STM32 dla początkujących (i nie tylko)

Przerwania

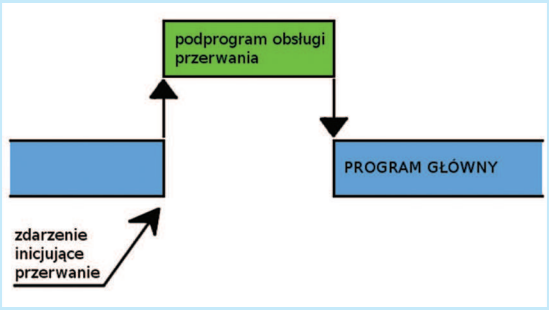
W kolejnej części kursu Panel Edukacyjny posłuży do zapoznania się z mechanizmem przerwań. Opiszę jak działają oraz pokażę sposób, w który można uruchomić i zaprząc do pracy we własnej aplikacji system przerwań mikrokontrolera STM32F.

Mówiąc w największym skrócie, przerwania są sposobem mikrokontrolera, by mógł zareagować na zdarzenia, których momentu pojawienia się nie sposób określić. Najlepiej w tym celu wyobrazić sobie hipotetyczny program, w którego pętli głównej zapracowany mikrokontroler bez przerwy obsługuje odczyt czasu z zegara, wyświetla ten czas na wyświetlaczu, obsługuje transmisję szeregową z zewnętrznego urządzenia. A jednocześnie musi zliczać krótkie impulsy na jednym z portów wejściowych, które mogą pojawić się w dowolnym momencie. Kontroler zajęty innymi zadaniami nie może bez przerwy sprawdzać stanu portu i przez to może przeoczyć fakt wystąpienia impulsu. Dzięki przerwaniu generowanemu w momencie wystąpienia impulsu wykonywanie głównej pętli programu zostanie zawieszone, nastąpi skok do podprogramu obsługującego zliczenie impulsu i powrót do wykonywania zadań w pętli głównej. Opisaną sytuację przedstawiono na **rysunku 1**.

Taka jest ogólna zasada działania mechanizmu przerwań. W mikrokontrolerze STM32F układami generującymi przerwania, czyli źródłami przerwań, mogą być nie tylko porty ale także: liczniki, przetworniki, porty transmisji szeregowej, zegar czasu rzeczywistego itd. W dokumentacji technicznej (RM0008) dostępnej na stronie producenta jako dokument *CD00171190.pdf* znaleźć można tabelę *Vector table for other STM32F10xxx devices* w której znajduje się zestawienie obsługiwanych przez kontroler źródeł przerwań. W tabeli obok opisu źródła przerwania podano jego priorytet (czyli określenie czy przerwanie jest mniej lub bardziej istotne a więc obsługiwane przed innymi) oraz adres w tabeli wektorów przerwań. Gdy zaistnieje przyczyna wywołująca przerwanie mikrokontroler po zawieszeniu wykonywania głównego programu skacze pod ten właśnie adres. Szuka tam kolejnego adresu (wektora) wskazującego na początek podprogramu obsługi przerwania. Na pierwszych pozycjach tabeli zaznaczone szarym kolorem tła znajdują się przerwania systemowe związane ze stanem samego kontrolera, debugowaniem programu itp.

Tabela 1. Wektory przerwań

Nazwa	Opis	Adres wektora
EXTI0	Przerwanie linii0	0x0000_0058
EXTI1	Przerwanie linii1	0x0000_005C
EXTI2	Przerwanie linii2	0x0000_0060
EXTI3	Przerwanie linii3	0x0000_0064
EXTI4	Przerwanie linii4	0x0000_0068
EXTI9_5	Przerwanie linii od 5 do 9	0x0000_009C
EXTI15_10	Przerwanie linii od 10 do 15	0x0000_00E0



Rysunek 1. Sposób działania programu

Dla łatwiejszego zrozumienia jak uruchamia się przerwanie w opisie skupię się na przerwaniach związanych z liniami I/O. Sposób uruchomienie przerwań z innych źródeł różni się tylko w szczegółach.

Mechanizm przerwań linii I/O

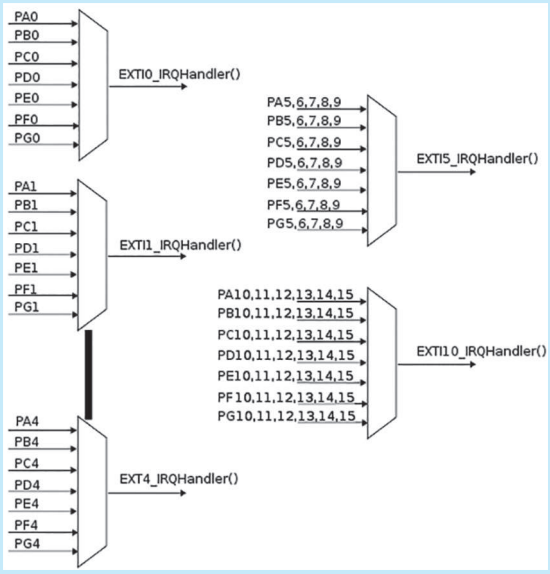
W **tabeli 1** umieszczono część zawartości *Vector table for other STM32F10xxx devices* dotycząca przerwań linii I/O

Ponieważ kontrolery STM32F mają dużą i zależną od typu i rodzaju obudowy liczbę portów I/O, w tabeli pogrupowano wektory przerwań generowane przez linie I/O. Wszystkie linie o numerze 0 portów A, B, C ... wywołują wspólne przerwanie EXTI0, linie 1 wywołują przerwanie EXTI1 itd. Z kolei, linie od 5 do 9 portów A, B, C ... wywołują wspólne przerwanie EXTI9_5 a linie od 10 do 15 wywołują przerwanie EXTI15_10. W sposób schematyczny pokazano to na **rysunku 2**.

Cały proces uruchamiania mechanizmu przerwań sprowadza się do kilku łatwych do zapamiętania kroków:

Konfiguracji wyprowadzenia I/O. Zazwyczaj wyprowadzenie będzie ustawiane jako wejściowe ale nie jest to konieczne. Wyprowadzenie może mieć wewnętrzne podciąganie do napięcia zasilania lub poziomu masy lub pozostawione bez podciągania. Dokładny opis konfiguracji wyprowadzeń I/O znajduje się w jednym z poprzednich odcinków kursu.

- Napisanie podprogramu obsługi przerwania wyprowadzenia I/O.
- Inicjacja kontrolera przerwań NVIC.



Rysunek 2. Struktura przerwań zewnętrznych

- Inicjacja parametrów przerwania i włączenie przerwania.

Poniżej zostaną dokładniej opisane punkty od 2 do 4. Dodatkowo można zapoznać się z przykładem znajdującym się w *STM32F10x Standard Peripherals Firmware Library* w podkatalogu Project/STM32F10x_StdPeriph_Examples/EXTI/EXTI_Config.

Ponieważ w przykładach wykorzystywane będą funkcje oferowane przez bibliotekę standardową warto zapoznać się z ich opisem. Można to zrobić w sposób podawany w poprzednich odcinkach kursu. Należy uruchomić plik *stm32f10x_stdperiph_lib_um.chm* a w polu wyszukiwań wpisać skrót „EXTI". Z wyświetlonych opcji należy wybrać drugą a następnie z wyświetlonej listy kliknąć na *EXTI_Exported_Functions*.

Podprogram obsługi przerwania

Podprogramy przerwań, na które wskazują adresy w tablicy wektorów przerwań, należy umieścić w pliku o nazwie *stm32f10x_it.c*. Ten standardowo generowany plik powinien znaleźć się w każdym programie, nawet tym, w którym użytkownik nie uruchamia własnych przerwań, ponieważ umieszczone są w nim podprogramy, do których skacze kontroler po wystąpienia któregoś z przerwań systemowych. Domyślnie są to nieskończone pętle typu *while (1) {}*. W razie wystąpienia błędu systemowego generowanego np. w przypadku wyjścia poza zadeklarowany obszar pamięci dla tablicy czy próby dzielenia przez 0, mikrokontroler zatrzyma program w takiej pętli nie dopuszczając do większych szkód, czyli zupełnie niekontrolowanego działania, co popularnie nazywa się „pójściem w maliny". Na końcu pliku *stm32f10x_it.c* jest miejsce na dopisanie obsługi własnych przerwań.

Na **listingu 1** pokazano szkielet podprogramu, który zostanie wywołany przez przerwanie związane z linią 0 dowolnego portu:

Nazwa podprogramu – funkcji obsługującej przerwanie *EXTI0_IRQHandler()* nie jest dowolna. Nazwy funkcji obsługujących przerwanie zostały zdefiniowane w pliku *startup_stm32f10x_md.s* i nie powinny być zmieniane. W instrukcji warunkowej za pomocą funkcji bibliotecznej *EXTI_GetITStatus* następuje sprawdzenie czy przerwanie zostało wywołane przez linię 0. Jeżeli tak, to funkcja *EXTI_ClearITPendingBit* zeruje flagę przerwania. Jest to konieczne, gdyż dopóki flaga nie zostanie wyzerowana podprogram obsługi przerwania będzie nieustannie wywoływany. W dalszych liniach użytkownik może umieścić własne procedury obsługi przerwania.

Funkcję *EXTI0_IRQHandler()* wywoła przerwanie każdej linii 0 dowolnego portu: PA, PB itd. Jeżeli jako źródła przerwań zostały wybrane linie 0 kilku różnych portów, to procedury obsługi utworzone przez użytkownika powinny mieć możliwość rozróżnienia, który port jest źródłem przerwania. Szkielet podprogramu – funkcji dla linii 5-9 zamieszczono na **listingu 2**.

Ponieważ funkcja zbiorczo obsługuje przerwania linii od 5 do 9 funkcja biblioteczna *EXTI_GetITStatus* pozwala na zidentyfikowanie czy źródłem przerwania była linia 5, czy 6. Przed zakończeniem obsługi przerwania należy wyzerować flagę przerwania.

Analogiczny szkielet podprogramu – funkcji dla linii 10-15 pokazano na listingu 3.

Inicjacja kontrolera przerwań NVIC

W STM32F obsługą przerwań zajmuje się specjalny wewnętrzny układ nazwany NVIC (*Nested Vectored Interrupt Controller*). Uruchomienie jakiegokolwiek przerwania wiąże się z zaprogramowaniem działania tego układu. Należy ustalić schemat priorytetów

Listing 1. Szkielet podprogramu, który zostanie wywołany przez przerwanie związane z linią 0

```
//obługa przerwania Linii_IO o numerze 0 (dowolnego
portu)
void EXTI0_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line0) !=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line0);
        /*tutaj treść podprogramu obsługi przerwania*/
    }
}
```

Listing 2. Szkielet podprogramu – funkcji dla linii 5-9

```
// obługa przerwania Linii_IO o numerze 5-9 dowolnego
portu
void EXTI9_5_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line5) !=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line5);
        /*tutaj treść podprogramu obsługi przerwania*/
    }
    if (EXTI_GetITStatus(EXTI_Line6) !=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line6);
        /*tutaj treść podprogramu obsługi przerwania*/
    }
}
```

Listing 3. Szkielet podprogramu – funkcji dla linii 10-15

```
// obługa przerwania Linii_IO o numerze 10-15 dowolnego
portu
void EXTI15_10_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line14) !=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line14);
        /*tutaj treść podprogramu obsługi przerwania*/
    }
    if (EXTI_GetITStatus(EXTI_Line15) !=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line15);
        /*tutaj treść podprogramu obsługi przerwania*/
    }
}
```

Tabela 2. Priorytety i podpriorytety w grupach

Schemat	Liczba grup (preemption prioryty)	Liczba podpriorytetów w grupie (subpriority)
NVIC_PriorityGroup_0	1	16
NVIC_PriorityGroup_1	2	8
NVIC_PriorityGroup_2	4	4
NVIC_PriorityGroup_3	8	2
NVIC_PriorityGroup_4	16	1

Listing 4. Przykładowa funkcja konfigurująca NVIC
```
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);//8 grup 2 podpriorytety w grupie
    /*Enable the EXTI0_IRQn przerwanie Linii_IO o numerze 0 dowolnego portu*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI0_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);
    /*Enable the EXTI1_IRQn przerwanie Linii_IO o numerze 1 dowolnego portu*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI1_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);
    /*Enable the EXTI2_IRQn przerwanie Linii_IO o numerze 2 dowolnego portu*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI2_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);
    /*Enable the EXTI3_IRQn przerwanie Linii_IO_3*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI3_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);
    /*Enable the EXTI4_IRQn przerwanie Linii_IO_4*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI4_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 4;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);
    /*Enable the EXTI9_5_IRQn przerwanie Linii_IO 5-9 dowolnego portu*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI9_5_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 5;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);  */
    /*Enable the EXTI15_10_IRQn przerwanie Linii_IO 10-15 dowolnego portu*/
    //NVIC_InitStructure.NVIC_IRQChannel =EXTI15_10_IRQn;
    //NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 5;
    //NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    //NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    //NVIC_Init(&NVIC_InitStructure);  */
}
```

i włączyć obsługę źródeł, które będą generowały przerwania.

Schemat priorytetów polega na ustalenia podziału pomiędzy ilością priorytetów grupowych (*preemption prioryty*) i ilością podpriorytetów (*subpriority*) w ramach każdej grupy.

Priorytet grupowy ustala znaczenie przerwania. Wystąpienie przerwania o wyższym priorytecie grupowym spowoduje, że zostanie ono obsłużone przed przerwaniem o niższym priorytecie. Dodatkowo obsługa przerwania o niższym priorytecie może zostać przerwana na czas obsługi przerwania o wyższym priorytecie.

Podpriorytet określa znaczenie przerwania w ramach grupy. Gdy wystąpią jednocześnie żądania obsługi przerwań z tej samej grupy, jako pierwsze zostanie obsłużone przerwanie o wyższym podpriorytecie.

Listing 5. Inicjowanie przerwania PA.0
```
Void Inicjacja_Wlaczenie_Przerwania_Linii0(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
    EXTI_InitStructure.EXTI_Line =EXTI_Line0;
    EXTI_InitStructure.EXTI_Mode =EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger =EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd =ENABLE;
    EXTI_Init(&EXTI_InitStructure);//włączenie przerwania
}
```

Listing 6. Podprogram obsługi przerwania pochodzącego od kanału CC1
```
//TIM4 obsługa przerwania
void TIM4_IRQHandler(void)
{
  if (TIM_GetITStatus(TIM4, TIM_IT_CC1) != RESET)
  {
    //kanał CC1
    TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
    //tutaj kod użytkownika obsługi przerwania
    //wyłączenie przerwania kanału
    TIM_ITConfig(TIM4, TIM_IT_CC1, DISABLE);
  }
}
```

Listing 7. Inicjowanie NVIC – włączenie przerwania od Timera 4
```
/* Enable the TIM4 global Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Schemat priorytetów polega na wyborze ilości grup priorytetów i ilości podpriorytetów w ramach każdej grupy. Możliwe schematy umieszczono w tabeli 2.

Wagi grup i podpriorytetów oznaczone są liczbami. Przerwania o wyższych priorytetach mają wagi o niższych numerach, 0 oznacza najwyższy priorytet.

Oprócz określenia schematu priorytetów należy zaprogramować NVIC do obsługi konkretnego źródła przerwania oraz określić jego priorytet. Przykładowa konfiguracja NVIC-a może wyglądać jak na **listingu 4**.

W przykładzie wybrany został następujący schemat: 8 grup po 2 poziomy podpriorytetów w grupie. Najwyższy priorytet będzie miało przerwanie *EXTI0_IRQn*, niższy *EXTI1_IRQn* itd. Przerwania *EXTI9_5_IRQn* i *EXTI15_10_IRQn* znalazły się w grupie o najniższym priorytecie. Przy czym pierwsze z nich ma wyższy podpriorytet.

W zależności od rzeczywiście użytych we własnym programie przerwań należy odkomentować właściwy fragment przykładowego kodu. Można też ustawić inny schemat priorytetów i ich przyporządkowanie.

Inicjowanie parametrów i włączenie przerwania

Ostatnim krokiem jest ustawienie parametrów związanych z konkretnym źródłem przerwania i włączenie samego przerwania. Fragment programu na **listingu 5** pokazuje jak można to zrobić dla linii PA.0. Tak jak wcześniej posłużą do tego procedury biblioteki *STM32F10x Standard Peripherals Firmware Library*.

Linia PA.0 wygeneruje przerwanie po wystąpieniu opadającego zbocza podanego sygnału.

Przykład włączenia przerwania Timera

Na koniec przykład programu pokazujący sposób włączenia przerwania licznika. Źródłem będzie Timer4, a konkretnie sytuacja, gdy wartość licznika będzie równa wartości wpisanej w rejestrze porównań CC1. Sposób konfigurowania liczników do różnych trybów pracy w tym, gdy wykorzystywane są rejestry porównań *Caption*, był opisany w poprzednich odcinkach kursu. Tutaj pokażę tylko fragmenty programu, które trzeba dodać, aby uruchomić mechanizm przerwania.

Na **listingu 6** zamieszczono podprogram obsługi przerwania pochodzącego od kanału CC1. Należy go umieścić w pliku *stm32f10x_it.c*. Na zakończenie obsługi podprogram wyłącza przerwanie. Kolejny fragment programu, który należy dodać do kodu inicjacji NVIC, zamieszczono na **listingu 7**. Na koniec należy włączyć przerwanie kanału CC1 Timera4 TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);

Podobny schemat postępowania obowiązuje w przypadku włączania innych źródeł przerwań.

Ryszard Szymaniak, EP