Mikrokontrolery STM32 z bliska

Jeszcze nie tak dawno możliwość zastosowania w swoich aplikacjach mikrokontrolerów 32–bitowych nawet nie przeszła przez myśl większości konstruktorom. Sytuacja uległa zmianie wraz z pojawieniem się na rynku procesorów z rdzeniami ARM7 i ARM9, które zadomowiły się u nas już na dobre. Warto się jednak też zainteresować kolejnym produktem spod znaku firmy ARM – rdzeniem Cortex–M3, który od niedawna jest dość intensywnie promowany. Obecnie można już zdobyć bez większych problemów 32–bitowe mikrokontrolery z rdzeniem Cortex–M3, produkowane m.in. przez firmę STMicroelektronics.

Mikrokontrolery 32–bitowe są już na tyle tanie, że z powodzeniem zaczynają zastępować swoich 8–bitowych braci. Może się to wydawać nieco dziwne, że ze swej natury bardzo silne pod względem obliczeniowym jednostki 32–bitowe są czasami wykorzystywane do prozaicznych zadań. Zmuszenie takiego mikrokontrolera np. do migania diodą LED nie jest proste. Uzasadnieniem takiego stanu rzeczy może być fakt, iż projekty często są systematycznie rozwijane, a zaimplementowanie systemu operacyjnego na platformie 32–bitowej jest stosunkowo proste i efektywne. Ponadto, niekiedy nawet w prostych, na pierwszy „rzut oka", projektach pojawia się „wąskie gardło", którego zlikwidowanie zawsze wymaga czasu, a co za tym idzie następuje przedłużenie terminu wykonania projektu. Widać zatem, że warto poświęcić nieco czasu na opanowanie mikrokontrolera 32–bitowego. Dzięki temu, w ogólnym przypadku, czas potrzebny na wykonanie projektu ulega znacznemu skróceniu. Jeszcze jedną ważną zaletą przemawiająca za stosowaniem mikrokontrolerów 32–bitowych jest fakt, że bardzo często producenci tych układów dostarczają do nich własne biblioteki API. Umożliwia to konstruktorowi skupienie się na istocie problemu, gdyż dzięki tym bibliotekom przechodzimy już na pewien poziom abstrakcji, w większości przypadków nie ma potrzeby manipulowania rejestrami procesora, zajmują się tym odpowiednie funkcje API.

Architektura Cortex–M3

Rdzeń Cortex–M3 jest historycznie pierwszym z rodziny Cortex–ów opracowanych przez firmę ARM (Advanced RISC Machines) w 2006 roku. Przez lata swojej działalności firma ta stworzyła kilka 32–bitowych architektur:

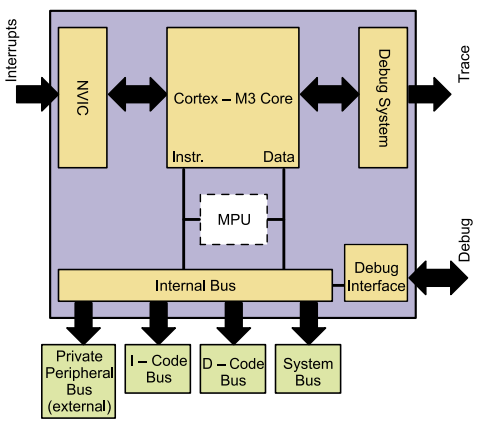
- ARMv4T – np. procesor ARM7TDMI, ‚T' oznacza, że mamy tu dostęp do 16–bitowego trybu pracy „Thumb",
- ARMv5E – m.in. dla zastosowań multimedialnych,
- ARMv6 – procesory ARM11, stworzone np. na rynek PDA i telefonów 3G,
- ARMv7 – rodzina procesorów Cortex

Rdzenie Cortex zostały podzielone na trzy grupy:

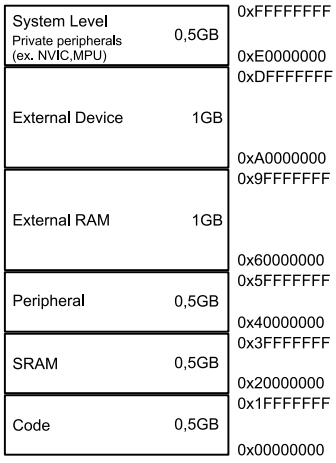
- A – grupa przeznaczona dla wymagających aplikacji z systemami operacyjnymi, takimi jak Symbian, Linux i Windows Embedded, które wymagają dużych mocy obliczeniowych, obsługi pamięci wirtualnej z MMU (*Memory Management Unit*), czy też Javy,
- R – grupa stworzona z myślą o systemach czasu rzeczywistego, w których czas reakcji jest krytyczny (np. ABS),
- M – grupa oferująca bardzo korzystny stosunek ceny do możliwości, jest przeznaczona do zastosowań konsumenckich i przemysłowych.

Na **rys. 1** jest przedstawiono w uproszczeniu budowę Cortex–M3. Architektura ARMv7 opiera się o listę instrukcji Thumb–2, która wspiera zarówno 16–, jak i 32–bitowe operacje. Nie ma zatem potrzeby, tak jak to było w przypadku rdzeni ARM7 i ARM9, przełączania się pomiędzy 16–bitowym trybem Thumb, a 32–bitowym ARM. Lista Thumb–2 ma również pewne rozszerzenia, np. sprzętowy dostęp dzielenia, problemem jest jednak to, że Cortex–M3 nie są kompatybilne ze swoimi poprzednikami. Niestety oznacza to, że nie można bezpośrednio uruchomić programu przeznaczonego np. dla ARM7 na Cortex–M3. Jednym z powodów takiego stanu rzeczy jest to, że te mikrokontrolery z tym rdzeniem nie mogą wykonywać instrukcji z trybu pracy ARM.

Rdzeń Cortex–M3 ma architekturę harvardzką, co oznacza, że magistrale danych i instrukcji są oddzielone. Takie podejście pozwala na dostęp do pamięci programu i pamięci danych w tym



Rys. 1. Uproszczona budowa procesorów Cortex–M3



Rys. 2. Przestrzeń adresowa mikrokontrolerów Cortex–M3

samym czasie, pamięci te współdzielą tę samą przestrzeń adresową (**rys. 2**).

Architektura harvardzka pozwoliła na znaczne zwiększenie efektywności, ponieważ dostęp do pamięci danych nie oddziałuje zbytnio na wykonywanie programu. Rdzenie Cortex mogą używać kodowania zarówno Big endian jak i Little endian. W przypadku bardziej zaawansowanego i wymagającego systemu rdzenie z tej rodziny mogą być wyposażone w jednostkę ochrony pamięci (MPU – *Memory Protection Unit*). Architektura Cortex'ów zawiera w swej strukturze już zaimplementowane bloki do debugowania z obsługą pułapek (*breakpoint*) i *watchpointów*.

Rejestry

Rdzeń Cortex–M3 posiada szesnaście podstawowych rejestrów (R0...R15), przy czym trzynaście z nich, od R0 do R12, są rejestrami ogólnego przeznaczenia (*General Purpose Register*). Trzeba jednak zaznaczyć, że niektóre instrukcje 16–bitowe mogą operować tylko na ośmiu młodszych rejestrach z zakresu R0...R7. Rejestr R13 jest wskaźnikiem stosu. Jest on podzielony na dwa oddzielne, bankowane rejestry, co oznacza, że w danym momencie jest widoczny tylko jeden. Są to:

Tab. 1. Priorytety poszczególnych wyjątków systemowych i przerwań

Numer	Wyjątek	Priorytet
1	Reset	–3
2	NMI	–2
3	Hard Fault	–1
4	MemManage Fault	Programowany
5	Bus Fault	Programowany
6	Usage Fault	Programowany
7...10	Reserved	–
11	SVCall	Programowany
12	Debug Monitor	Programowany
13	Reserved	–
14	PendSV	Programowany
15	SYSTICK	Programowany
16–255	External Interrupt	Programowany

- MSP (*Main Stack Pointer)* – domyślny rejestr używany przez przerwania i jądra systemów operacyjnych,
- PSP (*Process Stack Pointer)* – używany przez aplikację użytkownika.

Pozostałe rejestry: R14 (tzw. *Link Register*) – zawiera adres powrotu z podprogramu; R15 (liczniki rozkazów) – zawiera aktualny adres instrukcji, może być zapisywany w celu sterowania wykonywaniem programu.

Oprócz wyżej wymienionych rejestrów rdzeni Cortex–M3 posiadają oczywiście też rejestry specjalne: *Program Status Register*, *Interrupt Mask Register*, *Control Register*. Te rejestry sterują pracą procesora i sposobem wykonywania kodu, mogą być zmieniane tylko za pomącą instrukcji specjalnych, nie mogą być modyfikowane podczas normalnej pracy.

Tryby pracy

Rdzenie Cortex mają dwa możliwe tryby pracy o różnych stopniach uprzywilejowania:

- tryb uprzywilejowany,
- tryb użytkownika.

W rdzeniach Cortex program może być wykonywany w trybie tzw. uprzywilejowanym lub nieuprzywilejowanym (użytkownika). Po zerowaniu procesor zawsze rozpoczyna pracę w trybie uprzywilejowanym, a do jego zmiany służy najmłodszy bit rejestru specjalnego CONTROL. Gdy program jest wykonywany w drugim trybie (nieuprzywilejowanym), niektóre zasoby są niedostępne, dzięki czemu aplikacja użytkownika nie ma dostępu do kluczowych, ze względu na niezawodność, elementów systemu, takich jak niektóre krytyczne obszary pamięci. Dzięki takiemu rozwiązaniu łatwiej jest tworzyć na tej platformie niezawodne systemy operacyjne. Kernel pracujący w trybie uprzywilejowanym ma dostęp do wszystkich zasobów, natomiast aplikacja uruchamiana w systemie pracuje w bezpiecznym trybie nieuprzywilejowanym (użytkownika).

Przestrzeń adresowa

Przestrzeń adresowa Cortex'ów jest podzielona na segmenty, m.in.: pamięci programu, pamięci SRAM i pamięci zewnętrznej RAM, urządzeń peryferyjnych itd. Mapa pamięci została przedstawiona wcześniej na rys. 2.

Tak zdefiniowana przestrzeń adresowa jest zoptymalizowana pod kątem szybkości działania i prostoty implementacji. Nie zapominajmy, że firma ARM sprzedaje tylko bloki IP (*Intellectual Property*), natomiast to dopiero producent mikrokontrolerów fizycznie realizuje je w krzemie.

Mimo, że przestrzeń adresowa jest już wcześniej zdefiniowana, istnieje oczywiście możliwość odmiennego jej użycia. Przykładem może być umieszczenie programu w pamięci SRAM, zewnętrznej pamięci RAM, jak również np. umieszczenie danych w przestrzeni adresowej pamięci programu.

Sterownik przerwań NVIC

Rdzeń Cortex–M3 obsługuje piętnaście przerwań (wyjątków) systemowych i do 240 przerwań zewnętrznych – to ile tych przerwań będzie obsługiwanych zależy od producenta mikrokontrolera. Większość wyjątków systemowych i wszystkie przerwania zewnętrzne mogą mieć programowo ustalany priorytet, co zostało przedstawione w **tab. 1.** Na uwagę zasługują trzy pierwsze wyjątki, ich priorytety mają wartość ujemną dla podkreślenia tego, że to one są w systemie najważniejsze, przy czym priorytet najwyższy z możliwych ma wartość –3.

Cortex posiada wbudowany sprzętowy sterownik przerwań, tzw. *Nested Vectored Interrupt Controller* (NVIC). Jak wiadomo, we wcześniejszych procesorach ARM7/ARM9 dostępne były dwa rodzaje przerwań, a ich obsługa była dość skomplikowana. Inżynierowie z firmy ARM zauważyli to, i w rodzinie Cortex mamy już do dyspozycji NVIC, który znacznie upraszcza obsługę przerwań. Jego możliwości to:

- Zagnieżdżona obsługa przerwań – wszystkie zewnętrzne przerwania i większość przerwań systemowych mogą mieć ustawiane różne priorytety. W momencie wystąpienia przerwania, NVIC porównuje priorytet tego przerwania z priorytetem obecnego zadania. Jeżeli priorytet nowego przerwania jest wyższy od dotychczasowego, wtedy nowe zadanie o wyższym priorytecie jest wykonywane w pierwszej kolejności.
- Obsługa wektorów przerwań – NVIC oferuje również wsparcie dla wektorów przerwań. W momencie, kiedy przerwanie zostaje przyjęte do realizacji, adres funkcji obsługi przerwania (ISR) jest pobierany z wektora w pamięci. Nie ma potrzeby programowego wyznaczania adresu ISR, dzięki temu czas potrzebny na obsłużenie przerwania jest krótszy.
- Dynamiczne zmiany priorytetów przerwań – priorytety przerwań mogą być zmieniane programowo w trakcie pracy mikrokontrolera. Przerwanie, które jest w danym momencie obsługiwane zostaje zablokowane przed zmianą priorytetu dopóki nie opuści ISR, zatem nie występuje niebezpieczeństwo wielokrotnego obsłużenia tego samego przerwania podczas zmiany priorytetu.
- Zmniejszenie opóźnień związanych z przerwaniami – rdzenie Cortex–M3 zostały zoptymalizowane pod kątem możliwie najkrótszych opóźnień czasowych podczas obsługi przerwania. Jedną z podstawowych czynności mających na celu skrócenie tego czasu jest automatyczne zapisywanie i przywracanie kontekstu zadania, czyli zawartości kluczowych dla tego zadania rejestrów.
- Maskowanie przerwań – przerwania i wyjątki (*exceptions*) mogą być maskowane na podstawie ich priorytetów lub maskowane całkowicie poprzez odpowiednie użycie re-

jestrów maskujących. Ma to szczególne znaczenie dla zadań, w których wykonaniu czas jest parametrem krytycznym. W tym przypadku takie przerwania będą obsługiwane bez wywłaszczenia.

Lista rozkazów

Jedną z najważniejszych zalet procesorów z rdzeniem Cortex–M3 w stosunku do swoich poprzedników jest zdolność tych pierwszych do obsługi instrukcji 16- i 32-bitowych bez jakichkolwiek dodatkowych zabiegów (lista instrukcji Thumb-2). Takie rozwiązanie pozwoliło na zmniejszenie objętości kodu oraz na zwiększenie efektywności. Jak już było napisane poprzednie wersje procesorów ARM mogły pracować w dwóch trybach (16-bitowym Thumb i 32-bitowym ARM). W trybie pracy Thumb wszystkie instrukcje są 16-bitowe, zatem w stosunku do trybu ARM uzyskuje się znacznie mniejszy objętościowo kod. Problem polega na tym, że niektóre bardziej skomplikowane zadania trudno zaimplementować w tym trybie, często wymagają one również większej liczby instrukcji. W ten sposób doszliśmy do sedna sprawy, mianowicie w większości aplikacji implementowanych w mikrokontrolerach z rdzeniami ARM7/ARM9 muszą być one przełączane pomiędzy trybami Thumb i ARM, co oczywiście nie pozostaje bez wpływu na efektywność pracy. Tej wady pozbawione są procesory Cortex–M3 współpracujące z nową listą rozkazów Thumb–2, co sprawia, że używanie ich we własnych konstrukcjach jest prostsze.

W celu możliwie jak najlepszego wykorzystania możliwości drzemiących w zestawie instrukcji Thumb–2, została stworzona dedykowana do tego celu odmiana asemblera – *Unified Assembler Language* (UAL), czyli zunifikowany język asemblera. Dzięki temu operowanie instrukcjami 16– i 32–bitowymi stało się dużo prostsze i bardziej przejrzyste.

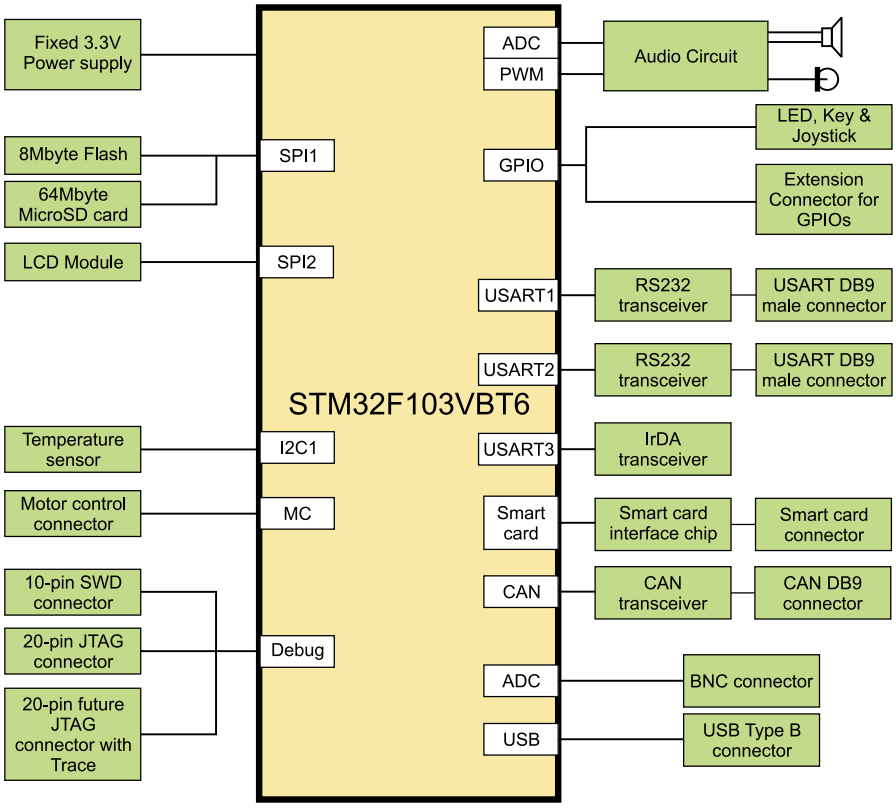
W zestawie instrukcji Thumb–2 niektóre operacje mogą być wykonywane zarówno jako 16–, jak i 32–bitowe. Przykładem może być dodawanie wartości liczbowej do rejestru:

```
ADDS R0, #5      ;Zostanie użyta 16-
bitowa instrukcja Thumb (domyślna,
dla mniejszego rozmiaru kodu)
ADDS.N R0, #5    ;Zostanie użyta
również 16 – bitowa instrukcja (N =
Narrow)
ADDS.W R0, #5    ;Tutaj będzie
zastosowana 32-bitowa instrukcja
Thumb-2 (W = Wide)
```

Bez użycia literki 'W' lub 'N' kompilator wybierze domyślny rodzaj instrukcji, jednak z reguły będzie to 16–bitowy Thumb.

Sprzęt – płyta ewaluacyjna

Złożoność problemów, z jakimi spotykamy się na co dzień w praktyce inżynierskiej nieustannie rośnie. Jeszcze nie tak dawno jeden człowiek był w stanie w sensownym czasie zaprojektować i wykonać zestaw ewaluacyjny



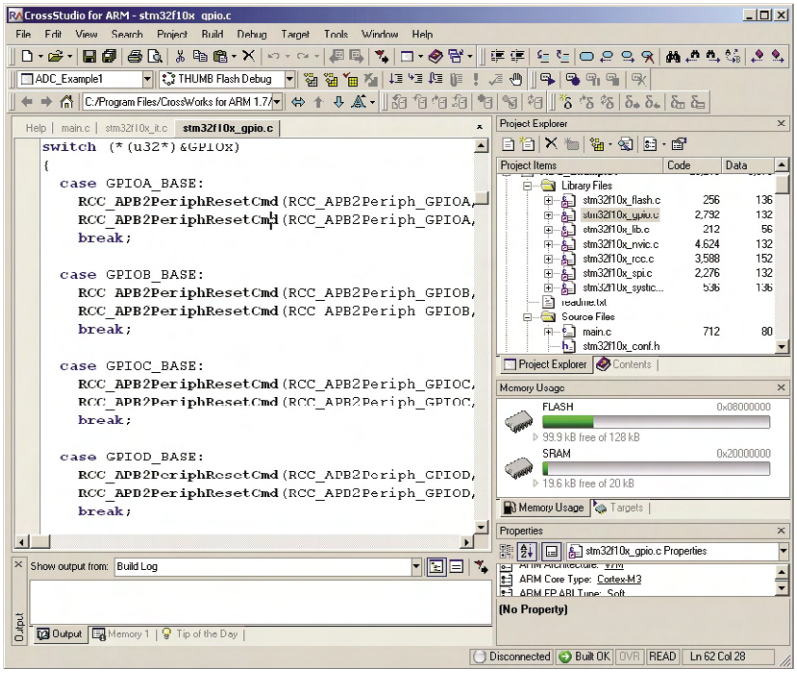
Rys. 3. Schemat blokowy zestawu ewaluacyjnego STM3210B–EVAL

oraz uruchomić na nim swój projekt. Aktualnie, jeśli ktoś zamierza zająć się na poważnie programowaniem mikrokontrolerów, to wykonywanie własnego zestawu uruchomieniowego mija się z celem. Powodem takiego stanu rzeczy jest fakt, iż zaprojektowanie dobrej uniwersalnej płyty prototypowej nie jest zadaniem łatwym, ponadto technologia wykonania takiego zestawu jest też dość zaawansowana (nierzadko są to płyty wielowarstwowe). Zatem na zaprojektowanie i wykonie zestawu trzeba poświęcić dużo czasu, który powinien być przeznaczony na uczenie się architektury mikrokontrolera i pisanie przykładowych aplikacji. Takie podejście, promujące gotowe zestawy, jest uzasadnione jeszcze jednym faktem, mianowicie od jakiegoś czasu producenci mikrokontrolerów oferują zestawy startowe w dość atrakcyjnych cenach. Do naszych celów wykorzystamy zestaw ewaluacyjny STM3210B–EVAL, który jest wyposażony w mikrokontroler STM32F103 produkcji STMicroelectronics.

Zestaw ewaluacyjny STM3210B–EVAL jest kompletną platformą umożliwiającą uruchamianie prototypowego oprogramowania i sprawdzania jego działania. Schemat blokowy przedstawiono na **rys. 3**. „Sercem" zestawu jest mikrokontroler STM32F103, będący przedstawicielem grupy Performance Line. Jest wyposażony w 128 kB pamięci Flash i 20 kB pamięci SRAM. Na pokładzie płyty ewaluacyjnej znajdziemy wszystko, co jest niezbędne do wykorzystania możliwości tego mikrokontrolera, a zatem mamy do dyspozycji między innymi interfejs USB2.0, CAN2.0, po dwa kanały I^2C i SPI, IrDA, dwa USART–y, wejście przetwornika A/C, gniazdo SMARTCARD oraz gniazdo kart MicroSD. Producent zadbał również o możliwość sterownia silnikami bezszczotkowymi, do czego jest przeznaczone złącze szpilkowe.

Mikrokontroler jest taktowany przebiegiem z oscylatora wykorzystującego rezonator 8 MHz, jednak w razie potrzeby można go bez najmniejszego problemu zamienić na inny, ponieważ został umieszczony w specjalnej podstawce. Sygnał zegarowy może być powielony przy pomocy wewnętrznej pętli PLL do maksymalnej wartości, z jaką mogą być taktowane mikrokontrolery STM32, czyli 72 MHz. Producent zadbał również o możliwość konfiguracji sposobu uruchamiania (*bootowania*) systemu. Dostępne są opcje uruchamiania: z wewnętrznej pamięci Flash, wewnętrznej pamięci RAM oraz z zewnętrznej pamięci RAM. Do wyboru tych opcji służą przełączniki SW1 i SW2. Dzięki temu mamy możliwość implementowania aplikacji, które zajmują objętościowo mniej kodu wprost do pamięci RAM, a co za tym idzie nie zużywamy pamięci Flash, która ma ograniczoną żywotność, określaną przez producenta na minimum 10000 cykli.

Jeśli dysponujemy własnymi modułami peryferyjnymi, to można je łatwo dołączyć do zestawu ewaluacyjnego. Na płycie są wyprowadzone dwa dwurzędowe złącza szpilkowe (2x50pin) służące do podłączenia ewentualnej płytki dodatkowej (tzw. *Daughter board*), zatem możliwość rozbudowy zestawu o moduły zwiększające jego funkcjonalność jest znacznie ułatwiona.



Rys. 4. Okno środowiska CrossStudio

Do nawiązania interakcji z człowiekiem służą: wyświetlacz LCD, przyciski nawigacyjne (joystick), cztery diody LED oraz system audio. Dołączony do zestawu wyświetlacz LCD o rozdzielczości 240x320 pikseli jest zamontowany na dodatkowej płytce umożliwiającej jego usunięcie, zastąpienie innym lub wymianę. Matrycą steruje układ ILI9320, z którym nasz mikrokontroler porozumiewa się poprzez magistralę SPI.

Interfejs graficzny oparty o wyświetlacz LCD może być wsparty przez interfejs dźwiękowy, ponieważ mamy możliwość zarówno odtwarzania dźwięków za pomocą głośniczka lub słuchawek, jak i je nagrywania przy użyciu zamontowanego na płytce mikrofonu. Głośność odtwarzania i czułość nagrywania mogą być regulowane za pomocą zamontowanych na płytce potencjometrów. Do nawigacji w bardziej złożonym systemie mamy do dyspozycji przyciski i joystick, który rozróżnia cztery kierunki wychylenia oraz nacisk (tak jak zwykły przycisk). Dzięki niemu poruszanie się na przykład po wielopoziomowym menu nie sprawia trudności, co ma istotne znaczenie przy obsłudze rozbudowanych aplikacji.

Debugowanie i programowanie mikrokontrolera można przeprowadzić przy użyciu wyprowadzonego złącza w standardzie JTAG.

Płytę ewaluacyjną STM3210B–EVAL można zasilić na trzy sposoby: za pomocą dedykowanego zasilacza dołączonego do zestawu, za pośrednictwem złącza USB lub też z dodatkowej płytki *Daughter board*. Ponadto mamy możliwość zasilenia tej ostatniej przy pomocy płyty ewaluacyjnej, jednak w tym przypadku należy pamiętać, aby *Daughter board* nie posiadała własnego zasilania. W tym trybie konfiguracji dokonujemy za pomocą zworki JP4, zworką JP11 natomiast decydujemy, czy chcemy korzystać z zasilania bateryjnego. Dioda LED LD5 sygnalizuje, czy zasilanie jest poprawne.

Oprogramowanie – środowisko CrossWorks

Wielu producentów oprogramowania stwarza możliwości współpracy swoich produktów z programatorem, problemem jest jednak to, że są to w przytłaczającej większości programatory specyficzne, dedykowane dla danego środowiska, a przy tym ich cena jest dość wysoka.

Warto zwrócić uwagę na środowisko CrossWorks firmy Rowley współpracujące z kilkoma różnymi programatorami. Pakiet CrossWorks jest oparty na kompilatorze GCC, ale IDE jest produktem firmy Rowley. Ogromnym plusem tego środowiska jest to, że jest ono również rozpowszechniane na licencji 30–dniowej, bez ograniczeń funkcjonalnych. Polityka licencyjna twórców CrossWorksa jest bardzo przyjazna dla osób korzystających z niego w celach edukacyjnych, jednostanowiskową licencję niekomercyjną możemy już mieć za £75. Wygląd środowiska został przedstawiony na **rys. 4.**

Czym programować?

Mikrokontroler zamontowany na płytce ewaluacyjnej można programować na kilka sposobów: przez port szeregowy UART wykorzystując *bootloader*, przez port USB (za pomocą aplikacji DfuSe) oraz za pomocą JTAG–a. Jakkolwiek dwa pierwsze rozwiązania są stosunkowo proste, ponieważ nie wymagają dodatkowego wsparcia sprzętowego, to ich możliwości są ograniczone. Programowanie przez interfejs JTAG oprócz swobody daje również możliwość debugowania mikrokontrolera w systemie.

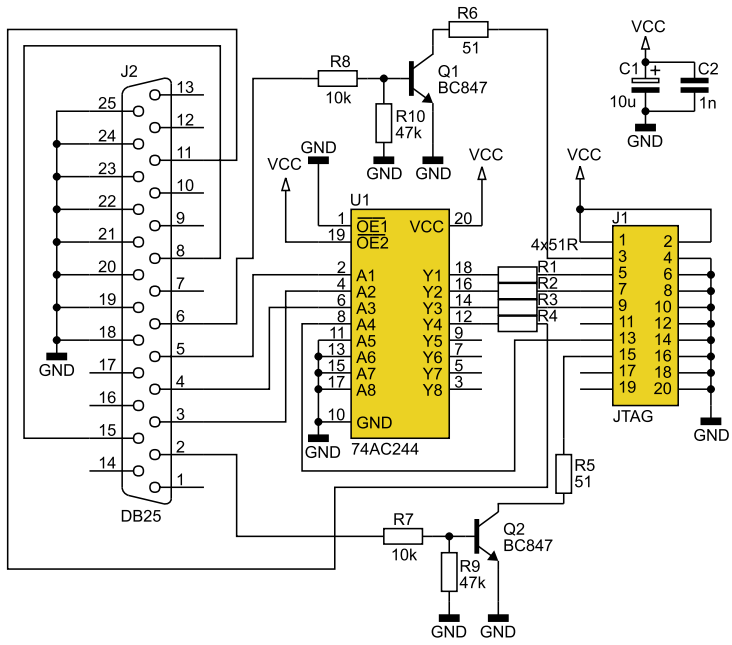
Rozwiązań programatorów JTAG jest wiele, jednym z najprostszych jest Wiggler. Jest to konwerter LPT <–> JTAG. Jego niebagatelną zaletą jest współpraca z wieloma środowiskami (m.in. CrossWorks). Schemat programatora przedstawiono na **rys. 5**. Należy pamiętać, aby kable pomiędzy płytą ewaluacyjną a programatorem były możliwie krótkie, to samo dotyczy połączenia Wigglera z portem LPT. Wzór płytki PCB jest zamieszczony pod adresem *http://paprocki.wemif.net/wiggler/wiggler.pcb*

Konfiguracja mikrokontrolera

W przedstawionym przykładzie zostanie wykorzystana biblioteka ST Firmware Library. Dzięki niej mamy dostęp do wszystkich elementów systemu bez konieczności operowania na rejestrach. Wszystkim zajmują się funkcje API.

Pierwszą czynnością wymaganą do poprawnej pracy mikrokontrolerów z rdzeniem Cortex–M3

Szczegółowe informacje o mikrokontrolerach STM32 prezentujemy na stronie 110.



Rys. 5. Schemat programatora procesorów Cortex–M3

jest konfiguracja sygnałów zegarowych i zerujących. Na **list. 1** przedstawiono sekwencję czynności przygotowujących CPU do pracy. W pierwszej kolejności ustawiamy wartości domyślne dla wszystkich rejestrów RCC (*Reset and clock control*). Służy do tego funkcja *RCC_DeInit()*. Następnie, aby system w ogóle wystartował, musimy mieć jakieś źródło sygnału zegarowego, do włączenia zewnętrznego rezonatora jest użyta funkcja *RCC_HSEConfig(RCC_HSE_ON)*. Po tej czynności czekamy, aż sygnał zegarowy będzie poprawny. Jeśli wszystko ruszyło bez błędów, to przechodzimy do kolejnych niezbędnych czynności konfiguracyjnych. Ustawiamy sposób pracy pamięci Flash i taktowanie wewnętrznych magistral. Ostatnią czynnością niezbędną do uruchomienia systemu jest poinformowanie mikrokontrolera o tym, z jakiego źródła ma pobierać sygnał taktujący. Można oczywiście użyć zewnętrznegy rezonator, jednak trzeba pamiętać, że płyta ewaluacyjna ma zamontowany kwarc 8 MHz. Zatem, aby uzyskać użyteczny sygnał 72 MHz, należy wykorzystać wbudowany w mikrokontroler układ PLL do powielenia częstotliwości pracy zewnętrznego oscylatora.

Na koniec pozostaje już tylko włączenie wykorzystywanych w danym projekcie peryferiów, w tym przypadku portów we/wy i magistrali SPI.

Krzysztof Paprocki
paprocki.krzysztof@gmail.com

List. 1. Sekwencja czynności przygotowujących CPU do pracy

```
void RCC_Configuration(void)
{
  RCC_DeInit();

  /* Enable HSE */
  RCC_HSEConfig(RCC_HSE_ON);

  /* Wait till HSE is ready */
  HSEStartUpStatus = RCC_WaitForHSEStartUp();
  if(HSEStartUpStatus == SUCCESS)
  {
   /* Enable Prefetch Buffer */
   FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
   /* Flash 2 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);

   /* PLLCLK = 8MHz * 9 = 72 MHz */
   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);

   /* Enable PLL */
   RCC_PLLCmd(ENABLE);
    /* Wait till PLL is ready */
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);

   /* Select PLL as system clock source */
   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

   /* Wait till PLL is used as system clock source */
   while(RCC_GetSYSCLKSource() != 0x08);
  }

/* Enable peripheral clocks ------------------------------------------------*/
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB
   | RCC_APB2Periph_GPIOC  | RCC_APB2Periph_GPIOD
   | RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
}
```