Akademia Górniczo-Hutnicza
w Krakowie
Instytut Elektroniki
WIET

Laboratorium
Technika Mikroprocesorowa 2

Ćwiczenie 3

Kontrola czasu i liczby zdarzeń
Przerwania

Autor: Mariusz Sokołowski

wer. 28.09.2021

1. WSTĘP

1.1. CEL

Celem ćwiczenia jest:

- zapoznanie się z mechanizmem przerwań w systemach wbudowanych,
- nabycie umiejętności w pisaniu programów obsługujących przerwania,
- konfiguracja i wykorzystanie klawiatury jako urządzenia zewnętrznego, pozwalającego użytkownikowi na asynchroniczne (losowe) zgłaszanie zewnętrznych przerwań,
- konfiguracja i wykorzystanie licznika systemowego SysTick do generacji interwałów czasowych, synchronizowanych z zegarem systemowym,
- napisanie programu zliczającego zdarzenia zewnętrzne (ilość wciśnięć danego klawisza),
- napisanie programu odmierzającego czas – stoper.

1.2. WYMAGANIA

Sprzętowe:

- komputer klasy PC, spełniający wymagania sprzętowe aplikacji KEIL v5,
- zestaw FRDMKL05Z

Programowe:

- system operacyjny Windows 7 lub wyższy (wszystkie instrukcje powstały w oparciu o Windows 7 Pro x64),
- środowisko Keil / uVision 5 MDK-ARM

Doświadczenie:

- podstawowa umiejętność obsługi komputera klasy PC,
- podstawowa znajomość systemów operacyjnych rodziny Windows,
- umiejętność programowania w języku C.

Literatura:

- KL05 Sub-Family Reference Manual, Freescale Semiconductor
- Kinetis L Peripheral Module Quick Reference, Freescale Semiconductor
- Joseph Yiu, The Definitive Guide to the ARM Cortex-M0, Elsevier, 2011
- ARM „Cortex-M0+ Devices Generic User Guide”

Wszystko wokół nas dzieje się w czasie. Na każdym kroku spotykamy się z okresowością zdarzeń. Pory roku, tygodnie, wypłata, itd. Nawet te zajęcia odbywają co pewien interwał czasowy, w pewnym czasie, mające swój okres (poza przerwami). Nawet te z pozoru chaotycznie ustawione harmonogramy są budowane w oparciu o ściśle trwające interwały czasowe. Również, aby zaplanować pojedyncze zdarzenie, musimy mu wyznaczyć dokładny odcinek czasu, od którego ma się zacząć i jak długo ma trwać. Wszystkie nowoczesne urządzenia elektroniczne działają w oparciu o jakiś zegar, wybijający jakiś rytm, w oparciu o który planowane są zdarzenia.

Równocześnie ze zdarzeniami okresowymi czy tymi pojedynczymi lub nieokresowymi, ale planowanymi, pojawiają się te nagłe, nieprzewidziane lub specjalnie wymuszone poza kolejnością i poza narzuconym planem.

Oprócz czasu, który zajmuje dane zdarzenie, może być także interesująca liczba samych zdarzeń.

Mikrokontroler MKL05Z32VLC4 jest wyposażony w szereg urządzeń pozwalających m.in. na:

- ✓ pomiar czasu,
- ✓ pomiar i generację interwałów czasowych,
- ✓ pomiar ilości zdarzeń tak wewnętrznych jak i zewnętrznych,
- ✓ reakcję na zdarzenia losowe i pomiar ich liczby,
- ✓ wyzwalanie pojedyncze lub okresowe urządzeń WE/WY, w ściśle określonym czasie.

Dziedziną czasu i zliczaniem ilości zdarzeń zajmują się liczniki. Natomiast reakcją na zdarzenia okresowe czy losowe rządzi system przerwań, który pozwala je również zliczać. Końcowy efekt zależy jednak od oprogramowania, napisanego przez programistę.

2. SYSTEM PRZERWAŃ

Źródłem przerwań w systemie KL05Z mogą być zdarzenia:

- ➢ zewnętrzne, podawane w formie impulsów (np. z klawiatury) na zewnętrzną końcówkę, która pracuje jako wejście GPIO (nie dotyczy tzw. przerwania niemaskowalnego NMI),
- ➢ zewnętrzne, pochodzące od urządzeń WE/WY, będących na wyposażeniu mikrokontrolera,
- ➢ wewnętrzne, systemowe (zwane również wyjątkami), generowane przez urządzenia jądra Cortex-M0+, np. błąd wewnętrzny systemu (np. błąd magistrali).

Oprócz NMI, wszystkie przerwania zewnętrzne są maskowalne, czyli odblokowywane na żądanie. Domyślnie, po sygnale RESET, wszystkie przerwania maskowalne są zablokowane. Ponieważ przerwania nie mogą ze sobą „walczyć" o dostęp do procesora, więc głównym ich nadzorcą jest moduł NVIC (ang. **N**ested **V**ectored **I**nterrupt **C**ontroller), będący częścią jądra procesora Cortex-M0+. Zarządza on włączaniem i wyłączaniem przerwań (ich maskowaniem) oraz kolejnością „dziobania", czyli priorytetami poszczególnych przerwań. Są dostępne cztery poziomy priorytetów: 0 (najwyższy), 1, 2 i 3 (najniższy).

Obsługą przerwań zajmują się podprogramy obsługi przerwań ISR (ang. **I**nterrupt **S**ervice **R**outine), których adresy są umieszczone na samym początku pamięci programu, w ściśle

określonej kolejności (tzw. wektory przerwań). Wektor przerwania to po prostu adres do podprogramu, który obsługuje dane przerwanie. Aby nie operować koszmarnymi, 32-bitowymi adresami, wektorom przypisano numery od 0 do 31, a numerom nazwy, które mają się nam kojarzyć z urządzeniem (źródłem), którego dany numer dotyczy. Numery przerwań, wraz z nazwami są zdefiniowane w znanym nam już zbiorze *MKL05Z4.h*. Numerów tych używa się do odblokowywania wybranych przerwań w module NVIC. Programując w języku C/C++ będziemy wykorzystywać bardziej przyjazne nazwy.

```
/** Interrupt Number Definitions */
typedef enum IRQn {
  /* Core interrupts */
  NonMaskableInt_IRQn          = -14,          /**< Non Maskable Interrupt */
  HardFault_IRQn               = -13,          /**< Cortex-M0 SV Hard Fault Interrupt */
  SVCall_IRQn                  = -5,           /**< Cortex-M0 SV Call Interrupt */
  PendSV_IRQn                  = -2,           /**< Cortex-M0 Pend SV Interrupt */
  SysTick_IRQn                 = -1,           /**< Cortex-M0 System Tick Interrupt */

  /* Device specific interrupts */
  DMA0_IRQn                    = 0,            /**< DMA channel 0 transfer complete/error interrupt */
  DMA1_IRQn                    = 1,            /**< DMA channel 1 transfer complete/error interrupt */
  DMA2_IRQn                    = 2,            /**< DMA channel 2 transfer complete/error interrupt */
  DMA3_IRQn                    = 3,            /**< DMA channel 3 transfer complete/error interrupt */
  Reserved20_IRQn              = 4,            /**< Reserved interrupt 20 */
  FTFA_IRQn                    = 5,            /**< FTFA command complete/read collision interrupt */
  LVD_LVW_IRQn                 = 6,            /**< Low Voltage Detect, Low Voltage Warning */
  LLW_IRQn                     = 7,            /**< Low Leakage Wakeup */
  I2C0_IRQn                    = 8,            /**< I2C0 interrupt */
  Reserved25_IRQn              = 9,            /**< Reserved interrupt 25 */
  SPI0_IRQn                    = 10,           /**< SPI0 interrupt */
  Reserved27_IRQn              = 11,           /**< Reserved interrupt 27 */
  UART0_IRQn                   = 12,           /**< UART0 status/error interrupt */
  Reserved29_IRQn              = 13,           /**< Reserved interrupt 29 */
  Reserved30_IRQn              = 14,           /**< Reserved interrupt 30 */
  ADC0_IRQn                    = 15,           /**< ADC0 interrupt */
  CMP0_IRQn                    = 16,           /**< CMP0 interrupt */
  TPM0_IRQn                    = 17,           /**< TPM0 fault, overflow and channels interrupt */
  TPM1_IRQn                    = 18,           /**< TPM1 fault, overflow and channels interrupt */
  Reserved35_IRQn              = 19,           /**< Reserved interrupt 35 */
  RTC_IRQn                     = 20,           /**< RTC interrupt */
  RTC_Seconds_IRQn             = 21,           /**< RTC seconds interrupt */
  PIT_IRQn                     = 22,           /**< PIT timer interrupt */
  Reserved39_IRQn              = 23,           /**< Reserved interrupt 39 */
  Reserved40_IRQn              = 24,           /**< Reserved interrupt 40 */
  DAC0_IRQn                    = 25,           /**< DAC0 interrupt */
  TSI0_IRQn                    = 26,           /**< TSI0 interrupt */
  MCG_IRQn                     = 27,           /**< MCG interrupt */
  LPTimer_IRQn                 = 28,           /**< LPTimer interrupt */
  Reserved45_IRQn              = 29,           /**< Reserved interrupt 45 */
  PORTA_IRQn                   = 30,           /**< Port A interrupt */
  PORTB_IRQn                   = 31            /**< Port B interrupt */
} IRQn_Type;
```

Bezpośrednio z nazwą (numerem) wektora jest związany tzw. handler, czyli etykieta podprogramu obsługującego konkretne przerwanie. Nazwa etykiety kojarzy się z nazwą przerwania, którego dotyczy. W projekcie, który wykorzystuje przerwania, podprogramy obsługujące ISR mają postać funkcji, których nazwy muszą być identyczne z nazwami handlerów. Nazwy handlerów są dostępne w zbiorze *startup_MKL05Z4.s*.

Funkcje te są wywoływane sprzętowo. Nie ma do nich odwołania w programie. Po przyjściu sygnału przerywającego, wykonywanie pętli głównej jest przerywane i wywoływana jest funkcja handlera. Po jej zakończeniu, wykonywanie pętli głównej jest kontynuowane od miejsca, w którym została przerwana.

```
DMA0_IRQHandler
DMA1_IRQHandler
DMA2_IRQHandler
DMA3_IRQHandler
Reserved20_IRQHandler
FTFA_IRQHandler
LVD_LVW_IRQHandler
LLW_IRQHandler
I2C0_IRQHandler
Reserved25_IRQHandler
SPI0_IRQHandler
Reserved27_IRQHandler
UART0_IRQHandler
Reserved29_IRQHandler
Reserved30_IRQHandler
ADC0_IRQHandler
CMP0_IRQHandler
TPM0_IRQHandler
TPM1_IRQHandler
Reserved35_IRQHandler
RTC_IRQHandler
RTC_Seconds_IRQHandler
PIT_IRQHandler
Reserved39_IRQHandler
Reserved40_IRQHandler
DAC0_IRQHandler
TSI0_IRQHandler
MCG_IRQHandler
LPTimer_IRQHandler
Reserved45_IRQHandler
PORTA_IRQHandler
PORTB_IRQHandler
DefaultISR
```

```
; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler\
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
                ENDP
SVC_Handler     PROC
                EXPORT  SVC_Handler               [WEAK]
                B       .
                ENDP
PendSV_Handler  PROC
                EXPORT  PendSV_Handler            [WEAK]
                B       .
                ENDP
SysTick_Handler PROC
                EXPORT  SysTick_Handler           [WEAK]
                B       .
                ENDP
```

W każdym urządzeniu, które ma możliwość zgłaszania przerwań, można, w specjalnym rejestrze, odblokować lub zablokować przerwanie. Gdy w urządzeniu zostanie spełniony warunek generacji sygnału przerwania, zostanie ono podane na odpowiednie wejście bloku NVIC, i jeśli tylko jest ono odblokowane, procesor przerywa aktualnie wykonywany program (zgodnie z priorytetem przerwania) i wykonuje funkcje handlera. Funkcje, ułatwiające komunikację z modułem NVIC, są zdefiniowane w zbiorze *core_cm0plus.h*.

```
/* ##########################   NVIC functions  #################################### */
/**
  \ingroup  CMSIS_Core_FunctionInterface
  \defgroup CMSIS_Core_NVICFunctions NVIC Functions
  \brief    Functions that manage interrupts and exceptions via the NVIC.
  @{
 */

#ifdef CMSIS_NVIC_VIRTUAL
  #ifndef CMSIS_NVIC_VIRTUAL_HEADER_FILE
    #define CMSIS_NVIC_VIRTUAL_HEADER_FILE "cmsis_nvic_virtual.h"
  #endif
  #include CMSIS_NVIC_VIRTUAL_HEADER_FILE
#else
  #define NVIC_SetPriorityGrouping    __NVIC_SetPriorityGrouping
  #define NVIC_GetPriorityGrouping    __NVIC_GetPriorityGrouping
  #define NVIC_EnableIRQ              __NVIC_EnableIRQ
  #define NVIC_GetEnableIRQ           __NVIC_GetEnableIRQ
  #define NVIC_DisableIRQ             __NVIC_DisableIRQ
  #define NVIC_GetPendingIRQ          __NVIC_GetPendingIRQ
  #define NVIC_SetPendingIRQ          __NVIC_SetPendingIRQ
  #define NVIC_ClearPendingIRQ        __NVIC_ClearPendingIRQ
/*#define NVIC_GetActive              __NVIC_GetActive             not available for Cortex-M0+ */
  #define NVIC_SetPriority            __NVIC_SetPriority
  #define NVIC_GetPriority            __NVIC_GetPriority
  #define NVIC_SystemReset            __NVIC_SystemReset
#endif /* CMSIS_NVIC_VIRTUAL */
```

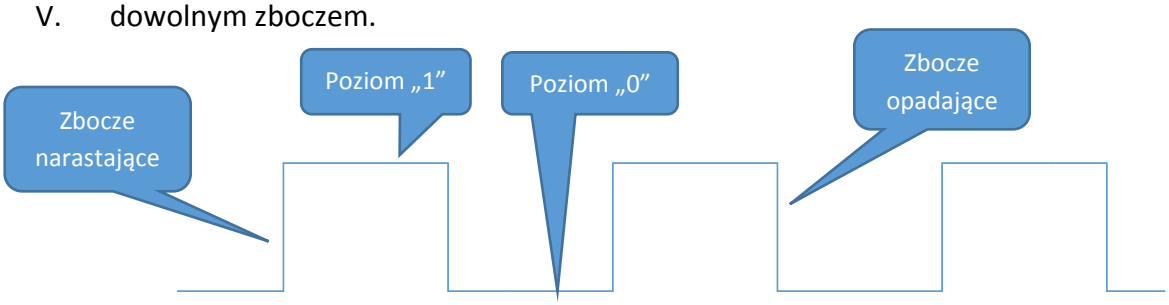
Wykorzystywanymi funkcjami na laboratorium będą:

- ***NVIC_ClearPendingIRQ(IRQn_Type IRQn)*** – funkcja kasująca ewentualne, niechciane (nieprzewidziane) przerwanie o numerze (nazwie) „IRQn”
  Np. *NVIC_ClearPendingIRQ(PORTA_IRQn)*
- ***NVIC_EnableIRQ(IRQn_Type IRQn)*** – odblokowanie przerwania o numerze (nazwie) „IRQn”
  Np. *NVIC_EnableIRQ(PORTA_IRQn)*
- ***NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority****)* – ustawianie priorytetu (o numerze *„priority”)* przerwania o numerze (nazwie) „IRQn”
  Np. *NVIC_SetPriority(PORTA_IRQn, 0)* – ustawienie najwyższego priorytetu

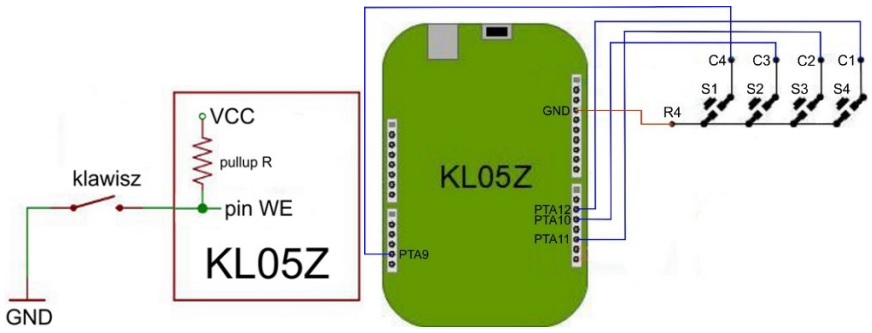
3. PRZERWANIA OD PINÓW ZEWNĘTRZNYCH

Do pinów GPIO, które mają wbudowaną funkcję zgłaszania przerwania, można podpiąć dowolne źródło sygnału cyfrowego, który będzie wywoływał funkcję przerywającą (będzie przerywał wykonywanie programu). Istnieje pięć możliwości zgłaszania przerwania na pinie:

I. poziomem „1”,
II. poziomem „0”,
III. zboczem narastającym,
IV. zboczem opadającym,
V. dowolnym zboczem.



Rozpatrzmy przypadek z klawiszem.
W momencie, gdy klawisz jest niewciśnięty, na pinie jest stan „1”. Więc punkt I odpada.
W momencie, gdy wciskamy klawisz, generujemy zbocze opadające, jak również poziom „0”. Punkty II i IV OK.
W momencie, gdy zwalniamy klawisz, generujemy zbocze narastające. Punkt II OK.
Zazwyczaj chcemy, aby wciśnięcie klawisza było zaliczone tylko raz, więc punkt V wątpliwy.
Jeśli stosujemy wyzwalanie zboczem, to nie ma możliwości tzw. powielania klawisza, czyli wielokrotnego zaliczenia wciśnięcia.
Biorąc pod uwagę drgania zestyków, najkorzystniejszym rozwiązaniem będą zbocza.



3.1. PROGRAMOWANIE PRZERWANIA OD PINÓW

Przygotowując dany port i jego piny do pracy, można jednocześnie włączyć funkcję obsługi przychodzących przerwań, ale tylko dla pinów, które takową funkcję mają zaimplementowaną. Najpierw w skróconej tabeli funkcji pinów sprawdzamy, które piny posiadają w kolumnie ALT1 napis IRQ_n przy nazwie pinu. Tylko te piny wchodzą w grę. Np. PTA10, PTA11 i PTA12 mają, a PTA9 nie ma.

KL05Z (32QFx)	Pin Name	On Board	IO header	DEFAULT	ALT0	ALT1	ALT2	ALT3
1	PTB6/IRQ_2/LPTMR0_ALT3		D6	DISABLED		PTB6/IRQ_2/LPTMR0_ALT3	TPM0_CH3	TPM_CLKIN1
2	PTB7/IRQ_3		D7	DISABLED		PTB7/IRQ_3	TPM0_CH2	
3	VDD	VDD	3V3	VDD	VDD			
4	VREFH	VREFH	AREF	VREFH	VREFH			
5	VREFL	VREFL	—	VREFL	VREFL			
6	VSS	VSS	GND	VSS	VSS			
7	PTA3	32 KHz XTAL	—	EXTAL0	EXTAL0	PTA3	I2C0_SCL	I2C0_SDA
8	PTA4/LLWU_P0	32 KHz XTAL	—	XTAL0	XTAL0	PTA4/LLWU_P0	I2C0_SDA	I2C0_SCL
9	PTA5/LLWU_P1/RTC_CLK_IN		D10	DISABLED		PTA5/LLWU_P1/RTC_CLK_IN	TPM0_CH5	SPI0_SS_b
10	PTA6/LLWU_P2		D12	DISABLED		PTA6/LLWU_P2	TPM0_CH4	SPI0_MISO
11	PTB8	Red LED	A0	ADC0_SE11	ADC0_SE11	PTB8	TPM0_CH3	
12	PTB9	Green LED	A1	ADC0_SE10	ADC0_SE10	PTB9	TPM0_CH2	
13	PTB10	Blue LED	D8	ADC0_SE9/TSI0_IN7	ADC0_SE9/TSI0_IN7	PTB10	TPM0_CH1	
14	PTB11		D9	ADC0_SE8/TSI0_IN6	ADC0_SE8/TSI0_IN6	PTB11	TPM0_CH0	
15	PTA7/IRQ_7/LLWU_P3		D11	ADC0_SE7/TSI0_IN5	ADC0_SE7/TSI0_IN5	PTA7/IRQ_7/LLWU_P3	SPI0_MISO	SPI0_MOSI
16	PTB0/IRQ_8/LLWU_P4		D13	ADC0_SE6/TSI0_IN4	ADC0_SE6/TSI0_IN4	PTB0/IRQ_8/LLWU_P4	EXTRG_IN	SPI0_SCK
17	PTB1/IRQ_9		D1	ADC0_SE5/TSI0_IN3/DAC0_OUT/CMP0_IN3	ADC0_SE5/TSI0_IN3/DAC0_OUT/CMP0_IN3	PTB1/IRQ_9	UART0_TX	UART0_RX
18	PTB2/IRQ_10/LLWU_P5		D0	ADC0_SE4/TSI0_IN2	ADC0_SE4/TSI0_IN2	PTB2/IRQ_10/LLWU_P5	UART0_RX	UART0_TX
19	PTA8		A2	ADC0_SE3/TSI0_IN1	ADC0_SE3/TSI0_IN1	PTA8		
20	PTA9		A4	ADC0_SE2/TSI0_IN0	ADC0_SE2/TSI0_IN0	PTA9		
21	PTA10/IRQ_12	Accel INT2	D4	DISABLED	TSI0_IN11	PTA10/IRQ_12		
22	PTA11/IRQ_13		D2	DISABLED	TSI0_IN10	PTA11/IRQ_13		
23	PTB3/IRQ_14	Accel I2C	D15	DISABLED		PTB3/IRQ_14	I2C0_SCL	UART0_TX
24	PTB4/IRQ_15/LLWU_P6	Accel I2C	D14	DISABLED		PTB4/IRQ_15/LLWU_P6	I2C0_SDA	UART0_RX
25	● PTB5/IRQ_16		D3	● NMI_b	ADC0_SE1/CMP0_IN1	PTB5/IRQ_16	TPM1_CH1	NMI_b
26	PTA12/IRQ_17		D5	ADC0_SE0/CMP0_IN0	ADC0_SE0/CMP0_IN0	PTA12/IRQ_17/LPTMR0_ALT2	TPM1_CH0	TPM_CLKIN0
27	PTA13	Touch Slider	—	TSI0_IN9	TSI0_IN9	PTA13		
28	PTB12	Touch Slider	—	TSI0_IN8	TSI0_IN8	PTB12		
29	PTB13		A5	ADC0_SE13	ADC0_SE13	PTB13	TPM1_CH1	RTC_CLKOUT
30	PTA0/IRQ_0/LLWU_P7	SWD_CLK	A3	SWD_CLK	ADC0_SE12/CMP0_IN2	PTA0/IRQ_0/LLWU_P7	TPM1_CH0	SWD_CLK
31	PTA1/IRQ_1/LPTMR0_ALT1	RESET	RESET	RESET_b		PTA1/IRQ_1/LPTMR0_ALT1	TPM_CLKIN0	RESET_b
32	PTA2	SWD_DIO	—	SWD_DIO		PTA2	CMP0_OUT	SWD_DIO

Na etapie programowania funkcji pinu, w rejestrze PCRn, bloku PORTx, można określić sposób reakcji na sygnał przerywający, który może się pojawić na danym pinie (dotyczy tylko pinów pracujących jako wejście). „x” oznacza literę odpowiedniego portu: A lub B. „n” oznacza numer pinu (bitu) portu „x”. Każda końcówka (pin) jest konfigurowana osobnym rejestrem PCR.

Pin Control Register n (PORTx_PCR*n*)

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0						MUX		0	DSE	0	PFE	0		PE	PS
W																
Reset	0	0	0	0	*	*	*	*	0	*	0	*	*	*	*	*

bit stanu przerwania:
0 – przerwania brak
1 – wykryto przerwanie

0000 przerwania zablokowane
1000 aktywny poziom „0”
1001 aktywne zbocze narastające
1010 aktywne zbocze opadające
1011 aktywne dowolne zbocze
1100 aktywny poziom „1”

Na końcu bloku konfigurującego piny należy dodać instrukcje:
NVIC_ClearPendingIRQ(PORTA_IRQn)
NVIC_EnableIRQ(PORTA_IRQn)
NVIC_SetPriority(PORTA_IRQn, 0) – jeśli potrzeba ustawiać priorytet (domyślnie wszystkie 0)

Przerwaniami każdego portu (A lub B) rządzi osobny, ale tylko jeden wektor (handler). Teoretycznie, jeden 32-bitowy port może być źródłem 32-u przerwań, które są obsługiwane prze jeden handler. Jak rozróżnić, który pin zgłosił przerwanie? W podprogramie obsługi przerwania, pierwszą czynnością jest ustalenie, który pin (piny) jest (są) odpowiedzialny(e) za zgłoszenie. Można to zrobić na dwa sposoby.

I. Odczytać stan bitu ISF w każdym rejestrze PCRn (odpytywanie) i na tej podstawie zdecydować o dalszym postępowaniu.

II. Odczytać rejestr ISFR, w bloku PORTx (ten sam co dla PCRn), który zawiera stany wszystkich bitów ISF, wszystkich 32-u rejestrów PCR i na tej podstawie zdecydować o dalszym postępowaniu.

W przypadku wielu zgłoszeń naraz, należy wymyślić sobie i zastosować wewnętrzny system priorytetów dla poszczególnych pinów. Po podjęciu decyzji, które przerwanie jest ważne, należy odpowiedni bit ISF wyzerować, aby nie był znowu źródłem „fałszywego" przerwania. Dokonuje się tego poprzez wpisanie wartości „1" do odpowiedniej pozycji ISF. Oczywiście, jeśli pozostałe, ewentualne przerwania, chcemy zignorować, to również i je należy wyzerować w podobny sposób.

Interrupt Status Flag Register (PORTx_ISFR)

Address: Base address + A0h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ISF																															
W	w1c																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Zestaw instrukcji programujących możliwość zgłaszania przerwania przez klawisze S2, S3 i S4 wygląda następująco:

PORTA -> PCR[S2] |= PORT_PCR_IRQC(0xa);

PORTA -> PCR[S3] |= PORT_PCR_IRQC(0xa);

PORTA -> PCR[S4] |= PORT_PCR_IRQC(0xa);

NVIC_ClearPendingIRQ(PORTA_IRQn);

NVIC_EnableIRQ(PORTA_IRQn);

Wartość **0xa** oznacza mod pracy z reakcją na opadające zbocze.

4. LICZNIK SYSTEMOWY SYSTICK

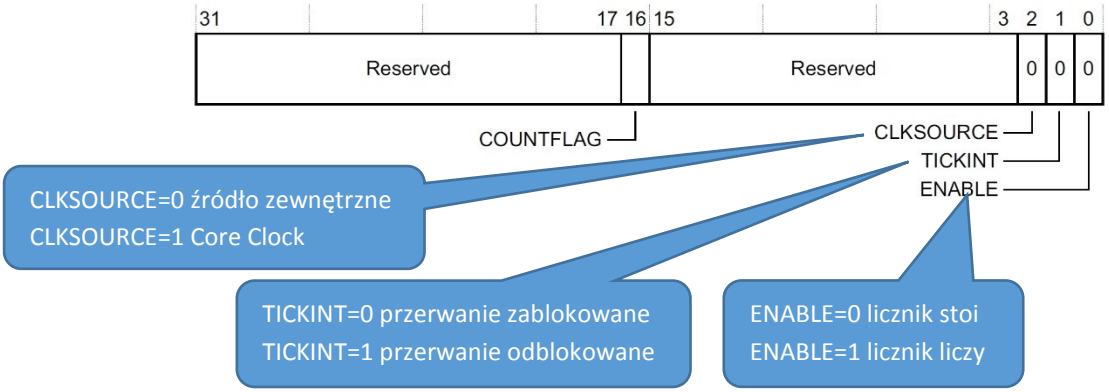
Jest on elementem należącym do rdzenia procesora Cortex-M0+. Dlatego jego opisu nie znajdziemy w instrukcji obsługi modułu KL05Z. Należy otworzyć „*ARM®v6-M Architecture Reference Manual*" lub „*Cortex™-M0+ Devices. Generic User Guide*". Licznik jest 24-bitowy i zlicza w tył, odejmując 1, za każdym „tyknięciem" zegara Core Clock. Pod dojściu do zera, do rejestru licznika jest przeładowywana wartość startowa i wszystko zaczyna się od początku. W rejestrze kontrolującym pracę licznika interesują nas trzy bity:

- ENABLE, który służy do uruchamiania i zatrzymywania zliczania,
- TICKINT, który odblokowuje możliwość zgłaszania przerwań przez licznik SysTick,
- CLKSOURCE, wybierający źródło dla zegara wejściowego, którego impulsy są zliczane.

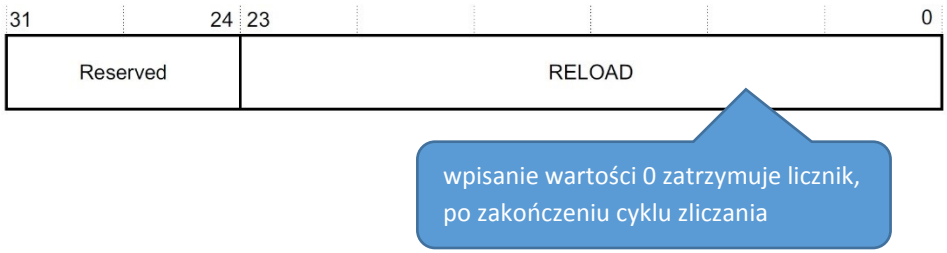
Aby zliczyć n impulsów zegarowych, do rejestru, zawierającego wartość startową, należy wpisać liczbę n-1. Przykładowo, jeśli chcemy zliczyć 100 impulsów, wartość ta musi wynosić 99. Należy jednak pamiętać, że rejestr licznika, który zlicza impulsy zegarowe, jest 24-bitowy. Wynika z tego, że maksymalna liczba zliczonych impulsów to 2^{24}, czyli 16777216 (0x1000000), czego efektem będzie wpisanie wartości startowej równej 16777215 (0xFFFFFF). Warto o tym pamiętać, w sytuacji, gdy jesteśmy przyzwyczajeni do rejestrów 32-bitowych. Poniżej jest zestawienie w/w rejestrów, odpowiedzialnych za pracę licznika SysTick. W nawiasach są nazwy, stosowane podczas programowania w środowisku Keil.

Praktycznie, licznik SysTick jest stworzony do zgłaszania periodycznego przerwania, dlatego używanie go bez jego odblokowania jest bez sensu.

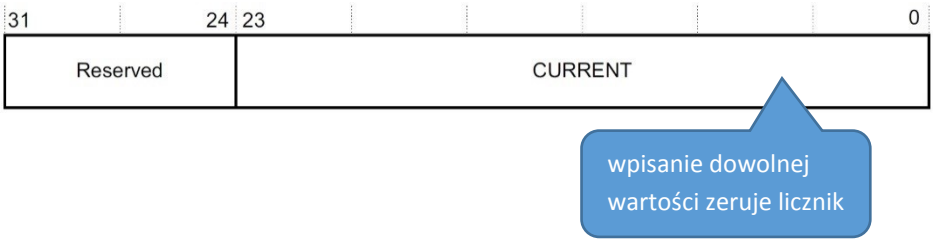
Rejestr SYST_CSR (CTRL), kontrolujący pracę SysTick'a



Rejestr SYST_RVR (LOAD), z wartością startową



Rejestr SYST_CVR (VAL), wartość bieżąca licznika



Kolejność programowania licznika SysTick:

1) Wpisanie wartości startowej do rejestru SYST_RVR (LOAD),
2) Wyzerowanie zawartości rejestru SYST_CVR(VAL),
3) Ustawienie bitów rejestru SYST_CSR (CTRL), zgodnie z wymaganiami programisty

Ponieważ nie ma zewnętrznego źródła taktującego, więc biorąc pod uwagę użyteczność licznika, pkt. nr 3) jest zdeterminowany: CLKSOURCE=1, TICKINT=1. Zostaje już tylko decyzja, w którym miejscu programu uruchomić licznik (ENABLE=1).

W zbiorze *core_cm0plus.h* jest (podobnie jak dla NVIC) jest zdefiniowana gotowa funkcja programująca licznik SysTick.

```
/**
  \brief   System Tick Configuration
  \details Initializes the System Timer and its interrupt, and starts the System Tick Timer.
           Counter is in free running mode to generate periodic interrupts.
  \param [in]  ticks  Number of ticks between two interrupts.
  \return          0  Function succeeded.
  \return          1  Function failed.
  \note    When the variable <b>__Vendor_SysTickConfig</b> is set to 1, then the
           function <b>SysTick_Config</b> is not included. In this case, the file <b><i>device</i>.h</b>
           must contain a vendor-specific implementation of this function.
 */
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
  {
    return (1UL);                                                   /* Reload value impossible */
  }

  SysTick->LOAD  = (uint32_t)(ticks - 1UL);                         /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL); /* set Priority for Systick Interrupt */
  SysTick->VAL   = 0UL;                                             /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;                         /* Enable SysTick IRQ and SysTick Timer */
  return (0UL);                                                     /* Function successful */
}
```

Wykorzystanie powyższej funkcji polega tylko na podaniu liczby impulsów zegarowych, które ma zliczyć licznik. Funkcja sama odejmie liczbę 1, sprawdzi czy nie przekroczono 24-bitowego zakresu argumentu, załaduje wartość do licznika, odblokuje przerwania i uruchomi licznik. Od tego czasu licznik cały czas zlicza, a po doliczeniu do końca zgłasza przerwanie i liczy dalej, aż go nie zatrzymamy. Aby odliczyć precyzyjnie czas, z dokładnością zegara systemowego (jądra – Core Clock), należy używać zmiennej systemowej SystemCoreClock, która jest zadeklarowana i zainicjowana w zbiorze *system_MKL05Z4.c*.

```
#define CLOCK_SETUP     0
/* Predefined clock setups
   0 ... Multipurpose Clock Generator (MCG) in FLL Engaged Internal (FEI) mode
         Reference clock source for MCG module is the slow internal clock source 32.768kHz
         Core clock = 41.94MHz, BusClock = 20.97MHz
   1 ... Multipurpose Clock Generator (MCG) in FLL Engaged External (FEE) mode
         Reference clock source for MCG module is an external crystal 32.768kHz
         Core clock = 47.97MHz, BusClock = 23.98MHz
   2 ... Multipurpose Clock Generator (MCG) in FLL Bypassed Low Power Internal (BLPI) mode
         Core clock/Bus clock derived directly from an fast internal 4MHz clock with no multiplication
         Core clock = 4MHz, BusClock = 4MHz
*/

/*----------------------------------------------------------------------------
  Define clock source values
 *----------------------------------------------------------------------------*/
#if (CLOCK_SETUP == 0)
    #define CPU_XTAL_CLK_HZ                 32768u   /* Value of the external crystal or oscillator clock frequency in Hz */
    #define CPU_INT_SLOW_CLK_HZ             32768u   /* Value of the slow internal oscillator clock frequency in Hz  */
    #define CPU_INT_FAST_CLK_HZ             4000000u /* Value of the fast internal oscillator clock frequency in Hz  */
    #define DEFAULT_SYSTEM_CLOCK            41943040u /* Default System clock value */
#elif (CLOCK_SETUP == 1)
    #define CPU_XTAL_CLK_HZ                 32768u   /* Value of the external crystal or oscillator clock frequency in Hz */
    #define CPU_INT_SLOW_CLK_HZ             32768u   /* Value of the slow internal oscillator clock frequency in Hz  */
    #define CPU_INT_FAST_CLK_HZ             4000000u /* Value of the fast internal oscillator clock frequency in Hz  */
    #define DEFAULT_SYSTEM_CLOCK            47972352u /* Default System clock value */
#elif (CLOCK_SETUP == 2)
    #define CPU_XTAL_CLK_HZ                 32768u /* Value of the external crystal or oscillator clock frequency in Hz */
    #define CPU_INT_SLOW_CLK_HZ             32768u   /* Value of the slow internal oscillator clock frequency in Hz  */
    #define CPU_INT_FAST_CLK_HZ             4000000u /* Value of the fast internal oscillator clock frequency in Hz  */
    #define DEFAULT_SYSTEM_CLOCK            4000000u /* Default System clock value */
#endif /* (CLOCK_SETUP == 2) */


/* ----------------------------------------------------------------------------
   -- Core clock
   ---------------------------------------------------------------------------- */

uint32_t SystemCoreClock = DEFAULT_SYSTEM_CLOCK;
```

Przykład:

Core Clock=4MHz (4 000 000Hz)

Aby zliczyć czas równy 1s, należy załadować do rejestru LOAD SysTick'a liczbę 3 999 999. Jeśli chcemy odliczyć 100ms, to liczbę 399 999, itd.

Wywołanie funkcji będzie następujące:

SysTick_Config(SystemCoreClock) dla 1s

SysTick_Config(SystemCoreClock/10) dla 100ms.

Należy pamiętać, że nasze projekty używają wartości SystemCoreClock=41.94MHz (41 943 040Hz). Jest to liczba za duża (nie mieści się na 24-ech bitach). Dlatego, najmniejszy interwał czasowy, będący pochodną sekundy to 100ms. Aby odmierzyć czas jednej sekundy należy zastosować zliczanie przerwań. Dziesięć przerwań to jedna sekunda. Będzie to wyjaśnione w przykładowym programie.

W celu zatrzymania licznika należy użyć poniższego wywołania w/w funkcji:

SysTick_Config(1).

5. ĆWICZENIE 1

Rozpakować zbiór *1_Klawiatura.zip*. Uruchomić projekt *Klaw_Przerw.uvprojx*. Na początku pojawia się ekran powitalny. Program czeka na wciśnięcie klawisza S1 (jak w przypadku Laboratorium 2). Po jego wciśnięciu, na ekranie pojawiają się dwie pozycje, pokazujące, ile razy został wciśnięty klawisz S3 i S4. Program czeka na trzy zdarzenia:

- wciśnięcie klawisza S2 – Ustawiany jest bit, informujący pętlę główną, że klawisz S2 został wciśnięty. Na podstawie jego stanu, pętla główna, tak jak w Laboratorium 2, steruje diodą czerwoną LED. Różnica jest taka, że klawisz nie jest odpytywany w pętli głównej, tylko obsługiwany na zasadzie przerwania (wciśnięcie wywołuje podprogram *PORTA_IRQHandler(void)*). Kolejne wciskanie tego klawisza gasi, zapala, gasi, zapala, itd.
- wciśnięcie klawisza S3 – Po zredukowaniu drgań zestyków, jest zwiększana wartość licznika wciśnięć klawisza S3 i ustawiany bit, informujący pętlę główną, że klawisz S3 został wciśnięty. Na podstawie jego stanu, pętla główna wyświetla aktualny stan licznika wciśnięć klawisza S3.
- wciśnięcie klawisza S4 – Bez redukcji drgań zestyków, jest zwiększana wartość licznika wciśnięć klawisza S4 i ustawiany bit, informujący pętlę główną, że klawisz S4 został wciśnięty. Na podstawie jego stanu, pętla główna wyświetla aktualny stan licznika wciśnięć klawisza S4.

  Wszystkie piny, do których są podpięte klawisze S2, S3 i S4, zaprogramowano w tryb pracy reakcji na zbocze opadające.

5.1. ZADANIE

- Wyjaśnić, dlaczego w czasie trzymania klawisza S2 w pozycji wciśniętej, dioda nie mruga?
- Zaobserwować, jak zachowuje się klawisz z redukcją drgań zestyków w porównaniu z klawiszem, który tej funkcji nie ma zaprogramowanej. Wciskać raz klawisz S3, raz S4. Gdyby nie było między nimi różnicy, wyniki byłyby takie same. Dlaczego się różnią?

6. ĆWICZENIE 2

Rozpakować zbiór *2_Licznik_SysTick.*zip. Uruchomić projekt *Licznik_SysTick.uvprojx*. Na początku, przez 5 sekund, pojawia się ekran powitalny. Po czym pokazuje się licznik „t", który, jak stoper, liczy sekundy do 99 i następnie zaczyna znowu od wartości 00. Również co sekundę, zmienia swój stan czerwona dioda LED: świeci przez sekundę, a następnie gaśnie na sekundę, itd. Wzorzec jednej sekundy wylicza podprogram, obsługujący przerwanie od licznika SysTick, który co 100ms zgłasza przerwanie. Pętla główna zajmuje się wyświetlaniem wyniku zliczania stopera.

6.1. ZADANIE

Na podstawie wiedzy zdobytej w Ćwiczeniu 1, dopisać następującą funkcjonalność programu:

- ✓ przerwanie od klawisza S2, będzie zatrzymywało zliczanie,
- ✓ przerwanie od klawisza S3, będzie wznawiało zliczanie, od wartości 00.