# *WPROWADZENIE DO PROCESORÓW SYGNAŁOWYCH*

Opracowanie: Grzegorz Szwoch

Politechnika Gdańska, Katedra Systemów Multimedialnych

# *Organizacja zajęć*

Przedmiot „Zastosowania procesorów sygnałowych”:

- Semestr 5 – wykład (1 godz. tygodniowo).
- Semestr 6 – projekt (2 godz. co 2 tygodnie).
  Studenci samodzielnie wykonują zadania projektowe podczas zajęć w laboratorium, na układach DSP.

Odpowiedzialny za przedmiot:

Grzegorz Szwoch

greg@multimed.org

pokój WETI 732

# *Kryteria zaliczenia wykładu*

- Kolokwium zaliczeniowe – na ostatnim wykładzie.
- 10 pytań × 3 pkt, razem max. 30 punktów.
- Kryterium zaliczenia: minimum 16 punktów.
- Kolokwium poprawkowe: w sesji poprawkowej.
- Ewentualne dodatkowe możliwości zaliczenia TYLKO po spełnieniu OBU warunków:
  - brakuje maksymalnie 3 punkty,
  - obecność na min. 80% wykładów.
- Obecność na wykładach będzie sprawdzana.

# *Materiały do nauki*

Prezentacje z wykładów – dostępne na stronie multimed.org, dział „Materiały pomocnicze”.

Sugerowana literatura uzupełniająca:

- T. Zieliński: „Cyfrowe przetwarzanie sygnałów w telekomunikacji”. PWN 2014.
- T. Zieliński: „Cyfrowe przetwarzanie sygnałów. Od teorii do zastosowań”. WKŁ 2005.
- S.W. Smith: „The Scientist and Engineer's Guide to Digital Signal Processing”. www.dspguide.com
- „Wszystko jest w Internecie” ☺

# *Ramowy program wykładu*

- Wprowadzenie do procesorów sygnałowych (PS)
- Architektura PS
- Systemy operacyjne PS
- Systemy liczbowe
- Interfejsy PS
- Przekształcenie Fouriera i splot
- Filtry cyfrowe FIR i IIR
- Generowanie sygnałów
- Zaawansowane algorytmy PS
- Zastosowania PS w aparatach fotograficznych i kamerach
- Medyczne zastosowania PS
- Kolokwium zaliczeniowe

# *Motywacja*

Dlaczego mówimy o tym na kierunku Telekomunikacja?

- Procesory sygnałowe znajdują się w wielu urządzeniach „telekomunikacyjnych”, zarówno profesjonalnych, jak i konsumenckich.
- W przyszłej pracy zawodowej, studenci prawdopodobnie zetkną się z zastosowaniem procesorów sygnałowych.
- Dobrze jest znać zasady działania PS i implementacje typowych algorytmów cyfrowego przetwarzania sygnałów.
- Chcemy pokazać, że wiedza przedstawiana na przedmiotach *Przetwarzanie sygnałów* i *Przetwarzanie dźwięków i obrazów* nie jest czystą teorią – jest ona wykorzystywana w praktyce, z użyciem procesorów sygnałowych!

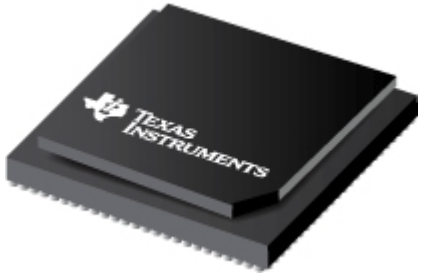
# *Procesor sygnałowy*

Próba definicji:

Cyfrowy procesor sygnałowy (*digital signal procesor*, DSP) jest to układ elektroniczny wyspecjalizowany w optymalnym przetwarzaniu próbek cyfrowego sygnału, wykonując powtarzalne operacje na kolejnych próbkach sygnału.

Określenie „optymalny” dotyczy:

- czasu przetwarzania próbek („w czasie rzeczywistym”),
- niskiego zużycia energii.




Te cechy odróżniają DSP od mikroprocesorów ogólnego przeznaczenia.

# *Metody przetwarzania sygnału*

Są dwie zasadnicze metody przetwarzania sygnału.

- Przetwarzanie *offline* („na plikach”)
  - mamy cały sygnał do dyspozycji od razu,
  - czas przetwarzania nie jest krytyczny,
  - tutaj klasyczny procesor lepiej się sprawdzi.
- Przetwarzanie *online* („w czasie rzeczywistym”):
  - sygnał jest nieskończony – próbki cały czas napływają,
  - każda próbka musi zostać przetworzona zanim nadejdzie kolejna,
  - ew. przetwarzanie dotyczy bloku próbek,
  - to jest typowe zastosowanie procesorów sygnałowych.

# *Przetwarzanie w czasie rzeczywistym*

- *ts* – czas pomiędzy kolejnymi próbkami sygnału
- *tp* – czas przetwarzania próbki
- *tn* – czas narzutu – operacji nie związanych z przetwarzaniem
- *tb* – czas bezczynności – czekanie na dane

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:

$$tp + tn < ts$$

$$tb > 0$$

# *Dlaczego nie mikroprocesor?*

Dlaczego do przetwarzania w czasie rzeczywistym nie użyć zwykłego mikroprocesora (CPU)?

- Mikroprocesor jest układem ogólnego przeznaczenia – wykonuje różne operacje, nie traktuje szczególnie przetwarzania próbek sygnału.
- Wymaga systemu operacyjnego, klasyczne systemy nie zapewniają przewidywanego czasu wykonania operacji (zmienny narzut).
- Wymagają osobnych elementów – pełny mikrokomputer, nie nadają się do systemów wbudowanych (duże rozmiary).
- Duże częstotliwości taktowania zegara, wysokie zużycie energii, wysoki koszt eksploatacji.

# *Dlaczego procesor sygnałowy?*

- Architektura i lista rozkazów wyspecjalizowana do przetwarzania próbek cyfrowego sygnału.
- Zwykle niskie częstotliwości taktowania (rzędu 100 MHz).
- 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.

# *Kiedy zastosować procesor sygnałowy?*

Kiedy zastosujemy procesor sygnałowy?

- złożone przetwarzanie w czasie rzeczywistym
- jako element urządzenia (np. aparatu fotograficznego)
- konieczność niskiego zużycia energii (np. z baterii)
- gdy przetwarzanie składa się z typowych operacji DSP

Kiedy zastosujemy mikrokomputer?

- przetwarzanie tylko offline
- konieczność stosowania systemu operacyjnego (np. smartfon)
- gdy mikrokomputer i tak jest potrzebny do innych operacji
- gdy potrzebna jest elastyczność, a zużycie energii i wymiary urządzenia są mniej istotne

# *Algorytmy przetwarzania sygnału*

Algorytmy pisane na procesor sygnałowy wykorzystują podstawowe procedury przetwarzania sygnału:

- mnożenie, dodawanie, operacje logiczne – zazwyczaj wykonywane na wektorze wartości
- przekształcenie Fouriera (FFT) i kosinusowe (DCT)
- splot / filtracja FIR, korelacja, autokorelacja
- filtracja IIR
- interpolacja
- mnożenie macierzy

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

# *Kod maszynowy*

- Kod maszynowy (*machine code*) – binarny zestaw instrukcji dla procesora (np. mnożenie, skok, pętla, itp.).
- Programista tworzy kod źródłowy (*source code*) programu w postaci tekstowej.
- 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 programy.
- Jest to trudne, wymaga dużego doświadczenia i dobrej znajomości architektury procesora.
- (Nie wymagamy stosowania asemblera na projekcie ZPS)

# Asembler

Fragment przykładowego programu w asemblerze na DSP:

```
 || 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 DSP jako język „wysokiego poziomu”.
- Polegamy na kompilatorze że stworzy z kodu źródłowego C optymalny kod maszynowy.
- Często nie jest to możliwe, kompilator nie potrafi odgadnąć wszystkich intencji programisty, często „gra bezpiecznie”.
- 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.
- (Projekt 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? Tak!
- Konsolidator pozwala łączyć moduły napisane w C i w asemblerze, kompilator obsługuje też asembler.
- Krytyczne operacje przetwarzania są pisane w asemblerze, zoptymalizowane pod kątem szybkości.
- Ogólna „logika” programu jest pisana w C.
- Zazwyczaj możemy wykorzystać zoptymalizowane procedury (np. FFT) napisane w asemblerze dla naszego procesora przez innych programistów, łącząc je z naszym kodem.
- Np. Texas Instruments dostarcza bibliotekę DSPLIB, zawierającą zoptymalizowane implementacje podstawowych operacji DSP.

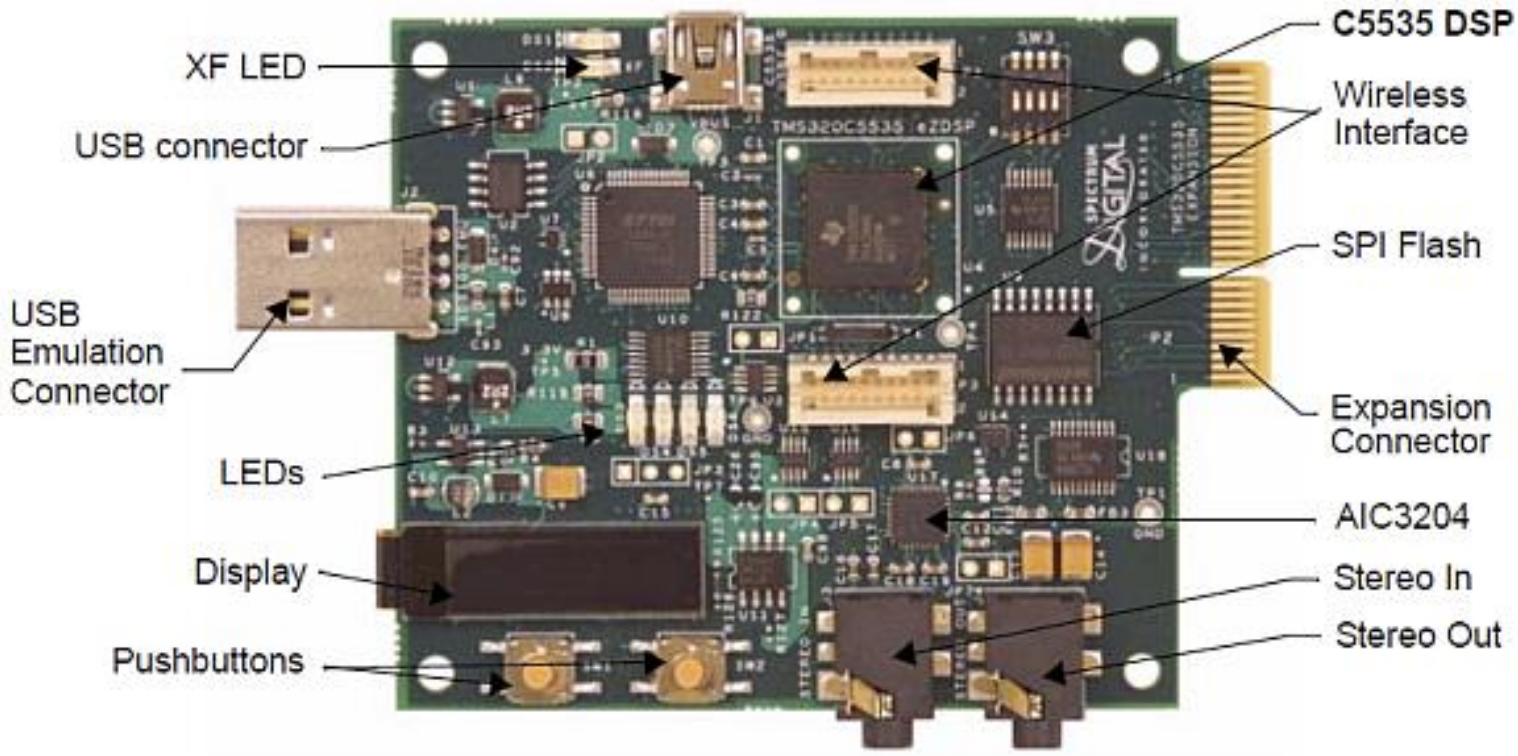
# *Moduły uruchomieniowe*

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 z procesorem Texas Instruments C5535, wykorzystywany w realizacji projektu ZPS




TMDX5535EZDSP USB Stick Development Kit

# *Środowisko programistyczne*

Do tworzenia oprogramowania na procesory sygnałowe używa się środowiska 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.
- 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*)

# *Producenci DSP*

Wiodący producenci procesorów sygnałowych:

- Texas Instruments (www.ti.com)
- Analog Devices (www.analog.com)
- NXP Semiconductors (www.nxp.com)
- Freescale (www.freescale.com)
- XMOS (www.xmos.com)
- CEVA (www.ceva-dsp.com)

# *Przykład zastosowania*

Procesor w cyfrowym aparacie fotograficznym – funkcje:

- automatyczny dobór parametrów ekspozycji
- automatyczne nastawianie ostrości
- automatyczny wybór programu
- demozaikowanie danych z matrycy
- redukcja szumu
- łączenie ujęć w obraz HDR
- efekty specjalne
- wykrywanie twarzy w obrazie
- podążanie za obiektem, utrzymywanie ostrości
- kompresja JPG, MPEG

# *Zastosowania – urządzenia konsumenckie*

- aparaty cyfrowe i kamery
- odtwarzacze muzyki i filmów
- syntezatory dźwięku i samplery
- zabawki elektroniczne
- elektroniczna niania
- sprzęt AGD, np. pralki
- zdalnie sterowane modele, drony
- „inteligentny dom”

# *Zastosowania - telekomunikacja*

- kodowanie/dekodowanie, kompresja dźwięku i obrazu
- redukcja szumu i zakłóceń
- usuwanie echa akustycznego i telekomunikacyjnego
- poprawa zrozumiałości mowy w obecności zakłóceń
- systemy telekonferencyjne – wzmacnianie kierunku mówcy
- rozpoznawanie i synteza mowy, sterowanie głosowe
- monitoring – analiza zawartości obrazu
- stacje pomiarowe – analiza danych z czujników

# *Zastosowania - medycyna*

- analiza obrazu z badań (USG, tomografia, itp.)
- analiza fal mózgowych (EEG)
- aparaty słuchowe
- elektroniczna krtań, protezy
- urządzenia do rehabilitacji
- monitorowanie stanu zdrowia pacjenta, telemedycyna
- monitorowanie wysiłku fizycznego („asystent joggera”)

# *Zastosowania - inne*

Zastosowania militarne:

- radar, sonar, LiDAR – detekcja obiektów, pomiar prędkości
- kryptografia i watermarking – zabezpieczanie danych
- nawigacja
- naprowadzanie pocisków

Zastosowania w przemyśle samochodowym:

- zarządzanie pracą elementów pojazdu (np. ABS)
- nawigacja
- zestawy głośnomówiące
- wykrywanie przeszkód
- sterowanie głosowe

# *Systemy wieloprocesorowe i wielordzeniowe*

- Czasami wydajność procesora nie wystarcza do zapewnienia przetwarzania w czasie rzeczywistym.
- Możliwe rozwiązania:
  - procesory sygnałowe wielordzeniowe,
  - układ wieloprocesorowy – połączenie kilku procesorów.
- Przykład: analiza obrazu „piksel po pikselu” – można podzielić obraz na kilka części, każdy procesor/rdzeń analizuje swój fragment, wyniki są łączone.
- Synchronizacja jest znacznie trudniejsza niż na zwykłym procesorze w komputerze.
- Czas narzutu jest duży, programowanie jest kłopotliwe.

# *Procesory hybrydowe*

- Połączenie dwóch typów w jednym układzie:
  - rdzeń (lub kilka) klasycznego procesora, zwykle ARM,
  - zwykle kilka rdzeni procesora sygnałowego.
- Na głównym rdzeniu uruchamiany jest system operacyjny, np. Linux.
- Program na głównym rdzeniu zarządza obliczeniami.
- Rdzenie DSP wykonują obliczenia.
- Skomplikowane programowanie (systemy kolejek).
- Duży narzut obliczeń.
- Duża elastyczność tworzenia programu.

# *Procesory hybrydowe - przykłady*

66AK2H12 Keystone (Texas Instruments)

- 4 rdzenie ARM A15
- 8 rdzeni DSP C66x

Snapdragon 835 (Qualcomm, 2019)

- CPU: 8 rdzeni ARM różnego typu (1+3+4)
- GPU (procesor graficzny)
- DSP: Hexagon 690, wielowątkowy, + *tensor accelerator*

# *Alternatywy dla DSP: FPGA*

## FPGA (*Field-Programmable Gate Array*)

- Układ, którego architektura sprzętowa może być konfigurowana przez programistę.
- Macierz programowalnych bloków logicznych z pamięcią – połączenia między nimi mogą być modyfikowane.
- Architektura może zostać zoptymalizowana pod kątem danego algorytmu (zastosowania).
- FPGA zastępują DSP w rozwiązaniach, w których wydajność DSP jest niewystarczająca (obliczenia równoległe, przetwarzanie z bardzo dużą częstotliwością).
- Duża elastyczność.
- Wysoki koszt.
- Wiodący producenci: Altera, Xilinx

# *Alternatywy dla DSP: ASIC*

## ASIC (*Application-Specific Integrated Circuit)*

- Specjalizowany układ scalony, którego architektura jest fabrycznie zoptymalizowana pod kątem danego zastosowania.
- Nie ma możliwości modyfikacji, takiej jak w FPGA.
- Duża (optymalna) wydajność w konkretnej aplikacji.
  Są szybsze niż FPGA i DSP, zużywają mniej mocy.
- Bardzo mała elastyczność – tylko wybrane zastosowanie.
- Duże koszty projektowania i produkcji.
- Jeżeli zostanie wykryty błąd w algorytmie – zwykle nie ma możliwości jego naprawienia. W DSP wystarczy zmienić oprogramowanie, w ASIC się nie da.

# *Alternatywy dla DSP: GPU*

## GPU (*Graphic Processing Unit*)

- Układ zoptymalizowany do tworzenia grafiki 3D.
- Procesor równoległy – wiele (kilkaset) jednostek obliczeniowych, wykonujących jednocześnie te same operacje na różnych danych (SIMD).
- Z tego względu, GPU są wykorzystywane w przetwarzaniu równoległym, np. obrazu lub sieci neuronowych.
- Duże zużycie energii, wysoki koszt układu i eksploatacji.
- Duży narzut obliczeniowy przetwarzania.
- Przykład zastosowania: systemy samochodowe, pojazdy autonomiczne (analiza otoczenia).
- Przykład GPU: NVidia Jetson TX2, Jetson Nano.

# *SoC*

## SoC – *System on a Chip*

- Układ elektroniczny integrujący wiele komponentów w jednej „kostce”:
  - mikroprocesor (CPU)
  - pamięć (różnego typu)
  - interfejsy wejściowe/wyjściowe, w tym sieciowe
  - DSP (niektóre implementacje)
  - GPU (niektóre implementacje)
  - dodatkowe koprocesory (zaawansowane implementacje)
- Praktycznie pełny mikrokomputer w jednym układzie.
- Układy typu SoC to obecny trend na rynku technologicznym, gł. w urządzeniach mobilnych i w systemach wbudowanych.