Ćwiczenie 3: Przerwania i timer systemowy

Zakres materiału

1. Kontroler przewań NVIC.
2. Timer systemowy SysTick
3. Wykorzystanie przerwań od timera systemowego i portów GPIOx.

Wprowadzenie do ćwiczeń

1. **Kontroler przerwań NVIC**

W mikrokontrolerach z rdzeniem Cortex-M3 za obslugę przerwań odpowiada sprzętowy kontroler przerwań (NVIC — Nested Vectored Interrupt Controller). Dzięki wykorzystaniu NVIC podprogram, kt6ry ma zosta6 wykonany po nadejściu przerwania, jest wywoływany szybciej, a sama implementacja obsługi przerwań jest łatwiejsza.

System ustalania priorytetów w mikrokontrolerach wyposażonych w rdzeń Cortex--M3 jest podzielony na dwie części. Przerwania mają przypisany priorytet grupowy tzw. preemption prioritet, ponadto jest ustalany dodatkowy poziom podpriorytetów (subpriority). Powyższy podział ma uzasadnienie w działaniu: gdy obsługiwane jest w danej chwili przerwanie i nadejdzie zgłoszenie od innego przerwania, to NVIC porównuje priorytety grupowe. Jeżeli nowe przerwanie dysponuje wyższym priorytetem, to następuje jego wywłaszczenie poprzedniego i teraz obsługiwane będzie przerwanie o wyższym priorytecie grupowym. Ponadto podział pomiędzy priorytetami grupowymi i podpriorytetami można dynamicznie modyfikować.

Odpowiednią grupę priorytetów można wybrać za pomocą funkcji:

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1)
```

Grupy priorytetów pokazano w poniższej tabeli:

Grupa	Preemption priority	Subpriority
NVIC_PriorityGroup_0	0 bitów	4 bity
NVIC_PriorityGroup_1	1 bit	3 bity
NVIC_PriorityGroup_2	2 bity	2 bity
NVIC_PriorityGroup_3	3 bity	1 bit
NVIC_PriorityGroup_4	4 bity	0 bitów

Im wyższa grupa, tym więcej można użyć priorytetów grupowych, ale mniej podpriorytetów, np. w grupie 1 mamy 2 priorytety grupowe i 8 podpriorytetów w każdej z grup, a w grupie 2 – 4 priorytety grupowe i 4 podpriorytety w każdej z grup.

2. **SysTick**

W kontroler przerwań jest wbudowany timer systemowy SysTick. Jest to 24-bitowy układ licznikowy, który zliczając w dół odmierza określone, zdefiniowane przez programistę, interwały czasowe. Każde przejście licznika przez zero i rozpoczęcie nowego cyklu odliczania powoduje wygenerowanie przerwania (jeśli jest ono włączone), które może zajmować się przełączaniem kontekstów uruchomionych w systemie zadań. Powyższe podejście ma poważną zaletę: zawieszenie się któregoś z realizowanych zadań nie spowoduje zawieszenia całego systemu, ponieważ zawieszony proces zostanie przerwany na rzecz innych zadań przy

najbliższym przepełnieniu timera SysTick. Oczywiście SysTick może być wykorzystywany do różnych zadań. Konstruktor ustala wartość, od jakiej timer ma liczyć w dół, tym samym wyznacza, jakie odcinki czasowe będą odmierzane, czyli może być on również wykorzystywany do standardowego odmierzania czasu. Konfigurację SysTick pokazano w przykładzie 1.

Domyślnie źródłem sygnału zegarowego dla timera SysTick jest sygnał zegara systemowego HCLK. Zmiany na drugą opcję taktowania, czyli z częstotliwością HCLK podzieloną przez 8, można dokonać za pomocą wywołania funkcji SysTick_CLKSourceConfig(). Całość dalszej konfiguracji timera do pracy jest przeprowadzana z wykorzystaniem tylko jednej funkcji — SysTickConfig(). Licznik timera zlicza w dół, więc regulacja długości odcinków czasowych, po których są generowane przerwania, obywa się poprzez ustawienie wartości początkowej, co dokonuje się poprzez przekazanie odpowiedniego argumentu do funkcji konfigurującej SysTickConfig(). W programie przedstawionym w przykładzie 1 przerwanie ma być generowane co 0,5 s, a więc przy częstotliwości taktowania timera wynoszącej 9 MHz, mikrokontroler musi zliczyć 4500000 taktów. Funkcja SysTickConfig() odblokowuje przerwanie od timera oraz włącza zliczanie, zatem po jej wykonaniu SysTick rozpoczyna pracę. Od tego momentu w określonych odstępach czasu będzie wywoływać się funkcja obsługi timera – SysTickHandler(); Przykładową zawartość tej funkcji pokazano w przykładzie 1.

3. Przerwania zewnętrzne

Do rodziny przerwań zewnętrznych zaliczamy wszystkie te przerwania, które do systemu trafiają poprzez fizyczne wyprowadzenia mikrokontrolera. Każde uniwersalne wejście/wyjście mikrokontrolera STM32F103CB może być skonfigurowane jako Źródło przerwania. Wszystkie dostępne w mikrokontrolerach STM32 przerwania są obsługiwane przez funkcje zdefiniowane przez firmę ST, które umieszczono w pliku stm32f10x_it.c.

Pięć młodszych pinów każdego portu ma przyporządkowaną własną funkcję obsługi przerwania. Innymi słowy, wszystkim wyprowadzeniom o nazwach Px od 0 do 4, gdzie x to określony port (A, B, C itd.) odpowiada funkcja obsługi przerwania o nazwie EXTIy_IRQHandler(), gdzie y to numer wyprowadzenia od 0 do 4. Pozostałe linie portów — od numeru 5 do 15 — podzielono na dwie grupy i dla każdej z nich zdefiniowano funkcję. Wyprowadzenia Px od 5 do 9 są obsługiwane przez funkcję EXTI9_5_IRQHandler(). Analogicznie wyprowadzenia Px 10 do Px 15 są powiązane z funkcją EXTI15_10_IRQHandler().

UWAGA! Aby skorzystać z funkcji i struktur obsługujących przerwania zewnętrzne, należy do projektu dodatkowo dodać moduł **EXTI** z biblioteki **Device → StdPeriph Drivers**.

Pytania kontrolne:

1. Co to jest SysTick?
2. Jaka jest organizacja priorytetów przerwań w mikrokontrolerze STM32?
3. Jakie funkcje odpowiadają za obsługę przerwań zewnętrznych i przerwania od SysTick?

Przykłady:

1. Konfiguracja timera systemowego SysTick

Program miga diodą LED z częstotliwością 1Hz. Diodę podłączono do wyprowadzenia 0 portu GPIOB i wykorzystano przerwanie od timera SysTick co 0,5s. Funkcja SysTickHandler() znajduje się w pliku stm32f10x_it.c, a kod główny w pliku main.c. Nie pokazano znanych już elementów konfiguracji portu i zegara. Kody źródłowe programu – w pliku cw3p1.zip:

```
#define SysTick_Frequency 9000000 // 9MHz

     int main(void)
     {
      RCC Conf();
      GPIO Conf();
```

```
      // Jeśli przerwanie ma byc co 1ms, f = 9MHz / 1000, czyli liczy od 9000
      if (SysTick_Config(SysTick_Frequency) / 2)  // przerwanie co ½ s
      {
            // W razie bledu petla nieskonczona
            while (1);
      }

      // SysTick bedzie taktowany z f = 72MHz/8 = 9MHz
      SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);

      while (1);
     }

     void SysTick_Handler(void)
     {
      GPIO_WriteBit(GPIOB,GPIO_Pin_0,(BitAction)(1-GPIO_ReadOutputDataBit(GPIOB,GPIO_Pin_0)));
     }
```

2. Przerwanie zewnętrzne

Program, który zmienia stan diody LED za pomocą przycisku wykorzystując przerwanie zewnętrzne. Przed uruchomieniem programu należy połączyć przewodami: PB1 – LED1, PA0 – SW0. Nie pokazano znanych już elementów konfiguracji portu i zegara. Funkcja EXTI0_IRQHandler() znajduje się w pliku stm32f10x_it.c, a kod główny w pliku main.c. (kod źródłowy znajduje się w pliku: cw3p2.zip).

```
int main(void)
//konfiguracja systemu
{
  RCC_Config();
  GPIO_Config();
  NVIC_Config();

  while (1) ;
  return 0;
}

void NVIC_Config(void)
{
//Konfigurowanie kontrolera przerwan NVIC

  NVIC_InitTypeDef  NVIC_InitStructure;
  EXTI_InitTypeDef  EXTI_InitStructure;

#ifdef  VECT_TAB_RAM
  // Jezeli tablica wektorow w RAM, to ustaw jej adres na 0x20000000
  NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  // VECT_TAB_FLASH
  // W przeciwnym wypadku ustaw na 0x08000000
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif

  //Konfiguracja NVIC - ustawienia priorytetow przerwania EXTI0
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);             //Wybor modelu grupowania przerwan

  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;               //Wybor konfigurowanego IRQ
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;      //Priorytet grupowy
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;             //Podpriorytet
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                //Wlaczenie obslugi IRQ
  NVIC_Init(&NVIC_InitStructure);

  GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);    //Ustawienie zrodla przerwania

  //Konfiguracja przerwania EXTI0 na linie 0
  EXTI_InitStructure.EXTI_Line = EXTI_Line0;                     //Wybor linii
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;    //Ustawienie generowania przerwania (a
nie zdarzenia)
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;        //Wyzwalanie zboczem opadajacym
(wcisniecie przycisku)
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;                      //Wlaczenie przerwania
  EXTI_Init(&EXTI_InitStructure);
```

```
}

void EXTI0_IRQHandler(void)
{
        //potwierdz zrodlo przerwania
  if (EXTI_GetITStatus(EXTI_Line0) != RESET){
    GPIO_WriteBit(GPIOB,GPIO_Pin_1,(BitAction) ((1-GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_1))));
    EXTI_ClearITPendingBit(EXTI_Line0);   //wyzeruj flage obslugi przerwania
  }

}
```

Zadania do samodzielnego wykonania:

1. Napisz funkcję opóźnienia void Delay (uint32_t ms) wykorzystującą timer SysTick, gdzie ms – wartość opóźnienia w milisekundach. Wykorzystaj ją np. do migania diodą LED.

2. Napisz program, który miga 8 diodami w taki sposób, że pierwsza miga z częstotliwością 2Hz, a każda następna miga z częstotliwością o 20% wyższą niż poprzednia.

3. Napisz program, który wykorzystując przerwania od wyprowadzeń zewnętrznych steruje miganiem diod LED w następujący sposób: Każdy z przycisków SW0 – SW4 włącza lub wyłącza miganie danej diody w następujący sposób: pierwsze naciśnięcie – dioda świeci, drugie dioda miga, trzecie – dioda gaśnie itd. Częstotliwość migania diod powinna być regulowana przy pomocy joysticka (np. góra-dół) w zakresie od 1 do 4Hz ze skokiem co 0,5 Hz (7 wartości).

4. Napisać program ilustrujący mechanizm priorytetów przerwań, np. przycisk SW0 o bardziej znaczącym priorytecie zapala diodę LED0 i gasi diodę LED1 na 3 sekundy (tyle czasu powinna trwać obsługa przerwania), a przycisk SW1 o mniej znaczącym priorytecie próbuje zapalić diodę LED1.