wyjątku SysTick ma numer 15. Zegar systemowy jest źródłem regularnych, okresowych przerwań dla oprogramowania systemowego.

Exception Type	Exception Number	Description
Reset	1	Power on reset or system reset.
NMI	2	Nonmaskable interrupt—highest priority exception that cannot be disabled. For safety critical events.
Hard fault	3	For fault handling—activated when a system error is detected.
SVCall	11	Supervisor call—activated when SVC instruction is executed. Primarily for OS applications.
PendSV	14	Pendable service (system) call—activated by writing to an interrupt control and status register. Primarily for OS applications.
SysTick	15	System Tick Timer exception—typically used by an OS for a regular system tick exception. The system tick timer (SysTick) is an optional timer unit inside the Cortex-M0 processor.
IRQ0 to IRQ31	16 - 47	Interrupts—can be from external sources or from on-chip peripherals.

Tabela 1: Wyjątki i przerwania ARM Cortex M0

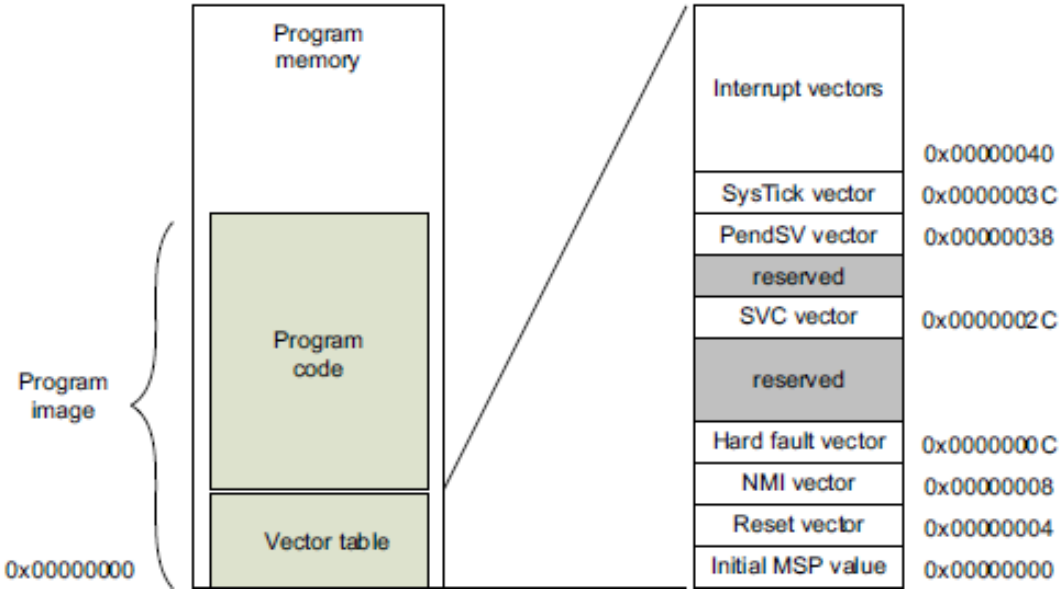
2.3 Programowanie wyjątków w CMSIS

Biblioteka programistyczna CMSIS (ang. Cortex Microcontroller Software Interface Standard) dostarcza funkcji standardowych do obsługi urządzeń systemowych mikrokontrolerów Cortex. CMSIS zawiera również funkcje służące do programowania bloku NVIC i przerwania SysTick. Dzięki CMSIS, procedury obsługi przerwań ISR mogą być programowane bezpośrednio w języku C, bez potrzeby sięgania po assembler.

2.4 Tablica wektorów przerwań

Organizacja tablicy wektorów przerwań i nazwy przerwań są różne dla różnych mikrokontrolerów. W ARM Cortex-M, tablica zawiera adresy pamięci programu, gdzie znajdują się początki procedur ISR dla odpowiednich źródeł wyjątków i przerwań. Właściwa definicja tablicy znajduje się w kodzie startowym każdego programu, który jest kompilowany dla danego procesora. Na przykład (patrz Rys. 1) adres początkowy ISR SysTick Timer znajduje się pod adresem 0x0000003C.

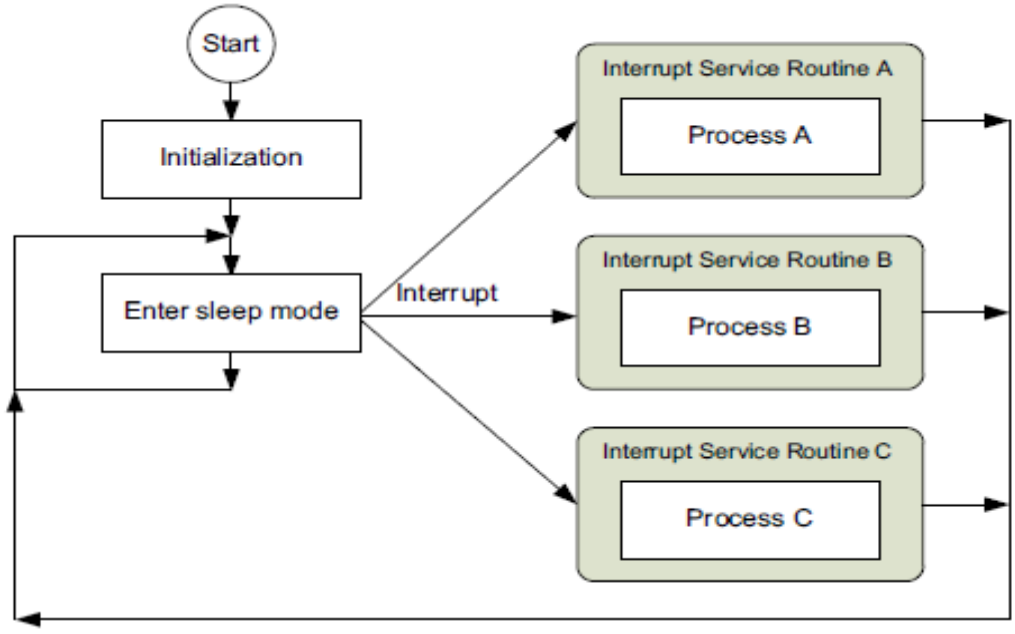
W projekcie uVision, dla mikrokontrolera MKL05Z32, tablica wektorów przerwań i ISR, są wstępnie zdefiniowane w pliku startup_MKL05Z4.s. W pliku startup_MKL05Z4.s każdy ISR ma zdefiniowaną unikalną nazwę funkcji, której adres początkowy jest umieszczony w odpowiednim miejscu tablicy wektorów przerwań. Programista C może nadpisać definicję tych funkcji ISR i w ten sposób zmusić kompilator do umieszczenia własnej funkcji ISR w tablicy wektorów.



Rysunek 1: Tablica wektorów przerwań w ARM Cortex M

2.5 Programowanie oparte na przerwaniach

Aby przyśpieszyć interakcję mikrokontrolera z otoczeniem oraz obniżyć średni prąd jaki mikrokontroler Cortex-M0 pobiera ze źródła zasilania należy przygotowywać aplikacje w taki sposób, aby pracowały one „na przerwaniach". W takim sposobie programowania procesor może być wprowadzany w stan uśpienia i budzić się w celu realizacji użytecznych operacji tylko w przypadku zgłoszenia odpowiedniego przerwania.



Rysunek 2. Programowanie oparte na przerwaniach

3 Przygotowanie do laboratorium

3.1 Ze strony laboratorium pobierz plik zip file (*t3_files.zip*).

3.2 Rozpakuj plik do nowo utworzonego katalogu na komputerze PC:

Np. Do katalogu: C:/MDK_ARM/<dzien>_<godzina>_<nazwiska_studentów>/IntLabProj

3.3 W katalogu znajdź i otwórz projekt programu uVision: 'int_tutorial.uvprojx'.

Projekt zawiera kilka plików. W katalogu 'src' mamy:

- main.c – zawiera funkcję main(),
- fsm.c – zawiera funkcję związane z realizacją maszyny stanów FSM (ang. Finite State Machine) i miganiem diodą RGB LED,
- buttons.c – zawiera funkcje obsługi zewnętrznych przycisków,
- pit.c – zawiera funkcje obsługi licznika PIT (ang. Periodic Interrupt Timer).

3.5 Zbuduj projekt (Build all) i załaduj program na płytkę FRDM. Jeżeli ćwiczenie przebiega poprawnie, to po RESET diody RGB powinna trzy razy zamigać w kolorze białym.

3.4 Otwórz plik main.c.

Każdy moduł programu (tj. 'fsm.c', 'buttons' i 'pit.c') posiada funkcję inicjującą, która jest uruchamiana na początku funkcji 'main()'. Później funkcja 'main()' zaczyna wykonywać pętlę nieskończoną (patrz rysunek 2).

3.5 Otwórz plik fsm.c i znajdź funkcję *fsmInitialize()*. Maszyna FSM realizuje miganie RGB LED, więc inicjalizacja polega na przygotowaniu GPIO do sterowania diodą:

- Zaprogramowanie obwodu kontrolującego sygnał zegarowy tak, aby włączyć zegar GPIO,
- konfiguracja GPIO,
- uruchomienie procedury powitalnej – migania diodami, która pozwala na weryfikację poprawności programowania płytki FRDM.

3.6 (ToDo 3.6) Zlokalizuj powyżej wymienione kroki procedury *fsmInitialize()* i dostosuj procedurę powitalną według sekwencji własnego pomysłu.

Uwaga: Modyfikacje, które powinieneś zrealizować w przykładowo dostarczonym kodzie programu są oznaczone w komentarzach przez znacznik 'ToDo <n>'. Numer <n> jest zgodny z numerem polecenia zawartym w tej instrukcji.

4 Programowanie wyjątku SysTick

4.1 Tablica przerwań

Tablica przerwań aplikacji jest zdefiniowana w sekwencji startowej kodu. Kod startowy znajduje się w pliku startup_MKL05Z4.s.

4.1.1 Otwórz startup_MKL05Z4.s

Tablicę przerwań można łatwo znaleźć, ponieważ zaczyna się ona od etykiety '__Vectors'. Sygnał RESET jest specjalnym wyjątkiem, a jego wektor znajduje się pod adresem 0x000_0004.

Uwaga: Wartość wpisana pod adresem 0x0000_0000 to początkowy adres stosu mikrokontrolera ARM Cortex-M0.

Po RESET, zanim program rozpocznie wykonywanie funkcji main(), najpierw wykonuje kod programu oznaczony jako 'Reset_Handler'. Zanim procesor zacznie wykonywać funkcję main() zdefiniowaną przez użytkownika, najpierw musi wykonać operacje inicjalizujące system.

4.1.2 Sprawdź nazwę funkcji inicjalizującej SystemInit(), która jest wykonywana po RESET, a następnie przeglądnij jej kod w pliku *system_MKL05Z4.c*

4.2 Włączanie wyjątku systemowego SysTick

Wektor wyjątku SysTick jest również umieszczony w tablicy wektorów.

4.2.1 Zlokalizuj nazwę funkcji, która jest zdefiniowana w *startup_MKL05Z4.s* jako funkcja obsługi wyjątku ISR dla SysTick.

Zwróć uwagę na klauzulę [WEAK] w definicji funkcji obsługi przerwania. Oznacza ona, że nazwa funkcji oraz jej definicja może być nadpisana przez definicję, którą programista umieści w swoim kodzie.

4.2.2 Zmień nazwę funkcji *runThisISREvery1ms()* w pliku main.c na nazwę właściwą funkcji obsługi wyjątku SysTick, która jest zdefiniowana w pliku startup_MKL05Z4.s. **[ToDo 4.1]**

W bibliotece CMISIS jest zdefiniowana funkcja 'SysTick_Config()'. Funkcja ta służy

inicjalizacji licznika systemowego i generowanego przez niego wyjątku. Funkcja 'SysTick_Config()' uruchamia licznik i aktywuje generowanie przez niego wyjątków. Parametrem funkcji jest liczba taktów zegara systemowego pomiędzy kolejnymi wywołaniami wyjątku.

Wartość globalnej zmiennej systemowej SystemCoreClock jest równa częstotliwości zegara systemowego i może być pomocna przy ustaleniu argumentu funkcji 'SysTick_Config()' dla danego okresu czasu między wyjątkami.

4.2.3 Usuń komentarz z linii kodu wywołującej SysTick_Config() oraz wpisz właściwą wartość parametru funkcji SysTick_Config(), tak aby przerwanie od licznika systemowego było wywoływane co 1 ms **[ToDo 4.2]**

4.2.4. Zbuduj aplikację i uruchom program na płytce the FRDM. Diody powinny się zmieniać w sekwencji: CZERWONA, ZIELONA, NIEBIESKA.

4.2.5. Zmodyfikuje program tak, aby diody zmieniały się co 1 s. **[ToDo 4.3]**

5 Przerwania od GPIO

Przerwania generowane przez zmiany stanów logicznych na pinach są zaimplementowane jako mechanizm portów GPIO. Moduł PORT definiuje następujące funkcje każdego pinu:

- włączenie przerwania dla wybranego pinu,
- wybranie aktywności przerwania: zbocze narastając, opadające, obydwa zbocza, poziom wysoki, poziom niski
- rezystor podciągający do zasilania lub do masy.

Bity kontrolne służące do konfiguracji pinów są umieszczone w rejestrze PCR (ang. Port Control Register) osobnym dla każdego pinu. Każdy pin ma swój własny 32 bitowy rejestr w module PORT. Będziemy używać przycisku do wygenerowania przerwania o nazwie PORTB.

Funkcję poszczególnych bitów w PCR można znaleźć w tabeli 'PORTx_PCRn field descriptions' w dokumencie 'KL05 Sub-Family Reference Manual' w sekcji PCR Register Bit Fields.

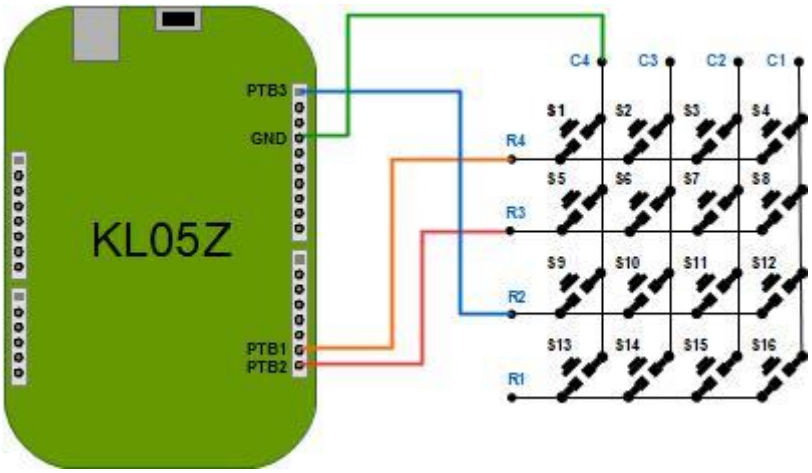
Ćwiczenie 5.1

W tabeli umieszczonej poniżej proszę uzupełnić opis wymienionych bitów rejestru PCR.

Lp.	Nazwa pola	Liczba bitów	Opis
1	MUX		
2	PE		
3	PS		
4	IRQC		
5	ISF		

5.1 Podłączenie przycisków do FRDM-KL05

Następne punkty ćwiczenia wymagają podłączenia do płytki FRDM-KL05 zewnętrznej klawiatury. Proszę podłączyć klawiaturę zgodnie z przedstawionym schematem (Rys. 3).



Rysunek 3. Podłączenie klawiatury do FRDM-KL05

5.2 Włączenie przerwania od pinu

Aby skonfigurować moduł GPIO do obsługi przerwań należy wykonać następujące kroki:

Konfiguracja pinu jako GPIO **przykładowy kod: PORTC->PCR[4] = PORT_PCR_MUX(0x1);**
Włączenie odpowiedniego przerwania dla danego pinu **przykładowy kod: PORTC->PCR[4] = PORT_PCR_IRQC(xxx);**
Wykasowanie oczekujących przerwań w NVIC **przykładowy kod: NVIC_ClearPendingIRQ(PORTC_D_IRQ_NBR);**
Inicjalizacja NVIC aby włączyć wybrany numer PORT IRQ **przykładowy kod: NVIC_EnableIRQ(PORTC_D_IRQ_NBR);**

5.2.1 W pliku buttons.c, zdefiniuj właściwy numer wektora przerwania dla przerwania PORTB **[ToDo 5.1]**. Można zastosować wstępnie definiowany symbol dla tej wartości, który można znaleźć w pliku MKL05Z4.h

5.2.2 Wprowadź właściwą nazwę funkcji ISR dla PORTB **[ToDo 5.2]**.

5.2.3 Upewnij się, że flaga ISR (ang. Interrupt Service Flag) jest kasowana w procedurze ISR. Jeżeli nie będzie to miało miejsca, to przerwanie zostanie zgłoszone ponownie (dla tego samego zdarzenia) zaraz po zakończeniu procedury ISR. **[ToDo 5.3]**

5.2.4 Ustaw odpowiedni bit w rejestrze PCR, aby uaktywnić rezystor podciągający. **[ToDo 5.4]**

5.2.5 Ustaw odpowiedni bit w rejestrze PCR, aby wybrać rezystor podciągający do zasilania. [**ToDo 5.5]**

5.2.6 Wybierz właściwą wartość pola bitowego IRQC, tak aby wybrać przerwanie aktywne opadającym zboczem sygnału na pinie. **[ToDo 5.6]**

5.2.7 Usuń komentarz i włącz do kodu funkcję inicjalizacji przycisków w pliku main.c. **[ToDo 5.7]**

5.2.8 Zbuduj i uruchom aplikację. Powinieneś móc kontrolować sekwencję diod LED za pomocą przycisków SW1 (prędkość) i SW5 (start/stop)

6 PIT Timer

6.1 Wprowadzenie

W tej części zostanie zademonstrowane, jak zaprogramować przerwanie z licznika PIT. Licznik PIT będzie wykorzystany (zamiast SysTick) do wykonywania zmian na diodzie RGB LED. W przeciwieństwie do SysTick, który jest częścią rdzenia przetwarzającego ARM, zegar PIT jest modułem peryferyjnym (jak GPIO). Podobnie jak SysTick, PIT jest prostym 32-bitowym zegarem przeznaczonym do generowania przerwań w programowalnych okresach. Jednak w przeciwieństwie do SysTick, który jest taktowany zegarem systemowym (maksymalnie 48 MHz dla MKL05Z32); PIT jest taktowany zegarem magistrali (zwykle połowa częstotliwości zegara systemu). PIT posiada dwa 32-bitowe liczniki czasowe (dwa kanały), które mogą pracować niezależnie lub w łańcuchu jako licznik 64-bitowy. Licznik 64-bitowy umożliwia zastosowanie PIT do realizacji licznika o długim okresie generacji zdarzeń.

Licznik PIT ma bardzo prostą strukturę. Każdy kanał tego licznika posiada niezależny zestaw rejestrów konfiguracyjnych:

- LDVAL – rejestr 32-bitowy w którym można ustawić okres generowania zdarzeń

$$T = \frac{LDVAL - 1}{freq_{bus\ clock}}$$

- TCTRL – rejestr kontrolny. Zawiera flagi sterujące licznikiem. Flaga TIE służy do włączania/wyłączania przerwań od licznika.
- TFLG – rejestr statusowy. Zawiera flagi stanu pracy licznika. Flaga TIF służy do weryfikacji nadejścia przerwania (przy odczycie) oraz do skasowania źródła przerwania (przy zapisie jedynki jest zerowana).
- CVAL – rejestr 32-bitowy zawiera aktualną wartość licznika (nie używana w prezentowanym ćwiczeniu).

Ponadto moduł PIT posiada również rejestr MCR, który jest wspólny dla wszystkich kanałów. Zawiera flagi FRZ i MDIS, które służą do tego, aby odpowiednio sterować pracą start/stop licznika PIT i działaniem PIT podczas debugowania.

6.2 Procedura konfiguracji licznika PIT

Aby skonfigurować licznik PIT należy wykonać następujące kroki.

Podłączenie zegara do modułu (inny niż zegar taktujący PIT) przykładowy kod: **SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;**
Włączenie modułu, wstrzymanie liczników w trybie debug przykładowy kod: **PIT->MCR &= ~PIT_MCR_MDIS_MASK;**
Inicjalizacja wartości LDVAL dla danego kanału PIT przykładowy kod: **IT->CHANNEL[0].LDVAL = PIT_LDVAL_TSV(period);**
Włączenie przerwania od danego kanału PIT przykładowy kod: **PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;**
Włączenie przerwań od PIT w module NVIC przykładowy kod: **NVIC_SetPriority(PIT_IRQn, 128); // 0, 64, 128 or 192** **NVIC_ClearPendingIRQ(PIT_IRQn);** **NVIC_EnableIRQ(PIT_IRQn);**

6.3 Programowanie PIT

6.3.1 W pliku w MKL05Z4.h znajdź numer (albo symbol) dla IRQ licznika PIT. Przypisz tą wartość do symbol myPIT_IRQn. **[ToDo 6.1]**

6.3.2 Włącz zegar modułu PIT w rejestrze SCGC6. **[ToDo 6.2]**

6.3.3 Wyzeruj flagę MDIS w rejestrze MCR. **[ToDo 6.3]**

6.3.4 Ustaw flagę FRZ w rejestrze MCR. **[ToDo 6.4]**

6.3.5 Usuń komentarz i uaktywnij funkcję pitInitialize(<period>) w pliku main.c. Ustaw odpowiednio parametr funkcji pitInitialize tak aby okres przerwań wynosił 1 ms. Załóż, że częstotliwość zegara wejściowego PIT (zegar magistrali) wynosi 20.9 MHz (zegar magistrali). **[ToDo 6.4]**

6.3.6 Usuń komentarz z linii z funkcją startPIT () w pliku main.c. **[ToDo 6.5]**

6.3.7 Aby wyłączyć przerwania od SysTick, wstaw komentarz w linii wywołania funkcji SysTick_Config() w pliku main.c. **[ToDo 6.6]**

6.3.8 Zbuduj aplikację. Program powinien działać dokładnie tak samo jak poprzednio z tą różnica, że teraz przerwania od PIT zastąpiły przerwania od SysTick (tego jednak nie można stwierdzić obserwując działający program).

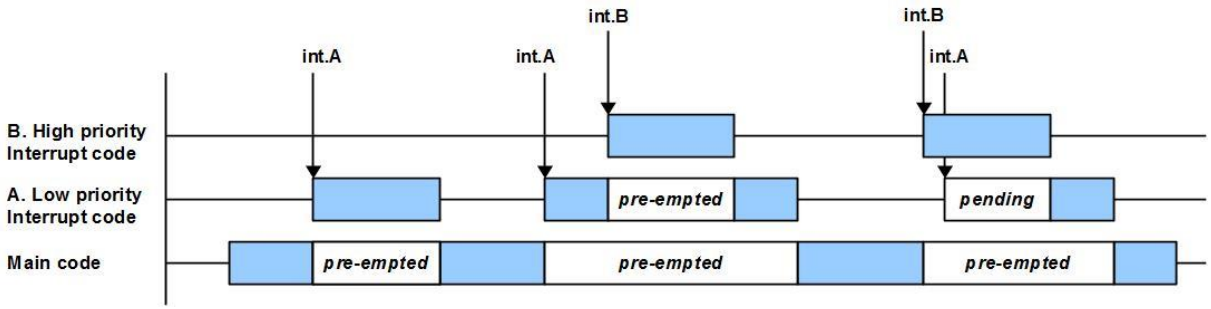
7 Priorytety przerwań

Każde przerwanie w systemie Cortex-M0 ma zdefiniowany przez programistę priorytet. Wysokość priorytetu decyduje o tym, czy zgłoszone przerwanie będzie obsłużone natychmiast, kiedy obsługiwane jest inne przerwanie, czy będzie trwało w stanie oczekiwania (ang. pending state). Dla urządzeń peryferyjnych procesory Cortex-M0 wspierają cztery programowalne poziomy priorytetów: 0, 1, 2 i 3. Im niższa wartość tym wyższy priorytet przerwania (urządzenia z poziomem zero mają największy priorytet). Dodatkowo wyjątki takie jak Reset, Non-Maskable Interrupt (NMI) i Hard-Fault mają dodatkowo przypisane poziomy przerwań odpowiednio -3, -2 i -1. Priorytety dla wyjątków nie są programowalne i mają wyższy poziom od przerwań.

7.1 Kolejność obsługi przerwań

Jeżeli wyjątek lub przerwanie zostanie zgłoszone, kiedy żaden inny wyjątek lub przerwanie nie jest obsługiwany wtedy funkcja obsługi ISR jest wywoływana natychmiast. Proces przełączenia procesora z aktualnie wykonywanego zadania do ISR nazywamy wywłaszczeniem (ang. preemption). Jeżeli procesor aktualnie wykonuje procedurę ISR jednego z przerwań, a w tym czasie zostanie zgłoszone nowe przerwanie z wyższym priorytetem, to znowu dojdzie do wywłaszczenia. Nowo przychodzące przerwanie będzie obsługiwane, a poprzednie wstrzymane (w stanie oczekiwania). Taka sytuacja jest określana jako zagnieżdżenie przerwań. Kiedy ISR nowego przerwania zostanie zakończone, realizacja poprzedniego ISR jest kontynuowana, a powrót do głównego wątku nastąpi po jego zakończeniu.

Kiedy jednak nowo przychodzące przerwanie ma niższy lub ten sam priorytet od aktualnie wykonywanego ISR, to musi czekać w stanie oczekiwania (pending), aż priorytet aktualnie wykonywanego kodu programu zostanie odpowiednio obniżony (Rys. 4).



Rysunek 4: Wywłaszczanie wątków procesora

7.2 Ćwiczenie

7.2.1 Zmodyfikuj ISR w taki sposób, aby CPU zatrzymało się w ISR dla PORTB tak długo jak długo przycisk SW1 jest trzymany wciśnięty **[ToDo 7.1]**

7.2.2 Ustaw priorytet przerwań dla PORTB i PIT na taki sam poziom priorytetu obsługi przerwań (np. najniższy poziom 3) **[ToDo 7.2a] [ToDo 7.2b]**

7.2.3 Zbuduj i uruchom aplikację. Wyjaśnij zachowanie diody RGB LED, kiedy SW1 jest trzymany wciśnięty.

Dioda RGB LED przestały migać, ponieważ .....

7.2.4 Zwiększ poziom priorytetu przerwania PIT (do poziomu 2) [ToDo 7.3]. Znów zbuduj aplikacje i zaprogramuj płytkę. Wyjaśnij zachowanie diody RGB LED, kiedy SW1 jest trzymany. **[ToDo 7.3]**

Dioda RGB LED miga cały czas, ponieważ .....

8 Zadanie dodatkowe

Stosując styl programowania na przerwaniach, dodaj do aplikacji przygotowanej w ćwiczeniu funkcjonalność dla przycisku SW9 (PTB3). Rozszerz funkcje programu tak, aby wciskanie przycisku SW9 zmieniało sposób zachowania diody RGB LED. Równocześnie przyciski SW1 i SW5 cały czas powinny spełniać wcześniejszą funkcjonalność.

Podpowiedź: Należy testować bity w rejestrze ISFR, aby rozróżnić przerwania od przycisków: SW1, SW5 i SW9.

Zadania dla grup (jedno zadanie na grupę).

Grupa 1	Zaprogramuj przycisk SW9, tak aby zmienić sekwencję włączania się diody RGB LED. Użyj dwóch różnych sekwencji. Na przykład: seq_1 = (RED_ON, GREEN_ON, BLUE_OFF, ALL_ON), seq_2 = (RED_ON, ALL_ON, GREEN_ON, ALL_OFF, BLUE_ON, ALL_OFF)
Grupa 2	Zaprogramuj przycisk SW9, tak aby rozpocząć/zatrzymać sekwencję migania diody LED.
Grupa 3	Zaprogramuj przycisk SW9, tak aby zresetować bieżącą sekwencję migania diody. Po naciśnięciu SW9 sekwencja przechodzi do początkowego stanu diody.
Grupa 4	Zaprogramuj przycisk SW9, aby zamrozić na trzy sekundy sekwencję migania diody LED.
Grupa 5	Zaprogramuj przycisk SW9, aby odwrócić sekwencję migania diody LED. Po naciśnięciu SW9 sekwencja zaczyna przebiegać w przeciwnym kierunku.
Grupa 6	Zaprogramuj przycisk SW9, aby zaimplementować opóźnione zatrzymanie uruchomionej sekwencji diod LED. Po naciśnięciu SW9 sekwencja przechodzi do stanu początkowego i zatrzymuje się. Zaczyna się ponownie, gdy SW9 zostaje wciśnięty ponownie.
Grupa 7	Zaprogramuj przycisk SW9, aby przełączać sekwencję diod LED między trybem normalnym, zatrzymania i migania w odwróconej kolejności.