Narzędzia dla Precision32 (3)

Oprogramowanie w postaci kodu źródłowego

W trzeciej części cyklu artykułów o narzędziach dla mikrokontrolerów Precision32 firmy Silicon Labs przygotowane przez producenta oprogramowanie przedstawione zostanie w postaci kodu źródłowego.

Dla przypomnienia - podział narzędzi

Dostępne dla mikrokontrolerów Precision32 narzędzia projektowo-rozwojowe można podzielić na trzy grupy: narzędzia sprzętowe, narzędzia programistyczne oraz oprogramowanie w postaci kodu źródłowego.

Do omówionych w pierwszym artykule narzędzi sprzętowych należą:

- Modułowa platforma ewaluacyjna UDP (*Unified Development Platform*), którą użytkownik może rekonfigurować łącząc płytkę bazową UDP Motherboard z wybranymi przez siebie płytkami trzech różnych rodzajów: z mikrokontrolerem (UDP MCU card), rozszerzeniowymi oraz komunikacji radiowej.
- Interfejsy sprzętowe (programatory/debugery): USB Debug Adapter (kompatybilny z pakietem do tworzenia oprogramowania Precision32 development suite oraz MDK-ARM), ULINK (MDK-ARM) i J-Link (Embedded Workbench for ARM).

Do omówionych w drugim artykule narzędzi programistycznych należą:

- Program komputerowy AppBuilder służący do generowania kodu źródłowego odpowiedzialnego za skonfigurowanie mikrokontrolera zgodnie z wybranymi przez użytkownika (za pomocą graficznego interfejsu) ustawieniami dotyczącymi: peryferiów (które z nich mają zostać włączone i z jakimi parametrów pracy), bloku zegarowego (wybór źródła sygnału zegarowego), przyporządkowania peryferiów do wyprowadzeń układu.
- Pakiety do tworzenia i rozwoju oprogramowania: pakiet MDK-ARM (Microcontroller Development Kit) firmy Keil/ARM z IDE o nazwie μVision, pakiet IAR EWARM (Embedded Workbench for ARM) firmy IAR Systems z IDE o nazwie EWARM IDE, pakiet Precision32 development suite firmy Silicon Labs z IDE o nazwie Precision32 IDE.

W tym artykule omówiona zostanie trzecia i zarazem ostatnia grupa narzędzi projektowo-rozwojowych dla mikrokontrolerów Precision32 - oprogramowanie w postaci kodu źródłowego, które nosi nazwę Precision32 SDK (*Software Development Kit*).

Ogólny model systemu z mikrokontrolerem z rdzeniem Cortex

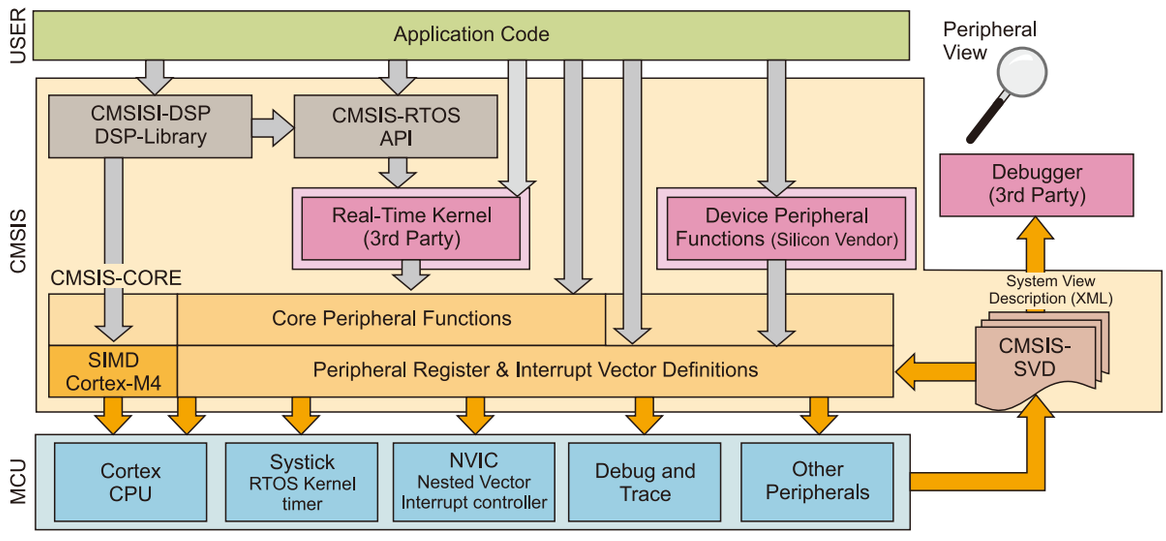
Aby lepiej zrozumieć, za co odpowiadają komponenty pakietu oprogramowania Precision32 SDK, warto w tym miejscu przedstawić zaproponowany przez firmę ARM warstwowy model systemu z mikrokontrolerem opartym na rdzeniu Cortex (**rysunek 1**). Model ten składa się z następujących warstw:

- zasobów sprzętowych mikrokontrolera,
- standardu CMSIS (*Cortex Microcontroller Software Interface Standard*),
- właściwej aplikacji.

Najniższą warstwę modelu stanowią zasoby sprzętowe mikrokontrolera, a więc jego rdzeń i peryferia.

Warstwą wyższą jest standard CMSIS, który jest autorskim pomysłem firmy ARM. Stanowi on uniwersalny interfejs programowy do zasobów sprzętowych każdego mikrokontrolera z rdzeniem ARM Cortex (obecnie CMSIS jest kompatybilny z rdzeniami ARM Cortex-M0, ARM Cortex-M3 i ARM Cortex-M4). Zgodnie z tą koncepcją w pierwotnej wersji 1.0 CMSIS składał się ze zestandaryzowanych funkcji i definicji (nazwanych wspólnie CMSIS-CORE) podzielonych na dwa moduły: bloku dostępu do rejestrów rdzenia mikrokontrolera i bloku dostępu do rejestrów bloków peryferyjnych mikrokontrolera. W wersji 2.0 standard CMSIS rozszerzono dodatkowo o blok cyfrowego przetwarzania sygnałów CMSIS-DSP obejmujący np. filtry, operacje matematyczne na liczbach zmiennoprzecinkowych i transformatę Fouriera. W najnowszej wersji 3.0 standard CMSIS zyskał jeszcze dwa bloki – standardowy interfejs dla systemów operacyjnych (CMSIS-RTOS) oraz standardowy interfejs do podglądu pracy systemu w postaci plików XML (CMSIS-SVD).

Ujednolicenie wymienionych wyżej mechanizmów to cecha, która sprawia, że oprogramowanie pisane dla mikrokontrolerów



Rysunek 1. Ogólny model systemu z mikrokontrolerem opartym na rdzeniu Cortex

082-085_precision32(3).indd 82 2013-05-29 10:02:34

opartych na rdzeniach Cortex, wytwarzanych przez różnych producentów, ma wspólną podstawę, którą jest właśnie CMSIS. Zalety wynikające z tego faktu to:

- ułatwienie rozpoczęcia pisania aplikacji przy zmianie producenta mikrokontrolerów,
- ułatwienie przenoszenia aplikacji między mikrokontrolerami różnych producentów.

Ostatnią, najwyższą warstwą modelu warstwowego systemu z mikrokontrolerem opartym na rdzeniu Cortex jest aplikacja. Odwołuje się ona do standardu CMSIS bezpośrednio, bądź z wykorzystaniem oprogramowania pośredniczącego (bibliotek, sterowników itp.) lub systemu operacyjnego.

Nazwy plików standardu CMSIS są zestandaryzowane. Nazwy i i opis zawartości głównych plików CMSIS przedstawiono w **tabeli 1**.

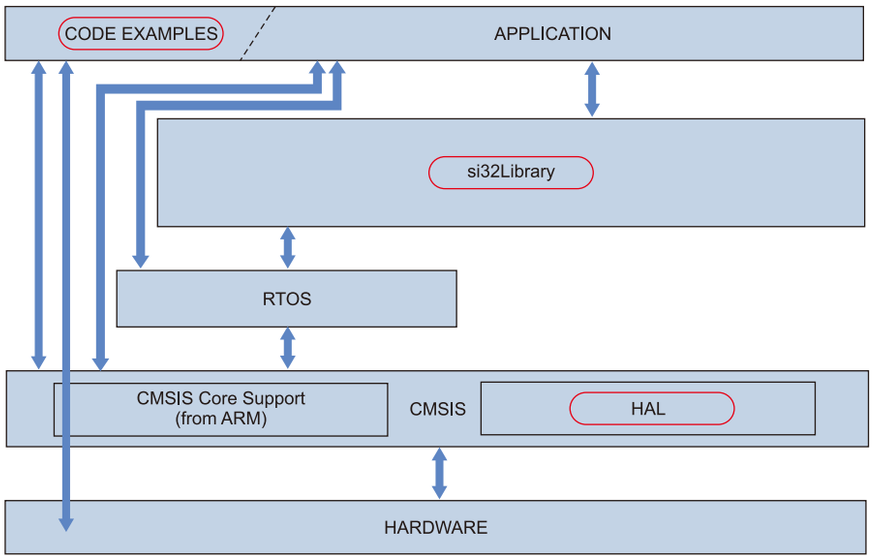
Jednym z modułów (opcjonalnym) przedstawionego powyżej standardu CMSIS jest blok dostępu do peryferiów mikrokontrolera za pomocą wysokopoziomowych funkcji. Nie jest on tworzony przez firmę ARM. Jest tak dlatego, gdyż każdy producent w swoich układach z rdzeniem Cortex integruje inne peryferia tworząc indywidualną mapę pamięci i rejestrów. Dlatego niemożliwym jest napisanie jednego takiego bloku, pasującego do różnych rodzin mikrokontrolerów. Z tego powodu obowiązek stworzenia bloku dostępu do peryferiów mikrokontrolera za pomocą wysokopoziomowych funkcji spoczywa na producentach mikrokontrolerów.

Zarys struktury oprogramowania dla mikrokontrolerów Precision32

Pakiet oprogramowania Precision32 SDK składa się z trzech komponentów: oprogramowania HAL (*Hardware Access Layer*), przykładów aplikacji oraz grupy bibliotek nazwanej si32Library. Każdy z tych komponentów jest w całości dostępny w postaci kodów źródłowych. Miejsce komponentów pakietu Precision32 SKD w zaproponowanym przez firmę ARM warstwowym modelu systemu z mikrokontrolerem opartym na rdzeniu Cortex przedstawiono na **rysunku 2**.

W modelu tym komponent HAL wchodzi w skład warstwy CMSIS i pełni rolę bloku dostępu do peryferiów mikrokontrolera za pomocą wysokopoziomowych funkcji. Komponent przykładów aplikacji można rozpatrywać jako część aplikacji użytkownika. Komponent *si32Library* nie należy do żadnej z warstw modelu. Można go umiejscowić między warstwą CMSIS a warstwą aplikacji użytkownika. Komponent ten zawiera implementacje różnych mechanizmów wykorzystywanych w aplikacjach.

Każdy z komponentów pakietu Precision32 SDK zostanie omówiony w oddzielnym rozdziale.



Rysunek 2. Umiejscowienie komponentów pakietu Precision32 SDK w zaproponowanym przez firmę ARM warstwowym modelu systemu z mikrokontrolerem opartym na rdzeniu Cortex

Oprogramowanie HAL

Jako pierwszy przedstawiony zostanie komponent HAL. Opis ogólny uzupełnia przykład właściwy dla interfejsu USART.

Budowa HAL zorganizowana została zgodnie z budową wewnętrzną mikrokontrolera Precision32. Oznacza to, że HAL składa się z modułów, z których każdy odpowiada za inne peryferium, a konkretniej grupę bliźniaczych peryferiów (peryferia o tej samej funkcjonalności są sterowane poprzez ten sam moduł - przykładowo interfejsy komunikacyjne USART1 i USART2 mają jeden, wspólny moduł w HAL - USART).

Każdy moduł składa się z kilku powiązanych ze sobą plików i zbudowany jest zawsze według tego samego schematu (**rysunek 3**):

- wspólny dla wszystkich modułów jest plik nagłówkowy sim3u1xx.h (dla serii mikrokontrolerów Precision32 z USB) lub sim3c1xx.h (dla serii mikrokontrolerów Precision32 bez USB),
- obowiązkowymi plikami nagłówkowymi i źródłowymi każdego modułu są: SI32_*_Registers.h, SI32_*_Type.h i SI32_*_Type.c.
- ponadto, część modułów zawiera dodatkowy plik nagłówkowy SI32_*_Support.h.

Znak „*” w nazwach plików SI32_*_Registers.h, SI32_*_Type.h, SI32_*_Type.c oraz SI32_*_Support.h oznacza nazwę modułu (przykładowo dla modułu USART pliki te noszą odpowiednio nazwy: SI32_USART_Registers.h, SI32_USART_Type.h, SI32_USART_Type.c oraz SI32_USART_Support.h).

We współdzielonych przez moduły HAL plikach nagłówkowych sim3u1xx.h i sim3c1xx.h umieszczone są:

- tworzące wektor przerwań przyporządkowania wartości liczbowych do nazw źródeł przerwań,
- definicje rejonów pamięci, które określają pojemność pamięci Flash i RAM poszczególnych modeli mikrokontrolerów,
- definicje początków rejonów pamięci (*Base Pointers*) poszczególnych peryferiów,

Pliki nagłówkowe SI32_*_Registers.h tworzą mapę rejestrów mikrokontrolera. Rejestry reprezentowane są w każdym z plików przez typy danych – struktury i unie. Bity rejestrów są polami bitowymi tychże struktur i unii. W plikach nagłówkowych SI32_*_Registers.h zawarto także definicje pozwalające na operacje manipulacji na bitach. Na końcu każdego pliku nagłówkowego z grupy SI32_*_Registers.h znajduje się główny typ strukturalny danego peryferium. Zgrupowano w nim wszystkie typy strukturalne rejestrów oraz używane przestrzenie pamięci przez peryferium. Przykładowy typ strukturalny dla modułu USART zamieszczono na **listingu 1**.

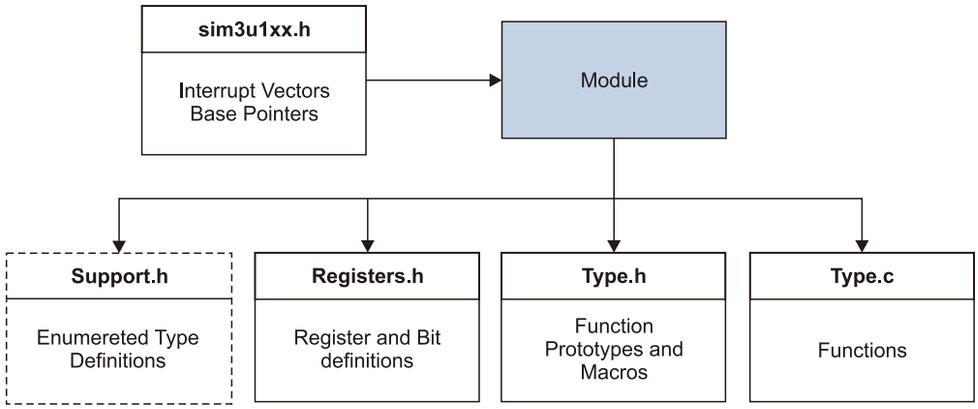
Pliki nagłówkowe sim3u1xx.h, sim3c1xx.h oraz SI32_*_Registers.h tworzą szkielet modułów HAL. W oparciu o ten szkielet właściwa funkcjonalność HAL polegająca na udostępnianiu programiście interfejsu dostępu do peryferiów została zaimplementowana w plikach nagłówkowych SI32_*_Type.h i źródłowych SI32_*_Type.c. Interfejs ten składa się makr i funkcji. Definicje makr i deklaracje funkcji znajdują się w plikach SI32_*_Type.h, natomiast ciała funkcji znajdują się w plikach SI32_*_Type.c. Nazwy makr i funkcji wskazują na wykonywane przez nie operacje, natomiast argumenty funkcji są w większości albo parametrami pracy peryferium, albo danymi do przetworzenia przez peryferium. Dzięki temu interfejs HAL jest wygodny w użyciu, gdyż programista, aby go zrozumieć, musi posiadać wiedzę o zasadzie działania pery-

082-085_precision32(3).indd 83 2013-05-29 10:02:34

feriów, nie musi natomiast zaznajamiać się z rejestrami mikrokontrolera sterującymi peryferiami. Wzór nazwy każdej funkcji jest następujący:

SI32_Peripheral_R_Operation_Mechanism (…);

Nazwa każdej funkcji i makra rozpoczyna się przedrostkiem „SI32_”. Po nim występuje nazwa peryferium, wersja peryferium, rodzaj wykonywanej operacji oraz nazwa mechanizmu, na którym operacja zostanie wykonana. Nazwę funkcji kończy opis wykonywanej operacji. Poszczególne wyrazy w nazwie funkcji rozdzielone są znakiem „_”. Dostępne operacje to między innymi: *initialize*, *enable*, *disable*, *select*, *reset*, *write*, *read*, *is*, *clear*, *start*.



Rysunek 3. Schemat budowy pojedynczego modułu w HAL

Przykładowa deklaracja funkcji z pliku SI32_USART_Type.h, służąca do ustawienia prędkości transmisji, z jaką interfejs komunikacyjny USART odbiera informacje:

```
void
_SI32_USART_A_set_rx_
baudrate(SI32_USART_A_Type*
/*basePointer*/, uwide16_t
/*rxbaud*/);
```

Ciało tej samej funkcji z pliku SI32_USART_Type.c:

```
void
_SI32_USART_A_set_rx_
baudrate(SI32_USART_A_Type *
basePointer, uwide16_t rxbaud)
{
  assert(rxbaud < 65536);   //
rxbaud < 2^16
  //{{
  basePointer->BAUDRATE.RBAUD =
rxbaud;
  //}}
}
```

Pliki nagłówkowe SI32_*_Support.h zawierają w sobie definicje struktur, które są typami wyliczeniowymi, przydatnymi podczas sterowania peryferiami. Przykładowa definicja struktury z pliku USART_A_Support.h jest na **listingu 2**.

Zgodnie z powyżej przedstawionym schematem zbudowany jest moduł dla każdego peryferium mikrokontrolera. Według tej koncepcji sterowanie peryferiami sprowadza się do wywoływania zaimplementowanych w plikach modułów funkcji. Programista najpierw wywołuje funkcje służące do konfiguracji parametrów pracy peryferiów, a następnie wywołuje funkcje sterujące działaniem peryferiów. Przykładowo fragment kodu dla interfejsu USART zamieszczono na **listingu 3**.

Biblioteki si32Library

si32Library to oprogramowanie, które tworzy warstwę pośrednią pomiędzy warstwą dostępu do peryferiów mikrokontrolera CMSIS (w tym też HAL), a warstwą właściwej aplikacji. Jest to zestaw bibliotek, które przez zaimplementowane w nich funkcjonalności mają za zadanie ułatwić programiście tworzenie bardziej skomplikowanych aplikacji, tym samym usprawniając ich działanie.

Firma Silicon Labs przygotowała osiem bibliotek, każda składa się z pewnej liczby plików źródłowych i nagłówkowych zawierających kod źródłowy. Poniżej przedstawiono nazwy i opisy dostępnych bibliotek:

- Biblioteka *si32BaseComponent*:
  - umożliwia logowanie (rejestrowanie) informacji w trakcie wykonywania programu,
  - udostępnia różne opcje alokacji (przydzielania i zwalniania) obszarów pamięci,
  - umożliwia przechowywanie informacji o błędach,
  - udostępnia szkielet mechanizmu rozwiązywania błędów.
- Biblioteka *si32ObjectComponent* udostępnia wybrane funkcjonalności języka C++. Biblioteka ta pozwala na pisanie kodu źródłowego przy wykorzystaniu podejścia obiektowego (np. hermetyzacji, dziedziczenia i polimorfizmu) w aplikacjach, w których język C++ jest niedostępny.
- Biblioteka i32ContainerComponent udostępnia obiekty do przechowywania bloków pamięci oraz innych obiektów.
- Biblioteka *si32PseudOsComponent* służy do tworzenia programów zgodnie z modelem działania systemu operacyjnego. Akcje wykonywane przez mikrokontroler podzielone są na zadania, których wykonywanie jest koordynowane przez mechanizm zarządzania zadaniami.
- Biblioteka *si32UsbComponent* udostępnia stos USB z zaimplementowanymi klasami: USB Audio Device Class, USB Mass Storage Class, USB HID Class, USB DFU Device Class.
- Biblioteki *si32HardwareComponent*, *si32IoComponent* i *si32McuComponent* udostępniają obiekty reprezentujące zasoby sprzętowe mikrokontrolera.

Listing 1. Typ strukturalny modułu USART

```
typedef struct SI32_USART_A_Struct
{
  struct SI32_USART_A_CONFIG_Struct  CONFIG  ;// Base Address + 0x0
  volatile uint32_t                  CONFIG_SET;
  volatile uint32_t                  CONFIG_CLR;
  uint32_t                           reserved0;
  struct SI32_USART_A_MODE_Struct    MODE    ;// Base Address + 0x10
  volatile uint32_t                  MODE_SET;
  volatile uint32_t                  MODE_CLR;
  uint32_t                           reserved1;
  struct SI32_USART_A_FLOWCN_Struct  FLOWCN  ;// Base Address + 0x20
  volatile uint32_t                  FLOWCN_SET;
  volatile uint32_t                  FLOWCN_CLR;
  uint32_t                           reserved2;
  struct SI32_USART_A_CONTROL_Struct CONTROL ;// Base Address + 0x30
  volatile uint32_t                  CONTROL_SET;
  volatile uint32_t                  CONTROL_CLR;
  uint32_t                           reserved3;
  struct SI32_USART_A_IPDELAY_Struct IPDELAY ;// Base Address + 0x40
  uint32_t                           reserved4;
  uint32_t                           reserved5;
  uint32_t                           reserved6;
  struct SI32_USART_A_BAUDRATE_Struct  BAUDRATE  ;// Base Address + 0x50
  uint32_t                           reserved7;
  uint32_t                           reserved8;
  uint32_t                           reserved9;
  struct SI32_USART_A_FIFOCN_Struct  FIFOCN  ;// Base Address + 0x60
  volatile uint32_t                  FIFOCN_SET;
  volatile uint32_t                  FIFOCN_CLR;
  uint32_t                           reserved10;
  struct SI32_USART_A_DATA_Struct    DATA    ;// Base Address + 0x70
  uint32_t                           reserved11;
  uint32_t                           reserved12;
  uint32_t                           reserved13;
} SI32_USART_A_Type;
```


082-085_precision32(3).indd 84 2013-05-29 10:02:35

Tabela 1. Nazwy i zawartość głównych plików CMSIS

Nazwa pliku	Zawartość pliku
core_cm<numer rdzenia Cortex-M: 0, 3 lub 4>.h (dla mikrokontrolerów Precision32: core_cm3.h)	Deklaracje i definicje (np. rejestrów) dla wybranego rdzenia ARM Cortex-M.
core_cm< numer rdzenia Cortex-M: 0, 3 lub 4>.c (dla mikrokontrolerów Precision32: core_cm3.c)	Definicje dla wybranego rdzenia ARM Cortex-M.
<nazwa_rodziny_mikrokontrolerów>.h (dla mikrokontrolerów Precision32: sim3u1xx.h lub sim3c1xx.h)	Definicje i deklaracje (w tym np. przerwań i adresów bazowych peryferiów) dla danej rodziny mikrokontrolerów.
system_<nazwa_rodziny_mikrokontrolerów >.h (dla mikrokontrolerów Precision32: system_sim3u1xx.h lub system_sim3c1xx.h)	Deklaracje (w tym np. funkcji SystemInit()) dla danej rodziny mikrokontrolerów.
system_<nazwa_rodziny_mikrokontrolerów >.c (dla mikrokontrolerów Precision32: system_sim3u1xx.c lub system_sim3c1xx.h.c)	Definicje (w tym np. funkcji SystemInit()) dla danej rodziny mikrokontrolerów.

Przykłady

Ostatnim z komponentów Precision32 SDK są przykładowe aplikacje. Każda z aplikacji jest dostępna w postaci kompletnego projektu programistycznego, który można uruchomić lub debugować na platformie sprzętowej z mikrokontrolerem. Projekty programistyczne są kompatybilne ze wszystkimi dostępnymi dla mikrokontrolerów Precision32 pakietami do tworzenia i rozwoju oprogramowania: IAR EWARM, MDK-ARM oraz *Precision32 Development Suite*. Każdy projekt programistyczny jest umieszczony w oddzielnym katalogu, którego nazwa wskazuje na to, co przedstawia przykładowa aplikacja. Dodatkowa dokumentacja dostępna jest w pliku README każdego projektu programistycznego, w którym zamieszczono opis działania aplikacji, listę wykorzystanych zasobów wewnętrznych mikrokontrolera, wartości sygnałów zegarowych taktujących użyte bloki mikrokontrolera oraz sposób postępowania podczas używania aplikacji. Kod źródłowy został napisany w sposób pozwalający na częściowe lub całkowite skopiowanie i użycie go w aplikacji docelowej użytkownika.

Przykładowe aplikacje zostały podzielone na dwie grupy. Pierwsza grupa przykładów to aplikacje, które za pomocą warstwy HAL pokazują jak skonfigurować i używać poszczególnych zasobów wewnętrznych mikrokontrolera. Dostępnych jest razem 25 przykładów, które dotyczą:

- Peryferiów, w tym między innymi: układów czasowych i liczników (tmery, licznik Watchdog, zegar czasu rzeczywistego), interfejsów komunikacyjnych (USART, UART, SPI, I2C), przetworników (A/C, C/A, prąd/napięcie), modułu AES, modułu CRC, portów wejścia/wyjścia.
- Pamięci wewnętrznej Flash (zapis i odczyt danych).
- Bloku zegarowego (ustawienie sygnałów zegarowych z użyciem PLL).

Listing 2. Definicja struktury z pliku USART_A_Support.h

```
typedef enum SI32_USART_STOP_BITS_Enum
{
  SI32_USART_A_STOP_BITS_0P5_BIT  = 0,
  SI32_USART_A_STOP_BITS_1_BIT    = 1,
  SI32_USART_A_STOP_BITS_1P5_BITS = 2,
  SI32_USART_A_STOP_BITS_2_BITS   = 3
} SI32_USART_A_STOP_BITS_Enum_Type;
```

Listing 3. Fragment kodu dla interfejsu USART

```
// Konfiguracja
SI32_PBSTD_A_set_pins_push_pull_output(SI32_PBSTD_0, 0x0001);
SI32_PBCFG_A_enable_xbar0l_peripherals(SI32_PBCFG_0, I32_PBCFG_A_XBAR0L_
USART0EN);
SI32_PBCFG_A_enable_crossbar_0(SI32_PBCFG_0);
SI32_USART_A_set_rx_baudrate(SI32_USART_0, 1041);
SI32_USART_A_set_tx_baudrate(SI32_USART_0, 1041);
SI32_USART_A_enable_rx(SI32_USART_0);
SI32_USART_A_enable_tx(SI32_USART_0);
//Wysłanie bajtu
while (SI32_USART_A_read_tx_fifo_count(SI32_USART_0) >= 4);
SI32_USART_A_write_data_u8(SI32_USART_0, val);
//Odebranie bajtu
uint8_t val;
while (SI32_USART_A_read_rx_fifo_count(SI32_USART_0) == 0);
val = SI32_USART_A_read_data_u8(SI32_USART_0);
```

- Trybu pracy PM9 o niskim poborze prądu.

Druga grupa to przykłady wykorzystujące biblioteki z zestawu *Si32Library*. Dostępnych jest razem 13 przykładów, w tym między innymi dwie aplikacje z systemem operacyjnym czasu rzeczywistego (FreeRTOS i RTX), aplikacja z timerem Systick oraz sześć aplikacji ze stosem USB.

Podsumowanie

Był to trzeci artykuł o narzędziach dla mikrokontrolerów Precision32 firmy Silicon Labs oraz piąty artykuł w całym cyklu poświęconemu tym układom. Tematyka następnych części to:

- rozpoczęcie pracy z mikrokontrolerem Precision32 krok po kroku, przy wykorzystaniu zestawu ewaluacyjnego,
- autorski projekt prostego zestawu startowego.

Dystrybutorem mikrokontrolerów Silicon Labs Precision32 oraz pakietu Keil MDK-ARM w Polsce jest firma WG Electronics.

Autor składa podziękowanie Panu Tadeuszowi Górnickiemu, prezesowi firmy WG Electronics oraz Panu Sándor Csüllög, inżynierowi aplikacyjnemu z firmy Silicon Labs, za pomoc w realizacji artykułu.

Szymon Panecki
Wydział Elektroniki
Politechnika Wrocławska
szymon.panecki@pwr.wroc.pl

Bibliografia

[1] www.silabs.com AN664: PRECISION32 CMSIS AND HAL USER'S GUIDE
[2] www.silabs.com AN668: PRECISION32 SOFTWARE DEVELOPMENT KIT CODE EXAMPLES OVERVIEW
[3] www.silabs.com AN672: PRECISION32 SI32LIBRARY OVERVIEW
[4] www.silabs.com AN673: PRECISION32 SOFTWARE DEVELOPMENT KIT OVERVIEW
[5] www.silabs.com AN675: PRECISION32™ DEVELOPMENT SUITE OVERVIEW

http://forum.ep.com.pl

082-085_precision32(3).indd 85 2013-05-29 10:02:35