Programowanie mikrokontrolerów 2.0

DMA, przerwania

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

26 października 2021

DMA

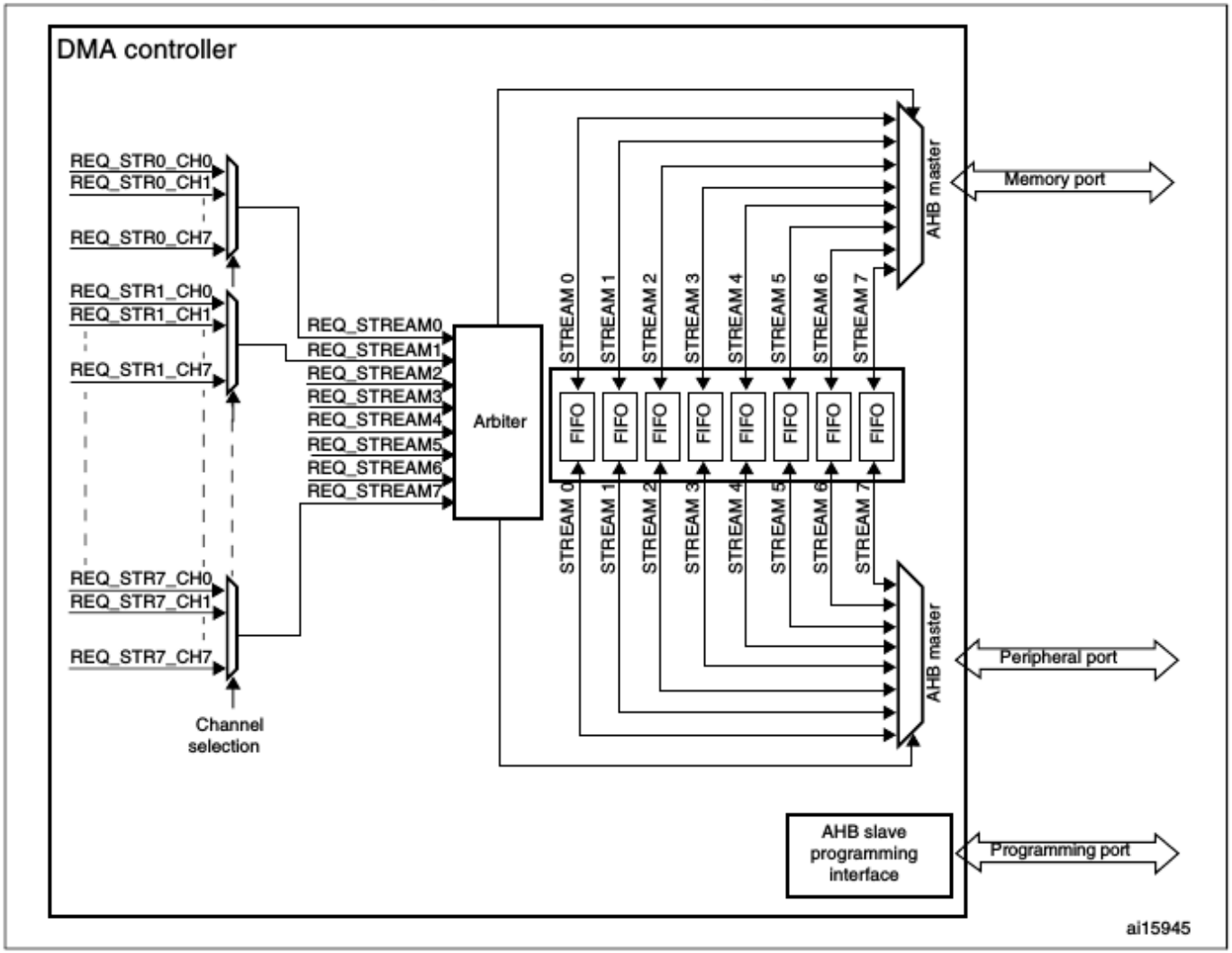
- Akronim Direct Memory Access
- Przesyłanie danych bez angażowania CPU
- W STM32, zależnie od modelu, do 6 układów:
  - DMA1
  - DMA2
  - Ethernet DMA
  - USB OTG HS DMA
  - LCD-TFT DMA
  - DMA2D

DMA ogólnego przeznaczenia w STM32F411

- Układ `DMA1` obsługuje transmisje na szynach AHB1 i APB1
  - z pamięci do układu peryferyjnego
  - z układu peryferyjnego do pamięci
- Układ `DMA2` obsługuje transmisje na szynach AHB1 i APB2
  - z pamięci do układu peryferyjnego
  - z układu peryferyjnego do pamięci
  - z pamięci do pamięci
- Każdy z układów ma 3 interfejsy
  - *memory port*
  - *peripheral port*
  - *programming port*
- Każdy z układów obsługuje 8 niezależnych strumieni
- Dla każdego strumienia można wybrać 1 z 8 kanałów (źródeł żądań transmisji)
- Dla każdego strumienia można ustalić 1 z 4 priorytetów

Schemat blokowy

Figure 22. DMA block diagram



Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Przydział strumieni i kanałów

Table 27. DMA1 request mapping (STM32F411xC/E)

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	I2C1_TX	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX	I2C3_RX				I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4						USART2_RX	USART2_TX	
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	I2C3_TX	TIM5_UP	USART2_RX
Channel 7			I2C2_RX	I2C2_RX				I2C2_TX

Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Przydział strumieni i kanałów

Table 28. DMA2 request mapping (STM32F411xC/E)

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1				ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1								
Channel 2			SPI1_TX	SPI5_RX	SPI5_TX			
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO	SPI4_RX	USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	SPI5_TX	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7						SPI5_RX	SPI5_TX	

Źródło: RM0383 – Reference manual, STM32F411xC/E advanced ARM-based 32-bit MCUs

Transfer danych

- Rozmiary przesyłanych danych
  - bajt – 8 bitów
  - półsłowo – 16 bitów
  - słowo – 32 bity
- Dane mogą być przesyłane w paczkach (ang. *burst*)
  - pojedyncze przesłanie (ang. *single*)
  - 4 przesłania (ang. *beats*) – INCR4
  - 8 przesłań – INCR8
  - 16 przesłań – INCR16
- Przesyłanie bezpośrednie
  - bezpośrednio między *memory port* a *peripheral port*
  - rozmiary danych na obu interfejsach są jednakowe i wyznaczone przez rozmiar ustawiony dla *peripheral port*
  - dozwolone są tylko pojedyncze przesłania
- Kolejkowanie
  - przesłania między *memory port* a *peripheral port* są kolejkowane w FIFO
  - rozmiary danych i paczek na obu interfejsach mogą być różne

Zaawansowane zagadnienia

- Konfigurowanie FIFO
- Cykliczny bufor
- Podwójny bufor
- Obsługa błędów

Prosty przykład

- Skorzystamy z DMA do obsługi wysyłania i odbierania danych za pomocą UART-u
- Jak zwykle najpierw trzeba włączyć taktowanie wykorzystywanych układów

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN |
                RCC_AHB1ENR_DMA1EN;
RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
```

Prosty przykład, konfigurowanie UART-u

- Linia TXD na PA2, RXD na PA3, funkcja alternatywna

```
GPIOafConfigure(GPIOA,
                2,
                GPIO_OType_PP,
                GPIO_Fast_Speed,
                GPIO_PuPd_NOPULL,
                GPIO_AF_USART2);
GPIOafConfigure(GPIOA,
                3,
                GPIO_OType_PP,
                GPIO_Fast_Speed,
                GPIO_PuPd_UP,
                GPIO_AF_USART2);
```

Prosty przykład, konfigurowanie UART-u, cd.

- Parametry transmisji: 8 bitów, bez kontroli parzystości, 1 bit stopu

```
USART2->CR1 = USART_CR1_RE | USART_CR1_TE;
USART2->CR2 = 0;
USART2->BRR = (PCLK1_HZ + (BAUD / 2U)) / BAUD;
```

- Wysyłanie i odbieranie za pomocą DMA

```
USART2->CR3 = USART_CR3_DMAT | USART_CR3_DMAR;
```

Prosty przykład, konfigurowanie nadawczego strumienia DMA

- USART2 TX korzysta ze strumienia 6 i kanału 4, tryb bezpośredni, transfery 8-bitowe, wysoki priorytet, zwiększanie adresu pamięci po każdym przesłaniu, przerwanie po zakończeniu transmisji

```
DMA1_Stream6->CR = 4U << 25        |
                   DMA_SxCR_PL_1   |
                   DMA_SxCR_MINC   |
                   DMA_SxCR_DIR_0  |
                   DMA_SxCR_TCIE;
```

- Adres układu peryferyjnego nie zmienia się

```
DMA1_Stream6->PAR = (uint32_t)&USART2->DR;
```

Prosty przykład, konfigurowanie odbiorczego strumienia DMA

- USART2 RX korzysta ze strumienia 5 i kanału 4, tryb bezpośredni, transfery 8-bitowe, wysoki priorytet, zwiększanie adresu pamięci po każdym przesłaniu, przerwanie po zakończeniu transmisji

```
DMA1_Stream5->CR = 4U << 25        |
                   DMA_SxCR_PL_1   |
                   DMA_SxCR_MINC   |
                   DMA_SxCR_TCIE;
```

- Adres układu peryferyjnego nie zmienia się

```
DMA1_Stream5->PAR = (uint32_t)&USART2->DR;
```

- USART2 RX może też korzystać alternatywnie ze strumienia 7 i kanału 6

Prosty przykład, aktywowanie przerwań DMA, włączenie UART-u

- Wyczyść znaczniki przerwań i włącz przerwania

```
DMA1->HIFCR = DMA_HIFCR_CTCIF6 |
              DMA_HIFCR_CTCIF5;
NVIC_EnableIRQ(DMA1_Stream6_IRQn);
NVIC_EnableIRQ(DMA1_Stream5_IRQn);
```

- Uaktywnij układ peryferyjny

```
USART2->CR1 |= USART_CR1_UE;
```

Prosty przykład, inicjowanie transferu

- Inicjowanie wysyłania

```
DMA1_Stream6->M0AR = (uint32_t)buff;
DMA1_Stream6->NDTR = len;
DMA1_Stream6->CR |= DMA_SxCR_EN;
```

- Inicjowanie odbierania

```
DMA1_Stream5->M0AR = (uint32_t)buff;
DMA1_Stream5->NDTR = 1;
DMA1_Stream5->CR |= DMA_SxCR_EN;
```

Prosty przykład, obsługa przerwania

- Szkielet procedury obsługi przerwania po zakończeniu wysyłania

```
void DMA1_Stream6_IRQHandler() {
  /* Odczytaj zgłoszone przerwania DMA1. */
  uint32_t isr = DMA1->HISR;
  if (isr & DMA_HISR_TCIF6) {
    /* Obsłuż zakończenie transferu
    w strumieniu 6. */
    DMA1->HIFCR = DMA_HIFCR_CTCIF6;
    ...
    /* Jeśli jest coś do wysłania,
    wystartuj kolejną transmisję. */
    ...
  }
}
```

- Uwaga: zakończenie transferu DMA nie oznacza, że UART zakończył wysyłanie

Prosty przykład, obsługa przerwania

- Szkielet procedury obsługi przerwania po zakończeniu odbierania

```
void DMA1_Stream5_IRQHandler() {
  /* Odczytaj zgłoszone przerwania DMA1. */
  uint32_t isr = DMA1->HISR;
  if (isr & DMA_HISR_TCIF5) {
    /* Obsłuż zakończenie transferu
    w strumieniu 5. */
    DMA1->HIFCR = DMA_HIFCR_CTCIF5;
    ...
    /* Ponownie uaktywnij odbieranie. */
    ...
  }
}
```

Problem

- Nie można inicjować wysyłania, gdy strumień DMA jest zajęty
- Bit `EN` w rejestrze `CR` strumienia jest
  - ustawiany programowo w celu zainicjowania transferu
  - zerowany sprzętowo po zakończeniu transferu
- Po wyzerowaniu bitu `EN` ustawiany jest bit `TCIFx` (ang. *transfer complete interrupt flag*) w rejestrze `LISR` lub `HISR`
- Zwykle w procedurze obsługi przerwania trzeba obsłużyć zakończenie wysyłania, np. zwolnić bufor
- Bit `TCIFx` zeruje się w procedurze obsługi przerwania, zapisując bit `CTCIFx` w rejestrze `LIFCR` lub `HIFCR` – patrz omawiany przed chwilą przykład
- Zatem transfer można zainicjować, jeśli wyzerowane są bity `EN` i `TCIFx`, czyli np. gdy

```
(DMA1_Stream6->CR & DMA_SxCR_EN) == 0 &&
(DMA1->HISR & DMA_HISR_TCIF6) == 0
```

- Ważna jest kolejność sprawdzania tych bitów!

Problem, cd.

- Jeśli ten warunek nie jest spełniony, transfer trzeba skolejkować
- W procedurze obsługi przerwania, jeśli w kolejce czeka żądanie, trzeba je zainicjować
- Dostęp do zasobów (pamięć, rejestry), z których korzysta się w procedurze obsługi przerwania, poza kontekstem przerwania musi odbywać się przy zablokowanym przerwaniu

USART-y w zestawie laboratoryjnym – podsumowanie

- USART2 – diagnostyczny port szeregowy
  - podłączony do szyny APB1
  - korzysta z DMA1
  - taktowany zegarem PCLK1
- USART1 – port szeregowy do komunikacji z modułem Bluetooth
  - podłączony do szyny APB2
  - korzysta z DMA2
  - taktowany zegarem PCLK2
- Konfiguracja

	USART2		USART1	
	RXD	TXD	RXD	TXD
Wyprowadzenie	PA3	PA2	PA10	PA9
Strumień/Kanał	5/4	6/4	5/4	7/4
Strumień/Kanał	7/6		2/4	

- W STM32F411 po uruchomieniu mikrokontrolera zegary PCLK1 i PCLK2 mają częstotliwość 16 MHz

Priorytety przerwań, przypomnienie

- Przerwanie może mieć ujemną lub nieujemną wartość priorytetu
- Im mniejsza wartość, tym wyższy priorytet
- Trzy przerwania mają ustalony, ujemny i niekonfigurowalny priorytet
  - *Reset* – priorytet -3
  - *NMI* – priorytet -2
  - *Hard fault* – priorytet -1
- Pozostałym przerwaniom można nadać priorytet, który jest liczbą nieujemną z zakresu od 0 do 15
- Przerwania są obsługiwane w kolejności malejących priorytetów
- Jeśli oczekujące przerwania mają ten sam priorytet, to najpierw obsługiwane jest to o mniejszym numerze

Blokowanie przerwań

Figure 5. PRIMASK bit assignments

Table 8. PRIMASK register bit definitions

Bits	Description
Bits 31:1	Reserved
Bit 0	**PRIMASK**: 0: No effect 1: Prevents the activation of all exceptions with configurable priority.

Źródło: PM0214 – Programming manual, STM32F3 and STM32F4 Series Cortex-M4

Blokowanie przerwań

- Patrz plik `irq.h` w katalogu `/opt/arm/stm32/inc`
- Typ przechowujący aktualny poziom blokowania przerwań

```
typedef uint32_t irq_level_t;
```

- Funkcja blokująca wszystkie przerwania o konfigurowalnym priorytecie i zwracająca aktualny stan blokowania

```
static inline irq_level_t
IRQprotectAll(void) {
  volatile irq_level_t level;
  level = __get_PRIMASK();
  __disable_irq();
  return level;
}
```

Blokowanie przerwań

- Funkcja przywracająca poprzedni stan blokowania przerwań

```
static inline void
IRQunprotectAll(irq_level_t level) {
  __set_PRIMASK(level);
}
```

- Dlaczego nie korzystamy z funkcji `__enable_irq()`?

Blokowanie przerwań

- Podobnie można też blokować wszystkie przerwania maskowalne
- Rejestr `FAULTMASK`
- Funkcje biblioteczne

```
void     __disable_fault_irq(void)
void     __enable_fault_irq(void)
uint32_t __get_FAULTMASK(void)
void     __set_FAULTMASK(uint32_t faultMask)
```

- Które przerwania są maskowalne?

Grupowanie priorytetów i przerwania zagnieżdżone, przypomnienie

- Czterobitowa wartość priorytetu składa się z dwóch pól:
  - priorytetu wywłaszczania – bardziej znaczące bity
  - podpriorytetu – mniej znaczące bity
- Domyślnie wszystkie bity składają się na priorytet wywłaszczania
- Przerwania zagnieżdżone:
  - przerwania o tym samym lub niższym priorytecie wywłaszczania oczekują na obsłużenie, nie przerywając wykonania bieżącej funkcji obsługi
  - przerwania o wyższym priorytecie wywłaszczania przerywają wykonanie bieżącej funkcji obsługi i są obsługiwane w pierwszej kolejności

Grupowanie priorytetów

- W rzeczywistości w rejestrach konfiguracyjnych pole zawierające priorytet jest ośmiobitowe
- Cortex-M3 i Cortex-M4 implementują tylko cztery starsze bity
- Cztery młodsze bity są zarezerwowane i są czytane jako zera

Grupowanie priorytetów

Figure 7. BASEPRI bit assignments

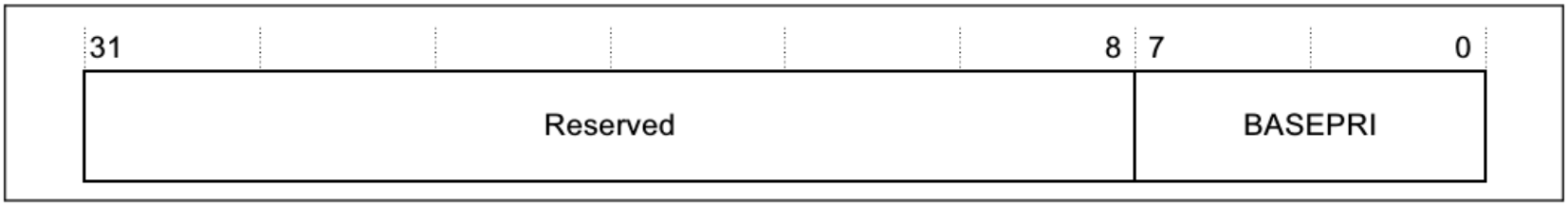


Table 10. BASEPRI register bit assignments

Bits	Function
Bits 31:8	Reserved
Bits 7:4	**BASEPRI[7:4]** Priority mask bits(1) 0x00: no effect Nonzero: defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.
Bits 3:0	Reserved

1. This field is similar to the priority fields in the interrupt priority registers. See *Interrupt priority registers (NVIC_IPRx) on page 200* for more information. Remember that higher priority field values correspond to lower exception priorities.

Źródło: PM0214 – Programming manual, STM32F3 and STM32F4 Series Cortex-M4

Grupowanie priorytetów

- Do definiowania miejsca podziału wartości priorytetu na pola służy funkcja biblioteczna

  `void NVIC_SetPriorityGrouping(uint32_t PriorityGroup)`

- Wartość parametru ustala postać priorytetu

`PriorityGroup`	Postać priorytetu
3	`0bxxxx`
4	`0bxxx.y`
5	`0bxx.yy`
6	`0bx.yyy`
7	`0byyyy`

Grupowanie priorytetów

- Wygodnie jest zdefiniować pomocniczą funkcję

```
#define PREEM_PRIO_BITS 2U

static inline void
IRQprotectionConfigure(void) {
  NVIC_SetPriorityGrouping(7U - PREEM_PRIO_BITS);
}
```

- Patrz plik `irq.h` w katalogu `/opt/arm/stm32/inc`

Blokowanie przerwań

- Funkcja blokująca przerwania o zadanym i niższym priorytecie i zwracająca aktualny stan blokowania

```
static inline irq_level_t
IRQprotect(uint32_t priority) {
  irq_level_t level;
  level = __get_BASEPRI();
  priority <<= 8U - PREEMP_PRIO_BITS;
  if (level == 0 || priority < level) {
     __set_BASEPRI(priority);
  }
  return level;
}
```

- Warunek `level == 0` oznacza, że żaden poziom ochrony nie był aktywny
- Warunek `priority < level` oznacza, że dopuszczamy jedynie podwyższenie poziomu ochrony

Blokowanie przerwań

- Funkcja przywracająca poprzedni stan blokowania przerwań

```
static inline void
IRQunprotect(irq_level_t level) {
  __set_BASEPRI(level);
}
```

Blokowanie przerwań

- Parametr `priority` funkcji `IRQprotect` nie może przyjmować wartości zero – dlaczego?
- Co zrobić, jeśli chcemy wyłączyć przerwanie o priorytecie 0 lub niższym?
- Użyć funkcji `IRQprotectAll`

Priorytety przerwań

- Do ustawiania priorytetu przerwania służy funkcja

```
void NVIC_SetPriority(IRQn_Type irq,
                      uint32_t priority)
```

- Wygodnie jest zdefiniować pomocniczą funkcję

```
static inline void
IRQsetPriority(IRQn_Type irq,
               uint32_t prio,
               uint32_t subprio) {
  NVIC_SetPriority(
    irq,
    NVIC_EncodePriority(
      NVIC_GetPriorityGrouping(),
      prio, subprio
    )
  );
}
```

- W tych funkcjach wartość `irq` może być ujemna

Priorytety przerwań

- Wygodnie jest zdefiniować stałe

```
#define VERY_HIGH_IRQ_PRIO      0U
#define HIGH_IRQ_PRIO           1U
#define MIDDLE_IRQ_PRIO         2U
#define LOW_IRQ_PRIO            3U

#define VERY_HIGH_IRQ_SUBPRIO   0U
#define HIGH_IRQ_SUBPRIO        1U
#define MIDDLE_IRQ_SUBPRIO      2U
#define LOW_IRQ_SUBPRIO         3U
```

- Patrz plik `irq.h` w katalogu `/opt/arm/stm32/inc`
- UWAGA: wywołanie `IRQprotect(VERY_HIGH_IRQ_PRIO)` nie ma żadnego efektu

Przykład sekcji krytycznej

- Wewnątrz funkcji obsługi przerwania o priorytecie LOW_IRQ_PRIO chcemy odwołać się do zmiennych, do których odwołuje się funkcja obsługi innego przerwania o priorytecie MIDDLE_IRQ_PRIO

```
void IRQHandler() {
  irq_level_t irq_level;

  ...

  irq_level = IRQprotect(MIDDLE_IRQ_PRIO);

  /* sekcja krytyczna */

  IRQunprotect(irq_level);

  ...
}
```