technische universität
dortmund

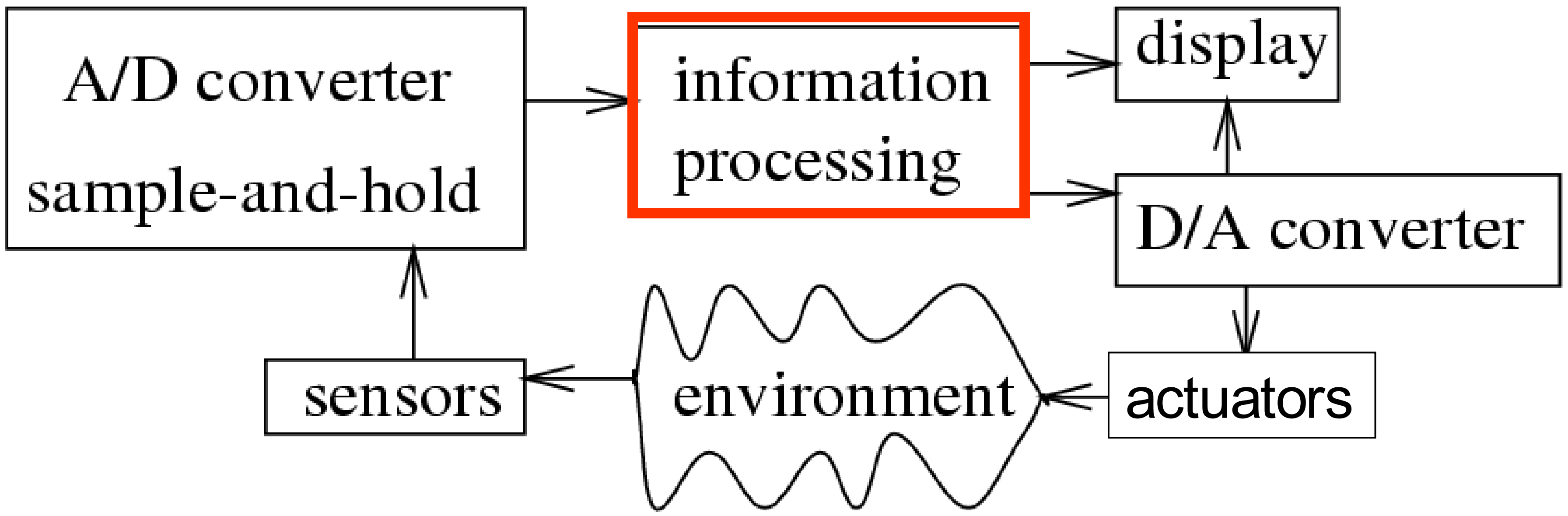
fakultät für informatik
informatik 12

Embedded System Hardware
- Processing -

Peter Marwedel
Informatik 12
TU Dortmund
Germany

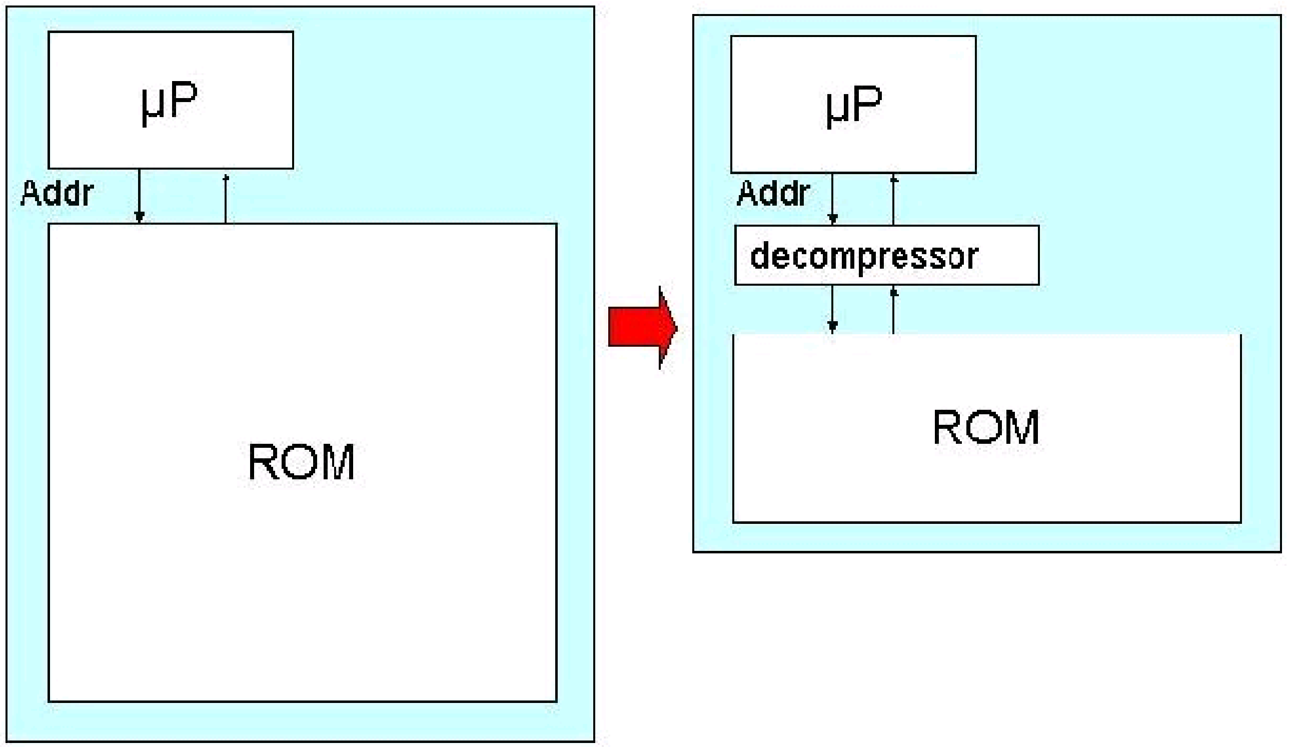
Embedded System Hardware

Embedded system hardware is frequently used in a loop („***hardware in a loop***“):



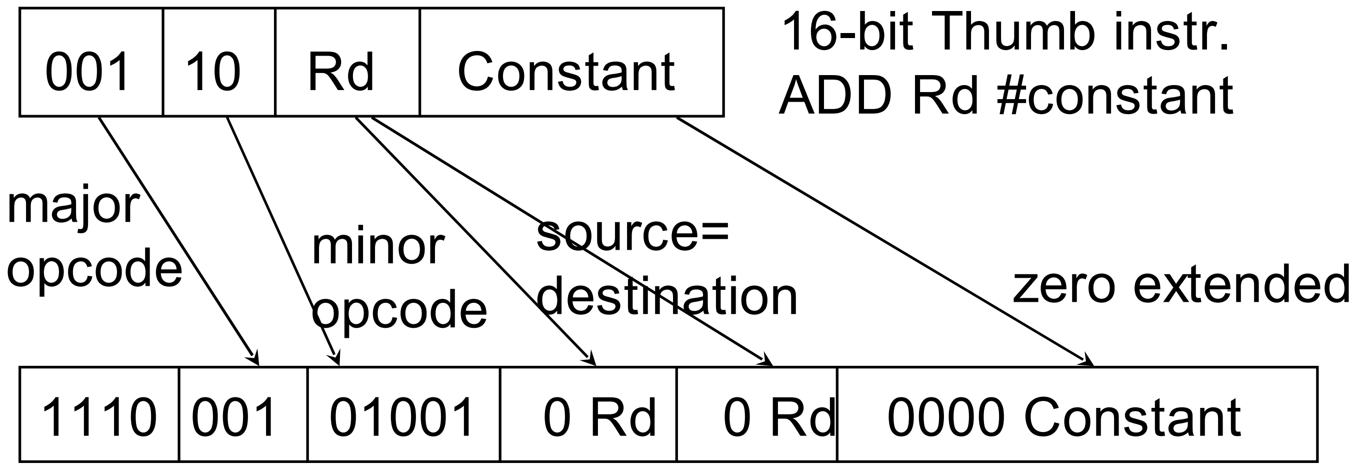
Key requirement #2: Code-size efficiency

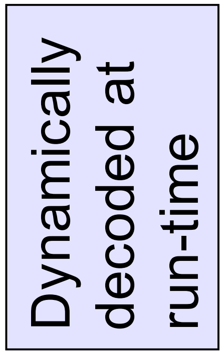
- **CISC machines**: RISC machines designed for run-time-, not for code-size-efficiency
- **Compression techniques:** key idea



Code-size efficiency

- **Compression techniques (continued):**
  - 2nd instruction set, e.g. ARM Thumb instruction set:





- Reduction to 65-70 % of original code size
- 130% of ARM performance with 8/16 bit memory
- 85% of ARM performance with 32-bit memory

[ARM, R. Gupta]

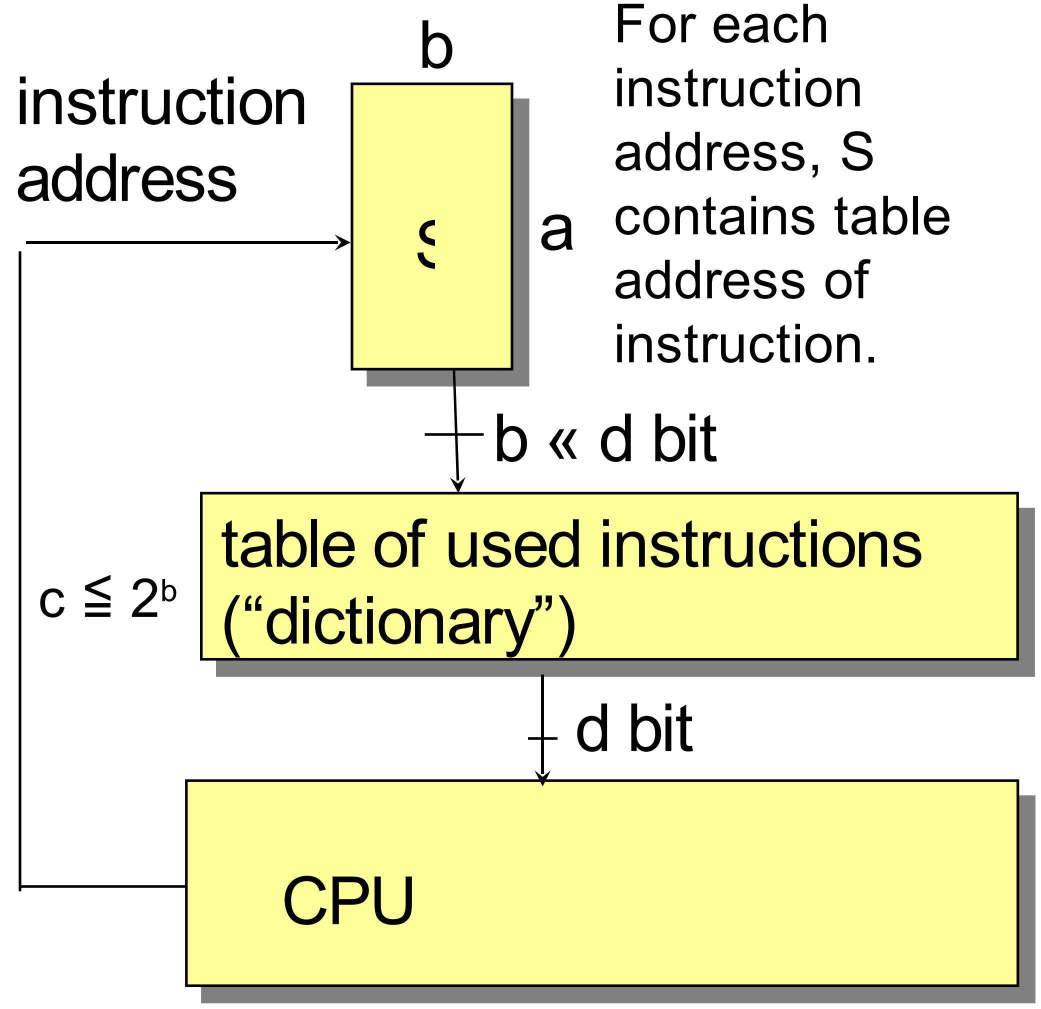
Same approach for LSI TinyRisc, …
Requires support by compiler, assembler etc.

Dictionary approach, two level control store (indirect addressing of instructions)

"Dictionary-based coding schemes cover a wide range of various coders and compressors.
Their common feature is that the methods use some kind of a dictionary that contains parts of the input sequence which frequently appear.
The encoded sequence in turn contains references to the dictionary elements rather than containing these over and over."

[Á. Beszédes et al.: Survey of Code size Reduction Methods, Survey of Code-Size Reduction Methods, *ACM Computing Surveys*, Vol. 35, Sept. 2003, pp 223-267]

Key idea (for *d* bit instructions)


b
instruction address
S
a
For each instruction address, S contains table address of instruction.
b « d bit
table of used instructions ("dictionary")
c ≦ 2^b
d bit
CPU

Uncompressed storage of *a* *d*-bit-wide instructions requires *axd* bits.

In compressed code, each instruction pattern is stored only once.

small

Hopefully, *axb+cxd* < *axd.*

Called nanoprogramming in the Motorola 68000.

Cache-based decompression

- Main idea: decompression whenever cache-lines are fetched from memory.
- Cache lines ↔ variable-sized blocks in memory ☞ line address tables (LATs) for translation of instruction addresses into memory addresses.
- Tables may become large and have to be bypassed by a line address translation buffer.

[A. Wolfe, A. Chanin, MICRO-92]

More information on code compaction

- Popular code compaction library by Rik van de Wiel [http://www.extra.research.philips.com/ccb] has been moved to

  http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html

  http://www.iro.umontreal.ca/~latendre/compactBib/

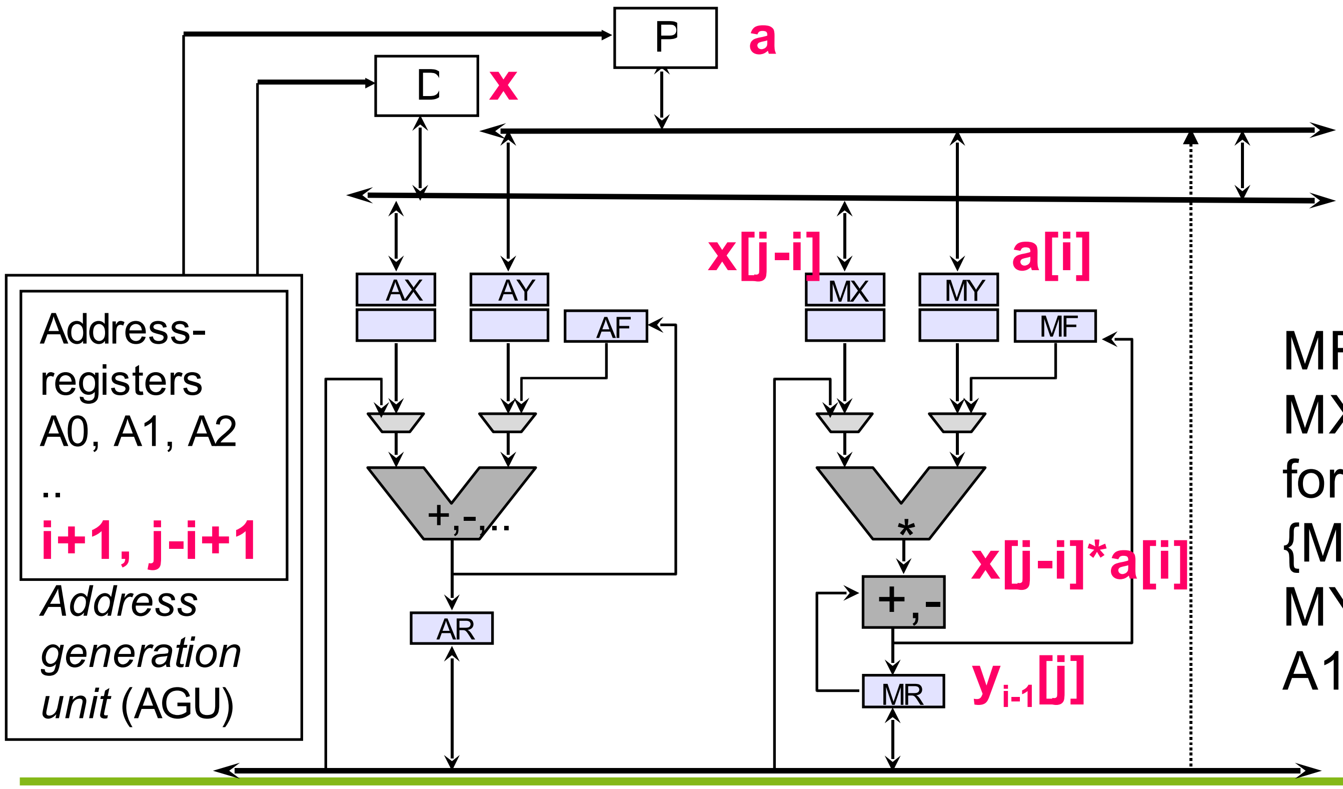
  (153 entries as per 11/2004)

Key requirement #3: Run-time efficiency
- Domain-oriented architectures -

Application: $y[j] = \sum_{i=0}^{n-1} x[j-i]*a[i]$

$\forall i: 0 \le i \le n-1: y_i[j] = y_{i-1}[j] + x[j-i]*a[i]$

Architecture: **Example: Data path ADSP210x**



Application maps nicely onto architecture

```
MR:=0; A1:=1; A2:=n-2;
MX:=x[n-1]; MY:=a[0];
for ( j:=1 to n)
{MR:=MR+MX*MY;
MY:=a[A1]; MX:=x[A2];
A1++; A2--}
```

DSP-Processors: multiply/accumulate (MAC) and zero-overhead loop (ZOL) instructions

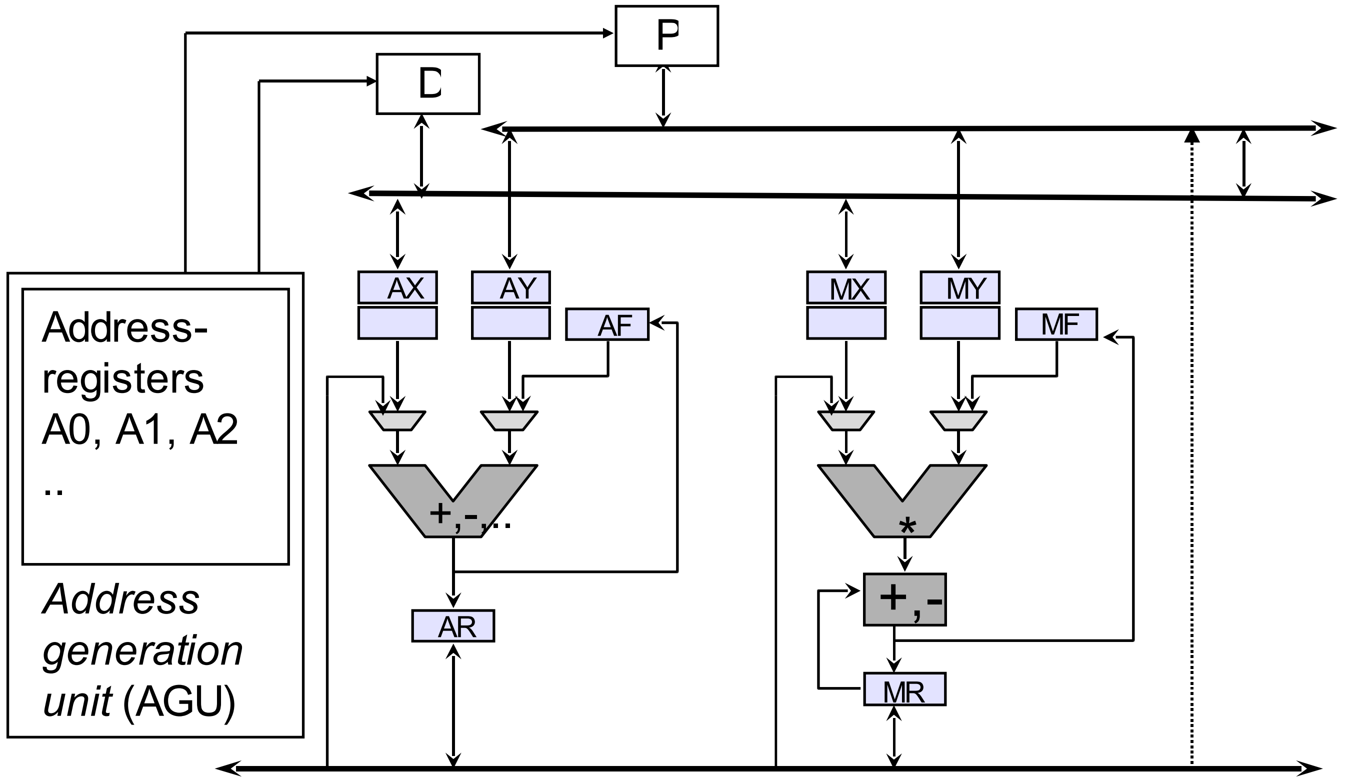
```
MR:=0; A1:=1; A2:=n-2; MX:=x[n-1]; MY:=a[0];
for ( j:=1 to n)
  {MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.
Loop testing done in parallel to MAC operations.

Heterogeneous registers

Example (ADSP 210x):

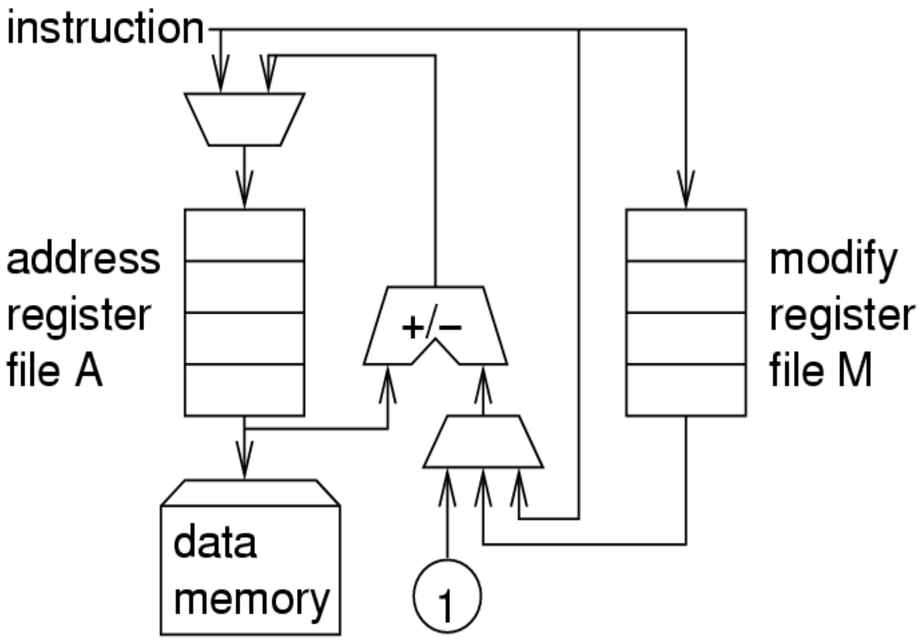


P
D
Address-registers A0, A1, A2 ..
Address generation unit (AGU)
AX
AY
AF
+,-,..
AR
MX
MY
MF
*
+,-
MR

Different functionality of registers An, AX, AY, AF,MX, MY, MF, MR

Separate address generation units (AGUs)

Example (ADSP 210x):



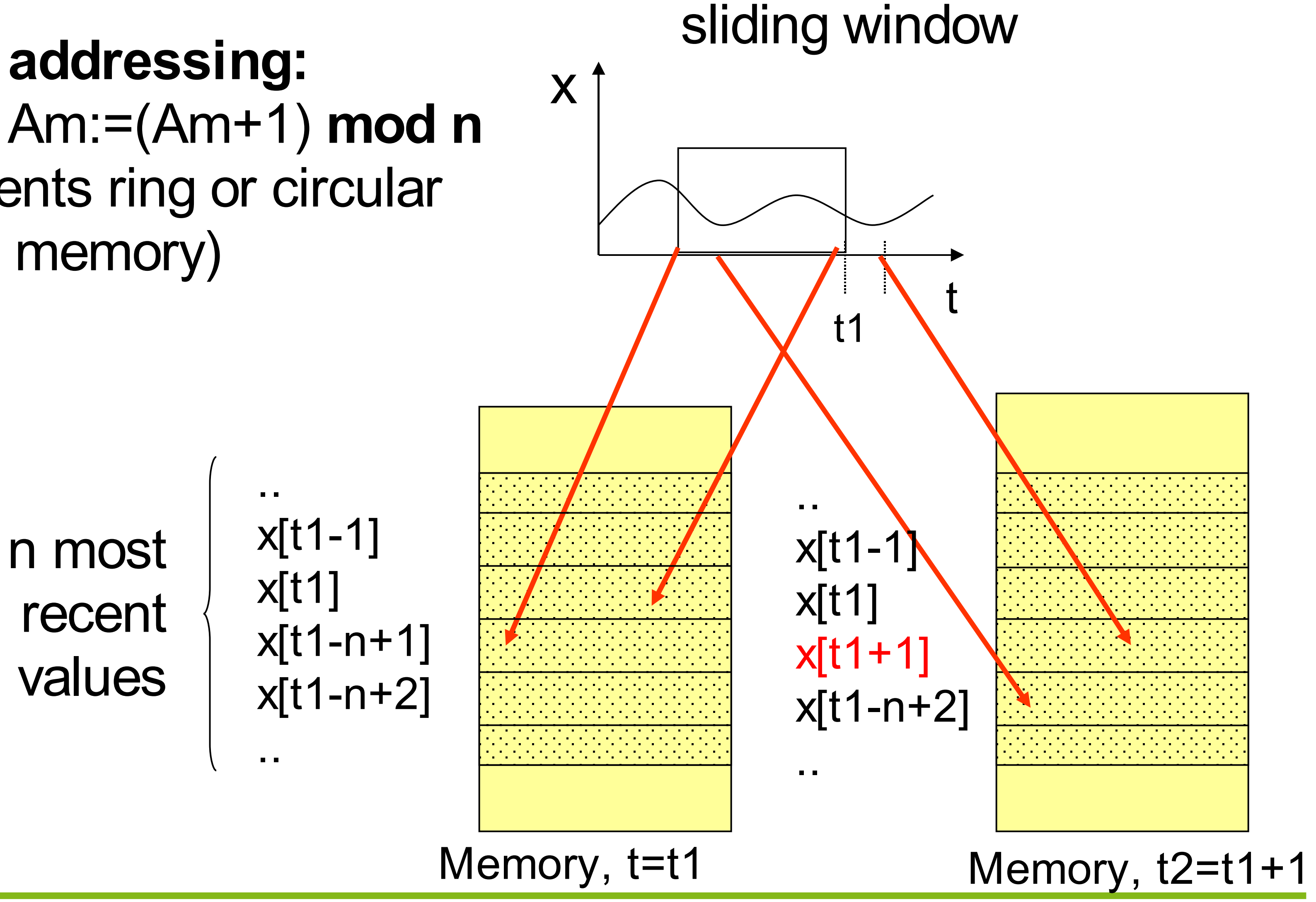
- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- A := A ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate in instruction> requires extra instruction

☞Minimize load immediates

☞Optimization in codesign chapter

Modulo addressing

Modulo addressing:
Am++ ≡ Am:=(Am+1) **mod n**
(implements ring or circular buffer in memory)



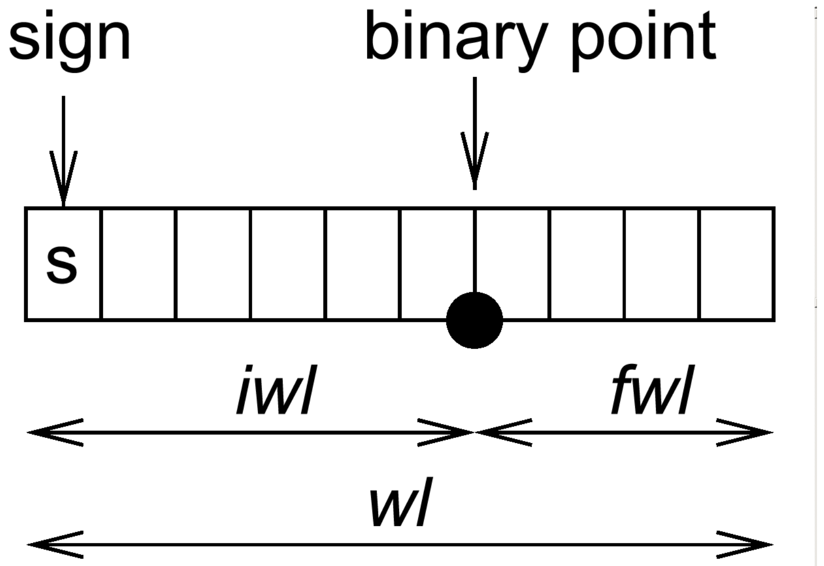
Saturating arithmetic

- Returns largest/smallest number in case of over/underflows
- Example:

a			0111
b		+	1001
standard wrap around arithmetic			(1)0000
saturating arithmetic			1111
(a+b)/2:	correct		1000
	wrap around arithmetic		0000
	saturating arithmetic + shifted		0111 „almost correct“

- Appropriate for DSP/multimedia applications:
  - No timeliness of results if interrupts are generated for overflows
  - Precise values less important
  - Wrap around arithmetic would be worse.

Fixed-point arithmetic



Shifting required after multiplications and divisions in order to maintain binary point.

Properties of fixed-point arithmetic

- Automatic scaling a key advantage for multiplications.
- Example:
  x= 0.5 x 0.125 + 0.25 x 0.125 = 0.0625 + 0.03125 = 0.09375
  For *iwl*=1 and *fwl*=3 decimal digits, the less significant digits are automatically chopped off: x = 0.093
  Like a floating point system with numbers $\in$ [0..1),
  with no stored exponent (bits used to increase precision).
- Appropriate for DSP/multimedia applications (well-known value ranges).

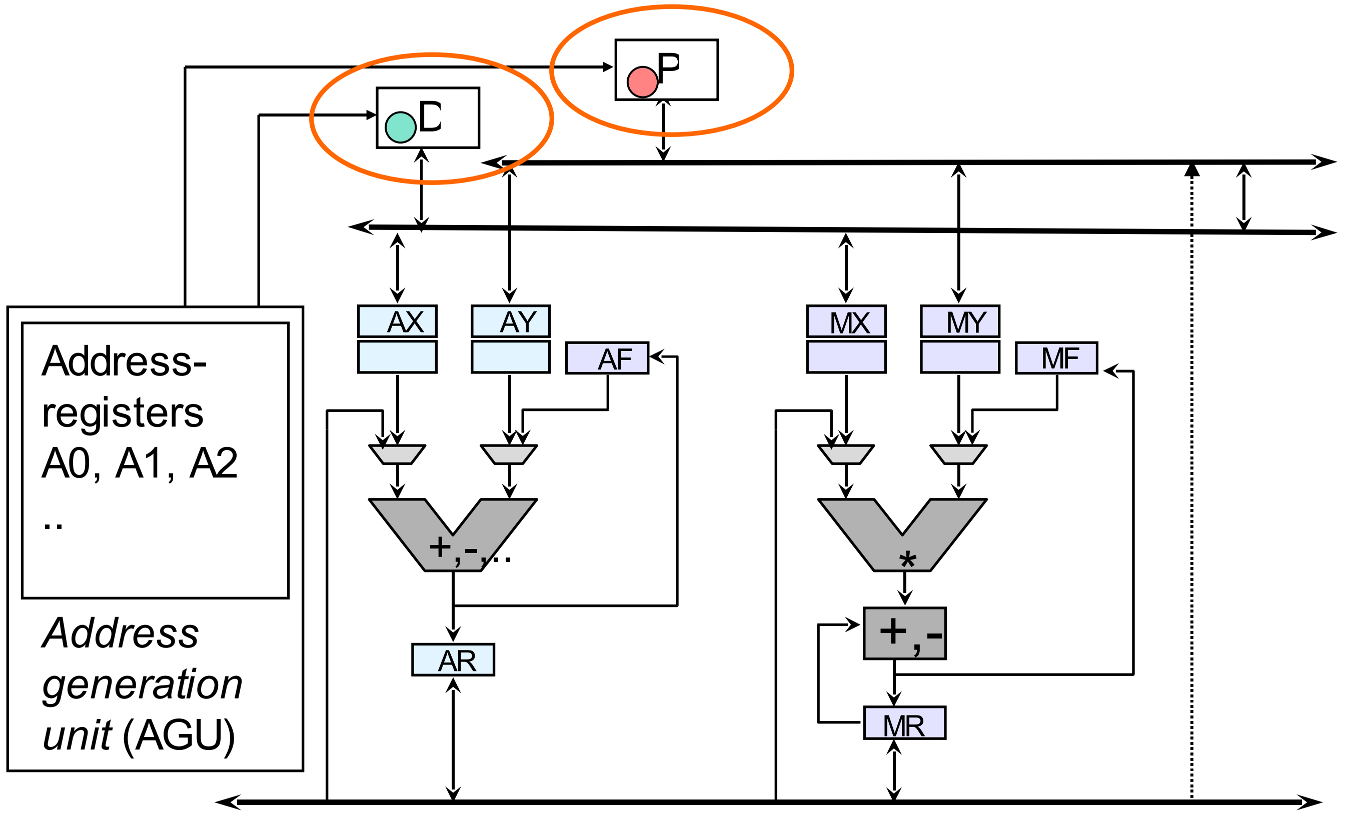
Real-time capability

- **Timing behavior has to be predictable**
  Features that cause problems:
  - Unpredictable access to shared resources
    - Caches with difficult to predict replacement strategies
    - Unified caches (conflicts betw. instructions and data)
    - Pipelines with difficult to predict stall cycles ("bubbles")
    - Unpredictable communication times for multiprocessors
  - Branch prediction, speculative execution
  - Interrupts that are possible any time
  - Memory refreshes that are possible any time
  - Instructions that have data-dependent execution times

  ☞ Trying to avoid as many of these as possible.

[Dagstuhl workshop on predictability, Nov. 17-19, 2003]

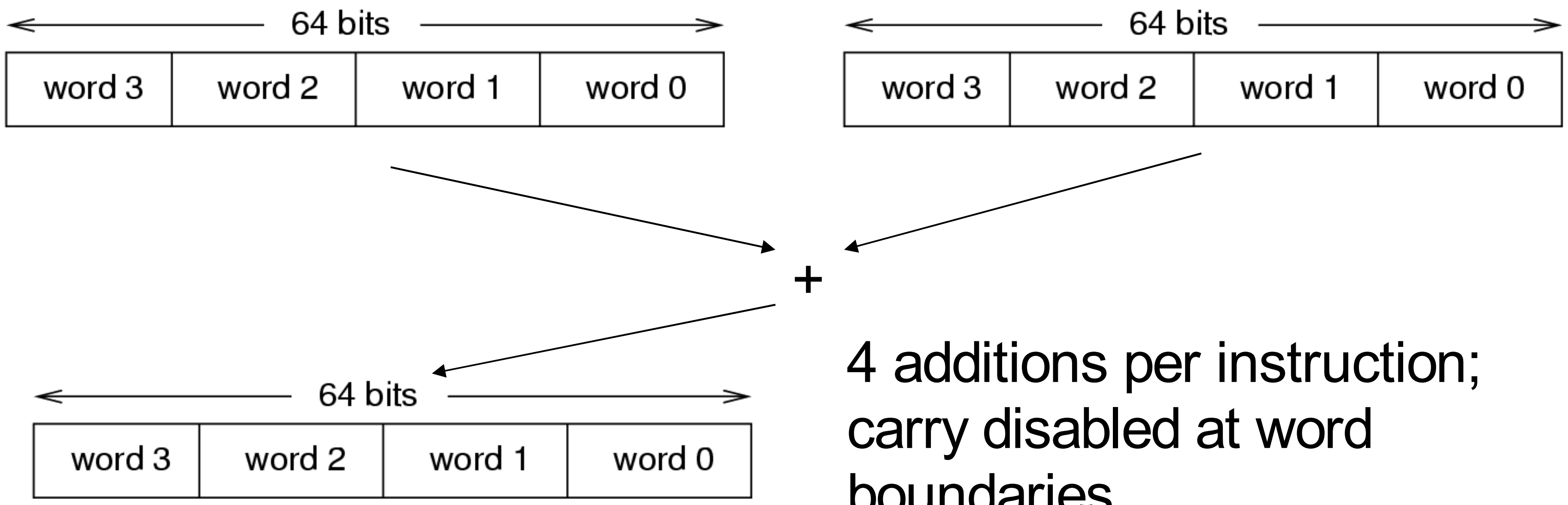
Multiple memory banks or memories


P
D
Address-registers A0, A1, A2 ..
Address generation unit (AGU)
AX
AY
AF
+,-,..
AR
MX
MY
MF
*
+,-
MR

Simplifies parallel fetches

Multimedia-Instructions/Processors

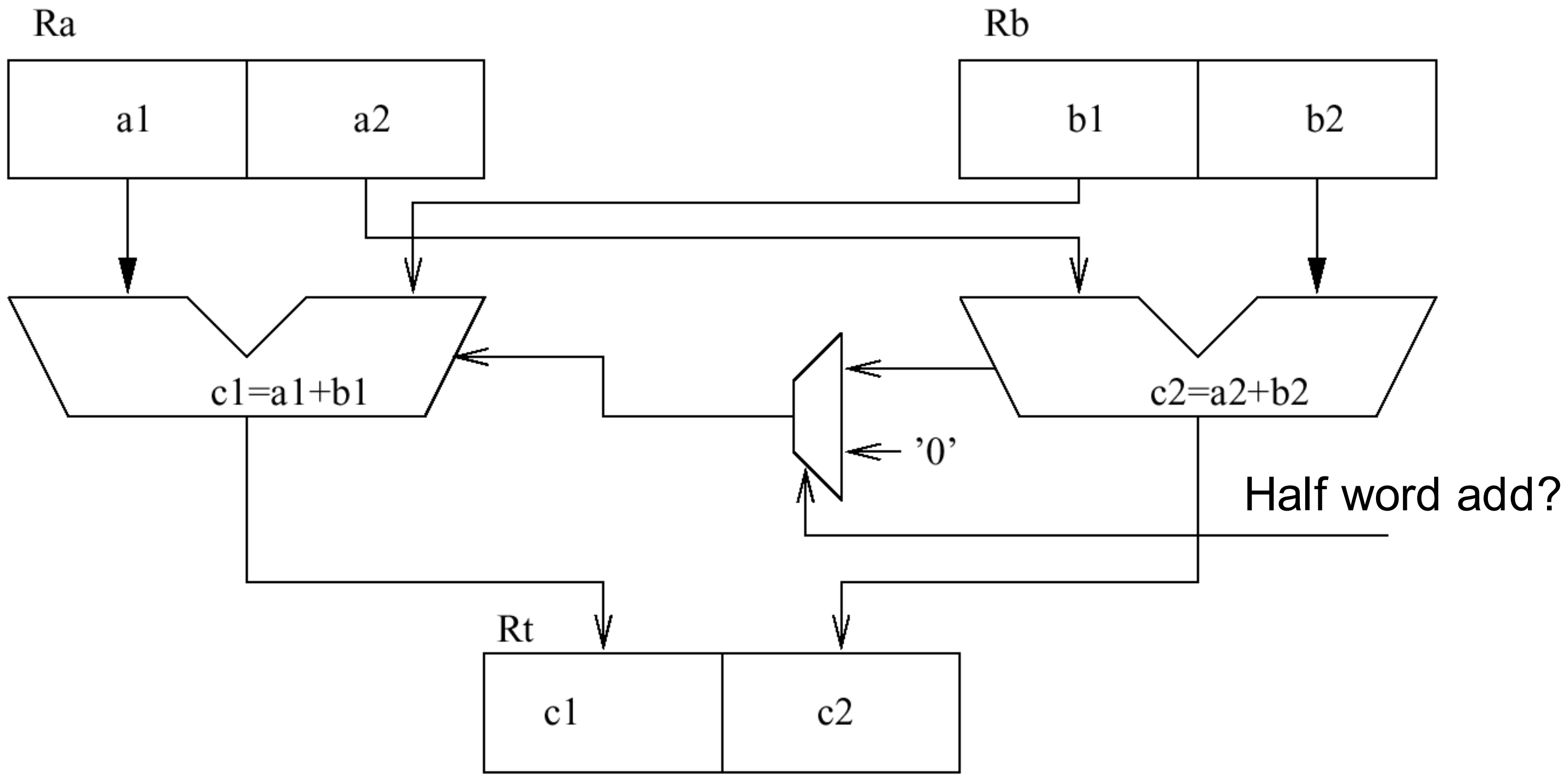
Multimedia instructions exploit that many registers, adders etc are quite wide (32/64 bit),
whereas most multimedia data types are narrow
(e.g. 8 bit per color, 16 bit per audio sample per channel)
☞ 2-8 values can be stored per register and added. E.g.:



4 additions per instruction; carry disabled at word boundaries.

Early example: HP *precision architecture* (hp PA)

Half word add instruction **`HADD`**:



Optional saturating arithmetic.

Up to 10 instructions can be replaced by **`HADD`**.

Pentium MMX-architecture (1)

64-bit vectors representing 8 byte encoded, 4 word encoded or 2 double word encoded numbers.

wrap around/*saturating* options.

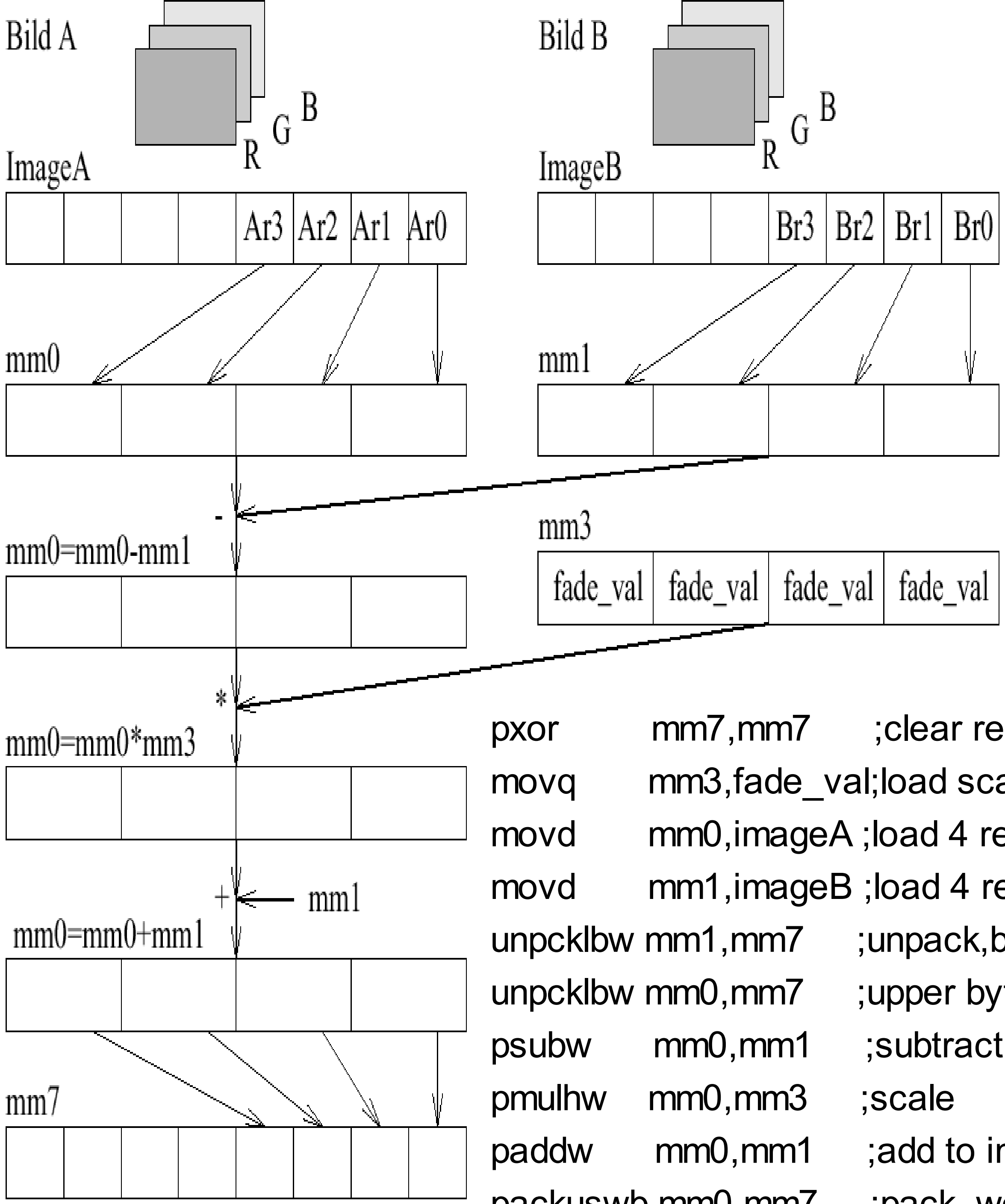
Multimedia registers mm0 - mm7,
consistent with floating-point registers (OS unchanged).

Instruction	Options	Comments
Padd[b/w/d] PSub[b/w/d]	*wrap around,* *saturating*	addition/subtraction of bytes, words, double words
Pcmpeq[b/w/d] Pcmpgt[b/w/d]		Result= "11..11" if true, "00..00" otherwise Result= "11..11" if true, "00..00" otherwise
Pmullw Pmulhw		multiplication, 4*16 bits, least significant word multiplication, 4*16 bits, most significant word

Pentium MMX-architecture (2)

Psra[w/d] Psll[w/d/q] Psrl[w/d/q]	No. of positions in register or instruction	Parallel shift of words, double words or 64 bit quad words
Punpckl[bw/wd/dq] Punpckh[bw/wd/dq]		Parallel unpack Parallel unpack
Packss[wb/dw]	*saturating*	Parallel pack
Pand, Pandn Por, Pxor		Logical operations on 64 bit words
Mov[d/q]		Move instruction

Appli-cation



Scaled interpolation between two images

Next word = next pixel, same color.

4 pixels processed at a time.

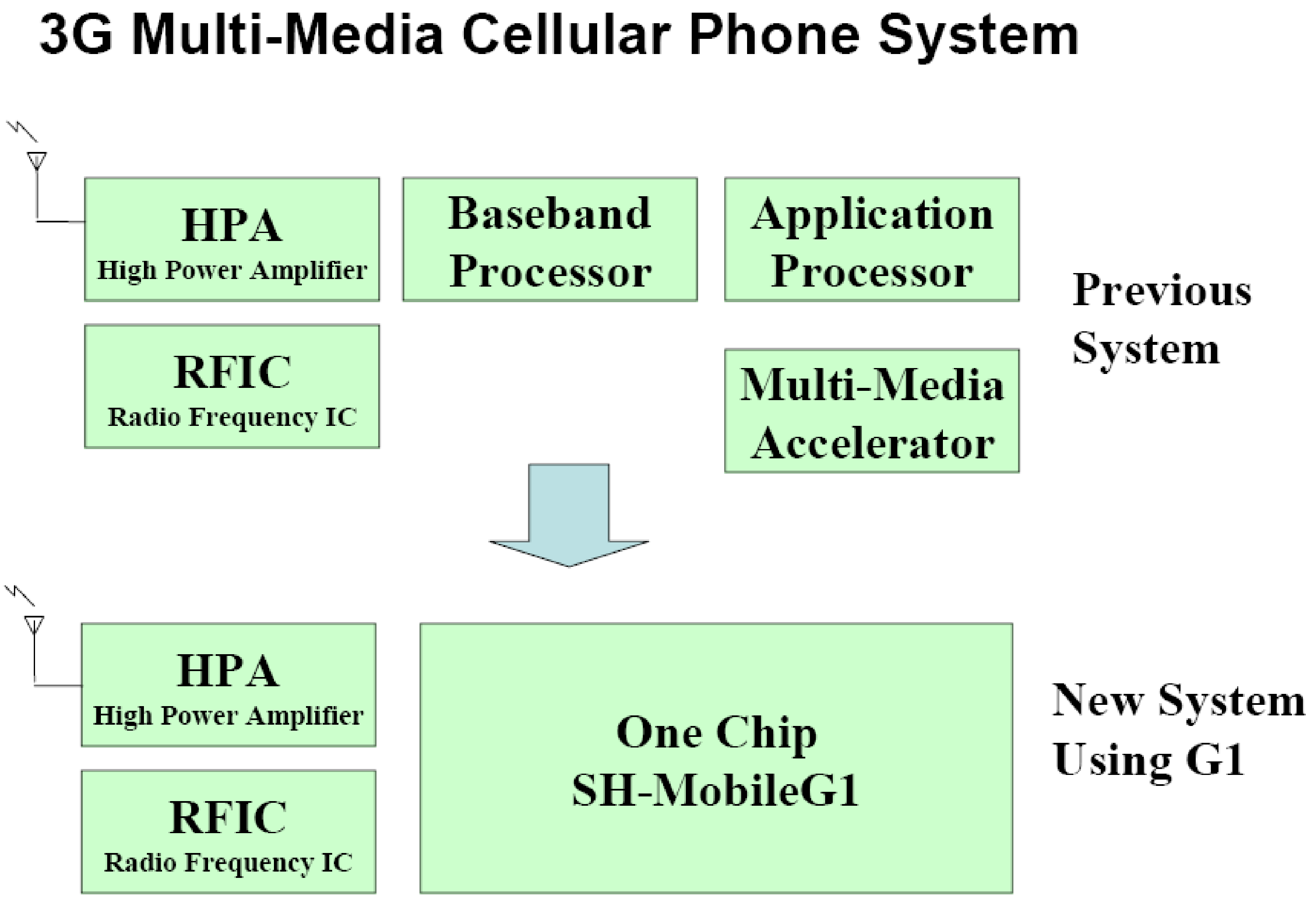
```
pxor       mm7,mm7        ;clear register mm7
movq       mm3,fade_val;load scaling value
movd       mm0,imageA ;load 4 red pixels for A
movd       mm1,imageB ;load 4 red pixels for B
unpcklbw mm1,mm7       ;unpack,bytes to words
unpcklbw mm0,mm7       ;upper bytes from mm7
psubw      mm0,mm1       ;subtract pixel values
pmulhw    mm0,mm3       ;scale
paddw      mm0,mm1       ;add to image B
packuswb mm0,mm7       ;pack, words to bytes
```

Ultra-SPARC Processor

visual instruction set (VIS)

- Instruction for MPEG motion estimation,
  includes 8 subtractions, 8 additions and 8 absolute value computations on 8 bit data in a single cycle.
  Replaces up to 1500 instructions by 32 of such instructions.
- ..

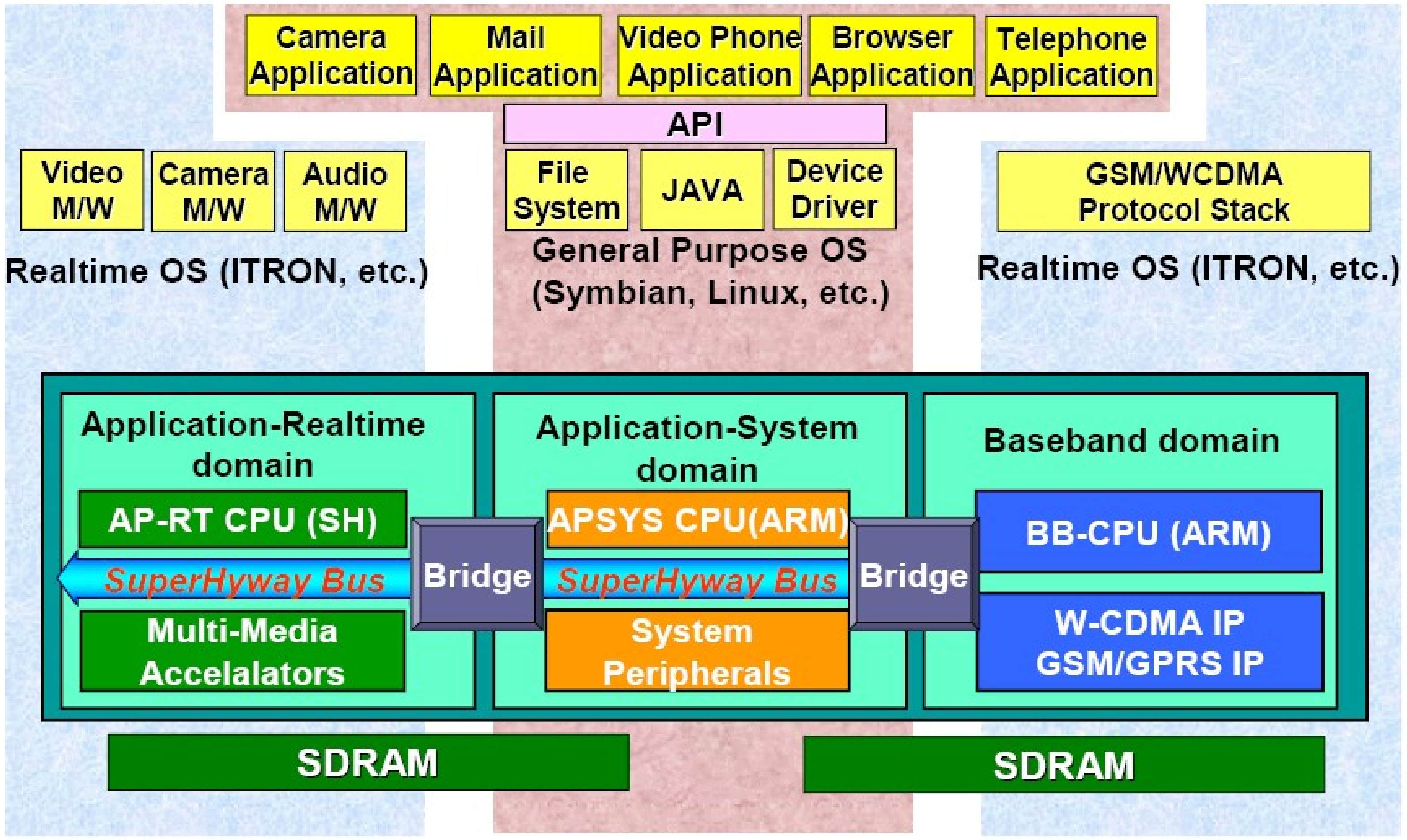
Trend: multiprocessor systems-on-a-chip (MPSoCs)



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (2)

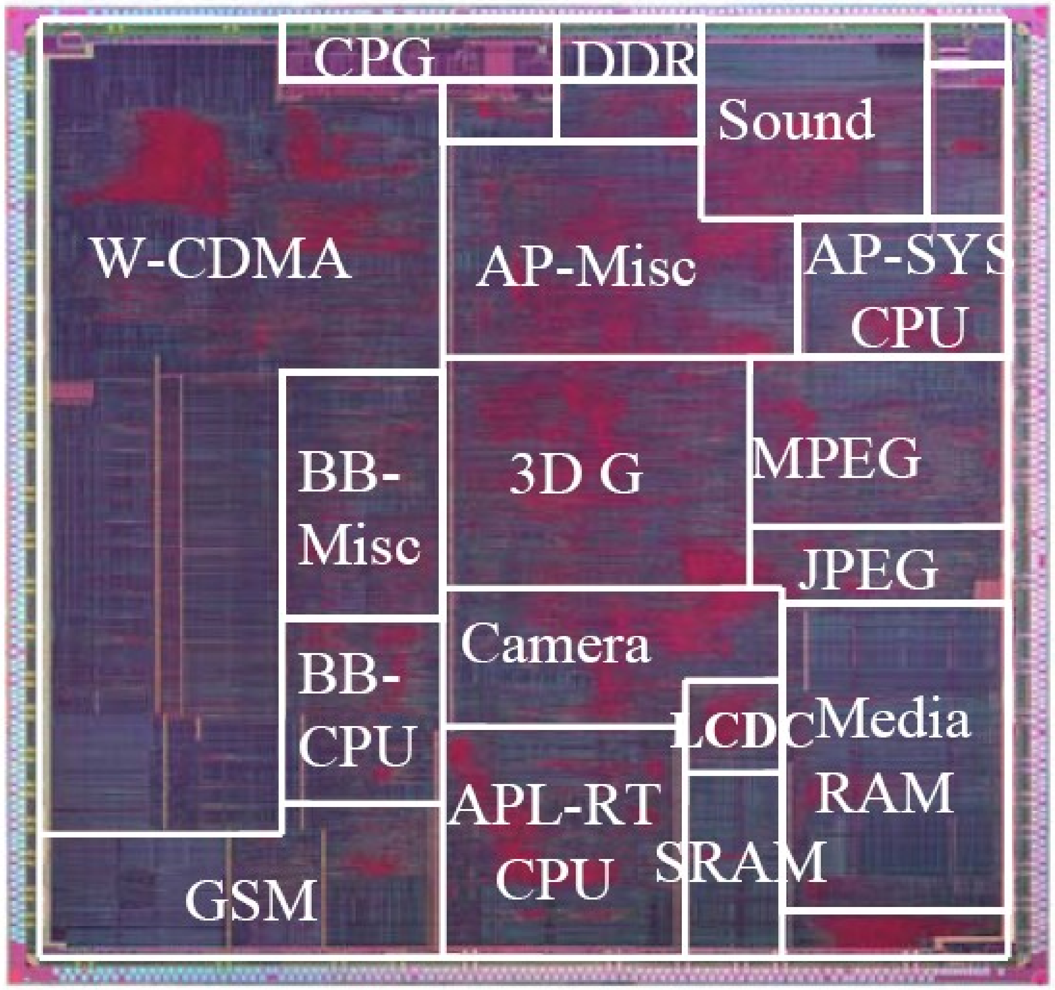
A Sample of System Architecture using G1



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (3)

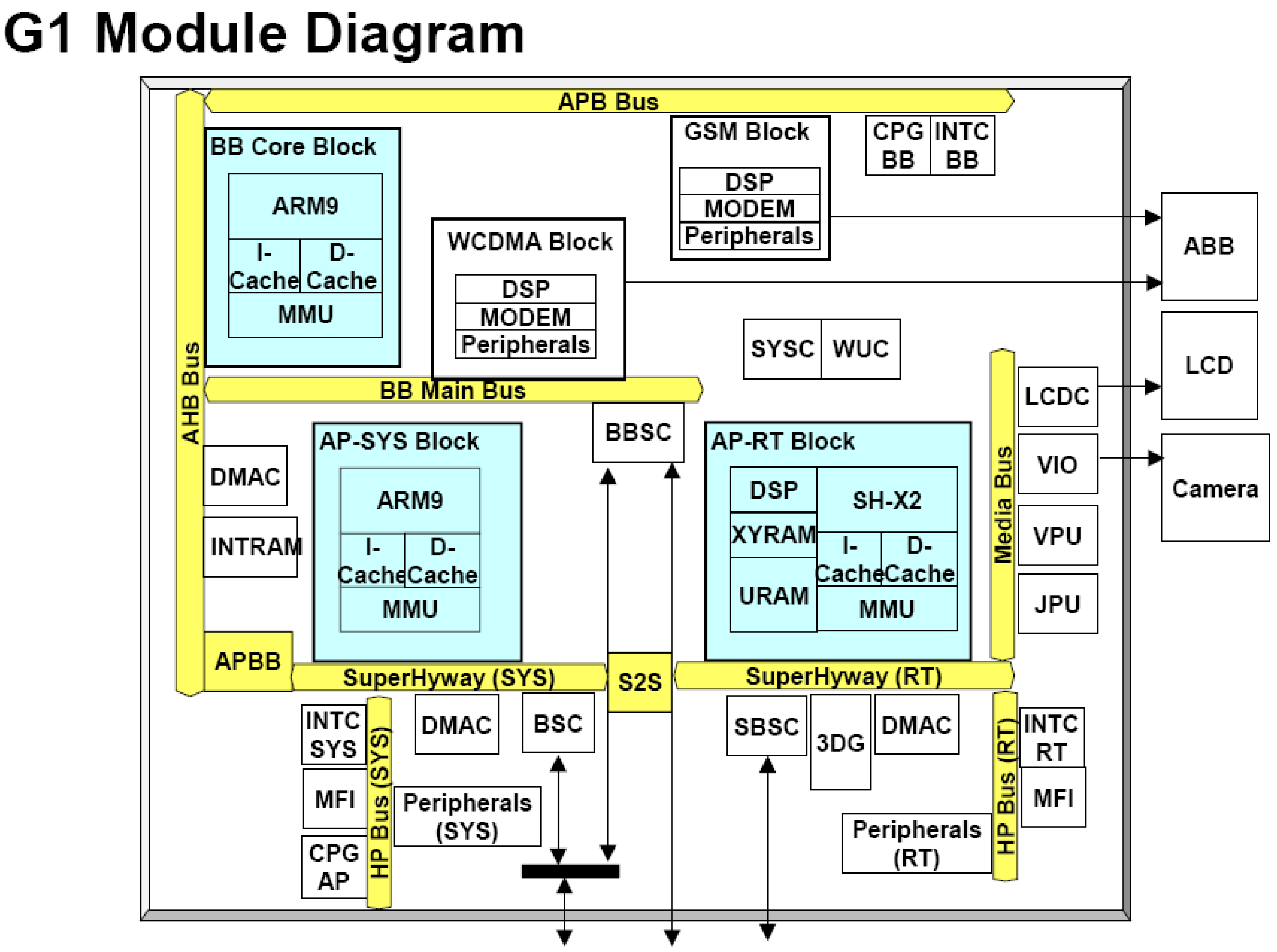
SH-MobileG1: Chip Overview



Die size	11.15mm x 11.15mm
Process	90nm LP 8M(7Cu+1Al) CMOS dual-Vth
Supply voltage	1.2V(internal), 1.8/2.5/3.3V(I/O)
# of TRs, gate, memory	181M TRs, 13.5M Gate 20.2 Mbit mem

http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (4)

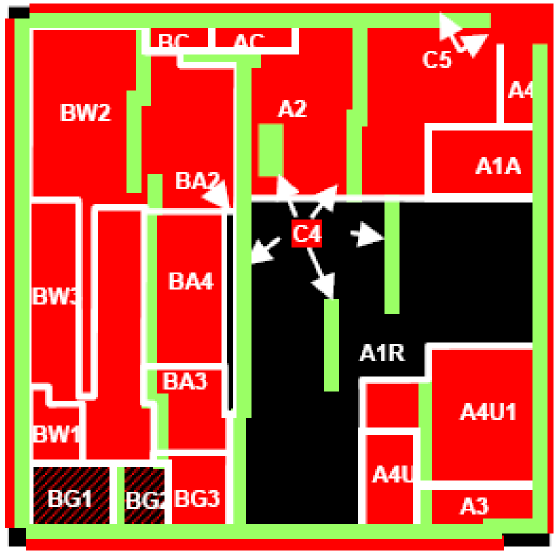


http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (5)

Leakage Current in Usage Scenes

(2)Telephony (W-CDMA)



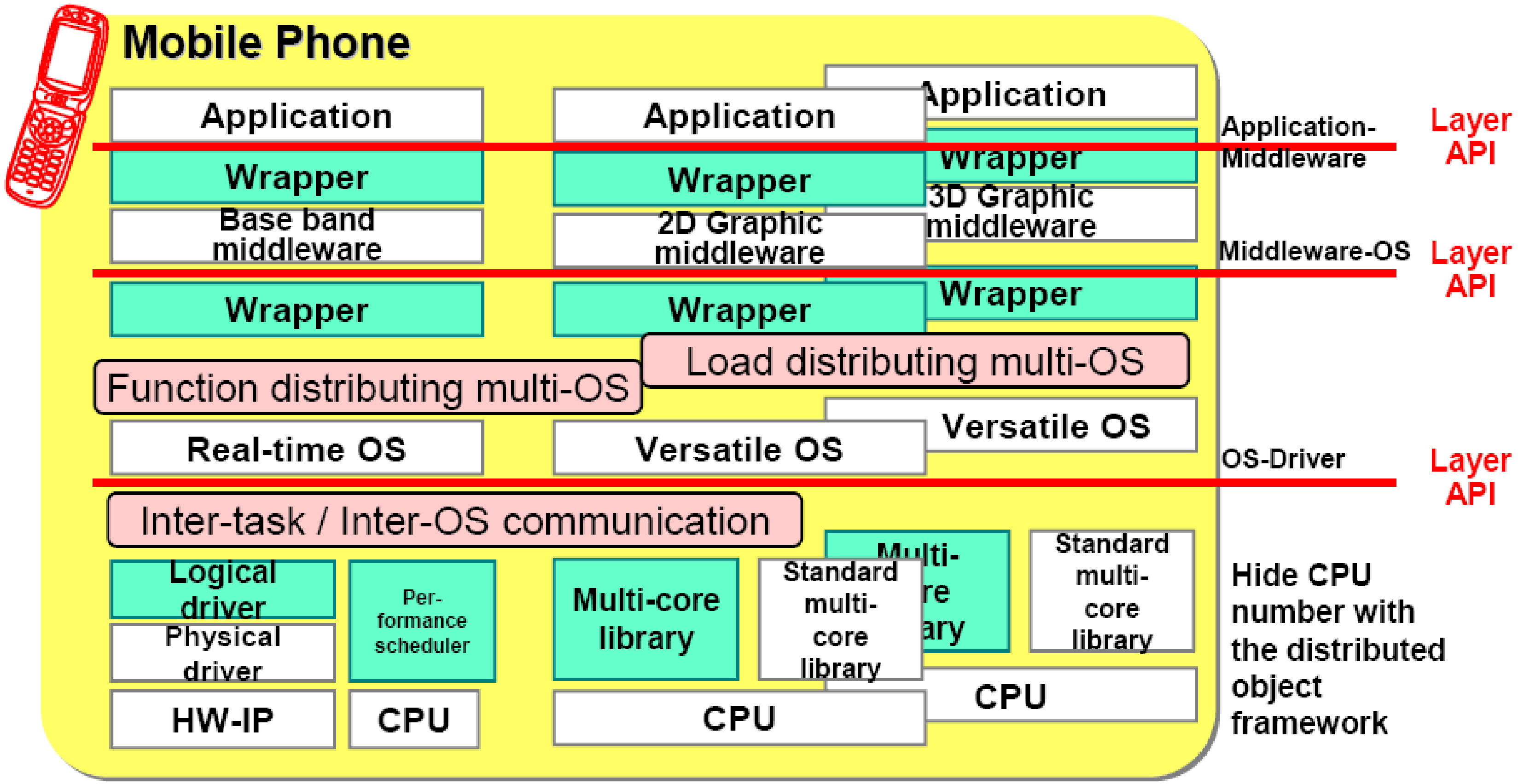
🟥	Power on
⬛	Power off

Baseband part	Control	ON
	W-CDMA	ON
	GSM	ON / OFF
Application part	System-domain	ON
	Realtime-domain	OFF
Measured Leakage Current (@ Room Temp, 1.2V)		407 μA

Multiprocessor systems-on-a-chip (MPSoCs) (6)

EXREAL Platform™ Software Interconnect

- Promote reuse of software assets through Wrapper and standardized layer API
- Control operating frequency and power on/off through the performance scheduler



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Embedded System Hardware

- Reconfigurable Hardware -

Peter Marwedel
Informatik 12
TU Dortmund
Germany

Reconfigurable Logic

Full custom chips may be too expensive, software too slow.

Combine the speed of HW with the flexibility of SW

- ☞ HW with programmable functions and interconnect.
- ☞ Use of configurable hardware;
  common form: field programmable gate arrays (FPGAs)

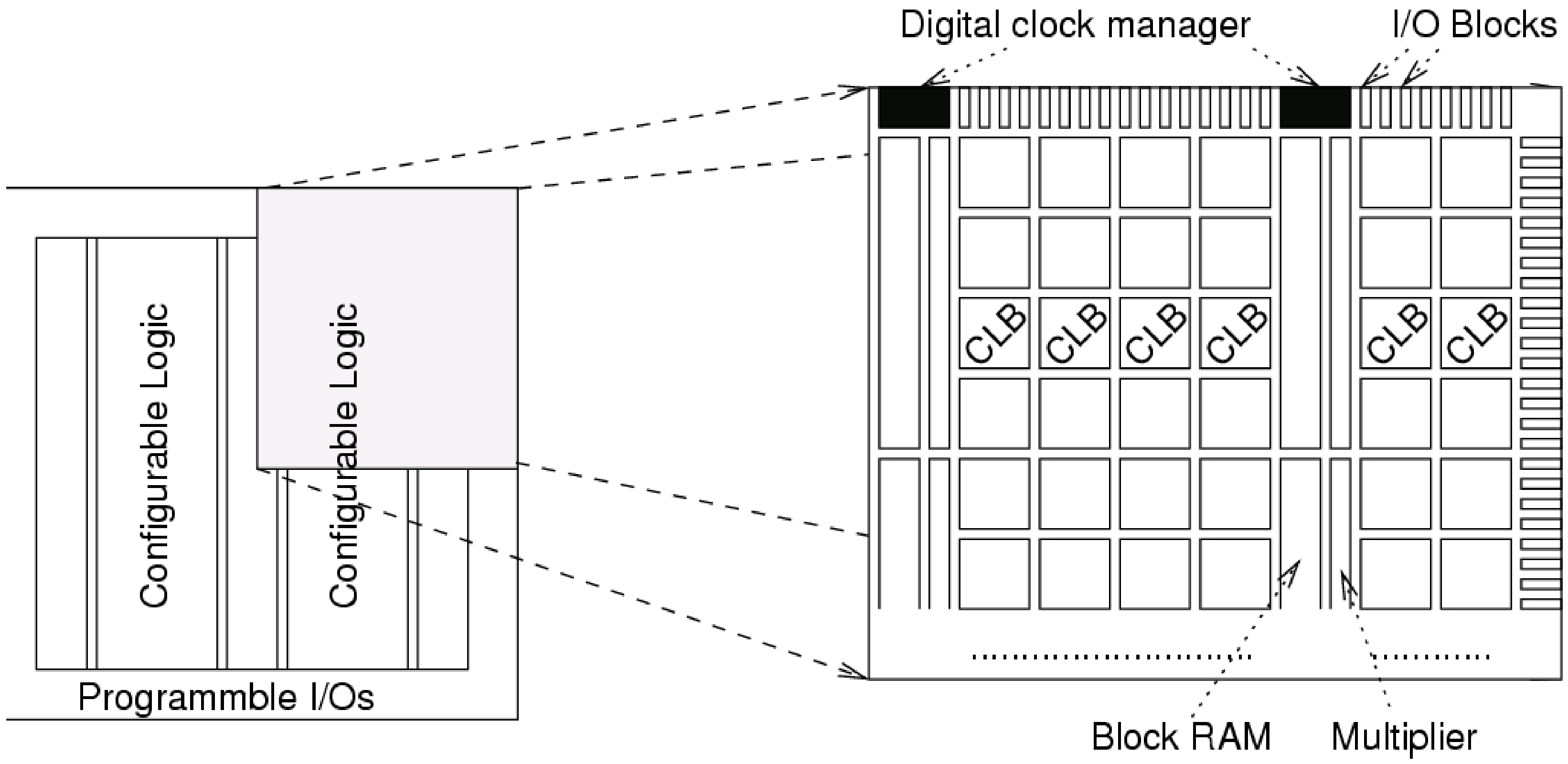
Applications: bit-oriented algorithms like

- encryption,
- fast “object recognition“ (medical and military)
- Adapting mobile phones to different standards.

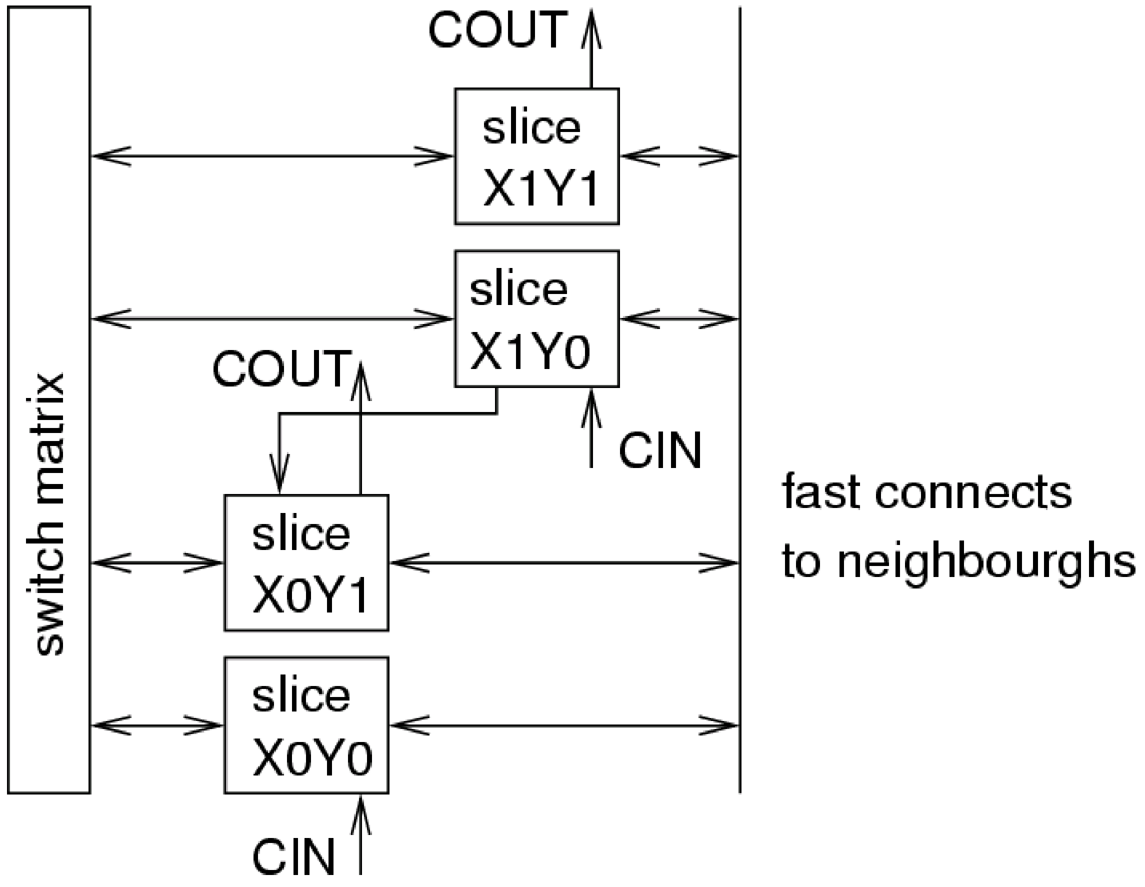
Very popular devices from

- XILINX (XILINX Vertex II are recent devices)
- Actel, Altera and others

Floor-plan of VIRTEX II FPGAs

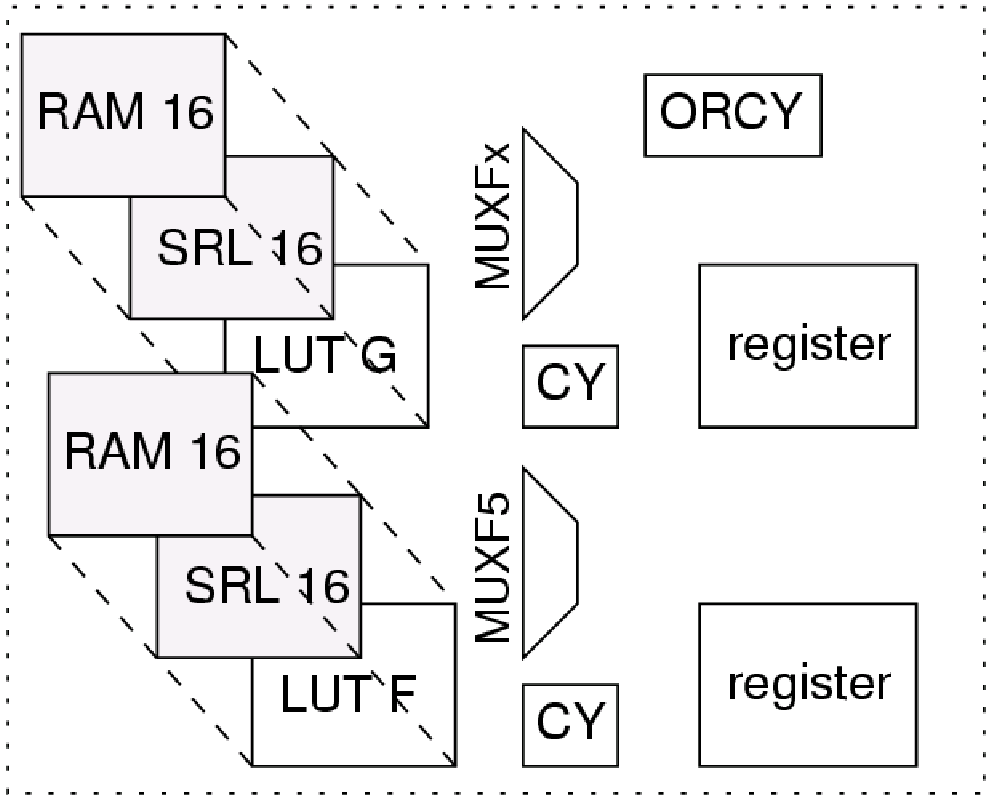


Virtex II Configurable Logic Block (CLB)



Virtex II Slice (simplified)

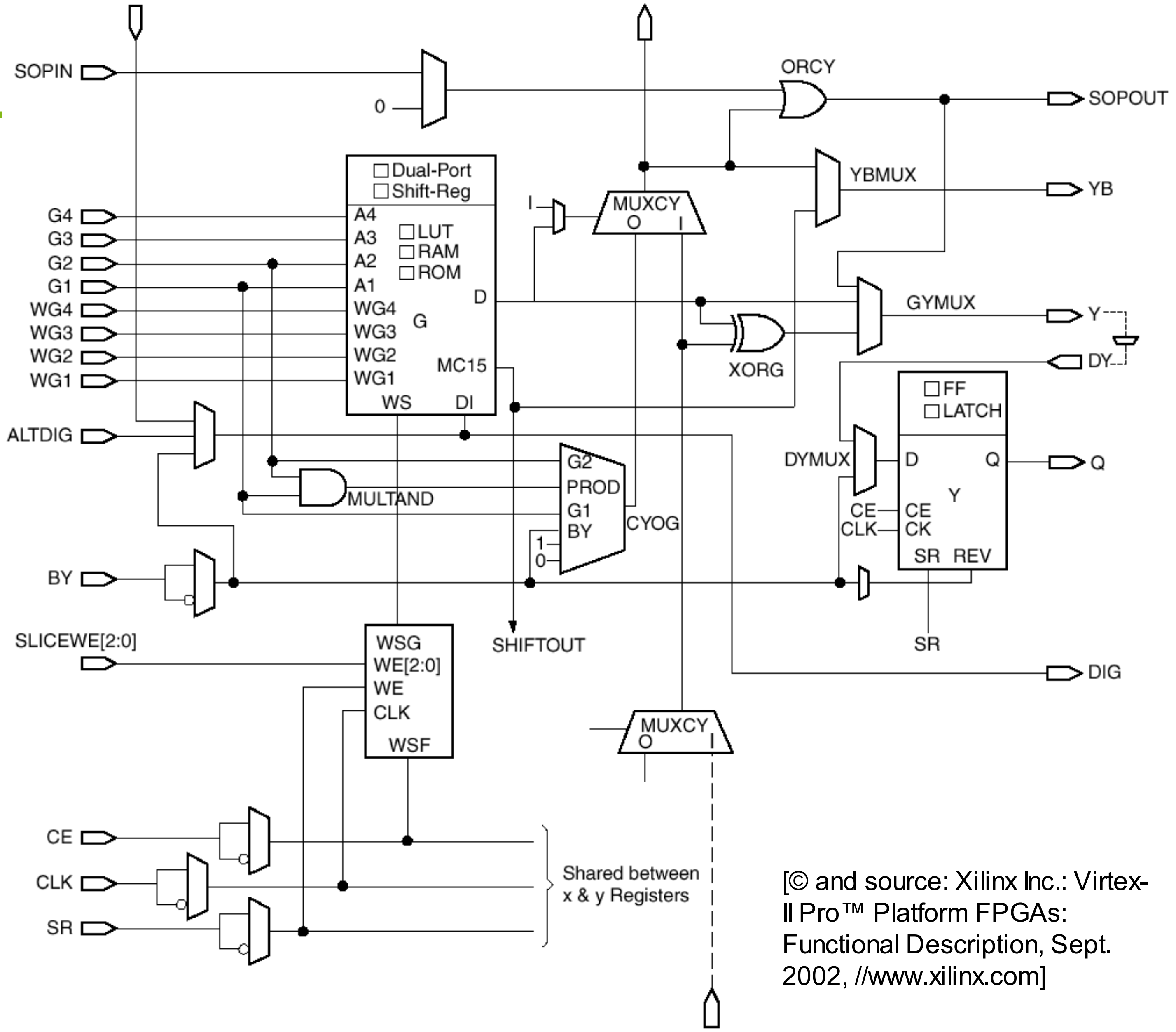
Look-up tables LUT F and G can be used to compute any Boolean function of ≤ 4 variables.



Example:

a	b	c	d	**G**
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

Virtex II (Pro) Slice



[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

2 carry paths per CLB (Vertex II Pro)

Enables efficient implementation of adders.

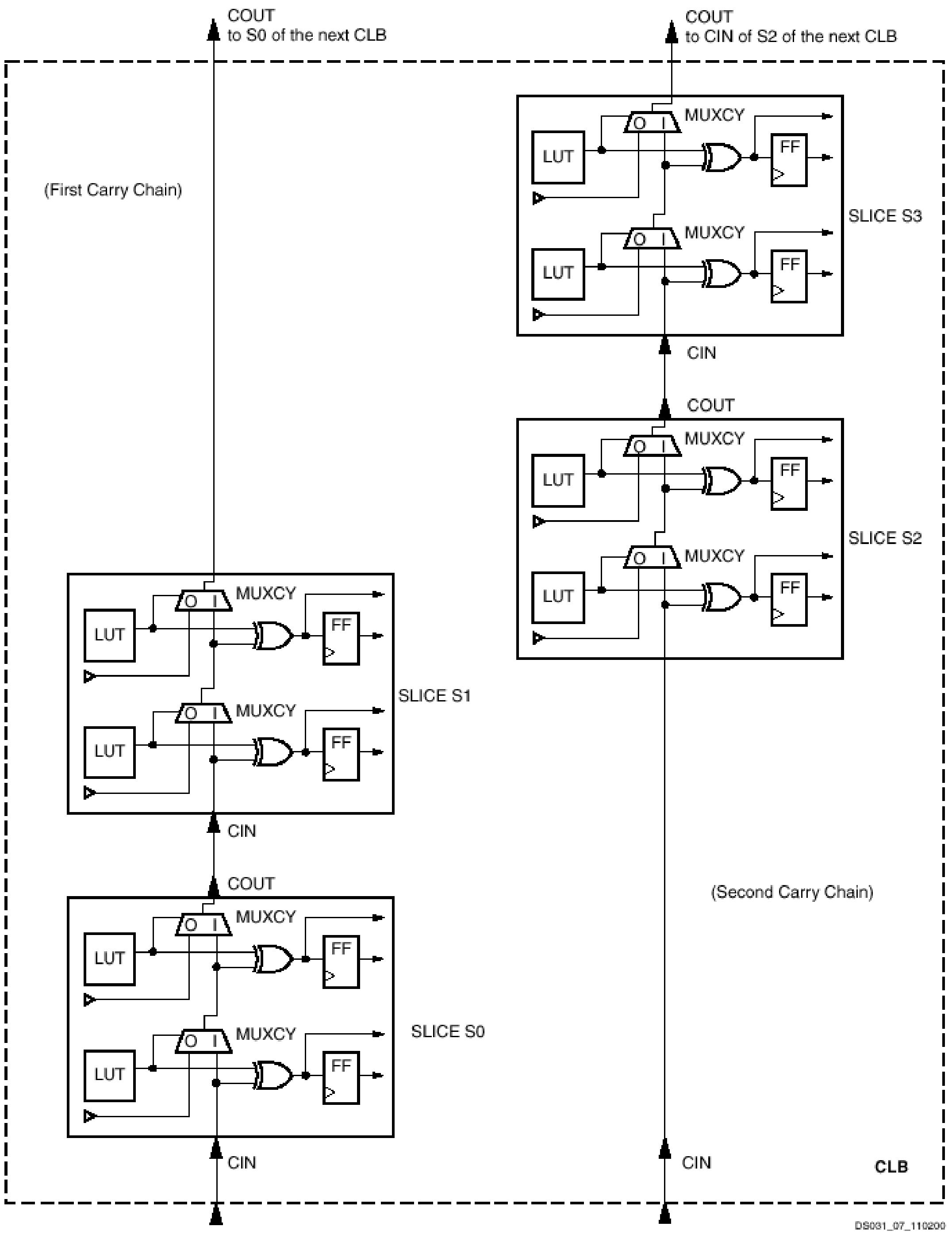


Figure 32: **Fast Carry Logic Path**

[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

Number of resources available in Virtex II Pro devices

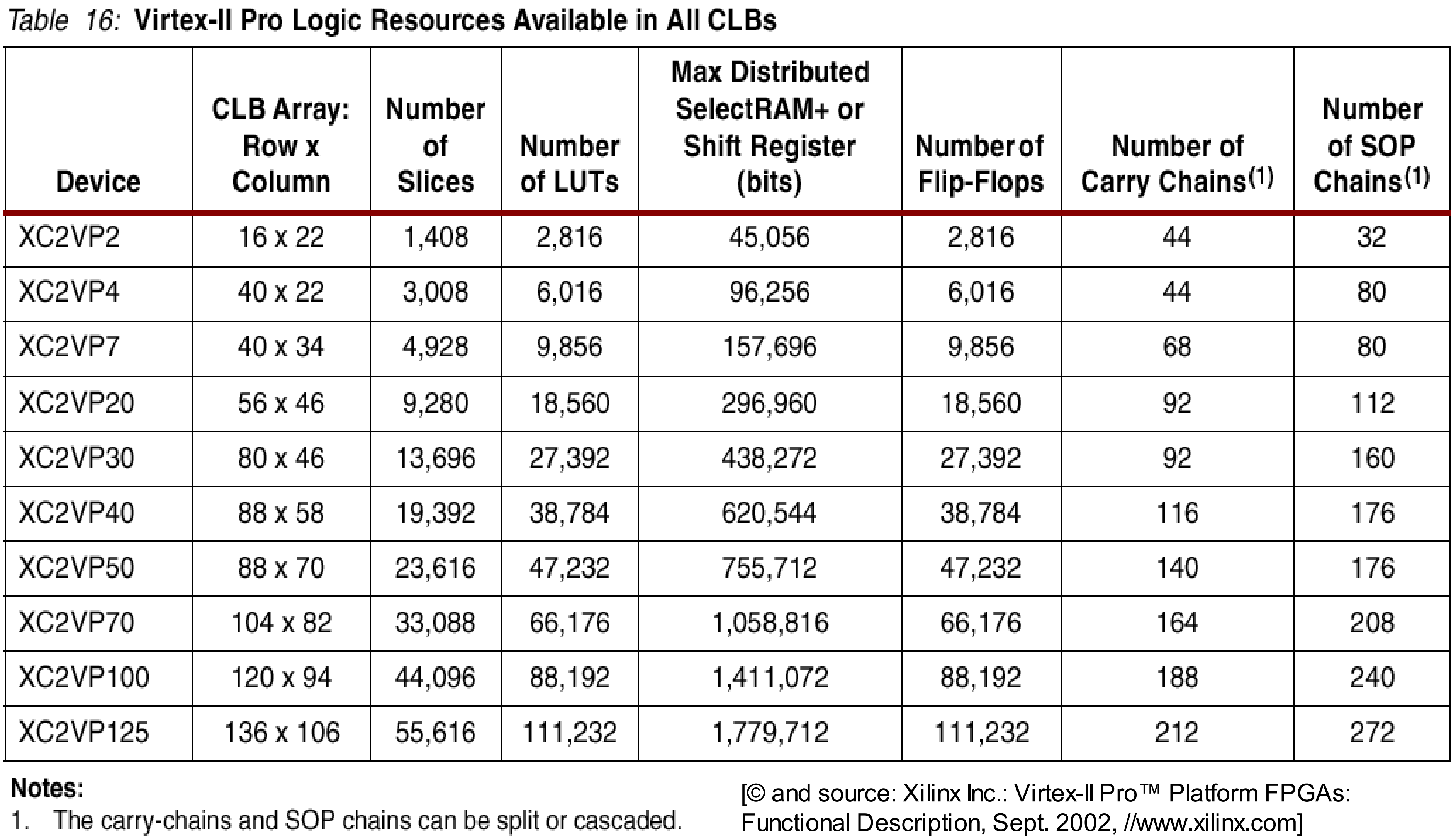


Table 16: **Virtex-II Pro Logic Resources Available in All CLBs**

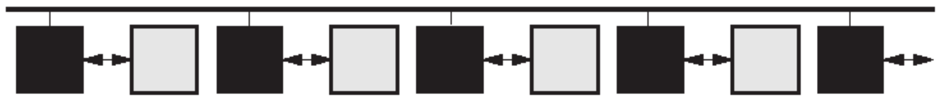
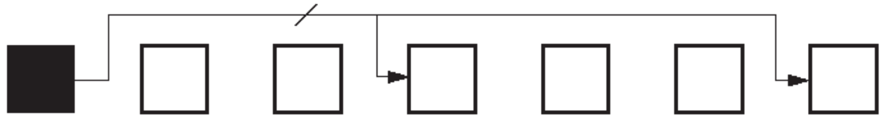
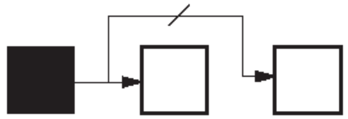
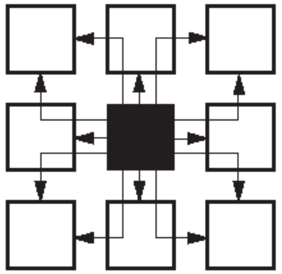
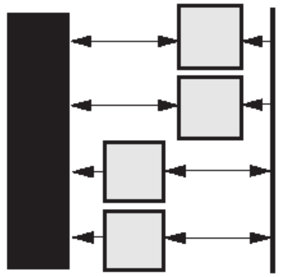
Device	CLB Array: Row x Column	Number of Slices	Number of LUTs	Max Distributed SelectRAM+ or Shift Register (bits)	Number of Flip-Flops	Number of Carry Chains[1]	Number of SOP Chains[1]
XC2VP2	16 x 22	1,408	2,816	45,056	2,816	44	32
XC2VP4	40 x 22	3,008	6,016	96,256	6,016	44	80
XC2VP7	40 x 34	4,928	9,856	157,696	9,856	68	80
XC2VP20	56 x 46	9,280	18,560	296,960	18,560	92	112
XC2VP30	80 x 46	13,696	27,392	438,272	27,392	92	160
XC2VP40	88 x 58	19,392	38,784	620,544	38,784	116	176
XC2VP50	88 x 70	23,616	47,232	755,712	47,232	140	176
XC2VP70	104 x 82	33,088	66,176	1,058,816	66,176	164	208
XC2VP100	120 x 94	44,096	88,192	1,411,072	88,192	188	240
XC2VP125	136 x 106	55,616	111,232	1,779,712	111,232	212	272

Notes:
1. The carry-chains and SOP chains can be split or cascaded.

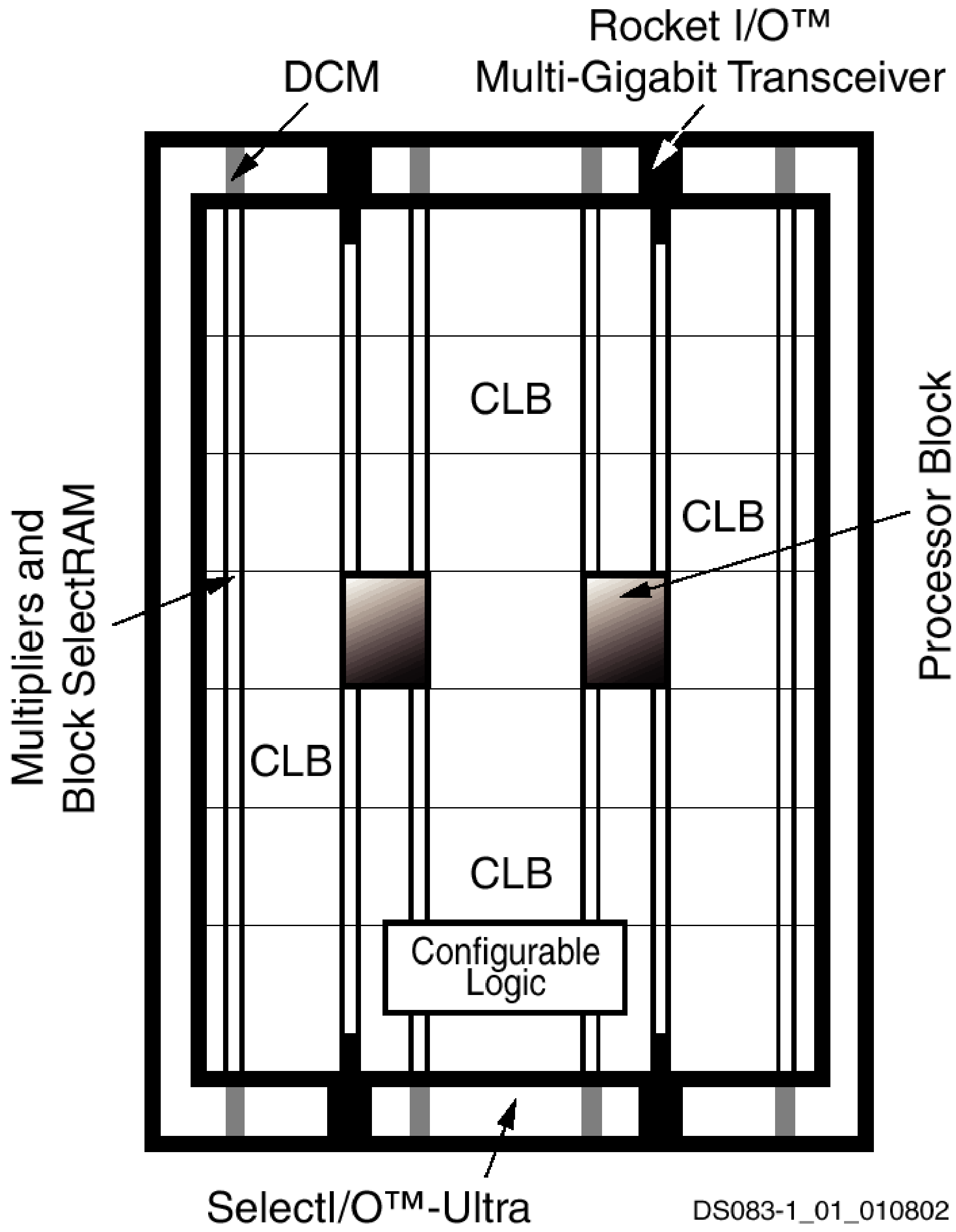
[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

Interconnect

Hierarchical Routing Resources

24 Horizontal Long Lines 24 Vertical Long Lines	
120 Horizontal Hex Lines 120 Vertical Hex Lines	
40 Horizontal Double Lines 40 Vertical Double Lines	
16 Direct Connections (total in all four directions)	
8 Fast Connects	

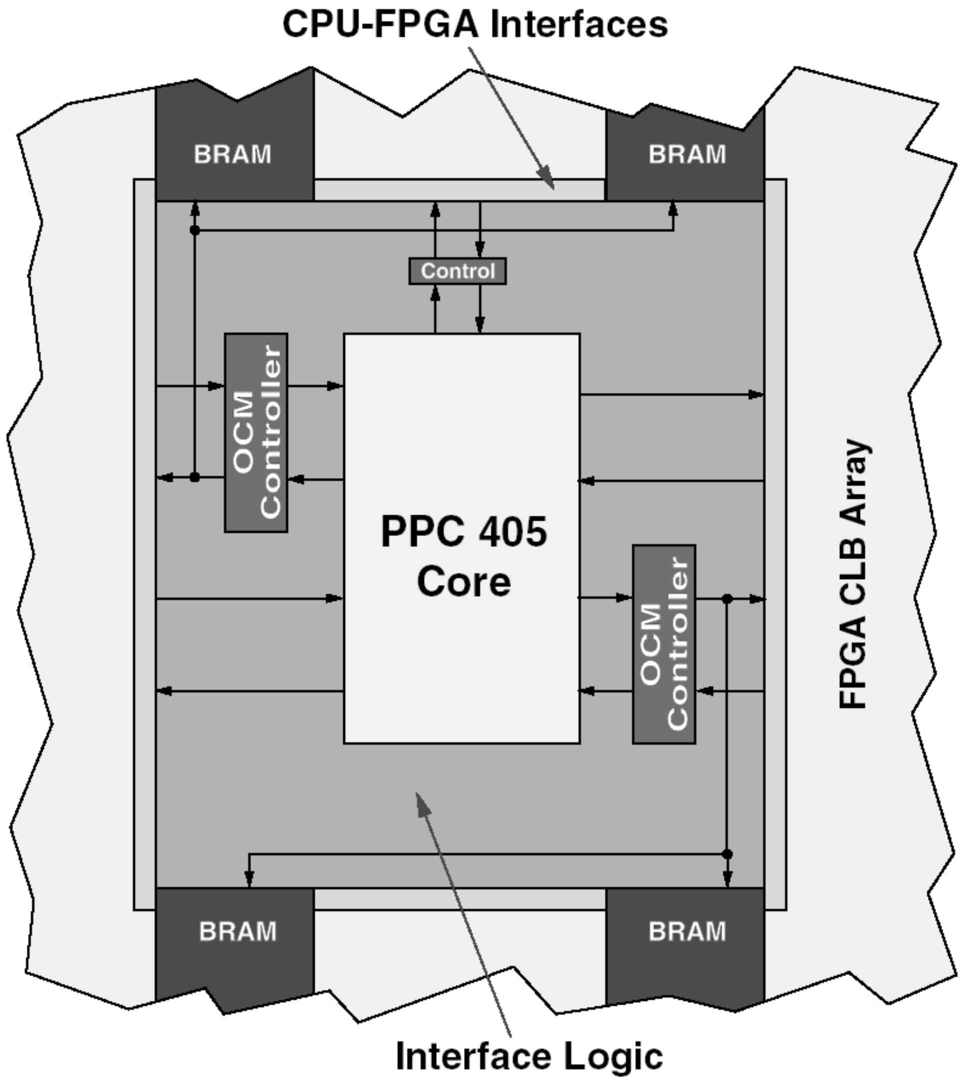
Virtex II Pro Devices include up to 4 PowerPC processor cores



[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

Memory for processor cores

Cores are connected to local block RAM that can be used as a scratchpad.



Summary

Hardware in a loop

- Sensors
- Discretization
- Information processing
  - Importance of energy efficiency
  - Special purpose HW very expensive
  - Energy efficiency of processors
  - Code size efficiency
  - Run-time efficiency
  - Reconfigurable Hardware
- D/A converters
- Actuators