Okiem praktyka

Kłopotliwy mapping portów w STM32 i sposoby radzenia sobie z tym problemem

Konstruktorzy już dawno dostrzegli zalety mikrokontrolerów 32-bitowych. Początkowo stosowali je na większą skalę głównie do bardziej wymagających zadań. Rewolucję zapoczątkowało NXP wprowadzając na rynek mikrokontrolery LPC21xx wyposażone w 32-bitowy rdzeń ARM7TDMI oraz oferując je w cenach konkurencyjnych dla „dużych" mikrokontrolerów 8-bitowych, np. ATmega128. Dzięki temu w systemach, w których niejednokrotnie, dawały się we znaki ograniczenia architektury 8-bitowej, jak mała przestrzeń adresowa wymagająca kłopotliwej segmentacji pamięci czy niewystarczająca wydajność rdzenia – rozwiązania 32-bitowe stopniowo wyparły rozwiązania 8-bitowe.

Mikrokontrolery 8-bitowe nadal królowały w nieskomplikowanych systemach, w których nie była wymagana duża wydajność CPU, a najważniejszym kryterium wyboru była niska cena. Sytuacja zaczęła stopniowo zmieniać się wraz z pojawieniem się rdzenia Cortex-M3, który w przeciwieństwie do ARM7TDMI wywodzącego się z mikrokomputerów, został w całości zaprojektowany do zastosowań mikrokontrolerowych. Udało się poprawić wydajność oraz znacząco obniżyć koszty produkcji, co przełożyło się na niższe ceny.

Jednymi z pierwszych przedstawicieli tej rodziny były mikrokontrolery firmy STMicroelectronic STM32F103 oraz STM32F101. Pierwszy, pracujący przy taktowaniu sygnałem o częstotliwości do 72 MHz jest przeznaczony do bardziej zaawansowanych aplikacji, natomiast drugi mogący pracować z taktowaniem do 24 MHz, jest przeznaczony do aplikacji niskobudżetowych. Pomimo iż cena STM32F101 była niższa niż układów opartych na ARM7TDMI (około 13 zł w handlu detalicznym), to jednak dla nieskomplikowanych aplikacji nadal nie była konkurencyjna w porównaniu z układami wyposażonymi w rdzeń 8-bitowy. Sytuacja zaczęła się stopniowo zmieniać, począwszy od kryzysu w 2008 r., kiedy wiele fabryk półprzewodników zostało zamkniętych. Spowodowało to wywindowanie cen mikrokontrolerów 8-bitowych (np. za ATmega8 trzeba było zapłacić ponad dziesięć złotych). W międzyczasie pojawiły się pierwsze mikrokontrolery z rdzeniem Cortex-M0, które kosztowały kilkanaście złotych, a STM wyprodukowało również tańsze odmiany układów z rdzeniem Cortex -M3 (STM32F100) w cenie kilkunastu złotych. Rynek nie znosi próżni, więc ta sytuacja i dostępność tanich narzędzi spowodowały zwrot ku tanim rozwiązaniom 32-bitowym. Czy będzie to druga rewolucja ARM-owa, która spowoduje zaprzestanie stosowania mikrokontrolerów 8-bitowych? Czas pokaże, jednak wszystko na to wskazuje.

Wybór mikrokontrolera do aplikacji niskobudżetowej, problemy z portami

Autor artykułu stanął przed problemem wyboru układu do przenośnego przyrządu pomiarowego, gdzie wymagane było około 32 kB pamięci Flash dla implementacji funkcji obliczeń zmiennoprzecinkowych, przetwornik A/C, przynajmniej trzy układy czasowo-licznikowe (w tym dwa z nich mające możliwość zliczania impulsów zewnętrznych) oraz jeden generator sygnału PWM. Ponieważ większość układów z pamięcią 32 kB dostępnych na rynku ma wymagane układy peryferyjne, głównym kryterium wyboru była cena. W **tabeli 1** zamieszczono wykaz najbardziej popularnych mikrokontrolerów z pamięcią Flash o pojemności 32 kB spełniające powyższe kryteria.

Rzut oka na wykaz w tab. 1 pokazuje, że ceny popularnych mikrokontrolerów 8-bitowych mocno zniechęcają do stosowania ich w nowych konstrukcjach. Jedynie mikrokontrolery z rodziny STM8 są atrakcyjne cenowo. Jednak zastosowanie mikrokontrolera z tej rodziny tworzy szereg problemów do przezwyciężenia. Mikrokontrolery te nie są zbyt popularne przez co problemem staje się skompletowanie odpowiednich narzędzi (np. brak kompilatora GCC). W związku z tym lepszym wyborem wydaje się być np. STM32F100R6T6B. Ten mikrokontroler jest dostępny w obudowie TQFP64, ma 5 układów czasowo-licznikowych, w tym jeden mogący pracować w trybie generatora sygnału PWM oraz przetwornik A/D. Ten wybór wydaje się idealny: niska cena, wszystkie potrzebne peryferia Ale czy na pewno, nie jest to rozwiązanie pozbawione wad?

Problemy z przetwornikiem A/C przy zasilaniu mikrokontrolera bezpośrednio z baterii

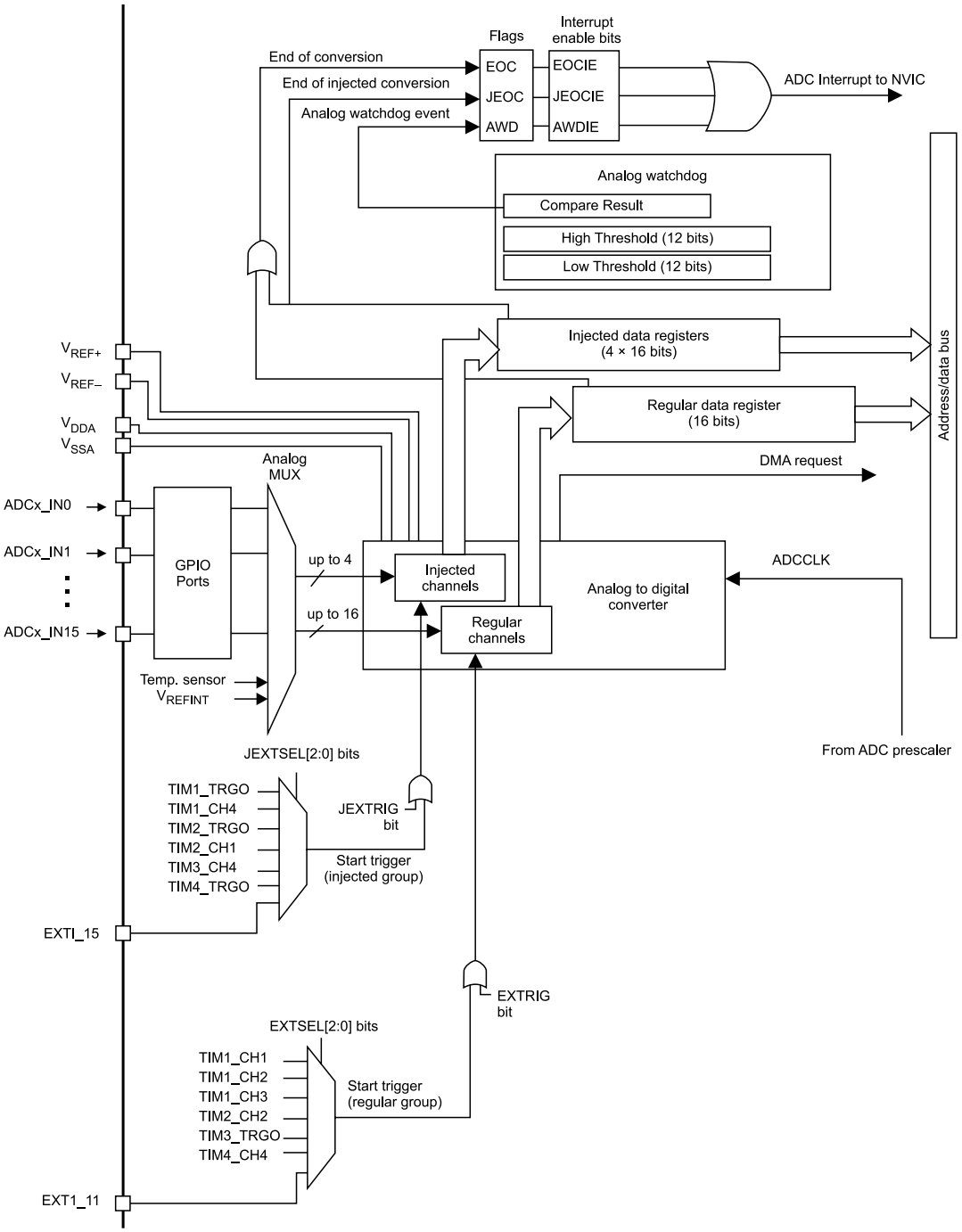
Mikrokontrolery STM32F100x wydają się wręcz idealne do zasilania bezpośrednio z pojedynczego ogniwa LiIon, którego nominalne napięcie pojedynczej celi wynosi około 3,6 V, natomiast napięcie dopuszczalne, końcowe (przy rozładowaniu) około 2,4 V. Jeżeli chcemy układ zasilać bezpośrednio z takiego ogniwa, to przy korzystaniu z przetwornika A/C konieczne będzie użycie dodatkowego źródła napięcia referencyjnego. Na **rysunku 1** przedstawiono wewnętrzną budowę przetwornika A/C mikrokontrolerów rodziny STM32F100

Przetwornik ma wewnętrzne źródło napięcia referencyjnego 1,2 V, ale nie jest ono doprowadzone do wejść Vref przetwornika, a jedynie dołączone wewnętrznie do kanału ADC16. Jedyną możliwością użycia napięcia referencyjnego jest dołączenie zewnętrznego źródła referencyjnego do wyprowadzeń Vref+

Tabela 1. Wykaz mikrokontrolerów spełniających kryteria wymienione w artykule

Typ	Rodzina/Architektura	FLASH/RAM	Orientacyjna cena detaliczna
ATmega32	AVR/8-bitowa	32 kB/2kB	15,87 PLN
ATmega32U4	AVR-8/8-bitowa	32 kB/2,5 kB	26,73 PLN
AT89C51RC-24	MCS51/8-bitowa	32 kB/0,5 kB	15,33 PLN
PIC18F4550	PIC/8-bitowa	32 kB/2 kB	23,62 PLN
STM8S105S6T6C	STM8/8-bitowa	32 kB/2 kB	7,76 PLN
LPC1114FBD48/302	ARM-Cortex-M0/32-bitowa	32 kB/4 kB	10,06 PLN
STM32F100R6T6B	ARM-Cortex-M3/32-bitowa	32 kB/4 kB	9,78 PLN
STM32F100VBT6B	ARM-Cortex-M3/32-bitowa	128 kB/16 kB	21,65 PLN

oraz Vref–. Trudno zrozumieć, dlaczego producent mając do dyspozycji wewnętrzne źródło referencyjne nie daje możliwości wyboru wewnętrznego źródła jako napięcia referencyjnego zwłaszcza, że jest to rozwiązanie standardowe, dostępne nawet w najprostszych mikrokontrolerach np. ATtiny. Niestety, jeżeli spojrzymy na wyprowadzenia układu w obudowie TQFP64 zobaczymy, że wejścia Vref+, Vref– nie są dostępne w tym rodzaju obudowy i są one dołączone wewnętrznie do zasilania bloków analogowych VDDA oraz VSS. Wyprowadzenia Vref dostępne są jedynie w obudowach TQFP100. Jeżeli teraz spojrzymy na tab. 1, to zauważymy, że taki mikrokontroler jest ponad dwukrotnie droższy oraz ma dużo większe zasoby np. pamięci FLASH niż to potrzebne w naszym projekcie (patrz specyfikacja wymagań). Czy jednak tylko z powodu braku wyprowadzeń będziemy zmuszeni do sięgnięcia po droższy mikrokontroler? Niekoniecznie.



Rysunek 1. Schemat blokowy przetwornika A/C mikrokontrolerów z rodziny STM32F100

Jedną z możliwości jest użycie, dodatkowego stabilizatora napięcia VDDA dla całego bloku analogowego. Producent deklaruje jednak, że napięcie VDDA oraz VDD muszą mieć taką samą wartość, a więc należy stabilizować napięcie zasilające cały mikrokontroler. Minimalne, dopuszczalne napięcie zasilające części analogowej VDDA wynosi 2,4 V, co przy zasilaniu mikrokontrolera z pojedynczego ogniwa LiIon jest kłopotliwe, ponieważ wymaga użycia dodatkowego stabilizatora.

Inną możliwością jest rozwiązanie problemu programowo z wykorzystaniem wewnętrznego źródła napięcia referencyjnego. Jak wiadomo, wartość napięcia zmierzonego za pomocą przetwornika A/C wyraża się za pomocą wzoru:

$$V_{ADC} = \frac{VAL}{(2^N - 1)} \cdot V_{ref}$$

gdzie:
N: rozdzielczość bitowa przetwornika,
VAL: liczba odczytana z przetwornika analogowo cyfrowego,
Vref: wartość napięcia referencyjnego.

Jeżeli chcemy zmierzyć napięcie zasilające procesor **VDDA**, wystarczy do wejścia przetwornika A/C dołączyć wewnętrzne źródło napięcia referencyjnego, (ustawić multiplekser na kanał CHN16) i odwrócić rolę napięcia wejściowego i referencyjnego. Po prostym przekształceniu wzoru (1), możemy wyznaczyć napięcie zasilające układ VDDA, na podstawie wartości odczytanej z przetwornika:

$$V_{ref} = V_{DDA}, V_{ADC} = V_{intref}$$

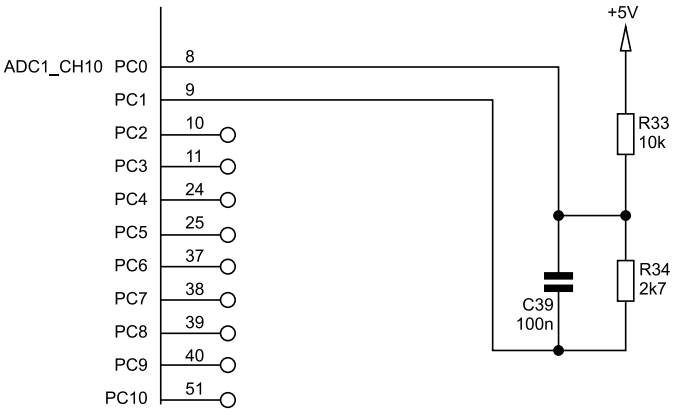
$$V_{DDA} = \frac{(2^N - 1) \cdot V_{refint}}{VAL}$$

gdzie:
Vdda: napięcie zasilające,
Vrefint: wartość wewnętrznego napięcia referencyjnego,
VAL: liczba odczytana z przetwornika A/C.

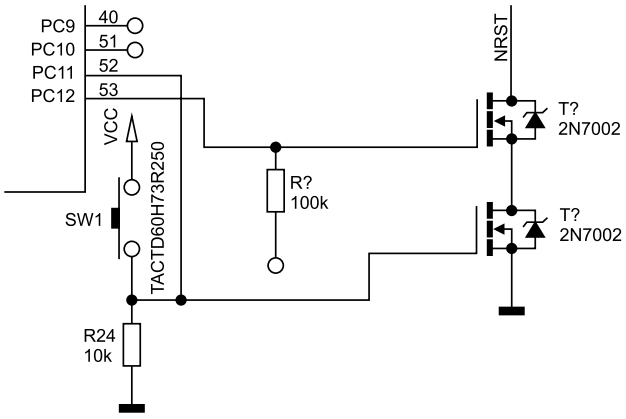
Tym sposobem mierząc napięcie VDDA możemy wyznaczyć wartość napięcia baterii zasilającego mikrokontroler. Dla STM32F100, w którym N=12 i Vrefint=1,2 V, wartość napięcia zasilającego wyraża się za pomocą wzoru:

$$V_{DDA} = \frac{4095 \cdot 1{,}2\ V}{VAL}$$

Po pogodzeniu się z mniejszą dokładnością, istnieje możliwość pomiaru nie tylko napięcia zasilającego układ, ale również wartości napięcia z innych kanałów. Możemy tutaj wykorzystać tryb, pracy przetwornika z automatycznym skanowaniem kilku



Rysunek 2. Schemat połączeń dla przykładu ilustrującego pomiar napięcia baterii



Rysunek 3. Schemat rozwiązania używającego linii RST do wyprowadzenia CPU z trybu uśpienia

kanałów jeden po drugim (*Multichannel single conversion mode*). Przetwornik należy skonfigurować tak, aby w jednym cyklu pomiarowym skanowane były kanały, w których chcemy zmierzyć wartość napięcia oraz kanał wewnętrznego napięcia referencyjnego (CHN16). Przy założeniu, że podczas pojedynczego skanowania napięcie Vdda nie zmieni się, możemy wyznaczyć napięcie dla poszczególnych kanałów przetwornika:

$$V_{mchx} = \frac{VAL_{chx}}{(2^N-1)} \cdot V_{DDA}$$

$$V_{refint} = \frac{VAL_{chref}}{(2^N-1)} \cdot V_{DDA}$$

W wyniku czego otrzymujemy wzór:

$$V_{mchx} = \frac{VAL_{chx}}{VAL_{chref}} \cdot V_{refint}$$

gdzie:

Vmchx: liczba odpowiadająca napięciu zmierzonemu w kanale chx,

VALchx: liczba odczytana z przetwornika w kanale chx,

Vrefint: wartość napięcia referencyjnego.

Wystarczy zatem odczytać wskazania przetwornika dla kanału napięcia referencyjnego oraz wskazania przetwornika dla mierzonych kanałów, a następnie podstawić do wzoru (4). Praktyczne próby wykazały, że dokładność tej metody jest wystarczająca dla większości zastosowań.

Posłużymy się teraz przykładem praktycznym, który pokaże w jaki sposób odczytać wartość napięcia zasilającego (baterii), oraz dodatkowe napięcie np., na wyjściu dodatkowej przetwornicy. Schemat aplikacji dla tego przykładu pokazano na **rysunku 2**.

Do wejścia przetwornika ADC1_CH10 jest dołączony dzielnik rezystorowy R33/R34. Rezystor R34 nie jest dołączony do masy układu, a do portu GPIO. Kosztem nieco mniejszej dokładności pomiaru zyskujemy możliwość odłączenia dzielnika w stanie uśpienia, ponieważ jego prąd byłby wielokrotnie większy od prądu pobieranego przez mikrokontroler. Kod programu, realizującego pomiar napięcia wspomnianą wcześniej metodą przedstawiono na **listingu 1**.

Przed rozpoczęciem pomiarów należy zainicjalizować przetwornik A/C, w trybie *multichannel*, *single conversion mode*. Użycie trybu wielokrotnej konwersji, wymaga zastosowania kontrolera DMA, tak aby wyniki pomiarów można było przesłać do pamięci bez angażowania jednostki centralnej. Na początku włączane są sygnały zegarowe dla przetwornika A/C oraz jest konfigurowany port PC0, tak aby pełnił rolę wejścia ADC10 przetwornika A/C. Kolejną czynnością jest przełączenie przetwornika ADC1 w tryb wielokanałowej konwersji pojedynczej, a następnie konfigurowanie kanałów ADC10 (napięcie mierzone) oraz ADC17 (napięcie referencyjne). Teraz jest włączany przetwornik A/C i jest wykonywane kalibrowanie przetwornika.

I to chyba wyjaśnia, dlaczego wewnętrzne napięcie jest dołączone do jednego z wejść przetwornika, a nie do Vref – czyżby ST nie potrafiło na czas poradzić sobie z przetwornikiem A/C niewymagającym kalibracji? Czyżby kolejny raz wygrał „time to market"?

Następnie jest konfigurowany kontroler DMA oraz włączane wewnętrzne źródło napięcia referencyjnego. Kolejną czynnością jest włączenie systemu przerwań. W momencie wywołania funkcji: stm32::adc_software_start_conv_cmd(ADC1, true); przetwornik rozpocznie wykonanie pomiaru dla kanałów CH10 a następnie CH17, po którego zakończeniu zostanie zgłoszone przerwanie od kontrolera DMA (**listing 2**).

Teraz w metodzie read_adc_values() możemy z tablicy m_adc_regs odczytać wartości A/C z poszczególnych kanałów i na tej podstawie wyznaczyć wartość napięcia zasilającego, oraz wartość napięcia wejściowego kanału CH10 co pokazano na **listingu 3**.

Listing 1. Kod programu realizujący pomiar napięcia

```
/* ---------- Configure ADC1 and DMA for transfer --------- */
  //Enable ADC1
  stm32::rcc_periph_clock_cmd( stm32::rcc_clk_apb2, RCC_APB2Periph_ADC1, true );
  //Configure GPIO as analog inputs
  stm32::io_config( ADC1_CH10_PORT, ADC1_CH10_PIN, stm32::GPIO_MODE_INPUT, stm32::GPIO_CNF_IN_ANALOG );
  //Configure ADC
  stm32::adc_init( ADC1, ADC_Mode_Independent, true, false,
  ADC_ExternalTrigConv_None, ADC_DataAlign_Right, 2 );
  stm32::adc_regular_channel_config( ADC1, ADC_Channel_10, 1, ADC_SampleTime_55Cycles5 );
  stm32::adc_regular_channel_config( ADC1, ADC_Channel_17, 2, ADC_SampleTime_55Cycles5 );
  stm32::adc_cmd( ADC1, true );
  stm32::adc_temp_sensor_vref_int_cmd( true );
  stm32::adc_dma_cmd(ADC1, true);
  stm32::adc_reset_calibration(ADC1);
  while(stm32::adc_get_reset_calibration_status(ADC1));
  stm32::adc_start_calibration(ADC1);
  while(stm32::adc_get_calibration_status(ADC1));
  //Configure DMA channel in ADC1
  stm32::dma_enable(stm32::DMACNTR_1);
  stm32::nvic_set_priority( DMA1_Channel1_IRQn, IRQ_PRIO, IRQ_SUBPRIO );
  stm32::nvic_irq_enable( DMA1_Channel1_IRQn, true );
  stm32::dma_channel_config(DMA1_Channel1,
  DMA_DIR_PeripheralSRC | DMA_PeripheralInc_Disable |
  DMA_MemoryInc_Enable | DMA_PeripheralDataSize_HalfWord |
  DMA_MemoryDataSize_HalfWord | DMA_Mode_Circular | DMA_Priority_High | DMA_M2M_Disable, m_adc_regs, &ADC1->DR, 2);
  stm32::dma_irq_enable(DMA1_Channel1,DMA_IT_TC);
  stm32::dma_channel_enable(DMA1_Channel1);
```

Listing 2. Obsługa przerwania od kontrolera DMA

```
extern "C" {
  void dma1_channel1_isr_vector(void) __attribute__((interrupt));
  void dma1_channel1_isr_vector(void)
  {
    stm32::dma_clear_flag( DMA1_FLAG_TC1);
    if( p_conv_ptr )
    p_conv_ptr->read_adc_values();
  }
}}
```

Listing 3. Wyznaczenie wartości napięcia zasilania

```
  //Get vbat in mV
  int get_Vbat() const
  {
    return (ADC_MAXVAL * VREF_INT_MV)/m_adc_regs[VREF_POSTAB];
  }
  //Get vzas in mV
  int get_Vconv() const
  {
    return (((m_adc_regs[VIN_POSTAB]*VREF_INT_MV)/
    m_adc_regs[VREF_POSTAB] ) * (64*(R1_DIV_VAL+R2_DIV_VAL)/R1_DIV_VAL))/64;
  }
private:
  enum { VIN_POSTAB, VREF_POSTAB };
  static const int VREF_INT_MV = 1200;
  static const int ADC_MAXVAL = 4095;
  static const int R1_DIV_VAL = 2700;
  static const int R2_DIV_VAL = 10000;
```

Metoda get_Vbat() odczytuje wartość napięcia zasilania i oblicza ją korzystając z wzoru (2). Aby uniknąć wykonywania zbędnych operacji zmiennoprzecinkowych, napięcie wyjściowe jest wyrażane w mV. Metoda get_Vconv() pozwala wyznaczyć napięcie wyjściowe na wejściu dzielnika na podstawie wzoru (4) oraz dodatkowo wykonywane są obliczenia związane z dzielnikiem napięcia. Również i w tym wypadku wykorzystano liczby całkowite, oraz zastosowano dodatkowe skalowanie (64), aby uzyskać maksymalną dokładność wyniku.

Problemy z wykorzystaniem urządzeń peryferyjnych w mikrokontrolerach z małymi obudowami

Kolejnym problemem STM32F1 jest mało elastyczny sposób mapowania wyprowadzeń funkcji bloków peryferyjnych do portów GPIO, który dodatkowo dla niektórych peryferiów jest nieprzemyślany. Układy STM32F1 mają ustalone na „sztywno" porty, do których dołączone są doprowadzenia bloków peryferyjnych. Przypisania te możemy co prawda zamienić z użyciem bloku AFIO (*Alternate Function IO*), niemniej jednak w wielu wypadkach wyprowadzenie danego układu peryferyjnego ma tylko jedno wyjście lub wejście alternatywne. Problem staje się szczególnie uciążliwy przy stosowaniu mniejszych układów w obudowach TQFP48 oraz TQFP64, gdzie często okazuje się, że alternatywne wyprowadzenie jest niedostępnie, natomiast wyprowadzenie podstawowe jest zajęte przez inny, używany w tej aplikacji układ peryferyjny. W takiej sytuacji możemy być zmuszeni do wybrania mikrokontrolera w większej obudowie, droższego i to tylko dlatego, że nie pasuje nam rozmieszczenie wyprowadzeń, a nie z powodu np. niewystarczających zasobów czy mocy obliczeniowej.

Przykładem braku elastyczności wyprowadzeń jest np. mikrokontroler STM-32F100R6T6B, który jest dostępny między innymi w obudowie LQFP64. Jeżeli chcemy zliczać impulsy zewnętrzne za pomocą timera TIM2, będziemy musieli użyć wejścia TIM2_ETR. Natomiast jeżeli chcemy użyć trybu uśpienia *Standby*, do realizacji programowego wyłącznika urządzenia będziemy musieli wykorzystać wejście WKUP. Patrząc na wyprowadzenie możemy zauważyć, że wejście WKUP jest dostępne tylko na porcie PA0. Spójrzmy teraz na wejście TIM2_ETR (**tabela 2**).

Jak łatwo możemy zauważyć jedyną możliwością użycia wejścia TIM2_ETR jest wykorzystanie pinu PA0 lub PA15. Ale niestety pinu PA0 nie będziemy mogli użyć, ponieważ chcemy go wykorzystać do wybudzania mikrokontrolera. Jedyną możliwością jest zatem użycie doprowadzenia PA15. Z kolei ono jest wykorzystywane jako jedno z wyprowadzeń złącza JTAG, a więc w wypadku, gdy nie chcemy rezygnować z możliwości używania JTAG'a, również i ta opcja nie jest możliwa do zrealizowania. Jest to ewidentny przykład nieprzemyślanego rozwiązania połączonego z brakiem elastyczności mapowania funkcji układów peryferyjnych.

Możliwości rozwiązania wyżej wspomnianego problemu jest co najmniej kilka:

- Rezygnacja z debugowania za pomocą JTAG lub użycie interfejsu SWD.

Tabela 2. Funkcje wybranych wyprowadzeń mikrokontrolera STM-32F100R6T6B w obudowie LQFP64

Wyprowadzenie	Funkcje
PA0	WKUP/USART2_CTS ADC1_IN0/TIM2_CH1_ETR
PA15	TIM2_CH1_ETR/ SPI1_NSS/JTDI

- Rezygnacja z używania trybu *Sleep* i użycie trybu *Stop Mode* kosztem zwiększonego poboru prądu (około 14 μA zamiast 2,5 μA w stanie uśpienia); w trybie STOP dowolna linia GPIO może wyprowadzić rdzeń z uśpienia.
- Użycie linii *Reset* do wyprowadzenia procesora z trybu *Sleep*, co niestety wymaga dodatkowego układu oraz dodatkowej linii GPIO, która będzie blokować aktywację zerowania po uruchomieniu mikrokontrolera.

Przykład takiego układu używającego linii *RST* do wyprowadzenia CPU z trybu uśpienia przedstawiono **rysunku 3**. Przycisk SW1 służy do obsługi urządzenia oraz do realizacji programowego wyłącznika urządzenia. Gdy układ znajduje się w trybie uśpienia, wejście PC12 jest w stanie wysokiej impedancji, a rezystor podciągający powoduje otwarcie kanału górnego tranzystora. Wciśnięcie przycisku SW1 powoduje otwarcie kanału dolnego tranzystora, w wyniku czego wejście *NRST* mikrokontrolera jest zwierane do masy. Powoduje to restart i uruchomienie mikrokontrolera. Po uruchomieniu się programu linia PC12 jest zerowana, co powoduje zablokowanie górnego tranzystora umożliwiając tym samym normalne korzystanie z przycisku SW1 poprzez odczyt stanu linii PC11, bez generowania sygnału RST.

Zakończenie

Mikrokontrolery STM32F100 krok po kroku zdobywają rynek niskobudżetowych projektów, gdzie do tej pory królowały mniejsze „ośmiobitowce". Czy im się uda pokaże czas. Dużą bolączkę tych układów stanowi problem z wyprowadzeniami układów peryferyjnych Czy jest to celowe działanie producenta, tak aby użytkownik był zmuszony wybrać większy mikrokontroler, gdy wcale tego nie potrzebuje? A może w pośpiechu za konkurencją wydając w błyskawicznym tempie kolejne wersje coraz bardziej rozbudowanych mikrokontrolerów producentom nie zostaje wiele czasu, aby dokładnie przemyśleć własne rozwiązania pod względem ergonomii?

Lucjan Bryndza, EP