Microarchitectural Power Modeling Techniques for Deep Sub-Micron Microprocessors*

Nam Sung Kim†, Taeho Kgil, Valeria Bertacco, Todd Austin, Trevor Mudge
Microprocessor Research, Intel Labs, Hillsboro, OR 97124†
Advanced Computer Architecture Lab, University of Michigan, Ann Arbor, MI 48109
nam.sung.kim@intel.com†, {tkgil, vale, austin, tnm}@eecs.umich.edu

ABSTRACT

The need to perform early design studies that combine architectural simulation with power estimation has become critical as power has become a design constraint whose importance has moved to the fore. To satisfy this demand several microarchitectural power simulators have been developed around SimpleScalar, a widely used microarchitectural performance simulator. They have proven to be very useful at providing insights into power/performance trade-offs. However, they are neither parameterized nor technology scalable. In this paper, we propose more accurate parameterized power modeling techniques reflecting the actual technology parameters as well as input switching-events for memory and execution units. Compared to HSPICE, the proposed techniques show 93% and 91% accuracies for those blocks, but with a much faster simulation time. We also propose a more realistic power modeling technique for external I/O. In general, our approach includes more detailed microarchitectural and circuit modeling than has been the case in earlier simulators, without incurring a significant simulation time overhead—it can be as small as a few percent.

Categories and Subject Descriptors:

B.6.3 [Logic Design]: Design Aids—*Simulation, Optimization*; **B.7.2 [Integrated Circuits]:** Design Aids—*Simulation, Layout*

General Terms: Design, Performance

Keywords: Power modeling, Deep sub-micron

1. Introduction

Power consumption has quickly become a key design constraint in microprocessor designs, from low-end embedded processors to high-end high-performance systems [1, 2]. The embedded processors found in PDAs and cell phones must utilize energy efficient designs, as their energy payload is limited by form factor and weight constraints. With battery power density improving only at a rate of about 5% per year, increase in battery lifetime comes about through improvements in the energy efficiency of system components. To create power-sensitive designs, accurate power estimation combined with architectural or system level performance simulation is a key design tool that permits rapid early design studies that gauge trade-offs between performance and power.

Recently, several microarchitectural-level power estimation tools have been introduced [3, 4, 5] in academia, and they have been widely adopted for use in design studies that require power modeling. In all of these tools, microprocessor power is estimated by accruing power as estimated by the power models for each access to microarchitectural functional blocks. In *Wattch* [3], Brooks et al. extended CACTI, an access and cycle time model for on-chip caches [6], to model the power dissipation of on-chip storage blocks such as caches, register files, and branch target buffers. The model used in Wattch resorts to a fast approximation that is well suited for the high-end designs containing large and complex memory, but the power consumption of datapath and execution blocks is estimated by a single, per-access value, which is not scalable for the technology nor the different circuit style. Although their approach is a good approximation for high-end application domain, we believe that embedded designs require a more accurate modeling, based on the specific switching activity within each execution block.

In *SimplePower* [4], Vijaykrishnan et al. incorporated *register-transfer level* (RTL) power models based on *look-up tables* (LUT) into a microarchitectural simulator. Each LUT contains a set of pre-characterized power dissipations for a datapath component, and each entry of the LUT, indexed by the hamming distance between subsequent input vector pairs, returns the estimated power of the component [7]. The objective of this tool is to provide a framework for quickly evaluating a range of architectural and algorithmic trade-offs during the early design stages. To this end, it targets a reference processor design for the pre-computation of the capacitance tables. This reference design, while accurate enough for the purpose of the trade-off analysis, is not easily modifiable to describe specific alternative designs that may have different datapath width, smaller feature size, or different technology. On the other hand, an evaluation of power dissipation in later design stages would obviously benefit from referencing the specific design under development. These microarchitectural-level power modeling tools have been invaluable in giving computer architects the insights necessary to develop first-generation microarchitectural power optimizations. However, a rapidly changing technology landscape combined with increasingly complex microarchitectural features has brought about an erosion in the fidelity of existing power models [8].

In this paper, we propose a parameterized and technology scalable microarchitectural-level power modeling technique that suits the needs of accurate power estimations of microprocessor designs. This technique combines simplified circuit-level capacitance extraction and cycle-based logic simulation embedded into a microarchitectural level simulator such as *SimpleScalar* [9], to obtain execution time and circuit specific power dissipation data. In addition, we introduce a more detailed microarchitectural event modeling methodology to give a cycle-accurate power estimation of long-latency multi-cycle operations such as external I/O access.

The remainder of this paper is organized as follows. First, we present the basic power modeling technique in Section 2, which including the capacitance extraction and switching-event estimation methodologies. Second, we illustrate case studies for the memory, datapath, I/O, and their calibration in Section 3, Section 4, and Section 5, respectively. Finally, Section 6 summarizes our contributions.

*This work is supported in part by DARPA PACC program, Semiconductor Research Corp., and Intel Graduate Fellowship.

2. Power Modeling

2.1 Modeling transistor capacitance components

For the accurate *dynamic* power estimation of a circuit, it is important to understand the intrinsic capacitance components of a transistor, because the dynamic power dissipation is estimated based on those capacitances plus the switching activity of the transistor nodes. Figure 1 shows the intrinsic capacitance components in the transistor (or MOSFET). In Figure 1-(a), G, B, S, D nodes represent gate, body, source, and drain. C_J and C_{JSW} in Figure 1-(b) represent the junction bottom area and sidewall capacitance. L, W, D_L, and D_W represent the channel length and width, and junction length and width of the transistor. In deep sub-micron technology, most of the dynamic power dissipation is due to the charging and discharging of gate and source/drain capacitances during each transition. Therefore, we need an accurate, yet simple model to estimate these capacitance components accurately for our power estimation technique. The gate capacitance can be computed as follows [10]:

$$C_g \cong (C_{ox} \times (L - 2\Delta L) + C_{ovlp} \times 2\Delta L) \times W \qquad (1)$$

where C_{ox}, C_{ovlp}, and ΔL are gate oxide and gate overlap capacitance per unit area, and gate overlap length, respectively. Moreover, L is usually fixed to be the minimum channel length of the technology for digital circuits, thus the only unknown in the expression is the channel width W. For the computation of source and drain capacitances we use:

$$C_D \;=\; (AD) \times C_J + PD \times C_{JSW} \qquad (2)$$

where $AD = D_L \times D_W$ is the drain area and $PD = 2\times(D_L+D_W)$ is the drain perimeter. *AD* and *PD* can usually be extracted from the physical layout; alternatively it is possible to obtain a rough estimate based on the design rule set of the target technology and the design structure: D_L and D_W can be approximated as $3 \times L$ and W for small size devices. In addition, the rest of parameters such as *Cox*, *AD*, and *PD* for a specific technology can be obtained from the SPICE technology parameters [11].

Figure 1.The intrinsic capacitance components in a transistor — lateral (a) and top (b) views.

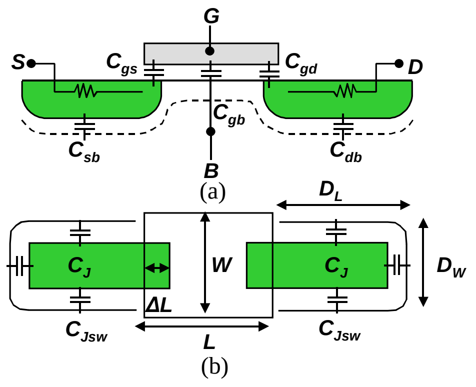


Figure 3.An example of modeling a set of gates.

```
/* create logic gate instances connect the nets */
Node A, B, C;
nor = Nor(&A, &B, WP1, WN1, WP2, WN2);
nand = Nand(nor.Y, &C, WP1, WN1, WP2, WN2));

/* levelized simulation */
energy += nor.GateOp(supply_voltage);
energy += nand.GateOp(supply_voltage);
```

2.2 Modeling switching events with an embedded cycle-based logic simulator

Once node capacitances are estimated, the next step is to gather node-switching information. We compute each switching *on the fly* during microarchitectural simulation, because total dynamic power dissipation is heavily dependent on the number of switching at the internal nodes [12][13]. The two-input CMOS NAND gate — the most basic logic component along with the inverter — consists of four transistors. Figure 2 illustrates how we model a two-input NAND logic gate using our proposed methodology using the corresponding classes and methods: for each node of the netlist, an instance of class *Node* stores *logicValue* and *capacitance*, which is estimated using Equation 1 and/or 2, based on the node's connections. Class *LGate_2* is a base class for any type of 2-input gate; it includes a constructor that takes into account the widths of the component transistors, and a virtual method for a generic 2-input operation evaluation. The derived class *Nand* provides a specific definition for the *GateOp* function that computes the new output value and the power dissipated if a transition has occurred. It is easy to observe how to define other 2-input gates as derived classes of *LGate_2* and how to create other types of generic gates. The last portion of the pseudo-code shows how to create one nand gate, set initial input values, and then perform the simulation.

At the netlist level, multiple gates are created and connected together to simulate the entire logic block. We levelize each gate or netlist primitive and simulate each gate one after the other, in a sequence compatible with the partial ordering imposed by the levelization. This approach corresponds to the levelized cycle-based simulation technique in logic simulation [14]. As a small example, Figure 3 illustrates how to create a netlist for a combinational circuit and how to simulate the internal node activity. The example shows a combinational circuit consisting of 2-input NOR and 2-input NAND gates. We are able to evaluate the correct output logic value by evaluating the gates in the order of increasing distance from the primary inputs. For the regular logic structures such as datapath, decoder, and memory, the SPICE netlists can be easily

Figure 2.Power modeling of a 2-input Nand gate

```
/* generic circuit node class */
class Node { public:
  Bit logicValue; /* node logic value */
  double capacitance;/* node capacitance */};

/* generic two input logic gate class */
class LGate_2 {
  Node Y; /* output node */                    transistor widths
  Node A, B; /* input nodes */
  LGate_2(I1, I2, WP1, WN1, WP2, WN2); /* constructor */
  virtual double GateOp (double voltage);/* gate eval fn*};

/* Nand derived class */
class Nand: public LGate_2 {
double GateOp (double voltage) {
```

```
  /* derived Nand function */
  Bit newY = !(A & B);
  if (Y.logicValue ==0 && newY==1)
  Y.logicValue = newY;
  return 0.5*Y.capacitance*voltage*voltage;}

  /* node capacitance extraction */
  Y.capacitance += drain_cap(WP1+WP2+WN1+WN2);
  A.capacitance += gate_cap(WP1+WN1);
  B.capacitance += gate_cap(WP2+WN2);}

/* create the netlist */
nand = Nand(&A, &B, WP1, WN1, WP2, WN2);
/*assign inputs and eval logic and power */
A.logicValue = 0x1; B.logicValue = 0x0;
energy = nand.GateOp(voltage);
```

translated to the embedded cycle-based logic simulator routines with the switching capacitance extraction from the transistor sizes.

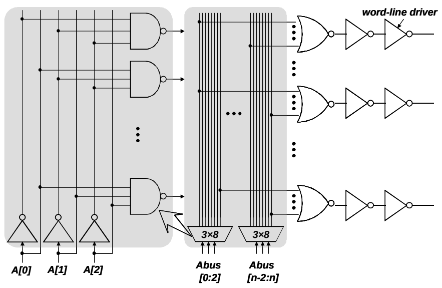
The levelized approach we use here usually provides better performance over an event-driven simulation, since we trade having to maintain an event queue at the expense of simulating every gate in the netlist at each time step [15]. A downside of the levelized approach is that we lose information on arrival times of signals, thus we cannot evaluate power dissipation due to glitches and temporary transitions. However, a well-designed combinational circuit should not generate many glitches, in which case our model is still accurate.

3. Memory Power Models

In modern microprocessors, *static random access memory* (SRAM) is extensively used for caches, TLBs, BTBs, branch predictors, register files, instruction queues, etc. For instance, 40% of the total power in the Alpha 21264® and 60% of the total power of the StrongARM® processor is devoted to cache and memory structures [16][17]. As the feature sizes have shrunk and supply voltages have decreased, bit-line voltage swings during read operations have been decreased to 100mV and combined with the word-line pulse technique [18]. This has dramatically reduced the power consumption by bit-lines. In contrast, the original CACTI model assumes that the bit-line is fully discharged. The decoder model is also affected by scaling to a lesser extent. We have revised the power model of both the decoder and the bit-lines and their sense-amps to accurately reflect in modern SRAM designs.

While the bit-line power dissipation is independent from the switching activity of the data due to the complementary structure of bit-lines, the power dissipation of the decoder is heavily dependent on the switching events of the decoder address inputs. Hence, we need to build a switching event-sensitive power model for the decoder. We present now an example of how to use the technique just presented in Section 2 to model a 7×128 decoder power consumption designed with the TSMC 0.18μm technology Artisan standard cell library and the Synopsys® design compiler®. Figure 4 shows the 7×128 decoder logic and its corresponding description for the power modeling. The decoder logic has a regular structure consisting of a set of NANDs, NORs, and INVs. The cycle-based logic simulator for the decoder was derived by instantiating and connecting those gates in an iterative way, see the loops in Figure 4. The switching capacitance of each node was automatically extracted and inserted into the logic simulator using our own software annotation procedures. Then, the resulting logic simulator annotated with the extracted capacitance is embedded in the microarchitectural simulator with an interface routine. This passes

Figure 4.An example of modeling an 7×128 decoder.



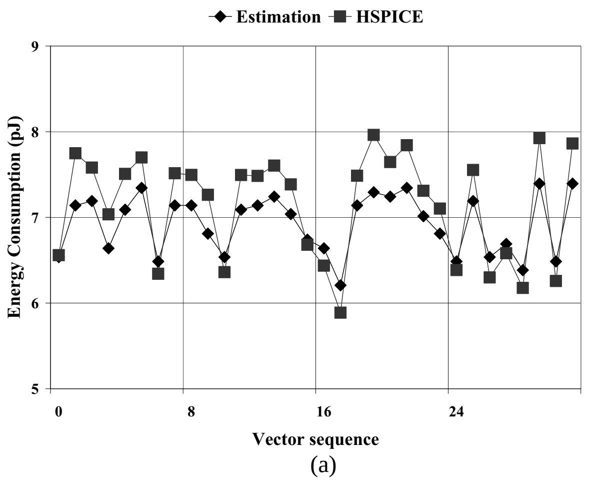
```
/* dec 3×8 class */
class Dec3×8: {
  Node A[3], Y[8]; Nand intNand3[8], Inv intInv[3];
Dec3×8(...); /*setup internal node connections */
double GateOp (double voltage) {
  /* compute 3×8 decode logic values and energy*/
  for(i = 0; i < 3; i++)energy += intInv[i].GateOp(volt)
  for(i = 0; i < 8; i++)energy += intNand[i].GateOp(volt
  return energy;}
};

/* create dec 7×128 netlist */
for(i = 1; i < 3; i++) {}
  /* create and connect Dec3x8 (pre-decoder) instances *
  Dec3×8[i] = Dec3×8(A[i*3], Dec3×8Y[i*8],...);}
for(i = 1; i < 8; i++) {}
  for(j = 1; j < 8; j++) {}
  /* create and connect Nor gate instances */
  intNor3[i*8+j] = Nor(Dec3×8Y[j], Dec3×8Y[i], ...);}
  ...
}
```

the current address bus value to the logic simulator and returns the estimated energy consumption to the microarchitectural simulator. For the bit-line energy consumption, we used the following equation:

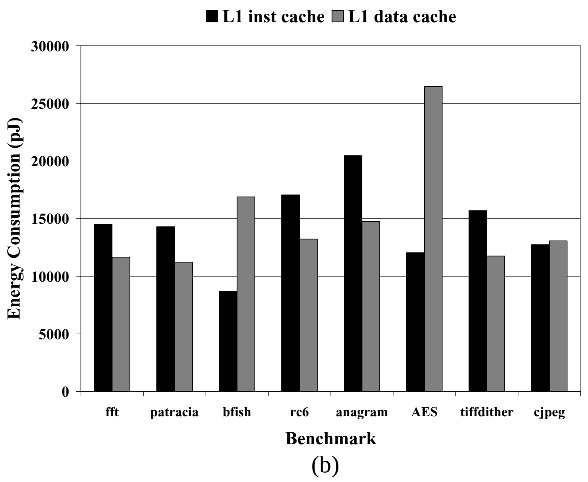
$$E = C_{bit\text{-}line} \times V_{DD} \times \Delta V_{swing}, \tag{3}$$

where $C_{bit\text{-}line}$ and ΔV_{swing} are bit-line capacitance per memory column and bit-line voltage swing. The bit-line capacitance per column includes the bit-line interconnect, the access transistor drain, and the pre-charge circuit drain capacitance. The bit-line interconnect capacitance was estimated based on the actual SRAM dimension and using available MOSIS parametric test results for the TSMC 0.18μm technology fabrication run [11]. The access transistor drain capacitance connected to the bit-line was estimated using Equation 2.

Figure 5.SRAM energy consumption model calibration in (a) and L1 instruction and data cache energy consumption in (b).



(a)



(b)

Figure 5-(a) shows the calibrated energy consumption of 4KB SRAM power model against HSPICE measurement. In the figure, each point represents the energy consumption for each applied vector. For the HSPICE experiment, we modeled and simulated the whole 7×128 decoder and a dummy 128×256 bit memory array: we modeled just one column of 128 cells multiplied by 256 to speed up the simulation. As seen in Figure 5-(a), the estimated energy consumption tracks with the actual measurement result closely for each applied vector. On average, the proposed technique shows around 7% estimation error for 1K vectors compared to the HSPICE measurement. However, comparing the simulation speed, the proposed technique completed the estimation within a few seconds while the HSPICE took 3.4 hours on UltraSparc80® 450MHz dual processors with a 4MB L2 cache.

Figure 5-(b) shows the total accumulated energy consumption of the 4KB L1 instruction and data caches obtained by running 10 million instructions for a subset of embedded benchmark programs from MiBench [19]. The proposed power models were embedded in SimpleScalar/ARM performance simulator suite with the StrongARM configuration for this experiment. During the energy estimation the address stream extracted from the microarchitectural simulator was applied to the SRAM power model on the fly.

The estimated energy consumption results show that total energy consumption can be significantly different depending on the benchmark programs although the same numbers of instructions are executed. Usually, the instruction caches consume more energy than the data caches although the *average energy dissipation per access* of data caches are usually higher than that of instruction caches. The primary reason for this energy consumption characteristic is that the instruction caches are more frequently accessed than the data caches while the address stream from the data caches are more non-sequential, which means more switching events in the address bus, than that from the instruction caches. These characteristics imply that both the application-specific functional block input switching and access activities must be modeled for accurate power estimation of the embedded microprocessor. In terms of microarchitectural simulator execution time overhead, the proposed technique increases it by 3% for both the instruction and data caches.

4. Datapath and Execution Unit Power Models

We present next an example of how to use the proposed technique to implement a datapath component and generate a power estimator that interfaces at run-time with the micro-architectural simulator to gather the proper input stimulus. For this example, we consider a 32-bit carry-select adder consisting of eight 8-bit ripple-carry adders as modeled in Figure 6.; for each 8-bit addition, two 8-

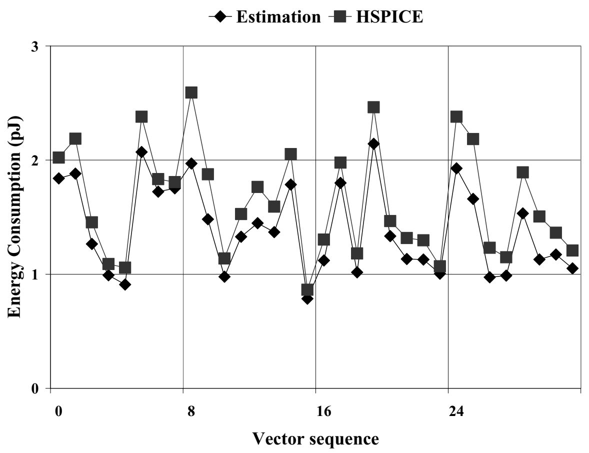
Figure 6.An example of modeling an 8-bit RCA.

```
class FullAdder: {
  Node A, B, S, CI, CO;
  Nand intNand1, intNand2...; Or intOr1, intOr2...;
FullAdder(...); /*setup internal node connections */
double GateOp (double voltage) {
  /* compute sum and carry logic values and energy*/
  double energy = intNand1.GateOp(voltage);
  energy += ...
  return energy;}
};

/* step 1: create netlist */
for(i = 1; i < WIDTH; i++) {
  /* create and connect FullAdder instances */
  FA[i] = FullAdder(A[i], B[i], FA[i-1].CI, CO[i],
SO[i]...);}

/* step 2: load input vectors */
A.apply(LOp); B.Apply(ROp);

/* step 3: logic and energy evaluation*/
for(i = 0; i < WIDTH; i++) {
  energy += FA[i].GateOp(voltage);}
```

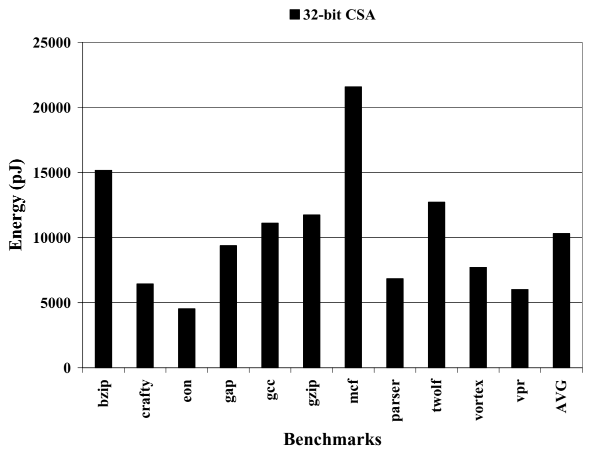
bit ripple-carry adders are used to compute the results in parallel for zero and one carry-ins, respectively. The first step is to construct the basic block for a full adder by instantiating the necessary logic gates: the definition of the class FullAdder creates all the internal gates and it properly connects them in its constructor so that two output nodes, *S* and *CO*, produce the correct functionality. At this point, the main program can create and connect the full adder blocks employing a loop shown in Figure 6. Note how the program structure lends itself naturally to the parameterization of the bus width. By instantiating eight 8-bit ripple-carry adders, we are able to build a 32-bit carry adder. At this point, the power estimator includes a complete description of the logic block under study. The last two steps provide an interface to the microarchitectural simulator by retrieving *on-the-fly* at each cycle the input vectors corresponding to the two operands of the addition operation, and proceeding with the power/logic simulation.

We calibrated our embedded power estimator by comparing the results with the corresponding HSPICE circuit simulation. Figure 7-(a) shows a calibration using the carry-select adder of the previous case study. Each point in the graph represents dissipated energy estimated or measured by applying each vector to the circuit. The diagram indicates that the technique proposed tracks the actual power dissipation of adders very well; we found that the average estimation error by applying 1K vectors is around 9%. The

Figure 7.32-bit CSA energy consumption model calibration in (a) and total energy consumption in (b).



(a)



(b)

steady under-approximation error of the power estimator can be explained by two sources of power dissipation that our model does not take into account: glitches occurring because of the relative delays among signal propagation times, and temporary short circuits due to both PMOS and NMOS transistors being turned on during the transition.

To produce the graph in Figure 7-(b), we simulated the SPEC2000 INT benchmark programs [20] while running the power simulator on our 32-bit adder component. For each benchmark program, we applied 32K vectors to the power model. The results show the total energy that was dissipated in the adder. Note how the total energy dissipation profiles present high variations over different benchmark programs: for instance *mcf* consumes 480% more energy than *eon*, which seems to indicate that the amount of data activity plays an important role in the accurate estimation of the power dissipation of a datapath component. Because of its accuracy and flexibility, this technique could easily be applied in trade-off studies of various solutions for datapath circuits, or for optimization of power dissipation for the embedded processors where the datapath constitutes a significant portion of the total power dissipation.

5. I/O Power Models

Generally, the I/O circuits (dis)charge a large amount of loading capacitance as well as require higher supply voltage than the microprocessor core (e.g., 3.3V). This makes the I/O circuits a major contributor to the peak power dissipation of the microprocessor. Although the microprocessor may not frequently access the external memory through the I/O in the presence of L1 and L2 on-chip caches, a significant amount of power will be still consumed by the I/O circuits; the Alpha 21264 I/O circuits consumes 5% (~3.5W) of total power in average [16]. Furthermore, the fraction of power dissipated by the I/O circuits will be significantly increased in those embedded processors with not on-chip L2 or L1 cache. However, the power consumed by I/O circuits has been ignored or not modeled properly in most microarchitectural power estimation frameworks. There are two sources of error: 1) the lack of detailed information about the external loading capacitance connected to the I/O circuit, and 2) the I/O bus transaction model used in microarchitectural simulator.

Figure 8 shows both the memory I/O access modeling in the microarchitectural simulator and the cycle-accurate I/O bus transaction modeling. For example SimpleScalar — baseline simulator for most microarchitectural power estimation frameworks — transfers all the request data blocks at the call time of external memory access function (e.g., `mem_access` in Figure 8) and returns just an access latency. The typical microprocessor transfers the blocks one by one over several I/O bus cycles with a more complex data transfer protocol. As we have noted, the cycle-based microarchitectural simulators derive their speed from abstracting out many of the physical details. Hence, we have no idea about the details of the memory transfer protocol including exact timing and bus switching activity of address and data I/O buses. To correct this, we need a mechanism or modification for tracing actual I/O address and data during the I/O transactions in a cycle accurate way. To provide this mechanism, it is necessary to augment the simulator to trace I/O bus streams and feed them to the power model at the pertinent I/O transaction cycle.

Figure 8.The memory access I/O modeling in the micro-architectural simulator.

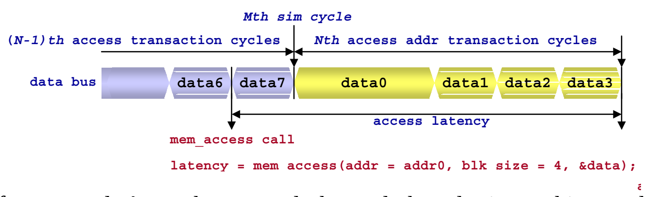
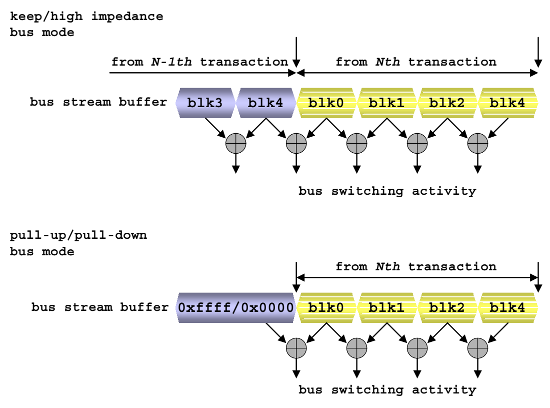

Mth sim cycle
(N-1)th access transaction cycles
Nth access addr transaction cycles
data bus
data6
data7
data0
data1
data2
data3
access latency
mem_access call
latency = mem access(addr = addr0, blk size = 4, &data);

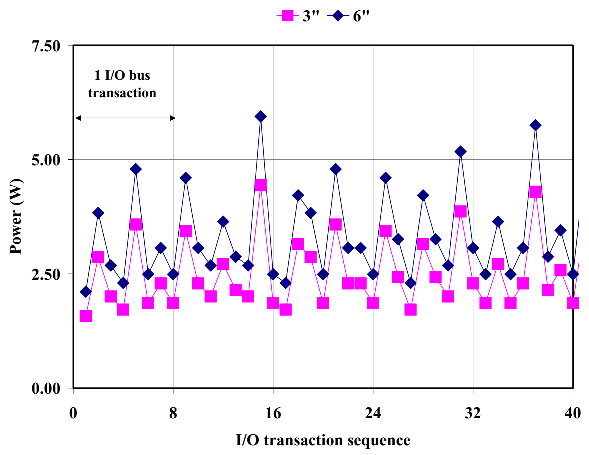
Figure 9-(a) shows an I/O bus power model accounting for the actual I/O bus switching activity during memory I/O bus cycles. In this model the number of “0” to “1” transitions of the I/O pin is counted by comparing the blocks transferred in the previous and current I/O bus cycles. At the initiation of the I/O bus transaction cycle, the high-impedance bus state is assumed. To estimate the power dissipation by the I/O bus at a particular I/O cycle, the count of the number of I/O pin transitions of each block is transferred to the I/O circuit power model. In general, the switching capacitance of the I/O circuit consists of the intrinsic (or internal) capacitance of the I/O circuit itself and the extrinsic (or external) capacitance of the connected chipset and the PCB interconnect between the microprocessor and chipset I/O pins. The amount of the extrinsic capacitance driven by the I/O circuit is more significant than the intrinsic capacitance. Therefore, it is important to estimate the extrinsic capacitance in a realistic way.

In most computer systems, the microprocessor is not directly connected to the memory module in the PC mother board, but it is connected to the memory controllers through a *front-side system bus* (or simple I/O bus), e.g., the Intel Pentium processors [21]. Hence, the I/O pin capacitance of the microprocessor and chipset should be known as well as the PCB interconnect capacitance of the front system bus. For illustration purposes the necessary interconnect dimension and layer were obtained from the Intel® 875P chipset for the Pentium 4® processor [21]. The PCB layout parameters and the interconnect capacitance was estimated using [22] for

Figure 9.The I/O bus switching activity modeling in (a) and a snapshot of power dissipation by 64-bit processor I/O bus in (b).



(a)



(b)

the external I/O bus capacitance as well as the chip package pin capacitances [23].

According to [23] the typical package pin capacitance of both the microprocessor and chipset is 5pF per I/O pin. The interconnect dimensions and layer information is usually found in the chipset or microprocessor specification [21]; in the case that neither the microprocessor nor the chip-set have been developed, the most recent available information can be used. The estimated PCB interconnect capacitance per inch is around 2.15pF for the given specification, and the minimum and maximum allowed front-side bus interconnect lengths are 3” and 6”, respectively. Therefore, the PCB interconnect capacitance of the front system bus is between 6.5pF and 13pF depending on the interconnect length; in case of the 6” front side system bus, the PCB interconnect capacitance is around 13pF, which results in total 23pF per pin including the package pin capacitance of both the microprocessor and chipset.

With the I/O bus capacitance and the detailed bus protocol modeling we were able to estimate the power dissipation of the 64-bit microprocessor I/O bus with realistic parameters (see Figure 9-(b) for a snapshot of I/O bus power dissipation when running *eon*). The experiment shows that the power dissipation by the I/O bus is substantial whether the front system bus interconnect length is 3” or 6”, and it has a great potential to contribute to the peak as well as the average power dissipation of the microprocessor during the I/O bus cycles. Furthermore, this experiment shows that counting switching activity in a cycle accurate way is important, because the power dissipation by I/O at a specific I/O cycle differs significantly depending on the number of I/O pin switching.

6. Conclusion

In this study, we provided power modeling methodologies for for deep sub-micron microprocessors. The following summarize our contributions in this study. First, we introduced a simple switching capacitance extraction methodology and a cycle-based logic simulation technique which can be easily embedded into a high-level microarchitectural simulator — we used SimpleScalar. The high-level microarchitectural simulator enables the user to explore a much larger design space quickly. Combining this high-level simulator with the embedded low-level logic simulator gives us more accurate power estimation results quickly for specific target functional blocks. Second, we illustrate and *calibrate* our power modeling for caches, execution units, and I/O. Our experiments show the power models track HSPICE closely for each applied vector as well as producing accurate average energy dissipation. This is achieved with a very small execution time overhead—it can be as small as a few percent.

References

[1] T. Mudge. Power: A first class design constraint. Computer, vol. 34, no. 4, April 2001, pp. 52-57.

[2] Nam Sung Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, N. Vijaykrishnan. Leakage Current: Moore's Law Meets Static Power. Computer, vol. 36, no. 12, Dec. 2003, pp. 65-77.

[3] D. Brooks et al., “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” *Proc. 27th Int. Symp. on Computer Architecture (ISCA27)*, May 2000.

[4] N. Vijaykrishnan, et al., “Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower,” *Proc. 27th Int. Symp. on Computer Architecture*, May 2000.

[5] G. Cai et al., “Architectural Level Power/Performance Optimization and Dynamic Power Estimation,” *Cool Chips Tutorial in conjunction with the 32nd Int. Symp. on Microarchitecture*, Nov 1999.

[6] S. Wilton et al., “An Enhanced Access and Cycle Time Model for On-Chip Caches,” *Western Research Laboratory Research Report 93/5*, July 1993.

[7] H. Mehta et al., “Energy Characterization based on Clustering,” *Proc. 33rd Design Automation Conf.*, June 1996.

[8] Nam Sung Kim, T. Austin, T. Mudge, D. Grunwald, “Challenges for Architectural Level Power Modeling in Power Aware Computing,” Kluwer Academic Publishers, Boston, MA, 2001.

[9] T. Austin et al., “SimpleScalar: An Infrastructure for Computer System Modeling,” *IEEE Computer*, Vol. 35, pp. 59-67, Feb. 2002.

[10] B. Geuskens, et al., “Modeling Microprocessor Performance,” Kluwer Academic Publishers, 1988.

[11] The MOSIS Service. http://www.mosis.com.

[12] P. E. Landman et al., “Activity-Sensitive Architectural Power Analysis,” *IEEE Transaction on CAD of Integrated Circuit and Systems*, Vol. 15, No. 6, June 1996

[13] P. E. Landman et al., “Architectural Power Analysis: The Dual Bit Type Method,” *IEEE Transaction on VLSI Systems*, Vol. 3, No. 2, June 1995.

[14] Z. Brazilai et al., “HSS: A High-Speed Simulator,” IEEE Trans. on CAD/ICAS, July 1987.

[15] L. T. Wang et al., “SSIM: A Software Levelized Compiled-Code Simulator,” *Proc. 24th Design Automation Conf.*, June 1987.

[16] M. K. Gowan et al., “Power Considerations in the Design of the Alpha 21264 Microprocessor,” *Proc. of 35th Design Automation Conf.*, June 1998.

[17] J. Montanaro, et al., “A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor,” *IEEE Journal of Solid-State Circuits*, Vol 31, Nov 1996.

[18] K. Roy and S. Prasad, “Low-Power CMOS VLSI Circuit Design,” *Wiley Interscience publication*, 2000.

[19] M. R. Guthaus et al., “MiBench: A Free, Commercially Representative Embedded Benchmark Suite,” *Proc. IEEE 4th Annual Workshop on Workload Characterization*, Dec. 2001.

[20] Standard Performance Evaluation Corporation. http://www.specbench.org.

[21] Intel 875 Chipset Datasheet — Platform Design Guide, ftp://download.intel.com/design/chipsets/datashts/25252703.pdf.

[22] Microstrip Impedance Calculator, http://www.emclab.umr.edu/pcbtlc2/microstrip.html

[23] Intel 875 Chipset Datasheet, ftp://download.intel.com/design/chipsets/datashts/25252501.pdf.

[24] A. Bellaouar et al., “Low-Power Digital VLSI Design: Circuit and Systems,” *Kluwer Academic Publishers*, 1996.

[25] K. Ghose and M. Kamble, “Reducing Power in Superscalar Processor Caches using Subbanking, Multiple Line Buffers and Bit-line Segmentation,” *Proc. Int. Symp. on Lower Power Electronics & Design*, Aug. 1999.

[26] R. Preston et al, “Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading”, *ISSCC Digest and Visuals Supplements*, Feb. 2002.

[27] M. Hrishikesh, N. Jouppi, K. Farkas, D. Burger, S. Keckler, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. *Proc. the 29th Int’l Symp. on Computer Architecture*, May 2002.

[28] S. Manne et al., “An Industrial Perspective on Low Power Processor Design,” *Cool Chips Tutorial in conjunction with the 32nd Int. Symp. on Microarchitecture*, Nov 1999.