Parallel digital signal processing in a mW power envelope: how and why?

Luca Benini
DEI-UNIBO & IIS-ETHZ
IOT or Data Deluge?

Highly parallel workloads!

CV is the energy bottleneck

In-situ stream processing & fusion + visual intelligence is a must! - In a few mW power envelope!!
How efficient?

10^{12}\text{ops/J}

\downarrow

1\text{pJ/op}

\downarrow

1\text{GOPS/mW}
The challenge of Energy Proportionality

From KOPS ($10^3$) to EOPS ($10^{18}$)!

0.003 GOPS/mW – 30 KW

0.04 GOPS/mW
A Short Review on CMOS power
and power minimization
Where is Power Dissipated in CMOS?

- **Active (Dynamic) power**
  - (Dis)charging capacitors
  - Short-circuit power
    - Both pull-up and pull-down on during transition
- **Static (leakage) power**
  - Transistors are imperfect switches
- **Static currents**
  - Biasing currents
Active (or Dynamic) Power

Key property of active power:

\[ P_{\text{dyn}} \propto f \]

with \( f \) the switching frequency

Sources:
- Charging and discharging capacitors
- Temporary glitches (dynamic hazards)
- Short-circuit currents
Charging Capacitors

Applying a voltage step

\[ E_{0\rightarrow1} = CV^2 \]

\[ E_R = \frac{1}{2} CV^2 \]

\[ E_C = \frac{1}{2} CV^2 \]

\[ E_{0\rightarrow1} = \int_0^\infty V C \frac{dV_C}{dt} \, dt = CV \int_0^V dV_C = CV^2 \]

Value of R does not impact energy!
One half of the power from the supply is consumed in the pull-up network and one half is stored on $C_L$

- Charge from $C_L$ is dumped during the $1 \rightarrow 0$ transition
- Independent of resistance of charging/discharging network
Circuits with Reduced Swing

Energy consumed is proportional to output swing
Charging Capacitors - Revisited

Driving from a constant current source

\[ E_{0\rightarrow1} = E_C + E_R \]

\[ E_R = \left(\frac{RC}{T}\right)CV^2 \]

\[ E_C = \frac{1}{2}CV^2 \]

\[ T = \frac{CV}{I} \]

\[ E_R = \int_{0}^{\infty} I(RI)dt = RI^2T = \left(\frac{RC}{T}\right)CV^2 \]

Energy dissipated in resistor can be reduced by increasing charging time \( T \) (that is, decreasing \( I \))
Charging Capacitors

Using constant voltage or current driver?

\[
E_{\text{constant\_current}} < E_{\text{constant\_voltage}}
\]

if

\[
T > 2RC
\]

Energy dissipated using constant current charging can be made arbitrarily small at the expense of delay:

**Adiabatic charging**

Note: \( t_p(\text{RC}) = 0.69 \text{ RC} \)

\( t_{0\rightarrow90\%}(\text{RC}) = 2.3 \text{ RC} \)
Dynamic Power Consumption

Power = Energy/transition • Transition rate

\[ = C_L V_{DD}^2 \cdot f_{0\rightarrow1} \]

\[ = C_L V_{DD}^2 \cdot f \cdot P_{0\rightarrow1} \]

\[ = C_{switched} V_{DD}^2 \cdot f \]

- Power dissipation is data dependent – depends on the switching probability
- Switched capacitance \( C_{switched} = P_{0\rightarrow1} C_L = \alpha C_L \) (\( \alpha \) is called the switching activity)
Short-Circuit Currents

(also called crowbar currents)

PMOS and NMOS simultaneously on during transition

\[ P_{sc} \sim f \]
Short-Circuit Currents

Equalizing rise/fall times of input and output signals limits $P_{sc}$ to 10-15% of the dynamic dissipation

[Ref: H. Veendrick, JSSC'84]
Modeling Short-Circuit Power

- Can be modeled as capacitor

\[ C_{SC} = k \left( a \frac{\tau_{in}}{\tau_{out}} + b \right) \]

a, b: technology parameters
k: function of supply and threshold voltages, and transistor sizes

\[ E_{SC} = C_{SC} V_{DD}^2 \]

Easily included in timing and power models
Transistors Leak

- **Drain leakage**
  - Diffusion currents
  - Drain-induced barrier lowering (DIBL)
- **Junction leakages**
  - Gate-induced drain leakage (GIDL)
- **Gate leakage**
  - Tunneling currents through thin oxide
Sub-threshold Leakage

Off-current increases exponentially when reducing $V_{TH}$

$$I_{\text{leak}} = I_0 \frac{W}{W_0} 10^{\frac{-V_{TH}}{S}}$$

$$P_{\text{leak}} = V_{DD} \cdot I_{\text{leak}}$$
Sub-Threshold Leakage

Leakage current increases with drain voltage (mostly due to DIBL)

\[ I_{\text{leak}} = I_0 \frac{W}{W_0} 10 \left( \frac{-V_{TH} + \lambda_d V_{DS}}{S} \right) \quad \text{(for } V_{DS} > 3 \frac{kT}{q}) \]

Hence

\[ P_{\text{leak}} = (I_0 \frac{W}{W_0} 10 \frac{-V_{TH}}{S}) (V_{DD} 10 \frac{\lambda_d V_{DD}}{S}) \]

Leakage Power strong function of supply voltage
Stack Effect

NAND gate:

Assume that body effect in short channel transistor is small

\[ I_{\text{leak}, M_1} = I_0 10 \left( 1 + \frac{-V_M - V_{TH} + \lambda_d (V_{DD} - V_M)}{s} \right) \]

\[ I_{\text{leak}, M_2} = I_0 10 \left( 1 + \frac{-V_{TH} + \lambda_d V_M}{s} \right) \]

\[ V_M \approx \frac{\lambda_d}{1 + 2\lambda_d} V_{DD} \]

\[ \frac{I_{\text{stack}}}{I_{\text{inv}}} \approx 10 \left( \frac{\lambda_d V_{DD}}{s} \frac{1 + \lambda_d}{1 + 2\lambda_d} \right) \] (instead of the expected factor of 2)
Stack Effect

90 nm NMOS

Leakage Reduction

<table>
<thead>
<tr>
<th>NMOS Count</th>
<th>Reduction Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>2 PMOS</td>
<td>8</td>
</tr>
<tr>
<td>3 PMOS</td>
<td>12</td>
</tr>
<tr>
<td>4 PMOS</td>
<td>16</td>
</tr>
</tbody>
</table>
Gate Tunneling

Exponential function of supply voltage

- \( I_{GD} \sim e^{-Tox} e^{VGD} \), \( I_{GS} \sim e^{-Tox} e^{VGS} \)
- Independent of the sub-threshold leakage

Modeled in BSIM4
Also in BSIM3v3 (but not always included in foundry models)
NMOS gate leakage usually worse than PMOS
Other sources of static power dissipation

- Diode (drain-substrate) reverse bias currents

- Electron-hole pair generation in depletion region of reverse-biased diodes
- Diffusion of minority carriers through junction
- For sub-50nm technologies with highly-doped pn junctions, tunneling through narrow depletion region becomes an issue

Strong function of temperature

Much smaller than other leakage components in general
Other sources of static power dissipation

- Circuit with dc bias currents:
  
sense amplifiers, voltage converters and regulators, sensors, mixed-signal components, etc

Should be turned off if not used, or standby current should be minimized
Summary of Power Dissipation Sources

\[
P \sim \alpha \cdot (C_L + C_{CS}) \cdot V_{\text{swing}} \cdot V_{DD} \cdot f + (I_{DC} + I_{\text{Leak}}) \cdot V_{DD}
\]

- \(\alpha\) – switching activity
- \(C_L\) – load capacitance
- \(C_{CS}\) – short-circuit capacitance
- \(V_{\text{swing}}\) – voltage swing
- \(f\) – frequency
- \(I_{DC}\) – static current
- \(I_{\text{Leak}}\) – leakage current

\[
P = \frac{\text{energy}}{\text{operation}} \times \text{rate} + \text{static power}
\]
The Traditional Design Philosophy

- Maximum performance is primary goal
  - Minimum delay at circuit level
- Architecture implements the required function with target throughput, latency
- Performance achieved through optimum sizing, logic mapping, architectural transformations.
- Supplies, thresholds set to achieve maximum performance, subject to reliability constraints
The New Design Philosophy

- Maximum performance (in terms of propagation delay) is too power-hungry, and/or not even practically achievable
- Many (if not most) applications either can tolerate larger latency, or can live with lower than maximum clock-speeds
- Excess performance (as offered by technology) to be used for energy/power reduction

Trading off speed for power
In energy-constrained world, design is trade-off process

- Minimize energy for a given performance requirement
- Maximize performance for given energy budget
Summary

- Power and energy are now primary design constraints
- Active power still dominating for most applications
  - Supply voltage, activity and capacitance the key parameters
- Leakage becomes major factor in sub-100nm technology nodes
  - Mostly impacted by supply and threshold voltages
- Design has become energy-delay trade-off exercise!
Reducing power @ all design levels

- Algoritmic level
- Compiler level
- Architecture level
- Organization level
- Circuit level
- Silicon level

Important concepts:
- Lower Vdd and freq. (even if errors occur) / dynamically adapt Vdd and freq.
- Reduce circuit
- Exploit locality
- Reduce switching activity, glitches, etc.
Algoritmic level

◆ The best indicator for energy is ..... 
    .... the number of cycles

◆ Try alternative algorithms with lower complexity
  ■ E.g. quick-sort, \(O(n \log n) \leftrightarrow \text{bubble-sort, } O(n^2)\)
  ■ ... but be aware of the 'constant' : \(O(n \log n) \Rightarrow c^*(n \log n)\)

◆ Heuristic approach
  ■ Go for a good solution, not the best !!

Biggest gains at this level !!
Compiler level

- Source-to-Source transformations
  - Loop trafo's to improve locality
- Strength reduction
  - E.g. replace Const * A with Add's and Shift's
  - Replace Floating point with Fixed point
- Reduce register pressure / number of accesses to register file
  - Use software bypassing
- Scenarios: current workloads are highly dynamic
  - Determine and predict execution modes
  - Group execution modes into scenarios
  - Perform special optimizations per scenario
    - DFVS: Dynamic Voltage and Frequency Scaling
    - More advanced loop optimizations
- Reorder instructions to reduce bit-transistions
Architecture level

◆ Going parallel

◆ Going heterogeneous
  ■ tune your architecture, exploit SFUs (special function units)
  ■ trade-off between flexibility / programmability / genericity and efficiency

◆ Add local memories
  ■ prefer scratchpad i.s.o. cache

◆ Cluster FUs and register files (see next slide)

◆ Reduce bit-width
  ■ sub-word parallelism (SIMD)
Organization (micro-arch.) level

◆ Enabling Vdd reduction
  ■ Pipelining
    ● cheap way of parallelism
  ■ Enabling lower freq. \( \Rightarrow \) lower \( V_{dd} \)
  
  ■ Note 1: don't pipeline if you don't need the performance
  ■ Note 2: don't exaggerate (like the 31-stage Pentium 4)

◆ Reduce register traffic
  ■ avoid unnecessary reads and write
  ■ make bypass registers visible
Circuit level

- Clock gating
- Power gating
- Multiple Vdd modes
- Reduce glitches: balancing digital path's
- Exploit Zeros
- Special SRAM cells
  - normal SRAM can not scale below Vdd = 0.7 - 0.8 Volt
- Razor method; replay
- Allow errors and add redundancy to architectural invisible structures
  - branch predictor
  - caches
- .. and many more ..


**Silicon level**

- Higher $V_t$ ($V_{\text{threshold}}$)
- Back Biasing control
  - see thesis Maurice Meijer (2011)
- SOI (Silicon on Insulator)
  - silicon junction is above an electr. insulator (silicon dioxide)
  - lowers parasitic device capacitance

- Better transistors: Finfet
  - multi-gate
  - reduce leakage (off-state curent)

- .. and many more
Two Ideas to Remember

...with their caveats
Go Simple+Parallel

13mm, **100W**, 48MB Cache, 4B Transistors, in 22nm

12 Cores

48 Cores

144 Cores

[S. Borkar, Intel, 2006]
Lower Voltage Supply

...but be careful with Leakage and its variability!

[V. De, Intel, 2013]
Introducing PULP
NTC Multicore?

2010 - With EPFL (Atienza, Burg)

90nm LP-CMOS

Leakage!

High Vdd

1GOPS/mW

10x @10MOPS

1.7 MOps/s
0.03 mW

167 MOps/s
15.8 mW

65.6 mW

6.0 mW

750 MOps/s

Power, mW

Workload, MOps/s
A pJ/OP Parallel ULP Computing Platform

- **pJ/OP** is traditionally* the target of ASIC + uCntr
  - Scalable: [KOPS,TOPS], 32bit architecture
  - Flexible: OpenMP, OpenVX
  - Open: Software & HW

*1.57TOPS/W: Kim et al., “A 1.22TOPS and 1.52mW/MHz Augmented Reality Multi-core Processor with Neural Network NoC for HMD Applications”, ISSCC 2014
Start with an simple+efficient processor (~1IPC)
Making PULP

Parallel processors for performance @ NT
Making PULP

Parallel access to shared memory → Flexibility
Making PULP

EXU Logarithmic Interconnect

TCDM Logarithmic Interconnect

OpenRISC Core #1

OpenRISC Core #2

OpenRISC Core #3

OpenRISC Core #N

Instruction Memory

Instruction Memory

Instruction Memory

Instruction Memory

Optional Instruction Extension → Acceleration
Making PULP

Add infrastructure to access off-cluster memory
Multiple clusters (f, Vdd, Vbb) form a PULP system
**Design choices**

- **I$**: high code locality & simple architecture
- **No D$**: low locality & high complexity: $B_{pmm^2_{DS}}/B_{pmm^2_{DTCDM}}<0.4$
- **Sharing L1**: less copies, easy work-balancing, low $T_{clk}$ overhead in NT
- **Multibank**: smaller energy per access, “almost” multiported
OpenRisc Optimization

- Superpipelining harmful for energy efficiency
  - Focused speed optimization on the critical path dominated by MEM
- Low pipeline depth $\rightarrow$ high IPC with simple microarchitecture

50% less energy per operation on average, 5% more area
Logarithmic Interconnect

Ultra-low latency $\rightarrow$ short wires + 1 clock cycle latency

World-level bank interleaving «emulates» multiported mem
Low latency programming Interface

- Each command queue is dedicated to a core of the cluster: arbitration is made in hardware
  → No need to reserve (lock/unlock) the programming channel

- COREs program DMA through register mapped on the DEMUX
  → The registers belongs to aliases, no need for the processors to calculate (per-core) offsets

- A programming sequence requires
  1. check a free command queue
  2. write address of buffer in TCDM
  3. write address of buffer in L2 memory
  4. Trigger data transfer
  5. Synchronization

Programming Latency: ~10 CLOCK CYCLES!!!
DMA Architecture Overview

- **CTRL UNIT:**
  - Arbitration – forwarding and synchronization of incoming requests

- **TRANSFER UNIT:**
  - FIFO Buffers for TX and RX channels

- **TCDM UNIT:**
  - Bridge to TCDM protocol – 4 channels (2 RD and 2 WR) 32 bit each

- **EXT UNIT:**
  - Bridge to AXI, 64 bit

Key idea: only channel packets buffered internally – no DMA transfers!
Cluster DMA

BF = 2
Typical config

Maximum BW for realistic applications

Maximum BW for realistic applications

Cluster DMA

BF = 2
Typical config

Maximum BW for realistic applications

Cluster DMA

BF = 2
Typical config

Maximum BW for realistic applications
Sharing functional units
(instruction set extensions)

- Floating point units are area expensive, leakage hungry components.
- Typical FPU density in application is no more than 20%.
- FPUs typically feature latencies of several clock cycles.
- 32 bit architecture
- ⇒ 4 GB of memory
- Clusters in Vdd, CLK domains
- L2 (2D & 3D) ready
- Host VM IF for Heterogeneous Computing
Programmability: OpenMP

while(1) {
    #pragma omp parallel num_threads(4)
    {
        #pragma sections
        {
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                ColorScaleConv();
            }
        }
    }
}

A. void ColorScaleConv()
    {
        #pragma omp for
        for(i = 0; i < FRAME_SIZE; i++)
        {
            [ALGORITHM]
        }
    }

A powerful abstraction for specifying structured parallelism

And very suitable for NUMA (cluster-based) systems

B. void cvThreshold()
    {
        #pragma omp parallel num_threads(16)
        cvMoments();
    }

C. void cvMoments()
    {
        #pragma omp parallel num_threads(16)
        cvAdd();
    }

D. void cvAdd()
    {
        #pragma omp parallel num_threads(16)
        cvAdd();
    }

+OpenVX → domain specific language

LLVM + OpenCL
Programming: OpenVX

- C-based standard API for vision kernel
  - Defines a set of **standard kernels**
  - Enables hardware vendors to **implement accelerated imaging and vision algorithms**
- Focus on enabling real-time vision
  - On mobile and embedded systems
- **Graph execution model**
  - Each node can be implemented in software or accelerated hardware
  - Data transfer between nodes may be optimized
Vanilla OVX Runtime

Leverages OpenCL runtime with OVX nodes treated as OCL kernels

HOST

DEVICE

L3 MEM

Kernel 1

Kernel 2

Intermediate results stored in L3 mem → each kernel boundary implies two accesses to all image data (write + read)

Host synchronization between successive kernels is imposed
Localized Execution

- Smaller image sub-regions (tiles) totally reside in TCDM
- All kernels are computed at tile level, no more at image level → intermediate results are also allocated in TCDM

- TCDM is partitioned into 3 buffers (B0, B1, B2)
- Output tile size is 160x120
- Sobel3x3 requires image overlapping (1 pixel for each direction), and so the tile size is 162x122
- ColorConvert does not require tile overlapping, but must provide a 162x122 result tile for the next stage → tile size propagation
Localized Execution

- Smaller image sub-regions (tiles) totally reside in TCDM
- All kernels are computed at tile level, no more at image level → *intermediate results are also allocated in TCDM*

- TCDM is partitioned into 3 buffers (B0, B1, B2)
- Output tile size is 160x120
- Sobel3x3 requires image overlapping (1 pixel for each direction), and so the tile size is 162x122
- ColorConvert does not require tile overlapping, but must provide a 162x122 result tile for the next stage → *tile size propagation*
Localized Execution Results

- Framework prototype
  - OpenVX support
  - A limited subset of kernel has been implemented
  - Polynomial time (i.e. fast) heuristics suitable for *just-in-time execution*

!![Cluster PE Efficiency](chart.png)
And what about Technology?
Near threshold FDSOI technology

Body bias: Highly effective knob for leakage control!
Near threshold FDSOI technology
PULP V1: Doing nothing well (with RBB)

Area Breakdown:
- 38% Cores
- 29% TCDM
- 23% Instruction Caches
- 10% Interconnect

Power Breakdown:
- 39% Cores
- 32% TCDM
- 25% Instruction Caches
- 3% Interconnect

More than 50% of power into memories… this is the next focus area!
Introducing PULP V2
Board/Application-ready chip

- Implementation of a master and a slave peripheral
  - Standard peripheral (e.g. SPI)
  - Integration with FPGAs or standard low power external memories

- PULP as multi-core accelerator for micro-controller host
  - STM32 host core
  - PULP multi-core accelerator

- Daisy chain of PULP chips
  - Pipeline of parallel processing units
  - Each core performs a stage of computation and forward temporary data to another stage

Standalone mode is also supported!
All-digital clock generation based on a Frequency Lock Loop (FLL)
- From 2GHz down to 15KHz (through clock division)
- Frequency step 10MHz (at lowest division factor)
- Small area 3300µm² (50 times smaller than classic PLL)
- Suited to fine-grain GALS architectures
- Frequency reprogramming in less than 180ns
- No frequency overshoot
- 15ps jitter

Near threshold FDSOI technology

Low-leakage vs. Low voltage (0.3V) \(\rightarrow\) reactive or proactive?
ULP Latch-based SCM

- Based exclusively on standard cells
- Voltage range identical to the core
  - Only static logic
- Layout based on guided P&R
  - Close to 100% density

Macro Size:
128 x 32 (4 kb)
86µm x 160µm

Input/Output Delay:
0.3ns/0.7ns @ss0.9V125°
2ns/3.3ns @tt0.3V25° (FBB)
Comparison to 64x64 SRAM

**Comparison to 64x64 SRAM**

**AREA**
- SCM: 14000
- SPL1CACHE: 7200
- SPHD: 6000
- SPREG: 4800

**WRITE ENERGY**
- SCM
- SPL1CACHE
- SPHD
- SPREG

**READ ENERGY**
- SCM
- SPL1CACHE
- SPHD
- SPREG

**LEAKAGE POWER @125°C**
- SCM
- SPL1CACHE
- SPHD
- SPREG
SCM Integration into PULP2

- 16 banks of 128x32 bit SCM (8 kbyte)
- 8 banks of 512x32 bit SRAM (16 kbyte)
- 2 banks of 128x32 bit SCM (2 kbyte) Instruction cache per core

PULP V2 is taping out this Week, PULP V3 is on the drawing board...
Reconfigurable address mapping

- Support for different address mappings:
  - Interleaved: horizontal shutdown and reduces conflicts in shared segments
  - Non-interleaved mapping: private memory avoids conflicts

- Basic MMU
  - Coexistence of both shared and private memory segments with different address mappings
  - Adapt address mapping is adapted to accommodate partial memory shut-down
An ULP Computing Ecosystem

**HARDWARE IPs**
- PROCESSOR
- INTERCONNECT (LOCAL, GLOBAL)
- MEMORY HIERARCHY (CACHES, MEMORY CONTROLLERS)
- HARDWARE ACCELERATORS
- ...

**SOFTWARE**
- COMPILER/TOOLCHAIN
- PROGRAMMING MODELS
- RUNTIME
- ...

**VALIDATION**
- VIRTUAL PLATFORM
- EMULATION PLATFORM (FPGA)
- BENCHMARKS
- REGRESSION TESTS
- ...

**SILICON**
- OPTIMIZATION FLOW
- IMPLEMENTATION FLOW
- VERIFICATION FLOW
- FULL CUSTOM IPS
- ...

Building an open-source ecosystem for exploring *(with silicon!)* next-generation parallel computing platforms

...
To do what?

@60fps 0.76MPx/s → with 1KOPS/pixel we need 0.75Gops!

[Wood13]
The Grand Challenge: Energy proportionality

**Goal:** reduce «bending up» of energy curve at low & high perf!

+ Liquid cooling
+ Managing extreme variability

PULP1-2 32b, Kops-Tops @ pJ/W
Thank you!

Multithermand AdG
Multiscale Thermal Management of Computing Systems