# Power estimation of digital systems

***System level power estimation***

# Outline

- Target
  - mixed Hw/Sw systems
- Power estimation of the Hw part
  - Based on constant additive model
  - Based on power FSMs
- Power estimation of the software part of the system
  - Gate and RT-level techniques
  - Bus activity based techniques
  - Instruction level techniques
- The POET project
  - Towards source-level power aware design and optimization

# Constant additive models

- PowerPlay
  - Spreadsheet framework with WWW interface
  - Power estimation is simply based on manipulation of power, timing and area of the models available for the functional blocks of the systems
  - Models are parametrized and scalable w.r.t. supply voltage and technology
  - The estimation engine gathers data from the models and computes the overall sum (constant additive model)
- Basic model template, power dissipated by a component

$$P=C_{sw}V^2_{dd}f+ I V_{dd}$$

  - $C_{sw}$ average switched capacitance (different model for various components); I static current drawn from the supply
  - No dynamic (profiling) parameters in the model->lower accuracy than behavioral estimation

# Constant additive models

- Computational block model
  - Relates switched capacitance to complexity (e.g., bit width)
  - The more complex the block, the larger the number of coefficients that must be used
  - Example: model with N capacitive coefficients
  - More accurate, analytical (closed) models can be derived if layout of the block is available
- Storage unit model
  - Registers can be treated as functional units
  - The model relates switched capacitance Csw to complexity
  - Example: N words of size W

$$Csw=C0+C1N+C2W+C3NW$$

# Constant additive models

- Controller model
  - Different models for the switched capacitance Csw are used depending on the implementation style (ROM-based vs random logic)
  - Example: Random logic ($N_1$ inputs, $N_0$ outputs, $N_m$ minterms)

$$C_{sw}=C_0\alpha_0 N_I N_o + C_1\alpha_1 N_I N_o$$

- Interconnect model
  - Derived from estimate of interconnect area
    - Rent's rule relates # of connection to a module to the number of gates (param. Feature size, Tr size, #layers, unit gate area, avg gate fanout)
  - Line capacitance is then parametrized by capacitance per unit of area

# Constant additive models

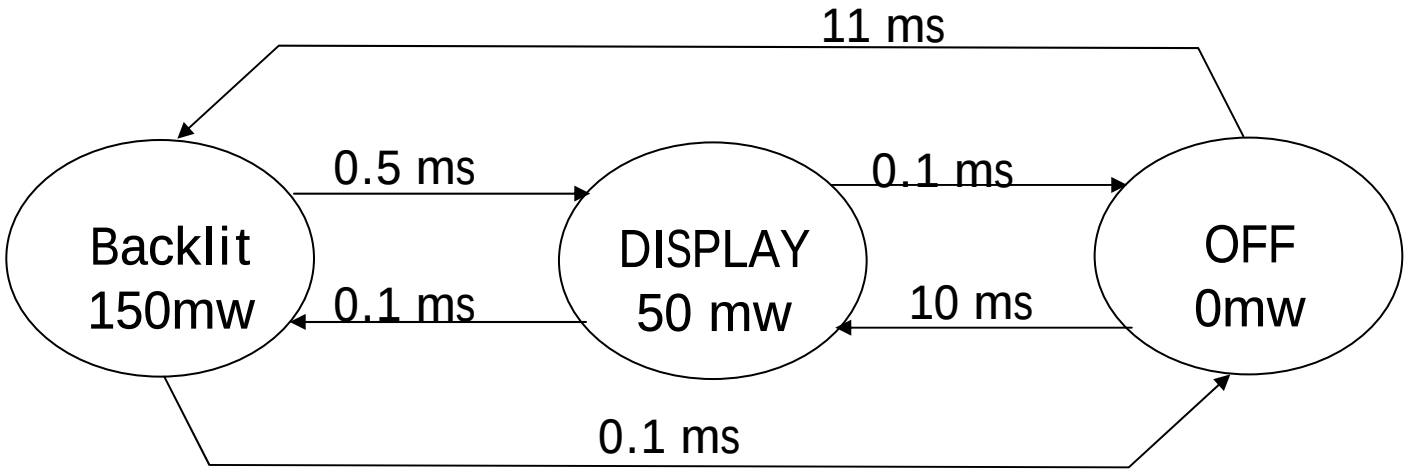
- Other available models
  - Programmable processor
  - Analog circuit
  - DC-DC converter
- Constant additive models of PowerPlay is limited by the fact that workload is left to the designer
  - Sizable estimation errors can be foreseen
- The estimation approach based on Power FSM tries to overcome such limitation
  - It is close to an implementation model, ready to compare alternatives

# Power FSMs (PFSMs)

- Event-driven system model
  - The system is composed by a power manager, a set of resources and a battery
  - Users (environment) interacts with resources by feeding requests to the power manager
- Key feature
  - No overhead for long inactivity (no events)
- Resource model: PFSM
  - States have peak power annotation
  - Transition have performance annotation
  - Transitions are driven by the power manager

# Power FSMs (PFSMs)

- Issues
  - State transition takes time
  - Transition times depend on head and tail states
  - Power depends on state
- Example: LCD display unit



# Power FSMs (PFSMs)

- Improvement resource model: activity-level PFSM
  - Power annotation on a state indicates the peak power
  - Actually power depends on the workload (activity) of the resource
  - Use workload dependent power annotation

    Ps= WL PeakPower
  - Where WL is a Workload factor
- Power manager
  - Receives service requests
  - Dispatches requests to resources
  - Controls the power state of resources according to the selected power management policy
  - Can be implemented either Hw or Sw

# Power FSMs (PFSMs)

- Additional models
  - User/environment (non deterministic FSM)
  - Power models for battery and DC-DC converters
- Power estimation engine
  - Based on simulation
  - Allows capturing non-uniform workloads
    - Activity bursts, inactivity periods
  - Enables to account for device latency
- Results
  - Benchmarked on real-life portable systems
  - Accuracy 10%-15% against measured data
  - High-simulation speed
  - Provides instantaneous power and performance estimates

# Software Power Estimation

- Software-related power is becoming a significant component of the overall power budget
- Fast design space exploration allows
  - Simulation of the specifications, possibly in conjunction with hardware-bound parts of the system
  - Analysis of the power consumption at different levels of abstraction, to tradeoff accuracy and simulation speed
  - Comparison of alternative software codings
  - Comparison of different microprocessors
- To estimate software dissipation it is necessary to identify
  - Sources of software power consumption
  - Power models for the software and the
- Note that the terms *software* encompasses both system and application code

# Sources of Sw power

- Memory system (caches and main memory)
  - Memory accesses are power consuming due to capacitive address line switching, control/decoding logic, internal word and data lines
  - Memory accesses are affected by software (e.g., locality of references)
- Bus (data, instruction, address)
  - Bus accesses are expensive because of line capacitances
  - Instruction and address buses are affected by SW; for data bused accesses mostly decided at runtime
- Data-path (ALUS, FPUs)
  - Datapath accesses are data-dependent, but operation sequence and data dependencies are imposed by software

# Approaches to Sw power estimation

- Different abstraction/accuracy pairs
  - Gate-level
  - RT-level
  - Bus activity based
  - Instruction level
- Gate level
  - Straight forward simulation on gate-level description of processor
  - Useful for benchmarking of other approaches
  - Characteristics
    - Slow but accurate
    - Unavailability of gate level description of non proprietary designs

# Sw power estimation: RT-Level

- Requires a model of the processor in terms of high-level blocks (ALUs and memory) and their relative power dissipation estimates
- Each instruction is characterized in terms of the block that it activates
- The average power is obtained as

$$Power=(1/N_{inst})(\Sigma_I \Sigma_M power_{IM})$$

- Where M is the module activated by instruction I
- Faster than Gate-Level

# Sw power estimation: Instruction-Level

- Possible approaches for instruction power characterization
  - Direct measurement of the currents drawn from the power supply while executing the instructions
  - HDL simulation
    - The instruction are simulated on a processor model in some HDL
    - The processor is plugged into a tester machine and simulation traces are applied. The current drawn is measured by the tester
  - Use simulation of a gate-level description of the processor

# Sw power estimation: Instruction-Level

- A power cost is assigned to each instruction
- Two component of the cost
  - Base cost: static component, without the notion of processor state
  - Dynamic component: account for the circuit state effects, namely the previous processor state
- Dynamic cost modeling allows to capture events depending on sequence of events, like cache misses and pipeline stalls

# Sw power estimation: Instruction-Level

- Base cost is computed
  - By repeating and infinite loop containing a total of N copies of the target instruction to be executed
  - The average current is then measured and the power cost is obtained from the current, supply voltage and cycle/instruction
  - N should be large enough to amortize loop control overhead
- Dynamic effects are currently modeled limiting the analysis windows to pairs of instructions
  - Marginally account for cache misses and pipelines stalls

# Sw power estimation: Instruction-Level

- Circuit state effects are computed as follows
  - Execution of an infinite loop containing N copies for an alternating sequence of target pair of instruction $I_1$->$I_2$
  - The power cost is obtained as for the base cost
  - The averaging process shields the costs for $I_1I_2$ and $I_2I_1$
- The cost of a program can be so summarized

  Cost (program) = $\Sigma_i (B_iN_i) + \Sigma_{i,j} (O_{ij} N_{ij}) + \Sigma_k E_k$
  - $B_i$ Base cost of instruction i
  - $N_i$ Number of occurrences of instruction i
  - $O_{ij}$ Dynamic cost of sequence i->j
  - $N_{ij}$ Number of occurrences of sequence i->j
  - $E_k$ Other effects obtained from program profiling

# A new functional approach working at instruction level

- Problem definition
- Model for functional decomposition
- Single processor model
- Multi processor model
- Experimental results
  - Identification of functionalities
  - Instruction characterization
  - Generalization properties
- Concluding remarks

# Goals

- Definition of a formal instruction-level model, based on a *functional* decomposition of the μP activities
- Intra-processor generalization
  - Capability to cope with incomplete instruction Set (IS) power characterization
- Inter-processor generalization
  - Capability to predict power characteristics of a μP based on the data of other μPs and its IS
- Simplification of the measurements
- Statistical validation of the measurement data

# The model: definitions

- *Functionality* $F_i$: set of activities involving -partially or totally- one or more μP units
- *Space-disjoint*: if $F_i$ and $F_j$ involve different units
- *Time-disjoint*: if $F_i$ and $F_j$ are accomplished at different times
- *Compatible model*: A set of functionalities forms a compatible model iff the current $i_s$ absorbed by each instruction can be expressed as a linear combination of the currents $if_j$ associated with $F_J$

$$i_s = \sum_{j=1..k} (if_j \cdot a_{s,j})$$

  - The coefficients $a_{s,j}$ characterize each instruction in terms of the different functionalities
  - Adherence to physical reality imposes $if_j > 0$

# The model: single processor

- The energy $e_s$ absorbed by instruction **s** is:

$$e_s = \sum_{j=1..k} e_{s,j} = V_{dd} \cdot i_s \cdot n_{ck,s} \cdot \tau$$

- Let $IN = \{i_s \cdot n_{ck,s}\}$

$$IN = A \times IF + R$$

- Solving this equation in the least square sense, gives estimate of the functionality currents

$$IF_{est} = IF + A^* \times R$$

where $A^* = (A' \times A)^{-1} \times A'$

# The model: single processor

- A statistical analysis proves the formal correctness
- Applicability to the available measures can be justified through a $Z_{0.95}$ or $Z_{0.99}$ test
- The linearity of the model and its statistical properties allows to
  - Derive the model parameters $IF_{est}$ from a limited set of instruction measures (*learning-set*)
  - Extrapolate the energy of other uncharacterized instructions (*generalization-set*)

# The model: multiple processors

- For the same type of instruction
  - the *absolute* current strongly depends on the considered μP,
  - the *relative* currents ($i_{rel,s}$) are nearly independent of the μP
- A general model based on a set P of p μPs for learning has been defined using the $i_{rel,s}$ defined as:

$$i_{rel,s} = i_s / i_{ref} = \sum_{j=1..k} if_{rel,j} \cdot a_{s,j}$$

k = # of functionalities considered

- For the generic q-th μP of P, characterized by $IN_{rel,q}=\{i_{s,rel} \cdot n_{ck,s}\}$ and $A_q$

$$IN_{rel,q} = A_q \times IF_{rel,q} + R_{rel,q}$$

# The model: multiple processors

- Solving the above equation in the least square sense and considering that the model should depend on a general set of parameters $IF_{rel}$ instead of microprocessor-specific:

$$IF_{est\ rel,q} = A_q^* \times IN_{rel,q} =$$
$$= IF_{rel} + A_q^* \times R_{rel,q}$$

- Adding up the relations for all the p μP of the set

$$(1/p) \cdot \Sigma_{q=1..p}\ IF_{est\ rel,q} =$$
$$= IF_{rel} + (1/p) \Sigma_{q=1..p}\ A_q^* \times R_{rel,q}$$

# The model: multiple processors

- Result
  - average of the estimated parameters of each μP composing P is an adequate estimator for the parameters of the general model
  - Statistical analysis proves the formal correctness of this choice
  - Increasing the number of considered processors the variance of the parameters decreases as:

$$VAR[IF_{est}] = (1/p^2)\sum_{q=1..p} VAR[IF_{est,rel,q}]$$

# Functionality identification

- Identification of the functionalities: compromise between accuracy and specific knowledge on μP architecture
- Selection of five functionalities, based on measured power consumption figures
  - Principal component analysis shows that no functionality can be neglected without affecting model accuracy

| Functionality | Activity | Normalized contribution (typ) |
|---|---|---|
| **F&D** | **Fetch and Decode** | **0.08** |
| **Br** | **Branch** | **0.08** |
| **WrReg** | **Write Register** | **0.13** |
| **A&L** | **Arithmetic and Logic** | **0.20** |
| **Ld&St** | **Load and Store** | **0.51** |

# Functionality identification

- The values of $a_{s,j}$ have to be determined according to the identified functionalities
- Example: decomposition in F&D and Exec

$$a_{s,F\&D} \cdot if_{F\&D} + a_{s,Exec} \cdot if_{Exec} = i_s \cdot n_{ck,s}$$

  - where, a reasonable choice for $a_{s,F\&D}$ and $a_{s,Exec}$ is:

$$a_{s,F\&D}=n_{ck,s,F\&D},\ a_{s,Exec}=n_{ck,s,Exec}$$

- General case: $F_j$ is involved in the execution of the instruction s if the *activation coefficients* $b_{s,j}=1$
- The relation between $b_{s,j}$ and $a_{s,j}$ is

$$a_{s,j} = \begin{cases} b_{s,j}\, n_{ck,s,F\&D} & j = 1 \\ b_{s,j}\, w_s & j = 2 \ldots k \end{cases} \qquad w_s = \begin{cases} 0 & \sum_{i=2}^{k} b_{s,j} = 0 \\ n_{ck,s,Exec} \Big/ \sum_{i=2}^{k} b_{s,j} & \text{otherwise} \end{cases}$$

# Estimation on a single processor

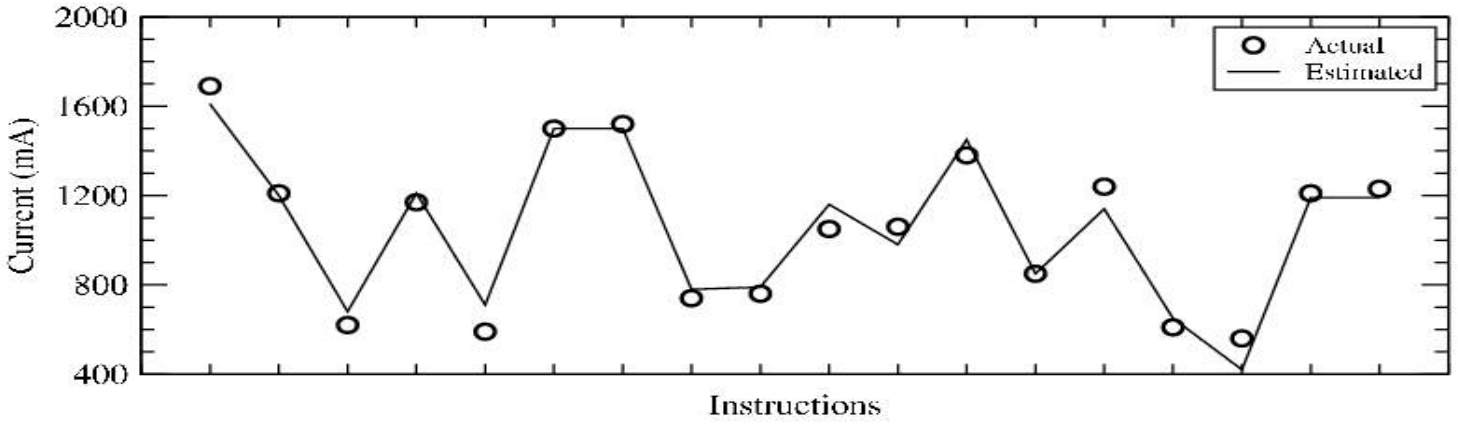
- The stimulated functionalities depend on the operation class and the addressing mode

| Class | Functionalities |
|---|---|
| **Arithmetic & Logic** | **F&D, A&L** |
| **Data Transfer** | **F&D** |
| **Control** | **F&D, Br** |
| **System** | **F&D, Br** |
| **Floating Point** | **F&D, A&L** |
| **Decimal** | **F&D, Br** |
| **String** | **F&D, A&L, WrReg** |

| Addressing mode | Sample | Functionalities |
|---|---|---|
| **Register** | **R2** | **WrReg** |
| **Relative** | **10(R2)** | **Ld&St** |
| **Auto-increment** | **(R2)+** | **Ld&St, A&L, WrReg** |

# Estimation on a single processor

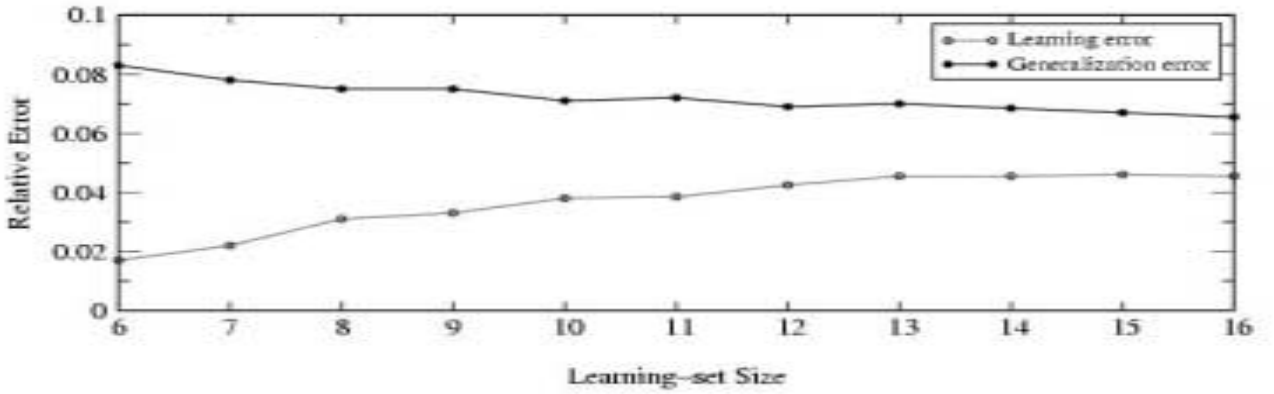
- 6 μPs considered, with similar results
  - ARM7, StrongARM, i960JF, i960HD, SPARC, i80486DX
    - Example I486DX. Gaussian noise Hypothesis holds: $\mu_R$=9.94 falls in the range $z_{0.95}$=±40.
    - Comparison with 18 energy characterized instructions




| Functionality | F&D | Br | WrReg | A&L | Ld&St |
|---|---|---|---|---|---|
| **Current [mA]** | **421** | **355** | **228** | **228** | **505** |
| **STD** | **48** | **26** | **46** | **38** | **39** |

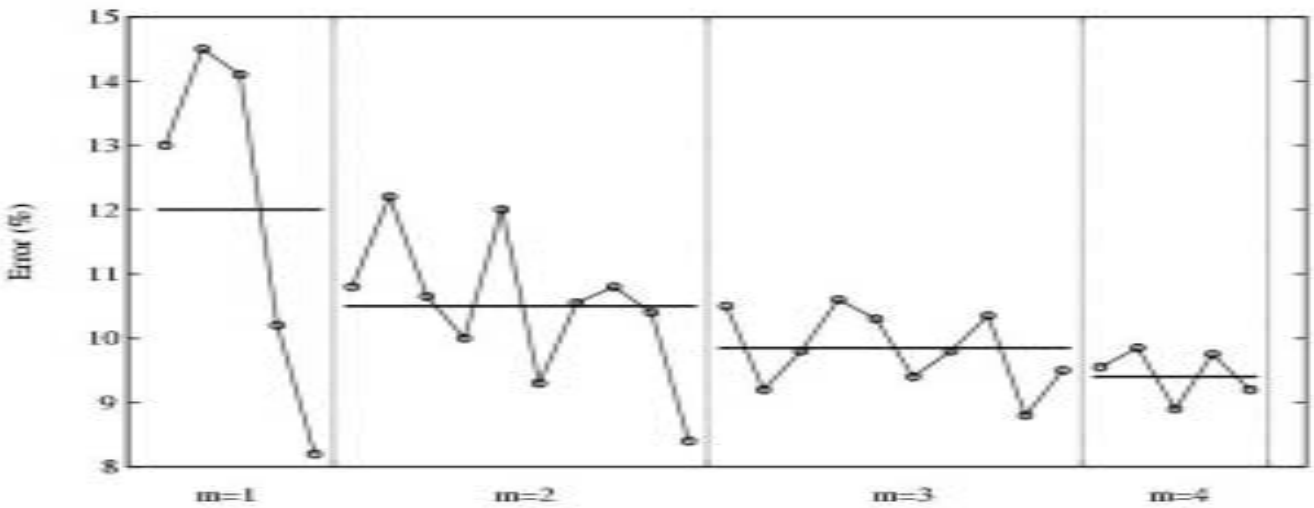
# Generalization on single processor

- Procedure
  - ◎ Select learning-set from characterized instructions
  1. Estimate the currents of the remaining instructions (generalization set)
  2. Compute the learning and generalization errors
     - Errors tend to compensate (mean ~$10^{-10}$): absolute error values have been used to assess the methodology accuracy, that is always < 9%



# Generalization over different μPs

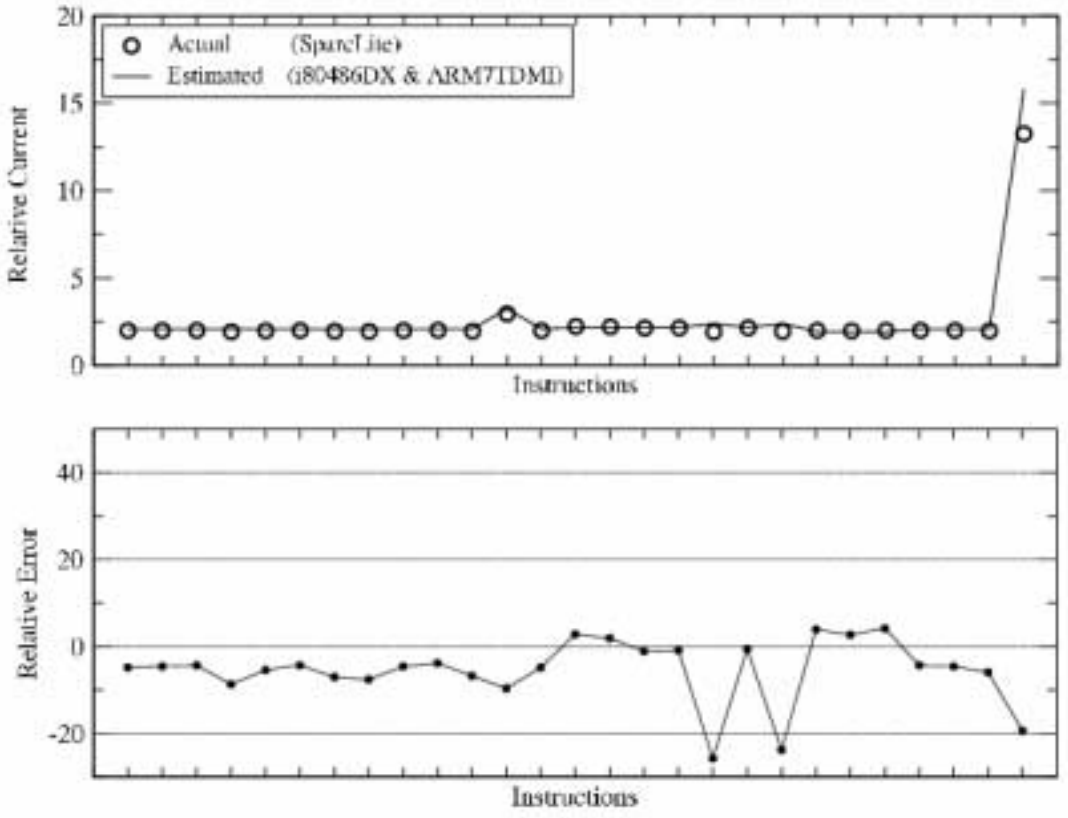
- Combinations of μPs are used to generate different μP learning-sets, whose estimated model parameters allow prediction of the parameters of other μPs




- According to the theoretical model, general mean error decreases as the μP learning-set size increases, and similarly the variance

# Generalization over different μPs

- Example: estimation of model parameters for the i486 based on those of SPARClite and ARM7TDMI



# Results

- Identification of a widely applicable general model to estimate power consumption of 32-bits μPs
  - Single μP: copes with partially characterized Instruction Sets
  - Multiple μPs: considering relative currents, allows prediction of the characteristics of a μP capitalizing the knowledge of other characterized μPs
  - Multiple μPs: if needed, the current per clock cycle of a single specific instruction might be used as reference to compute absolute values

# Current enhancements

- The presence of dynamic effects (assembly level) has been added according to other models considering higher-level features of code execution (see POET project)
- Execution time on a pipelined and/or superscalar architecture depends on
  - The nominal execution time *n(s)* of instruction s
    - Is the nominal latency of an instruction (static measure)
    - Can be found in the documentation of the processor
  - The execution time overhead *oh(s)* due to stalls
    - Depends on the status of the pipeline(s)
    - Is a dynamic measure
  - The actual parallelism *p(s)* that instruction s exploits
    - Depends on the status of the pipeline(s)
    - Is a dynamic measure
- This is summarized by the relation

$$CPI(s) = p(s) \times [\, n(s) + oh(s) \,]$$

# POET tools and methodology

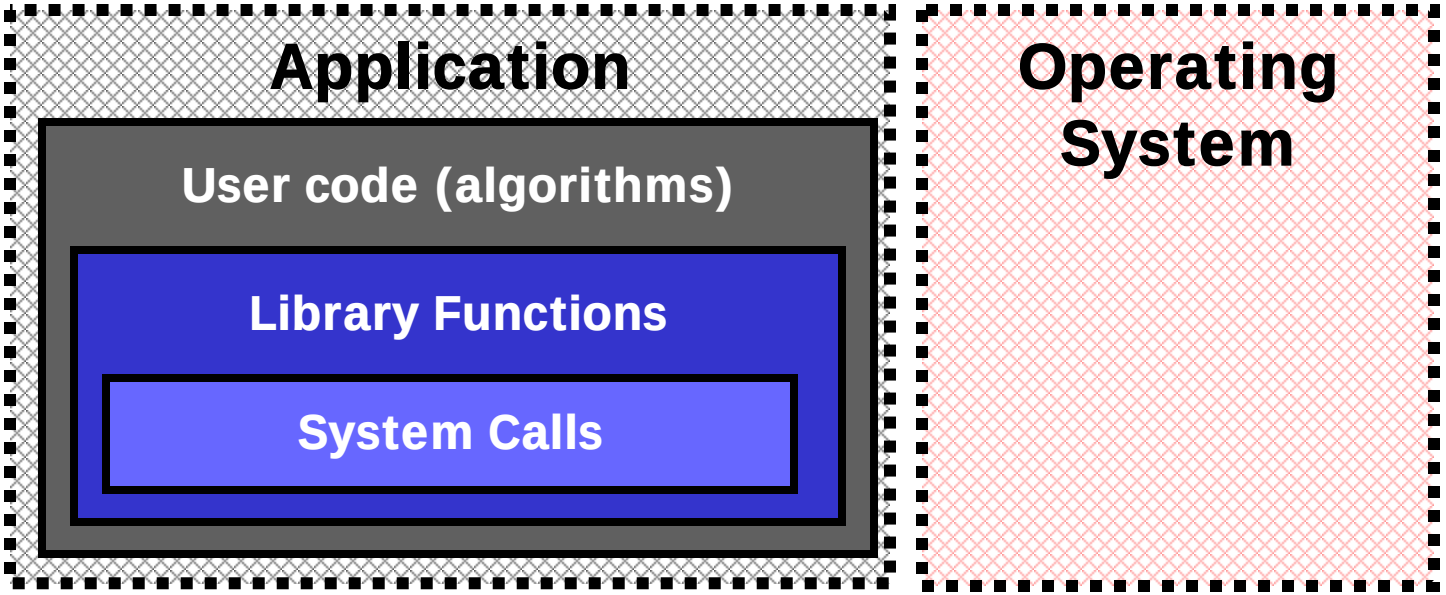
| Assembly-level | Source-level |
|---|---|
| **Third-party tools** | **Third-party tools** |
| Compiler/Linker | C preprocessor |
| Disassembler | OS SDK |
| **Proprietary tools** | **Proprietary tool** |
| Assembly annotator | Code estimation |
| Behavioral simulator | Code transformation |
| **Model** | **Model** |
| Pipeline architecture | Grammar-based |
| Memory hierarchy | Library BlackBox Models |
| | OS BlackBox Models |

**Libraries and Operating System**

**Graphical User Interface**

# Source level - Models

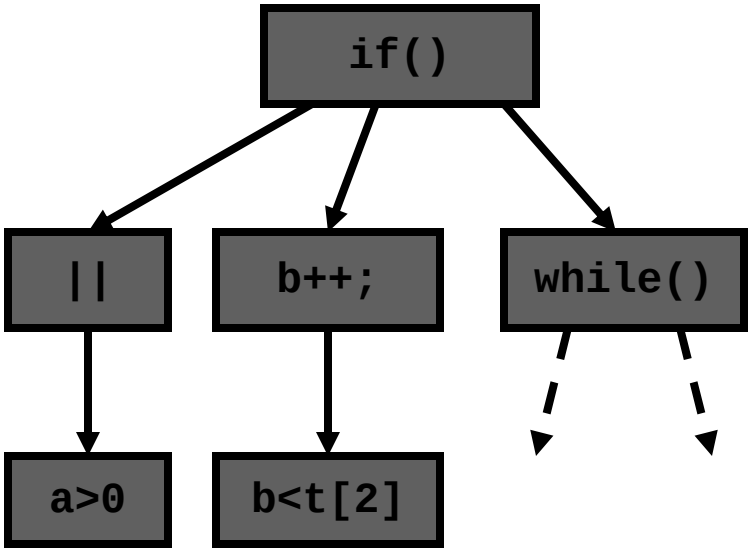
- Timing model
  - Deterministic execution time contribution
  - Statistical execution time correction
  - Statistical black-box library function and OS models



# Source level - Models

- Source code
  - A tree of atoms
- Atom
  - Statement
    - Entire
    - Part
  - Defined by a grammar
  - Atom name
    - Left-hand side symbol
  - Atom structure
    - Right-hand side terminal symbols

```
...
  if( a>0 || b<t[2] ) {
    b++;
  } else {
    while( ... ) {
    }
  }
...
```



# Source level - Models

- Each atom $\gamma_k$ is characterized by
  - A determinisc time contribution t
    - Local to the specific atom
    - Based on a virtual architecture/compiler pair
  - A statistical time correction $\delta$
    - Global to more atoms
    - Based on the results of actual compilation
  - An average current absorption per clock cycle
- Thus:

$$t_{est}(\gamma_k) = t(\gamma_k) + \delta(\gamma_k)$$

$$e_{est}(\gamma_k) = t_{est}(\gamma_k) \times i_{ave}(\gamma_k) \times V_{dd} \times T_{ck}$$

# Library analysis - Models

- Concentrates on execution timing
  - Highly data-dependent
  - Huge amount of data to be processed
- Energy is derived based on average power dissipation data of the target architecture
- Based on statistical analysis
  - Library functions used in different contexts
    - Computation-dominted applications
    - Control-dominated application
    - Text processing
  - Construction of a function timing catalog

# Library analysis - Models

- To derive a black-box model
  - A function must be executed with different data
- This requires
  - Describing the arguments semantics in a formal way
    - The C typesystem is not very tight
    - The same C type is used for data with different semantics
    - The semantic of a formal argument gives hints on the way the datum is manipulated by the function
  - Describing the statistical distribution of the data that the function is supposed to process in the specific application

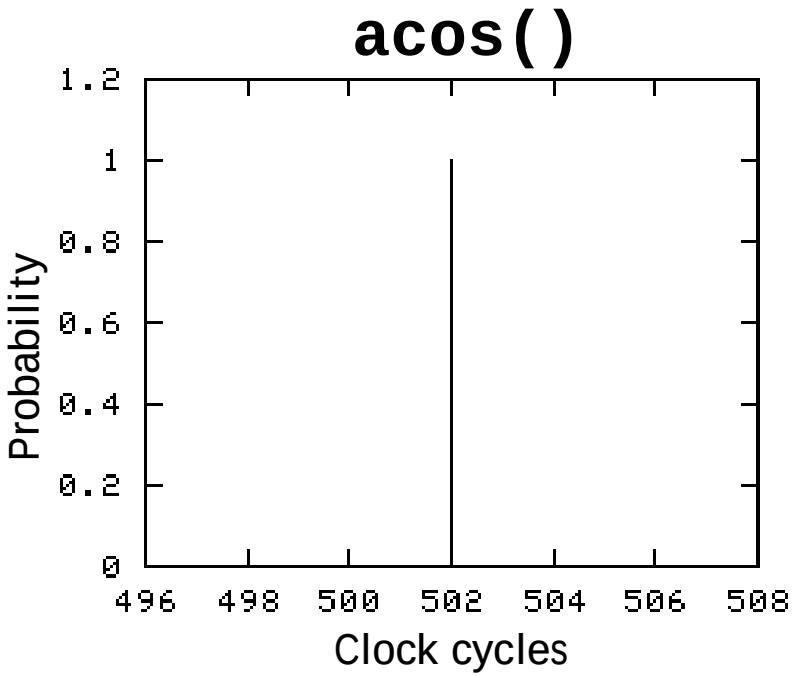
# Library analysis - Models

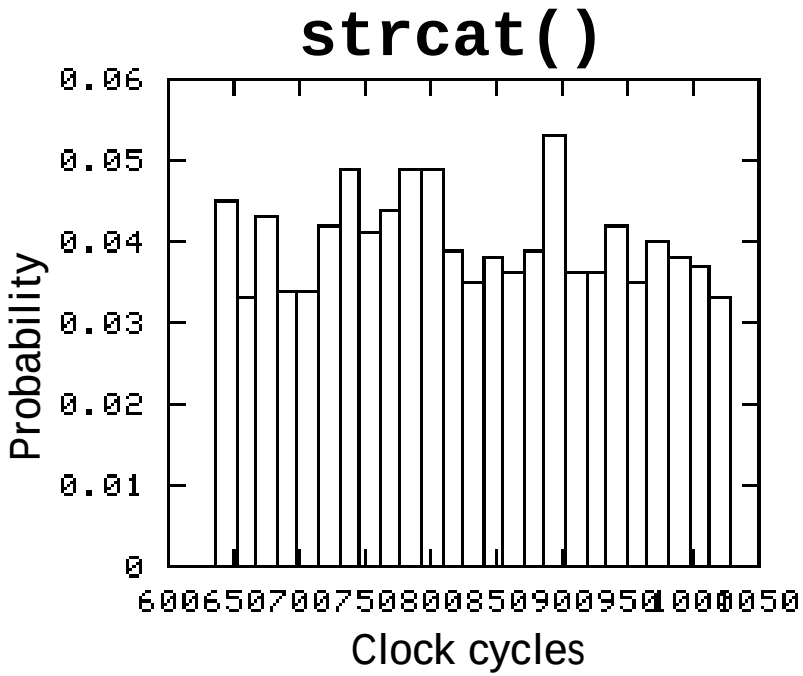
```
int strlen( char* s )
```

| | |
|---|---|
| `char*` | **Formal type** |
| `string` | **Semantic type:** `string`<br>**D-parameter:** `length` |
| `length` | **Distribution:** `gaussian`<br>**Parameters:** `avg`, `std` |
| `avg=60`<br>`std=12` | **Distribution parameters** |

# Library analysis - Results

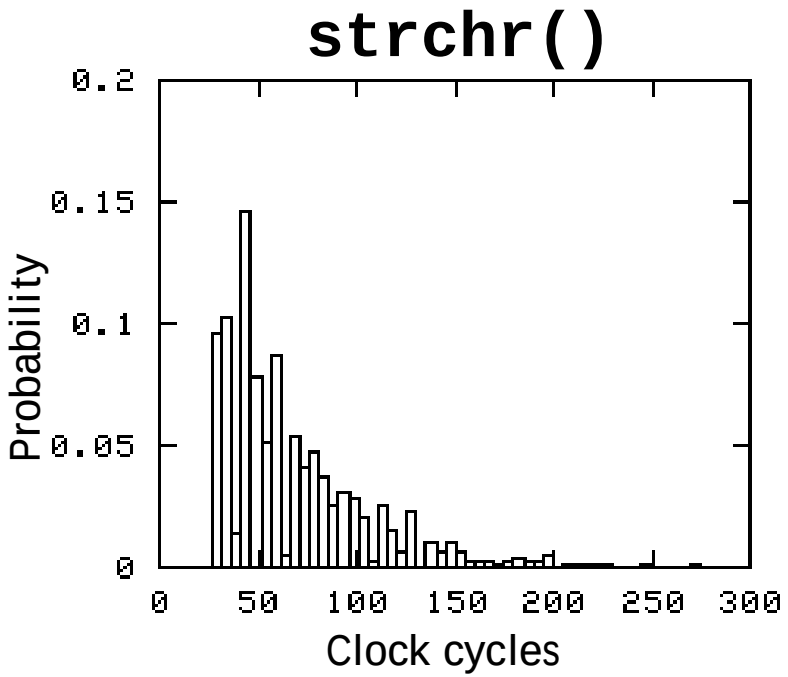
- Routines with simple behavior
  - Post-processing can extract analytical models

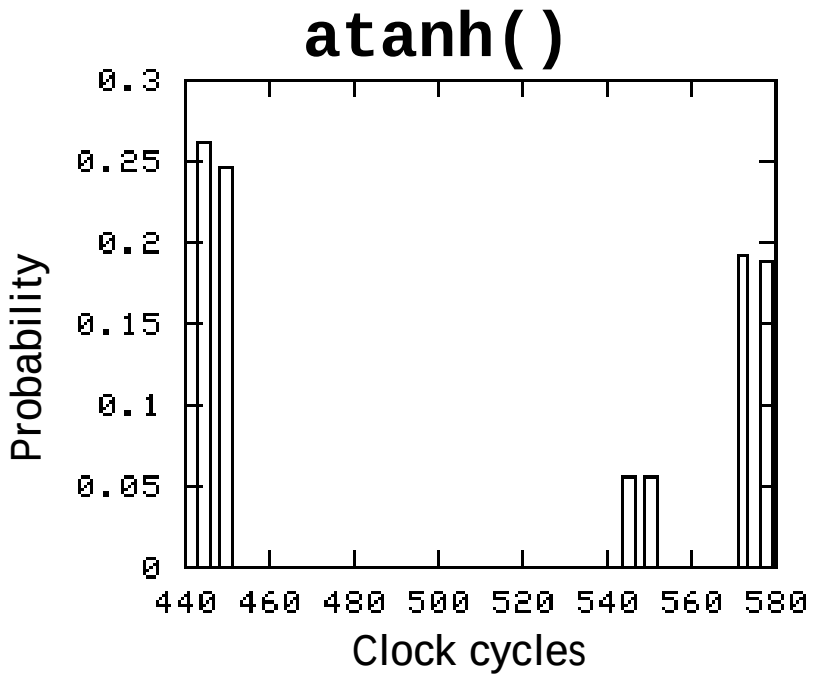


# Library analysis - Results

- Routines with complex behavior
  - Post-processing can hardly extract analytical models



# Source level - Results

- Based on the deterministic contribution only

| Program | Time (Clock cycles) | | | Energy (μJ) | | |
|---|---|---|---|---|---|---|
| | Real | Estimated | Error (%) | Real | Estimated | Error (%) |
| `loop` | 25256 | 25394 | -0.54 | 1867 | 1888 | -1.11 |
| `factorial` | 56940 | 61030 | -6.70 | 4169 | 5137 | -18.84 |
| `mcd3` | 32193 | 32094 | +0.31 | 2379 | 2373 | +0.27 |
| `arith` | 48294 | 49949 | -3.31 | 3553 | 3678 | -3.41 |
| `crc16` | 226093 | 246012 | -8.10 | 17510 | 18539 | -5.55 |
| `base64enc` | 162445 | 149211 | +8.88 | 11501 | 11098 | +3.63 |
| `real2frac` | 46735 | 48688 | -4.01 | 3407 | 3533 | -3.57 |
| `pfactor` | 84058 | 84158 | -0.12 | 6264 | 6500 | -3.63 |
| `adpcm` | 41384 | 37629 | -9.07 | 3213 | 3275 | +1.93 |
| **Average** | | | **4.55** | | | **4.65** |

# Source level - Results

- Effect of the statistical contribution
  - A linear error can be corrected assuming $\delta = const$