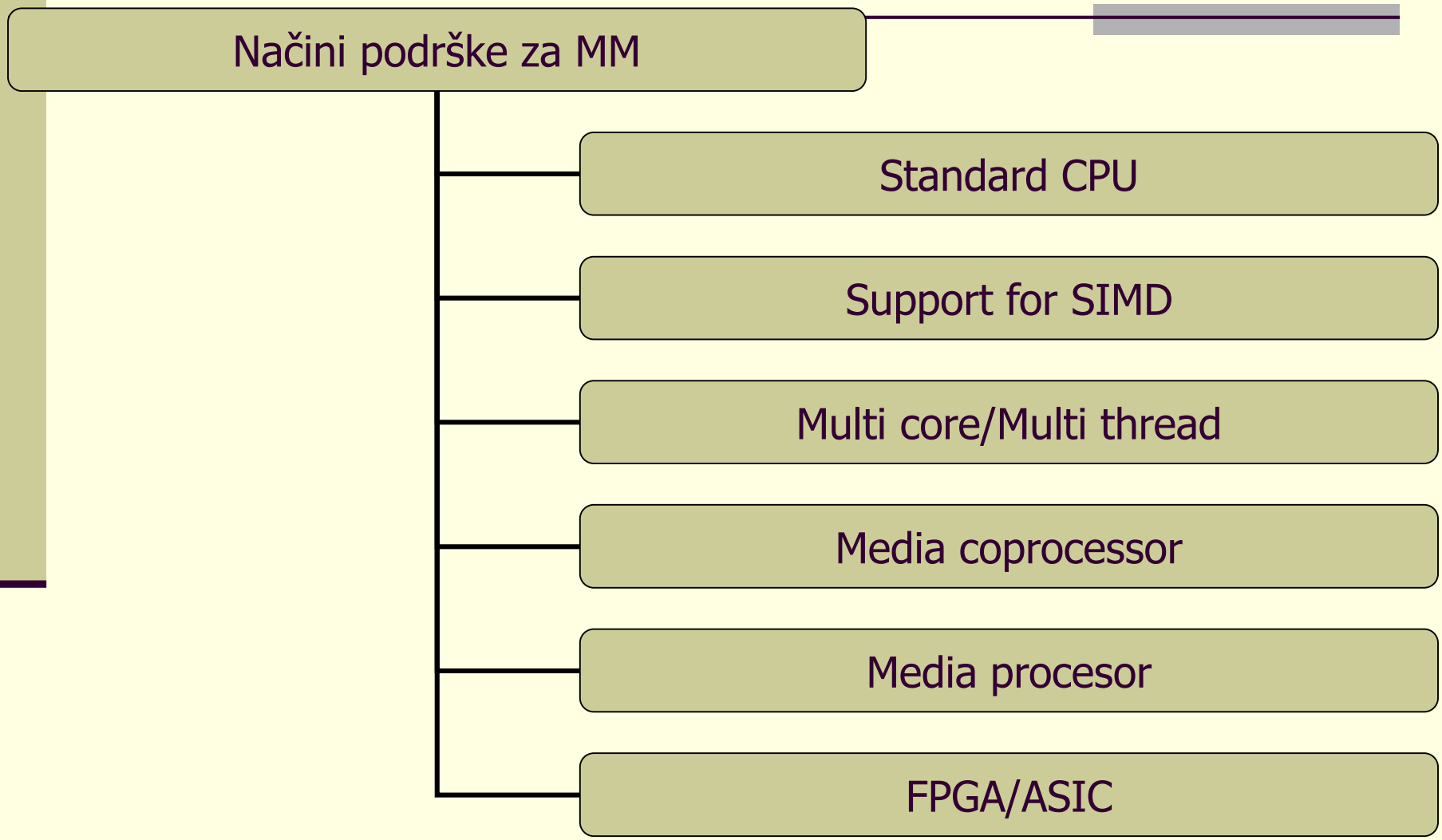
Osnovni načini CPU podrške za MM



Standardni CPU

- Ako se prisjetimo Arhitekture računala onda znamo da obični procesori mogu izvesti jednu ALU operaciju po periodu
- Operacija je širine koju ima ALU

- Kako bi ubrzali izvođenje moramo mnogo pažnje posvetiti dohvatu podataka te optimizacijama petlji

Standardni CPU

- Za DSP operacije (npr DCT) izuzetno je korisno ako procesor ima sklopovski izvedeno množenje i pripadnu naredbu

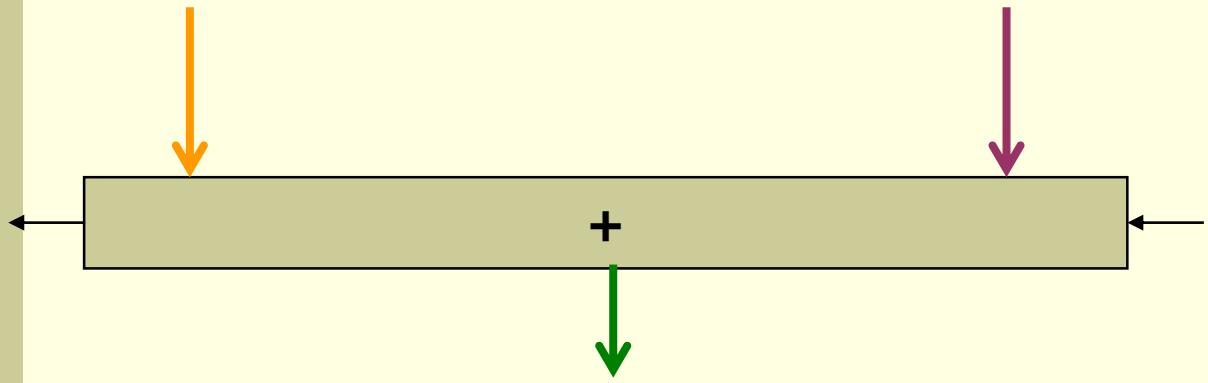
- Dodatna značajna prednost ako postoji naredba množenja sa zbrajanjem (Multiply Accumulate)
  - Kod primjera računanja DCT, DFT i slično imamo većinu “leptir” operacija kod kojih je MLA osnovna karika

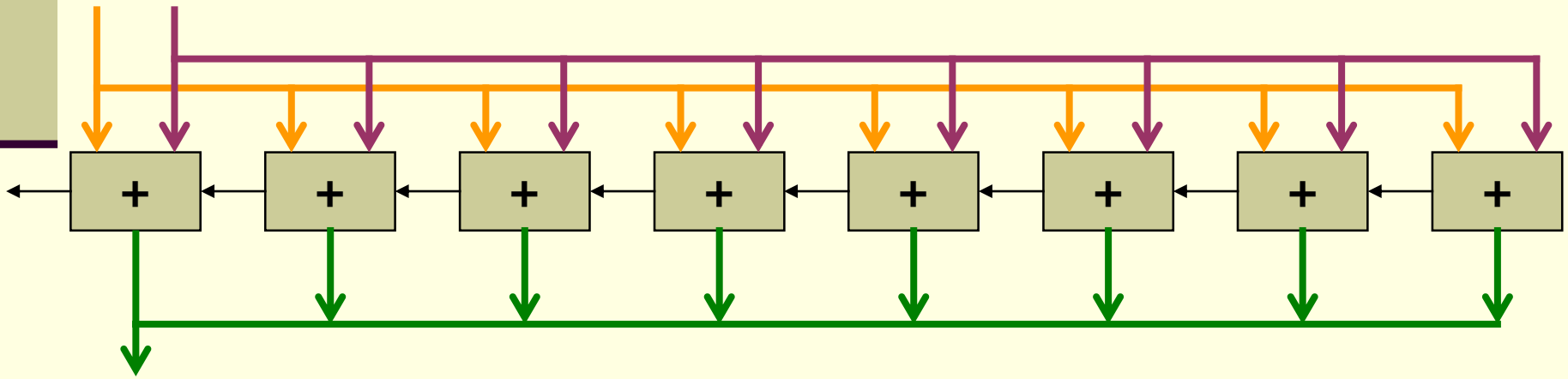
Podrška za SIMD

- SIMD (Single Instruction Multiple Data)
  - Arhitektura puta podataka u procesoru koja omogućuje obradu više podataka (u načelu manje preciznosti) sa jednom naredbom

- Osnovna ideja:
  - Ako imamo npr 64 bitovnu ALU onda je potpuno neefikasno s njom obrađivati 8 bitovne podatke
  - Reorganizirati ALU na način da se može “podijeliti”

ALU – obična i SIMD





MM proširenja

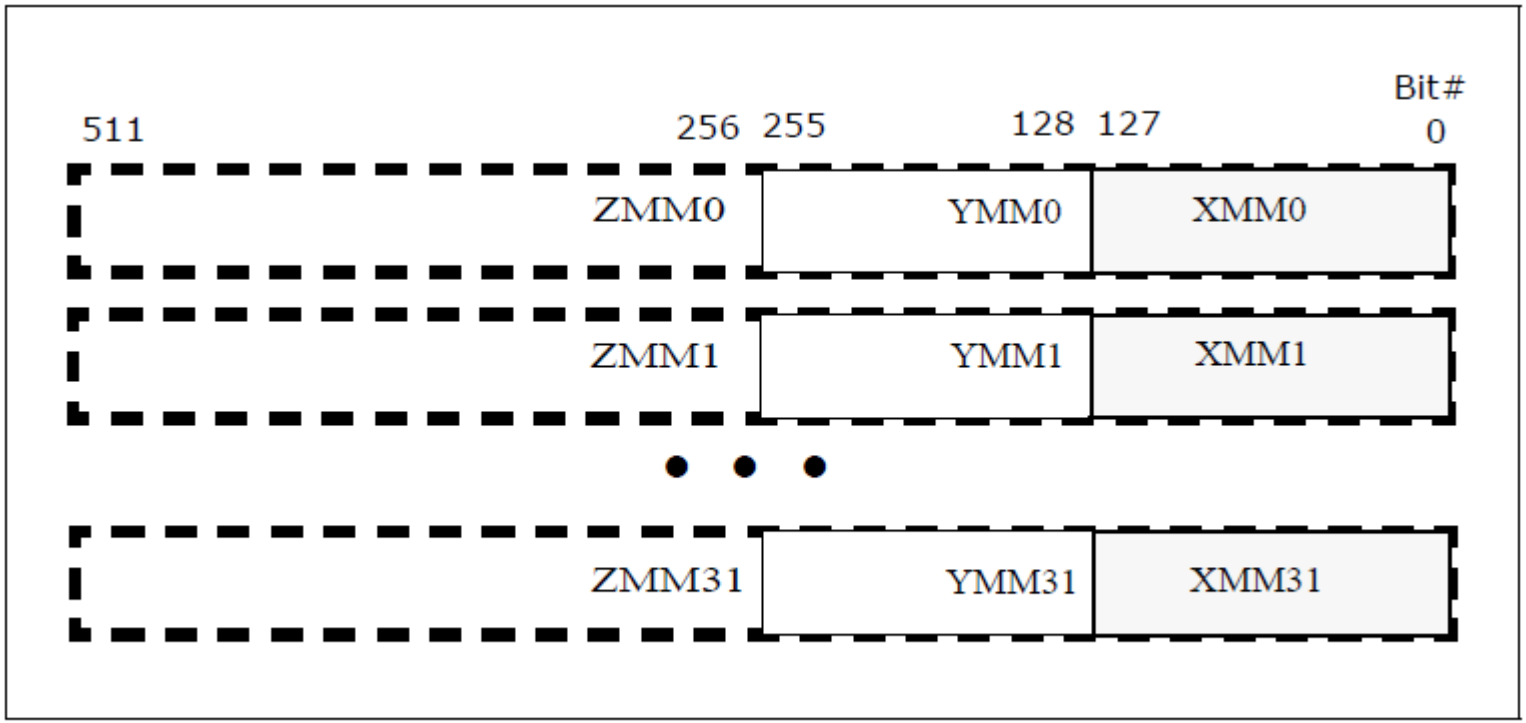
- Ovo (osnovno) načelo služi kako bi se obrada multimedijskih algoritama značajno ubrzala

- Primjeri:
  - Prvi: VIS, PA-RISC
  - Stari: MMX, 3DNow!
  - Ne tako stari: SSE4
  - Novi: AVX, AVX2, NEON (ARM)
  - Najnoviji: Intel AVX-512, ARM SVE

- AVX-512 consists of multiple extensions not all meant to be supported by all processors implementing them. The instruction set consists of the following:
- AVX-512 Foundation – adds several new instructions and expands most 32-bit and 64-bit floating point SSE-SSE4.1 and AVX/AVX2 instructions with EVEX coding scheme to support the 512-bit registers, operation masks, parameter broadcasting, and embedded rounding and exception control
- AVX-512 Conflict Detection Instructions (CDI) – efficient conflict detection to allow more loops to be vectorized, supported by Knights Landing[1]
- AVX-512 Exponential and Reciprocal Instructions (ERI) – exponential and reciprocal operations designed to help implement transcendental operations, supported by Knights Landing[1]
- AVX-512 Prefetch Instructions (PFI) – new prefetch capabilities, supported by Knights Landing[1]
- AVX-512 Vector Length Extensions (VL) – extends most AVX-512 operations to also operate on XMM (128-bit) and YMM (256-bit) registers (including XMM16-XMM31 and YMM16-YMM31 in x86-64 mode)[21]
- AVX-512 Byte and Word Instructions (BW) – extends AVX-512 to cover 8-bit and 16-bit integer operations[21]
- AVX-512 Doubleword and Quadword Instructions (DQ) – enhanced 32-bit and 64-bit integer operations[21]
- AVX-512 Integer Fused Multiply Add (IFMA) - fused multiply add for 52-bit integers.[22]:746
- AVX-512 Vector Byte Manipulation Instructions (VBMI) adds vector byte permutation instructions which are not present in AVX-512BW.

- Only the core extension AVX-512F (AVX-512 Foundation) is required by all implementations; desktop processors will additionally support CDI, VL, and BW/DQ, while computing coprocessors will support CDI, ERI and PFI.

AVX-512



512-Bit Wide Vectors and SIMD Register Set

Primjer: VDBPSADBW

VDBPSADBW—Double Block Packed Sum-Absolute-Differences (SAD) on Unsigned Bytes

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.NDS.128.66.0F3A.W0 42 /r ib VDBPSADBW xmm1 {k1}{z}, xmm2, xmm3/m128, imm8	FVM	V/V	AVX512VL AVX512BW	Compute packed SAD word results of unsigned bytes in dword block from xmm2 with unsigned bytes of dword blocks transformed from xmm3/m128 using the shuffle controls in imm8. Results are written to xmm1 under the writemask k1.
EVEX.NDS.256.66.0F3A.W0 42 /r ib VDBPSADBW ymm1 {k1}{z}, ymm2, ymm3/m256, imm8	FVM	V/V	AVX512VL AVX512BW	Compute packed SAD word results of unsigned bytes in dword block from ymm2 with unsigned bytes of dword blocks transformed from ymm3/m256 using the shuffle controls in imm8. Results are written to ymm1 under the writemask k1.
EVEX.NDS.512.66.0F3A.W0 42 /r ib VDBPSADBW zmm1 {k1}{z}, zmm2, zmm3/m512, imm8	FVM	V/V	AVX512BW	Compute packed SAD word results of unsigned bytes in dword block from zmm2 with unsigned bytes of dword blocks transformed from zmm3/m512 using the shuffle controls in imm8. Results are written to zmm1 under the writemask k1.

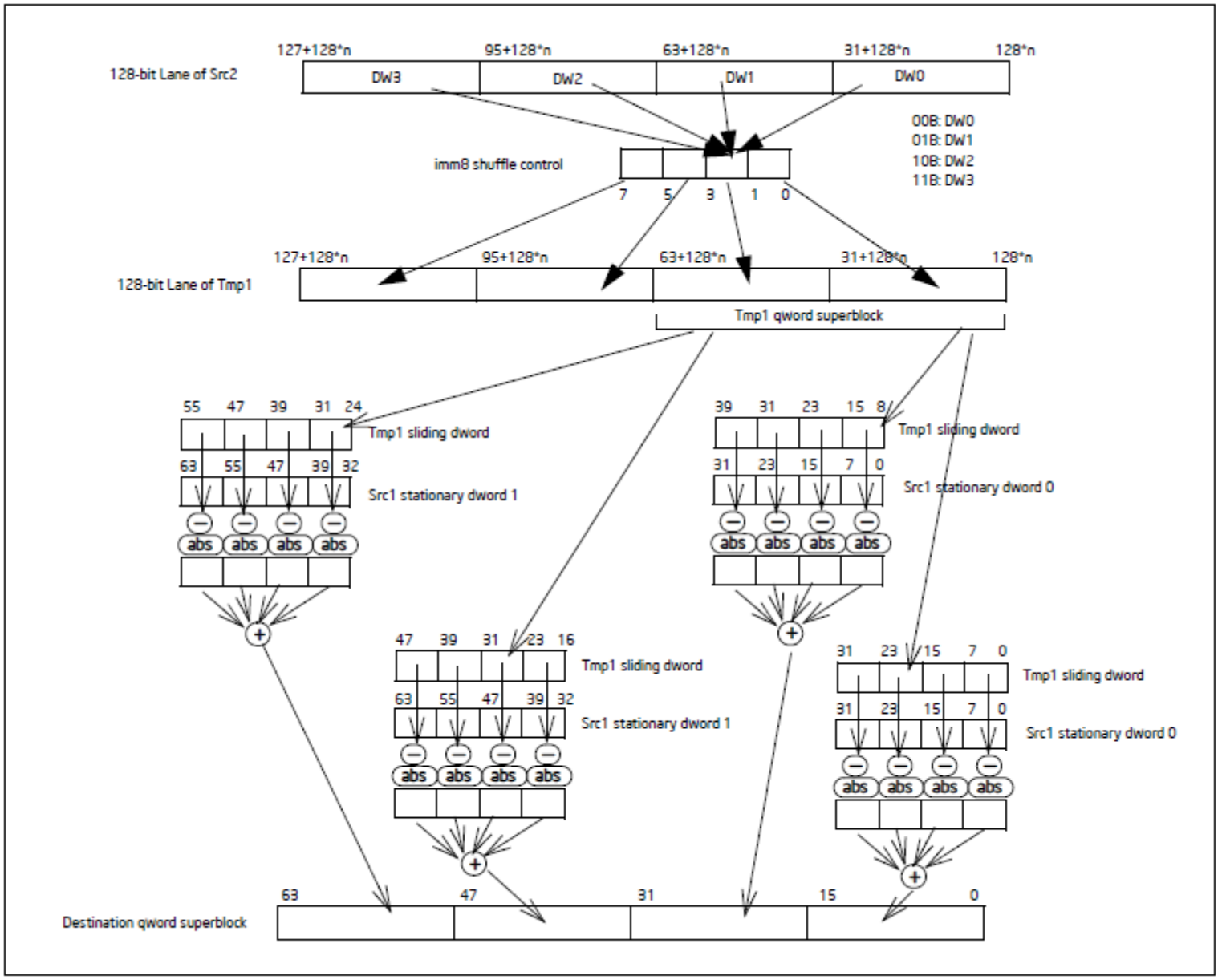
Description

Compute packed SAD (sum of absolute differences) word results of unsigned bytes from two 32-bit dword elements. Packed SAD word results are calculated in multiples of qword superblocks, producing 4 SAD word results in each 64-bit superblock of the destination register.

Within each super block of packed word results, the SAD results from two 32-bit dword elements are calculated as follows:

- The lower two word results are calculated each from the SAD operation between a sliding dword element within a qword superblock from an intermediate vector with a stationary dword element in the corresponding qword superblock of the first source operand. The intermediate vector, see "Tmp1" in Figure 5-18, is constructed from the second source operand the imm8 byte as shuffle control to select dword elements within a 128-bit lane of the second source operand. The two sliding dword elements in a qword superblock of Tmp1 are located at byte offset 0 and 1 within the superblock, respectively. The stationary dword element in the qword superblock from the first source operand is located at byte offset 0.
- The next two word results are calculated each from the SAD operation between a sliding dword element within a qword superblock from the intermediate vector Tmp1 with a second stationary dword element in the corresponding qword superblock of the first source operand. The two sliding dword elements in a qword superblock of Tmp1 are located at byte offset 2and 3 within the superblock, respectively. The stationary dword element in the qword superblock from the first source operand is located at byte offset 4.
- The intermediate vector is constructed in 128-bits lanes. Within each 128-bit lane, each dword element of the intermediate vector is selected by a two-bit field within the imm8 byte on the corresponding 128-bits of the second source operand. The imm8 byte serves as dword shuffle control within each 128-bit lanes of the intermediate vector and the second source operand, similarly to PSHUFD.

The first source operand is a ZMM/YMM/XMM register. The second source operand is a ZMM/YMM/XMM register, or a 512/256/128-bit memory location. The destination operand is conditionally updated based on writemask k1 at 16-bit word granularity.



64-bit Super Block of SAD Operation in VDBPSADBW

Operation

VDBPSADBW (EVEX encoded versions)

(KL, VL) = (8, 128), (16, 256), (32, 512)

Selection of quadruplets:

```
FOR I = 0 to VL step 128
    TMP1[I+31:I] ← select (SRC2[I+127: I], imm8[1:0])
    TMP1[I+63: I+32] ← select (SRC2[I+127: I], imm8[3:2])
    TMP1[I+95: I+64] ← select (SRC2[I+127: I], imm8[5:4])
    TMP1[I+127: I+96]← select (SRC2[I+127: I], imm8[7:6])
END FOR
```

SAD of quadruplets:

```
FOR I =0 to VL step 64
    TMP_DEST[I+15:I] ← ABS(SRC1[I+7: I] - TMP1[I+7: I]) +
        ABS(SRC1[I+15: I+8]- TMP1[I+15: I+8]) +
        ABS(SRC1[I+23: I+16]- TMP1[I+23: I+16]) +
        ABS(SRC1[I+31: I+24]- TMP1[I+31: I+24])

    TMP_DEST[I+31: I+16] ←ABS(SRC1[I+7: I] - TMP1[I+15: I+8]) +
        ABS(SRC1[I+15: I+8]- TMP1[I+23: I+16]) +
        ABS(SRC1[I+23: I+16]- TMP1[I+31: I+24]) +
        ABS(SRC1[I+31: I+24]- TMP1[I+39: I+32])
    TMP_DEST[I+47: I+32] ←ABS(SRC1[I+39: I+32] - TMP1[I+23: I+16]) +
        ABS(SRC1[I+47: I+40]- TMP1[I+31: I+24]) +
        ABS(SRC1[I+55: I+48]- TMP1[I+39: I+32]) +
        ABS(SRC1[I+63: I+56]- TMP1[I+47: I+40])

    TMP_DEST[I+63: I+48] ←ABS(SRC1[I+39: I+32] - TMP1[I+31: I+24]) +
        ABS(SRC1[I+47: I+40] - TMP1[I+39: I+32]) +
        ABS(SRC1[I+55: I+48] - TMP1[I+47: I+40]) +
        ABS(SRC1[I+63: I+56] - TMP1[I+55: I+48])
ENDFOR

FOR j ← 0 TO KL-1
    i ←  j * 16
    IF k1[j] OR *no writemask*
        THEN DEST[i+15:i] ← TMP_DEST[i+15:i]
        ELSE
            IF *merging-masking*                    ; merging-masking
                THEN *DEST[i+15:i] remains unchanged*
                ELSE                                ; zeroing-masking
                    DEST[i+15:i] ← 0
            FI
    FI;
ENDFOR
DEST[MAX_VL-1:VL] ← 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VDBPSADBW __m512i _mm512_dbsad_epu8(__m512i a, __m512i b);
VDBPSADBW __m512i _mm512_mask_dbsad_epu8(__m512i s, __mmask32 m, __m512i a, __m512i b);
VDBPSADBW __m512i _mm512_maskz_dbsad_epu8(__mmask32 m, __m512i a, __m512i b);
VDBPSADBW __m256i _mm256_dbsad_epu8(__m256i a, __m256i b);
VDBPSADBW __m256i _mm256_mask_dbsad_epu8(__m256i s, __mmask16 m, __m256i a, __m256i b);
VDBPSADBW __m256i _mm256_maskz_dbsad_epu8(__mmask16 m, __m256i a, __m256i b);
VDBPSADBW __m128i _mm_dbsad_epu8(__m128i a, __m128i b);
VDBPSADBW __m128i _mm_mask_dbsad_epu8(__m128i s, __mmask8 m, __m128i a, __m128i b);
VDBPSADBW __m128i _mm_maskz_dbsad_epu8(__mmask8 m, __m128i a, __m128i b);

Multicore/multithread

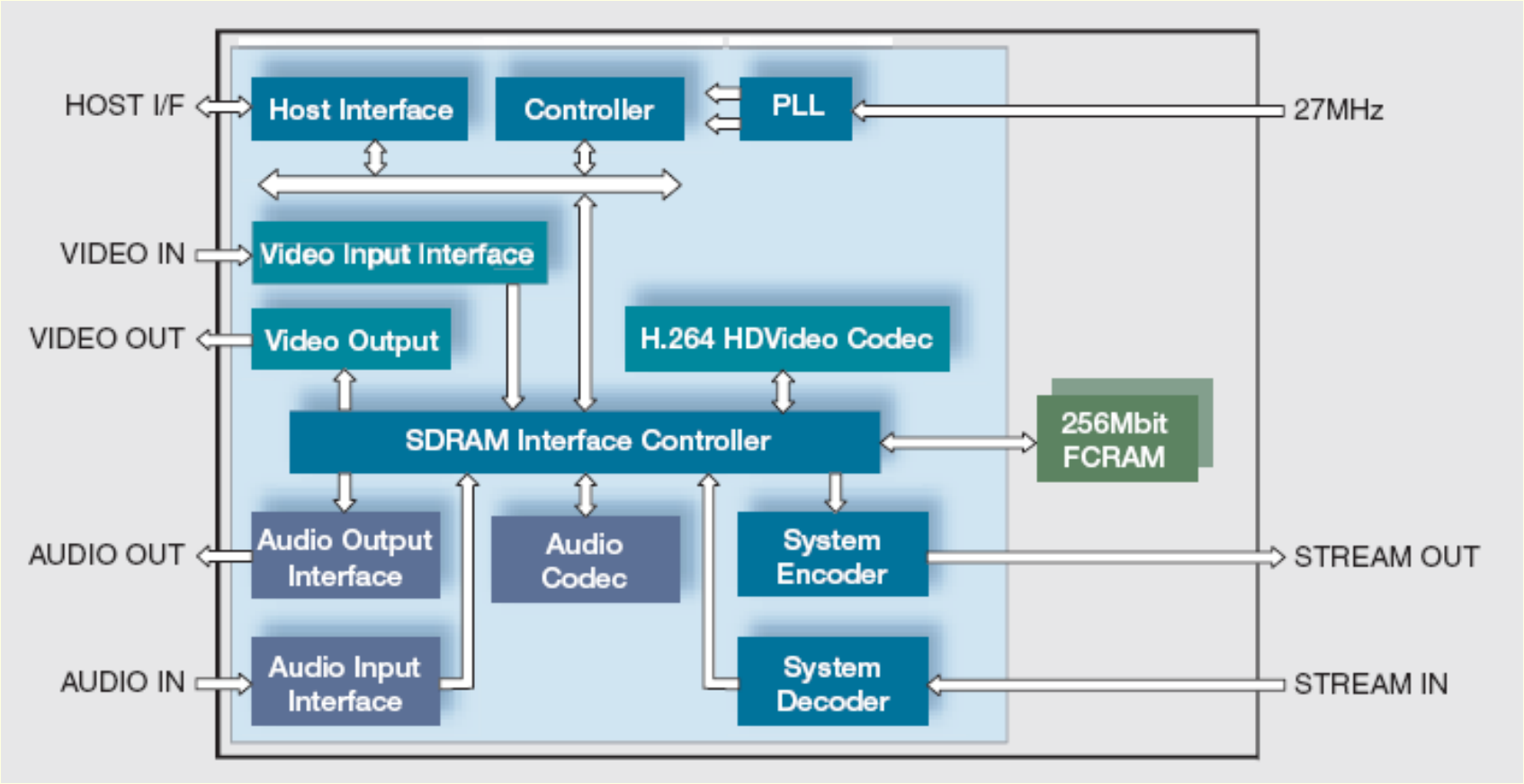
- U prethodnim predavanjima:
  - jedan CPU
- Danas: vrlo zahtjevni zadaci (MM je jedan od njih) mogu se na još jedan način ubrzati koristeći više dretvi (thread) ili više jezgri (core)
- Bez ulaženja u teoriju, jasno je da se djelomičnom paralelizacijom procesa može postići veća brzina

Što je medijski koprocesor

- Standardni procesori nisu pogodni za mnoge primjene
- Iz dosadašnjeg izlaganja znamo kakva arhitektura je pogodna za visokozahtjevne algoritme i puno podataka
- Moguće rješenje:
  - Zahtjevni dijelovi algoritma obrađuju se u posebnom procesoru koji se stavlja u sustav i služi samo toj funkciji
  - -> Medijski koprocesor

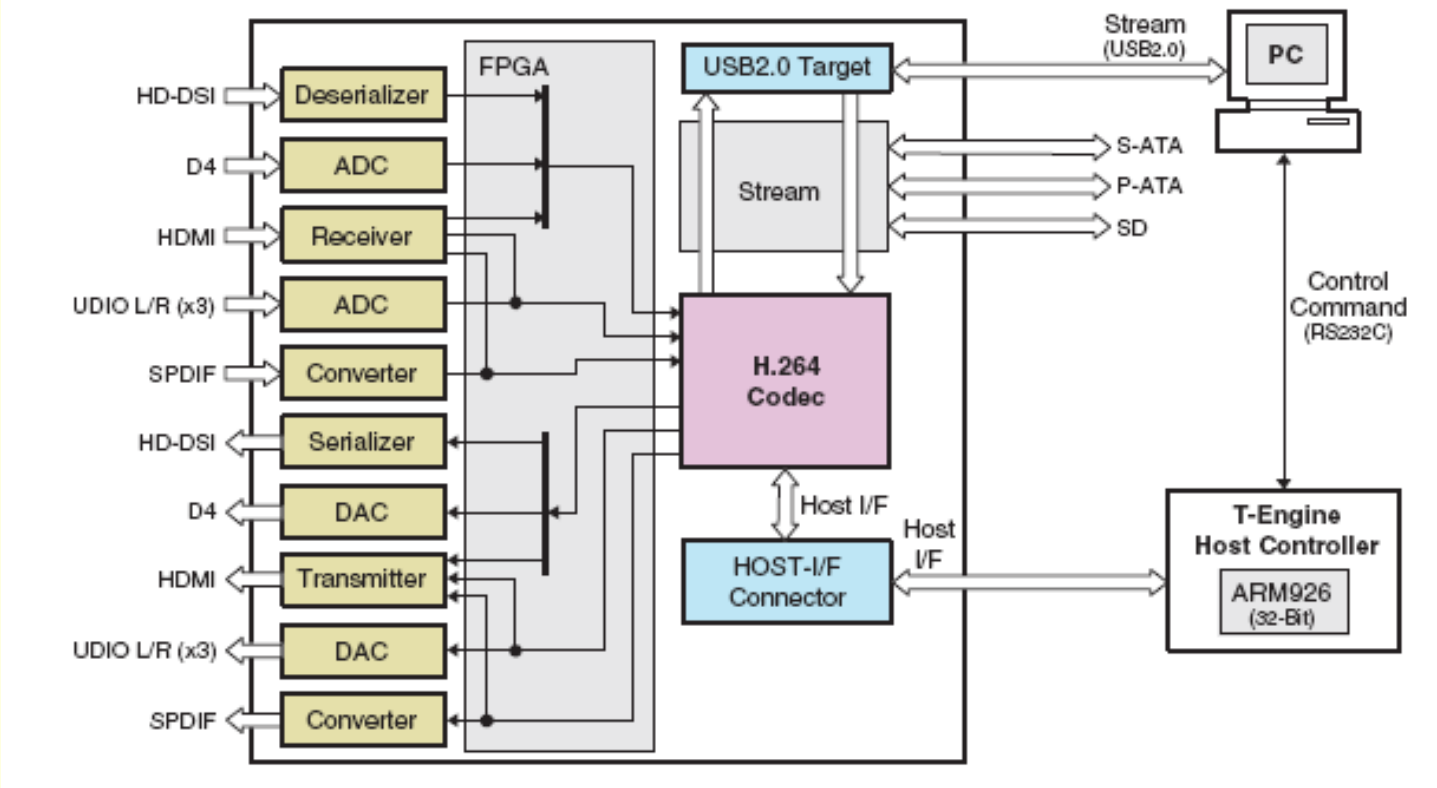
Primjeri: Fujitsu

- MB86H50: H.264 Video Processing IC

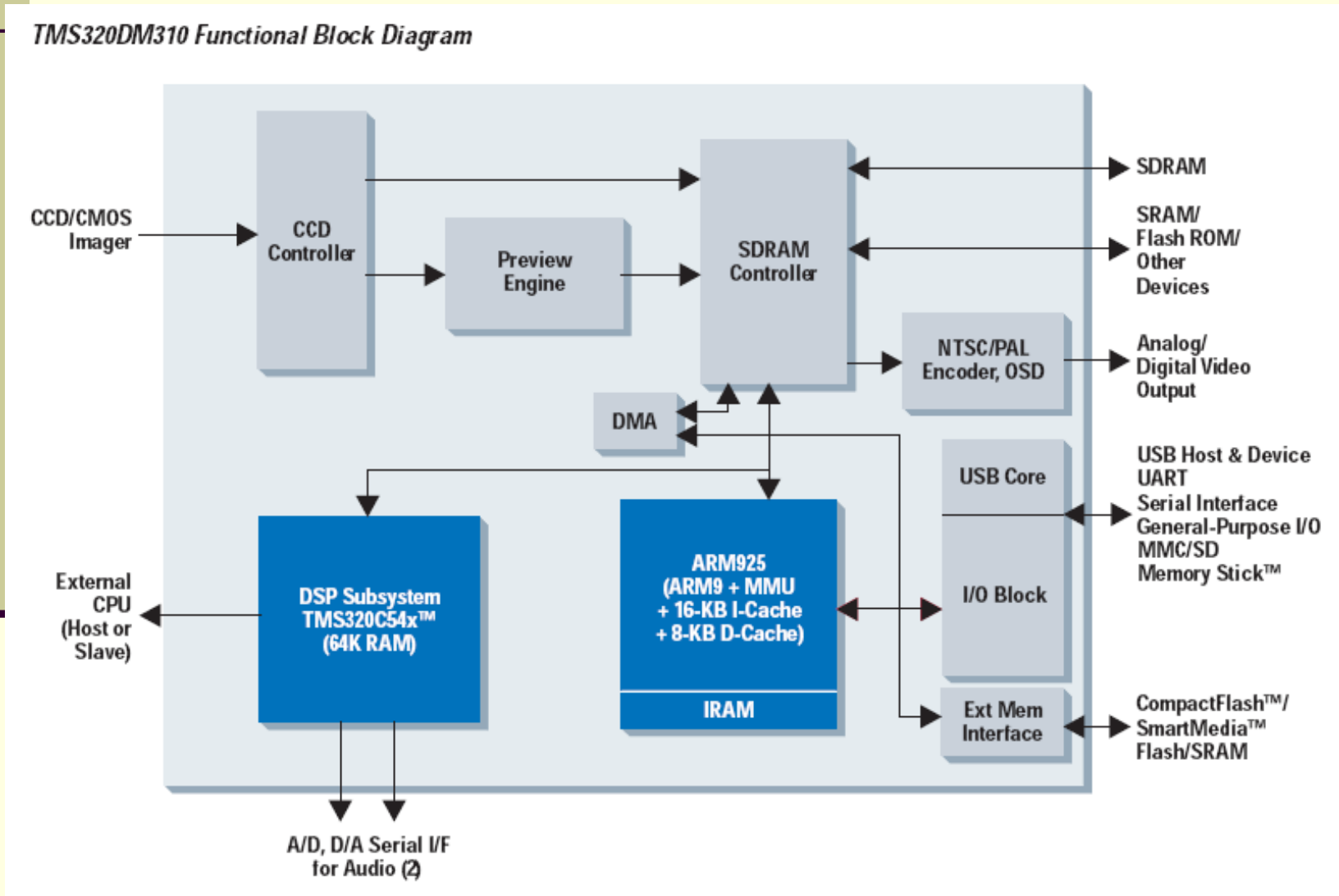


Fujitsu

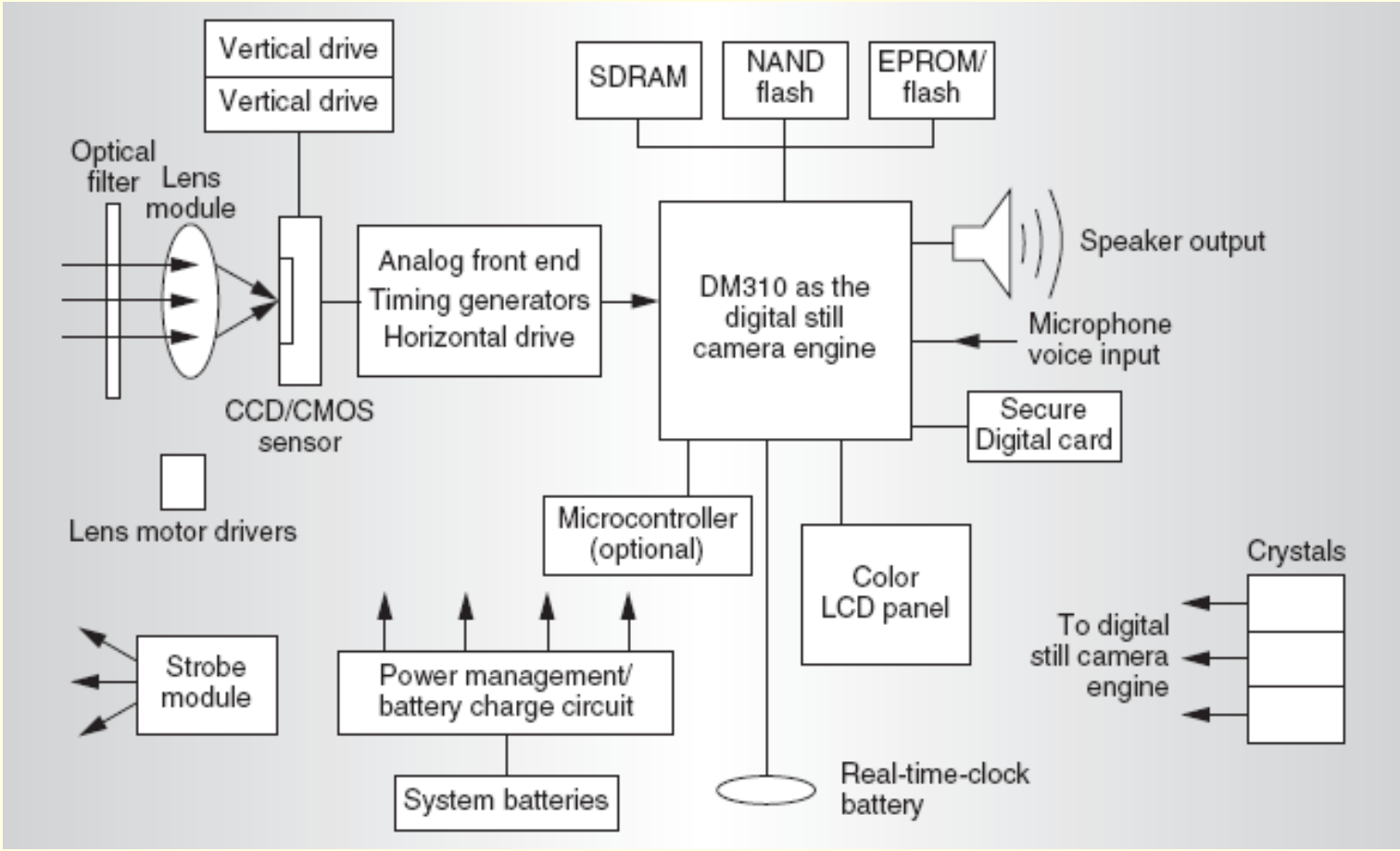
- Primjer sustava



Primjer: TI

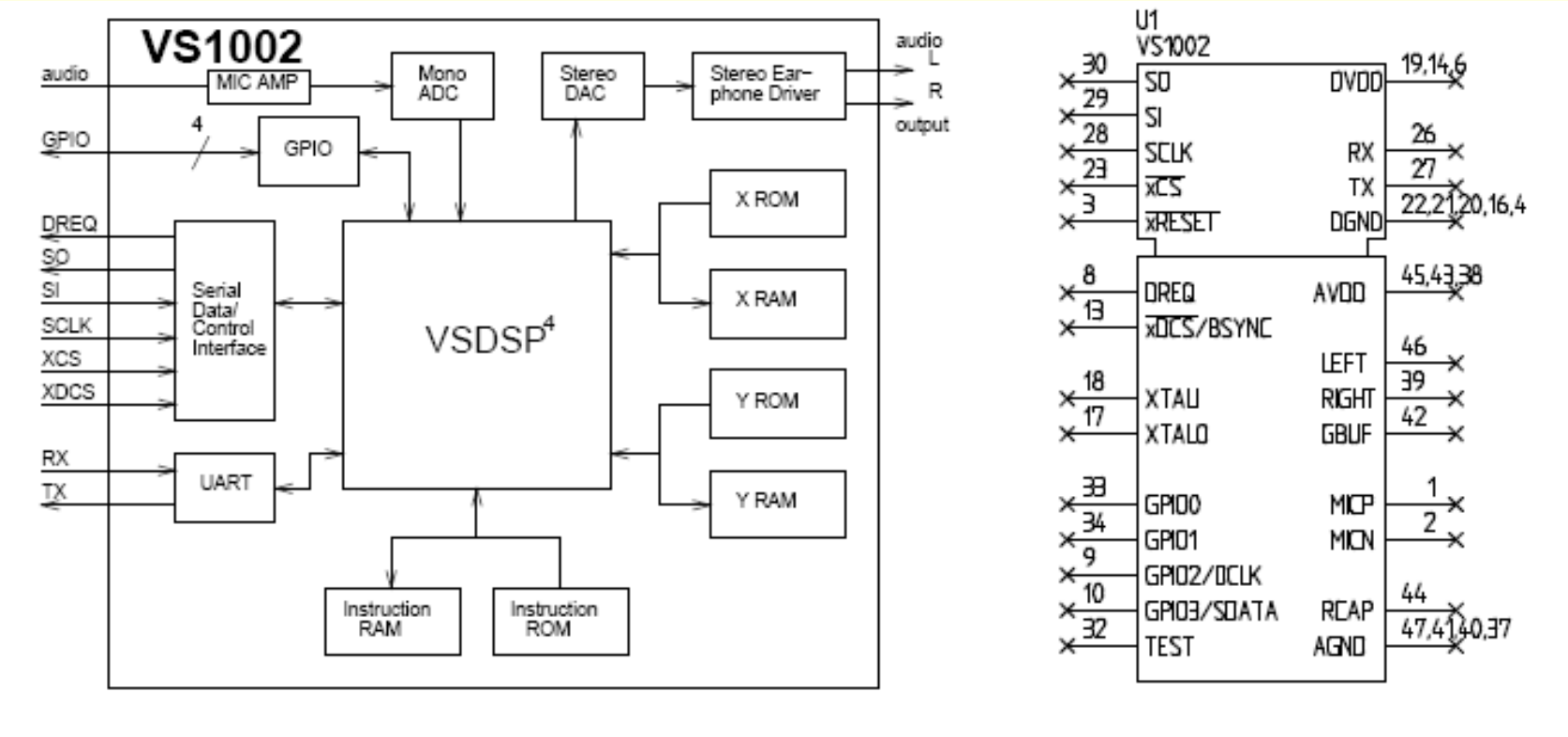

TMS320DM310 Functional Block Diagram
CCD/CMOS Imager
CCD Controller
Preview Engine
SDRAM Controller
SDRAM
SRAM/ Flash ROM/ Other Devices
NTSC/PAL Encoder, OSD
Analog/ Digital Video Output
DMA
USB Core
USB Host & Device
UART
Serial Interface
General-Purpose I/O
MMC/SD
Memory Stick™
External CPU (Host or Slave)
DSP Subsystem TMS320C54x™ (64K RAM)
ARM925 (ARM9 + MMU + 16-KB I-Cache + 8-KB D-Cache)
IRAM
I/O Block
Ext Mem Interface
CompactFlash™/ SmartMedia™ Flash/SRAM
A/D, D/A Serial I/F for Audio (2)

TI


Vertical drive
Vertical drive
SDRAM
NAND flash
EPROM/ flash
Optical filter
Lens module
Analog front end
Timing generators
Horizontal drive
DM310 as the digital still camera engine
Speaker output
Microphone voice input
Secure Digital card
CCD/CMOS sensor
Lens motor drivers
Microcontroller (optional)
Color LCD panel
Crystals
To digital still camera engine
Strobe module
Power management/ battery charge circuit
System batteries
Real-time-clock battery

Medijski koprocesor ne mora biti samo u high-end sustavima

- :VLSI Solution VS 1002: MP3 decoder IC



VS1002

- Opći procesor:
  - **PIC18LF45x** jednostavan CPU, radi na 20MHz, upravlja radom sustava, upravlja s LCD, tipkama, FLASH karticom,...
  - **VS1002D** MP3 koprocesor, radi samo dekodiranje

Medijski koprocesor

- U osnovi : DSP sa mnoštvom periferija
- Jeftiniji od procesora opće namjene
- Performanse prilagođene aplikaciji (manje od GPP)
- Manja potrošnja
- Programabilan !! (mogućnost poboljšanja i dodavanja aplikacija)
- Predviđen za porodicu algoritama
- 32-bitovni CPU (npr. ARM) obično dobar za ostale poslove (mreža, user i/f, ...)

Medijski procesor

- U dosadašnjim primjerima procesor je bio prvenstveno projektiran za “opće” zadatke
- Za obradu multimedije koristile su se “poboljšanja”

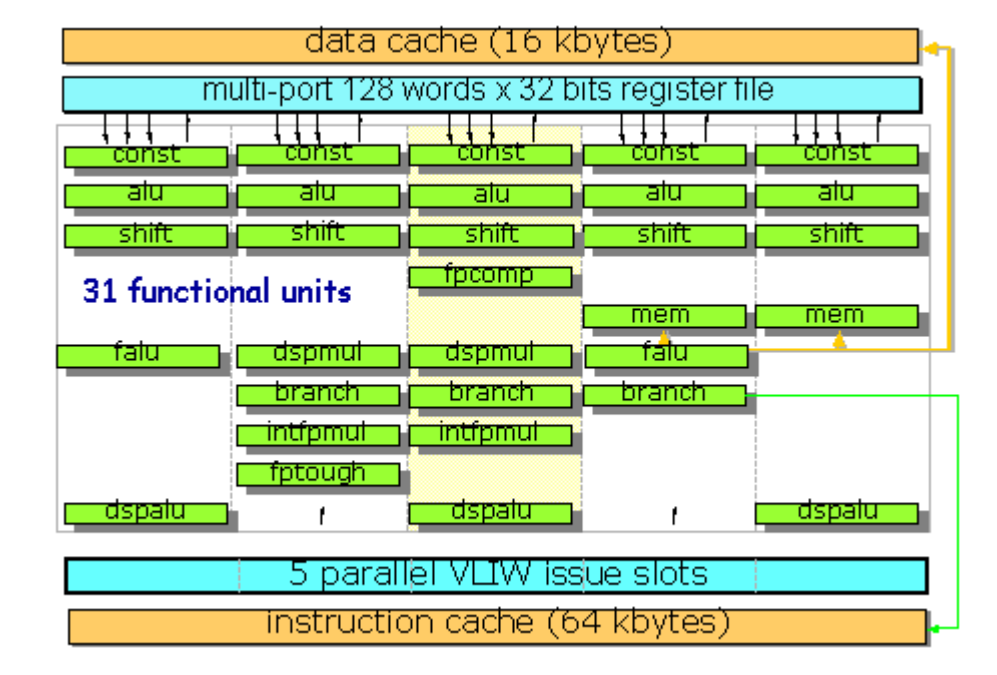
- Novi pristup: procesor se projektira PRVENSTVENO za obradu multimedijskih podataka, a ostale stvari može obrađivati ali tek sporedno
  - -> Medijski procesor

Medijski procesor

- Procesor se projektira sa ciljem visokih performansi za određen skup algoritama
- No to je još uvijek PROCESOR: može se programirati
  - Prednosti: promjena algoritama, dodavanje funkcionalnosti,....

Primjer: NXP

- Trimedia TM3270 CPU




```
Issue slot
1:      NOP,
2:      IF r34 MUL r87 r54 -> r123,
3:      IF r45 QUADUMIN r3 r67 -> r23,
4:      NOP,
5:      LD32D (4) r22 -> r14;
```

TM

- 31 funkcionalna jedinica
- 5 paralelnih izvedbenih polja
- Very Large Instruction Word (VLIW) arhitektura
- Neke naredbe omogućuju SIMD
- VLIW ima prednosti nad superscalar arhitekturom jer paralelizam ne određuje procesor (scheduling unit on CPU) već programer i compiler tijekom dizajna
  - Procesor jeftiniji i brži

Medijski procesor

- Izuzetno visoke performanse (veće od GPP, medijskih koprocesora)
- Složen postupak programiranja
- Paralelno izvođenje operacija unutar jedne naredbe
- Još uvijek programabilan
- Podrška za opće zadatke mora biti osigurana od dodatnog procesora

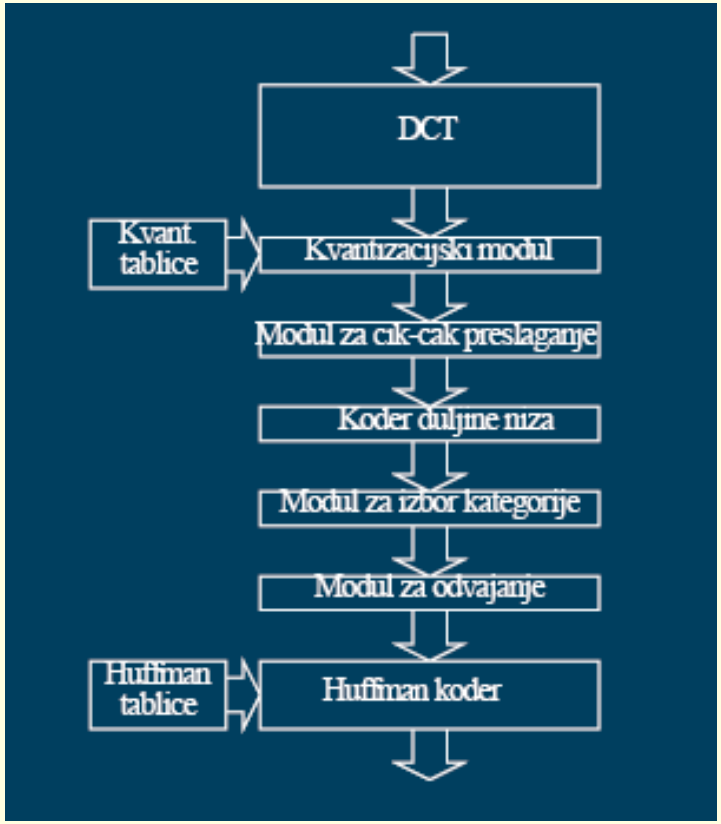
FPGA/ASIC

- Radi što većih performansi a niže cijene krajnja mogućnost je projektirati sklop koji sklopovski izvodi izabran algoritam

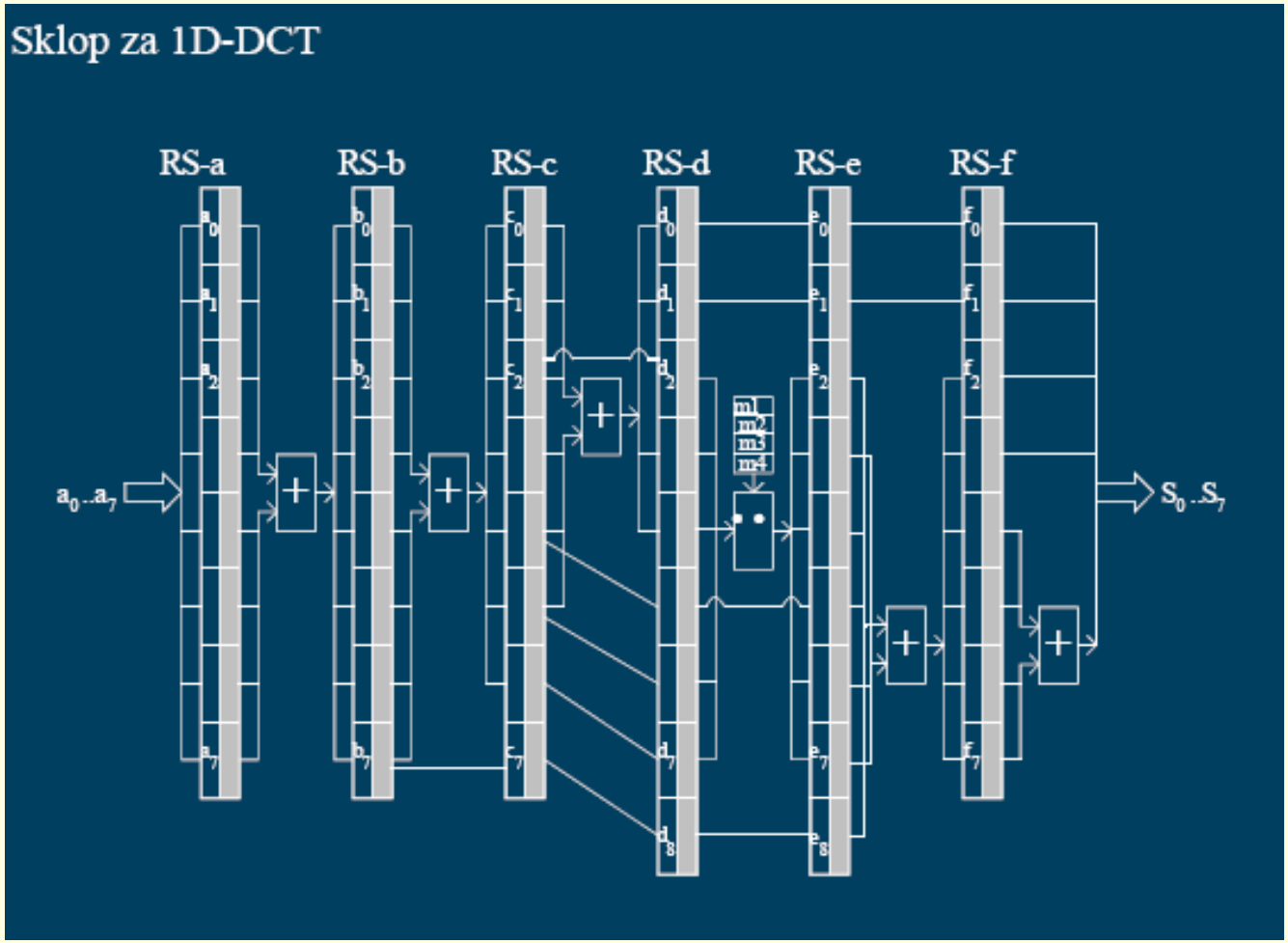
- Dva pristupa
  - FPGA (Field Programmable Gate Array)
  - ASIC (Application Specific IC)

JAGUAR

- HW JPEG encoder



JAGUAR



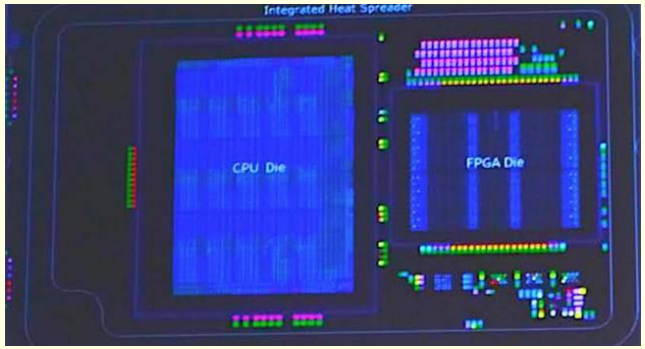
VLSI/ASIC

- Najpogodnije rješenje za zadani algoritam: performanse, potrošnja, cijena,...
- Jednostavni za integraciju i korištenje

- Nemogućnost poboljšanja, nadogradnje
- Samo jedan dobavljač: opasnost

Heterogeni procesori

- Najnoviji pristup poboljšanju arhitekture
- CPU + GPU
  - **Arm + NVIDIA (2020!?)**
- CPU + FPGA
  - **Intel Broadwell Xeon with a built-in FPGA**



- CPU+GPU+FPGA
- CPU+HW Accelerators

MANGO Arhitektura

- Deeply heterogeneous QoS aware architecture