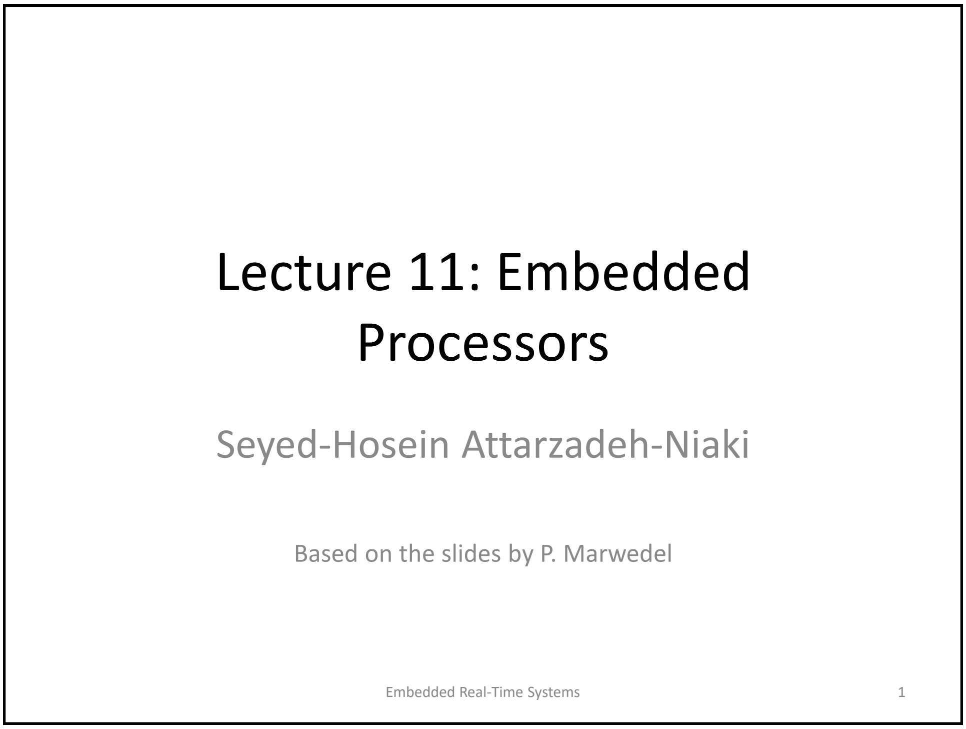
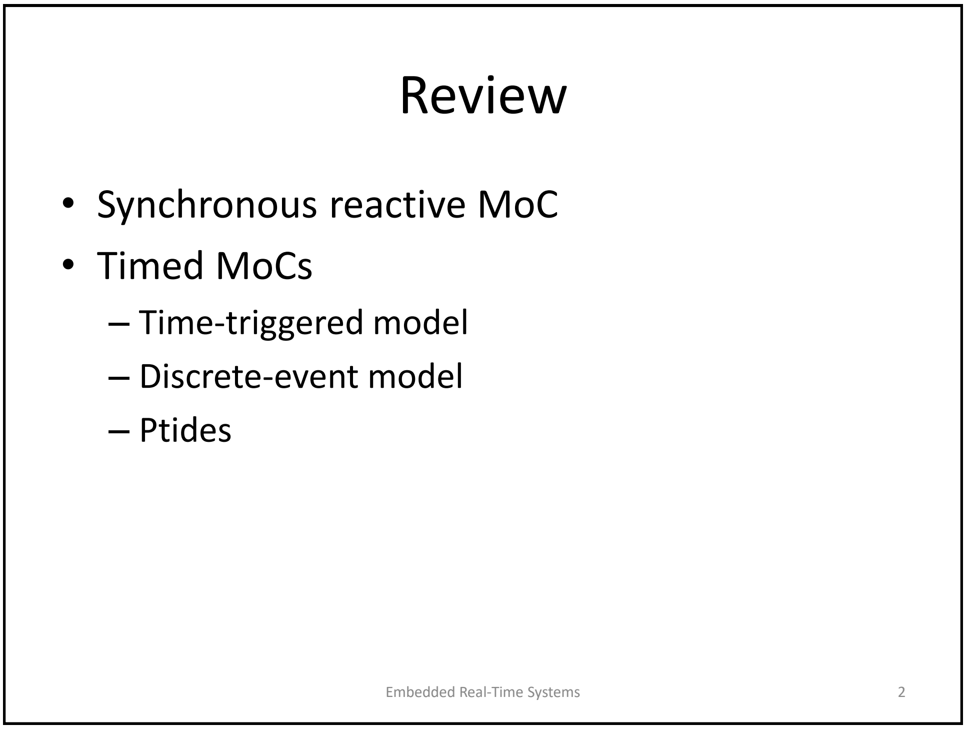


Lecture 11: Embedded Processors

Seyed-Hosein Attarzadeh-Niaki

Based on the slides by P. Marwedel



Review

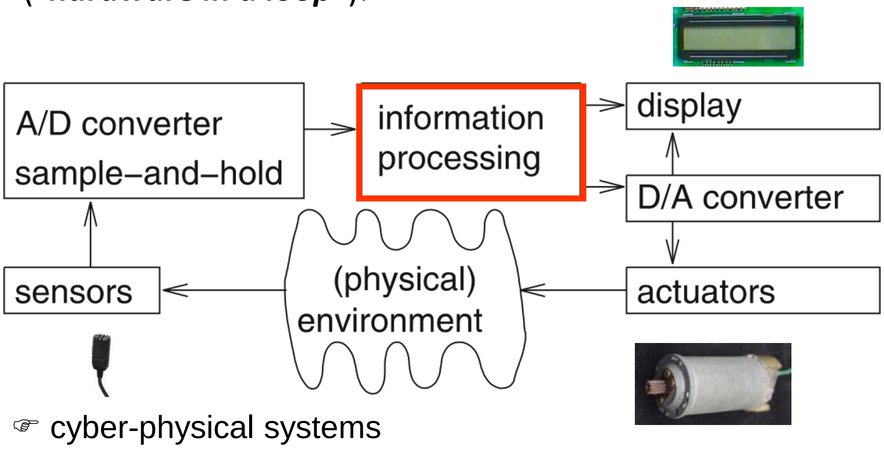
- Synchronous reactive MoC
- Timed MoCs
  - Time-triggered model
  - Discrete-event model
  - Ptides

Outline

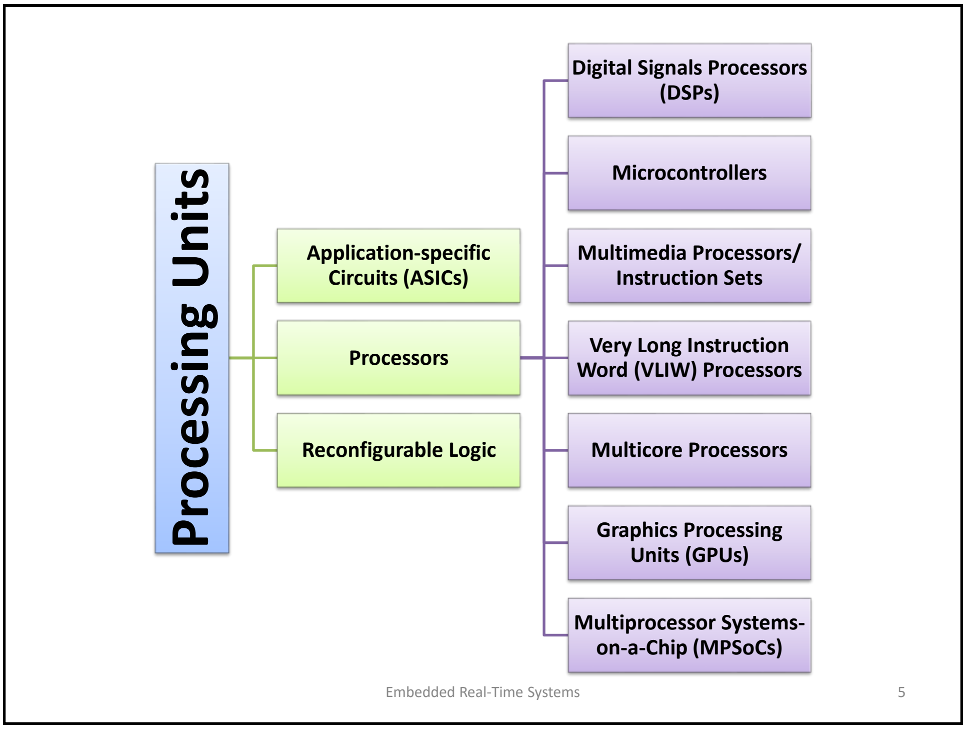
- Types of processing units
- Efficiency of embedded processors
  - Power/energy efficiency
  - Code size efficiency
  - Runtime efficiency
- Realtime capability

Embedded & CPS System Hardware

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



☞ cyber-physical systems



Processing Units

Application-specific Circuits (ASICs)

Processors

Reconfigurable Logic

Digital Signals Processors (DSPs)

Microcontrollers

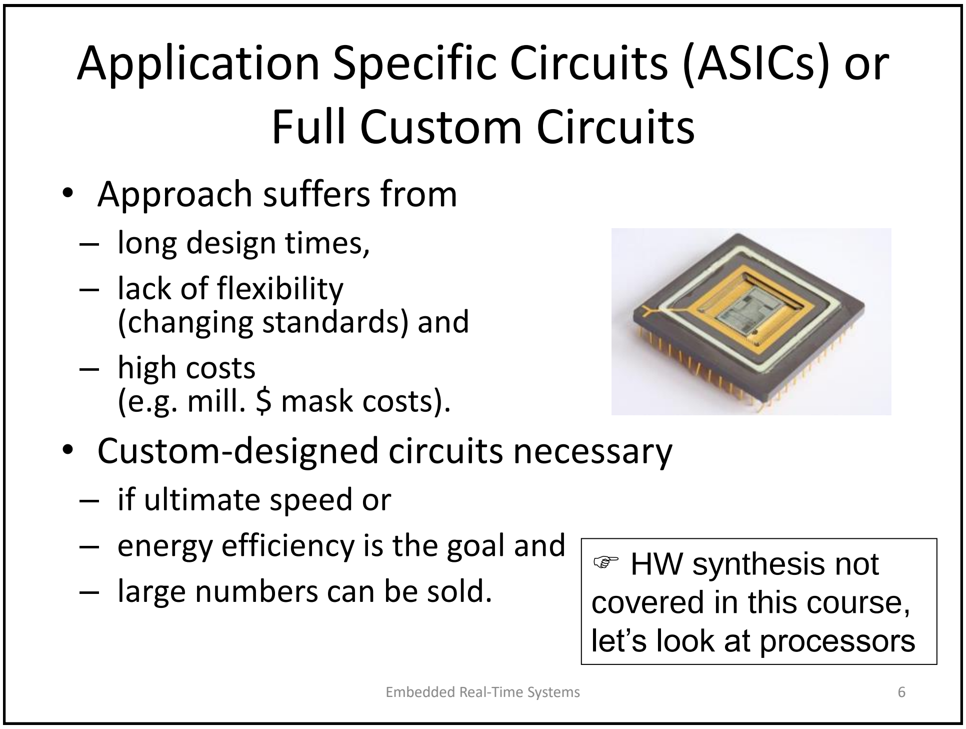
Multimedia Processors/ Instruction Sets

Very Long Instruction Word (VLIW) Processors

Multicore Processors

Graphics Processing Units (GPUs)

Multiprocessor Systems-on-a-Chip (MPSoCs)



Application Specific Circuits (ASICs) or Full Custom Circuits

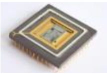
- Approach suffers from
  - long design times,
  - lack of flexibility (changing standards) and
  - high costs (e.g. mill. $ mask costs).
- Custom-designed circuits necessary
  - if ultimate speed or
  - energy efficiency is the goal and
  - large numbers can be sold.

☞ HW synthesis not covered in this course, let’s look at processors

Efficiency: Applied to Processing

– CPS & ES must be **efficient**

➡ • Code-size efficient (especially for systems on a chip)



➡ • Run-time efficient

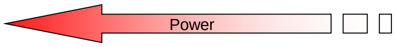
• Weight efficient

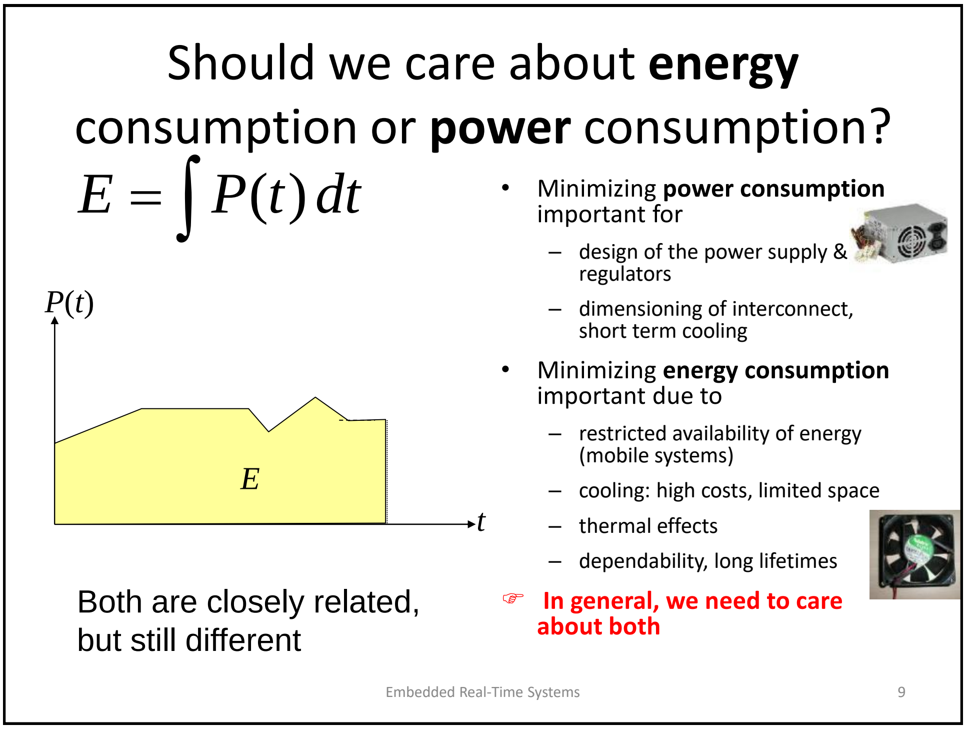
• Cost efficient

➡ • Energy efficient

Why care about energy efficiency ?

	Relevant during use?		
Execution platform	Plugged	Uncharged periods	Unplug-ged
E.g.	Factory	Car	Sensor
Global warming	☑	□	□
Cost of energy	☑	□	□
Increasing performance	☑	☑	☑
Problems with cooling, avoiding hot spots	☑	☑	☑
Avoiding high currents & metal migration	☑	☑	☑
Reliability	☑	☑	☑
Energy a very scarce resource	□	☑	☑





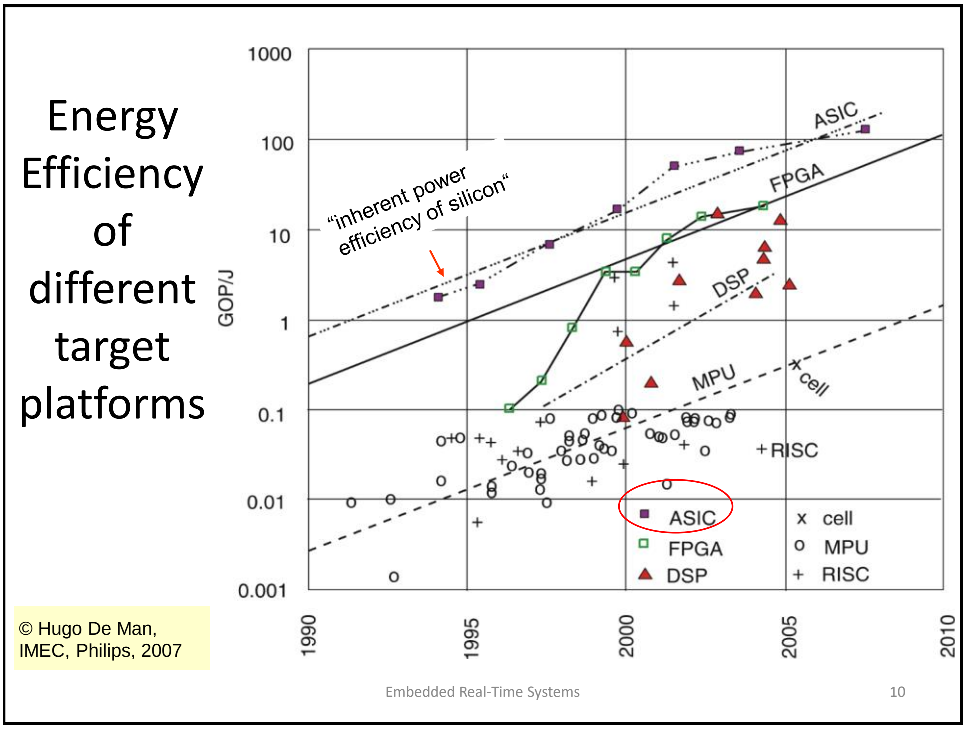
Should we care about **energy** consumption or **power** consumption?

$$E = \int P(t)\,dt$$

Both are closely related, but still different

- Minimizing **power consumption** important for
  - design of the power supply & regulators
  - dimensioning of interconnect, short term cooling
- Minimizing **energy consumption** important due to
  - restricted availability of energy (mobile systems)
  - cooling: high costs, limited space
  - thermal effects
  - dependability, long lifetimes

☞ **In general, we need to care about both**



Energy Efficiency of different target platforms

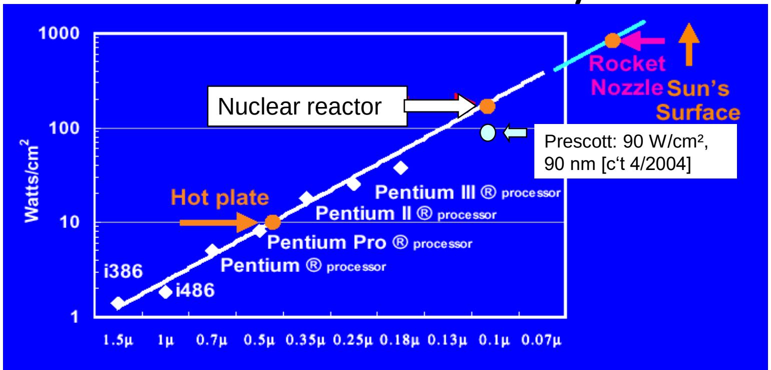
© Hugo De Man, IMEC, Philips, 2007

PCs: Surpassed hot (kitchen) plate …? Why not use it?

Strictly speaking, energy is not “consumed”, but converted from electrical energy into heat energy

http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

PCs
Problem: Power density increasing

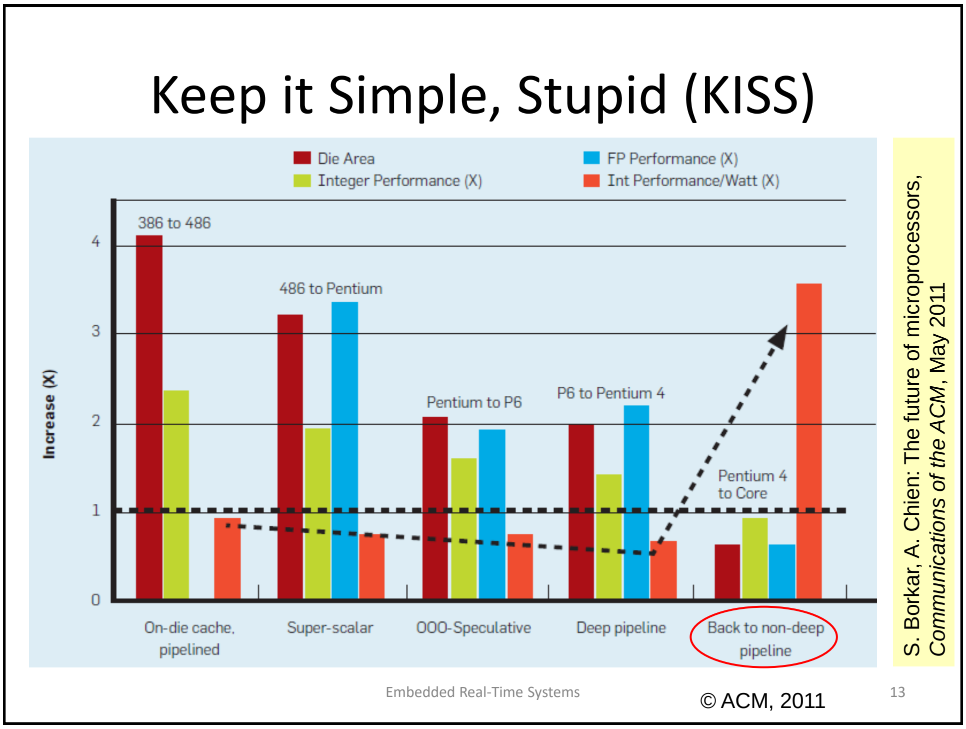


Prescott: 90 W/cm², 90 nm [c‘t 4/2004]

Surpassed hot-plate power density in 0.5µ

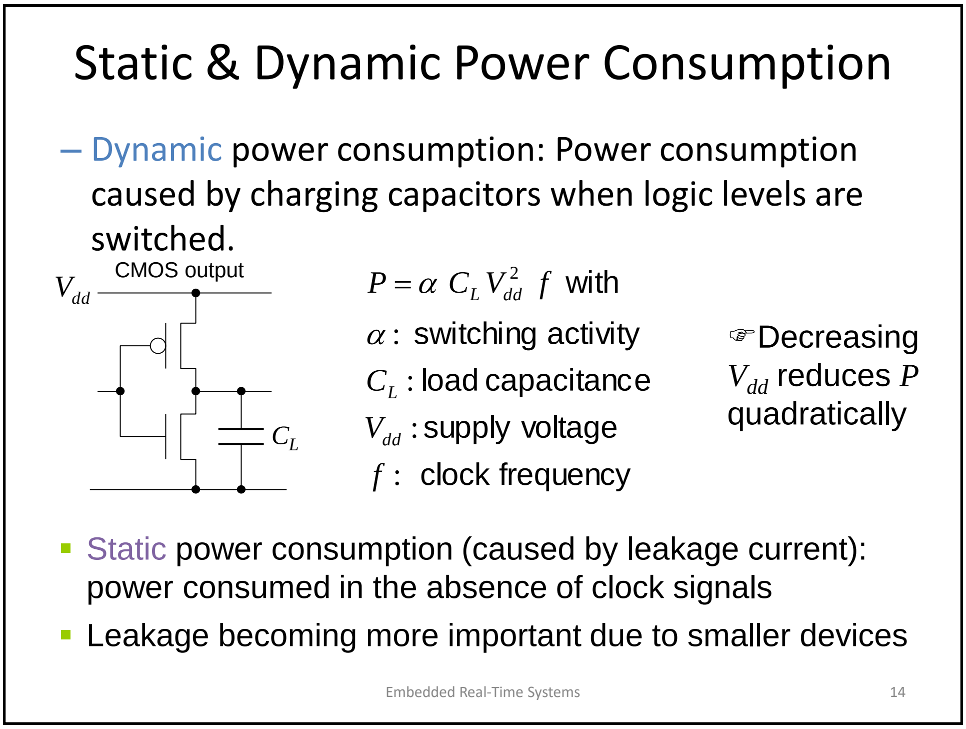
Not too long to reach nuclear reactor

© Intel
M. Pollack,
Micro-32



Keep it Simple, Stupid (KISS)

S. Borkar, A. Chien: The future of microprocessors, *Communications of the ACM*, May 2011



Static & Dynamic Power Consumption

– Dynamic power consumption: Power consumption caused by charging capacitors when logic levels are switched.

$P = \alpha \, C_L V_{dd}^2 \, f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

☞Decreasing V_{dd} reduces P quadratically

- Static power consumption (caused by leakage current): power consumed in the absence of clock signals
- Leakage becoming more important due to smaller devices

Static & Dynamic Power Consumption

Power consumption of CMOS circuits (ignoring leakage):

$P = \alpha \; C_L V_{dd}^2 \; f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

$\tau = k \, C_L \dfrac{V_{dd}}{(V_{dd} - V_t)^2}$ with

V_t : threshhold voltage

$(V_t <$ than $V_{dd})$

☞Decreasing V_{dd} reduces P quadratically,
while the run-time of algorithms is only linearly increased

Making processors Energy-Efficient

- Three techniques
  - Parallel execution
  - Dynamic power management (DPM)
  - Dynamic voltage and frequency scaling (DVFS)

Low voltage, parallel operation more efficient than high voltage, sequential

Basic equations

Power:	$P \sim V_{DD}^2$,
Maximum clock frequency:	$f \sim V_{DD}$,
Energy to run a program:	$E = P \times t$, with: t = runtime (fixed)
Time to run a program:	$t \sim 1/f$

Changes due to parallel processing, with β operations per clock:

Clock frequency reduced to:	$f' = f / \beta$,
Voltage can be reduced to:	$V_{DD}' = V_{DD} / \beta$,
Power for parallel processing:	$P° = P / \beta^2$ per operation,
Power for β operations per clock:	$P' = \beta \times P° = P / \beta$,
Time to run a program is still:	$t' = t$,
Energy required to run program:	$E' = P' \times t = E / \beta$

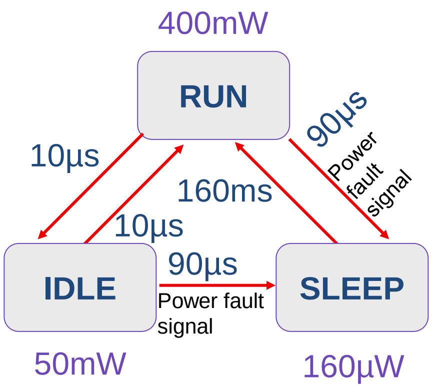
☞Argument in favour of voltage scaling, and parallel processing

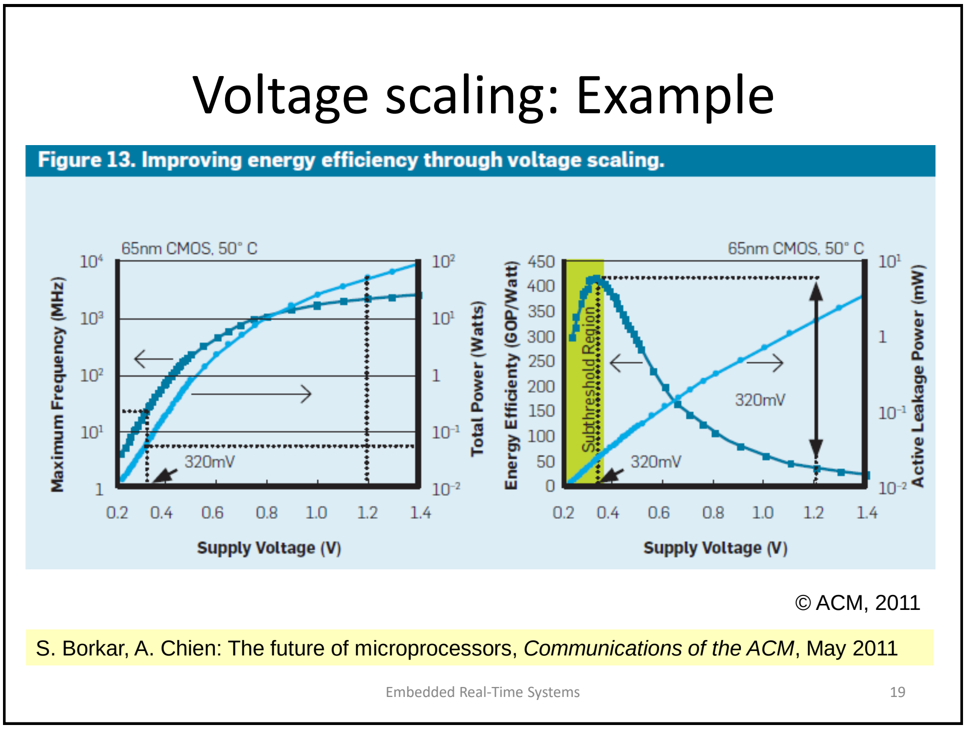
Rough approximations!

Dynamic Power Management (DPM)

- RUN: operational
- IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts
- SLEEP: Shutdown of on-chip activity

Example: STRONGARM SA1100



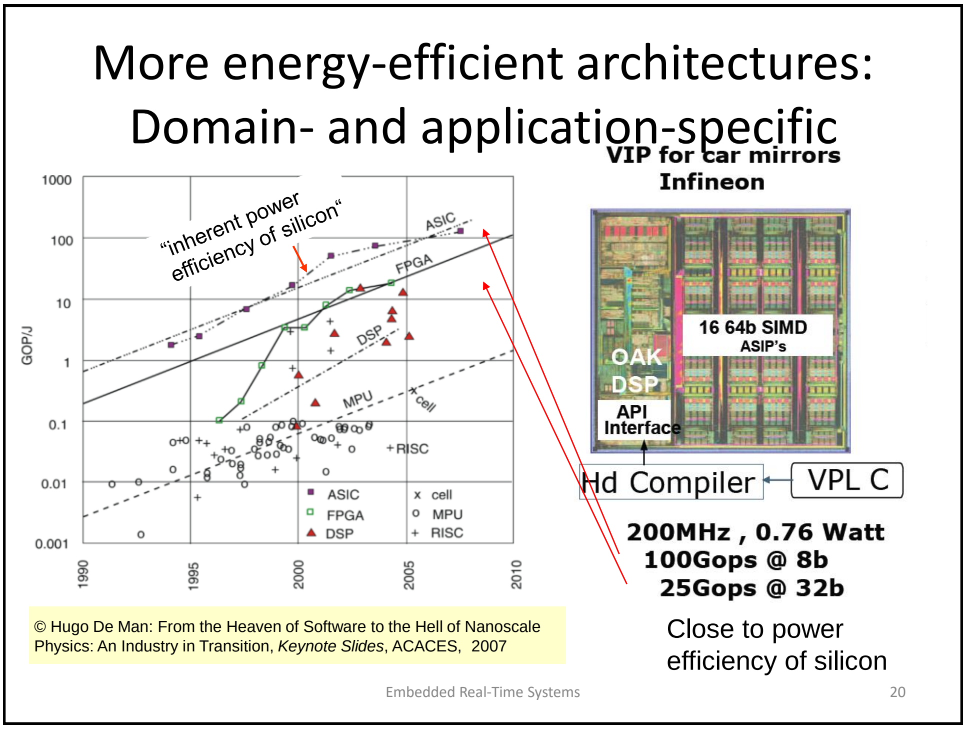


Voltage scaling: Example

Figure 13. Improving energy efficiency through voltage scaling.

© ACM, 2011

S. Borkar, A. Chien: The future of microprocessors, *Communications of the ACM*, May 2011



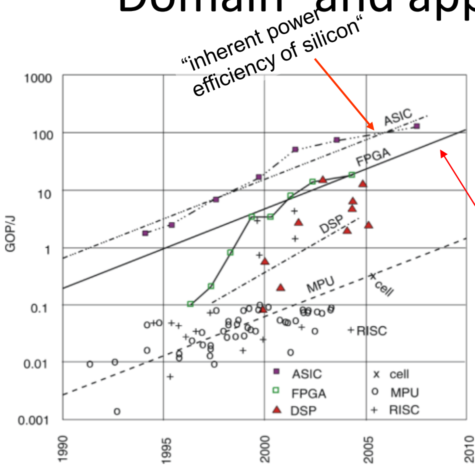
More energy-efficient architectures: Domain- and application-specific

200MHz , 0.76 Watt
100Gops @ 8b
25Gops @ 32b

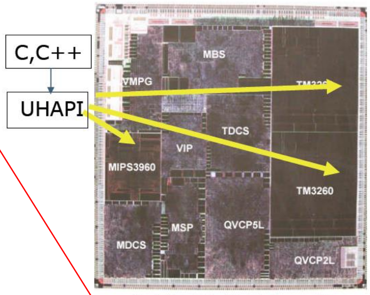
Close to power efficiency of silicon

© Hugo De Man: From the Heaven of Software to the Hell of Nanoscale Physics: An Industry in Transition, *Keynote Slides*, ACACES, 2007

Energy-efficient architectures: Domain- and application-specific



Nexperia Digital Video Platform NXP



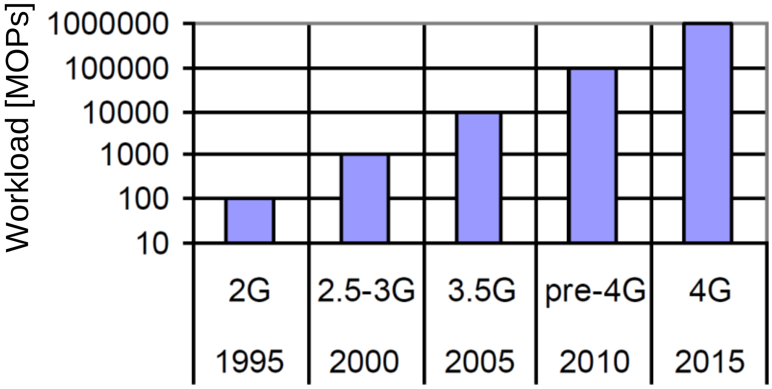
1 MIPS, 2 Trimedia
60 coproc, 250 RAM's
266MHz, 1.5 watt 100 Gops

Close to power efficiency of silicon

© Hugo De Man: From the Heaven of Software to the Hell of Nanoscale Physics: An Industry in Transition, *Keynote Slides*, ACACES, 2007

Mobile phones: Increasing performance requirements

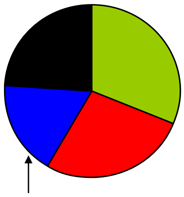
C.H. van Berkel: Multi-Core for Mobile Phones, DATE, 2009;



Many more instances of the power/energy problem

Mobile phones: Where does the power go?

- Mobile phone use, breakdown by type of computation



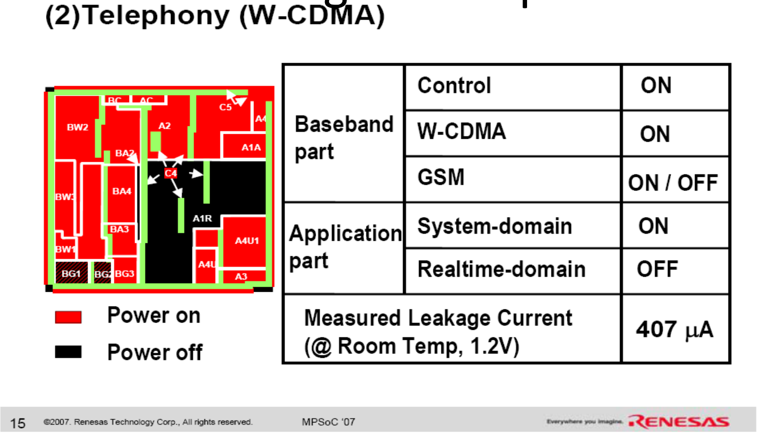
🟩 Graphics	(geometry processing, rasterization, pixel shading)
🟥 Media	(display & camera processing, video (de)coding)
🟦 Radio	(front-end, demodulation, decoding, protocol)
⬛ Application	(user interface, browsing, …)

With special purpose HW!

C.H. van Berkel: Multi-Core for Mobile Phones, DATE, 2009; (no explicit percentages in original paper)

☞ During use, all components & computations relevant

Energy-efficient architectures: Heterogeneous processors



(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

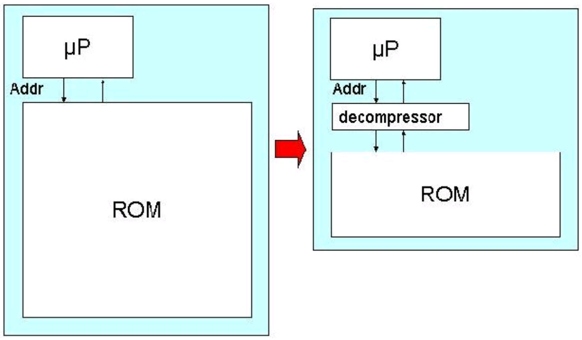
15 ©2007. Renesas Technology Corp., All rights reserved. MPSoC '07 Everywhere you imagine. RENESAS

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

☞ ***“Dark silicon”*** (not all silicon can be powered at the same time, due to current, power or temperature constraints)

Key requirement #2: Code-size efficiency

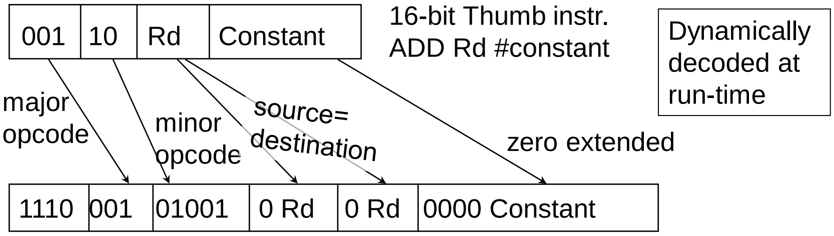
- CISC machines
- **Compression techniques:** key idea
  - Overview: http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html



Code-size efficiency

– Compression techniques (continued):

- 2nd instruction set, e.g. ARM Thumb instruction set:



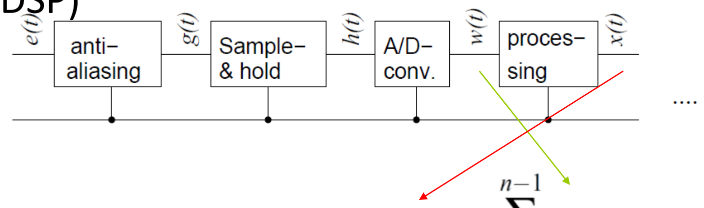
- Reduction to 65-70 % of original code size
- 130% of ARM performance with 8/16 bit memory
- 85% of ARM performance with 32-bit memory

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

[ARM, R. Gupta]

Key requirement #3: Run-time efficiency

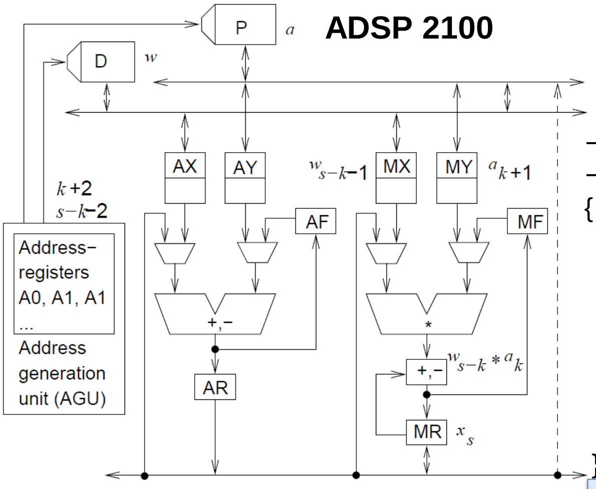
- Domain-oriented architectures
- Example: Filtering in Digital signal processing (DSP)



$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

Signal at t=t_s (sampling points)

Filtering in digital signal processing


ADSP 2100

$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

```
-- outer loop over
-- sampling times t_s
{ MR:=0; A1:=1; A2:=s-1;
  MX:=w[s]; MY:=a[0];
  for (k=0; k <= (n−1); k++)
   { MR:=MR + MX * MY;
     MX:=w[A2]; MY:=a[A1];
     A1++; A2−−;
   }
  x[s]:=MR;
}
```

Maps nicely

DSP-Processors

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

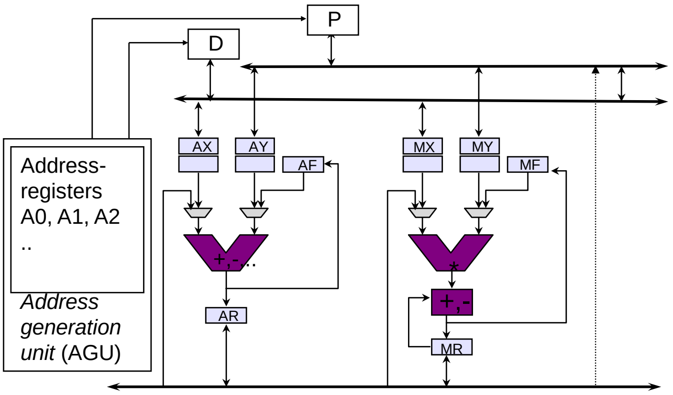
```
MR:=0; A1:=1; A2:=s-1; MX:=w[s]; MY:=a[0];
for ( k:=0 <= n-1)
  {MR:=MR+MX*MY; MY:=a[A1]; MX:=w[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) instruction

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

Heterogeneous registers

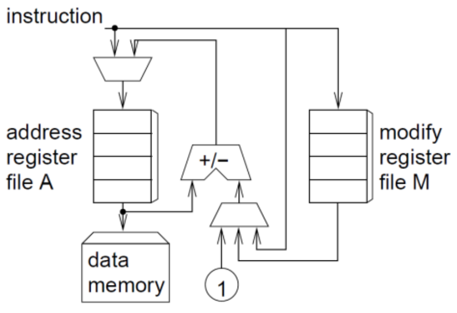
Example (ADSP 210x):


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

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

Separate address generation units (AGUs)

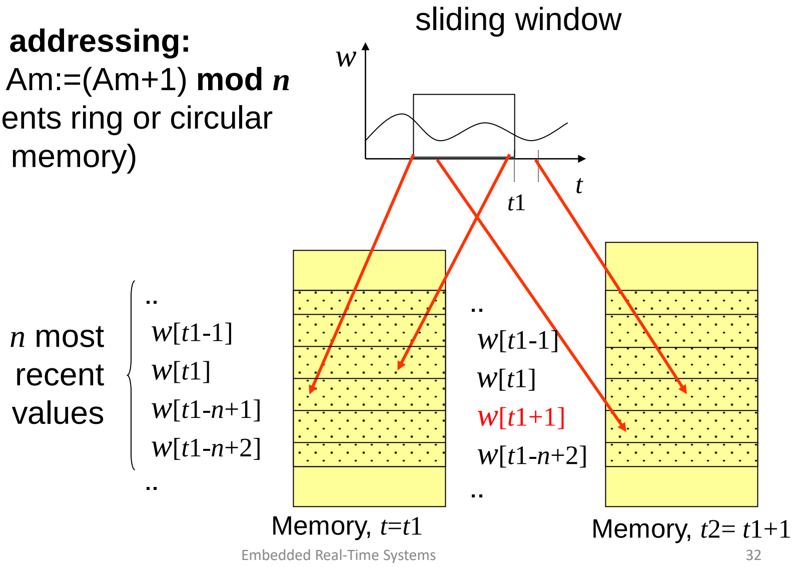
Example (ADSP 210x):

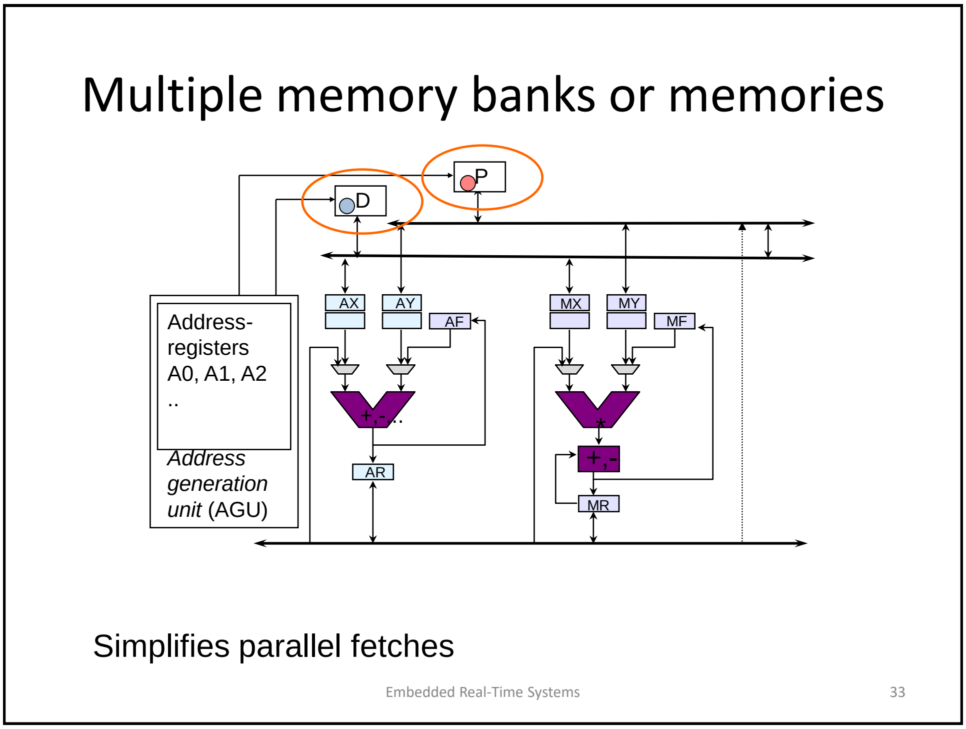


- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- A := A ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate in instruction> requires extra instruction
- ☞ Minimize load immediates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

Modulo addressing

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





Multiple memory banks or memories

D

P

Address-registers A0, A1, A2 ..

Address generation unit (AGU)

AX

AY

AF

+,-,..

AR

MX

MY

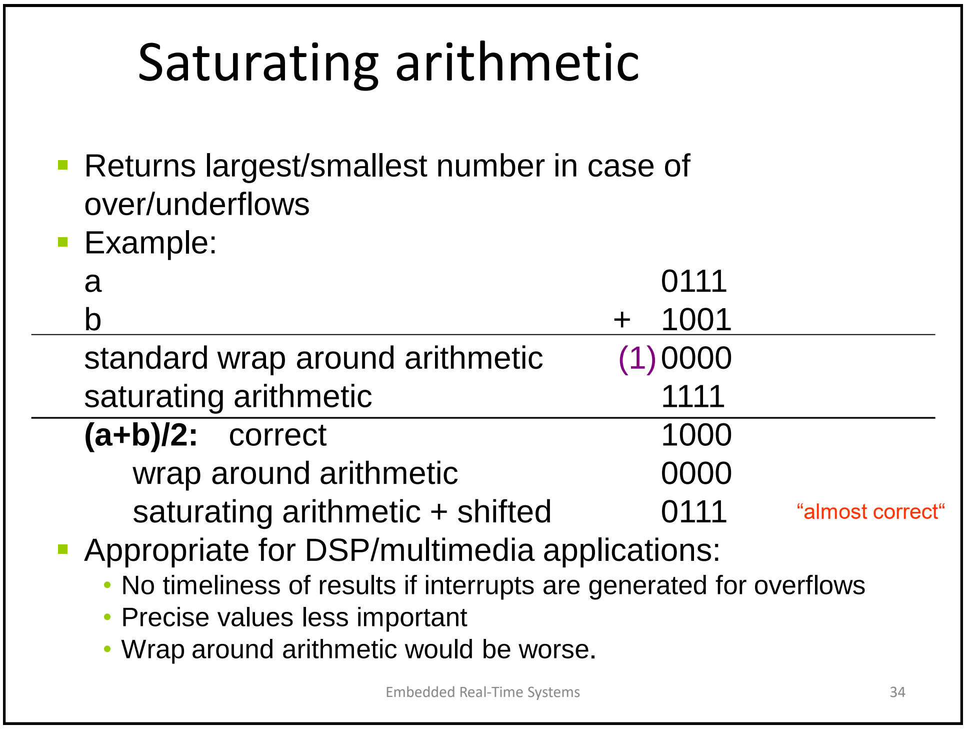
MF

*

+,-

MR

Simplifies parallel fetches



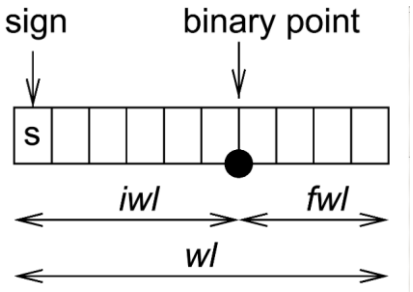
Saturating arithmetic

- 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



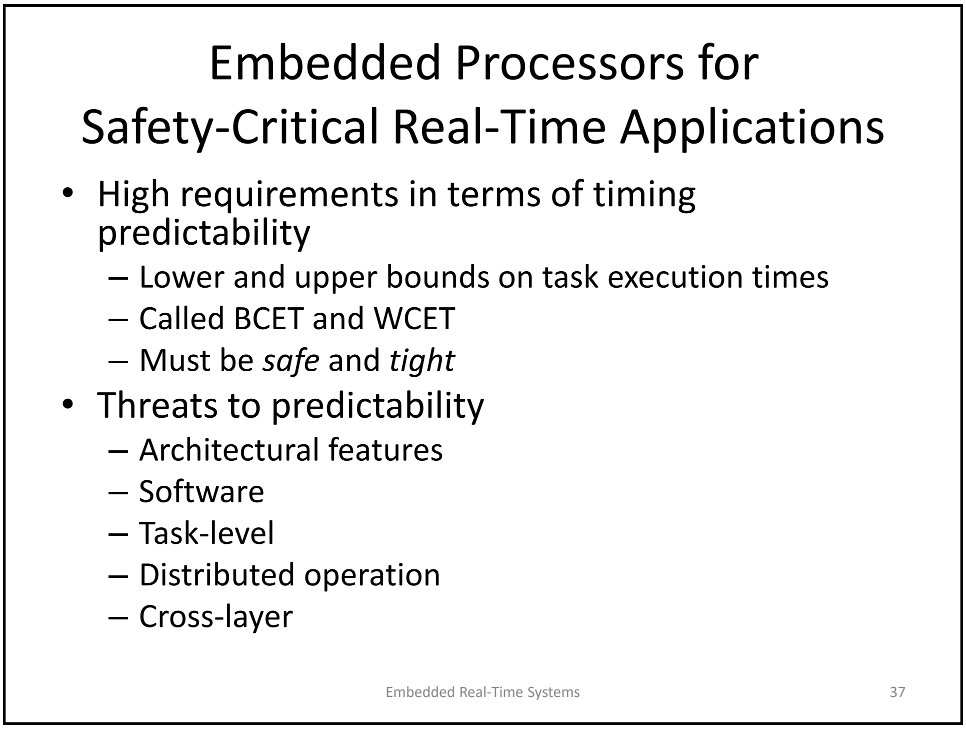
Shifting required after multiplications and divisions in order to maintain binary point.

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 between 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

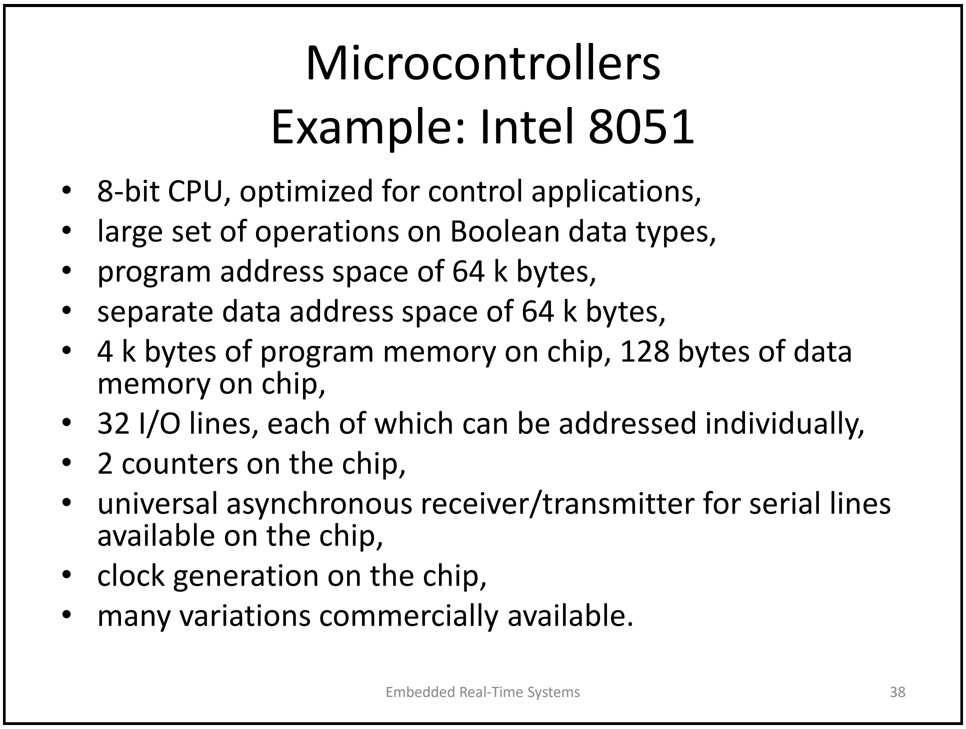
  ☞ Trying to avoid as many of these as possible.

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



Embedded Processors for Safety-Critical Real-Time Applications

- High requirements in terms of timing predictability
  - Lower and upper bounds on task execution times
  - Called BCET and WCET
  - Must be *safe* and *tight*
- Threats to predictability
  - Architectural features
  - Software
  - Task-level
  - Distributed operation
  - Cross-layer



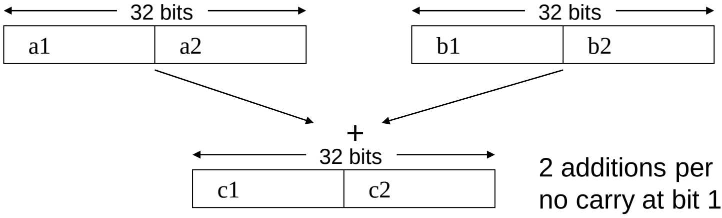
Microcontrollers Example: Intel 8051

- 8-bit CPU, optimized for control applications,
- large set of operations on Boolean data types,
- program address space of 64 k bytes,
- separate data address space of 64 k bytes,
- 4 k bytes of program memory on chip, 128 bytes of data memory on chip,
- 32 I/O lines, each of which can be addressed individually,
- 2 counters on the chip,
- universal asynchronous receiver/transmitter for serial lines available on the chip,
- clock generation on the chip,
- many variations commercially available.

Multimedia-Instructions, Short vector extensions, Streaming extensions, SIMD instructions

- Multimedia instructions exploit that many registers, adders etc. are quite wide (32/64 bit), whereas most multimedia data types are narrow

☞ 2-8 values can be stored per register and added. E.g.:


32 bits
a1
a2
32 bits
b1
b2
+
32 bits
c1
c2

2 additions per instruction;
no carry at bit 16

- Cheap way of using parallelism

☞ SSE instruction set extensions, SIMD instructions

Single ISA Heterogeneous Multi-core Processors ARM's big.LITTLE as an example

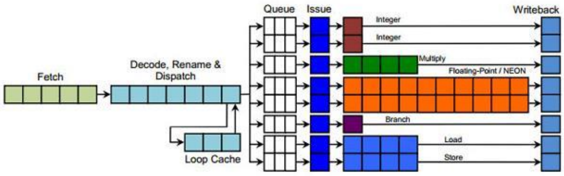


Figure 2 Cortex-A15 Pipeline

Used in Samsung S4

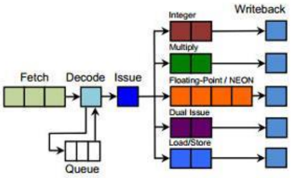
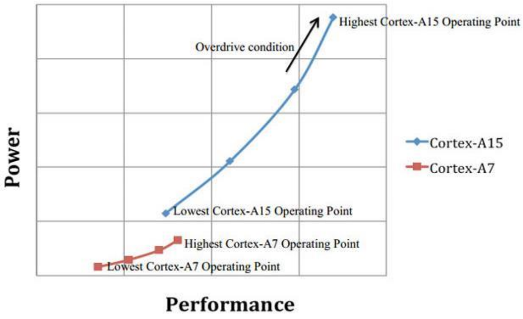


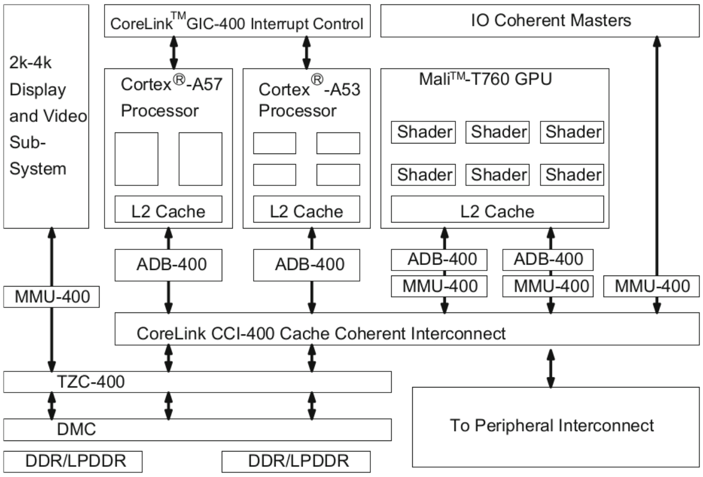
Figure 1 Cortex-A7 Pipeline



© ARM, 2013

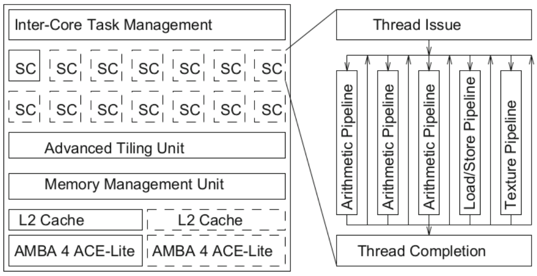
Multiprocessor Systems-on-a-Chip (MPSoCs)

ARM ® big.LITTLE System on Chip (SoC)



Graphics Processing Units (GPU)

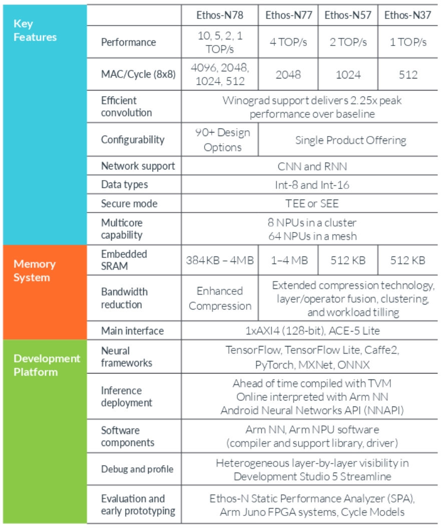
- Programmable GPUs
- Run many fine-grained threads at the same time
- Power efficiency important in embedded systems
- Interface to OpenGL, OpenCL, etc.



ARM ® Mali TM T880 GPU

ARM’s Neural Processing Units

- Object classification
- Object detection
- Face detection/identification
- Human pose detection/hand-gesture recognition
- Image segmentation
- Image beautification
- Super resolution
- Framerate adjustment (super slow-mo)
- Speech recognition
- Sound recognition
- Noise cancellation
- Speech synthesis
- Language translation



		Ethos-N78	Ethos-N77	Ethos-N57	Ethos-N37
Key Features	Performance	10, 5, 2, 1 TOP/s	4 TOP/s	2 TOP/s	1 TOP/s
	MAC/Cycle (8x8)	4096, 2048, 1024, 512	2048	1024	512
	Efficient convolution	Winograd support delivers 2.25x peak performance over baseline			
	Configurability	90+ Design Options	Single Product Offering		
	Network support	CNN and RNN			
	Data types	Int-8 and Int-16			
	Secure mode	TEE or SEE			
	Multicore capability	8 NPUs in a cluster 64 NPUs in a mesh			
Memory System	Embedded SRAM	384KB – 4MB	1–4 MB	512 KB	512 KB
	Bandwidth reduction	Enhanced Compression	Extended compression technology, layer/operator fusion, clustering, and workload tilling		
	Main interface	1xAXI4 (128-bit), ACE-5 Lite			
Development Platform	Neural frameworks	TensorFlow, TensorFlow Lite, Caffe2, PyTorch, MXNet, ONNX			
	Inference deployment	Ahead of time compiled with TVM Online interpreted with Arm NN Android Neural Networks API (NNAPI)			
	Software components	Arm NN, Arm NPU software (compiler and support library, driver)			
	Debug and profile	Heterogeneous layer-by-layer visibility in Development Studio 5 Streamline			
	Evaluation and early prototyping	Ethos-N Static Performance Analyzer (SPA), Arm Juno FPGA systems, Cycle Models			

Arm’s ML processor architecture key features

Efficient convolutions

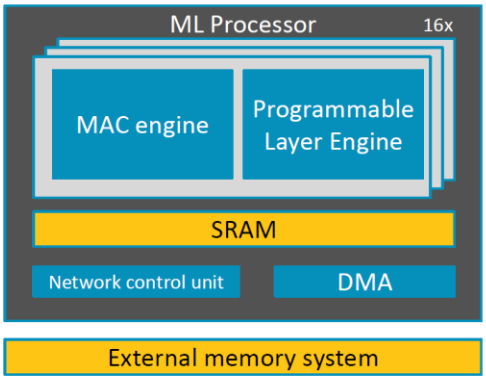
- Convolutions represent the bulk of computation
- We provide dedicated 8-bit hardware for convolutions

Efficient data movement

- More energy is spent moving data than computing
- We amortize activation accesses and compress weights

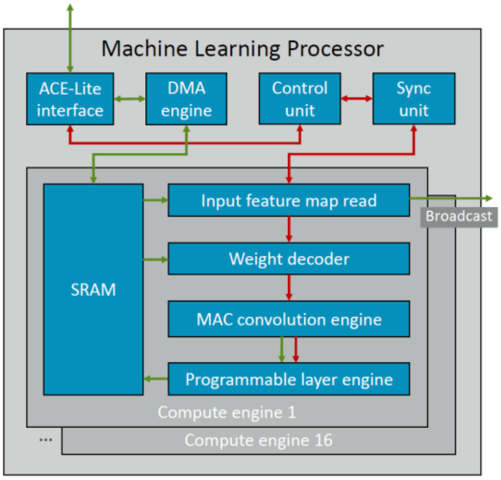
Sufficient programmability

- New operators are invented and topology is changed frequently
- We provide programmability to future-proofed as new network architectures appear



Arm’s ML processor

- A microcontroller and DMA engine manage overall network scheduling
- The compute engine processes major sections of the neural network
  - Stores weights
  - Stores and manipulates activation data
  - Handles convolution in 128-wide MAC units
  - Handles other layer operators via PLE
  - Pipelines data to and from SRAM
- Internal broadcast network manages SRAM population and synchronization

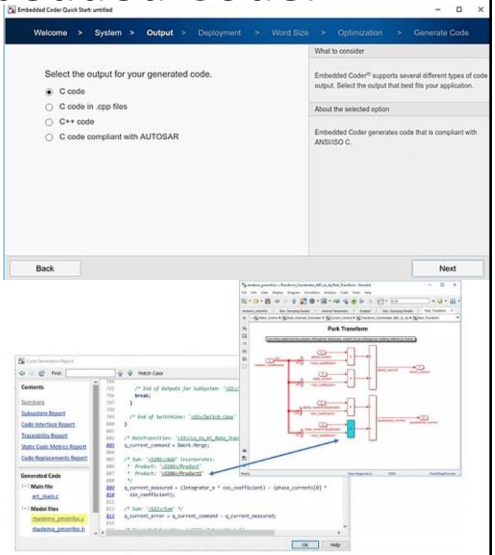


Reconfigurable Logic

- Fast prototyping
- Low-volume applications
- Real-time systems
- High level of parallel processing
- Field programmable gate arrays (FPGAs) are the most common (Xilinx, Intel, Lattice, etc.)
  - Configurable logic
  - Memory
  - IO
  - Hard cores

MATLAB Embedded Coder

- Generates readable, compact, portable, and fast C and C++ code for embedded processors
- Optimizations improve code efficiency and facilitate integration with legacy code, data types, and calibration parameters.
- Supports software-in-the-loop (SIL) and processor-in-the-loop (PIL) testing.



Select the output for your generated code.

● C code
○ C code in .cpp files
○ C++ code
○ C code compliant with AUTOSAR

Embedded Coder® supports several different types of code output. Select the output that best fits your application.

Embedded Coder generates code that is compliant with ANSI/ISO C.

```
      /* End of Outputs for SubSystem: '<S3>/
      break;
    }

  /* End of SwitchCase: '<S3>/Switch Case' */
}

/* RateTransition: '<S3>/Iq_to_el_Rate_Tran
q_current_command = DWork.Merge;

/* Sum: '<S186>/Add' incorporates:
 *  Product: '<S186>/Product'
 *  Product: '<S186>/Product1'
 */
q_current_measured = (Integrator_o * cos_coefficient) - (phase_currents[0] *
  sin_coefficient);

/* Sum: '<S12>/Sum' */
q_current_error = q_current_command - q_current_measured;
```