

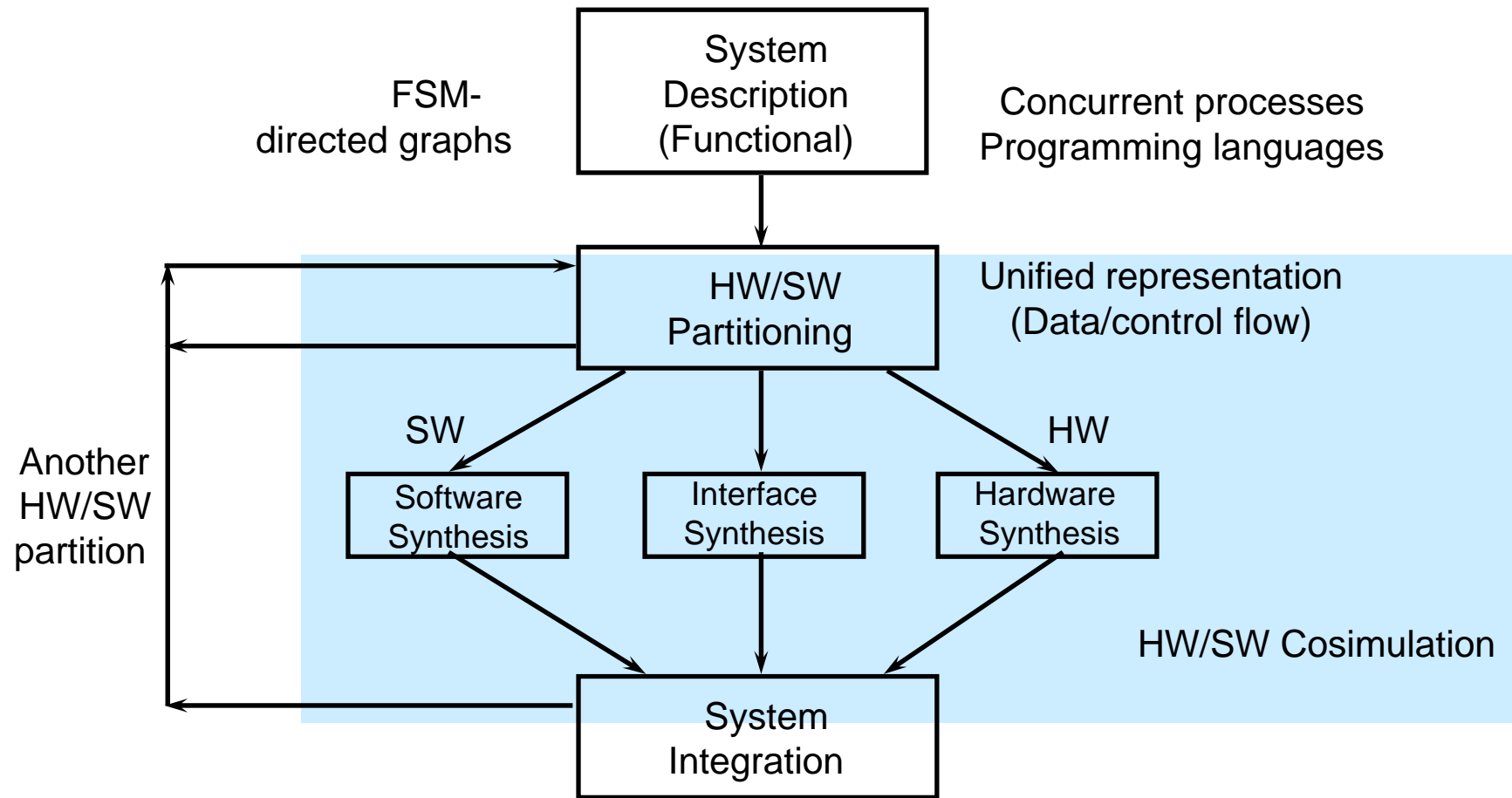
Embedded Systems



Codesign Definition and Key Concepts

- Codesign
 - The meeting of system-level objectives by exploiting the **trade-offs between hardware and software** in a system through their concurrent design
- Key concepts
 - **Concurrent:** hardware and software developed at the same time on parallel paths
 - **Integrated:** interaction between hardware and software development to produce design **meeting performance criteria and functional specs**

Typical Codesign Process



Main Tasks of the Codesign Problem

- Specification of the system
- Hardware/Software Partitioning
 - **Architectural assumptions** - type of processor, interface style between hardware and software, etc.
 - **Partitioning objectives** - maximize speedup, latency requirements, minimize size, cost, etc.
 - **Partitioning strategies** - high level partitioning by hand, automated partitioning using various techniques, etc.
- Scheduling
 - Operation scheduling in hardware
 - Instruction scheduling in compilers
 - Process scheduling in operating systems
- Modeling/Simulation of the hardware/software system during the design process

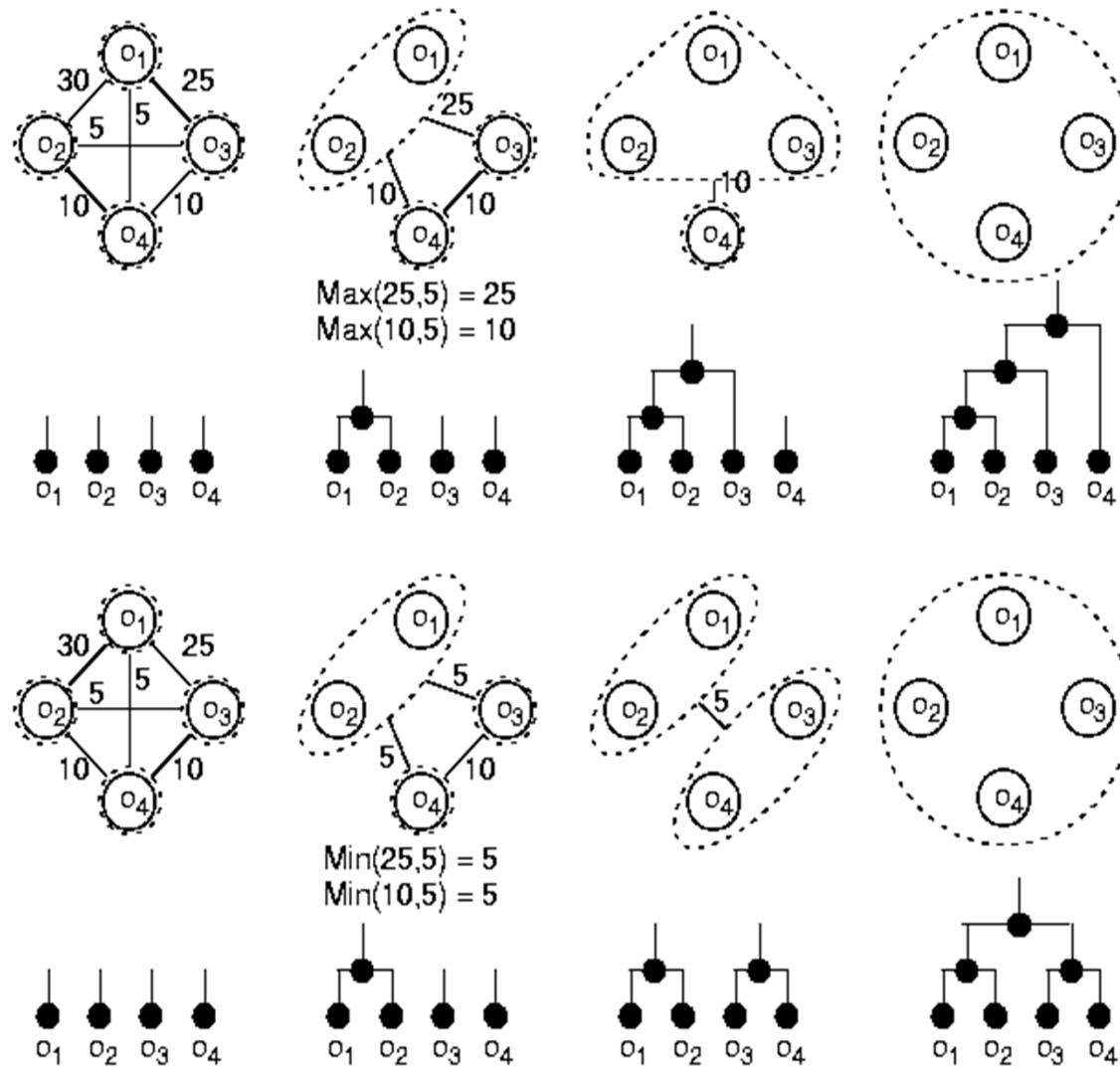
Issues in Partitioning

- Specification abstraction level
- Granularity
- System-component allocation
- Metrics and **estimations**
- Partitioning algorithms
- Objective and closeness functions
- Partitioning algorithms
- Flow of control and designer interaction

Partitioning Methods

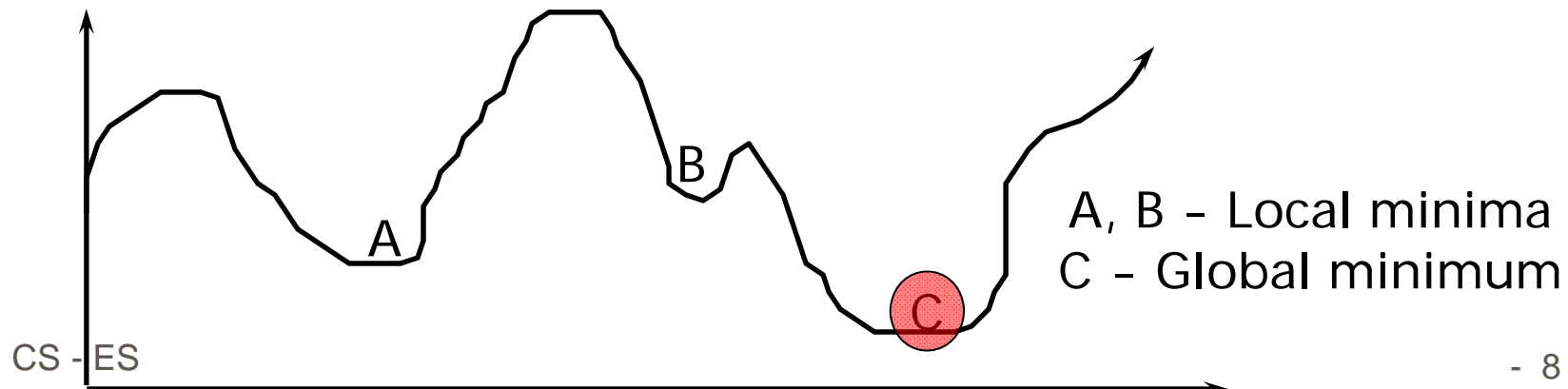
- Exact methods
 - Integer Linear Programming (ILP)
 - ...
- Heuristic methods
 - Constructive methods
 - Random mapping
 - Hierarchical clustering
 - Iterative methods
 - Kernighan-Lin Algorithm
 - Simulated Annealing
 - ...

Example: Hierarchical Clustering



Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent **evaluating large numbers of partitions**
- Iterative algorithms differ from one another primarily in the ways in which they **modify the partition** and in which they **accept or reject bad modifications**
- The goal is to **find global minimum** while performing as little computation as possible



And more ...

Paper	Dynamic/ Static	Strategy	Criteria	Model/ Data Structure	Granularity of Partitioning	Time Complexity
[58]	Static	Simulated Annealing	n/a	n/a	n/a	n/a
[42]	Static	Greedy	Minimal area, data-rate constraints	System Graph Model (like H-CDFG)	operations	linear
[41]	Static	Greedy (see [42])	Minimal area, data-rate constraints	Hierarchical Sequence Graph	operations	n/a
[77]	Static	Simulated Annealing	Minimal communication cost	Petri-nets, (annotated) CDFG	operations	$O(tn)$ t=temperature steps
[34]	Static	Simulated Annealing	Hardware suitability (compare local phase [54])	(extended) C^k syntax graph	basic blocks	n/a
[54]	Static	GCLP	GC objective function (e.g. Area combined with speed)	n/a	Tasks (instruction level subgraphs)	$O(ne)$, e=edges
[96]	Static	Binary Constraint Search	Constraints of encapsulated partitioning algorithm	n/a	n/a	$O(part(S))$ part(S) = encaps. part. alg.
[50]	Static	Dynamic Programming	Temporal size of loops / leaf functions	n/a	loops, leaf functions	n/a
[53]	Static	GCLP (MIBS)	See [54]	CDFG	Tasks	$O(n^3 + n^2B)$, B=bins
[82]	Static	Evolutionary (Genetic)	Minimal area, timing and concurrency constraints	CDFG	functional elements	$O(gp)$, g=generations, p=population
[30]	Static	Clustering	Minimal cost, minimal power, timing and power constraints	Task Graph	task clusters	n/a
[29]	Dynamic	Greedy, Clustering	Minimize area, timing constraints	Task Graph	task clusters	n/a
[65]	Dynamic	Clustering	Area constraints	CDFG	loop clusters	linear
[89]	Static	Evolutionary (Genetic)	maximize fitness (minimize area and interconnect)	DFG	fine:operations coarse:DFGs	n/a
[84]	Dynamic	Evolutionary	Maximum rank (Pareto ranking in power and price)	Task Graph	Tasks	n/a
[91]	Static	Greedy	Temporal size of loops / leaf functions	n/a	loops	n/a
[14]	Static	Dynamic Programming	Minimum latency, resource constraints	DFG	Tasks	polynomial
[12]	Static	Simulated Annealing, Kernighan-Lin	Minimize latency, area constraints	Call graph	functions	n/a

Table 2.1: Inventarization of several papers on hardware software partitioning with corresponding partitioning schemes, criteria, and data structures

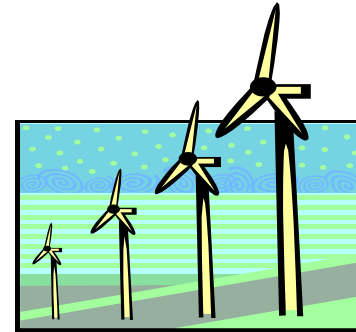
- Architecture Synthesis
- HW/SW Codesign

- ➔ ■ Power Aware Computing
 - (mobile) PA Embedded Systems
 - Power Aware Computing - Introduction
 - Power Optimization
 - Power Estimation

Processing units

- Need for efficiency (power + energy):

Why worry about energy and power?



*“Power is considered as the **most important constraint** in embedded systems“*

[in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

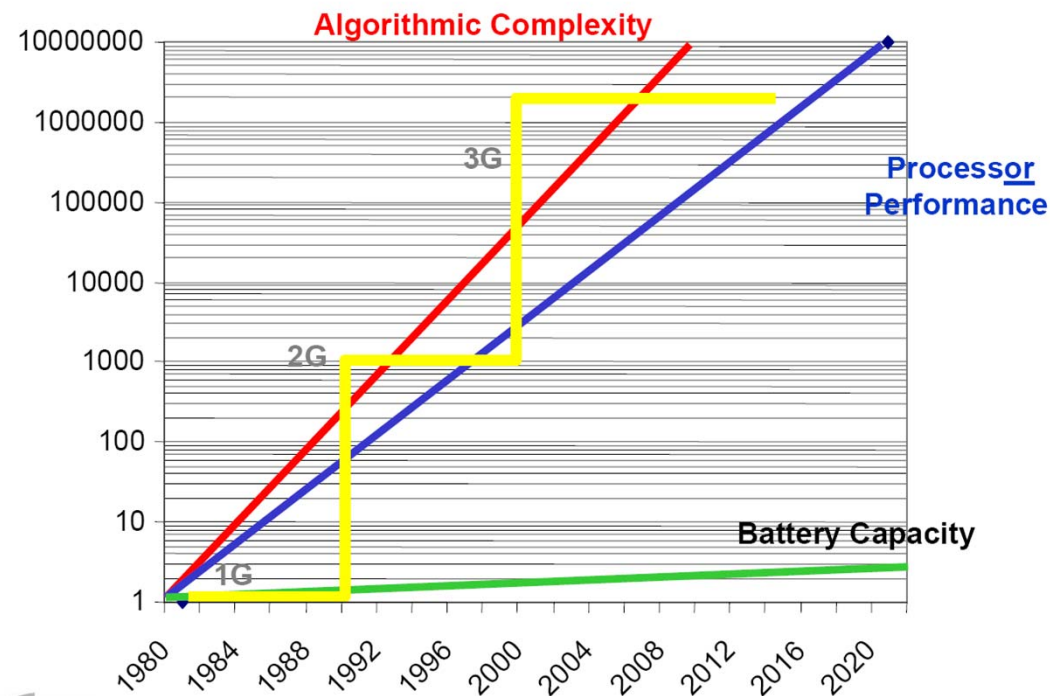
Energy consumption by IT is the key concern of **green computing** initiatives (**embedded computing leading the way**)



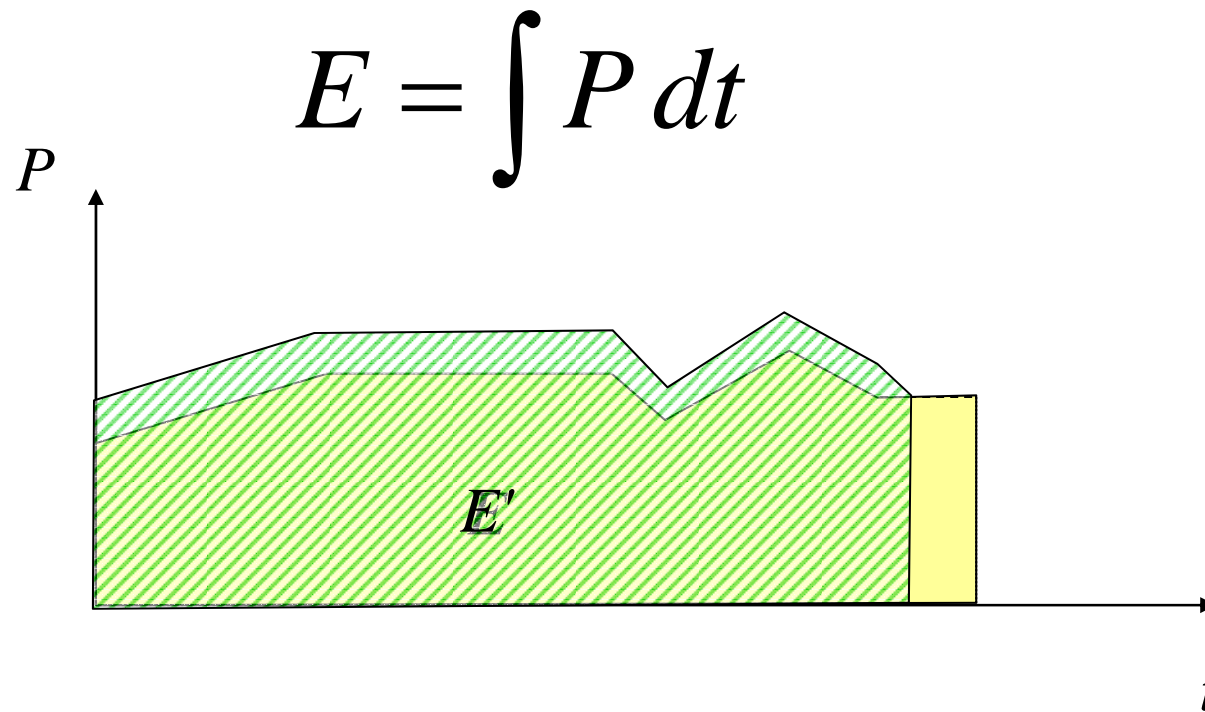
<http://www.esa.int/images/earth,4.jpg>

Motivation (1)

- Rapidly growing market for portable devices
- Requirements: light weight, long battery life, high performance, security
- Moore's law
- Battery technology can't keep up with that pace



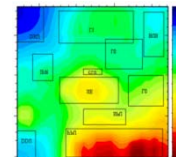
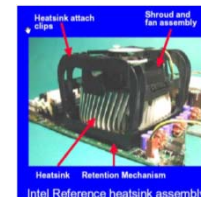
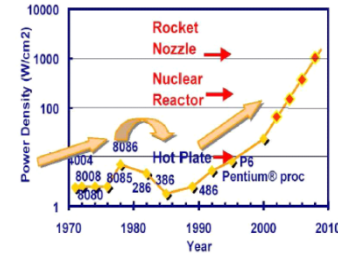
Power and energy are related to each other



In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

Low Power vs. Low Energy Consumption

- Minimizing **power consumption** important for
 - the design of the power supply
 - the design of voltage regulators
 - the dimensioning of interconnect
 - short term **cooling**
- Minimizing **energy consumption** important due to
 - restricted availability of energy (mobile systems)
 - limited battery capacities (only slowly improving)
 - very high costs of energy (solar panels, in space)
 - **RF-powered devices**
 - cooling
 - high costs
 - limited space
 - dependability
 - long lifetimes, low temperatures



CS - ES

Design Constraints (1)

- Battery-powered vs. RF-powered devices

– Battery-powered devices

Requirement: long lifetime (ENERGY opt.)

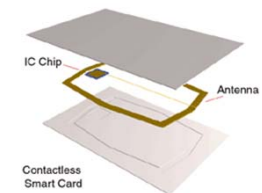
- avoidance of intervals of high discharge current
- reduction of average load current
- allow the battery to recover (power idle states)

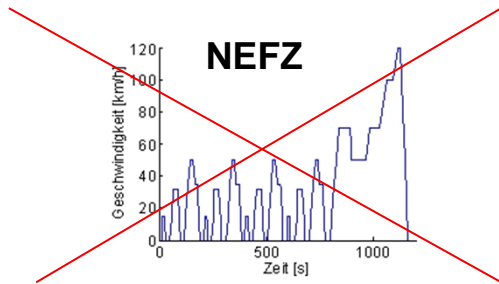


– RF-powered devices

Requirement: no reset and stable communication (POWER opt.)

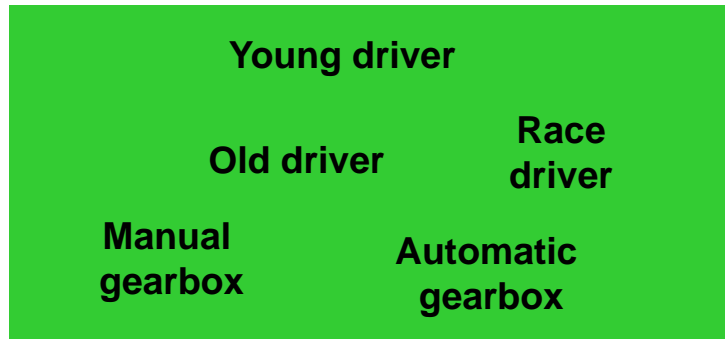
- avoidance of energy consumption exceeding the budget available from the field and internal capacities



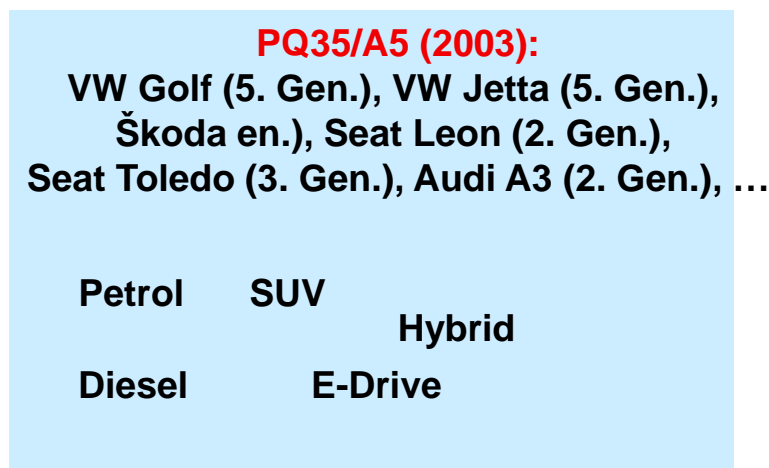
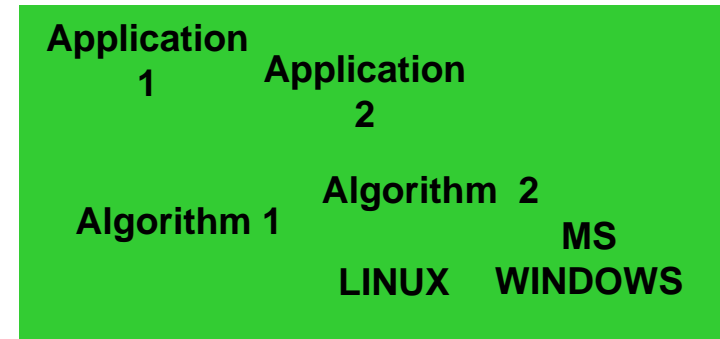


“There’s a great deal that software can contribute to making a system manage power more effectively.”

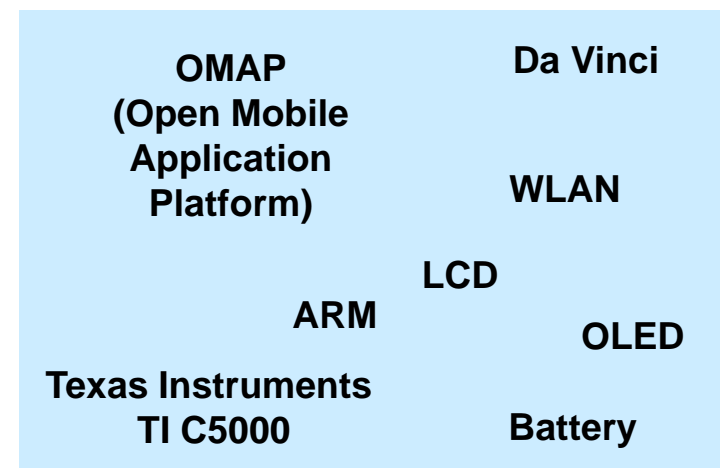
Richard Wirt
Intel Senior Fellow,
General Manager, Software
and Systems Group



Responsible for
the consumption
(fuel/energy) after
production



platform



CS - ES

CAR

ES

Power Aware Computing

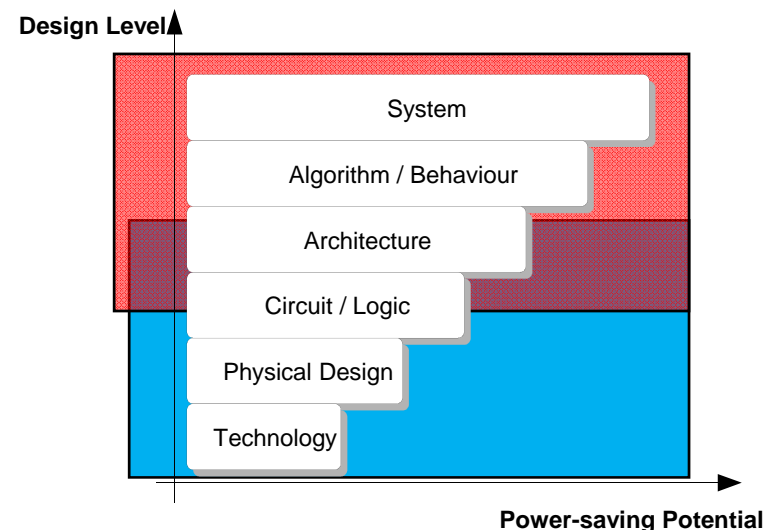
- Power aware computing

- Low power design

- Mobile devices

- high performance
- small
- less power consumption

- Temperature Aware Computing (in general purpose processors)



Power/Energy Optimization Levels

- HW level
 - Low power design (transistors, gates, clock gating, ...)

- Machine code optimization
 - Operand switching
 - Instruction reordering: minimize circuit state overhead
 - Instruction replacing: use low power instructions

- Source level optimization
 - Algorithmic transformations: simplify computation by reducing quality of service
 - Loop optimization

- HW-System level Power Optimization
 - Data Representation (bus encoding)
 - Memory Design Optimization (access, architecture, partitioning)

- System Level
 - Dynamic Power Management
 - Dynamic voltage scaling / dynamic frequency scaling
 - Remote processing

Introduction

Introduction

$$P = P_{SC} + P_{SW} + P_{LK}$$

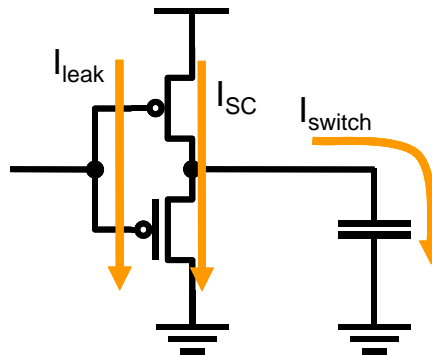
P_{SC} ...Short Circuit Power

P_{SW} ...Switching Power

P_{LK} ...Leakage Power

Minimize I_{leak} by:

- Reducing operating voltage
- Fewer leaking transistors



Introduction

Power consumption of CMOS circuits (ignoring leakage):

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

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

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

V_t : threshold voltage

($V_t < V_{dd}$)

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

Switching Power Minimization

- Supply Voltage Scaling
 - VDD versus delay
 - Compensation of delay overhead
 - ... Reduces circuit speed and throughput
 - Different voltage domains on a chip (Unified power format UPF)
- Switched Capacitance Optimization

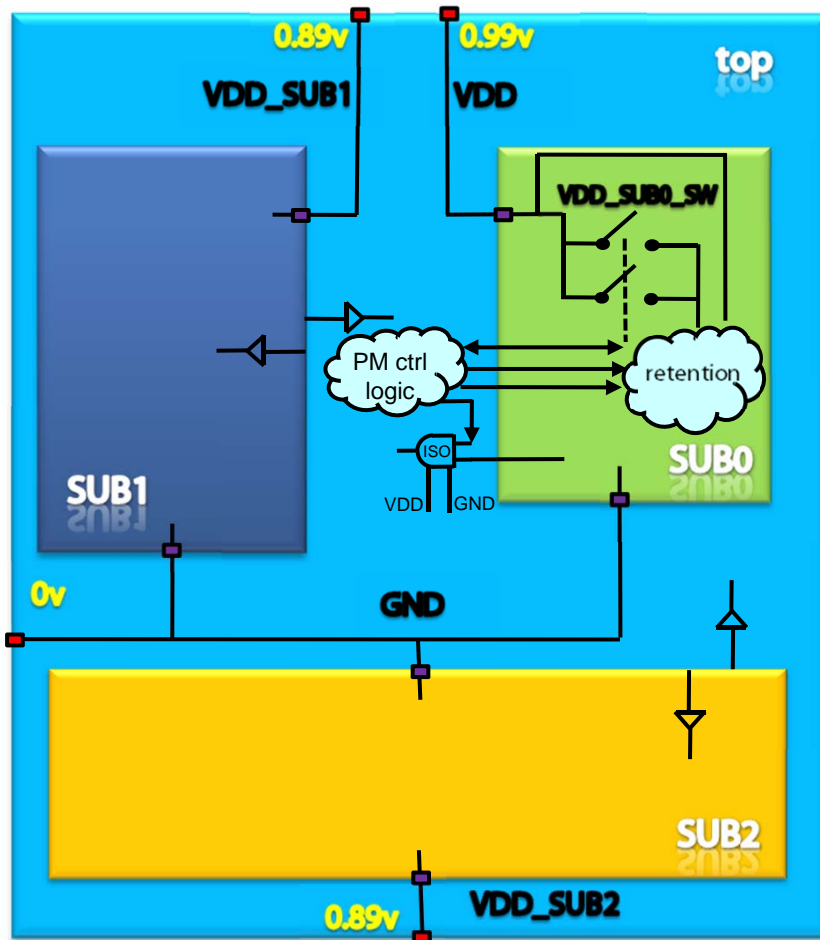
$$C_{eff} = C_L * E_{SW}$$



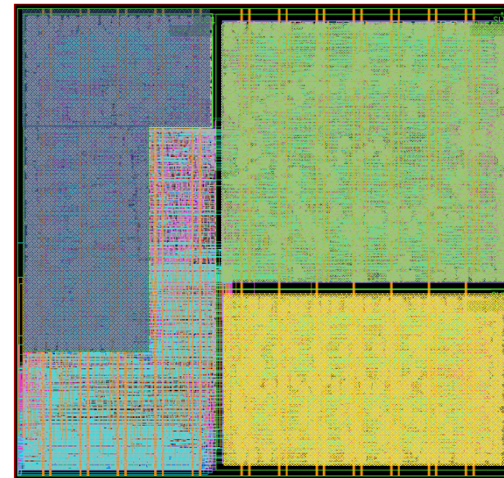
Unified Power Format (UPF)

- Accellera defined the Unified Power Format (UPF) to fit requirements
- IEEE standardization as project 1801
- UPF supplement common HDLs
- UPF extends the functional specification with low power intent
- UPF is specified in a separate file

UPF by example



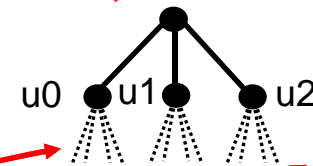
- 45nm TSMC library
- 4 Domains, 2 Hierarchies
 - Top Level – 1.0v constant
 - Secondary level SUB0
 - 0.99v switched constant
 - Secondary level SUB1
 - 0.89v constant
 - Secondary level SUB2
 - 0.89v constant



UPF – domain creation

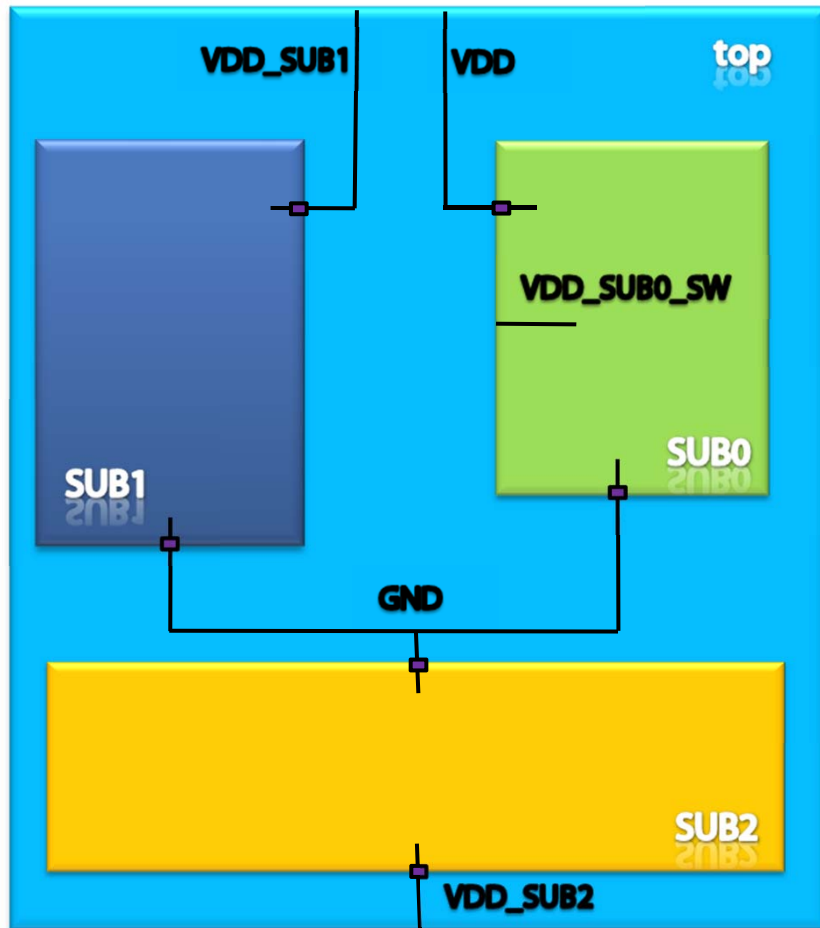


UPF Commands



```
create_power_domain top -include_scope  
create_power_domain SUB0 -elements {u0}  
create_power_domain SUB1 -elements {u1}  
create_power_domain SUB2 -elements {u2}
```

UPF – supply network creation



UPF Commands

```
#Supply net creation for domain top
create_supply_net VDD -domain top
create_supply_net VDD_SUB1 -domain top
create_supply_net VDD_SUB2 -domain top
create_supply_net GND -domain top

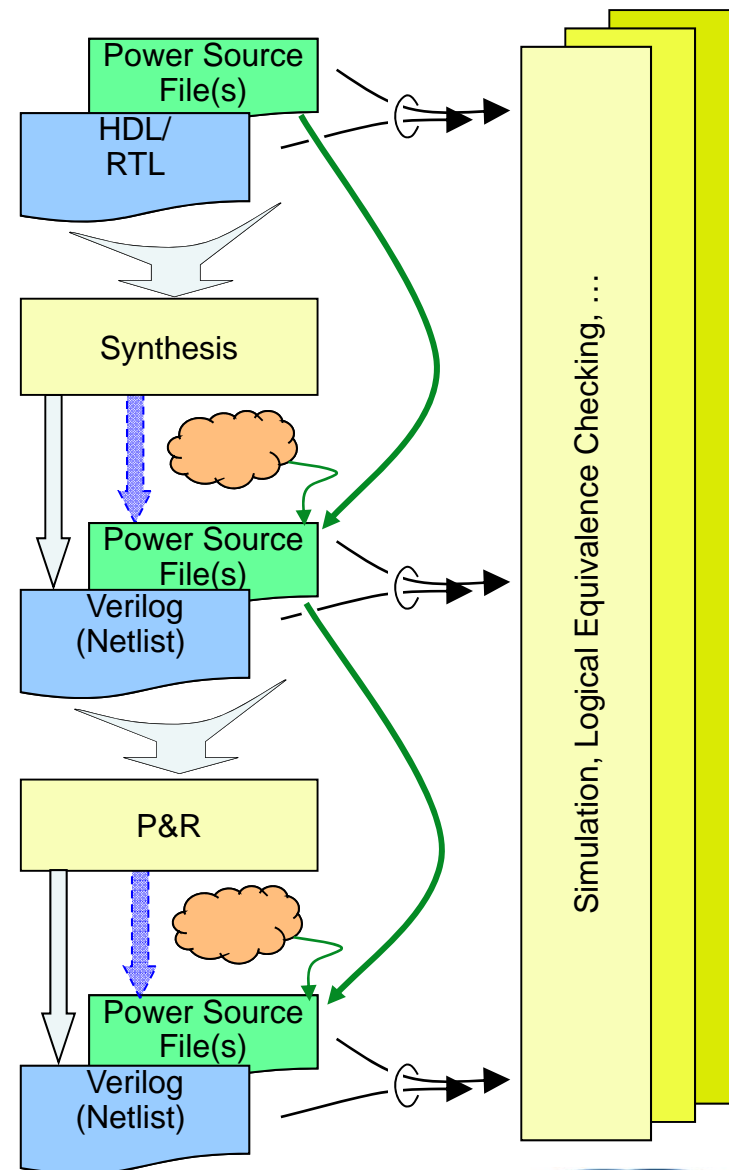
#Supply net creation for domain SUB0
create_supply_net VDD -domain SUB0 -reuse
create_supply_net VDD_SUB0_SW -domain SUB0
create_supply_net GND -domain SUB0 -reuse

#Supply net creation for domain SUB1
create_supply_net VDD_SUB1 -domain SUB1 -reuse
create_supply_net GND -domain SUB1 -reuse

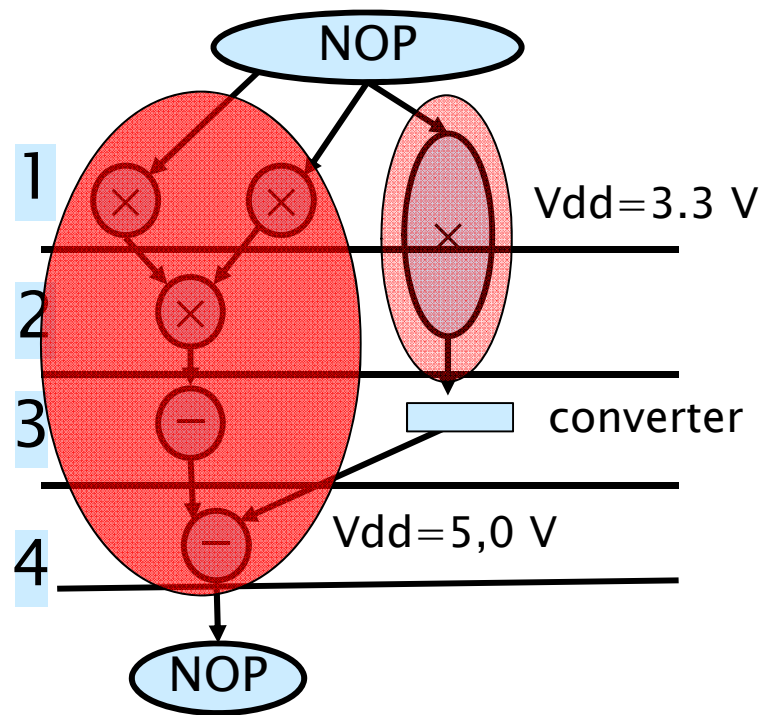
#Supply net creation for domain SUB2
create_supply_net VDD_SUB2 -domain SUB2 -reuse
create_supply_net GND -domain SUB2 -reuse
```

Power Management: Source & Flow

- The traditional Synthesis flow
 - Is **augmented with Power**
- Power source files are part of the design source.
 - **Combined with the RTL**, the power files are used to describe the intent of the designer.
 - This collection of source files is the **input to several tools, e.g., simulation, synthesis, STA, test, formal verification, power consistency checking.**
- The details of the “What” (IP developers → low power) and the “How” (System Integrators) are often produced by different parties.
 - Design refinement



Voltage domains



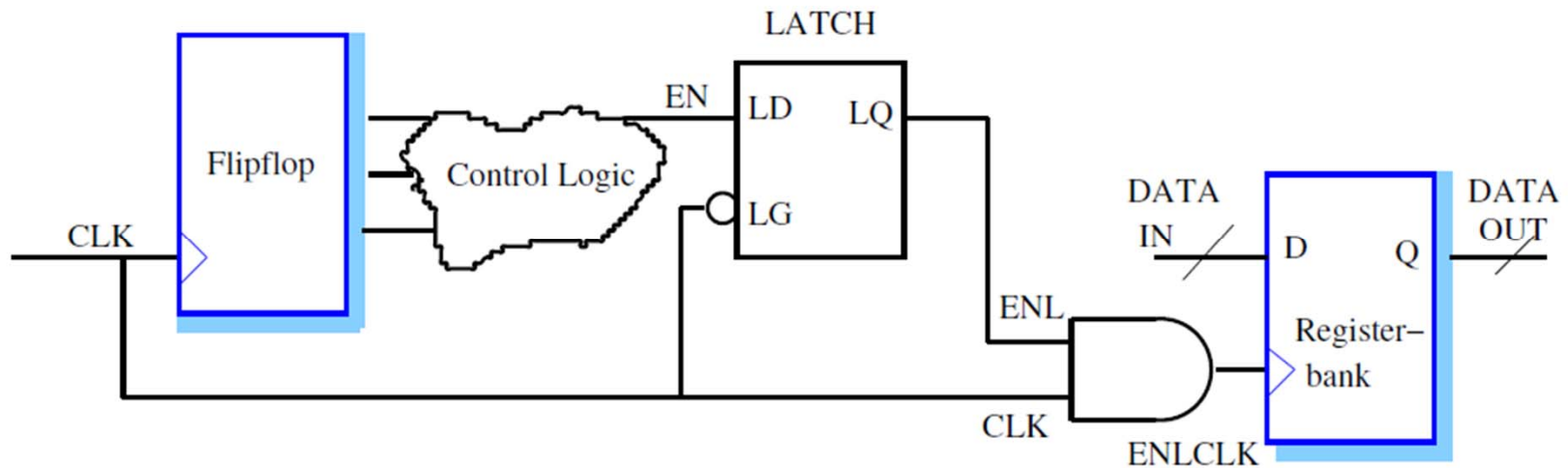
RTL Optimization

RLT Power Optimisation

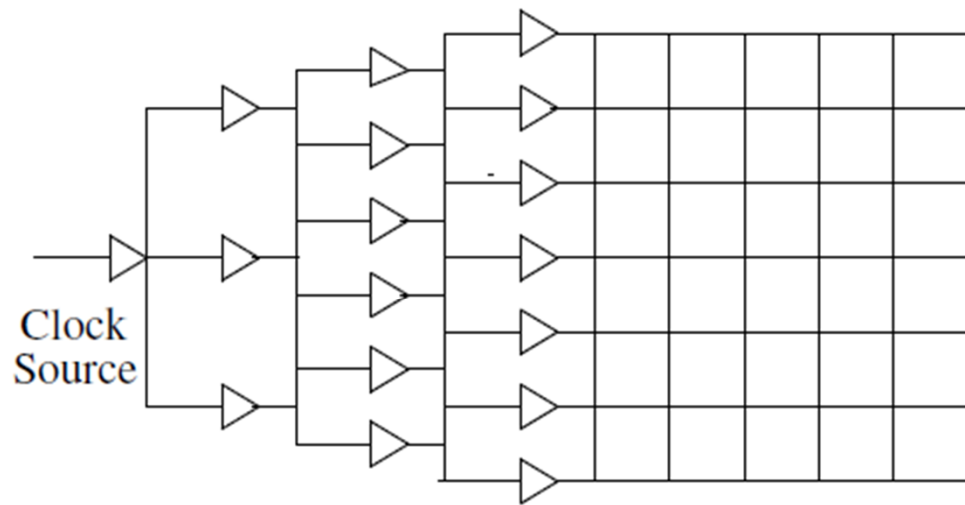
- Dynamic Power Management (DPM)
 - Shut down the blocks which are not in use during some particular clock cycles
 - Approaches:
 - Pre-computation
Selectively pre-compute the circuit output value before they are required Identify small and efficient predictor functions
 - Operand-Isolation
Identify redundant computation of datapath components and isolate them. Can be implemented in HDL
 - Clock-Gating
 - ...

Clock Gating

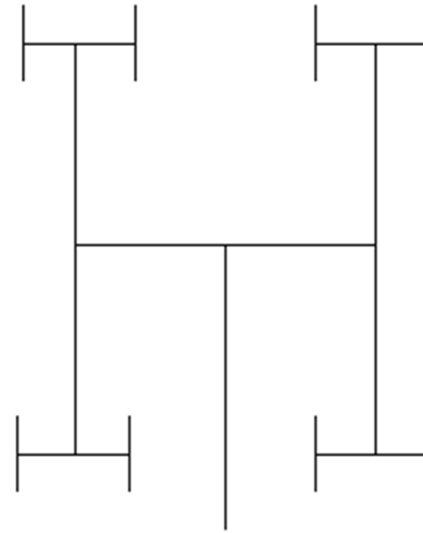
- Stop the clock to registers which are not in use
- Activity-driven clock gating
 - Clock gating should not be applied to high switching activity registers, choice is based on threshold



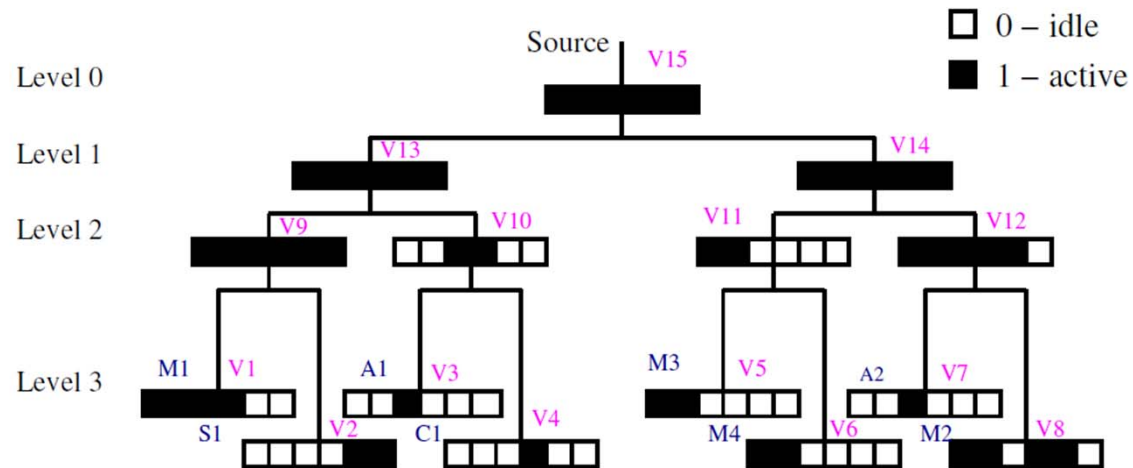
Clock tree



Mesh



H-Tree



Software Power Estimation/Optimization

“Software” Power Dissipation

- Memory System
 - For portable computers: 10% - 25%
 - Reading/writing memory
 - High capacitive data, address lines
 - Cache
 - Avoid Cache Misses -> memory access expensive
 - Smaller
 - Shorter and less capacitive data, address lines
 - Higher performance **and** more energy efficient

“Software” Power Dissipation

- Busses
 - High capacitance components
 - Multiple busses
 - Address
 - Instruction
 - Data
 - Switching activity determined by software
 - Instruction bus: sequence of op-codes
 - Address bus: sequence of data and instruction accesses

“Software” Power Dissipation

- Data Paths
 - ALU (Arithmetic Logic Unit)
 - FPU (Floating Point Unit)
 - Pipelining, Parallelism
 - More stages active
 - Multiple execution units
 - Energy to evaluate expression
 - Shift left vs. multiply by 2

“Software” Power Dissipation

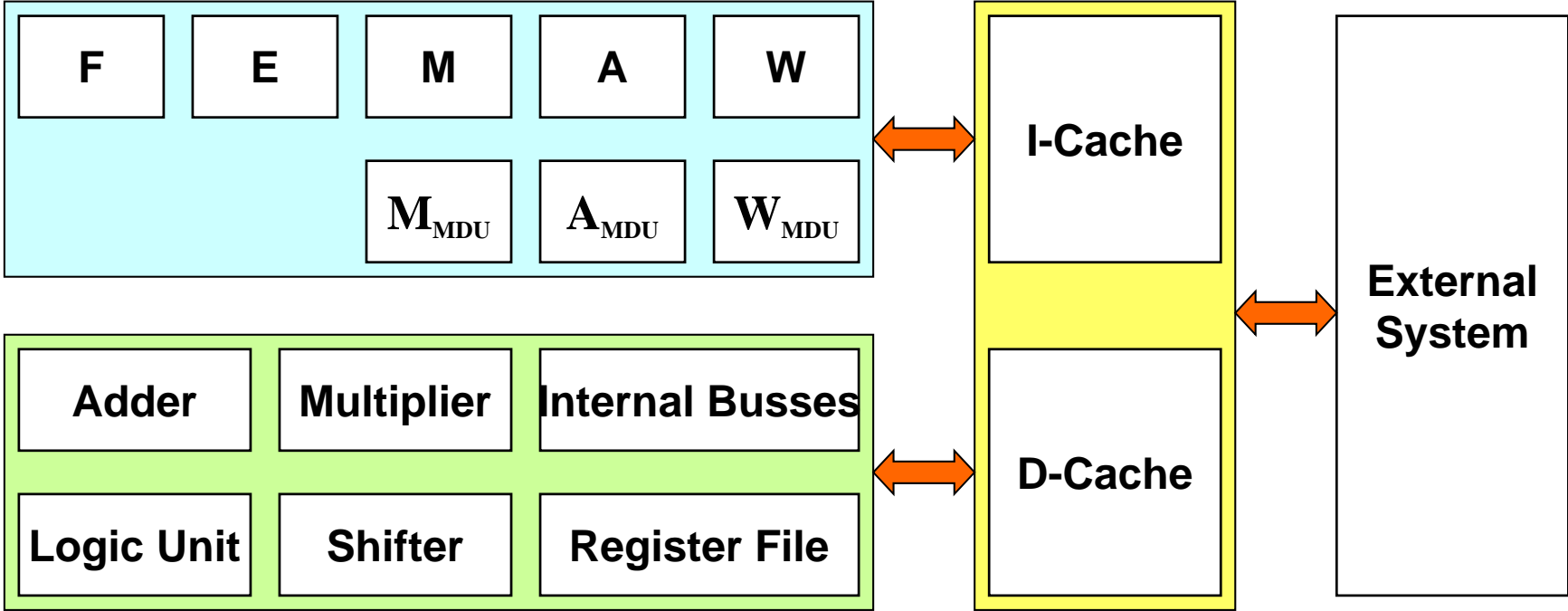
- Other sources
 - Control logic
 - Clock distribution
 - Each instruction cycle -> energy consumption
 - Less cycles -> less energy
 - Make programs fast!

Generic Energy Model

$$E_{total} = \sum_{n=0}^{n_{total}} [E_i(n) + E_d(n) + E_c(n) + E_p(n)]$$

- The overall energy consumption is split into 4 parameters
 - **E_i instruction dependent energy dissipation**
 - Independent on source and target operands and operand values
 - Estimation based on base cost and CSO (Tiwari et all.)
 - **E_d data dependent energy dissipation**
 - Energy consumption of each instruction depends on operands and operand values
 - Hamming distance and hamming weight
 - **E_c energy dissipation of the cache system**
 - Cash hit / miss
 - **E_p memories and peripherals**
 - Power state models
- Huge number of parameters, which have to be characterized

Concept



$$E_{\text{cycle}} = E_{\text{instr}} + E_{\text{data}} + E_{\text{caches}} + E_{\text{external}}$$

Instruction-level power modeling

$$\bar{i} = \frac{\sum_{j=1}^N BC_j + \sum_{j=1}^{N-1} CSO_{j,j+1} + \sum_{j=1}^N IIC_j}{\#CC}$$

- BC ... Base Costs
- CSO ... Circuit State Overhead
- IIC ... Inter Instruction Costs
- #CC ... Number of clock cycles
- N ... Number of instructions
- \bar{i} ... average current

Instruction Path Characterization

Basecosts:

10: loop:

20: add t0, t1, t2

30: add t0, t1, t2

40: add t0, t1, t2

50: ...

2000: b loop

2010: nop

- Independent to the previous state
 - Excludes pipeline stalls, cache misses, bus switching, ...
- Instruction sequence for characterization
 - Repeat the same instruction in a loop
 - Should be long enough to amortize the loop overhead (jump statement at the end of the loop)
 - But, short enough not to cause cache misses
- Calculate the average power and energy
 - By measuring the current, supply voltage and # of execution cycles per instruction

Instruction Path Characterization

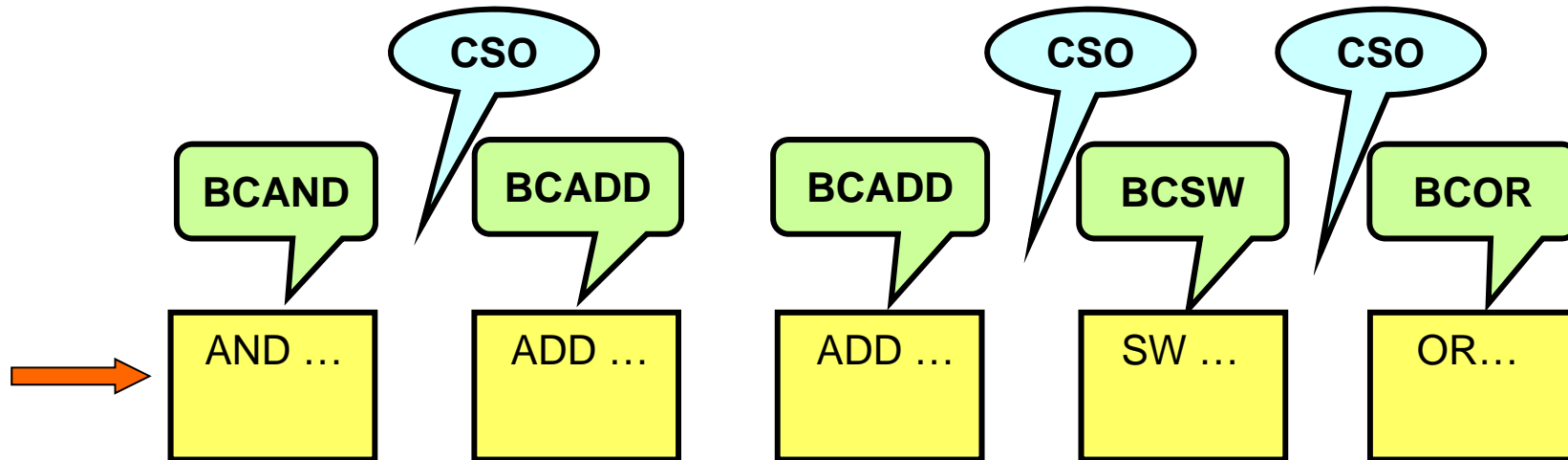
- Associated “power” cost
 - Instruction bus as the op-code switches
 - Switching of control lines
 - Mode changes within the ALU
 - Switching of data lines to reroute signals between ALU and register file
- Pair-wise characterization
- Large space

Circuit State Overhead:

10: loop:
20: add t0, t1, t2
30: or t0, t1, t2
40: add t0, t1, t2
50: or t0, t1, t2
60: ...
2000: b loop
2010: nop

Instruction Path Energy Dissipation

- Considers only instruction flow in pipeline
 - Base Costs (BC)
 - Circuit State Overhead (CSO)



$$E_i = \sum_k BC(i) + \sum_{k=0}^{n-1} CSO(instr[k], instr[k+1], k)$$

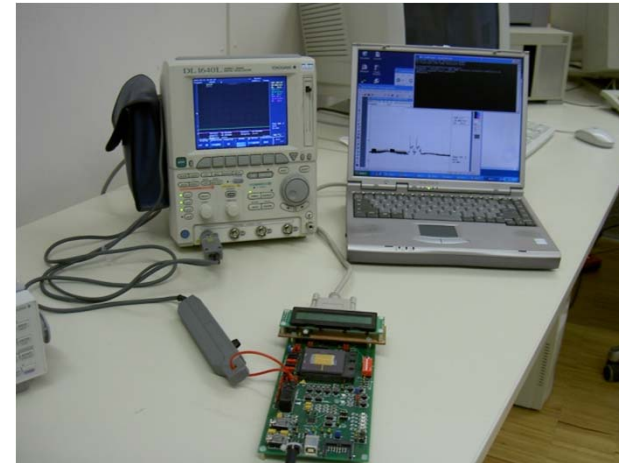
Instruction-level power modeling

- Inter Instruction Costs
 - Instruction/data cache misses
 - Exceptions
 - In general: pipeline stallings
 - Characterization: Define Assembly programs which cause such costs

Instruction Path Characterization

- Measurement Setup
 - High precision ampere meter
 - Measurement of the average current

- BC, CSO calculation

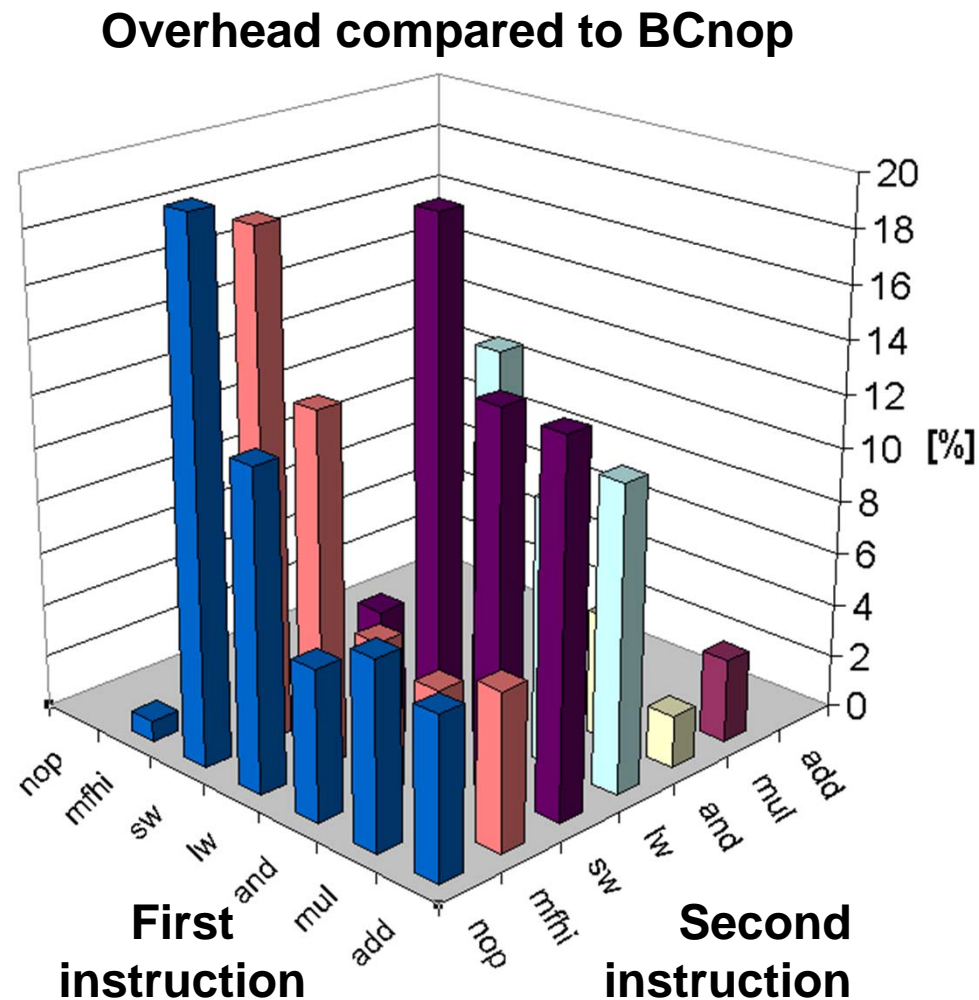


$$BC(i, j) = \frac{T_i \times \overline{BC}_i - N_{i,nop} \times BC_{nop,j} - N_{i,stall} \times BC_{stall} - \alpha_i}{N_i}$$

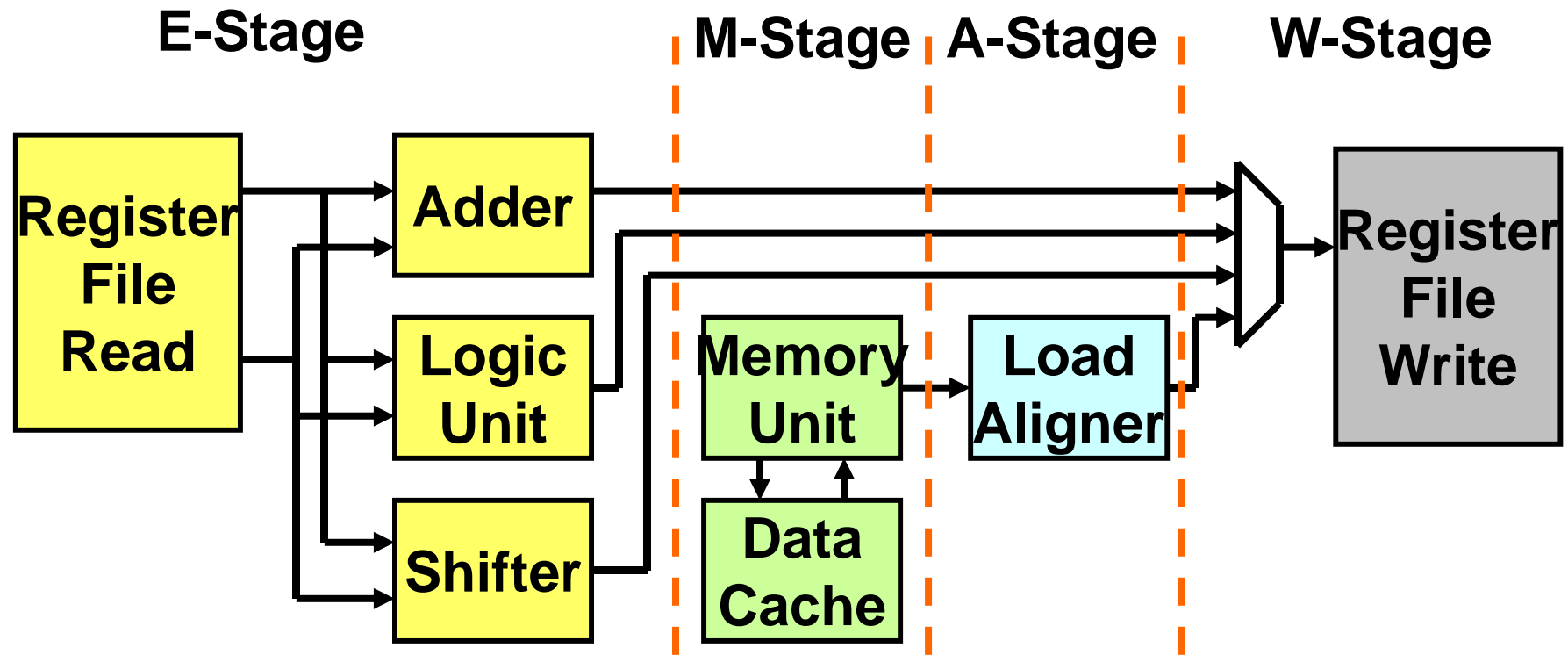
$$CS(i_1, i_2, j) = \gamma_j \left\{ \overline{CS} - \sum_{k=1}^{n_{MDU}} BC_{nop,k} - \frac{1}{2} \sum_{k=1}^{n_{IU}} BC(i_1, k) + BC(i_2, k) \right\}$$

Instruction Path Energy Dissipation

- Basecosts
 - Nearly constant
 - Fast characterization possible
- Circuit State Overhead
 - Strong variation
 - High characterization effort







Data Path Energy Dissipation

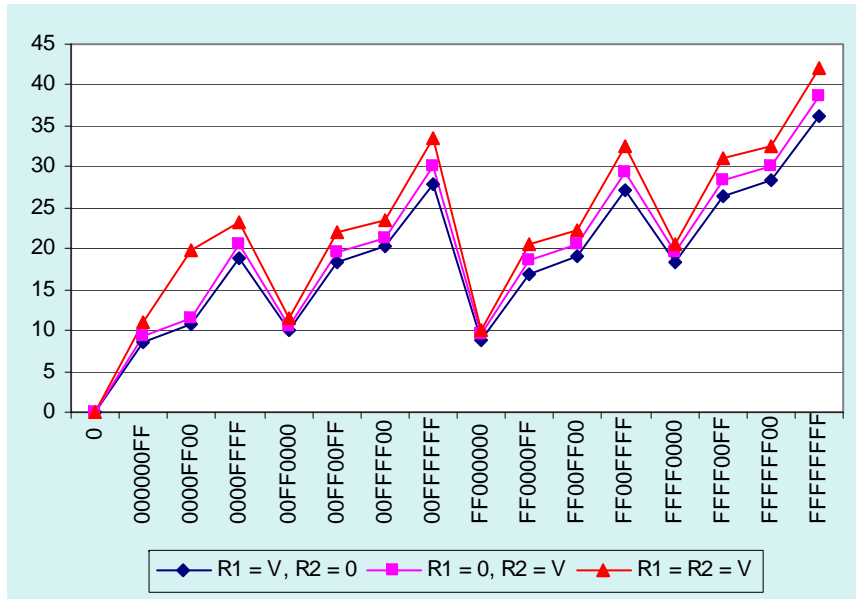


Data Path Energy Dissipation

- Considers only data flow
- Data path model consists of functional units
 - Adder, Multiplier, Logic Unit, Shifter
 - Macro models for power estimation

10:	add t0, t1, t2		t1 = t2 = 0
20:	add t3, t4, t5		t4, t5 = variable
30:	add t0, t1, t2		t1 = t2 = 0
40:	add t3, t4, t5		t4, t5 = variable

Data Path Energy Dissipation

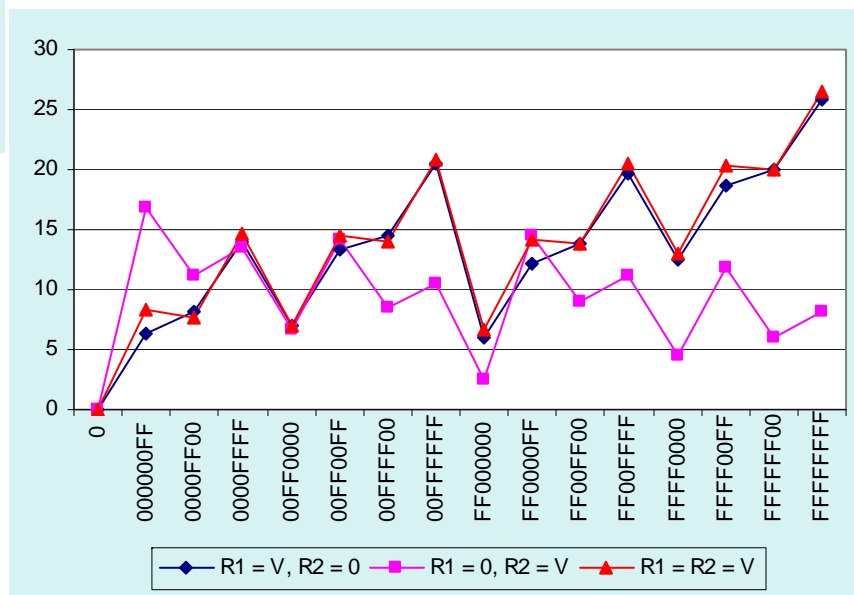


Power footprint of the **adder**

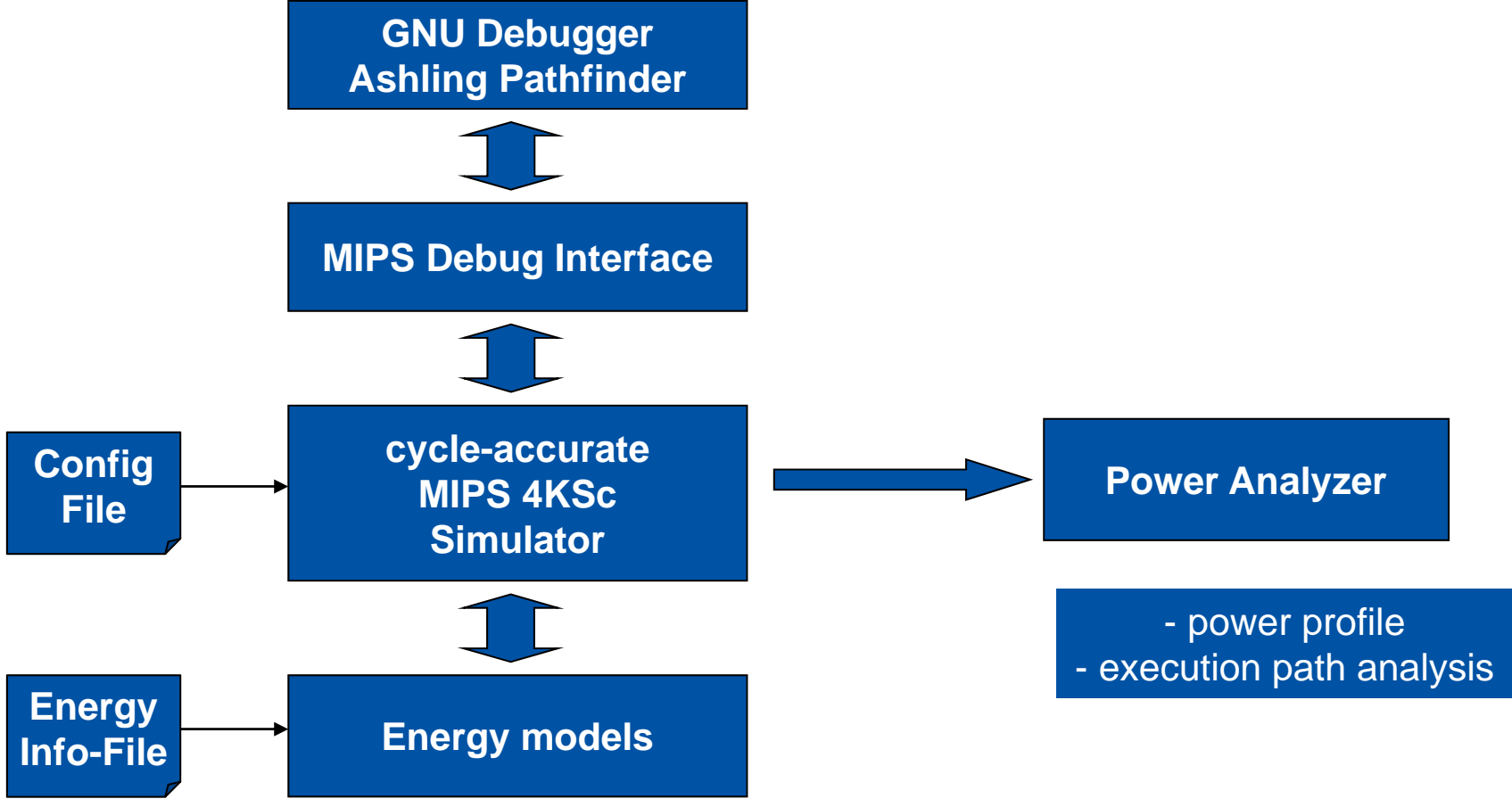
- Symmetric behavior
- Depends on hamming distances
- same behavior for add, sub, ... with regard to the arithmetic function

Power footprint of the **logical unit**

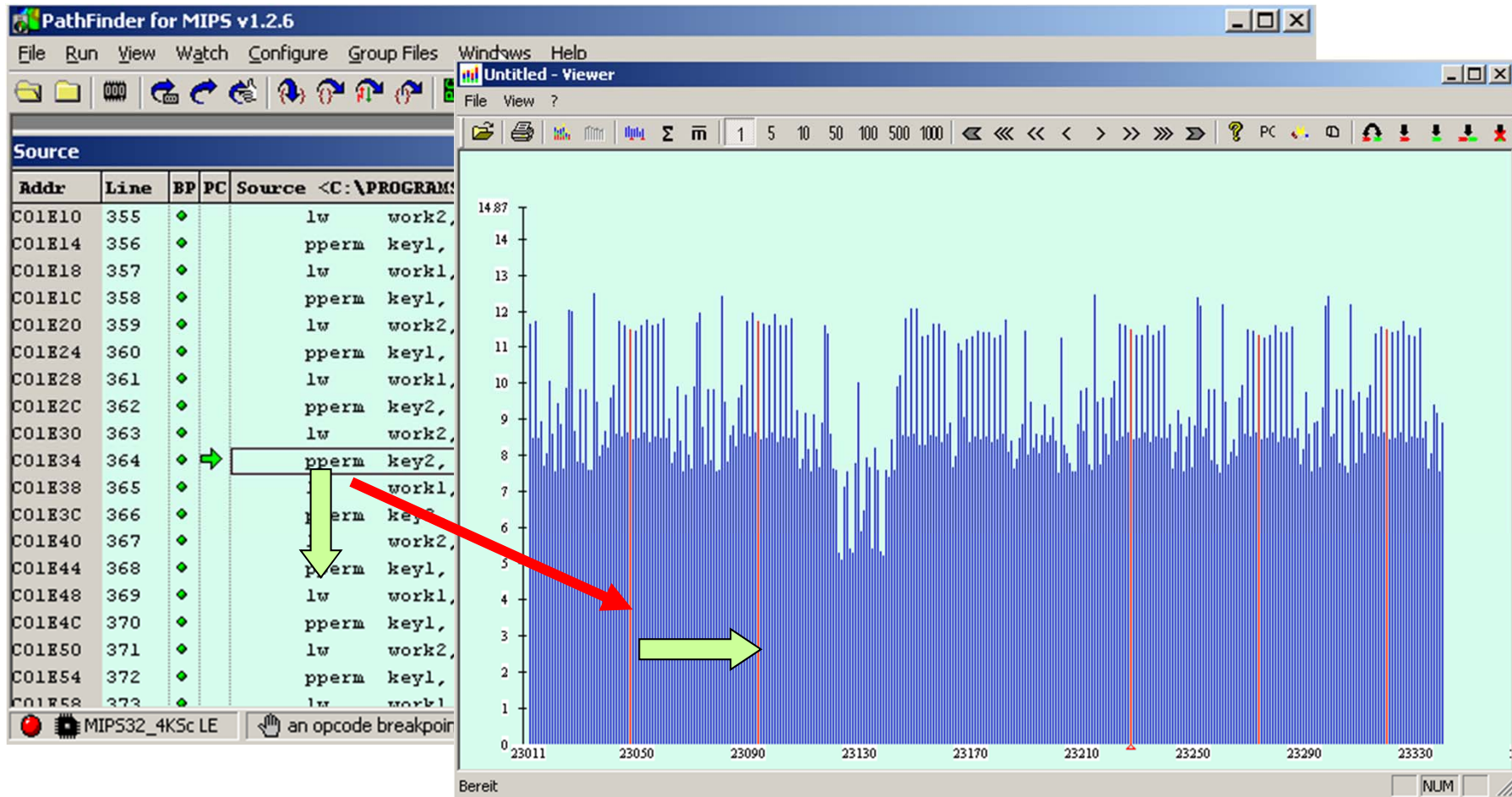
- Asymmetric behavior
- Depends on hamming distances



Software Power Estimation

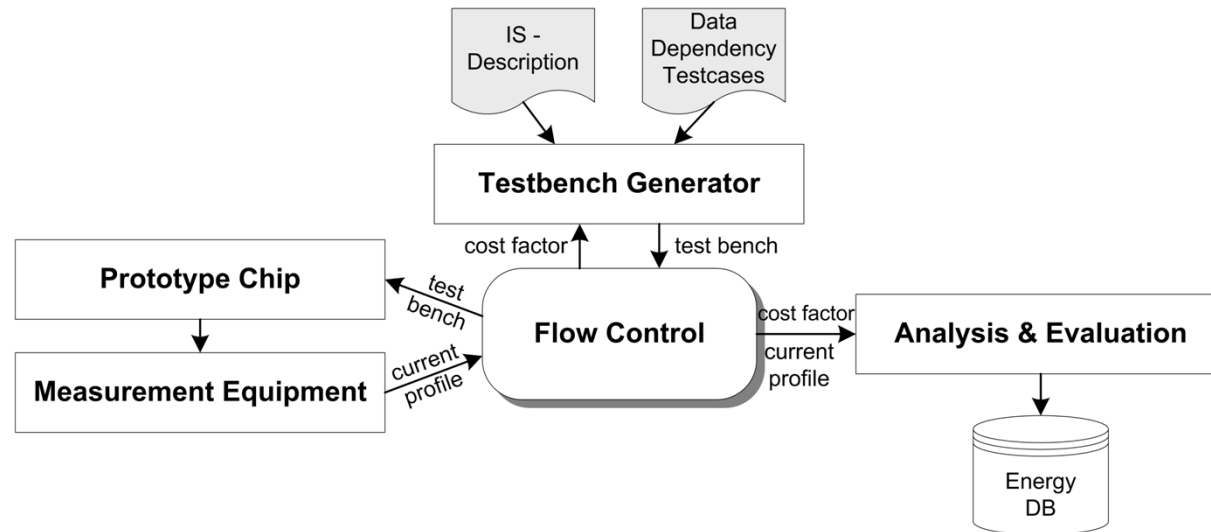


System Integration of IS Power Simulator



Huge number of parameters, which have to be characterized → automated characterization

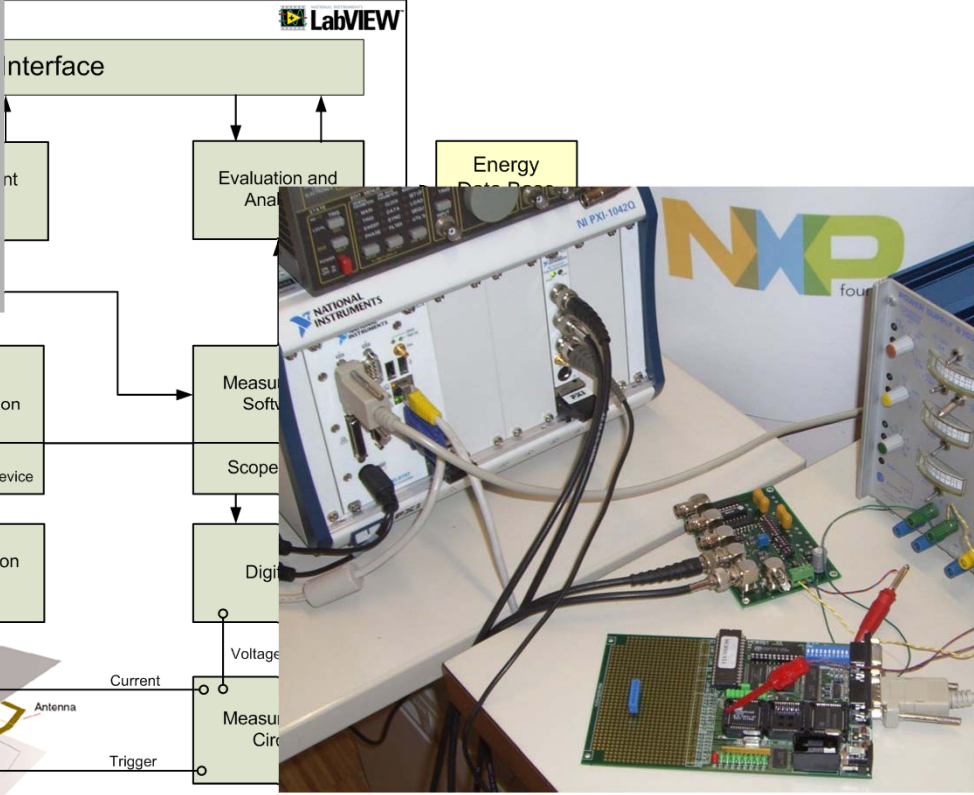
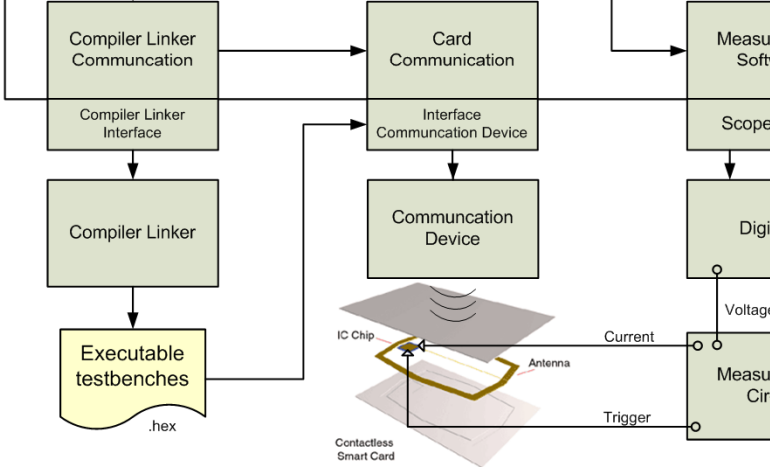
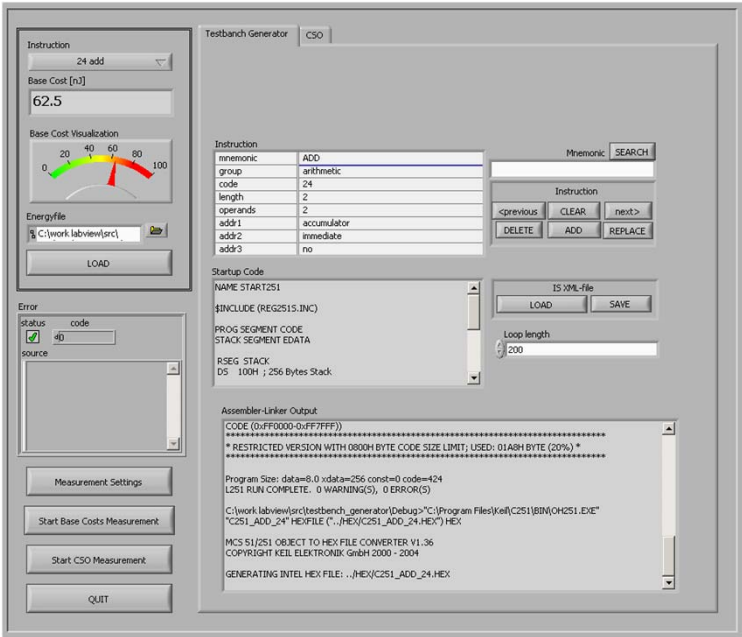
Automated Characterization



- **Testbench Generator**
 - Creates all needed testbenches for a complete characterization
- **Analysis and evaluation unit**
 - Signal preparation and parameter extraction
- **Flow controller**
 - All steps are controlled by the central flow controller - GUI

Automated Processor Characterization

- Use case:
 - 8051 based secure smart card platform
 - ARM7TDMI



Compiler Optimization Techniques

- Reducing memory accesses
 - Control Flow Transformations
 - Special Data Structures
 - Optimized utilization of registers and cache to avoid external memory access
- Low power code generation
 - Usage of low energy instructions, e.g., Shift left vs. multiply by 2
 - **Instruction scheduling** for lower switching activity,
MULT .. after MULT...
 - Alternative register assignments
 - **Operand Sharing / Swapping** (reduction of CSO)

Compiler Optimizations

Optimization Techniques (2)

- Quality of Result / Service
 - Quality depends on available energy
 - Data Types (e.g. float instead of double)
 - Lower quality of multimedia data
- High-Level Code Transformations
 - Loop Fission
 - Loop Fusion
 - Loop Tiling
 - Loop Unrolling

Loop Fission

- Split a loop into **independent loops**
- Achieve **temporal locality**
- When the instruction **cache is small**

```
For i = 1 TO N DO
  A[i] = A[i] + B[i-1]
  B[i] = C[i-1] * X + C
  C[i] = 1/B[i]
  D[i] = sqrt(C[i])
ENDFOR
```



```
For ib = 0 TO N-1 DO
  B[ib+1] = C[ib] * X + C
  C[ib+1] = 1/B[ib+1]
ENDFOR
```

```
For ib = 0 TO N-1 DO
  A[ib+1] = A[ib+1] + B[ib]
```

```
For ib = 0 TO N-1 DO
  D[ib+1] = sqrt(C[ib+1])
```

i = N+1

Loop Fusion

- Merge two loops to reduce the memory accesses
- $B[i]$ is reused
- Instruction cache should be large enough to have both function f and g

```
For i = 1 TO N DO  
  B[i] = f(A[i])  
For i = 1 TO N DO  
  C[i] = g(B[i])
```




```
For i = 1 TO N DO  
  B[i] = f(A[i])  
  C[i] = g(B[i])  
ENDFOR
```

Loop Tiling

- Improve register/cache locality
- Split a **loop nest into tiles (blocks)**

```
for i = 1 to n
  for j = 1 to n
    a[i,j] = b[j,i];
```



```
for tj = 1 to n step 64
  for ti = 1 to n step 64
    for i = ti to min(ti+63,n)
      for j = tj to min(tj+63,m)
        a[i,j] = b[j,i];
```

Loop unrolling

- **for** (j=0; j<=n; j++)
- p[j]= ... ;



```
for (j=0; j<=n; j+=2)
{
    p[j]= ... ;
    p[j+1]= ...
}
```

factor = 2

Better locality for access to p.

Less branches per execution of the loop.

More opportunities for optimizations.

Tradeoff between code size and improvement.

Extreme case: completely unrolled loop (no branch).

Parallel loads and memory bank assignement

- By Lee and Tiwari
- Formulated as a **graph-partitioning problem**
- Solved by simulated annealing
- **Cost function**
 - # of execution cycles required to accomplish all of the memory transfers indicated by the graph
- Graph
 - Vertex: **variable**
 - Edge: **there exists an ALU operation requiring two variables as arguments**
 - After partition
- If an **edge crosses partition**, possible to compact two memory transfers into a **dual-load operation**
- If not, two memory operations are performed sequentially

Parallel loads and memory bank assignment

- Example code

ADD $R_1 \leftarrow a, e$

ADD $R_2 \leftarrow e, b$

ADD $R_3 \leftarrow a, b$

ADD $R_4 \leftarrow b, d$

ADD $R_5 \leftarrow a, c$

ADD $R_6 \leftarrow c, d$

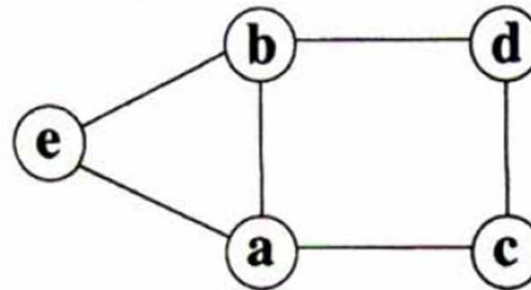
- Total cost

- $e(a,e) = 2$

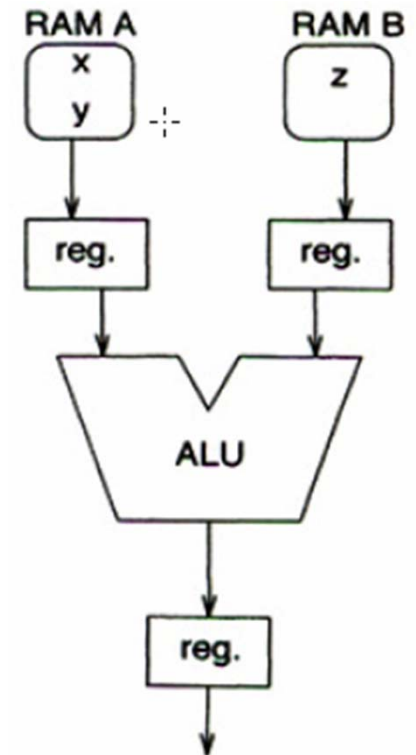
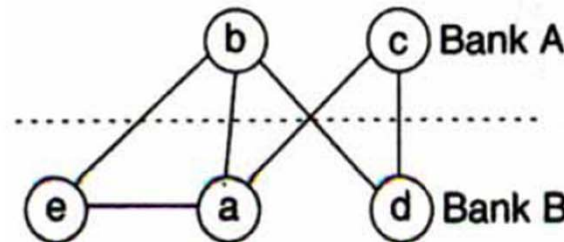
- All others = 1

- Total = 7

- Corresponding access graph



- Partitioned access graph



Roundup - MIPS Energy Optimization

- Software Optimization by Instruction Reordering
 - Only inter-block optimization
 - Depends on algorithm and base block size
 - Optimization result between 0-15%
 - Sometimes pipeline stall can be avoided: up to 25%
- Conclusions Energy Optimization
 - Effects at instruction-level 5%-10% (literature)
 - Source-level transformations necessary
 - Power-aware programming
 - Total profile/energy effected by memory system and periphery