Zastosowania procesorów sygnałowych

WPROWADZENIE DO PROCESORÓW SYGNAŁOWYCH

Opracowanie: Grzegorz Szwoch

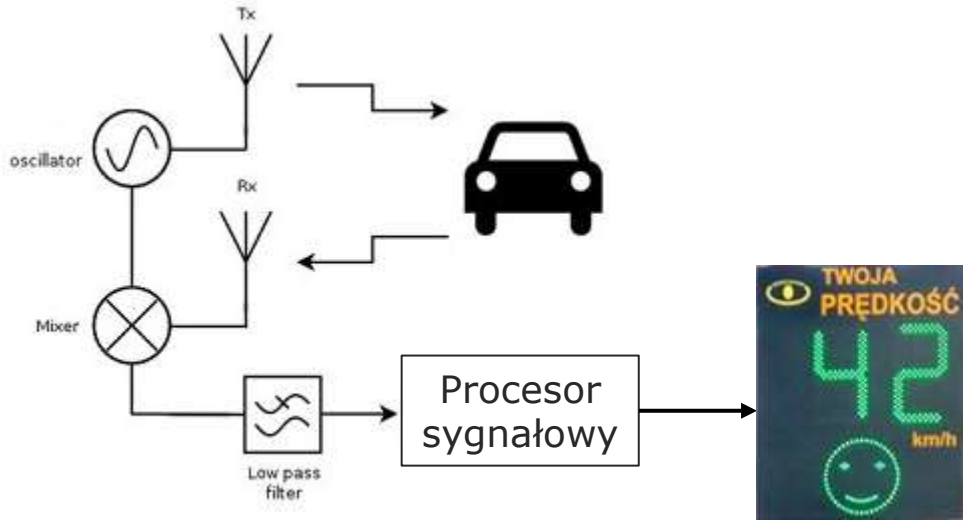
Politechnika Gdańska, Katedra Systemów Multimedialnych

Wprowadzenie

- Sygnał definiujemy jako ciągły zbiór wartości pozyskiwanych z danego źródła (czujnika, mikrofonu, itp.), położonych na osi czasu.
- Sygnał cyfrowy – zbiór próbek pozyskiwanych ze źródła, zapisanych w postaci bitów.
- Cyfrowe przetwarzanie sygnału za pomocą cyfrowych algorytmów obejmuje:
  - obróbkę sygnału, np. filtrację w celu usunięcia zakłóceń,
  - analizę sygnału, np. obliczenie wartości średniej, wyodrębnienie informacji,
  - przesłanie wyników, np. wysłanie informacji sterującej.
- Cyfrowy procesor sygnałowy (*digital signal processor*) – układ elektroniczny wyspecjalizowany w cyfrowym przetwarzaniu sygnałów, zwykle wykonywanym „w czasie rzeczywistym”, w odróżnieniu od procesorów ogólnego przeznaczenia (CPU).

Przykład praktycznego zastosowania

- Mikrofalowy czujnik radarowy jest podłączony do procesora sygnałowego.
- Wiązka radarowa jest wysyłana (Tx), odbierana jest wiązka odbita (Rx).
- Procesor sygnałowy oczyszcza (filtruje) sygnał oraz oblicza częstotliwość sygnału, a na jej podstawie oblicza prędkość pojazdu.
- Wynik przetwarzania steruje wyświetlaczem prędkości.



Dlaczego nie mikroprocesor?

Dlaczego do przetwarzania sygnału nie użyć zwykłego mikroprocesora (CPU)?

- Mikroprocesor jest układem ogólnego przeznaczenia – wykonuje różnorodne operacje, nie traktuje przetwarzania sygnału w szczególny sposób.
- Wymaga systemu operacyjnego; klasyczne systemy nie zapewniają przewidywanego czasu wykonania operacji (potrzebny jest system operacyjny czasu rzeczywistego).
- Wymagają dodatkowych elementów (pełny mikrokomputer), ze względu na duże rozmiary nie nadają się do systemów wbudowanych (*embedded systems*).
- Wymagają dużych częstotliwości taktowania zegara, mają wysokie zużycie energii, wysoki koszt eksploatacji.

Współcześnie, mikrokomputery (zwykle oparte na procesorach ARM) często zastępują klasyczne procesory sygnałowe w ich dotychczasowych zastosowaniach.

Dlaczego procesor sygnałowy?

- Architektura i lista rozkazów wyspecjalizowana do przetwarzania próbek cyfrowego sygnału.
- Zwykle niskie częstotliwości taktowania, zwłaszcza w procesorach stałoprzecinkowych.
- Małe zużycie energii w porównaniu do mikroprocesora.
- Wbudowane liczne interfejsy – łatwość w pozyskiwaniu próbek sygnałów.
- Małe rozmiary – możliwość zastosowania w systemach wbudowanych.
- Stosunek czasu przetwarzania do „kosztów” (zużycia zasobów) znacznie lepszy niż dla mikroprocesora.

Zastosowania

Kiedy zastosujemy procesor sygnałowy?

- do złożonego przetwarzania sygnału w czasie rzeczywistym,
- jako element urządzenia (np. aparatu fotograficznego),
- przy konieczności niskiego zużycia energii (np. zasilanie z baterii),
- gdy przetwarzanie składa się z typowych operacji PS (filtry, FFT).

Kiedy zastosujemy mikrokomputer?

- do przetwarzania tylko *offline* („na plikach”),
- gdy jest konieczność stosowania systemu operacyjnego (np. w smartfonie),
- gdy mikrokomputer i tak jest potrzebny do wykonania innych operacji,
- gdy potrzebna jest elastyczność, a zużycie energii i wymiary urządzenia są mniej istotne.

Przetwarzanie sygnału w trybie online

Procesory sygnałowe stosujemy zazwyczaj do przetwarzania sygnałów „na żywo” (w trybie *online*).

- Sygnał jest nieskończony – nowe próbki cały czas napływają.
- Każda próbka musi zostać przetworzona zanim nadejdzie kolejna (przetwarzanie w czasie rzeczywistym, próbka po próbce).
- Ewentualnie, próbki mogą być przetwarzane po zebraniu ich wymaganej liczby – przewarzanie blokowe (np. obliczanie FFT).

Przykład: analiza dźwięku z mikrofonu.

- Częstotliwość próbkowania jest równa 48 000 próbek na sekundę.
- Maksymalny czas przetwarzania jednej próbki: 1 / 48 = 0,0208 ms.

Przetwarzanie w czasie rzeczywistym

Pojęcie „przetwarzanie w czasie rzeczywistym” (*real-time processing*) jest definiowane różnorodnie, zależnie od kontekstu.

Na potrzeby cyfrowego przetwarzania sygnałów możemy przyjąć definicję: przetwarzanie w czasie rzeczywistym wymaga, aby:

$$tp + tn < ts \quad \text{czyli} \quad tb > 0$$

gdzie:

- *ts* – czas pomiędzy kolejnymi próbkami sygnału na wejściu,
- *tp* – czas przetwarzania próbki,
- *tn* – czas narzutu – operacji nie związanych z przetwarzaniem sygnału,
- *tb* – czas bezczynności – czekania na kolejną próbkę.

Czas narzutu *ts* w systemie czasu rzeczywistego powinien być stały.

Algorytmy przetwarzania sygnału

Algorytmy uruchamiane na procesorach sygnałowych wykorzystują podstawowe procedury przetwarzania sygnału:

- mnożenie, dodawanie, operacje logiczne (również wykonywane wektorowo),
- przekształcenie Fouriera (FFT) i kosinusowe (DCT),
- splot / filtracja FIR,
- korelacja, autokorelacja,
- filtracja IIR,
- mnożenie macierzy

Procesor sygnałowy jest zoptymalizowany do wykonywania tych operacji.
Na ich podstawie budowane są złożone algorytmy, np. kompresji sygnału.

Moduły uruchomieniowe

Moduł uruchomieniowy (*evaluation board/module*):

- płytka zawierająca procesor sygnałowy, interfejsy zewnętrzne, pomocnicze układy (kodeki, pamięć), itp.,
- służy do uruchamiania, testowania, poprawiania, optymalizowania algorytmów tworzonych przez programistę,
- płytka współpracuje z komputerem osobistym (przez USB),
- pozwala na wykonywanie programu krok po kroku i podgląd stanu procesora, za pomocą *debug probe,*
- w docelowym urządzeniu montowany jest sam procesor, nie cała płytka uruchomieniowa.

Moduły uruchomieniowe

Moduł uruchomieniowy z procesorem Texas Instruments C5535, wykorzystywany podczas realizacji laboratorium ZPS

TMDX5535EZDSP USB Stick Development Kit

Tworzenie programu na procesor sygnałowy

- Kod maszynowy (*machine code*) – binarny zestaw instrukcji dla procesora (np. mnożenie, skok, pętla, itp.).
- Programista tworzy kod źródłowy (*source code*) programu w postaci tekstowej:
  - asembler – język niskopoziomowy,
  - C, C++, itp. – język wysokopoziomowy.
- Kompilator (*compiler*) przekształca kod źródłowy na kod maszynowy.
- Konsolidator (*linker*) łączy skompilowane moduły (*modules*) w program wykonywalny (*executable*).
- Gotowy program jest uruchamiany na procesorze.

Asembler

- Asembler (*assembler*) jest tekstową reprezentacją kodu maszynowego.
- Instrukcje procesora są zapisywane w formie mnemoników,
  np. MOV – skopiuj dane, MPY – przemnóż, itp.
- Kod asemblera jest pisany na konkretny typ procesora.
- Pracuje się bezpośrednio na procesorze (np. na rejestrach).
- Programista ma niemal pełną kontrolę nad wynikowym kodem maszynowym.
- Można w ten sposób pisać zoptymalizowane algorytmy.
- Jest to trudne, wymaga dużego doświadczenia i dobrej znajomości procesora.
- (Nie piszemy kodu asemblera na laboratorium ZPS, ale wykorzystujemy gotowe algorytmy napisane w asemblerze.)

Asembler

Fragment przykładowego programu w asemblerze na PS:

```
 || RPTBLOCAL OuterLoopEnd-1                          ;outer loop: process a new input
                MOV     *AR0+ << #16, AC1                 ; HI(AC1) = x(n)
                ||RPTBLOCAL     InnerLoopEnd-1            ;inner loop: process a bi-quad
                        NOP_16                            ; CPU_116: Remark 5682
                     || MPYM    *AR1+, AC1, AC0           ; AC0 = b0*x(n)
                        MACM    *AR1+, *(AR6+T0), AC0     ; AC0 += b1*x(n-1)
                        MACM    *AR1+, *AR6, AC0          ; AC0 += b2*x(n-2)
                        MOV     HI(AC1), *(AR6+T1)        ; x(n) replaces x(n-2)
                        MASM    *AR1+, *(AR4+T0), AC0     ; AC0 -= a0*y(n-1)
                        MASM    *AR1+, *AR4, AC0, AC1     ; AC1 -= a1*y(n-2)
                        SFTS    AC1, #1                   ; AC1 *= 2 (correction for Q14)
                        MOV     rnd(HI(AC1)), *(AR4+T1)   ; y(n) replaces y(n-2)
InnerLoopEnd:
                AMOV    AR7, AR1                          ;reinitialize coeff pointer
              || MOV    rnd(HI(AC1)), *AR2+               ;store result to output buffer
OuterLoopEnd:

        MOV     AR4, *AR3                                 ; Update delay pointer
```

Język C

- Język C (czasami też C++) jest często stosowany w programowaniu PS jako język „wysokiego poziomu”.
- Polegamy na kompilatorze, że stworzy z kodu źródłowego C wystarczająco wydajny kod maszynowy.
- Często nie jest to możliwe, kompilator nie potrafi odgadnąć wszystkich intencji programisty, często „gra bezpiecznie” (program ma działać poprawnie, niekoniecznie najszybciej).
- Trzeba używać specjalnych dyrektyw kompilatora (*pragma*).
- W porównaniu z asemblerem, wynikowy kod jest zwykle wolniejszy i zajmuje więcej pamięci.
- Pisanie programów jest za to o wiele szybsze i prostsze.
- (Kod na laboratorium ZPS będzie tworzony w języku C.)

Język C

Fragment przykładowego programu w C na DSP:

```
// Real FFT of length N/2
for (i = 0; i < N / 2; i++) {
          pRFFT_In[2 * i] = pInput[2 * i];           //arrange real input sequence to
          pRFFT_In[2 * i + 1] = pInput[2 * i + 1];   //N/2 complex sequence..
}

memcpy (pRFFT_InOrig, pRFFT_In, N * sizeof (float));
tw_gen (w, N / 2);
split_gen (A, B, N / 2);
twiddle = (float *) w;

// Forward FFT Calculation using N/2 complex FFT..
DSPF_sp_fftSPxSP (N / 2, pRFFT_In, twiddle, pTemp, brev, rad, 0, N / 2);
// FFT Split call to get complex FFT out of length N..
FFT_Split (N / 2, pTemp, A, B, pRFFT_Out);

// Inverse FFT calculation
// IFFT Split call to get complex Inv FFT out of length N..
IFFT_Split (N / 2, pRFFT_Out, A, B, pTemp);
// Inverse FFT Calculation using N/2 complex IFFT..
DSPF_sp_ifftSPxSP (N / 2, pTemp, twiddle, pRFFT_InvOut, brev, rad, 0, N / 2);
```

C i Asembler razem

- Czy można połączyć oba języki w jednym programie? Tak!
- Konsolidator pozwala łączyć moduły napisane w C i w asemblerze.
- Krytyczne operacje przetwarzania są pisane w asemblerze, zoptymalizowane pod kątem szybkości (np. obliczanie FFT).
- Ogólna „logika” programu jest pisana w C.
- Zazwyczaj możemy wykorzystać zoptymalizowane procedury (np. FFT, filtry) napisane w asemblerze dla naszego procesora przez innych programistów, łącząc je z naszym kodem.
- Np. Texas Instruments dostarcza dla naszego procesora C5535 bibliotekę *DSPLIB*, zawierającą zoptymalizowane implementacje najczęściej wykorzystywanych operacji przetwarzania sygnałów.

Środowisko programistyczne

Do tworzenia oprogramowania na procesory sygnałowe używa się środowiska programistycznego złożonego z:

- narzędzi programistycznych (IDE)
  - edytora kodu źródłowego,
  - kompilatora,
  - debuggera,
- bibliotek programistycznych (SDK)
  - funkcje obsługi procesora (*Processor SDK*), w tym system operacyjny,
  - funkcje obsługi płytki uruchomieniowej (*Board SDK*),
  - funkcje przetwarzania sygnału (np. *DSPLIB*),
  - funkcje pomocnicze (np. obsługa sieci).

Etap tworzenia programu

- Moduł uruchomieniowy podłączony do komputera PC przez USB.
- Program skompilowany w trybie *Debug* – wyłączone optymalizacje kodu.
- Skompilowany program jest przesyłany na procesor sygnałowy i uruchamiany.
- Możliwość debugowania – zatrzymania programu, sprawdzenia stanu zmiennych, znalezienia błędów.
- Sprawdzenie poprawności działania programu.
- Do testowania wydajności należy skompilować program w trybie *Release* – włączone optymalizacje kodu. Uwaga: taki program nie nadaje się do debugowania!

Gotowy program

- Program jest gotowy gdy działa zgodnie z oczekiwaniami – bez błędów i wystarczająco szybko.
- Program jest kompilowany w trybie *Release*.
- Program jest wgrywany do pamięci *flash* samodzielnego procesora (bez płytki) za pomocą specjalnego modułu.
- Procesor jest gotowy do montażu w docelowym urządzeniu.
- Nie ma już możliwości wglądu w pracę procesora – mamy tylko wyniki jego pracy.
- Dlatego trzeba włożyć wystarczająco dużo wysiłku w tworzenie prawidłowo działającego programu.

Miary wydajności procesora sygnałowego

Wydajność obliczeniowa:

- MIPS – liczba milionów instrukcji na sekundę,
- FLOPS – liczba operacji zmiennoprzecinkowych na sekundę, zwykle z przedrostkiem, np. MFLOPS – mega (miliony),
- MMACS – liczba milionów operacji MAC ($x \leftarrow x + a \times b$), specyficznych dla DSP, na sekundę.

Wydajność energetyczna:

- zużycie mocy na 1 MHz częstotliwości zegara procesora, w mW/MHz, przy podanym napięciu zasilania, mierzone pod obciążeniem (*active*) i w spoczynku (*standby*).

Cykle procesora

- Procesor jest taktowany zegarem o ustalonej częstotliwości, np. 100 MHz.
- Cykl procesora to jeden takt sygnału zegarowego. Np. 100 MHz oznacza 100 milionów cykli na sekundę.
- Każda instrukcja procesora wymaga określonej liczby cykli procesora (zwykle jeden cykl, czasami więcej) do jej wykonania.
- Budżet procesora oznacza liczbę cykli, jaką możemy przeznaczyć na operacje przetwarzania.
- Przykład: taktowanie 100 MHz, przetwarzamy próbki dźwięku z częstotliwością 48 kHz → na jedną próbkę mamy ok. 2083 cykli procesora.
- Jeżeli nie wykorzystujemy budżetu, często możemy obniżyć częstotliwość taktowania procesora (np. do 80 MHz) i oszczędzić energię.

Alternatywy dla procesorów sygnałowych

Współczesne alternatywy dla klasycznych procesorów sygnałowych:

- procesory hybrydowe (np. ARM + DSP),
- układy SoC (*system on chip*) - zwykle mikroprocesory ARM + GPU + pamięć, czasami posiadają rdzenie DSP,
- układy FPGA – programowalna architektura sprzętowa,
- układy ASIC – mikroprocesory o stałej architekturze, dostosowanej do programu,
- GPU – procesory graficzne, wspomaganie zrównoleglonych obliczeń,
- NPU – *neural processor unit*, do obliczeń związanych z sieciami neuronowymi.