STM32 dla początkujących (i nie tylko)

Porty wejścia/wyjścia, pakiet kompilatora i pierwszy program

Druga część artykułu o Panelu Edukacyjnym z mikrokontrolerem STM32 (EP 9/2014) zostanie poświęcona portom wejścia/wyjścia (GPIO). Dla zrozumienia zasady działania i przetestowania tych peryferii posłuży prosty program demonstracyjny. To także okazja do pokazania sposobu korzystania z firmowej biblioteki STM32F i wybranego zintegrowanego środowiska programistycznego.

W „dużej informatyce" pierwszy program demonstracyjny zazwyczaj wyświetla słynne „Hello world!", a w świecie mikrokontrolerów pierwszy program zwykle migoce diodą LED dołączoną do którejś linii wejścia/wyjścia. Linie te umożliwiają mikrokontrolerowi kontakt ze światem zewnętrznym, pobieranie informacji i sterowanie dołączanymi peryferiami. Typowo do wymiany danych wystarczają dwa poziomy: niski „L", gdy napięcie na wyprowadzeniu GPIO ma potencjał bliski potencjałowi masy i poziom wysoki „H", gdy napięcie na nóżce GPIO jest bliskie zasilającemu. Niektóre porty mogą mieć funkcje dodatkowe i działać z innymi pośrednimi poziomami sygnałów, o czym napiszę później.

W mikrokontrolerach STM32 linie GPIO są połączone w grupy po 16. Zwykle określenie „port" jest używane wymiennie – zarówno w odniesieniu do pojedynczej linii GPIO, jak i całej grupy w obszarze pojedynczego rejestru sterującego.

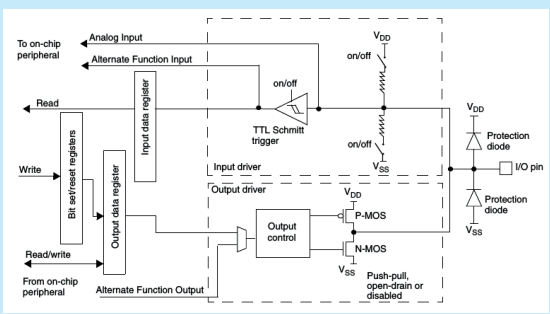
Mikrokontrolery STM32F nawet tego samego rodzaju różnią liczbą linii GPIO i portów. Liczba linii zależy od typu obudowy mikrokontrolera. Obudowy QFP mogą mieć od 48 do nawet 176 wyprowadzeń. Kolejne 16-bitowe porty oznaczone są indeksem literowym: PA, PB, PC... Zaletą STM-ów jest to, że wszystkimi portami steruje się identycznie. Informacji o ilości dostępnych portów w zastosowanym typie mikrokontrolera należy szukać w dokumentacji. W przypadku STM32F103RCT dokumentacja na internetowej stronie producenta jest oznaczona *CD00191185.pdf*.

Porty ogólna budowa i rejestry sterujące

Mikrokontrolery STM32F pozwalają na bardzo elastyczne dostosowanie sposobu działania indywidualnej linii GPIO do potrzeb współpracujących z nią układów. Można się o tym przekonać przeglądając dokumentację techniczną, chociażby plik *Reference manual* oznaczony *CD00171190.pdf*. Tabele *Port bit configuration table* numer 20 i następna 21 na stronie 156 zawierają zestawienie dostępnych do ustawienia parametrów linii. Między innymi można określić czy port ma działać jako wejście lub wyjście, czy ma być włączone wewnętrzne podciąganie do napięcia zasilającego, jaka ma być maksymalna szybkość przełączania. Takie możliwości zapewnia złożona struktura wewnętrzna odpowiadająca za funkcjonowanie każdej linii GPIO. Uproszczony schemat blokowy obwodu sterującego pojedynczą linią GPIO pokazano na **rysunku 1**. Za funkcje wyjściowe i wejściowe odpowiadają osobne bloki zaznaczone liniami przerywanymi. W przypadku wyjścia przełączane tranzystory P-MOS i N-MOS powodują podanie na linię portu naprzemiennie albo napięcia o potencjale masy, albo zasilania. Gdy linia pracuje jako wejście, tranzystory są wyłączane. Wejście może znajdować się w stanie wysokiej impedancji lub być podciągane do masy albo zasilania poprzez wewnętrzne oporniki. Na to jak port działa mają wpływ ustawienia wewnętrznych rejestrów mikrokontrolera. Ich opis, adresy i funkcje można znaleźć w pliku *Reference manual* (od strony 166).

Za obsługę każdego z portów PA, PB, PC itd. odpowiada grupa osobnych rejestrów. Poniżej przedstawiam ich zestawienie i funkcje. Indeks „x" określa oznaczenie literowe portu (PA, PB, PC... itd.).

- *GPIOx_CRL* rejestr konfigurujący działanie linii I/O o numerach 0-7. Działanie każdej linii 0…7 określają ustawienia 4 bitów. Pierwsze 4 bity dotyczą linii 0 (np. PA.0), kolejne 4 linii 1 (np. PA.1) itd. Każde 4 bity składają się z 2 podgrup:
  - *MODEy[2 bity]* (‚y' oznacza numer linii) określają podstawowy tryb pracy linii I/O (00 – linia jest wejściem; 01 – linia jest wyjściem pracującym z maksymalną szybkością przełączania do 10 MHz; 10 – linia jest wyjściem pracują-



Rysunek 1. Uproszczony schemat blokowy obwodu sterującego pojedynczą linią GPIO

cym z maksymalną szybkością przełączania do 2 MHz, 11 – linia jest wyjściem pracującym z maksymalną szybkością przełączania do 50 MHz; uwaga: szybkość przełączania wpływa na poziom generowanych zaburzeń oraz na pobór prądu, mniejsza szybkość to mniejszy poziom zakłóceń i mniejszy pobór energii).

– *CNFy[2 bity]* (‚y' oznacza numer linii)

• Druga grupa bitów w zależności od wybranego powyżej trybu wejścia lub wyjścia określa dodatkowe funkcje linii I/O:

– dla portu pracującego jako wejście: 00 – port będzie współpracował z sygnałami analogowymi podawanymi do przetworników A/C; 01 – wejście w trybie wysokiej impedancji bez podciągania; 10 – wejście z wewnętrznym podciąganiem do poziomu napięcia zasilania.

– dla portu pracującego jako wyjście: 00 – wyjście z przełączaniem do poziomów napięć zasilających, 01 – wyjście w trybie otwartego drenu może sterować zewnętrznym obciążeniem (np. LED lub cewką przekaźnika), 10 i 11 – alternatywne tryby pracy (zostaną omówione przy innej okazji).

• *GPIOx_CRH* rejestr konfigurujący tryb pracy linii I/O o numerach 8…15. Działanie analogiczne do opisanego powyżej.

• *GPIOx_IDR* rejestr odczytujący jednocześnie stan 16 linii portu.

• *GPIOx_ODR* bity ustawiające poziom na tych liniach portu, które pracują jako wyjścia. Jeżeli linia jest wejściem z podciąganiem wewnętrznym, ustawienie indywidualnych bitów dla każdej z linii określi czy linia będzie podciągana do masy, czy do zasilania.

• *GPIOx_BSRR* rejestr pozwalający ustawiać indywidualne poziomy na wybranych liniach portu. BR15-1 wpisanie „1" na pozycję bitu o wybranym indeksie spowoduje ustawienie poziomu niskiego na linii o odpowiadającym numerze. Wpisanie „0" niczego nie zmienia. BS15-1 wpisanie „1" na pozycję bitu o wybranym indeksie spowoduje ustawienie poziomu wysokiego na linii o odpowiadającym numerze. Wpisanie „0" niczego nie zmienia. Jednoczesne wpisanie „1" na te same pozycje bitów BR i BS spowoduje ustawienie linii na poziomie wysokim.

• *GPIOx_BRR* jednoczesne zerowanie (ustawianie poziomu niskiego) na liniach I/O pracujących jako wyjściowe. Wyzerowane zostaną linie GPIO, których bity w rejestrze będą ustawione. Wyzerowanie bitu w rejestrze nie zmienia stanu linii GPIO.

• *GPIOx_LCKR* port, którego odpowiadający mu bit w tym rejestrze będzie miał wartość „1" ulega „zamrożeniu". Oznacza to, że do czasu kolejnego restartu jego stan i funkcja nie będą mogły być zmienione.

STM32F10x Standard Peripherals Firmware Library – łatwiejszy sposób

Bezpośrednie operowanie na rejestrach sterujących jest kłopotliwe. Trzeba pamiętać nie tylko to, jaką funkcję pełni rejestr, ale także jak należy ustawić poszczególne bity. W dodatku każdy port ma swój zestaw rejestrów. Żeby ułatwić życie programistom firma ST bezpłatnie udostępniła biblioteki realizujące to samo w nieco łatwiejszy sposób. Biblioteki zwalniają z obowiązku wyszukiwania adresów rejestrów w pamięci mikrokontrolera i z uciążliwego wyliczania pozycji bitu, który np. ustawia poziom wysoki na wyprowadzeniu PC.3. Korzystając z bibliotek można to zrobić za pomocą procedur o łatwiejszych do zapamiętania nazwach. Zastosowanie bibliotek ma także swoje wady – może zwiększyć objętość kodu wynikowego, skomplikować strukturę programu i spowolnić jego działanie, jednak na ogół przeważają korzyści wynikające z użycia bibliotek standardowych.

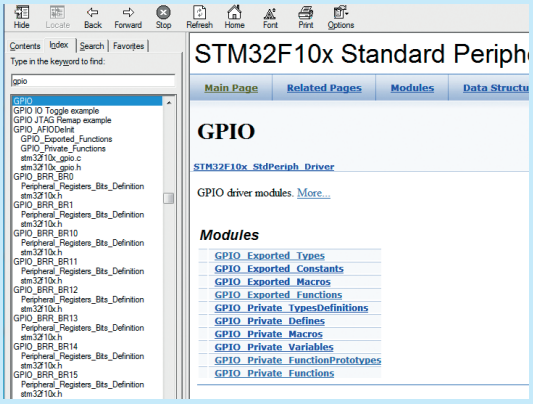
W obszernym pliku archiwum *STM32F10x Standard Peripherals Firmware Library* mieszczą się nie tylko biblioteka i związane z nią pliki definicji, ale także szablony gotowych projektów i przykłady. Po rozpakowaniu otrzymujemy rozbudowaną strukturę katalogów, podkatalogów, plików i początkowo – aż trudno się w tym gąszczu poruszać. Ponieważ w tej chwili interesują nas funkcje związane z portami GPIO, trzeba zapamiętać, że odnoszące się do tego tematy będą zawierały w nazwie „GPIO". Do poruszania się po bibliotekach najprościej wykorzystać plik pełniący funkcję menedżera *stm32f10x_stdperiph_lib_um.chm*.

Po kliknięciu na plik i uruchomieniu menedżera należy wybrać opcję *Index*, a w polu poszukiwań wpisać GPIO. Po dwukrotnym kliknięciu na liście z lewej na pozycję GPIO, z prawej strony wyświetli się zestawienie *MODULES* – pokazano to na **rysunku 2**.

Po kliknięciu na pozycję *GPIO_Exported_Functions* wyświetli się lista funkcji związanych ze sterowaniem, zapisem i odczytem danych z portów I/O. I tak np. funkcja *GPIO_ReadInputData* zwraca 16 bitową wartość odpowiadającą ustawieniu wszystkich linii odczytywanego portu. Jako argument należy przy wywołaniu funkcji podać stałą oznaczającą port, który ma być odczytany. Z kolei funkcja *GPIO_ReadInputDataBit* odczytuje stan pojedynczej linii. Jako argumenty wywołania należy podać oznaczenie portu i numer linii. Jeżeli zwracana wartość będzie różna od ‚0' linia ma poziom wysoki. Oprócz opisu funkcji wyświetlana jest także specyfikacja argumentów jej wywołania oraz zwracane wartości.

Korzystanie z biblioteki

Biblioteka musi być właściwie powiązana z naszym projektem oprogramowania, a sam projekt przystosowany



Rysunek 2. Okno menedżera biblioteki *STM32F10x Standard Peripherals Firmware Library*

do używanego środowiska programistycznego. Twórcy pakietu pomyśleli o tym przygotowując gotowe szablony, a także projekty przykładowe dla kilku środowisk programistycznych. Można się o tym przekonać otwierając podkatalog: *\Project\STM32F10x_StdPeriph_Examples*. Znajdują się tam przykłady związane z różnymi peryferiami mikrokontrolera, w tym i z GPIO. Z kolei, w podkatalogu *\Project\STM32F10x_StdPeriph_Template* umieszczono szkielety projektów dla kilku środowisk programistycznych, w tym dla kompilatora firmy Keil. W dalszych częściach darmowa wersja pakietu (MDK-ARM v5) będzie służyła do pracy nad programami demonstracyjnymi.

MDK-ARM jak zacząć?

Zanim przejdziemy do omówienia przykładowego programu do testowania portów mikrokontrolera STM32F zamontowanego na Panelu Edukacyjnym, należy wspomnieć o przygotowaniu do pracy zintegrowanego środowiska programistycznego.

Do celów edukacyjnych doskonale się nadaje pakiet MDK-ARM v5 firmy Keil. Dla zastosowań niekomercyjnych i przy ograniczeniu kodu wynikowego do 32 kB jest on darmowy. Pakiet współpracuje z programatorem *ST-Link/V2* pozwalając nie tylko na programowanie mikrokontrolerów na Panelu Edukacyjnym, ale również debugowanie oprogramowania.

Po zainstalowaniu na twardym dysku komputera, wszystkie składniki pakietu są od razu gotowe do pracy. MDK-ARM v5 to profesjonalne środowisko o dużych możliwościach, korzystające z tzw. *projektów*. Każdy projekt jest utworzony przez zestaw plików zawierających poszczególne fragmenty tworzonego programu (pliki C, pliki nagłówkowe H i inne) oraz pliki pomocnicze, które określają ustawienia samego kompilatora, takie jak opcje pracy, sposób edycji dokumentów, wybór mikrokontrolera, dla którego pisane jest oprogramowanie itd. Słowem – wszystko, co potrzebne, by uniknąć żmudnego każdorazowego ustawiania tych samych opcji.

Przykłady będą dostępne właśnie w formie gotowego projektu. Ponieważ będą wykorzystywały bibliotekę *STM32F10x Standard Peripherals Firmware Library* dla prawidłowej kompilacji i debugowania ich położenie nie może być przypadkowe. Wszystkie przykłady należy umieścić w podkatalogu wewnątrz biblioteki: *\Project\STM32F10x_StdPeriph_Template\Podkatalog_przykładu*. Oczywiście, pliki przykładu można zapisać gdzie indziej, ale wtedy kompilacja będzie możliwa po samodzielnym podaniu ścieżek dostępu do plików nagłówkowych biblioteki.

Główny plik projektu nosi nazwę *„Project.uvproj”*. Korzystając z menu MDK-ARM v5 należy wybrać *Project Open Project...* i dalej, korzystając ze standardowego okna nawigacyjnego, wskazać na podkatalog *Podkatalog_przykładu*, po otwarciu którego wyświetli się plik *Project.uvproj*. Alternatywnie, można skorzystać z ikony otwarcia folderu na belce edytora – w otwartym oknie wyszukiwania zaznaczyć opcję rozszerzenia *.uvproj* i otworzyć folder oraz plik w sposób opisany wyżej. Po kliknięciu na *Project.uvproj* otwiera się okno edytora pakietu MDK-ARM v5, które po wczytaniu projektu naszego programu demonstracyjnego powinno wyglądać podobnie jak na **rysunku 3**.

Z lewej widoczne jest drzewo plików otwartego projektu. Drzewo wskazuje pliki programu np. main.c, pliki biblioteki, do których odwołują się procedury programu jak i pewne pliki dodatkowe. Z prawej jest pole edycyjne, na którym jest wyświetlana zawartość plików programu. Poszczególne pliki można otwierać za pośrednictwem menu lub klikając na nazwę pliku w drzewie projektu. Poniżej krótka lista najważniejszych narzędzi dostępnych z poziomu menu okna edytora pakietu MDK-ARM v5:

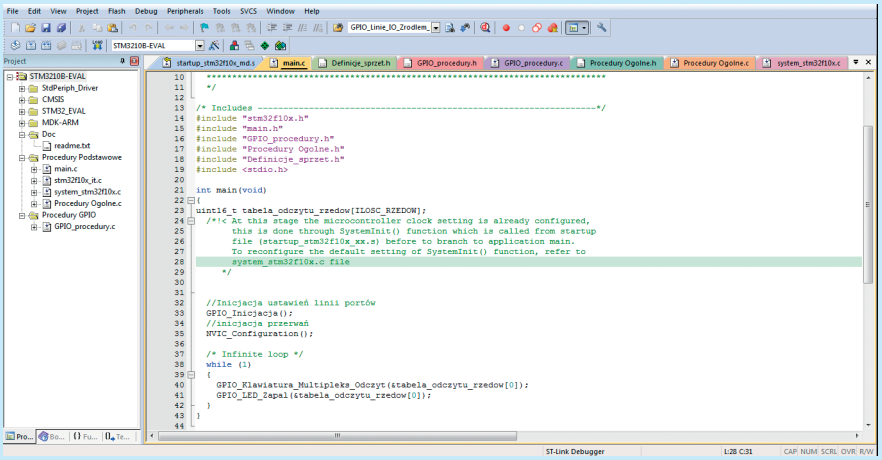
- *Project* ➔ *Rebuild all target files* – kompilowanie wszystkich plików wchodzących w skład projektu oraz *Project* ➔ *Build target* – kompilowanie tylko tych plików w obrębie projektu, które zostały zmienione po ostatnim skompilowaniu projektu. W wyniku działania obu opcji otrzymujemy pliki wynikowe, które mogą służyć do zapisu do pamięci Flash mikrokontrolera, względnie do debugowania oprogramowania. Do obu opcji można uzyskać dostęp poprzez ikonki na belce edytora.
- *Project* ➔ *Options for Target*. Opcje kompilatora. **Uwaga!** Na ogół nie ma potrzeby niczego zmieniać, a jeśli już, to ostrożnie. Wśród opcji kompilatora znajdziemy:

– *Device* – wybór typu mikrokontrolera, dla którego program ma być kompilowany.

– *Target* – ustawienia wewnętrznych pamięci mikrokontrolera i częstotliwość zastosowanego kwarcu. Dla Panelu jest to 8 MHz.

– *Output* – pliki generowane podczas kompilacji. Jeżeli będzie wykorzystywany debuger, należy zaznaczyć *Debug Information (kompilacja trwa wtedy nieco dłużej)*. Jeżeli ma być generowany plik wynikowy w formacie HEX, należy zaznaczyć *Create HEX file*. Pliki wynikowe są standardowo umieszczane w podkatalogu *STM3210B-EVAL* z nazwami takimi, jak nazwa podkatalogu. Nazwę można zmienić wpisując nową w polu *Name of Executable*.

– *C/C++*. Opcje kompilatora związane z plikami języka C. Dodając nowy plik można tu wskazać po-



Rysunek 3. Okno edycji środowiska Keil MDK-ARM5

łożenie jego pliku nagłówkowego „.h”. Przy polu *Include Paths* należy nacisnąć przycisk, co spowoduje otwarcie dodatkowego okienka narzędziowego. Po naciśnięciu ikony *Insert* w standardowy sposób można wpisać lub wskazać położenie podkatalogu zawierającego nowe pliki nagłówkowe.

- *Debug* (opcje debugowania). Zaznaczenie *Use Simulator* włączy symulowanie działania programu w pamięci komputera PC. Opcja przydatna podczas testowania fragmentów procedur bez konieczności programowania mikrokontrolera. Opcja *Use* pozwala na wybór typu programatora np. ST-LINK. Po naciśnięciu *Settings* przechodzimy do ustawień przystosowujących pakiet kompilatora do współpracy z programatorem. Na zakładce *Debug* należy wybrać opcję *Port: JTAG*. Na zakładce *Flash Download* zaznaczamy opcje: *Erase Sectors, Program, Verify*. Kliknąć przycisk *Add* i wybrać algorytm programowania o nazwie *STM32F10x Med-density Flash*.
- *Utilities*. Opcje współpracy z programatorem. Powinny być takie same jak w sekcji *Debug*.
- *Debug* → *Start/Stop Debug Session*. Rozpoczynanie i kończenie sesji debugera. Do Panelu Edukacyjnego powinien być dołączony programator. Na początku sesji zawartość pliku wynikowego jest zapisywana do pamięci Flash mikrokontrolera. Po uruchomieniu mikrokontroler wykonuje instrukcje programu pod nadzorem debugera. Pozwala to w każdej chwili na zawieszenie wykonywania programu, ustawienie pułapek, podejrzenie zawartości rejestrów i zmiennych itp. Opcja dostępna także po naciśnięciu ikony w kształcie lupy z literą „D” na belce edytora.

PanEduSTM32F_Demo1_GPIO – przykładowy program demonstrujący sterowanie GPIO

Projekt z przykładowym programem demonstrującym sterowanie liniami GPIO nosi nazwę PanEduSTM32F_Demo1_GPIO. Najpierw na dysku w obrębie zainstalowanej biblioteki *STM32F10x Standard Peripherals Firmware Library* należy utworzyć podkatalog np. *\Project\STM32F10x_StdPeriph_Template\Demo1_GPIO*. Do utworzonego podkatalogu należy przekopiować wszystkie pliki projektu przykładu. Następnie uruchamiamy środowisko MDK-ARM i w opisany wcześniej sposób otwieramy projekt.

Program przykładowy pokazuje sposób, w który można sterować liniami GPIO mikrokontrolera STM32 z użyciem biblioteki *STM32F10x Standard Peripherals Firmware Library*. Do demonstracji wykorzystano 8 przycisków S1…S8 oraz 8 diod LED D1…D8 zamontowanych na płytce Panelu Edukacyjnego. Klawiatura pracuje w trybie multipleksowanym. Tworzy ją matryca 3 kolumn i 4 rzędów, której w węzłach znajdują się przyciski. Żeby wykryć, który przycisk jest naciskany zeruje się kolejne linie GPIO kolumn. Po wyzerowaniu danej kolumny jest odczytywany poziom linii GPIO połączonych z rzędami klawiatury. Odczytanie poziomu niskiego na linii rzędów w połączeniu z informacją o wystawieniu takiego stanu na linii kolumn pozwala na zidentyfikowanie naciśniętego przycisku.

Dla zwiększenia przejrzystości cały program został podzielony na kilka plików:

- *main.c* – plik zawierający program główny.
- *Procedury_Ogolne.c* – plik z procedurami pomocniczymi.
- *GPIO_procedury.c* – plik z procedurami obsługi linii GPIO.

Na początku procedury *main()* zostaje przeprowadzona inicjacja, czyli skonfigurowanie używanych linii GPIO. Inicjacja samego mikrokontrolera zostaje przeprowadzona automatycznie przez środowisko programistyczne. Dzieje się to poprzez niejawne wywołanie procedur z pliku *startup_stm32f10x_md.s*.

Opisany dalej sposób inicjacji i sterowania liniami GPIO odbywa się poprzez odwołania do nadanych tym liniom nazw symbolicznych. Najpierw w pliku *Definicje_sprzet.h* jest deklarowany typ wyliczeniowy *Linie_IO_TypeDef*. Tutaj, wszystkim używanym dalej w programie liniom GPIO, zostają nadane łatwiejsze do zapamiętania nazwy symboliczne np. *LINIA_LED1_WY* czy *LINIA_KLAW_MUL_K1_WY*. Następnie, dyrektywami #define tworzone są makrodefinicje dla stałych, które są używane przez procedury sterujące liniami GPIO. Stałe zapisane są w tabelach na początku pliku *GPIO_procedury.c* w kolejności zgodnej z numeracją linii zadeklarowaną w typie wyliczeniowym. Dzięki temu odwołanie do stałej związanej z określoną linią będzie się odbywać poprzez jej nazwę symboliczną.

Inicjacja linii GPIO odbywa się poprzez procedurę *GPIO_procedura.c* → *GPIO_Inicjacja*. Procedura ta kolejno dla wszystkich linii wywołuje następną procedurę *GPIO_Linie_IO_Konfig()*. Ta procedura, korzystając z funkcji bibliotecznej *GPIO_Init()*, ustawia parametry pracy linii. Przed wywołaniem funkcji w strukturze *GPIO_InitStructure* są inicjowane odpowiednie pola. Inna funkcja biblioteczna – *RCC_APB2PeriphClockCmd* – dołącza wewnętrzny zegar do bloków mikrokontrolera odpowiedzialnych za pracę każdej z linii.

Inicjacja linii kończy się wstępnym ustawieniem poziomów na liniach wyjściowych. Użyte są w tym celu procedury *GPIO_Linie_IO_High* i *GPIO_Linie_IO_Low*. Dla przykładu procedury te nie korzystają z funkcji bibliotecznych a odwołują się bezpośrednio do rejestrów mikrokontrolera.

Obsługa klawiatury odbywa się w nieskończonej pętli procedury main(). Najpierw jest wywoływana procedura *GPIO_Klawiatura_Multipleks_Odczyt()*. Jej działanie polega na zerowaniu kolejnych linii kolumn. Jednocześnie jest odczytywany stan linii rzędów i zapamiętywany w tabeli *tabela_odczytu_rzedow[]*. Procedura *GPIO_Klawiatura_Multipleks_Odczyt()* używa funkcji bibliotecznych *GPIO_WriteBit* i *GPIO_ReadInputData*.

W dalszej kolejności w pętli głównej funkcji *main()* jest wywoływana druga procedura *GPIO_LED_Zapal()*. Na podstawie danych zapisanych w tabeli *tabela_odczytu_rzedow[]* procedura oblicza, który z przycisków został naciśnięty i zapala odpowiadającą mu diodę LED. Procedura korzysta z funkcji bibliotecznej *GPIO_WriteBit*. Ze względu na uproszczoną budowę klawiatury na Panelu Edukacyjnym program będzie działał prawidłowo dla kolejnych naciśnięć tylko pojedynczego przycisku.

Jeżeli wszystko zostało ustawione tak jak to wcześniej opisano program powinien się dać skompilować bez żadnego problemu. Można go zapisać do pamięci Flash mikrokontrolera bezpośrednio ze środowiska MDK-ARM korzystając z opcji *Debug* → *Start* lub wykorzystując wygenerowany plik HEX znajdujący się w podkatalogu *STM3210B-EVAL*.

Ryszard Szymaniak, EP