Zestaw DSP controlStick w praktyce

Szybkie przekształcenie Fouriera

Z pewnością nie będzie przesady w stwierdzeniu, że szybkie przekształcenie Fouriera jest jednym z najważniejszych narzędzi w DSP. Liczba dziedzin nauki i techniki, w których znalazła zastosowanie transformata FFT, jest ogromna. Nie można poważnie zajmować się systemami cyfrowego przetwarzania sygnałów bez zrozumienia i zastosowania FFT. W artykule podano informacje, jak wykorzystać bibliotekę FFT firmy Texas Instruments w połączeniu z mikrokontrolerami sygnałowym serii Piccolo, z zestawu startowego controlSTICK.

Przekształcenie Fouriera pozwala ustalić relację pomiędzy dziedziną czasu a dziedziną częstotliwości sygnału. Zwykle w technice mamy do czynienia z reprezentacją napięcia, prądu czy mocy sygnału w dziedzinie czasu (np. na ekranie oscyloskopu), natomiast po wykonaniu transformaty Fouriera można przejść na reprezentację sygnału w dziedzinie częstotliwości. Jako wynik przekształcenia Fouriera otrzymuje się reprezentację sygnału wejściowego w dziedzinie częstotliwości (widmo sygnału).

Jak wynika z definicji transformaty DFT, podczas obliczeń jest wykonywanych wiele zespolonych mnożeń i dodawań. Taki sposób wyznaczania transformaty Fouriera jest wysoce nieefektywny. Do połowy lat 60. ubiegłego wieku z dobrodziejstw przekształcenia Fouriera mogli korzystać tylko nieliczni. Dopiero opublikowanie w roku 1965 algorytmu szybkiego przekształcenia Fouriera (FFT – *Fast Fourier Transform*) pozwoliło na szerokie wykorzystanie mocy tkwiącej w transformacie. Należy tutaj podkreślić, że FFT nie jest przybliżeniem DFT, ale **FFT to jest DFT**, zatem wyniki obliczeń dla jednego i drugiego są identyczne. Algorytm szybkiego przekształcenia Fouriera wykorzystuje obecność w dyskretnym przekształceniu Fouriera nadmiarowych (niepotrzebnych) operacji, które są wyeliminowane. Algorytm FFT był wielokrotnie dokładnie omawiany w literaturze, dlatego też szczegóły jego funkcjonowania będą w artykule pominięte.

a) normalna kolejność

	MSB LSB
x(0)	000
x(1)	001
x(2)	010
x(3)	011
x(4)	100
x(5)	101
x(6)	110
x(7)	111

b) odwrócenie bitowe

	LSB MSB
x(0)	000
x(4)	100
x(2)	010
x(6)	110
x(1)	001
x(5)	101
x(3)	011
x(7)	111

Rys. 1. Adresowanie odwrócone bitowo (bit-reversed)

Biblioteka FFT

Texas Instruments udostępnia nieodpłatnie bibliotekę FFT przeznaczoną dla układów z rdzeniem TMS320C28x. Mikrokontroler sygnałowy zamontowany w zestawie controlSTICK (TMS320F28027) jest wyposażony w taki właśnie stałoprzecinkowy rdzeń. W związku z powyższym, można bibliotekę FFT wykorzystać w aplikacjach tworzonych dla układów Piccolo. Inżynierowie z Texas Instruments zaimplementowali popularny algorytm FFT o podstawie 2, a więc liczba przetwarzanych próbek musi się wyrażać za pomocą liczby będącej potęgą 2. Przykładowy program omówiony w dalszej części artykułu wykonuje obliczenia na, stosunkowo niewielkiej liczbie 128 próbek.

Biblioteka FFT jest podzielona na dwa podstawowe moduły: CFFT32 i RFFT32.

Jak nietrudno się domyślić, pierwszy służy do obliczania zespolonej (complex), natomiast drugi rzeczywistej (real) transformaty Fouriera. W celu użycia biblioteki we własnym projekcie, należy dodać do niego plik nagłówkowy *fft.h* oraz biblioteczny *fft.lib*. Niezbędne jest również umieszczenie w katalogu projektu używanych plików źródłowych. Cały kod biblioteki odpowiedzialny za FFT napisano w asemblerze, dzięki czemu jest zoptymalizowany dla architektury rdzenia C28x. Pliki z kodem asemblerowym zostały podzielone według wykonywanych zadań, takich jak operacje związane z akwizycją danych, obliczaniem transformaty itd.

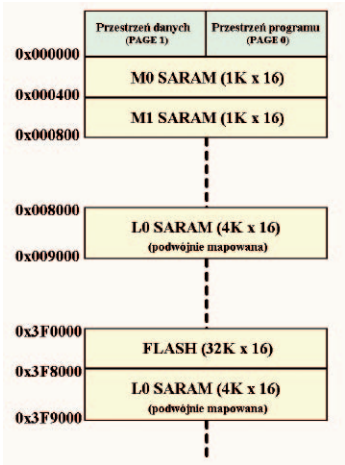
Pomimo rozdzielenia biblioteki na dwa moduły, sam proces obliczeń przekształcenia jest zawsze wykonywany przez moduł zespolony. Różnice polegają na długości przekazywanych buforów oraz interpretacji wyników. Kod bloku obliczeniowego jest zamieszczony w pliku cfft32c.a. Warto zapoznać się z jego zawartością, ponieważ można sobie uświadomić stopień skomplikowania zastosowanego algorytmu FFT.

Dodatkowe informacje:
Bibliotekę FFT firmy Texas Instruments wraz z dokumentacją i przykładowymi programami można pobrać ze strony TI spod adresu *http://focus.ti.com/docs/toolsw/folders/print/sprc081.html*

Adresowanie odwrócone bitowo

Algorytm FFT wymaga, aby dane wejściowe były uporządkowane w odpowiedniej kolejności. Załóżmy, że należy wyznaczyć 8 punktową FFT. W takim przypadku w buforze wejściowym próbki muszą być umieszczone w następującej kolejności: x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7). Kolejność ta nazywa się adresowaniem odwróconym bitowo (bit-reversed) lub też sortowaniem z odwróceniem bitowym. Wyjaśnienie, skąd wzięła się ta nazwa, pokazano na **rys. 1**. Jeśli podane na **rys. 1a** adresy zinterpretować od końca, czyli przyjąć, że bit MSB to LSB itp., wtedy kolejność próbek będzie taka, jak wymaga tego algorytm FFT (patrz **rys. 1b**).

Ciekawą cechą algorytmu FFT jest to, że obliczenia są przeprowadzane „w miejscu", czyli każdy etap przekształcenia wykorzystuje ten sam bufor (z tego powodu bufor da-



Rys. 2. Rozmieszczenie bloków pamięci RAM w przestrzeni pamięciowej układu TMS320F28027

List. 1. Fragment skryptu linkera

```
MEMORY
{
PAGE 0 :

         BEGIN            : origin = 0x000000,      length = 0x000002
         BOOT_RSVD        : origin = 0x000002,      length = 0x00004E
         RAMM0            : origin = 0x000050,      length = 0x0003B0
         progRAM          : origin = 0x008000,      length = 0x000C20

         RESET            : origin = 0x3FFFC0,      length = 0x000002
         BOOTROM          : origin = 0x3FF27C,      length = 0x000D44

PAGE 1 :

         RAMM1            : origin = 0x000480,      length = 0x000380
         dataRAM          : origin = 0x008C20,      length = 0x0003E0
}

SECTIONS
{
         codestart        : > BEGIN,        PAGE = 0
         ramfuncs         : > RAMM0,        PAGE = 0

         .text            : > progRAM,      PAGE = 0
         FFTtf            >   progRAM,      PAGE = 0

         .cinit           : > RAMM0,       PAGE = 0
         .pinit           : > RAMM0,       PAGE = 0
         .switch          : > RAMM0,       PAGE = 0
         .reset           : > RESET,       PAGE = 0, TYPE = DSECT
         .stack           : > RAMM1,       PAGE = 1
         .ebss            : > dataRAM,      PAGE = 1
         .econst          : > dataRAM,      PAGE = 1
         .esysmem : > RAMM1,       PAGE = 1

         FFTipcb  ALIGN(256)  > RAMM0,       PAGE = 1
         FFTmag                > dataRAM,    PAGE = 1
}
```

nych nazywa się ipcp – *in place computation buffer*). W efekcie, po wykonaniu wszystkich operacji związanych z wyznaczeniem FFT, w buforze ipcp będzie znajdował się wynik transformaty, już w naturalnej kolejności.

Biblioteka FFT ma funkcje, które mogą służyć do zmiany kolejności elementów tablicy z naturalnego na odwrócony bitowo. Są to funkcje (ściślej jest to kod asemblera wywoływany z poziomu Języka C) zakończone przyrostkiem *_brev*. Jeśli wykonywana ma być szybka transformata Fouriera w wersji rzeczywistej, to należy skorzystać z funkcji *RFFT_brev()*. Jej zadanie polega na odczytywaniu rzeczywistych danych umieszczonych w pamięci w naturalnej kolejności, a następnie na zapisaniu tychże danych w porządku odwróconym bitowo w formie liczb zespolonych. W liście argumentów funkcji należy przekazać odpowiednio: wskaźniki do źródłowej tablicy danych i do tablicy przeznaczenia oraz liczbę rzeczywistych próbek. Ponieważ funkcja *RFFT_brev()* wspiera algorytm liczenia „w miejscu”, więc tablice źródłowa i przeznaczenia mogą być tymi samymi obszarami pamięci.

Interpretacja wyników

Szybkie przekształcenie Fouriera jest potężnym narzędziem, ale aby je w pełni wykorzystać, należy rozumieć, co oznaczają wyniki tej transformaty. Określenie „rzeczywiste FFT” nie oznacza bynajmniej, że wynik będzie w postaci rzeczywistej, a jedynie, że sygnał wejściowy ma charakter rzeczywisty. Wynik przekształcenia jest zespolony. Tutaj rodzi się problem: jak interpretować otrzymane wyniki?

Najprostszym sposobem jest wyznaczenie modułu zespolonego wyniku lub jego kwadratu. Takie działanie pozwala otrzymać informację na temat widma częstotliwościowego sygnału. Omawiana w artykule biblioteka FFT ma zaimplementowany mechanizm liczenia kwadratu modułu wyniku transformaty. Wyniki przechowywane są w tablicy o długości N/2+1 elementów. Analiza zawartości tej tablicy pozwala określić składowe częstotliwościowe badanego sygnału.

Argument (faza) jest zwykle pomijany, jednak warto wspomnieć, że zawiera on informację o kształcie sygnału. Innymi słowy, jeśli informacja zakodowana jest w dziedzinie czasu (w kształcie sygnału), to interpretacja fazy może okazać się pomocna w analizie badanego sygnału.

Przeciek FFT

Oryginalne zależności Fouriera operują na nieskończonych sygnałach czasowych i wówczas dla sygnału harmonicznego otrzymujemy – jak w rzeczywistości – pojedynczy prążek. W przypadku uruchomienia algorytmu DFT dysponujemy zaledwie fragmentem sygnału ograniczonym pewnym oknem prostokątnym o długości N, a otrzymane widmo jest widmem sygnału złożonego z periodycznie powtarzającego się wycinka. Ponieważ wycinek sygnału zaczyna się i kończy bardzo gwałtownie, jego widmo ulega istotnej modyfikacji. Analizując sytuację przedstawioną na rys. 3, możemy dojść do wniosku, że gdyby „wycinany” fragment sygnału zawierał całkowitą liczbę okresów – efekt zniekształcenia nie powinien wystąpić. Jeśli więc sygnał wyświetlany na ekranie oscyloskopu – który definiuje nasze okienko czasowe – będzie zawierał całkowitą liczbę okresów, to cykliczne powtarzanie takiego wycinka da w efekcie sygnał taki sam jak oryginalny sygnał wejściowy. W każdym innym przypadku pojawią się zniekształcenia przypominające modulację, a właściwie manipulację, fazy.

Oryginalne równanie Fouriera operuje na sygnałach nieskończonych w dziedzinie czasu i dla takich sygnałów harmonicznych otrzymujemy po wykonaniu transformaty pojedyncze prążki. W rzeczywistości jednak algorytm FFT operuje na pewnej ograniczonej liczbie próbek, a otrzymane widmo sygnału jest złożone z tego periodycznie powtarzającego się wycinka. Ten wycinek zaczyna się i kończy bardzo gwałtownie, więc widmo ulega istotnej modyfikacji. Analizując całą sytuację, można dojść do wniosku, że transformata Fouriera da jednoznaczne wyniki tylko w przypadku obliczeń odpowiednio dobranych sygnałów. Jeśli sygnał wejściowy zawiera całkowitą liczbę okresów, to na wyjściu FFT otrzymany zostanie jednoznaczny wynik w postaci jednego prążka dla każdej składowej częstotliwości harmonicznej sygnału. We wszystkich pozostałych przypadkach przekształcenie Fouriera będzie jedynie przybliżeniem sygnału wejściowego. Zjawisko to nazywa się to przeciekiem DFT.

Skoro nieciągła zmiana sygnału na krańcach przedziału jest powodem przecieku, to można zaradzić jego powstawaniu, wycinając sygnał za pomocą okna o łagodnych zboczach. W praktyce realizuje się to, wykonując operację splotu ciągu wejściowego i funkcji okna. W jej wyniku wartości sygnału stają się takie same na obu końcach przedziału próbkowania. Jednocześnie operacja okienkowania redukuje moc sygnału i w konsekwencji zmniejsza też amplitudy wszystkich prążków widma, przy czym najbardziej minimalizuje jego składowe o wysokiej częstotliwości powodujące przeciek.

W zależności od aplikacji, stosuje się różne funkcje okien (o różnych kształtach). W bibliotece Texas Instruments zaimplementowano funkcje okien: Hamminga, Hanninga

List. 2. Niezbędne dla przekształcenia Fouriera deklaracje
```
/* Create an Instance of FFT module          */
#define     N   128

#pragma DATA_SECTION(ipcb, „FFTipcb”);
#pragma DATA_SECTION(mag, „FFTmag”);

RFFT32  fft=RFFT32_128P_DEFAULTS;
long ipcb[N+2];
long mag[N/2+1];

/* Define window Co-efficient Array  and place the
.constant section in ROM memory                */
const long win[N/2]=HAMMING128;

/* Create an instance of FFTRACQ module      */
RFFT32_ACQ  acq=FFTRACQ_DEFAULTS;
```

List. 3. Budowa struktury RFFT32
```
typedef struct {
        long *ipcbptr;
        long *tfptr;
        int size;
        int nrstage;
        long *magptr;
        long *winptr;
        long peakmag;
        int peakfrq;
        int ratio;
        void (*init)(void *);
        void (*calc)(void *);
        void (*split)(void *);
        void (*mag)(void *);
        void (*win)(void *);
        }RFFT32;
```

List. 4. Inicjalizacja modułu akwizycji danych i modułu FFT
```
/* Initialize acquisition module */
acq.buffptr=ipcb;
acq.tempptr=ipcb;
acq.size=N;
acq.count=N;
acq.acqflag=1;

/* Initialize FFT module */
fft.ipcbptr=ipcb;
fft.magptr=mag;
fft.winptr=(long *)win;
fft.init(&fft);
```

i Blackmana. Wszystkie w czterech wersjach dla 128-, 256-, 512- i 1024-punktowej FFT.

Aby zminimalizować niekorzystny efekt przecieku, w przykładowej aplikacji do okienkowania sygnału wejściowego zastosowano okno Hamminga.

Przykład rzeczywistego FFT

Sposób użycia biblioteki zostanie pokazany za pomocą aplikacji wyznaczającej rzeczywistą transformatę Fouriera z zestawu 128 próbek. Do realizacji projektu wykorzystano platformę eksperymentalną omówioną w artykule „Zestaw DSP controlSTICK w praktyce” zamieszczonym w poprzednim numerze EP. Warto w tym miejscu jedynie przypomnieć, że do sterowania i wizualizacji wyników wykorzystano aplikację stworzoną w MS Excel.

Program dla procesora sygnałowego jest w całości ładowany do pamięci RAM, co wymagało edycji skryptu linkera. Układ TMS320F28027 jest wyposażony w 12 kB pamięci RAM, jednak nie jest ona mapowana jako ciągły obszar w przestrzeni adresowej, lecz została podzielona na trzy bloki. Uproszczoną mapę, która dobrze pokazuje rozmieszczenie obszarów pamięci RAM, zamieszczono na **rys. 2**. Cała pamięć RAM jest dostępna zarówno dla danych, jak i programu. Najistotniejszy fragment skryptu linkera dla omawianego przykładu przedstawiono na **list. 1**. Jeśli aplikacja będzie rozwijana, to z pewnością zaistnieje potrzeba edycji skryptu, ponieważ niemal cała dostępna pamięć RAM została już użyta.

Budowa szkieletu programu dla procesora i sposób komunikacji z komputerem zostały już omówione w EP 1/2010, dlatego w bieżącym artykule przedstawione zostaną jedynie fragmenty kodu związane bezpośrednio z obliczaniem FFT.

Deklaracje niezbędnych dla przekształcenia Fouriera zmiennych i stałych zamieszczono na **list. 2**. Dyrektywy preprocesora (*#pragma*) nakazują umieszczenie zmiennych: bufora obliczeń (*ipcp*) i tablicy modułu (*mag*) w przeznaczonych dla nich przestrzeniach adresowych. Obie sekcje zostały określone w skrypcie linkera. Należy zwrócić uwagę na utworzenie zmiennej fft, która jest de facto strukturą zdefiniowaną tak jak to przedstawia **list. 3**. Wszystkie operacje powiązane z FFT wykonuje się przez odwołania do pól struktury. Przykładowo obliczenie transformaty odbywa się przez wywołanie metody *calc*, czyli: *fft.calc(&fft)*.

Jeszcze przed rozpoczęciem zbierania próbek sygnału wejściowego należy przeprowadzić inicjalizację używanych elementów biblioteki. Kod odpowiedzialny za przygotowanie do pracy bloków wykorzystywanych w przykładowej aplikacji pokazano na **list. 4**. Jako pierwszy konfigurowany jest moduł akwizycji danych (acquisition module), a następnie moduł FFT. Niezbędne parametry, jakie należy podczas inicjalizacji do struktury fft przekazać, to:

- adres początku bufora obliczeniowego ipcb,
- adres początku tablicy mag, w której będzie przechowywany moduł FFT,
- adres tablicy współczynników funkcji okna win.

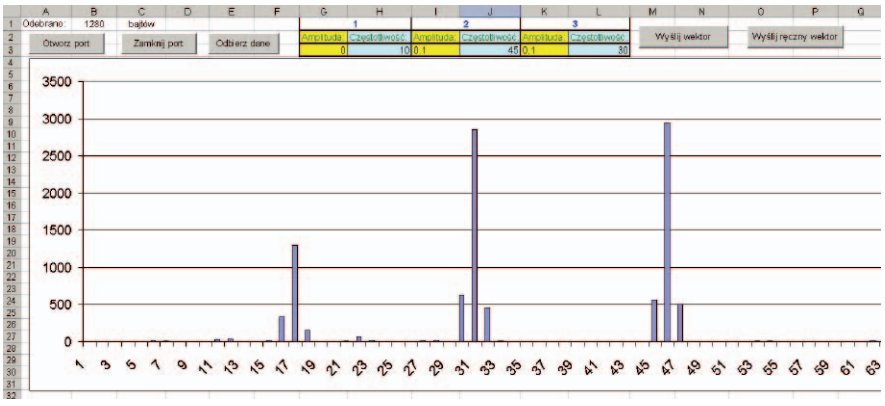
Następnie, po przygotowaniu do pracy wszystkich urządzeń peryferyjnych, mikrokontroler sygnałowy rozpoczyna obliczenia i jeśli zostanie zgłoszona taka potrzeba, wysyła do komputera zestaw danych.

Za przeprowadzenie właściwych obliczeń odpowiada fragment programu przedstawiony na **list. 5**. Zostanie on wykonany tylko wtedy, gdy zgromadzona będzie wymagana liczba próbek (w tym przykładzie 128).

W stosunku do aplikacji sterującej przedstawionej w EP1/10 tutaj należy zmienić jedynie typ wykresu oraz zmniejszyć zakres wyświetlanych danych. Istnieje oczywiście możliwość zaprogramowania mikrokontrolera sygnałowego tak, aby wysyłał do komputera inny zestaw danych niż kwadrat modułu transformaty, wszystko zależy od tego, co chce się zbadać.

Efekty działania omówionego wyżej programu w połączeniu z excelową aplikacją pokazano na **rys. 3**. Na rysunku widać, że pomimo wysłania pełnej liczby okresów składowych sygnałów, otrzymano więcej niż jeden prążek DFT dla każdej składowej. Dobitnie pokazuje to, jak bardzo różni się przetwarzanie sztucznie generowanych sygnałów w specjalizowanych pakietach od przetwarzania w rzeczywistych układach. Praktyka nie pozwala na ucieczkę od problemów związanych z analogową naturą prądu.

Krzysztof Paprocki
paprocki.krzysztof@gmail.com



Rys. 3. Wynik szybkiego przekształcenia Fouriera

List. 5. Obliczenia FFT
```
if (acq.acqflag==0)     // If the samples are acquired
{
        RFFT32_brev(ipcb,ipcb,N);
        RFFT32_brev(ipcb,ipcb,N);  // Input samples in Real Part

        fft.win(&fft);
        RFFT32_brev(ipcb,ipcb,N);
        RFFT32_brev(ipcb,ipcb,N);  // Input after windowing

        fft.calc(&fft);
        fft.split(&fft);
        fft.mag(&fft);
        acq.acqflag=1;      // Enable the next acquisition
}
```