Mikrokontrolery STM32

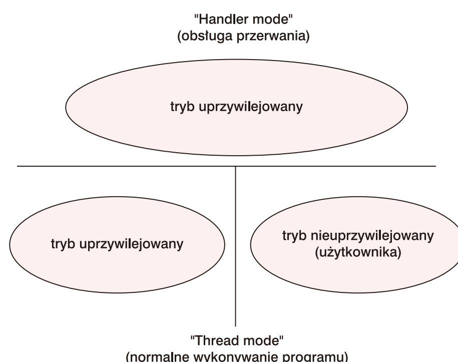
Bezpieczeństwo i stabilność aplikacji

Architektura rdzenia Cortex M3 oferuje wiele udogodnień istotnych przy tworzeniu wbudowanych systemów operacyjnych. Dzięki przedstawionym w niniejszym artykule mechanizmom implementacja RTOS jest znacznie uproszczona w stosunku do mikrokontrolerów wyposażonych w inne rdzenie. Dodatkowym atutem rdzenia Cortex M3 jest to, że aktualnie już kilku wiodących producentów ma w swojej ofercie mikrokontrolery z tym rdzeniem, co znakomicie poprawia przenośność aplikacji. Podane w artykule informacje dotyczą mikrokontrolerów STM32.

Aplikacje pisane z myślą o wykorzystaniu w systemie operacyjnym działającym na mikrokontrolerze określonego producenta z rdzeniem Cortex M3, mogą być uruchomione na dowolnym innym mikrokontrolerze z tym rdzeniem. Aplikacja korzysta z API, którego zadaniem jest należyta komunikacja ze sprzętem, stąd tak duża przenośność w obrębie mikrokontrolerów z rdzeniem Cortex. Jest to poważny argument przemawiający za stosowaniem we własnych rozwiązaniach układów z tym rdzeniem.

Tryby pracy rdzenia Cortex

Z punktu widzenia wykonywanego kodu, mikrokontroler może pracować wykonując program normalnie (*Thread mode* – ***TM***), albo może obsługiwać przerwanie (*Handler mode* – ***HM***). Obie sytuacje przedstawiono na **rys. 1**. Takie rozróżnienie ma kluczowe znaczenie dla aplikacji opartych o systemy operacyjne, sprawa ta poruszona jest w dalszej części tego artykułu.



Rys. 2.

Dodatkowo rozróżnia się dwa poziomy o różnych prawach dostępu do kluczowych obszarów w przestrzeni adresowej:
- tryb uprzywilejowany (*Privileged level* – ***PL***),
- tryb użytkownika (*Unprivileged*/*User level* – ***UL***).

Powyższy podział ma uzasadnienie w aplikacjach pracujących pod kontrolą systemu operacyjnego (***OS*** – *Operating system*). System operacyjny pracuje w trybie uprzywilejowanym, natomiast program użytkownika uruchamiany jest przez **OS** i wykonywany w trybie o ograniczonych możliwościach.

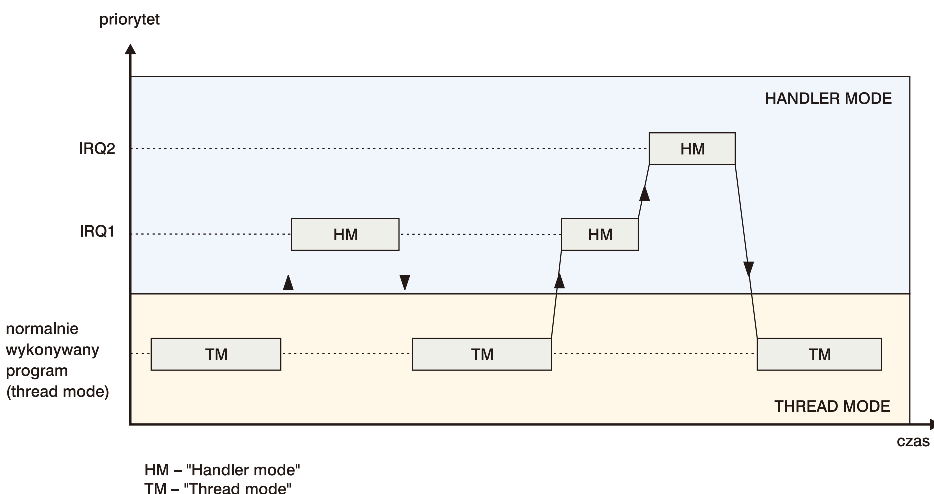
Mikrokontroler obsługując przerwanie zawsze pracuje w trybie uprzywilejowanym (**PL**) nawet, jeśli w trakcie normalnego wykonywania programu ustawiony jest tryb użytkownika (**UL**). Te relacje przedstawiono na **rys. 2**, natomiast **rys. 3** przedstawia pracę mikrokontrolera z włączonym trybem użytkownika. Z ostatniego rysunku jasno wynika, że rdzeń Cortex pracując w trybie **UL,** w czasie wchodzenia od obsługi przerwania, przełącza się w tryb **PL**, natomiast kończąc obsługę przerwania wraca do poprzedniego trybu nieuprzywilejowanego (użytkownika).

Przedstawiony mechanizm znacznie zwiększa stabilność systemu mikroprocesorowego, ponieważ aplikacja użytkownika, która ze swej natury posiada większe prawdopodobieństwo wygenerowania błędnych zachowań, ma ograniczony dostęp do kluczowych zasobów (np. NVIC i SysTick), a co za tym idzie, nie może bezpośrednio zachwiać stabilności pracy całego układu.

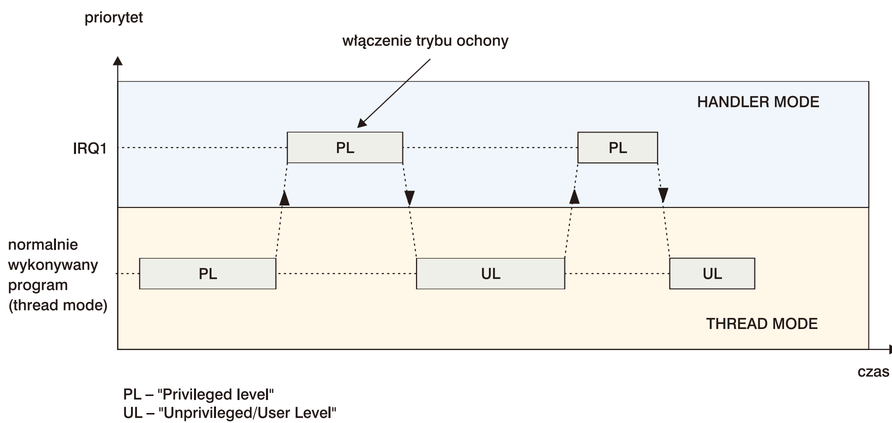
Po uruchomieniu mikrokontroler zawsze rozpoczyna pracę w trybie uprzywilejowanym, a zatem do zadań systemu operacyjnego w pierwszej kolejności należy, jeszcze przed uruchomieniem aplikacji użytkownika, włączenie trybu nieuprzywilejowanego – **UL**.

Przejście z trybu użytkownika do trybu uprzywilejowanego nie jest możliwe bezpośrednio. Aplikacja uruchomiona w systemie operacyjnym nie ma możliwości wyłączenia trybu użytkownika, ponieważ nie ma dostępu do specjalnego rejestru kontrolnego CONTROL. Zapis do niego z poziomu aplikacji jest ignorowany. Zmiany można dokonać tylko podczas obsługi przerwania, która jest przeprowadzana w trybie pełnych uprawnień – **PL**. Wygląd całego rejestru CONTROL (z wartościami domyślnymi), wraz z opisem ważnych bitów, przedstawiono na **rys. 4**.

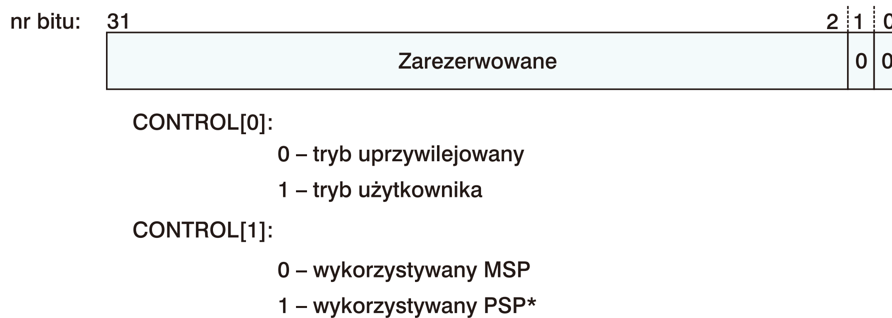
System operacyjny, który zajmuje się przełączaniem kontekstów zadań i nadzorem nad poprawnością pracy całego układu, może zmienić (pracując w przerwaniach) status uprawnień. Innymi słowy, aby przejść do trybu uprzywilejowanego (**PL**) wykonywania programu należy



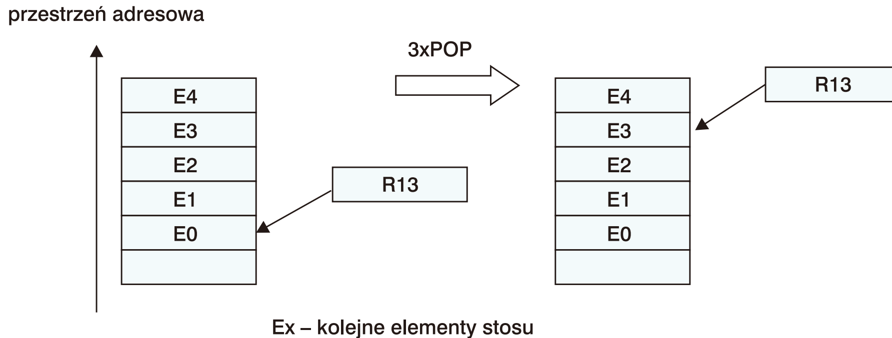
Rys. 1.



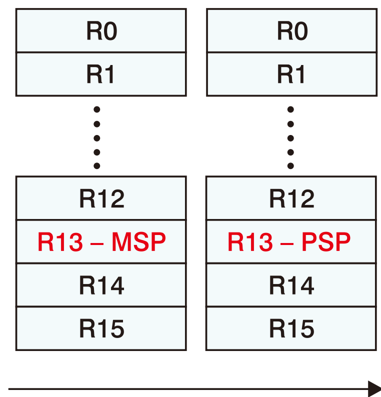
Rys. 3.



Rys. 4.



Rys. 5.



Rys. 6.

zmienić w funkcji obsługi przerwania ustawienia specjalnego rejestru kontrolnego CONTROL, czyli wyzerować najmłodszy bit.

Stos

Stos jest elementem systemu mikroprocesorowego, który pozwala przechowywać kluczowe informacje. Fizycznie jest to obszar w pamięci, który współpracuje ze specjalnym rejestrem wskaźnikowym. Na stosie informacje wpisywane są w sposób liniowy, co oznacza, że aby odczytać zagrzebane dane, należy najpierw ściągnąć ze stosu dane nowsze (**rys. 5**).

Z perspektywy programisty, dostęp do stosu realizowany jest przez użycie polecenia zdejmowania (POP) i wkładania (PUSH) na stos. Po każdej operacji PUSH rejestr wskaźnikowy (w przypadku rdzenia Cortex jest to rejestr R13) jest dekrementowany, czyli wskazuje młodszy adres. Analogicznie operacja POP powoduje inkrementację rejestru wskaźnika stosu. Mikrokontrolery STM32 są 32–bitowe, a więc stos jest inkrementowany (lub dekrementowany) zawsze o 4.

Dwa stosy, MSP i PSP

Rejestr R13 jest wskaźnikiem stosu – ściślej są to dwa bankowane wskaźniki stosu. W zależności od ustawienia drugiego bitu w specjalnym rejestrze kontrolnym CONTROL (**rys. 4**), może on wskazywać na stos systemowy MSP (*Main Stack Pointer*) lub na stos użytkownika PSP (*Process Stack Pointer*). Skoro wskaźnik stosu jest bankowany, to w danej chwili R13 może wskazywać tylko jeden stos (MSP lub PSP). Przedstawiono to na **rys. 6**. Ponadto, obsługując przerwanie mikrokontroler zawsze korzysta ze stosu MSP.

Przy okazji opisu trybów pracy mikrokontrolerów STM32 pojawiła się wzmianka o tym, że po uruchomieniu układ zawsze pracuje w trybie uprzywilejowanym. Konsekwencją tego jest, że również wskaźnik stosu R13 wskazuje na stos systemowy MSP tuż po uruchomieniu się mikrokontrolera.

Wykorzystanie dwóch stosów dodatkowo zwiększa odporność całego systemu na błędy, ponieważ jeśli aplikacja użytkownika zacznie wykonywać na stosie nieprawidłowe operacje, to nie wpłynie to negatywnie na stabilność działania układu. System operacyjny (jeżeli pracuje wykorzystując przerwania) ma swój, oddzielny stos MSP.

Mikrokontroler rozpoczynając obsługę przerwania umieszcza zawartość kluczowych rejestrów systemowych na stosie, natomiast powracając z obsługi przerwania zdejmuje te wartości, na powrót zapisując je do rejestrów. Dzięki temu przerwany proces nie traci informacji i może być dalej wykonywany bez przeszkód.

Jeśli wykorzystywany jest tylko stos MSP, to sprawa jest oczywista i mikrokontroler zachowuje się tak, jak to opisano powyżej. Wątpliwości rodzą się dopiero w przypadku, gdy wykorzystywane są oba stosy. W takiej sytuacji, do przechowywania informacji o przerywanym procesie wykorzystywany jest stos użytkownika PSP, natomiast MSP wykorzystywany jest tylko wewnątrz obsługi przerwania. Zachowanie takie przedstawiono na **rys. 7**.

Gdy mikrokontroler pracuje w trybie uprzywilejowanym, to dostęp (lub zapis) do obydwu stosów jest możliwy bezpośrednio, za pomocą instrukcji odczytu (MRS) lub zapisu (MSR) rejestru specjalnego. Odczyt i zapis stosu MSP i PSP w asemblerze może wyglądać następująco:

```
MRS r0, PSP   ; Odczyt PSP do R0
MSR PSP, r0   ; Zapis R0 do PSP
MRS r0, MSP   ; Odczyt MSP do R0
MSR MSP, r0   ; Zapis R0 do MSP
```

Powyższe instrukcje można, rzecz jasna, zapisać przy użyciu zdefiniowanych przez firmę ST funkcji, które zadeklarowano jako makra asemblerowe. W takiej sytuacji, odczyt stosu użytkownika realizowany jest za pomocą następującego fragmentu kodu:

```
Wartość_PSP = __MRS_PSP();
```

Procedura zapisu na stos PSP natomiast polega na wywołaniu odpowiedniego fragmentu kodu asemblera:

```
__MSR_PSP( (u32)NOWA_WARTOSC );
```

Analogicznie zapis i odczyt stosu MSP:

```
Wartość_MSP = __MRS_MSP();
__MSR_MSP( (u32)NOWA_WARTOSC );
```

Zastosowanie powyższych instrukcji umożliwia systemowi operacyjnemu dostęp do stosu aplikacji (PSP), a więc pełną kontrolę nad programem użytkownika.

Tryb użytkownika i PSP

Oddzielne wykorzystanie mechanizmu poziomów uprzywilejowania i modelu dwóch stosów jest oczywiście użyteczne, ale znaczne zwiększenie możliwości uzyskuje się przez połączenie obydwu. Na **list. 1** przedstawiono fragment programu, który jest odpowiedzialny za włączenie trybu użytkownika i obsługi stosu PSP. Gdy te operacje zostaną zrealizowane, to następuje wywołanie przerwania od wyjątku systemowego SVC (omówiony w dalszej części artykułu). Funkcja obsługi przerwania SVC wyłącza tryb użytkownika. Wywołanie przerwania jest zabiegiem koniecznym do zmiany poziomu uprzywilejowania. Jak było to już napisane, jego zmiana z trybu użytkownika do trybu uprzywilejowanego jest możliwa tylko w funkcji obsługi przerwania. Ilustruje to przykład funkcji obsługi przerwania SVC umieszczony poniżej:

```
void SVCHandler(void)
{
__MSR_CONTROL(0);
}
```

Instrukcja wewnątrz ciała funkcji ma za zadanie wpisać do specjalnego rejestru kontrolnego wartość 0, co odpowiada wyzerowaniu bitów CONTROL[0] i CONTROL[1], które odpowiedzialne są za aktualny poziom uprzywilejowania oraz wykorzystywany stos.

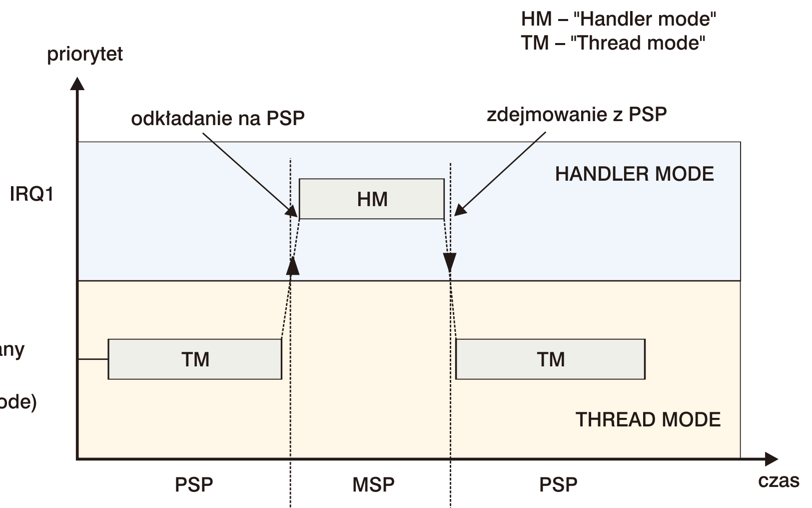
Omawiane możliwości zwiększenia stabilności pracy systemu zaimplementowano przede wszystkim z myślą o systemach operacyjnych, jednak nic nie stoi na przeszkodzie, aby wykorzystać je w aplikacjach nie pracujących pod kontrolą OS.

Wyjątki systemowe

Kontrolę wykonywanych przez STM32 zadań znacznie ułatwiają trzy wyjątki systemowe, dostępne w architekturze Cortex. Docelowym ich zadaniem jest praca pod kontrolą systemu operacyjnego, aczkolwiek w aplikacjach bez OS również można je z doskonałym Skutkiem wykorzystać do zapewnienia większej kontroli i stabilności pracy.

Do cyklicznego przełączania kontekstu zadań stworzono systemowy, 24–bitowy, timer SysTick. Jego zadaniem jest generowanie w określonych odstępach czasu przerwania, a funkcja jego obsługi może zajmować się właśnie przełączaniem kontekstu zadań. Bardziej szczegółowo zasadę działania i sposób konfiguracji timera SysTick omówiono w **EP12/08**.

Pozostałe dwa wyjątki systemowe, to SVC i PendSV. Pierwsza instrukcja przerwania jest analogiczna do instrukcji przerwania SWI, którą miały mikrokontrolery z rdzeniem ARM7. Zmiana nazwy wynika z potrzeby zabezpieczenia poprawności przenoszenia aplikacji pomiędzy rdzeniami ARM7 i Cortex M3.

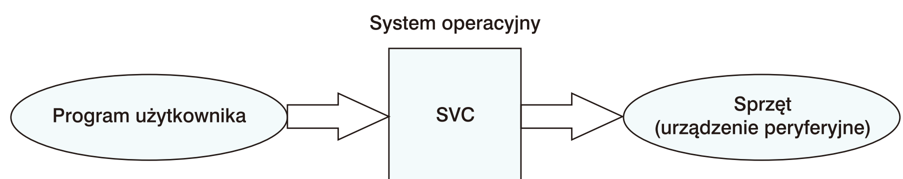


Rys. 7.

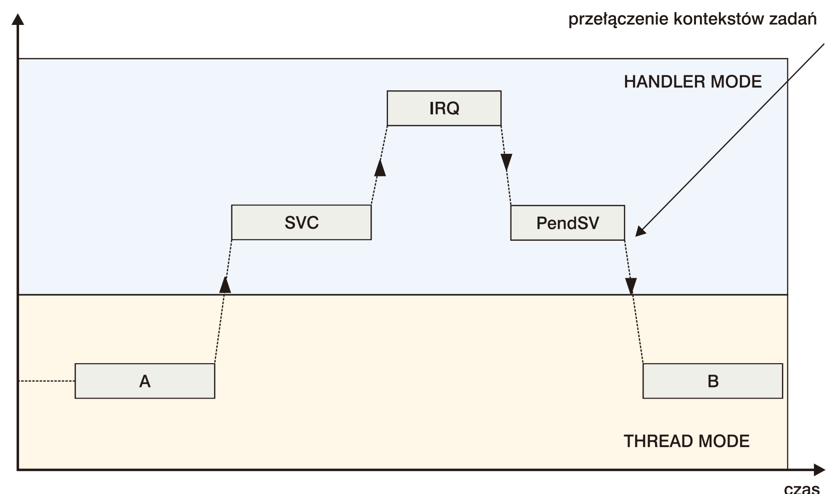
List. 1.
```
// Inicjalizacja PSP
for(Index = 0; Index < 0x200; Index++)
        PSPMem[Index] = 0x00;

__MSR_PSP((u32)PSPMem + 0x200);

// Wybor PSP i trybu uzytkownika
__MSR_CONTROL(0x03);

// Wygenerowanie SVC, powrot to trybu uprzywilejowanego
__SVC();
```

Wyjątek SVC *(System service call)*

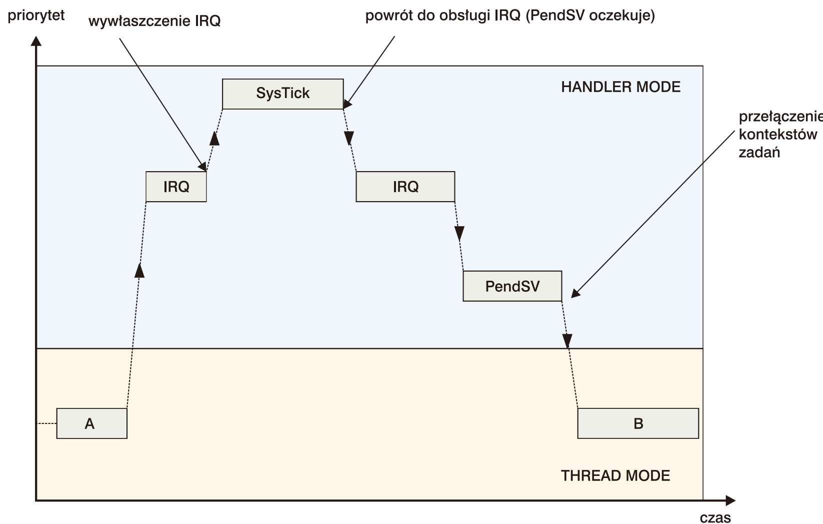
W dobrze zaprojektowanym systemie operacyjnym, uruchomiona w nim aplikacja nie może bezpośrednio odwołać się do sprzętu. Odnosząc to zdanie do konkretnego przypadku można powiedzieć, że aplikacja użytkownika nie ma możliwości operowania na portach wejścia/wyjścia inaczej, niż za pośrednictwem systemu operacyjnego. Takie ograniczenia w stosunku do aplikacji uruchamianych w systemie operacyjnym mają bardzo istotne znaczenie ze względu na ograniczone zaufanie dla programów użytkownika. W związku z powyższym musi istnieć mechanizm pozwalający na bezpieczne korzystanie ze sprzętu przez uruchomiony w systemie operacyjnym program użytkownika. Do realizacji tego zadania przeznaczono wyjątek SVC.



Rys. 8.



Rys. 9.



Rys. 10.

Jeśli program użytkownika chce skorzystać ze sprzętu, to musi wywołać funkcję SVC, a dopiero ta realizuje zadanie z użyciem wymaganego sprzętu. W dużym uproszczeniu przedstawiono to na **rys. 8**. Dzięki temu wszystkie operacje z użyciem np. urządzeń peryferyjnych są pod kontrolą systemu operacyjnego.

Wyjątek PendSV

Jak napisano wcześniej, w najprostszym systemie operacyjnym, za przełączanie kontekstów uruchomionych zadań odpowiada timer SysTick. Podczas pracy takiego systemu może powstać prosty, choć nie zawsze oczywisty problem.

Podczas realizacji zadanie (program użytkownika) może zostać zgłoszone przerwanie, które wywłaszczy dotychczasowy proces. Jeśli teraz, czyli w trakcie obsługi zgłoszonego przerwania, timer SysTick przerwie je i system operacyjny rozpocznie przełączanie kontekstów zadań, to wychodząc z funkcji obsługi przerwania od timera SysTick, **OS** będzie próbował zmusić mikrokontroler do rozpoczęcia realizacji nowego zadania. Jest to rzecz jasna zachowanie błędne, ponieważ obsługa pierwszego przerwania zostanie znacznie opóźniona. Poza tym, może zostać wygenerowany błąd.

Rozwiązaniem powyższego problemu jest zastosowanie wyjątku PendSV. Jego programowalny priorytet jest ustawiany na najniższy możliwy, dzięki czemu przerwanie to nigdy nie wywłaszczy innych obsługiwanych przerwań.

Przeanalizujmy teraz zachowanie systemu z zaimplementowaną obsługą PendSV. Załóżmy, że zadanie realizowane w systemie nie ma aktualnie nic do zrobienia. Generuje wyjątek SVC, którego zadaniem jest przygotowanie do przełączenia kontekstu zadań i wywołanie przerwania PendSV. Dopiero to ostatnie przerwanie wykonuje właściwe przełączenie kontekstów tak, że gdy mikrokontroler powraca do normalnego wykonywania programu, to wówczas podejmowane już jest wykonywanie następnego zadania.

Jeśli w trakcie przełączania kontekstów zadań w funkcji obsługi przerwania PendSV system zarejestruje inne przerwanie, to przełączanie kontekstów zostaje wstrzymane przez wywłaszczenie PendSV (pamiętajmy, że jego priorytet jest najniższy). Cały omawiany proces przedstawiono na **rys. 9**.

PendSV i SysTick

Podobnie ma się sprawa wtedy, gdy system operacyjny przygotowuje przełączanie kontekstów zadań przy pomocy przerwania od timera SysTick. W takiej sytuacji, zakładając, że przerwanie od SysTick ma wysoki priorytet, a chwilę wcześniej był obsługiwany jakiś inny wyjątek, nastąpi wywłaszczenie tego ostatniego na rzecz SysTick.

W związku z tym, że obsługa wywłaszczonego przerwania jest zdecydowanie ważniejsza od wykonywania uruchomionych w systemie zadań, to funkcja obsługi przerwania od timera SysTick (podobnie jak SVC) tylko przygotowuje system do przełączenia kontekstu zadań i generuje wyjątek PendSV. Teraz, skoro PendSV ma najniższy priorytet, mikrokontroler wraca do obsługi wywłaszczonego wcześniej przerwania. Gdy czynności z tą obsługą zostaną zakończone, to oczekujący wyjątek PendSV zaczyna być realizowany, konsekwencją czego jest przełączenie kontekstu i rozpoczęcie obsługi kolejnego uruchomionego w systemie zadania, patrz **rys. 10**.

Krzysztof Paprocki
paprocki.krzysztof@gmail.com

R E K L A M A

polecamy:
wyświetlacze
tekstowe LCD
graficzne LCD
graficzne COG
graficzne TFT
TFT FULLHD
μ'S MICROS Sp. j.
Kraków, ul. Godlewskiego 38
tel. 12 636 95 66
biuro@micros.com.pl
www.micros.com.pl

AVT414 Uniwersalna karta portów we/wy na USB
UNIVERSAL SERIAL BUS
AVT-Korporacja Sp. z o.o., 03-197 Warszawa, ul. Leszczynowa 11
tel. 022 257 84 50, fax 022 257 84 55, e-mail: handlowy@avt.pl
www.sklep.avt.pl