STM32
Obsługa interfejsu I²S
Dekoder audio

Zdecydowana większość interfejsów maszyna – człowiek korzysta z ludzkiego zmysłu wzroku. Niekiedy jednak zadowalające efekty komunikacji mogą być osiągnięte za pomocą słuchu. Interfejs dźwiękowy może być podstawowym do komunikacji urządzenia z operatorem lub być uzupełnieniem graficznego. W artykule przedstawiono informacje na temat obsługi magistrali I²S w roli medium łączącego mikrokontroler STM32 z dekoderem audio.

Dodatkowe informacje:
Dodatkowe informacje na temat mikrokontrolerów STM32 oraz środowiska Atollic TrueSTUDIO można znaleźć w poprzednich numerach EP, m. in:
– „Atollic TrueSTUDIO. Sposób na mikrokontrolery STM32", EP 02/2010,
– „Obsługa portów we/wy, przerwań i timerów w mikrokontrolerach STM32. Wykorzystanie funkcji API", EP 12/2008

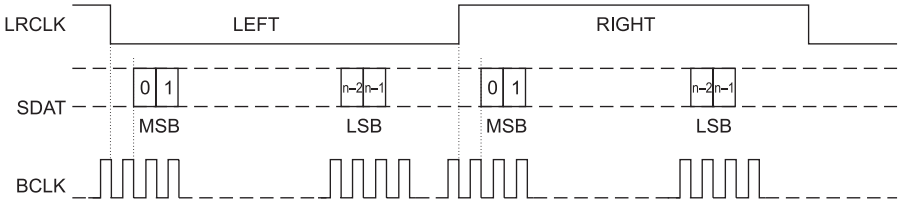
Interfejs I²S

Coraz większa liczba urządzeń, szczególnie należących do segmentu elektroniki konsumenckiej, jest wyposażona w interfejs dźwiękowy. Tor audio składa się zazwyczaj z pamięci, w której zapisane są sekwencje dźwiękowe, mikrokontrolera sterującego oraz dekodera audio. Komunikacja z pamięcią zależy od jej typu, natomiast wiele dekoderów audio może korzystać z interfejsu I²S (ang. *Inter-IC Sound*) zaprojektowanego przez firmę Philips do przesyłania dźwięku w formie cyfrowej pomiędzy układami scalonymi. Standaryzacja przesyłania sygnału audio w postaci cyfrowej miała na celu uporządkowanie i zunifikowanie interfejsów stosowanych przez różnych producentów układów scalonych.

Zadaniem interfejsu I²S jest więc przesyłanie tylko danych audio. Pozostałe sygnały, jeśli takie są wymagane, muszą być przesyłane w inny sposób. W minimalnej konfiguracji, do przesyłu informacji za pomocą magistrali I²S wymagane są tylko trzy linie:
– zegarowa SCK,
– aktywnego kanału audio LRCK,
– danych SDATA.

Norma interfejsu przewiduje przesyłanie jedynie sygnału stereofonicznego. Jeśli dekoder audio nie ma własnego generatora sygnału zegarowego, to istnieje możliwość dostarczenia go do dekodera za pomocą czwartej linii sygnałowej MCLK. Częstotliwość sygnału MCLK powinna być 256 razy większa od częstotliwości linii zegarowej SCK. Przebiegi czasowe przykładowego fragmentu komunikacji pokazano na **rysunku 1**.

Urządzenia dołączone do magistrali I²S mogą pracować w kilku trybach w zależności od tego, które jest urządzeniem nadrzędnym, a które podrzędnym. W bardziej rozbudowanych aplikacjach może być konieczne dołączenie do interfejsu kontrolera magistrali. Jego zadaniem będzie takie zarządzanie

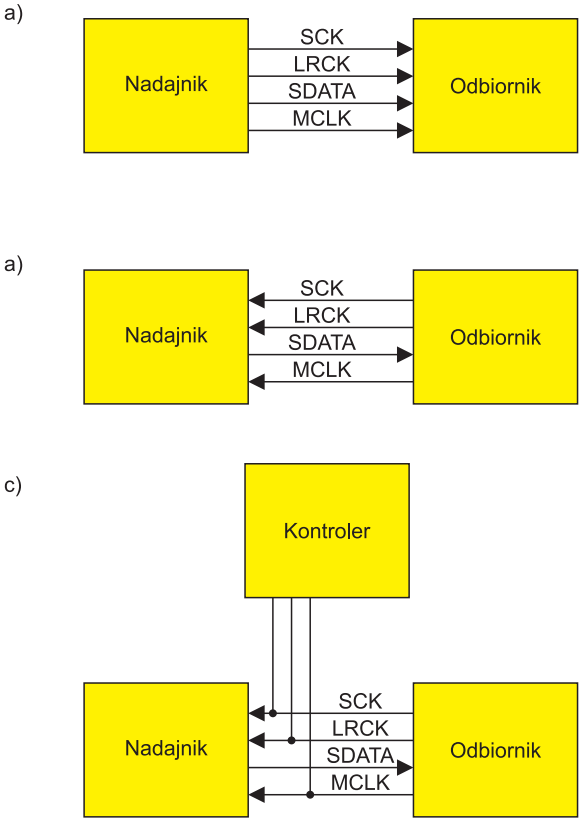


Rysunek 1. Komunikacja za pomocą interfejsu I²S

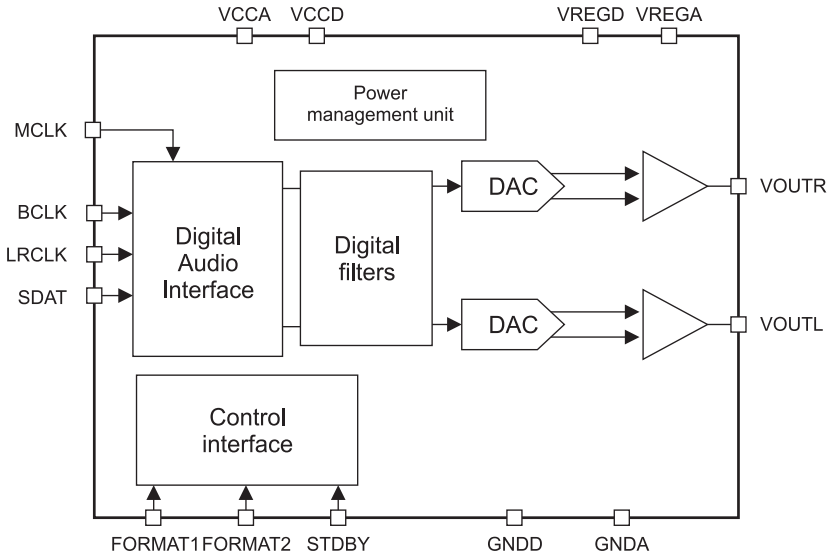
stanami linii sygnałowych, aby nie wystąpiły konflikty, np. przełączenie się dwóch urządzeń w tryb master. Możliwe tryby pracy przedstawiono na **rysunku 2**. Sygnały SCK i LRCK są zawsze generowane po stronie układu nadrzędnego. Należy zaznaczyć, że długości ramek danych nie są z góry zdefiniowane w normie. Nadajnik nie musi wiedzieć ilu bitów spodziewa się odbiornik, analogicznie odbiornik nie musi mieć informacji, ile bitów zawierają ramki wysyłane przez nadajnik. Aby osiągnąć prawidłową komunikację przy takich założeniach, jako pierwszy transmitowany jest zawsze bit najbardziej znaczący MSB. Podziału strumienia danych na linii SDATA na kanał lewy i prawy dokonuje linia aktywnego kanału LRCK. Poziom wysoki na LRCK oznacza, że przesyłana jest próbka należąca do kanału lewego, a niski, że do prawego. Częstotliwość sygnału aktywnego kanału określa zarazem częstotliwość próbkowania sygnału audio.

Narzędzia

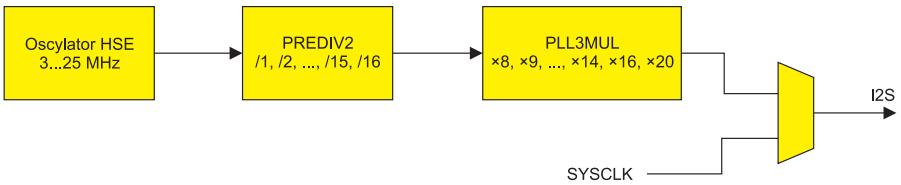
Punktem wyjścia dla każdego projektu mikroprocesorowego jest odpowiednia baza sprzętowa i środowisko programowe. Do implementacji cyfrowego toru audio wykorzystano dwa zestawy: STM32Butterfly2 i KAmodDIGA. Zestaw STM32Butterfly2 jest nowszą i zarazem bogatszą wersją zesta-



Rysunek 2. Tryby pracy urządzeń dołączonych do interfejsu I²S



Rysunek 3. Schemat blokowy dekodera audio TS4657 [Źródło: STMicroelectonics]



Rysunek 4. Źródła sygnału zegarowego dla kontrolera I²S

PREDIV2		PLL3		I2SDIV		I2SODD		MCLK	Target F_S (Hz)	Real F_S (kHz)		Error	
16-bit	32-bit	16-bit	32-bit	16-bit	32-bit	16-bit	32-bit			16-bit	32-bit	16-bit	32-bit
3	3	10	10	16	8	0	0	No	96000	96000	96000	0%	0%
6	6	20	20	32	16	0	0	No	48000	48000	48000	0%	0%
11	11	20	20	19	9	0	1	No	44100	44095.69	44095.69	0.0098%	0.0098%
2	2	10	10	72	36	0	0	No	32000	32000	32000	0%	0%
11	11	10	10	19	9	0	1	No	22050	22047.84	22047.84	0.0098%	0.0098%
4	4	20	20	144	72	0	0	No	16000	16000	16000	0%	0%

Rysunek 5. Tabela określania częstotliwości próbkowania [Źródło: STMicroelectonics]

wu STM32Butterfly, znanego z cyklu warsztatów STM32TechDays przeprowadzonych przez firmę STMicroelectronics. Najważniejsze zmiany to m. in. dodanie złącz ułatwiających dołączanie zewnętrznych modułów, interfejsu USB Host oraz złącz dla kart pamięci SD. Realizacja projektów przedstawionych w artykule wymagała rozszerzenia zestawu STM32Butterfly2 o moduł KAmodDIGA. Do przygotowania i przetestowania zaprezentowanego dalej oprogramowania użyto darmowej wersji środowiska Atollic TrueSTUDIO.

Użyty moduł „muzyczny" jest podstawową aplikacją dekodera audio TS4657. Jest to stereofoniczny, 16-bitowy przetwornik C/A audio. Schemat blokowy układu przedstawiono na **rysunku 3**. Układ jest prosty w obsłudze. Wymaga tylko jednego napięcia zasilania z przedziału 3...5,5 V. Pewnym mankamentem w przypadku montowania urządzeń prototypowych może być mała obudowa dekodera typu QFN20, choć jej montaż nawet w warunkach amatorskich jest możliwy (co nie oznacza, że jest łatwy!). Korzystanie z gotowych modułów upraszcza montaż, ponieważ zazwyczaj na płytce modułu jest wyprowadzone standardowe złącze szpilkowe.

Do prawidłowej pracy układ TS4657 wymaga sygnału taktującego wynoszącego 256×częstotliwość próbkowania sygnału audio F_S. Jak już wspomniano, przebieg zegarowy jest dostarczany jedną linią interfejsu I²S, w postaci sygnału MCLK. Moc wyjściowa układu jest wystarczająca do zadowalającego wysterowania słuchawek. Należy zwrócić uwagę, że dekoder może pracować tylko częstotliwościami próbkowania z zakresu 32...48 kHz, dlatego należy przygotowywać sekwencje audio spróbkowane z częstotliwością należącą do tego przedziału.

Kilka słów wyjaśnienia wymagają akceptowane przez dekoder formaty danych wejściowych. Typ formatu jest wybierany za pomocą stanów wyprowadzeń FORMAT1 i FORMAT2. Na płytce modułu audio (zastosowanego na potrzeby artykułu) można go wybrać zworkami JP1 i JP2. Spośród czterech trybów pracy najistotniejszym z punktu widzenia przedstawionej niżej aplikacji jest tryb z obsługą magistrali I²S, któremu odpowiadają poziomy wysokie na wyprowadzeniach FORMAT. Dla takiego ustawienia długość ramki danych jednego kanału może zmieniać się w przedziale od 16 do 24-bitów. Połączenie obydwu płytek nie powinno sprawić żadnych problemów, ponieważ obydwa moduły mają dobrze opisane wyprowadzenia sygnałów.

Ustawianie sygnałów zegarowych

Wraz ze wzrostem poziomu skomplikowania układów, a co ważniejsze, ze wzrostem ich stopnia scalenia, coraz ważniejsza staje się potrzeba zarządzania w układach mikroprocesorowych sygnałami zegarowymi o różnych częstotliwościach, zależnych od rodzaju 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. Implikuje to, w przypadku symultanicznej pracy wielu urządzeń, konieczność generowania sygnałów zegarowych o różnych a co ważniejsze – ściśle określonych częstotliwościach. W przypadku interfejsu I²S normalizacja częstotliwości próbkowania sygnału audio powoduje, że mikrokontroler powinien generować sygnał zegarowy związany liczbowo z wartością tej częstotliwości.

Układy z rodziny STM32 *connectivity line* mają wbudowane aż trzy układy PLL, dzięki którym możliwości generowania przebiegów taktujących są naprawdę duże. Źródłem sygnału zegarowego dla I²S może być zegar systemowy SYSCLK lub PLL3. Na **rysunku 4** przedstawiono bloki, które mogą brać udział w generowaniu sygnału zegarowego dla I²S. Pomimo tego, że możliwości ich konfigurowania są spore, to mnożniki i dzielniki zostały zoptymalizowane pod kątem rezonatorów kwarcowych o częstotliwościach 25 MHz i 14,7456 MHz. Zestaw STM32butterfly2 ma zamontowany rezonator o częstotliwości 14,7456 MHz.

Najprostszym sposobem wyznaczenia wszystkich mnożników i dzielników jest ich pobranie z tabeli zamieszczonej w dokumencie „*RM0008 Reference manual*", dostępnym na stronie www.st.com/stm32. Fragment tej tabeli zamieszczono na **rysunku 5**. Są w niej podane wszystkie niezbędne wartości oraz błędy wynikające z zastosowanych ustawień. W programie konfigurującym sygnały zegarowe przedstawionym na **listingu 1** zastosowano zalecane w tabeli wartości dla F_S = 44,1 kHz. Program z listingu pozwala na uzyskanie sygnału taktowania kontrolera I²S, a częstotliwości pracy poszczególnych linii interfejsu są określane już przez samo urządzenie peryferyjne, co opisano poniżej.

Konfigurowanie I²S

Sprzętowy kontroler interfejsu I²S jest wbudowany w mikrokontrolery STM32 należących do segmentu *connectivity line* oraz *high-density performance line*. Interfejs I²S zaimplementowano jako dodatkową cechę funkcjonalną interfejsu SPI, a więc oba korzystają z tych samych wyprowadzeń. Konfigurowanie wyprowadzeń dzięki bibliotece API nie jest skomplikowana i nie nastręcza trudności. Na **listingu 2** zamieszczono fragment programu, którego zadaniem jest ustawienie podstawowych parametrów pracy magistrali, a następnie włączenie kontrolera I²S.

Mikrokontroler jest układem nadrzędnego nadajnika dla dekodera audio, dlatego jako tryb pracy *I2S_Mode* należy wybrać *I2S_Mode_MasterTx*. Kontrolery I²S w układach STM32 obsługują sprzętowo pięć podstawowych formatów ramek danych: *I2S_Standard_LSB*, *I2S_Standard_MSB*, *I2S_Standard_PCMLong*, *I2S_Standard_PCMShort*, *I2S_Standard_Philips*. W aplikacjach omawianych w dalszej części artykułu wykorzystano ostatni format.

Próbki audio mogą być zapisywane przy użyciu różnej liczby bitów. Zazwyczaj w prostych interfejsach dźwiękowych na jedną próbkę przypada 16 bitów. Pracę z taką długością ramki umożliwia ustawienie pola *I2S_DataFormat* na wartość określoną definicją *I2S_DataFormat_16b*. Ponadto biblioteka API umożliwia ustawienie długości ramek na 24 lub 32 bity. Częstotliwość próbkowania jest ustalana przez modyfikację pola *I2S_AudioFreq* struktury inicjującej kontroler I²S. Dostępne częstotliwości są zapisane w pliku pomocy dostarczanym wraz z biblioteką programistyczną API. W aplikacjach opisanych poniżej używa się częstotliwości próbkowania 44,1 kHz, czyli wymienione pole struktury inicjującej powinno zostać zapisane wartością o definicji *I2S_AudioFreq_44k*. Parametr *I2S_CPOL* określa zachowanie się linii zegarowej w stanie nieaktywnym - tutaj będzie ona sprowadzana do poziomu niskiego.

Ostatnią czynnością związaną z konfigurowaniem interfejsu (wypełnianiem struktury inicjującej) jest – jeśli istnieje taka potrzeba – włączenie sygnału MCLK. Dekoder audio, zamontowany w module, do poprawnej pracy wymaga zewnętrznego sygnału taktowania, a zatem wyjście MCLK zostało aktywowane.

Wysyłanie danych może się odbywać co najmniej trzema sposobami. Najprostszy przypadek polega na ciągłym odpytywaniu kontrolera I²S, czy jego bufory nadawcze są już puste. Nie jest to jednak rozwiązanie ani efektywne, ani eleganckie. Znacznie lepszym jest użycie przerwania od kontrolera I²S. Jeszcze lepszym sposobem jest użycie DMA do bezpośredniego przenoszenia danych pomiędzy pamięcią (układem peryfe-

Listing 1. Konfigurowanie sygnałów zegarowych dla I²S

```
    /* Wlacznenie bufora dla pamieci Flash */
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

    /* dwa „wait state” */
    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);

    /* Konfiguracja PLL ------------------------------------------*/

    /* PREDIV2: PREDIV2CLK = HSE / 6 = 2.45757 MHz */
    RCC_PREDIV2Config(RCC_PREDIV2_Div6 );

    /* PLL3: PLL3CLK = (HSE / 2) * 20 = 49.1513 MHz */
    RCC_PLL3Config(RCC_PLL3Mul_20);

    RCC_PLL3Cmd(ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_PLL3RDY) == RESET);

    /* PLL2: PLL2CLK = (HSE / 2) * 20 = 49.1513 MHz */
    RCC_PLL2Config(RCC_PLL2Mul_20);

    RCC_PLL2Cmd(ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_PLL2RDY) == RESET);

    /* PREDIV1: PREDIV1CLK = PLL2 / 3 = 16.38378 MHz */
    RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2,RCC_PREDIV1_Div3);

    /* PLL1: PLLCLK = PREDIV1 * 4 = 65.535 MHz */
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_4);

    RCC_PLLCmd (ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);

    /* PLL zrodlem zegara systemowego */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

    /* Czekaj na poprawny sygnal zegarowy */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08);

    RCC_I2S2CLKConfig(RCC_I2S2CLKSource_PLL3_VCO);
```

Listing 2. Konfigurowanie kontrolera I²S

```
    I2S_InitTypeDef I2S_InitStructure;
    I2S_InitStructure.I2S_Standard = I2S_Standard_Phillips;
    I2S_InitStructure.I2S_DataFormat = I2S_DataFormat_16b;
    I2S_InitStructure.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
    I2S_InitStructure.I2S_AudioFreq = I2S_AudioFreq_44k;
    I2S_InitStructure.I2S_CPOL = I2S_CPOL_Low;

    I2S_InitStructure.I2S_Mode = I2S_Mode_MasterTx;
    I2S_Init(SPI2, &I2S_InitStructure);

    SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_TXE, ENABLE);

    I2S_Cmd(SPI2, ENABLE);
```

Listing 3. Skrypt Octave/Matlab tworzący tablicę próbek w formacie pliku tekstowego

```
function wav2txt;

[fid] = fopen(,samples.txt’, ,w’);
[y, Fs, Nbits] = wavread(,input.wav’);
z = (y+1) * 32768;

len = length(z)
Fs
Nbits

for i=1:len
    fprintf(fid, ,0x%X, ,, z(i));
    rows = mod (i, 16);
    if ~rows
        fprintf(fid, ,\r\n’);
    end;
end;
plot(z);
fclose(fid);
```

Listing 4. Funkcja NVIC_Configuration()

```
void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    NVIC_InitStructure.NVIC_IRQChannel = SPI2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

Listing 5. Obsługa przerwania od kontrolera I²S

```
void SPI2_IRQHandler(void)
{
    /* Sprawdzenie zrodla przerwania */
    if (SPI_I2S_GetITStatus(SPI2, SPI_I2S_IT_TXE) == SET)
    {

        /* Wyslij dane przez I2S2 */
        SPI_I2S_SendData(SPI2, (I2S2_Buffer_Tx[TxIdx] - 32768));
        TxIdx++;
    }

    TxIdx &= (table_size + 1)
}
```

ryjnym), a interfejsem I²S. Ostatni przypadek jest najbardziej efektywny pod względem użycia mocy obliczeniowej jednostki centralnej. Sprawa się nieco komplikuje, jeżeli dane odczytywane z pamięci mają być modyfikowane. W programie zaprezentowanym na **listingu 2** skorzystano z drugiej opcji, a więc przyjęto, że wysyłanie danych następuje po przerwaniu od pustego bufora nadawczego kontrolera I²S. Na koniec konfigurowania I²S trzeba jeszcze włączyć urządzenia przez wywołanie funkcji *I2S_Cmd()*. Nie należy także zapominać o odpowiednim konfigurowaniu wyprowadzeń oraz włączeniu niezbędnych sygnałów zegarowych.

Odtwarzacz audio

Dane audio mają być zapisane w pamięci Flash mikrokontrolera, a więc należy je wyłuskać np. z pliku *.wav. Prosty skrypt, odczytujący dane z pliku wav i zapisujący je do pliku tekstowego przedstawiono na **listingu 3**. Skrypt można uruchomić w programie Octave lub Matlab. Efektem jego wykonania jest utworzenie w katalogu roboczym pliku tekstowego z ułożonymi w tablicę wartościami próbek w zapisie szesnastkowym. Wystarczy taką tablicę skopiować do programu przeznaczonego dla mikrokontrolera jako tablicę stałych wartości.

W aplikacji korzystano z mechanizmu przerwań, dlatego przed ich użyciem i włączeniem należy skonfigurować sterownik przerwań NVIC. Odpowiada za to funkcja *NVIC_Configuration()*, której kod zamieszczono na **listingu 4**. Aby przerwanie działało prawidłowo, należy dodać do pliku *stm32f10x_it.c* funkcję jego obsługi. Plik *stm32f10x_it.c* jest automatycznie tworzony i dodawany do projektu przez środowisko Atollic TrueSTUDIO. Nazwę funkcji można odczytać z tablicy wektorów przerwań, w tym przypadku jest to SPI2_IRQHandler(). Kod omawianej funkcji zamieszczono na **listingu 5**.

Po uruchomieniu programu w mikrokontrolerze sekwencja audio zapisana w pamięci Flash będzie odtwarzana dopóty, dopóki będzie włączone napięcie zasilania.

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