Sygnały zegarowe w mikrokontrolerach STM32

Wraz z pojawieniem się na rynku mikrokontrolerów STM32 należących do komunikacyjnego segmentu connectivity line, pojawiło się wiele pytań o poprawną konfigurację sygnałów zegarowych. W odpowiedzi na te pytania publikujemy artykuł ukazujący sposób konfiguracji zaawansowanego systemu generowania sygnałów taktowania w układach STM32F107.

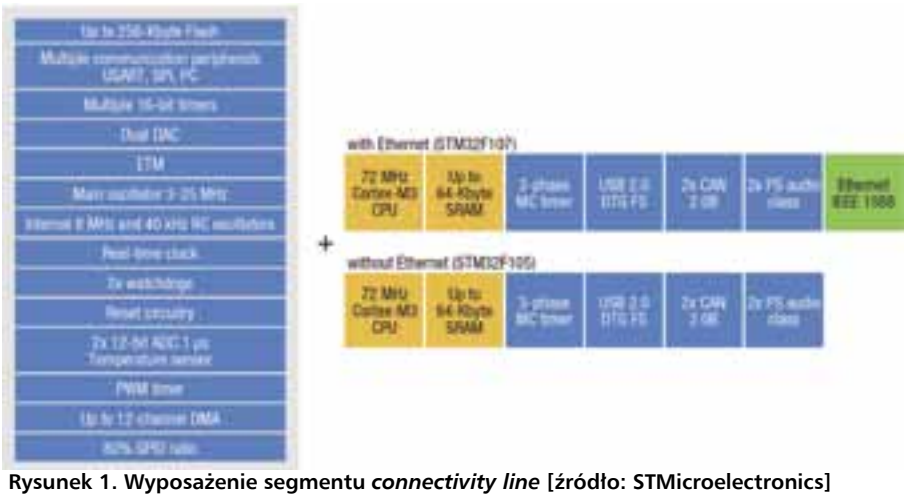
Każdy synchroniczny system cyfrowy, a do takich z pewnością należą układy mikroprocesorowe, wymaga do poprawnej pracy odpowiednich sygnałów zegarowych. W najprostszym przypadku cały system jest taktowany z jednego źródła tą samą częstotliwością.

Wraz ze wzrostem poziomu skomplikowania, a co ważniejsze, ze wzrostem skali integracji, istnieje potrzeba generowania w układach mikroprocesorowych sygnałów zegarowych o różnych częstotliwościach, determinowanych przez rodzaj zastosowanych układów peryferyjnych.

Mikrokontrolery STM32 należące od rodziny *connectivity line* są wyposażone w imponującą liczbę peryferiów komunikacyjnych, m. in. Ethernet MAC (opcjonalnie), USB OTG, I²S. Bardziej szczegółowy opis wyposażenia układów *connectivity line* przestawia **rysunek 1**. Taka mnogość urządzeń implikuje w przypadku symultanicznej ich pracy, konieczność generowania jednocześnie wielu przebiegów zegarowych o różnych częstotliwościach. W artykule przedstawiona zostanie konfiguracja sygnałów zegarowych dla aplikacji korzystającej z USB i Ethernet. Z tej racji należy wygenerować trzy sygnały zegarowe o trzech różnych częstotliwościach (dla: USB, Ethernet i rdzenia MCU).

Układy warstwy fizycznej PHY Ethernet wymagają zazwyczaj częstotliwości 25 MHz, jeśli stosowany jest interfejs MII (Media Independence Interface), lub 50 MHz, jeżeli wykorzystano interfejs RMII (Reduced MII). Najczęściej mikrokontrolery z wbudowanym sterownikiem Ethernet mogą pracować jedynie z zewnętrznym źródłem sygnału zegarowego (np. rezonator kwarcowy) o częstotliwości 25 MHz. Konstruktorzy, którzy zdecydowali się na stosowanie układu z rodziny STM32 nie są jednak skazani na stosowanie w swoich aplikacjach ethernetowych tylko 'kwarców' o tej częstotliwości. Jak pokażemy w dalszej części artykułu, architektura mikrokontrolerów firmy ST pozwala na odstępstwa od tej reguły.

Kontroler magistrali USB wymaga do poprawnej pracy taktowania z częstotliwością 48 MHz. Jeśli tylko aplikacja nie ma zbyt dużych wymagań i nie korzysta z urządzeń peryferyjnych wymagających specyficznych częstotliwości pracy, to wtedy można cały system mikroprocesorowy taktować zegarem 48 MHz. A co, jeśli owe 48 MHz to za mało dla naszej aplikacji? Tutaj na ratunek przychodzi możliwość taktowania rdzenia mikrokontrolera z innego źródła sygnału zegarowego.



Rysunek 1. Wyposażenie segmentu *connectivity line* [źródło: STMicroelectronics]

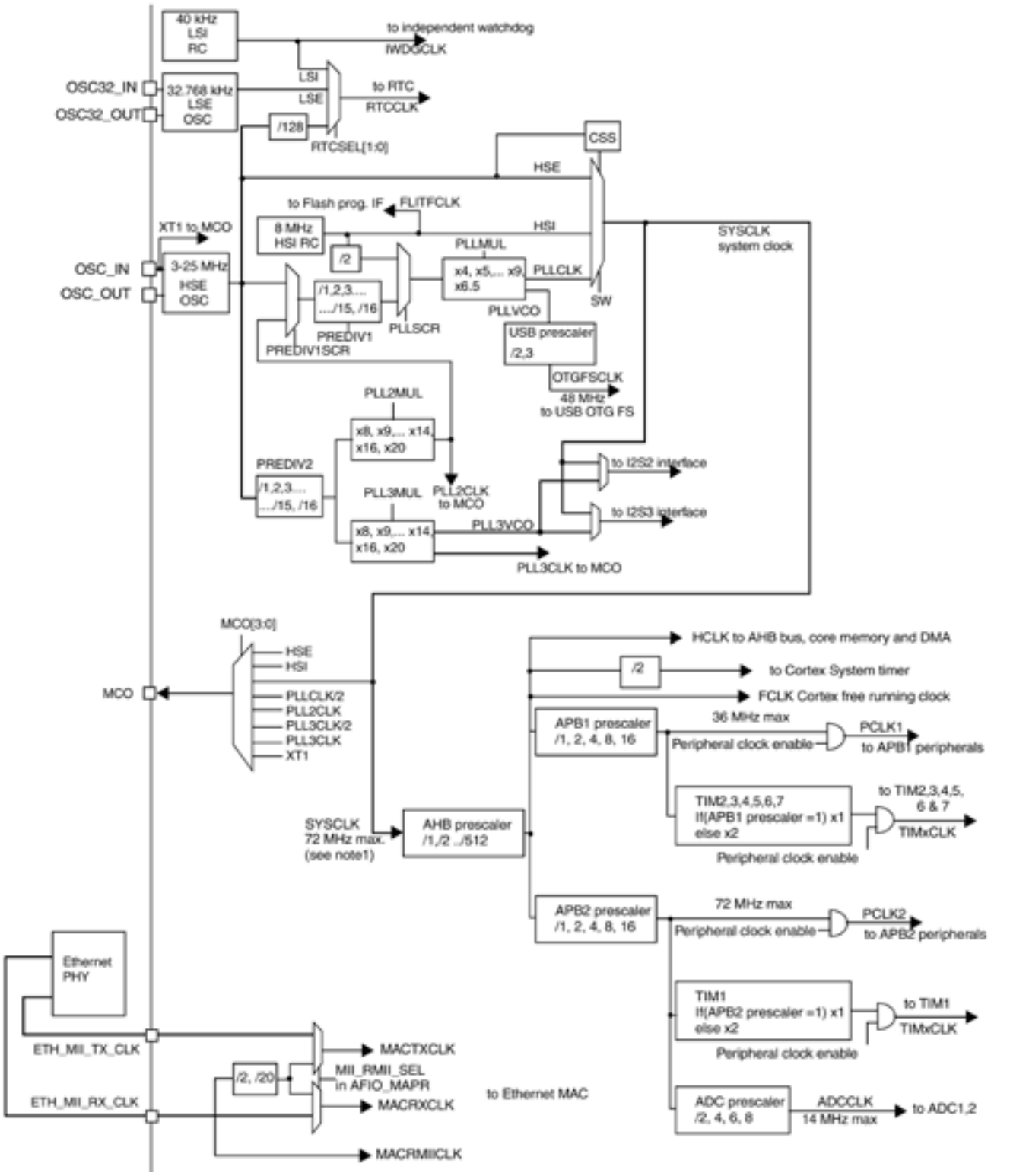
Taktowanie w STM32

Mikrokontrolery STM32F107, czyli należące do szerszej rodziny *connectiviy line*, mają wbudowane aż trzy układy PLL, które umożliwiają mnożenie wejściowego sygnału przez wiele współczynników. Do dyspozycji są również dwa układy wstępnego dzielenia sygnału oraz liczne multipleksery sygnałów zegarowych.

Uproszczony schemat bloku RCC (*Reset and Clock Control*) przedstawiono na **rysunku 2**. Zazwyczaj, ze względu na wymóg dokładności generowanego sygnału zegarowego, stosowane są zewnętrzne źródła, najczęściej rezonatory kwarcowe. Mikrokontrolery *connectivity line* mogą pracować z rezonatorami o częstotliwościach z przedziału 3...25 MHz. Dokumentacje dostarczane przez firmę ST podają, że celem rozszerzenia górnej granicy do 25 MHz było osiągnięci możliwości pracy wszystkich interfejsów komunikacyjnych po podłączeniu tylko jednego źródła sygnału zegarowego. Pokażemy, że niezwykła elastyczność układu taktowania tej podrodziny mikrokontrolerów STM32 pozwala na wykorzystanie również rezonatorów o innych częstotliwościach. Omówiona zostanie konfiguracja dla rezonatora 10 MHz z przeznaczeniem dla aplikacji z obsługą Ethernet i USB.

Konfiguracja dla Ethernet i USB

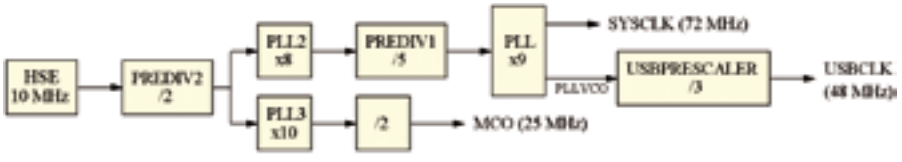
Obsługa interfejsu Ethernet i USB wymaga już dwóch różnych sygnałów zegarowych, odpowiednio 25 MHz i 48 MHz. Jeśli mikrokontroler STM32 ma dodatkowo wykorzystywać pełnię swoich możliwości, to potrzebny jest trzeci przebieg o częstotliwości 72 MHz. Proces uzyskiwania wymaganych sygnałów przedstawiono na **rysunku 3**. Jak widać droga do zamierzonego celu wydaje dość kręta,



Rysunek 2. Schemat RCC (*Reset and Clock Control*) [źródło: STMicroelectronics]

jednak jak zaraz przekonamy, nie jest aż tak źle.

Wszystkie czynności związane z konfiguracją sygnałów zegarowych mogą być wykonywane w jednej funkcji, w przedstawianym przykładzie jest to *RCC_Configuration()*, której ciało zamieszczono na **listingu 1**. Domyślnie po włączeniu zasilania mikrokontroler wykorzystuje do pracy wewnętrzny generator RC o częstotliwości około 8 MHz. Szybki oscylator zewnętrzny (HSE – *High Speed External*), zanim zostanie użyty, musi być włączony. Dokonuje tego wywołanie funkcji *RCC_HSE_Config()*. Następnie w pętli jest sprawdzany jest stan flagi informującej, która mówi o tym, czy HSE się uruchomił. Oczekiwanie na poprawne włączenie się oscylatora trwa przez 1280 (szesnastkowo 0x0500) cykli pętli. Wartość ta jest określona przez definicję *HSEStartUp_TimeOut*. W bibliotece API dostarczanej przez firmę STMicroelectronics



Rysunek 3. Proces generowania sygnałów zegarowych dla rdzenia, USB i Ethernet

dyrektywy preprocesora są umieszczone w pliku *stm32f10x.h*.

Jeśli flaga HSERDY po przekroczeniu czasu wyznaczonego przez pętlę jest nadal wyzerowana, to wtedy instrukcja warunkowa zapisuje do zmiennej statusu *HSEStatus* wartość *ERROR*. Wystąpienie takiej sytuacji skutkuje przerwaniem procesu konfiguracji sygnałów zegarowych, włączane jest jedynie taktowanie dla portów wejścia/wyjścia. W przypadku poprawnego startu HSE rozpoczyna się właściwy proces ustawiania m. in. mnożników i dzielników sygnałów zegarowych.

Pamięć FLASH wbudowana w mikrokontrolery STM32 może pracować z maksymalną częstotliwością 24 MHz, w związku z czym należy ustawić dla niej tzw. „*wait state*", czyli po prostu opóźnienie.

Dalej konfiguracja odbywa się jakby od końca. Najpierw ustalane są częstotliwości sygnałów wyznaczanych na podstawie zegara systemowego SYSCLK, a więc taktowanie wszystkiego, co jest podłączone do magistrali AHB, patrz **rysunek 1**. Wybierane jest docelowe źródło sygnału zegarowego dla podstawowych elementów systemu mikroprocesorowego, są to m. in.: rdzeń mikrokontrolera, magistrale APB1 (PCLK1) i APB2 (PCLK2), opcjonalnie również taktowanie przetworników A/C, timerów itd. Dodać tu należy, że magistrala APB1 może być taktowana sygnałem o maksymalnej częstotliwości 36 MHz, dla APB2 jest to 72 MHz.

Gdy fragment bloku RCC podłączony do magistrali AHB ma ustalone swoje parametry pracy, należy przystąpić do właściwej konfiguracji układów PLL, multiplekserów i dzielników.

Sygnał z zewnętrznego oscylatora HSE jest dzielony przez 2 w drugim bloku wstępnego dzielenia sygnału PREDIV2, w ten sposób uzyskany jest sygnał o częstotliwości 5 MHz. W następnym kroku włączany jest układ PLL3 z mnożnikiem ×10. Wyjście PLL3 może być podłączone przez multiplekser do wyprowadzenia MCO (*Microcontroller Clock Output*), ale sygnał z PLL3 może być również dzielony przez 2 zanim zostanie doprowadzony do MCO. Właśnie ta druga opcja została wykorzystana w przestawionym przykładzie, zatem na wyprowadzeniu MCO układu będzie sygnał o częstotliwości 25 MHz, czyli tyle, ile wymaga interfejs MII, który jest stosowany do podłączenia warstwy fizycznej Ethernet.

W założeniach zaznaczono, że rdzeń mikrokontrolera powinien pracować z zegarem 72 MHz. Aby to osiągnąć konfigurowany jest układ PLL2 z mnożnikiem ×8, by uzyskane w ten sposób 40 MHz następnie podzielić przez 5 w bloku PREDIV1. Dzięki temu na wejście układu PLL dostarczany jest przebieg o częstotliwości

Listing 1. Konfiguracja bloku RCC (*Reset and Clock Control*)

```
void RCC_Configuration(void)
{
    u16 StartUpCounter=0;
    uint32_t HSEStatus;
    /* Wlacz HSE */
    RCC_HSEConfig(RCC_HSE_ON);
    /* Odczekaj, az HSE sie uruchomi lub zostanie przekroczony dozwolony czas
*/
    do
    {
        StartUpCounter++;
    } while((RCC_GetFlagStatus(RCC_FLAG_HSERDY) == RESET) &&
                        (StartUpCounter != HSEStartUp_TimeOut));
    if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) == SET)
    {
        HSEStatus = SUCCESS;
    }
    else
    {
        HSEStatus = ERROR;
    }
    if (HSEStatus == SUCCESS)
    {
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
        /* 2x „waitstate" */
        FLASH_SetLatency(FLASH_Latency_2);
        /* HCLK = SYSCLK */
        RCC_HCLKConfig(RCC_SYSCLK_Div1);
        /* PCLK2 = HCLK */
        RCC_PCLK2Config(RCC_HCLK_Div1);
        /* PCLK1 = HCLK/2 */
        RCC_PCLK1Config(RCC_HCLK_Div2);
        /* PREDIV2: PREDIV2CLK = HSE / 2 = 5 MHz */
        RCC_PREDIV2Config(RCC_PREDIV2_Div2 );
        /* PLL3: PLL3CLK = (HSE / 2) * 10 = 50 MHz */
        RCC_PLL3Config(RCC_PLL3Mul_10);
        /* Wlacz PLL3 */
        RCC_PLL3Cmd(ENABLE);
        while(RCC_GetFlagStatus(RCC_FLAG_PLL3RDY) == RESET);
        /* MCO: MCO_OUT = PLL3 / 2 = 25 MHz*/
        RCC_MCOConfig(RCC_MCO_PLL3CLK_Div2);
        /* PLL2: PLL2CLK = (HSE / 2) * 8 = 40 MHz */
        RCC_PLL2Config(RCC_PLL2Mul_8);
        RCC_PLL2Cmd(ENABLE);
        while(RCC_GetFlagStatus(RCC_FLAG_PLL2RDY) == RESET);
        /* PREDIV1: PREDIV1CLK = PLL2 / 5 = 8 MHz */
        RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2,RCC_PREDIV1_Div5);
        /* PLL: PLLCLK = PREDIV1 * 9 = 72 MHz */
        RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_9);
        /* Wlasz PLL */
        RCC_PLLCmd (ENABLE);
        while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
        /* PLL zrodlem SYSCLK */
             RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
        while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08);
        /* Zrodlo taktowania USB */
        RCC_OTGFSCLKConfig(RCC_OTGFSCLKSource_PLLVCO_Div3);
        /* Wlaczenie taktowania ETHERNET i USB */
        RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ETH_MAC | RCC_AHBPeriph_ETH_MAC_Tx
|
                              RCC_AHBPeriph_ETH_MAC_Rx | RCC_AHBPeriph_OTG_FS,
ENABLE);
    }
    /* Wlacznie taktowania GPIO */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
                           RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD |
                           RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO,
ENABLE);
}
```

Listing 2. Funkcja konfiguracji timera SysTick

```
void SysTick_Configuration(void)
{
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
    /* Sygnal zegarowy rdzenia bedzie dzielony
     * przez 10000 */
    SysTick_Config(10000);
}
```

Listing 3. Obsługa przerwania od timera SysTick

```
void SysTick_Handler(void)
{
    GPIO_WriteBit(GPIOE, GPIO_Pin_7, (BitAction)
                    (1 - GPIO_ReadOutputDataBit(GPIOE, GPIO_Pin_7)));
}
```

Listing 4. Funkcja konfigurująca wyprowadzenie GPIOE7

```
void GPIO_Configuration(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

8 MHz. Teraz wystarczy już tylko ustawić mnożnik PLL na ×9, by uzyskać oczekiwane 72 MHz.

Brakuje jeszcze sygnału zegarowego dla interfejsu USB. Częstotliwość przebiegu na wyjściu PLLVCO układu PLL jest w omawianym przypadku powielana 18 razy. W efekcie uzyskuje się 144 MHz, co jest poprawną wartością (maksymalną) dla wyjścia PLLVCO. Ostatecznie ten sygnał jest dzielony przez 3 w preskalerze USB, by uzyskać w ten sposób wymagane 48 MHz.

Ostatni etap konfiguracji bloku RCC to włączenie układów peryferyjnych: USB, Ethernet i portów wejścia/wyjścia.

Z jaką częstotliwością pracuje mój mikrokontroler?

Pracując z tak zaawansowanym systemem sygnałów zegarowych może niekiedy dochodzić do stanu, że wszystko powinno działać, ale układ wydaje się nie pracować z wymaganą częstotliwością. Bezpośrednią i zarazem najprostszą metodą sprawdzenia szybkości pracy zegara systemowego jest jego wyprowadzenie na zewnątrz. Metoda ta posiada dwa ograniczenia. Pierwsze dotyczy wymogu posiadania miernika częstotliwości o dość dużym zakresie. Drugie ograniczenie daje o sobie znać przy częstotliwościach będących rzędu wielu dziesiątek megaherców. Tutaj ograniczeniem są możliwości układów sterujących wyjściami mikrokontrolera. W przypadku układów STM32, maksymalna częstotliwość przełączania wyprowadzenia MCO wynosi 50 MHz. Z odsieczą przychodzą jednak pośrednie sposoby wyznaczenia częstotliwości pracy jednostki centralnej.

Mikrokontrolery STM32 są wyposażone w systemowy timer SYSTICK. Jest to 24-bitowy układ licznikowy, liczący w dół. Może być taktowany z taką samą częstotliwością jak rdzeń, lub ośmiokrotnie mniejszą. Pomiar prędkości pracy MCU polega na podzieleniu częstotliwości systemowej za pomocą SYSTICK i zmianie stanu wyprowadzenia w przerwaniu. W ten sposób można uzyskać na wyprowadzeniu sygnał rzędu np. kilku, kilkunastu kHz, a taką wartość można już zmierzyć zwykłym multimetrem posiadającym opcję pomiaru częstotliwości, lub też drugim mikrokontrolerem, który pracuje ze znaną częstotliwością. Jest oczywistym, że taki pomiar nie będzie dokładny, ale pozwala rozstrzygnąć, czy wszystkie dzielniki i mnożniki sygnału zegarowego zostały ustawione w poprawny sposób oraz czy rezonator kwarcowy uruchomił się na odpowiedniej częstotliwości.

Konfiguracja timera SysTICK nie jest zbyt wyrafinowana, wymaga jedynie dwóch linii kodu. Funkcję ustawiającą parametry timera przedstawiono na **listingu 2**. Również funkcja obsługi przerwania od timera nie jest skomplikowana, zamieszczono ją na **listingu 3**. To jeszcze nie wszystko. Wyprowadzenie pomiarowe, w zasadzie dowolne, przykładowo niech będzie to GPIOE7 powinno zostać skonfigurowane, a dla portu GPIOE należy włączyć taktowanie. Wspomniane zadania wykonuje funkcja z **listingu 4**.

Andrzej Gawryluk