Wszystkie listingi publikujemy na CD >>

Tryby obniżonego poboru mocy

Od pewnego czasy wymaga się, aby urządzenia zasilane energią elektryczną pobierały jej jak najmniej. Ma to związek po pierwsze z ochroną środowiska naturalnego a po drugie z wygodą użytkowania urządzeń przenośnych zasilanych z baterii. W tym artykule prezentujemy w jaki sposób wykorzystać tryby obniżonego poboru energii mikrokontrolera STM32.

Obniżenie poboru energii przez mikrokontroler można osiągnąć bądź to na drodze spowolnienia jego pracy, bądź też wyłączając niepotrzebne, wbudowane w strukturę układy peryferyjne. W większości aplikacji przez znaczną część czasu mikrokontroler wykonuje pętlę nieskończoną (nie wykonując przy tym żadnych pożytecznych zadań) oczekując na zdarzenie wymagające reakcji programu. Stąd idea trybów obniżonego poboru mocy. Dzięki wyłączeniu zbędnych układów peryferyjnych można znacznie zredukować pobór prądu

Mikrokontroler, który zamontowano na płytce ewaluacyjnej STM3210B – EVAL/A, może pracować w jednym z trzech głównych trybów obniżonego poboru mocy. Wyboru, którego z nich użyć, należy dokonać na podstawie analizy określającej, które z układów peryferyjnych można wyłączyć bez zmniejszania jego funkcjonalności.

Tryb uśpienia *(sleep mode)*

W tym trybie wyłączany jest rdzeń, natomiast praca wszystkich urządzeń peryferyjnych i źródeł sygnałów zegarowych pozostaje niezakłócona. Czas wyjścia z trybu uśpienia i powrót do normalnej pracy jest w tym przypadku najkrótszy ze wszystkich dostępnych trybów obniżonego poboru mocy. Prąd, jaki jest pobierany ze źródła zasilania, zależy przede wszystkim od częstotliwości pracy oscylatorów i ilości włączonych układów peryferyjnych. W skrajnym przypadku, gdy mikrokontroler pracuje z zegarem 72 MHz (włączone HSE i PLL) i są włączone wszystkie peryferia, to pobiera on blisko 15 mA. Drugim skrajnym przypadkiem jest praca systemu z zegarem 125 kHz przy wykorzystaniu wewnętrznego oscylatora HSI i po wyłączeniu wszystkich peryferiów. W takich warunkach ze źródła zasilania pobierane jest około 0,5 mA.

Tryb uśpienia jest ściśle związany z architekturą Cortex M3 i stanowi integralną część tego rdzenia, stąd dodatkowych informacji na jego temat należy szukać nie w dokumentacji firmy STMicroelectronics, ale w dokumentacji firmy ARM.

Zależnie od mechanizmów „usypiania" i „wybudzania" rozróżniane jest kilka rodzajów trybów pracy. Za sterowanie tymi trybami odpowiadają dwie instrukcje. Pierwszą jest instrukcja WFI (*Wait For Interrupt*), a drugą WFE (*Wait For Event*). Są to rozkazy asemblera, a zatem użycie ich w programie napisanym w C wymaga zastosowania podwójnego podkreślenia w roli przedrostka. Np., aby wywołać instrukcję WFI należy zastosować zapis: `__WFI();`.

Sleep-on-exit

Wykorzystanie instrukcji WFI umożliwia między innymi wprowadzenie rdzenia w stan nazwany *Sleep-on-exit*. W tym stanie rdzeń wprowadzany jest w tryb uśpienia dopiero po zakończeniu obsługi wszystkich przerwań. Wyjście z trybu jest wówczas, gdy w systemie zostanie wykryte żądanie obsługi przerwania. Fragment programu, który działa w opisany sposób przedstawiono na **list. 1**. Nadzór nad trybami uśpienia sprawowany jest przez NVIC, dlatego wykorzystano funkcje sterujące jego pracą.

Przerwanie od wyprowadzenia PB9, do którego podłączono przycisk, skonfigurowano do reakcji na zbocze opadające. Po jego pojawieniu rdzeń wyjdzie z trybu uśpienia i nastąpi wywołanie funkcji obsługi przerwania. Funkcję obsługi przerwania przedstawiono niżej

```
void EXTI9_5_IRQHandler(void)
{
  EXTI_ClearITPendingBit(EXTI_
Line9);
  GPIO_SetBits(GPIOC,GPIO_Pin_
6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9);
  delay_ms(2000);
  GPIO_ResetBits(GPIOC,GPIO_Pin_
6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9);
}
```

Rdzeń wybudza się na czas około 2 sekund zaświecając cztery diody LED, gasi je i ponownie „usypia".

Uruchomienie opisanego trybu w trakcie debugowania spowoduje przerwanie pracy debugera. Rdzeń nie pracuje, więc nie ma potrzeby poszukiwania błędów. Istnieje jednak możliwość takiej konfiguracji mikrokontrolera, że nawet w trybach obniżonego poboru mocy, połączenie debugera i mikrokontrolera nie zostanie zerwane. Do tego celu służy funkcja *DBGMCU_Config()*.Przykładowo, jeśli wykorzystywany jest tryb uśpienia, to należy umieścić w programie linijkę:

```
DBGMCU_Config(DBGMCU_SLEEP, ENABLE);
```

Sleep-now

Działanie tego trybu polega na natychmiastowym wprowadzeniu rdzenia w tryb obniżonego poboru mocy. Można to zrobić zarówno przy pomocy instrukcji WFI jak i WFE. Różnica polega na tym, że w pierwszym przypadku system będzie oczekiwał nadejścia przerwania, a w drugim zdarzenia. Czas potrzebny na wybudzenie rdzenia z trybu uśpienia w oczekiwaniu na zdarzenie (rozkaz WFE) jest prawie dwukrotnie krótszy i równy około 2 μs. Wynika to z faktu, że nie jest potrzebny czas na obsługę przerwania. Nasuwa się wniosek, że wykorzystanie instrukcji WFE bardzo dobrze nadaje się do współpracy z zegarem czasu rzeczywistego RTC, którego zadaniem może być generowanie zdarzeń (alarmów).

W programie z **list. 2** RTC skonfigurowano w taki sposób, aby wybudzać mikrokontroler co pół minuty. W trybie normalnej pracy świeci dioda LD1. Dodatkowo w tym przykładzie status aktualnego zadania osiąga najwyższy priorytet, a co za tym idzie NVIC nie przerwie jego wykonywania na rzecz obsługi przerwania. Można to zaobserwować, jeśli włączy się np. przerwanie od przycisku (PB9), a w funkcji obsługi przerwania umieści się fragment kodu, który będzie powodował widoczne efekty na płytce ewaluacyjnej. Przykład takiej funkcji umieszczono poniżej:

```
void EXTI0_IRQHandler(void)
{
  EXTI_ClearITPendingBit(EXTI_
Line0);
  GPIO_SetBits(GPIOC, GPIO_Pin_9);
}
```

W celu sprawdzenia, czy omawiana aplikacja działa poprawnie można uruchomić program z list. 2, z dodaną na początku głównej, nieskończonej pętli następującą linijką:

```
NVIC_SystemLPConfig(NVIC_LP_SEVON-
PEND, DISABLE);
```

Przerwania nie są blokowane, więc naciśnięcie przycisku w stanie aktywnym powinno spowodować zaświecenie diody LD4.

Deep-sleep

Ostatnim trybem obniżonego poboru mocy rdzenia jest „głębokie uśpienie". Nie jest to kolejny oddzielny tryb. Uzyskuje się go łącząc tryb natychmiastowego uśpienia (*Sleep-now*), lub z trybem uśpienia po obsłużeniu wszystkich przerwań (*Sleep-on-exit*). Tryb aktywowany jest przez ustawienie bitu SLEEPDEEP w rejestrze kontrolnym zarządzania zasilaniem kontrolera przerwań NVIC.

Programista używający biblioteki API nie musi zagłębiać się w te szczegóły, wystarczy użyć odpowiedniej funkcji: jeżeli aplikacja wymaga wprowadzenia rdzenia w stan głębokiego uśpienia, to należy umieścić w kodzie dodatkowo linijkę: `NVIC_SystemLPConfig(NVIC_LP_SLEEPDEEP, ENABLE);`

Konsekwencją włączenia *„Deep-sleep"* jest wyłączenie zegara rdzenia, łącznie z układem PLL. Pozwala to dalsze, w stosunku do poprzednich trybów, obniżenie poboru mocy. Negatywnym efektem zastosowania głębokiego uśpienia jest dłuższy czas potrzebny na całkowite wybudzenie i rozpoczęcie normalnej pracy. Jest to związane z tym, że pętla synchronizacji fazowej wymaga czasu, aby wygenerować stabilny sygnał zegarowy.

Tryb zatrzymania *(Stop mode)*

Wykorzystuje on omówiony wcześniej tryb głębokiego uśpienia. Wyłączone zostają wszystkie sygnały zegarowe (układ PLL i oba szybkie oscylatory: HSI i HSE). Ponadto wewnętrzny regulator napięcia 1,8 V można wprowadzić kosztem późniejszego dłuższego startu systemu w stan obniżonego poboru mocy.

W *„Stop mode"* nadal pracuje kilka urządzeń. Należy do nich między innymi układ niezależnego watchdoga. Należy brać to pod uwagę w aplikacjach wykorzystujących IWDG i pracujących w trybach obniżonego poboru mocy.

Jeżeli wcześniej nie zostaną wyłączone zegar czasu rzeczywistego i wolne oscylatory (LSE i LSI), to we wszystkich trybach obniżonego poboru mocy są w stanie aktywnej pracy.

Sposób wprowadzenie mikrokontrolera w tryb *„stop mode"* przedstawiono na **list. 3**. Ten prosty program ma zadanie wprowadzać mikrokontroler w tryb zatrzymania co ok. 2 sekundy, przy czym samo zatrzymanie trwa 10 sekund. Sygnalizacja normalnej pracy odbywa się przez zaświecenie diody LD1. Wybudzanie następuje pod wpływem alarmów generowanych za pomocą zegara czasu rzeczywistego. Po wyczyszczeniu flagi od zdarzenia od RTC następuje ustawienie czasu, po jakim ma być uformowany kolejny sygnał alarmu. W przykładzie mikrokontroler ma się wybudzać co 10 sekund, więc za każdym razem do aktualnej wartości licznika RTC należy dodać wartość 10. Osiągnięcie przez licznik otrzymanej w ten sposób liczby będzie skutkowało alarmem. Zadaniem kolejnej wywołanej funkcji (`PWR_EnterSTOPMode()` jest właściwe wprowadzenie układu w stan zatrzymania. W programie, ze względu na brak specjalnych wymagań co do minimalnego czasu startu systemu, regulator napięcia wprowadzany jest w tryb *„Low power"*.

W tym miejscu praca mikrokontrolera zostaje wstrzyma i od tego miejsca rozpocznie się po wybudzeniu. W związku z tym, że w trybie zatrzymania wyłączane są zarówno pętla synchronizacji fazowej jak i szybkie oscylatory, to po powrocie do normalnej pracy należy je ponownie włączyć. Jeżeli aplikacja korzysta z wewnętrznego oscylatora HSI i nie wykorzystuje układu PLL, to ponowna konfiguracja tych urządzeń nie jest potrzebna: system automatycznie startuje biorąc HSI za źródło swojego sygnału zegarowego.

Tryb czuwania (standby mode)

Użycie tego trybu pozwala na najbardziej drastyczne obniżenie poboru energii. Podobnie do trybu zatrzymania, *„Standby mode"* korzysta z głębokiego uśpienia rdzenia (*deep-sleep*), jednak stabilizator 1,8 V jest wyłączany bez dodatkowej interwencji użytkownika. Wejście w tryb czuwania powoduje utratę danych zawartych w pamięci RAM inaczej, niż w poprzednich trybach. Nienaruszone zostają natomiast dane zawarte w rejestrach chronionych przed utratą oraz nadal pracują wolne oscylatory (Backup domain), pod takim warunkiem, że do mikrokontrolera doprowadzono zasilanie awaryjne bateryjne Vbatt. Pracę kontynuuje układ niezależnego watchdoga (o ile został wcześniej włączony) i układy wybudzania.

Czas, jaki jest potrzebny na powrót do pracy z trybu czuwania wynosi minimum 50 μs, dla startu systemu z wewnętrznym oscylatorem HSI. Jeśli układ będzie się wybudzał do pracy z zewnętrznym oscylatorem HSE, a do uzyskania użytecznego sygnału zegarowego będzie wykorzystywał układ PLL, to czas potrzebny na wybudzanie będzie wielokrotnie dłuższy.

Pobór prądu, gdy włączone są: wolny oscylator i układ zegara czasu rzeczywistego, waha się od 3…3,5 mA, zależnie od wartości napięcia zasilającego.

Mikrokontroler wybudza się, pojawi się sygnał zerowania (*Reset*), po przepełnieniu niezależnego watchdoga (IWDG), przy narastającym zboczu sygnału na linii Wakeup, lub po alarmie zegara RTC. Układ rozpoczyna normalną pracę w taki sam sposób, jak po wystąpieniu sprzętowego sygnału *Reset*. Z tego powodu czas potrzebny na osiągnięcie normalnej pracy systemu jest najdłuższy. Oznaką tego, że uruchomienie następuje po wybudzeniu z trybu czuwania jest ustawiona flaga SBF w rejestrze PWR_CSR. W bibliotece funkcji API nazywa się ona *PWR_FLAG_SB.*

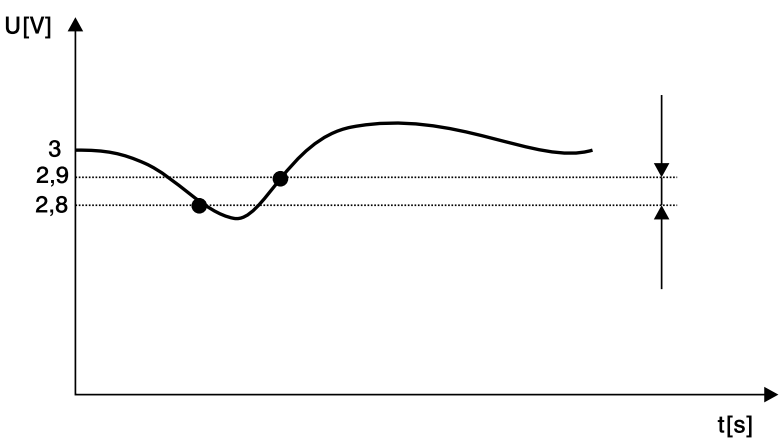
Na **list. 4** przedstawiono fragment programu wykorzystującego tryb czuwania. Aplikacja o stanie pracy mikrokontrolera informuje za pomocą diody LD1. Jej świecenie sygnalizuje normalną pracę, natomiast zgaszenie tryb czuwania. Układ powraca do normalnego stanu po naciśnięciu przycisku RESET lub WAKEUP.

Poza wymienionymi funkcjami, konfigurowany jest również układ zegara czasu rzeczywistego. Dzięki temu podczas debugowania można zaobserwować aktualny stan mikrokontrolera. Ponowne wprowadzenie trybu „standby mode" następuje po naciśnięciu przycisku (PB9).

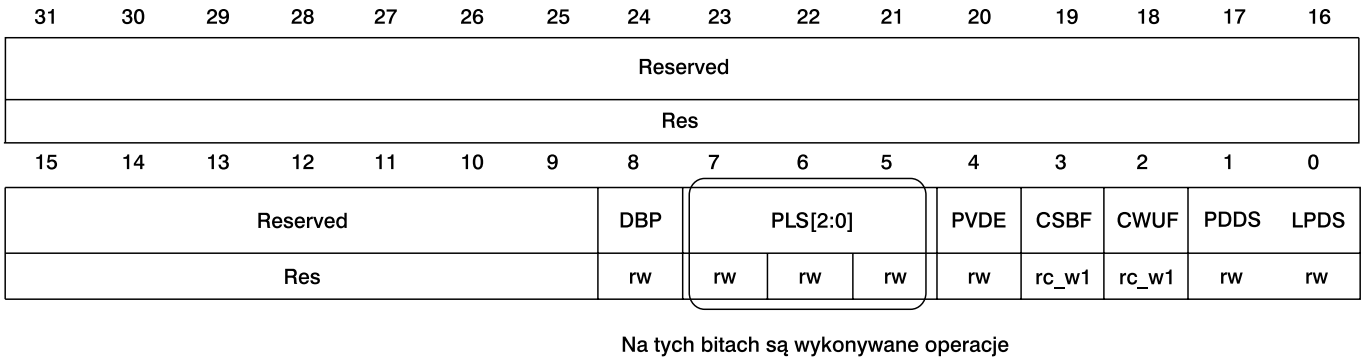
Zadaniem głównej, nieskończonej pętli programu jest nieustanne sprawdzanie stanu linii PB9. Gdy pojawi się na niej stan niski, to system jest przygotowywany do wejścia w tryb czuwania. Ostatecznie, wywołanie funkcji `PWR_EnterSTANDBYMode()` powoduje włączenie trybu *„Standby mode"*.

Na początku programu, po standardowym skonfigurowaniu systemu do pracy, włączana jest możliwość debugowania mikrokontrolera po wprowadzeniu do trybu czuwania oraz obsługa *Wakeup* i dostęp do rejestrów chronionych przed utratą. Kolejną, istotną czynnością, jest sprawdzenie, czy mikrokontroler rozpoczyna pracę po wybudzeniu, czy też nie. Jeżeli flaga PWR_FLAG_SB jest ustawiona, to zaświecana jest diody LD2, a wywołanie funkcji `PWR_ClearFlag()` powoduje wyzerowanie flagi trybu czuwania. Na koniec funkcja `RTC_WaitForSynchro()` sprawia, że mikrokontroler czeka na synchronizację z układem zegara czasu rzeczywistego. Jest ona niezbędna, ponieważ układ ten podłączony jest do wewnętrznej magistrali APB1, a w czasie czuwania jej taktowanie zostało wyłączone.

W przypadku, gdy flaga PWR_FLAG_SB była wyzerowana, konfigurowany jest układ zegara czasu rzeczywistego i jego sygnał zegarowy.



Rys. 1.



Rys. 2.

Programowany detektor napięcia

PVD (*Programmable Voltage Detector*) jest układem mającym za zadanie monitorowanie wartości napięcia zasilania. Progi, po których przekroczeniu generowane będzie przerwanie, mogą być programowane w szerokich granicach. PVD posiada histerezę równą 100 mV, która jest gwarancją stabilnej pracy układu. Programowany detektor poziomu napięcia jest wewnętrznie połączony z przerwaniem EXTI-16, które może być generowane w chwili, gdy napięcie będzie mniejsze, większe lub różne od podanej wartości.

Na **rys. 1** przedstawiono sytuację, w której układ PVD został skonfigurowany do generowania przy napięciu różnym od podanej wartości. Za dobre napięcie zasilania w tym przykładzie jest uznawane napięcie o wartości powyżej 3 V, natomiast próg PVD został ustalony na 2,8 V. Na rysunku pokazano również, w jaki sposób działa 100 mV histereza. Czarne punkty wskazują miejsca wygenerowania przerwania.

Napięcie progu wyzwalania może być programowane w granicach od 2,2 V do 2,9 V, z krokiem co 0,1 V. Tak szeroki zakres pozwala na dobór parametru stosownie do wymagań aplikacji. Napięcie ustala się wywołując funkcję `PWR_PVDLevelConfig()` z odpowiednim parametrem. Przykładowo, jeśli projektowane urządzenie wymaga generowania przerwania po przekroczeniu napięcia 2,8 V, to należy w programie umieścić linijkę: `PWR_PVDLevelConfig(PWR_PVDLevel_2V8);`.

Oczywiście nie należy zapominać o wcześniejszym włączeniu programowanego detektora poziomu napięcia wywołując funkcję `PWR_PVDCmd()` z argumentem *ENABLE*.

Układ monitorujący może zwracać wartość napięcia zasilania, jednak tego typu funkcja nie została zaimplementowana w bibliotekach firmy STMicroelectronics i do tego celu należ napisać własną, której przykład przedstawiono niżej:

```
u8 PWR_PVDGetLevel(void)
{
  u8 temp_val = 0;
  temp_val = (u8) ((PWR->CR >> 5) &
0x07);
  return temp_val;
}
```

Nieco zawiłe działanie na wartości wpisywanej do zmiennej `temp_val` podyktowane jest pozycją bitów ustawiających próg napięcia w rejestrze kontrolnym PWR_CR. Zajmują one pozycje 5…7, dlatego należy wykonać operację pięciokrotnego przesunięcia bitowego w prawo. Następnie w tym przykładzie zerowane są wszystkie bity oprócz trzech najmłodszych a wynik rzutowany na 8–bitowy typ bez znaku.

Funkcję wykorzystano w przykładzie programu zmieniającym stan wyprowadzeń zależnie od wartości napięcia. Został on podzielony na dwie części: program główny i funkcję obsługi przerwania od PVD. Zadaniem pierwszego jest wstępna konfiguracja PVD i jego przerwania. Stosowny fragment programu przedstawiono na **list. 5**. Przerwania zostały dokładnie omówione w **EP12/08**, stąd tutaj tylko pobieżnie opisano sposób ich konfiguracji i włączenia.

W pierwszej kolejności konfigurowany jest NVIC tak, aby był przygotowany na przyjęcie i obsługę przerwania z kanału 16. Następnie ustawiane są parametry samego przerwania. Tutaj ważny jest wybór zbocza, przy którym będzie ono generowane. Przedstawiony przykład działa w oparciu o reakcję na zejście napięcia zasilającego poniżej danego progu. W związku z tym przerwanie musi być generowane na zboczu opadającym. Po wykonaniu czynności konfiguracyjnych, mikrokontroler przechodzi do nieskończonej pętli `while()` i oczekuje na nadejście przerwania od programowanego detektora poziomu napięcia.

Drugim blokiem prezentowanej aplikacji jest funkcja obsługi przerwania od PVD – `PVD_IRQHandler()`. Umieszczono ją na **list. 6.** Wykonuje ona sprawdzenie źródła przerwania oraz za pomocą wcześniej opisywanej funkcji, odczytuje wartość napięcia. W tym przykładzie, po przekroczeniu każdego z progów ustalany jest stan wysoki kolejnego wyprowadzenia mikrokontrolera i ustawiany kolejny próg napięcia.

Pewne wątpliwości może budzić sposób realizacji tego przykładu na płytce ewaluacyjnej STM3210B – EVAL/A, która ma już wmontowany na stałe stabilizator napięcia 3,3 V. Można to w łatwy sposób obejść. Wystarczy po zaprogramowaniu mikrokontrolera odłączyć źródło napięcia zasilania i wpiąć się z zewnętrznym napięciem o wartości około 3 V na wyjście stabilizatora 3,3 V zyskując w ten sposób możliwość jego regulacji. W trakcie zmniejszania napięcia zasilania na kolejnych wyprowadzeniach portu GPIOB, od PB8 zaczynając, będą pojawiać się stany wysokie.

Podsumowanie

Różnorodność trybów obniżonego poboru mocy sprawia, że czasem nie do końca wiadomo, który tryb będzie optymalny dla danej aplikacji. Dlatego warto dokładnie zapoznać się możliwościami oferowanymi przez poszczególne tryby. Dzięki temu będzie można zaprojektować urządzenie lepiej zoptymalizowane, o dużo mniejszym zużyciu energii, a przez to tańsze w eksploatacji. Ma to szczególne znaczenie w przypadku urządzeń zasilanych z baterii.

Krzysztof Paprocki, EP
krzysztof.paprocki@ep.com.pl