Kurs programowania mikrokontrolerów XMEGA (9)

Przetwornik C/A

Przetwornik cyfrowo-analogowy to kolejna nowość w XMEGA w porównaniu do starszych rodzin AVR, w których był on dostępny jedynie w mikrokontrolerach przeznaczonych do zastosowań specjalistycznych, jak np. AT90PWM3B. W XMEGA przetwornik C/A jest dostępny w zdecydowanej większości mikrokontrolerów – jedynie najtańsze i najuboższe nie mają tego układu.

Przetwornik C/A przetwarza wartość cyfrową na napięcie, którego możemy użyć do sterowania układów analogowych, np. jasnością świecenia diody LED. Można zbudować generator sygnałów analogowych, co zademonstruję dalej lub można pójść o krok naprzód i wykonać odtwarzacz plików audio, co z użyciem przetwornika C/A i DMA nie jest bardzo skomplikowane.

Przetwornik C/A w XMEGA ma rozdzielczość 12 bitów, co oznacza, że jest możliwe uzyskanie 4096 poziomów napięcia pomiędzy poziomem odniesienia zasilania, a napięciem referencyjnym przetwornika, które wyznacza maksymalne napięcie, które przetwornik może wygenerować.

Mamy do wyboru cztery napięcia odniesienia:

- AVCC – napięcie zasilające podłączone do pinu zasilającego procesor. Ze względu na liczne szumy na liniach zasilających, nie zaleca się stosowania tego rozwiązania w aplikacjach wymagających dużej precyzji.
- Wewnętrzne o wartości 1 V – jest to najlepsze wbudowane źródło odniesienia o wystarczającej stabilności do większości zastosowań.
- AREFA i AREFB – są to odpowiednio piny A0 i B0, do których możemy doprowadzić zewnętrzne źródło napięcia odniesienia.

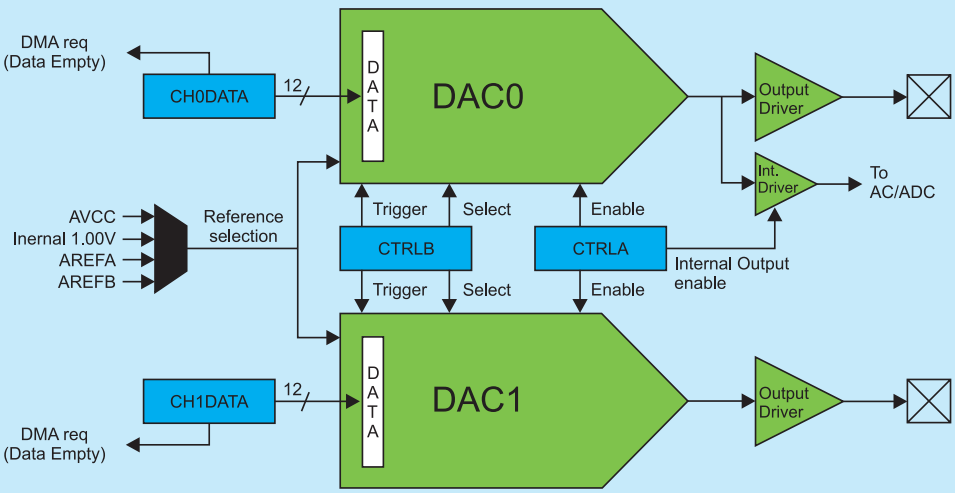
Napięcie referencyjne wyznacza napięcie maksymalne, które można uzyskać na wyjściu przetwornika. Jest ono liniowo proporcjonalne do liczby wpisanej do rejestru CHxDATA, a formalnie napięcie wyjściowe opisuje równanie:

$$V_{OUT} = V_{REF} \cdot \frac{CHxDATA}{4095}$$

Przetwornik C/A ma dwa niezależne kanały o numerach 0 i 1, a każdy ma przepustowość rzędu miliona próbek na sekundę. Wyjścia kanałów w ATxmega128A3U są niestety na sztywno połączone z pinami B2 i B3 (oznaczonymi na płytce eXtrino XL etykietami DAC0 i DAC1). Kanał 0 można wewnętrznie połączyć z komparatorem analogowym (omówionym w EP 2014/08) oraz przetwornikiem analogowo-cyfrowym.

Budowę przetwornika C/A przedstawiono na **rysunku 1**.

Konwersja rozpoczyna się w dwóch przypadkach. Pierwszy polega na „ręcznym" wpisaniu wartości do rejestru CHxDATA, po czym przetwornik samoczynnie generuje właściwe napięcie na swoim wyjściu. Przed wpisaniem kolejnej wartości należy sprawdzić, czy przetwornik zakończył już przetwarzanie, by nie zakłócić jego pracy. Druga metoda zdecydowanie automatyzuje proces przetwarzania i dobra jest, kiedy chcemy uzyskać generator jakiegoś sygnału. Sztuczka polega na tym, że po zakończeniu konwersji, przetwornik zgłasza do układu DMA żądanie przesłania kolejnej próbki. Konwersja na-



Rysunek 1. Budowa przetwornika cyfrowo-analogowego

stępuje po podaniu sygnału wyzwalającego przez system zdarzeń, pochodzącego np. od timera. Dzięki temu można łatwo i efektywnie wysyłać kolejne próbki w równych odstępach czasu. W dalszej części kursu przećwiczymy obie metody obsługi przetwornika.

Podstawy budowy i obsługi przetwornika C/A w XMEGA

W pierwszym przykładzie poznamy elementarne podstawy inicjalizacji przetwornika cyfrowo-analogowego. Inicjalizacja jest bardzo łatwa i sprawdza się do skonfigurowania zaledwie dwóch rejestrów.

W rejestrze **CTRLA** znajduje się pięć bitów konfiguracyjnych:

- **DAC_ENABLE_bm** – włączenie przetwornika.
- **DAC_CH0EN_bm** – wyjście kanału CH0 ma być dostępne na pinie B2, zgodnie z tym, co napisano w dokumentacji.
- **DAC_CH1EN_bm** – wyjście kanału CH1 ma być dostępne na wyprowadzeniu B3.
- **DAC_IDOEN_bm** – wyjście kanału CH0 ma być dostępne dla komparatora analogowego lub przetwornika analogowo-cyfrowego. Warto pamiętać, że kiedy DAC jest połączony z jakimś układem wewnętrznym, wówczas nie można wyprowadzić jego wyjścia na pin mikrokontrolera, bez znaczenia czy CH0EN jest ustawiony czy nie.
- **DAC_LPMODE_bm** – uruchomienie trybu oszczędzania energii, przez co przetwornik będzie działał z dwukrotnie mniejszą prędkością.

Rejestr *CTRLB* omówimy w drugim przykładzie, a w rejestrze *CTRLC* musimy wybrać napięcie odniesienia. Wpisujemy do niego następujące bity:

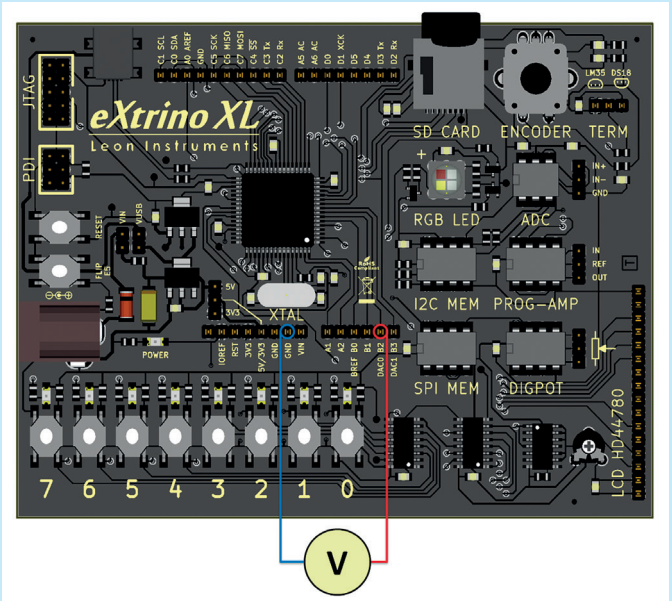
- **DAC_REFSEL_x_gc** – jest to grupa konfiguracyjna, odpowiedzialna za wybór napięcia odniesienia. Za x należy wpisać **INT1V**, **AVCC**, **AREFA**, **AREFB**.

Po wgraniu do procesora programu z **listingu 1**, należy podłączyć woltomierz pomiędzy pin B2 (oznaczony jako DAC0) oraz masę (GND), co zostało przedstawione na **rysunku 2**. Napięcie możemy zwiększać przyciskiem 0 oraz zmniejszać przyciskiem 1.

Czy przetwornik cyfrowo-analogowy jest lepszy od PWM? Jak to zwykle – zależy od sytuacji. Przede wszystkim, sygnał jest wolny od szumów, które powstają w wyniku naprzemiennego załączania i wyłączania wyjścia. Nie musimy zatem przykładać dużej uwagi do filtracji otrzymanego sygnału, a 12-bitowa rozdzielczość jest wystarczająca do większości zastosowań. Przetwornik DAC jest także szybszy od PWM – możemy zatem uzyskać sygnał o większej częstotliwości. Jednak DAC nie nadaje się do sterowania układów mocy, takich jak silniki czy grzałki z uwagi na to, że sterowanie ich przez PWM generuje mniejsze straty mocy niż w przypadku sterowania liniowego.

Generator DDS

DDS oznacza Direct Digital Synthesis, czyli bezpośrednią syntezę cyfrową. Jest to rozwiązanie, które odesłało do lamusa tradycyjne analogowe generatory funkcyjne. Generator DDS do pracy wykorzystuje przetwornik cyfrowo-analogowy, więc mikrokontrolery XMEGA nadadzą się znakomicie do tego celu. Generator potrzebuje tablicy z próbkami sygnału, który ma zostać wytworzony. Co ściśle określony czas, z tablicy pobierana jest próbka i przekazywana do przetwornika. Bardzo istotną zaletą generatorów DDS nad analogowymi jest to, że zawartość tablicy może być absolutnie dowolna i dzięki temu możemy wygenerować sygnał o dowolnym kształcie. Ze względu na duże możliwości, dobre parametry i umiarkowany stopień skomplikowania, generatory DDS znajdują bardzo szerokie zastosowanie. Są dostępne nawet generatory zintegrowane w jednym układzie scalonym, jednak niestety są one dość drogie.

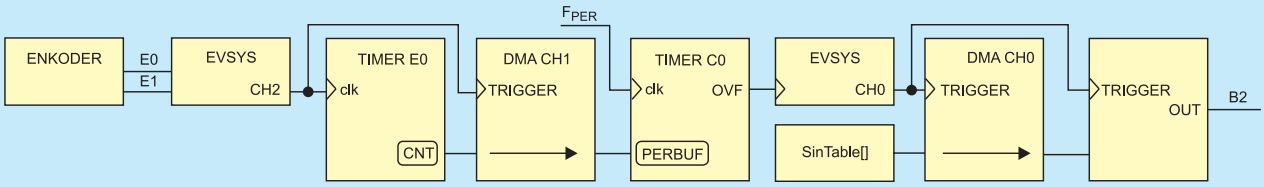


Rysunek 2. Podłączenie woltomierza do płytki testowej

Oczywiste jest, by do odmierzania czasu wykorzystać timer, poznany w EP 2014/02. Kolejnym dobrym pomysłem będzie zastosowanie układu DMA, opisanego w EP 2014/08. Wykorzystamy również system zdarzeń, który będzie synchronizował pracę DMA i DAC.

Nie będziemy się ograniczać jedynie do trywialnego przesyłania próbek z pamięci do DAC. W takiej sytuacji nie mielibyśmy żadnego wpływu na generowany sygnał, a przecież podstawową funkcją generatora jest możliwość uzyskania sygnału o żądanej częstotliwości i amplitudzie. Do zmiany amplitudy wykorzystamy potencjometr cyfrowy, w jaki wyposażona jest płytka eXtrino XL, ale będzie to w jednym z przyszłych odcinków. Dobrze by było, by regulacja częstotliwości możliwa była przy pomocy pokrętła, na wzór tradycyjnych generatorów. Przypomnimy sobie obsługę dekodera kwadraturowego, który umożliwia bardzo proste podłączenie enkodera obrotowego. Zauważ, że nasze układy, omawiane w kolejnych odcinkach kursu, stają się coraz bardziej skomplikowane, a mimo to realizujemy je całkowicie sprzętowo!

Schemat połączeń układów peryferyjnych wewnątrz mikrokontrolera przedstawiono na **rysunku 3**. Zacznę go omawiać od końca, czyli od przetwornika cyfrowo-analogowego. Działa on inaczej niż w pierwszej części tego odcinka. Próbki ładowane są do rejestru **CH0DATA**, jednak sam fakt wpisania nowej wartości do rejestru nie wywołuje przetwarzania. Do rejestru **CTRLB** wpisano bity **DAC_CH0TRIG_bm** oraz **DAC_CHSEL_SINGLE_gc**, co powoduje, że konwersję na kanale 0 przetwornika musi wyzwolić któryś kanał systemu zdarzeń. Który to kanał – musimy to określić, wpisując **DAC_EVSEL_0_gc** do rejestru **EVCTRL**.



Rysunek 3. Schemat generator DDS w mikrokontrolerze ATxmega128A3U

Musimy zastanowić się, z ilu próbek ma składać się tablica danych, co pociąga za sobą tryb pracy przetwornika – może on być 12-bitowy lub 8-bitowy. Tryb 12-bitowy oznacza, że każda próbka zajmuje 2 bajty, choć de facto użyteczne jest tylko półtora bajtu, a pozostałe pół jest zmarnowane. Większa ilość próbek pozwala bardziej wiarygodnie odwzorować sygnał, taki jak sinus. Często przyjmuje się, że liczba próbek powinna być równa liczbie kombinacji napięć możliwych do uzyskania, wynikającej z rozdzielczości przetwornika (chociaż nie jest to jakaś żelazna reguła, raczej jest to tylko wskazówka projektowa). Zatem w przypadku przetwornika 12-bitowego, mamy 4096 możliwych poziomów napięć, które może przetwornik DAC może uzyskać na wyjściu. Skoro każda z tych próbek zajmuje 2 bajty, to cała tablica musiałaby zajmować 8192 bajty. Jest tylko jeden problem – w ten sposób byśmy zajęli całą pamięć mikrokontrolera ATxmega128A3U. Należałoby w takiej sytuacji zastosować zewnętrzny RAM, wykorzystując interfejs równoległy EBI. Jeśli nie zależy nam na bardzo dużej dokładności, warto rozważyć tryb 8-bitowy. W takiej sytuacji będziemy mieli 256 próbek o rozmiarze 1 bajta, czyli 256B, a więc potrzebujemy 32 razy mniej miejsca w pamięci! Taką tablicę bez problemu zmieścimy w RAM. Tablicę próbek można sobie wygenerować w Excelu. W plikach załączonych do kursu znajduje się przykładowy arkusz, który wykorzystałem, by stworzyć program generatora.

Podczas wybierania trybu 8- lub 12-bitowego, musimy uświadomić dobie, że rejestr *CHxDATA*, do którego wcześniej bez zastanowienia wpisywaliśmy liczby od 0 do 4095 to dwa rejestry 8-bitowe, o nazwach *CHxDATAH* oraz *CHxDATAL*. Używając nazwy *CHxDATA* bez H i L na końcu, przerzucamy na kompilator obowiązek rozdzielenia próbki na dwa bajty i zapisania ich w odpowiedniej kolejności. Jako że przetwornik jest 12-bitowy, a w tych rejestrach mamy 16 bitów, oznacza to, że 4 bity pozostaną niewykorzystane. Które z nich – określić to możemy wyrównując rejestr do prawej lub do lewej.

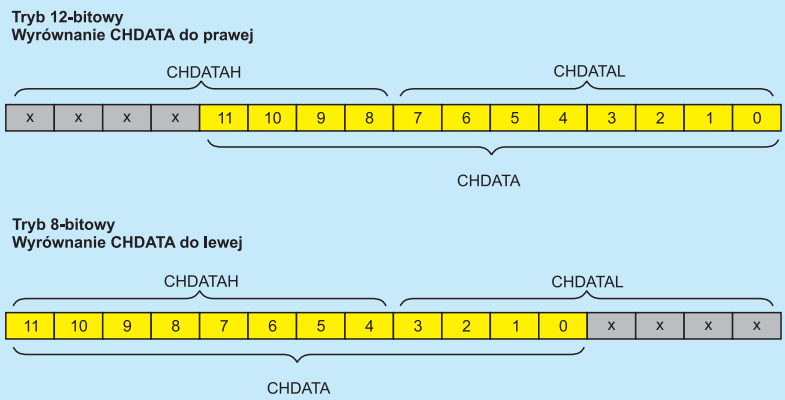
Spójrz na **rysunek 4**. Domyślnie mamy rejestr *CHDATA* wyrównany do prawej, przez co zupełnie naturalnie możemy do niego wpisywać liczby od 0 do 4095, a kompilator sam zadba o to, by wpisać właściwe wartości do połówek H i L. Liczby większe od 4095 wymagają użycia 12 bitów lub więcej. W przypadku wyrównania do lewej, wpisujemy tylko 8 bitów do rejestru H, a rejestr L pozostawiamy wyzerowany. Tak naprawdę, przetwornik cały czas pracuje z 12-bitową rozdzielczością, ale przesunięcie rejestru do lewej strony pozwala nam zapisać 8 najbardziej znaczących bitów w jednym takcie zegara. W ten sposób oszczędzamy czas, zachowując elegancką prostotę. Aby wyrównać rejestr *CHDATA* do lewej, należy wpisać bit *DAC_LEFTADJ_bm* do rejestru *CTRLC*. Trzeba pamiętać, że to ustawienie dotyczy obu kanałów przetwornika!

Próbki do przetwornika ładowane są z tablicy *SinWave[]* zdefiniowanej w pliku sin.h. Kopiowaniem danych z tablicy do DAC zajmuje się układ DMA, który poznaliśmy w EP 2014/09 i konfiguracja DMA zastosowana w niniejszym ćwiczeniu nie różni się od tej, opisanej w poprzednim odcinku kursu.

Koniecznie jednak należy zwrócić uwagę, że transfer DMA i konwersja C/A jest wywoływana tym samym kanałem systemu zdarzeń – kanałem 0. Czy to oznacza, że po wystąpieniu wyzwalacza w systemie zdarzeń próbka z tablicy natychmiast pojawia się na wyjściu przetwornika? Nie! W chwili wystąpienia sygnału wyzwalającego, następuje rozpoczęcie przetwarzania próbki, która w tej chwili znajduje się w rejestrze *CH0DATA*. Jest to próbka, która została skopiowana przy poprzednim wyzwalaczu, a teraz kopiowana jest próbka, która zostanie przetworzona przez DAC przy kolejnym sygnale. Mamy tu przykład klasycznego przetwarzania potokowego (ang. *pipelining*) – w czasie, gdy jeden układ przetwarza dane, drugi układ równolegle ładuje do pierwszego kolejną porcję informacji.

Źródłem sygnałów wyzwalających dla DMA i DAC jest timer C0. Timery w XMEGA poznaliśmy w EP 2014/02. Krótkie przypomnienie – z każdym taktem sygnału zegarowego CLK_{PER} (równego częstotliwości taktowania procesora) jest zwiększany rejestr licznika *CNT*. Gdy wartość rejestru *CNT* zrówna się z rejestrem *PER*, następuje wyzerowanie licznika i wygenerowanie sygnału przepełnienia, który może wywoływać przerwanie lub zdarzenie. W tym przypadku mamy do czynienia ze zdarzeniem, transmitowanym przez kanał 0 do DMA i przetwornika C/A. Zatem częstotliwość pobierania kolejnych próbek (a i również częstotliwość sygnału generowanego na wyjściu przetwornika C/A) określa rejestr *PER* – im jest on większy, tym zdarzenie będą generowane rzadziej, a w rezultacie ta częstotliwość będzie mniejsza.

Parę akapitów wcześniej pisałem, że częstotliwość sygnału będziemy regulować enkoderem obrotowym



Rysunek 4. Rejestr CHDATA wyrównany do prawej lub do lewej

Listing 1. Kod programu do pierwszego ćwiczenia

```
#include <avr/io.h>
#include „extrino_portx.h”
#include „hd44780.h”

int main(void) {
    PortxInit();
    LcdInit();

    // inicjalizacja DAC
    DACB.CTRLA = DAC_CH0EN_bm|  //uaktywnienie kanału 0
    DAC_ENABLE_bm;  //włączenie DAC
    DACB.CTRLC = DAC_REFSEL_INT1V_gc;  // źródło napięcia odniesienia 1V
    DACB.CH0DATA = 1000;  // wartość początkowa
    while(1) {
    // wyświetlenie rejestru CH0DATA
        Lcd1;
        Lcd(„CH0DATA = „);
        LcdDec(DACB.CH0DATA);
        Lcd(„     „);
        // wyświetlenie napięcia na wyjściu
        Lcd2;
        Lcd(„Vout = „);
        LcdDec((uint32_t)DACB.CH0DATA * 1000 / 4096);  // przeliczenie na miliwolty
        Lcd(„mV    „);
        if((PORTX.IN & PIN0_bm) && (DACB.CH0DATA < 4095)) {  // czy przycisk 0
            while((DACB.STATUS & DAC_CH0DRE_bm) == 0);       // czekaj na gotowość
            DACB.CH0DATA++;                                  // zwiększanie napięcia
        }
        if((PORTX.IN & PIN1_bm) && (DACB.CH0DATA > 0)) {  //czy przycisk 1
            while((DACB.STATUS & DAC_CH0DRE_bm) == 0);  //czekaj na gotowość
            DACB.CH0DATA--;  // zmniejszanie napięcia
        }
    }
}
```

(opisanym w EP 2014/03). Enkoder na płytce eXtrino XL podłączony jest do pinów E0 i E1. Dekoder kwadraturowy, odpowiedzialny za sprzętową obsługę enkodera, zlokalizowany jest w systemie zdarzeń, ale (uwaga!) jest on dostępny tylko w kanałach 0, 2 i 4 – kanał 0 jest już zajęty, więc wybrałem kanał 2. Aby aktywować dekoder, musimy w rejestrze *EVSYS.CH2MUX* wskazać pierwszy pin procesora, do którego jest dołączony enkoder (a mianowicie *EVSYS_CHMUX_PORTE_PIN0_gc*), a następnie w rejestrze *EVSYS.CH2CTRL* włączamy enkoder kwadraturowy (*EVSYS_QDEN_bm*) oraz filtr cyfrowy na 8 taktów zegara (*EVSYS_DIGFILT_8SAMPLES_gc*).

Dekoder w systemie zdarzeń musi przekazywać impulsy do timera, który w tym przypadku pracuje jako licznik. W tym przykładzie będzie to timer E0. Obracając pokrętło enkodera, rejestr *CNT* licznika E0 będzie się zwiększał lub zmniejszał, w zależności od kierunku obrotu.

Ale jak ożenić timer C0 i E0? Jak powiązać obrót enkodera ze zmianą rejestru *PER*, która reguluje częstotliwość pobierania próbek sygnału? Najprościej będzie wykorzystać do tego… kolejny kanał DMA! Mikrokontrolery XMEGA mają nawet 4 kanały DMA, więc wykorzystajmy kolejny z nich.

Uwaga! Łatwo tu wpaść w pewną pułapkę! Zastanówmy się – rejestr *TCC0.CNT* ma wartość, dla przykładu, równą 99, a rejestr *TCC0.PER* niech ma wartość100. Oznacza to, że w następnym takcie zegara rejestry te zrównają się, *CNT* zostanie wyzerowane, a system zdarzeń przekaże sygnał do DMA i DAC. Co by było w sytuacji, kiedy właśnie wtedy przekręcamy enkoder, przez co rejestr *PER* zostaje zmniejszony o 4 (wynika to z budowy enkodera – jedno kliknięcie pokrętła to 4 impulsy) do wartości 96? Wtedy rejestr *CNT* nie zrówna się z *PER*, przez co *CNT* będzie zwiększał się do wartości maksymalnej (65535), dopiero wtedy zacznie kolejny cykl. W tym czasie przetwornik DAC nie będzie dostawał żadnych nowych próbek, co będzie objawiało się długą poziomą linią na oscyloskopie, zupełnie tak jakby przetwornik zawiesił się na chwilę.

Rozwiązaniem problemu jest wpisywanie nowych wartości do rejestrów buforowanych *CNTBUF*, *CCxBUF* oraz *PERBUF*. Zapisanie do nich nowej wartości spowoduje uaktualnienie odpowiadających rejestrów *CNT*, *CCx* oraz *PER* dopiero w chwili zakończenia cyklu pracy timera. Dzięki temu unikniemy sytuacji opisanej w poprzednim akapicie.

Musimy zdefiniować w rejestrze źródłowym *DMA.CH1.SRCADDRx* adres rejestru *TCE0.CNT*, a do rejestru docelowego *DMA.CH1.DESTDDRx* wpisujemy adres rejestru *TCC0.PERBUF*. Zwracam uwagę na słowo *adres* – nie wypisujemy wartości tych rejestrów, ale ich adresy i dlatego posługujemy się operatorem &. Jako że mamy do czynienia z rejestrami 16-bitowymi, całą transakcję podzielimy na bloki, a każdy z nich będzie składał się z pojedynczego transferu burst o rozmiarze 2 bajtów.

Wyzwalaczem układu DMA będzie kanał 2 systemu zdarzeń – ten sam, który powoduje zwiększanie lub zmniejszania licznika E0. Jest to dobre rozwiązanie – w ten sposób DMA będzie transferować dane tylko wtedy, kiedy nastąpi ich zmiana na skutek obrotu pokrętła enkodera.

Kiedy mamy włączone dwa kanały DMA, koniecznie musimy rozważyć, jak rozwiązać sprawę priorytetów, który z nich ma mieć pierwszeństwo w dostępnie do magistrali. Zastanówmy się:

- przesyłanie próbek do przetwornika C/A ze ściśle określoną częstotliwością
- zmiana częstotliwości na skutek obrotu enkodera

Wybór jest oczywisty – w przypadku jednoczesnego zgłoszenia dostępu do magistrali, pierwszeństwo dostanie kanał 0, który jest odpowiedzialny za przesyłanie próbek. Kopiowanie rejestrów liczników nie jest takie ważne i nawet, jeśli kanał będzie musiał poczekać na dostęp, to my i tak tego nie zauważymy. W przypadku opóźnienia przesłania próbki mogłaby nam się pojawić niepotrzebna szpilka (*glitch*) na wyjściu przetwornika lub nieznaczna zmiana okresu generowanego sygnału. Dlatego w rejestrze *DMA.CTRL*, gdzie są wspólne ustawienia dla wszystkich kanałów, ustawiłem grupę *DMA_PRIMODE_RR0123_gc*. Powoduje to, że pierwszeństwo w dostępnie będą miało kanały o niższych numerach.

To tyle, jeśli chodzi o konfigurację rejestrów i peryferiów. Dochodzimy do pętli głównej while(1) i... co właściwie nasz program powinien robić? Wszystko się robi sprzętowo i rdzeń procesora można by w ogóle wyłączyć! To może niech na wyświetlaczu LCD pokazuje się aktualna częstotliwość sygnału wyjściowego. Tego (jeszcze) całkowicie sprzętowo zrobić się nie da.

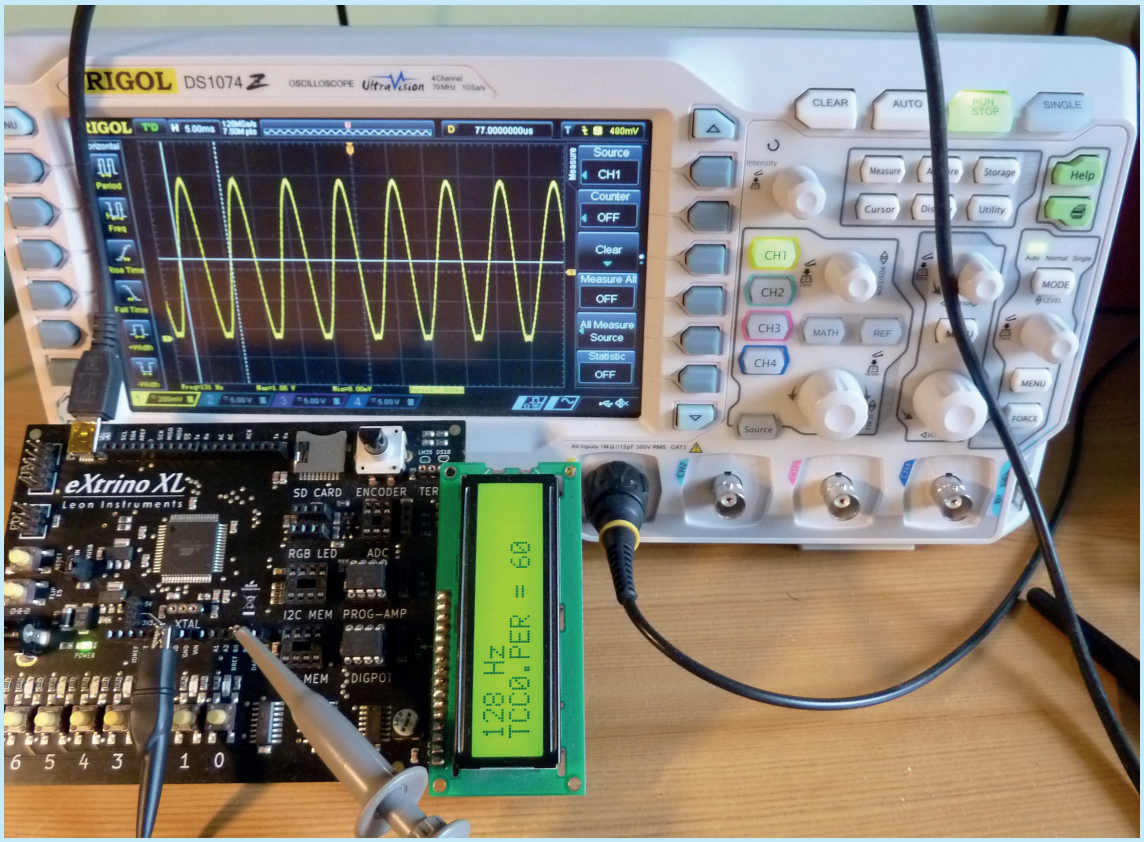
Program przedstawiono na **listingu 2**. Celowo kolejność konfiguracji peryferiów jest taka sama jak w powyższych wyjaśnieniach.

Po wgraniu programu do procesora, podłączamy sondę oscyloskopu do pinu B2, a krokodylek masy podpinamy do najbliższego punku GND. Na oscyloskopie powinna ukazać się sinusoida o amplitudzie 1 V, a obracając pokrętło enkodera powinniśmy obserwować zmianę

Listing 2. Kod programu generatora DDS

```
#include <avr/io.h>
#include „hd44780.h”
#include „sin.h”

int main(void) {
    // inicjalizacja wyświetlacza
    LcdInit();
    // inicjalizacja DAC
    DACB.CTRLA = DAC_CH0EN_bm|  // uaktywnienie kanału 0
                 DAC_ENABLE_bm;  // włączenie DAC
    DACB.CTRLB = DAC_CHSEL_SINGLE_gc|  // tylko kanał 0
                 DAC_CH0TRIG_bm;  // konwersję wywołuje system zdarzeń
    DACB.CTRLC = DAC_REFSEL_INT1V_gc|  // źródło napięcia odniesienia 1V
                 DAC_LEFTADJ_bm;  // wyrównanie rej CH0DATA do lewej (tryb 8-bitowy)
    DACB.EVCTRL = DAC_EVSEL_0_gc;  // konwersję inicjuje CH0 systemu zdarzeń
    DACB.CH0DATA = 0;  // wartość początkowa
    // konfiguracja kontrolera DMA
    DMA.CTRL = DMA_ENABLE_bm|  // włączenie kontrolera
               DMA_DBUFMODE_DISABLED_gc|  // bez podwójnego buforowania
               DMA_PRIMODE_RR0123_gc;  // priorytet od najniższych numerów
    // konfiguracja kanału DMA
    DMA.CH0.SRCADDR0 = (uint16_t)SinWave & 0xFF;  // adres źródła
    DMA.CH0.SRCADDR1 = (uint16_t)SinWave >> 8;
    DMA.CH0.SRCADDR2 = 0;
    DMA.CH0.DESTADDR0 = (uint16_t)&DACB.CH0DATAH & 0xFF; // adres celu
    DMA.CH0.DESTADDR1 = (uint16_t)&DACB.CH0DATAH >> 8;
    DMA.CH0.DESTADDR2 = 0;
    DMA.CH0.TRFCNT = sizeof(SinWave);  // rozmiar bloku = rozmiar tablicy SinWave
    DMA.CH0.REPCNT = 0;  // ile bloków, 0 oznacza wysyłanie w nieskończoność
    DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_DACB_CH0_gc;  // DAC powoduje transfer
    DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_BLOCK_gc|  // przeładowanie adresu źródła po zakończeniu bloku
                       DMA_CH_SRCDIR_INC_gc|  // zwiększanie adresu źródła po każdym bajcie
                       DMA_CH_DESTRELOAD_NONE_gc|  // przeładowanie adresu celu nigdy
                        DMA_CH_DESTDIR_FIXED_gc;  // stały adres docelowy
    DMA.CH0.CTRLA = DMA_CH_ENABLE_bm|  // włączenie kanału
                    DMA_CH_BURSTLEN_1BYTE_gc|  // burst = 1 bajt
                    DMA_CH_SINGLE_bm|  // pojedynczy burst po każdym zdarzeniu
                    DMA_CH_REPEAT_bm;  // powtarzanie
    // konfiguracja timera regulujące częstotliwość przesyłania kolejnych próbek
    TCC0.CTRLB = TC_WGMODE_NORMAL_gc;  // tryb normalny
    TCC0.CTRLA = TC_CLKSEL_DIV1_gc;  // ustawienie preskalera i uruchomienie
    TCC0.PER = 100;  // okres timera
    // konfiguracja systemu zdarzeń - połączenie TCC0 z DMA i DAC
    EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;  // zdarzenie na CH0 wywołuje przepełnienie TC0
    // enkoder obrotowy
    PORTCFG.MPCMASK = PIN1_bm | PIN0_bm;  // wybór pinów 0 i 1 do konfiguracji
    PORTE.PIN0CTRL = PORT_ISC_LEVEL_gc |  // reagowanie na poziom niski
    PORT_OPC_PULLUP_gc;  // podciągnięcie do zasilania
    EVSYS.CH2MUX = EVSYS_CHMUX_PORTE_PIN0_gc;  // pin E0 wywołuje zdarzenie
    EVSYS.CH2CTRL = EVSYS_QDEN_bm|  // włączenie dekodera w systemie zdarzeń
    EVSYS_DIGFILT_8SAMPLES_gc;  // filtr cyfrowy
    // konfiguracja timera obsługującego enkoder
    TCE0.CNT = 100;  // wartość początkowa
    TCE0.CTRLA = TC_CLKSEL_EVCH2_gc;  // taktowanie systemem zdarzeń
    TCE0.CTRLD = TC_EVACT_QDEC_gc |  // włączenie dekodera kwadraturowego
                 TC_EVSEL_CH2_gc;  // dekoder zlicza impulsy z kanału 0
    // konfiguracja kanału DMA - przesyłanie TCE0.CNT do TCC0.PER
    DMA.CH1.SRCADDR0 = (uint16_t)&TCE0.CNT & 0xFF;  // adres źródła
    DMA.CH1.SRCADDR1 = (uint16_t)&TCE0.CNT >> 8;
    DMA.CH1.SRCADDR2 = 0;
    DMA.CH1.DESTADDR0 = (uint16_t)&TCC0.PERBUF & 0xFF;   // adres celu
    DMA.CH1.DESTADDR1 = (uint16_t)&TCC0.PERBUF >> 8;
    DMA.CH1.DESTADDR2 = 0;
    DMA.CH1.TRFCNT = sizeof(uint16_t);  // rozmiar bloku = 2 bajty
    DMA.CH1.REPCNT = 0;  // ile bloków, 0 oznacza wysyłanie w nieskończoność
    DMA.CH1.TRIGSRC = DMA_CH_TRIGSRC_EVSYS_CH2_gc;  // DAC powoduje transfer
    DMA.CH1.ADDRCTRL = DMA_CH_SRCRELOAD_BLOCK_gc|  // przeładowanie adresu źródła po zakończeniu bloku
                       DMA_CH_SRCDIR_INC_gc|  // zwiększanie adresu źródła po każdym bajcie
                       DMA_CH_DESTRELOAD_BLOCK_gc|  // przeładowanie adresu celu nigdy
                       DMA_CH_DESTDIR_INC_gc;  // zwiększanie adresu celu po każdym bajcie
    DMA.CH1.CTRLA = DMA_CH_ENABLE_bm|  // włączenie kanału
                    DMA_CH_BURSTLEN_2BYTE_gc|  // burst = 2 bajty
                    DMA_CH_REPEAT_bm;  // powtarzanie w nieskończoność
    while(1) {
        // wyświetlenie częstotliwości sygnału sinus
        Lcd1;
        LcdDec(2000000UL / (256*(TCC0.PER+1)));
        Lcd(„ Hz    „);
        // wyświetlenie zawartości rejestru TCC0.PER
        Lcd2;
        Lcd(„TCC0.PER = „);
        LcdDec(TCC0.PER);
        Lcd(„     „);
    }
}
```



Fotografia 5. Płytka eXtrino XL podłączona do oscyloskopu

częstotliwości sinusoidy. Na **fotografii 5** pokazano działający układ podłączony do oscyloskopu, a **rysunek 6** przedstawia przykładowy oscylogram zapisany podczas kręcenia pokrętłem enkodera.

Jednak coś jest nie tak! Zauważyć można, że dolne fragmenty sinusoidy są trochę zdeformowane. Czy to świadczy o jakiejś ukrytej wadzie mikrokontrolerów XMEGA? Nie! **Przestrzegam przed „ekspertami internetowymi", którzy na różnych forach wypisują bzdury o wadach przetworników w XMEGA.** Rozwiązanie problemu jest banale – przetwornik trzeba po prostu SKALIBROWAĆ. Można to zrobić na dwa sposoby:

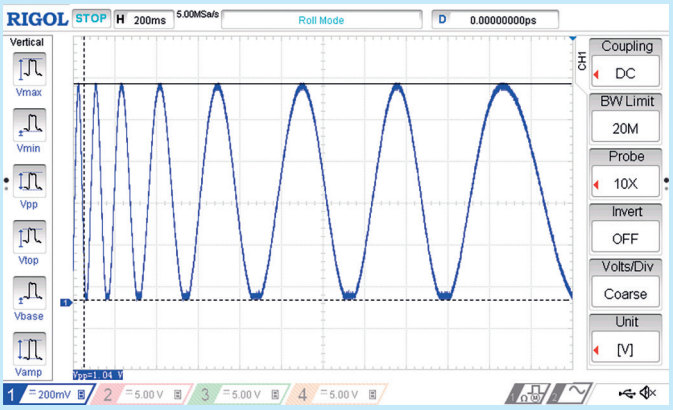
- W sygnaturze procesora są współczynniki kalibracyjne, zapisane podczas produkcji procesora. Trzeba je odczytać kontrolerem pamięci NVM i wpisać do rejestrów kalibracji przetwornika.
- Przetwornik analogowo-cyfrowy może pracować jako sprzężenie zwrotne dla cyfrowo-analogowego. Aby skalibrować C/A za pomocą A/C, należy wystawić na wyjście wartość najmniejszą oraz największą i odczytać z A/C jakie napięcie wygenerował C/A. Kolejnym krokiem jest określenie współczynników kalibracyjnych.

Są dwa współczynniki, które możemy modyfikować, a mianowicie – offset i wzmocnienie. Dzięki temu możemy w dość znacznym stopniu modyfikować charakterystykę przetwornika. Jednak w kursie nie omawialiśmy jeszcze A/C ani NVM, więc niestety temat kalibracji musimy odłożyć na późniejszy termin. Możesz zasięgnąć informacji np. z książek Tomasza Francuza – procedura kalibracji jest tam bardzo dobrze opisana.

Co dalej?

Zrobiliśmy bardzo nieskomplikowany generator, ale wykorzystuje on zaledwie ułamek możliwości mikrokontrolera ATxmega128A3U. Dobrym pomysłem byłoby dodanie innych sygnałów oprócz sinusa. Aby to zrobić, wystarczy wygenerować inną tablicę próbek (np. w Excelu). Przydałaby się także regulacja amplitudy za pomocą potencjometru cyfrowego zawartego w płytce eXtrino XL. Ostatnim, bardzo ważnym elementem, o którym warto pamiętać, jest wtórnik napięciowy, jaki trzeba dać na wyjście sygnału. Obciążalność wyjścia przetwornika C/A jest niewielka i jeśli będziemy chcieli wysterować układ o rezystancji mniejszej niż 1 kΩ, to wówczas będziemy mieli zniekształcony sygnał. Dlatego należy zastosować wzmacniacz operacyjny, pracujący w układzie wtórnika napięciowego. Ciekawym rozwiązaniem jest też zastosowanie wzmacniacza programowalnego, który również znajduje się na płytce eXtrino XL. Tematów do omówienia jest jeszcze sporo i odcinki kursu mikrokontrolerów XMEGA mogą ciągnąć się w nieskończoność, jak brazylijska telenowela.

Dominik Leon Bieczyński
www.leon-instruments.pl



Rysunek 6. Przykładowy przebieg uzyskany generatorem z procesorem XMEGA