Mikrokontrolery STM32

Wykorzystanie ADC i DMA

Otaczająca nas rzeczywistość ma postać analogową, zatem każdy system mikroprocesorowy, który ma pracować w oparciu o informacje pochodzące z tej rzeczywistości, musi być wyposażony w przetworniki A/C. Większość z obecnie produkowanych mikrokontrolerów posiada już wbudowane ADC na tyle dobrej jakości, że często są one wystarczające dla poprawnego działania aplikacji. Odpada zatem konieczność stosowania zewnętrznych przetworników, co poprawia niezawodność i upraszcza budowę urządzenia. W artykule przestawiamy w jaki sposób rozpocząć pracę z ADC i DMA wbudowanymi w mikrokontrolery STM32. Wszystkie projekty zostały przygotowane i uruchomione na płycie ewaluacyjnej STM3210B-EVAL.

Każdy z mikrokontrolerów, należący do rodziny STM32, jest wyposażony w przynajmniej jeden przetwornik analogowo-cyfrowy oraz sprzętowy kontroler DMA. Aby oswoić się nieco z przetwarzaniem A/C, uruchomimy na początek prostą aplikację. Jej zadaniem będzie pokazywanie na wyświetlaczu LCD w formie wykresu U=f(t), wartości napięcia na doprowadzeniu PC4, do którego dołączony jest potencjometr RV1. Poznamy sposób, w jaki przetworniki są konfigurowane i obsługiwane, co pozwoli na zbudowanie bardziej skomplikowanych aplikacji. Gdy ADC będzie już pracował zgodnie z założeniami, to wykorzystamy możliwości jakie drzemią w kontrolerze DMA w połączeniu z ADC. Materiały do projektów są dostępne na stronie *paprocki.wemif.net* oraz na płycie CD-EP1/2009B dołączonej do numeru. W pierwszej kolejności jednak zapoznamy się nieco bliżej z budową przetworników A/C, w które wyposażone są mikrokontrolery STM32.

Budowa przetwornika analogowo–cyfrowego

Zamontowany na płytce ewaluacyjnej układ STM32F103VB ma wbudowane dwa 12-bitowe 16-kanałowe ADC, które mogą pracować w wielu różnych trybach. Na **rys. 1** przedstawiono uproszczoną budowę przetwornika analogowo–cyfrowego zaimplementowanego w wykorzystywanym przez nas mikrokontrolerze. Nasz bohater ma wbudowane dwa takie przetworniki oznaczone jako ADC1 i ADC2. Firma ST umieszczając w swoich produktach wielokrotne, autonomiczne przetworniki analogowo-cyfrowe, zrobiła ukłon w kierunku konstruktorów wykorzystujących w swoich aplikacjach bezszczotkowe silniki trójfazowe prądu stałego. W tego typu rozwiązaniach, aby kontrolować parametry pracy silnika, należy wykonać dwa pomiary prądu dokładnie w tym samym czasie. Oczywistym jest, że taka jednoczesna praca obydwu przetworników może być wykorzystywana również wszędzie tam, gdzie jest wymagany równoczesny pomiar kilku napięć (lub pośrednio – prądów).

Wbudowane w mikrokontroler przetworniki A/C są wyposażone w układy kalibracji. Dzięki nim znacznie zmniejsza się błąd przetwarzania wynikający z niedokładności pojemności kondensatorów pamiętających próbkowane napięcie. Typowo pojemność takich kondensatorów wynosi 12 pF, jednak wykonywane są one z pewną tolerancją, zatem wartość pojemności może odbiegać od deklarowanej. Wpływ różnic pojemności na wynik pomiaru niwelowany jest w czasie kalibracji.

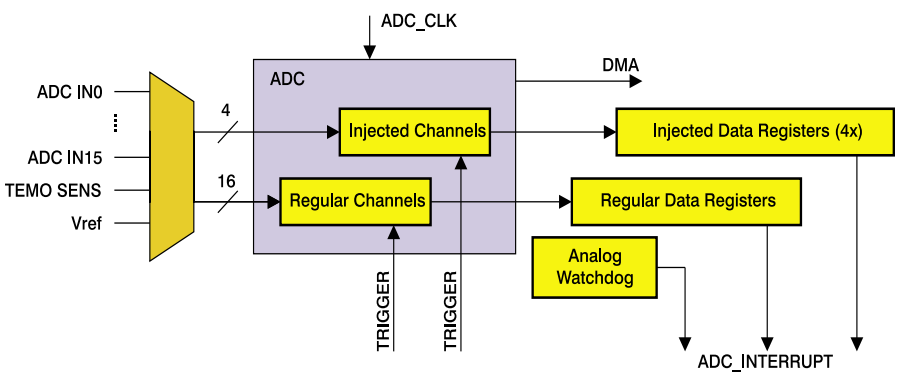
Z rys. 1 wynika, że ADC może przetwarzać sygnały w dwóch grupach: *regular group* (regularna) oraz *injected group* („wstrzykiwana"). Wyjaśnimy pokrótce, na czym polegają różnice w obsłudze tych dwóch grup.

Regular group – jest to grupa podstawowa, do której możemy przypisać do szesnastu kanałów pomiarowych, w odróżnieniu od *injected group*, do której mogą być przypisane maksymalnie cztery. Zasadnicza różnica pomiędzy tymi dwiema grupami polega na tym, że konwersja kanałów należących do *injected group* ma wyższy priorytet, niż *regular group*. Jeżeli wykonanie przetwarzania A/C jest krytyczne i nie może być mowy o jakichkolwiek opóźnieniach, to wówczas należy konwersję wykonać przy pomocy kanałów ADC należących do *injected group*. Jeżeli konwersja *regular group* jest w trakcie wykonywania i MCU otrzyma żądanie wykonania konwersji grupy „wstrzykiwanej", to wtedy następuje zawieszenie przetwarzania na jej rzecz. W momencie, gdy proces jej przetwarzania zostanie zakończony, to wywłaszczona konwersja zostaje wznowiona od momentu jej przerwania. Zachowanie to pokazano na **rys. 2**. Ponadto *injected group* ma oddzielne rejestry danych dla każdego z kanałów pomiarowych, czyli w sumie cztery, co zaznaczono na rys. 1.

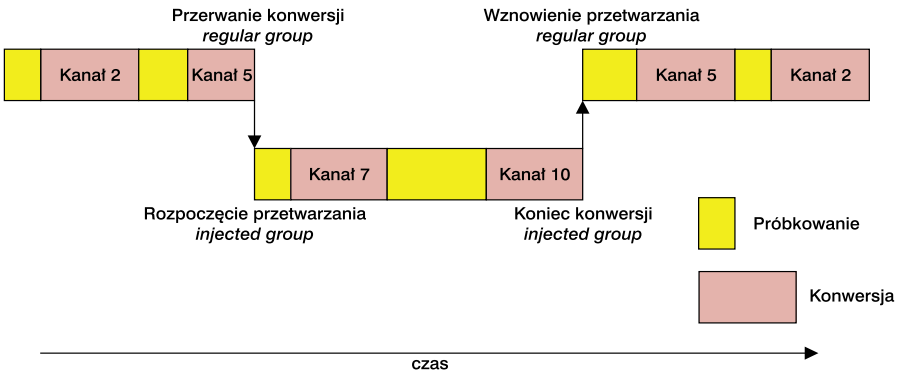
Każda nieco bardziej zaawansowana aplikacja wykorzystująca przetworniki A/C, wymaga specyficznego podejścia. Z tego powodu producenci mikrokontrolerów implementują w swoich ADC coraz bardziej wymyślne tryby ich działania. Jest to najczęściej ściśle związane z możliwymi zastosowaniami, do których przeznaczony jest mikrokontroler. Również i w STM32 mamy do dyspozycji kilka trybów działania ADC:

- ciągła lub jednorazowa konwersja pojedynczego kanału, lub wielu kanałów,
- tryb nieciągły (*discontinuous*),
- jednoczesna praca dwóch przetworników,
- wyzwalanie przetwornika za pomocą timera lub zewnętrznego przerwania.

Przetwornik jest również wyposażony w sprzętowy, analogowy Watchdog, który ma ustawiane progi (niski i wysoki), po przekroczeniu których może być generowane przerwanie. Do tego wszystkiego możemy również ustawić czas próbkowania sygnału. Dla potrzeb projek-



Rys. 1. Uproszczony schemat blokowy przetwornika A/D



Rys. 2. Ilustracja przerwy w konwersji grupy sygnałów *regular* na rzecz *injected*

List. 1. Program odczytujący napięcie przyłożone do PC4

```
void RCC_Conf(void);
void NVIC_Conf(void);
void GPIO_Conf(void);
void SysTick_Conf(void);

int index = 0;
int wyniki[320] = {0};

int main(void)
{
   ADC_InitTypeDef ADC_InitStruct;

   RCC_Conf(); NVIC_Conf(); GPIO_Conf();
   SysTick_Conf();        // SysTick wykorzystywany przez funkcje Delay()

   // Jeden przetwornik, pracujacy niezaleznie
   ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;

   // Pomiar jednego kanalu, wylacz opcje skanowania
   ADC_InitStruct.ADC_ScanConvMode = DISABLE;

   // Wlacz pomiar w trybie ciaglym
   ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;

   // Nie bedzie wyzwalania zewnetrznego
   ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

   // Dane wyrownane do prawej - znaczacych bedzie 12 mlodszych bitow
   ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;

   // Jeden kanal
   ADC_InitStruct.ADC_NbrOfChannel = 1;

   // Inicjuj przetwornik
   ADC_Init(ADC1, &ADC_InitStruct);

   // Grupa regularna, czas probkowania 71,5 cykla czyli 5,1us
   ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_71Cycles5);

   // Wlacz ADC1
   ADC_Cmd(ADC1, ENABLE);

   // Resetuj rejestry kalibracyjne
   ADC_ResetCalibration(ADC1);
   // Czekaj, az skonczy resetowac
   while(ADC_GetResetCalibrationStatus(ADC1));

   // Start kalibracji ADC1
   ADC_StartCalibration(ADC1);
   // Czekaj na zakonczenie kalibracji ADC1
   while(ADC_GetCalibrationStatus(ADC1));

   // Start przetwarzania
   ADC_SoftwareStartConvCmd(ADC1, ENABLE);

   // Inicjalizuj LCD
   STM3210B_LCD_Init();
   // Wyczysc LCD, tlo niebieskie
   LCD_Clear(Blue);

   while(1)
   {
         Delay(1);  // Odswierzanie co 10ms
         if(index==320) index=0;   // wyswietlacz posiada 320 kolumn

         // Czyszczenie LCD ze starych danych
         LCD_SetCursor(wyniki[index], 320 - index);
         LCD_WriteRAMWord(Blue);

         // Odczytanie wartosci a ADC i obliczenia:
         // 12 bitow = 4096 poziomow
         // 240 wierszy LCD, zatem 4096/240 = 17
         wyniki[index] = ADC_GetConversionValue(ADC1) / 17;

         LCD_SetCursor(wyniki[index], 320 - index); // rysowanie punktow
         LCD_WriteRAMWord(Red);
         index++;
   }
}
```

tów przykładowych przedstawionych w artykule zostanie użyty tryb ciągły, z pomiarem jednego lub dwóch kanałów.

Pomiar w trybie ciągłym

Na **list. 1** przedstawiono program, który realizuje odczyt napięcia na nóżce PC4 mikrokontrolera (wyprowadzenie 33 dla obudowy LQFP100). Na płytce STM3210B-EVAL wyprowadzenie to jest podłączone do potencjometru RV1. Wartość napięcia zasilającego PC4 wskazywana jest na graficznym wyświetlaczu LCD w formie wykresu U=f(t).

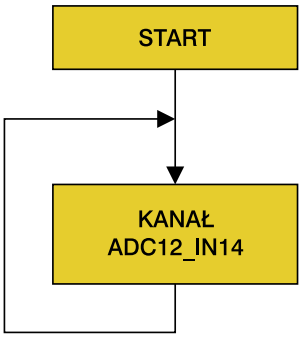
Z noty katalogowej mikrokontrolera STM32F103VB można odczytać, że domyślną funkcją alternatywną tego wyprowadzenia jest ADC12_IN14, zatem będziemy korzystać z 14 kanału przetwornika. W tym przykładzie (jak również w pozostałych) są wykorzystywane standardowe funkcje obsługi LCD dostarczane przez firmę ST. Są dobrze opisane w dokumentacji, toteż nie będziemy się nimi szczegółowo zajmować.

Aby przetwornik analogowo–cyfrowy działał poprawnie, należy go w pierwszej kolejności skonfigurować odpowiednio do potrzeb aplikacji. Będziemy korzystać z pierwszego przetwornika ADC1, o czym poinformujemy MCU w odpowiednim miejscu. Tak, jak miało to miejsce w przypadku innych układów peryferyjnych (artykuły w EP11/08 i EP12/08), konfigurowanie ADC odbywa się poprzez wypełnianie pól struktury inicjującej i późniejsze jej przekazanie do odpowiedniej funkcji.

Po utworzeniu zmiennej *ADC_InitStruct* rozpoczynamy wypełnianie jej pól. Pierwszy parametr określa, czy przetwornik ma pracować samodzielnie, czy też wraz z drugim ADC. Dla nas interesujący jest tryb pracy niezależnej (*independent*). Ponieważ przetwarzamy tylko jeden kanał, to wyłączamy opcję skanowania wielu kanałów oraz włączamy pracę ciągłą (*continuous*). Taki sposób pracy jest symbolicznie przedstawiony na **rys. 3**.

W naszym przykładzie nie będziemy korzystać z wyzwalania ADC za pomocą np. któregoś z timerów, czy też przerwania zewnętrznego, zatem informujemy o tym MCU.

Wbudowany w mikrokontrolery STM32 przetwornik daje możliwość programowego ustalenia, czy dane zapisywane do rejestru DR (*Data Register*) przetwornika mają być wyrów-



Rys. 3. Uproszczony schemat pracy A/D w trybie pomiaru pojedynczego kanału

nywane do lewej, czy do prawej strony. Tutaj używamy standardowego wyrównywania do prawej, zatem znaczących jest 12 młodszych bitów rejestru DR. Ostatnią informacją konfiguracyjną jest deklaracja liczby wykorzystywanych kanałów przetwornika. Jako powiedziano wcześniej, w naszym przypadku pomiar będzie wykonywany z użyciem pojedynczego kanału.

Analogicznie jak dla innych układów peryferyjnych, nazwa funkcji inicjujące jest zbieżna z nazwą układu peryferyjnego i nastawy przetwornika analogowo-cyfrowego wprowadzane są pomocą funkcji *ADC_Init()*.

Po konfiguracji przetwornika, należy dokonać wyboru, czy ADC ma pracować w grupie *injected*, czy *regular*. Służy do tego funkcja *ADC_RegularChannelConfig()*. W naszym przykładzie przetwarzany jest jeden kanał w grupie regularnej. Ponadto poprzez tę samą funkcję jest ustalany czas, jaki będzie przeznaczony na próbkowanie sygnału.

Po wykonaniu wszystkich niezbędnych wstępnych czynności konfiguracyjnych, włączamy przetwornik ADC1 wywołując funkcję *ADC_Cmd()*. Aby uzyskać możliwie dokładny wynik przetwarzania, musimy jeszcze dokonać kalibracji ADC. W związku z tym, najpierw zerujemy ustawienia kalibracyjne, czekamy na wykonanie tego polecenia, a następnie każemy przetwornikowi skalibrować się i również czekamy na zakończenie operacji. Teraz można już rozpocząć (programowo) właściwe przetwarzanie, za co odpowiada funkcja *ADC_SoftwareStartConvCmd()*. Od tego momentu rejestr danych DR jest aktualizowany cyklicznie, po zakończeniu każdej konwersji. Po odczytaniu jego wartości możemy już ją według potrzeb dalej przetwarzać. W tym przykładzie, po przeliczeniach jest ona wyświetlana na LCD w postaci wykresu. W efekcie otrzymujemy bardzo prosty rejestrator przebiegów analogowych, w którym długość rekordu wynosi 320 próbek, a napięcie jest próbkowane co 10 ms.

Programowany czas próbkowania

Wróćmy jeszcze raz do zagadnienia czasu próbkowania. Przetwornik A/C, w jaki wyposażony jest nasz mikrokontroler umożliwia (niezależnie dla każdego kanału, patrz rys. 2) programowanie czasu, który będzie przeznaczony na próbkowanie sygnału. Ma to szczególne znaczenie przy dopasowywaniu pracy ADC do środowiska, w którym wykonywane są pomiary. Sterując czasem próbkowania można optymalnie dobierać nastawy w zależności od impedancji, jaka jest podłączona do wejścia pomiarowego przetwornika. Pozwala to na zminimalizowanie błędów wynikających z niedopasowania ADC.

Maksymalna częstotliwość, z jaką może pracować wbudowany w mikrokontrolery przetwornik analogowo-cyfrowy, wynosi 1 MHz (czas konwersji 1 μs). Aby osiągnąć taki wynik, należy ustawić częstotliwość taktowania szyny, do której jest podłączony ADC, na 14, 28 lub 56 MHz. Sam przetwornik może być taktowany z maksymalną częstotliwością równą 14 MHz. W związku z tym, że sygnały zegarowe można dzielić tylko przez wartości będące potęgą liczby dwa, maksymalna częstotliwość sygnału zegarowego rdzenia, dla którego jest możliwe osiągnięcie czasu konwersji równego 1 μs, to 56 MHz. W takim przypadku dzielnik częstotliwości ADC będzie wynosił cztery, co da w efekcie 14 MHz.

Generalnie czas konwersji jest wyznaczany z zależności:

T = programowany czas próbkowania +12,5 cyklu zegarowego

Minimalny programowany czas próbkowania jest równy 1,5 cyklu zegarowego. Podsumowując, minimalny czas konwersji wynosi 1,5+12,5=14 cykli, a skoro częstotliwość taktowania wynosi 14 MHz, to całkowity czas przetwarzania A/C wynosi 1 μs.

Żeby przetwornik pracował z takim czasem konwersji, należy w funkcji konfiguracji sygnałów zegarowych i resetu RCC_Conf(), zmodyfikować linijkę odpowiadającą za mnożnik sygnału PLL. Musi to być wykonane w taki sposób, aby z podłączonego do mikrokontrolera rezonatora 8 MHz uzyskać częstotliwość 56 MHz. Mnożnik „razy 7" można ustawić podając go jako parametr wywołania funkcji:

```
RCC_PLLConfig(RCC_PLLSource_HSE_
Div1, RCC_PLLMul_7);
```

Następnym krokiem prowadzącym do uzyskania wymaganej w naszym przypadku częstotliwości taktowania ADC równej 14 MHz, jest podzielenie częstotliwości taktującej wewnętrzną szynę danych, do której podłączony jest ADC (magistrala APB2), przez 4. Aby tego dokonać, należy w funkcji *RCC_Conf()* umieścić linię kodu:

```
RCC_ADCCLKConfig(RCC_PCLK2_Div4);
```

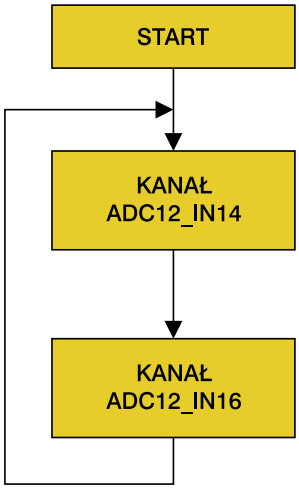
Zapewni to częstotliwość taktowania ADC1 równą 14 MHz, a w konsekwencji czas przetwarzania równy 1 μs.

Wiele kanałów w trybie ciągłym

Często aplikacja wymaga pomiaru kilku napięć. Wówczas wykorzystuje się kilka wejść pomiarowych przetwornika analogowo-cyfrowego. W przypadku mikrokontrolerów STM32 nie ma potrzeby wykonywania oddzielnych

List. 2. Program demonstrujący wyniki pomiaru napięcia na PC4 i pomiaru temperatury

```
#define ADC1_DR_Address    ((u32)0x4001244C)

int index = 0;
int wyniki_RV1[320] = {0};
u16 temperatura;
char wynik_temperatura[8] = {0};
u16 ADCVal[2];

int main(void)
{
        ADC_InitTypeDef ADC_InitStruct;

        RCC_Conf(); NVIC_Conf(); GPIO_Conf(); SysTick_Conf(); DMA_Conf();

        ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
        // Pomiar wielu kanalow, wlacz opcje skanowania
        ADC_InitStruct.ADC_ScanConvMode = ENABLE;
        ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
        ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
        ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
        // Dwa kanaly
        ADC_InitStruct.ADC_NbrOfChannel = 2;
        ADC_Init(ADC1, &ADC_InitStruct);

        ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_
71Cycles5);
        ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 2, ADC_SampleTime_
239Cycles5);

        // Wlaczenie czujnika temperatury
        ADC_TempSensorVrefintCmd(ENABLE);
        // Wlaczenie DMA
        ADC_DMACmd(ADC1, ENABLE);

        ADC_Cal(); // Kalibracja ADC1

    // Inicjalizuj LCD
        STM3210B_LCD_Init();
        // Wyczysc LCD, tlo niebieskie
        LCD_Clear(Blue);

        while(1)
        {
                Delay(1);  // Odswierzanie co 10ms
                if(index==320) index=0;   // wyswietlacz posiada 320 kolumn

                LCD_SetCursor(wyniki_RV1[index], 320 - index);
                LCD_WriteRAMWord(Blue);

                wyniki_RV1[index] = ADCVal[0] / 17;
                // Pomiar temperatury i obliczenia
                temperatura = (1430 - ADCVal[1]*0.805) / 4.3 + 25;
                LCD_SetCursor(wyniki_RV1[index], 320 - index);
                LCD_WriteRAMWord(Red);
                sprintf(wynik_temperatura, „T=%d stC", temperatura);
                // Odswierzanie temperatury co okolo 320ms
                if(!(index % 32))
                        LCD_DisplayStringLine(1,(u8*)wynik_temperatura);
                index++;
        }
}
```



Rys. 4. Uproszczony schemat pracy A/D w trybie skanowania kanałów pomiarowych

pomiarów dla każdego z kanałów. Proces ten jest zautomatyzowany. Do tego celu służy tryb przemiatania (skanowania) wejść. Wybierając ten tryb ustalamy, które kanały mają być przetwarzane, w jakiej kolejności i jaki ma być czas ich próbkowania.

Aby pokazać jak działa ten tryb, uruchomimy program, który będzie pokazywał na wyświetlaczu LCD graficzny wynik pomiaru napięcia na potencjometrze RV1 oraz temperaturę mikrokontrolera w formie liczbowej. Dodatkowo, nieco na wyrost, czasy próbkowania dla pomiaru napięcia i temperatury będą różne, a przetwornik będzie pracował w trybie ciągłym. Na **rys. 4** jest pokazano w sposób poglądowy zasada tego pomiaru, natomiast stosowny program został umieszczony na **list. 2**. Powtarzający się w stosunku do programu z list.1 kod źródłowy, został umieszczony w funkcjach, aby niepotrzebnie nie zaciemniać istotnych funkcji. Można zauważyć, że w odróżnieniu od poprzedniego przykładu, tutaj włączamy opcję skanowania (przemiatania) kanałów, informujemy, że przetwarzane będą dwa (a nie jak poprzednio jeden) kanały. To są wszystkie zmiany, jakich należy dokonać podczas wypełniania struktury inicjującej przetwornik. Następnie ustalamy grupy kanałów, ich kolejność przetwarzania i czasy próbkowania.

Dane z przetwornika są przesyłane za pomocą kanału 1 DMA bezpośrednio do pamięci o rozmiarze dwóch 16–bitowych komórek. Ten fragment pamięci to nic innego jak tablica, której zadaniem jest przechowywanie wyników pomiarów. Są one następnie przeliczane i wyświetlane na LCD.

Dodatkowego komentarza może wymagać odczyt napięcia z czujnika temperatury. Czujnik ten jest widziany z perspektywy mikrokontrolera jako kanał 16 (ADC12_IN16), zatem taki wybieramy do konwersji. Producent zaleca, aby czas próbkowania wynosił minimum 17 μs, więc ustalamy czas próbkowania na 239,5 cyklu, co przekłada się na czas 17,1 μs. Następnie trzeba włączyć czujnik temperatury, przełączając go z trybu czuwania do normalnej pracy. Należy to zrobić wywołując funkcję *ADC_TempSensorVrefintCmd()*. Dalej, po uruchomieniu przetwornika i zakończeniu konwersji, musimy przeliczyć wartość, która jest zawarta w rejestrze danych ADC, na wartość wyrażoną w stopniach Celsjusza. Równanie pozwalające obliczyć aktualną wartość temperatury jest następujące:

$$T(^oC) = \frac{V_{25} - V_{SENSE}}{Avg\_Slope} + 25$$

Poszczególne jego składniki to:

V_{25} – napięcie w temperaturze 25°C, może się zawierać w granicach 1,34 do 1,52 V, typowo wynosi 1,43 V;

V_{SENSE} – zmierzona wartość napięcia czujnika temperatury;

Avg_Slope – stała wartość, może przyjmować wartości z przedziału 4 do 4,6 $\frac{mV}{^oC}$, typowo wynosi 4,3 $\frac{mV}{^oC}$.

W przedstawionym przykładzie zostały przyjęte wartości typowe, czyli odpowiednio V_{25}=*1,43 V*, *Avg_Slope*=4,3 $\frac{mV}{^oC}$. Aby lepiej zrozumieć, w jaki sposób jest obliczana temperatura przeliczymy jeden przykład. Załóżmy, że wartość zmiennej ADCVal[1], a tym samym wynik pomiaru, wynosi 1785. Potrzebujemy tą wartość wyrażoną w woltach, a zatem skoro napięcie odniesienia wynosi 3,3 V, to rozdzielczość przetwarzania:

$$U_r = \frac{3{,}3V}{4095} \approx 805\mu V \approx 0{,}805mV$$

Stąd wartość z przetwornika wyrażona w mV będzie wynosić:

$$U_T = 1785 \cdot 0{,}805mV \approx 1436mV$$

Wyjaśnienia może wymagać jeszcze, dlaczego napięcia wyrażamy w miliwoltach, a nie woltach. Jeśli by wszystkie napięcia wyrazić w woltach, to trzeba by było mnożyć i dzielić przez bardzo małe liczby, co nie jest ani przejrzyste, ani efektywne. W tym równaniu można wszystkie napięcia wyrazić w miliwoltach, ponieważ i tak one się skracają. Wróćmy do meritum. Podstawiając otrzymane napięcia do równania otrzymujemy:

$$T(^oC) = \frac{1430mV - 1436mV}{4{,}3\frac{mV}{^oC}} + 25$$

Ostateczny wynik:

$T=23^oC$

Taka wartość zostanie wyświetlona na LCD.

Należy pamiętać, że pomiary temperatury są obarczone błędem ±1,5°C.

Wynik ten można osiągnąć po przeprowadzeniu kalibracji. Jeśli kalibracja nie zostanie wykonana, to błąd pomiaru może być większy. Kalibracja polega na odczycie wyniku pomiaru temperatury przy 25°C. Na jego podstawie można wprowadzić stosowne poprawki do równania podanego przez producenta.

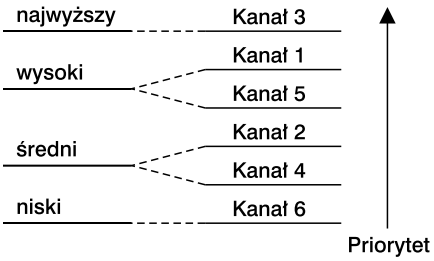
DMA

W poprzednim przykładzie został wykorzystany kontroler DMA, zatem warto nieco bliżej zapoznać się z jego budową i zasadą działania.

Wiele zadań we współczesnych systemach cyfrowych polega na przesyłaniu danych z jednego miejsca w pamięci do drugiego. Stąd zrodziło się pytanie: po co angażować do tego celu CPU? Jeżeli dane są tylko kopiowane lub przenoszone z miejsca na miejsce, to nie ma potrzeby wykorzystywania mocy obliczeniowej i rejestrów CPU. Zrodziła się wówczas idea budowy

List. 3. Funkcje konfigurujące DMA

```
void DMA_Conf(void)
{
        // Ustawienia domyslne DMA
        DMA_DeInit(DMA1_Channel1);

        // Adres rejestru danych ADC (Data Register)
        DMA_InitStruct.DMA_PeripheralBaseAddr = ADC1_DR_Address;

        // Adres pamieci, pod jaki beda zapisywane dane
        DMA_InitStruct.DMA_MemoryBaseAddr = (u32)&ADCVal;

        // Kierunek: zrodlem jest ADC
        DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;

        // Rozmiar burora: dwa kanaly = rozmiar bufora 2
        DMA_InitStruct.DMA_BufferSize = 2;

        // Wylaczenie licznika inkrementujacego adres dla peryferia
        // i wlaczenie go dla pamieci
        DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
        DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;

        // Dane 12 - bitowe, zatem wystarczy pol slowa
        DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_
HalfWord;
        DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;

        // Dane beda przesylane ciagle
        DMA_InitStruct.DMA_Mode = DMA_Mode_Circular;

        // Priorytet wysoki
        DMA_InitStruct.DMA_Priority = DMA_Priority_High;

        // Wylaczenie przesylania z pamieci do pamieci
        DMA_InitStruct.DMA_M2M = DMA_M2M_Disable;

        DMA_Init(DMA1_Channel1, &DMA_InitStruct);
        // Wlacz DMA
        DMA_Cmd(DMA1_Channel1, ENABLE);
}
```



Rys. 5. Priorytety obsługi kanałów DMA

układu służącego do transmisji bloków danych w pamięci. Układy z tej grupy noszą nazwę kontrolerów DMA (*Direct Memory Access*). Zajmują się całą pracą związaną z kopiowaniem i przenoszeniem dużych bloków danych, zwalniając tym samym z tego obowiązku CPU.

Mikrokontroler STM32F103VB jest wyposażony w 7-kanałowy kontroler DMA o ustawianej na 8, 16 lub 32 bity długości słowa danych. Każdemu z kanałów można przypisać określony priorytet. Możliwa jest transmisja pomiędzy dwoma układami peryferyjnymi, z układu peryferyjnego do pamięci, z pamięci do układu peryferyjnego oraz z pamięci do pamięci. Dane można pojedynczo lub w postaci całych bloków danych, przy czym maksymalny rozmiar takiego bloku może wynosić 65535.

System priorytetów obsługi kanałów DMA.

Kontroler DMA, wbudowany w mikrokontrolery STM32 rozróżnia cztery programowo ustalane priorytety:

- najwyższy (*very high priority*),
- wysoki (*high priority*),
- średni (*medium priority*),
- niski (*low priority*).

Każdemu z dostępnych kanałów można przyporządkować któryś z wyżej wymienionych priorytetów. Powstaje jednak pytanie, co się dzieje w chwili, gdy pojawiają się dwa żądania dostępu do kontrolera DMA z dwóch kanałów o takim samym programowym priorytecie? W takim przypadku pierwszeństwo ma kanał o mniejszym numerze, wyjaśnia to **rys. 5**. Przykładowo priorytet wysoki mają kanały 1 i 5, ale gdy obydwa zażądają dostępu do DMA w tym samym czasie, to w pierwszej kolejności zostanie obsłużony kanał 1.

Wykorzystany w poprzedniej aplikacji (na list. 2) kontroler DMA został skonfigurowany do przesyłu danych z przetwornika ADC do pamięci (czyli w efekcie do zmiennej). Funkcję konfigurującą DMA przedstawiono na **list. 3**. Ponadto, aby kontroler DMA pracował poprawnie, w pierwszej kolejności należy włączyć jego sygnał zegarowy, umieszczając w funkcji *RCC_Conf()* linijkę:

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_
DMA1, ENABLE);
```

Podsumowanie

Przedstawione w artykule przykłady ukazują tylko niewielką część możliwości, jakie oferują konstruktorom przetworniki A/C wbudowane w mikrokontrolery STM32. Ogromna różnorodność trybów pracy oraz elastyczność konfiguracji sprawiają, że te peryferia mogą być wykorzystywane w aplikacjach, w których do niedawna niezbędne było używanie zewnętrznych przetworników. Gdy do współpracy z ADC zostanie wykorzystany kontroler DMA, to w konsekwencji programista otrzymuje system zdolny przetwarzać spore ilości informacji, pozostawiając jeszcze dużo wolnych zasobów CPU. Wolną moc obliczeniową mikrokontrolera można w takim przypadku wykorzystać do realizacji innych zadań.

Krzysztof Paprocki