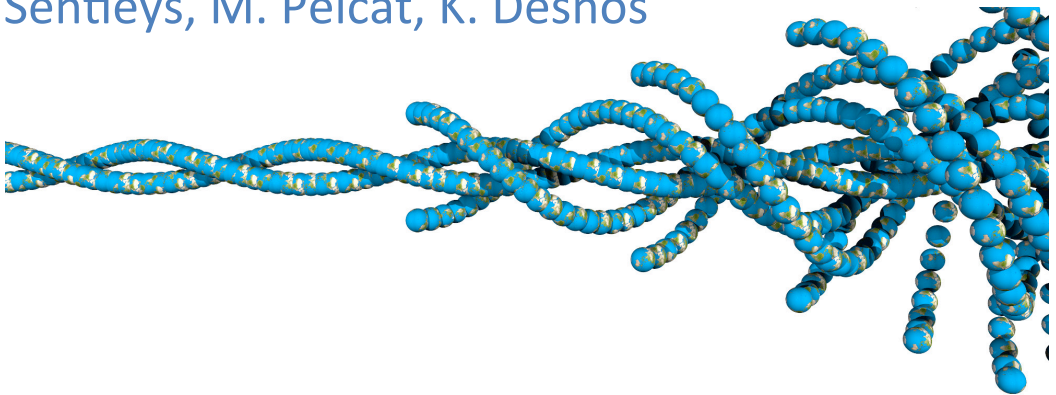
Architectures DSP
du MAC au multi-cœurs

Daniel MENARD
INSA Rennes, Dépt. EII
UMR CNRS IETR
daniel.menard@insa-rennes.fr

With slides from O. Sentieys, M. Pelcat, K. Desnos

ÉCOLE THÉMATIQUE ARCHI'15



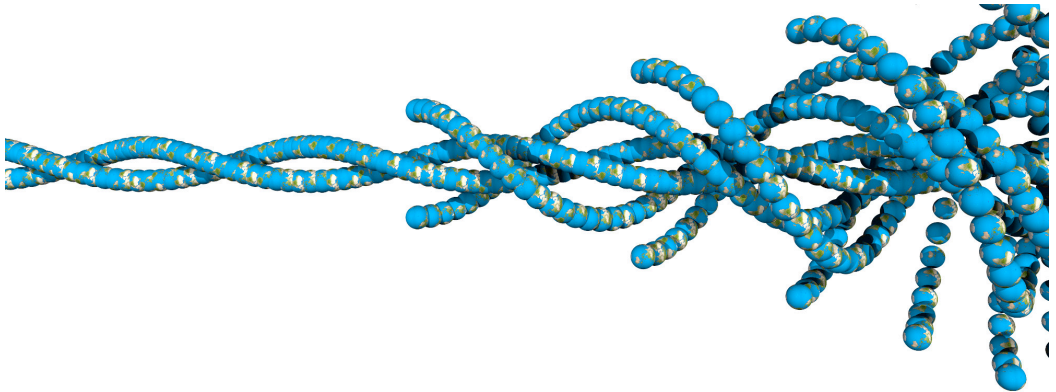
Outlines

- Introduction
- Architecture of Low Power DSP
- Architecture of High Performance single-core DSP
- Architecture of High Performance multi-core DSP
- Arithmetic for DSP Applications
- Conclusion

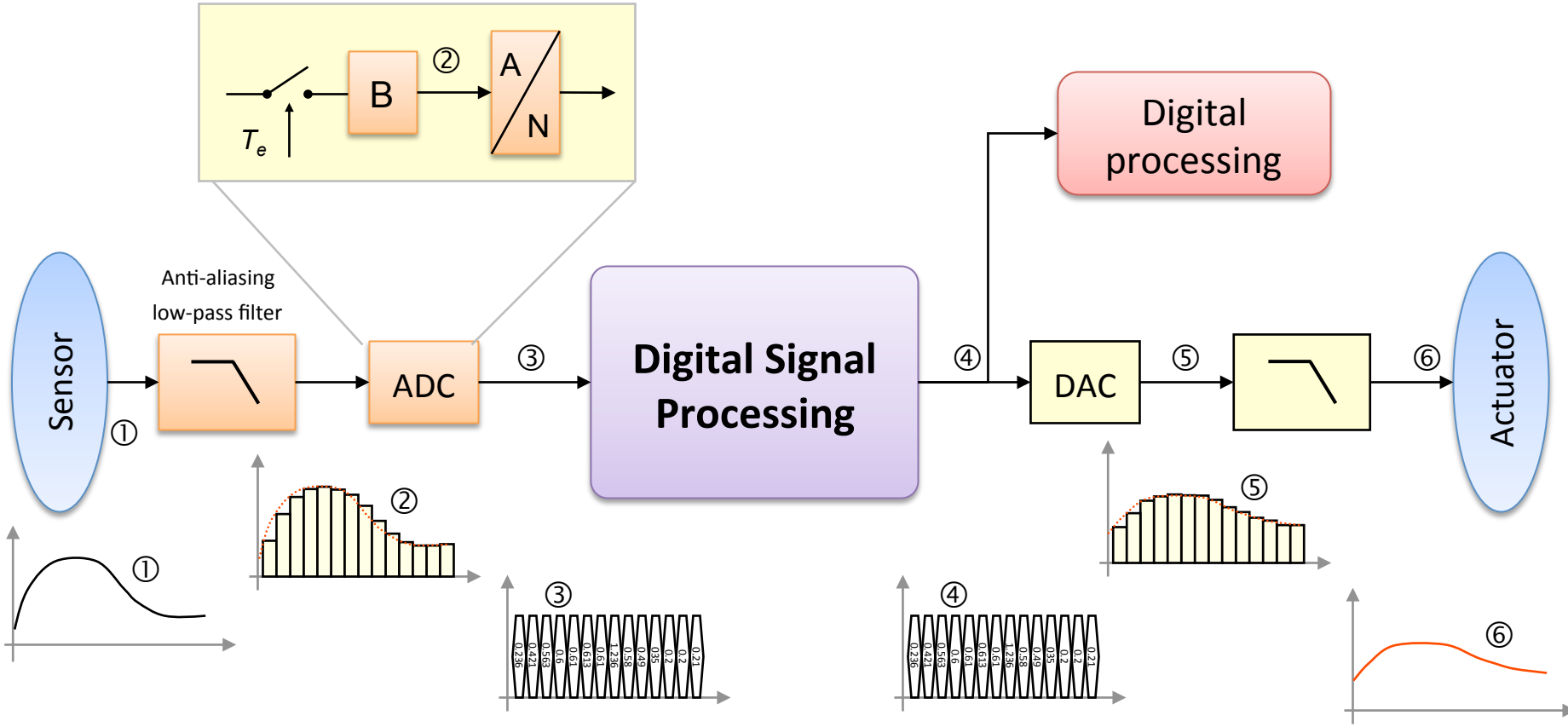
Part 1

Introduction

- DSP applications, algorithms and kernels
- Architecture for DSP applications
- Digital Signal Processors



Signal processing framework



Why Digital Signal Processing ?

- **Benefits**
  - No drift
    - Temperature, aging, component characteristics
  - Accuracy
    - Guaranteed number of bits
  - Flexibility
    - Multitasks
  - Prediction
    - Computer simulation
  - Rapid prototyping
    - Modification by software changes

Why Digital Signal Processing ?

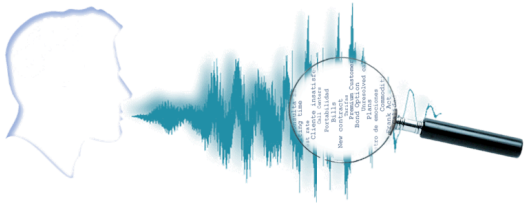
- **Benefits**
  - Performance
    - No phase distortion, adaptive filtering, etc..
  - VLSI Integration progress
  - Production
    - No adjustment, reproducibility
- **Disadvantages**
  - Cost:
    - High for simple applications
  - Processing complexity
    - Proportional to signal bandwidth
  - Complexity:
    - Development of both hardware and software systems

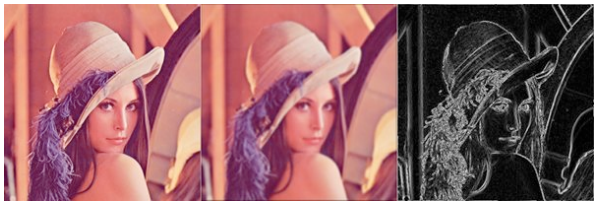
Semiconductor manufacturing processes

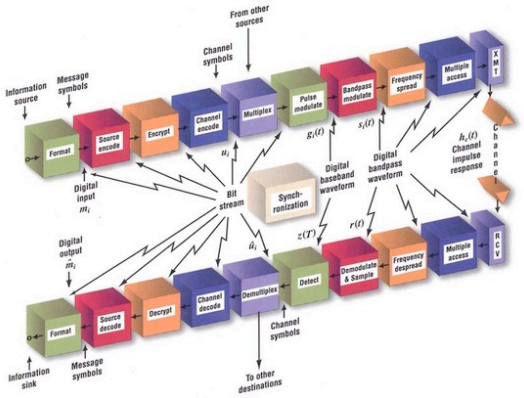
10 µm — 1971
3 µm — 1975
1.5 µm — 1982
1 µm — 1985
800 nm — 1989
600 nm — 1994
350 nm — 1995
250 nm — 1997
180 nm — 1999
130 nm — 2002
90 nm — 2004
65 nm — 2006
45 nm — 2008
32 nm — 2010
22 nm — 2012
14 nm — 2014
10 nm — est. 2017
7 nm — est. 2020
5 nm — est. 2022

Applications integrating Digital Signal Processing

- Audio
- Speech
- Image
- Video
- Digital communications
- Radar
- Sonar
- Seismology
- Biomedicine
- ….





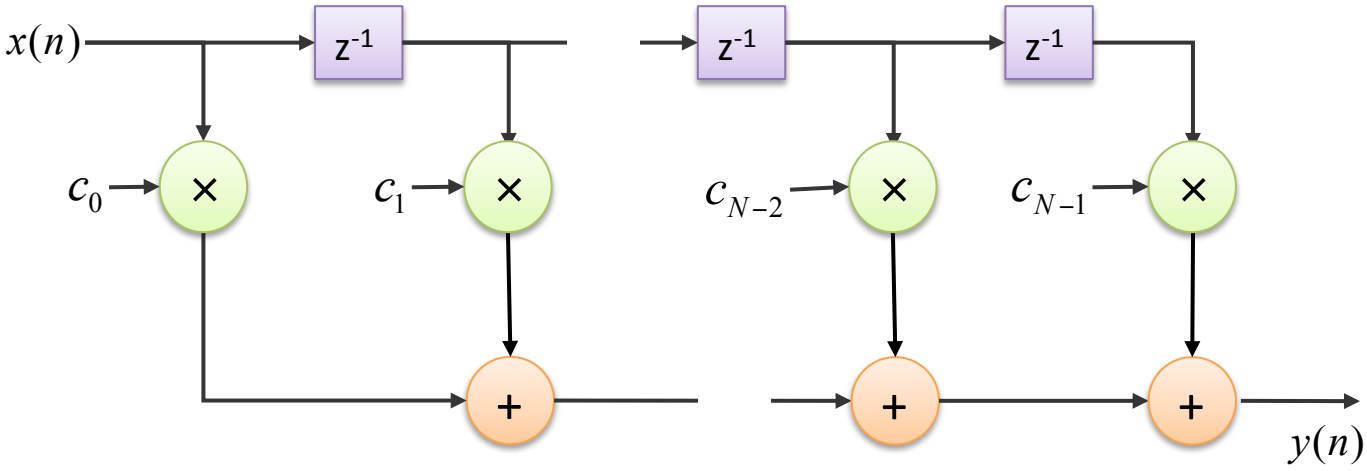


Algorithms for Digital Signal Processing

- **Digital filters**
  - Linear Time Invariant
    - Finite Impulse Response (FIR), Infinite Impulse Response (IIR)

Finite Impulse Response (FIR)

$$y(n) = \sum_{i=0}^{N-1} c_i . x(n-i)$$



Algorithms for Digital Signal Processing

- **Digital filters**
  - Linear Time Invariant
    - Finite Impulse Response (FIR), Infinite Impulse Response (IIR)
  - Adaptive filters
    - Least Mean Squares (LMS), Normalized Least Mean Squares (NLMS), Recursive Least Squares (RLS), Affine Projection (AP)
- **Transforms**
  - Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT)
- **Forward Error Correction (communication systems)**
  - Viterbi, turbo-code, LDPC

Kernels for Digital Signal Processing

- **Filtering, convolution**

$$y = y + x \times h$$

MAC (multiplication-accumulation)

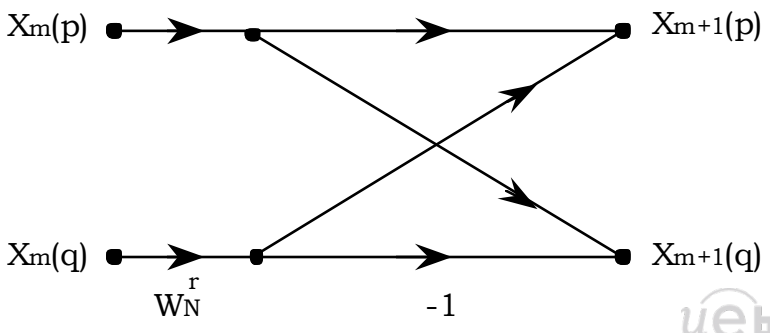
- **Adaptation**

$$y_n = y_{n+1} + x \times h$$

MAD (multiplication-addition)

- **Complex multiplication, FFT**

$$X_{m+1}(p) = X_m(p) + W_N^r . X_m(q)$$
$$X_{m+1}(q) = X_m(p) - W_N^r . X_m(q)$$

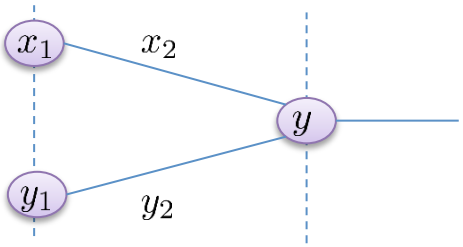


Kernels for Digital Signal Processing

- **Viterbi decoding**

$$a1 = x1 + x2$$
$$a2 = y1 + y2$$
$$y = (a1 > a2)?a1 : a2$$

ACS : Add Compare Select



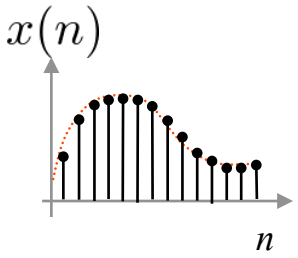
- **Motion estimation**

$$y = y + |u - v|$$

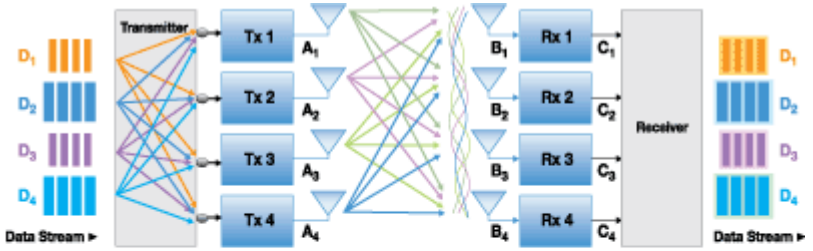
SAD : Sum of absolute difference

Signals in Digital Signal Processing

- **Types**
  - Scalar



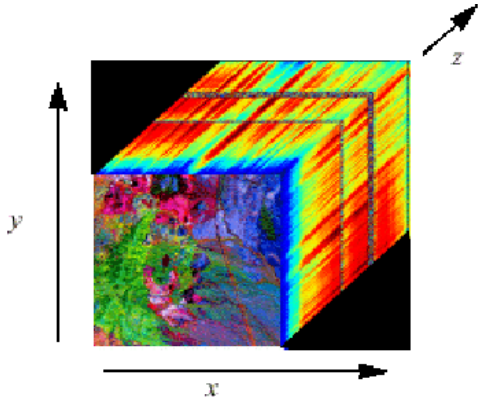
  - Vector



MIMO systems

  - Matrix
    - Image

  - Multidimensional
    - Hyper-spectral image



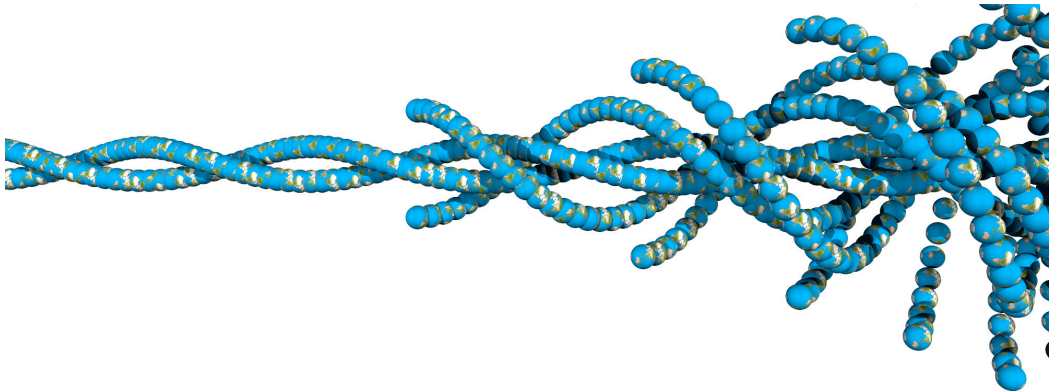
DSP applications features

- **Real time**
  - Link with the sampling period
- **High amount of data to process**
  - Data flow oriented
    - Few conditional processing
  - High bandwidth with the memory
  - Repetitive and intensive processing
    - Loops and high complexity (MOPS)
  - Simple pattern
    - MAC, MAD, …
  - Parallelism
  - Complex addressing modes

Part 1

Introduction

- DSP applications, algorithms and kernels
- **Architecture for DSP applications**
- Digital Signal Processors



Challenges for embedded systems

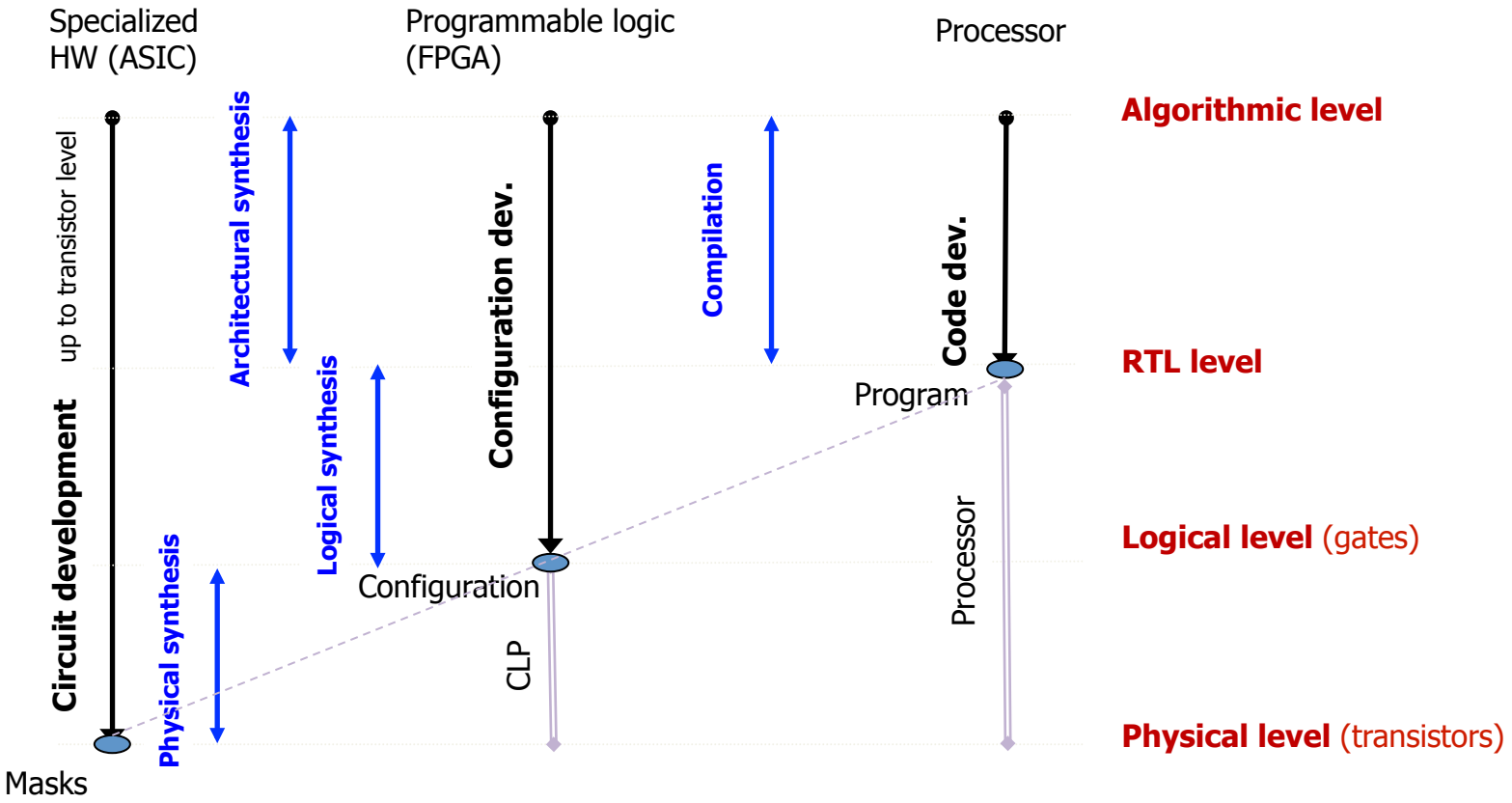
- **Different criteria to optimize**
  - Area (Size):
    - Chip area: [mm²], [eq. gates]
  - Speed (Performance): [MHz], [op/s]
    - Operations/second
    - Time required to perform a task
  - Power consumption [mW]
    - Static power
    - Dynamic power

$$P_{dyn} = \alpha . C_{eq} . V_{DD}^2 . f_{CLK}$$

  - Cost
    - Platform Cost [€]
      - ✓ For high volumes : proportional to the area
    - Design effort [person-month]
      - ✓ Link to the flexibility of the platform and level of abstraction for the development

Flexibility

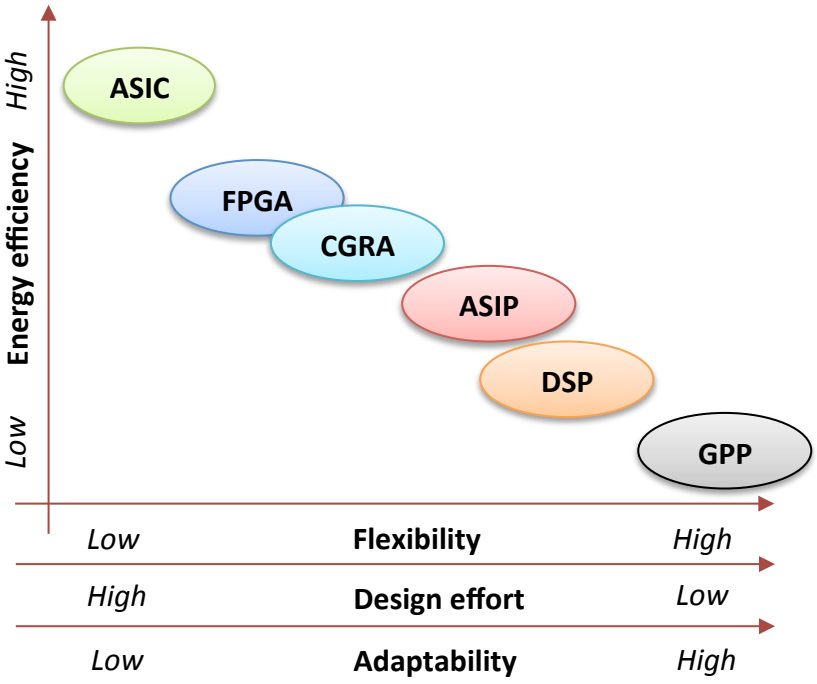
- **Design effort**

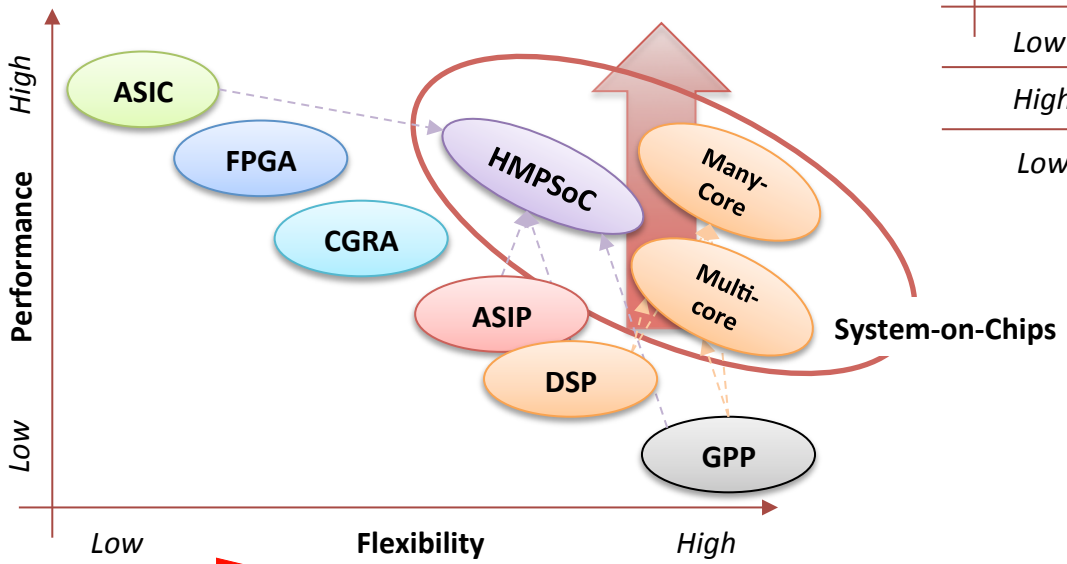


Processor classification

- **Non-specialized processors**
  - GPP : General-Purpose Processors
- **Domain-specific processors**
  - Signal processing: DSP (Digital Signal Processor)
  - Image and video processing: GPU (Graphics processing unit)
  - Network: Network processor
- **Processors specific for several applications**
  - ASIP : Application Specific Instruction set Processor

Trade-off between flexibility and performance or energy

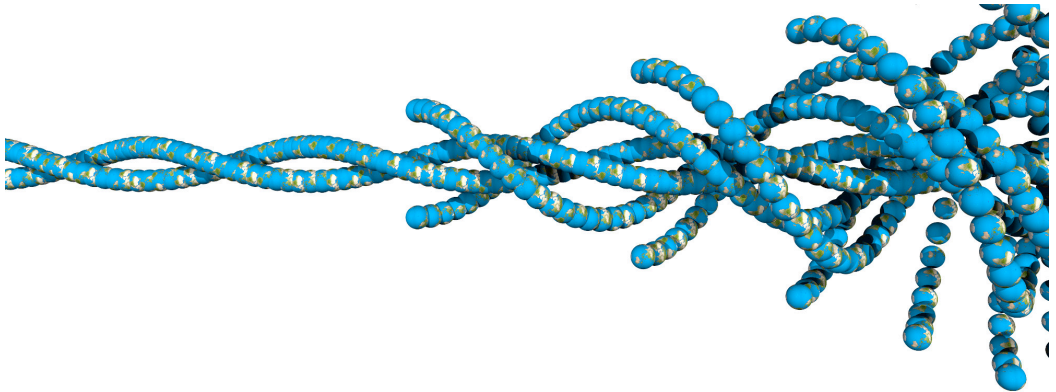




Part 1

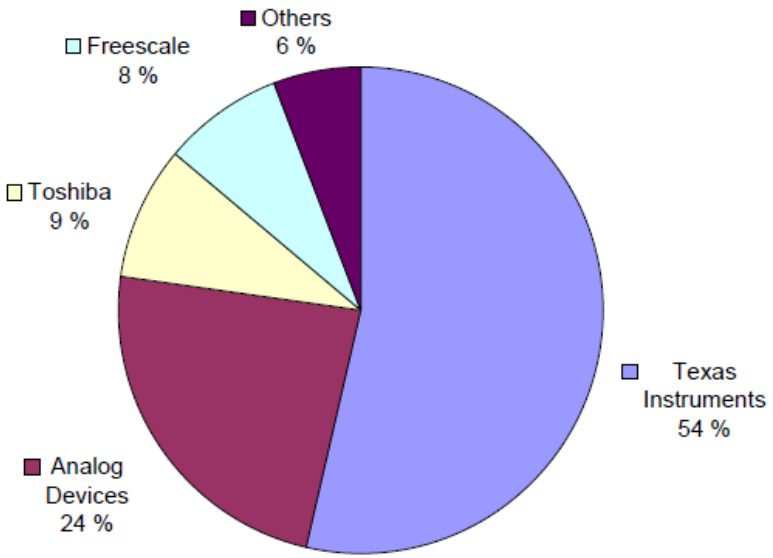
Introduction

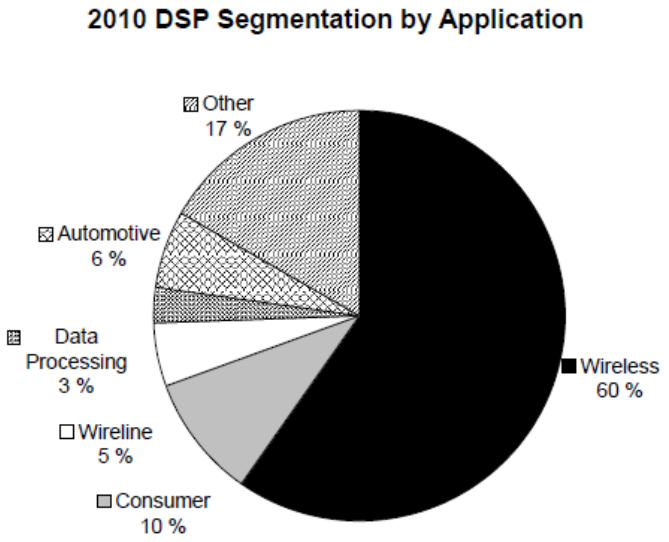
- DSP applications, algorithms and kernels
- Architecture for DSP applications
- **Digital Signal Processors**



Semiconductor market

- **DSP Market**





Source: WSTS, Oppenheimer & Co.

Semiconductor market

- **Semiconductor market in 2010**
  - *Off-the-shelf DSP revenue : 4,5 billion $*
  - *Integration of DSP cores in ASSP (58 billion $)*

Components	Revenue (B.$)
DSP	4,5
MCU (µC)	14,8
MPU (µP)	39,9
ASSP	58
Display driver	5,1
ASIC	8
FPGA	3,5
Digital logic	1,6
Memory	69,6
Discrete, opto, sensor, actuors	48,4
Analog Standard (SLIC)	17,8
Analog ASSP	24,5
Total	**295,7**

Source : Oppenheimer Semiconductors: Technology and Market Primer 7.0

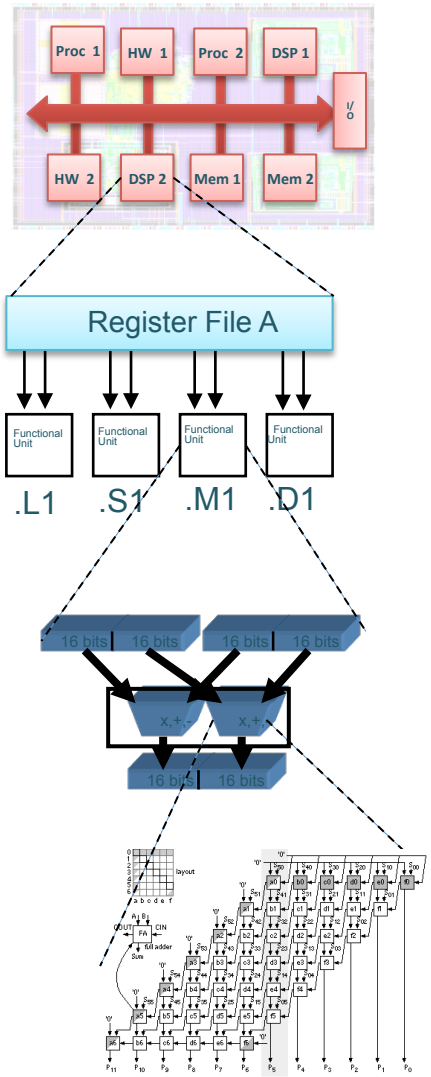
Many-core
(distributed memory)

Multi-core (MPSoc)
(shared memory)

Instruction
(ILP)

Data (SWP)

Bit (HW ops.)



Levels of parallelism

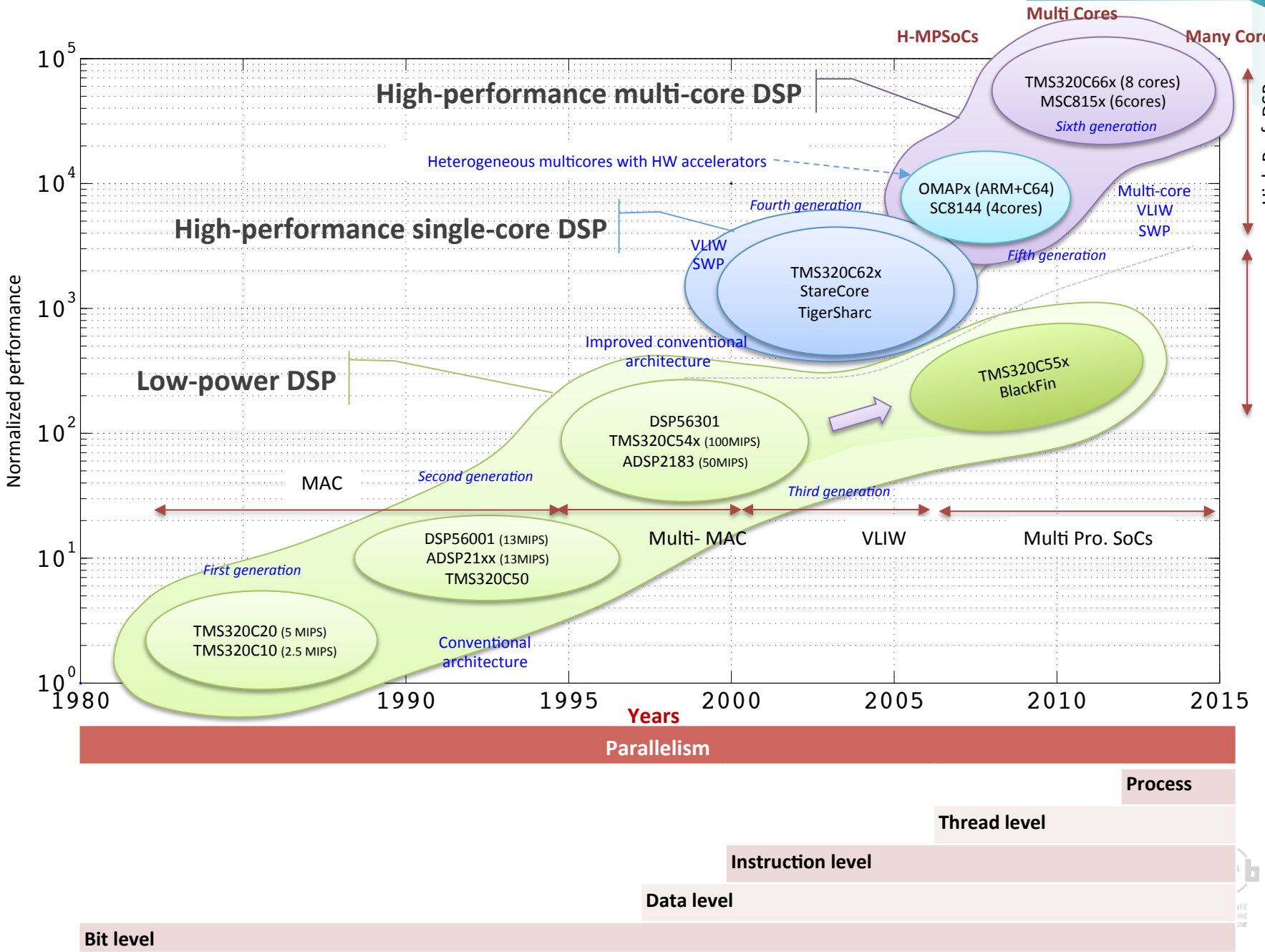
Process

Threads

Operation

Bit

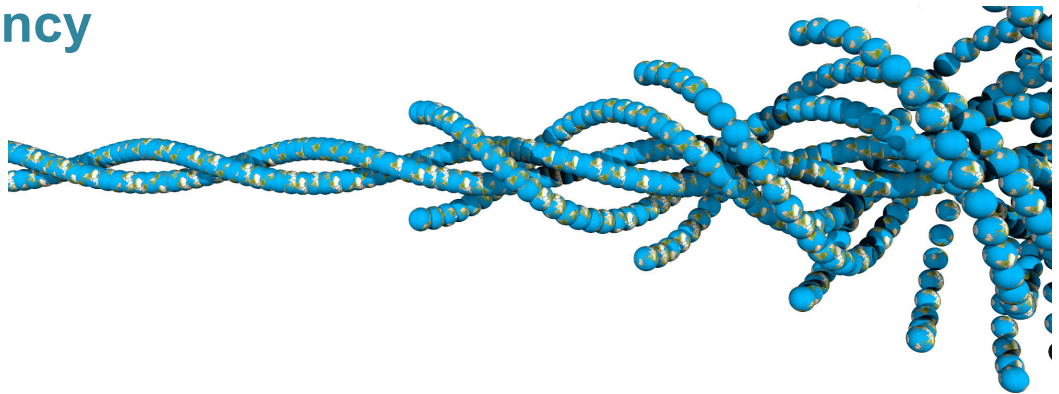
DSP Evolution



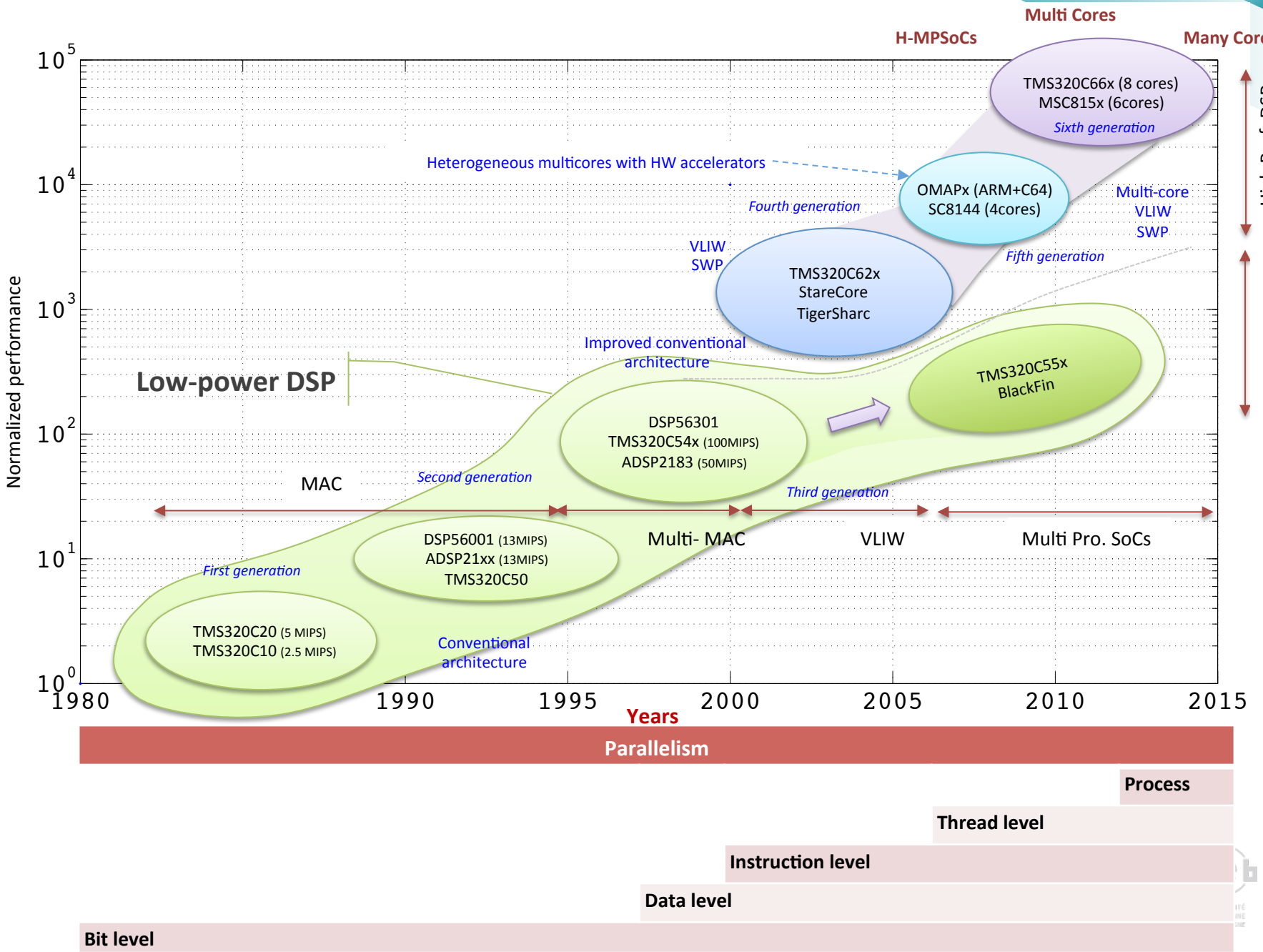
Part 2

Architecture of low Power DSP

- Requirements
- Processing unit
- Memory unit
- Control unit
- Compiler inefficiency



DSP Evolution



Inefficiency of classical RISC processors

FIR filter on a ultra-low power RISC : MSP430

```
1.  LOOP:   MOV.W    @r10,r13      /* Read of sample z[i] */
2.          MOV.W    @r11,r12      /* Read of coefficient c[i] */
3.
4.          CALL     #__mpyi       /* Multiplication */
5.          ADD.W    r12,r9        /* Accumulation  */
6.
7.          MOV.W    r4,0(r15)     /* Shift data in the z buf */
8.
9.          SUB.W    #2,r11        /* Pointer management */
10.         SUB.W    #2,r10
11.         SUB.W    #2,r15
12.
13.         SUB.W    #1,r5         /* Loop management */
14.         JNE      #LOOP
```

Source operand loading

MAC operation

Sample delay

Address management

Loop management

- Around 30 cycles per tap

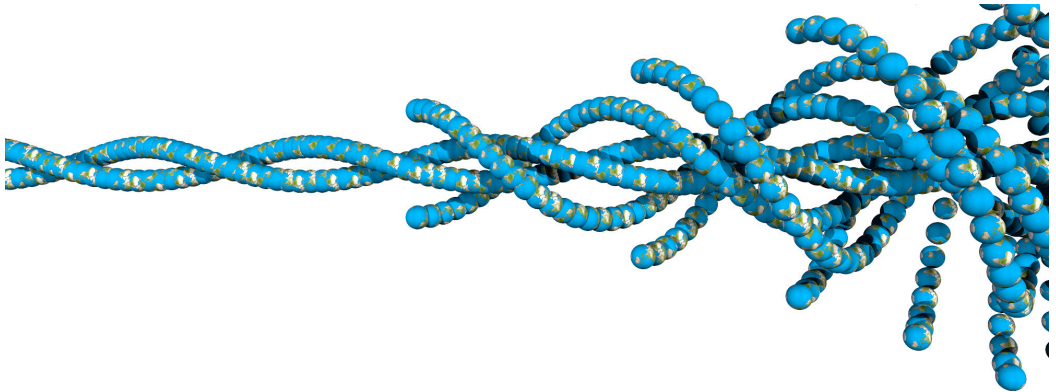
Inefficiency of classical RISC processors

- **Problems with RISC for DSP**
  - Processing Unit
    - Long execution time for multiplication if software emulation of arithmetic operation
  - Memory Unit
    - Memory bandwidth
    - Management of address pointers
    - Management of delays in z buffer
  - Control unit
    - Management of loops

Part 2

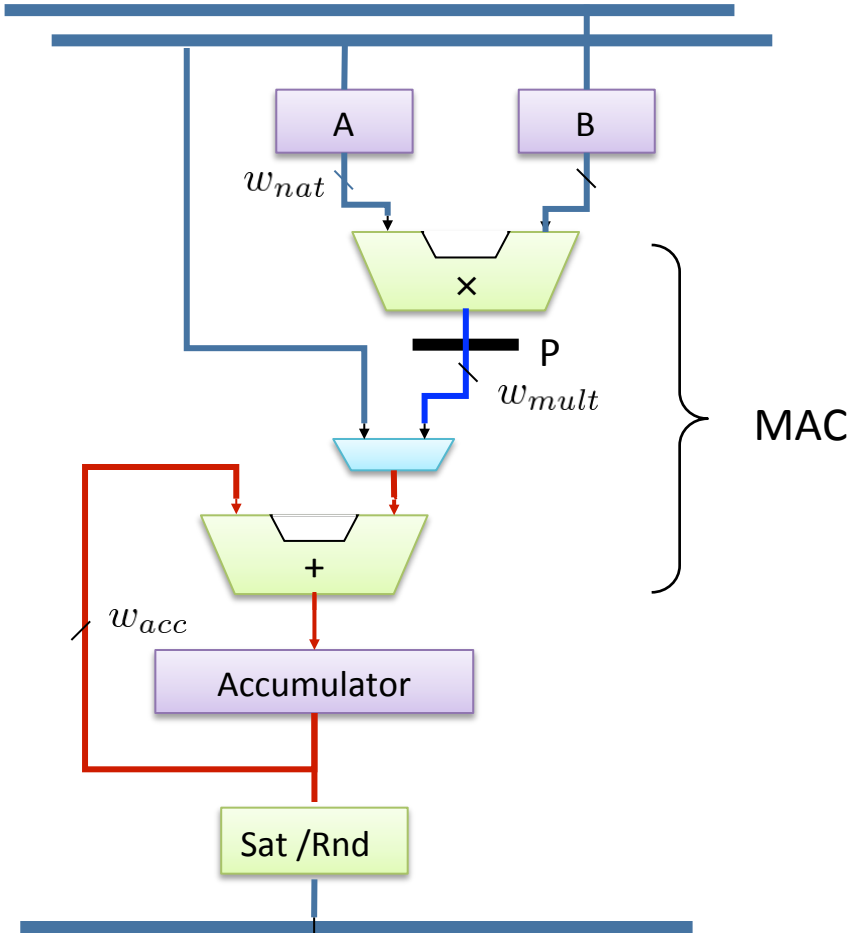
Architecture of low Power DSP

- Requirements
- **Processing unit**
- Memory unit
- Control unit



Specialized data-path for DSP

- **MAC based architecture**
  - Specialized operator-register interconnection
    - Heterogeneous structure
    - Good performances for area and energy consumption

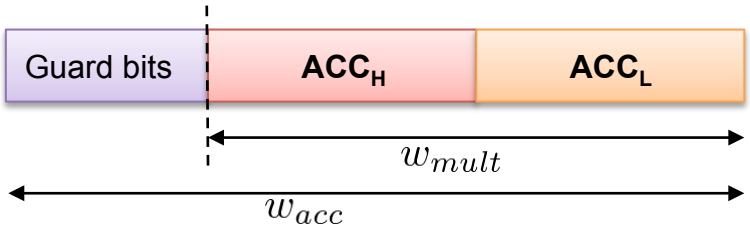


Elements of the processing unit

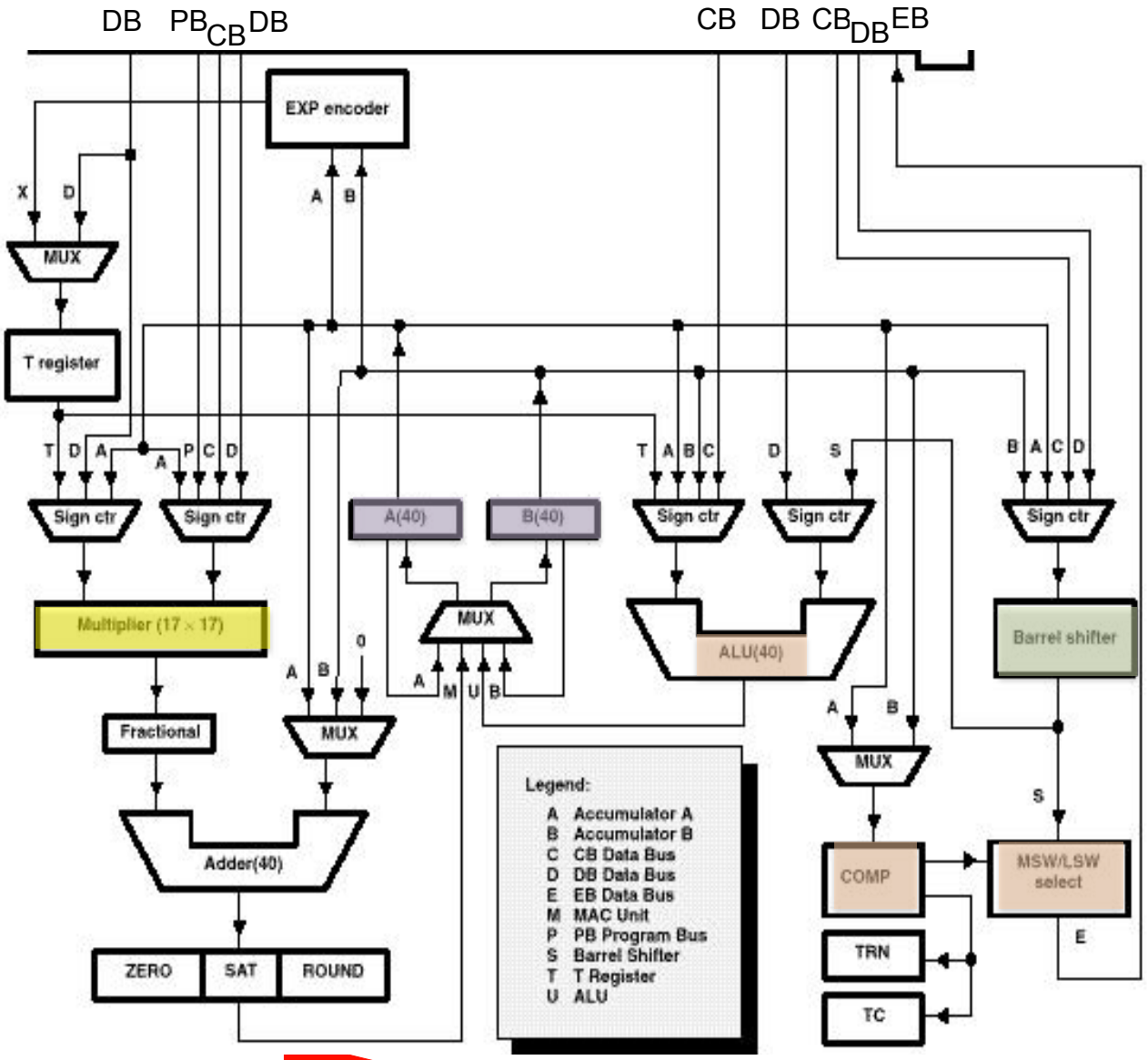
- **Operators**
  - HW multiplier
    - Fast operation
      - ✓ Latency : 1 to several cycles
      - ✓ Throughput : 1 cycle with pipelined execution
  - Specific adder
    - Independent of the ALU
  - Shifter
    - Specialized shifter
    - Barrel shifter : any shift in 1 cycle
  - Arithmetic specific units
    - Rounding
    - Saturation

Elements of the processing unit

- **Operators**
  - Application specific units
    - HW acceleration of specific DSP kernels
- **Storage units**
  - Dedicated registers
    - Operand registers
      - ✓ Storage of source operands or intermediate results
    - Accumulation registers
      - ✓ Addition result
        - ✓ Double precision + guard bits
      - ✓ Limited number of registers : 1 to 4



TMS320C54x



- 1 16*16 bit-multiplier
  - Op source 1 : T register
  - Op source 2 : memory
- 1 40 bit-adder
- 1 ALU (40 bits)
- 2 accumulation registers (40 bits)
- 1 barrel shifter
- 1 unit for Viterbi decoder

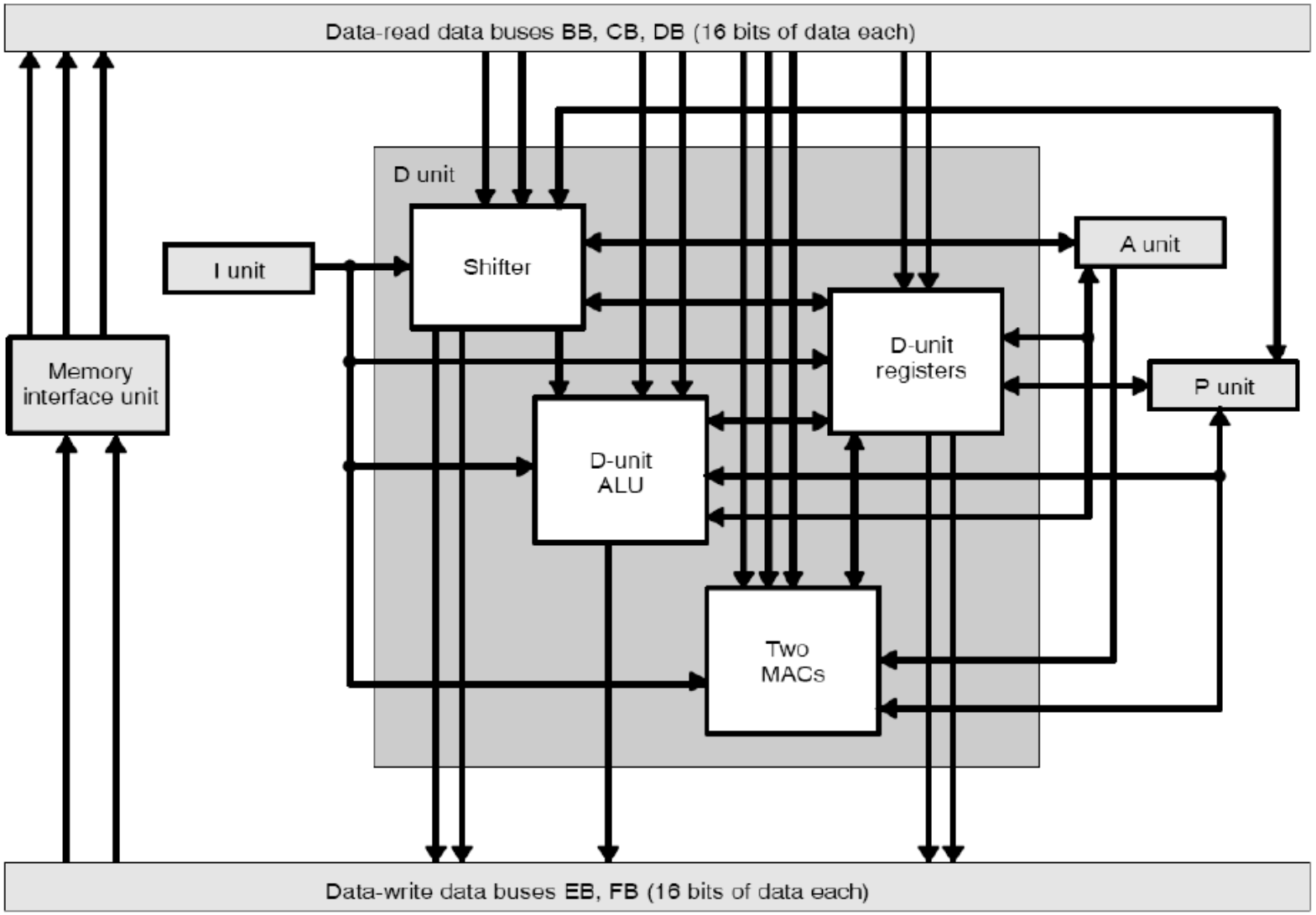
Specialized instructions set C54x

- **Filter instructions**
  - FIRS: Symmetric FIR filter
  - LMS: Adaptive filtering
- **Mathematic instructions**
  - Scalar arithmetic
    - ABS: Absolute value
    - SQUR: Square
    - POLY: Polynomial evaluation
  - Vector arithmetic acceleration
    - ABDIST: Absolute difference of vectors
    - SQDIST: Squared distance between vectors
    - SQURA: Sum of squares of vector elements
    - SQURS: Difference of squares of vector elements

Specialized instructions set C54x

- **Application specific instructions**
  - Code book search
    - STRCD, SACCD, SRCCD
  - Viterbi
    - DADST, DSADT, CMPS

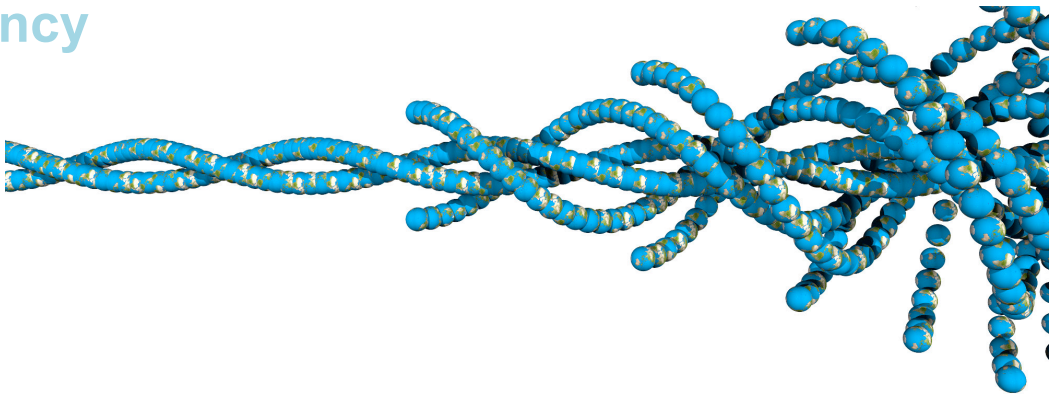
TMS320C55x


Data-read data buses BB, CB, DB (16 bits of data each)
D unit
I unit
Shifter
A unit
Memory interface unit
D-unit registers
P unit
D-unit ALU
Two MACs
Data-write data buses EB, FB (16 bits of data each)

Part 2

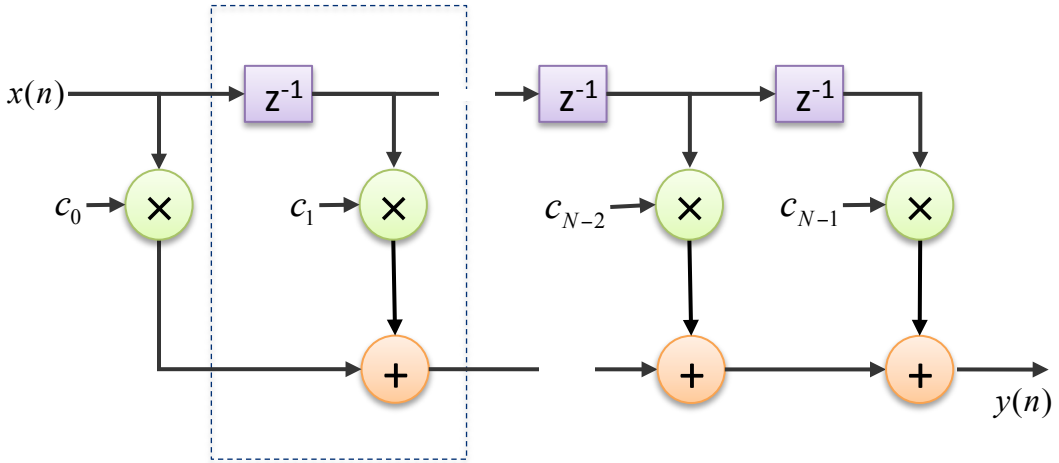
Architecture of low Power DSP

- Requirements
- Processing unit
- **Memory unit**
- Control unit
- Compiler inefficiency



Requirements

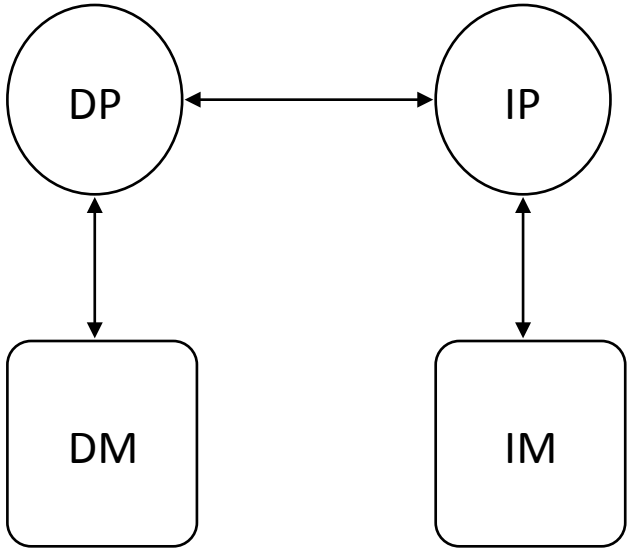
- **Memory accesses for one tap**
  - Instruction fetch
  - Data read x_{n-k}
  - Coefficient read h_k
  - Data shift $x_{n-k-1} = x_{n-k}$



One FIR tap

Harvard architecture

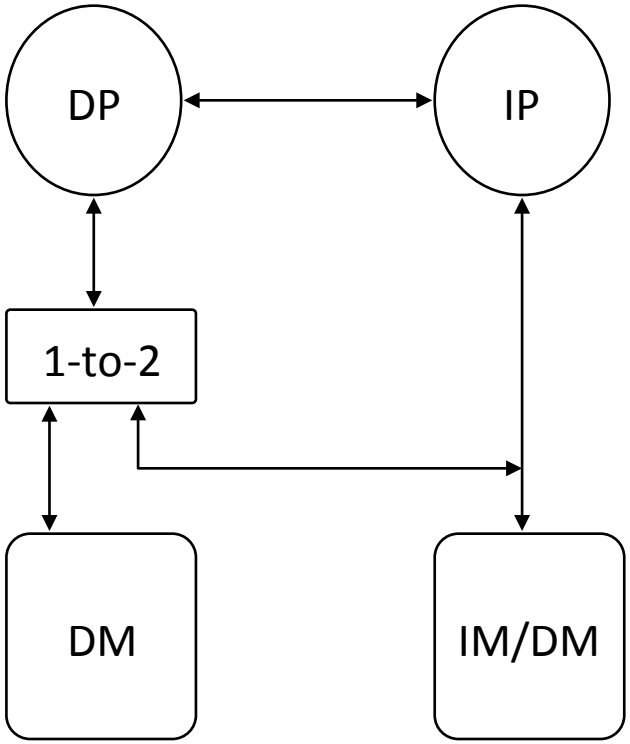
- **Principles**
  - Split of instruction and data memory
    - *Instruction Fetch* pipelined with *Operand Fetch*



Ex: TMS320C10

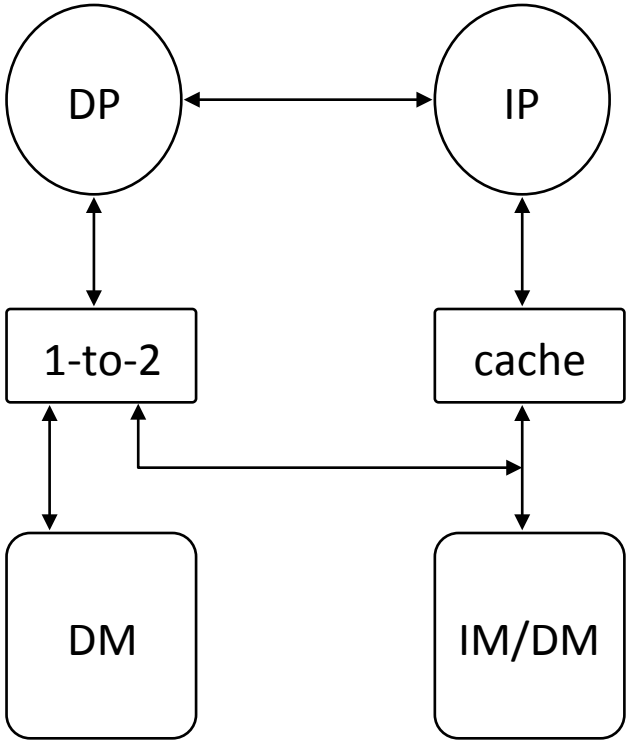
Harvard architecture

- **First modification**
  - Coefficient storage in Instruction Processing Unit
  - Fetch of the coefficient and the data in parallel
  - Example :
    - AT&T DSP32
    - DSP32C



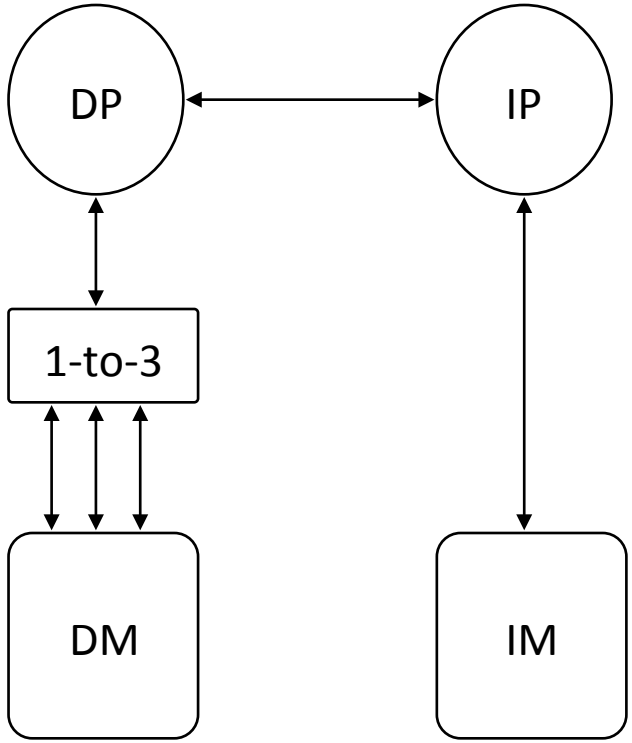
Harvard architecture

- **Second modification**
  - Cache for frequently used instructions in loop
  - Examples
    - TMS320C25: 1 instruction (loop)
    - DSP16: 15 instructions
    - ADSP-2100: 16 instructions



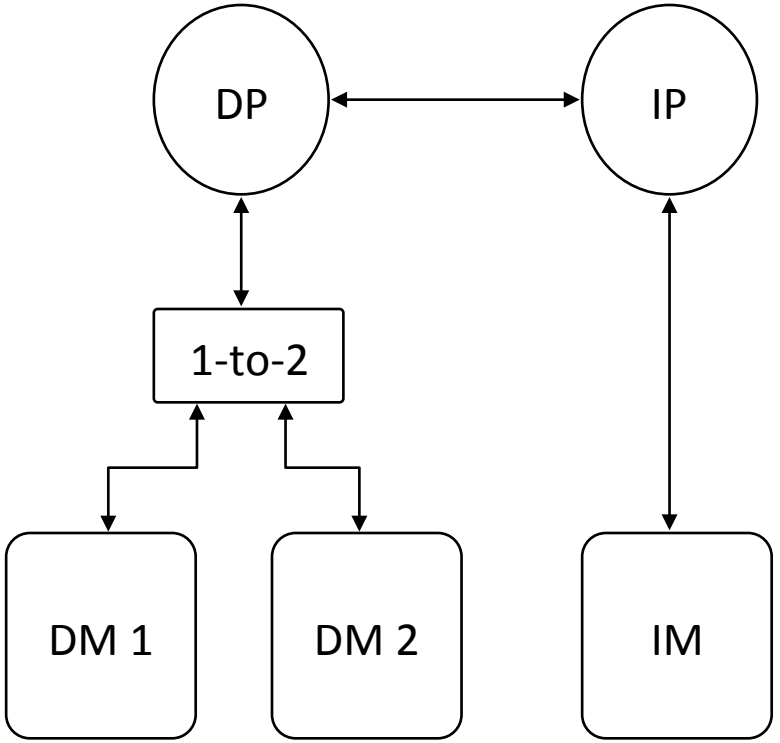
Harvard architecture

- **Third modification**
  - Memory with several ports
    - Read of several data in parallel
    - Internal memory
  - Examples
    - Fujitsu MB86232 (3 ports)



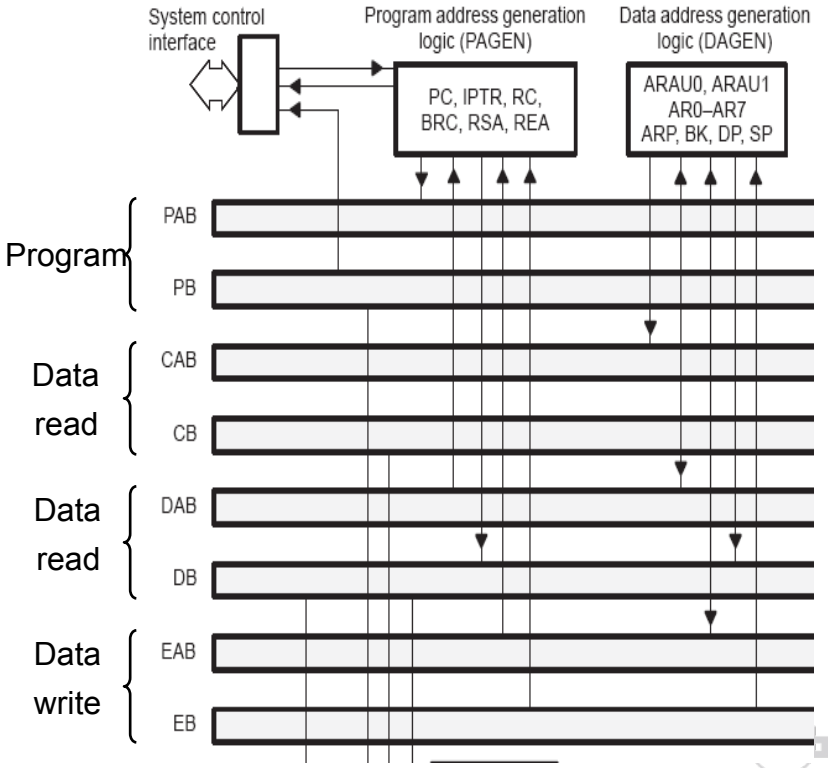
Harvard architecture

- **Fourth modification**
  - Memory with several banks
    - Read of several data in parallel
  - Examples
    - Motorola DSP 56001 et 96002
    - TMS320C30 et C40



Example of TMS320C54x

- **Internal memory**
  - ROM :
    - Bootloader
    - Used in data or program space
  - DARAM (Dual Acces RAM)
    - 1 read + 1 write per cycle
  - SARAM Single Acces
    - 1 read or 1 write per cycle



Specific addressing modes for DSP

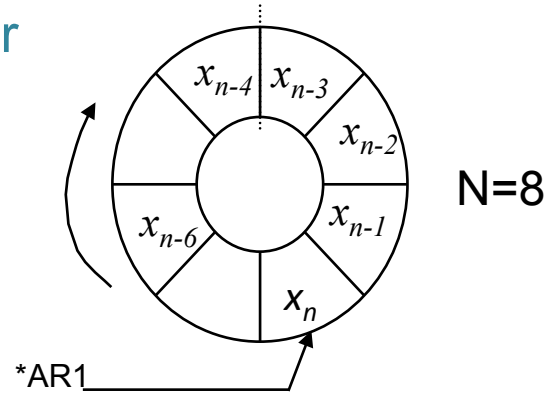
- **Indirect register addressing :**
  - Address register (AR) pointing on the data

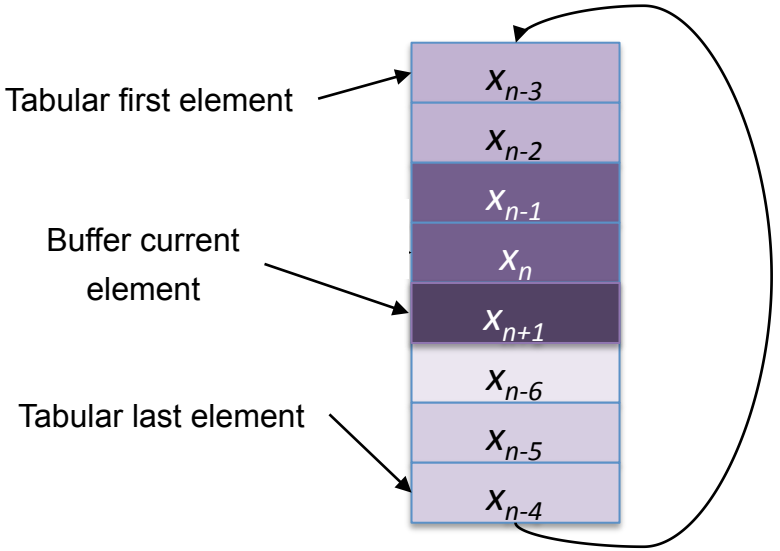
```
LD   *AR1, A // @ = AR1
```

  - Post modifications
    - Linear : AR:= AR ± 1 — `LD *AR1+,A` — `// @ = AR1, AR1 = AR1+1`
    - Indexed : AR:= AR ± MR — `LD *AR1+0,A` — `// @ = AR1, AR1 = AR1 + AR0`
      - ✓ MR: index register
    - Modulo : $(AR:= AR \pm 1)_N$ — `LD *AR1+%,A` — `// @ = AR1, AR1 = (AR1 + 1)%BK`
      - ✓ BK specify the size of the circular buffer
    - Bit-reverse : FFT — `LD *AR1+0B ,A` — `// @ = AR1 // AR1 = bitrev(AR1 + AR0)`
      - ✓ After access, AR0 is added to ARx with reverse carry (rc) propagation.

DSP specialized addressing mode

- **Circular buffer**
  - Automatic data shift in linear filter



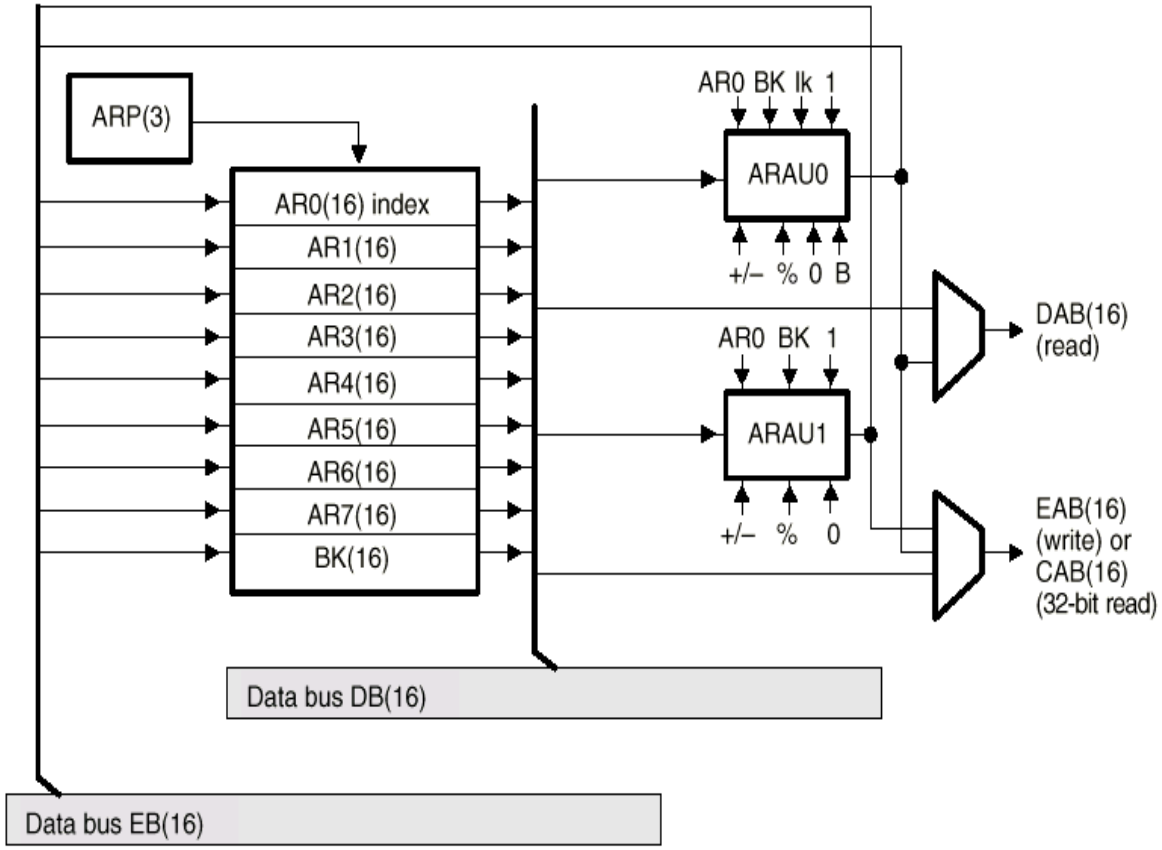


- **Bit-reversed addressing**
  - Addressing used for FFT
  - Start address xxxx00..00

Index	bit	bit reverse	index bit reverse
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Addressing unit

- **TMS320C54x**

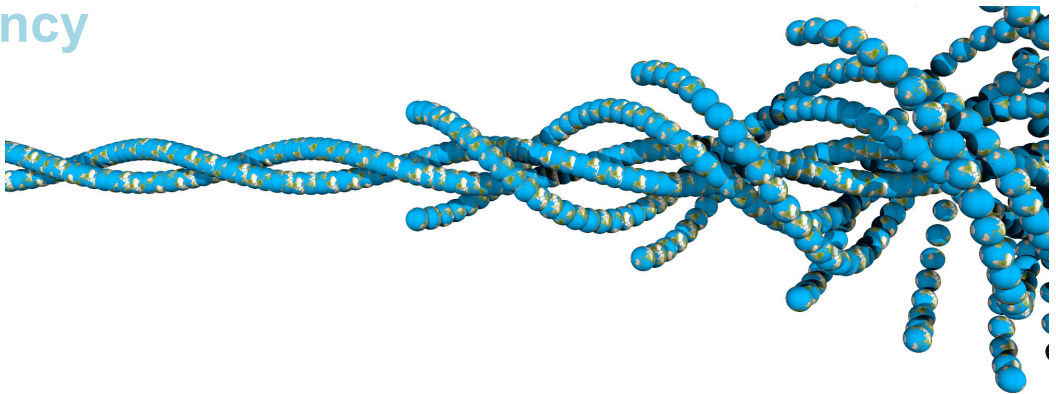


- 8 address registers (AR0…AR7)
- 2 processing units ARAU
- Specific registers
  - BK : size circular buffer
  - AR0 : index register

Part 2

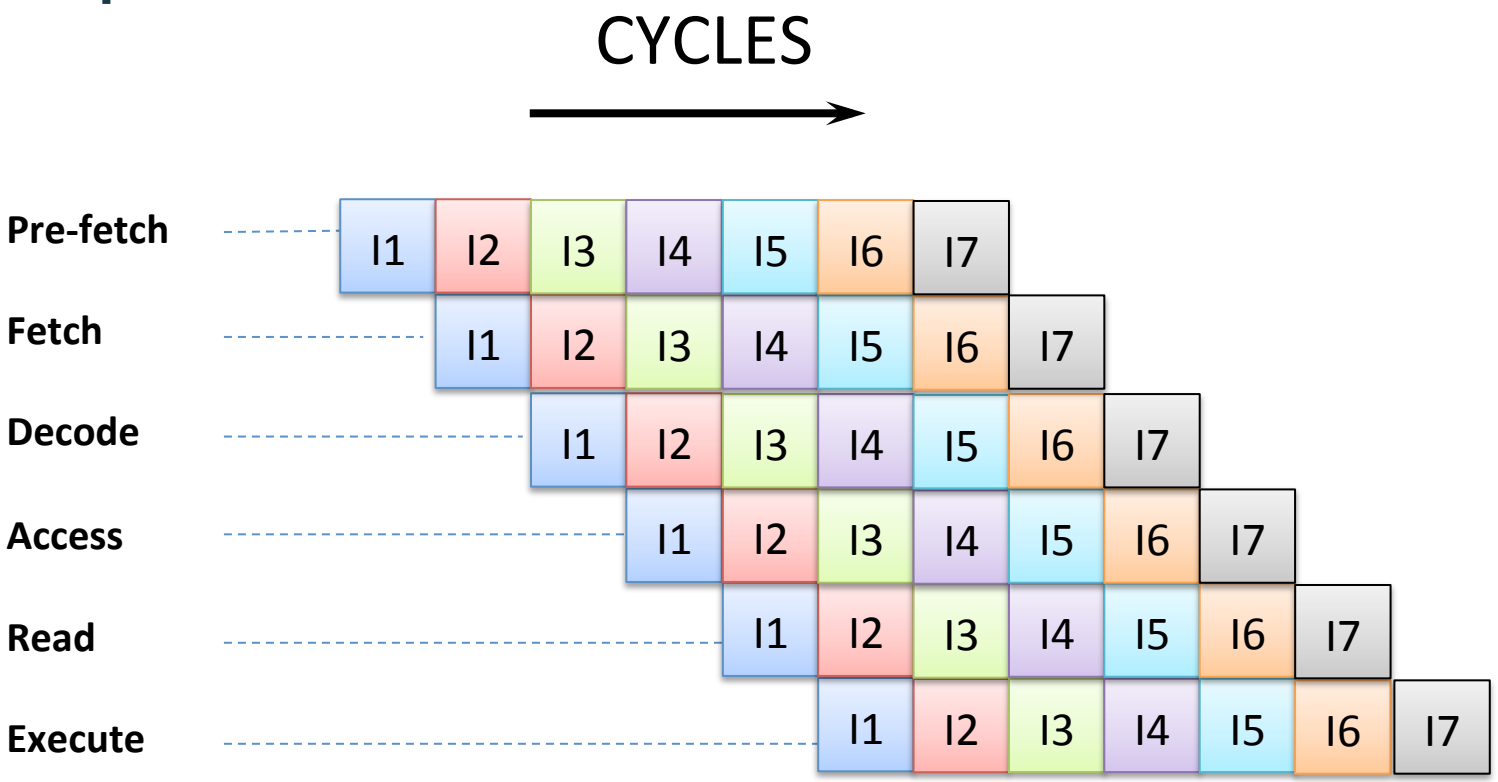
Architecture of low Power DSP

- Requirements
- Processing unit
- Memory unit
- **Control unit**
- Compiler inefficiency



Pipeline architecture

- **Pipeline for C54x**



Instruction set format

- **Trade-off**
  - Energy & area efficiency
    - Reduce the memory & bus size
      - ✓ Reduce the instruction word-length (size)
  - Performance
    - Instruction Level Parallelism
      - ✓ Increase the instruction word-length to encode several operations executed in parallel
- **Encoded Instruction set (16 or 32 bits)**
  - Restrictions leading to heterogeneous instruction set
    - Number of available operations
    - Addressing modes
    - Sources and destination operands
  - Use of mode bits

Control structures

- **HW loops**
  - Optimize the processing of small repetitive loops
    - Minimize the overhead due to loop management
      - ✓ HW initialization in 1 cycle
      - ✓ No supplementary instruction for loop management

  - Single instruction loop

SW loop

```
        MOVE    #16,B
LOOP    MAC     (R0)+,(R4)+,A
        DEC     B
        JNE     LOOP
```

HW loop

```
RPT     #16
MAC     (R0)+,(R4)+,A
```

  - Multiple instruction loop

DSP 56000

```
        DO      #16,END
        ...     Loop body
END
```

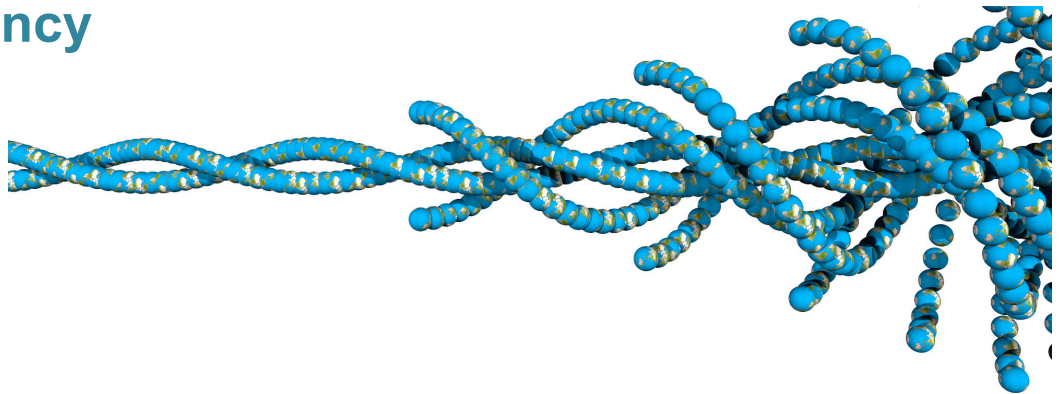
C50

```
        SPLK    #16
        RPTB    END-1
        ...     Loop body
END
```

Part 2

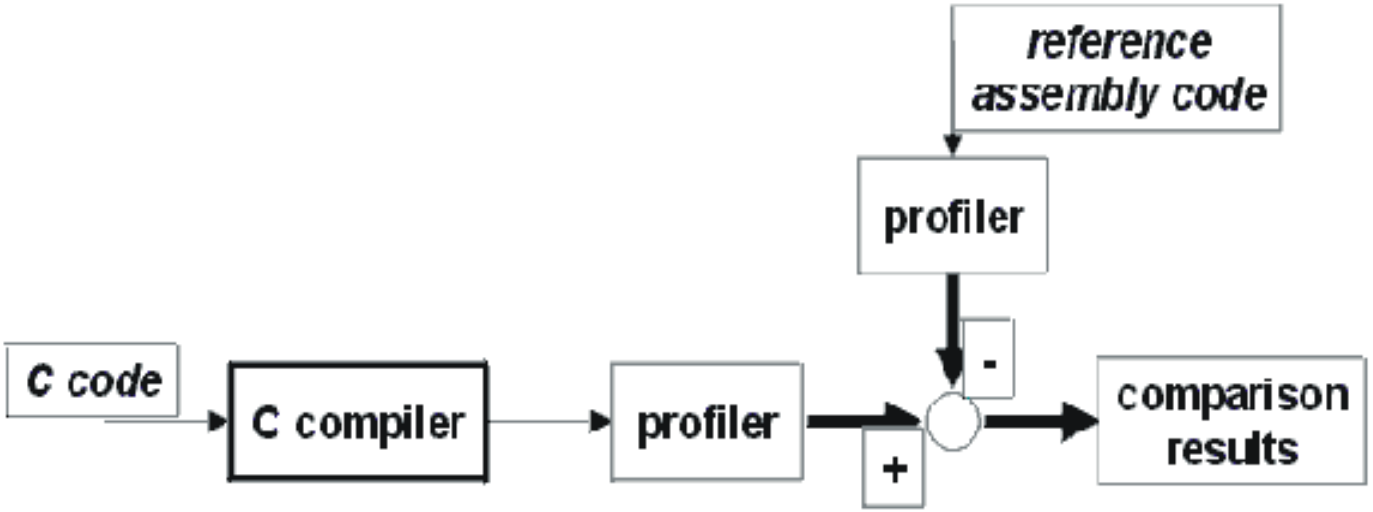
Architecture of low Power DSP

- Requirements
- Processing unit
- Memory unit
- Control unit
- **Compiler inefficiency**



DSPstone : compiler performances

- **Analysis of compiler quality**



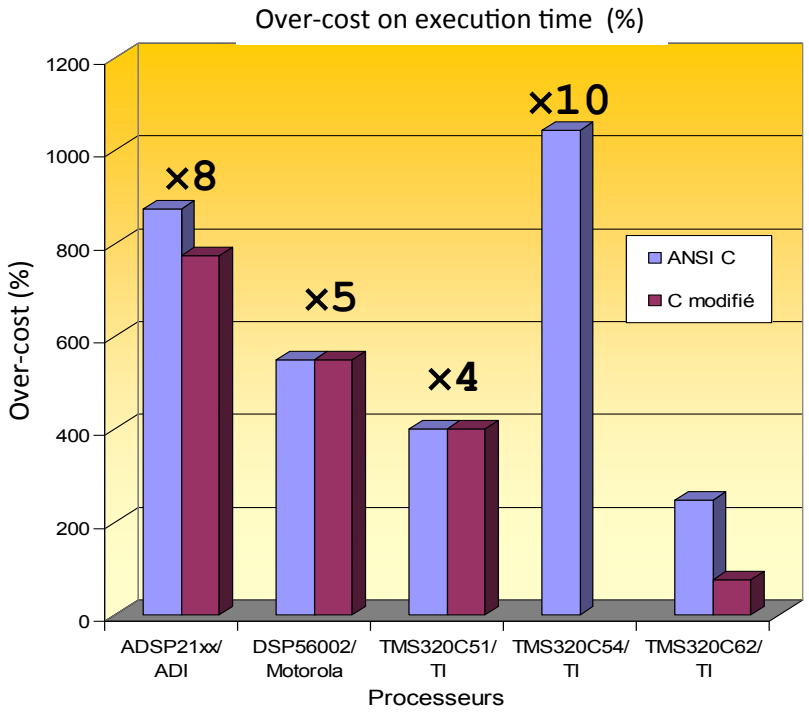
DSPstone : compiler performances

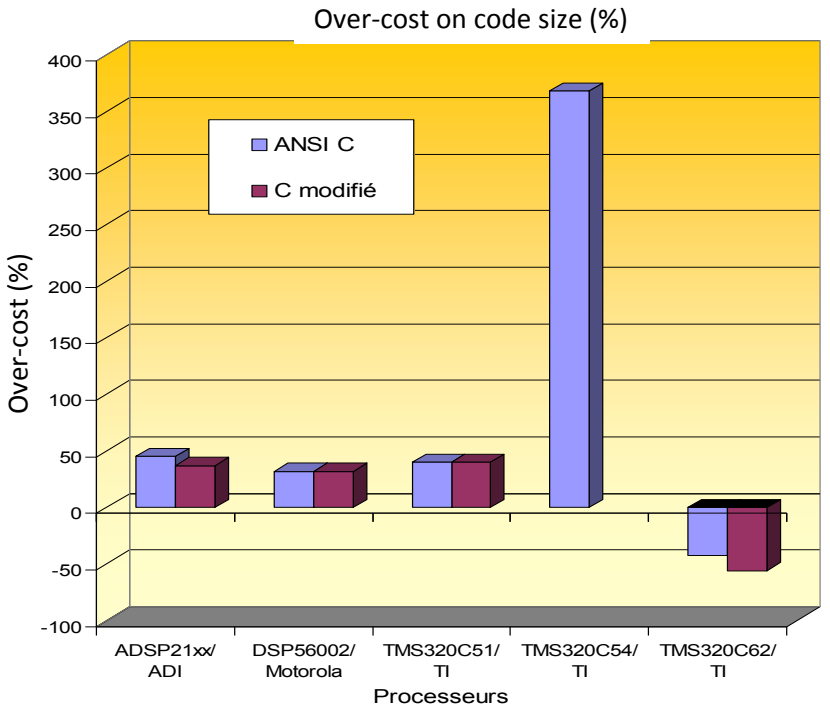
- **Results for TI C54x**

Benchmark	Δc_g [%]	Δp_g [%]	Δd_g [%]	Δm_g [%]
real_update	-25	1455	850	1269
n_real_update	373	1306	53	303
complex_update	22	766	425	661
n_complex_update	137	882	26	183
dot_product	620	1309	680	1112
matrix_1x3	673	705	226	494
matrix	24	414	5	048
convolution	1076	1545	100	461
fir	1045	769	97	368
fir2dim	427	743	158	398
iir_one_biquad	0	544	366	497
iir_n_biquad	140	764	55	365
lms	406	-35	78	19

C compiler quality

- **Over-cost due to C compiler: DSPStone**



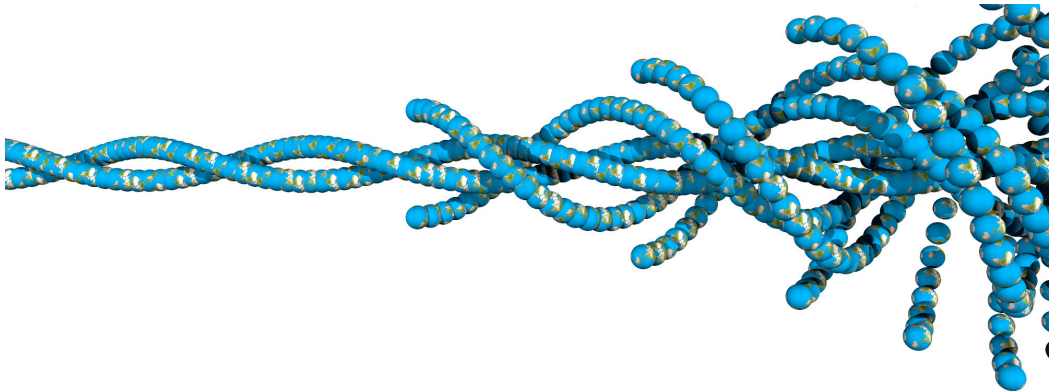


- Better performances for high performance DSP

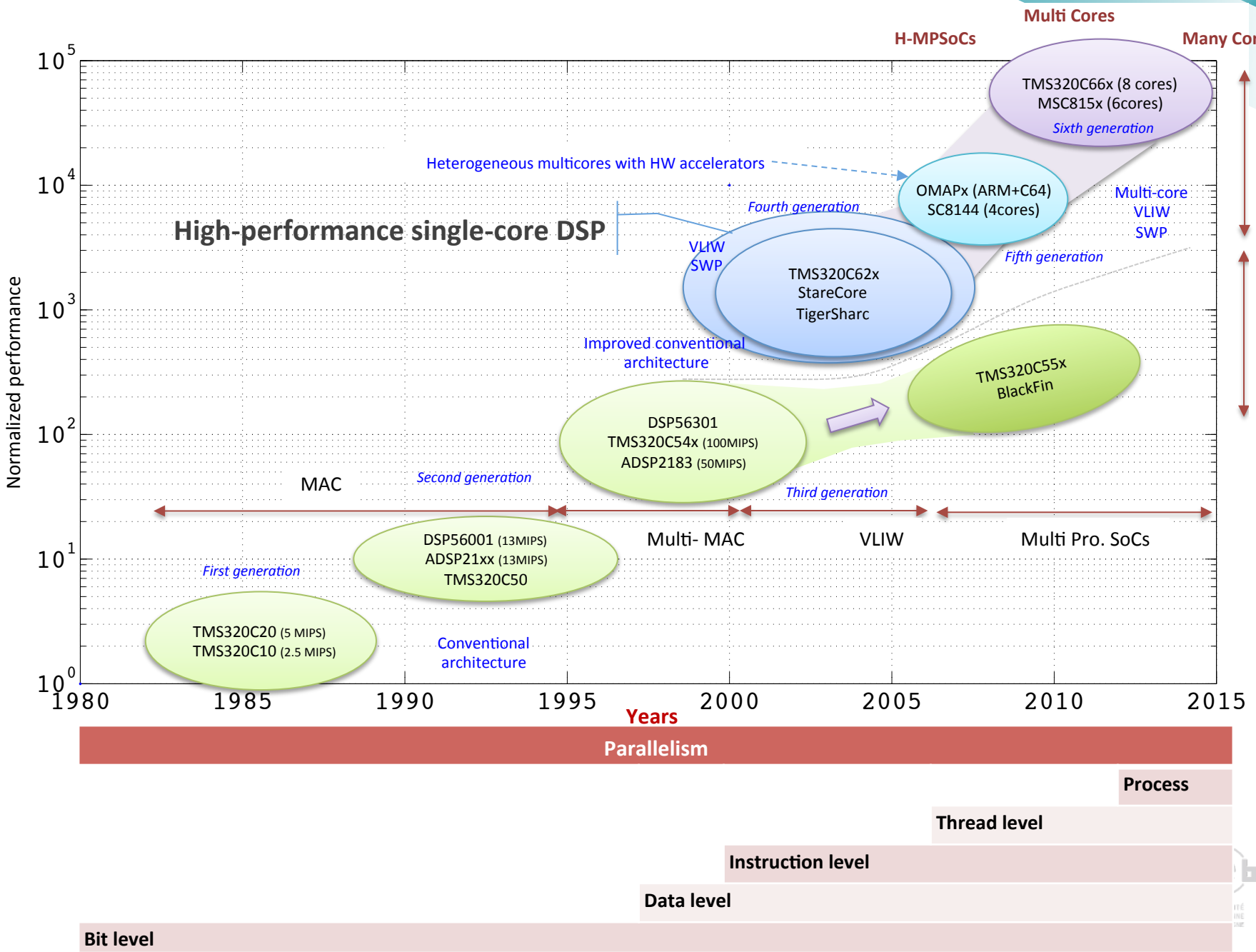
Part 3

Architecture of high performance single core DSP

- Data level parallelism
- Instruction level parallelism

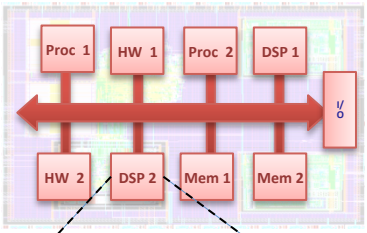


DSP Evolution



Levels of parallelism

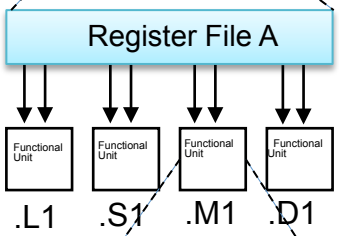
Many-core
(distributed memory)



Multi-core (MPSoc)
(shared memory)

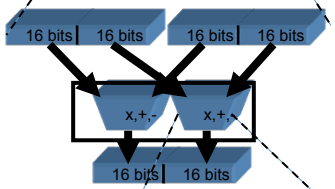
Process

Threads

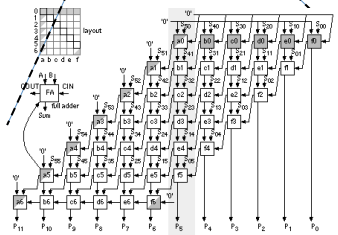


Instruction
(ILP)

Operation



Data (SWP)



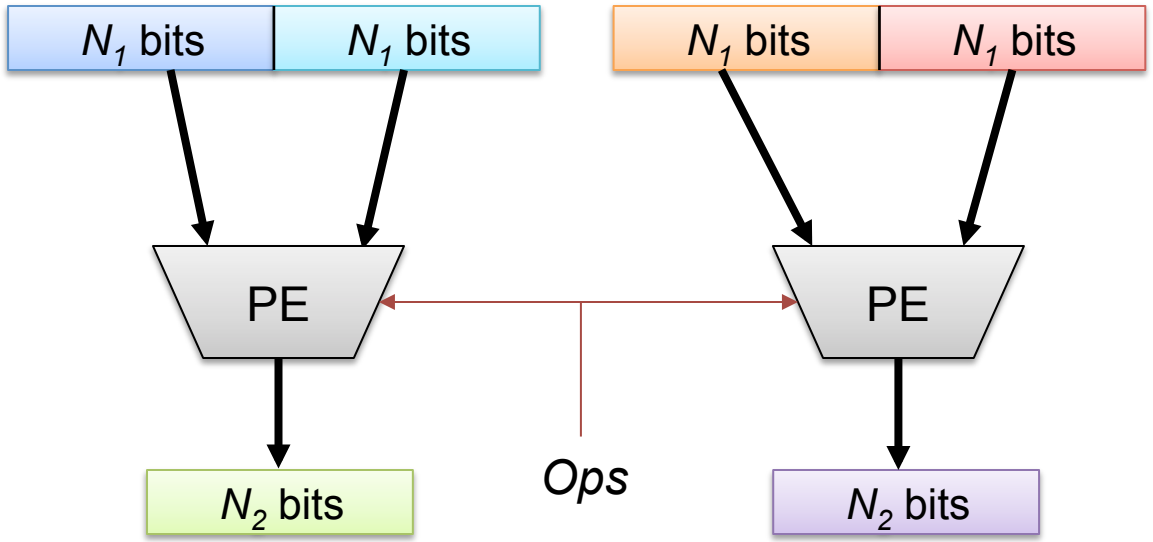
Bit

Bit (HW ops.)

Single Instruction Multiple Data SIMD

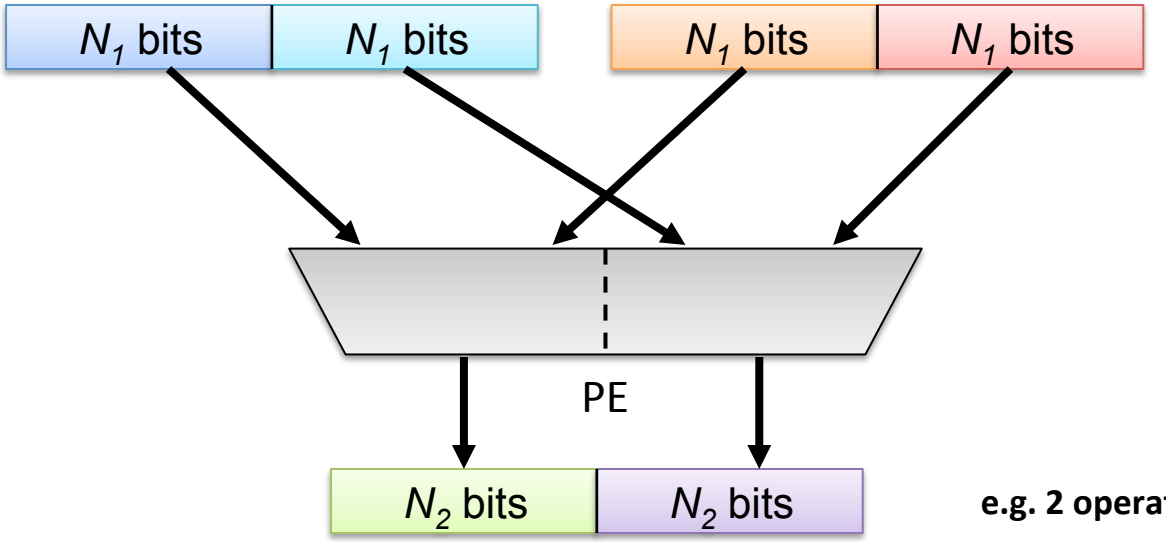
- **Same operation executed by several operators**
  - Simultaneous execution

- **Examples**
  - TI C66x, ADI TigerSHARC



Sub Word Parallelism SWP

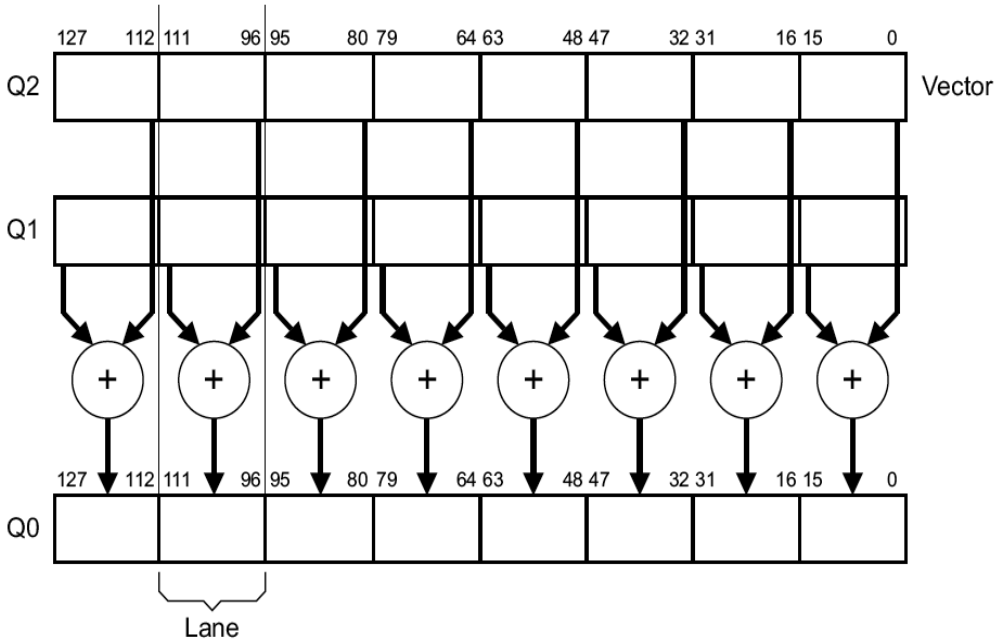
- **Parallel operations on *k* sub-words of *N/k* bits**
  - Split of an execution unit
  - Multiple executions on the same unit
- **Examples**
  - Lucent DSP16xxx, ADI ADSP-2116x, ADI TigerSHARC, TI C6x



e.g. 2 operations on 16 bits

ARM NEON

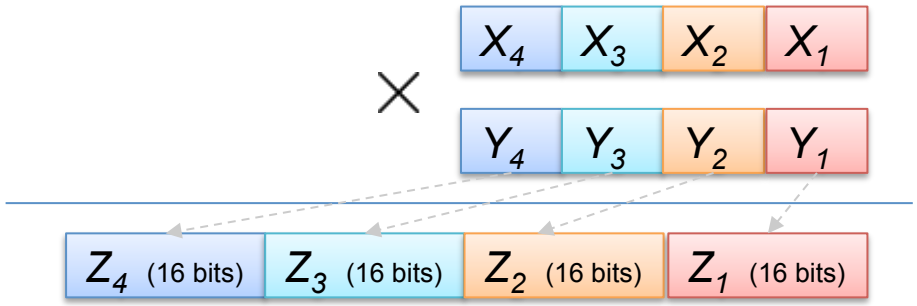
- **NEON Vector unit**
  - Supported data types
    - Signed and unsigned integers on 8-bits, 16-bits, 32-bits, and 64-bits
    - 32-bit single-precision floating point



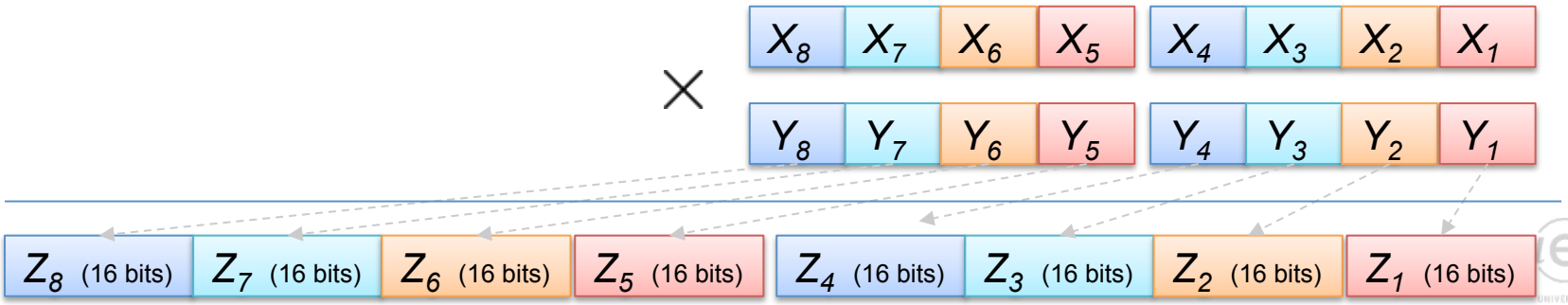
k	*N/k (bits)*
2	64
4	32
8	16
16	8

Texas Instrument C64x+

- **MPYSU4**
  - SWP 4 parallel MULTs $Z_i = Y_i \times X_i$

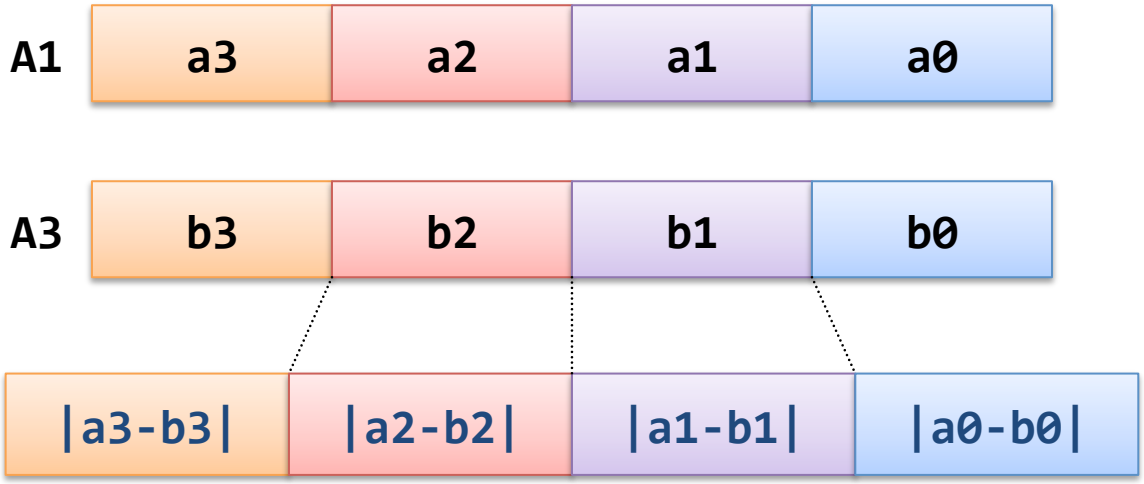


- **DMPYSU4**
  - SIMD 2 × 4 parallel MULTs



Texas Instrument C64x+

- **SUBABS4 :**
  - Absolute value of difference between the packed 8-bit datum in sources
  - Use in motion-estimation




```
LDW     *A0, A1

LDW     *A2, A3

SUBABS4 A1,A3,A5
```

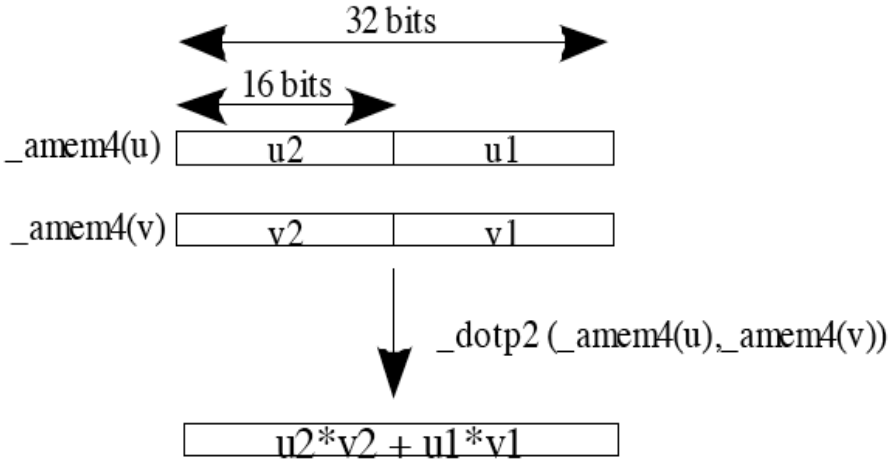
Product scalar example

- Initial C source code

```
for(n=0;n<N;n++)
{
    acc = acc+VectU[n]*VectV[n];
}
```

- SWP on 16-bit data, load of 32-bit data

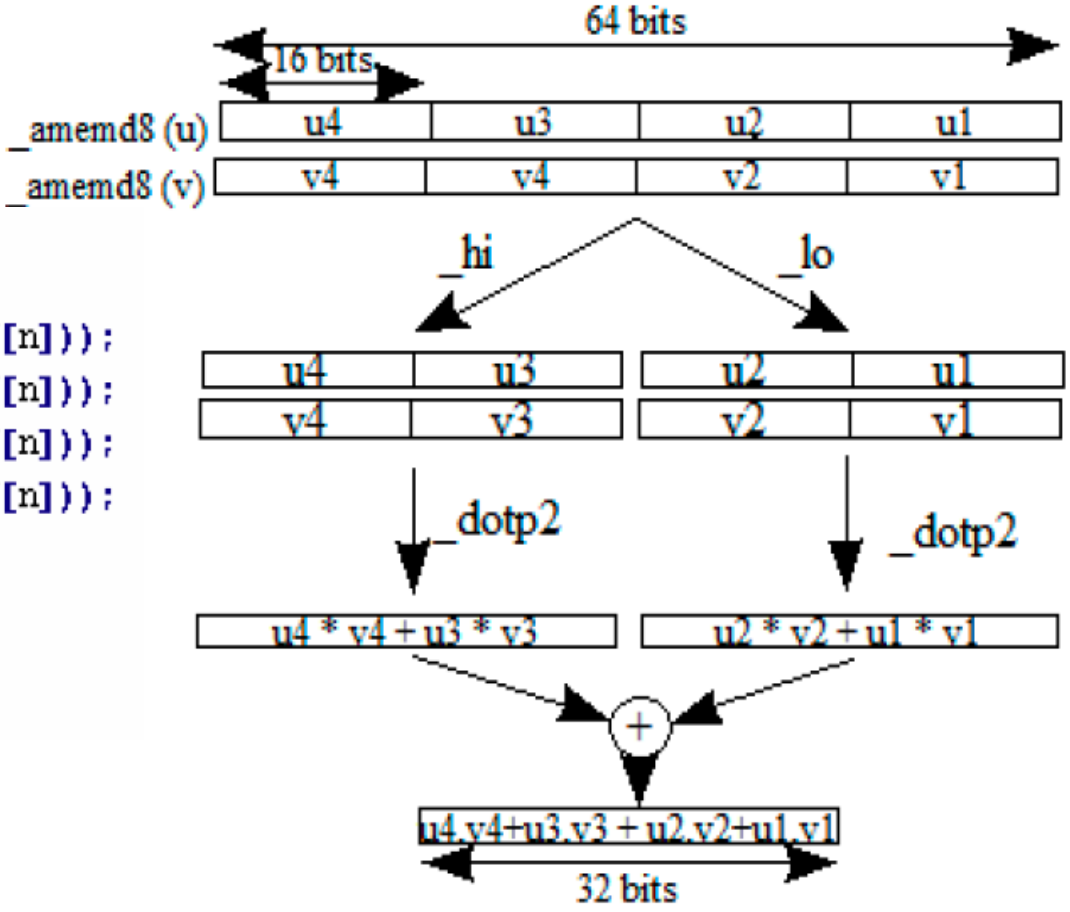
```
for(n=0;n<N;n=n+2)
{
    al_a0 = _amem4_const(&VectU[n]);
    bl_b0 = _amem4_const(&VectV[n]);
    acc += _dotp2(al_a0, bl_b0);
}
```



Product scalar example

- SWP on 16-bit data, load of 64-bit data

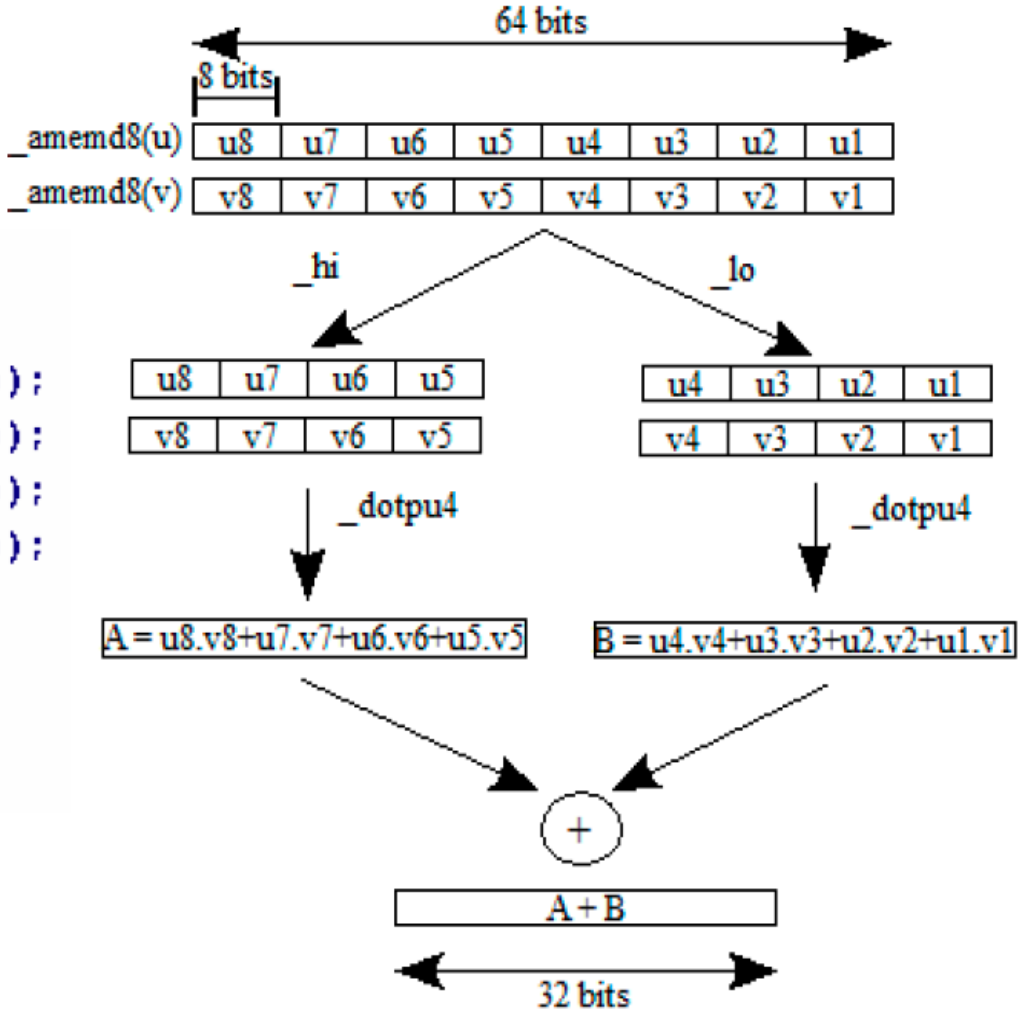
```
for(n=0;n<N;n=n+4)
{
    a3_a2 = _hi(_amemd8_const(&VectU[n]));
    a1_a0 = _lo(_amemd8_const(&VectU[n]));
    b3_b2 = _hi(_amemd8_const(&VectV[n]));
    b1_b0 = _lo(_amemd8_const(&VectV[n]));
    acc1 += _dotp2(a3_a2, b3_b2);
    acc2 += _dotp2(a1_a0, b1_b0);
}
acc=acc1+acc2;
```



Product scalar example

- SWP on 8-bit data, load of 64-bit data

```
for(n=0;n<N;n=n+8)
{
    a3_a2 = _hi(_amemd8_const(&VectU[n]));
    a1_a0 = _lo(_amemd8_const(&VectU[n]));
    b3_b2 = _hi(_amemd8_const(&VectV[n]));
    b1_b0 = _lo(_amemd8_const(&VectV[n]));
    acc1 += _dotpsu4(a3_a2, b3_b2);
    acc2 += _dotpsu4(a1_a0, b1_b0);
}
acc=acc1+acc2;
```



Scalar product

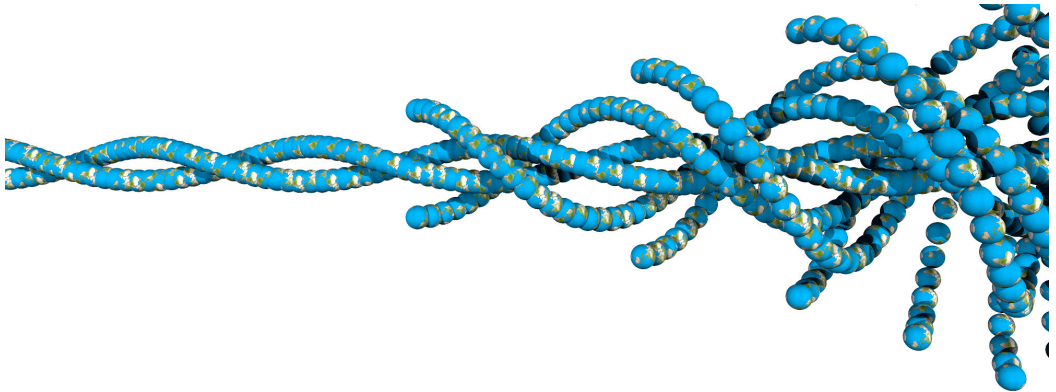
- **Results for IPC and execution time**

Options	T_{exec} (Cycles/sample)	IPC
none	39	0,76
-O0 (register)	26	0,79
-O1 (local)	23	0,81
-O2 (function)	0,5	7
-O3 (file)	0,5	7
SWP 16 bits	0,25	7
SWP 8 bits	0,125	7

Part 3

Architecture of high performance single core DSP

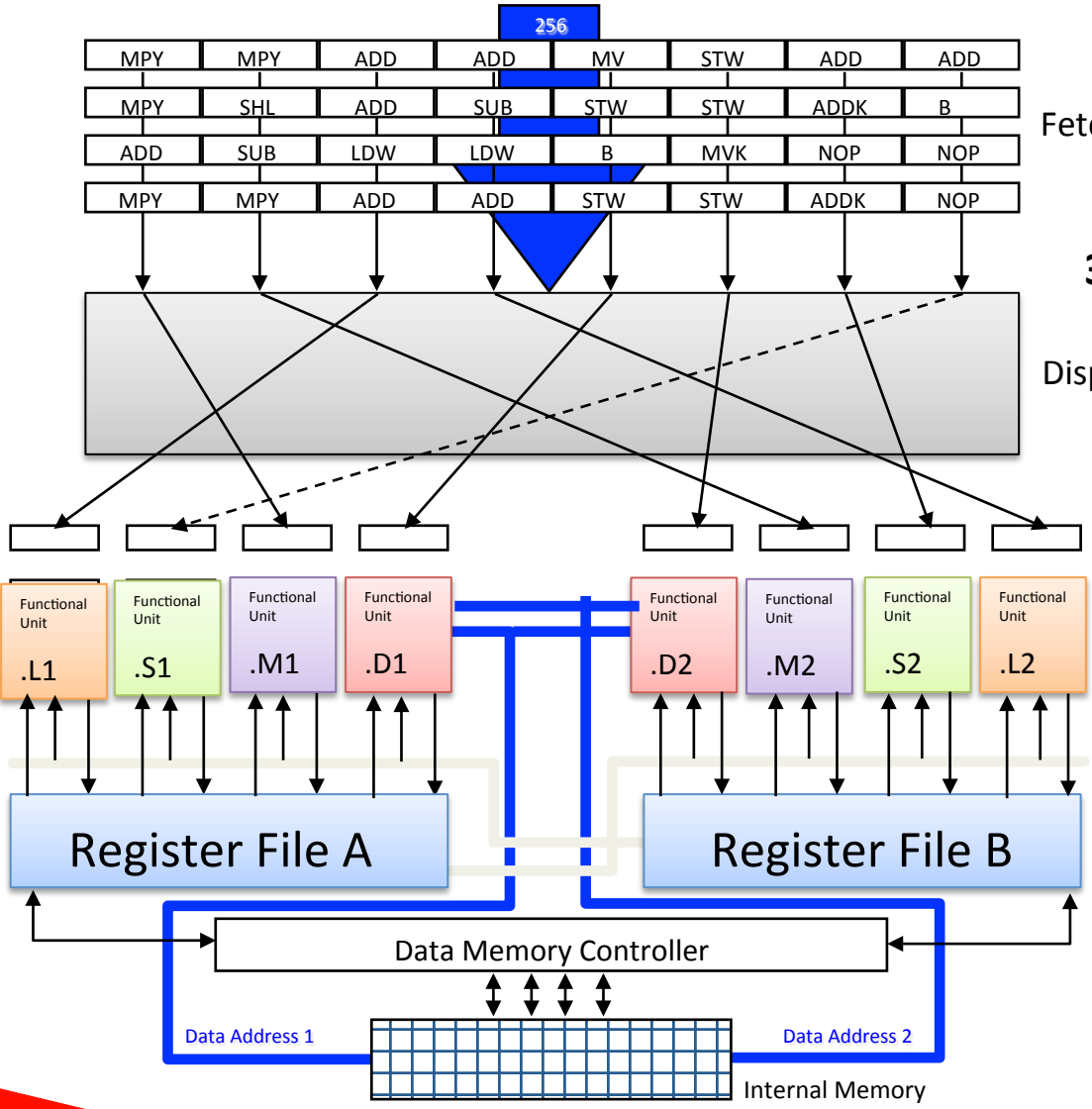
- Data level parallelism
- Instruction level parallelism



Very long instruction Word (VLIW)

- **VLIW characteristic**
  - Architecture made-up of parallel FUs
  - Several instructions per cycle
    - Embedded in a macro-instruction
  - Homogeneous architecture
    - More orthogonal, close to RISC processor
  - Uniform register file made-up of several registers
  - Load-Store architecture
- **Examples**
    - TI TMS320 C6xx
    - Infineon Carmel
    - ADI TigerSHARC
    - StarCore SC140 (Lucent + freescale)

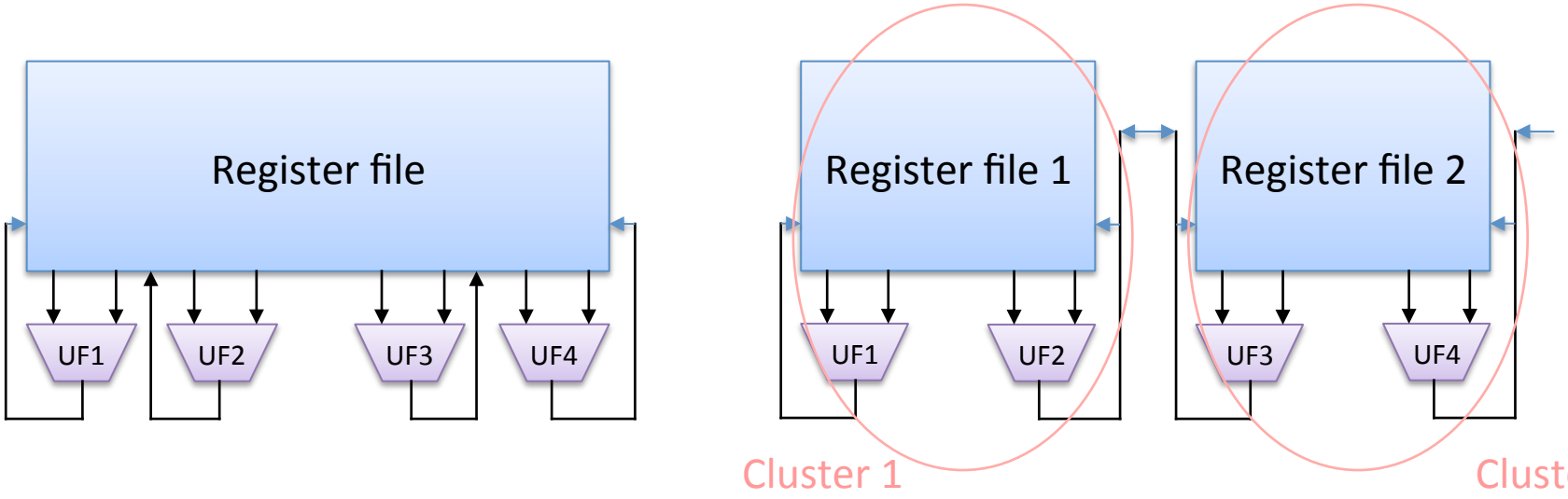
Texas Instrument TMS320C6000



L:ALU
S:Shift+ALU
M:Multplier
D:Address U

Cluster concept

- **Clustered architecture**
  - Collection of clusters executing in the same control step
- **Cluster = register file + functional unit**
  - Simple access to the cluster internal registers
  - Operations for data moving between clusters



Features for VLIW

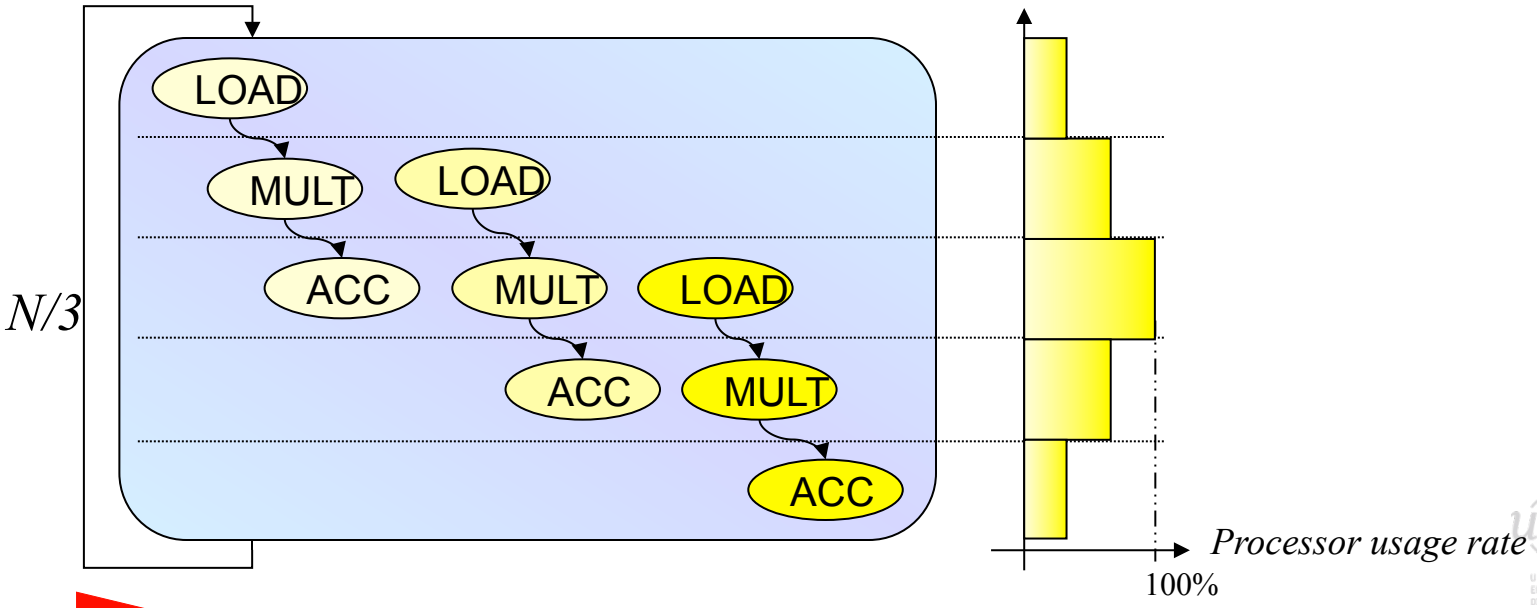
- **Very Long Instruction Word**
  - High performances
    - C6000 : up to 8 instructions per cycle
  - Homogeneous architecture
    - Suitable for compiler
  - Increase the cost and power vs. traditional DSP
    - C6000 ; macro instruction 8×32 = 256 bits
      - ✓ High WL for buses and memory

Loop unrolling

- **Scalar product**

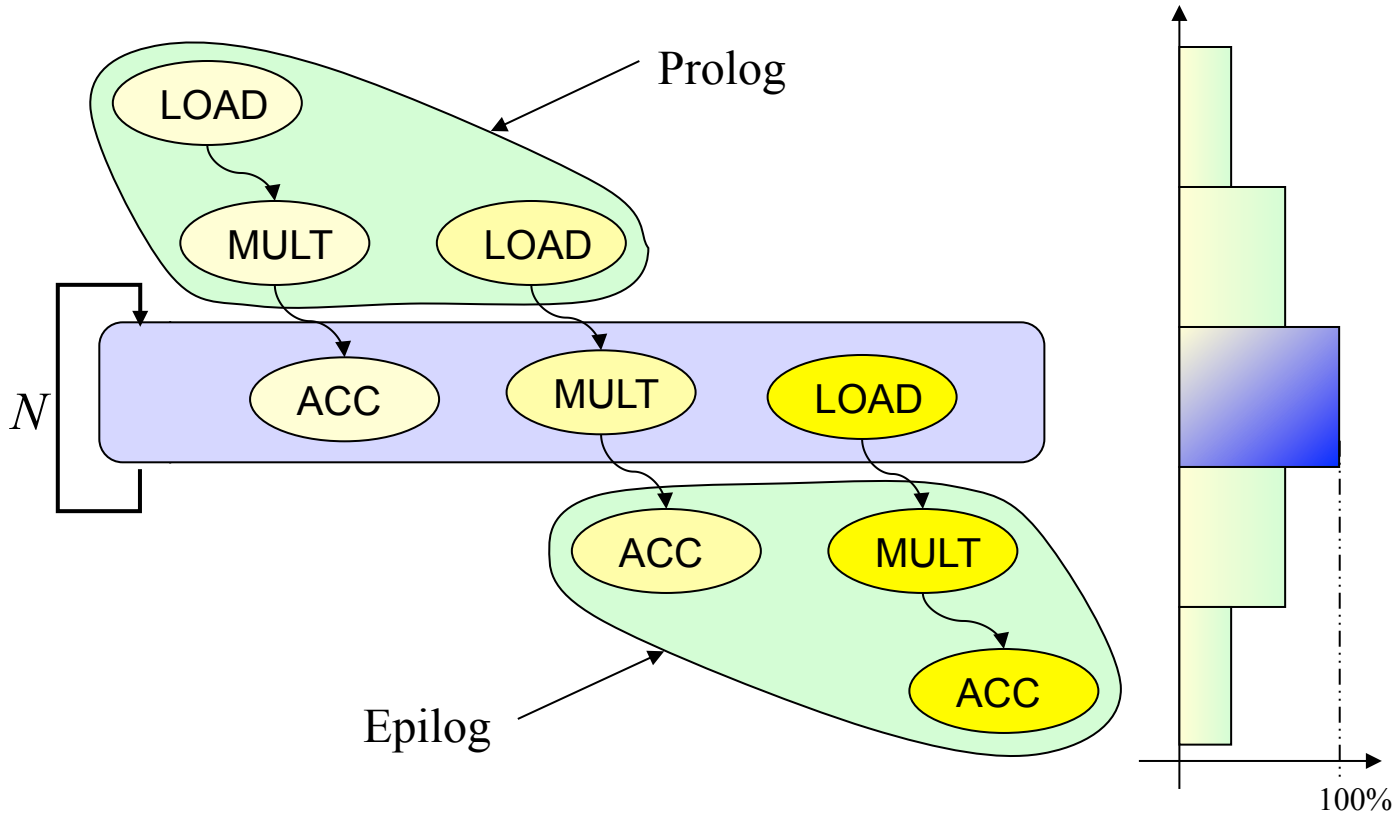
```
For(i=0;i<N;i++)
{
  ACC=ACC + x[i].h[i]
}
```

```
For(i=0;i<N;i+=3)
{
  ACC=ACC + x[i].h[i]
  ACC=ACC + x[i+1].h[i+1]
  ACC=ACC + x[i+2].h[i+2]
}
```

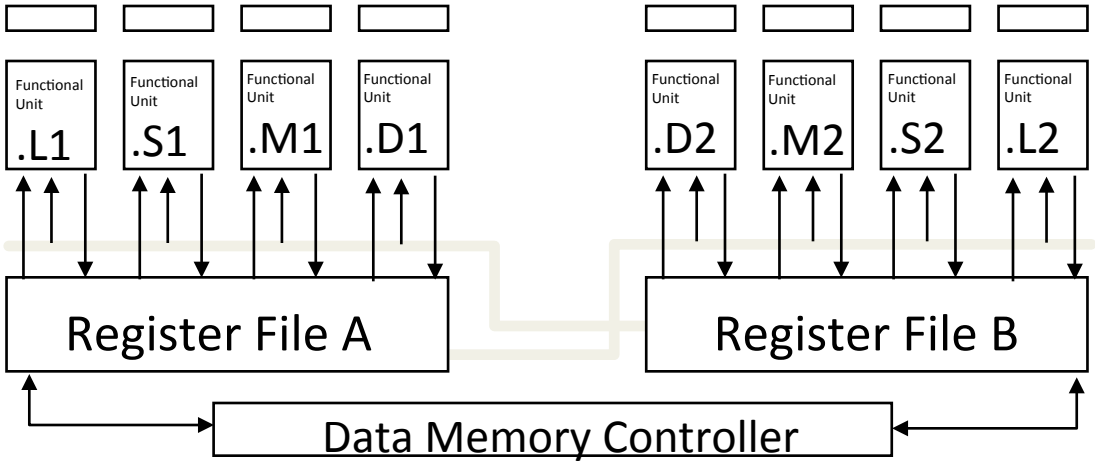


Software pipelining

- **Prolog – Loop kernel - Epilog**



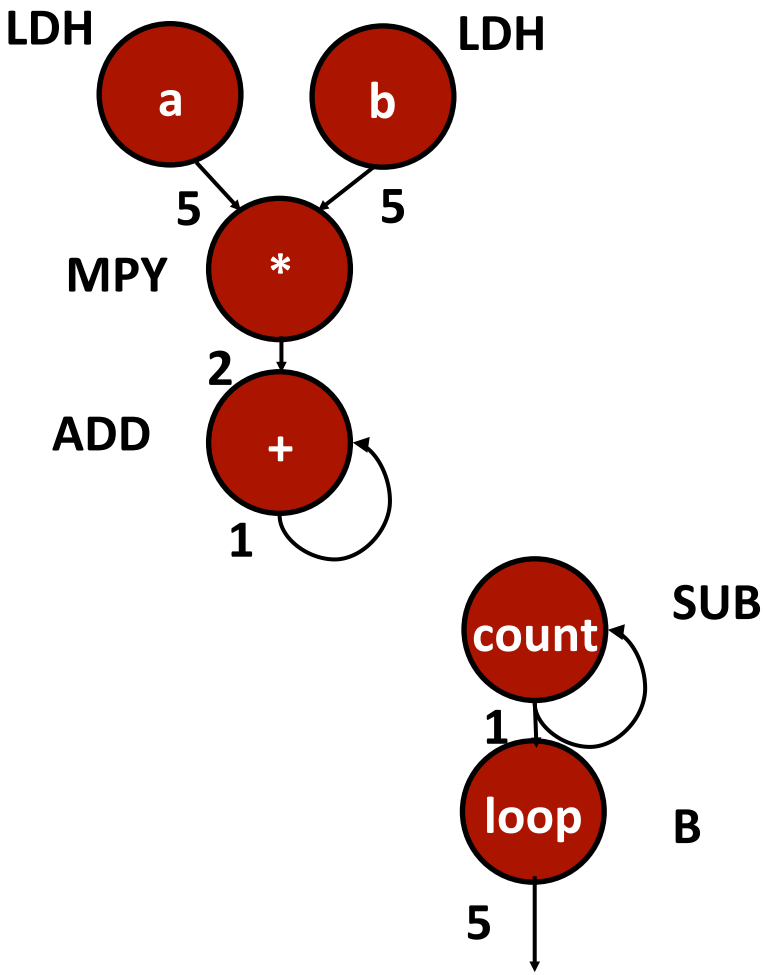
C6000 characteristics



Instruction	Description	Nombre de cycles	Unités .M	Unités .L	Unités .S	Unités .D
LDH	Charge une donnée 16 bits venant de la mémoire dans un registre	5	-	-	-	.D1 .D2
LDW	Charge deux données 16 bits consécutives venant de la mémoire	5	-	-	-	.D1 .D2
MPY	Multiplication entre 2 registres, résultat dans un troisième	2	.M1 .M2	-	-	-
ADD	Addition	1	-	.L1 .L2	.S1 .S2	.D1 .D2
SUB	Soustraction	1	-	.L1 .L2	.S1 .S2	.D1 .D2
B	Branchement	5	-	-	.S1 .S2	-
NOP	aucune opération	1	-	-	-	-

Data Flow Graph

- Scalar product



Case 1: No parallelism

Allocation table

Cycle	.D1	.D2	.L1	.L2	.M1	.M2	.S1	.S2	NOP
1	LDH								
2	LDH								
3									
4									
5									
6									
7					MPY				
8									
9			ADD						
10				SUB					
11							B		
12									
13									
14									
15									
16									

Execution time : *15.N* cycles

Code for VLIW DSP

- C code and assembly code

```
x[0] = input;                  // Update most recent sample.
acc = 0;                       // Zero accumulator.
for (i=0; i<8; i++)            // 8 taps
     {
          prod = (h[i]*x[i]);     // perform Q.15 multiplication
          acc = acc + prod;       // Update 32-bit accumulator
     }
output = (short) (acc>>15); // Cast output to 16 bits.
```

```
1 Start          MVKL .S2  8, B0          ; Intialize the loop counter (B0) to 8
2                MVKL .S1  0, A5          ; Intialize the accumulator (A5) to 0
3 Loop           LDH .D1 *A8++,A2 ; Load x(i) in  A2
4                LDH .D1 *A9++,A3 ; Load h(i) in A3
5                NOP 4                    ; LDH has a latency of 5 cycles
6                MPY .M1 A2,A3,A4 ; Multiply x(i) and h(i) in A4
7                NOP                      ; MPY has a latency of 2
8                ADD .L1 A4,A5,A5 ; Add A4 to the accumulator A5
9  [B0] SUB .L2  B0,1,B0 ; Sub 1 to the counter B0
10 [B0] B .S1  loop              ; Branch to loop if B0 <> 0
```

Case 2: Parallel instruction + re-sheduling

Allocation table

Cycle	.D1	.D2	.L1	.L2	.M1	.M2	.S1	.S2	NOP
1	LDH	LDH							
2				SUB					
3									
4							B		
5									
6					MPY				
7									
8			ADD						
9									
10									
11									
12									
13									
14									
15									
16									

Execution time : *8.N* cycles

Case 3: double word access

Allocation table

Cycle	.D1	.D2	.L1	.L2	.M1	.M2	.S1	.S2	NOP
1	LDW	LDW							
2									
3								SUB	
4							B		
5									
6					MPY	MPY			
7									
8			ADD	ADD					
9									
10									
11									
12									
13									
14									
15									
16									

Execution time : *4.N* cycles

Case 4: Loop unrolling

Allocation table

Cycle	.D1	.D2	.L1	.L2	.M1	.M2	.S1	.S2	NOP
1	LDW	LDW							
2	LDW	LDW							
3	LDW	LDW						SUB	
4	LDW	LDW					B	SUB	
5	LDW	LDW					B	SUB	
6	LDW	LDW			MPY	MPY	B	SUB	
7	LDW	LDW			MPY	MPY	B	SUB	
8	LDW	LDW	ADD	ADD	MPY	MPY	B	SUB	
9	LDW	LDW	ADD	ADD	MPY	MPY	B	SUB	
10			ADD	ADD	MPY	MPY	B	SUB	
11			ADD	ADD	MPY	MPY	B	SUB	
12			ADD	ADD	MPY	MPY	B		
13			ADD	ADD	MPY	MPY			
14			ADD	ADD	MPY	MPY			
15			ADD	ADD					
16			ADD	ADD					

Execution time : *k+N/2* cycles

Case 5: Software pipelining

Allocation table

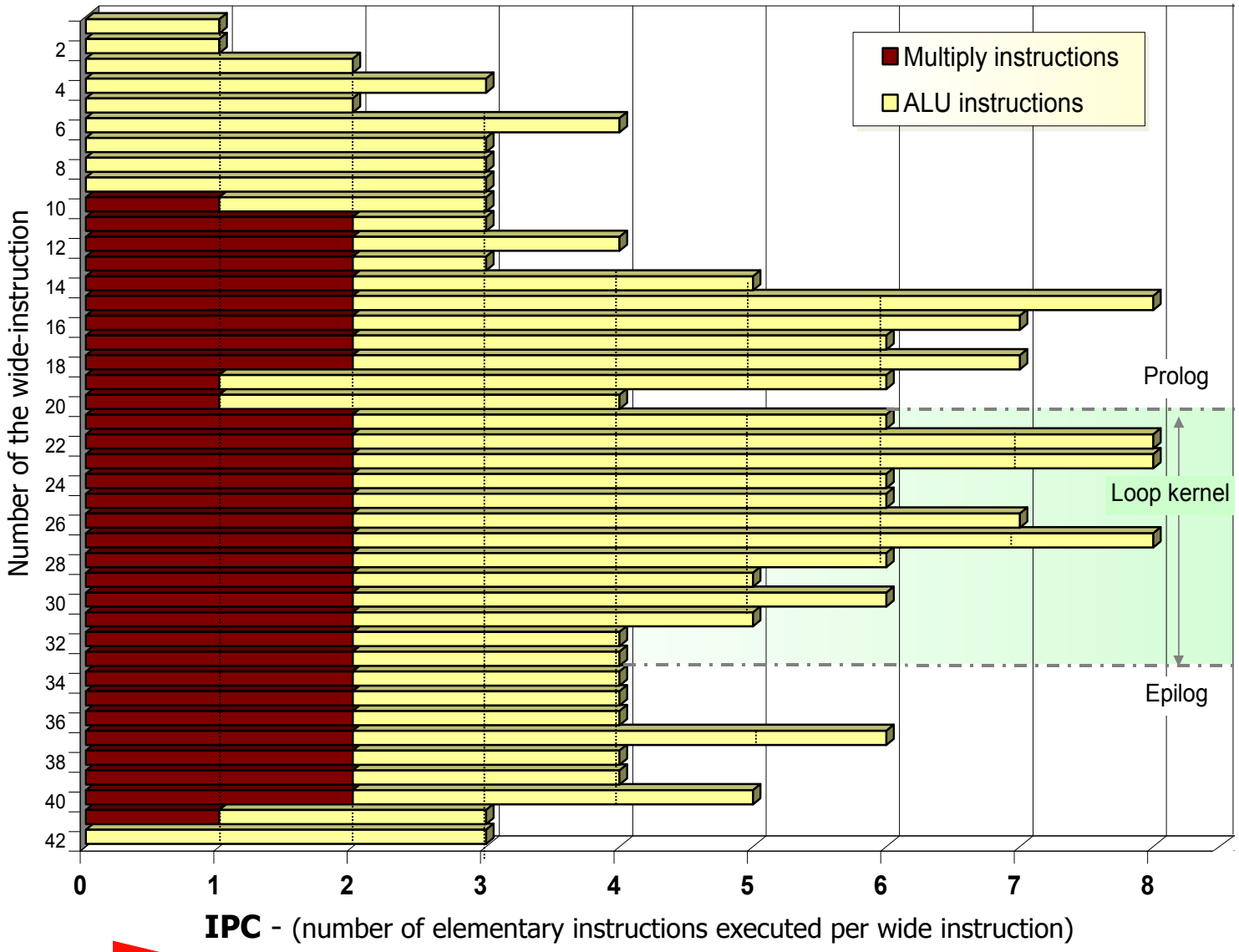
Cycle	.D1	.D2	.L1	.L2	.M1	.M2	.S1	.S2	NOP
1	LDW	LDW							
2	LDW	LDW							
3	LDW	LDW						SUB	
4	LDW	LDW					B	SUB	
5	LDW	LDW					B	SUB	
6	LDW	LDW			MPY	MPY	B	SUB	
7	LDW	LDW			MPY	MPY	B	SUB	
8	LDW	LDW	ADD	ADD	MPY	MPY	B	SUB	
9	LDW	LDW	ADD	ADD	MPY	MPY	B	SUB	
10			ADD	ADD	MPY	MPY	B	SUB	
11			ADD	ADD	MPY	MPY	B	SUB	
12			ADD	ADD	MPY	MPY	B		
13			ADD	ADD	MPY	MPY			
14			ADD	ADD	MPY	MPY			
15			ADD	ADD					
16			ADD	ADD					

Prolog

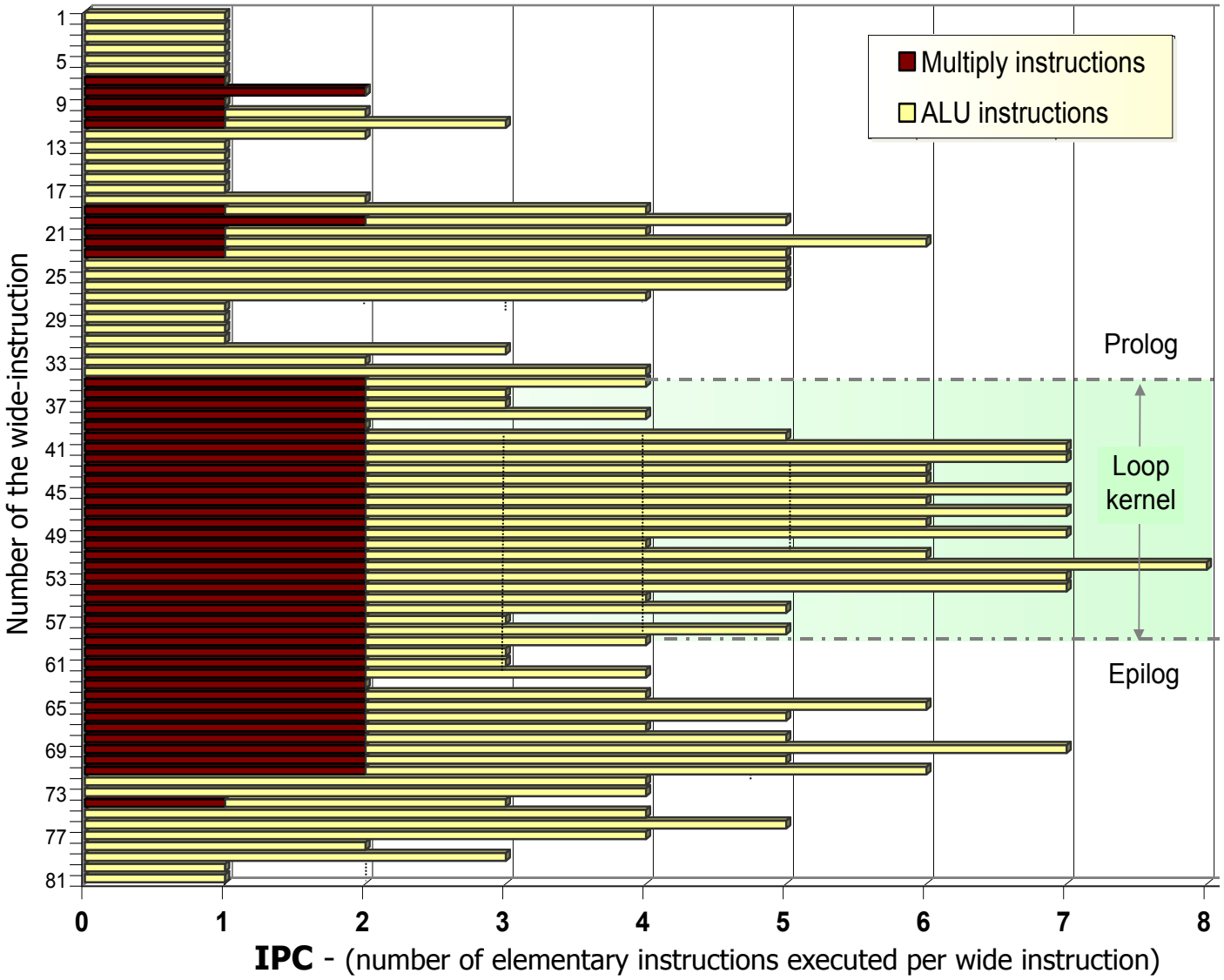
Kernel

Epilog

Wireless application - 1



Wireless application - 2

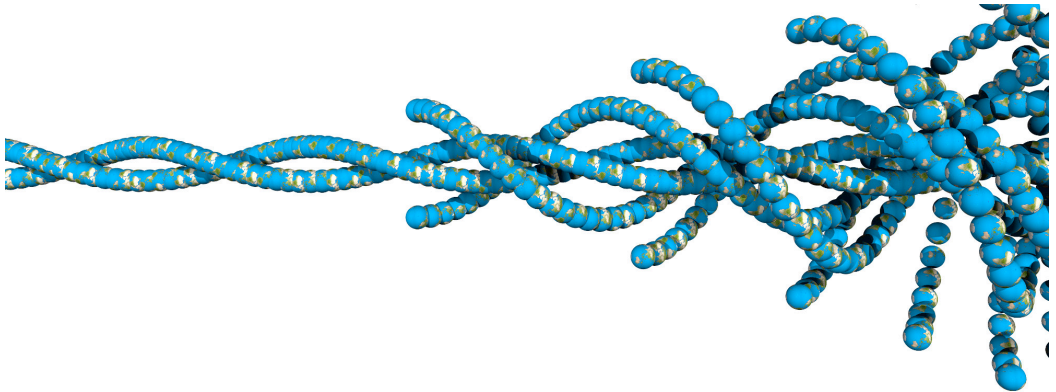


Part 4

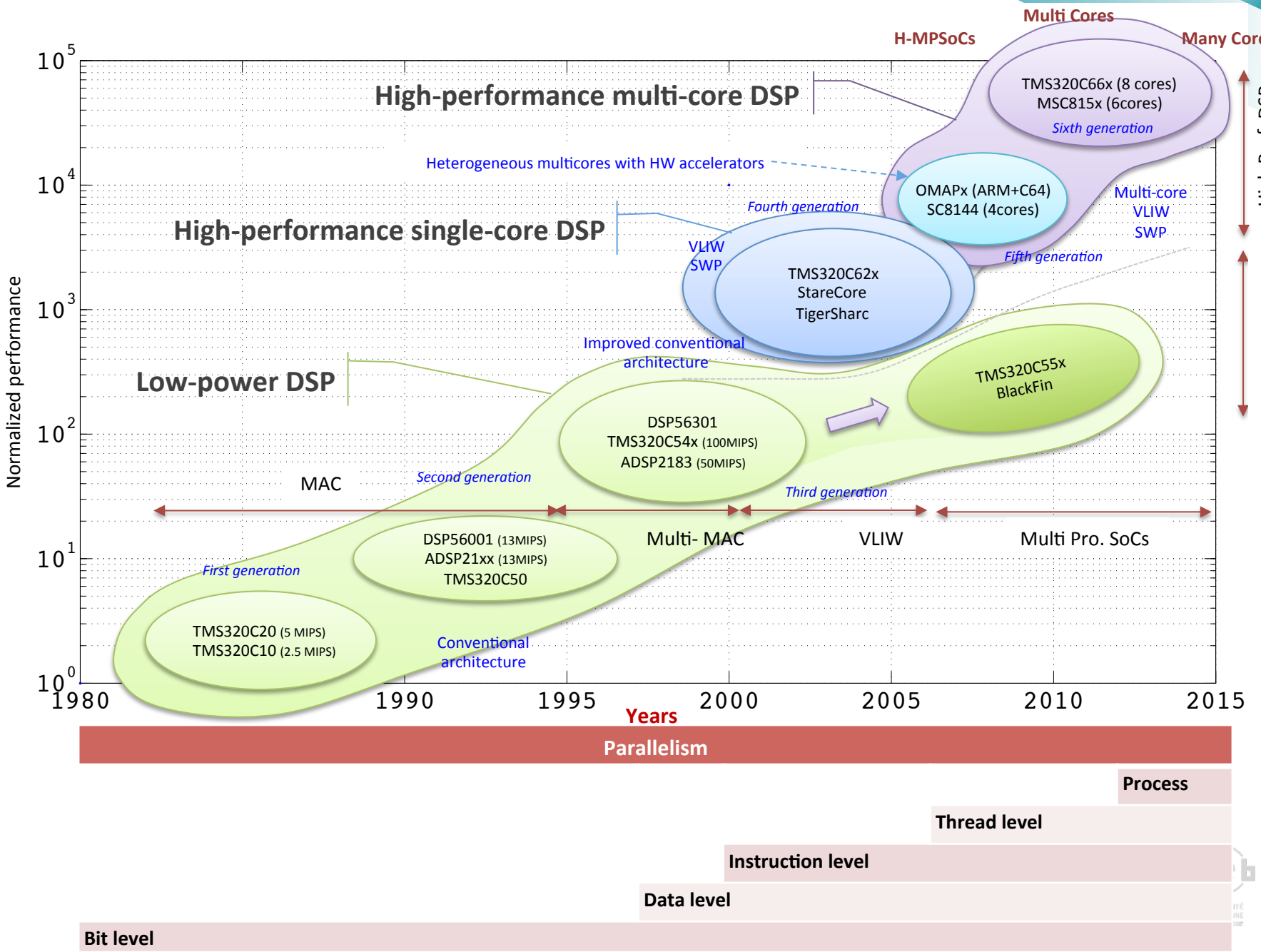
Architecture of high performance Multi-core DSP

- Heterogeneous MPSoCs
- Multi-core Processors

ÉCOLE THÉMATIQUE ARCHI'15



DSP Evolution



Elements of Heterogeneous MPSoCs

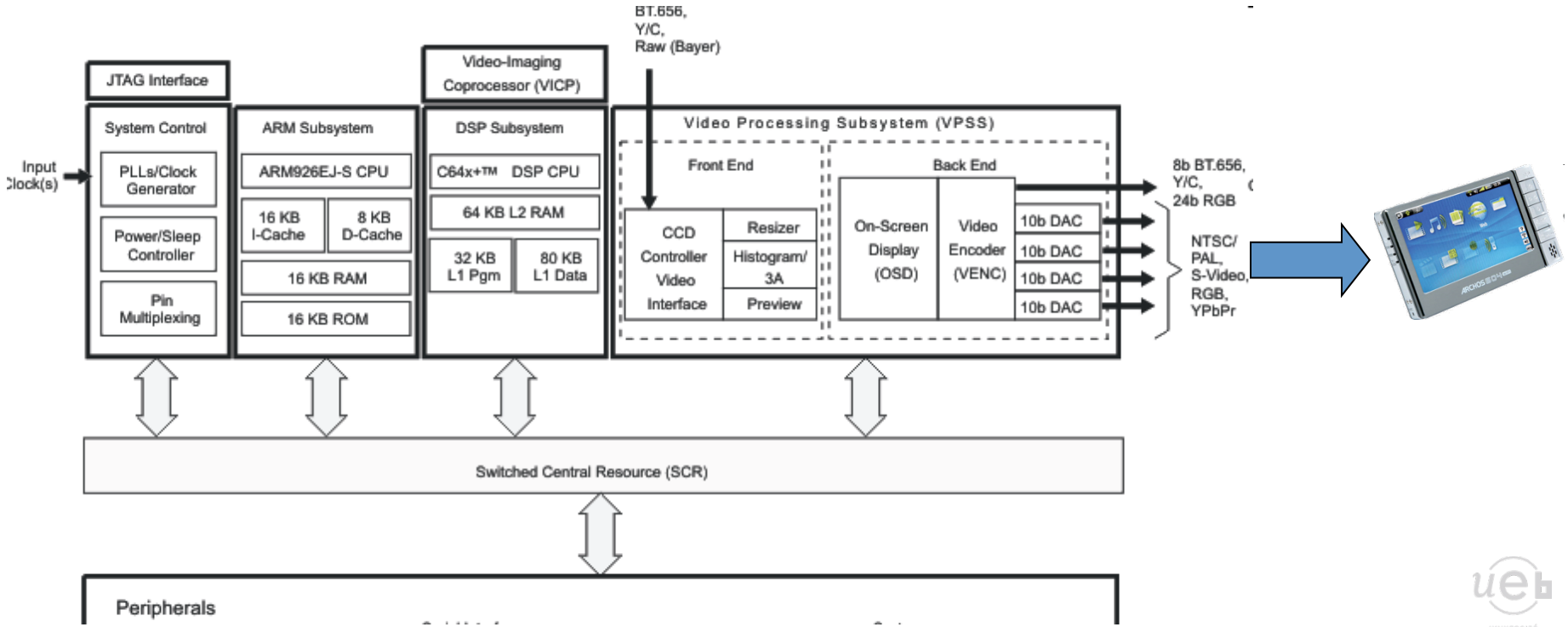
- **Heterogeneous blocks**
  - General Purpose Processor
    - ARM
  - Graphics Processing Unit
    - For video processing applications
  - Digital Signal Processor
    - Low power / High-performance / Both
  - Hardware accelerator
    - Implement standard
      - ✓ Video H.264
    - Common functions (filtering, transforms, …)

Elements of Heterogeneous MPSoCs

- **Dedicated to a domain**
  - Specific HW accelerators
  - Telecom
    - FFT, viterbi / turbo decoder
  - Video
    - Video coding and decoding
    - Video processing

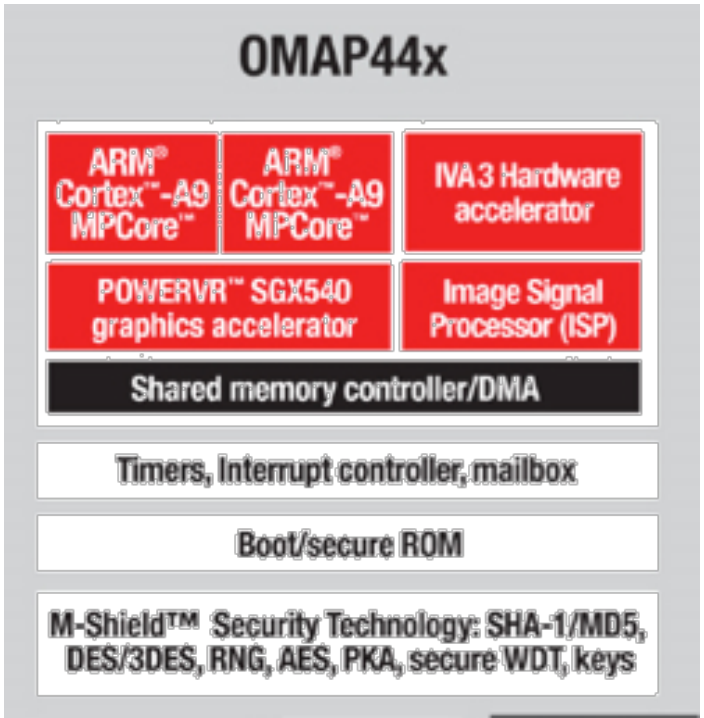
Example of Heterogeneous MPSoCs

- **Da Vinci TMS320DM644x**
  - 1 DSP tms320c64x+ (720MHz max)
  - 1GPP ARM9
  - 1 HW Video Accelerator



Example of Heterogeneous MPSoCs

- **OMAP 4 Platform: OMAP4430/OMAP4440**
  - 2 GPP ARM Cortex A9 cores (1GHz)
  - 1 HW Accelerator : IVA 1080p video encoding/decoding
  - 1 GPU
  - 1 Image/Signal processing

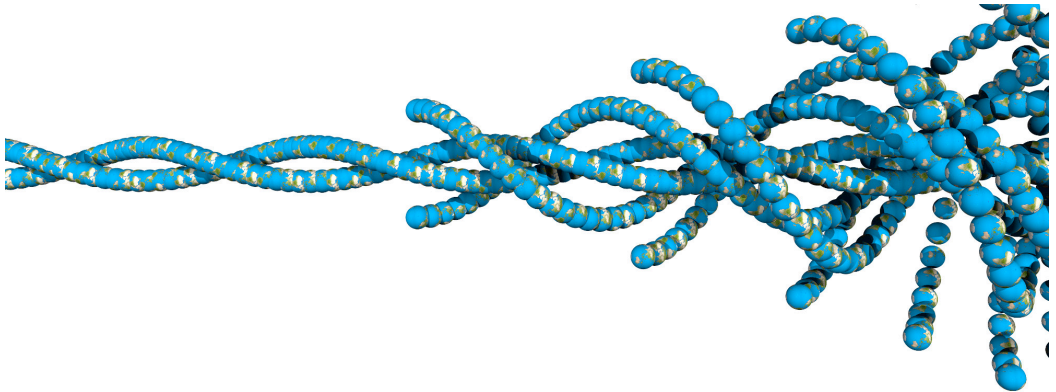


Part 4

Architecture of high performance Multi-core DSP

- Heterogeneous MPSoCs
- Multi-core Processors

ÉCOLE THÉMATIQUE ARCHI'15



Levels of parallelism

Many-core
(distributed memory)

Multi-core (MPSoc)
(shared memory)

Instruction
(ILP)

Data (SWP)

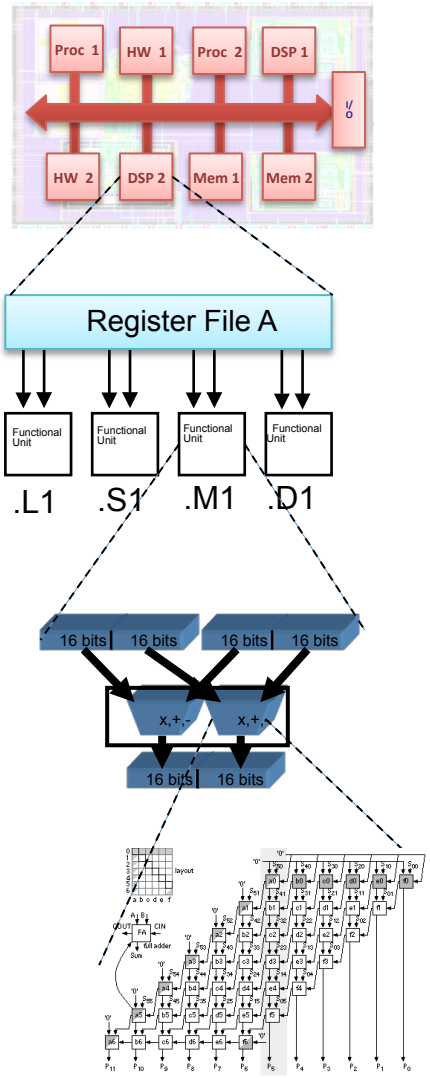
Bit (HW ops.)

Process

Threads

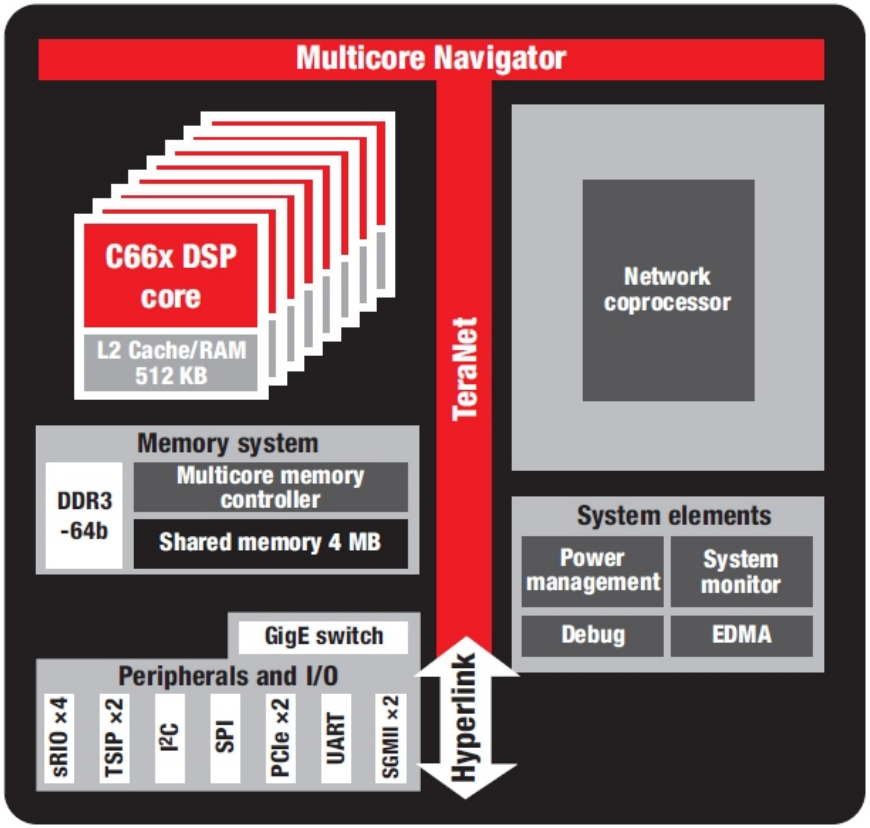
Operation

Bit



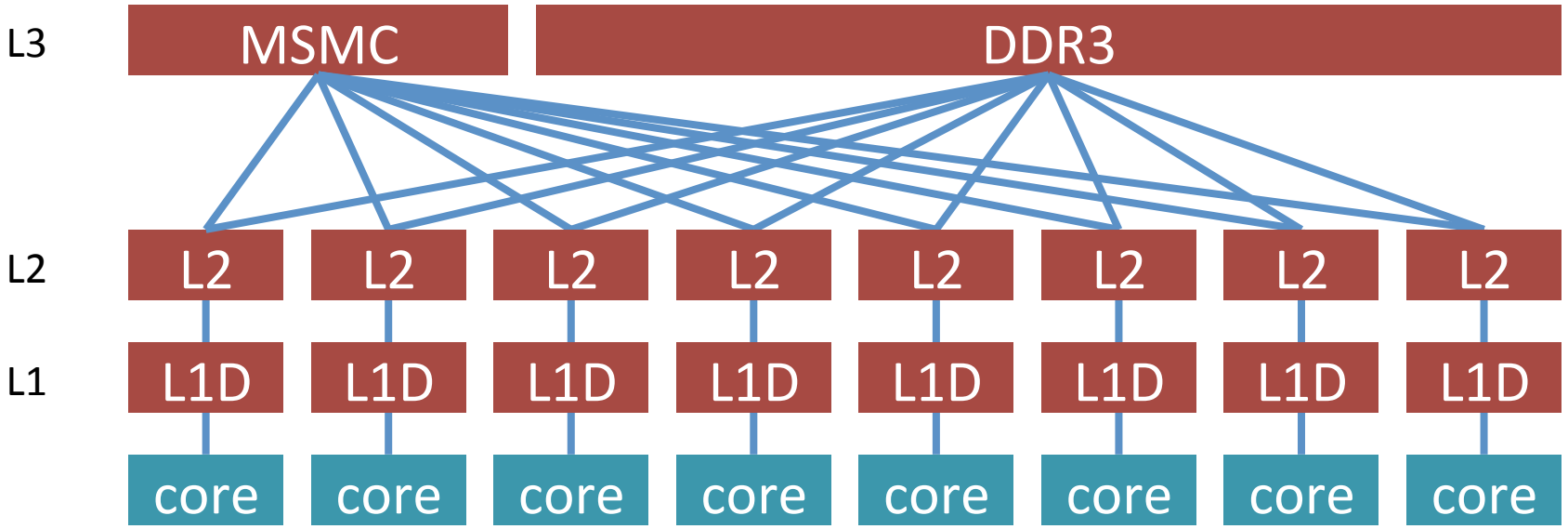
Multicore processors

- **Keystone I (Texas Instrument, 2011)**
  - 8 cores C66x
  - Shared memory
  - Multicore Navigator
    - Central Queue Manager
      - ✓ Access to HW Queue
    - Multiple Packet DMA engines



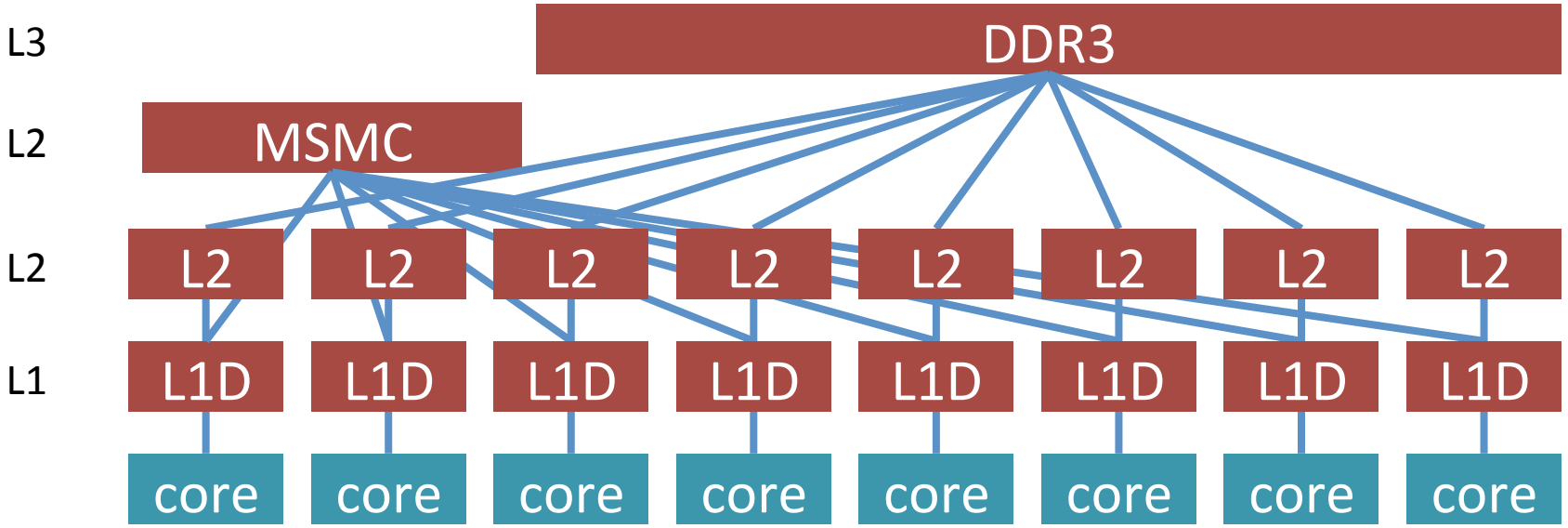
Keystone I

- **Memory hierarchy**
  - L2 can cache MSMC (Multicore Shared Memory Controller) and external DDR3



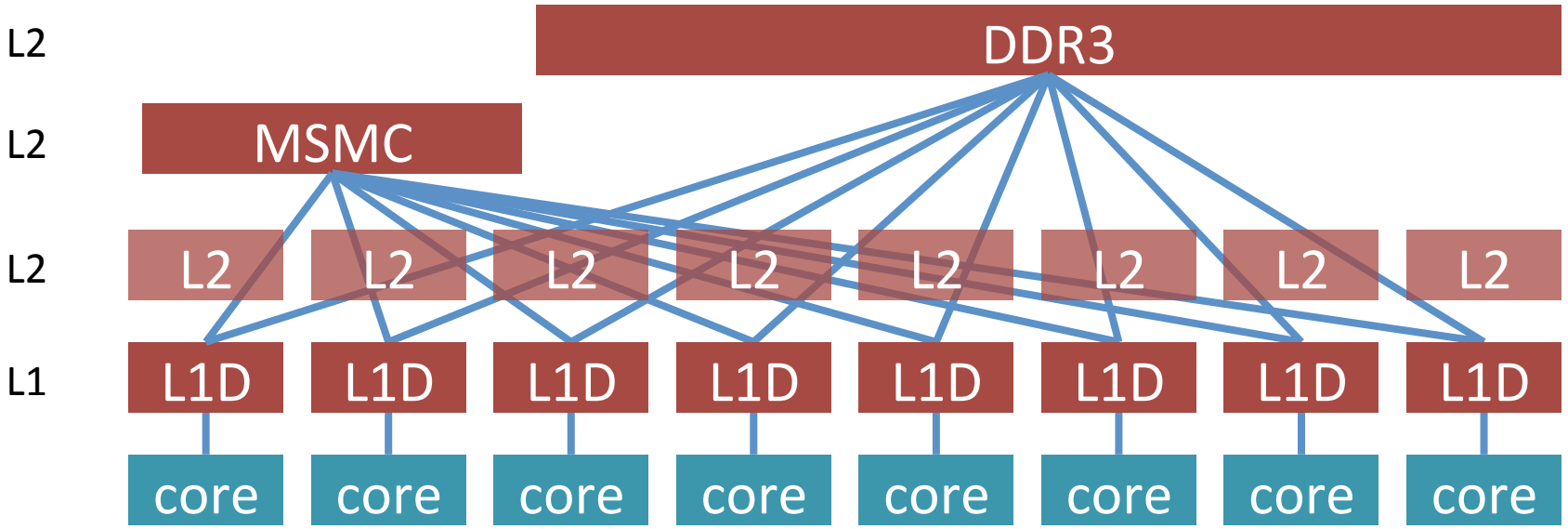
Keystone I

- **Memory hierarchy**
  - L2 can cache only external DDR3
  - MSMC is then Layer 2 memory



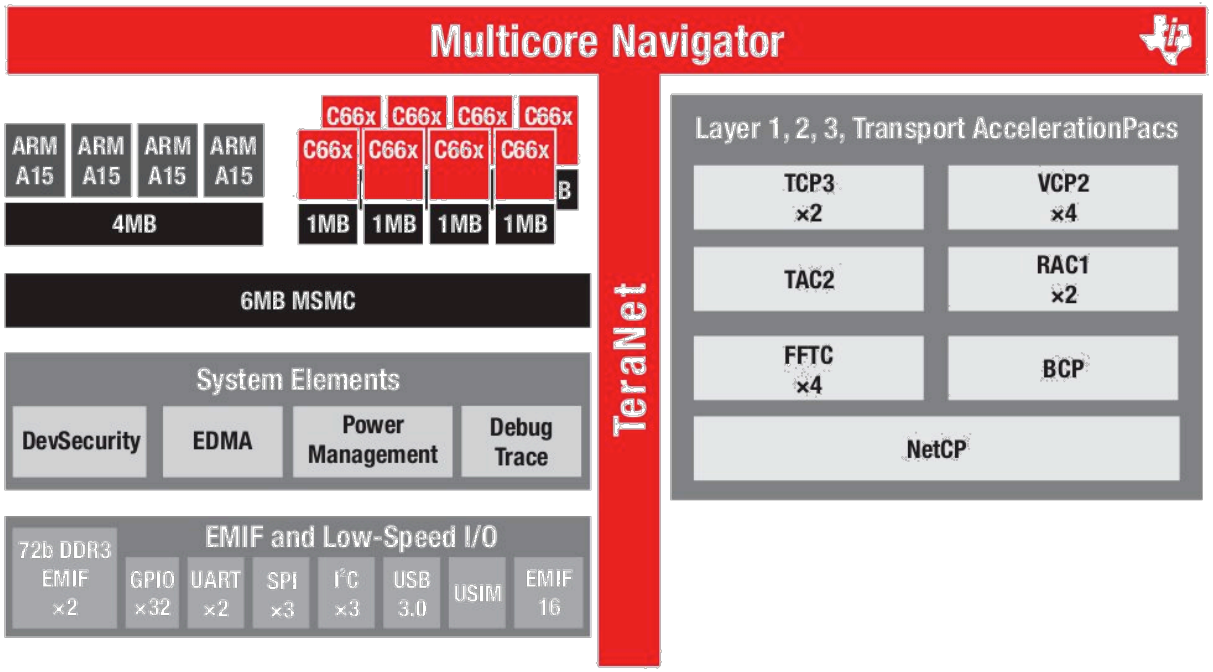
Keystone I

- **Memory hierarchy**
  - L2 cache is deactivated
  - MSMC is then Layer 2 memory



Multicore processors

- **Keystone II (Texas Instrument)**
  - 8 cores C66x
  - 4 cores Arm Cortes A15
  - Shared memory



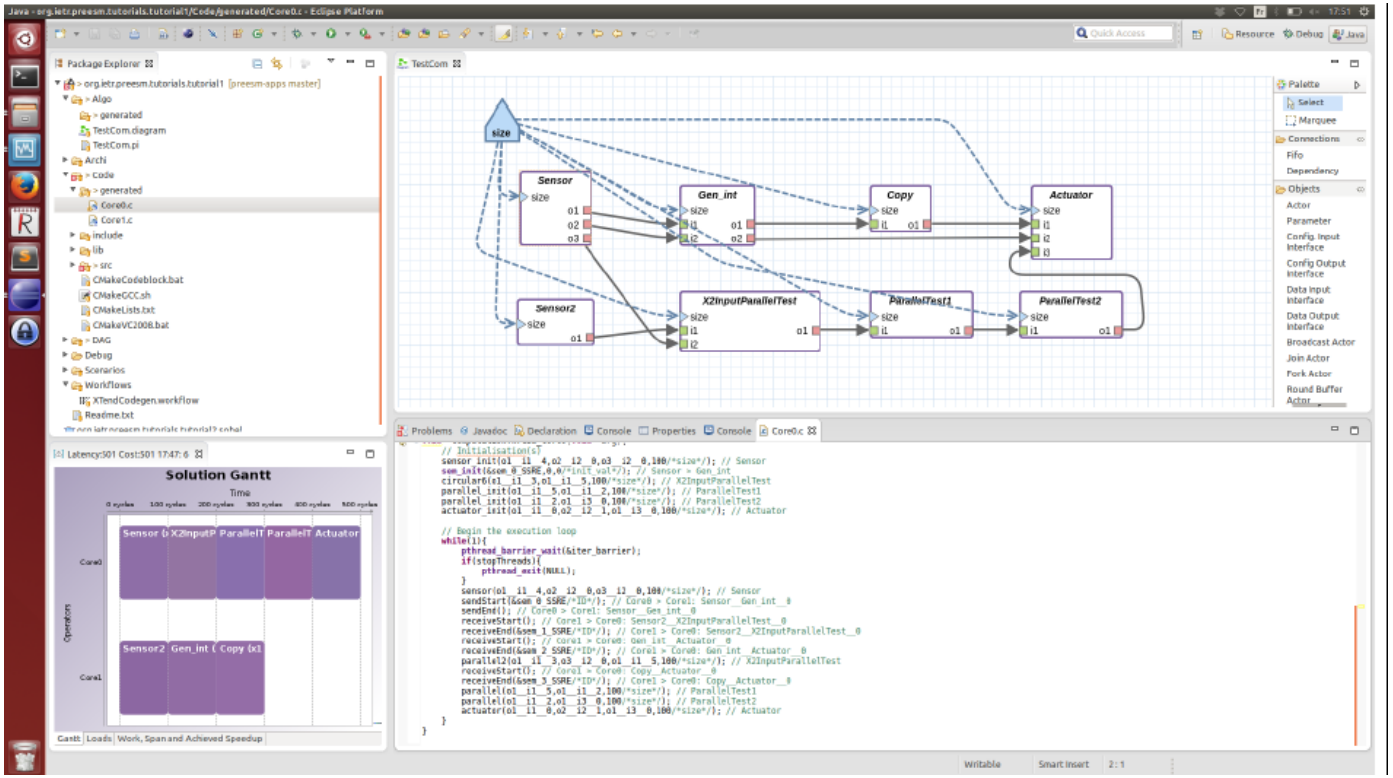
Multicore processors

- **Parallel programming for Keystone**
  - OpenEM : services to manage
    - events
    - event queues
    - execution objects
  - Support of OpenMP
    - Industry standard for shared memory parallel programming
    - Code annotations with compiler directives (pragmas)
    - Adaptation for DSP RT-OS proposed by TI

Multicore processors

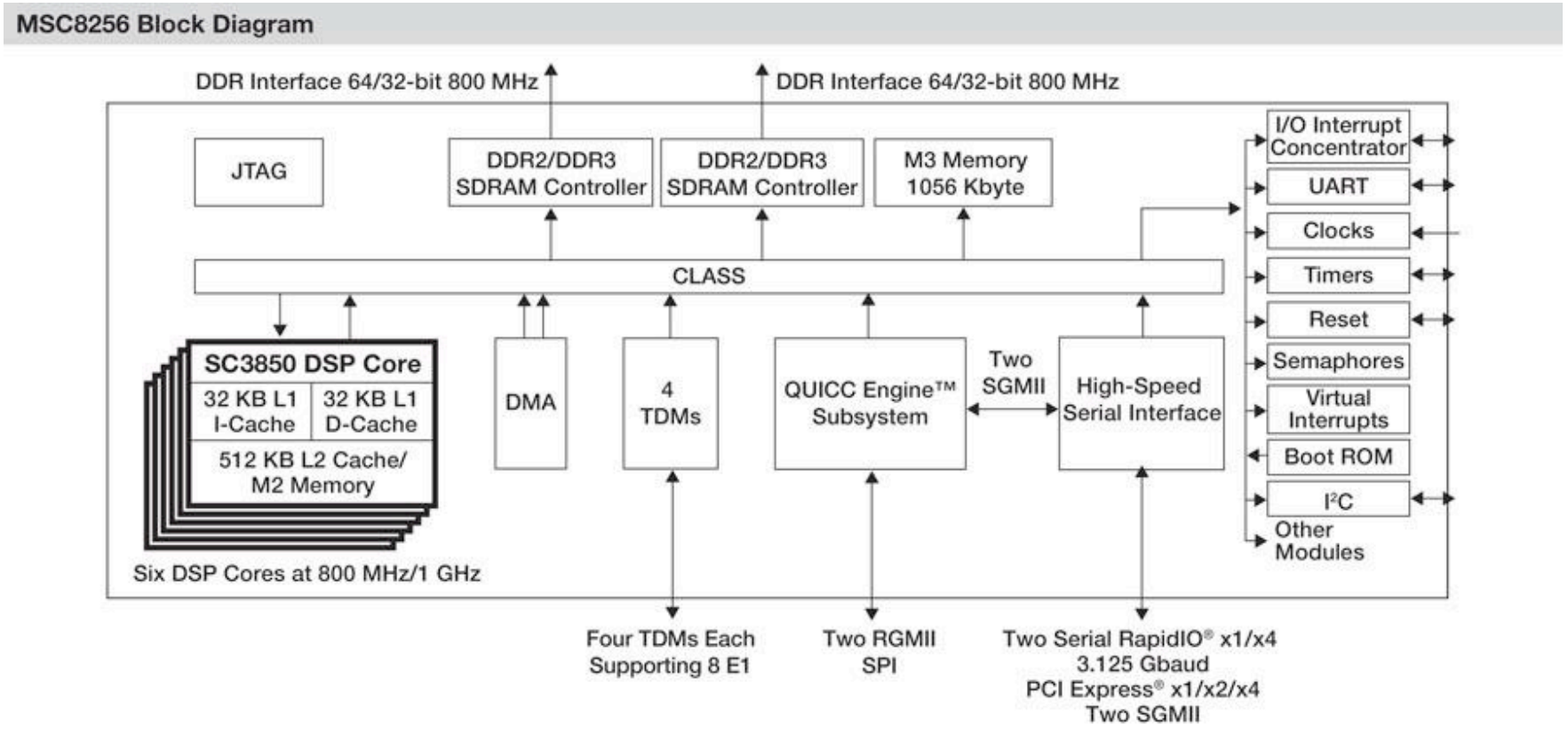
- **Parallel programming for Keystone**
  - PREESM
    - Data Flow programming

http://preesm.sourceforge.net/



Multicore processors

- **StareCore (Freescale)**
  - 6 DSP cores SC3850

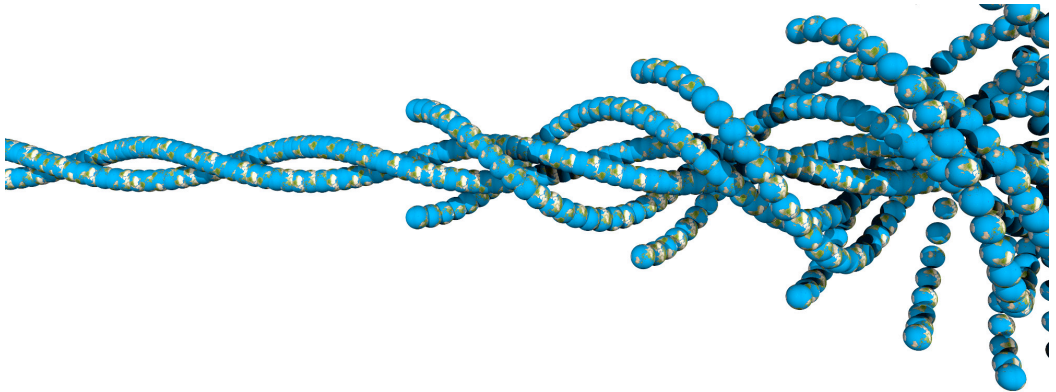


Part 5

Arithmetic for DSP applications

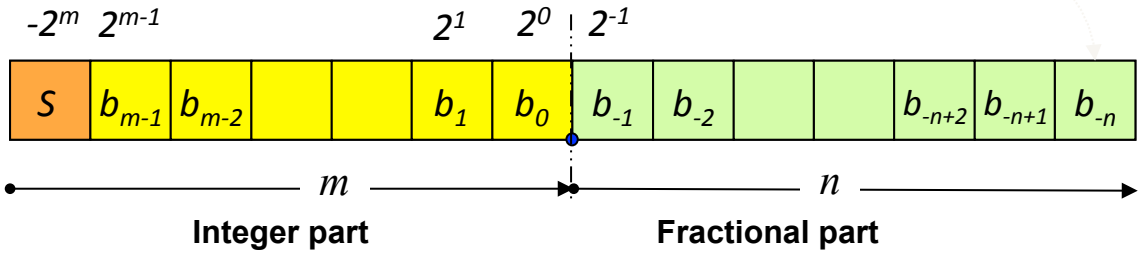
- Fixed-point and floating arithmetic
- Comparison

ÉCOLE THÉMATIQUE ARCHI'15



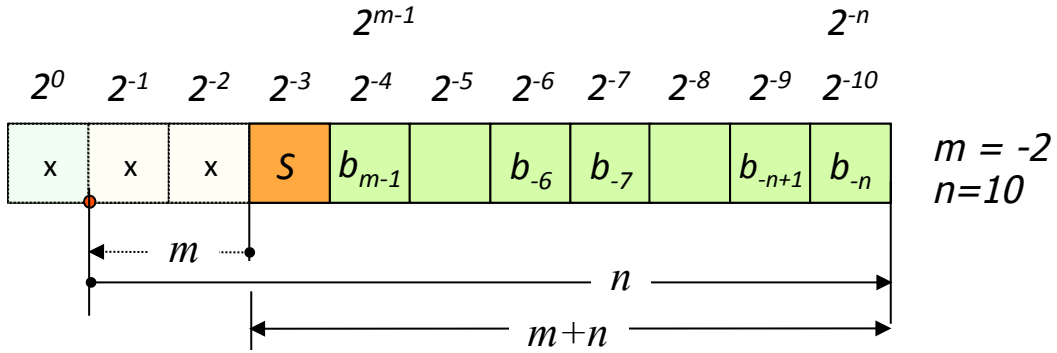
2's complement fixed-point number

- **Definition :**



  - Fixed-point format $Q_{m.n}$
    - m : distance (number of bits) between the most significant bit p_{MSB} and the radix-point position p_V
    - n : distance (number of bits) between the radix-point position and the least significant bit

$$x = -2^m S + \sum_{i=-n}^{m-1} b_i 2^i$$



*0.09472 in $Q_{-2.10}$: **0**1100001* *$0.0625+0.03125+2^{-10}$*

2 complement fixed-point number

- **Definition domain** $[-2^m; 2^m - 2^{-n}]$
- **Quantization step**
  - Distance between two consecutive values
    - LSB weight

$$q = 2^{-n}$$

- **Example of fixed-point coding for $Q_{4.2}$**

0111.11	7.75
0111.10	7.5
0000.11	0.75
0000.10	0.5
0000.01	0.25
0000.00	0
1111.11	-0.25
1111.10	-0.5
1111.01	-0.75
1000.01	-7.75
1000.00	-8

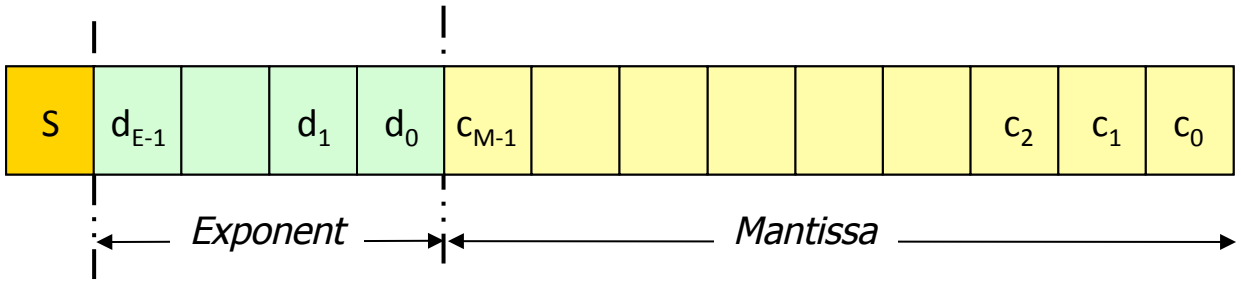
-0.5 = -8+7. 5 →

-7.75 = -8+0.25 →

Floating-point numbers

Definition

- IEEE 754 norm, simple precision (E=8, M=23)



$$x = 2^{u}(-1)^{S}\left(1+\sum_{i=0}^{M-1} C_i 2^{i-M}\right) \qquad \text{avec} \quad u = \sum_{i=1}^{E-1} d_i 2^{i} \; -(2^{E-1}-1)$$

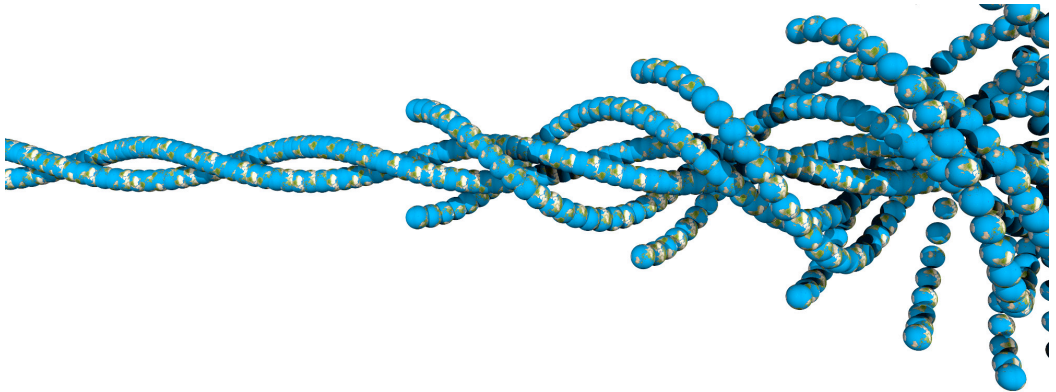
- Explicit scaling factor
- Exponent to scale according to the data value

Part 2

Arithmetic for DSP applications

- Fixed-point and floating arithmetic
- Comparison

ÉCOLE THÉMATIQUE ARCHI'15



Numerical quality

- **Dynamic level**
  - Ability to represent small and high values

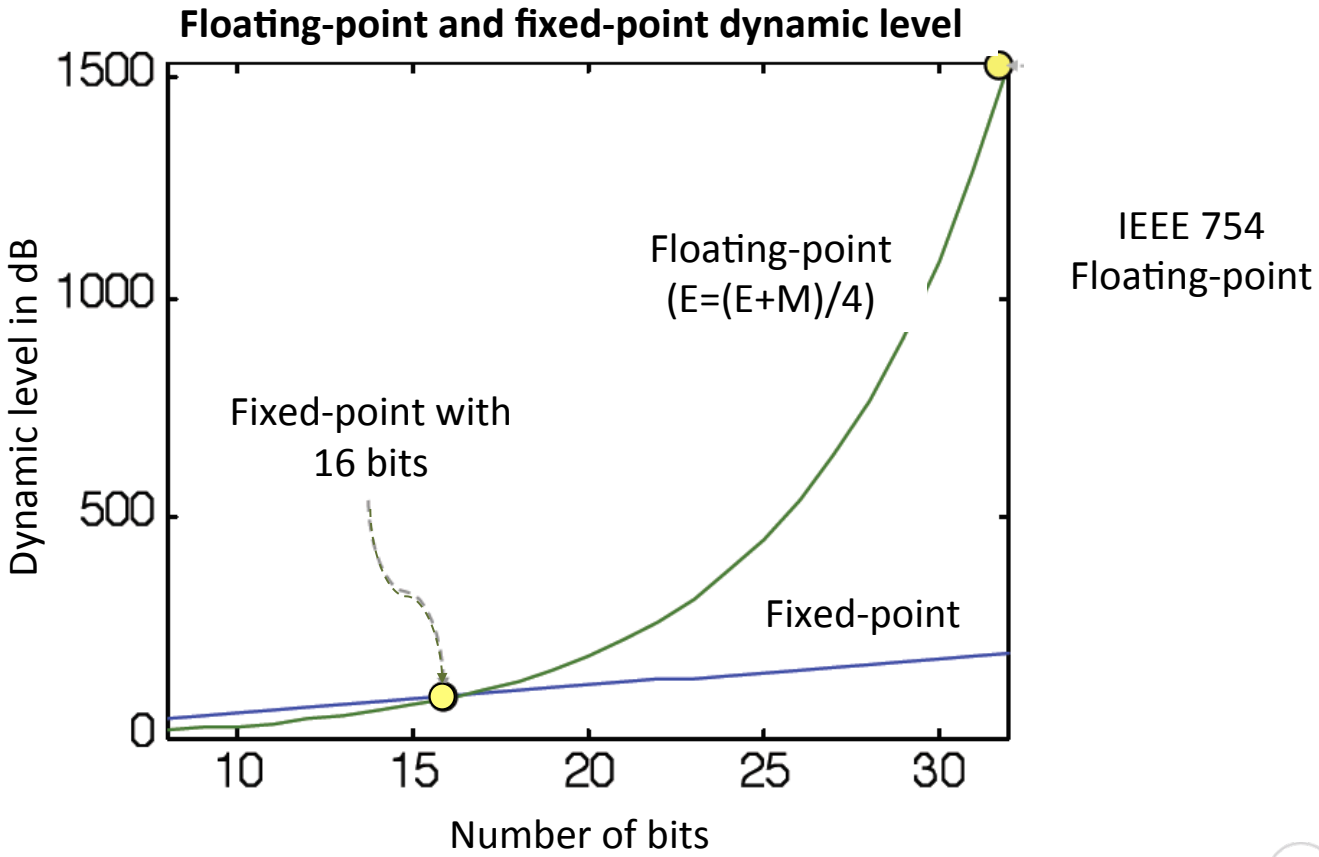
$$D_{N(dB)} = 20.\log\left(\frac{\max(|x|)}{\min(|x|)}\right)$$

- **Signal-to-Quantization-Noise Ratio**
  - Ratio between signal power P_s and quantization error power P_e

$$\rho_{dB} = 10.\log\left(\frac{P_s}{P_e}\right)$$

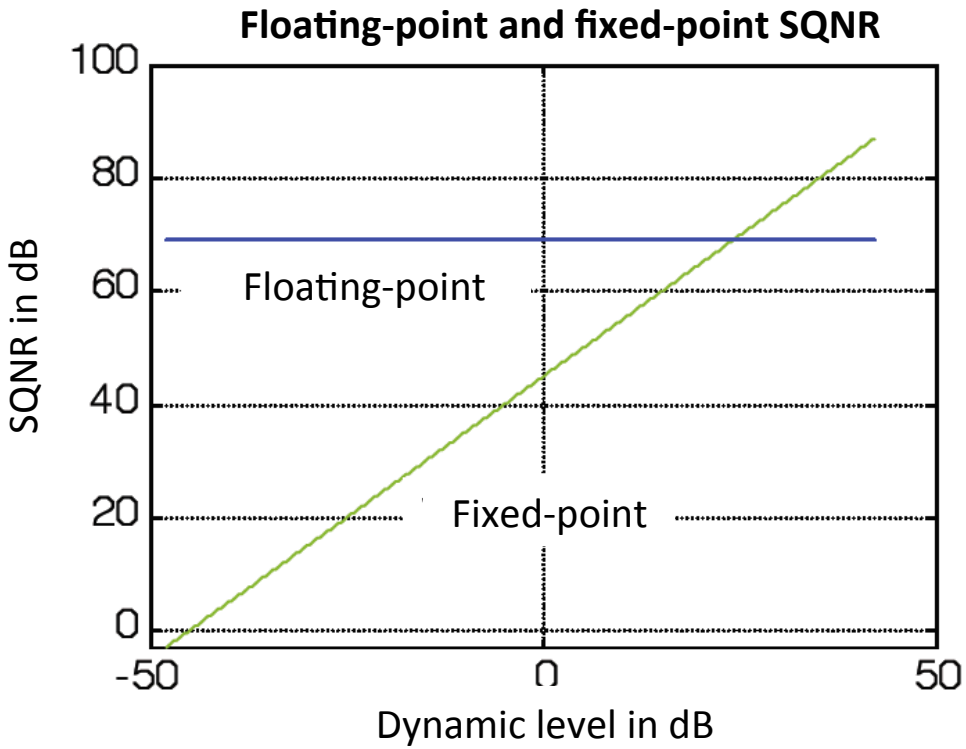
Numerical quality

- **Dynamic level evolution according to the number of bits**



Numerical quality

- **Signal-to-Quantization-Noise Ratio**
  - Evolution of SQNR according to the dynamic level



Application development

- **Larger development times for fixed-point arithmetic**
  - Dynamic range analysis
    - Low dynamic level (16 bits, ≈ 96 dB)
    - High risk of overflow
  - Numerical accuracy evaluation
    - Analyze of computational error influence on the application performance
    - ➢ These analysis are not required for the implementation of DSP applications with floating-point arithmetic

Architecture

- **Lower implementation cost for fixed-point arithmetic**
  - More simple operators
    - No processing for the mantissa and the exponent
    - Normalization with shift operations
      - ✓ Lower area and energy consumption
      - ✓ Lower latency

		Fixed-point	Floating-point
32-bit adder	Area (µm²) Latency (ns)	588 1,55	3977 4,3
32-bit multiplier	Area (µm²) Latency (ns)	23124 2,32	10351 3,92
16 bit multiplier	Area (µm²) Latency (ns)	5683 1,17	

Source Bertrand Le Gal

Architecture

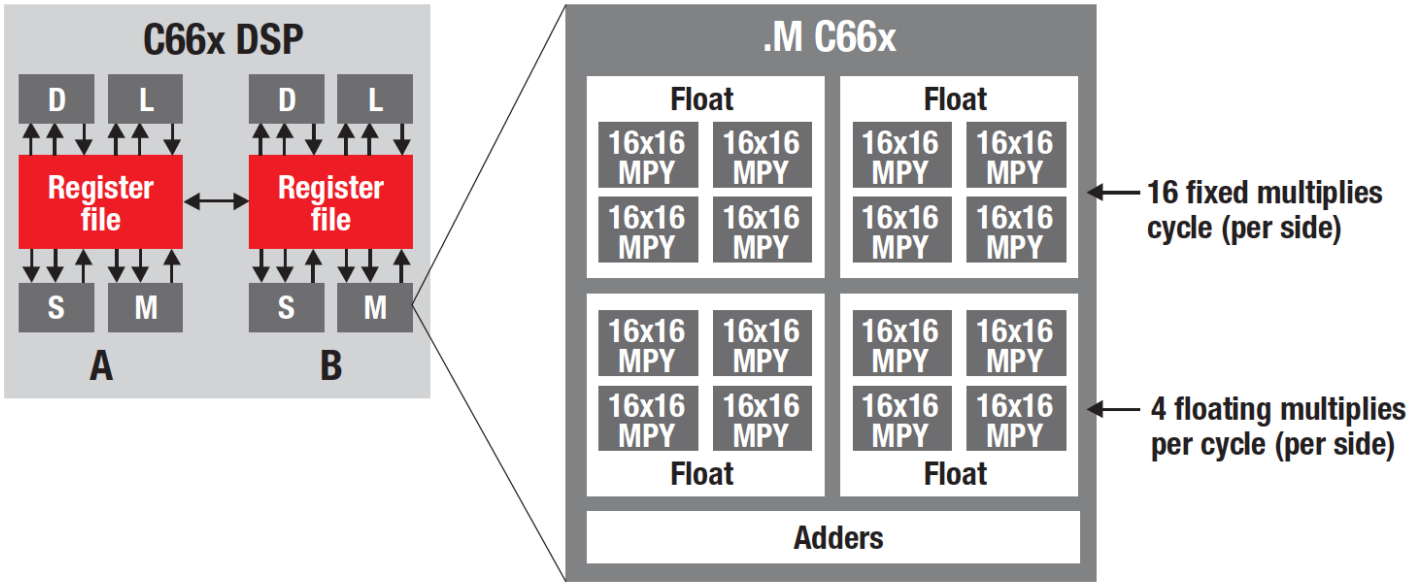
- **Lower implementation cost for fixed-point**
  - Low data word-length (typically 16 bits)
    - Lower width for the buses and the memory
      - ✓ Lower area and energy consumption

    - ➢ Cheaper processor, faster and consuming less energy

TI C6000 family

- **Fixed-point DSP processor**
  - TMS320C62x :
    - f_{CLK} : 150 MHz - 300 MHz
    - On Chip Memory 72 Kbytes - 896 Kbytes
    - Prix : 9$ − 102$
  - TMS320C64x :
    - f_{CLK} : 300 MHz - 1GHz
    - On Chip Memory 160 Kbytes - 1056 Kbytes
    - Prix : 18$ − 219$
- **Floating-point DSP processor**
  - TMS320C67x
    - f_{CLK} : 100 MHz - 350 MHz
    - On Chip Memory 72 Kbytes - 264 Kbytes
    - Price : 14$ − 105$

C66xx fixed-point and floating-point processor



- Fixed-point Arithmetic
  - 32 multiplications 16x16 bits per cycle
  - 64 multiplications 8x8 bits per cycle
- Floating-point arithmetic
  - 8 single precision floating-point operations per cycle

Fixed-point arithmetic

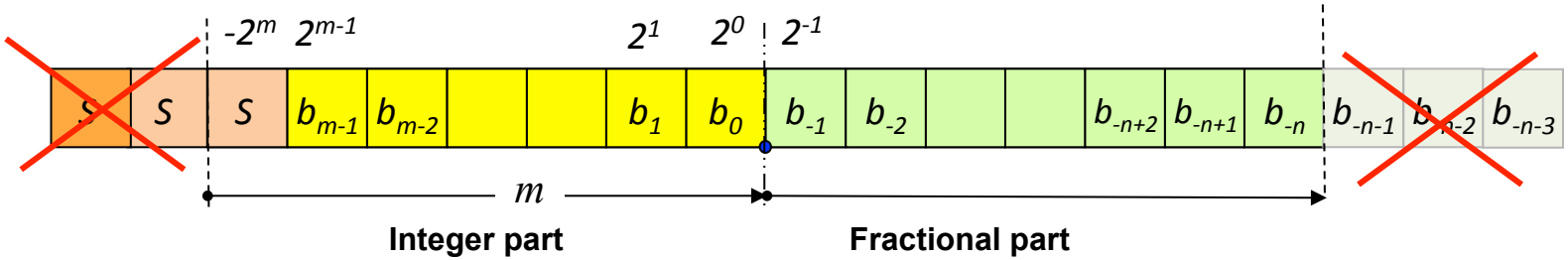
- **Fixed-point arithmetic adapted to application domain**

Application domain	Data	Coefficients	Internal processing	Arithmetic
Telecom	8	16	32	Fixed-point
Modem	8	16	32	Fixed-point
Audio	24	32	48	Floating-point
Video	8-10	16	32	Fixed-point
Image	8-10	16	32	Fixed-point
Control	8-10	16	32	Fixed-point

- **Market for fixed-point systems**
  - Consumer electronics

Fixed-point arithmetic

- **Reduction of the number of bits**



- **Overflow management**
  - Values to choose in case of overflow
- **Wrap-around**
  - Sign modification
  - No supplementary hardware
- **Saturation**
  - No sign modification
  - Supplementary hardware
- **Truncation**
  - Elimination of the k last bits
  - No supplementary HW
- **Rounding**
  - Truncation of the k last bits after addition of 2^{-n-1}
  - Supplementary HW

Fixed-point C code for FIR filter example

```
01. int Input[M] = {4030,..., ...}                          /* Signal x in Q1.15 */
02. int c[N] = {13107, 14336, 15565, ..., 13107, 14336};    /* Coefficients in Q0.16  */
03.
04. int main()
05. {
06.  int x[N]; y[M];
07.  long  acc;
08.  int i,j;
09.
10.   for(i=0; i<N; i++){x[i] = 0;}  /* Internal variable initialization  */
11.
12.   for(j=0;  j<M; j++)                /* Input vector filtering  */
13.   {
14.     x[0] = Input[j];
15.     acc = (long)(x[0]*c[0]) >> 4;
16.
17.     for(i=N-1; i>0; i--)
18.     {
19.        acc = acc + ((long)(x[i]*c[i]) >> 4);  /* Tap-filter computation */
20.         x[i] = x[i-1];                  /* Delay */
21.     }
22.         y[j] = (int)(acc>>16);
23.   }
24. }
```

Input signal and coefficients are specified in the C code with 16-bit integer (no fixed-point data types in C) :

- Integer for *Input* Q1.15 obtained by multiplying *Input* by 2^{15}
- Integer for c Q0.16 obtained by multiplying c by 2^{16}

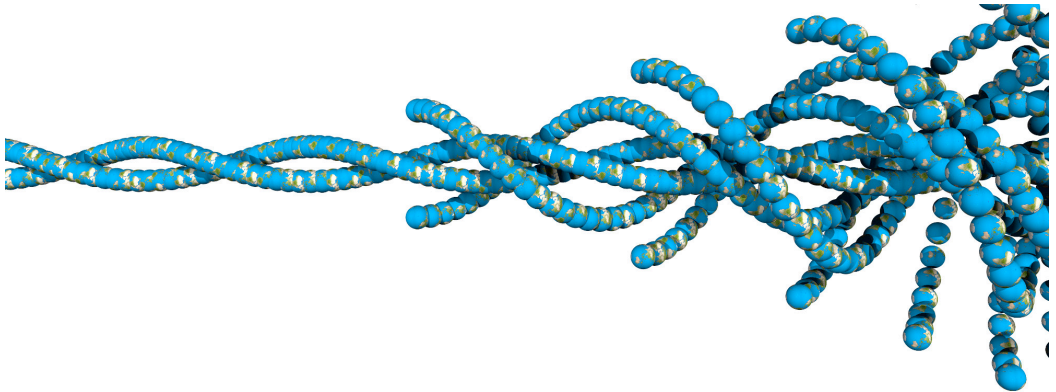
Multiplication output scaling :
Q1.31 ⇨ Q5.27

Cast from 32 bits to 16 bits.
The 16 most significant bits are used *(a classical cast on* acc *allows obtaining the least significant bits)*

Part 6

Conclusion

ÉCOLE THÉMATIQUE ARCHI'15



Conclusions

- **Arithmetic**
  - Convergence of fixed-point and floating-point
- **Low power DSP**
  - Specialized architecture for cost and energy efficiency
  - Bad quality of code generated by compiler
- **High-performance DSP**
  - Exploitation of the different levels of parallelism
    - Data (SWP, SIMD)
      - ✓ Supported by developer, intrinsic functions
    - Instruction (VLIW)
      - ✓ Good support of software pipelining by compiler
    - Multi-cores
      - ✓ Support of standard API