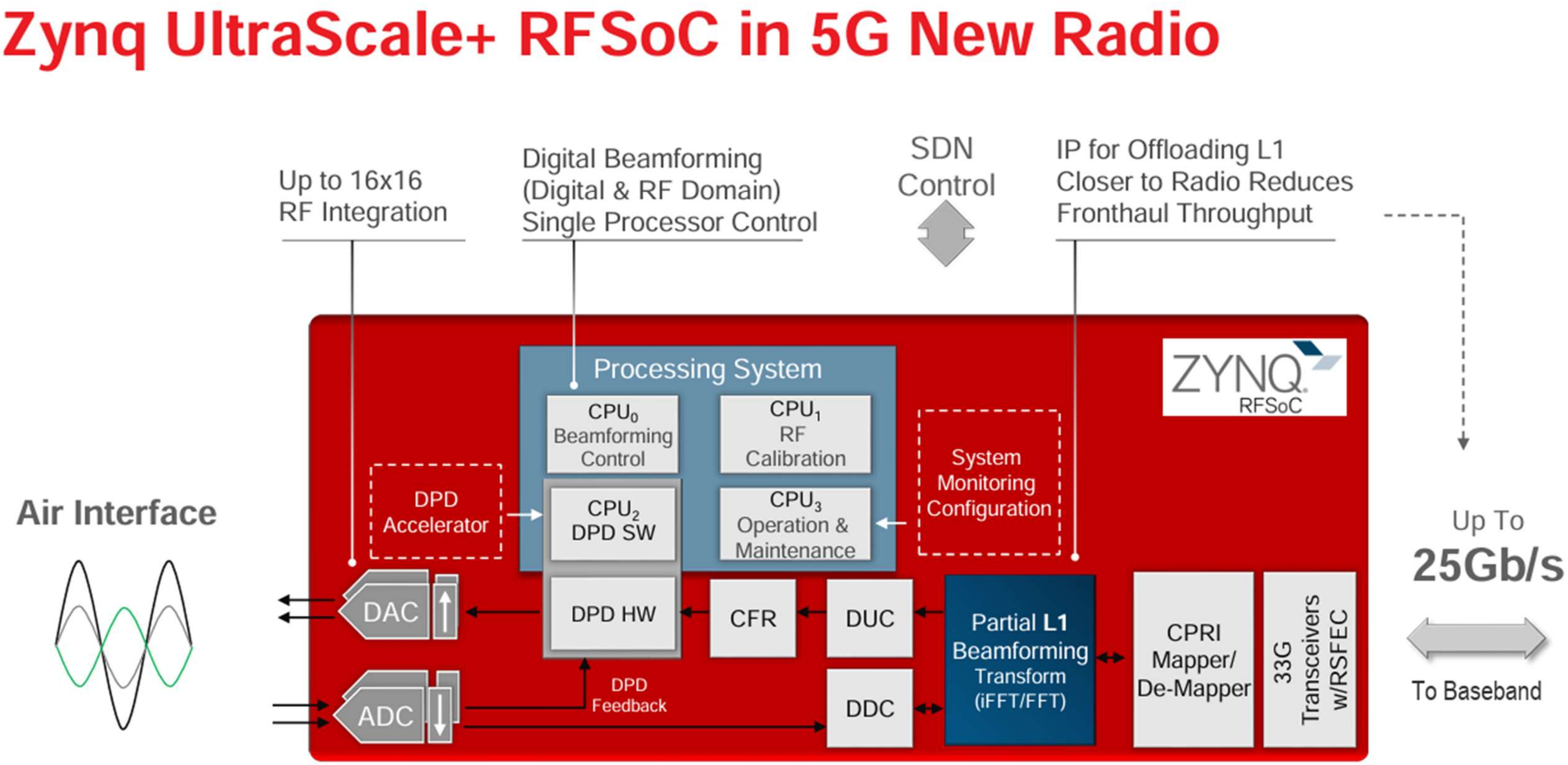
“Modern” Digital Processing Systems : AMD/Xilinx RFSoC type 1



Zynq UltraScale+ RFSoC in 5G New Radio

Up to 16x16 RF Integration

Digital Beamforming (Digital & RF Domain) Single Processor Control

SDN Control

IP for Offloading L1 Closer to Radio Reduces Fronthaul Throughput

Air Interface

Processing System

CPU_0 Beamforming Control

CPU_1 RF Calibration

CPU_2 DPD SW

CPU_3 Operation & Maintenance

DPD Accelerator

System Monitoring Configuration

ZYNQ RFSoC

DAC

ADC

DPD HW

CFR

DUC

DDC

DPD Feedback

Partial L1 Beamforming Transform (iFFT/FFT)

CPRI Mapper/ De-Mapper

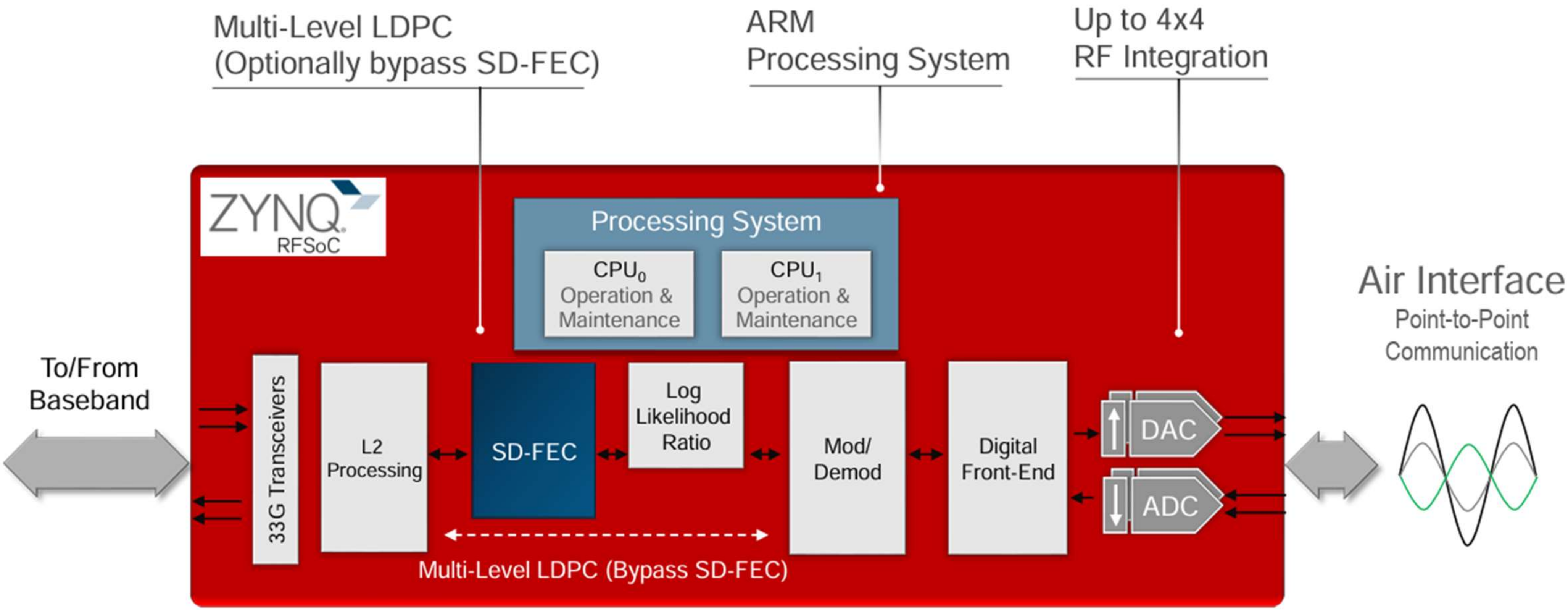
33G Transceivers w/RSFEC

Up To 25Gb/s

To Baseband

“Modern” Digital Processing Systems : AMD/Xilinx RFSoC type 2

Zynq UltraScale+ RFSoC in Wireless Backhaul

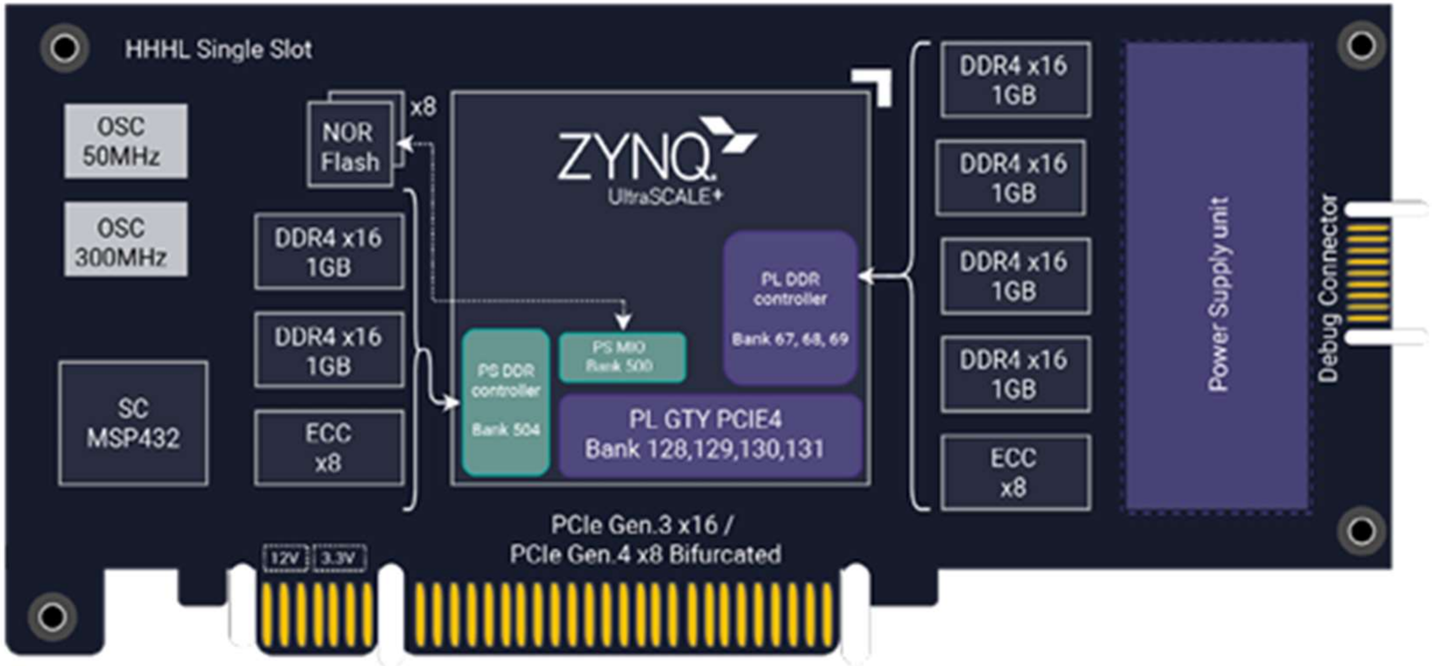


“Modern” Digital Processing Systems : AMD/Xilinx RFSoC type 3 (HW Accelerator)

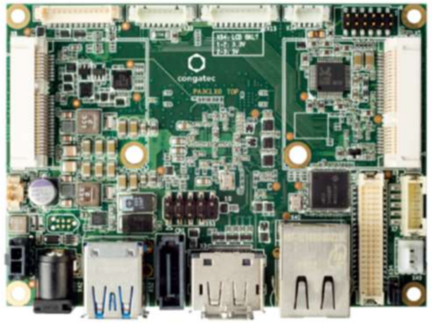
HIGH LEVEL BLOCK DIAGRAM

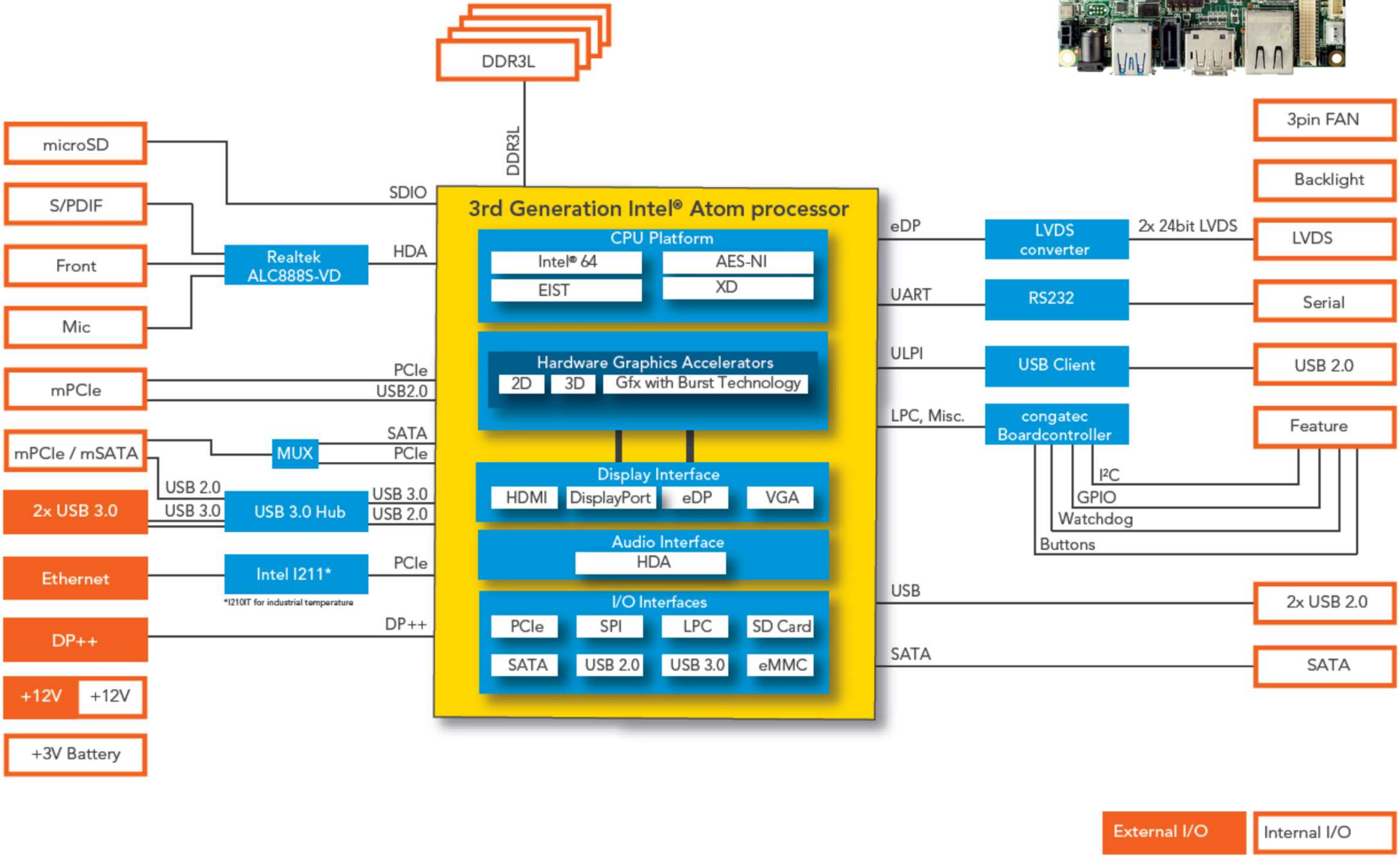
L1 Offload

- LDPC/TURBO codecs
- Polar codecs
- HARQ management
- Codec Surrounding/Wrapper logic
  - CRC logic
  - Rate Matching/De-Matching

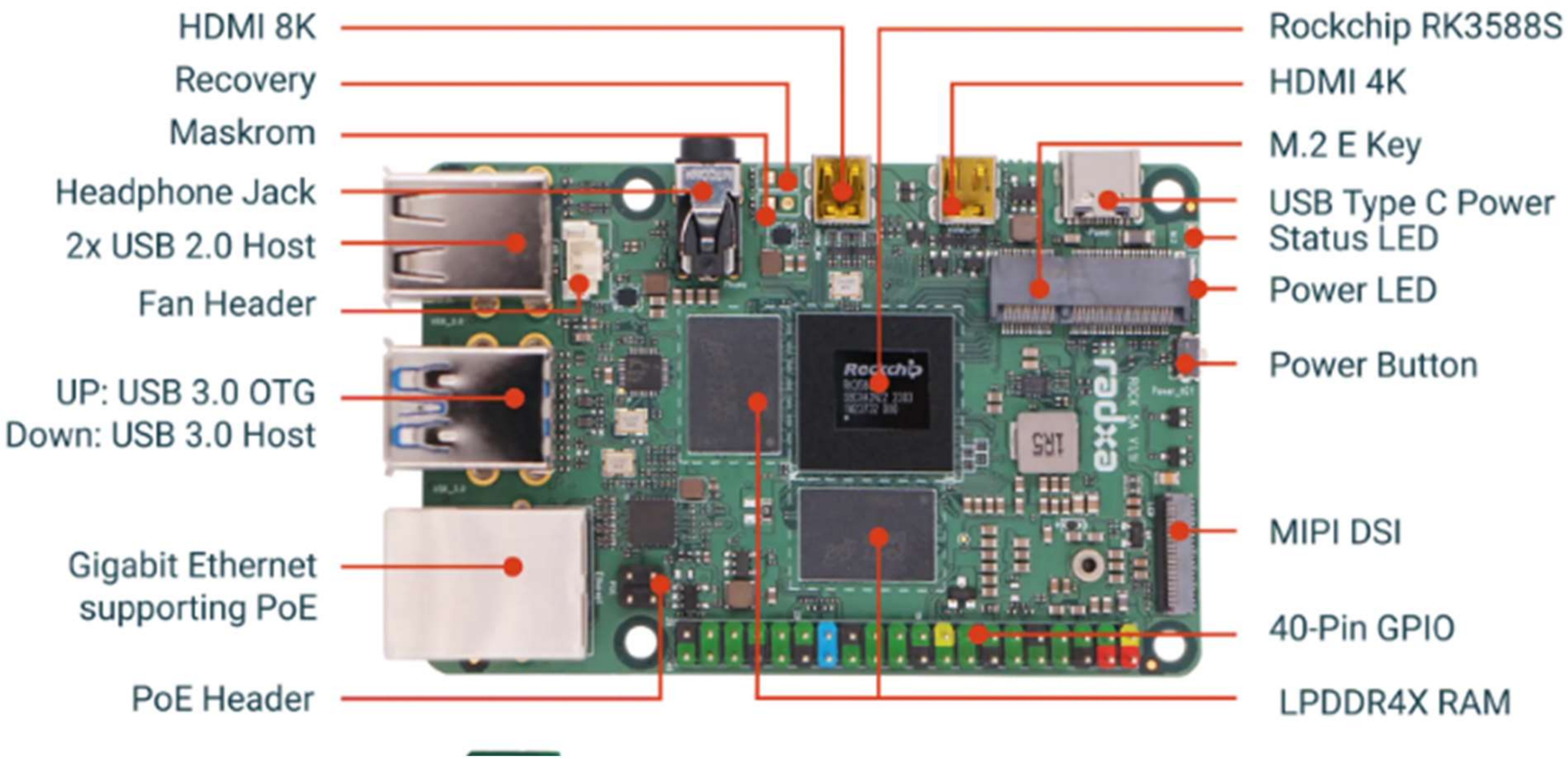


x86-64 solution (Atom quad core)

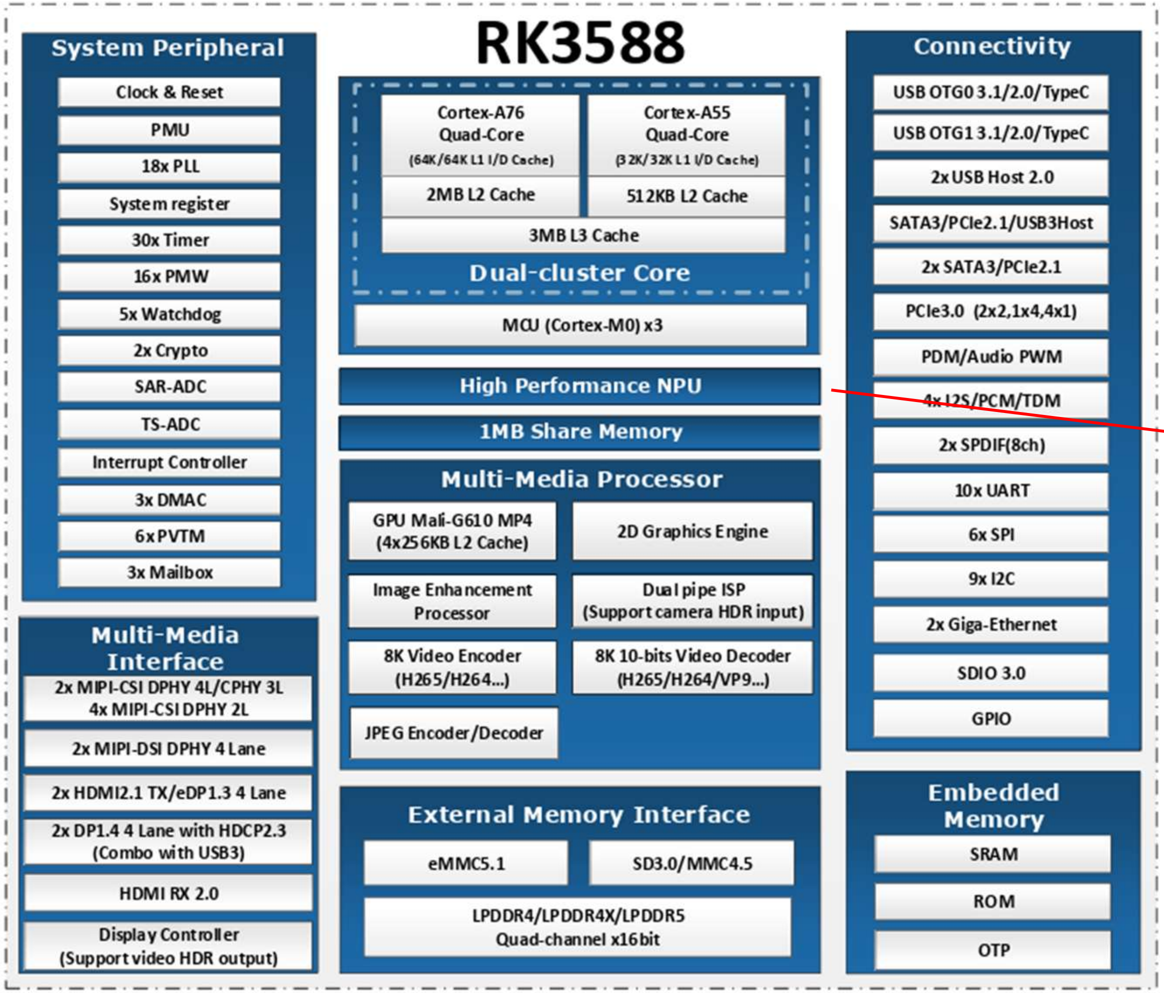




ARM v8 solution : RK5388 (ARM cores, GPU, NPU)



RK3588 internals



RK3588

System Peripheral

Clock & Reset

PMU

18x PLL

System register

30x Timer

16x PMW

5x Watchdog

2x Crypto

SAR-ADC

TS-ADC

Interrupt Controller

3x DMAC

6x PVTM

3x Mailbox

Multi-Media Interface

2x MIPI-CSI DPHY 4L/CPHY 3L
4x MIPI-CSI DPHY 2L

2x MIPI-DSI DPHY 4 Lane

2x HDMI2.1 TX/eDP1.3 4 Lane

2x DP1.4 4 Lane with HDCP2.3
(Combo with USB3)

HDMI RX 2.0

Display Controller
(Support video HDR output)

Cortex-A76
Quad-Core
(64K/64K L1 I/D Cache)

Cortex-A55
Quad-Core
(32K/32K L1 I/D Cache)

2MB L2 Cache

512KB L2 Cache

3MB L3 Cache

Dual-cluster Core

MCU (Cortex-M0) x3

High Performance NPU

1MB Share Memory

Multi-Media Processor

GPU Mali-G610 MP4
(4x256KB L2 Cache)

2D Graphics Engine

Image Enhancement
Processor

Dual pipe ISP
(Support camera HDR input)

8K Video Encoder
(H265/H264...)

8K 10-bits Video Decoder
(H265/H264/VP9...)

JPEG Encoder/Decoder

External Memory Interface

eMMC5.1

SD3.0/MMC4.5

LPDDR4/LPDDR4X/LPDDR5
Quad-channel x16bit

Connectivity

USB OTG0 3.1/2.0/TypeC

USB OTG1 3.1/2.0/TypeC

2x USB Host 2.0

SATA3/PCIe2.1/USB3Host

2x SATA3/PCIe2.1

PCIe3.0 (2x2,1x4,4x1)

PDM/Audio PWM

4x I2S/PCM/TDM

2x SPDIF(8ch)

10x UART

6x SPI

9x I2C

2x Giga-Ethernet

SDIO 3.0

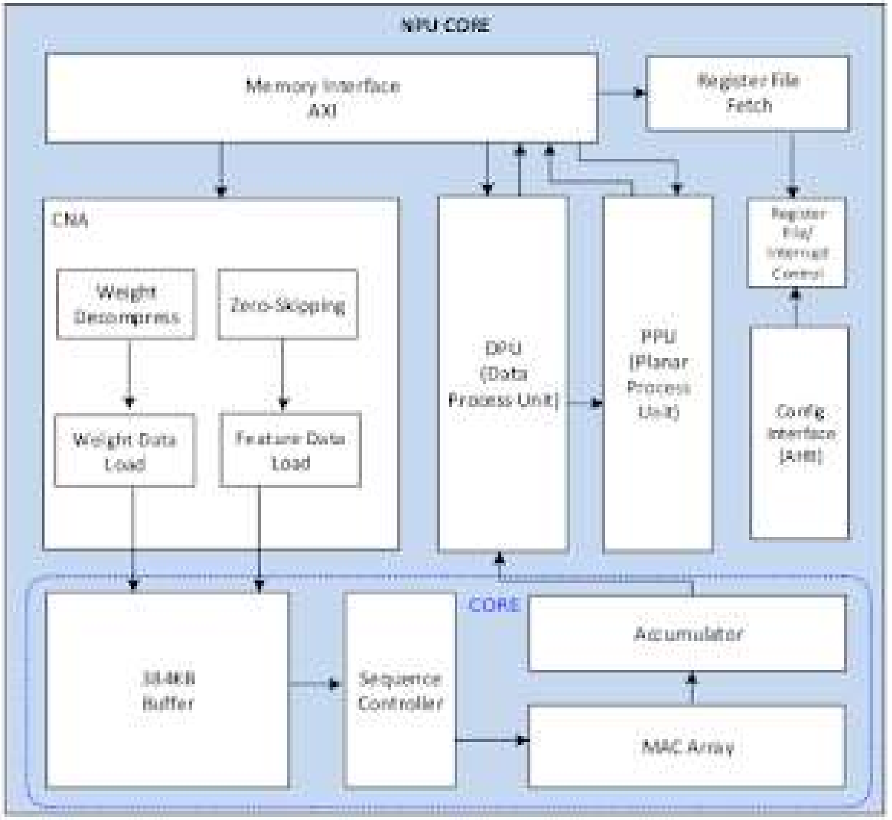
GPIO

Embedded Memory

SRAM

ROM

OTP



NPU CORE

Memory Interface
AXI

Register File
Fetch

CNA

Weight
Decompress

Zero-Skipping

Weight Data
Load

Feature Data
Load

DPU
(Data
Process Unit)

PPU
(Planar
Process
Unit)

Register
File/
Interrupt
Control

Config
Interface
(AHB)

CORE

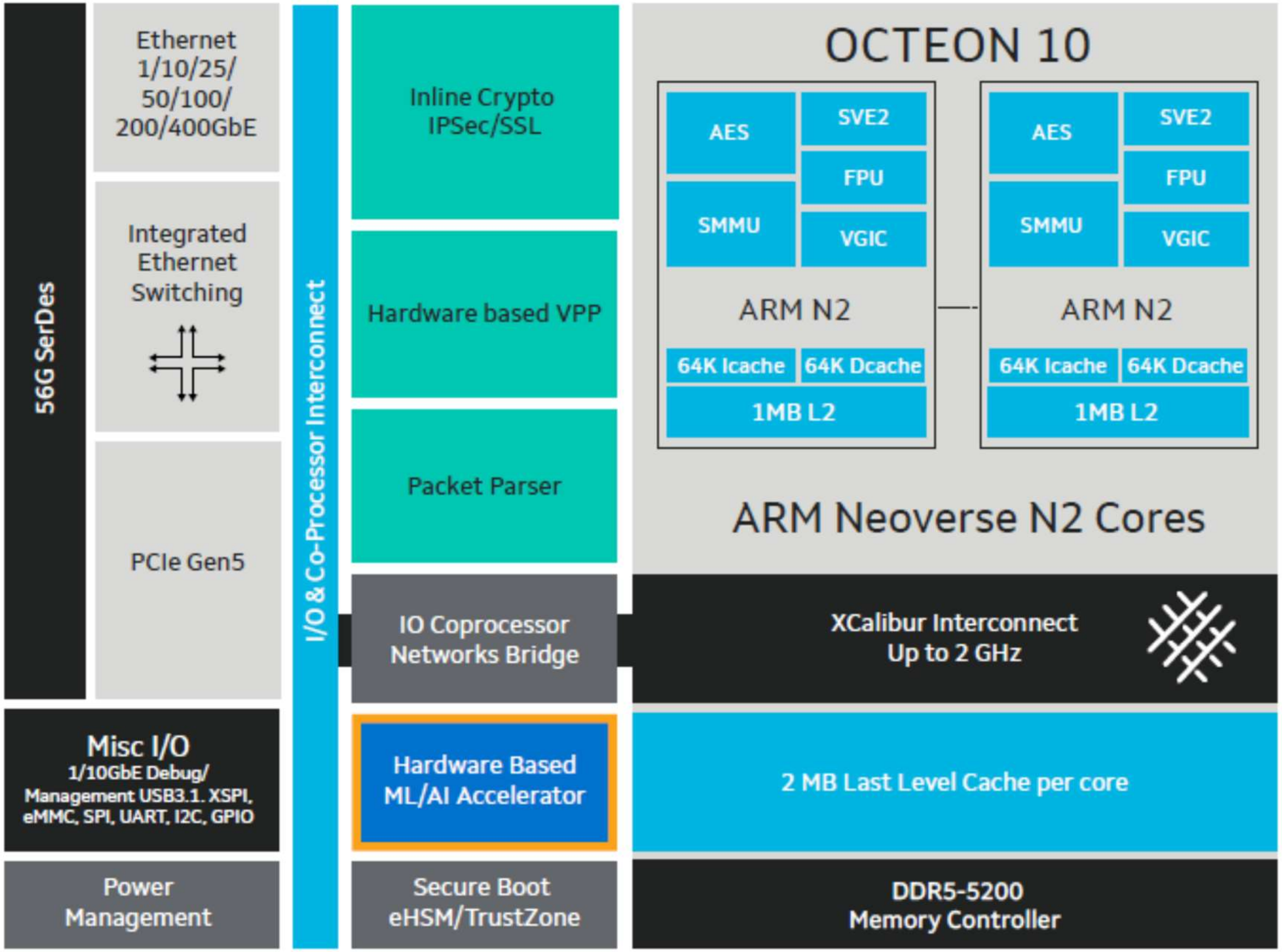
384KB
Buffer

Sequence
Controller

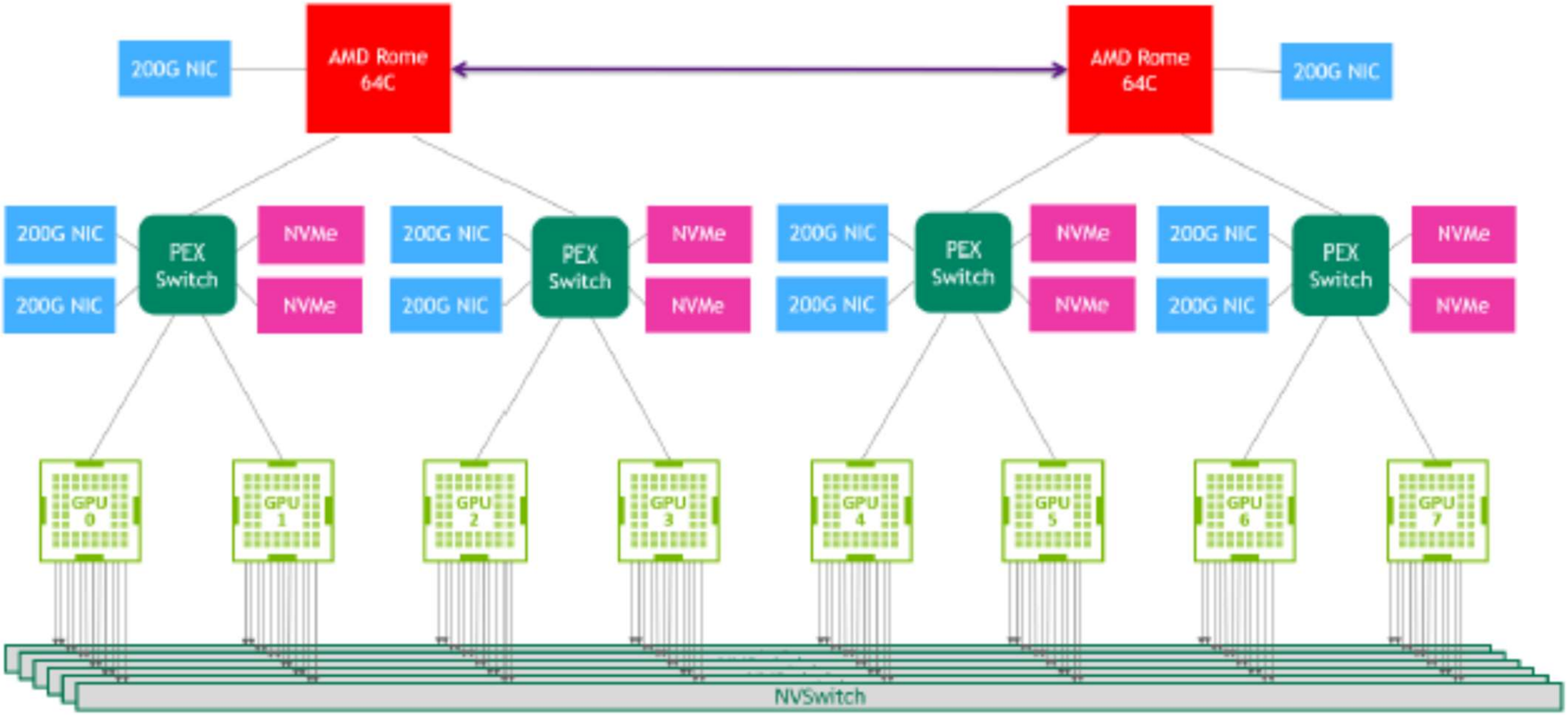
Accumulator

MAC Array

Recent Data Processing Unit (Marvell OCTEON DPU)


56G SerDes
Ethernet 1/10/25/ 50/100/ 200/400GbE
Integrated Ethernet Switching
PCIe Gen5
I/O & Co-Processor Interconnect
Inline Crypto IPSec/SSL
Hardware based VPP
Packet Parser
IO Coprocessor Networks Bridge
Hardware Based ML/AI Accelerator
Secure Boot eHSM/TrustZone
Misc I/O
1/10GbE Debug/ Management USB3.1. XSPI, eMMC, SPI, UART, I2C, GPIO
Power Management
OCTEON 10
AES
SVE2
FPU
SMMU
VGIC
ARM N2
64K Icache
64K Dcache
1MB L2
AES
SVE2
FPU
SMMU
VGIC
ARM N2
64K Icache
64K Dcache
1MB L2
ARM Neoverse N2 Cores
XCalibur Interconnect Up to 2 GHz
2 MB Last Level Cache per core
DDR5-5200 Memory Controller

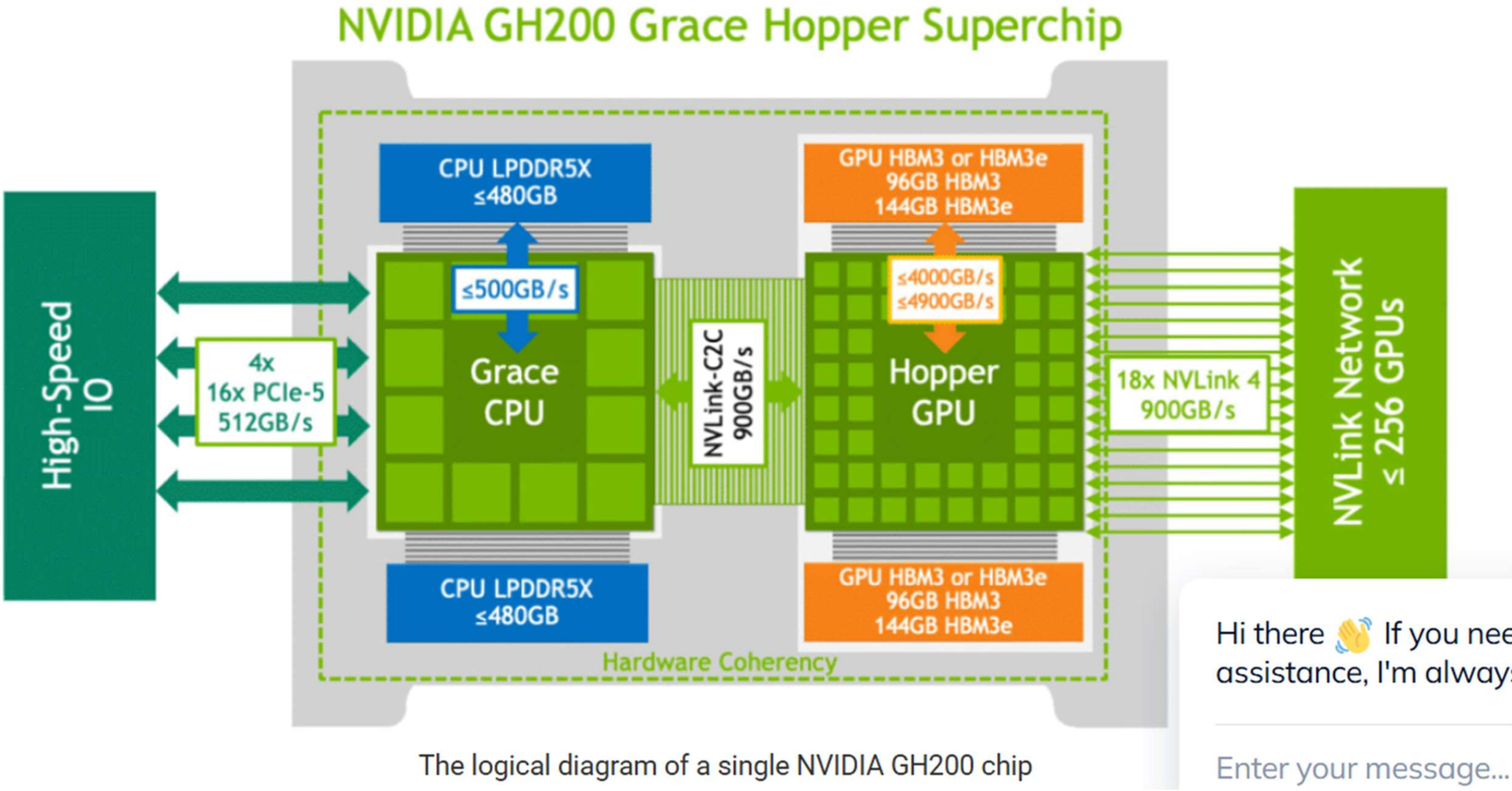
GPU-based HPC (original A100 DGX)



Note Third-Generation NVLink connectivity through NVSwitches.

Figure 26. NVIDIA DGX A100 with Eight A100 GPUs

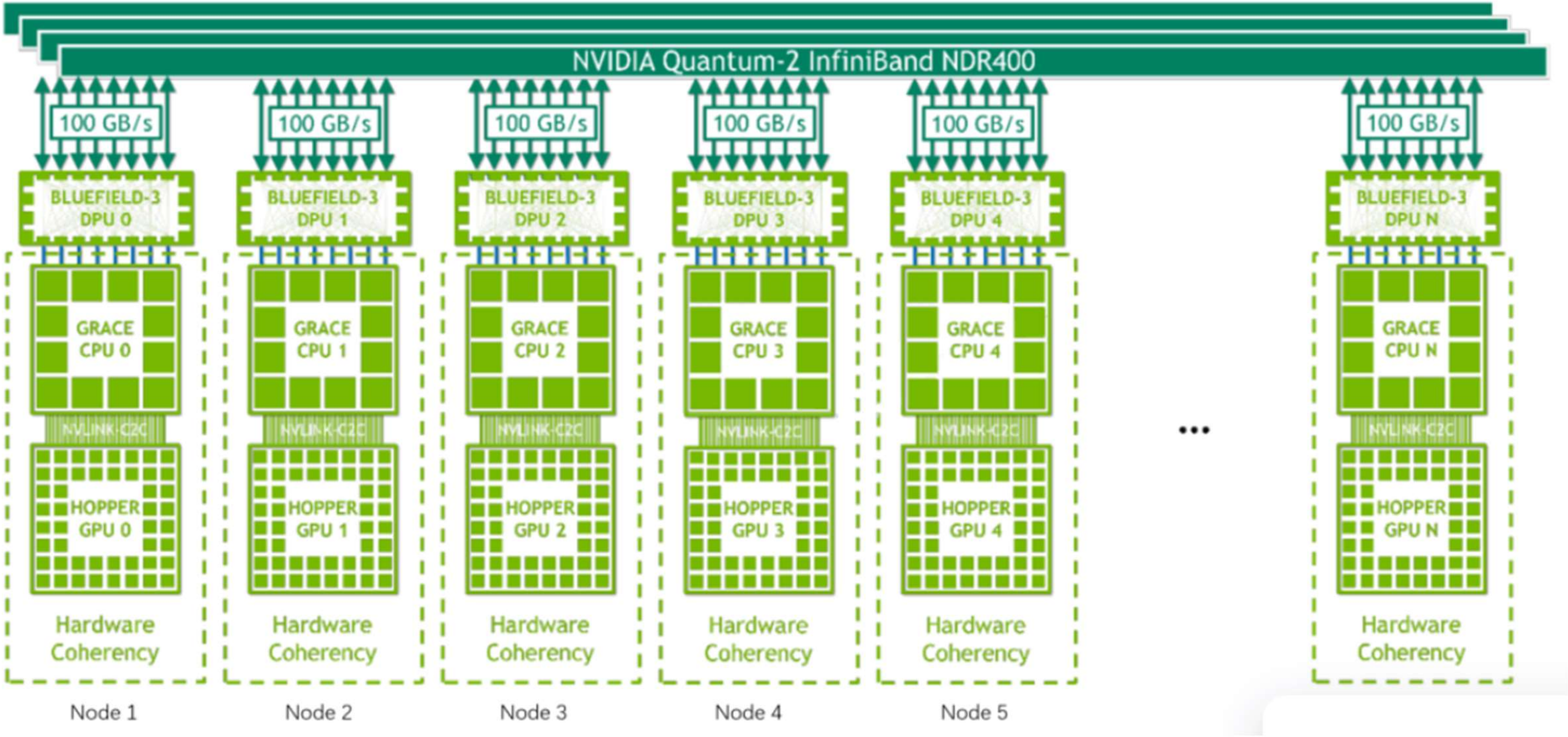
Nvidia GraceHopper GH200



The logical diagram of a single NVIDIA GH200 chip

Hi there 👋 If you nee
assistance, I'm alway

GH200 NVL system (e.g. NVL72)



History

Since the invention of the transistor and integrated circuit, DSP functions have been implemented on platforms ranging from special-purpose architectures to general-purpose computers.

One of the earliest descriptions of a special-purpose hardware architecture for digital filtering was described by Bell Labs in 1968.

Problem at beginning: *lack of flexibility*.

- *Need to perform functions that go beyond simple filtering (control, adaptive filtering, and non-linear functions such as detection)*

Solution : to use an architecture more like a general-purpose processor BUT with basic signal processing functionality

The ability to perform a multiply and add operation in parallel in the time of one instruction (basic scalar-product operation, for filtering, projections, etc.)

The ability to perform data moves to and from the arithmetic unit in parallel with arithmetic operations and modification of address pointers (filtering)

The ability to perform logical operations on data and alter control flow based on the results of these operations.

The ability to control operations by sequencing through a stored program

History (2)

In the 1960s and 1970s, multiple chips or special-purpose computers were designed for computing DSP algorithms efficiently.

- ***too costly to be used for consumer electronics***
  - **Reserved for research or military radar applications.**
  - **This functionality (arithmetic, addressing, control, I/O, data storage, control storage) needed to be realized on a single chip -> DSP could become an alternative to analog signal processing (for cell phones, MODEMS, etc.)**

late 1970s:

- ***large-scale integration reached maturity***
- ***practical to consider realizing a single chip DSP***
  - **AMI, Intel, NEC, and Bell Labs marketed solutions**

First Generation DSP

AMI S2811 (1978)

- *"Signal Processing Peripheral" in 1978.[1] was designed to operate in conjunction with a microprocessor such as the 6800 and depended upon it for initialization and configuration.[2] With a small, nonexpandable program memory of only 256 words, the S2811 was intended to be used to offload some math intensive subroutines from the microprocessor. Therefore, as a peripheral, it could not "stand alone" as could DSPs from Bell Labs, NEC, and other companies. The part was to be implemented in an exotic process technology called "V-groove." Availability of first silicon was after 1979 and was never used in any volume product.[3]*

Intel 2920 (1979)

- *"Analog Signal Processor" : on-chip analog/digital and digital/analog converter capability (like today's wireless ASICs.)*
- *Drawback: lack of a multiplier (needed to be done by a series of instructions, shifting/adding) , limited to direct addressing, no branching.*
- *a little more efficient than a general-purpose microprocessor -> sacrificed flexibility, has little resemblance to today's single-chip DSP. Too slow for any complete application, it was used as a component for part of a modem.*

NECµPD7720 (1980)

- *"Digital signal processor" all of the attributes of a modern single chip DSP.*
- *devices and tools not available (in U.S.) until 1981.*

The Bell Labs DSP1 (1979)

- *key component in AT&T's first digital switch, 5ESS, and many other telecommunications products*
- *devices with this architecture are still in manufacture today!*
- *contained all of the functional elements found in today's DSPs*

  multiplier-accumulator (MAC), parallel addressing unit, control, control memory, data memory, I/O

Comparative Table

Date	Features	Example processors
First generation : 1979 - 1985	Harvard architecture, hardwired multiplier	NECµPD7720, Intel 2920, Bell Labs DSP1, Texas Instruments TMS320C10
Second generation: 1985 - 1988	Concurrency, multiple busses, on-chip memory	TMS320C25, MC56001, DSP16 (AT&T)
Third generation: 1988 - 1992	On-chip floating point operations	TMS320C30, MC96002, DSP32C (AT&T),
Fourth generation: 1992 - 1997	Multi-processing features Image and video processors Low-power DSPs (AT&T)	TMS320C40&50 TMS320C80
Fifth generation: 1997 –	VLIW,SIMD	TMS320C6x, Philips TriMedia, Motorola/Lucent Starcore, XScale (SIMD version)

Typical Compute applications

Different Computing Architectures are used/tailored for varieties of applications

- military radar systems to consumer electronics.
- no one processor can meet the needs of all applications.
- Criteria: performance, cost, integration, ease of development, power consumptions are key points to examine when designing or selecting a particular computing architecture for a class of applications.

FIR Filtering example

basic DSP operations:

- *additions and multiplications*
- *delays*
- *array handling to access coefficients*

Each has its own special set of requirements:

additions and multiplications require

- *fetch two operands*
- *perform the addition or multiplication (usually both)*
- *store the result or hold it for a repetition*

delays require

- *hold a value for later use*

array handling requires

- *fetch values from consecutive memory locations*
- *copy data from memory to memory*

To suit these fundamental operations DSP processors often have:

- *parallel multiply and add (MAC operation)*
- *multiple memory accesses (to fetch two operands and store the result)*
- *lots of registers to hold intermediate results*
- *efficient address generation for array handling*
- *special features such as delays or circular addressing*

FIR Filtering (canonical example)

$$y_n = \sum_k h_k x_{n-k}$$

basic DSP operations:

- *additions and multiplications*
- *delays*
- *array handling to access coefficients*

Each has its own special set of requirements:

- *additions and multiplications require*
  - fetch two operands
  - perform the addition or multiplication (usually both)
  - store the result or hold it for a repetition
- *delays require*
  - hold a value for later use
- *array handling requires*
  - fetch values from consecutive memory locations
  - copy data from memory to memory

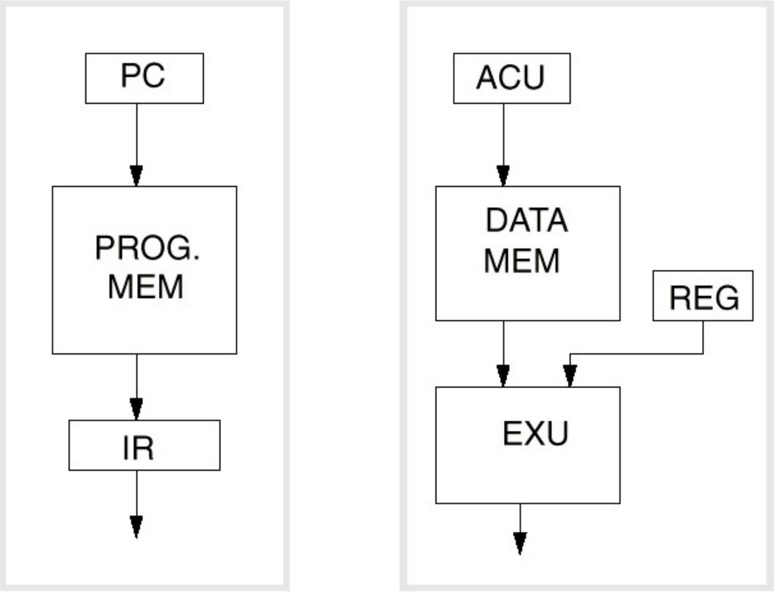
To suit these fundamental operations DSP processors often have:

- *parallel multiply and add (MAC operation)*
- *multiple memory accesses (to fetch two operands and store the result)*
- *lots of registers to hold intermediate results*
- *efficient address generation for array handling*
- *special features such as delays or circular addressing*

Harvard Architecture

***Von Neumann architecture*: Same address+data lines for instruction and data. Bad for DSP – better if MAC instruction executes in one cycle**

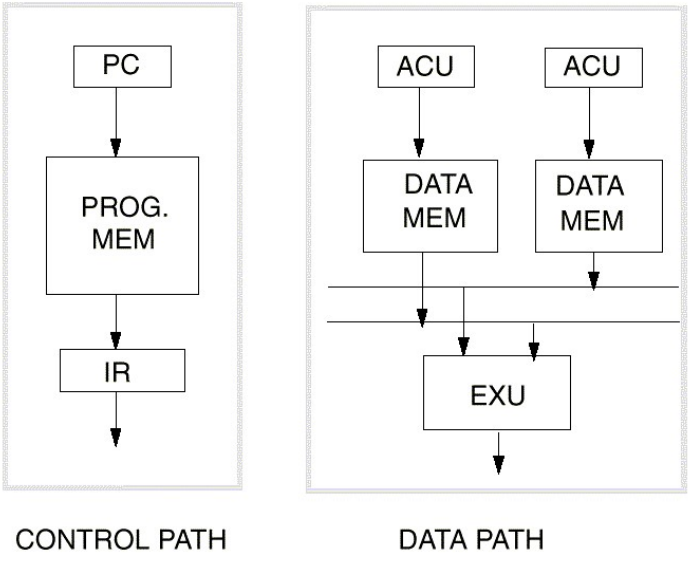
***Harvard Architecture*: 4 buses - 2 address, 2 data**



Modified Harvard Architecture

Two data memory accesses per cycle (for MAC!)

- *Three memory banks, 1 program 2 data*



Data Path

Where dominant arithmetic operations are performed

- *DSP processor data paths are highly specialized to achieve high performance - multiply-accumulate operations. Registers, Adders, Multipliers, Comparators, Logic operators, Multiplexers, Buffers represent 95% of a typical DSP data path.*

Multiplier

- *single-cycle multiplier required. Important distinction between multipliers in DSPs is the size of the product according to the size of the operands. In general, multiplying two n-bit fixed-point numbers requires a 2xn bits to represent the correct result. For this reason DSPs have in general a multiplier, which is twice the word length of the native operands.*

Accumulator Registers

- *Accumulators registers hold intermediate and final results of multiply-accumulate and other arithmetic operations. Most DSP processors have two or more accumulators. In general, the size of the accumulator is larger than the size of the result of a product. These additional bits are called guard bits. These bits allow accumulating values without the risk of overflow and without rescaling. N additional bits allow up to 2^N accumulations to be performed without overflow. Guards bits method is more advantageous than scaling the multiplier product since it allows the maximum precision to be retained in intermediate steps of computations and does not require scaling at intermediate steps which would incur a performance loss.*

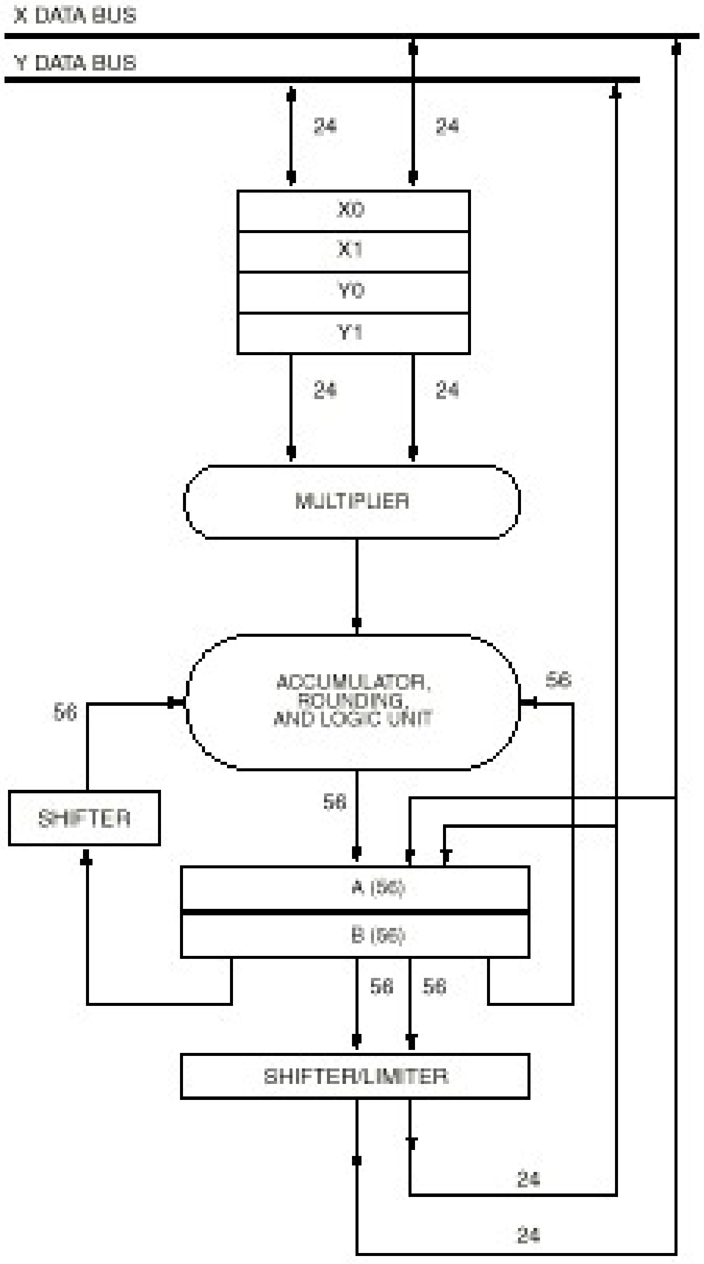
ALU

- *Arithmetic logic units implement basic arithmetic and logical operations. Operations such as addition, subtraction, and, or are performed in the ALU.*

Shifter

- *In fixed-point arithmetic, multiplications and accumulations often induce a growth in the bit width of results. Scaling is then necessary to pass results from stage to stage and is performed through the use of shifters.*

Datapath example (MC56002)



Addressing

ability to generate new addresses efficiently is a characteristic feature of DSP processors.

- ***Most DSP processors include one or more special address generation units (AGUs)***
- ***AGU can perform one or more special address generation per instruction cycle without using the processor main data path.***
- ***address calculation takes place in parallel with arithmetic operations on data, improving processor performance.***

register-indirect addressing

- ***Data addressed is in memory and the address of the memory location containing the data is held in a register***
- ***natural way to work with arrays of data (e.g. filter coefs)***
- ***efficiency from an instruction-set point of view -> allows powerful and flexible addressing with few bits in the instruction word.***
- ***When operand is fetched from memory, address register can be incremented to point to the next needed value in the array.***

Typical Increment formats

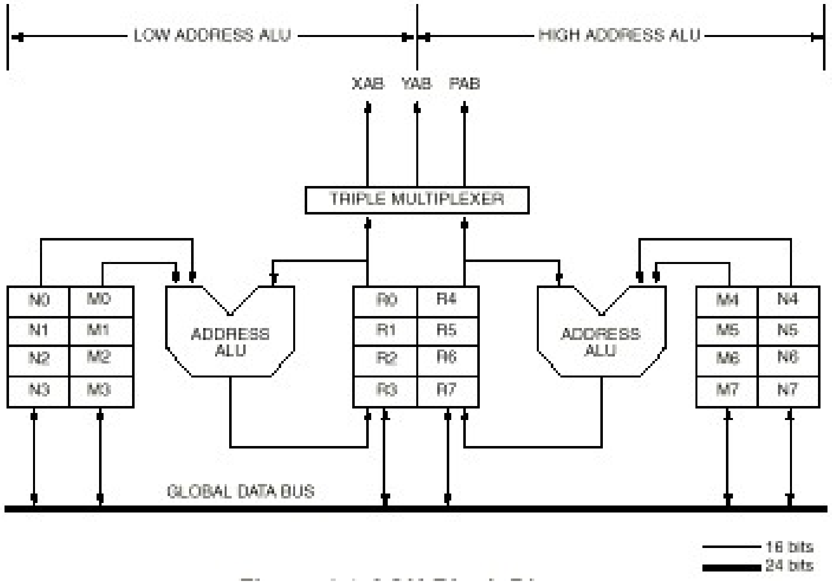
*rP	Register indirect	read the data pointed to by the address in register rP
*rP++	Postincrement	read the data pointed to by the address in register rP Having read the data, postincrement the address pointer to point to the next value in the array
*rP--	Postdecrement	Having read the data, postdecrement the address pointer to point to the previous value in the array
*rP++rI	Register postincrement	Having read the data, postincrement the address pointer by the amount held in register rI to point to rI values further down the array
*rP--rIr not on Intel	Bit reversed (FFT)	having read the data, postincrement the address pointer to point to the next value in the array, as if the address bits were in bit reversed order

Addressing (2)

Modulo-N/circular addressing

- *Circular access of a memory buffer (x[BASEADR + i%N])*
- *Instead of comparing an address counter to a calculated value to see whether or not the end of the buffer has been reached, dedicated registers are used to automatically perform this check*

Ex: MC56002



Peripherals

Most DSP processors have on-chip peripherals and interfaces to allow the DSP to be used in an embedded system -> minimize external hardware to support its operation and interfacing (to converters, busses, fifos, other DSPs in multiprocessor systems, etc.)

Serial port

- ***A serial interface transmits and receives data one bit at a time. These ports have a variety of applications like sending and receiving data samples to and from A/D and D/A converters and codecs, sending and receiving data to and from other microprocessors or DSPs, communicating with other hardware.***

Host Port

- ***Some DSPs provide a host port for connection to a general-purpose processor or another DSP. Host ports are usually specialized 8 or 16 bit bi-directional parallel ports that can be used to transfer data between the DSP and the host processor.***
- ***Ex: (TMS320C6x : configurable 32-bit port can even implement PCI protocol, useful in high-end systems, e.g. basestations, DSLAMs, etc)***

Link ports or communication ports

- ***This kind of port is dedicated to multiprocessor operations. It is in general a parallel port intended for communication between the same types of DSPs.***

Peripherals (2)

Interrupt controller

- *An interrupt is an external event that causes the processor to stop executing its current program and branch to a special block of code called an interrupt service routine. Typically this code deals with the origin of the interrupt and then returns from the interrupt. There are different interrupt sources:*
- *On-chip peripherals generate interrupts: serial ports, timers, DMA,…*
- *External interrupt lines: dedicated pins on the chip to be asserted by external circuitry (synchronization events)*
- *Software interrupts: also called exceptions or traps, these interrupts are generated under software control or occurs for example for floating-point exceptions (division-by-zero, overflow and so on).*
- *DSPs associate interrupts with different memory locations. These locations are called interrupt vectors. These vectors contain the address of the interrupt routines. When an interrupt occurs, the following scenario is often encountered:*

  Save program counter in a stack
  Branch to the relevant address given by the interrupt vector table
  Save all registers used in the interrupt routine
  Perform dedicated operations
  Restore all registers
  Restore program counter
- *Priority levels can be assigned to the different interrupt through the use of dedicated registers. An interrupt is acknowledged when its priority level is strictly higher that current priority level.*

Timers

- ***Programmable timers are often used as a source of periodic interrupts. Completely software-controlled to activate specific tasks at chosen times. It is generally a counter that is preloaded with a desired value and decremented on clock cycles. When zero is reached, an interrupt is issued.***
- ***Used to sequence events (real-time processing), like perform a filtering operation periodically with a period depending on the input event***
- ***Allows for partitioning of DSP between different processing threads***

DMA – Direct Memory Access

- ***technique whereby data can be transferred to or from the processor's memory without the involvement of the processor itself. DMA is commonly used to provide improved performance with input/output devices.***
- ***software loads the DMA controller***

  the starting address for the transfer

  the number of words to be transferred

  the source and the destination.
- ***Once obtaining bus-mastership, the DMA controller performs the transfer and optionally signals completion through an interrupt.***
- ***Some processors have multiple DMA channels***

Superscalar Architectures

The key to higher performance in microprocessors for a broad range of applications is the ability to exploit fine-grain, instruction-level parallelism. Some methods for exploiting fine-grain parallelism include:

- *pipelining*
- *multiple processors*
- *superscalar implementation*
- *specifying multiple independent operations per instruction*

Superscalar Architectures

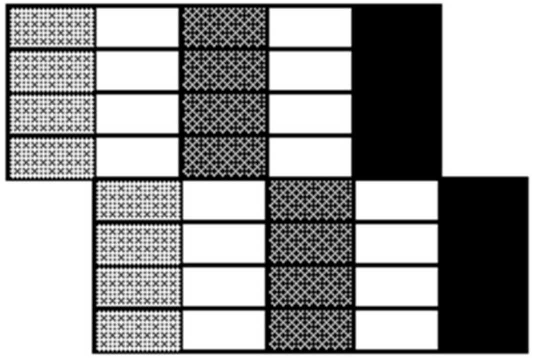
Pipelining is now universally implemented in high-performance processors. Little more can be gained by improving the implementation of a single pipeline.

Using multiple processors improves performance for only applications designed to do so.

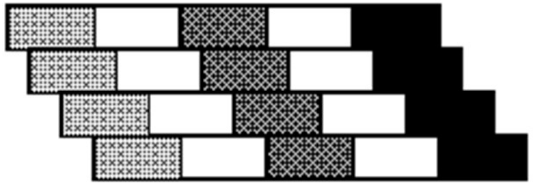
Superscalar implementations can improve performance for all types of applications. **Superscalar** (*super*:beyond; *scalar*: one dimensional) means the ability to fetch, issue to execution units, and complete more than one instruction at a time.

Superscalar implementations are required when **architectural compatibility must be preserved**, and they will be used for entrenched architectures with legacy software, such as the x86 architecture that dominates the PC market.

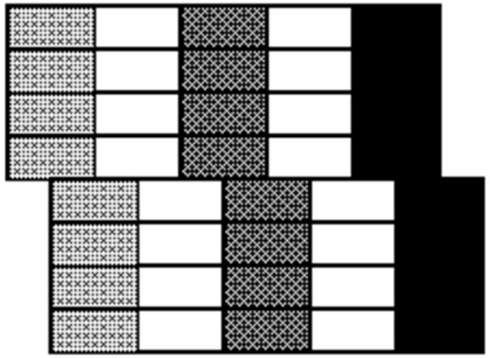
Superscaler alternatives



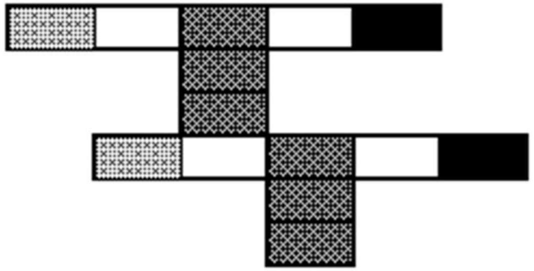
Superscaler



Superpipelined



Superscaler/Superpipelined



VLIW

Accelaration

VLIW – Very Long Instruction Word

Specifying multiple operations per instruction creates a very-long instruction word architecture or VLIW. A VLIW implementation has capabilities very similar to those of a superscalar processor—issuing and completing more than one operation at a time

- *exception: the VLIW* hardware *is not responsible for discovering opportunities to execute multiple operations concurrently. For the VLIW implementation, the long instruction word already encodes the concurrent operations. This explicit encoding leads to dramatically reduced hardware complexity compared to a high-degree superscalar implementation of a RISC or CISC.*

The big advantage of VLIW, then, is that a highly concurrent (parallel) implementation is much simpler and cheaper to build than equivalently concurrent RISC or CISC chips. VLIW is a simpler way to build a superscalar microprocessor.

- *Reason for its popularity in high-end DSPs (TI C6x,FreeScale StarCore, TriMedia)*

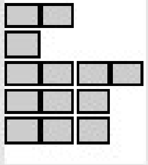
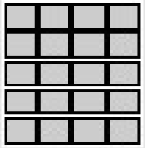
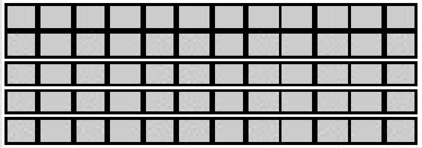
ARCHITECTURE CHARACTERISTIC	CISC	RISC	VLIW
INSTRUCTION SIZE	Varies	One size, usually 32 bits	One size
INSTRUCTION FO RMAT	Field placement varies	Regular, consistent placement of fields	Regular, consistent placement of fields
INSTRUCTION SEMANTICS	Varies from simple to complex; possibly many dependent operations per instruction	Almost always one simple operation	Many simple, independent operations
REGISTERS	Few, sometimes special	Many, general-purpose	Many, general-purpose
MEMORY REFE RENCES	Bundled with operations in many different types of instructions	Not bundled with operations, i.e., load/store architecture	Not bundled with operations, i.e., load/store architecture
HARDWARE DESIGN FOCUS	Exploit microcoded implementations	Exploit implementations with one pipeline and & no microcode	Exploit implementations with multiple pipelines, no microcode & no complex dispatch logic
PICTURE OF FIVE TYPICAL INSTRU CTIONS ▭ = 1 BYTE			

TABLE 1

Hardware vs. Software Parallelism

From the point of view of the programmer the primary difference is that in Superscaler RISC/CISC (Pentium, PowerPC, Sparc, etc.) processors, parallelism is guaranteed by the processors hardware

- *So, parallelism is transparent to the programmer, who has to live with the choices*
- *Results in complex circuits*

In VLIW, parallelism is up to the programmer (i.e. compiler)

- *Requires either tricky programming (in assembler) to achieve optimality or a very good compiler (a problem too) to achieve optimal performance*
- *Putting the optimization in the software reduces the time to market.*

Hardware vs. Software Parallelism

HW/SW trade-off has a significant side benefit: **the complexity is paid for only once, when the compiler is written instead of every time a chip is fabricated**.

Benefits:

- *smaller chip, which leads to increased profits for the microprocessor vendor and/or cheaper prices for the customers that use the microprocessors.*

Complexity is usually easier to deal with in a software design than in a hardware design. Thus, the chip may cost less to design, be quicker to design, and may require less debugging, all of which are factors that can make the design cheaper.

Improvements to the compiler can be made after chips have been fabricated!

Practical VLIW

The simplest VLIW instruction format encodes an operation for every execution unit in the machine. This makes sense under the assumption that every instruction will always have something useful for every execution unit to do.

it is typically not possible to pack every instruction with work for all execution units. Also, in a VLIW machine that has both integer and floating-point execution units, the best compiler would not be able to keep the floating-point units busy during the execution of an integer-only application.

VLIW has had large practical impact on DSP systems (why?)

Problems and optimizations with VLIW

The problem with instructions that do not make full use of all execution units is that they waste precious processor resources: instruction memory space, instruction cache space, and bus bandwidth.

There are at least two solutions to reducing the waste of resources due to sparse instructions. First

- *instructions can be compressed (e.g. source coding!) to allocate the fewest bits to the most frequently used operations.*

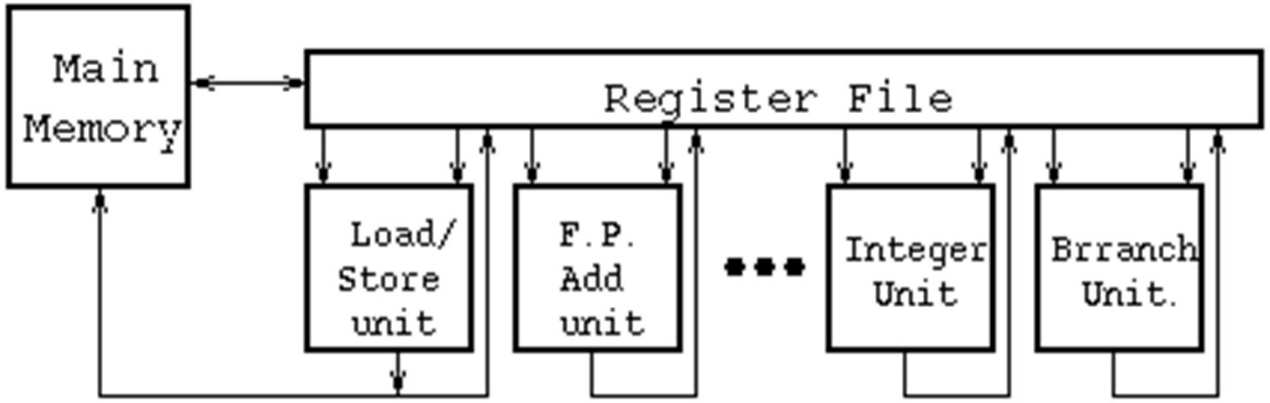
Second, it is possible to define an instruction word that encodes fewer operations than the number of available execution units.

- *Imagine a VLIW machine with ten execution units but an instruction word that can describe only five operations. In this scheme, a unit number is encoded along with the operation; the unit number specifies to which execution unit the operation should be sent.*
- *The benefit is better utilization of resources. A potential problem is that the shorter instruction prohibits the machine from issuing the maximum possible number of operations at any one time. To prevent this problem from limiting performance, the size of the instruction word can be tuned based on analysis of simulations of program behavior.*

Of course, it is completely reasonable to combine these two techniques: use compression on shorter-than maximum-length instructions.

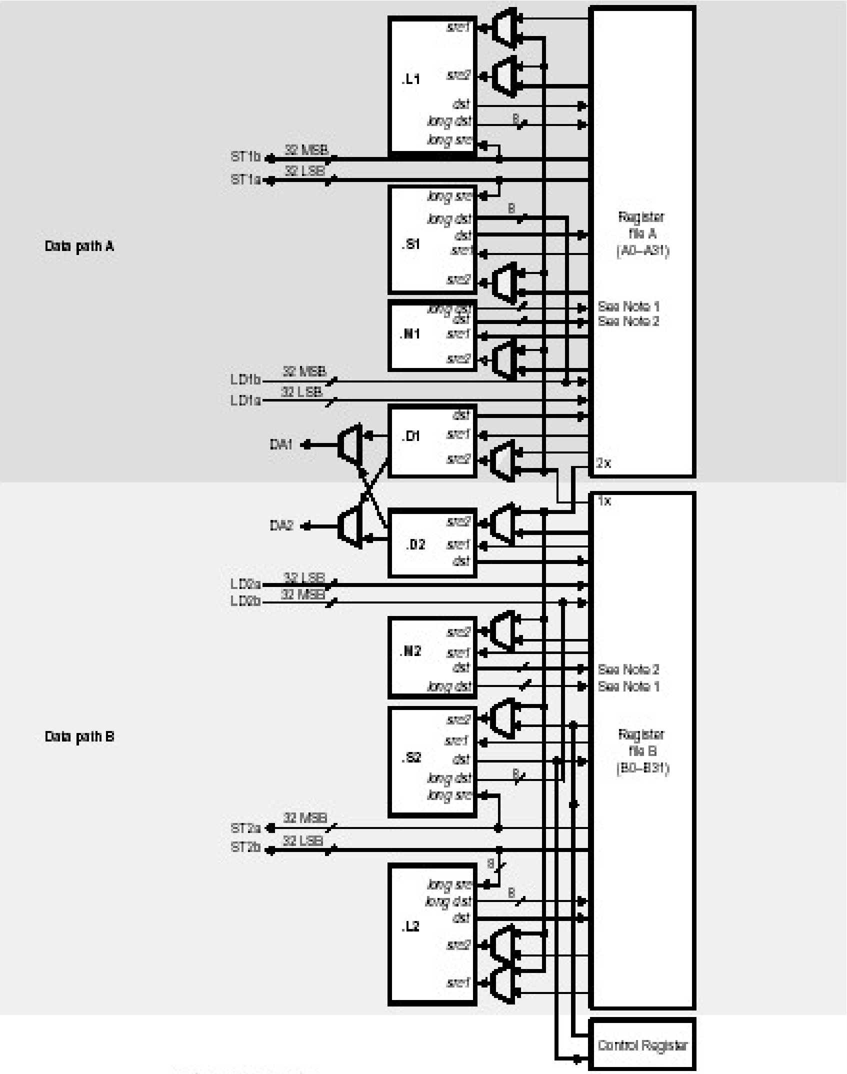
Software must be opt!..

VLIW Processor and Instruction Format



Load/ store	Fp add	FP Multiply	Branch	■■■	Integer ALU

C6x Datapaths and Example instruction (FIR Filter)



Notes for .M unit:
1. *long dst* is 32 MSB
2. *dst* is 32 LSB

Load a halfword (16 bits)

Do this on unit D1

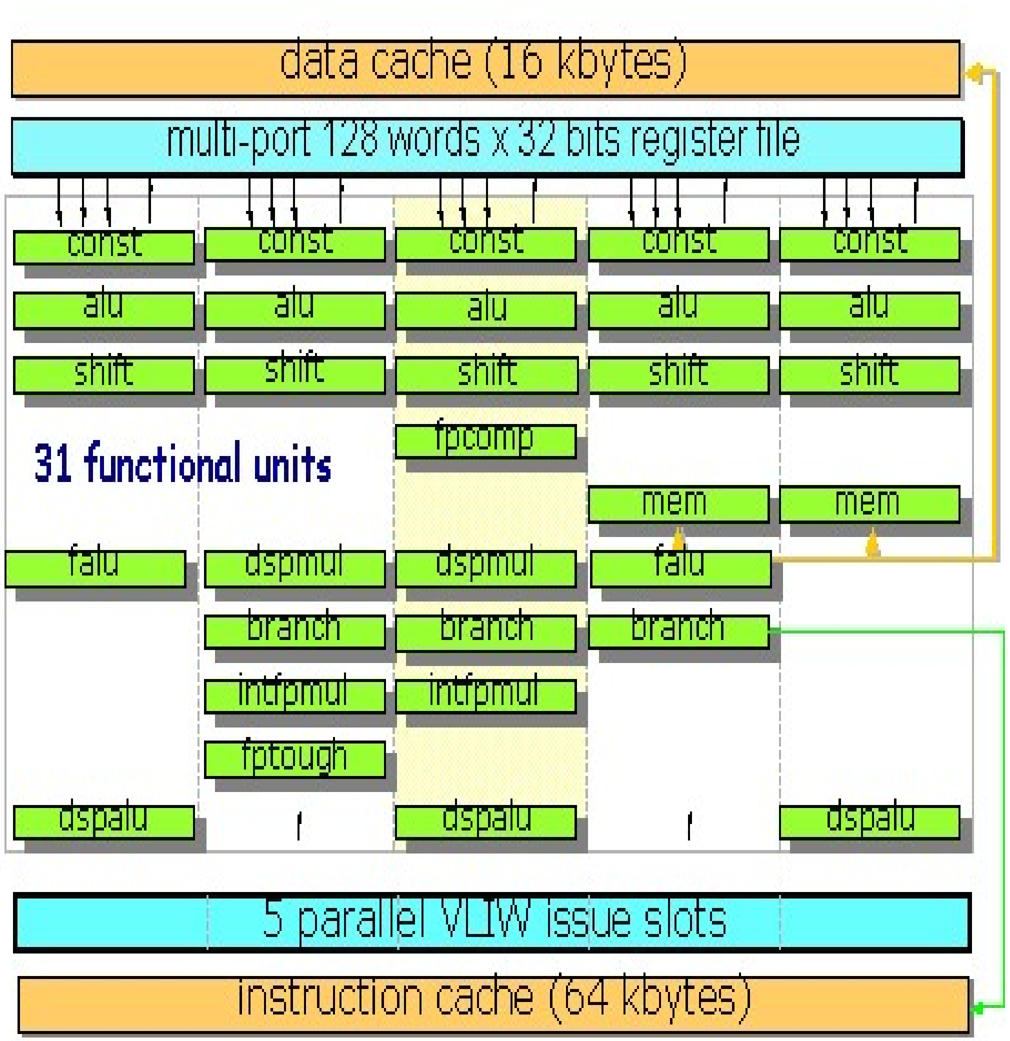
```
FIRLOOP:
            LDH  .D1   *A1++, A2    ; Fetch next sample
||          LDH  .D2   *B1++, B2    ; Fetch next coeff.
||   [B0]   SUB  .L2   B0, 1, B0    ; Decrement count
||   [B0]   B    .S2   FIRLOOP      ; Branch if non-zero
||          MPY  .M1X  A2, B2, A3   ; Sample × Coeff.
||          ADD  .L1   A4, A3, A4   ; Accumulate result
```

Use the cross path

Predicated instruction (only if B0 non-zero)

Run these instruction in parallel

TriMedia Core Organization



31 Functional Units

5-way parallelism

Some are SIMD!

Popular processor for imaging

Single-Instruction Multiple-Data (SIMD)

SIMD provides vector processing instructions to both CISC/RISC architectures and VLIW

- ***CISC Examples: Pentium 2/3/4 (MultiMedia eXtensions – MMX, SSE/SSE2/SSSE3/SSE4/AVX/AVX2/AVX512)***
- ***RISC Examples: UltraSparc (VIS), PowerPC (AltiVec), XScale (Wireless MMX), ARM (NEON)***
- ***VLIW/SIMD Examples: TI C64, TriMedia, Itanium/Itanium2, StarCore, Hexagon***

Targets audio/image processing (for PCs and PDAs/Smartphones) and signal processing in general (for DSPs)

Classical MMX Example

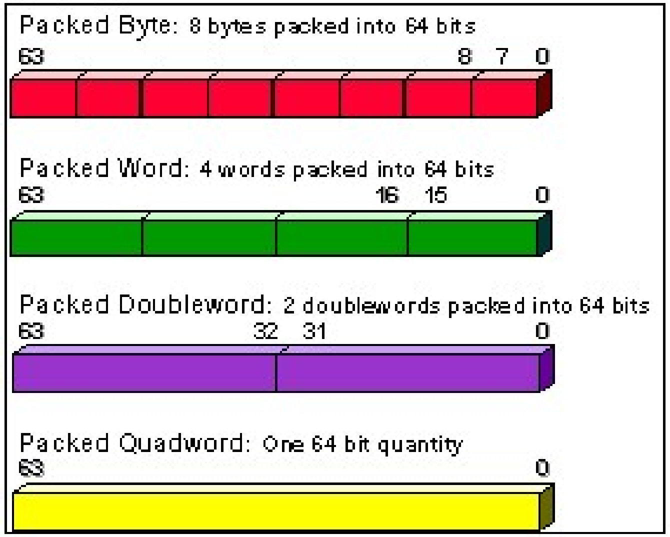

Packed Byte: 8 bytes packed into 64 bits
63
8
7
0
Packed Word: 4 words packed into 64 bits
63
16
15
0
Packed Doubleword: 2 doublewords packed into 64 bits
63
32
31
0
Packed Quadword: One 64 bit quantity
63
0

Figure 3. MMX Technology Packed Data Types

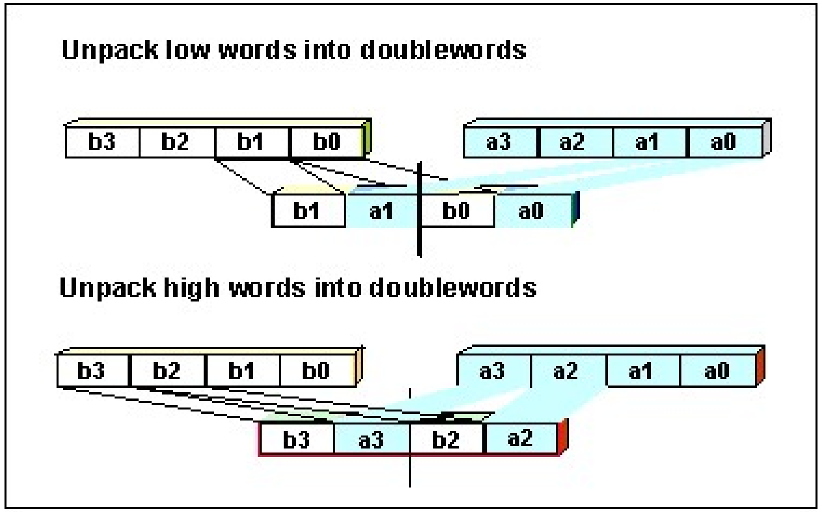

Unpack low words into doublewords
b3
b2
b1
b0
a3
a2
a1
a0
b1
a1
b0
a0
Unpack high words into doublewords
b3
b2
b1
b0
a3
a2
a1
a0
b3
a3
b2
a2

Figure 2. MMX Technology Unpacked Instruction

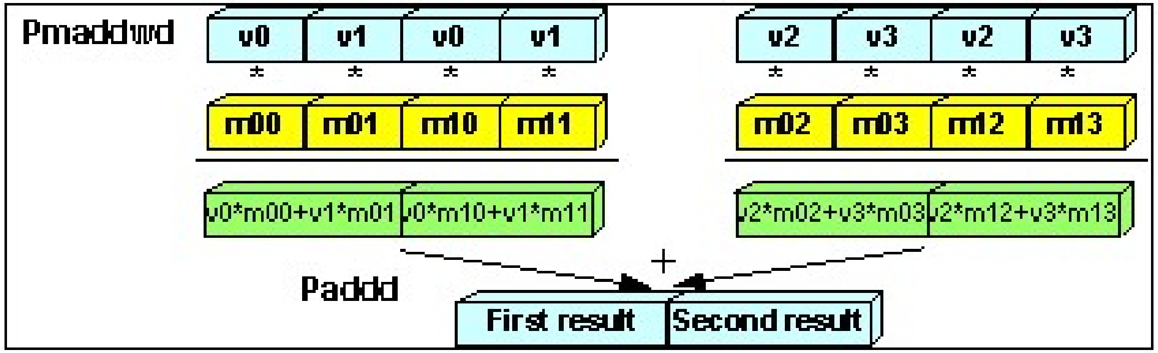


Figure 5. MMX Technology Matrix-Vector Multiplication

Important programming issues

- **SIMD can be a powerful technique for DSP if coding is done properly**
  - ***Most operations can be vectorized***
    - **FIR filtering is essentially sliding-window inner-products**
    - **Viterbi Algorithm can be easily vectorized by grouping state transitions**
    - **Complex multiply is 1 SIMD MAC -> FFT!**
- **Memory-alignment is an issue for filters**

DSP on FPGAs

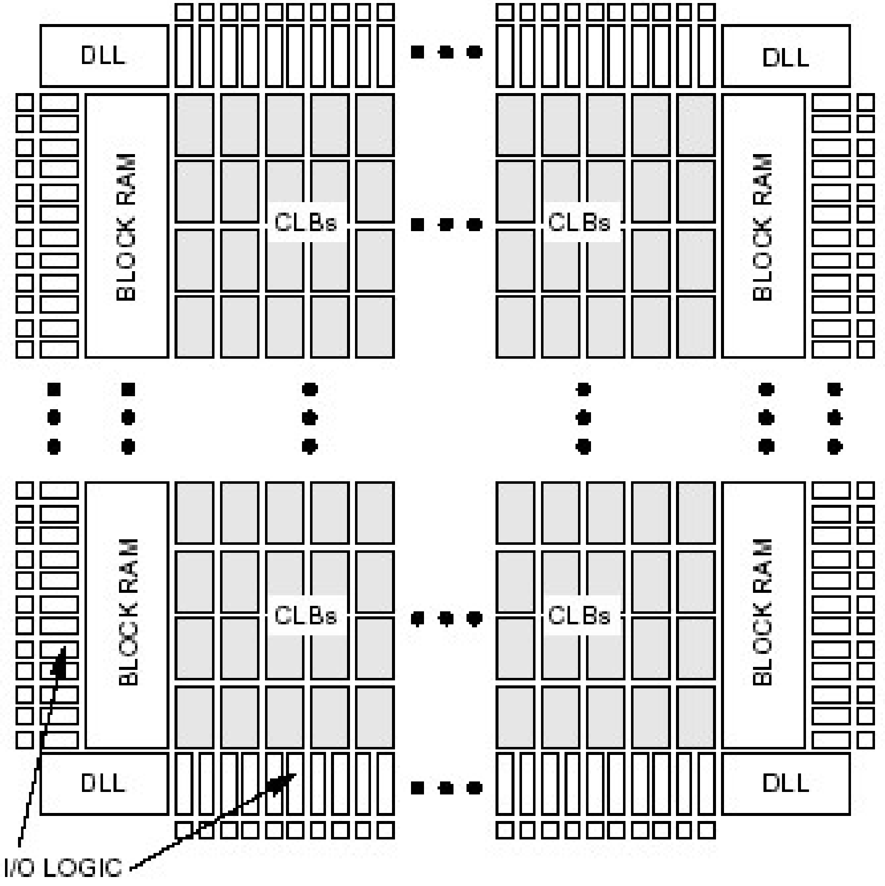
An FPGA is a reconfigurable circuit made up of memory, logic and switching fabric and a simple interconnection network

It was traditionally a prototyping technology for ASICs since efficiency (power, size) is sacrificed to provide for reconfigurability

It is commonly used for high-end DSP and networking applications (basestations, DSLAMs, specialized routers, etc.)

It is possible to design custom SoC architectures (processor, co-processors + OS) on modern FPGAs

Typical FPGA Floorplan



FPGA contains

Block RAM

Computational Logic Blocks (CLBs)

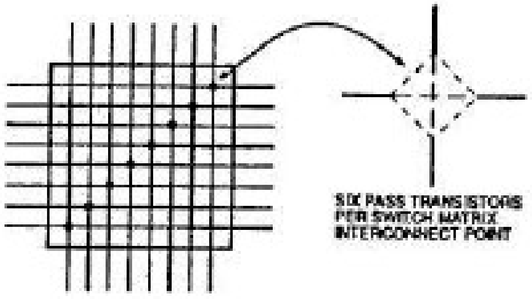
Frequency synthesizers (DLLs)

I/O Pads+Logic

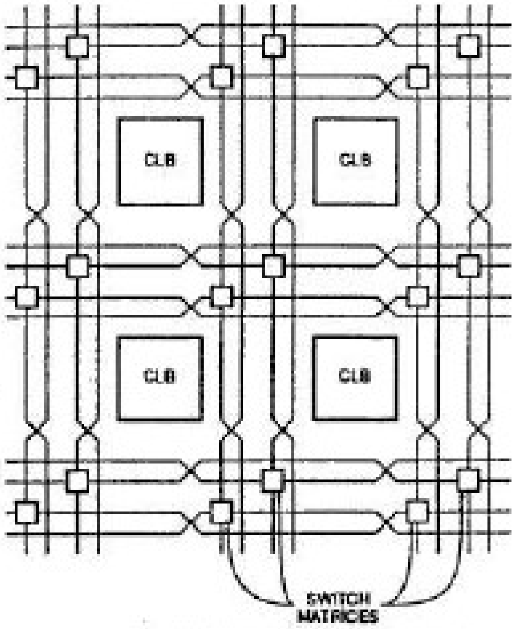
DSP functions (Multipliers)

Typical FPGA Routing

Routing

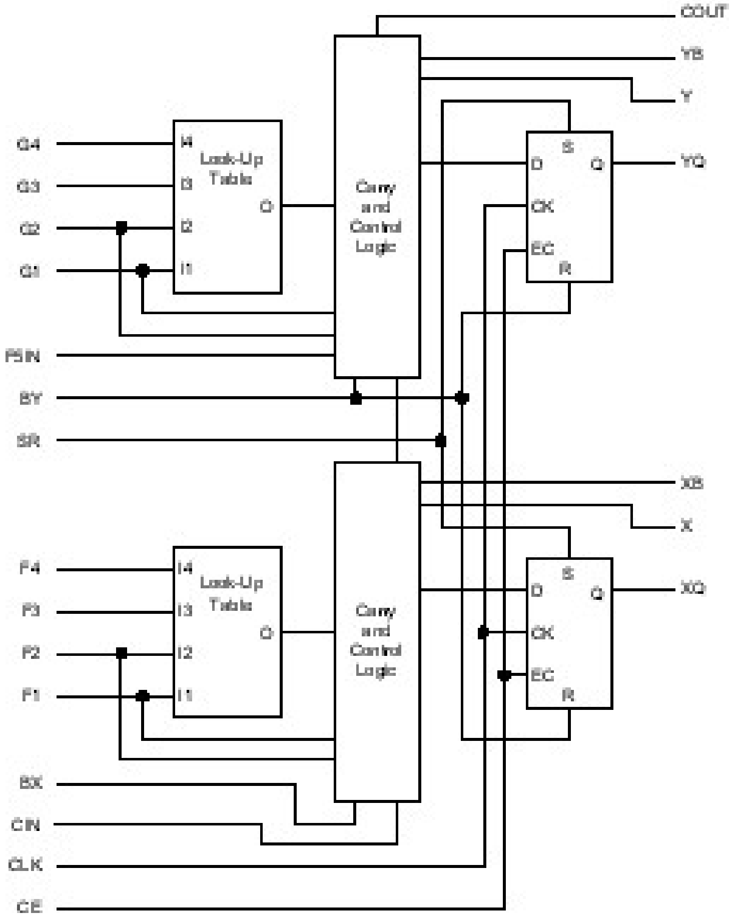


Single-length line Switch Matrix connections



Double-length lines in CLB array

Typical CLB



CLB contains

2 Look-up-tables (LUTs) 4x1

2 Carry/control logic blocks

2 Flip-flops

Building Complex DSP Systems on FPGAs

DSP systems can be standalone on an FPGA

- *Front-end signal processing in a basestation*
  - Filtering, channelization, up/down conversion

A more complete solution (baseband MODEM) can benefit from software control and mini-OS via

- *Hardcore embedded processors (e.g. ARM)*
- *Softcore embedded processor using FPGA resources (MicroBlaze-Xilinx, NIOS-Altera, LEON 3-generic)*
- *Busses and peripherals on FPGA fabric*
  - Xilinx – IBM CoreConnect (PLB/OBP Bus) => ARM Cortex + AMBA/AXI (partially on FPGA fabric)
  - Altera – Avalon, AMBA
  - Leon 3 - AMBA
  - RISC-V

DSP circuits can then be tailored to the application at hand either as

- *Co-processors on processor busses (local or peripheral)*
- *Extensions to instruction set (e.g. with PowerPC/LEON)*

Rich IP library

Many FPGA vendors give away IP blocks to encourage use of their technology

- *Modulators/Demodulators*
- *FFTs,DCTs*
- *Reconfigurable Filters*
- *Channel encoder/decoders*

See web for examples

This time …

Superscalar and Parallel architectures

- *SIMD, VLIW*

DSP on FPGAs